Page 1

UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE PATENT TRIAL AND APPEAL BOARD

LG ELECTRONICS, INC.,	)	
Petitioner,	)	
vs.	)	Nos. IPR2015-00326 IPR2015-00330
ATI TECHNOLOGIES ULC,	)	
Patent Owner.	)	

VIDEOTAPED DEPOSITION OF NADER BAGHERZADEH, Ph.D.
Los Angeles, California
Tuesday, September 15, 2015
Volume I

Veritext Legal Solutions
Mid-Atlantic Region
1250 Eye Street NW - Suite 1201
Washington, D.C. 20005

ATI 2073
LG v. ATI
IPR2015-00326

Page 2

UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE PATENT TRIAL AND APPEAL BOARD

____
)
LG ELECTRONICS, INC.,)
)
Petitioner,)
)
vs.) Nos. IPR2015-00326
) IPR2015-00330
ATI TECHNOLOGIES ULC,)
)
Patent Owner.)
___________________________)

Videotaped deposition of NADER BAGHERZADEH, Ph.D., Volume I, taken on behalf of Patent Owner, at 350 South Grand Avenue, Suite 2500, Los Angeles, California, beginning at 9:38 a.m. and ending at 1:42 p.m. on Tuesday, September 15, 2015, before NADIA NEWHART, Certified Shorthand Reporter No. 8714.

Page 3

APPEARANCES:

For Petitioner:

MAYER BROWN, LLP
BY: ROBERT G. PLUTA
Attorney at Law
71 South Wacker Drive
Chicago, Illinois 60606-4637
312-701-8641
rpluta@mayerbrown.com

MAYER BROWN, LLP
BY: JOHN X. ZHU
Attorney at Law
1999 K Street, N.W.
Washington, D.C. 20006-1101
202-263-3318
jzhu@mayerbrown.com

Page 4

APPEARANCES (Continued):

For Patent Owner:

STERNE KESSLER GOLDSTEIN FOX
BY: JONATHAN TUMINARO, Ph.D.
BY: TYLER J. DUTTON
Attorneys at Law
1100 New York Avenue, NW
Washington, D.C. 20005
202-371-2600
jtuminar@skgf.com
tdutton@skgf.com

Videographer:
GRANT CIHLAR

Page 5

INDEX

WITNESS	EXAMINATION
NADER BAGHERZADEH, Ph.D. Volume I	
BY MR. TUMINARO	8
BY MR. PLUTA	123

EXHIBITS

NUMBER	DESCRIPTION	PAGE
Exhibit 1	Declaration of Dr. Nader Bagherzadeh; 111 pages	10
Exhibit 2	Declaration of Dr. Nader Bagherzadeh; 86 pages	11
Exhibit 3	Consultant Curriculum Vitae for Nader Bagherzadeh, Ph.D.; 33 pages	15
Exhibit 4	Article entitled "How GPUs Work"; 5 pages	36
Exhibit 5	Article entitled "Exploiting the Shader Model 4.0 Architecture"; 9 pages	40

Page 6

INDEX (Continued):

EXHIBITS

NUMBER	DESCRIPTION	PAGE
Exhibit 6	United States Patent Number 5,808,690; 50 pages	61
Exhibit 7	United States Patent Number 6,897,871; 10 pages	61
Exhibit 8	United States Patent Number 7,015,913; 20 pages	84
Exhibit 9	United States Patent Number 7,376,811; 12 pages	98
Exhibit 10	Patent Trial and Appeal Board Decision; 28 pages	104

Page 7

Los Angeles, California, Tuesday, September 15, 2015
9:38 a.m.

THE VIDEOGRAPHER: Good morning. We're on the record. The time is 9:38 a.m. on September 15th, 2015. This is the video-recorded deposition of Dr. Nader Bagherzadeh.

My name is Grant Cihlar, here with our court reporter, Nadia Newhart. We are here from Veritext Legal Solutions at the request of counsel for the patent owner. This deposition is being held at Mayer Brown in Los Angeles, California.

The caption of this case is LG Electronics, Incorporated versus ATI Technologies ULC. The case numbers are IPR2015-00330 and IPR2015-00326.

Please note that audio and video recording will take place unless all parties agree to go off the record. Microphones are sensitive and may pick up whispers, private conversations and cellular interference. I am a notary public. I am not related to any party in this action, nor am I financially interested in the outcome in any way. If there are any objections to proceeding, please state them at the time of your appearance beginning with the noticing attorney.

Page 8

MR. TUMINARO: Jonathan Tuminaro from the law firm of Sterne Kessler Goldstein & Fox on behalf of the patent owner, ATI Technologies ULC. And with me is Tyler Dutton, also from Sterne Kessler.

MR. PLUTA: Robert Pluta from Mayer Brown on behalf of LG Electronics.

MR. ZHU: John Zhu also of Mayer Brown for LG Electronics.

THE VIDEOGRAPHER: Thank you.

The witness will be sworn in, and counsel may begin the examination.

NADER BAGHERZADEH, Ph.D.,
having been first duly sworn, was examined and
testified as follows:

EXAMINATION

BY MR. TUMINARO:

Q Good morning, sir.

A Good morning.

Q This is the second time I'm taking your deposition, right?

A Correct.

Q Correct. Okay. Just a couple ground rules for this deposition as we did last time. We're

Page 9

trying to get a clear record from the court reporter, so I'll ask that you don't speak over me and I'll try not to speak over you; is that fair?

A That's good.

Q Okay. I'm going to try to ask clear questions, but if at any time you don't understand my question, will you let me know?

A Sure.

Q Okay. If you answer one of my questions, I'm going to assume you understood it; is that fair?

A That's fair.

Q Okay. I'm going to take periodic breaks, but if at any time you need a break, will you let me know?

A Yes.

Q Okay. One thing I'd ask, though, if there's a pending question, I ask that you answer the que- -- answer the question before we take a break; is that fair?

A That is fair.

Q Okay. You understand that you are testifying under oath here today?

A Correct.

Q Is there any reason you cannot do that?

A No.

Page 10

Q Okay. You submitted two declarations in these two cases?

A Two cases, two declarations, yes.

MR. TUMINARO: Okay. I'd like to have this marked.

(Exhibit 1 was marked for identification by the court reporter and is attached hereto.)

BY MR. TUMINARO:

Q Sir, you've been handed what's been marked as Exhibit 1 for identification purposes. This is the declaration that you submitted in the 326 IPR; is that right?

A Correct.

Q And if you would turn with me to the last page before Appendix A.

A Exhibit 1?

Q Exhibit 1.

A Last page?

Q Before Appendix A.

A Yes.

Q Is that your signature that appears on that last page that's not numbered?

A Correct.

Q Okay. And you signed this declaration on December 10th, 2014?

Page 11

A Correct.

MR. TUMINARO: Okay.

(Exhibit 2 was marked for identification by the court reporter and is attached hereto.)

BY MR. TUMINARO:

Q If you turn with me to Exhibit 2, this is the declaration that you submitted in 330 IPR; is that right?

A Correct.

Q Okay. And again, if you would just turn with me to the last page before the appendix. Is that your signature?

A Yes.

Q Okay. And you signed the declaration in the 330 IPR on December 9th, 2014; is that right?

A That is correct.

Q Okay. All right. With respect to Exhibit 1, is there anything that you would like to add?

MR. PLUTA: Object to the form.

THE WITNESS: Not that I can think of right now.

BY MR. TUMINARO:

Q Is there anything that you would like to delete?

A Not that I can think of right now.

Page 12

Q Are there any changes at all that you would like to make to your declaration in the 326 IPR?

A Nothing comes to mind right now.

Q Okay. Same with respect to Exhibit 2, which is your declaration in the 330 IPR. Anything that you would like to add to that declaration?

A No.

Q Anything that you'd like to delete?

A No.

Q Any changes at all that you'd like to make to that declaration?

A Not at this time.

Q Okay. When were you first contacted with respect to these two IPRs?

A So this was signed December. I would say -- I'm guessing early fall.

Q 2014?

A Yeah.

Q Okay. And who contacted you?

A I think it was Mr. Maas and Mr. Zhu and others from that team, I think.

Q Okay.

A I'm -- I'm not certain about this, but these are the names that come to mind.

Q That's your best recollection?

Page 13

A That's right.

Q Okay. And how were you contacted? By telephone? By e-mail? In what form were you contacted?

A First time?

Q Yes.

A Phone call, I would say, yeah.

Q Okay. And since the time that you were first contacted until you submitted your declaration, how much time did you spend preparing your declaration?

A Oh, I'm going to give you a lower bound like I have done in the past for my depositions. It was at least 40 hours.

Q At least 40 hours on each or combined? Each declaration --

A On each.

Q On each.

So a total of at least 80 hours working on your declaration in the 326 --

A And the --

Q -- and the 330?

A I'm sorry. Because there was some other case, as well, so that's -- there were -- sorry. There were other patents involved, so I would say -- I would say 40 hours each of these two, no. I would

Page 14

say total, about 40 hours or more.

Q Okay.

A Yeah, I think that's correct.

Q Okay. And what did you do to prepare for today's deposition?

A I reviewed my declaration. I looked at all the appendix references. I can -- I looked at the patent. I glanced over the patent history. I've looked at Lindholm's patent, Rich Kizhepat, Kurihara. I looked at certain portions of OpenGL that I was interested in, and that's it.

Q Okay. And did you also -- so with respect -- strike that.

You looked at the 871 patent; is that right?

A Yes.

Q Did you also look at the 369 patent?

A Yes, I did.

Q Okay. Did you also look at the prosecution history of the 369 patent?

A Glanced over it, yes.

Q Okay. And did you look at -- in preparing for your deposition here today, did you review any other exhibits that are not listed in your materials considered for your declarations?

A Not that I recall.

Page 15

Q Okay. Did you look at the patent owner's response in the 053 case?

MR. PLUTA: Object to form.

THE WITNESS: 053 case?

BY MR. TUMINARO:

Q Yes.

A In preparation, I only looked at what's related to these two patents.

Q Okay. Have you seen the patent owner's response in the 053 case?

MR. PLUTA: Object to form, object to relevance.

THE WITNESS: I don't understand your question, Counsel. You mean since my deposition, there was a response; is that what you're saying?

BY MR. TUMINARO:

Q Yeah. Since your deposition, there was a response in the 053 case. Have you seen that response?

A No.

MR. TUMINARO: Okay.

(Exhibit 3 was marked for identification by the court reporter and is attached hereto.)

BY MR. TUMINARO:

Q Sir, you've been handed what's been marked as

Page 16

Exhibit 3 for identification purposes. This is the consultant CV that was attached to -- as an exhibit to each of your declarations; is that right?

A That's correct.

Q All right. Again, anything that you would like to add to this CV?

A No. I mean, I've published additional papers, but I -- I think this is fine.

Q You've published additional papers since you submitted this declaration in -- on December 9th or December 10th of 2014?

A Yes. We publish all the time.

Q Okay. Apart from any other papers, any- -- anything that you'd want to add to this --

A No.

Q -- CV?

A No.

Q Anything that you'd want to delete from the CV?

A Oh, no.

Q Any changes at all that you would like to make?

A I mean, if there's a typo somewhere here, I would like to change it, but I have not seen a typo. If there is any misspelling or, you know, some

Page 17

cosmetic things, you know, I might want to change that. But I don't see anything that I can think of to be changed here.

Q Okay. Do you have any other CVs that you use in your professional capacity?

A Yes.

Q Apart from this consultant CV?

A That's right.

Q How many other CVs do you use?

A Different venues; different CVs. I have a CV for National Science Foundation grants. Those are two-page CVs. I have a CV for merit and promotions at the University of California. That's over 70 pages.

Q Do you have any other CVs besides those two that you mentioned?

A There are other ones for different applications. You know, the -- the way they ask you, a one-page CV or like an executive summary or something like that, yeah.

Q Do you -- do you talk on a regular basis, give -- give presentations on a regular basis?

A Yes, I do.

Q Do you have a CV that you use to indicate your experience with talking engagements?

Page 18

A No, because they know me so they don't ask for a CV.

Q Okay.

A That's right. It would be a little weird for them to ask for a CV for somebody they're inviting for a talk.

Q Okay.

A That's usually not done.

Q Is your NSF CV publicly available?

A That's a good question. I don't know if it is publicly available.

Q Do you -- you're a -- you're a professor at the University of California, Irvine; is that right?

A Correct.

Q Do you publish your CV on -- on the university's webpage?

A No, we don't. We just have a summary on the website. If you Google, you'll find out.

Q So based on your CV, you've -- you've never worked at a graphics processing company, correct?

MR. PLUTA: Object to form.

THE WITNESS: Well, I worked for our start-up, and one of the application was for graphics processing.

BY MR. TUMINARO:

Page 19

Q The start-up being the MorphoSys?

A Yes, Morpho Technology.

Q Morpho Technologies.

Okay. If we look at Morpho Technologies on your CV, it says:

"Duties: Cofounder; DSP design
for communication and multimedia
systems."

Is that right? What about that signifies that you worked on graphics?

A So we added the functionality of multimedia graphics processing, because we were seeking customers and clients to adopt a technology. And we had a -- you know, we had a hammer. We were looking for a nail.

So basically, we had a parallel processor, an SIMD, to say it more specifically. We were very proud of it because it worked really nicely. And many of these problems are data parallel, so we felt very strongly about being able to apply it to the pixel processing and so on.

Q Okay. So you felt strongly that you could apply it to pixel processing. Did you actually apply it to pixel processing?

MR. PLUTA: Object to form.

Page 20

THE WITNESS: We did. We did, yeah.

BY MR. TUMINARO:

Q Did you actually do that work?

A I'm sorry?

Q Were you the one that did that work?

A I was the key engineer, yes.

Q Okay. Is it -- you've worked on digital signal processors?

A Sure, yes.

Q DSP for short; is that right?

A Yes.

Q You've worked on system-on-a-chip architectures?

A I have, and I continue to, yes.

Q You've continued to work on that?

A Uh-huh.

Q You've published papers on a system-on-a-chip architecture?

A All the time.

Q You continue to do research on system-on-a-chip architecture?

A That's what we do.

Q Okay. And you also work on processor architecture in general?

A That's all my life, yes.

Page 21

Q That's your research?

A Yes.

Q You've written papers on processor architecture?

A Yes.

Q Okay. All right. Let -- let's -- let's look at your papers, if you would. On page 5 of your CV, there's a section that reads "Journal" -- well, it starts with "Publications."

Do you see that?

A Yes.

Q And in the "Journals" section, there are listed, starting on page 5 and spanning to page 13, 88 journal articles; is that right?

A Right.

Q And then starting on page 13, there's a section that reads "Journal" -- "Journals (other)."

Do you see that?

A Yes.

Q What does that refer to?

A These are not -- okay. Yeah. When -- when you have a special issue in a journal -- actually, I see a typo here.

When you see a special issue in a journal, then there would be editors that would be writing a

Page 22

summary, an overview of what the special is all -- special issue is all about. So the "Journal (others)" means that we wrote an opening statement about what this particular special issue is about. For instance, let's say there was a special issue for configurable computing, like jo3.

So Fadi Kurdahi and I, plus others, wrote an opening remark. So this is really not peer-reviewed. That's why it's called "Journals (other)."

Q I see.

A All the other ones are peer-reviewed, and we really don't count these for promotions. And this is just a service to the society, IEEE society, and whoever else is in charge. We have to do that.

I -- I don't know how it is in other fields, but it takes time because you have to review those selected papers and make a summary of what -- what's coming up, and so it takes time.

Q Okay. So under "Journals (other)," you have six articles that are listed there; is that right?

A Right.

Q Okay. And after "Journals," starting on page 13 of your CV, there's book chapters; is that right?

Page 23

A That's right.

Q And there's five listed book -- book chapters that you've contributed?

A Correct.

Q Okay. Then on page 14, there's conference papers, I -- I assume; is that right? Is that what that means?

A Yes.

Q They're refereed?

A That's right.

Q Okay. And starting on page 14 and spanning all the way until -- I guess it's page 29, there's 160 conference papers listed on your CV?

A Yes.

Q Okay. Then starting on page 29, there's another section that reads "Technical Reports," correct?

A Correct.

Q Okay. And from page 29 to page 30 of your CV, there's 13 technical reports that are listed?

A Yes.

Q So in total, if you add up all those papers on your CV, you have 272 papers listed on your CV?

A Yes.

Q Okay. And only eight of them are directed to

Page 24

graphics processing, right?

MR. PLUTA: Object to form.

THE WITNESS: I don't know.

BY MR. TUMINARO:

Q Well, I counted, and there's only eight of them. We could go through them if you'd like. If you look at j54.

A Okay. I appreciate your due diligence on this. I -- I would be glad to verify them.

Q On page 10, j54.

A Okay. Okay. j54.

Can I -- I just want to put a checkmark to -- j54. Okay.

Q Is that directed to a graphics paper?

A It is.

Q Okay. How about j71? Is that a graphics paper on page 11?

A Yes.

MR. PLUTA: Object to form.

BY MR. TUMINARO:

Q It is?

A Yes, it is.

Q Okay. How about c47, which appears on page 18 of your declaration -- I mean of your CV. I apologize.

Page 25

A Yes.

Q c76 on page 21 of your CV, is that related to graphics?

A Yes.

Q All right. c81, page 21, is that related to graphics?

A Yes.

Q c87 on page 22, is that related to graphics?

A Yes.

Q c89, is that related to graphics?

A Yes.

Q And c95 on page 23, is that related to graphics?

A Yes.

Q Those are eight papers that you say are related to graphics.

Are there any others that are listed on your CV?

MR. PLUTA: Object to form.

THE WITNESS: Yes.

BY MR. TUMINARO:

Q Where?

A So let's start.

Q If you'd --

A Yes.

Page 26

Q -- take a look at it and tell me if there's others.

A Sure. So some of these are tangentially related. Like j7, "Finding circular shapes in an image on a pyramid architecture." Some of these techniques could be used to create a primitive, like primitives for 3D graphics because you're trying to figure out the shape. And primitives could be triangles or could be some other exotic shape. So that's -- that -- those algorithms are useful, so I would consider that as one related to that.

So j12, although it says "image processing," but the concept of a hierarchical pyramid architecture, it's very relevant to the graphics because of the way -- as -- as you may recall, we do the vertex processing, and then it goes down to the pixel processing.

So this hierarchical level works for image processing and graphics. And people have looked at this hierarchical architecture. And to some extent a unified shader probably is trying to do something like that. So I would put a little bit of checkmark, at least from my point of view.

So let me finish all of this, and then if you have any questions --

Page 27

Q Sure.

A -- I will answer that. But I'd like to go through this.

Q Okay. You want to --

A It would not take much time.

Q Okay. Sure. Go -- go right ahead. Please do.

A So the j14, also circle detection, it goes back to primitives and identifying primitive objects, although usually people use triangles, but that helps consolidate, you know, triangles or whatever into circles. So that is -- that could be used for graphics, as well.

Q Well, I'll stop you. In that paper, was it actually directed to graphics?

MR. PLUTA: Object to form.

THE WITNESS: It was for identifying circles.

BY MR. TUMINARO:

Q Okay. And you mentioned that circles could be used in graphics, but was it actually -- in that paper, were you discussing graphics processing?

MR. PLUTA: Object to form.

THE WITNESS: Not directly. Indirectly.

BY MR. TUMINARO:

Q Okay.

Page 28

A Again, j19 is about pyramid architectures. So that's -- let me go through this more here.

So I'm down to j40, j41.

So I -- let me -- before forgetting, what we did at Morpho, as you can imagine, it was a start-up. We had investors, and we had -- it was not possible to publish some of that work. So we did a lot of work on using SIMD for graphics processing, in particular, pixel processing, and they were not publishable.

I just wanted to mention that to you for the record, because there is no -- any documentation on that. And that's true for many companies, by the way.

Q Okay. And before you go on, I notice that some of these papers, you're listed as the first named author, and some you're -- you're the second or third or fourth or fifth named author.

A Uh-huh.

Q How is that determined, what order that you're an author?

A So that's a very interesting point of view for faculty. All of these are our ideas or, in this case, my ideas. It's just when a student is working with me, we try to give him more credit. Some

Page 29

faculty don't follow this strategy. I do and many others do.

So we put their names first and ours in the back, kind of in the middle or at the end. And if they help write some of this, they will get some credit for that, as well. We do writing, too, but it's the style of the faculty. I would say 75 percent follow what I do. Some like to see their name first.

Q So if you're listed as a first named inventor -- I mean, first named author, I apologize, that means you did more of the work on that paper?

MR. PLUTA: Object to form.

THE WITNESS: Not really, no, no, because the ideas come from us.

BY MR. TUMINARO:

Q Come from us, who's us?

A The faculty.

Q Okay.

A Me in this case. We give the ideas. We provide the -- I mean, the whole idea of MorphoSys architecture was my idea. But writing the code and implementing it, some of the students helped out.

Q Okay. All right. So then I'll ask, why don't you go through your -- your papers, and if you

Page 30

find any additional ones that you think relate to graphics --

A Sure. I'm almost done.

Q -- you can let me know.

A So again, I repeat that what happened in Morpho was not published related to graphics.

c78, it's "Hardware Accelerated Voxel Carving." Voxel is a three-dimensional pixel, so it is definitely graphics. And it's published in graphics -- computer graphics symposium, so definitely that should be counted in your list, among other things that I've mentioned, c78.

Q j78, you mean?

A c78.

Q c78.

A And you can see that it says "computer graphics symposium."

"Recovering 3D Metric Structure and Motion for Multiple Uncalibrated Camera;" that's also computer graphics, c83.

There are a couple of papers here. We tried to improve the SIMD to have branches. And that was in preparation for applying it for graphics computation, because you want to do a conditional statement in SIMD, which we did not have before.

Page 31

And that's what c85 with Anido and also with c81 which you already identified.

"Persepolis: Recovering History with a Handheld Camera," that is definitely graphics you're trying to identify.

"Image Based Mesh Reconstruction and Rendering," that's also --

Q You're saying c90?

A c90, c91, "Camera Calibration Long Image Sequences," those are all related to graphics.

"Automatic creation of three-dimensional avatars," that's also graphics.

Again, c98, anything with avatars, you're trying to do -- carving these three-dimensional pixels, some of the details, I mean -- yes, c100, c98.

So I would say there are at least twice as many as you mentioned that are directly or indirectly related.

Anyway, so...

Q Okay. Let's look at your litigation support experience that starts on page 2 of your --

A Yeah, sure.

Q -- CV, and that spans to page 5 of your CV; is that right?

Page 32

A Yes.

Q Okay. So before you got involved in these cases between LG and ATI at the patent office, had you ever -- well, strike that.

Before you got involved with these cases, LG versus ATI, you never worked on a graphics case; is that right?

MR. PLUTA: Object to form.

THE WITNESS: No, that's not correct.

BY MR. TUMINARO:

Q That's not correct.

Which one of the cases listed on your CV relates to graphics?

A Fish and Richardson, RIM ITC versus GPH.

There's also one more, but it's just -- the Milberg -- 2011, Milberg, Class versus NVIDIA.

Q Any others?

A I just looked at it quickly. And that's about it, I think, for now.

Q Okay. When you were at Morpho Technologies, I think you said eventually you added more graphics capabilities to the MorphoSys?

A It -- it had the capability. We just tried different applications on it to just be able to do pixel processing and so on. You kind of -- you can

Page 33

see that a graphics computation is really an array, to the array. And the MorphoSys was perfectly designed for that purpose.

Q Switching gears, I'd like to look at your declaration, Exhibit 1.

A Okay.

Q Starting on page 11 of that declaration, there is a large heading number III, "Technology Background."

A Yes.

Q Okay. In this technology background, it spans all the way to page -- the bottom of page 16; is that right?

A Yes.

Q Oh, actually, there's one word at the top of page 17, "processing"; is that right?

A Yes.

Q Okay. In this entire section about the technology background, there's not a single citation; is that right?

MR. PLUTA: Object to form.

THE WITNESS: I can check.

I don't see one, yes.

BY MR. TUMINARO:

Q Where did the information for this section

Page 34

come from?

A My knowledge of the field.

Q Okay. So as an expert in graphics, you're knowledgeable about graphics technology?

A Yes.

Q You're knowledgeable about the evolution of graphics technology?

A Yes.

Q If you turn with me to page 27, there is a Figure 1 that is shown in --

A Page 27?

Q I'm sorry, I meant paragraph 27. If I said page 27 --

A Right.

Q Paragraph 27, there's a figure shown in paragraph 27.

A Correct.

Q Where did that figure come from?

A From one of my sources that I had or what's on the web.

Q Did you generate this figure?

A It's from one of my references or from what's available on the Internet. It could be from a specific source. I -- I -- I did not exactly draw this.

Page 35

Q Okay. So this came from something on the Internet or some other source?

A Yeah.

Q If you look at your materials considered, there's -- that Internet or other source is not listed, is it?

A That source? No.

Q Okay. If you turn with me to paragraph 31 of your declaration --

A Yes.

Q -- the first sentence of paragraph 31 reads:

"The first programable pipeline (PP) was introduced in 2001."

Do you see that?

A Yes.

Q What's the basis for that statement?

A Just my knowledge, what I -- what I know from what I've read.

Q What pipeline was it that you were referring to there?

A Used for graphics.

Q Well, what was the product? Is there a product that's associated with that?

A No.

MR. PLUTA: Object to form.

Page 36

THE WITNESS: I don't have a product. I just remember that.

BY MR. TUMINARO:

Q You just remember that?

A Right.

MR. TUMINARO: I'll like to have that marked.

(Exhibit 4 was marked for identification by the court reporter and is attached hereto.)

BY MR. TUMINARO:

Q I've handed you what's been marked as Exhibit 4 for identification purposes. This is a document entitled "How GPUs Work."

Do you see that?

A Yeah.

Q And if you'll turn with me to page -- the second page of -- of Exhibit --

A Uh-huh.

Q -- 4.

MR. PLUTA: I'm just going to object to the introduction of this evidence as irrelevant at this point, lack of foundation.

MR. TUMINARO: I'm trying to get to the foundation.

Q If you look with me to the right of page -- it's listed 127, in the right-hand column there's

Page 37

a -- sort of midway through there's a sentence that starts with "for example."

Do you see that?

A Yes.

Q I'll read it for the record:

"For example, the NVIDIA GeForce 3, launched in February 2001, introduced programmable vertex shaders. These shaders provide units that the programmer can use for performing matrix-vector multiplication, exponentiation, and square root calculations, as well as short default program that use these units to perform vertex transformation and lighting."

Do you see that?

A Yes.

Q Is that the product that you were referring to that was a programmable pipeline in 2001?

A I'm not sure. I'm not sure if that was the product or there were other products, but it seems to be consistent with what I said.

Q Okay. So when you wrote this sentence in paragraph 31, you didn't have any specific product

Page 38

in mind?

A No, no.

Q Just your general knowledge of the GPU evolution in technology?

A Correct.

Q If you turn with me to paragraph 32 of your declaration.

A Yeah.

Q The first sentence reads:

> "OpenGL and DirectX are now the common standards that most hardware vendors support as part of their graphics card development environment."

Do you see that?

A Yes.

Q When you say "now," what time frame are you referring to?

A Well, now meant December 2014.

Q Okay. So you're aware of your general knowledge of GPUs that at the time that the 871 patent was filed, neither DirectX nor OpenGL provided a unified shader architecture, correct?

MR. PLUTA: Object to form.

THE WITNESS: So am I aware that at the time

Page 39

frame that 871 was filed, there was a unified shader or not?

BY MR. TUMINARO:

Q There --

A Can you repeat the question? I'm sorry.

Q Sure. So you were aware, based on your general knowledge of GPUs that at the time that the 871 patent was filed --

A Which is --

Q -- which is 2003, neither DirectX nor OpenGL provided a unified shader architecture, correct?

A I can't be certain on that one, sorry.

Q How about this? You're aware that DirectX 10 was the first version of DirectX that had a unified shader model, right?

MR. PLUTA: Object to form.

THE WITNESS: Did I say that in my report?

BY MR. TUMINARO:

Q I'm asking you the question.

A It's not in here, so I don't -- I did not opine on that, but, you know --

Q So based on your general knowledge of graphics, you're not aware one way or the other whether DirectX 10 was the first API to have a unified shader architecture?

Page 40

A The --

MR. PLUTA: Object to form.

THE WITNESS: The first --

MR. PLUTA: Lack of foundation.

THE WITNESS: -- I could not tell you, but the DirectX 10 does because it's the latest one or one of the latest ones that we have. So -- so I would say -- I can't tell you if it was the first one or not. I cannot tell you that.

MR. TUMINARO: I'll have this marked.

(Exhibit 5 was marked for identification by the court reporter and is attached hereto.)

BY MR. TUMINARO:

Q So you've been handed what's been marked as Exhibit 5. This is titled "Exploiting the Shader Model 4.0 Architecture."

Do you see that?

A Yeah, yes.

Q And it says in the abstract -- I'll read -- the first sentence says:

> "The Direct 3D10/SM4.0 system is the 4th generation programmable graphics processing units (GPUs) architecture. The new pipeline introduces significant additions and

Page 41

> changes to prior generation pipeline."

And then skipping a sentence, it says:

> "The main facilities introduced that we ponder upon are, Unified Architecture providing common features set for all programmable stages."

And then it goes on.

Do you see that?

A Okay.

MR. PLUTA: I'm going to object to the relevance of this exhibit and also lack of foundation. Also, hearsay.

THE WITNESS: I'm seeing this for the first time, so -- I mean, judging from that paragraph, it says what it says. What do you want me to add to that?

BY MR. TUMINARO:

Q I was going to ask the question.

A Right.

Q So does this -- does this jog your memory, as an expert in graphics, that Shader Model 4.0 in DirectX 10 was the first to introduce a unified shader architecture?

Page 42

MR. PLUTA: Object to form.

THE WITNESS: I could not answer that based on this document. I have to do more research on that.

BY MR. TUMINARO:

Q So you don't know one way or the other; is that your testimony?

A That is not my testimony. I need more work on that. I have some ideas, but I'm not going to render my ideas based on just one article.

Q Well, what's your idea, then --

(Simultaneous speaking - unreportable.)

MR. PLUTA: Object to form, calls for speculation.

THE WITNESS: I need more work on that to give you an answer.

BY MR. TUMINARO:

Q You can't -- do you have an idea of when the first unified shader architecture came out?

A I can't give you for certain what date.

Q What's your best understanding of what the date is?

A I can't guess.

Q I -- I didn't ask you to guess. I asked, what's your best understanding?

Page 43

MR. PLUTA: Object to form.

THE WITNESS: I'd have to guess, and I'm not going to guess.

BY MR. TUMINARO:

Q Okay. You are aware, though, now, as of today, OpenGL and DirectX provide a unified shader architecture?

MR. PLUTA: Object to form, relevance.

THE WITNESS: Yes.

BY MR. TUMINARO:

Q Okay. And you know that -- you've heard of NVIDIA?

A Yes.

Q And you understand that NVIDIA initially was skeptical of whether unified architecture would work?

MR. PLUTA: Object to form, relevance.

THE WITNESS: I have no idea about their position on that.

BY MR. TUMINARO:

Q So based on your general knowledge of shader architecture and graphics evolution, you have no -- no understanding one way or the other?

MR. PLUTA: Object to form.

THE WITNESS: I would not guess what NVIDIA

Page 44

felt. We just looked at their architecture; we still do. We look at their papers. I'm interested in architectures. I'm not interested in how they view and what they interpret what is possible or not.

BY MR. TUMINARO:

Q And just to be clear, you didn't consider that in forming your declar- -- the opinions expressed in your declaration; is that right?

MR. PLUTA: Object to form.

THE WITNESS: I did not mention it here.

BY MR. TUMINARO:

Q So given that now the two major APIs, OpenGL and DirectX 10, provide a unified shader model, it would be more efficient for graphics hardware to implement a unified shader; isn't that right?

MR. PLUTA: Object to form.

THE WITNESS: What do you define efficiency?

BY MR. TUMINARO:

Q How would you define efficiency?

A You asked me the question about efficiency. What efficiency do you have in mind?

Q Well, that's what I'm asking you, how do you understand the word "efficiency" in the context of graphics processing?

Page 45

A This is a very broad question. What is -- what is the application?

Q Graphics processing.

A Counsel, there's a graphic processing in your smartphone. There's a graphic processing in your desktop.

Q Okay.

A There's a graphic processing for applications in military and civilian applications. Which one do you have in mind? You're asking a very broad question.

Q Okay. So I have in mind graphics processing in my cell phone or in a desktop.

A You are asking that?

Q Yeah, in either one of those, what would it mean to be efficient?

A Okay. Efficiency means computation efficiency. Efficiency could be battery power efficiency. And between the two applications, I would think that you agree with me that the smartphone battery power efficiency's a lot more important than your computation efficiency.

Efficiency could mean cost. It's more efficient to have -- well, maybe you don't use the word "efficiency" for cost, but it means lower cost

Page 46

that gives you the targeted efficiency. So it really depends.

If your -- if your unified shader is going to cost you more for grandma's phone, I might opt for a -- just a hardwire approach. Just it's cheaper. I might just do that. I don't think NVIDIA or anybody will -- will dispute that.

So it really depends on what your target application is and what your cost is. If the cost of unified -- unified shader comes down to the level of that is acceptable to many target applications, then that would be the way to go.

Q You mentioned that if you were going to put it in grandma's phone, you might want to just use a hardwire approach.

A Uh-huh.

Q What's a hardwire approach?

A It's the FFP, the fixed function pipeline basically, or used to be.

Let's say the unified shader is very exotic. It probably is all the latest technologies that you have, but I really don't want that. I really don't want -- you don't want to have an 8-cylinder car. You're just going to the grocery store, and that's all you're doing. You're not racing. That would be

Page 47

the grandma's. Then you may want to use -- then you may want to use an ASIC approach, basically, where -- where you just -- simple and already proven, and it's very efficient cost-wise and probably power-wise, too.

So it really depends. And I don't think anybody would dispute that.

Q So are you suggesting that graphics processors are now moving toward more of a hardwire approach?

MR. PLUTA: Object to form.

THE WITNESS: That's not what I said. I said it depends on your application. A unified shader gives you the capability through software means. It's a lot more advanced. It's the way to go. But there are applications that you may find out that unified shader is too much of a burden in terms -- in terms of power consumption, cost and so on.

If the cost and power consumption comes down towards the ASIC solutions, then yes, that would be the ultimate graphics engine.

BY MR. TUMINARO:

Q So you're --

A We're struggling with this right now in all -- all the things that we do that are

Page 48

programmable.

Q So you're aware that in graphics processing, initially it was fixed pipeline, correct?

A Yes.

Q Okay. And then it moved to a programmable pipeline, correct?

A That's right.

Q And unified pipeline is now sort of the --

A Program.

Q -- state of the art kind of pipeline?

MR. PLUTA: Object to form.

THE WITNESS: Yeah.

BY MR. TUMINARO:

Q And you said it's the way to go; is that right?

A How often do you charge your phone? You charge it a lot. You've got an 8-core processor on your phone, so we are getting to a point of wanting to know the efficiency of the execution. A programmable solution is good for lots of people.

I'm not saying we're going back to the ASIC solutions. I'm just saying when you talk about efficiency, you have to be more accurate in terms of what is -- what problem are you trying to solve? You have to -- you have to qualify that. What --

Page 49

what do you mean by efficiency. That's -- that's what I'm trying to tell you.

I'm not saying we're dropping off unified shader or programmable solutions. I'm not saying that at all. If I said that, I maybe mis- -- misspoke.

Q So you're aware in your understanding of graphics processing that the industry has moved away from a fixed pipeline, right?

MR. PLUTA: Object to form, lack of foundation.

THE WITNESS: I said as much in my report, but I'm saying if -- if efficiency -- power efficiency's important, there might be solutions that are not that way. That's what I'm trying to tell you.

BY MR. TUMINARO:

Q Okay. My question was not about efficiency. I'm just saying, are you aware that it moved -- the industry has moved away from fixed pipeline and now it's toward a programmable pipeline?

MR. PLUTA: Object to form --

BY MR. TUMINARO:

Q You said that in your declaration, correct?

A Correct. But your original question was

Page 50

about efficiency, if you recall. Are we -- are we out of that question?

Q Just to remind you how this works, I ask a question; you answer the question that I ask; is that right?

A Yes. Then I'm confused about what question you're asking.

Q Okay. So now I'm asking the question.

You're aware that the industry has moved away from a fixed pipeline and has moved toward a programmable pipeline, correct?

MR. PLUTA: Object to form.

THE WITNESS: I said as much in my report.

BY MR. TUMINARO:

Q Okay. And the industry -- after the -- the programmable pipeline is moving toward a unified shader pipeline, is that correct, the graphics processing industry?

MR. PLUTA: Object to form.

THE WITNESS: That is correct.

BY MR. TUMINARO:

Q Okay. And the unified shader architecture, it's the thing of the future, correct?

A What do you --

MR. PLUTA: Object to form.

Page 51

THE WITNESS: What do you mean by the thing --

BY MR. TUMINARO:

Q Well, you said it's the way to go.

A If I talk about efficiency -- if your power -- if your power consumption is an issue for you, we may find changes on that model.

BY MR. TUMINARO:

Q Okay. How about this. What if your factor that you're considering is performance, not -- not power consumption, performance? If you want to increase performance, would you move toward a unified shader architecture?

MR. PLUTA: Object to form.

THE WITNESS: Okay. You may be shocked by this, but an ASIC solution is far more efficient in computation than a programmable unified shader, but it's not programmable because it will have problems in case there's a bug or there are fixes to be made. So that's why unified shader is better, because you can change it, you can modify it easy, and you don't have to retape the device.

But it is a well-known concept that the hardwired solutions are more power efficient and they're more computation efficient. But a

Page 52

programmable unified shader is the way to go because people don't want to design new chips every time there's a bug. So I hope you understand my position on that.

BY MR. TUMINARO:

Q Okay. Switching topics, I'd like to talk about -- you've heard of what a -- a register? You know what a register is?

A Register in the context of?

Q Computer processing.

A Yes.

Q What's a register?

A Stores information.

Q Okay. A register is typically a multi-ported storage unit?

A Typically. What's your application?

Q Graphics processing.

A I mean, yes and no, depending on what you're trying to do. I have to see what the diagram looks -- schematic looks like to tell you what you're trying to do.

Q Okay. A register is typically the closest piece of memory to an ALU?

MR. PLUTA: Object to form.

THE WITNESS: Generally, yes.

Page 53

BY MR. TUMINARO:

Q And a memory is typically something different than a register, correct?

MR. PLUTA: Object to form.

THE WITNESS: I don't understand why you distinguish between the two.

BY MR. TUMINARO:

Q Well, a memory is typically further from an ALU compared to a register?

MR. PLUTA: Object to form.

THE WITNESS: Proximity to ALU doesn't identify the differences. It just stores information. Remember what I defined register to be, stores information; so does memory.

BY MR. TUMINARO:

Q And a memory is typically a single-ported storage unit as compared to a register, which is typically a multi-ported?

A No. You can have dual port memories, too. I have designed dual port memories.

Q Well, let me ask you this. A register is a different thing than a memory, right?

A In terms of storing information, no.

Q But in terms of difference -- they're different, correct? A register is different than a

Page 54

memory?

A In what way? I don't -- I mean, they have different names, yes, they are different names. But they store information and you retrieve information. So from that point of view --

Q So then why does the industry use a different --

MR. PLUTA: Hold on, Counsel. I don't --

MR. TUMINARO: I thought he was done.

MR. PLUTA: I don't think he was done.

THE WITNESS: I wasn't done, yeah. Sorry.

Let me identify what a block of storage is. You write to it and you read from it. Both of these entities do that. Register and memory are indistinguishable in terms of writing to it and reading it.

In terms of volatility, they both could be nonvolatile or could be volatile, both of them. So I don't see any difference in terms of functionality.

BY MR. TUMINARO:

Q Well, the industry has come up with two different names for these storage units, right? A register and a memory, correct?

A Correct.

Page 55

Q So are you telling me that it's -- it's typically not the case that a register is multi-ported?

A I don't know if I said that.

Q So is a -- is a register typically multi-ported?

A It depends on your application. I mean, you could have an accumulator, which is a register, and it could be one-ported. It really depends on what application you have. It's really -- you cannot generalize that, application dependent.

Q So in your mind, is a register the same thing as a memory?

MR. PLUTA: Object to form.

THE WITNESS: The functionality is very similar in terms of reading, writing, storing information for a period of time. From that point of view, the same. So it stores data.

BY MR. TUMINARO:

Q Are there any differences?

A So register is a little faster than a memory -- actually, a lot faster, yeah.

Q Anything else?

A As in -- you asked the question earlier, it's closer to the ALU or the processor, so that's the

Page 56

case, as well. So those are very important issues.

Q Those are two differences. Any others between a register and a memory?

A The actual circuit design could be different because you're trying to save power, save energy and so on. So yeah, if you are asking memory, it would be a little different, but the functionality's identical.

Q Okay. Anything -- any other differences that you can think of between an AL- -- between a register and a memory?

A It stores data -- no, I think I mentioned all the big differences. Performance, proximity to ALU and the circuit actually is being different.

Q Okay. You've heard of the term "ALU"?

A Yes, I have for a long time, unfortunately.

Q ALU means what?

A Arithmetic logic unit.

Q Okay. You've heard of the term "processor unit" -- strike that, "processor"?

A Yes.

Q Okay. Is a processor the same thing as an ALU?

MR. PLUTA: Object to form.

THE WITNESS: Okay. Traditionally, if this

Page 57

is a class, which I will have in two weeks, a processor includes an ALU and register file and any other things that you can think of, a processor, a processor, a microprocessor.

BY MR. TUMINARO:

Q So -- so you're saying a processor has additional structure that's not in an ALU?

A If you define the microprocessor the way we define it in our textbooks, a microprocessor has those components.

Q An ALU and register?

A Register file and other things, address decoder and so on --

Q So --

A -- and program counter and so on. Sorry.

Q So an ALU can do functions like add, subtract, compare, correct?

A Sure, yes.

Q Whereas a processor can run instructions, right?

A No. Remember when I put the -- the -- the area around the processor, the processor included the ALU and the register file.

Q So, what, you're saying a processor doesn't run instructions?

Page 58

A I didn't say that.
MR. PLUTA: Object to form.
BY MR. TUMINARO:
Q So -- so my question was, a processor can run instructions, right? And you said no.
A So let me --
Q So let me ask the question again.
A Yeah, please.
Q A processor can run instructions, right?
A A processor executes instructions from its instruction memory. And based on those instructions, it will activate the control signals for all the internal components, which means activates the register file, activates the ALU and a few other blocks.
Q Just so I'm clear because I'm not -- that means yes, a processor can run instructions; is that right?
A I think I just said that, yeah.
Q I just wasn't clear. I didn't know if that was a yes or not.
A The answer is the processor executes instructions based on a program counter, which is basically sequencing through the instruction memory, and activates -- I'm giving you a little bit more

Page 59

answer because --
Q Okay.
A -- I think it has to be clear that ALU does the function on behest of the instruction decoded.
Q Okay. So just to be clear, though, an ALU by itself does not run instructions, right, or cannot run instructions by itself?
MR. PLUTA: Object to form.
THE WITNESS: Instruction has to be decoded to activate the control signals for the ALU. Some people mistakenly call ALU the microprocessor. That's -- that's their definition. But, yes, ALU gets its control signals from the decoded instructions.
BY MR. TUMINARO:
Q Okay. So you would say it's a mistake to call an ALU a processor?
MR. PLUTA: Object to form.
THE WITNESS: The ALU by itself, you may call it the processor, but that's not what textbooks usually talk about.
BY MR. TUMINARO:
Q So the textbook definition would say that a microprocessor is not the same thing as an ALU?
A This is like saying that mechanical textbooks

Page 60

will say the engine is different from the car. The car includes the engine. This is the same relationship. So it's the only way I can answer your question without confusing myself.
So if the processor is the car, engine is the ALU, is that -- does that help you with -- to answer?
Q That helps. But -- so just in your example, a car is not the same thing as an engine, right?
A It includes the engine.
Q It includes the engine, but they're not the same thing?
A Where do you say the car is? I mean, I don't -- I don't understand. Would you call the shell of the car to be the car? I don't know.
Q Okay. At least I understand your testimony.
A Okay. Thank you.
MR. PLUTA: We've been going for about an hour. Is it a good time for a break?
MR. TUMINARO: Oh, yeah, sure, yeah, yeah.
THE VIDEOGRAPHER: We are off the record. The time is 10:45 a.m.
(Recess.)
THE VIDEOGRAPHER: We're back on the record. The time is 10:59 a.m.

Page 61

Please continue.
(Exhibit 6 was marked for identification by the court reporter and is attached hereto.)
BY MR. TUMINARO:
Q Welcome back, Dr. Bagherzadeh.
A Thank you.
Q You have in front of you Exhibit 6, which is the Rich patent, U.S. patent number 5,808,690.
You recognize this document, right?
A Yes, I do.
Q You considered this, the Rich patent, in preparing your declarations --
A Yes.
Q -- in this case?
(Exhibit 7 was marked for identification by the court reporter and is attached hereto.)
BY MR. TUMINARO:
Q Okay. You also have in front of you Exhibit 7, which is U.S. patent number 6,897,871.
You -- I'll refer to this as the 871 patent. You'll understand what I'm talking about?
A Yes.
Q And you considered the 871 patent in preparing your declaration, correct?
A Yes.

Page 62

Q Okay. And, in fact, just to be clear, it's your opinion that Rich renders claim 15 of the 871 patent obvious, correct?

A Yes. Right. Rich teaches claim 15, correct.

Q Okay. Now, if we look at claim 15 of the 871 patent, there are sort of three elements recited. There's a unified shader comprising a general purpose register block, a processor unit and a sequencer, right?

A Yes.

Q Okay. Now, I think I have this right. You're saying that -- well, okay. Strike that.

Let's look at Figure 2 of Rich.

A Uh-huh, yes.

Q Figure 2 of Rich, it shows an architecture that's disclosed, a computer architecture that's disclosed in Rich, right?

A Correct.

Q And there is an ALU 33 and a memory 34 in Figure 2 of Rich?

A Correct.

Q And there is a processing element array control 40 in Figure 2 of Rich, correct?

A Correct.

Q Now, correct --

Page 63

A Processing element array controller, yes.

Q Okay. Now, correct me if I'm wrong, but I -- I think this is what you're saying, that in your opinion, memory 34 corresponds to the claimed general purpose register block; is that right?

A So let's look at -- let's look at the details of this.

Q I'll help you out. You talk about Rich in claim 15 starting at paragraph 214 of your declaration.

How -- how about we do this? How about we do this?

A Okay. Yes.

Q How about what you can do for me is circle in Figure 2 of Rich what you say corresponds to the claimed elements in claim 15 of the 871 patent.

MR. PLUTA: Object to form.

THE WITNESS: I would rather not do that and answer your question, because this is a complicated, very high-level block diagram. And I -- I would not do service to the inventors and what they meant.

But let me try to do it differently and just refer to you -- to -- refer you to my report and see if that's satisfactory to you.

BY MR. TUMINARO:

Page 64

Q Well, I -- I read your report, and that's what I'm trying to understand is -- well, let me ask you this.

Are you saying that general purpose register block claim -- in claim 15 of the 871 patent corresponds to memory 34 in Figure 2 of Rich?

A I was just about to answer that question, if you allow me. It will take a few minutes.

It does provide the resources to the ALU, and you can see it from Figure 12. And that's consistent with providing operands to the ALU from what -- what I --

THE REPORTER: "Providing" what?

THE WITNESS: Operands, o-p-e-r-a-n-d.

So that combination of memory in ALU satisfies that operands are available, and ALU executes them.

BY MR. TUMINARO:

Q Okay. And then --

A And I will continue to --

Q Oh, there's more? Okay.

A And Rich says:

"Each processing element 32 comprises an 8-bit multifunction arithmetic logic unit 33, directly

Page 65

coupled to its own bank of" --

Too fast?

THE REPORTER: Yes.

THE WITNESS: Sorry.

I repeat:

"Each processing element 32 comprises an 8-bit multifunction arithmetic logic unit 33, directly coupled to its own bank of 128 bytes of memory, 34. Each ALU is capable of simultaneously accessing its own memory and can share data with its neighbors via an interconnecting bus architect structure."

So -- so it does provide that functionality of a -- of a register in terms of providing operands.

BY MR. TUMINARO:

Q So when you say "it does," you're saying memory 34, in your opinion, corresponds to the general purpose of register block claimed in claim 15 of the 871?

MR. PLUTA: Object to form.

THE WITNESS: It provides the data that it needs in combination with other components.

Page 66

BY MR. TUMINARO:

Q Okay. What in Rich are you saying corresponds to the claimed processor unit in claim 15 of the 871 patent?

MR. PLUTA: Object to form.

THE WITNESS: So the processing units are those -- those ALUs. That's what Rich says.

BY MR. TUMINARO:

Q Okay.

A The processing elements 32 operating -- processing elements 32. It says that here --

Q Okay.

A -- "processing elements 32
operating normally as a single
instruction multiple data
configuration."

Q Okay.

A Each processing element 32. So that -- that box is called a processing element, which includes an ALU.

Q Okay. Great. That's it?

A Yes.

Q Okay. Now, what in Rich are you saying corresponds to the claimed sequencer in claim 15 of the 871 patent?

Page 67

A The control unit 40:
"The processing element, array
element control unit 40 is primarily
responsible for sequencing
instructions and addresses to the
processing element array 30."

Q Great.

A It's in my chart.

Q Excellent. We're on the same page now. Thank you.

A You are so excited, Counsel. Okay. Okay. Good.

Q All right. Now, you would agree with me that Rich never discloses that vertex data is retrieved from memory 34, correct?

MR. PLUTA: Object to form.

THE WITNESS: Well, this is a graphics processing engine, and clearly talks about performing those operations, so it would have been very clear to whoever is looking at this technology that that's what Rich is talking about, that it can access vertex or pixel.

BY MR. TUMINARO:

Q Okay. I'll ask the question again, because I think I didn't get an answer to my question.

Page 68

A Right.

Q You'll agree with me that Rich never discloses that vertex data is retrieved from memory 34, right?

MR. PLUTA: Object to form.

THE WITNESS: It -- it is -- well, I mean, it does perform the operations for graphics computation. And even if it doesn't say it specifically, it is capable of doing that.

BY MR. TUMINARO:

Q Okay. Let's --

A Because -- because it's -- it is a processing element, it has access to the memory. And it is done for interpolation of the pixels, rasterizations and other functions. So, therefore, vertex processing is part of that, and therefore, it is consistent with my understanding.

Q Okay. Let -- let's look at what you said in paragraph 216. The last sentence says:
"Rich does not explicitly disclose
vertex processing on data retrieved
from processing memory 34."
Do you see that?

A Yes.

Q Do you agree with that statement?

Page 69

A I say it.

Q So it's a true statement?

A It's what I've said, yeah.

Q Okay.

A But -- but it doesn't do it explicitly, but it's implicitly.

Q So then you would agree with me that Rich does not disclose that vertex data is retrieved from memory 34, right?

MR. PLUTA: Object to form.

THE WITNESS: I think I can read what I have here. It doesn't say it explicitly --

MR. TUMINARO: Okay.

THE WITNESS: -- but for an engine that is doing graphics computation, it is -- to me, that's clear.

BY MR. TUMINARO:

Q Okay. But it's your opinion that it would be obvious to modify Rich to store both vertex data and pixel data in memory 34; is that right?

A That is correct.

Q Okay. But doesn't Rich actually teach away from storing both vertex and pixel data in memory 34?

MR. PLUTA: Object to form.

Page 70

THE WITNESS: Where does it say that?

BY MR. TUMINARO:

Q Well, Rich explicitly discloses the vertex data by itself, without even considering the pixel data; the vertex data is too big to fit in memory 34, right?

A Can you refer me to what section of the Rich you're talking about?

Q How about I'll refer you to your own declaration at page -- paragraph 33.

A Paragraph 33?

Q Paragraph 233.

A Okay.

Q And the second sentence reads -- and I'll read it for the record:

"For example, Rich discloses that the user of" --

I think that should be "use," not user; is that right?

A Okay.

Q So I'll read it again --

A Yes.

Q -- with the correction:

"For example, Rich discloses that the use of external memory for the

Page 71

storage of data for processing is necessary because the processing elements 32 only have a small amount of dedicated memory."

Do you see that?

A Yeah.

Q And the next sentence reads:

"Further, transformed vertex data is stored in external memory after the geometry function."

Do you see that?

A Yeah, but it -- but --

Q There's --

A -- I think --

Q Okay.

A -- adding more memory to the processing element 32, it's a minor change, extremely trivial. And the technology slope for that -- when was this disclosed? 1998. That's not an issue anymore for us.

Q Okay. But --

A This is almost 20 years ago.

Q But it's true that on chip memory, because that is so small, Rich teaches that it's necessary to store the vertex data in memory, external memory?

Page 72

A For what he saw then. But a POSITA looking at that technology will not say that I don't have enough memory to put the vertexes in there, because the design is consistent with being able to do vertex processing.

Q Well, didn't you opine that a POSITA would actually read Rich to interpret that the vertex data is stored in external memory?

A POSITA was doing that around early 2003. That's almost seven years or six years after this came out. I would -- I would submit to you that the memory technology had improved probably by -- by at least 30, 40 percent. So I -- I don't see that as a problem.

Q In paragraph 233 of your declaration, did you or did you not opine that Rich discloses that vertex data is stored in external memory, not processor memory 34?

A I'm repeating what Rich says, right? I am quoting what Rich says. I did not -- it says basically -- referring to column 16, 52 through 55, which is consistent with what Rich disclosed and what was the understanding at that time.

What I'm trying to explain to you is that five years later, it would have been definitely

Page 73

clear to the designers to not be limited but at local memory. It would have been -- one of the first things that would improve is the size of the memory. So I would not find that as a problem.

Q All right. That wasn't my question on whether it was a problem. Let's just see if we agree that Rich teaches that vertex data is stored in external memory, not in the local memory 34. Do we agree on that?

A That's what he says directly in his -- in his -- in his invention, yes.

Q Okay.

A But -- but if I could qualify that response by saying that it would have been very clear to a designer to not be limited by that, because the design is so versatile. So you could do vertex and pixel at the same time.

Q Okay. And isn't it a fact that the pixel data is larger than vertex data?

MR. PLUTA: Object to form.

THE WITNESS: It's -- it's hard to say, depending on what format you have in mind. But it could be -- because you have more data for the -- for the triangle, there are more pixels than vertices. But then vertices are -- you know,

Page 74

they're described in 32-bit or 64-bit floating point descriptions. So it depends; the size of the triangle, how many RGB bits you have for the pixel, so it depends.

BY MR. TUMINARO:

Q Let's look at what you said in paragraph 235 of your declaration.

A Right.

Q "That is, the data size of pixels after rasterization is more than the data size of transformed vertices after vertex operations."

A Okay.

Q Do you see that?

A Yeah.

Q So you said that, right?

A I said that and I agree with that, but it depends on the size of the triangles and the type of pixels you have.

If you use a 2-bit RGB, it will not be the case, right?

Q So now, we agree that Rich discloses that vertex data is stored in external memory, correct?

A Rich does that, yeah.

Q Okay. And we agree that pixel data is bigger

Page 75

than vertex data, correct?

MR. PLUTA: Object to form.

THE WITNESS: From what -- what -- what it could be considered that way, yes.

BY MR. TUMINARO:

Q Okay. So then why would you store both vertex data and pixel data in memory 34 and not in external memory?

A The answer is obvious, because you need the data to be closer to the ALU.

Q Well, in fact, didn't you actually opine that it would be obvious in view of Rich to store pixel data in external memory?

A Could you show me where I opine that?

Q Paragraph 234:

"It would have been obvious to one of ordinary skill in the art that pixel data can also be stored in external memory after rasterization and pixel processing" -- "and before pixel processing because of the same" --

A Yeah.

Q -- "concern emphasized by Rich about the small amount of dedicated memory

Page 76

on processing elements 32 themselves."

Do you see that?

A Yeah. It's okay. It's -- it's good. You can keep larger data outside the processor. We do it all the time.

Q So --

A What -- what is the -- what -- the -- the concern you have is -- I'm a little -- you know, I mean, I don't know what -- what concerns you, but I answer your question, because that's what I have to do, is that if you have larger sized data, you leave it outside the processor and you bring it in. That's how systems work. Whether it's the vertex data or it's pixel data, it's really irrelevant.

So if he doesn't say it, that's how it would be done if the internal memory was large enough to hold the data.

So I don't see any inconsistency with what I've said here, what is disclosed here or what an engineer could figure out in 2003.

Q But just so that we're clear, Rich explicitly discloses that vertex data is stored in external memory, not local memory 34?

A The pixel data?

Q Vertex data.

Page 77

A Vertex data. Yes, he says that.

Q And you opine that it would be obvious to modify Rich to store pixel data in external memory, as well, correct?

A Correct.

Q Okay. Yet you still think it's obvious to store both of those things in memory 34 on chip?

A The job of memory 34 chip is to provide executable data to the ALUs. It doesn't matter what it is, it's pixel, vertices, whatever you have, that is -- the ALU wants to see things close to the ALU. So if you have vertex data in the outside memory, in the larger memory, you have to kind of mosey them down to the -- to the local memory or registers, what have you.

So that is the standard practice. That is the standard practice. That's what a POSITA would know, that a processing element would need its operations very close to it, whether it's a vertex or pixel. So that's -- that is inconsistent with the knowledge of a POSITA in 2003 time frame and what has been disclosed and what I am saying in this report.

Q Let's look at paragraph 41 of your declaration.

Page 78

A 41.

Q Paragraph 41 reads:

"I believe that a person of ordinary skill in the art relating to the 871 Patent would be someone with a good working knowledge of computer graphic processing architecture, as well as the systems and programs that support such architecture. The person would also be familiar with graphics standards as well as general data processing architecture and techniques."

Do you see that?

A Yes.

Q It goes on to read:

"The person would have gained this knowledge through a Master's Degree in Electrical or Computer Engineering, or equivalent thereof, and 2+ years of practical working experience in the relevant field."

Do you see that?

A Yes.

Q Are you a person of ordinary skill in the art

Page 79

with respect to your own definition?

A Yes, Counsel.

Q Okay. What do you mean when you say that a person would be familiar with graphics standards as well as general data processing architecture and techniques?

A I think it speaks for itself.

Q Well, what does -- what does it mean to be familiar? What are you referring to?

A Understand them.

Q Understand them. Understand all the intricacies of them? I'm trying to understand the scope of what they would understand.

A Whatever you need to understand, whatever the job duty is, you have to understand.

Q What time frame are you referring to? What graphics standards?

A Early 2003.

Q So that's not in, like, the current graphics standards. It's only the 2003 graphics standards?

A Well, I mean -- I mean, if he has been working or she has been working in the field, will -- will continue if the job required, of course.

Q So they would be familiar, for example -- a

Page 80

person of ordinary skill in the art would know whether a graphics standard required a unified architecture?

A Could --

MR. PLUTA: Object to form.

THE WITNESS: Could be, yeah, depending on what the job requirement was. You could be designing a piece of the graphics engine that would require you limited amount of knowledge. So it's -- really depends on what you're doing.

BY MR. TUMINARO:

Q Turn to paragraph 19 of your declaration.

Paragraph 19 reads:

"I also understand that the relevant inquiry into obviousness requires consideration of four factors."

Do you see that?

A I do.

Q And the fourth factor that's listed there is objective factors, correct?

A They are what factors?

Q Objective factors.

A As opposed to?

Q That's what's written here --

Page 81

A Okay.

Q -- in your declaration:

"Objective factors indicating obviousness or non-obviousness."

Do you see that?

A Yeah. Yes, go ahead.

Q Did you consider, in forming your opinions expressed in your declaration, any objective factors indicating obviousness or nonobviousness?

A I -- I considered these factors that I have mentioned here.

Q Which factors?

A The four factors, scope and content of the prior art -- do you want me to read that, all four of them?

Q Well, in your declaration, did -- did you discuss any of the objective factors in your analysis?

A Discuss --

MR. PLUTA: Object to form.

THE WITNESS: Discuss it with who?

BY MR. TUMINARO:

Q Do you have any discussion in your declaration relating to objective factors?

A These are legal descriptions of what

Page 82

obviousness factors are, and in my analysis, I included whatever was relevant.

Q Well, apart from what appears in -- in paragraph 19, do you discuss objective factors anywhere else in your declaration?

A I can -- I mean, let's go through them one by one. It seems to be that would be easier for me if we break it down into -- than just a general question. The differences between the prior art and the claim at issue, sure, I've done that. I've done that in my -- in my analysis. We can go through that.

The knowledge of a person of ordinary skill in the pertinent art, we just went through that. So I don't understand -- you want me to identify every one of these factors?

Q No, that was not my question.

A Then I don't understand your question.

Q Okay. You say that there are four factors that must be considered in an obviousness analysis, correct? That's what you say in paragraph 19.

A That's right.

Q Okay. And there's -- A is:

"The scope and content of the prior art."

Page 83

Correct?

A Uh-huh.

Q B is:

"The differences between the prior art and the claims at issue."

Correct?

A Uh-huh, yes.

Q C is:

"The knowledge of a person of ordinary skill in the pertinent art."

Correct?

A Correct.

Q And D lists:

"Objective factors indicating obviousness or nonobviousness."

A Correct.

Q Okay. Now, my question is -- I'm not concerned about the first three factors. I want to know about objective factors.

A Okay. I get that.

Q Did you discuss objective factors anywhere in your declaration apart from paragraph 19?

A I considered them, but I did not report them. I mean, it was -- it was definitely something I did

Page 84

not put in my report; that's for sure.

Q So discussion of objective factors apart from paragraph 19 doesn't appear anywhere in your report, correct?

A Yeah. If you find it, let me know.

(Exhibit 8 was marked for identification by the court reporter and is attached hereto.)

BY MR. TUMINARO:

Q Sir, you've been handed what's been marked as Exhibit 8 for identification purposes. This is U.S. patent number 7,015,913 to Lindholm.

Do you see that?

A Yes.

Q You considered -- I'm going to refer to this as the Lindholm patent; is that fair?

A That is fair.

Q And you considered the Lindholm patent when forming the opinions expressed in your declarations, correct?

A Yes.

Q Okay. And if you turn to page 32 of your declaration, there's a section that reads "Ground 1."

Do you see that?

A Yeah, yes.

Page 85

Q So just so we're clear, it's your opinion that Lindholm anticipates certain claims of the 871 patent, correct?

A Correct.

Q Okay.

MR. PLUTA: I'm sorry. Which declaration were you referring to?

MR. TUMINARO: Oh, I'm sorry. I was referring to the -- what's Exhibit 1 with respect to the 871 patent.

MR. PLUTA: Thank you.

BY MR. TUMINARO:

Q And if you look at -- starting on page 32 of your declaration, Exhibit 1, with respect to the 871 patent, there is a section that is -- has a heading labeled "Lindholm Discloses Claim 1."

Do you see that?

A Yes, I do.

Q And that spans all the way to page 34, where, then, there's a claim chart --

A I see that.

Q -- correct? Okay.

And on page 33 of your declaration, you reproduce Figure 4 from Lindholm, correct?

A Yes, I do, correct.

Page 86

Q So if you would go to Figure 4 of Lindholm for me and circle in Lindholm Figure 4 what you say corresponds to the claimed elements in claim 1 of the 871 patent.

MR. PLUTA: Object to form.

THE WITNESS: You want in Figure 4 to identify what's mentioned in claim 1, those components; is that what the question is?

BY MR. TUMINARO:

Q Claim 1 of the 871 patent, yes.

A I would refer you to my chart. I think that's more accurate. If you want, you can go through it one by one. I -- I think it would be easier for me to explain. I would rather do that.

Q Well, I'm asking you if you can mark up Figure 4 of Lindholm and show me where in Lindholm you're saying -- what in Lindholm Figure 4 corresponds to the elements claimed in claim 1 of the 871 patent.

A I would rather read it from the report, which I think is very clear, in my opinion.

So the Lindholm thread control buffer -- so claim 1 of 871 -- so claim 1 of 871 basically says an arbiter -- it's a graphic processing -- the Lindholm talks about graphic processing in the

Page 87

abstract.

Then the -- the next part or the -- of the claim 1 talks about an arbiter circuit for selecting one of plurality of inputs in response to a control signal. And in this case, we have the TCB, which is working as an arbiter here, deciding between the inputs, 215 and 220. These are corresponding to vertex and pixel input data.

Q All right. Let me try this, sir. If you would turn with me to paragraph 90 of your declaration.

A Yes.

Q Paragraph 90 reads:

"Because Lindholm's Execution Pipeline 240 performs both vertex operations such as transforming vertices and pixel operations such as texture mapping and blending, components of the Pipeline 240 correspond to the 'shader' of claim 1."

Do you see that?

A Yes.

Q So what I want to know is what components of execut- -- execution pipeline 240 correspond to the

Page 88

shader of claim 1.

A The shader is the -- the processor that is doing the job in general, right? So that's the -- the execution units.

Q Let -- let me ask again. In paragraph 90, you say:

"Components of Execution Pipeline 240 correspond to the 'shader' of claim 1."

Correct?

A Right, the components of 240. And it's shown here, Figure 4, and all the components are there.

Q So now my question is, which components of execution 240 correspond to the shader claimed in claim 1?

A The shader in claim 1 is identified as Box 62, correct? And that particular function is handled by the execution units.

Q So is it -- is it your testimony -- I just want to make sure I understand -- that execution unit 470 corresponds to shader -- the claimed shader?

A Including all the components that goes with it in terms of the instruction, the sequencers and so on.

Page 89

Q So that's -- that's my question. I want to know what your opinion is. In -- in your -- in paragraph 90, you say:

"Components of Execution Pipeline 240 correspond to the 'shader' of claim 1."

I want to know what components of execution pipeline 240 you're talking about.

Can you identify them for me in Lindholm?

A Sure. I say it in paragraph 95:

"The Execution Unit 470 eventually processes the retrieved samples along with received instructions from Instruction Dispatcher 440 'to perform operations such as linear interpolation, derivative calculation, blending... and output the processed sample to a destination specified by'" -- "'by the instruction.'"

So that should give you the answer, paragraph 95.

Q So is the answer to my question, then, that instruction dispatcher 240 and execution unit 470 correspond to the claimed shader --

Page 90

MR. PLUTA: Object to form.

BY MR. TUMINARO:

Q -- of --

MR. PLUTA: I'm sorry.

BY MR. TUMINARO:

Q -- the 871 patent?

MR. PLUTA: Object to form.

THE WITNESS: There are -- there are components that are related to the shader that help 470 to execute. So I -- I don't know where you want to put the -- the border in terms of where the shader is.

But the shader, if you want to count it as the execution unit, it would be the 470. The instructions are coming from above, as you see in Figure 4.

BY MR. TUMINARO:

Q It's not where I want to put the borders. I want to know where you put the borders. I want to know what your opinion is. Which components --

A And I --

Q -- are -- are corresponding to the shader?

A The shader is --

MR. PLUTA: Object to form.

THE WITNESS: -- defined as the execution unit. I've -- I've said it several times -- I mean, I've repeated my answer several times. The execution unit 470 processes the received samples along with the received instructions from instruction patch -- instruction dispatch 440. So instruction dispatch provides the instructions to 470 to do the computation.

Page 91

BY MR. TUMINARO:

Q Okay. So it's -- 440 and 470 correspond to the unit -- to the shader claimed in claim 1.

Is there anything else that you're pointing to that corresponds to the shader in claim 1?

MR. PLUTA: Object to form.

MR. TUMINARO: I'll ask again.

Q Is there anything else in Lindholm besides instruction dispatcher 440 and execution -- execution unit 470? Is there anything else besides those two components that you're saying corresponds to the claimed shader in claim 1 of the 871 patent?

A The -- the way that I interpret your question is that the shader is the execution unit part of the computation, and that's the one 470 in terms of the computation.

But there are other components that feed into this block. So there are instructions that come to

Page 92

the shader and perform the operation. So that's -- to me, that's clear, at least from what I explained.

Q I guess it's -- it's not clear to me. I don't know -- is it 470 and instruction dispatcher 240 from Lindholm that make up the -- the claimed shader, or are there other things that you are pointing to in Lindholm that correspond to the shader?

A And the define -- the definition of the shader is? You -- the definition that is in 871?

Q The claimed shader in 871.

A The claimed shader in 871. Well, it shows the unified shader in this figure, right, if you're referring to that. Assuming what's put into that 871 -- Figure 5. Hold on a second.

Okay. So the -- looking at 871, the unified shader is defined to be including what's in here, Figure 5, correct?

So that means it includes the instruction store. So I would put anything to do with instructions as part of the shader based on what this -- these guys have defined in 871.

Q Okay.

A And that is consistent with this understanding.

Page 93

Q So what structures, then, are you talking about that correspond to the instructions -- strike that.

What structure in Lindholm are you referring to that corresponds to the instruction store?

A Anywhere you see an instruction, it's referring to this information that is in Figure 5. It says "instruction store." They -- they identified this figure to be the shader based on their definition of a shader. And that's consistent with this figure. So that includes anything to do with the instruction, Counsel.

Q Sir -- sir, I --

A Let me finish my -- you're not -- you're not letting me finish.

Q I didn't say anything.

A You started talking, sir.

But whatever has to do with instruction is part of this shader.

Q That's -- that's what I want to know. What things in -- in Figure 4 or in Lindholm generally are you pointing to that correspond to the claimed shader? You said the execution unit 470, the instruction dispatcher 440.

Is there anything else in Lindholm that

Page 94

you're pointing to that corresponds to the claimed shader?

A So --

MR. PLUTA: Objection; asked and answered.

THE WITNESS: It -- it includes instructions, it includes the register file, it includes anything to do with what has been defined in 871 to be the shader, consistent with that.

BY MR. TUMINARO:

Q So -- just so I understand now, register file 450 in Lindholm you're saying corresponds to the claimed shader, as well?

A Because there are source registers here for the CPU based on Figure 5 of -- of the 871, so there are registers corresponding to this.

Q Okay.

A If -- if you open up execution unit 470, if there are registers in there, that will be corresponding to the -- to the Figure 5.

Q Okay. Anything else in -- in Lindholm, besides instruction dispatcher 440, execution unit 470, register file 450, anything else in Lindholm that you're saying corresponds to the claimed shader in 871?

MR. PLUTA: Object to form.

Page 95

THE WITNESS: Whatever 871 -- 871 defines in Figure 5 to be the unified shader, they have coined that for their -- for their understanding of unified shader. It may be different from NVIDIA. It might be different from somebody else.

So referring to Figure 5, referring to Figure 5 -- I'm going to repeat what's in Figure 5. There are instruction stores. There's a -- there's a constant, which are -- these are specialized functions. There are source register operands. There is a CPU which does the computation, and there is block 92. And that's consistent what's in that page.

(Unintelligible reading) to be process (unintelligible reading) to the general purpose register block 92. So that's -- that's -- if they define that to be the part of the shader, and it's right here, the register file.

BY MR. TUMINARO:

Q Okay. Just -- we'll agree that the 871 Figure 5 is not referring to Lindholm, right?

A Excuse me?

Q Figure 5 of the 871 patent is not referring to Lindholm, right?

A I -- no. I'm just trying to explain to you

Page 96

as defined here for the unified shader, I'm identifying -- that's what you wanted me to do.

Q Right.

A I've identified those pieces that match Figure 5 here.

Q Okay. So Figure 5 is defining the invention or the -- it's describing what's in the 871 patent, right, not what's in Lindholm, right?

A No. I was using that as a model to explain to you what the unified shader defined here corresponds to here. That's what you wanted me to do.

Q Right. And it's your opinion that Lindholm discloses what's in the -- claim 1 of the 871 patent, right?

A Based on Figure 5, yes.

Q Okay. So what I'm trying to understand is the scope of your opinion. I want to know what -- if there's anything else in Lindholm besides instruction dispatcher 440, register file 450, execution unit 470, is there anything else that you're pointing to in Lindholm that corresponds to the claimed shader, in your opinion? I want to know what your opinion is.

MR. PLUTA: Object to form.

Page 97

THE WITNESS: They did not explain the details about how they were --

BY MR. TUMINARO:

Q Who's they?

A 871.

Q Okay.

A There are additional information -- there's just one Figure 5 to show the exploded description -- they use the word "exploded" description of the shader. This is not sufficient to explain whether the score boarding algorithms are in there, whether the other components are there. Because it talks about resources not being available.

If you read 871, which I'm sure you have, it says we will switch from vertices to pixels, depending on resources being available. There is nothing here to tell me how that switching is done.

But -- but Lindholm has a lot more detail in terms of explaining what's going on. So given what's in Figure 5, I have identified what's available in Lindholm, but there are other things in Lindholm that 871 did not discuss.

MR. TUMINARO: I'd like to take a break, I guess, to -- to change the tape, and we'll come

Page 98

back.

THE VIDEOGRAPHER: This -- we are off the record. The time is 11:46 a.m., and this is the end of the first media.

(Lunch recess.)

THE VIDEOGRAPHER: This is the beginning of the second media. We're back on the record. The time is 12:37 p.m.

Please continue.

BY MR. TUMINARO:

Q Welcome back, sir.

A Good afternoon.

Q During the break, did you talk with your counsel about the substance of your testimony?

A No.

MR. TUMINARO: Okay.

Did you give him the exhibit?

THE REPORTER: Yes.

(Exhibit 9 was marked for identification by the court reporter and is attached hereto.)

BY MR. TUMINARO:

Q You have in front of you Exhibit 9. It's U.S. patent number 7,376,811. This is the Kizhepat reference.

A Kizhepat.

Page 99

Q Yes?

A Yes, I have it.

Q And you considered Exhibit 9 in forming the opinions expressed in both of your declarations?

A I believe so.

Q Okay. All right. If we look at the "Field of the Invention," it's in column 1. It reads:

> "The present invention relates to system architectures for data processing, and more particularly to an architecture based upon a hardware engine which performs operations and computations on data as the data traverses paths controlled by software."

Do you see that?

A Column 1, "Field of Invention," right? "Traverses paths controlled by software," yes. You read the paragraph correctly, yes.

Q I apologize. In the -- the declaration that you submitted for the 330 IPR, Exhibit Number 2, you did not consider the Kizhepat reference for that declaration, right? If you look at the materials considered -- strike that.

Just look at the materials considered in

Page 100

Exhibit 2, your declaration, number 330.

A Okay.

Q Are you there?

A Yes.

Q All right. I just want to confirm, I misspoke earlier. You did not actually consider Kizhepat in forming the opinions expressed in the declaration for the 369 patent, which is Exhibit 2?

A Yeah.

Q Right. Okay.

Thank you, Counsel.

All right. Now -- so we read the field of the invention for Kizhepat.

A Kizhepat.

Q Yeah, okay. It's a fact, is it not, that the acronym GPU appears nowhere in Kizhepat?

A Okay. I have to check.

So it's mostly talking about data processing system. By looking at it very quickly, I did not see the word "GPU," but data processing of this type of architecture is consistent with the understanding of doing graphics operations.

Q And the fact is Kizhepat doesn't use the phrase "graphics processor," correct?

A It's -- it -- it -- it is data processing,

Page 101

and some aspects of graphic processing is data processing.

Q Well, answer me this. Is the word -- does the word "graphics" appear anywhere in Kizhepat?

A I looked at it quickly, and I did not come across it, but that doesn't mean I looked at every single sentence.

Q Well, look at every single sentence and tell me, does the word "graphics" appears anywhere in Kizhepat?

A Okay. So all the claims refer to data processing. I looked at the claims. I think, judging from 871, which is the preamble of claim 1, and I think Lindholm also talks about graphics processing. I just want to make sure that's the case.

Yeah, so it's not in the claim description.

Q So we agree that Kizhepat neither uses the acronym "GPU" nor the word "graphics" anywhere --

A In the claim disclosures, right.

Q In fact, it doesn't appear -- in addition to the claims, it doesn't appear anywhere in the patent?

MR. PLUTA: Object to form.

THE WITNESS: I would assume that if they had

Page 102

mentioned it, it would be in the description specification, as well, so...

BY MR. TUMINARO:

Q And in your declaration, when you're talking about Kizhepat, you didn't cite anyplace in your declaration where Kizhepat discloses the acronym "GPU" or the word "graphics," correct?

A I was looking at that. And what I used -- Kizhepat is for combining with Lindholm regarding your multiplexers, the --

THE REPORTER: "Regarding" what?

THE WITNESS: Multiplexers.

And the fact that you're able to select one of those input to the processors.

So that -- that's basically the level of -- as I say in paragraph 158:

> "A typical and common multiplexing system is described in Kizhepat. Kizhepat discloses a hardware engine for data processing that includes a plurality of functional units and data routing units that interconnect the functional units."

So that's been --

BY MR. TUMINARO:

Page 103

Q Okay. I appreciate -- I appreciate the answer, but I think it didn't answer my question.

My question was, in your declaration, when you're talking about Kizhepat, you didn't cite anywhere in Kizhepat just -- that Kizhepat discloses the word "GPU" or "graphics processor," right?

A It's not in my report.

Q Okay. Switching gears to the materials considered, did you review the board's institution decision with respect to the 871 patent?

A I think you asked that this morning, right?

Q Well, I asked if you reviewed anything else. I'm just wondering if you looked at the institution decision.

A You did ask, and I said no.

Q So are you aware of the fact that the board construed --

A I'm sorry. You asked me about the response you gave to -- I apologize. You asked me about the response you gave for some other case.

Did I look at the review -- did I look at the board's response? Yes, I did.

MR. TUMINARO: Okay. Maybe this will make it faster.

Can you grab that.

Page 104

(Exhibit 10 was marked for identification by the court reporter and is attached hereto.)

MR. TUMINARO: This is 10, right?

THE REPORTER: 10.

BY MR. TUMINARO:

Q You've been handed what's been marked as Exhibit 10 for identification purposes. This is -- this is the decision by the PTAB regarding institution of the IPR2015-00326, correct?

A Correct.

Q And you've seen this document before?

A I have.

Q Okay. So I'd like to direct your attention to page 9 at the bottom, the last sentence that -- that bridges across page 9 and into page 10. I'll read it for the record:

> "Thus, for purposes of this decision, we construe the 'means for performing vertex operations and pixel operations and performing one of the vertex operations or pixel operations based on the selected one of the plurality of inputs' to include a register, an instruction sequencer capable of providing

Page 105

> instructions for performing vertex operations and pixel operations, and a processor capable of floating point, arithmetic, and logical operations on a selected input."

Do you see that?

A Yes.

Q Did I read it correctly?

A I didn't check.

Q Okay. My question is, do you agree with the board's construction for this means for limitation?

A My understanding is that we have to accept the board's understanding -- or construction. That's my understanding from legal point of view.

I don't know if there is -- and I agree with it.

> "To include the register, instruction sequencer capable of" -- "we construe the means of performing vertex operations and pixel operations and" (unintelligible reading) --

THE REPORTER: I'm sorry.

THE WITNESS: I'm sorry. I'm just -- I should read it for myself.

Page 106

It's identifying what they say should be included. I would say that plus additional stuff, if you want to get to the details. That's the minimum requirement, I would say.

BY MR. TUMINARO:

Q Are you -- are you saying that the board missed out some structure and there should be additional stuff in their construction?

A No, I'm not saying that. I'm saying these are the basic components. And depending what else you want to do, you may add additional stuff to it. It de- -- it depends on the level of -- so I -- it's consistent with my understanding.

Q Okay. You've heard of the word "arbiter" before?

A In the context of?

Q Graphics processing.

A Yes.

Q Okay. And an arbiter selects from available inputs?

A Correct.

Q Okay. If you'd turn with me to the 871 patent, it's Exhibit 7.

A 87- -- 871. What page?

Q 871 patent, Figure 4A.

Page 107

A 4A.

Q Are you there, 4A?

A Yep, yes.

Q All right. My question is, you see MUX 66 at the top of Figure 4A?

A I see it.

Q Is MUX 66 an arbiter?

MR. PLUTA: Object to form.

THE WITNESS: Does it say it's an arbiter? It says it's a MUX.

BY MR. TUMINARO:

Q Well, that's my question.

A Unless it's a typo.

Q That's my question. Do you consider, in your opinion, MUX 66 to be an arbiter?

MR. PLUTA: Object to form.

THE WITNESS: If -- if you want to implement an arbiter, you need to have a control, and you need to have a way of selecting between inputs. A MUX does that on its own, and a simple control would basically let you select between the inputs. You can basically select which one of the inputs would be the output. You can call that an arbiter if you want, depending on what -- how you control it.

A simple control will be just selecting

Page 108

between the inputs. A more sophisticated control will be the -- the logic behind that control signal.

BY MR. TUMINARO:

Q In the past, you've opined that an arbiter receives and provides, correct?

MR. PLUTA: Object to form. Object to relevance.

THE WITNESS: In the past meaning what? What do you mean in the past?

BY MR. TUMINARO:

Q In --

A In my history of my life?

Q -- in another -- in -- in another proceeding you have opined that arbiter means -- refers to structure that receives and provides, correct?

MR. PLUTA: Object to form. Object to relevance. Counsel, it's a different patent, different references, different terms.

THE WITNESS: I just explained to you what a MUX is. It has two inputs and has outputs in this case, so it's a two-to-one selector.

BY MR. TUMINARO:

Q Is it or is it not true that in the past, in another proceeding, you opined that arbiter is structure that receives and provides?

Page 109

MR. PLUTA: Object to form. Same objection as to relevance.

THE WITNESS: I -- I don't know in what context that was, but in general, a MUX has an input and output. There's no question about it. It's a circuit that has an input and output.

I don't know what -- what -- can you repeat what you just said about what I --

BY MR. TUMINARO:

Q Sure. I'll repeat the question again.

Is it or is it not true that in the past, in another proceeding, you opined that an arbiter is structure that receives and provides?

MR. PLUTA: Object to form. Object to relevance.

THE WITNESS: My answer to you is that it inputs and outputs. Receives an input and generates an output.

BY MR. TUMINARO:

Q An arbiter receives an input and generates an output?

A Of course.

I mean, I -- I don't know what else I can add to what I just said.

Q Going back to your declaration, with respect

Page 110

to the 871 patent, Exhibit 1.

A What page?

Q I'm trying to get there.

Page 32.

A Okay.

Q Paragraph 91 reads:

"Lindholm also discloses a Thread Control Buffer 420 which is read as the 'arbiter circuit' of claim 1."

Do you see that?

A I see that.

Q So thread control buffer 420 appears in Figure 4 of Lindholm, right?

A It does.

Q And that thread control buffer 420 is what you're saying corresponds to the claimed arbiter circuit, correct?

A Correct.

Q Is there anything else in Lindholm that you're pointing to or that you opine that corresponds to the claimed arbiter circuit?

A That's just one example.

Q The thread -- what's just one example?

A TCB is an arbiter.

Q TCB 420, you're saying, is an arbiter?

Page 111

A Right.

Q Right. Okay. My question is -- so you say that corresponds to the claimed arbiter circuit of claim 1?

A Yes.

Q Is there any other structure in Lindholm that you're relying on to meet the claimed arbiter circuit?

A My position is that's the arbiter that I'm using to discuss claim 1 in my charts. I think that's what I've said. I have not --

Q Okay.

A -- opined on anything else.

Q Okay. All right. Earlier I asked you, is it your opinion that execution unit 470, which appears in Figure 4, corresponds to the claimed shader, and you answered:

"Including all the components that goes with it in terms of the instruction, the sequencers and so on."

A Right.

Q So you -- is it your opinion that anything in Figure 4 that relates to instructions, corresponds to the claimed shader of the 871 patent?

Page 112

MR. PLUTA: Object to form.

THE WITNESS: I have it -- the sequencer, which is identified in Figure 5, it's also discussed here to be the instruction scheduler and dispatcher and the -- and the components related to that. Then the register file and the execution unit. So I've already discussed that with you earlier. I don't know what other component you're referring to.

The sequencer is the one that sequences through the -- through the steps, and that's what -- what instruction scheduler and dispatcher do.

BY MR. TUMINARO:

Q I'm still very confused. I apologize.

In claim 1 of the 871 patent, there's claimed a shader, correct?

A Yes.

Q Okay. In paragraph 90 of your declaration, you say:

"Components of the Execution Pipeline 240" --

Which you're referring to Figure 4 of Lindholm. You say that certain:

"Components of Execution Pipeline 240 correspond to 'shader' of claim 1."

Page 113

Correct? You said that in paragraph 90?

A I said register file, the processing unit, which is the execution unit, and the sequencer, which in this case would be sequencing the execution unit.

Q What are you saying is the sequencer in Figure 4 of Lindholm? I don't see those words.

A Instruction scheduler, instruction dispatcher.

Q Anything else?

A That's all I said.

MR. TUMINARO: Okay.

Could we go off the record.

THE VIDEOGRAPHER: We are off the record. The time is 1:04 p.m.

(Recess.)

THE VIDEOGRAPHER: We're back on the record. The time is 1:27 p.m.

Please continue.

BY MR. TUMINARO:

Q Would you turn back to Exhibit 1 for me.

A Yeah.

Q And paragraph 22, which is on page 11.

A Yeah.

Q I'll read it:

Page 114

"I have been informed that the application that issued as the 871 patent was filed in November 2003. As a result, I will assume the relevant time period for determining what one of ordinary skill in the art knew is early to mid 2003."

Do you see that?

A Yes.

Q So you didn't consider what a person of ordinary skill would have known in, for example, 2002, right?

A It would be not from that statement, but it's additive. Whatever they knew in 2002, they have gained some knowledge since then. So I -- I see it like an additive.

Q Okay. Is there anywhere in your dec- -- in your declaration where you talk about what a person of ordinary skill in the art would have known in 2002?

MR. PLUTA: Object to form.

THE WITNESS: No, but it is consistent with what I understand, they keep adding to their knowledge to that point, not all happening in that time frame. It would be gradual.

Page 115

BY MR. TUMINARO:

Q Okay. Switching gears --

A Okay.

Q -- Rich performs vertex operations and pixel operations sequentially, correct?

MR. PLUTA: Object to form.

BY MR. TUMINARO:

Q Let me ask that again.

The system disclosed in Rich performs vertex operations and pixel operations sequentially, correct?

A Let me check. Is there a specific section you want me to refer to, or is this a general question? Do you have a reference so I can look at it?

Q All right. Look at column 9.

A Okay. I don't see the word "sequentially executed."

Q All right. Let me help you out, then.

A Yeah.

Q Look at the first sentence in column 9. It reads:

"The first function carried out by one particular embodiment of the image generation system is

Page 116

illustrated in Figure 3."

Do you see that?

A Yeah.

Q Okay. And if you turn to Figure 5, the third line -- are you there? I mean column 5, the third line. Excuse me.

A Uh-huh.

Q Column 5 --

A Yes.

Q -- the third line reads:

"Figure 3 is a flow chart of geometry processing aspects of the image generation system of Figure 1."

A Okay.

Q So those two sentences in combination, if you read line 1 of column 9 and the fact that Figure 3 relates to geometry processing, that means the first function carried out by the system is geometry processing as illustrated in Figure 3, correct?

MR. PLUTA: Object to form.

THE WITNESS: It says it's geometry processing, but it doesn't say that it -- it is limited to doing that sequentially with respect to pixel. I don't get that --

Page 117

MR. TUMINARO: Okay.

THE WITNESS: -- information out of that.

BY MR. TUMINARO:

Q Okay. Then let's look at line 40 in column 9. It says:

"After geometry processing, the next function carried out by the image generation system is rasterization."

Do you see that?

A Yes.

Q And rasterization is a pixel operation?

A Yes, but --

Q So pixel operations occur after geometry processing?

A You have to have the primitives before you can do rasterization, but -- it says -- first of all, in one embodiment, I think it said that as much. There is no limitation from what I see here whereby the control unit can activate primitives being executed into the corresponding rasterization while some other primitive's being evaluated, because you have multiple primitives. So it is not limiting that because you have multiple processing units, and the control unit can control these

Page 118

independent of each other.

Q I'm asking about what's actually disclosed in Rich. And in column 9, it says first you do geometry operations, and after the geometry operations, you do pixel operations; is that right?

A In one particular embodiment, but it doesn't say that's the only way to do it.

Q Point to me an embodiment where it says they're not done sequentially.

A It didn't need to, because it was obvious to somebody who was looking at this -- or it was clear to somebody who was looking at this that I have multiple processing elements. I have a central control unit that I can control these processing elements.

What I want to do is to able to do a linear expression evaluation on a subset of these primitives, convert them into pixels while I'm working on the primitives of some other portions of the graphics image.

So I would say it's -- in other embodiments, that could happen. I don't see any limitations here on that.

Q Okay. I'm not asking about what could happen, what -- I want to know, is there anything in

Page 119

disclo- -- disclosed in Rich --

A The block diagram.

Q -- that says that you do geometry operations and pixel operations not sequentially?

MR. PLUTA: Object to form.

THE WITNESS: I think the block diagram gives you that -- that information. But if you want to see it written and happening concurrently, it doesn't say anything about sequentiality either. So I don't have any specific sentence to saying it's happening sequentially or otherwise concurrently.

But -- but Figure 2 in this case is indication to me that the -- the operation could happen in a partition way, namely, portions of the primitives could be handling -- could be working on rasterization while the other parts are doing vertex operations. So I -- I would say that is quite possible with this design, although it didn't say it specifically.

BY MR. TUMINARO:

Q Okay. And given your knowledge of graphics processing generally, you're aware that there were no graphics processors --

A Uh-huh.

Q -- in 2003 or 2002 that had a unified shader,

Page 120

correct?

MR. PLUTA: Object to form.

THE WITNESS: I haven't checked all the designs at that time frame. There are hundreds of patents and disclosures and so on. I couldn't tell you that for a fact, but it is possible that they were there.

BY MR. TUMINARO:

Q Well, you're aware of the fact that ATI's Xenos processor is recognized as -- in the industry as the first to provide a graphics processor with a unified shader, correct?

MR. PLUTA: Object to form, lack of foundation --

THE WITNESS: I don't have any --

MR. PLUTA: -- relevance.

THE WITNESS: -- indication of that. But I can assure you in conferences and journals, which they do the foundation work that ends up in these products, there has been reports of unified shaders prior to that.

BY MR. TUMINARO:

Q And you didn't cite any of those reports anywhere in your declaration that talked about a unified shader before 2003, correct?

Page 121

A I didn't need to. I was convinced with what I had.

Q If you turn back with me to the institution decision --

A Yeah.

Q -- in the 326 case.

A Yes.

Q On page 22, the second paragraph, the third sentence of the second paragraph, it starts with "Nevertheless."

Do you see that?

A Yes.

Q I'll read it:

"Nevertheless, as discussed above, both Petitioner and Patent Owner recognize that Rich discloses performing vertex operations and pixel operations sequentially."

Do you see that?

A Yeah.

Based on what it --

MR. PLUTA: There's no question pending.

THE WITNESS: Oh, sorry.

BY MR. TUMINARO:

Q So during vertex operations, the system

Page 122

disclosed in Rich will be performing operations on vertex data, correct?

MR. PLUTA: Object to the form of the question, lacks foundation. I also object to the relevance because, Counsel, you're reading from a ground that was not instituted by the board in this decision.

THE WITNESS: Okay. Thank you.

You need an answer?

BY MR. TUMINARO:

Q I do.

A I think I gave you my answer regarding what Rich can do and cannot do. From understanding of the technology, my position is that one could implement that in parallel because it has all the resources as defined in the description.

MR. TUMINARO: All right. I have no more questions at this time.

THE WITNESS: Okay.

MR. PLUTA: I have a couple questions. Should I -- I'll just --

THE WITNESS: Okay.

MR. PLUTA: It's a little awkward for the video, but...

EXAMINATION

Page 123

BY MR. PLUTA:

Q Do you remember the questions regarding your CV earlier about graphics processing?

A I do.

Q And you remember -- you listed quite a few publications or journals that counsel did not include that were on your CV related to graphics processing, correct?

A Correct.

MR. TUMINARO: Objection; leading.

THE WITNESS: Correct.

BY MR. PLUTA:

Q What's your experience with graphics processing outside of just published papers or conference papers?

A The work I did at Morpho Technologies developing the SIMD architecture for pixel processing.

THE REPORTER: "Developing" what?

THE WITNESS: The SIMD processor for pixel processing.

There are too many processing.

BY MR. PLUTA:

Q You remember you were -- you remember you were asked some questions about the terminologies of

Page 124

ALU and microprocessors?

A Yes.

Q Is it your testimony that sometimes the terminology of microprocessor and ALU is used interchangeably?

A Sometimes, yeah.

Q Going to the Rich reference, is it your opinion that vertex data and pixel data in Rich can be stored on external memory?

A It is, yeah.

Q Is it your opinion that vertex data and pixel data in Rich may also be stored in an internal memory?

A That's right.

MR. PLUTA: I don't have any ques- -- further questions at this time.

MR. TUMINARO: I have no more.

THE VIDEOGRAPHER: We are off the record. The time is 1:42 p.m., and this concludes today's testimony given by Dr. Nader Bagherzadeh. The total number of media used was two and will be retained by Veritext Legal Solutions.

(TIME NOTED: 1:42 p.m.)

Page 125

I, the undersigned, a Certified Shorthand Reporter of the State of California, do hereby certify:

That the foregoing proceedings were taken before me at the time and place herein set forth; that any witnesses in the foregoing proceedings, prior to testifying, were placed under oath; that a verbatim record of the proceedings was made by me using machine shorthand which was thereafter transcribed under my direction; further, that the foregoing is an accurate transcription thereof.

I further certify that I am neither financially interested in the action nor a relative or employee of any attorney of any of the parties.

IN WITNESS WHEREOF, I have this date subscribed my name.

Dated: September, 18 2015

Nadia Newhart

NADIA NEWHART
CSR No. 8714

Page 126

LG Electronics Inc v. ATI Technologies ULC
Dr Nader Bagherzadeh

INSTRUCTIONS TO THE WITNESS

Please read your deposition over carefully and make any necessary corrections. You should state the reason in the appropriate space on the errata sheet for any corrections that are made.

After doing so, please sign the errata sheet and date it.

You are signing same subject to the changes you have noted on the errata sheet, which will be attached to your deposition.

It is imperative that you return the original errata sheet to the deposing attorney within thirty (30) days of receipt of the deposition transcript by you. If you fail to do so, the deposition transcript may be deemed to be accurate and may be used in court.

2134835

Page 127

LG Electronics Inc v. ATI Technologies ULC
Dr Nader Bagherzadeh

E R R A T A
- - - - -

PAGE LINE CHANGE
___ ___ _________________
Reason:_______________________________________
___ ___ _________________
Reason:_______________________________________
___ ___ _________________
Reason:_______________________________________
___ ___ _________________
Reason:_______________________________________
___ ___ _________________
Reason:_______________________________________
___ ___ _________________
Reason:_______________________________________
___ ___ _________________
Reason:_______________________________________
___ ___ _________________
Reason:_______________________________________
___ ___ _________________

2134835

Page 128

LG Electronics Inc v. ATI Technologies ULC
Dr Nader Bagherzadeh

ACKNOWLEDGMENT OF DEPONENT

I, _______________________, do hereby certify that I have read the foregoing pages and that the same is a correct transcription of the answers given by me to the questions therein propounded, except for the corrections or changes in form or substance, if any, noted in the attached Errata Sheet.

___________ __________________________
DATE SIGNATURE

Subscribed and sworn to before me this
_____________ day of _______________, 20__.

My commission expires: __________________
____
Notary Public

2134835

Federal Rules of Civil Procedure

Rule 30

(e) Review By the Witness; Changes.

(1) Review; Statement of Changes. On request by the deponent or a party before the deposition is completed, the deponent must be allowed 30 days after being notified by the officer that the transcript or recording is available in which:

(A) to review the transcript or recording; and

(B) if there are changes in form or substance, to sign a statement listing the changes and the reasons for making them.

(2) Changes Indicated in the Officer's Certificate. The officer must note in the certificate prescribed by Rule 30(f)(1) whether a review was requested and, if so, must attach any changes the deponent makes during the 30-day period.

DISCLAIMER: THE FOREGOING FEDERAL PROCEDURE RULES ARE PROVIDED FOR INFORMATIONAL PURPOSES ONLY. THE ABOVE RULES ARE CURRENT AS OF SEPTEMBER 1, 2014. PLEASE REFER TO THE APPLICABLE FEDERAL RULES OF CIVIL PROCEDURE FOR UP-TO-DATE INFORMATION.

Page 1

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

LG ELECTRONICS, INC.)
)
Petitioner,)
)
vs.) No. Ipr 2015-00325
)
ATI TECHNOLOGIES ULC,)
)
Respondent.)

The deposition of NADER BAGHERZADEH, taken before JO ANN LOSOYA, C.S.R., pursuant to the provisions of the Illinois Code of Civil Procedure and the Rules of the Supreme Court thereof pertaining to the taking of depositions for the purpose of discovery at 71 South Wacker Drive, Chicago, Illinois commencing at 9:05 a.m. on August 14, 2015.

ATI 2074
LG v. ATI
IPR2015-00326

Page 2

PRESENT:
MAYER BROWN ROWE & MAW
MR. ROBERT G. PLUTA
MR. JOHN X. ZHU
71 South Wacker Drive
Chicago, Illinois 60606-4637
(312) 701-8641
rpluta@mayerbrown.com
rzhu@mayerbrown.com
Appeared on behalf of the Petitioner;

STERNE KESSLER GOLDSTEIN FOX
MR. JONATHAN TUMINARO, Ph.D.
MR. ZHU HE
1100 New York Avenue NW
Washington, DC 20005
(202) 371-2600
jtuminar@skgf.com
zhe@skgf.com
Appeared on behalf of the Respondent.

ROBIN KAPLAN
MR. BRYAN J. MECHELL
800 LaSalle Avenue
Suite 2800
Minneapolis, Minnesota 55402
(612) 349-0172
bmechell@robinskaplan.com
Appeared on behalf of the Respondent.

ALSO PRESENT:
Mary Ann Naas, Videographer.

REPORTED BY: JO ANN LOSOYA
LICENSE #: 084-002437

Page 3

Page 4

EXAMINATION

Witness	Page	Line
NADER BAGHERZADEH		
By Mr. Tuminaro	6	10
By Mr. Pluta	154	10
By Mr. Tuminaro	158	20

****

EXHIBITS

Deposition Exhibit	Page	Line
Exhibit No. 1................	8	17
Exhibit No. 2................	14	12
Exhibit No. 3................	18	14
Exhibit No. 4................	36	12
Exhibit No. 5................	37	1
Exhibit No. 6................	40	15
Exhibit No. 7................	69	24
Exhibit No. 8................	93	12
Exhibit No. 9................	113	17
Exhibit No. 10..............	148	8

Page 5

THE VIDEOGRAPHER: We are on record. My name is Mary Ann Naas representing Veritext. Today's date is 8-14-2015. The time is approximately 9:07 a.m.

This deposition is being held at Mayer Brown located at 71 South Wacker Drive, Chicago, Illinois, and is taken by the patent owner. The caption of this case is LG Electronics, Inc., versus ATI Technologies ULC. This case is being held before the Patent and Trial Appeal Board, case number IPR 2015-00325. The name of the witness is Nader Bagherzadeh.

At this time, will the attorneys please announce their presence on record.

MR. TUMINARO: Jonathan Tuminaro from the law firm Sterne, Kessler, Goldstein, and Fox on behalf of the patent owner, ATI Technologies ULC. And with me is Zhu He, also from Sterne Kessler.

MR. MECHELL: Also Bryan Mechell from Robins Kaplan on behalf of the patent owner, ATI Technologies ULC.

MR. PLUTA: Robert Pluta from Mayer Brown on behalf of LG Electronics.

MR. ZHU: John Zhu from Mayer Brown also

Page 6

on behalf of LG Electronics.

THE VIDEOGRAPHER: Our court reporter, JoAnn Losoya, representing Veritext, will now swear in the witness.

WHEREUPON:

NADER BAGHERZADEH,

called as a witness herein, having been first duly sworn, was examined and testified as follows:

EXAMINATION

BY MR. TUMINARO:

Q. Good morning. Please state your name, sir.

A. Nader Bagherzadeh.

Q. And please state your home address?

A. My address is 415 Hilledge in Laguna Beach, California.

Q. Your work address?

A. University of California, Irvine, Irvine, California.

Q. Sir, I understand that you have been deposed several times before; is that right?

A. That's correct.

Q. Just a couple of ground rules for this deposition. As you know, we're trying to get a

Page 7

clear record. So I'll ask that you don't speak over me, and I'll try not to speak over you. Is that fair?

A. That is fair.

Q. Okay. I'm going to try to ask clear questions; but if at any point you don't understand my question, will you let me know?

A. Sure. Can I ask you to slow down a little bit?

Q. Oh, sure. Sure. Okay.

If you answer one of my questions, I'm going to assume that you understood it. Is that fair?

A. Yes. If I answer, that means I understood it, correct.

Q. Okay. I'm going to take periodic breaks, but if at any time you need a break, would you let me know?

A. Absolutely.

Q. Okay. But if there's a pending question, I'd ask that you answer the question before we take a break. Is that fair?

A. Sure.

Q. You understand that you're testifying

Page 8

under oath here this morning?

A. Absolutely.

Q. And all day today actually.

A. Absolutely.

Q. Is there any reason you cannot do that?

A. No, I cannot see a reason why I could not do that.

Q. Okay. Do you know why you're here today?

A. Yes.

Q. Why?

A. There's a case pending.

Q. And you submitted a declaration in that case?

A. Yes. If you have it, I can show it to you.

(WHEREUPON, document marked as Exhibit No. 1.)

BY MR. TUMINARO:

Q. Sir, you've been handed what has been marked as Exhibit 1. This is titled Declaration of Dr. Nader Bagherzadeh. Is this your declaration?

A. Yes, sir.

Q. If you turn to the last page of the declaration before the appendix, there's a signature

Page 9

there?

A. Yes.

Q. Is that your signature?

A. Yes.

Q. And you signed this declaration on December 9, 2014?

A. Yes.

Q. Is there anything that you'd like to add to this declaration?

A. No.

MR. PLUTA: Counsel, did you -- I thought I heard you refer to this as Exhibit 1. Are you referring to it as a different exhibit number?

MR. TUMINARO: Just Exhibit 1 to this deposition.

MR. PLUTA: Okay. So you are not going to refer to it as Exhibit 1003.

MR. TUMINARO: I may during the course of it. I just wanted to mark this so that it is clear this is Exhibit 1 to Dr. Bagherzadeh's deposition.

MR. PLUTA: Fair enough.

BY MR. TUMINARO:

Q. Is there anything that you would like to delete from your declaration?

Page 10

A. No, I don't think so.

Q. Are there any changes that you'd like to make to your declaration at all?

A. No.

Q. If you would turn with me to Paragraph 2 of your declaration.

A. Okay.

Q. Paragraph 2 reads: "I have been asked to provide my opinions and views on materials I have reviewed in this case related to U.S. Patent No. 7742053 (the '053 patent)."

Do you see that?

A. Yes.

Q. What materials are you referring to that you reviewed?

A. I think we have a list of that. If I refer you to Page 87 of Exhibit 1, you can see the list.

Q. That's titled Materials Considered and Exhibit List, correct?

A. That is correct.

Q. Does this list all the materials that you reviewed in preparing your declaration?

A. Yes.

Page 11

Q. In fact, there are no other materials that you reviewed in preparation for your declaration?

A. If they were, I would have put them in here.

Q. Okay. If you would turn with me to Page 10 of your declaration.

A. Yes.

Q. At the top of Page 10, there's a figure or a picture.

Do you see that?

A. Hm-hmm.

Q. You didn't generate this picture, did you?

A. It was probably from one of my lectures.

Q. One of the lectures that you prepared?

A. Yeah.

Q. But that's not listed on your materials considered, is it?

A. It's one of my lectures, yeah.

Q. So. One of your lectures is one of the things that you considered in preparing your declaration?

A. My knowledge is what was considered for

Page 12

this.

Q. But, in fact, a figure from one of those lectures is actually in your declaration, right?

A. It could be. I couldn't say for certain.

Q. Let's look at the next page, Page 11. In Paragraph 31, there's also a figure. Where did this figure come from?

A. From one of my lectures.

Q. Your lectures aren't listed in the materials considered, right?

A. It's from my knowledge that I have.

Q. Well, I appreciate the answer, but I think it didn't answer my question. This lecture is not listed on your materials considered, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. It's what I know. So the lecture is not cited, that's right, but it's what I know. I have a lot of other knowledge that is not cited here about computer architecture, the textbooks that I've used over the years, my papers I've written over the years. No, they're not cited here, but they're up in my head.

Page 13

BY MR. TUMINARO:

Q. Is there anything else that you relied upon in preparing your declaration that you did not list in your exhibits considered?

A. I cannot think of anything right now unless you point it out.

Q. The figure in Paragraph 31, do you know what lecture this refers to?

A. Computer architecture.

Q. Is that publicly available?

A. No.

MR. TUMINARO: Counsel, I'd guess I'd state for the record that we want a copy of that lecture. It's not listed on the materials considered, and it appears to be something that was used in generating this declaration.

MR. PLUTA: Other than the figure and what he just testified, I don't see how the relevance of the lecture in its entirety would be relevant to this proceeding.

BY MR. TUMINARO:

Q. Did you write your declaration?

A. Yes, sir.

Q. How much time did you spend in preparing

Page 14

your declaration?

A. Over 40 hours.

Q. Over 40 hours. Was it over 50 hours?

A. I can't tell you.

Q. When were you first contacted with respect to this matter?

A. I don't remember.

Q. Who contacted you?

A. Oh, that's a -- it's an agency that ties the experts to the lawyers.

(WHEREUPON, document marked as Exhibit No. 2.)

BY MR. TUMINARO:

Q. You have been handed what's been marked as Exhibit 2 for identification purposes. The title is Consultant Curriculum Vitae, Nader Bagherzadeh, Ph.D.

Do you see that?

A. Yes.

Q. You have seen this document before?

A. Yes.

Q. This is your current CV?

A. Yes.

Q. Just to be clear, is this just your

Page 15

consultant CV? It says consultant CV.

A. It says so, yes.

Q. Do you have a different CV?

A. I have different CVs, yeah.

Q. This is the one that you use for patent matters?

A. It seems that way. I mean, it's here so...

Q. And anything you want to add to this CV?

A. No.

Q. Anything you want to delete?

A. No.

Q. Any changes at all?

A. No.

Q. Okay. You worked at AT&T Bell Labs, right?

A. Yes, sir.

Q. While you were there, did that work involve any sort of work on microchips?

A. It did.

Q. What did you do?

A. Used them.

Q. You used them. Did you make any microchips at Bell Labs?

Page 16

A. No. We designed boards.

Q. And you left Bell Labs in 1984?

A. Yes, sir.

Q. So for the past 30 years, you have been in academia, right?

A. Not accurate.

Q. What's not accurate about my statement?

A. Because I went to school.

Q. Oh so -- I see. In 1987, you started in academia; is that right?

A. That's right.

Q. Since 1987, you have been in academia, right?

A. Correct, with a little period of running a start-up as you may have seen it in my resume.

Q. You have never actually built a computer chip, have you?

A. I did, yes.

MR. PLUTA: Object to form.

BY MR. TUMINARO:

Q. You did build a computer chip.

A. Absolutely.

Q. What did you do then?

A. I designed it.

Page 17

Q. You designed it?

A. Yes.

Q. You're telling me that just designing a chip is building it?

A. Excuse me.

Q. Are you telling me that just designing a chip is building a chip?

A. You didn't ask me if I fabricated it. You asked me if I designed it, and I said I designed it. Designing means to design something. Then you have to fabricate it.

Q. Have you built a chip?

A. Fabricate it, is that what you are asking me?

Q. Do you know what the word "build" means?

A. Do you know what the fabrication means?

Q. I do. Do you know what the word "build" means?

A. Yes. We don't use building. We use the word fabrication. We fabricated the chip, yes.

Q. You fabricated the chip, and you designed the chip?

A. You just asked me, and I said yes.

Q. Did you ever design a GPU?

Page 18

MR. PLUTA: Object to form.

BY THE WITNESS:

A. We designed pieces of the GPU for our chip.

BY MR. TUMINARO:

Q. Which chip are you referring to?

A. Multiple chips.

Q. Did you build a chip called morphosys?

MR. PLUTA: Object to form. Object to relevance.

BY THE WITNESS:

A. Yes, I did.

(WHEREUPON, document marked as Exhibit No. 3.)

BY MR. TUMINARO:

Q. Sir, you have been handed what has been marked as Exhibit 3 for identification purposes. Do you recognize this document?

A. Yes.

Q. What is it?

A. It's a paper I have written with my colleagues and former students.

MR. PLUTA: I'm going to object to the relevance of this exhibit at this time.

Page 19

BY MR. TUMINARO:

Q. If you turn with me to I guess what is labeled as Page 10-3. Well, first of all -- first of all, if you will turn to your CV again, Exhibit 2, on Page 19 of your CV.

A. Yeah.

Q. Are you there, sir, Page 19?

A. Yes.

Q. Okay. If you look at C57, there's a paper there that's listed, and the title of the paper is: "Morphosys: An Integrated Reconfigurable Architecture."

Do you see that?

A. Yes.

Q. If you look at the title of Exhibit 3, it's the same, right?

A. Yes.

Q. Exhibit 3 would be an example of some of the stuff that's in your head as you said earlier, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. It's an example of what's in my head?

Page 20

BY MR. TUMINARO:

Q. Your knowledge about --

A. Knowledge, yes.

Q. So Exhibit 3 is an example of your knowledge?

A. That's right. That's better. Yes, thank you.

Q. If you will turn with me in Exhibit 3 to Page 10-3.

A. Hm-hmm.

Q. And on the left-hand column, about midway through the page, there's a heading four, Morphosys, System Model.

Do you see that?

A. Yes.

Q. And the last sentence, it says: "The system model and architecture details for the first implementation of morphosys (M1 chip) are described hereafter."

Do you see that?

A. Yes, I see that.

Q. Was this morphosys actually a chip?

A. Yes, sir.

Q. So the architectural details were

Page 21

sufficient to be an implementation of a chip?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. So your question is by looking at this paper, do I prove that I fabricated the chip or not?

BY MR. TUMINARO:

Q. That was not my question.

A. But I'm getting that from your question. I'm repeating it because I didn't understand what you said.

Q. Then I'll clarify.

A. Right. Please.

Q. In this sentence in your paper, it says: "The system model and architectural details for the first implementation of the morphosys chip are described hereafter." Right?

A. Right.

Q. So, you had an implementation?

A. I had an implementation. That's right.

Q. That implementation was based on architectural details of the chip?

A. That is described in this document, yes.

Q. What were the architectural details that you had made at that time?

Page 22

MR. PLUTA: Object to form. Object to relevance.

BY THE WITNESS:

A. It's in the paper.

BY MR. TUMINARO:

Q. Let's look at what the paper says.

A. I'll be glad to go over it with you.

Q. If you will turn with me to Page 10-10. On the right-hand side, there's a number 7, Interactive Software Environment.

Do you see that?

A. 10-10. Which column?

Q. On the right-hand side.

A. Yes.

Q. The first sentence after Interactive Software Environment, it reads: "The morphosys reconfigurable system has been specified in behavioral VHDL."

Do you see that?

A. I do.

Q. What is VHDL?

A. It stands for a hardware description language.

Q. Verilog Hardware Description Language?

Page 23

A. It says VHDL. It doesn't say verilog.

Q. Does VHDL stand for Verilog Hardware Description Language?

A. No, it does not. I don't think it does.

Q. What does the V stand for, then?

A. I don't know. It could be.

Q. Having read that sentence, turning back to the sentence on Page 3 that talked about architectural details, were the architectural details that are described on Page 3 the VHDL code that you were talking about?

MR. PLUTA: Object to form. Object to relevance.

BY THE WITNESS:

A. Well, the statement there is accurate. It says the morphosys reconfigurable system has been specified in behavior VHDL.

BY MR. TUMINARO:

Q. You didn't answer my question. On Page 3, you said the system -- the system model and architectural details for the implementation of morphosys (M1 chip) are described hereafter, right?

A. You read that. Yes.

Q. I asked you earlier about what

Page 24

architectural details are, correct?

A. Hm-hmm.

Q. You said it's described in the paper, right?

A. Yes.

Q. And I went to Page 10 and the architectural details. Now, my question is are the architectural details the VHDL that's described on Page 10 of your paper?

MR. PLUTA: Object to form. Object to relevance.

BY THE WITNESS:

A. Sir, if I read that sentence --

MR. PLUTA: Object to foundation.

BY THE WITNESS:

A. If I read that sentence, it should give you the information. I'll read it again. It says -- let me finish the sentence, please.

The morphosys reconfigurable system has been specified in behavior VHDL. Which morphosys am I talking about? It's the one that that is discussed in this paper. So the relevance is obvious. You have to deduce it from this statement. If you are not getting that information,

Page 25

maybe we didn't write it properly.

BY MR. TUMINARO:

Q. That's what I'm asking you. I could read the paper. I'm asking you are the architectural details the VHDL?

MR. PLUTA: Object to form. Object to relevance. Object to foundation.

BY THE WITNESS:

A. I repeat my -- at the beginning, we talk about architectural details. This is referring to the morphosys reconfigurable system. Which other architecture are we talking about if it's not what I just said in Page 2 that you had? Can you tell me that I could be talking about something else? I just don't see why it's so puzzling to you.

BY MR. TUMINARO:

Q. Is the answer to my question, yes, then, that the architectural details are the VHDL?

MR. PLUTA: Object to form. Object to relevance.

BY THE WITNESS:

A. It says morphosys reconfigurable system has been specified in VHDL. It could not be any other morphosys circuit.

Page 26

BY MR. TUMINARO:

Q. So that means that the VHDL was an implementation of that morphosys chip?

MR. PLUTA: Object to form. Object to relevance. Asked and answered.

BY THE WITNESS:

A. Yes. That's what it says. Of course.

BY MR. TUMINARO:

Q. If you turn back with me to your declaration at Paragraph 38, I'll read it for the record. Are you there?

A. Yeah.

Q. Paragraph 38 reads: "I believe that a person of ordinary skill in the art relating to the '053 patent would be someone with a good working knowledge of computer graphic processing architecture as well as the systems and programs that support such architecture."

Do you see that?

A. Yes.

Q. How did you come up with that definition?

A. I just wrote it.

Q. Based on what?

A. Based on my knowledge.

Page 27

Q. Do you satisfy that definition?

A. It's in the report. So I'm happy with it.

Q. No, no. That wasn't my question.

Are you a person of ordinary skill in the art with respect to your own definition?

A. Sorry about that. I took it as satisfied differently.

Yes, I am.

Q. Were you a person of ordinary skill in the art as of the time that the '053 patent was filed?

A. Yes, I was.

Q. Have you ever designed a GPU?

A. I designed some of the architectures that we have that were used for computer graphics, yes.

Q. Architectures other than the morphosys?

A. Yes. I have done several architectures, but morphosys was used for graphics.

Q. That had graphics in it?

A. Yes.

Q. You never worked in a graphics company, did you?

MR. PLUTA: Object to form.

Page 28

BY THE WITNESS:

A. My resume speaks for itself. I never did.

BY MR. TUMINARO:

Q. You never programmed a 3D graphics on a GPU, did you?

A. I taught a course, and I did some programming in open GL for the course to prepare the exams, the homework, and so on. Yeah.

Q. Have you ever used a -- strike that. You mentioned the open GL. Have you heard of DX10, 11?

A. Yes. I have heard of those.

Q. Have you ever used those?

A. Not specifically, but I have -- I am supervising projects or supervised projects, we tried to use GPUs for high performance computing and we're still doing that.

Q. By high performance computing, you don't mean 3D graphics, do you?

A. Well, you're using the GPUs, but you are doing it for number crunching.

Q. Right.

A. It's called GPGPUs.

Q. It's general purpose GPU, right?

Page 29

A. Yes. But we're using all the resources on the GPU. All the resources.

Q. Okay. My question is you are not using the GPU to do 3D graphics processing, right?

A. Not now.

Q. Have you ever?

A. Before we did, yes.

Q. If you look back with me, I guess, at Paragraph 31, there's that figure that you put in from your lecture.

A. Yes.

Q. That figure is relating to a multi-threaded CPU, right?

A. It's related to multi-threading in general. And you could think about it as a CPU. You could think about it as a GPU. You can think about it as any computation form. It's not target application specific.

Q. If you look at the previous page, Page 10, that figure is for multi-threading on a CPU, right?

A. It says CPU, right. I couldn't tell you that's a GPU. It says CPU.

Q. The '053 patent is about a GPU, not a

Page 30

CPU?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. '053 is about multi-threading for GPUs. So this is to give you information about the background of multi-threading. It's not talking about -- I'm not trying to address GPUs here. I'm talking about what is multi-threading for the layman person.

BY MR. TUMINARO:

Q. Let's go back to your materials considered.

A. Yes, sir.

Q. Just to be clear, your lectures aren't shown in the materials considered, right?

MR. PLUTA: Object to form. Asked and answered.

BY THE WITNESS:

A. I think we established that, Counsel.

BY MR. TUMINARO:

Q. Is the answer yes?

A. It's clear that it's not.

Q. Earlier you mentioned that you had designed GPUs or certain aspects of GPUs for your

Page 31

chips. Those previous designs aren't listed in your previous materials considered?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. No, sir.

BY MR. TUMINARO:

Q. Let's go back to your technical background which starts on Page 7 of your declaration. And the technical background spans from Page 7 through to the top of Page 12, correct?

A. Yes.

Q. And in that entire section, there's not a single cite to any external material, correct?

A. Let me check.

MR. PLUTA: Object to form.

BY MR. TUMINARO:

Q. Let me ask the question again.

In that entire section, you didn't cite any extrinsic evidence to support any statement that you made in this section, right?

A. Would you like me to check? I was just about to do that for you.

Q. Yeah. I wanted to clarify my question.

A. Yes. I think I understood your question.

Page 32

So I'm going to look for references to see if there are.

Q. Okay. Great.

A. If I'm misunderstanding, please let me know.

It doesn't seem to be.

Q. Where does this material come from then?

A. I think I addressed that earlier. From my knowledge and lectures, research, all of it done over the years.

Q. So the entirety of the technical background section just comes from stuff that you have in your head?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. Yes. That's why I'm here. 25 years of experience in education. If you want me to cite everything I know, then it will be a book.

BY MR. TUMINARO:

Q. If you'd turn with me to Page 8, I'd like to focus on the Bullet No. 2 that says rasterization.

A. Yes.

Q. Rasterization, that's about generating

Page 33

pixels; is that right?

A. Yep.

Q. If you'd turn with me to Page 10, the last bullet there talks about cache misses.

There's a penalty for a cache miss, right?

A. Yes.

Q. So if you increase the size of the cache, that will likely decrease cache misses; right?

A. There is a reason for that, but you cannot assume increasing the cache size will always improve your performance. There is a limit to that. This is what is called a bathtub curve.

Q. Bathtub curve, you moved your hand --

A. Just like a bathtub. You cannot keep increasing the cache size and expect improvements, but it gets better. You're right.

Q. And by -- and you said increasing the cache size will increase your performance?

A. Yeah.

Q. By increased performance, you mean decreased cache misses?

A. It will give you more room to put stuff in there so you can keep track of more instructions

Page 34

or data, and then there's something called how big is the cache block. It's all related.

Q. Maybe my question is not perfect. So, a cache miss is a penalty, right? A timing penalty?

A. Yes. You're right.

Q. If you miss in the cache, you have to go out and get whatever it is you're looking for in memory, right?

A. From the slower memory, that's right.

Q. So it's a penalty in time?

A. That is right, and penalty in power consumption.

Q. Great. So if you decrease the cache size, that will likely increase cache misses; right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. What application do you have in mind?

BY MR. TUMINARO:

Q. I'm asking you in general. If you have a smaller cache or fewer things in there, it is more likely that you are going to have a miss -- a cache miss, right?

A. Well, if your application is three lines, it will fit in a small cache. So you don't need

Page 35

that. So you have to qualify your question is for what type of application you're talking about.

Q. All right. Since this case is about GPUs, let's talk about GPUs.

A. You didn't qualify your question with that. You asked me a general question.

Q. Okay. How about in GPUs? If you decrease the cache size, it will likely increase the likelihood of a cache miss in a GPU, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. You can write two lines of GPUs, and it still would be a very good size cache. So your qualification is still not sufficient for me to answer that. But let me help you out. If you want a larger program, yes, you need a bigger cache.

BY MR. TUMINARO:

Q. A larger program in a typical type GPU that you would use in a commercial implementation, decreasing the cache size will increase cache misses?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. Yes. You asked me questions that are

Page 36

related to general topics.

MR. TUMINARO: I could keep going, but I'm going to switch into something new. So do you want to take a break now or do you want me to keep going? It is up to you, sir.

THE WITNESS: I'm okay. If I get tired, I'll let you know if it's not in the middle of a question like you said.

MR. TUMINARO: Okay. Great.

(WHEREUPON, document marked as
Exhibit No. 4.)

BY MR. TUMINARO:

Q. You have been handed what has been marked as Exhibit 4 for identifications purposes. This is U.S. Patent No. 7233335.

Have you seen this document before?

A. Yes.

Q. This is the Morton patent, correct?

A. Correct.

Q. And this is listed on your materials considered, correct?

A. Correct.

(WHEREUPON, document marked as

Page 37

Exhibit No. 5.)

BY MR. TUMINARO:

Q. Sir, you have been handed what has been marked as Exhibit 5 for identification purposes. This is U.S. Patent No. 7742053. This is the '053 patent that we have been talking about.

A. Correct.

Q. It's listed on your materials considered?

A. Correct.

Q. If you would turn with me to the last column, column A in claim No. 5. Are you there, sir?

A. Yeah.

Q. The last element in claim No. 5 is a plurality of command processing engines.

Do you see that?

A. Yes.

Q. And I'll read it for the record. "A plurality of command processing engines, coupled to the arbiter, each operable to receive and process the command thread."

Do you see that?

A. Yes.

Q. And in the previous element, it cites an

Page 38

arbiter.

Do you see that?

A. Yes.

Q. I'll read that one. "An arbiter, coupled to be at least one memory device, operable to select a command thread from either of the plurality of pixel command threads and the plurality of vertex command threads."

Do you see that?

A. Yes.

Q. So looking at the plurality of command processing engines, each of those has to be able to process a vertex command thread or a pixel command thread, right?

MR. PLUTA: Object to form.

THE WITNESS: You are asking a yes or no question?

BY MR. TUMINARO:

Q. Yes.

A. No.

Q. It does not?

A. No.

Q. When it says in claim 5, "each operable to receive and process the command thread," that's

Page 39

not referring to vertex command threads or pixel command threads?

A. It doesn't say that. It just says receive and process. I don't get the --

Q. It doesn't say "the command thread"?

A. Yeah, but it doesn't say that it should be able to do both. I don't see that here.

Q. What do you understand the command thread to be then?

A. It's -- it doesn't say what you're saying.

Q. You don't understand it to be a pixel command thread or a vertex command thread?

A. Not to do both. There will be some that would do pixel commands. The other ones would be to thread commands.

Counsel, do you want to take a break?

MR. TUMINARO: Sure. Now is a good time to take a break.

THE VIDEOGRAPHER: We're going off record. The time is 9:55 a.m.

(Whereupon, a break in the proceedings was taken.)

THE VIDEOGRAPHER: We're back on record.

Page 40

The time is 10:09 a.m.

BY MR. TUMINARO:

Q. If you would turn -- Welcome back, Dr. Bagherzadeh. Did you talk with counsel about the substance of your testimony during the break?

A. No.

Q. If you'd turn with me to Exhibit 1, your declaration again. In particular, I would like to go to Page 72. I'm sorry. Paragraph 72, this is referring to the Lindholm reference, is that right, starting on Page 20?

A. Correct.

(WHEREUPON, document marked as Exhibit No. 6.)

BY MR. TUMINARO:

Q. You have been handed what's been marked as Exhibit 6 for identification purposes. This is U.S. Patent No. 7015913.

A. Yes.

Q. This is the Lindholm patent?

A. Correct.

Q. This is listed on your materials considered.

Page 41

A. Correct.

Q. Now, Paragraph 72 of your declaration says and I'll read it for the record: "Because thread control buffer 420 stores information for both pixel and vertex threads, they are the claimed memory device."

Do you see that?

A. I see that.

Q. So, Paragraph 72 of your declaration is referring to -- just to be clear, the thread controlled buffer 420, which is illustrated, for example, in figure 4 of Lindholm; is that right?

A. Yes.

Q. If you will look with me back to Exhibit 5.

A. Yes.

Q. In claim 1, which appears on column 7, the memory device that's claimed in claim 1 reads, and I'll read it for the record: "At least one memory device comprising a first portion operative to store a plurality of pixel command threads and a second portion operative to store a plurality of vertex command threads."

Do you see that?

Page 42

A. Yes.

Q. Claim 1, the thing that's required to be stored are the pixel command threads, command threads -- strike that. Let me start over. It was all mixed up.

In claim 1, the thing that is stored by the memory device are the pixel command threads and the vertex command threads; is that right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. You just repeated that claim, right? It talks about -- is that what you did?

BY MR. TUMINARO:

Q. What I'm trying to get at is the memory device stores a plurality of pixel command threads and a plurality of vertex command threads.

MR. PLUTA: Object to form, if that's a question.

BY THE WITNESS:

A. It says a memory device comprising of a first portion operative to store a plurality of pixel command threads and a second portion operative to store a plurality of vertex command threads. I agree with what it says.

Page 43

BY MR. TUMINARO:

Q. I'm not trying to trick you. There are two things that are stored in the memory device, right?

MR. PLUTA: Object to form.

BY MR. TUMINARO:

Q. Let me try again. There are two types that are stored by the memory device in claim 1; is that right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. You're saying at most two or at least two or exactly two?

BY MR. TUMINARO:

Q. At least two.

MR. PLUTA: Still objection.

BY THE WITNESS:

A. I would say it says two here right now. I would not know if there was more or less -- I mean more.

BY MR. TUMINARO:

Q. So what is stored, what does the claim require that the memory device stores?

MR. PLUTA: Object to form.

Page 44

BY THE WITNESS:

A. It says to store a several pixel command threads and several vertex command threads.

BY MR. TUMINARO:

Q. Okay. Great. We agree that it's pixel command threads and vertex command threads, right?

A. These are instructions for the threads, yes.

Q. In Paragraph 72 of your declaration, what you say that is stored is "information" for both pixels and vertex threads; is that right?

A. You're reading my statement.

Q. I just want to be clear that you're saying information about the pixels and vertices satisfies the claimed vertex command thread and pixel command thread?

A. I'm not saying that.

Q. What are you saying then?

A. It's a combination of TCB plus the instruction cache that satisfies that particular element of claim 1.

Q. Well, in Paragraph 72 of your declaration, you're saying that the TCB, as you put it, stores information, right?

Page 45

A. Yes.

Q. The claim requires command threads, right?

A. That's what it says, yes.

Q. I just want to be clear. So you're saying that the information stored by the TCB satisfies the claimed command threads?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. I did not say that.

BY MR. TUMINARO:

Q. That's not what you're saying in Paragraph 72?

A. All I'm saying is -- I think you're mixing information with instructions.

Q. That's what I'm trying to understand. I'm trying to understand exactly what it is that you're mapping in Lindholm that corresponds to the claimed vertex command threads and pixel command threads.

A. As I think I just alluded to just a few minutes ago, it's a combination of TCB plus the instruction cache.

Q. So what is the thing that is stored in

Page 46

the TCB and instruction cache that you're saying is the claimed command threads?

A. It is the instructions plus information.

Q. Just to be clear -- I guess what I'm not clear about is what is the information that you're talking about?

A. Okay.

MR. PLUTA: Object to form.

BY MR. TUMINARO:

Q. All right. Let me clarify. No, I appreciate his objection. So let me clarify.

In Paragraph 72 of your declaration, what information is it that you're talking about?

A. Let me refer you to my report. I wish I could do a search.

Referring to my claim chart on Page 25 of my report, it's just an example of this. It says TCB or thread control buffer 420 includes storage resources to retain thread state data for a subset of predetermined number of threads and that's the data. It's the state data because you have to track this information while instructions are alive, and once they retire, you let go of that state.

Q. Just to be clear, the thread control

Page 47

buffer 420 stores this state data that you just referred to?

A. At least.

Q. At least the state data. And that state data corresponds to the information that you refer to in Paragraph 72 of your declaration?

A. Some of the information is state data, right.

Q. If you turn with me to Paragraph 73 of your declaration.

A. Okay.

Q. And the first word is "while" in that paragraph, right?

A. Yes.

Q. After that, it reads: "The thread control buffer 42 does not explicitly disclose a first portion for storing pixel command threads or a second portion for storing vertex command threads."

Do you see that?

A. I think you read it correctly.

Q. Is that a true statement? It's an accurate statement?

A. Everything I have here is true.

Q. Okay. So you would agree with me that

Page 48

Lindholm does not explicitly disclose thread -- a thread control buffer that has a first portion for storing pixel command threads and a second portion for storing vertex command threads?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. Explicitly? I say it, but a POSITA would have figured it out from the block diagram and the knowledge that it is --

BY MR. TUMINARO:

Q. Okay --

A. -- a combination.

Q. -- I'm not talking about that for a second. I'll get to that. I'll get to that. I promise you. Just for now you would agree that there is no explicit disclosure in Lindholm about a first portion that stores pixel command threads and a second portion that stores vertex command threads?

A. So taking the first part of that sentence, you are repeating it correctly, but there is more to that.

Q. So you would agree with me -- strike that.

And that storing of a first portion

Page 49

operative to store a plurality of pixel command threads and a second portion operative to store a plurality of vertex command threads, that's a requirement of claim 1 of the '053 patent, right?

You could look at the '053 patent if you need to.

A. You repeated claim 1, the first element of that claim 1, right?

Q. Right.

A. Your question is that is a requirement?

Q. Yes.

A. It's a requirement and Lindholm satisfies it.

Q. You would agree with me that Lindholm doesn't explicitly disclose it, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. I think it says in my report, but it's my opinion that it would have been obvious to one of the ordinary skill. Because if you take the sentence out of the context, just the first part, it's not really my position.

BY MR. TUMINARO:

Q. I appreciate that. I just want to be

Page 50

clear on what your position is. I'll get to obviousness. I promise. You are not saying that Lindholm anticipates claim 1, right?

A. I don't think we said that it anticipates claim 1.

Q. Okay. Okay.

A. I don't think we did, but I can check for you. In view of AAPA, so we're not saying anticipates, right. I'm not saying it's anticipating that.

Q. I just want to be clear. You agree Lindholm does not anticipate claim 1?

A. On it's own, correct.

Q. Okay. All right. But you think that it's obvious, right?

A. Yes.

Q. Okay. I told you I would get to obviousness. That's what I'm trying to get to.

MR. PLUTA: Object to form.

BY MR. TUMINARO:

Q. And the reason -- if you look at Paragraph 73, it says about -- after that first subordinate clause that I read, it says: "It is my opinion."

Page 51

Do you see that in Paragraph 73?

A. Yes.

Q. I'll read it for the record just so it's clear. "It is my opinion that it would have been obvious to one of ordinary skill in the art to modify Lindholm's multi-threaded system to store different types of command threads in separate portions of memory because it is well known that pixel threads and vertex threads are necessarily different types of data."

Do you see that?

A. Yes.

Q. You stand by that statement?

MR. PLUTA: I'm going to object, Counsel. This is outside the scope of the instituted grounds.

BY MR. TUMINARO:

Q. If you would turn with me to Paragraph 100 of your declaration.

A. Yep.

Q. And Paragraph 100, this is related to your position that this paragraph falls within the section related to ground No. 2 where you're saying that Lindholm and so-called admitted prior art render obvious certain claims of '053; is that

Page 52

right?

A. It's Lindholm in view of AAPA, that's right.

Q. And the AAPA that you're relying on, you talk about that in Paragraph 100, right?

A. Correct.

Q. You mention here in Paragraph 100, it says, "the '053 patent discloses that buffer 104 stores ALU resource command threads while buffer 106 stores texture fetch resource command threads."

Do you see that?

A. Correct.

Q. Okay. So this background material talks about ALU threads and texture threads, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. It talks about instructions belonging to the -- okay.

BY MR. TUMINARO:

Q. Let me clarify. There's a first buffer, buffer 104 for storing ALU threads, resource command threads, and a second buffer, 106, for storing texture fetch resource command threads, right?

A. Yes.

Page 53

Q. Okay. If you go back to claim 1 of the '053 patent, claim 1 is not about -- it doesn't say anything about ALU command threads or texture command threads. It talks about pixel command threads and vertex command threads, correct?

A. That's what it says. But I can qualify that if you are trying to tie it into this paragraph.

Q. I guess in paragraph -- okay. In Paragraph 101, you say: "In graphics processing, it is well known that vertex command threads belong in the category of ALU resource division because of the nature of vertex operations."

A. Correct.

Q. You provide no cite for that, correct?

A. It says well known, in my experience, teaching, research.

Q. You think it's a true statement?

A. It is.

Q. In fact, pixel commands can also be ALU operations; is that right?

A. Sometimes. Predominantly, it is vertex.

Q. You didn't qualify it like that in Paragraph 101, did you?

Page 54

A. It's obvious.

Q. But you said -- so you would agree with me then that pixel command threads could also be ALU operations, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. Predominantly, it's vertex.

BY MR. TUMINARO:

Q. I don't think you answered my question. You would agree with me that pixel command threads can also be ALU operations?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. They could, but it's not very common.

BY MR. TUMINARO:

Q. Alpha blending would be an example of which pixels commands would be associated with an ALU type operation, right?

MR. PLUTA: Object to form, lack of foundation.

BY THE WITNESS:

A. As an architect, when we evaluate systems, we look at dominance of certain functions in terms of performance, power consumption, and so

Page 55

on. A POSITA would have looked at this and would have said, yeah, vertex computations are taking a lot of computations in terms of using the ALUs. So, yeah. It would be the dominant one, 80, 70 percent, and that's -- that's what you do, part of the design.

BY MR. TUMINARO:

Q. None of that answered my question, respectfully.

A. I'm so sorry about that.

Q. Alpha blending would be an example in which -- well, strike that.

Alpha blend is a pixel type operation?

A. It is. But it's not the dominant computation in terms of -- how you design the architecture is you want to be able to address the most computation intensive functions. So, vertex will dominant here.

Q. In alpha blending, the pixel command involves multiplication, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. It includes some sort of a computation.

Page 56

Multiplication could be one of them, but it is usually not floating point. It's integer.

BY MR. TUMINARO:

Q. Multiplication is an ALU, not a texture type of operation, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. It is multiplication. It is ALU.

BY MR. TUMINARO:

Q. So, in fact, alpha blending is an example of a pixel command that involves ALU operations?

MR. PLUTA: Object to form. It is also outside the scope of the report.

BY THE WITNESS:

A. I think I have answered you to the best of my abilities. I explained to you that predominantly it is vertex operations. I don't have anything more to add to that.

BY MR. TUMINARO:

Q. In paragraph 101 you say that vertex command threads belong to the category of ALU resource division, right?

A. That's what it says.

Q. But isn't it a fact that vertex commands

Page 57

can also involve texture operations?

A. Well, again, predominantly they're computation intensive operations. So they belong to the ALU sections.

Q. But it is a fact that vertex commands can involve texture operations, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. It's a small percentage in terms of overall computation. So if you want to take a hundred to be the example, just using as an example, a hundred to be the amount of time, 80 percent perhaps would be -- 90 percent would be in the ALU portion, there will be a small portion for texture.

BY MR. TUMINARO:

Q. Just to be clear, the answer to my question is yes?

MR. PLUTA: Object to form.

BY MR. TUMINARO:

Q. Vertex commands can involve texture operations?

A. I explained to you that it's predominantly a computation of ALU.

Q. But it can be -- vertex commands can be

Page 58

texture operations?

MR. PLUTA: Object to form, asked and answered.

BY THE WITNESS:

A. I cannot add any more to what I just said. I mean, it is what it is. You can design it the way you want. But it's predominantly a computation of the ALU.

BY MR. TUMINARO:

Q. The morphosys chip that you developed --

A. We're switching topics back to morphosys? Is that what you are asking me?

Q. Yes. Could that system handle alpha blending?

MR. PLUTA: Object to form. Object to the relevance.

BY THE WITNESS:

A. Yes.

BY MR. TUMINARO:

Q. If you will turn with me to the '053 patent and claim 1.

A. Yes, sir.

Q. And the second element is an arbiter. And it reads, for the record, "an arbiter, coupled

Page 59

to be at least one memory device, operable to select a command thread from either of the plurality of pixel command threads and the plurality of vertex command threads." I'll stop there.

Do you see that?

A. I see that.

Q. So the claimed arbiter has to select from two types of things, right, vertex command threads and pixel command threads?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. The arbiter has to select a command thread from either of the plurality of pixel command threads. Yeah, that's what it says.

BY MR. TUMINARO:

Q. In order to select either a pixel or a vertex, those options both must be available to the arbiter at the same time, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. What do you mean, those options have to be available? I don't understand that.

BY MR. TUMINARO:

Q. Well, if you are going to select

Page 60

something, select one of two different things, you have to have the option to select either of those two things, right?

A. You could have a computation that only the vertex is available. I mean, it's not an -- it's not against whatever they say here. It could be just one chute, if I could use the word chute, is available, and you just pick from that one.

Q. Well, if I say to you do you want pizza or a sub for lunch, you select. You have to have both of those options available at the same time, right?

A. If the Subway is closed, I'm going to have pizza. So until the subway opens.

Q. In which case you haven't made a selection, right?

A. Sure. I made a selection, but it was not available.

Q. Right but you didn't make that the selection, right?

A. No. I had the option to going to the Subway, but it was closed. It was not available.

Q. Is it your position if an option is not available to you, you're still selecting that

Page 61

option?

A. In what context are we talking about now? The Subway or the computer architecture?

Q. Let's talk about the context of the arbiter. If the pixel is not available to the arbiter, would you say the arbiter is selecting a vertex even though a pixel is not available to it?

A. The job of the arbiter -- I have to explain to you, Counsel, if you allow me to finish. If you are not going to allow me to finish, then I won't be able to answer your question.

Q. Sir, I didn't say anything.

A. The way you shaked your face, it was like you were frustrated with me.

The job of the arbiter is to select between the available options. Is that clear what I said? And that's all it does. You gave an example of two for this particular claim. It could be three, it could be four, and depending when it is available. So it's very simple. No tricks.

Q. In claim 1, the available options are pixel command threads and vertex command threads?

A. If they're available, yes.

Q. In claim 1, they are available, right?

Page 62

MR. PLUTA: Object to form.

BY THE WITNESS:

A. Does it say that, that it has to be available? I don't see that.

BY MR. TUMINARO:

Q. Well, it doesn't say if they are available, does it?

A. Yeah. So, to be more general, it doesn't say it's a necessary condition for the arbiter to work.

Q. I apologize.

A. If you let me finish the thought, please. I'm trying to clarify what's on the claim. As you probably know better than I do, these claims sometimes are not very well written.

But in this case it doesn't say the requirement of having both available. It says it has the option of choosing between the two. I agree with that.

Q. You said the job of the arbiter is to select between available options?

A. Yeah.

Q. Right. You're talking about the claimed arbiter.

Page 63

A. That was a general comment, but it applies here.

Q. The claimed arbiter, the options are pixel command threads and vertex command threads?

A. If they're available.

Q. If they're available is not a limitation that appears in claim 1, right?

A. It doesn't say that.

Q. Okay.

A. But it could be interpreted by whoever is looking at this.

Q. But it's not a limitation that's in the claim?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. Right.

BY MR. TUMINARO:

Q. So an arbiter, the claimed arbiter, has to decide or resolve something?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. I'm not sure if I understand that question.

Page 64

BY MR. TUMINARO:

Q. Well, it has to select?

A. It has to select. I agree with that.

MR. PLUTA: Object to form.

BY MR. TUMINARO:

Q. Does a FIFO select? Do you know what a FIFO is?

A. First-in first-out.

Q. First-in first-out buffer?

A. Okay.

Q. Does a FIFO select?

A. You could design a FIFO that has a select line.

Q. But Does a FIFO by itself select?

A. I mean, you're talking about a FIFO without giving me any information about what you want to do. So it's a very broad question. If I give you a FIFO description, and I say I want to have an arbiter in front of it, you can design it.

Q. I'll tell you what. What is a FIFO?

A. First-in first-out.

Q. What does it do?

A. The first-in first-out, basically that's how it operates.

Page 65

Q. It stores the first thing that it gets and it outputs -- the first thing that's input is the first thing that's output, is that what you're saying?

A. I think FIFO means that.

Q. So does a FIFO select between available options?

A. You can design it to do that.

Q. How would you design it to do that?

A. You put an arbiter in it.

Q. So a FIFO is not an arbiter itself?

A. No. It depends on what you want to do. If you want it to have an arbitration, you could. I could design a FIFO with an arbiter.

Q. You would have to put an arbiter in front of the FIFO for the FIFO arbitrate?

A. You have to put the functionality in the FIFO. If you need that, you put it in, and you call that a FIFO. That will be my FIFO because I need that functionality. There is no set -- well, you could look up a dictionary and come up and say FIFO means this; but if I want to design it, I can add anything I want to it and it will be a FIFO. I have done that.

Page 66

Q. Do you know what a bus is in a computer context?

A. Yes, I do.

Q. A bus provides data?

A. It does.

Q. It receives data?

A. Not only data.

Q. What else does it receive?

A. Other things.

Q. Is a bus an arbiter?

A. It could be.

Q. How?

A. Put an arbiter in it as part of the bus.

Q. So the arbiter is something different than the bus?

A. It could be integrated with the bus.

Q. What's a register?

A. It's memory.

Q. Memory. What's a register do?

A. Store information.

Q. So it could receive information?

A. It has to if it wants to store information.

Q. It could provide information?

Page 67

A. It can read it and write it.

Q. Is a register an arbiter?

A. Tough question.

Q. You are the expert.

A. You are asking a very abstract question. A register is part of the arbiter.

Q. It is part of an arbiter?

A. If you design an arbiter without a register, you have a design that may not work.

Q. So it's a portion of an arbiter. What else would you need to add to the register to make it an arbiter?

A. Oh, this is a lot of things.

Q. Some sort of logic to select, right?

A. Yeah. You asked me outside the scope of this case. Are you asking me a general question about computer architecture? Because I want to know what the relevance is to what my report is and so on. If you could kindly direct me to what I'm prepared to talk to you about here. I mean, I can talk about these things if I have to because that's my job to answer your questions, but I don't see the relevance to this.

Q. Earlier you said a register can read and

Page 68

write information.

A. In the context of, sorry, computer architecture, yes.

Q. In the context of computer architecture?

A. In general, yes.

Q. Would that be true also in a GPU?

MR. PLUTA: Counsel, I'm going to object. I gave you a little leeway as the witness said. Can you tie this back to the scope of his report and particular information in his report? If not, I'm going to object to the relevance at this point.

MR. TUMINARO: Counsel, I'll draw your attention to the scheduling order, and it reminds everyone about the testimonial guidelines, and the testimonial guidelines are explicit that objections should be limited to a single word or term. Examples of objections that would be properly stated are objection, form; objection, hearsay; objection, relevance; and objection, foundation.

So I'm going to ask you to refrain from speaking objections from now on.

Page 69

MR. PLUTA: Then I'll restate my objection. I object to the relevance of this line of questioning.

BY MR. TUMINARO:

Q. If you would turn with me to in your declaration to Page 10. And there's a figure at the top of Page 10, the figure from your lectures.

A. Yes.

Q. On the right-hand block that says CPU and on the left-hand block, there's a block inside with what is labeled as CPU and a block is R-E-G-S.

Do you see that?

A. Yes.

Q. That stands for register?

A. Yes.

Q. Is that register that you're showing in your diagram an arbiter?

A. Not in that case.

Q. Because it doesn't select?

A. That one is used for holding temporary data for the PC -- for the CPU. I get confused that PC is program counter.

(WHEREUPON, document marked as Exhibit No. 7.)

Page 70

BY MR. TUMINARO:
Q. You have been handed what has been marked as Exhibit 7 for your identification. This is an U.S. Patent 7363427. This is the Stuttard patent.
A. Yes.
Q. You have reviewed this patent?
A. Yes, sir.
Q. It's listed on your materials considered?
A. Correct.
Q. If you look at figure 3 of Stuttard, what is shown in figure 3 what's labeled as a processing block 106.
Do you see that?
A. Yes.
Q. And each of these processing blocks include a plurality or an array of processing elements 1061?
A. Yes.
Q. If you turn with me to column 4, line 33, it reads in the example shown in Figure 2, "the processing core 10 is provided with eight processing blocks 106."
Do you see that?
A. Sorry. Column four, line what?

Page 71

Q. 33.
A. 33. Core 10? Processing core 10 is provided with eight processing blocks 106. Processing core 10. Where is processing core 10?
Q. If you look at figure 2B. I'll try to help you.
A. Thank you.
Q. Are you there, sir?
A. Yes.
Q. You see it sort of toward the top left, there's a label 106, and it says eight times fusion blocks.
Do you see that?
A. Yes.
Q. And that 106 is the same number as shown in figure 3, the processing blocks?
A. Okay.
Q. Right. So in this example, there are eight processing blocks; is that right?
A. Eight 106s, that's correct.
Q. Now, Stuttard says that the -- well, strike that.
Stuttard does geometry processing?
MR. PLUTA: Object to form.

Page 72

BY MR. TUMINARO:
Q. Let me clarify. The system disclosed in Stuttard does geometry processing?
A. Let me check.
Yes.
Q. Okay. And the system disclosed in Stuttard also does rasterization or pixel-type operations?
A. Correct.
Q. In fact, Stuttard discloses that the geometry operation and rasterization operations happen in phases, right?
MR. PLUTA: Object to form.
BY THE WITNESS:
A. Can you show me what you're talking about?
BY MR. TUMINARO:
Q. Sure. If you turn to column 18. I think it is line 11. I'll read it for you.
A. Do you mind if I read it on my own and then you can read it for the record. I would prefer that, if you don't mind.
Q. Sure.
A. Go ahead.

Page 73

Q. I'll read it for the record.
A. Thank you.
Q. "The bidding process must maintain primitive order between the geometry and rasterization phases due to the requirement of most host systems."
Do you see that?
A. Yes. You read it correctly.
Q. That means Stuttard discloses a geometry phase as distinct from a rasterization phase, right?
A. It says phases. I don't know what these phases mean.
Q. You reviewed Stuttard, right?
A. I didn't memorize it.
Q. You provided an opinion about what Stuttard discloses, right?
A. I did. But I don't recall what the phase means here. We can look at it if you want.
Q. Sure. The next sentence in column 8, it says since both phases, referring back to the geometry and rasterization phases, are block parallel, there needs to be a mechanism for transferring data between any block to any of the bins between any bin and any block.

Page 74

Do you see that?

A. Yes.

Q. When it says that the phases are block parallel, that means all the processing blocks 106 do geometry processing during the geometry phase and all the processing blocks 106 do pixel processing during the rasterization phase, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. I could not tell you that.

BY MR. TUMINARO:

Q. Well, let's go down then to line 18. It says: "This is implemented by creating multiple bins -- strike that. Let me start again.

"This is implemented by creating multiple bin lists per region, one for every processing block 106 that is processing geometry data. This allows the geometry output phase to proceed in block parallel mode."

Do you see that?

A. You're reading it correctly.

Q. That means the geometry phase is performed in block parallel mode, right?

MR. PLUTA: Object to form.

Page 75

BY THE WITNESS:

A. You're reading what it says or you're asserting what it says.

BY MR. TUMINARO:

Q. So what that means is that block parallel, during the geometry phase, all processing blocks 106 are doing geometry operations; right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. I could not tell you that. The phases could happen at the same time or in different levels of -- you know, this is very detailed analysis of what Stuttard does.

BY MR. TUMINARO:

Q. But Stuttard says they occur in block parallel, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. That has nothing to do with phases. That means it is data parallel.

BY MR. TUMINARO:

Q. It says there is geometry phases.

A. No. Block parallel means other things. Data parallel is a different business.

Page 76

Q. Let's look further down at line 29. "A record is kept of how many primitives are written to each bin so that regions can be sorted into similar size groups for block parallel rasterization."

Do you see that?

A. Yes.

Q. Block parallel rasterization means all blocks are doing rasterization, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. Yes. I think you're taking this out of the original context. Block parallel doesn't mean that there's no other activities going on. That means there's a data parallelism in the computation meaning that you apply a single instruction to multiple data to some extent. So I think you're taking a block parallel out of its original context, which means multiple things could happen at the same time.

So, I mean, I'll be glad to look at the phases and so on, but that's not the way I read this. To have more information, I mean I have to look at the phases.

I don't want to take too much time to

Page 77

explain what the block parallel is, but it has to do with data parallelism, and I think that means the processing elements are working, all of them, on one block, like you said, but there are multiple processes that could do that.

Q. In fact, in Stuttard, there is one core processor, that is core processor 10, right? Look at figure 2B.

A. 2B. Okay. Yes.

Q. And that core processor has eight blocks, right?

A. Yes.

Q. And column 18 says that geometry output phase proceeds in block parallel mode, talking about all the blocks 106 on core 10?

A. Yes. But figure 3 shows you -- you see the dot dots on the processor unit? Inside the belly of 106, there are whole bunch of processors inside that. You have got eight times whatever that is. I have to read more to figure out how the block is done. Is it done across this or inside this? I would be glad to look at all of this if you want. But I disagree with your statement there.

Q. Let's look at what each processing

Page 78

element within a block does. If you'll turn with me to column 9.

A. Sure.

Q. Line 57.

A. Okay.

Q. It says, "the data is loaded into the PEs of the graphic system so that each PE contains data for one vertex." Do you see that?

A. Which PE are we talking about?

Q. Those are the PEs that are inside a block 106 if you look at the figure 3.

A. Okay.

Q. There's processing elements 1061.

A. I understand.

Q. You agree that at column 9, paragraph -- line 57, Stuttard is saying that each of the processing elements stores vertex data, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. Reading that, it says each -- the data is loaded into the PEs of the graphic system so that each PE contains the data for one vertex. That I understand.

Page 79

BY MR. TUMINARO:

Q. So each PE has vertex data?

A. One of the things it does, it has the vertex data among other things.

Q. Each PE in block 106 has vertex data according to what Stuttard is saying?

A. For this example, yes, but it could have other things.

Q. You. But Stuttard doesn't say that, right? Stuttard said each PE has vertex data.

A. In that sentence, it says that. But I have to look at the rest to see if it does other things.

Q. It continues on to say "that each PE then represents a vertex of a primitive that could be at an end of a line or part of a two-dimensional shape such as a triangle."

A. For that paragraph, that is correct. But, again, PEs could be doing other things.

Q. Then let's look at what Stuttard says about pixel processing. Let's look at what it says about pixel processing.

At column 10, line 30, "each PE then transfers in its data concerning its primitive to

Page 80

the ME for processing in to the pixel data."

Do you see that?

A. Can you tell me the column again?

Q. Column 10, line 30.

A. Thanks.

Q. Do you see that?

A. Yes.

Q. The last sentence in that paragraph, line 39, it says: "Each PE also includes data about a respective pixel (ie, data is stored on a pixel per PE basis)."

Do you see that?

A. You're reading it as it says, yes.

Q. In line 45 -- well, I'll read line 42. "Once each pixel is determined to be outside or inside the triangle (primitive) concern, the processing for the primitive can be carried out only on those pixels occurring inside the perimeter."

Do you see that?

A. You're reading it correctly, yes.

Q. And then it goes on to say that "the remainder of the PEs in the processing block do not take any further part in the processing until that primitive is processed."

Page 81

Do you see that?

A. Yes.

Q. That means during the pixel operation, all the PEs are doing pixel operations or nothing, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. With the caveat that there are eight slices of those, and other slices could be doing other things. You just looked at one slice.

BY MR. TUMINARO:

Q. Okay.

A. I agree with what you said about one slice. Across -- let me finish, please.

MR. PLUTA: You guys are talking over each other.

THE WITNESS: I'm trying to finish my explanation if you don't mind. There are eight slices, and they could be doing other things including vertex.

BY MR. TUMINARO:

Q. But each slice, when it's doing pixel operations -- the slice you're talking about at block 106, right? When it's doing pixel operations,

Page 82

every processing element within block 106 does either a pixel operation or nothing, right?

A. Can I -- not having read the whole thing again here, by looking at what you showed me that you are correct, but I want to qualify that that I need to check the whole document to make sure that it doesn't say another embodiment we can do these. I want to be clear on that.

Q. But here, that's what this means?

A. In this example as you provided -- if it's one embodiment, you're right. But if the other embodiments are there, I'm not agreeing to what you say.

Q. Okay. So that's for pixels.

And then for vertices, within a processing block, each processing element within that processing block does vertex operations, right?

A. As I tried to explain, it could be that each slice is doing different things at the same time -- at the same time.

Q. I appreciate that. But what I'm trying to get out now is what a particular slice or processing block is doing. And if a processing block is doing pixels, all the processing elements

Page 83

within that processing block do pixels or nothing, we established that, right?

A. For this embodiment.

Q. Now, I want to talk for vertices. If a processing block is doing vertices, all the processing elements within that block do vertices, right?

A. Can you show me that section because you showed me for the pixel, not vertices.

Q. Let's look for the vertices.

Column 9, line 57, "the data is loaded into the PEs of the graphic system so that each PE contains data for one vertex."

That means every PE has vertex data, right?

MR. PLUTA: Object to form.

BY MR. TUMINARO:

Q. Did you nod your head? Is that a yes?

A. Sorry. Apologies. I'm reading and listening to you at the same time.

Okay. Yes. You're reading what it says, yes.

Q. Well, just to be clear, that means that all the processing elements within that block 106

Page 84

are doing vertex operations when block 106 is doing a vertex operation?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. Okay. It doesn't say it has the data for one vertex only. It could have data for pixels already in the register. So I cannot say from this statement that it does not have other information. If it said only vertex, you're right.

BY MR. TUMINARO:

Q. It does say each PE, right?

A. No.

Q. I'll read it again.

A. I didn't -- if I could qualify what I said.

Q. Let me read it again for the record.

A. Sure.

Q. "The data is loaded into the PEs for the graphic system so that each PE contains data for one vertex."

Did I read that correctly?

A. You did. It's not convincing to me that it's the only thing that it has.

Q. You don't read each PE to mean that every

Page 85

single PE in the block has vertex data?

A. No, Counsel. It should have said that only vertex data, then I agree with you. It doesn't say only. Does it say?

Q. It says each PE --

A. Right.

Q. -- has vertex data. So you're saying a PE could have other data?

A. Absolutely. From previous computations. It actually does something like that because you need to keep track of the state of what you did. The registers -- you showed me the register file. They have registers. They keep track of the state.

MR. TUMINARO: Why don't we take a break.

THE VIDEOGRAPHER: We're going off the record. This is the end of Media 2. The time is 11:15 a.m.

(Whereupon, a break in the proceedings was taken.)

THE VIDEOGRAPHER: We're back on record. This is the beginning of Media 2. The time is 11:36.

BY MR. TUMINARO:

Q. Welcome back, sir.

Page 86

A. Thank you.
Q. You have heard of the phrase unified shader?
A. Yes.
Q. And unified shader refers to an architecture where the same hardware does both vertex operations and pixel operations, right?
MR. PLUTA: Object to form.
BY THE WITNESS:
A. That's the general idea.
BY MR. TUMINARO:
Q. The claims of the '053 patent are directed to a unified shader, right?
A. Did they use the word unified shader anywhere? Can you point me to it?
Q. I guess, if you look at claim 1, it is talking about a single set of hardware that does both vertex operations and pixel operations, right?
A. Where do you get that information from claim 1?
Q. Well, the memory stores both vertex commands and pixel demands, right?
A. That's the memory.
Q. And the arbiter selects either vertex

Page 87

commands or pixel commands, right?
A. Correct.
Q. It's a single set of hardware that's doing --
A. Overall, it's hardware with multiple functional units.
Q. It's doing vertex operations and pixel operations?
A. It's really a broad definition, unified shader. You could have specialized hardware, but I mean, I haven't really opined on that for this case.
Q. I guess I'm asking you right now as an expert in this field, would you consider claim 1 to be directed to a unified shader?
MR. PLUTA: Objection, relevance. Objection, form.
BY THE WITNESS:
A. I haven't formed an idea on this because it's really not a matter of just being a programmable shader. One has to see more details about it. I couldn't tell you for certain that this is a unified shader because there are no more details about what these functional units do. I cannot say from what I have here.

Page 88

But the understanding of unified shader in general, it's a programmable system that can handle multiple functions as part of shading. That's the understanding I have.
BY MR. TUMINARO:
Q. Does claim 1 handle vertex operations and pixel operations?
MR. PLUTA: Object to form.
BY THE WITNESS:
A. Claim 1 says that an arbiter -- okay. Does it handle vertex operations? It refers to vertex data. I agree with that. It says that it will select between vertex and pixel data. That, I see and I agree with the statement here.
BY MR. TUMINARO:
Q. If you would turn with me with Paragraph 19 of your declaration.
A. Paragraph 19, okay.
Q. Page 5 if that helps. It reads, "I also understand that the relevance" -- excuse me. I'll start again.
"I also understand that the relevant inquiry into obviousness requires consideration of four factors."

Page 89

Do you see that?
A. Yes.
Q. And one of the factors that's listed as (d) is objective factors, right?
A. Correct.
Q. And one of the objective factors is copying of the invention by others in the field?
A. Commercial success, long-felt needs, copying of the invention by others in the field. That's what it says, yes.
Q. You would agree with me that copying of the invention by others is objective evidence of nonobviousness?
MR. PLUTA: Object to form.
BY THE WITNESS:
A. Okay. You are asking me -- I know it's in my report, but this is a legal matter. My level of understanding, it is exactly what it says here. I cannot dwell on this more than what it is. You know, I'm not here for legal issues; but you're right, it is in my report and that's what it is. I cannot add any more than what you see there unfortunately. Sorry.

Page 90

BY MR. TUMINARO:

Q. Okay. You're aware, as an expert, that nearly the entire computing industry now uses unified shader?

MR. PLUTA: Object to form. Objection, relevance.

BY THE WITNESS:

A. Yeah. I could not tell you that. I have not opined on that for my report. It requires looking at the entire industry.

BY MR. TUMINARO:

Q. You have heard of the DX10?

MR. PLUTA: Object to form. Sorry. Withdraw the objection to form. But objection to relevance.

BY THE WITNESS:

A. I have heard of it.

BY MR. TUMINARO:

Q. It's an application programming interface developed by Microsoft?

MR. PLUTA: Objection, relevance.

BY THE WITNESS:

A. You are asking me something outside the scope of my report. It is an API, but that's all I can tell you about that. I need to evaluate it more.

Page 91

BY MR. TUMINARO:

Q. I'll just say for the record your report talks about objective indicia and copying is relevant inquiry into obviousness. You opined on obviousness, right?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. I did opine on obviousness to the level of an expert using it to evaluate the prior art, but secondary indices or whatever you call it, that's not my specialty. I will not be able to give you any meaningful example, although I understand some of it.

BY MR. TUMINARO:

Q. As an expert -- you claim to be an expert in computer graphics?

A. Yes.

Q. You have heard of DX10?

MR. PLUTA: Objection, relevance.

BY THE WITNESS:

A. Yes.

Page 92

BY MR. TUMINARO:

Q. And computer graphics chips have to be compatible with DX10 if they want to work on that platform, right?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. Which computer graphic chips? There are a whole bunch of computer graphic chips.

BY MR. TUMINARO:

Q. Any graphic chip that's want to be compatible with DX10 would have to implement what DX10 talks about, right?

MR. PLUTA: Same objection. I'm sorry, Counsel. That prior objection also included a relevance objection.

BY THE WITNESS:

A. Without having studied it, if I say something is able to do a particular application or API, then I have to be able to do it. I can answer that in a general term. I would not advertise that I could do graphics for DX10 API and not be able to do it.

Page 93

BY MR. TUMINARO:

Q. Let me ask this, I guess.

A. Sure.

Q. DX10 includes a unified shader mode, right?

MR. PLUTA: Objection, form. Objection relevance.

BY THE WITNESS:

A. I have not looked at that.

MR. TUMINARO: I'll have this marked.

(WHEREUPON, document marked as Exhibit No. 8.)

BY MR. TUMINARO:

Q. You have been handed what has been marked as Exhibit 8. This is the DX10 architecture.

Do you see that?

A. I do.

MR. PLUTA: I'm going to object to the introduction of this exhibit as irrelevant. Also hearsay.

BY MR. TUMINARO:

Q. If you will turn with me to Page 14. Page 14 at the top says DX10 shader model 4.0.

Do you see that?

Page 94

A. Yes.

Q. After it reads "shader model 4.0 (SM 4.0) is the new instruction set architecture (ISA) for DX 10 that looks at the graphics in a unified way."

Do you see that?

A. Yes.

Q. If you look at the second bullet, it says, "flexible load balancing."

Do you see that?

A. Yes.

Q. And the third sentence says that "the unified shader is made up of shader blocks that could handle all vertex, pixel, and geometry instructions."

Do you see that?

A. Yes. You're reading the sentence.

Q. Okay. So if a GPU wants to be compatible with DX10 shader model 4.0, the GPU has to offer a unified shader, right?

MR. PLUTA: Objection, relevance. Objection, hearsay.

BY THE WITNESS:

A. I could not tell you that from that sentence. You picked one sentence out of this

Page 95

document. I'm a very precise person. If I have to make a comment on that, I cannot do that with one sentence. You lifted one sentence out of this document that I have never seen before.

BY MR. TUMINARO:

Q. Have you used DX10 shader model 4.0?

MR. PLUTA: Objection, form and relevance.

BY THE WITNESS:

A. No, I have not.

BY MR. TUMINARO:

Q. Have you used any of the later versions of DX, DX11, DX12 offered by Microsoft?

MR. PLUTA: Objection, relevance. Objection, form.

BY THE WITNESS:

A. I don't know how it's related to my report.

BY MR. TUMINARO:

Q. You are an expert in graphics, right?

A. Yes.

Q. Supposedly.

A. I do not appreciate the comment about supposedly. But I have not used this one, no, and

Page 96

then later on.

Q. You don't know one way or the other whether DX10 or its later version require a unified shader?

MR. PLUTA: Objection, form. Objection relevance.

BY THE WITNESS:

A. If I read the documents, I can tell you for sure.

BY MR. TUMINARO:

Q. Why don't you read it.

A. If I need additional documents, can I get them if it's not sufficient?

Q. Why don't read the document and we'll go from there.

A. Okay.

MR. PLUTA: While he's doing that, I'm just going to also lodge an objection to the authenticity of this document.

In addition to the authenticity, I'm going to object also to the foundation with the introduction of this exhibit.

THE WITNESS: The question was on 14. Okay, your question is?

Page 97

BY MR. TUMINARO:

Q. After reviewing this --

A. Up to here. I haven't finished. Do you want me to finish?

Q. Page 14?

A. Yeah.

Q. You can look at Page 14.

A. No. I haven't finished the whole document. I stopped at 14. Do you want me to read more?

Q. I guess -- let's see if you can answer my question.

After reviewing this, does DX10 shader model 4.0 require a unified shader?

MR. PLUTA: Objection, form. Objection relevance.

BY THE WITNESS:

A. Although it is not related to my report and I had a quick review of this document, it provides a programmable shader.

BY MR. TUMINARO:

Q. A unified shader?

MR. PLUTA: Objection, form. Objection relevance.

Page 98

BY THE WITNESS:

A. A computing resource that can be programmed to do different shading.

BY MR. TUMINARO:

Q. So before you submitted your declaration, you didn't know one way or the other whether DX10 required a unified shader?

MR. PLUTA: Objection, form. Objection relevance.

BY THE WITNESS:

A. I'm very familiar with unified shaders.

BY MR. TUMINARO:

Q. But my question is did you know that DX10 required a unified shader?

MR. PLUTA: Same objections.

BY THE WITNESS:

A. No, I did not.

BY MR. TUMINARO:

Q. You have heard of open GL?

A. Yes.

Q. Before you submitted your declaration, did you know one way or the other whether the current version of open GL required the unified shader?

Page 99

MR. PLUTA: Objection, form. Objection relevance.

BY THE WITNESS:

A. I actually knew about open GL and unified shaders. And now I recall that DX. I also knew about unified shaders for another.

BY MR. TUMINARO:

Q. So they both required unified shader?

A. At that time I knew, yes.

MR. PLUTA: Objection, form. Objection relevance.

BY MR. TUMINARO:

Q. In Paragraph 19 you mentioned -- of your declaration.

A. Okay.

Q. You mentioned objective factors are one of the things that must be required for an obviousness analysis.

A. Okay. I say that. And the question is?

Q. Well, that's true, right? Objective indicia is one of the things that must be analyzed in an obviousness consideration?

MR. PLUTA: Objection, form.

Page 100

BY THE WITNESS:

A. I could not tell you. That's a legal issue. I cannot opine on that. I'm sorry. I mean I know I have it in my report. It is just to tell you that I know about this information, but I cannot go beyond what this sentence says. Analyze it or discuss it in any way.

BY MR. TUMINARO:

Q. Okay. In your report, you didn't analyze any secondary factors of nonobviousness, right?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. I don't recall I did that. If I'm missing something, please let me know. I don't recall doing that.

BY MR. TUMINARO:

Q. In this case, counsel provided you with all the prior art that you considered, right?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. I identified prior art myself.

BY MR. TUMINARO:

Q. You identified the prior art that you applied?

Page 101

A. Right.

Q. Did Counsel provide you with any evidence of objective indicia of nonobviousness?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. Again, I cannot comment on that because I'm not familiar with the exact term in legal.

BY MR. TUMINARO:

Q. Did you go out and try to see if anyone was copying the claimed invention?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. Going out means what? Going visiting companies and looking at chips?

BY MR. TUMINARO:

Q. Did you do any researching, any analysis, any inquiry to see if there was copying of the claimed invention?

MR. PLUTA: Same objections.

BY THE WITNESS:

A. Copying of the claimed inventions. Outside what I have in this report, I have nothing

Page 102

else to add.

BY MR. TUMINARO:

Q. So the answer is you didn't do that?

A. To see if anybody has copied the claims for the product?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. No. What is in this report is what I have done.

BY MR. TUMINARO:

Q. You didn't do any analysis about long-felt need for the claimed invention?

A. It's not in this report, correct?

Q. That's correct.

A. Whatever is in this report I have done.

Q. You didn't do any analysis of failed attempts by others to create the claimed invention?

A. Again, I don't recall; but if it's here, it's in the report.

Q. The fact is you didn't consider any objective indicia of nonobviousness in forming your opinion about obviousness; is that right?

MR. PLUTA: Objection, form.

Page 103

BY THE WITNESS:

A. That's a, again, legal matter. I will not want to opine on something I'm not very familiar with, but I mean, it says what it says.

BY MR. TUMINARO:

Q. You considered only prior art in forming your opinions on obviousness?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. I considered what's in the exhibits, whatever you see here. Prosecution history, patents and so on. These are the items I considered. I think we went through that with you. If I had done something, it would be here.

BY MR. TUMINARO:

Q. So now you remember now that DX10 and open GL require a unified shader?

MR. PLUTA: Objection, relevance.

BY THE WITNESS:

A. Yes, I looked at that before, yeah.

BY MR. TUMINARO:

Q. So if a graphics processing architecture wants to be compatible with either of those APIs, the GPU would have to have a unified shader, right?

Page 104

MR. PLUTA: Objection, form, objection relevance.

BY THE WITNESS:

A. I can't comment on that, had to. There might be other options.

BY MR. TUMINARO:

Q. If they want to be compatible with those APIs, they would have to include it, right?

MR. PLUTA: Same objections.

BY THE WITNESS:

A. It requires a thorough assessment. I cannot comment on that.

BY MR. TUMINARO:

Q. What assessment would you have to do to determine what --

MR. PLUTA: Objection, form. Objection relevance.

BY THE WITNESS:

A. Not during a deposition. I have to spend time to look into it, the details and so on. I will not be able to opine on that right here based on what you showed me.

BY MR. TUMINARO:

Q. And you didn't consider that -- any of that in forming your opinions on obviousness?

Page 105

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. Whatever I opined on is right here.

MR. TUMINARO: Let's break for lunch.

MR. PLUTA: Sure.

THE VIDEOGRAPHER: We're going off record. The time is 12:06.

(Whereupon, a short break in the proceedings was taken.)

THE VIDEOGRAPHER: We're back on record. The time is 12:50.

BY MR. TUMINARO:

Q. Welcome back, sir. You testified earlier I believe that you designed GPU?

A. I designed a processor that had GPU capabilities.

Q. What's the first step that you took in designing that processor that had GPU capabilities?

MR. PLUTA: Objection, form. Objection relevance.

BY THE WITNESS:

A. You are asking me a general question, not related to this case, right? Because I haven't

Page 106

opined about that particular question in this document.

BY MR. TUMINARO:

Q. What was the process -- what was the first step that you took --

MR. PLUTA: Same objections. Sorry.

BY MR. TUMINARO:

Q. -- in developing the chip that had a GPU capability?

A. Coming up with a new idea. Epiphany. You take the idea to the process of simulating it, you simulate using some sort of a cycle accurate simulator.

Q. You said?

A. Cycle accurate simulator.

Q. Cycle accurate, is that what you said?

A. Yes. And then once you're happy with the results, you move onto the level of, like we talked about, writing the VHDL code.

Q. You simulate it. What are you simulating then at that point?

MR. PLUTA: Objection, form. Objection relevance.

Page 107

BY THE WITNESS:

A. The ideas. The architectural ideas.

BY MR. TUMINARO:

Q. I guess what I'm confused about, what I don't understand is what form are the architectural ideas in? Are they on paper? Are they in your head?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. Well, I mean, they go from your head to the piece of paper, and then finally you write the software to represent the architecture.

Let's use the morphosys as an example since you had done your due diligence and looked at that paper. We had to simulate it before we go into fabricating it. We had funding from the government agencies. It's unusual for a university to actually fabricate a chip as you probably were alluding to. But we had funding. We went simulations -- we did simulations and then ultimately we went to the VHDL model and did the layout.

BY MR. TUMINARO:

Q. Oh, okay. You simulated, what, some kind

Page 108

of an C code or something?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. Yeah. Right.

BY MR. TUMINARO:

Q. After you did the simulations of the C code, that's when you went to the VHDL?

MR. PLUTA: Same objections.

BY THE WITNESS:

A. Yes. That's what we do. That's what we do. That's what we did then, and we continue to do the same thing.

BY MR. TUMINARO:

Q. Is that what is done in the industry?

A. It is.

MR. PLUTA: Objection, form. Objection, relevance.

BY MR. TUMINARO:

Q. Is that what you teach your students?

MR. PLUTA: Objection, relevance.

BY THE WITNESS:

A. Yeah. That's what we do. We not only we teach our students, we also -- we continue our

Page 109

research in that direction. It would be unwise to go and design the hardware without having it modeled. And that's true for every field I would say, not just us. And I think -- I would hope not thinking that it is my personal style. It's the style of all the designers do -- go for it. I mean, they basically model simulate. Then they go to the hardware.

BY MR. TUMINARO:

Q. What does the model tell you?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. If you recall, I said cycle accurate. So it gives you the number of cycles it will take to do a computation.

So, the morphosys project, we had a cycle accurate or close to cycle accurate model of the system. We massaged certain things.

BY MR. TUMINARO:

Q. What does that mean?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. Meaning that we had -- I don't know your

Page 110

background but engineering is all about tradeoffs and, you know, we're coming up with new things instead of finding out what is already there. Like, what physicists and chemists will do, they find things that are already there. We're trying to come up with new things. There's a lot of trial and error. So that's why when you model and simulate based on the application, that application presents itself, it says this particular piece of architecture is not the right choice for me. We go and massage it and go change it. I don't know when I start whether I have the optimal solution or not. It is a very trial and error basis.

Q. By massage, you're saying it helps you determine what the appropriate architecture should be?

A. Adjusted. Yeah. Think of it as dials. You used the word cache as an example. So let's use that as an example. You find out the size of the cache is not appropriate, then you change the cache.

Q. And you could tell that kind of stuff from your model of the chip?

MR. PLUTA: Objection, form. Objection, relevance.

Page 111

BY THE WITNESS:

A. Yeah. I mean, that's basically what it is.

BY MR. TUMINARO:

Q. Switching topics. I think earlier you mentioned that an arbiter selects from available options, right?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. Arbiter is -- the job of arbiter is to selecting available inputs.

BY MR. TUMINARO:

Q. So, is an accurate definition of a arbiter any computer hardware, software, or combination thereof that receives and provides a command thread?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. I think that's the claim construction that I offered, I think. Let me check that, please, just to make sure that you got it right.

Reading for the record Paragraph 51, "the '053 patent discloses that an arbiter may be an implementation of hardware, software, or a

Page 112

combination thereof such that it receives the command thread and thereupon provides the command thread to a command processing engine."

So that's my definition.

BY MR. TUMINARO:

Q. What part of that definition describes that the arbiter selects from available inputs?

A. It basically describes how the arbiter is implemented, and arbiter as we described is the one that selects between inputs.

Q. Just to be clear, your interpretation of arbiter is in Paragraph 52 of your declaration, right?

A. Yes. That's the implementation of it. This is the definition of it, right. 51 is the -- how you implement it and 52 explains what it does.

Q. So where in your definition in Paragraph 52 does it explain that the arbiter is selecting from available inputs?

A. It says it receives and provides a thread.

Q. You're reading receives and provides to mean selects from available inputs?

A. Yes.

Page 113

Q. If you turn with me to the Lindholm reference.

A. Okay.

Q. At figure 4 in Lindholm, there's the thread control buffer 420?

A. Yes.

Q. I just want to make sure I understand your position. Is it your position that the thread control buffer 420 in Lindholm is the claimed memory device in claim 1 of the '053 patent?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. In combination with instruction cache.

MR. TUMINARO: Let's have this marked, please.

(WHEREUPON, document marked as Exhibit No. 9.)

BY MR. TUMINARO:

Q. Okay. You have been handed what's been marked as Exhibit 9. This is the petition for inter partes review of U.S. Patent 00742053. You have seen this document before?

A. I have seen it. It's not on my list of materials considered, but I've seen it, yes.

Page 114

Q. If you will turn with me to paragraph -- I mean Page 13?

A. Page 13. All right.

Q. And the first full paragraph, there's in the first sentence, after the comma, it reads: "Lindholm's thread control buffer." Do you see that?

A. Yeah.

Q. I'll read it for the record. "Lindholm's thread control buffer 420 corresponds to the claimed memory device."

Do you see that?

A. Yes.

Q. You disagree with the petitioner that the thread control buffer is the memory device?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. It's part of the memory device. It completes the memory device. It is an important part of the memory device.

BY MR. TUMINARO:

Q. So you have a different position than the petitioner?

MR. PLUTA: Object to form.

Page 115

BY THE WITNESS:

A. It is consistent with the petitioner in my opinion.

BY MR. TUMINARO:

Q. But you will agree that on Page 13 of the petition, it doesn't say anything about the memory device -- the claimed memory device also including the instruction cache, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. It also does not exclude it and it doesn't say it's the only thing so....

BY MR. TUMINARO:

Q. Let's go back to your declaration. And I'd like to go to Paragraph 22. It's on Page 7.

A. 22. Yes.

Q. And the second sentence of that paragraph reads: "As a result, I will assume the relevant time period for determining what one of ordinary skill in the art knew is mid-to-late 2003."

Do you see that?

A. Yes.

Q. That's an accurate statement?

A. That's what it says. I mean, this is my

Page 116

report. I agree with what's in there.

Q. You didn't consider any other time frame besides mid-to-late 2003?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. What do you mean by that?

BY MR. TUMINARO:

Q. Well, you considered the relevant time period for determining what a person of ordinary skill in the art to be -- to have known would be mid-to-late 2003?

A. That's what it says.

Q. That's the time frame you're considering. My question is: You didn't consider any other time period, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. Any other time meaning before that time. It could not be after that time, right?

BY MR. TUMINARO:

Q. It could be before. It could be after. You didn't consider any other times when determining what a person of ordinary skill in the art would have known?

Page 117

MR. PLUTA: Same objection.

BY THE WITNESS:

A. Well, if this was filed September, 2003, it would be very illogical for me to assume anybody after the filing date or after the application filing date, right? So your question is probably before that. It cannot be after that.

BY MR. TUMINARO:

Q. So you didn't consider after because you think it is illogical?

A. I mean, I'm not a lawyer, but I'm assuming that would not work very well.

Q. How about before? Did you consider before mid-to-late 2003?

A. Well, the person of ordinary skill in the art should have all those prior art references available to them, and if they were available, it would be sufficient, whatever that time frame is. They're taking the knowledge that they have, and they're combining those references. So that's all I can tell you about that. It should be inclusive of all those references.

Q. If you could go with me to Paragraph 116 of your declaration. It's on Page 40. Further, I

Page 118

guess at the top of Page 40, you have a figure 4 from the Stuttard reference, right?

A. That is correct.

Q. And if you look at Paragraph 116, the last sentence says "because cache unit 1024 includes both vertex and pixel command threads, it is the claimed memory device."

Do you see that?

A. Yes.

Q. What's your basis for saying that the cache unit 1024 in Stuttard includes both vertex and pixel command threads?

A. Because it feeds the instructions for the graphics processing all the instructions.

Q. What feeds that?

A. The cache.

Q. So how do you know it has both pixel command threads and vertex command threads?

A. It's identified as one cache and that's what it does. If there were more than one, it would have shown it, and also I can check to see what I have in my claim chart. Let me see if I can find you the text.

I mean, there's just the first part.

Page 119

It says the thread manager 102 is shown in more detail in figure 4, and comprises a cache memory unit 1024 for storing instructions fetch for each thread. And Stuttard does vertex and pixel so it is correct.

Later on, if you look at Page 43, the top of it, it says, for example, thread zero may be assigned for general system control, thread one assigned to execute 2D, two-dimensional activities, and threads 2 through 7 assigned to executing 3D activities, such as calculating vertices, primitives, or rastering.

The citation is column 5, lines 56 through 64.

Q. If you'd turn with me in Stuttard, Exhibit 7.

A. Okay.

Q. We talked about column 10, lines 42 through 47. Do you recall that?

A. Where -- when did I talk about that?

Q. Earlier today, I read this into the record. Starting at line 43, "once each pixel is determined to be outside or inside the triangle (primitive) concerned, the processing for the

Page 120

primitive can be carried out only those pixels occurred inside the primitive. The remainder of the PEs in the processing block do not take any further part in the processing until that primitive is processed."

A. I remember that discussion.

Q. We agreed that that embodiment discloses that in a given processing block all the PEs do either pixel operations or nothing, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. I don't believe I agreed with that.

BY MR. TUMINARO:

Q. You didn't agree to that earlier?

A. You can read what's on the record. I agreed with that's what you read which was the PEs were doing the same thing, and I also interjected that they could have state from other computations that are going on. And I also interjected that those slices could be doing different things.

Q. I guess my question is in this embodiment that is disclosed at column 10, line 45, the sentence reads that "the remainder of the PEs in the processing block do not take any further part in the

Page 121

processing until the primitive is processed."

So those PEs in the processing block do nothing, right?

A. That's what he's saying for that block.

Q. And the other PEs in that block in the sentence before that, he says those other PEs are doing pixel operations, right?

A. It says they're doing pixel, yeah. Yes.

Q. In column 10 in that paragraph that we're talking about, it starts at line 43 and goes to line 47, the first sentence says that the PEs in the processing block are doing pixel operations and the second sentence in that paragraph says that pixel -- the PEs are not doing anything, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. The remainder of the PEs. Not the PEs are not doing anything. It just -- that means if there are eight of them, maybe four are doing PE and four are not doing anything, although they may have residual computation from previous operations because they may hand it over to the next step.

BY MR. TUMINARO:

Q. Okay. Just to be clear, I just want to

Page 122

make it clear. This paragraph says that within a processing block, the PEs will either do -- if that processing block is doing pixels, the PEs in that processing block will either do pixel operations or nothing. Is that what this paragraph says?

A. Counsel, let's identify the processing blocks so there's no misunderstanding between you and me and the records. So I want it to be clear what I'm saying. So let's understand for the record that the eight fusion blocks, I'm saying that those could be doing different things. What you're saying what's happening inside the fusion block --

Q. That's right.

A. -- which is, for the record, it's figure 3, identified as 106 processors. Okay.

So, if that is referring to that -- no, I'm sorry, what you just read refers to 106 blocks, yes, that's what he says he's doing. Although we don't know anything about what residual state is in there and how the other blocks perform the vertex processes simultaneously.

Q. I want to make sure we agree on what the paragraph says at column 10 starting at line 43 and going to line 47. That paragraph is referring to

Page 123

processing going on within one processing block 106, right?

A. One slice of 106, yes.

Q. And that paragraph is saying that the processing elements within one processing block will be doing either pixel operations or nothing. Do we agree?

A. Yes, that's what it says. But I qualified my response -- my response to you that those other blocks, fusion blocks, are doing other things including vertex operations.

Q. Okay. Where does it say that?

A. We are going to find out.

Q. Look at it.

A. Can I have a pen or something to mark the areas that I want to refer to later on? Can I mark the exhibit?

Q. Yes, please do.

A. Okay. I didn't know what the rules were. Let me give you some explanations. If you are not happy, then I'll continue reading. How's that? We call that prefetching.

Okay. So are you ready?

Q. I am. At some point I am going to want

Page 124

my pen back.

A. I will give you your pen back.

MR. PLUTA: Is there a question pending just to refresh?

MR. TUMINARO: I'll ask the question again. Thank you.

BY MR. TUMINARO:

Q. Okay. Sir, you mentioned that the other blocks 106 could potentially be doing vertex operations when a single block is doing a pixel operation, is that right?

A. Yes, I am reconfirmed my position on that by reading the text.

Q. Tell me where.

A. I'll tell you the text and then we can see if it is convincing for you. So if you go to column 2, line -- I'm going to give you a whole bunch of excerpts from this to see how things are with this with respect if it is satisfactory. It says --

Q. What line?

A. I was just going to get to that. Let's read that top paragraph, "according to another aspect of the present invention, there is provided a

Page 125

data -- there is provided a data processing apparatus comprising an array of processing elements which are operable to process respective data items in accordance with a common received instruction. When the processing elements are operably divided into plurality of processing blocks having at least one processing element and the processing blocks being operable to process respective group of data items, this data items refers to a general group of data that is related to graphic processing including vertices and pixels."

So let's continue with that. Same column.

Let me give you all of the evidence that I found and then you can go and attack it if you wish. But let me finish the thought up to that point. Is that okay?

Q. Hm-hmm.

A. Column 2, line 52 or 51, "however, this embodiment is purely exemplary. This goes to the embodiment issue that we have talked about before. And it will be readily apparent that techniques and architecture described here for processing graphical data are equally applicable to other data types such

Page 126

as video data and so on."

If it covers video data, it for sure covers vertex and pixel because it is inherent in the discussion of graphics processing. So I would say this guy basically went beyond graphics.

So let's go to column 4, line 25. "As will be explained in greater detail below the processing core 10 includes a number of control units, thread manager 102, a rate controller 104, and channel controller 108 and bidding unit 1069 per block, and micro code store 105. These control units control the operation of a number of processing blocks which perform the graphic processing itself."

Again, vertex and pixel are a divisible part of the graphic processing. So it is also there for somebody who is reading this as an expert or a person with ordinary skill in the art.

Again, the same column, it says the array of --

Q. What line?

A. Sorry. Line 58, "the array 1061 of processing elements provide a single instruction multiple data processing structure." That's a data

Page 127

block parallel that I mentioned to you before. It is SIMD.

"Each PE in the array 1061 is supplied with the same instruction, which is used to process data specific to the PE concerned." So basically they have all the instructions they need to perform the different data types that we just talked or I just talked about earlier.

Now, notice that there is the thread controller at the top of that page, right, if you could look at it. It says a thread controller, and the job of that thread controller is basically to dole out --

Q. What page?

A. Thread manager. Thread manager is 102, figure 3 at the very top. Okay.

So the thread manager for a multi-threaded processor, which is responsible for different data types, is equally responsible for launching vertices. And the way it's shown in its design, it could easily provide that to these PEs simultaneously. But not to one group of PEs because they would be doing SIMD for pixels, as you said, and I also agree. But they could be doing different

Page 128

things because they're different data types, it's a multi-threaded, and it has the capability to launch these instructions to each slice of the PEs.

So, with that respect and the fact that he talks about embodiments that could do more, I consider that his invention or their invention is consistent with the fact that they can do PE and -- they could do pixel and vertices simultaneously.

Q. When you say they could do pixel and vertices simultaneously, you mean the processing blocks, one processing block could do vertex and one processing block could do pixel, that's your position?

A. That's right.

Q. The section that you didn't cite is column 18.

A. I didn't get to it. I thought people were getting impatient.

Q. In column 18 as we talked about earlier says at line 19 -- I guess, starting at line 17, "this is implemented by creating multiple bin lists per region, one for every processing block 106 that is processing geometry data. This allows the geometry output phase to proceed in block parallel

Page 129

mode."

A. Hm-hmm.

Q. The block parallel mode means that all the blocks 106 are doing geometry operations, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. It's an SIMD, yes. I think if you recall, I read that these blocks operate in SIMD.

BY MR. TUMINARO:

Q. If the block parallel mode is referring to blocks 106, right?

A. Yes. Each slice can do an SIMD operation.

Q. And in line 30 of that same column 18, it says, "a record is kept of how many primitives are written to each bin so that regions can be sorted into similar size groups for block parallel rasterization."

Do you see that? So block parallel rasterization means that all the blocks 106 are doing rasterization, right?

A. One slice.

Q. Block parallel means only one slice?

A. Again, if you go back to -- so this

Page 130

is parallelism -- the only way I can explain it to you is -- why don't we just use books as these fusion blocks. I have ten books on a book shelf. One book is doing rasterization. The other book is doing vertex operations. The other book may be doing something else. The design is not preventing you from designing it that way and the inventors talk about it, the flexibility of the design.

So I agree with you that one slice or one book of that bookshelf is doing what you said but the other books are doing different things and they can do different things because the thread is distributing information to different blocks.

Q. What's your interpretation of Stuttard's disclosure of block parallel rasterization? Doesn't that mean that all the processing blocks 106 are doing rasterization, they're doing that in parallel?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. One slice is doing the same thing. The other slice is doing different things.

BY MR. TUMINARO:

Q. What is the parallel -- what does parallel mean in that?

Page 131

A. SIMD, single instruction multiple data. The best way to explain SIMD -- morpho was an SIMD. Morphosys is an SIMD. If you read the paper, you saw I mentioned that it was SIMD. SIMD is single instruction multiple data.

Q. You don't interpret block parallel to mean that all the blocks are doing the same parallel type of operation?

A. That's what SIMD is, single instruction multiple data, but you have multiple slices. Think of it as a stack of these guys. Each slice -- so, if I wanted to do the morpho this way, I would have had two layers of morpho. One would be doing one type and the other another type. That is consistent with what he says.

Q. Maybe we're talking past each other. The slices that you're talking about in Stuttard -- strike that.

The slices that you're talking about Stuttard calls processing block 106, right?

A. Yes.

Q. So, when he says block parallel, doesn't he mean that the processing blocks 106 are doing parallel operations?

Page 132

MR. PLUTA: Objection to form.

BY THE WITNESS:

A. You have your hands like this so I have a feeling that you want to say that all the fusion blocks you're talking about. I disagree with that. One is doing that. The other slice might be doing something else. He has that flexibility.

BY MR. TUMINARO:

Q. So it's your interpretation that when he says block parallel, the block that he's talking about there is not processing block 106; is that your --

MR. PLUTA: Object to form.

BY THE WITNESS:

A. No. It is processing block 106. Let help refer you to the figure if you don't mind. For the record, we're looking at figure 3.

BY MR. TUMINARO:

Q. Mark it up.

A. Okay. I'm going to letter name them. So it will be A, B, and C. So I have shown you -- of course, there's a dot dot, here, right? It could be like E, G, F. So that is one slice. Okay.

Q. A is one slice?

Page 133

A. That's right because you got the PEs. They can work in SIMD fashion. There is another slice right behind it.

Q. Slice B?

A. That's right. It could do the SIMD, and it could be doing something else. He says that. We work on different data types. The slice D and so on. So it gives you the capability to function multiple SIMDs at the same time.

Q. Okay. Referring to figure 3 -- thank you. Referring to figure 3, block 106 on the top, which you labeled as A, is shown as being parallel to block B, correct?

A. Yeah. I see where you're -- Sorry, you might be misunderstanding here. We're not talking about parallel there. We're talking about parallel here.

Q. So -- but just to be clear --

A. I understand what -- I see what your confusion may be.

Q. Your interpreting block parallel -- when Stuttard is talking about block parallel in column 18, he's not referring to processing block 106?

A. No, no. Not by any stretch. If he's

Page 134

inventing his own definition, that's a different ball game. Somebody who is working in the field, a POSITA, would understand a block parallel, it means that PEs here are working in parallel. They're doing an SIMD function. So just think of it as 106 slice A is a morpho.

Q. How would a person of ordinary skill in the art know that when Studdard talks about block parallel mode in column 18 -- let me start over.

How would a person of ordinary skill in the art know in 2003, reading Stuttard, that Stuttard meant block parallel mode in column 18 to refer to an SIMD device?

A. Did we not say what degree the person should have as a POSITA? I think we did opine that he has to have a degree in electrical engineering or computer science, and that topic is covered. And somebody working in the field with a background, somebody taking course in engineering, especially computer engineering and computer science would know that what SIMD is. We cover it. Our degree is a very typical ABET accredited. I think that's true for everybody.

Q. SIMD was well known in 2003?

Page 135

A. Morpho was an SIMD, so...

Q. In fact, at other times when Stuttard meant SIMD, he explicitly said SIMD, right?

A. That's right.

Q. For example, in column 1, line 21, it says, "the present invention relates to parallel data processing apparatus, in particular to SIMD, (single instruction multiple data) processing apparatus." Right?

A. Yes.

MR. PLUTA: Objection, form.

BY MR. TUMINARO:

Q. So when Stuttard meant SIMD, he said SIMD, right?

MR. PLUTA: Object to form.

BY THE WITNESS:

A. When he said SIMD, he meant SIMD. I don't know. I mean, that's probably what he meant.

BY MR. TUMINARO:

Q. So when he wanted to refer to a single instruction multiple data, he said SIMD, right?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. He says a previously proposed -- he talks

Page 136

about SIMD, yeah.

BY MR. TUMINARO:

Q. Column 18 in the lines that we read doesn't say SIMD?

A. It is implied.

Q. Doesn't say it, right?

A. A POSITA would know.

Q. Just SIMD does not appear in column 18?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. Out of 33 pages, you are referring to a paragraph that doesn't have SIMD, but he talks about SIMD. It is clear to a POSITA that it is SIMD. So you can take whatever you want from that paragraph. I disagree with that.

BY MR. TUMINARO:

Q. Switching gears --

A. My resume.

Q. Earlier we talked about DX10 and open GL or the current version of DX and the current version of open GL and we agreed that they require a unified shader?

MR. PLUTA: Objection, form. Objection, relevance.

Page 137

BY THE WITNESS:

A. From the document, you provided and I read, that's what it says.

BY MR. TUMINARO:

Q. And do you also remember that now?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. I believe that at the time we were looking at the DX and open GL, they were referring, but I don't know the exact year of the open GL we looked at. They transitioned to this as you can see. So I cannot say for certain that when I looked at open GL for another thing.

BY MR. TUMINARO:

Q. Okay. So a GPU today, as of today, to be fully compatible with the current version of DX and the current version of open GL, that GPU would have to provide a unified shader, right?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. I could not tell you that for sure. Can there be a design around where you do it without a

Page 138

unified shader? Anything is possible.

Q. Well, I'm asking to be fully compliant with those APIs, the GPU would have to provide a unified shader, to be fully compliant?

MR. PLUTA: Objection, form.

MR. TUMINARO: Not a design around.

MR. PLUTA: Objection. Objection, form. Objection, relevance.

BY THE WITNESS:

A. It's hard to say, Counsel. It's hard to say.

BY MR. TUMINARO:

Q. Why is it hard to say?

A. Because you could design it that it is interacting with the APIs the DX10 and 11 and open GL and still not completely a unified shader. It's hard to say. You have to design it. I cannot answer that the way you are asking me.

Q. What do you mean by completely a unified shader? What is the qualifier "completely" mean?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. A unified shader has to be a fully

Page 139

programmable solution. There might be ways of doing this without adhering to that objective. Yeah. That's basically what I have to say. I cannot answer the question without more analysis.

BY MR. TUMINARO:

Q. I guess what I'm hung up about, if the GPU has to satisfy everything that's in those APIs, and one of things that's in the API is a unified shader, how is it that the GPU could not provide a unified shader?

MR. PLUTA: Object, form. Objection, relevance.

BY THE WITNESS:

A. This is outside the scope of what I worked on. So I will not be able to give you a very accurate and professional answer. It requires more work. I would be glad to work on that and get back to you.

BY MR. TUMINARO:

Q. Because you didn't provide any opinion on objective indicia of nonobviousness?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. Now you're wrapping a legal issue on

Page 140

that. I cannot answer. Sorry.

BY MR. TUMINARO:

Q. You don't know one way or the other if you provided an opinion on objective indicia of nonobviousness?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. I would be glad to look at my report. I have prior art. I have other items that I've discussed.

BY MR. TUMINARO:

Q. Take a look at your report. Tell me. Did you provide an opinion on objective indicia of nonobviousness?

A. I don't believe I have.

MR. PLUTA: We've been going about an hour and a half. Is this a good breaking point?

MR. TUMINARO: Sure. We can take a break.

THE VIDEOGRAPHER: We're going off record. The time is 1:49 p.m.

(Whereupon, a short break in the proceedings was taken.)

THE VIDEOGRAPHER: We're back on record.

Page 141

This is the beginning of Media 3. The time is 2:12 p.m.

BY MR. TUMINARO:

Q. Would you turn with me to Lindholm reference. I forget what exhibit number it is.

MR. PLUTA: 6.

MR. TUMINARO: Thank you.

BY MR. TUMINARO:

Q. Would you circle what you're calling the arbiter? What in Lindholm corresponds to the arbiter?

A. Which figure?

Q. Whatever figure you think best depicts the arbiter.

MR. PLUTA: Object to the form.

BY MR. TUMINARO:

Q. Would you write arbiter next to it?

A. (Witness marking document.)

Q. Would you circle for me what you're saying corresponds to the claim claimed plurality of command processing engines from the '053 patent -- that was a terrible question. Let me start again.

Would you please look at the Lindholm reference and circle in Lindholm what you say

Page 142

corresponds to the claimed plurality of command processing engines?

A. (Witness marking document.)

Q. Just to be clear, what figure are you marking up?

A. Figure 4.

Q. Figure 4 of Lindholm?

A. Yes. I think so, yes.

Q. Okay. Would you turn with me to the Morton reference, which is Exhibit 4?

A. Are we done with this?

Q. You can put it aside.

A. Yes.

Q. Sorry. I didn't realize you were there. I apologize.

Would you circle for me in Morton what corresponds to the claimed plurality of command processing engines?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. He talks about this being one slice or multiple slices, but at least figure 1 shows one slice. I'm going to make the assumption that -- so as I pointed out, it's 140 through 170. These are

Page 143

the graphics processors.

BY MR. TUMINARO:

Q. Would you write next to that command processing engines.

A. (Witness marking document.)

Q. Just so we're clear, you're marking figure 1 of Morton?

A. Yeah.

Q. Claim 1 of the '053 patent recites a first portion and a second portion for the memory device.

A. Yes.

Q. Would you circle the first portion and the second portion that you say -- strike that.

Would you circle in Morton what you say corresponds to the claimed first portion and second portion from the '053 patent?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. So referring to column 15, 48 through 67. Also referring to the claim construction page -- chart -- sorry -- my claim construction chart, figure 5 shows a conceptual diagram of a memory resource 200 containing memory spaces that are

Page 144

reserved and managed according to an alternative memory space reservations process. Figure 5 is described in relation to figures 1 and 2. Memory resource 200 of figure 5 contains each element shown in figure 2. Memory resource also contains a second memory section 520 having at least two memory spaces 505 for use by threads of second set of threads executing on graphics processor 125. All threads of the first set of threads are of the first thread type and are each reserved in memory space 205.

So basically memory space 205 is used for the first set of threads and then on memory space 505 is for the other type. So, this is referring to this passage. Memory space 505 having a second memory space second thread type being different than the first thread type.

BY MR. TUMINARO:

Q. Is memory space 505 on figure 5?

A. That's what it's referring to in this section of the document.

Q. Is that one of the -- is that the first portion or the second portion?

A. Second memory thread type.

Q. Would you circle that then?

Page 145

A. It's done.

Q. Would you label it?

A. (Witness marking document.)

Q. Thank you.

It was figure 5 that you marked up, just so the record is clear?

A. Yeah.

Q. Would you turn with me to Paragraph 131 of your declaration. I'll read for the record with respect to claim 7. "Because Stuttard also discloses that the processing elements 1061 perform 'lighting and shading' functions, the processing elements 1061 must necessarily include a texture processing engine."

Do you see that?

A. Yes.

Q. Is it your position that lighting functions necessarily require texture processing?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. It speaks for itself. Lighting and shading functions must necessarily include the texture processing engine. I cannot add anything more than what is in here.

Page 146

BY MR. TUMINARO:

Q. Isn't it true, though, that a shading function can be performed without texture processing?

A. In my opinion, predominantly, this is how it is done.

Q. Predominantly, but it is possible that a shading function can be performed without texture processing, right?

A. No. In my opinion, that's the way it's done.

Q. So shading functions have to include texture processing?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. It says, must necessarily include the texture processing engine. So that's clear what it says: Must necessarily.

BY MR. TUMINARO:

Q. Is it possible that a shading function can be performed, and in doing that shading function, there's no texture processing? Is that possible?

MR. PLUTA: Objection, form.

Page 147

BY THE WITNESS:

A. It is my opinion that a texture processing engine must include lighting and shading.

BY MR. TUMINARO:

Q. I can read your declaration. My question is a little bit different. Is it possible that a shading function can be performed; and in doing that shading function, there's no texture processing?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. I mean, somebody could design something, but it is my opinion that that should include lighting and shading.

BY MR. TUMINARO:

Q. Is it possible that a lighting function can be performed that does not involve any texture processing?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. It is my opinion that lighting and shading should be part of the texture processing.

Page 148

BY MR. TUMINARO:

Q. Can you answer my question?

MR. PLUTA: Objection, form. Objection, relevance. Asked and answered.

BY THE WITNESS:

A. I did the best I could.

(WHEREUPON, document marked as Exhibit No. 10.)

MR. PLUTA: I'm going to object to the relevance of this, the introduction of this exhibit into evidence.

BY MR. TUMINARO:

Q. This is Exhibit 10. You have been handed what has been marked as Exhibit 10 for identification purposes. This paper is titled Design and Implementation of a Rendering Algorithm in a SIMD Reconfigurable Architecture (Morphosys).

Do you see that?

A. Yes.

Q. In the listing of inventors there is one Nader Bagherzadeh. Do you see that?

MR. PLUTA: Objection, form.

BY THE WITNESS:

A. Not correct. We're -- this is an article. It's not a patent. So it's not an invention.

Page 149

BY MR. TUMINARO:

Q. Sorry. I said inventors? Sorry.

A. We don't consider ourselves inventors.

Q. One of the authors listed on Exhibit 10 is an individual named Nader Bagherzadeh?

A. Yes.

Q. Is that you?

A. Yes.

Q. You're one of the authors on this paper?

A. Yes.

Q. If you would turn with me to the third page. It is not numbered. The third page of text.

On the right-hand side of the page, right hand column, there's a formula, which is the first line on the right-hand column. It says Id equals I and it goes on. Do you see that?

A. Yes.

Q. If you look to the left-hand column in the last full paragraph, it starts with "in order to obtain."

Do you see that?

A. Yes.

Page 150

Q. I'll read it. "In order to obtain an image with enough realism, we apply to each pixel a Gouraud shading algorithm." Did I say that correctly?

A. Gouraud shading.

Q. Can you explain what Gouraud shading is?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. It is says we calculate the light intensity for each triangle, basically like perpendicular to the surface of a vertex to that other primitive.

BY MR. TUMINARO:

Q. A Gouraud shading, does that include texture mapping?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. It could. But in this case, we didn't talk about it.

BY MR. TUMINARO:

Q. In this case, does it include texture mapping?

Page 151

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. Yeah, in this case, we did not talk about texture mapping, but we could have. It was just not part of the design. It would have been appropriate to add that as well.

BY MR. TUMINARO:

Q. But it is a fact that in this shading algorithm there is no texting mapping?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. What it is is that it gives you an opportunity to find Gouraud shading, but a box that does texture -- what did we call it? A texture processing engine would include this particular function as well. So -- so, this is part of the texture mapping engine.

BY MR. TUMINARO:

Q. This is your paper and you talk about Gouraud shading, right? And this Gouraud shading that you talk about in your paper, it is an example where a shading function does not include texture

Page 152

mapping, right?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. The function that is responsible for lighting and shading would be called texture processing engine, and this would be part of that. But we did not call it that much because this was a limited time. But an expanded version of this should have a texture processing engine named and included this function.

BY MR. TUMINARO:

Q. Okay. There might be an expanded version, but yes or no, this shading function that's disclosed in your own paper is an example of a shading function that does not include texture mapping?

MR. PLUTA: Objection, form and relevance.

BY THE WITNESS:

A. This function is part of a texture processing engine, the box responsible for that.

BY MR. TUMINARO:

Q. Is it your testimony that this Gouraud

Page 153

shading that's described in your paper necessarily occurs in a texture processing engine?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. It belongs necessarily to a texture processing engine, yes.

BY MR. TUMINARO:

Q. Is that how it was implemented in the example that you described in your paper?

MR. PLUTA: Same objections.

BY THE WITNESS:

A. I think we basically simulated a lot of stuff here. We did not implement this as far as I remember. This was just a paper analysis.

BY MR. TUMINARO:

Q. Did this paper analysis include the simulation of a texture processing engine?

MR. PLUTA: Objection, form. Objection, relevance.

BY THE WITNESS:

A. It is assumed that that is part of the texture processing engine, yes.

MR. TUMINARO: We have no more questions

Page 154

at this time.

MR. PLUTA: Let's go off the record.

THE VIDEOGRAPHER: Going off record. The time is 2:32 p.m.

(Whereupon, a short break in the proceedings was taken.)

THE VIDEOGRAPHER: We're back on record. The time is 3:06 p.m.

EXAMINATION

BY MR. PLUTA:

Q. Welcome back. I just have a couple of questions for you.

Do you recall the discussion about VHDL and in specifically in relation to Exhibit 3 --

A. Yes.

Q. -- your paper?

A. Yes.

Q. Now, putting aside my objections to the paper, I have a couple of questions for you on that topic. You recall the questions counsel asked you about building a chip, correct?

A. Yes.

Q. You also mentioned, I'm paraphrasing your testimony here, that to you building a chip means

Page 155

fabrication of a chip, correct?

A. Right.

Q. You also mentioned a process of massaging during the process of designing a computer chip in particular after simulation of it?

A. Yes.

Q. And is it true that you testified that there's a lot of trial and error involved in the design of a computer chip?

A. Yes.

Q. I also like to direct you now to Lindholm, which is Exhibit 6, just for reference, and then I'm also going to take you to Paragraph 72 of your declaration.

A. Yep.

Q. Do you recall questions about the thread control buffer 420?

A. Yes.

Q. Is your opinion that the thread control buffer is the only element that corresponds to the claimed memory device?

A. No.

Q. What else are you pointing to as corresponding to the claimed memory device?

Page 156

A. Instruction cache, which is Box 410.

Q. Let's look at another document, the petition, which is Exhibit 9. Do you recall Counsel took you to Page 13 I believe of the --

A. Okay.

Q. -- petition? Do you recall that counsel asked you about Page 13 and regarding the thread control buffer 420?

A. Correct.

Q. And you recall counsel asked you whether you agree with petitioner's statement and you said -- and I'm paraphrasing here -- you agree with it because it includes the thread control buffer 420, right?

MR. TUMINARO: Objection, leading.

BY MR. PLUTA:

Q. Do you recall your testimony regarding Page 13 of the petition?

A. Yes.

Q. And counsel asked you about the position with respect to I believe it's the first paragraph of Page 13?

A. Okay.

Q. And you agreed that the memory device

Page 157

includes the thread control buffer, correct?

A. Correct.

Q. And you just testified that you also believed that it also includes the instruction cache, correct?

A. Correct.

Q. I'd like to turn to Page 22 of the petition and direct your attention to the second full paragraph on that page. Could you read that second full paragraph, please, for the record?

A. "As discussed above, Lindholm discloses instruction cache 410 and thread control buffer 420 which corresponds to the claim at least one memory device."

Q. Now, after reading that, is it your position that the thread control buffer in combination with the instruction cache corresponds to the claimed memory device consistent with the petitioner's petition here in this case?

A. Yep. This is exactly what I had in mind.

Q. Let's switch one more topic.

A. Okay.

Q. Do you recall the questioning regarding the ALU resource and texture fetch resource

Page 158

divisions with respect to the background of the '053 patent?

A. Yes.

Q. The command threads in the ALU resource division and texture fetch resource divisions are different types of threads, correct?

A. Correct.

MR. TUMINARO: Objection, leading.

BY MR. PLUTA:

Q. Are the command threads in the ALU resource division and the texture fetch resource division different types of threads?

A. They are.

Q. Are pixel command threads and vertex command threads also different types of threads?

A. Yes.

MR. PLUTA: No further questions at this time.

EXAMINATION

BY MR. TUMINARO:

Q. I have questions. Did you talk with your counsel about the substance of your testimony after I said I had no further questions at this time?

A. Absolutely not. They did not even say

Page 159

hello to me, which I was disappointed.

MR. TUMINARO: I have nothing else.

THE VIDEOGRAPHER: This concludes the videotaped deposition of Nader Bagherzadeh. The time is 3:12 p.m. We're off record.

(Witness excused at 3:12 p.m.)

Page 160

REPORTER CERTIFICATE

I, JO ANN LOSOYA, a Certified Shorthand Reporter within and for the County of Cook and State of Illinois, do hereby certify:

That previous to the commencement of the examination of the witness, the witness was duly sworn to testify the whole truth concerning the matters herein;

That the foregoing deposition transcript was reported stenographically by me, was thereafter reduced to typewriting under my personal direction and constitutes a true record of the testimony given and the proceedings had;

That the said deposition was taken before me at the time and place specified;

That I am not a relative or employee or attorney or counsel, nor a relative or employee of such attorney or counsel for any of the parties hereto, nor interested directly or indirectly in the outcome of this action.

Page 161

IN WITNESS WHEREOF, I do hereunto set my hand this August 15, 2015.

____

JO ANN LOSOYA, CSR
C.S.R. No. 84-002437

Page 169

Page 177

Page 178

United States District Court
For the Northern District of California

IN THE UNITED STATES DISTRICT COURT

FOR THE NORTHERN DISTRICT OF CALIFORNIA

UNIRAM TECHNOLOGY, INC,

Plaintiff,

v

TAIWAN SEMICONDUCTOR
MANUFACTURING COMPANY, LTD and
TSMC NORTH AMERICA,

Defendants.
_______________________________/

No C 04-1268 VRW

FINDINGS OF FACT AND
CONCLUSIONS OF LAW

This is a patent dispute between UniRAM Technology, Inc (“UniRAM”) and Taiwan Semiconductor Manufacturing Company Ltd and TSMC North America (collectively “TSMC”). UniRAM sued TSMC for patent infringement and misappropriation of trade secrets, among other claims. TSMC counterclaimed that one of UniRAM’s patents was unenforceable due to inequitable conduct in front of the Patent and Trademark Office.

On January 14 and 15, 2008, the court heard testimony concerning TSMC’s allegation that UniRAM committed inequitable conduct in the prosecution of US Patent No 6,108,229 (“the ‘229 patent”). The court heard live testimony from Bo-In Lin, Jeng-Jye

Shau, David Taylor, and Carl Sechen, and deposition testimony from Richard Killworth and Peter Gillingham. That testimony supplemented testimony heard by the court from September 9 to 21, 2007, at the jury trial of UniRAM's trade secret misappropriation (and related) claims.

FINDINGS OF FACT

1. United States Patent 6,108,229 ("the '229 Patent," Ex. 353) issued on August 22, 2000, from United States Application Serial No 09/114,538 ("the '538 Application," Ex. 5133), which was filed on July 13, 1998. The '538 Application was filed as a continuation-in-part ("CIP") both of United States Application Serial No 08/805,290 (the "'290 Application," Ex 5132) and United States Application Serial No 08/653,620 (the "'620 Application," Ex 5109).

2. The '620 Application was filed on May 24, 1996, and issued as United States Patent No 5,748,547 ("the '547 Patent") on May 5, 1998. Ex 7.

3. The '290 Application was filed on February 25, 1997, as a continuation-in-part of the '620 Application, and issued as United States Patent No. 5,825,704 ("the '704 Patent") on October 20, 1998. Ex 9.

4. All three patents name Dr Jeng-Jye Shau as the sole inventor. Exs 7, 9 & 353.

5. Dr Shau's native language is Mandarin Chinese. Although he can communicate in English, Shau's English is imperfect. Doc #621 85:15 to 86:10. TSMC's expert stated that Shau's "patent

obviously is written by someone who doesn’t have a great command of English.” Doc #621 154:24 to 155:15.

6. Shau is an integrated circuit designer. Jury Trial 146:23-24.

7. The person of ordinary skill for purposes of Shau’s patents is an integrated-circuit designer, and not an integrated-circuit manufacturer. Doc #621 134:4-9; 175:2 to 176:11. TSMC’s own expert, Mr Taylor, defined a person of ordinary skill as having “a Bachelor’s degree in electrical engineering and three to five years of experience in the *design* of semiconductor memory products, preferably DRAM products.” Doc #621 134:7-9 (emphasis added). TSMC’s expert therefore defined a person of ordinary skill with respect to experience and training in circuit design.

8. The work of an integrated-circuit designer is performed almost entirely on a computer. Circuit designers make extensive use of software-based circuit simulation, verification and layout tools. Doc #621 96:13 to 97:7; 101:9-12; 166:10-12; 177:9-22; 180:19 to 181:2.

9. The entire integrated-circuit design industry relies heavily on software simulation tools to predict accurately how a circuit will perform once manufactured. Doc #621 177:23 to 178:4; 188:5-18.

10. As a general practice, circuit designers simulate their designs extensively before having their circuit designs fabricated into a physical integrated-circuit, commonly called a “chip.” Doc #621 188:9-16.

11. A circuit designer’s product is his design, and his or

her final product is an electronic design file known as a "tape out" file. Doc #621 181:16-20; 190:3-17. Even TSMC sometimes refers to design files as products. Doc #621 191:4-24. The tape out file serves as a blueprint from which the entirety of a design's details may be discerned and from which a circuit manufacturer can fabricate a chip. Jury Trial 177:8-12.

12. Once the design stages are completed - including simulations, verifications, and the creation of a tape out file - the design in the form of a tape out file is generally submitted to a circuit fabrication facility, commonly termed a "fab." Doc #621 181:12-15.

13. Whereas the final product of a circuit designer is a tape out and a circuit design, the final product of a fab, or foundry, is a physical chip.

14. Although circuit design and circuit manufacturing are part of the overall integrated-circuit production process, they are typically separate and distinct activities. Generally, circuit designers do not fabricate chips, and generally foundries do not design tape out files. Doc #621 181:16-24. Circuit designers and manufacturers by and large employ distinct vocabularies. Doc #621 177:1-5.

15. Integrated circuits from the point of view of a circuit-designer are under production when the circuit designer begins computer layout work on his or her design. This includes computer-simulations of the anticipated performance of the design, verification on a computer of the design's compliance with foundry-specific "design rules," creation of a tape-out file on a computer and continues until the tape out file is submitted to the

fab and sometimes continues therein and thereafter during an iterative refinement process. Doc #621 177:6 to 182:3.

16. UniRAM presented Dr Carl Sechen as an expert on how a circuit-designer with an appropriate background would interpret certain of Shau's patent statements that TSMC has challenged in the instant case. Dr Sechen holds a PhD from the University of California at Berkeley, is a fellow in the IEEE, has authored 150 publications in the field of integrated circuit design and has taught at Yale University, the University of Washington and the University of Texas at Dallas (where he is currently a full professor). Doc #621 170:13 to 172:21.

17. TSMC's expert on how one of ordinary skill would interpret certain of Shau's patent statements, Mr David Taylor, is less qualified than Dr Sechen. Mr Taylor lacks a PhD, is not a fellow in the IEEE and lacks the teaching and research experience of Dr Sechen. Ex 5291.

18. Mr Taylor, unlike Dr Sechen, cited no documents in either his expert report or his trial testimony to corroborate his opinions. Doc #621 152:3-20.

19. Dr Sechen consistently offered opinions that were more specific and complete than the answers given by Mr Taylor. Dr Sechen's opinions were supported by reference to objective standards in the industry, such as terms of art. Mr Taylor, by contrast, relied more often on his view of common sense and his personal interpretations of disputed phrases. Dr Sechen explained his interpretations of disputed terms in greater detail, usually by a more thorough consideration of the context in which the term appears.

20. In May 1996, Shau began developing an integrated-circuit design that included a dynamic random access memory ("DRAM") architecture. Doc #621 88:22 to 89:2.

21. In May 1996, Shau began layout work on a tape out file using a software program named MAGIC and completed simulations using software programs named SPICE and RSIM. Doc #621 89:20 to 90:1; 90:10 to 91:3; Jury Trial 161:1 to 162:2.

22. The simulations reflected that the "access time" for Shau's DRAM architecture using 0.6 micron ("μm") technology design rules was 4 nanoseconds ("ns"). Doc #621 55:11-13; 72:17-24.

23. After confirming the viability of his DRAM architecture by simulation, and while he continued work on the tape out, Shau began drafting the '620 Application. Doc #621 91:11-19.

24. Shau is not a lawyer, and he drafted the '620 Application by himself, without assistance from counsel. Doc #621 88:11-21; Jury Trial 156:19-24.

25. The '620 Application as drafted included the following passage:

> Our results show that a memory of the present invention is faster than an SRAM of the same memory capacity.

Ex. 5109, at 5109-014. The court finds this statement was not false or misleading.

26. At the time Shau made this statement, no physical "chip" had yet been fabricated. The '620 Application would not, however, suggest to a person of ordinary skill in the art that any physical semiconductor chips had been fabricated. Instead, this statement about "results" refers to simulations conducted on Shau's design.

Doc #621 202:11-18.

27. Circuit designers simulate their designs regularly. Doc #621 94:5-10. UniRAM's expert stated that "[t]he whole [integrated circuit] industry relies extensively on simulation results to accurately predict the performance of the design." Doc #621 188:12-14. TSMC's expert stated that simulations provide "realistic predictions of just what the performance will be" once the design is fabricated. Doc #621 138:14-19.

28. From the perspective of a circuit designer, the term "results" in this sentence refers to simulation results that pertain to a design file, not results from testing a physical chip. Doc #621 93:20 to 94:10; 186:18 to 187:11.

29. The challenged statement refers to "results" only, not "measured results." The choice of the word "results" is important because the phrase "measured results" is a term of art in the semiconductor industry. Doc #621 187:2-4. When persons of ordinary skill in the art discuss test results on a physical, fabricated chip as opposed to test results on a design, they often use the phrase "measured results." Doc #621 187:2-11. Creators of integrated circuits "get a lot of accolades for actually producing a part and having measured results." Doc #621 187:6-8. The modifier "measured" in the phrase "measured results" is a very "prestigious and important qualifier" on the word "results." Doc #622 231:7-10. Accordingly, if and only if a circuit designer has a physical chip, he will use the term of art "measured results" to describe his test results. The absence of that term of art suggests strongly that the designer does not have a physical chip. Doc #621 187:9-11. Because Shau never in May 1996 made any

United States District Court
For the Northern District of California

reference to "measured results," a person of ordinary skill would not read his statements to suggest the existence of a physical chip. Doc #621 186:18 to 187:11.

30. The Figures included in the '547 patent prove to one skilled in the art that the application discussed simulations and not actual chips. Figures 7a, 7b, and 7c in the '547 patent include artificial curves, input labels such as "BLKSEL" or "KWL" and output labels such as "BL" or "BL#" that a circuit designer would understand come from the SPICE simulation program and not a physical chip. Doc #621 91:9 to 93:19.

31. Circuit designers would also understand Figure 7a necessarily to relate to simulation results and not physical chip measurements. Figure 7a includes data on bitline outputs. It is not possible, however, to measure the bitline output of a physical chip. Accordingly, a circuit designer reviewing Figure 7a would know that the author of the patent had to have been reporting simulation results. Doc #621 98:15 to 99:4.

32. Similarly, the absence in the patent of any "chip photomicrographs" suggests that a person of ordinary skill in the art would infer that no physical chips had been produced. Doc #621 187:12 to 188:4. A chip photomicrograph is essentially a photograph that shows the details of the actual fabricated silicon chip. 187:16-22. Without such evidence, "no one would believe you have actually produced an actual chip." 187:23-24. Shau's patent did not include any reference to a chip photomicrograph. 187:25 to 188:4.

33. The '620 Application includes the following passage:

> Although the bit line structure in FIG. 3b is the actual bit line structure used in our product, for simplicity, we will use the simpler two-dimensional bit line structure in FIG. 3a as example in the following discussions.

Ex. 5109, at 5109-020. The court finds this statement was not false or misleading.

34. At the time Shau made this statement, no physical "chip" had yet been fabricated. A person of ordinary skill in the art, however, would also not interpret this statement to suggest that a physical "chip" had been manufactured. TSMC does not offer credible evidence to the contrary.

35. The audience of the patent is a circuit designer, and the subject matter of the patent is a circuit design. For circuit designers, the final product is a circuit design and a tape out file. As UniRAM's expert stated, because the person of ordinary skill in the art for the '620 Application is a circuit designer, the term "product" logically can mean a circuit design or a tape out file. Doc #621 190:3-17.

36. TSMC's expert stated that to a person of ordinary skill in the art, "product" refers to a chip that is "well beyond a test chip" and is "working to meet all specifications and is something that is available to be commercialized and sold." That testimony is less credible than the testimony of UniRAM's expert because the context and subject matter of the patent is not a physical chip that has been tested, but rather a design that has undergone simulations as described above.

37. There is no credible evidence that the statement "Although the bit line structure in FIG 3b is the actual bit line

structure used in our product, for simplicity, we will use the simpler two-dimensional bit line structure in FIG 3a as example in the following discussions" was false or misleading.

38. The '620 Application includes the following passage:

> A memory device of the present invention is under production. Using 0.6 micron technology to build a memory array containing one million memory cells, we are able to achieve 4 ns access time, which is more than 10 times faster then existing memories devices of the same storage capacity.

Ex. 5109, at 5109-026. The court finds this statement was not false or misleading.

39. TSMC offers no credible evidence that the term "memory device" refers necessarily to physical fabricated chips. To the contrary, TSMC's expert stated that there is no "'nice, clear-cut totally clear definition of what a memory device is.'" Doc #621 154:6-23.

40. UniRAM's expert testified that in the context of the '229 patent, "memory device" refers to an integrated circuit – which can exist as a design on a computer – and is completely different from a "physical integrated circuit" or a "physical chip." #621 222:7-23.

41. UniRAM's expert stated that a memory device can exist "way before fabrication," including "in the design stage on a computer." #621 225:12-14. He specifically rejected the idea that a memory device refers to a physical chip. #621 222:16-23 and 220:10-11. In the context of the patent at issue, which discussed circuit design, it was "inconceivable" that the author was referring to a physical chip. Doc #622 226:4-10.

42. UniRAM's expert also testified that the phrase "device has been fabricated" is a term of art. Doc #621 183:16-20. Without that term of art, a person of ordinary skill in the art would be "convinced beyond any doubt that nothing has been actually fabricated." Doc #621 184:1-3. That term of art does not appear in the patent. Doc #621 183:16 to 184:3.

43. TSMC's only evidence that "memory device" refers necessarily to physical fabricated chips is its expert's statement that "I think in my view, it's pretty clear that when it says a device, it means something physical." Doc #621 152:24-25. He did not connect his interpretation to any terms of art or industry standards.

44. TSMC offers no credible evidence that the term "under production" means necessarily that physical chips are being fabricated for sale.

45. Unlike "under production," the phrase "in production" is a term of art in the semiconductor industry that "means that products intended for sale are actually being fabricated and produced." Doc #621 192:5-15; 182:22 to 183:4. UniRAM's expert compared having chips "in production" to producing automobiles on an assembly line in Detroit. Doc #622 215:6-24.

46. By contrast, the phrase "under production" does not have any widely understood specific meaning. Doc #621 183:1-4. In fact, because the phrase "in production" is a term of art referring specifically to mass production of chips, the use of the phrase "under production" suggests that physical chips are not being fabricated. Doc #622 215:6-24.

47. Because the meaning of "under production" is open-ended, the phrase should be interpreted in context. Doc #622 212:3-7. The context of the statement in the patent is circuit design, and thus the phrase "under production" refers to "the steps of the design process, including design, simulation, verification and on to the tapeout." Doc #621 183:5-12.

48. In addition, other assorted terms of art in the industry that also refer to the physical production of chips, such as "fabricated" or "manufactured," do not appear in the patent. Doc #621 183:16 to 184:3. The absence of any terms of art referring to physical production of chips suggests strongly that a person of ordinary skill in the art would not interpret "under production" as referring to the physical production of chips.

49. TSMC's expert testified that "under production" refers to the process of producing thousands of copies of a chip to test its viability before moving on to high-volume mass production, which runs in the millions of copies. Doc #621 139:25 to 141:16. His description of this pre-production process may be an accurate statement of production practices in the industry, but he offers no testimony that the specific term "under production" refers necessarily to the specific pre-production process he describes. He did not state that "under production" is a term of art, and he did not discuss whether his interpretation of "under production" is the same for the purposes of manufacturing processes and for the purposes of disclosures in a circuit design patent.

50. In the specific context of design disclosures in a circuit design patent, the use of the phrase "under production" to mean fabrication would be "highly unlikely." Doc #622 217:9-16.

This is because the usual practice for designers is to obtain a valid patent on a design <u>before</u> proceeding to manufacture and sale of the invention. Doc #622 215:19 to 216:15.

51. Because a circuit designer's final product is a circuit design and tape out, the statement in the '620 Application that a "memory device" was "under production" is consistent with a circuit design being in progress and the tape out being under production. Shau's design was "under production" as long as he was working on the layout of the chip design, including simulations. Doc #621 182:9 to 184:3; Jury Trial 269:1-19; 270:12-15. At the time the '620 Application was filed on May 24, 1996, most if not all of Shau's simulation work was finished and his layout work had begun. Doc #621 89:20 to 90:1. When he filed the '620 Application, Shau had completed the entire layout for the memory device that was claimed (but had not completed the external logic circuitry design). Doc #621 44:6-22. Shau's tape out, which is a circuit designer's product, was thus "under production" as of May 24, 1996.

52. The passage in the '620 Application "[u]sing 0.6 micron technology to build a memory array containing one million memory cells" referred to Shau's process of designing an array of a million memory cells and implementing the design with 0.6 micron technology design rules, not to a physical chip. Doc #621 95:9 to 96:8.

53. The reference to a "memory array" further shows that Shau was referring to a design and not a physical chip because an "array" is not a physical chip from a circuit designer's point of view. Doc #621 95:9 to 96:1; Jury Trial 271:8-15. Shau demonstrated a "memory array" for the jury on his computer -

navigating a software file, not a physical chip - while explaining his USRAM tape out. Jury Trial 188:20 to 189:3.

54. The reference to "build[ing]" an array also does not suggest necessarily a physical chip, because circuit designers - those of ordinary skill in the art - use the infinitive "to build" to discuss their work on design files, performed on a computer. Doc #621 96:13 to 97:10; 184:15 to 185:1; Jury Trial 271:19 to 272:8. The word "build" is not a term of art among circuit designers. Doc #621 184:21-22. Even if "build" refers commonly to physical structures, TSMC offers no evidence that "build" cannot be used as a metaphor for other purposes. According to Webster's Third International Dictionary, for instance, one can "build" an argument, a work of art or a piece of literature. <u>Webster's Third New International Dictionary</u> 291-92 (1981).

55. Shau's simulation results at the time of the '620 Application reflected a 4 ns access time, which was at relevant times more than ten times faster than traditional DRAM memories of the same size. Doc #621 94:2-17; 55:24 to 56:3; 97:18 to 98:3.

56. The reference to "4 ns" access time also would indicate to a person or ordinary skill in the art that Shau was referring to a design - and not a physical chip - because circuit designers use more than one digit of precision when referring to a physical, tested chip (for example, 4.x nanosecond access time) rather than a design (for example, 4 ns access time). Doc #621 185:25 to 186:17. Accordingly, Shau was not claiming to achieve a specific result, as he would if he were referring to a physical chip. TSMC's expert did not address Shau's use of one rather than two significant figures in his patent.

57. Shau was not referring to Sunaga prior art when he claimed that his invention was ten times faster than existing prior art. Doc #621 55:24 to 56:2; 39:16 to 44:5.

58. The reference to "one million memory cells" further indicates that a person of ordinary skill in the art would not read this patent language to suggest the presence of a physical chip. In the industry, memory cells in finished chips are always counted as a power of two, and one million is not a power of two. Accordingly, a person of ordinary skill in the art would interpret Shau's imprecise description as suggesting that a design is being discussed. Doc #621 185:2-24.

59. In the summer of 1996, Shau completed his first tape out and submitted it to MOSIS. Doc #621 102:10-15.

60. MOSIS (not to be confused with MoSys, a former defendant in this case) is a relatively low-cost semiconductor manufacturing service often used for academic or research purposes. Doc #621 102:2-9.

61. Shau realized that MOSIS could support only small test chips, so his original million memory cell array was reduced in size for the MOSIS tape out. Doc #621 107:23 to 108:14. Ex. 1378.

62. The MOSIS design was a test chip with a simple interface for testing purposes only. It was not configured as a commercial product. Doc #621 103:24 to 104:15.

63. MOSIS sent Shau manufactured and packaged chips in late 1996. Doc #621 102:10-24.

64. Shau's initial tape out to MOSIS had bugs, but the bugs related to the logic portions of the tape out. The bugs did not relate to Shau's DRAM architecture, the subject of the claimed

inventions of Shau's patents. Shau was able to bypass the bugs to determine that the architecture of his design worked. Doc #621 102:10-21. The MOSIS tape out was successful. Doc #621 102:10-17.

65. Shau's success in verifying his memory architecture using the MOSIS prototype is reflected by his contemporaneous statement to TSMC in December 1996 that the "USRAM architecture has been verified on silicon by a 8K x 9 device." Ex 443 at 0041660.

66. Shau wished to verify his DRAM architecture on a full-size chip. Shau decided to approach TSMC, the world's largest independent commercial foundry, to begin commercial fabrication activities on his DRAM architecture designs. Doc #621 102:22 to 103:9.

67. In the latter part of 1996, pursuant to a nondisclosure agreement, Shau disclosed his DRAM architecture to TSMC and secured permission from TSMC to submit a tape out as a paying customer. Doc #621 103:10-17; Jury Trial 162:20 to 164:7.

68. In late 1996, Shau sent TSMC a tape out file. Shau's tape out to TSMC employed the same general DRAM architecture as the MOSIS tape out but was modified significantly. Doc #621 103:24 to 104:7. For example, the chip that Shau taped out to TSMC was much larger and had a more complex interface. Doc #621 104:2-7; 108:3-14.

69. Unlike his earlier MOSIS tape out, the design embodied in Shau's first tape out to TSMC was a test chip fully compatible with a popular commercial chip at the time, called CacheRAM. Doc #621 104:13-15; 106:16 to 107:8.

70. TSMC successfully completed the physical fabrication of Shau's first TSMC tape out in January 1997 and sent him a small number of test chips. Jury Trial 211:6 to 211:24.

71. Shau's initial TSMC tape out had bugs. However, the bugs were outside the portion of the tape out corresponding to Shau's DRAM architecture that is the subject of the claimed inventions of Shau's patents. Doc #621 109:17-25. Shau was able to bypass those errors for purposes of establishing that he could read and write data to the memory cells in the memory array. This meant that his memory array design was working properly. Doc #621 109:2 to 111:16; Jury Trial 211:25 to 213:1-18; 469:3-6; 214:17-22.

72. On February 25, 1997, Shau filed the '290 Application. The '290 Application included the following additional text in its specification:

> **Using this memory cell 1400 and a memory architecture disclosed in this invention and in our previous patent application, commercial memory products were manufactured successfully. The major advantage of the logic memory cell 1400 is that it can be manufactured using standard logic technology. The resulting memory product achieved unprecedented high performance.**

Ex. 5132, at 5132-035.

73. As of the time of the filing of the '290 Application, physical chips corresponding to Shau's design had been fabricated successfully by TSMC. Doc #621 111:9-16; 116:21 to 117:2.

74. Because a physical semiconductor chip had been fabricated by February 1997, Shau included text in the '290 Application to reflect that a product had been "manufactured," whereas the term "manufactured" had not previously been employed in the '620

Application. Compare Ex 5109, with Ex 5132; see also Doc #621 183:23 to 184:9.

75. Because he had actually achieved (past tense) measured results from a manufactured chip, Shau used the term "achieved" in the '290 Application - rather than the present tense term "we are able to achieve" as was used in the '620 Application in reference to simulation results. Compare Ex 5109, at 5109-026, with Ex 5132, at 5132-035. The statement that the chip achieved unprecedented high performance was accurate. Doc #621 72:7; 72:22-24; 114:11-14.

76. The initial TSMC test chip was a "commercial" chip in the sense that it was configured as a commercial CacheRAM chip and was manufactured by a commercial foundry (TSMC) rather than an academic foundry (MOSIS). Doc #621 104:13-15; 106:16 to 107:8; 112:22-25. It was also a "commercial" chip in the sense that Shau intended it as a profit-making vehicle. Doc #621 49:19 to 50:15.

77. Most importantly, a person of ordinary skill in the art would consider a design to be a "commercial product" so long as it is intended and designed for eventual commercial sale - and not, for example, for experimental or academic purposes. Doc #621 194:18 to 195:3.

78. Based upon Shau's experience at Intel Corporation, he considered a product to be "commercial" if it was designed to be sold. Doc #621 112:24 to 113:5. Indeed, even the personnel at TSMC referred to Shau's USRAM product as a commercial product. Doc #621 113:6-8.

79. The phrase "manufactured successfully" is not a term of art, and its meaning is ambiguous rather than clear and unequivocal. Doc #621 195:4-10. Interpreted literally, the term

"manufactured successfully" in the '290 Application would indicate to a person of ordinary skill at a bare minimum that the circuit manufacturer had fabricated a chip successfully, the chip achieved a degree of functionality and the chip might lead to a successful commercial product. Doc #621 195:11 to 196:5.

80. The term "manufactured successfully" does not mean that the design is free of errors or bugs as long as the design is likely to lead to commercial success. Doc #621 195:16-21.

81. The first TSMC test chip had completed the manufacturing process, and, despite the presence of bugs in the extraneous logic portions of the chip, the memory portion of the chip (which contained the inventions claimed by the '290 Application) functioned successfully. Doc #621 109:2-25. Accordingly, the test chip was "manufactured successfully." Doc #621 114:2-5.

82. A person of ordinary skill would not interpret the term "manufactured successfully" in the '290 Application to reflect "full" or perfect functionality. The phrase "fully functional" is a term of art in the industry for perfect functionality, including functionality in areas of the chip that may have absolutely nothing to do with the claimed invention. Nor would a person of ordinary skill interpret "manufactured successfully" to suggest that the design was ready for mass production. Doc #621 195:4 to 196:22. Shau's statements to TSMC that his design was not worthy for mass production and not fully functional are consistent with his statement to the PTO that his claimed invention was successful; the claimed invention in his patent application related to the memory array (which worked perfectly), while the errors precluding mass production referred to the external logic circuitry.

83. In the latter half of 1997, Shau began designing his DRAM architecture using TSMC's 0.35μm technology. That same year, Shau submitted to TSMC a tape out designed for TSMC's 0.35μm generic logic process. Jury Trial 217:10-23.

84. After revision of his 0.35μm TSMC tape out, Shau received fabricated chips from TSMC that were fully functional. Based on these fully functional chips, Shau filed the '538 Application, which ultimately issued as the '229 Patent. Jury Trial 217:10 to 218:20. Thus, a fully functional chip existed at the time the application for the '229 Patent was filed.

85. TSMC filed its motion for summary judgment on its inequitable conduct counterclaim on the eve of a mediation with UniRAM, suggesting that its pursuit of this claim was intended as settlement leverage against UniRAM. Doc #621 116:14-20.

86. In 2006, UniRAM reached a settlement with MoSys, a former defendant in this case. Doc #293.

87. To provide MoSys with complete relief, UniRAM had to include a patent release in the MoSys settlement that covered TSMC. Doc #287 at 3.

88. After executing the MoSys settlement, UniRAM pursued only trade-secret, and not patent, claims against TSMC. In November of 2006, UniRAM and TSMC filed jointly with the court a case management statement in which UniRAM stated that it was no longer seeking to enforce the '229 patent against TSMC. Doc #287 at 3.

89. Despite no longer needing to pursue its inequitable conduct theory as a defense against UniRAM's patent claims - because UniRAM relinquished those claims - TSMC did not drop its inequitable conduct counterclaim.

90. At the hearing on TSMC's motion for summary judgment, counsel for TSMC informed the court that a motivation behind filing the motion was to deny UniRAM's and UniRAM's counsel access to the MoSys settlement funds: "UniRAM's a corporate entity and they could pay [the settlement money] out to their shareholders. Whether that's appropriate or not is a different question. There's probably preexisting payments to law firms that need to be made. They don't have a lot of money." Doc #381 (1/11/07 Inequitable Conduct Summary Judgment Hearing Tr) at 3.

91. Relatedly, if TSMC had only been interested in protecting the settlement funds, it could have chosen a less drastic measure, such as an injunction, rather than seeking a judgment on its counterclaim. Doc #381 at 3-13. This suggests that TSMC's continued pursuit of its counterclaim was pretextual and was not in good faith.

92. In February 2007, the court denied TSMC's motion for summary judgment on inequitable conduct. Doc #349.

93. In the court's order denying TSMC's motion for summary judgment, the court indicated that TSMC had failed to identify any bases by which a PTO examiner could have made an obviousness or enablement rejection of the UniRAM patents. The court also labeled TSMC's interpretation of its evidence of Shau's intent to deceive the PTO as "odd." Doc #349 at 16.

94. Despite the summary judgment order that called into question the fundamental merits of TSMC's inequitable conduct counterclaim, TSMC continued to pursue that counterclaim after issuance of the order.

95. In continuing to pursue its inequitable conduct counterclaim after the summary judgment order, TSMC did not act to rectify the deficiencies in its position identified in that order. Doc #601 at 3-4.

96. UniRAM's trade secret claims proceeded to trial in September 2007.

97. During the trade secret trial, TSMC cross-examined Shau about most of the same supposed misstatements in his patent applications that form the basis of its inequitable conduct counterclaim. Jury Trial 265:25 to 287:5.

98. During the trade secret trial, TSMC's counsel argued to the jury in closing arguments that these supposed misstatements meant that Shau lacked credibility. Jury Trial 1841:15 to 1847:18.

99. Despite TSMC's arguments about the supposed misstatements in UniRAM's patents, the jury found in favor of Shau and UniRAM. Doc #544.

100. Even though TSMC's use of its supposed inequitable conduct evidence had not persuaded the jury to find against UniRAM, TSMC continued to pursue its inequitable conduct counterclaim against UniRAM following the jury verdict.

CONCLUSIONS OF LAW

1. A breach of the duties of candor, good faith, and honesty when prosecuting patent applications constitutes inequitable conduct. Molins PLC v Textron, Inc, 48 F3d 1172, 1178 (Fed Cir 1995). The proponent of an inequitable conduct defense has a "heavy burden to meet." UniRAM Tech, Inc v Taiwan Semiconductor

Mfg Co, No 04-1268 (VRW), 2007 WL 596397, at *2 (N D Cal Feb 21, 2007), citing Hoffman-La Roche, Inc v Promega Corp, 323 F3d 1354, 1359 (Fed Cir 2003).

2. Inequitable conduct based upon an affirmative misstatement of fact or omission requires clear and convincing evidence of:

a. a misrepresentation (by statement or omission) by the applicant;

b. the materiality of the misrepresentation and

c. intent to deceive the PTO.

Honeywell Intl Inc v Universal Avionics Sys Corp, 488 F3d 982, 999 (Fed Cir 2007); Syntex (USA) LLC v Apotex, Inc, 407 F3d 1371, 1384 (Fed Cir 2005) ("Materiality and intent to deceive are distinct factual inquiries, and each must be shown by clear and convincing evidence."), quoting Life Techs, Inc v Clontech Labs, Inc, 224 F3d 1320, 1324 (Fed Cir 2000).

3. Once a material misstatement or omission and intent to deceive have been established, the district court must weigh these factors in light of all of the circumstances to determine whether a finding that inequitable conduct occurred is appropriate. Dayco Prods, Inc v Total Containment, Inc, 329 F3d 1358, 1362-63 (Fed Cir 2003).

4. "[A] court must conduct a balancing test between the levels of materiality and intent, with a greater showing of one factor allowing a lesser showing of the other. Life Techs, 224 F3d at 1324.

5. Where the proponent of the inequitable conduct defense seeks to have a later patent declared unenforceable based upon alleged inequitable conduct in related patents, the proponent "must prove an 'immediate and necessary relation' between the inequitable conduct in the earlier patents and the enforcement of the descendent patent." UniRAM, 2007 WL 596397, at *2; see Hoffman-La Roche, Inc v Promega Corp, 319 F Supp 2d 1011 (N D Cal 2004).

6. The fact that an applicant uses an imprecise term in a patent application does not provide "clear and convincing" evidence of a misstatement. Kothmann Enterprises, Inc v Trinity Indus, Inc, 455 F Supp 2d 608, 618-24 (S D Tex 2006), citing Hoffmann-La Roche, Inc v Promega Corp, 323 F3d 1354, 1363 (Fed Cir 2003), Purdue Pharma LP v Endo Pharms, Inc, 410 F3d 690 (Fed Cir 2005) and Frazier v Roessel Cine Photo Tech, Inc, 417 F3d 1230 (Fed Cir 2005). Although courts have found inequitable conduct based on "applicants' representations that they had performed experimental testing when they had not done so and that they had achieved test results that simply did not exist" (see Kothmann, 455 F Supp 2d at 623), here the evidence shows that Shau did perform tests. The language Shau used to describe his claimed invention may at points have been imprecise. Shau's testimony describing the testing he did and the language he chose to use in his patent applications makes clear that the statements TSMC challenges were not misrepresentations. See Kothmann, 455 F Supp 2d at 623-24.

7. If the PTO needs more information in interpreting a patent application, the PTO examiner is authorized to seek clarification or additional information. Star Fruits SNC v United States, 393 F3d 1277, 1283 (Fed Cir 2005) ("The Office is clearly

entitled to use section 1.105 to seek information that may support a rejection. Just as the applicant produces information it deems pertinent to patentability under section 1.56, the examiner is free to request information under section 1.105 that the examiner deems pertinent to the issue of patentability.”).

8. The PTO has adequate resources to investigate the representations made by applicants, if necessary. The PTO has the benefit of “hundreds of experts in the relevant arts to make independent inquiries.” Aptix Corp v Quickturn Design Systems, Inc, 269 F3d 1369, 1379 (Fed Cir 2001) (“The courts have no greater resources to uncover fraud than the PTO. Although patent prosecutions are ex parte and judicial proceedings are adversarial, the PTO has the benefit of hundreds of experts in the relevant arts to make independent inquiries.”) (Mayer dissenting in part).

9. Attorney fees may be awarded in an exceptional case. 35 USC § 285.

10. A finding that a case is exceptional imposes a more stringent requirement than the standard for proof of inequitable conduct, and in turn, an award of attorney fees imposes a more stringent requirement than the “exceptional case” standard. Argus Chemical Corp v Fibre Glass-Evercoat Co, 812 F2d 1381, 1387 (Fed Cir 1987) (Nies concurring).

11. Whether a case is “exceptional” is a question of fact. Brasseler, USA I, LP v Stryker Sales Corp, 267 F3d 1370, 1378 (Fed Cir 2001); Graco, Inc v Binks Mfg Co, 60 F3d 785, 794-95 (Fed Cir 1995) (“A finding by a court that a case is exceptional is a factual determination whereas the decision to award fees is discretionary.”) (internal citations omitted). Direct or

circumstantial evidence that is clear and convincing is needed to establish an "exceptional case." Brasseler, 267 F3d at 1378-79.

12. Over-assertion of the defense of inequitable conduct has become an "absolute plague." Burlington Indus, Inc v Dayco Corp, 849 F2d 1418, 1422 (Fed Cir 1988).

13. Often litigants improperly assert the defense as a delay tactic, a tactic to obfuscate the issues before the court or to drive up the patentee's litigation costs. Chiron Corp v Abbott Labs, 156 FRD 219, 221 (N D Cal 1994).

14. "A patent litigant should be made to feel * * * that an unsupported charge of inequitable conduct in the Patent Office is a negative contribution to the rightful administration of justice." Burlington Indus, 849 F2d at 1422; see Fiskars, Inc v Hunt Mfg Co, 221 F3d 1318, 1328 (Fed Cir 2000) (affirming an award of attorney fees for an inequitable conduct defense "so lacking in substance as to constitute a waste of the time and resources of all the participants").

15. Under 28 USC § 1927, courts have statutory authority to sanction attorneys for improper litigation conduct: "Any attorney * * * who so multiplies the proceedings in any case unreasonably and vexatiously may be required by the court to satisfy personally the excess costs, expenses, and attorneys' fees reasonably incurred because of such conduct."

16. Having reviewed the evidence and considered the testimony at trial, the court considers the testimony of UniRAM's witnesses to be more credible as to the existence or nonexistence of alleged misstatements in the patents and patent applications at issue in this proceeding than the evidence and testimony presented by TSMC.

17. The testimony of TSMC's primary witness as to falsity rested primarily upon his experience alone and was uncorroborated by any documents or other witnesses. Accordingly, it fails to provide clear and convincing evidence of falsity.

18. The basis of TSMC's claim is that some of Shau's statements can be possibly read in a context that would make the statements misleading. TSMC does not come close to proving that the alleged misstatements were false or misleading.

19. To the extent that the words used in the patents and patent applications at issue were imprecise, this imprecision does not give rise to clear and convincing evidence of falsity.

20. The court concludes that TSMC failed to prove by a preponderance of the evidence, much less clear and convincing evidence, that Shau made any misstatements in the patents and patent applications at issue.

21. Because TSMC has failed to establish by clear and convincing evidence the existence of any misstatements, the court concludes that there was no inequitable conduct in the prosecution of any of the patents at issue. The issues of materiality and intent are moot.

22. In light of the circumstances, the court finds that this is an exceptional case because TSMC pursued its counterclaim long after UniRAM dropped its patent infringement case. Moreover, circumstantial evidence suggests that TSMC brought this counterclaim not because it believed that Shau lied to the patent office but because it wanted to intimidate UniRAM. Most importantly, TSMC's evidence of falsity was barren. TSMC's pursuit of its inequitable conduct contention is highly questionable.

Nonetheless, the court declines to order TSMC to pay the attorney fees UniRAM incurred in defending the defense and counterclaim. Awarding fees is an extraordinary sanction, and the Federal Circuit has not developed detailed criteria for such awards in the context of inequitable conduct claims.

IT IS SO ORDERED.

VAUGHN R WALKER
United States District Chief Judge

PATENT APPLICATION
ATTY. DOCKET NO. 00100.02.0001

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

FILING OF A UNITED STATES PATENT APPLICATION

A GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

INVENTORS:

Steven Morein 10 Magazine, Apt. 801 Cambridge, Massachusetts 02139	Laurent Lefebvre 124 Parenchere Lachenaie Quebec Canada J6W 6A5
Andy Gruber 215 Pleasant Street Arlington, Massachusetts 02476	Andi Skende 49 Sheridan Drive, #11 Shrewsbury, Massachusetts 01545

ASSIGNEE:

ATI Technologies, Inc.
1 Commerce Valley Drive East
Markham, Ontario
Canada L3T 7X6

ATTORNEY OF RECORD:
CHRISTOPHER J. RECKAMP
REGISTRATION NO. 34,414
VEDDER, PRICE, KAUFMAN & KAMMHOLZ, P.C.
222 NORTH LASALLE STREET, SUITE 2600
CHICAGO, ILLINOIS 60601
PHONE (312) 609-7500
FAX (312) 609-5005

Express Mail Label No. EL982266165US

Date of Deposit: November 20, 2003
I hereby certify that this paper is being deposited with the U.S. Postal Service "Express Mail Post Office to Addressee" service under 37 C.F.R. Section 1.10 on the date of deposit, indicated above, and is addressed to: Mail Stop Patent Application, Commissioner for Patents, P. O. Box 1450, Alexandria, VA 22313-1450.

Name of Depositor: Margaret Caruso

Signature: Margaret Caruso

A GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

FIELD OF THE INVENTION

[0001] The present invention generally relates to graphics processors and, more particularly, to a graphics processor architecture employing a single shader.

BACKGROUND OF THE INVENTION

[0002] In computer graphics applications, complex shapes and structures are formed through the sampling, interconnection and rendering of more simple objects, referred to as primitives. An example of such a primitive is a triangle, or other suitable polygon. These primitives, in turn, are formed by the interconnection of individual pixels. Color and texture are then applied to the individual pixels that comprise the shape based on their location within the primitive and the primitives orientation with respect to the generated shape; thereby generating the object that is rendered to a corresponding display for subsequent viewing.

[0003] The interconnection of primitives and the application of color and textures to generated shapes are generally performed by a graphics processor. Conventional graphics processors include a series of shaders that specify how and with what corresponding attributes, a final image is drawn on a screen, or suitable display device. As illustrated in FIG. 1, a conventional shader 10 can be represented as a processing block 12 that accepts a plurality of bits of input data, such as, for example, object shape data (14) in object space (x,y,z); material properties of the object, such as color (16); texture information (18); luminance information (20); and viewing angle information (22) and provides output data (28) representing the object with texture and other appearance properties applied thereto (x′, y′, z′).

[0004] In exemplary fashion, as illustrated in FIGS. 2A-2B, the shader accepts the vertex coordinate data representing cube 30 (FIG. 2A) as inputs and provides data representing, for example, a perspectively corrected view of the cube 30′ (FIG. 2B) as an output. The corrected view may be provided, for example, by applying an appropriate transformation matrix to the data representing the initial cube 30. More specifically, the representation illustrated in FIG. 2B is provided by a vertex shader that accepts as inputs the data representing, for example, vertices V_X, V_Y and V_Z, among others of cube 30 and providing angularly oriented vertices $V_{X'}$,$V_{Y'}$ and $V_{Z'}$, including any appearance attributes of corresponding cube 30′.

[0005] In addition to the vertex shader discussed above, a shader processing block that operates on the pixel level, referred to as a pixel shader is also used when generating an object for display. Generally, the pixel shader provides the color value associated with each pixel of a rendered object. Conventionally, both the vertex shader and pixel shader are separate components that are configured to perform only a single transformation or operation. Thus, in order to perform a position and a texture transformation of an input, at least two shading operations and hence, at least two shaders, need to be employed. Conventional graphics processors require the use of both a vertex shader and a pixel shader in order to generate an object. Because both types of shaders are required, known graphics processors are relatively large in size, with most of the real estate being taken up by the vertex and pixel shaders.

[0006] In addition to the real estate penalty associated with conventional graphics processors, there is also a corresponding performance penalty associated therewith. In conventional graphics processors, the vertex shader and the pixel shader are juxtaposed in a sequential, pipelined fashion, with the vertex shader being positioned before and operating on vertex data before the pixel shader can operate on individual pixel data.

[0007] Thus, there is a need for an improved graphics processor employing a shader that is both space efficient and computationally effective.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] The present invention and the associated advantages and features thereof, will become better understood and appreciated upon review of the following detailed description of the invention, taken in conjunction with the following drawings, where like numerals represent like elements, in which:

[0009] FIG. 1 is a schematic block diagram of a conventional shader;

[00010] FIGS. 2A-2B are graphical representations of the operations performed by the shader illustrated in FIG. 1;

[00011] FIG. 3 is a schematic block diagram of a conventional graphics processor architecture;

[00012] FIG. 4A is a schematic block diagram of a graphics processor architecture according to the present invention;

[00013] FIG. 4B is a schematic block diagram of an optional input component to the graphics processor according to an alternate embodiment of the present invention; and

[00014] FIG. 5 is an exploded schematic block diagram of the unified shader employed in the graphics processor illustrated in FIG. 4A.

DETAILED DESCRIPTION OF THE INVENTION

[00015] Briefly stated, the present invention is directed to a graphics processor that employs a unified shader that is capable of performing both the vertex operations and the pixel operations in a space saving and computationally efficient manner. In an exemplary embodiment, a graphics processor according to the present invention includes an arbiter circuit for selecting one of a plurality of inputs for processing in response to a control signal; and a shader, coupled to the arbiter, operative to process the selected one of the plurality of inputs, the shader including means for performing vertex operations and pixel operations, and wherein the shader performs one of the vertex operations or pixel operations based on the selected one of the plurality of inputs.

[00016] The shader includes a general purpose register block for storing at least the plurality of selected inputs, a sequencer for storing logical and arithmetic instructions that are used to perform vertex and pixel manipulation operations and a processor capable of executing both floating point arithmetic and logical operations on the selected inputs according to the instructions maintained in the sequencer. The shader of the present invention is referred to as a "unified" shader because it is configured to perform both vertex and pixel operations. By employing the unified shader of the present invention, the associated graphics processor is more space efficient than conventional graphics processors because the unified shader takes up less real estate than the conventional multi-shader processor architecture.

[00017] In addition, according to the present invention, the unified shader is more computationally efficient because it allows the shader to be flexibly allocated to pixels or vertices based on workload.

[00018] Referring now to FIG. 3, illustrated therein is a graphics processor incorporating a conventional pipeline architecture. As shown, the graphics processor 40 includes a vertex fetch block 42 which receives vertex information relating to a primitive to be rendered from an off-chip memory 55 on line 41. The fetched vertex data is then transmitted to a vertex cache 44 for storage on line 43. Upon request, the vertex data maintained in the vertex cache 44 is transmitted to a vertex shader 46 on line 45. As discussed above, an example of the information that is requested by and transmitted to the vertex shader 46 includes the object shape, material properties (e.g. color), texture information, and viewing angle. Generally, the vertex shader 46 is a programmable mechanism which applies a transformation position matrix to the input position information (obtained from the vertex cache 44), thereby providing data representing a perspectively corrected image of the object to be rendered, along with any texture or color coordinates thereof.

[00019] After performing the transformation operation, the data representing the transformed vertices are then provided to a vertex store 48 on line 47. The vertex store 48 then transmits the modified vertex information contained therein to a primitive assembly block 50 on line 49. The primitive assembly block 50 assembles, or converts, the input vertex information into a plurality of primitives to be subsequently processed. Suitable methods of assembling the input vertex information into primitives is known in the art and will not be discussed in greater detail here. The assembled primitives are then transmitted to a rasterization engine 52, which converts the previously assembled primitives into pixel data through a process referred to as walking. The resulting pixel data is then transmitted to a pixel shader 54 on line 53.

[00020] The pixel shader 54 generates the color and additional appearance attributes that are to be applied to a given pixel, and applies the appearance attributes to the respective pixels. In addition, the pixel shader 54 is capable of fetching texture data from a texture map 57 as indexed by the pixel data from the rasterization engine 52 by transmitting such information on line 55 to the texture map. The requested texture data is then transmitted back from the texture map 57 on line 57′ and stored in a texture cache 56 before being routed to the pixel shader on line 58. Once the texture data has been received, the pixel shader 54 then performs specified logical or arithmetic operations on the received texture data to generate the pixel color or other appearance attribute of interest. The generated pixel appearance attribute is then combined with a base color, as provided by the rasterization engine on line 53, to thereby provide a pixel color to the pixel corresponding at the position of interest. The pixel appearance attribute present on line 59 is then transmitted to post raster processing blocks (not shown).

[00021] As described above, the conventional graphics processor 40 requires the use of two separate shaders: a vertex shader 46 and a pixel shader 54. A drawback associated with such an architecture is that the overall footprint of the graphics processor is relatively large as the two shaders take up a large amount of real estate. Another drawback associated with conventional graphics processor architectures is that can exhibit poor computational efficiency.

[00022] Referring now to FIG. 4A, in an exemplary embodiment, the graphics processor 60 of the present invention includes a multiplexer 66 having vertex (e.g. indices) data provided at a first input thereto and interpolated pixel parameter (e.g. position) data and attribute data from a rasterization engine 74 provided at a second input. A control signal generated by an arbiter 64 is transmitted to the multiplexer 66 on line 63. The arbiter 64 determines which of the two inputs to the multiplexer 66 is transmitted to a unified shader 62 for further processing. The arbitration scheme employed by the arbiter 64 is as follows: the vertex data on the first input of the multiplexer 66 is transmitted to the unified shader 62 on line 65 if there is enough resources available in the unified shader to operate on the vertex data; otherwise, the interpolated pixel parameter data present on the second input will be passed to the unified shader 62 for further processing.

[00023] Referring briefly to FIG. 5, the unified shader 62 will now be described. As illustrated, the unified shader 62 includes a general purpose register block 92, a plurality of source registers: including source register A 93, source register B 95, and source register C 97, a processor (e.g. CPU) 96 and a sequencer 99. The general purpose register block 92 includes sixty four registers, or available entries, for storing the information transmitted from the multiplexer 66 on line 65 or any other information to be maintained within the unified shader. The data present in the general purpose register block 92 is transmitted to the plurality of source registers via line 109.

[00024] The processor 96 may be comprised of a dedicated piece of hardware or can be configured as part of a general purpose computing device (i.e. personal computer). In an exemplary embodiment, the processor 96 is adapted to perform 32-bit floating point arithmetic operations as well as a complete series of logical operations on corresponding operands. As shown, the processor is logically partitioned into two sections. Section 96 is configured to execute, for example, the 32-bit floating point arithmetic operations of the unified shader. The second section, 96A, is configured to perform scaler operations (e.g. log, exponent, reciprocal square root) of the unified shader.

[00025] The sequencer 99 includes constants block 91 and an instruction store 98. The constants block 91 contains, for example, the several transformation matrices used in connection with vertex manipulation operations. The instruction store 98 contains the necessary instructions that are executed by the processor 96 in order to perform the respective arithmetic and logic operations on the data maintained in the general purpose register block 92 as provided by the source registers 93-95. The instruction store 98 further includes memory fetch instructions that, when executed, causes the unified shader 62 to fetch texture and other types of data, from memory 82 (FIG. 4A). In operation, the sequencer 99 determines whether the next instruction to be executed (from the instruction store 98) is an arithmetic or logical instruction or a memory (e.g. texture fetch) instruction. If the next instruction is a memory instruction or request, the sequencer 99 sends the request to a fetch block (not shown) which retrieves the required information from memory 82 (FIG. 4A). The retrieved information is then transmitted to the sequencer 99, through the vertex texture cache 68 (FIG. 4A) as described in greater detail below.

[00026] If the next instruction to be executed is an arithmetic or logical instruction, the sequencer 99 causes the appropriate operands to be transferred from the general purpose register block 92 into the appropriate source registers (93, 95, 97) for execution, and an appropriate signal is sent to the processor 96 on line 101 indicating what operation or series of operations are to be executed on the several operands present in the source registers. At this point, the processor 96 executes the instructions on the operands present in the source registers and provides the result on line 85. The information present on line 85 may be transmitted back to the general purpose register block 92 for storage, or transmitted to succeeding components of the graphics processor 60.

[00027] As discussed above, the instruction store 98 maintains both vertex manipulation instructions and pixel manipulation instructions. Therefore, the unified shader 99 of the present invention is able to perform both vertex and pixel operations, as well as execute memory fetch operations. As such, the unified shader 62 of the present invention is able to perform both the vertex shading and pixel shading operations on data in the context of a graphics controller based on information passed from the multiplexer. By being adapted to perform memory fetches, the unified shader of the present invention is able to perform additional processes that conventional vertex shaders cannot perform; while at the same time, perform pixel operations.

[00028] The unified shader 62 has ability to simultaneously perform vertex manipulation operations and pixel manipulation operations at various degrees of completion by being able to freely switch between such programs or instructions, maintained in the instruction store 98, very quickly. In application, vertex data to be processed is transmitted into the general purpose register block 92 from multiplexer 66.

The instruction store 98 then passes the corresponding control signals to the processor 96 on line 101 to perform such vertex operations. However, if the general purpose register block 92 does not have enough available space therein to store the incoming vertex data, such information will not be transmitted as the arbitration scheme of the arbiter 64 is not satisfied. In this manner, any pixel calculation operations that are to be, or are currently being, performed by the processor 96 are continued, based on the instructions maintained in the instruction store 98, until enough registers within the general purpose register block 92 become available. Thus, through the sharing of resources within the unified shader 62, processing of image data is enhanced as there is no down time associated with the processor 96.

[00029] Referring back to FIG. 4A, the graphics processor 60 further includes a cache block 70, including a parameter cache 70A and a position cache 70B which accepts the pixel based output of the unified shader 62 on line 85 and stores the respective pixel parameter and position information in the corresponding cache. The pixel information present in the cache block 70 is then transmitted to the primitive assembly block 72 on line 71. The primitive assembly block 72 is responsible for assembling the information transmitted thereto from the cache block 70 into a series of triangles, or other suitable primitives, for further processing. The assembled primitives are then transmitted on line 73 to rasterization engine block 74, where the transmitted primitives are then converted into individual pixel data information through a walking process, or any other suitable pixel generation process. The resulting pixel data from the rasterization engine block 74 is the interpolated pixel parameter data that is transmitted to the second input of the multiplexer 66 on line 75.

[00030] In those situations when vertex data is transmitted to the unified shader 62 through the multiplexer 66, the resulting vertex data generated by the processor 96, is transmitted to a render back end block 76 which converts the resulting vertex data into at least one of several formats suitable for later display on display device 84. For example, if a stained glass appearance effect is to be applied to an image, the information corresponding to such appearance effect is associated with the appropriate position data by the render back end 76. The information from the render back end 76 is then transmitted to memory 82 and a display controller line 80 via memory controller 78. Such appropriately formatted information is then transmitted on line 83 for presentation on display device 84.

[00031] Referring now to FIG. 4B, shown therein is a vertex block 61 which is used to provide the vertex information at the first input of the multiplexer 66 according to an alternate embodiment of the present invention. The vertex block 61 includes a vertex fetch block 61A which is responsible for retrieving vertex information from memory 82, if requested, and transmitting that vertex information into the vertex cache 61B. The information stored in the vertex cache 61B comprises the vertex information that is coupled to the first input of multiplexer 66.

[00032] As discussed above, the graphics processor 60 of the present invention incorporates a unified shader 62 which is capable of performing both vertex manipulation operations and pixel manipulation operations based on the instructions stored in the instruction store 98. In this fashion, the graphics processor 60 of the present invention takes up less real estate than conventional graphics processors as separate vertex shaders and pixel shaders are no longer required. In addition, as the unified shader 62 is capable of alternating between performing vertex manipulation operations and pixel manipulation operations, graphics processing efficiency is enhanced as one type of data operations is not dependent upon another type of data operations. Therefore, any performance penalties experienced as a result of dependent operations in conventional graphics processors are overcome.

[00033] The above detailed description of the present invention and the examples described therein have been presented for the purposes of illustration and description. It is therefore contemplated that the present invention cover any and all modifications, variations and equivalents that fall within the spirit and scope of the basic underlying principles disclosed and claimed herein.

CLAIMS

What is claimed is:

1. A graphics processor, comprising:

an arbiter circuit for selecting one of a plurality of inputs in response to a control signal; and

a shader, coupled to the arbiter circuit, operative to process the selected one of the plurality of inputs, the shader including means for performing vertex operations and pixel operations, and performing one of the vertex operations or pixel operations based on the selected one of the plurality of inputs, wherein the shader provides a appearance attribute.

2. The graphics processor of claim 1, further including a vertex storage block for maintaining vertex information.

3. The graphics processor of claim 2, wherein the vertex storage block further includes a parameter cache operative to maintain appearance attribute data for a corresponding vertex and a position cache operative to maintain position data for a corresponding vertex.

4. The graphics processor of claim 1, wherein the appearance attribute is color, and the color is associated with a corresponding pixel when the selected one of the plurality inputs is pixel data.

5. The graphics processor of claim 1, wherein the appearance attribute is position, and the position attribute is associated with a corresponding vertex when the selected one of the plurality of inputs is vertex data.

6. The graphics processor of claim 5, wherein the appearance attribute is color, and the color attribute is associated with a corresponding pixel when the selected one of the plurality of inputs is pixel data.

7. The graphics processor of claim 5, wherein the appearance attribute is one of the following: color, lighting, texture, normal and position data.

8. The graphics processor of claim 1, wherein the appearance value is depth.

9. The graphics processor of claim 1, wherein the selection circuit is a multiplexer, and the control signal is provided by an arbiter, wherein the arbiter is coupled to the multiplexer.

10. The graphics processor of claim 1, wherein the shader provides vertex position data and further including a primitive assembly block, coupled to the shader, operative to generate primitives in response to the vertex position data.

11. The graphics processor of claim 10, further including a raster engine, coupled to the primitive assembly block, operative to generate pixel parameter data in response to the assembled vertex data.

12. The graphics processor of claim 1, wherein the shader generates pixel color information in response to the selected one of the plurality of inputs.

13. The graphics processor of claim 1, wherein the shader includes a register block for maintaining the selected one of the plurality of inputs, a computation element operative to perform arithmetic and logical operations on the data maintained in the register block, and a sequencer for maintaining the instructions that are executed by the computation element.

14. The graphics processor of claim 1, wherein the shader further includes circuitry operative to access a memory.

15. A unified shader, comprising:

a general purpose register block for maintaining data;

a processor unit; and

a sequencer, coupled to the general purpose register block and the processor unit, the sequencer maintaining instructions operative to cause the processor unit to execute vertex calculation and pixel calculation operations on selected data maintained in the general purpose register block.

16. The shader of claim 15, wherein the sequencer further includes circuitry operative to fetch data from a memory.

17. The shader of claim 15, further including a selection circuit operative to provide information to the general purpose block in response to a control signal.

18. The shader of claim 15, wherein the processor unit executes instructions that generate a pixel color in response to the selected one of the plurality of inputs.

19. The shader of claim 15, wherein the processor unit executes vertex calculations while the pixel calculations are still in progress.

20. The shader of claim 15, wherein the processor unit generates vertex position and appearance data in response to a selected one of the plurality of inputs.

21. The shader of claim 17, wherein the selection circuit is a multiplexer and the control signal is provided by an arbiter.

A GRAPHICS PROCESSING ARCHITECTURE EMPLOYING A UNIFIED SHADER

ABSTRACT

A graphics processing architecture employing a single shader is disclosed. The architecture includes a circuit operative to select one of a plurality of inputs in response to a control signal; and a shader, coupled to the arbiter, operative to process the selected one of the plurality of inputs, the shader including means for performing vertex operations and pixel operations, and wherein the shader performs one of the vertex operations or pixel operations based on the selected one of the plurality of inputs. The shader includes a register block which is used to store the plurality of selected inputs, a sequencer which maintains vertex manipulation and pixel manipulations instructions and a processor capable of executing both floating point arithmetic and logical operations on the selected inputs in response to the instructions maintained in the sequencer.

CHICAGO/#837098.1

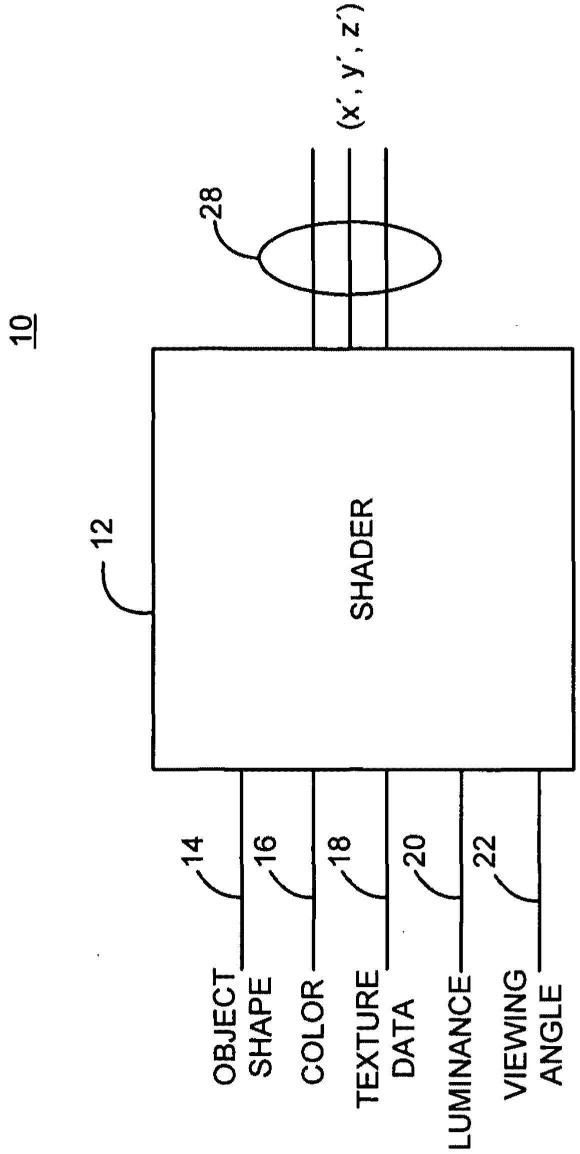


FIG. 1
(PRIOR ART)

"A Graphics Processing Architecture
Employing A Unified Shader"
Inventors: Morein et al.
Docket No. 0100.02.0001

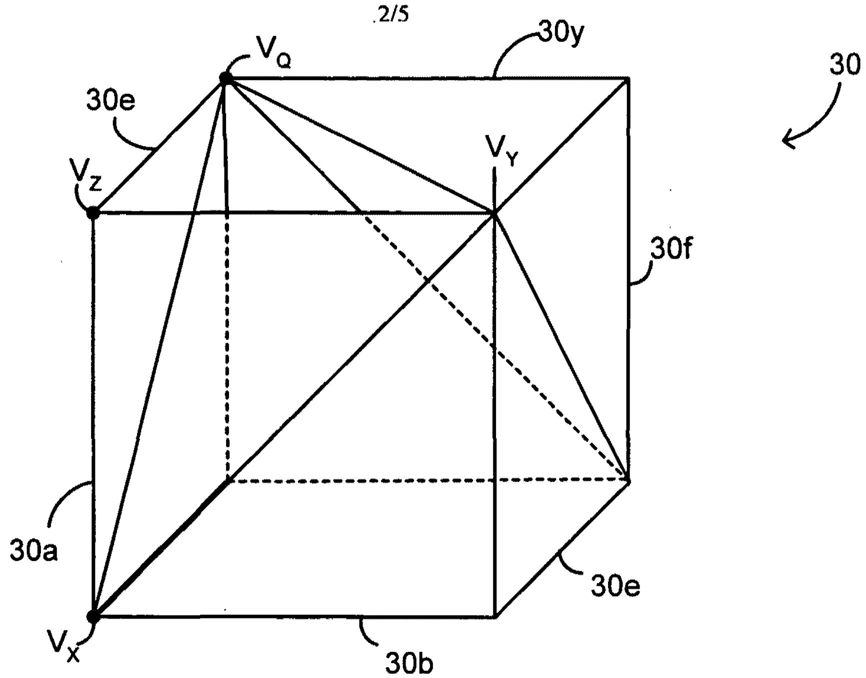


FIG. 2A
(PRIOR ART)

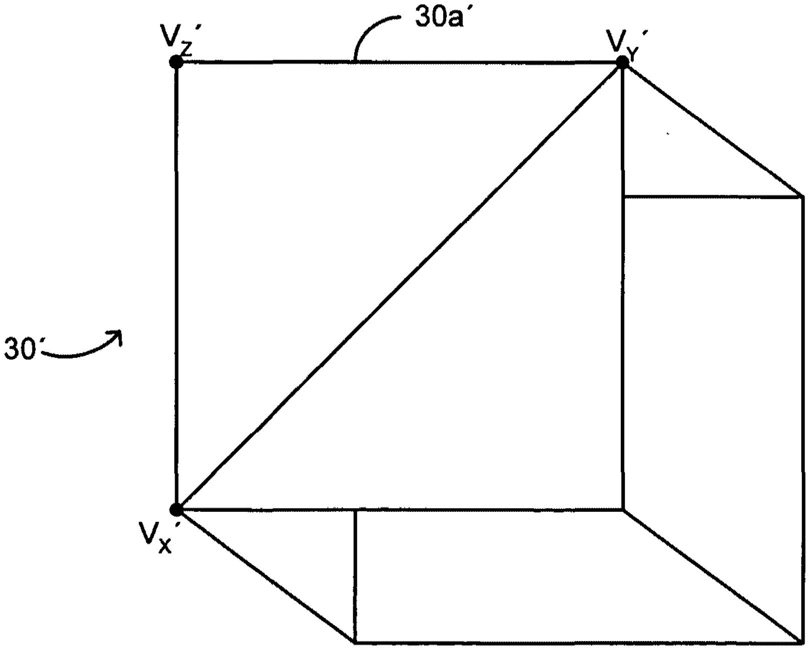


FIG. 2B
(PRIOR ART)

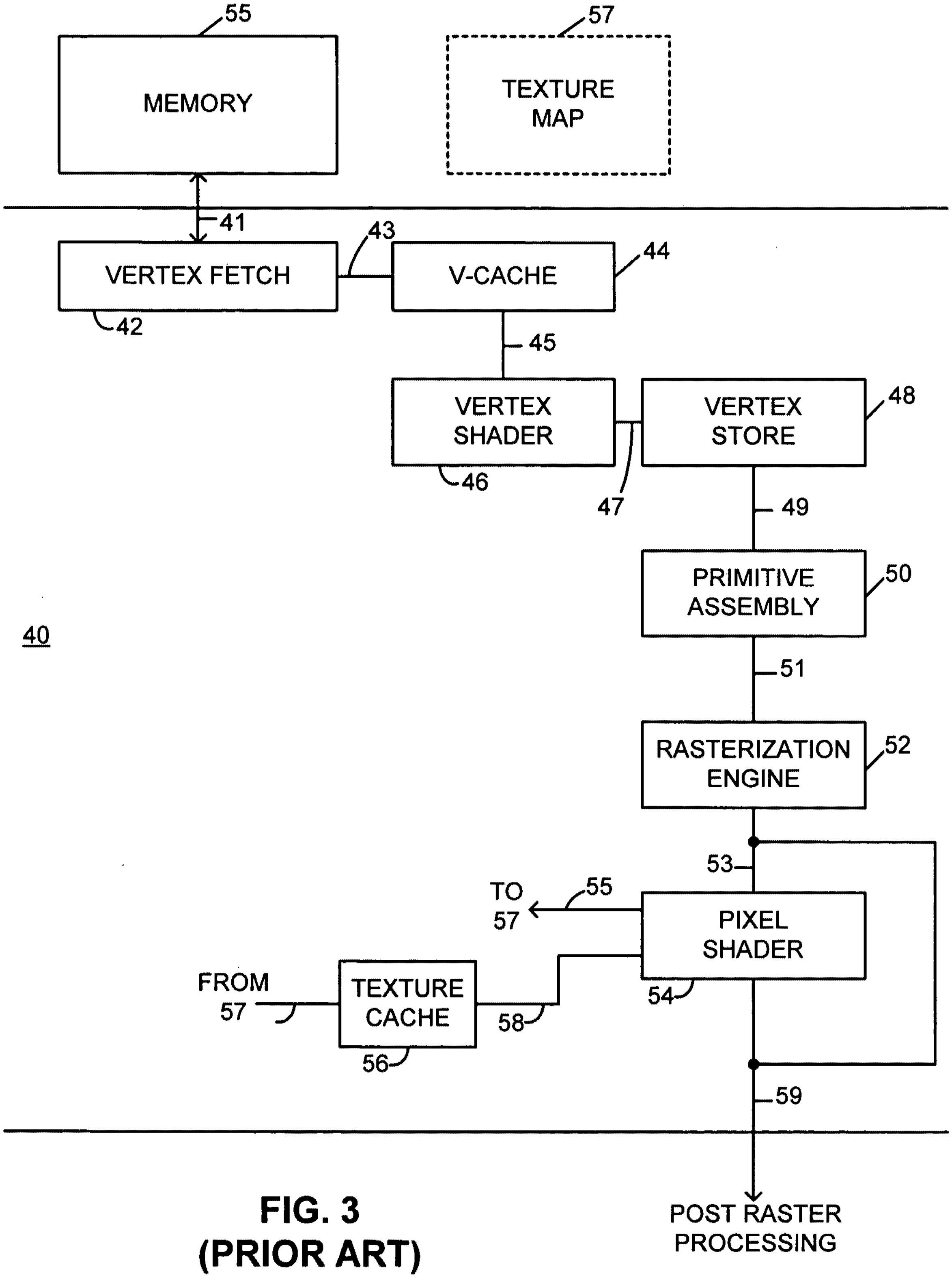


FIG. 3
(PRIOR ART)

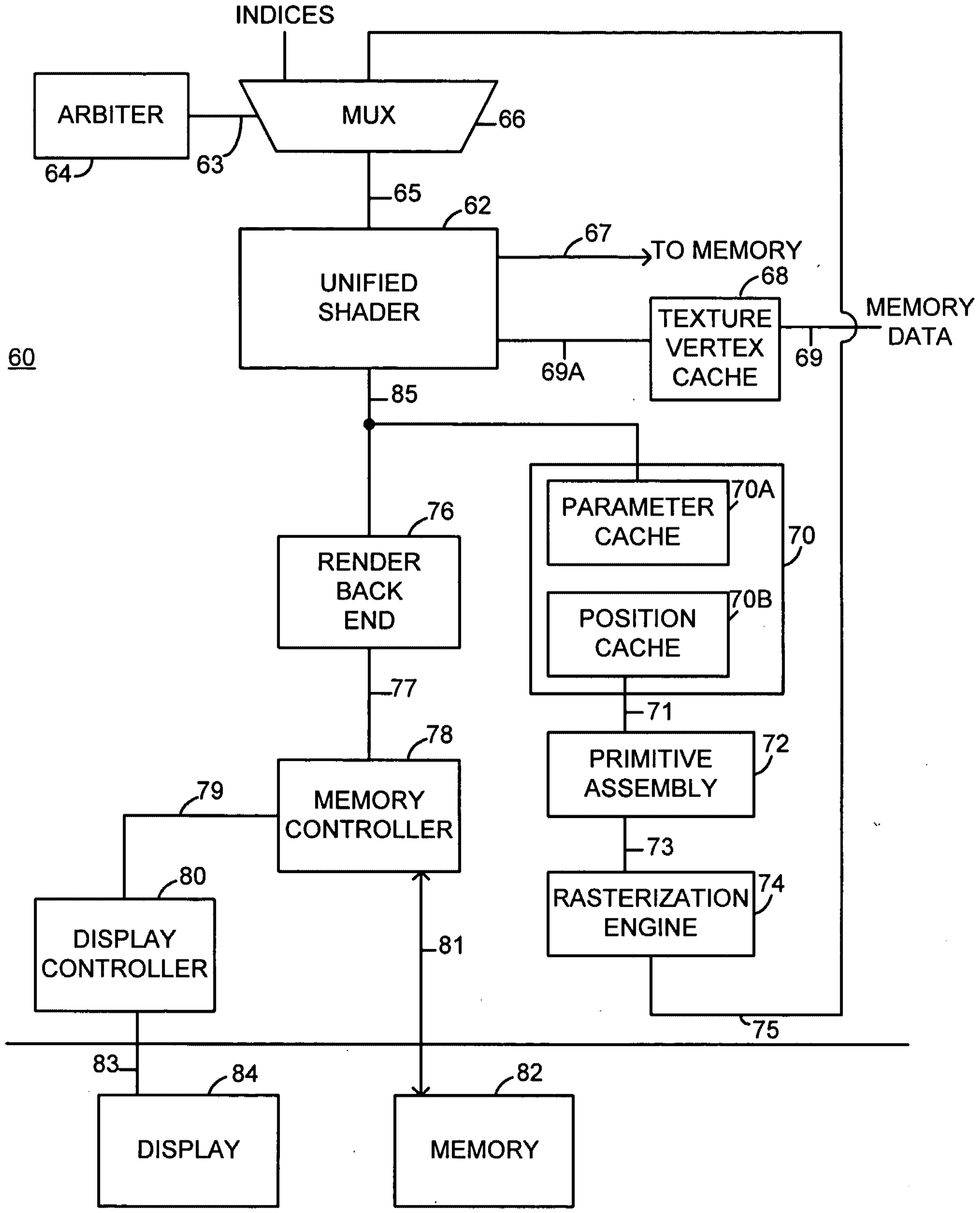

INDICES
ARBITER
64
63
MUX
66
65
62
UNIFIED SHADER
67
TO MEMORY
68
TEXTURE VERTEX CACHE
MEMORY DATA
69
69A
60
85
PARAMETER CACHE
70A
70
POSITION CACHE
70B
76
RENDER BACK END
71
PRIMITIVE ASSEMBLY
72
77
78
MEMORY CONTROLLER
79
73
80
74
RASTERIZATION ENGINE
DISPLAY CONTROLLER
81
75
83
84
82
DISPLAY
MEMORY

FIG. 4A

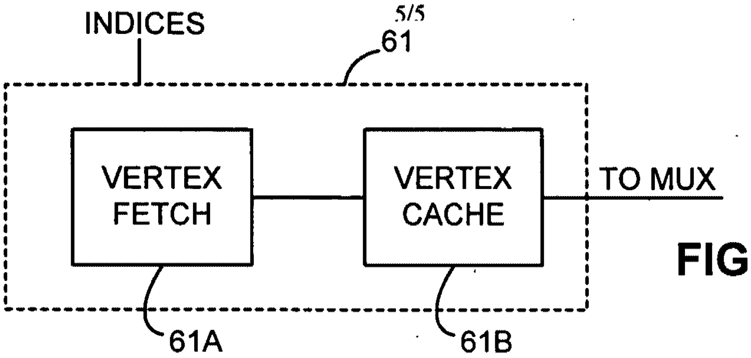

INDICES
61
VERTEX FETCH
VERTEX CACHE
TO MUX
61A
61B

FIG. 4B

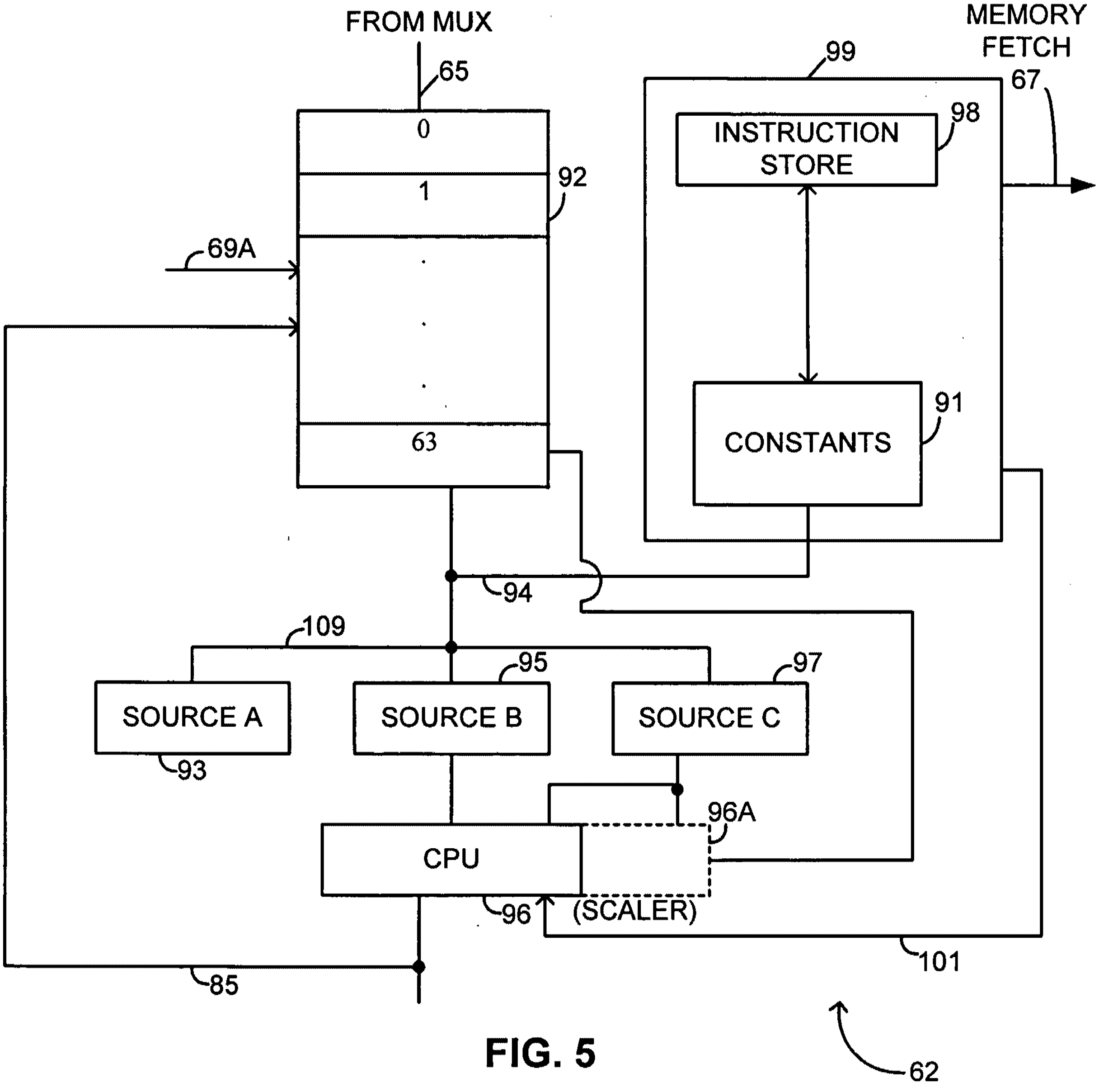

FROM MUX
65
0
1
92
63
69A
MEMORY FETCH
67
99
INSTRUCTION STORE
98
CONSTANTS
91
94
109
SOURCE A
93
SOURCE B
95
SOURCE C
97
96A
CPU
96
(SCALER)
101
85
62

FIG. 5

also @ TechSpot: Iranian scientist says he has built a machine that can predict the future

TECHSPOT
TECHNOLOGY NEWS AND ANALYSIS

TECHSPOT | NEWS | PRODUCTS | DOWNLOADS | FORUMS

Search

Home | Reviews | Guides | Product Finder | Forums | Downloads | Drivers | Extras | Sign in | About

CPU & Motherboards | Graphics | Desktops | Laptops | Tablets | Smartphones | Storage | Cases | Input | Software & Games

HOME › FEATURES

History of the Modern Graphics Processor, Part 3

By **Graham Singer** on April 10, 2013

Find and Compare the Best Tech Products

Read user comments (22)

58

Tweet

The Fall of 3Dfx and The Rise of Two Giants

With the turn of the century the graphics industry bore witness to further consolidation.

The pro market saw iXMICRO leave graphics entirely, while NEC and Hewlett-Packard both produced their last products, the TE5 and VISUALIZE FX10 series respectively. Evans & Sutherland also parted ways with the sale of its RealVision line to focus on the planetaria and fulldome projection systems.

In the consumer graphics market, ATI announced the acquisition of ArtX Inc. in February 2000, for around $400 million in stock. ArtX was developing the GPU codenamed Project Dolphin (eventually named “Flipper”) for the Nintendo GameCube, which added significantly to ATI’s bottom line.

Also in February, 3dfx announced a 20% workforce cut, then promptly moved to acquire Gigapixel for $186

TechSpot on: Follow

Most Popular

Trending | Featured

Hacker demonstrates how to hijack an airplane using an Android app
37 Comments

The History of the Modern Graphics Processor, Part 3: The Nvidia vs. ATI era begins
21 Comments

Iranian scientist says he has built a machine that can predict the future
15 Comments

PC industry posts steepest quarterly sales decline in history
21 Comments

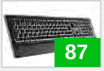
Next Xbox to feature AMD processor, not compatible with 360 titles
35 Comments

More Trending Topics

Featured on Keyboards

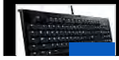
79

Metadot Das Keyboard Professional
6 reviews



87

Logitech Illuminated Keyboard
24 reviews



Razer BlackWidow Ultimate
24 reviews

million and gained the company's tile-based rendering IP.

Meanwhile, S3 and Nvidia settled their outstanding patent suits and signed a seven-year cross-license agreement.

ATI GameCube GPU

VIA assumed control of S3 around April-May which itself was just finishing a restructuring process from the acquisition of Number Nine. As part of S3's restructuring, the company merged with Diamond Multimedia in a stock swap valued at $165 million. Diamond's high-end professional graphics division, FireGL, was spun off as SONICblue and later sold to ATI in March 2001 for $10 million.

3DLabs acquired Intergraph's Intense3D in April, while the final acts of 3dfx played out towards the end of the year, despite 2000 kicking off with the promise of a better future as the long-awaited Voodoo 5 5500 neared its debut in July. The latter ended up trading blows with the GeForce 256 DDR and won the high-resolution battle.

> Where 3dfx was once a byword for raw performance, its strengths around this time laid in its full screen antialiasing image quality.

But where 3dfx was once a byword for raw performance, its strengths around this time laid in its full screen antialiasing image quality. The Voodoo 5 introduced T-buffer technology as an alternative to transformation and lighting, by basically taking a few rendered frames and aggregating them into one image. This produced a slightly blurred picture that, when run in frame sequence, smoothed out the motion of the animation.

3dfx's technology became the forerunner of many image quality enhancements seen today, like soft shadows and reflections, motion blur, as well as depth of field blurring.

3dfx's swan song, the Voodoo 4 4500, arrived October 19 after several delays – unlike the 4200 and 4800 that were never released. The card was originally scheduled for spring as a competitor to Nvidia's TNT2, but ended up going against the company's iconic GeForce 256 DDR instead, as well as the much better performing GeForce 2 GTS and ATI Radeon DDR.

On November 14, 3dfx announced they were belatedly ceasing production and sale of their own-branded graphics cards, something that had been rumoured for some time but largely discounted. Adding fuel to the fire, news got out that upcoming Pentium 4 motherboards would not support the 3.3V AGP signalling required Voodoo 5 series.

Email:

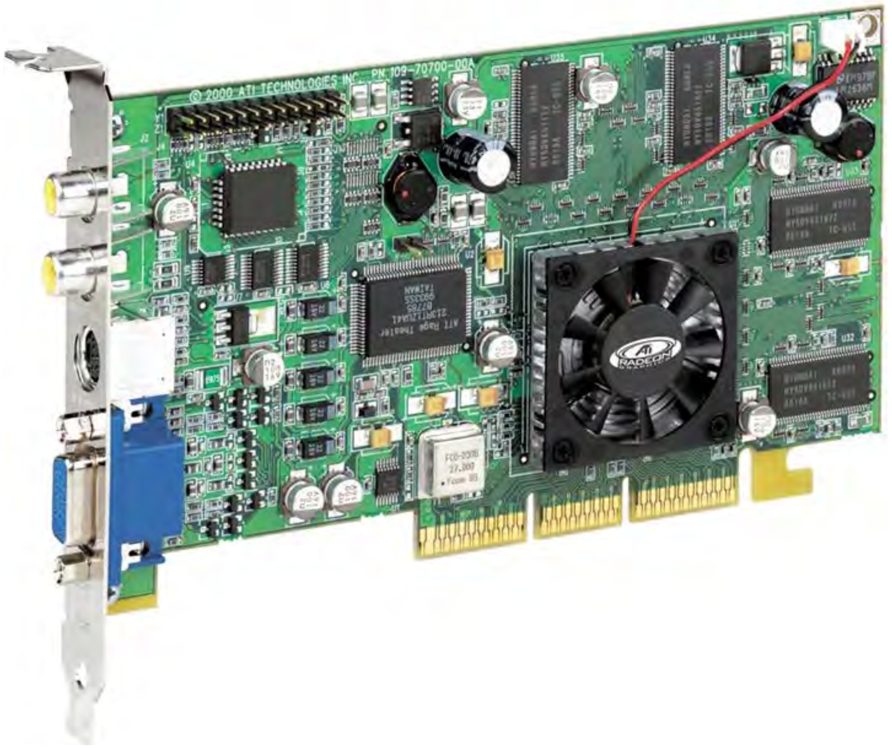
Voodoo5 5500 AGP box art

The death knell sounded a month later for 3dfx when Nvidia purchased its IP portfolio for $70 million plus one million shares of common stock. A few internet wits later noted that the 3dfx design team which had moved to Nvidia eventually got both their revenge and lived up to their potential, by delivering the underperforming NV30 graphics chip powering the FX 5700 and FX 5800 cards behind schedule.

The Nvidia vs. ATI Era Begins

Prior to the Voodoo 5's arrival, ATI had announced the Radeon DDR as "the most powerful graphics processor ever designed for desktop PCs." Previews of the card had already gone public on April 25, and only twenty-four hours later Nvidia countered with the announcement of the GeForce 2 GTS (GigaTexel Shader). The latter included Nvidia's version of ATI's Pixel Tapestry Architecture, named Nvidia Shading Rasterizer, allowing for effects such as specular shading, volumetric explosion, refraction, waves, vertex blending, shadow volumes, bump mapping and elevation mapping to be applied on a per-pixel basis via hardware.

The feature was believed to have made it to the previous NV10 (GeForce 256) chip but it remained disabled due to a hardware fault. The GTS also followed ATI's Charisma Engine in allowing for all transform, clipping and lighting calculations to be supported by the GPU. That said, ATI went a step further with vertex skinning for a more fluid movement of polygons, and keyframe interpolation, where developers designed a starting and finishing mesh for an animation and the Charisma core calculated the intervening meshes.



ATI Radeon DDR

The ATI Radeon DDR eventually launched for retail in August 2000. Backed by a superior T&L implementation and support for several of the upcoming DirectX 8 features, the Radeon DDR alongside the GeForce 2 GTS ushered in the use of DVI outputs by integrating support for the interface into the chip itself. The DVI output was more often found on OEM cards, however, as the retail variety usually sported VIVO plugs.

One downside to the Radeon DDR is that boards shipped with their core and memory downclocked from the promised 200MHz and 183MHz, respectively. In addition, drivers were once again less than optimal at launch. There were issues with 16-bit color and compatibility problems with VIA chipsets, but this did not stop the card from dominating the competition at resolutions higher than 1024x768x32. A price of $399 for the 64MB version stacked up well versus $349-399 for the 64MB GeForce 2 GTS, which it beat by a margin of 10-20% in benchmarks, and helped ATI maintain its number one position in graphics market share over Nvidia.

Nvidia wasn’t doing all that bad for themselves either. The company reported net income of $98.5 million for the fiscal year on record revenue of $735.3 million, driven in large part by its market segmentation strategy, releasing a watered-down MX version of the card in June and a higher clocked Ultra model in August. The latter dethroned the Radeon in terms of performance but it also cost $499. A Pro model arrived in December.

Besides releasing a GeForce 2 card at every price point, from the budget MX to the professional Quadro 2 range, Nvidia also released its first mobile chip in the form of the GeForce2 Go.

By the time 2001 dawned, the PC graphics market consisted of a discrete card duopoly, with both

As 3dfx was undergoing its death throes in November, Imagination Tech (ex-VideoLogic) and ST Micro attempted to address the high volume budget market with the PowerVR series 3 KYRO. Typically ranging in price from $80 to $110 depending on the memory framebuffer, the card represented good value for the money in gaming at resolutions of 1024x768 or lower. It would have become more popular, had the GeForce2 MX arrived later, or not so aggressively priced at ~$110.

The KYRO II arrived in April 2001 with a bump in clock speeds compared to the original and manufactured on a smaller 180nm process by ST Micro. But once again the card faced stiff competition from the GeForce 2 MX. Nvidia rebadged the card as the MX200 and lopped 40% off its price, while adding a higher clocked MX400 card at the same price as the Kyro II.

of them in addition to Intel supplying the vast majority of integrated graphics chipsets.

When PowerVR failed to secure game development impetus for tile based rendering, and ST Micro closed down its graphics business in early 2002, Imagination Technologies moved from desktop graphics to mobile and leveraged that expertise into system on chip graphics. They licenced the Series 5/5XT/6 for use with ARM-based processors in the ultra portable and smartphone markets.

By the time 2001 dawned, the PC graphics market consisted of a discrete card duopoly, with both of them in addition to Intel supplying the vast majority of integrated graphics chipsets.

Meanwhile, Matrox and S3/VIA clung to the margins of traditional markets.

Building on the strides made with the GeForce 2 series, Nvidia unveiled the GeForce 3 on February 27, 2001 priced between $339 and $449. The card became the new king of the hill, but it really only came into its own at the (then) extreme resolution of 1600x1200, preferably with full screen antialiasing applied.

Nvidia's stock GeForce 3 card

Initial drivers were buggy, especially in some OpenGL titles. What the new GeForce did bring to the table was DirectX 8, multisampling AA, quincunx AA (basically 2xMSAA + post process blur), 8x anisotrophic filtering as well as the unrivalled ability to handle 8xAF + trilinear filtering, and a programmable vertex shader which allowed for closer control of polygon mesh motion and a more fluid animation sequence.

There was also LMA (Lightspeed Memory Architecture) support -- basically Nvidia's version of HyperZ -- for culling pixels that would end up hidden behind others on screen (Z occlusion culling) as well as compressing and decompressing data to optimize use of bandwidth (Z compression).

Lastly, Nvidia implemented load-balancing algorithms as part of what they called the Crossbar Memory Controller, which consisted of four independent memory sub-controllers as opposed to the industry standard single controller, allowing incoming memory requests to be routed more effectively.

233 MHz NVIDIA Xbox Graphics Processing Unit
NVIDIA Xbox Media and Communications Processor

Nvidia NV2A inside Microsoft's Xbox

Nvidia's product line later added the NV2A, a derivative of the GeForce 3 with GeForce4 attributes that was used in Microsoft's Xbox game console.

At this point, Nvidia controlled 31% of the graphics market to Intel's 26% and ATI's 17%.

As Nvidia complemented the GF3 line-up with underclocked Ti 200 and overclocked Ti 500 models, ATI hurried to ramp up deliveries of the Radeon 8500. The card was built around the R200 GPU using TSMC's 150nm process (the same used by GeForce 3's NV20). The chip had been announced in August and was eagerly awaited since John Carmack of id software talked it up saying it would run the new Doom 3 "twice as well" as the GeForce 3.

ATI's official R8500 announcement was no less enthusiastic. But reality kicked in once the card launched in October and was found to perform at the level of the underclocked GF3 Ti 200 in games. Unfinished drivers and a lack of workable Smoothvision antialiasing weighted heavily against the R8500 in its initial round of reviews. By the time the holiday season arrived, a second round of reviews showed that the drivers had matured to a degree and raised the R8500's performance in-between the Ti 200 and the standard GF3.

Spec comparison snapshot

	Core clock (MHz)	Pixel pipelines	Fill rate (Mpixels/s	Texture units per pixel pipeline	Fill rate (Mtexels/s)	Memory clock (MHz)	Memory bus width (bits)	Memory bandwidth (GB/s)
GeForce3 Ti 200	175	4	700	2	1400	400	128	6.4
GeForce3	200	4	800	2	1600	460	128	7.4
GeForce3 Ti 500	240	4	960	2	1920	500	128	8.0
Radeon 64MB DDR	183	2	366	3	1100	366	128	5.9
Radeon 8500	275	4	1100	2	2200	550	128	8.8

Very competitive pricing and a better all around feature set (2D image quality, video playback, performance under antialiasing) made the card a worthy competitor to the GF3 and Ti 500 nonetheless.

ATI's sales for the year dropped to $1.04 billion as the company recorded a net loss of $54.2 million. The company began granting licenses to board partners to build and market graphics boards, while refocusing their resources on design and chip making.

ATI also debuted the Set-Top-Wonder Xilleon, a development platform based on the Xilleon 220 SoC which provided a full processor, graphics, I/O, video and audio for set-top boxes integrated into digital TV designs.

To complement Xilleon, ATI acquired NxtWave

ATI Xilleon board

Communications for $20 million in June 2002. The company specialized in digital signal processing and applications for set-top boxes and terrestrial digital solutions.

Keeping up with their product launch cycle, Nvidia released the GeForce 4 in February 2002. Three MX parts, three mobile parts based on the MX models, and two performance Titanium models (Ti 4400 and Ti 4600) made up the initial line up -- built on TSMC's 150nm process. The GeForce 4 was effectively ready for release two months earlier but the launch was delayed to avoid eating into GeForce 3 sales over the holiday season.

The MX series cards were intended for the budget segment but they were still largely uninspiring as they were based on the old GeForce 2 architecture. MPEG2 decode added but the cards reverted to DirectX 7.0/7.1 support as the earlier GF2 MX line. Pricing at $99-179 reflected the reduced feature set.

The Titanium models on the other hand were excellent performers and in some instances managed a 50+% increase in performance over the GeForce3 Ti 500. The Ti 4600 became the performance champ overnight, easily disposing of the Radeon 8500, while the Ti 4200 at $199 represented the best value for money card.

But then came the Radeon 9700 Pro and promptly consigned every other card to also-ran status.

ATI Radeon 9700 Pro (FIC A97P)

Developed by a team that had originally formed the core of ArtX, the ATI R300 GPU delivered spectacularly and arrived very promptly. It was the first to bring DirectX 9.0 support, and by extension, the first architecture to support shader model 2.0, vertex shader 2.0, and pixel shader

2.0. Other notable achievements: it was the second GPU series to support AGP 8x -- SiS's Xabre 80/200/400 line was first -- and implementing the first flip-chip GPU package.

ATI complemented the line-up in October by adding a non-Pro 9700 at $299 for those unable to part with $399 for the top model. Meanwhile, the cut down 9500 Pro ($199) and 9500 ($179) reached down through mainstream market segments, and the FireGL Z1/X1 filled in the $550-950 bracket for professional graphics. The All-In-Wonder 9700 Pro ($449) was also added in December.

About flip-chip GPU packages: Previous generations of graphics chips and other ICs used wire-bonding mounting. With this method, the chip sits on the board with the logic blocks sitting under the metal layers whose pads would be connected by thin wires arranged around the edges of the chip down to solder balls or pins on the underside. Flip–chip does away with the wire component through contact points (usually soldered in a ball grid array) directly on the "top" of the chip, which is then inverted, or "flipped" so that the solder points directly contact the substrate or circuit board. The chip then undergoes localised heating (reflow) to melt the solder that then forms the connection with the underlying contact points of the board.

ATI's sales are likely to have taken a hit when it was found that many cards could be modded to their more expensive counterparts. Examples of this included the ability to turn a 9500 card into a 9700 using its reference board (with the full complement of memory traces), or a 9800 Pro to its XT counterpart. For the latter, a driver patch was made available to check if it would accept the mod, which consisted of soldering in a resistor or using a pencil to tweak the GPU and memory voltage control chip. Hard mods also included upgrading various 9800 models into a FireGL X2, while a patched/Omega driver had the ability to turn a $250 9800 SE 256MB into a $499 9800 Pro 256MB.

In addition to discrete graphics, ATI also introduced desktop integrated graphics and chipsets. These included the A3/ IGP 320 meant to be paired with AMD CPUs, RS200/IGP 330 & 340 for Intel chips, as well as the mobile series U1/IGP 320M for AMD platforms and RS200M for Pentium 4-M. All of them were complemented with ATI southbridges, specifically the IXP200/250.

SiS unveiled the Xabre line between the launch of the GeForce4 and the R300. The cards were consistently slower than Nvidia and ATI's offerings at the same price points, and were handicapped by the lack of vertex shader pipelines. This translated into a heavy reliance upon drivers and game developers to get the most out of software emulation, thus keeping SiS in the margins of desktop discrete 3D graphics.

The Xabre line also implemented "Turbo Texturing", where framerates were increased by drastically reducing texture quality, and lacked anisotrophic filtering . All this did little to endear reviewers to the cards.

The Xabre line was the last under the SiS banner, as the company spun off its graphics division (renamed XGI) and merged with Trident Graphics a couple of months later in June.

The first of Nvidia's FX series arrived on January 27, 2003 with the infamous "Dustbuster" FX 5800 and the slightly faster (read: less slow) FX 5800 Ultra. When compared to the reigning champ, the ATI Radeon 9700 Pro (and non-Pro), the FX was much louder, it delivered inferior anisotrophic filtering (AF) quality and antialiasing (AA) performance, and was overall much slower. ATI was so far ahead that a second-tier Radeon 9700 card launched five months earlier comfortably outperformed the Ultra, and it was $100 cheaper ($299 vs $399).

> The 3dfx design team which had moved to Nvidia got both their revenge and lived up to their potential, by delivering the underperforming NV30 graphics chip

behind schedule.

The NV30 chip was supposed to debut in August, around the same time as the Radeon 9700, but ramping problems and high defect rates on TSMC's Low-K 130nm process held Nvidia back. Some circles also argued that the company was strapped for engineering resources, with more than a few tied up with the NV2A Xbox console chip, the SoundStorm APU, as well as the motherboard chipsets.

Looking to move things forward Nvidia undertook a project to have several FX series chips fabricated on IBM's more conventional Fluorosilicate glass (FSG) low-K 130nm process.

ATI refreshed its line of cards in March, starting with the 9800 Pro, featuring a R350 GPU that was basically an R300 with some enhancements to the Hyper-Z caching and compression instruction.

The RV350 and RV280 followed in April. The first of these, found inside the Radeon 9600, was built using the same TSMC 130nm low-K process that Nvidia had adopted, Meanwhile, the RV280 powering the Radeon 9200 was little more than a rebadged RV250 of the Radeon 9000 with AGP 8x support.

The same month saw ATI and Nintendo sign a technology agreement that would eventually lead to the Hollywood GPU for the Nintendo Wii console. ATI added a second console coup in August, when Microsoft awarded the Xbox 360 GPU contract to them.

A scant three and a half months after the inglorious debut of the FX 5800, Nvidia took another shot with the NV35 (FX 5900 and FX 5900 Ultra). The new Detonator FX driver greatly improved AA and AF, almost matching ATI's solution in terms of quality. However the 5900 achieved what the 5800 could not. It knocked ATI's Radeon 9800 Pro from its spot as the fastest card around, although at $499 apiece, few would actually take advantage of this.

Xbox 360 GPU (ATI C1 / Xenos)

As expected, ATI regained bragging rights in September with the release of the 9800 XT. Superior driver support – mainly with some DX9 games – also made the XT a better overall card than Nvidia's counterpart, ensuring that ATI ended the year with the performance crown. The 9700 Pro remained the standout mainstream board, while the FX 5700 Ultra at $199 won the sub-$200 price segment.

ATI bounced back with a $35.2 million profit in 2003 after posting a $47.5 million loss in 2002. A good chunk of this came from higher selling prices for the dominant 9800 and 9600 cards. Meanwhile, Nvidia retained 75% of the DirectX 9 value segment market, thanks to the popularity of the FX 5200.

Source DirectX 9.0 Effects Trailer, shown during ATI's presentation of the Radeon 9800 XT and 9600 XT

The newly formed XGI launched the Xabre successor in a staggered release between September and November. Renamed Volari, the card line-up ranged from the $49 V3 to the dual GPU Duo V8 Ultra. The V3 was virtually a rebrand of Trident's Blade XP4 and a DX 8.1 part, while the rest of the series (V5 and V8) was developed from the previous SiS Xabre and featured DX9.0 support.

For the most part, all of the models underdelivered, with the exception of the entry-level V3 which offered performance equal to the GeForce FX 5200 Ultra and and Radeon 9200. The Duo V8 Ultra was priced ~20% higher than the Radeon 9800 Pro 128MB, yet delivered performance on par or lower than the 9600XT.

> Another company making a comeback into desktop graphics was S3. Unfortunately, the buying public now generally saw desktop graphics as a two horse race – and S3 wasn't one of the two.

XGI's Volari line lingered on with the 8300 in late 2005, which was more or less on par with the Radeon X300SE/GeForce 6200 at $49, as well as the Z9/Z11 and XP10. The company was reabsorbed back into SiS in October 2010.

Another company making a comeback into desktop graphics was S3. After the graphics division was sold to VIA for $208 million plus the company's $60 million debt, the restructured venture concentrated primarily on chipset projects.

DeltaChrome desktop cards were announced in January, but in time-honoured S3 fashion, the first S4 and S8 models didn't start appearing in the retail channel until December. The new cards featured most of the new must-haves of 2003; DirectX 9 support, 16x AF, HD 1080p support, and portrait-mode display support.

Unfortunately, the buying public now generally saw desktop graphics as a two horse race – and S3 wasn't one of the two. While S3 was looking to keep competitive, ATI and Nvidia were driving each other to achieve ever-increasing levels of performance and image quality.

The DeltaChrome was succeeded by the GammaChrome in 2005.

Nvidia and ATI continued in 2005 their staggered launches. The former launched its first GDDR3 card in March as the FX 5700 Ultra, followed by the GeForce 6 series with the high-end 6800 range. The initial line up comprised the 6800 ($299), GT ($399), the Ultra ($499), and an overclocked variant known as the Ultra Extreme ($549) to counter ATI's X800 XT Platinum Edition. The latter was sold by a select band of add-in board partners.

The 6800 Ultra 512MB was added on March 14 2005 and sold for the unbelievable price of $899 -- BFG added an overclocked version for $999. The midrange was well catered for with the 6600 series in September.

Nvidia's feature set for the 6000 series included DirectX 9.0c support, shader model 3.0 (although the cards were never able to fully exploit this), Nvidia's PureVideo decode and playback engine, and SLI support -- the multi-GPU performance multiplier IP that was acquired from 3dfx.

Reintroducing an old feature: SLI

Where the 3dfx implementation resulted in each processing unit being responsible for alternate line scans, Nvidia handled things in a few different ways. The company implemented split frame rendering (SFR), in which each GPU rendered the top or bottom half of the frame, alternate frame rendering (AFR) so GPUs rendered frames in turn, and in some cases the driver just disabled SLI depending on whether the game supported the feature. This last feature was a hit-or-miss early in driver development.

While the technology was announced in June, it required a motherboard with an nForce4 chipset to enable multi-GPU setups, and these didn't start reaching the retail channel in numbers until late November. Adding fuel to the fire, initial driver releases where sporadic (at best) until into the following year.

Reviews at the time generally mirrored current performance, showing that two lower tier cards (like the 6600 GT SLI which could be had for $398) generally equalled one enthusiast card at lower resolutions and image quality. At highest resolutions and with antialiasing applied, however, single card setups still gained the upper hand. SLI and ATI's CrossFire performance was as erratic then as it sometimes is now, running the full gamut from perfect scaling to not working at all.

While Nvidia's SLI was announced in June 2004, the required nForce4 motherboards didn't hit the retail channel in numbers until November, and initial driver releases where sporadic until

Nvidia's board partners immediately saw marketing opportunities with the re-invented tech, with Gigabyte offering a dual 6600 GT SLI card (the 3D1), followed by a dual 6600 (3D1-XL), and into the following year.

the 6800 GT (3D1-68GT). These cards not only required an nF4 chipset but also a Gigabyte branded motherboard as well.

Of the high-end single GPU cards, the 6800 Ultra and X800 XT/XT PE were fairly evenly matched, both in price and performance. But they weren't without their issues. The latter arrived in May and suffered supply constraints throughout its entire production life, while Nvidia's flagship 6800 Ultra was extremely late arriving in August and suffered supply constraints too depending on distribution area, since the card was only made available by a percentage of board partners.

The 6800 GT generally bested the X800 Pro at $399, while the 6600 GT cleaned up in the $199 bracket.

Intense competition with Nvidia that year didn't have an adverse effect on ATI's bottom line, as profit peaked at $204.8 million for the year from nearly $2 billion in revenue.

One quirk associated with the well-received 6600 GT was that it initially launched as a PCI Express card, at a time when PCI-E was an Intel-only feature for motherboards designed for Pentium 4 processors. These chips generally lagged in gaming performance behind AMD's offerings, which of course used the AGP data bus.

Nvidia's 7000 series started rolling off the assembly lines well before the 6000 series had completed its model line-up. The 7800 GTX arrived a full five months before the reduced bill of materials (BoM) 6800 GS saw the light of day. The first iteration of the 7800 series was based around the G70 GPU on TSMC's 110nm process, but quickly gave way to the G71-based 7900 series, made on TSMC's 90nm process.

While the naming convention changed from "NV" to "G", the latter were architecturally related to the NV40 series of the GeForce 6000. And while only fractionally larger than the NV40-45 at 334mm², the G70 packed in an extra eighty million transistors (for a total of 302 million), adding a third more vertex pipelines and 50% more pixel pipelines. In most cases, the G70 was superseded within nine months, and in the case of the GS and GTX 512MB, the figure was 3 and 4 months respectively.

At the entry level, the 7100 GS continued the use of TurboCache (the ability for the board to use some system memory), which was introduced with the previous generation GeForce 6200 TC.

Nvidia GeForce 7800 GTX

At the other end of the spectrum, the 7800 GTX 256MB hit retail on June 22 with an MSRP of $599, though its actual street price was higher in many instances. ATI wrested the single-GPU crown back with the X1800 XT, but Nvidia countered with a 512MB version of the 7800 GTX thirty-five days later and promptly regained the title.

Two months later, ATI launched the X1900 XTX, which traded blows with Nvidia's flagship. This particular graphics horsepower race resulted in both cards being priced at $650. One spinoff of the cards moving to a 512MB frame buffer was that gaming at 2560x1600 with 32-bit color and a high level of image quality enabled was now possible via dual link DVI.

ATI's original CrossFire design required using an external Y cable

ATI announced their multi-card Crossfire technology in May 2005 and made it available in September with the launch of the Xpress 200 Crossfire Edition chipset, and X850 XT Crossfire Master board. Due to a single-link TMDS, resolution and refresh rates were initially limited to 1600x1200 @60Hz, but a dual-link TMDS for 2560x1600 would soon replace it.

Unlike Nvidia's solution of two identical cards communicating via a bridge connector, ATI implemented a master card with TMDS receiver, which accepted input from a slave card via external dongle and a Xilinx compositing chip.

Like Nvidia's SLI, CrossFire offered alternative frame rendering (AFR) and split frame rendering (SFR), but also a rendering technique called SuperTiling. The latter offered a performance increase in certain applications, but it did not work with OpenGL or support accelerated geometry processing. Also like SLI, Crossfire faced its share of driver-related troubles.

ATI intended to have their R520 based cards – their first to incorporate Shader Model 3.0 – ready by the June-July timeframe, but the late discovery of a bug in the cell library forced a 4 month delay.

Initial launches comprised the X1800 XL/XT using the R520 core, the X1300 budget cards using the RV515 with essentially one quarter of the graphics pipelines of the R520, and the X1600 Pro/XT based on the RV530, which was similar to the RV515 but with a higher shader and vertex pipeline-to-TMU and ROP ratio.

Due to the initial delay with the R520, the GPU and its derivations were being replaced a scant three and a half months later by the R580-based X1900 series which used TSMC's new 80nm process. Continuing with the roll out, half the graphics pipeline resources went into the RV570 (X1650 GT/XT and X1950 GT/Pro), while a shrunk RV530 became the RV535 powering the X1650 Pro as well as the X1300 XT.

ATI's revenue rose to a record $2.2 billion for the year, the highest in the company's history, aided by shipments of Xenos GPUs for the Xbox 360. Net profit, however, slumped to $16.9 million.

> ATI's revenue rose to a record $2.2 billion in 2005, the highest in the company's history, aided by shipments of Xenos GPUs for the Xbox 360. Net profit, however, slumped to $16.9 million.

By this stage, any graphics card launch not based on an Nvidia or ATI GPU was received with a certain amount of curiosity, if not enthusiasm. Such was the scene when S3's overhauled graphics line-up debuted in November.

The Chrome S25 and S27 promised good gaming performance based on their high clocks, but delivered a mostly sub-par product. Initial pricing at $99 (S25) and $115 (S27) put the cards in competition against Nvidia's 6600/6600GT and ATI's X1300Pro/X1600Pro, but neither S3 card stood up to the competition in any meaningful way, aside from power consumption. That slight advantage evaporated as ATI/AMD and Nvidia addressed the HTPC and entry-level market segment, effectively killing S3's subsequent Chrome 400 and 500 series.

An added issue for S3 was that the cost of building the cards resulted in razor thin profits. The company needed high volume sales in a market dominated by two vendors. HTC were to acquire S3 in July 2012 for $300 million, a move originally seen as leverage in HTC's and S3's separate legal disputes with Apple.

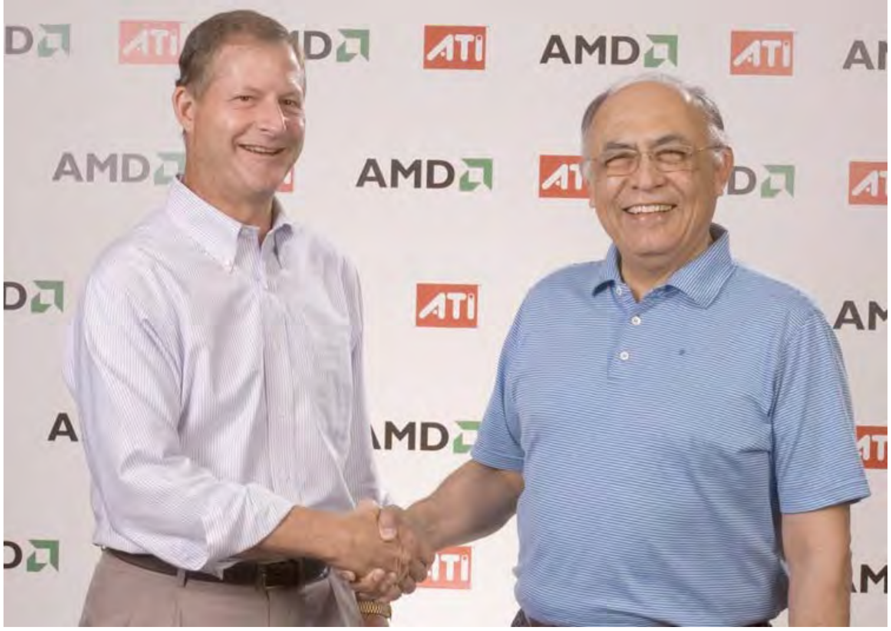
Nvidia and ATI continued to hog the press coverage in 2006.

ATI acquired Macrosynergy, a Shanghai based design and engineering centre with personnel working in California and previously part of the XGI group. Then in May the company bought BitBoys in a $44 million deal.

Meanwhile, Nvidia's first foray into dual-GPU single board products came in March, following in the footsteps of ATI, 3dfx, and XGI. The 7900 GX2 would sandwich two custom boards essentially carrying a couple of downclocked 7900 GTXs. But Asustek didn't wait around for Nvidia's dual-GPU solution, however, and released its own take as the Extreme N7800GT Dual ($900, 2000 units built), which paired two 7800 GT GPUs instead.

This card started Asus interest in limited edition dual-GPU boards, and possibly hardened Nvidia's attitude towards board partners', as Asustek products took the spotlight from their reference models at launch.

In the higher volume mainstream market, the 7600 GT and GS both provided solid performance and remarkable longevity, while ATI's X1950 XTX and Crossfire ruled the top end enthusiast benchmarks for single GPU cards. The X1900 XT and GeForce 7900 GT were fairly evenly matched in the upper mainstream bracket.



ATI's David Orton and AMD's Hector Ruiz officially announce the historic merger

After twenty-one years as an independent company, ATI was bought out by AMD on October 25 2006 for a total price of $5.4 billion – split between $1.7 billion from AMD, $2.5 billion borrowed from lending institutions, 57 million AMD shares and 11 million options/restricted stock units valued at $1.2 billion. At the time of the buy out, around 60-70% of ATI's chipset/IGP revenues were accrued from a partnership with Intel based motherboards.

Two weeks after the ATI buy-out, Nvidia ushered in the age of unified shader architectures for PC graphics.

With a large part of Intel's IGP chipset market moving to Nvidia, market share dropped dramatically. The logic behind the buy was a seemingly quick path to GPU technology, rather than use the $5.4 billion to develop AMD's own IP and add licenced technology where needed. At the time, AMD was aiming at the quick introduction of Torrenza and the associated Fusion projects.

Two weeks after the ATI buy-out, Nvidia ushered in the age of unified shader architectures for PC graphics. ATI's Xenos GPU for the Xbox 360 had already introduced the unified architecture to consoles.

This article is the third installment on a series of four. Next week we'll wrap things up, following the development of Radeon products under AMD's wing, the continued rivalry between GeForce and Radeon CPUs, the transition toward stream processing, and what the present a near future holds for graphics processors.

Part 1: (1976 - 1995) The Early Days of 3D Consumer Graphics

Part 2: (1995 - 1999) 3Dfx Voodoo: The Game-changer

Part 3: (2000 - 2006) The Nvidia vs. ATI Era Begins

Part 4: (2006 - Present) The Modern GPU: Stream processing units a.k.a. GPGPU

- **22 comments** - Read user comments, write your own
- Share this:
- Contact the author: Graham Singer
- Check out other recent features published at TechSpot

Back to TechSpot

Main Sections
Technology News
Reviews
Guides
Product Finder
Downloads
Drivers

Community
TechSpot Forums
Recent Topics
Recent Comments
Community Activity

Useful Resources
Trending Topics
Featured on TechSpot
PC Buying Guide
Hot Downloads
RSS Feeds

About TechSpot
About Us
Advertising
News Archive
The TechSpot Blog

How GPUs Work

David Luebke, NVIDIA Research
Greg Humphreys, University of Virginia

GPUs have moved away from the traditional fixed-function 3D graphics pipeline toward a flexible general-purpose computational engine.

In the early 1990s, ubiquitous interactive 3D graphics was still the stuff of science fiction. By the end of the decade, nearly every new computer contained a graphics processing unit (GPU) dedicated to providing a high-performance, visually rich, interactive 3D experience.

This dramatic shift was the inevitable consequence of consumer demand for videogames, advances in manufacturing technology, and the exploitation of the inherent parallelism in the feed-forward graphics pipeline. Today, the raw computational power of a GPU dwarfs that of the most powerful CPU, and the gap is steadily widening.

Furthermore, GPUs have moved away from the traditional fixed-function 3D graphics pipeline toward a flexible general-purpose computational engine. Today, GPUs can implement many parallel algorithms directly using graphics hardware. Well-suited algorithms that leverage all the underlying computational horsepower often achieve tremendous speedups. Truly, the GPU is the first widely deployed commodity desktop parallel computer.

THE GRAPHICS PIPELINE

The task of any 3D graphics system is to synthesize an image from a description of a scene—60 times per second for real-time graphics such as videogames. This scene contains the geometric primitives to be viewed as well as descriptions of the lights illuminating the scene, the way that each object reflects light, and the viewer's position and orientation.

GPU designers traditionally have expressed this image-synthesis process as a hardware pipeline of specialized stages. Here, we provide a high-level overview of the classic graphics pipeline; our goal is to highlight those aspects of the real-time rendering calculation that allow graphics application developers to exploit modern GPUs as general-purpose parallel computation engines.

Pipeline input

Most real-time graphics systems assume that everything is made of triangles, and they first carve up any more complex shapes, such as quadrilaterals or curved surface patches, into triangles. The developer uses a computer graphics library (such as OpenGL or Direct3D) to provide each triangle to the graphics pipeline one vertex at a time; the GPU assembles vertices into triangles as needed.

Model transformations

A GPU can specify each logical object in a scene in its own locally defined coordinate system, which is convenient for objects that are naturally defined hierarchically. This convenience comes at a price: before rendering, the GPU must first transform all objects into a common coordinate system. To ensure that triangles aren't warped or twisted into curved shapes, this transformation is limited to simple affine operations such as rotations, translations, scalings, and the like.

As the "Homogeneous Coordinates" sidebar explains, by representing each vertex in homogeneous coordinates, the graphics system can perform the entire hierarchy of transformations simultaneously with a single matrix-vector multiply. The need for efficient hardware to perform floating-point vector arithmetic for millions of vertices each second has helped drive the GPU parallel-computing revolution.

The output of this stage of the pipeline is a stream of triangles, all expressed in a common 3D coordinate system in which the viewer is located at the origin, and the direction of view is aligned with the *z*-axis.

Lighting

Once each triangle is in a global coordinate system, the GPU can compute its color based on the lights in the scene. As an example, we describe the calculations for a single-point light source (imagine a very small lightbulb). The GPU handles multiple lights by summing the contributions of each individual light. The traditional graphics pipeline supports the Phong lighting equation (B-T. Phong, "Illumination for Computer-Generated Images," *Comm. ACM*, June 1975, pp. 311-317), a phenomenological appearance model that approximates the look of plastic. These materials combine a dull diffuse base with a shiny specular high-

Homogeneous Coordinates

Points in three dimensions are typically represented as a triple (*x*,*y*,*z*). In computer graphics, however, it's frequently useful to add a fourth coordinate, *w*, to the point representation. To convert a point to this new representation, we set $w = 1$. To recover the original point, we apply the transformation (*x*,*y*,*z*,*w*) —> (*x*/*w*, *y*/*w*, *z*/*w*).

Although at first glance this might seem like needless complexity, it has several significant advantages. As a simple example, we can use the otherwise undefined point (*x*,*y*,*z*,0) to represent the direction vector (*x*,*y*,*z*). With this unified representation for points and vectors in place, we can also perform several useful transformations such as simple matrix-vector multiplies that would otherwise be impossible. For example, the multiplication

$$\begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

can accomplish translation by an amount D*x*, D*y*, D*z*.

Furthermore, these matrices can encode useful nonlinear transformations such as perspective foreshortening.

light. The Phong lighting equation gives the output color $C = K_d L_i (N \cdot L) + K_s L_i (R \cdot V)^s$.

Table 1 defines each term in the equation. The mathematics here isn't as important as the computation's structure; to evaluate this equation efficiently, GPUs must again operate directly on vectors. In this case, we repeatedly evaluate the dot product of two vectors, performing a four-component multiply-and-add operation.

Camera simulation

The graphics pipeline next projects each colored 3D triangle onto the virtual camera's film plane. Like the model transformations, the GPU does this using matrix-vector multiplication, again leveraging efficient vector operations in hardware. This stage's output is a stream of triangles in screen coordinates, ready to be turned into pixels.

Rasterization

Each visible screen-space triangle overlaps some pixels on the display; determining these pixels is called rasterization. GPU designers have incorporated many rasterizatiom algorithms over the years, which all exploit one crucial observation: Each pixel can be treated independently from all other pixels. Therefore, the machine can handle all pixels in parallel—indeed, some exotic machines have had a processor for each pixel. This inherent independence has led GPU designers to build increasingly parallel sets of pipelines.

Texturing

The actual color of each pixel can be taken directly from the lighting calculations, but for added realism, images called textures are often draped over the geometry to give the illusion of detail. GPUs store these textures in high-speed memory, which each pixel calculation must access to determine or modify that pixel's color.

In practice, the GPU might require multiple texture accesses per pixel to mitigate visual artifacts that can result when textures appear either smaller or larger on screen than their native resolution. Because the access pattern to texture memory is typically very regular (nearby pixels tend to access nearby texture image locations), specialized cache designs help hide the latency of memory accesses.

Hidden surfaces

In most scenes, some objects obscure other objects. If each pixel were simply written to display memory, the most recently submitted triangle would appear to be in front. Thus, correct hidden surface removal would require sorting all triangles from back to front for each view, an expensive operation that isn't even always possible for all scenes.

All modern GPUs provide a depth buffer, a region of memory that stores the distance from each pixel to the viewer. Before writing to the display, the GPU compares a pixel's distance to the distance of the pixel that's already present, and it updates the display memory only if the new pixel is closer.

THE GRAPHICS PIPELINE, EVOLVED

GPUs have evolved from a hardwired implementation of the graphics pipeline to a programmable computational substrate that can support it. Fixed-function units for transforming vertices and texturing pixels have been subsumed by a unified grid of processors, or *shaders*, that can perform these tasks and much more. This evolution has taken place over several generations by gradually replacing individual pipeline stages with increasingly programmable units. For example, the NVIDIA GeForce 3, launched in February 2001, introduced programmable vertex shaders. These shaders provide units that the programmer can use for performing matrix-vector multiplication, exponentiation, and square root calculations, as

Table 1. Phong lighting equation terms.

Term	Meaning
K_d	Diffuse color
L_i	Light color
N	Surface normal
L	Vector to light
K_s	Specular color
R	Reflected light vector
V	Vector to camera
s	"Shininess"

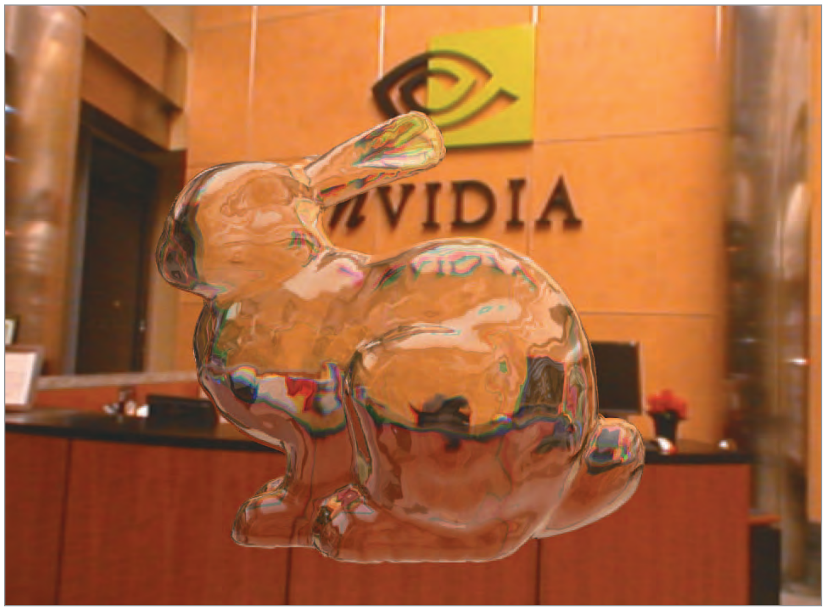


Figure 1. Programmable shading. The introduction of programmable shading in 2001 led to several visual effects not previously possible, such as this simulation of refractive chromatic dispersion for a "soap bubble" effect.

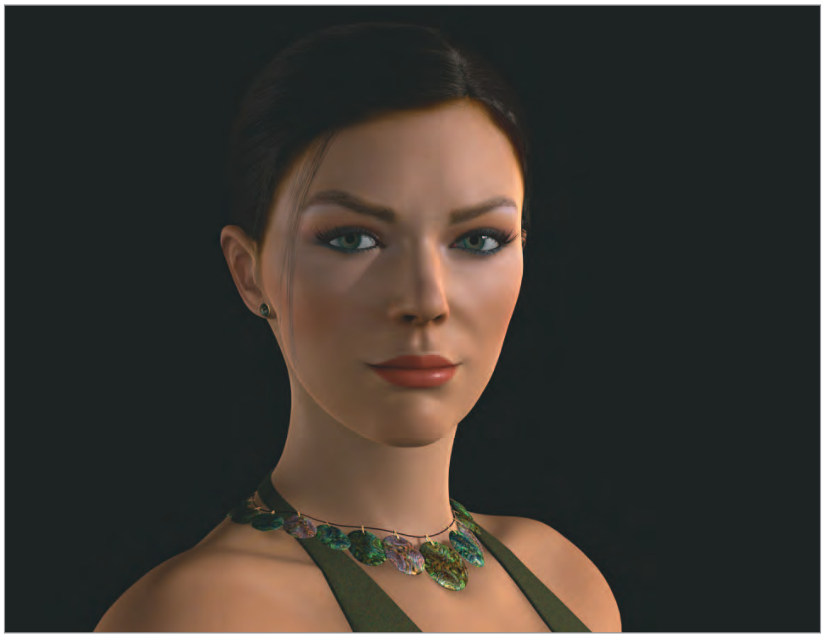


Figure 2. Unprecedented visual realism. Modern GPUs can use programmable shading to achieve near-cinematic realism, as this interactive demonstration shows, featuring actress Adrianne Curry on an NVIDIA GeForce 8800 GTX.

well as a short default program that uses these units to perform vertex transformation and lighting.

GeForce 3 also introduced limited reconfigurability into pixel processing, exposing the texturing hardware's functionality as a set of *register combiners* that could achieve novel visual effects such as the "soap-bubble" look demonstrated in Figure 1. Subsequent GPUs introduced increased flexibility, adding support for longer programs, more registers, and control-flow primitives such as branches, loops, and subroutines.

The ATI Radeon 9700 (July 2002) and NVIDIA GeForce FX (January 2003) replaced the often awkward register combiners with fully programmable pixel shaders. NVIDIA's latest chip, the GeForce 8800 (November 2006), adds programmability to the primitive assembly stage, allowing developers to control how they construct triangles from transformed vertices. As Figure 2 shows, modern GPUs achieve stunning visual realism.

Increases in precision have accompanied increases in programmability. The traditional graphics pipeline provided only 8-bit integers per color channel, allowing values ranging from 0 to 255. The ATI Radeon 9700 increased the representable range of color to 24-bit floating point, and NVIDIA's GeForce FX followed with both 16-bit and 32-bit floating point. Both vendors have announced plans to support 64-bit double-precision floating point in upcoming chips.

To keep up with the relentless demand for graphics performance, GPUs have aggressively embraced parallel design. GPUs have long used four-wide vector registers much like Intel's Streaming SIMD Extensions (SSE) instruction sets now provide on Intel CPUs. The number of such four-wide processors executing in parallel has increased as well, from only four on GeForce FX to 16 on GeForce 6800 (April 2004) to 24 on GeForce 7800 (May 2005). The GeForce 8800 actually includes 128 scalar shader processors that also run on a special *shader clock* at 2.5 times the clock rate (relative to pixel output) of former chips, so the computational performance might be considered equivalent to 128 ¥ 2.5/4 = 80 four-wide pixel shaders.

UNIFIED SHADERS

The latest step in the evolution from hardwired pipeline to flexible computational fabric is the introduction of unified shaders. Unified shaders were first realized in the ATI Xenos chip for the Xbox 360 game console, and NVIDIA introduced them to PCs with the GeForce 8800 chip.

Instead of separate custom processors for vertex shaders, geometry shaders, and pixel shaders, a unified shader architecture provides one large grid of data-parallel floating-point processors general enough to run all these shader workloads. As Figure 3 shows, vertices, triangles, and pixels recirculate through the grid rather than flowing through a pipeline with stages of fixed width.

This configuration leads to better overall utilization because demand for the various shaders varies greatly between applications, and indeed even within a single frame of one application. For example, a videogame might begin an image by using large triangles to draw the sky and distant terrain. This quickly saturates the pixel shaders in a traditional pipeline, while leaving the vertex shaders mostly idle. One millisecond later, the game might use highly detailed geometry to draw intricate characters and objects. This behavior will swamp the vertex shaders and leave the pixel shaders mostly idle.

These dramatic oscillations in resource demands in a single image present a load-balancing nightmare for the game designer and can also vary unpredictably as the players' viewpoint and actions change. A unified shader architecture, on the other hand, can allocate a varying percentage of its pool of processors to each shader type.

For this example, a GeForce 8800 might use 90 percent of its 128 processors as pixel shaders and 10 percent as vertex shaders while drawing the sky, then reverse that ratio when drawing a distant character's geometry. The net result is a flexible parallel architecture that improves GPU utilization and provides much greater flexibility for game designers.

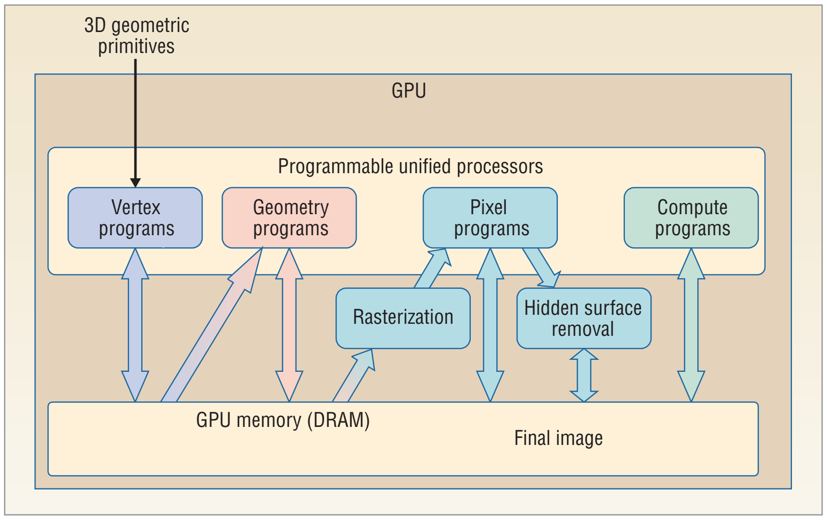


Figure 3. Graphics pipeline evolution. The NVIDIA GeForce 8800 GPU replaces the traditional graphics pipeline with a unified shader architecture in which vertices, triangles, and pixels recirculate through a set of programmable processors. The flexibility and computational power of these processors invites their use for general-purpose computing tasks.

GPGPU

The highly parallel workload of real-time computer graphics demands extremely high arithmetic throughput and streaming memory bandwidth but tolerates considerable latency in an individual computation since final images are only displayed every 16 milliseconds. These workload characteristics have shaped the underlying GPU architecture: Whereas CPUs are optimized for low latency, GPUs are optimized for high throughput.

The raw computational horsepower of GPUs is staggering: A single GeForce 8800 chip achieves a sustained 330 billion floating-point operations per second (Gflops) on simple benchmarks (http://graphics.stanford.edu/projects/gpubench). The ever-increasing power, programmability, and precision of GPUs have motivated a great deal of research on general-purpose computation on graphics hardware—GPGPU for short. GPGPU researchers and developers use the GPU as a computational coprocessor rather than as an image-synthesis device.

The GPU's specialized architecture isn't well suited to every algorithm. Many applications are inherently serial and are characterized by incoherent and unpredictable memory access. Nonetheless, many important problems require significant computational resources, mapping well to the GPU's many-core arithmetic intensity, or they require streaming through large quantities of data, mapping well to the GPU's streaming memory subsystem.

Porting a judiciously chosen algorithm to the GPU often produces speedups of five to 20 times over mature, optimized CPU codes running on state-of-the-art CPUs, and speedups of more than 100 times have been reported for some algorithms that map especially well.

Notable GPGPU success stories include Stanford University's Folding@home project, which uses spare cycles that users around the world donate to study protein folding (http://folding.stanford.edu). A new GPU-accelerated Folding@home client contributed 28,000 Gflops in the month after its October 2006 release—more than 18 percent of the total Gflops that CPU clients contributed running on Microsoft Windows since October 2000.

In another GPGPU success story, researchers at the University of North Carolina and Microsoft used GPU-based code to win the 2006 Indy PennySort category of the TeraSort competition, a sorting benchmark testing price/performance for database operations (http://gamma.cs.unc.edu/GPUTERASORT). Closer to home for the GPU business, the HavokFX product uses GPGPU techniques to accelerate tenfold the physics calculations used to add realistic behavior to objects in computer games (www.havok.com).

Modern GPUs could be seen as the first generation of commodity data-parallel processors. Their tremendous computational capacity and rapid growth curve, far outstripping traditional CPUs, highlight the advantages of domain-specialized data-parallel computing.

We can expect increased programmability and generality from future GPU architectures, but not without limit; neither vendors nor users want to sacrifice the specialized architecture that made GPUs successful in the first place. Today, GPU developers need new high-level programming models for massively multithreaded parallel computation, a problem soon to impact multicore CPU vendors as well.

Can GPU vendors, graphics developers, and the GPGPU research community build on their success with commodity parallel computing to transcend their computer graphics roots and develop the computational idioms, techniques, and frameworks for the desktop parallel computing environment of the future? ■

David Luebke is a research scientist at NVIDIA Research. Contact him at dluebke@nvidia.com.

Greg Humphreys is a faculty member in the Computer Science Department at the University of Virginia. Contact him at humper@cs.virginia.edu.

Computer welcomes your submissions to this bimonthly column. For additional information, or to suggest topics that you would like to see explained, contact column editor Alf Weaver at weaver@cs.virginia.edu.

Microsoft

News Center

Menu ≡ N

Microsoft and ATI Technologies Announce Technology Development Agreement

REDMOND, Wash., Aug. 14, 2003 — Microsoft Corp. (Nasdaq "MSFT") today announced it has entered into a technology development agreement with ATI Technologies Inc. (TSX: ATY, NASDAQ: ATYT). Under the agreement, ATI is developing custom, leading-edge graphics technologies for use in future Xbox® products and services.

"We're combining Microsoft's vision, software experience and R & D resources with ATI's pioneering leadership in graphics technologies to create innovative future Xbox products and services that meet the lifestyle needs of consumers in the Digital Decade,"
said Robbie Bach, senior vice president of the Home and Entertainment Division at Microsoft.
"We selected ATI after reviewing the top graphics technologies in development and determining that ATI's technical vision fits perfectly with the future direction of Xbox."

"Microsoft shares our passion for cutting-edge innovation,"
said K. Y. Ho, chairman and chief executive officer, ATI Technologies Inc.
"Our success working with Microsoft in the past gives us great confidence as we move forward, and our broad experience and wealth of engineering resources will ensure that we deliver. This agreement cements ATI's position as the prime graphics supplier for the future of the games industry."

About ATI Technologies Inc.

ATI Technologies Inc. is a world leader in the design and manufacture of innovative 3-D graphics and digital media silicon solutions. An industry pioneer since 1985, ATI is the world's foremost visual processor unit (VPU) provider and is dedicated to delivering leading-edge performance solutions for the full range of PC and Mac desktop and notebook platforms, workstation, set-top and digital television, game console, and handheld markets. With 2002 revenues in excess of U.S. $1 billion, ATI has more than 2,000 employees in the Americas, Europe and Asia. ATI common shares trade on Nasdaq (ATYT) and the Toronto Stock Exchange (ATY).

About Xbox

Xbox (http://www.xbox.com/) is Microsoft's future-generation video game system that delivers the most powerful games experiences ever. Xbox empowers game artists by giving them the technology to fulfill their creative visions as never before, creating games that blur the lines between fantasy and reality. Xbox is now available in the continents of North America, Europe, Asia and Australia.

About Microsoft

Founded in 1975, Microsoft (Nasdaq
"MSFT"
) is the worldwide leader in software, services and Internet technologies for personal and business computing. The company offers a wide range of products and services designed to empower people through great software — any time, any place and on any device.

Microsoft and Xbox are either registered trademarks or trademarks of Microsoft Corp. in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Note to editors : If you are interested in viewing additional information on Microsoft, please visit the Microsoft® Web page at http://www.microsoft.com/presspass/ on Microsoft's corporate information pages. Web links, telephone numbers and titles were correct at time of publication, but may since have changed. For additional assistance, journalists and analysts may contact Microsoft's Rapid Response Team or other appropriate contacts listed at http://www.microsoft.com/presspass/contactpr.asp .

Featured Posts

Microsoft and Dell partner to deliver Windows 10 devices and unmatched sales, services and support to the enterprise
ROUND ROCK, TX and REDMOND, WA – Sept. 8, 2015 – ... Read more »

Microsoft reinvents productivity with upcoming release of customer engagement solution
REDMOND, Wash. — Sept. 8, 2015 — Microsoft Corp. (Nasdaq ... Read more »

Xbox unveils more of its greatest games lineup in history
COLOGNE, Germany — Aug. 4, 2015 — Today Xbox opened gamescom ... Read more »

Most Popular

Microsoft and Dell partner to deliver Windows 10 devices and unmatched sales, services and support to the enterprise

Microsoft reinvents productivity with upcoming release of customer engagement solution

Windows 10 Upgrade Your World Initiative: Beagle Freedom Project wins; more chances to vote for your favorite nonprofits in 10 countries

Follow us:

Follow @MSFTNews 808K followers

Share this page:

0

Tweet 0

Other Microsoft sites

Windows

Office

Surface

Windows Phone

Mobile devices

Xbox

Skype

Bing

Microsoft Store

Downloads

Download Center

Windows downloads

Office downloads

Support

Support home

Knowledge base

Security

Security home

Microsoft Security Essentials

About

Microsoft

Careers

Company News

Microsoft Account

Investor relations

Site map

Popular resources

Windows Phone devices

Windows Phone app and games

Laptops and desktop computers

Microsoft computer security

Malware removal tool

Free antivirus software

Cloud computing solutions

Microsoft Dynamics CRM Online

News Center

Contact us　Privacy & Cookies　Terms of Use　Trademarks　About our ads

Search

Advanced search

Services

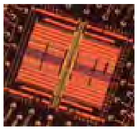
Print article
Send link to friend

Information

X-bit labs Year 2004 Poll

Once again X-bit labs asks you what you think about the companies participating in the hardware market and their results in 2004. Who is the leader and who is the follower? What is really fast and what is a marketing gimmick? Share your opinion with us - choose the best of the best in 2004.

Video



ATI and NVIDIA Proclaim Different Graphics Processors Architecture Goals

ATI Says Unified Rendering Engine – the Way to Go, NVIDIA Disagrees

by Anton Shilov
12/23/2004 | 07:55 AM

While particular approaches in graphics processing units design have been pretty different for leading computer visual companies ATI Technologies and NVIDIA Corp., in future the architecture of GPUs from the firms may be fundamentally different, as executives from both companies proclaim different approaches for chip internal architectures.

NVIDIA Disagrees with ATI Technologies

ATI Technologies' developer relations manager Richard Huddy said last month during a conference in London, UK, that the company's future visual processing units will feature unified pixel and shader processing. While he declined to elaborate on the timeframes for such chips, he said unified pixel and vertex data processing is a required capability for Windows Graphics Foundation 2.0 that comes out together with Microsoft's next-generation operating system called Windows Longhorn. On of the benefits the unified approach brings is ability to dynamically allocate chip resources depending on the demand for pixel and vertex processing, Mr. Huddy said. Another one is simplified software development.

NVIDIA Corp.'s chief architect David Kirk called the unified graphics engines as an implementation detail, not a feature, but admitted the unified architecture would be nice for programmers, who would have one instruction set for vertex and pixel shaders.

"It's not clear to me that an architecture for a good, efficient, and fast vertex shader is the same as the architecture for a good and fast pixel shader. A pixel shader would need far, far more texture math performance and read bandwidth than an optimized vertex shader. So, if you used that pixel shader to do vertex shading, most of the hardware would be idle, most of the time. Which is better – a lean and mean optimized vertex shader and a lean and mean optimized pixel shader or two less-efficient hybrid shaders? There is an old saying: 'Jack of all trades, master of none'," Mr. Kirk said in an interview with ExtremeTech web-site.

ATI: Bridging Today and Tomorrow

Not much is known about the architecture and capabilities of the code-named R520 product scheduled for release in Q2 2005 that was initially referred as the R500. What is clear now is that the new graphics chip will

News Archive

Video
December, 2004

			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

News Categories

- CPU
- Chipsets
- Memory
- Mainboards
- Video
- Storage
- Multimedia
- Mobile
- Other

Latest News

Thursday, December 23, 2004

- [5:17 pm | Video] ATI's Rialto Bridge: "Over the Next Couple of Months". ATI's AGP-to-PCI Express Bridge to be Ready Soon
- [2:22 pm | CPU] IBM's, Apple's Computers to Get Virtualization in 2005. IBM to Incorporate Virtualization Technologies into Power Chips Next Year
- [10:39 am | Other] Intel to Strengthen Concentration on Platform at Upcoming IDF. Intel Developer Forum Puts Tomorrow's Platforms Center Stage
- [7:55 am | Video] ATI and NVIDIA Proclaim Different Graphics Processors Architecture Goals. ATI Says

ATI 2080
LG v. ATI
IPR2015-00326

sport Shader Model 3.0 – pixel shaders 3.0 and vertex shaders 3.0 – bringing additional programming capabilities to ATI’s future graphics processors as well as some other innovations.

ATI’s R5xx architecture will not resemble that of the previous generation products and NVIDIA’s GeForce 6 architecture known as NV4x, particularly ATI will implement efficient flow-control, a crucial feature for pixel shaders 3.0, that will not bring speed penalty it does on existing SM3.0 hardware, according to sources. The future of the graphics hardware lies in higher number of ALUs ops per texture ops, unified pixel and vertex shaders as well as some other requirements of Microsoft Windows Longhorn operating system, such as virtualisation and context switches. While ATI agrees on the long-term goals for its roadmap, it does not name feature-set of actual products and says all the architectural changes will be implemented gradually, not at once.

Some sources claim that the R500 is a code-name of ATI’s graphics processor that will be submitted for Microsoft’s next Xbox console. The shader core of the R500 was reported to have 48 Arithmetic Logic Units (ALUs) that can execute 64 simultaneous threads on groups of 64 vertices or pixels. ALUs are automatically and dynamically assigned to either pixel or vertex processing depending on load. The ALUs can each perform one vector and one scalar operation per clock cycle, for a total of 96 shader operations per clock cycle. Texture loads can be done in parallel to ALU operations. At peak performance, the GPU can issue 48 billion shader operations per second, it was said.

The R520 is also expected to feature advanced memory interface, presumably supporting GDDR4 memory.

NVIDIA: Plans Unclear

While NVIDIA remains extremely tight-lipped over its future products, it is known that the company is readying its code-named NV47 visual processing unit, a massively revamped GeForce 6 architecture with 24 pixel pipelines. The NV47 is expected to be released sometime in Spring, 2005, but it is unknown whether NVIDIA is ahead, or behind ATI’s R520 product. NVIDIA also reportedly plans to release a chip called NV48 in Q2 2005.

The status of NVIDIA’s future architecture code-named NV50 is also uncertain: some reported recently that the chip had been cancelled, but officials decline to confirm or deny the information.

Related news:

- NVIDIA’s PureVideo Accelerates Media on the PC. NVIDIA Finally Delivers GeForce6 Video Engine
- Gigabyte Plugs Two Graphics Processors on One Graphics Card. Gigabyte 3D1 with Two GeForce 6600 GT Outperforms Top VPUs
- NVIDIA’s NV48 to be Made at TSMC, Says Article. NVIDIA's NV48 May Still Be in Plans, Contrary to Reports
- Industry Welcomes NVIDIA’s New Low-End Graphics Cards. NVIDIA's GeForce 6200 with TurboCache Set to Redefine Entry-Level
- Samsung Tests Memory for 512MB Graphics Cards. Samsung Samples 512Mb GDDR3 Chips
- NVIDIA Strikes PCI Express Graphics Cards Promotion Deal with Intel. Intel Corp. to Supply NVIDIA's Graphics Cards with its Mainboards
- NVIDIA Begins to Certify Hardware for Multi-GPU Deployments. NVIDIA Launches SLI Certification and Logo Program

Unified Rendering Engine – the Way to Go, NVIDIA Disagrees

- [5:51 am | Chipsets] Intel to Remove DDR2 Support from Forthcoming Mainstream Chipsets. Intel's 915PL and 915GL Chipsets to Lack DDR2 Support

News Archive

Sign In

Discussion

Comments currently: 8
Discussion started: 12/23/04
View comments

Add your Comment

Name/Nickname

Password

Registered users, please, enter your password. Register on our forums here.
E-mail

Your Comments

E-mail: info@xbitlabs.com

Contact information
Advertising
About us

Homepage
SiteMap
Terms & Conditions

Search

Advanced search

Services

Print article
Send link to friend

Video

NVIDIA Chief Architect: Unified Pixel and Vertex Pipelines – The Way to Go.

NVIDIA Says It Would Make a Chip with Unified Pipes "When it Makes Sense"

Category: Video

by Anton Shilov
[**07/11/2005** | 11:07 PM]

NVIDIA Corp.'s chief architect David Kirk said in an interview that the company would make a graphics processor with unified pixel and vertex processing engines – an approach backed by Microsoft Corp. and ATI Technologies, but denounced by NVIDIA earlier – in future. The executive said that from manufacturing standpoint such a chip would be easier to make, but harder to design.

"We will do a unified architecture in hardware when it makes sense. When it's possible to make the hardware work faster unified, then of course we will. It will be easier to build in the future, but for the meantime, there's plenty of mileage left in this architecture," David Kirk said in an interview with Bit-tech.net web-site.

Late last year ATI and NVIDIA revealed their standpoints on the so-called unified shader processing engines within a graphics chip. ATI said that such an approach was very efficient and would allow to dynamically allocate resources of a chip. The company has developed a graphics processor for Microsoft's Xbox 360 console, which actually has unified shader processors. NVIDIA late last year said that unified shader processor was not really efficient for both vertex and pixel processing compared to specifically optimized pipes.

Microsoft pushes unified shader language for pixel and vertex shaders in its next-generation Xbox 360 game console ad well as graphics API of Windows Longhorn – Windows Graphics Foundation 2.0. As a result of that graphics hardware designers should deliver their chips with unified shader engines at some point in future in order to more efficiently support the new API. But at this point NVIDIA thinks the approach is not something required tremendously in short-term future, which may be a feasible, as Longhorn is not expected to be launched earlier than in mid-2006.

"It's far harder to design a unified processor - it has to do, by design, twice as much. Another word for 'unified' is 'shared', and another word for 'shared' is 'competing'. It's a challenge to create a chip that does load balancing and performance prediction. It's extremely important, especially in a console architecture, for the performance to be predicable. With all that balancing, it's difficult to make the performance predictable," Mr. Kirk said.

Related news

- **NVIDIA's GeForce 7800 GTX Runs on Multiple Clock-Speeds.**
  NVIDIA's New GPU Operates at Different Frequencies
- **NVIDIA Preps More GeForce 7 Graphics Chips.**
  NVIDIA's G72, NV48 Tested by the Company
- **Graphics Card Makers Boost NVIDIA GeForce 7800 GTX Performance [UPDATED].**
  ASUS, Leadtek Overclock NVIDIA GeForce 7800 GTX
- **ATI Delays Next-Gen Graphics Processors Family.**
  ATI's R5-Series Possibly Postponed to August
- **Samsung Electronics Shows Off 2.0GHz GDDR3 Memory.**
  Samsung Develops Memory for Future Graphics Cards, Consoles
- **ASUS Cancels Dual-Chip GeForce 6800 Graphics Cards.**
  ASUS Decides Not to Proceed with Dual GeForce 6800 Graphics Cards
- **NVIDIA Shows "The Way It's Meant to Be Launched".**

News Archive

Video

July, 2005						
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						
< June, 2005				August, 2005 >		

News Categories

CPU
Chipsets
Memory
Mainboards
Video
Storage
Multimedia
Other
Coolers

Information

The results of our annual **2004 Readers' Choice poll** have been summarized and the winners have been finally identified. 12 prestigious awards have been given by over 20,000 of regular X-bit labs readers. Let's find out, who the winners are! The results of our Readers' Choice the Best of 2004 Poll are available here!

Graphics Company NVIDIA Unleashes GeForce 7800 GTX

Discussion

Comments currently: 0

Add your Comment

Name/Nickname

Your Comments

Mobile

Latest News

Friday, November 25, 2005

- 9:35 am | CPU AMD Set to Adopt New Interconnect Architecture, DDR3 in Three Years. AMD Promises to Improve Server Processors, Add DDR3 Support in 2008
- 7:59 am | Multimedia Customers Report Instabilities with Xbox 360. Some of the Xbox 360 Consoles Are Unstable, Users Claim
- 3:47 am | CPU Intel's Chips to Boost Processor System Bus Speed. Intel's Xeon Chips Set to Acquire 1333MHz Bus

Thursday, November 24, 2005

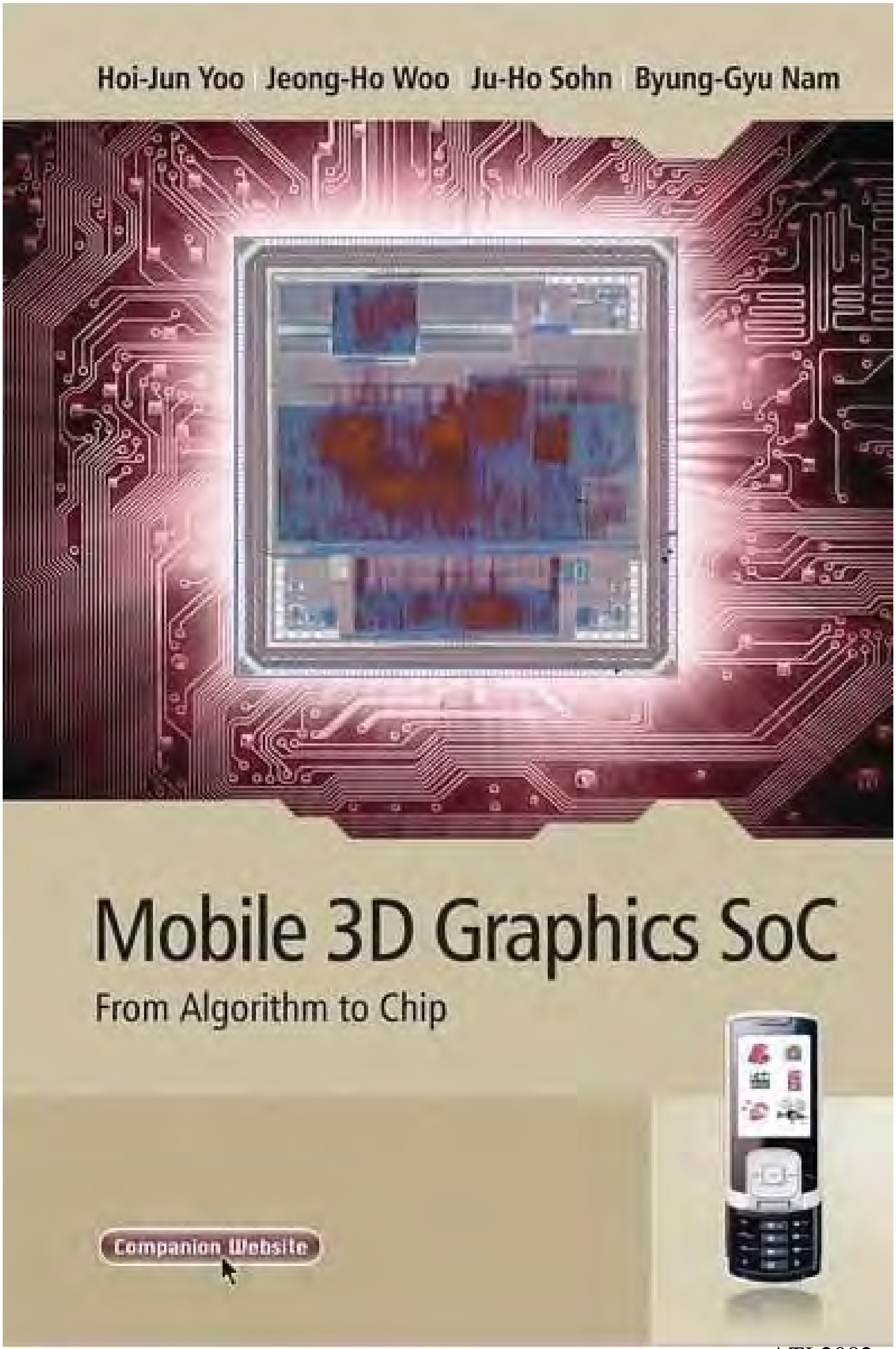
- 10:03 pm | Multimedia 40 Million Game Consoles to Be Sold in 2006 – Analysts. Console Market to Explode Next Year, Analysts Believe
- 9:42 pm | Chipsets Intel to Manufacture Chipsets Using 90nm Process Technology. Intel to Use 90nm Fabrication Process for Chipsets
- 1:06 pm | CPU AMD, Sun Intro Faster Dual-Core Server Processors. AMD, Sun Debut AMD Athlon 64 FX-60-Like Chips
- 8:47 am | Video GeForce 7800 GTX 512 Crusher: RADEON X1800 XT PE or R580? ATI Readies Competitor for the GeForce 7800 GTX 512

News Archive

Contact information
Advertising
About us

Homepage
SiteMap
Terms & Conditions

E-mail: info@xbitlabs.com



Hoi-Jun Yoo | Jeong-Ho Woo | Ju-Ho Sohn | Byung-Gyu Nam

Mobile 3D Graphics SoC

From Algorithm to Chip

Companion Website

MOBILE 3D GRAPHICS SoC

MOBILE 3D GRAPHICS SoC

From Algorithm to Chip

Jeong-Ho Woo
Korea Advanced Institute of Science and Technology, Republic of Korea
Ju-Ho Sohn
LG Electronics Institute of Technology, Republic of Korea
Byeong-Gyu Nam
Samsung Electronics, Republic of Korea
Hoi-Jun Yoo
Korea Advanced Institute of Science and Technology, Republic of Korea

John Wiley & Sons (Asia) Pte Ltd

Visit our Home Page on www.wiley.com

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold on the understanding that the Publisher is not engaged in rendering professional services. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

Other Wiley Editorial Offices

John Wiley & Sons, Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, UK

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstrasse 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 42 McDougall Street, Milton, Queensland 4064, Australia

John Wiley & Sons Canada Ltd, 5353 Dundas Street West, Suite 400, Toronto, ONT, M9B 6H8, Canada

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging-in-Publication Data

Mobile 3D graphics SoC : from algorithm to chip / Jeong-Ho Woo ... [et al.].
p. cm.
Includes index.
ISBN 978-0-470-82377-4 (cloth)
1. Computer graphics. 2. Mobile computing. 3. Systems on a chip. 4. Three dimensional display systems.
I. Woo, Jeong-Ho.
T385.M62193 2010
621.3815–dc22
2009049311

ISBN 978-0-470-82377-4 (HB)

Typeset in 10/12pt Times by Thomson Digital, Noida, India.
Printed and bound in Singapore by Markono Print Media Pte Ltd, Singapore.
This book is printed on acid-free paper responsibly manufactured from sustainable forestry in which at least two trees are planted for each one used for paper production.

Table 5.4 SRAM characteristics in 0.13 μm CMOS processor (200 MHz, 1.2 V)

Capacity	Size (μm^2)	AC current (mA)[a]	Peak current (mA)	Standby current (mA)
32 × 16 (512 b)	316 × 124	12.5	154.1	0.002
64 × 16 (1 Kb)	559 × 124	14.9	144.8	0.002
128 × 16 (2 Kb)	559 × 134	14.9	144.8	0.002
256 × 16 (4 Kb)	563 × 154	15.0	144.8	0.002
512 × 16 (1 KB)	568 × 195	15.0	145.0	0.002
1024 × 16 (2 KB)	753 × 125	13.0	183.4	0.003
32 × 32 (1 Kb)	559 × 124	20.8	271.5	0.003
64 × 32 (2 Kb)	1044 × 124	25.5	277.0	0.003
128 × 32 (4 Kb)	1044 × 134	25.6	277.1	0.002
256 × 32 (1 KB)	1044 × 154	25.6	277.2	0.002
512 × 32 (2 KB)	1053 × 195	25.6	277.4	0.002
1024 × 32 (4 KB)	1451 × 125	24.8	354.5	0.006

[a] AC current is measured at 25% read, 25% write, 50% idle state.

5.2.3 Mobile Unified Shader

As shaders have developed, vertex shaders and pixel shaders have similar instruction set architecture and register files, except for some unique instructions such as texture sampling. Therefore a unified shader architecture, which can compute vertex shading and pixel shading with the same hardware, has been developed to reduce

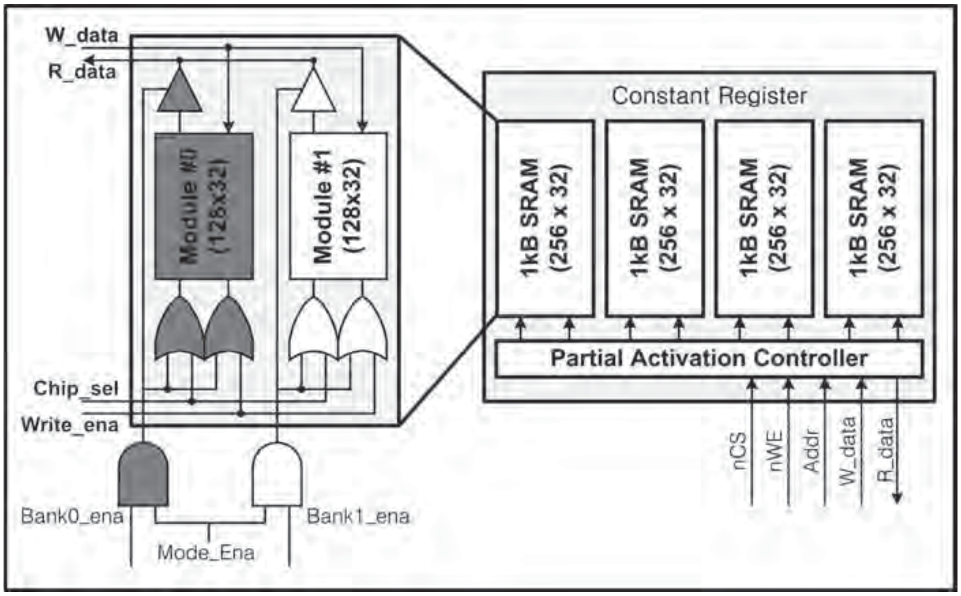


Figure 5.16 Partial activation

design complexity and turnaround time. The first unified shader was implemented in Xenos by ATI for X-Box 360 [26]. In PC and console devices, a few tens of unified shaders are integrated and controlled with multi-thread control. Therefore, programmable graphics operations can be mapped to those unified shaders dynamically in real time, and the 3D graphics processor with unified shader can utilize the hardware resources more efficiently than conventional architecture with vertex shader and pixel shader.

In the mobile environment, a fully programmable 3D graphics pipeline is required. Owing to the need for low power consumption and small area, the conventional architecture with separate vertex shader and pixel shader is hard to implement. Since a unified shader can compute vertex shading and pixel shading in a single hardware, it is a good solution for programmable 3D graphics [21]. Figure 5.17a shows the block diagram of the 3D graphics processor, in which the unified shader performs vertex and pixel shading and other blocks perform other operations of the 3D graphics pipeline such as clipping, rasterization, and blending.

The unified shader is a 4-way SIMD processor. It consists of 128 b, 4 × 32-bit SIMD datapath, a special functional unit, a texture engine, a low-power lighting engine, register files, and control logic, as shown in Figure 5.17b. The SIMD datapath is responsible for vector and matrix arithmetic operations such as addition (ADD), multiplication (MUL), and inner product (DOT), and the special functional unit is dedicated to special functional scalar operations such as logarithm (LOG), exponent

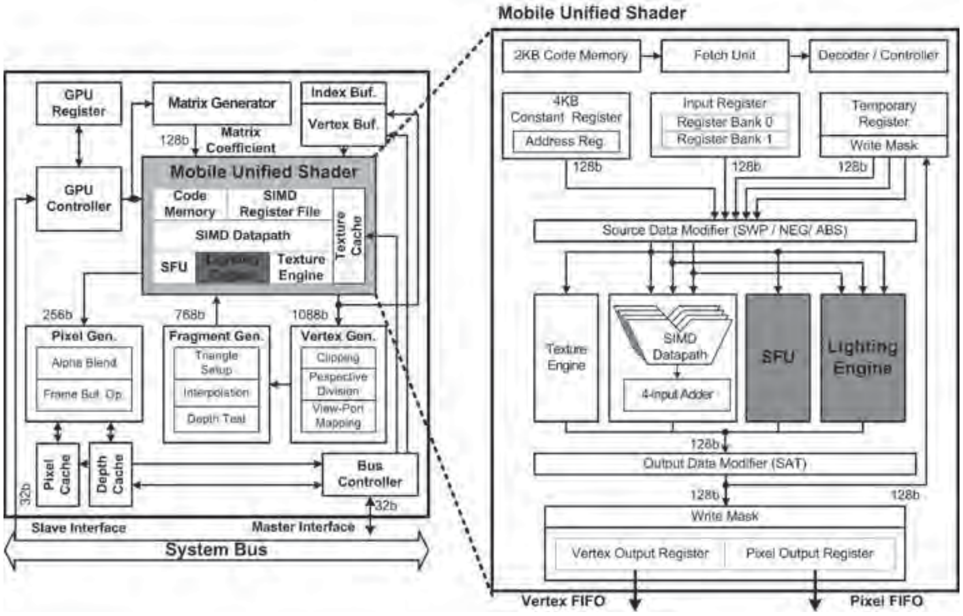


Figure 5.17 Mobile unified shader: (a) 3D graphics processor, and (b) mobile unified shader

(EXP), reciprocal (RCP), and reciprocal square-root (RSQ). The texture engine performs texture address generation, texture fetching, and texture filtering.

Since the single shader performs both vertex shading and pixel shading, task scheduling is crucial. Figure 5.18a shows a data flow diagram of the programmable 3D graphics pipeline. In conventional architecture, the primitive vertices are computed in the vertex shader for per-vertex operations such as transformation and lighting. After per-vertex operations, vertex generator and fragment generator perform clipping and

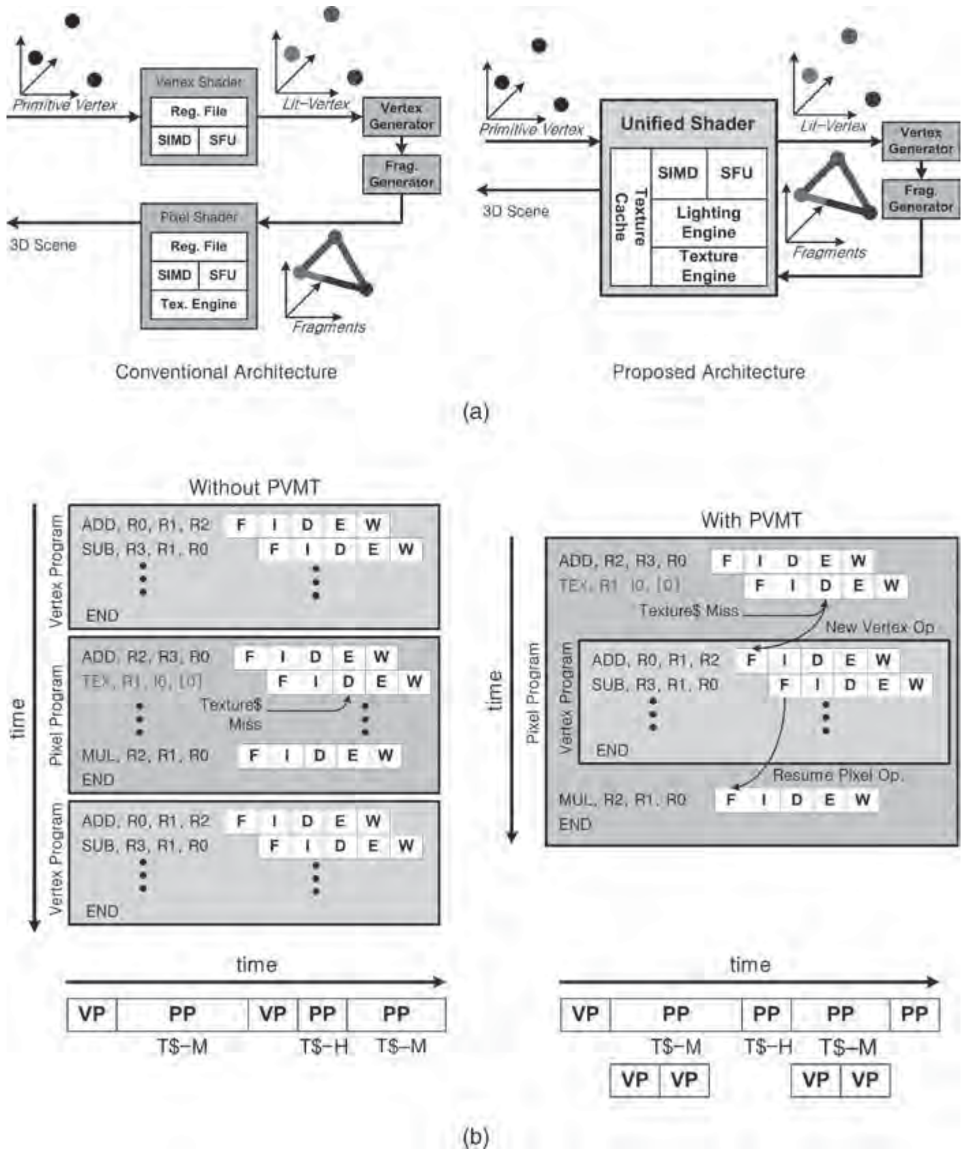
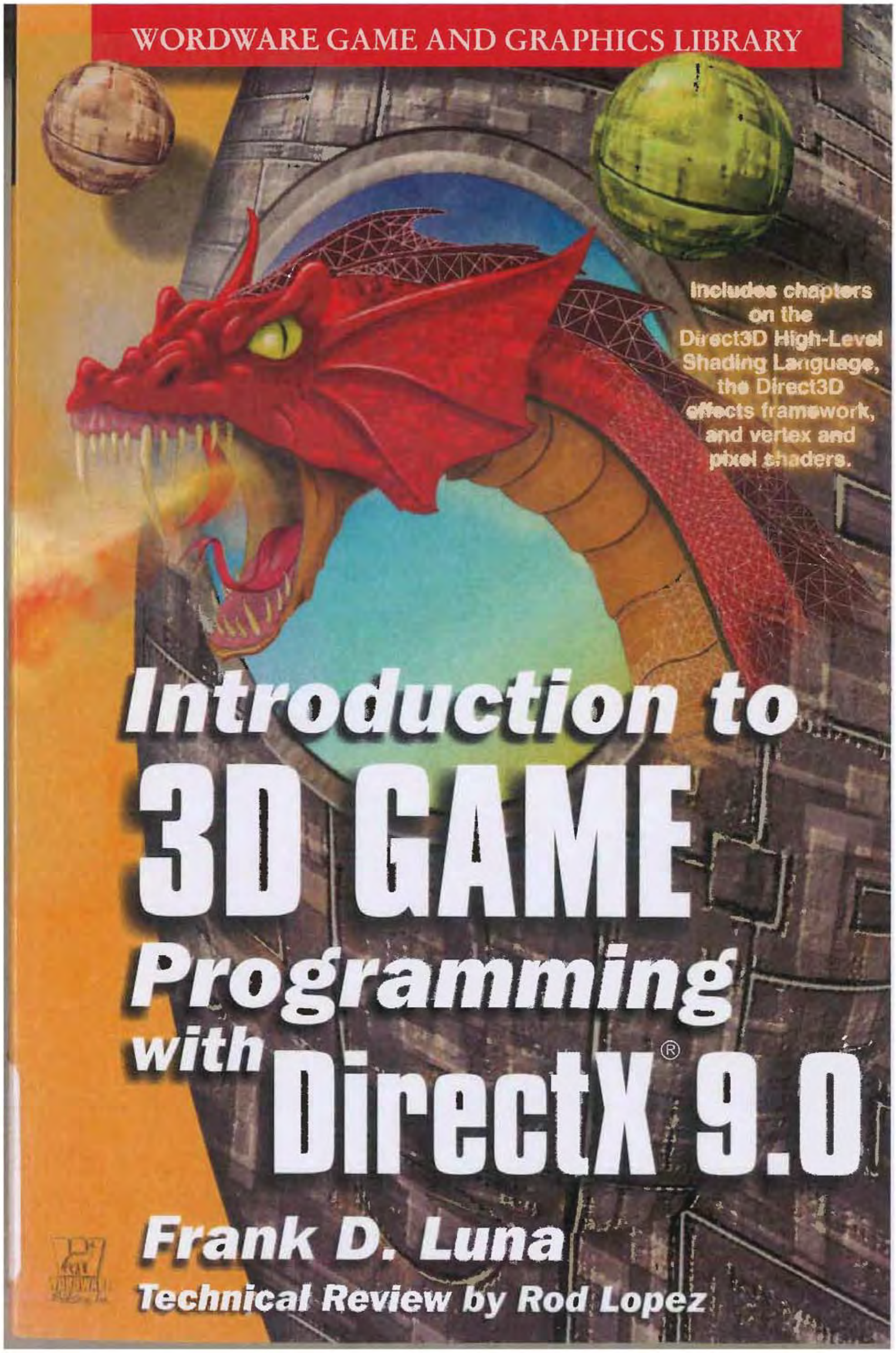


Figure 5.18 Pixel–vertex multi-threading: (a) data flow diagram, and (b) pixel–vertex multi-threading



WORDWARE GAME AND GRAPHICS LIBRARY

Includes chapters on the Direct3D High-Level Shading Language, the Direct3D effects framework, and vertex and pixel shaders.

Introduction to 3D GAME Programming with DirectX® 9.0

Frank D. Luna

Technical Review by Rod Lopez

Introduction to 3D Game Programming with DirectX® 9.0

Frank D. Luna

Technical review by Rod Lopez

Wordware Publishing, Inc.

Library of Congress Cataloging-in-Publication Data

Luna, Frank D.
Introduction to 3D game programming with DirectX 9.0 / by Frank D. Luna.
p. cm.
ISBN-10: 1-55622-913-5 (pbk.)
1. Computer games--Programming. 2. DirectX. I. Title.
QA76.76.C672L83 2003
794.8'15268--dc21 2003005834
CIP

2320 Los Rios Boulevard
Plano, Texas 75074

Printed in the United States of America

ISBN-13: 978-1-55622-913-8
ISBN-10: 1-55622-913-5

10 9 8 7 6 5
0305

All inquiries for volume purchases of this book should be addressed to Wordware Publishing, Inc., at the above address. Telephone inquiries may be made by calling:

(972) 423-0090

```
    const D3DXCOLOR     YELLOW( D3DCOLOR_XRGB(255, 255,   0) );
    const D3DXCOLOR       CYAN( D3DCOLOR_XRGB(  0, 255, 255) );
    const D3DXCOLOR    MAGENTA( D3DCOLOR_XRGB(255,   0, 255) );
}
```

4.2 Vertex Colors

The color of a primitive is determined by the color of the vertices that make it up. Therefore, we must add a color member to our vertex data structure. Note that a D3DCOLORVALUE type cannot be used here because Direct3D expects a 32-bit value to describe the color of a vertex. (Acually, by using a vertex shader we could use 4D color vectors for the vertex color, and thereby gain 128-bit color, but that is getting ahead of ourselves for now. Vertex shaders are covered in Chapter 17.)

```
struct ColorVertex
{
    float _x, _y, _z;
    D3DCOLOR _color;
    static const DWORD FVF;
}
const DWORD ColorVertex::FVF = D3DFVF_XYZ | D3DFVF_DIFFUSE;
```

4.3 Shading

Shading occurs during rasterization and specifies how the vertex colors are used to compute the pixel colors that make up the primitive. There are two shading modes that are presently used: flat shading and Gouraud shading.

With flat shading, the pixels of a primitive are uniformly colored by the color specified in the *first* vertex of the primitive. So the triangle formed by the following three vertices would be red, since the first vertex color is red. The colors of the second and third vertices are ignored with flat shading.

```
ColorVertex t[3];
t[0]._color = D3DCOLOR_XRGB(255, 0, 0);
t[1]._color = D3DCOLOR_XRGB(0, 255, 0);
t[2]._color = D3DCOLOR_XRGB(0, 0, 255);
```

Flat shading tends to make objects appear blocky because there is no smooth transition from one color to the next. A much better form of shading is called Gouraud shading (also called smooth shading). With Gouraud shading, the colors at each vertex are interpolated linearly across the face of the primitive. Figure 4.2 shows a red flat shaded triangle and a triangle colored using Gouraud shading.

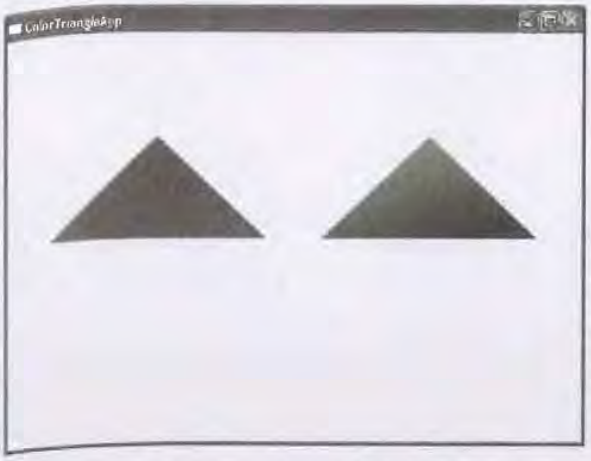


Figure 4.2: On the left is a triangle colored red with flat shading. On the right is a triangle with vertex colors red, green, and blue; notice that with Gouraud shading, the vertex colors are interpolated across the triangle.

Like many things in Direct3D, the shading mode is controlled through the Direct3D state machine.

```
// set flat shading
Device->SetRenderState(D3DRS_SHADEMODE, D3DSHADE_FLAT);

// set Gouraud shading
Device->SetRenderState(D3DRS_SHADEMODE, D3DSHADE_GOURAUD);
```

4.4 Sample Application: Colored Triangle

The sample program for this chapter demonstrates a triangle colored using flat shading and a triangle colored using Gouraud shading. It renders the image shown in Figure 4.2. First we add the following global variables:

```
D3DXMATRIX              World;
IDirect3DVertexBuffer9* Triangle = 0;
```

We include a D3DXMATRIX that is used to store the world transformation of the triangles that we are going to draw. The Triangle variable is the vertex buffer that stores the vertex data of a triangle. Notice that we only have to store the geometry of one triangle because we can draw it multiple times at different positions in the world using the world matrix.

The Setup method creates the vertex buffer and fills it with the data of a triangle with colored vertices. The first vertex in the triangle is full-intensity red (255), the second is full-intensity green (255), and the third is full-intensity blue (255). Finally, we disable lighting for this sample. Notice that this sample uses the new ColorVertex structure, as explained in section 4.2.

```
bool Setup()
{
    // create vertex buffer
    Device->CreateVertexBuffer(
        3 * sizeof(ColorVertex),
        D3DUSAGE_WRITEONLY,
        ColorVertex::FVF,
        D3DPOOL_MANAGED,
        &Triangle,
        0);

    // fill the buffers with the triangle data
    ColorVertex* v;
    Triangle->Lock(0, 0, (void**)&v, 0);

    v[0] = ColorVertex(-1.0f, 0.0f, 2.0f, D3DCOLOR_XRGB(255, 0, 0));
    v[1] = ColorVertex( 0.0f, 1.0f, 2.0f, D3DCOLOR_XRGB(0, 255, 0));
    v[2] = ColorVertex( 1.0f, 0.0f, 2.0f, D3DCOLOR_XRGB(0, 0, 255));

    Triangle->Unlock();

    // set projection matrix
    D3DXMATRIX proj;
    D3DXMatrixPerspectiveFovLH(
        &proj,
        D3DX_PI * 0.5f, // 90 - degree
        (float)Width / (float)Height,
        1.0f,
        1000.0f);
    Device->SetTransform(D3DTS_PROJECTION, &proj);

    // set the render states
    Device->SetRenderState(D3DRS_LIGHTING, false);

    return true;
}
```

Then, the `Display` function draws `Triangle` twice in two different positions and with different shade modes. The position of each triangle is controlled with the world matrix—`World`.

```
bool Display(float timeDelta)
{
    if( Device )
    {
        Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
                      0xffffffff, 1.0f, 0);
        Device->BeginScene();

        Device->SetFVF(ColorVertex::FVF);
        Device->SetStreamSource(0, Triangle, 0, sizeof(ColorVertex));
        // draw the triangle to the left with flat shading
        D3DXMatrixTranslation(&World, -1.25f, 0.0f, 0.0f);
        Device->SetTransform(D3DTS_WORLD, &World);

        Device->SetRenderState(D3DRS_SHADEMODE, D3DSHADE_FLAT);
        Device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 1);

        // draw the triangle to the right with gouraud shading
        D3DXMatrixTranslation(&World, 1.25f, 0.0f, 0.0f);
        Device->SetTransform(D3DTS_WORLD, &World);

        Device->SetRenderState(D3DRS_SHADEMODE, D3DSHADE_GOURAUD);
        Device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 1);

        Device->EndScene();
        Device->Present(0, 0, 0, 0);
    }
    return true;
}
```

4.5 Summary

- Colors are described by specifying an intensity of red, green, and blue. The additive mixing of these three colors at different intensities allows us to describe millions of colors. In Direct3D, we can use the `D3DCOLOR`, the `D3DCOLORVALUE`, or the `D3DXCOLOR` type to describe a color in code.
- We sometimes treat a color as a 4D vector (r, g, b, a). Color vectors are added, subtracted, and scaled just like regular vectors. On the other hand, dot and cross products do *not* make sense for color vectors, but component-wise multiplication does make sense for colors. The symbol $\otimes$ denotes component-wise multiplication, and it is defined as: $(c_1, c_2, c_3, c_4) \otimes (k_1, k_2, k_3, k_4) = (c_1k_1, c_2k_2, c_3k_3, c_4k_4)$.
- We specify the color of each vertex, and then Direct3D uses the current shade mode to determine how these vertex colors are used to compute the pixel colors of the triangle during rasterization.
- With flat shading, the pixels of a primitive are uniformly colored by the color specified in the *first* vertex of the primitive. With Gouraud shading, the colors at each vertex are interpolated linearly across the face of the primitive.

Example:

```
Device->SetTexture(0, _stonewall);
```

Note: In Direct3D, you can set up to eight textures that can be combined to create a more detailed image. This is called multitexturing. We do not use multitexturing in this book until Part IV; therefore we always set the texture's stage to 0, for now.

To disable a texture at a particular texturing stage, set `pTexture` to 0. For instance, if we don't want to render an object with a texture, we would write:

```
Device->SetTexture(0, 0);
renderObjectWithoutTexture();
```

If our scene has triangles that use different textures, we would have to do something similar to the following code:

```
Device->SetTexture(0, _tex0);
drawTrisUsingTex0();

Device->SetTexture(0, _tex1);
drawTrisUsingTex1();
```

6.3 Filters

As mentioned previously, textures are mapped to triangles in screen space. Usually, the texture triangle is not the same size as the screen triangle. When the texture triangle is smaller than the screen triangle, the texture triangle is *magnified* to fit. When the texture triangle is larger than the screen triangle, the texture triangle is *minified* to fit. In both cases, distortion will occur. *Filtering* is a technique Direct3D uses to help smooth out these distortions.

Direct3D provides three different types of filters; each one provides a different level of quality. The better the quality, the slower it is, so you must make the trade-off between quality and speed. Texture filters are set with the `IDirect3DDevice9::SetSamplerState` method.

- **Nearest point sampling**—This is the default filtering method and produces the worst-looking results, but it is also the fastest to compute. The following code sets nearest point sampling as the minification and magnification filter:

```
Device->SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_POINT);
Device->SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_POINT);
```

- **Linear filtering** –This type of filtering produces fairly good results and can be done very fast on today's hardware. It is

Once a light is registered, we can turn it on and off using what this next example illustrates:

```
Device->LightEnable(
    0,      // the element in the light list to enable/disable
    true); // true = enable, false = disable
```

5.5 Sample Application: Lighting

The sample for this chapter creates the scene shown in Figure 5.7. It demonstrates how to specify vertex normals, how to create a material, and how to create and activate a directional light. Note that in this sample program we do not make use of the d3dUtility.h/cpp material and light functionality code because we want to show how it is done manually first. However, the rest of the samples in this book do use the material and light utility code.

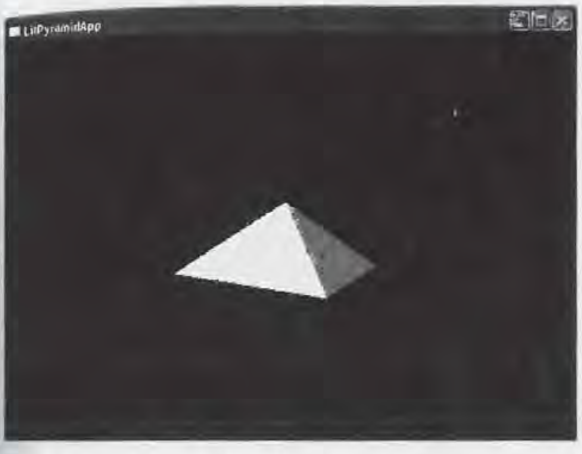


Figure 5.7: Screen shot from the LitPyramid sample

The steps for adding light to a scene are:

1. Enable lighting.
2. Create a material for each object and set the material before rendering the corresponding object.
3. Create one or more light sources, set the light sources, and enable them.
4. Enable any additional lighting states, such as specular highlights.

First we instantiate a global vertex buffer that stores the pyramid's vertices:

```
IDirect3DVertexBuffer9* Pyramid = 0;
```

The `Setup` function contains all the code relevant to this chapter, so we omit the other functions to save space. It implements the previously discussed steps to add lighting to a scene. The `Setup` method starts by enabling lighting, which isn't necessary because it's enabled by default (but it doesn't hurt either).

```
bool Setup()
{
    Device->SetRenderState(D3DRS_LIGHTING, true);
```

Next, we create the vertex buffer, lock it, and specify the vertices that form triangles of the pyramid. The vertex normals were precomputed using the algorithm covered in section 5.3. Notice that while the triangles share vertices, they do not share normals; thus it is not very advantageous to use an index list for this object. For example, all the triangles share the peak point (0, 1, 0); however, for each triangle, the peak vertex normal points in a different direction.

```
Device->CreateVertexBuffer(
          12 * sizeof(Vertex),
          D3DUSAGE_WRITEONLY,
          Vertex::FVF,
          D3DPOOL_MANAGED,
          &Pyramid,
          0);

     // fill the vertex buffer with pyramid data
     Vertex* v;
     Pyramid->Lock(0, 0, (void**)&v, 0);

     // front face
     v[0] = Vertex(-1.0f, 0.0f, -1.0f, 0.0f, 0.707f, -0.707f);
     v[1] = Vertex( 0.0f, 1.0f,  0.0f, 0.0f, 0.707f, -0.707f);
     v[2] = Vertex( 1.0f, 0.0f, -1.0f, 0.0f, 0.707f, -0.707f);

     // left face
     v[3] = Vertex(-1.0f, 0.0f,  1.0f, -0.707f, 0.707f, 0.0f);
     v[4] = Vertex( 0.0f, 1.0f,  0.0f, -0.707f, 0.707f, 0.0f);
     v[5] = Vertex(-1.0f, 0.0f, -1.0f, -0.707f, 0.707f, 0.0f);

     // right face
     v[6] = Vertex( 1.0f, 0.0f, -1.0f, 0.707f, 0.707f, 0.0f);
     v[7] = Vertex( 0.0f, 1.0f,  0.0f, 0.707f, 0.707f, 0.0f);
     v[8] = Vertex( 1.0f, 0.0f,  1.0f, 0.707f, 0.707f, 0.0f);

     // back face
     v[9]  = Vertex( 1.0f, 0.0f,  1.0f, 0.0f, 0.707f, 0.707f);
     v[10] = Vertex( 0.0f, 1.0f,  0.0f, 0.0f, 0.707f, 0.707f);
     v[11] = Vertex(-1.0f, 0.0f,  1.0f, 0.0f, 0.707f, 0.707f);

     Pyramid->Unlock();
```

After we have generated the vertex data of our object, we describe how the object interacts with light by describing its materials. In this sample, the pyramid reflects white lights, emits no light, and produces some highlights.

```
D3DMATERIAL9 mtrl;
    mtrl.Ambient  = d3d::WHITE;
    mtrl.Diffuse  = d3d::WHITE;
    mtrl.Specular = d3d::WHITE;
    mtrl.Emissive = d3d::BLACK;
    mtrl.Power    = 5.0f;

    Device->SetMaterial(&mtrl);
```

Second to last, we create and enable a directional light. The directional light rays run parallel to the x-axis in the positive direction. The light emits strong white diffuse light (dir.Diffuse = WHITE), weak white specular light (dir.Specular = WHITE * 0.3f), and a medium amount of white ambient light (dir.Ambient = WHITE * 0.6f).

```
D3DLIGHT9 dir;
    ::ZeroMemory(&dir, sizeof(dir));
    dir.Type      = D3DLIGHT_DIRECTIONAL;
    dir.Diffuse   = d3d::WHITE;
    dir.Specular  = d3d::WHITE * 0.3f;
    dir.Ambient   = d3d::WHITE * 0.6f;
    dir.Direction = D3DXVECTOR3(1.0f, 0.0f, 0.0f);

    Device->SetLight(0, &dir);
    Device->LightEnable(0, true);
```

Finally, we set the state to renormalize normals and enable specular highlights.

```
    Device->SetRenderState(D3DRS_NORMALIZENORMALS, true);
    Device->SetRenderState(D3DRS_SPECULARENABLE, true);

    // ... code to set up the view matrix and projection matrix
    // omitted

    return true;
}
```

5.6 Additional Samples

Three additional samples are included for this chapter in the companion files. They use the D3DXCreate* functions to create the 3D objects that compose the scene. The D3DXCreate* functions create vertex data with the format D3DFVF_XYZ | D3DFVF_NORMAL. In addition, these functions compute the vertex normals of each mesh for us. The additional samples demonstrate how to use directional lights, point lights, and spotlights. Figure 5.8 shows a screen shot from the directional light sample.

OpenGL - Lighting, Material,Shading and Texture Mapping

CS475 - Computer Graphics
Sumair Ahmed
IIT Bombay

August 28, 2009

ATI 2084
LG v. ATI
IPR2015-00326

This document briefs about the ways to deal with the prominent features existing in OpenGL. Here we discuss the Lighting, Shading,Material and Texture Mapping in programming perspective of OpenGL.

Lighting

Light makes the scene look real. Effectively, in a real world, light is present everywhere. Objects also have their own reaction in presence of a light. OpenGL supports a number of lighting effects - Directional, Spot, Ambient lights and attenuation. The Lighting and the Shading models define the light in OpenGL. But before setting these up, lighting mode has to be enabled. Lighting support needs the depth buffer to be enabled.

$$glEnable\,(GL\_DEPTH\_TEST)\,;$$
$$glEnable\,(GL\_LIGHTING)\,;$$

The above function *glEnable* can enable many features of OpenGL; the feature you want to enable is provided as an input parameter. The above code enables the lighting, however we need lights as well. OpenGL has direct support for about 8 lights . To enable a light, call:

$$glEnable\,(GL\_LIGHT0)\,;$$

Likewise, $LIGHT1$ or 2..8 shall be enabled when you want to handle multiple lights. Similar to *glEnable* function, OpenGL also has *glDisable* function that disables the features set before by *glEnable*. These functions turn a particular feature ON or OFF. OpenGL also provides a set of specific global properties to specify the visual behavior of the lighting model. This is done in two ways:

$$glLightModelf(GLenum\ pname, GLfloat\ param);\ //\ \text{scalar params}$$
$$glLightModelfv(GLenum\ pname, const\ GLfloat\ params);\ //\ \text{vector params}$$

For Instance, suppose you want to have a $GLOBAL$ Ambient light that casts on all the rendered objects. The following code sets the *global_ambient* colour on all the objects.

$$Glfloat\ global\_ambient[] = \{0.5f, 0.5f, 0.5f, 1.0f\}\ \ //\ \text{R,G,B,Alpha}$$
$$glLightModelfv\,(GL\_LIGHT\_MODEL\_AMBIENT, global\_ambient)\,;$$

Similarly, we have $GL\_LIGHT\_MODEL\_TWO\_SIDE$ and GL_LIGHT_MODEL $\_LOCAL\_VIEWER$ parameters. The first parameter defines if the light is to be applied on two sides of an object or one side. Second defines how specular component is calculated. The specular highlight depends on the direction from the vertex to the viewpoint and also the direction from the vertex to the light source. So, the highlight depends on the eye position. With a default infinite viewpoint, the direction from the vertex to the viewpoint remains the same for all vertices. 0 is considered as infinite viewpoint, while 1 is local view point.

$$glLightModelf\,(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, GL\_TRUE)\,;$$

There are two types of light properties you should consider while programming the lighting model in OpenGL. The first type describes a light source and the second type describes the light reflected by the material of an object's surface. These four independent models define the light - $GL\_AMBIENT$, $GL\_DIFFUSE$, $GL\_SPECULAR$ and $GL\_EMISSIVE$

For every type of light source you need to call the $glLightfv$ function with parameters. For example, To add a component of specular light to a light source, you would make the following function call:

$$GLfloat\ specular[] = \{1.0f, 1.0f, 1.0f, 1.0f\};$$
$$glLightfv\,(GL\_LIGHT0, GL\_SPECULAR, specular)\,;$$

The first three parameters of *specular* variable are the RGB values which can range between 0.0f and 1.0f. 0 being no colour, and 1 being full colour. The final parameter, Alpha, is used to define the translucency factor of an object and is used to simulate materials made out of translucent matter such as glass. The colour format here is often referred as RGBA format.

OpenGL also provides the light attenuation feature, that reduces the light intensity with the distance. Light Intensity should decrease with the distance to mimic the real world. It shall be inversely related to the distance. The Intensity attenuation formula would be :

$$\begin{aligned}\text{Attenuated Intensity} &= \text{Intensity of light Source} * \tfrac{1}{C+L*d+Q*d^2}\\ &= \text{Intensity of light Source} * \text{Attenuation Factor}\end{aligned}$$

where

$$\begin{gathered}\text{d} = \text{distance between light source and the vertex}\\ \text{C} = \textit{GL\_Constant\_Attenuation} \text{ (default is 1)}\\ \text{L} = \textit{GL\_Linear\_Attenuation} \text{ (default is 0)}\\ \text{Q} = \textit{GL\_Quadratic\_Attenuation} \text{ (default is 0)}\end{gathered}$$

are the three attenuation provided by OpenGL. In order to have an attenuation with the distance, you should set to a value different to 0 the linear or quadratic. It calculates an attenuation factor (between 0 and 1) which is multiplied to the ambient, diffuse and specular colours. By default, attenuation factor is 1, it means there is no attenuation w.r.t the distance.

$$glLightf(GL\_LIGHT0, GL\_LINEAR\_ATTENUATION, 0.2f);$$

But assigning ambient, diffuse and specular types of light to a light source is not usually enough. You also have to specify the position of the light source. You can have Directional light source instead of the below Positional light source.

$$\begin{gathered}Glfloat\ position1[] = \{-1.5f, 1.0f, -4.0f, 1.0f\};\ //\text{x,y,z,w}\\ Glfloat\ position2[] = \{-1.05f, 1.0f, -1.0f, 0.0f\};\ //\text{x,y,z,w}\\ glLightfv\,(GL\_LIGHT0, GL\_POSITION, position1)\,;\\ glLightfv\,(GL\_LIGHT0, GL\_POSITION, position2)\,;\end{gathered}$$

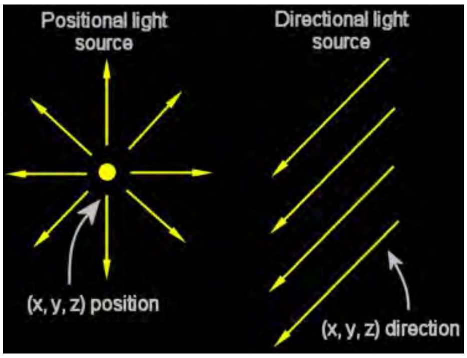


Figure 1: Positional and Directional light source

The light source can be a positional (w > 0) or directional (w = 0) light source depending on the w value. A positional light source is positioned at the location (x, y, z) as shown above. The source emits light from that particular location towards all directions. for example: lamp, bulb.

A directional one does not have any location. The source emits light from an infinite location, the rays are all parallel and have the direction (x, y, z). A directional light is not subject to attenuation since it is at an infinite distance. for example: Sun.

Shading

The Shading model is set up with a call to *glShadeModel* and can be either set to $SMOOTH$ or $FLAT$ model. The $SMOOTH$ shading model specifies the use of Gouraud-shaded polygons to describe light while the $FLAT$ shading model specifies the use of single-colored polygons.

$$glShadeModel\,(GL\_SMOOTH)\,;$$

When $SMOOTH$ model is selected, Lighting is evaluated at each vertex, and pixel colours are linearly interpolated across polygons. However, in $FLAT$ model, Lighting is evaluated once for a polygon, and the resulting colour value is used for the complete object.

Material

This defines the reaction of an object when its surface is hit with the light. For example, some objects absorb a particular colour or reflects light. Usually when the lighting is enabled it is equally likely to assign material properties with the $glMaterialf$ command as shown in the following code sample:

$$Glfloat\ mcolor[] = \{1.0f, 0.0f, 0.0f, 1.0f\};$$
$$glMaterialfv\,(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, mcolor)\,;$$

The material property of an object (defined by RGB colour format), is usually the colour reflected by that object. The first parameter of the $glMaterialfv$ command indicates which face of the polygon should reflect the light specified by *mcolor*. Apparently, there are two sides to a polygon - front and back. OpenGl provides two ways to specify a polygon in 3D space in order to decide the front face. The clockwise or counterclockwise direction describes which side is the front and which is the back. OpenGL lets you specify these rules with the glFrontFace command. The Following code denotes that counter clockwise direction direction of the polygons is considered to be Front Face.

$$glFrontFace(GL\_CCW);$$

glMaterial command should be called prior to defining the polygon's vertices to apply these material properties to the surface.

A convenient alternative to *glmaterial* is color tracking. Material properties are specified by merely calling the *glColor* command prior to each object or polygon. Also it has to be enabled.

$$glEnable\,(GL\_COLOR\_MATERIAL)\,;$$
$$glColorMaterial\,(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE)\,;$$
$$glColor3f(0.0f, 0.0f, 1.0f);\text{// blue reflective properties}$$

Example: This code would typically be placed with the OpenGL initialization code

//set the global lighting / shading
$glShadeModel\,(GL\_SMOOTH)\,;$ // or GL_FLAT
$glEnable\,(GL\_NORMALIZE)\,;$
$glEnable\,(GL\_LIGHTING)\,;$

//set the global ambient light
$GLfloat\ ambient = \{.2, .2, .2, 1\};$
$glLightModelfv\,(GL\_LIGHT\_MODEL\_AMBIENT, globalAmb)\,;$

//set up a light and enable it
$GLfloat\ diffuse[] = \{1, 0, 0, 1\};$
$GLfloat\ ambient[] = \{.5, 0, 0, 1\};$
$GLfloat\ specular[] = \{1, 1, 1, 1\};$

$glLightfv\left(GL\_LIGHT0, GL\_DIFFUSE, diffuse\right);$
$glLightfv\left(GL\_LIGHT0, GL\_AMBIENT, ambient\right);$
$glLightfv\left(GL\_LIGHT0, GL\_SPECULAR, specular\right);$
$glEnable\left(GL\_LIGHT0\right);$ //enable the light

//set light position
// set last term to 0 for a spotlight
$Glfloat\ lightpos[] = \{1, 1, 1, 1\};$
$glLightfv\left(GL\_LIGHT0, GL\_POSITION, lightpos\right);$

//This code sets a simple material property
$GLfloat\ ambient[] = \{.5, 0, 0, 1\};$
$GLfloat\ specular[] = \{1, 1, 1, 1\};$

//set params for front and back separately (GL_BACK, GL_FRONT_AND_BACK)
$glMaterialfv\left(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, ambient\right);$
$glMaterialfv\left(GL\_FRONT, GL\_SPECULAR, ambient\right);$

Example2: A Cube in an environment with two diffuse lights and an ambient light. Two Diffuse lights are of different colours (blue & green) whereas the ambient light is red colour.

$GLfloatDiffuseLight1[] = \{0, 0, 1\};$
$GLfloatDiffuseLight2[] = \{0, 1, 0\};$
$GLfloatAmbientLight[] = \{1, 0, 0\};$
$glLightfv\left(GL\_LIGHT0, GL\_DIFFUSE, DiffuseLight1\right);$
$glLightfv\left(GL\_LIGHT1, GL\_AMBIENT, AmbientLight\right);$
$glLightfv\left(GL\_LIGHT2, GL\_DIFFUSE, DiffuseLight2\right);$
$GLfloatLightPosition1[] = \{0, 0, 3, 0\};$
$GLfloatLightPosition2[] = \{3, 0, 0, 0\};$
$glLightfv\left(GL\_LIGHT0, GL\_POSITION, LightPosition1\right);$
$glLightfv\left(GL\_LIGHT2, GL\_POSITION, LightPosition2\right);$
$gluLookAt\left(3.0, -2.0, 4.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0\right);$

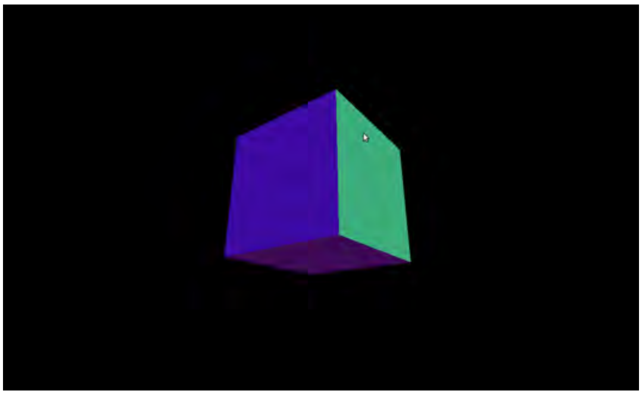


Figure 2: Output of Example2

You can see three colours on three sides of a cube. There is one diffuse light at position (3,0,0) and the other at (0,0,3). There is also an Ambient light of red colour. The eye position is at (3,-2,4) and is looking at (0,0,0) i.e., the center of cube.

Texture Mapping

Another feature in OpenGL is Texture Mapping feature where you can apply the textures to your geometry. Once a texture is uploaded to the video memory it can be used throughout the program. There are certain steps to be followed before a texture is readily available to the program. We first need a texture name. This is essentially a number that OpenGL uses to index all the different textures.

$$GLuint\ texture;\ \text{// allocate a texture name}$$
$$glGenTextures\,(1, \&texture)\,;\ \text{//get a free texture id}$$

Now that we have our texture name, it has to be bound before doing anything to it. Note that there are two forms of textures in OpenGL, 1D and 2D. You can load different textures, however only one is selected at a time.

$$\text{// select our current texture}$$
$$glBindTexture\,(GL\_TEXTURE\_2D, texture)\,;$$

Now we need to set some texture parameters and load the texture data on the current texture. OpenGL has four texture parameters to setup. Here, it defines several effects like bilinear, trilinear texture filtering, and mipmapping. We also can define whether the texture wraps over at the edges or is clamped at the ends.

// the texture wraps over at the edges (repeat)
$glTexParameteri\,(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT)\,;$
$glTexParameteri\,(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT)\,;$

// when texture area is large, bilinear filter the original
$glTexParameteri\,(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR)\,;$
$glTexParameteri\,(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR)\,;$

The default state of MIN_FILTER is $GL\_LINEAR\_MIPMAP\_NEAREST$, if it is not defined. In such a case, the texture is considered incomplete and it renders a white texture on the object.

Also we need to set environment variables for the current texture. This tells the OpenGL how the texture should act when it is rendered into a scene.

$glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);$

Here it sets the active texture to $GL\_MODULATE$. This attribute allows to apply effects such as lighting and colouring to your texture. If you would like to display the texture unchanged then replace it with GL_DECAL.

After all these parameters are set, OpenGL calls $glTexImage2D$ that will upload the texture to the video memory and will be ready for us to use in our programs.

$glTexImage2D(GL\_TEXTURE\_2D, level, internalFormat, width, height,$
$border, format, type, ptexels);$

- $internalFormat$ - This tells OpenGL how many colour components are needed to represent internally from the texture that is uploaded.ex: GL_RGB
- $format$ - Format of the pixel data that will be uploaded. ex: GL_RGB
- $type$ - Type of data that will be uploaded. ex: $GL\_UNSIGNED\_BYTE$
- $ptexels$ - Pointer to the image data.

Note that after your call to glTexImage2D you can free this memory with $free$ function since the texture is already uploaded into video memory. A good alternative to $glTexImage2D$ is to build your texture mipmaps. This can be done by:

$$gluBuild2DMipmaps(GL\_TEXTURE\_2D, 3, width, height, GL\_RGB,\\ GL\_UNSIGNED\_BYTE, data);$$

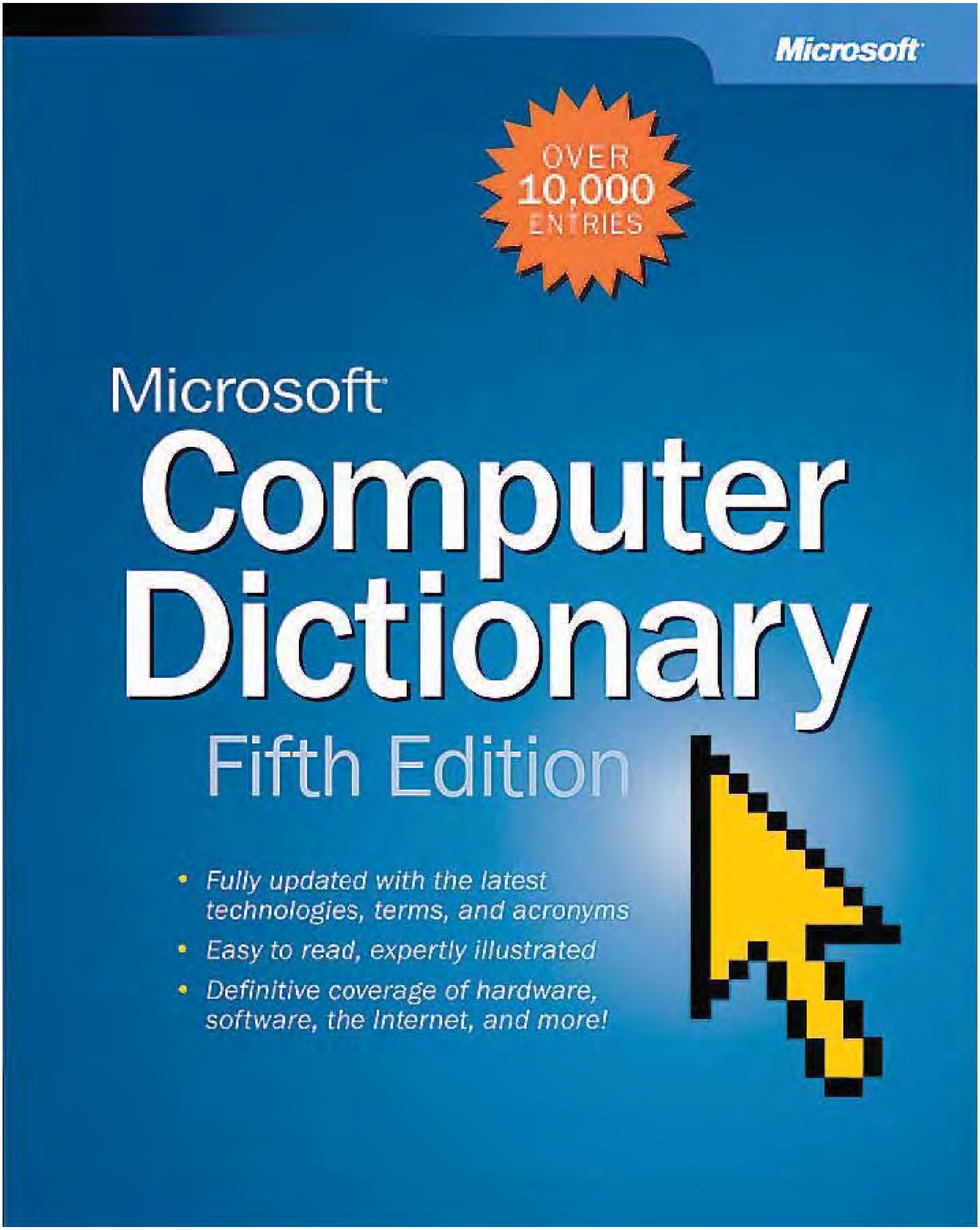
Now the texture is ready to be applied to your geometry, with all the above parameters set. Remember Texturing has to be enabled.

Example: Texture Quad

// enable texturing
$glEnable\,(GL\_TEXTURE\_2D)\,;$
$glBegin\,(GL\_QUADS)\,;$
$glTexCoord2d\,(0.0, 0.0)\,;\; glVertex2d\,(0.0, 0.0)\,;$
$glTexCoord2d\,(1.0, 0.0)\,;\; glVertex2d\,(1.0, 0.0)\,;$
$glTexCoord2d\,(1.0, 1.0)\,;\; glVertex2d\,(1.0, 1.0)\,;$
$glTexCoord2d\,(0.0, 1.0)\,;\; glVertex2d\,(0.0, 1.0)\,;$
$glEnd\,()\,;$

References

- OpenGL Tutorials - $http : //www.swiftless.com/tutorials/opengl/opengltuts.html$
- Texture Mapping - $http : //www.opengl.org/wiki/Texture\_Mapping$
- OpenGL Lighting - $http : //www.falloutsoftware.com/tutorials/gl/gl8.htm$
- OpenGL Texture Mapping - $http : //www.gamedev.net/reference/articles/article947.asp$
- OpenGL Texture Tutorial - $http : //www.nullterminator.net/gltexture.html$



Microsoft

OVER 10,000 ENTRIES

Microsoft Computer Dictionary

Fifth Edition

- Fully updated with the latest technologies, terms, and acronyms
- Easy to read, expertly illustrated
- Definitive coverage of hardware, software, the Internet, and more!

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Cataloging-in-Publication Data
Microsoft Computer Dictionary.--5th ed.
p. cm.
ISBN 0-7356-1495-4
1. Computers--Dictionaries. 2. Microcomputers--Dictionaries.

AQ76.5. M52267 2002
004'.03--dc21 200219714

Printed and bound in the United States of America.

2 3 4 5 6 7 8 9 QWT 7 6 5 4 3 2

Distributed in Canada by H.B. Fenn and Company Ltd.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to *mspinput@microsoft.com.*

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Acquisitions Editor: Alex Blanton
Project Editor: Sandra Haynes

Body Part No. X08-41929

application processor *n.* A processor dedicated to a single application.

application program *n. See* application.

application program interface. *n. See* application programming interface.

application programming interface *n.* A set of routines used by an application program to direct the performance of procedures by the computer's operating system. *Acronym:* API.*Also called:* application program interface.

application server *n.* **1.** A server program on a computer in a distributed network that handles the business logic between users and backend business applications or databases. Application servers also can provide transaction management, failover, and load balancing. An application server is often viewed as part of a three-tier application consisting of a front-end GUI server such as an HTTP server (first tier), an application server (middle tier), and a backend database and transaction server (third tier). *Also called:* appserver. *Compare* HTTP server (definition 1). **2.** Any machine on which an application-server program is running. *Also called:* appserver.

application service provider *n.* A third-party company or organization that hosts applications or services for individuals or business customers. The customer connects to a data center maintained by the application service provider (ASP) through Internet or private lines to access applications that would otherwise need to be housed on the customer's local servers or individual PCs. This arrangement allows the customer to free up disk space that would otherwise be taken by applications, as well as to access the most recent software updates. ASPs deliver solutions ranging from high-end applications to services for small and medium-sized businesses. *Acronym:* ASP.

application shortcut key *n.* A key or combination of keys that when pressed will quickly perform an action within an application that would normally require several user actions, such as menu selections. *Also called:* keyboard shortcut.

application software *n. See* application.

application-specific integrated circuit *n. See* gate array.

application suite *n. See* suite (definition 1).

appserver *n. See* application server.

Aqua *n.* The graphical user interface (GUI) of Macintosh OS X. Aqua was designed to maintain familiarity and a comfort level for users of the earlier Macintosh system while allowing access to newer Macintosh OS X capabilities. The Aqua GUI features updated versions of Macintosh staples such as the Finder alongside new features like the Dock, a new type of organizational tool. *See also* Dock, Macintosh OS X.

arbitration *n.* A set of rules for resolving competing demands for a machine resource by multiple users or processes. *See also* contention.

.arc *n.* The file extension that identifies compressed archive files encoded using the Advanced RISC Computing Specification (ARC) format. *See also* compressed file.

arcade game *n.* **1.** A coin-operated computer game for one or more players that features high-quality screen graphics, sound, and rapid action. **2.** Any computer game designed to mimic the style of a coin-operated arcade game, such as games marketed for the home computer. *See also* computer game.

Archie *n.* An Internet utility for finding files in public archives obtainable by anonymous FTP. The master Archie server at McGill University in Montreal downloads FTP indexes from participating FTP servers, merges them into a master list, and sends updated copies of the master list to other Archie servers each day. Archie is a shortened form of *archive. See also* anonymous FTP, FTP^1 (definition 1). *Compare* Jughead, Veronica.

Archie client *n. See* Archie.

Archie server *n.* On the Internet, a server that contains Archie indexes to the names and addresses of files in public FTP archives. *See also* Archie, FTP^1 (definition 1), server (definition 2).

architecture *n.* **1.** The physical construction or design of a computer system and its components. *See also* cache, CISC, closed architecture, network architecture, open architecture, pipelining, RISC. **2.** The data-handling capacity of a microprocessor. **3.** The design of application software incorporating protocols and the means for expansion and interfacing with other programs.

archive^1 *n.* **1.** A tape or disk containing files copied from another storage device and used as backup storage. **2.** A compressed file. **3.** A file directory on the Internet that is

CD. A permutation is a grouping of elements taken from a larger set with regard to the order of the elements. For example, in making permutations of two objects from the same set of four objects, there would be four candidates to choose from for the first selection (A), and three left over to choose from for the second selection (B), or 12 permutations in all: AB, AC, AD, BA, BC, BD, CA, CB, CD, DA, DB, DC. *See also* combinatorial explosion.

COM callable wrapper *n.* A proxy object generated by the runtime so that existing COM applications can use managed classes, including .NET Framework classes, transparently. *Acronym:* CCW.

COMDEX *n.* Any of a series of annual computer trade shows operated by Softbank COMDEX, Inc. One of these shows takes place in Las Vegas each November and is the largest computer trade show in the United States.

Comité Consultatif International Télégraphique et Téléphonique *n. See* CCITT.

comma-delimited file *n.* A data file consisting of fields and records, stored as text, in which the fields are separated from each other by commas. Use of comma-delimited files allows communication between database systems that use different formats. If the data in a field contains a comma, the field is further surrounded with quotation marks.

command *n.* An instruction to a computer program that, when issued by the user, causes an action to be carried out. Commands are usually either typed at the keyboard or chosen from a menu.

command buffer *n.* An area in memory in which commands entered by the user are kept. A command buffer can enable the user to repeat commands without retyping them completely, edit past commands to change some argument or correct a mistake, undo commands, or obtain a list of past commands. *See also* history, template (definition 4).

command button *n.* A control shaped like a pushbutton in a dialog box in a graphical user interface. By clicking a command button, the user causes the computer to perform some action, such as opening a file that has just been selected using the other controls in the dialog box.

COMMAND.COM *n.* The command interpreter for MS-DOS. *See also* command interpreter.

command-driven *adj.* Accepting commands in the form of code words or letters, which the user must learn. *Compare* menu-driven.

command-driven system *n.* A system in which the user initiates operations by a command entered from the console. *Compare* graphical user interface.

command interpreter *n.* A program, usually part of the operating system, that accepts typed commands from the keyboard and performs tasks as directed. The command interpreter is responsible for loading applications and directing the flow of information between applications. In OS/2 and MS-DOS, the command interpreter also handles simple functions, such as moving and copying files and displaying disk directory information. *See also* shell^1.

Command key *n.* On the original Macintosh keyboard, a key labeled with the special symbol, sometimes called the propeller or puppy foot. This key is found on one or both sides of the Spacebar, depending on the version of the Apple keyboard. The key serves some of the same functions as the Control key on IBM keyboards. *See also* Control key.

command language *n.* The set of keywords and expressions that are accepted as valid by the command interpreter. *See also* command interpreter.

command line *n.* A string of text written in the command language and passed to the command interpreter for execution. *See also* command language.

command-line interface *n.* A form of interface between the operating system and the user in which the user types commands, using a special command language. Although systems with command-line interfaces are usually considered more difficult to learn and use than those with graphical interfaces, command-based systems are usually programmable; this gives them flexibility unavailable in graphics-based systems that do not have a programming interface. *Compare* graphical user interface.

command mode *n.* A mode of operation in which a program waits for a command to be issued. *Compare* edit mode, insert mode.

command processing *n. See* command-driven system.

command processor *n. See* command interpreter.

command prompt window *n.* A window displayed on the desktop used to interface with the MS-DOS operating

insider attack *n.* An attack on a network or system carried out by an individual associated with the hacked system. Insider attacks are typically the work of current or former employees of a company or organization who have knowledge of passwords and network vulnerabilities. *Compare* intruder attack.

Ins key *n. See* Insert key.

install *vb.* To set in place and prepare for operation. Operating systems and application programs commonly include a disk-based installation, or setup, program that does most of the work of preparing the program to work with the computer, printer, and other devices. Often such a program can check for devices attached to the system, request the user to choose from sets of options, create a place for the program on the hard disk, and modify system startup files as necessary.

installable device driver *n.* A device driver that can be embedded within an operating system, usually in order to override an existing, less-functional service.

Installable File System Manager *n.* In Windows 9x and Windows 2000, the part of the file system architecture responsible for arbitrating access to the different file system components. *Acronym:* IFS.

installation program *n.* A program whose function is to install another program, either on a storage medium or in memory. An installation program, also called a setup program, might be used to guide a user through the often complex process of setting up an application for a particular combination of machine, printer, and monitor.

Installer *n.* A program, provided with the Apple Macintosh operating system, that allows the user to install system upgrades and make bootable (system) disks.

instance *n.* An object, in object-oriented programming, in relation to the class to which it belongs. For example, an object *myList* that belongs to a class *List* is an instance of the class *List*. *See also* class, instance variable, instantiate, object (definition 2).

instance variable *n.* A variable associated with an instance of a class (an object). If a class defines a certain variable, each instance of the class has its own copy of that variable. *See also* class, instance, object (definition 2), object-oriented programming.

instantiate *vb.* To create an instance of a class. *See also* class, instance, object (definition 2).

instant messaging *n.* A service that alerts users when friends or colleagues are on line and allows them to communicate with each other in real time through private online chat areas. With instant messaging, a user creates a list of other users with whom he or she wishes to communicate; when a user from his or her list is on line, the service alerts the user and enables immediate contact with the other user. While instant messaging has primarily been a proprietary service offered by Internet service providers such as AOL and MSN, businesses are starting to employ instant messaging to increase employee efficiency and make expertise more readily available to employees.

Institute of Electrical and Electronics Engineers *n. See* IEEE.

instruction *n.* An action statement in any computer language, most often in machine or assembly language. Most programs consist of two types of statements: declarations and instructions. *See also* declaration, statement.

instruction code *n. See* operation code.

instruction counter *n. See* instruction register.

instruction cycle *n.* The cycle in which a processor retrieves an instruction from memory, decodes it, and carries it out. The time required for an instruction cycle is the sum of the instruction (fetch) time and the execution (translate and execute) time and is measured by the number of clock ticks (pulses of a processor's internal timer) consumed.

instruction mix *n.* The assortment of types of instructions contained in a program, such as assignment instructions, mathematical instructions (floating-point or integer), control instructions, and indexing instructions. Knowledge of instruction mixes is important to designers of CPUs because it tells them which instructions should be shortened to yield the greatest speed, and to designers of benchmarks because it enables them to make the benchmarks relevant to real tasks.

instruction pointer *n. See* program counter.

instruction register *n.* A register in a central processing unit that holds the address of the next instruction to be executed.

instruction set *n.* The set of machine instructions that a processor recognizes and can execute. *See also* assembler, microcode.

frequently requested data) on computers running at bus speeds of 75 MHz or higher. *Acronym:* PB SRAM. *See also* burst (definition 2), L2 cache, pipelining, static RAM. *Compare* asynchronous static RAM, dynamic RAM, synchronous burst static RAM.

pipeline processing *n.* A method of processing on a computer that allows fast parallel processing of data. This is accomplished by overlapping operations using a *pipe*, or a portion of memory that passes information from one process to another. *See also* parallel processing, pipe (definition 1), pipelining (definition 3).

pipelining *n.* **1.** A method of fetching and decoding instructions (preprocessing) in which, at any given time, several program instructions are in various stages of being fetched or decoded. Ideally, pipelining speeds execution time by ensuring that the microprocessor does not have to wait for instructions; when it completes execution of one instruction, the next is ready and waiting. *See also* superpipelining. **2.** In parallel processing, a method in which instructions are passed from one processing unit to another, as on an assembly line, and each unit is specialized for performing a particular type of operation. **3.** The use of pipes in passing the output of one task as input to another until a desired sequence of tasks has been carried out. *See also* pipe (definition 1), pour.

piracy *n.* **1.** The theft of a computer design or program. **2.** Unauthorized distribution and use of a computer program.

.pit *n.* A file extension for an archive file compressed with PackIT. *See also* PackIT.

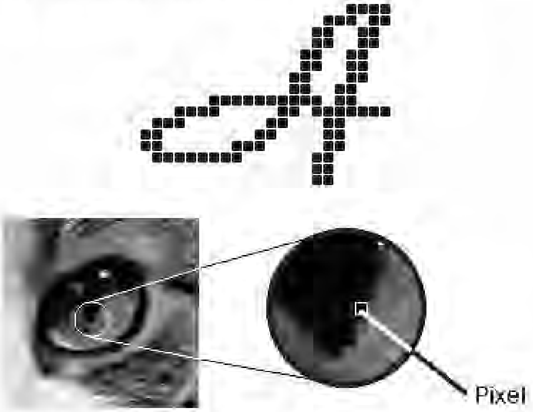
pitch *n.* A measure, generally used with monospace fonts, that describes the number of characters that fit in a horizontal inch. *See also* characters per inch, screen pitch. *Compare* point^{1} (definition 1).

PivotChart *adj.* A graphical tool in Microsoft Excel or Access that can be used to display data from a list or database in chart form. Based on user-selected information incorporated in an Excel PivotTable report or list, a PivotChart report provides the ability to chart the data interactively—for example, to "pivot" the chart's point of view from product sales by category to product sales by region or by salesperson. *See also* PivotTable.

PivotTable *adj.* An interactive table in Microsoft Excel or Access that can show the same data from a list or a database in more than one arrangement. A user can manipulate the rows and columns in a PivotTable to view or summarize the information in different ways for purposes of analysis. In Excel, a PivotTable report is the basis for creating a PivotChart report that displays the same data in chart form. *See also* PivotChart.

pivot year *n.* In Year 2000 windowing, a date in a 100-year period that serves as the point from which correct dates can be calculated in systems or software that can store only 2-digit years. For example, a pivot year of 1970 means that the numbers 70 through 99 are interpreted as the years 1970 to 1999, and the numbers 00 through 69 as the years 2000 through 2069. *See also* windowing.

pixel *n.* Short for picture (**pix**) element. One spot in a rectilinear grid of thousands of such spots that are individually "painted" to form an image produced on the screen by a computer or on paper by a printer. A pixel is the smallest element that display or print hardware and software can manipulate in creating letters, numbers, or graphics. See the illustration. *Also called:* pel.



Pixel. *The letter A is actually made up of a pattern of pixels in a grid, as is the cat's eye.*

pixel image *n.* The representation of a color graphic in a computer's memory. A pixel image is similar to a bit image, which also describes a screen graphic, but a pixel image has an added dimension, sometimes called depth, that describes the number of bits in memory assigned to each on-screen pixel.

point of sale *n. See* POS.

point-to-point configuration *n.* A communications link in which dedicated links exist between individual origins and destinations, as opposed to a point-to-multipoint configuration, in which the same signal goes to many destinations (such as a cable TV system), or a switched configuration, in which the signal moves from the origin to a switch that routes the signal to one of several possible destinations. *Also called:* point-to-point connection.

point-to-point connection *n. See* point-to-point configuration.

point-to-point message system *n.* In Sun Microsystems's J2EE network platform, a messaging system that uses message queues to store asynchronous, formatted data for coordinating enterprise applications. Each message is addressed to a specific queue, and client applications retrieve messages from the queues. *See also* asynchronous, J2EE.

Point-to-Point Protocol *n. See* PPP.

point-to-point tunneling *n.* A means of setting up secure communications over an open, public network such as the Internet. *See also* PPTP.

Point-to-Point Tunneling Protocol *n. See* PPTP.

Poisson distribution *n.* A mathematical curve often used in statistics and simulation to represent the likelihood of some event occurring, such as the arrival of a customer in a queue, when the average likelihood is known. This distribution, named after the French mathematician S. D. Poisson, is simpler to calculate than the normal and binomial distributions. *See also* binomial distribution, normal distribution.

poke *vb.* To store a byte into an absolute memory location. PEEK (read a byte from memory) and POKE commands are often found in programming languages, such as Basic, that do not normally allow access to specific memory locations.

polar coordinates *n.* Coordinates of the form (*r*, q) used to locate a point in two dimensions (on a plane). The polar coordinate *r* is the length of the line that starts at the origin and ends at the point, and q (Greek theta) is the angle between that line and the positive *x*-axis. *Compare* Cartesian coordinates.

polarity *n.* The sign of the potential (voltage) difference between two points in a circuit. When a potential difference exists between two points, one point has a positive polarity and the other a negative polarity. Electrons flow from negative to positive; by convention, however, current is considered to flow from positive to negative.

polarized component *n.* A circuit component that must be installed with its leads in a particular orientation with respect to the polarity of the circuit. Diodes, rectifiers, and some capacitors are examples of polarized components.

polarizing filter *n.* A transparent piece of glass or plastic that polarizes the light passing through it; that is, it allows only waves vibrating in a certain direction to pass through. Polarizing filters are often used to reduce glare on monitor screens. *See also* glare filter.

Polish notation *n. See* prefix notation.

polling *n. See* autopolling.

polling cycle *n.* The time and sequence required for a program to poll each of its devices or network nodes. *See also* autopolling.

polygon *n.* Any two-dimensional closed shape composed of three or more line segments, such as a hexagon, an octagon, or a triangle. Computer users encounter polygons in graphics programs.

polyline *n.* An open shape consisting of multiple connected segments. Polylines are used in CAD and other graphics programs. *See also* CAD.

polymorphism *n.* In an object-oriented programming language, the ability to redefine a routine in a derived class (a class that inherited its data structures and routines from another class). Polymorphism allows the programmer to define a base class that includes routines that perform standard operations on groups of related objects, without regard to the exact type of each object. The programmer then redefines the routines in the derived class for each type, taking into account the characteristics of the object. *See also* class, derived class, object (definition 2), object-oriented programming.

Pong *n.* The first commercial video game, a table tennis simulation, created by Nolan Bushnell of Atari in 1972.

preventive maintenance *n.* Routine servicing of hardware intended to keep equipment in good operating condition and to find and correct problems before they develop into severe malfunctions.

preview *n.* In word processors and other applications, the feature that formats a document for printing but displays it on the video monitor rather than sending it directly to the printer.

PRI *n.* Acronym for Primary Rate Interface. One of two ISDN transmission rate services (the other is the basic rate interface, BRI). PRI has two variations. The first, which operates at 1.536 Mbps, transmits data over 23 B channels and sends signaling information at 64 Kbps over one D channel in the United States, Canada, and Japan. The second, which operates at 1.984 Mbps, transmits data over 30 B channels and sends signaling information at 64 Kbps over one D channel in Europe and Australia. *See also* BRI, ISDN.

primary channel *n.* The data-transmission channel in a communications device, such as a modem. *Compare* secondary channel.

Primary Domain Controller *n.* **1.** In Windows NT, a database providing a centralized administration site for resources and user accounts. The database allows users to log onto the domain, rather than onto a specific host machine. A separate account database keeps track of the machines in the domain and allocates the domain's resources to users. **2.** In any local area network, the server that maintains the master copy of the domain's user accounts database and that validates logon requests. *Acronym:* PDC.

primary key *n.* In databases, the key field that serves as the unique identifier of a specific tuple (row) in a relation (database table). *Also called:* major key. *See also* alternate key (definition 1), candidate key. *Compare* secondary key.

Primary Rate Interface *n.* *See* PRI.

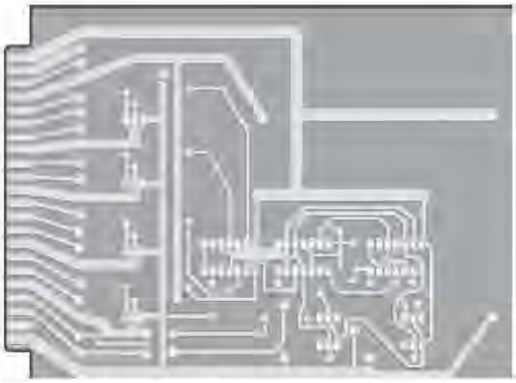
primary storage *n.* Random access memory (RAM); the main general-purpose storage region to which the microprocessor has direct access. A computer's other storage options, such as disks and tape, are called *secondary storage* or (sometimes) *backing storage.*

primitive *n.* **1.** In computer graphics, a shape, such as a line, circle, curve, or polygon, that can be drawn, stored, and manipulated as a discrete entity by a graphics program. A primitive is one of the elements from which a large graphic design is created. **2.** In programming, a fundamental element in a language that can be used to create larger procedures that do the work a programmer wants to do.

print *vb.* In computing, to send information to a printer. The word is also sometimes used in the sense of "show me" or "copy this." For example, the PRINT statement in Basic causes output to be displayed (printed) on the screen. Similarly, an application program that can be told to print a file to disk interprets the command as an instruction to route output to a disk file instead of to a printer.

print buffer *n.* A section of memory to which print output can be sent for temporary storage until the printer is ready to handle it. A print buffer can exist in a computer's random access memory (RAM), in the printer, in a separate unit between the computer and the printer, or on disk. Regardless of its location, the function of a print buffer is to free the computer for other tasks by taking print output at high speed from the computer and passing it along at the much slower rate required by the printer. Print buffers vary in sophistication: some simply hold the next few characters to be printed, and others can queue, reprint, or delete documents sent for printing.

printed circuit board *n.* A flat board made of nonconducting material, such as plastic or fiberglass, on which chips and other electronic components are mounted, usually in predrilled holes designed to hold them. The component holes are connected electrically by predefined conductive metal pathways that are printed on the surface of the board. The metal leads protruding from the electronic components are soldered to the conductive metal pathways to form a connection. A printed circuit board should be held by the edges and protected from dirt and static electricity to avoid damage. See the illustration. *Acronym:* PCB.



Printed circuit board.

privileged mode *n.* A mode of execution, supported by the protected mode of the Intel 80286 and higher microprocessors, in which software can carry out restricted operations that manipulate critical components of the system, such as memory and input/output ports (channels). Application programs cannot be executed in privileged mode; the heart (kernel) of the OS/2 operating system can be, as can the programs (device drivers) that control devices attached to the system.

privileges *n. See* access privileges.

PRN *n.* The logical device name for *printer.* A name reserved by the MS-DOS operating system for the standard print device. PRN usually refers to a system's first parallel port, also known as LPT1.

.pro *n.* One of seven new top-level domain names approved in 2000 by the Internet Corporation for Assigned Names and Numbers (ICANN), .pro is meant for use in Web sites relating to professions such as physicians, accountants, and lawyers. Six of the new domains became available for use in the spring of 2001; negotiations are still underway for the final registry agreement for the .pro domain.

probability *n.* The likelihood that an event will happen, which can often be estimated mathematically. In mathematics, statistics and probability theory are related fields. In computing, probability is used to determine the likelihood of failure or error in a system or device.

problem solving *n.* **1.** The process of devising and implementing a strategy for finding a solution or for transforming a less desirable condition into a more desirable one. **2.** An aspect of artificial intelligence wherein the task of problem solving is performed solely by a program. *See also* artificial intelligence.

procedural language *n.* A programming language in which the basic programming element is the procedure (a named sequence of statements, such as a routine, subroutine, or function). The most widely used high-level languages (C, Pascal, Basic, FORTRAN, COBOL, Ada) are all procedural languages. *See also* procedure. *Compare* nonprocedural language.

procedural rendering *n.* The rendering of a two-dimensional image from three-dimensional coordinates with texturing according to user-specified conditions, such as direction and degree of lighting.

procedure *n.* In a program, a named sequence of statements, often with associated constants, data types, and variables, that usually performs a single task. A procedure can usually be called (executed) by other procedures, as well as by the main body of the program. Some languages distinguish between a procedure and a function, with the latter (the function) returning a value. *See also* function, parameter, procedural language, routine, subroutine.

procedure call *n.* In programming, an instruction that causes a procedure to be executed. A procedure call can be located in another procedure or in the main body of the program. *See also* procedure.

process^1 *n.* A program or part of a program; a coherent sequence of steps undertaken by a program.

process^2 *vb.* To manipulate data with a program.

process-bound *adj.* Limited in performance by processing requirements. *See also* computation-bound.

process color *n.* A method of handling color in a document in which each block of color is separated into its subtractive primary color components for printing: cyan, magenta, and yellow (as well as black). All other colors are created by blending layers of various sizes of halftone spots printed in cyan, magenta, and yellow to create the image. *See also* color model, color separation (definition 1). *Compare* spot color.

processing *n.* The manipulation of data within a computer system. Processing is the vital step between receiving data (input) and producing results (output)—the task for which computers are designed.

processor *n. See* central processing unit, microprocessor.

Processor Direct Slot *n. See* PDS (definition 1).

Processor Input/Output *n. See* PIO.

Procmail *n.* An open-source e-mail-processing utility for Linux and other UNIX-based computers and networks. Procmail can be used to create mail servers and mailing lists, filter mail, sort incoming mail, preprocess mail, and perform other mail-related functions.

Prodigy *n.* An Internet service provider (ISP) that offers Internet access and a wide range of related services. Prodigy was founded by IBM and Sears as a proprietary online service, was acquired by International Wireless in 1996, and in 1999 entered into a partnership with SBC Communications. The addition of SBC's Internet customer base made Prodigy the third largest ISP in the United States.

Prodigy Information Service *n.* An online information service founded by IBM and Sears. Like its competitors America Online and CompuServe, Prodigy offers access to databases and file libraries, online chat, special interest groups, e-mail, and Internet connectivity. *Also called:* Prodigy.

product *n.* **1.** An operator in the relational algebra used in database management that, when applied to two existing relations (tables), results in the creation of a new table containing all possible ordered concatenations (combinations) of tuples (rows) from the first relation with tuples from the second. The number of rows in the resulting relation is the product of the number of rows in the two source relations. *Also called:* Cartesian product. *Compare* inner join. **2.** In mathematics, the result of multiplying two or more numbers. **3.** In the most general sense, an entity conceived and developed for the purpose of competing in a commercial market. Although computers are products, the term is more commonly applied to software, peripherals, and accessories in the computing arena.

production system *n.* In expert systems, an approach to problem solving based on an "IF this, THEN that" approach that uses a set of rules, a database of information, and a "rule interpreter" to match premises with facts and form a conclusion. Production systems are also known as rule-based systems or inference systems. *See also* expert system.

Professional Graphics Adapter *n.* A video adapter introduced by IBM, primarily for CAD applications. The Professional Graphics Adapter is capable of displaying 256 colors, with a horizontal resolution of 640 pixels and a vertical resolution of 480 pixels. *Acronym:* PGA.

Professional Graphics Display *n.* An analog display introduced by IBM, intended for use with their Professional Graphics Adapter. *See also* Professional Graphics Adapter.

profile^1 *n. See* user profile.

profile^2 *vb.* To analyze a program to determine how much time is spent in different parts of the program during execution.

profiler *n.* A diagnostic tool for analyzing the run-time behavior of programs.

Profiles for Open Systems Internetworking Technology *n. See* POSIT.

program^1 *n.* A sequence of instructions that can be executed by a computer. The term can refer to the original source code or to the executable (machine language) version. *Also called:* software. *See also* program creation, routine, statement.

program^2 *vb.* To create a computer program, a set of instructions that a computer or other device executes to perform a series of actions or a particular type of work.

program button *n.* On a handheld device, a navigation control that is pressed to launch an application. *Also called:* application button.

program card *n. See* PC Card, ROM card.

program cartridge *n. See* ROM cartridge.

program comprehension tool *n.* A software engineering tool that facilitates the process of understanding the structure and/or functionality of computer applications. *Acronym:* PCT. *Also called:* software exploration tool.

program counter *n.* A register (small, high-speed memory circuit within a microprocessor) that contains the address (location) of the instruction to be executed next in the program sequence.

program creation *n.* The process of producing an executable file. Traditionally, program creation comprises three steps: (1) compiling the high-level source code into assembly language source code; (2) assembling the assembly language source code into machine-code object files; and (3) linking the machine-code object files with various data files, run-time files, and library files into an executable file. Some compilers go directly from high-level source to machine-code object, and some integrated development environments compress all three steps into a single command. *See also* assembler, compiler (definition 2), linker, program.

program encapsulation *n.* A method of dealing with programs with Year 2000 problems that entailed modifying the data with which a program worked. The input data is modified to reflect a parallel date in the past that the program can handle. When output is generated, that data is changed again, to reflect the correct date. The program itself remains unchanged.

program file *n.* A disk file that contains the executable portions of a computer program. Depending on its size and net, remote login is done primarily by rlogin and telnet. *See also* rlogin[1] (definition 1), telnet[1].

remote monitoring *n. See* RMON.

remote network monitoring *n. See* RMON.

Remote PC *n. See* remote system.

remote procedure call *n.* In programming, a call by one program to a second program on a remote system. The second program generally performs a task and returns the results of that task to the first program. *Acronym:* RPC.

remote system *n.* The computer or network that a remote user is accessing via a modem. *See also* remote access. *Compare* remote terminal.

remote terminal *n.* A terminal that is located at a site removed from the computer to which it is attached. Remote terminals rely on modems and telephone lines to communicate with the host computer. *See also* remote access. *Compare* remote system.

removable disk *n.* A disk that can be removed from a disk drive. Floppy disks are removable; hard disks usually are not. *Also called:* exchangeable disk.

REM statement *n.* Short for **remark statement**. A statement in the Basic programming language and the MS-DOS and OS/2 batch file languages that is used to add comments to a program or batch file. Any statement beginning with the word *REM* is ignored by the interpreter or compiler or the command processor. *See also* comment.

rename *n.* A command in most file transfer protocol (FTP) clients and in many other systems that allows the user to assign a new name to a file or files.

render *vb.* To produce a graphic image from a data file on an output device such as a video display or printer.

rendering *n.* The creation of an image containing geometric models, using color and shading to give the image a realistic look. Usually part of a geometric modeling package such as a CAD program, rendering uses mathematics to describe the location of a light source in relation to the object and to calculate the way in which the light would create highlights, shading, and variations in color. The degree of realism can range from opaque, shaded polygons to images approximating photographs in their complexity. *See also* ray tracing.

RenderMan Shading Language *n.* A C-like graphics and rendering language developed by Pixar.

repaginate *vb.* To recalculate the page breaks in a document.

Repeat *n.* A command in Microsoft Word that causes all information contained in either the last command dialog box or the last uninterrupted editing session to be repeated.

repeat counter *n.* A loop counter; typically, a register that holds a number representing how many times a repetitive process has been or is to be executed.

Repeat delay *n.* A delay for the amount of time that elapses before a character begins repeating when you hold down a key.

repeater *n.* A device used on communications circuits that decreases distortion by amplifying or regenerating a signal so that it can be transmitted onward in its original strength and form. On a network, a repeater connects two networks or two network segments at the physical layer of the ISO/OSI reference model and regenerates the signal.

repeating Ethernet *n. See* repeater.

repeat key *n.* On some keyboards, a key that must be held down at the same time as a character key to cause the character key's key code to be sent repeatedly. On most computer keyboards, however, a repeat key is not needed because a key automatically repeats if held down for longer than a brief delay. *Compare* typematic.

RepeatKeys *n.* A feature of Windows 9x and Windows NT that allows a user to adjust or disable the typematic keyboard feature so as to accommodate users with restricted mobility, who may activate typematic by accident because they have trouble lifting their fingers from the keys. *See also* typematic. *Compare* BounceKeys, FilterKeys, MouseKeys, ShowSounds, SoundSentry, StickyKeys, ToggleKeys.

repetitive strain injury *n.* An occupational disorder of the tendons, ligaments, and nerves caused by the cumulative effects of prolonged repetitious movements. Repetitive strain injuries are appearing with increasing frequency among office workers who spend long hours typing at computerized workstations that are not equipped with safeguards such as wrist supports. *Acronym:* RSI. *See also* carpal tunnel syndrome, ergonomic keyboard, wrist support.

replace *vb.* To put new data in the place of other data, usually after conducting a search for the data to be replaced. Text-based applications such as word processors typically include search-and-replace commands. In such

Xerox PARC *n.* Short for **X**erox **P**alo **A**lto **R**esearch **C**enter. Xerox's research and development facility in Palo Alto, California. Xerox PARC is the birthplace of such innovations as the local area network (LAN), the laser printer, and the graphical user interface (GUI).

XFCN *n.* Short for e**x**ternal **f**un**c**tio**n**. An external code resource that returns a value after it has completed executing. XFCNs are used in HyperCard, a hypermedia program developed for the Macintosh. *See also* HyperCard, XCMD.

XFDL *n.* Short for Extensible Forms Description Language, a document description language introduced and submitted to the World Wide Web Committee in 1998 by the Canadian Internet forms company UWI.Com. XFDL is an XML-based language for describing complex forms, such as legal and government documents. It is designed to allow for interactivity yet remain consistent with Internet standards.

XGA *n. See* Extended Graphics Array.

x-height *n.* In typography, the height of the lowercase letter x in a particular font. The x-height thus represents the height of the body only of a lowercase letter, excluding ascenders (such as the top of the letter b) and descenders (such as the tail on the letter g). *See also* ascender, descender.

XHTML *n.* Short for Extensible Hypertext Markup Language. A markup language incorporating elements of HTML and XML. Web sites designed using XHTML can be more readily displayed on handheld computers and digital phones equipped with microbrowsers. XHTML was released for comments by the World Wide Web Consortium (W3C) in September 1999. *See also* HTML, microbrowser, XML.

XIP *n. See* execute in place.

XLANG *n.* A derivative XML language that describes the logical sequencing of business processes, as well as the implementation of the business process by using various application services.

XLink *n.* An XML language that provides a set of attributes that are used to create links between resources. XLink provides complex extended linking, link behavior, and management capabilities. XLink is able to describe links that connect sets of resources, point to multiple targets, or serve multiple roles within an XML document.

XLL *n.* Acronym for e**X**tensible **L**inking **L**anguage. Broad term intended to denote the family of XML linking/pointing/addressing languages, which include XLink, XPointer, and XPath.

XMI *n.* **1.** Acronym for **X**ML **M**etadata **I**nterchange Format. An object-based model for exchanging program data across the Internet. XMI is sponsored by IBM, Unisys, and others and was submitted as a proposed standard to the Object Management Group (OMG); it is now one of OMG's recommended technologies. XMI is designed to allow for storing and sharing programming information and exchanging data among tools, applications, and storage locations through a network or the Internet so that software developers can collaborate on applications, even if they are not all using the same development tools. **2.** As XMI *bus*, a 64-bit parallel bus supported on certain DEC and AlphaServer processors. An XMI bus is capable of transferring data, exclusive of addressing overhead, at 100 Mbps.

XML *n.* Acronym for e**X**tensible **M**arkup **L**anguage, a condensed form of SGML (Standard Generalized Markup Language). XML lets Web developers and designers create customized tags that offer greater flexibility in organizing and presenting information than is possible with the older HTML document coding system. XML is defined as a language standard published by the W3C and supported by the industry. *See also* SGML.

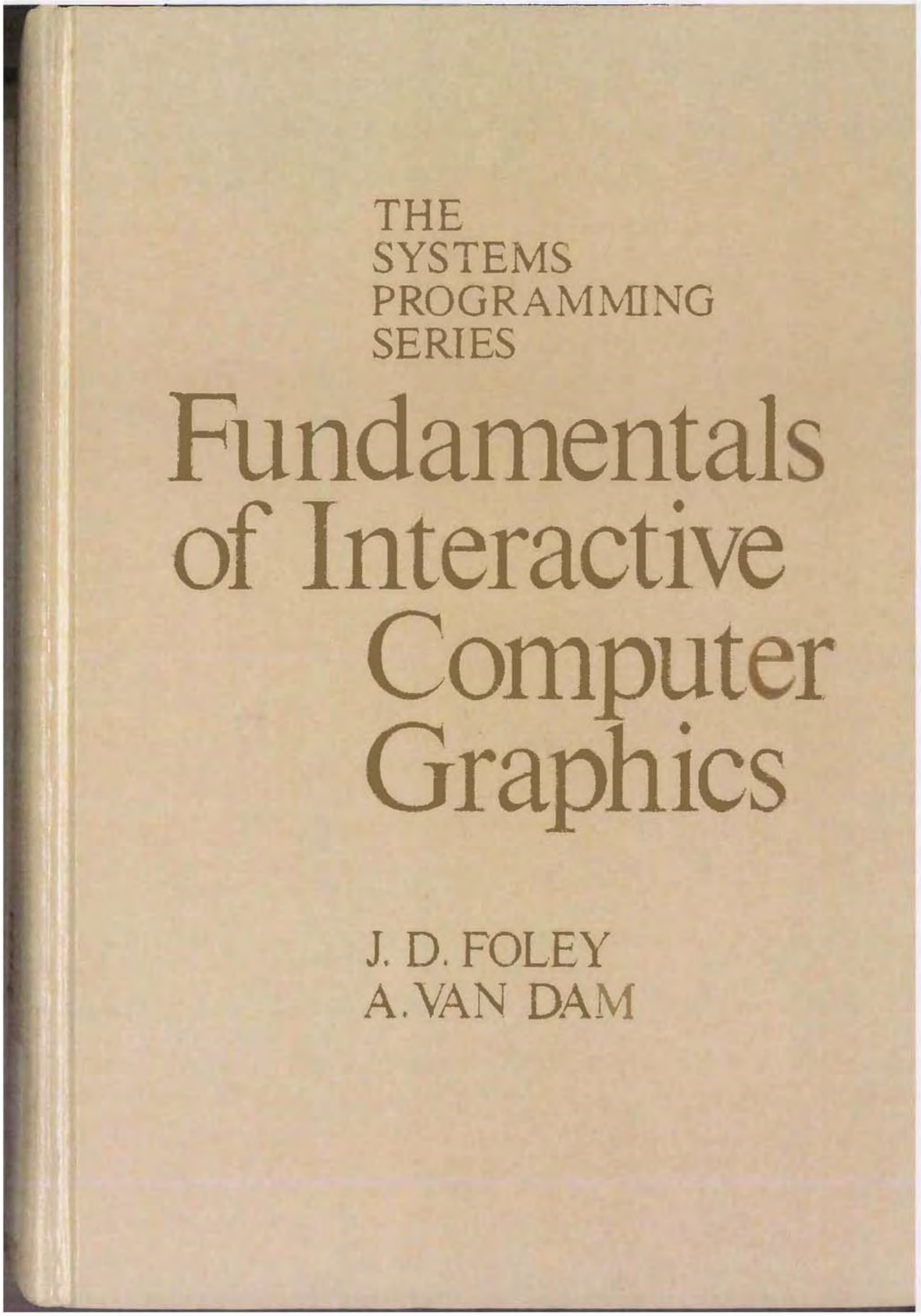
XML attribute *n.* Information added to a tag to provide more information about the tag, such as &lt;ingredient quantity="2"units="cups">flour&lt;/ingredient&gt;.

XML element *n.* Information delimited by a start tag and an end tag in an eXtensible Markup Language (XML) document. An example would be &lt;Lastname> Davalio</LastName>.

XML entities *n.* Combinations of characters and symbols that replace other characters when an XML document is parsed, usually those that have other meanings in XML. For example, &amp;lt; represents the &lt; symbol, which is also the opening bracket for a tag.

XML Metadata Interchange Format *n. See* XMI (definition 1).

XML-RPC *n.* Acronym for e**X**tensible **M**arkup **L**anguage-**R**emote **P**rocedure **C**all. A set of XML-based implementations that allows cross-platform and cross-programming language procedure calls over the Internet. XML-RPC



THE SYSTEMS PROGRAMMING SERIES

Fundamentals of Interactive Computer Graphics

J. D. FOLEY
A. VAN DAM

Fundamentals of Interactive Computer Graphics

JAMES D. FOLEY
The George Washington University
ANDRIES VAN DAM
Brown University

Addison-Wesley Publishing Company
Reading, Massachusetts • Menlo Park, California
London • Amsterdam • Don Mills, Ontario • Sydney

Sponsoring Editor: William B. Gruener
Production Editor: Rima Zolina
Designer: Herb Caswell
Illustration: ANCO/Boston
Cover Design: Richard Hannus
Art coordinator: Dick Morton

This book is in the
Addison-Wesley Systems Programming Series
Consulting editors: IBM Editorial Board

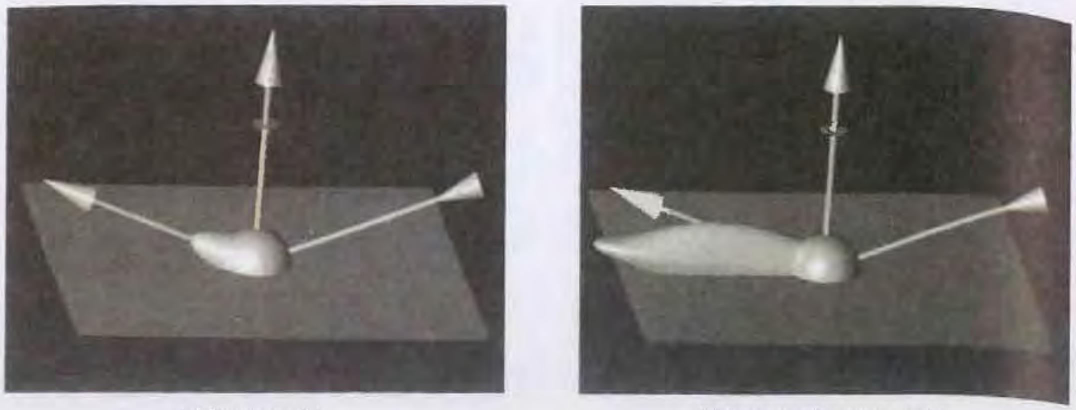
Library of Congress Cataloging in Publication Data

Foley, James D 1942–
Fundamentals of interactive computer graphics.

(The Systems programming series)
Bibliography: p.
Includes index.
1. Computer graphics. 2. Interactive computer
systems. I. Van Dam, Andries, 1938– joint author.
II. Title.
T385.F63 001.64'43 80-24311
ISBN 0-201-14468-9

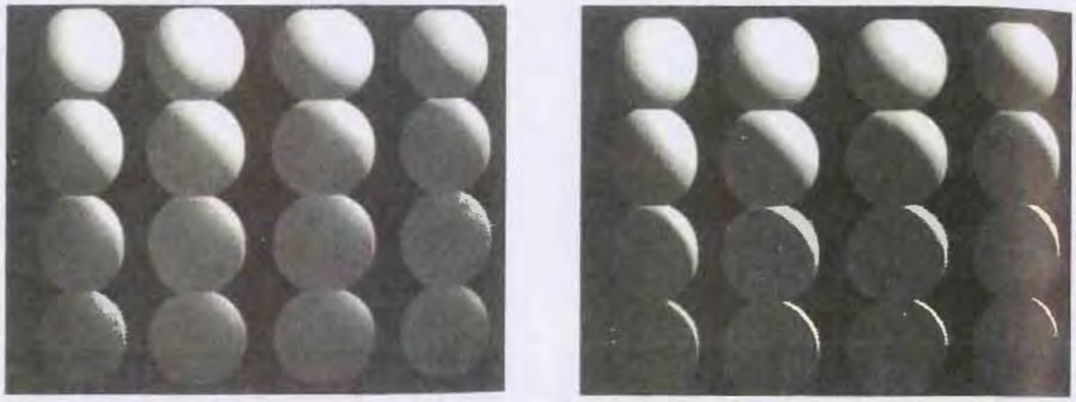
Reprinted with corrections, July 1984

ISBN: 0-201-14468-9
NOPQR-HA-8987



(a) Phong model (b) Torrance–Sparrow model

Fig. 16.5 Comparison of Phong and Torrance–Sparrow models for a 70° angle of incident light (by J. Blinn [BLIN77a], courtesy University of Utah).



(a) Phong model (b) Torrance-Sparrow model

Fig. 16.6 Comparison of the Phong and Torrance–Sparrow models for a metallic sphere illuminated by a light source from different directions. Differences are most apparent for back-lit cases (bottom rows) (by J. Blinn [BLIN77a], courtesy University of Utah).

16.4 POLYGON MESH SHADING

There are three basic ways to shade objects defined by polygon meshes. In order of increasing complexity, they are: constant shading, intensity interpolation shading, and normal-vector interpolation shading. In each case, any of the shading models from the previous two sections can be used. Recall that color shading just involves three equations rather than one.

Constant shading calculates a single intensity value for shading an entire polygon. Several assumptions are made:

1. The light source is at infinity, so $\overline{N} \cdot \overline{L}$ is constant across the polygonal face;
2. The viewer is at infinity, so $\overline{N} \cdot \overline{V}$ is constant across the polygon face;
3. The polygon represents the actual surface being modeled, and is not an approximation to a curved surface.

If either of the first two assumptions is unacceptable, then an average $\overline{L}$ and $\overline{V}$ might be used, perhaps calculated at the center of the polygon.

The final assumption is most often the one which is incorrect and has a much more substantial effect on the resulting image than the other two. The effect is that each visible polygonal facet of the approximated surface is distinguishable, because each is a slightly different intensity than its neighbors. The difference in shading on adjacent facets is accentuated by the Mach band effect, which was discovered in 1865 by E. Mach and is described in detail in [RATL65]. The effect is one of exaggeration of intensity change at any edge where there is a discontinuity in magnitude or slope of intensity. Figure 16.7 shows, for two separate cases, the actual and perceived changes in intensity along a surface. The effect is caused by *lateral inhibition* of the receptors in the eye, whose response to light is influenced by adjacent receptors in inverse relation to the distance to the adjacent receptor. Receptors immediately to the brighter side of an intensity change have more response than those further from the edge, because they receive less inhibition from their neighbors on the darker side. Similarly, receptors immediately to the darker side of an intensity change will have less response than those further into the darker area, because they receive more inhibition from their neighbors on the brighter side.

Figure 16.8(b) shows a car with constant shading. The Mach band effect is quite evident. Even though the polygonal patches are quite noticeable, the image is much more realistic than that in Fig. 16.8(a), which shows only the polygon edges.

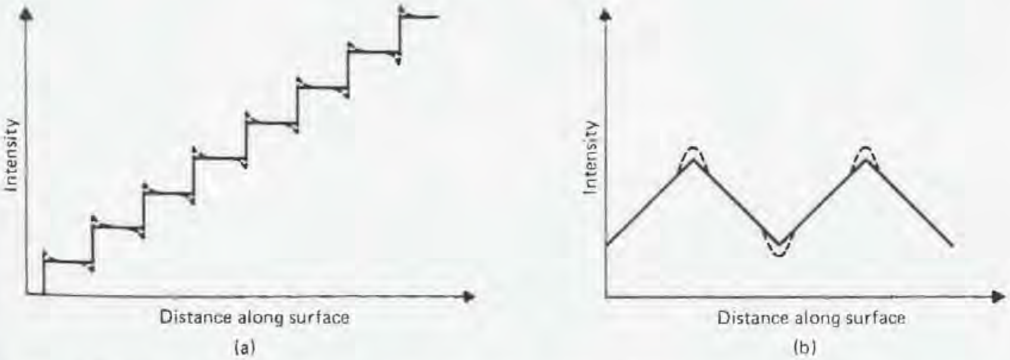
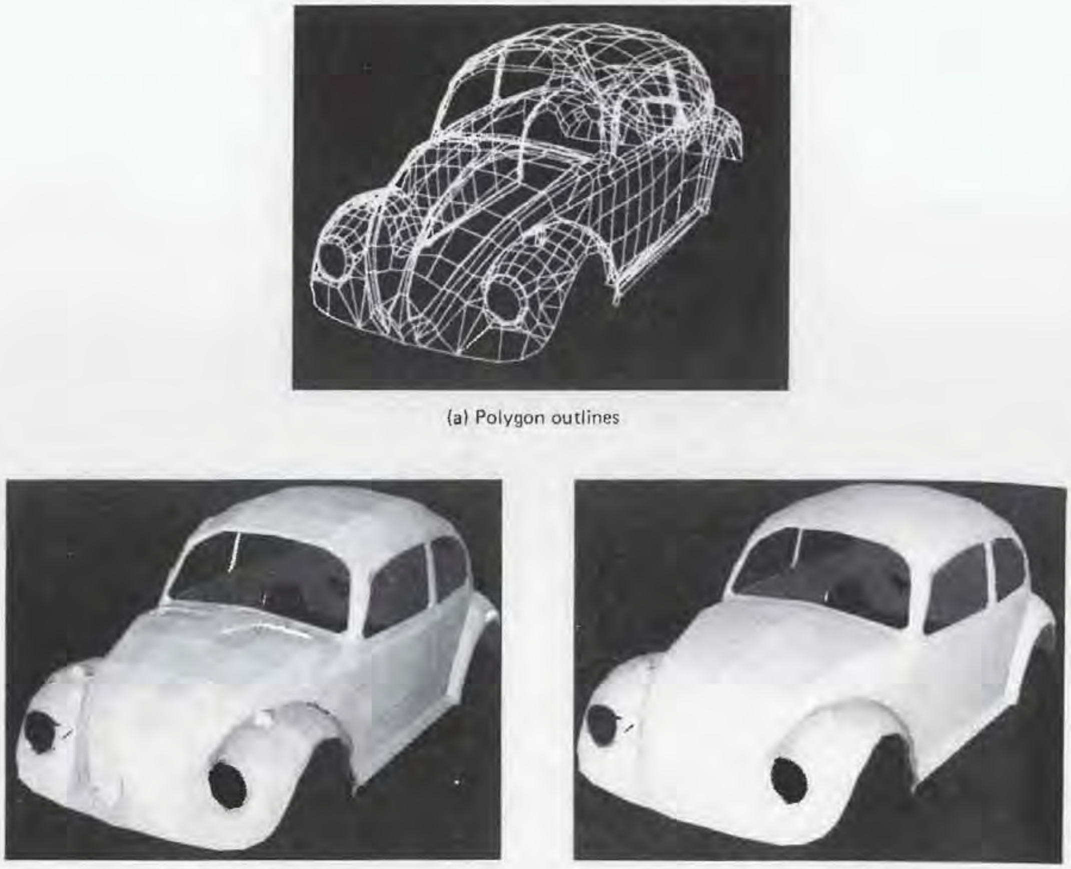


Fig. 16.7 Mach band effect—actual and perceived intensities: dashed lines—perceived intensity; solid lines—actual intensity.


(a) Polygon outlines

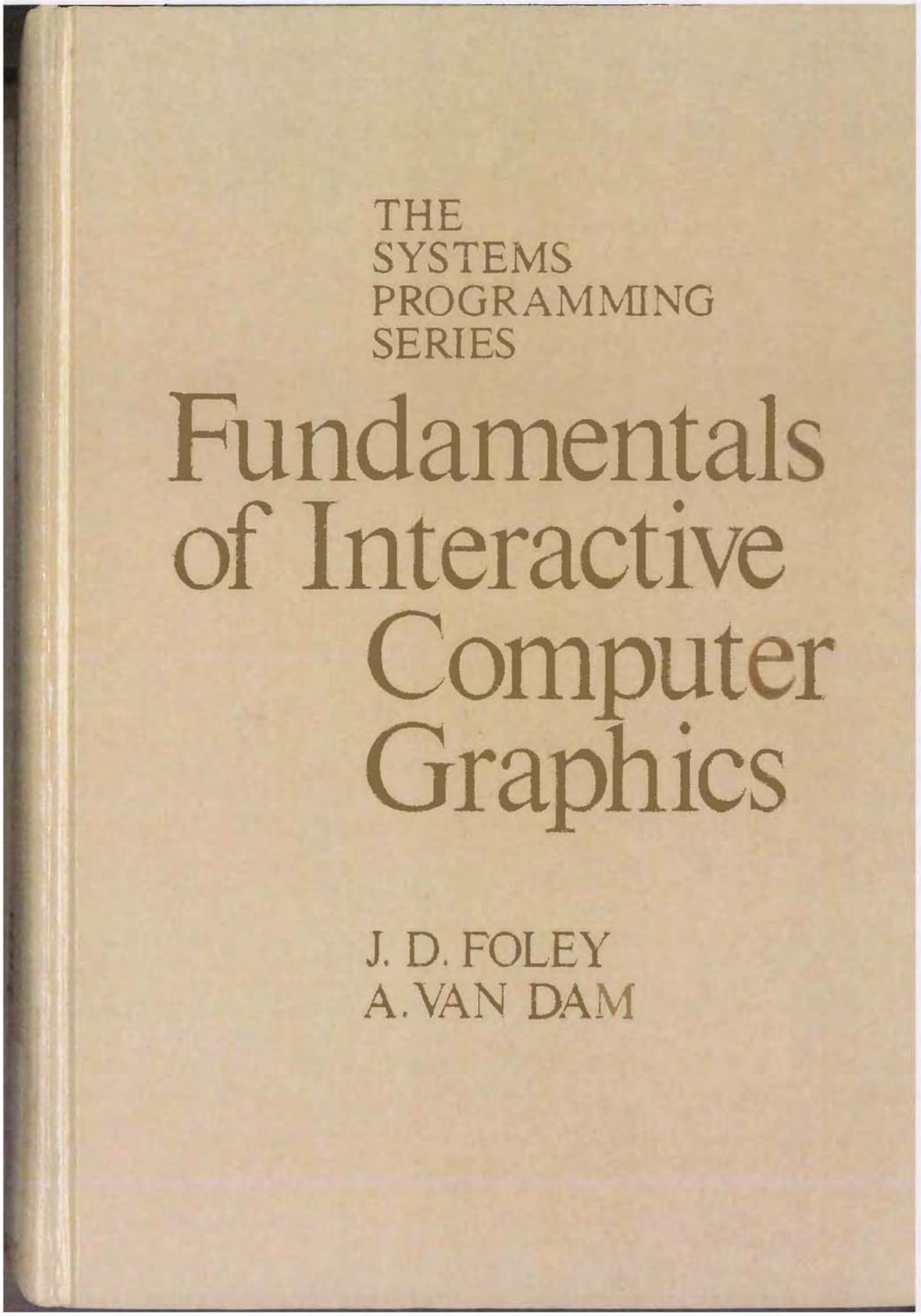
(b) Constant shading

(c) Gouraud shading

Fig. 16.8 Car body displayed three ways (courtesy University of Utah).

Intensity interpolation shading, usually known from the name of its developer as *Gouraud* shading [GOUR71], eliminates intensity discontinuities. Figure 16.8(c) shows a Gouraud-shaded car. The intensity ridge running down the hood on the right side of the picture, close to the fender, is a Mach band caused by a rapid change in the slope of the intensity curve: Gouraud shading does not completely eliminate such intensity changes.

The Gouraud shading process consists of four steps. First, surface normals are calculated. Second, *vertex normals* are calculated by averaging the surface normals of all polygonal facets that share the vertex (Fig. 16.9). If an edge is meant to be visible (such as at the joint between a planes' wing and body), then two vertex normals, one for each side of the edge, are found by separately averaging the normals of polygons on each side of the edge. Third, *vertex intensities* are found by using the vertex normals with any desired shading model. Finally, each polygon is shaded by linear interpolation of vertex intensities along each edge and then between edges along each scan line (Fig. 16.10).



THE
SYSTEMS
PROGRAMMING
SERIES

Fundamentals of Interactive Computer Graphics

J. D. FOLEY
A. VAN DAM

Fundamentals of Interactive Computer Graphics

JAMES D. FOLEY
The George Washington University
ANDRIES VAN DAM
Brown University

Addison-Wesley Publishing Company
Reading, Massachusetts • Menlo Park, California
London • Amsterdam • Don Mills, Ontario • Sydney

Sponsoring Editor: William B. Gruener
Production Editor: Rima Zolina
Designer: Herb Caswell
Illustration: ANCO/Boston
Cover Design: Richard Hannus
Art coordinator: Dick Morton

This book is in the
Addison-Wesley Systems Programming Series
Consulting editors: IBM Editorial Board

Library of Congress Cataloging in Publication Data

Foley, James D 1942–
Fundamentals of interactive computer graphics.

(The Systems programming series)
Bibliography: p.
Includes index.
1. Computer graphics. 2. Interactive computer
systems. I. Van Dam, Andries, 1938– joint author.
II. Title.
T385.F63 001.64'43 80-24311
ISBN 0-201-14468-9

Reprinted with corrections, July 1984

ISBN: 0-201-14468-9
NOPQR-HA-8987

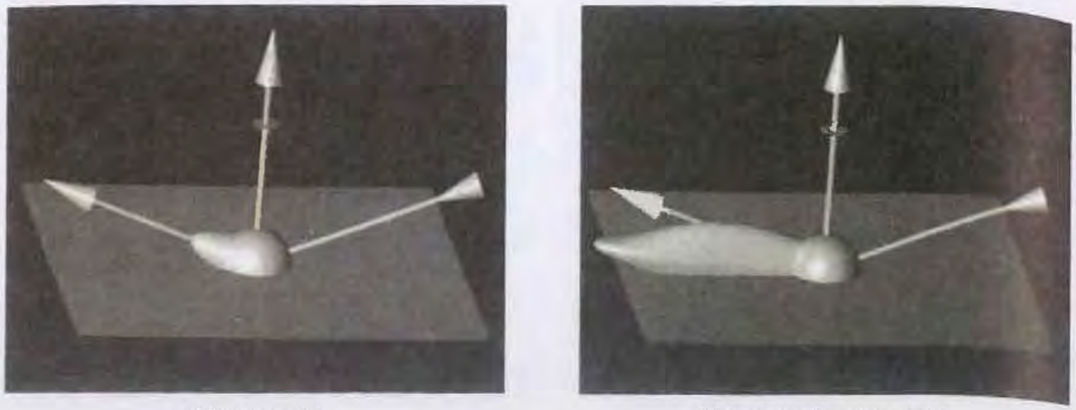


Fig. 16.5 Comparison of Phong and Torrance–Sparrow models for a 70° angle of incident light (by J. Blinn [BLIN77a], courtesy University of Utah).

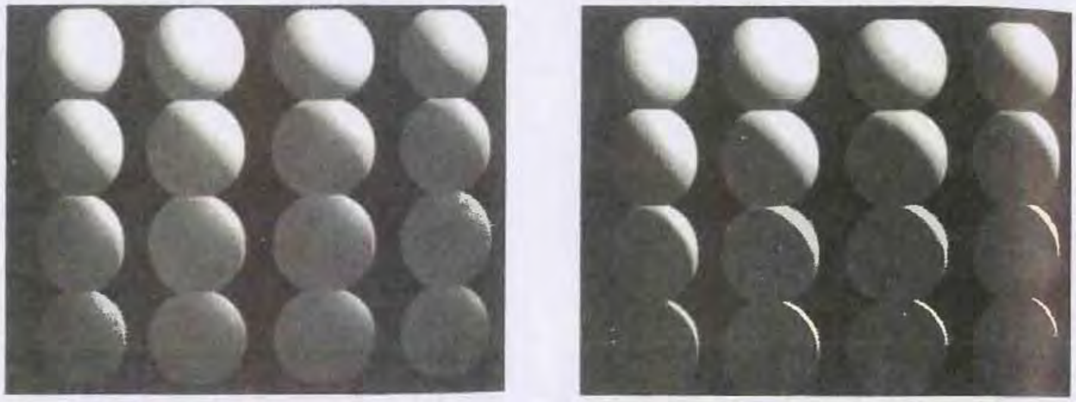


Fig. 16.6 Comparison of the Phong and Torrance–Sparrow models for a metallic sphere illuminated by a light source from different directions. Differences are most apparent for back-lit cases (bottom rows) (by J. Blinn [BLIN77a], courtesy University of Utah).

16.4 POLYGON MESH SHADING

There are three basic ways to shade objects defined by polygon meshes. In order of increasing complexity, they are: constant shading, intensity interpolation shading, and normal-vector interpolation shading. In each case, any of the shading models from the previous two sections can be used. Recall that color shading just involves three equations rather than one.

Constant shading calculates a single intensity value for shading an entire polygon. Several assumptions are made:

1. The light source is at infinity, so $\overline{N} \cdot \overline{L}$ is constant across the polygonal face;
2. The viewer is at infinity, so $\overline{N} \cdot \overline{V}$ is constant across the polygon face;
3. The polygon represents the actual surface being modeled, and is not an approximation to a curved surface.

If either of the first two assumptions is unacceptable, then an average $\overline{L}$ and $\overline{V}$ might be used, perhaps calculated at the center of the polygon.

The final assumption is most often the one which is incorrect and has a much more substantial effect on the resulting image than the other two. The effect is that each visible polygonal facet of the approximated surface is distinguishable, because each is a slightly different intensity than its neighbors. The difference in shading on adjacent facets is accentuated by the Mach band effect, which was discovered in 1865 by E. Mach and is described in detail in [RATL65]. The effect is one of exaggeration of intensity change at any edge where there is a discontinuity in magnitude or slope of intensity. Figure 16.7 shows, for two separate cases, the actual and perceived changes in intensity along a surface. The effect is caused by *lateral inhibition* of the receptors in the eye, whose response to light is influenced by adjacent receptors in inverse relation to the distance to the adjacent receptor. Receptors immediately to the brighter side of an intensity change have more response than those further from the edge, because they receive less inhibition from their neighbors on the darker side. Similarly, receptors immediately to the darker side of an intensity change will have less response than those further into the darker area, because they receive more inhibition from their neighbors on the brighter side.

Figure 16.8(b) shows a car with constant shading. The Mach band effect is quite evident. Even though the polygonal patches are quite noticeable, the image is much more realistic than that in Fig. 16.8(a), which shows only the polygon edges.

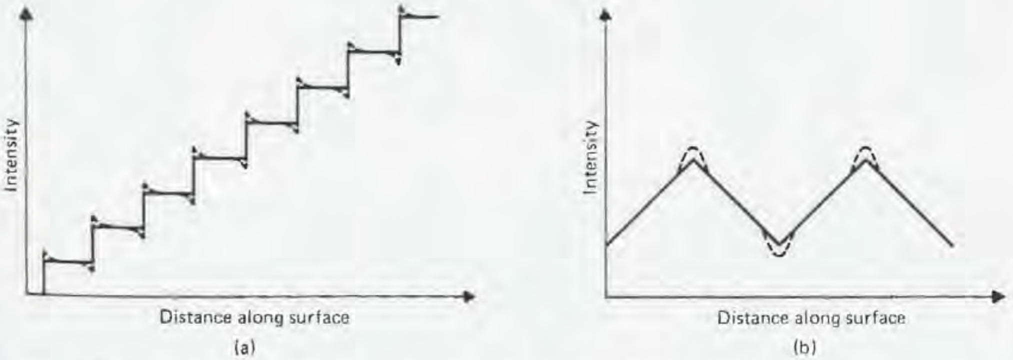
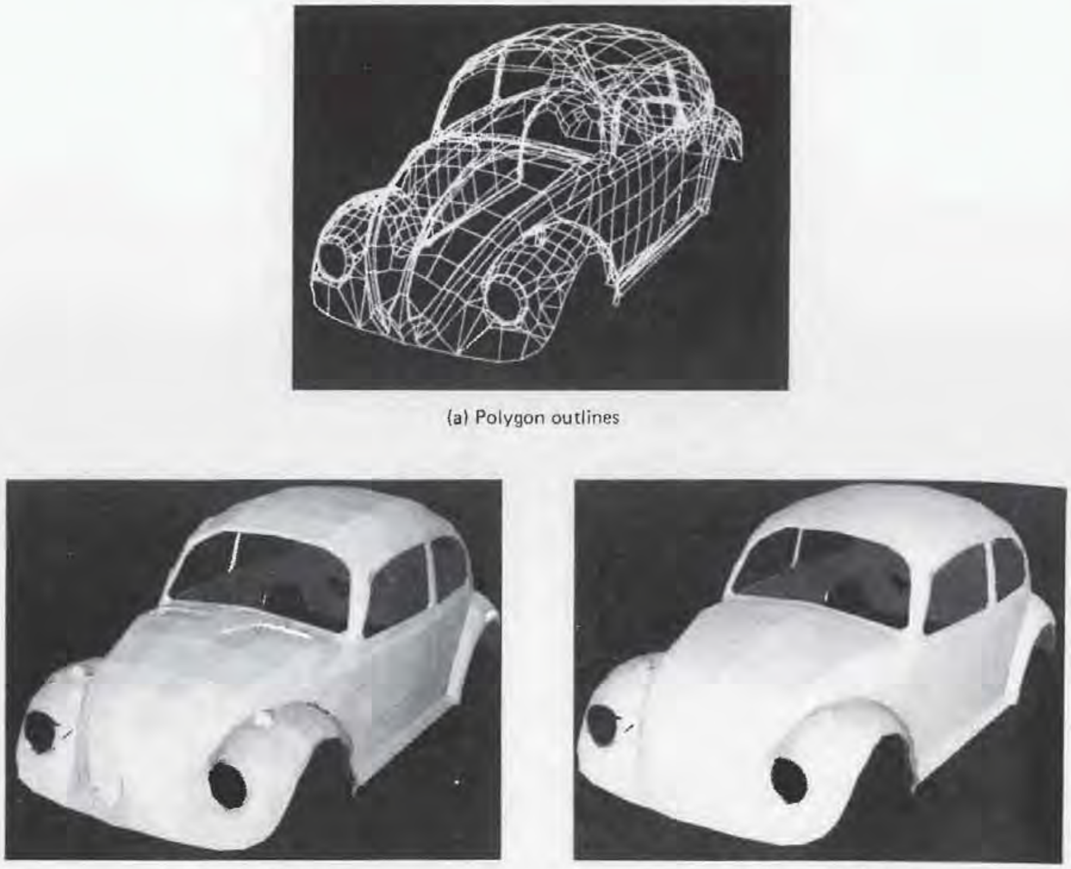


Fig. 16.7 Mach band effect—actual and perceived intensities: dashed lines—perceived intensity; solid lines—actual intensity.



(a) Polygon outlines

(b) Constant shading

(c) Gouraud shading

Fig. 16.8 Car body displayed three ways (courtesy University of Utah).

Intensity interpolation shading, usually known from the name of its developer as *Gouraud* shading [GOUR71], eliminates intensity discontinuities. Figure 16.8(c) shows a Gouraud-shaded car. The intensity ridge running down the hood on the right side of the picture, close to the fender, is a Mach band caused by a rapid change in the slope of the intensity curve: Gouraud shading does not completely eliminate such intensity changes.

The Gouraud shading process consists of four steps. First, surface normals are calculated. Second, *vertex normals* are calculated by averaging the surface normals of all polygonal facets that share the vertex (Fig. 16.9). If an edge is meant to be visible (such as at the joint between a planes' wing and body), then two vertex normals, one for each side of the edge, are found by separately averaging the normals of polygons on each side of the edge. Third, *vertex intensities* are found by using the vertex normals with any desired shading model. Finally, each polygon is shaded by linear interpolation of vertex intensities along each edge and then between edges along each scan line (Fig. 16.10).

DirectX 10 Architecture

for
Chrome 400 Series
Discrete Graphics Processors

A
S3 Graphics
White Paper

ATI 2087
LG v. ATI
IPR2015-00326

www.s3graphics.com

WP016-A.0
7/21/2007

Introduction

This White Paper provides an overview of Microsoft's DirectX 10 architecture. DirectX 10 compatible features provide an important component of the enhanced experience available to users of systems featuring S3 Graphics Chrome 400 Series graphics processors. These processors are designed especially for Microsoft DirectX 10 and Windows Vista.

DirectX 3D is the standard API (application programming interface) that allows graphics hardware to render and support graphics on Microsoft Windows platforms. This API is a common interface or middleware that provides a hardware abstraction layer which allows developers of an application, such as a 3D game or CAD program, to access the graphics hardware via programming calls to the operating system (OS). When the application makes a request to draw an image on the screen, the API calls the OS, which in turn will invoke the graphics processor (GPU) driver to communicate with the graphics hardware to draw the corresponding image and output the result to the display. By using the standard DirectX interface, application developers need only need be concerned about their specific application, without needing to be concerned about details of the underlying hardware implementation. This allows developers to quickly create many visually stunning images and realistic detail, by providing fast access to the advanced hardware capabilities of today's leading edge GPUs.

Previous generations of Microsoft's DirectX (DX) 3D, had significant changes. Fixed function hardware units were used in DirectX generations up to DX7. Then programmable hardware shader units with new user-defined programming capabilities appeared for DX8. DX9 featured added hardware functionality and programmability. The latest release from Microsoft is DX10, which introduces a new architecture that is the subject of this white paper. S3 Graphics continues to work closely with Microsoft's DirectX team to extend its leadership in graphics technology with high performance parts, including the Chrome S20 Series processors based on DX9.0c, and the Chrome 400 Series graphics processor line with advanced DX10 support.

Microsoft's latest DirectX release, DX10, extends the API beyond the limitations of previous generations. The DX10 API is the first redesign to the underlying architecture of DirectX 3D. New are optimized run-time features, CPU off-loading during state changes, a hardware geometry shader, texture arrays, and other graphics rendering enhancements that allow cinematic-like image quality. S3 Graphics Chrome 400 Series product line fully supports these new capabilities of DX10.

DirectX 9 Hardware Pipeline

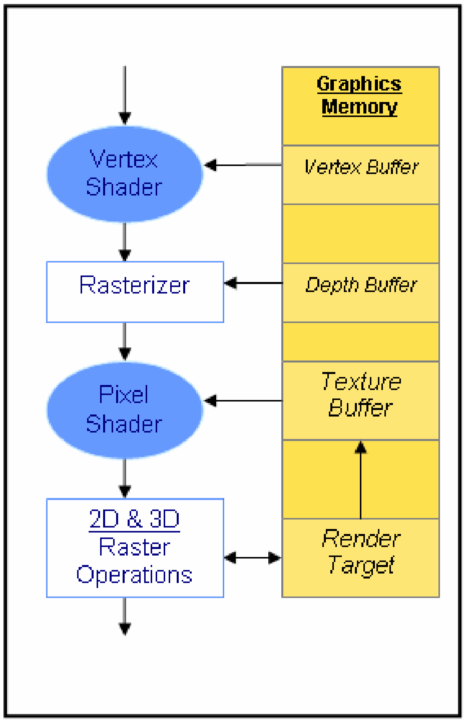


Figure 1. DirectX 8/9 Graphics Pipeline

The DirectX 8/9 pipeline is diagrammed in the above figure, which shows the basic features of a DX8/9 GPU. With the introduction of DX8/9 GPUs such as S3 Graphics' Chrome 20 Series, the transition from a fixed to a programmable pipeline changed the graphics landscape. Fixed function pipelines meant the hardware blocks were hard-coded with specific graphics algorithms. Application developers had to limit their development to what the hardware supported. DirectX 8 introduced and DirectX 9 expanded programmable pipelines which provided an additional programmable API layer closer to the graphics hardware. Developers can take advantage of this layer by using shader assembly language to creatively write specific code to control the different shaders and elements of the programmable pipeline. The main parts of the pipeline are as follows:

- ***Vertex shaders*** **(VS)** replace the "Transform and lighting engine" logic prevalent in previous generations of graphics hardware. The VS can only manipulate vertices and transform the shapes of objects from the 3D model space to be displayed on a 2D screen. The VS also does per-vertex lighting based on computed color to give the vertex more detail. The VS cannot create or destroy any vertices and the unit can only work on one vertex at a time (in the API level). Actual graphics hardware may process batches or packets of vertices in parallel to increase throughput.

- ***Rasterization*** is the process of mapping a triangle from object to image space (combining vertices from the VS output) and determining which screen pixels cover the triangle. All pixels inside the triangle are tested for visibility using the depth buffer and are kept if the triangle being rasterized is closer from a viewer perspective than other triangles. All invisible triangles and pixels are discarded since they will not be seen onscreen. This step prepares the object to be modified at the pixel level by the pixel shader.
- ***Pixel shaders*** **(PS)** calculate color and texture on each individual pixel. They give flexibility to developers by allowing high quality details to be shown on each object.
- The 2D and 3D ***Raster Operation Pipeline*** **(ROP)** is responsible for outputting the rendered object to the render target buffer from the pipeline after textures and blending have been applied.
- The graphics memory stores vertex and texture data in addition to the final object and frame that is to be drawn onscreen.

Limitations of DirectX 8/9

Microsoft designed DirectX 10 to address some of the following key disadvantages identified in the DirectX 8/9 architectures. These include the following.

1. API overhead is high for DX8/9

- When a DX9 application requires use of the graphics hardware to draw an object onscreen, the application needs to perform a call to the DX8/9 API to tell the OS what to do. The OS would then call the graphics driver, which would instruct the graphics hardware to perform the assigned task for the application.
- The DX8/9 API runtime provides resource management like allocation, virtualization, and initialization for the graphics hardware for vertex buffers, texture maps, and state changes. With the introduction of programmable shaders, runtime allocation tends to be harder to manage since there are more levels of abstraction, control and detail per scene.
- As Figure 2 illustrates, all DX8/9 functional or runtime calls from an application to the graphics hardware were done by the CPU (once per object), causing CPU bottlenecks whenever many objects need to be rendered for the current frame. The high API overhead also limits the number of objects per scene, causing potential loss of detail in each frame.
- State changes within the GPU for the shaders and textures generate additional overhead as the CPU had to decode state change instructions in order to implement visual details for realistic rendering to object surfaces and textures. If multiple visual effects need to be performed on an object, multiple passes through the hardware could be required. That translates into multiple state changes per pass performed by the CPU.
- The CPU overhead required to direct the GPU, in essence, became the bottleneck between the DX8/9 application and DX8/9 hardware, which limits the overall capacity for creating stunning visuals.

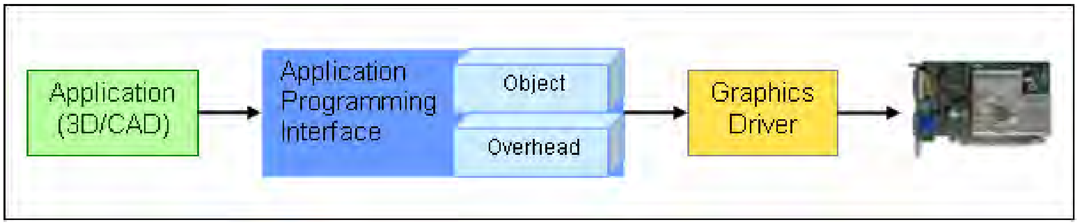


Figure 2. DirectX 8/9 Graphics API Interface

2. Vendor Variations in DirectX features

- Differences in DirectX feature support across GPU vendors, and even within a vendor’s product lines, frequently cause problems for application development, because of the numerous levels of support that must be implemented across these multiple hardware platforms. Typical issues that must be addressed include allowing for the resource limitations of different hardware. The lack of support for optional features may prevent an application from running at its highest level on a particular platform. Differences in arithmetic precision in the hardware shaders, instruction and data types, and variations in storage of intermediate data formats may all affect the rendering process.

3. Hardware resource limitations

- The number of separate VS and PS units in the GPU are fixed. Applications requiring heavy use of one type of shader will cause the other shader to be idle. For example, large triangles create heavy loading for the PS, idling the VS. Since there is the possibility of an application being shader-limited, throughput of the rendering pipeline will be limited by the number and type of shaders. The DirectX 10 API allows graphics hardware to overcome this resource bottleneck by introducing a novel architecture which has been incorporated into all S3 Graphics Chrome 400 Series graphics processors.
- Another drawback of DX8/9 generation GPU architectures is their specialized focus on one task, that of 3D graphics rendering. A DirectX 8/9 GPU was designed to optimize that task, which in effect limited its ability to perform additional computing tasks. With the introduction of GPU architecture designed for DirectX 10, the GPU can now take on additional capabilities and complexity. The processor now becomes more of a general compute processor, capable of offloading the CPU from some basic tasks, and thus takes a significant step towards becoming a general purpose GPU (GPGPU).

DirectX 10 New Capabilities and Benefits

The main objective for redesigning the entire DirectX 10 API and hardware architecture was to provide a solution for the CPU overhead problem and the hardware capability issues. This re-design also incorporated benefits from the application development, API, and hardware perspectives. The improvements based on this extensive research and re-design are as follows:

1. Efficient runtime (lower API overhead)

- In DirectX 10 the number of possible states that need to be tracked by the system has been reduced, minimizing the overhead related to state changes.
- Overhead per object has been reduced which allows more objects per frame. This produces better graphic realism and a higher level of detail than was possible with previous generations of the API.
- The validation of objects has been redefined and the process is now more efficient. Validation checks the format of commands and the integrity of data sent by the application to make sure there are no interoperability issues with the hardware. The drawback of validation is a large CPU overhead at runtime. DX10 uses this feature minimally by only validating each object once when it is created, rather than every time the object is used, as was the case in DX9.

2. Reduced CPU loading

Rendering an object or applying multiple textures to an object in a repeated manner uses up valuable CPU cycles and overhead. DX10 has introduced several new instructions and hardware capabilities to help overcome rendering limitations.

- A new 3D pipeline unit called the geometry shader (GS) has been introduced with this iteration of DirectX. The GS can modify, create, or destroy primitive vertex data from the VS without CPU intervention, so no resource-intensive state changes or associated overhead is required by the API. In the past, any changes to the vertex data needed CPU-GPU coordination and state changes.
- The new hardware model in the DX10 pipeline gives more capability to the GPU to handle state changes and instructions. The DX10 GPU now includes built-in arithmetic and flow control logic, thereby providing flexibility in primitive shading and state change handling, and offloading the tasks once performed by the CPU.
- ***Stream out*** is a new feature that allows the VS/GS to output data directly into graphics memory where the data can be accessed automatically and repeatedly by the shader units. This is a great new feature controlled entirely by the GPU (with no CPU overhead), for recursive rendering on objects that require multiple passes through the pipeline. In addition, data from any step in the pipeline can go directly to memory. By avoiding the need to send data completely through the pipeline, resources are not wasted on processing intermediate vertices or pixels.
- ***Arrayed resources*** allow texture maps to be created as a linear array of up to 512 elements. Developers now have index instructions to access elements within the array in a single pass, so the GPU can work on multiple elements without any static switching overhead. For example, an environmental cube map can be stored in an

array as six elements (one for each face of the cube), and the GPU can work on all six elements concurrently in one pass.

- ***Multiple render targets*** allow the GPU to create different versions of a scene in a single pass. DX10 has the ability to create up to eight render targets at a time.
- ***High dynamic-range rendering*** is another feature that brings realistic graphics rendering to the user experience. Formats used in the past to represent color in floating-point representation took at least twice the amount of storage compared to integer formats with half the precision. DX10 provides more efficient mechanism for storage of color components by providing floating point format RGB 11:11:10 (R/G 11-bits each, B 10-bits) and RGBE format (5-bit shared exponent for R/G/B with 9-bit mantissas for each). These formats allow a wider range of color and more vivid detail to be represented as seen in the examples below.

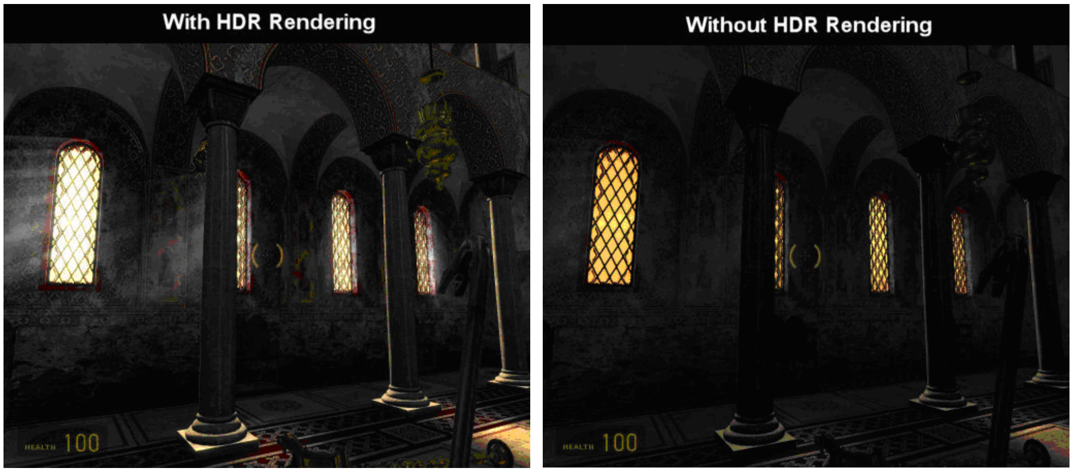


Figure 3. High Dynamic Range Rendering in Half-Life 2: Lost Coast

- A more complex method of utilizing occlusion query has also been implemented to conserve valuable GPU and CPU resources. *Occlusion query* is a method where non-visible primitives and objects in the Z-buffer are not rendered to save hardware computational resources. Because frame-to-frame rendering is very dynamic and implementation may vary with the application, occlusion query is not guaranteed to omit unseen pixels. DX10 takes it one step further by allowing the GPU to render complex objects in simple line drawing approximations. If the object needs to be drawn onscreen, the GPU already has the framework ready. If the object is not needed or is invisible onscreen, the GPU can throw away the object approximation, without wasting many CPU and GPU cycles.
- *Data and resource mapping* enhancements improve the ability of the GPU to access data in a timely manner. As an example, vertex buffer data for an application needs to be mapped to its memory address space, since that data can only be used by the application. The API and driver will allocate this buffer space at runtime either from the graphics memory (frame buffer) or system memory. While access to frame buffer is almost instantaneous, access to system memory is many magnitudes slower because of the communication lag between the GPU and system memory via the chipset. In DX10 resources are mapped according to how frequently they are used

(reads/writes) with four resource classes: default, immutable, dynamic, and staging. Using these new resource classes, developers can optimize performance by putting frequently used data in the frame buffer to be closer to the GPU and putting seldom-used data in system memory.

3. Additional constructs to improve efficiency

In graphics, multiple iterations of textures and blending usually take place to produce realistic images, such as re-creating hair moving in the wind or the ripples on the surface of a lake. These multiple loops previously required state changes and extensive CPU work to be performed. With DX10, state objects and constant buffers are now available to manage multiple loops in the rendering pipeline and increase the range of processing that can be done in one pass.

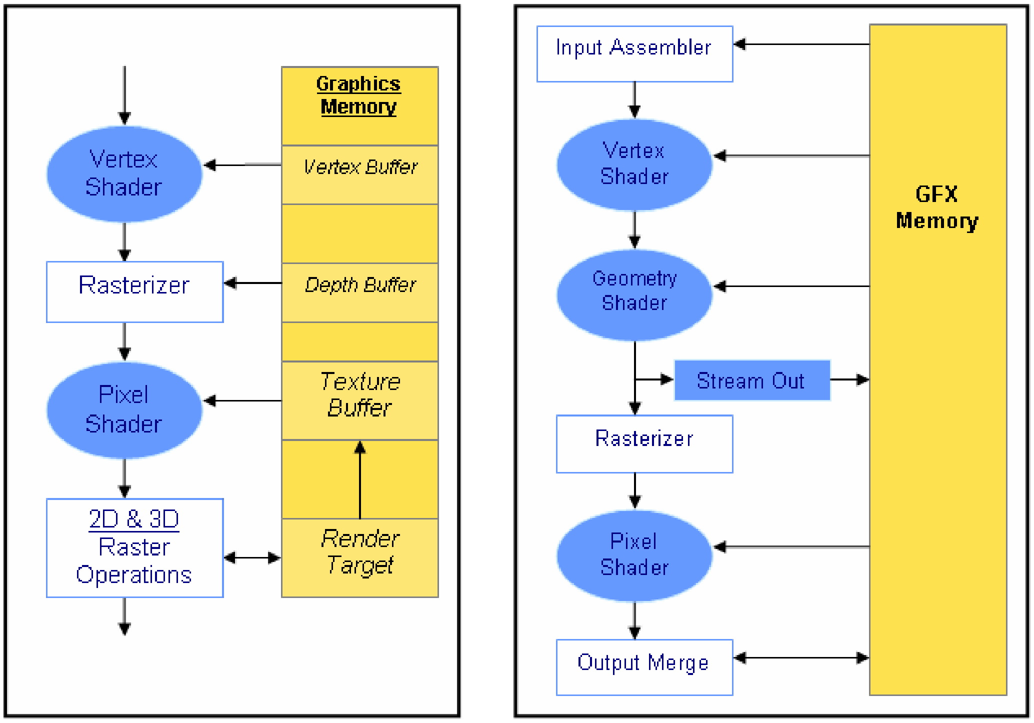
- *State objects* define what the graphics pipeline units should do as an object travels through the pipeline. The state objects have information to tell the pipeline which textures to blend or to tell the GS to create more detail for specific vertices in a part of the rendered frame. DX10 handles all of these details by introducing five state object commands, and programmers can work using a high-level language, instead of low-level constructs where they would need to keep track of all the pipeline stage units. The commands *InputLayout*, *Sampler*, *Rasterizer*, *DepthStencil*, and *Blend* are performed in the GPU, with minimal CPU intervention for state changes.
- *Constant buffers* store large amounts of predefined values (data) for items in a scene, so the CPU or GPU does not have to keep track of those values constantly. Each buffer stores up to 4096 constants hold information such as camera view/projection and light source color/position/intensity. Since these items have update intervals which may be once per frame or once per object, doing several hundred of these constant updates one at a time required significant CPU overhead when done using DirectX 9 or earlier. In DX10, the constant buffer groups the constants based on frequency of use and does batch processing to update the constants, which significantly reduces CPU use and dependence.

4. DX10 hardware specification set

- In pre-DX10 revisions of the API, hardware vendors were able to provide capability bits to inform the system about what features were supported in hardware. DX10 has changed this scheme. DX10 binds the 3D hardware feature set with a DirectX version number, so consistency across all hardware vendors exists and identifies support for a set of same basic features. Implementation of these features is the key for differentiating the graphics quality seen across vendors. S3 Graphics Chrome 400 Series processors have proprietary design implementations that give them an edge over the competition in visual quality and rendering capability.
- The basic programmable and fixed function pipelines of the past have been redesigned or eliminated. New and powerful hardware is capable of extending the implementation scope beyond the limits of rendering-only applications to provide high-throughput computing implementations for physics and AI.

The additional capabilities available in DX10 hardware and software offload most of the runtime events associated with rendering an object from the CPU to the GPU. With new and even more powerful functional units, an expanded instruction set, a new architecture that streamlines the graphics pipeline, efficient memory access methods, and multi-pass rendering capability, DX10 has introduced an astounding ability to generate graphics realism into our computing lives today and in the immediate future.

DX10 Pipeline Introduction



DX8/9 Pipeline **DX10 Pipeline**

Figure 4. DX8/9 Pipeline Compared to DX10 Pipeline

Figure 4 shows a DX8/9 pipeline (left) compared to the latest DX10 hardware. Common functional units like the vertex/pixel shaders, rasterizer, and ROP/Output Merge (OM) block exist in both architectures and perform similar functions. The Input Assembler (IA) converts or replicates input vertex data from incoming streams (vertex structure) to be used by the pipeline. The key difference is the introduction of the Geometry Shader (GS) and Stream Output (SO) described below.

- The *Geometry Shader* (GS) takes the vertices of a primitive, such as a line, point, or triangle, and will either create additional vertices (generate data) or destroy the vertices. The GS increases the number of vertices by creating additional primitives composed of up to 1024 32-bit vertex data per instance. If a vertex is not needed, the GS can delete it from the rendering pipeline. The GS can also add additional elements to a primitive without needing to create a new vertex stream. In the past, the pipeline could not create or destroy vertices, only modify them. DX10 moves one step ahead by allowing even more flexibility and power in hardware, where the GS performs per-primitive modifications and also accesses adjacent primitive information. For example, in a GS implementation for a realistic shadow rendering, the GS can control a point or line and its neighboring primitive, as well as control the displacement mapping, where more detail can be shown with the creation of new primitives based on height maps.

- *Stream output* can write vertex or primitive information from the VS/GS to a stream buffer in memory immediately after the GS stage. In the past a primitive had to exit the PS and then it could be written to the render target buffer in memory. Now data in the stream buffer can be used recursively or iteratively by other functional blocks in the pipeline on an as-needed basis to improve data re-use efficiency. Other uses of this feature are physics calculation support such as used in particle systems where ongoing calculations are needed to generate and destroy primitives to simulate water, smoke, and clouds.
- Graphics memory has been changed from an area that stored vertex and texture data separately, to memory where each independent shader unit can access the same data. The data storage formats have also been updated to allow the pipeline to store and use multiple format types to increase flexibility. The memory can store data in arrays which allows recirculation of data and texture fetches by the VS, GS, and PS.

DX10 introduces a unified architecture that builds upon the pipeline diagramed in Figure 4. The DX10 pipeline combines three types of shaders into unified execution units capable of handling VS, GS, and PS instructions. The DX10 pipeline architectural design has solved many issues seen in previous generations of DirectX. It has added significant changes to the software, instruction set, instruction support/capability, as well as requiring new hardware blocks and features.

This new design still has a limitation in the utilization rate for each shader type based on the application (see Figure 5 below). If complex geometry calculations and processing are needed, then the VS may be fully utilized while the PS might remain idle. If an application is pixel processing heavy, then the PS may be running at full speed while the VS will only be partially loaded.

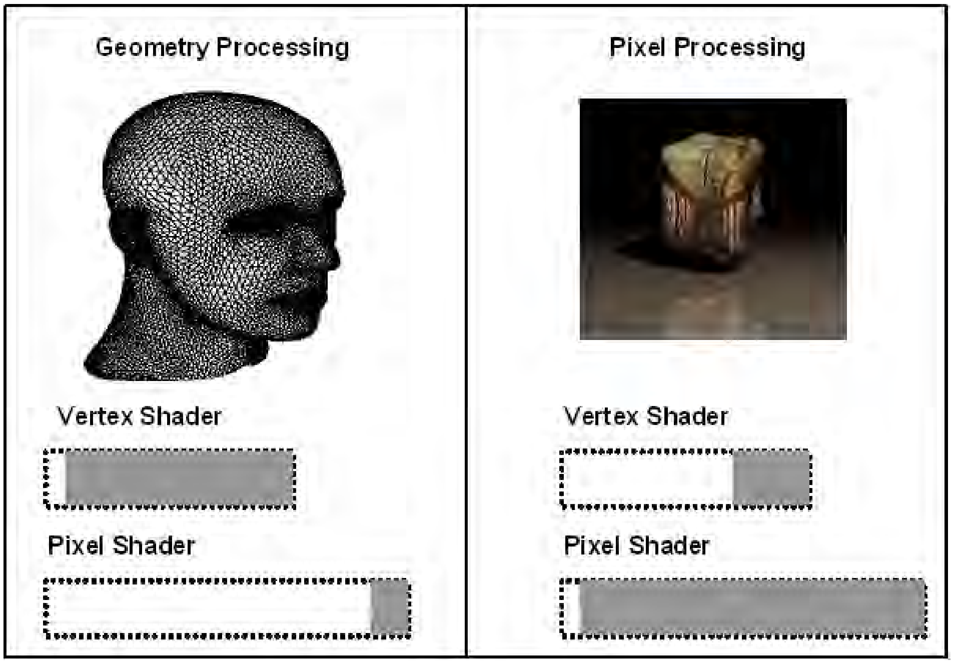


Figure 5. Vertex and Pixel Shader Resource Utilization for Different Applications

DX10 Shader Model 4.0

Shader Model 4.0 (SM4.0) is the new instruction set architecture (ISA) for DX10 that looks at the graphics in a unified way. Some keys advantages of SM4.0 are:

- **Easy Programmability:** Developers do not need to be bogged down with the low-level details of the hardware. In the past, programmers needed to control and write different low-level program code for each individual shader (VS/PS). Each shader was also considered an individual virtual machine that had separate input/output/general registers that had to be tracked for shader I/O, memory transfers, and intermediate data storage. SM4.0 instructions hide the low-level implementation details and incorporate all the pipeline flow control.
- **Flexible Load Balancing:** The new unified ISA allows flexibility. Developers can now look at these shaders units as one cohesive block (single common core virtual machine) instead of separate blocks, as shown in Figure 6. The unified shader is made up of shader blocks that can handle all vertex, pixel, and geometry instructions, so the GPU is fully utilized without concern for shader loading imbalances (geometry processing vs. pixel processing, as shown in Figure 5). There is also additional logic to load balance the shader units to keep all functional units fully utilized. If more pixel processing is needed, then more of the unified shader blocks can be allocated to pixel processing to increase throughput. The same shader-type allocation can be done with the VS and GS, as seen in Figure 7.
- **Unified Shader Code**: Developers code in “shader” instructions, not VS/PS/GS specific code.
- **Programmable Offloading:** The unified model helps offload the CPU state change overhead by incorporating flow control logic that programmers can control.

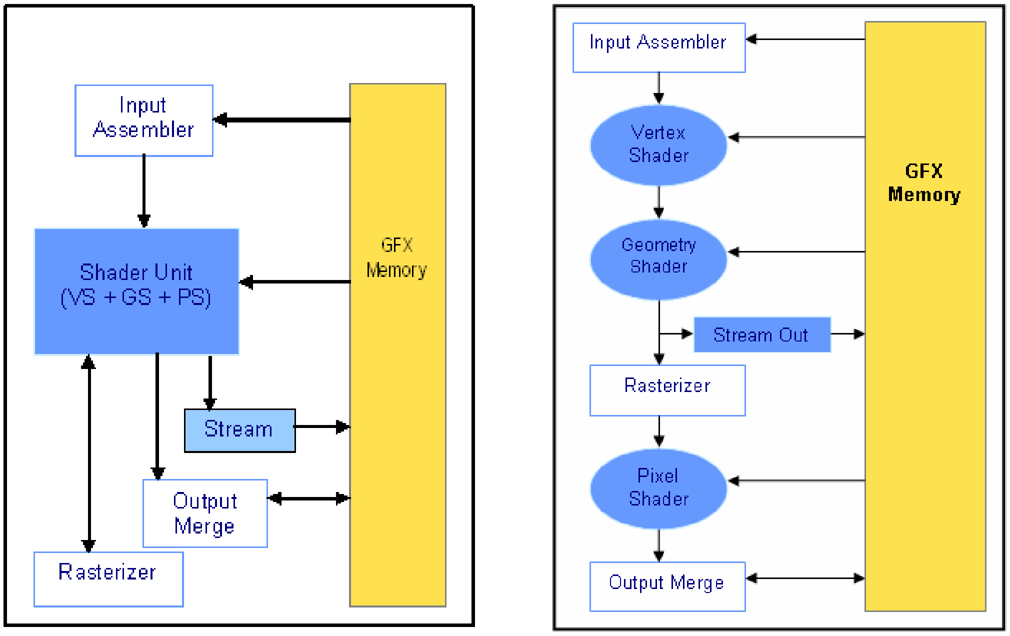


Figure 6. Unified Shader Model (Left) Compared to Basic DX10 Pipeline (Right)

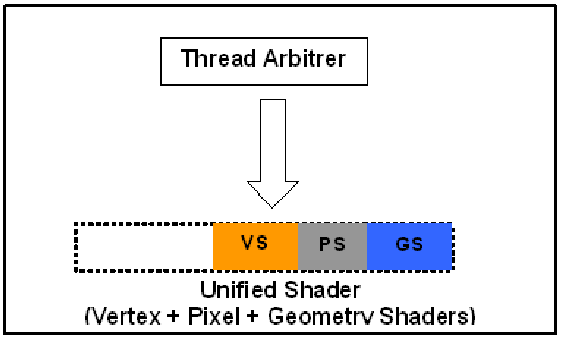


Figure 7. Unified Shader Utilized for Different Shader Types

- Additional resources, data formats, and instructions are available to the programmer for more efficient use and coupling of the graphics hardware to the application level.

Feature	DX9	DX10
Instruction Slots	512	64K
Constant Registers	256	4096 (x16)
Temporary Registers	32	4096
Render Targets	4	8
Textures	16	128
Texture Size	2K x 2K	8K x 8K
Load Operations	No	Yes
Sample Offsets	No	Yes
Flow Control	Static/Dynamic	Dynamic

Table 1. Basic Comparison Table of DX9 and DX10

- *Full integer and bitwise instructions* allow the GPU to compute complex algorithms more efficiently in integer format instead of converting between floating point and integer.

- *Switch statements* are another great addition because they provide multiple paths/options when rendering objects on a per-primitive basis. This means the GPU can replicate objects (instancing) and also provide unique characteristics for each object independently of the other objects in the scene.

- Increased texture support and size greatly enhance visual quality.
    - In DX9, developers only had 16 textures to work with at a given time and their size limitation was 4096 x 4096. Since the application of multiple textures required multiple texture changes and multiple state changes (large CPU overhead), developers were limited in what they could do. If developers needed multiple textures, they created a texture atlas that combined many smaller textures that could be accessed by indexing into the atlas. This method proved very inefficient since the boundaries between smaller textures were not as clearly defined and the atlas could only hold a certain amount of textures so there was a trade-off between storing fewer (larger) textures or more (smaller) textures.
    - SM4.0 has new instructions and indexed texture arrays, which can store 512 textures with resolutions of up to 8192 x 8192. This method effectively replaces the texture atlas with a large array that can be indexed into easy-to-access multiple textures. In addition, the number of textures a shader can use has increased from 16 to 128, allowing hardware to take advantage of the texture array to add more detail to all objects in a frame.

High-Level Shader Language (HLSL 10)

HLSL 10 is the name given to the programming language developers use to take advantage of DX10 shader and hardware capabilities. The advantages of HLSL are many.

- Application developers do not need to worry about using assembly-like instructions to control the shader and pipeline at the hardware level.
- Applications can offload the task of resource management.
- *Bind-by-name* to *bind-by-position* allows less overhead at runtime. Bind-by-name in DX9 performs checks like matching input/output between hardware functional units and matching vertex buffer format with the vertex shader. Any type mismatch would cause huge overhead since the hardware was not as flexible. In DX10 shader units have associated signatures with their inputs and outputs. As long as the output of the preceding stage is compatible with the input of the following stage, then the data type mismatch is allowed since DX10 allows multiple data formats to be used at any stage in the pipeline.
- DX10 has a "view" method for representing resources such as vertex buffers or texture maps, which can be read in many different formats, so that they are not type-set. This allows resources to be used in multiple parts of the pipeline. Data from one shader can be used in another for on-the-fly updates, regardless of the format type of the intermediate data.

Examples of DX10 Visual Effects

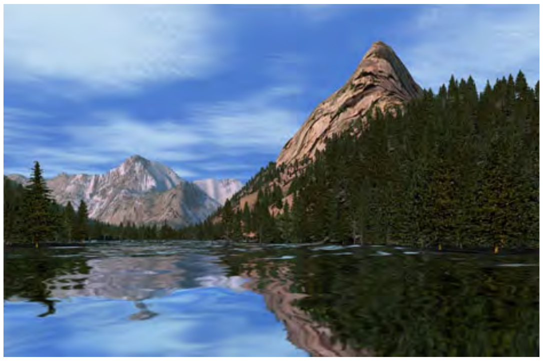


Figure 8. DX9 Screenshot

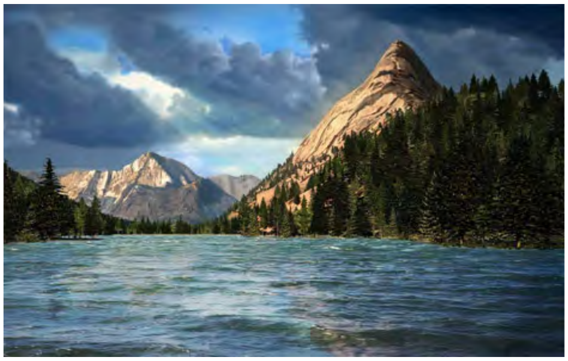


Figure 9. DX10 Screenshot

(Source: Flight Simulator X game)

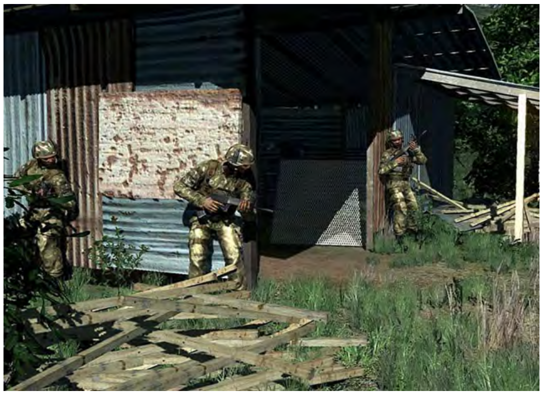


Figure 10. DX9 Screenshot

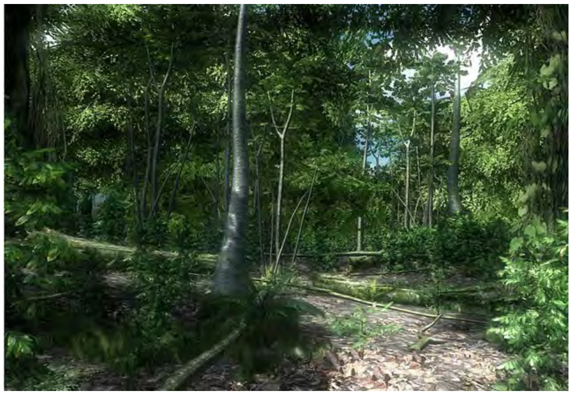


Figure 11. DX10 Screenshot

(Source: Crysis game)

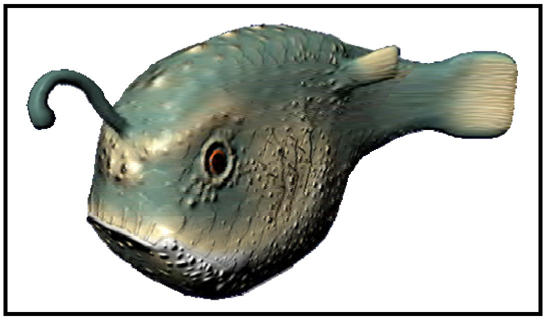


Figure 12.
DX9 (normal mapping)

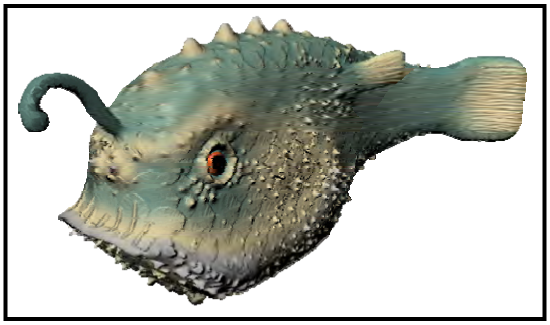


Figure 13.
DX10 (displacement mapping)

(Source: "DirectX 10: The Next Generation in Gaming" – http://windowsvistablog.com)

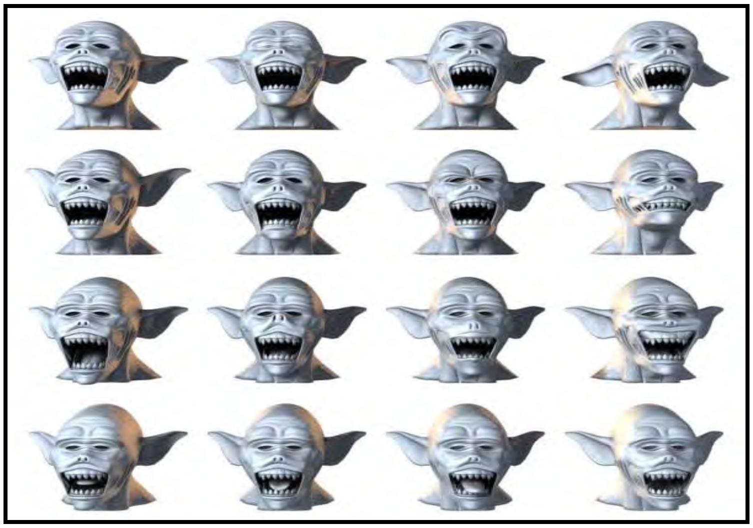


Figure 14. DX10 Morphing using the geometry shader and stream output
(Source: Microsoft MSDN Direct3D 10 Samples, http://msdn2.microsoft.com)

Figure 15. DX10 Alpha to Coverage
(Source: Microsoft "Intro to Direct3D 10" presentation by Sam Glassenberg)

Figure 16. DX10 Instancing
(Source: Microsoft MSDN Direct3D 10 Samples, http://msdn2.microsoft.com)

Conclusion

The introduction of DirectX 10 brings an inflection point to the graphics market where the rendering capabilities of advanced hardware and the creativity of software developers can now bring real-life 3D graphics to our daily lives. Features, which once were found only in luxury high-end graphics products costing several thousands of dollars, can now be achieved using the new DirectX 10 companion GPUs of the S3 Graphics Chrome 400 Series product line.

S3 Graphics Chrome 400 Series graphics processors are technological marvels with their multiple programmable DX10/SM4.0 execution units which provide a unified shader architecture. With support for all the new DX10 features such as *stream processing*, *geometry shaders*, and *HLSL 10 programming*, S3 Graphics continues its position as an industry leader in visual computing for current and future generations of the DirectX 3D API.

DICTIONARY OF COMPUTING

FOURTH EDITION

S.M.H. Collin

PETER COLLIN PUBLISHING

Fourth edition published 2002
Third edition published 1998
Second edition published 1994
First published in Great Britain 1988

Published by Peter Collin Publishing, an imprint of
Bloomsbury Publishing Plc
38 Soho Square, London W1D 3HB

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library

ISBN 1-901659-46-1

Text computer typeset by PCP
Printed and bound in Italy by Legoprint
Cover artwork by Gary Weston

vectored interrupt *noun* interrupt signal which directs the processor to a routine at a particular address

> QUOTE: the great advantage of the vector-scan display is that it requires little memory to store a picture
>
> *Electronics & Power*

Veitch diagram *noun* graphical representation of a truth table

velocity *noun* speed; ***the disk drive motor spins at a constant velocity***

velocity (of sound) *noun* speed of sound which is equal to 331 metres per second through air; the speed of sound varies in different materials

vendor *noun* person who manufactures *or* sells or supplies hardware *or* software products; **vendor independent** = hardware *or* software that will work with hardware and software manufactured by other vendors; *opposite is* PROPRIETARY; **vendor-independent messaging** = *see* VIM

vendor-independent messaging *see* VIM

Venn diagram *noun* graphical representation of the relationships between the states in a system *or* circuit

verification *noun* checking that data has been keyboarded correctly *or* that data transferred from one medium to another has been transferred correctly; **keystroke verification** = check made on each key pressed to make sure it is valid for a particular application; **verification and validation (V & V)** = testing a system to check that it is functioning correctly and that it is suitable for the tasks intended

verifier *noun* special device for verifying input data

verify *verb* to check that data recorded *or* entered is correct

Veronica tool that works with Gopher to help a user find information or files on the Internet

version *noun* copy *or* program *or* statement which is slightly different from others; ***the latest version of the software includes an improved graphics routine***; **version control** = utility software that allows several programmers to work on a source file and monitors the changes that have been made by each programmer; **version number** = number of the version of a product

vertex *noun* point in space defined by the three coordinates x, y, and z

vertical *adjective* at right angles to the horizontal; **vertical application** = application software that has been designed for a specific use, rather than for general use; ***your new software to manage a florist's is a good vertical application***; **vertical blanking interval** = *see* RASTER; **vertical format unit (VFU)** = part of the control system of a printer which governs the vertical format of the document to be printed (such as vertical spacing, page length); **vertical interval time code** = *see* VITC; **vertical justification** = adjustment of the spacing between lines of text to fit a section of text into a page; **vertical parity check** = error detection test in which the bits of a word are added and compared with a correct total; **vertical portal (VORTAL)** = website that contains information for just one particular industry or interest group; **vertical redundancy check (VRC)** = (odd) parity check on each character of a block received, to detect any errors; **vertical scan frequency** = number of times a picture beam in a monitor moves from the last line back up to the first; **vertical scrolling** = displayed text which moves up or down the computer screen one line at a time; **vertical sync signal** = (in a video signal) signal which indicates the end of the last trace at the bottom of the display; **vertical tab** = number of lines that should be skipped before printing starts again

vertically *adverb* from top to bottom *or* going up and down at right angles to the horizontal; ***the page has been justified vertically***

very large scale integration (VLSI) *noun* integrated circuit with 10,000 to 100,000 components

VESA = VIDEO ELECTRONICS STANDARDS ASSOCIATION; **VESA local bus** *or* **VL-bus** = (in an IBM PC) standard defined by VESA which allows up to three special expansion slots that provide direct, bus-master control of the central processor and allow very high speed data transfers between main memory and the expansion card without using the processor;

A 195 mW/152 mW Mobile Multimedia SoC With Fully Programmable 3-D Graphics and MPEG4/H.264/JPEG

Jeong-Ho Woo, *Student Member, IEEE*, Ju-Ho Sohn, *Associate Member, IEEE*, Hyejung Kim, *Student Member, IEEE*, and Hoi-Jun Yoo, *Fellow, IEEE*

***Abstract*—In this paper, we present a low power multimedia SoC with fully programmable 3-D graphics, MPEG4 codec, H.264 decoder, and JPEG codec for mobile devices. The mobile unified shader in 3-D graphics engine provides fully programmable 3-D graphics pipeline with 35% area and 28% power reduction. Low power lighting engine which employs logarithmic number datapath and the specialized lighting instruction enable 9.1 Mvertices/s vertex fill rate, which is 2.5 times improvement compared with previous works including transformations and OpenGL lighting. The SoC consumes less than 152 mW for video applications and less than 195 mW for 3-D graphics applications. The mobile unified shader and merged JPEG/MPEG4 codec reduce the silicon area and the SoC consumes 6.4 mm × 6.4 mm in 0.13 μm CMOS logic process.**

***Index Terms*—Low power design, mobile multimedia SoC, mobile unified shader, programmable 3-D graphics.**

I. Introduction

RECENTLY, multiple multimedia applications are merged into the mobile devices to be a personal multimedia terminal [2]. The digital camera and real-time audio playback are widely incorporated and recently even digital multimedia broadcasting (DMB) and real-time 3-D graphics are employed for mobile entertainments. The real-time 3-D graphics has been used for various applications such as 3-D games or 3-D user-interfaces and recently portable navigation devices (PND) try to employ the real-time 3-D graphics for 3-D map displaying and 3-D navigation services.

In mobile devices, since users often hold the small screens closer to their eyes, the average eye-to-pixel angle is larger than that of a PC [3]. Therefore, every pixel in mobile applications should be drawn with realistic 3-D graphics effects, which can be achieved by a fully programmable 3-D graphics. In PC graphics, dedicated vertex shader and pixel shader carry out the fully programmable 3-D graphics and they make realistic 3-D images [9]. But, that PC graphics architecture cannot be migrated into the mobile devices due to its silicon area and power consumption.

Since the mobile devices restrict silicon area and power consumption, various multimedia applications should be implemented with small area and low power consumption. Although there have been many publications on mobile multimedia solutions [2]–[8], these chips did not integrate the full multimedia functions such as digital camera, video, audio and real-time 3-D graphics on a single die due to its huge gate counts and design complexity. Moreover, they could not provide a fully programmable 3-D graphics pipeline, which is required for realistic 3-D graphics effects compatible with OpenGL|ES-2.0.

In this work [1], a low power multimedia SoC with full integration of a fully programmable 3-D graphics and MPEG4, H.264 and JPEG processing is presented for mobile devices.

For the purpose to achieve low power, small area, and high performance, the programmable 3-D graphics engine with unique unified shader architecture [10] is employed. Its mobile unified shader is power optimized single shader for mobile devices in contrast to that of console device which has multiple general purpose unified shaders. The lighting engine and specialized lighting instruction are adopted for low power and high performance.

This paper consists of six sections. The system architecture and video engine will be discussed in Section II, and the programmable 3-D graphics engine and details of mobile unified shader will follow in Section III and Section IV, respectively. The chip implementation of the SoC and performance comparison will be described in Section V, and finally the conclusion of our work will be made in Section VI.

II. System Architecture and Video Engine

Fig. 1(a) shows a block diagram of the developed SoC. It integrates a 3-D graphics engine dedicated for acceleration of the fully programmable 3-D graphics pipeline, the ARM9 RISC processor, video engine, display engine and other peripheral IPs. Since most of the current mobile multimedia SoCs employ the AMBA bus so that the IPs are connected to the single layer AMBA bus. The video engine is employed to support video application such as DMB or digital camera. It is dedicated to MPEG encoding/decoding, H.264 decoding and JPEG image processing using dedicated hardwired blocks. Since both the JPEG codec and the MPEG codec use DCT, IDCT, VLC and VLD units, the JPEG and MPEG codec shares those functional blocks to reduce silicon area and power consumption. The fully hardwired H.264 decoder consists of a content-addressable variable-length decoder (CAVLD), an inverse-transform, a motion compensator and de-blocking filter. The video engine

Manuscript received October 29, 2007; revised May 5, 2008. Current version published September 10, 2008.

J.-H. Woo, H. Kim, and H.-J. Yoo are with the Division of Electrical Engineering, Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 305-701, Korea (e-mail: denber@eeinfo.kaist.ac.kr).

J.-H. Sohn is with LG Electronics Company, Ltd., Seoul 137-724, Korea (e-mail: sohnjuho@lge.com).

Digital Object Identifier 10.1109/JSSC.2008.2001911

ATI 2089
LG v. ATI
IPR2015-00326

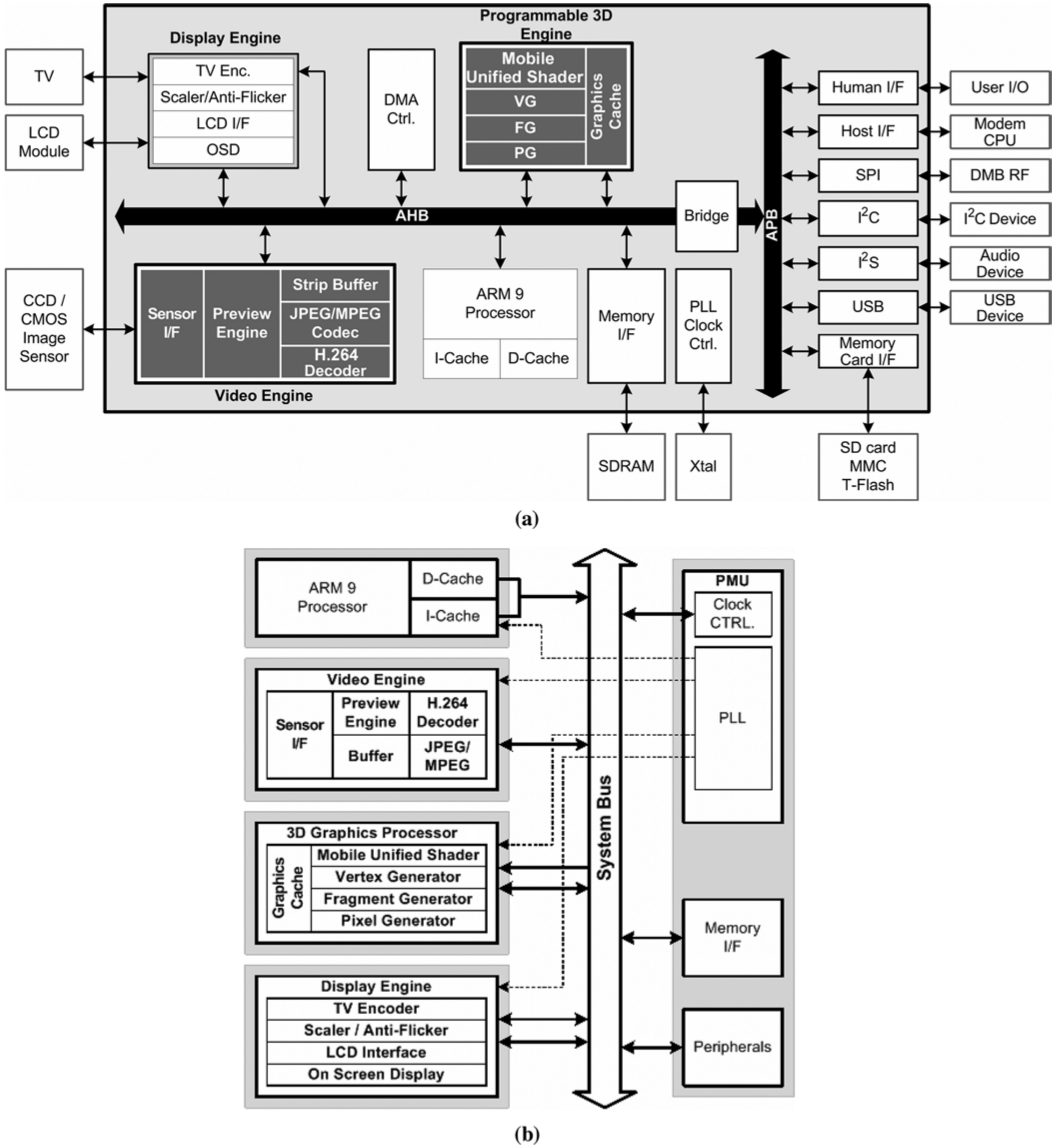


Fig. 1. Mobile multimedia SoC. (a) System architecture. (b) Power and clock domains.

supports up to 3M pixels image sensors, MPEG4 @ simple profile Lv.0∼3 encoding/decoding and H.264 baseline Lv.0∼3 decoding of 30 fps for CIF image resolution. During video operation, it consumes less than 152 mW at 1.2 V supply voltage and 48 MHz operating frequency.

For low power consumption, the developed SoC adopts block level clock gating and power gating. The SoC consists of five voltage islands and five clock domains as shown in Fig. 1(b). According to the operating mode, the power sources and clocks of the functional blocks are selectively turned off. In the sleep mode, all IPs except wake-up logics are turned off and the SoC consumes less than 150 μW. In normal operation mode, the RISC processor controls whether the IPs are turned off or not.

Since the IPs shares bus bandwidth, the bandwidth occupation is one of the important design issues in mobile multimedia SoC. During the 3-D graphics applications, the ARM processor computes application programs and transfer graphics processor (GPU) commands and primitive vertex indexes to the GPU

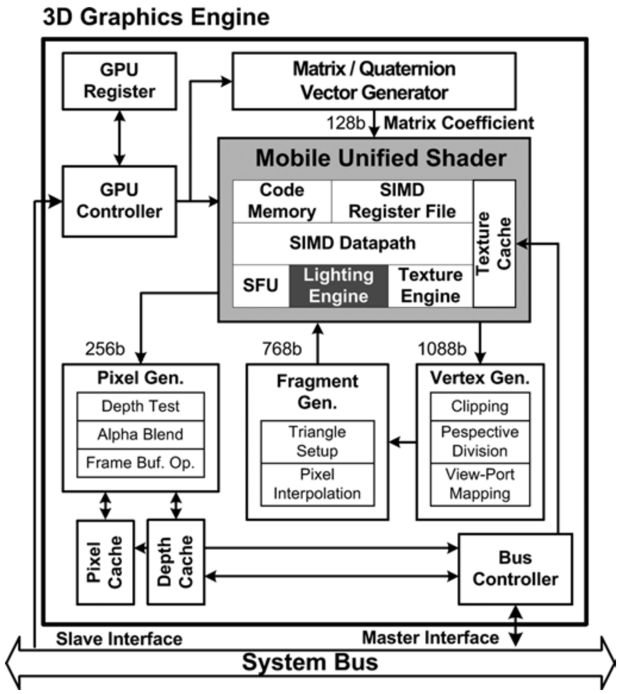


Fig. 2. Programmable 3-D graphics processor.

through the system bus. Using the GPU commands, the GPU performs geometry operations and rendering operations. For geometry and rendering operations, the GPU fetches primitive vertex attributes, depth, color, and texture from the system memory. Those graphics data are transferred through the system bus and they consumes about 700 MBs/s if the GPU computes the graphics data at the speed of 10 Mvertices/s–296 MB/s for primitive vertices and 400 MB/s for rendering. Although the AMBA bus has 400 MB/s bus bandwidth, it can provide less than 120 MB/s bus bandwidth to GPU because the ARM processor and other peripherals operate simultaneously with the GPU. To reduce the data transaction of the GPU, the GPU employs vertex cache [4] for primitive vertex, texture cache with 1:4 texture compression for texture, and 2-D direct mapped cache [3] for color and depth. With those techniques, the required data bandwidth is reduced to 116.4 MB/s for drawing 10 Mvertices/s.

III. Programmable 3-D Graphics Engine

A. Internal Architecture

The programmable 3-D graphics engine consists of Mobile Unified Shader, Vertex Generator, Fragment Generator, Pixel Generator, Matrix/Quaternion-Vector Generator and graphics caches as shown in Fig. 2.

The mobile unified shader is designed to perform both programmable vertex operation and programmable pixel operation, which are fully compatible with the mobile 3-D graphics API – OpenGL|ES2.0. Since the redundant functional blocks of the vertex shader and the pixel shader into a single hardware, the mobile unified shader reduces silicon area and power consumption. The SIMD datapath and special functional unit (SFU) of the mobile unified shader are responsible for the computational works and the texture engine [3] is responsible for texture address generation, texture fetch, and texture filtering. The vertex generator, fragment generator, and pixel generator are responsible for the rest of the 3-D graphics pipelines, clipping, triangle setup, rasterization, and blending, respectively. To reduce power consumption, the fragment generator employs low power SlimShader architecture [3].

TABLE I
Matrix Generator Command Set

Commands	Description
MG_Clear	Set Identify Matrix
MG_SET	Set Matrix With User Inputs
MG_SCALE	Scale a Matrix
MG_TRANSLATE	Translate a Matrix
MG_ROTATE	Rotate a Matrix
MG_TRANSPOSE	Transpose a Matrix
MG_INVERT	Inverse a Matrix
MG_Determinant	Get Determinant of Matrix
MG_MMUL	Matrix Multiplication

During the per-vertex operation, the mobile unified shader uses several floating-point matrices related to rotation, transportation, scale, and projection and those matrices are generated by the embedded RISC processor. Since the embedded RISC processor has only integer datapaths, the floating point computations are done by complex emulation equations. Therefore, thousands of cycles are consumed for matrix operations and this computation cycles limit the 3-D graphics performance. In order to accelerate those matrix operations, the mobile unified shader employs the matrix generator. The matrix generator consists of 6 floating-point multipliers, 6 floating-point adders, a floating-point divider, a floating-point square-root block and a floating-point trigonometric function block and it is controlled by the RISC processor using the dedicated matrix generator instructions as shown in Table I. With those matrix instructions, the matrix generator calculates transformation and projection matrices by matrix basis while embedded RISC processor computes by components basis. In addition, since the matrix generator and the mobile unified shader have independent commands and datapaths respectively, user can use the matrix generator and the mobile unified shader at the same time. For example, when the shader program contains complex matrix operations, users can use matrix generator instead of shader datapath to simplify the shader operations.

B. Pixel-Vertex Multi Threading

Fig. 3(a) shows a data flow diagram of the programmable 3-D graphics pipeline. In conventional architecture, the primitive vertices are computed in the vertex shader for per-vertex operations such as transformation and lighting. After per-vertex operations, vertex generator and fragment generator perform clipping and rasterization and they generate interpolated pixels (fragments). After that, the fragments are modified in the pixel shader using per-pixel effects and blending operations generate final pixel data. In contrast with conventional architecture, the

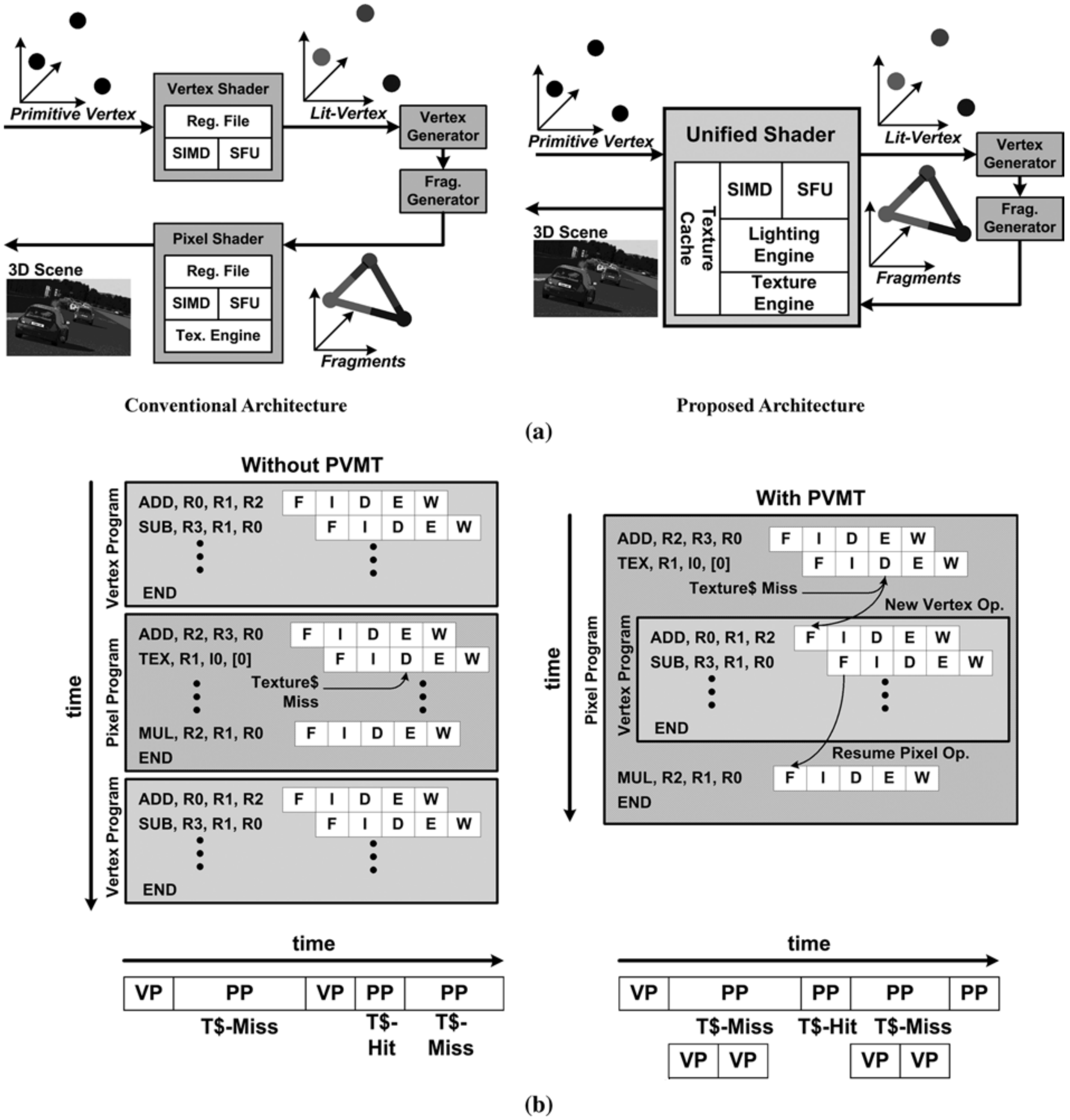


Fig. 3. Pixel-vertex multi-threading. (a) Data flow of the programmable 3-D graphics pipeline. (b) Pixel-vertex multi-threading.

mobile unified shader is responsible for both per-vertex operations and per-pixel operations in a single hardware. Therefore, the graphics data traverse the mobile unified shader twice in a single 3-D graphics pipeline as shown in Fig. 3(a) and thus, the 3-D graphics performance is limited less than the half of the its peak performance. Moreover, texture cache miss wastes a few tens of cycles until the cache is filled up and it degrades the graphics performance is degraded further.

To improve the 3-D graphics performance, the 3-D graphics processor adopts a Pixel-Vertex-Multi-Threading (PVMT), which utilizes datapaths of the mobile unified shader in parallel. Since the texture engine performs texture fetching and filtering independent with SIMD datapath and SFU, those datapaths are idle during the texture operations. Therefore, the PVMT enables SIMD datapath and SFU to compute per-vertex operations during the texture cache miss as shown in Fig. 3(b). When the texture cache miss occurs during per-pixel operations and vertex buffer contains vertices to be computed, PVMT issues next vertices and SIMD datapath and SFU perform per-vertex operations. If the texture cache filling is finished during the per-vertex operations, the mobile unified shader moves back to per-pixel operation and finalize the remaining pixel operations. Otherwise, the PVMT issues next vertices and performs per-vertex operations continuously.

Since the PVMT uses wasted cycles by the texture cache miss, the efficiency of the PVMT depends on the application characteristics such as vertex/pixel ratio or texture cache miss rate. In mobile 3-D graphics, the texture cache miss is occurred with about 10% probability in average and it consumes as short as 64 cycles, or 148 cycles at the most in the developed mobile

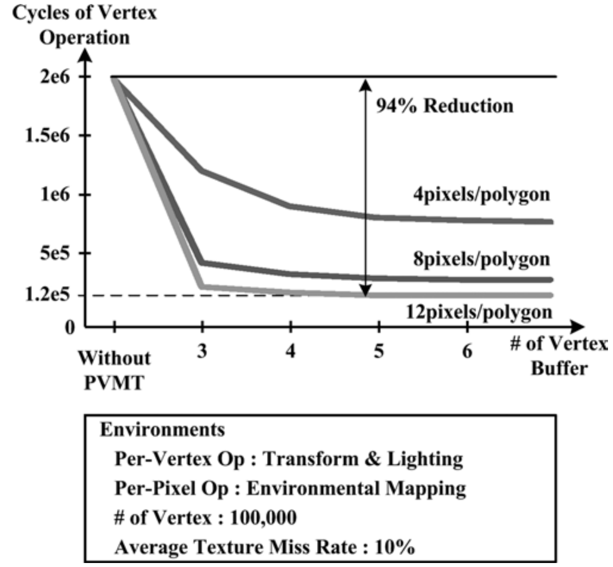


Fig. 4. Efficiency of pixel-vertex multi-threading.

multimedia SoC. Since the per-vertex operations such as transform and lighting (TnL) consume about 20 cycles in the test contents, more than three vertices could be computed during the texture cache miss if the vertex buffers are enough. Fig. 4 shows the efficiency of the PVMT versus the number of vertex buffers and the vertex/pixel ratio. While the effects of the PVMT vary from the vertex/pixel ratio, the PVMT reduces the cycles of the vertex operation as short as 60%, or 94% at the most. Although the PVMT reduces cycles of the vertex operations, the effect of the PVMT is bounded when the vertex buffer has five entries because the PVMT uses wasted cycles by the texture cache miss and the cycle counts are limited around 100 cycles in average. Therefore, the developed 3-D graphics processor has 5 entries vertex buffer for trade-off between hardware cost and the performance. By employing PVMT with five-entries vertex buffer, about 90% of the vertex operations are interleaved into the pixel operations and we can remove the cycle time of the vertex operations from the graphics pipeline.

IV. Mobile Unified Shader

A. Internal Architecture

The mobile unified shader is a SIMD processor for fully programmable 3-D graphics. It consists of 128 bits, 4 × 32 bit SIMD datapath, SFU, texture engine, logarithmic lighting engine, dedicated register file and control logic as shown in Fig. 5. The SIMD datapath is responsible to vector arithmetic operations such as Addition (ADD), Multiplication (MUL) and inner product (DOT). The SFU is dedicated to special functional scalar operations such as Logarithm (LOG), Exponent (EXP), Reciprocal (RCP) and Reciprocal-square-root (RSQ). And the texture engine and the lighting engine perform texture related operations and lighting operations, respectively.

Since the LOG, EXP, RCP and RSQ operations are difficult to compute in ordinary number datapaths, the SFU employs the logarithmic number system (LNS) [11]. In the LNS, since the ordinary number data should be converted into the LNS and the results should be converted into ordinary number, one of the critical issues is the data conversion error. Therefore, we employed more precise LOG and EXP converts, which use 16 piecewise-linear-regions. By employing precise LNS datapath, the SFU computes the LOG, EXP, RCP and RSQ in only two cycles keeping the error bound much below the standard graphics API, OpenGL|ES-2.0. Since the vertex shading is performed in IEEE-754 floating point number system and pixel shading is performed in 32 bit fixed-point number system, the SFU is designed to handle both number systems in a single hardware.

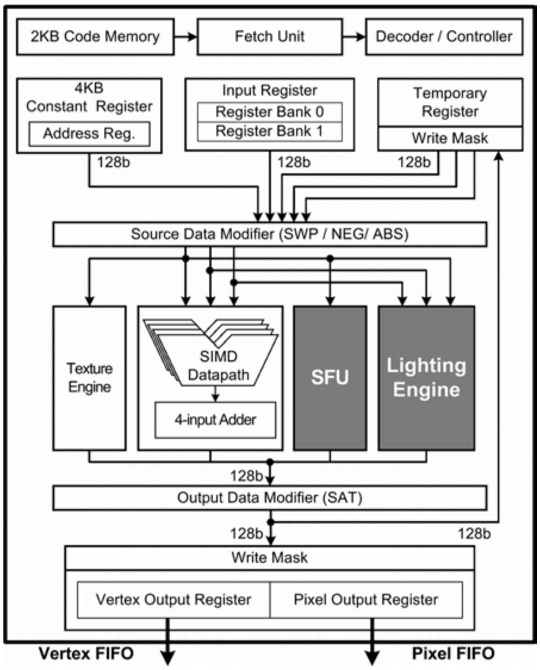


Fig. 5. Mobile unified shader architecture.

For streaming graphics processing, the mobile unified shader contains multiple register files—input registers, output registers, constant register and temporary SIMD registers. The input register is used to hold the vertex attributes such as position and normal vector and pixel attributes such as position, color and texture coordinate. In order to reduce data fetch time, the input register consists of two register banks for double buffering. The constant register stores the coefficients for the 3-D graphics operations. The temporal register is used to store temporary results during vertex program and pixel program execution. The modified vertex and modified pixel information are transformed into output register.

B. Unified SIMD Datapaths

The SIMD datapath is responsible for vector and matrix arithmetic operations such as Addition (ADD), Multiplication (MUL), and Inner Product (DOT). Since, in mobile 3-D graphics API-OpenGL|ES, per-vertex operations use IEEE-754 floating-point number system for wide dynamic range and per-pixel operations use fixed-point number system for high throughput, the conventional vertex shader has floating-point

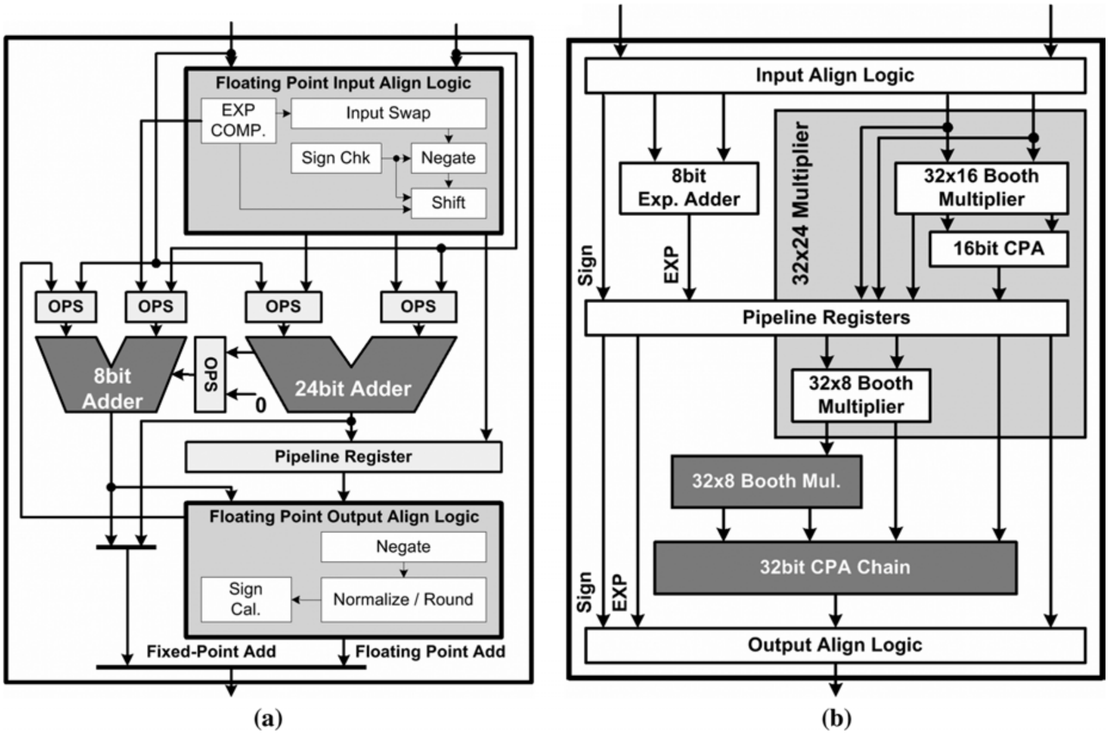


Fig. 6. Unified SIMD datapath. (a) Unified adder. (b) Unified multiplier.

SIMD datapath and pixel shader has fixed-point SIMD datapath. To compute both operations in a single mobile unified shader, we employed the unified datapaths, adder and multiplier, which computes IEEE-754 floating-point numbers and fixed-point numbers in a single hardware.

Since the fixed-point adder requires 32 bit addition and the floating point adder requires 8 bit addition for exponent and 24 bit addition for mantissa, the 32 bit adder can be shared for both number systems. Therefore, the unified adder has configurable 32 bit adder, which consists of a 24 bit adder and 8 bit adder as shown in Fig. 6(a). The configurable 32 bit adder is configured by operands selection and it computes floating-point addition with 2 cycle latency and 1 cycle throughput, and fixed-point addition in a single cycle. The unified multiplier consists of common 32 bit × 24 bit multiplier, optional 32 bit × 8 bit multiplier, and 8 bit adder for floating-point exponent as shown in Fig. 6(b) because the floating point multiplication includes 24 bit by 24 bit multiplication for mantissa and fixed point multiplication includes 32 bit by 32 bit multiplication. The final 32 × 8 multiplier is conditionally enabled and the CPA chain selects the input between 24 bit result and 32 bit result. The unified multiplier calculates both floating-point MUL and floating-point MUL with 2 cycle latency and 1 cycle throughput. By sharing the common functional blocks of floating point datapath and fixed point datapath, the unified SIMD datapath reduces silicon area by 47% and power consumption by 42% compared with separated floating-point and fixed-point SIMD datapaths.

C. Low Power Lighting Engine

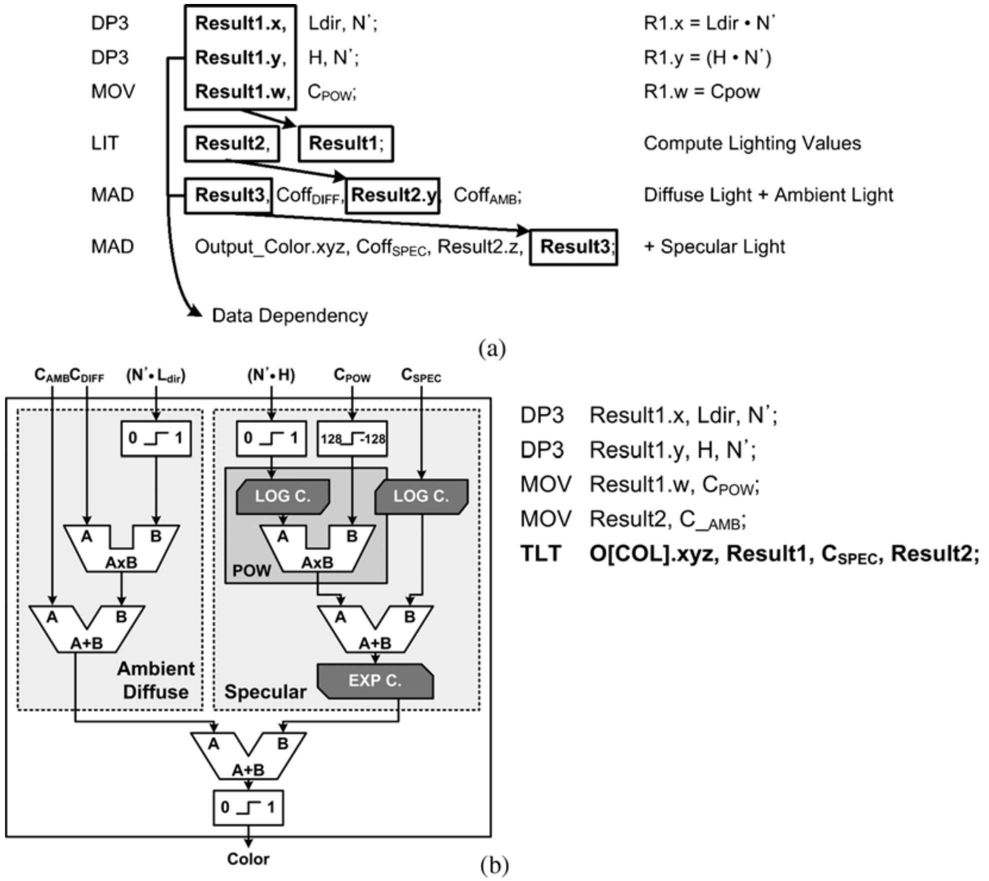
The lighting equation is the most complex operation during the vertex operation due to the power (POW) operation of specular lighting. To accelerate the lighting equation with low power consumption, the low power lighting engine is employed. The OpenGL lighting equation which includes an ambient light, a diffuse light and a specular light is described in (1):

$$\begin{aligned}\text{Color}_{\text{ORD}} = C_{\text{amb}} &+ \{(N' \bullet L_{\text{dir}}) \times C_{\text{diff}}\} \\ &+ \left\{(N' \bullet H)^{C_{\text{pow}}} \times C_{\text{spec}}\right\}. \quad (1)\end{aligned}$$

In conventional implementation [9], the lighting equation is performed sequential fashion as shown in Fig. 7(a). At first, the LIT instruction computes lighting coefficients. Then multiplications and additions are performed. During lighting computation, the data dependency wastes several cycles and it degrades graphics performance. However, as shown in (2), the lighting equation consists of three independent components and it can be computed in parallel:

$$\begin{aligned}\text{Color}_{\text{LNS}} = C_{\text{amb}} &+ \{(N' \bullet L_{\text{dir}}) \times C_{\text{diff}}\} \\ &+ 2^{\left\{\log_2(N' \bullet H) \times C_{\text{pow}} + \log_2 C_{\text{spec}}\right\}}. \quad (2)\end{aligned}$$

To improve lighting computation, the lighting engine and specialized lighting instruction are designed to compute three components simultaneously. Since the POW operation can be converted into a multiplication as shown in (2), the lighting engine employs the logarithmic number datapaths [11] for specular lighting. Therefore, the lighting engine consists of ordinary number datapaths for ambient and diffuser lighting computation and logarithmic number datapaths to accelerate specular lighting computation as shown in Fig. 7(b). To utilize the lighting engine efficiently by graphics APIs, the specialized lighting instruction, TLT, is proposed. The TLT instruction



```
DP3   Result1.x, Ldir, N';                                  R1.x = Ldir • N'
DP3   Result1.y, H, N';                                     R1.y = (H • N')
MOV   Result1.w, CPOW;                                      R1.w = Cpow
LIT   Result2, Result1;                                     Compute Lighting Values
MAD   Result3, CoffDIFF, Result2.y, CoffAMB;                Diffuse Light + Ambient Light
MAD   Output_Color.xyz, CoffSPEC, Result2.z, Result3;       + Specular Light
```

```
DP3  Result1.x, Ldir, N';
DP3  Result1.y, H, N';
MOV  Result1.w, CPOW;
MOV  Result2, C_AMB;
TLT  O[COL].xyz, Result1, CSPEC, Result2;
```

Fig. 7. Low power lighting engine. (a) Conventional lighting computation. (b) Proposed lighting engine.

combines the coefficient calculation and the multiplication of coefficients and materials together as shown in Fig. 7(b). The lighting engine with TLT instruction computes the lighting equation without data dependency so that, the lighting engine with TLT instruction generates lit vertex in every two cycles. For compatibility with OpenGL|ES interface, the lighting engine supports both the conventional LIT instruction and TLT instruction. In the conventional lighting operation, the lighting engine accelerates the POW operations and it computes only the lighting coefficients, which are used for complex 3-D graphics effects such as user-defined lighting. Otherwise, the TLT instruction is used to accelerate the common lighting environments, which is widely used in commercial mobile 3-D contents. By adopting logarithmic LE and TLT instruction, the mobile unified shader generates a lit-vertex in every two cycles and it achieves 9.1 Mverticies/s, which is 2.5 times higher performance compared with previous implementation [3].

D. Micro-Level Power Management

For low power consumption of the mobile unified shader, it implements instruction-level power management. To activate the 3-D graphics processor, the ARM processor must drive activation signal to 3-D graphics processor and the 3-D GPU controller drive enable signal to the unified shader only when the current shader instruction is valid. Using this enable signal, the clock signals of the SIMD register files can be gated off when the write operations of the register files are not required. The read operations of the register files are still possible in the clock-off state. Like the same way, the enable signal also reduces the power dissipation of SIMD arithmetic units by eliminating the unnecessary signal transition. Therefore, the power consumption of the mobile unified shader is controlled on an instruction level. In the mobile unified shader, since the SIMD register files and datapath consumes about 80% of power, about 85% activation ratio in full 3-D graphics pipeline achieves up to 12% power reduction of the mobile unified shader. For power management of the pixel-operations, it implements pixel-level clock gating. The pixel operation includes fragment generator for triangle setup and rasterization, mobile unified shader for per-pixel operations, and pixel generator for blending operations. To reduce power consumption, the fragment generator allows clock gating, which uses depth-comparison results generated in early stage of rendering pipeline. If a new pixel to be drawn is already covered by the pixels near from the viewpoint, the remaining operations does not need to process further. To use this property, per-pixel operations of the mobile unified shader is killed and the clock signal of the pixel generator is gated-off to prevent unnecessary operations.

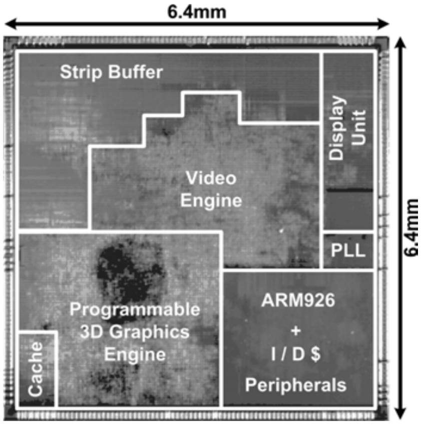


Fig. 8. Die microphoto.

TABLE II
FEATURE SUMMARY

Process Technology		0.13μm 7M CMOS
Power Supply		1.2V
Transistor Counts		18.6M Transistors (including 128KB SRAM)
Die Size		6.4mm x 6.4mm
Operating Frequency		100MHz
Power Consumption		**152mW @ Video Processing 195mW @ Full 3D Graphics Processing**
Package		144pin CABGA
Multimedia Performance	Image Processing	Up to 3M pixels Camera Resolution Support Real-Time Baseline JPEG Encoding / Decoding
	Video Processing	MPEG : CIF 30 fps Codec @ SP Lv.0~3 H.264 : CIF 30 fps Decode @ BL Lv.0~3
	3D - Geometry	**9.1Mvertices/s (Including Full OpenGL Lighting)**
	3D - Rendering	100Mpixels/s, 400Mtexels/s
Graphics Functions	**Programmability**	**Programmable Vertex Shader Programmable Pixel Shader**
	Standard API	OpenGL|ES ver. 1.1 OpenGL|ES ver. 2.0
	Screen Resolution	Up to 1024 x 1024 pixels

Fig. 9. System evaluation board.

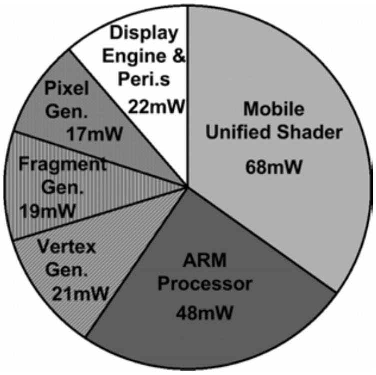


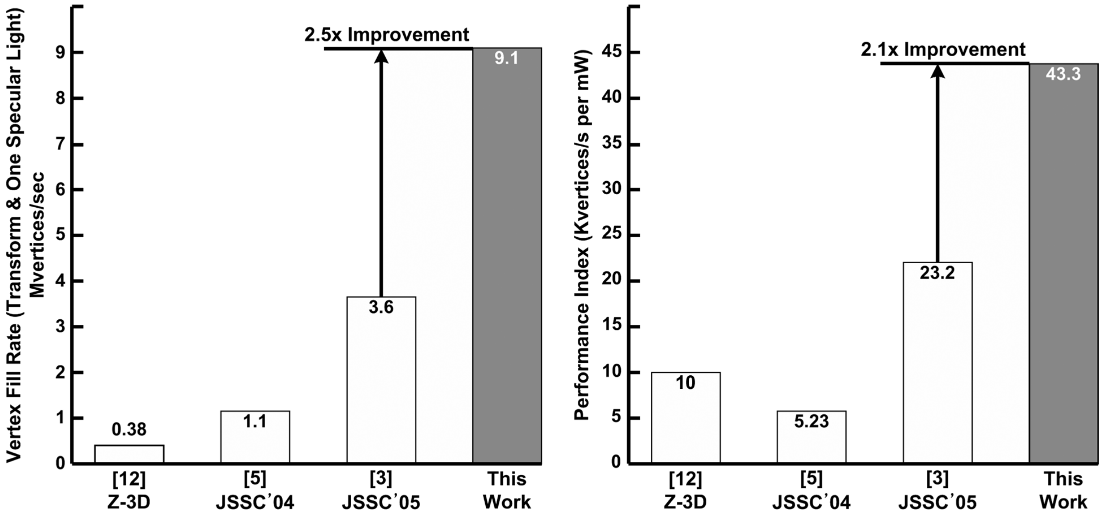
Fig. 10. Power dissipation of the 3-D graphics applications.

V. IMPLEMENTATION RESULTS

The mobile multimedia SoC is fabricated using a 0.13 μm seven-metal CMOS logic process. It contains 18.6 M transistors including 128 KB SRAM in 6.4 × 6.4 mm^2. Fig. 8 shows the chip microphotograph and Table II summarizes chip features. By using this chip, realistic 3-D graphics effects can be processed with 9.1 Mvertices/s peak graphics performance and 30 fps MPEG4 encoding/decoding and 30 fps H.264 decoding can be processed in the mobile devices. The 3-D graphics images with realistic graphics effects are successfully demonstrated by the fabricated chip on the system evaluation board as shown in Fig. 9. For a demo content which uses both the per-vertex operations such as transformation and lighting and the per-pixel operations such as environmental mapping, the developed SoC continuously draws the 3-D images at the speed of 7.4 Mpixels/s.

With the mobile unified shader, the developed SoC can provide fully programmable 3-D graphics pipeline, per-vertex operations and per-pixel operations, and thus, it can completely generate realistic 3-D graphics effects such as environmental mapping or per-pixel lighting. The proposed mobile unified shader reduces silicon area of the 3-D graphics processor by 35% with the help of the unified SIMD datapaths, logarithmic datapath. The logarithmic datapaths, low power register file, and micro-level (instruction-level and pixel-level) clock gating reduces the power consumption by 28%. The developed SoC consumes 195 mW in continuous calculation of 9.1 Mvertices/s full 3-D graphics pipeline including programmable per-vertex operations and programmable per-pixel operations at 100 MHz operating frequency and 1.2 V supply voltage and it consumes 152 mW in continuous calculation of CIF 30 fps H.264 decoding including audio playback at 48 MHz operating frequency and 1.2 V supply voltage. The power dissipations of the functional blocks are summarized in Fig. 10.

Fig. 11 shows the 3-D graphics performance comparison with that of the previous implementations [3], [5], [12]. The 3-D graphics performance is measured including transformation and OpenGL lighting with one ambient light, one diffuse light, and one specular light for geometry and texture blending for rendering. The developed SoC provides 9.1 Mvertices/s fully programmable 3-D graphics including transformation and lighting and, with the help of the lighting engine and specialized lighting instruction, it improves 2.1 times vertex fill rate compared with



	[12] Z-3D	[5] JSSC '04	[3] JSSC '06	**This Work**
Operation Frequency (MHz)	132/33	132/33	200	**100**
Vertex Fill Rate (Mvertices/s)	0.38	1.1	3.6	**9.1**
Power Consumption (mW)	38	210	155	**195**
Process Technology	0.18μm Logic	0.16μm DRAM	0.18μm Logic	**0.13μm Logic**
Programmability	N/A	N/A	Vertex Shading	**Vertex Shading Pixel Shading**

Fig. 11. 3-D graphics performance comparison.

previous implementation [3]. The 3-D graphics performance in the mobile devices cannot be compared directly in terms of processing speed such as vertex fill rate because the power consumption must be taken into account as well [5]. Based on the performance index of the mobile 3-D graphics [3], which is the vertex fill rate normalized by the power consumption, the developed SoC shows 46.67 Kvertices/s/mW, which is about 2 times improvement over the previous implementation [3].

VI. Conclusion

A low power multimedia SoC is designed for mobile devices. It integrates fully programmable 3-D graphics engine, MPEG4/JPEG codec and H.264 decoder in a single chip. The mobile unified shader provides a fully programmable 3-D graphics with 35% area reduction and 28% power reduction. Low power lighting engine and the specialized lighting instruction achieves 9.1 MVertices/s vertex fill rate which is 2.5 times improvement over previous works including transformation and OpenGL lighting. The PVMT improves graphics performance by interleaving the vertex operations into pixel operations and as a result, up to 94% vertex operations are hided into the pixel operations. The SoC consumes less than 195 mW at 1.2 V supply voltage and 100 MHz operating frequency for 3-D graphics and less than 152 mW at 1.2 V supply voltage and 48 MHz operating frequency for video operations. With the help of the mobile unified shader and merged JPEG/MPEG4 codec engine, the SoC reduces silicon area and it is implemented in 6.4 mm × 6.4 mm using 0.13 μm CMOS logic process.

References

[1] J. H. Woo *et al.*, “A 152mW mobile multimedia SoC with fully programmable 3D graphics and MPEG4/H.264/JPEG,” in *IEEE Symp. VLSI Circuits*, 2007, pp. 220–221.

[2] J. H. Sohn *et al.*, “Low power 3D graphics processors for mobile terminals,” *IEEE Commun. Mag.*, vol. 43, no. 12, pp. 90–99, Dec. 2005.

[3] J. H. Sohn *et al.*, “A 155-mW 50-Mvertices/s graphics processor with fixed-point programmable vertex shader for mobile applications,” *IEEE J. Solid-State Circuits*, vol. 41, no. 5, pp. 1081–1091, May 2006.

[4] B. G. Nam *et al.*, “A low-power unified arithmetic unit for programmable handheld 3-D graphics systems,” *IEEE J. Solid-State Circuits*, vol. 42, no. 8, pp. 1767–1778, Aug. 2007.

[5] R. Woo *et al.*, “A 210 mW graphics lsi implementing full 3-D pipeline with 264 Mtexels/s texturing for mobile multimedia applications,” in *IEEE ISSCC Dig. Tech. Papers*, 2003, pp. 44–45.

[6] T. M. Liu *et al.*, “A 125 μW, fully scalable MPEG-2 and H.264/AVC video decoder for mobile applications,” in *IEEE ISSCC Dig. Tech. Papers*, 2006, pp. 402–403.

[7] T. Hashimoto *et al.*, “A 27-MHz/54-MHz 11-mW MPEG-4 video decoder LSI for mobile applications,” *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1574–1581, Nov. 2002.

[8] M. Imai *et al.*, “A 109.5 mW 1.2 V 600 Mtexel/s 3D graphics engine,” in *IEEE ISSCC Dig. Tech. Papers*, 2004, pp. 332–333.

[9] J. Montrym *et al.*, “Geforce4,” in *ACM SIGGRAPH/Eurograph Graphics Hardware Workshop*, 2002.

[10] M. Doggett, “Xenos: XBOX360 GPU,” in *ACM Eurographics*, 2005.

[11] H. Kim *et al.*, “A 231 MHz, 2.18 mW 32 bit logarithmic arithmetic unit for fixed-point 3-D graphics system,” *IEEE J. Solid-State Circuits*, vol. 41, no. 11, pp. 2373–2381, Nov. 2006.

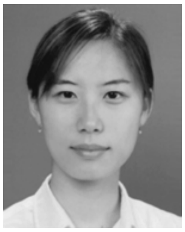
[12] M. Kameyama *et al.*, “3-D graphics LSI core for mobile phone “Z3-D”,” in *Graphics Hardware Workshop*, 2003.

Jeong-Ho Woo (S'02) received the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2002 and 2004, respectively. He is currently working toward the Ph.D. degree in electrical engineering at KAIST.

His research includes design of low-power mobile multimedia system with 3-D computer graphics and its implementation. In particular, he is now working on a low-power programmable 3-D graphics processor for mobile devices.

Ju-Ho Sohn (A'07) received the B.S. (*summa cum laude*), M.S., and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2001, 2003, and 2006, respectively. During the period in KAIST, he developed low-power programmable 3-D graphics processor integrated with vertex shader and DSP engine.

His research interests include low-power high-performance circuits and multimedia system design with specific interest in 3-D computer graphics and video processing architecture. Currently, he is with LG Electronics, Seoul, Korea, designing a digital TV image enhancement chip.



Hyejung Kim (S'04) received the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2004 and 2006, respectively. She is currently working toward the Ph.D. degree in the same department at KAIST.

Her research interests include low-power arithmetic unit and microprocessor design for mobile and implantable system applications.

Hoi-Jun Yoo (M'95–SM'04–F'08) graduated from the Electronic Department of Seoul National University, Seoul, Korea, in 1983 and received the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 1985 and 1988, respectively. His Ph.D. work concerned the fabrication process for GaAs vertical optoelectronic integrated circuits.

From 1988 to 1990, he was with Bell Communications Research, Red Bank, NJ, where he invented the two-dimensional phase-locked VCSEL array, the front-surface-emitting laser, and the high-speed lateral HBT. In 1991, he became Manager of a DRAM design group at Hyundai Electronics and designed a family of fast-1 M DRAMs and 256 M synchronous DRAMs. In 1998 he joined the faculty of the Department of Electrical Engineering at KAIST and now is a full Professor. From 2001 to 2005, he was the Director of the System Integration and IP Authoring Research Center (SIPAC), funded by the Korean government to promote worldwide IP authoring and its SoC application. From 2003 to 2005, he was the full-time Advisor to the Minister of Korea Ministry of Information and Communication and National Project Manager for SoC and Computer. In 2007, he founded SDIA (System Design Innovation and Application Research Center) at KAIST to research and develop SoCs for intelligent robots, wearable computers and biosystems. His current interests are high-speed and low-power networks-on-chips, 3-D graphics, body area networks, biomedical devices and circuits, and memory circuits and systems. He is the author of the books *DRAM Design* (Hongleung, 1996; in Korean), *High Performance DRAM* (Sigma, 1999; in Korean), and chapters of *Networks on Chips* (Morgan Kaufmann, 2006).

Dr. Yoo received the Electronic Industrial Association of Korea Award for his contribution to DRAM technology in 1994, the Hynix Development Award in 1995, the Korea Semiconductor Industry Association Award in 2002, the Best Research of KAIST Award in 2007, the Design Award of 2001 ASP-DAC, and Outstanding Design Awards at 2005, 2006, and 2007 A-SSCC. He is a member of the executive committees of ISSCC, Symposium on VLSI, and A-SSCC. He is the TPC chair of A-SSCC 2008.

Technical Brief

NVIDIA GeForce® GTX 200 GPU Architectural Overview

Second-Generation Unified GPU
Architecture for Visual Computing

ATI 2090
LG v. ATI
IPR2015-00326

Table of Contents

Figures

Tables

Introduction

In this technical brief we introduce NVIDIA's new GeForce® GTX 200 GPU family, the first GPUs to implement NVIDIA's second-generation unified graphics and computing architecture. The high-end, enthusiast-class GeForce GTX 280 GPU and performance-oriented GeForce GTX 260 GPU are the first members of the GeForce GTX 200 GPU family and deliver the ultimate visual computing and extreme high-definition (HD) gaming experience.

We'll begin by describing architectural design goals and key features, and then dive into the technical implementation of the GeForce GTX 200 GPUs. We assume you have a basic understanding of first-generation NVIDIA unified GPU architecture, including unified shader design, scalar processing cores, decoupled texture and math units, and other architectural features. If you are not well versed in NVIDIA unified GPU architecture, we suggest you first read the Technical Brief titled *NVIDIA GeForce 8800 GPU Architecture Overview*. You can also refer to *Appendix A* for a historical retrospective.

GeForce GTX 200 Architectural Design Goals and Key Capabilities

GeForce GTX 200 GPUs are massively multithreaded, many-core, visual computing processors that incorporate both a second-generation unified graphics architecture and an enhanced high-performance, parallel-computing architecture.

Two overarching themes drove GeForce GTX 200 architectural design and are represented by two key phrases: **"Beyond Gaming"** and **"Gaming Beyond."**

Beyond Gaming means the GPU has evolved beyond being used primarily for 3D games and driving standard PC display capabilities. More and more, GPUs are accelerating non-gaming, computationally-intensive applications for both professionals and consumers.

Gaming Beyond means that the GeForce GTX 200 GPUs enable amazing new gaming effects and dynamic realism, delivering much higher levels of scene and character detail, more natural character motion, and very accurate and convincing physics effects.

The GeForce GTX 200 GPUs are designed to be fully compliant with Microsoft DirectX 10 and Open GL 2.1.

Architectural Design Goals

NVIDIA engineers specified the following design goals for the GeForce GTX 200 GPUs:

- ❑ Design a processor with up to twice the performance of GeForce 8800 GTX
- ❑ Rebalance the architecture for future games that use more complex shaders and more memory
- ❑ Improve architectural efficiency per watt and per square millimeter
- ❑ Improve performance for DirectX 10 features such as geometry shading and stream out
- ❑ Provide significantly enhanced computation ability for high-performance CUDA™ applications and GPU physics
- ❑ Deliver improved power management capability, including a substantial reduction in idle power.

GeForce GTX 200 GPUs enable major new graphics and compute capabilities, providing the most realistic 3D graphics effects ever rendered by GPUs to date, while also providing nearly a teraflop of computational power.

Gaming Beyond: Dynamic 3D Realism

While prior-generation GPUs could deliver real-time images that appeared true-to-life in many cases, frame rates could drop to unplayable levels in complex scenes with significant animation, numerous physical effects, and multiple characters. The combination of the sheer shader processing power of GeForce GTX 200 GPUs and NVIDIA's new PhysX™ technology facilitates many new high-end graphics effects including:

- Convincing facial and character animation
- Multiple ultra-high polygon characters in complex environments
- Advanced volumetric effects (smoke, fog, mist, etc.)
- Fluid and cloth simulation
- Fully simulated physical effects such as live debris, explosions, and fires.
- Physical weather effects such as accumulating snow and water, sand storms, soaking, drying, dampening, overheating, and freezing
- Better lighting for dramatic and spectacular effect, including ambient occlusion, global illumination, soft shadows, color bleeding, indirect lighting, and accurate reflections.

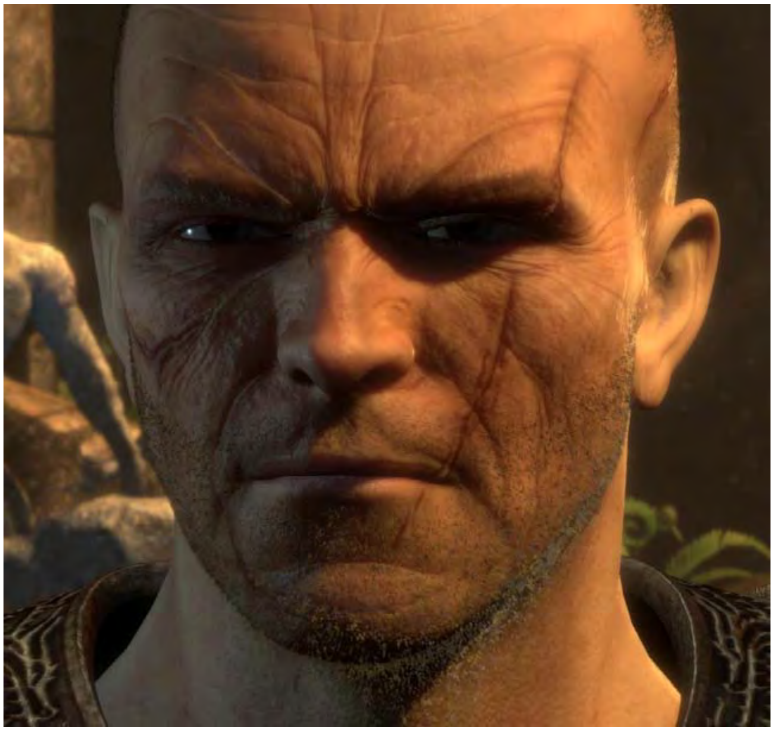


Figure 1: Realistic warrior from NVIDIA "Medusa" demo

Gaming Beyond: Extreme HD

GeForce GTX 200 GPUs provide 50-100% more performance over prior-generation GPUs, permitting increased frame rates and higher visual quality settings at extreme resolutions, resulting in a truly cinematic gaming experience.

Figure 2: Far Cry 2 – Extreme HD Dynamic Beauty! (Ubisoft)

Support for the new DisplayPort interface allows resolutions beyond 2560 × 1600, and 10-bit color support permits up to a billion different colors on screen (driver, display, and application support is also required). Note that prior-generation GPUs included internal 10-bit processing, but could only output 8-bit component colors (RGB). GeForce GTX 200 GPUs permit both 10-bit internal processing and 10-bit color output.

Gaming Beyond: SLI

NVIDIA's SLI® technology is the industry's leading multi-GPU technology, giving you an easy, low-cost, high-impact performance upgrade. PC gaming simply doesn't get any faster or more realistic than running GeForce GTX 200 GPU-based boards in SLI mode on the latest nForce® motherboards.

Two flavors of SLI are supported by the initial GeForce GTX 200 GPUs:

- Standard SLI (two GPU boards), which typically boosts supported game performance by 60-90% and permits higher quality settings
- 3-way SLI, which provides even higher frame rates and permits higher quality settings for the ultimate experience in PC gaming when connected to a high-end, high-resolution monitor.

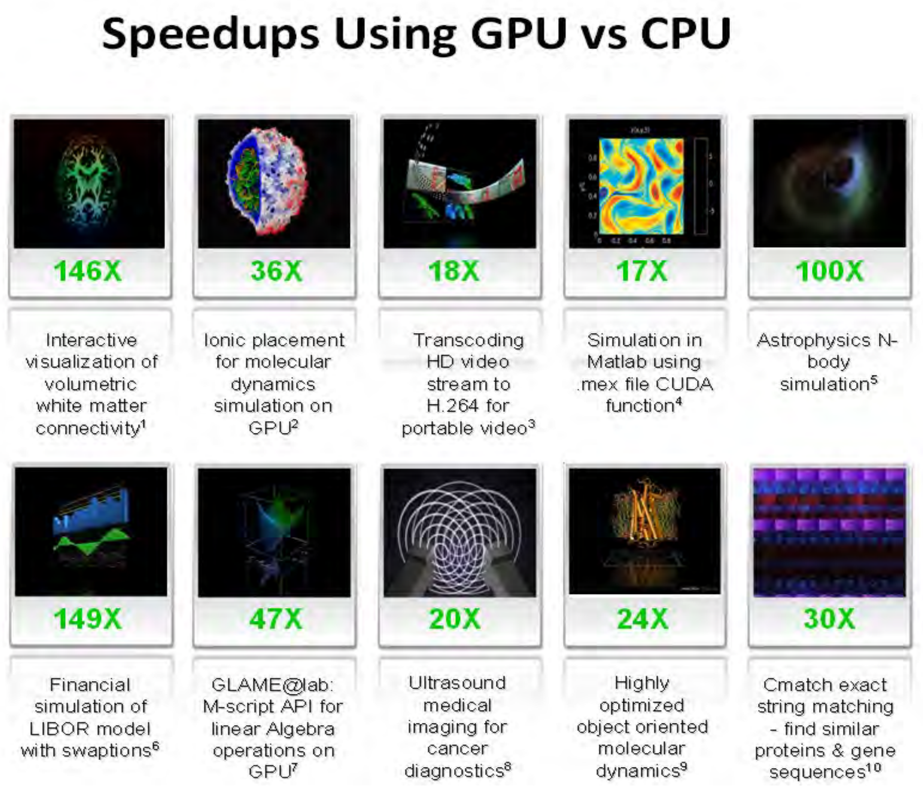
GeForce GTX 200 GPUs process and display complex DirectX 10 and OpenGL game environments with amazing graphics effects and high frame rates at extreme, high-definition resolutions.

Beyond Gaming: High-Performance Visual Computing and Professional Computation

With the power of CUDA technology and the new CUDA runtime for Windows Vista, intensive computational tasks can be offloaded from the CPU to the GPU. GeForce GTX 200 GPUs can accelerate numerous rich-media and computationally-intensive applications such as video and audio transcoding, or running distributed computing applications like Folding@home in the background while surfing the web. Examples of GPU-enabled applications include the RapidHD video transcoding application from Elemental and various video and photo editing applications.

Many engineering, scientific, medical, and financial areas demand high-performance computational horsepower for numerous applications.

Figure 3 shows the amazing speedups that can be achieved by using a GPU instead of a CPU in a number of professional visual computing applications, in addition to mainstream video transcoding. Appendix B lists references and details for these applications.



Speedups Using GPU vs CPU

146X — Interactive visualization of volumetric white matter connectivity[1]

36X — Ionic placement for molecular dynamics simulation on GPU[2]

18X — Transcoding HD video stream to H.264 for portable video[3]

17X — Simulation in Matlab using .mex file CUDA function[4]

100X — Astrophysics N-body simulation[5]

149X — Financial simulation of LIBOR model with swaptions[6]

47X — GLAME@lab: M-script API for linear Algebra operations on GPU[7]

20X — Ultrasound medical imaging for cancer diagnostics[8]

24X — Highly optimized object oriented molecular dynamics[9]

30X — Cmatch exact string matching - find similar proteins & gene sequences[10]

Figure 3: Significant Speedup Using GPU

With an understanding of the GeForce GTX 200 GPU design goals and key objectives, let's delve deeper into its internal architecture, looking at both the graphics and parallel processing capabilities.

GeForce GTX 200 GPU Architecture

GeForce GTX 200 GPUs are the first to implement NVIDIA's second-generation unified shader and compute architecture. The GeForce GTX 200 GPUs include significantly enhanced features and deliver, on average, 1.5× the performance of GeForce 8 or 9 Series GPUs.

Manufactured using TSMC's 65 nm fabrication process, GeForce GTX 200 GPUs include 1.4 billion transistors and are the largest, most powerful, and most complex GPU ever made. All GTX 200 GPUs are built to operate comfortably within the power and heat specifications of high-end PCs.

You may recall that the first-generation NVIDIA unified visual computing architecture in GeForce 8 and 9 Series GPUs was based on a Scalable Processor Array (SPA) framework. The second-generation architecture in GeForce GTX 200 GPUs is based on a reengineered, enhanced, and extended SPA architecture.

The SPA architecture consists of a number of TPCs, which stands for "Texture Processing Clusters" in graphics processing mode, and "Thread Processing Clusters" in parallel compute mode. Each TPC is in turn made up of a number of streaming multiprocessors (SMs), and each SM contains eight processor cores (also called streaming processors (SPs) or thread processors). Every SM also includes texture filtering processors used in graphics processing, but also useful for various filtering operations in compute mode, such as filtering images as they are zoomed in and out.

More Processor Cores

The new second-generation SPA architecture in the GeForce GTX 280 improves performance compared to the prior generation G80 and G92 designs on two levels. First, it increases the number of SMs per TPC from two to three. Second, it increases the maximum number of TPCs per chip from 8 to 10. The effect is multiplicative, resulting in 240 processor cores.

Chip	TPCs	SMs per TPC	SPs per SM	Total SPs
GeForce 8 & 9 Series	**8**	**2**	**8**	**128**
GeForce GTX 200 GPUs	**10**	**3**	**8**	**240**

Table 1: Number of GPU Processing Cores

Based on traditional processing core designs that can perform integer and floating-point math, memory operations, and logic operations, each processing core is a hardware-multithreaded processor with multiple pipeline stages that execute an instruction for each thread every clock.

Various types of threads exist, including pixel, vertex, geometry, and compute. For graphics processing, threads execute a shader program and many related threads often simultaneously execute the same shader program for greater efficiency.

All GeForce GTX 200 GPUs include a substantial portion of die area dedicated to processing, unlike CPUs where a majority of die area is dedicated to onboard cache memory. Rough estimates show 20% of the transistors of a CPU are dedicated to computation, compared to 80% of GPU transistors. GPU processing is centered on computation and throughput, where CPUs focus heavily on reducing latency and keeping their pipelines busy (high cache hit rates and efficient branch prediction).

Graphics Processing Architecture

As mentioned earlier, the GeForce GTX 200 GPUs include two different architectural personalities—graphics and computing. Figure 4 represents the GeForce 280 GTX in graphics mode. You can see the shader thread dispatch logic at the top, in addition to setup and raster units. The ten TPCs each include three SMs, and each SM has 24 processing cores for a total of 240 scalar processing cores. ROP (raster operations processors) and memory interface units are located at the bottom.

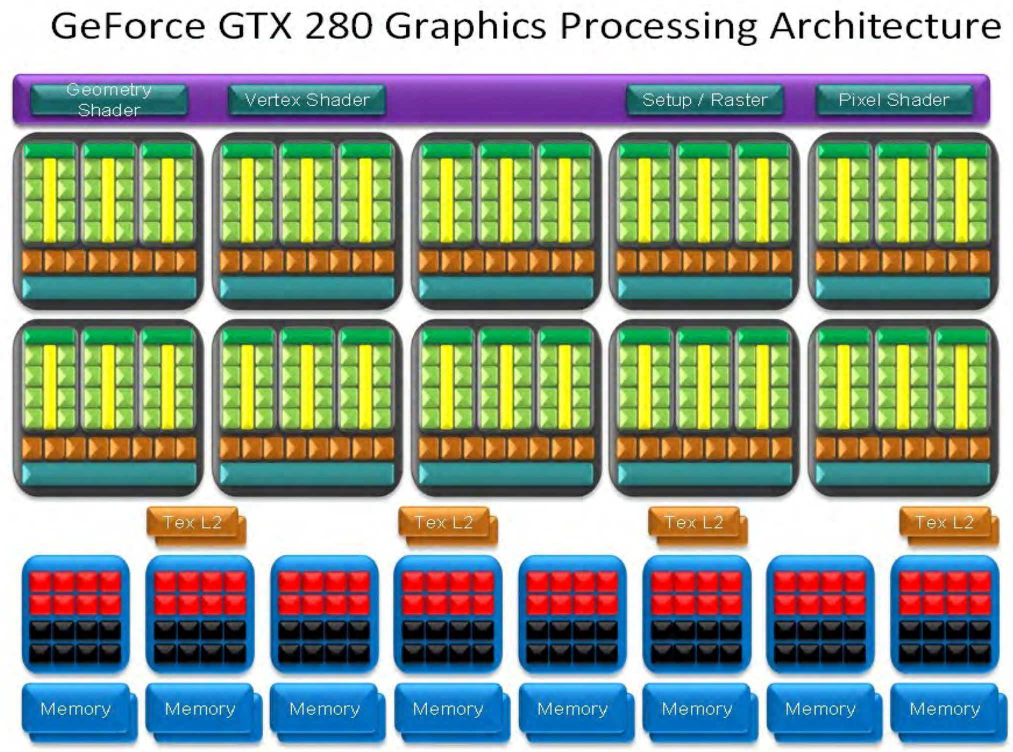


Figure 4: GeForce GTX 280 GPU Graphics Processing Architecture

Although not apparent in the above diagram, the architectural efficiency of the GeForce GTX 200 GPUs is substantially enhanced over the prior generation. We'll be discussing many areas that were improved in more detail, such as texture processing, geometry shading, dual issue, and stream out. In directed tests, GeForce GTX 200 GPUs can attain efficiencies closer to the theoretical performance limits than could prior generations.

Table 2 compares the GeForce 8800 GTX to the new GeForce GTX 280 GPU. You will notice sizable increases in a number of important measurable parameters.

Features	8800 GTX	GTX 280	% Increase
Cores	128	240	87.5 %
TEX	64t/clk	80t/clk	25 %
ROP Blend	12p/clk	32p/clk	167 %
Precision	fp32	fp64	--
GFLOPs	518	933	80 %
FB Bandwidth	86 GB	142 GB	65 %
Texture Fill	37 GT/s	48 GT/s	29.7 %
ROP Blend	7 GBL/s	19 GBL/s	171 %
PCI Express	6.4 GB	12.8 GB	100 %
Video	VP1	VP2	--

Table 2: GeForce 8800 GTX vs GeForce GTX 280

Parallel Computing Architecture

Figure 5 depicts a high-level view of the GeForce GTX 280 GPU parallel computing architecture. A hardware-based thread scheduler at the top manages scheduling threads across the TPCs. You'll also notice the compute mode includes texture caches and memory interface units. The texture caches are used to combine memory accesses for more efficient and higher bandwidth memory read/write operations. The elements indicated as "atomic" refer to the ability to perform atomic read-modify-write operations to memory. Atomic access provides granular access to memory locations and facilitates parallel reductions and parallel data structure management.

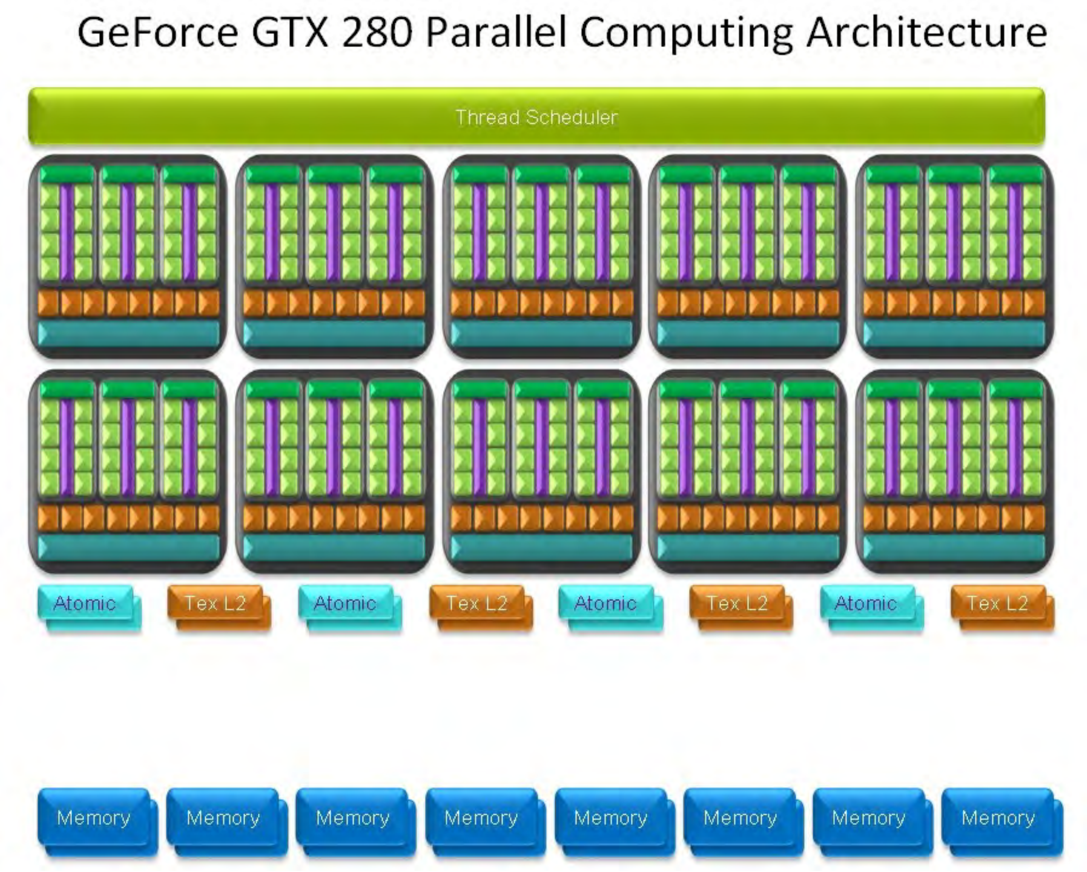


Figure 5: GeForce GTX 280 GPU Parallel Computing Architecture

A TPC in compute mode is represented in Figure 6 below. You can see local shared memory is included in each of the three SMs. Each processing core in an SM can share data with other processing cores in the SM via the shared memory, without having to read or write to or from an external memory subsystem. This contributes greatly to increased computational speed and efficiency for a variety of algorithms.

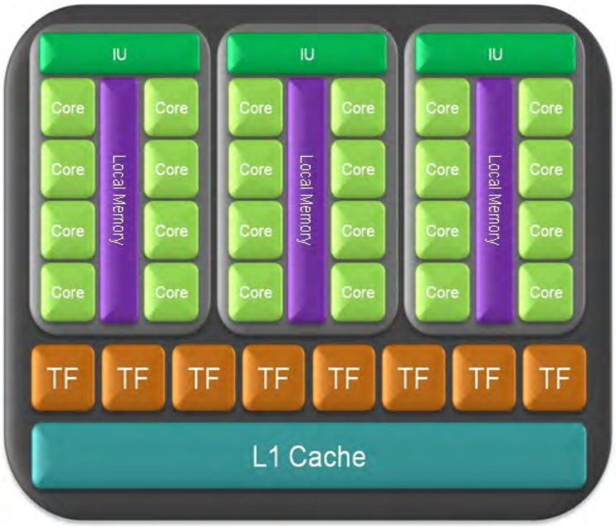


Figure 6: TPC (Thread Processing Cluster)

SIMT Architecture

NVIDIA's unified shading and compute architecture uses two different processing models. For execution across the TPCs, the architecture is MIMD (multiple instruction, multiple data). For execution across each SM, the architecture is SIMT (single instruction, multiple thread).

SIMT improves upon pure SIMD (single instruction, multiple data) designs in both performance and ease of programmability. Being scalar, SIMT has no set vector width and therefore performs at full speed irrespective of vector sizes.

In contrast, SIMD machines operate at a reduced capacity if the input is smaller than the MIMD or SIMD width. SIMT ensures the processing cores are fully utilized at all times.

From the programmer's perspective, SIMT also allows each thread to take on its own path. Since branching is handled by the hardware, there is no need to manually manage branching within the vector width.

Greater Number of Threads in Flight

GeForce GTX 200 GPUs support over thirty thousand threads in flight. Hardware thread scheduling ensures all processing cores attain nearly 100% utilization. The GPU architecture is latency-tolerant—if a particular thread is waiting for a memory access, the GPU can perform zero-cost hardware-based context switching and immediately switch to another thread to process.

The SIMT multithreaded instruction unit within an SM creates, manages, schedules, and executes threads in groups of 32 parallel threads called "warps." Up to 32 warps/SM are supported in GeForce GTX 200 GPUs, versus 24 warps/SM in GeForce 8 or 9 Series GPUs.

Chip	TPCs	SM per TPC	Threads per SM	Total Threads Per Chip
GeForce 8 & 9 Series	8	2	768	12,288
GeForce GTX 200 GPUs	10	3	1,024	30,720

Table 3: Maximum Number of Threads

Doing the math results in 32 x 32, or 1,024 maximum concurrent threads that can be managed by each SM. With 30 SMs in total, the GeForce GTX 280 supports up to 30,720 concurrent threads in hardware (versus 768 threads/SM × 2 SMs/TPC × 8 TPCs = 12,288 maximum concurrent threads in GeForce 8800 GTX).

Larger Register File

The local register file size has doubled per SM in GeForce GTX 200 GPUs compared to GeForce 8 & 9 Series GPUs. The older GPUs could run into situations with long shaders where registers would be exhausted, generating the need to swap to memory. A much larger register file permits larger and more complex shaders to be run on the GeForce GTX 200 GPUs faster and more efficiently. In terms of die size increase, the additional register file takes only a small fraction of SM die area.

Games are employing more and more complex shaders that require more register space. Figure 7 below highlights performance improvements 2× register file size in 3D Mark Vantage.

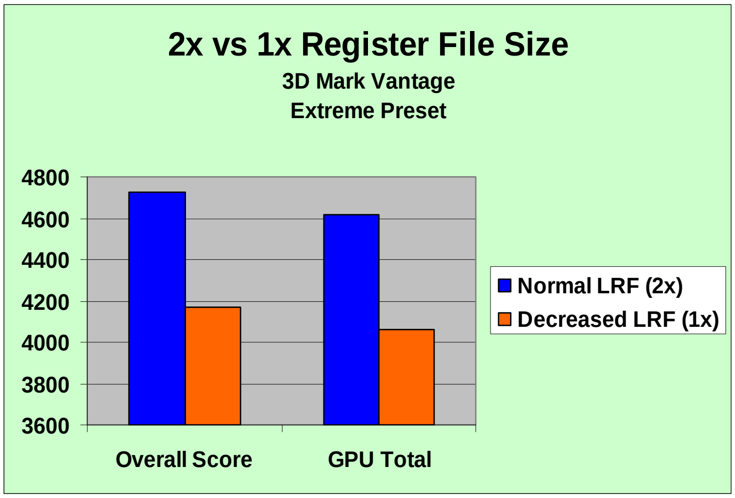


Figure 7: Local Register File 2× versus 1×

Improved Dual Issue

Special function units (SFUs) in the SMs compute transcendental math, attribute interpolation (interpreting pixel attributes from a primitive's vertex attributes), and perform floating-point MUL instructions. The individual streaming processing cores of GeForce GTX 200 GPUs can now perform near full-speed dual-issue of multiply-add operations (MADs) and MULs (3 flops/SP) by using the SP's MAD unit to perform a MUL and ADD per clock, and using the SFU to perform another MUL in the same clock. Optimized and directed tests can measure around 93-94% efficiency.

The entire GeForce GTX 200 GPU SPA delivers nearly one teraflop of peak, single-precision, IEEE 754, floating-point performance.

Double Precision Support

A very important new addition to the GeForce GTX 200 GPU architecture is double-precision, 64-bit floating point computation support. This benefits various high-end scientific, engineering, and financial computing applications or any computational task requiring very high accuracy of results. Each SM incorporates a double-precision 64-bit floating math unit, for a total of 30 double-precision 64-bit processing cores.

The double-precision unit performs a fused MAD, which is a high-precision implementation of a MAD instruction that is also fully IEEE 754R floating-point specification compliant. The overall double-precision performance of all 10 TPCs of a GeForce GTX 280 GPU is roughly equivalent to an eight-core Xeon CPU, yielding up to 78 gigaflops.

Improved Texturing Performance

The eight TPCs of the GeForce 8800 GTX allowed for 64 pixels per clock of texture filtering, 32 pixels per clock of texture addressing, 32 pixels per clock of 2× anisotropic bilinear filtering (8-bit integer), or 32-bilinear-filtered pixels per clock (8-bit integer or 16-bit floating point). Subsequent GeForce 8 and 9 Series GPUs balanced texture addressing and filtering.

- For example, the GeForce 9800 GTX can address and filter 64 pixels per clock, supporting 64-bilinear-filtered pixels per clock (8-bit integer) or 32-bilinear-filtered pixels per clock (16-bit floating point).

GeForce GTX 200 GPUs also provide balanced texture addressing and filtering and each of the 10 TPCs includes a dual-quad texture unit capable of addressing and filtering eight bilinear pixels/clock, or four 2:1 anisotropic filtered pixels/clock, or four FP16 bilinear-filtered pixels/clock. Total bilinear texture addressing and filtering capability for an entire high-end GeForce GTX 200 GPU is 80 pixels per clock.

GeForce GTX 200 GPUs employ a more efficient scheduler, allowing the chips to attain close to theoretical peak performance in texture filtering. In real world measurements, it is 22% more efficient than the GeForce 9 Series.

Chip	Theoretical Bilinear Fillrate	Measured Rate (3DMark multitex)	Measured Performance / Theoretical Performance
GeForce 9 Series	33,600	25,600	76.2%
GeForce GTX 200 GPUs	51,840	48,266	93.1%

Table 4: Theoretical vs Measured Texture Filtering Rates

Higher Shader to Texture Ratio

Because games and other visual applications are continually employing more and more complex shaders, the GeForce GTX 200 GPU design shifts the balance to a higher shader to texture ratio. By adding one more SM to each TPC, and keeping texturing hardware constant, the shader to texture ratio is increased by 50%. This shift allows the GeForce GTX 200 GPUs to perform efficiently for both today's and tomorrow's games.

ROP Improvements

The previous-generation GeForce 8 series ROP subsystem supported multisampled, supersampled, transparency adaptive, and coverage sampling antialiasing. It also supported frame buffer (FB) blending of floating-point (FP16 and FP32) render target surfaces, and either type of FP surface could be used in conjunction with multisampled antialiasing for outstanding HDR rendering quality.

The new GeForce GTX 200 GPU ROP subsystem supports all of the previous generation features, and delivers a maximum of 32 pixels per clock output, equating to 4 pixels/clock per ROP partition × 8 partitions. Up to 32 color and Z samples per clock for 8 × MSAA are supported per ROP partition. Pixels using U8 (8-bit unsigned integer) data format can be blended at twice the rate per TPC of the older-generation GPUs. Given the prior generation GPU had six ROP partitions, it could output 24 pixels/clock and blend 12 pixels/clock. In contrast the GeForce GTX 280 can output and blend 32 pixels/clock.

1 GB Framebuffer

Today's 3D games use a variety of textures to attain realism. Normal maps are used to enhance surface realism, cubemaps for reflections, and high-resolution perspective shadow maps for soft shadows. This means much more memory is needed to render a single scene than classic rendering which relied mainly on the base texture. Deferred rendering engines also make extensive use of multiple render targets, where attributes of the image are rendered off screen before the final image is composed. These techniques consume an immense amount of video memory and memory bandwidth, especially when used in conjunction with antialiasing.

The GeForce GTX 280 and GeForce GTX 260 support 1,024 MB and 896 MB of frame buffer respectively, a two-fold improvement from over prior generation GPUs. With 1 GB of frame buffer, high-resolution antialiasing performance is dramatically improved. For example, deferred rendered games like S.T.A.L.K.E.R. can now be enjoyed with antialiasing.

Geometry Shading and Stream Out

Internal output buffer structures have been significantly upsized by a factor of 6× in GeForce GTX 200 GPUs compared to the prior generation, providing much faster geometry shading and stream out performance. Figure 8 shows the latest RightMark 3D 2.0 benchmark results, including geometry shading tests. The GeForce GTX 280 GPU is significantly faster than prior generation NVIDIA GPUs and competitive products.

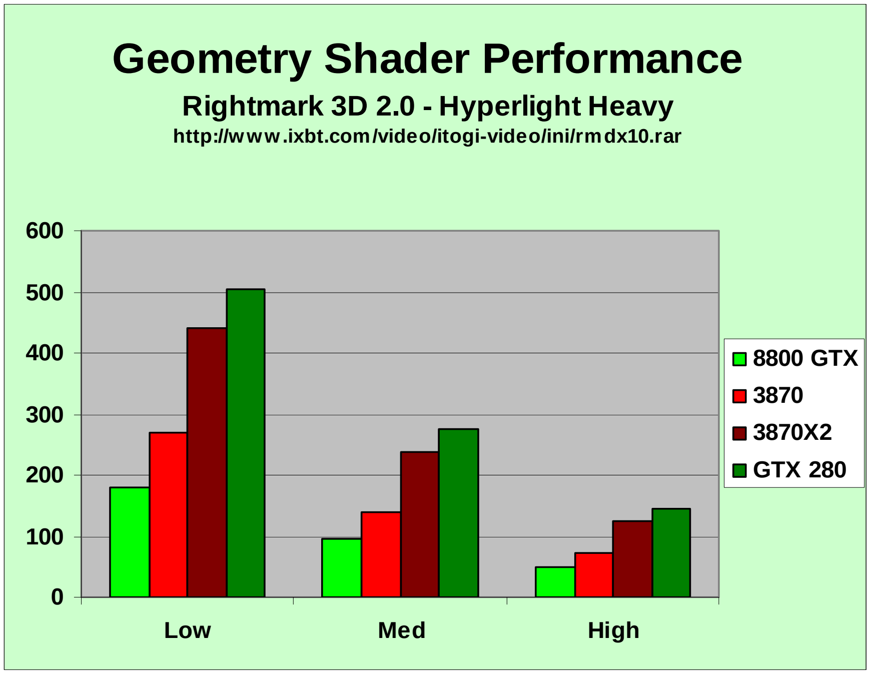


Figure 8: Geometry Shading Performance

Our own Medusa demo is highly dependent on the much faster geometry and stream out performance.

512-bit Memory Interface

Maximum memory interface width is expanded from 384 bits in previous-generation GPUs to 512 bits in GeForce GTX 200 GPUs, using eight 64-bit-wide frame buffer interface units. Memory bandwidth has been significantly increased.

In terms of rebalancing the architecture versus prior generations, the texture to frame buffer (TEX:FB) bandwidth ratio has also been modified to best support current and future workloads. NVIDIA engineers tested many applications to arrive at the right balance of frame buffer bandwidth required to keep the texture units fully utilized and not starved for data.

General frame buffer efficiency has been improved for GeForce GTX 200 GPUs. We reworked the critical paths in the frame buffer to allow higher speed memory operation, up to 1.1 GHz GDDR3 stock speed. Memory bank access patterns and caching algorithms have also been improved. Additional compression hardware in GeForce GTX 200 GPUs effectively increase frame buffer bandwidth by permitting more data to traverse the interface per unit time, enabling better performance at higher resolutions.

Power Management Enhancements

GeForce GTX 200 GPUs include a more dynamic and flexible power management architecture than past generation NVIDIA GPUs. Four different performance / power modes are employed:

- Idle/2D power mode (approx 25 W)
- Blu-ray DVD playback mode (approx 35 W)
- Full 3D performance mode (varies—worst case TDP 236 W)
- HybridPower™ mode (effectively 0 W)

Using a HybridPower-capable nForce motherboard, such as those based on the nForce 780a chipset, a GeForce GTX 200 GPU can be fully powered off when not performing intensive graphics operations and graphics output can be handled by the motherboard GPU (mGPU).

For 3D graphics-intensive applications, the NVIDIA driver can seamlessly switch between the power modes based on utilization of the GPU. Each of the new GeForce GTX 200 GPUs integrates utilization monitors ("digital watchdogs") that constantly check the amount of traffic occurring inside of the GPU. Based on the level of utilization reported by these monitors, the GPU driver can dynamically set the appropriate performance mode (i.e., a defined clock and voltage level) that minimizes the power draw of the graphics card—all fully transparent to the end user.

The GPU also has clock-gating circuitry, which effectively "shuts down" blocks of the GPU which are not being used at a particular time (where time is measured in milliseconds), further reducing power during periods of non-peak GPU utilization.

All this enables GeForce GTX 200 graphics cards to deliver idle power that is nearly 1/10th of its maximum power (approximately 25 W on GeForce GTX 280 GPUs). This dynamic power range gives you incredible power efficiency across a full range of applications (gaming, video playback, surfing the web, etc).

Many other areas of the GeForce GTX 200 GPU pipeline have been reworked to improve performance and reduce various processing bottlenecks.

Additional Pipeline and Architecture Enhancements

Starting from the top of the GeForce GTX 200 GPUs, the front-end unit communicates with the graphics driver running on the host system to accept commands and data. The communication protocol and certain software classes have been modified to improve efficiency of data transfer between the driver and the front end.

The memory crossbar between the data assembler and the frame buffer units has been optimized, allowing the GeForce GTX 200 GPUs to run at full speed when performing indexed primitive fetches (unlike the prior generation which suffered some contention between the front end and data assembler).

The post-transform cache size has been increased, resulting in fewer pipeline stalls and faster communication from the geometry and vertex stages to the viewport clip/cull stage. (Setup rates are similar to prior generation, supporting up to one primitive per clock).

Z-Culling performance has also been improved, especially at high resolutions. Early-Z rejection rates have been increased because the number of ZROPs was increased. The maximum ZROP cull rate is 256 samples/clock or 32 pixels/clock.

GeForce GTX 200 GPUs also include significant micro-architectural improvements in register allocation, instruction scheduling, and instruction issue. The GPUs can now feed the execution units more swiftly. These improvements are responsible for the ability to dual-issue instructions to SPs and SFUs as previously discussed. Scheduling of work between texture units and the SM controller has also been improved.

Summary

NVIDIA's second generation unified visual computing architecture as embodied in the new GeForce GTX 200 GPUs is a significant evolution over the original unified architecture of GeForce 8 and 9 series GPUs. Numerous extensions and functional enhancements to the architecture permit a performance increase averaging 1.5× the prior architecture. Improvements in sheer processing power combined with improved architectural efficiency allow amazing speedups in gaming, visual computing, and high-end computation.

Compared to earlier GPUs such as GeForce 8800 GTX, the GeForce GTX 280 provides:

- ❑ 1.88× more processing cores
- ❑ 2.5× more threads per chip
- ❑ Doubled register file size
- ❑ Double-precision floating-point support
- ❑ Much faster geometry shading
- ❑ 1 GB frame buffer with 512-bit memory interface
- ❑ More efficient instruction scheduling and instruction issue
- ❑ Higher clocked and more efficient frame buffer memory access
- ❑ Improvements in on-chip communications between various units
- ❑ Improved Z-cull and compression supporting higher performance at high resolutions, and
- ❑ 10-bit color support

These all result in enough graphics and compute power to deliver the most intensive and extreme 3D gaming experiences and teraflop performance for demanding high-end compute-intensive applications.

NVIDIA SLI technology is taken to new levels with GeForce GTX 200 GPUs and NVIDIA PhysX technology will add amazing new graphical effects to upcoming game titles. CUDA applications will benefit from additional cores, far more threads, double-precision math, and increased register file size.

Wise users purchasing new systems will conduct performance analyses to optimize their PC architecture. They will find that a lower-end CPU paired with a higher-end GPU produces more performance than the reverse and for the same price. This heterogeneous computing using the right processors for the right tasks and designing optimized PCs to take advantage of it is the wave of the future.

Appendix A: Retrospective

Over the past decade, NVIDIA's graphics processing units (GPUs) have evolved from specialized, fixed-function 3D graphics processors to highly programmable, massively multithreaded, parallel-processing architectures used for visual computing and high-performance computation.

NVIDIA GeForce GPUs enable incredibly realistic 3D gaming and outstanding high-definition video playback, while NVIDIA Quadro® GPUs provide the highest quality and fastest workstation graphics for professional design and creation. For high-performance computing tasks in various engineering, scientific, medical, and financial fields, NVIDIA's new Tesla™ GPUs and CUDA parallel programming environment enable supercomputing-level performance on the desktop, at a fraction of the cost of comparably performing CPU-based multiprocessor clusters.

The GeForce 8800 GPU was launched in November 2006. It was the world's first DirectX 10 GPU with a unified shader architecture. This was important as each of the unified shader processing cores could be dynamically allocated to vertex, pixel, and geometry workloads, making it far more efficient than prior-generation GPUs, which used a fixed number of pixel processing units and a fixed number of vertex processing units. This same unified architecture provided the framework for efficient high-end computation using NVIDIA CUDA software technology.

The GeForce 9 Series GPUs were introduced in 2007, offering a vastly improved price-performance ratio and advanced PureVideo® features. Its smaller chip allowed dual-GPU GeForce 9800 GX2 graphics boards to be built more efficiently, while offering up to twice the performance of the GeForce 8800 GTX.

As of May 2008, over 70 million NVIDIA GeForce 8 and 9 Series GPUs have shipped and each supports CUDA technology, allowing greatly accelerated performance for mainstream visual computing applications like audio and video encoding and transcoding, image processing, and photo editing. These GPUs also support the new NVIDIA PhysX technology for enabling real-time physics in games.

GPUs are the most important and most powerful processors in the new era of visual computing. High-end GeForce GTX 200 GPUs provide the best user experience when running intensive DirectX 10-based games like *Crysis* at high quality and high resolution settings. Very capable motherboard and mid-range GPUs are also needed for stutter-free, high-definition video playback on the PC while simultaneously displaying the Aero 3D user interface of Windows Vista.

Appendix B: Figure 3 References

1. "Interactive Visualization of Volumetric White Matter Connectivity in DT-MRI Using a Parallel-Hardware Hamilton-Jacobi Solver," by Won-Ki Jeong, P. Thomas Fletcher, Ran Tao, and Ross T. Whitaker

2. "GPU Acceleration of Molecular Modeling Applications."

3. Video encoding uses iTunes on the CPU and Elemental on the GPU running under Windows XP. CPUs tested were Intel Core 2 Duo 1.66 GHz and Intel Core 2 Quad Extreme 3 GHz. GPUs tested were GeForce 8800M on the Gateway P-Series FX notebook and GeForce 8800 GTS 512 MB. CPUs and GeForce 8800 GTS 512 were run on Asus P5K-V motherboard (Intel G33 based) with 2 GB DDR2 system memory. Based on an extrapolation of 1 min 50 sec 1280 × 720 high-definition movie clip.

4. http://developer.nvidia.com/object/matlab_cuda.html

5. "High performance direct gravitational N-body simulations on graphics processing units paper," communicated by E.P.J. van den Heuvel

6. "LIBOR," by Mike Giles and Su Xiaoke.

7. "FLAG@lab: An M-script API for Linear Algebra Operations on Graphics Processors."

8. http://www.techniscanmedicalsystems.com/

9. "General Purpose Molecular Dynamics Simulations Fully Implemented on Graphics Processing Units," by Joshua A. Anderson, Chris D. Lorenz, and A. Travesset

10. "Fast Exact String Matching On the GPU," presentation by Michael C. Schatz and Cole Trapnell

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA, the NVIDIA logo, GeForce, Quadro, Tesla, CUDA, PhysX, nForce, PureVideo, and SLI are trademarks or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated

Copyright

Intel® Processor Graphics DirectX* Developer's Guide

How to maximize graphics performance on Intel® microarchitecture codename Sandy Bridge

Document Number: 321371-002

Revision: 2.9.6

Contributors: Jeff Freeman, Chris McVay, Chuck DeSylva, Luis Gimenez, Katen Shah, Jeff Frizzell, Ben Sluis, Anthony Bernecky, Raghu Muthyalampalli, Ganeshkumar Doraisamy, Steven Smith, Axel Mamode

World Wide Web: http://www.intel.com

Document Number: 321671-005US

ATI 2091
LG v. ATI
IPR2015-00326

Disclaimer and Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Software Source Code Disclaimer

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license:

Intel Sample Source Code License Agreement

This license governs use of the accompanying software. By installing or copying all or any part of the software components in this package, you ("you" or "Licensee") agree to the terms of this agreement. Do not install or copy the software until you have carefully read and agreed to the following terms and conditions. If you do not agree to the terms of this agreement, promptly return the software to Intel Corporation ("Intel").

1. Definitions:

   A. "Materials" are defined as the software (including the Redistributables and Sample Source as defined herein), documentation, and other materials, including any updates and upgrade thereto, that are provided to you under this Agreement.

   B. "Redistributables" are the files listed in the "redist.txt" file that is included in the Materials or are otherwise clearly identified as redistributable files by Intel.

   C. "Sample Source" is the source code file(s) that: (i) demonstrate(s) certain functions for particular purposes; (ii) are identified as sample source code; and (iii) are provided hereunder in source code form.

   D. "Intel's Licensed Patent Claims" means those claims of Intel's patents that (a) are infringed by the Sample Source or Redistributables, alone and not in combination, in their unmodified form, as furnished by Intel to Licensee and (b) Intel has the right to license.

2. License Grant: Subject to all of the terms and conditions of this Agreement:

   A. Intel grants to you a non-exclusive, non-assignable, copyright license to use the Material for your internal development purposes only.

   B. Intel grants to you a non-exclusive, non-assignable copyright license to reproduce the Sample Source, prepare derivative works of the Sample Source and distribute the Sample Source or any derivative works thereof that you create, as part of the product or application you develop using the Materials.

   C. Intel grants to you a non-exclusive, non-assignable copyright license to distribute the Redistributables, or any portions thereof, as part of the product or application you develop using the Materials.

   D. Intel grants Licensee a non-transferable, non-exclusive, worldwide, non-sublicenseable license under Intel's Licensed Patent Claims to make, use, sell, and import the Sample Source and the Redistributables.

3. Conditions and Limitations:

   A. This license does not grant you any rights to use Intel's name, logo or trademarks.

   B. Title to the Materials and all copies thereof remain with Intel. The Materials are copyrighted and are protected by United States copyright laws. You will not remove any copyright notice from the Materials. You agree to prevent any unauthorized copying of the Materials. Except as expressly provided herein, Intel does not grant any express or implied right to you under Intel patents, copyrights, trademarks, or trade secret information.

C. You may NOT: (i) use or copy the Materials except as provided in this Agreement; (ii) rent or lease the Materials to any third party; (iii) assign this Agreement or transfer the Materials without the express written consent of Intel; (iv) modify, adapt, or translate the Materials in whole or in part except as provided in this Agreement; (v) reverse engineer, decompile, or disassemble the Materials not provided to you in source code form; or (vii) distribute, sublicense or transfer the source code form of any components of the Materials and derivatives thereof to any third party except as provided in this Agreement.

4. No Warranty:

THE MATERIALS ARE PROVIDED "AS IS". INTEL DISCLAIMS ALL EXPRESS OR IMPLIED WARRANTIES WITH RESPECT TO THEM, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR ANY PARTICULAR PURPOSE.

5. Limitation of Liability: NEITHER INTEL NOR ITS SUPPLIERS SHALL BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF INTEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

6. USER SUBMISSIONS: You agree that any material, information or other communication, including all data, images, sounds, text, and other things embodied therein, you transmit or post to an Intel website or provide to Intel under this Agreement will be considered non-confidential ("Communications"). Intel will have no confidentiality obligations with respect to the Communications. You agree that Intel and its designees will be free to copy, modify, create derivative works, publicly display, disclose, distribute, license and sublicense through multiple tiers of distribution and licensees, incorporate and otherwise use the Communications, including derivative works thereto, for any and all commercial or non-commercial purposes

7. TERMINATION OF THIS LICENSE: This Agreement becomes effective on the date you accept this Agreement and will continue until terminated as provided for in this Agreement. Intel may terminate this license at any time if you are in breach of any of its terms and conditions. Upon termination, you will immediately return to Intel or destroy the Materials and all copies thereof.

8. U.S. GOVERNMENT RESTRICTED RIGHTS: The Materials are provided with "RESTRICTED RIGHTS". Use, duplication or disclosure by the Government is subject to restrictions set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor. Use of the Materials by the Government constitutes acknowledgment of Intel's rights in them.

9. APPLICABLE LAWS: Any claim arising under or relating to this Agreement shall be governed by the internal substantive laws of the State of Delaware, without regard to principles of conflict of laws. You may not export the Materials in violation of applicable export laws.

Revision History

Document Number	Revision Number	Description	Revision Date
321671-001US	1.0	Re-drafted for Intel® 4-Series Chipsets.	Sept 2008
321671-001US	1.1	Re-drafted for Intel® 4-Series Chipsets.	Sept 2008
321671-002US	2.6.6	Intel HD Graphics DirectX* Developer's Guide (Sandy Bridge).	March 2009
321671-003US	2.6.7	Intel® HD Graphics DirectX* Developer's Guide (Sandy Bridge).	April 2009
321671-004US	2.7.1	Intel® HD Graphics DirectX* Developer's Guide (Sandy Bridge) featuring Intel® HD Graphics	Feb 2010
321671-005US	2.8.0	Intel® HD Graphics DirectX* Developer’s Guide for the Intel® microarchitecture codename Sandy Bridge.	August 2010
321671-006US	2.9.5	Incremental changes and additions related to Intel microarchitecture codename Sandy Bridge	December 2010
	2.9.6	Incremental changes prior to GDC release	February 2011

Contents

§

1 About this Document

This document provides development hints and tips to ensure that your customers will have a great experience playing your games and running other interactive 3D graphics applications on Intel® Processor Graphics. This document details software development practices using the latest generation of Intel processor graphics: Intel® Processor Graphics as well as two previous generations of the Intel® Graphics Media Accelerator with a focus on performance analysis using Microsoft DirectX*. Intel tools useful in optimizing graphics applications are introduced in a section detailing performance analysis with the Intel® Graphics Performance Analyzers (Intel® GPA).

Intel® Graphics is split into product generations. The latest one was introduced in 2011 with Intel® microarchitecture codename Sandy Bridge. This family of processors is now on the same silicon as the CPU and it is now called *Intel® Processor Graphics*. In addition to vastly improved performance, Intel Processor Graphics also adds significant new features and functionalities over the previous generation of Intel Graphics called Intel® HD Graphics. Intel Processor Graphics currently represents the most common graphics solution chosen by new PC purchasers. This document is meant to help you include this broad market as a target for your applications and optimize the experience for widest people. By following the tips and tricks in this document, you have the opportunity to make your application shine with the graphics volume market leader.

1.1 Intended Audience

This document is targeted at experienced graphics developers who are familiar with OpenGL*/Microsoft DirectX*, C/C++, multithreading and shader programming, Microsoft Windows* operating systems, and 3D graphics.

1.2 Conventions, Symbols, and Terms

Table 1 Coding Style used in the Document

Source code:

```
for( int i = 0; i < 10; ++i )
{
   cout << i << endl;
}
```

The following terms are used in this document.

Table 2 Terms Used in this Document

1. **HDG –**Intel® HD Graphics, the generation of graphics from Intel which was characterized by the graphics subsystem being on a separate die from the CPU
2. **Processor Graphics** - The latest generation of graphics from Intel® included in the same processor die of the Intel® microarchitecture codename Sandy Bridge family of processors
3. **UMA** – Unified Memory Architecture - an architecture where the graphics subsystem does not have exclusive dedicated memory and uses the host system's memory (SDRAM)
4. **DVMT** – Dynamic Video Memory Technology – a memory allocation scheme in UMA systems which allocates an exclusive, dynamically resizable chunk of main memory to the graphics (driver)
5. **VF** – Vertex Fetch
6. **VS** – Vertex Shader
7. **PS** – Pixel Shader
8. **GS** – Geometry Shader
9. **EU** – Execution Unit, a vector machine component
10. **I$** - Instruction cache
11. **SO** – Stream Output
12. **HWVP** – Hardware vertex processing - see the following Intel document for more information on this: http://www.**intel**.com/assets/pdf/whitepaper/318888.pdf

1.3 Further Help Beyond this Guide

There are several other places you can look for additional information on Intel Processor Graphics, including the following sites:

Intel® HD Graphics (previous generation): http://software.intel.com/en-us/articles/intel-graphics-developers-guides/

Intel® 4 Series Chipsets (the Intel® 4500, X4500, and X4500HD GMAs) Developer's Guide: http://software.intel.com/en-us/articles/intel-graphics-media-accelerator-developers-guide/

Intel® 3 Series Express Chipsets including the Intel® 3000 GMA and Intel® X3000 GMA Developer's Guide: http://software.intel.com/en-us/articles/intel-gma-3000-and-x3000-developers-guide/.

We hope your questions are covered in these resources, including this Guide. We are constantly updating these resources and welcome your comments and suggestions. If you have questions not answered in these resources, or have suggestions on improving the Guide, please get in touch with us at: GameDevInput@intel.com. If you are actively working with Intel already, you can also reach us through your engineering or account management contacts.

2 Intel® HD Graphics to Intel® Processor Graphics

2.1 Intel® HD Graphics is Becoming Core

Several versions of the Intel® Core™ i3, Core™ i5, and Core™ i7 processors have launched in 2011 using Intel® microarchitecture codename Sandy Bridge. These represent the first instantiation of complete platform integration, with Intel® Processor Graphics co-residing on the CPU die.

Two key versions of graphics will be available, Intel® Processor Graphics 2000 and Intel® Processor Graphics 3000, with Intel® Processor Graphics 2000 targeting lower voltage (lower power) applications and Intel® Processor Graphics 3000 a more mainstream set of applications.

Table 3 Evolution of Intel® Processor Graphics

2009	2010	2011	
GMA Series 4 Chipset	Intel codename Ironlake - 1st Gen CPU Integration	Intel® microarchitecture codename Sandy Bridge - 1st Gen CPU/Graphics single die	
DirectX* 10 SM4, OpenGL* 2.0	DirectX* 10, SM4, OpenGL* 2.1	DirectX* 10.1 SM4.1 , OpenGL* 3.0, DirectX* 11 API on DirectX* 10 hardware	
Mobile / Desktop*: 21 / 32 GFLOPs	Mobile / Desktop: 37 / 43 GFLOPs	Mobile / Desktop: ~105-125 GFLOPs for Processor Graphics 3000 & ~55-65 GFLOPS for Processor Graphics 2000	
Mobile/Desktop 400 - 533 MHz/ 800 MHz	Mobile/Desktop 500-766 MHz/ 533 - 900 MHz	Mobile/Desktop Base: 350-650 MHz/650-850 MHz Turbo: 900-1250 MHz/1100-1350 MHz	
	1.7x 3D performance increase	Intel® Processor Graphics 2000: 6 Execution Units ≥1x with lower voltage requirements.	Intel® Processor Graphics 3000: 12 Execution Units 1.5-2.5x performance increase

* Single precision peak values with MAD instructions.

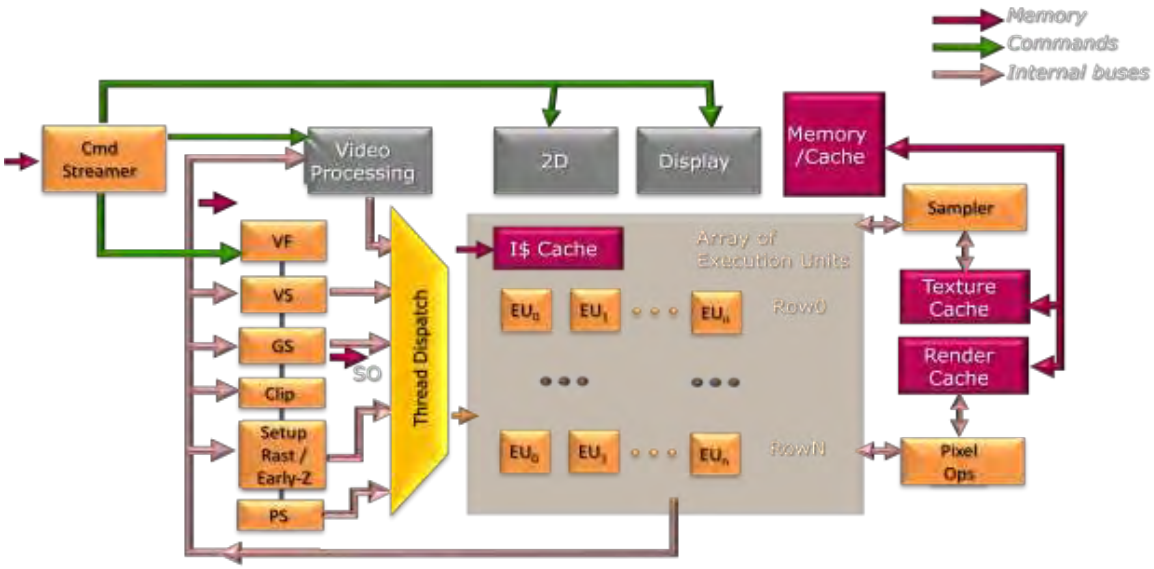


Figure 1 Intel® Processor Graphics Architecture Diagram

Intel® Processor Graphics has been architected to support Microsoft DirectX* 10.1 and take advantage of a generalized unified shader model including support for Shader Model 4.1 and lower. The platform also has support for DirectX* 11 on DirectX* 10 hardware. The graphics core executes vertex, geometry, and pixel shaders on the programmable array of Execution Units (EUs). The EUs have programmable SIMD (Single Instruction, Multiple Data) widths of 4 and 8 element vectors (two 4 element vectors paired) for geometry processing and 8 and 16 single data element vectors for pixel processing). Each EU is capable of executing multiple threads to cover latency. The new generation of Intel® Processor Graphics now integrates transcendental shader instructions into the EU units, rather than a shared math box found in prior generations, resulting in improved processing of instructions such as POW, COS, SIN. Clipping and setup have moved to Fixed Function units, further increasing performance by reducing contention within the EU's. The end result is the fastest Intel® Processor Graphics to date.

2.2 What's New in Intel® Processor Graphics

2.2.1.1 Graphics Features

The latest version of Intel® Processor Graphics includes several performance changes since the previous generation:

Table 4 Intel® Processor Graphics Feature Specifications

3D Pipeline	Key Improvements in Intel® Processor Graphics
Geometry Processing	• Improved throughput up to 2x better than previous generations • Sharing of the last level cache with the CPU • Increased number of threads for vertex shading • Faster clip, cull, and setup • Improved throughput of geometry shader and stream out • OpenGL* driver now uses hardware geometry processing
Rasterization and Z	• Improved hierarchical Z performance • Improved clear performance • Added OpenGL* MSAA 4X support • Added 2X and 4X MSAA support under DirectX* 9 and DirectX* 10
Computes	• >3X increase in transcendental computations • Overall arithmetic performance improvement in shaders due to math box integration within EU's
Texture and Pixel Processing	• Added support for Gather4, Target Independent Blend Modes, Per Sample Shader Frequency, TextureCubeArray, VS/GS 32 Registers, per the DirectX* 10.1 specification • Improved fill rate for short shaders due to fixed function setup management of barycentric coefficients

2.2.1.2 Intel® Turbo Boost Technology

Intel® Processor Graphics utilizes a dynamic frequency on mobile and desktop graphics to automatically increase the clock frequencies of the CPU cores and the graphics cores to boost performance when the workload demands it and also to scale back the frequencies when demand decreases. Intel® microarchitecture codename Sandy Bridge supports higher performance boosts after extended CPU idle periods. In addition to the Intel® Turbo Boost Technology on the CPU, a similar technology has been extended to graphics on both the mobile and desktop platforms. This allows the graphics subsystem to run at higher frequencies when the CPU is not using its nominal *thermal design power* (TDP). In combination, these technologies dynamically manage the CPU and graphics core performances based on workload demands, to allow for better performance when needed and minimize power usage when possible.

2.3 Intel® Processor Graphics Specifications

Table 5 Intel® Processor Graphics Feature Specifications

<table>
<tr><th>Intel Graphics Core</th><th colspan="2">Intel® Processor Graphics</th></tr>
<tr><td><b>Intel Chipset</b>
(<b>see</b> Error! Reference source not found.<b>)</b></td><td colspan="2"><b>Intel® microarchitecture codename Sandy Bridge</b></td></tr>
<tr><td><b>Graphics Architecture</b></td><td><b>Intel® Processor Graphics 2000</b></td><td><b>Intel® Processor Graphics 3000</b></td></tr>
<tr><td><b>Segment</b></td><td><b>Desktop/Mobile</b></td><td><b>Desktop/Mobile</b></td></tr>
<tr><td><b>Maximum Video Memory</b></td><td colspan="2">Depends on system memory and operating system.
Windows Vista*/Windows* 7: refer to the table below:<table><tr><th>System Memory</th><th>1GB</th><th>2GB</th><th>>2GB</th></tr><tr><td><b>Max Available Video Memory</b></td><td>256 MB</td><td>783 MB</td><td>1692 MB</td></tr></table></td></tr>
<tr><td><b>DirectX* Support</b></td><td colspan="2">10.1, DirectX* 11 on DirectX* 10 hardware,
Compute Shader 4.x,
DirectX* 11 API multi-threaded rendering on DirectX* 10 hardware (asynchronous object creation supported, software support for asynchronous command list in the runtime)</td></tr>
<tr><td><b>OpenGL* Support</b></td><td colspan="2">3.0</td></tr>
<tr><td><b>Shader Model Support</b></td><td colspan="2">4.1</td></tr>
</table>

3 Quick Tips: Graphics Performance Tuning

3.1.1 Optimizing for the vertex cache

Use `IDirect3DDevice9::DrawIndexedPrimitive` (DirectX* 9) or `ID3D10Device::DrawIndexed` (DirectX* 10) to maximize reuse of the vertex cache.

Pre- and post-transform vertex cache sizes can vary significantly, even across different generations of Intel Graphics platforms. To maximize performance of vertex processing, it can help to optimize the ordering of vertices and triangle indices in your vertex and index buffers.

There are two methods you can use to optimize your data for the vertex cache:

The preferred method is to use the `D3DXOptimizeFaces` and `D3DXOptimizeVertices` APIs. These apply a good generalized optimization that will work well for all cache sizes (and across all hardware). Since this optimization works well across all hardware, it can be applied at authoring time (e.g. when exporting mesh data from your content creation pipeline), cutting down level load times.

The code snippet below demonstrates how to use these APIs on mesh data:

```
void OptimizeMesh( WORD * Indices, // [In/Out] - Index buffer data
                   DWORD NumFaces, // Number of faces in the mesh
                   Vertex * Vertices, // [In/Out] - Vertex buffer data
                   DWORD NumVertices ) // Number of vertices in the mesh
{
    HRESULT hr;

    // Create a "re-map" buffer for the new face ordering, and
    // calculate the new order.
    DWORD *Remap = new DWORD[ NumFaces ];
    hr = D3DXOptimizeFaces( IndicesIn, NumFaces, NumVertices,
                            FALSE, Remap );

    // Make a copy of the old indices, which we'll pull from for the new
    // re-ordered list of indices.
    WORD *OldIndices = new WORD[ NumFaces * 3 ];
    memcpy( OldIndices, IndicesIn, sizeof(WORD) * NumFaces * 3 );
    WORD * NewFaces = Indices;

    // Apply the new mapping.
    for( DWORD i = 0; i < NumFaces; ++i )
    {
        WORD *OldFaceBase = OldIndices + ( Remap[ i ] * 3 );
        NewFaces[0] = OldFaceBase[0];
        NewFaces[1] = OldFaceBase[1];
        NewFaces[2] = OldFaceBase[2];

        NewFaces += 3;
    }

    delete[] Remap;
    delete[] OldIndices;

    // Create a "re-map" buffer for the new vertex ordering, and
    // calculate the new order.
    DWORD ActualVertexCount = 0;
    Remap = new DWORD[ NumVertices ];
    hr = D3DXOptimizeVertices( Indices, NumFaces, NumVertices,
                 FALSE, Remap );

    // Count how many vertices we actually have. Remap indices of
    // 0xffffffff indicate a vertex that was not referenced by any faces.

    DWORD dwVertexCount = 0;
    for( DWORD i = 0; i < NumVertices; ++i )
    {
        if( aRemap[ i ] == 0xffffffff )
            break;

        ++ActualVertexCount;
    }

        // Copy the vertex data into a temp buffer.
    Vertex *OldVertices = new Vertex[ NumVertices ];
    memcpy( OldVertices, Vertices, sizeof(Vertex) * NumVertices );

    // Perform the remapping
    const DWORD *CurrentRemap = Remap;
    const SVertex *OldVertex = OldVertices;
    for( DWORD i = 0; i < ActualVertexCount; ++i )
    {
        aVertices[ *( CurrentRemap++ ) ] = *( OldVertex++ );
    }

    // We'll also need to re-index our vertices to point to the new
    // vertex locations.
    for( DWORD i = 0; i < NumFaces * 3; i++ )
    {
        Indices[ i ] = ( WORD )aRemap[ Indices[ i ]];
    }

    delete[] Remap;
    delete[] OldVertices;
}
```

3.1.2 Other Tips On Vertex/Primitive Processing

1. Ensure adequate batching of primitives to amortize runtime and driver overhead.
    a. Use instancing to enable better vertex throughput especially for small batch sizes. This also minimizes state changes and Draw calls.

   b. Aim for 2000 or fewer draw-calls per frame (or less than 50,000 draws per second). Above this number the CPU overhead in the driver can become prohibitive.
   c. Minimize render state changes between batches to reduce the number of pipeline flushes.
2. Use static vertex buffers where possible.
3. Use visibility tests to reject objects that fall outside the view frustum to reduce the impact of objects that are not visible.
   a. Set `D3DRS_CLIPPING` to FALSE for objects that do not need clipping.
4. Take advantage of Early-Z rejection.
   a. Render with a Z-only pass (meaning no color buffer writes or pixel shader execution) followed by a normal render pass. This uses the higher performance of Early-Z to reject occluded fragments which reduces compute times and raster operations.
   b. Balance a Z-only pass against the inherent cost of such an additional pass. A Z-only pass increases the number of draw calls (which can impact the CPU usage) as well as the amount of work done up to the Rasterizer. Measure the performance difference between the two approaches to assess the actual benefit.
   c. Avoid writing Z values (depth) in the pixel shader. Writing the depth value will skip the Early-Z hardware optimization algorithm since it potentially changes the visibility of the fragment.

5. Use the Occlusion Query feature of Microsoft DirectX* to reduce overdraws for complex scenes. Render the bounding box or a simplified model – if it returns zero, then the object does not need to be rendered at all.
   a. Allow sufficient time between Occlusion Query tests and verifying the results to avoid serializing the platform. See the Microsoft DirectX* 10 "Draw Predicated" sample included in the Microsoft DirectX* SDK for more information on this.

      See Section 3.10.5 *Avoid Tight Polling on Queries* for more tips on using queries properly.
6. Consider drawing the opaque overlays in the scene such as heads up displays (HUD) first and writing them to the Z buffer. This reduces the screen rendering area leading to considerable performance improvement.

3.2 Shader Capabilities

Intel® Processor Graphics includes support for Microsoft DirectX* Shader Model 4.1 for 10.1 devices and Shader Model 3.0 for DirectX* 9 devices. Intel® microarchitecture codename Sandy Bridge provides significantly improved computational capability and better handling of large and complicated shaders over prior architectures.

3.2.1 DirectX* 11 on 10

In addition to DirectX* 10.1 hardware support, Intel® Processor Graphics includes partial DirectX11 API support. This allows developers to use the DirectX* 11 API on DirectX* 10 (or 10.1) level hardware. There are two DirectX11-specific features exposed through this paradigm, specifically:

1. Multithreading: The D3D11 API provides means to record command lists on multiple threads (multithreaded rendering) as well as object creation on multiple threads. The Processor Graphics driver currently supports driver-level concurrent object creation.
2. Compute Shader 4 (CS4): A limited subset of DirectX11-level compute shader (CS5) a number of restrictions, specifically:
   - Maximum threads is limited to 768.
   - Z dimension of numthreads/dispatch limited to 1.
   - Only one unordered access view bound per shader (and only RWStructuredBuffers and RWByteAddressBuffers are bindable).
   - Threads are limited to writing to only their own region of groupshared memory (although they can thread from any location).
   - SV_GroupIndex must be used when accessing groupshared memory for writing.
   - Groupshared memory limited to 16KB per group.
   - Atomics and double-precision are not available.

Other DirectX* 11 features (Tessellation, CS5, extended texture formats such as BC6/7) are not supported.

Utilize the DirectX* `ID3D11Device::CheckFormatSupport()` API for individual format support of the D3D11_FORMAT_SUPPORT enumeration.

3.2.2 Tips on Shader Capabilities

1. Reduce texture sampling intensive operations when possible. The following common shader effects typically affect performance and should be tested for performance and optimization. Pay special attention to full screen post processing effects including per-pixel and multiple pass techniques when evaluating graphics related performance bottlenecks.
   a. Glow/Bloom
   b. Motion blur
   c. Depth of field
   d. HDR/tone mapping
   e. Heat distortion
   f. Atmospheric effects
   g. Dynamic Ambient Occlusion
   h. High quality soft shadows
   i. Parallax occlusion mapping with a wide radius
2. Balance texture samples and shader complexity.

   a. Recommend greater than 4:1 ratio of ALU:Sample for better latency coverage. A larger ratio may be better for floating point textures, higher order filtering, and 3D textures.

3. Space your texture sampling calls away from where it is used in pixel shaders when possible and practical to maximize EU utilization.
4. Generically speaking where a choice is available between using programmable shading and fixed function pipeline, programmable shading gives better performance.
   a. In particular*, use sh*ader-based fog instead of fixed function fog on DirectX* 9. Fixed Function fog has been deprecated on SM 3.0.
   b. One notable exception to this rule is the use of the *texkill* instruction. Consider using alpha test to achieve the same result if possible, as it performs better on this platform.
5. Use flow control wisely - it can be expensive. In some cases, it might be advantageous to split a single shader into multiple ones, to avoid flow control statements.
   a. The pixel shader operates on up to 16 pixels in parallel. This means the benefits of dynamic flow control will depend on the likelihood of the number of pixels taking the same branch. If any 2 of those pixels take different paths, all pixels will execute both branches of the control flow.
   b. Dynamic flow control can provide significant benefits by skipping a large number of computations. Ensure that this is used where a large number of instructions can be skipped.
6. Shader Model 3.0 supports dynamic flow instructions such as *if bool, break,* and *break_comp.* It also supports predication registers. Often the dynamic flow control instructions can be eliminated by using predication registers in their place. When possible, use predication instructions over dynamic flow control for shorter branching instruction sequences. Optimize your shader performance by adequate use of your processor graphics - mask alpha if you are not using it.
7. Minimize the usage of geometry shaders, especially in cases where they result in geometry amplification (more geometry output from the GS than came in). If in doubt, examine your application's performance with and without geometry shaders to determine if they are a significant performance issue.
8. In general, minimize use of StreamOut and DrawAuto() for optimal performance, since they can incur a significant bandwidth penalty with their usage of system memory.

3.3 Texture Sample and Pixel Operations

Table 6 Intel® Processor Graphics Texture Sampling and Pixel Specifications

CPU Product	**See** Error! Reference source not found.
Graphics Architecture	**Desktop and Mobile (Processor Graphics 2000 & 3000)**
Format Support	16/32-bit fixed point 16/32-bit floating point operations
Max # of Samplers	Up to 16
Vertex Textures	Yes
Max 2D/3D/Cube Textures	8Kx8K/2Kx2Kx2K/8K
Filtering Type Support	BLF, TLF and Dynamic AF with up to 16 degrees of anisotropic filtering + DirectX* 10.1
Texture Compression	DirectX* 9: DXT1/3/5, ATI1, ATI2; DirectX* 10: BCx
Multi-Sample Render	MSAA 4X
Multi-Target Render	8
Alpha-Blend FP formats	FP16 and FP32 formats are supported. For a complete list, do a caps check on DirectX* 9 and on DirectX* 10, utilize the DirectX* CheckFormatSupport() call as format support may be added in future driver versions.

Table 7 Intel® Processor Graphics Sampler Filtering Specifications

Product	**See** Error! Reference source not found.
32-bit Texels (per clock cycle) **e.g. RGBA UNORM8, RG FP16, R FP32**	
Point/Bilinear	1X
Trilinear	1X
Anisotropic (n samples)	0.5X/n
64-bit Texels (per clock cycle) **e.g. RG FP32, RGBA FP16, RGBA UINT16**	
Point/Bilinear	1X
Trilinear	0.5X
Anisotropic (n samples)	0.25X/n
128-bit Texels (per clock cycle) **e.g. RGBA FP32, RGBA UINT32**	
Point	0.25X
Bilinear	0.25X
Trilinear	0.125X

All sampler filtering types are supported, including dynamic anisotropic filtering.

3.3.1 Tips on Texture Sampling / Pixel Operations

1. Use compressed textures and mip-maps and minimize the use of large textures. Even though the architecture supports up to 8K×8K textures, for optimal performance it is best to use smaller textures.
2. Minimize the use of Anisotropic Filtering, and both Trilinear and Anisotropic filtering for floating point textures. With floating point texture formats, the performance of bilinear and trilinear filtering are not equivalent.

Examine the scene to determine where you can make performance/quality trade-offs with texture filtering. Prefer bilinear filtering where there is little visual difference.

3. Generically speaking, the more compact the texture format being used, the better the performance. DXT compressed formats are best. Minimize the use of 32-bit floating point textures, since they carry a heavy bandwidth penalty and fill the texture caches faster.
4. Use as few render targets as possible, ideally keeping it to less than four. More render targets requires more bandwidth. If in doubt, analyze your performance using a tool such as Intel® GPA to determine if you are fill bound.
5. Minimize the number of Clear calls. Clear surfaces, Color and Z/Stencil buffer at the same time when required.
6. Avoid stencil shadows as they are fill intensive.

3.4 Microsoft DirectX* 10 Optional Features

D3D10 specifies optional features that can be checked for support in the code through API functions like *CheckFormatSupport(...)*, *CheckMultipleQualityLevels(...)*, *CheckFeatureSupport(...)*

The current platform supports more of those optional features than the previous ones and even more features will likely be supported in future. So it is better to explicitly test for such features using those APIs rather than relying on vendor and device ID's for the platform.

For example, DirectX* 10 specifies a large number of resource types and data formats that are optional. Utilize the DirectX* 10 CheckFormatSupport(...) call to determine which ones are supported. Also, utilize the DirectX* 10 CheckFormatSupport(...) call for UNORM and SNORM blending support

The following optional features are supported at the time of review of this Guide:

1. MSAA 2X and 4X on DirectX* 10.
2. 32-bit floating point blending

3.5 Managing Constants on Microsoft DirectX*

Constants are external variables passed as parameters to the shaders; their values remain "constant" during each invocation of the shader program. Despite their name, constants are one of the most frequently changing values in a Microsoft DirectX* application. A shader program can initialize a constant value statically to a value in the shader file or at runtime through the application.

Many of the recommendations described here are standard in the industry. They are very much applicable to Intel processor graphics and the recommendations attempt to detail them in a cohesive manner.

In addition to these points it is worth noting that care should be taken when porting from Microsoft DirectX* 9 to Microsoft DirectX* 10 to maintain performance. For more information on this topic, see the Intel publication "DirectX* Constants Optimizations For Intel® Processor graphics" [2] available on the Intel Software Network.

3.5.1 Tips on Managing Constants on Microsoft DirectX* 9

1. The driver optimizes access to the most frequently used constants. Use less than 32 (FLOAT4) constants per shader to achieve the best performance gain from this feature. Limit the use of dynamic indexed constants (C[ax], C[r]) as these cannot be optimized by the driver and will result in high latency in shaders (dynamic indexed constants are normally found in vertex shaders).
2. Prefer local constants over global constants - the former are better for performance.
3. Immediate constants provide better performance than dynamic indexed constants. In dynamic indexed constants the driver cannot determine a previous index value and needs to create a full size constant buffer space in memory, instead of using the hardware constant buffer.

3.5.2 Tips on Managing Constants on Microsoft DirectX* 10

1. As previously detailed for DirectX* 9 above, the driver optimizes access to the most frequently used constants. The same advice applies for DirectX* 10: use less than 32 constants per shader to achieve the best performance gain from this feature, and limit the use of dynamic indexed constants (C[ax], C[r]) as these cannot be optimized by the driver.
2. For better performance prefer multiple, smaller constant buffers. The constant buffers need to be loaded to the graphics subsystem ahead of the shader execution, and the entire buffer needs to reloaded every time its contents change. Larger the size of the buffer, longer it takes to load the buffer to the graphics subsystem, causing significant performance impacts. If multiple buffers are combined into a single larger buffer, every time any of the contents changes, the entire large buffer will have to be loaded again, degrading the performance. And, because of the performance impact that reloading a constant buffer can have, where possible, if multiple shaders share the same buffer that could help the performance. Whether that performance gain is actually realized depends on whether the shared buffer is still resident when the second shader sharing the buffer is executed. But sharing the buffers between shaders can help but does not hurt the performance. For optimal constant buffer management, smaller packed constant buffers grouped by frequency of update and access pattern are

ideal for higher performance. As an example: organize Per Frame/ Per Pass/ Per Instance constant buffers first which tend to be smaller in size and have a low update rate followed by Per Draw/Per Material constant buffers which may also be small but have a higher update rate. Put large constant buffers like skinning constants last.

3. If there are constants that are unused by most of the shaders move those to the bottom of the constant definition list so that you can bind a smaller buffer to those shaders.
4. Break up constant buffers based on features that are optional in games (e.g. shadows, post-processing effects, etc.). Very likely, due to performance constraints for integrated platforms, some of these features are either going to be disabled or run with a lower setting. So it would be beneficial to break up the constants into separate buffers and then selectively disable the updates to these constant buffers based on the settings selected by the user.
5. When using indexed constant buffers, it is recommended to keep the buffer size tailored to actual needs. For example, if the shader iterates over five elements only, declare a 5-element constant buffer for this shader rather than a general purpose 50-element constant buffer shared among shaders. This allows the driver to optimize placement so that it incurs a low latency path.

3.6 Advanced DirectX* 9 Capabilities

Several advanced features beyond those required by the DirectX* 9 specifications are supported by the Intel Graphics Platforms. This section provides the details on some of them.

The feature list is current for Intel® microarchitecture codename Sandy Bridge, when this Guide is being prepared and is likely to be extended over time. Further, support for the features vary between different Intel platforms. We strongly advise the developers to first confirm in their code that the feature of interest is supported and provide alternative execution paths in the code where features are not supported.

These are capabilities not directly exposed by DirectX* 9 interfaces. Developers will have to use indirect methods to check for their availability.   Code segments showing how to check for such features are given in this section.

3.6.1 FourCC and other surface and texture formats

Intel Graphics platforms support multiple surface-formats beyond those required by the DirectX* 9 specifications. These are listed in Table 8.

Table 8 Additional DirectX* 9 texture and surface formats supported on Intel platforms

Format	Resource Type	Usage	Description
INTZ	Texture	Depth/Stencil	For reading depth buffer as a texture
DF16	Texture	Depth/Stencil	For reading depth buffer as a texture
RESZ	Surface	Render Target	For converting a multisampled depth buffer into a depth stencil texture.
ATI1N	Texture	Texture	Single Channel Texture Compression. Functionally equivalent (but not identical) to DirectX* 10 BC4 format.
ATI2N	Texture	Texture	Two-channel Texture Compression. Functionally equivalent (but not identical) to DirectX* 10 BC5 format.
NULL	Texture	Render Target	Render Target with no memory allocated. Useful as a dummy render target to render exclusively to a depth buffer.
ATOC	Surface	Render Target	Alpha to coverage multisampling.

The following code segment shows a sample outline and works for most FourCC formats.

```
... ... ...
// format of the display mode into which the adapter will be placed
D3DFORMAT AdapterFmt = D3DFMT_X8R8G8B8;
// check for FourCC formats like INTZ
if (pD3D->CheckDeviceFormat( D3DADAPTER_DEFAULT,
                            D3DDEVTYPE_HAL,
                            AdapterFmt,
                            D3DUSAGE_DEPTHSTENCIL,
                               D3DRTYPE_TEXTURE,
                            (D3DFORMAT)(MAKEFOURCC('I','N','T','Z')
                           ) == D3D_OK )
{
    return  true;
}
... ... ...
```

See the code accompanying this Guide for an example of how to test for more formats.

3.6.2 Notes on supported FourCC texture formats

1. INTZ is a depth texture format meant for accessing 24-bit depth buffer and 8-bit stencil buffer. In addition to depth operations this allows for using it for stencil operations as well.

This surface cannot be used as a texture when it is being used for depth buffering, *unless the depth writes are disabled.*

2. DF24 and DF16 formats are meant for use in Percentage-Closer Filtering (PCF) used in shadow mapping applications that use PCF. DF16 has better performance.

3. Intel® microarchitecture codename Sandy Bridge supports MSAA under DirectX* 9. RESZ provides the ability to use multisampling in depth buffer and then copy the result as a single value into a depth-stencil buffer. This process is often referred to as "resolving" a multi-sample depth-stencil buffer into a single-sample depth-stencil buffer.

4. Some ISV's use FourCC formats at times known as ATI1N which allows for the compression of single channel textures. The latest Intel platforms support this for the benefit of those ISV's.

5. 3DC format is also known in the industry as ATI2N format. This format is also supported in the latest Intel platforms.

6. A *NULL Render Target* is a dummy render target format which acts like a valid render target with a crucial difference that the driver will not allocate any memory for it. DX9 requires that a color render target be used for *all* rendering operations. Since a color render target is often not used with depth-only render targets, using a NULL render target in such cases can avoid memory allocation for a color render target will not be used.

7. ATOC, short for "Alpha To Coverage", is meant for interpreting the alpha channel value for approximating the multichannel pixel coverage. The texture itself has no practical use; but you use it as an argument in the *SetRenderState(…)* function, to convert the incoming alpha value output by the pixel shader into multichannel pixel coverage value.

3.6.3 MSAA Under DirectX* 9

2x and 4X MSAA are supported in DX9 on the latest Intel platforms. In general MSAA requires the graphics engine to do more work and hence impacts the performance to varying degrees. The developers should weigh in the tradeoffs ahead of using MSAA in their titles.

3.7 Graphics Memory Allocation

Processor graphics will continue to use the Unified Memory Architecture (UMA) and Dynamic Video Memory Technology (DVMT) as noted in the chart below. As with past processor graphics solutions, UMA specifies that memory resources can be used for video memory when needed. DVMT is an enhancement of the UMA concept, wherein system memory is allocated for balanced graphics and system performance. DVMT dynamically responds to system requirements and application demands, by allocating the proper amount of display, texturing, and buffer. The operating system views the Intel graphics driver as an application, which uses system memory to request allocation of additional memory for 3D applications, and returns the memory to the operating system when no longer required.

Table 9 Intel® Processor Graphics Memory Specifications

CPU Product	**See** Error! Reference source not found.	
Segment	**Processor Graphics 2000**	**Processor Graphics 3000**
Memory BW (GB/s)	17.1 - 21.3	17.1-25.6
UMA Capability	2x DDR3 up to 1333	2x DDR3 up to 1600
Max DVMT (XP) 1 or 2GB System Memory	Limited to 1GB max for all system memory configurations	
Max DVMT (Windows Vista*/Windows* 7) x86/x64: System Memory	The memory is managed by the operating system and the driver.	
Memory Interface	64 bits	

3.7.1 Checking for Available Memory

Graphics applications often check for the amount of available free video memory early in execution. Developers typically want to know the amount of memory that the graphics device can access at full performance.

As a result of the dynamic allocation of graphics memory performed by the Intel® Processor Graphics devices (based on application requests), memory checks that only request the amount of 'local' or 'dedicated' graphics memory available do not supply a number that is appropriate for the Intel® Processor Graphics devices.

The Microsoft DirectX* SDK (June 2010) includes the VideoMemory sample code which demonstrates 5 commonly used methods to detect the total amount of video memory. Of these tests, GetVideoMemoryViaWMI is recommended. All other methods either return the local/dedicated graphics memory and consequently will report incorrect results on Intel® Processor Graphics, or will report the sum of the dedicated memory and the shared memory, something that is typically not suitable for discrete graphics devices. For more information, see the sample code: http://msdn.microsoft.com/en-us/library/ee419018(v=VS.85).aspx

3.7.2 Tips On Resource Management

1. Allocate surfaces in priority order. The render surfaces that will be used most frequently should be allocated first. On Microsoft DirectX* 10, memory is taken care of for you by the OS. On Microsoft DirectX* 9:

   a. Use D3DPOOL_DEFAULT for lockable memory (dynamic vertex/index buffers).
   b. Use D3DPOOL_MANAGED for non-lockable memory (textures, back buffers, etc).

2. On D3D10, use of the Copy…() methods are preferred over calling the Update…() operations. Partial or sub-resource copies should be used sparingly. For example when updating all or most of the LODs of a resource use CopyResource() rather than multiple CopySubResource().

3.8 Identifying Intel® Processor Graphics and Specifying Graphics Presets

Games often specify a range of graphics capabilities and presets to identify with a Low, Medium, and High fidelity level. Please refer to the Appendix for sample code that demonstrates how to identify Intel® Processor Graphics versions and set fidelity levels for older generations (Low) to the most recent (Medium) based on the requested D3D*_FEATURE_LEVEL.

Note that on Windows* 7, multiple graphics adapters are supported so care should be taken in determining which adapter will be used for rendering.

3.9 Surviving a Graphics Hardware Switch on the Fly

Intel in combination with third party graphics vendors jointly developed a switchable graphics solution that allows end users to switch on-the-fly, between two heterogeneous graphics hardware systems without a reboot. This functionality can incorporate the energy efficiency of Intel processor graphics with the 3D performance of discrete graphics hardware in a single notebook solution. This technology is applicable to about 30 million discrete graphics hardware notebooks purchased annually. Currently most applications running on PC platforms with heterogeneous graphics hardware do not survive when switched between the two at run-time as they do not re-query underlying graphics capability when the active adapter changes.

Keys to handling graphics hardware switches:

- New applications should be aware of multiple graphics hardware configurations and handle the D3DERR_DEVICELOST and WM_DISPLAYCHANGE messages properly.
- Legacy applications where possible should develop and distribute patches for old games to handle the D3DERR_DEVICELOST and WM_DISPLAYCHANGE messages.

3.9.1 Microsoft DirectX* 9 Algorithm

Microsoft DirectX* 9 applications should follow the below procedure to query GFX adapter's capabilities (re-create DX object/device) on D3DERR_DEVICELOST:

1. Manually save the current context including state and draw information in the application.
2. Query if the graphics adapter has changed, using the Windows* API's EnumDisplaySettings() or EnumDisplayDevices().
3. If the adapter has changed, then:
    a. Recreate a Microsoft DirectX* device.
    b. Restore the context.
    c. Continue rendering from last scene rendered before the `D3DERR_DEVICELOST` event occurred.

3.9.2 Algorithm for DirectX* 10

By design, DirectX* 10 does not have the concept of device lost until the next Present, and the developer is guaranteed the API will keep working until then. The changes in Microsoft DirectX* 10 applications are:

1. Check for `WM_DISPLAYCHANGE` windows message in the message handler.
2. Query if the graphics adapter has changed using the Windows* API's EnumDisplaySettings() or EnumDisplayDevices().
3. If yes, then save off the current context including state and draw information in the application and then:
    a. Recreate the Microsoft DirectX* device.
    b. Restore the context.
    c. Continue rendering from the last scene rendered before the `WM_DISPLAYCHANGE` message occurred.

3.10 Some suggestions, tips & tricks from the field

The items in this section are based on the observations of Intel engineers with code from developers with different levels of experiences. These are collected here as a checklist for reference for developers. Some of the items are generic to all graphics platforms.

3.10.1 Device Capabilities

Intel® microarchitecture codename Sandy Bridge supports DirectX* functionality up to and including full D3D10.1 support.

If you encounter a Direct3D feature that does not work on the latest drivers on this device, please contact your Intel Account Manager. Intel will investigate these issues for future drivers and should be able to suggest workarounds.

3.10.2 DirectX* 9 Extensions

Several hardware vendors support their own extensions to the DirectX* 9 specifications through specific texture formats and render paths that are not part of Microsoft's official DirectX* 9 specifications.

To ensure maximum compatibility with these extensions, Intel now supports many of those extensions. A list of those, current as of the release of this Guide, is given in section 3.6 Advanced DirectX* 9 Capabilities (page 24).

If there are additional extensions that you believe are useful, (or have OpenGL* extensions that you require) please let your Intel Account Manager know.

3.10.3 Revisit Assumptions on Performance

Intel® Processor Graphics is continually increasing functionality and performance. As well as the addition of full D3D10.1 support and increased capabilities previously mentioned, the performance profile has been improved significantly for this platform. As such, it is advised to remove previous restrictions and scale your title to match this increased performance and functionality.

Should you see unexpected issues, please follow these steps:

1. Verify you are running the latest drivers. This platform is evolving, so there will be frequent driver updates. Check for updates at http://www.intel.com and if you are an Intel software partner, at http://platformsw.intel.com.

   Intel graphics drivers follow a naming convention that use four field numbers - for example, 15.21.8.2279. The number in the last field - 2279 in the example - increases sequentially with each driver release. So a driver with a higher number there is more recent. You always want to have the most recent driver installed on your system.
2. If you suspect that it is a functionality bug, try to recreate the bug with the reference rasterizer.
3. Look for easily fixed hotspots using the Intel® Graphics Performance Analyzers (Intel® GPA). Talk to your Intel Account Manager if you do not already have access to this tool.
4. If the above steps do not resolve the issue, or you need additional help determining the root cause, please contact us - see the section Further Help Beyond this Guide in this document. Intel engineers are available to help enable your title to run effectively on our platforms. This enabling can potentially include (but is not limited to):
   a. Training on using Intel GPA to get the best results for your title.
   b. In-depth performance analysis of your code running on our platform, with specific feedback on optimization opportunities.
   c. Championing the resolution of your issues within Intel, such as helping resolve or workaround driver issues, addressing tool issues, etc.

3.10.4 Avoid Writing to Unlocked Buffers

We have found multiple cases of corruption which were caused by the writes to unlocked buffers. Typically the title first locked the buffer, then unlocked the buffer and then attempted to write to the buffer that is already unlocked. This *sometimes* works because the system might have not moved the buffer and has not dispatched the rendering commands to the graphics hardware. This is inconsistent with Microsoft DirectX* specifications and it is not safe for applications to expect it to work consistently.

Avoid writing to buffers that have not been locked. Some drivers on other hardware platforms are more forgiving of this approach, and may handle it gracefully. The Intel driver makes optimizations based on the specified behavior of the API, so the behavior of this platform will be undefined in this case.

Always test your application with the DirectX* debug runtime enabled. The debug runtime will catch scenarios like this and help you avoid problems should driver behavior change in the future.

3.10.5 Avoid Tight Polling on Queries

Tight polling on event/occlusion queries degrades performance by interfering with the graphics subsystem's turbo mode operation.

Allow for queries to work asynchronously and avoid waiting on the query response immediately after sending the query. Issue queries as early as possible in the frame, and issue their dependent draws as late as possible. If the query result is not available at the time the draw is to be submitted just issue the draw. Any additional delay at this point will cause a pipeline stall.

4 Appendix: Sample Code for Identifying Intel® Processor Graphics and Specifying Graphics Presets

The following sample code and configuration file demonstrates how to identify Intel® Processor Graphics versions and set fidelity levels for older generations (Low) to the most recent (Medium) based on the requested D3D*_FEATURE_LEVEL.

Example Source Code:

```
// Copyright 2010 Intel Corporation
// All Rights Reserved
//
// Permission is granted to use, copy, distribute and prepare derivative works of this
// software for any purpose and without fee, provided, that the above copyright notice
// and this statement appear in all copies.  Intel makes no representations about the
// suitability of this software for any purpose.  THIS SOFTWARE IS PROVIDED ""AS IS.""
// INTEL SPECIFICALLY DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, AND ALL LIABILITY,
// INCLUDING CONSEQUENTIAL AND OTHER INDIRECT DAMAGES, FOR THE USE OF THIS SOFTWARE,
// INCLUDING LIABILITY FOR INFRINGEMENT OF ANY PROPRIETARY RIGHTS, AND INCLUDING THE
// WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  Intel does not
// assume any responsibility for any errors which may appear in this software nor any
// responsibility to update it.


//
// DeviceId.cpp : Implements the Graphics Device detection and graphics settings
//                configuration functions.
//

#include <stdio.h>
#include <tchar.h>

#include <D3DCommon.h>
#include <DXGI.h>
#include <D3D9.h>

static const int FIRST_GFX_ADAPTER = 0;

// Define settings to reflect Fidelity abstraction levels you need
typedef enum {
               NotCompatible,   // Found Graphics is not compatible with the app
               Low,
               Medium,
               High,
               Undefined        // No predefined setting found in cfg file.
                                // Use a default level for unknown video cards.
             }
             PresetLevel;

/*****************************************************************************************
 * getGraphicsDeviceID
 *
 *     Function to get the primary graphics device's Vendor ID and Device ID, either
 *     through the new DXGI interface or through the older D3D9 interfaces.
 *
 *****************************************************************************************/
```

```
bool getGraphicsDeviceID( unsigned int& VendorId, unsigned int& DeviceId )
{
    bool retVal = false;
    bool bHasWDDMDriver = false;

    HMODULE hD3D9 = LoadLibrary( L"d3d9.dll" );
    if ( hD3D9 == NULL )
        return false;

    /*
     * Try to create IDirect3D9Ex interface (also known as a DX9L interface).
     * This interface can only be created if the driver is a WDDM driver.
     */

    // Define a function pointer to the Direct3DCreate9Ex function.
    typedef HRESULT (WINAPI *LPDIRECT3DCREATE9EX)( UINT, void **);

    // Obtain the address of the Direct3DCreate9Ex function.
    LPDIRECT3DCREATE9EX pD3D9Create9Ex = NULL;
    pD3D9Create9Ex = (LPDIRECT3DCREATE9EX) GetProcAddress( hD3D9, "Direct3DCreate9Ex" );

    bHasWDDMDriver = (pD3D9Create9Ex != NULL);

    if( bHasWDDMDriver )
    {
        // Has WDDM Driver (Vista, and later)
        HMODULE hDXGI = NULL;

        hDXGI = LoadLibrary( L"dxgi.dll" );

        // DXGI libs should really be present when WDDM driver present.
        if ( hDXGI )
        {
            // Define a function pointer to the CreateDXGIFactory1 function.
            typedef HRESULT (WINAPI *LPCREATEDXGIFACTORY)(REFIID riid, void **ppFactory);

            // Obtain the address of the CreateDXGIFactory1 function.
            LPCREATEDXGIFACTORY pCreateDXGIFactory = NULL;
            pCreateDXGIFactory = (LPCREATEDXGIFACTORY) GetProcAddress( hDXGI,
                                                                      "CreateDXGIFactory" );

            if ( pCreateDXGIFactory )
            {
                // Got the function hook from the DLL
                // Create an IDXGIFactory object.
                IDXGIFactory * pFactory;
                if ( SUCCEEDED( (*pCreateDXGIFactory)(__uuidof(IDXGIFactory),
                                                     (void**)(&pFactory) ) ) )
                {
                    // Enumerate adapters.
                    // Code here only gets the info for the first adapter.
                    // If secondary or multiple Gfx adapters will be used,
                    // the code needs to be modified to accomodate that.
                    IDXGIAdapter *pAdapter;
                    if ( SUCCEEDED( pFactory->EnumAdapters( FIRST_GFX_ADAPTER,
                                                            &pAdapter ) ) )
                    {
                        DXGI_ADAPTER_DESC adapterDesc;
                        pAdapter->GetDesc( &adapterDesc );

                        // Extract Vendor and Device ID information from adapter descriptor
                        VendorId = adapterDesc.VendorId;
                        DeviceId = adapterDesc.DeviceId;

                        pAdapter->Release();

                        retVal = true;
                    }
                }
            }

            FreeLibrary( hDXGI );
        }
    }
    else
    {
        /*
         * Does NOT have WDDM Driver. We must be on XP.
         * Let's try using the Direct3DCreate9 function (instead of DXGI)
         */

        // Define a function pointer to the Direct3DCreate9 function.
        typedef IDirect3D9* (WINAPI *LPDIRECT3DCREATE9)( UINT );

        // Obtain the address of the Direct3DCreate9 function.
        LPDIRECT3DCREATE9 pD3D9Create9 = NULL;
        pD3D9Create9 = (LPDIRECT3DCREATE9) GetProcAddress( hD3D9, "Direct3DCreate9" );
        if( pD3D9Create9 )
        {
            // Initialize the D3D9 interface
            LPDIRECT3D9 pD3D = NULL;
            if ( (pD3D = (*pD3D9Create9)(D3D_SDK_VERSION)) != NULL )
            {
                D3DADAPTER_IDENTIFIER9    adapterDesc;
                // Enumerate adapters._Code here only gets the info for the first adapter.
                if ( pD3D->GetAdapterIdentifier( FIRST_GFX_ADAPTER, 0,
                                                  &adapterDesc ) == D3D_OK )
                {
                    VendorId = adapterDesc.VendorId;
                    DeviceId = adapterDesc.DeviceId;

                    retVal = true;
                }

                pD3D->Release();
            }
        }
    }

    FreeLibrary( hD3D9 );
    return retVal;
}



/******************************************************************************************
 * getDefaultFidelityPresets
 *
 *     Function to find / set the default fidelity preset level, based on the type
 *     of graphics adapter present.
 *
 *     The guidelines for graphics preset levels for Intel devices is a generic one
 *     based on our observations with various contemporary games. You would have to
 *     change it if your game already plays well on the older hardware even at high
 *     settings.
 *
 ******************************************************************************************/

PresetLevel getDefaultFidelityPresets( void )
{
    unsigned int VendorId, DeviceId;

    PresetLevel presets = Undefined;

    if ( !getGraphicsDeviceID ( VendorId, DeviceId ) )
    {
        return NotCompatible;
    }

    FILE *fp = NULL;
    switch( VendorId )
    {
        case 0x8086:
            fopen_s ( &fp, "IntelGfx.cfg", "r" );
            break;

           // Add cases to handle other graphics vendors…

        default:
            break;
    }
    if( fp )
    {
        char line[100];
        char *context = NULL;

        char *szVendorId = NULL;
        char *szDeviceId = NULL;
        char *szPresetLevel = NULL;

        while ( fgets ( line, _countof(line), fp ) )   // read one line at a time till EOF
        {
             // Parse and remove the comment part of any line
            int i; for( i = 0; line[i] && line[i]!=';'; ++i ); line[i] = '\0';

            // Try to extract VendorId, DeviceId and recommended Default Preset Level
            szVendorId    = strtok_s( line, ",\n", &context );
            szDeviceId    = strtok_s( NULL, ",\n", &context );
            szPresetLevel = strtok_s( NULL, ",\n", &context );

            if( ( szVendorId == NULL ) ||
                ( szDeviceId == NULL ) ||
                ( szPresetLevel == NULL ) )
            {
                continue;  // blank or improper line in cfg file - skip to next line
            }

            unsigned int vId, dId;
            sscanf_s( szVendorId, "%x", &vId );
            sscanf_s( szDeviceId, "%x", &dId );

           // If current graphics device is found in the cfg file, use the
           // pre-configured default Graphics Presets setting.
            if( ( vId == VendorId ) && ( dId == DeviceId ) )
            {
                // Found the device
                char s[10];
                sscanf_s( szPresetLevel, "%s", s, _countof(s) );

                if (!_stricmp(s, "Low"))
                    presets = Low;
                else if (!_stricmp(s, "Medium"))
                    presets = Medium;
                else if (!_stricmp(s, "High"))
                    presets = High;
                else
                    presets = NotCompatible;

                break; // Done reading file.
            }
        }

        fclose( fp );  // Close open file handle
    }

    // If the current graphics device was not listed in any of the config
   // files, or if config file not found, use Low settings as default.
    if ( presets == Undefined )
        presets = Low;

    return presets;
}
```

Example Configuration File:

```
;
; Intel Graphics Preset Levels
;
; Format:
; VendorIDHex, DeviceIDHex, CapabilityEnum      ;Commented name of cards
;


0x8086, 0x2582, Low    ;  SM2   ; Intel(R) 82915G/GV/910GL Express Chipset Family
0x8086, 0x2782, Low    ;  SM2   ; Intel(R) 82915G/GV/910GL Express Chipset Family
0x8086, 0x2592, Low    ;  SM2   ; Mobile Intel(R) 82915GM/GMS, 910GML Express Chipset Family
```

```
0x8086, 0x2792, Low    ;  SM2   ; Mobile Intel(R) 82915GM/GMS, 910GML Express Chipset Family
0x8086, 0x2772, Low    ;  SM2   ; Intel(R) 82945G Express Chipset Family
0x8086, 0x2776, Low    ;  SM2   ; Intel(R) 82945G Express Chipset Family
0x8086, 0x27A2, Low    ;  SM2   ; Mobile Intel(R) 945GM Express Chipset Family
0x8086, 0x27A6, Low    ;  SM2   ; Mobile Intel(R) 945GM Express Chipset Family
0x8086, 0x2972, Low    ;  SM2   ; Intel(R) 946GZ Express Chipset Family
0x8086, 0x2973, Low    ;  SM2   ; Intel(R) 946GZ Express Chipset Family
0x8086, 0x2992, Low    ;  SM2   ; Intel(R) Q965/Q963 Express Chipset Family
0x8086, 0x2993, Low    ;  SM2   ; Intel(R) Q965/Q963 Express Chipset Family
0x8086, 0x29B2, Low    ;  SM2   ; Intel(R) Q35 Express Chipset Family
0x8086, 0x29B3, Low    ;  SM2   ; Intel(R) Q35 Express Chipset Family
0x8086, 0x29C2, Low    ;  SM2   ; Intel(R) G33/G31 Express Chipset Family
0x8086, 0x29C3, Low    ;  SM2   ; Intel(R) G33/G31 Express Chipset Family
0x8086, 0x29D2, Low    ;  SM2   ; Intel(R) Q33 Express Chipset Family
0x8086, 0x29D3, Low    ;  SM2   ; Intel(R) Q33 Express Chipset Family

0x8086, 0xA001, Low    ;  SM2   ; Intel(R) NetTop Atom D410 (GMA 3150)
0x8086, 0xA002, Low    ;  SM2   ; Intel(R) NetTop Atom D510 (GMA 3150)
0x8086, 0xA011, Low    ;  SM2   ; Intel(R) NetBook Atom N4x0 (GMA 3150)
0x8086, 0xA012, Low    ;  SM2   ; Intel(R) NetBook Atom N4x0 (GMA 3150)

0x8086, 0x29A2, Low    ;  SM3   ; Intel(R) G965 Express Chipset Family
0x8086, 0x29A3, Low    ;  SM3   ; Intel(R) G965 Express Chipset Family

0x8086, 0x8108, Low    ;  SM3   ; Intel(R) GMA 500 (Poulsbo) on MID platforms
0x8086, 0x8109, Low    ;  SM3   ; Intel(R) GMA 500 (Poulsbo) on MID platforms

0x8086, 0x2982, Low    ;  SM4   ; Intel(R) G35 Express Chipset Family
0x8086, 0x2983, Low    ;  SM4   ; Intel(R) G35 Express Chipset Family
0x8086, 0x2A02, Low    ;  SM4   ; Mobile Intel(R) 965 Express Chipset Family
0x8086, 0x2A03, Low    ;  SM4   ; Mobile Intel(R) 965 Express Chipset Family
0x8086, 0x2A12, Low    ;  SM4   ; Mobile Intel(R) 965 Express Chipset Family
0x8086, 0x2A13, Low    ;  SM4   ; Mobile Intel(R) 965 Express Chipset Family
0x8086, 0x2A42, Low    ;  SM4   ; Mobile Intel(R) 4 Series Express Chipset Family
0x8086, 0x2A43, Low    ;  SM4   ; Mobile Intel(R) 4 Series Express Chipset Family
0x8086, 0x2E02, Low    ;  SM4   ; Intel(R) 4 Series Express Chipset
0x8086, 0x2E03, Low    ;  SM4   ; Intel(R) 4 Series Express Chipset
0x8086, 0x2E22, Low    ;  SM4   ; Intel(R) G45/G43 Express Chipset
0x8086, 0x2E23, Low    ;  SM4   ; Intel(R) G45/G43 Express Chipset
0x8086, 0x2E12, Low    ;  SM4   ; Intel(R) Q45/Q43 Express Chipset
0x8086, 0x2E13, Low    ;  SM4   ; Intel(R) Q45/Q43 Express Chipset
0x8086, 0x2E32, Low    ;  SM4   ; Intel(R) G41 Express Chipset
0x8086, 0x2E33, Low    ;  SM4   ; Intel(R) G41 Express Chipset
0x8086, 0x2E42, Low    ;  SM4   ; Intel(R) B43 Express Chipset
0x8086, 0x2E43, Low    ;  SM4   ; Intel(R) B43 Express Chipset
0x8086, 0x2E92, Low    ;  SM4   ; Intel(R) B43 Express Chipset
0x8086, 0x2E93, Low    ;  SM4   ; Intel(R) B43 Express Chipset
0x8086, 0x0046, Low    ;  SM4   ; Intel(R) Processor Graphics - Core i3/i5/i7 Mobile
Processors
0x8086, 0x0042, Low    ;  SM4   ; Intel(R) Processor Graphics - Core i3/i5 + Pentium G9650
Processors

0x8086, 0x0106, Low    ;  SM4.1 ; Mobile SandyBridge Processor GRAPHICS 2000
0x8086, 0x0102, Low    ;  SM4.1 ; Desktop SandyBridge Processor GRAPHICS 2000
0x8086, 0x010A, Low    ;  SM4.1 ; SandyBridge Server

0x8086, 0x0112, Medium ;  SM4.1 ; Desktop SandyBridge Processor GRAPHICS 3000
0x8086, 0x0122, Medium ;  SM4.1 ; Desktop SandyBridge Processor GRAPHICS 3000
0x8086, 0x0116, Medium ;  SM4.1 ; Mobile SandyBridge Processor GRAPHICS 3000

0x8086, 0x0126, Medium ;  SM4.1 ; Mobile SandyBrdige Processor GRAPHICS 3000
```

5 Support

- Intel's processor graphics chipset development community forum:

  http://software.intel.com/en-us/forums/developing-software-for-visual-computing/

- Game programming resources:

  http://software.intel.com/en-us/visual-computing/

- Intel® Software Network:

  http://software.intel.com/en-us/

- Intel® Software Partner Program:

  http://www.intel.com/software/partner/visualcomputing/

- Intel® Visual Adrenaline graphics and gaming campaign:

  http://www.intel.com/software/visualadrenaline/

- Intel® Graphics Performance Analyzers (Intel® GPA):

  http://software.intel.com/en-us/articles/intel-gpa/

- Intel® Parallel Studio:

  http://software.intel.com/en-us/articles/intel-parallel-studio-home/

- Intel® VTune™ Amplifier XE (Formerly Intel® VTune™ Performance Analyzer):

  http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/

6 References

[1] "Copying and Accessing Resource Data (Direct3D 10)". Direct3D Programming Guide. Microsoft DirectX* SDK (November 2008).

[2] "DirectX* Constants Optimizations for Intel processor graphics". Intel Software Network, Intel: http://software.intel.com/en-us/articles/directx-constants-optimizations-for-intel-integrated-graphics/.

THE RISE OF MOBILE GAMING ON ANDROID: QUALCOMM® SNAPDRAGON™ TECHNOLOGY LEADERSHIP

ATI 2092
LG v. ATI
IPR2015-00326

Qualcomm Technologies, Inc.

Qualcomm Snapdragon, Qualcomm Adreno, Qualcomm TruPalette, Qualcomm ecoPix, FlexRender, Trepn, Brew, Hexagon and Krait are products of Qualcomm Technologies, Inc. Qualcomm Vuforia is a product of Qualcomm Connected Experiences, Inc. Smart Terrain is a feature of the Qualcomm Vuforia SDK. AllJoyn open source project is hosted by the Allseen Alliance.

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Table of Contents

1 Executive summary

Mobile gaming is the fastest growing segment in the game industry.[1] Today's consumers want immersive, connected gaming with all-day battery life that provides visually stunning graphics and high-fidelity audio. Android is helping fulfill that desire. With a large user base, Android devices provide a growing opportunity for game developers to generate revenue on a global scale.

The rapid iteration and fast innovation of Android have created a thriving ecosystem with numerous custom-designed form factors across device tiers. However, there are three challenges to taking Android gaming to the next level, including:

- How to create immersive gaming experiences within the power and thermal constraints of mobile devices?
- How to develop a game that addresses a sizeable portion of the mobile gaming segment with minimal code variations?
- How to take advantage of the hardware's capabilities without sacrificing time-to-market?

Qualcomm Technologies, Inc. (QTI) was the number one provider of Android smartphone application processor (AP) shipments in 2013.[2] Its Qualcomm® Snapdragon™ processors[3] are fully integrated system on a chip (SoC) solutions designed with mobile in mind, handling everything from the most advanced, console-quality mobile games to the most popular casual games, all while delivering long battery life. This paper will describe the rise of mobile gaming on Android and how QTI is meeting the three challenges by:

- Creating low-power mobile processors by taking a heterogeneous computing (HC) approach and designing an efficient SoC architecture.
- Providing a consistent development platform across tiers due to its scalable architecture.
- Offering tools, support, and advanced technologies to easily unlock the full potential of Snapdragon processors for developers.

2 The rise of mobile gaming on Android

Mobile gaming is the fastest growing segment in the game industry, with a 30% compound annual revenue growth rate projected from 2013 to 2015.[1]

Android burst onto the scene in 2008. The *open* nature and rapid iteration of the platform, combined with improving hardware, and a growing installed base of devices worldwide, has created excellent opportunities for the mobile gaming ecosystem.

[1] Source: Gartner, October 2013, "Forecast Video Game Ecosystem Worldwide"
[2] Source: Strategy Analytics, April 2014
[3] Snapdragon processors extend across four product tiers: Snapdragon 800, Snapdragon 600, Snapdragon 400, and Snapdragon 200.

Through standard application program interfaces (APIs), Android exposes key platform capabilities to developers, such as connectivity, sensors, and graphics rendering. Android also has a history of quickly adopting the latest graphics standards, like OpenGL ES. The consistent release of new APIs helps limit fragmentation, improve ease of use, and decrease time-to market.

To take advantage of increasing consumer engagement on the Google Play app store, developers are now creating games that utilize the latest Android capabilities (e.g., 3D graphics). As a result, new game brands have been introduced (e.g., *Real Soccer*, *Asphalt*, *Real Racing*, and *Modern Combat*). Debuting in the smartphone, Android is now extending to other form factors, including tablets, handheld gaming devices, set-top-boxes, TVs, and more.

Total mobile gaming revenues (for all platforms) are projected to grow from $13 billion in 2013 to $22 billion in 2015[4]. Within this category, the Android platform provides a large and growing opportunity for developers. On the device side, 78.4% of smartphones shipped last year were Android, outselling devices based on the next leading smartphone mobile OS by almost 5x.[5] Android device shipments (smartphones and tablets) are projected to exceed one billion units in 2014.[6] On the apps side, 75% of Android users play games[7], accounting for 90% of the app-generated revenue on Google Play[8].

3 Immersive mobile gaming experiences at low power

Immersive gaming requires more than just console-quality graphics. Other elements include cinema-quality sound, realistic effects, instantaneous response times (low latency), and the ability to play anywhere. Figure 1 highlights the key components that contribute to the overall gaming experience**.**

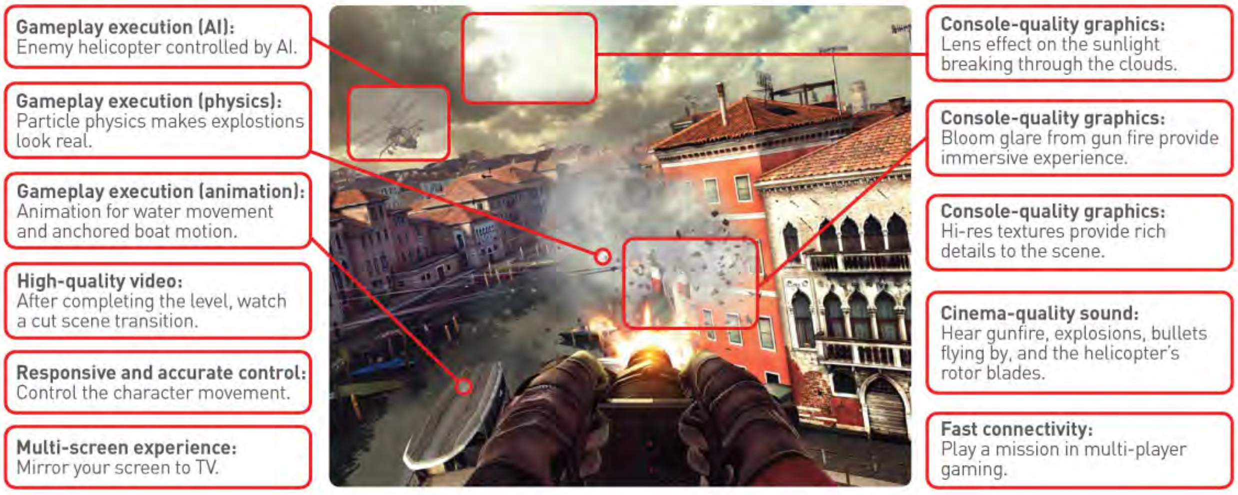

Gameplay execution (AI): Enemy helicopter controlled by AI.
Gameplay execution (physics): Particle physics makes explostions look real.
Gameplay execution (animation): Animation for water movement and anchored boat motion.
High-quality video: After completing the level, watch a cut scene transition.
Responsive and accurate control: Control the character movement.
Multi-screen experience: Mirror your screen to TV.
Console-quality graphics: Lens effect on the sunlight breaking through the clouds.
Console-quality graphics: Bloom glare from gun fire provide immersive experience.
Console-quality graphics: Hi-res textures provide rich details to the scene.
Cinema-quality sound: Hear gunfire, explosions, bullets flying by, and the helicopter's rotor blades.
Fast connectivity: Play a mission in multi-player gaming.

Figure 1: Example of key gaming components in Modern Combat 5: Blackout by Gameloft

[4] Gartner, October 2013, "Forecast Video Game Ecosystem Worldwide"
[5] Gartner, "Gartner Says Annual Smartphone Sales Surpassed Sales of Feature Phones for the First Time in 2013"
[6] Gartner, "Gartner Says Worldwide Traditional PC, Tablet, Ultramobile and Mobile Phone Shipments On Pace to Grow 7.6% in 2014"
[7] Kris Holt, "Google Adds Some Serious Oomph to Play Games"
[8] eMarketer, "Game Apps Are No. 1 for Amazon, Apple and Google"

Today's mobile graphics are approaching the visual quality of consoles and PCs with high-end graphics cards.

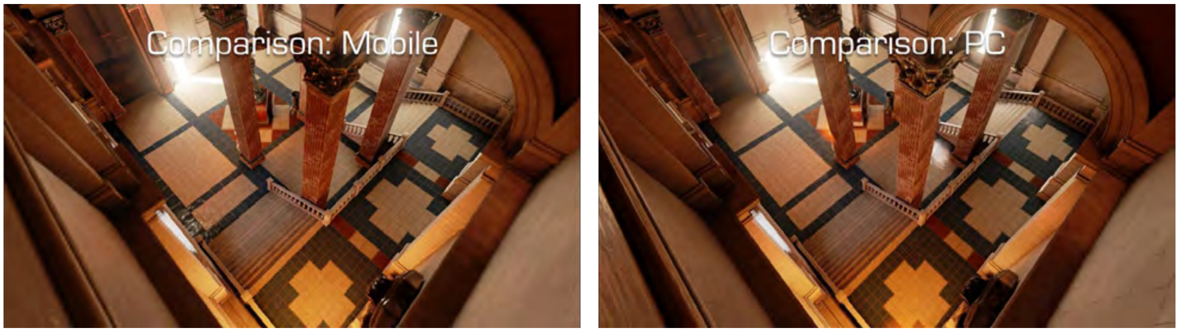


Figure 2: Mobile (left) vs. PC (right) rendering for Epic Unreal Engine 4

Consumers expect console-quality graphics and long battery life when playing any mobile game, from simple casual games like *Candy Crush* to more immersive games like *Modern Combat 5: Blackout*. However, enabling advanced mobile gaming experiences for sustained periods is challenging. Consoles and PCs have the luxury of being able to use fans (and large heat sinks) while drawing triple-digit wattages from an external power outlet. Mobile devices, on the other hand, are *passively* cooled, powered by a battery, and limited to single-digit wattages in order to meet the power and thermal constraints of the thin form factor.

Through its Snapdragon processors, QTI is addressing the performance and power challenge by:

- Taking a power-optimized, *heterogeneous approach to mobile computing.*
- Analyzing performance bottlenecks and designing an *efficient SoC architecture* that improves each generation.

Figure 3: Asphalt 8: Airborne by Gameloft

"*Qualcomm® Snapdragon™ processors really help enable us to create an immersive experience for the gamer...*"

Sylvain Baudry
Business Development Director
Gameloft

3.1 Heterogeneous computing: specialized engines designed for efficient processing

QTI has a long history of taking a heterogeneous computing approach. This approach intelligently utilizes specialized engines, such as the GPU, DSP, and display engine, to support new immersive experiences, while helping to maximize battery life and thermal efficiency.

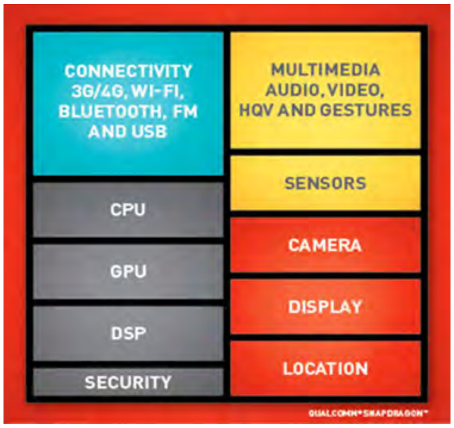

CONNECTIVITY 3G/4G, WI-FI, BLUETOOTH, FM AND USB
MULTIMEDIA AUDIO, VIDEO, HQV AND GESTURES
SENSORS
CPU
GPU
CAMERA
DSP
DISPLAY
SECURITY
LOCATION

Figure 4: Snapdragon processing engines

As mentioned in Section 3, games are comprised of multiple components. These components may be implemented as one or more tasks. To support the optimal gaming experience at the lowest power and thermal levels, each task should be designed to run on the most appropriate engine. Figure 5 is an example of this.

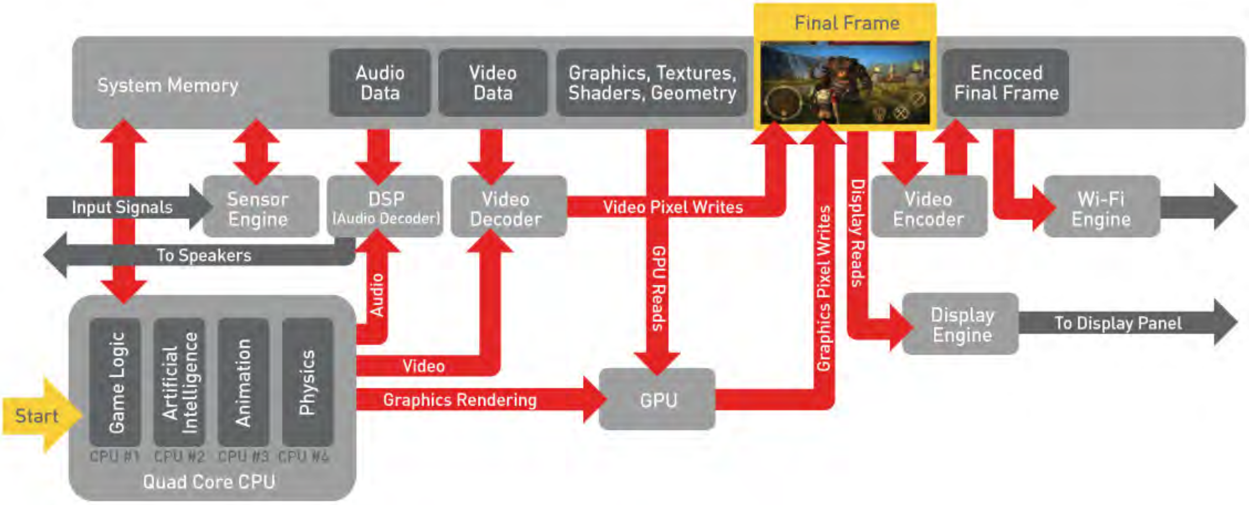


Figure 5: High-level view of gaming tasks being appropriately distributed to specialized engines

The rest of this section provides examples of how Snapdragon processors are designed to run key gaming tasks on the most appropriate processing engines.

Controlling gameplay execution — keeping everything in sync: The CPU dispatches work to the appropriate specialized engine and processes individual functions within the game as separate tasks. For example AI, animation, physics, and gameplay are processed on the CPU.

Functions like physics and AI require high-precision math, which can be efficiently processed on the VeNum floating point units of QTI's custom-designed Krait™ CPU. In addition, because the tasks are independent, they can be written as separate threads and run on separate CPUs for optimal performance and latency.

QTI has been innovating its power management and scheduling software for years to provide the different amounts of processing that various tasks require. For example, Krait CPUs have long employed asynchronous symmetric multi-processing (aSMP) so that optimized performance is delivered per CPU core, thus saving power.

Console-quality graphics — advanced features and graphics rendering (at low power): When most people think of modern mobile gaming, it is the console-quality graphics that immediately come to mind. Rendering graphics at low power and at high frame rates requires a specialized engine, the GPU. QTI's custom-designed Qualcomm® Adreno™ GPUs are a family of low power, fully programmable GPUs that are designed for optimal mobile gaming performance.

Adreno GPUs provide comprehensive support for the latest graphics APIs. The Khronos OpenGL ES specifications define the primary graphics APIs for Android gaming. QTI works closely with the Khronos group to help define the standards, and the Adreno GPU is optimized to support OpenGL ES (from the silicon to the drivers and software stack).

Adreno GPUs have a long history of graphics technology leadership. They support many design innovations for efficient graphics processing, including a flexible, power-efficient unified shader architecture, which is designed to support dynamic resource allocation (for optimal shader processing).

> *"Epic now has brought Unreal Engine 4 to Android with the Snapdragon 800 and 805 chipsets from Qualcomm Technologies. Recently we worked with Qualcomm [Technologies] to elevate graphics to the next level on the … Adreno GPU hardware, which delivers some of the most power-efficient unified shader capabilities we've seen yet for Android smartphones and tablets."*
> *— Niklas "Smedis" Smedberg, Epic Games*

The Adreno GPU also supports dynamic FlexRender™ technology, which is designed to intelligently choose between immediate/direct mode and deferred/tile-based rendering, to render various tasks (such as user interface and gameplay) in a more efficient manner.

Custom designing the Adreno GPUs allows QTI to evolve the architecture in a timely manner for emerging mobile use cases, such as new gaming features and APIs. For example, the newest Adreno 4x series GPUs are designed to the DirectX11FL_11_2 specification, the same graphics specification as today's high-end desktop graphics cards and the latest gaming consoles.

To support next-gen features (beyond the newly announced Khronos OpenGL ES 3.1 APIs), Adreno 4x series GPUs extended their unified shader architecture by adding several completely new shaders (e.g., geometry and tessellation).

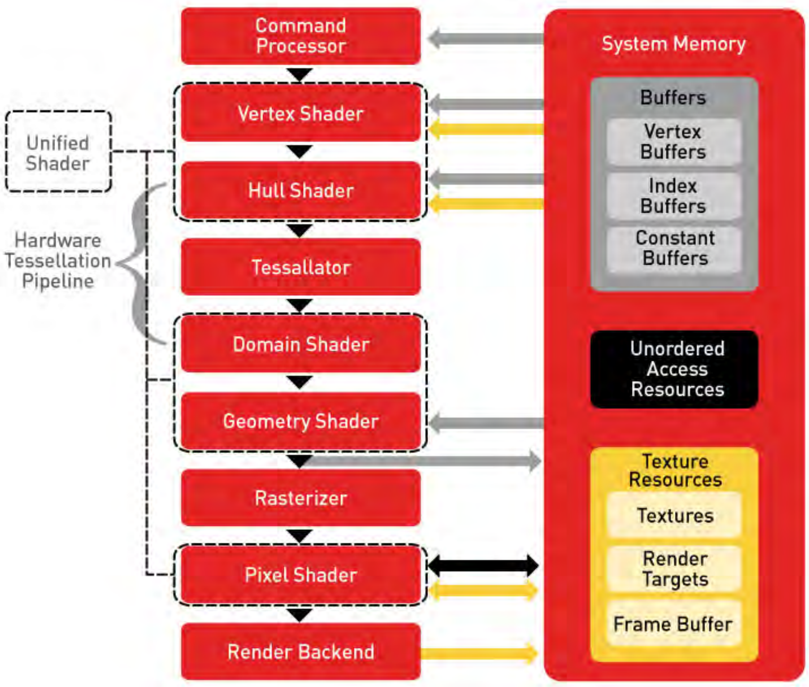


Figure 6: Adreno 4x series GPU's DirectX11 FL_12 based 3D hardware pipeline

The Adreno 420 GPU is one of the first commercial mobile GPUs to support dynamic hardware tessellation. Dynamic hardware tessellation is designed to help developers to provide greater detail for more visually realistic scenes in a manner that requires less memory bandwidth, lower power consumption, and lower overall memory footprint for the application.

Traditionally, in order for most 3D games to be visually immersive, the programmer must include a substantial amount of geometry detail per object. The denser the geometric mesh used to create an object in the scene (human or monster for example) of a game, the smoother and more realistic its surfaces will look. In traditional mobile GPUs, these additional geometry assets (required for improved visual realism) need to be stored in memory as part of the game binary package. These assets have to be copied or written by the CPU to the system memory and eventually have to be read into the GPU for further processing. These additional read and write operations could increase the memory bandwidth usage and power consumption to unsustainable levels (for console-quality games) on mobile devices.

Dynamic hardware tessellation helps solve this problem by allowing the GPU to generate additional geometry on-chip, without requiring additional data transfers from off-chip, system memory. Dynamic hardware tessellation can help significantly reduce bandwidth and power consumption. Another advantage is that the developer doesn't have to author and store these additional geometry details into their game package, which significantly reduces the memory footprint and the overall binary size of the game.

The image below shows the additional detail that tessellation provides to both the wireframe and the final image rendered. For this simple "Hornet" graphics scene, hardware tessellation delivers a bandwidth savings of ~360MB/s, and a memory footprint savings of ~20MB[9]. For larger games, the savings on memory footprint could be in GBs.

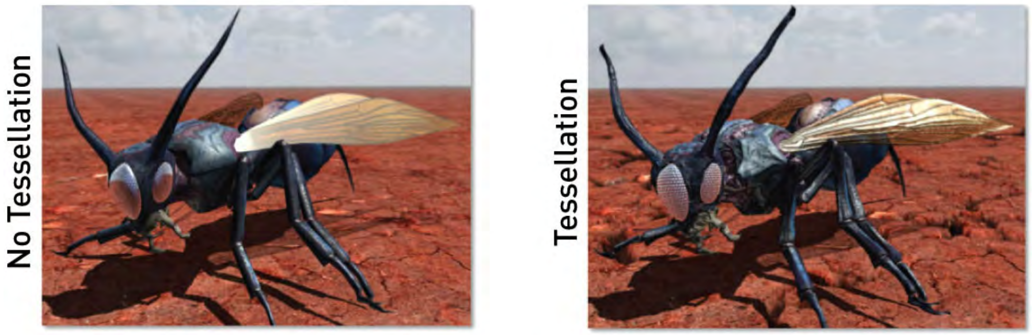


Figure 7: Tessellation (OFF: left, ON: right) provides more realistic and detailed graphics

By providing more detail, tessellation is designed to help significantly improve the gaming experience, not only for high resolution content, but also for low resolution content that needs to be upscaled.

The Adreno 4x series GPUs are a great example of QTI's commitment to bringing console-quality gaming up to 4K Ultra HD (4K) resolution to mobile devices.

> *"[Adreno 420 is] the most aggressive move in mobile graphics by any company, to add all the shader types, and HW tessellation, on top of what they did in Subdiv for Motorola, shows Qualcomm [Technologies] as the most committed mobile graphics supplier today. It really is bringing console-class graphics to mobile devices."*[10] — Jon Peddie, Jon Peddie Research.

Multi-screen experiences — high fidelity on your screens: Playing a game on a high-quality, high-resolution screen significantly enhances the gaming experience. The display engine enhances images, composites multiple images, and supports high resolution (both on-device and on external displays) of up to 4K.

For image enhancement, the display engine utilizes QTI's Qualcomm® TruPalette™ and Qualcomm® EcoPix™ feature sets, which include high quality post-processing algorithms for superior picture quality, including picture adjustment, color enhancement, contrast enhancement, scaling, sharpening, and power efficiency. For example, using the ecoPix sunlight visibility improvement technology, the display engine can use information from the device's light sensor to enhance the rendered game content to make it much more visible in bright conditions (e.g., outdoors on a sunny day) through tone correction.

[9] Numbers based on QTI internal testing

[10] Jon Peddie, "Qualcomm Moves to 4K with Snapdragon 805"

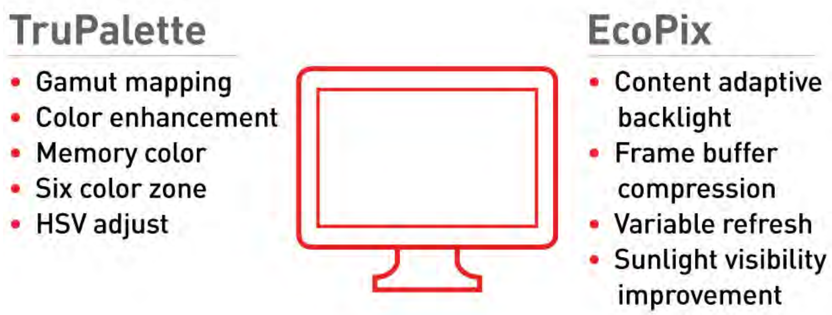


Figure 8: Snapdragon display engine TruPalette and EcoPix feature sets

Device-to-device connectivity further elevates the mobile gaming experience. External displays, such as a TV, are driven by either a wired (e.g., HDMI) or wireless connection (e.g., Miracast). It is the task of the display engine to efficiently process both types of connections. With the wireless display feature, you can send the contents of your mobile device screen to your smart TV screen. QTI supports wireless display standards, like Miracast, and is adding wireless display support to an increasing number of Android devices.

Cinema-quality sound — efficient audio processing in sync with the graphics: Just as sound is a huge part of the movie-going experience, it is also important in the world of immersive mobile gaming. Accordingly, QTI provides a comprehensive audio solution, including hardware and software, which offers high-fidelity audio and advanced features like 24-bit/96kHz play back and cinema-quality 7.1 surround sound audio.

The processing required to efficiently support these computationally complex audio features is primarily handled by QTI’s Hexagon™ DSP, which is custom designed for heavy signal processing tasks, like audio. The real-time processing capabilities of QTI’s Hexagon DSP are designed to keep the audio in sync with the graphics rendering. QTI has also worked with industry leaders, such as DTS and Dolby, to provide an optimal audio experience on Snapdragon processors, including support for headphone surround virtualization.

High-quality video — specialized video engine: Some games incorporate pre-rendered cut scenes to enhance the overall gaming experience. To save memory, these scenes are compressed into encoded video. To playback video, Snapdragon processors are designed to use a specialized video engine to decompress encoded videos at low power. For higher resolutions and more complex codecs, a specialized engine becomes even more important.

Figure 9 shows that running H.265 HEVC decode on a specialized video engine versus a CPU reduces power consumption, while still meeting the frame rate requirements[11].

[11] Estimated numbers based on QTI internal testing

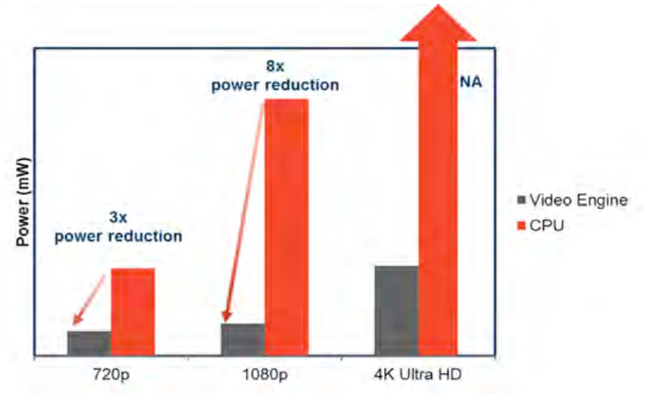


Figure 9: Power reduction by running HEVC decode on specialized video engine versus the CPU

Fast connectivity — multi-player gaming: Connected gaming experiences, like massively multiplayer online (MMO) games, require fast and reliable connectivity solutions with low latency where the difference between winning and losing often comes down to milliseconds. To achieve that, Snapdragon processors include connectivity technologies, such as 802.11ac Wi-Fi and 3G/4G LTE.

QTI has a long history of being an industry leader in advanced connectivity solutions. Its latest Snapdragon 810 processor continues the tradition, integrating a Cat 6 LTE Advanced multimode modem that is designed to support reliable communication at speeds of up to 300 Mbps. QTI optimizes the software stack for connectivity to achieve low client-server ping times on Snapdragon processors, so gamers can worry less about a slow connection and focus more on enjoying the game.

Responsive and accurate control — supporting multiple input methods: A good gaming experience requires an input method that gives precise control at low latency. There are multiple ways to control and interface to games, such as through a touch screen, device movement, a game controller, and gestures. Snapdragon processors have been designed to reduce latency and provide accurate control for these methods. For example, device movement generally is determined by processing data from motion sensors, such as the accelerometer or gyroscope. Sensor processing requires intensive signal processing, control processing, and real-time processing. QTI's specialized sensor engine, which excels at handling these tasks, is integrated in Snapdragon processors. As a result, the response time for device movement is faster — and at low power.

3.2 Efficient SoC architecture: smart management of system resources

An efficient SoC architecture is required in order to sustain console-quality gaming at high frame rates within the thermal and power constraints of mobile devices. As noted above, the majority of the processing engines within the SoC are running concurrently in a heterogeneous manner to efficiently process gaming tasks. Beyond heterogeneous computing, smartly managing the shared system resources, such as memory bandwidth, power budget, and thermal budget, is necessary to support sustained gameplay of a visually complex game.

Efficient memory bandwidth: While playing a game, the processing engines need to be fed data (starving a processing engine of data can lead to lower frame rates). To prevent bottlenecks, Snapdragon processors use advanced memory management techniques. The memory controller is designed to deliver high quality of service (QoS) for different throughput and latency requirements of different processing engines, while still maximizing the memory utilization. By minimizing the overhead associated with memory transactions, the memory controller helps increase memory utilization and minimizes power consumption.

Snapdragon processors are also designed to deploy smart caching mechanisms in many processing engines to help minimize the need for frequent DRAM access. For example, the Adreno GPU's tile-based architecture, which subdivides the graphics image into smaller tiles and renders them to the tile-buffer cache, helps minimize DRAM bandwidth and saves power.

Advanced power and thermal management technology: Drawing high power not only reduces battery life, it also releases excess heat. This will raise the skin temperature of the device, making it uncomfortable to hold. To reduce the system power, Snapdragon processors are designed to deploy sophisticated algorithms that manage power based on workload requirements. They support a wide range Dynamic Clock and Voltage Scaling (DCVS). DCVS dynamically varies the clock frequencies and voltages of the processing engines. For example, the Adreno 420 GPU now has more granular DCVS power control levels, so it can run most use cases at a nominal voltage state, thus saving power.

Several of the specialized engines have innovative, power-saving techniques. For example, the display engine also uses a proprietary compression scheme called Frame Buffer Compression (FBC), which compresses display data by up to 66% in a visually lossless manner—before transmission to the display panel.[12]

The Adreno 420 GPU also has an increased Z-reject rate for graphics rendering so that pixels that are not going to be visible (because they are blocked by a pixel on top) are not processed. An increased Z-reject rate means lower power per pixel—and improved performance.

4 Providing a consistent development platform at scale

To attract game developers, devices with Snapdragon processors provide an excellent opportunity with a consistent development environment.

4.1 Android gaming on Snapdragon: an excellent opportunity

As of January 2014, over 1,350 devices with Snapdragon processors had been announced or were commercially shipping. In addition, more than 525 new device designs are in development, and QTI is currently working with over 85 manufacturers worldwide. QTI was the #1 provider of Android smartphone AP shipments in 2013.[13]

[12] QTI internal metrics
[13] Strategy Analytics, April 2014

By optimizing their games on Snapdragon processors, developers have the opportunity to reach this very large and growing installed base of Android devices with Snapdragon processors that span across various price tiers and regions.

Figure 10: Devices with Snapdragon processors

Examples include iconic flagship Android-based devices like the Nexus 5 by Google, LG G Pro2, and Samsung Galaxy Note 3. Recently launched devices, including the HTC One (M8), One+ One, LG G3, Samsung Galaxy S5, and the Sony Xperia Z2, are powered by the Snapdragon 801 processor. Amazon Kindle Fire HDX 8.9 and Fire TV, based on Fire OS, are also powered by Snapdragon processors.

Refer to the smartphones and tablets websites for the latest devices powered by Snapdragon processors. With Adreno 4x series GPUs on the verge of being launched in commercial devices, Snapdragon processors continue to provide graphics and gaming technology leadership.

According to Jon Peddie Research, "Qualcomm [Technologies] is clearly the largest SoC supplier".[14] For Q2 2013 and as indicated in Figure 11, QTI was the single largest GPU supplier for personal devices, which includes smartphones, tablets, and dedicated handheld game consoles.[14]

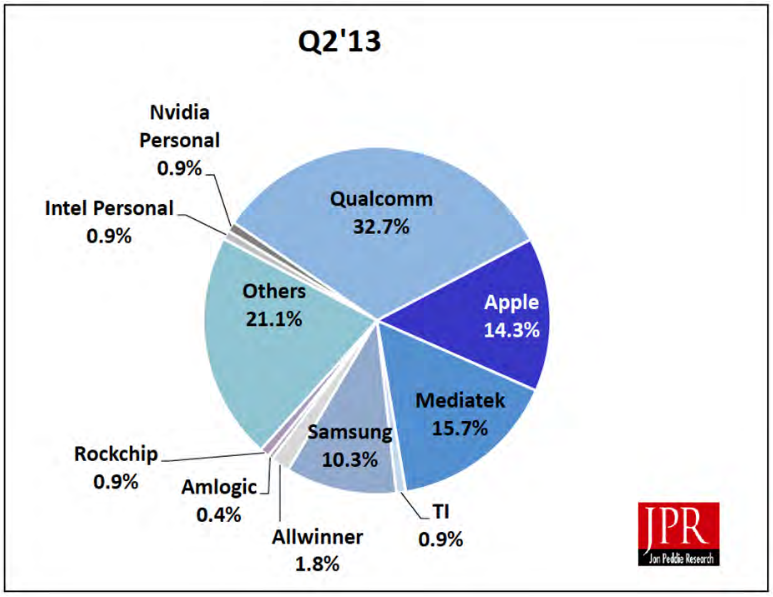


Figure 11: Share of personal systems SoC suppliers for Q2 2013

[14] Jon Peddie Research, Oct. 2013, "Mobile Devices and the GPUs inside"

4.2 Scalable architecture: consistent gaming experiences across various tiers

Consumers expect immersive gaming experiences whether they own a premium or an entry-level mobile device. Snapdragon processors are designed to support great gaming experiences across the spectrum of device tiers. Apart from offering backwards-compatible software between different tiers, Snapdragon processors with the scalable Adreno GPU are designed to provide consistent features and APIs across tiers. Using APIs as an example, Adreno 3x series GPUs (and above) support OpenGL ES 3.0. Having common support of APIs and features ranging from entry-level to premium devices helps make it easier for developers to introduce next-generation features to mass audiences, without having to custom design for each tier.

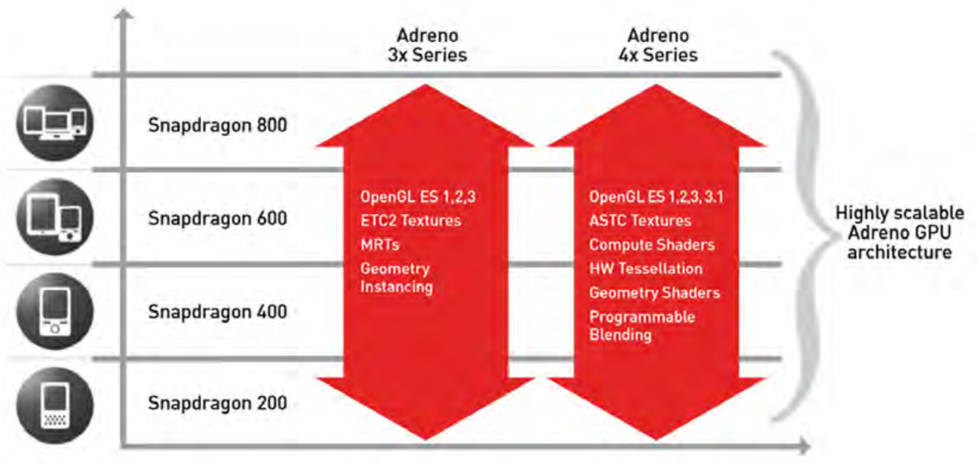


Figure 12: Snapdragon processors with the scalable Adreno GPU are designed to provide consistent features across tiers

Snapdragon processors are software compatible, helping both OEMs and developers to efficiently invest and develop across multiple device types and tiers. With a consistent software stack, including graphics drivers, devices with Snapdragon processors are designed to provide a consistent and optimized gaming platform across various tiers.

5 Unlocking the potential of Snapdragon processors for developers

QTI and its affiliates have a long history of technology leadership in supporting the mobile gaming ecosystem, dating back to 2001 with the Brew™ platform. This breakthrough development platform allowed native C/C++ games to be written “closer to the metal,” supporting higher-quality and higher-performance games, and helped usher in some of the world’s first 3D games in mobile. Brew was also one of the earliest monetization platforms for mobile applications, paying out more than $3 billion to developers[15].

[15] https://www.brewmp.com/about

The introduction of 3G in the early 2000s enabled faster downloads of larger games, multi-player gaming, and more robust client-server-based game designs. In addition, Qualcomm Ventures also helped fund mobile gaming pioneers such as JAMDAT Mobile and more recently invested in companies such as Bluestacks, Gaikai, Grand Cru, Playdek, Playnery, and TabTale.

It takes more than just great hardware to advance the mobile gaming industry. To support game developers in producing immersive games, QTI provides comprehensive tools, extensive support, and advanced technologies.

5.1 Supporting developers with comprehensive tools

QTI supports developers in unlocking the performance and advanced features of Snapdragon processors by not only offering comprehensive development tools, but by also working closely with the gaming ecosystem.

Game engine optimization — access to the latest features and reduced time-to-market: QTI works closely with the world's leading third-party game engine providers, such as Unity Technologies and Epic Games, to optimize their engines for Snapdragon processors. Additionally, QTI helps expose the latest graphics APIs and advanced features of Adreno GPUs to developers through these engines. For example, QTI worked closely with Epic Games to optimize Unreal Engine's advanced lighting and post processing pipeline for Snapdragon processors.

> *"Bringing Unreal Engine 4 PC AAA graphics to mobile has enabled us to do content that we've never been able to do before. A big advantage has been our close relationship with Qualcomm [Technologies]. Without that close relationship we wouldn't have been able to reach this point."*
>
> Niklas "Smedis" Smedberg
> Senior Engine Programmer – Mobile Graphics
> Epic Games

Figure 13: Unreal Engine 4 demo showcasing optimized post-processing on Adreno GPU

QTI also worked with Unity Technologies to help accelerate the support of new features, making the Unity Engine one of the first gaming engines to support important OpenGL ES 3.0 features like ETC2 textures, multiple render targets (MRT), and transform feedback in Unity 4.x. Unity and QTI continue this work by helping bring new Unity 5.0 innovative features like physically based shading to Snapdragon processors.

"We're excited to work with an innovative global mobile technology provider such as Qualcomm Technologies and support their incredible Snapdragon processors, which are at the heart of many Android and Windows mobile devices across the globe."

David Hegalson
CEO
Unity Technologies

Figure 14: Unity 5 demo showcasing physically based shading technique on Adreno GPU

By using these popular game engines, which have been optimized for Snapdragon processors, developers can focus their time on content creation.

In addition, QTI also works closely with developers to help optimize their proprietary in-house game engines so that they run well on Snapdragon processors. In turn, these developers can then utilize their optimized engines across their internal studios when developing mobile games.

Game development tools — developing, debugging, and optimizing games: For those who are programming directly to the hardware and not using a third-party game engine, QTI has created (and made available) powerful game development tools, which are designed to help debug and optimize games, while reducing time-to-market. These tools include the Adreno SDK, Adreno Profiler, and Trepn™ Profiler.

The Adreno SDK includes tools, emulators, libraries, documentation, samples, and tutorials. The desktop OpenGL ES emulator is designed to eliminate the need to have a device early in the development process. Adreno SDK supports the most common APIs such as OpenGL ES, DirectX, and OpenCL. In addition, the Adreno SDK contains several time-saving utilities and over 100 samples and tutorials, including 50 advanced shader effects.

The Adreno Profiler provides developers with detailed GPU utilization analysis to help them optimize their games for faster frame rates, smoother rendering, and longer battery life. It works on commercial devices without making changes within a game, device drivers, or builds, which helps further save development time and reduce setup complexity.

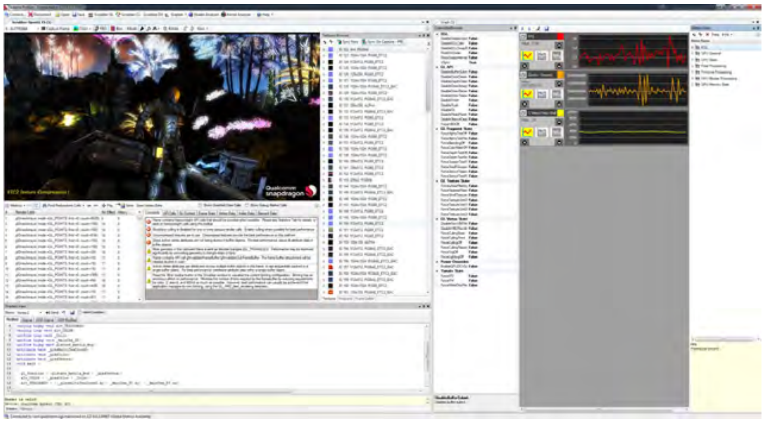


Figure 15: Adreno Profiler helps optimize games

The Trepn Profiler is designed to integrate directly into a developer's workflow, which helps him or her see how much power a game is consuming, pinpoint issues and quickly resolve them. Consumers expect all day battery life, consistent performance, and a device that stays cool. In fact, battery life has become an extremely important decision factor for consumers when buying smartphones. Managing power consumption and remaining within the thermal envelope of mobile devices are key development considerations. Exceeding the thermal envelope will not only heat up the device, but also throttle the game's frame rate, which negatively affects gameplay.

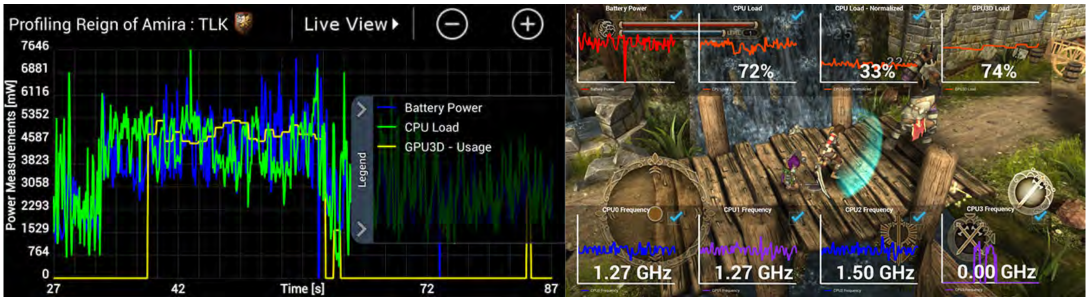


Figure 16: Trepn Profiler provides diagnostic views, such as battery power, CPU load, and GPU usage

Figure 17: BombSquad by Eric Froemling

"*I would definitely recommend developers work with Qualcomm [Technologies] and use the Snapdragon tools. I know they've been a big go-to for me*."

Eric Froemling

Development devices — optimizing games for real hardware: Mobile development platforms (MDP) powered by certain Snapdragon processors in the form of tablets and smartphones are currently available to game developers prior to the launch of commercial OEM devices. These MDPs provide early access to new features of Snapdragon processors and are designed to allow developers to optimize their games in advance of the commercial launch of devices with Snapdragon processors. Devices with Snapdragon processors are available across tiers, and developers can select from a wide range of existing commercial devices to complete final testing and validation of their games.

Figure 18: Snapdragon Mobile Development Platform

"*Developing on the kits is a great help for us as a development team because it allows us to address a wide installed base since many devices use the same Snapdragon chipset*."

Francois Bodson
Studio Manager
Ubisoft Paris / Mobile

Figure 19: Assassin's Creed Pirates by Ubisoft Entertainment

5.2 Supporting the developer community

QTI can provide game developers with technical support to shorten time-to-market and improve game quality. QTI also can provide business and marketing support to help developers promote their games and facilitate potential business opportunities.

Technical support for developers — providing expert game optimization: Such technical support is provided via QTI's internal game studio, which is comprised of developers with comprehensive prior work experience in console, PC, and mobile gaming. Support activities range from tools, training, technical feedback on optimizations and builds (from alpha to beta to release candidates), and new feature support.

To take advantage of the capabilities of Snapdragon processors and to extend, enhance, and differentiate their games, QTI can help developers add full screen post-processing effects, OpenGL ES 3.0/3.1 features, and more. There is also a developer forum where developers can ask questions and engage with the QTI team and others in the Snapdragon developer community.

"*The Qualcomm® Snapdragon™ processor has allowed us to create a better experience in our game Gates of Osiris because it's allowed us to develop faster, push our graphics, and understand how to optimize our game to be the best it can possibly be*."

Brian McRae
CEO
Fenix Fire

Figure 20: QTI worked closely with Fenix Fire to optimize Gates of Osiris on Snapdragon

Business and marketing support for developers — facilitating business opportunities: QTI is engaged with some of the world's leading publishers and developers, from traditional console and PC to indies. The list includes companies such as EA, Gameloft, Activision Blizzard, Ubisoft, Square Enix, Mojang, and Take-Two Interactive. QTI also facilitates introductions for engagements with OEMs and carriers across the global gaming ecosystem.

Throughout the year, QTI has a presence at several key mobile and gaming tradeshows, where it helps developers obtain exposure to show attendees, press, and OEMs. In the past year, QTI participated in shows such as CES, MWC, GDC, E3, SIGGRAPH, Unite, and many more, reaching hundreds of thousands of attendees. In addition, QTI often features games on the latest Snapdragon development platforms and commercial devices with Snapdragon processors.

Figure 21: Showcasing games at the QTI booth during CES 2014 (left); GDC 2013 (right)

5.3 Next-generation gaming experiences

QTI continues to push the mobile gaming industry forward by developing new and innovative technologies.

QTI actively develops tech demos to showcase the latest Snapdragon features and advanced capabilities. The "Swimmer" demo is a recent example of how developers can utilize OpenGL ES 3.0 APIs for advanced rendering techniques. The demo shows advanced skin rendering using subsurface scattering techniques running optimally at 60 frames per second, at 2K resolutions.

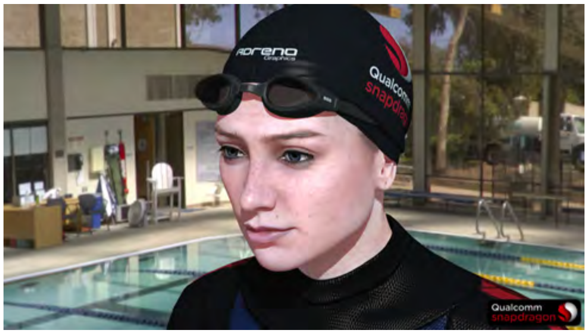


Figure 22: QTI "Swimmer" technology demo

To help further advance the evolution of gaming, QTI and its affiliates have developed (and continue to develop) innovative technologies that support ever-more advanced capabilities and features including vision-based augmented reality (Qualcomm® Vuforia™), proximal peer-to-peer networking (AllJoyn™), biometrics, and more. For example, Vuforia supports the Smart Terrain™ feature, which is designed to provide real-time 3D mapping of a physical play area, including intelligent interaction with objects and surfaces. With this technology, gamers can create user-generated, playable content from physical objects by using their mobile device as a level editor. Another example of a promising new technology that can be utilized by game designers is biometrics data from a wearable. By capturing measurements from pulse and blood pressure sensors, gameplay can be adjusted accordingly to either reward or penalize a player.

6 Conclusion

Consumer demand for gaming on mobile devices is growing rapidly, generating strong momentum and opportunity. The Android platform is thriving and offers game developers a growing opportunity to generate revenue on a global scale. QTI is helping the mobile gaming industry to take advantage of this opportunity by solving three key challenges for developers:

- To support immersive gaming experiences within the power and thermal constraints of mobile devices, QTI creates low-power SoCs by taking a heterogeneous computing approach and designing an efficient SoC architecture.
- To efficiently address a sizeable portion of the Android gaming segment with minimal code variations, QTI's scalable architecture is designed to provide a consistent development platform across tiers.
- To take advantage of Snapdragon processors' capabilities and to help reduce development time and costs, QTI offers developers a comprehensive set of tools, support, and advanced technologies.

Snapdragon processors are purpose-built and custom designed with mobile in mind, supporting gaming experiences from simple casual games to console-quality games, so consumers can play longer and recharge less. This is yet another example of how QTI is once again re-inventing the mobile world we live in.

References

Developer tools and technologies:

1. Adreno SDK: https://developer.qualcomm.com/mobile-development/maximize-hardware/mobile-gaming-graphics-adreno/tools-and-resources
2. Adreno Profiler: https://developer.qualcomm.com/mobile-development/maximize-hardware/mobile-gaming-graphics-adreno/tools-and-resources
3. Trepn Profiler: https://developer.qualcomm.com/mobile-development/increase-app-performance/trepn-profiler
4. Mobile Development Platforms: https://developer.qualcomm.com/mobile-development/development-devices/snapdragon-mobile-development-platform-mdp
5. Snapdragon SDK: https://developer.qualcomm.com/mobile-development/add-advanced-features/snapdragon-sdk-android
6. Snapdragon LLVM: https://developer.qualcomm.com/mobile-development/increase-app-performance/snapdragon-llvm-compiler-android
7. Snapdragon Performance Visualizer: https://developer.qualcomm.com/mobile-development/increase-app-performance/snapdragon-performance-visualizer

Developer support:

1. Questions/feedback/contact QTI: https://developer.qualcomm.com/contact. Use the "Mobile Gaming & Graphics Optimization (Adreno)" recipient option.
2. Adreno developer forum: https://developer.qualcomm.com/forums/qdevnet-forums/mobile-gaming-graphics-adreno
3. Developer guide: https://developer.qualcomm.com/download/adreno-sdk.zip

SoC technologies:

1. FlexRender: http://www.qualcomm.com/media/videos/flexrender-rendered-useful

Advanced technologies:

1. Vuforia: https://developer.qualcomm.com/mobile-development/add-advanced-features/augmented-reality-vuforia
2. AllJoyn: https://developer.qualcomm.com/mobile-development/create-connected-experiences/peer-peer-alljoyn

Snapdragon devices:

1. Smartphones: http://www.qualcomm.com/snapdragon/smartphones
2. Tablets: http://www.qualcomm.com/snapdragon/tablets-pcs

```
`include "header.v"
//---------------------------------------------------------------------------
//
// $Id: //depot/r400/devel/parts_lib/src/gfx/sq/sq.v#62 $
//
// $Change: 44294 $
//
// Copyright:   Trade secret of ATI Technologies, Inc.
//              © Copyright 2001-2002, ATI Technologies, Inc., (unpublished)
//
//              All rights reserved.  This notice is intended as a precaution against
//              inadvertent publication and does not imply publication or any waiver
//              of confidentiality.  The year included in the foregoing notice is the
//              year of creation of the work.
//
//---------------------------------------------------------------------------
`include "register_addr.v"
`include "sq_reg.v"
`include "sq_defs.v"
`include "../sq_reg_old.v"

module sq (/*AUTOARG*/
  // Outputs
  SQ_SC_free_buff, SQ_SC_dec_cntr_cnt, SQ_SP_interp_prim_type,
  SQ_SP_interp_mode, SQ_SP_interp_ijline, SQ_SP_interp_buff_swap,
```

```
SQ_SP_interp_gen_i0, SQ_SP_interp_valid, SQ_SX_interp_flat_vtx,
SQ_SX_interp_flat_gouraud, SQ_SX_interp_cyl_wrap, SQ_SX_pc_ptr0,
SQ_SX_pc_ptr1, SQ_SX_pc_ptr2, SQ_SX_rt_sel, SQ_VGT_rtr,
SQ_SP_vsr_data, SQ_SP_vsr_double, u0_SQ_SP_vsr_valid,
u1_SQ_SP_vsr_valid, u2_SQ_SP_vsr_valid, u3_SQ_SP_vsr_valid,
SQ_SP_vsr_vu_valid, SQ_SP_vsr_read, SQ_CP_vs_event,
SQ_CP_vs_eventid, SQ_CP_ps_event, SQ_CP_ps_eventid, SQ_TP_send,
SQ_TP_instr, SQ_TP_const, SQ_TP_gpr_phase, SQ_TP_gpr_wr_addr,
SQ_TP_thread_id, u0_SQ_TP_lod_correct, u0_SQ_TP_pix_mask,
u1_SQ_TP_lod_correct, u1_SQ_TP_pix_mask, u2_SQ_TP_lod_correct,
u2_SQ_TP_pix_mask, u3_SQ_TP_lod_correct, u3_SQ_TP_pix_mask,
SQ_SP_gpr_wr_addr, u0_SQ_SP_gpr_wr_en, u1_SQ_SP_gpr_wr_en,
u2_SQ_SP_gpr_wr_en, u3_SQ_SP_gpr_wr_en, SQ_SP_gpr_rd_addr,
SQ_SP_gpr_rd_en, SQ_SP_gpr_phase_mux, SQ_SP_channel_mask,
u0_SQ_SP_pix_mask, u1_SQ_SP_pix_mask, u2_SQ_SP_pix_mask,
u3_SQ_SP_pix_mask, SQ_SP_gpr_input_mux, SQ_SP_auto_count,
SQ_SP_instruct_start, SQ_SP_instruct, SQ_SP_const, SQ_SP_stall,
SQ_SP_exporting, SQ_SP_exp_id, u0_SQ_SP_exp_pvalid,
u1_SQ_SP_exp_pvalid, u2_SQ_SP_exp_pvalid, u3_SQ_SP_exp_pvalid,
SQ_SX_pc_wr_addr, SQ_SX_pc_wr_en, SQ_SX_pc_channel_mask,
SQ_SX_exp_type, SQ_SX_exp_number, SQ_SX_exp_id, SQ_SX_exp_valid,
SQ_SX_exp_state, SQ_SX_free_done, SQ_SX_free_id, SQ_RBB_rd,
SQ_RBB_rs, SQ_RBBM_nrtrtr, SQ_RBBM_rtr, SQ_RBBM_cntx0_busy,
SQ_RBBM_cntx17_busy, SQ_CG_threshold_hi, SQ_CG_threshold_lo,
// Inputs
```

```
  SC_SQ_data, SC_SQ_valid, VGT_SQ_vsisr_data,
  VGT_SQ_vsisr_continued, VGT_SQ_event, VGT_SQ_end_of_vtx_vect,
  VGT_SQ_indx_valid, VGT_SQ_state, VGT_SQ_send, TP_SQ_type,
  TP_SQ_data_rdy, TP_SQ_thread_id, TP_SQ_fetch_stall,
  u0_SX_SQ_exp_count_rdy, u0_SX_SQ_exp_pos_avail,
  u0_SX_SQ_exp_buf_avail, u1_SX_SQ_exp_count_rdy,
  u1_SX_SQ_exp_pos_avail, u1_SX_SQ_exp_buf_avail,
  u0_SP_SQ_const_addr, u0_SP_SQ_valid, u1_SP_SQ_const_addr,
  u1_SP_SQ_valid, u2_SP_SQ_const_addr, u2_SP_SQ_valid,
  u3_SP_SQ_const_addr, u3_SP_SQ_valid, u0_SP_SQ_kill_vect,
  u1_SP_SQ_kill_vect, u2_SP_SQ_kill_vect, u3_SP_SQ_kill_vect,
  RBBM_a, RBBM_wd, RBBM_we, RBBM_re, RBB_rs, RBB_rd, RBBM_be,
  RBBM_SQ_soft_reset, CG_SQ_pm_enb, sclk_global, srst
  );

  //-------------------------------------------------------------------------------------------
  // SC-SQ IJ control interface
  //-------------------------------------------------------------------------------------------
  input [53:0]  SC_SQ_data;
  input [0:0]   SC_SQ_valid;
  /*
  // 1st cycle fields
  input [0:0]  SC_SQ_event;
  input [3:0]  SC_SQ_event_id;
  input [2:0]  SC_SQ_pc_dealloc;
```

```
input [0:0]  SC_SQ_new_vector;
input [3:0]  SC_SQ_quad_mask;
input [0:0]  SC_SQ_end_of_prim;
input [2:0]  SC_SQ_state_id;
input [15:0] SC_SQ_pix_mask;
input [2:0]  SC_SQ_prim_type;
input [10:0] SC_SQ_pc_ptr0;
// 2nd cycle fields
input [10:0] SC_SQ_pc_ptr1;
input [10:0] SC_SQ_pc_ptr2;
input [23:0] SC_SQ_lod_correct;
*/

output [0:0] SQ_SC_free_buff;
output [0:0] SQ_SC_dec_cntr_cnt;


//----------------------------------------------------------------------------------------------------
// SQ-SP/SX Interpolator Bus interface
//----------------------------------------------------------------------------------------------------
output [2:0]  SQ_SP_interp_prim_type;
output [0:0]  SQ_SP_interp_mode;
output [1:0]  SQ_SP_interp_ijline;
output [0:0]  SQ_SP_interp_buff_swap;
output [0:0]  SQ_SP_interp_gen_i0;
output [0:0]  SQ_SP_interp_valid;

output [1:0]  SQ_SX_interp_flat_vtx;
output [0:0]  SQ_SX_interp_flat_gouraud;
output [3:0]  SQ_SX_interp_cyl_wrap;


//---------------------------------------------------------------------------
// SQ-SX Parameter Cache Read control
//---------------------------------------------------------------------------
output [10:0] SQ_SX_pc_ptr0;
output [10:0] SQ_SX_pc_ptr1;
output [10:0] SQ_SX_pc_ptr2;
output [0:0]  SQ_SX_rt_sel;


//------------------------------------------------------------------------
// VGT-SQ Vertex Interface
//------------------------------------------------------------------------
input [95:0] VGT_SQ_vsisr_data;
input [0:0]    VGT_SQ_vsisr_continued;
input [0:0]    VGT_SQ_event;
input [0:0]    VGT_SQ_end_of_vtx_vect;
input [0:0]    VGT_SQ_indx_valid;
input [2:0]    VGT_SQ_state;
```

```
input [0:0]    VGT_SQ_send;

output [0:0]  SQ_VGT_rtr;



//------------------------------------------------------------------------
// SQ-SP Vertex Interface
//------------------------------------------------------------------------
output [95:0] SQ_SP_vsr_data;
output [0:0] SQ_SP_vsr_double;
output [0:0] u0_SQ_SP_vsr_valid;
output [0:0] u1_SQ_SP_vsr_valid;
output [0:0] u2_SQ_SP_vsr_valid;
output [0:0] u3_SQ_SP_vsr_valid;
output [3:0] SQ_SP_vsr_vu_valid;
output [0:0] SQ_SP_vsr_read;


//------------------------------------------------------------------------
// SQ-CP Event Status
//------------------------------------------------------------------------
output [0:0]  SQ_CP_vs_event;
output [1:0]  SQ_CP_vs_eventid;
output [0:0]  SQ_CP_ps_event;
output [1:0]  SQ_CP_ps_eventid;
```

```


//---------------------------------------------------------------------------
// SQ-TP (Texture Pipe) (part of this interface is broadcast)
//-----------------------------------------------------------------------------
output [0:0]  SQ_TP_send;
output [17:0] SQ_TP_instr;
output [47:0] SQ_TP_const;
output [1:0]  SQ_TP_gpr_phase;
/*
 send these over 4 cycles:
output [6:0]  SQ_TP_gpr_wr_addr;
output [0:0]  SQ_TP_type;
output [5:0]  SQ_TP_thread_id;
output [0:0]  SQ_TP_end_of_group;
*/
output [1:0]  SQ_TP_gpr_wr_addr;
output [1:0]  SQ_TP_thread_id;

output [5:0]  u0_SQ_TP_lod_correct;
output [3:0]  u0_SQ_TP_pix_mask;
output [5:0]  u1_SQ_TP_lod_correct;
output [3:0]  u1_SQ_TP_pix_mask;
output [5:0]  u2_SQ_TP_lod_correct;
output [3:0]  u2_SQ_TP_pix_mask;
```

```
output [5:0]  u3_SQ_TP_lod_correct;
output [3:0]  u3_SQ_TP_pix_mask;

input [0:0]   TP_SQ_type;
input [0:0]     TP_SQ_data_rdy;
input [5:0]     TP_SQ_thread_id;

input  [0:0]  TP_SQ_fetch_stall;



//-----------------------------------------------------------
// SQ-SP GPR control Interface
//-----------------------------------------------------------
output [6:0] SQ_SP_gpr_wr_addr;
output [0:0] u0_SQ_SP_gpr_wr_en;
output [0:0] u1_SQ_SP_gpr_wr_en;
output [0:0] u2_SQ_SP_gpr_wr_en;
output [0:0] u3_SQ_SP_gpr_wr_en;
output [6:0] SQ_SP_gpr_rd_addr;
output [0:0] SQ_SP_gpr_rd_en;
output [1:0] SQ_SP_gpr_phase_mux;
output [3:0] SQ_SP_channel_mask;

output [3:0] u0_SQ_SP_pix_mask;
output [3:0] u1_SQ_SP_pix_mask;
output [3:0] u2_SQ_SP_pix_mask;
output [3:0] u3_SQ_SP_pix_mask;

output [1:0]  SQ_SP_gpr_input_mux;
output [11:0] SQ_SP_auto_count;



//----------------------------------------------------------------
// SQ-SP : Instruction interface
//----------------------------------------------------------------
output [0:0] SQ_SP_instruct_start;
output [20:0] SQ_SP_instruct;
output [127:0] SQ_SP_const;
output [0:0] SQ_SP_stall;

output [0:0] SQ_SP_exporting;
output [0:0] SQ_SP_exp_id;
output [3:0] u0_SQ_SP_exp_pvalid;
output [3:0] u1_SQ_SP_exp_pvalid;
output [3:0] u2_SQ_SP_exp_pvalid;
output [3:0] u3_SQ_SP_exp_pvalid;



//---------------------------------------------------------------------------
// SQ-SX Parameter Cache Write control
```

```
//---------------------------------------------------------------------------
output [6:0]  SQ_SX_pc_wr_addr;
output [0:0]  SQ_SX_pc_wr_en;
output [3:0]  SQ_SX_pc_channel_mask;



//---------------------------------------------------------------------------
// SQ-SX Export Control Bus
//---------------------------------------------------------------------------
output [1:0] SQ_SX_exp_type;
output [1:0] SQ_SX_exp_number;
output [0:0] SQ_SX_exp_id;
output [0:0] SQ_SX_exp_valid;
output [2:0] SQ_SX_exp_state;

output [0:0] SQ_SX_free_done;
output [0:0] SQ_SX_free_id;



//-----------------------------------------------------------------------
// SX-SQ Output File control
//-----------------------------------------------------------------------
input [0:0]    u0_SX_SQ_exp_count_rdy;
input [0:0]    u0_SX_SQ_exp_pos_avail;
input [6:0]    u0_SX_SQ_exp_buf_avail;
```

```

input [0:0]    u1_SX_SQ_exp_count_rdy;
input [0:0]    u1_SX_SQ_exp_pos_avail;
input [6:0]    u1_SX_SQ_exp_buf_avail;



//-----------------------------------------------------------------------------------
// SP-SQ: Constant Address Load
//---------------------------------------------------------------------------
input [35:0] u0_SP_SQ_const_addr;
input [0:0]    u0_SP_SQ_valid;
input [35:0] u1_SP_SQ_const_addr;
input [0:0]    u1_SP_SQ_valid;
input [35:0] u2_SP_SQ_const_addr;
input [0:0]    u2_SP_SQ_valid;
input [35:0] u3_SP_SQ_const_addr;
input [0:0]    u3_SP_SQ_valid;

//---------------------------------------------------------------------
// SQ-SP Kill Vector Load
//-----------------------------------------------------------------
input [3:0]      u0_SP_SQ_kill_vect;
input [3:0]      u1_SP_SQ_kill_vect;
input [3:0]      u2_SP_SQ_kill_vect;
input [3:0]      u3_SP_SQ_kill_vect;
```

```


//---------------------------------------------------------------------------------
// SQ-RBBM
//-------------------------------------------------------------------------------------
output [31:0] SQ_RBB_rd;
output [0:0] SQ_RBB_rs;
output [0:0] SQ_RBBM_nrtrtr;
output [0:0] SQ_RBBM_rtr;
output [0:0] SQ_RBBM_cntx0_busy;
output [0:0] SQ_RBBM_cntx17_busy;


//-----------------------------------------------------------------------
// RBBM_SQ
//---------------------------------------------------------------------
input [14:0] RBBM_a;
input [31:0] RBBM_wd;
input       RBBM_we;
input            RBBM_re;
input            RBB_rs;
input [31:0] RBB_rd;
input [3:0]    RBBM_be;
input [0:0]    RBBM_SQ_soft_reset;

```

```

  //----------------------------------------------------------------------
  // MISC
  //----------------------------------------------------------------------

  input  [0:0] CG_SQ_pm_enb;
  output [0:0] SQ_CG_threshold_hi;
  output [0:0] SQ_CG_threshold_lo;

  input [0:0] sclk_global;
  input [0:0] srst;


  //
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++
  // ---------------------------------------------------------------------------
  // -- Module Instantiations --
  // ---------------------------------------------------------------------------
  //
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++

  parameter HI = 1'b1;
  parameter LO = 1'b0;

  // - wires for local registers
```

```

  wire [(`SQ_PROGRAM_REG_COUNT_SHADING_PS_NUM_REG_SIZE * 8) - 1: 0]
       ps_num_reg_set;
  wire [(`SQ_PROGRAM_REG_COUNT_SHADING_VS_NUM_REG_SIZE * 8) - 1: 0]
       vs_num_reg_set;
  wire [(`SQ_PROGRAM_REG_COUNT_SHADING_PARAM_SHADE_SIZE * 8) - 1: 0]
       param_shade_set;
  wire [(`SQ_INST_STORE_MANAGMENT_INST_BASE_VTX_SIZE) - 1: 0]
       inst_base_vtx;
  wire [(`SQ_INST_STORE_MANAGMENT_INST_BASE_PIX_SIZE) - 1: 0]
       inst_base_pix;
  wire [(`SQ_WRAPPING_1_REG_SIZE * 8) - 1: 0]
       sq_wrapping_1_set;
  wire [(`SQ_WRAPPING_0_REG_SIZE * 8) - 1: 0]
       sq_wrapping_0_set;
  wire [(`SQ_SAMPLING_MODE_SAMPLING_MODE_SIZE * 8) - 1: 0]
       sampling_mode_set;
  wire [(`SQ_VS_EXPORT_COUNT_COUNT7_SIZE * 8) - 1: 0]
       vs_exp_count7_set;
  wire [(`SQ_IMPORTS_EXPORTS_GEN_INDEX_SIZE * 8) - 1: 0]
       gen_index_set;
  wire [(`SQ_IMPORTS_EXPORTS_PARAM_GEN_I0_SIZE * 8) - 1: 0]
       param_gen_i0_set;
  wire [(`SQ_PROGRAM_CNTL_VS_RESOURCE_SIZE * 8) -1: 0]
       vs_resource_set;
  wire [(`SQ_PROGRAM_CNTL_PS_RESOURCE_SIZE * 8) - 1: 0] ps_resource_set;

  wire [(`SQ_PROGRAM_CNTL_VS_EXPORT_COUNT_SIZE * 8) - 1: 0]
vs_export_count_set;
  wire [(`SQ_PROGRAM_CNTL_VS_EXPORT_MODE_SIZE * 8) - 1: 0]
vs_export_mode_set;
```

```
  wire [(`SQ_PROGRAM_CNTL_PS_EXPORT_MODE_SIZE * 8) - 1: 0]
ps_export_mode_set;
  wire [(`SQ_PS_PROGRAM_BASE_SIZE * 8) - 1: 0] ps_program_base_set;
  wire [(`SQ_PS_PROGRAM_BASE_SIZE * 8) - 1: 0] vs_program_base_set;

  // --------------------------------------------------------------------
  // --------------------------------------------------------------------
  // -- Vertex Input Control --
  // --------------------------------------------------------------------
  // --------------------------------------------------------------------

  // - interconnect wires

  //wire [8*6-1:0] vs_num_reg_set;  //  connected  to
SQ_PROGRAM_REG_COUNT_SHADING.VS_NUM_REG (6 bits)
  //wire [8*1-1:0] gen_index_set;   // connected to SQ_IMPORTS_EXPORTS.GEN_INDEX
(1 bit)

  wire [0:0] vtx_alloc_req;
  wire [5:0] vtx_alloc_space;
  wire [0:0] vtx_alloc_ack;

  wire [0:0] vtx_dealloc_req;
  wire [5:0] vtx_dealloc_space;
  wire [0:0] vtx_dealloc_ack;

```

```
  wire [0:0] vtx_write_gnt;
  wire [0:0] vtx_write_req;
  wire [0:0] vtx_write_busy;

  wire [6:0] gpr_base;

  wire [00:0] vism_ctl_pkt_rts;
  wire [11:0] vism_instr_ptr;
  wire [63:0] vism_valid_bits;
  wire [06:0] vism_gpr_base;
  wire [02:0] vism_context_id;
  wire [00:0] vism_resource;
  wire [00:0] vism_first_thread;
  wire [00:0] vtb_rtr;

  wire [6:0] vi_gpr_wr_addr;
  wire [0:0] vi_gpr_wr_en;
  wire [7:0] acs_context_valid;


  // --------------------------------
  // -- Vertex Input State Machine --
  // --------------------------------

  sq_vism
```

```
u_sq_vism
(
  // VGT Interface
  .i_vgt_vsisr_data         (VGT_SQ_vsisr_data),
  .i_vgt_vsisr_double       (VGT_SQ_vsisr_continued),
  .i_vgt_end_of_vector      (VGT_SQ_end_of_vtx_vect),
  .i_vgt_indx_valid         (VGT_SQ_indx_valid),
  .i_vgt_vsisr_state  (VGT_SQ_state),
  .i_vgt_send               (VGT_SQ_send),
  .o_vgt_rtr          (SQ_VGT_rtr),

  // SP Interface
  .o_sp_vsr_data            (SQ_SP_vsr_data),
  .o_sp_vsr_double          (SQ_SP_vsr_double),
  .o_sp_vsr_valid           ({u3_SQ_SP_vsr_valid,                  u2_SQ_SP_vsr_valid,
u1_SQ_SP_vsr_valid, u0_SQ_SP_vsr_valid}),
  .o_sp_vsr_vu_valid        (SQ_SP_vsr_vu_valid),
  .o_sp_vsr_read            (SQ_SP_vsr_read),

  // to output mux (currently called ais_output), then to SP
  .o_v_gpr_addr             (vi_gpr_wr_addr),    // VISM gpr write address
  .o_v_gpr_we               (vi_gpr_wr_en),             // VISM gpr write enable

  // local register inputs
  .i_vs_num_reg          (vs_num_reg_set),
  .i_gen_index          (gen_index_set),
```

```
  .i_vs_base_set         (vs_program_base_set),
  //.i_vs_resource_set     (vs_resource_set),
  .i_vs_resource_set     (8'b11111111),

  // input arbiter interface
  .o_v_gpr_wrt_req           (vtx_write_req),
  .i_v_gpr_wrt_grant         (vtx_write_gnt),
  .o_vism_busy               (vtx_write_busy),

  // gpr alloc Interface
  .o_v_gpr_space_req       (vtx_alloc_req),
  .o_v_gpr_space             (vtx_alloc_space),
  .i_v_gpr_space_grant       (vtx_alloc_ack),
  .i_v_gpr_base_addr       (gpr_base),

  // control packet to VTB
  .o_v_ld_cntl_pkt        (vism_ctl_pkt_rts),
  .o_vs_first_thread      (vism_first_thread),
  .o_vs_resource          (vism_resource),
  .o_vs_instr_ptr         (vism_instr_ptr),
  .o_vector_valid            (vism_valid_bits),
  .o_v_gpr_base              (vism_gpr_base),
  .o_vgt_state               (vism_context_id),
  .i_vtb_rtr           (vtb_rtr),
  .i_context_valid        (acs_context_valid),
```

```
  .i_clk                    (sclk_global),
  .i_reset                  (srst)
);


// -------------------------------------------------------------------
// -------------------------------------------------------------------
// -- Pixel Input Control --
// -------------------------------------------------------------------
// -------------------------------------------------------------------

// - interconnect wires

wire [0:0]          pb_rts;
wire [`SQ_PB_WIDTH-1:0] pb_rd_data0;
wire [`SQ_PB_WIDTH-1:0] pb_rd_data1;
wire [`SQ_PB_WIDTH-1:0] pb_rd_data2;
wire [`SQ_PB_WIDTH-1:0] pb_rd_data3;

wire [1:0] pb_max_wcnt;
wire [2:0] pb_pix_state;

wire [0:0] pb_read_en;
wire [0:0] free_buff;
wire [0:0] pi_rtr;
```

```

wire [0:0] pix_alloc_req;
wire [5:0] pix_alloc_space;
wire [0:0] pix_alloc_ack;

wire [0:0] pix_dealloc_req;
wire [5:0] pix_dealloc_space;
wire [0:0] pix_dealloc_ack;

wire [0:0] pix_write_gnt;
wire [0:0] pix_write_req;
wire [0:0] pix_write_busy;

wire [6:0] pi_gpr_wr_addr;
wire [3:0] pi_gpr_wr_en;

wire [00:0] pism_ctl_pkt_rts;
wire [143:0] pism_lod_correct;
wire [11:0] pism_instr_ptr;
wire [63:0] pism_valid_bits;
wire [06:0] pism_gpr_base;
//wire [02:0] pism_context_id;
wire [00:0] pism_resource;
wire [00:0] pism_first_thread;
wire [00:0] ptb_rtr;

wire [0:0] pb_rd_en;

wire [0:0] vtx_vector_done;

wire [0:0] pb_event_rts;
wire [3:0] pb_event_id;
wire [2:0] pb_event_state;
wire [2:0] pb_dealloc_cnt;
wire [0:0] pb_dealloc_vld;

wire [1:0] gpr_phase;
wire [1:0] is_phase;
wire [1:0] is_subphase;


// --------------------
// -- Pointer Buffer --
// --------------------

sq_ptr_buff
u_sq_ptr_buff
(
   // SC interface
   .SC_SQ_data         (SC_SQ_data),
```

```
   .SC_SQ_valid        (SC_SQ_valid),

   .SQ_SC_free_buff(SQ_SC_free_buff),
   .SQ_SC_dec_cntr_cnt(SQ_SC_dec_cntr_cnt),

   // inputs from PISM
   .pi_rtr             (pi_rtr),
   .pi_read_en         (pi_read_en),
   .pi_free_buff       (pi_free_buff),

   // outputs to PISM
   .pb_rts             (pb_rts),
   .pb_rd_data0        (pb_rd_data0),
   .pb_rd_data1        (pb_rd_data1),
   .pb_rd_data2        (pb_rd_data2),
   .pb_rd_data3        (pb_rd_data3),

   .pb_max_wcnt        (pb_max_wcnt),
   .pb_pix_state       (pb_pix_state),

   .pb_event_rts       (pb_event_rts),
   .pb_event_id        (pb_event_id),
   .pb_event_state     (pb_event_state),

   .pb_dealloc_cnt     (pb_dealloc_cnt),
```

```
   .pb_dealloc_vld     (pb_dealloc_vld),

   .vtx_vector_done  (vtx_vector_done),
   .pix_write_busy     (pix_write_busy),

   .clk(sclk_global),
   .reset(srst)
);


// -------------------------------
// -- Pixel Input State Machine --
// -------------------------------

sq_pism
u_sq_pism
(
   // pointer buffer interface
   .pb_rts             (pb_rts),
   .pb_rd_data0        (pb_rd_data0),
   .pb_rd_data1        (pb_rd_data1),
   .pb_rd_data2        (pb_rd_data2),
   .pb_rd_data3        (pb_rd_data3),

   .pb_max_wcnt        (pb_max_wcnt),
```

```
   .pb_pix_state       (pb_pix_state),

   .pb_event_rts       (pb_event_rts),
   .pb_event_id        (pb_event_id),
   .pb_event_state     (pb_event_state),

   .pi_read_en         (pi_read_en),
   .pi_free_buff       (pi_free_buff),
   .pi_rtr             (pi_rtr),

   // interfaces to gpr alloc and input arb
   .pix_alloc_req      (pix_alloc_req),
   .pix_alloc_space    (pix_alloc_space),
   .pix_alloc_ack      (pix_alloc_ack),
   .pix_alloc_base     (gpr_base),

   .pix_write_req      (pix_write_req),
   .pix_write_busy     (pix_write_busy),
   .pix_write_gnt      (pix_write_gnt),

   // inputs from local registers
   .num_reg_set        (ps_num_reg_set),     //     connected     to SQ_PROGRAM_CNTL.PS_NUM_REG (6 bits)
   .num_param_set    (vs_export_count_set),      //     connected     to SQ_PROGRAM_CNTL.VS_EXPORT_COUNT (4 bits)
```

```
    .param_shade_set (param_shade_set),    // connected to SQ_INTRPOLATOR_CNTL.PARAM_SHADE (16 bits)

    .param_gen_i0_set         (param_gen_i0_set),  // connected to SQ_PROGRAM_CNTL.PARAM_GEN_I0 (1 bit)

    //.sampling_mode_set(sampling_mode_set),     // connected to SQ_SAMPLING_MODE.SAMPLING_MODE (1 bit)

    .sq_wrapping_0_set(sq_wrapping_0_set),       // connected to SQ_WRAPPING_0 (32 bits)

    .sq_wrapping_1_set(sq_wrapping_1_set),       // connected to SQ_WRAPPING_1 (32 bits)


    .ps_program_base_set(ps_program_base_set),// connected to SQ_PS_PROGRAM.BASE (12 bits)

    .ps_resource_set   (ps_resource_set),     // connected to SQ_PROGRAM_CNTL.PS_RESOURCE (1 bit)


    // pix control packet outputs to pixel thread buffer
    .ctl_pkt_rts_q      (pism_ctl_pkt_rts),
    .lod_correct_q      (pism_lod_correct),   // state
    .valid_bits_q       (pism_valid_bits),    // state
    .ps_program_base_q (pism_instr_ptr),      // state
    .gpr_base_q         (pism_gpr_base),      // state
    .ps_resource_q      (pism_resource),      // status: resource bit :  tex=1, alu=0
    .first_thread_q     (pism_first_thread),  // status: first thread of a new state
    .ptb_rtr            (ptb_rtr),            // PTB is ready when not full OR when not doing a CFS update

    // outputs to ais_output
    .pix_gpr_wr_addr (pi_gpr_wr_addr),
```

```
    .pix_gpr_wr_en    (pi_gpr_wr_en),

    // outputs to SP interp ctl
    .SQ_SP_interp_prim_type         (SQ_SP_interp_prim_type),
    .SQ_SP_interp_mode      (SQ_SP_interp_mode),   // ????
    .SQ_SP_interp_ijline    (SQ_SP_interp_ijline),
    .SQ_SP_interp_buff_swap         (SQ_SP_interp_buff_swap),
    .SQ_SP_interp_gen_i0    (SQ_SP_interp_gen_i0),
    .SQ_SP_interp_valid     (SQ_SP_interp_valid),

    .SQ_SX_interp_flat_vtx   (SQ_SX_interp_flat_vtx),
    .SQ_SX_interp_flat_gouraud(SQ_SX_interp_flat_gouraud),
    .SQ_SX_interp_cyl_wrap(SQ_SX_interp_cyl_wrap),

    // outputs to SX param cache
    .SQ_SX_pc_ptr0  (SQ_SX_pc_ptr0),
    .SQ_SX_pc_ptr1  (SQ_SX_pc_ptr1),
    .SQ_SX_pc_ptr2  (SQ_SX_pc_ptr2),
    .SQ_SX_rt_sel     (SQ_SX_rt_sel),

    .clk(sclk_global),
    .reset(srst)
  );
```

```
  // -----------------------------------------------------------------
  // -----------------------------------------------------------------
  // -- GPR Allocation and Input Arbitration --
  // -----------------------------------------------------------------
  // -----------------------------------------------------------------

  // -------------------
  // -- GPR Allocation --
  // -------------------

  sq_gpr_alloc
  u_sq_gpr_alloc
  (
    .vtx_alloc_req      (vtx_alloc_req),
    .vtx_alloc_space   (vtx_alloc_space),
    .vtx_dealloc_req   (vtx_dealloc_req),
    .vtx_dealloc_space(vtx_dealloc_space),

    .pix_alloc_req      (pix_alloc_req),
    .pix_alloc_space   (pix_alloc_space),
    .pix_dealloc_req   (pix_dealloc_req),
    .pix_dealloc_space(pix_dealloc_space),

    .base_ptr           (gpr_base),

```

```
    .pix_alloc_ack      (pix_alloc_ack),
    .pix_dealloc_ack   (pix_dealloc_ack),
    .vtx_alloc_ack      (vtx_alloc_ack),
    .vtx_dealloc_ack   (vtx_dealloc_ack),

    .clk(sclk_global),
    .reset(srst)
  );


  // ----------------------
  // -- Input Arbitration --
  // ----------------------

  sq_input_arb
  u_sq_input_arb
  (
    .vtx_req     (vtx_write_req),
    .vtx_busy  (vtx_write_busy),
    .pix_req     (pix_write_req),
    .pix_busy  (pix_write_busy),

    .gpr_phase(gpr_phase),

    .vtx_gnt     (vtx_write_gnt),
    .vtx_sel     (ia_vertex_sel),          // select VISM gpr write address, enable to drive to SP when 1 (select pixel if 0)
    .pix_gnt     (pix_write_gnt),

    .clk(sclk_global),
    .reset(srst)
  );


  // -------------------------------------------------------------------
  // -------------------------------------------------------------------
  // -- Thread Buffers, Thread Arbiters, Control Flow Sequencers --
  // -------------------------------------------------------------------
  // -------------------------------------------------------------------

  // - interconnect wires

  parameter  TB_DEPTH = 16;
  parameter  TB_ADDR_WIDTH = 4;

  wire [0:0] state_read_phase;
  wire [1:0] cfs_phase;

  wire [11:0] tcfs_is_read_addr;
  wire [11:0] acfs0_is_read_addr;
  wire [11:0] acfs1_is_read_addr;
```

```
  wire [11:0] tif_is_read_addr;
  wire [11:0] aif0_is_read_addr;
  wire [11:0] aif1_is_read_addr;

  wire [95:0] is_read_data;

  //
  wire [0:0]                          tcfs_update;
  wire [5:0]                          tcfs_thread_id;
  wire [`SQ_CFS_STATE_WIDTH-1:0]tcfs_state;
  wire [`SQ_STATUS_WIDTH-1:0] tcfs_status;

  //
  wire [0:0]                          acfs0_update;
  wire [5:0]                          acfs0_thread_id;
  wire [`SQ_CFS_STATE_WIDTH-1:0]acfs0_state;
  wire [`SQ_STATUS_WIDTH-1:0] acfs0_status;

  wire [0:0]                          acfs1_update;
  wire [5:0]                          acfs1_thread_id;
  wire [`SQ_CFS_STATE_WIDTH-1:0]acfs1_state;
  wire [`SQ_STATUS_WIDTH-1:0] acfs1_status;

  wire [TB_DEPTH-1:0]                 vtx_tex_req_q;
  wire [`SQ_VTX_STATE_WIDTH-1:0] vtx_tex_state;
```

```
  wire [`SQ_STATUS_WIDTH-1:0]          vtx_tex_status;
  wire [0:0]                          vtx_tex_winner_ack;
  wire [0:0]                          vtx_tex_state_vld;
  wire [TB_ADDR_WIDTH-1:0]      vtx_tex_winner_q;

  wire [TB_DEPTH-1:0]                 pix_tex_req_q;
  wire [`SQ_PIX_STATE_WIDTH-1:0] pix_tex_state;
  wire [`SQ_STATUS_WIDTH-1:0]          pix_tex_status;
  wire [0:0]                          pix_tex_winner_ack;
  wire [0:0]                          pix_tex_state_vld;
  wire [TB_ADDR_WIDTH-1:0]      pix_tex_winner_q;

  wire [TB_DEPTH-1:0]                 vtx_alu_req_q;
  wire [`SQ_VTX_STATE_WIDTH-1:0] vtx_alu_state;
  wire [`SQ_STATUS_WIDTH-1:0]          vtx_alu_status;
  wire [0:0]                          vtx_alu_winner_ack;
  wire [0:0]                          vtx_alu_state_vld;
  wire [TB_ADDR_WIDTH-1:0]      vtx_alu_winner_q;

  wire [TB_DEPTH-1:0]                 pix_alu_req_q;
  wire [`SQ_PIX_STATE_WIDTH-1:0] pix_alu_state;
  wire [`SQ_STATUS_WIDTH-1:0]          pix_alu_status;
  wire [0:0]                          pix_alu_winner_ack;
  wire [0:0]                          pix_alu_state_vld;
  wire [TB_ADDR_WIDTH-1:0]      pix_alu_winner_q;
```

```

  wire [0:0]                          ais0_done;
  wire [0:0]                          ais0_thread_type;
  wire [5:0]                          ais0_thread_id;

  wire [0:0]                          ais1_done;
  wire [0:0]                          ais1_thread_type;
  wire [5:0]                          ais1_thread_id;

  wire                    vtx_state_change;
  wire [2:0]              vtx_old_state;

  wire                    pix_state_change;
  wire [2:0]              pix_old_state;

  wire [0:0]                          tarb_rts;
  wire [`SQ_STATE_WIDTH-1:0]    tarb_state;
  wire [`SQ_STATUS_WIDTH-1:0] tarb_status;
  wire [0:0]                          tarb_thread_type;
  wire [0:0]                          tcfs_rtr;

  wire [0:0]                          tcfs_rts;
  wire [`SQ_CTL_PKT_WIDTH-1:0]          tcfs_ctl_pkt;
  wire [11:0]                         tcfs_tgt_instr_ptr;
  wire [11:0]                         tcfs_tgt_instr_cnt;
```

```
wire [0:0]                    tcfs_thread_type;
wire [0:0]                    tif_rtr;

wire [0:0]                    aarb_rts0;
wire [0:0]                    aarb_rts1;
wire [`SQ_VTX_STATE_WIDTH-1:0]     aarb_state;
wire [`SQ_STATUS_WIDTH-1:0] aarb_status;
wire [0:0]                    aarb_thread_type;
wire [0:0]                    acfs0_rtr;
wire [0:0]                    acfs1_rtr;

wire [0:0]                    acfs0_rts;
wire [`SQ_VTX_CTL_PKT_WIDTH-1:0] acfs0_ctl_pkt;
wire [11:0]                   acfs0_tgt_instr_ptr;
wire [11:0]                   acfs0_tgt_instr_cnt;
wire [0:0]                    aif0_rtr;

wire [0:0]                    acfs1_rts;
wire [`SQ_VTX_CTL_PKT_WIDTH-1:0] acfs1_ctl_pkt;
wire [11:0]                   acfs1_tgt_instr_ptr;
wire [11:0]                   acfs1_tgt_instr_cnt;
wire [0:0]                    aif1_rtr;

wire [6:0]                    param_cache_wptr_q;

```

```
wire [0:0]                    vtx_tb_busy;
wire [0:0]                    pix_tb_busy;
wire [0:0]                    tcfs_busy;
wire [0:0]                    acfs0_busy;
wire [0:0]                    acfs1_busy;
wire [0:0]                    tif_busy;
wire [0:0]                    tis_busy ;
wire [0:0]                    aif0_busy;
wire [0:0]                    aif1_busy;
wire [0:0]                    ais0_busy;
wire [0:0]                    ais1_busy;

wire [0:0] ais0_free_done;
wire [0:0] ais0_free_id;
wire [0:0] ais1_free_done;
wire [0:0] ais1_free_id;


// --------------------------
// -- Vertex Thread Buffer --
// --------------------------

sq_thread_buff
#(
  `SQ_VTX_STATE_WIDTH,                    `SQ_VTX_ISM_STATE_WIDTH, `SQ_CFS_STATE_WIDTH, `SQ_VTX_STATUS_WIDTH
```

```
 )
u_sq_vtx_thread_buff
 (
   .thread_type_strap(HI),              // a strap that tells this module if it's a vertex or pixel thread buffer
   .state_read_phase (state_read_phase),   // share read access to TB State Mem btwn the tex and alu arbiters
   .cfs_phase (cfs_phase),          // 00:alu0, 01:tex, 10:alu1, 11:tex

   // control packet input (from ISM) - initial values for state and status
   .ism_rts           (vism_ctl_pkt_rts),   // control packet rts
   .ism_lod_correct   (),                  // state - not used for VTB (PTB only)
   .ism_instr_ptr     (vism_instr_ptr),     // state
   .ism_valid_bits    (vism_valid_bits),    // state
   .ism_gpr_base      (vism_gpr_base),      // state
   .ism_context_id    (vism_context_id),    // state
   .ism_resource      (vism_resource),      // status: resource bit :  tex=1, alu=0
   .ism_first_thread  (vism_first_thread),  // status: first thread of a new state
   .tb_rtr            (vtb_rtr),            // rtr when not full OR when not doing a CFS update

   // tex control flow seq update of state and status
   .tcfs_update       (tcfs_update),        // load updated status info from CFS
   .tcfs_thread_type  (tcfs_thread_type),          //
   .tcfs_state  (tcfs_state),          // state
   .tcfs_status(tcfs_status),          // status
```

```

   // alu control flow seq update of state and status
   .acfs0_update      (acfs0_update),          // load updated status info from CFS
   .acfs0_thread_type(acfs0_thread_type),  //
   .acfs0_state       (acfs0_state),        // state
   .acfs0_status      (acfs0_status),       // status

   .acfs1_update      (acfs1_update),          // load updated status info from CFS
   .acfs1_thread_type(acfs1_thread_type),  //
   .acfs1_state       (acfs1_state),        // state
   .acfs1_status      (acfs1_status),       // status

   // tex thread arbiter interface
   .tex_req_q (vtx_tex_req_q),       // tex request from every thread in the buffer
   .tex_winner_q      (vtx_tex_winner_q),   // tex winner
   .tex_winner_ack    (vtx_tex_winner_ack),        // tex winner valid (request acknowledge) from tex arbiter
   .tex_state_q       (vtx_tex_state),      // winning state read from State Mem
   .tex_status_q      (vtx_tex_status),     // winning status read from Status Regs

   // done info from TP
   .TP_SQ_data_rdy (TP_SQ_data_rdy),    // data ready (done) indicator from TPC
   .TP_SQ_type        (TP_SQ_type),            // the vector type: pixel=0, vertex=1
   .TP_SQ_thread_id(TP_SQ_thread_id),   //

   // alu thread arbiter interface
```

```
    .alu_req_q (vtx_alu_req_q),         // tex request from every thread in the buffer
    .alu_winner_q       (vtx_alu_winner_q),   // alu winner
    .alu_winner_ack   (vtx_alu_winner_ack),           // alu winner valid from alu arbiter
    .alu_state_q          (vtx_alu_state),      // winning state read from State Mem
    .alu_status_q         (vtx_alu_status),     // winning status read from Status Regs

    // done info from AIS's
    .ais0_done (ais0_done),               // done indicator from AIS0
    .ais0_thread_type  (ais0_thread_type),     // the vector type: pixel=0       (), vertex=1
    .ais0_thread_id      (ais0_thread_id),         //
    .ais1_done (ais1_done),               // done indicator from AIS1
    .ais1_thread_type  (ais1_thread_type),     // the vector type: pixel=0, vertex=1
    .ais1_thread_id      (ais1_thread_id),         //

     // SX export buffer availability
    .u0_SX_SQ_exp_count_rdy(u0_SX_SQ_exp_count_rdy),        // position available from SX
    .u0_SX_SQ_exp_pos_avail(u0_SX_SQ_exp_pos_avail),          // position available from SX
    .u0_SX_SQ_exp_buf_avail(u0_SX_SQ_exp_buf_avail),          // buffer available from SX (0 to 127 2kbit buffers(), 2kbit = 32bits for 64 pixels)
    .u1_SX_SQ_exp_count_rdy(u1_SX_SQ_exp_count_rdy),        // position available from SX
    .u1_SX_SQ_exp_pos_avail(u1_SX_SQ_exp_pos_avail),          // position available from SX
    .u1_SX_SQ_exp_buf_avail(u1_SX_SQ_exp_buf_avail),          // buffer available from SX (0 to 127 2kbit buffers(), 2kbit = 32bits for 64 pixels)

```

```
    // outputs from exit SM to constant stores and gpr alloc
    // - this needs to be fixed...
    .state_change        (vtx_state_change),    // a pulse high indicates that the state exiting the SS has changed
    .old_state  (vtx_old_state),          // the state that has finished (because a new state has emerged)
    .dealloc_req          (vtx_dealloc_req),      // request to deallocate GPRs
    .dealloc_ack          (vtx_dealloc_ack),      // the dealloc request has been acknowleged

    .pop_thread           (vtx_vector_done),      // vtx shader sync output back to pix input ctl

    .param_cache_wptr_q(param_cache_wptr_q),    // from export_alloc - needed for status reg request logic

    .busy(vtx_tb_busy),                     // vtx TB busy
    .clk(sclk_global),
    .reset(srst)
  );


  // -------------------------
  // -- Pixel Thread Buffer --
  // -------------------------

  sq_thread_buff
  #(
```

```
    `SQ_PIX_STATE_WIDTH, `SQ_PIX_ISM_STATE_WIDTH, `SQ_CFS_STATE_WIDTH, `SQ_PIX_STATUS_WIDTH
   )
  u_sq_pix_thread_buff
  (
    .thread_type_strap(LO),                         // a strap that tells this module if it's a vertex or pixel thread buffer
    .state_read_phase  (state_read_phase),     // share read access to TB State Mem btwn the tex and alu arbiters
    .cfs_phase (cfs_phase),               // 00:alu0, 01:tex, 10:alu1, 11:tex

    // control packet input (from ISM) - initial values for state and status
    .ism_rts                 (pism_ctl_pkt_rts),     // control packet rts
    .ism_lod_correct   (pism_lod_correct),     // state
    .ism_instr_ptr        (pism_instr_ptr),         // state
    .ism_valid_bits       (pism_valid_bits),       // state
    .ism_gpr_base        (pism_gpr_base),         // state
    .ism_context_id     (pb_pix_state),                       // state
    .ism_resource        (pism_resource),          // status: resource bit :  tex=1, alu=0
    .ism_first_thread   (pism_first_thread),    // status: first thread of a new state
    .tb_rtr                   (ptb_rtr),                    // rtr when not full OR when not doing a CFS update

    // tex control flow seq update of state and status
    .tcfs_update          (tcfs_update),              // load updated status info from CFS
    .tcfs_thread_type  (tcfs_thread_type),                //
    .tcfs_state  (tcfs_state),               // state
```

```
    .tcfs_status(tcfs_status),              // status

    // alu control flow seq update of state and status
    .acfs0_update       (acfs0_update),                      // load updated status info from CFS
    .acfs0_thread_type(acfs0_thread_type),   //
    .acfs0_state          (acfs0_state),             // state
    .acfs0_status         (acfs0_status),            // status

    .acfs1_update       (acfs1_update),                      // load updated status info from CFS
    .acfs1_thread_type(acfs1_thread_type),   //
    .acfs1_state          (acfs1_state),             // state
    .acfs1_status         (acfs1_status),            // status

    // tex thread arbiter interface
    .tex_req_q (pix_tex_req_q),         // tex request from every thread in the buffer
    .tex_winner_q        (pix_tex_winner_q),   // tex winner
    .tex_winner_ack   (pix_tex_winner_ack),           // tex winner valid (request acknowledge) from tex arbiter
    .tex_state_q          (pix_tex_state),      // winning state read from State Mem
    .tex_status_q         (pix_tex_status),     // winning status read from Status Regs

    // done info from TP
    .TP_SQ_data_rdy (TP_SQ_data_rdy),     // data ready (done) indicator from TPC
    .TP_SQ_type         (TP_SQ_type),                       // the vector type: pixel=0, vertex=1
    .TP_SQ_thread_id(TP_SQ_thread_id),   //

```

```
        // alu thread arbiter interface
        .alu_req_q (pix_alu_req_q),         // alu req from every thread
        .alu_winner_q       (pix_alu_winner_q),   // alu winner
        .alu_winner_ack   (pix_alu_winner_ack),           // alu winner valid from alu arbiter
        .alu_state_q          (pix_alu_state),      // winning state read from State Mem
        .alu_status_q         (pix_alu_status),     // winning status read from Status Regs

        // done info from AIS's
        .ais0_done (ais0_done),               // done indicator from AIS0
        .ais0_thread_type  (ais0_thread_type),     // the vector type: pixel=0       (), vertex=1
        .ais0_thread_id      (ais0_thread_id),         //
        .ais1_done (ais1_done),               // done indicator from AIS1
        .ais1_thread_type  (ais1_thread_type),     // the vector type: pixel=0, vertex=1
        .ais1_thread_id      (ais1_thread_id),         //

         // SX export buffer availability
        .u0_SX_SQ_exp_count_rdy(u0_SX_SQ_exp_count_rdy),        // position available from SX
        .u0_SX_SQ_exp_pos_avail(u0_SX_SQ_exp_pos_avail),          // position available from SX
        .u0_SX_SQ_exp_buf_avail(u0_SX_SQ_exp_buf_avail),          // buffer available from SX (0 to 127 2kbit buffers(), 2kbit = 32bits for 64 pixels)
        .u1_SX_SQ_exp_count_rdy(u1_SX_SQ_exp_count_rdy),        // position available from SX
        .u1_SX_SQ_exp_pos_avail(u1_SX_SQ_exp_pos_avail),          // position available from SX
        .u1_SX_SQ_exp_buf_avail(u1_SX_SQ_exp_buf_avail),          // buffer available from SX (0 to 127 2kbit buffers(), 2kbit = 32bits for 64 pixels)
```

```

        // outputs from exit SM to constant stores and gpr alloc
        // - this needs to be fixed...
        .state_change         (pix_state_change),    // a pulse high indicates that the state exiting the SS has changed
        .old_state  (pix_old_state),          // the state that has finished (because a new state has emerged)
        .dealloc_req          (pix_dealloc_req),      // request to deallocate GPRs
        .dealloc_ack          (pix_dealloc_ack),      // the dealloc request has been acknowleged

        .pop_thread           (),        // no connect for pix TB

        .param_cache_wptr_q(param_cache_wptr_q),    // from export_alloc - needed for status reg request logic
                                                    // not used by pix TB since type must be VTX in request logic

        .busy(pix_tb_busy),          // pix TB busy
        .clk(sclk_global),
        .reset(srst)
    );


    // ----------------------------
    // -- Texture Thread Arbiter --
    // ----------------------------

```

```
    sq_thread_arb
    #(
      `SQ_STATE_WIDTH, `SQ_STATUS_WIDTH
     )
    u_sq_tex_thread_arb
    (
        .arb_type_strap      (HI),    // tex = 1, alu = 0
        .state_read_phase  (state_read_phase),     // share read access to TB State Mem btwn the tex and alu arbiters

        // vertex and pixel thread buffer interface
        .vtx_req_q (vtx_tex_req_q),         // 16 vtx_thread_buff requests
        .vtx_winner_q       (vtx_tex_winner_q),   // winning vertex thread_id sent back to Vertex Thread Buffer
        .vtx_winner_ack   (vtx_tex_winner_ack),           //
        .vtx_state  (vtx_tex_state),          // state selected by winner
        .vtx_status (vtx_tex_status),         // status selected by winner

        .pix_req_q (pix_tex_req_q),         // 16 pix_thread_buff requests
        .pix_winner_q       (pix_tex_winner_q),   // winning pixel thread_id sent back to Pixel Thread Buffer
        .pix_winner_ack   (pix_tex_winner_ack),           //
        .pix_state  (pix_tex_state),          //
        .pix_status (pix_tex_status),         //

        // control flow sequencer interface
```

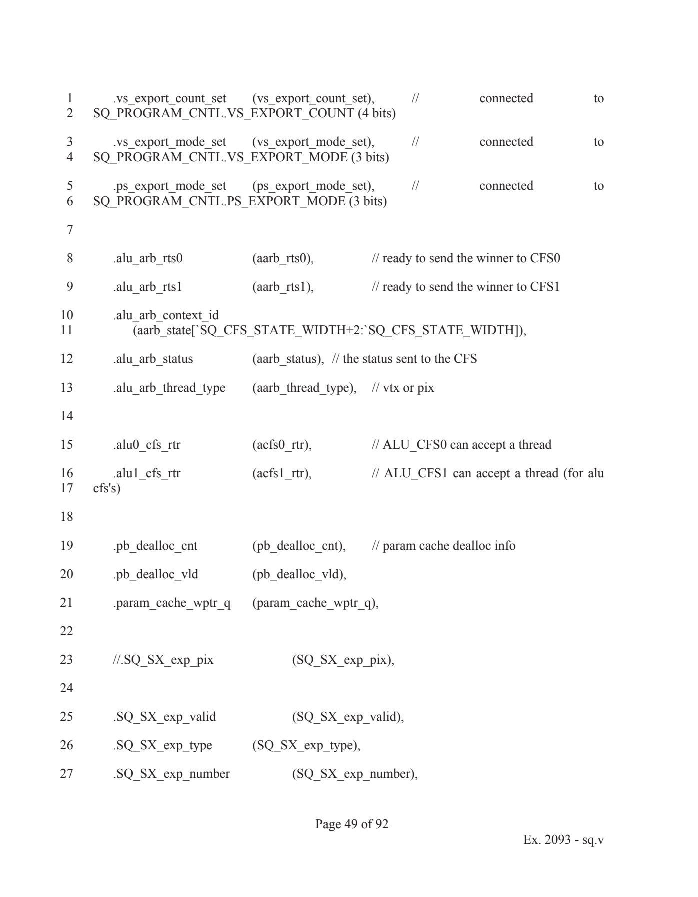
```
        .arb_rts0             (tarb_rts),                  // ready to send the winner to CFS0
        .arb_rts1             (),                          // no connect for tex thread arb
        .arb_state   (tarb_state),              // the state sent to the CFS
        .arb_status (tarb_status),              // the status sent to the CFS
        .arb_thread_type   (tarb_thread_type),     // vtx or pix

        .cfs_rtr0             (tcfs_rtr),                  // CFS0 can accept a thread
        .cfs_rtr1             (LO),                        // always tied LO for tex thread arb

        .cfs1_enable          (LO),                        // always tied LO for tex thread arb

        .clk(sclk_global),
        .reset(srst)
    );


    // -------------
    // -- Tex CFS --
    // -------------

    sq_ctl_flow_seq
    #(
      `SQ_CTL_PKT_WIDTH, `SQ_STATE_WIDTH, `SQ_STATUS_WIDTH
     )
    u_sq_tex_ctl_flow_seq
```

```
  (
    .cfs_type_strap     (2'b01),                  // 00:alu0, 01:tex, 10:alu1
    .is_phase           (is_phase),               // 00:CF, 01:Tex, 10:ALU, 11:CP
    .is_subphase        (is_subphase),            // 00:alu0, 01:tex, 10:alu1, 11:tex
    .cfs_phase (cfs_phase),              // 00:alu0, 01:tex, 10:alu1, 11:tex

    // local registers
    // - per chip
    .inst_base_vtx      (inst_base_vtx),          // vertex base (wrap point)
    .inst_base_pix      (inst_base_pix),          // pixel base  (wrap point)
    // - per context
    .vs_program_base_set(vs_program_base_set),// connected to SQ_VS_PROGRAM.BASE (12 bits)
    .ps_program_base_set(ps_program_base_set),// connected to SQ_PS_PROGRAM.BASE (12 bits)

    // thread arbiter interface
    .arb_rts            (tarb_rts),               //
    .arb_state   (tarb_state),            //
    .arb_status (tarb_status),            //
    .arb_thread_type   (tarb_thread_type),     // vertex or pixel
    .cfs_rtr_q   (tcfs_rtr),              // CFS can take a new packet

    // instruction store interface
    .is_read_addr_q     (tcfs_is_read_addr),   // instruction store read address
    .is_read_data_q     (is_read_data),           // instruction store read data
```

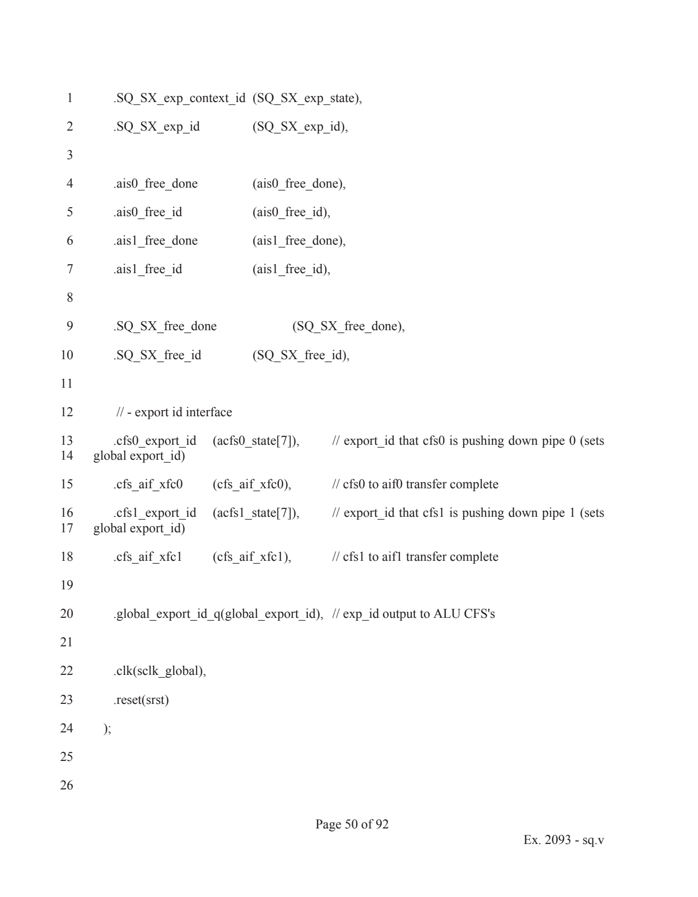
```

    // interface to the thread buffer (for thread updates)
    .cfs_update_q       (tcfs_update),            // load updated status info from CFS
    .cfs_state   (tcfs_state),            // state
    .cfs_status (tcfs_status),            // status

    // outputs to the target instruction fetcher
    .cfs_rts_q   (tcfs_rts),              // ctl packet and ptr are valid
    .cfs_ctl_pkt_q      (tcfs_ctl_pkt),           // the control packet (lod_correct, valid_bits, gpr_base, context_id)
    .cfs_tgt_instr_ptr_q(tcfs_tgt_instr_ptr), // the instr store address of the first target instruction
    .cfs_tgt_instr_cnt_q(tcfs_tgt_instr_cnt), // the number of target instructions to be fetched
    .cfs_thread_type_q(tcfs_thread_type),     // vertex or pixel
    //.cfs_param_ptr_q(),        // param cache ptr - not needed for texture
    .tif_rtr            (tif_rtr),                // TIF can take a new packet

    .busy(tcfs_busy),
    .clk(sclk_global),
    .reset(srst)
  );


  // ------------------------
  // -- ALU Thread Arbiter --
  // ------------------------
```

```

  sq_thread_arb
  #(
    `SQ_VTX_STATE_WIDTH, `SQ_STATUS_WIDTH
   )
  u_sq_alu_thread_arb
  (
    .arb_type_strap     (LO),  // tex = 1, alu = 0
    .state_read_phase  (state_read_phase),  // share read access to TB State Mem btwn the tex and alu arbiters

    // vertex and pixel thread buffer interface
    .vtx_req_q (vtx_alu_req_q),       // 16 vtx_thread_buff requests
    .vtx_winner_q       (vtx_alu_winner_q),  // winning vertex thread_id sent back to Vertex Thread Buffer
    .vtx_winner_ack   (vtx_alu_winner_ack),          //
    .vtx_state   (vtx_alu_state),         // state selected by winner
    .vtx_status (vtx_alu_status),         // status selected by winner

    .pix_req_q (pix_alu_req_q),       // 16 pix_thread_buff requests
    .pix_winner_q       (pix_alu_winner_q),  // winning pixel thread_id sent back to Pixel Thread Buffer
    .pix_winner_ack   (pix_alu_winner_ack),          //
    .pix_state   (pix_alu_state),         //
    .pix_status (pix_alu_status),         //

```

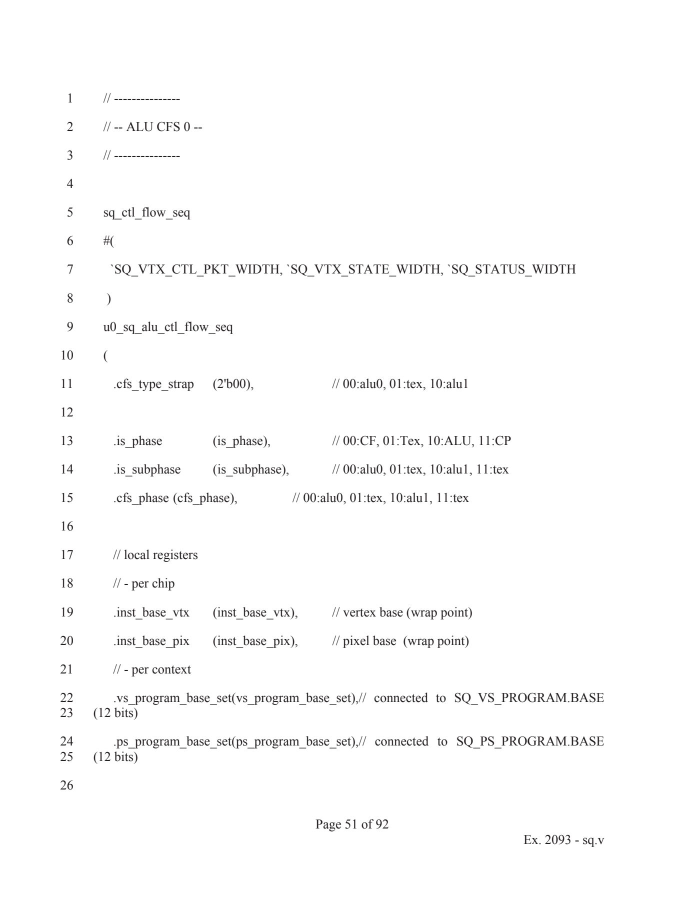
```
    // control flow sequencer interface
    .arb_rts0           (aarb_rts0),              // ready to send the winner to CFS0
    .arb_rts1           (aarb_rts1),              // ready to send the winner to CFS1
    .arb_state   (aarb_state),            // the state sent to the CFS
    .arb_status (aarb_status),            // the status sent to the CFS
    .arb_thread_type   (aarb_thread_type),     // vtx or pix
    .cfs_rtr0           (acfs0_rtr),              // CFS0 can accept a thread
    .cfs_rtr1           (acfs1_rtr),              // CFS1 can accept a thread

    //.cfs1_enable      (LO),                     // enable sending packets to CFS1
    .cfs1_enable        (HI),                     // enable sending packets to CFS1
                                     // (this a local register setting: SQ_FLOW_CTL.ONE_ALU)
    .clk(sclk_global),
    .reset(srst)
  );


  // ------------------
  // -- Export Alloc --
  // ------------------

  sq_export_alloc
  u_sq_export_alloc
  (
    // inputs from local registers
```



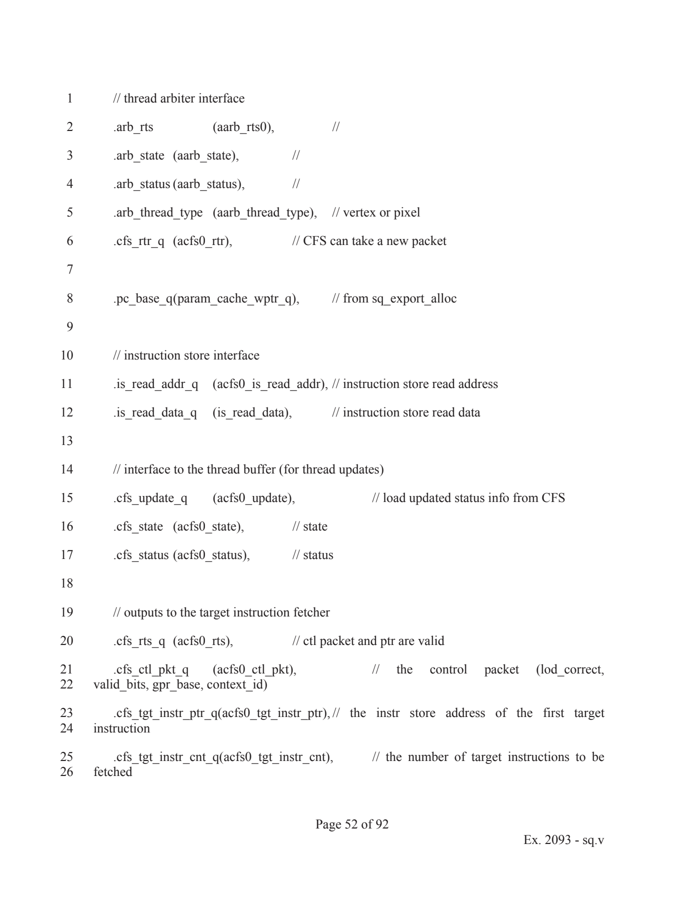
```
      .vs_export_count_set      (vs_export_count_set),     //     connected     to SQ_PROGRAM_CNTL.VS_EXPORT_COUNT (4 bits)
      .vs_export_mode_set       (vs_export_mode_set),      //     connected     to SQ_PROGRAM_CNTL.VS_EXPORT_MODE (3 bits)
      .ps_export_mode_set       (ps_export_mode_set),      //     connected     to SQ_PROGRAM_CNTL.PS_EXPORT_MODE (3 bits)

      .alu_arb_rts0             (aarb_rts0),          // ready to send the winner to CFS0
      .alu_arb_rts1             (aarb_rts1),          // ready to send the winner to CFS1
      .alu_arb_context_id
          (aarb_state[`SQ_CFS_STATE_WIDTH+2:`SQ_CFS_STATE_WIDTH]),
      .alu_arb_status           (aarb_status),  // the status sent to the CFS
      .alu_arb_thread_type      (aarb_thread_type),    // vtx or pix

      .alu0_cfs_rtr             (acfs0_rtr),          // ALU_CFS0 can accept a thread
      .alu1_cfs_rtr             (acfs1_rtr),          // ALU_CFS1 can accept a thread (for alu cfs's)

      .pb_dealloc_cnt           (pb_dealloc_cnt),     // param cache dealloc info
      .pb_dealloc_vld           (pb_dealloc_vld),
      .param_cache_wptr_q       (param_cache_wptr_q),

      //.SQ_SX_exp_pix                 (SQ_SX_exp_pix),

      .SQ_SX_exp_valid               (SQ_SX_exp_valid),
      .SQ_SX_exp_type           (SQ_SX_exp_type),
      .SQ_SX_exp_number              (SQ_SX_exp_number),
```



```
      .SQ_SX_exp_context_id  (SQ_SX_exp_state),
      .SQ_SX_exp_id             (SQ_SX_exp_id),

      .ais0_free_done           (ais0_free_done),
      .ais0_free_id             (ais0_free_id),
      .ais1_free_done           (ais1_free_done),
      .ais1_free_id             (ais1_free_id),

      .SQ_SX_free_done               (SQ_SX_free_done),
      .SQ_SX_free_id            (SQ_SX_free_id),

      // - export id interface
      .cfs0_export_id    (acfs0_state[7]),      // export_id that cfs0 is pushing down pipe 0 (sets global export_id)
      .cfs_aif_xfc0      (cfs_aif_xfc0),        // cfs0 to aif0 transfer complete
      .cfs1_export_id    (acfs1_state[7]),      // export_id that cfs1 is pushing down pipe 1 (sets global export_id)
      .cfs_aif_xfc1      (cfs_aif_xfc1),        // cfs1 to aif1 transfer complete

      .global_export_id_q(global_export_id),  // exp_id output to ALU CFS's

      .clk(sclk_global),
      .reset(srst)
   );
```



```
   // ---------------
   // -- ALU CFS 0 --
   // ---------------

   sq_ctl_flow_seq
   #(
     `SQ_VTX_CTL_PKT_WIDTH, `SQ_VTX_STATE_WIDTH, `SQ_STATUS_WIDTH
    )
   u0_sq_alu_ctl_flow_seq
   (
      .cfs_type_strap     (2'b00),              // 00:alu0, 01:tex, 10:alu1

      .is_phase           (is_phase),           // 00:CF, 01:Tex, 10:ALU, 11:CP
      .is_subphase        (is_subphase),        // 00:alu0, 01:tex, 10:alu1, 11:tex
      .cfs_phase (cfs_phase),            // 00:alu0, 01:tex, 10:alu1, 11:tex

      // local registers
      // - per chip
      .inst_base_vtx      (inst_base_vtx),      // vertex base (wrap point)
      .inst_base_pix      (inst_base_pix),      // pixel base  (wrap point)
      // - per context
      .vs_program_base_set(vs_program_base_set),//  connected  to  SQ_VS_PROGRAM.BASE (12 bits)
      .ps_program_base_set(ps_program_base_set),//  connected  to  SQ_PS_PROGRAM.BASE (12 bits)
```



```
      // thread arbiter interface
      .arb_rts            (aarb_rts0),          //
      .arb_state   (aarb_state),          //
      .arb_status (aarb_status),          //
      .arb_thread_type   (aarb_thread_type),   // vertex or pixel
      .cfs_rtr_q  (acfs0_rtr),              // CFS can take a new packet

      .pc_base_q(param_cache_wptr_q),       // from sq_export_alloc

      // instruction store interface
      .is_read_addr_q    (acfs0_is_read_addr), // instruction store read address
      .is_read_data_q    (is_read_data),       // instruction store read data

      // interface to the thread buffer (for thread updates)
      .cfs_update_q      (acfs0_update),            // load updated status info from CFS
      .cfs_state   (acfs0_state),          // state
      .cfs_status (acfs0_status),          // status

      // outputs to the target instruction fetcher
      .cfs_rts_q  (acfs0_rts),             // ctl packet and ptr are valid
      .cfs_ctl_pkt_q      (acfs0_ctl_pkt),          //   the   control   packet   (lod_correct, valid_bits, gpr_base, context_id)
      .cfs_tgt_instr_ptr_q(acfs0_tgt_instr_ptr), //  the  instr  store  address  of  the  first  target instruction
      .cfs_tgt_instr_cnt_q(acfs0_tgt_instr_cnt),      //  the  number  of  target  instructions  to  be fetched
```

```
   .cfs_thread_type_q(acfs0_thread_type),  // vertex or pixel
   .tif_rtr                (aif0_rtr),                  // TIF can take a new packet

   .global_export_id  (global_export_id),     // exp_id input from sq_exp_alloc
   .cfs_tif_xfc           (cfs_aif_xfc0),           // cfs0 to aif0 transfer complete (to sq_exp_alloc)

   .busy(acfs0_busy),
   .clk(sclk_global),
   .reset(srst)
 );


 // ---------------
 // -- ALU CFS 1 --
 // ---------------

 sq_ctl_flow_seq
 #(
   `SQ_VTX_CTL_PKT_WIDTH, `SQ_VTX_STATE_WIDTH, `SQ_STATUS_WIDTH
  )
 u1_sq_alu_ctl_flow_seq
 (
   .cfs_type_strap      (2'b10),                       // 00:alu0, 01:tex, 10:alu1

   .is_phase               (is_phase),                 // 00:CF, 01:Tex, 10:ALU, 11:CP
```

```
   .is_subphase          (is_subphase),             // 00:alu0, 01:tex, 10:alu1, 11:tex
   .cfs_phase (cfs_phase),                // 00:alu0, 01:tex, 10:alu1, 11:tex

   // local registers
   // - per chip
   .inst_base_vtx       (inst_base_vtx),           // vertex base (wrap point)
   .inst_base_pix       (inst_base_pix),           // pixel base  (wrap point)
   // - per context
   .vs_program_base_set(vs_program_base_set),//  connected  to  SQ_VS_PROGRAM.BASE
(12 bits)
   .ps_program_base_set(ps_program_base_set),//  connected  to  SQ_PS_PROGRAM.BASE
(12 bits)

   // thread arbiter interface
   .arb_rts                  (aarb_rts1),                //
   .arb_state   (aarb_state),               //
   .arb_status (aarb_status),              //
   .arb_thread_type   (aarb_thread_type),    // vertex or pixel
   .cfs_rtr_q   (acfs1_rtr),                  // CFS can take a new packet

   .pc_base_q(param_cache_wptr_q),          // from sq_export_alloc

   // instruction store interface
   .is_read_addr_q     (acfs1_is_read_addr), // instruction store read address
   .is_read_data_q     (is_read_data),            // instruction store read data

```

```
   // interface to the thread buffer (for thread updates)
   .cfs_update_q        (acfs1_update),                       // load updated status info from CFS
   .cfs_state   (acfs1_state),               // state
   .cfs_status (acfs1_status),              // status

   // outputs to the target instruction fetcher
   .cfs_rts_q  (acfs1_rts),                   // ctl packet and ptr are valid
   .cfs_ctl_pkt_q        (acfs1_ctl_pkt),                    //     the     control     packet     (lod_correct,
valid_bits, gpr_base, context_id)
   .cfs_tgt_instr_ptr_q(acfs1_tgt_instr_ptr), //   the   instr  store  address  of  the  first  target
instruction
   .cfs_tgt_instr_cnt_q(acfs1_tgt_instr_cnt),           //  the  number  of  target  instructions  to  be
fetched
   .cfs_thread_type_q(acfs1_thread_type),  // vertex or pixel
   .tif_rtr                (aif1_rtr),                  // TIF can take a new packet

   .global_export_id  (global_export_id),     // exp_id input from sq_exp_alloc
   .cfs_tif_xfc           (cfs_aif_xfc1),           // cfs0 to aif0 transfer complete (to sq_exp_alloc)

   .busy(acfs1_busy),
   .clk(sclk_global),
   .reset(srst)
 );


 // -------------------------------------------------------------------
```

```
  // -------------------------------------------------------------------
  // -- Texture Instruction Pipe --
  // -------------------------------------------------------------------
  // -------------------------------------------------------------------

  // - interconnect wires


  wire [0:0]  alu_phase;

  wire [04:0] texconst_rd_addr;         // texture constant store read address (logical addr - up to
32 consts)
  wire [95:0] texconst_rd_data;         // texture constant store read data

  wire [0:0] tif_instr_rts;       // the target instr register is valid
  wire [95:0] tif_instr;// the target instruction register (TIR)
  wire [`SQ_CTL_PKT_WIDTH-1:0] tif_ctl_pkt;    // the target control packet (pipelined from
reg'd input)
  wire [0:0] tif_last_in_group;           // last instruction flag
  wire [0:0] tif_thread_type;  // vert:1, pix:0
  wire [5:0] tif_thread_id;       // the target thread_id (pipelined from reg'd input)

  wire  [00:0] tiq_rtr;

  wire [02:0] tiq_context_id;
  wire [63:0] tiq_valid_bits;
  wire [95:0] tiq_lod_correct;
wire [00:0] tiq_thread_type;
wire [05:0] tiq_thread_id;
wire [95:0] tiq_instr;

wire [0:0] tis_rtr;

wire [6:0] tis_gpr_rd_addr; // GPR read address for Fetch Address
wire [0:0]     tiq_rts;



// ------------------------------
// -- Texture Instruction Fetch --
// ------------------------------

sq_target_instr_fetch
#(
 `SQ_CTL_PKT_WIDTH   // tex pipe needs LOD bits from PIX ctl pkts
 )
u_sq_tex_instr_fetch
(
   .target_strap(`SQ_TEX_STRAP),          // hardwired to TEX_STRAP, ALU0_STRAP, or ALU1_STRAP

   // local registers
   .inst_base_vtx(inst_base_vtx),        // vertex base
   .inst_base_pix(inst_base_pix),        // pixel base
```

```

   // cfs interface
   .cfs_rts            (tcfs_rts),              // ctl packet and ptr are valid
   .cfs_ctl_pkt        (tcfs_ctl_pkt),          // the control packet (lod for pix_tex, valid_bits, gpr_base, context_id)
   .cfs_instr_ptr      (tcfs_tgt_instr_ptr),    // the Instruction Store address of the first target instruction
   .cfs_instr_cnt      (tcfs_tgt_instr_cnt),    // the number of instructions to be fetched
   .cfs_pc_base        (tcfs_state[7:1]),       // the param cache base  - not used by tex instr pipe
   .cfs_thread_type    (tcfs_thread_type),      // vertex or pixel
   .cfs_thread_id      (tcfs_status[21:16]),    //
   .cfs_last_in_thread(tcfs_status[12]),        // last instr in shader prog
   .tif_rtr            (tif_rtr),               // TIF can take a new packet

   // instruction store interface
   .is_read_addr       (tif_is_read_addr),      // instruction store read address
   .is_read_data       (is_read_data),          // instruction store read data
   .is_phase           (is_phase),              // instruction store phase
   .alu_phase (LO),                        // tied low for TEX instance, to alu_phase for ALU instances

   // outputs to the target instruction decoder
   .tif_pc_base_q      (),                 // the param cache base output - not connected to anything...
   .tif_ctl_pkt_q      (tif_ctl_pkt),           // the target control packet    (pipelined from reg'd input)
   .tif_last_in_group_q(tif_last_in_group),  // last instruction flag
```

```
   .tif_thread_type_q(tif_thread_type),        // vert:1, pix:0
   .tif_thread_id_q    (tif_thread_id),        // the target thread_id (pipelined from reg'd input)
   .tif_instr_q(tif_instr),               // the target instruction register (TIR)
   .tif_instr_rts_q      (tif_instr_rts),  // the target instr register is valid
   .tiq_rtr               (tiq_rtr),            // the target instr decode is ready to take the TIR (and other pipeline data)

   .busy(tif_busy),
   .clk(sclk_global),
   .reset(srst)
);



// -------------------------------
// -- Texture Instruction Queue --
// -------------------------------

parameter TIQ_NUM_WORDS = 4;
parameter TIQ_ADDR_BITS = 2;

sq_tex_instr_queue
// #(TIQ_NUM_WORDS, TIQ_ADDR_BITS)
u_sq_tex_instr_queue
(
   .write_rts   (tif_instr_rts),
   .write_rtr   (tiq_rtr),
```

```

   // queue inputs
   //write_data(),
   .tif_ctl_pkt_q      (tif_ctl_pkt),           // control packet
   .tif_thread_id_q    (tif_thread_id),         // thread_id
   .tif_last_instr_q   (tif_last_in_group),     // last instruction flag
   .tif_thread_type_q(tif_thread_type),         // 0: pixel, 1: vertex
   .tir_q              (tif_instr),             // instruction register (TIR)

   .read_rts           (tiq_rts),
   .read_rtr           (tis_rtr),
   //read_data(),                  // {control packet, clause num, instruction}

   // queue outputs
   .tiq_last_instr     (tiq_last_instr),        // last instruction flag
   .tiq_thread_type    (tiq_thread_type),       // 0: pixel, 1: vertex
   .tiq_context_id     (tiq_context_id),        // context_id (from ctl packet)
   .tiq_valid_bits     (tiq_valid_bits),        // valid bits (from ctl packet)
   .tiq_lod_correct    (tiq_lod_correct),       // lod_correct bits (from ctl packet)
   .tiq_thread_id      (tiq_thread_id),         // thread_id
   .tiq_instr   (tiq_instr),               // instruction

   .clk(sclk_global),
   .reset(srst)
 );
```

```


  // -----------------------------------
  // -- Texture Instruction Sequencer --
  // -----------------------------------

  sq_tex_instr_seq
  u_sq_tex_instr_seq
  (
    // TIQ interface
    .tiq_rts            (tiq_rts),            // rts from TIQ FIFO
    .tiq_last_instr     (tiq_last_instr),     // last instruction flag
    .tiq_thread_type    (tiq_thread_type),    // 0: pixel, 1: vertex
    .tiq_context_id     (tiq_context_id),     // context_id (from ctl packet)
    .tiq_valid_bits     (tiq_valid_bits),     // valid bits (from ctl packet)
    .tiq_lod_correct    (tiq_lod_correct),    // lod_correct bits (from ctl packet)
    .tiq_thread_id      (tiq_thread_id),      // thread_id
    .tiq_instr    (tiq_instr),          // instruction

    .tis_rtr            (tis_rtr),            // TIQ FIFO pop

    .gpr_phase (gpr_phase),

    // TIS outputs to other SQ blocks
    .tis_gpr_rd_addr    (tis_gpr_rd_addr),    // GPR read address for Fetch Address: to gpr_ra_output_mux
```

```
    .texconst_rd_addr (texconst_rd_addr),    // texture constant store read address
    .texconst_rd_data  (texconst_rd_data),    // texture constant store read data

    // outputs to TP
    .SQ_TP_vld          (SQ_TP_send),
    .SQ_TP_instr        (SQ_TP_instr),
    .SQ_TP_const        (SQ_TP_const),
    .SQ_TP_gpr_phase           (SQ_TP_gpr_phase),
    .SQ_TP_gpr_wr_addr(SQ_TP_gpr_wr_addr),    // sends gpr_wr_addr plus type over 4 cycles
    .SQ_TP_thread_id(SQ_TP_thread_id),   // sends thread_id plus end_of_group over 4 cycles

    .u0_SQ_TP_lod_correct(u0_SQ_TP_lod_correct),
    .u1_SQ_TP_lod_correct(u1_SQ_TP_lod_correct),
    .u2_SQ_TP_lod_correct(u2_SQ_TP_lod_correct),
    .u3_SQ_TP_lod_correct(u3_SQ_TP_lod_correct),

    .u0_SQ_TP_pix_mask(u0_SQ_TP_pix_mask),
    .u1_SQ_TP_pix_mask(u1_SQ_TP_pix_mask),
    .u2_SQ_TP_pix_mask(u2_SQ_TP_pix_mask),
    .u3_SQ_TP_pix_mask(u3_SQ_TP_pix_mask),

    // stall
    .TP_SQ_fetch_stall(TP_SQ_fetch_stall),           // stall input from TP

    .busy(tis_busy),
```

```
    .clk(sclk_global),
    .reset(srst)
  );


  // -------------------------------------------------------------------
  // -------------------------------------------------------------------
  // -- ALU Instruction Pipe 0, Pipe 1, and ALU Pipe Output Mux --
  // -------------------------------------------------------------------
  // -------------------------------------------------------------------

  // - interconnect wires

  // - aif -
  wire  [0:0]                          aif0_instr_rts;
  wire [95:0]                          aif0_instr;
  wire [`SQ_VTX_CTL_PKT_WIDTH-1:0] aif0_ctl_pkt;
  wire [0:0]                           aif0_thread_type;
  wire [5:0]                           aif0_thread_id;
  wire [6:0]                           aif0_pc_base;

  wire                                 aif1_instr_rts;
  wire [95:0]                          aif1_instr;
  wire [`SQ_VTX_CTL_PKT_WIDTH-1:0] aif1_ctl_pkt;
  wire [5:0]                           aif1_thread_id;
```

```
  wire [6:0]                           aif1_pc_base;


  // - aiq -
  wire [6:0]   aiq0_pc_base;
  wire [63:0]  aiq0_valid_bits;
  wire [101:0] aiq0_instr;

  wire [2:0] aiq0_context_id;
  wire [5:0] aiq0_thread_id;

  wire [6:0]   aiq1_pc_base;
  wire [63:0]  aiq1_valid_bits;
  wire [101:0] aiq1_instr;

  wire [2:0] aiq1_context_id;
  wire [5:0] aiq1_thread_id;

  // - ais -
  wire [0:0] ais0_acs_rd_rts;
  wire [8:0] ais0_acs_rd_addr;
  wire [0:0] ais0_instr_start;
  wire [0:0] ais0_instr_stall;
  //wire [0:0] ais0_ld_isr;

```

```
    wire [0:0] ais1_acs_rd_rts;
    wire [8:0] ais1_acs_rd_addr;
    wire [0:0] ais1_instr_start;
    wire [0:0] ais1_instr_stall;
    //wire [0:0] ais1_ld_isr;

// this next line moved before sq_vism which now uses this signal
//   wire [7:0] acs_context_valid;

    //
    wire        acs_rd_rts;
    wire [8:0]   acs_rd_addr;
    wire [2:0]   acs_rd_context_id;
    wire [127:0] acs_rd_data;
    wire [0:0]     aiq0_rts;
    wire [0:0]     aiq1_rts;

    wire [1:0] aif0_export_info;
    wire [1:0] aif1_export_info;
    wire [1:0] aiq0_export_info;
    wire [1:0] aiq1_export_info;

    // -----------------------------
    // -- ALU Instruction Fetch 0 --
    // -----------------------------
```

```

    sq_target_instr_fetch
    #(
     `SQ_VTX_CTL_PKT_WIDTH    // ALU does not need lod correct bits from PIX ctl pkts
     )
    u0_sq_alu_instr_fetch
    (
      .target_strap         (`SQ_ALU0_STRAP),         //  hardwired  to  TEX_STRAP, ALU0_STRAP, or ALU1_STRAP

      // local registers
      .inst_base_vtx       (inst_base_vtx),          // vertex base
      .inst_base_pix       (inst_base_pix),          // pixel base

      // cfs interface
      .cfs_rts             (acfs0_rts),              // ctl packet and ptr are valid
      .cfs_ctl_pkt         (acfs0_ctl_pkt),          // the control packet (lod for pix_tex, valid_bits, gpr_base, context_id)
      .cfs_instr_ptr       (acfs0_tgt_instr_ptr), // the Instruction Store address of the first target instruction
      .cfs_instr_cnt       (acfs0_tgt_instr_cnt), // the number of instructions to be fetched
      .cfs_pc_base         (acfs0_state[6:0]),       // the param cache base (alloc'd in arbiter)
      .cfs_thread_type   (acfs0_thread_type),   // vertex or pixel
      .cfs_thread_id       (acfs0_status[21:16]), //
      //.cfs_pulse_sx       (acfs0_status[13]),      //
      //.cfs_export_id      (acfs0_state[7]),        // export_id that cfs0 is pushing down pipe 0 (sets global export_id)
```

```
      .cfs_export_info   ({acfs0_state[7], acfs0_status[13]}),  // export_info = {exp_id, pulse_sx}
      .cfs_last_in_thread(acfs0_status[12]),      // last instr in shader prog
      .tif_rtr                  (aif0_rtr),                   // AIF0 can take a new packet

      // instruction store interface
      .is_read_addr        (aif0_is_read_addr),    // instruction store read address
      .is_read_data        (is_read_data),// instruction store read data
      .is_phase              (is_phase),                 // instruction store phase
      .alu_phase (alu_phase),               //  tied  low  for  TEX  instance,  to  alu_phase  for  ALU instances

      // outputs to the target instruction decoder/queue
      .tif_pc_base_q       (aif0_pc_base),           // the target control packet (pipelined from input)
      .tif_ctl_pkt_q        (aif0_ctl_pkt),            // the target control packet (pipelined from input)
      .tif_export_info_q(aif0_export_info),      //
      .tif_last_in_thread_q(aif0_last_in_thread),          // last instruction flag
      .tif_last_in_group_q(aif0_last_in_group),           // last instruction flag
      .tif_thread_type_q(aif0_thread_type),      // 0: pixel, 1: vertex
      .tif_thread_id_q     (aif0_thread_id),                   //   the   target   thread_id   (pipelined   from input)
      .tif_instr_q(aif0_instr),                           // the target instruction data register (TIR)
      .tif_instr_rts_q      (aif0_instr_rts),                     // the target instr register is valid
      .tiq_rtr                 (aiq0_rtr),                  //  the  target  instr  queue  is  ready  to  take  the  AIR (and other pipeline data)

      .busy(aif0_busy),
```

```
      .clk(sclk_global),
      .reset(srst)
    );


    // -----------------------------
    // -- ALU Instruction Queue 0 --
    // -----------------------------

    parameter AIQ_NUM_WORDS = 4;
    parameter AIQ_ADDR_BITS = 2;

    sq_alu_instr_queue
    // #(AIQ_NUM_WORDS, AIQ_ADDR_BITS)
    u0_sq_alu_instr_queue
    (
      .write_rts   (aif0_instr_rts),
      .write_rtr   (aiq0_rtr),

      // AIQ inputs
      //write_data(),
      .aif_export_info    (aif0_export_info),
      .aif_pc_base_q      (aif0_pc_base),                     // param cache base
      .aif_last_in_thread_q(aif0_last_in_thread),         // last instruction flag
      .aif_last_in_group_q(aif0_last_in_group),           // last instruction flag
    .aif_thread_type_q(aif0_thread_type),     // 0: pixel, 1: vertex
    .aif_ctl_pkt_q       (aif0_ctl_pkt),            // control packet
    .aif_thread_id_q   (aif0_thread_id),         // thread_id
    .aif_instr_q           (aif0_instr),                 // instruction register (TIR)

    .read_rts               (aiq0_rts),
    .read_rtr               (ais0_rtr),
    //read_data(),                    // {control packet, clause num, instruction}

    // AIQ outputs
    .aiq_export_info   (aiq0_export_info),     // {exp_id, pulse_sx}
    .aiq_pc_base         (aiq0_pc_base),                     // param cache base addr
    .aiq_last_in_thread(aiq0_last_in_thread),// last instruction flag
    .aiq_last_in_group(aiq0_last_in_group), // last instruction flag

    .aiq_context_id     (aiq0_context_id),                 // context_id (from ctl packet)
    .aiq_valid_bits      (aiq0_valid_bits),        // valid bits (from ctl packet)
    .aiq_thread_id       (aiq0_thread_id),                  // thread_id
    .aiq_thread_type   (aiq0_thread_type),              // thread_id
    .aiq_instr   (aiq0_instr),               // instruction

    .clk(sclk_global),
    .reset(srst)
  );

```

```

  // ---------------------------------
  // -- ALU Instruction Sequencer 0 --
  // ---------------------------------

  sq_alu_instr_seq
  u0_sq_alu_instr_seq
  (
     .alu_strap  (LO),                           // tells whether alu0 or alu1

     // AIQ interface
     .aiq_rts                  (aiq0_rts),                  // rts from AIQ FIFO
     .aiq_export_info    (aiq0_export_info),     // {exp_id, pulse_sx}
     .aiq_last_in_thread(aiq0_last_in_thread),// last instruction flag
     .aiq_last_in_group(aiq0_last_in_group), // last instruction flag
     .aiq_context_id     (aiq0_context_id),                 // context_id (from ctl packet)
     .aiq_thread_id       (aiq0_thread_id),                  // thread_id
     .aiq_thread_type   (aiq0_thread_type),              // thread_id
     .aiq_instr   (aiq0_instr),               // instruction

     .ais_rtr(ais0_rtr),              // AIQ FIFO pop

     // phase inputs
     .gpr_phase (gpr_phase),
     .alu_phase (alu_phase),
```

```

    // AIS outputs to shader seq
    .ais_done              (ais0_done),
    .ais_thread_type_q(ais0_thread_type),
    .ais_thread_id_q   (ais0_thread_id),

    .ais_free_done                 (ais0_free_done),
    .ais_free_id_q                 (ais0_free_id),

    // to AIS output module
    .ais_instr_start      (ais0_instr_start),
    .ais_instr_stall      (ais0_instr_stall),
    .ais_acs_rd_rts      (ais0_acs_rd_rts),
    .ais_acs_rd_addr   (ais0_acs_rd_addr),

    //.aluconst_context_valid(acs_context_valid),

    .busy(ais0_busy),
    .clk(sclk_global),
    .reset(srst)
  );


  // -------------------------------
  // -- ALU Instruction Fetch  1  --
```

```
  // -------------------------------

  sq_target_instr_fetch
  #(
    `SQ_VTX_CTL_PKT_WIDTH
   )
  u1_sq_alu_instr_fetch
  (
     .target_strap          (`SQ_ALU1_STRAP),          //        hardwired        to        TEX_STRAP, ALU0_STRAP, or ALU1_STRAP

     // local registers
     .inst_base_vtx       (inst_base_vtx),          // vertex base
     .inst_base_pix       (inst_base_pix),          // pixel base (only in separate mode)

     // cfs interface
     .cfs_rts                  (acfs1_rts),                 // ctl packet and ptr are valid
     .cfs_ctl_pkt           (acfs1_ctl_pkt),                    //  the  control  packet  (lod  for  pix_tex, valid_bits, gpr_base, context_id)
     .cfs_instr_ptr        (acfs1_tgt_instr_ptr),  // the Instruction Store address of the first target instruction
     .cfs_instr_cnt        (acfs1_tgt_instr_cnt), // the number of instructions to be fetched
     .cfs_pc_base          (acfs1_state[6:0]),       // the param cache base (alloc'd in arbiter)
     //.cfs_pulse_sx       (acfs1_status[13]),      //
     //.cfs_export_id      (acfs1_state[7]),         // export_id that cfs0 is pushing down pipe 0 (sets global export_id)
     .cfs_export_info    ({acfs1_state[7], acfs1_status[13]}),  // export_info = {exp_id, pulse_sx}
    .cfs_thread_type   (acfs1_thread_type),   // vertex or pixel
    .cfs_thread_id     (acfs1_status[21:16]), //
    .cfs_last_in_thread(acfs1_status[12]),    // last instr in shader prog
    .tif_rtr           (aif1_rtr),            // AIF1 can take a new packet

    // instruction store interface
    .is_read_addr      (aif1_is_read_addr),   // instruction store read address
    .is_read_data      (is_read_data),// instruction store read data
    .is_phase          (is_phase),            // instruction store phase
    .alu_phase (alu_phase),    // tied low for TEX instance, to alu_phase for ALU instances

    // outputs to the target instruction decoder/queue
    .tif_pc_base_q     (aif1_pc_base),        // the target control packet (pipelined from input)
    .tif_ctl_pkt_q     (aif1_ctl_pkt),        // the target control packet (pipelined from input)
    .tif_export_info_q(aif1_export_info),     //
    .tif_last_in_group_q(aif1_last_in_group),        // last instruction flag
    .tif_last_in_thread_q(aif1_last_in_thread),      // last instruction flag
    .tif_thread_type_q(aif1_thread_type),     // 0: pixel, 1: vertex
    .tif_thread_id_q   (aif1_thread_id),             // the target thread_id (pipelined from input)
    .tif_instr_q(aif1_instr),                 // the target instruction data register (TIR)
    .tif_instr_rts_q   (aif1_instr_rts),             // the target instr register is valid
    .tiq_rtr           (aiq1_rtr),            // the target instr queue is ready to take the AIR (and other pipeline data)

    .busy(aif1_busy),
```

```
    .clk(sclk_global),
    .reset(srst)
  );


  // -----------------------------
  // -- ALU Instruction Queue 1 --
  // -----------------------------

  sq_alu_instr_queue
  //#(AIQ_NUM_WORDS, AIQ_ADDR_BITS)
  u1_sq_alu_instr_queue
  (
    .write_rts   (aif1_instr_rts),
    .write_rtr   (aiq1_rtr),

    // AIQ inputs
    //write_data(),
    .aif_export_info   (aif1_export_info),           // control packet
    .aif_pc_base_q     (aif1_pc_base),               // control packet
    .aif_last_in_thread_q(aif1_last_in_thread),      // last instruction flag
    .aif_last_in_group_q(aif1_last_in_group),        // last instruction flag
    .aif_ctl_pkt_q     (aif1_ctl_pkt),        // control packet
    .aif_thread_type_q(aif1_thread_type),     // 0: pixel, 1: vertex
    .aif_thread_id_q   (aif1_thread_id),      // thread_id
```

```
    .aif_instr_q       (aif1_instr),          // instruction register (TIR)

    .read_rts          (aiq1_rts),
    .read_rtr          (ais1_rtr),
    //read_data(),            // {control packet, clause num, instruction}

    // AIQ outputs
    .aiq_export_info   (aiq1_export_info),    // {exp_id, pulse_sx}
    .aiq_pc_base       (aiq1_pc_base),               // param cache base addr
    .aiq_last_in_thread(aiq1_last_in_thread),// last instruction flag
    .aiq_last_in_group(aiq1_last_in_group), // last instruction flag
    .aiq_context_id    (aiq1_context_id),            // context_id (from ctl packet)
    .aiq_valid_bits    (aiq1_valid_bits),     // valid bits (from ctl packet)
    .aiq_thread_type   (aiq1_thread_type),    // 0: pixel, 1: vertex
    .aiq_thread_id     (aiq1_thread_id),             // thread_id
    .aiq_instr   (aiq1_instr),          // instruction

    .clk(sclk_global),
    .reset(srst)
  );


  // ---------------------------------
  // -- ALU Instruction Sequencer 1 --
  // ---------------------------------
```

```

  sq_alu_instr_seq
  u1_sq_alu_instr_seq
  (
    .alu_strap  (HI),                   // whether ALU 0 or ALU 1

    // AIQ interface
    .aiq_rts           (aiq1_rts),            // rts from AIQ FIFO
    .aiq_export_info   (aiq1_export_info),    // {exp_id, pulse_sx}
    .aiq_last_in_thread(aiq1_last_in_thread),// last instruction flag
    .aiq_last_in_group(aiq1_last_in_group), // last instruction flag
    .aiq_context_id    (aiq1_context_id),     // context_id (from ctl packet)
    .aiq_thread_id     (aiq1_thread_id),      // thread_id
    .aiq_thread_type   (aiq1_thread_type),    // thread type (vtx/pix)
    .aiq_instr   (aiq1_instr),          // instruction

    .ais_rtr           (ais1_rtr),            // AIQ FIFO pop

    // phase inputs
    .gpr_phase (gpr_phase),          // GPR phase
    .alu_phase (alu_phase),

    // AIS outputs to shader seq
    .ais_done          (ais1_done),   //
    .ais_thread_type_q(ais1_thread_type),    //
    .ais_thread_id_q   (ais1_thread_id),        //

    .ais_free_done     (ais1_free_done),
    .ais_free_id_q     (ais1_free_id),

    // to AIS output module
    .ais_instr_start   (ais1_instr_start),
    .ais_instr_stall   (ais1_instr_stall),
    .ais_acs_rd_rts    (ais1_acs_rd_rts),
    .ais_acs_rd_addr   (ais1_acs_rd_addr),

    //.aluconst_context_valid(acs_context_valid),

    .busy(ais1_busy),
    .clk(sclk_global),
    .reset(srst)
  );


  // -----------------------------------------
  // -- ALU Instruction Sequencer Output Mux --
  // -----------------------------------------

  sq_ais_output
  u_sq_ais_output
```

```
  (
    // AIQ inputs
    .aiq0_export_id    (aiq0_export_info[1]),//
    .aiq0_pc_base      (aiq0_pc_base),              // param cache base ptr
    .aiq0_valid_bits   (aiq0_valid_bits),      // valid bits (from ctl packet)
    .aiq0_context_id   (aiq0_context_id),           // context_id (from ctl packet)
    .aiq0_instr (aiq0_instr),             // instruction
    .aiq0_gpr_rd_en    (aiq0_rts),

    .aiq1_export_id    (aiq1_export_info[1]),//
    .aiq1_pc_base      (aiq1_pc_base),
    .aiq1_valid_bits   (aiq1_valid_bits),
    .aiq1_context_id   (aiq1_context_id),
    .aiq1_instr (aiq1_instr),
    .aiq1_gpr_rd_en    (aiq1_rts),

    // AIS inputs
    .ais0_acs_rd_rts   (ais0_acs_rd_rts),      // alu const store read addr valid
    .ais0_acs_rd_addr (ais0_acs_rd_addr),    // alu constant store read address (from instr)
    .ais0_instr_start   (ais0_instr_start),     // just OR these guys before reg to SP
    .ais0_instr_stall   (ais0_instr_stall),
    //.ais0_ld_isr(LO),

    .ais1_acs_rd_rts   (ais1_acs_rd_rts),
    .ais1_acs_rd_addr (ais1_acs_rd_addr),
```

```
    .ais1_instr_start    (ais1_instr_start),
    .ais1_instr_stall    (ais1_instr_stall),
    //.ais1_ld_isr(LO),

    // other inputs that get muxed out to the SP or SX
    .tis_gpr_rd_addr   (tis_gpr_rd_addr),     // texture fetch read address
    .tis_gpr_rd_en     (tiq_rts),             // texture fetch read enable
    .ia_vertex_sel     (ia_vertex_sel),       // select VISM gpr write address vs. PISM gpr write address
    .vi_gpr_wr_addr   (vi_gpr_wr_addr),     // VISM gpr write address
    .vi_gpr_wr_en     (vi_gpr_wr_en),              // VISM gpr write enable
    .pi_gpr_wr_addr   (pi_gpr_wr_addr),     // PISM gpr write address
    .pi_gpr_wr_en     (pi_gpr_wr_en),              // PISM gpr write enable

    // phase inputs
    .gpr_phase(gpr_phase),
    .alu_phase(alu_phase),

    // ALU Const Store Interface
    .acs_rd_rts (acs_rd_rts),            // alu constant store read address valid
    .acs_rd_addr       (acs_rd_addr), // alu constant store read address
    .acs_rd_context_id          (acs_rd_context_id),   // alu constant store read state (context)
    .acs_rd_data       (acs_rd_data), // alu constant store read data

    // outputs to SP
    .SQ_SP_gpr_wr_addr(SQ_SP_gpr_wr_addr),
```

```
    .SQ_SP_gpr_wr_en        ({u0_SQ_SP_gpr_wr_en,                u1_SQ_SP_gpr_wr_en, u2_SQ_SP_gpr_wr_en, u3_SQ_SP_gpr_wr_en}),
    .SQ_SP_gpr_rd_addr(SQ_SP_gpr_rd_addr),
    .SQ_SP_gpr_rd_en        (SQ_SP_gpr_rd_en),
    .SQ_SP_gpr_phase        (SQ_SP_gpr_phase_mux),
    .SQ_SP_gpr_input_sel(SQ_SP_gpr_input_mux),
    .SQ_SP_gpr_channel_mask(SQ_SP_channel_mask),

    .SQ_SP_instr_start(SQ_SP_instruct_start),
    .SQ_SP_instr_stall(SQ_SP_stall),
    .SQ_SP_instr       (SQ_SP_instruct),
    .SQ_SP_const       (SQ_SP_const),

    //
    .SQ_SP_exporting(SQ_SP_exporting),
    .SQ_SP_exp_id     (SQ_SP_exp_id),
    .u0_SQ_SP_write_mask(u0_SQ_SP_pix_mask),
    .u1_SQ_SP_write_mask(u1_SQ_SP_pix_mask),
    .u2_SQ_SP_write_mask(u2_SQ_SP_pix_mask),
    .u3_SQ_SP_write_mask(u3_SQ_SP_pix_mask),

    // outputs to SX
    .SQ_SX_pc_wr_addr       (SQ_SX_pc_wr_addr),
    .SQ_SX_pc_wr_en         (SQ_SX_pc_wr_en),
    .SQ_SX_pc_channel_mask(SQ_SX_pc_channel_mask),

   .clk(sclk_global),
   .reset(srst)

 );


// ---------------------------------------------------------------------
// ---------------------------------------------------------------------
// -- RBBM Interface, Local Registers --
// ---------------------------------------------------------------------
// ---------------------------------------------------------------------

wire [31:0] rbi_data;
wire [14:0] rbi_addr;
wire [00:0] rbi_is_rts;
wire [00:0] rbi_is_rtr;
wire [00:0] rbi_tcs_rts;
wire [00:0] rbi_tcs_rtr;
wire [00:0] rbi_acs_rts;
wire [00:0] rbi_acs_rtr;
wire [00:0] rbi_draw_command;
wire [15:0] sq_busy_bits = {vtx_write_busy, pix_write_busy, vtx_tb_busy,
                            pix_tb_busy, tcfs_busy, acfs0_busy, acfs1_busy,
                            tif_busy, tis_busy, aif0_busy, aif1_busy,
                            ais0_busy, ais1_busy, aiq0_rts, aiq1_rts, tiq_rts};
```

```

sq_rbbm_interface
u_sq_rbbm_interface
(
  // RBBM
  .RBBM_a         (RBBM_a),
  .RBBM_wd             (RBBM_wd),
  .RBBM_we             (RBBM_we),
  .RBBM_be             (RBBM_be),
  .RBBM_re        (RBBM_re),
  .RBBM_SQ_soft_reset(RBBM_SQ_soft_reset),
  .SQ_RBBM_nrtrtr(SQ_RBBM_nrtrtr),
  .SQ_RBBM_rtr    (SQ_RBBM_rtr),
  .SQ_RBBM_cntx0_busy  (SQ_RBBM_cntx0_busy),
  .SQ_RBBM_cntx17_busy (SQ_RBBM_cntx17_busy),

  //RBBM     read data daisy chain
  .RBB_rs_in      (RBB_rs),
  .RBB_rs_out     (SQ_RBB_rs),
  .RBB_rd_in      (RBB_rd),
  .RBB_rd_out     (SQ_RBB_rd),

  // common
  .o_rbi_data     (rbi_data),
  .o_rbi_addr     (rbi_addr),
```

```
   .o_context_switch (rbi_draw_command),
   .i_sq_busy_bits   (sq_busy_bits),
   // aluconst
   .o_aluconst_rts     (rbi_acs_rts),
   .i_aluconst_rtr     (rbi_acs_rtr),

   // texconst
   .o_texconst_rts     (rbi_tcs_rts),
   .i_texconst_rtr     (rbi_tcs_rtr),

   // instr store
   .o_ins_rts  (rbi_is_rts),
   .i_ins_rtr   (rbi_is_rtr),

   // ctl flow const
   //.o_cfc_rts          (),
   .i_cfc_rtr   (LO),

   // local registers
   //.ps_num_reg_set          (ps_num_reg_set),
   //.vs_num_reg_set          (vs_num_reg_set),
   //.param_shade_set                (param_shade_set),

   //.o_ps_base_set           (ps_base_set),
   //.o_vs_base_set           (vs_base_set),
```

```
  .o_inst_base_vtx           (inst_base_vtx),
  .o_inst_base_pix           (inst_base_pix),
  .o_sq_wrapping_1_set     (sq_wrapping_1_set),
  .o_sq_wrapping_0_set     (sq_wrapping_0_set),
  //.o_sampling_mode_set   (sampling_mode_set),
  //.o_gen_index_set         (gen_index_set),
  //.o_param_gen_i0_set      (param_gen_i0_set),
  //.o_num_param_set         (vs_exp_count7_set),

  .i_clk                  (sclk_global),
  .i_reset                (srst),

// New State Register Outputs

.o_vs_program_base_set               (vs_program_base_set),
.o_vs_program_size_set               (),
.o_ps_program_base_set               (ps_program_base_set),
.o_ps_program_size_set               (),
.o_sq_cf_program_size_vs_cf_size_set     (),
.o_sq_cf_program_size_ps_cf_size_set     (),
.o_interpolator_cntl_param_shade_set      (param_shade_set),
.o_interpolator_cntl_sampling_pattern_set(),
.o_program_cntl_vs_num_reg_set  (vs_num_reg_set),
.o_program_cntl_ps_num_reg_set  (ps_num_reg_set),
.o_program_cntl_vs_resource_set  (vs_resource_set),
.o_program_cntl_ps_resource_set   (ps_resource_set),
.o_program_cntl_param_gen_i0_set          (param_gen_i0_set),
.o_program_cntl_gen_index_set     (gen_index_set),
.o_program_cntl_vs_export_count_set       (vs_export_count_set),
.o_program_cntl_vs_export_mode_set        (vs_export_mode_set),
.o_program_cntl_ps_export_mode_set        (ps_export_mode_set),
.o_wrapping_0_param_wrap_0_set(),
.o_wrapping_0_param_wrap_1_set(),
.o_wrapping_0_param_wrap_2_set(),
.o_wrapping_0_param_wrap_3_set(),
.o_wrapping_0_param_wrap_4_set(),
.o_wrapping_0_param_wrap_5_set(),
.o_wrapping_0_param_wrap_6_set(),
.o_wrapping_0_param_wrap_7_set(),
.o_wrapping_1_param_wrap_8_set(),
.o_wrapping_1_param_wrap_9_set(),
.o_wrapping_1_param_wrap_10_set           (),
.o_wrapping_1_param_wrap_11_set           (),
.o_wrapping_1_param_wrap_12_set           (),
.o_wrapping_1_param_wrap_13_set           (),
.o_wrapping_1_param_wrap_14_set           (),
.o_wrapping_1_param_wrap_15_set           (),
.o_vs_const_base_set                      (),
.o_vs_const_size_set                      (),
.o_ps_const_base_set                      (),
```

```
.o_ps_const_size_set                      (),
.o_context_misc_inst_pred_optimize_set(),
.o_cf_rd_base_rd_base_set            (),
.o_provoking_vtx_provoking_vtx_set        (),
.o_debug_misc_0_db_prob_on_set (),
.o_debug_misc_0_db_prob_break_set         (),
.o_debug_misc_0_db_prob_addr_set          (),
.o_debug_misc_0_db_prob_count_set         (),
.o_debug_misc_1_db_on_pix_set   (),
.o_debug_misc_1_db_on_vtx_set   (),
.o_debug_misc_1_db_inst_count_set         (),
.o_debug_misc_1_db_break_addr_set         ()
);

// ---------------------------------------------------------------------
// ---------------------------------------------------------------------
// -- Instruction Store, ALU and Texture Constant Stores --
// ---------------------------------------------------------------------
// ---------------------------------------------------------------------

wire [01:0] texconst_phase;

// -----------------------
// -- Instruction Store --
// -----------------------
```

```

sq_instruction_store
u_sq_inst_store
(
   // memory access phase control
   .i_is_phase(is_phase),
   .i_is_sub_phase     (is_subphase),

   // RBI
   .i_rbi_data (rbi_data),
   .i_rbi_addr(rbi_addr[14:0]),
   .i_rts                  (rbi_is_rts),
   .o_rtr                  (rbi_is_rtr),

   // SQ
   .i_tex_cf_addr      (tcfs_is_read_addr),
   .i_alu0_cf_addr     (acfs0_is_read_addr),
   .i_alu1_cf_addr     (acfs1_is_read_addr),
   .i_tex_addr         (tif_is_read_addr),
   .i_alu0_addr        (aif0_is_read_addr),
   .i_alu1_addr        (aif1_is_read_addr),

   .o_is_data  (is_read_data),

   .i_clk                  (sclk_global),
```

```
   .i_reset                (srst)
);



// ----------------------------
// -- Texture Constant Store --
// ----------------------------

sq_texconst_top
u_sq_texconst_top
(
   // from RBI
   .i_data_in  (rbi_data),
   .i_addr_in  (rbi_addr[7:3]),
   .i_rts                  (rbi_tcs_rts),
   .o_rtr                  (rbi_tcs_rtr),

   .i_context_switch_temp(rbi_draw_command),

   // From SQ
   .i_sq_read_laddr   (texconst_rd_addr),     // this really comes directly from TIQ (now passes
thru TIS)
   .i_sq_read_context(tiq_context_id),
   .i_texconst_phase  (texconst_phase),        // two cycles for read (0,1) and two for write (2,3)
   .o_read_data         (texconst_rd_data),

    //.o_context_valid (),

    .i_sq_context_done(pix_state_change),
    .i_sq_context        (pix_old_state),

    .i_clk                   (sclk_global),
    .i_reset                 (srst)
  );


  // ----------------------------
  // -- ALU Constant Store --
  // ----------------------------

  sq_aluconst_top
  u_sq_aluconst_top
  (
    // RBI interface
    .i_data_in  (rbi_data),
    .i_addr_in  (rbi_addr[10:4]),
    .i_rts                   (rbi_acs_rts),
    .o_rtr                   (rbi_acs_rtr),

    .i_context_switch_temp(rbi_draw_command),

```

```
    // AIS output read interface
    .i_sq_read_laddr   (acs_rd_addr), // this really comes directly from TIQ (now passes thru TIS)
    .i_sq_read_context(acs_rd_context_id),
    .o_read_data         (acs_rd_data),

    .o_context_valid   (acs_context_valid),

    .i_sq_context_done(pix_state_change),
    .i_sq_context        (pix_old_state),

    .i_clk                   (sclk_global),
    .i_reset                 (srst)
  );



  // --------------------------------------------------------------------
  // --------------------------------------------------------------------
  // -- Miscellaneous --
  // --------------------------------------------------------------------
  // --------------------------------------------------------------------

  // ----------------------
  // -- Phase Generation --
  // ----------------------

```

```
  sq_phase_gen
  u_sq_phase_gen
  (
    .gpr_phase (gpr_phase),
    .texconst_phase    (texconst_phase),
    .is_phase   (is_phase),
    .is_subphase         (is_subphase),
    .alu_phase (alu_phase),
    .cfs_phase (cfs_phase),
    .state_read_phase  (state_read_phase),

    .clk            (sclk_global),
    .reset                   (srst)
  );

/*
  always @(posedge sclk_global)
    begin
       $fsdbDumpMem(testbench.top.gc.shader_0.uvector0.umacc_gpr0.udum_mem.bram);
    end
*/

endmodule // sequencer_top


```

```
`include "header.v"
//--------------------------------------------------------------------------
//
// $Id: //depot/r400/devel/parts_lib/src/gfx/sq/ais/sq_ais_output.v#16 $
//
// $Change: 41217 $
//
// Copyright:   Trade secret of ATI Technologies, Inc.
//              © Copyright 2001-2002, ATI Technologies, Inc., (unpublished)
//
//              All rights reserved.  This notice is intended as a precaution against
//              inadvertent publication and does not imply publication or any waiver
//              of confidentiality.  The year included in the foregoing notice is the
//              year of creation of the work.
//
//--------------------------------------------------------------------------
////////////////////////////////////////////////////////////////////////////
// sq_ais_output.v
//
// - takes ALU instruction data from both AIQs
// - selects AIQ output to send to SP based on alu_phase
// - selects AIQ constant store read addr to send to ACS based on alu_phase
//
// - also takes tex instr's gpr read addr from TIQ
//
```

```
// issues:
// -
//
////////////////////////////////////////////////////////////////////////////
`include "sq_defs.v"

module sq_ais_output
(
  // inputs from the AIQs
  aiq0_export_id,      //
  aiq0_pc_base,        // param cache base
  aiq0_valid_bits,     // valid bits (from ctl packet)
  aiq0_context_id,             // state (context)
  aiq0_instr,          // instruction
  aiq0_gpr_rd_en,      //

  aiq1_export_id,      //
  aiq1_pc_base,        // param cache base
  aiq1_valid_bits,     // valid bits (from ctl packet)
  aiq1_context_id,             // state (context)
  aiq1_instr,          // instruction
  aiq1_gpr_rd_en,      //

  // inputs from the AISs
  ais0_acs_rd_rts,     // alu const store read addr valid
```

```
  ais0_acs_rd_addr,    // alu constant store read address (from instr)
  ais0_instr_start,    // just OR these guys before reg to SP
  ais0_instr_stall,

  ais1_acs_rd_rts,     // alu const store read addr valid
  ais1_acs_rd_addr,    // alu const store read addr
  ais1_instr_start,
  ais1_instr_stall,

  // other inputs that get muxed out to the SP or SX
  tis_gpr_rd_addr,     // texture fetch read address
  tis_gpr_rd_en,       // texture fetch read address
  ia_vertex_sel,       // select VISM gpr write address, enable to drive to SP when 1 (select pixel if 0)
  vi_gpr_wr_addr,      // VISM gpr write address
  vi_gpr_wr_en,        // VISM gpr write enable
  pi_gpr_wr_addr,      // PISM gpr write address
  pi_gpr_wr_en,        // VISM gpr write enable

  // phase inputs
  gpr_phase,           // GPR phase
  alu_phase,           // alu interleaving phase

  // ALU Constant Store interface
  acs_rd_rts,          // alu constant store read addr valid
  acs_rd_addr,         // alu constant store read addr
```

```
  acs_rd_context_id,   // alu constant store read context_id
  acs_rd_data,         // alu constant store read data

  // outputs to SP
  SQ_SP_gpr_wr_addr,
  SQ_SP_gpr_wr_en,
  SQ_SP_gpr_rd_addr,
  SQ_SP_gpr_rd_en,
  SQ_SP_gpr_phase,
  SQ_SP_gpr_input_sel,
  SQ_SP_gpr_channel_mask,

  SQ_SP_instr_start,
  SQ_SP_instr_stall,
  SQ_SP_instr,
  SQ_SP_const,

  //
  SQ_SP_exporting,
  SQ_SP_exp_id,
  u0_SQ_SP_write_mask,
  u1_SQ_SP_write_mask,
  u2_SQ_SP_write_mask,
  u3_SQ_SP_write_mask,

    // outputs to SX
    SQ_SX_pc_wr_addr,
    SQ_SX_pc_wr_en,
    SQ_SX_pc_channel_mask,

    clk,
    reset
  );

    // -- parameters --

    parameter LO = 1'b0;
    parameter HI = 1'b1;
    parameter X  = 1'bx;


    // ------------------------------------------------------------------------------------------------------
    // -- ios --
    // ------------------------------------------------------------------------------------------------------

    input [6:0]   aiq0_pc_base;
    input [63:0]  aiq0_valid_bits;
    input [2:0]   aiq0_context_id;
    input [101:0] aiq0_instr;
    input [0:0]   aiq0_gpr_rd_en;
```

```
    input [0:0]   aiq0_export_id;

    input [6:0]   aiq1_pc_base;
    input [63:0]  aiq1_valid_bits;
    input [2:0]   aiq1_context_id;
    input [101:0] aiq1_instr;
    input [0:0]   aiq1_gpr_rd_en;
    input [0:0]   aiq1_export_id;

    input [0:0]   ais0_acs_rd_rts;
    input [8:0]   ais0_acs_rd_addr;
    input [0:0]   ais0_instr_start;
    input [0:0]   ais0_instr_stall;

    input [0:0]   ais1_acs_rd_rts;
    input [8:0]   ais1_acs_rd_addr;
    input [0:0]   ais1_instr_start;
    input [0:0]   ais1_instr_stall;

    //
    input [6:0]    tis_gpr_rd_addr;
    input [0:0]    tis_gpr_rd_en;
    input [0:0]    ia_vertex_sel;
    input [6:0]    vi_gpr_wr_addr;
    input [0:0]    vi_gpr_wr_en;
```

```
    input [6:0]    pi_gpr_wr_addr;
    input [3:0]    pi_gpr_wr_en;


    //
    input [1:0] gpr_phase;
    input [0:0] alu_phase;

    //
    output         acs_rd_rts;
    output [8:0]   acs_rd_addr;
    output [2:0]   acs_rd_context_id;
    input  [127:0] acs_rd_data;

    //
    output [6:0] SQ_SP_gpr_wr_addr;
    output [3:0] SQ_SP_gpr_wr_en;
    output [6:0] SQ_SP_gpr_rd_addr;
    output [0:0] SQ_SP_gpr_rd_en;
    output [1:0] SQ_SP_gpr_phase;
    output [1:0] SQ_SP_gpr_input_sel;
    output [3:0] SQ_SP_gpr_channel_mask;

    output [0:0]   SQ_SP_instr_start;
    output [0:0]   SQ_SP_instr_stall;
```

```
    output [20:0]  SQ_SP_instr;
    output [127:0] SQ_SP_const;

    output [0:0] SQ_SP_exporting;
    output [0:0] SQ_SP_exp_id;
    output [3:0] u0_SQ_SP_write_mask;
    output [3:0] u1_SQ_SP_write_mask;
    output [3:0] u2_SQ_SP_write_mask;
    output [3:0] u3_SQ_SP_write_mask;

    output [6:0] SQ_SX_pc_wr_addr;
    output [0:0] SQ_SX_pc_wr_en;
    output [3:0] SQ_SX_pc_channel_mask;


    reg [6:0] SQ_SP_gpr_wr_addr;
    reg [3:0] SQ_SP_gpr_wr_en;
    reg [6:0] SQ_SP_gpr_rd_addr;
    reg [0:0] SQ_SP_gpr_rd_en;
    reg [1:0] SQ_SP_gpr_phase;
    reg [1:0] SQ_SP_gpr_input_sel;
    reg [3:0] SQ_SP_gpr_channel_mask;

    reg [0:0]   SQ_SP_instr_start;
    reg [0:0]   SQ_SP_instr_stall;
```

```
reg [20:0]  SQ_SP_instr;
reg [127:0] SQ_SP_const;

reg [0:0] SQ_SP_exporting;
reg [0:0] SQ_SP_exp_id;
reg [3:0] u0_SQ_SP_write_mask;
reg [3:0] u1_SQ_SP_write_mask;
reg [3:0] u2_SQ_SP_write_mask;
reg [3:0] u3_SQ_SP_write_mask;

reg [6:0] SQ_SX_pc_wr_addr;
reg [0:0] SQ_SX_pc_wr_en;
reg [3:0] SQ_SX_pc_channel_mask;

input clk;
input reset;


// -----------------------------------------------------------------------------------------------------
// -- internal signals --
// -----------------------------------------------------------------------------------------------------

// ISR - save this part of instruction when IQ is popped
// - needed for GPR result write and exports to PC
reg [7:0] isr_vector_dest_q;
```

```
reg [7:0] isr_scalar_dest_q;
reg [3:0] isr_vector_mask_q;
reg [3:0] isr_scalar_mask_q;
reg [1:0] isr_pred_sel_q;
reg [6:0] isr_pc_base_q;
reg [0:0] isr_instr_stall_q;

reg [7:0] isr_vector_dest_q1;
reg [7:0] isr_scalar_dest_q1;
reg [3:0] isr_vector_mask_q1;
reg [3:0] isr_scalar_mask_q1;
reg [1:0] isr_pred_sel_q1;
reg [6:0] isr_pc_base_q1;
reg [0:0] isr_instr_stall_q1;

//wire      scalar_export;
wire      scalar_export_pc;
//wire      vector_export;
wire      vector_export_pc;
wire      export;
wire      export_pc;


// -----------------------------------------------------------------------------------------------------
// -- module instatiations --
```

```
// -----------------------------------------------------------------------------------------------------


// -----------------------------------------------------------------------------------------------------
// -- combinational logic --
// -----------------------------------------------------------------------------------------------------

// - mux ACS read addr based on alu phase
//   - NOTE the addr is the opposite phase since const store read starts 4 cycles before the instr start

assign acs_rd_addr = alu_phase ? ais0_acs_rd_addr : ais1_acs_rd_addr;
assign acs_rd_context_id = alu_phase ? aiq0_context_id : aiq1_context_id;
//assign acs_rd_rts  = alu_phase ? ais1_acs_rd_rts  : ais0_acs_rd_rts;
assign acs_rd_rts  = ais1_acs_rd_rts | ais0_acs_rd_rts;


// - decode ISR instruction info for GPR writeback and Param Cache Writes

//assign scalar_export = ~isr_pred_sel_q1[1] & isr_scalar_dest_q1[7];
//assign vector_export = ~isr_pred_sel_q1[1] & isr_vector_dest_q1[7];
//assign scalar_export = isr_scalar_dest_q1[7];
//assign vector_export = isr_scalar_dest_q1[7];
assign export = isr_scalar_dest_q1[7];

assign scalar_export_pc = export & ~|(isr_scalar_dest_q1[5:4]);
```

```
assign vector_export_pc = export & ~|(isr_vector_dest_q1[5:4]);

assign export_pc = scalar_export_pc | vector_export_pc;


// -----------------------------------------------------------------------------------------------------
// -- registers --
// -----------------------------------------------------------------------------------------------------


// ----------------------------------------
// -- Instruction Input Staging Register --
// ----------------------------------------
// - holds the instruction data from the AIQ for use by GPR and PC writes (is reloaded by other thread
//   before GPR and PC writes occur, so relavent info must be kept here)
// - need to save stall to know whether to assert WE to gprs or PC also
// - must reload after every instruction even if AIS is idle to get the stall info saved
// - actually need two stages here since the AIQ must be popped for the next constant access

always @(posedge clk)
  begin
        if (reset)
          begin
        // stall forces a NOP to the shader pipe
             // - all instruction bits are don't care when stall == 1, so they don't need to be reset
             // - stall forces WE to GPR and PC to be deasserted
```

```
       //isr_scalar_dest_q <= 0;
       //isr_scalar_mask_q <= 0;
       //isr_vector_dest_q <= 0;
       //isr_vector_mask_q <= 0;
       //isr_pred_sel_q    <= 0;
       //isr_pc_base_q     <= 0;
           isr_instr_stall_q <= HI;
        end

      else if ( (gpr_phase == 2'b11) & (alu_phase == LO) ) //(ais0_ld_isr)
        begin
       isr_scalar_dest_q <= aiq0_instr[15:8];
       isr_vector_dest_q <= aiq0_instr[ 7:0];
       isr_scalar_mask_q <= aiq0_instr[23:20];
       isr_vector_mask_q <= aiq0_instr[19:16];
       isr_pred_sel_q    <= aiq0_instr[60:59];
       isr_pc_base_q     <= aiq0_pc_base;
       isr_instr_stall_q <= ais0_instr_stall;
        end

      else if ( (gpr_phase == 2'b11) & (alu_phase == HI) ) //(ais1_ld_isr)
        begin
       isr_scalar_dest_q <= aiq1_instr[15:8];
       isr_vector_dest_q <= aiq1_instr[ 7:0];
       isr_scalar_mask_q <= aiq1_instr[23:20];
```

```
           isr_vector_mask_q <= aiq1_instr[19:16];
           isr_pred_sel_q    <= aiq1_instr[60:59];
           isr_pc_base_q     <= aiq1_pc_base;
           isr_instr_stall_q <= ais1_instr_stall;
            end


          else
            begin
           isr_scalar_dest_q <= isr_scalar_dest_q;
               isr_vector_dest_q <= isr_vector_dest_q;
           isr_scalar_mask_q <= isr_scalar_mask_q;
               isr_vector_mask_q <= isr_vector_mask_q;
           isr_pred_sel_q    <= isr_pred_sel_q;
           isr_pc_base_q     <= isr_pc_base_q;
           isr_instr_stall_q <= isr_instr_stall_q;
            end
   end


 // ISR1 - need to pipe ISR0 to keep it around for the GPR/PC write

 always @(posedge clk)
   begin
          //if (reset)
            //begin
```

```
       //isr_scalar_dest_q <= 0;
       //isr_scalar_mask_q <= 0;
       //isr_vector_dest_q <= 0;
       //isr_vector_mask_q <= 0;
       //isr_pred_sel_q    <= 0;
       //isr_pc_base_q     <= 0;
           //isr_instr_stall_q <= HI;
        //end

      if ( (gpr_phase == 2'b11) )
        begin
       isr_scalar_dest_q1 <= isr_scalar_dest_q;
           isr_vector_dest_q1 <= isr_vector_dest_q;
       isr_scalar_mask_q1 <= isr_scalar_mask_q;
           isr_vector_mask_q1 <= isr_vector_mask_q;
       isr_pred_sel_q1    <= isr_pred_sel_q;
       isr_pc_base_q1     <= isr_pc_base_q;
       isr_instr_stall_q1 <= isr_instr_stall_q;
        end

      else
        begin
       isr_scalar_dest_q1 <= isr_scalar_dest_q1;
           isr_vector_dest_q1 <= isr_vector_dest_q1;
       isr_scalar_mask_q1 <= isr_scalar_mask_q1;
```

```
               isr_vector_mask_q1 <= isr_vector_mask_q1;
           isr_pred_sel_q1    <= isr_pred_sel_q1;
           isr_pc_base_q1     <= isr_pc_base_q1;
           isr_instr_stall_q1 <= isr_instr_stall_q1;
            end
   end



 // --------------------------------
 // -- SP instruction, write_mask --
 // --------------------------------
 // - valid with instruction start

 always @(posedge clk)
  begin
    case (gpr_phase)
     `SQ_SRCB_PHASE: begin
       case (alu_phase)
          LO: begin
         SQ_SP_instr    <=    {3'b000,    aiq0_instr[06:00],    aiq0_instr[55:48],    aiq0_instr[58],
aiq0_instr[101:99]};
         u0_SQ_SP_write_mask    <=    aiq0_valid_bits       [3:0];    u1_SQ_SP_write_mask    <=
aiq0_valid_bits  [7:4];
         u2_SQ_SP_write_mask    <=    aiq0_valid_bits    [11:8];    u3_SQ_SP_write_mask    <=
aiq0_valid_bits [15:12];
        end
```

```
          HI: begin
        SQ_SP_instr <= {aiq1_instr[07:00], aiq1_instr[55:48], aiq1_instr[58], aiq1_instr[101:99]};
        u0_SQ_SP_write_mask <= aiq1_valid_bits [3:0]; u1_SQ_SP_write_mask <= aiq1_valid_bits [7:4];
        u2_SQ_SP_write_mask <= aiq1_valid_bits [11:8]; u3_SQ_SP_write_mask <= aiq1_valid_bits [15:12];
        end
      endcase
    end
    `SQ_SRCC_PHASE: begin
      case (alu_phase)
          LO: begin
        SQ_SP_instr <= {aiq0_instr[15:08], aiq0_instr[47:40], aiq0_instr[57], aiq0_instr[98:96]};
        u0_SQ_SP_write_mask <= aiq0_valid_bits [19:16]; u1_SQ_SP_write_mask <= aiq0_valid_bits [23:20];
        u2_SQ_SP_write_mask <= aiq0_valid_bits [27:24]; u3_SQ_SP_write_mask <= aiq0_valid_bits [31:28];
        end
          HI: begin
        SQ_SP_instr <= {aiq1_instr[15:08], aiq1_instr[47:40], aiq1_instr[57], aiq1_instr[98:96]};
        u0_SQ_SP_write_mask <= aiq1_valid_bits [19:16]; u1_SQ_SP_write_mask <= aiq1_valid_bits [23:20];
        u2_SQ_SP_write_mask <= aiq1_valid_bits [27:24]; u3_SQ_SP_write_mask <= aiq1_valid_bits [31:28];
        end
      endcase
```

```
    end
    `SQ_FA_PHASE: begin
      case (alu_phase)
          LO: begin
        SQ_SP_instr <= {aiq0_instr[23:16], aiq0_instr[39:32], aiq0_instr[56], aiq0_instr[95:93]};
        u0_SQ_SP_write_mask <= aiq0_valid_bits [35:32]; u1_SQ_SP_write_mask <= aiq0_valid_bits [39:36];
        u2_SQ_SP_write_mask <= aiq0_valid_bits [43:40]; u3_SQ_SP_write_mask <= aiq0_valid_bits [47:44];
        end
          HI: begin
        SQ_SP_instr <= {aiq1_instr[23:16], aiq1_instr[39:32], aiq1_instr[56], aiq1_instr[95:93]};
        u0_SQ_SP_write_mask <= aiq1_valid_bits [35:32]; u1_SQ_SP_write_mask <= aiq1_valid_bits [39:36];
        u2_SQ_SP_write_mask <= aiq1_valid_bits [43:40]; u3_SQ_SP_write_mask <= aiq1_valid_bits [47:44];
        end
      endcase
    end
    `SQ_SRCA_PHASE: begin
      case (alu_phase)
          LO: begin
        SQ_SP_instr <= {aiq0_instr[23:16], aiq0_instr[25:24], aiq0_instr[31:26], aiq0_instr[92:88]};
        u0_SQ_SP_write_mask <= aiq0_valid_bits [51:48]; u1_SQ_SP_write_mask <= aiq0_valid_bits [55:52];
```

```
        u2_SQ_SP_write_mask <= aiq0_valid_bits [59:56]; u3_SQ_SP_write_mask <= aiq0_valid_bits [63:60];
        end
          HI: begin
        SQ_SP_instr <= {aiq1_instr[23:16], aiq1_instr[25:24], aiq1_instr[31:26], aiq1_instr[92:88]};
        u0_SQ_SP_write_mask <= aiq1_valid_bits [51:48]; u1_SQ_SP_write_mask <= aiq1_valid_bits [55:52];
        u2_SQ_SP_write_mask <= aiq1_valid_bits [59:56]; u3_SQ_SP_write_mask <= aiq1_valid_bits [63:60];
        end
      endcase
    end
   endcase
  end


  // SQ_SP_instr_start
  // SQ_SP_instr_stall
  // SQ_SP_exporting
  // SQ_SP_exp_id
  //
  always @(posedge clk)
    begin
    case (alu_phase)
     LO:
       begin
```

```
       SQ_SP_instr_start <= ais0_instr_start;
       SQ_SP_instr_stall <= ais0_instr_stall;
       // logic for exporting = ~pred_sel[1] & (scalar_dest[7] | vector_dest[7])
       // new logic for exporting = scalar_dest[7]
       // remove this stall mux
         if ( ais0_instr_stall )
        SQ_SP_exporting   <= LO;
         else
        //SQ_SP_exporting   <= ~aiq0_instr[60] & (aiq0_instr[15] | aiq0_instr[7]);
        SQ_SP_exporting   <= aiq0_instr[15];
       SQ_SP_exp_id <= aiq0_export_id;  //
       end
     HI:     // if interleaving is disabled, need to make sure controls are not driven
       begin
       SQ_SP_instr_start <= ais1_instr_start;
       SQ_SP_instr_stall <= ais1_instr_stall;
       // remove this stall mux
         if ( ais1_instr_stall ) SQ_SP_exporting   <= LO;
         else                    SQ_SP_exporting   <= aiq1_instr[15];
       SQ_SP_exp_id <= aiq1_export_id;  //
       end
    endcase
   end


```

```
  // SQ_SP_gpr_phase
  // SQ_SP_gpr_input_sel
  //
  always @(posedge clk)
    begin
     SQ_SP_gpr_phase <= gpr_phase;
     SQ_SP_gpr_input_sel <= {ia_vertex_sel, ~ia_vertex_sel}; // 00: cnt, 01: pix, 10: vtx (fix needed for count)
    end


  // --------------------------------------
  // -- SP gpr read address, read enable --
  // --------------------------------------
  // - the read address comes directly from the ALU or Texture Instruction Queue (IQ)
  // - the read address was calculated prior to being loaded into the IQ
  // - the read enable is just the RTS out of the IQ
/*
  reg [0:0] aiq_gpr_rd_en;

  always @(alu_phase or aiq0_gpr_rd_en or aiq1_gpr_rd_en)
    begin
      case (alu_phase)
           LO: aiq_gpr_rd_en = aiq0_gpr_rd_en;
           HI: aiq_gpr_rd_en = aiq1_gpr_rd_en;
      endcase
```

```
    end
*/

  always @(posedge clk)
    begin
      case (gpr_phase)
       `SQ_SRCA_PHASE: begin
         case (~alu_phase)                    // have to invert this to get the srcA addr in a cycle early
              LO: SQ_SP_gpr_rd_addr <= aiq0_instr[86:80];
              HI: SQ_SP_gpr_rd_addr <= aiq1_instr[86:80];
         endcase
         case (~alu_phase)                    // have to invert this to get the srcA addr in a cycle early
              LO: SQ_SP_gpr_rd_en <= aiq0_gpr_rd_en;
              HI: SQ_SP_gpr_rd_en <= aiq1_gpr_rd_en;
         endcase
       end
       `SQ_SRCB_PHASE: begin
         case (alu_phase)
              LO: SQ_SP_gpr_rd_addr <= aiq0_instr[78:72];
              HI: SQ_SP_gpr_rd_addr <= aiq1_instr[78:72];
         endcase
         case (alu_phase)
              LO: SQ_SP_gpr_rd_en <= aiq0_gpr_rd_en;
              HI: SQ_SP_gpr_rd_en <= aiq1_gpr_rd_en;
```

```
         endcase
       end
       `SQ_SRCC_PHASE: begin
         case (alu_phase)
              LO: SQ_SP_gpr_rd_addr <= aiq0_instr[70:64];
              HI: SQ_SP_gpr_rd_addr <= aiq1_instr[70:64];
         endcase
         case (alu_phase)
              LO: SQ_SP_gpr_rd_en <= aiq0_gpr_rd_en;
              HI: SQ_SP_gpr_rd_en <= aiq1_gpr_rd_en;
         endcase
       end
       `SQ_FA_PHASE: begin
            SQ_SP_gpr_rd_addr <= tis_gpr_rd_addr;
         SQ_SP_gpr_rd_en   <= tis_gpr_rd_en;
       end
      endcase
    end


  // -----------------------------------------------------
  // -- SP gpr write address, write enable, channel mask --
  // -----------------------------------------------------
  // - don't have to use alu_phase as a select here since it was used to load ISR
  // - the write address was calcualted prior to being loaded into the Instruction Queue
```

```

  always @(posedge clk)
    begin
      SQ_SP_gpr_wr_en <= 0; // clear WE by default
      case (gpr_phase)
       `SQ_PS_PHASE:
           begin
          SQ_SP_gpr_wr_addr <= isr_scalar_dest_q1[6:0];
          SQ_SP_gpr_channel_mask <= isr_scalar_mask_q1;
          //if ( ~scalar_export & ~isr_instr_stall_q1)
          if ( ~export & ~isr_instr_stall_q1)
             SQ_SP_gpr_wr_en <= 4'b1111;          // assert gpr write enable when not exporting and not stalling
         end
       `SQ_PV_PHASE:
           begin
             SQ_SP_gpr_wr_addr <= isr_vector_dest_q1[6:0];
          SQ_SP_gpr_channel_mask <= isr_vector_mask_q1;
          if (~export & ~isr_instr_stall_q1)
             SQ_SP_gpr_wr_en <= 4'b1111;
         end
       `SQ_ID_PHASE:
         begin
          case (ia_vertex_sel)
              LO: begin
             SQ_SP_gpr_wr_addr <= pi_gpr_wr_addr;
```

```
                SQ_SP_gpr_wr_en   <= pi_gpr_wr_en;
                  end
                HI: begin
                SQ_SP_gpr_wr_addr <= vi_gpr_wr_addr;
                SQ_SP_gpr_wr_en   <= {4{vi_gpr_wr_en}};
                  end
            endcase
            SQ_SP_gpr_channel_mask <= 4'b1111;
          end
        `SQ_FD_PHASE: begin
              SQ_SP_gpr_wr_addr <= 7'b0;
              SQ_SP_gpr_wr_en   <= 4'b0;
          SQ_SP_gpr_channel_mask <= 4'b0;
        end
      endcase
    end


  // -------------------------------
  // -- PC Write Address Register --
  // -------------------------------
  // - load when phase is PS (this lines up with the ISR load and IQ pop, i.e. the last cycle the data
  //   from the IQ is valid for this instruction)

  reg [6:0] pc_wr_addr_q;
```

```
  reg [6:0] pc_wr_addr_q1;
  reg [6:0] pc_wr_addr_q2;
  reg         pc_wr_en_q;
  reg         pc_wr_en_q1;
  reg         pc_wr_en_q2;
  reg [3:0] pc_channel_mask_q;
  reg [3:0] pc_channel_mask_q1;
  reg [3:0] pc_channel_mask_q2;

  wire      ld_pc_wr_addr = (gpr_phase == `SQ_PS_PHASE);

  always @(posedge clk)
    begin
          case ( ld_pc_wr_addr )
        HI:
          begin
              case ( scalar_export_pc )
              HI:
            begin
              pc_wr_addr_q <= isr_pc_base_q1 + isr_scalar_dest_q1[5:0];
              pc_channel_mask_q <= isr_scalar_mask_q1;
            end
              LO:
            begin
              pc_wr_addr_q <= isr_pc_base_q1 + isr_vector_dest_q1[5:0];
```

```
                pc_channel_mask_q <= isr_vector_mask_q1;
              end
                endcase
          end
        LO:
          begin
                case ( pc_wr_en_q )
                  HI: pc_wr_addr_q <= pc_wr_addr_q + 1;
                  LO: pc_wr_addr_q <= pc_wr_addr_q;
                endcase
            pc_channel_mask_q <= pc_channel_mask_q;
          end
            endcase // case( ld_pc_wr_addr )
      end


  // SX param cache write address
  // SX param cache write enable
  // SX param cache channel mask
  // - these guys need to be delayed 2 cycles due to the block interface delay btwn SP and SX
  always @(posedge clk)
    begin
          pc_wr_addr_q1 <= pc_wr_addr_q;
          pc_wr_addr_q2 <= pc_wr_addr_q1;
          SQ_SX_pc_wr_addr <= pc_wr_addr_q2;      // these all need to go back to q1 ...
```

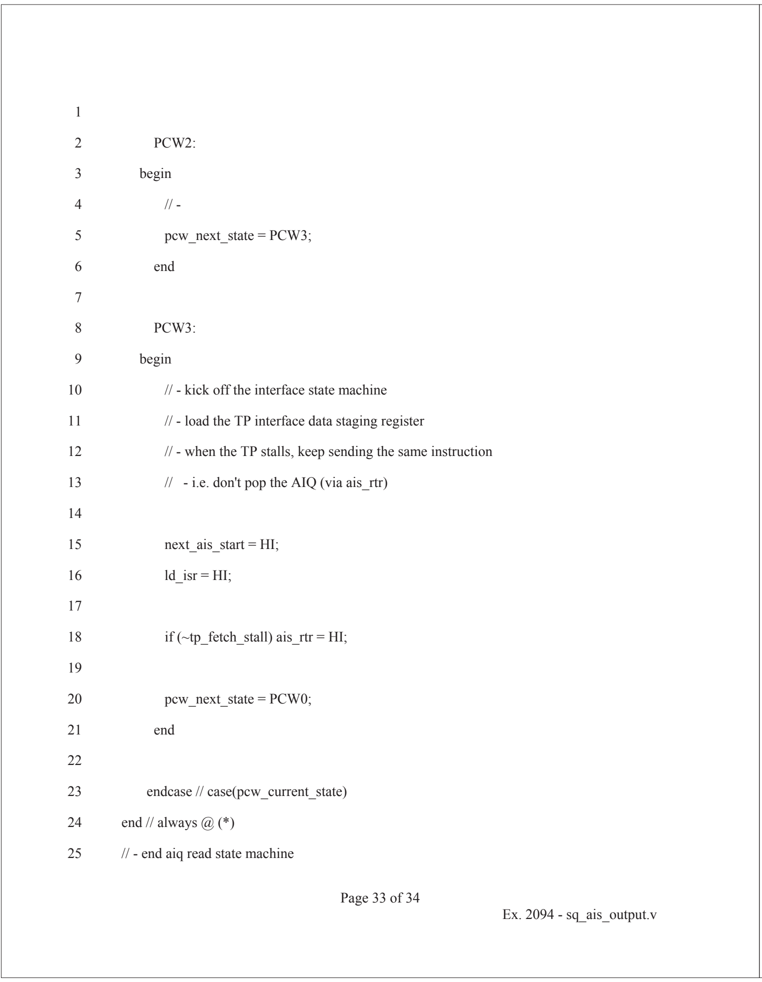
```

          pc_wr_en_q1 <= pc_wr_en_q;
          pc_wr_en_q2 <= pc_wr_en_q1;
          SQ_SX_pc_wr_en <= pc_wr_en_q2;

      pc_channel_mask_q1 <= pc_channel_mask_q;
      pc_channel_mask_q2 <= pc_channel_mask_q1;
      SQ_SX_pc_channel_mask <= pc_channel_mask_q2;
    end


  // ALU Constant Store read data

  always @(posedge clk)
    begin
          SQ_SP_const = acs_rd_data;
    end


  // misc interface registers
/*
  wire ais_fetch_stall = tp_fetch_stall;  // need to add the correct timing for ais_fetch_stall
                                          // why does it not go directly from TPC to SPs?
  always @(posedge clk)
    begin
```

```
            tp_fetch_stall <= TP_SQ_fetch_stall;
        end
*/


    // ----------------------------------------------------------------------------------------------------
    // -- one-bit state machines --
    // ----------------------------------------------------------------------------------------------------

    // pc write enable
    always @(posedge clk)
      begin
        if (reset)
          pc_wr_en_q <= LO;
        else
          case ( pc_wr_en_q )
            LO:
              // - set when loading pc_wr_addr and we're exporting to the PC and we're not stalling
              // - ld_pc_wr_addr is asserted based on phase above for PC write addr register
              if ( ld_pc_wr_addr & export_pc & ~isr_instr_stall_q1 )
                 pc_wr_en_q <= HI;
            HI:
              // - clear when loading pc_wr_addr and we're not exporting to the PC or we're stalling
              if ( ld_pc_wr_addr & (~export_pc | isr_instr_stall_q1) )
```

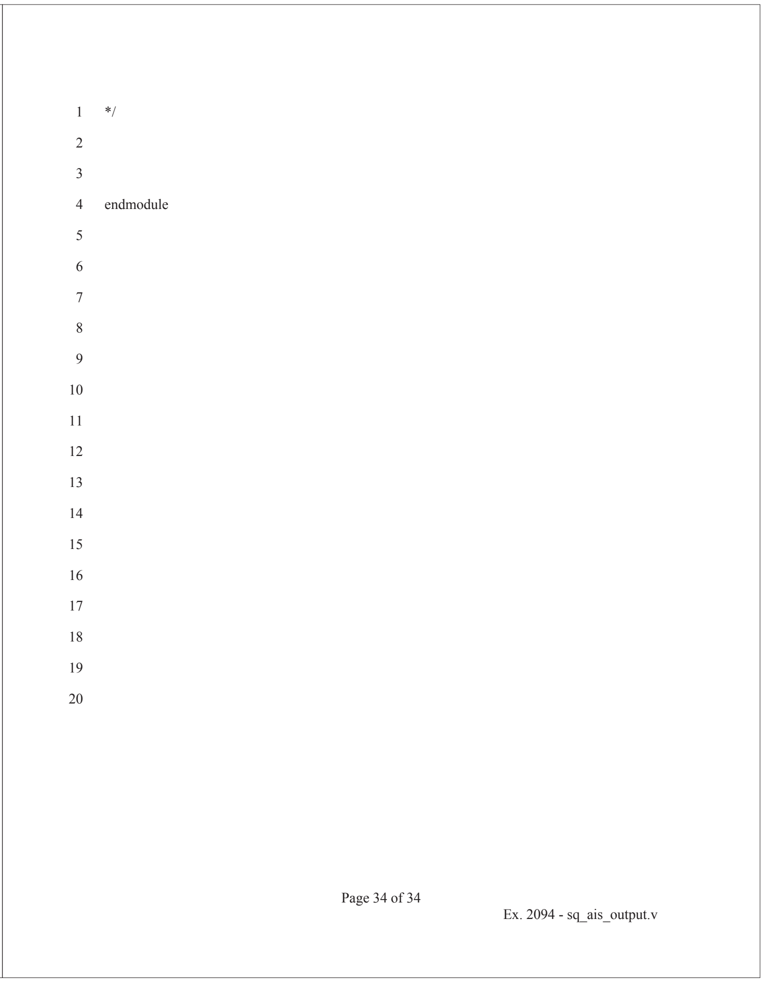
```
                  pc_wr_en_q <= LO;
            endcase
      end


    // ----------------------------------------------------------------------------------------------------
    // -- state machines --
    // ----------------------------------------------------------------------------------------------------

    // Parameter Cache Write State Machine:
    // - load either scalar or vector param cache address
    // - increment param cache address based on gpr phase and whether scalar or vector is exporting
    //   to the param cache (can't both export to param cache in same instruction)
/*
    parameter PCW0     = 2'b00;
    parameter PCW1     = 2'b01;
    parameter PCW2     = 2'b10;
    parameter PCW3     = 2'b11;

    reg [1:0] pcw_current_state;
    reg [1:0] pcw_next_state;

    // reg'd outputs

    // un-reg'd (combinatorial) outputs
```

```
    //reg inc_pc_wr_addr;
    //reg ld_isr;

    // state and output registers
    always @(posedge clk)
       begin
         if (reset)
           begin
             pcw_current_state <= PCW0;
            ais_start <= LO;
          end
         else
           begin
             pcw_current_state <= pcw_next_state;
            ais_start <= next_ais_start;
          end
       end

    // next state logic
    always @(
            aiq_rts or gpr_phase or tp_fetch_stall or
              pcw_current_state
           )
      begin
        // default assignments
```

```
            pcw_next_state = PCW0;
          next_ais_start = LO;

          ais_rtr = LO;               // this guy pops the AIQ
            ld_isr = LO;              // this guy loads the ISR

            case (pcw_current_state)
              PCW0:
          begin
              // - wait until the AIQ output is valid and the GPR read phase is at the Fetch Address phase
              // - the GPR read addr goes from the AIQ to the output phase mux, and is selected
              //   when phase == `SQ_FA
              // - the aluconst read address also goes directly from the AIQ

              if (aiq_rts & (gpr_phase == `SQ_FA_PHASE))
                begin
               pcw_next_state = PCW1;
             end
            end

              PCW1:
          begin
              // -
              pcw_next_state = PCW2;
            end
```



```

      PCW2:
    begin
        // -
        pcw_next_state = PCW3;
      end

      PCW3:
    begin
        // - kick off the interface state machine
        // - load the TP interface data staging register
        // - when the TP stalls, keep sending the same instruction
        //   - i.e. don't pop the AIQ (via ais_rtr)

        next_ais_start = HI;
        ld_isr = HI;

        if (~tp_fetch_stall) ais_rtr = HI;

        pcw_next_state = PCW0;
      end

    endcase // case(pcw_current_state)
end // always @ (*)
// - end aiq read state machine
```



```
*/


endmodule
```

```
`include "header.v"
//---------------------------------------------------------------------------
//
// $Id: //depot/r400/devel/parts_lib/src/gfx/sq/ais/sq_alu_instr_queue.v#14 $
//
// $Change: 41796 $
//
// Copyright:   Trade secret of ATI Technologies, Inc.
//              © Copyright 2001-2002, ATI Technologies, Inc., (unpublished)
//
//              All rights reserved.  This notice is intended as a precaution against
//              inadvertent publication and does not imply publication or any waiver
//              of confidentiality.  The year included in the foregoing notice is the
//              year of creation of the work.
//
//---------------------------------------------------------------------------
///////////////////////////////////////////////////////////////////////////////
// sq_alu_instr_queue.v
//
// Alu Instruction Queue
//
// - instantiates fifo_regs and a simple fifo controller for the queue
// - calculates source and destination GPR addresses from gpr_base, which is part
//   of the input ctl packet, and from offset in the instruction (absoulte mode)
//   or from offset and loop index (relative mode)
```

```
//
// issues:
// - relative mode not implemented yet
// - could this just be a ping-pong buffer?
// - does context_id_id need to go into the IQ? (currently it does, but can think of no use)
//
///////////////////////////////////////////////////////////////////////////////
`include "sq_defs.v"

module sq_alu_instr_queue
(
  write_rts,
  write_rtr,

  // inputs from AIF (ALU Instruction Fetch)
  aif_export_info,      // {export_id, pulse_sx}
  aif_pc_base_q,        // param cache base addr
  aif_last_in_group_q,          // last instruction in series of instructions (done sent back fron ais)
  aif_last_in_thread_q,// last instruction in the whole shader program (no done sent back fron ais)
  aif_thread_type_q,    // vector type (0: pixel, 1: vertex)
  aif_thread_id_q,      // thread id
  aif_ctl_pkt_q,        // control packet (valid_bits, gpr_base, context_id)
  aif_instr_q,          // instruction register (registered read from IS - 96 bits)

  read_rts,
```

```
  read_rtr,
  //read_data,          // {control packet, clause num, instruction}

  // outputs to AIS
  aiq_export_info,      //
  aiq_pc_base,          //
  aiq_last_in_group,    //
  aiq_last_in_thread,   //
  aiq_thread_type,      //
  aiq_context_id,
  aiq_valid_bits,
  aiq_thread_id,
  aiq_instr,            // instruction

  clk,
  reset
);

  // -- parameters --

  parameter NUM_WORDS = 4;
  parameter ADDR_BITS = 2;

  // aiq_export_info + aiq_pc_base + aiq_last_in_group
  // + aiq_last_in_thread + aiq_thread_type + aiq_context + aiq_valid_bits + aiq_thread_id + aiq_instr
```

```

  parameter DATA_BITS = 2 + 7 + 1 + 1 + 1 + 3 + 64 + 6 + 102;

  parameter LO = 1'b0;
  parameter HI = 1'b1;
  parameter X  = 1'bx;



  // -----------------------------------------------------------------------------------------------------
  // -- ios --
  // -----------------------------------------------------------------------------------------------------

  input                         write_rts;
  output                        write_rtr;

  input [95:0]                  aif_instr_q;
  input [`SQ_VTX_CTL_PKT_WIDTH-1:0]         aif_ctl_pkt_q;
  input [1:0]                   aif_export_info;
  input [6:0]                   aif_pc_base_q;
  input [5:0]                   aif_thread_id_q;
  input [0:0]                   aif_last_in_group_q;
  input [0:0]                   aif_last_in_thread_q;
  input [0:0]                   aif_thread_type_q;

  output                        read_rts;
```

ATI 2095
LG v. ATI
IPR2015-00326

```
input                       read_rtr;

output [1:0]        aiq_export_info;
output [6:0]        aiq_pc_base;
output [0:0]        aiq_last_in_group;
output [0:0]        aiq_last_in_thread;
output [0:0]        aiq_thread_type;
output [5:0]        aiq_thread_id;
output [63:0]       aiq_valid_bits;
output [2:0]        aiq_context_id;
output [101:0]              aiq_instr;

input                       clk;
input                       reset;


// ----------------------------------------------------------------------------------------------------------
// -- internal signals --
// ----------------------------------------------------------------------------------------------------------

wire [ADDR_BITS-1:0]    write_ptr_q;
wire                write_en;

wire [ADDR_BITS-1:0]    read_ptr_q;

```

```
wire [DATA_BITS-1:0]    write_data;
wire [DATA_BITS-1:0]    read_data;



// ----------------------------------------------------------------------------------------------------------
// -- combinational logic --
// ----------------------------------------------------------------------------------------------------------

wire [63:0] aif_valid_bits = aif_ctl_pkt_q[73:10];
wire [6:0] gpr_base       = aif_ctl_pkt_q[9:3];
wire [2:0] aif_context_id  = aif_ctl_pkt_q[2:0];

wire [0:0] src_a_type = aif_instr_q[95];
wire [0:0] src_b_type = aif_instr_q[94];
wire [0:0] src_c_type = aif_instr_q[93];

wire [2:0] src_a_sel = {aif_instr_q[95], aif_instr_q[87:86]};
wire [2:0] src_b_sel = {aif_instr_q[94], aif_instr_q[79:78]};
wire [2:0] src_c_sel = {aif_instr_q[93], aif_instr_q[71:70]};

wire [4:0] vector_opcode = aif_instr_q[92:88];

wire [7:0] src_a_offset = aif_instr_q[87:80];
wire [7:0] src_b_offset = aif_instr_q[79:72];
wire [7:0] src_c_offset = aif_instr_q[71:64];
```

```

wire [5:0] scalar_dest_offset = aif_instr_q[13:8];
wire [5:0] vector_dest_offset = aif_instr_q[5:0];

wire [0:0] scalar_dest_addr_mode = aif_instr_q[14];
wire [0:0] vector_dest_addr_mode = aif_instr_q[6];

wire [1:0] pred_sel = aif_instr_q[60:59];
wire [0:0] scalar_dest_pred_sel = aif_instr_q[15];
wire [0:0] vector_dest_pred_sel = aif_instr_q[7];

// --> old emulator export decode
//wire [0:0] scalar_export = ~pred_sel[1] & scalar_dest_pred_sel;
//wire [0:0] vector_export = ~pred_sel[1] & vector_dest_pred_sel;

// --> new emulator export decode
wire [0:0] scalar_export = aif_instr_q[15];
wire [0:0] vector_export = aif_instr_q[15];


// - calculate gpr read addresses (src) and gpr write addresses (dst)
//   - if type is "constant", just pass the C number
//   - if type is "register", add the gpr base
//   - if exporting, do not add gpr base since dest is in this case an export address

```

```
wire [7:0]     src_a_addr = src_a_type ? src_a_offset[5:0] + gpr_base : src_a_offset;
wire [7:0]     src_b_addr = src_b_type ? src_b_offset[5:0] + gpr_base : src_b_offset;
wire [7:0]     src_c_addr = src_c_type ? src_c_offset[5:0] + gpr_base : src_c_offset;

wire [6:0]     scalar_dest_addr = scalar_export ? {LO, scalar_dest_offset} : scalar_dest_offset
+ gpr_base;
wire [6:0]     vector_dest_addr    =    vector_export    ?    {LO,    vector_dest_offset}    :
vector_dest_offset + gpr_base;

// - substitute actual gpr addresses into src and dest instruction fields
// - concatenate AIF input data and gpr addresses into fifo write data

wire [101:0]  instruction =
        {
        src_a_sel, src_b_sel, src_c_sel,
        vector_opcode, src_a_addr, src_b_addr, src_c_addr,
        aif_instr_q[63:61], pred_sel, aif_instr_q[58:16],
     scalar_dest_pred_sel, scalar_dest_addr, vector_dest_pred_sel, vector_dest_addr
        };

assign write_data = {aif_export_info, aif_pc_base_q,
               aif_last_in_group_q, aif_last_in_thread_q, aif_thread_type_q,
               aif_context_id, aif_valid_bits, aif_thread_id_q,
               instruction};

// - connect fifo read data to individual AIQ outputs
```

```

    assign aiq_export_info = read_data[DATA_BITS-1:DATA_BITS-2];
    assign aiq_pc_base     = read_data[DATA_BITS-1-2:DATA_BITS-2-7];
    assign aiq_last_in_group  = read_data[DATA_BITS-1-2-7];
    assign aiq_last_in_thread  = read_data[DATA_BITS-1-2-7-1];
    assign aiq_thread_type = read_data[DATA_BITS-1-2-7-1-1];
    assign aiq_context_id  = read_data[DATA_BITS-1-2-7-1-1-1:DATA_BITS-2-7-1-1-1-3];
    assign  aiq_valid_bits    =  read_data[DATA_BITS-1-2-7-1-1-1-3:DATA_BITS-2-7-1-1-1-3-64];
    assign aiq_thread_id   = read_data[DATA_BITS-1-2-7-1-1-1-3-64:DATA_BITS-2-7-1-1-1-3-64-6];
    assign aiq_instr       = read_data[101:0];
  /*
    assign aiq_pc_base     = read_data[181:175];
    assign aiq_last_in_group  = read_data[174];
    assign aiq_last_in_thread  = read_data[173];
    assign aiq_thread_type = read_data[172];
    assign aiq_context_id  = read_data[171:169];
    assign aiq_valid_bits  = read_data[168:105];
    assign aiq_thread_id   = read_data[104:102];
    assign aiq_instr       = read_data[101:0];
  */
    // -----------------------------------------------------------------------------------------------------
    // -- module instatiations --
    // -----------------------------------------------------------------------------------------------------

```

```
    // fifo control for based fifo

    fifo_regs_ctl
     #(
       NUM_WORDS, ADDR_BITS
      )
    u_fifo_regs_ctl
      (
       .write_rts(write_rts),                   // in
       .write_rtr(write_rtr),                   // out
       .write_ptr_q(write_ptr_q),
       .write_en(write_en),

       .read_rts_q(read_rts),                   // out
       .read_rtr(read_rtr),            // in
       .read_ptr_q(read_ptr_q),

       //.used_slots(used_count),            // out

       .clk(clk),
       .reset(reset)
      );


    // fifo storage
```

```

    fifo_regs
     #(
       ADDR_BITS, DATA_BITS, NUM_WORDS
      )
    u_fifo_regs
      (
       .wr_addr(write_ptr_q),
       .wr_en(write_en),
       .wr_data(write_data),

       .rd_addr(read_ptr_q),
       .rd_data(read_data),

       .clk(clk)
      );



    // -----------------------------------------------------------------------------------------------------
    // -- one-bit state machines --
    // -----------------------------------------------------------------------------------------------------



```

```
endmodule
```

```
`include "header.v"
//---------------------------------------------------------------------------
//
// $Id: //depot/r400/devel/parts_lib/src/gfx/sq/ais/sq_alu_instr_seq.v#13 $
//
// $Change: 44201 $
//
// Copyright:   Trade secret of ATI Technologies, Inc.
//              © Copyright 2001-2002, ATI Technologies, Inc., (unpublished)
//
//              All rights reserved.  This notice is intended as a precaution against
//              inadvertent publication and does not imply publication or any waiver
//              of confidentiality.  The year included in the foregoing notice is the
//              year of creation of the work.
//
//---------------------------------------------------------------------------
/////////////////////////////////////////////////////////////////////////////
// sq_alu_instr_seq.v
//
// - receives instruction from alu instr queue (AIQ)
// - sends instruction to SP on the correct phase
// - sends instruction to SP over four cycles
//
// issues:
// -
```

```
//
/////////////////////////////////////////////////////////////////////////////
`include "sq_defs.v"

module sq_alu_instr_seq
(
  alu_strap,            // for alu0 vs alu1

  // inputs from AIQ
  aiq_rts,              // rts from AIQ FIFO
  aiq_export_info,      // {exp_id, pulse_sx}
  aiq_last_in_group,    // last instruction in group (but not usually the last in the shader program)
  aiq_last_in_thread,   // last instruction in shader program
  aiq_context_id,               // context_id
  aiq_thread_type,      //
  aiq_thread_id,        //
  aiq_instr,            // instruction

  ais_rtr,              // AIQ FIFO pop

  // phase inputs
  gpr_phase,            //
  alu_phase,            //

  // interface back to thread buffers
```

```
  ais_done,
  ais_thread_type_q,
  ais_thread_id_q,

  // interface back to SX via exp_alloc
  ais_free_done,        // export buffer dealloc to SX
  ais_free_id_q,                // export buffer id

  // to ais_output
  ais_instr_start,
  ais_instr_stall,
  //ais_ld_isr,
  ais_acs_rd_rts,       // alu constant store read address valid
  ais_acs_rd_addr,      // alu constant store read address (to ais_output)

  //aluconst_context_valid,

  busy,
  clk,
  reset
);

  // -- parameters --

  parameter LO = 1'b0;
```

```
  parameter HI = 1'b1;
  parameter X  = 1'bx;


  // -----------------------------------------------------------------------------------
  // -- ios --
  // -----------------------------------------------------------------------------------

  input     alu_strap;

  //
  input [1:0] aiq_export_info;// {exp_id, pulse_sx}
  input [0:0] aiq_last_in_group;        // last instruction flag
  input [0:0] aiq_last_in_thread;       // last instruction flag
  input [0:0] aiq_thread_type;// 0: pixel, 1: vertex
  input [2:0] aiq_context_id;  // context_id (from ctl packet)
  input [5:0] aiq_thread_id;   // clause number
  input [101:0] aiq_instr;      // instruction
  input [0:0] aiq_rts;
  output [0:0] ais_rtr;
  reg [0:0] ais_rtr;

  //
  input [1:0] gpr_phase;        //
  input [0:0] alu_phase;        //
```

```

//
output [0:0]  ais_acs_rd_rts;// alu constant store read address (logical addr - up to 256 consts)
reg [0:0]  ais_acs_rd_rts;     // alu constant store read address (logical addr - up to 256 consts)

//
output [0:0] ais_free_done; // export buffer dealloc to SX
output [0:0] ais_free_id_q;             // export buffer id

output [0:0]  ais_thread_type_q;
output [5:0]  ais_thread_id_q;
output [0:0]  ais_done;
reg [0:0]  ais_thread_type_q;
reg [5:0]  ais_thread_id_q;
reg [0:0]  ais_done;
reg [0:0]  ais_free_done;

//
output [0:0]  ais_instr_start;
output [0:0]  ais_instr_stall;
output [8:0]  ais_acs_rd_addr;        // alu constant store read address (logical addr - up to 512 consts)

reg [0:0]  ais_instr_start;
reg [0:0]  ais_instr_stall;
reg [8:0]  ais_acs_rd_addr;  // alu constant store read address (logical addr - up to 512 consts)
```

```

//input [7:0] aluconst_context_valid;


output busy;
input clk;
input reset;


// ---------------------------------------------------------------------------------------
// -- internal signals --
// ---------------------------------------------------------------------------------------

reg [2:0] acs_current_state;
reg [2:0] ais_current_state;

reg ais_start;

wire ca_fetch = ~aiq_instr[101];
wire cb_fetch = ~aiq_instr[98];
wire cc_fetch = ~aiq_instr[95];

wire [8:0] ca_addr = {1'b0, aiq_instr[87:80]};
wire [8:0] cb_addr = {1'b0, aiq_instr[79:72]};
wire [8:0] cc_addr = {1'b0, aiq_instr[71:64]};
```

```

//reg [2:0]  ais_state_q;

// ---------------------------------------------------------------------------------------
// -- module instatiations --
// ---------------------------------------------------------------------------------------


// ---------------------------------------------------------------------------------------
// -- combinational logic --
// ---------------------------------------------------------------------------------------

wire busy = (|acs_current_state) | (|ais_current_state);

// - constant store valid (one bit per context from ALU Const Store)

//wire [0:0] aluconst_valid;

//assign     aluconst_valid = aluconst_context_valid[aiq_context_id];

// ---------------------------------------------------------------------------------------
// -- registers --
// ---------------------------------------------------------------------------------------

// ----------------------------
```

```
// -- Input Staging Register --
// ----------------------------
// - need to send the vector type and the thread_id back to the thread buffers when
//   the all the instructions we wanted to run for this thread are done (this will
//   cause the thread to become valid again)
// - register this info in from the AIS on an AIQ pop in order to hold it until the
//   AIS is done

reg [0:0]  ais_last_in_group_q;
reg [0:0]  ais_last_in_thread_q;
reg [0:0]  ais_free_id_q;
reg [0:0]  ais_pulse_sx_q;

always @(posedge clk)
  begin
      if ( ais_rtr )
        begin
          ais_thread_type_q <= aiq_thread_type;
          ais_thread_id_q  <= aiq_thread_id;
          //ais_context_id_q       <= aiq_context_id;
          ais_last_in_group_q  <= aiq_last_in_group;
          ais_last_in_thread_q  <= aiq_last_in_thread;
          ais_free_id_q  <= aiq_export_info[1];
          ais_pulse_sx_q  <= aiq_export_info[0];
        end
        else
         begin
            ais_thread_type_q <= ais_thread_type_q;
            ais_thread_id_q  <= ais_thread_id_q;
            //ais_context_id_q       <= ais_context_id_q;
            ais_last_in_group_q  <= ais_last_in_group_q;
            ais_last_in_thread_q  <= ais_last_in_thread_q;
            ais_free_id_q  <= ais_free_id_q;
            ais_pulse_sx_q  <= ais_pulse_sx_q;
         end
 end


// --------------------------------------------------------------------------------------
// -- one-bit state machines --
// --------------------------------------------------------------------------------------


// --------------------------------------------------------------------------------------
// -- state machines --
// --------------------------------------------------------------------------------------

// - the following two state machines work together in a staged manner (1st triggers 2nd)
// - each cycles thru 8 states to match the 8 cycle transfer to the SP
//
```

```
 // ACS (ALU Constant Store) Read State Machine:
 //   - assert the ACS read rts
 //   - select the ACS read address from the 3 source addresses present in the instruction
 //   - kick off the AIS stage
 //
 // AIS (ALU Instruction Sequencer) State Machine:
 //   - assert instruction start
 //    - load the ISR in the ais_output module (to hold dest write addrs after AIQ has been popped)
 //   - pop the AIQ

 // ACS read state machine

 parameter ACS0      = 3'b000;
 parameter ACS1      = 3'b001;
 parameter ACS2      = 3'b010;
 parameter ACS3      = 3'b011;
 parameter ACS4      = 3'b100;
 parameter ACS5      = 3'b101;
 parameter ACS6      = 3'b110;
 parameter ACS7      = 3'b111;

 reg [2:0] acs_next_state;

 // state and output registers
 always @(posedge clk)
```

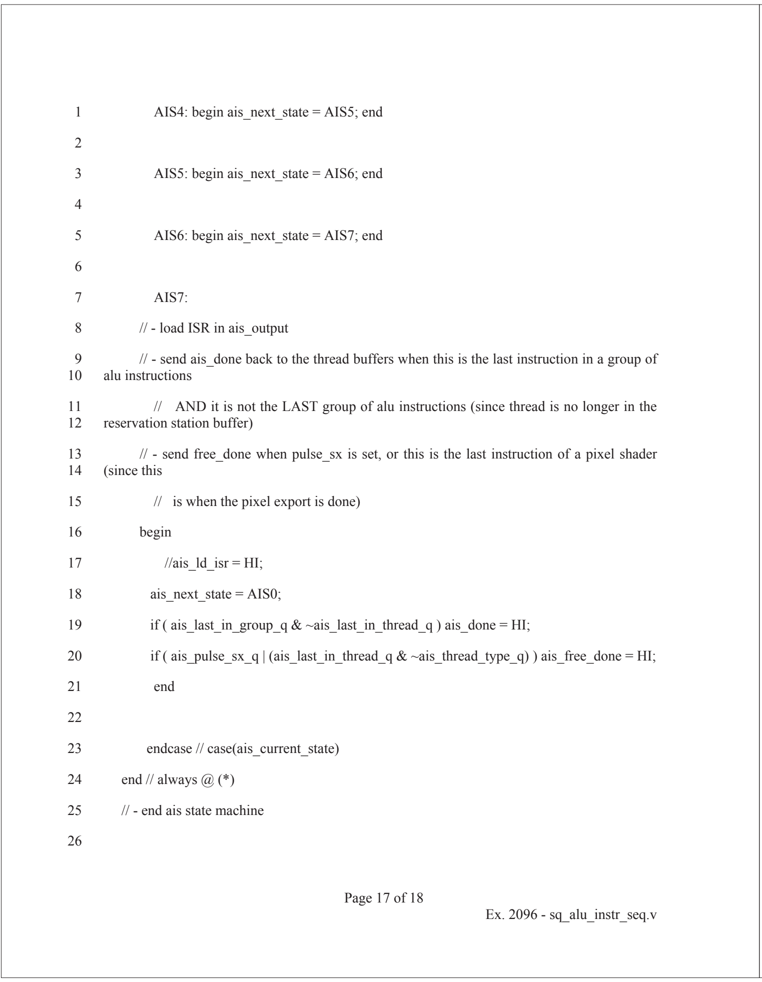
```
  begin
      if (reset)
       begin
          acs_current_state <= ACS0;
     end
      else
       begin
          acs_current_state <= acs_next_state;
     end
  end

// next state logic
always @(
     aiq_rts or gpr_phase or
     ca_fetch or ca_addr or cb_fetch or cb_addr or cc_fetch or cc_addr or
         acs_current_state
     )
 begin
      // default assignments
      acs_next_state = ACS0;

    ais_start = LO;
    ais_acs_rd_rts = LO;       // read request valid to the alu const store
    ais_rtr = LO;
      ais_acs_rd_addr = ca_addr;
```

```

         case (acs_current_state)
           ACS0:
       // - start when AIQ output is valid and gpr read phase is srcB
           //    (this phase is chosen due to ACS read latency - if latency changes, this also must change)
       // - also need to start on the correct alu_phase (opposite of alu_strap since this is starting
           //    4 cycles early)
       begin
              if (aiq_rts & (gpr_phase == `SQ_SRCB_PHASE) & (alu_phase == ~alu_strap) )// & aluconst_valid )
                begin
             if (ca_fetch)                       // read first constant
                      begin
                        ais_acs_rd_rts = HI;
                        ais_acs_rd_addr = ca_addr;
                      end
             acs_next_state = ACS1;
          end
         end


           ACS1:
            begin
          if (cb_fetch)
                   begin
                      ais_acs_rd_rts = HI;           // read second constant
                ais_acs_rd_addr = cb_addr;
              end
        acs_next_state = ACS2;
        end

      ACS2:
        begin
        if (cc_fetch)
              begin
                ais_acs_rd_rts = HI;        // read third constant
                ais_acs_rd_addr = cc_addr;
              end
          acs_next_state = ACS3;
        end

      ACS3: begin acs_next_state = ACS4; end

      ACS4:
    begin
          ais_start = HI;
          acs_next_state = ACS5;
      end

      ACS5: begin acs_next_state = ACS6; end

```

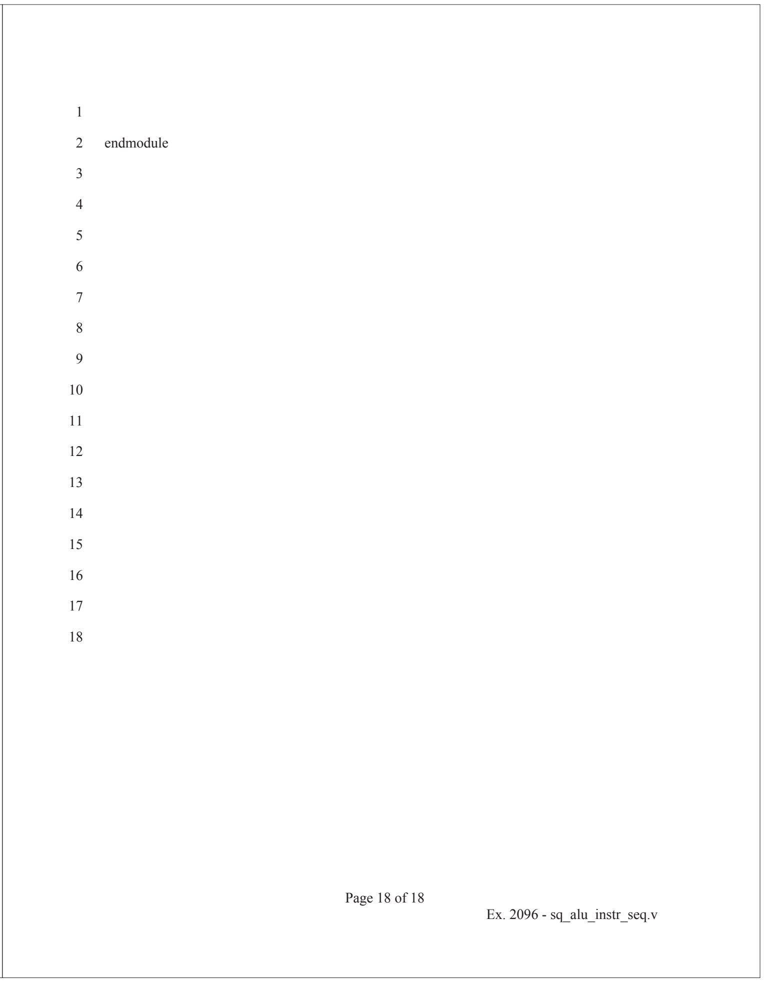
```
          ACS6: begin acs_next_state = ACS7; end

          ACS7:
      // - pop the AIQ
        begin
            ais_rtr = HI;
          acs_next_state = ACS0;
        end

        endcase // case(acs_current_state)
  end // always @ (*)
  // - end acs read state machine


// AIS state machine

parameter AIS0 = 3'b000;
parameter AIS1 = 3'b001;
parameter AIS2 = 3'b010;
parameter AIS3 = 3'b011;
parameter AIS4 = 3'b100;
parameter AIS5 = 3'b101;
parameter AIS6 = 3'b110;
parameter AIS7 = 3'b111;

```

```
//reg [2:0] ais_current_state;
reg [2:0] ais_next_state;

// state and output registers
always @(posedge clk)
  begin
      if (reset)
        begin
          ais_current_state <= AIS0;
      end
        else
        begin
          ais_current_state <= ais_next_state;
      end
  end

// next state logic
always @(
      ais_start or
        ais_current_state
      )
  begin
      // default assignments
      ais_next_state = AIS0;

```

```
    ais_instr_start = LO;
    ais_instr_stall = LO;
    //ais_ld_isr = LO;
    ais_done = LO;
    ais_free_done = LO;

      case (ais_current_state)
        AIS0:
    begin
          // - wait until this machine is started by the AIQ read SM
        ais_instr_stall = HI;
          if (ais_start)
            begin
              ais_instr_start = HI;
            ais_instr_stall = LO;
            ais_next_state = AIS1;
          end
        end

        AIS1: begin ais_next_state = AIS2; end

        AIS2: begin ais_next_state = AIS3; end

        AIS3: begin ais_next_state = AIS4; end

```



```
        AIS4: begin ais_next_state = AIS5; end

        AIS5: begin ais_next_state = AIS6; end

        AIS6: begin ais_next_state = AIS7; end

        AIS7:
      // - load ISR in ais_output
      // - send ais_done back to the thread buffers when this is the last instruction in a group of alu instructions
        //   AND it is not the LAST group of alu instructions (since thread is no longer in the reservation station buffer)
      // - send free_done when pulse_sx is set, or this is the last instruction of a pixel shader (since this
        //   is when the pixel export is done)
      begin
          //ais_ld_isr = HI;
        ais_next_state = AIS0;
        if ( ais_last_in_group_q & ~ais_last_in_thread_q ) ais_done = HI;
        if ( ais_pulse_sx_q | (ais_last_in_thread_q & ~ais_thread_type_q) ) ais_free_done = HI;
        end

       endcase // case(ais_current_state)
   end // always @ (*)
   // - end ais state machine
```



```
endmodule
```

```
`include "header.v"
//---------------------------------------------------------------------------
//
// $Id: //depot/r400/devel/parts_lib/src/gfx/sq/ca/sq_thread_arb.v#19 $
//
// $Change: 43237 $
//
// Copyright:   Trade secret of ATI Technologies, Inc.
//              © Copyright 2001-2002, ATI Technologies, Inc., (unpublished)
//
//              All rights reserved.  This notice is intended as a precaution against
//              inadvertent publication and does not imply publication or any waiver
//              of confidentiality.  The year included in the foregoing notice is the
//              year of creation of the work.
//
//---------------------------------------------------------------------------
/////////////////////////////////////////////////////////////////////////////
// sq_thread_arb.v
//
// - pick a thread from requests being sent by the Vertex and Pixel Thread Buffers
// - thread state and status is selected by the winner in each thread buffer and sent
//   back to the arbiter for the final muxing (btwn vtx and pix) to the CFS
//
// issues:
// -
```

```
//
/////////////////////////////////////////////////////////////////////////////

`include "../misc/sq_defs.v"

module sq_thread_arb
(
  arb_type_strap,       // tex = 1, alu = 0
  state_read_phase,     // share read access between tex and alu arbs

  // vertex and pixel thread buffer interface

  vtx_req_q,            // 16 vtx_thread_buff requests

  vtx_winner_q,         // winning vertex thread_id sent back to Vertex Thread Buffer
  vtx_winner_ack,       // request acknowledge - indicates to TB that the winner is valid

  vtx_state,            //
  vtx_status,           //

  pix_req_q,            // 16 pix_thread_buff requests

  pix_winner_q,         // winning pixel thread_id sent back to Pixel Thread Buffer
  pix_winner_ack,       //

```

```
  pix_state,            //
  pix_status,           //

  // control flow sequencer interface

  arb_rts0,      // ready to send the winner to CFS0
  arb_rts1,      // ready to send the winner to CFS1
  arb_state,     // the state sent to the CFS
  arb_status,    // the status sent to the CFS
  arb_thread_type,      // vtx or pix

  cfs_rtr0,      // CFS0 can accept a thread
  cfs_rtr1,      // CFS1 can accept a thread (for alu cfs's)

  cfs1_enable,          // enable sending packets to CFS1 (this a local register setting: SQ_FLOW_CTL.ONE_ALU)

  clk,
  reset
);

  // -- parameters --

  parameter STATE_WIDTH = 8;    //
  parameter STATUS_WIDTH = 8;    //

```

```
  parameter LO = 1'b0;
  parameter HI = 1'b1;
  parameter X  = 1'bx;



  // -----------------------------------------------------------------------------------------------
  // -- ios --
  // -----------------------------------------------------------------------------------------------

  input [0:0] arb_type_strap;
  input [0:0] state_read_phase;

  input  [15:0]                     vtx_req_q;
  input  [`SQ_VTX_STATE_WIDTH-1:0]   vtx_state;
  input  [`SQ_VTX_STATUS_WIDTH-1:0] vtx_status;

  output [3:0]                      vtx_winner_q;
  output [0:0]                      vtx_winner_ack;

  input  [15:0]                     pix_req_q;
  input  [`SQ_PIX_STATE_WIDTH-1:0]    pix_state;
  input  [`SQ_PIX_STATUS_WIDTH-1:0]  pix_status;

  output [3:0]                      pix_winner_q;
  output [0:0]                      pix_winner_ack;
```

```

    // interface to the control flow sequencer
    output [0:0]                          arb_rts0;
    output [0:0]                          arb_rts1;
    output [STATE_WIDTH-1:0]              arb_state;
    output [STATUS_WIDTH-1:0]             arb_status;
    output [0:0]                          arb_thread_type;
    input  [0:0]                          cfs_rtr0;
    input  [0:0]                          cfs_rtr1;

    input  [0:0]                          cfs1_enable;

    input clk;
    input reset;


    // -----------------------------------------------------------------------------------------------------
    // -- internal signals --
    // -----------------------------------------------------------------------------------------------------

    reg [3:0] vtx_winner_q;
    reg [3:0] pix_winner_q;

    reg [0:0] vtx_winner_vld_q;
    reg [0:0] pix_winner_vld_q;
```

```

    reg [STATE_WIDTH-1:0]          arb_state;
    reg [STATUS_WIDTH-1:0]                 arb_status;

    // one bit state machine
    reg cfs_turn;

    // thread_type_sm
    reg [0:0] type_winner;
    reg [0:0] vtx_winner_ack;
    reg [0:0] pix_winner_ack;

    // thread_read_sm
    reg [0:0] ld_winner;
    reg [0:0] arb_rts;


    // -----------------------------------------------------------------------------------------------------
    // -- module instatiations --
    // -----------------------------------------------------------------------------------------------------


    // -----------------------------------------------------------------------------------------------------
    // -- combinational logic --
    // -----------------------------------------------------------------------------------------------------
```

```

    // * arbiter *
    // - the highest thread number has the highest priority, and verts have priority over pixels
    // - try with loop/tree coding style

  /*
    wire [31:0] req_vector = {vs_req, ps_req};
    reg [0:0]   winner_vld;
    reg [4:0]   winner;

    always @(req_vector)
      begin
         casez (req_vector)
           32'b1???_????_????_????_????_????_????_????: begin winner_vld = HI; winner = 5'h1f; end
           32'b01??_????_????_????_????_????_????_????: begin winner_vld = HI; winner = 5'h1e; end
           32'b001?_????_????_????_????_????_????_????: begin winner_vld = HI; winner = 5'h1d; end
           32'b0001_????_????_????_????_????_????_????: begin winner_vld = HI; winner = 5'h1c; end
           32'b0000_1???_????_????_????_????_????_????: begin winner_vld = HI; winner = 5'h1b; end
           32'b0000_01??_????_????_????_????_????_????: begin winner_vld = HI; winner = 5'h1a; end
           32'b0000_001?_????_????_????_????_????_????: begin winner_vld = HI; winner = 5'h19; end
           32'b0000_0001_????_????_????_????_????_????: begin winner_vld = HI; winner = 5'h18; end
```

```
           32'b0000_0000_1???_????_????_????_????_????: begin winner_vld = HI; winner = 5'h17; end
           32'b0000_0000_01??_????_????_????_????_????: begin winner_vld = HI; winner = 5'h16; end
           32'b0000_0000_001?_????_????_????_????_????: begin winner_vld = HI; winner = 5'h15; end
           32'b0000_0000_0001_????_????_????_????_????: begin winner_vld = HI; winner = 5'h14; end
           32'b0000_0000_0000_1???_????_????_????_????: begin winner_vld = HI; winner = 5'h13; end
           32'b0000_0000_0000_01??_????_????_????_????: begin winner_vld = HI; winner = 5'h12; end
           32'b0000_0000_0000_001?_????_????_????_????: begin winner_vld = HI; winner = 5'h11; end
           32'b0000_0000_0000_0001_????_????_????_????: begin winner_vld = HI; winner = 5'h10; end
           32'b0000_0000_0000_0000_1???_????_????_????: begin winner_vld = HI; winner = 5'h0f; end
           32'b0000_0000_0000_0000_01??_????_????_????: begin winner_vld = HI; winner = 5'h0e; end
           32'b0000_0000_0000_0000_001?_????_????_????: begin winner_vld = HI; winner = 5'h0d; end
           32'b0000_0000_0000_0000_0001_????_????_????: begin winner_vld = HI; winner = 5'h0c; end
           32'b0000_0000_0000_0000_0000_1???_????_????: begin winner_vld = HI; winner = 5'h0b; end
           32'b0000_0000_0000_0000_0000_01??_????_????: begin winner_vld = HI; winner = 5'h0a; end
           32'b0000_0000_0000_0000_0000_001?_????_????: begin winner_vld = HI; winner = 5'h09; end
           32'b0000_0000_0000_0000_0000_0001_????_????: begin winner_vld = HI; winner = 5'h08; end
```

```
          32'b0000_0000_0000_0000_0000_0000_1???_????: begin winner_vld = HI; winner = 5'h07; end
          32'b0000_0000_0000_0000_0000_0000_01??_????: begin winner_vld = HI; winner = 5'h06; end
          32'b0000_0000_0000_0000_0000_0000_001?_????: begin winner_vld = HI; winner = 5'h05; end
          32'b0000_0000_0000_0000_0000_0000_0001_????: begin winner_vld = HI; winner = 5'h04; end
          32'b0000_0000_0000_0000_0000_0000_0000_1???: begin winner_vld = HI; winner = 5'h03; end
          32'b0000_0000_0000_0000_0000_0000_0000_01??: begin winner_vld = HI; winner = 5'h02; end
          32'b0000_0000_0000_0000_0000_0000_0000_001?: begin winner_vld = HI; winner = 5'h01; end
          32'b0000_0000_0000_0000_0000_0000_0000_0001: begin winner_vld = HI; winner = 5'h00; end
          32'b0000_0000_0000_0000_0000_0000_0000_0000: begin winner_vld = LO; winner = 5'h00; end  // winner is really don't care here
          default:                              begin winner_vld = X;  winner = {5{X}}; end
           endcase
     end // always @ (req_vector)
*/

   // - vertex request priority encoder

   reg          vtx_winner_vld;
   reg [3:0]   vtx_winner;

   always @(vtx_req_q)
```

```
     begin
        casez (vtx_req_q)
          16'b0000_0000_0000_0000: begin vtx_winner_vld = LO; vtx_winner = 4'hf; end
          16'b1000_0000_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'hf; end
          16'b?100_0000_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'he; end
          16'b??10_0000_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'hd; end
          16'b???1_0000_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'hc; end
          16'b????_1000_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'hb; end
          16'b????_?100_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'ha; end
          16'b????_??10_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'h9; end
          16'b????_???1_0000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'h8; end
          16'b????_????_1000_0000: begin vtx_winner_vld = HI; vtx_winner = 4'h7; end
          16'b????_????_?100_0000: begin vtx_winner_vld = HI; vtx_winner = 4'h6; end
          16'b????_????_??10_0000: begin vtx_winner_vld = HI; vtx_winner = 4'h5; end
          16'b????_????_???1_0000: begin vtx_winner_vld = HI; vtx_winner = 4'h4; end
          16'b????_????_????_1000: begin vtx_winner_vld = HI; vtx_winner = 4'h3; end
          16'b????_????_????_?100: begin vtx_winner_vld = HI; vtx_winner = 4'h2; end
          16'b????_????_????_??10: begin vtx_winner_vld = HI; vtx_winner = 4'h1; end
          16'b????_????_????_???1: begin vtx_winner_vld = HI; vtx_winner = 4'h0; end
          default:                 begin vtx_winner_vld = X;  vtx_winner = 4'bxxxx; end
           endcase
     end



   // - pixel request priority encoder
```

```

   reg          pix_winner_vld;
   reg [3:0]   pix_winner;

   always @(pix_req_q)
     begin
        casez (pix_req_q)
          //16'b0000_0000_0000_0000: begin pix_winner_vld = LO; pix_winner = 4'hf; end
          16'b1000_0000_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'hf; end
          16'b?100_0000_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'he; end
          16'b??10_0000_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'hd; end
          16'b???1_0000_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'hc; end
          16'b????_1000_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'hb; end
          16'b????_?100_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'ha; end
          16'b????_??10_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'h9; end
          16'b????_???1_0000_0000: begin pix_winner_vld = HI; pix_winner = 4'h8; end
          16'b????_????_1000_0000: begin pix_winner_vld = HI; pix_winner = 4'h7; end
          16'b????_????_?100_0000: begin pix_winner_vld = HI; pix_winner = 4'h6; end
          16'b????_????_??10_0000: begin pix_winner_vld = HI; pix_winner = 4'h5; end
          16'b????_????_???1_0000: begin pix_winner_vld = HI; pix_winner = 4'h4; end
          16'b????_????_????_1000: begin pix_winner_vld = HI; pix_winner = 4'h3; end
          16'b????_????_????_?100: begin pix_winner_vld = HI; pix_winner = 4'h2; end
          16'b????_????_????_??10: begin pix_winner_vld = HI; pix_winner = 4'h1; end
          16'b????_????_????_???1: begin pix_winner_vld = HI; pix_winner = 4'h0; end
          //default:                 begin pix_winner_vld = X;  pix_winner = 4'bxxxx; end
```

```
          default:                 begin pix_winner_vld = LO;  pix_winner = 4'bxxxx; end
           endcase
     end


   // - if cfs1 is enabled, alternate btwn rts0 and rts1
   // - if cfs1 is disabled, mask rts1 and always use rts0
   // - what is the algorithm here?   really want to send the thread to the CFS that's available (default
   //   to cfs0 if both are available)
   //   - so getting rid of forced toggle btwn cfs0 and cfs1 - remember to to comment out cfs_turn

   //assign arb_rts0 = arb_rts & (~cfs_turn | ~cfs1_enable);
   //assign arb_rts1 = arb_rts &  cfs_turn & cfs1_enable;

   //wire [0:0] cfs_rtr = cfs_rtr0 | cfs_rtr1;

   wire [0:0] send_to_cfs0 = cfs_rtr0;
   wire [0:0] send_to_cfs1 = ~cfs_rtr0 & cfs_rtr1 & cfs1_enable;

   assign     arb_rts0 = arb_rts & send_to_cfs0;
   assign     arb_rts1 = arb_rts & send_to_cfs1;

   wire [0:0] arb_xfc0 = arb_rts0 & cfs_rtr0;
   wire [0:0] arb_xfc1 = arb_rts1 & cfs_rtr1;

```

```
wire [0:0] arb_xfc = arb_xfc0 | arb_xfc1;


// --------------------
// -- Arb Output Mux --
// --------------------
// - choose between tex state/status and pix state/status depending on overall winner
// - vtx tex has no lod
// - vtx alu has no lod
// - pix tex does have LOD (PIX_CTL_PKT_WIDTH and CTL_PKT_WIDTH have lod)
// - pix alu has no lod

always @(type_winner or vtx_state or pix_state)
 begin
   //arb_state = {STATE_WIDTH{LO}};
   case (type_winner)
     HI: arb_state = vtx_state;    // these are unequal - msb's get 0's by above assignment
     LO: arb_state = pix_state;
     //default: arb_state = {STATE_WIDTH{X}};
   endcase
 end

always @(type_winner or vtx_status or pix_status)
 begin
   //arb_status = {STATUS_WIDTH{LO}};
```

```
   case (type_winner)
     HI: arb_status = vtx_status;
     LO: arb_status = pix_status;
     //default: arb_status = {STATUS_WIDTH{X}};
   endcase
 end


// ---------------------------------------------------------------------------------------------------------
// -- registers --
// ---------------------------------------------------------------------------------------------------------

// register the winner based on ld_winner

always @(posedge clk)
  begin
        if (reset)
          begin
             vtx_winner_q <= 4'h0;
             vtx_winner_vld_q <= LO;
             pix_winner_q <= 4'h0;
             pix_winner_vld_q <= LO;
          end
        else if (ld_winner)
          begin
```

```
             vtx_winner_q <= vtx_winner;
                 vtx_winner_vld_q <= vtx_winner_vld;
             pix_winner_q <= pix_winner;
                 pix_winner_vld_q <= pix_winner_vld;
              end
            else
              begin
                 vtx_winner_q <= vtx_winner_q;
                 vtx_winner_vld_q <= vtx_winner_vld_q;
                 pix_winner_q <= pix_winner_q;
                 pix_winner_vld_q <= pix_winner_vld_q;
              end
  end



// -----------------------------------------------------------------------------------------------------
// -- one-bit state machines --
// -----------------------------------------------------------------------------------------------------

// cfs_turn
// - just toggle when either cfs rts is asserted

wire toggle_turn = arb_rts0 | arb_rts1;

```

```
always @(posedge clk)
  begin
        if ( reset ) cfs_turn <= LO;
        else
          case ( toggle_turn )
            HI:
              cfs_turn <= ~cfs_turn;
            LO:
              cfs_turn <= cfs_turn;
          endcase
  end


// -----------------------------------------------------------------------------------------------------
// -- state machines --
// -----------------------------------------------------------------------------------------------------

// -----------------------------------
// -- Thread Type Arb state machine --
// -----------------------------------

// - arbitrates between the vertex winner and the pixel winner
// - acknowledges the corresponding request
// - waits until the state/status info associated with the current winner is sent to the CFS
//   before returning to pick a new thread type winner
```

```

parameter [1:0] TTA0 = 2'b00;
parameter [1:0] TTA1 = 2'b01;
parameter [1:0] TTA2 = 2'b10;
parameter [1:0] TTA3 = 2'b11;

reg [1:0] tta_current_state;
reg [1:0] tta_next_state;

// state and output registers
always @(posedge clk)
 begin
    if (reset)
     begin
       tta_current_state <= TTA0;
   end
    else
     begin
       tta_current_state <= tta_next_state;
   end
 end

// next state logic
always @(
      state_read_phase or arb_type_strap or
```

```
      vtx_winner_vld_q or pix_winner_vld_q or
      arb_xfc or
      tta_current_state
    )
 begin
    // default assignments
    tta_next_state = TTA0;
    type_winner = LO;
  vtx_winner_ack = LO;
  pix_winner_ack = LO;

    case (tta_current_state)
    TTA0:
   begin
       // - ack is connected to TB State Mem read enable, so have to
       //   wait until the correct phase to ack

       if ( state_read_phase == arb_type_strap )

        if ( vtx_winner_vld_q )               // simply give verts the priority
         begin
        vtx_winner_ack = HI;
            tta_next_state = TTA1;
         end

```

```
        else if ( pix_winner_vld_q )
         begin
        pix_winner_ack = HI;
            tta_next_state = TTA2;
         end

     end

    TTA1:
   begin
       // - wait here until the xfer from the arb to the cfs is complete
       // - type_winner becomes the vector_type output to the cfs, and also
       //   is the final mux select between vertex state and pixel state heading for the CFS

       type_winner = HI;

       if ( arb_xfc )
        tta_next_state = TTA0;
       else
        tta_next_state = TTA1;
     end

    TTA2:
   begin
       type_winner = LO;
```

```

       if ( arb_xfc )
        tta_next_state = TTA0;
       else
        tta_next_state = TTA2;
     end

    TTA3:
   begin
       // -
     tta_next_state = TTA0;
     end

    endcase // case(tta_current_state)
 end
 // - end thread type arb state machine


wire [0:0] arb_thread_type = type_winner; // type is sent to CFS


// ------------------------------
// -- Thread Read state machine --
// ------------------------------

```

```
    // - loads the winner register with new winner from the priority encoder
    // - will initiate a TB state mem read, loading the state into a TB register that is sent back here to
    //   the arbiter for final muxing between vertex and pixel (e.g. tex_winner goes to both VTB and PTB,
    //      but the state read is specific to the TB - there's both a vtx_tex_ld_state and pix_tex_ld_state
    //   - TB State Mem read is enabled when there is any active request

    parameter [1:0] TR0 = 2'b00;
    parameter [1:0] TR1 = 2'b01;
    parameter [1:0] TR2 = 2'b10;
    parameter [1:0] TR3 = 2'b11;

    reg [1:0] tr_current_state;
    reg [1:0] tr_next_state;

    // state and output registers
    always @(posedge clk)
       begin
             if (reset)
               begin
                  tr_current_state <= TR0;
           end
             else
               begin
```

```
                  tr_current_state <= tr_next_state;
           end
       end

    // next state logic
    always @(
            vtx_winner_vld_q or pix_winner_vld_q or type_winner or
            state_read_phase or arb_type_strap or
            //cfs_rtr or
            arb_xfc or
            tr_current_state
        )
      begin
             // default assignments
             tr_next_state = TR0;
             ld_winner = LO;
             arb_rts = LO;

             case (tr_current_state)
             TR0:
           begin
                  ld_winner = HI;

                  if ( vtx_winner_vld_q | pix_winner_vld_q )
                    begin
```

```
                           ld_winner = LO;
                           tr_next_state = TR1;
                    end
               end

             TR1:
           begin
                  // - may need to wait here a cycle depending on phase
                  // - the read data load signal is generated inside the TB - it's just a flopped ack

                  if ( state_read_phase == ~arb_type_strap )
                    begin
                   /*
                       if ( type_winner ) ld_vtb_state = HI;
                       else               ld_ptb_state = HI;
                   */
                           tr_next_state = TR2;
                    end
                  else
                    begin
                       tr_next_state = TR1;
                    end

               end

```

```
             TR2:
           begin
                  // - now the TB state read should be done (reg'd out of TB), and
                  //   this module is muxing btwn vtx and pix, with the result going to the CFS
                  // - assert the rts to the cfs until the xfer is complete

                  arb_rts = HI;

                  //if ( cfs_rtr )
                  if ( arb_xfc )
                    begin
                           ld_winner = HI;
                           tr_next_state = TR0;
                    end
                  else
                    begin
                           tr_next_state = TR2;
                    end

               end

             TR3:
           begin
                  // - this state is not used
              tr_next_state = TR0;
```

```
          end

         endcase
     end
     // - end thread read state machine

/*
   // wire [0:0] tr_rts = (vtx_state_vld | pix_state_vld);

   // arb rts
   // -

   always @(posedge clk)
     begin
         if ( reset ) tr_rts <= LO;
         else
          case ( tr_rts )
           LO:
            tr_rts <= (vtx_state_vld | pix_state_vld);
           HI:
            tr_rts <= ~cfs_rtr;
          endcase
     end

   // ld_winner
```

```
// - just toggle every clock - arbitration is done every other cycle
// - this allows one cycle to pick and register the winner (from the vector of flopped requests),
//   and one cycle for the winner_q to go back and clear the request_q
aarbays @(posedge clk)
  begin
      if (reset)
       ld_winner <= LO;
      else
       ld_winner <= ~ld_winner;
  end
assign ld_winner = tr_current_state[1];

// type select
// - need to save the winner type until cfs acks the arb's rts

always @(posedge clk)
  begin
      if ( reset ) type_winner <= LO;
      else
       case ( type_winner )
        LO:
         type_winner <= winner_thread_type;
        HI:
         type_winner <= ~cfs_rtr;
       endcase
```

```
     end

   // - this is the ack to the thread buffer; it says the winner is valid for its current request
   // - ack the vtx TB when there's a vtx winner and the overall winner is vtx
   // - ack the pix TB when there's a pix winner and the overall winner is pix
   // - can't ack the request when we're stalled by the CFS
   // - the request, and thus the winner, will remain unchanged until the thread buff gets an ack

   //assign vtx_winner_ack =  winner_thread_type & vtx_winner_vld_q & ~tr_stall;
   //assign pix_winner_ack = ~winner_thread_type & pix_winner_vld_q & ~tr_stall;

   // - determine the overall winner between vertex winner and pixel winner
   // - for now verts always have priority

   wire [0:0] winner_thread_type = vtx_winner_vld_q;



*/


endmodule
```

```
`include "header.v"
//---------------------------------------------------------------------------
//
// $Id: //depot/r400/devel/parts_lib/src/gfx/sq/ia/sq_input_arb.v#6 $
//
// $Change: 42997 $
//
// Copyright:   Trade secret of ATI Technologies, Inc.
//              © Copyright 2001-2002, ATI Technologies, Inc., (unpublished)
//
//              All rights reserved.  This notice is intended as a precaution against
//              inadvertent publication and does not imply publication or any waiver
//              of confidentiality.  The year included in the foregoing notice is the
//              year of creation of the work.
//
//---------------------------------------------------------------------------
/////////////////////////////////////////////////////////////////////////////
// sq_input_arb.v
//
// - arbitrate between vertex and pixel input to the GPRs
// - verts have priority over pixels
//
// issues:
// -
//
```

```
/////////////////////////////////////////////////////////////////////////////
`include "sq_defs.v"

module sq_input_arb
(
   vtx_req,       // request from VISM
   vtx_busy,      // busy from VISM - tells arb to keep gpr write mux set to verts
   pix_req,       // request from PISM
   pix_busy,      // busy from PISM - tells arb to keep gpr write mux set to pixels

   gpr_phase,   //

   vtx_gnt,       // grant back to VISM
   pix_gnt,       // grant back to PISM

   vtx_sel,       // this goes to ais_output to select GPRs controls from VISM/PISM

   clk,
   reset
);

   // -- parameters --

   parameter LO = 1'b0;
   parameter HI = 1'b1;
```

```
   parameter X  = 1'bx;


   // ------------------------------------------------------------------
   // -- ios --
   // ------------------------------------------------------------------

   input [0:0] vtx_req;
   input [0:0] vtx_busy;
   input [0:0] pix_req;
   input [0:0] pix_busy;

   input [1:0] gpr_phase;

   output [0:0] vtx_gnt;
   output [0:0] pix_gnt;
   output [0:0] vtx_sel;

   reg [0:0] vtx_gnt;
   reg [0:0] pix_gnt;
   reg [0:0] vtx_sel;

   input clk;
   input reset;

```

```

   // ------------------------------------------------------------------
   // -- internal signals --
   // ------------------------------------------------------------------


   // ------------------------------------------------------------------
   // -- module instatiations --
   // ------------------------------------------------------------------


   // ------------------------------------------------------------------
   // -- combinational logic --
   // ------------------------------------------------------------------


   // ------------------------------------------------------------------
   // -- registers --
   // ------------------------------------------------------------------


   // ------------------------------------------------------------------
   // -- one-bit state machines --
   // ------------------------------------------------------------------

```

```

// ------------------------------------------------------------------
// -- state machines --
// ------------------------------------------------------------------

// input arbiter state machine
// -

parameter IDLE       = 2'b00;
parameter V_XFER = 2'b01;
parameter P_XFER  = 2'b10;
parameter UNUSED           = 2'b11;

reg [1:0] current_state;
reg [1:0] next_state;

reg next_vtx_gnt;
reg next_vtx_sel;
reg next_pix_gnt;

// state and output registers
always @(posedge clk)
  begin
     if (reset)
       begin
```

```
       current_state <= IDLE;
       vtx_gnt <= LO;
       vtx_sel <= LO;
       pix_gnt <= LO;
   end
    else
     begin
       current_state <= next_state;
       vtx_gnt <= next_vtx_gnt;
       vtx_sel <= next_vtx_sel;
       pix_gnt <= next_pix_gnt;
   end
 end


// next state logic
always @(
    vtx_req or pix_req or vtx_busy or pix_busy or gpr_phase or
      current_state
    )
 begin
    // default assignments
    next_state = IDLE;
    next_vtx_gnt = LO;
    next_pix_gnt = LO;
```

```
    next_vtx_sel = LO;

    case (current_state)
     IDLE:
   begin
       // - assert grants based on gpr phase
       //   - gnt is reg'd out, so need to look for phase before the one that lines up
     // - the phase for pix gnt is calculated based on interp latency
       if ( vtx_req & (gpr_phase == `SQ_ID_PHASE) )
        begin
           next_vtx_gnt = HI;
           next_vtx_sel = HI;
       next_state = V_XFER;
      end
       else if ( pix_req & (gpr_phase == `SQ_PV_PHASE) )
        begin
           next_pix_gnt = HI;
       next_state = P_XFER;
      end

     end

     V_XFER:
   begin
       // - hold vtx_sel high while VISM is busy
```

```
       if ( vtx_busy )
        begin
           next_vtx_sel = HI;
           next_state = V_XFER;
        end
       else
        begin
           next_state = IDLE;
        end
     end

     P_XFER:
   begin
       // - first check if there's another pix req (and no vtx req)
       //   - if so, grant it and stay here
       // - otherwise continue to hold vtx_sel low while PISM is busy

       if ( pix_req & ~vtx_req & (gpr_phase == `SQ_PV_PHASE) )
        begin
           next_pix_gnt = HI;
       next_state = P_XFER;
      end
       else if ( pix_busy )
        begin
           next_state = P_XFER;
```

```
              end
            else
              begin
                  next_state = IDLE;
              end
          end

        endcase // case(current_state)

    end // always @ (*)


endmodule
```

```
`include "header.v"
//--------------------------------------------------------------------------
//
// $Id: //depot/r400/devel/parts_lib/src/gfx/sq/is/sq_instruction_store.v#11 $
//
// $Change: 41218 $
//
//
// Notes:       DFF library including plain dff, dff w/reset, dff w/clk enable,
//              and dff w/reset and clk enable.  Reset causes rval to be loaded
//              into the flop
// Copyright:   Trade secret of ATI Technologies, Inc.
//              @ Copyright 2001-2002, ATI Technologies, Inc., (unpublished)
//
//              All rights reserved.  This notice is intended as a precaution against
//              inadvertent publication and does not imply publication or any waiver
//              of confidentiality.  The year included in the foregoing notice is the
//              year of creation of the work.
//
//--------------------------------------------------------------------------


module sq_instruction_store
  (/*AUTOARG*/
// Outputs
```

```
o_rtr, o_is_data,
// Inputs
i_is_phase, i_is_sub_phase, i_rbi_data, i_rbi_addr, i_rts,
i_tex_addr, i_alu0_addr, i_alu1_addr, i_tex_cf_addr, i_alu0_cf_addr,
i_alu1_cf_addr, i_clk, i_reset
);

// memory access phase control
input [1:0] i_is_phase;
input [1:0] i_is_sub_phase;

// RBI
input [31:0] i_rbi_data;
input [14:0] i_rbi_addr;
input        i_rts;
output       o_rtr;

// SQ
input [11:0] i_tex_addr;
input [11:0] i_alu0_addr;
input [11:0] i_alu1_addr;

input [11:0] i_tex_cf_addr;
input [11:0] i_alu0_cf_addr;
input [11:0] i_alu1_cf_addr;
```

```

output [95:0] o_is_data;


// general
input        i_clk;
input        i_reset;

reg [11:0]   addr, d_addr;

reg [14:0]   q_rbi_addr_in, d_rbi_addr_in;

reg [95:0]   wrt_data, d_wrt_data,
             read_data;

reg          we, d_we,
             o_rtr, d_rtr;

wire [95:0]  o_is_data = read_data;

parameter    TEX_PHASE = 2'd1,
             ALU_PHASE = 2'd2,
             CP_PHASE  = 2'd3,
             CF_PHASE  = 2'd0;

```

```
// Access to the is (instruction store) is divided into 4 phases:
// 0: texture instruction read
// 1: alu instruction read
// The alu phase alternates between phases for alu0 and alu1.
// 2: CP write (or read for debug)
// 3: control flow instruction read
// The control flow phase is shared for accesses by alu0, alu1 and tex
// controlled by is_sub_phase.


// address mux
always @(/*AUTOSENSE*/addr or i_alu0_addr or i_alu0_cf_addr
            or i_alu1_addr or i_alu1_cf_addr or i_is_phase
            or i_is_sub_phase or i_rbi_addr or i_rbi_data or i_rts
            or i_tex_addr or i_tex_cf_addr or o_rtr or q_rbi_addr_in
            or wrt_data)
  begin
   // default values
   d_addr     = addr;
   d_rtr      = 1'b0;
   d_we       = 1'b0;
   d_wrt_data = wrt_data;
   case (i_is_phase)

     TEX_PHASE :
```

ATI 2099
LG v. ATI
IPR2015-00326

```
        begin
          d_addr          = i_tex_addr;
          d_rtr           = o_rtr;
          d_wrt_data[63:32] = i_rbi_data;
          d_rbi_addr_in     = q_rbi_addr_in/3;
        end

      ALU_PHASE :
        begin
          d_addr          = i_is_sub_phase[0] ? i_alu1_addr : i_alu0_addr;
          d_rtr           = 1'b0;
          d_we            = o_rtr;
          d_wrt_data[95:64] = i_rbi_data;
          d_rbi_addr_in     = q_rbi_addr_in;
        end

      CP_PHASE :
        begin
          d_addr          = q_rbi_addr_in;
          d_rtr           = i_rts;
          d_wrt_data[31:0] = i_rbi_data;
          d_rbi_addr_in    = i_rbi_addr;
        end

      CF_PHASE :
```

```
        begin
          case (i_is_sub_phase)
            2'b00 :
              d_addr      = i_alu0_cf_addr;
            2'b10 :
              d_addr      = i_alu1_cf_addr;
            default :
              d_addr      = i_tex_cf_addr;
          endcase // case(i_is_sub_phase)

          d_rtr           = o_rtr;
          d_wrt_data[31:0] = i_rbi_data;
//        d_rbi_addr_in    = q_rbi_addr_in/3;
          // FIXME
          // next line needs to replace h5000 with parameter value from register_addr.v
          d_rbi_addr_in    = q_rbi_addr_in - 15'h5000;
        end
    endcase // case(i_is_phase)

  end // always @ (...


wire [95:0] mem_read_data;

`ifdef USE_BEHAVE_MEM
```

```
dum_mem_p2
  #(12, 96, 4096)
u0_dum_mem_p1_4096x96
  (
   // Outputs
   .oQ            (mem_read_data),
   // Inputs
   .iRCLK         (i_clk),
   .iWCLK         (i_clk),
   .iMER          (1'b1),
   .iMEW          (1'b1),
   .iWEN          (we),
   .iWADR         (d_addr),
   .iRADR         (d_addr),
   .iD            (wrt_data));

`else // !ifdef USE_BEHAVE_MEM
// due to the speed of this ram, it had to be split into two 2048x96 rams

wire        vdd = 1'b1;
wire        vss = 1'b0;
wire [95:0] mem0_rd_data;
wire [95:0] mem1_rd_data;

assign      mem_read_data = d_addr[11] ? mem1_rd_data : mem0_rd_data;
```

```


hdsd1_2048x96cm8sw0 u0_is_ram
  (/*VRGIO hdsd1_2048x96cm8sw0 wrt_data mem0_rd_data d_addr null we 1'b1 null*/
// READ/WRITE INTERFACE
      .CLK(i_clk), // Read & Write Clock
      .WE(we), // Write enable
      .OE(1'b1), // Output enable
//    .ME(vdd), // Read enable
      .ME(~d_addr[11]), // Read enable
      .ADR0(d_addr[0]), .ADR1(d_addr[1]), .ADR2(d_addr[2]), .ADR3(d_addr[3]),  // Address
      .ADR4(d_addr[4]), .ADR5(d_addr[5]), .ADR6(d_addr[6]), .ADR7(d_addr[7]),  // Address
      .ADR8(d_addr[8]), .ADR9(d_addr[9]), .ADR10(d_addr[10]),  // Address
      .D0(wrt_data[0]), .D1(wrt_data[1]), .D2(wrt_data[2]), .D3(wrt_data[3]),  // Write Data
      .D4(wrt_data[4]), .D5(wrt_data[5]), .D6(wrt_data[6]), .D7(wrt_data[7]),  // Write Data
      .D8(wrt_data[8]), .D9(wrt_data[9]), .D10(wrt_data[10]), .D11(wrt_data[11]),  // Write Data
      .D12(wrt_data[12]), .D13(wrt_data[13]), .D14(wrt_data[14]), .D15(wrt_data[15]),  // Write
Data
      .D16(wrt_data[16]), .D17(wrt_data[17]), .D18(wrt_data[18]), .D19(wrt_data[19]),  // Write
Data
      .D20(wrt_data[20]), .D21(wrt_data[21]), .D22(wrt_data[22]), .D23(wrt_data[23]),  // Write
Data
      .D24(wrt_data[24]), .D25(wrt_data[25]), .D26(wrt_data[26]), .D27(wrt_data[27]),  // Write
Data
      .D28(wrt_data[28]), .D29(wrt_data[29]), .D30(wrt_data[30]), .D31(wrt_data[31]),  // Write
Data
      .D32(wrt_data[32]), .D33(wrt_data[33]), .D34(wrt_data[34]), .D35(wrt_data[35]),  // Write
Data
```

```
    .D36(wrt_data[36]), .D37(wrt_data[37]), .D38(wrt_data[38]), .D39(wrt_data[39]),  // Write Data
    .D40(wrt_data[40]), .D41(wrt_data[41]), .D42(wrt_data[42]), .D43(wrt_data[43]),  // Write Data
    .D44(wrt_data[44]), .D45(wrt_data[45]), .D46(wrt_data[46]), .D47(wrt_data[47]),  // Write Data
    .D48(wrt_data[48]), .D49(wrt_data[49]), .D50(wrt_data[50]), .D51(wrt_data[51]),  // Write Data
    .D52(wrt_data[52]), .D53(wrt_data[53]), .D54(wrt_data[54]), .D55(wrt_data[55]),  // Write Data
    .D56(wrt_data[56]), .D57(wrt_data[57]), .D58(wrt_data[58]), .D59(wrt_data[59]),  // Write Data
    .D60(wrt_data[60]), .D61(wrt_data[61]), .D62(wrt_data[62]), .D63(wrt_data[63]),  // Write Data
    .D64(wrt_data[64]), .D65(wrt_data[65]), .D66(wrt_data[66]), .D67(wrt_data[67]),  // Write Data
    .D68(wrt_data[68]), .D69(wrt_data[69]), .D70(wrt_data[70]), .D71(wrt_data[71]),  // Write Data
    .D72(wrt_data[72]), .D73(wrt_data[73]), .D74(wrt_data[74]), .D75(wrt_data[75]),  // Write Data
    .D76(wrt_data[76]), .D77(wrt_data[77]), .D78(wrt_data[78]), .D79(wrt_data[79]),  // Write Data
    .D80(wrt_data[80]), .D81(wrt_data[81]), .D82(wrt_data[82]), .D83(wrt_data[83]),  // Write Data
    .D84(wrt_data[84]), .D85(wrt_data[85]), .D86(wrt_data[86]), .D87(wrt_data[87]),  // Write Data
    .D88(wrt_data[88]), .D89(wrt_data[89]), .D90(wrt_data[90]), .D91(wrt_data[91]),  // Write Data
    .D92(wrt_data[92]), .D93(wrt_data[93]), .D94(wrt_data[94]), .D95(wrt_data[95]),  // Write Data
    .Q0(mem0_rd_data[0]),    .Q1(mem0_rd_data[1]),    .Q2(mem0_rd_data[2]),    .Q3(mem0_rd_data[3]),  // Read Data
```

```
    .Q4(mem0_rd_data[4]),    .Q5(mem0_rd_data[5]),    .Q6(mem0_rd_data[6]),    .Q7(mem0_rd_data[7]),  // Read Data
    .Q8(mem0_rd_data[8]),    .Q9(mem0_rd_data[9]),    .Q10(mem0_rd_data[10]),    .Q11(mem0_rd_data[11]),  // Read Data
    .Q12(mem0_rd_data[12]),   .Q13(mem0_rd_data[13]),   .Q14(mem0_rd_data[14]),   .Q15(mem0_rd_data[15]),  // Read Data
    .Q16(mem0_rd_data[16]),   .Q17(mem0_rd_data[17]),   .Q18(mem0_rd_data[18]),   .Q19(mem0_rd_data[19]),  // Read Data
    .Q20(mem0_rd_data[20]),   .Q21(mem0_rd_data[21]),   .Q22(mem0_rd_data[22]),   .Q23(mem0_rd_data[23]),  // Read Data
    .Q24(mem0_rd_data[24]),   .Q25(mem0_rd_data[25]),   .Q26(mem0_rd_data[26]),   .Q27(mem0_rd_data[27]),  // Read Data
    .Q28(mem0_rd_data[28]),   .Q29(mem0_rd_data[29]),   .Q30(mem0_rd_data[30]),   .Q31(mem0_rd_data[31]),  // Read Data
    .Q32(mem0_rd_data[32]),   .Q33(mem0_rd_data[33]),   .Q34(mem0_rd_data[34]),   .Q35(mem0_rd_data[35]),  // Read Data
    .Q36(mem0_rd_data[36]),   .Q37(mem0_rd_data[37]),   .Q38(mem0_rd_data[38]),   .Q39(mem0_rd_data[39]),  // Read Data
    .Q40(mem0_rd_data[40]),   .Q41(mem0_rd_data[41]),   .Q42(mem0_rd_data[42]),   .Q43(mem0_rd_data[43]),  // Read Data
    .Q44(mem0_rd_data[44]),   .Q45(mem0_rd_data[45]),   .Q46(mem0_rd_data[46]),   .Q47(mem0_rd_data[47]),  // Read Data
    .Q48(mem0_rd_data[48]),   .Q49(mem0_rd_data[49]),   .Q50(mem0_rd_data[50]),   .Q51(mem0_rd_data[51]),  // Read Data
    .Q52(mem0_rd_data[52]),   .Q53(mem0_rd_data[53]),   .Q54(mem0_rd_data[54]),   .Q55(mem0_rd_data[55]),  // Read Data
    .Q56(mem0_rd_data[56]),   .Q57(mem0_rd_data[57]),   .Q58(mem0_rd_data[58]),   .Q59(mem0_rd_data[59]),  // Read Data
    .Q60(mem0_rd_data[60]),   .Q61(mem0_rd_data[61]),   .Q62(mem0_rd_data[62]),   .Q63(mem0_rd_data[63]),  // Read Data
    .Q64(mem0_rd_data[64]),   .Q65(mem0_rd_data[65]),   .Q66(mem0_rd_data[66]),   .Q67(mem0_rd_data[67]),  // Read Data
```

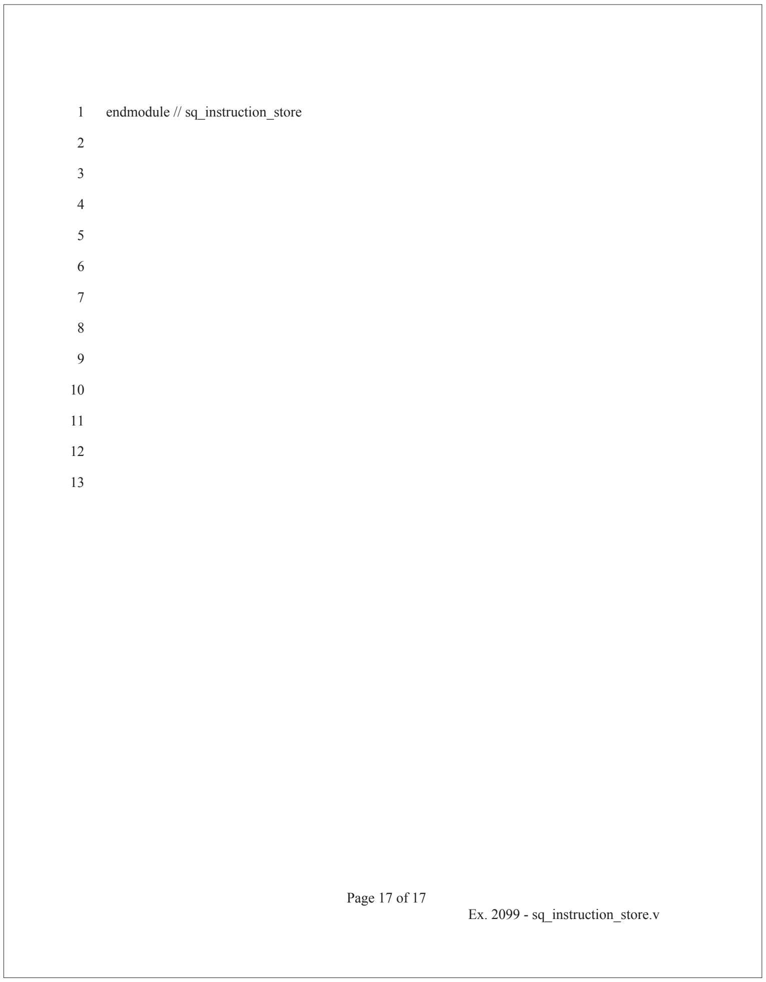
```
    .Q68(mem0_rd_data[68]),   .Q69(mem0_rd_data[69]),   .Q70(mem0_rd_data[70]),   .Q71(mem0_rd_data[71]),  // Read Data
    .Q72(mem0_rd_data[72]),   .Q73(mem0_rd_data[73]),   .Q74(mem0_rd_data[74]),   .Q75(mem0_rd_data[75]),  // Read Data
    .Q76(mem0_rd_data[76]),   .Q77(mem0_rd_data[77]),   .Q78(mem0_rd_data[78]),   .Q79(mem0_rd_data[79]),  // Read Data
    .Q80(mem0_rd_data[80]),   .Q81(mem0_rd_data[81]),   .Q82(mem0_rd_data[82]),   .Q83(mem0_rd_data[83]),  // Read Data
    .Q84(mem0_rd_data[84]),   .Q85(mem0_rd_data[85]),   .Q86(mem0_rd_data[86]),   .Q87(mem0_rd_data[87]),  // Read Data
    .Q88(mem0_rd_data[88]),   .Q89(mem0_rd_data[89]),   .Q90(mem0_rd_data[90]),   .Q91(mem0_rd_data[91]),  // Read Data
    .Q92(mem0_rd_data[92]),   .Q93(mem0_rd_data[93]),   .Q94(mem0_rd_data[94]),   .Q95(mem0_rd_data[95]),  // Read Data
    // READ/WRITE TEST SIGNALS
    .BISTE(vss),
    .TWE(vss),
    .TOE(vss),
    .TME(vss),
    .TADR0(d_addr[0]),  .TADR1(d_addr[1]),  .TADR2(d_addr[2]),  .TADR3(d_addr[3]),    // Write Test Address
    .TADR4(d_addr[4]),  .TADR5(d_addr[5]),  .TADR6(d_addr[6]),  .TADR7(d_addr[7]),    // Write Test Address
    .TADR8(d_addr[8]), .TADR9(d_addr[9]), .TADR10(d_addr[10]),  // Write Test Address
    .RM0(vss), .RM1(vss), .RM2(vss), .RM3(vss),  // Read Margin
    .AWT(vss)
    );

hdsd1_2048x96cm8sw0 u1_is_ram
```

```
 (/*VRGIO hdsd1_2048x96cm8sw0 wrt_data mem1_rd_data d_addr null we 1'b1 null*/
// READ/WRITE INTERFACE
    .CLK(i_clk), // Read & Write Clock
    .WE(we), // Write enable
    .OE(1'b1), // Output enable
//    .ME(vdd), // Read enable
    .ME(d_addr[11]), // Read enable
    .ADR0(d_addr[0]), .ADR1(d_addr[1]), .ADR2(d_addr[2]), .ADR3(d_addr[3]),  // Address
    .ADR4(d_addr[4]), .ADR5(d_addr[5]), .ADR6(d_addr[6]), .ADR7(d_addr[7]),  // Address
    .ADR8(d_addr[8]), .ADR9(d_addr[9]), .ADR10(d_addr[10]),  // Address
    .D0(wrt_data[0]), .D1(wrt_data[1]), .D2(wrt_data[2]), .D3(wrt_data[3]),  // Write Data
    .D4(wrt_data[4]), .D5(wrt_data[5]), .D6(wrt_data[6]), .D7(wrt_data[7]),  // Write Data
    .D8(wrt_data[8]), .D9(wrt_data[9]), .D10(wrt_data[10]), .D11(wrt_data[11]),  // Write Data
    .D12(wrt_data[12]), .D13(wrt_data[13]), .D14(wrt_data[14]), .D15(wrt_data[15]),  // Write Data
    .D16(wrt_data[16]), .D17(wrt_data[17]), .D18(wrt_data[18]), .D19(wrt_data[19]),  // Write Data
    .D20(wrt_data[20]), .D21(wrt_data[21]), .D22(wrt_data[22]), .D23(wrt_data[23]),  // Write Data
    .D24(wrt_data[24]), .D25(wrt_data[25]), .D26(wrt_data[26]), .D27(wrt_data[27]),  // Write Data
    .D28(wrt_data[28]), .D29(wrt_data[29]), .D30(wrt_data[30]), .D31(wrt_data[31]),  // Write Data
    .D32(wrt_data[32]), .D33(wrt_data[33]), .D34(wrt_data[34]), .D35(wrt_data[35]),  // Write Data
    .D36(wrt_data[36]), .D37(wrt_data[37]), .D38(wrt_data[38]), .D39(wrt_data[39]),  // Write Data
    .D40(wrt_data[40]), .D41(wrt_data[41]), .D42(wrt_data[42]), .D43(wrt_data[43]),  // Write Data
    .D44(wrt_data[44]), .D45(wrt_data[45]), .D46(wrt_data[46]), .D47(wrt_data[47]),  // Write
Data
    .D48(wrt_data[48]), .D49(wrt_data[49]), .D50(wrt_data[50]), .D51(wrt_data[51]),  // Write
Data
    .D52(wrt_data[52]), .D53(wrt_data[53]), .D54(wrt_data[54]), .D55(wrt_data[55]),  // Write
Data
    .D56(wrt_data[56]), .D57(wrt_data[57]), .D58(wrt_data[58]), .D59(wrt_data[59]),  // Write
Data
    .D60(wrt_data[60]), .D61(wrt_data[61]), .D62(wrt_data[62]), .D63(wrt_data[63]),  // Write
Data
    .D64(wrt_data[64]), .D65(wrt_data[65]), .D66(wrt_data[66]), .D67(wrt_data[67]),  // Write
Data
    .D68(wrt_data[68]), .D69(wrt_data[69]), .D70(wrt_data[70]), .D71(wrt_data[71]),  // Write
Data
    .D72(wrt_data[72]), .D73(wrt_data[73]), .D74(wrt_data[74]), .D75(wrt_data[75]),  // Write
Data
    .D76(wrt_data[76]), .D77(wrt_data[77]), .D78(wrt_data[78]), .D79(wrt_data[79]),  // Write
Data
    .D80(wrt_data[80]), .D81(wrt_data[81]), .D82(wrt_data[82]), .D83(wrt_data[83]),  // Write
Data
    .D84(wrt_data[84]), .D85(wrt_data[85]), .D86(wrt_data[86]), .D87(wrt_data[87]),  // Write
Data
    .D88(wrt_data[88]), .D89(wrt_data[89]), .D90(wrt_data[90]), .D91(wrt_data[91]),  // Write
Data
    .D92(wrt_data[92]), .D93(wrt_data[93]), .D94(wrt_data[94]), .D95(wrt_data[95]),  // Write
Data
    .Q0(mem1_rd_data[0]),        .Q1(mem1_rd_data[1]),        .Q2(mem1_rd_data[2]),
.Q3(mem1_rd_data[3]),  // Read Data
    .Q4(mem1_rd_data[4]),        .Q5(mem1_rd_data[5]),        .Q6(mem1_rd_data[6]),
.Q7(mem1_rd_data[7]),  // Read Data
    .Q8(mem1_rd_data[8]),       .Q9(mem1_rd_data[9]),       .Q10(mem1_rd_data[10]),
.Q11(mem1_rd_data[11]),  // Read Data
```

```
    .Q12(mem1_rd_data[12]),     .Q13(mem1_rd_data[13]),     .Q14(mem1_rd_data[14]),
.Q15(mem1_rd_data[15]),  // Read Data
    .Q16(mem1_rd_data[16]),     .Q17(mem1_rd_data[17]),     .Q18(mem1_rd_data[18]),
.Q19(mem1_rd_data[19]),  // Read Data
    .Q20(mem1_rd_data[20]),     .Q21(mem1_rd_data[21]),     .Q22(mem1_rd_data[22]),
.Q23(mem1_rd_data[23]),  // Read Data
    .Q24(mem1_rd_data[24]),     .Q25(mem1_rd_data[25]),     .Q26(mem1_rd_data[26]),
.Q27(mem1_rd_data[27]),  // Read Data
    .Q28(mem1_rd_data[28]),     .Q29(mem1_rd_data[29]),     .Q30(mem1_rd_data[30]),
.Q31(mem1_rd_data[31]),  // Read Data
    .Q32(mem1_rd_data[32]),     .Q33(mem1_rd_data[33]),     .Q34(mem1_rd_data[34]),
.Q35(mem1_rd_data[35]),  // Read Data
    .Q36(mem1_rd_data[36]),     .Q37(mem1_rd_data[37]),     .Q38(mem1_rd_data[38]),
.Q39(mem1_rd_data[39]),  // Read Data
    .Q40(mem1_rd_data[40]),     .Q41(mem1_rd_data[41]),     .Q42(mem1_rd_data[42]),
.Q43(mem1_rd_data[43]),  // Read Data
    .Q44(mem1_rd_data[44]),     .Q45(mem1_rd_data[45]),     .Q46(mem1_rd_data[46]),
.Q47(mem1_rd_data[47]),  // Read Data
    .Q48(mem1_rd_data[48]),     .Q49(mem1_rd_data[49]),     .Q50(mem1_rd_data[50]),
.Q51(mem1_rd_data[51]),  // Read Data
    .Q52(mem1_rd_data[52]),     .Q53(mem1_rd_data[53]),     .Q54(mem1_rd_data[54]),
.Q55(mem1_rd_data[55]),  // Read Data
    .Q56(mem1_rd_data[56]),     .Q57(mem1_rd_data[57]),     .Q58(mem1_rd_data[58]),
.Q59(mem1_rd_data[59]),  // Read Data
    .Q60(mem1_rd_data[60]),     .Q61(mem1_rd_data[61]),     .Q62(mem1_rd_data[62]),
.Q63(mem1_rd_data[63]),  // Read Data
    .Q64(mem1_rd_data[64]),     .Q65(mem1_rd_data[65]),     .Q66(mem1_rd_data[66]),
.Q67(mem1_rd_data[67]),  // Read Data
    .Q68(mem1_rd_data[68]),     .Q69(mem1_rd_data[69]),     .Q70(mem1_rd_data[70]),
.Q71(mem1_rd_data[71]),  // Read Data
    .Q72(mem1_rd_data[72]),     .Q73(mem1_rd_data[73]),     .Q74(mem1_rd_data[74]),
.Q75(mem1_rd_data[75]),  // Read Data
```

```
    .Q76(mem1_rd_data[76]),     .Q77(mem1_rd_data[77]),     .Q78(mem1_rd_data[78]),
.Q79(mem1_rd_data[79]),  // Read Data
    .Q80(mem1_rd_data[80]),     .Q81(mem1_rd_data[81]),     .Q82(mem1_rd_data[82]),
.Q83(mem1_rd_data[83]),  // Read Data
    .Q84(mem1_rd_data[84]),     .Q85(mem1_rd_data[85]),     .Q86(mem1_rd_data[86]),
.Q87(mem1_rd_data[87]),  // Read Data
    .Q88(mem1_rd_data[88]),     .Q89(mem1_rd_data[89]),     .Q90(mem1_rd_data[90]),
.Q91(mem1_rd_data[91]),  // Read Data
    .Q92(mem1_rd_data[92]),     .Q93(mem1_rd_data[93]),     .Q94(mem1_rd_data[94]),
.Q95(mem1_rd_data[95]),  // Read Data
    // READ/WRITE TEST SIGNALS
    .BISTE(vss),
    .TWE(vss),
    .TOE(vss),
    .TME(vss),
    .TADR0(d_addr[0]),  .TADR1(d_addr[1]),  .TADR2(d_addr[2]),  .TADR3(d_addr[3]),   //
Write Test Address
    .TADR4(d_addr[4]),  .TADR5(d_addr[5]),  .TADR6(d_addr[6]),  .TADR7(d_addr[7]),   //
Write Test Address
    .TADR8(d_addr[8]), .TADR9(d_addr[9]), .TADR10(d_addr[10]),  // Write Test Address
    .RM0(vss), .RM1(vss), .RM2(vss), .RM3(vss),  // Read Margin
    .AWT(vss)
    );
`endif

// register instantiation
always @(posedge i_clk)
  begin
```

```
    if (i_reset)
      begin
         we          <= 1'b0;
//       addr        <= 12'd0;
         read_data   <= 96'd0;
         o_rtr       <= 1'b0;
         wrt_data    <= 96'd0;
         q_rbi_addr_in <= 12'd0;
      end
    else
      begin
         we          <= d_we;
//       addr        <= d_addr;
         read_data   <= mem_read_data;
         o_rtr       <= d_rtr;
         wrt_data    <= d_wrt_data;
         q_rbi_addr_in <= d_rbi_addr_in;
      end
  end


// Local Variables:
// verilog-library-directories:("." "../../../common/")
// End:

```



```
endmodule // sq_instruction_store
```

```
`define SQ_VTX_CTL_PKT_WIDTH        88      // number of bits in the vertex control packet
                                // first:1, instr_ptr:12, resource:1, valid_bits:64, gpr_base:7, context_id:3
`define SQ_PIX_CTL_PKT_WIDTH        184     // number of bits in the pixel control packet
                                // first:1, base_ptr:12, resource:1, lod_correct:96, valid_bits:64, gpr_base:7, context_id:3
`define SQ_CTL_PKT_WIDTH        184     // number of bits in the pixel control packet

// State Mem
`define SQ_VTX_STATE_WIDTH  99      // ism + cfs state
`define SQ_PIX_STATE_WIDTH  236     //
`define SQ_STATE_WIDTH          236     // max of the above

// ISM State Mem
`define SQ_VTX_ISM_STATE_WIDTH      74      // number of bits in the vertex thread ISM state
                                // valid_bits:64, gpr_base:7, context_id:3 (note that valid_bits will move to alu_state)
`define SQ_PIX_ISM_STATE_WIDTH      218     // number of bits in the pixel thread state
                                // lod_correct:144, valid_bits:64, gpr_base:7, context_id:3
`define SQ_ISM_STATE_WIDTH  218     // max of the above

// CFS State Mem
`define SQ_VTX_CFS_STATE_WIDTH      25      // number of bits in the vertex thread CFS state
                                // instr_ptr:13, exec_cnt:4, param_ptr:7, export_id:1
`define SQ_PIX_CFS_STATE_WIDTH      18      // number of bits in the pixel thread state
```

```
                                // instr_ptr:13, exec_cnt:4, export_id:1
`define SQ_CFS_STATE_WIDTH  25      // max of the above

//`define SQ_TEX_CFS_STATE_WIDTH    18      // number of bits in the tex thread CFS state
//`define SQ_ALU_CFS_STATE_WIDTH    25      // number of bits in the ALU thread state

// Status Regs
`define SQ_VTX_STATUS_WIDTH         22      // number of bits in the vertex thread status
                                // thread_id:4, status:14
`define SQ_PIX_STATUS_WIDTH 22      // number of bits in the pixel thread status
                                // thread_id:4, status:14
`define SQ_STATUS_WIDTH             22      // max of the above

// instruction sequencer type - compared to inst store phase and alu phase
`define SQ_TEX_STRAP    3'b010      // - this is what {is_phase, alu_phase} should match (except that
`define SQ_ALU0_STRAP  3'b100   //   alu_phase is also tied low on the tex instance)
`define SQ_ALU1_STRAP  3'b101

// instr store phase
`define SQ_IS_CFS_PHASE 2'b00
`define SQ_IS_TEX_PHASE 2'b01
`define SQ_IS_ALU_PHASE 2'b10
`define SQ_IS_CP_PHASE 2'b11

// instr store CFS sub phase
```

```
`define SQ_IS_ALU0_SUBPHASE 2'b00
`define SQ_IS_TEX0_SUBPHASE 2'b01
`define SQ_IS_ALU1_SUBPHASE 2'b10
`define SQ_IS_TEX1_SUBPHASE 2'b11     // for tex, only care if subphase lsb == 1

// CFS phase
`define SQ_CFS_ALU0_PHASE 2'b00
`define SQ_CFS_TEX0_PHASE 2'b01
`define SQ_CFS_ALU1_PHASE 2'b10
`define SQ_CFS_TEX1_PHASE 2'b11                 // for tex, only care if phase lsb == 1

// gpr write phase
`define SQ_ID_PHASE 2'b00           // Input Data gpr write phase
`define SQ_FD_PHASE 2'b01           // Fetch Data gpr write phase
`define SQ_PV_PHASE 2'b10           // Vector Result (PV) gpr write phase
`define SQ_PS_PHASE 2'b11           // Scalar Result (PS) gpr write phase

// gpr read phase
`define SQ_SRCB_PHASE 2'b00     // source B gpr read phase
`define SQ_SRCC_PHASE 2'b01     // source C gpr read phase
`define SQ_FA_PHASE   2'b10         // fetch address gpr read phase
`define SQ_SRCA_PHASE 2'b11     // source A gpr read phase


`define SQ_SC_DATA_WIDTH        54      // width of the scan converter input data bus
```

```
`define SQ_PB_WIDTH                 53      // width of the pointer buffer

// vector type: 1 = VTX, 0 = PIX
`define SQ_VTX  1'b1
`define SQ_PIX  1'b0
```

```
`include "header.v"
//---------------------------------------------------------------------------
//
// $Id: //depot/r400/devel/parts_lib/src/gfx/sq/ss/sq_thread_buff.v#27 $
//
// $Change: 44010 $
//
// Copyright:   Trade secret of ATI Technologies, Inc.
//              © Copyright 2001-2002, ATI Technologies, Inc., (unpublished)
//
//              All rights reserved.  This notice is intended as a precaution against
//              inadvertent publication and does not imply publication or any waiver
//              of confidentiality.  The year included in the foregoing notice is the
//              year of creation of the work.
//
//---------------------------------------------------------------------------
/////////////////////////////////////////////////////////////////////////////////
// sq_thread_buff.v
//
// - this is the clause-less thread buffer (a.k.a. reservation station)
// - contains Thread State Memory and Thread Status Registers
// - each line (thread) has both a texture and an alu request output (only one can
//   be active at a time)
// - tex requests go to the tex arb, and a tex_winner is returned
// - alu requests go to the alu arb, and a alu_winner is returned
```

```
// - the winning thread_id is used to read the State Mem and the Status Registers so that
//   the selected thread info can be sent to the associated Control Flow Sequencer (CFS) (via the arb)
//
// - the sq top level will contain two instances of the thread buffer (one for verts, on for pixels)
// - the tex requests from both TBs go to the tex thread arbiter, and the alu requests from both TBs
//   go to the alu thread arbiter
// - so there are 4 total winners from two arbiters: vtx_tex, pix_tex, vtx_alu, and pix_alu
// - each arbiter then does the final muxing between vtx and pix
//
// issues:
// -
//
/////////////////////////////////////////////////////////////////////////////////

module sq_thread_buff
(
  thread_type_strap,   // a strap that tells this module if it's a vertex or pixel thread buffer
  state_read_phase,    // share read access between tex and alu arbs
  cfs_phase,           // share write (update) access between the tex and alu CFSs

  // inputs from local registers
  num_reg_set,                  // connected to SQ_PROGRAM_CNTL.VS_NUM_REG (6 bits) (or PS_NUM_REG)

```

```
  // control packet input (from ISM) - initial values for state and status
  ism_rts,             // control packet rts
  ism_lod_correct,     // state (pix only)
  ism_instr_ptr,       // state
  ism_valid_bits,      // state
  ism_gpr_base,        // state
  ism_context_id,      // state
  ism_resource,        // status: resource bit :  tex=1, alu=0
  ism_first_thread,    // status: first thread of a new state
  tb_rtr,              // rtr when not full AND not doing a CFS update

  // tex control flow seq update of state and status
  tcfs_update,         // load updated status info from CFS
  tcfs_thread_type,    // the vector type: pixel=0, vertex=1
  tcfs_state,          // state returned from cfs
  tcfs_status,         // status returned from cfs

  // alu control flow seq update of state and status
  acfs0_update,                 // load updated status info from CFS
  acfs0_thread_type,   // the vector type: pixel=0, vertex=1
  acfs0_state,         // state returned from cfs
  acfs0_status,        // status returned from cfs

  acfs1_update,                 // load updated status info from CFS
  acfs1_thread_type,   // the vector type: pixel=0, vertex=1
```

```
  acfs1_state,         // state returned from cfs
  acfs1_status,        // status returned from cfs


  // tex thread arbiter interface
  tex_req_q,           // tex request from every thread in the buffer
  tex_winner_q,        // tex winner from arbiter
  tex_winner_ack,      // tex winner valid (request acknowledge) from tex arbiter
  tex_state_q,         // winning state read from State Mem back to tex arbiter
  tex_status_q,        // winning status read from Status Regs back to tex arbiter

  TP_SQ_data_rdy,      // data ready (done) indicator from TPC
  TP_SQ_type,                   // the vector type: pixel=0, vertex=1
  TP_SQ_thread_id,     //

  // alu thread arbiter interface
  alu_req_q,           // alu req from every thread
  alu_winner_q,        // alu winner from arbiter
  alu_winner_ack,      // alu winner valid from alu arbiter
  alu_state_q,         // winning state read from State Mem
  alu_status_q,        // winning status read from Status Regs

  ais0_done,           // done indicator from AIS0
  ais0_thread_type,    // the vector type: pixel=0, vertex=1
  ais0_thread_id,      //
```

```

    ais1_done,              // done indicator from AIS1
    ais1_thread_type,     // the vector type: pixel=0, vertex=1
    ais1_thread_id,        //

    // SX export buffer availability
    u0_SX_SQ_exp_count_rdy,// avaliability info is valid
    u0_SX_SQ_exp_pos_avail,// position available from SX
    u0_SX_SQ_exp_buf_avail,// buffer available from SX (0 to 127 2kbit buffers(), 2kbit = 32bits for 64 pixels)

    u1_SX_SQ_exp_count_rdy,
    u1_SX_SQ_exp_pos_avail,
    u1_SX_SQ_exp_buf_avail,

    // outputs from exit SM to constant stores and gpr alloc
    // - this needs to be checked...
    state_change,           // a pulse high indicates that the state exiting the SS has changed
    old_state,              // the state that has finished (because a new state has emerged)
    dealloc_req,            // request to deallocate GPRs
    dealloc_space,        // number of locations to dealloc (from local gfx reg)
    dealloc_ack,            // the dealloc request has been acknowleged

    pop_thread,             // syncs vtx shader with pix input (latter must wait for former)

    param_cache_wptr_q,         // input to status regs
```

```

    busy,           // TB is busy when there are any threads in the buffer
    clk,
    reset
);



    // -- parameters --

    parameter STATE_WIDTH = 8;
    parameter ISM_STATE_WIDTH = 8;
    parameter CFS_STATE_WIDTH = 8;
    //parameter ALU_STATE_WIDTH = 8;

    parameter STATUS_WIDTH = 8;

    parameter TB_DEPTH = 16;       // number of locations
    parameter TB_ADDR_WIDTH = 4;    // log2 (number of locations rounded to nearest power of 2)

    parameter TID_WIDTH = 6;   // number ob bits in the thread ID

    parameter LO = 1'b0;
    parameter HI = 1'b1;
    parameter X  = 1'bx;

```

```

    // ---------------------------------------------------------------------------------------
    // -- ios --
    // ---------------------------------------------------------------------------------------

    input [0:0] thread_type_strap;
    input [0:0] state_read_phase;
    input [1:0] cfs_phase;

    input [8*6-1:0] num_reg_set;

    //
    input [00:0] ism_rts;
    input [143:0] ism_lod_correct;
    input [11:0] ism_instr_ptr;
    input [63:0] ism_valid_bits;
    input [06:0] ism_gpr_base;
    input [02:0] ism_context_id;
    input [00:0] ism_resource;
    input [00:0] ism_first_thread;

    output [0:0] tb_rtr;

    //
    input [0:0]                         tcfs_update;
```

```
    input [0:0]                         tcfs_thread_type;
    input [CFS_STATE_WIDTH-1:0]  tcfs_state;
    input [STATUS_WIDTH-1:0]        tcfs_status;

    //
    input [0:0]                         acfs0_update;
    input [0:0]                         acfs0_thread_type;
    input [CFS_STATE_WIDTH-1:0]  acfs0_state;
    input [STATUS_WIDTH-1:0]        acfs0_status;

    //
    input [0:0]                         acfs1_update;
    input [0:0]                         acfs1_thread_type;
    input [CFS_STATE_WIDTH-1:0]  acfs1_state;
    input [STATUS_WIDTH-1:0]        acfs1_status;


    output [TB_DEPTH-1:0]     tex_req_q;
    output [STATE_WIDTH-1:0]        tex_state_q;
    output [STATUS_WIDTH-1:0]      tex_status_q;

    input  [TB_ADDR_WIDTH-1:0]    tex_winner_q;
    input  [0:0]              tex_winner_ack;

    input [0:0]                         TP_SQ_data_rdy;
```

```
input [0:0]              TP_SQ_type;
input [TID_WIDTH-1:0]    TP_SQ_thread_id;

output [TB_DEPTH-1:0]    alu_req_q;
output [STATE_WIDTH-1:0]      alu_state_q;
output [STATUS_WIDTH-1:0]     alu_status_q;

input  [TB_ADDR_WIDTH-1:0]    alu_winner_q;
input  [0:0]             alu_winner_ack;

input [0:0]                   ais0_done;
input [0:0]                   ais0_thread_type;
input [TID_WIDTH-1:0]    ais0_thread_id;

input [0:0]                   ais1_done;
input [0:0]                   ais1_thread_type;
input [TID_WIDTH-1:0]    ais1_thread_id;

input [0:0]    u0_SX_SQ_exp_count_rdy;
input [0:0]    u0_SX_SQ_exp_pos_avail;
//tri1 [0:0]    u0_SX_SQ_exp_pos_avail;
input [6:0]    u0_SX_SQ_exp_buf_avail;

input [0:0]    u1_SX_SQ_exp_count_rdy;
input [0:0]    u1_SX_SQ_exp_pos_avail;
```

```
//tri1 [0:0]    u1_SX_SQ_exp_pos_avail;
input [6:0]    u1_SX_SQ_exp_buf_avail;

output                   state_change;
output [2:0]             old_state;
output                   dealloc_req;
output [5:0]             dealloc_space;
input           dealloc_ack;

output                   pop_thread;
input [6:0]              param_cache_wptr_q;

output                   busy;

input                    clk;
input                    reset;


// ---------------------------------------------------------------------------------------
// -- internal signals --
// ---------------------------------------------------------------------------------------

reg [STATE_WIDTH-1:0]  tex_state_q;
reg [STATUS_WIDTH-1:0]          tex_status_q;
reg [STATE_WIDTH-1:0]  alu_state_q;
```

```
reg [STATUS_WIDTH-1:0]          alu_status_q;

reg [0:0] tp_done_q;
reg [0:0] tp_thread_type_q;
reg [TID_WIDTH-1:0] tp_thread_id_q;

reg [TID_WIDTH-1:0] state_head_ptr_q;
reg [TID_WIDTH-1:0] state_tail_ptr_q;
reg [TID_WIDTH-1:0] status_tail_ptr_q;

reg [TID_WIDTH:0] full_cnt_q;

reg [0:0] tex_winner_ack_q;
reg [0:0] alu_winner_ack_q;
reg [0:0] alu_winner_ack_q1;

reg [0:0] sx_pos_avail;
reg [6:0] sx_buf_avail;

wire [STATUS_WIDTH-1:0] status_data_0;
wire [STATUS_WIDTH-1:0] status_data_1;
wire [STATUS_WIDTH-1:0] status_data_2;
wire [STATUS_WIDTH-1:0] status_data_3;
wire [STATUS_WIDTH-1:0] status_data_4;
wire [STATUS_WIDTH-1:0] status_data_5;
```

```
wire [STATUS_WIDTH-1:0] status_data_6;
wire [STATUS_WIDTH-1:0] status_data_7;
wire [STATUS_WIDTH-1:0] status_data_8;
wire [STATUS_WIDTH-1:0] status_data_9;
wire [STATUS_WIDTH-1:0] status_data_10;
wire [STATUS_WIDTH-1:0] status_data_11;
wire [STATUS_WIDTH-1:0] status_data_12;
wire [STATUS_WIDTH-1:0] status_data_13;
wire [STATUS_WIDTH-1:0] status_data_14;
wire [STATUS_WIDTH-1:0] status_data_15;

reg [STATUS_WIDTH-1:0] tex_status_read_data;
reg [STATUS_WIDTH-1:0] alu_status_read_data;

wire [0:0] cfs_update;

reg [0:0] sx0_exp_count_rdy_q;
reg [0:0] sx0_exp_pos_avail_q;
reg [6:0] sx0_exp_buf_avail_q;
reg [0:0] sx1_exp_count_rdy_q;
reg [0:0] sx1_exp_pos_avail_q;
reg [6:0] sx1_exp_buf_avail_q;

reg [0:0] pos_avail_q;
reg [6:0] buf_avail_q;
```

```
  reg [1:0] cfs_phase_q;



  // -----------------------------------------------------------------------------------------
  // -- combinational logic --
  // -----------------------------------------------------------------------------------------

  // - qualify the TP and AIS done signals with the vector type (since same done goes to both
  //   vtx and pix thread buffers)
  // - the ais dones will not occur on the same cycle since they're interleaved

  wire [0:0] qual_tp_done = tp_done_q & (tp_thread_type_q == thread_type_strap);

  wire [0:0] qual_ais0_done = ais0_done & (ais0_thread_type == thread_type_strap);
  wire [0:0] qual_ais1_done = ais1_done & (ais1_thread_type == thread_type_strap);

  wire [0:0] qual_ais_done = qual_ais0_done | qual_ais1_done;

  wire [TID_WIDTH-1:0] ais_thread_id = ais0_done ? ais0_thread_id : ais1_thread_id;


  // - mux the thread_id/winner down from two sources to one before sending to the status bits
  //   (to update thread_valid)
  // - note that tex and alu arbiters provide winner_acks on alternate cycles, so tex_winner_ack
```

```
  //   and alu_winner_ack cannot occur on the same cycle

  wire [0:0] winner_ack = tex_winner_ack | alu_winner_ack;
  wire [TID_WIDTH-1:0] winner = tex_winner_ack ? tex_winner_q : alu_winner_q;


  // - buffer full bit is just the MSB of the location counter

  wire [0:0] buffer_full_q = full_cnt_q[TID_WIDTH];


  // - rtr the ISM ctl pkt data when buffer is not full AND when not loading CFS update data
  //   - CFS updates take priority over ISM loads
  //   - CFS updates are coordinates by cfs_phase in the ctl flow sequencers

  assign tb_rtr = ~buffer_full_q & ~cfs_update;


  // - busy whenever the buffer is not empty (i.e. when full count != 0)

  wire [0:0] busy = |full_cnt_q;


  // - push a thread when there's a valid transfer from the ISM

```

```
  wire [0:0] push_thread = ism_rts & tb_rtr;


  // - pop a thread when alu_status last_instr bit is set (and delayed winner ack pulses)

  wire [0:0] pop_thread = alu_status_q[12] & alu_winner_ack_q1;


  // ---------------------------
  // -- State Mem connections --
  // ---------------------------

  // --------
  // - read -
  // --------
  // - state mem reads are shared btwn tex arb and alu arb (tex and alu reqs are mixed in the buffer)

  //wire [3:0] state_rd_addr = state_read_phase ? tex_winner_q  : alu_winner_q;
  // - note: the state mem read address is the thread_id from the winning status register
  wire [TID_WIDTH-1:0] tex_winner_thread_id = tex_status_read_data[STATUS_WIDTH-1:STATUS_WIDTH-TID_WIDTH];
  wire [TID_WIDTH-1:0] alu_winner_thread_id = alu_status_read_data[STATUS_WIDTH-1:STATUS_WIDTH-TID_WIDTH];
  wire [TID_WIDTH-1:0] state_rd_addr = state_read_phase ? tex_winner_thread_id : alu_winner_thread_id;

```

```
  wire [0:0] tex_rd_en = tex_winner_ack & state_read_phase;
  wire [0:0] alu_rd_en = alu_winner_ack & ~state_read_phase;
  wire [0:0] state_rd_en = tex_rd_en | alu_rd_en;

  wire [ISM_STATE_WIDTH-1:0] ism_state_rd_data;
  wire [CFS_STATE_WIDTH-1:0] cfs_state_rd_data;
  wire [STATE_WIDTH-1:0]        state_rd_data;

  // -------------
  // - ISM write -
  // -------------
  // - state mem writes are shared btwn ISM ctl_pkt loads and CFS updates from 3 CFSs
  //   (updates have priority), and ALU Instr Seq updates
  // - the state mem is divided into three parts: ism_state, cfs_state, and alu_state

  // - ism info is loaded at the tail of the buffer

  wire [TID_WIDTH-1:0] ism_state_wr_addr = state_tail_ptr_q;
  wire [0:0] ism_state_wr_en   = push_thread;

  wire [ISM_STATE_WIDTH-1:0] ism_state_wr_data =
    {
     ism_lod_correct,  // 144 bits - gets dropped for vertex thread buffer
     ism_valid_bits,   // 64 bits - will get moved to alu state
     ism_gpr_base,     // 7 bits
```

```
      ism_context_id    // 3 bits
     };

   // -------------
   // - CFS write -
   // -------------
   // - cfs mem is written normally by cfs update, but part of cfs mem is initialized by ism load

   // - first mux between the two ALU CFS update inputs

   wire [0:0]                          acfs_update = acfs0_update | acfs1_update;  //  these  are  asserted at different times
   wire [0:0]                          acfs_thread_type   =   cfs_phase_q[1]   ?   acfs1_thread_type      : acfs0_thread_type;
   wire [CFS_STATE_WIDTH-1:0]   acfs_state = cfs_phase_q[1] ? acfs1_state  : acfs0_state;
   wire [STATUS_WIDTH-1:0]         acfs_status     =     cfs_phase_q[1]     ?     acfs1_status         : acfs0_status;

   // - second mux between the tex and alu cfs update info
   // - need to swap the mux sel polarity since the update is reg'd out of the CFS and thus occurs
   //   one cycle after its associated phase

   wire [CFS_STATE_WIDTH-1:0] cfs_state     = cfs_phase_q[0] ? tcfs_state  : acfs_state;
   wire [STATUS_WIDTH-1:0]    cfs_status    = cfs_phase_q[0] ? tcfs_status : acfs_status;
   //wire    [TID_WIDTH-1:0]          cfs_thread_id    =    cfs_phase_q[0]    ?    tcfs_status[19:16]    : acfs_status[19:16];
   //wire [0:0]              cfs_thread_type  = cfs_phase[0] ? tcfs_thread_type  : acfs_thread_type;
```

```

   assign cfs_update = (tcfs_update & (tcfs_thread_type == thread_type_strap)) |
                   (acfs_update   &   (acfs_thread_type   ==   thread_type_strap));     //   asserted   on different cfs_phases

   wire [12:0] cfs_instr_ptr      =   cfs_state[CFS_STATE_WIDTH-1:CFS_STATE_WIDTH-13];

   wire [3:0] cfs_exec_cnt        =       cfs_state[CFS_STATE_WIDTH-14:CFS_STATE_WIDTH-17];
   wire [0:0] cfs_export_id      = cfs_state[CFS_STATE_WIDTH-18];
   wire [6:0] cfs_param_ptr     =       cfs_state[CFS_STATE_WIDTH-19:CFS_STATE_WIDTH-25];

   wire [TID_WIDTH-1:0] cfs_thread_id = cfs_status[STATUS_WIDTH-1:STATUS_WIDTH-TID_WIDTH];
   // status[15] is reserved
   wire [0:0] cfs_alu_instr_pending = cfs_status[14];
   wire [0:0] cfs_pulse_sx       = cfs_status[13];
   wire [0:0] cfs_last_instr      = cfs_status[12];
   wire [0:0] cfs_pos_allocated           = cfs_status[10];
   wire [1:0] cfs_alloc_type     = cfs_status[9:8];
   wire [3:0] cfs_alloc_size     = cfs_status[7:4];
   wire [0:0] cfs_tex_read_pending = cfs_status[3];
   wire [0:0] cfs_serial = cfs_status[2];
   wire [0:0] cfs_resource        = cfs_status[1];
   wire [0:0] cfs_thread_valid = cfs_status[0];

```

```
   wire [TID_WIDTH-1:0] cfs_state_wr_addr = cfs_update ? cfs_thread_id : state_tail_ptr_q;
   wire [0:0] cfs_state_wr_en   = push_thread | cfs_update;

   wire [12:0] instr_ptr = cfs_update ? cfs_instr_ptr : {ism_instr_ptr, LO};  // 13 bits
   wire [03:0] exec_cnt  = cfs_update ? cfs_exec_cnt : 4'b0;  // 4 bits

   wire [CFS_STATE_WIDTH-1:0] cfs_state_wr_data =
      {
      instr_ptr,  // 13 bits
      exec_cnt,  // 4 bits
      cfs_export_id,      // 1 bit
      cfs_param_ptr      // 7 bits - gets dropped for pixel thread buffer
      };

   // --> note that valid bits and predicate bits will go into a separate memory - alu_state


   // ----------------------------
   // -- Status Read Data Muxes --
   // ----------------------------

   // - need one mux for the tex info, and one for the alu info (and since either can be in any line, need
   //   to mux all lines)

  always @(tex_winner_q or
```

```
          status_data_0 or status_data_1 or status_data_2 or status_data_3 or
          status_data_4 or status_data_5 or status_data_6 or status_data_7 or
          status_data_8 or status_data_9 or status_data_10 or status_data_11 or
          status_data_12 or status_data_13 or status_data_14 or status_data_15
         )
     begin
          case (tex_winner_q)
            4'h0: tex_status_read_data = status_data_0;
            4'h1: tex_status_read_data = status_data_1;
            4'h2: tex_status_read_data = status_data_2;
            4'h3: tex_status_read_data = status_data_3;
            4'h4: tex_status_read_data = status_data_4;
            4'h5: tex_status_read_data = status_data_5;
            4'h6: tex_status_read_data = status_data_6;
            4'h7: tex_status_read_data = status_data_7;
            4'h8: tex_status_read_data = status_data_8;
            4'h9: tex_status_read_data = status_data_9;
            4'ha: tex_status_read_data = status_data_10;
            4'hb: tex_status_read_data = status_data_11;
            4'hc: tex_status_read_data = status_data_12;
            4'hd: tex_status_read_data = status_data_13;
            4'he: tex_status_read_data = status_data_14;
            4'hf: tex_status_read_data = status_data_15;
            default: tex_status_read_data = {STATUS_WIDTH{X}};
          endcase // case(tex_winner)
```

```
    end // always @ (tex_winner or...

  always @(alu_winner_q or
          status_data_0 or status_data_1 or status_data_2 or status_data_3 or
          status_data_4 or status_data_5 or status_data_6 or status_data_7 or
          status_data_8 or status_data_9 or status_data_10 or status_data_11 or
          status_data_12 or status_data_13 or status_data_14 or status_data_15
         )
    begin
        case (alu_winner_q)
          4'h0: alu_status_read_data = status_data_0;
          4'h1: alu_status_read_data = status_data_1;
          4'h2: alu_status_read_data = status_data_2;
          4'h3: alu_status_read_data = status_data_3;
          4'h4: alu_status_read_data = status_data_4;
          4'h5: alu_status_read_data = status_data_5;
          4'h6: alu_status_read_data = status_data_6;
          4'h7: alu_status_read_data = status_data_7;
          4'h8: alu_status_read_data = status_data_8;
          4'h9: alu_status_read_data = status_data_9;
          4'ha: alu_status_read_data = status_data_10;
          4'hb: alu_status_read_data = status_data_11;
          4'hc: alu_status_read_data = status_data_12;
          4'hd: alu_status_read_data = status_data_13;
          4'he: alu_status_read_data = status_data_14;
```

```
          4'hf: alu_status_read_data = status_data_15;
          default: alu_status_read_data = {STATUS_WIDTH{X}};
        endcase // case(tex_winner)
    end // always @ (tex_winner or...


// --------------------------
// -- Status Write Decoder --
// --------------------------

// - selects status register to write initial status info from ISM

reg [15:0] ism_status_sel;

always @( push_thread or status_tail_ptr_q )
  begin
        ism_status_sel = 16'h0;
        if ( push_thread )
          begin
             ism_status_sel[status_tail_ptr_q] = HI;
          end
  end


// --------------------------
```

```
  // -- Status Winner Decoder --
  // --------------------------

  // - selects winning status register so its valid bit can be cleared (once the winning thread is
picked,
  //   it becomes invalid until it has finished running its target instructions)

  reg [15:0] winner_status_sel;

  always @( winner_ack or winner )
    begin
        winner_status_sel = 16'h0;
        if ( winner_ack )
          begin
             winner_status_sel[winner] = HI;
          end
    end


  // ----------------------------------------------------------------------------------------
  // -- registers --
  // ----------------------------------------------------------------------------------------

  // ---------------------------------
  // -- State Mem Head and Tail Ptr --
  // ---------------------------------
```

```

// state head and tail - works like a ring

wire [0:0] inc_state_tail_ptr = push_thread;
wire [0:0] inc_state_head_ptr = pop_thread;

always @(posedge clk)
  begin
        if ( reset )                  state_tail_ptr_q <= 0;
        else if ( inc_state_tail_ptr )  state_tail_ptr_q <= state_tail_ptr_q + 1;
        else                          state_tail_ptr_q <= state_tail_ptr_q;
  end

always @(posedge clk)
  begin
        if ( reset )                  state_head_ptr_q <= 0;
        else if ( inc_state_head_ptr )  state_head_ptr_q <= state_head_ptr_q + 1;
        else                          state_head_ptr_q <= state_head_ptr_q;
  end


// --------------------------
// -- Status Regs Tail Ptr --
// --------------------------

// status tail - status regs are shifted forward on a pop, so head ptr is always zero
// - the reason for shifting the status regs is to keep the ordering fixed so a
//   simple priority encoder can be used for arbitration (i.e. the request leaving
//   the head of the list is wired directly to the highest priority input of the
//   arbiter - on a pop, the entire list is shifted up one keeping the highest
//   piority thread at the top of the list)
// - so the tail pointer needs to be decremented on a pop, and incremented on a push

wire [0:0] inc_status_tail_ptr = push_thread;
wire [0:0] dec_status_tail_ptr = pop_thread;

always @(posedge clk)
  begin
       if ( reset )                    status_tail_ptr_q <= 0;
       else if ( inc_status_tail_ptr ) status_tail_ptr_q <= status_tail_ptr_q + 1;
       else if ( dec_status_tail_ptr ) status_tail_ptr_q <= status_tail_ptr_q - 1;
       else                            status_tail_ptr_q <= status_tail_ptr_q;
  end


// -----------------------------
// -- Buffer Location Counter --
// -----------------------------

always @(posedge clk)
```

```
  begin
       if ( reset )                  full_cnt_q <= 0;
       else if ( inc_state_tail_ptr ) full_cnt_q <= full_cnt_q + 1;  // adding a thread
       else if ( inc_state_head_ptr ) full_cnt_q <= full_cnt_q - 1;  // removing a thread
       else                          full_cnt_q <= full_cnt_q;
  end


// --------------------------------
// -- State Mem Output Registers --
// --------------------------------

// - register state mem read data for texture and alu winner lookup

wire [0:0] load_tex_state = tex_winner_ack_q;

always @(posedge clk)
  begin
       //if ( reset )              tex_state_q <= 0;
       if ( load_tex_state ) tex_state_q <= state_rd_data;
       else                  tex_state_q <= tex_state_q;
  end

wire [0:0] load_alu_state = alu_winner_ack_q;

```

```
always @(posedge clk)
  begin
       if ( load_alu_state ) alu_state_q <= state_rd_data;
       else                  alu_state_q <= alu_state_q;
  end


// ----------------------------------
// -- Status Regs Output Registers --
// ----------------------------------

// - register status reg read data for texture and alu winner lookup

wire [0:0] load_tex_status = tex_winner_ack_q;

always @(posedge clk)
  begin
       if ( reset )               tex_status_q <= 0;
       else if ( load_tex_status ) tex_status_q <= tex_status_read_data;
       else                       tex_status_q <= tex_status_q;
  end

wire [0:0] load_alu_status = alu_winner_ack_q;

always @(posedge clk)
```

```
  begin
       if ( reset )               alu_status_q <= 0;
       else if ( load_alu_status ) alu_status_q <= alu_status_read_data;
       else                       alu_status_q <= alu_status_q;
  end


// - register delays for tex_winner_ack and alu_winner_ack

always @(posedge clk)
  begin
       tex_winner_ack_q  <= tex_winner_ack;
       alu_winner_ack_q  <= alu_winner_ack;
       alu_winner_ack_q1 <= alu_winner_ack_q;
  end


// - input registers for signals from TPC

always @(posedge clk)
  begin
       tp_done_q        <= TP_SQ_data_rdy;
       tp_thread_type_q <= TP_SQ_type;
       tp_thread_id_q   <= TP_SQ_thread_id;
  end
```

```


// - register the output of the state sel muxes

// local gfx reg 8:1 muxes (the output is reg'd before use)

// --> state needs to come back from the final inst seq SM in order to do the gpr dealloc

parameter NUM_REG_WIDTH = 6;
wire [NUM_REG_WIDTH-1:0] num_reg;
reg [NUM_REG_WIDTH-1:0] num_reg_q;
sq_state_mux #( NUM_REG_WIDTH )
num_reg_sel (.state(3'b0), .input_set(num_reg_set), .mux_data_out(num_reg) );

always @(posedge clk)
  begin
        num_reg_q <= num_reg;
  end

wire [5:0] dealloc_space = num_reg_q;



// - input register for the SX buffer availability

always @(posedge clk)
```

```
    begin
      sx0_exp_count_rdy_q <= u0_SX_SQ_exp_count_rdy;
          sx0_exp_pos_avail_q <= u0_SX_SQ_exp_pos_avail;
          sx0_exp_buf_avail_q <= u0_SX_SQ_exp_buf_avail;
      sx1_exp_count_rdy_q <= u1_SX_SQ_exp_count_rdy;
          sx1_exp_pos_avail_q <= u1_SX_SQ_exp_pos_avail;
          sx1_exp_buf_avail_q <= u1_SX_SQ_exp_buf_avail;
    end

  // - save the availability info when valid


  always @(posedge clk)
    begin
          if ( sx0_exp_count_rdy_q | sx1_exp_count_rdy_q )
            begin
               pos_avail_q <= sx0_exp_pos_avail_q | sx1_exp_pos_avail_q;
               buf_avail_q <= (sx0_exp_buf_avail_q > sx1_exp_buf_avail_q) ? sx0_exp_buf_avail_q : sx1_exp_buf_avail_q;
            end
    end

  // - need to register cfs_phase since it's used for update data, and update data is reg'd out

  always @(posedge clk)
    begin
          cfs_phase_q <= cfs_phase;
```

```
  end


// -------------------------------------------------------------------------------------
// -- state machines --
// -------------------------------------------------------------------------------------



// ------------------------
// -- exit state machine --
// ------------------------

// the new state is the current state, which is part of the state of the thread leaving the buffer

sq_exit_sm
u_sq_exit_sm
(
  .new_state_rts(LO),
  .new_state(alu_state_q[CFS_STATE_WIDTH+2:CFS_STATE_WIDTH]),
  //.new_state_rtr(exit_sm_rtr),

  .state_diff(state_change),
  .old_state_q(old_state),

  .dealloc_req(dealloc_req),
```

```
    .dealloc_ack(dealloc_ack),

    .clk(clk),
    .reset(reset)
  );



  // -------------------------------------------------------------------------------------
  // -- module instatiations --
  // -------------------------------------------------------------------------------------

  // ------------------
  // -- State Memory --
  // ------------------

  // - ISM State Mem

  dum_mem_p2
   #( TB_ADDR_WIDTH, ISM_STATE_WIDTH, TB_DEPTH, 1 )
  ism_state_mem
  (
     .iWEN      (ism_state_wr_en),
     .iMEW      (ism_state_wr_en),
     .iWADR     (ism_state_wr_addr[3:0]),
    .iD (ism_state_wr_data),
    .iWCLK    (clk),

    .iMER     (state_rd_en),
    .iRADR    (state_rd_addr[3:0]),
    .oQ(ism_state_rd_data),
    .iRCLK    (clk)
  );

  // - CFS State Mem

  dum_mem_p2
   #( TB_ADDR_WIDTH, CFS_STATE_WIDTH, TB_DEPTH, 1 )
  cfs_state_mem
  (
    .iWEN     (cfs_state_wr_en),
    .iMEW     (cfs_state_wr_en),
    .iWADR    (cfs_state_wr_addr[3:0]),
    .iD (cfs_state_wr_data),
    .iWCLK    (clk),

    .iMER     (state_rd_en),
    .iRADR    (state_rd_addr[3:0]),
    .oQ(cfs_state_rd_data),
    .iRCLK    (clk)
```

```
  );


  assign state_rd_data = {ism_state_rd_data, cfs_state_rd_data};



  // ----------------------
  // -- Status Registers --
  // ----------------------

  sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
  u0_sq_status_reg
  (
    .thread_type_strap(thread_type_strap),

    .ism_load           (ism_status_sel[0]),    // loads ctl pkt info (and sets thread valid status bit)
    .ism_thread_id      (state_tail_ptr_q),     // thread id of newly written thread (state tail ptr)
    .ism_resource       (ism_resource),                  // resource bit :  tex=1, alu=0
    .ism_first_thread   (ism_first_thread),     // first thread of a new state

    .cfs_update         (cfs_update),           // load updated status info from CFS
    .cfs_thread_id      (cfs_thread_id),        // thread ID from CFS
    .cfs_alu_instr_pending(cfs_alu_instr_pending),// alu instr(s) from this thread in alu pipe
    .cfs_pulse_sx       (cfs_pulse_sx),                  // pulse SX from CFS
    .cfs_last_instr     (cfs_last_instr),       // last instruction from CFS
```

```
    .cfs_pos_allocated(cfs_pos_allocated),    // position allocated bit from CFS
    .cfs_alloc_type     (cfs_alloc_type),       // alloc type from CFS
    .cfs_alloc_size     (cfs_alloc_size),       // alloc size from CFS
    .cfs_tex_read_pending(cfs_tex_read_pending),// tex pend bit from CFS
    .cfs_serial  (cfs_serial),              // serial bit from CFS
    .cfs_resource       (cfs_resource),         // resource bit from CFS
    .cfs_thread_valid  (cfs_thread_valid),    // valid bit from CFS

    .sx_pos_avail       (pos_avail_q), // position available from SX
    .sx_buf_avail       (buf_avail_q), // buffer available from SX (0 to 127 2kbit buffers(), 2kbit = 32bits for 64 pixels)
    .param_cache_wptr_q       (param_cache_wptr_q),

    //.winner_ack       (winner_ack),  // winner is selected thread - clears thread_valid
    //.winner           (winner),         //
    .winner_sel         (winner_status_sel[0]),           //

    .tp_done            (qual_tp_done),         // tp done clears tex_read_pending
    .tp_thread_id       (tp_thread_id_q),       //

    .ais_done           (qual_ais_done),// ais done sets thread_valid
    .ais_thread_id      (ais_thread_id),//

    .pop_thread         (pop_thread),  // this will shift the status regs list

```

```
    .tex_req_q (tex_req_q[0]),          // tex request: simply the thread_valid with resource == tex
    .alu_req_q (alu_req_q[0]),          // alu request: function of the status bits and export buffer availability

    .status_in_q        (status_data_1),        // the status input (for shifting)
    .status_out_q       (status_data_0),        // the status output (for shifting, and sending to CFS)

    .clk(clk),
    .reset(reset)
  );

  sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
  u1_sq_status_reg (
    .thread_type_strap(thread_type_strap),
    .ism_load(ism_status_sel[1]), .ism_thread_id(state_tail_ptr_q),
    .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
    .cfs_update(cfs_update),
    .cfs_thread_id(cfs_thread_id),
    .cfs_alu_instr_pending(cfs_alu_instr_pending),
    .cfs_pulse_sx(cfs_pulse_sx),
    .cfs_last_instr(cfs_last_instr),
    .cfs_pos_allocated(cfs_pos_allocated),
    .cfs_alloc_type(cfs_alloc_type),
    .cfs_alloc_size(cfs_alloc_size),
      .cfs_tex_read_pending(cfs_tex_read_pending),
      .cfs_serial(cfs_serial),
      .cfs_resource(cfs_resource),
      .cfs_thread_valid(cfs_thread_valid),
      .sx_pos_avail(pos_avail_q),                         .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel        (winner_status_sel[1]),
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
      .tex_req_q(tex_req_q[1]), .alu_req_q(alu_req_q[1]),
      .status_in_q(status_data_2), .status_out_q(status_data_1),
      .clk(clk), .reset(reset)
    );

    sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
    u2_sq_status_reg (
      .thread_type_strap(thread_type_strap),
      .ism_load(ism_status_sel[2]), .ism_thread_id(state_tail_ptr_q),
      .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
      .cfs_update(cfs_update),         .cfs_serial(cfs_serial),         .cfs_resource(cfs_resource),
  .cfs_alloc_type(cfs_alloc_type),
      .cfs_alloc_size(cfs_alloc_size),                    .cfs_pos_allocated(cfs_pos_allocated),
  .cfs_last_instr(cfs_last_instr),
      .cfs_pulse_sx(cfs_pulse_sx),                        .cfs_thread_valid(cfs_thread_valid),
  .cfs_tex_read_pending(cfs_tex_read_pending),
      .cfs_thread_id(cfs_thread_id), .cfs_alu_instr_pending(cfs_alu_instr_pending),
```

```
      .sx_pos_avail(pos_avail_q),                         .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel        (winner_status_sel[2]),
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
      .tex_req_q(tex_req_q[2]), .alu_req_q(alu_req_q[2]),
      .status_in_q(status_data_3), .status_out_q(status_data_2),
      .clk(clk), .reset(reset)
    );

    sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
    u3_sq_status_reg (
      .thread_type_strap(thread_type_strap),
      .ism_load(ism_status_sel[3]), .ism_thread_id(state_tail_ptr_q),
      .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
      .cfs_update(cfs_update),         .cfs_serial(cfs_serial),         .cfs_resource(cfs_resource),
  .cfs_alloc_type(cfs_alloc_type),
      .cfs_alloc_size(cfs_alloc_size),                    .cfs_pos_allocated(cfs_pos_allocated),
  .cfs_last_instr(cfs_last_instr),
      .cfs_pulse_sx(cfs_pulse_sx),                        .cfs_thread_valid(cfs_thread_valid),
  .cfs_tex_read_pending(cfs_tex_read_pending),
      .cfs_thread_id(cfs_thread_id), .cfs_alu_instr_pending(cfs_alu_instr_pending),
      .sx_pos_avail(pos_avail_q),                         .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel        (winner_status_sel[3]),
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
```

```
      .tex_req_q(tex_req_q[3]), .alu_req_q(alu_req_q[3]),
      .status_in_q(status_data_4), .status_out_q(status_data_3),
      .clk(clk), .reset(reset)
    );

    sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
    u4_sq_status_reg (
      .thread_type_strap(thread_type_strap),
      .ism_load(ism_status_sel[4]), .ism_thread_id(state_tail_ptr_q),
      .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
      .cfs_update(cfs_update),         .cfs_serial(cfs_serial),         .cfs_resource(cfs_resource),
  .cfs_alloc_type(cfs_alloc_type),
      .cfs_alloc_size(cfs_alloc_size),                    .cfs_pos_allocated(cfs_pos_allocated),
  .cfs_last_instr(cfs_last_instr),
      .cfs_pulse_sx(cfs_pulse_sx),                        .cfs_thread_valid(cfs_thread_valid),
  .cfs_tex_read_pending(cfs_tex_read_pending),
      .cfs_thread_id(cfs_thread_id), .cfs_alu_instr_pending(cfs_alu_instr_pending),
      .sx_pos_avail(pos_avail_q),                         .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel        (winner_status_sel[4]),
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
      .tex_req_q(tex_req_q[4]), .alu_req_q(alu_req_q[4]),
      .status_in_q(status_data_5), .status_out_q(status_data_4),
      .clk(clk), .reset(reset)
    );

```

```
    sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
    u5_sq_status_reg (
      .thread_type_strap(thread_type_strap),
      .ism_load(ism_status_sel[5]), .ism_thread_id(state_tail_ptr_q),
      .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
      .cfs_update(cfs_update),         .cfs_serial(cfs_serial),         .cfs_resource(cfs_resource),
  .cfs_alloc_type(cfs_alloc_type),
      .cfs_alloc_size(cfs_alloc_size),                    .cfs_pos_allocated(cfs_pos_allocated),
  .cfs_last_instr(cfs_last_instr),
      .cfs_pulse_sx(cfs_pulse_sx),                        .cfs_thread_valid(cfs_thread_valid),
  .cfs_tex_read_pending(cfs_tex_read_pending),
      .cfs_thread_id(cfs_thread_id), .cfs_alu_instr_pending(cfs_alu_instr_pending),
      .sx_pos_avail(pos_avail_q),                         .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel        (winner_status_sel[5]),
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
      .tex_req_q(tex_req_q[5]), .alu_req_q(alu_req_q[5]),
      .status_in_q(status_data_6), .status_out_q(status_data_5),
      .clk(clk), .reset(reset)
    );

    sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
    u6_sq_status_reg (
      .thread_type_strap(thread_type_strap),
      .ism_load(ism_status_sel[6]), .ism_thread_id(state_tail_ptr_q),
      .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
      .cfs_update(cfs_update),     .cfs_serial(cfs_serial),     .cfs_resource(cfs_resource),
  .cfs_alloc_type(cfs_alloc_type),
      .cfs_alloc_size(cfs_alloc_size),                 .cfs_pos_allocated(cfs_pos_allocated),
  .cfs_last_instr(cfs_last_instr),
      .cfs_pulse_sx(cfs_pulse_sx),                  .cfs_thread_valid(cfs_thread_valid),
  .cfs_tex_read_pending(cfs_tex_read_pending),
      .cfs_thread_id(cfs_thread_id), .cfs_alu_instr_pending(cfs_alu_instr_pending),
      .sx_pos_avail(pos_avail_q),                         .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel      (winner_status_sel[6]),
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
      .tex_req_q(tex_req_q[6]), .alu_req_q(alu_req_q[6]),
      .status_in_q(status_data_7), .status_out_q(status_data_6),
      .clk(clk), .reset(reset)
    );

    sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
    u7_sq_status_reg (
      .thread_type_strap(thread_type_strap),
      .ism_load(ism_status_sel[7]), .ism_thread_id(state_tail_ptr_q),
      .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
      .cfs_update(cfs_update),     .cfs_serial(cfs_serial),     .cfs_resource(cfs_resource),
  .cfs_alloc_type(cfs_alloc_type),
      .cfs_alloc_size(cfs_alloc_size),                 .cfs_pos_allocated(cfs_pos_allocated),
  .cfs_last_instr(cfs_last_instr),
      .cfs_pulse_sx(cfs_pulse_sx),                  .cfs_thread_valid(cfs_thread_valid),
  .cfs_tex_read_pending(cfs_tex_read_pending),
```

```
      .cfs_thread_id(cfs_thread_id), .cfs_alu_instr_pending(cfs_alu_instr_pending),
      .sx_pos_avail(pos_avail_q),                         .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel      (winner_status_sel[7]),
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
      .tex_req_q(tex_req_q[7]), .alu_req_q(alu_req_q[7]),
      .status_in_q(status_data_8), .status_out_q(status_data_7),
      .clk(clk), .reset(reset)
    );

    sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
    u8_sq_status_reg (
      .thread_type_strap(thread_type_strap),
      .ism_load(ism_status_sel[8]), .ism_thread_id(state_tail_ptr_q),
      .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
      .cfs_update(cfs_update),     .cfs_serial(cfs_serial),     .cfs_resource(cfs_resource),
  .cfs_alloc_type(cfs_alloc_type),
      .cfs_alloc_size(cfs_alloc_size),                 .cfs_pos_allocated(cfs_pos_allocated),
  .cfs_last_instr(cfs_last_instr),
      .cfs_pulse_sx(cfs_pulse_sx),                  .cfs_thread_valid(cfs_thread_valid),
  .cfs_tex_read_pending(cfs_tex_read_pending),
      .cfs_thread_id(cfs_thread_id), .cfs_alu_instr_pending(cfs_alu_instr_pending),
      .sx_pos_avail(pos_avail_q),                         .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel      (winner_status_sel[8]),
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
```

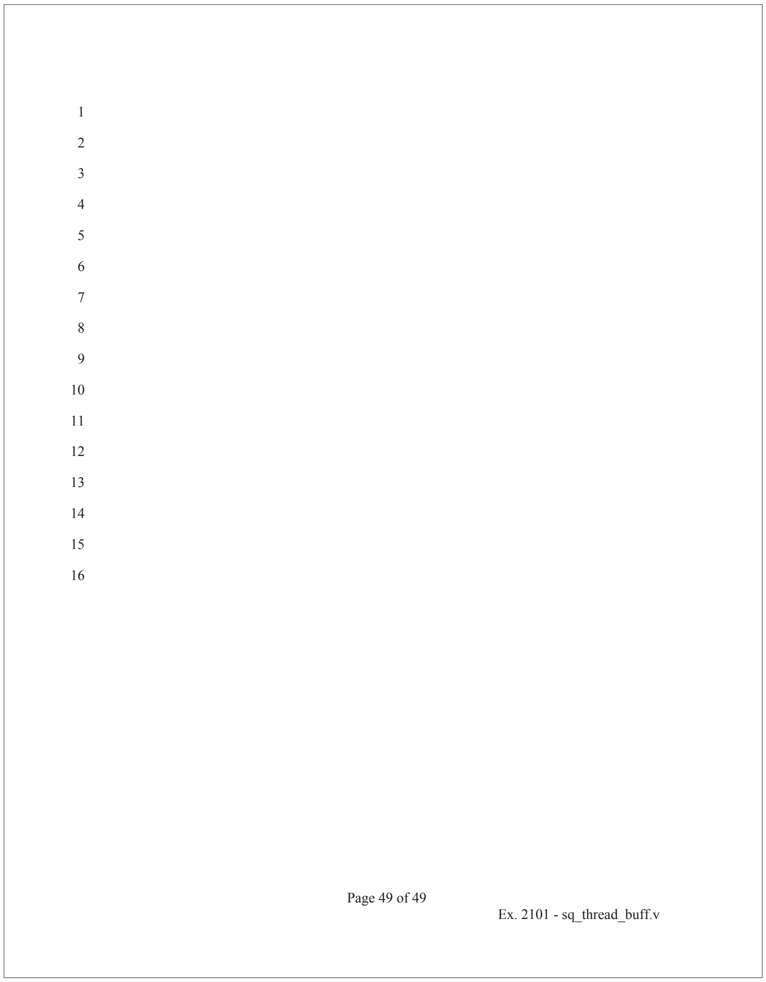
```
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
      .tex_req_q(tex_req_q[8]), .alu_req_q(alu_req_q[8]),
      .status_in_q(status_data_9), .status_out_q(status_data_8),
      .clk(clk), .reset(reset)
    );

    sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
    u9_sq_status_reg (
      .thread_type_strap(thread_type_strap),
      .ism_load(ism_status_sel[9]), .ism_thread_id(state_tail_ptr_q),
      .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
      .cfs_update(cfs_update),     .cfs_serial(cfs_serial),     .cfs_resource(cfs_resource),
  .cfs_alloc_type(cfs_alloc_type),
      .cfs_alloc_size(cfs_alloc_size),                 .cfs_pos_allocated(cfs_pos_allocated),
  .cfs_last_instr(cfs_last_instr),
      .cfs_pulse_sx(cfs_pulse_sx),                  .cfs_thread_valid(cfs_thread_valid),
  .cfs_tex_read_pending(cfs_tex_read_pending),
      .cfs_thread_id(cfs_thread_id), .cfs_alu_instr_pending(cfs_alu_instr_pending),
      .sx_pos_avail(pos_avail_q),                         .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel      (winner_status_sel[9]),
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
      .tex_req_q(tex_req_q[9]), .alu_req_q(alu_req_q[9]),
      .status_in_q(status_data_10), .status_out_q(status_data_9),
      .clk(clk), .reset(reset)
    );
```

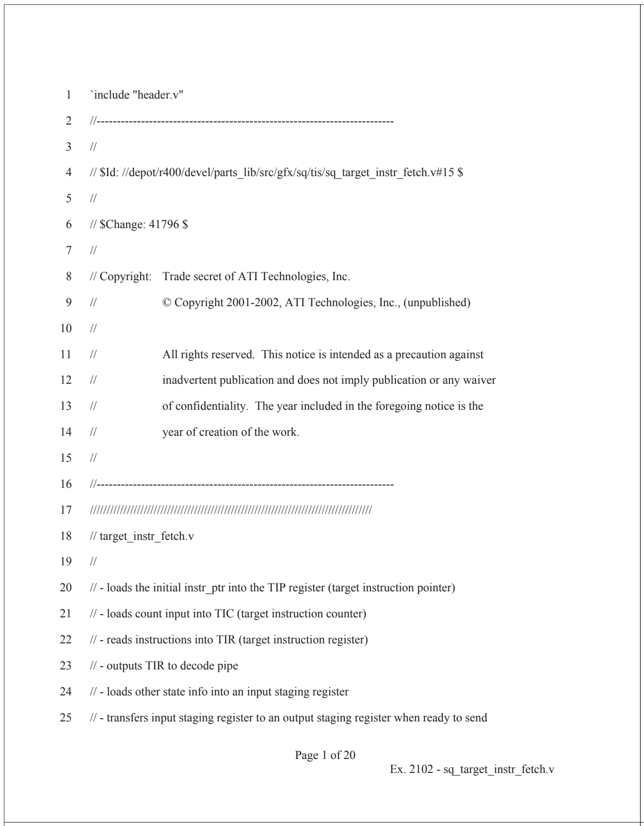
```

    sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
    u10_sq_status_reg (
      .thread_type_strap(thread_type_strap),
      .ism_load(ism_status_sel[10]), .ism_thread_id(state_tail_ptr_q),
      .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
      .cfs_update(cfs_update),     .cfs_serial(cfs_serial),     .cfs_resource(cfs_resource),
  .cfs_alloc_type(cfs_alloc_type),
      .cfs_alloc_size(cfs_alloc_size),                 .cfs_pos_allocated(cfs_pos_allocated),
  .cfs_last_instr(cfs_last_instr),
      .cfs_pulse_sx(cfs_pulse_sx),                  .cfs_thread_valid(cfs_thread_valid),
  .cfs_tex_read_pending(cfs_tex_read_pending),
      .cfs_thread_id(cfs_thread_id), .cfs_alu_instr_pending(cfs_alu_instr_pending),
      .sx_pos_avail(pos_avail_q),                         .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel      (winner_status_sel[10]),
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
      .tex_req_q(tex_req_q[10]), .alu_req_q(alu_req_q[10]),
      .status_in_q(status_data_11), .status_out_q(status_data_10),
      .clk(clk), .reset(reset)
    );

    sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
    u11_sq_status_reg (
      .thread_type_strap(thread_type_strap),
      .ism_load(ism_status_sel[11]), .ism_thread_id(state_tail_ptr_q),
      .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
      .cfs_update(cfs_update), .cfs_serial(cfs_serial), .cfs_resource(cfs_resource),
  .cfs_alloc_type(cfs_alloc_type),
      .cfs_alloc_size(cfs_alloc_size), .cfs_pos_allocated(cfs_pos_allocated),
  .cfs_last_instr(cfs_last_instr),
      .cfs_pulse_sx(cfs_pulse_sx), .cfs_thread_valid(cfs_thread_valid),
  .cfs_tex_read_pending(cfs_tex_read_pending),
      .cfs_thread_id(cfs_thread_id), .cfs_alu_instr_pending(cfs_alu_instr_pending),
      .sx_pos_avail(pos_avail_q), .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel        (winner_status_sel[11]),
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
      .tex_req_q(tex_req_q[11]), .alu_req_q(alu_req_q[11]),
      .status_in_q(status_data_12), .status_out_q(status_data_11),
      .clk(clk), .reset(reset)
    );

    sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
    u12_sq_status_reg (
      .thread_type_strap(thread_type_strap),
      .ism_load(ism_status_sel[12]), .ism_thread_id(state_tail_ptr_q),
      .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
      .cfs_update(cfs_update), .cfs_serial(cfs_serial), .cfs_resource(cfs_resource),
  .cfs_alloc_type(cfs_alloc_type),
      .cfs_alloc_size(cfs_alloc_size), .cfs_pos_allocated(cfs_pos_allocated),
  .cfs_last_instr(cfs_last_instr),
```

```
      .cfs_pulse_sx(cfs_pulse_sx), .cfs_thread_valid(cfs_thread_valid),
  .cfs_tex_read_pending(cfs_tex_read_pending),
      .cfs_thread_id(cfs_thread_id), .cfs_alu_instr_pending(cfs_alu_instr_pending),
      .sx_pos_avail(pos_avail_q), .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel        (winner_status_sel[12]),
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
      .tex_req_q(tex_req_q[12]), .alu_req_q(alu_req_q[12]),
      .status_in_q(status_data_13), .status_out_q(status_data_12),
      .clk(clk), .reset(reset)
    );

    sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
    u13_sq_status_reg (
      .thread_type_strap(thread_type_strap),
      .ism_load(ism_status_sel[13]), .ism_thread_id(state_tail_ptr_q),
      .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
      .cfs_update(cfs_update), .cfs_serial(cfs_serial), .cfs_resource(cfs_resource),
  .cfs_alloc_type(cfs_alloc_type),
      .cfs_alloc_size(cfs_alloc_size), .cfs_pos_allocated(cfs_pos_allocated),
  .cfs_last_instr(cfs_last_instr),
      .cfs_pulse_sx(cfs_pulse_sx), .cfs_thread_valid(cfs_thread_valid),
  .cfs_tex_read_pending(cfs_tex_read_pending),
      .cfs_thread_id(cfs_thread_id), .cfs_alu_instr_pending(cfs_alu_instr_pending),
      .sx_pos_avail(pos_avail_q), .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel        (winner_status_sel[13]),
```

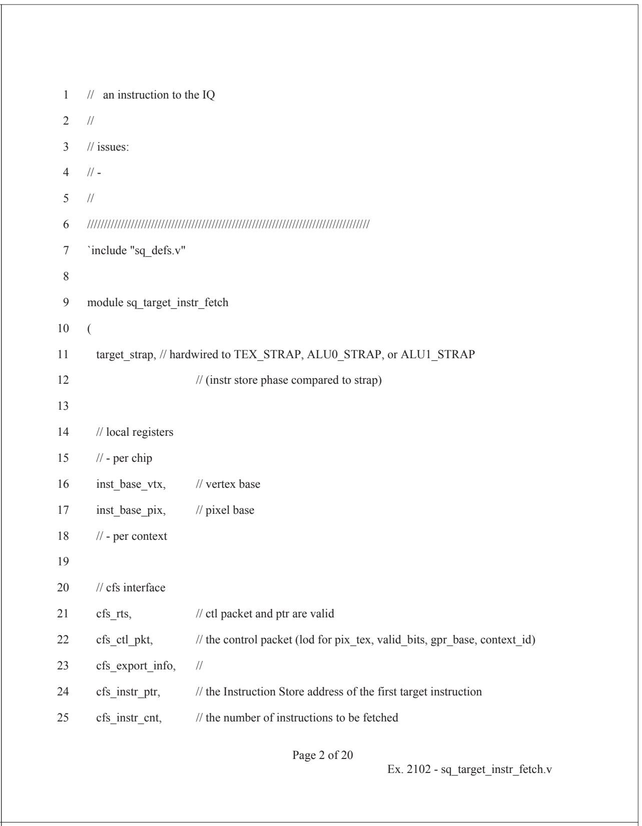
```
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
      .tex_req_q(tex_req_q[13]), .alu_req_q(alu_req_q[13]),
      .status_in_q(status_data_14), .status_out_q(status_data_13),
      .clk(clk), .reset(reset)
    );

    sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
    u14_sq_status_reg (
      .thread_type_strap(thread_type_strap),
      .ism_load(ism_status_sel[14]), .ism_thread_id(state_tail_ptr_q),
      .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
      .cfs_update(cfs_update), .cfs_serial(cfs_serial), .cfs_resource(cfs_resource),
  .cfs_alloc_type(cfs_alloc_type),
      .cfs_alloc_size(cfs_alloc_size), .cfs_pos_allocated(cfs_pos_allocated),
  .cfs_last_instr(cfs_last_instr),
      .cfs_pulse_sx(cfs_pulse_sx), .cfs_thread_valid(cfs_thread_valid),
  .cfs_tex_read_pending(cfs_tex_read_pending),
      .cfs_thread_id(cfs_thread_id), .cfs_alu_instr_pending(cfs_alu_instr_pending),
      .sx_pos_avail(pos_avail_q), .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel        (winner_status_sel[14]),
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
      .tex_req_q(tex_req_q[14]), .alu_req_q(alu_req_q[14]),
      .status_in_q(status_data_15), .status_out_q(status_data_14),
      .clk(clk), .reset(reset)
```

```
    );

    sq_status_reg #( TID_WIDTH, STATUS_WIDTH )
    u15_sq_status_reg (
      .thread_type_strap(thread_type_strap),
      .ism_load(ism_status_sel[15]), .ism_thread_id(state_tail_ptr_q),
      .ism_resource(ism_resource), .ism_first_thread(ism_first_thread),
      .cfs_update(cfs_update), .cfs_serial(cfs_serial), .cfs_resource(cfs_resource),
  .cfs_alloc_type(cfs_alloc_type),
      .cfs_alloc_size(cfs_alloc_size), .cfs_pos_allocated(cfs_pos_allocated),
  .cfs_last_instr(cfs_last_instr),
      .cfs_pulse_sx(cfs_pulse_sx), .cfs_thread_valid(cfs_thread_valid),
  .cfs_tex_read_pending(cfs_tex_read_pending),
      .cfs_thread_id(cfs_thread_id), .cfs_alu_instr_pending(cfs_alu_instr_pending),
      .sx_pos_avail(pos_avail_q), .sx_buf_avail(buf_avail_q),
  .param_cache_wptr_q(param_cache_wptr_q),
      .winner_sel        (winner_status_sel[15]),
      .tp_done(qual_tp_done), .tp_thread_id(tp_thread_id_q),
      .ais_done(qual_ais_done), .ais_thread_id(ais_thread_id), .pop_thread(pop_thread),
      .tex_req_q(tex_req_q[15]), .alu_req_q(alu_req_q[15]),
      .status_in_q({STATUS_WIDTH{LO}}), .status_out_q(status_data_15),
      .clk(clk), .reset(reset)
    );




  endmodule
```

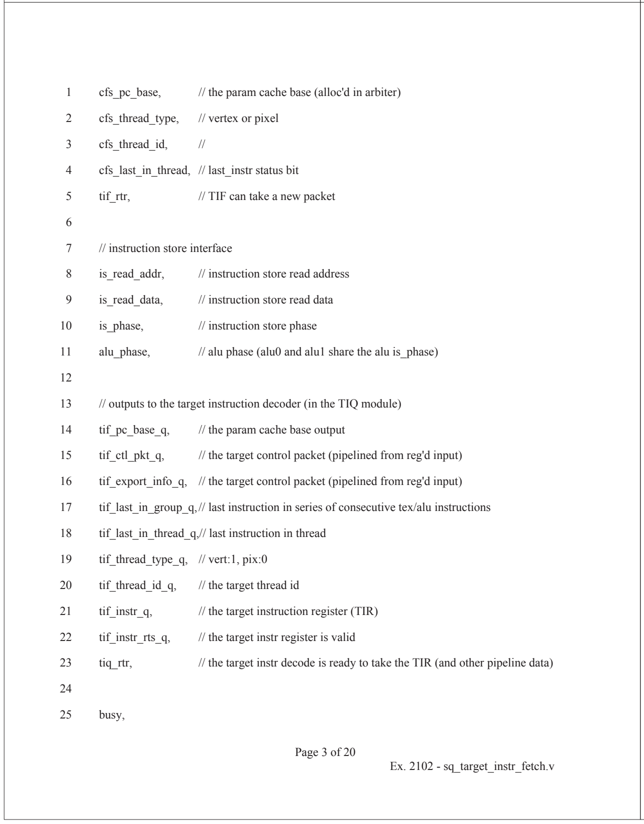




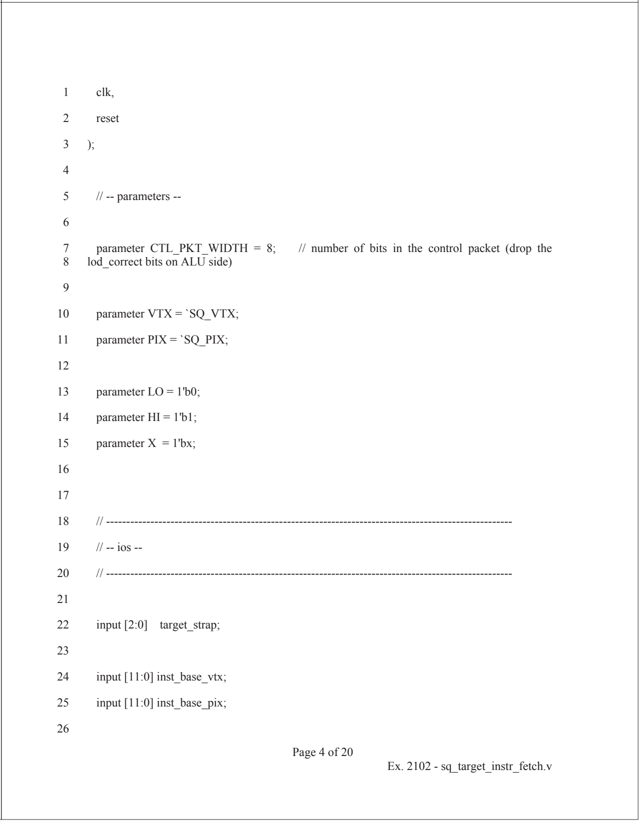
```
`include "header.v"
//---------------------------------------------------------------------------
//
// $Id: //depot/r400/devel/parts_lib/src/gfx/sq/tis/sq_target_instr_fetch.v#15 $
//
// $Change: 41796 $
//
// Copyright:   Trade secret of ATI Technologies, Inc.
//              © Copyright 2001-2002, ATI Technologies, Inc., (unpublished)
//
//              All rights reserved.  This notice is intended as a precaution against
//              inadvertent publication and does not imply publication or any waiver
//              of confidentiality.  The year included in the foregoing notice is the
//              year of creation of the work.
//
//---------------------------------------------------------------------------
/////////////////////////////////////////////////////////////////////////////////
// target_instr_fetch.v
//
// - loads the initial instr_ptr into the TIP register (target instruction pointer)
// - loads count input into TIC (target instruction counter)
// - reads instructions into TIR (target instruction register)
// - outputs TIR to decode pipe
// - loads other state info into an input staging register
// - transfers input staging register to an output staging register when ready to send
```



```
//   an instruction to the IQ
//
// issues:
// -
//
/////////////////////////////////////////////////////////////////////////////////
`include "sq_defs.v"

module sq_target_instr_fetch
(
  target_strap, // hardwired to TEX_STRAP, ALU0_STRAP, or ALU1_STRAP
                // (instr store phase compared to strap)

  // local registers
  // - per chip
  inst_base_vtx,        // vertex base
  inst_base_pix,        // pixel base
  // - per context

  // cfs interface
  cfs_rts,              // ctl packet and ptr are valid
  cfs_ctl_pkt,          // the control packet (lod for pix_tex, valid_bits, gpr_base, context_id)
  cfs_export_info,      //
  cfs_instr_ptr,        // the Instruction Store address of the first target instruction
  cfs_instr_cnt,        // the number of instructions to be fetched
```



```
  cfs_pc_base,          // the param cache base (alloc'd in arbiter)
  cfs_thread_type,      // vertex or pixel
  cfs_thread_id,        //
  cfs_last_in_thread,   // last_instr status bit
  tif_rtr,              // TIF can take a new packet

  // instruction store interface
  is_read_addr,         // instruction store read address
  is_read_data,         // instruction store read data
  is_phase,             // instruction store phase
  alu_phase,            // alu phase (alu0 and alu1 share the alu is_phase)

  // outputs to the target instruction decoder (in the TIQ module)
  tif_pc_base_q,        // the param cache base output
  tif_ctl_pkt_q,        // the target control packet (pipelined from reg'd input)
  tif_export_info_q,    // the target control packet (pipelined from reg'd input)
  tif_last_in_group_q,// last instruction in series of consecutive tex/alu instructions
  tif_last_in_thread_q,// last instruction in thread
  tif_thread_type_q,    // vert:1, pix:0
  tif_thread_id_q,      // the target thread id
  tif_instr_q,          // the target instruction register (TIR)
  tif_instr_rts_q,      // the target instr register is valid
  tiq_rtr,              // the target instr decode is ready to take the TIR (and other pipeline data)

  busy,
```



```
  clk,
  reset
);

  // -- parameters --

  parameter CTL_PKT_WIDTH = 8;      // number of bits in the control packet (drop the lod_correct bits on ALU side)

  parameter VTX = `SQ_VTX;
  parameter PIX = `SQ_PIX;

  parameter LO = 1'b0;
  parameter HI = 1'b1;
  parameter X  = 1'bx;


  // ---------------------------------------------------------------------------------------------------
  // -- ios --
  // ---------------------------------------------------------------------------------------------------

  input [2:0]    target_strap;

  input [11:0] inst_base_vtx;
  input [11:0] inst_base_pix;

input                   cfs_rts;
input [11:0]            cfs_instr_ptr;
input [11:0]            cfs_instr_cnt;
input [CTL_PKT_WIDTH-1:0]       cfs_ctl_pkt;
input [5:0]                     cfs_thread_id;
input [1:0]                     cfs_export_info;
input [6:0]                     cfs_pc_base;
input                   cfs_thread_type;
input                   cfs_last_in_thread;
output                          tif_rtr;
reg                             tif_rtr;

output [11:0]           is_read_addr;
input  [95:0] is_read_data;
input  [1:0]            is_phase;
input  [0:0]            alu_phase;

output [95:0]           tif_instr_q;
reg    [95:0]           tif_instr_q;
output [CTL_PKT_WIDTH-1:0]    tif_ctl_pkt_q;
reg    [CTL_PKT_WIDTH-1:0]    tif_ctl_pkt_q;
output [5:0]            tif_thread_id_q;
reg    [5:0]                    tif_thread_id_q;
output [1:0]            tif_export_info_q;
reg    [1:0]                    tif_export_info_q;
```

```
output [6:0]            tif_pc_base_q;
reg    [6:0]                    tif_pc_base_q;
output                          tif_instr_rts_q;
reg                             tif_instr_rts_q;
output                          tif_last_in_group_q;
reg                             tif_last_in_group_q;
output                          tif_last_in_thread_q;
reg                             tif_last_in_thread_q;
output                          tif_thread_type_q;
reg                             tif_thread_type_q;

input                   tiq_rtr;

output busy;
input clk;
input reset;


// -----------------------------------------------------------------------------------------------------
// -- internal signals --
// -----------------------------------------------------------------------------------------------------

reg [11:0]      tic_q;
reg [11:0]      tip_q;

```

```
reg [CTL_PKT_WIDTH-1:0]         isr_ctl_pkt_q;
reg [1:0]                       isr_export_info_q;
reg [6:0]                       isr_pc_base_q;
reg [5:0]                       isr_thread_id_q;
reg                             isr_thread_type_q;
reg                             isr_last_in_thread_q;

reg ld_tir;
reg ld_tip;
reg inc_tip;
reg dec_tic;
reg last_in_group;

reg [1:0] current_state;
reg [1:0] next_state;

reg    tif_instr_rts_q1;

// -----------------------------------------------------------------------------------------------------
// -- module instatiations --
// -----------------------------------------------------------------------------------------------------


// -----------------------------------------------------------------------------------------------------
// -- combinational logic --
```

```
// -----------------------------------------------------------------------------------------------------

// ld_tir is a SM reg'd out signal
wire      busy = |current_state | ld_tir | tif_instr_rts_q | tif_instr_rts_q1;

// - just re-assign the IS read address to the TIP

assign is_read_addr = tip_q;


// -----------------------------------------------------------------------------------------------------
// -- registers --
// -----------------------------------------------------------------------------------------------------

// --------------------------------------
// -- Target Instruction Pointer (TIP) --
// --------------------------------------
// - initially loaded with instr_ptr from CFS
// - inc'd by 1 to next sequential address
// - wraps back to inst_base

wire tip_eq_end_of_mem = (tip_q == 12'hfff);
wire tip_eq_pix_base   = (tip_q == inst_base_pix);

wire vtx_wrap = (cfs_thread_type == VTX) & tip_eq_pix_base;
```

```
  wire pix_wrap = (cfs_thread_type == PIX) & tip_eq_end_of_mem;

  always @(posedge clk)
    begin
        if ( ld_tip )          tip_q <= cfs_instr_ptr;
        else if ( inc_tip )
          if ( vtx_wrap )      tip_q <= inst_base_vtx;
          else if ( pix_wrap ) tip_q <= inst_base_pix;
          else                 tip_q <= tip_q + 1;
        else                   tip_q <= tip_q;
    end


  // --------------------------------------
  // -- Target Instruction Counter (TIC) --
  // --------------------------------------
  // - reset to 0
  // - loaded with reg'd in count at the same time TIP is loaded with the initial IS addr
  // - dec'd by 1 for every instruction fetched

  always @(posedge clk)
    begin
        if (reset)        tic_q <= 0;
        else if (ld_tip)  tic_q <= cfs_instr_cnt;
        else if (dec_tic) tic_q <= tic_q - 1;
```

```
        else              tic_q <= tic_q;
    end


  // --------------------------------------
  // -- Input Staging Register --
  // --------------------------------------
  // - register in control packet, vector type, and pc_base from the Arbiter
  //   when starting a new series of instruction fetches

  always @(posedge clk)
    begin
        if (cfs_rts & tif_rtr)
          begin
            isr_export_info_q  <= cfs_export_info;
            isr_pc_base_q  <= cfs_pc_base;
            isr_ctl_pkt_q  <= cfs_ctl_pkt;
          isr_thread_type_q <= cfs_thread_type;
          isr_thread_id_q <= cfs_thread_id;
          isr_last_in_thread_q <= cfs_last_in_thread;
          end
        else
          begin
            isr_export_info_q  <= isr_export_info_q;
            isr_pc_base_q  <= isr_pc_base_q;
```

```
            isr_ctl_pkt_q  <= isr_ctl_pkt_q;
          isr_thread_type_q <= isr_thread_type_q;
            isr_thread_id_q <= isr_thread_id_q;
          isr_last_in_thread_q <= isr_last_in_thread_q;
          end
    end


  // --------------------------------------
  // -- Output Staging Register --
  // --------------------------------------
  // - hold misc info that must be passed on to the TIQ along with the instruction that was fetched from IS
  // - register ctl packet, type, cfs num, and last flag when loading the TIR
  // - this is a stallable pipeline stage - stall when TIQ is not ready (input staging register can in the
  //   mean time be loaded with new cfs data)

  always @(posedge clk)
    begin
        if (ld_tir)
          begin
            tif_export_info_q    <= isr_export_info_q;
            tif_pc_base_q    <= isr_pc_base_q;
            tif_ctl_pkt_q    <= isr_ctl_pkt_q;
          tif_last_in_group_q <= last_in_group;
```

```
          tif_last_in_thread_q <= isr_last_in_thread_q;
          tif_thread_type_q <= isr_thread_type_q;
          tif_thread_id_q <= isr_thread_id_q;
          end
        else
          begin
            tif_export_info_q    <= tif_export_info_q;
            tif_pc_base_q    <= tif_pc_base_q;
            tif_ctl_pkt_q    <= tif_ctl_pkt_q;
          tif_last_in_group_q <= tif_last_in_group_q;
          tif_last_in_thread_q <= tif_last_in_thread_q;
          tif_thread_type_q <= tif_thread_type_q;
            tif_thread_id_q <= tif_thread_id_q;
          end
    end

  // --------------------------------------
  // -- Target Instruction Register (TIR) --
  // --------------------------------------
  // - loaded with data read from instruction store
  // - the TIR is output to the target instruction queue (which does some decode in front of the queue)

  always @(posedge clk)
    begin
        if (ld_tir)  tif_instr_q <= is_read_data;
```

```
            else        tif_instr_q <= tif_instr_q;
        end


    // ----------------------------------------------------------------------------------------------------
    // -- one-bit state machines --
    // ----------------------------------------------------------------------------------------------------

    // tif_instr_rts_q
    // - sets and clears the valid (rts) bit for the TIR
    // - the TIR can be valid and the TIQ not ready to take it when the next series of fetches is
  started
    //   (i.e. in Input Staging Reg can be reloaded while the Output Staging Reg still has the last
    //    instruction of the previous cfs)

    always @(posedge clk)
       begin
            if (reset) tif_instr_rts_q <= LO;
            else
              case (tif_instr_rts_q)
                LO: tif_instr_rts_q <= ld_tir;          // - set when loading TIR
                HI: tif_instr_rts_q <= ~tiq_rtr;        // - clear when decoder/IQ is rdy
              endcase
        end

    // reg delayed version for busy
```

```

    always @(posedge clk)
      begin
            if (reset) tif_instr_rts_q1 <= LO;
            else       tif_instr_rts_q1 <= tif_instr_rts_q;
      end



    // ----------------------------------------------------------------------------------------------------
    // -- state machines --
    // ----------------------------------------------------------------------------------------------------


    // tif state machine
    // - load TIP with main_base + context_base + offset when starting a series of instruction
  fetches
    // - read IS using TIP as read address
    // - load IS read data into TIR
    // - decrement TIC and increment TIP with every load until count equals zero

    parameter IDLE       = 2'b00;
    parameter LD_TIP   = 2'b01;
    parameter FETCH    = 2'b10;
    parameter LD_TIR   = 2'b11;

    // reg'd outputs
    reg next_tif_rtr;
```

```
    reg next_ld_tir;
    reg next_last_in_group;

    // un-reg'd outputs
    //reg ld_tip;
    //reg inc_tip;
    //reg dec_tic;

    // state and output registers
    always @(posedge clk)
       begin
            if (reset)
              begin
                 current_state <= IDLE;
                 tif_rtr <= HI;
                 ld_tir <= LO;
             last_in_group <= LO;
          end
            else
              begin
                 current_state <= next_state;
                 tif_rtr <= next_tif_rtr;
                 ld_tir <= next_ld_tir;
             last_in_group <= next_last_in_group;
          end
```

```
       end

    // next state logic
    //always @(*)
    always @(
            cfs_rts or alu_phase or is_phase or target_strap or tif_instr_rts_q or tic_q or
               current_state
            )
      begin
            // default assignments
            next_state = IDLE;
            next_tif_rtr = LO;
            next_ld_tir = LO;
         next_last_in_group = LO;

         ld_tip = LO;
         inc_tip = LO;
         dec_tic = LO;

            case (current_state)
              IDLE:
           begin
                 // - send rdy back to the arbiter/ctl flow mgr while waiting for cfs_rts
                 // - the inputs will be registered on cfs_rts
                 // - kick off the state machine and deassert tif rdy on cfs_rts
            next_tif_rtr = HI;

            if (cfs_rts)
              begin
                 ld_tip = HI;
                 next_tif_rtr = LO;
             next_state = FETCH;
          end
         end

         LD_TIP:
     begin
            // - not used...
            // - wait one cycle for instr_ptr to be reg'd into this module
            //ld_tip = HI;
            //next_state = FETCH;
         end

         FETCH:
     begin
            // - synch with instr store phase (wait until phase == hardwired type)
         //
            //   {is_phase, alu_phase}    type
            //   --------------------     ----
```

```
            //   001          TEX   - note that alu_phase is tied low on tex instance
            //   010          ALU0
            //   110          ALU1
         //
            // - also make sure the TIR does not still contain valid data (if it does, then we don't want to
            //   overwrite it by loading new data from the IS)

            if ( ({is_phase, alu_phase} == target_strap) & (~tif_instr_rts_q) )
              begin
                 inc_tip = HI;
                 dec_tic = HI;
             next_state = LD_TIR;
          end
         else
              begin
             next_state = FETCH;
          end
         end

         LD_TIR:
     begin
            // - load TIR at the end of the next cycle
            // - if TIC is zero, we're done; otherwise, fetch another instruction

            next_ld_tir = HI;
```

```
            if (tic_q == 0)
              begin
             next_last_in_group = HI;
             next_tif_rtr = HI;
             next_state = IDLE;
          end
         else
              begin
             next_state = FETCH;
          end
         end

       endcase // case(current_state)
    end // always @ (*)
    // - end tif state machine



endmodule
```

```
`include "header.v"
//--------------------------------------------------------------------------
//
// $Id: //depot/r400/devel/parts_lib/src/gfx/sq/misc/sq_export_alloc.v#13 $
//
// $Change: 44314 $
//
// Copyright:   Trade secret of ATI Technologies, Inc.
//              © Copyright 2001-2002, ATI Technologies, Inc., (unpublished)
//
//              All rights reserved.  This notice is intended as a precaution against
//              inadvertent publication and does not imply publication or any waiver
//              of confidentiality.  The year included in the foregoing notice is the
//              year of creation of the work.
//
//--------------------------------------------------------------------------
/////////////////////////////////////////////////////////////////////////////
// sq_export_alloc.v
//
// -
//
// issues:
// -
//
/////////////////////////////////////////////////////////////////////////////
```

```
`include "sq_defs.v"

module sq_export_alloc
(
   // - inputs from local registers
   vs_export_count_set,     // connected to SQ_PROGRAM_CNTL.VS_EXPORT_COUNT (4 bits)
   vs_export_mode_set,      // connected to SQ_PROGRAM_CNTL.VS_EXPORT_MODE (3 bits)
   ps_export_mode_set,      // connected to SQ_PROGRAM_CNTL.PS_EXPORT_MODE (3 bits)

   alu_arb_rts0,// ready to send the winner to CFS0
   alu_arb_rts1,// ready to send the winner to CFS1
   alu_arb_context_id, // the state sent to the CFS
   alu_arb_status,     // the status sent to the CFS
   alu_arb_thread_type,     // vtx or pix

   alu0_cfs_rtr, // ALU_CFS0 can accept a thread
   alu1_cfs_rtr, // ALU_CFS1 can accept a thread (for alu cfs's)

   pb_dealloc_cnt,     // param cache dealloc info from SC via the sq_ptr_buff
   pb_dealloc_vld,
   param_cache_wptr_q,      // output to SX and status regs

   // - sx export alloc interface
```

```
   SQ_SX_exp_valid,
   SQ_SX_exp_type,
   SQ_SX_exp_number,
   SQ_SX_exp_context_id,
   SQ_SX_exp_id,

   ais0_free_done,
   ais0_free_id,
   ais1_free_done,
   ais1_free_id,

   // - sx export dealloc interface
   SQ_SX_free_done,
   SQ_SX_free_id,

   // - export id interface
   cfs0_export_id,          // export_id that cfs0 is pushing down pipe 0 (sets global export_id)
   cfs_aif_xfc0,// cfs0 to aif0 transfer complete
   cfs1_export_id,          // export_id that cfs1 is pushing down pipe 1 (sets global export_id)
   cfs_aif_xfc1,// cfs1 to aif1 transfer complete

   global_export_id_q, // exp_id taken on arb_xfc, toggled on exp_valid, loaded on cfs_xfc

   clk,
```

```
   reset
);

   // -- parameters --

   parameter LO = 1'b0;
   parameter HI = 1'b1;
   parameter X  = 1'bx;


   // --------------------------------------------------------------------------------------------------
   // -- ios --
   // --------------------------------------------------------------------------------------------------

   //
   input [8*4-1:0] vs_export_count_set;
   input [8*3-1:0] vs_export_mode_set;
   input [8*4-1:0] ps_export_mode_set;

   input [0:0]              alu_arb_rts0;
   input [0:0]              alu_arb_rts1;
   input [2:0]              alu_arb_context_id;
   input [`SQ_STATUS_WIDTH-1:0]       alu_arb_status;
   input [0:0]              alu_arb_thread_type;
   input [0:0]              alu0_cfs_rtr;
input [0:0]                    alu1_cfs_rtr;

input [2:0] pb_dealloc_cnt;
input [0:0] pb_dealloc_vld;
output [6:0] param_cache_wptr_q;

//output [0:0] SQ_SX_exp_pix;

output [0:0] SQ_SX_exp_valid;
output [1:0] SQ_SX_exp_type;
output [1:0] SQ_SX_exp_number;
output [2:0] SQ_SX_exp_context_id;
output [0:0] SQ_SX_exp_id;

input  [0:0] ais0_free_done;
input  [0:0] ais0_free_id;
input  [0:0] ais1_free_done;
input  [0:0] ais1_free_id;

output [0:0] SQ_SX_free_done;
output [0:0] SQ_SX_free_id;

input  [0:0] cfs0_export_id;
input  [0:0] cfs_aif_xfc0;
input  [0:0] cfs1_export_id;
```

```
input  [0:0] cfs_aif_xfc1;

output [0:0] global_export_id_q;     // exp_id taken on arb_xfc, toggled on exp_valid, loaded on cfs_xfc


input clk;
input reset;


// -----------------------------------------------------------------------------------------------------
// -- internal signals --
// -----------------------------------------------------------------------------------------------------

reg [0:0]  exp_valid_q;        // one bit state machine



// -----------------------------------------------------------------------------------------------------
// -- module instatiations --
// -----------------------------------------------------------------------------------------------------

parameter VS_EXPORT_COUNT_WIDTH = 4;
wire [VS_EXPORT_COUNT_WIDTH-1:0] vs_export_count;
reg [VS_EXPORT_COUNT_WIDTH-1:0] vs_export_count_q;
sq_state_mux #( VS_EXPORT_COUNT_WIDTH )
vs_export_count_sel          (.state(alu_arb_context_id),          .input_set(vs_export_count_set), .mux_data_out(vs_export_count) );
```

```

parameter VS_EXPORT_MODE_WIDTH = 3;
wire [VS_EXPORT_MODE_WIDTH-1:0] vs_export_mode;
reg [VS_EXPORT_MODE_WIDTH-1:0] vs_export_mode_q;
sq_state_mux #( VS_EXPORT_MODE_WIDTH )
vs_export_mode_sel          (.state(alu_arb_context_id),          .input_set(vs_export_mode_set), .mux_data_out(vs_export_mode) );

parameter PS_EXPORT_MODE_WIDTH = 4;
wire [PS_EXPORT_MODE_WIDTH-1:0] ps_export_mode;
reg [PS_EXPORT_MODE_WIDTH-1:0] ps_export_mode_q;
sq_state_mux #( PS_EXPORT_MODE_WIDTH )
ps_export_mode_sel          (.state(alu_arb_context_id),          .input_set(ps_export_mode_set), .mux_data_out(ps_export_mode) );

always @(posedge clk)
  begin
       vs_export_count_q <= vs_export_count;
       vs_export_mode_q  <= vs_export_mode;
       ps_export_mode_q  <= ps_export_mode;
  end


// -----------------------------------------------------------------------------------------------------
// -- combinational logic --
// -----------------------------------------------------------------------------------------------------
```

```

wire [0:0] alu_arb_xfc0 = alu_arb_rts0 & alu0_cfs_rtr;
wire [0:0] alu_arb_xfc1 = alu_arb_rts1 & alu1_cfs_rtr;
wire [0:0] alu_arb_xfc  = alu_arb_xfc0 | alu_arb_xfc1;

// - alloc type/size status to SX export type/number decode

wire [0:0] thread_type = alu_arb_thread_type;
wire [1:0] alloc_type = alu_arb_status[9:8];
wire [3:0] alloc_size = alu_arb_status[7:4];

wire [7:0] alloc_cmd = {thread_type, alloc_type, alloc_size, ps_export_mode_q[0]};

reg [4:0] sx_exp_cmd;

always @( alloc_cmd )
  begin
      casez ( alloc_cmd )
        // - vtx pos alloc
        8'b1_01_0000_? : sx_exp_cmd = 5'b10_00_1;
        8'b1_01_0001_? : sx_exp_cmd = 5'b10_01_1;

        // - vtx pass thru
        8'b1_11_0011_? : sx_exp_cmd = 5'b11_00_1;
        8'b1_11_0111_? : sx_exp_cmd = 5'b11_01_1;
```

```
            8'b1_11_1011_? : sx_exp_cmd = 5'b11_10_1;

            // - pix without z
            8'b0_10_0000_0 : sx_exp_cmd = 5'b00_00_1;
            8'b0_10_0001_0 : sx_exp_cmd = 5'b00_01_1;
            8'b0_10_0010_0 : sx_exp_cmd = 5'b00_10_1;
            8'b0_10_0011_0 : sx_exp_cmd = 5'b00_11_1;

            // - pix with z
            8'b0_10_0001_1 : sx_exp_cmd = 5'b01_00_1;
            8'b0_10_0010_1 : sx_exp_cmd = 5'b01_01_1;
            8'b0_10_0011_1 : sx_exp_cmd = 5'b01_10_1;
            8'b0_10_0100_1 : sx_exp_cmd = 5'b01_11_1;

            // - pix pass thru
            8'b0_11_0011_? : sx_exp_cmd = 5'b11_00_1;
            8'b0_11_0111_? : sx_exp_cmd = 5'b11_01_1;
            8'b0_11_1011_? : sx_exp_cmd = 5'b11_10_1;

            default: sx_exp_cmd = 5'bxxxx0;
          endcase
      end

  //wire [0:0] exp_pix = sx_exp_cmd[5];

```

```
  wire [1:0] exp_type = sx_exp_cmd[4:3];
  wire [1:0] exp_number = sx_exp_cmd[2:1];
  wire [0:0] exp_valid = sx_exp_cmd[0];

  // also need to decode for param cache alloc

  wire [0:0] pc_alloc = (thread_type == HI) & (alloc_type == 2'b10) & alu_arb_xfc;

  // ------------------------------------------------------------------------------------------------------
  // -- registers --
  // ------------------------------------------------------------------------------------------------------

  // -- global export id --
  // - toggled when ID is assigned to a thread (same as when alloc is sent to SX)
  // - overwritten when thread leaves the CFS and enters the ALU Instr Fetch

  reg [0:0] global_export_id_q;

  always @(posedge clk)
    begin
        if ( reset )            global_export_id_q <= LO;
        else if ( cfs_aif_xfc0 )  global_export_id_q <= cfs0_export_id;
        else if ( cfs_aif_xfc1 )  global_export_id_q <= cfs1_export_id;
        else if ( exp_valid_q )   global_export_id_q <= ~global_export_id_q;
        else                      global_export_id_q <= global_export_id_q;
```

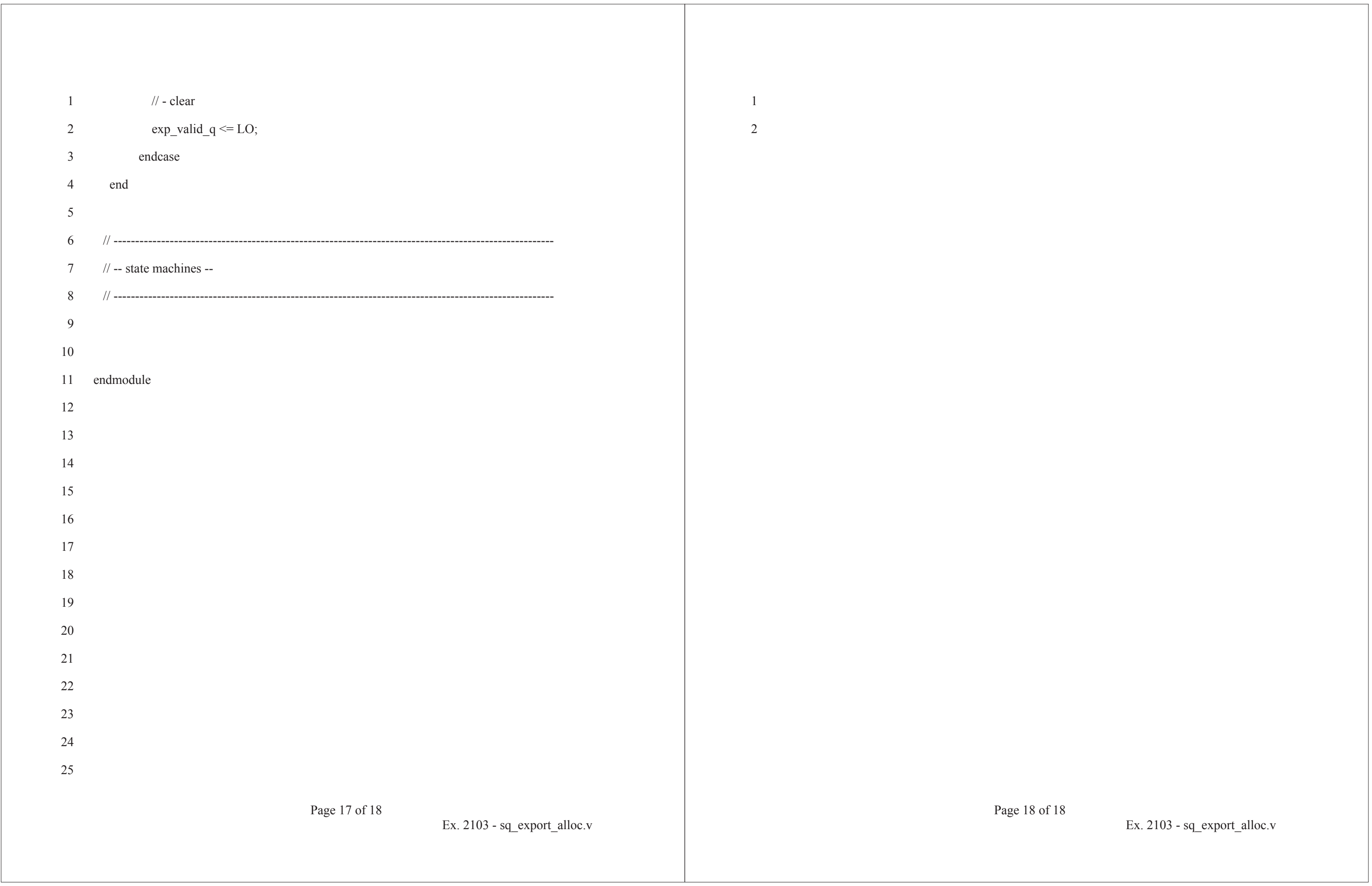
```
      end


  // - reg delay arb xfc for load of SQ_SX export alloc interface output registers

  reg [0:0] alu_arb_xfc_q;

  always @(posedge clk)
    begin
        alu_arb_xfc_q <= alu_arb_xfc;
      end


  // -------------------------------
  // -- param cache write pointer --
  // -------------------------------

  // - multiply the counts by 4 (left shift 2x) since there are 16 verts per P$ line, and thus 4 lines per vector
  // - the vs_export count ranges from 0 to 15 for 1 to 16 params, so need to add 1 to get the true count

  reg [6:0] param_cache_wptr_q;

  wire [6:0] vs_export_count_x4 = (vs_export_count_q + 1) << 2;  // 4 + 1 + 2 = 7 bits
  wire [4:0] pb_dealloc_cnt_x4  = pb_dealloc_cnt << 2;           // 3 + 2 = 5 bits
```

```

  always @(posedge clk)
    begin
        if ( reset )             param_cache_wptr_q <= 7'b0;
        else if ( pc_alloc )     param_cache_wptr_q <= param_cache_wptr_q + vs_export_count_x4;
        else if ( pb_dealloc_vld ) param_cache_wptr_q <= param_cache_wptr_q - pb_dealloc_cnt_x4;
        else                     param_cache_wptr_q <= param_cache_wptr_q;
    end


  // --------------------------------
  // -- SQ_SX_exp output registers --
  // --------------------------------

  //reg [0:0] SQ_SX_exp_pix;

  reg [0:0] SQ_SX_exp_valid;
  reg [1:0] SQ_SX_exp_type;
  reg [1:0] SQ_SX_exp_number;
  reg [0:0] SQ_SX_exp_id;
  reg [2:0] SQ_SX_exp_context_id;

  always @(posedge clk)
    begin

    SQ_SX_exp_valid <= exp_valid_q;

      if ( alu_arb_xfc )
        begin
        //SQ_SX_exp_pix <= ~exp_pix;
        SQ_SX_exp_context_id <= alu_arb_context_id;
        SQ_SX_exp_id <= alu_arb_xfc1;
        SQ_SX_exp_type <= exp_type;
        SQ_SX_exp_number <= exp_number;
        end
      else
        begin
        //SQ_SX_exp_pix <= SQ_SX_exp_pix;
        SQ_SX_exp_context_id <= SQ_SX_exp_context_id;
        SQ_SX_exp_id <= SQ_SX_exp_id;
        SQ_SX_exp_type <= SQ_SX_exp_type;
        SQ_SX_exp_number <= SQ_SX_exp_number;
        end
  end

// - delay free done and id to line up with last cycle of data out of the SP

reg [0:0] SQ_SX_free_done;
reg [0:0] SQ_SX_free_id;
```

```

reg [0:0] free_done_q0;
reg [0:0]   free_id_q0;

reg [0:0] free_done_q1;
reg [0:0]   free_id_q1;

reg [0:0] free_done_q2;
reg [0:0]   free_id_q2;

reg [0:0] free_done_q3;
reg [0:0]   free_id_q3;

reg [0:0] free_done_q4;
reg [0:0]   free_id_q4;

reg [0:0] free_done_q5;
reg [0:0]   free_id_q5;

reg [0:0] free_done_q6;
reg [0:0]   free_id_q6;

reg [0:0] free_done_q7;
reg [0:0]   free_id_q7;

```

```
reg [0:0] free_done_q8;
reg [0:0]   free_id_q8;

reg [0:0] free_done_q9;
reg [0:0]   free_id_q9;

always @(posedge clk)
  begin
    free_done_q0 <= ais0_free_done | ais1_free_done;
    free_done_q1 <= free_done_q0;
    free_done_q2 <= free_done_q1;
    free_done_q3 <= free_done_q2;
    free_done_q4 <= free_done_q3;
    free_done_q5 <= free_done_q4;
    free_done_q6 <= free_done_q5;
    free_done_q7 <= free_done_q6;
    free_done_q8 <= free_done_q7;
    free_done_q9 <= free_done_q8;
    SQ_SX_free_done <= free_done_q9;

    free_id_q0 <= ais1_free_done ? ais1_free_id : ais0_free_id;
    free_id_q1 <= free_id_q0;
    free_id_q2 <= free_id_q1;
    free_id_q3 <= free_id_q2;
    free_id_q4 <= free_id_q3;
```

```
    free_id_q5 <= free_id_q4;
    free_id_q6 <= free_id_q5;
    free_id_q7 <= free_id_q6;
    free_id_q8 <= free_id_q7;
    free_id_q9 <= free_id_q8;
    SQ_SX_free_id <= free_id_q9;
  end



// -----------------------------------------------------------------------------------------------------
// -- one-bit state machines --
// -----------------------------------------------------------------------------------------------------

// exp_valid

always @(posedge clk)
  begin
      if (reset)
        exp_valid_q <= LO;
      else
        case ( exp_valid_q )
          LO:
            // - set
            exp_valid_q <= exp_valid & alu_arb_xfc;
          HI:
```



```
                        // - clear
                        exp_valid_q <= LO;
                    endcase
        end

    // -----------------------------------------------------------------------------------------------------
    // -- state machines --
    // -----------------------------------------------------------------------------------------------------


endmodule
```

```
//                              -*- Mode: Verilog -*-
// Filename        : vector.v
// Description     : This module represent the implementation of the vector unit.
//                   There are 4 macc_reg modules  instantiated from this module.
// Author          : Andi Skende
// Created On      : Wed Jan 30 15:51:38 2002
// Last Modified By: .
// Last Modified On: .
// Update Count    : 0
// Status          : Unknown, Use with caution!

//`timescale 1ns / 1ps

module vector(/*AUTOARG*/
   // Outputs
   sp_sx_data, sp_tp_data, sp_sx_exp_dst, sp_sx_exp_pvalid,
   sp_sx_exporting, sp_sx_exp_alu_id,
   // Inputs
   sq_sp_instruct_start, sq_sp_instruct, sq_sp_stall, sclk, srst,
   sq_sp_exp_pvalid, sq_sp_exporting, sq_sp_exp_alu_id,
   sq_sp_wr_addr, sq_sp_gpr_rd_addr, sq_sp_gpr_phase_mux,
   sq_sp_channel_mask, sq_sp_pixel_mask, sq_sp_gpr_input_mux,
   sq_sp_mem_rd_ena, sq_sp_mem_wr_ena, sq_sp_wr_ena, iInterpolated,
   iAutoCount, sq_sp_constant, tp_sp_data, tp_sp_gpr_dst,
   tp_sp_gpr_cmask, tp_sp_data_valid, iVertexIndices
```

```
);

//--------------------------------------------------------------------/
//Instruction Interface
//--------------------------------------------------------------------/
input [0:0] sq_sp_instruct_start;
input [20:0] sq_sp_instruct; //four cycle transaction
input [0:0]    sq_sp_stall;   //when high...execute a NOP
input          sclk;
input          srst;

//---------------------------------------------------------------------//
//Export controls signals
//---------------------------------------------------------------------//
input [3:0]    sq_sp_exp_pvalid;
input [0:0]    sq_sp_exporting ;
input [0:0]    sq_sp_exp_alu_id;

//export destination pointer
wire [5:0]     sq_sp_exp_dst;

//--------------------------------------------------------------------/
//GPR read/write control interface
//--------------------------------------------------------------------/
input [6:0]    sq_sp_wr_addr;
```

```
   input [6:0]    sq_sp_gpr_rd_addr;
   input [1:0]    sq_sp_gpr_phase_mux;
   input [3:0]    sq_sp_channel_mask;
   input [3:0]    sq_sp_pixel_mask;
   input [1:0]    sq_sp_gpr_input_mux;
   input [0:0]    sq_sp_mem_rd_ena;
   input [0:0]    sq_sp_mem_wr_ena;
   input [0:0]    sq_sp_wr_ena;

   //--------------------------------------------------------------------/
   //Input Data from the top of the pipe
   //1. Interpolated Data
   //2. Vertex Indices
   //3. Autocount
   //--------------------------------------------------------------------/
   input [127:0] iInterpolated, iAutoCount,sq_sp_constant;
   input [127:0] tp_sp_data ; //texture data comming from the Texture Pipe/ Fetch Engine
   input [6:0]    tp_sp_gpr_dst; //destination address into gpr(s) for texture return data
   input [3:0]    tp_sp_gpr_cmask;//channel mask for the texture data being written into gpr(s)
   input [0:0]    tp_sp_data_valid;
   input [95:0]  iVertexIndices;

   reg [6:0]       q0_tp_gpr_dst, q1_tp_gpr_dst, q2_tp_gpr_dst;
   reg [3:0]       q0_tp_gpr_cmask, q1_tp_gpr_cmask, q2_tp_gpr_cmask;
   reg [0:0]       q0_tp_data_valid, q1_tp_data_valid, q2_tp_data_valid;
```

```

//---------------------------------------------------------------------/
//Output data going out from each vector unit to the SX blocks
//---------------------------------------------------------------------/
output [127:0] sp_sx_data;
reg [127:0]    osp_sx_data;
wire [127:0]  VectorResult0,VectorResult1,VectorResult2,VectorResult3;

//-------------------------------------------------------------------------//
//Register (GPR) data going out to the texture pipe
//-------------------------------------------------------------------------//
output [95:0] sp_tp_data;
reg [95:0]       osp_tp_data;

//-------------------------------------------------------------------------//
//Pipelined sequencer inputs...related to export functionality
//These signals are asserted at the same time as sq_sp_instruction_start
//--------------------------------------------------------------------------//

output [5:0]   sp_sx_exp_dst;
output [3:0]   sp_sx_exp_pvalid;
output [0:0]   sp_sx_exporting ;
output [0:0]   sp_sx_exp_alu_id;

//----------------------------------------------------------------------/
```

```
//There are four macc units present in a vector unit
//The macc units are phased out by one cycle from each other in their execution sequence
//------------------------------------------------------------------------/
reg [0:0]       q0_instruct_start, q1_instruct_start,q2_instruct_start;
reg [0:0]       q0_instruct_stall, q1_instruct_stall,q2_instruct_stall;
reg [20:0]      q0_instruct,q1_instruct,q2_instruct;
reg [6:0]       q0_gpr_wr_addr,q1_gpr_wr_addr,q2_gpr_wr_addr;
reg [6:0]       q0_gpr_rd_addr,q1_gpr_rd_addr,q2_gpr_rd_addr;
reg [0:0]       q0_gpr_mre,q1_gpr_mre,q2_gpr_mre;
reg [0:0]       q0_gpr_mwe,q1_gpr_mwe,q2_gpr_mwe;
reg [0:0]       q0_gpr_we,q1_gpr_we,q2_gpr_we;
reg [1:0]       q0_gpr_phase_mux,q1_gpr_phase_mux,q2_gpr_phase_mux;
reg [1:0]       q0_gpr_input_mux,q1_gpr_input_mux,q2_gpr_input_mux;
reg [3:0]       q0_gpr_cmask, q1_gpr_cmask,q2_gpr_cmask;

//-------------------------------------------------------------------------------------------
//The outputs of the mux unit selecting between iIntepolated, iAutoCount and iVertexIndices
//-------------------------------------------------------------------------------------------

reg [127:0]     InputData0, InputData1, InputData2,InputData3;

//---------------------------------------------------------------------------------------------
//GPR data coming from four different MACC_GPR units of the same vector unit
//-------------------------------------------------------------------------------------------
wire [127:0]  RegData0, RegData1, RegData2, RegData3;
```

```

//-----------------------------------------------------------------------------------
// This data comes from the Scalar Unit
//---------------------------------------------------------------------------------
wire [127:0]  ScalarData ;
reg [127:0]    q0_ScalarData, q1_ScalarData, q2_ScalarData;
wire [31:0]    ScalarResult;

//-------------------------------------------------------------------------------------
wire [31:0]    ScalarInput0, ScalarInput1,ScalarInput2,ScalarInput3;
reg [31:0]     ScalarInput;
reg [5:0]      ScalarOpcode;
wire [5:0]     ScalarOpcode0, ScalarOpcode1, ScalarOpcode2,ScalarOpcode3;

//------------------------------------------------------------------------
//daizy-chaining the above buses
//------------------------------------------------------------------------

always@(posedge sclk)
  begin
        if(srst)
          begin
             q0_instruct_start <= 1'b0;
             q1_instruct_start <= 1'b0;
             q2_instruct_start <= 1'b0;
```

```
             q0_instruct_stall <= 1'b0;
             q1_instruct_stall <= 1'b0;
             q2_instruct_stall <= 1'b0;
             q0_instruct <= 21'b0;
             q1_instruct <= 21'b0;
             q2_instruct <= 21'b0;
             q0_gpr_wr_addr<= 7'b0;
             q1_gpr_wr_addr<= 7'b0;
             q2_gpr_wr_addr<= 7'b0;
             q0_gpr_rd_addr<= 7'b0;
             q1_gpr_rd_addr<= 7'b0;
             q2_gpr_rd_addr<= 7'b0;
             q0_gpr_mre <= 1'b0;
             q1_gpr_mre <= 1'b0;
             q2_gpr_mre <= 1'b0;
             q0_gpr_mwe <= 1'b0;
             q1_gpr_mwe <= 1'b0;
             q2_gpr_mwe <= 1'b0;
             q0_gpr_we <= 1'b0;
             q1_gpr_we <= 1'b0;
             q2_gpr_we <= 1'b0;
             q0_gpr_phase_mux <= 2'b0;
             q1_gpr_phase_mux <= 2'b0;
             q2_gpr_phase_mux <= 2'b0;
             q0_gpr_input_mux <= 2'b0;
```

```
             q1_gpr_input_mux <= 2'b0;
             q2_gpr_input_mux <= 2'b0;
          end // if (srst)
        else
          begin
             q0_instruct_start <= sq_sp_instruct_start;
             q1_instruct_start <= q0_instruct_start;
             q2_instruct_start <= q1_instruct_start;
             q0_instruct_stall <= sq_sp_stall;
             q1_instruct_stall <= q0_instruct_stall;
             q2_instruct_stall <= q1_instruct_stall;
             q0_instruct <= sq_sp_instruct;
             q1_instruct <= q0_instruct;
             q2_instruct <= q1_instruct;
             q0_gpr_wr_addr<= sq_sp_wr_addr;
             q1_gpr_wr_addr<= q0_gpr_wr_addr;
             q2_gpr_wr_addr<= q1_gpr_wr_addr;
             q0_gpr_rd_addr<= sq_sp_gpr_rd_addr;
             q1_gpr_rd_addr<= q0_gpr_rd_addr;
             q2_gpr_rd_addr<= q1_gpr_rd_addr;
             q0_gpr_mre <= sq_sp_mem_rd_ena;
             q1_gpr_mre <= q0_gpr_mre;
             q2_gpr_mre <= q1_gpr_mre;
             q0_gpr_mwe <= sq_sp_mem_wr_ena;
             q1_gpr_mwe <= q0_gpr_mwe;
```

```
              q2_gpr_mwe <= q1_gpr_mwe;
              q0_gpr_we <= sq_sp_wr_ena;
              q1_gpr_we <= q0_gpr_we;
              q2_gpr_we <= q1_gpr_we;
              q0_gpr_phase_mux <= sq_sp_gpr_phase_mux;
              q1_gpr_phase_mux <= q0_gpr_phase_mux;
              q2_gpr_phase_mux <= q1_gpr_phase_mux;
              q0_gpr_input_mux <= sq_sp_gpr_input_mux;
              q1_gpr_input_mux <= q0_gpr_input_mux;
              q2_gpr_input_mux <= q1_gpr_input_mux;
              q0_gpr_cmask    <=  sq_sp_channel_mask;
              q1_gpr_cmask    <=  q0_gpr_cmask;
              q2_gpr_cmask    <=  q1_gpr_cmask;
              q0_tp_gpr_cmask  <= tp_sp_gpr_cmask;
              q1_tp_gpr_cmask  <= q0_tp_gpr_cmask;
              q2_tp_gpr_cmask  <= q1_tp_gpr_cmask;
              q0_tp_gpr_dst  <= tp_sp_gpr_dst;
              q1_tp_gpr_dst  <= q0_tp_gpr_dst;
              q2_tp_gpr_dst  <= q1_tp_gpr_dst;
              q0_tp_data_valid  <= tp_sp_data_valid;
              q1_tp_data_valid  <= q0_tp_data_valid;
              q2_tp_data_valid  <= q1_tp_data_valid;
           end // else: !if(srst)
      end // always@ (posedge sclk)

```

```

   //------------------------------------------------------------------------------------------------------------------------

   //Muxing logic to select from data comming from the Interpolators(in reality more than just interpolated data....there can be
   //also faceness and XY data), AutoCount data and Vertex Indices comming from the staging registers.
   //Each MACC unit has its own mux logic since the controls are phased out by one cycle from one MACC to the other.
   //------------------------------------------------------------------------------------------------------------------------
   //muxing logic for the inputs of the first MACC
   always @(/*AUTOSENSE*/iAutoCount or iInterpolated or iVertexIndices
              or sq_sp_gpr_input_mux)
     begin
          case(sq_sp_gpr_input_mux)
            2'b00: InputData0 = iAutoCount ;
            2'b01: InputData0 = iInterpolated ;
            2'b10: InputData0 = iVertexIndices ;
            default: InputData0 = iInterpolated;
          endcase // case(sq_sp_gpr_input_mux)
     end

   //muxing logic for the inputs of the second MACC
   always @(/*AUTOSENSE*/iAutoCount or iInterpolated or iVertexIndices
              or q0_gpr_input_mux)
     begin
```

```
          case(q0_gpr_input_mux)
            2'b00: InputData1 = iAutoCount ;
            2'b01: InputData1 = iInterpolated ;
            2'b10: InputData1 = iVertexIndices ;
            default: InputData1 = iInterpolated;
          endcase // case(q0_gpr_input_mux)
     end

   //muxing logic for the inputs of the third MACC
   always @(/*AUTOSENSE*/iAutoCount or iInterpolated or iVertexIndices
              or q1_gpr_input_mux)
     begin
          case(q1_gpr_input_mux)
            2'b00: InputData2 = iAutoCount ;
            2'b01: InputData2 = iInterpolated ;
            2'b10: InputData2 = iVertexIndices ;
            default: InputData2 = iInterpolated;
          endcase // case(q1_gpr_input_mux)
     end

   //muxing logic for the inputs of the fourth MACC
   always @(/*AUTOSENSE*/iAutoCount or iInterpolated or iVertexIndices
              or q2_gpr_input_mux)
     begin
          case(q2_gpr_input_mux)
```

```
            2'b00: InputData3 = iAutoCount ;
            2'b01: InputData3 = iInterpolated ;
            2'b10: InputData3 = iVertexIndices ;
            default: InputData3 = iInterpolated;
          endcase // case(q2_gpr_input_mux)
     end

   //-----------------------------------------------------------------------------------------------------------------
   //muxing logic for the scalar input into the scalar unit
   //there's one scalar unit for each vector unit in the shader pipe
   //alpha channel of SrcC is used as input argument into the scalar unit only in cases of an coissued instruction.
   //---------------------------------------------------------------------------------------------------------------
   always @(/*AUTOSENSE*/ScalarInput0 or ScalarInput1 or ScalarInput2
              or ScalarInput3 or sq_sp_gpr_phase_mux)
     begin
          case(sq_sp_gpr_phase_mux)
            2'b00: ScalarInput = ScalarInput0;
            2'b01: ScalarInput = ScalarInput1;
            2'b10: ScalarInput = ScalarInput2;
            2'b11: ScalarInput = ScalarInput3;
            default: ScalarInput = 2'bxx;
          endcase // case(sq_sp_gpr_phase_mux)
     end
```

```
  always @(/*AUTOSENSE*/ScalarOpcode0 or ScalarOpcode1
           or ScalarOpcode2 or ScalarOpcode3 or sq_sp_gpr_phase_mux)
    begin
       case(sq_sp_gpr_phase_mux)
         2'b00: ScalarOpcode = ScalarOpcode0;
         2'b01: ScalarOpcode = ScalarOpcode1;
         2'b10: ScalarOpcode = ScalarOpcode2;
         2'b11: ScalarOpcode = ScalarOpcode3;
         default: ScalarOpcode = 2'bxx;
       endcase // case(sq_sp_gpr_phase_mux)
    end

  //module scalar_lut( iAG_ME_OPCODE, iAG_ME_IN_A, iAG_ME_IN_B, iAG_ME_IN_C,
  //        iAG_ME_ABS_A, iAG_ME_ABS_B, iAG_ME_ABS_C, iAG_ME_A_NEGATE, iAG_ME_B_NEGATE
  // , iAG_ME_C_NEGATE, oME_RESULT, sclk );

  //--------------------------------------------------------------------------------
  //Scalar Unit instantiation
  //--------------------------------------------------------------------------------
  scalar_lut          uscalar(ScalarOpcode,ScalarInput,          ScalarInput,ScalarInput, 1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,ScalarResult,sclk);
```

```
  //-------------------------------------------------------------------------------------------------------------------------------------
  //Instantiation of all four MACC units that create a Vector Unit
  //------------------------------------------------------------------------------------------------------------------------------------
  macc_gpr umacc_gpr0(.oVectorOutput(VectorResult0),.oScalarInput(ScalarInput0),.oScalarOpcode(ScalarOpcode0),.oRegData(RegData0),.oexport_dst(sq_sp_exp_dst),

.sq_sp_instruct(sq_sp_instruct),.sq_sp_instruct_start(sq_sp_instruct_start),.sq_sp_gpr_rd_addr(sq_sp_gpr_rd_addr),

.sq_sp_gpr_wr_addr(sq_sp_wr_addr),.sq_sp_wr_ena(sq_sp_wr_ena),.sq_sp_mem_rd_ena(sq_sp_mem_rd_ena),.sq_sp_mem_wr_ena(sq_sp_mem_wr_ena),
                      .sq_sp_gpr_cmask(sq_sp_channel_mask),

.sq_sp_gpr_phase_mux(sq_sp_gpr_phase_mux),.iInterpolated(InputData0),.sq_sp_constant(sq_sp_constant),
                      .iScalarData(ScalarData),.tp_sp_data(tp_sp_data),
                      .tp_sp_gpr_dst(tp_sp_gpr_dst),
.tp_sp_gpr_cmask(tp_sp_gpr_cmask),.tp_sp_data_valid(tp_sp_data_valid),
                      .sclk(sclk), .srst(srst));

  macc_gpr umacc_gpr1(.oVectorOutput(VectorResult1),.oScalarInput(ScalarInput1),.oScalarOpcode(ScalarOpcode1),.oRegData(RegData1),.sq_sp_instruct(q0_instruct),.sq_sp_instruct_start(q0_instruct_start),.sq_sp_gpr_rd_addr(q0_gpr_rd_addr),

.sq_sp_gpr_wr_addr(q0_gpr_wr_addr),.sq_sp_wr_ena(q0_gpr_we),.sq_sp_mem_rd_ena(q0_gpr_mre),.sq_sp_mem_wr_ena(q0_gpr_mwe),
                      .sq_sp_gpr_cmask(q0_gpr_cmask),
```

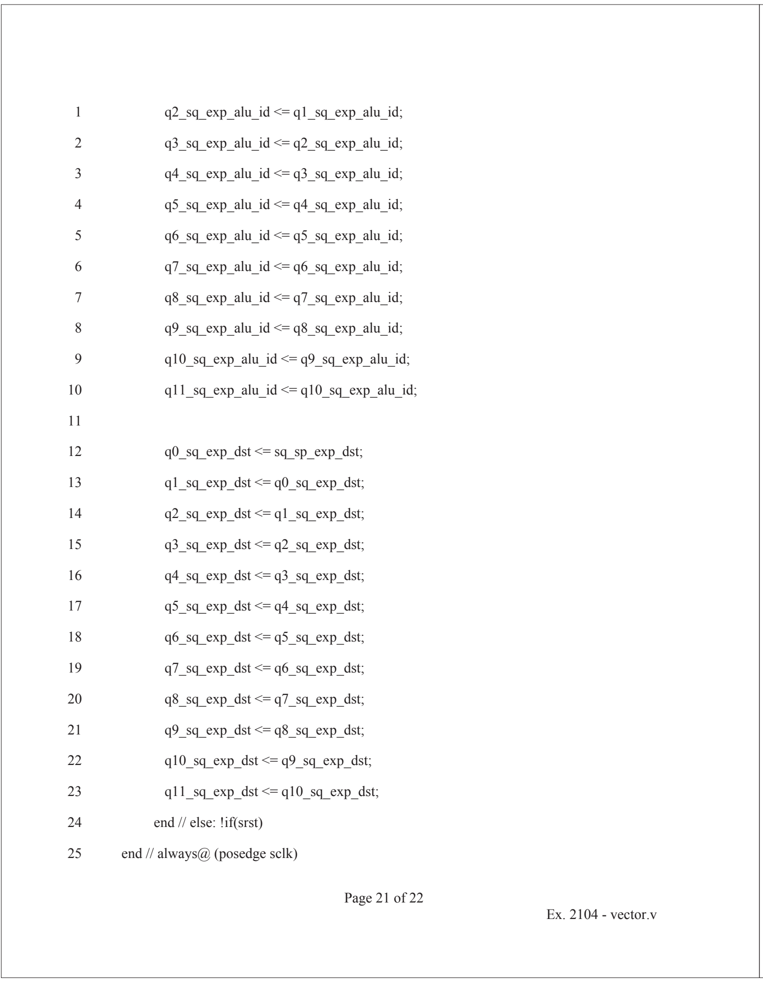
```
.sq_sp_gpr_phase_mux(q0_gpr_phase_mux),.iInterpolated(InputData1),.sq_sp_constant(sq_sp_constant),
                      .iScalarData(q0_ScalarData),.tp_sp_data(tp_sp_data),
                      .tp_sp_gpr_dst(q0_tp_gpr_dst),
.tp_sp_gpr_cmask(q0_tp_gpr_cmask),.tp_sp_data_valid(q0_tp_data_valid),
                      .sclk(sclk), .srst(srst));

  macc_gpr umacc_gpr2(.oVectorOutput(VectorResult2),.oScalarInput(ScalarInput2),.oScalarOpcode(ScalarOpcode2),.oRegData(RegData2),.sq_sp_instruct(q1_instruct),.sq_sp_instruct_start(q1_instruct_start),.sq_sp_gpr_rd_addr(q1_gpr_rd_addr),

.sq_sp_gpr_wr_addr(q1_gpr_wr_addr),.sq_sp_wr_ena(q1_gpr_we),.sq_sp_mem_rd_ena(q1_gpr_mre),.sq_sp_mem_wr_ena(q1_gpr_mwe),
                      .sq_sp_gpr_cmask(q1_gpr_cmask),

.sq_sp_gpr_phase_mux(q1_gpr_phase_mux),.iInterpolated(InputData2),.sq_sp_constant(sq_sp_constant),
                      .iScalarData(q1_ScalarData),.tp_sp_data(tp_sp_data),
                      .tp_sp_gpr_dst(q1_tp_gpr_dst),
.tp_sp_gpr_cmask(q1_tp_gpr_cmask),.tp_sp_data_valid(q1_tp_data_valid),
                      .sclk(sclk), .srst(srst));

  macc_gpr umacc_gpr3(.oVectorOutput(VectorResult3),.oScalarInput(ScalarInput3),.oScalarOpcode(ScalarOpcode3),.oRegData(RegData3),.sq_sp_instruct(q2_instruct),.sq_sp_instruct_start(q2_instruct_start),.sq_sp_gpr_rd_addr(q2_gpr_rd_addr),

.sq_sp_gpr_wr_addr(q2_gpr_wr_addr),.sq_sp_wr_ena(q2_gpr_we),.sq_sp_mem_rd_ena(q2_gpr_mre),.sq_sp_mem_wr_ena(q2_gpr_mwe),
                      .sq_sp_gpr_cmask(q2_gpr_cmask),
```

```
.sq_sp_gpr_phase_mux(q2_gpr_phase_mux),.iInterpolated(InputData3),.sq_sp_constant(sq_sp_constant),
                      .iScalarData(q2_ScalarData),.tp_sp_data(tp_sp_data),.sclk(sclk),
                      .tp_sp_gpr_dst(q2_tp_gpr_dst),
.tp_sp_gpr_cmask(q2_tp_gpr_cmask),.tp_sp_data_valid(q2_tp_data_valid),
                      .srst(srst));
  //---------------------------------------------------------------------------------------------------------------------------------------
  //Muxing the gpr vector results into one final vector result conrolled by the phase_mux signal or a registered version of it
  //----------------------------------------------------------------------------------------------------------------------------------------

  always @(/*AUTOSENSE*/VectorResult0 or VectorResult1
           or VectorResult2 or VectorResult3 or sq_sp_gpr_phase_mux)
    begin
       case(sq_sp_gpr_phase_mux)
         2'b00: osp_sx_data = VectorResult0;
         2'b01: osp_sx_data = VectorResult1;
         2'b10: osp_sx_data = VectorResult2;
         2'b11: osp_sx_data = VectorResult3;
       endcase // case(sq_sp_gpr_phase_mux)
    end

  assign sp_sx_data = osp_sx_data;
  //---------------------------------------------------------------------------------------------------------------
-----------------------
  //Muxing the gpr outputs (texture data) into one 128 bit bus going out to the texture unit
  //---------------------------------------------------------------------------------------------------------------
------------------------
  always @(/*AUTOSENSE*/RegData0 or RegData1 or RegData2 or RegData3
            or sq_sp_gpr_phase_mux)
    begin
       case(sq_sp_gpr_phase_mux)
         2'b00: osp_tp_data = RegData0[95:0];
         2'b01: osp_tp_data = RegData1[95:0];
         2'b10: osp_tp_data = RegData2[95:0];
         2'b11: osp_tp_data = RegData3[95:0];
       endcase // case(sq_sp_gpr_phase_mux)
    end

  assign sp_tp_data = osp_tp_data;

  //-----------------------------------------------------------------------------------------------
  //replicate the scalar result into all four channels so that it can be used as an 128-bit input
  //into the vector unit argument selection logic as PS (Previous Scalar Result)
  //-------------------------------------------------------------------------------------------
  assign ScalarData = {ScalarResult,ScalarResult,ScalarResult,ScalarResult};

  always @(posedge sclk)
    begin
```

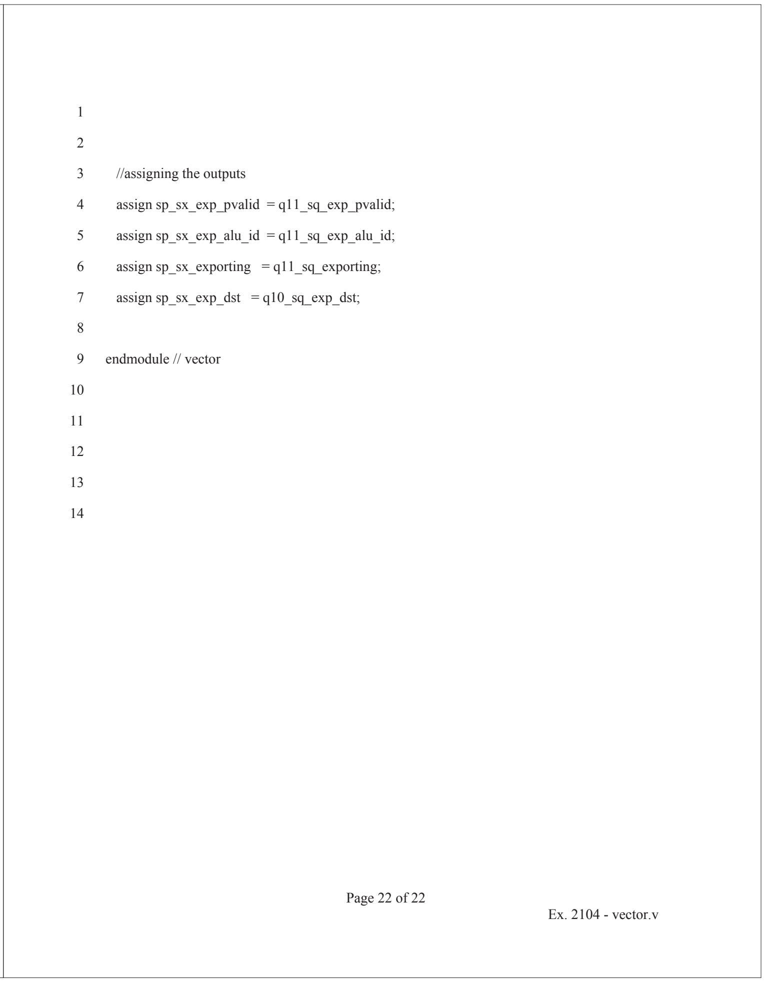
```
         q0_ScalarData <= ScalarData;
         q1_ScalarData <= q0_ScalarData;
         q2_ScalarData <= q1_ScalarData;
     end

   //simple pipelining to delay the sq to sx control signals (via sp)
   reg                                                                                              [3:0]
          q0_sq_exp_pvalid,q1_sq_exp_pvalid,q2_sq_exp_pvalid,q3_sq_exp_pvalid,q4_sq_exp_
pvalid;
   reg                                                                                              [3:0]
          q5_sq_exp_pvalid,q6_sq_exp_pvalid,q7_sq_exp_pvalid,q8_sq_exp_pvalid,q9_sq_exp_
pvalid;
   reg [3:0]       q10_sq_exp_pvalid,q11_sq_exp_pvalid;


   reg                                                                                              [0:0]
          q0_sq_exporting,q1_sq_exporting,q2_sq_exporting,q3_sq_exporting,q4_sq_exporting;
   reg                                                                                              [0:0]
          q5_sq_exporting,q6_sq_exporting,q7_sq_exporting,q8_sq_exporting,q9_sq_exporting;
   reg [0:0]       q10_sq_exporting,q11_sq_exporting;


   reg                                                                                              [0:0]
          q0_sq_exp_alu_id,q1_sq_exp_alu_id,q2_sq_exp_alu_id,q3_sq_exp_alu_id,q4_sq_exp_
alu_id;
   reg                                                                                              [0:0]
          q5_sq_exp_alu_id,q6_sq_exp_alu_id,q7_sq_exp_alu_id,q8_sq_exp_alu_id,q9_sq_exp_
alu_id;
   reg [0:0]       q10_sq_exp_alu_id,q11_sq_exp_alu_id;


   reg [5:0]       q0_sq_exp_dst,q1_sq_exp_dst,q2_sq_exp_dst,q3_sq_exp_dst,q4_sq_exp_dst;
   reg [5:0]       q5_sq_exp_dst,q6_sq_exp_dst,q7_sq_exp_dst,q8_sq_exp_dst,q9_sq_exp_dst;
```

```
  reg [5:0]       q10_sq_exp_dst,q11_sq_exp_dst;


  always@(posedge sclk)
    begin
       if(srst)
         begin
            q0_sq_exporting <= 2'b00;
            q1_sq_exporting <= 2'b00;
            q2_sq_exporting <= 2'b00;
            q3_sq_exporting <= 2'b00;
            q4_sq_exporting <= 2'b00;
            q5_sq_exporting <= 2'b00;
            q6_sq_exporting <= 2'b00;
            q7_sq_exporting <= 2'b00;
            q8_sq_exporting <= 2'b00;
            q9_sq_exporting <= 2'b00;
            q10_sq_exporting <= 2'b00;
            q11_sq_exporting <= 2'b00;
         end
       else
         begin
            q0_sq_exp_pvalid <= sq_sp_exp_pvalid;
            q1_sq_exp_pvalid <= q0_sq_exp_pvalid;
            q2_sq_exp_pvalid <= q1_sq_exp_pvalid;
```

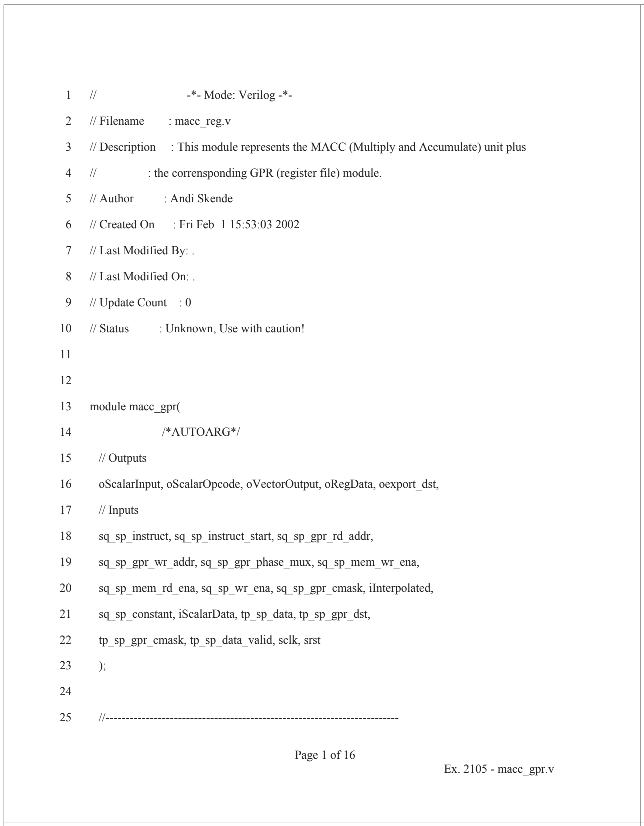
```
            q3_sq_exp_pvalid <= q2_sq_exp_pvalid;
            q4_sq_exp_pvalid <= q3_sq_exp_pvalid;
            q5_sq_exp_pvalid <= q4_sq_exp_pvalid;
            q6_sq_exp_pvalid <= q5_sq_exp_pvalid;
            q7_sq_exp_pvalid <= q6_sq_exp_pvalid;
            q8_sq_exp_pvalid <= q7_sq_exp_pvalid;
            q9_sq_exp_pvalid <= q8_sq_exp_pvalid;
            q10_sq_exp_pvalid <= q9_sq_exp_pvalid;
            q11_sq_exp_pvalid <= q10_sq_exp_pvalid;

            q0_sq_exporting <= sq_sp_exporting;
            q1_sq_exporting <= q0_sq_exporting;
            q2_sq_exporting <= q1_sq_exporting;
            q3_sq_exporting <= q2_sq_exporting;
            q4_sq_exporting <= q3_sq_exporting;
            q5_sq_exporting <= q4_sq_exporting;
            q6_sq_exporting <= q5_sq_exporting;
            q7_sq_exporting <= q6_sq_exporting;
            q8_sq_exporting <= q7_sq_exporting;
            q9_sq_exporting <= q8_sq_exporting;
            q10_sq_exporting <= q9_sq_exporting;
            q11_sq_exporting <= q10_sq_exporting;

            q0_sq_exp_alu_id <= sq_sp_exp_alu_id;
            q1_sq_exp_alu_id <= q0_sq_exp_alu_id;
```



```
        q2_sq_exp_alu_id <= q1_sq_exp_alu_id;
        q3_sq_exp_alu_id <= q2_sq_exp_alu_id;
        q4_sq_exp_alu_id <= q3_sq_exp_alu_id;
        q5_sq_exp_alu_id <= q4_sq_exp_alu_id;
        q6_sq_exp_alu_id <= q5_sq_exp_alu_id;
        q7_sq_exp_alu_id <= q6_sq_exp_alu_id;
        q8_sq_exp_alu_id <= q7_sq_exp_alu_id;
        q9_sq_exp_alu_id <= q8_sq_exp_alu_id;
        q10_sq_exp_alu_id <= q9_sq_exp_alu_id;
        q11_sq_exp_alu_id <= q10_sq_exp_alu_id;

        q0_sq_exp_dst <= sq_sp_exp_dst;
        q1_sq_exp_dst <= q0_sq_exp_dst;
        q2_sq_exp_dst <= q1_sq_exp_dst;
        q3_sq_exp_dst <= q2_sq_exp_dst;
        q4_sq_exp_dst <= q3_sq_exp_dst;
        q5_sq_exp_dst <= q4_sq_exp_dst;
        q6_sq_exp_dst <= q5_sq_exp_dst;
        q7_sq_exp_dst <= q6_sq_exp_dst;
        q8_sq_exp_dst <= q7_sq_exp_dst;
        q9_sq_exp_dst <= q8_sq_exp_dst;
        q10_sq_exp_dst <= q9_sq_exp_dst;
        q11_sq_exp_dst <= q10_sq_exp_dst;
      end // else: !if(srst)
  end // always@ (posedge sclk)
```



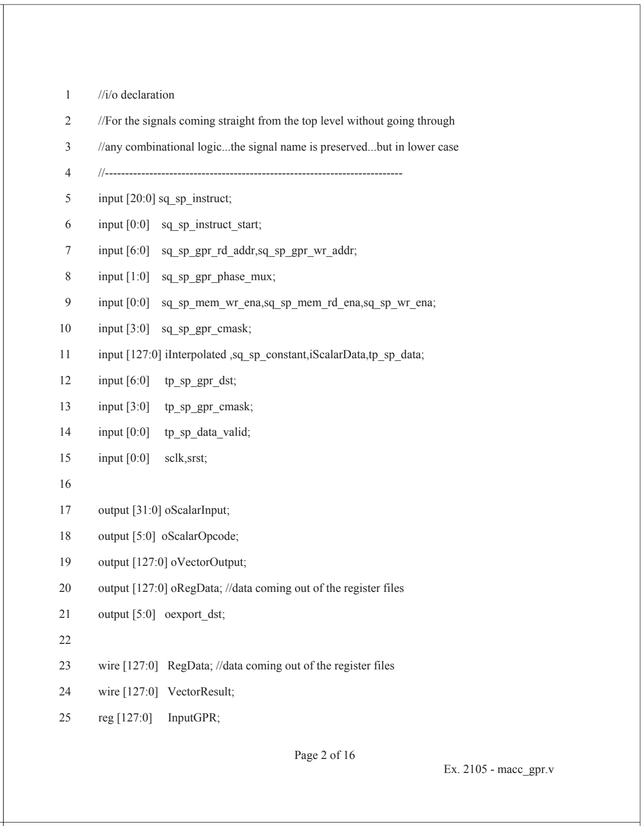
```


  //assigning the outputs
  assign sp_sx_exp_pvalid  = q11_sq_exp_pvalid;
  assign sp_sx_exp_alu_id  = q11_sq_exp_alu_id;
  assign sp_sx_exporting   = q11_sq_exporting;
  assign sp_sx_exp_dst   = q10_sq_exp_dst;

endmodule // vector
```



```
//                              -*- Mode: Verilog -*-
// Filename        : macc_reg.v
// Description     : This module represents the MACC (Multiply and Accumulate) unit plus
//                 : the corrensponding GPR (register file) module.
// Author          : Andi Skende
// Created On      : Fri Feb  1 15:53:03 2002
// Last Modified By: .
// Last Modified On: .
// Update Count    : 0
// Status          : Unknown, Use with caution!


module macc_gpr(
                        /*AUTOARG*/
   // Outputs
   oScalarInput, oScalarOpcode, oVectorOutput, oRegData, oexport_dst,
   // Inputs
   sq_sp_instruct, sq_sp_instruct_start, sq_sp_gpr_rd_addr,
   sq_sp_gpr_wr_addr, sq_sp_gpr_phase_mux, sq_sp_mem_wr_ena,
   sq_sp_mem_rd_ena, sq_sp_wr_ena, sq_sp_gpr_cmask, iInterpolated,
   sq_sp_constant, iScalarData, tp_sp_data, tp_sp_gpr_dst,
   tp_sp_gpr_cmask, tp_sp_data_valid, sclk, srst
   );

   //--------------------------------------------------------------------------
```

Page 1 of 16

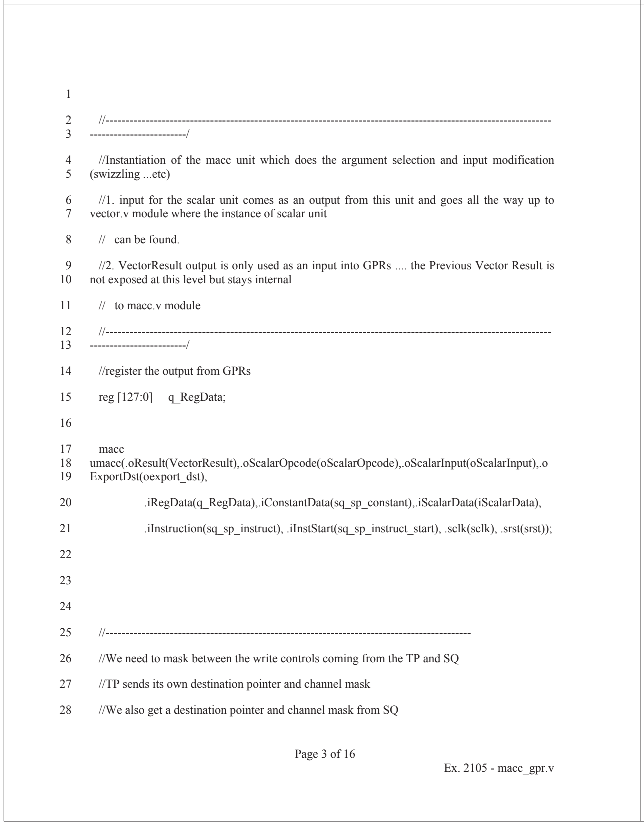
Ex. 2105 - macc_gpr.v



```
//i/o declaration
//For the signals coming straight from the top level without going through
//any combinational logic...the signal name is preserved...but in lower case
//--------------------------------------------------------------------------
input [20:0] sq_sp_instruct;
input [0:0]    sq_sp_instruct_start;
input [6:0]    sq_sp_gpr_rd_addr,sq_sp_gpr_wr_addr;
input [1:0]    sq_sp_gpr_phase_mux;
input [0:0]    sq_sp_mem_wr_ena,sq_sp_mem_rd_ena,sq_sp_wr_ena;
input [3:0]    sq_sp_gpr_cmask;
input [127:0] iInterpolated ,sq_sp_constant,iScalarData,tp_sp_data;
input [6:0]     tp_sp_gpr_dst;
input [3:0]     tp_sp_gpr_cmask;
input [0:0]     tp_sp_data_valid;
input [0:0]     sclk,srst;

output [31:0] oScalarInput;
output [5:0]  oScalarOpcode;
output [127:0] oVectorOutput;
output [127:0] oRegData; //data coming out of the register files
output [5:0]   oexport_dst;

wire [127:0]   RegData; //data coming out of the register files
wire [127:0]   VectorResult;
reg [127:0]     InputGPR;
```

Page 2 of 16

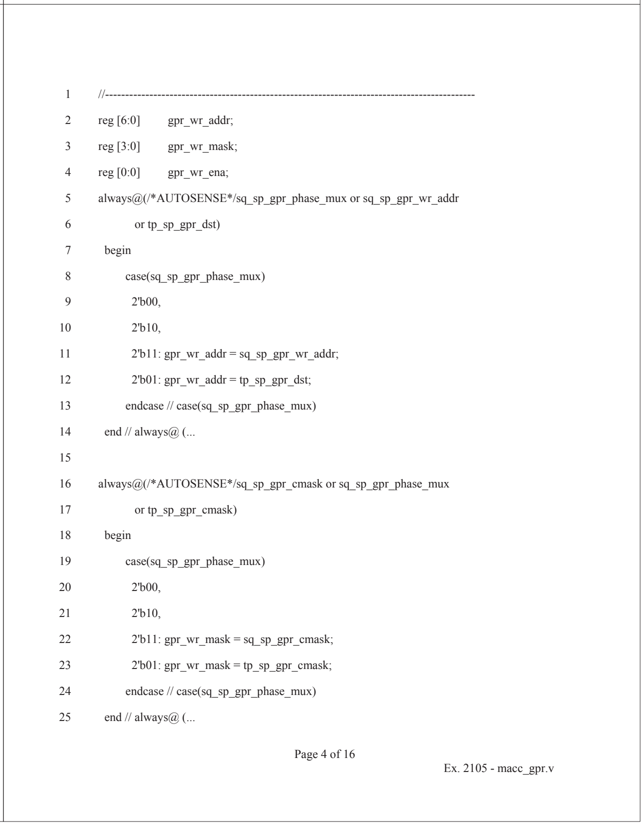
Ex. 2105 - macc_gpr.v



```

   //------------------------------------------------------------------------------------------------------------------------------------/

   //Instantiation of the macc unit which does the argument selection and input modification (swizzling ...etc)

   //1. input for the scalar unit comes as an output from this unit and goes all the way up to vector.v module where the instance of scalar unit
   //   can be found.
   //2. VectorResult output is only used as an input into GPRs .... the Previous Vector Result is not exposed at this level but stays internal
   //   to macc.v module
   //------------------------------------------------------------------------------------------------------------------------------------/
   //register the output from GPRs
   reg [127:0]     q_RegData;

   macc umacc(.oResult(VectorResult),.oScalarOpcode(oScalarOpcode),.oScalarInput(oScalarInput),.oExportDst(oexport_dst),
                    .iRegData(q_RegData),.iConstantData(sq_sp_constant),.iScalarData(iScalarData),
                    .iInstruction(sq_sp_instruct), .iInstStart(sq_sp_instruct_start), .sclk(sclk), .srst(srst));



   //------------------------------------------------------------------------------------------
   //We need to mask between the write controls coming from the TP and SQ
   //TP sends its own destination pointer and channel mask
   //We also get a destination pointer and channel mask from SQ
```

Page 3 of 16

Ex. 2105 - macc_gpr.v



```
//------------------------------------------------------------------------------------------
reg [6:0]         gpr_wr_addr;
reg [3:0]         gpr_wr_mask;
reg [0:0]         gpr_wr_ena;
always@(/*AUTOSENSE*/sq_sp_gpr_phase_mux or sq_sp_gpr_wr_addr
            or tp_sp_gpr_dst)
  begin
        case(sq_sp_gpr_phase_mux)
          2'b00,
          2'b10,
          2'b11: gpr_wr_addr = sq_sp_gpr_wr_addr;
          2'b01: gpr_wr_addr = tp_sp_gpr_dst;
        endcase // case(sq_sp_gpr_phase_mux)
  end // always@ (...

always@(/*AUTOSENSE*/sq_sp_gpr_cmask or sq_sp_gpr_phase_mux
            or tp_sp_gpr_cmask)
  begin
        case(sq_sp_gpr_phase_mux)
          2'b00,
          2'b10,
          2'b11: gpr_wr_mask = sq_sp_gpr_cmask;
          2'b01: gpr_wr_mask = tp_sp_gpr_cmask;
        endcase // case(sq_sp_gpr_phase_mux)
  end // always@ (...
```

Page 4 of 16

Ex. 2105 - macc_gpr.v

ATI 2105
LG v. ATI
IPR2015-00326

```


  wire tp_sp_wr_ena;
  assign tp_sp_wr_ena = tp_sp_data_valid;

  always@(/*AUTOSENSE*/sq_sp_gpr_phase_mux or sq_sp_wr_ena
           or tp_sp_wr_ena)
    begin
       case(sq_sp_gpr_phase_mux)
         2'b00,
         2'b10,
         2'b11: gpr_wr_ena = sq_sp_wr_ena;
         2'b01: gpr_wr_ena = tp_sp_wr_ena;
       endcase // case(sq_sp_gpr_phase_mux)
    end // always@ (...


  //-------------------------------------------------------------------------------------------------------------
------------/
  //The phase mux controlling the write input port into GPRs (register file write port)
  //-------------------------------------------------------------------------------------------------------------
------------/
  always@(/*AUTOSENSE*/VectorResult or iInterpolated or iScalarData
           or sq_sp_gpr_phase_mux or tp_sp_data)
    begin
       case(sq_sp_gpr_phase_mux)
```

```
         2'b00: InputGPR = iInterpolated;
         2'b01: InputGPR = tp_sp_data;
         2'b10: InputGPR = VectorResult;
         2'b11: InputGPR = iScalarData;
         default: InputGPR = iInterpolated;
       endcase // case(sq_sp_gpr_phase_mux)
    end // always@ (...


  //----------------------------------------------------------------------------
  //Behavioral model of a 128x128 Register File used to emulate GPRs
  //----------------------------------------------------------------------------
  // dum_mem_p2 #(7,128,128) udum_mem(.iRCLK(sclk),
  //                                  .iWCLK(sclk),
  //                     .iMER(sq_sp_mem_rd_ena),
  //          .iMEW(sq_sp_mem_wr_ena),
  //   .iWEN(sq_sp_wr_ena),
  //  .iRADR(sq_sp_gpr_rd_addr),
  // .iWADR(sq_sp_gpr_wr_addr),
  // .iD(InputGPR),
  // .oQ(RegData)
  // );

  wire [127:0] subword_write_mask;

```

```
  //since the virage behavioral model does not give a better mask/pin ration than 1:1
  //i am generating the 32-bit-subword mask in the following fashion
  assign         subword_write_mask[31:0] = {32{gpr_wr_mask[0]}};
  assign         subword_write_mask[63:32] = {32{gpr_wr_mask[1]}};
  assign         subword_write_mask[95:64] = {32{gpr_wr_mask[2]}};
  assign         subword_write_mask[127:96] = {32{gpr_wr_mask[3]}};


`ifdef USE_BEHAVE_MEM
  rfsd2_128x128cm1sw8_core ugpr_mem(.QB(RegData),
                                    .ADRA_buf(gpr_wr_addr),
                                    .DA_buf(InputGPR),
                                    .WEMA_buf(subword_write_mask),
                                    .WEA_buf(gpr_wr_ena),
                                    .MEA_buf(gpr_wr_ena),
                                    .CLKA(sclk),
                                    .BISTEA(1'b0),
                                    .ADRB_buf(sq_sp_gpr_rd_addr),
                                    .OEB_buf(1'b1),
                                    .MEB_buf(sq_sp_mem_rd_ena),
                                    .CLKB(sclk),
                                    .BISTEB(1'b0),
                                    .AWTB(1'b0)
                                    );
`else // !`ifdef USE_BEHAVE_MEM
```

```
  rfsd2_128x128cm2sw1 ugpr_mem
    (/*VRGIO rfsd2_128x128cm2sw1 InputGPR RegData gpr_wr_addr sq_sp_gpr_rd_addr gpr_wr_ena sq_sp_gpr_rd_ena null*/
     // READ INTERFACE
     .CLKB(iSCLK), // Read Clock
     .OEB(sq_sp_gpr_rd_ena), // Output enable
     .MEB(vdd), // Read enable
     .ADRB0(sq_sp_gpr_rd_addr[0]),                    .ADRB1(sq_sp_gpr_rd_addr[1]), .ADRB2(sq_sp_gpr_rd_addr[2]), .ADRB3(sq_sp_gpr_rd_addr[3]),  // Read Address
     .ADRB4(sq_sp_gpr_rd_addr[4]),                    .ADRB5(sq_sp_gpr_rd_addr[5]), .ADRB6(sq_sp_gpr_rd_addr[6]),  // Read Address
     .QB0(RegData[0]),  .QB1(RegData[1]),  .QB2(RegData[2]),  .QB3(RegData[3]),   // Read Data
     .QB4(RegData[4]),  .QB5(RegData[5]),  .QB6(RegData[6]),  .QB7(RegData[7]),   // Read Data
     .QB8(RegData[8]),  .QB9(RegData[9]),  .QB10(RegData[10]),  .QB11(RegData[11]),   // Read Data
     .QB12(RegData[12]), .QB13(RegData[13]), .QB14(RegData[14]), .QB15(RegData[15]),  // Read Data
     .QB16(RegData[16]), .QB17(RegData[17]), .QB18(RegData[18]), .QB19(RegData[19]),  // Read Data
     .QB20(RegData[20]), .QB21(RegData[21]), .QB22(RegData[22]), .QB23(RegData[23]),  // Read Data
     .QB24(RegData[24]), .QB25(RegData[25]), .QB26(RegData[26]), .QB27(RegData[27]),  // Read Data
     .QB28(RegData[28]), .QB29(RegData[29]), .QB30(RegData[30]), .QB31(RegData[31]),  // Read Data
     .QB32(RegData[32]), .QB33(RegData[33]), .QB34(RegData[34]), .QB35(RegData[35]),  // Read Data
     .QB36(RegData[36]), .QB37(RegData[37]), .QB38(RegData[38]), .QB39(RegData[39]),  // Read Data
    .QB40(RegData[40]), .QB41(RegData[41]), .QB42(RegData[42]), .QB43(RegData[43]), // Read Data
    .QB44(RegData[44]), .QB45(RegData[45]), .QB46(RegData[46]), .QB47(RegData[47]), // Read Data
    .QB48(RegData[48]), .QB49(RegData[49]), .QB50(RegData[50]), .QB51(RegData[51]), // Read Data
    .QB52(RegData[52]), .QB53(RegData[53]), .QB54(RegData[54]), .QB55(RegData[55]), // Read Data
    .QB56(RegData[56]), .QB57(RegData[57]), .QB58(RegData[58]), .QB59(RegData[59]), // Read Data
    .QB60(RegData[60]), .QB61(RegData[61]), .QB62(RegData[62]), .QB63(RegData[63]), // Read Data
    .QB64(RegData[64]), .QB65(RegData[65]), .QB66(RegData[66]), .QB67(RegData[67]), // Read Data
    .QB68(RegData[68]), .QB69(RegData[69]), .QB70(RegData[70]), .QB71(RegData[71]), // Read Data
    .QB72(RegData[72]), .QB73(RegData[73]), .QB74(RegData[74]), .QB75(RegData[75]), // Read Data
    .QB76(RegData[76]), .QB77(RegData[77]), .QB78(RegData[78]), .QB79(RegData[79]), // Read Data
    .QB80(RegData[80]), .QB81(RegData[81]), .QB82(RegData[82]), .QB83(RegData[83]), // Read Data
    .QB84(RegData[84]), .QB85(RegData[85]), .QB86(RegData[86]), .QB87(RegData[87]), // Read Data
    .QB88(RegData[88]), .QB89(RegData[89]), .QB90(RegData[90]), .QB91(RegData[91]), // Read Data
    .QB92(RegData[92]), .QB93(RegData[93]), .QB94(RegData[94]), .QB95(RegData[95]), // Read Data
    .QB96(RegData[96]), .QB97(RegData[97]), .QB98(RegData[98]), .QB99(RegData[99]), // Read Data
    .QB100(RegData[100]), .QB101(RegData[101]), .QB102(RegData[102]), .QB103(RegData[103]), // Read Data
```

```
    .QB104(RegData[104]), .QB105(RegData[105]), .QB106(RegData[106]), .QB107(RegData[107]), // Read Data
    .QB108(RegData[108]), .QB109(RegData[109]), .QB110(RegData[110]), .QB111(RegData[111]), // Read Data
    .QB112(RegData[112]), .QB113(RegData[113]), .QB114(RegData[114]), .QB115(RegData[115]), // Read Data
    .QB116(RegData[116]), .QB117(RegData[117]), .QB118(RegData[118]), .QB119(RegData[119]), // Read Data
    .QB120(RegData[120]), .QB121(RegData[121]), .QB122(RegData[122]), .QB123(RegData[123]), // Read Data
    .QB124(RegData[124]), .QB125(RegData[125]), .QB126(RegData[126]), .QB127(RegData[127]), // Read Data
    // WRITE INTERFACE
    .CLKA(iSCLK), // Write Clock
    .WEA(gpr_wr_ena), // Write enable
    .MEA(vdd), // Memory enable
    .ADRA0(gpr_wr_addr[0]), .ADRA1(gpr_wr_addr[1]), .ADRA2(gpr_wr_addr[2]), .ADRA3(gpr_wr_addr[3]), // Write Address
    .ADRA4(gpr_wr_addr[4]), .ADRA5(gpr_wr_addr[5]), .ADRA6(gpr_wr_addr[6]), // Write Address
    .DA0(InputGPR[0]), .DA1(InputGPR[1]), .DA2(InputGPR[2]), .DA3(InputGPR[3]), // Write Data
    .DA4(InputGPR[4]), .DA5(InputGPR[5]), .DA6(InputGPR[6]), .DA7(InputGPR[7]), // Write Data
    .DA8(InputGPR[8]), .DA9(InputGPR[9]), .DA10(InputGPR[10]), .DA11(InputGPR[11]), // Write Data
    .DA12(InputGPR[12]), .DA13(InputGPR[13]), .DA14(InputGPR[14]), .DA15(InputGPR[15]), // Write Data
    .DA16(InputGPR[16]), .DA17(InputGPR[17]), .DA18(InputGPR[18]), .DA19(InputGPR[19]), // Write Data
    .DA20(InputGPR[20]), .DA21(InputGPR[21]), .DA22(InputGPR[22]), .DA23(InputGPR[23]), // Write Data
```

```
    .DA24(InputGPR[24]), .DA25(InputGPR[25]), .DA26(InputGPR[26]), .DA27(InputGPR[27]), // Write Data
    .DA28(InputGPR[28]), .DA29(InputGPR[29]), .DA30(InputGPR[30]), .DA31(InputGPR[31]), // Write Data
    .DA32(InputGPR[32]), .DA33(InputGPR[33]), .DA34(InputGPR[34]), .DA35(InputGPR[35]), // Write Data
    .DA36(InputGPR[36]), .DA37(InputGPR[37]), .DA38(InputGPR[38]), .DA39(InputGPR[39]), // Write Data
    .DA40(InputGPR[40]), .DA41(InputGPR[41]), .DA42(InputGPR[42]), .DA43(InputGPR[43]), // Write Data
    .DA44(InputGPR[44]), .DA45(InputGPR[45]), .DA46(InputGPR[46]), .DA47(InputGPR[47]), // Write Data
    .DA48(InputGPR[48]), .DA49(InputGPR[49]), .DA50(InputGPR[50]), .DA51(InputGPR[51]), // Write Data
    .DA52(InputGPR[52]), .DA53(InputGPR[53]), .DA54(InputGPR[54]), .DA55(InputGPR[55]), // Write Data
    .DA56(InputGPR[56]), .DA57(InputGPR[57]), .DA58(InputGPR[58]), .DA59(InputGPR[59]), // Write Data
    .DA60(InputGPR[60]), .DA61(InputGPR[61]), .DA62(InputGPR[62]), .DA63(InputGPR[63]), // Write Data
    .DA64(InputGPR[64]), .DA65(InputGPR[65]), .DA66(InputGPR[66]), .DA67(InputGPR[67]), // Write Data
    .DA68(InputGPR[68]), .DA69(InputGPR[69]), .DA70(InputGPR[70]), .DA71(InputGPR[71]), // Write Data
    .DA72(InputGPR[72]), .DA73(InputGPR[73]), .DA74(InputGPR[74]), .DA75(InputGPR[75]), // Write Data
    .DA76(InputGPR[76]), .DA77(InputGPR[77]), .DA78(InputGPR[78]), .DA79(InputGPR[79]), // Write Data
    .DA80(InputGPR[80]), .DA81(InputGPR[81]), .DA82(InputGPR[82]), .DA83(InputGPR[83]), // Write Data
    .DA84(InputGPR[84]), .DA85(InputGPR[85]), .DA86(InputGPR[86]), .DA87(InputGPR[87]), // Write Data
```

```
    .DA88(InputGPR[88]), .DA89(InputGPR[89]), .DA90(InputGPR[90]), .DA91(InputGPR[91]), // Write Data
    .DA92(InputGPR[92]), .DA93(InputGPR[93]), .DA94(InputGPR[94]), .DA95(InputGPR[95]), // Write Data
    .DA96(InputGPR[96]), .DA97(InputGPR[97]), .DA98(InputGPR[98]), .DA99(InputGPR[99]), // Write Data
    .DA100(InputGPR[100]), .DA101(InputGPR[101]), .DA102(InputGPR[102]), .DA103(InputGPR[103]), // Write Data
    .DA104(InputGPR[104]), .DA105(InputGPR[105]), .DA106(InputGPR[106]), .DA107(InputGPR[107]), // Write Data
    .DA108(InputGPR[108]), .DA109(InputGPR[109]), .DA110(InputGPR[110]), .DA111(InputGPR[111]), // Write Data
    .DA112(InputGPR[112]), .DA113(InputGPR[113]), .DA114(InputGPR[114]), .DA115(InputGPR[115]), // Write Data
    .DA116(InputGPR[116]), .DA117(InputGPR[117]), .DA118(InputGPR[118]), .DA119(InputGPR[119]), // Write Data
    .DA120(InputGPR[120]), .DA121(InputGPR[121]), .DA122(InputGPR[122]), .DA123(InputGPR[123]), // Write Data
    .DA124(InputGPR[124]), .DA125(InputGPR[125]), .DA126(InputGPR[126]), .DA127(InputGPR[127]), // Write Data
    // WRITE TEST SIGNALS
    .BISTEA(vss),
    .TWEA(vss),
    .TMEA(vss),
    .TADRA0(gpr_wr_addr[0]), .TADRA1(gpr_wr_addr[1]), .TADRA2(gpr_wr_addr[2]), .TADRA3(gpr_wr_addr[3]), // Write Test Address
    .TADRA4(gpr_wr_addr[4]), .TADRA5(gpr_wr_addr[5]), .TADRA6(gpr_wr_addr[6]), // Write Test Address
    .TDA0(InputGPR[0]), .TDA1(InputGPR[1]), .TDA2(InputGPR[2]), .TDA3(InputGPR[3]), // Write Test Data
    .TDA4(InputGPR[4]), .TDA5(InputGPR[5]), .TDA6(InputGPR[6]), .TDA7(InputGPR[7]), // Write Test Data
```

```
        .TDA8(InputGPR[8]),                .TDA9(InputGPR[9]),                .TDA10(InputGPR[10]),
   .TDA11(InputGPR[11]),  // Write Test Data
        .TDA12(InputGPR[12]),              .TDA13(InputGPR[13]),              .TDA14(InputGPR[14]),
   .TDA15(InputGPR[15]),  // Write Test Data
        .TDA16(InputGPR[16]),              .TDA17(InputGPR[17]),              .TDA18(InputGPR[18]),
   .TDA19(InputGPR[19]),  // Write Test Data
        .TDA20(InputGPR[20]),              .TDA21(InputGPR[21]),              .TDA22(InputGPR[22]),
   .TDA23(InputGPR[23]),  // Write Test Data
        .TDA24(InputGPR[24]),              .TDA25(InputGPR[25]),              .TDA26(InputGPR[26]),
   .TDA27(InputGPR[27]),  // Write Test Data
        .TDA28(InputGPR[28]),              .TDA29(InputGPR[29]),              .TDA30(InputGPR[30]),
   .TDA31(InputGPR[31]),  // Write Test Data
        .TDA32(InputGPR[32]),              .TDA33(InputGPR[33]),              .TDA34(InputGPR[34]),
   .TDA35(InputGPR[35]),  // Write Test Data
        .TDA36(InputGPR[36]),              .TDA37(InputGPR[37]),              .TDA38(InputGPR[38]),
   .TDA39(InputGPR[39]),  // Write Test Data
        .TDA40(InputGPR[40]),              .TDA41(InputGPR[41]),              .TDA42(InputGPR[42]),
   .TDA43(InputGPR[43]),  // Write Test Data
        .TDA44(InputGPR[44]),              .TDA45(InputGPR[45]),              .TDA46(InputGPR[46]),
   .TDA47(InputGPR[47]),  // Write Test Data
        .TDA48(InputGPR[48]),              .TDA49(InputGPR[49]),              .TDA50(InputGPR[50]),
   .TDA51(InputGPR[51]),  // Write Test Data
        .TDA52(InputGPR[52]),              .TDA53(InputGPR[53]),              .TDA54(InputGPR[54]),
   .TDA55(InputGPR[55]),  // Write Test Data
        .TDA56(InputGPR[56]),              .TDA57(InputGPR[57]),              .TDA58(InputGPR[58]),
   .TDA59(InputGPR[59]),  // Write Test Data
        .TDA60(InputGPR[60]),              .TDA61(InputGPR[61]),              .TDA62(InputGPR[62]),
   .TDA63(InputGPR[63]),  // Write Test Data
        .TDA64(InputGPR[64]),              .TDA65(InputGPR[65]),              .TDA66(InputGPR[66]),
   .TDA67(InputGPR[67]),  // Write Test Data
        .TDA68(InputGPR[68]),              .TDA69(InputGPR[69]),              .TDA70(InputGPR[70]),
   .TDA71(InputGPR[71]),  // Write Test Data
```

```
        .TDA72(InputGPR[72]),              .TDA73(InputGPR[73]),              .TDA74(InputGPR[74]),
   .TDA75(InputGPR[75]),  // Write Test Data
        .TDA76(InputGPR[76]),              .TDA77(InputGPR[77]),              .TDA78(InputGPR[78]),
   .TDA79(InputGPR[79]),  // Write Test Data
        .TDA80(InputGPR[80]),              .TDA81(InputGPR[81]),              .TDA82(InputGPR[82]),
   .TDA83(InputGPR[83]),  // Write Test Data
        .TDA84(InputGPR[84]),              .TDA85(InputGPR[85]),              .TDA86(InputGPR[86]),
   .TDA87(InputGPR[87]),  // Write Test Data
        .TDA88(InputGPR[88]),              .TDA89(InputGPR[89]),              .TDA90(InputGPR[90]),
   .TDA91(InputGPR[91]),  // Write Test Data
        .TDA92(InputGPR[92]),              .TDA93(InputGPR[93]),              .TDA94(InputGPR[94]),
   .TDA95(InputGPR[95]),  // Write Test Data
        .TDA96(InputGPR[96]),              .TDA97(InputGPR[97]),              .TDA98(InputGPR[98]),
   .TDA99(InputGPR[99]),  // Write Test Data
        .TDA100(InputGPR[100]),        .TDA101(InputGPR[101]),        .TDA102(InputGPR[102]),
   .TDA103(InputGPR[103]),  // Write Test Data
        .TDA104(InputGPR[104]),        .TDA105(InputGPR[105]),        .TDA106(InputGPR[106]),
   .TDA107(InputGPR[107]),  // Write Test Data
        .TDA108(InputGPR[108]),        .TDA109(InputGPR[109]),        .TDA110(InputGPR[110]),
   .TDA111(InputGPR[111]),  // Write Test Data
        .TDA112(InputGPR[112]),        .TDA113(InputGPR[113]),        .TDA114(InputGPR[114]),
   .TDA115(InputGPR[115]),  // Write Test Data
        .TDA116(InputGPR[116]),        .TDA117(InputGPR[117]),        .TDA118(InputGPR[118]),
   .TDA119(InputGPR[119]),  // Write Test Data
        .TDA120(InputGPR[120]),        .TDA121(InputGPR[121]),        .TDA122(InputGPR[122]),
   .TDA123(InputGPR[123]),  // Write Test Data
        .TDA124(InputGPR[124]),        .TDA125(InputGPR[125]),        .TDA126(InputGPR[126]),
   .TDA127(InputGPR[127]),  // Write Test Data

        //READ TEST SIGNALS

        .BISTEB(vss),

        .TOEB(vss),

        .TMEB(vss),
```

```
        .TADRB0(sq_sp_gpr_rd_addr[0]),                                .TADRB1(sq_sp_gpr_rd_addr[1]),
   .TADRB2(sq_sp_gpr_rd_addr[2]), .TADRB3(sq_sp_gpr_rd_addr[3]),  // Read Test Address
        .TADRB4(sq_sp_gpr_rd_addr[4]),                                .TADRB5(sq_sp_gpr_rd_addr[5]),
   .TADRB6(sq_sp_gpr_rd_addr[6]),  // Read Test Address

        .AWTB(vss)

        );

   `endif // !`ifdef USE_BEHAVE_MEM




     always @(posedge sclk)

       begin

              q_RegData <= RegData;

       end


     assign      oVectorOutput = VectorResult;

     assign      oRegData = q_RegData;


   endmodule // macc_gpr
```

```
//                              -*- Mode: Verilog -*-
// Filename        : export_buffer.v
// Description     : The module implements the control for export buffers as well as the quad interface between SC and SX blocks
// Author          : Andi Skende
// Created On      : Thu Apr  4 19:13:50 2002
// Last Modified By: .
// Last Modified On: .
// Update Count    : 0
// Status          : Unknown, Use with caution!

`timescale 1ns / 1ps
module export_control(/*AUTOARG*/
   // Outputs
   sx_sc_quad_rtr, sx_sq_exp_count_rdy, sx_sq_exp_pos_avail,
   sx_sq_exp_buf_avail, sx_rb_quad_x, sx_rb_quad_y, sx_rb_quad_mask,
   sx_rb_quad_type, sx_rb_quad_pixel, sx_rb_quad_index,
   sx_rb0_quad_send, sx_rb1_quad_send, sx_rb2_quad_send,
   sx_rb3_quad_send, sx_rb0_color_data, sx_rb1_color_data,
   sx_rb2_color_data, sx_rb3_color_data, sx_rb0_color_send,
   sx_rb1_color_send, sx_rb2_color_send, sx_rb3_color_send,
   sx_rb0_index_rtr, sx_rb1_index_rtr, sx_rb2_index_rtr,
   sx_rb3_index_rtr, sx_pa_send, sx_pa_data,
   // Inputs
   sc_sx_quad_x, sc_sx_quad_y, sc_sx_quad_mask, sc_sx_quad_tilex,
   sc_sx_quad_tiley, sc_sx_quad_send, sclk, srst, sp0_sx_exp_pvalid,
```

```
sp1_sx_exp_pvalid, sp0_sx_exp_alu_id, sp1_sx_exp_alu_id,
sp0_sx_exporting, sp1_sx_exporting, sp0_sx_exp_dest,
sp1_sx_exp_dest, sp0_sx_data0, sp0_sx_data1, sp0_sx_data2,
sp0_sx_data3, sp1_sx_data0, sp1_sx_data1, sp1_sx_data2,
sp1_sx_data3, sq_sx_exp_type, sq_sx_exp_number, sq_sx_exp_pix,
sq_sx_exp_state, sq_sx_exp_id, sq_sx_exp_valid, sq_sx_free_done,
sq_sx_free_id, rb0_sx_quad_rtr, rb1_sx_quad_rtr, rb2_sx_quad_rtr,
rb3_sx_quad_rtr, rb0_sx_color_rtr, rb1_sx_color_rtr,
rb2_sx_color_rtr, rb3_sx_color_rtr, rb0_sx_index, rb1_sx_index,
rb2_sx_index, rb3_sx_index, rb0_sx_index_send, rb1_sx_index_send,
rb2_sx_index_send, rb3_sx_index_send, rb0_sx_index_op,
rb1_sx_index_op, rb2_sx_index_op, rb3_sx_index_op,
rbbm_sx_soft_reset, rbbm_we, rbbm_wd, rbbm_a, rbbm_be, rbbm_re,
rbb_rs_in, rbb_rd_in, pa_sx_req, pa_sx_sp_id, pa_sx_offset,
pa_sx_aux, pa_sx_last
);


input [1:0]             sc_sx_quad_x;
input [1:0]             sc_sx_quad_y;
input [31:0]    sc_sx_quad_mask;
input [1:0]             sc_sx_quad_tilex;
input                   sc_sx_quad_tiley;
input                   sc_sx_quad_send;
output                  sx_sc_quad_rtr;
```

```
   input                sclk,srst;


   //ANDI ???? revisit this interface
   input [3:0]          sp0_sx_exp_pvalid, sp1_sx_exp_pvalid; //pixel valid mask
   input                sp0_sx_exp_alu_id,  sp1_sx_exp_alu_id;  //isn't one of these signals redundand ??? ANDI
   input [0:0]          sp0_sx_exporting,  sp1_sx_exporting;  //isn't one of these signals redundand ??? ANDI
   input [5:0]          sp0_sx_exp_dest, sp1_sx_exp_dest;

   //-------------------------------------------------------------------------------//
   input [127:0]        sp0_sx_data0,sp0_sx_data1,sp0_sx_data2,sp0_sx_data3;
   input [127:0]        sp1_sx_data0,sp1_sx_data1,sp1_sx_data2,sp1_sx_data3;

   //-----------------------------------------------------------------------------/
   //SQ to SX Control Bus
   //-----------------------------------------------------------------------------/
   input [1:0]          sq_sx_exp_type;
   input [1:0]          sq_sx_exp_number;
   input [0:0]          sq_sx_exp_pix;
   input [2:0]          sq_sx_exp_state;  //ANDI ....just a hack

   assign               sq_sx_exp_state = 3'b0;

   input [0:0]          sq_sx_exp_id;
```

```
input [0:0]             sq_sx_exp_valid; //valid cycle
input [0:0]             sq_sx_free_done;
input [0:0]             sq_sx_free_id;


//-------------------------------------------------------------------------------/
//Export Buffer status control interface
//-------------------------------------------------------------------------------/
output                  sx_sq_exp_count_rdy;
output                  sx_sq_exp_pos_avail;
output [6:0]    sx_sq_exp_buf_avail;

//---------------------------------------------------------------------------------//
//SX-RB interface
//---------------------------------------------------------------------------------//

//---------------------//
//--Quad Interface-----//
//---------------------//
output [1:0]    sx_rb_quad_x;
output [1:0]    sx_rb_quad_y;
output [31:0]           sx_rb_quad_mask;
output [0:0]    sx_rb_quad_type;
output [3:0]    sx_rb_quad_pixel;
output [7:0]    sx_rb_quad_index;
```

```
    output [0:0]     sx_rb0_quad_send,                sx_rb1_quad_send,                sx_rb2_quad_send,
sx_rb3_quad_send;
    input [0:0]                  rb0_sx_quad_rtr, rb1_sx_quad_rtr, rb2_sx_quad_rtr, rb3_sx_quad_rtr;



    //--------------------------------------//
    //--Pixel Color Data interface----------//
    //--------------------------------------//
    output [127:0] sx_rb0_color_data,sx_rb1_color_data,sx_rb2_color_data,sx_rb3_color_data;
    output [0:0]   sx_rb0_color_send,sx_rb1_color_send,sx_rb2_color_send,sx_rb3_color_send;
    input [0:0]      rb0_sx_color_rtr,rb1_sx_color_rtr,rb2_sx_color_rtr,rb3_sx_color_rtr;



    //--------------------------------------------//
    //SX to RB quad index interface-------------//
    //------------------------------------------//
    input [7:0]      rb0_sx_index,rb1_sx_index,rb2_sx_index,rb3_sx_index;
    input [0:0]
rb0_sx_index_send,rb1_sx_index_send,rb2_sx_index_send,rb3_sx_index_send;
    input [0:0]      rb0_sx_index_op, rb1_sx_index_op, rb2_sx_index_op, rb3_sx_index_op;
    output [0:0]   sx_rb0_index_rtr, sx_rb1_index_rtr,sx_rb2_index_rtr,sx_rb3_index_rtr;


     //--------------------------------------------------------------------------------/
    //CP/rbbm Interface for Real Time data and snooping state registers
```

```
//--------------------------------------------------------------------------------/
//There's no rtr nrtrtr signals ...tie them high or low at the top level
input             rbbm_sx_soft_reset;
input             rbbm_we;
input [31:0]  rbbm_wd;
input [14:0]  rbbm_a;
input [3:0]     rbbm_be;
input             rbbm_re;

input         rbb_rs_in;
input [31:0]  rbb_rd_in;

reg               state_soft_reset;
reg               state_we;
reg [31:0]      state_wd;
reg [14:0]      state_a;
reg [3:0]       state_be;
reg               state_re;

reg               state_rs_in;
reg [31:0]      state_rd_in;

reg               state_rs_out;
reg [31:0]      state_rd_out;

```

```

    //-----------------------------------------------------------------------------//
    //PA(Primitive Assembly) - SX position export interface
    //-----------------------------------------------------------------------------//
    input                       pa_sx_req;
    input                       pa_sx_sp_id;
    input [1:0]                 pa_sx_offset;  //used to be pa_sx_export_phase
    input                       pa_sx_aux; //used to be pa_sx_2ndbuff
    input                       pa_sx_last;
    output                      sx_pa_send;
    output [127:0]              sx_pa_data;


    reg [127:0]                 q0_sp0_data0, q0_sp0_data1,q0_sp0_data2,q0_sp0_data3;
    reg [127:0]                 q0_sp1_data0, q0_sp1_data1,q0_sp1_data2,q0_sp1_data3;
    reg [127:0]                 q1_sp0_data0, q1_sp0_data1,q1_sp0_data2,q1_sp0_data3;
    reg [127:0]                 q1_sp1_data0, q1_sp1_data1,q1_sp1_data2,q1_sp1_data3;

    reg [31:0]                  state_import_export0,state_import_export1;
    reg [31:0]                  state_import_export2,state_import_export3;
    reg [31:0]                  state_import_export4,state_import_export5;
    reg [31:0]                  state_import_export6,state_import_export7;


    //-----------------------------------------------------------------------------
```

```
// SX - SC interface signals
//-----------------------------------------------------------------------------
wire                 sx_sc_rtr;
reg                            q_sx_sc_rtr;

//-----------------------------------------------------------------------------//
// Register all the inputs
//-----------------------------------------------------------------------------//

always @(posedge sclk)
  begin
        if(srst)
          begin

              q_sx_sc_rtr <= 1'b0;
              state_soft_reset <= 1'b0;
              state_we <= 1'b0;
              state_wd <= 32'b0;
              state_a <= 15'b0;
              state_be <= 4'b0;
              state_re <= 1'b0;
              state_rs_in <= 1'b0;
              state_rd_in <= 32'b0;
              state_rs_out <= 1'b0;
              state_rd_out <= 32'b0;
```

```
      end
    else
      begin
        q0_sp0_data0 <= sp0_sx_data0;
        q0_sp0_data1 <= sp0_sx_data1;
        q0_sp0_data2 <= sp0_sx_data2;
        q0_sp0_data3 <= sp0_sx_data3;
        q0_sp1_data0 <= sp1_sx_data0;
        q0_sp1_data1 <= sp1_sx_data1;
        q0_sp1_data2 <= sp1_sx_data2;
        q0_sp1_data3 <= sp1_sx_data3;
        q1_sp0_data0 <= q0_sp0_data0;
        q1_sp0_data1 <= q0_sp0_data1;
        q1_sp0_data2 <= q0_sp0_data2;
        q1_sp0_data3 <= q0_sp0_data3;
        q1_sp1_data0 <= q0_sp1_data0;
        q1_sp1_data1 <= q0_sp1_data1;
        q1_sp1_data2 <= q0_sp1_data2;
        q1_sp1_data3 <= q0_sp1_data3;
        q_sx_sc_rtr <= sx_sc_rtr;
        state_soft_reset <= rbbm_sx_soft_reset;
        state_we <= rbbm_we;
        state_wd <= rbbm_wd;
        state_a <= rbbm_a;
        state_be <= rbbm_be;
```

```
        state_re <= rbbm_re;
        state_rs_in <= rbb_rs_in;
        state_rd_in <= rbb_rd_in;
      end
  end // always @ (posedge sclk)




// The fifo below is two-quad wide to allow for double reads on the read side
// For this reason, I write the fifo only every other valid cycle from the SC
// Quad control/info bus from SC

//counts two valid quads before it writes into the skid buffer
reg [0:0] quad_valid_count;

always @(posedge sclk)
  begin
    if(srst)
      begin
        quad_valid_count <= 2'b0;
      end
    else
      begin
        if(sc_sx_quad_send)
```

```
          quad_valid_count <= quad_valid_count+1;
      end
  end // always @ (posedge sclk)


reg [1:0]          q_quad_x;
reg [1:0]          q_quad_y;
reg [31:0]         q_quad_mask;
reg [1:0]          q_quad_tilex;
reg                q_quad_tiley;
reg                q_quad_send;

always @(posedge sclk)
  begin
    if(srst)
      begin
        q_quad_x <= 2'b0;
        q_quad_y <= 2'b0;
        q_quad_mask <= 32'b0;
        q_quad_tilex <= 2'b0;
        q_quad_tiley <= 1'b0;
        q_quad_send <= 1'b0;
      end
    else
      begin
```

```
        q_quad_x <= sc_sx_quad_x;
        q_quad_y <= sc_sx_quad_y;
        q_quad_mask <= sc_sx_quad_mask;
        q_quad_tilex <= sc_sx_quad_tilex;
        q_quad_tiley <= sc_sx_quad_tiley;
        q_quad_send <= sc_sx_quad_send;
      end
  end // always @ (posedge sclk)


wire [38:0] quad_data;
wire [38:0] q_quad_data;
wire [77:0] double_quad;
wire [0:0]  quad_write;

//meaning of each bit going into quad buffer
//sc_sx_quad1_x     1:0
//sc_sx_quad1_y     3:2
//sc_sx_quad1_mask  35:4
//sc_sx_quad1_tilex 37:36
//sc_sx_quad1_tiley 38
//sc_sx_quad0_x     40:39
//sc_sx_quad0_y     42:41
//sc_sx_quad0_mask  74:43
//sc_sx_quad0_tilex 76:75
```

```
    //sc_sx_quad0_tiley 77

    assign     quad_data = {sc_sx_quad_tiley,sc_sx_quad_tilex,sc_sx_quad_mask,sc_sx_quad_y,sc_sx_quad_x};
    assign     q_quad_data = {q_quad_tiley,q_quad_tilex,q_quad_mask,q_quad_y,q_quad_x};
    assign     double_quad = {q_quad_data,quad_data};

    //second valid quad of the pair coming from SC
    assign     quad_write = sc_sx_quad_send & quad_valid_count;

    assign     sx_sc_quad_rtr = q_sx_sc_rtr; //do I need another register stage here ????ANDI

    reg        read_quad_cmd_rtr;
    wire [77:0] read_data ;


    //--------------------------------------------------------------
    //skid buffer at SC-SX quad control/info data
    //--------------------------------------------------------------
    skid_buff_top #(78,128) sc_sx_quad_skid(
                            .write_rts(quad_write), //sc is ready to send
                            .write_rtr(sx_sc_rtr),  //sx is ready to receive ...fifo not yet full
                            .write_data(double_quad), // quad info coming from sc

```

```
                            .read_rts(read_rts),     //quad command data available in the buffer
                            .read_rtr(read_quad_cmd_rtr),
                            .read_data(read_data),

                            .clk(sclk),        //clock and reset
                            .reset(srst)

                            );

    //--------------------------------------------------------------
    //SC-SX interface skid buffer internal read
    //--------------------------------------------------------------

    wire [0:0]  valid_export;
    wire [0:0]  valid_pixel_export;
    reg [0:0]   q0_valid_export,q1_valid_export ; //we have valid exports coming out of SPs
    reg [0:0] q0_valid_pixel_export, q1_valid_pixel_export,q2_valid_pixel_export,q3_valid_pixel_export;
    reg [0:0]   valid_data;

    //if there's quad data in the buffer and valid data coming from SP then pop the fifo.
    always @(posedge sclk)
      begin
         if(read_rts & valid_pixel_export)
           begin
```

```
              read_quad_cmd_rtr <= 1'b1;
              valid_data <= q0_valid_export;
           end
         else
           begin
              read_quad_cmd_rtr <= 1'b0;
              valid_data <= 1'b0;
           end
      end // always @ (posedge sclk)

    //This valid_export bit is passed through with data into the ALPHA/RGBA test logic or pipeline once (depending on ALPHA/RGBA logic latency).
    //The output of it is than send to the read side of the skid buffer sitting at the quad interface between the SX and SC...
    //so the respective quad info is read from the quad buffer.

    assign valid_export = |sp0_sx_exporting; //exporing pixel or vertices vs. no exports

    //hack for now ...ANDI
    //we have an export from shader pipe and this is a pixel export
    reg [0:0] q_exp_pix_alu0;
    assign valid_pixel_export = valid_export & q_exp_pix_alu0;

    //delay the valid by one cycle to account for ALPHA/RGBA data processing logic latency
    always @(posedge sclk)
      begin
```

```
         if(srst)
           begin
              q0_valid_export <= 1'b0;
              q1_valid_export <= 1'b0;
           end
         else
           begin
              q0_valid_export <= valid_export;
              q1_valid_export <= q0_valid_export;
           end
      end // always @ (posedge sclk)


    //delay the valid by one cycle to account for ALPHA/RGBA data processing logic latency
    always @(posedge sclk)
      begin
         if(srst)
           begin
              q0_valid_pixel_export <= 1'b0;
              q1_valid_pixel_export <= 1'b0;
              q2_valid_pixel_export <= 1'b0;
              q3_valid_pixel_export <= 1'b0;
           end
         else
           begin
```

```
                q0_valid_pixel_export <= valid_pixel_export;
                q1_valid_pixel_export <= q0_valid_pixel_export;
                q2_valid_pixel_export <= q1_valid_pixel_export;
                q3_valid_pixel_export <= q2_valid_pixel_export;
             end
       end // always @ (posedge sclk)


   //Next....the quad data just read from the "quad buffer" goes into
   //the "detailed quad buffer" after the 32-bit mask has been modified
   //accordingly based on the result of the ALPHA/RGBA >1.0/0.0 test logic
   //The tile_x and tile_y informantion as been stripped out of the quad packet.
   //A two bit id has been attached based on which RB the quad belongs to.
   //Also, a index on where the quad resides into Export Buffer has been attached to
   //the "detailed quad packet" written into the "detailed quad buffer".

   wire [6:0] export_index;
   wire [1:0] rb_id0,rb_id1; //used to identify which rb the quad belongs to based on the state and math on tile_x and tile_y data

   wire [1:0] tile_x0; //first quad tile coordinates
   wire [0:0] tile_y0;
   wire [1:0] tile_x1; //second quad tile coordinates
   wire [0:0] tile_y1;
   wire [3:0] rb_sys_config;

```

```
   //four RBs present in the current configuration ...hadwired for the time-being
   assign     rb_sys_config = 4'b1111;

   reg [77:0] quad_data_read ;

   //latching the output of the quad command skid buffer before going into the quad command fifo
   //with the modified data...modified mask and rb id added.
   always @(posedge sclk)
     begin
           if(read_quad_cmd_rtr)
             quad_data_read <= read_data;
     end



   reg [6:0] q_export_index;
   always @(posedge sclk)
     begin
           q_export_index <= export_index;
     end


   //the meaning of each bit going into quad buffer
   //sc_sx_quad1_x      1:0
   //sc_sx_quad1_y      3:2
```

```
   //sc_sx_quad1_mask  35:4
   //sc_sx_quad1_tilex 37:36
   //sc_sx_quad1_tiley 38
   //sc_sx_quad0_x      40:39
   //sc_sx_quad0_y      42:41
   //sc_sx_quad0_mask  74:43
   //sc_sx_quad0_tilex 76:75
   //sc_sx_quad0_tiley 77


   assign     tile_x1 = quad_data_read[37:36];
   assign     tile_y1 = quad_data_read[38];

   assign     tile_x0 = quad_data_read[76:75];
   assign     tile_y0 = quad_data_read[77];


   //this module is used to define depending on RB/MC config, wich RB unit the current quad (tile_x, tile_y) belongs to
   //there's a one cycle delay through the rb id calculation logic as well as the ALPHA test logic
   mod_rb_id                    urb_id0(.block_id(rb_id0),                    .tile_x(tile_x0), .tile_y(tile_y0),.config(rb_sys_config),.sclk(sclk)); //quad 0
   mod_rb_id                    urb_id1(.block_id(rb_id1),                    .tile_x(tile_x1), .tile_y(tile_y1),.config(rb_sys_config),.sclk(sclk)); //quad 1


```

```
   //delay the read_data bus by one cycle to compensate for the above delay
   reg [77:0] q_quad_data_read;
   always @(posedge sclk)
     begin
           q_quad_data_read <= quad_data_read;
     end

   //modify the mask based on the ALPHA/RGBA test outcome and pvalid bits.


   //remember ....the quad info out of "quad buffer" is in the following format
   //{iquad_tiley,iquad_tilex,iquad_mask,iquad_y,iquad_x};
   //the new format going into "detailed quad buffer" will be {rb_id,export_index,iquad_mask,iquad_y,iquad_x}

   wire [91:0]  detailed_quad_data;
   wire [7:0]     export_index0, export_index1;

   assign          export_index0 = {1'b0,q_export_index[6:0]};
   assign       export_index1 = {1'b1,q_export_index[6:0]};

   //for now there's no need to register the quad_data_read by one cycle
   //the latency through mode_rb_id module is less than one cycle
   assign          detailed_quad_data    =    {rb_id0,    export_index0,quad_data_read[74:43], quad_data_read[42:41],quad_data_read[40:39],
                                    rb_id1,                     export_index1,quad_data_read[35:4], quad_data_read[3:2],quad_data_read[1:0]};

  //writting the detailed quad data into the "detailed quad buffer"

  wire [91:0]    quad_data_rb ;

  wire [0:0]     detailed_quad_valid;

  wire [0:0]     read_detailed_quad;



  //-----------------------------------------------------------------------------------------------------------------

  //simple fifo carrying quad data (eventually sent to RBs) with the correct mask values (post ALPHA test)

  //------------------------------------------------------------------------------------------------------------------

  skid_buff_top #(92,96,0) sx_rb_detailed_quad_data(

                                                   .write_rts(q1_valid_pixel_export),     //modified quad command data is being tranfered over from the quad buffer to

                                                   //to the index fifo

                                                   //.write_rtr(sx_sc_rtr),

                                                   .write_data(detailed_quad_data),  //  quad info with the modified 32 bit mask

                                                   .read_rts(detailed_quad_valid),        //this quad data is eventually send out to RBs once all exports are in

                                                   .read_rtr(read_detailed_quad),

                                                   .read_data(quad_data_rb),

```

```
                                                                      .clk(sclk),           //clock and reset

                                                                      .reset(srst)




                                                                      );

  reg [0:0]      exp_data_in_buff0,exp_data_in_buff1;    //this bit signifies that all the export data for that quad is in the Export Buffers

  wire [0:0]     rb_ready;

  wire [0:0]     sx_rb_quad_send;

  wire [1:0]     sx_rb_id;

  wire [0:0]     quad_data_ready;



  //----------------------------------------------------------------------------------------------------

  //SX-RB "index" interface

  //reading the "detailed quad buffer" and sending the data to RBs on the quad broadcast bus.

  //----------------------------------------------------------------------------------------------------

  assign     rb_ready  =  rb0_sx_quad_rtr  &  rb0_sx_quad_rtr  &  rb0_sx_quad_rtr  &  rb0_sx_quad_rtr;



  //send the index quad data to the RBs only when all the RBs are ready to receive it

  //and  there's outstanding quads sitting in the export buffers, attributes of which have

  //been completly exported from SP into SX

  //another hack for now

  //assign      read_detailed_quad = rb_ready & quad_data_ready ;
```

```

  reg [0:0]        rb_quad_data_count;

  assign       read_detailed_quad = rb_ready & rb_quad_data_count;


  always @(posedge sclk)

    begin

         if(srst)

           rb_quad_data_count <= 1'b0;

         else if(detailed_quad_valid)

           rb_quad_data_count <= rb_quad_data_count + 1;

    end


  wire [0:0] rb_quad_word_mux ;

  assign     rb_quad_word_mux = rb_quad_data_count;


 //     detailed_quad_data      =      {rb_id0,      export_index0,q_quad_data_read[74:43], q_quad_data_read[42:41],q_quad_data_read[40:39],

        //                        rb_id1,                       export_index1,q_quad_data_read[35:4], q_quad_data_read[3:2],q_quad_data_read[1:0]};

 // 91:0


  assign       sx_rb_quad_send = detailed_quad_valid & rb_ready;

  assign       sx_rb_id = (rb_quad_word_mux) ? quad_data_rb[45:44] : quad_data_rb[91:90];


  wire [1:0]     osx_quad_x;

  wire [1:0]     osx_quad_y;
```

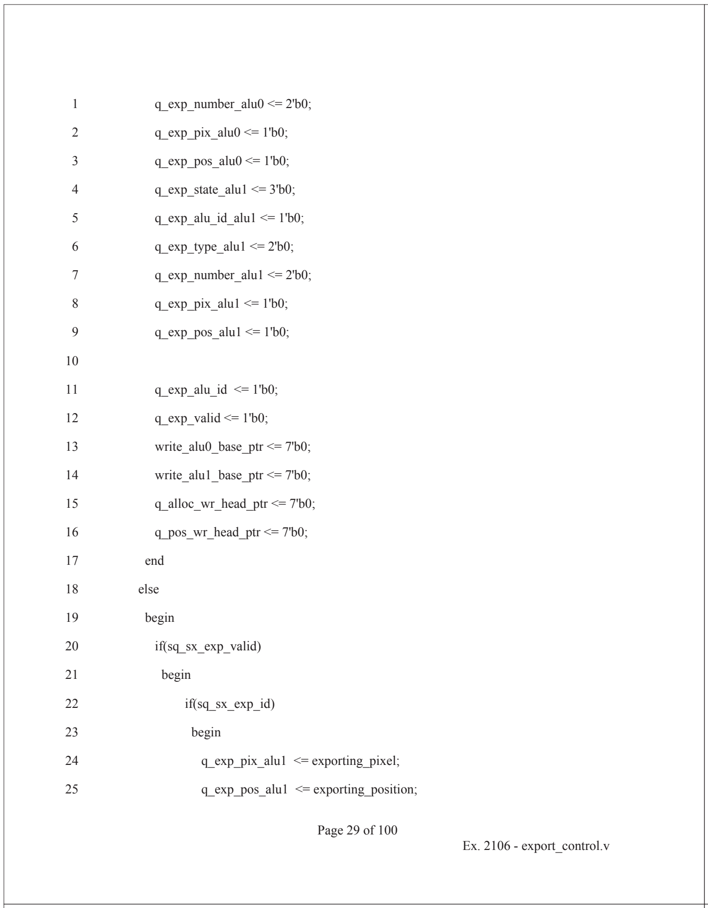
```
  wire [31:0]    osx_quad_mask;

  wire [0:0]     osx_quad_type;

  wire [3:0]     osx_quad_pixel;

  wire [7:0]     osx_quad_index;

  wire [0:0]     osx_rb0_quad_send,          osx_rb1_quad_send,           osx_rb2_quad_send, osx_rb3_quad_send;


  reg [1:0]      q_osx_quad_x;

  reg [1:0]      q_osx_quad_y;

  reg [31:0]     q_osx_quad_mask;

  reg [0:0]      q_osx_quad_type;

  reg [3:0]      q_osx_quad_pixel;

  reg [7:0]      q_osx_quad_index;

  reg [0:0]      q_osx_rb0_quad_send,     q_osx_rb1_quad_send,     q_osx_rb2_quad_send, q_osx_rb3_quad_send;


  assign         osx_quad_x = (rb_quad_word_mux) ? quad_data_rb[1:0] :quad_data_rb[47:46] ;

  assign         osx_quad_y      =       (rb_quad_word_mux)       ?       quad_data_rb[3:2] :quad_data_rb[49:48];

  assign         osx_quad_mask      =      (rb_quad_word_mux)      ?      quad_data_rb[35:4] :quad_data_rb[81:50];

  assign         osx_quad_index      =      (rb_quad_word_mux)      ?      quad_data_rb[43:36] :quad_data_rb[89:82];


  assign         osx_rb0_quad_send =  sx_rb_quad_send & (~sx_rb_id[0]) & (~sx_rb_id[1]);

  assign         osx_rb1_quad_send =  sx_rb_quad_send & (sx_rb_id[0]) & (~sx_rb_id[1]);

  assign         osx_rb2_quad_send =  sx_rb_quad_send & (~sx_rb_id[0]) & (sx_rb_id[1]);
```

```
assign          osx_rb3_quad_send =  sx_rb_quad_send & (sx_rb_id[0]) & (sx_rb_id[1]);

//registering the outputs  of  SX to RB(s) interface

always @(posedge sclk)
  begin
       if(srst)
         begin
            q_osx_rb0_quad_send <= 1'b0;
            q_osx_rb1_quad_send <= 1'b0;
            q_osx_rb2_quad_send <= 1'b0;
            q_osx_rb3_quad_send <= 1'b0;

         end
       else
         begin
            q_osx_quad_x <= osx_quad_x;
            q_osx_quad_y <= osx_quad_y;
            q_osx_quad_mask  <= osx_quad_mask;
            q_osx_quad_index <= osx_quad_index;
            q_osx_rb0_quad_send <= osx_rb0_quad_send ;
            q_osx_rb1_quad_send <= osx_rb1_quad_send ;
            q_osx_rb2_quad_send <= osx_rb2_quad_send ;
            q_osx_rb3_quad_send <= osx_rb3_quad_send ;
         end
```

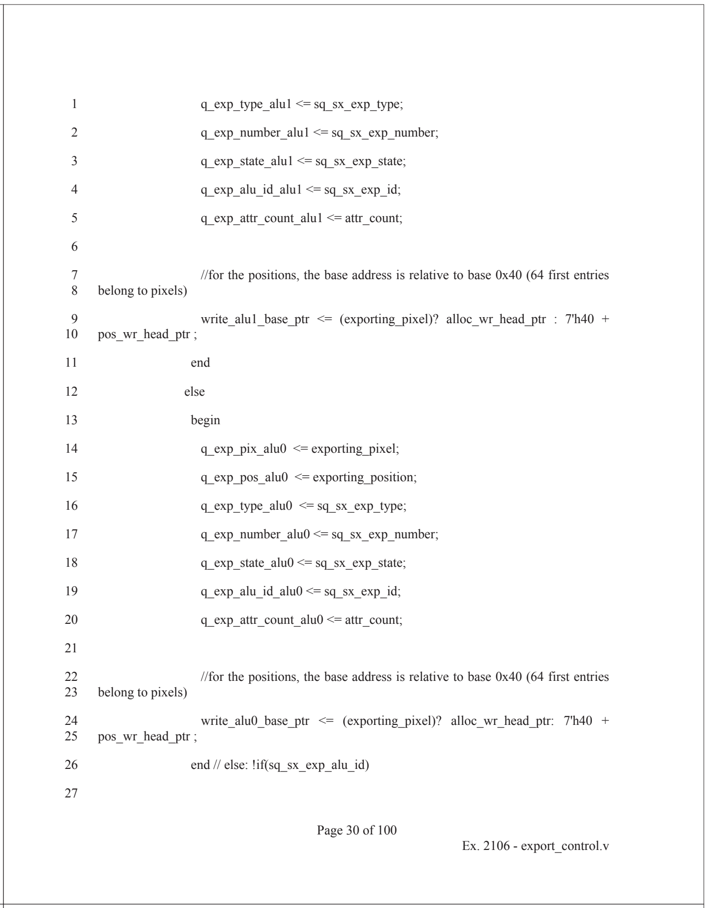
```
   end // always @ (posedge sclk)


 //assigning the output signals for the SX - RB quad interface
 assign     sx_rb_quad_x = q_osx_quad_x;
 assign         sx_rb_quad_y = q_osx_quad_y;
 assign         sx_rb_quad_mask = q_osx_quad_mask;
 //assign        sx_rb_quad_type; //pixel exports vs. pass through data ???? ANDI
 //assign        sx_rb_quad_pixel; // valid bits for the pixels  ???? ANDI
 assign         sx_rb_quad_index = q_osx_quad_index;
 assign         sx_rb0_quad_send = q_osx_rb0_quad_send;
 assign         sx_rb1_quad_send = q_osx_rb1_quad_send;
 assign         sx_rb2_quad_send = q_osx_rb2_quad_send;
 assign         sx_rb3_quad_send = q_osx_rb3_quad_send;



 //-------------------------------------------------------------------------------------------
 //SQ_SX  Export Control (allocation/deallocation)  Bus Interface ....registering the data
 //-------------------------------------------------------------------------------------------

 //register the last request on thread 0
 reg [1:0]      q_exp_type_alu0;
 reg [1:0]      q_exp_number_alu0;
 reg [2:0]      q_exp_state_alu0;
```

```
reg [0:0]      q_exp_alu_id_alu0;
reg [0:0]      q_exp_pos_alu0;



reg [0:0]      q_exp_alu_id;

//register the last request on thread 1
reg [1:0]      q_exp_type_alu1;
reg [1:0]      q_exp_number_alu1;
reg [2:0]      q_exp_state_alu1;
reg [0:0]      q_exp_alu_id_alu1;
reg [0:0]      q_exp_pix_alu1;
reg [0:0]      q_exp_pos_alu1;

reg [0:0]      q_exp_valid;
reg [6:0]      write_alu0_base_ptr, write_alu1_base_ptr;
reg [6:0]      alloc_wr_head_ptr,q_alloc_wr_head_ptr ; //points to where the next allocated
space will start
reg [6:0]      alloc_wr_tail_ptr ; //points to where the end of free export space is



//point to where the next position allocation space will be when the sequencer asks for it
//this value is relative to location 0x40 (64 first entries in the buffer belong to pixel data)
reg [3:0]      pos_wr_head_ptr,q_pos_wr_head_ptr ;
reg [3:0]      pos_wr_tail_ptr ; //points to where the end of free export space is

```

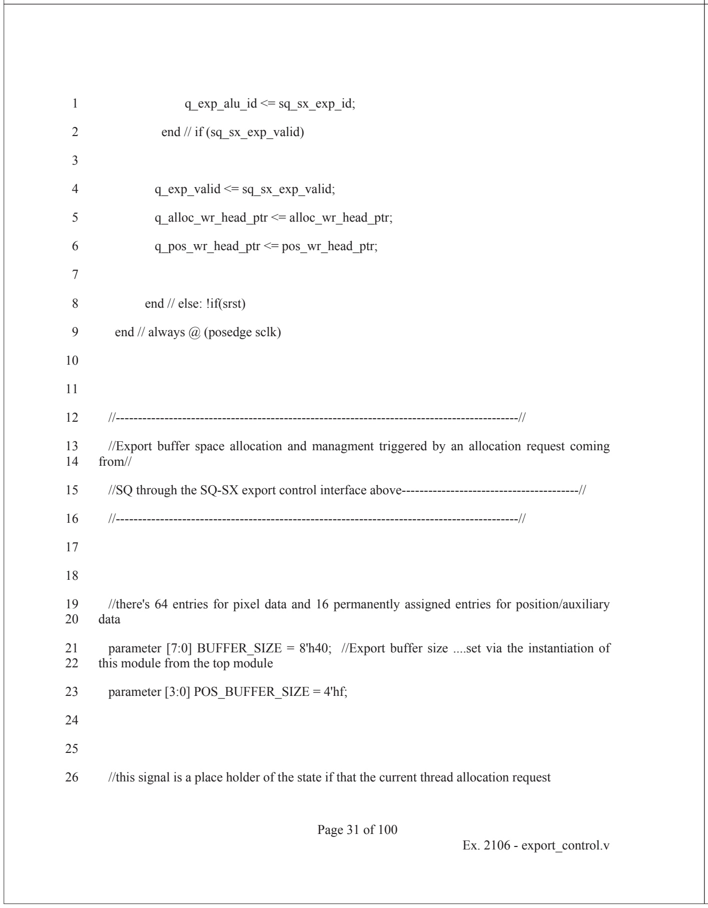
```
 wire [3:0]     attr_count;
 reg [2:0]      q_exp_attr_count_alu0, q_exp_attr_count_alu1;


 //a different write pointer for each alu thread
 reg [6:0]      write_alu0_ptr,write_alu1_ptr;



 //decoding on what kind of type of export request we are getting ....(pixel vs. vertex vs.
memory exports)

 wire [0:0]     exporting_pixel;
 assign         exporting_pixel = ~sq_sx_exp_type[1] & (sq_sx_exp_type[0] |
~sq_sx_exp_type[0]);

 wire [0:0]     exporting_position;
 assign         exporting_position = ~sq_sx_exp_type[0] & sq_sx_exp_type[1];



 always @(posedge sclk)
   begin
        if(srst)
          begin
             q_exp_state_alu0 <= 3'b0;
             q_exp_alu_id_alu0 <= 1'b0;
             q_exp_type_alu0 <= 2'b0;
```



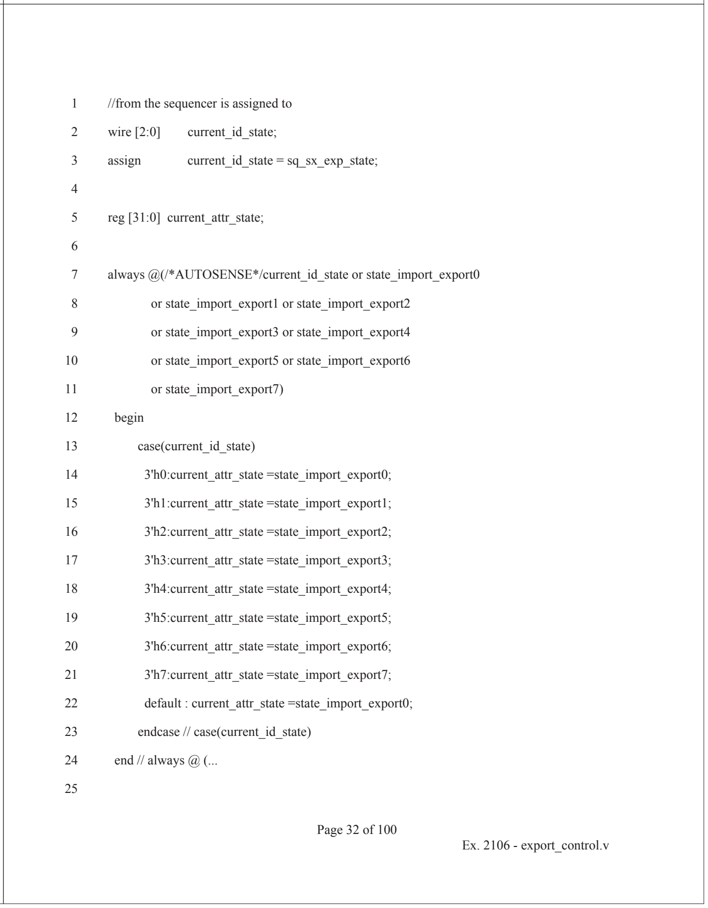
```
        q_exp_number_alu0 <= 2'b0;
        q_exp_pix_alu0 <= 1'b0;
        q_exp_pos_alu0 <= 1'b0;
        q_exp_state_alu1 <= 3'b0;
        q_exp_alu_id_alu1 <= 1'b0;
        q_exp_type_alu1 <= 2'b0;
        q_exp_number_alu1 <= 2'b0;
        q_exp_pix_alu1 <= 1'b0;
        q_exp_pos_alu1 <= 1'b0;

        q_exp_alu_id  <= 1'b0;
        q_exp_valid <= 1'b0;
        write_alu0_base_ptr <= 7'b0;
        write_alu1_base_ptr <= 7'b0;
        q_alloc_wr_head_ptr <= 7'b0;
        q_pos_wr_head_ptr <= 7'b0;
      end
    else
      begin
        if(sq_sx_exp_valid)
          begin
            if(sq_sx_exp_id)
              begin
                q_exp_pix_alu1  <= exporting_pixel;
                q_exp_pos_alu1  <= exporting_position;
```



```
                q_exp_type_alu1 <= sq_sx_exp_type;
                q_exp_number_alu1 <= sq_sx_exp_number;
                q_exp_state_alu1 <= sq_sx_exp_state;
                q_exp_alu_id_alu1 <= sq_sx_exp_id;
                q_exp_attr_count_alu1 <= attr_count;

                //for the positions, the base address is relative to base 0x40 (64 first entries belong to pixels)
                write_alu1_base_ptr  <=  (exporting_pixel)?  alloc_wr_head_ptr  :  7'h40  +  pos_wr_head_ptr ;
              end
            else
              begin
                q_exp_pix_alu0  <= exporting_pixel;
                q_exp_pos_alu0  <= exporting_position;
                q_exp_type_alu0  <= sq_sx_exp_type;
                q_exp_number_alu0 <= sq_sx_exp_number;
                q_exp_state_alu0 <= sq_sx_exp_state;
                q_exp_alu_id_alu0 <= sq_sx_exp_id;
                q_exp_attr_count_alu0 <= attr_count;

                //for the positions, the base address is relative to base 0x40 (64 first entries belong to pixels)
                write_alu0_base_ptr  <=  (exporting_pixel)?  alloc_wr_head_ptr:  7'h40  +  pos_wr_head_ptr ;
              end // else: !if(sq_sx_exp_alu_id)

```



```
                q_exp_alu_id <= sq_sx_exp_id;
          end // if (sq_sx_exp_valid)

        q_exp_valid <= sq_sx_exp_valid;
        q_alloc_wr_head_ptr <= alloc_wr_head_ptr;
        q_pos_wr_head_ptr <= pos_wr_head_ptr;

      end // else: !if(srst)
  end // always @ (posedge sclk)


  //-----------------------------------------------------------------------------------------//
  //Export buffer space allocation and managment triggered by an allocation request coming from//
  //SQ through the SQ-SX export control interface above----------------------------------------//
  //-----------------------------------------------------------------------------------------//


  //there's 64 entries for pixel data and 16 permanently assigned entries for position/auxiliary data
  parameter [7:0] BUFFER_SIZE = 8'h40;  //Export buffer size ....set via the instantiation of this module from the top module
  parameter [3:0] POS_BUFFER_SIZE = 4'hf;


  //this signal is a place holder of the state if that the current thread allocation request
```



```
  //from the sequencer is assigned to
  wire [2:0]        current_id_state;
  assign            current_id_state = sq_sx_exp_state;

  reg [31:0]  current_attr_state;

  always @(/*AUTOSENSE*/current_id_state or state_import_export0
           or state_import_export1 or state_import_export2
           or state_import_export3 or state_import_export4
           or state_import_export5 or state_import_export6
           or state_import_export7)
    begin
      case(current_id_state)
        3'h0:current_attr_state =state_import_export0;
        3'h1:current_attr_state =state_import_export1;
        3'h2:current_attr_state =state_import_export2;
        3'h3:current_attr_state =state_import_export3;
        3'h4:current_attr_state =state_import_export4;
        3'h5:current_attr_state =state_import_export5;
        3'h6:current_attr_state =state_import_export6;
        3'h7:current_attr_state =state_import_export7;
        default : current_attr_state =state_import_export0;
      endcase // case(current_id_state)
    end // always @ (...

  //------------------------------------------------------------------------------------------------------------------//

  //Generation of the write index into the Export Buffers
  //Decoding the export data type based on the destination pointer coming down from SP

  //------------------------------------------------------------------------------------------------------------------//


`define COLOR0 6'h00
`define COLOR1 6'h01
`define COLOR2 6'h02
`define COLOR3 6'h03
`define COLORFOG0 6'h08
`define COLORFOG1 6'h09
`define COLORFOG2 6'h0a
`define COLORFOG3 6'h0b
`define Z_DATA    6'h3f
`define POSITION  6'h3e
`define SPRITE_EDGE 6'h3f
//`define NO_EXPORT    2'h0
`define PIXEL_EXPORT 2'h2
`define VERTEX_EXPORT 2'h1

```

```
  reg [2:0] attribute_offset, q_attribute_offset;
  wire [1:0] export_type;
  reg [0:0] position_aux, q_position_aux;

  //00:no export
  //01:vertex export
  //10:pixel export
  assign export_type = (sp0_sx_exp_alu_id) ? {sp0_sx_exporting[0] & q_exp_pix_alu1,sp0_sx_exporting[0] & ~q_exp_pix_alu1}:
         {sp0_sx_exporting[0] & q_exp_pix_alu0,sp0_sx_exporting[0] & ~q_exp_pix_alu0};

  always @(/*AUTOSENSE*/`COLOR0 or `COLOR1 or `COLOR2 or `COLOR3
           or `COLORFOG0 or `COLORFOG1 or `COLORFOG2 or `COLORFOG3
           or `PIXEL_EXPORT or `POSITION or `SPRITE_EDGE
           or `VERTEX_EXPORT or `Z_DATA or export_type
           or sp0_sx_exp_dest)
    begin
       case(export_type)
         `PIXEL_EXPORT:
           begin
              position_aux = 1'b0;
              case(sp0_sx_exp_dest)
                `COLOR0:attribute_offset = 3'h0;
                `COLOR1:attribute_offset = 3'h1;
                `COLOR2:attribute_offset = 3'h2;
```

```
                `COLOR3:attribute_offset = 3'h3;
                `COLORFOG0:attribute_offset = 3'h0;
                `COLORFOG1:attribute_offset = 3'h1;
                `COLORFOG2:attribute_offset = 3'h2;
                `COLORFOG3:attribute_offset = 3'h3;
                `Z_DATA:attribute_offset = 3'h4;
              endcase // case(sp0_sx_exp_dest)
           end // case: VERTEX
         `VERTEX_EXPORT:
           begin
              case(sp0_sx_exp_dest)
                `POSITION:
                  begin
                     attribute_offset = 3'h0;      // + count of the position vectors that have been exported so far
                     position_aux = 1'b0;
                  end
                `SPRITE_EDGE:
                  begin
                     attribute_offset = 3'h4; //starting offset is always relative position 4
                     position_aux = 1'b1;
                  end
              endcase // case(sp0_sx_exp_dest
           end // case: VERTEX
         default : attribute_offset = 3'h0;
       endcase // case(sp0_sx_exporting)
```

```
    end // always @ (...


  //We mantain two write counts and two write pointers ...one for each ALU thread space allocated in export buffers.
  //Export Buffer write state machine
  reg [1:0] write_state,next_write_state;
  reg [2:0] alu0_attrib_count, alu1_attrib_count; //counts up to number of attributes for a  pixel (2 full quads accross two shader pipes)

`define NO_EXPORT   2'b00
`define EXPORT_ALU0 2'b01
`define EXPORT_ALU1 2'b10

  always @(posedge sclk)
    begin
       if(srst)
         begin
            write_state <= `NO_EXPORT;
         end
       else
         begin
            write_state <= next_write_state;
         end
    end // always @ (posedge sclk)

```

```

reg  no_export_state, export_alu0_state, export_alu1_state;

always @(/*AUTOSENSE*/`EXPORT_ALU0 or `EXPORT_ALU1 or `NO_EXPORT
        or sp0_sx_exp_alu_id or valid_export or write_state)
 begin
    case(write_state)
      `NO_EXPORT:
        begin
          if(valid_export & (~sp0_sx_exp_alu_id))
              next_write_state = `EXPORT_ALU0;
          else if(valid_export & sp0_sx_exp_alu_id)
              next_write_state = `EXPORT_ALU1;
          else
              next_write_state = `NO_EXPORT;
        end
      `EXPORT_ALU0:
        begin
          if(~valid_export)
              next_write_state = `NO_EXPORT;
          else if(valid_export & sp0_sx_exp_alu_id)
              next_write_state = `EXPORT_ALU1;
          else
              next_write_state = `EXPORT_ALU0;
        end
```

```
      `EXPORT_ALU1:
        begin
          if(~valid_export)
              next_write_state = `NO_EXPORT;
          else if(valid_export & ~sp0_sx_exp_alu_id)
              next_write_state = `EXPORT_ALU0;
          else
              next_write_state = `EXPORT_ALU1;
        end
      default : next_write_state = `NO_EXPORT;
    endcase // case(write_state)
  end // always @ (...



  //counting the number of attributes per pixel being transfered from SP into SX
  //when all the attributes are present in Export Bufffers, the RB "detailed quad buffer" is popped
  //and the quad data is sent to the RB it belongs to.
  reg [6:0] exported_quads_count; //how many outstanding quads to be sent to RB
  wire      exported_quad_inc;   //increment the above count
  wire      exported_quad_dec;   //decrement the above count
  wire [1:0] quad_inc_dec;
  assign     quad_inc_dec = {exported_quad_inc,exported_quad_dec};
  assign     exported_quad_dec = read_detailed_quad;

```

```
always @(posedge sclk)
 begin
    if(srst)
      begin
        alu0_attrib_count <= 3'b0;
        alu1_attrib_count <= 3'b0;
      end
    else
      begin
        if(export_alu0_state)
          begin
              if(alu0_attrib_count >= q_exp_attr_count_alu0 )
                begin
                  alu0_attrib_count <= 3'b0;
                  exp_data_in_buff0 <= 1'b1;
                end
              else
                begin
                  alu0_attrib_count <= alu0_attrib_count + 1;
                  exp_data_in_buff0 <= 1'b0;
                end
          end // if (export_alu0_state)
        if(export_alu1_state)
          begin
              if(alu1_attrib_count >= q_exp_attr_count_alu1 )
```

```
                begin
                  alu1_attrib_count <= 3'b0;
                  exp_data_in_buff1 <= 1'b1;

                end
              else
                begin
                  alu1_attrib_count <= alu1_attrib_count + 1;
                  exp_data_in_buff1 <= 1'b0;
                end
          end // if (export_alu1_state)
      end // else: !if(srst)
  end // always @ (posedge sclk)

  //--------------------------------------------------------------------------


  //if any of the thread finished a given pixel...2 quads accross the 8 export buffers beloning to one SX
  assign exported_quad_inc = exp_data_in_buff0 | exp_data_in_buff1;
  assign quad_data_ready = exported_quads_count > 7'b0;


  //-------------------------------------------------------
  //exported_quads_count process
  //the count below is indicating the number of quads outstanding in detailed_quad_buffer that
```

```
 //have been completly exported into Export Buffers but not yet issued to RBs because the RBs are not
 //ready yet
 always @(posedge sclk)
   begin
        if(srst)
          begin
             exported_quads_count <= 7'b0;
          end
        else
          begin
             case(quad_inc_dec)
               00:; //do nothing
               01: exported_quads_count <= exported_quads_count - 1;
               10: exported_quads_count <= exported_quads_count + 2;
               11:; //do nothing
             endcase // case(quad_inc_dec)
          end // else: !if(srst)
   end

 //combinational logic based on write_state current state value
 //------------------------------------------------------------------
 //final index into the export buffer
 //this values gets sent to export_buffer logic as a write address
 reg q_sp0_exp_alu_id;

```

```
 always @(posedge sclk)
   begin
        if(srst)
          q_sp0_exp_alu_id <= 1'b0;
        else
          q_sp0_exp_alu_id <= sp0_sx_exp_alu_id;
   end
 assign    export_index =  (q_sp0_exp_alu_id) ? write_alu1_ptr:write_alu0_ptr;


 reg         export_buffer_wen,export_buffer_wew;
 //counts the number of position export transactions comming from the SP
 //the count would count up to 4 if only positions, and 8 if positions + sprite/edges
 //64 positions would come first and then sprite/edges (if state indicates their presence)

 reg [1:0] pos0_exp_count ; //first position thread
 reg [1:0] pos1_exp_count ; //second position thread
 wire      pos0_exported, pos1_exported;
 reg [1:0] q_pos0_exp_count, q_pos1_exp_count;
 wire      pos_data_ready;


 //depending on what kind of export we are dealing with (pixel vs. position)
 //the write_alu_ptr is calculated differently
 //in the case of the position data, the exports are stored sequentially in the upper
```

```
 //16 entries of the export buffer. In the case of pixel exports depending on the order of the
 //exports the storing is not done in sequential order. Within a buffer assigned to a given thread
 //the colors 0-3 are stored first and then the z vector.


 //mantaining a vector count per attribute per thread.
 //in other word : for each attribute being exported we count up to 4 vectors of 16 (128) values each
 //this count would go from 0 to 3.

 reg [3:0] vector_count0;
 reg [3:0] vector_count1;


 //forcing the number of attributes per pixel to 1 ...hack   ANDI
 //eventually, the number of attributes per pixel will be factored in
 //the sequencer allocation request once we go to the new Sequencer allocation scheme.

 parameter [3:0] attr_exp_count = 1'b1;

 wire [5:0]        vector_wr0_offset;
 wire [5:0]        vector_wr1_offset;

 assign            vector_wr0_offset = attr_exp_count * vector_count0;
  assign           vector_wr1_offset = attr_exp_count * vector_count1;

```

```
 always @(posedge sclk)
   begin
        if(srst)
          vector_count0 <= 2'b00;
        else if((q0_valid_pixel_export | q1_valid_pixel_export) & export_alu0_state)
          vector_count0 <= vector_count0 + 1;
   end

 always @(posedge sclk)
   begin
        if(srst)
          vector_count1 <= 2'b00;
        else if(valid_pixel_export & export_alu1_state)
          vector_count1 <= vector_count1 + 1;
   end


 reg [6:0]  write_pos0_ptr, write_color0_ptr;
 reg [6:0]  write_pos1_ptr, write_color1_ptr;



 //registering the pointer values before assigning them in the case statement tied to the write state machine

 always @(posedge sclk)
```

```
  begin
      if(srst)
        begin
           write_pos0_ptr <= 7'b0;
           write_color0_ptr <= 7'b0;
           write_pos1_ptr <= 7'b0;
           write_color1_ptr <= 7'b0;
        end
      else
        begin
           write_pos0_ptr <= write_alu0_base_ptr + q_attribute_offset + pos0_exp_count;
           write_color0_ptr <= write_alu0_base_ptr + q_attribute_offset + vector_wr0_offset;
           write_pos1_ptr <= write_alu1_base_ptr + q_attribute_offset + pos1_exp_count;
           write_color1_ptr <= write_alu1_base_ptr + q_attribute_offset + vector_wr1_offset;
        end
  end // always @ (posedge sclk)



always @(/*AUTOSENSE*/`EXPORT_ALU0 or `EXPORT_ALU1 or `NO_EXPORT
          or pos0_exp_count or pos1_exp_count or q_attribute_offset
          or q_exp_pix_alu0 or q_exp_pix_alu1 or vector_wr0_offset
          or vector_wr1_offset or write_alu0_base_ptr
          or write_alu1_base_ptr or write_state)
  begin
```

```
      case(write_state)
        `NO_EXPORT:
          begin
             //turn off the write enables
             no_export_state = 1'b1;
             export_alu0_state = 1'b0;
             export_alu1_state = 1'b0;
             export_buffer_wen = 1'b0;
             export_buffer_wew = 1'b0;
          end
        `EXPORT_ALU0:
          begin
             //turn on the write enables
             write_alu0_ptr = (q_exp_pix_alu0) ? write_color0_ptr : write_pos0_ptr;
             no_export_state = 1'b0;
             export_alu0_state = 1'b1;
             export_alu1_state = 1'b0;
             export_buffer_wen = 1'b1;
             export_buffer_wew = 1'b1;
          end
        `EXPORT_ALU1:
          begin
             //turn on the write enables
             write_alu1_ptr = (q_exp_pix_alu1) ? write_color1_ptr: write_pos1_ptr;
             no_export_state = 1'b0;
```

```
             export_alu0_state = 1'b0;
             export_alu1_state = 1'b1;
             export_buffer_wen = 1'b1;
             export_buffer_wew = 1'b1;
          end
        default :
          begin
             //turn off the write enables
             no_export_state = 1'b1;
             export_alu0_state = 1'b0;
             export_alu1_state = 1'b0;
             export_buffer_wen = 1'b0;
             export_buffer_wew = 1'b0;
          end
      endcase // case(write_state)
  end // always @ (...


//------------------------------------------------------------------------------------
//Write Pointer management for the pixel/poisition exports
//------------------------------------------------------------------------------------
//alloc_wr_head_ptr indicates the offset location of the next available space that
//can be used by the next allocation request coming from the Sequencer
//Once a given valid export allocation request comes, the value of alloc_wr_head_ptr is
//incemented by buffer_size
```

```

   reg [7:0]  buffer_size;
   reg [3:0]  pos_buffer_size;
   reg        exported_pos_inc;
   wire       exported_pos_dec;
   reg [6:0]  exported_pos_count;
   wire       clipp_outstanding_req;
   wire       sx_pa_req_rtr;
   wire [4:0] pa_req_packet;
   wire       pa_pos_req;
   wire [4:0] pa_req_control;
   reg [4:0]  q_pa_req_control;
   reg        no_aux_buffer;

   //deriving the size of the buffer space required for current thread allocation request coming from
   //sequencer via SQ_SX_exp interface.

   always@(/*AUTOSENSE*/sq_sx_exp_number or sq_sx_exp_type)
     begin
        case(sq_sx_exp_type)
          2'b00: //pixels...no Z
            begin
               case(sq_sx_exp_number)
                 2'b00: buffer_size = 8'h4;
                 2'b01: buffer_size = 8'h8;
```

```
                2'b10: buffer_size = 8'hc;
                2'b11: buffer_size = 8'h10;
            endcase // case(sq_sx_exp_number)
          end
        2'b01: //pixels...with Z
          begin
            case(sq_sx_exp_number)
                2'b00:buffer_size = 8'h8;
                2'b01:buffer_size = 8'hc;
                2'b10:buffer_size = 8'h10;
                2'b11:buffer_size = 8'h14;
            endcase // case(sq_sx_exp_number)
          end
        2'b10:
          begin
            case(sq_sx_exp_number)
                2'b00:
                  begin
                    pos_buffer_size = 8'h4;
                    no_aux_buffer = 1'b1;
                  end
                2'b01:
                  begin
                    pos_buffer_size = 8'h8;
                    no_aux_buffer = 1'b0;
```

```
                  end
                default:
                  begin
                    pos_buffer_size = 8'h4;
                    no_aux_buffer = 1'b0;
                  end
               endcase // case(sq_sx_exp_number)
          end
        2'b11:
          begin
            case(sq_sx_exp_number)
                2'b00:;
                2'b01:;
                2'b10:;
                2'b11:;
               endcase // case(sq_sx_exp_number)
          end
      endcase // case(sq_sx_exp_type)
    end // always@ (...

  wire [2:0] colors_present;
  //assign     colors_present = current_attr_state[3:1];

  assign colors_present = 2'b10;
  wire [0:0] z_present;
```

```
  assign     z_present = 1'b0;
  wire [2:0] pixel_size = colors_present + z_present;
  reg        q1_exp_valid;


  always @(posedge sclk)
    begin
        if(srst)
          begin
            alloc_wr_head_ptr <= 7'b0;
            alloc_wr_tail_ptr <= BUFFER_SIZE;
            q1_exp_valid <= 1'b0;
          end
        else
          begin
            if(sq_sx_exp_valid & exporting_pixel)
              begin
                  alloc_wr_head_ptr <= alloc_wr_head_ptr + buffer_size;
              end
            q1_exp_valid <= q_exp_valid;
          end // else: !if(srst)
    end // always @ (posedge sclk)

  wire [0:0] ring_wrapped;
  wire [6:0] space_avail;
```

```
  assign     ring_wrapped = alloc_wr_head_ptr > alloc_wr_tail_ptr;
  assign          space_avail = (ring_wrapped) ? BUFFER_SIZE - alloc_wr_head_ptr + alloc_wr_tail_ptr : alloc_wr_tail_ptr -alloc_wr_head_ptr;

  //----------------------------------------------------------------------------------------------------------//
  //--Position Export write pointer managment
  //----------------------------------------------------------------------------------------------------------//


  //increment vs. decrement logic for the available position data in
  //the position count
  wire [1:0] pos_inc_dec;

  //00: do nothing
  //01: decrement the count of remaining position data in the position buffer
  //10: increment count of the available position data in the position buffer
  //11: do nothing
  assign     pos_inc_dec = {exported_pos_inc,exported_pos_dec};

  wire       pos_outstanding;
  assign     pos_outstanding = (exported_pos_count) ? 1'b1 : 1'b0;

  assign     clipp_outstanding_req = pos_outstanding & pa_pos_req;  //there's a position request from Clipper/PA

  //two separate counters,one for each position thread
```

```
   //increment the counter if we have a position export coming down from the shader pipe

   always @(posedge sclk)
     begin
            if(srst)
              begin
                 pos0_exp_count <= 2'b0;
                 pos1_exp_count <= 2'b0;
                 q_position_aux <= 1'b0;
                 q_attribute_offset <= 2'b0;
              end
            else
              begin
                 q_position_aux <= position_aux;
                 q_attribute_offset <= attribute_offset;
                 if(q_sp0_exp_alu_id)
                   begin
                          if(~no_aux_buffer)
                            begin
                               pos1_exp_count <= (~q_exp_pix_alu1 & q0_valid_export &
q_position_aux) ? pos1_exp_count + 1'b1:pos1_exp_count ;
                               exported_pos_inc <= (~q_exp_pix_alu1 & q0_valid_export &
q_position_aux) ? 1: 0;
                            end
                          else
                            begin
```

```
                               pos1_exp_count <= (~q_exp_pix_alu1 & q0_valid_export) ?
pos1_exp_count + 1'b1:pos1_exp_count ;

                               //use to increment the position count
                               exported_pos_inc <= (~q_exp_pix_alu1 & q0_valid_export) ? 1: 0;
                            end // else: !if(~no_aux_buffer)
                   end
                 else
                   begin
                          if(~no_aux_buffer)
                            begin
                               pos0_exp_count <= (~q_exp_pix_alu0 & q0_valid_export &
q_position_aux) ? pos0_exp_count + 1'b1:pos0_exp_count ;
                               exported_pos_inc <= (~q_exp_pix_alu0 & q0_valid_export &
q_position_aux) ? 1: 0;
                            end
                          else
                            begin
                               pos0_exp_count <= (~q_exp_pix_alu0 & q0_valid_export) ?
pos0_exp_count + 1'b1:pos0_exp_count ;

                               //use to increment the position count
                               exported_pos_inc <= (~q_exp_pix_alu0 & q0_valid_export) ? 1: 0;
                            end // else: !if(~no_aux_buffer)
                   end
              end // else: !if(srst)
     end // always @ (posedge sclk)
```

```

   //------------------------------------------------------------------------------------------------------//



   //not sure on whether we need these registered versions of count ANDI ???
   always @(posedge sclk)
     begin
            if(srst)
              begin
                 q_pos0_exp_count <= 2'b0;
                 q_pos1_exp_count <= 2'b0;
              end
            else
              begin
                 q_pos0_exp_count <= pos0_exp_count;
                 q_pos1_exp_count <= pos1_exp_count;
              end
     end

   //this flags may potentially be used in the logic
   //but for now they only have a debug purpose
   reg      pos_buff0_ready , pos_buff1_ready;

   always @(posedge sclk)
     begin
```

```
            if((q_pos0_exp_count == 2'b11)&(~pos_buff0_ready))
              pos_buff0_ready <= 1'b1;
            else
              pos_buff0_ready <= 1'b0;

            if((q_pos1_exp_count == 2'b11)&(~pos_buff1_ready))
              pos_buff1_ready <= 1'b1;
            else
              pos_buff1_ready <= 1'b0;
     end // always @ (posedge sclk)


   assign     pos_data_ready = pos_buff0_ready & pos_buff1_ready;



   always @(posedge sclk)
     begin
            if(srst)
              begin
                 exported_pos_count <= 7'b0;
              end
            else
              begin
                 case(pos_inc_dec)
                   2'b00:; //do nothing
```

```
                    2'b01: exported_pos_count <= exported_pos_count - 1;
                    2'b10: exported_pos_count <= exported_pos_count + 1;
                    2'b11:; //do nothing
                  endcase // case(pos_inc_dec)
               end // else: !if(srst)
        end

    always @(posedge sclk)
        begin
            if(srst)
               begin
                  pos_wr_head_ptr <= 4'h0;   // location 80 ...the upper 16 locations reserved for position data
                  pos_wr_tail_ptr <= 4'hf;  //last location in the position export buffer
               end
            else
               begin
                  if(sq_sx_exp_valid & exporting_position)  //exporting vertex positions
                    begin
                         pos_wr_head_ptr <= pos_wr_head_ptr + pos_buffer_size;
                    end
               end // else: !if(srst)
        end // always @ (posedge sclk)


    //-----Export buffer status output interface to SQ-------------------/
```

```

    reg [0:0]  oexp_count_rdy;
    reg [0:0]  oexp_pos_avail;
    reg [6:0]  oexp_buff_avail;


    //registering the outputs
    always @(posedge sclk)
      begin
            oexp_count_rdy <= q_exp_valid;
            oexp_buff_avail <= space_avail;
      end

    assign sx_sq_exp_count_rdy = oexp_count_rdy;
    assign sx_sq_exp_buf_avail = oexp_buff_avail;


    //-------------------------------------------------------------------------------------//
    //State/rbbm bus decoding logic-----------------------------------------------------//
    //-------------------------------------------------------------------------------------//



   // SQ_IMPORTS_EXPORTS         <GFXDEC0:0x043C>    32 {
    //                                                PS_EXPORT_MODE     4:0       NUM DEF=2;
```

```
    //                                           VS_EXPORT_MODE     9:8       NUM DEF=0;
    //                                       PARAM_GEN_I0     12:12 ALPHA {
    //                                                                    "Take parameter 0 from the Parameter Cache.",   //                                                                    "Generate Parameter 0."
    //                                           } DEF=0;
    //                             GEN_INDEX          16:16         ALPHA {
    //                                                                    "Do not auto generate index adresses.",
    //                                                                   "Auto generate index adresses."
    //                                                         } DEF=0;
    //};
    // SQ_IMPORTS_EXPORTS "Import export control"
    //   {
    //    PS_EXPORT_MODE      "Pixel Shader exporting mode\n. 0xxxx: Normal mode\n. 1xxxx: Multipass.\n If normal,
    //                        bbbz where bbb is how many color we export (0-4) and z is export z or not.\n If multipass mode, 1-12 exports for color.";
    //    VS_EXPORT_MODE      "0: Position (1 vector).\n 1: Position (2 vectors).\n 2: Multipass ";
    //    PARAM_GEN_I0            "Do we overwrite or not parameter 0 with generated XYST or XYxF.";
    //    GEN_INDEX        "Auto generates an address from 0 to XX. Puts the results into R0 or 1 for pixels shaders and R2 for vertex shaders";
    //    };


//   RB_ALPHA_REF        ""
```

```
//{  ALPHA_REF        "Reference value for alpha test, which is specified in IEEE floating point but stored in the RB internal format and converted back to IEEE floating point on reads. Therefore, this register may read back as a different value.";
//};

//   RB_ALPHA_REF       <GFXDEC0:0x081C>  32 FA {
//   ALPHA_REF             31:0   DATA_TYPE="float";
//};





    //-------------------------------------------------------//
    //GFX decode space values
    //---------------------------------------------------//
    parameter [3:0]  gfxdec0 = 4'h8,
                              gfxdec1 = 4'h9,
                              gfxdec2 = 4'ha,
                              gfxdec3 = 4'hb,
                              gfxdec4 = 4'hc,
                              gfxdec5 = 4'hd,
                              gfxdec6 = 4'he,
                              gfxdec7 = 4'hf;

    // register offsets
```

```
parameter [9:0] sq_imports_exports = 10'h10F;
parameter [9:0] rb_alpha_ref = 10'h207;
parameter [9:0] rb_alpha_test = 10'h207;

//----------------------------------------------------------------------//
wire [14:0]       import_export0_a;
wire [14:0]       import_export1_a;
wire [14:0]       import_export2_a;
wire [14:0]       import_export3_a;
wire [14:0]       import_export4_a;
wire [14:0]       import_export5_a;
wire [14:0]       import_export6_a;
wire [14:0]       import_export7_a;

wire [14:0]       alpha_ref0_a;
wire [14:0]       alpha_ref1_a;
wire [14:0]       alpha_ref2_a;
wire [14:0]       alpha_ref3_a;
wire [14:0]       alpha_ref4_a;
wire [14:0]       alpha_ref5_a;
wire [14:0]       alpha_ref6_a;
wire [14:0]       alpha_ref7_a;

wire [14:0]       alpha_test0_a;
wire [14:0]       alpha_test1_a;
```

```
wire [14:0]       alpha_test2_a;
wire [14:0]       alpha_test3_a;
wire [14:0]       alpha_test4_a;
wire [14:0]       alpha_test5_a;
wire [14:0]       alpha_test6_a;
wire [14:0]       alpha_test7_a;


//Absolute memory mapped registers for the state registers
assign            import_export0_a = {gfxdec0,sq_imports_exports};
assign            import_export1_a = {gfxdec1,sq_imports_exports};
assign            import_export2_a = {gfxdec2,sq_imports_exports};
assign            import_export3_a = {gfxdec3,sq_imports_exports};
assign            import_export4_a = {gfxdec4,sq_imports_exports};
assign            import_export5_a = {gfxdec5,sq_imports_exports};
assign            import_export6_a = {gfxdec6,sq_imports_exports};
assign            import_export7_a = {gfxdec7,sq_imports_exports};

assign       alpha_ref0_a = {gfxdec0,rb_alpha_ref};
assign       alpha_ref1_a = {gfxdec1,rb_alpha_ref};
assign       alpha_ref2_a = {gfxdec2,rb_alpha_ref};
assign       alpha_ref3_a = {gfxdec3,rb_alpha_ref};
assign       alpha_ref4_a = {gfxdec4,rb_alpha_ref};
assign       alpha_ref5_a = {gfxdec5,rb_alpha_ref};
assign       alpha_ref6_a = {gfxdec6,rb_alpha_ref};
```

```
assign       alpha_ref7_a = {gfxdec7,rb_alpha_ref};

assign            alpha_test0_a = {gfxdec0,rb_alpha_test};
assign            alpha_test1_a = {gfxdec1,rb_alpha_test};
assign            alpha_test2_a = {gfxdec2,rb_alpha_test};
assign            alpha_test3_a = {gfxdec3,rb_alpha_test};
assign            alpha_test4_a = {gfxdec4,rb_alpha_test};
assign            alpha_test5_a = {gfxdec5,rb_alpha_test};
assign            alpha_test6_a = {gfxdec6,rb_alpha_test};
assign            alpha_test7_a = {gfxdec7,rb_alpha_test};


//---------------------------------------------------------------------------
//State Registers
//----------------------------------------------------------------------------
// reg [31:0]     state_import_export0,state_import_export1;
// reg [31:0]     state_import_export2,state_import_export3;
// reg [31:0]     state_import_export4,state_import_export5;
// reg [31:0]     state_import_export6,state_import_export7;

reg [31:0]        state_alpha_ref0;
reg [31:0]        state_alpha_ref1;
reg [31:0]        state_alpha_ref2;
reg [31:0]        state_alpha_ref3;
reg [31:0]        state_alpha_ref4;
```

```
reg [31:0]        state_alpha_ref5;
reg [31:0]        state_alpha_ref6;
reg [31:0]        state_alpha_ref7;

reg [31:0]        state_alpha_test0;
reg [31:0]        state_alpha_test1;
reg [31:0]        state_alpha_test2;
reg [31:0]        state_alpha_test3;
reg [31:0]        state_alpha_test4;
reg [31:0]        state_alpha_test5;
reg [31:0]        state_alpha_test6;
reg [31:0]        state_alpha_test7;


always @(posedge sclk)
  begin
        if(state_soft_reset)
          begin
            state_import_export0 <= 32'b0;
            state_import_export1 <= 32'b0;
            state_import_export2 <= 32'b0;
            state_import_export3 <= 32'b0;
            state_import_export4 <= 32'b0;
            state_import_export5 <= 32'b0;
            state_import_export6 <= 32'b0;
```

```
                state_import_export7 <= 32'b0;
                state_alpha_ref0 <= 32'b0;
                state_alpha_ref1 <= 32'b0;
                state_alpha_ref2 <= 32'b0;
                state_alpha_ref3 <= 32'b0;
                state_alpha_ref4 <= 32'b0;
                state_alpha_ref5 <= 32'b0;
                state_alpha_ref6 <= 32'b0;
                state_alpha_ref7 <= 32'b0;
                state_alpha_test0 <= 32'b0;
                state_alpha_test1 <= 32'b0;
                state_alpha_test2 <= 32'b0;
                state_alpha_test3 <= 32'b0;
                state_alpha_test4 <= 32'b0;
                state_alpha_test5 <= 32'b0;
                state_alpha_test6 <= 32'b0;
                state_alpha_test7 <= 32'b0;

             end
          else if(state_we)
             begin
                case(state_a)
                  import_export0_a:state_import_export0 <= state_wd;
                  import_export1_a:state_import_export1 <= state_wd;
                  import_export2_a:state_import_export2 <= state_wd;
```

```
                  import_export3_a:state_import_export3 <= state_wd;
                  import_export4_a:state_import_export4 <= state_wd;
                  import_export5_a:state_import_export5 <= state_wd;
                  import_export6_a:state_import_export6 <= state_wd;
                  import_export7_a:state_import_export7 <= state_wd;
                  alpha_ref0_a : state_alpha_ref0 <= state_wd;
                  alpha_ref1_a : state_alpha_ref1 <= state_wd;
                  alpha_ref2_a : state_alpha_ref2 <= state_wd;
                  alpha_ref3_a : state_alpha_ref3 <= state_wd;
                  alpha_ref4_a : state_alpha_ref4 <= state_wd;
                  alpha_ref5_a : state_alpha_ref5 <= state_wd;
                  alpha_ref6_a : state_alpha_ref6 <= state_wd;
                  alpha_ref7_a : state_alpha_ref7 <= state_wd;
                  alpha_test0_a : state_alpha_test0 <= state_wd;
                  alpha_test1_a : state_alpha_test1 <= state_wd;
                  alpha_test2_a : state_alpha_test2 <= state_wd;
                  alpha_test3_a : state_alpha_test3 <= state_wd;
                  alpha_test4_a : state_alpha_test4 <= state_wd;
                  alpha_test5_a : state_alpha_test5 <= state_wd;
                  alpha_test6_a : state_alpha_test6 <= state_wd;
                  alpha_test7_a : state_alpha_test7 <= state_wd;
                endcase // case(state_a)
             end // if (state_we)
     end // always @ (posedge sclk)

```

```

  //----------------------------------------------------------------------------------------------------------//
  //Select Logic used to decide which contex will be used at any given time (selecting between
state(s) 0-7)
  //----------------------------------------------------------------------------------------------------------//

  reg [31:0] state_alpha_test,state_alpha_ref,state_import_export;

  wire [2:0] current_exp_state;
  assign    current_exp_state = (sp0_sx_exp_alu_id)? q_exp_state_alu1: q_exp_state_alu0;

  always @(/*AUTOSENSE*/current_exp_state or state_alpha_ref0
           or state_alpha_ref1 or state_alpha_ref2
           or state_alpha_ref3 or state_alpha_ref4
           or state_alpha_ref5 or state_alpha_ref6
           or state_alpha_ref7 or state_alpha_test0
           or state_alpha_test1 or state_alpha_test2
           or state_alpha_test3 or state_alpha_test4
           or state_alpha_test5 or state_alpha_test6
           or state_alpha_test7 or state_import_export0
           or state_import_export1 or state_import_export2
           or state_import_export3 or state_import_export4
           or state_import_export5 or state_import_export6
           or state_import_export7)
    begin
       case(current_exp_state)
```

```
          3'h0:
            begin
               state_alpha_test = state_alpha_test0;
               state_alpha_ref = state_alpha_ref0;
               state_import_export = state_import_export0;
            end
          3'h1:
            begin
               state_alpha_test = state_alpha_test1;
               state_alpha_ref = state_alpha_ref1;
               state_import_export = state_import_export1;
            end
          3'h2:
             begin
               state_alpha_test = state_alpha_test2;
               state_alpha_ref = state_alpha_ref2;
               state_import_export = state_import_export2;
            end
          3'h3:
            begin
               state_alpha_test = state_alpha_test3;
               state_alpha_ref = state_alpha_ref3;
               state_import_export = state_import_export3;
            end
          3'h4:
```

```
          begin
            state_alpha_test = state_alpha_test4;
            state_alpha_ref = state_alpha_ref4;
            state_import_export = state_import_export4;
          end
        3'h5:
          begin
            state_alpha_test = state_alpha_test5;
            state_alpha_ref = state_alpha_ref5;
            state_import_export = state_import_export5;
          end
        3'h6:
          begin
            state_alpha_test = state_alpha_test6;
            state_alpha_ref = state_alpha_ref6;
            state_import_export = state_import_export6;
          end
        3'h7:
          begin
            state_alpha_test = state_alpha_test7;
            state_alpha_ref = state_alpha_ref7;
            state_import_export = state_import_export7;
          end
        default:
          begin
```

```
            state_alpha_test = state_alpha_test0;
            state_alpha_ref = state_alpha_ref0;
            state_import_export = state_import_export0;
          end
        endcase // case(q_exp_state)
    end // always @ (...


  //-------------------------------------------------------------------------------------------------------------//
  //decoding the "attributes per pixel"  state for each alu thread.
  //-------------------------------------------------------------------------------------------------------------//
  reg [3:0] attr_state; //attribute state
  always @(/*AUTOSENSE*/sq_sx_exp_state or state_import_export0
           or state_import_export1 or state_import_export2
           or state_import_export3 or state_import_export4
           or state_import_export5 or state_import_export6
           or state_import_export7)
    begin
        case(sq_sx_exp_state)
          3'h0:attr_state = state_import_export0[3:0]; //bbbz (0-4 color attributes + z attribute)
          3'h1:attr_state = state_import_export1[3:0];
          3'h2:attr_state = state_import_export2[3:0];
          3'h3:attr_state = state_import_export3[3:0];
          3'h4:attr_state = state_import_export4[3:0];
          3'h5:attr_state = state_import_export5[3:0];
```

```
          3'h6:attr_state = state_import_export6[3:0];
          3'h7:attr_state = state_import_export7[3:0];
          default:attr_state = state_import_export0[3:0];
        endcase // case(sq_sx_exp_state)
    end // always @ (...

  assign attr_count = attr_state[3:1] + attr_state[0:0];


  //-------------------------------------------------------------------------------------------------------------//

  wire [0:0] mask_pix00, mask_pix01,mask_pix02,mask_pix03;
  wire [0:0] discard00, discard01,discard02,discard03;


  //FIRST QUAD data processing logic

  //first pixel of quad0
//        alpha_color_test   u00_alpha_test(    .mask_out(mask_pix00),    .pixel_discard(discard00),
.data_in(sp0_sx_data0), .alpha_test(state_alpha_test[2:0]),
              //              .alpha_test_enable(state_alpha_test[3]),
.alpha_to_mask_enable(state_alpha_test[4]), .alpha_ref(state_alpha_ref),
                    //
.discard_color_func(state_alpha_test[9:8]),.discard_alpha_func(state_alpha_test[11:10]),
.sclk(sclk) );

  //second pixel of quad0
```

```
//      alpha_color_test   u01_alpha_test(    .mask_out(mask_pix01),    .pixel_discard(discard01),
.data_in(sp0_sx_data1), .alpha_test(state_alpha_test[2:0]),
              //              .alpha_test_enable(state_alpha_test[3]),
.alpha_to_mask_enable(state_alpha_test[4]), .alpha_ref(state_alpha_ref),
                    //
.discard_color_func(state_alpha_test[9:8]),.discard_alpha_func(state_alpha_test[11:10]),
.sclk(sclk) );

  //second pixel of quad0
  //    alpha_color_test    u02_alpha_test(    .mask_out(mask_pix02),    .pixel_discard(discard02),
.data_in(sp0_sx_data2), .alpha_test(state_alpha_test[2:0]),
              //              .alpha_test_enable(state_alpha_test[3]),
.alpha_to_mask_enable(state_alpha_test[4]), .alpha_ref(state_alpha_ref),
                    //
.discard_color_func(state_alpha_test[9:8]),.discard_alpha_func(state_alpha_test[11:10]),
.sclk(sclk) );

  //second pixel of quad0
  //    alpha_color_test    u03_alpha_test(    .mask_out(mask_pix03),    .pixel_discard(discard03),
.data_in(sp0_sx_data3), .alpha_test(state_alpha_test[2:0]),
              //              .alpha_test_enable(state_alpha_test[3]),
.alpha_to_mask_enable(state_alpha_test[4]), .alpha_ref(state_alpha_ref),
                    //
.discard_color_func(state_alpha_test[9:8]),.discard_alpha_func(state_alpha_test[11:10]),
.sclk(sclk));



  wire [0:0] mask_pix10, mask_pix11,mask_pix12,mask_pix13;
  wire [0:0] discard10, discard11,discard12,discard13;


  //SECOND QUAD data processing logic


  //first pixel of quad1

  //    alpha_color_test    u10_alpha_test(    .mask_out(mask_pix10),    .pixel_discard(discard10),
.data_in(sp0_sx_data0), .alpha_test(state_alpha_test[2:0]),

            //              .alpha_test_enable(state_alpha_test[3]),
.alpha_to_mask_enable(state_alpha_test[4]), .alpha_ref(state_alpha_ref),

                        //
.discard_color_func(state_alpha_test[9:8]),.discard_alpha_func(state_alpha_test[11:10]),
.sclk(sclk) );


  //second pixel of quad1

  //    alpha_color_test    u11_alpha_test(    .mask_out(mask_pix11),    .pixel_discard(discard11),
.data_in(sp0_sx_data1), .alpha_test(state_alpha_test[2:0]),

            //              .alpha_test_enable(state_alpha_test[3]),
.alpha_to_mask_enable(state_alpha_test[4]), .alpha_ref(state_alpha_ref),

                        //
.discard_color_func(state_alpha_test[9:8]),.discard_alpha_func(state_alpha_test[11:10]),
.sclk(sclk) );


   //second pixel of quad1

  //    alpha_color_test    u12_alpha_test(    .mask_out(mask_pix12),    .pixel_discard(discard12),
.data_in(sp0_sx_data2), .alpha_test(state_alpha_test[2:0]),

            //              .alpha_test_enable(state_alpha_test[3]),
.alpha_to_mask_enable(state_alpha_test[4]), .alpha_ref(state_alpha_ref),

                        //
.discard_color_func(state_alpha_test[9:8]),.discard_alpha_func(state_alpha_test0[11:10]),
.sclk(sclk) );


  //second pixel of quad1
```

```
  //    alpha_color_test    u13_alpha_test(    .mask_out(mask_pix13),    .pixel_discard(discard13),
.data_in(sp0_sx_data3), .alpha_test(state_alpha_test[2:0]),

            //              .alpha_test_enable(state_alpha_test[3]),
.alpha_to_mask_enable(state_alpha_test[4]), .alpha_ref(state_alpha_ref),

                        //
.discard_color_func(state_alpha_test[9:8]),.discard_alpha_func(state_alpha_test[11:10]),
.sclk(sclk));


  //export buffers instatiated ...two banks of 4 128x128 register files each.


  reg [8:0]  exp_read_pointer;  //final read pointer in Export Buffers //8 bits for address + 1 bit
for keep/discard quad

  reg [8:0]  q_exp_read_pointer; //registered version of the above


  reg       exp_buff_read_en;

  reg       q_exp_buff_read_en;


  //flags representing a read request from the 5  different clients into SX (Clipper, RB0-3)

  reg          service_rb0, service_rb1,service_rb2, service_rb3, service_clipper;

  reg                                                                                    [1:0]
rb0_grant_count,rb1_grant_count,rb2_grant_count,rb3_grant_count,clipper_grant_count;

  reg   [1:0]     q_rb0_grant_count,q_rb1_grant_count,q_rb2_grant_count,q_rb3_grant_count   ,
q_clipp_grant_count;

  reg          read_valid_rb0, read_valid_rb1, read_valid_rb2, read_valid_rb3;

  reg          q_read_valid_rb0, q_read_valid_rb1, q_read_valid_rb2, q_read_valid_rb3;


  reg          read_valid_clipp, q_read_valid_clipp;

```

```
  //bypassing the alpha compare and color compare 1.0/0.0 logic ....revisit this logic ANDI !!!!!

  export_buffers  uexport_buffers(

                                  // Outputs

                                  .orb0_data(sx_rb0_color_data), .orb1_data(sx_rb1_color_data),

                                  .orb2_data(sx_rb2_color_data), .orb3_data(sx_rb3_color_data),

.orb0_data_valid(sx_rb0_color_send),.orb1_data_valid(sx_rb1_color_send),

.orb2_data_valid(sx_rb2_color_send),.orb3_data_valid(sx_rb3_color_send),

                                  .oclipp_data(sx_pa_data), .oclipp_data_valid(sx_pa_send),


                                  // Inputs

                                  .iread_addr(q_exp_read_pointer[7:0]),

                                  .iwrite_addr(export_index),

                                  .ipixel_data0(q0_sp0_data0),.ipixel_data1(q0_sp0_data1),

                                  .ipixel_data2(q0_sp0_data2),.ipixel_data3(q0_sp0_data3),

                                  .ipixel_data4(q0_sp1_data0),.ipixel_data5(q0_sp1_data1),

                                  .ipixel_data6(q0_sp1_data2),.ipixel_data7(q0_sp1_data3),

.iphase_rb0(q_rb0_grant_count),.iphase_rb1(q_rb1_grant_count),

.iphase_rb2(q_rb2_grant_count),.iphase_rb3(q_rb3_grant_count),

                                  .iphase_clipp(q_clipp_grant_count),

                                  .sclk(sclk),

                                  .srst(srst),

                                  .imem_wen(export_buffer_wen),

                                  .imem_wew(export_buffer_wew),
```

```
                                  .imem_re(q_exp_buff_read_en),

                                  .iread_valid_rb0(q_read_valid_rb0  &  q_exp_read_pointer[8]),
.iread_valid_rb1(q_read_valid_rb1 & q_exp_read_pointer[8]),

                                  .iread_valid_rb2(q_read_valid_rb2  &  q_exp_read_pointer[8]),
.iread_valid_rb3(q_read_valid_rb3 & q_exp_read_pointer[8]),

                                  .iread_valid_clipp(q_read_valid_clipp)

                                  );




  //------------------------------------------------------------------------------------------------------------
---------//

  //-----PA request position request interface skid buffer------------------------------------------------
------------------//

  //------------------------------------------------------------------------------------------------------------
---------//


  //this the request packet from PA

  //pa_sp_id : which bank of SPs data should come from

  //pa_offset : 0-3 in the group of 4 position vectors

  //pa_aux : read from the auxiliary buffer

  //pa_last: the last position request of this specific vector ...increment the read pointer by 1 and
free the space


  assign    pa_req_packet = {pa_sx_sp_id,pa_sx_offset,pa_sx_aux,pa_sx_last};


  skid_buff_top #(5,8) pa_pos_req_buff(

                                        .write_rts(pa_sx_req),
                                        .write_rtr(sx_pa_req_rtr),
                                        .write_data(pa_req_packet),
                                        .read_rts(pa_pos_req),
                                        .read_rtr(service_clipper),
                                        .read_data(pa_req_control),
                                        .clk(sclk),
                                        .reset(srst)
                                        );

 always @(posedge sclk)
  begin
     if(srst)
      q_pa_req_control <= 5'b0;
     else if(service_clipper)
      q_pa_req_control <= pa_pos_req;
  end


 //----------------------------------------------------------------------------------------------------------
---------//
 //Export Buffer Read logic
 //----------------------------------------------------------------------------------------------------------
--------//
 //------------------------------------------------------------ -------------//
 //There are four skid buffers ..one for each rb on the index interface
 //--------------------------------------------------------------------//
```

```

 wire     sx_to_rb0_index_rtr,sx_to_rb1_index_rtr,sx_to_rb2_index_rtr,sx_to_rb3_index_rtr;
 reg       osx_rb0_index_rtr,osx_rb1_index_rtr,osx_rb2_index_rtr,osx_rb3_index_rtr;
 wire     rb0_read_req,rb1_read_req, rb2_read_req,rb3_read_req;

 wire     rb0_color_ready, rb1_color_ready,rb2_color_ready,rb3_color_ready;

 //there's an outstanding request for color and RB is ready to accept color data for the next four
cycles
 assign    rb0_color_ready = rb0_read_req & rb0_sx_color_rtr;
 assign    rb1_color_ready = rb1_read_req & rb1_sx_color_rtr;
 assign    rb2_color_ready = rb2_read_req & rb2_sx_color_rtr;
 assign    rb3_color_ready = rb3_read_req & rb3_sx_color_rtr;

 wire [8:0] rb0_read_index, rb1_read_index, rb2_read_index, rb3_read_index;
 reg [8:0]  q_rb0_read_index,q_rb1_read_index,q_rb2_read_index,q_rb3_read_index;

 wire [8:0] rb0_index,rb1_index,rb2_index,rb3_index;



 assign    exported_pos_dec =  service_clipper;

 assign    rb0_index = {rb0_sx_index_op, rb0_sx_index};
 assign    rb1_index = {rb1_sx_index_op, rb1_sx_index};
 assign    rb2_index = {rb2_sx_index_op, rb2_sx_index};
 assign    rb3_index = {rb3_sx_index_op, rb3_sx_index};
```

```



 //latching in the export read index for rb color requests
 always @(posedge sclk)
  begin
     if(srst)
      q_rb0_read_index <= 8'b0;
     else if(service_rb0)
      q_rb0_read_index <= rb0_read_index;
  end

 always @(posedge sclk)
  begin
     if(srst)
      q_rb1_read_index <= 8'b0;
     else if(service_rb1)
      q_rb1_read_index <= rb1_read_index;
  end

 always @(posedge sclk)
  begin
     if(srst)
      q_rb2_read_index <= 8'b0;
     else if(service_rb2)
```

```
      q_rb2_read_index <= rb2_read_index;
  end

 always @(posedge sclk)
  begin
     if(srst)
      q_rb3_read_index <= 8'b0;
     else if(service_rb3)
      q_rb3_read_index <= rb3_read_index;
  end


 skid_buff_top #(9,8) rb0_index_buff(
                                        .write_rts(rb0_sx_index_send),
                                        .write_rtr(sx_to_rb0_index_rtr),
                                        .write_data(rb0_index),
                                        .read_rts(rb0_read_req),
                                        .read_rtr(service_rb0),
                                        .read_data(rb0_read_index),
                                        .clk(sclk),
                                        .reset(srst)
                                        );

 skid_buff_top #(9,8) rb1_index_buff(
                                        .write_rts(rb1_sx_index_send),
                                   .write_rtr(sx_to_rb1_index_rtr),
                                   .write_data(rb1_index),
                                   .read_rts(rb1_read_req),
                                   .read_rtr(service_rb1),
                                   .read_data(rb1_read_index),
                                   .clk(sclk),
                                   .reset(srst)
                                   );
skid_buff_top #(9,8) rb2_index_buff(
                                   .write_rts(rb2_sx_index_send),
                                   .write_rtr(sx_to_rb2_index_rtr),
                                   .write_data(rb2_index),
                                   .read_rts(rb2_read_req),
                                   .read_rtr(service_rb2),
                                   .read_data(rb2_read_index),
                                   .clk(sclk),
                                   .reset(srst)
                                   );
skid_buff_top #(9,8) rb3_index_buff(
                                   .write_rts(rb3_sx_index_send),
                                   .write_rtr(sx_to_rb3_index_rtr),
                                   .write_data(rb3_index),
                                   .read_rts(rb3_read_req),
                                   .read_rtr(service_rb3),
                                   .read_data(rb3_read_index),
```

```
                                   .clk(sclk),
                                   .reset(srst)
                                   );

  //------------------------------------------------------------------------
  //round robin arbriter (with a timer) to service read requests from each
  //RB client into Export Buffers
  //---------------------------------------------------------------------------

  reg [2:0]  read_state, next_read_state;
  reg        rb0_timeup, rb1_timeup, rb2_timeup, rb3_timeup;
  reg      clipper_timeup;

`define READ_IDLE      3'b000
`define GRANT_CLIPPER  3'b001
`define GRANT_RB0      3'b010
`define GRANT_RB1      3'b011
`define GRANT_RB2      3'b100
`define GRANT_RB3      3'b101

  always @(posedge sclk)
    begin
       if(srst)
         read_state <= `READ_IDLE;
       else
```

```
         read_state <= next_read_state;
    end

wire clipp_count_inc;
wire rb0_count_inc;
wire rb1_count_inc;
wire rb2_count_inc;
wire rb3_count_inc;

assign clipp_count_inc = (clipper_grant_count != 2'b00) |service_clipper;
assign rb0_count_inc = (rb0_grant_count != 2'b00) |service_rb0;
assign rb1_count_inc = (rb1_grant_count != 2'b00) |service_rb1;
assign rb2_count_inc = (rb2_grant_count != 2'b00) |service_rb2;
assign rb3_count_inc = (rb3_grant_count != 2'b00) |service_rb3;

//time-out request service counter
always @(posedge sclk)
  begin
     if(srst)
       begin
          rb0_grant_count <= 2'b0;
          rb1_grant_count <= 2'b0;
          rb2_grant_count <= 2'b0;
          rb3_grant_count <= 2'b0;
          clipper_grant_count <= 2'b0;
```

```
         end
       else
         begin
            if(clipp_count_inc)
              clipper_grant_count <= clipper_grant_count + 1;
            if(rb0_count_inc)
              rb0_grant_count <= rb0_grant_count + 1;
            if(rb1_count_inc)
              rb1_grant_count <= rb1_grant_count + 1;
            if(rb2_count_inc)
              rb2_grant_count <= rb2_grant_count + 1;
            if(rb3_count_inc)
              rb3_grant_count <= rb3_grant_count + 1;
         end // else: !if(srst)
    end // always @ (posedge sclk)


  always @(posedge sclk)
    begin
       if(srst)
         begin
            q_rb0_grant_count <= 2'b0;
            q_rb1_grant_count <= 2'b0;
            q_rb2_grant_count <= 2'b0;
            q_rb3_grant_count <= 2'b0;
            q_clipp_grant_count <= 2'b0;
            q_exp_read_pointer <= 9'b0;
            q_exp_buff_read_en <= 1'b0;
            q_read_valid_clipp <= 1'b0;
            q_read_valid_rb0 <= 1'b0;
            q_read_valid_rb1 <= 1'b0;
            q_read_valid_rb2 <= 1'b0;
            q_read_valid_rb3 <= 1'b0;
          end
        else
          begin
            q_rb0_grant_count <= rb0_grant_count;
            q_rb1_grant_count <= rb1_grant_count;
            q_rb2_grant_count <= rb2_grant_count;
            q_rb3_grant_count <= rb3_grant_count;
            q_clipp_grant_count <= clipper_grant_count;
            q_exp_read_pointer <= exp_read_pointer;
            q_exp_buff_read_en <= exp_buff_read_en;
            q_read_valid_clipp <= read_valid_clipp;
            q_read_valid_rb0 <= read_valid_rb0;
            q_read_valid_rb1 <= read_valid_rb1;
            q_read_valid_rb2 <= read_valid_rb2;
            q_read_valid_rb3 <= read_valid_rb3;
          end // else: !if(srst)
end // always @ (posedge sclk)
```

```
always @(/*AUTOSENSE*/`GRANT_CLIPPER or `GRANT_RB0 or `GRANT_RB1
            or `GRANT_RB2 or `GRANT_RB3 or `READ_IDLE
            or clipp_outstanding_req or clipper_timeup
            or rb0_color_ready or rb0_timeup or rb1_color_ready
            or rb1_timeup or rb2_color_ready or rb2_timeup
            or rb3_color_ready or rb3_timeup or read_state)
  begin
        case(read_state)
          `READ_IDLE:
            begin
               if(clipp_outstanding_req & clipper_timeup)
                     next_read_state = `GRANT_CLIPPER;
               else if(rb0_color_ready & rb0_timeup)
                     next_read_state = `GRANT_RB0;
               else if(rb1_color_ready & rb1_timeup)
                     next_read_state = `GRANT_RB1;
               else if(rb2_color_ready & rb2_timeup)
                     next_read_state = `GRANT_RB2;
               else if(rb3_color_ready & rb3_timeup)
                     next_read_state = `GRANT_RB3;
               else
                     next_read_state = `READ_IDLE;
```

```
            end
          `GRANT_CLIPPER:
            begin
               if(clipp_outstanding_req & clipper_timeup)
                     next_read_state = `GRANT_CLIPPER;
               else if(rb0_color_ready & rb0_timeup)
                     next_read_state = `GRANT_RB0;
               else if(rb1_color_ready & rb1_timeup)
                     next_read_state = `GRANT_RB1;
               else if(rb2_color_ready & rb2_timeup)
                     next_read_state = `GRANT_RB2;
               else if(rb3_color_ready & rb3_timeup)
                     next_read_state = `GRANT_RB3;
               else
                     next_read_state = `READ_IDLE;
            end
          `GRANT_RB0:
            begin
               if(clipp_outstanding_req & clipper_timeup)
                     next_read_state = `GRANT_CLIPPER;
               else if(rb1_color_ready & rb1_timeup)
                     next_read_state = `GRANT_RB1;
               else if(rb2_color_ready & rb2_timeup)
                     next_read_state = `GRANT_RB2;
               else if(rb3_color_ready & rb3_timeup)
```

```
                     next_read_state = `GRANT_RB3;
               else
                     next_read_state = `READ_IDLE;
            end
          `GRANT_RB1:
            begin
               if(clipp_outstanding_req & clipper_timeup)
                     next_read_state = `GRANT_CLIPPER;
               else if(rb2_color_ready)
                     next_read_state = `GRANT_RB2;
               else if(rb3_color_ready)
                     next_read_state = `GRANT_RB3;
               else if(rb0_color_ready)
                     next_read_state = `GRANT_RB0;
               else
                     next_read_state = `READ_IDLE;
            end
          `GRANT_RB2:
            begin
               if(clipp_outstanding_req & clipper_timeup)
                     next_read_state = `GRANT_CLIPPER;
               else if(rb3_color_ready & rb3_timeup)
                     next_read_state = `GRANT_RB3;
               else if(rb0_color_ready & rb0_timeup)
                     next_read_state = `GRANT_RB0;
            else if(rb1_color_ready & rb1_timeup)
                next_read_state = `GRANT_RB1;
            else
                next_read_state = `READ_IDLE;
          end
        `GRANT_RB3:
          begin
            if(clipp_outstanding_req & clipper_timeup)
                next_read_state = `GRANT_CLIPPER;
            else if(rb0_color_ready & rb0_timeup)
                next_read_state = `GRANT_RB0;
            else if(rb1_color_ready & rb1_timeup)
                next_read_state = `GRANT_RB1;
            else if(rb2_color_ready & rb2_timeup)
                next_read_state = `GRANT_RB2;
            else
                next_read_state = `READ_IDLE;
          end
        default:
          begin
            next_read_state = `READ_IDLE;
          end
      endcase // case(read_state)
   end // always @ (...

```

```

//pa_req_packet = {ipa_sp_id,ipa_offset,ipa_aux,ipa_last};
wire [7:0]  pos_exp_read_ptr;
reg [3:0]   pos_exp_read_offset;

//creating the read pointer for clipper request
//0x40 base offset
//pos_exp_read_offset : 4 entry buffer offset (possible values are 0,4,8,C)
//pa_req_control[3:2] == pa_sx_offset at the interface ...an offset within the 4-entry buffer

//last request for the current buffer
//increment the pointer to the next buffer
wire        pa_req_buff_last = q_pa_req_control[0] ;

//updating pos_exp_read_offset
//depending on the mode of export the offset increase can be by 4 or 8
//when an auxiliary buffer is present increment by 8
//when no auxiliary buffer ...increment by 4
always @(posedge sclk)
  begin
       if(srst)
         begin
            pos_exp_read_offset <= 4'b0;
         end
       else
```

```
        begin
           if(pa_req_buff_last)  //ipa_last...move to the next buffer
             pos_exp_read_offset <= pos_exp_read_offset + 4;
        end
   end // always @ (posedge sclk)

   wire        pa_req_aux;
   assign      pa_req_aux = q_pa_req_control[1];

   wire [1:0]  pa_req_offset;
   assign      pa_req_offset =q_pa_req_control[3:2];

   //0x40 is the bottom of the position export buffer
   //in a single (no aux buffer mode) there can be 4 buffers present in the position export buffer
   //at the following offsets : 0x40 , 0x44 , 0x48, 0x4c
   //otherwise only two buffers can fit in...at offsets 0x40 and 0x48 with their aux buffer at 0x44
and 0x4c respectivly

   assign        pos_exp_read_ptr[6] = 1'b1; //located at offset 0x40...the upper 16 entries of the
export buffer are dedicated to position data
   assign        pos_exp_read_ptr[5:4] = 2'b00; //only 16 entries dedicated to position...use only
bits 3:0
   assign      pos_exp_read_ptr[3:0] = (pa_req_aux) ? pos_exp_read_offset + pa_req_offset + 4 :
pos_exp_read_offset + pa_req_offset;

   wire        pa_req_sp_id;
   assign      pa_req_sp_id = q_pa_req_control[4];
```

```

assign      pos_exp_read_ptr[7] = pa_req_sp_id ; //which SP is the data coming from

always @(/*AUTOSENSE*/clipper_grant_count or service_clipper)
  begin
       if(service_clipper)
         clipper_timeup = 1'b0;
       else if(~clipper_grant_count[0] & ~clipper_grant_count[1])
         clipper_timeup = 1'b1;
       else
         clipper_timeup = 1'b0;
  end

always @(/*AUTOSENSE*/rb0_grant_count or service_rb0)
   begin
       if(service_rb0)
         rb0_timeup = 1'b0;
       else if(~rb0_grant_count[0] & ~rb0_grant_count[1])
         rb0_timeup = 1'b1;
       else
         rb0_timeup = 1'b0;
  end

always @(/*AUTOSENSE*/rb1_grant_count or service_rb1)
   begin
```

```
         if(service_rb1)
           rb1_timeup = 1'b0;
         else if(~rb1_grant_count[0] & ~rb1_grant_count[1])
           rb1_timeup = 1'b1;
         else
           rb1_timeup = 1'b0;
   end

always @(/*AUTOSENSE*/rb2_grant_count or service_rb2)
    begin
         if(service_rb2)
           rb2_timeup = 1'b0;
         else if(~rb2_grant_count[0] & ~rb2_grant_count[1])
           rb2_timeup = 1'b1;
         else
           rb2_timeup = 1'b0;
   end

always @(/*AUTOSENSE*/rb3_grant_count or service_rb3)
   begin
         if(service_rb3)
           rb3_timeup = 1'b0;
         else if(~rb3_grant_count[0] & ~rb3_grant_count[1])
           rb3_timeup = 1'b1;
         else
```

```
           rb3_timeup = 1'b0;
   end

//combinational logic for the above state machine
always @(/*AUTOSENSE*/`GRANT_CLIPPER or `GRANT_RB0 or `GRANT_RB1
             or `GRANT_RB2 or `GRANT_RB3 or `READ_IDLE
             or pos_exp_read_ptr or rb0_read_index or rb1_read_index
             or rb2_read_index or rb3_read_index or read_state)
   begin
         case(read_state)
           `READ_IDLE:
             begin
                service_rb0 = 1'b0;
                service_rb1 = 1'b0;
                service_rb2 = 1'b0;
                service_rb3 = 1'b0;
                service_clipper = 1'b0;
                exp_read_pointer = 9'b0;
                exp_buff_read_en = 1'b0;
                read_valid_clipp = 1'b0;
                read_valid_rb0 = 1'b0;
                read_valid_rb1 = 1'b0;
                read_valid_rb2 = 1'b0;
                read_valid_rb3 = 1'b0;
             end
```

```
           `GRANT_CLIPPER:
             begin
                service_rb0 = 1'b0;
                service_rb1 = 1'b0;
                service_rb2 = 1'b0;
                service_rb3 = 1'b0;
                service_clipper = 1'b1;
                exp_read_pointer =  pos_exp_read_ptr; //clipper base address ???? ANDI
                exp_buff_read_en = 1'b1;
                read_valid_clipp = 1'b1;
                read_valid_rb0 = 1'b0;
                read_valid_rb1 = 1'b0;
                read_valid_rb2 = 1'b0;
                read_valid_rb3 = 1'b0;
             end
           `GRANT_RB0:
             begin
                service_rb0 = 1'b1;
                service_rb1 = 1'b0;
                service_rb2 = 1'b0;
                service_rb3 = 1'b0;
                service_clipper = 1'b0;
                exp_read_pointer = rb0_read_index;
                exp_buff_read_en = 1'b1;
                read_valid_clipp = 1'b0;
```

```
                read_valid_rb0 = 1'b1;
                read_valid_rb1 = 1'b0;
                read_valid_rb2 = 1'b0;
                read_valid_rb3 = 1'b0;
             end
           `GRANT_RB1:
             begin
                service_rb0 = 1'b0;
                service_rb1 = 1'b1;
                service_rb2 = 1'b0;
                service_rb3 = 1'b0;
                service_clipper = 1'b0;
                exp_read_pointer = rb1_read_index;
                exp_buff_read_en = 1'b1;
                read_valid_clipp = 1'b0;
                read_valid_rb0 = 1'b0;
                read_valid_rb1 = 1'b1;
                read_valid_rb2 = 1'b0;
                read_valid_rb3 = 1'b0;
             end
           `GRANT_RB2:
             begin
                service_rb0 = 1'b0;
                service_rb1 = 1'b0;
                service_rb2 = 1'b1;
```

```
         service_rb3 = 1'b0;
         service_clipper = 1'b0;
         exp_read_pointer = rb2_read_index;
         exp_buff_read_en = 1'b1;
         read_valid_clipp = 1'b0;
         read_valid_rb0 = 1'b0;
         read_valid_rb1 = 1'b0;
         read_valid_rb2 = 1'b1;
         read_valid_rb3 = 1'b0;
        end
       `GRANT_RB3:
        begin
         service_rb0 = 1'b0;
         service_rb1 = 1'b0;
         service_rb2 = 1'b0;
         service_rb3 = 1'b1;
         service_clipper = 1'b0;
         exp_read_pointer = rb3_read_index;
         exp_buff_read_en = 1'b1;
         read_valid_clipp = 1'b0;
         read_valid_rb0 = 1'b0;
         read_valid_rb1 = 1'b0;
         read_valid_rb2 = 1'b0;
         read_valid_rb3 = 1'b1;
        end
```

```
     default:
      begin
        service_rb0 = 1'b0;
        service_rb1 = 1'b0;
        service_rb2 = 1'b0;
        service_rb3 = 1'b0;
        service_clipper = 1'b0;
        exp_read_pointer = 9'b0;
        exp_buff_read_en = 1'b0;
        read_valid_clipp = 1'b0;
        read_valid_rb0 = 1'b0;
        read_valid_rb1 = 1'b0;
        read_valid_rb2 = 1'b0;
        read_valid_rb3 = 1'b0;
      end
    endcase // case(read_state)
 end


always@(posedge sclk)
 begin
    if(srst)
     begin
       osx_rb0_index_rtr <= 1'b0;
       osx_rb1_index_rtr <= 1'b0;
```

```
        osx_rb2_index_rtr <= 1'b0;
        osx_rb3_index_rtr <= 1'b0;
       end
      else
       begin
        osx_rb0_index_rtr <= sx_to_rb0_index_rtr;
        osx_rb1_index_rtr <= sx_to_rb1_index_rtr;
        osx_rb2_index_rtr <= sx_to_rb2_index_rtr;
        osx_rb3_index_rtr <= sx_to_rb3_index_rtr;
       end // else: !if(srst)
  end // always@ (posedge sclk)

 assign sx_rb0_index_rtr = osx_rb0_index_rtr;
 assign sx_rb1_index_rtr = osx_rb1_index_rtr;
 assign sx_rb2_index_rtr = osx_rb2_index_rtr;
 assign sx_rb3_index_rtr = osx_rb3_index_rtr;



endmodule // export_control
```

```
//                              -*- Mode: Verilog -*-
// Filename        : macc.v
// Description       : Verilog wrapper for the MACC unit which includes the rest of the ALU
opcodes
// Author          : Andi Skende
// Created On      : Mon Oct  8 15:54:00 2001
// Last Modified By: .
// Last Modified On: .
// Update Count    : 0
// Status          : Unknown, Use with caution!

 module macc(/*AUTOARG*/
   // Outputs
   oResult, oScalarInput, oScalarOpcode, oExportDst,
   // Inputs
   iRegData, iConstantData, iScalarData, iInstruction, iInstStart,
   sclk, srst
   );



   //-----------------------------------------------------------------------------
   //These inputs represent all the possible sources from where the MACC arguments
   //can be chosen from
   //-----------------------------------------------------------------------------
   input [127:0]  iRegData, iConstantData, iScalarData;

```

```
//-----------------------------------------------------------------------------
//this bus represents the ALU instruction word comming from the Sequencer
//at different cycles withing the 4 cycle window, the bus content represents
//different informantion as shown below.
//Instruction word. The argument select part of the
//instruction is send over through four cycles since
//we only need one instruction every four cycles
// cycle 0:SRC A Select
//         SRC A Argument Modifier
//         SRC A Swizzle
// cycle 1:SRC B Select
//         SRC B Argument Modifier
//         SRC B Swizzle
// cycle 2:SRC C Select
//         SRC C Argument Modifier
//         SRC C Swizzle
// cycle 3:Vector Opcode
//         Scalar Opcode
//         Vector Clamp
//         Scalar Clamp
//         Vector Write Mask
//         Scalar Write Mask
//-------------------------------------------------------------------------------

input [20:0]   iInstruction;
```

```
   input [0:0]      iInstStart;
   input              sclk;
   input          srst;

   //-----------------------------------------------------------------------------
   output [127:0] oResult;
   output [31:0]  oScalarInput;
   output [5:0]   oScalarOpcode;
   output [5:0]   oExportDst;      //represents the destination pointer for exports

   //ALU opcodes declared as parameters
   //this definition is subject to change as more
   //opcodes are added. for the latest definition
   //please refer to Shader Pipe Spec: ALU instruction definition
   parameter [4:0] ADD = 5'h00,
                          MUL = 5'h01,
                          MAX = 5'h02,
                          MIN = 5'h03,
                          SETE = 5'h04,
                          SETGT = 5'h05,
                          SETGE = 5'h06,
                          SETNE = 5'h07,
                          FRACT = 5'h08,
                          TRUNC = 5'h09,
                          FLOOR = 5'h0a,
```

```
                          MULADD = 5'h0b,
                          CNDE = 5'h0c,
                          CNDGE = 5'h0d,
                          CNDGT = 5'h0e;

//registering four sets of data from four different instruction transfer cycles
//---------------------------------------------------------------------------------
reg [20:0]      q_Instruction0, q_Instruction1, q_Instruction2, q_Instruction3;

wire [4:0]      VectorOpcode;
wire [5:0]    ScalarOpcode;
wire [0:0]      VectorClamp,ScalarClamp;
wire [3:0]      VectorWriteMask,ScalarWriteMask;

wire [31:0]    MaccResult,ScalarResult;
reg [31:0]      MaccResultClamp;
reg [31:0]      q0_MaccResultClamp, q1_MaccResultClamp , q2_MaccResultClamp;
reg [31:0]      ResultMin;
wire [31:0]    ResultMaxMin;
reg [31:0]      ResultMax;
reg [31:0]      q0_ResultMaxMin, q1_ResultMaxMin, q2_ResultMaxMin, q3_ResultMaxMin;

//--------------------------------------------------------------------------------------/
//represents the previous instruction vector result
//--------------------------------------------------------------------------------------/
```

```
  wire [127:0]  VectorData;

  reg  [127:0]  InputDataA,InputDataB,InputDataC;
  reg  [127:0]  InputData;

  //select logic controls
  //-------------------------------------------------------------------------------------------
  wire [0:0]          ResultClamp;
  wire [2:0]          SrcASel,SrcBSel,SrcCSel;          //select Reg vs. Vector vs. Scalar vs. Constant
  wire [0:0]          SrcANegate, SrcBNegate, SrcCNegate;     // input argument modifiers
  wire [1:0]          SrcARedSwizzle,     SrcAGreenSwizzle,     SrcABlueSwizzle, SrcAAlphaSwizzle;
  wire [1:0]          SrcBRedSwizzle,     SrcBGreenSwizzle,     SrcBBlueSwizzle, SrcBAlphaSwizzle;
  wire [1:0]          SrcCRedSwizzle,     SrcCGreenSwizzle,     SrcCBlueSwizzle, SrcCAlphaSwizzle;
  reg    [0:0]        decode_SrcA, decode_SrcB, decode_SrcC,decode_Opcode;

  reg [31:0]          SrcAAlphaBus, SrcARedBus, SrcAGreenBus, SrcABlueBus;
  reg [31:0]          SrcBAlphaBus, SrcBRedBus, SrcBGreenBus, SrcBBlueBus;
  reg [31:0]          SrcCAlphaBus, SrcCRedBus, SrcCGreenBus, SrcCBlueBus;
  reg [31:0]          SrcAAlphaBusLatch1,     SrcARedBusLatch0,SrcARedBusLatch1, SrcAGreenBusLatch0,SrcAGreenBusLatch1, SrcABlueBusLatch0,SrcABlueBusLatch1 ;
  reg [31:0]          SrcBGreenBusLatch0,    SrcBGreenBusLatch1,   SrcBBlueBusLatch0, SrcBBlueBusLatch1, SrcBRedBusLatch1, SrcBAlphaBusLatch1;
  reg [31:0]          SrcCBlueBusLatch0,     SrcCBlueBusLatch1,     SrcCGreenBusLatch1, SrcCRedBusLatch1, SrcCAlphaBusLatch1;
```

```
  reg [31:0]          OperandA,    OperandB,    OperandC,OperandAMod,    OperandBMod, OperandCMod;

  //-----------------------------------------------------------------------------------------------------------
  //this state machine controls the four cycle GPR read/write and source sampling sequence
  //state machine related variables
  reg [1:0]           alu_state, next_alu_state;
  parameter           Opcode     = 2'b11;
  parameter           SrcA       = 2'b00;
  parameter           SrcB       = 2'b01;
  parameter           SrcC       = 2'b10;
  parameter        ONE        = 32'h3f800000;
  parameter           ZERO       = 32'h00000000;
  parameter        GT_ONE_EXP = 8'h7f;

  //-------------------------------------------------------------------------------------------------
  //state machine implementation
  //This state machine has four states
  //InstStart      :  first or default state
  //SrcA           : SrcA is sampled, selecled and swizzled on its way to the first level of latches
  //SrcB           : SrcB is sampled, selected and swizzled on its way to the first level of latches
  //SrcC           : SrcC is sampled, selected and swizzled on its way to the first level of latches
  //-------------------------------------------------------------------------------------------------


  //--------------------------------------
```

```
  //registering the iInstStart event
  //-------------------------------------
  reg                   instruct_issued;

  always @(posedge sclk)
    begin
        if(srst)
          begin
            instruct_issued <= 1'b0;
          end
        else if(iInstStart)
          begin
            instruct_issued <= 1'b1;
          end
    end // always @ (posedge sclk)


  //------------------------------------------------------

  always @(posedge sclk)
    begin
        if(srst)
          alu_state <= SrcA;
        else
          alu_state <= next_alu_state;
```

```
    end

  always @(alu_state or iInstStart)
    begin
        case(alu_state)
          SrcA:
            begin
              if(iInstStart | instruct_issued)
                  next_alu_state = SrcB;
              else
                  next_alu_state = SrcA;
            end
          SrcB :
            next_alu_state = SrcC;
          SrcC:
            next_alu_state = Opcode;
          Opcode:
            next_alu_state = SrcA;
        endcase
    end // always @ (state)

  //---------------------------------------------------------------------------------------------------
  //the decode_* signals are used throughout the logic to mark the current state of the state machine at
  //any cycle out of the 4 cycle window from one instruction start to another
  //---------------------------------------------------------------------------------------------------
```

```

always @(alu_state)
  begin
      case (alu_state)
        SrcA:
          begin
            decode_SrcA = 1'b1;
            decode_SrcB = 1'b0;
            decode_SrcC = 1'b0;
            decode_Opcode = 1'b0;
          end
        SrcB:
          begin
            decode_SrcA = 1'b0;
            decode_SrcB = 1'b1;
            decode_SrcC = 1'b0;
            decode_Opcode = 1'b0;
          end
        SrcC:
          begin
            decode_SrcA = 1'b0;
            decode_SrcB = 1'b0;
            decode_SrcC = 1'b1;
            decode_Opcode = 1'b0;
          end
```

```
        Opcode:
          begin
            decode_SrcA = 1'b0;
            decode_SrcB = 1'b0;
            decode_SrcC = 1'b0;
            decode_Opcode = 1'b1;
          end
      endcase
  end


//-----------------------------------------------------------------------
//----------------------------------------------------------------------
//Registering the Instruction word (20 bits) in four consecutive cycles
//---------------------------------------------------------------------
always@(posedge sclk)
  if(srst)
    q_Instruction0 <= 21'b0;
  else if(decode_SrcA)
    q_Instruction0 <= iInstruction;

always@(posedge sclk)
  if(srst)
    q_Instruction1 <= 21'b0;
  else if(decode_SrcB)
```

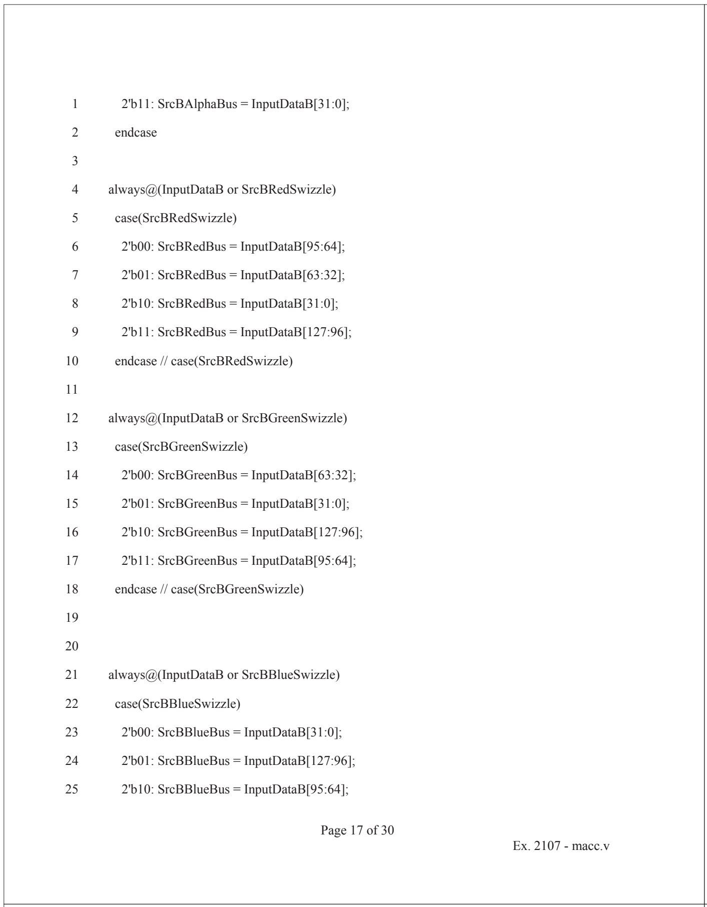
```
    q_Instruction1 <= iInstruction;

 always@(posedge sclk)
  if(srst)
    q_Instruction2 <= 21'b0;
  else if(decode_SrcC)
    q_Instruction2 <= iInstruction;

always@(posedge sclk)
  if(srst)
    q_Instruction3 <= 21'b0;
  else if(decode_Opcode)
    q_Instruction3 <= iInstruction;

//grabing the export destination ID.
//If we are dealing with an export instruction...this value identifies which
//attribute is being exported ...please refer to the shader pipe spec for more details
//on this
//---------------------------------------------
assign oExportDst = q_Instruction0[17:12];

//-----------------------------------------------------------------------
//decoding the instruction word into a set of select/modify signals used
//for argument selection and input modification on the way to MACC unit
//----------------------------------------------------------------------
```

```

assign SrcASel = q_Instruction0[2:0];
assign SrcANegate = q_Instruction0[3:3];
assign SrcAAlphaSwizzle = q_Instruction0[11:10];
assign SrcARedSwizzle = q_Instruction0[5:4];
assign SrcAGreenSwizzle = q_Instruction0[7:6];
assign SrcABlueSwizzle = q_Instruction0[9:8];

assign SrcBSel = q_Instruction1[2:0];
assign SrcBNegate = q_Instruction1[3:3];
assign SrcBAlphaSwizzle = q_Instruction1[11:10];
assign SrcBRedSwizzle = q_Instruction1[5:4];
assign SrcBGreenSwizzle = q_Instruction1[7:6];
assign SrcBBlueSwizzle = q_Instruction1[9:8];


assign SrcCSel = q_Instruction2[2:0];
assign SrcCNegate = q_Instruction2[3:3];
assign SrcCAlphaSwizzle = q_Instruction2[11:10];
assign SrcCRedSwizzle = q_Instruction2[5:4];
assign SrcCGreenSwizzle = q_Instruction2[7:6];
assign SrcCBlueSwizzle = q_Instruction2[9:8];
assign VectorOpcode = q_Instruction3[4:0];
assign ScalarOpcode = q_Instruction3[10:5];
assign VectorClamp = q_Instruction3[11:11];
assign ScalarClamp = q_Instruction3[12:12];
assign VectorWriteMask = q_Instruction3[16:13];
assign ScalarWriteMask = q_Instruction3[20:17];

//-----------------------------------------------------------------------------
//Argument Selectin for the three source operands going into the MACC unit
//All information required for the selection logic in embedded into the ALU
//Instrution Word. Please refer to the Shade Processor Spec for a delailed
//definition of the select fields for the three sources
//-----------------------------------------------------------------------------

always@(SrcASel or iConstantData or iRegData or VectorData or iScalarData)
 begin
     case(SrcASel)
      3'b000 : InputDataA = iConstantData;
      3'b100 : InputDataA = iRegData;
      3'b101 : InputDataA = iRegData;
      3'b110 : InputDataA = VectorData;
      3'b111 : InputDataA = iScalarData;
      default: InputDataA = iRegData;
     endcase // case(SrcASel)
 end // always@ (SrcASel or iConstantData or iRegData or iVectorData or iScalarData)
```

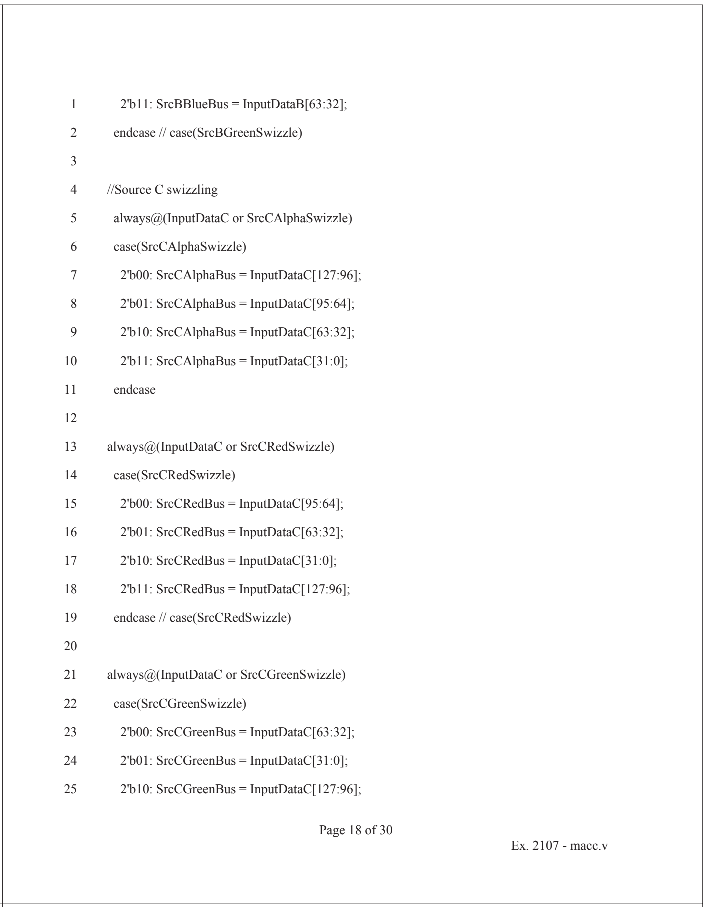
```

always@(SrcBSel or iConstantData or iRegData or VectorData or iScalarData)
 begin
     case(SrcBSel)
      3'b000 : InputDataB = iConstantData;
      3'b100 : InputDataB = iRegData;
      3'b101 : InputDataB = iRegData;
      3'b110 : InputDataB = VectorData;
      3'b111 : InputDataB = iScalarData;
      default: InputDataB = iRegData;
     endcase // case(SrcBSel)
 end // always@ (SrcBSel or iConstantData or iRegData or iVectorData or iScalarData)

always@(SrcCSel or iConstantData or iRegData or VectorData or iScalarData)
 begin
     case(SrcCSel)
      3'b000 : InputDataC = iConstantData;
      3'b100 : InputDataC = iRegData;
      3'b101 : InputDataC = iRegData;
      3'b110 : InputDataC = VectorData;
      3'b111 : InputDataC = iScalarData;
      default: InputDataC = iRegData;
     endcase // case(SrcCSel)
 end // always@ (SrcCSel or iConstantData or iRegData or iVectorData or iScalarData)
//-------------------------------------------------------------------------------------
```

```


//-------------------------------------------------------------------------------------
//Input Modifiers ie. swizzle and negate are begin applied
//-------------------------------------------------------------------------------------

//Source A swizzling

always@(InputDataA or SrcAAlphaSwizzle)
 case(SrcAAlphaSwizzle)
   2'b00: SrcAAlphaBus = InputDataA[127:96];
   2'b01: SrcAAlphaBus = InputDataA[95:64];
   2'b10: SrcAAlphaBus = InputDataA[63:32];
   2'b11: SrcAAlphaBus = InputDataA[31:0];
 endcase // case(SrcAAlphaSwizzle)


always@(InputDataA or SrcARedSwizzle)
 case(SrcARedSwizzle)
   2'b00: SrcARedBus = InputDataA[95:64];
   2'b01: SrcARedBus = InputDataA[63:32];
   2'b10: SrcARedBus = InputDataA[31:0];
   2'b11: SrcARedBus = InputDataA[127:96];
 endcase // case(SrcARedSwizzle)

```

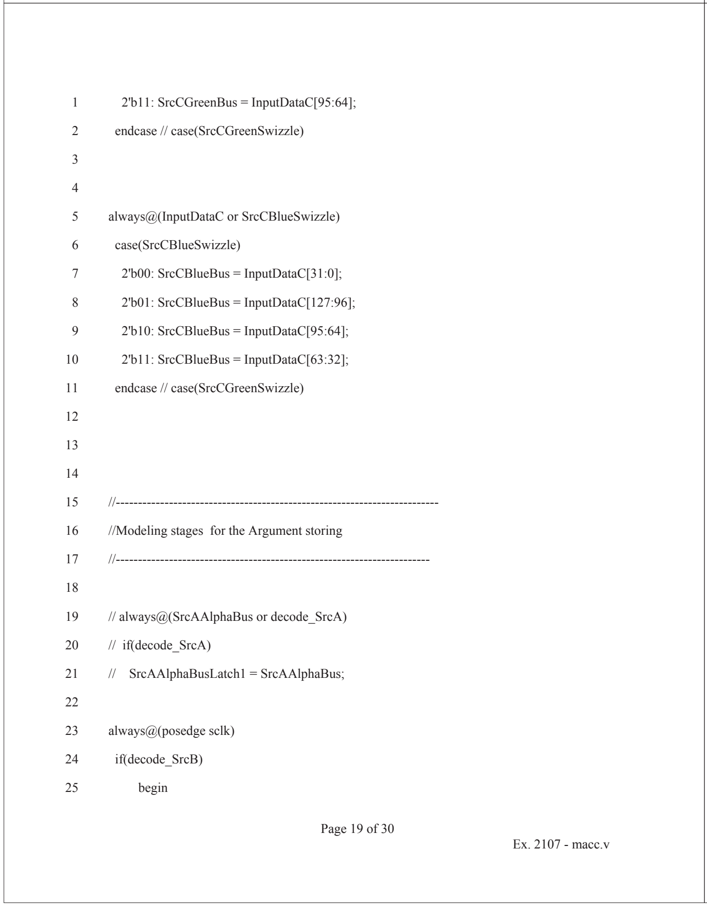
```


always@(InputDataA or SrcAGreenSwizzle)
 case(SrcAGreenSwizzle)
   2'b00: SrcAGreenBus = InputDataA[63:32];
   2'b01: SrcAGreenBus = InputDataA[31:0];
   2'b10: SrcAGreenBus = InputDataA[127:96];
   2'b11: SrcAGreenBus = InputDataA[95:64];
 endcase // case(SrcAGreenSwizzle)

always@(InputDataA or SrcABlueSwizzle)
 case(SrcABlueSwizzle)
   2'b00: SrcABlueBus = InputDataA[31:0];
   2'b01: SrcABlueBus = InputDataA[127:96];
   2'b10: SrcABlueBus = InputDataA[95:64];
   2'b11: SrcABlueBus = InputDataA[63:32];
 endcase // case(SrcAGreenSwizzle)

//Source B swizzling

always@(InputDataB or SrcBAlphaSwizzle)
 case(SrcBAlphaSwizzle)
   2'b00: SrcBAlphaBus = InputDataB[127:96];
   2'b01: SrcBAlphaBus = InputDataB[95:64];
   2'b10: SrcBAlphaBus = InputDataB[63:32];
```



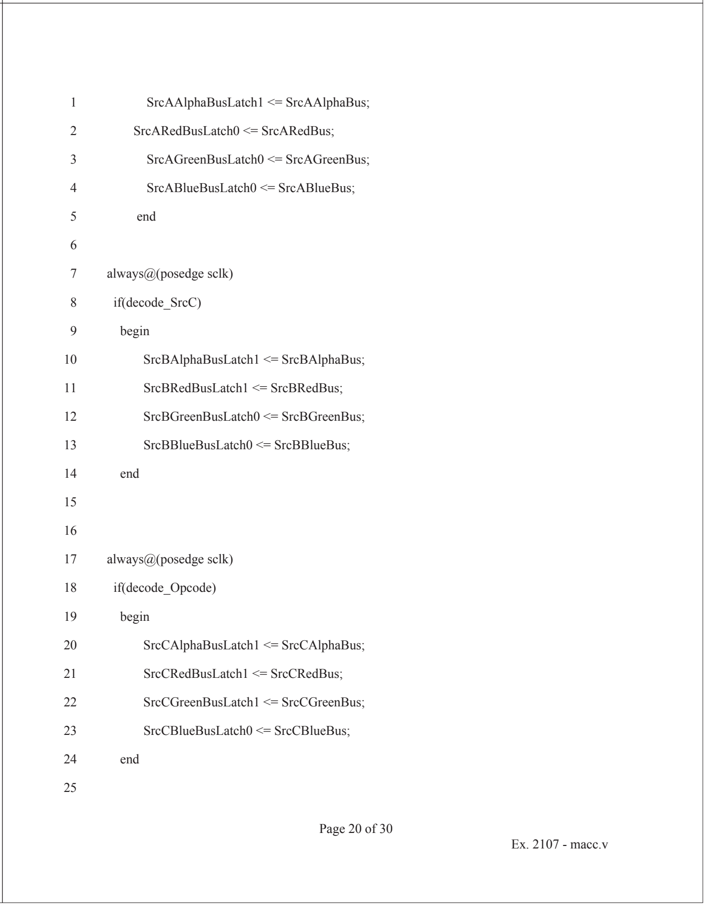
```
      2'b11: SrcBAlphaBus = InputDataB[31:0];
    endcase

  always@(InputDataB or SrcBRedSwizzle)
    case(SrcBRedSwizzle)
      2'b00: SrcBRedBus = InputDataB[95:64];
      2'b01: SrcBRedBus = InputDataB[63:32];
      2'b10: SrcBRedBus = InputDataB[31:0];
      2'b11: SrcBRedBus = InputDataB[127:96];
    endcase // case(SrcBRedSwizzle)

  always@(InputDataB or SrcBGreenSwizzle)
    case(SrcBGreenSwizzle)
      2'b00: SrcBGreenBus = InputDataB[63:32];
      2'b01: SrcBGreenBus = InputDataB[31:0];
      2'b10: SrcBGreenBus = InputDataB[127:96];
      2'b11: SrcBGreenBus = InputDataB[95:64];
    endcase // case(SrcBGreenSwizzle)


  always@(InputDataB or SrcBBlueSwizzle)
    case(SrcBBlueSwizzle)
      2'b00: SrcBBlueBus = InputDataB[31:0];
      2'b01: SrcBBlueBus = InputDataB[127:96];
      2'b10: SrcBBlueBus = InputDataB[95:64];
```



```
      2'b11: SrcBBlueBus = InputDataB[63:32];
    endcase // case(SrcBGreenSwizzle)

  //Source C swizzling
    always@(InputDataC or SrcCAlphaSwizzle)
    case(SrcCAlphaSwizzle)
      2'b00: SrcCAlphaBus = InputDataC[127:96];
      2'b01: SrcCAlphaBus = InputDataC[95:64];
      2'b10: SrcCAlphaBus = InputDataC[63:32];
      2'b11: SrcCAlphaBus = InputDataC[31:0];
    endcase

  always@(InputDataC or SrcCRedSwizzle)
    case(SrcCRedSwizzle)
      2'b00: SrcCRedBus = InputDataC[95:64];
      2'b01: SrcCRedBus = InputDataC[63:32];
      2'b10: SrcCRedBus = InputDataC[31:0];
      2'b11: SrcCRedBus = InputDataC[127:96];
    endcase // case(SrcCRedSwizzle)

  always@(InputDataC or SrcCGreenSwizzle)
    case(SrcCGreenSwizzle)
      2'b00: SrcCGreenBus = InputDataC[63:32];
      2'b01: SrcCGreenBus = InputDataC[31:0];
      2'b10: SrcCGreenBus = InputDataC[127:96];
```



```
      2'b11: SrcCGreenBus = InputDataC[95:64];
    endcase // case(SrcCGreenSwizzle)


  always@(InputDataC or SrcCBlueSwizzle)
    case(SrcCBlueSwizzle)
      2'b00: SrcCBlueBus = InputDataC[31:0];
      2'b01: SrcCBlueBus = InputDataC[127:96];
      2'b10: SrcCBlueBus = InputDataC[95:64];
      2'b11: SrcCBlueBus = InputDataC[63:32];
    endcase // case(SrcCGreenSwizzle)



  //---------------------------------------------------------------------------
  //Modeling stages  for the Argument storing
  //-------------------------------------------------------------------------

  // always@(SrcAAlphaBus or decode_SrcA)
  //  if(decode_SrcA)
  //    SrcAAlphaBusLatch1 = SrcAAlphaBus;

  always@(posedge sclk)
    if(decode_SrcB)
          begin
```



```
            SrcAAlphaBusLatch1 <= SrcAAlphaBus;
        SrcARedBusLatch0 <= SrcARedBus;
            SrcAGreenBusLatch0 <= SrcAGreenBus;
            SrcABlueBusLatch0 <= SrcABlueBus;
         end

  always@(posedge sclk)
    if(decode_SrcC)
      begin
           SrcBAlphaBusLatch1 <= SrcBAlphaBus;
           SrcBRedBusLatch1 <= SrcBRedBus;
           SrcBGreenBusLatch0 <= SrcBGreenBus;
           SrcBBlueBusLatch0 <= SrcBBlueBus;
      end


  always@(posedge sclk)
    if(decode_Opcode)
      begin
           SrcCAlphaBusLatch1 <= SrcCAlphaBus;
           SrcCRedBusLatch1 <= SrcCRedBus;
           SrcCGreenBusLatch1 <= SrcCGreenBus;
           SrcCBlueBusLatch0 <= SrcCBlueBus;
      end
```

```
//second level of latches
always@(posedge sclk)
  if(decode_SrcA)
    begin
          SrcARedBusLatch1 <= SrcARedBusLatch0;
          SrcAGreenBusLatch1 <= SrcAGreenBusLatch0;
          SrcABlueBusLatch1 <= SrcABlueBusLatch0;
          SrcBGreenBusLatch1 <= SrcBGreenBusLatch0;
          SrcBBlueBusLatch1 <= SrcBBlueBusLatch0;
          SrcCBlueBusLatch1 <= SrcCBlueBusLatch0;
    end

//------------------------------------------------------------------------------------
// register the outputs from the latches into the MACC unit
//------------------------------------------------------------------------------------
always@(posedge sclk)
  begin
        if(decode_SrcA)
          OperandA <= SrcAAlphaBusLatch1;
        else if(decode_SrcB)
          OperandA <= SrcARedBusLatch1;
        else if(decode_SrcC)
          OperandA <= SrcAGreenBusLatch1;
        else
          OperandA <= SrcABlueBusLatch1;
```

```
  end // always@ (sclk)


always@(posedge sclk)
  begin
        if(decode_SrcA)
          OperandB <= SrcBAlphaBusLatch1;
        else if(decode_SrcB)
          OperandB <= SrcBRedBusLatch1;
        else if(decode_SrcC)
          OperandB <= SrcBGreenBusLatch1;
        else
          OperandB <= SrcBBlueBusLatch1;
  end // always@ (sclk)

always@(posedge sclk)
  begin
        if(decode_SrcA)
          OperandC <= SrcCAlphaBusLatch1;
        else if(decode_SrcB)
          OperandC <= SrcCRedBusLatch1;
        else if(decode_SrcC)
          OperandC <= SrcCGreenBusLatch1;
        else
          OperandC <= SrcCBlueBusLatch1;
```

```
  end // always@ (sclk)

//----------------------------------------------------------
//Input Modifier ....NEGATE.
//---------------------------------------------------------
always@(SrcANegate or OperandA)
  if(SrcANegate)
    OperandAMod[31:0]= {OperandA[31]^SrcANegate,OperandA[30:0]};
  else
    OperandAMod = OperandA;

always@(SrcBNegate or OperandB)
  if(SrcBNegate)
    OperandBMod[31:0]= {OperandB[31]^SrcBNegate,OperandB[30:0]};
  else
    OperandBMod = OperandB;

always@(SrcCNegate or OperandC)
  if(SrcCNegate)
    OperandCMod[31:0]= {OperandC[31]^SrcCNegate,OperandC[30:0]};
  else
    OperandCMod = OperandC;
//----------------------------------------------------------------

//------------------------------------------------------------------------------------
```

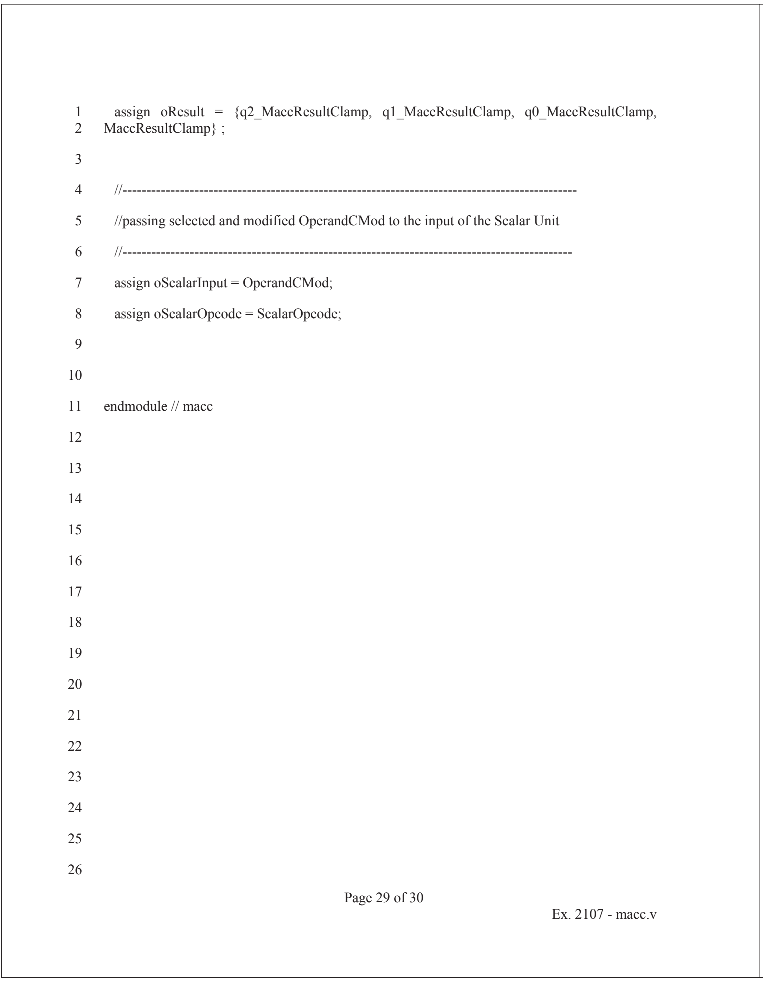
```
  wire [1:0] opcode_mux_ctl;
  reg                                                                                      [1:0]
q0_opcode_mux_ctl,q1_opcode_mux_ctl,q2_opcode_mux_ctl,q3_opcode_mux_ctl,q4_opcode
_mux_ctl;

  //generating control signals for routing the proper path into the final result
  //00 : other
  //01 : min
  //10 : max

  assign    opcode_mux_ctl[0] = (VectorOpcode == MIN) ;
  assign    opcode_mux_ctl[1] = (VectorOpcode == MAX) ;


  always @(posedge sclk)
    begin
        q0_opcode_mux_ctl <= opcode_mux_ctl;
        q1_opcode_mux_ctl <= q0_opcode_mux_ctl;
        q2_opcode_mux_ctl <= q1_opcode_mux_ctl;
        q3_opcode_mux_ctl <= q2_opcode_mux_ctl;
        q4_opcode_mux_ctl <= q3_opcode_mux_ctl;
    end


  //Floating point Multiply and Accumulate
```

```
   macc32 mad(OperandAMod, OperandBMod, OperandCMod, VectorOpcode,MaccResult,sclk);



   //--------------------------------------------------------------------------------
   //some of the opcodes do not have to be implemented via the MACC unit
   //for example : MAX can be implemented via compares of the exponents and/or mantissas of
   //the two numbers assuming that the numbers are normalized
   //this is a separate parallel pipeline from the MACC
   //--------------------------------------------------------------------------------------

   //MIN or MAX
   //revisit this logic for the case when exp = 0 ...ANDI
   always @(/*AUTOSENSE*/OperandAMod or OperandBMod)
     begin
        if(OperandAMod[30:0] >= OperandBMod[30:0])
          begin
             if(!OperandAMod[31])
               begin
                  ResultMax = OperandAMod;
                  ResultMin = OperandBMod;
               end
             else
               begin
                  ResultMax = OperandBMod;
                  ResultMin = OperandAMod;
```

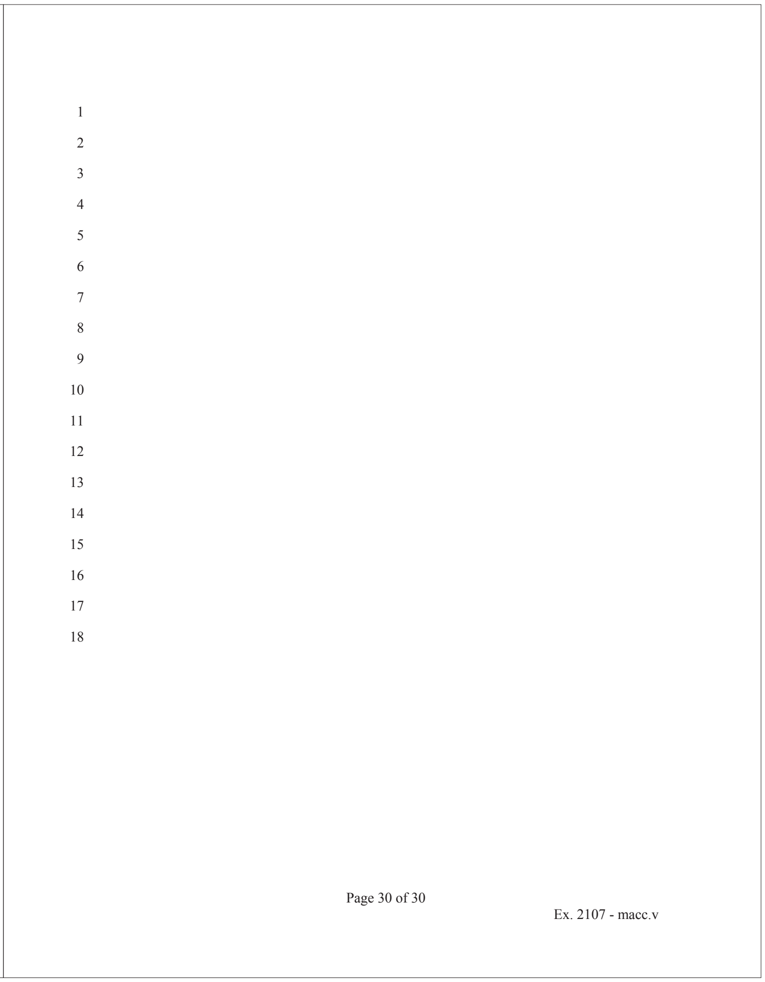
```
               end
          end // if (OperandAMod[30:0] >= OperandBMod[30:0])
        else if (OperandBMod[30:0] >= OperandAMod[30:0])
          begin
             if(!OperandBMod[31])
               begin
                  ResultMax = OperandBMod;
                  ResultMin = OperandAMod;
               end
             else
               begin
                  ResultMax = OperandAMod;
                  ResultMin = OperandBMod;
               end
          end // if (OperandBMod[30:0] >= OperandAMod[30:0])
     end // always @ (...


   //choose  MIN vs. MAX
   assign ResultMaxMin = (opcode_mux_ctl[1]) ? ResultMax : ResultMin;

   //delay the ResultMaxMin to match with the other path of the pipeline that goes through the MACC
   always@(posedge sclk)
     begin
        q0_ResultMaxMin <= ResultMaxMin;
```

```
        q1_ResultMaxMin <= q0_ResultMaxMin;
        q2_ResultMaxMin <= q1_ResultMaxMin;
        q3_ResultMaxMin <= q2_ResultMaxMin;
     end

   reg [31:0] MaccResultMux;

   //-----------------------------------------------------------------------------------------------
   //Routing the Result into MaccResultMux based on the opcode
   //-----------------------------------------------------------------------------------------------
   always @(/*AUTOSENSE*/MaccResult or q3_ResultMaxMin
            or q4_opcode_mux_ctl)
     begin
        case(q4_opcode_mux_ctl)
          2'b00: MaccResultMux = MaccResult;
          2'b01: MaccResultMux = q3_ResultMaxMin;
          2'b10: MaccResultMux = q3_ResultMaxMin;
          default : MaccResultMux = MaccResult;
        endcase // case(opcode_mux_ctl)
     end

   //-----------------------------------------------------------------------------------------------
   //Clamping the result and other output modifiers
   always@(/*AUTOSENSE*/MaccResultMux or ResultClamp)
     begin
```

```
        if(ResultClamp)
          begin
             if(MaccResultMux[31])
               MaccResultClamp = ZERO;
             else if(MaccResultMux[30:24] > GT_ONE_EXP)
            MaccResultClamp = ONE;
             else
               MaccResultClamp = MaccResultMux;
          end
        else
          MaccResultClamp = MaccResultMux;
     end // always@ (MaccResult or ResultClamp)

   //-----------------------------------------------------------------------------------------------
   //pipeline delays for the code....creating the 4 stage delay for the VectorResult
   //-----------------------------------------------------------------------------------------------
   always@(posedge sclk)
     begin
        q0_MaccResultClamp <= MaccResultClamp;
        q1_MaccResultClamp <= q0_MaccResultClamp;
        q2_MaccResultClamp <= q1_MaccResultClamp;
     end

   assign VectorData = {q2_MaccResultClamp, q1_MaccResultClamp, q0_MaccResultClamp, MaccResultClamp};
```



```
  assign  oResult  =  {q2_MaccResultClamp,  q1_MaccResultClamp,  q0_MaccResultClamp,
MaccResultClamp} ;

  //--------------------------------------------------------------------------------------------
  //passing selected and modified OperandCMod to the input of the Scalar Unit
  //--------------------------------------------------------------------------------------------
  assign oScalarInput = OperandCMod;
  assign oScalarOpcode = ScalarOpcode;


endmodule // macc
```



```
//                              -*- Mode: Verilog -*-
// Filename        : macc32.mc
// Description     : This file represents the implementation of the MACC unit (Multiply and
Accumulate)
// Author          : Andi Skende
// Created On      : Mon Jan 28 16:04:47 2002
// Last Modified By: .
// Last Modified On: .
// Update Count    : 0
// Status          : Unknown, Use with caution!

module macc32(in1,in2,in3,opcode,result);
   directive(delay = 2400,pipeline = "on");
   directive(clock = "iSCLK");
   directive(multtype="booth");
   directive(fatype = "fastcla");



   //----------------------------------------------------------------------------------------------
   //declaration of the input data as 32-bit IEEE floating point with an implicit 1 as bit 24 of the
   //mantissa
   //----------------------------------------------------------------------------------------------
   input unsigned [31:0] in1;
   input unsigned [31:0] in2;
   input unsigned [31:0] in3;
   input unsigned [4:0] opcode;
```

```
output unsigned [31:0] result;

//------------------------------------------------------------------------------
// ALU opcode list declaration
//-----------------------------------------------------------------------------
#define ADD 5'h00
#define MUL 5'h01
#define MAX 5'h02
#define MIN 5'h03
#define SETE 5'h04
#define SETGT 5'h05
#define SETGE 5'h06
#define SETNE 5'h07
#define FRACT 5'h08
#define TRUNC 5'h09
#define FLOOR 5'h0a
#define MULADD 5'h0b
#define CNDE   5'h0c
#define CNDGE  5'h0d
#define CNDGT  5'h0e

//declaring a couple of constants
//-------------------------------------------------------
wire   unsigned [0:0] one = 1'h1;
wire   unsigned [0:0] zero = 1'h0;
```

```
   wire   unsigned [31:0] one_ieee = 31'h3f800000;

   //control signals related to opcode
   //--------------------------------------------------------------------------------------------
   wire unsigned [0:0] opcode_add = (opcode == ADD) ? one :zero;
   wire unsigned [0:0] opcode_trunc = (opcode == TRUNC) ? one : zero;
   wire unsigned [0:0] opcode_fract = (opcode == FRACT) ? one : zero;
   wire unsigned [0:0] opcode_muladd = (opcode == MULADD) ? one : zero;
   wire unsigned [0:0] opcode_mul = (opcode == MUL) ? one : zero;

   //breaking the input number up into respective fields
   //------------------------------------------------------
   wire   unsigned [0:0]  sign1 = in1[31];
   wire   unsigned [0:0]  sign2 = (opcode_add) ? one_ieee[31]:in2[31];
   wire   unsigned [0:0]  sign3 = (opcode_add) ? in2[31]:in3[31];

   wire   unsigned [7:0]  exp1 =  in1[30:23];
   wire   unsigned [7:0]  exp2 =  (opcode_add) ? one_ieee[30:23]:in2[30:23];
   wire   unsigned [7:0]  exp3 =  (opcode_add) ? in2[30:23]:in3[30:23];

   wire   unsigned [23:0] mant1_one = cat(one, in1[22:0]);
   wire      unsigned   [23:0]   mant2_one   =   (opcode_add)   ?   cat(one,   one_ieee[22:0]):cat(one,
in2[22:0]);
   wire   unsigned [23:0] mant3_one = (opcode_add) ? cat(one, in2[22:0]): cat(one, in3[22:0]);


```

```
wire   signed [24:0] mant1_signed = -mant1_one;
wire   signed [24:0] mant1 = (sign1) ? mant1_signed : mant1_one ;
wire   signed [24:0] mant2_signed = -mant2_one;
wire   signed [24:0] mant2 = (sign2) ? mant2_signed : mant2_one ;
wire   signed [24:0] mant3_signed = -mant3_one;
wire   signed [24:0] mant3 = (sign3) ? mant3_signed : mant3_one ;



//---------------------------------------------------------------------------
// selection logic for all three paths exp, mantissa and sign based on the opcode
//-----------------------------------------------------------------------------
//1: muladd, mul, add
//0: fraction
wire  unsigned [0:0] opcode_mul_add = opcode_add | opcode_muladd | opcode_mul;
wire  unsigned [0:0] opcode_frac_trunc = opcode_fract | opcode_trunc;
//wire  unsigned [1:0] opcode_sel = cat( opcode_mul_add, opcode_frac_trunc);
wire  unsigned [0:0]   opcode_sel = opcode_mul_add;

//----------------------------------------------------------------------------
//calculating the exponent for in1*in2 and the exponent delta
//---------------------------------------------------------------------------

// if opcode == FRACT write the exp_delta with exp1 ...and the real delta for all
// the other cases
```

```
    wire   signed [8:0] exp1_plus_exp2 = exp1 + exp2;
    wire   signed [8:0] exp_mul = (opcode_sel) ? exp1_plus_exp2 : exp1;
    wire   signed [8:0] exp_mul_minus_exp3 = exp_mul - exp3;
    wire   signed [8:0] exp_delta =(opcode_sel) ? exp_mul_minus_exp3 : exp_mul;

    //detection of whether the number (in1) is all-fraction value (number < 1)
    wire  unsigned [0:0] fract_all_fract = (exp_delta < 127);

    wire   signed [7:0] exp_delta_unbiased = exp_delta - 127;
    wire   unsigned [7:0] exp_delta_unsigned = - exp_delta_unbiased;
    wire   unsigned [7:0] exp_delta_abs = (exp_delta_unbiased > 0)? exp_delta_unbiased: exp_delta_unsigned;
    wire   unsigned [0:0] exp_delta_gt_24 = (exp_delta_abs > 5'h18) ?  one  : zero ; //flag to avoid shifting out for more than 24 positions

    //special logic related to FRACT and TRUNC logic
    wire   unsigned [23:0] mant_fract = mant1_one << exp_delta_abs;
    wire   unsigned [23:0] mant_fract_unormalized = (fract_all_fract) ? mant1_one: mant_fract[22:0];

    //----------------------------------------------------------------------------
    //multiplying ....in1*in2
    //---------------------------------------------------------------------------
    wire   signed [48:0] mant_in1_in2 = mant1 * mant2;
    wire    signed [25:0] mant_in1_in2_shf = mant_in1_in2 >> 23; //andi ...no need to shift...just mask out the bottom 23 bits

```

```
    //----------------------------------------------------------------------------
    // denormalizing of the data before addition
    //----------------------------------------------------------------------------
    wire    signed [25:0] mant_in1_in2_shf_24 = (exp_delta_gt_24) ? 26'h0 : mant_in1_in2_shf >> exp_delta_abs;
    wire    signed [25:0] mant_mul = (exp_delta > 127) ? mant_in1_in2_shf: (mant_in1_in2_shf_24);
    wire   signed [25:0] mant3_shft = (exp_delta_gt_24) ? 25'h0 : mant3 >> exp_delta_abs;
    wire   signed [24:0] mant3_denorm = (exp_delta > 127) ? mant3_shft : mant3;

    //-------------------------------------------------------------------------------
    //addition of the product result with the third argument into the macc unit
    //-----------------------------------------------------------------------------
    wire   signed [26:0] mant_res = mant_mul + mant3_denorm;
    //----------------------------------------------------------------------------

    wire   signed [26:0] mant_result = mant_res >> 1;
    wire   signed [8:0] exp_mul_minus126 = exp_mul - 126;
    //wire   signed [8:0] exp_mul_m126 = (opcode_sel) ? exp_mul_minus126 : 9'h7f;
    wire   signed [8:0] exp2_plus1 = exp2 + 1;
    wire   signed [8:0] exp_res = (exp_delta >= 127) ? exp_mul_minus126 : exp2_plus1;

    wire   signed  [7:0] exp_fract = (fract_all_fract) ? exp1 : 8'h7f;
    wire   signed  [8:0] exp_result  = (opcode_sel) ? exp_res : exp_fract;
    //-----------------------------------------------------------------------------------
    wire   unsigned  [26:0] mant_result_neg = -mant_result;
```

```
    wire   unsigned [26:0] mant_res_unsigned = (mant_res > 0) ? mant_result: mant_result_neg;


    // this mux can be larger with the other opcodes being added in time
    wire    unsigned [23:0] mant_res_unormalized = (opcode_sel) ?  mant_res_unsigned[23:0]: mant_fract_unormalized;

    wire   unsigned [7:0] exp_modify = LZ(mant_res_unormalized);
    wire   unsigned [7:0] exp_final = exp_result - exp_modify;

    wire   unsigned [0:0] sign_add_final = (mant_res > 0) ? 0:1;
    wire   unsigned [0:0] sign_final = (opcode_sel) ? sign_add_final : sign1;

    wire   unsigned [23:0] mant_final_shifted = mant_res_unormalized << exp_modify;
    wire   unsigned [23:0] mant_final = mant_final_shifted;

    wire   unsigned [31:0] result_temp = cat(sign_final, exp_final, mant_final[22:0]);

    //-----------------------------------------------------------------------------------
    //defining the delay as 4 pipeline stages
    //-----------------------------------------------------------------------------------
    result = ResolveLatency(result_temp,5);
    //result = result_temp;


endmodule // macc32
```

```
//                              -*- Mode: Verilog -*-
// Filename        : sx.v
// Description     : Shader Export top level
// Author          : Andi Skende
// Created On      : Thu Mar 21 13:59:48 2002
// Last Modified By: .
// Last Modified On: .
// Update Count    : 0
// Status          : Unknown, Use with caution!

`timescale 1ns / 1ps
module sx(/*AUTOARG*/
   // Outputs
   SX_SC_quad_rtr, SX_RBB_rs_out, SX_RBB_rd_out, SX_RBBM_busy,
   SX_pipe_we, SX_pipe_re, SX_pipe_a, SX_pipe_wd,
   SX_SQ_exp_count_rdy, SX_SQ_exp_pos_avail, SX_SQ_exp_buf_avail,
   SX_SP_vtx_data0, SX_SP_vtx_data1, SX_SP_vtx_data2,
   SX_out_vtx_data0, SX_out_vtx_data1, SX_out_vtx_data2,
   SX_RB_quad_x, SX_RB_quad_y, SX_RB_quad_mask, SX_RB_quad_type,
   SX_RB_quad_pixel, SX_RB_quad_index, SX_RB0_quad_send,
   SX_RB1_quad_send, SX_RB2_quad_send, SX_RB3_quad_send,
   SX_RB0_color_data, SX_RB1_color_data, SX_RB2_color_data,
   SX_RB3_color_data, SX_RB0_color_send, SX_RB1_color_send,
   SX_RB2_color_send, SX_RB3_color_send, SX_RB0_index_rtr,
   SX_RB1_index_rtr, SX_RB2_index_rtr, SX_RB3_index_rtr, SX_PA_send,
```

```
   SX_PA_data,
   // Inputs
   CG_SX_pm_enb, SC_SX_quad_x, SC_SX_quad_y, SC_SX_quad_mask,
   SC_SX_quad_tilex, SC_SX_quad_tiley, SC_SX_quad_send, sclk_global,
   srst, SQ_SX_interp_flat_vtx, SQ_SX_interp_flat_gouraud,
   SQ_SX_interp_cyl_wrap, SQ_SX_pc_ptr0, SQ_SX_pc_ptr1,
   SQ_SX_pc_ptr2, SQ_SX_rt_sel, SQ_SX_pc_wr_en, SQ_SX_pc_wr_addr,
   SQ_SX_pc_channel_mask, SP0_SX_data0, SP0_SX_data1, SP0_SX_data2,
   SP0_SX_data3, SP1_SX_data0, SP1_SX_data1, SP1_SX_data2,
   SP1_SX_data3, SP0_SX_exp_pvalid, SP1_SX_exp_pvalid,
   SP0_SX_exp_alu_id, SP1_SX_exp_alu_id, SP0_SX_exporting,
   SP1_SX_exporting, SP0_SX_exp_dest, SP1_SX_exp_dest,
   SQ_SX_exp_type, SQ_SX_exp_number, SQ_SX_exp_state, SQ_SX_exp_id,
   SQ_SX_exp_valid, SQ_SX_free_done, SQ_SX_free_id,
   RBBM_SX_soft_reset, RBBM_we, RBBM_wd, RBBM_a, RBBM_be, RBBM_re,
   RBB_rs_in, RBB_rd_in, SX_in_vtx_data0, SX_in_vtx_data1,
   SX_in_vtx_data2, RB0_SX_quad_rtr, RB1_SX_quad_rtr,
   RB2_SX_quad_rtr, RB3_SX_quad_rtr, RB0_SX_color_rtr,
   RB1_SX_color_rtr, RB2_SX_color_rtr, RB3_SX_color_rtr,
   RB0_SX_index, RB1_SX_index, RB2_SX_index, RB3_SX_index,
   RB0_SX_index_send, RB1_SX_index_send, RB2_SX_index_send,
   RB3_SX_index_send, RB0_SX_index_op, RB1_SX_index_op,
   RB2_SX_index_op, RB3_SX_index_op, PA_SX_req, PA_SX_sp_id,
   PA_SX_offset, PA_SX_aux, PA_SX_last
   );
```

```

   //-----------------------------------------------------------------------//
   parameter unit_id = 1'b0;


   //-----------------------------------------------------------------//
   //Power managment control interface
   input [0:0] CG_SX_pm_enb;

   //-----------------------------------------------------------------------//
   //SC to SX Quad info interface
   //-----------------------------------------------------------------------//
   input [1:0]          SC_SX_quad_x;
   input [1:0]          SC_SX_quad_y;
   input [31:0]     SC_SX_quad_mask;
   input [1:0]          SC_SX_quad_tilex;
   input                SC_SX_quad_tiley;
   input                SC_SX_quad_send;
   output               SX_SC_quad_rtr;
   input                sclk_global;
   input           srst;

   wire            sclk;
   assign          sclk = sclk_global;

```

```
   wire [0:0]           sx_quad_rtr, q_sx_sc_quad_rtr;
   ati_dff_out #(1)  usx_sc_quad_rtr(sclk,sx_quad_rtr,q_sx_sc_quad_rtr);

   assign          SX_SC_quad_rtr = q_sx_sc_quad_rtr;

   wire [1:0]           q_sc_sx_quad_x;
   wire [1:0]           q_sc_sx_quad_y;
   wire [31:0]          q_sc_sx_quad_mask;
   wire [1:0]           q_sc_sx_quad_tilex;
   wire [0:0]           q_sc_sx_quad_tiley;
   wire [0:0]           q_sc_sx_quad_send;

   ati_dff_in #(2)  usc_sx_quad_x(sclk,SC_SX_quad_x,q_sc_sx_quad_x);
   ati_dff_in #(2)  usc_sx_quad_y(sclk,SC_SX_quad_y,q_sc_sx_quad_y);
   ati_dff_in #(32)  usc_sx_quad_mask(sclk,SC_SX_quad_mask,q_sc_sx_quad_mask);
   ati_dff_in #(2)  usc_sx_quad_tilex(sclk,SC_SX_quad_tilex,q_sc_sx_quad_tilex);
   ati_dff_in #(1)  usc_sx_quad_tiley(sclk,SC_SX_quad_tiley,q_sc_sx_quad_tiley);
   ati_dff_in #(1)  usc_sx_quad_send(sclk, SC_SX_quad_send, q_sc_sx_quad_send);

   //---------------------------------------------------------------------/
   //SQ to SX Interpolation Bus/Parameter Cache--------------------------/
   //---------------------------------------------------------------------/
   input [1:0]          SQ_SX_interp_flat_vtx;
   input [0:0]          SQ_SX_interp_flat_gouraud;
   input [3:0]          SQ_SX_interp_cyl_wrap;
```

```
input [10:0]    SQ_SX_pc_ptr0, SQ_SX_pc_ptr1, SQ_SX_pc_ptr2;
input [0:0]               SQ_SX_rt_sel;
input [0:0]               SQ_SX_pc_wr_en;
input [6:0]               SQ_SX_pc_wr_addr;
input [3:0]               SQ_SX_pc_channel_mask;

//-----------------------------------------------------------------------------------//
//Export Data Interface coming from SP going into Parameter Cache or Export Buffers
//-----------------------------------------------------------------------------------//
input [127:0] SP0_SX_data0,SP0_SX_data1,SP0_SX_data2,SP0_SX_data3;
input [127:0] SP1_SX_data0,SP1_SX_data1,SP1_SX_data2,SP1_SX_data3;

wire [127:0]  q_sp0_sx_data0, q_sp0_sx_data1, q_sp0_sx_data2, q_sp0_sx_data3;
wire [127:0]  q_sp1_sx_data0, q_sp1_sx_data1, q_sp1_sx_data2, q_sp1_sx_data3;

ati_dff_in #(128) usp0_sx_data0(sclk,SP0_SX_data0,q_sp0_sx_data0);
ati_dff_in #(128) usp0_sx_data1(sclk,SP0_SX_data1,q_sp0_sx_data1);
ati_dff_in #(128) usp0_sx_data2(sclk,SP0_SX_data2,q_sp0_sx_data2);
ati_dff_in #(128) usp0_sx_data3(sclk,SP0_SX_data3,q_sp0_sx_data3);

ati_dff_in #(128) usp1_sx_data0(sclk,SP1_SX_data0,q_sp1_sx_data0);
ati_dff_in #(128) usp1_sx_data1(sclk,SP1_SX_data1,q_sp1_sx_data1);
ati_dff_in #(128) usp1_sx_data2(sclk,SP1_SX_data2,q_sp1_sx_data2);
ati_dff_in #(128) usp1_sx_data3(sclk,SP1_SX_data3,q_sp1_sx_data3);

```

```

  input [3:0]    SP0_SX_exp_pvalid, SP1_SX_exp_pvalid; //pixel valid mask
  input          SP0_SX_exp_alu_id,  SP1_SX_exp_alu_id;  //isn't  one  of  these  signals
redundand ??? ANDI
  input [0:0]    SP0_SX_exporting, SP1_SX_exporting; //isn't one of these signals redundand
??? ANDI
  input [5:0]    SP0_SX_exp_dest,  SP1_SX_exp_dest;  //these  are  coming  straight  from  the
destination pointer of the ALU instruction
                                              //SP does nothing else other than pipelining them through.


  wire [3:0]     q_sp0_sx_exp_pvalid, q_sp1_sx_exp_pvalid;
  wire        q_sp0_sx_exp_alu_id, q_sp1_sx_exp_alu_id;
  wire [0:0]     q_sp0_sx_exporting, q_sp1_sx_exporting;
  wire [5:0]     q_sp0_sx_exp_dest, q_sp1_sx_exp_dest;


  ati_dff_in #(4) usp0_sx_exp_pvalid(sclk,SP0_SX_exp_pvalid,q_sp0_sx_exp_pvalid);
  ati_dff_in #(4) usp1_sx_exp_pvalid(sclk,SP1_SX_exp_pvalid,q_sp1_sx_exp_pvalid);
  ati_dff_in #(1) usp0_sx_exp_alu_id(sclk,SP0_SX_exp_alu_id,q_sp0_sx_exp_alu_id);
  ati_dff_in #(1) usp1_sx_exp_alu_id(sclk,SP1_SX_exp_alu_id,q_sp1_sx_exp_alu_id);
  ati_dff_in #(1) usp0_sx_exporting(sclk,SP0_SX_exporting,q_sp0_sx_exporting);
  ati_dff_in #(1) usp1_sx_exporting(sclk,SP1_SX_exporting,q_sp1_sx_exporting);
  ati_dff_in #(6) usp0_sx_exp_dst(sclk,SP0_SX_exp_dest,q_sp0_sx_exp_dest);
  ati_dff_in #(6) usp1_sx_exp_dst(sclk,SP1_SX_exp_dest,q_sp1_sx_exp_dest);

```

```
//-------------------------------------------------------------------------------/
//SQ to SX Export Control Bus
//-------------------------------------------------------------------------------/
input [1:0]    SQ_SX_exp_type;
input [1:0]    SQ_SX_exp_number;
input [2:0]    SQ_SX_exp_state;
input [0:0]    SQ_SX_exp_id;
input [0:0]    SQ_SX_exp_valid;
input [0:0]    SQ_SX_free_done;
input [0:0]    SQ_SX_free_id;

wire [1:0]     q_sq_sx_exp_type;
wire [1:0]     q_sq_sx_exp_number;
wire [2:0]     q_sq_sx_exp_state;
wire [0:0]     q_sq_sx_exp_id;
wire [0:0]     q_sq_sx_exp_valid;
wire [0:0]     q_sq_sx_free_done;
wire [0:0]     q_sq_sx_free_id;

ati_dff_in #(2) usq_sx_exp_type(sclk,SQ_SX_exp_type,q_sq_sx_exp_type);
ati_dff_in #(2) usq_sx_exp_number(sclk,SQ_SX_exp_number,q_sq_sx_exp_number);
ati_dff_in #(3) usq_sx_exp_state(sclk,SQ_SX_exp_state,q_sq_sx_exp_state);
ati_dff_in #(1) usq_sx_exp_id(sclk,SQ_SX_exp_id,q_sq_sx_exp_id);
ati_dff_in #(1) usq_sx_exp_valid(sclk,SQ_SX_exp_valid,q_sq_sx_exp_valid);
ati_dff_in #(1) usq_sx_free_done(sclk,SQ_SX_free_done,q_sq_sx_free_done);
```

```
  ati_dff_in #(1) usq_sx_free_id(sclk,SQ_SX_free_id,q_sq_sx_free_id);

  //-------------------------------------------------------------------------------/
  //CP/RBBM Interface for Real Time data and snooping state registers
  //-------------------------------------------------------------------------------/
  //There's no rtr nrtrtr signals ...tie them high or low at the top level

  input          RBBM_SX_soft_reset;
  input          RBBM_we;
  input [31:0]  RBBM_wd;
  input [14:0]  RBBM_a;
  input [3:0]    RBBM_be;
  input          RBBM_re;
  input       RBB_rs_in;
  input [31:0]  RBB_rd_in;

  wire   q_rbbm_sx_soft_reset;
  wire   q_rbbm_we;
  wire [31:0]    q_rbbm_wd;
  wire [14:0]    q_rbbm_a;
  wire [3:0]     q_rbbm_be;
  wire   q_rbbm_re;
  wire   q_rbb_rs_in;
  wire [31:0]    q_rbb_rd_in;

```

```
ati_dff_in #(1) urbbm_sx_soft_reset(sclk,RBBM_SX_soft_reset,q_rbbm_sx_soft_reset);

output        SX_RBB_rs_out;
output [31:0] SX_RBB_rd_out;
output        SX_RBBM_busy; //tie this high

wire   sx_rbbm_busy, q_sx_rbbm_busy;

//ati_dff_out #(1) usx_rbb_rs_out(sclk, sx_rbb_rs_out,q_sx_rbb_rs_out);
ati_dff_out #(1) usx_rbbm_busy(sclk, sx_rbbm_busy,q_sx_rbbm_busy);
//ati_dff_out #(32) usx_rbb_rd_out(sclk, sx_rbb_rd_out,q_sx_rbb_rd_out);


output [0:0]  SX_pipe_we, SX_pipe_re;
output [14:0] SX_pipe_a;
output [31:0] SX_pipe_wd;

//instantiating the rbbm interface common module

ati_rbbm_intf  urbbm_interface (
                    .sclk_reg(sclk),
                    .rbbm_we(RBBM_we),
                    .rbbm_re(RBBM_re),
                    .rbbm_a(RBBM_a),
                    .rbbm_wd(RBBM_wd),
```

```

                    //.reg_we,
                    //.reg_re,
                    //.reg_a,
                    //.reg_wd,

                    .pipe_we(SX_pipe_we),
                    .pipe_re(SX_pipe_re),
                    .pipe_a(SX_pipe_a),
                    .pipe_wd(SX_pipe_wd),

                    .rbbm_rs_in(RBB_rs_in),
                    .rbbm_rd_in(RBB_rd_in),

                    .block_rs(1'b0),
                    .block_rd(32'b0),

                    .rbbm_rs_out(SX_RBB_rs_out),
                    .rbbm_rd_out(SX_RBB_rd_out)
                    );


//--------------------------------------------------------------------------------/
//Export Buffer status control interface
//--------------------------------------------------------------------------------/
```

```
output        SX_SQ_exp_count_rdy;
output        SX_SQ_exp_pos_avail;
output [6:0]  SX_SQ_exp_buf_avail;

wire   sx_sq_exp_count_rdy, q_sx_sq_exp_count_rdy;
wire        sx_sq_exp_pos_avail, q_sx_sq_exp_pos_avail;
wire [6:0]      sx_sq_exp_buf_avail, q_sx_sq_exp_buf_avail;

//assign       SX_SQ_exp_count_rdy = q_sx_sq_exp_count_rdy;
//assign       SX_SQ_exp_pos_avail = q_sx_sq_exp_pos_avail;
//assign       SX_SQ_exp_buf_avail = q_sx_sq_exp_buf_avail;


//hack for now ....ANDI
assign         SX_SQ_exp_pos_avail = 1'b1;
assign       SX_SQ_exp_buf_avail = 7'h40;
assign       SX_SQ_exp_count_rdy = 1'b1;


//--------------------------------------------------------------------------------//
//Attribute Data into SP interpolators
//--------------------------------------------------------------------------------//
output [127:0] SX_SP_vtx_data0;
output [131:0] SX_SP_vtx_data1, SX_SP_vtx_data2; //these are (denormalized) outputs from the difference engines

```

```
wire [127:0]   sx_vtx_data0;
wire [131:0]   sx_vtx_data1, sx_vtx_data2;
wire [127:0]   q_sx_vtx_data0;
wire [131:0]   q_sx_vtx_data1, q_sx_vtx_data2;


//-----------------------------------------------------------------------------------//
//SX-SX interface ....Vertex Parameter Data sharing between two SXs
// FRANK !
// This is the exception we talked about
//-----------------------------------------------------------------------------------//

input [127:0]  SX_in_vtx_data0, SX_in_vtx_data1,SX_in_vtx_data2;
output [127:0] SX_out_vtx_data0, SX_out_vtx_data1,SX_out_vtx_data2;

//------------------------------------------------------------------------------------//
//SX-RB interface
//------------------------------------------------------------------------------------//

//--------------------//
//--Quad Interface-----//
//--------------------//
output [1:0]   SX_RB_quad_x;
output [1:0]   SX_RB_quad_y;
output [31:0]  SX_RB_quad_mask;
output [0:0]   SX_RB_quad_type;
output [3:0]   SX_RB_quad_pixel;
output [7:0]   SX_RB_quad_index;
output    [0:0]            SX_RB0_quad_send,    SX_RB1_quad_send,    SX_RB2_quad_send,
SX_RB3_quad_send;

wire [1:0]   q_sx_rb_quad_x;
wire [1:0]   q_sx_rb_quad_y;
wire [31:0]  q_sx_rb_quad_mask;
wire [0:0]   q_sx_rb_quad_type;
wire [3:0]   q_sx_rb_quad_pixel;
wire [7:0]   q_sx_rb_quad_index;
wire    [0:0]            q_sx_rb0_quad_send,    q_sx_rb1_quad_send,    q_sx_rb2_quad_send,
q_sx_rb3_quad_send;

wire [1:0]     sx_rb_quad_x;
wire [1:0]   sx_rb_quad_y;
wire [31:0]  sx_rb_quad_mask;
wire [0:0]   sx_rb_quad_type;
wire [3:0]   sx_rb_quad_pixel;
wire [7:0]   sx_rb_quad_index;
wire [0:0]   sx_rb0_quad_send, sx_rb1_quad_send, sx_rb2_quad_send, sx_rb3_quad_send;


ati_dff_out #(2)  usx_rb_quad_x(sclk, sx_rb_quad_x,q_sx_rb_quad_x);
ati_dff_out #(2)  usx_rb_quad_y(sclk, sx_rb_quad_y,q_sx_rb_quad_y);
```

```
ati_dff_out #(32) usx_rb_quad_mask(sclk, sx_rb_quad_mask,q_sx_rb_quad_mask);
ati_dff_out #(1)  usx_rb_quad_type(sclk, sx_rb_quad_type,q_sx_rb_quad_type);
ati_dff_out #(4)  usx_rb_quad_pixel(sclk, sx_rb_quad_pixel,q_sx_rb_quad_pixel);
ati_dff_out #(8)  usx_rb_quad_index(sclk, sx_rb_quad_index,q_sx_rb_quad_index);
ati_dff_out #(1)  usx_rb0_quad_send(sclk, sx_rb0_quad_send,q_sx_rb0_quad_send);
ati_dff_out #(1)  usx_rb1_quad_send(sclk, sx_rb1_quad_send,q_sx_rb1_quad_send);
ati_dff_out #(1)  usx_rb2_quad_send(sclk, sx_rb2_quad_send,q_sx_rb2_quad_send);
ati_dff_out #(1)  usx_rb3_quad_send(sclk, sx_rb3_quad_send,q_sx_rb3_quad_send);

assign          SX_RB_quad_x = q_sx_rb_quad_x;
assign      SX_RB_quad_y = q_sx_rb_quad_y;
assign      SX_RB_quad_mask = q_sx_rb_quad_mask;
assign      SX_RB_quad_type = q_sx_rb_quad_type;
assign      SX_RB_quad_pixel = q_sx_rb_quad_pixel;
assign      SX_RB_quad_index = q_sx_rb_quad_index;
assign      SX_RB0_quad_send = q_sx_rb0_quad_send;
assign      SX_RB1_quad_send = q_sx_rb1_quad_send;
assign      SX_RB2_quad_send = q_sx_rb2_quad_send;
assign      SX_RB3_quad_send = q_sx_rb3_quad_send;


input [0:0]     RB0_SX_quad_rtr, RB1_SX_quad_rtr, RB2_SX_quad_rtr, RB3_SX_quad_rtr;
wire [0:0]      q_rb0_sx_quad_rtr,q_rb1_sx_quad_rtr,q_rb2_sx_quad_rtr,q_rb3_sx_quad_rtr;

ati_dff_in #(1) urb0_sx_quad_rtr(sclk,RB0_SX_quad_rtr,q_rb0_sx_quad_rtr);
```

```
ati_dff_in #(1) urb1_sx_quad_rtr(sclk,RB1_SX_quad_rtr,q_rb1_sx_quad_rtr);
ati_dff_in #(1) urb2_sx_quad_rtr(sclk,RB2_SX_quad_rtr,q_rb2_sx_quad_rtr);
ati_dff_in #(1) urb3_sx_quad_rtr(sclk,RB3_SX_quad_rtr,q_rb3_sx_quad_rtr);

assign          SX_RB_quad_pixel = 4'b0;
assign          SX_RB_quad_type =  1'b0;


//--------------------------------------//
//--Pixel Color Data interface----------//
//--------------------------------------//
output                                                                  [127:0]
SX_RB0_color_data,SX_RB1_color_data,SX_RB2_color_data,SX_RB3_color_data;
output                                                                    [0:0]
SX_RB0_color_send,SX_RB1_color_send,SX_RB2_color_send,SX_RB3_color_send;


wire [127:0] sx_rb0_color_data,sx_rb1_color_data,sx_rb2_color_data,sx_rb3_color_data;
wire [0:0]   sx_rb0_color_send,sx_rb1_color_send,sx_rb2_color_send,sx_rb3_color_send;

wire                                                                    [127:0]
q_sx_rb0_color_data,q_sx_rb1_color_data,q_sx_rb2_color_data,q_sx_rb3_color_data;
wire                                                                      [0:0]
q_sx_rb0_color_send,q_sx_rb1_color_send,q_sx_rb2_color_send,q_sx_rb3_color_send;

ati_dff_out #(128) usx_rb0_color_data(sclk, sx_rb0_color_data,q_sx_rb0_color_data);
ati_dff_out #(128) usx_rb1_color_data(sclk, sx_rb1_color_data,q_sx_rb1_color_data);
```

```
ati_dff_out #(128) usx_rb2_color_data(sclk, sx_rb2_color_data,q_sx_rb2_color_data);
ati_dff_out #(128) usx_rb3_color_data(sclk, sx_rb3_color_data,q_sx_rb3_color_data);

ati_dff_out #(1) usx_rb0_color_send(sclk, sx_rb0_color_send,q_sx_rb0_color_send);
ati_dff_out #(1) usx_rb1_color_send(sclk, sx_rb1_color_send,q_sx_rb1_color_send);
ati_dff_out #(1) usx_rb2_color_send(sclk, sx_rb2_color_send,q_sx_rb2_color_send);
ati_dff_out #(1) usx_rb3_color_send(sclk, sx_rb3_color_send,q_sx_rb3_color_send);

assign      SX_RB0_color_data = q_sx_rb0_color_data;
assign      SX_RB1_color_data = q_sx_rb1_color_data;
assign      SX_RB2_color_data = q_sx_rb2_color_data;
assign      SX_RB3_color_data = q_sx_rb3_color_data;

assign          SX_RB0_color_send = q_sx_rb0_color_send;
assign          SX_RB1_color_send = q_sx_rb1_color_send;
assign          SX_RB2_color_send = q_sx_rb2_color_send;
assign          SX_RB3_color_send = q_sx_rb3_color_send;


input [0:0]     RB0_SX_color_rtr,RB1_SX_color_rtr,RB2_SX_color_rtr,RB3_SX_color_rtr;
wire  [0:0]     q_rb0_sx_color_rtr,q_rb1_sx_color_rtr,q_rb2_sx_color_rtr,q_rb3_sx_color_rtr;

ati_dff_in #(1) urb0_sx_color_rtr(sclk,RB0_SX_color_rtr,q_rb0_sx_color_rtr);
ati_dff_in #(1) urb1_sx_color_rtr(sclk,RB1_SX_color_rtr,q_rb1_sx_color_rtr);
ati_dff_in #(1) urb2_sx_color_rtr(sclk,RB2_SX_color_rtr,q_rb2_sx_color_rtr);
```

```
    ati_dff_in #(1) urb3_sx_color_rtr(sclk,RB3_SX_color_rtr,q_rb3_sx_color_rtr);



    //--------------------------------------------//

    //SX to RB quad index interface-------------//

    //-------------------------------------------//

    input [7:0]     RB0_SX_index,RB1_SX_index,RB2_SX_index,RB3_SX_index;

    input [0:0]
  RB0_SX_index_send,RB1_SX_index_send,RB2_SX_index_send,RB3_SX_index_send;

    input [0:0]     RB0_SX_index_op,            RB1_SX_index_op,            RB2_SX_index_op,
  RB3_SX_index_op;

    wire  [7:0]     q_rb0_sx_index,q_rb1_sx_index,q_rb2_sx_index,q_rb3_sx_index;

    wire [0:0]
  q_rb0_sx_index_send,q_rb1_sx_index_send,q_rb2_sx_index_send,q_rb3_sx_index_send;

    wire [0:0]      q_rb0_sx_index_op,           q_rb1_sx_index_op,           q_rb2_sx_index_op,
  q_rb3_sx_index_op;


    output                  [0:0]                                         SX_RB0_index_rtr,
  SX_RB1_index_rtr,SX_RB2_index_rtr,SX_RB3_index_rtr;


    wire [0:0]      sx_rb0_index_rtr, sx_rb1_index_rtr,sx_rb2_index_rtr,sx_rb3_index_rtr;

    wire [0:0]      q_sx_rb0_index_rtr,
  q_sx_rb1_index_rtr,q_sx_rb2_index_rtr,q_sx_rb3_index_rtr;


    ati_dff_in #(1) urb0_sx_index_send(sclk, RB0_SX_index_send,q_rb0_sx_index_send);

    ati_dff_in #(1) urb1_sx_index_send(sclk, RB1_SX_index_send,q_rb1_sx_index_send);

    ati_dff_in #(1) urb2_sx_index_send(sclk, RB2_SX_index_send,q_rb2_sx_index_send);

    ati_dff_in #(1) urb3_sx_index_send(sclk, RB3_SX_index_send,q_rb3_sx_index_send);
```

```

    ati_dff_in #(8) urb0_sx_index(sclk, RB0_SX_index,q_rb0_sx_index);

    ati_dff_in #(8) urb1_sx_index(sclk, RB1_SX_index,q_rb1_sx_index);

    ati_dff_in #(8) urb2_sx_index(sclk, RB2_SX_index,q_rb2_sx_index);

    ati_dff_in #(8) urb3_sx_index(sclk, RB3_SX_index,q_rb3_sx_index);


    ati_dff_in #(1) urb0_sx_index_op(sclk, RB0_SX_index_op,q_rb0_sx_index_op);

    ati_dff_in #(1) urb1_sx_index_op(sclk, RB1_SX_index_op,q_rb1_sx_index_op);

    ati_dff_in #(1) urb2_sx_index_op(sclk, RB2_SX_index_op,q_rb2_sx_index_op);

    ati_dff_in #(1) urb3_sx_index_op(sclk, RB3_SX_index_op,q_rb3_sx_index_op);


    ati_dff_out #(1) usx_rb0_index_rtr(sclk,sx_rb0_index_rtr, q_sx_rb0_index_rtr);

    ati_dff_out #(1) usx_rb1_index_rtr(sclk,sx_rb1_index_rtr, q_sx_rb1_index_rtr);

    ati_dff_out #(1) usx_rb2_index_rtr(sclk,sx_rb2_index_rtr, q_sx_rb2_index_rtr);

    ati_dff_out #(1) usx_rb3_index_rtr(sclk,sx_rb3_index_rtr, q_sx_rb3_index_rtr);


    assign          SX_RB0_index_rtr = q_sx_rb0_index_rtr;

    assign          SX_RB1_index_rtr = q_sx_rb1_index_rtr;

    assign          SX_RB2_index_rtr = q_sx_rb2_index_rtr;

    assign          SX_RB3_index_rtr = q_sx_rb3_index_rtr;


    //------------------------------------------------------------------------------//

    //PA(Primitive Assembly) - SX position export interface

    //------------------------------------------------------------------------------//

    input           PA_SX_req;
```

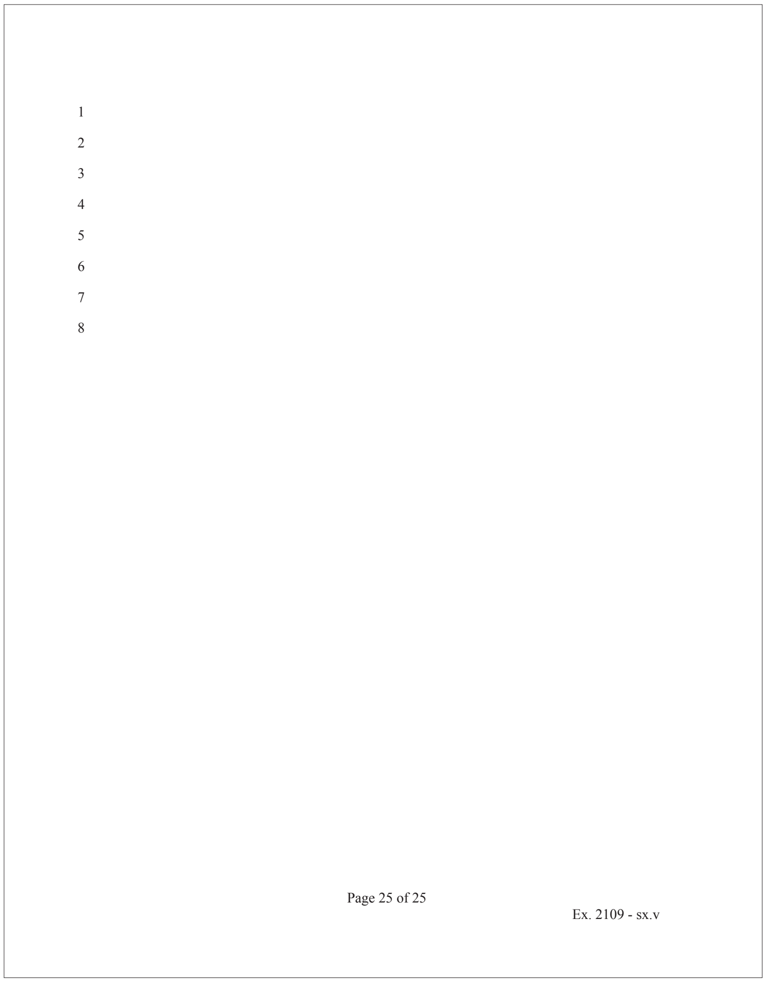
```
    input           PA_SX_sp_id;

    input [1:0]     PA_SX_offset;  //used to be PA_SX_export_phase

    input           PA_SX_aux; //used to be PA_SX_2ndbuff

    input           PA_SX_last;


    wire [0:0]      q_pa_sx_req;

    wire [0:0]      q_pa_sx_sp_id;

    wire [1:0]      q_pa_sx_offset;

    wire [0:0]      q_pa_sx_aux;

    wire [0:0]      q_pa_sx_last;


    ati_dff_in #(1) upa_sx_req(sclk,PA_SX_req,q_pa_sx_req);

    ati_dff_in #(1) upa_sx_sp_id(sclk,PA_SX_sp_id,q_pa_sx_sp_id);

    ati_dff_in #(2) upa_sx_offset(sclk,PA_SX_offset,q_pa_sx_offset);

    ati_dff_in #(1) upa_sx_aux(sclk,PA_SX_aux,q_pa_sx_aux);

    ati_dff_in #(1) upa_sx_last(sclk,PA_SX_last,q_pa_sx_last);


    output          SX_PA_send;

    output [127:0] SX_PA_data;


    wire    sx_pa_send, q_sx_pa_send;

    wire [127:0]    sx_pa_data, q_sx_pa_data;


    ati_dff_out #(1) usx_pa_send(sclk,sx_pa_send, q_sx_pa_send);

    ati_dff_out #(128) usx_pa_data(sclk,sx_pa_data, q_sx_pa_data);
```

```

    assign          SX_PA_send = q_sx_pa_send;

    assign          SX_PA_data = q_sx_pa_data;



  //=====================================================================
  ==

    wire    sx_id;

    assign          sx_id = unit_id;


    //..............Insert Logic...........


    parameter_caches  uparam_caches(// Outputs

                                          .SX_out_vtx_data0(SX_out_vtx_data0),
  .SX_out_vtx_data1(SX_out_vtx_data1), .SX_out_vtx_data2(SX_out_vtx_data2),

                                          .vtx_data0(sx_vtx_data0),.vtx_data1(sx_vtx_data1),
  .vtx_data2(sx_vtx_data2),

                                          // Inputs

                                          .SQ_SX_ptr0(SQ_SX_pc_ptr0),
  .SQ_SX_ptr1(SQ_SX_pc_ptr1),                                         .SQ_SX_ptr2(SQ_SX_pc_ptr2),
  .SQ_SX_pc_wr_en(SQ_SX_pc_wr_en),


  .SQ_SX_pc_wr_addr(SQ_SX_pc_wr_addr),.SQ_SX_pc_cmask(SQ_SX_pc_channel_mask),
  .SP_SX_data0(SP0_SX_data0), .SP_SX_data1(SP0_SX_data1),


  .SP_SX_data2(SP0_SX_data2),.SP_SX_data3(SP0_SX_data3),.SP_SX_data4(SP1_SX_data0),
  .SP_SX_data5(SP1_SX_data1),

                                          .SP_SX_data6(SP1_SX_data2),.SP_SX_data7(SP1_SX_data3),
  .sclk(sclk), .srst(srst), .sx_id(sx_id), .SX_in_vtx_data0(SX_in_vtx_data0),


  .SX_in_vtx_data1(SX_in_vtx_data1),.SX_in_vtx_data2(SX_in_vtx_data2),
.SQ_SX_interp_flat_vtx(SQ_SX_interp_flat_vtx),
.SQ_SX_interp_flat_gouraud(SQ_SX_interp_flat_gouraud)
                                        );


   export_control uexport_control(
                                        // Outputs
                                        .sx_sc_quad_rtr(sx_quad_rtr),
.sx_sq_exp_count_rdy(sx_sq_exp_count_rdy),
                                        //.SX_SQ_exp_pos_avail(sx_sq_exp_pos_avail),
                                        .sx_sq_exp_buf_avail(sx_sq_exp_buf_avail),
                                        .sx_rb_quad_x(sx_rb_quad_x), .sx_rb_quad_y(sx_rb_quad_y),
                                        .sx_rb_quad_mask(sx_rb_quad_mask),
.sx_rb_quad_type(sx_rb_quad_type),
                                        .sx_rb_quad_pixel(sx_rb_quad_pixel),
.sx_rb_quad_index(sx_rb_quad_index),
                                        .sx_rb0_quad_send(sx_rb0_quad_send),
.sx_rb1_quad_send(sx_rb1_quad_send),

.sx_rb2_quad_send(sx_rb2_quad_send),.sx_rb3_quad_send(sx_rb3_quad_send),
                                        .sx_rb0_color_data(sx_rb0_color_data),
.sx_rb1_color_data(sx_rb1_color_data),
                                        .sx_rb2_color_data(sx_rb2_color_data),
.sx_rb3_color_data(sx_rb3_color_data),

.sx_rb0_color_send(sx_rb0_color_send),.sx_rb1_color_send(sx_rb1_color_send),
                                        .sx_rb2_color_send(sx_rb2_color_send),
.sx_rb3_color_send(sx_rb3_color_send),
                                        .sx_rb0_index_rtr(sx_rb0_index_rtr),
.sx_rb1_index_rtr(sx_rb1_index_rtr),

.sx_rb2_index_rtr(sx_rb2_index_rtr),.sx_rb3_index_rtr(sx_rb3_index_rtr),
```

```
                                        .sx_pa_send(sx_pa_send), .sx_pa_data(sx_pa_data),
                                        // Inputs
                                        .sc_sx_quad_x(q_sc_sx_quad_x),
.sc_sx_quad_y(q_sc_sx_quad_y),
                                        .sc_sx_quad_mask(q_sc_sx_quad_mask),
.sc_sx_quad_tilex(q_sc_sx_quad_tilex),
                                        .sc_sx_quad_tiley(q_sc_sx_quad_tiley),
.sc_sx_quad_send(q_sc_sx_quad_send),
                                        .sclk(sclk), .srst(srst),

.sp0_sx_exp_pvalid(q_sp0_sx_exp_pvalid),.sp1_sx_exp_pvalid(q_sp1_sx_exp_pvalid),
                                        .sp0_sx_exp_alu_id(q_sp0_sx_exp_alu_id),
.sp1_sx_exp_alu_id(q_sp1_sx_exp_alu_id),
                                        .sp0_sx_exporting(q_sp0_sx_exporting),
.sp1_sx_exporting(q_sp1_sx_exporting),
                                        .sp0_sx_exp_dest(q_sp0_sx_exp_dest),
.sp1_sx_exp_dest(q_sp1_sx_exp_dest),
                                        .sp0_sx_data0(q_sp0_sx_data0),
.sp0_sx_data1(q_sp0_sx_data1),
                                        .sp0_sx_data2(q_sp0_sx_data2),
.sp0_sx_data3(q_sp0_sx_data3),
                                        .sp1_sx_data0(q_sp1_sx_data0),
.sp1_sx_data1(q_sp1_sx_data1),
                                        .sp1_sx_data2(q_sp1_sx_data2),
.sp1_sx_data3(q_sp1_sx_data3),

.sq_sx_exp_type(q_sq_sx_exp_type),.sq_sx_exp_number(q_sq_sx_exp_number),

.sq_sx_exp_state(q_sq_sx_exp_state),.sq_sx_exp_id(q_sq_sx_exp_id),

.sq_sx_exp_valid(q_sq_sx_exp_valid),.sq_sx_free_done(q_sq_sx_free_done),
                                        .sq_sx_free_id(q_sq_sx_free_id),
```

```

.rb0_sx_quad_rtr(q_rb0_sx_quad_rtr),.rb1_sx_quad_rtr(q_rb1_sx_quad_rtr),

.rb2_sx_quad_rtr(q_rb2_sx_quad_rtr),.rb3_sx_quad_rtr(q_rb3_sx_quad_rtr),

.rb0_sx_color_rtr(q_rb0_sx_color_rtr),.rb1_sx_color_rtr(q_rb1_sx_color_rtr),

.rb2_sx_color_rtr(q_rb2_sx_color_rtr),.rb3_sx_color_rtr(q_rb3_sx_color_rtr),
                                        .rb0_sx_index(q_rb0_sx_index),
                                        .rb1_sx_index(q_rb1_sx_index),
.rb2_sx_index(q_rb2_sx_index),
                                        .rb3_sx_index(q_rb3_sx_index),
.rb0_sx_index_send(q_rb0_sx_index_send),
                                        .rb1_sx_index_send(q_rb1_sx_index_send),
.rb2_sx_index_send(q_rb2_sx_index_send),
                                        .rb3_sx_index_send(q_rb3_sx_index_send),

.rb0_sx_index_op(q_rb0_sx_index_op),.rb1_sx_index_op(q_rb1_sx_index_op),

.rb2_sx_index_op(q_rb2_sx_index_op),.rb3_sx_index_op(q_rb3_sx_index_op),
                                        .rbbm_sx_soft_reset(q_rbbm_sx_soft_reset),
                                        .rbbm_we(q_rbbm_we), .rbbm_wd(q_rbbm_wd),
                                        .rbbm_a(q_rbbm_a), .rbbm_be(q_rbbm_be),
                                        .rbbm_re(q_rbbm_re),
                                        .rbb_rs_in(q_rbb_rs_in), .rbb_rd_in(q_rbb_rd_in),
                                        .pa_sx_req(q_pa_sx_req), .pa_sx_sp_id(q_pa_sx_sp_id),
                                        .pa_sx_offset(q_pa_sx_offset),.pa_sx_aux(q_pa_sx_aux),
                                        .pa_sx_last(q_pa_sx_last)
                                        );
```

```


   //registering the top level IO
   //--------------------------------------------------

   ati_dff_out #(128) usx_vtx_data0(sclk, sx_vtx_data0, q_sx_vtx_data0);
   ati_dff_out #(132) usx_vtx_data1(sclk, sx_vtx_data1, q_sx_vtx_data1);
   ati_dff_out #(132) usx_vtx_data2(sclk, sx_vtx_data2, q_sx_vtx_data2);

   assign        SX_SP_vtx_data0 = q_sx_vtx_data0;
   assign        SX_SP_vtx_data1 = q_sx_vtx_data1;
   assign        SX_SP_vtx_data2 = q_sx_vtx_data2;


endmodule
```



```
//                              -*- Mode: Verilog -*-
// Filename        : parameter_caches.v
// Description     : This is a wrapper around 8 parameter cache memories that each SX has
// Author          : Andi Skende
// Created On      : Tue Mar 26 20:07:29 2002
// Last Modified By: .
// Last Modified On: .
// Update Count    : 0
// Status          : Unknown, Use with caution!

`timescale 1ns / 1ps
module parameter_caches
 (/*AUTOARG*/
  // Outputs
  SX_out_vtx_data0, SX_out_vtx_data1, SX_out_vtx_data2, vtx_data0,
  vtx_data1, vtx_data2,
  // Inputs
  SQ_SX_ptr0, SQ_SX_ptr1, SQ_SX_ptr2, SQ_SX_pc_wr_en,
  SQ_SX_pc_wr_addr, SQ_SX_pc_cmask, SP_SX_data0, SP_SX_data1,
  SP_SX_data2, SP_SX_data3, SP_SX_data4, SP_SX_data5, SP_SX_data6,
  SP_SX_data7, sclk, srst, sx_id, SQ_SX_interp_flat_vtx,
  SQ_SX_interp_flat_gouraud, SX_in_vtx_data0, SX_in_vtx_data1,
  SX_in_vtx_data2
  );

```

```


  input [10:0] SQ_SX_ptr0, SQ_SX_ptr1,SQ_SX_ptr2;
  input        SQ_SX_pc_wr_en;
  input [6:0]    SQ_SX_pc_wr_addr;
  input [3:0]    SQ_SX_pc_cmask;
  input                                                                   [127:0]
SP_SX_data0,SP_SX_data1,SP_SX_data2,SP_SX_data3,SP_SX_data4,SP_SX_data5,SP_SX_data6,SP_SX_data7;
  input            sclk, srst; //clock and reset
  input        sx_id;
  input [1:0]    SQ_SX_interp_flat_vtx;
  input [0:0]    SQ_SX_interp_flat_gouraud;

  output [127:0] SX_out_vtx_data0 , SX_out_vtx_data1 , SX_out_vtx_data2;

  output [127:0] vtx_data0;
  output [131:0] vtx_data1, vtx_data2;
  reg  [127:0] vtx_data0_final, vtx_data1_final, vtx_data2_final;
  reg [127:0]   q0_vtx_data0_final,q0_vtx_data1_final,q0_vtx_data2_final;
  reg [127:0]   q1_vtx_data0_final,q2_vtx_data0_final;
  reg [127:0]   q1_vtx_data1_final,q2_vtx_data1_final;




```

```
  //-------------------------------------------------------------//
  //input from the other sx-------------------------------------//
  //-----------------------------------------------------------//
  input [127:0] SX_in_vtx_data0, SX_in_vtx_data1, SX_in_vtx_data2;
  reg [127:0]    other_vtx_data0, other_vtx_data1, other_vtx_data2;


  parameter [3:0] pc0_id =  4'h0,
                      pc1_id =  4'h1,
                      pc2_id =  4'h2,
                      pc3_id =  4'h3,
                      pc4_id =  4'h4,
                      pc5_id =  4'h5,
                      pc6_id =  4'h6,
                      pc7_id =  4'h7;

  parameter [3:0] pc8_id =  4'h8,
                      pc9_id =  4'h9,
                      pc10_id =  4'ha,
                      pc11_id =  4'hb,
                      pc12_id =  4'hc,
                      pc13_id =  4'hd,
                      pc14_id =  4'he,
                      pc15_id =  4'hf;

```

```
  wire [3:0]
sx_pc0_id,sx_pc1_id,sx_pc2_id,sx_pc3_id,sx_pc4_id,sx_pc5_id,sx_pc6_id,sx_pc7_id;

  //-----------------------------------------------------------------------------------------------------------
-------
  //creating an id for each of the parameter caches.
  //this id will be used inside the module uparam_cache_ctl to decode the read pointers into the
pc memory coming from SQ
  //-----------------------------------------------------------------------------------------------------------
-------

  assign           sx_pc0_id = sx_id ? pc4_id : pc0_id;
  assign           sx_pc1_id = sx_id ? pc5_id : pc1_id;
  assign           sx_pc2_id = sx_id ? pc6_id : pc2_id;
  assign           sx_pc3_id = sx_id ? pc7_id : pc3_id;
  assign           sx_pc4_id = sx_id ? pc12_id : pc8_id;
  assign           sx_pc5_id = sx_id ? pc13_id : pc9_id;
  assign           sx_pc6_id = sx_id ? pc14_id : pc10_id;
  assign           sx_pc7_id = sx_id ? pc15_id : pc11_id;




  wire                                                                   [127:0]
ovtx_data0,ovtx_data1,ovtx_data2,ovtx_data3,ovtx_data4,ovtx_data5,ovtx_data6,ovtx_data7;
  reg [127:0]      mux_vtx_data0, mux_vtx_data1, mux_vtx_data2;
  reg [127:0]      q0_mux_vtx_data0, q0_mux_vtx_data1,q0_mux_vtx_data2;
```

ATI 2110
LG v. ATI
IPR2015-00326

```
 reg [127:0]      q1_mux_vtx_data0, q1_mux_vtx_data1,q1_mux_vtx_data2;
 reg [127:0]      q2_mux_vtx_data0, q2_mux_vtx_data1,q2_mux_vtx_data2;
 reg [127:0]      q3_mux_vtx_data0, q3_mux_vtx_data1,q3_mux_vtx_data2;
 wire [131:0]    vtx_diff10, vtx_diff20;
 reg [127:0]      q0_vtx_diff10, q0_vtx_diff20;
 reg [127:0]      vtx_provoking;
 reg [127:0]      vtx_flat_gouraud0, vtx_flat_gouraud1,vtx_flat_gouraud2;
 reg [127:0]      q0_vtx_flat_gouraud0, q0_vtx_flat_gouraud1,q0_vtx_flat_gouraud2;
 reg [127:0]     q1_vtx_flat_gouraud0, q2_vtx_flat_gouraud0;
 reg [127:0]      q1_vtx_flat_gouraud2, q2_vtx_flat_gouraud2;
 reg [127:0]      q1_vtx_flat_gouraud1, q2_vtx_flat_gouraud1;


 param_cache_ctl uparam_cache_ctl0(
                              .ovtx_data(ovtx_data0),
                              .SQ_SX_ptr0(SQ_SX_ptr2),.SQ_SX_ptr1(SQ_SX_ptr0),
.SQ_SX_ptr2(SQ_SX_ptr1),

.SQ_SX_pc_wr_en(SQ_SX_pc_wr_en),.SQ_SX_pc_wr_addr(SQ_SX_pc_wr_addr),

.SQ_SX_pc_cmask(SQ_SX_pc_cmask),.SP_SX_data(SP_SX_data0),
                              .pc_memory_id(sx_pc0_id), .sclk(sclk),.srst(srst)
                              );


 param_cache_ctl uparam_cache_ctl1(
```

```
                              .ovtx_data(ovtx_data1),
                              .SQ_SX_ptr0(SQ_SX_ptr2),.SQ_SX_ptr1(SQ_SX_ptr0),
.SQ_SX_ptr2(SQ_SX_ptr1),

.SQ_SX_pc_wr_en(SQ_SX_pc_wr_en),.SQ_SX_pc_wr_addr(SQ_SX_pc_wr_addr),

.SQ_SX_pc_cmask(SQ_SX_pc_cmask),.SP_SX_data(SP_SX_data1),
                              .pc_memory_id(sx_pc1_id), .sclk(sclk),.srst(srst)
                              );

 param_cache_ctl uparam_cache_ctl2(
                              .ovtx_data(ovtx_data2),
                              .SQ_SX_ptr0(SQ_SX_ptr2),.SQ_SX_ptr1(SQ_SX_ptr0),
.SQ_SX_ptr2(SQ_SX_ptr1),

.SQ_SX_pc_wr_en(SQ_SX_pc_wr_en),.SQ_SX_pc_wr_addr(SQ_SX_pc_wr_addr),

.SQ_SX_pc_cmask(SQ_SX_pc_cmask),.SP_SX_data(SP_SX_data2),
                              .pc_memory_id(sx_pc2_id), .sclk(sclk),.srst(srst)
                              );
 param_cache_ctl uparam_cache_ctl3(
                              .ovtx_data(ovtx_data3),
                              .SQ_SX_ptr0(SQ_SX_ptr2),.SQ_SX_ptr1(SQ_SX_ptr0),
.SQ_SX_ptr2(SQ_SX_ptr1),

.SQ_SX_pc_wr_en(SQ_SX_pc_wr_en),.SQ_SX_pc_wr_addr(SQ_SX_pc_wr_addr),

.SQ_SX_pc_cmask(SQ_SX_pc_cmask),.SP_SX_data(SP_SX_data3),
                              .pc_memory_id(sx_pc3_id), .sclk(sclk),.srst(srst)
```

```
                              );
 param_cache_ctl uparam_cache_ctl4(
                              .ovtx_data(ovtx_data4),
                              .SQ_SX_ptr0(SQ_SX_ptr2),.SQ_SX_ptr1(SQ_SX_ptr0),
.SQ_SX_ptr2(SQ_SX_ptr1),

.SQ_SX_pc_wr_en(SQ_SX_pc_wr_en),.SQ_SX_pc_wr_addr(SQ_SX_pc_wr_addr),

.SQ_SX_pc_cmask(SQ_SX_pc_cmask),.SP_SX_data(SP_SX_data4),
                              .pc_memory_id(sx_pc4_id), .sclk(sclk),.srst(srst)
                              );
 param_cache_ctl uparam_cache_ctl5(
                              .ovtx_data(ovtx_data5),
                              .SQ_SX_ptr0(SQ_SX_ptr2),.SQ_SX_ptr1(SQ_SX_ptr0),
.SQ_SX_ptr2(SQ_SX_ptr1),

.SQ_SX_pc_wr_en(SQ_SX_pc_wr_en),.SQ_SX_pc_wr_addr(SQ_SX_pc_wr_addr),

.SQ_SX_pc_cmask(SQ_SX_pc_cmask),.SP_SX_data(SP_SX_data5),
                              .pc_memory_id(sx_pc5_id), .sclk(sclk),.srst(srst)
                              );
 param_cache_ctl uparam_cache_ctl6(
                              .ovtx_data(ovtx_data6),
                              .SQ_SX_ptr0(SQ_SX_ptr2),.SQ_SX_ptr1(SQ_SX_ptr0),
.SQ_SX_ptr2(SQ_SX_ptr1),

.SQ_SX_pc_wr_en(SQ_SX_pc_wr_en),.SQ_SX_pc_wr_addr(SQ_SX_pc_wr_addr),

.SQ_SX_pc_cmask(SQ_SX_pc_cmask),.SP_SX_data(SP_SX_data6),
```

```
                              .pc_memory_id(sx_pc6_id), .sclk(sclk),.srst(srst)
                              );
 param_cache_ctl uparam_cache_ctl7(
                              .ovtx_data(ovtx_data7),
                              .SQ_SX_ptr0(SQ_SX_ptr2),.SQ_SX_ptr1(SQ_SX_ptr0),
.SQ_SX_ptr2(SQ_SX_ptr1),

.SQ_SX_pc_wr_en(SQ_SX_pc_wr_en),.SQ_SX_pc_wr_addr(SQ_SX_pc_wr_addr),

.SQ_SX_pc_cmask(SQ_SX_pc_cmask),.SP_SX_data(SP_SX_data7),
                              .pc_memory_id(sx_pc7_id), .sclk(sclk),.srst(srst)
                              );

//--------------------------------------------------------------------------------//
//Selecting three vertex vectors---------------------------------------------------//
//--------------------------------------------------------------------------------//

 reg [3:0]        vtx_sel0, vtx_sel1, vtx_sel2;
 reg [3:0]        q0_vtx_sel0, q0_vtx_sel1, q0_vtx_sel2;
 reg [3:0]        q1_vtx_sel0, q1_vtx_sel1, q1_vtx_sel2;
 reg [3:0]        q2_vtx_sel0, q2_vtx_sel1, q2_vtx_sel2;
 reg [3:0]        q3_vtx_sel0, q3_vtx_sel1, q3_vtx_sel2;

 reg [127:0]      vertex0, vertex1,vertex2;


```

```
  reg [1:0]       flat_vtx_sel, q0_flat_vtx_sel, q1_flat_vtx_sel,q2_flat_vtx_sel,q3_flat_vtx_sel;
  reg [0:0]
flat_gouraud_sel,q0_flat_gouraud_sel,q1_flat_gouraud_sel,q2_flat_gouraud_sel,q3_flat_goura
ud_sel;



  always @(posedge sclk)
    begin
        if(srst)
          begin

            vtx_sel0<= 4'b0;
            vtx_sel1<= 4'b0;
            vtx_sel2<= 4'b0;
            q0_vtx_sel0<= 4'b0;
            q0_vtx_sel1<= 4'b0;
            q0_vtx_sel2<= 4'b0;
            q1_vtx_sel0<= 4'b0;
            q1_vtx_sel1<= 4'b0;
            q1_vtx_sel2<= 4'b0;
            q2_vtx_sel0<= 4'b0;
            q2_vtx_sel1<= 4'b0;
            q2_vtx_sel2<= 4'b0;
            q3_vtx_sel0<= 4'b0;
            q3_vtx_sel1<= 4'b0;
            q3_vtx_sel2<= 4'b0;
```

```
            flat_vtx_sel<= SQ_SX_interp_flat_vtx;
            q0_flat_vtx_sel<= 2'b0;
            q1_flat_vtx_sel <= 2'b0;
            q2_flat_vtx_sel <= 2'b0;
            q3_flat_vtx_sel <= 2'b0;
            flat_gouraud_sel <= 1'b0;
            q0_flat_gouraud_sel <= 1'b0;
            q1_flat_gouraud_sel <= 1'b0;
            q2_flat_gouraud_sel <= 1'b0;
            q3_flat_gouraud_sel <= 1'b0;
          end
        else
          begin
            vtx_sel0 <= SQ_SX_ptr0[10:7];
            vtx_sel1 <= SQ_SX_ptr1[10:7];
            vtx_sel2 <= SQ_SX_ptr2[10:7];
            other_vtx_data0 <= SX_in_vtx_data0;
            other_vtx_data1 <= SX_in_vtx_data1;
            other_vtx_data2 <= SX_in_vtx_data2;
            q0_vtx_sel0 <= vtx_sel0;
            q0_vtx_sel1 <= vtx_sel1;
            q0_vtx_sel2 <= vtx_sel2;
            q1_vtx_sel0 <= q0_vtx_sel0;
            q1_vtx_sel1 <= q0_vtx_sel1;
            q1_vtx_sel2 <= q0_vtx_sel2;
```

```
            q2_vtx_sel0 <= q1_vtx_sel0 ;
            q2_vtx_sel1 <= q1_vtx_sel1;
            q2_vtx_sel2 <= q1_vtx_sel2;
            q3_vtx_sel0 <= q2_vtx_sel0 ;
            q3_vtx_sel1 <= q2_vtx_sel1;
            q3_vtx_sel2 <= q2_vtx_sel2;
            q0_mux_vtx_data0 <= mux_vtx_data0;
            q0_mux_vtx_data1 <= mux_vtx_data1;
            q0_mux_vtx_data2 <= mux_vtx_data2;
            q1_mux_vtx_data0 <= q0_mux_vtx_data0;
            q1_mux_vtx_data1 <= q0_mux_vtx_data1;
            q1_mux_vtx_data2 <= q0_mux_vtx_data2;
            q2_mux_vtx_data0 <= q1_mux_vtx_data0;
            q2_mux_vtx_data1 <= q1_mux_vtx_data1;
            q2_mux_vtx_data2 <= q1_mux_vtx_data2;
            q3_mux_vtx_data0 <= q2_mux_vtx_data0;
            q3_mux_vtx_data1 <= q2_mux_vtx_data1;
            q3_mux_vtx_data2 <= q2_mux_vtx_data2;
            q0_vtx_data0_final <= vtx_data0_final;
            q1_vtx_data0_final <= q0_vtx_data0_final;
            q2_vtx_data0_final <= q1_vtx_data0_final;
            q0_vtx_data1_final <= vtx_data1_final;
            q0_vtx_data2_final <= vtx_data2_final;
            q0_vtx_diff10 <= vtx_diff10;
            q0_vtx_diff20 <= vtx_diff20;
```

```
            flat_vtx_sel<= 2'b0;
            q0_flat_vtx_sel<=flat_vtx_sel;
            q1_flat_vtx_sel <= q0_flat_vtx_sel;
            q2_flat_vtx_sel <= q1_flat_vtx_sel;
            q3_flat_vtx_sel <= q2_flat_vtx_sel;
            flat_gouraud_sel <= SQ_SX_interp_flat_gouraud;
            q0_flat_gouraud_sel <= flat_gouraud_sel;
            q1_flat_gouraud_sel <= q0_flat_gouraud_sel;
            q2_flat_gouraud_sel <= q1_flat_gouraud_sel ;
            q3_flat_gouraud_sel <= q2_flat_gouraud_sel;
            q0_vtx_flat_gouraud0 <= vtx_flat_gouraud0;
            q0_vtx_flat_gouraud1 <= vtx_flat_gouraud1;
            q1_vtx_flat_gouraud1 <= q0_vtx_flat_gouraud1;
            q2_vtx_flat_gouraud1 <= q1_vtx_flat_gouraud1;
            q0_vtx_flat_gouraud2 <= vtx_flat_gouraud2;
            q1_vtx_flat_gouraud2 <= q0_vtx_flat_gouraud2;
            q2_vtx_flat_gouraud2 <= q1_vtx_flat_gouraud2;
            q1_vtx_flat_gouraud0 <= q0_vtx_flat_gouraud0;
            q2_vtx_flat_gouraud0 <= q1_vtx_flat_gouraud0;
          end // else: !if(srst)
    end // always @ (posedge sclk)

  //-----------------------------------------------------------------------
  //three 8:1 muxes used to select the vertices
  //the 8:1 muxes are followed by a 2:1 mux each to select from this SX data
```

```
//or the "other" data coming from the other SX
//---------------------------------------------------------------------------
always@(/*AUTOSENSE*/ovtx_data0 or ovtx_data1 or ovtx_data2
           or ovtx_data3 or ovtx_data4 or ovtx_data5 or ovtx_data6
           or ovtx_data7 or q1_vtx_sel0 or sx_pc0_id or sx_pc1_id
           or sx_pc2_id or sx_pc3_id or sx_pc4_id or sx_pc5_id
           or sx_pc6_id or sx_pc7_id)
  begin
        case(q1_vtx_sel0)
          sx_pc0_id:mux_vtx_data0 = ovtx_data0;
          sx_pc1_id:mux_vtx_data0 = ovtx_data1;
          sx_pc2_id:mux_vtx_data0 = ovtx_data2;
          sx_pc3_id:mux_vtx_data0 = ovtx_data3;
          sx_pc4_id:mux_vtx_data0 = ovtx_data4;
          sx_pc5_id:mux_vtx_data0 = ovtx_data5;
          sx_pc6_id:mux_vtx_data0 = ovtx_data6;
          sx_pc7_id:mux_vtx_data0 = ovtx_data7;
          default : mux_vtx_data0 = ovtx_data0;
        endcase // case(q0_vtx_sel0[2:0])
  end // always@ (q0_vtx_sel0 or ovtx_data0 or ovtx_data1 or ovtx_data2 or..


always@(/*AUTOSENSE*/ovtx_data0 or ovtx_data1 or ovtx_data2
           or ovtx_data3 or ovtx_data4 or ovtx_data5 or ovtx_data6
           or ovtx_data7 or q1_vtx_sel1 or sx_pc0_id or sx_pc1_id
```

```
           or sx_pc2_id or sx_pc3_id or sx_pc4_id or sx_pc5_id
           or sx_pc6_id or sx_pc7_id)
  begin
        case(q1_vtx_sel1)
          sx_pc0_id:mux_vtx_data1 = ovtx_data0;
          sx_pc1_id:mux_vtx_data1 = ovtx_data1;
          sx_pc2_id:mux_vtx_data1 = ovtx_data2;
          sx_pc3_id:mux_vtx_data1 = ovtx_data3;
          sx_pc4_id:mux_vtx_data1 = ovtx_data4;
          sx_pc5_id:mux_vtx_data1 = ovtx_data5;
          sx_pc6_id:mux_vtx_data1 = ovtx_data6;
          sx_pc7_id:mux_vtx_data1 = ovtx_data7;
          default : mux_vtx_data1 = ovtx_data0;
        endcase // case(q1_vtx_sel1[2:0])
  end // always@ (q1_vtx_sel1 or ovtx_data0 or ovtx_data1 or ovtx_data2 or...

always@(/*AUTOSENSE*/ovtx_data0 or ovtx_data1 or ovtx_data2
           or ovtx_data3 or ovtx_data4 or ovtx_data5 or ovtx_data6
           or ovtx_data7 or q1_vtx_sel2 or sx_pc0_id or sx_pc1_id
           or sx_pc2_id or sx_pc3_id or sx_pc4_id or sx_pc5_id
           or sx_pc6_id or sx_pc7_id)
  begin
        case(q1_vtx_sel2)
          sx_pc0_id:mux_vtx_data2 = ovtx_data0;
          sx_pc1_id:mux_vtx_data2 = ovtx_data1;
```

```
          sx_pc2_id:mux_vtx_data2 = ovtx_data2;
          sx_pc3_id:mux_vtx_data2 = ovtx_data3;
          sx_pc4_id:mux_vtx_data2 = ovtx_data4;
          sx_pc5_id:mux_vtx_data2 = ovtx_data5;
          sx_pc6_id:mux_vtx_data2 = ovtx_data6;
          sx_pc7_id:mux_vtx_data2 = ovtx_data7;
          default : mux_vtx_data2 = ovtx_data0;
        endcase // case(q1_vtx_sel2[2:0])
  end // always@ (q1_vtx_sel2 or ovtx_data0 or ovtx_data1 or ovtx_data2 or...

wire [0:0] final_sel0;
wire [0:0] final_sel1;

assign     final_sel0 = sx_id ? 1'b1: 1'b0;
assign     final_sel1 = ~ final_sel0;

always @(/*AUTOSENSE*/final_sel0 or final_sel1 or other_vtx_data0
            or q0_mux_vtx_data0 or q2_vtx_sel0)
  begin
        case(q2_vtx_sel0[2])
          final_sel0:vtx_data0_final = q0_mux_vtx_data0;
          final_sel1:vtx_data0_final = other_vtx_data0;
        endcase // case(q1_vtx_sel0[2])
  end

```

```
 always @(/*AUTOSENSE*/final_sel0 or final_sel1 or other_vtx_data1
             or q0_mux_vtx_data1 or q2_vtx_sel1)
  begin
        case(q2_vtx_sel1[2])
          final_sel0:vtx_data1_final = q0_mux_vtx_data1;
          final_sel1:vtx_data1_final = other_vtx_data1;
        endcase // case(q1_vtx_sel1[2])
  end

 always @(/*AUTOSENSE*/final_sel0 or final_sel1 or other_vtx_data2
             or q0_mux_vtx_data2 or q2_vtx_sel2)
  begin
        case(q2_vtx_sel2[2])
          final_sel0:vtx_data2_final = q0_mux_vtx_data2;
          final_sel1:vtx_data2_final = other_vtx_data2;
        endcase // case(q1_vtx_sel2[2])
  end


//-------------------------------------------------------------
//Flat vs. Gouraud shading vertex selection...two level muxing
//-----------------------------------------------------------

always @(/*AUTOSENSE*/q0_vtx_data0_final or q0_vtx_data1_final
            or q0_vtx_data2_final or q3_flat_vtx_sel)
  begin
     case(q3_flat_vtx_sel)
       2'b00:vtx_provoking = q0_vtx_data0_final;
       2'b01:vtx_provoking = q0_vtx_data1_final;
       2'b10:vtx_provoking = q0_vtx_data2_final;
       default: vtx_provoking = q0_vtx_data0_final;
     endcase // case(SQ_SP_interp_flat_vtx)
  end

 always @(/*AUTOSENSE*/q0_vtx_data0_final or q3_flat_gouraud_sel
          or vtx_provoking)
  begin
     case(q3_flat_gouraud_sel)
       1'b0:vtx_flat_gouraud0 =vtx_provoking  ;
       1'b1:vtx_flat_gouraud0 =q0_vtx_data0_final ;
     endcase // case(SQ_SP_interp_flat_gouraud)
  end

 always @(/*AUTOSENSE*/q0_vtx_data1_final or q3_flat_gouraud_sel
          or vtx_provoking)
  begin
     case(q3_flat_gouraud_sel)
       1'b0:vtx_flat_gouraud1 =vtx_provoking  ;
       1'b1:vtx_flat_gouraud1 =q0_vtx_data1_final ;
     endcase // case(SQ_SP_interp_flat_gouraud)
```

```
  end

 always @(/*AUTOSENSE*/q0_vtx_data2_final or q3_flat_gouraud_sel
          or vtx_provoking)
  begin
     case(q3_flat_gouraud_sel)
       1'b0:vtx_flat_gouraud2 =vtx_provoking  ;
       1'b1:vtx_flat_gouraud2 =q0_vtx_data2_final ;
     endcase // case(SQ_SP_interp_flat_gouraud)
  end



 //------------------------------------------------------------------------------------------------------------//
 //Difference engine for the vertex attributes. This difference is used by the barycentric interpolators
 //------------------------------------------------------------------------------------------------------------//
 wire [32:0] vertex10_red, vertex10_green,vertex10_blue,vertex10_alpha;
 wire [32:0] vertex20_red, vertex20_green,vertex20_blue,vertex20_alpha;

 param_sub param10_red(q0_vtx_flat_gouraud1[31:0], q0_vtx_flat_gouraud0[31:0], vertex10_red, sclk);
 param_sub param10_green(q0_vtx_flat_gouraud1[63:32], q0_vtx_flat_gouraud0[63:32], vertex10_green, sclk);
 param_sub param10_blue(q0_vtx_flat_gouraud1[95:64], q0_vtx_flat_gouraud0[95:64], vertex10_blue, sclk);
 param_sub param10_alpha(q0_vtx_flat_gouraud1[127:96], q0_vtx_flat_gouraud0[127:96], vertex10_alpha, sclk);
```

```

 param_sub param20_red(q0_vtx_flat_gouraud2[31:0], q0_vtx_flat_gouraud0[31:0], vertex20_red, sclk);
 param_sub param20_green(q0_vtx_flat_gouraud2[63:32], q0_vtx_flat_gouraud0[63:32], vertex20_green, sclk);
 param_sub param20_blue(q0_vtx_flat_gouraud2[95:64], q0_vtx_flat_gouraud0[95:64], vertex20_blue, sclk);
 param_sub param20_alpha(q0_vtx_flat_gouraud2[127:96], q0_vtx_flat_gouraud0[127:96], vertex20_alpha, sclk);



 //-------------------------------------------------------------
 //Outputs to the other SX block
 //-----------------------------------------------------------
 assign SX_out_vtx_data0 = mux_vtx_data0;
 assign SX_out_vtx_data1 = mux_vtx_data1;
 assign SX_out_vtx_data2 = mux_vtx_data2;


 //-----------------------------------------------------------//
 //Output to the SP interpolator--------------------------------//
 //-----------------------------------------------------------//

 assign vtx_diff10 = {vertex10_alpha,vertex10_blue,vertex10_green,vertex10_red};
 assign vtx_diff20 = {vertex20_alpha,vertex20_blue,vertex20_green,vertex20_red};
 assign vtx_data0 = q2_vtx_flat_gouraud0;
```

```
 assign vtx_data1 = q0_vtx_diff10;
 assign vtx_data2 = q0_vtx_diff20;




endmodule // parameter_caches
```

```
//                              -*- Mode: Verilog -*-
// Filename        : param_cache_ctl.v
// Description     : This module implements the read/write logic for the parameter caches
// Author          : Andi Skende
// Created On      : Tue Mar 26 19:57:49 2002
// Last Modified By: .
// Last Modified On: .
// Update Count    : 0
// Status          : Unknown, Use with caution!

`timescale 1ns / 1ps

module param_cache_ctl
  (/*AUTOARG*/
   // Outputs
   ovtx_data,
   // Inputs
   SQ_SX_ptr0, SQ_SX_ptr1, SQ_SX_ptr2, SQ_SX_pc_wr_en,
   SQ_SX_pc_wr_addr, SQ_SX_pc_cmask, SP_SX_data, pc_memory_id, sclk,
   srst
   );


   //------------------------------------------
   //PC control and interpolation interface
```

```
//------------------------------------------
input [10:0] SQ_SX_ptr0, SQ_SX_ptr1,SQ_SX_ptr2;
input        SQ_SX_pc_wr_en;
input [6:0]    SQ_SX_pc_wr_addr;
input [3:0]    SQ_SX_pc_cmask;
input [127:0] SP_SX_data;
input [3:0]    pc_memory_id; //0...16 a different id for each instance.
                           //The first 8 PCs 0-7 belong to SX0, while 8-15 to SX1
input            sclk, srst; //clock and reset


//------------------------------------------------
//Data out of the parameter caches
//------------------------------------------------

output [127:0] ovtx_data;

reg [10:0]     pc_ptr0, pc_ptr1,pc_ptr2;
reg              pc_wr_en;
reg [6:0]      pc_wr_addr;
reg [3:0]      pc_cmask;
reg [127:0]   vertex_data_in;
wire [127:0] vertex_data_out;
reg [127:0]   q0_vertex_data_out;

```

```
   wire [3:0]     mem_id;
   assign           mem_id = pc_memory_id;

   always @(posedge sclk)
     begin
          if(srst)
            begin
               pc_ptr0 <= 11'b0;
               pc_ptr1 <= 11'b0;
               pc_ptr2 <= 11'b0;
               pc_wr_en <= 1'b0;
               pc_wr_addr <= 7'b0;
               pc_cmask <= 4'b0;
               vertex_data_in <= 127'b0;
               q0_vertex_data_out <= 127'b0;
            end
          else
            begin
               pc_ptr0 <= SQ_SX_ptr0;
               pc_ptr1 <= SQ_SX_ptr1;
               pc_ptr2 <= SQ_SX_ptr2;
               pc_wr_en <= SQ_SX_pc_wr_en;
               pc_wr_addr <= SQ_SX_pc_wr_addr;
               pc_cmask <=  SQ_SX_pc_cmask;
               vertex_data_in <= SP_SX_data;
```

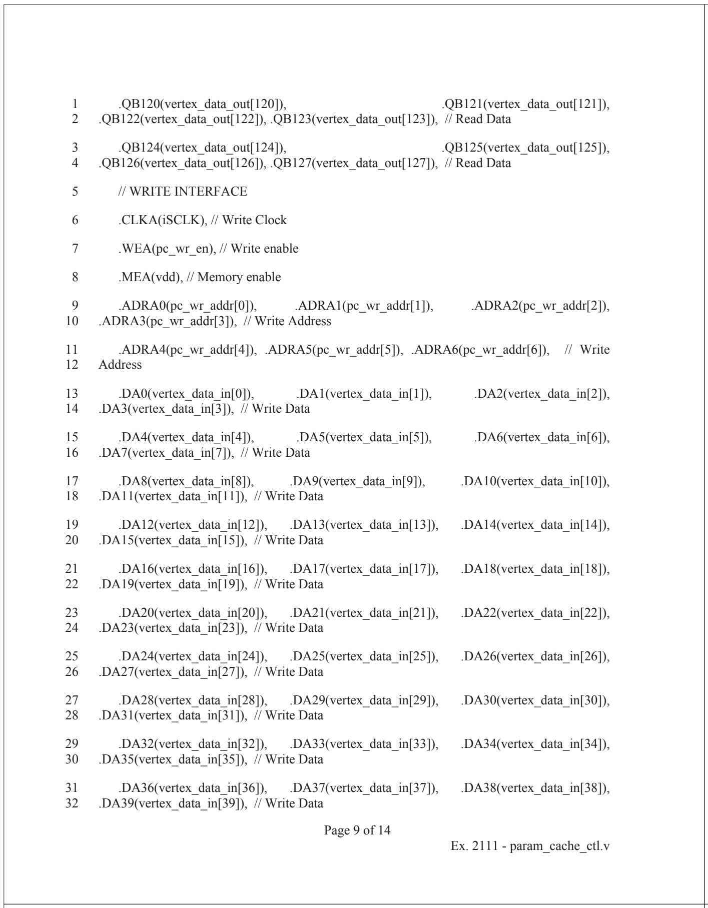
```
               q0_vertex_data_out <= vertex_data_out;
            end
  end // always @ (posedge sclk)


//-----------------------------------------------------------------------------------
//parameter cache write address decoding
//finding out whether any of the above addresses applies to this parameter cache memory
//in other words if the vertex0 1 or 2 data is in this specific memory
//-----------------------------------------------------------------------------------

wire is_ptr0, is_ptr1,is_ptr2;
wire [2:0] sel_ptr;

assign is_ptr0 = &(pc_ptr0[10:7] ^~ mem_id);
assign is_ptr1 = &(pc_ptr1[10:7] ^~ mem_id);
assign is_ptr2 = &(pc_ptr2[10:7] ^~ mem_id);
assign sel_ptr = {is_ptr2,is_ptr1,is_ptr0};

reg [6:0] pc_index; //read pointer from the Parameter caches
reg [0:0] pc_rd_en;

always@(is_ptr0  or is_ptr1 or is_ptr2 or pc_ptr0 or pc_ptr1 or pc_ptr2 or sel_ptr)
  begin
       case(sel_ptr)
```

```
          3'b000:
            begin
              pc_index = pc_ptr0;
              pc_rd_en = 1'b0;
            end
          3'b001:
            begin
              pc_index = pc_ptr0;
              pc_rd_en = 1'b1;
            end
          3'b010:
            begin
              pc_index = pc_ptr1;
              pc_rd_en = 1'b1;
            end
          3'b100:
            begin
              pc_index = pc_ptr2;
              pc_rd_en = 1'b1;
            end
          default:
            begin
              pc_index = pc_ptr0;
              pc_rd_en = 1'b0;
            end
```

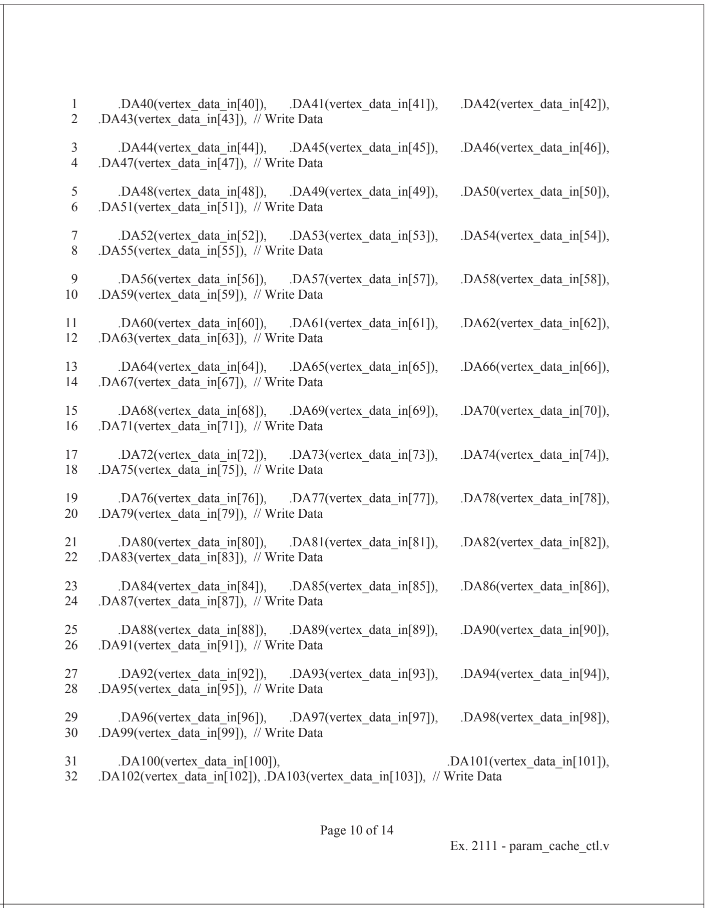
```
          endcase // case(sel_ptr)
      end // always@ (is_ptr0  or is_ptr1 or is_ptr2 or pc_ptr0 or pc_ptr1 or pc_ptr2)


`ifdef USE_BEHAVE_MEM
   dum_mem_p2 #(7,128) u_pc(.iRCLK(sclk),
                            .iWCLK(sclk),
                            .iMER(pc_rd_en),
                            .iMEW(pc_wr_en),
                            .iWEN(pc_wr_en),
                            .iRADR(pc_index),
                            .iWADR(pc_wr_addr),
                            .iD(vertex_data_in),
                            .oQ(vertex_data_out)
                            );
`else // !`ifdef USE_BEHAVE_MEM
   rfsd2_128x128cm2sw0 u_pc
     //   tp_coord_fifo_ram utp_coord_fifo_ram_0
     //          (/*VRGIO tp_coord_fifo_ram cfifo_in cfifo_out q_cfifo_wptr q_cfifo_rptr cfifo_ram_wen cfifo_ram_ren I0*/
     //      );
     (/*VRGIO rfsd2_128x128cm2sw0 vertex_data_in vertex_data_out pc_wr_addr pc_index pc_wr_en pc_rd_en null*/
      // READ INTERFACE
      .CLKB(iSCLK), // Read Clock
      .OEB(pc_rd_en), // Output enable
```

```
      .MEB(vdd), // Read enable
      .ADRB0(pc_index[0]),      .ADRB1(pc_index[1]),      .ADRB2(pc_index[2]), .ADRB3(pc_index[3]),  // Read Address
      .ADRB4(pc_index[4]), .ADRB5(pc_index[5]), .ADRB6(pc_index[6]),  // Read Address
      .QB0(vertex_data_out[0]),     .QB1(vertex_data_out[1]),     .QB2(vertex_data_out[2]), .QB3(vertex_data_out[3]),  // Read Data
      .QB4(vertex_data_out[4]),     .QB5(vertex_data_out[5]),     .QB6(vertex_data_out[6]), .QB7(vertex_data_out[7]),  // Read Data
      .QB8(vertex_data_out[8]),     .QB9(vertex_data_out[9]),     .QB10(vertex_data_out[10]), .QB11(vertex_data_out[11]),  // Read Data
      .QB12(vertex_data_out[12]),   .QB13(vertex_data_out[13]),   .QB14(vertex_data_out[14]), .QB15(vertex_data_out[15]),  // Read Data
      .QB16(vertex_data_out[16]),   .QB17(vertex_data_out[17]),   .QB18(vertex_data_out[18]), .QB19(vertex_data_out[19]),  // Read Data
      .QB20(vertex_data_out[20]),   .QB21(vertex_data_out[21]),   .QB22(vertex_data_out[22]), .QB23(vertex_data_out[23]),  // Read Data
      .QB24(vertex_data_out[24]),   .QB25(vertex_data_out[25]),   .QB26(vertex_data_out[26]), .QB27(vertex_data_out[27]),  // Read Data
      .QB28(vertex_data_out[28]),   .QB29(vertex_data_out[29]),   .QB30(vertex_data_out[30]), .QB31(vertex_data_out[31]),  // Read Data
      .QB32(vertex_data_out[32]),   .QB33(vertex_data_out[33]),   .QB34(vertex_data_out[34]), .QB35(vertex_data_out[35]),  // Read Data
      .QB36(vertex_data_out[36]),   .QB37(vertex_data_out[37]),   .QB38(vertex_data_out[38]), .QB39(vertex_data_out[39]),  // Read Data
      .QB40(vertex_data_out[40]),   .QB41(vertex_data_out[41]),   .QB42(vertex_data_out[42]), .QB43(vertex_data_out[43]),  // Read Data
      .QB44(vertex_data_out[44]),   .QB45(vertex_data_out[45]),   .QB46(vertex_data_out[46]), .QB47(vertex_data_out[47]),  // Read Data
      .QB48(vertex_data_out[48]),   .QB49(vertex_data_out[49]),   .QB50(vertex_data_out[50]), .QB51(vertex_data_out[51]),  // Read Data
      .QB52(vertex_data_out[52]),   .QB53(vertex_data_out[53]),   .QB54(vertex_data_out[54]), .QB55(vertex_data_out[55]),  // Read Data
```

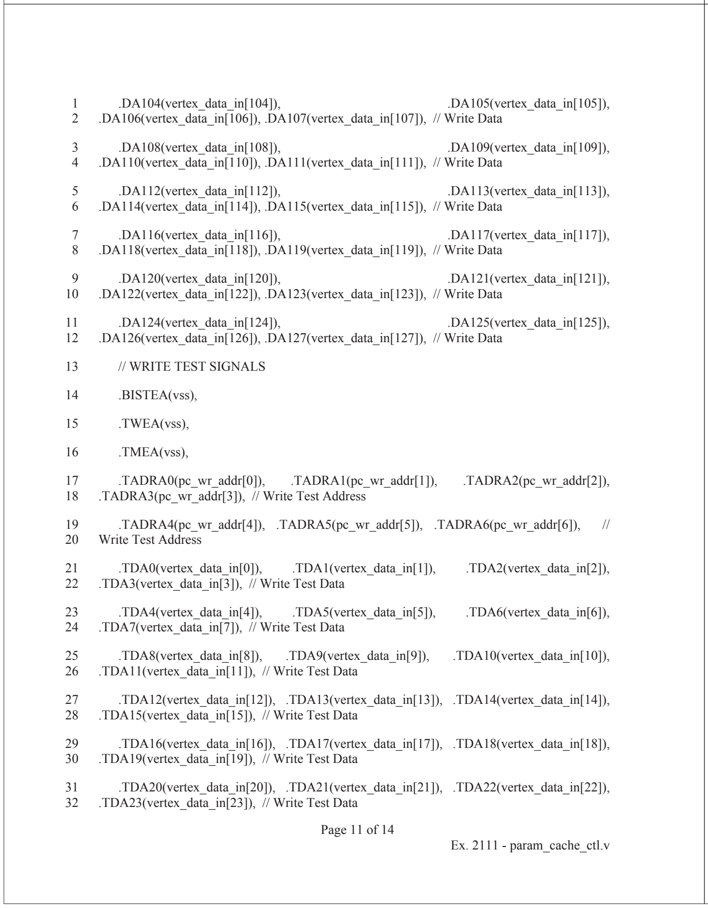
```
      .QB56(vertex_data_out[56]),   .QB57(vertex_data_out[57]),   .QB58(vertex_data_out[58]), .QB59(vertex_data_out[59]),  // Read Data
      .QB60(vertex_data_out[60]),   .QB61(vertex_data_out[61]),   .QB62(vertex_data_out[62]), .QB63(vertex_data_out[63]),  // Read Data
      .QB64(vertex_data_out[64]),   .QB65(vertex_data_out[65]),   .QB66(vertex_data_out[66]), .QB67(vertex_data_out[67]),  // Read Data
      .QB68(vertex_data_out[68]),   .QB69(vertex_data_out[69]),   .QB70(vertex_data_out[70]), .QB71(vertex_data_out[71]),  // Read Data
      .QB72(vertex_data_out[72]),   .QB73(vertex_data_out[73]),   .QB74(vertex_data_out[74]), .QB75(vertex_data_out[75]),  // Read Data
      .QB76(vertex_data_out[76]),   .QB77(vertex_data_out[77]),   .QB78(vertex_data_out[78]), .QB79(vertex_data_out[79]),  // Read Data
      .QB80(vertex_data_out[80]),   .QB81(vertex_data_out[81]),   .QB82(vertex_data_out[82]), .QB83(vertex_data_out[83]),  // Read Data
      .QB84(vertex_data_out[84]),   .QB85(vertex_data_out[85]),   .QB86(vertex_data_out[86]), .QB87(vertex_data_out[87]),  // Read Data
      .QB88(vertex_data_out[88]),   .QB89(vertex_data_out[89]),   .QB90(vertex_data_out[90]), .QB91(vertex_data_out[91]),  // Read Data
      .QB92(vertex_data_out[92]),   .QB93(vertex_data_out[93]),   .QB94(vertex_data_out[94]), .QB95(vertex_data_out[95]),  // Read Data
      .QB96(vertex_data_out[96]),   .QB97(vertex_data_out[97]),   .QB98(vertex_data_out[98]), .QB99(vertex_data_out[99]),  // Read Data
      .QB100(vertex_data_out[100]),                               .QB101(vertex_data_out[101]), .QB102(vertex_data_out[102]), .QB103(vertex_data_out[103]),  // Read Data
      .QB104(vertex_data_out[104]),                               .QB105(vertex_data_out[105]), .QB106(vertex_data_out[106]), .QB107(vertex_data_out[107]),  // Read Data
      .QB108(vertex_data_out[108]),                               .QB109(vertex_data_out[109]), .QB110(vertex_data_out[110]), .QB111(vertex_data_out[111]),  // Read Data
      .QB112(vertex_data_out[112]),                               .QB113(vertex_data_out[113]), .QB114(vertex_data_out[114]), .QB115(vertex_data_out[115]),  // Read Data
      .QB116(vertex_data_out[116]),                               .QB117(vertex_data_out[117]), .QB118(vertex_data_out[118]), .QB119(vertex_data_out[119]),  // Read Data
```



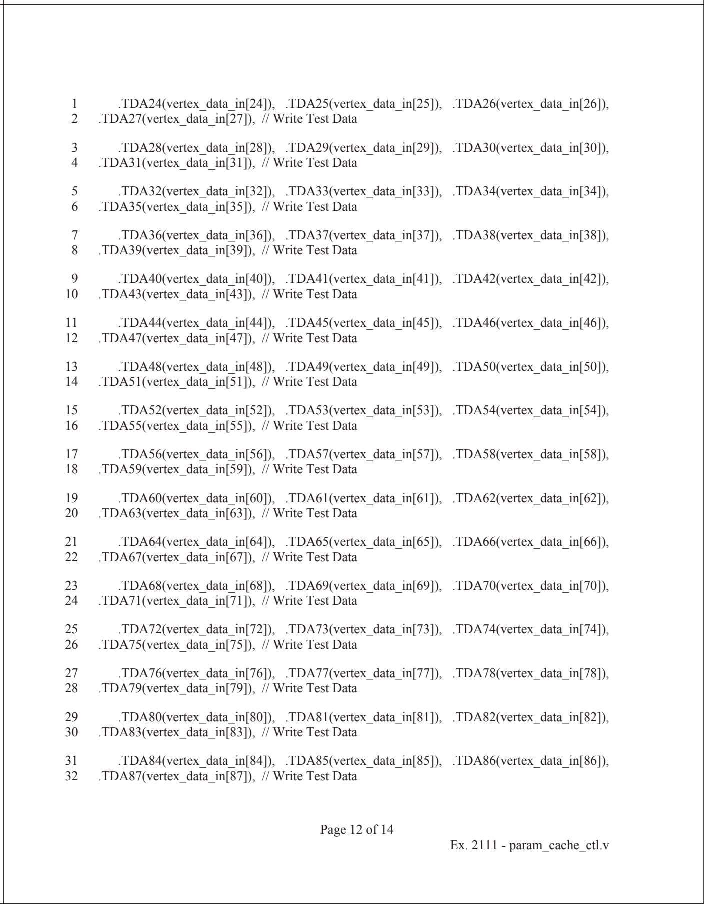
```
    .QB120(vertex_data_out[120]), .QB121(vertex_data_out[121]),
.QB122(vertex_data_out[122]), .QB123(vertex_data_out[123]), // Read Data

    .QB124(vertex_data_out[124]), .QB125(vertex_data_out[125]),
.QB126(vertex_data_out[126]), .QB127(vertex_data_out[127]), // Read Data

    // WRITE INTERFACE

    .CLKA(iSCLK), // Write Clock

    .WEA(pc_wr_en), // Write enable

    .MEA(vdd), // Memory enable

    .ADRA0(pc_wr_addr[0]), .ADRA1(pc_wr_addr[1]), .ADRA2(pc_wr_addr[2]),
.ADRA3(pc_wr_addr[3]), // Write Address

    .ADRA4(pc_wr_addr[4]), .ADRA5(pc_wr_addr[5]), .ADRA6(pc_wr_addr[6]), // Write
Address

    .DA0(vertex_data_in[0]), .DA1(vertex_data_in[1]), .DA2(vertex_data_in[2]),
.DA3(vertex_data_in[3]), // Write Data

    .DA4(vertex_data_in[4]), .DA5(vertex_data_in[5]), .DA6(vertex_data_in[6]),
.DA7(vertex_data_in[7]), // Write Data

    .DA8(vertex_data_in[8]), .DA9(vertex_data_in[9]), .DA10(vertex_data_in[10]),
.DA11(vertex_data_in[11]), // Write Data

    .DA12(vertex_data_in[12]), .DA13(vertex_data_in[13]), .DA14(vertex_data_in[14]),
.DA15(vertex_data_in[15]), // Write Data

    .DA16(vertex_data_in[16]), .DA17(vertex_data_in[17]), .DA18(vertex_data_in[18]),
.DA19(vertex_data_in[19]), // Write Data

    .DA20(vertex_data_in[20]), .DA21(vertex_data_in[21]), .DA22(vertex_data_in[22]),
.DA23(vertex_data_in[23]), // Write Data

    .DA24(vertex_data_in[24]), .DA25(vertex_data_in[25]), .DA26(vertex_data_in[26]),
.DA27(vertex_data_in[27]), // Write Data

    .DA28(vertex_data_in[28]), .DA29(vertex_data_in[29]), .DA30(vertex_data_in[30]),
.DA31(vertex_data_in[31]), // Write Data

    .DA32(vertex_data_in[32]), .DA33(vertex_data_in[33]), .DA34(vertex_data_in[34]),
.DA35(vertex_data_in[35]), // Write Data

    .DA36(vertex_data_in[36]), .DA37(vertex_data_in[37]), .DA38(vertex_data_in[38]),
.DA39(vertex_data_in[39]), // Write Data
```



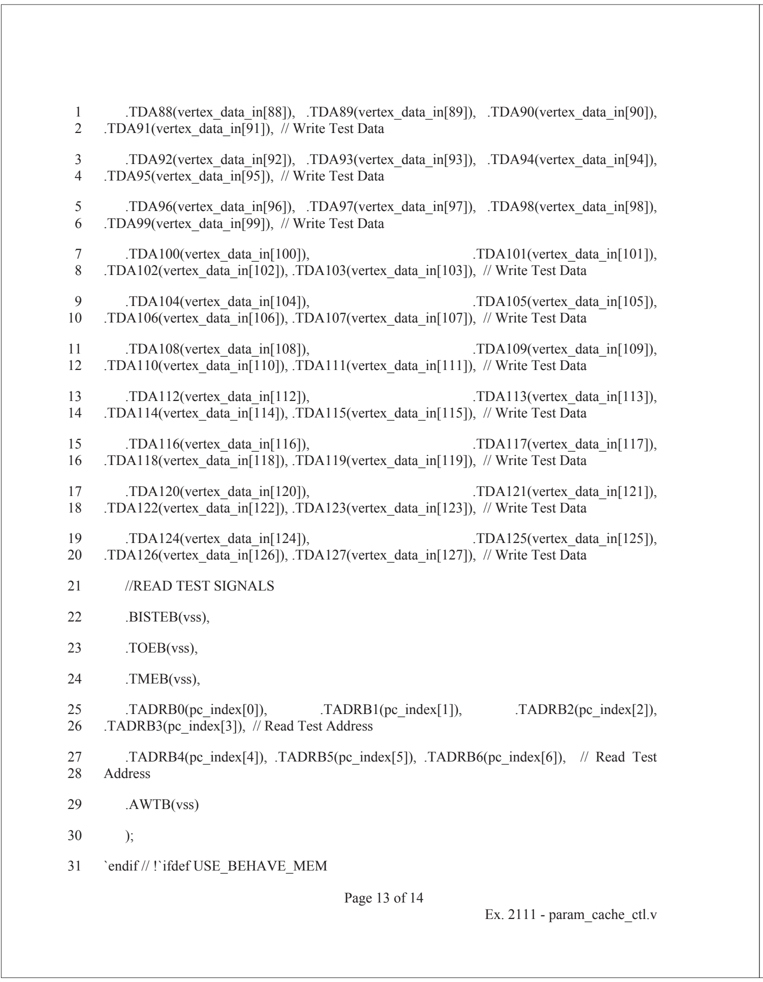
```
    .DA40(vertex_data_in[40]), .DA41(vertex_data_in[41]), .DA42(vertex_data_in[42]),
.DA43(vertex_data_in[43]), // Write Data

    .DA44(vertex_data_in[44]), .DA45(vertex_data_in[45]), .DA46(vertex_data_in[46]),
.DA47(vertex_data_in[47]), // Write Data

    .DA48(vertex_data_in[48]), .DA49(vertex_data_in[49]), .DA50(vertex_data_in[50]),
.DA51(vertex_data_in[51]), // Write Data

    .DA52(vertex_data_in[52]), .DA53(vertex_data_in[53]), .DA54(vertex_data_in[54]),
.DA55(vertex_data_in[55]), // Write Data

    .DA56(vertex_data_in[56]), .DA57(vertex_data_in[57]), .DA58(vertex_data_in[58]),
.DA59(vertex_data_in[59]), // Write Data

    .DA60(vertex_data_in[60]), .DA61(vertex_data_in[61]), .DA62(vertex_data_in[62]),
.DA63(vertex_data_in[63]), // Write Data

    .DA64(vertex_data_in[64]), .DA65(vertex_data_in[65]), .DA66(vertex_data_in[66]),
.DA67(vertex_data_in[67]), // Write Data

    .DA68(vertex_data_in[68]), .DA69(vertex_data_in[69]), .DA70(vertex_data_in[70]),
.DA71(vertex_data_in[71]), // Write Data

    .DA72(vertex_data_in[72]), .DA73(vertex_data_in[73]), .DA74(vertex_data_in[74]),
.DA75(vertex_data_in[75]), // Write Data

    .DA76(vertex_data_in[76]), .DA77(vertex_data_in[77]), .DA78(vertex_data_in[78]),
.DA79(vertex_data_in[79]), // Write Data

    .DA80(vertex_data_in[80]), .DA81(vertex_data_in[81]), .DA82(vertex_data_in[82]),
.DA83(vertex_data_in[83]), // Write Data

    .DA84(vertex_data_in[84]), .DA85(vertex_data_in[85]), .DA86(vertex_data_in[86]),
.DA87(vertex_data_in[87]), // Write Data

    .DA88(vertex_data_in[88]), .DA89(vertex_data_in[89]), .DA90(vertex_data_in[90]),
.DA91(vertex_data_in[91]), // Write Data

    .DA92(vertex_data_in[92]), .DA93(vertex_data_in[93]), .DA94(vertex_data_in[94]),
.DA95(vertex_data_in[95]), // Write Data

    .DA96(vertex_data_in[96]), .DA97(vertex_data_in[97]), .DA98(vertex_data_in[98]),
.DA99(vertex_data_in[99]), // Write Data

    .DA100(vertex_data_in[100]), .DA101(vertex_data_in[101]),
.DA102(vertex_data_in[102]), .DA103(vertex_data_in[103]), // Write Data
```



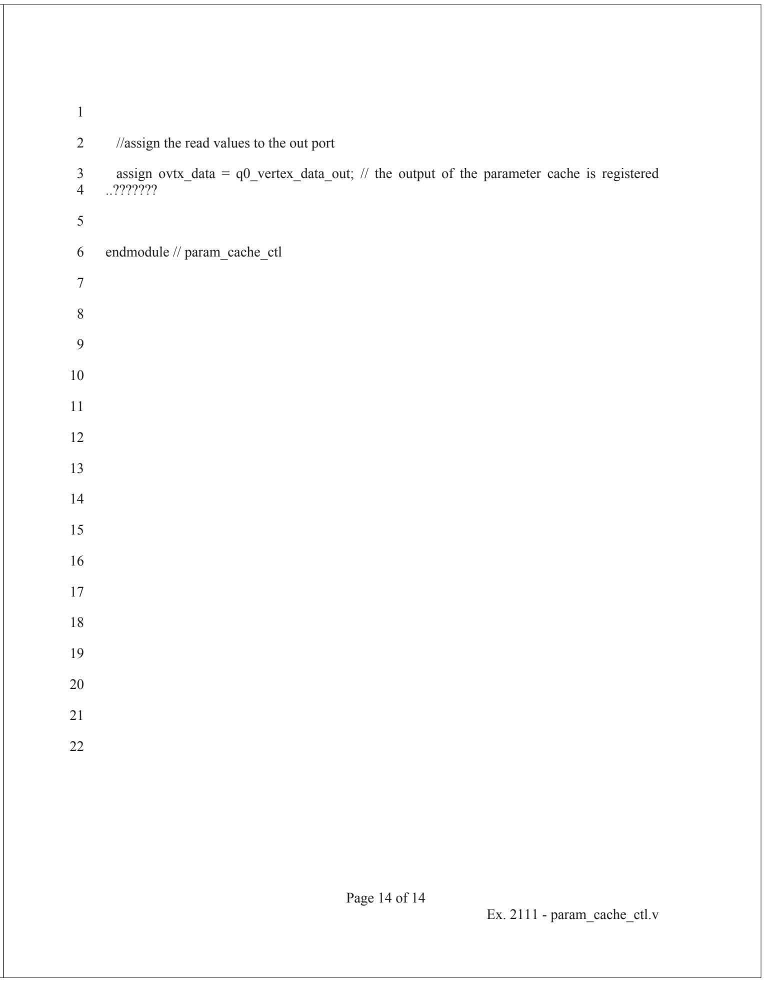
```
    .DA104(vertex_data_in[104]), .DA105(vertex_data_in[105]),
.DA106(vertex_data_in[106]), .DA107(vertex_data_in[107]), // Write Data

    .DA108(vertex_data_in[108]), .DA109(vertex_data_in[109]),
.DA110(vertex_data_in[110]), .DA111(vertex_data_in[111]), // Write Data

    .DA112(vertex_data_in[112]), .DA113(vertex_data_in[113]),
.DA114(vertex_data_in[114]), .DA115(vertex_data_in[115]), // Write Data

    .DA116(vertex_data_in[116]), .DA117(vertex_data_in[117]),
.DA118(vertex_data_in[118]), .DA119(vertex_data_in[119]), // Write Data

    .DA120(vertex_data_in[120]), .DA121(vertex_data_in[121]),
.DA122(vertex_data_in[122]), .DA123(vertex_data_in[123]), // Write Data

    .DA124(vertex_data_in[124]), .DA125(vertex_data_in[125]),
.DA126(vertex_data_in[126]), .DA127(vertex_data_in[127]), // Write Data

    // WRITE TEST SIGNALS

    .BISTEA(vss),

    .TWEA(vss),

    .TMEA(vss),

    .TADRA0(pc_wr_addr[0]), .TADRA1(pc_wr_addr[1]), .TADRA2(pc_wr_addr[2]),
.TADRA3(pc_wr_addr[3]), // Write Test Address

    .TADRA4(pc_wr_addr[4]), .TADRA5(pc_wr_addr[5]), .TADRA6(pc_wr_addr[6]), //
Write Test Address

    .TDA0(vertex_data_in[0]), .TDA1(vertex_data_in[1]), .TDA2(vertex_data_in[2]),
.TDA3(vertex_data_in[3]), // Write Test Data

    .TDA4(vertex_data_in[4]), .TDA5(vertex_data_in[5]), .TDA6(vertex_data_in[6]),
.TDA7(vertex_data_in[7]), // Write Test Data

    .TDA8(vertex_data_in[8]), .TDA9(vertex_data_in[9]), .TDA10(vertex_data_in[10]),
.TDA11(vertex_data_in[11]), // Write Test Data

    .TDA12(vertex_data_in[12]), .TDA13(vertex_data_in[13]), .TDA14(vertex_data_in[14]),
.TDA15(vertex_data_in[15]), // Write Test Data

    .TDA16(vertex_data_in[16]), .TDA17(vertex_data_in[17]), .TDA18(vertex_data_in[18]),
.TDA19(vertex_data_in[19]), // Write Test Data

    .TDA20(vertex_data_in[20]), .TDA21(vertex_data_in[21]), .TDA22(vertex_data_in[22]),
.TDA23(vertex_data_in[23]), // Write Test Data
```



```
    .TDA24(vertex_data_in[24]), .TDA25(vertex_data_in[25]), .TDA26(vertex_data_in[26]),
.TDA27(vertex_data_in[27]), // Write Test Data

    .TDA28(vertex_data_in[28]), .TDA29(vertex_data_in[29]), .TDA30(vertex_data_in[30]),
.TDA31(vertex_data_in[31]), // Write Test Data

    .TDA32(vertex_data_in[32]), .TDA33(vertex_data_in[33]), .TDA34(vertex_data_in[34]),
.TDA35(vertex_data_in[35]), // Write Test Data

    .TDA36(vertex_data_in[36]), .TDA37(vertex_data_in[37]), .TDA38(vertex_data_in[38]),
.TDA39(vertex_data_in[39]), // Write Test Data

    .TDA40(vertex_data_in[40]), .TDA41(vertex_data_in[41]), .TDA42(vertex_data_in[42]),
.TDA43(vertex_data_in[43]), // Write Test Data

    .TDA44(vertex_data_in[44]), .TDA45(vertex_data_in[45]), .TDA46(vertex_data_in[46]),
.TDA47(vertex_data_in[47]), // Write Test Data

    .TDA48(vertex_data_in[48]), .TDA49(vertex_data_in[49]), .TDA50(vertex_data_in[50]),
.TDA51(vertex_data_in[51]), // Write Test Data

    .TDA52(vertex_data_in[52]), .TDA53(vertex_data_in[53]), .TDA54(vertex_data_in[54]),
.TDA55(vertex_data_in[55]), // Write Test Data

    .TDA56(vertex_data_in[56]), .TDA57(vertex_data_in[57]), .TDA58(vertex_data_in[58]),
.TDA59(vertex_data_in[59]), // Write Test Data

    .TDA60(vertex_data_in[60]), .TDA61(vertex_data_in[61]), .TDA62(vertex_data_in[62]),
.TDA63(vertex_data_in[63]), // Write Test Data

    .TDA64(vertex_data_in[64]), .TDA65(vertex_data_in[65]), .TDA66(vertex_data_in[66]),
.TDA67(vertex_data_in[67]), // Write Test Data

    .TDA68(vertex_data_in[68]), .TDA69(vertex_data_in[69]), .TDA70(vertex_data_in[70]),
.TDA71(vertex_data_in[71]), // Write Test Data

    .TDA72(vertex_data_in[72]), .TDA73(vertex_data_in[73]), .TDA74(vertex_data_in[74]),
.TDA75(vertex_data_in[75]), // Write Test Data

    .TDA76(vertex_data_in[76]), .TDA77(vertex_data_in[77]), .TDA78(vertex_data_in[78]),
.TDA79(vertex_data_in[79]), // Write Test Data

    .TDA80(vertex_data_in[80]), .TDA81(vertex_data_in[81]), .TDA82(vertex_data_in[82]),
.TDA83(vertex_data_in[83]), // Write Test Data

    .TDA84(vertex_data_in[84]), .TDA85(vertex_data_in[85]), .TDA86(vertex_data_in[86]),
.TDA87(vertex_data_in[87]), // Write Test Data
```



```
    .TDA88(vertex_data_in[88]),  .TDA89(vertex_data_in[89]),  .TDA90(vertex_data_in[90]), .TDA91(vertex_data_in[91]),  // Write Test Data
    .TDA92(vertex_data_in[92]),  .TDA93(vertex_data_in[93]),  .TDA94(vertex_data_in[94]), .TDA95(vertex_data_in[95]),  // Write Test Data
    .TDA96(vertex_data_in[96]),  .TDA97(vertex_data_in[97]),  .TDA98(vertex_data_in[98]), .TDA99(vertex_data_in[99]),  // Write Test Data
    .TDA100(vertex_data_in[100]),     .TDA101(vertex_data_in[101]), .TDA102(vertex_data_in[102]), .TDA103(vertex_data_in[103]),  // Write Test Data
    .TDA104(vertex_data_in[104]),     .TDA105(vertex_data_in[105]), .TDA106(vertex_data_in[106]), .TDA107(vertex_data_in[107]),  // Write Test Data
    .TDA108(vertex_data_in[108]),     .TDA109(vertex_data_in[109]), .TDA110(vertex_data_in[110]), .TDA111(vertex_data_in[111]),  // Write Test Data
    .TDA112(vertex_data_in[112]),     .TDA113(vertex_data_in[113]), .TDA114(vertex_data_in[114]), .TDA115(vertex_data_in[115]),  // Write Test Data
    .TDA116(vertex_data_in[116]),     .TDA117(vertex_data_in[117]), .TDA118(vertex_data_in[118]), .TDA119(vertex_data_in[119]),  // Write Test Data
    .TDA120(vertex_data_in[120]),     .TDA121(vertex_data_in[121]), .TDA122(vertex_data_in[122]), .TDA123(vertex_data_in[123]),  // Write Test Data
    .TDA124(vertex_data_in[124]),     .TDA125(vertex_data_in[125]), .TDA126(vertex_data_in[126]), .TDA127(vertex_data_in[127]),  // Write Test Data
    //READ TEST SIGNALS
    .BISTEB(vss),
    .TOEB(vss),
    .TMEB(vss),
    .TADRB0(pc_index[0]),     .TADRB1(pc_index[1]),     .TADRB2(pc_index[2]), .TADRB3(pc_index[3]),  // Read Test Address
    .TADRB4(pc_index[4]),  .TADRB5(pc_index[5]),  .TADRB6(pc_index[6]),   // Read Test Address
    .AWTB(vss)
    );
`endif // !`ifdef USE_BEHAVE_MEM
```



```

   //assign the read values to the out port
   assign ovtx_data = q0_vertex_data_out; // the output of the parameter cache is registered ..???????


endmodule // param_cache_ctl
```

```
//$Id$
//$Change$
//                -*- Mode: Verilog -*-
// Filename       : shader.v
// Description    : This module represents the Shader Pipe unit.
//                : There are 4 instances of this module in the chip.
//                : Each shader pipe includes four vector units.
//                : Each vector unit has four GPR/MAC instances and one scalar unit.
// Author         : Andi Skende
// Created On     : Fri Nov 16 18:48:25 2001
// Last Modified By: .
// Last Modified On: .
// Update Count   : 0
// Status         : Unknown, Use with caution!


`timescale 1ns / 1ps

module sp(/*AUTOARG*/
  // Outputs
  SP_TP_fetch_addr0, SP_TP_fetch_addr1, SP_TP_fetch_addr2,
  SP_TP_fetch_addr3, SP_SX_data0, SP_SX_data1, SP_SX_data2,
  SP_SX_data3, SP_SQ_const_addr, SP_SQ_valid, SP_SQ_kill_vect,
  SP_SX_exp_pvalid, SP_SX_exporting, SP_SX_exp_alu_id,
  SP_SX_exp_dest,
  // Inputs
```

```
sclk_global, srst, TP_SP_data0, TP_SP_data1, TP_SP_data2,
TP_SP_data3, TP_SP_data_valid, TP_SP_gpr_dst, TP_SP_gpr_cmask,
SQ_SP_instruct_start, SQ_SP_instruct, SQ_SP_stall,
SQ_SP_exp_pvalid, SQ_SP_exporting, SQ_SP_exp_id, SQ_SP_const,
SQ_SP_gpr_wr_addr, SQ_SP_gpr_rd_addr, SQ_SP_gpr_rd_en,
SQ_SP_gpr_wr_en, SQ_SP_gpr_phase_mux, SQ_SP_channel_mask,
SQ_SP_pix_mask, SQ_SP_gpr_input_mux, SQ_SP_auto_count, SC_SP_data,
SC_SP_valid, SC_SP_type, SC_SP_last_quad, SQ_SP_vsr_data,
SQ_SP_vsr_double, SQ_SP_vsr_valid, SQ_SP_vsr_read,
SQ_SP_interp_prim_type, SQ_SP_interp_ijline, SQ_SP_interp_mode,
SQ_SP_interp_valid, SQ_SP_interp_buff_swap, SQ_SP_interp_gen_i0,
CG_SP_pm_enb, SX_SP_vtx_data0, SX_SP_vtx_data1, SX_SP_vtx_data2
);


input       sclk_global;
input   srst;

wire    sclk ;
assign      sclk = sclk_global;
//-----------------------------------------------------------------------------------
//SHADER(SP)-TEXTURE(TP)
//These buses represent the texture fetch request data.
//One 96 bit bus comming out of each vector unit....4 X 96 bits = 384 bits comming
//out of each shader pipe
```

```
  //-----------------------------------------------------------------------------------
  output                    [95:0]                    SP_TP_fetch_addr0,
SP_TP_fetch_addr1,SP_TP_fetch_addr2,SP_TP_fetch_addr3;
  wire [95:0] sp_fetch_addr0, sp_fetch_addr1,sp_fetch_addr2,sp_fetch_addr3;
  wire [95:0] q_sp_fetch_addr0, q_sp_fetch_addr1,q_sp_fetch_addr2,q_sp_fetch_addr3;

  ati_dff_out #(96) usp_fetch_addr0(sclk,sp_fetch_addr0,q_sp_fetch_addr0);
  ati_dff_out #(96) usp_fetch_addr1(sclk,sp_fetch_addr1,q_sp_fetch_addr1);
  ati_dff_out #(96) usp_fetch_addr2(sclk,sp_fetch_addr2,q_sp_fetch_addr2);
  ati_dff_out #(96) usp_fetch_addr3(sclk,sp_fetch_addr3,q_sp_fetch_addr3);


  assign      SP_TP_fetch_addr0 = q_sp_fetch_addr0;
  assign      SP_TP_fetch_addr1 = q_sp_fetch_addr1;
  assign      SP_TP_fetch_addr2 = q_sp_fetch_addr2;
  assign      SP_TP_fetch_addr3 = q_sp_fetch_addr3;

  //-------------------------------------------------------------------------------------
  //SHADER(SP) - SX(SHADER EXPORT)
  //This interface represents pixel/parameter data being exported out of the shader pipe
  //into the SX block
  //-------------------------------------------------------------------------------------
  output [127:0] SP_SX_data0, SP_SX_data1,SP_SX_data2,SP_SX_data3;
  wire [127:0]   q_sp_sx_data0, q_sp_sx_data1 , q_sp_sx_data2 , q_sp_sx_data3;
  wire [127:0]   osp_sx_data0, osp_sx_data1 , osp_sx_data2 , osp_sx_data3;

```

```
ati_dff_out #(128) usp_sx_data0(sclk, osp_sx_data0,q_sp_sx_data0);
ati_dff_out #(128) usp_sx_data1(sclk, osp_sx_data1,q_sp_sx_data1);
ati_dff_out #(128) usp_sx_data2(sclk, osp_sx_data2,q_sp_sx_data2);
ati_dff_out #(128) usp_sx_data3(sclk, osp_sx_data3,q_sp_sx_data3);

//export data going out to SX (shader export)
assign      SP_SX_data0 = q_sp_sx_data0;
assign      SP_SX_data1 = q_sp_sx_data1;
assign      SP_SX_data2 = q_sp_sx_data2;
assign      SP_SX_data3 = q_sp_sx_data3;

//----------------------------------------------------------------------------------
//SP-SQ constant address load.
//The transfer on this interface is done via a MOVA instruction used to calculate the
//address that SQ uses to access the Constant Store.
//----------------------------------------------------------------------------------
output [35:0]  SP_SQ_const_addr;
output [0:0]   SP_SQ_valid ;



//--------------------------------------------------------------------------------//
//Kill return interface-----------------------------------------------------------//
//--------------------------------------------------------------------------------//
output [3:0]   SP_SQ_kill_vect;

```

```
//--------------------------------------------------------------------------------------
//TEXTURE(TP)-SHADER(SP)
//This interface represents the texture fetch return data
//TP_SP_dst and TP_SP_cmask represent the GPR destination of this data
//and a mask value per 32 channel respectivly.
//TP_SP_data* is currently defined as a 128 bit bus.
//This bus will propably be reduced to 64 bit.
//--------------------------------------------------------------------------------------
input [127:0]  TP_SP_data0,TP_SP_data1,TP_SP_data2,TP_SP_data3;
input [0:0]      TP_SP_data_valid;
input [6:0]      TP_SP_gpr_dst;
input [3:0]      TP_SP_gpr_cmask;

wire [127:0]   q_tp_data0, q_tp_data1,q_tp_data2,q_tp_data3;
wire [6:0]       q_tp_gpr_dst;
wire [3:0]       q_tp_gpr_cmask;
wire [0:0]       q_tp_data_valid;

//registering the inputs from Texture pipe
ati_dff_in #(128) tp_data0(sclk,TP_SP_data0,q_tp_data0);
ati_dff_in #(128) tp_data1(sclk,TP_SP_data1,q_tp_data1);
ati_dff_in #(128) tp_data2(sclk,TP_SP_data2,q_tp_data2);
ati_dff_in #(128) tp_data3(sclk,TP_SP_data3,q_tp_data3);
ati_dff_in #(7)   tp_gpr_dst(sclk, TP_SP_gpr_dst, q_tp_gpr_dst);
ati_dff_in #(4)   tp_gpr_cmask(sclk, TP_SP_gpr_cmask, q_tp_gpr_cmask);
```

```
ati_dff_in #(1)   tp_data_valid(sclk,TP_SP_data_valid ,q_tp_data_valid);



//-----------------------------------------------------
//SEQUENCER(SQ)-SHADER(SP)
//the controls needed for writting
//and reading the register files(GPRs).
//ALU Instruction related data is also
//present in this interface
//------------------------------------------------------
input [0:0]      SQ_SP_instruct_start;
input [20:0]   SQ_SP_instruct;
input [0:0]      SQ_SP_stall;
input [3:0]      SQ_SP_exp_pvalid;
input [0:0]      SQ_SP_exporting ;
input [0:0]      SQ_SP_exp_id;

wire [0:0]       q_sq_instruct_start;
wire [20:0]     q_sq_instruct;
wire [0:0]       q_sq_stall;
wire [3:0]       q_sq_exp_pvalid;
wire [0:0]       q_sq_exporting ;
wire [0:0]       q_sq_exp_alu_id;

```

```


ati_dff_in #(1)  sq_instruct_start(sclk,SQ_SP_instruct_start,q_sq_instruct_start);
ati_dff_in #(21) sq_instruct(sclk,SQ_SP_instruct,q_sq_instruct);
ati_dff_in #(1)  sq_stall(sclk,SQ_SP_stall,q_sq_stall);
ati_dff_in #(4)  sq_exp_pvalid(sclk,SQ_SP_exp_pvalid,q_sq_exp_pvalid);
ati_dff_in #(1)  sq_exporting(sclk,SQ_SP_exporting,q_sq_exporting);
ati_dff_in #(1)  sq_exp_alu_id(sclk,SQ_SP_exp_id,q_sq_exp_alu_id);

//------------------------------------------------------------------------//
//The three buses below are pipelined through and outputed to SX
//------------------------------------------------------------------------//

output [3:0]   SP_SX_exp_pvalid;
output [0:0]   SP_SX_exporting ;
output [0:0]   SP_SX_exp_alu_id;
output [5:0]   SP_SX_exp_dest;

wire [3:0]       sp_exp_pvalid;
wire [0:0]       sp_exporting ;
wire [0:0]       sp_exp_alu_id;
wire [5:0]       sp_exp_dst;

wire [3:0]       q_sp_exp_pvalid;
wire [0:0]       q_sp_exporting ;
```

```
wire [0:0]       q_sp_exp_alu_id;
wire [5:0]       q_sp_exp_dst;

ati_dff_out #(4) usp_exp_pvalid(sclk,sp_exp_pvalid,q_sp_exp_pvalid);
ati_dff_out #(1) usp_exporting(sclk,sp_exporting,q_sp_exporting);
ati_dff_out #(1) usp_exp_alu_id(sclk,sp_exp_alu_id,q_sp_exp_alu_id);
ati_dff_out #(6) usp_exp_dst(sclk,sp_exp_dst,q_sp_exp_dst);

assign           SP_SX_exp_pvalid = q_sp_exp_pvalid;
assign           SP_SX_exporting = q_sp_exporting ;
assign           SP_SX_exp_alu_id = q_sp_exp_alu_id;
assign           SP_SX_exp_dest = q_sp_exp_dst;

//--------------------------------------------------------/
//SEQUENCER(SQ)-SHADER(SP)
//Constant Broadcast interface
//--------------------------------------------------------/
input [127:0]  SQ_SP_const;
wire [127:0]   q_sq_const;

ati_dff_in #(128)  sq_const(sclk,SQ_SP_const,q_sq_const);


//---------------------------------------------------------/
//SEQUENCER(SQ)-SHADER(SP)
```

```
//GPR control and auto-counter interface
//--------------------------------------------------------/
input [6:0]      SQ_SP_gpr_wr_addr;
input [6:0]      SQ_SP_gpr_rd_addr;
input [0:0]      SQ_SP_gpr_rd_en,SQ_SP_gpr_wr_en;               //these to read/write enable
signals
                                                               //are used to enable the TP - GPR write path also
input [1:0]      SQ_SP_gpr_phase_mux;                       //control into GPR write port
input [3:0]      SQ_SP_channel_mask;
input [3:0]      SQ_SP_pix_mask;
input [1:0]      SQ_SP_gpr_input_mux;
input [11:0]   SQ_SP_auto_count;


wire [6:0]       q_sq_gpr_wr_addr;
wire [6:0]       q_sq_gpr_rd_addr;
wire [0:0]       q_sq_gpr_rd_en,q_sq_gpr_wr_en;             //these to read/write enable signals
                                                             //are used to enable the TP - GPR write path also
wire [1:0]       q_sq_gpr_phase_mux;                    //control into GPR write port
wire [3:0]       q_sq_channel_mask;
wire [3:0]       q_sq_pix_mask;
wire [1:0]       q_sq_gpr_input_mux;
wire [11:0]      q_sq_auto_count;


ati_dff_in #(7) sq_gpr_wr_addr(sclk,SQ_SP_gpr_wr_addr,q_sq_gpr_wr_addr);
```

```
ati_dff_in #(7) sq_gpr_rd_addr(sclk,SQ_SP_gpr_rd_addr,q_sq_gpr_rd_addr);
ati_dff_in #(1) sq_gpr_rd_en(sclk,SQ_SP_gpr_rd_en,q_sq_gpr_rd_en);
ati_dff_in #(1) sq_gpr_wr_en(sclk,SQ_SP_gpr_wr_en,q_sq_gpr_wr_en);
ati_dff_in #(2) sq_gpr_phase_mux(sclk,SQ_SP_gpr_phase_mux,q_sq_gpr_phase_mux);
ati_dff_in #(4) sq_channel_mask(sclk,SQ_SP_channel_mask,q_sq_channel_mask);
ati_dff_in #(4) sq_pix_mask(sclk,SQ_SP_pix_mask,q_sq_pix_mask);
ati_dff_in #(2) sq_gpr_input_mux(sclk,SQ_SP_gpr_input_mux,q_sq_gpr_input_mux);
ati_dff_in #(12) sq_auto_count(sclk,SQ_SP_auto_count,q_sq_auto_count);

//--------------------------------------------------------
//Scan Converter (SC) - Shader Pipe (SP)
//IJ interface
//----------------------------------------------------------
input [99:0]   SC_SP_data;
input [0:0]      SC_SP_valid; //quad write mask
input [1:0]      SC_SP_type; //specifies the interpolation type ....center vs. centroid vs. XY
input [0:0]      SC_SP_last_quad;

wire [99:0]      q_sc_data;
wire [0:0]       q_sc_valid; //quad write mask
wire [1:0]       q_sc_type; //specifies the interpolation type ....center vs. centroid vs. XY
wire [0:0]       q_sc_last_quad;

ati_dff_in #(100) sc_data(sclk,SC_SP_data,q_sc_data);
ati_dff_in #(1) sc_valid(sclk,SC_SP_valid,q_sc_valid);
```

```
ati_dff_in #(2) sc_type(sclk,SC_SP_type,q_sc_type);
ati_dff_in #(1) sc_last_quad(sclk,SC_SP_last_quad,q_sc_last_quad);

//----------------------------------------------------------//
//Staging registers control interface.
//-----------------------------------------------------------//
input [95:0]   SQ_SP_vsr_data;
input [0:0]      SQ_SP_vsr_double;
input [0:0]      SQ_SP_vsr_valid;
input [0:0]      SQ_SP_vsr_read;

wire [95:0]      q_sq_vsr_data;
wire [0:0]       q_sq_vsr_double;
wire [0:0]       q_sq_vsr_valid;
wire [0:0]       q_sq_vsr_read;

ati_dff_in #(96) sq_vsr_data(sclk,SQ_SP_vsr_data,q_sq_vsr_data);
ati_dff_in #(1) sq_vsr_double(sclk,SQ_SP_vsr_double,q_sq_vsr_double);
ati_dff_in #(1) sq_vsr_valid(sclk,SQ_SP_vsr_valid,q_sq_vsr_valid);
ati_dff_in #(1) sq_vsr_read(sclk,SQ_SP_vsr_read,q_sq_vsr_read);

//------------------------------------------------------
//SQ-SP
//Interpolation control interface
//------------------------------------------------------
```

```
input [2:0]      SQ_SP_interp_prim_type;
input [1:0]      SQ_SP_interp_ijline;
input [0:0]      SQ_SP_interp_mode;
input [0:0]      SQ_SP_interp_valid;
input [0:0]      SQ_SP_interp_buff_swap;
input [0:0]      SQ_SP_interp_gen_i0 ;

wire [2:0]       q_sq_interp_prim_type;
wire [1:0]       q_sq_interp_ijline;
wire [0:0]       q_sq_interp_mode;
wire [0:0]       q_sq_interp_buff_swap;
wire [0:0]       q_sq_interp_gen_i0 ;
wire [0:0]       q_sq_interp_valid;

ati_dff_in #(3) usq_interp_prim_type(sclk,SQ_SP_interp_prim_type,q_sq_interp_prim_type);
ati_dff_in #(2) usq_interp_ijline(sclk,SQ_SP_interp_ijline,q_sq_interp_ijline);
ati_dff_in #(1) usq_interp_mode(sclk,SQ_SP_interp_mode,q_sq_interp_mode);
ati_dff_in                                                                                   #(1)
usq_interp_buff_swap(sclk,SQ_SP_interp_buff_swap,q_sq_interp_buff_swap);
ati_dff_in #(1) usq_interp_gen_i0(sclk,SQ_SP_interp_gen_i0,q_sq_interp_gen_i0);
ati_dff_in #(1) usq_interp_valid(sclk, SQ_SP_interp_valid,q_sq_interp_valid);


//---------------------------------------------------------------//
//Power managment control interface
//---------------------------------------------------------------//
```

```
input [0:0]      CG_SP_pm_enb;
wire [0:0]       q_cg_sp_pm_enb;
ati_dff_in  #(1)  ucg_sp_pm_enb(sclk,CG_SP_pm_enb,q_cg_sp_pm_enb);

//-------------------------------------------------------------------
//SX - SP Interpolators : Parameter Cache return bus
//data0 represents P0
//data1 represents P1-P0
//data2 represents P2-P0
//remember : the difference engines are in the SX blocks
//-------------------------------------------------------------------

input [127:0]  SX_SP_vtx_data0;
input [131:0]  SX_SP_vtx_data1, SX_SP_vtx_data2;

wire [127:0]   q_sx_vtx_data0;
wire [131:0]   q_sx_vtx_data1, q_sx_vtx_data2;

//registering the inputs

ati_dff_in #(128) sx_vtx_data0(sclk,SX_SP_vtx_data0,q_sx_vtx_data0);
ati_dff_in #(132) sx_vtx_data1(sclk,SX_SP_vtx_data1,q_sx_vtx_data1);
ati_dff_in #(132) sx_vtx_data2(sclk,SX_SP_vtx_data2,q_sx_vtx_data2);
```

```
 wire [127:0]   Interpolated0, Interpolated1,Interpolated2,Interpolated3;
 wire [95:0]    VertexIndex0, VertexIndex1,VertexIndex2, VertexIndex3;


 //------------------------------------------------------------------------------------------//
 //Interpolation Units---------------------------------------------------------------------------//
 //------------------------------------------------------------------------------------------//
 interpolator uinterpolator(.oInterpolated0(Interpolated0), .oInterpolated1(Interpolated1),
                            .oInterpolated2(Interpolated2), .oInterpolated3(Interpolated3),
                            .sx_sp_vtx_data0(q_sx_vtx_data0),

.sx_sp_vtx_delta10(q_sx_vtx_data1),.sx_sp_vtx_delta20(q_sx_vtx_data2),

.sq_sp_interp_ijline(q_sq_interp_ijline),.sq_sp_interp_valid(q_sq_interp_valid),
                            .sq_sp_interp_buff_swap(q_sq_interp_buff_swap),

.sc_sp_data(q_sc_data),.sc_sp_valid(q_sc_valid),.sq_sp_interp_mode(q_sq_interp_mode),
                            .sc_sp_type(q_sc_type),.sc_sp_quad_last(q_sc_last_quad),
                      .sclk(sclk),.srst(srst));


 //------------------------------------------------------------------------------------------//
 //Vertex Indices Staging registers and Control
 //------------------------------------------------------------------------------------------//
 sp_vsr_ctl usp_vsr_ctl(.ovtx_index0(VertexIndex0), .ovtx_index1(VertexIndex1),
                          .ovtx_index2(VertexIndex2), .ovtx_index3(VertexIndex3),
```

```
                  .isq_vsr_data(q_sq_vsr_data), .isq_vsr_double(q_sq_vsr_double),
                  .isq_vsr_valid(q_sq_vsr_valid), .isq_vsr_read(q_sq_vsr_read),
                  .sclk(sclk),.srst(srst)
                  );

//--------------------------------------------------------------------
//Instantiation of 4 Vector units (vector.v module)
//--------------------------------------------------------------------

vector uvector0(//outputs
              .sp_sx_data(osp_sx_data0),
              .sp_sx_exporting(sp_exporting),
              .sp_sx_exp_dst(sp_exp_dst),
          .sp_sx_exp_alu_id(sp_exp_alu_id),
              .sp_sx_exp_pvalid(sp_exp_pvalid),
              .sp_tp_data(sp_fetch_addr0),

              //inputs
              .sq_sp_instruct_start(q_sq_instruct_start),
              .sq_sp_instruct(q_sq_instruct),.sq_sp_stall(q_sq_stall),
              .sclk(sclk), .srst(srst),
              .sq_sp_wr_addr(q_sq_gpr_wr_addr), .sq_sp_gpr_rd_addr(q_sq_gpr_rd_addr),

.sq_sp_mem_rd_ena(q_sq_gpr_rd_en),.sq_sp_mem_wr_ena(q_sq_gpr_wr_en),.sq_sp_wr_ena(
q_sq_gpr_wr_en),
```

```
              .sq_sp_gpr_phase_mux(q_sq_gpr_phase_mux),
.sq_sp_channel_mask(q_sq_channel_mask),
              .sq_sp_pixel_mask(q_sq_pix_mask),
.sq_sp_gpr_input_mux(q_sq_gpr_input_mux),
              .iInterpolated(Interpolated0),// iAutoCount,
              .iVertexIndices(VertexIndex0),
              .sq_sp_constant(q_sq_const),
              .tp_sp_data(q_tp_data0),.tp_sp_gpr_dst(q_tp_gpr_dst),
              .tp_sp_gpr_cmask(q_tp_gpr_cmask),.tp_sp_data_valid(q_tp_data_valid),
              .sq_sp_exp_pvalid(q_sq_exp_pvalid),
              .sq_sp_exporting(q_sq_exporting),
              .sq_sp_exp_alu_id(q_sq_exp_alu_id)
              );


 vector uvector1(.sp_sx_data(osp_sx_data1),
              .sp_tp_data(sp_fetch_addr1),
              .sq_sp_instruct_start(q_sq_instruct_start),
              .sq_sp_instruct(q_sq_instruct),.sq_sp_stall(q_sq_stall),
              .sclk(sclk), .srst(srst),
              .sq_sp_wr_addr(q_sq_gpr_wr_addr), .sq_sp_gpr_rd_addr(q_sq_gpr_rd_addr),

.sq_sp_mem_rd_ena(q_sq_gpr_rd_en),.sq_sp_mem_wr_ena(q_sq_gpr_wr_en),.sq_sp_wr_ena(
q_sq_gpr_wr_en),
              .sq_sp_gpr_phase_mux(q_sq_gpr_phase_mux),
.sq_sp_channel_mask(q_sq_channel_mask),
              .sq_sp_pixel_mask(q_sq_pix_mask),
.sq_sp_gpr_input_mux(q_sq_gpr_input_mux),
                .iInterpolated(Interpolated1),// iAutoCount,
                .iVertexIndices(VertexIndex1),
                .sq_sp_constant(q_sq_const),
                .tp_sp_data(q_tp_data1),.tp_sp_gpr_dst(q_tp_gpr_dst),
                .tp_sp_gpr_cmask(q_tp_gpr_cmask),.tp_sp_data_valid(q_tp_data_valid));

  vector uvector2(.sp_sx_data(osp_sx_data2),
                .sp_tp_data(sp_fetch_addr2),
                .sq_sp_instruct_start(q_sq_instruct_start),
                .sq_sp_instruct(q_sq_instruct),.sq_sp_stall(q_sq_stall),
                .sclk(sclk), .srst(srst),
                .sq_sp_wr_addr(q_sq_gpr_wr_addr), .sq_sp_gpr_rd_addr(q_sq_gpr_rd_addr),

.sq_sp_mem_rd_ena(q_sq_gpr_rd_en),.sq_sp_mem_wr_ena(q_sq_gpr_wr_en),.sq_sp_wr_ena(
q_sq_gpr_wr_en),
                .sq_sp_gpr_phase_mux(q_sq_gpr_phase_mux),
.sq_sp_channel_mask(q_sq_channel_mask),
                .sq_sp_pixel_mask(q_sq_pix_mask),
.sq_sp_gpr_input_mux(q_sq_gpr_input_mux),
                .iInterpolated(Interpolated2),// iAutoCount,
                .iVertexIndices(VertexIndex2),
                .sq_sp_constant(q_sq_const),
                .tp_sp_data(q_tp_data2),.tp_sp_gpr_dst(q_tp_gpr_dst),
                .tp_sp_gpr_cmask(q_tp_gpr_cmask),.tp_sp_data_valid(q_tp_data_valid));

  vector uvector3(.sp_sx_data(osp_sx_data3),
                .sp_tp_data(sp_fetch_addr3),
```

```
                .sq_sp_instruct_start(q_sq_instruct_start),
                .sq_sp_instruct(q_sq_instruct),.sq_sp_stall(q_sq_stall),
                .sclk(sclk), .srst(srst),
                .sq_sp_wr_addr(q_sq_gpr_wr_addr), .sq_sp_gpr_rd_addr(q_sq_gpr_rd_addr),

.sq_sp_mem_rd_ena(q_sq_gpr_rd_en),.sq_sp_mem_wr_ena(q_sq_gpr_wr_en),.sq_sp_wr_ena(
q_sq_gpr_wr_en),
                .sq_sp_gpr_phase_mux(q_sq_gpr_phase_mux),
.sq_sp_channel_mask(q_sq_channel_mask),
                .sq_sp_pixel_mask(q_sq_pix_mask),
.sq_sp_gpr_input_mux(q_sq_gpr_input_mux),
                .iInterpolated(Interpolated3),// iAutoCount,
                .iVertexIndices(VertexIndex3),
                .sq_sp_constant(q_sq_const),
                .tp_sp_data(q_tp_data3),.tp_sp_gpr_dst(q_tp_gpr_dst),
                .tp_sp_gpr_cmask(q_tp_gpr_cmask),.tp_sp_data_valid(q_tp_data_valid));

endmodule // sp
```

```
//                              -*- Mode: Verilog -*-
// Filename        : export_buffers.v
// Description     : This file represent the two banks of the export buffers (8 all together)
//                 : and the data muxing at the output of the buffers into four distinct buses
//                 : one for each RB block
// Author          : Andi Skende
// Created On      : Tue Apr 16 17:13:53 2002
// Last Modified By: .
// Last Modified On: .
// Update Count    : 0
// Status          : Open issues: generate the appropriate write and read enable controls for the
export buffers.
`timescale 1ns / 1ps
module  export_buffers(/*AUTOARG*/
   // Outputs
   orb0_data, orb1_data, orb2_data, orb3_data, oclipp_data,
   orb0_data_valid, orb1_data_valid, orb2_data_valid,
   orb3_data_valid, oclipp_data_valid,
   // Inputs
   iread_addr, iwrite_addr, ipixel_data0, ipixel_data1, ipixel_data2,
   ipixel_data3, ipixel_data4, ipixel_data5, ipixel_data6,
   ipixel_data7, sclk, srst, imem_wen, imem_wew, imem_re, iphase_rb0,
   iphase_rb1, iphase_rb2, iphase_rb3, iphase_clipp, iread_valid_rb0,
   iread_valid_rb1, iread_valid_rb2, iread_valid_rb3,
   iread_valid_clipp
   );
```

```


input [7:0]  iread_addr;
input [6:0]    iwrite_addr;
input [127:0] ipixel_data0,ipixel_data1,ipixel_data2,ipixel_data3;
input [127:0] ipixel_data4,ipixel_data5,ipixel_data6,ipixel_data7;
input             sclk,srst;
input [0:0]     imem_wen, imem_wew,imem_re;

//we need four phase counters for each rb request so we can serialize the data back to RBs
//into four consecutive cycles.
input [1:0] iphase_rb0;
input [1:0] iphase_rb1;
input [1:0] iphase_rb2;
input [1:0] iphase_rb3;
input [1:0] iphase_clipp;

//valid read request from RBs ...stays high for four cycles.
//this signal is eventually pipelined out into orb#_data_valid
input [0:0]     iread_valid_rb0, iread_valid_rb1,iread_valid_rb2,iread_valid_rb3;
input [0:0]     iread_valid_clipp;


output [127:0] orb0_data, orb1_data, orb2_data,orb3_data;
output [127:0] oclipp_data;
```

```
   output [0:0]   orb0_data_valid,orb1_data_valid,orb2_data_valid,orb3_data_valid;
   output [0:0]   oclipp_data_valid;

   reg [0:0]   rb0_data_valid,rb1_data_valid,rb2_data_valid,rb3_data_valid;
   reg [0:0]   clipp_data_valid;


   //export buffers bank select
   wire    sp_bank_sel;
   reg            q_sp_bank_sel;

   wire           sp_bank_sel0;
   wire           sp_bank_sel1;
   wire           sp_bank_sel2;
   wire           sp_bank_sel3;
   reg            q_sp_bank_sel0;
   reg            q_sp_bank_sel1;
   reg            q_sp_bank_sel2;
   reg            q_sp_bank_sel3;

   assign             sp_bank_sel = iread_addr[7];

   assign         sp_bank_sel0 = (iphase_rb0==0) ? iread_addr[7] : q_sp_bank_sel0;
   assign         sp_bank_sel1 = (iphase_rb1==0) ? iread_addr[7] : q_sp_bank_sel1;
   assign         sp_bank_sel2 = (iphase_rb2==0) ? iread_addr[7] : q_sp_bank_sel2;
```

```
assign         sp_bank_sel3 = (iphase_rb3==0) ? iread_addr[7] : q_sp_bank_sel3;

//constants
wire    HIGH;
wire           LOW;

assign             HIGH = 1'b1;
assign         LOW = ~HIGH;

wire [127:0]   buff0_out;
wire [127:0]   buff1_out;
wire [127:0]   buff2_out;
wire [127:0]   buff3_out;

wire [127:0]   buff4_out;
wire [127:0]   buff5_out;
wire [127:0]   buff6_out;
wire [127:0]   buff7_out;

reg [127:0]   rb0_data, rb1_data, rb2_data,rb3_data;
reg [127:0]    q_rb0_data, q_rb1_data, q_rb2_data,q_rb3_data;

reg [127:0]     bank0_data0;
reg [127:0]     bank0_data1;
reg [127:0]     bank0_data2;
reg [127:0]     bank0_data3;
reg [127:0]     bank0_clipp_data;
reg [127:0]     bank1_data0;
reg [127:0]     bank1_data1;
reg [127:0]     bank1_data2;
reg [127:0]     bank1_data3;
reg [127:0]     bank1_clipp_data;


reg [1:0] q_phase_rb0 , q_phase_rb1 , q_phase_rb2 , q_phase_rb3;
reg [1:0] q_phase_clipp;

reg [0:0] q0_read_valid_rb0, q1_read_valid_rb0;
reg [0:0] q2_read_valid_rb0, q3_read_valid_rb0;
reg [0:0] q0_read_valid_rb1, q1_read_valid_rb1;
reg [0:0] q2_read_valid_rb1, q3_read_valid_rb1;
reg [0:0] q0_read_valid_rb2, q1_read_valid_rb2;
reg [0:0] q2_read_valid_rb2, q3_read_valid_rb2;
reg [0:0] q0_read_valid_rb3, q1_read_valid_rb3;
reg [0:0] q2_read_valid_rb3, q3_read_valid_rb3;
reg [0:0] q0_read_valid_clipp, q1_read_valid_clipp;
reg [0:0] q2_read_valid_clipp, q3_read_valid_clipp;

always @(posedge sclk)
  begin
```

```
        q_phase_rb0 <= iphase_rb0;
        q_phase_rb1 <= iphase_rb1;
        q_phase_rb2 <= iphase_rb2;
        q_phase_rb3 <= iphase_rb3;
        q_phase_clipp <= iphase_clipp;
        q0_read_valid_rb0 <= iread_valid_rb0;
        q1_read_valid_rb0 <= q0_read_valid_rb0;
        q2_read_valid_rb0 <= q1_read_valid_rb0;
        q3_read_valid_rb0 <= q2_read_valid_rb0;
        q0_read_valid_rb1 <= iread_valid_rb1;
        q1_read_valid_rb1 <= q0_read_valid_rb1;
        q2_read_valid_rb1 <= q1_read_valid_rb1;
        q3_read_valid_rb1 <= q2_read_valid_rb1;
        q0_read_valid_rb2 <= iread_valid_rb2;
        q1_read_valid_rb2 <= q0_read_valid_rb2;
        q2_read_valid_rb2 <= q1_read_valid_rb2;
        q3_read_valid_rb2 <= q2_read_valid_rb2;
        q0_read_valid_rb3 <= iread_valid_rb3;
        q1_read_valid_rb3 <= q0_read_valid_rb3;
        q2_read_valid_rb3 <= q1_read_valid_rb3;
        q3_read_valid_rb3 <= q2_read_valid_rb3;
        q0_read_valid_clipp <= iread_valid_clipp;
        q1_read_valid_clipp <= q0_read_valid_clipp;
        q2_read_valid_clipp <= q1_read_valid_clipp;
        q3_read_valid_clipp <= q2_read_valid_clipp;
```

```
  end


reg [7:0]       q0_read_addr, q1_read_addr,q2_read_addr;
reg [0:0]       q0_mem_re, q1_mem_re, q2_mem_re;


//skewing the read enable and read address buses
always @(posedge sclk)
  begin
      q0_read_addr <= iread_addr;
      q1_read_addr <= q0_read_addr;
      q2_read_addr <= q1_read_addr;
      q0_mem_re <= imem_re;
      q1_mem_re <= q0_mem_re;
      q2_mem_re <= q1_mem_re;
  end


//skewing the write enable and write address buses so they line up with
//the data bus

reg [6:0] q_write_addr;
reg [0:0] q_mem_wew, q_mem_wen;

```

```
  always @(posedge sclk)
    begin
        q_write_addr <= iwrite_addr;
        q_mem_wew <= imem_wew;
        q_mem_wen <= imem_wen;
    end


  wire [6:0] read_mem_addr ;
  assign    read_mem_addr = iread_addr[6:0];

`ifdef USE_BEHAVE_MEM
  dum_mem_p2 #(7,128) bank0_buff0(.iRCLK(sclk),
                                  .iWCLK(sclk),
                                  .iMER(imem_re),
                                  .iMEW(imem_wen),
                                  .iWEN(imem_wew),
                                  .iRADR(iread_addr[6:0] ),
                                  .iWADR(iwrite_addr),
                                  .iD(ipixel_data0),
                                  .oQ(buff0_out)
                                  );
`else // !`ifdef USE_BEHAVE_MEM

  rfsd2_80x128cm2sw0 ubank0_buff0
```

```
    (/*VRGIO rfsd2_80x128cm2sw0 ipixel_data0 buff0_out iwrite_addr iread_addr imem_wen imem_re null*/
    // READ INTERFACE
    .CLKB(iSCLK), // Read Clock
    .OEB(vdd), // Output enable
    .MEB(imem_re), // Read enable
    .ADRB0(iread_addr[0]), .ADRB1(iread_addr[1]), .ADRB2(iread_addr[2]), .ADRB3(iread_addr[3]), // Read Address
    .ADRB4(iread_addr[4]), .ADRB5(iread_addr[5]), .ADRB6(iread_addr[6]), // Read Address
    .QB0(buff0_out[0]), .QB1(buff0_out[1]), .QB2(buff0_out[2]), .QB3(buff0_out[3]), // Read Data
    .QB4(buff0_out[4]), .QB5(buff0_out[5]), .QB6(buff0_out[6]), .QB7(buff0_out[7]), // Read Data
    .QB8(buff0_out[8]), .QB9(buff0_out[9]), .QB10(buff0_out[10]), .QB11(buff0_out[11]), // Read Data
    .QB12(buff0_out[12]), .QB13(buff0_out[13]), .QB14(buff0_out[14]), .QB15(buff0_out[15]), // Read Data
    .QB16(buff0_out[16]), .QB17(buff0_out[17]), .QB18(buff0_out[18]), .QB19(buff0_out[19]), // Read Data
    .QB20(buff0_out[20]), .QB21(buff0_out[21]), .QB22(buff0_out[22]), .QB23(buff0_out[23]), // Read Data
    .QB24(buff0_out[24]), .QB25(buff0_out[25]), .QB26(buff0_out[26]), .QB27(buff0_out[27]), // Read Data
    .QB28(buff0_out[28]), .QB29(buff0_out[29]), .QB30(buff0_out[30]), .QB31(buff0_out[31]), // Read Data
    .QB32(buff0_out[32]), .QB33(buff0_out[33]), .QB34(buff0_out[34]), .QB35(buff0_out[35]), // Read Data
    .QB36(buff0_out[36]), .QB37(buff0_out[37]), .QB38(buff0_out[38]), .QB39(buff0_out[39]), // Read Data
    .QB40(buff0_out[40]), .QB41(buff0_out[41]), .QB42(buff0_out[42]), .QB43(buff0_out[43]), // Read Data
```

```
    .QB44(buff0_out[44]), .QB45(buff0_out[45]), .QB46(buff0_out[46]), .QB47(buff0_out[47]), // Read Data
    .QB48(buff0_out[48]), .QB49(buff0_out[49]), .QB50(buff0_out[50]), .QB51(buff0_out[51]), // Read Data
    .QB52(buff0_out[52]), .QB53(buff0_out[53]), .QB54(buff0_out[54]), .QB55(buff0_out[55]), // Read Data
    .QB56(buff0_out[56]), .QB57(buff0_out[57]), .QB58(buff0_out[58]), .QB59(buff0_out[59]), // Read Data
    .QB60(buff0_out[60]), .QB61(buff0_out[61]), .QB62(buff0_out[62]), .QB63(buff0_out[63]), // Read Data
    .QB64(buff0_out[64]), .QB65(buff0_out[65]), .QB66(buff0_out[66]), .QB67(buff0_out[67]), // Read Data
    .QB68(buff0_out[68]), .QB69(buff0_out[69]), .QB70(buff0_out[70]), .QB71(buff0_out[71]), // Read Data
    .QB72(buff0_out[72]), .QB73(buff0_out[73]), .QB74(buff0_out[74]), .QB75(buff0_out[75]), // Read Data
    .QB76(buff0_out[76]), .QB77(buff0_out[77]), .QB78(buff0_out[78]), .QB79(buff0_out[79]), // Read Data
    .QB80(buff0_out[80]), .QB81(buff0_out[81]), .QB82(buff0_out[82]), .QB83(buff0_out[83]), // Read Data
    .QB84(buff0_out[84]), .QB85(buff0_out[85]), .QB86(buff0_out[86]), .QB87(buff0_out[87]), // Read Data
    .QB88(buff0_out[88]), .QB89(buff0_out[89]), .QB90(buff0_out[90]), .QB91(buff0_out[91]), // Read Data
    .QB92(buff0_out[92]), .QB93(buff0_out[93]), .QB94(buff0_out[94]), .QB95(buff0_out[95]), // Read Data
    .QB96(buff0_out[96]), .QB97(buff0_out[97]), .QB98(buff0_out[98]), .QB99(buff0_out[99]), // Read Data
    .QB100(buff0_out[100]), .QB101(buff0_out[101]), .QB102(buff0_out[102]), .QB103(buff0_out[103]), // Read Data
    .QB104(buff0_out[104]), .QB105(buff0_out[105]), .QB106(buff0_out[106]), .QB107(buff0_out[107]), // Read Data
```

```
    .QB108(buff0_out[108]), .QB109(buff0_out[109]), .QB110(buff0_out[110]), .QB111(buff0_out[111]), // Read Data
    .QB112(buff0_out[112]), .QB113(buff0_out[113]), .QB114(buff0_out[114]), .QB115(buff0_out[115]), // Read Data
    .QB116(buff0_out[116]), .QB117(buff0_out[117]), .QB118(buff0_out[118]), .QB119(buff0_out[119]), // Read Data
    .QB120(buff0_out[120]), .QB121(buff0_out[121]), .QB122(buff0_out[122]), .QB123(buff0_out[123]), // Read Data
    .QB124(buff0_out[124]), .QB125(buff0_out[125]), .QB126(buff0_out[126]), .QB127(buff0_out[127]), // Read Data
    // WRITE INTERFACE
    .CLKA(iSCLK), // Write Clock
    .WEA(imem_wen), // Write enable
    .MEA(vdd), // Memory enable
    .ADRA0(iwrite_addr[0]), .ADRA1(iwrite_addr[1]), .ADRA2(iwrite_addr[2]), .ADRA3(iwrite_addr[3]), // Write Address
    .ADRA4(iwrite_addr[4]), .ADRA5(iwrite_addr[5]), .ADRA6(iwrite_addr[6]), // Write Address
    .DA0(ipixel_data0[0]), .DA1(ipixel_data0[1]), .DA2(ipixel_data0[2]), .DA3(ipixel_data0[3]), // Write Data
    .DA4(ipixel_data0[4]), .DA5(ipixel_data0[5]), .DA6(ipixel_data0[6]), .DA7(ipixel_data0[7]), // Write Data
    .DA8(ipixel_data0[8]), .DA9(ipixel_data0[9]), .DA10(ipixel_data0[10]), .DA11(ipixel_data0[11]), // Write Data
    .DA12(ipixel_data0[12]), .DA13(ipixel_data0[13]), .DA14(ipixel_data0[14]), .DA15(ipixel_data0[15]), // Write Data
    .DA16(ipixel_data0[16]), .DA17(ipixel_data0[17]), .DA18(ipixel_data0[18]), .DA19(ipixel_data0[19]), // Write Data
    .DA20(ipixel_data0[20]), .DA21(ipixel_data0[21]), .DA22(ipixel_data0[22]), .DA23(ipixel_data0[23]), // Write Data
    .DA24(ipixel_data0[24]), .DA25(ipixel_data0[25]), .DA26(ipixel_data0[26]), .DA27(ipixel_data0[27]), // Write Data
```

```
    .DA28(ipixel_data0[28]), .DA29(ipixel_data0[29]), .DA30(ipixel_data0[30]), .DA31(ipixel_data0[31]), // Write Data
    .DA32(ipixel_data0[32]), .DA33(ipixel_data0[33]), .DA34(ipixel_data0[34]), .DA35(ipixel_data0[35]), // Write Data
    .DA36(ipixel_data0[36]), .DA37(ipixel_data0[37]), .DA38(ipixel_data0[38]), .DA39(ipixel_data0[39]), // Write Data
    .DA40(ipixel_data0[40]), .DA41(ipixel_data0[41]), .DA42(ipixel_data0[42]), .DA43(ipixel_data0[43]), // Write Data
    .DA44(ipixel_data0[44]), .DA45(ipixel_data0[45]), .DA46(ipixel_data0[46]), .DA47(ipixel_data0[47]), // Write Data
    .DA48(ipixel_data0[48]), .DA49(ipixel_data0[49]), .DA50(ipixel_data0[50]), .DA51(ipixel_data0[51]), // Write Data
    .DA52(ipixel_data0[52]), .DA53(ipixel_data0[53]), .DA54(ipixel_data0[54]), .DA55(ipixel_data0[55]), // Write Data
    .DA56(ipixel_data0[56]), .DA57(ipixel_data0[57]), .DA58(ipixel_data0[58]), .DA59(ipixel_data0[59]), // Write Data
    .DA60(ipixel_data0[60]), .DA61(ipixel_data0[61]), .DA62(ipixel_data0[62]), .DA63(ipixel_data0[63]), // Write Data
    .DA64(ipixel_data0[64]), .DA65(ipixel_data0[65]), .DA66(ipixel_data0[66]), .DA67(ipixel_data0[67]), // Write Data
    .DA68(ipixel_data0[68]), .DA69(ipixel_data0[69]), .DA70(ipixel_data0[70]), .DA71(ipixel_data0[71]), // Write Data
    .DA72(ipixel_data0[72]), .DA73(ipixel_data0[73]), .DA74(ipixel_data0[74]), .DA75(ipixel_data0[75]), // Write Data
    .DA76(ipixel_data0[76]), .DA77(ipixel_data0[77]), .DA78(ipixel_data0[78]), .DA79(ipixel_data0[79]), // Write Data
    .DA80(ipixel_data0[80]), .DA81(ipixel_data0[81]), .DA82(ipixel_data0[82]), .DA83(ipixel_data0[83]), // Write Data
    .DA84(ipixel_data0[84]), .DA85(ipixel_data0[85]), .DA86(ipixel_data0[86]), .DA87(ipixel_data0[87]), // Write Data
    .DA88(ipixel_data0[88]), .DA89(ipixel_data0[89]), .DA90(ipixel_data0[90]), .DA91(ipixel_data0[91]), // Write Data
    .DA92(ipixel_data0[92]),        .DA93(ipixel_data0[93]),        .DA94(ipixel_data0[94]),
.DA95(ipixel_data0[95]),  // Write Data
    .DA96(ipixel_data0[96]),        .DA97(ipixel_data0[97]),        .DA98(ipixel_data0[98]),
.DA99(ipixel_data0[99]),  // Write Data
    .DA100(ipixel_data0[100]),      .DA101(ipixel_data0[101]),      .DA102(ipixel_data0[102]),
.DA103(ipixel_data0[103]),  // Write Data
    .DA104(ipixel_data0[104]),      .DA105(ipixel_data0[105]),      .DA106(ipixel_data0[106]),
.DA107(ipixel_data0[107]),  // Write Data
    .DA108(ipixel_data0[108]),      .DA109(ipixel_data0[109]),      .DA110(ipixel_data0[110]),
.DA111(ipixel_data0[111]),  // Write Data
    .DA112(ipixel_data0[112]),      .DA113(ipixel_data0[113]),      .DA114(ipixel_data0[114]),
.DA115(ipixel_data0[115]),  // Write Data
    .DA116(ipixel_data0[116]),      .DA117(ipixel_data0[117]),      .DA118(ipixel_data0[118]),
.DA119(ipixel_data0[119]),  // Write Data
    .DA120(ipixel_data0[120]),      .DA121(ipixel_data0[121]),      .DA122(ipixel_data0[122]),
.DA123(ipixel_data0[123]),  // Write Data
    .DA124(ipixel_data0[124]),      .DA125(ipixel_data0[125]),      .DA126(ipixel_data0[126]),
.DA127(ipixel_data0[127]),  // Write Data
    // WRITE TEST SIGNALS
    .BISTEA(vss),
    .TWEA(vss), // Test write enable
    .TMEA(vss), // Test memory enable
    .TADRA0(iwrite_addr[0]),      .TADRA1(iwrite_addr[1]),      .TADRA2(iwrite_addr[2]),
.TADRA3(iwrite_addr[3]),  // Write Test Address
    .TADRA4(iwrite_addr[4]), .TADRA5(iwrite_addr[5]), .TADRA6(iwrite_addr[6]),  // Write
Test Address
    .TDA0(ipixel_data0[0]),        .TDA1(ipixel_data0[1]),        .TDA2(ipixel_data0[2]),
.TDA3(ipixel_data0[3]),  // Write Test Data
    .TDA4(ipixel_data0[4]),        .TDA5(ipixel_data0[5]),        .TDA6(ipixel_data0[6]),
.TDA7(ipixel_data0[7]),  // Write Test Data
    .TDA8(ipixel_data0[8]),        .TDA9(ipixel_data0[9]),        .TDA10(ipixel_data0[10]),
.TDA11(ipixel_data0[11]),  // Write Test Data
```

```
    .TDA12(ipixel_data0[12]),      .TDA13(ipixel_data0[13]),      .TDA14(ipixel_data0[14]),
.TDA15(ipixel_data0[15]),  // Write Test Data
    .TDA16(ipixel_data0[16]),      .TDA17(ipixel_data0[17]),      .TDA18(ipixel_data0[18]),
.TDA19(ipixel_data0[19]),  // Write Test Data
    .TDA20(ipixel_data0[20]),      .TDA21(ipixel_data0[21]),      .TDA22(ipixel_data0[22]),
.TDA23(ipixel_data0[23]),  // Write Test Data
    .TDA24(ipixel_data0[24]),      .TDA25(ipixel_data0[25]),      .TDA26(ipixel_data0[26]),
.TDA27(ipixel_data0[27]),  // Write Test Data
    .TDA28(ipixel_data0[28]),      .TDA29(ipixel_data0[29]),      .TDA30(ipixel_data0[30]),
.TDA31(ipixel_data0[31]),  // Write Test Data
    .TDA32(ipixel_data0[32]),      .TDA33(ipixel_data0[33]),      .TDA34(ipixel_data0[34]),
.TDA35(ipixel_data0[35]),  // Write Test Data
    .TDA36(ipixel_data0[36]),      .TDA37(ipixel_data0[37]),      .TDA38(ipixel_data0[38]),
.TDA39(ipixel_data0[39]),  // Write Test Data
    .TDA40(ipixel_data0[40]),      .TDA41(ipixel_data0[41]),      .TDA42(ipixel_data0[42]),
.TDA43(ipixel_data0[43]),  // Write Test Data
    .TDA44(ipixel_data0[44]),      .TDA45(ipixel_data0[45]),      .TDA46(ipixel_data0[46]),
.TDA47(ipixel_data0[47]),  // Write Test Data
    .TDA48(ipixel_data0[48]),      .TDA49(ipixel_data0[49]),      .TDA50(ipixel_data0[50]),
.TDA51(ipixel_data0[51]),  // Write Test Data
    .TDA52(ipixel_data0[52]),      .TDA53(ipixel_data0[53]),      .TDA54(ipixel_data0[54]),
.TDA55(ipixel_data0[55]),  // Write Test Data
    .TDA56(ipixel_data0[56]),      .TDA57(ipixel_data0[57]),      .TDA58(ipixel_data0[58]),
.TDA59(ipixel_data0[59]),  // Write Test Data
    .TDA60(ipixel_data0[60]),      .TDA61(ipixel_data0[61]),      .TDA62(ipixel_data0[62]),
.TDA63(ipixel_data0[63]),  // Write Test Data
    .TDA64(ipixel_data0[64]),      .TDA65(ipixel_data0[65]),      .TDA66(ipixel_data0[66]),
.TDA67(ipixel_data0[67]),  // Write Test Data
    .TDA68(ipixel_data0[68]),      .TDA69(ipixel_data0[69]),      .TDA70(ipixel_data0[70]),
.TDA71(ipixel_data0[71]),  // Write Test Data
    .TDA72(ipixel_data0[72]),      .TDA73(ipixel_data0[73]),      .TDA74(ipixel_data0[74]),
.TDA75(ipixel_data0[75]),  // Write Test Data
```

```
    .TDA76(ipixel_data0[76]),      .TDA77(ipixel_data0[77]),      .TDA78(ipixel_data0[78]),
.TDA79(ipixel_data0[79]),  // Write Test Data
    .TDA80(ipixel_data0[80]),      .TDA81(ipixel_data0[81]),      .TDA82(ipixel_data0[82]),
.TDA83(ipixel_data0[83]),  // Write Test Data
    .TDA84(ipixel_data0[84]),      .TDA85(ipixel_data0[85]),      .TDA86(ipixel_data0[86]),
.TDA87(ipixel_data0[87]),  // Write Test Data
    .TDA88(ipixel_data0[88]),      .TDA89(ipixel_data0[89]),      .TDA90(ipixel_data0[90]),
.TDA91(ipixel_data0[91]),  // Write Test Data
    .TDA92(ipixel_data0[92]),      .TDA93(ipixel_data0[93]),      .TDA94(ipixel_data0[94]),
.TDA95(ipixel_data0[95]),  // Write Test Data
    .TDA96(ipixel_data0[96]),      .TDA97(ipixel_data0[97]),      .TDA98(ipixel_data0[98]),
.TDA99(ipixel_data0[99]),  // Write Test Data
    .TDA100(ipixel_data0[100]),  .TDA101(ipixel_data0[101]),  .TDA102(ipixel_data0[102]),
.TDA103(ipixel_data0[103]),  // Write Test Data
    .TDA104(ipixel_data0[104]),  .TDA105(ipixel_data0[105]),  .TDA106(ipixel_data0[106]),
.TDA107(ipixel_data0[107]),  // Write Test Data
    .TDA108(ipixel_data0[108]),  .TDA109(ipixel_data0[109]),  .TDA110(ipixel_data0[110]),
.TDA111(ipixel_data0[111]),  // Write Test Data
    .TDA112(ipixel_data0[112]),  .TDA113(ipixel_data0[113]),  .TDA114(ipixel_data0[114]),
.TDA115(ipixel_data0[115]),  // Write Test Data
    .TDA116(ipixel_data0[116]),  .TDA117(ipixel_data0[117]),  .TDA118(ipixel_data0[118]),
.TDA119(ipixel_data0[119]),  // Write Test Data
    .TDA120(ipixel_data0[120]),  .TDA121(ipixel_data0[121]),  .TDA122(ipixel_data0[122]),
.TDA123(ipixel_data0[123]),  // Write Test Data
    .TDA124(ipixel_data0[124]),  .TDA125(ipixel_data0[125]),  .TDA126(ipixel_data0[126]),
.TDA127(ipixel_data0[127]),  // Write Test Data
    //READ TEST SIGNALS
    .BISTEB(vss),
    .TOEB(vss),
    .TMEB(vss),
    .TADRB0(iread_addr[0]),       .TADRB1(iread_addr[1]),       .TADRB2(iread_addr[2]),
.TADRB3(iread_addr[3]),  // Read Test Address
```

```
    .TADRB4(iread_addr[4]),  .TADRB5(iread_addr[5]),  .TADRB6(iread_addr[6]),    // Read
Test Address
    .AWTB(vss)
    );

`endif // !`ifdef USE_BEHAVE_MEM


  wire [6:0] q0_read_mem_addr;
  assign    q0_read_mem_addr = q0_read_addr[6:0];

`ifdef USE_BEHAVE_MEM

  dum_mem_p2 #(7,128) bank0_buff1(.iRCLK(sclk),
                                  .iWCLK(sclk),
                                  .iMER(q0_mem_re),
                                  .iMEW(imem_wen),
                                  .iWEN(imem_wew),
                                  .iRADR(q0_read_addr[6:0]),
                                  .iWADR(iwrite_addr),
                                  .iD(ipixel_data1),
                                  .oQ(buff1_out)
                                  );
`else // !`ifdef USE_BEHAVE_MEM
  rfsd2_80x128cm2sw0 ubank0_buff1
```

```
    (/*VRGIO rfsd2_80x128cm2sw0 ipixel_data1 buff0_out iwrite_addr q0_read_addr imem_wen q0_mem_re null*/
    // READ INTERFACE
    .CLKB(iSCLK), // Read Clock
    .OEB(vdd), // Output enable
    .MEB(q0_mem_re), // Read enable
    .ADRB0(q0_read_addr[0]), .ADRB1(q0_read_addr[1]), .ADRB2(q0_read_addr[2]), .ADRB3(q0_read_addr[3]), // Read Address
    .ADRB4(q0_read_addr[4]), .ADRB5(q0_read_addr[5]), .ADRB6(q0_read_addr[6]), // Read Address
    .QB0(buff0_out[0]), .QB1(buff0_out[1]), .QB2(buff0_out[2]), .QB3(buff0_out[3]), // Read Data
    .QB4(buff0_out[4]), .QB5(buff0_out[5]), .QB6(buff0_out[6]), .QB7(buff0_out[7]), // Read Data
    .QB8(buff0_out[8]), .QB9(buff0_out[9]), .QB10(buff0_out[10]), .QB11(buff0_out[11]), // Read Data
    .QB12(buff0_out[12]), .QB13(buff0_out[13]), .QB14(buff0_out[14]), .QB15(buff0_out[15]), // Read Data
    .QB16(buff0_out[16]), .QB17(buff0_out[17]), .QB18(buff0_out[18]), .QB19(buff0_out[19]), // Read Data
    .QB20(buff0_out[20]), .QB21(buff0_out[21]), .QB22(buff0_out[22]), .QB23(buff0_out[23]), // Read Data
    .QB24(buff0_out[24]), .QB25(buff0_out[25]), .QB26(buff0_out[26]), .QB27(buff0_out[27]), // Read Data
    .QB28(buff0_out[28]), .QB29(buff0_out[29]), .QB30(buff0_out[30]), .QB31(buff0_out[31]), // Read Data
    .QB32(buff0_out[32]), .QB33(buff0_out[33]), .QB34(buff0_out[34]), .QB35(buff0_out[35]), // Read Data
    .QB36(buff0_out[36]), .QB37(buff0_out[37]), .QB38(buff0_out[38]), .QB39(buff0_out[39]), // Read Data
    .QB40(buff0_out[40]), .QB41(buff0_out[41]), .QB42(buff0_out[42]), .QB43(buff0_out[43]), // Read Data
```

```
    .QB44(buff0_out[44]), .QB45(buff0_out[45]), .QB46(buff0_out[46]), .QB47(buff0_out[47]), // Read Data
    .QB48(buff0_out[48]), .QB49(buff0_out[49]), .QB50(buff0_out[50]), .QB51(buff0_out[51]), // Read Data
    .QB52(buff0_out[52]), .QB53(buff0_out[53]), .QB54(buff0_out[54]), .QB55(buff0_out[55]), // Read Data
    .QB56(buff0_out[56]), .QB57(buff0_out[57]), .QB58(buff0_out[58]), .QB59(buff0_out[59]), // Read Data
    .QB60(buff0_out[60]), .QB61(buff0_out[61]), .QB62(buff0_out[62]), .QB63(buff0_out[63]), // Read Data
    .QB64(buff0_out[64]), .QB65(buff0_out[65]), .QB66(buff0_out[66]), .QB67(buff0_out[67]), // Read Data
    .QB68(buff0_out[68]), .QB69(buff0_out[69]), .QB70(buff0_out[70]), .QB71(buff0_out[71]), // Read Data
    .QB72(buff0_out[72]), .QB73(buff0_out[73]), .QB74(buff0_out[74]), .QB75(buff0_out[75]), // Read Data
    .QB76(buff0_out[76]), .QB77(buff0_out[77]), .QB78(buff0_out[78]), .QB79(buff0_out[79]), // Read Data
    .QB80(buff0_out[80]), .QB81(buff0_out[81]), .QB82(buff0_out[82]), .QB83(buff0_out[83]), // Read Data
    .QB84(buff0_out[84]), .QB85(buff0_out[85]), .QB86(buff0_out[86]), .QB87(buff0_out[87]), // Read Data
    .QB88(buff0_out[88]), .QB89(buff0_out[89]), .QB90(buff0_out[90]), .QB91(buff0_out[91]), // Read Data
    .QB92(buff0_out[92]), .QB93(buff0_out[93]), .QB94(buff0_out[94]), .QB95(buff0_out[95]), // Read Data
    .QB96(buff0_out[96]), .QB97(buff0_out[97]), .QB98(buff0_out[98]), .QB99(buff0_out[99]), // Read Data
    .QB100(buff0_out[100]), .QB101(buff0_out[101]), .QB102(buff0_out[102]), .QB103(buff0_out[103]), // Read Data
    .QB104(buff0_out[104]), .QB105(buff0_out[105]), .QB106(buff0_out[106]), .QB107(buff0_out[107]), // Read Data
```

```
    .QB108(buff0_out[108]), .QB109(buff0_out[109]), .QB110(buff0_out[110]), .QB111(buff0_out[111]), // Read Data
    .QB112(buff0_out[112]), .QB113(buff0_out[113]), .QB114(buff0_out[114]), .QB115(buff0_out[115]), // Read Data
    .QB116(buff0_out[116]), .QB117(buff0_out[117]), .QB118(buff0_out[118]), .QB119(buff0_out[119]), // Read Data
    .QB120(buff0_out[120]), .QB121(buff0_out[121]), .QB122(buff0_out[122]), .QB123(buff0_out[123]), // Read Data
    .QB124(buff0_out[124]), .QB125(buff0_out[125]), .QB126(buff0_out[126]), .QB127(buff0_out[127]), // Read Data
    // WRITE INTERFACE
    .CLKA(iSCLK), // Write Clock
    .WEA(imem_wen), // Write enable
    .MEA(vdd), // Memory enable
    .ADRA0(iwrite_addr[0]), .ADRA1(iwrite_addr[1]), .ADRA2(iwrite_addr[2]), .ADRA3(iwrite_addr[3]), // Write Address
    .ADRA4(iwrite_addr[4]), .ADRA5(iwrite_addr[5]), .ADRA6(iwrite_addr[6]), // Write Address
    .DA0(ipixel_data1[0]), .DA1(ipixel_data1[1]), .DA2(ipixel_data1[2]), .DA3(ipixel_data1[3]), // Write Data
    .DA4(ipixel_data1[4]), .DA5(ipixel_data1[5]), .DA6(ipixel_data1[6]), .DA7(ipixel_data1[7]), // Write Data
    .DA8(ipixel_data1[8]), .DA9(ipixel_data1[9]), .DA10(ipixel_data1[10]), .DA11(ipixel_data1[11]), // Write Data
    .DA12(ipixel_data1[12]), .DA13(ipixel_data1[13]), .DA14(ipixel_data1[14]), .DA15(ipixel_data1[15]), // Write Data
    .DA16(ipixel_data1[16]), .DA17(ipixel_data1[17]), .DA18(ipixel_data1[18]), .DA19(ipixel_data1[19]), // Write Data
    .DA20(ipixel_data1[20]), .DA21(ipixel_data1[21]), .DA22(ipixel_data1[22]), .DA23(ipixel_data1[23]), // Write Data
    .DA24(ipixel_data1[24]), .DA25(ipixel_data1[25]), .DA26(ipixel_data1[26]), .DA27(ipixel_data1[27]), // Write Data
```

```
    .DA28(ipixel_data1[28]), .DA29(ipixel_data1[29]), .DA30(ipixel_data1[30]), .DA31(ipixel_data1[31]), // Write Data
    .DA32(ipixel_data1[32]), .DA33(ipixel_data1[33]), .DA34(ipixel_data1[34]), .DA35(ipixel_data1[35]), // Write Data
    .DA36(ipixel_data1[36]), .DA37(ipixel_data1[37]), .DA38(ipixel_data1[38]), .DA39(ipixel_data1[39]), // Write Data
    .DA40(ipixel_data1[40]), .DA41(ipixel_data1[41]), .DA42(ipixel_data1[42]), .DA43(ipixel_data1[43]), // Write Data
    .DA44(ipixel_data1[44]), .DA45(ipixel_data1[45]), .DA46(ipixel_data1[46]), .DA47(ipixel_data1[47]), // Write Data
    .DA48(ipixel_data1[48]), .DA49(ipixel_data1[49]), .DA50(ipixel_data1[50]), .DA51(ipixel_data1[51]), // Write Data
    .DA52(ipixel_data1[52]), .DA53(ipixel_data1[53]), .DA54(ipixel_data1[54]), .DA55(ipixel_data1[55]), // Write Data
    .DA56(ipixel_data1[56]), .DA57(ipixel_data1[57]), .DA58(ipixel_data1[58]), .DA59(ipixel_data1[59]), // Write Data
    .DA60(ipixel_data1[60]), .DA61(ipixel_data1[61]), .DA62(ipixel_data1[62]), .DA63(ipixel_data1[63]), // Write Data
    .DA64(ipixel_data1[64]), .DA65(ipixel_data1[65]), .DA66(ipixel_data1[66]), .DA67(ipixel_data1[67]), // Write Data
    .DA68(ipixel_data1[68]), .DA69(ipixel_data1[69]), .DA70(ipixel_data1[70]), .DA71(ipixel_data1[71]), // Write Data
    .DA72(ipixel_data1[72]), .DA73(ipixel_data1[73]), .DA74(ipixel_data1[74]), .DA75(ipixel_data1[75]), // Write Data
    .DA76(ipixel_data1[76]), .DA77(ipixel_data1[77]), .DA78(ipixel_data1[78]), .DA79(ipixel_data1[79]), // Write Data
    .DA80(ipixel_data1[80]), .DA81(ipixel_data1[81]), .DA82(ipixel_data1[82]), .DA83(ipixel_data1[83]), // Write Data
    .DA84(ipixel_data1[84]), .DA85(ipixel_data1[85]), .DA86(ipixel_data1[86]), .DA87(ipixel_data1[87]), // Write Data
    .DA88(ipixel_data1[88]), .DA89(ipixel_data1[89]), .DA90(ipixel_data1[90]), .DA91(ipixel_data1[91]), // Write Data
    .DA92(ipixel_data1[92]),    .DA93(ipixel_data1[93]),    .DA94(ipixel_data1[94]),    .DA95(ipixel_data1[95]),  // Write Data
    .DA96(ipixel_data1[96]),    .DA97(ipixel_data1[97]),    .DA98(ipixel_data1[98]),    .DA99(ipixel_data1[99]),  // Write Data
    .DA100(ipixel_data1[100]),  .DA101(ipixel_data1[101]),  .DA102(ipixel_data1[102]),  .DA103(ipixel_data1[103]),  // Write Data
    .DA104(ipixel_data1[104]),  .DA105(ipixel_data1[105]),  .DA106(ipixel_data1[106]),  .DA107(ipixel_data1[107]),  // Write Data
    .DA108(ipixel_data1[108]),  .DA109(ipixel_data1[109]),  .DA110(ipixel_data1[110]),  .DA111(ipixel_data1[111]),  // Write Data
    .DA112(ipixel_data1[112]),  .DA113(ipixel_data1[113]),  .DA114(ipixel_data1[114]),  .DA115(ipixel_data1[115]),  // Write Data
    .DA116(ipixel_data1[116]),  .DA117(ipixel_data1[117]),  .DA118(ipixel_data1[118]),  .DA119(ipixel_data1[119]),  // Write Data
    .DA120(ipixel_data1[120]),  .DA121(ipixel_data1[121]),  .DA122(ipixel_data1[122]),  .DA123(ipixel_data1[123]),  // Write Data
    .DA124(ipixel_data1[124]),  .DA125(ipixel_data1[125]),  .DA126(ipixel_data1[126]),  .DA127(ipixel_data1[127]),  // Write Data
    // WRITE TEST SIGNALS
    .BISTEA(vss),
    .TWEA(vss), // Test write enable
    .TMEA(vss), // Test memory enable
    .TADRA0(iwrite_addr[0]),    .TADRA1(iwrite_addr[1]),    .TADRA2(iwrite_addr[2]),    .TADRA3(iwrite_addr[3]),  // Write Test Address
    .TADRA4(iwrite_addr[4]), .TADRA5(iwrite_addr[5]), .TADRA6(iwrite_addr[6]),  // Write Test Address
    .TDA0(ipixel_data1[0]),     .TDA1(ipixel_data1[1]),     .TDA2(ipixel_data1[2]),     .TDA3(ipixel_data1[3]),  // Write Test Data
    .TDA4(ipixel_data1[4]),     .TDA5(ipixel_data1[5]),     .TDA6(ipixel_data1[6]),     .TDA7(ipixel_data1[7]),  // Write Test Data
    .TDA8(ipixel_data1[8]),     .TDA9(ipixel_data1[9]),     .TDA10(ipixel_data1[10]),   .TDA11(ipixel_data1[11]),  // Write Test Data
```

```
    .TDA12(ipixel_data1[12]),   .TDA13(ipixel_data1[13]),   .TDA14(ipixel_data1[14]),   .TDA15(ipixel_data1[15]),  // Write Test Data
    .TDA16(ipixel_data1[16]),   .TDA17(ipixel_data1[17]),   .TDA18(ipixel_data1[18]),   .TDA19(ipixel_data1[19]),  // Write Test Data
    .TDA20(ipixel_data1[20]),   .TDA21(ipixel_data1[21]),   .TDA22(ipixel_data1[22]),   .TDA23(ipixel_data1[23]),  // Write Test Data
    .TDA24(ipixel_data1[24]),   .TDA25(ipixel_data1[25]),   .TDA26(ipixel_data1[26]),   .TDA27(ipixel_data1[27]),  // Write Test Data
    .TDA28(ipixel_data1[28]),   .TDA29(ipixel_data1[29]),   .TDA30(ipixel_data1[30]),   .TDA31(ipixel_data1[31]),  // Write Test Data
    .TDA32(ipixel_data1[32]),   .TDA33(ipixel_data1[33]),   .TDA34(ipixel_data1[34]),   .TDA35(ipixel_data1[35]),  // Write Test Data
    .TDA36(ipixel_data1[36]),   .TDA37(ipixel_data1[37]),   .TDA38(ipixel_data1[38]),   .TDA39(ipixel_data1[39]),  // Write Test Data
    .TDA40(ipixel_data1[40]),   .TDA41(ipixel_data1[41]),   .TDA42(ipixel_data1[42]),   .TDA43(ipixel_data1[43]),  // Write Test Data
    .TDA44(ipixel_data1[44]),   .TDA45(ipixel_data1[45]),   .TDA46(ipixel_data1[46]),   .TDA47(ipixel_data1[47]),  // Write Test Data
    .TDA48(ipixel_data1[48]),   .TDA49(ipixel_data1[49]),   .TDA50(ipixel_data1[50]),   .TDA51(ipixel_data1[51]),  // Write Test Data
    .TDA52(ipixel_data1[52]),   .TDA53(ipixel_data1[53]),   .TDA54(ipixel_data1[54]),   .TDA55(ipixel_data1[55]),  // Write Test Data
    .TDA56(ipixel_data1[56]),   .TDA57(ipixel_data1[57]),   .TDA58(ipixel_data1[58]),   .TDA59(ipixel_data1[59]),  // Write Test Data
    .TDA60(ipixel_data1[60]),   .TDA61(ipixel_data1[61]),   .TDA62(ipixel_data1[62]),   .TDA63(ipixel_data1[63]),  // Write Test Data
    .TDA64(ipixel_data1[64]),   .TDA65(ipixel_data1[65]),   .TDA66(ipixel_data1[66]),   .TDA67(ipixel_data1[67]),  // Write Test Data
    .TDA68(ipixel_data1[68]),   .TDA69(ipixel_data1[69]),   .TDA70(ipixel_data1[70]),   .TDA71(ipixel_data1[71]),  // Write Test Data
    .TDA72(ipixel_data1[72]),   .TDA73(ipixel_data1[73]),   .TDA74(ipixel_data1[74]),   .TDA75(ipixel_data1[75]),  // Write Test Data
```

```
    .TDA76(ipixel_data1[76]),   .TDA77(ipixel_data1[77]),   .TDA78(ipixel_data1[78]),   .TDA79(ipixel_data1[79]),  // Write Test Data
    .TDA80(ipixel_data1[80]),   .TDA81(ipixel_data1[81]),   .TDA82(ipixel_data1[82]),   .TDA83(ipixel_data1[83]),  // Write Test Data
    .TDA84(ipixel_data1[84]),   .TDA85(ipixel_data1[85]),   .TDA86(ipixel_data1[86]),   .TDA87(ipixel_data1[87]),  // Write Test Data
    .TDA88(ipixel_data1[88]),   .TDA89(ipixel_data1[89]),   .TDA90(ipixel_data1[90]),   .TDA91(ipixel_data1[91]),  // Write Test Data
    .TDA92(ipixel_data1[92]),   .TDA93(ipixel_data1[93]),   .TDA94(ipixel_data1[94]),   .TDA95(ipixel_data1[95]),  // Write Test Data
    .TDA96(ipixel_data1[96]),   .TDA97(ipixel_data1[97]),   .TDA98(ipixel_data1[98]),   .TDA99(ipixel_data1[99]),  // Write Test Data
    .TDA100(ipixel_data1[100]),   .TDA101(ipixel_data1[101]),   .TDA102(ipixel_data1[102]),   .TDA103(ipixel_data1[103]),  // Write Test Data
    .TDA104(ipixel_data1[104]),   .TDA105(ipixel_data1[105]),   .TDA106(ipixel_data1[106]),   .TDA107(ipixel_data1[107]),  // Write Test Data
    .TDA108(ipixel_data1[108]),   .TDA109(ipixel_data1[109]),   .TDA110(ipixel_data1[110]),   .TDA111(ipixel_data1[111]),  // Write Test Data
    .TDA112(ipixel_data1[112]),   .TDA113(ipixel_data1[113]),   .TDA114(ipixel_data1[114]),   .TDA115(ipixel_data1[115]),  // Write Test Data
    .TDA116(ipixel_data1[116]),   .TDA117(ipixel_data1[117]),   .TDA118(ipixel_data1[118]),   .TDA119(ipixel_data1[119]),  // Write Test Data
    .TDA120(ipixel_data1[120]),   .TDA121(ipixel_data1[121]),   .TDA122(ipixel_data1[122]),   .TDA123(ipixel_data1[123]),  // Write Test Data
    .TDA124(ipixel_data1[124]),   .TDA125(ipixel_data1[125]),   .TDA126(ipixel_data1[126]),   .TDA127(ipixel_data1[127]),  // Write Test Data
    //READ TEST SIGNALS
    .BISTEB(vss),
    .TOEB(vss),
    .TMEB(vss),
    .TADRB0(q0_read_addr[0]),   .TADRB1(q0_read_addr[1]),   .TADRB2(q0_read_addr[2]),   .TADRB3(q0_read_addr[3]),  // Read Test Address
```

```
    .TADRB4(q0_read_addr[4]), .TADRB5(q0_read_addr[5]), .TADRB6(q0_read_addr[6]),  // Read Test Address
    .AWTB(vss)
    );
`endif // !`ifdef USE_BEHAVE_MEM



  wire [6:0] q1_read_mem_addr;
  assign   q1_read_mem_addr = q1_read_addr[6:0];

`ifdef USE_BEHAVE_MEM
  dum_mem_p2 #(7,128) bank0_buff2(.iRCLK(sclk),
                                  .iWCLK(sclk),
                                  .iMER(q1_mem_re),
                                  .iMEW(imem_wen),
                                  .iWEN(imem_wew),
                                  .iRADR(q1_read_addr[6:0]),
                                  .iWADR(iwrite_addr),
                                  .iD(ipixel_data2),
                                  .oQ(buff2_out)
                                  );
`else // !`ifdef USE_BEHAVE_MEM
  rfsd2_80x128cm2sw0 ubank0_buff2
   (/*VRGIO   rfsd2_80x128cm2sw0   ipixel_data2   buff2_out   iwrite_addr   q1_read_addr   imem_wen q1_mem_re null*/
// READ INTERFACE
.CLKB(iSCLK), // Read Clock
.OEB(vdd), // Output enable
.MEB(q1_mem_re), // Read enable
.ADRB0(q1_read_addr[0]), .ADRB1(q1_read_addr[1]), .ADRB2(q1_read_addr[2]), .ADRB3(q1_read_addr[3]), // Read Address
.ADRB4(q1_read_addr[4]), .ADRB5(q1_read_addr[5]), .ADRB6(q1_read_addr[6]), // Read Address
.QB0(buff2_out[0]), .QB1(buff2_out[1]), .QB2(buff2_out[2]), .QB3(buff2_out[3]), // Read Data
.QB4(buff2_out[4]), .QB5(buff2_out[5]), .QB6(buff2_out[6]), .QB7(buff2_out[7]), // Read Data
.QB8(buff2_out[8]), .QB9(buff2_out[9]), .QB10(buff2_out[10]), .QB11(buff2_out[11]), // Read Data
.QB12(buff2_out[12]), .QB13(buff2_out[13]), .QB14(buff2_out[14]), .QB15(buff2_out[15]), // Read Data
.QB16(buff2_out[16]), .QB17(buff2_out[17]), .QB18(buff2_out[18]), .QB19(buff2_out[19]), // Read Data
.QB20(buff2_out[20]), .QB21(buff2_out[21]), .QB22(buff2_out[22]), .QB23(buff2_out[23]), // Read Data
.QB24(buff2_out[24]), .QB25(buff2_out[25]), .QB26(buff2_out[26]), .QB27(buff2_out[27]), // Read Data
.QB28(buff2_out[28]), .QB29(buff2_out[29]), .QB30(buff2_out[30]), .QB31(buff2_out[31]), // Read Data
.QB32(buff2_out[32]), .QB33(buff2_out[33]), .QB34(buff2_out[34]), .QB35(buff2_out[35]), // Read Data
.QB36(buff2_out[36]), .QB37(buff2_out[37]), .QB38(buff2_out[38]), .QB39(buff2_out[39]), // Read Data
.QB40(buff2_out[40]), .QB41(buff2_out[41]), .QB42(buff2_out[42]), .QB43(buff2_out[43]), // Read Data
.QB44(buff2_out[44]), .QB45(buff2_out[45]), .QB46(buff2_out[46]), .QB47(buff2_out[47]), // Read Data
```

```
.QB48(buff2_out[48]), .QB49(buff2_out[49]), .QB50(buff2_out[50]), .QB51(buff2_out[51]), // Read Data
.QB52(buff2_out[52]), .QB53(buff2_out[53]), .QB54(buff2_out[54]), .QB55(buff2_out[55]), // Read Data
.QB56(buff2_out[56]), .QB57(buff2_out[57]), .QB58(buff2_out[58]), .QB59(buff2_out[59]), // Read Data
.QB60(buff2_out[60]), .QB61(buff2_out[61]), .QB62(buff2_out[62]), .QB63(buff2_out[63]), // Read Data
.QB64(buff2_out[64]), .QB65(buff2_out[65]), .QB66(buff2_out[66]), .QB67(buff2_out[67]), // Read Data
.QB68(buff2_out[68]), .QB69(buff2_out[69]), .QB70(buff2_out[70]), .QB71(buff2_out[71]), // Read Data
.QB72(buff2_out[72]), .QB73(buff2_out[73]), .QB74(buff2_out[74]), .QB75(buff2_out[75]), // Read Data
.QB76(buff2_out[76]), .QB77(buff2_out[77]), .QB78(buff2_out[78]), .QB79(buff2_out[79]), // Read Data
.QB80(buff2_out[80]), .QB81(buff2_out[81]), .QB82(buff2_out[82]), .QB83(buff2_out[83]), // Read Data
.QB84(buff2_out[84]), .QB85(buff2_out[85]), .QB86(buff2_out[86]), .QB87(buff2_out[87]), // Read Data
.QB88(buff2_out[88]), .QB89(buff2_out[89]), .QB90(buff2_out[90]), .QB91(buff2_out[91]), // Read Data
.QB92(buff2_out[92]), .QB93(buff2_out[93]), .QB94(buff2_out[94]), .QB95(buff2_out[95]), // Read Data
.QB96(buff2_out[96]), .QB97(buff2_out[97]), .QB98(buff2_out[98]), .QB99(buff2_out[99]), // Read Data
.QB100(buff2_out[100]), .QB101(buff2_out[101]), .QB102(buff2_out[102]), .QB103(buff2_out[103]), // Read Data
.QB104(buff2_out[104]), .QB105(buff2_out[105]), .QB106(buff2_out[106]), .QB107(buff2_out[107]), // Read Data
.QB108(buff2_out[108]), .QB109(buff2_out[109]), .QB110(buff2_out[110]), .QB111(buff2_out[111]), // Read Data
```

```
.QB112(buff2_out[112]), .QB113(buff2_out[113]), .QB114(buff2_out[114]), .QB115(buff2_out[115]), // Read Data
.QB116(buff2_out[116]), .QB117(buff2_out[117]), .QB118(buff2_out[118]), .QB119(buff2_out[119]), // Read Data
.QB120(buff2_out[120]), .QB121(buff2_out[121]), .QB122(buff2_out[122]), .QB123(buff2_out[123]), // Read Data
.QB124(buff2_out[124]), .QB125(buff2_out[125]), .QB126(buff2_out[126]), .QB127(buff2_out[127]), // Read Data
// WRITE INTERFACE
.CLKA(iSCLK), // Write Clock
.WEA(imem_wen), // Write enable
.MEA(vdd), // Memory enable
.ADRA0(iwrite_addr[0]), .ADRA1(iwrite_addr[1]), .ADRA2(iwrite_addr[2]), .ADRA3(iwrite_addr[3]), // Write Address
.ADRA4(iwrite_addr[4]), .ADRA5(iwrite_addr[5]), .ADRA6(iwrite_addr[6]), // Write Address
.DA0(ipixel_data2[0]), .DA1(ipixel_data2[1]), .DA2(ipixel_data2[2]), .DA3(ipixel_data2[3]), // Write Data
.DA4(ipixel_data2[4]), .DA5(ipixel_data2[5]), .DA6(ipixel_data2[6]), .DA7(ipixel_data2[7]), // Write Data
.DA8(ipixel_data2[8]), .DA9(ipixel_data2[9]), .DA10(ipixel_data2[10]), .DA11(ipixel_data2[11]), // Write Data
.DA12(ipixel_data2[12]), .DA13(ipixel_data2[13]), .DA14(ipixel_data2[14]), .DA15(ipixel_data2[15]), // Write Data
.DA16(ipixel_data2[16]), .DA17(ipixel_data2[17]), .DA18(ipixel_data2[18]), .DA19(ipixel_data2[19]), // Write Data
.DA20(ipixel_data2[20]), .DA21(ipixel_data2[21]), .DA22(ipixel_data2[22]), .DA23(ipixel_data2[23]), // Write Data
.DA24(ipixel_data2[24]), .DA25(ipixel_data2[25]), .DA26(ipixel_data2[26]), .DA27(ipixel_data2[27]), // Write Data
.DA28(ipixel_data2[28]), .DA29(ipixel_data2[29]), .DA30(ipixel_data2[30]), .DA31(ipixel_data2[31]), // Write Data
```

```
.DA32(ipixel_data2[32]), .DA33(ipixel_data2[33]), .DA34(ipixel_data2[34]), .DA35(ipixel_data2[35]), // Write Data
.DA36(ipixel_data2[36]), .DA37(ipixel_data2[37]), .DA38(ipixel_data2[38]), .DA39(ipixel_data2[39]), // Write Data
.DA40(ipixel_data2[40]), .DA41(ipixel_data2[41]), .DA42(ipixel_data2[42]), .DA43(ipixel_data2[43]), // Write Data
.DA44(ipixel_data2[44]), .DA45(ipixel_data2[45]), .DA46(ipixel_data2[46]), .DA47(ipixel_data2[47]), // Write Data
.DA48(ipixel_data2[48]), .DA49(ipixel_data2[49]), .DA50(ipixel_data2[50]), .DA51(ipixel_data2[51]), // Write Data
.DA52(ipixel_data2[52]), .DA53(ipixel_data2[53]), .DA54(ipixel_data2[54]), .DA55(ipixel_data2[55]), // Write Data
.DA56(ipixel_data2[56]), .DA57(ipixel_data2[57]), .DA58(ipixel_data2[58]), .DA59(ipixel_data2[59]), // Write Data
.DA60(ipixel_data2[60]), .DA61(ipixel_data2[61]), .DA62(ipixel_data2[62]), .DA63(ipixel_data2[63]), // Write Data
.DA64(ipixel_data2[64]), .DA65(ipixel_data2[65]), .DA66(ipixel_data2[66]), .DA67(ipixel_data2[67]), // Write Data
.DA68(ipixel_data2[68]), .DA69(ipixel_data2[69]), .DA70(ipixel_data2[70]), .DA71(ipixel_data2[71]), // Write Data
.DA72(ipixel_data2[72]), .DA73(ipixel_data2[73]), .DA74(ipixel_data2[74]), .DA75(ipixel_data2[75]), // Write Data
.DA76(ipixel_data2[76]), .DA77(ipixel_data2[77]), .DA78(ipixel_data2[78]), .DA79(ipixel_data2[79]), // Write Data
.DA80(ipixel_data2[80]), .DA81(ipixel_data2[81]), .DA82(ipixel_data2[82]), .DA83(ipixel_data2[83]), // Write Data
.DA84(ipixel_data2[84]), .DA85(ipixel_data2[85]), .DA86(ipixel_data2[86]), .DA87(ipixel_data2[87]), // Write Data
.DA88(ipixel_data2[88]), .DA89(ipixel_data2[89]), .DA90(ipixel_data2[90]), .DA91(ipixel_data2[91]), // Write Data
.DA92(ipixel_data2[92]), .DA93(ipixel_data2[93]), .DA94(ipixel_data2[94]), .DA95(ipixel_data2[95]), // Write Data
```

```
      .DA96(ipixel_data2[96]),       .DA97(ipixel_data2[97]),       .DA98(ipixel_data2[98]), .DA99(ipixel_data2[99]),  // Write Data
      .DA100(ipixel_data2[100]),     .DA101(ipixel_data2[101]),     .DA102(ipixel_data2[102]), .DA103(ipixel_data2[103]),  // Write Data
      .DA104(ipixel_data2[104]),     .DA105(ipixel_data2[105]),     .DA106(ipixel_data2[106]), .DA107(ipixel_data2[107]),  // Write Data
      .DA108(ipixel_data2[108]),     .DA109(ipixel_data2[109]),     .DA110(ipixel_data2[110]), .DA111(ipixel_data2[111]),  // Write Data
      .DA112(ipixel_data2[112]),     .DA113(ipixel_data2[113]),     .DA114(ipixel_data2[114]), .DA115(ipixel_data2[115]),  // Write Data
      .DA116(ipixel_data2[116]),     .DA117(ipixel_data2[117]),     .DA118(ipixel_data2[118]), .DA119(ipixel_data2[119]),  // Write Data
      .DA120(ipixel_data2[120]),     .DA121(ipixel_data2[121]),     .DA122(ipixel_data2[122]), .DA123(ipixel_data2[123]),  // Write Data
      .DA124(ipixel_data2[124]),     .DA125(ipixel_data2[125]),     .DA126(ipixel_data2[126]), .DA127(ipixel_data2[127]),  // Write Data
      // WRITE TEST SIGNALS
      .BISTEA(vss),
      .TWEA(vss), // Test write enable
      .TMEA(vss), // Test memory enable
      .TADRA0(iwrite_addr[0]),     .TADRA1(iwrite_addr[1]),     .TADRA2(iwrite_addr[2]), .TADRA3(iwrite_addr[3]),  // Write Test Address
      .TADRA4(iwrite_addr[4]), .TADRA5(iwrite_addr[5]), .TADRA6(iwrite_addr[6]),  // Write Test Address
      .TDA0(ipixel_data2[0]),       .TDA1(ipixel_data2[1]),       .TDA2(ipixel_data2[2]), .TDA3(ipixel_data2[3]),  // Write Test Data
      .TDA4(ipixel_data2[4]),       .TDA5(ipixel_data2[5]),       .TDA6(ipixel_data2[6]), .TDA7(ipixel_data2[7]),  // Write Test Data
      .TDA8(ipixel_data2[8]),       .TDA9(ipixel_data2[9]),       .TDA10(ipixel_data2[10]), .TDA11(ipixel_data2[11]),  // Write Test Data
      .TDA12(ipixel_data2[12]),     .TDA13(ipixel_data2[13]),     .TDA14(ipixel_data2[14]), .TDA15(ipixel_data2[15]),  // Write Test Data
```

```
      .TDA16(ipixel_data2[16]),     .TDA17(ipixel_data2[17]),     .TDA18(ipixel_data2[18]), .TDA19(ipixel_data2[19]),  // Write Test Data
      .TDA20(ipixel_data2[20]),     .TDA21(ipixel_data2[21]),     .TDA22(ipixel_data2[22]), .TDA23(ipixel_data2[23]),  // Write Test Data
      .TDA24(ipixel_data2[24]),     .TDA25(ipixel_data2[25]),     .TDA26(ipixel_data2[26]), .TDA27(ipixel_data2[27]),  // Write Test Data
      .TDA28(ipixel_data2[28]),     .TDA29(ipixel_data2[29]),     .TDA30(ipixel_data2[30]), .TDA31(ipixel_data2[31]),  // Write Test Data
      .TDA32(ipixel_data2[32]),     .TDA33(ipixel_data2[33]),     .TDA34(ipixel_data2[34]), .TDA35(ipixel_data2[35]),  // Write Test Data
      .TDA36(ipixel_data2[36]),     .TDA37(ipixel_data2[37]),     .TDA38(ipixel_data2[38]), .TDA39(ipixel_data2[39]),  // Write Test Data
      .TDA40(ipixel_data2[40]),     .TDA41(ipixel_data2[41]),     .TDA42(ipixel_data2[42]), .TDA43(ipixel_data2[43]),  // Write Test Data
      .TDA44(ipixel_data2[44]),     .TDA45(ipixel_data2[45]),     .TDA46(ipixel_data2[46]), .TDA47(ipixel_data2[47]),  // Write Test Data
      .TDA48(ipixel_data2[48]),     .TDA49(ipixel_data2[49]),     .TDA50(ipixel_data2[50]), .TDA51(ipixel_data2[51]),  // Write Test Data
      .TDA52(ipixel_data2[52]),     .TDA53(ipixel_data2[53]),     .TDA54(ipixel_data2[54]), .TDA55(ipixel_data2[55]),  // Write Test Data
      .TDA56(ipixel_data2[56]),     .TDA57(ipixel_data2[57]),     .TDA58(ipixel_data2[58]), .TDA59(ipixel_data2[59]),  // Write Test Data
      .TDA60(ipixel_data2[60]),     .TDA61(ipixel_data2[61]),     .TDA62(ipixel_data2[62]), .TDA63(ipixel_data2[63]),  // Write Test Data
      .TDA64(ipixel_data2[64]),     .TDA65(ipixel_data2[65]),     .TDA66(ipixel_data2[66]), .TDA67(ipixel_data2[67]),  // Write Test Data
      .TDA68(ipixel_data2[68]),     .TDA69(ipixel_data2[69]),     .TDA70(ipixel_data2[70]), .TDA71(ipixel_data2[71]),  // Write Test Data
      .TDA72(ipixel_data2[72]),     .TDA73(ipixel_data2[73]),     .TDA74(ipixel_data2[74]), .TDA75(ipixel_data2[75]),  // Write Test Data
      .TDA76(ipixel_data2[76]),     .TDA77(ipixel_data2[77]),     .TDA78(ipixel_data2[78]), .TDA79(ipixel_data2[79]),  // Write Test Data
```

```
      .TDA80(ipixel_data2[80]),     .TDA81(ipixel_data2[81]),     .TDA82(ipixel_data2[82]), .TDA83(ipixel_data2[83]),  // Write Test Data
      .TDA84(ipixel_data2[84]),     .TDA85(ipixel_data2[85]),     .TDA86(ipixel_data2[86]), .TDA87(ipixel_data2[87]),  // Write Test Data
      .TDA88(ipixel_data2[88]),     .TDA89(ipixel_data2[89]),     .TDA90(ipixel_data2[90]), .TDA91(ipixel_data2[91]),  // Write Test Data
      .TDA92(ipixel_data2[92]),     .TDA93(ipixel_data2[93]),     .TDA94(ipixel_data2[94]), .TDA95(ipixel_data2[95]),  // Write Test Data
      .TDA96(ipixel_data2[96]),     .TDA97(ipixel_data2[97]),     .TDA98(ipixel_data2[98]), .TDA99(ipixel_data2[99]),  // Write Test Data
      .TDA100(ipixel_data2[100]),   .TDA101(ipixel_data2[101]),   .TDA102(ipixel_data2[102]), .TDA103(ipixel_data2[103]),  // Write Test Data
      .TDA104(ipixel_data2[104]),   .TDA105(ipixel_data2[105]),   .TDA106(ipixel_data2[106]), .TDA107(ipixel_data2[107]),  // Write Test Data
      .TDA108(ipixel_data2[108]),   .TDA109(ipixel_data2[109]),   .TDA110(ipixel_data2[110]), .TDA111(ipixel_data2[111]),  // Write Test Data
      .TDA112(ipixel_data2[112]),   .TDA113(ipixel_data2[113]),   .TDA114(ipixel_data2[114]), .TDA115(ipixel_data2[115]),  // Write Test Data
      .TDA116(ipixel_data2[116]),   .TDA117(ipixel_data2[117]),   .TDA118(ipixel_data2[118]), .TDA119(ipixel_data2[119]),  // Write Test Data
      .TDA120(ipixel_data2[120]),   .TDA121(ipixel_data2[121]),   .TDA122(ipixel_data2[122]), .TDA123(ipixel_data2[123]),  // Write Test Data
      .TDA124(ipixel_data2[124]),   .TDA125(ipixel_data2[125]),   .TDA126(ipixel_data2[126]), .TDA127(ipixel_data2[127]),  // Write Test Data
      //READ TEST SIGNALS
      .BISTEB(vss),
      .TOEB(vss),
      .TMEB(vss),
      .TADRB0(q1_read_addr[0]),   .TADRB1(q1_read_addr[1]),   .TADRB2(q1_read_addr[2]), .TADRB3(q1_read_addr[3]),  // Read Test Address
      .TADRB4(q1_read_addr[4]), .TADRB5(q1_read_addr[5]), .TADRB6(q1_read_addr[6]),  // Read Test Address
```

```
      .AWTB(vss)
      );

`endif // !`ifdef USE_BEHAVE_MEM



   wire [6:0] q2_read_mem_addr;
   assign     q2_read_mem_addr = q2_read_addr[6:0];


`ifdef USE_BEHAVE_MEM
   dum_mem_p2 #(7,128) bank0_buff3(.iRCLK(sclk),
                                   .iWCLK(sclk),
                                   .iMER(q2_mem_re),
                                   .iMEW(imem_wen),
                                   .iWEN(imem_wew),
                                   .iRADR(q2_read_addr[6:0]),
                                   .iWADR(iwrite_addr),
                                   .iD(ipixel_data3),
                                   .oQ(buff3_out)
                                   );
`else // !`ifdef USE_BEHAVE_MEM
   rfsd2_80x128cm2sw0 ubank0_buff3
    (/*VRGIO   rfsd2_80x128cm2sw0   ipixel_data3   buff3_out   iwrite_addr   q2_read_addr imem_wen q2_mem_re null*/
// READ INTERFACE
.CLKB(iSCLK), // Read Clock
.OEB(vdd), // Output enable
.MEB(q2_mem_re), // Read enable
.ADRB0(q2_read_addr[0]), .ADRB1(q2_read_addr[1]), .ADRB2(q2_read_addr[2]), .ADRB3(q2_read_addr[3]), // Read Address
.ADRB4(q2_read_addr[4]), .ADRB5(q2_read_addr[5]), .ADRB6(q2_read_addr[6]), // Read Address
.QB0(buff3_out[0]), .QB1(buff3_out[1]), .QB2(buff3_out[2]), .QB3(buff3_out[3]), // Read Data
.QB4(buff3_out[4]), .QB5(buff3_out[5]), .QB6(buff3_out[6]), .QB7(buff3_out[7]), // Read Data
.QB8(buff3_out[8]), .QB9(buff3_out[9]), .QB10(buff3_out[10]), .QB11(buff3_out[11]), // Read Data
.QB12(buff3_out[12]), .QB13(buff3_out[13]), .QB14(buff3_out[14]), .QB15(buff3_out[15]), // Read Data
.QB16(buff3_out[16]), .QB17(buff3_out[17]), .QB18(buff3_out[18]), .QB19(buff3_out[19]), // Read Data
.QB20(buff3_out[20]), .QB21(buff3_out[21]), .QB22(buff3_out[22]), .QB23(buff3_out[23]), // Read Data
.QB24(buff3_out[24]), .QB25(buff3_out[25]), .QB26(buff3_out[26]), .QB27(buff3_out[27]), // Read Data
.QB28(buff3_out[28]), .QB29(buff3_out[29]), .QB30(buff3_out[30]), .QB31(buff3_out[31]), // Read Data
.QB32(buff3_out[32]), .QB33(buff3_out[33]), .QB34(buff3_out[34]), .QB35(buff3_out[35]), // Read Data
.QB36(buff3_out[36]), .QB37(buff3_out[37]), .QB38(buff3_out[38]), .QB39(buff3_out[39]), // Read Data
.QB40(buff3_out[40]), .QB41(buff3_out[41]), .QB42(buff3_out[42]), .QB43(buff3_out[43]), // Read Data
.QB44(buff3_out[44]), .QB45(buff3_out[45]), .QB46(buff3_out[46]), .QB47(buff3_out[47]), // Read Data
```

```
.QB48(buff3_out[48]), .QB49(buff3_out[49]), .QB50(buff3_out[50]), .QB51(buff3_out[51]), // Read Data
.QB52(buff3_out[52]), .QB53(buff3_out[53]), .QB54(buff3_out[54]), .QB55(buff3_out[55]), // Read Data
.QB56(buff3_out[56]), .QB57(buff3_out[57]), .QB58(buff3_out[58]), .QB59(buff3_out[59]), // Read Data
.QB60(buff3_out[60]), .QB61(buff3_out[61]), .QB62(buff3_out[62]), .QB63(buff3_out[63]), // Read Data
.QB64(buff3_out[64]), .QB65(buff3_out[65]), .QB66(buff3_out[66]), .QB67(buff3_out[67]), // Read Data
.QB68(buff3_out[68]), .QB69(buff3_out[69]), .QB70(buff3_out[70]), .QB71(buff3_out[71]), // Read Data
.QB72(buff3_out[72]), .QB73(buff3_out[73]), .QB74(buff3_out[74]), .QB75(buff3_out[75]), // Read Data
.QB76(buff3_out[76]), .QB77(buff3_out[77]), .QB78(buff3_out[78]), .QB79(buff3_out[79]), // Read Data
.QB80(buff3_out[80]), .QB81(buff3_out[81]), .QB82(buff3_out[82]), .QB83(buff3_out[83]), // Read Data
.QB84(buff3_out[84]), .QB85(buff3_out[85]), .QB86(buff3_out[86]), .QB87(buff3_out[87]), // Read Data
.QB88(buff3_out[88]), .QB89(buff3_out[89]), .QB90(buff3_out[90]), .QB91(buff3_out[91]), // Read Data
.QB92(buff3_out[92]), .QB93(buff3_out[93]), .QB94(buff3_out[94]), .QB95(buff3_out[95]), // Read Data
.QB96(buff3_out[96]), .QB97(buff3_out[97]), .QB98(buff3_out[98]), .QB99(buff3_out[99]), // Read Data
.QB100(buff3_out[100]), .QB101(buff3_out[101]), .QB102(buff3_out[102]), .QB103(buff3_out[103]), // Read Data
.QB104(buff3_out[104]), .QB105(buff3_out[105]), .QB106(buff3_out[106]), .QB107(buff3_out[107]), // Read Data
.QB108(buff3_out[108]), .QB109(buff3_out[109]), .QB110(buff3_out[110]), .QB111(buff3_out[111]), // Read Data
```

```
.QB112(buff3_out[112]), .QB113(buff3_out[113]), .QB114(buff3_out[114]), .QB115(buff3_out[115]), // Read Data
.QB116(buff3_out[116]), .QB117(buff3_out[117]), .QB118(buff3_out[118]), .QB119(buff3_out[119]), // Read Data
.QB120(buff3_out[120]), .QB121(buff3_out[121]), .QB122(buff3_out[122]), .QB123(buff3_out[123]), // Read Data
.QB124(buff3_out[124]), .QB125(buff3_out[125]), .QB126(buff3_out[126]), .QB127(buff3_out[127]), // Read Data
// WRITE INTERFACE
.CLKA(iSCLK), // Write Clock
.WEA(imem_wen), // Write enable
.MEA(vdd), // Memory enable
.ADRA0(iwrite_addr[0]), .ADRA1(iwrite_addr[1]), .ADRA2(iwrite_addr[2]), .ADRA3(iwrite_addr[3]), // Write Address
.ADRA4(iwrite_addr[4]), .ADRA5(iwrite_addr[5]), .ADRA6(iwrite_addr[6]), // Write Address
.DA0(ipixel_data3[0]), .DA1(ipixel_data3[1]), .DA2(ipixel_data3[2]), .DA3(ipixel_data3[3]), // Write Data
.DA4(ipixel_data3[4]), .DA5(ipixel_data3[5]), .DA6(ipixel_data3[6]), .DA7(ipixel_data3[7]), // Write Data
.DA8(ipixel_data3[8]), .DA9(ipixel_data3[9]), .DA10(ipixel_data3[10]), .DA11(ipixel_data3[11]), // Write Data
.DA12(ipixel_data3[12]), .DA13(ipixel_data3[13]), .DA14(ipixel_data3[14]), .DA15(ipixel_data3[15]), // Write Data
.DA16(ipixel_data3[16]), .DA17(ipixel_data3[17]), .DA18(ipixel_data3[18]), .DA19(ipixel_data3[19]), // Write Data
.DA20(ipixel_data3[20]), .DA21(ipixel_data3[21]), .DA22(ipixel_data3[22]), .DA23(ipixel_data3[23]), // Write Data
.DA24(ipixel_data3[24]), .DA25(ipixel_data3[25]), .DA26(ipixel_data3[26]), .DA27(ipixel_data3[27]), // Write Data
.DA28(ipixel_data3[28]), .DA29(ipixel_data3[29]), .DA30(ipixel_data3[30]), .DA31(ipixel_data3[31]), // Write Data
```

```
.DA32(ipixel_data3[32]), .DA33(ipixel_data3[33]), .DA34(ipixel_data3[34]), .DA35(ipixel_data3[35]), // Write Data
.DA36(ipixel_data3[36]), .DA37(ipixel_data3[37]), .DA38(ipixel_data3[38]), .DA39(ipixel_data3[39]), // Write Data
.DA40(ipixel_data3[40]), .DA41(ipixel_data3[41]), .DA42(ipixel_data3[42]), .DA43(ipixel_data3[43]), // Write Data
.DA44(ipixel_data3[44]), .DA45(ipixel_data3[45]), .DA46(ipixel_data3[46]), .DA47(ipixel_data3[47]), // Write Data
.DA48(ipixel_data3[48]), .DA49(ipixel_data3[49]), .DA50(ipixel_data3[50]), .DA51(ipixel_data3[51]), // Write Data
.DA52(ipixel_data3[52]), .DA53(ipixel_data3[53]), .DA54(ipixel_data3[54]), .DA55(ipixel_data3[55]), // Write Data
.DA56(ipixel_data3[56]), .DA57(ipixel_data3[57]), .DA58(ipixel_data3[58]), .DA59(ipixel_data3[59]), // Write Data
.DA60(ipixel_data3[60]), .DA61(ipixel_data3[61]), .DA62(ipixel_data3[62]), .DA63(ipixel_data3[63]), // Write Data
.DA64(ipixel_data3[64]), .DA65(ipixel_data3[65]), .DA66(ipixel_data3[66]), .DA67(ipixel_data3[67]), // Write Data
.DA68(ipixel_data3[68]), .DA69(ipixel_data3[69]), .DA70(ipixel_data3[70]), .DA71(ipixel_data3[71]), // Write Data
.DA72(ipixel_data3[72]), .DA73(ipixel_data3[73]), .DA74(ipixel_data3[74]), .DA75(ipixel_data3[75]), // Write Data
.DA76(ipixel_data3[76]), .DA77(ipixel_data3[77]), .DA78(ipixel_data3[78]), .DA79(ipixel_data3[79]), // Write Data
.DA80(ipixel_data3[80]), .DA81(ipixel_data3[81]), .DA82(ipixel_data3[82]), .DA83(ipixel_data3[83]), // Write Data
.DA84(ipixel_data3[84]), .DA85(ipixel_data3[85]), .DA86(ipixel_data3[86]), .DA87(ipixel_data3[87]), // Write Data
.DA88(ipixel_data3[88]), .DA89(ipixel_data3[89]), .DA90(ipixel_data3[90]), .DA91(ipixel_data3[91]), // Write Data
.DA92(ipixel_data3[92]), .DA93(ipixel_data3[93]), .DA94(ipixel_data3[94]), .DA95(ipixel_data3[95]), // Write Data
```

```
    .DA96(ipixel_data3[96]),    .DA97(ipixel_data3[97]),    .DA98(ipixel_data3[98]), .DA99(ipixel_data3[99]),  // Write Data
    .DA100(ipixel_data3[100]),  .DA101(ipixel_data3[101]),  .DA102(ipixel_data3[102]), .DA103(ipixel_data3[103]),  // Write Data
    .DA104(ipixel_data3[104]),  .DA105(ipixel_data3[105]),  .DA106(ipixel_data3[106]), .DA107(ipixel_data3[107]),  // Write Data
    .DA108(ipixel_data3[108]),  .DA109(ipixel_data3[109]),  .DA110(ipixel_data3[110]), .DA111(ipixel_data3[111]),  // Write Data
    .DA112(ipixel_data3[112]),  .DA113(ipixel_data3[113]),  .DA114(ipixel_data3[114]), .DA115(ipixel_data3[115]),  // Write Data
    .DA116(ipixel_data3[116]),  .DA117(ipixel_data3[117]),  .DA118(ipixel_data3[118]), .DA119(ipixel_data3[119]),  // Write Data
    .DA120(ipixel_data3[120]),  .DA121(ipixel_data3[121]),  .DA122(ipixel_data3[122]), .DA123(ipixel_data3[123]),  // Write Data
    .DA124(ipixel_data3[124]),  .DA125(ipixel_data3[125]),  .DA126(ipixel_data3[126]), .DA127(ipixel_data3[127]),  // Write Data
    // WRITE TEST SIGNALS
    .BISTEA(vss),
    .TWEA(vss), // Test write enable
    .TMEA(vss), // Test memory enable
    .TADRA0(iwrite_addr[0]),  .TADRA1(iwrite_addr[1]),  .TADRA2(iwrite_addr[2]), .TADRA3(iwrite_addr[3]), // Write Test Address
    .TADRA4(iwrite_addr[4]), .TADRA5(iwrite_addr[5]), .TADRA6(iwrite_addr[6]),  // Write Test Address
    .TDA0(ipixel_data3[0]),    .TDA1(ipixel_data3[1]),    .TDA2(ipixel_data3[2]), .TDA3(ipixel_data3[3]),  // Write Test Data
    .TDA4(ipixel_data3[4]),    .TDA5(ipixel_data3[5]),    .TDA6(ipixel_data3[6]), .TDA7(ipixel_data3[7]),  // Write Test Data
    .TDA8(ipixel_data3[8]),    .TDA9(ipixel_data3[9]),    .TDA10(ipixel_data3[10]), .TDA11(ipixel_data3[11]),  // Write Test Data
    .TDA12(ipixel_data3[12]),  .TDA13(ipixel_data3[13]),  .TDA14(ipixel_data3[14]), .TDA15(ipixel_data3[15]),  // Write Test Data
```

```
    .TDA16(ipixel_data3[16]),  .TDA17(ipixel_data3[17]),  .TDA18(ipixel_data3[18]), .TDA19(ipixel_data3[19]),  // Write Test Data
    .TDA20(ipixel_data3[20]),  .TDA21(ipixel_data3[21]),  .TDA22(ipixel_data3[22]), .TDA23(ipixel_data3[23]),  // Write Test Data
    .TDA24(ipixel_data3[24]),  .TDA25(ipixel_data3[25]),  .TDA26(ipixel_data3[26]), .TDA27(ipixel_data3[27]),  // Write Test Data
    .TDA28(ipixel_data3[28]),  .TDA29(ipixel_data3[29]),  .TDA30(ipixel_data3[30]), .TDA31(ipixel_data3[31]),  // Write Test Data
    .TDA32(ipixel_data3[32]),  .TDA33(ipixel_data3[33]),  .TDA34(ipixel_data3[34]), .TDA35(ipixel_data3[35]),  // Write Test Data
    .TDA36(ipixel_data3[36]),  .TDA37(ipixel_data3[37]),  .TDA38(ipixel_data3[38]), .TDA39(ipixel_data3[39]),  // Write Test Data
    .TDA40(ipixel_data3[40]),  .TDA41(ipixel_data3[41]),  .TDA42(ipixel_data3[42]), .TDA43(ipixel_data3[43]),  // Write Test Data
    .TDA44(ipixel_data3[44]),  .TDA45(ipixel_data3[45]),  .TDA46(ipixel_data3[46]), .TDA47(ipixel_data3[47]),  // Write Test Data
    .TDA48(ipixel_data3[48]),  .TDA49(ipixel_data3[49]),  .TDA50(ipixel_data3[50]), .TDA51(ipixel_data3[51]),  // Write Test Data
    .TDA52(ipixel_data3[52]),  .TDA53(ipixel_data3[53]),  .TDA54(ipixel_data3[54]), .TDA55(ipixel_data3[55]),  // Write Test Data
    .TDA56(ipixel_data3[56]),  .TDA57(ipixel_data3[57]),  .TDA58(ipixel_data3[58]), .TDA59(ipixel_data3[59]),  // Write Test Data
    .TDA60(ipixel_data3[60]),  .TDA61(ipixel_data3[61]),  .TDA62(ipixel_data3[62]), .TDA63(ipixel_data3[63]),  // Write Test Data
    .TDA64(ipixel_data3[64]),  .TDA65(ipixel_data3[65]),  .TDA66(ipixel_data3[66]), .TDA67(ipixel_data3[67]),  // Write Test Data
    .TDA68(ipixel_data3[68]),  .TDA69(ipixel_data3[69]),  .TDA70(ipixel_data3[70]), .TDA71(ipixel_data3[71]),  // Write Test Data
    .TDA72(ipixel_data3[72]),  .TDA73(ipixel_data3[73]),  .TDA74(ipixel_data3[74]), .TDA75(ipixel_data3[75]),  // Write Test Data
    .TDA76(ipixel_data3[76]),  .TDA77(ipixel_data3[77]),  .TDA78(ipixel_data3[78]), .TDA79(ipixel_data3[79]),  // Write Test Data
```

```
    .TDA80(ipixel_data3[80]),  .TDA81(ipixel_data3[81]),  .TDA82(ipixel_data3[82]), .TDA83(ipixel_data3[83]),  // Write Test Data
    .TDA84(ipixel_data3[84]),  .TDA85(ipixel_data3[85]),  .TDA86(ipixel_data3[86]), .TDA87(ipixel_data3[87]),  // Write Test Data
    .TDA88(ipixel_data3[88]),  .TDA89(ipixel_data3[89]),  .TDA90(ipixel_data3[90]), .TDA91(ipixel_data3[91]),  // Write Test Data
    .TDA92(ipixel_data3[92]),  .TDA93(ipixel_data3[93]),  .TDA94(ipixel_data3[94]), .TDA95(ipixel_data3[95]),  // Write Test Data
    .TDA96(ipixel_data3[96]),  .TDA97(ipixel_data3[97]),  .TDA98(ipixel_data3[98]), .TDA99(ipixel_data3[99]),  // Write Test Data
    .TDA100(ipixel_data3[100]),  .TDA101(ipixel_data3[101]),  .TDA102(ipixel_data3[102]), .TDA103(ipixel_data3[103]),  // Write Test Data
    .TDA104(ipixel_data3[104]),  .TDA105(ipixel_data3[105]),  .TDA106(ipixel_data3[106]), .TDA107(ipixel_data3[107]),  // Write Test Data
    .TDA108(ipixel_data3[108]),  .TDA109(ipixel_data3[109]),  .TDA110(ipixel_data3[110]), .TDA111(ipixel_data3[111]),  // Write Test Data
    .TDA112(ipixel_data3[112]),  .TDA113(ipixel_data3[113]),  .TDA114(ipixel_data3[114]), .TDA115(ipixel_data3[115]),  // Write Test Data
    .TDA116(ipixel_data3[116]),  .TDA117(ipixel_data3[117]),  .TDA118(ipixel_data3[118]), .TDA119(ipixel_data3[119]),  // Write Test Data
    .TDA120(ipixel_data3[120]),  .TDA121(ipixel_data3[121]),  .TDA122(ipixel_data3[122]), .TDA123(ipixel_data3[123]),  // Write Test Data
    .TDA124(ipixel_data3[124]),  .TDA125(ipixel_data3[125]),  .TDA126(ipixel_data3[126]), .TDA127(ipixel_data3[127]),  // Write Test Data
    //READ TEST SIGNALS
    .BISTEB(vss),
    .TOEB(vss),
    .TMEB(vss),
    .TADRB0(q2_read_addr[0]),  .TADRB1(q2_read_addr[1]),  .TADRB2(q2_read_addr[2]), .TADRB3(q2_read_addr[3]), // Read Test Address
    .TADRB4(q2_read_addr[4]), .TADRB5(q2_read_addr[5]), .TADRB6(q2_read_addr[6]),  // Read Test Address
```

```
    .AWTB(vss)
    );

`endif // !`ifdef USE_BEHAVE_MEM


`ifdef USE_BEHAVE_MEM
  dum_mem_p2 #(7,128) bank1_buff0(.iRCLK(sclk),
                                  .iWCLK(sclk),
                                  .iMER(imem_re),
                                  .iMEW(imem_wen),
                                  .iWEN(imem_wew),
                                  .iRADR(iread_addr[6:0]),
                                  .iWADR(iwrite_addr),
                                  .iD(ipixel_data4),
                                  .oQ(buff4_out)
                                  );
`else // !`ifdef USE_BEHAVE_MEM

  rfsd2_80x128cm2sw0 ubank1_buff0
   (/*VRGIO rfsd2_80x128cm2sw0 ipixel_data4 buff4_out iwrite_addr iread_addr imem_wen imem_re null*/
    // READ INTERFACE
    .CLKB(iSCLK), // Read Clock
    .OEB(vdd), // Output enable
    .MEB(imem_re), // Read enable
```

```
      .ADRB0(iread_addr[0]), .ADRB1(iread_addr[1]), .ADRB2(iread_addr[2]), .ADRB3(iread_addr[3]),  // Read Address
      .ADRB4(iread_addr[4]), .ADRB5(iread_addr[5]), .ADRB6(iread_addr[6]),  // Read Address
      .QB0(buff4_out[0]), .QB1(buff4_out[1]), .QB2(buff4_out[2]), .QB3(buff4_out[3]),  // Read Data
      .QB4(buff4_out[4]), .QB5(buff4_out[5]), .QB6(buff4_out[6]), .QB7(buff4_out[7]),  // Read Data
      .QB8(buff4_out[8]), .QB9(buff4_out[9]), .QB10(buff4_out[10]), .QB11(buff4_out[11]),  // Read Data
      .QB12(buff4_out[12]), .QB13(buff4_out[13]), .QB14(buff4_out[14]), .QB15(buff4_out[15]),  // Read Data
      .QB16(buff4_out[16]), .QB17(buff4_out[17]), .QB18(buff4_out[18]), .QB19(buff4_out[19]),  // Read Data
      .QB20(buff4_out[20]), .QB21(buff4_out[21]), .QB22(buff4_out[22]), .QB23(buff4_out[23]),  // Read Data
      .QB24(buff4_out[24]), .QB25(buff4_out[25]), .QB26(buff4_out[26]), .QB27(buff4_out[27]),  // Read Data
      .QB28(buff4_out[28]), .QB29(buff4_out[29]), .QB30(buff4_out[30]), .QB31(buff4_out[31]),  // Read Data
      .QB32(buff4_out[32]), .QB33(buff4_out[33]), .QB34(buff4_out[34]), .QB35(buff4_out[35]),  // Read Data
      .QB36(buff4_out[36]), .QB37(buff4_out[37]), .QB38(buff4_out[38]), .QB39(buff4_out[39]),  // Read Data
      .QB40(buff4_out[40]), .QB41(buff4_out[41]), .QB42(buff4_out[42]), .QB43(buff4_out[43]),  // Read Data
      .QB44(buff4_out[44]), .QB45(buff4_out[45]), .QB46(buff4_out[46]), .QB47(buff4_out[47]),  // Read Data
      .QB48(buff4_out[48]), .QB49(buff4_out[49]), .QB50(buff4_out[50]), .QB51(buff4_out[51]),  // Read Data
      .QB52(buff4_out[52]), .QB53(buff4_out[53]), .QB54(buff4_out[54]), .QB55(buff4_out[55]),  // Read Data
```

```
      .QB56(buff4_out[56]), .QB57(buff4_out[57]), .QB58(buff4_out[58]), .QB59(buff4_out[59]),  // Read Data
      .QB60(buff4_out[60]), .QB61(buff4_out[61]), .QB62(buff4_out[62]), .QB63(buff4_out[63]),  // Read Data
      .QB64(buff4_out[64]), .QB65(buff4_out[65]), .QB66(buff4_out[66]), .QB67(buff4_out[67]),  // Read Data
      .QB68(buff4_out[68]), .QB69(buff4_out[69]), .QB70(buff4_out[70]), .QB71(buff4_out[71]),  // Read Data
      .QB72(buff4_out[72]), .QB73(buff4_out[73]), .QB74(buff4_out[74]), .QB75(buff4_out[75]),  // Read Data
      .QB76(buff4_out[76]), .QB77(buff4_out[77]), .QB78(buff4_out[78]), .QB79(buff4_out[79]),  // Read Data
      .QB80(buff4_out[80]), .QB81(buff4_out[81]), .QB82(buff4_out[82]), .QB83(buff4_out[83]),  // Read Data
      .QB84(buff4_out[84]), .QB85(buff4_out[85]), .QB86(buff4_out[86]), .QB87(buff4_out[87]),  // Read Data
      .QB88(buff4_out[88]), .QB89(buff4_out[89]), .QB90(buff4_out[90]), .QB91(buff4_out[91]),  // Read Data
      .QB92(buff4_out[92]), .QB93(buff4_out[93]), .QB94(buff4_out[94]), .QB95(buff4_out[95]),  // Read Data
      .QB96(buff4_out[96]), .QB97(buff4_out[97]), .QB98(buff4_out[98]), .QB99(buff4_out[99]),  // Read Data
      .QB100(buff4_out[100]), .QB101(buff4_out[101]), .QB102(buff4_out[102]), .QB103(buff4_out[103]),  // Read Data
      .QB104(buff4_out[104]), .QB105(buff4_out[105]), .QB106(buff4_out[106]), .QB107(buff4_out[107]),  // Read Data
      .QB108(buff4_out[108]), .QB109(buff4_out[109]), .QB110(buff4_out[110]), .QB111(buff4_out[111]),  // Read Data
      .QB112(buff4_out[112]), .QB113(buff4_out[113]), .QB114(buff4_out[114]), .QB115(buff4_out[115]),  // Read Data
      .QB116(buff4_out[116]), .QB117(buff4_out[117]), .QB118(buff4_out[118]), .QB119(buff4_out[119]),  // Read Data
```

```
      .QB120(buff4_out[120]), .QB121(buff4_out[121]), .QB122(buff4_out[122]), .QB123(buff4_out[123]),  // Read Data
      .QB124(buff4_out[124]), .QB125(buff4_out[125]), .QB126(buff4_out[126]), .QB127(buff4_out[127]),  // Read Data
      // WRITE INTERFACE
      .CLKA(iSCLK), // Write Clock
      .WEA(imem_wen), // Write enable
      .MEA(vdd), // Memory enable
      .ADRA0(iwrite_addr[0]), .ADRA1(iwrite_addr[1]), .ADRA2(iwrite_addr[2]), .ADRA3(iwrite_addr[3]),  // Write Address
      .ADRA4(iwrite_addr[4]), .ADRA5(iwrite_addr[5]), .ADRA6(iwrite_addr[6]),  // Write Address
      .DA0(ipixel_data4[0]), .DA1(ipixel_data4[1]), .DA2(ipixel_data4[2]), .DA3(ipixel_data4[3]),  // Write Data
      .DA4(ipixel_data4[4]), .DA5(ipixel_data4[5]), .DA6(ipixel_data4[6]), .DA7(ipixel_data4[7]),  // Write Data
      .DA8(ipixel_data4[8]), .DA9(ipixel_data4[9]), .DA10(ipixel_data4[10]), .DA11(ipixel_data4[11]),  // Write Data
      .DA12(ipixel_data4[12]), .DA13(ipixel_data4[13]), .DA14(ipixel_data4[14]), .DA15(ipixel_data4[15]),  // Write Data
      .DA16(ipixel_data4[16]), .DA17(ipixel_data4[17]), .DA18(ipixel_data4[18]), .DA19(ipixel_data4[19]),  // Write Data
      .DA20(ipixel_data4[20]), .DA21(ipixel_data4[21]), .DA22(ipixel_data4[22]), .DA23(ipixel_data4[23]),  // Write Data
      .DA24(ipixel_data4[24]), .DA25(ipixel_data4[25]), .DA26(ipixel_data4[26]), .DA27(ipixel_data4[27]),  // Write Data
      .DA28(ipixel_data4[28]), .DA29(ipixel_data4[29]), .DA30(ipixel_data4[30]), .DA31(ipixel_data4[31]),  // Write Data
      .DA32(ipixel_data4[32]), .DA33(ipixel_data4[33]), .DA34(ipixel_data4[34]), .DA35(ipixel_data4[35]),  // Write Data
      .DA36(ipixel_data4[36]), .DA37(ipixel_data4[37]), .DA38(ipixel_data4[38]), .DA39(ipixel_data4[39]),  // Write Data
```

```
      .DA40(ipixel_data4[40]), .DA41(ipixel_data4[41]), .DA42(ipixel_data4[42]), .DA43(ipixel_data4[43]),  // Write Data
      .DA44(ipixel_data4[44]), .DA45(ipixel_data4[45]), .DA46(ipixel_data4[46]), .DA47(ipixel_data4[47]),  // Write Data
      .DA48(ipixel_data4[48]), .DA49(ipixel_data4[49]), .DA50(ipixel_data4[50]), .DA51(ipixel_data4[51]),  // Write Data
      .DA52(ipixel_data4[52]), .DA53(ipixel_data4[53]), .DA54(ipixel_data4[54]), .DA55(ipixel_data4[55]),  // Write Data
      .DA56(ipixel_data4[56]), .DA57(ipixel_data4[57]), .DA58(ipixel_data4[58]), .DA59(ipixel_data4[59]),  // Write Data
      .DA60(ipixel_data4[60]), .DA61(ipixel_data4[61]), .DA62(ipixel_data4[62]), .DA63(ipixel_data4[63]),  // Write Data
      .DA64(ipixel_data4[64]), .DA65(ipixel_data4[65]), .DA66(ipixel_data4[66]), .DA67(ipixel_data4[67]),  // Write Data
      .DA68(ipixel_data4[68]), .DA69(ipixel_data4[69]), .DA70(ipixel_data4[70]), .DA71(ipixel_data4[71]),  // Write Data
      .DA72(ipixel_data4[72]), .DA73(ipixel_data4[73]), .DA74(ipixel_data4[74]), .DA75(ipixel_data4[75]),  // Write Data
      .DA76(ipixel_data4[76]), .DA77(ipixel_data4[77]), .DA78(ipixel_data4[78]), .DA79(ipixel_data4[79]),  // Write Data
      .DA80(ipixel_data4[80]), .DA81(ipixel_data4[81]), .DA82(ipixel_data4[82]), .DA83(ipixel_data4[83]),  // Write Data
      .DA84(ipixel_data4[84]), .DA85(ipixel_data4[85]), .DA86(ipixel_data4[86]), .DA87(ipixel_data4[87]),  // Write Data
      .DA88(ipixel_data4[88]), .DA89(ipixel_data4[89]), .DA90(ipixel_data4[90]), .DA91(ipixel_data4[91]),  // Write Data
      .DA92(ipixel_data4[92]), .DA93(ipixel_data4[93]), .DA94(ipixel_data4[94]), .DA95(ipixel_data4[95]),  // Write Data
      .DA96(ipixel_data4[96]), .DA97(ipixel_data4[97]), .DA98(ipixel_data4[98]), .DA99(ipixel_data4[99]),  // Write Data
      .DA100(ipixel_data4[100]), .DA101(ipixel_data4[101]), .DA102(ipixel_data4[102]), .DA103(ipixel_data4[103]),  // Write Data
```

```
    .DA104(ipixel_data4[104]),    .DA105(ipixel_data4[105]),    .DA106(ipixel_data4[106]),
.DA107(ipixel_data4[107]),  // Write Data

    .DA108(ipixel_data4[108]),    .DA109(ipixel_data4[109]),    .DA110(ipixel_data4[110]),
.DA111(ipixel_data4[111]),  // Write Data

    .DA112(ipixel_data4[112]),    .DA113(ipixel_data4[113]),    .DA114(ipixel_data4[114]),
.DA115(ipixel_data4[115]),  // Write Data

    .DA116(ipixel_data4[116]),    .DA117(ipixel_data4[117]),    .DA118(ipixel_data4[118]),
.DA119(ipixel_data4[119]),  // Write Data

    .DA120(ipixel_data4[120]),    .DA121(ipixel_data4[121]),    .DA122(ipixel_data4[122]),
.DA123(ipixel_data4[123]),  // Write Data

    .DA124(ipixel_data4[124]),    .DA125(ipixel_data4[125]),    .DA126(ipixel_data4[126]),
.DA127(ipixel_data4[127]),  // Write Data

    // WRITE TEST SIGNALS

    .BISTEA(vss),

    .TWEA(vss), // Test write enable

    .TMEA(vss), // Test memory enable

    .TADRA0(iwrite_addr[0]),    .TADRA1(iwrite_addr[1]),    .TADRA2(iwrite_addr[2]),
.TADRA3(iwrite_addr[3]),  // Write Test Address

    .TADRA4(iwrite_addr[4]), .TADRA5(iwrite_addr[5]), .TADRA6(iwrite_addr[6]),  // Write
Test Address

    .TDA0(ipixel_data4[0]),      .TDA1(ipixel_data4[1]),      .TDA2(ipixel_data4[2]),
.TDA3(ipixel_data4[3]),  // Write Test Data

    .TDA4(ipixel_data4[4]),      .TDA5(ipixel_data4[5]),      .TDA6(ipixel_data4[6]),
.TDA7(ipixel_data4[7]),  // Write Test Data

    .TDA8(ipixel_data4[8]),      .TDA9(ipixel_data4[9]),      .TDA10(ipixel_data4[10]),
.TDA11(ipixel_data4[11]),  // Write Test Data

    .TDA12(ipixel_data4[12]),    .TDA13(ipixel_data4[13]),    .TDA14(ipixel_data4[14]),
.TDA15(ipixel_data4[15]),  // Write Test Data

    .TDA16(ipixel_data4[16]),    .TDA17(ipixel_data4[17]),    .TDA18(ipixel_data4[18]),
.TDA19(ipixel_data4[19]),  // Write Test Data

    .TDA20(ipixel_data4[20]),    .TDA21(ipixel_data4[21]),    .TDA22(ipixel_data4[22]),
.TDA23(ipixel_data4[23]),  // Write Test Data
```

```
    .TDA24(ipixel_data4[24]),    .TDA25(ipixel_data4[25]),    .TDA26(ipixel_data4[26]),
.TDA27(ipixel_data4[27]),  // Write Test Data

    .TDA28(ipixel_data4[28]),    .TDA29(ipixel_data4[29]),    .TDA30(ipixel_data4[30]),
.TDA31(ipixel_data4[31]),  // Write Test Data

    .TDA32(ipixel_data4[32]),    .TDA33(ipixel_data4[33]),    .TDA34(ipixel_data4[34]),
.TDA35(ipixel_data4[35]),  // Write Test Data

    .TDA36(ipixel_data4[36]),    .TDA37(ipixel_data4[37]),    .TDA38(ipixel_data4[38]),
.TDA39(ipixel_data4[39]),  // Write Test Data

    .TDA40(ipixel_data4[40]),    .TDA41(ipixel_data4[41]),    .TDA42(ipixel_data4[42]),
.TDA43(ipixel_data4[43]),  // Write Test Data

    .TDA44(ipixel_data4[44]),    .TDA45(ipixel_data4[45]),    .TDA46(ipixel_data4[46]),
.TDA47(ipixel_data4[47]),  // Write Test Data

    .TDA48(ipixel_data4[48]),    .TDA49(ipixel_data4[49]),    .TDA50(ipixel_data4[50]),
.TDA51(ipixel_data4[51]),  // Write Test Data

    .TDA52(ipixel_data4[52]),    .TDA53(ipixel_data4[53]),    .TDA54(ipixel_data4[54]),
.TDA55(ipixel_data4[55]),  // Write Test Data

    .TDA56(ipixel_data4[56]),    .TDA57(ipixel_data4[57]),    .TDA58(ipixel_data4[58]),
.TDA59(ipixel_data4[59]),  // Write Test Data

    .TDA60(ipixel_data4[60]),    .TDA61(ipixel_data4[61]),    .TDA62(ipixel_data4[62]),
.TDA63(ipixel_data4[63]),  // Write Test Data

    .TDA64(ipixel_data4[64]),    .TDA65(ipixel_data4[65]),    .TDA66(ipixel_data4[66]),
.TDA67(ipixel_data4[67]),  // Write Test Data

    .TDA68(ipixel_data4[68]),    .TDA69(ipixel_data4[69]),    .TDA70(ipixel_data4[70]),
.TDA71(ipixel_data4[71]),  // Write Test Data

    .TDA72(ipixel_data4[72]),    .TDA73(ipixel_data4[73]),    .TDA74(ipixel_data4[74]),
.TDA75(ipixel_data4[75]),  // Write Test Data

    .TDA76(ipixel_data4[76]),    .TDA77(ipixel_data4[77]),    .TDA78(ipixel_data4[78]),
.TDA79(ipixel_data4[79]),  // Write Test Data

    .TDA80(ipixel_data4[80]),    .TDA81(ipixel_data4[81]),    .TDA82(ipixel_data4[82]),
.TDA83(ipixel_data4[83]),  // Write Test Data

    .TDA84(ipixel_data4[84]),    .TDA85(ipixel_data4[85]),    .TDA86(ipixel_data4[86]),
.TDA87(ipixel_data4[87]),  // Write Test Data
```

```
    .TDA88(ipixel_data4[88]),    .TDA89(ipixel_data4[89]),    .TDA90(ipixel_data4[90]),
.TDA91(ipixel_data4[91]),  // Write Test Data

    .TDA92(ipixel_data4[92]),    .TDA93(ipixel_data4[93]),    .TDA94(ipixel_data4[94]),
.TDA95(ipixel_data4[95]),  // Write Test Data

    .TDA96(ipixel_data4[96]),    .TDA97(ipixel_data4[97]),    .TDA98(ipixel_data4[98]),
.TDA99(ipixel_data4[99]),  // Write Test Data

    .TDA100(ipixel_data4[100]),  .TDA101(ipixel_data4[101]),  .TDA102(ipixel_data4[102]),
.TDA103(ipixel_data4[103]),  // Write Test Data

    .TDA104(ipixel_data4[104]),  .TDA105(ipixel_data4[105]),  .TDA106(ipixel_data4[106]),
.TDA107(ipixel_data4[107]),  // Write Test Data

    .TDA108(ipixel_data4[108]),  .TDA109(ipixel_data4[109]),  .TDA110(ipixel_data4[110]),
.TDA111(ipixel_data4[111]),  // Write Test Data

    .TDA112(ipixel_data4[112]),  .TDA113(ipixel_data4[113]),  .TDA114(ipixel_data4[114]),
.TDA115(ipixel_data4[115]),  // Write Test Data

    .TDA116(ipixel_data4[116]),  .TDA117(ipixel_data4[117]),  .TDA118(ipixel_data4[118]),
.TDA119(ipixel_data4[119]),  // Write Test Data

    .TDA120(ipixel_data4[120]),  .TDA121(ipixel_data4[121]),  .TDA122(ipixel_data4[122]),
.TDA123(ipixel_data4[123]),  // Write Test Data

    .TDA124(ipixel_data4[124]),  .TDA125(ipixel_data4[125]),  .TDA126(ipixel_data4[126]),
.TDA127(ipixel_data4[127]),  // Write Test Data

    //READ TEST SIGNALS

    .BISTEB(vss),

    .TOEB(vss),

    .TMEB(vss),

    .TADRB0(iread_addr[0]),      .TADRB1(iread_addr[1]),      .TADRB2(iread_addr[2]),
.TADRB3(iread_addr[3]),  // Read Test Address

    .TADRB4(iread_addr[4]),  .TADRB5(iread_addr[5]),  .TADRB6(iread_addr[6]),    //  Read
Test Address

    .AWTB(vss)

    );

```

```
`endif // !`ifdef USE_BEHAVE_MEM


`ifdef USE_BEHAVE_MEM

  dum_mem_p2 #(7,128) bank1_buff1(.iRCLK(sclk),

                                  .iWCLK(sclk),

                                  .iMER(q0_mem_re),

                                  .iMEW(imem_wen),

                                  .iWEN(imem_wew),

                                  .iRADR(q0_read_addr[6:0]),

                                  .iWADR(iwrite_addr),

                                  .iD(ipixel_data5),

                                  .oQ(buff5_out)

                                  );

`else // !`ifdef USE_BEHAVE_MEM

  rfsd2_80x128cm2sw0 ubank1_buff1

  (/*VRGIO   rfsd2_80x128cm2sw0   ipixel_data5   buff5_out   iwrite_addr   q0_read_addr
imem_wen q0_mem_re null*/

    // READ INTERFACE

    .CLKB(iSCLK), // Read Clock

    .OEB(vdd), // Output enable

    .MEB(q0_mem_re), // Read enable

    .ADRB0(q0_read_addr[0]),    .ADRB1(q0_read_addr[1]),    .ADRB2(q0_read_addr[2]),
.ADRB3(q0_read_addr[3]),  // Read Address

    .ADRB4(q0_read_addr[4]),   .ADRB5(q0_read_addr[5]),   .ADRB6(q0_read_addr[6]),     //
Read Address

    .QB0(buff5_out[0]), .QB1(buff5_out[1]), .QB2(buff5_out[2]), .QB3(buff5_out[3]),  // Read
Data
```

```
    .QB4(buff5_out[4]), .QB5(buff5_out[5]), .QB6(buff5_out[6]), .QB7(buff5_out[7]), // Read
Data
    .QB8(buff5_out[8]), .QB9(buff5_out[9]), .QB10(buff5_out[10]), .QB11(buff5_out[11]), //
Read Data
    .QB12(buff5_out[12]), .QB13(buff5_out[13]), .QB14(buff5_out[14]),
.QB15(buff5_out[15]), // Read Data
    .QB16(buff5_out[16]), .QB17(buff5_out[17]), .QB18(buff5_out[18]),
.QB19(buff5_out[19]), // Read Data
    .QB20(buff5_out[20]), .QB21(buff5_out[21]), .QB22(buff5_out[22]),
.QB23(buff5_out[23]), // Read Data
    .QB24(buff5_out[24]), .QB25(buff5_out[25]), .QB26(buff5_out[26]),
.QB27(buff5_out[27]), // Read Data
    .QB28(buff5_out[28]), .QB29(buff5_out[29]), .QB30(buff5_out[30]),
.QB31(buff5_out[31]), // Read Data
    .QB32(buff5_out[32]), .QB33(buff5_out[33]), .QB34(buff5_out[34]),
.QB35(buff5_out[35]), // Read Data
    .QB36(buff5_out[36]), .QB37(buff5_out[37]), .QB38(buff5_out[38]),
.QB39(buff5_out[39]), // Read Data
    .QB40(buff5_out[40]), .QB41(buff5_out[41]), .QB42(buff5_out[42]),
.QB43(buff5_out[43]), // Read Data
    .QB44(buff5_out[44]), .QB45(buff5_out[45]), .QB46(buff5_out[46]),
.QB47(buff5_out[47]), // Read Data
    .QB48(buff5_out[48]), .QB49(buff5_out[49]), .QB50(buff5_out[50]),
.QB51(buff5_out[51]), // Read Data
    .QB52(buff5_out[52]), .QB53(buff5_out[53]), .QB54(buff5_out[54]),
.QB55(buff5_out[55]), // Read Data
    .QB56(buff5_out[56]), .QB57(buff5_out[57]), .QB58(buff5_out[58]),
.QB59(buff5_out[59]), // Read Data
    .QB60(buff5_out[60]), .QB61(buff5_out[61]), .QB62(buff5_out[62]),
.QB63(buff5_out[63]), // Read Data
    .QB64(buff5_out[64]), .QB65(buff5_out[65]), .QB66(buff5_out[66]),
.QB67(buff5_out[67]), // Read Data
```

```
    .QB68(buff5_out[68]), .QB69(buff5_out[69]), .QB70(buff5_out[70]),
.QB71(buff5_out[71]), // Read Data
    .QB72(buff5_out[72]), .QB73(buff5_out[73]), .QB74(buff5_out[74]),
.QB75(buff5_out[75]), // Read Data
    .QB76(buff5_out[76]), .QB77(buff5_out[77]), .QB78(buff5_out[78]),
.QB79(buff5_out[79]), // Read Data
    .QB80(buff5_out[80]), .QB81(buff5_out[81]), .QB82(buff5_out[82]),
.QB83(buff5_out[83]), // Read Data
    .QB84(buff5_out[84]), .QB85(buff5_out[85]), .QB86(buff5_out[86]),
.QB87(buff5_out[87]), // Read Data
    .QB88(buff5_out[88]), .QB89(buff5_out[89]), .QB90(buff5_out[90]),
.QB91(buff5_out[91]), // Read Data
    .QB92(buff5_out[92]), .QB93(buff5_out[93]), .QB94(buff5_out[94]),
.QB95(buff5_out[95]), // Read Data
    .QB96(buff5_out[96]), .QB97(buff5_out[97]), .QB98(buff5_out[98]),
.QB99(buff5_out[99]), // Read Data
    .QB100(buff5_out[100]), .QB101(buff5_out[101]), .QB102(buff5_out[102]),
.QB103(buff5_out[103]), // Read Data
    .QB104(buff5_out[104]), .QB105(buff5_out[105]), .QB106(buff5_out[106]),
.QB107(buff5_out[107]), // Read Data
    .QB108(buff5_out[108]), .QB109(buff5_out[109]), .QB110(buff5_out[110]),
.QB111(buff5_out[111]), // Read Data
    .QB112(buff5_out[112]), .QB113(buff5_out[113]), .QB114(buff5_out[114]),
.QB115(buff5_out[115]), // Read Data
    .QB116(buff5_out[116]), .QB117(buff5_out[117]), .QB118(buff5_out[118]),
.QB119(buff5_out[119]), // Read Data
    .QB120(buff5_out[120]), .QB121(buff5_out[121]), .QB122(buff5_out[122]),
.QB123(buff5_out[123]), // Read Data
    .QB124(buff5_out[124]), .QB125(buff5_out[125]), .QB126(buff5_out[126]),
.QB127(buff5_out[127]), // Read Data
    // WRITE INTERFACE
    .CLKA(iSCLK), // Write Clock
```

```
    .WEA(imem_wen), // Write enable
    .MEA(vdd), // Memory enable
    .ADRA0(iwrite_addr[0]), .ADRA1(iwrite_addr[1]), .ADRA2(iwrite_addr[2]),
.ADRA3(iwrite_addr[3]), // Write Address
    .ADRA4(iwrite_addr[4]), .ADRA5(iwrite_addr[5]), .ADRA6(iwrite_addr[6]), // Write
Address
    .DA0(ipixel_data5[0]), .DA1(ipixel_data5[1]), .DA2(ipixel_data5[2]),
.DA3(ipixel_data5[3]), // Write Data
    .DA4(ipixel_data5[4]), .DA5(ipixel_data5[5]), .DA6(ipixel_data5[6]),
.DA7(ipixel_data5[7]), // Write Data
    .DA8(ipixel_data5[8]), .DA9(ipixel_data5[9]), .DA10(ipixel_data5[10]),
.DA11(ipixel_data5[11]), // Write Data
    .DA12(ipixel_data5[12]), .DA13(ipixel_data5[13]), .DA14(ipixel_data5[14]),
.DA15(ipixel_data5[15]), // Write Data
    .DA16(ipixel_data5[16]), .DA17(ipixel_data5[17]), .DA18(ipixel_data5[18]),
.DA19(ipixel_data5[19]), // Write Data
    .DA20(ipixel_data5[20]), .DA21(ipixel_data5[21]), .DA22(ipixel_data5[22]),
.DA23(ipixel_data5[23]), // Write Data
    .DA24(ipixel_data5[24]), .DA25(ipixel_data5[25]), .DA26(ipixel_data5[26]),
.DA27(ipixel_data5[27]), // Write Data
    .DA28(ipixel_data5[28]), .DA29(ipixel_data5[29]), .DA30(ipixel_data5[30]),
.DA31(ipixel_data5[31]), // Write Data
    .DA32(ipixel_data5[32]), .DA33(ipixel_data5[33]), .DA34(ipixel_data5[34]),
.DA35(ipixel_data5[35]), // Write Data
    .DA36(ipixel_data5[36]), .DA37(ipixel_data5[37]), .DA38(ipixel_data5[38]),
.DA39(ipixel_data5[39]), // Write Data
    .DA40(ipixel_data5[40]), .DA41(ipixel_data5[41]), .DA42(ipixel_data5[42]),
.DA43(ipixel_data5[43]), // Write Data
    .DA44(ipixel_data5[44]), .DA45(ipixel_data5[45]), .DA46(ipixel_data5[46]),
.DA47(ipixel_data5[47]), // Write Data
    .DA48(ipixel_data5[48]), .DA49(ipixel_data5[49]), .DA50(ipixel_data5[50]),
.DA51(ipixel_data5[51]), // Write Data
```

```
    .DA52(ipixel_data5[52]), .DA53(ipixel_data5[53]), .DA54(ipixel_data5[54]),
.DA55(ipixel_data5[55]), // Write Data
    .DA56(ipixel_data5[56]), .DA57(ipixel_data5[57]), .DA58(ipixel_data5[58]),
.DA59(ipixel_data5[59]), // Write Data
    .DA60(ipixel_data5[60]), .DA61(ipixel_data5[61]), .DA62(ipixel_data5[62]),
.DA63(ipixel_data5[63]), // Write Data
    .DA64(ipixel_data5[64]), .DA65(ipixel_data5[65]), .DA66(ipixel_data5[66]),
.DA67(ipixel_data5[67]), // Write Data
    .DA68(ipixel_data5[68]), .DA69(ipixel_data5[69]), .DA70(ipixel_data5[70]),
.DA71(ipixel_data5[71]), // Write Data
    .DA72(ipixel_data5[72]), .DA73(ipixel_data5[73]), .DA74(ipixel_data5[74]),
.DA75(ipixel_data5[75]), // Write Data
    .DA76(ipixel_data5[76]), .DA77(ipixel_data5[77]), .DA78(ipixel_data5[78]),
.DA79(ipixel_data5[79]), // Write Data
    .DA80(ipixel_data5[80]), .DA81(ipixel_data5[81]), .DA82(ipixel_data5[82]),
.DA83(ipixel_data5[83]), // Write Data
    .DA84(ipixel_data5[84]), .DA85(ipixel_data5[85]), .DA86(ipixel_data5[86]),
.DA87(ipixel_data5[87]), // Write Data
    .DA88(ipixel_data5[88]), .DA89(ipixel_data5[89]), .DA90(ipixel_data5[90]),
.DA91(ipixel_data5[91]), // Write Data
    .DA92(ipixel_data5[92]), .DA93(ipixel_data5[93]), .DA94(ipixel_data5[94]),
.DA95(ipixel_data5[95]), // Write Data
    .DA96(ipixel_data5[96]), .DA97(ipixel_data5[97]), .DA98(ipixel_data5[98]),
.DA99(ipixel_data5[99]), // Write Data
    .DA100(ipixel_data5[100]), .DA101(ipixel_data5[101]), .DA102(ipixel_data5[102]),
.DA103(ipixel_data5[103]), // Write Data
    .DA104(ipixel_data5[104]), .DA105(ipixel_data5[105]), .DA106(ipixel_data5[106]),
.DA107(ipixel_data5[107]), // Write Data
    .DA108(ipixel_data5[108]), .DA109(ipixel_data5[109]), .DA110(ipixel_data5[110]),
.DA111(ipixel_data5[111]), // Write Data
    .DA112(ipixel_data5[112]), .DA113(ipixel_data5[113]), .DA114(ipixel_data5[114]),
.DA115(ipixel_data5[115]), // Write Data
```

```
        .DA116(ipixel_data5[116]),      .DA117(ipixel_data5[117]),      .DA118(ipixel_data5[118]),
.DA119(ipixel_data5[119]),  // Write Data
        .DA120(ipixel_data5[120]),      .DA121(ipixel_data5[121]),      .DA122(ipixel_data5[122]),
.DA123(ipixel_data5[123]),  // Write Data
        .DA124(ipixel_data5[124]),      .DA125(ipixel_data5[125]),      .DA126(ipixel_data5[126]),
.DA127(ipixel_data5[127]),  // Write Data
        // WRITE TEST SIGNALS
        .BISTEA(vss),
        .TWEA(vss), // Test write enable
        .TMEA(vss), // Test memory enable
        .TADRA0(iwrite_addr[0]),        .TADRA1(iwrite_addr[1]),        .TADRA2(iwrite_addr[2]),
.TADRA3(iwrite_addr[3]),  // Write Test Address
        .TADRA4(iwrite_addr[4]), .TADRA5(iwrite_addr[5]), .TADRA6(iwrite_addr[6]),  // Write
Test Address
        .TDA0(ipixel_data5[0]),             .TDA1(ipixel_data5[1]),             .TDA2(ipixel_data5[2]),
.TDA3(ipixel_data5[3]),  // Write Test Data
        .TDA4(ipixel_data5[4]),             .TDA5(ipixel_data5[5]),             .TDA6(ipixel_data5[6]),
.TDA7(ipixel_data5[7]),  // Write Test Data
        .TDA8(ipixel_data5[8]),           .TDA9(ipixel_data5[9]),           .TDA10(ipixel_data5[10]),
.TDA11(ipixel_data5[11]),  // Write Test Data
        .TDA12(ipixel_data5[12]),       .TDA13(ipixel_data5[13]),       .TDA14(ipixel_data5[14]),
.TDA15(ipixel_data5[15]),  // Write Test Data
        .TDA16(ipixel_data5[16]),       .TDA17(ipixel_data5[17]),       .TDA18(ipixel_data5[18]),
.TDA19(ipixel_data5[19]),  // Write Test Data
        .TDA20(ipixel_data5[20]),       .TDA21(ipixel_data5[21]),       .TDA22(ipixel_data5[22]),
.TDA23(ipixel_data5[23]),  // Write Test Data
        .TDA24(ipixel_data5[24]),       .TDA25(ipixel_data5[25]),       .TDA26(ipixel_data5[26]),
.TDA27(ipixel_data5[27]),  // Write Test Data
        .TDA28(ipixel_data5[28]),       .TDA29(ipixel_data5[29]),       .TDA30(ipixel_data5[30]),
.TDA31(ipixel_data5[31]),  // Write Test Data
        .TDA32(ipixel_data5[32]),       .TDA33(ipixel_data5[33]),       .TDA34(ipixel_data5[34]),
.TDA35(ipixel_data5[35]),  // Write Test Data
```

```
        .TDA36(ipixel_data5[36]),       .TDA37(ipixel_data5[37]),       .TDA38(ipixel_data5[38]),
.TDA39(ipixel_data5[39]),  // Write Test Data
        .TDA40(ipixel_data5[40]),       .TDA41(ipixel_data5[41]),       .TDA42(ipixel_data5[42]),
.TDA43(ipixel_data5[43]),  // Write Test Data
        .TDA44(ipixel_data5[44]),       .TDA45(ipixel_data5[45]),       .TDA46(ipixel_data5[46]),
.TDA47(ipixel_data5[47]),  // Write Test Data
        .TDA48(ipixel_data5[48]),       .TDA49(ipixel_data5[49]),       .TDA50(ipixel_data5[50]),
.TDA51(ipixel_data5[51]),  // Write Test Data
        .TDA52(ipixel_data5[52]),       .TDA53(ipixel_data5[53]),       .TDA54(ipixel_data5[54]),
.TDA55(ipixel_data5[55]),  // Write Test Data
        .TDA56(ipixel_data5[56]),       .TDA57(ipixel_data5[57]),       .TDA58(ipixel_data5[58]),
.TDA59(ipixel_data5[59]),  // Write Test Data
        .TDA60(ipixel_data5[60]),       .TDA61(ipixel_data5[61]),       .TDA62(ipixel_data5[62]),
.TDA63(ipixel_data5[63]),  // Write Test Data
        .TDA64(ipixel_data5[64]),       .TDA65(ipixel_data5[65]),       .TDA66(ipixel_data5[66]),
.TDA67(ipixel_data5[67]),  // Write Test Data
        .TDA68(ipixel_data5[68]),       .TDA69(ipixel_data5[69]),       .TDA70(ipixel_data5[70]),
.TDA71(ipixel_data5[71]),  // Write Test Data
        .TDA72(ipixel_data5[72]),       .TDA73(ipixel_data5[73]),       .TDA74(ipixel_data5[74]),
.TDA75(ipixel_data5[75]),  // Write Test Data
        .TDA76(ipixel_data5[76]),       .TDA77(ipixel_data5[77]),       .TDA78(ipixel_data5[78]),
.TDA79(ipixel_data5[79]),  // Write Test Data
        .TDA80(ipixel_data5[80]),       .TDA81(ipixel_data5[81]),       .TDA82(ipixel_data5[82]),
.TDA83(ipixel_data5[83]),  // Write Test Data
        .TDA84(ipixel_data5[84]),       .TDA85(ipixel_data5[85]),       .TDA86(ipixel_data5[86]),
.TDA87(ipixel_data5[87]),  // Write Test Data
        .TDA88(ipixel_data5[88]),       .TDA89(ipixel_data5[89]),       .TDA90(ipixel_data5[90]),
.TDA91(ipixel_data5[91]),  // Write Test Data
        .TDA92(ipixel_data5[92]),       .TDA93(ipixel_data5[93]),       .TDA94(ipixel_data5[94]),
.TDA95(ipixel_data5[95]),  // Write Test Data
        .TDA96(ipixel_data5[96]),       .TDA97(ipixel_data5[97]),       .TDA98(ipixel_data5[98]),
.TDA99(ipixel_data5[99]),  // Write Test Data
```

```
        .TDA100(ipixel_data5[100]),   .TDA101(ipixel_data5[101]),   .TDA102(ipixel_data5[102]),
.TDA103(ipixel_data5[103]),  // Write Test Data
        .TDA104(ipixel_data5[104]),   .TDA105(ipixel_data5[105]),   .TDA106(ipixel_data5[106]),
.TDA107(ipixel_data5[107]),  // Write Test Data
        .TDA108(ipixel_data5[108]),   .TDA109(ipixel_data5[109]),   .TDA110(ipixel_data5[110]),
.TDA111(ipixel_data5[111]),  // Write Test Data
        .TDA112(ipixel_data5[112]),   .TDA113(ipixel_data5[113]),   .TDA114(ipixel_data5[114]),
.TDA115(ipixel_data5[115]),  // Write Test Data
        .TDA116(ipixel_data5[116]),   .TDA117(ipixel_data5[117]),   .TDA118(ipixel_data5[118]),
.TDA119(ipixel_data5[119]),  // Write Test Data
        .TDA120(ipixel_data5[120]),   .TDA121(ipixel_data5[121]),   .TDA122(ipixel_data5[122]),
.TDA123(ipixel_data5[123]),  // Write Test Data
        .TDA124(ipixel_data5[124]),   .TDA125(ipixel_data5[125]),   .TDA126(ipixel_data5[126]),
.TDA127(ipixel_data5[127]),  // Write Test Data
        //READ TEST SIGNALS
        .BISTEB(vss),
        .TOEB(vss),
        .TMEB(vss),
        .TADRB0(q0_read_addr[0]),    .TADRB1(q0_read_addr[1]),   .TADRB2(q0_read_addr[2]),
.TADRB3(q0_read_addr[3]),  // Read Test Address
        .TADRB4(q0_read_addr[4]), .TADRB5(q0_read_addr[5]), .TADRB6(q0_read_addr[6]),  //
Read Test Address
        .AWTB(vss)
        );

`endif // !`ifdef USE_BEHAVE_MEM


`ifdef USE_BEHAVE_MEM
```

```
    dum_mem_p2 #(7,128) bank1_buff2(.iRCLK(sclk),
                                            .iWCLK(sclk),
                                            .iMER(q1_mem_re),
                                            .iMEW(imem_wen),
                                            .iWEN(imem_wew),
                                            .iRADR(q1_read_addr[6:0]),
                                            .iWADR(iwrite_addr),
                                            .iD(ipixel_data6),
                                            .oQ(buff6_out)
                                            );
`else // !`ifdef USE_BEHAVE_MEM
    rfsd2_80x128cm2sw0 ubank1_buff2
      (/*VRGIO   rfsd2_80x128cm2sw0   ipixel_data6   buff6_out   iwrite_addr   q1_read_addr
imem_wen q1_mem_re null*/
       // READ INTERFACE
       .CLKB(iSCLK), // Read Clock
       .OEB(vdd), // Output enable
       .MEB(q1_mem_re), // Read enable
       .ADRB0(q1_read_addr[0]),      .ADRB1(q1_read_addr[1]),      .ADRB2(q1_read_addr[2]),
.ADRB3(q1_read_addr[3]),  // Read Address
       .ADRB4(q1_read_addr[4]),   .ADRB5(q1_read_addr[5]),   .ADRB6(q1_read_addr[6]),     //
Read Address
       .QB0(buff6_out[0]), .QB1(buff6_out[1]), .QB2(buff6_out[2]), .QB3(buff6_out[3]),  // Read
Data
       .QB4(buff6_out[4]), .QB5(buff6_out[5]), .QB6(buff6_out[6]), .QB7(buff6_out[7]),  // Read
Data
       .QB8(buff6_out[8]), .QB9(buff6_out[9]), .QB10(buff6_out[10]), .QB11(buff6_out[11]),  //
Read Data
```

```
    .QB12(buff6_out[12]),     .QB13(buff6_out[13]),     .QB14(buff6_out[14]),
.QB15(buff6_out[15]),  // Read Data

    .QB16(buff6_out[16]),     .QB17(buff6_out[17]),     .QB18(buff6_out[18]),
.QB19(buff6_out[19]),  // Read Data

    .QB20(buff6_out[20]),     .QB21(buff6_out[21]),     .QB22(buff6_out[22]),
.QB23(buff6_out[23]),  // Read Data

    .QB24(buff6_out[24]),     .QB25(buff6_out[25]),     .QB26(buff6_out[26]),
.QB27(buff6_out[27]),  // Read Data

    .QB28(buff6_out[28]),     .QB29(buff6_out[29]),     .QB30(buff6_out[30]),
.QB31(buff6_out[31]),  // Read Data

    .QB32(buff6_out[32]),     .QB33(buff6_out[33]),     .QB34(buff6_out[34]),
.QB35(buff6_out[35]),  // Read Data

    .QB36(buff6_out[36]),     .QB37(buff6_out[37]),     .QB38(buff6_out[38]),
.QB39(buff6_out[39]),  // Read Data

    .QB40(buff6_out[40]),     .QB41(buff6_out[41]),     .QB42(buff6_out[42]),
.QB43(buff6_out[43]),  // Read Data

    .QB44(buff6_out[44]),     .QB45(buff6_out[45]),     .QB46(buff6_out[46]),
.QB47(buff6_out[47]),  // Read Data

    .QB48(buff6_out[48]),     .QB49(buff6_out[49]),     .QB50(buff6_out[50]),
.QB51(buff6_out[51]),  // Read Data

    .QB52(buff6_out[52]),     .QB53(buff6_out[53]),     .QB54(buff6_out[54]),
.QB55(buff6_out[55]),  // Read Data

    .QB56(buff6_out[56]),     .QB57(buff6_out[57]),     .QB58(buff6_out[58]),
.QB59(buff6_out[59]),  // Read Data

    .QB60(buff6_out[60]),     .QB61(buff6_out[61]),     .QB62(buff6_out[62]),
.QB63(buff6_out[63]),  // Read Data

    .QB64(buff6_out[64]),     .QB65(buff6_out[65]),     .QB66(buff6_out[66]),
.QB67(buff6_out[67]),  // Read Data

    .QB68(buff6_out[68]),     .QB69(buff6_out[69]),     .QB70(buff6_out[70]),
.QB71(buff6_out[71]),  // Read Data

    .QB72(buff6_out[72]),     .QB73(buff6_out[73]),     .QB74(buff6_out[74]),
.QB75(buff6_out[75]),  // Read Data
```

```
    .QB76(buff6_out[76]),     .QB77(buff6_out[77]),     .QB78(buff6_out[78]),
.QB79(buff6_out[79]),  // Read Data

    .QB80(buff6_out[80]),     .QB81(buff6_out[81]),     .QB82(buff6_out[82]),
.QB83(buff6_out[83]),  // Read Data

    .QB84(buff6_out[84]),     .QB85(buff6_out[85]),     .QB86(buff6_out[86]),
.QB87(buff6_out[87]),  // Read Data

    .QB88(buff6_out[88]),     .QB89(buff6_out[89]),     .QB90(buff6_out[90]),
.QB91(buff6_out[91]),  // Read Data

    .QB92(buff6_out[92]),     .QB93(buff6_out[93]),     .QB94(buff6_out[94]),
.QB95(buff6_out[95]),  // Read Data

    .QB96(buff6_out[96]),     .QB97(buff6_out[97]),     .QB98(buff6_out[98]),
.QB99(buff6_out[99]),  // Read Data

    .QB100(buff6_out[100]),   .QB101(buff6_out[101]),   .QB102(buff6_out[102]),
.QB103(buff6_out[103]),  // Read Data

    .QB104(buff6_out[104]),   .QB105(buff6_out[105]),   .QB106(buff6_out[106]),
.QB107(buff6_out[107]),  // Read Data

    .QB108(buff6_out[108]),   .QB109(buff6_out[109]),   .QB110(buff6_out[110]),
.QB111(buff6_out[111]),  // Read Data

    .QB112(buff6_out[112]),   .QB113(buff6_out[113]),   .QB114(buff6_out[114]),
.QB115(buff6_out[115]),  // Read Data

    .QB116(buff6_out[116]),   .QB117(buff6_out[117]),   .QB118(buff6_out[118]),
.QB119(buff6_out[119]),  // Read Data

    .QB120(buff6_out[120]),   .QB121(buff6_out[121]),   .QB122(buff6_out[122]),
.QB123(buff6_out[123]),  // Read Data

    .QB124(buff6_out[124]),   .QB125(buff6_out[125]),   .QB126(buff6_out[126]),
.QB127(buff6_out[127]),  // Read Data

    // WRITE INTERFACE

    .CLKA(iSCLK), // Write Clock

    .WEA(imem_wen), // Write enable

    .MEA(vdd), // Memory enable

    .ADRA0(iwrite_addr[0]),   .ADRA1(iwrite_addr[1]),   .ADRA2(iwrite_addr[2]),
.ADRA3(iwrite_addr[3]),  // Write Address
```

```
    .ADRA4(iwrite_addr[4]),  .ADRA5(iwrite_addr[5]),  .ADRA6(iwrite_addr[6]),   //  Write
Address

    .DA0(ipixel_data6[0]),      .DA1(ipixel_data6[1]),      .DA2(ipixel_data6[2]),
.DA3(ipixel_data6[3]),  // Write Data

    .DA4(ipixel_data6[4]),      .DA5(ipixel_data6[5]),      .DA6(ipixel_data6[6]),
.DA7(ipixel_data6[7]),  // Write Data

    .DA8(ipixel_data6[8]),      .DA9(ipixel_data6[9]),      .DA10(ipixel_data6[10]),
.DA11(ipixel_data6[11]),  // Write Data

    .DA12(ipixel_data6[12]),    .DA13(ipixel_data6[13]),    .DA14(ipixel_data6[14]),
.DA15(ipixel_data6[15]),  // Write Data

    .DA16(ipixel_data6[16]),    .DA17(ipixel_data6[17]),    .DA18(ipixel_data6[18]),
.DA19(ipixel_data6[19]),  // Write Data

    .DA20(ipixel_data6[20]),    .DA21(ipixel_data6[21]),    .DA22(ipixel_data6[22]),
.DA23(ipixel_data6[23]),  // Write Data

    .DA24(ipixel_data6[24]),    .DA25(ipixel_data6[25]),    .DA26(ipixel_data6[26]),
.DA27(ipixel_data6[27]),  // Write Data

    .DA28(ipixel_data6[28]),    .DA29(ipixel_data6[29]),    .DA30(ipixel_data6[30]),
.DA31(ipixel_data6[31]),  // Write Data

    .DA32(ipixel_data6[32]),    .DA33(ipixel_data6[33]),    .DA34(ipixel_data6[34]),
.DA35(ipixel_data6[35]),  // Write Data

    .DA36(ipixel_data6[36]),    .DA37(ipixel_data6[37]),    .DA38(ipixel_data6[38]),
.DA39(ipixel_data6[39]),  // Write Data

    .DA40(ipixel_data6[40]),    .DA41(ipixel_data6[41]),    .DA42(ipixel_data6[42]),
.DA43(ipixel_data6[43]),  // Write Data

    .DA44(ipixel_data6[44]),    .DA45(ipixel_data6[45]),    .DA46(ipixel_data6[46]),
.DA47(ipixel_data6[47]),  // Write Data

    .DA48(ipixel_data6[48]),    .DA49(ipixel_data6[49]),    .DA50(ipixel_data6[50]),
.DA51(ipixel_data6[51]),  // Write Data

    .DA52(ipixel_data6[52]),    .DA53(ipixel_data6[53]),    .DA54(ipixel_data6[54]),
.DA55(ipixel_data6[55]),  // Write Data

    .DA56(ipixel_data6[56]),    .DA57(ipixel_data6[57]),    .DA58(ipixel_data6[58]),
.DA59(ipixel_data6[59]),  // Write Data
```

```
    .DA60(ipixel_data6[60]),    .DA61(ipixel_data6[61]),    .DA62(ipixel_data6[62]),
.DA63(ipixel_data6[63]),  // Write Data

    .DA64(ipixel_data6[64]),    .DA65(ipixel_data6[65]),    .DA66(ipixel_data6[66]),
.DA67(ipixel_data6[67]),  // Write Data

    .DA68(ipixel_data6[68]),    .DA69(ipixel_data6[69]),    .DA70(ipixel_data6[70]),
.DA71(ipixel_data6[71]),  // Write Data

    .DA72(ipixel_data6[72]),    .DA73(ipixel_data6[73]),    .DA74(ipixel_data6[74]),
.DA75(ipixel_data6[75]),  // Write Data

    .DA76(ipixel_data6[76]),    .DA77(ipixel_data6[77]),    .DA78(ipixel_data6[78]),
.DA79(ipixel_data6[79]),  // Write Data

    .DA80(ipixel_data6[80]),    .DA81(ipixel_data6[81]),    .DA82(ipixel_data6[82]),
.DA83(ipixel_data6[83]),  // Write Data

    .DA84(ipixel_data6[84]),    .DA85(ipixel_data6[85]),    .DA86(ipixel_data6[86]),
.DA87(ipixel_data6[87]),  // Write Data

    .DA88(ipixel_data6[88]),    .DA89(ipixel_data6[89]),    .DA90(ipixel_data6[90]),
.DA91(ipixel_data6[91]),  // Write Data

    .DA92(ipixel_data6[92]),    .DA93(ipixel_data6[93]),    .DA94(ipixel_data6[94]),
.DA95(ipixel_data6[95]),  // Write Data

    .DA96(ipixel_data6[96]),    .DA97(ipixel_data6[97]),    .DA98(ipixel_data6[98]),
.DA99(ipixel_data6[99]),  // Write Data

    .DA100(ipixel_data6[100]),  .DA101(ipixel_data6[101]),  .DA102(ipixel_data6[102]),
.DA103(ipixel_data6[103]),  // Write Data

    .DA104(ipixel_data6[104]),  .DA105(ipixel_data6[105]),  .DA106(ipixel_data6[106]),
.DA107(ipixel_data6[107]),  // Write Data

    .DA108(ipixel_data6[108]),  .DA109(ipixel_data6[109]),  .DA110(ipixel_data6[110]),
.DA111(ipixel_data6[111]),  // Write Data

    .DA112(ipixel_data6[112]),  .DA113(ipixel_data6[113]),  .DA114(ipixel_data6[114]),
.DA115(ipixel_data6[115]),  // Write Data

    .DA116(ipixel_data6[116]),  .DA117(ipixel_data6[117]),  .DA118(ipixel_data6[118]),
.DA119(ipixel_data6[119]),  // Write Data

    .DA120(ipixel_data6[120]),  .DA121(ipixel_data6[121]),  .DA122(ipixel_data6[122]),
.DA123(ipixel_data6[123]),  // Write Data
```

```
    .DA124(ipixel_data6[124]),   .DA125(ipixel_data6[125]),   .DA126(ipixel_data6[126]),
.DA127(ipixel_data6[127]),  // Write Data

    // WRITE TEST SIGNALS

    .BISTEA(vss),

    .TWEA(vss), // Test write enable

    .TMEA(vss), // Test memory enable

    .TADRA0(iwrite_addr[0]),   .TADRA1(iwrite_addr[1]),   .TADRA2(iwrite_addr[2]),
.TADRA3(iwrite_addr[3]),  // Write Test Address

    .TADRA4(iwrite_addr[4]), .TADRA5(iwrite_addr[5]), .TADRA6(iwrite_addr[6]),  // Write
Test Address

    .TDA0(ipixel_data6[0]),      .TDA1(ipixel_data6[1]),      .TDA2(ipixel_data6[2]),
.TDA3(ipixel_data6[3]),  // Write Test Data

    .TDA4(ipixel_data6[4]),      .TDA5(ipixel_data6[5]),      .TDA6(ipixel_data6[6]),
.TDA7(ipixel_data6[7]),  // Write Test Data

    .TDA8(ipixel_data6[8]),     .TDA9(ipixel_data6[9]),     .TDA10(ipixel_data6[10]),
.TDA11(ipixel_data6[11]),  // Write Test Data

    .TDA12(ipixel_data6[12]),   .TDA13(ipixel_data6[13]),   .TDA14(ipixel_data6[14]),
.TDA15(ipixel_data6[15]),  // Write Test Data

    .TDA16(ipixel_data6[16]),   .TDA17(ipixel_data6[17]),   .TDA18(ipixel_data6[18]),
.TDA19(ipixel_data6[19]),  // Write Test Data

    .TDA20(ipixel_data6[20]),   .TDA21(ipixel_data6[21]),   .TDA22(ipixel_data6[22]),
.TDA23(ipixel_data6[23]),  // Write Test Data

    .TDA24(ipixel_data6[24]),   .TDA25(ipixel_data6[25]),   .TDA26(ipixel_data6[26]),
.TDA27(ipixel_data6[27]),  // Write Test Data

    .TDA28(ipixel_data6[28]),   .TDA29(ipixel_data6[29]),   .TDA30(ipixel_data6[30]),
.TDA31(ipixel_data6[31]),  // Write Test Data

    .TDA32(ipixel_data6[32]),   .TDA33(ipixel_data6[33]),   .TDA34(ipixel_data6[34]),
.TDA35(ipixel_data6[35]),  // Write Test Data

    .TDA36(ipixel_data6[36]),   .TDA37(ipixel_data6[37]),   .TDA38(ipixel_data6[38]),
.TDA39(ipixel_data6[39]),  // Write Test Data

    .TDA40(ipixel_data6[40]),   .TDA41(ipixel_data6[41]),   .TDA42(ipixel_data6[42]),
.TDA43(ipixel_data6[43]),  // Write Test Data
```

```
    .TDA44(ipixel_data6[44]),   .TDA45(ipixel_data6[45]),   .TDA46(ipixel_data6[46]),
.TDA47(ipixel_data6[47]),  // Write Test Data

    .TDA48(ipixel_data6[48]),   .TDA49(ipixel_data6[49]),   .TDA50(ipixel_data6[50]),
.TDA51(ipixel_data6[51]),  // Write Test Data

    .TDA52(ipixel_data6[52]),   .TDA53(ipixel_data6[53]),   .TDA54(ipixel_data6[54]),
.TDA55(ipixel_data6[55]),  // Write Test Data

    .TDA56(ipixel_data6[56]),   .TDA57(ipixel_data6[57]),   .TDA58(ipixel_data6[58]),
.TDA59(ipixel_data6[59]),  // Write Test Data

    .TDA60(ipixel_data6[60]),   .TDA61(ipixel_data6[61]),   .TDA62(ipixel_data6[62]),
.TDA63(ipixel_data6[63]),  // Write Test Data

    .TDA64(ipixel_data6[64]),   .TDA65(ipixel_data6[65]),   .TDA66(ipixel_data6[66]),
.TDA67(ipixel_data6[67]),  // Write Test Data

    .TDA68(ipixel_data6[68]),   .TDA69(ipixel_data6[69]),   .TDA70(ipixel_data6[70]),
.TDA71(ipixel_data6[71]),  // Write Test Data

    .TDA72(ipixel_data6[72]),   .TDA73(ipixel_data6[73]),   .TDA74(ipixel_data6[74]),
.TDA75(ipixel_data6[75]),  // Write Test Data

    .TDA76(ipixel_data6[76]),   .TDA77(ipixel_data6[77]),   .TDA78(ipixel_data6[78]),
.TDA79(ipixel_data6[79]),  // Write Test Data

    .TDA80(ipixel_data6[80]),   .TDA81(ipixel_data6[81]),   .TDA82(ipixel_data6[82]),
.TDA83(ipixel_data6[83]),  // Write Test Data

    .TDA84(ipixel_data6[84]),   .TDA85(ipixel_data6[85]),   .TDA86(ipixel_data6[86]),
.TDA87(ipixel_data6[87]),  // Write Test Data

    .TDA88(ipixel_data6[88]),   .TDA89(ipixel_data6[89]),   .TDA90(ipixel_data6[90]),
.TDA91(ipixel_data6[91]),  // Write Test Data

    .TDA92(ipixel_data6[92]),   .TDA93(ipixel_data6[93]),   .TDA94(ipixel_data6[94]),
.TDA95(ipixel_data6[95]),  // Write Test Data

    .TDA96(ipixel_data6[96]),   .TDA97(ipixel_data6[97]),   .TDA98(ipixel_data6[98]),
.TDA99(ipixel_data6[99]),  // Write Test Data

    .TDA100(ipixel_data6[100]),   .TDA101(ipixel_data6[101]),   .TDA102(ipixel_data6[102]),
.TDA103(ipixel_data6[103]),  // Write Test Data

    .TDA104(ipixel_data6[104]),   .TDA105(ipixel_data6[105]),   .TDA106(ipixel_data6[106]),
.TDA107(ipixel_data6[107]),  // Write Test Data
```

```
    .TDA108(ipixel_data6[108]),   .TDA109(ipixel_data6[109]),   .TDA110(ipixel_data6[110]),
.TDA111(ipixel_data6[111]),  // Write Test Data

    .TDA112(ipixel_data6[112]),   .TDA113(ipixel_data6[113]),   .TDA114(ipixel_data6[114]),
.TDA115(ipixel_data6[115]),  // Write Test Data

    .TDA116(ipixel_data6[116]),   .TDA117(ipixel_data6[117]),   .TDA118(ipixel_data6[118]),
.TDA119(ipixel_data6[119]),  // Write Test Data

    .TDA120(ipixel_data6[120]),   .TDA121(ipixel_data6[121]),   .TDA122(ipixel_data6[122]),
.TDA123(ipixel_data6[123]),  // Write Test Data

    .TDA124(ipixel_data6[124]),   .TDA125(ipixel_data6[125]),   .TDA126(ipixel_data6[126]),
.TDA127(ipixel_data6[127]),  // Write Test Data

    //READ TEST SIGNALS

    .BISTEB(vss),

    .TOEB(vss),

    .TMEB(vss),

    .TADRB0(q1_read_addr[0]),   .TADRB1(q1_read_addr[1]),   .TADRB2(q1_read_addr[2]),
.TADRB3(q1_read_addr[3]),  // Read Test Address

    .TADRB4(q1_read_addr[4]), .TADRB5(q1_read_addr[5]), .TADRB6(q1_read_addr[6]),  //
Read Test Address

    .AWTB(vss)

    );


`endif // !`ifdef USE_BEHAVE_MEM


`ifdef USE_BEHAVE_MEM

  dum_mem_p2 #(7,128) bank1_buff3(.iRCLK(sclk),

                                  .iWCLK(sclk),

                                  .iMER(q2_mem_re),

                                  .iMEW(imem_wen),
```

```
                                  .iWEN(imem_wew),

                                  .iRADR(q2_read_addr[6:0]),

                                  .iWADR(iwrite_addr),

                                  .iD(ipixel_data7),

                                  .oQ(buff7_out)

                                  );

`else // !`ifdef USE_BEHAVE_MEM

  rfsd2_80x128cm2sw0 ubank1_buff3

   (/*VRGIO   rfsd2_80x128cm2sw0   ipixel_data7   buff7_out   iwrite_addr   q2_read_addr
imem_wen q2_mem_re null*/

    // READ INTERFACE

    .CLKB(iSCLK), // Read Clock

    .OEB(vdd), // Output enable

    .MEB(q2_mem_re), // Read enable

    .ADRB0(q2_read_addr[0]),   .ADRB1(q2_read_addr[1]),   .ADRB2(q2_read_addr[2]),
.ADRB3(q2_read_addr[3]),  // Read Address

    .ADRB4(q2_read_addr[4]),   .ADRB5(q2_read_addr[5]),   .ADRB6(q2_read_addr[6]),   //
Read Address

    .QB0(buff7_out[0]), .QB1(buff7_out[1]), .QB2(buff7_out[2]), .QB3(buff7_out[3]),  // Read
Data

    .QB4(buff7_out[4]), .QB5(buff7_out[5]), .QB6(buff7_out[6]), .QB7(buff7_out[7]),  // Read
Data

    .QB8(buff7_out[8]), .QB9(buff7_out[9]), .QB10(buff7_out[10]), .QB11(buff7_out[11]),  //
Read Data

    .QB12(buff7_out[12]),      .QB13(buff7_out[13]),      .QB14(buff7_out[14]),
.QB15(buff7_out[15]),  // Read Data

    .QB16(buff7_out[16]),      .QB17(buff7_out[17]),      .QB18(buff7_out[18]),
.QB19(buff7_out[19]),  // Read Data
```

```
    .QB20(buff7_out[20]), .QB21(buff7_out[21]), .QB22(buff7_out[22]), .QB23(buff7_out[23]),  // Read Data
    .QB24(buff7_out[24]), .QB25(buff7_out[25]), .QB26(buff7_out[26]), .QB27(buff7_out[27]),  // Read Data
    .QB28(buff7_out[28]), .QB29(buff7_out[29]), .QB30(buff7_out[30]), .QB31(buff7_out[31]),  // Read Data
    .QB32(buff7_out[32]), .QB33(buff7_out[33]), .QB34(buff7_out[34]), .QB35(buff7_out[35]),  // Read Data
    .QB36(buff7_out[36]), .QB37(buff7_out[37]), .QB38(buff7_out[38]), .QB39(buff7_out[39]),  // Read Data
    .QB40(buff7_out[40]), .QB41(buff7_out[41]), .QB42(buff7_out[42]), .QB43(buff7_out[43]),  // Read Data
    .QB44(buff7_out[44]), .QB45(buff7_out[45]), .QB46(buff7_out[46]), .QB47(buff7_out[47]),  // Read Data
    .QB48(buff7_out[48]), .QB49(buff7_out[49]), .QB50(buff7_out[50]), .QB51(buff7_out[51]),  // Read Data
    .QB52(buff7_out[52]), .QB53(buff7_out[53]), .QB54(buff7_out[54]), .QB55(buff7_out[55]),  // Read Data
    .QB56(buff7_out[56]), .QB57(buff7_out[57]), .QB58(buff7_out[58]), .QB59(buff7_out[59]),  // Read Data
    .QB60(buff7_out[60]), .QB61(buff7_out[61]), .QB62(buff7_out[62]), .QB63(buff7_out[63]),  // Read Data
    .QB64(buff7_out[64]), .QB65(buff7_out[65]), .QB66(buff7_out[66]), .QB67(buff7_out[67]),  // Read Data
    .QB68(buff7_out[68]), .QB69(buff7_out[69]), .QB70(buff7_out[70]), .QB71(buff7_out[71]),  // Read Data
    .QB72(buff7_out[72]), .QB73(buff7_out[73]), .QB74(buff7_out[74]), .QB75(buff7_out[75]),  // Read Data
    .QB76(buff7_out[76]), .QB77(buff7_out[77]), .QB78(buff7_out[78]), .QB79(buff7_out[79]),  // Read Data
    .QB80(buff7_out[80]), .QB81(buff7_out[81]), .QB82(buff7_out[82]), .QB83(buff7_out[83]),  // Read Data
```

```
    .QB84(buff7_out[84]), .QB85(buff7_out[85]), .QB86(buff7_out[86]), .QB87(buff7_out[87]),  // Read Data
    .QB88(buff7_out[88]), .QB89(buff7_out[89]), .QB90(buff7_out[90]), .QB91(buff7_out[91]),  // Read Data
    .QB92(buff7_out[92]), .QB93(buff7_out[93]), .QB94(buff7_out[94]), .QB95(buff7_out[95]),  // Read Data
    .QB96(buff7_out[96]), .QB97(buff7_out[97]), .QB98(buff7_out[98]), .QB99(buff7_out[99]),  // Read Data
    .QB100(buff7_out[100]), .QB101(buff7_out[101]), .QB102(buff7_out[102]), .QB103(buff7_out[103]),  // Read Data
    .QB104(buff7_out[104]), .QB105(buff7_out[105]), .QB106(buff7_out[106]), .QB107(buff7_out[107]),  // Read Data
    .QB108(buff7_out[108]), .QB109(buff7_out[109]), .QB110(buff7_out[110]), .QB111(buff7_out[111]),  // Read Data
    .QB112(buff7_out[112]), .QB113(buff7_out[113]), .QB114(buff7_out[114]), .QB115(buff7_out[115]),  // Read Data
    .QB116(buff7_out[116]), .QB117(buff7_out[117]), .QB118(buff7_out[118]), .QB119(buff7_out[119]),  // Read Data
    .QB120(buff7_out[120]), .QB121(buff7_out[121]), .QB122(buff7_out[122]), .QB123(buff7_out[123]),  // Read Data
    .QB124(buff7_out[124]), .QB125(buff7_out[125]), .QB126(buff7_out[126]), .QB127(buff7_out[127]),  // Read Data
    // WRITE INTERFACE
    .CLKA(iSCLK), // Write Clock
    .WEA(imem_wen), // Write enable
    .MEA(vdd), // Memory enable
    .ADRA0(iwrite_addr[0]), .ADRA1(iwrite_addr[1]), .ADRA2(iwrite_addr[2]), .ADRA3(iwrite_addr[3]),  // Write Address
    .ADRA4(iwrite_addr[4]), .ADRA5(iwrite_addr[5]), .ADRA6(iwrite_addr[6]),  // Write Address
    .DA0(ipixel_data7[0]), .DA1(ipixel_data7[1]), .DA2(ipixel_data7[2]), .DA3(ipixel_data7[3]),  // Write Data
```

```
    .DA4(ipixel_data7[4]), .DA5(ipixel_data7[5]), .DA6(ipixel_data7[6]), .DA7(ipixel_data7[7]),  // Write Data
    .DA8(ipixel_data7[8]), .DA9(ipixel_data7[9]), .DA10(ipixel_data7[10]), .DA11(ipixel_data7[11]),  // Write Data
    .DA12(ipixel_data7[12]), .DA13(ipixel_data7[13]), .DA14(ipixel_data7[14]), .DA15(ipixel_data7[15]),  // Write Data
    .DA16(ipixel_data7[16]), .DA17(ipixel_data7[17]), .DA18(ipixel_data7[18]), .DA19(ipixel_data7[19]),  // Write Data
    .DA20(ipixel_data7[20]), .DA21(ipixel_data7[21]), .DA22(ipixel_data7[22]), .DA23(ipixel_data7[23]),  // Write Data
    .DA24(ipixel_data7[24]), .DA25(ipixel_data7[25]), .DA26(ipixel_data7[26]), .DA27(ipixel_data7[27]),  // Write Data
    .DA28(ipixel_data7[28]), .DA29(ipixel_data7[29]), .DA30(ipixel_data7[30]), .DA31(ipixel_data7[31]),  // Write Data
    .DA32(ipixel_data7[32]), .DA33(ipixel_data7[33]), .DA34(ipixel_data7[34]), .DA35(ipixel_data7[35]),  // Write Data
    .DA36(ipixel_data7[36]), .DA37(ipixel_data7[37]), .DA38(ipixel_data7[38]), .DA39(ipixel_data7[39]),  // Write Data
    .DA40(ipixel_data7[40]), .DA41(ipixel_data7[41]), .DA42(ipixel_data7[42]), .DA43(ipixel_data7[43]),  // Write Data
    .DA44(ipixel_data7[44]), .DA45(ipixel_data7[45]), .DA46(ipixel_data7[46]), .DA47(ipixel_data7[47]),  // Write Data
    .DA48(ipixel_data7[48]), .DA49(ipixel_data7[49]), .DA50(ipixel_data7[50]), .DA51(ipixel_data7[51]),  // Write Data
    .DA52(ipixel_data7[52]), .DA53(ipixel_data7[53]), .DA54(ipixel_data7[54]), .DA55(ipixel_data7[55]),  // Write Data
    .DA56(ipixel_data7[56]), .DA57(ipixel_data7[57]), .DA58(ipixel_data7[58]), .DA59(ipixel_data7[59]),  // Write Data
    .DA60(ipixel_data7[60]), .DA61(ipixel_data7[61]), .DA62(ipixel_data7[62]), .DA63(ipixel_data7[63]),  // Write Data
    .DA64(ipixel_data7[64]), .DA65(ipixel_data7[65]), .DA66(ipixel_data7[66]), .DA67(ipixel_data7[67]),  // Write Data
```

```
    .DA68(ipixel_data7[68]), .DA69(ipixel_data7[69]), .DA70(ipixel_data7[70]), .DA71(ipixel_data7[71]),  // Write Data
    .DA72(ipixel_data7[72]), .DA73(ipixel_data7[73]), .DA74(ipixel_data7[74]), .DA75(ipixel_data7[75]),  // Write Data
    .DA76(ipixel_data7[76]), .DA77(ipixel_data7[77]), .DA78(ipixel_data7[78]), .DA79(ipixel_data7[79]),  // Write Data
    .DA80(ipixel_data7[80]), .DA81(ipixel_data7[81]), .DA82(ipixel_data7[82]), .DA83(ipixel_data7[83]),  // Write Data
    .DA84(ipixel_data7[84]), .DA85(ipixel_data7[85]), .DA86(ipixel_data7[86]), .DA87(ipixel_data7[87]),  // Write Data
    .DA88(ipixel_data7[88]), .DA89(ipixel_data7[89]), .DA90(ipixel_data7[90]), .DA91(ipixel_data7[91]),  // Write Data
    .DA92(ipixel_data7[92]), .DA93(ipixel_data7[93]), .DA94(ipixel_data7[94]), .DA95(ipixel_data7[95]),  // Write Data
    .DA96(ipixel_data7[96]), .DA97(ipixel_data7[97]), .DA98(ipixel_data7[98]), .DA99(ipixel_data7[99]),  // Write Data
    .DA100(ipixel_data7[100]), .DA101(ipixel_data7[101]), .DA102(ipixel_data7[102]), .DA103(ipixel_data7[103]),  // Write Data
    .DA104(ipixel_data7[104]), .DA105(ipixel_data7[105]), .DA106(ipixel_data7[106]), .DA107(ipixel_data7[107]),  // Write Data
    .DA108(ipixel_data7[108]), .DA109(ipixel_data7[109]), .DA110(ipixel_data7[110]), .DA111(ipixel_data7[111]),  // Write Data
    .DA112(ipixel_data7[112]), .DA113(ipixel_data7[113]), .DA114(ipixel_data7[114]), .DA115(ipixel_data7[115]),  // Write Data
    .DA116(ipixel_data7[116]), .DA117(ipixel_data7[117]), .DA118(ipixel_data7[118]), .DA119(ipixel_data7[119]),  // Write Data
    .DA120(ipixel_data7[120]), .DA121(ipixel_data7[121]), .DA122(ipixel_data7[122]), .DA123(ipixel_data7[123]),  // Write Data
    .DA124(ipixel_data7[124]), .DA125(ipixel_data7[125]), .DA126(ipixel_data7[126]), .DA127(ipixel_data7[127]),  // Write Data
    // WRITE TEST SIGNALS
    .BISTEA(vss),
    .TWEA(vss), // Test write enable
    .TMEA(vss), // Test memory enable
    .TADRA0(iwrite_addr[0]),     .TADRA1(iwrite_addr[1]),     .TADRA2(iwrite_addr[2]), .TADRA3(iwrite_addr[3]),  // Write Test Address
    .TADRA4(iwrite_addr[4]), .TADRA5(iwrite_addr[5]), .TADRA6(iwrite_addr[6]),  // Write Test Address
    .TDA0(ipixel_data7[0]),          .TDA1(ipixel_data7[1]),          .TDA2(ipixel_data7[2]), .TDA3(ipixel_data7[3]),  // Write Test Data
    .TDA4(ipixel_data7[4]),          .TDA5(ipixel_data7[5]),          .TDA6(ipixel_data7[6]), .TDA7(ipixel_data7[7]),  // Write Test Data
    .TDA8(ipixel_data7[8]),         .TDA9(ipixel_data7[9]),         .TDA10(ipixel_data7[10]), .TDA11(ipixel_data7[11]),  // Write Test Data
    .TDA12(ipixel_data7[12]),     .TDA13(ipixel_data7[13]),     .TDA14(ipixel_data7[14]), .TDA15(ipixel_data7[15]),  // Write Test Data
    .TDA16(ipixel_data7[16]),     .TDA17(ipixel_data7[17]),     .TDA18(ipixel_data7[18]), .TDA19(ipixel_data7[19]),  // Write Test Data
    .TDA20(ipixel_data7[20]),     .TDA21(ipixel_data7[21]),     .TDA22(ipixel_data7[22]), .TDA23(ipixel_data7[23]),  // Write Test Data
    .TDA24(ipixel_data7[24]),     .TDA25(ipixel_data7[25]),     .TDA26(ipixel_data7[26]), .TDA27(ipixel_data7[27]),  // Write Test Data
    .TDA28(ipixel_data7[28]),     .TDA29(ipixel_data7[29]),     .TDA30(ipixel_data7[30]), .TDA31(ipixel_data7[31]),  // Write Test Data
    .TDA32(ipixel_data7[32]),     .TDA33(ipixel_data7[33]),     .TDA34(ipixel_data7[34]), .TDA35(ipixel_data7[35]),  // Write Test Data
    .TDA36(ipixel_data7[36]),     .TDA37(ipixel_data7[37]),     .TDA38(ipixel_data7[38]), .TDA39(ipixel_data7[39]),  // Write Test Data
    .TDA40(ipixel_data7[40]),     .TDA41(ipixel_data7[41]),     .TDA42(ipixel_data7[42]), .TDA43(ipixel_data7[43]),  // Write Test Data
    .TDA44(ipixel_data7[44]),     .TDA45(ipixel_data7[45]),     .TDA46(ipixel_data7[46]), .TDA47(ipixel_data7[47]),  // Write Test Data
    .TDA48(ipixel_data7[48]),     .TDA49(ipixel_data7[49]),     .TDA50(ipixel_data7[50]), .TDA51(ipixel_data7[51]),  // Write Test Data
```

```
    .TDA52(ipixel_data7[52]),     .TDA53(ipixel_data7[53]),     .TDA54(ipixel_data7[54]), .TDA55(ipixel_data7[55]),  // Write Test Data
    .TDA56(ipixel_data7[56]),     .TDA57(ipixel_data7[57]),     .TDA58(ipixel_data7[58]), .TDA59(ipixel_data7[59]),  // Write Test Data
    .TDA60(ipixel_data7[60]),     .TDA61(ipixel_data7[61]),     .TDA62(ipixel_data7[62]), .TDA63(ipixel_data7[63]),  // Write Test Data
    .TDA64(ipixel_data7[64]),     .TDA65(ipixel_data7[65]),     .TDA66(ipixel_data7[66]), .TDA67(ipixel_data7[67]),  // Write Test Data
    .TDA68(ipixel_data7[68]),     .TDA69(ipixel_data7[69]),     .TDA70(ipixel_data7[70]), .TDA71(ipixel_data7[71]),  // Write Test Data
    .TDA72(ipixel_data7[72]),     .TDA73(ipixel_data7[73]),     .TDA74(ipixel_data7[74]), .TDA75(ipixel_data7[75]),  // Write Test Data
    .TDA76(ipixel_data7[76]),     .TDA77(ipixel_data7[77]),     .TDA78(ipixel_data7[78]), .TDA79(ipixel_data7[79]),  // Write Test Data
    .TDA80(ipixel_data7[80]),     .TDA81(ipixel_data7[81]),     .TDA82(ipixel_data7[82]), .TDA83(ipixel_data7[83]),  // Write Test Data
    .TDA84(ipixel_data7[84]),     .TDA85(ipixel_data7[85]),     .TDA86(ipixel_data7[86]), .TDA87(ipixel_data7[87]),  // Write Test Data
    .TDA88(ipixel_data7[88]),     .TDA89(ipixel_data7[89]),     .TDA90(ipixel_data7[90]), .TDA91(ipixel_data7[91]),  // Write Test Data
    .TDA92(ipixel_data7[92]),     .TDA93(ipixel_data7[93]),     .TDA94(ipixel_data7[94]), .TDA95(ipixel_data7[95]),  // Write Test Data
    .TDA96(ipixel_data7[96]),     .TDA97(ipixel_data7[97]),     .TDA98(ipixel_data7[98]), .TDA99(ipixel_data7[99]),  // Write Test Data
    .TDA100(ipixel_data7[100]),   .TDA101(ipixel_data7[101]),   .TDA102(ipixel_data7[102]), .TDA103(ipixel_data7[103]),  // Write Test Data
    .TDA104(ipixel_data7[104]),   .TDA105(ipixel_data7[105]),   .TDA106(ipixel_data7[106]), .TDA107(ipixel_data7[107]),  // Write Test Data
    .TDA108(ipixel_data7[108]),   .TDA109(ipixel_data7[109]),   .TDA110(ipixel_data7[110]), .TDA111(ipixel_data7[111]),  // Write Test Data
    .TDA112(ipixel_data7[112]),   .TDA113(ipixel_data7[113]),   .TDA114(ipixel_data7[114]), .TDA115(ipixel_data7[115]),  // Write Test Data
```

```
    .TDA116(ipixel_data7[116]),   .TDA117(ipixel_data7[117]),   .TDA118(ipixel_data7[118]), .TDA119(ipixel_data7[119]),  // Write Test Data
    .TDA120(ipixel_data7[120]),   .TDA121(ipixel_data7[121]),   .TDA122(ipixel_data7[122]), .TDA123(ipixel_data7[123]),  // Write Test Data
    .TDA124(ipixel_data7[124]),   .TDA125(ipixel_data7[125]),   .TDA126(ipixel_data7[126]), .TDA127(ipixel_data7[127]),  // Write Test Data
    //READ TEST SIGNALS
    .BISTEB(vss),
    .TOEB(vss),
    .TMEB(vss),
    .TADRB0(q2_read_addr[0]),   .TADRB1(q2_read_addr[1]),   .TADRB2(q2_read_addr[2]), .TADRB3(q2_read_addr[3]),  // Read Test Address
    .TADRB4(q2_read_addr[4]), .TADRB5(q2_read_addr[5]), .TADRB6(q2_read_addr[6]),  // Read Test Address
    .AWTB(vss)
    );

`endif // !`ifdef USE_BEHAVE_MEM

  //-------------------------------------------------------//

  always @(posedge sclk)
    begin
        if(srst)
          begin
            q_sp_bank_sel <= 1'b0;
          end
```

```
        else
          begin
            q_sp_bank_sel <= sp_bank_sel;
            q_sp_bank_sel0 <= sp_bank_sel0;
            q_sp_bank_sel1 <= sp_bank_sel1;
            q_sp_bank_sel2 <= sp_bank_sel2;
            q_sp_bank_sel3 <= sp_bank_sel3;
          end // else: !if()
    end // always @ (posedge sclk)





   //buffer 0  4x1 mux
   always @(/*AUTOSENSE*/buff0_out or buff1_out or buff2_out
              or buff3_out or q_phase_rb0)
     begin
        case(q_phase_rb0)
          2'b00:bank0_data0 = buff0_out;
          2'b01:bank0_data0 = buff1_out;
          2'b10:bank0_data0 = buff2_out;
          2'b11:bank0_data0 = buff3_out;
          default:bank0_data0 = buff0_out;
        endcase // case(q_phase_rb0)
```

```
  end // always @ (...

 //buffer 1  4x1 mux
 always @(/*AUTOSENSE*/buff0_out or buff1_out or buff2_out
         or buff3_out or q_phase_rb1)
  begin
      case(q_phase_rb1)
        2'b00:bank0_data1 = buff0_out;
        2'b01:bank0_data1 = buff1_out;
        2'b10:bank0_data1 = buff2_out;
        2'b11:bank0_data1 = buff3_out;
        default:bank0_data1 = buff0_out;
      endcase // case(q_phase_rb1)
  end // always @ (...


  //buffer 2  4x1 mux
 always @(/*AUTOSENSE*/buff0_out or buff1_out or buff2_out
         or buff3_out or q_phase_rb2)
  begin
      case(q_phase_rb2)
        2'b00:bank0_data2 = buff0_out;
        2'b01:bank0_data2 = buff1_out;
        2'b10:bank0_data2 = buff2_out;
        2'b11:bank0_data2 = buff3_out;
```

```
        default:bank0_data2 = buff2_out;
      endcase // case(q_phase_rb2)
  end // always @ (...


  //buffer 3  4x1 mux
 always @(/*AUTOSENSE*/buff0_out or buff1_out or buff2_out
         or buff3_out or q_phase_rb3)
  begin
      case(q_phase_rb3)
        2'b00:bank0_data3 = buff0_out;
        2'b01:bank0_data3 = buff1_out;
        2'b10:bank0_data3 = buff2_out;
        2'b11:bank0_data3 = buff3_out;
        default:bank0_data3 = buff0_out;
      endcase // case(q_phase_rb3)
  end // always @ (...


  //buffer 4  4x1 mux
 always @(/*AUTOSENSE*/buff4_out or buff5_out or buff6_out
         or buff7_out or q_phase_rb0)
  begin
      case(q_phase_rb0)
        2'b00:bank1_data0 = buff4_out;
```

```
        2'b01:bank1_data0 = buff5_out;
        2'b10:bank1_data0 = buff6_out;
        2'b11:bank1_data0 = buff7_out;
        default:bank1_data0 = buff4_out;
      endcase // case(q_phase_rb0)
  end // always @ (...


  //buffer 5  4x1 mux
 always @(/*AUTOSENSE*/buff4_out or buff5_out or buff6_out
         or buff7_out or q_phase_rb1)
  begin
      case(q_phase_rb1)
        2'b00:bank1_data1 = buff4_out;
        2'b01:bank1_data1 = buff5_out;
        2'b10:bank1_data1 = buff6_out;
        2'b11:bank1_data1 = buff7_out;
        default:bank1_data1 = buff4_out;
      endcase // case(q_phase_rb1)
  end // always @ (...

  //buffer 6  4x1 mux
 always @(/*AUTOSENSE*/buff4_out or buff5_out or buff6_out
         or buff7_out or q_phase_rb2)
  begin
```

```
      case(q_phase_rb2)
        2'b00:bank1_data2 = buff4_out;
        2'b01:bank1_data2 = buff5_out;
        2'b10:bank1_data2 = buff6_out;
        2'b11:bank1_data2 = buff7_out;
        default:bank1_data2 = buff4_out;
      endcase // case(q_phase_rb2)
  end // always @ (...

  //buffer 7  4x1 mux
 always @(/*AUTOSENSE*/buff4_out or buff5_out or buff6_out
         or buff7_out or q_phase_rb3)
  begin
      case(q_phase_rb3)
        2'b00:bank1_data3 = buff4_out;
        2'b01:bank1_data3 = buff5_out;
        2'b10:bank1_data3 = buff6_out;
        2'b11:bank1_data3 = buff7_out;
        default:bank1_data3 = buff4_out;
      endcase // case(q_phase_rb3)
  end // always @ (...

 //clipper data from bank0 (sp0)
 always @(/*AUTOSENSE*/buff0_out or buff1_out or buff2_out
         or buff3_out or q_phase_clipp)
  begin
      case(q_phase_clipp)
        2'b00:bank0_clipp_data = buff0_out;
        2'b01:bank0_clipp_data = buff1_out;
        2'b10:bank0_clipp_data = buff2_out;
        2'b11:bank0_clipp_data = buff3_out;
        default:bank0_clipp_data = buff0_out;
      endcase // case(q_phase_clipp)
  end // always @ (...

//clipper data from bank1 (sp1)
always @(/*AUTOSENSE*/buff4_out or buff5_out or buff6_out
          or buff7_out or q_phase_clipp)
  begin
      case(q_phase_clipp)
        2'b00:bank1_clipp_data = buff4_out;
        2'b01:bank1_clipp_data = buff5_out;
        2'b10:bank1_clipp_data = buff6_out;
        2'b11:bank1_clipp_data = buff7_out;
        default:bank0_clipp_data = buff4_out;
      endcase // case(q_phase_clipp)
  end // always @ (...
```

```
reg [127:0] clipp_data;


//final mux selecting between left and right banks of the export buffers (sp0 vs. sp1)
always @(/*AUTOSENSE*/bank0_clipp_data or bank1_clipp_data
          or q_sp_bank_sel)
  begin
      case(q_sp_bank_sel)
        1'b0:clipp_data=bank0_clipp_data;
        1'b1:clipp_data=bank1_clipp_data;
      endcase // case(q_sp_bank_sel)
  end

  //four outputs ...one for each RB

always @(/*AUTOSENSE*/bank0_data0 or bank1_data0 or q_sp_bank_sel0)
  begin
      case(q_sp_bank_sel0)
        1'b0:rb0_data=bank0_data0;
        1'b1:rb0_data=bank1_data0;
      endcase // case(q_sp_bank_sel0)
  end
always @(/*AUTOSENSE*/bank0_data1 or bank1_data1 or q_sp_bank_sel1)
```

```
  begin
      case(q_sp_bank_sel1)
        1'b0:rb1_data=bank0_data1;
        1'b1:rb1_data=bank1_data1;
      endcase // case(q_sp_bank_sel1)
  end
always @(/*AUTOSENSE*/bank0_data2 or bank1_data2 or q_sp_bank_sel2)
  begin
      case(q_sp_bank_sel2)
        1'b0:rb2_data=bank0_data2;
        1'b1:rb2_data=bank1_data2;
      endcase // case(q_sp_bank_sel2)
  end
always @(/*AUTOSENSE*/bank0_data3 or bank1_data3 or q_sp_bank_sel3)
  begin
      case(q_sp_bank_sel3)
        1'b0:rb3_data=bank0_data3;
        1'b1:rb3_data=bank1_data3;
      endcase // case(q_sp_bank_sel3)
  end


//generating the data valid for the export buffer read over four cycles
always @(/*AUTOSENSE*/q0_read_valid_clipp or q1_read_valid_clipp
          or q2_read_valid_clipp or q3_read_valid_clipp
```

```
          or q_phase_clipp)
  begin
      case(q_phase_clipp)
        2'b00:clipp_data_valid = q0_read_valid_clipp;
        2'b01:clipp_data_valid = q1_read_valid_clipp;
        2'b10:clipp_data_valid = q2_read_valid_clipp;
        2'b11:clipp_data_valid = q3_read_valid_clipp;
        default:clipp_data_valid = q0_read_valid_clipp;
      endcase // case(q_phase_clipp)
  end // always @ (...


//generating the data valid for the export buffer read over four cycles
always @(/*AUTOSENSE*/q0_read_valid_rb0 or q1_read_valid_rb0
          or q2_read_valid_rb0 or q3_read_valid_rb0 or q_phase_rb0)
  begin
      case(q_phase_rb0)
        2'b00:rb0_data_valid = q0_read_valid_rb0;
        2'b01:rb0_data_valid = q1_read_valid_rb0;
        2'b10:rb0_data_valid = q2_read_valid_rb0;
        2'b11:rb0_data_valid = q3_read_valid_rb0;
        default:rb0_data_valid = q0_read_valid_rb0;
      endcase // case(q_phase_rb0)
  end // always @ (...

//generating the data valid for the export buffer read over four cycles
always @(/*AUTOSENSE*/q0_read_valid_rb1 or q1_read_valid_rb1
        or q2_read_valid_rb1 or q3_read_valid_rb1 or q_phase_rb1)
  begin
      case(q_phase_rb1)
        2'b00:rb1_data_valid = q0_read_valid_rb1;
        2'b01:rb1_data_valid = q1_read_valid_rb1;
        2'b10:rb1_data_valid = q2_read_valid_rb1;
        2'b11:rb1_data_valid = q3_read_valid_rb1;
        default:rb1_data_valid = q0_read_valid_rb1;
      endcase // case(q_phase_rb1)
  end // always @ (...

 //generating the data valid for the export buffer read over four cycles
always @(/*AUTOSENSE*/q0_read_valid_rb2 or q1_read_valid_rb2
        or q2_read_valid_rb2 or q3_read_valid_rb2 or q_phase_rb2)
  begin
      case(q_phase_rb2)
        2'b00:rb2_data_valid = q0_read_valid_rb2;
        2'b01:rb2_data_valid = q1_read_valid_rb2;
        2'b10:rb2_data_valid = q2_read_valid_rb2;
        2'b11:rb2_data_valid = q3_read_valid_rb2;
        default:rb2_data_valid = q0_read_valid_rb2;
      endcase // case(q_phase_rb2)
```

```
  end // always @ (...


//generating the data valid for the export buffer read over four cycles
always @(/*AUTOSENSE*/q0_read_valid_rb3 or q1_read_valid_rb3
        or q2_read_valid_rb3 or q3_read_valid_rb3 or q_phase_rb3)
  begin
      case(q_phase_rb3)
        2'b00:rb3_data_valid = q0_read_valid_rb3;
        2'b01:rb3_data_valid = q1_read_valid_rb3;
        2'b10:rb3_data_valid = q2_read_valid_rb3;
        2'b11:rb3_data_valid = q3_read_valid_rb3;
        default:rb3_data_valid = q0_read_valid_rb3;
      endcase // case(q_phase_rb3)
  end // always @ (...



reg [127:0]  q_clipp_data;
reg            q_clipp_data_valid;
reg         q_rb0_data_valid , q_rb1_data_valid, q_rb2_data_valid, q_rb3_data_valid;

always @(posedge sclk)
  begin
      q_rb0_data <= rb0_data;
```

```
       q_rb0_data_valid <= rb0_data_valid;
       q_rb1_data <= rb1_data;
       q_rb1_data_valid <= rb1_data_valid;
       q_rb2_data <= rb2_data;
       q_rb2_data_valid <= rb2_data_valid;
       q_rb3_data <= rb3_data;
       q_rb3_data_valid <= rb3_data_valid;
       q_clipp_data <= clipp_data;
       q_clipp_data_valid <= clipp_data_valid;
   end

  assign orb0_data = q_rb0_data;
  assign orb1_data = q_rb1_data;
  assign orb2_data = q_rb2_data;
  assign orb3_data = q_rb3_data;
  assign oclipp_data = q_clipp_data;

  assign orb0_data_valid = q_rb0_data_valid;
  assign orb1_data_valid = q_rb1_data_valid;
  assign orb2_data_valid = q_rb2_data_valid;
  assign orb3_data_valid = q_rb3_data_valid;
  assign oclipp_data_valid = q_clipp_data_valid;

endmodule // export_buffers
```

```
`include "header.v"
//                              -*- mode: verilog -*-
// filename        : pa.v
// description     : pa top block
// author          : mike  mantor
// created on      : sunday march 17 2002
// last modified by: .
// last modified on: .
// update count    : 0
// status          : initial
//--------------------------------------------------------------------------
//
// $id: //depot/r400/devel/parts_lib/src/gfx/pa/pa.v#34 $
//
// $change: 32279 $
//
//
// copyright:    trade secret of ati technologies, inc.
//                    © copyright 2001-2002, ati technologies, inc., (unpublished)
//
//                    all rights reserved.  this notice is intended as a precaution against
//                    inadvertent publication and does not imply publication or any waiver
//                    of confidentiality.  the year included in the foregoing notice is the
//                    year of creation of the work.
//
```

```
//--------------------------------------------------------------------------
module pa (
    // -------------------------------------------
    // chip signals
    // -------------------------------------------
    sclk_global,
    srst,
    RBBM_PA_soft_reset,         //primitive assembly soft reset
    CG_PA_pm_enb,
    RBBM_regclk_active,
    // -------------------------------------------
    // interface to the register bus (rbbm)
    // -------------------------------------------
    RBBM_a,                     // address
    RBBM_we,                    // write enable
    RBBM_wd,                    // write data
    RBBM_re,                    // read enable
    RBB_rs_in,                  // read strobe daisy chain in
    RBB_rs_out,                 // read strobe daisy chain out
    RBB_rd_in,                  // read data daisy chain in
    RBB_rd_out,                 // read data daisy chain out
    //PA_RBBM_nrtrtr,           // non-real-time ready-to-receive
    PA_RBBM_busy,               // busy signal reported to pa
    PA_a,                       // register address for daisy chain out
    PA_we,                      // register we for daisy chain out
```

```
    PA_wd,                      // register wd for daisy chain out
    PA_re,                      // register we for daisy chain out
    // -------------------------------------------
    // interface to rom
    // -------------------------------------------
    ROM_SP0_disable,
    ROM_SP1_disable,
    ROM_SP2_disable,
    ROM_SP3_disable,
    // -------------------------------------------
    // interface to vgt -- per vertex
    // -------------------------------------------
    VGT_PA_clip_v_vec_size,     // number of vertices in current vector
    VGT_PA_clip_v_state,        // state select
    VGT_PA_clip_v_send,         // ready-to-send
    PA_VGT_clip_v_rtr,          // ready-to-receive
    // -------------------------------------------
    // interface to vgt -- per primitive
    // -------------------------------------------
    VGT_PA_clip_p_indx0,        // internal vertex index 0
    VGT_PA_clip_p_indx1,        // internal vertex index 1
    VGT_PA_clip_p_indx2,        // internal vertex index 2
    VGT_PA_clip_p_edge_flags,   // edge flags
    VGT_PA_clip_p_eop,          // end-of-packet for state synchronization
    VGT_PA_clip_p_null_prim,
```

```
    VGT_PA_clip_p_dealloc,      // deallocation bits
    VGT_PA_clip_p_new_vtx_vect, // primitive contains vtx that was the first vtx in a vertex vector process
    VGT_PA_clip_p_send,         // ready-to-send
    PA_VGT_clip_p_rtr,          // ready-to-receive
    // -------------------------------------------
    // interface to vgt -- per state
    // -------------------------------------------
    VGT_PA_clip_s_type,         // clipper prim type. this is a sub-set of the input prim types.
    VGT_PA_clip_s_event,        // event
    VGT_PA_clip_s_state,        // state select
    VGT_PA_clip_s_send,         // ready-to-send
    PA_VGT_clip_s_rtr,          // ready-to-receive
    // -------------------------------------------
    // interface to the shader export 0 block
    // -------------------------------------------
    u0_SX_PA_send,
    u0_SX_PA_data,
    u0_PA_SX_req,
    u0_PA_SX_sp_id,
    u0_PA_SX_offset,
    u0_PA_SX_aux,
    u0_PA_SX_last,
    // -------------------------------------------
    // interface to the shader export 1 block
    // -------------------------------------------
```

```
u1_SX_PA_send,
u1_SX_PA_data,
u1_PA_SX_req,
u1_PA_SX_sp_id,
u1_PA_SX_offset,
u1_PA_SX_aux,
u1_PA_SX_last,
// -------------------------------------------
// interface to the scan converter
// -------------------------------------------
PA_SC_p0,
PA_SC_p1,
PA_SC_p2,
PA_SC_p3,
PA_SC_p4,
PA_SC_xy0,
PA_SC_xy1,
PA_SC_xy2,
PA_SC_zminmax,
PA_SC_cntl,
PA_SC_phase,
PA_SC_valid,
PA_SC_v0_indx,
SC_PA_earlyfrz
);
```

```

//`include "PA_clip_pkg.v"

  // ***********************************
  // i/o definitions
  // ***********************************
  // chip signals
  input         sclk_global;
  input         srst;
  input         RBBM_PA_soft_reset;
  input         CG_PA_pm_enb;
  input         RBBM_regclk_active;
  // interface to the register bus (rbbm)
  input   [16:2] RBBM_a;
  input         RBBM_we;
  input   [31:0] RBBM_wd;
  input         RBBM_re;
  input         RBB_rs_in;
  output         RBB_rs_out;
  input   [31:0] RBB_rd_in;
  output  [31:0] RBB_rd_out;
  //output         PA_RBBM_nrtrtr;
  output         PA_RBBM_busy;
  output  [16:2] PA_a;
  output         PA_we;
```

```
output  [31:0] PA_wd;
output         PA_re;
// interface to rom
input          ROM_SP0_disable;
input          ROM_SP1_disable;
input          ROM_SP2_disable;
input          ROM_SP3_disable;
// interface to vgt -- per vertex
input    [5:0] VGT_PA_clip_v_vec_size;
input    [2:0] VGT_PA_clip_v_state;
input          VGT_PA_clip_v_send;
output         PA_VGT_clip_v_rtr;
// interface to vgt -- per primitive
input    [5:0] VGT_PA_clip_p_indx0;
input    [5:0] VGT_PA_clip_p_indx1;
input    [5:0] VGT_PA_clip_p_indx2;
input    [2:0] VGT_PA_clip_p_edge_flags;
input          VGT_PA_clip_p_eop;
input          VGT_PA_clip_p_null_prim;
input    [2:0] VGT_PA_clip_p_dealloc;
input          VGT_PA_clip_p_new_vtx_vect;
input          VGT_PA_clip_p_send;
output         PA_VGT_clip_p_rtr;
// interface to vgt -- per state
input    [3:0] VGT_PA_clip_s_type;
```

```
input          VGT_PA_clip_s_event;
input    [2:0] VGT_PA_clip_s_state;
input          VGT_PA_clip_s_send;
output         PA_VGT_clip_s_rtr;
// interface to shader export east/0 sp0,2
input         u0_SX_PA_send;
input [127:0] u0_SX_PA_data;
output                u0_PA_SX_req;
output                u0_PA_SX_sp_id;
output [1:0]  u0_PA_SX_offset;
output                u0_PA_SX_aux;
output                u0_PA_SX_last;
// interface to shader export west/1 sp1,3
input         u1_SX_PA_send;
input [127:0] u1_SX_PA_data;
output                u1_PA_SX_req;
output                u1_PA_SX_sp_id;
output [1:0]  u1_PA_SX_offset;
output                u1_PA_SX_aux;
output                u1_PA_SX_last;
// interface to scan converter
output [17:0]  PA_SC_xy0;
output [17:0]  PA_SC_xy1;
output [17:0]  PA_SC_xy2;
output [31:0]  PA_SC_p0;
output [39:0]  PA_SC_p1;
output [31:0]  PA_SC_p2;
output [31:0]  PA_SC_p3;
output [31:0]  PA_SC_p4;
output [13:0]  PA_SC_zminmax;
output [29:0]  PA_SC_cntl;
output  [1:0]  PA_SC_phase;
output         PA_SC_valid;
output  [1:0]  PA_SC_v0_indx;
input          SC_PA_earlyfrz;

// clock gating internal wire
wire           sclk;
wire           sclk_reg;
wire           sclk_pa;

wire           pa_srst;
wire           pa_hard_srst;
wire           pa_soft_srst;
wire           cg_blk_gated_clk_override;
wire           regclk_active;
wire           reg_clk_en;
wire           pa_clk_en;

wire           SC_PA_earlyfrz_q;
```

```


// rbbm interface wires
wire  [16:2]     RBBM_a_q1;
wire                       RBBM_we_q1;
wire  [31:0]     RBBM_wd_q1;
wire                       RBBM_re_q1;
wire                       rbiu_block_rs;
wire  [31:0]     rbiu_block_rd;
wire             rbiu_ag_dx_clip_sp_def_sel;
wire  [3:0]      rbiu_ag_ucp0_sel;
wire  [3:0]      rbiu_ag_ucp1_sel;
wire  [3:0]      rbiu_ag_ucp2_sel;
wire  [3:0]      rbiu_ag_ucp3_sel;
wire  [3:0]      rbiu_ag_ucp4_sel;
wire  [3:0]      rbiu_ag_ucp5_sel;
wire  [3:0]      rbiu_ag_gb_sel;
wire  [3:0]      rbiu_ag_pntsz_sel;

wire             rbiu_ag_cpy;
wire  [31:0]     ag_rbiu_rdata;

wire             rbiu_cl_dx_clip_sp_def_sel;
wire                       rbiu_cl_status_sel;
wire             rbiu_cl_cpy;
```

```
    wire  [31:0]     cl_rbiu_rdata;

    wire             rbiu_ccg_expcntmd_sel;

    wire             rbiu_ccg_cpy;
    wire  [31:0]     ccg_rbiu_rdata;

    wire                       rbiu_vte_cpy;
    wire                       rbiu_vte_xscale_sel;
    wire                       rbiu_vte_xoffset_sel;
    wire                       rbiu_vte_yscale_sel;
    wire                       rbiu_vte_yoffset_sel;
    wire                       rbiu_vte_zscale_sel;
    wire                       rbiu_vte_zoffset_sel;
    wire                       rbiu_vte_cntl_sel;
    wire                       rbiu_vte_vtx_cntl_sel;
    wire                       rbiu_vte_window_offset_sel;
    wire                       rbiu_vte_window_offset_en_sel;
    wire                       rbiu_vte_persp_corr_dis_sel;
    wire     [31:0]  vte_rbiu_rdata;

    wire                       rbiu_su_imp_exp_sel;
    wire                       rbiu_su_draw_init_sel;
    wire                       rbiu_su_expand_lw_sel;
    wire                       rbiu_su_status_sel;
```

```
wire                       rbiu_su_point_size_sel;
wire                       rbiu_su_point_min_max_sel;
wire                       rbiu_su_line_cntl_sel;
wire                       rbiu_su_sc_mode_cntl_sel;
wire                       rbiu_su_cpy;
wire     [31:0]  su_rbiu_rdata;

// setup unit wires
// i/o for clip interface
wire  [31:0] clip_su_pt_size;
wire  [17:0] clip_su_x0;
wire  [17:0] clip_su_x1;
wire  [17:0] clip_su_x2;
wire  [17:0] clip_su_y0;
wire  [17:0] clip_su_y1;
wire  [17:0] clip_su_y2;
wire  [31:0] clip_su_z0;
wire  [31:0] clip_su_z1;
wire  [31:0] clip_su_z2;
wire  [31:0] clip_su_w0;
wire  [31:0] clip_su_w1;
wire  [31:0] clip_su_w2;
wire  [0:0] clip_su_ef0;
wire  [0:0] clip_su_ef1;
wire  [0:0] clip_su_ef2;
```

```
    wire  [31:0] clip_su_i0;
    wire  [31:0] clip_su_i1;
    wire  [31:0] clip_su_i2;
    wire  [31:0] clip_su_j0;
    wire  [31:0] clip_su_j1;
    wire  [31:0] clip_su_j2;
    wire  [31:0] clip_su_k0;
    wire  [31:0] clip_su_k1;
    wire  [31:0] clip_su_k2;
    wire  [10:0] clip_su_attr_indx0;
    wire  [10:0] clip_su_attr_indx1;
    wire  [10:0] clip_su_attr_indx2;
    wire  [2:0] clip_su_type;
    wire  [2:0] clip_su_st_indx;
    wire  [2:0] clip_su_dealloc_slot;
    wire  [0:0] clip_su_null_prim;
    wire  [0:0] clip_su_clipped;
    wire  [0:0] clip_su_fpov;
    wire  [0:0] clip_su_eop;
    wire  [0:0] clip_su_event;
    wire  [3:0] clip_su_event_id;
    wire  [0:0] clip_su_rts;
    wire  [0:0] clip_su_baryc_rts;
    wire  [0:0] su_clip_rtr;
    wire  [0:0] su_clip_baryc_rtr;
```

```
// ==========================
// wires for common rbiu bus
// ==========================
wire        rbiu_we;
        wire        rbiu_re;
        wire  [2:0] rbiu_waddr;
        wire  [2:0] rbiu_raddr;
        wire [31:0] rbiu_wdata;
// ==========================
// wires for registered input
// ==========================
wire [46:0] VGT_PA_input_data;
wire [46:0] VGT_PA_input_data_q;
wire [5:0]  VGT_PA_clip_v_vec_size_q;
wire [2:0]  VGT_PA_clip_v_state_q;
wire        VGT_PA_clip_v_send_q;
wire [5:0]  VGT_PA_clip_p_indx0_q;
wire [5:0]  VGT_PA_clip_p_indx1_q;
wire [5:0]  VGT_PA_clip_p_indx2_q;
wire [2:0]  VGT_PA_clip_p_edge_flags_q;
wire        VGT_PA_clip_p_eop_q;
wire        VGT_PA_clip_p_null_prim_q;
wire [2:0]  VGT_PA_clip_p_dealloc_q;
wire        VGT_PA_clip_p_new_vtx_vect_q;
wire        VGT_PA_clip_p_send_q;
```

```
wire [3:0]  VGT_PA_clip_s_type_q;
wire        VGT_PA_clip_s_event_q;
wire [2:0]  VGT_PA_clip_s_state_q;
wire        VGT_PA_clip_s_send_q;
// ==========================
// wires for registered output
// ==========================
wire [2:0]  PA_VGT_output_data;
wire [2:0]  PA_VGT_output_data_q;
wire        PA_VGT_clip_v_d;
wire        PA_VGT_clip_p_d;
wire        PA_VGT_clip_s_d;
wire [17:0] PA_SC_xy0_d;
wire [17:0] PA_SC_xy1_d;
wire [17:0] PA_SC_xy2_d;
wire [31:0] PA_SC_p0_d;
wire [39:0] PA_SC_p1_d;
wire [31:0] PA_SC_p2_d;
wire [31:0] PA_SC_p3_d;
wire [31:0] PA_SC_p4_d;
wire [13:0] PA_SC_zminmax_d;
wire [29:0] PA_SC_cntl_d;
wire  [1:0] PA_SC_phase_d;
wire        PA_SC_valid_d;
wire  [1:0] PA_SC_v0_indx_d;
```

```

wire [17:0] PA_SC_xy0_q;
wire [17:0] PA_SC_xy1_q;
wire [17:0] PA_SC_xy2_q;
wire [31:0] PA_SC_p0_q;
wire [39:0] PA_SC_p1_q;
wire [31:0] PA_SC_p2_q;
wire [31:0] PA_SC_p3_q;
wire [31:0] PA_SC_p4_q;
wire [13:0] PA_SC_zminmax_q;
wire [29:0] PA_SC_cntl_q;
wire  [1:0] PA_SC_phase_q;
wire        PA_SC_valid_q;
wire  [1:0] PA_SC_v0_indx_q;
wire        su_busy;

wire                u0_PA_SX_req_d;
wire                u0_PA_SX_sp_id_d;
wire [1:0]  u0_PA_SX_offset_d;
wire                u0_PA_SX_aux_d;
wire                u0_PA_SX_last_d;
wire                u1_PA_SX_req_d;
wire                u1_PA_SX_sp_id_d;
wire [1:0]  u1_PA_SX_offset_d;
wire                u1_PA_SX_aux_d;
wire                u1_PA_SX_last_d;

wire        u0_PA_SX_req_q;
wire        u0_PA_SX_sp_id_q;
wire [1:0]  u0_PA_SX_offset_q;
wire        u0_PA_SX_aux_q;
wire        u0_PA_SX_last_q;

wire        u1_PA_SX_req_q;
wire        u1_PA_SX_sp_id_q;
wire [1:0]  u1_PA_SX_offset_q;
wire        u1_PA_SX_aux_q;
wire        u1_PA_SX_last_q;

// shader export interface/clip code generator wires
wire [128:0] SX0_PA_input_data;
wire [128:0] SX0_PA_input_data_q;
wire SX0_PA_input_data_write;
wire [127:0] SX0_PA_input_data_wrdata;

wire [128:0] SX1_PA_input_data;
wire [128:0] SX1_PA_input_data_q;
wire SX1_PA_input_data_write;
wire [127:0] SX1_PA_input_data_wrdata;

```

```

// =========================
// wires for pa_sxifccg
// =========================
wire [1:0] cl_ccg_outsm_clr_orig_vertices;
wire cl_ccg_ccgen_to_clipcc_fifo_full;
wire arb_ccg_xfc;
wire ccg_ag_pos_mem_we;
wire [5:0] ccg_ag_pntsz_mem_wraddr;
wire [5:0] ccg_ag_pos_mem_wraddr;
wire [127:0] ccg_ag_pos_pntsz_mem_wrdata;
wire ccg_ag_pntsz_mem_we;
wire [31:0] ccg_ag_pntsz_mem_wrdata;
wire [17-1:0] ccg_cl_wrdata;
wire ccg_cl_write;
wire [15:0] ccg_arb_data;
wire [7-1:0] cl_ccg_state0;
wire [7-1:0] cl_ccg_state1;
wire [7-1:0] cl_ccg_state2;
wire [7-1:0] cl_ccg_state3;
wire [7-1:0] cl_ccg_state4;
wire [7-1:0] cl_ccg_state5;
wire [7-1:0] cl_ccg_state6;
wire [7-1:0] cl_ccg_state7;
wire [6:0] ccg_rbiu_rdata_26_downto_20;
```

```
wire [7-1:0] sxif_state0;
wire [7-1:0] sxif_state1;
wire [7-1:0] sxif_state2;
wire [7-1:0] sxif_state3;
wire [7-1:0] sxif_state4;
wire [7-1:0] sxif_state5;
wire [7-1:0] sxif_state6;
wire [7-1:0] sxif_state7;

// =========================
// wires for pa_clipper
// =========================
//wire [clip_state_width-1:0] clip_st_w_data;
//wire [14-1:0] clip_st_w_data;
//wire clip_st_sel;
wire arb_cl_xfc;
wire [5:0] ag_cl_vertex_store_indx;
wire [1:0] ag_cl_valid_bit_set;
wire [3:0] ag_cl_user_clip_indx;
wire ag_cl_vv_cc_test;
wire ag_cl_ucp_cc_test;
wire ag_cl_bcc_cc_test;
wire ag_cl_ps_ucp_cc_test;
wire ag_cl_ps_engh_test;
wire cl_arb_ve_valid;
```

```
wire [29:0] cl_arb_data;
wire vmb_cl_rei_r0vld;
wire vmb_cl_rei_r1vld;
wire cl_rei_clear_result;
wire clipper_busy;
wire ag_cl_pntsz_mem_blocked;
wire [31:0] ag_cl_pntsz_mem_rdata;
wire cl_ag_pntsz_mem_re;
wire [5:0] cl_ag_pntsz_mem_raddr;
wire clip_to_ag_point_buf_re;
wire [5:0] clip_to_ag_point_buf_raddr;
wire [31:0] ag_to_clip_point_size;

// =========================
// wires for PA_ag
// =========================
wire [2:0] ag_ve_opcode;
wire [31:0] ag_ve_in_a0;
wire [31:0] ag_ve_in_a1;
wire [31:0] ag_ve_in_a2;
wire [31:0] ag_ve_in_a3;
wire [31:0] ag_ve_in_b0;
wire [31:0] ag_ve_in_b1;
wire [31:0] ag_ve_in_b2;
wire [31:0] ag_ve_in_b3;
```

```
  wire ag_ve_a_is_wwww;
  wire ag_ve_broadcast_x;
  wire ag_ve_abs_a;
  wire ag_ve_abs_b;
  wire ag_ve_abs_c;
  wire ag_ve_ax_negate;
  wire ag_ve_ay_negate;
  wire ag_ve_az_negate;
  wire ag_ve_aw_negate;
  wire ag_ve_bx_negate;
  wire ag_ve_by_negate;
  wire ag_ve_bz_negate;
  wire ag_ve_bw_negate;
  wire ag_ve_cx_negate;
  wire ag_ve_cy_negate;
  wire ag_ve_cz_negate;
  wire ag_ve_cw_negate;
  wire ag_ve_bcc_flat_tst;
  wire [2:0] ag_ve_out_mem_sel;
  wire [5:0] ag_ve_out_addr;
  wire [3:0] ag_ve_out_we;
  wire ag_ve_accum_sel;
  wire [3:0] ag_ve_pre_accum_we;
  wire [9:0] vmb_ve_tst_rtn_stat;
  wire [5:0] ve_waddr;
```

```
wire [3:0] ve_veoc_vector_back_we;
wire [3:0] ve_cliptemp_vector_we;
wire [127:0] ve_wdata;

// ==========================
// wires for PA_ag
// ==========================
wire [2:0] ag_vte_opcode;
wire [2:0] ag_vte_st_indx;
wire [1:0] ag_vte_vertex_store_indx;

// ==========================
// wires for PA_vte
// ==========================
wire [1:0] vte_vertex_store_indx;
wire [2:0] vte_opcode;
wire [127:0] vte_d;
    wire    [31:0]  vte_rcp_d;
    wire            vte_rcp_rts;
    wire            vcm_rcp_rei_xfc;
    wire    [31:0]  vcm_rcp_rei_d;

reg pa_sc_valid;
wire set_PA_RBBM_busy;
wire PA_RBBM_busy_d;
```

```
  reg [3:0] PA_RBBM_busy_delay;
  reg PA_RBBM_busy_reg;
  wire vcm_rcp_rei_rtr;
  wire [31:0] vmc_rei_rcp_d;
  wire vmc_rei_rcp_rts;
  wire [31:0] rei_sc_r0data;
  wire [31:0] rei_sc_r1data;


//================================================
// pick off part of position data
//================================================
  assign ccg_ag_pntsz_mem_wrdata = ccg_ag_pos_pntsz_mem_wrdata[31:0];

  assign ccg_rbiu_rdata = {5'h0, ccg_rbiu_rdata_26_downto_20, 20'h0};

//================================================
// create sclk and interface to clock gating logic
//================================================
  //create sclk by registering sclk_global
  ati_master_clock_permanent uati_master_clock_permanent(
     .clk_in(sclk_global),
     .clk_out(sclk)
  );

```

```
// register the perm clk gate override signal
ati_dff_in #(1) uati_dff_in_pm_en(
   .clk(sclk),
   .d(CG_PA_pm_enb),
   .q(cg_blk_gated_clk_override)
);

// create registered clock enable signals based on active signals
ati_dff_in #(1) uati_dff_in_regclk_active(
   .clk(sclk),
   .d(RBBM_regclk_active),
   .q(regclk_active)
);

// duplicate the function of the PA_SC_valid flop
always @(posedge sclk_pa) begin
  if (SC_PA_earlyfrz_q == 'h1) begin
    pa_sc_valid <= PA_SC_valid_d;
  end
end

// busy signal generation
assign set_PA_RBBM_busy = VGT_PA_clip_p_send_q |
                          VGT_PA_clip_s_send_q |
                          VGT_PA_clip_v_send_q |
               clipper_busy |
               su_busy |
               pa_sc_valid;

  always @(posedge sclk_pa) begin
   if (srst) begin
    PA_RBBM_busy_delay <= 'h0;
    end
   else begin
    PA_RBBM_busy_delay <= {PA_RBBM_busy_delay[2:0], set_PA_RBBM_busy};
   end
  end

  assign PA_RBBM_busy_d = PA_RBBM_busy_delay != 'h0;


  //this enable would be intiated by RBBM_regclk_active and held high in the
  //block as long as necessary to ensure all data could be read
  assign reg_clk_en = regclk_active | PA_RBBM_busy;

  //this active signal would be a collection of request from external blocks that require
  //the block clocks to be enabled along with internal busy signals that require the clocks
  //to stay on
  assign pa_clk_en = !cg_blk_gated_clk_override | PA_RBBM_busy;

```

```

  //generate the sclk_reg clock tree
  ati_master_clock_gater uati_master_clock_gater_sclk_reg (
    .clk_in(sclk_global),
    .clk(sclk),
    .en(reg_clk_en),
    .pm_enb(cg_blk_gated_clk_override),
    .clk_out(sclk_reg)
  );

  //generate sclk_sc clock tree
  ati_master_clock_gater uati_master_clock_gater_sc_clk (
    .clk_in(sclk_global),
    .clk(sclk),
    .en(pa_clk_en),
    .pm_enb(cg_blk_gated_clk_override),
    .clk_out(sclk_pa)
  );
//================================================
// instantiate common rbbm interface block
//================================================
  //rbbm interface register
  ati_rbbm_intf uati_rbbm_intf(
    .sclk_reg(sclk_reg),
    .rbbm_we(RBBM_we),
```

```
    .rbbm_re(RBBM_re),
    .rbbm_a(RBBM_a),
    .rbbm_wd(RBBM_wd),
    .reg_we(RBBM_we_q1),
    .reg_re(RBBM_re_q1),
    .reg_a(RBBM_a_q1),
    .reg_wd(RBBM_wd_q1),
    .pipe_we(PA_we),
    .pipe_re(PA_re),
    .pipe_a(PA_a),
    .pipe_wd(PA_wd),
    .rbbm_rs_in(RBB_rs_in),
    .rbbm_rd_in(RBB_rd_in),
    .block_rs(rbiu_block_rs),
    .block_rd(rbiu_block_rd),
    .rbbm_rs_out(RBB_rs_out),
    .rbbm_rd_out(RBB_rd_out)
  );

//================================================
// create hard and soft reset signal
//================================================
  //register input reset
  ati_dff_in #(1) uati_dff_in_pa_hard_srst(
    .clk(sclk),
```

```
    .d(srst),
    .q(pa_hard_srst)
  );

  //register input soft resets
  ati_dff_in #(1) uati_dff_in_pa_soft_srst(
    .clk(sclk),
    .d(RBBM_PA_soft_reset),
    .q(pa_soft_srst)
  );

  //use this in the block for the srst everywhere except
  //state storage that can only get reset by a hard reset
  assign pa_srst = pa_soft_srst | pa_hard_srst;

// ===================
// register outputs
// ===================
  ati_dff_out #(3) uVGT_PA_out_intf(
     .clk(sclk_pa),
     .d(PA_VGT_output_data),
     .q(PA_VGT_output_data_q)
  );

  ati_dff_en_out #(32) uPA_SC_out0(
    .clk(sclk_pa),
    .en(SC_PA_earlyfrz_q),
    .d(PA_SC_p0_d),
    .q(PA_SC_p0_q)
);

ati_dff_en_out #(40) uPA_SC_out1(
    .clk(sclk_pa),
    .en(SC_PA_earlyfrz_q),
    .d(PA_SC_p1_d),
    .q(PA_SC_p1_q)
);

ati_dff_en_out #(32) uPA_SC_out2(
    .clk(sclk_pa),
    .en(SC_PA_earlyfrz_q),
    .d(PA_SC_p2_d),
    .q(PA_SC_p2_q)
);

ati_dff_en_out #(32) uPA_SC_out3(
    .clk(sclk_pa),
    .en(SC_PA_earlyfrz_q),
    .d(PA_SC_p3_d),
    .q(PA_SC_p3_q)
```

```
);

ati_dff_en_out #(32) uPA_SC_out4(
    .clk(sclk_pa),
    .en(SC_PA_earlyfrz_q),
    .d(PA_SC_p4_d),
    .q(PA_SC_p4_q)
);

ati_dff_en_out #(54) uPA_SC_out5(
    .clk(sclk_pa),
    .en(SC_PA_earlyfrz_q),
    .d({PA_SC_xy2_d,PA_SC_xy1_d,PA_SC_xy0_d}),
    .q({PA_SC_xy2_q,PA_SC_xy1_q,PA_SC_xy0_q})
);

ati_dff_en_out #(49) uPA_SC_out6(
    .clk(sclk_pa),
    .en(SC_PA_earlyfrz_q),

.d({PA_SC_v0_indx_d,PA_SC_valid_d,PA_SC_cntl_d,PA_SC_phase_d,PA_SC_zminmax_d}
),

.q({PA_SC_v0_indx_q,PA_SC_valid_q,PA_SC_cntl_q,PA_SC_phase_q,PA_SC_zminmax_q}
)
    );

```

```
// ====================
// u0_PA_SX
// ====================

ati_dff_out #(6) uPA_SX0_data_out (
    .clk(sclk_pa),
    .d({u0_PA_SX_req_d,
                        u0_PA_SX_sp_id_d,
                        u0_PA_SX_offset_d,
                        u0_PA_SX_aux_d,
                        u0_PA_SX_last_d}),
    .q({u0_PA_SX_req_q,
        u0_PA_SX_sp_id_q,
        u0_PA_SX_offset_q,
        u0_PA_SX_aux_q,
        u0_PA_SX_last_q})
);

// ====================
// u1_PA_SX
// ====================

ati_dff_out #(6) uPA_SX1_data_out (
    .clk(sclk_pa),
    .d({u1_PA_SX_req_d,
```

```
                        u1_PA_SX_sp_id_d,
                        u1_PA_SX_offset_d,
                        u1_PA_SX_aux_d,
                        u1_PA_SX_last_d}),
    .q({u1_PA_SX_req_q,
        u1_PA_SX_sp_id_q,
        u1_PA_SX_offset_q,
        u1_PA_SX_aux_q,
        u1_PA_SX_last_q})
);

// ====================
// PA_RBBM_busy
// ====================
ati_dff_out #(1) uPA_RBBM_busy (
  .clk(sclk_pa),
  .d(PA_RBBM_busy_d),
  .q(PA_RBBM_busy)
);

// ==================
// register inputs
// ==================
ati_dff_in #(47) uVGT_PA_in_inft(
    .clk(sclk_pa),
    .d(VGT_PA_input_data),
    .q(VGT_PA_input_data_q)
);

ati_dff_in #(129) uu0_SX_PA_data_inft(
    .clk(sclk_pa),
    .d(SX0_PA_input_data),
    .q(SX0_PA_input_data_q)
);

ati_dff_in #(129) uu1_SX_PA_data_inft(
    .clk(sclk_pa),
    .d(SX1_PA_input_data),
    .q(SX1_PA_input_data_q)
);

        assign u0_PA_SX_req                     = u0_PA_SX_req_q;
        assign u0_PA_SX_sp_id           = u0_PA_SX_sp_id_q;
        assign u0_PA_SX_offset          = u0_PA_SX_offset_q;
        assign u0_PA_SX_aux                     = u0_PA_SX_aux_q;
        assign u0_PA_SX_last            = u0_PA_SX_last_q;
        assign u1_PA_SX_req                     = u1_PA_SX_req_q;
        assign u1_PA_SX_sp_id           = u1_PA_SX_sp_id_q;
        assign u1_PA_SX_offset          = u1_PA_SX_offset_q;
        assign u1_PA_SX_aux                     = u1_PA_SX_aux_q;
```

```
        assign u1_PA_SX_last            = u1_PA_SX_last_q;
assign PA_SC_v0_indx      = PA_SC_v0_indx_q;
assign PA_SC_valid        = PA_SC_valid_q;
assign PA_SC_cntl         = PA_SC_cntl_q;
assign PA_SC_phase        = PA_SC_phase_q;
assign PA_SC_zminmax      = PA_SC_zminmax_q;
assign PA_SC_xy0          = PA_SC_xy0_q;
assign PA_SC_xy1          = PA_SC_xy1_q;
assign PA_SC_xy2          = PA_SC_xy2_q;
assign PA_SC_p0           = PA_SC_p0_q;
assign PA_SC_p1           = PA_SC_p1_q;
assign PA_SC_p2           = PA_SC_p2_q;
assign PA_SC_p3           = PA_SC_p3_q;
assign PA_SC_p4           = PA_SC_p4_q;

assign {VGT_PA_clip_v_vec_size_q,     // [46:41]
        VGT_PA_clip_v_state_q,        // [40:38]
        VGT_PA_clip_v_send_q,         // [37]
        VGT_PA_clip_p_indx0_q,        // [36:31]
        VGT_PA_clip_p_indx1_q,        // [30:25]
        VGT_PA_clip_p_indx2_q,        // [24:19]
        VGT_PA_clip_p_edge_flags_q,   // [18:16]
        VGT_PA_clip_p_eop_q,          // [15]
        VGT_PA_clip_p_null_prim_q,    // [14]
        VGT_PA_clip_p_dealloc_q,      // [13:11]
```

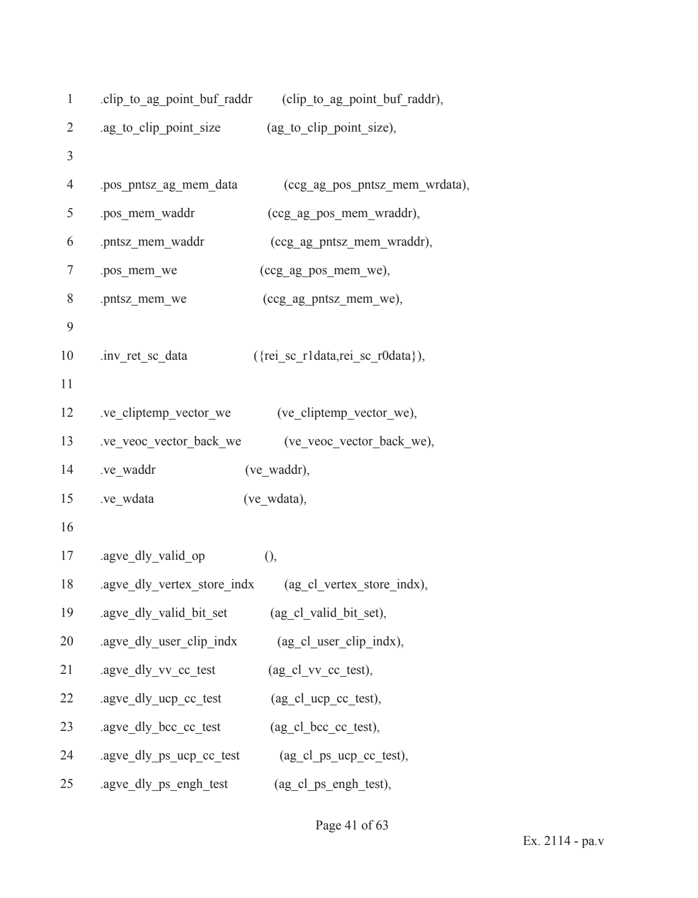
```
        VGT_PA_clip_p_new_vtx_vect_q,  // [10]
        VGT_PA_clip_p_send_q,          // [9]
        VGT_PA_clip_s_event_q,         // [8]
        VGT_PA_clip_s_type_q,          // [7:4]
        VGT_PA_clip_s_state_q,         // [3:1]
        VGT_PA_clip_s_send_q}          // [0]
                        = VGT_PA_input_data_q;

assign VGT_PA_input_data = {VGT_PA_clip_v_vec_size,    // [46:41]
                    VGT_PA_clip_v_state,       // [40:38]
                    VGT_PA_clip_v_send,        // [37]
                    VGT_PA_clip_p_indx0,       // [36:31]
                    VGT_PA_clip_p_indx1,       // [30:25]
                    VGT_PA_clip_p_indx2,       // [24:19]
                    VGT_PA_clip_p_edge_flags,  // [18:16]
                    VGT_PA_clip_p_eop,         // [15]
                    VGT_PA_clip_p_null_prim,   // [14]
                    VGT_PA_clip_p_dealloc,     // [13:11]
                    VGT_PA_clip_p_new_vtx_vect,// [10]
                    VGT_PA_clip_p_send,        // [9]
                    VGT_PA_clip_s_event,       // [8]
                    VGT_PA_clip_s_type,        // [7:4]
                    VGT_PA_clip_s_state,       // [3:1]
                    VGT_PA_clip_s_send};       // [0]

```

```
assign PA_VGT_clip_v_rtr        = PA_VGT_output_data_q[2];
assign PA_VGT_clip_p_rtr        = PA_VGT_output_data_q[1];
assign PA_VGT_clip_s_rtr        = PA_VGT_output_data_q[0];

assign PA_VGT_output_data       = {PA_VGT_clip_v_d,
                                   PA_VGT_clip_p_d,
                                   PA_VGT_clip_s_d};

assign SX0_PA_input_data = {u0_SX_PA_send,
                        u0_SX_PA_data};

assign {SX0_PA_input_data_write,
        SX0_PA_input_data_wrdata} = SX0_PA_input_data_q;

assign SX1_PA_input_data = {u1_SX_PA_send,
                        u1_SX_PA_data};

assign {SX1_PA_input_data_write,
        SX1_PA_input_data_wrdata} = SX1_PA_input_data_q;


//register pipe freeze signal before using it
ati_dff_in #(1) uati_dff_in_earlyfrz(
   .clk(sclk_pa),
   .d(SC_PA_earlyfrz),
      .q(SC_PA_earlyfrz_q)
    );

// rbbm interface
pa_rbiu upa_rbiu (
        // chip signals
        .CG_PA_sclk_reg(sclk_reg),
        // interface to the global register bus (rbbm)
        .RBBM_a_q1(RBBM_a_q1),                  // address
        .RBBM_we_q1(RBBM_we_q1),                  // write enable
        .RBBM_wd_q1(RBBM_wd_q1),                  // write data
        .RBBM_re_q1(RBBM_re_q1),                  // read enable
        // rbbm read data daisy chain
        .rbiu_block_rs(rbiu_block_rs),              // read strobe daisy chain out
        .rbiu_block_rd(rbiu_block_rd),            // read data daisy chain in
   // common rbiu interface to vte,su,ccg,cl,ag
   .rbiu_we(rbiu_we),
   .rbiu_re(rbiu_re),
   .rbiu_waddr(rbiu_waddr),
   .rbiu_raddr(rbiu_raddr),
   .rbiu_wdata(rbiu_wdata),
        // interface to ag
        .ag_rbiu_rdata(ag_rbiu_rdata),
        .rbiu_ag_cpy(rbiu_ag_cpy),
        .rbiu_ag_dx_clip_sp_def_sel(rbiu_ag_dx_clip_sp_def_sel),
```

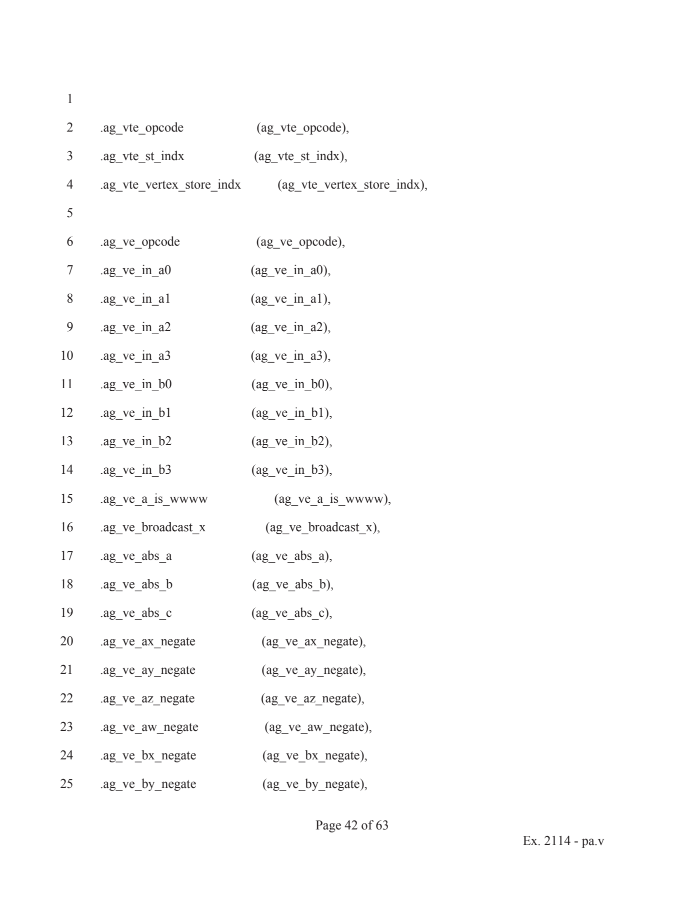
```
        .rbiu_ag_ucp0_sel(rbiu_ag_ucp0_sel),
        .rbiu_ag_ucp1_sel(rbiu_ag_ucp1_sel),
        .rbiu_ag_ucp2_sel(rbiu_ag_ucp2_sel),
        .rbiu_ag_ucp3_sel(rbiu_ag_ucp3_sel),
        .rbiu_ag_ucp4_sel(rbiu_ag_ucp4_sel),
        .rbiu_ag_ucp5_sel(rbiu_ag_ucp5_sel),
        .rbiu_ag_gb_sel(rbiu_ag_gb_sel),
        .rbiu_ag_pntsz_sel(rbiu_ag_pntsz_sel),
        // interface to cl
        .cl_rbiu_rdata(cl_rbiu_rdata),
        .rbiu_cl_cpy(rbiu_cl_cpy),
        .rbiu_cl_dx_clip_sp_def_sel(rbiu_cl_dx_clip_sp_def_sel),
        .rbiu_cl_status_sel(rbiu_cl_status_sel),
        // interface to ccg
        .ccg_rbiu_rdata(ccg_rbiu_rdata),
        .rbiu_ccg_cpy(rbiu_ccg_cpy),
        .rbiu_ccg_expcntmd_sel(rbiu_ccg_expcntmd_sel),
        // interface to vte
        .vte_rbiu_rdata(vte_rbiu_rdata),
        .rbiu_vte_cpy(rbiu_vte_cpy),
        .rbiu_vte_xscale_sel(rbiu_vte_xscale_sel),
        .rbiu_vte_xoffset_sel(rbiu_vte_xoffset_sel),
        .rbiu_vte_yscale_sel(rbiu_vte_yscale_sel),
        .rbiu_vte_yoffset_sel(rbiu_vte_yoffset_sel),
        .rbiu_vte_zscale_sel(rbiu_vte_zscale_sel),
```

```
        .rbiu_vte_zoffset_sel(rbiu_vte_zoffset_sel),
        .rbiu_vte_cntl_sel(rbiu_vte_cntl_sel),
        .rbiu_vte_vtx_cntl_sel(rbiu_vte_vtx_cntl_sel),
        .rbiu_vte_window_offset_sel(rbiu_vte_window_offset_sel),
        .rbiu_vte_window_offset_en_sel(rbiu_vte_window_offset_en_sel),
        .rbiu_vte_persp_corr_dis_sel(rbiu_vte_persp_corr_dis_sel),
        // interface to su
        .su_rbiu_rdata(su_rbiu_rdata),
        .rbiu_su_cpy(rbiu_su_cpy),
        .rbiu_su_expand_lw_sel(rbiu_su_expand_lw_sel),
        .rbiu_su_imp_exp_sel(rbiu_su_imp_exp_sel),
        .rbiu_su_draw_init_sel(rbiu_su_draw_init_sel),
        .rbiu_su_status_sel(rbiu_su_status_sel),
        .rbiu_su_point_size_sel(rbiu_su_point_size_sel),
        .rbiu_su_point_min_max_sel(rbiu_su_point_min_max_sel),
        .rbiu_su_line_cntl_sel(rbiu_su_line_cntl_sel),
        .rbiu_su_sc_mode_cntl_sel(rbiu_su_sc_mode_cntl_sel)
);

pa_ag
upa_ag
 (
  .sclk                (sclk_pa),
  .srst                (srst),

```

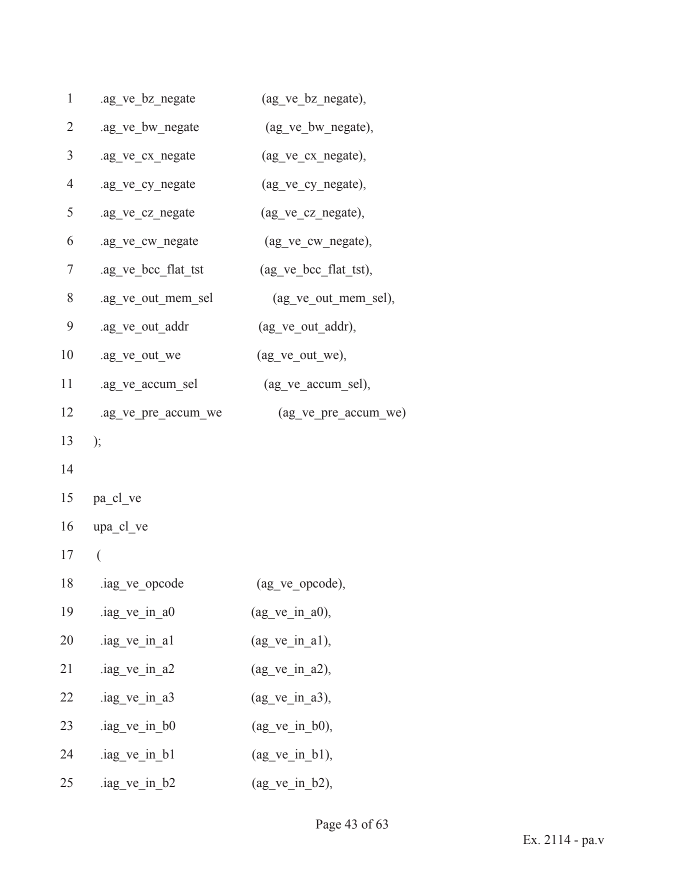
```
.rbiu_ag_dx_clip_sp_def_sel     (rbiu_ag_dx_clip_sp_def_sel),
.rbiu_ag_ucp0_sel               (rbiu_ag_ucp0_sel),
.rbiu_ag_ucp1_sel               (rbiu_ag_ucp1_sel),
.rbiu_ag_ucp2_sel               (rbiu_ag_ucp2_sel),
.rbiu_ag_ucp3_sel               (rbiu_ag_ucp3_sel),
.rbiu_ag_ucp4_sel               (rbiu_ag_ucp4_sel),
.rbiu_ag_ucp5_sel               (rbiu_ag_ucp5_sel),
.rbiu_ag_gb_sel                 (rbiu_ag_gb_sel),
.rbiu_ag_pntsz_sel              (rbiu_ag_pntsz_sel),

.rbiu_we                    (rbiu_we),
.rbiu_wa                    (rbiu_waddr),
.rbiu_wd                    (rbiu_wdata),
.rbiu_cpy                   (rbiu_su_cpy),
.rbiu_re                   (rbiu_re),
.ag_rbiu_rd                  (ag_rbiu_rdata),

.ccg_to_arb_data              (ccg_arb_data),
.arb_ccg_xfc                 (arb_ccg_xfc),

.clip_ve_valid               (cl_arb_ve_valid),
.clip_to_arb_data             (cl_arb_data),
.arb_clip_xfc                (arb_cl_xfc),

.clip_to_ag_point_buf_re        (clip_to_ag_point_buf_re),
```



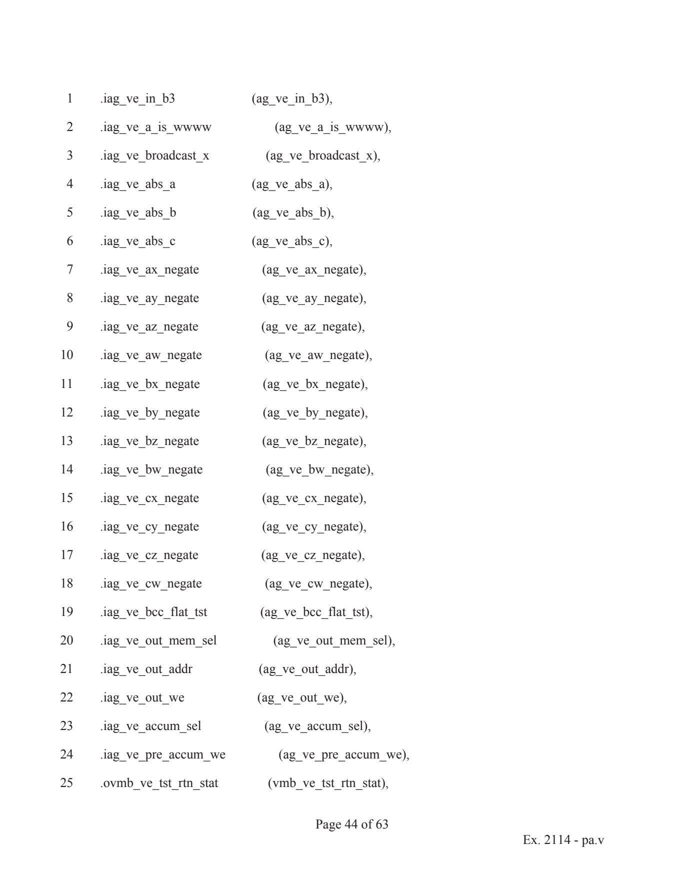
```
    .clip_to_ag_point_buf_raddr        (clip_to_ag_point_buf_raddr),
    .ag_to_clip_point_size             (ag_to_clip_point_size),

    .pos_pntsz_ag_mem_data             (ccg_ag_pos_pntsz_mem_wrdata),
    .pos_mem_waddr                     (ccg_ag_pos_mem_wraddr),
    .pntsz_mem_waddr                   (ccg_ag_pntsz_mem_wraddr),
    .pos_mem_we                        (ccg_ag_pos_mem_we),
    .pntsz_mem_we                      (ccg_ag_pntsz_mem_we),

    .inv_ret_sc_data                   ({rei_sc_r1data,rei_sc_r0data}),

    .ve_cliptemp_vector_we             (ve_cliptemp_vector_we),
    .ve_veoc_vector_back_we            (ve_veoc_vector_back_we),
    .ve_waddr                          (ve_waddr),
    .ve_wdata                          (ve_wdata),

    .agve_dly_valid_op                 (),
    .agve_dly_vertex_store_indx        (ag_cl_vertex_store_indx),
    .agve_dly_valid_bit_set            (ag_cl_valid_bit_set),
    .agve_dly_user_clip_indx           (ag_cl_user_clip_indx),
    .agve_dly_vv_cc_test               (ag_cl_vv_cc_test),
    .agve_dly_ucp_cc_test              (ag_cl_ucp_cc_test),
    .agve_dly_bcc_cc_test              (ag_cl_bcc_cc_test),
    .agve_dly_ps_ucp_cc_test           (ag_cl_ps_ucp_cc_test),
    .agve_dly_ps_engh_test             (ag_cl_ps_engh_test),
```



```

    .ag_vte_opcode                     (ag_vte_opcode),
    .ag_vte_st_indx                    (ag_vte_st_indx),
    .ag_vte_vertex_store_indx          (ag_vte_vertex_store_indx),

    .ag_ve_opcode                      (ag_ve_opcode),
    .ag_ve_in_a0                       (ag_ve_in_a0),
    .ag_ve_in_a1                       (ag_ve_in_a1),
    .ag_ve_in_a2                       (ag_ve_in_a2),
    .ag_ve_in_a3                       (ag_ve_in_a3),
    .ag_ve_in_b0                       (ag_ve_in_b0),
    .ag_ve_in_b1                       (ag_ve_in_b1),
    .ag_ve_in_b2                       (ag_ve_in_b2),
    .ag_ve_in_b3                       (ag_ve_in_b3),
    .ag_ve_a_is_wwww                   (ag_ve_a_is_wwww),
    .ag_ve_broadcast_x                 (ag_ve_broadcast_x),
    .ag_ve_abs_a                       (ag_ve_abs_a),
    .ag_ve_abs_b                       (ag_ve_abs_b),
    .ag_ve_abs_c                       (ag_ve_abs_c),
    .ag_ve_ax_negate                   (ag_ve_ax_negate),
    .ag_ve_ay_negate                   (ag_ve_ay_negate),
    .ag_ve_az_negate                   (ag_ve_az_negate),
    .ag_ve_aw_negate                   (ag_ve_aw_negate),
    .ag_ve_bx_negate                   (ag_ve_bx_negate),
    .ag_ve_by_negate                   (ag_ve_by_negate),
```



```
    .ag_ve_bz_negate                   (ag_ve_bz_negate),
    .ag_ve_bw_negate                   (ag_ve_bw_negate),
    .ag_ve_cx_negate                   (ag_ve_cx_negate),
    .ag_ve_cy_negate                   (ag_ve_cy_negate),
    .ag_ve_cz_negate                   (ag_ve_cz_negate),
    .ag_ve_cw_negate                   (ag_ve_cw_negate),
    .ag_ve_bcc_flat_tst                (ag_ve_bcc_flat_tst),
    .ag_ve_out_mem_sel                 (ag_ve_out_mem_sel),
    .ag_ve_out_addr                    (ag_ve_out_addr),
    .ag_ve_out_we                      (ag_ve_out_we),
    .ag_ve_accum_sel                   (ag_ve_accum_sel),
    .ag_ve_pre_accum_we                (ag_ve_pre_accum_we)
  );

  pa_cl_ve
  upa_cl_ve
   (
    .iag_ve_opcode                     (ag_ve_opcode),
    .iag_ve_in_a0                      (ag_ve_in_a0),
    .iag_ve_in_a1                      (ag_ve_in_a1),
    .iag_ve_in_a2                      (ag_ve_in_a2),
    .iag_ve_in_a3                      (ag_ve_in_a3),
    .iag_ve_in_b0                      (ag_ve_in_b0),
    .iag_ve_in_b1                      (ag_ve_in_b1),
    .iag_ve_in_b2                      (ag_ve_in_b2),
```



```
    .iag_ve_in_b3                      (ag_ve_in_b3),
    .iag_ve_a_is_wwww                  (ag_ve_a_is_wwww),
    .iag_ve_broadcast_x                (ag_ve_broadcast_x),
    .iag_ve_abs_a                      (ag_ve_abs_a),
    .iag_ve_abs_b                      (ag_ve_abs_b),
    .iag_ve_abs_c                      (ag_ve_abs_c),
    .iag_ve_ax_negate                  (ag_ve_ax_negate),
    .iag_ve_ay_negate                  (ag_ve_ay_negate),
    .iag_ve_az_negate                  (ag_ve_az_negate),
    .iag_ve_aw_negate                  (ag_ve_aw_negate),
    .iag_ve_bx_negate                  (ag_ve_bx_negate),
    .iag_ve_by_negate                  (ag_ve_by_negate),
    .iag_ve_bz_negate                  (ag_ve_bz_negate),
    .iag_ve_bw_negate                  (ag_ve_bw_negate),
    .iag_ve_cx_negate                  (ag_ve_cx_negate),
    .iag_ve_cy_negate                  (ag_ve_cy_negate),
    .iag_ve_cz_negate                  (ag_ve_cz_negate),
    .iag_ve_cw_negate                  (ag_ve_cw_negate),
    .iag_ve_bcc_flat_tst               (ag_ve_bcc_flat_tst),
    .iag_ve_out_mem_sel                (ag_ve_out_mem_sel),
    .iag_ve_out_addr                   (ag_ve_out_addr),
    .iag_ve_out_we                     (ag_ve_out_we),
    .iag_ve_accum_sel                  (ag_ve_accum_sel),
    .iag_ve_pre_accum_we               (ag_ve_pre_accum_we),
    .ovmb_ve_tst_rtn_stat              (vmb_ve_tst_rtn_stat),
  .ove_waddr                  (ve_waddr),
  .ove_veoc_vector_back_we    (ve_veoc_vector_back_we),
  .ove_cliptemp_vector_we     (ve_cliptemp_vector_we),
  .ove_inverse_we             (ve_inverse_we),
  .ove_wdata                  (ve_wdata),
  .isclk                      (sclk_pa)
 );

pa_vte
upa_vte
 (
  .rbiu_vte_wdata             (rbiu_wdata),
  .rbiu_vte_waddr             (rbiu_waddr),
  .rbiu_vte_raddr             (rbiu_raddr),
  .rbiu_vte_we                (rbiu_we),
  .rbiu_vte_re                (rbiu_re),
  .rbiu_vte_cpy               (rbiu_vte_cpy),
  .rbiu_vte_xscale_sel        (rbiu_vte_xscale_sel),
  .rbiu_vte_xoffset_sel       (rbiu_vte_xoffset_sel),
  .rbiu_vte_yscale_sel        (rbiu_vte_yscale_sel),
  .rbiu_vte_yoffset_sel       (rbiu_vte_yoffset_sel),
  .rbiu_vte_zscale_sel        (rbiu_vte_zscale_sel),
  .rbiu_vte_zoffset_sel       (rbiu_vte_zoffset_sel),
  .rbiu_vte_cntl_sel          (rbiu_vte_cntl_sel),
  .rbiu_vte_vtx_cntl_sel      (rbiu_vte_vtx_cntl_sel),
```

```
  .rbiu_vte_window_offset_sel     (rbiu_vte_window_offset_sel),
  .rbiu_vte_window_offset_en_sel  (rbiu_vte_window_offset_en_sel),
  .rbiu_vte_persp_corr_dis_sel    (rbiu_vte_persp_corr_dis_sel),
  .ag_vte_ix                  (ag_ve_in_a0),
  .ag_vte_jy                  (ag_ve_in_a1),
  .ag_vte_z                   (ag_ve_in_a2),
  .ag_vte_w                   (ag_ve_in_a3),
  .ag_vte_negate_ix           (ag_ve_ax_negate),
  .ag_vte_negate_jy           (ag_ve_ay_negate),
  .ag_vte_negate_z            (ag_ve_az_negate),
  .ag_vte_negate_w            (ag_ve_aw_negate),
  .ag_vte_vertex_store_indx   (ag_vte_vertex_store_indx),
  .ag_vte_opcode              (ag_vte_opcode),
  .ag_vte_st_indx             (ag_vte_st_indx),
  .rcp_d                      (vcm_rcp_rei_d),
  .rcp_a                      (vcm_rcp_rei_a),
  .rcp_xfc                    (vcm_rcp_rei_xfc),
  .vte_rcp_d                  (vte_rcp_d),
  .vte_rbiu_rdata             (vte_rbiu_rdata),
  .vte_rcp_rts                (vte_rcp_rts),
  .vte_d                      (vte_d),
  .vte_opcode                 (vte_opcode),
  .vte_vertex_store_indx      (vte_vertex_store_indx),
  .vte_busy                   (vte_busy),
  .srst                       (srst),
```

```
  .sclk                       (sclk_pa)
 );

pa_cl_rei
upa_cl_rei
 (
  .isc_wdata0                 (ve_wdata[31:0]),
  .isc_rei_we                 (ve_inverse_we),

  .ovmc_rei_rcp_d             (vmc_rei_rcp_d),
  .ovmc_rei_rcp_rts           (vmc_rei_rcp_rts),
  .ivcm_rcp_rei_rtr           (vcm_rcp_rei_rtr),

  .ivcm_rcp_rei_d             (vcm_rcp_rei_d),
  .ivcm_rcp_rei_a             ({vcm_rcp_rei_a,1'b0}),
  .ivcm_rcp_rei_xfc           (vcm_rcp_rei_xfc),

  .ovmb_cl_rei_r0vld          (vmb_cl_rei_r0vld),
  .ovmb_cl_rei_r1vld          (vmb_cl_rei_r1vld),
  .ivbm_cl_rei_r0r1clr        (cl_rei_clear_result),

  .orei_sc_r0data             (rei_sc_r0data),
  .orei_sc_r1data             (rei_sc_r1data),

  .isrst                      (srst),
```

```
  .isclk                      (sclk_pa)
 );

pa_cl_rcpeng
upa_cl_rcpeng
 (
  .ireset                     (srst),

  .iport1_rts                 (vte_rcp_rts),
  .iport1_sign                (vte_rcp_d[31]),
  .iport1_exp                 (vte_rcp_d[30:23]),
  .iport1_mant                (vte_rcp_d[22:0]),

  .iport2_rts                 (vmc_rei_rcp_rts),
  .iport2_sign                (vmc_rei_rcp_d[31]),
  .iport2_exp                 (vmc_rei_rcp_d[30:23]),
  .iport2_mant                (vmc_rei_rcp_d[22:0]),

  .oport2_rtr                 (vcm_rcp_rei_rtr),

  .oout_xfc                   (vcm_rcp_rei_xfc),
  .oout_recip_sign            (vcm_rcp_rei_d[31]),
  .oout_recip_exp             (vcm_rcp_rei_d[30:23]),
  .oout_recip_mant            (vcm_rcp_rei_d[22:0]),
  .oout_sel                   (vcm_rcp_rei_a),

  .orcp_busy                        (),

  .idbug_muxsel                     (6'h0),
  .odbug_data_out                   (),

  .isclk                            (sclk_pa)
 );


// shader export interface and clip code generator
pa_sxifccg
upa_sxifccg
 (
  ////////////////////////////////////////////////////////////////////////////
  // inputs
  ////////////////////////////////////////////////////////////////////////////
  // clock and reset
  .clk                              (sclk_pa),
  .reset                            (srst),

  // ati_state_storage (sxif)
  .isxif_st_w_data                  (rbiu_wdata[26:20]),
  .isxif_st_w_addr                  (rbiu_waddr),
  .isxif_st_we                      (rbiu_we),
```

```
  .isxif_st_r_addr                  (rbiu_raddr),
  .isxif_st_re                      (rbiu_re),
  .isxif_st_sel                     (rbiu_ccg_expcntmd_sel),
  .isxif_st_cpy                     (rbiu_ccg_cpy),

  // vgt_to_ccgen fifo
  .ivgt_to_ccgen_fifo_write         (VGT_PA_clip_v_send_q),
  .ivgt_to_ccgen_fifo_active_verts  (VGT_PA_clip_v_vec_size_q),
  .ivgt_to_ccgen_fifo_state_var_indx (VGT_PA_clip_v_state_q),

  // sx0 receive fifo
  .isx0_receive_fifo_write          (SX0_PA_input_data_write),
  .isx0_receive_fifo_wrdata         (SX0_PA_input_data_wrdata),

  // sx1 receive fifo
  .isx1_receive_fifo_write          (SX1_PA_input_data_write),
  .isx1_receive_fifo_wrdata         (SX1_PA_input_data_wrdata),

  // ccg state
  .iccg_state0                      (cl_ccg_state0),
  .iccg_state1                      (cl_ccg_state1),
  .iccg_state2                      (cl_ccg_state2),
  .iccg_state3                      (cl_ccg_state3),
  .iccg_state4                      (cl_ccg_state4),
  .iccg_state5                      (cl_ccg_state5),
```

```
  .iccg_state6                      (cl_ccg_state6),
  .iccg_state7                      (cl_ccg_state7),

  // ccgen_to_clipcc/clip
  .ioutsm_clr_orig_vertices         (cl_ccg_outsm_clr_orig_vertices),
  .iccgen_to_clipcc_fifo_full       (cl_ccg_ccgen_to_clipcc_fifo_full),

  // arbiter
  .iarb_to_ccgen_xfc                (arb_ccg_xfc),


  ////////////////////////////////////////////////////////////////////////////
  // outputs
  ////////////////////////////////////////////////////////////////////////////
  // state
  .osxif_st_r_data                  (ccg_rbiu_rdata_26_downto_20),

  // state to clipper
  .osxif_state0                     (sxif_state0),
  .osxif_state1                     (sxif_state1),
  .osxif_state2                     (sxif_state2),
  .osxif_state3                     (sxif_state3),
  .osxif_state4                     (sxif_state4),
  .osxif_state5                     (sxif_state5),
  .osxif_state6                     (sxif_state6),
```

```
  .osxif_state7                     (sxif_state7),

  // vgt_to_ccgen fifo
  .ovgt_to_ccgen_fifo_notfull       (PA_VGT_clip_v_d),

  // sx0, request
  .opa_to_sx0_req                   (u0_PA_SX_req_d),
  .opa_to_sx0_sp_id                 (u0_PA_SX_sp_id_d),
  .opa_to_sx0_offset                (u0_PA_SX_offset_d),
  .opa_to_sx0_aux                   (u0_PA_SX_aux_d),
  .opa_to_sx0_last                  (u0_PA_SX_last_d),

  // sx1, request
  .opa_to_sx1_req                   (u1_PA_SX_req_d),
  .opa_to_sx1_sp_id                 (u1_PA_SX_sp_id_d),
  .opa_to_sx1_offset                (u1_PA_SX_offset_d),
  .opa_to_sx1_aux                   (u1_PA_SX_aux_d),
  .opa_to_sx1_last                  (u1_PA_SX_last_d),

  // position memory
  .oposition_write                  (ccg_ag_pos_mem_we),
  .oposition_wraddr                 (ccg_ag_pos_mem_wraddr),
  .oposition_wrdata                 (ccg_ag_pos_pntsz_mem_wrdata),

  // point memory
  .opoint_write                  (ccg_ag_pntsz_mem_we),
  .opoint_wraddr                 (ccg_ag_pntsz_mem_wraddr),
  .opoint_wrdata                 (ccg_ag_pntsz_mem_wrdata),

  // ccgen_to_clipcc/clip
  .occgen_to_clipcc_data         (ccg_cl_wrdata),
  .occgen_to_clipcc_write        (ccg_cl_write),

  // arbiter
  .occgen_to_arb_data            (ccg_arb_data)
 );

// temp connections
assign ag_cl_pntsz_mem_blocked = 'h0;
assign ag_cl_pntsz_mem_rdata = 'h0;

// clipper
pa_clipper
upa_clipper
 (
  ////////////////////////////////////////////////////////////////////
  // inputs
  ////////////////////////////////////////////////////////////////////
  // clock and reset
  .clk                           (sclk_pa),
```

```
  .reset                         (srst),

  // ati_state_storage
  .ist_w_data                    (rbiu_wdata),
  .ist_w_addr                    (rbiu_waddr),
  .ist_we                        (rbiu_we),
  .ist_r_addr                    (rbiu_raddr),
  .ist_re                        (rbiu_re),
  .ist_cpy                       (rbiu_cl_cpy),
  .ist_sel                       (rbiu_cl_dx_clip_sp_def_sel),

  // state from sxif
  .isxif_state0                  (sxif_state0),
  .isxif_state1                  (sxif_state1),
  .isxif_state2                  (sxif_state2),
  .isxif_state3                  (sxif_state3),
  .isxif_state4                  (sxif_state4),
  .isxif_state5                  (sxif_state5),
  .isxif_state6                  (sxif_state6),
  .isxif_state7                  (sxif_state7),

  // ccg
  .iccgen_to_clipcc_fifo_write   (ccg_cl_write),
  .iccgen_to_clipcc_fifo_wrdata  (ccg_cl_wrdata),

```

```
  // vgt_to_clips
  .ivgt_to_clips_fifo_write                (VGT_PA_clip_s_send_q),
  .ivgt_to_clips_fifo_event                (VGT_PA_clip_s_event_q),
  .ivgt_to_clips_fifo_prim_type            (VGT_PA_clip_s_type_q),
  .ivgt_to_clips_fifo_state_var_indx       (VGT_PA_clip_s_state_q),

  // vgt_to_clipp
  .ivgt_to_clipp_fifo_write                (VGT_PA_clip_p_send_q),
  .ivgt_to_clipp_fifo_null_primitive       (VGT_PA_clip_p_null_prim_q),
  .ivgt_to_clipp_fifo_first_prim_of_slot   (VGT_PA_clip_p_new_vtx_vect_q),
  .ivgt_to_clipp_fifo_deallocate_slot      (VGT_PA_clip_p_dealloc_q),
  .ivgt_to_clipp_fifo_end_of_packet        (VGT_PA_clip_p_eop_q),
  .ivgt_to_clipp_fifo_edge_flag            (VGT_PA_clip_p_edge_flags_q),
  .ivgt_to_clipp_fifo_vertex_store_indx_0  (VGT_PA_clip_p_indx0_q),
  .ivgt_to_clipp_fifo_vertex_store_indx_1  (VGT_PA_clip_p_indx1_q),
  .ivgt_to_clipp_fifo_vertex_store_indx_2  (VGT_PA_clip_p_indx2_q),

  // reciprocal engine
  .iinv_to_clip_data_valid_0               (vmb_cl_rei_r0vld),
  .iinv_to_clip_data_valid_1               (vmb_cl_rei_r1vld),

  //arbiter
  .ive_xfc                                 (arb_cl_xfc),

  // clip_to_ga
```

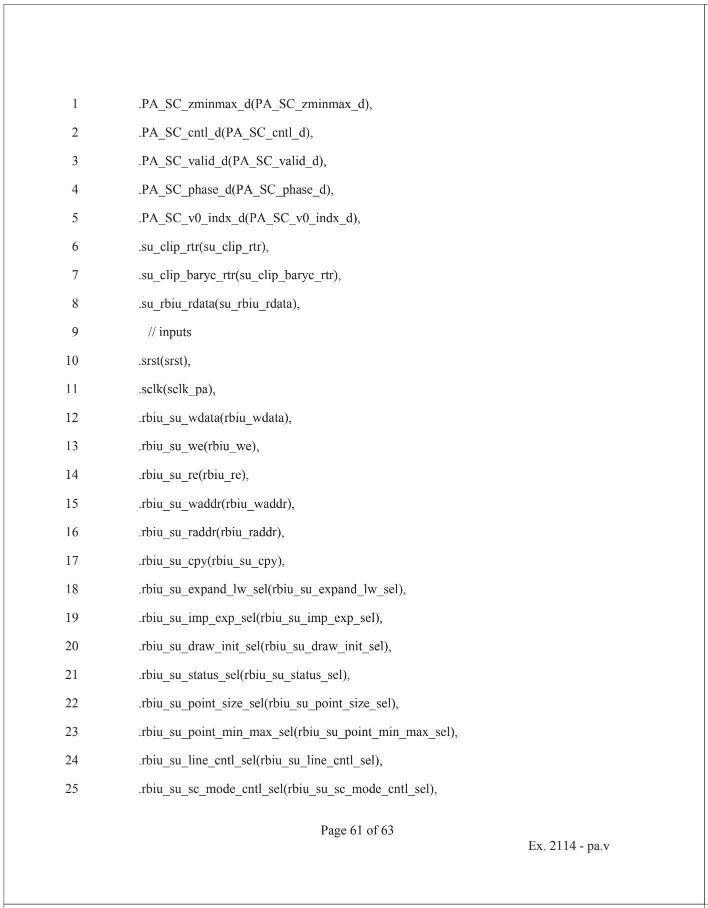
```
  .iclip_to_ga_fifo_notfull          (su_clip_rtr),
  .iclip_ga_bc_fifo_notfull          (su_clip_baryc_rtr),

  // from vte
  .ivte_out_vertex_store_indx        (vte_vertex_store_indx),
  .ivte_out_opcode                   (vte_opcode),
  .ivte_out_vector_data              (vte_d),

  // from ag-ve delay
  .iag_ve_out_vertex_store_indx      (ag_cl_vertex_store_indx[3:0]),
  .iag_ve_out_valid_bit_set          (ag_cl_valid_bit_set),
  .iag_ve_out_user_clip_indx         (ag_cl_user_clip_indx),
  .iag_ve_out_vv_cc_test             (ag_cl_vv_cc_test),
  .iag_ve_out_ucp_cc_test            (ag_cl_ucp_cc_test),
  .iag_ve_out_bcc_cc_test            (ag_cl_bcc_cc_test),
  .iag_ve_out_ps_ucp_cc_test         (ag_cl_ps_ucp_cc_test),
  .iag_ve_out_ps_engh_test           (ag_cl_ps_engh_test),

  // from vector engine
  .ive_out_test_rtn_status           (vmb_ve_tst_rtn_stat),

  // from ag point size memory
  .iag_to_clip_point_size            (ag_to_clip_point_size),

  ////////////////////////////////////////////////////////////////////
```

```
// outputs
//////////////////////////////////////////////////////////////////
// state
.ost_r_data                    (cl_rbiu_rdata),

// vgt_to_clips
.ovgt_to_clips_fifo_notfull        (PA_VGT_clip_s_d),

// vgt_to_clipp
.ovgt_to_clipp_fifo_notfull        (PA_VGT_clip_p_d),

// to ccg
.occgen_to_clipcc_fifo_full        (cl_ccg_ccgen_to_clipcc_fifo_full),
.occg_state0                   (cl_ccg_state0),
.occg_state1                   (cl_ccg_state1),
.occg_state2                   (cl_ccg_state2),
.occg_state3                   (cl_ccg_state3),
.occg_state4                   (cl_ccg_state4),
.occg_state5                   (cl_ccg_state5),
.occg_state6                   (cl_ccg_state6),
.occg_state7                   (cl_ccg_state7),

// to ccgen
.ooutsm_clr_orig_vertices          (cl_ccg_outsm_clr_orig_vertices),

```

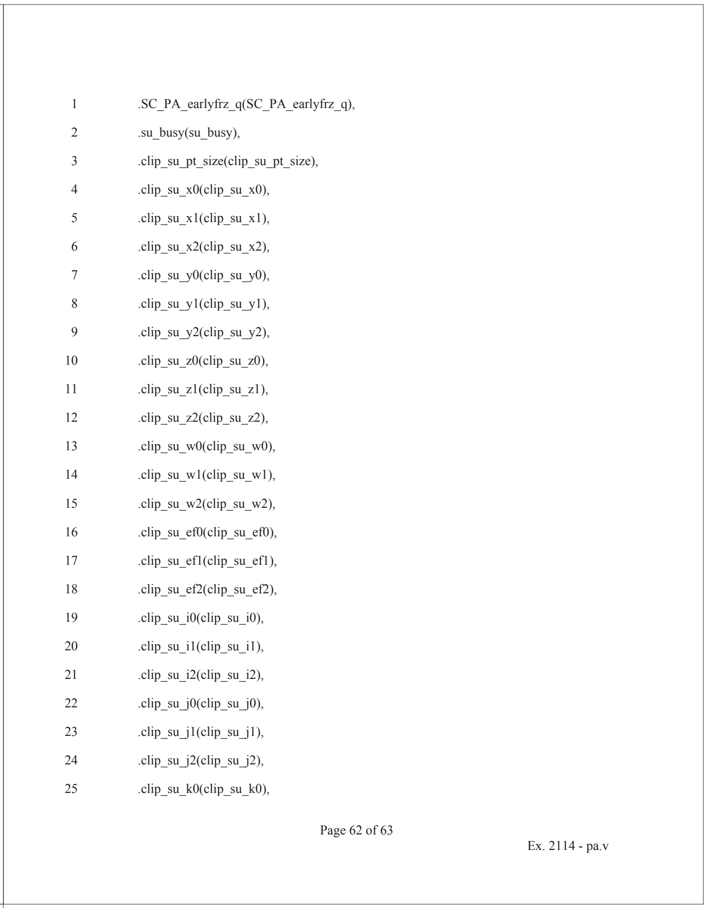
```
 // to clip_to_ga fifo
 .oclip_to_ga_fifo_write            (clip_su_rts),
 .oclip_to_ga_point_size            (clip_su_pt_size),
 .oclip_to_ga_position_x_0          (clip_su_x0),
 .oclip_to_ga_position_x_1          (clip_su_x1),
 .oclip_to_ga_position_x_2          (clip_su_x2),
 .oclip_to_ga_position_y_0          (clip_su_y0),
 .oclip_to_ga_position_y_1          (clip_su_y1),
 .oclip_to_ga_position_y_2          (clip_su_y2),
 .oclip_to_ga_position_z_0          (clip_su_z0),
 .oclip_to_ga_position_z_1          (clip_su_z1),
 .oclip_to_ga_position_z_2          (clip_su_z2),
 .oclip_to_ga_position_w_0          (clip_su_w0),
 .oclip_to_ga_position_w_1          (clip_su_w1),
 .oclip_to_ga_position_w_2          (clip_su_w2),
 .oclip_to_ga_edge_flag_0           (clip_su_ef0),
 .oclip_to_ga_edge_flag_1           (clip_su_ef1),
 .oclip_to_ga_edge_flag_2           (clip_su_ef2),
 .oclip_to_ga_param_cache_indx_0        (clip_su_attr_indx0),
 .oclip_to_ga_param_cache_indx_1        (clip_su_attr_indx1),
 .oclip_to_ga_param_cache_indx_2        (clip_su_attr_indx2),
 .oclip_to_ga_prim_type             (clip_su_type),
 .oclip_to_ga_state_var_indx        (clip_su_st_indx),
 .oclip_to_ga_deallocate_slot       (clip_su_dealloc_slot),
 .oclip_to_ga_null_primitive        (clip_su_null_prim),
```

```
.oclip_to_ga_clipped_prim          (clip_su_clipped),
.oclip_to_ga_first_prim_of_slot      (clip_su_fpov),
.oclip_to_ga_end_of_packet          (clip_su_eop),
.oclip_to_ga_event              (clip_su_event),
.oclip_to_ga_event_id             (clip_su_event_id),

.oclip_ga_bc_fifo_write            (clip_su_baryc_rts),
.oclip_ga_bc_baryc_coord_x_0         (clip_su_i0),
.oclip_ga_bc_baryc_coord_x_1         (clip_su_i1),
.oclip_ga_bc_baryc_coord_x_2         (clip_su_i2),
.oclip_ga_bc_baryc_coord_y_0         (clip_su_j0),
.oclip_ga_bc_baryc_coord_y_1         (clip_su_j1),
.oclip_ga_bc_baryc_coord_y_2         (clip_su_j2),
.oclip_ga_bc_baryc_coord_z_0         (clip_su_k0),
.oclip_ga_bc_baryc_coord_z_1         (clip_su_k1),
.oclip_ga_bc_baryc_coord_z_2         (clip_su_k2),

// arbiter
.oclip_to_arb_data_ve_valid          (cl_arb_ve_valid),
.oclip_to_arb_data              (cl_arb_data),

// reciprocal engine
.oclip_to_inv_data_reset_valids      (cl_rei_clear_result),

// ag point size memory
```

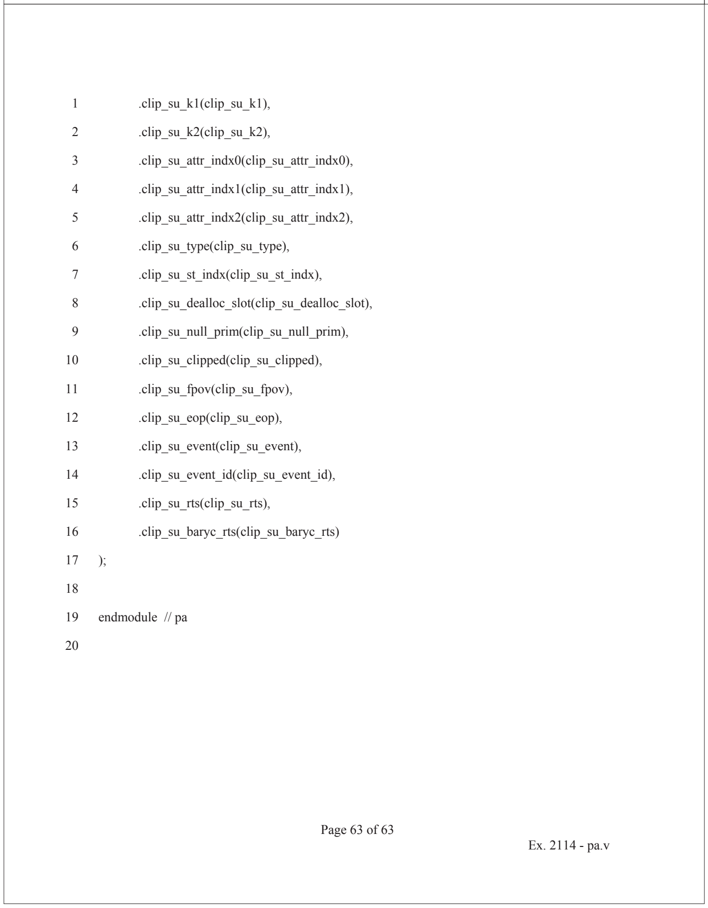
```
 .oclip_to_ag_point_buf_re          (clip_to_ag_point_buf_re),
 .oclip_to_ag_point_buf_raddr         (clip_to_ag_point_buf_raddr),

 // busy
 .oclipper_busy                 (clipper_busy)
);

// temp connections until clipper has these
//assign clip_su_k0 = 32'h00000000;
//assign clip_su_k1 = 32'h00000000;
//assign clip_su_k2 = 32'h00000000;
//assign clip_su_baryc_rts = 1'b0;

// setup unit
pa_su upa_su (
   // outputs

        .PA_SC_p0_d(PA_SC_p0_d),
        .PA_SC_p1_d(PA_SC_p1_d),
        .PA_SC_p2_d(PA_SC_p2_d),
        .PA_SC_p3_d(PA_SC_p3_d),
        .PA_SC_p4_d(PA_SC_p4_d),
        .PA_SC_xy0_d(PA_SC_xy0_d),
        .PA_SC_xy1_d(PA_SC_xy1_d),
        .PA_SC_xy2_d(PA_SC_xy2_d),
```



```
    .PA_SC_zminmax_d(PA_SC_zminmax_d),
    .PA_SC_cntl_d(PA_SC_cntl_d),
    .PA_SC_valid_d(PA_SC_valid_d),
    .PA_SC_phase_d(PA_SC_phase_d),
    .PA_SC_v0_indx_d(PA_SC_v0_indx_d),
    .su_clip_rtr(su_clip_rtr),
    .su_clip_baryc_rtr(su_clip_baryc_rtr),
    .su_rbiu_rdata(su_rbiu_rdata),
      // inputs
    .srst(srst),
    .sclk(sclk_pa),
    .rbiu_su_wdata(rbiu_wdata),
    .rbiu_su_we(rbiu_we),
    .rbiu_su_re(rbiu_re),
    .rbiu_su_waddr(rbiu_waddr),
    .rbiu_su_raddr(rbiu_raddr),
    .rbiu_su_cpy(rbiu_su_cpy),
    .rbiu_su_expand_lw_sel(rbiu_su_expand_lw_sel),
    .rbiu_su_imp_exp_sel(rbiu_su_imp_exp_sel),
    .rbiu_su_draw_init_sel(rbiu_su_draw_init_sel),
    .rbiu_su_status_sel(rbiu_su_status_sel),
    .rbiu_su_point_size_sel(rbiu_su_point_size_sel),
    .rbiu_su_point_min_max_sel(rbiu_su_point_min_max_sel),
    .rbiu_su_line_cntl_sel(rbiu_su_line_cntl_sel),
    .rbiu_su_sc_mode_cntl_sel(rbiu_su_sc_mode_cntl_sel),
```



```
    .SC_PA_earlyfrz_q(SC_PA_earlyfrz_q),
    .su_busy(su_busy),
    .clip_su_pt_size(clip_su_pt_size),
    .clip_su_x0(clip_su_x0),
    .clip_su_x1(clip_su_x1),
    .clip_su_x2(clip_su_x2),
    .clip_su_y0(clip_su_y0),
    .clip_su_y1(clip_su_y1),
    .clip_su_y2(clip_su_y2),
    .clip_su_z0(clip_su_z0),
    .clip_su_z1(clip_su_z1),
    .clip_su_z2(clip_su_z2),
    .clip_su_w0(clip_su_w0),
    .clip_su_w1(clip_su_w1),
    .clip_su_w2(clip_su_w2),
    .clip_su_ef0(clip_su_ef0),
    .clip_su_ef1(clip_su_ef1),
    .clip_su_ef2(clip_su_ef2),
    .clip_su_i0(clip_su_i0),
    .clip_su_i1(clip_su_i1),
    .clip_su_i2(clip_su_i2),
    .clip_su_j0(clip_su_j0),
    .clip_su_j1(clip_su_j1),
    .clip_su_j2(clip_su_j2),
    .clip_su_k0(clip_su_k0),
```



```
    .clip_su_k1(clip_su_k1),
    .clip_su_k2(clip_su_k2),
    .clip_su_attr_indx0(clip_su_attr_indx0),
    .clip_su_attr_indx1(clip_su_attr_indx1),
    .clip_su_attr_indx2(clip_su_attr_indx2),
    .clip_su_type(clip_su_type),
    .clip_su_st_indx(clip_su_st_indx),
    .clip_su_dealloc_slot(clip_su_dealloc_slot),
    .clip_su_null_prim(clip_su_null_prim),
    .clip_su_clipped(clip_su_clipped),
    .clip_su_fpov(clip_su_fpov),
    .clip_su_eop(clip_su_eop),
    .clip_su_event(clip_su_event),
    .clip_su_event_id(clip_su_event_id),
    .clip_su_rts(clip_su_rts),
    .clip_su_baryc_rts(clip_su_baryc_rts)
);

endmodule  // pa

```

```
`include "header.v"
//---------------------------------------------------------------------------
//
// $Id: //depot/r400/devel/parts_lib/src/gfx/pa/pa_ag.v#19 $
//
// $Change: 43657 $
//
//
// Notes:    This file is the pa_ag address generator
//
// Copyright: Trade secret of ATI Technologies, Inc.
//            © Copyright 2001-2002, ATI Technologies, Inc., (unpublished)
//
//        All rights reserved.  This notice is intended as a precaution against
//        inadvertent publication and does not imply publication or any waiver
//        of confidentiality.  The year included in the foregoing notice is the
//        year of creation of the work.
//
//---------------------------------------------------------------------------


module pa_ag (
  sclk, srst,

  rbiu_ag_dx_clip_sp_def_sel,
```

```
  rbiu_ag_ucp0_sel,
  rbiu_ag_ucp1_sel,
  rbiu_ag_ucp2_sel,
  rbiu_ag_ucp3_sel,
  rbiu_ag_ucp4_sel,
  rbiu_ag_ucp5_sel,
  rbiu_ag_gb_sel,
  rbiu_ag_pntsz_sel,

  rbiu_we,
  rbiu_wa,
  rbiu_wd,
  rbiu_cpy,
  rbiu_re,
  ag_rbiu_rd,

  ccg_to_arb_data,
  arb_ccg_xfc,

  clip_ve_valid,
  clip_to_arb_data,
  arb_clip_xfc,

  clip_to_ag_point_buf_re,
  clip_to_ag_point_buf_raddr,
```

```
  ag_to_clip_point_size,

  pos_pntsz_ag_mem_data,
  pos_mem_waddr,
  pntsz_mem_waddr,
  pos_mem_we,
  pntsz_mem_we,

  inv_ret_sc_data,

  ve_cliptemp_vector_we,
  ve_veoc_vector_back_we,
  ve_waddr,
  ve_wdata,

  agve_dly_valid_op,
  agve_dly_vertex_store_indx,
  agve_dly_valid_bit_set,
  agve_dly_user_clip_indx,
  agve_dly_vv_cc_test,
  agve_dly_ucp_cc_test,
  agve_dly_bcc_cc_test,
  agve_dly_ps_ucp_cc_test,
  agve_dly_ps_engh_test,

```

```
//  ag_vte_ix,        ag_vte_jy,
//  ag_vte_z,         ag_vte_w,
//  ag_vte_negate_ix, ag_vte_negate_jy,
//  ag_vte_negate_z,  ag_vte_negate_w,
  ag_vte_opcode,
  ag_vte_st_indx,
  ag_vte_vertex_store_indx,

  ag_ve_opcode,
  ag_ve_in_a0,       ag_ve_in_a1,       ag_ve_in_a2,     ag_ve_in_a3,
  ag_ve_in_b0,       ag_ve_in_b1,       ag_ve_in_b2,     ag_ve_in_b3,
  ag_ve_a_is_wwww,    ag_ve_broadcast_x,
  ag_ve_abs_a,       ag_ve_abs_b,       ag_ve_abs_c,
  ag_ve_ax_negate,    ag_ve_ay_negate,   ag_ve_az_negate, ag_ve_aw_negate,
  ag_ve_bx_negate,    ag_ve_by_negate,   ag_ve_bz_negate, ag_ve_bw_negate,
  ag_ve_cx_negate,    ag_ve_cy_negate,   ag_ve_cz_negate, ag_ve_cw_negate,
  ag_ve_bcc_flat_tst, ag_ve_out_mem_sel, ag_ve_out_addr,
  ag_ve_out_we,       ag_ve_accum_sel,   ag_ve_pre_accum_we
);

`include "pa_ag_pkg.v"

// ************  PARAMETERS   **************************
parameter       u0_clptmp_ADDR_WIDTH = 6;
parameter       u0_clptmp_DATA_WIDTH = 32;
parameter        u0_clptmp_WORDS      = 35;
parameter        u0_clptmp_DEBUG      = 0;
parameter        u1_clptmp_ADDR_WIDTH = 6;
parameter        u1_clptmp_DATA_WIDTH = 32;
parameter        u1_clptmp_WORDS      = 35;
parameter        u1_clptmp_DEBUG      = 0;
parameter        u2_clptmp_ADDR_WIDTH = 6;
parameter        u2_clptmp_DATA_WIDTH = 32;
parameter        u2_clptmp_WORDS      = 35;
parameter        u2_clptmp_DEBUG      = 0;
parameter        u3_clptmp_ADDR_WIDTH = 6;
parameter        u3_clptmp_DATA_WIDTH = 32;
parameter        u3_clptmp_WORDS      = 35;
parameter        u3_clptmp_DEBUG      = 0;
parameter        u_pntsz_ADDR_WIDTH = 6;
parameter        u_pntsz_DATA_WIDTH = 32;
parameter        u_pntsz_WORDS      = 64;
parameter        u_pntsz_DEBUG      = 0;
parameter        u_pos_ADDR_WIDTH = 6;
parameter        u_pos_DATA_WIDTH = 128;
parameter        u_pos_WORDS      = 64;
parameter        u_pos_DEBUG      = 0;
parameter        u0_stve_ADDR_WIDTH = 6;
parameter        u0_stve_DATA_WIDTH = 32;
parameter        u0_stve_WORDS      = 64;
```

```
parameter        u0_stve_DEBUG      = 0;
parameter        u1_stve_ADDR_WIDTH = 6;
parameter        u1_stve_DATA_WIDTH = 32;
parameter        u1_stve_WORDS      = 64;
parameter        u1_stve_DEBUG      = 0;
parameter        u2_stve_ADDR_WIDTH = 6;
parameter        u2_stve_DATA_WIDTH = 32;
parameter        u2_stve_WORDS      = 64;
parameter        u2_stve_DEBUG      = 0;
parameter        u3_stve_ADDR_WIDTH = 6;
parameter        u3_stve_DATA_WIDTH = 32;
parameter        u3_stve_WORDS      = 64;
parameter        u3_stve_DEBUG      = 0;

// ************  System stuff ***************************
input        sclk;
input        srst;

// ************  State interface ***********************
input          rbiu_ag_dx_clip_sp_def_sel;
input    [3:0] rbiu_ag_ucp0_sel;
input    [3:0] rbiu_ag_ucp1_sel;
input    [3:0] rbiu_ag_ucp2_sel;
input    [3:0] rbiu_ag_ucp3_sel;
input    [3:0] rbiu_ag_ucp4_sel;
```

```
input    [3:0] rbiu_ag_ucp5_sel;

input    [3:0] rbiu_ag_gb_sel;
input    [3:0] rbiu_ag_pntsz_sel;

input          rbiu_we;
input    [2:0] rbiu_wa;
input   [31:0] rbiu_wd;
input          rbiu_cpy;
input          rbiu_re;
output  [31:0] ag_rbiu_rd;


// ************  interface with the ccg ****************
input   [15:0] ccg_to_arb_data;  //Data from the ccg
output         arb_ccg_xfc;

// ************  interface with the clipper ************
input           clip_ve_valid;
input   [29:0]  clip_to_arb_data;
output          arb_clip_xfc;

input clip_to_ag_point_buf_re;
input [5:0] clip_to_ag_point_buf_raddr;
output [31:0] ag_to_clip_point_size;
```

```

// ********* position and pntsz mem ********************
input  [127:0] pos_pntsz_ag_mem_data;
input    [5:0] pos_mem_waddr;
input    [5:0] pntsz_mem_waddr;
input          pos_mem_we;
input          pntsz_mem_we;

// *********** interface with reciprocal engine ********
input [63:0] inv_ret_sc_data;

// ********** interface with ve output *****************
input    [3:0] ve_cliptemp_vector_we;
input    [3:0] ve_veoc_vector_back_we;
input    [5:0] ve_waddr;
input  [127:0] ve_wdata;

// ************  Ouputs to the pa_cl_ve*****************
output   [2:0] ag_ve_opcode;
output  [31:0] ag_ve_in_a0;
output  [31:0] ag_ve_in_a1;
output  [31:0] ag_ve_in_a2;
output  [31:0] ag_ve_in_a3;
output  [31:0] ag_ve_in_b0;
output  [31:0] ag_ve_in_b1;
output  [31:0] ag_ve_in_b2;
output  [31:0] ag_ve_in_b3;
output         ag_ve_a_is_wwww;
output         ag_ve_broadcast_x;
output         ag_ve_abs_a;
output         ag_ve_abs_b;
output         ag_ve_abs_c;
output         ag_ve_ax_negate;
output         ag_ve_ay_negate;
output         ag_ve_az_negate;
output         ag_ve_aw_negate;
output         ag_ve_bx_negate;
output         ag_ve_by_negate;
output         ag_ve_bz_negate;
output         ag_ve_bw_negate;
output         ag_ve_cx_negate;
output         ag_ve_cy_negate;
output         ag_ve_cz_negate;
output         ag_ve_cw_negate;
output         ag_ve_bcc_flat_tst;
output   [2:0] ag_ve_out_mem_sel;
output   [5:0] ag_ve_out_addr;
output   [3:0] ag_ve_out_we;
output         ag_ve_accum_sel;
output   [3:0] ag_ve_pre_accum_we;
```

```

// ************  Ouputs from the delay pipe **********************
output         agve_dly_valid_op;
output   [5:0] agve_dly_vertex_store_indx;
output   [1:0] agve_dly_valid_bit_set;
output   [3:0] agve_dly_user_clip_indx;
output         agve_dly_vv_cc_test;
output         agve_dly_ucp_cc_test;
output         agve_dly_bcc_cc_test;
output         agve_dly_ps_ucp_cc_test;
output         agve_dly_ps_engh_test;

// ************  Ouputs to the pa_vte**********************
//output  [31:0] ag_vte_ix;      //   i/x Vector Data
//output  [31:0] ag_vte_jy;      //   j/y Vector Data
//output  [31:0] ag_vte_w;       //   w Vector Data
//output  [31:0] ag_vte_z;       //   z Vector Data
//output   [0:0] ag_vte_negate_ix; //   Negate i/x Vector Data
//output   [0:0] ag_vte_negate_jy; //   Negated j/y Vector Data
//output   [0:0] ag_vte_negate_w;  //   Negate w Vector Data
//output   [0:0] ag_vte_negate_z;  //   Nagate z Vector Data
output   [1:0] ag_vte_vertex_store_indx;
output   [2:0] ag_vte_opcode;    //   Opcode
output   [2:0] ag_vte_st_indx;   //   Context ID

```

```
//*****************************************************************
//   Declare and map internal ccg interface signal names
reg       ccg_ve_cc_valid;
reg       ccg_ve_valid;
reg [1:0] ccg_sm_state_indx;
reg [2:0] ccg_state_var_indx;
reg [5:0] ccg_vertex_store_indx;
reg [2:0] ccg_ve_ucp_indx;

//   Declare and map internal clipper interface signal names
reg [3:0] clip_plane_indx;
reg [5:0] clip_dst_vertex_indx;
reg [6:0] clip_src_vertex_indx;
reg       clip_src_vertex_type;
reg       clip_ve_ucp_valid;
reg [6:0] clip_sm_state_indx;
reg [2:0] clip_state_var_indx;

//  ve_veoc_vector_back read port signal declarations
reg           ve_veoc_vector_back_re;
reg           ve_veoc_vector_back_re_r1;
reg     [5:0] ve_veoc_vector_back_raddr;
wire  [127:0] ve_veoc_vector_back_rdata;
reg   [127:0] ve_veoc_vector_back_rdata_r2;

```

```
//  pos read port signal declarations
reg           pos_re;
reg           pos_re_r1;
reg     [5:0] pos_raddr;
wire  [127:0] pos_rdata;
reg   [127:0] pos_rdata_r2;

//  pntsz read port signal declarations
reg           pntsz_re;
reg           pntsz_re_r1;
reg     [5:0] pntsz_raddr;
wire   [31:0] pntsz_rdata;
reg    [31:0] pntsz_rdata_r2;

//  stve read port signal declarations
reg           stve_re_r0;
reg           stve_re_r1;
reg     [5:0] stve_raddr;
wire  [127:0] stve_rdata;
reg   [127:0] stve_rdata_r2;
reg           nxt_stve_re;
reg     [5:3] nxt_state_type;
reg     [5:3] state_type_r0;

//  stve write port signal declarations
```

```
reg    [5:0] stve_wa;
reg    [3:0] stve_we;

// ag signal declarations
reg    [2:0] nxt_arb_state_var_indx;
reg    [2:0] arb_state_var_indx_r0;
reg    [2:0] arb_ucp_indx;
reg  [127:0] ve_cliptemp_vec;

// State storage signal declarations
wire dx_clip_space_def;
wire agrd_dx_clip_space_def;

wire [2:0] ucp0_rd_off;
wire nxt_ucp0_write_after_cpy;
reg  ucp0_write_after_cpy;
wire [2:0] nxt_ucp0_write_ptr;
reg  [2:0] ucp0_write_ptr;
wire [2:0] agrd_ucp0_write_ptr;

wire [2:0] ucp1_rd_off;
wire nxt_ucp1_write_after_cpy;
reg  ucp1_write_after_cpy;
wire [2:0] nxt_ucp1_write_ptr;
reg  [2:0] ucp1_write_ptr;
```

```
wire [2:0] agrd_ucp1_write_ptr;

wire [2:0] ucp2_rd_off;
wire nxt_ucp2_write_after_cpy;
reg  ucp2_write_after_cpy;
wire [2:0] nxt_ucp2_write_ptr;
reg  [2:0] ucp2_write_ptr;
wire [2:0] agrd_ucp2_write_ptr;

wire [2:0] ucp3_rd_off;
wire nxt_ucp3_write_after_cpy;
reg  ucp3_write_after_cpy;
wire [2:0] nxt_ucp3_write_ptr;
reg  [2:0] ucp3_write_ptr;
wire [2:0] agrd_ucp3_write_ptr;

wire [2:0] ucp4_rd_off;
wire nxt_ucp4_write_after_cpy;
reg  ucp4_write_after_cpy;
wire [2:0] nxt_ucp4_write_ptr;
reg  [2:0] ucp4_write_ptr;
wire [2:0] agrd_ucp4_write_ptr;

wire [2:0] ucp5_rd_off;
wire nxt_ucp5_write_after_cpy;
```

```
reg  ucp5_write_after_cpy;
wire [2:0] nxt_ucp5_write_ptr;
reg  [2:0] ucp5_write_ptr;
wire [2:0] agrd_ucp5_write_ptr;

wire [2:0] gb_rd_off;
wire nxt_gb_write_after_cpy;
reg  gb_write_after_cpy;
wire [2:0] nxt_gb_write_ptr;
reg  [2:0] gb_write_ptr;
wire [2:0] agrd_gb_write_ptr;

wire [2:0] pntsz_rd_off;
wire nxt_pntsz_write_after_cpy;
reg  pntsz_write_after_cpy;
wire [2:0] nxt_pntsz_write_ptr;
reg  [2:0] pntsz_write_ptr;
wire [2:0] agrd_pntsz_write_ptr;

//CCG decode results
reg   [2:0] ccg_vte_opcode;
reg   [2:0] ccg_vte_st_indx;

reg   [2:0] ccg_ve_opcode;
reg   [1:0] ccg_ve_a_memsel;
```

```
reg   [1:0] ccg_ve_b_memsel;
reg   [2:0] ccg_ve_ax_select;
reg   [2:0] ccg_ve_ay_select;
reg   [2:0] ccg_ve_az_select;
reg   [2:0] ccg_ve_aw_select;
reg   [2:0] ccg_ve_bx_select;
reg   [2:0] ccg_ve_by_select;
reg   [2:0] ccg_ve_bz_select;
reg   [2:0] ccg_ve_bw_select;
reg         ccg_ve_a_is_wwww;
reg         ccg_ve_broadcast_x;
reg         ccg_ve_abs_a;
reg         ccg_ve_abs_b;
reg         ccg_ve_abs_c;
reg         ccg_ve_ax_negate;
reg         ccg_ve_ay_negate;
reg         ccg_ve_az_negate;
reg         ccg_ve_aw_negate;
reg         ccg_ve_bx_negate;
reg         ccg_ve_by_negate;
reg         ccg_ve_bz_negate;
reg         ccg_ve_bw_negate;
reg         ccg_ve_cx_negate;
reg         ccg_ve_cy_negate;
reg         ccg_ve_cz_negate;
reg         ccg_ve_cw_negate;
reg         ccg_ve_bcc_flat_tst;
reg   [2:0] ccg_ve_out_mem_sel;
reg   [5:0] ccg_ve_out_addr;
reg   [3:0] ccg_ve_out_we;
reg         ccg_ve_accum_sel;
reg   [3:0] ccg_ve_pre_acc_we;

reg   [5:0] ccg_agve_dly_vertex_store_indx;
reg   [1:0] ccg_agve_dly_valid_bit_set;
reg   [3:0] ccg_agve_dly_user_clip_indx;
reg         ccg_agve_dly_vv_cc_test;
reg         ccg_agve_dly_ucp_cc_test;
reg         ccg_agve_dly_bcc_cc_test;
reg         ccg_agve_dly_ps_ucp_cc_test;
reg         ccg_agve_dly_ps_engh_test;

reg   [1:0] get_baryc_a_memsel;
reg   [2:0] get_baryc_ax_select;
reg   [2:0] get_baryc_ay_select;
reg   [2:0] get_baryc_az_select;
reg         get_baryc_ax_negate;
reg         get_baryc_cx_negate;

reg   [1:0] get_pos_a_memsel;
```

```

reg   [1:0] get_clipdist_a_memsel;
reg   [2:0] get_clipdist_a_select;

//CLIPPER decode results
reg   [2:0] clip_vte_opcode;
reg   [2:0] clip_vte_st_indx;

reg   [2:0] clip_ve_opcode;
reg   [1:0] clip_ve_a_memsel;
reg   [1:0] clip_ve_b_memsel;
reg   [2:0] clip_ve_ax_select;
reg   [2:0] clip_ve_ay_select;
reg   [2:0] clip_ve_az_select;
reg   [2:0] clip_ve_aw_select;
reg   [2:0] clip_ve_bx_select;
reg   [2:0] clip_ve_by_select;
reg   [2:0] clip_ve_bz_select;
reg   [2:0] clip_ve_bw_select;
reg         clip_ve_a_is_wwww;
reg         clip_ve_broadcast_x;
reg         clip_ve_abs_a;
reg         clip_ve_abs_b;
reg         clip_ve_abs_c;
reg         clip_ve_ax_negate;
```

```
reg         clip_ve_ay_negate;
reg         clip_ve_az_negate;
reg         clip_ve_aw_negate;
reg         clip_ve_bx_negate;
reg         clip_ve_by_negate;
reg         clip_ve_bz_negate;
reg         clip_ve_bw_negate;
reg         clip_ve_cx_negate;
reg         clip_ve_cy_negate;
reg         clip_ve_cz_negate;
reg         clip_ve_cw_negate;
reg         clip_ve_bcc_flat_tst;
reg   [2:0] clip_ve_out_mem_sel;
reg   [5:0] clip_ve_out_addr;
reg   [3:0] clip_ve_out_we;
reg         clip_ve_accum_sel;
reg   [3:0] clip_ve_pre_acc_we;

reg   [5:0] clip_agve_dly_vertex_store_indx;
reg   [1:0] clip_agve_dly_valid_bit_set;
reg   [3:0] clip_agve_dly_user_clip_indx;
reg         clip_agve_dly_vv_cc_test;
reg         clip_agve_dly_ucp_cc_test;
reg         clip_agve_dly_bcc_cc_test;
reg         clip_agve_dly_ps_ucp_cc_test;
```

```
reg         clip_agve_dly_ps_engh_test;



// arbiter mux of control
reg   [2:0] arbsel_vte_opcode;
reg   [2:0] arbsel_vte_st_indx;
reg   [2:0] arbsel_ve_opcode;
reg   [1:0] arbsel_ve_a_memsel;
reg   [1:0] arbsel_ve_b_memsel;
reg   [2:0] arbsel_ve_ax_select;
reg   [2:0] arbsel_ve_ay_select;
reg   [2:0] arbsel_ve_az_select;
reg   [2:0] arbsel_ve_aw_select;
reg   [2:0] arbsel_ve_bx_select;
reg   [2:0] arbsel_ve_by_select;
reg   [2:0] arbsel_ve_bz_select;
reg   [2:0] arbsel_ve_bw_select;
reg         arbsel_ve_a_is_wwww;
reg         arbsel_ve_broadcast_x;
reg         arbsel_ve_abs_a;
reg         arbsel_ve_abs_b;
reg         arbsel_ve_abs_c;
reg         arbsel_ve_ax_negate;
reg         arbsel_ve_ay_negate;
reg       arbsel_ve_az_negate;
reg       arbsel_ve_aw_negate;
reg       arbsel_ve_bx_negate;
reg       arbsel_ve_by_negate;
reg       arbsel_ve_bz_negate;
reg       arbsel_ve_bw_negate;
reg       arbsel_ve_cx_negate;
reg       arbsel_ve_cy_negate;
reg       arbsel_ve_cz_negate;
reg       arbsel_ve_cw_negate;
reg       arbsel_ve_bcc_flat_tst;
reg  [2:0] arbsel_ve_out_mem_sel;
reg  [5:0] arbsel_ve_out_addr;
reg  [3:0] arbsel_ve_out_we;
reg       arbsel_ve_accum_sel;
reg  [3:0] arbsel_ve_pre_acc_we;

reg  [5:0] arbsel_agve_dly_vertex_store_indx;
reg  [1:0] arbsel_agve_dly_valid_bit_set;
reg  [3:0] arbsel_agve_dly_user_clip_indx;
reg       arbsel_agve_dly_vv_cc_test;
reg       arbsel_agve_dly_ucp_cc_test;
reg       arbsel_agve_dly_bcc_cc_test;
reg       arbsel_agve_dly_ps_ucp_cc_test;
reg       arbsel_agve_dly_ps_engh_test;
```

```

reg       agve_valid_op;

//Declare AG_R0 register items
reg      ccg_xfc_r0;
reg      ccg_ve_cc_valid_r0;
reg [1:0] ccg_sm_state_indx_r0;
reg [2:0] ccg_state_var_indx_r0;
reg [5:0] ccg_vertex_store_indx_r0;
reg [2:0] ccg_ve_ucp_indx_r0;

reg      clip_xfc_r0;
reg [3:0] clip_plane_indx_r0;
reg [5:0] clip_dst_vertex_indx_r0;
reg [6:0] clip_src_vertex_indx_r0;
reg      clip_src_vertex_type_r0;
reg      clip_ve_ucp_valid_r0;
reg [6:0] clip_sm_state_indx_r0;
reg [2:0] clip_state_var_indx_r0;

//Declare AG_R1 register items
reg       ccg_xfc_r1;
//CCG R1 Delay
reg  [2:0] ccg_vte_opcode_r1;
reg  [2:0] ccg_vte_st_indx_r1;
```

```
reg  [2:0] ccg_ve_opcode_r1;
reg  [1:0] ccg_ve_a_memsel_r1;
reg  [1:0] ccg_ve_b_memsel_r1;
reg  [2:0] ccg_ve_ax_select_r1;
reg  [2:0] ccg_ve_ay_select_r1;
reg  [2:0] ccg_ve_az_select_r1;
reg  [2:0] ccg_ve_aw_select_r1;
reg  [2:0] ccg_ve_bx_select_r1;
reg  [2:0] ccg_ve_by_select_r1;
reg  [2:0] ccg_ve_bz_select_r1;
reg  [2:0] ccg_ve_bw_select_r1;
reg       ccg_ve_a_is_wwww_r1;
reg       ccg_ve_broadcast_x_r1;
reg       ccg_ve_abs_a_r1;
reg       ccg_ve_abs_b_r1;
reg       ccg_ve_abs_c_r1;
reg       ccg_ve_ax_negate_r1;
reg       ccg_ve_ay_negate_r1;
reg       ccg_ve_az_negate_r1;
reg       ccg_ve_aw_negate_r1;
reg       ccg_ve_bx_negate_r1;
reg       ccg_ve_by_negate_r1;
reg       ccg_ve_bz_negate_r1;
reg       ccg_ve_bw_negate_r1;
reg       ccg_ve_cx_negate_r1;
```

```
reg       ccg_ve_cy_negate_r1;
reg       ccg_ve_cz_negate_r1;
reg       ccg_ve_cw_negate_r1;
reg       ccg_ve_bcc_flat_tst_r1;
reg  [2:0] ccg_ve_out_mem_sel_r1;
reg  [5:0] ccg_ve_out_addr_r1;
reg  [3:0] ccg_ve_out_we_r1;
reg       ccg_ve_accum_sel_r1;
reg  [3:0] ccg_ve_pre_acc_we_r1;

reg  [5:0] ccg_agve_dly_vertex_store_indx_r1;
reg  [1:0] ccg_agve_dly_valid_bit_set_r1;
reg  [3:0] ccg_agve_dly_user_clip_indx_r1;
reg       ccg_agve_dly_vv_cc_test_r1;
reg       ccg_agve_dly_ucp_cc_test_r1;
reg       ccg_agve_dly_bcc_cc_test_r1;
reg       ccg_agve_dly_ps_ucp_cc_test_r1;
reg       ccg_agve_dly_ps_engh_test_r1;

reg       clip_xfc_r1;
//CLIPPER R1 Delay
reg  [2:0] clip_vte_opcode_r1;
reg  [2:0] clip_vte_st_indx_r1;
reg  [2:0] clip_ve_opcode_r1;
reg  [1:0] clip_ve_a_memsel_r1;
reg [1:0] clip_ve_b_memsel_r1;
reg [2:0] clip_ve_ax_select_r1;
reg [2:0] clip_ve_ay_select_r1;
reg [2:0] clip_ve_az_select_r1;
reg [2:0] clip_ve_aw_select_r1;
reg [2:0] clip_ve_bx_select_r1;
reg [2:0] clip_ve_by_select_r1;
reg [2:0] clip_ve_bz_select_r1;
reg [2:0] clip_ve_bw_select_r1;
reg       clip_ve_a_is_wwww_r1;
reg       clip_ve_broadcast_x_r1;
reg       clip_ve_abs_a_r1;
reg       clip_ve_abs_b_r1;
reg       clip_ve_abs_c_r1;
reg       clip_ve_ax_negate_r1;
reg       clip_ve_ay_negate_r1;
reg       clip_ve_az_negate_r1;
reg       clip_ve_aw_negate_r1;
reg       clip_ve_bx_negate_r1;
reg       clip_ve_by_negate_r1;
reg       clip_ve_bz_negate_r1;
reg       clip_ve_bw_negate_r1;
reg       clip_ve_cx_negate_r1;
reg       clip_ve_cy_negate_r1;
reg       clip_ve_cz_negate_r1;
```

```
reg       clip_ve_cw_negate_r1;
reg       clip_ve_bcc_flat_tst_r1;
reg [2:0] clip_ve_out_mem_sel_r1;
reg [5:0] clip_ve_out_addr_r1;
reg [3:0] clip_ve_out_we_r1;
reg       clip_ve_accum_sel_r1;
reg [3:0] clip_ve_pre_acc_we_r1;

reg [5:0] clip_agve_dly_vertex_store_indx_r1;
reg [1:0] clip_agve_dly_valid_bit_set_r1;
reg [3:0] clip_agve_dly_user_clip_indx_r1;
reg       clip_agve_dly_vv_cc_test_r1;
reg       clip_agve_dly_ucp_cc_test_r1;
reg       clip_agve_dly_bcc_cc_test_r1;
reg       clip_agve_dly_ps_ucp_cc_test_r1;
reg       clip_agve_dly_ps_engh_test_r1;

//Declare AG_R1 register items
reg [2:0] vte_opcode_r2;
reg [2:0] vte_st_indx_r2;
reg [2:0] ve_opcode_r2;
reg [1:0] ve_a_memsel_r2;
reg [1:0] ve_b_memsel_r2;
reg [2:0] ve_ax_select_r2;
reg [2:0] ve_ay_select_r2;
```

```
reg [2:0] ve_az_select_r2;
reg [2:0] ve_aw_select_r2;
reg [2:0] ve_bx_select_r2;
reg [2:0] ve_by_select_r2;
reg [2:0] ve_bz_select_r2;
reg [2:0] ve_bw_select_r2;
reg       ve_a_is_wwww_r2;
reg       ve_broadcast_x_r2;
reg       ve_abs_a_r2;
reg       ve_abs_b_r2;
reg       ve_abs_c_r2;
reg       ve_ax_negate_r2;
reg       ve_ay_negate_r2;
reg       ve_az_negate_r2;
reg       ve_aw_negate_r2;
reg       ve_bx_negate_r2;
reg       ve_by_negate_r2;
reg       ve_bz_negate_r2;
reg       ve_bw_negate_r2;
reg       ve_cx_negate_r2;
reg       ve_cy_negate_r2;
reg       ve_cz_negate_r2;
reg       ve_cw_negate_r2;
reg       ve_bcc_flat_tst_r2;
reg [2:0] ve_out_mem_sel_r2;
```

```
reg [5:0] ve_out_addr_r2;
reg [3:0] ve_out_we_r2;
reg       ve_accum_sel_r2;
reg [3:0] ve_pre_acc_we_r2;

reg [5:0] agve_dly_vertex_store_indx_r2;
reg [1:0] agve_dly_valid_bit_set_r2;
reg [3:0] agve_dly_user_clip_indx_r2;
reg       agve_dly_vv_cc_test_r2;
reg       agve_dly_ucp_cc_test_r2;
reg       agve_dly_bcc_cc_test_r2;
reg       agve_dly_ps_ucp_cc_test_r2;
reg       agve_dly_ps_engh_test_r2;

//Declare AG_R3 output register items

reg [2:0] ag_vte_opcode;
reg [2:0] ag_vte_st_indx;
reg [1:0] ag_vte_vertex_store_indx;
reg [2:0] ag_ve_opcode;
reg [31:0] ag_ve_in_a0;
reg [31:0] ag_ve_in_a1;
reg [31:0] ag_ve_in_a2;
reg [31:0] ag_ve_in_a3;
reg [31:0] ag_ve_in_b0;
reg  [31:0] ag_ve_in_b1;
reg  [31:0] ag_ve_in_b2;
reg  [31:0] ag_ve_in_b3;
reg         ag_ve_a_is_wwww;
reg         ag_ve_broadcast_x;
reg         ag_ve_abs_a;
reg         ag_ve_abs_b;
reg         ag_ve_abs_c;
reg         ag_ve_ax_negate;
reg         ag_ve_ay_negate;
reg         ag_ve_az_negate;
reg         ag_ve_aw_negate;
reg         ag_ve_bx_negate;
reg         ag_ve_by_negate;
reg         ag_ve_bz_negate;
reg         ag_ve_bw_negate;
reg         ag_ve_cx_negate;
reg         ag_ve_cy_negate;
reg         ag_ve_cz_negate;
reg         ag_ve_cw_negate;
reg         ag_ve_bcc_flat_tst;
reg   [2:0] ag_ve_out_mem_sel;
reg   [5:0] ag_ve_out_addr;
reg   [3:0] ag_ve_out_we;
reg         ag_ve_accum_sel;
```

```
reg   [3:0] ag_ve_pre_accum_we;

reg  [127:0] amem_sel_data;
reg  [127:0] bmem_sel_data;
wire  [31:0] agswz_ve_in_a0;
wire  [31:0] agswz_ve_in_a1;
wire  [31:0] agswz_ve_in_a2;
wire  [31:0] agswz_ve_in_a3;
wire  [31:0] agswz_ve_in_b0;
wire  [31:0] agswz_ve_in_b1;
wire  [31:0] agswz_ve_in_b2;
wire  [31:0] agswz_ve_in_b3;

reg         agve_dly_valid_op_r3;
reg   [5:0] agve_dly_vertex_store_indx_r3;
reg   [1:0] agve_dly_valid_bit_set_r3;
reg   [3:0] agve_dly_user_clip_indx_r3;
reg         agve_dly_vv_cc_test_r3;
reg         agve_dly_ucp_cc_test_r3;
reg         agve_dly_bcc_cc_test_r3;
reg         agve_dly_ps_ucp_cc_test_r3;
reg         agve_dly_ps_engh_test_r3;

//delay pipe
reg  [17:0] agve_dly0;
```

```
reg  [17:0] agve_dly1;
reg  [17:0] agve_dly2;
reg  [17:0] agve_dly3;
reg  [17:0] agve_dly4;
reg  [17:0] agve_dly5;

reg         agve_dly_valid_op;
reg   [5:0] agve_dly_vertex_store_indx;
reg   [1:0] agve_dly_valid_bit_set;
reg   [3:0] agve_dly_user_clip_indx;
reg         agve_dly_vv_cc_test;
reg         agve_dly_ucp_cc_test;
reg         agve_dly_bcc_cc_test;
reg         agve_dly_ps_ucp_cc_test;
reg         agve_dly_ps_engh_test;

//**********************************************************************
// Combinational logic
//**********************************************************************

//map ccg and clip controller signals to broken out names
always @(ccg_to_arb_data or
         clip_to_arb_data)
 begin
    {ccg_ve_cc_valid,
```

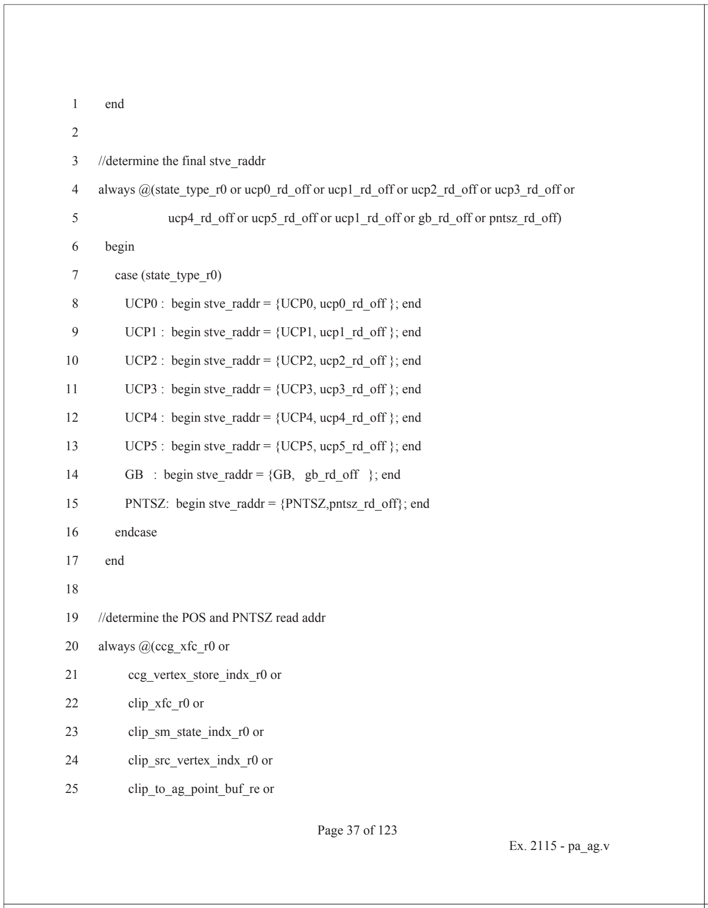
```
     ccg_ve_valid,
     ccg_sm_state_indx,
     ccg_state_var_indx,
     ccg_vertex_store_indx,
     ccg_ve_ucp_indx} = ccg_to_arb_data ;

    {clip_plane_indx,
     clip_dst_vertex_indx,
     clip_src_vertex_indx,
     clip_src_vertex_type,
     clip_ve_ucp_valid,
     clip_sm_state_indx,
     clip_state_var_indx} = clip_to_arb_data ;
 end

//maintain state dirty bits
assign nxt_ucp0_write_after_cpy  = dirty(srst, rbiu_cpy, rbiu_ag_ucp0_sel);
assign nxt_ucp1_write_after_cpy  = dirty(srst, rbiu_cpy, rbiu_ag_ucp1_sel);
assign nxt_ucp2_write_after_cpy  = dirty(srst, rbiu_cpy, rbiu_ag_ucp2_sel);
assign nxt_ucp3_write_after_cpy  = dirty(srst, rbiu_cpy, rbiu_ag_ucp3_sel);
assign nxt_ucp4_write_after_cpy  = dirty(srst, rbiu_cpy, rbiu_ag_ucp4_sel);
assign nxt_ucp5_write_after_cpy  = dirty(srst, rbiu_cpy, rbiu_ag_ucp5_sel);
assign nxt_gb_write_after_cpy    = dirty(srst, rbiu_cpy, rbiu_ag_gb_sel);
assign nxt_pntsz_write_after_cpy = dirty(srst, rbiu_cpy, rbiu_ag_pntsz_sel);

```

```
//maintain state next write ptr's
assign nxt_ucp0_write_ptr  = stwtptr(srst, rbiu_cpy, ucp0_write_after_cpy,  ucp0_write_ptr);
assign nxt_ucp1_write_ptr  = stwtptr(srst, rbiu_cpy, ucp1_write_after_cpy,  ucp1_write_ptr);
assign nxt_ucp2_write_ptr  = stwtptr(srst, rbiu_cpy, ucp2_write_after_cpy,  ucp2_write_ptr);
assign nxt_ucp3_write_ptr  = stwtptr(srst, rbiu_cpy, ucp3_write_after_cpy,  ucp3_write_ptr);
assign nxt_ucp4_write_ptr  = stwtptr(srst, rbiu_cpy, ucp4_write_after_cpy,  ucp4_write_ptr);
assign nxt_ucp5_write_ptr  = stwtptr(srst, rbiu_cpy, ucp5_write_after_cpy,  ucp5_write_ptr);
assign nxt_gb_write_ptr    = stwtptr(srst, rbiu_cpy, gb_write_after_cpy,    gb_write_ptr);
assign nxt_pntsz_write_ptr = stwtptr(srst, rbiu_cpy, pntsz_write_after_cpy, pntsz_write_ptr);

//select and assemble state mem write address
always @(rbiu_ag_ucp0_sel or  ucp0_write_ptr or
         rbiu_ag_ucp1_sel or  ucp1_write_ptr or
         rbiu_ag_ucp2_sel or  ucp2_write_ptr or
         rbiu_ag_ucp3_sel or  ucp3_write_ptr or
         rbiu_ag_ucp4_sel or  ucp4_write_ptr or
         rbiu_ag_ucp5_sel or  ucp5_write_ptr or
         rbiu_ag_gb_sel or    gb_write_ptr or
         rbiu_ag_pntsz_sel or pntsz_write_ptr)
  begin
    stve_wa = 6'b000000;
    stve_we = 4'b0000;
    if (rbiu_ag_ucp0_sel) begin
      stve_wa = {3'b000, ucp0_write_ptr };
      stve_we = rbiu_ag_ucp0_sel;
```

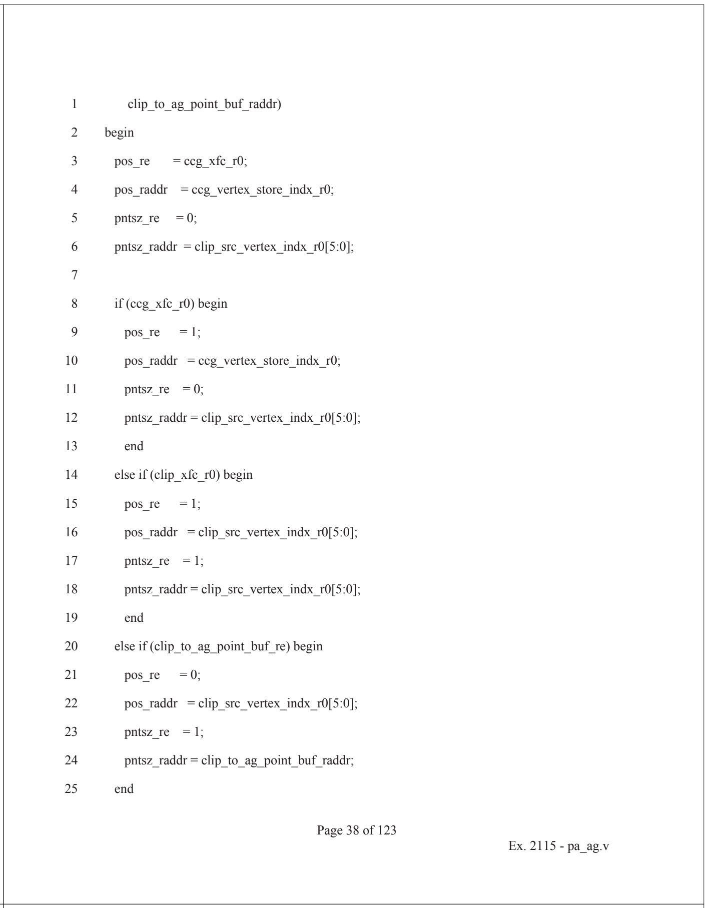
```
    end if (rbiu_ag_ucp1_sel)begin
      stve_wa = {3'b001, ucp1_write_ptr };
      stve_we = rbiu_ag_ucp1_sel;
    end if (rbiu_ag_ucp2_sel)begin
      stve_wa = {3'b010, ucp2_write_ptr };
      stve_we = rbiu_ag_ucp2_sel;
    end if (rbiu_ag_ucp3_sel)begin
      stve_wa = {3'b011, ucp3_write_ptr };
      stve_we = rbiu_ag_ucp3_sel;
    end if (rbiu_ag_ucp4_sel)begin
      stve_wa = {3'b100, ucp4_write_ptr };
      stve_we = rbiu_ag_ucp4_sel;
    end if (rbiu_ag_ucp5_sel)begin
      stve_wa = {3'b101, ucp5_write_ptr };
      stve_we = rbiu_ag_ucp5_sel;
    end if (rbiu_ag_gb_sel)begin
      stve_wa = {3'b110, gb_write_ptr };
      stve_we = rbiu_ag_gb_sel;
    end if (rbiu_ag_pntsz_sel)begin
      stve_wa = {3'b111, pntsz_write_ptr };
      stve_we = rbiu_ag_pntsz_sel;
    end
  end

//Arbiter logic
```

```
assign {arb_clip_xfc, arb_ccg_xfc} = arb(srst, clip_ve_valid, ccg_ve_valid);

//Arbiter generation of intermediate stve raddr
always @(clip_ve_valid or
         clip_state_var_indx or
         clip_plane_indx or
         clip_sm_state_indx or
         ccg_ve_valid or
         ccg_state_var_indx or
         ccg_ve_ucp_indx or
         ccg_sm_state_indx)
  begin
    nxt_stve_re           = 1'b0;
    nxt_state_type[5:3]    = 3'b000;
    nxt_arb_state_var_indx = 3'b000;
    arb_ucp_indx           = 3'b000;

    if (clip_ve_valid == 1'b1) begin
      nxt_arb_state_var_indx = clip_state_var_indx;
      arb_ucp_indx           = clip_plane_indx;

      //need to add clipper read address for stve
      if ((clip_sm_state_indx == SMC_PS_CULL_RADIUS_VPORT) ||
         (clip_sm_state_indx == SMC_PS_XY_RADIUS_VPORT)) begin
        nxt_stve_re        = clip_ve_valid;
```

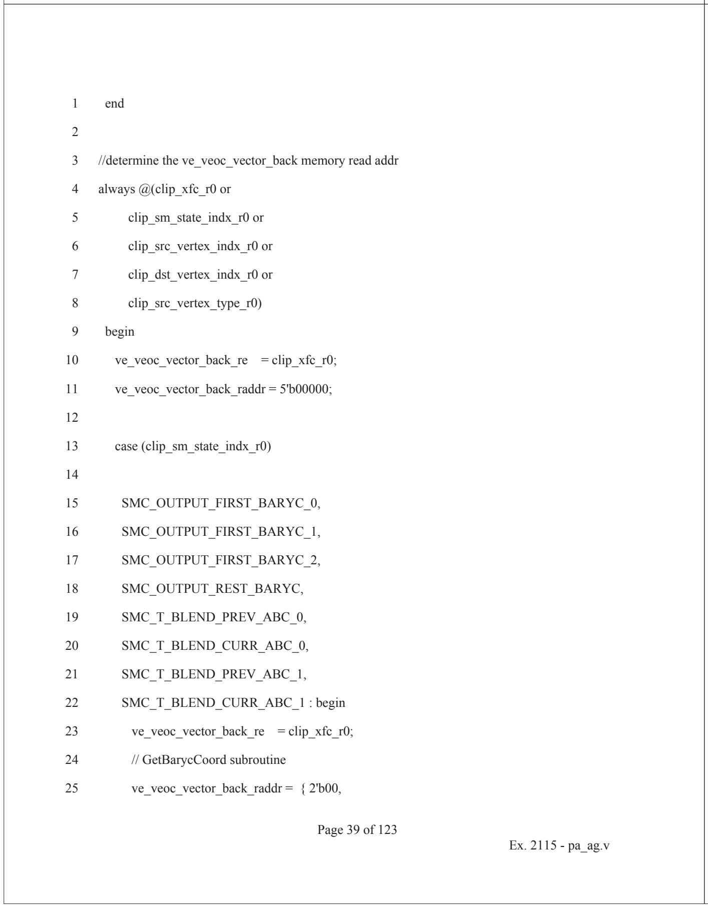
```
        nxt_state_type[5:3] = PNTSZ;
      end else if (clip_sm_state_indx == SMC_CLIP_DIST_VV) begin
        nxt_stve_re        = clip_ve_valid;
        nxt_state_type[5:3] = GB;
      end else if ((clip_sm_state_indx == SMC_CLIP_DIST_UCP) ||
               (clip_sm_state_indx == SMC_PS_CLIP_DIST_UCP) ||
               (clip_sm_state_indx == SMC_PS_UCP_DIST_UL) ||
               (clip_sm_state_indx == SMC_PS_UCP_DIST_UR) ||
               (clip_sm_state_indx == SMC_PS_UCP_DIST_LR) ||
               (clip_sm_state_indx == SMC_PS_UCP_DIST_LL)) begin
        nxt_stve_re        = clip_ve_valid;
        nxt_state_type[5:3] = clip_plane_indx;
      end
    end else begin //ccg wins arbitration
      nxt_arb_state_var_indx = ccg_state_var_indx;
      arb_ucp_indx           = ccg_ve_ucp_indx;
      if(ccg_sm_state_indx == SMCC_EMPTY) begin
        nxt_stve_re        = ccg_ve_valid;
        nxt_state_type[5:3] = GB; //guard band lookup
      end else begin
        nxt_stve_re        = ccg_ve_valid;
        nxt_state_type[5:3] = ccg_ve_ucp_indx;
      end

    end
```



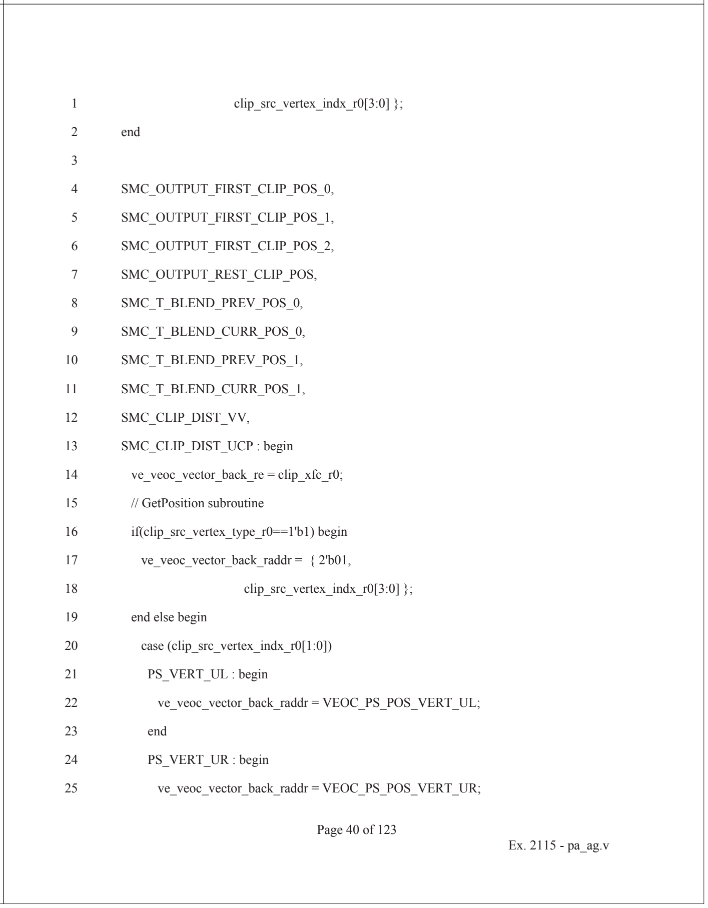
```
  end

//determine the final stve_raddr
always @(state_type_r0 or ucp0_rd_off or ucp1_rd_off or ucp2_rd_off or ucp3_rd_off or
              ucp4_rd_off or ucp5_rd_off or ucp1_rd_off or gb_rd_off or pntsz_rd_off)
  begin
    case (state_type_r0)
      UCP0 :  begin stve_raddr = {UCP0, ucp0_rd_off }; end
      UCP1 :  begin stve_raddr = {UCP1, ucp1_rd_off }; end
      UCP2 :  begin stve_raddr = {UCP2, ucp2_rd_off }; end
      UCP3 :  begin stve_raddr = {UCP3, ucp3_rd_off }; end
      UCP4 :  begin stve_raddr = {UCP4, ucp4_rd_off }; end
      UCP5 :  begin stve_raddr = {UCP5, ucp5_rd_off }; end
      GB   :  begin stve_raddr = {GB,   gb_rd_off   }; end
      PNTSZ:  begin stve_raddr = {PNTSZ,pntsz_rd_off}; end
    endcase
  end

//determine the POS and PNTSZ read addr
always @(ccg_xfc_r0 or
      ccg_vertex_store_indx_r0 or
      clip_xfc_r0 or
      clip_sm_state_indx_r0 or
      clip_src_vertex_indx_r0 or
      clip_to_ag_point_buf_re or
```



```
    clip_to_ag_point_buf_raddr)
begin
  pos_re      = ccg_xfc_r0;
  pos_raddr   = ccg_vertex_store_indx_r0;
  pntsz_re    = 0;
  pntsz_raddr  = clip_src_vertex_indx_r0[5:0];

  if (ccg_xfc_r0) begin
    pos_re      = 1;
    pos_raddr   = ccg_vertex_store_indx_r0;
    pntsz_re    = 0;
    pntsz_raddr = clip_src_vertex_indx_r0[5:0];
    end
  else if (clip_xfc_r0) begin
    pos_re      = 1;
    pos_raddr   = clip_src_vertex_indx_r0[5:0];
    pntsz_re    = 1;
    pntsz_raddr = clip_src_vertex_indx_r0[5:0];
    end
  else if (clip_to_ag_point_buf_re) begin
    pos_re      = 0;
    pos_raddr   = clip_src_vertex_indx_r0[5:0];
    pntsz_re    = 1;
    pntsz_raddr = clip_to_ag_point_buf_raddr;
  end
```



```
  end

//determine the ve_veoc_vector_back memory read addr
always @(clip_xfc_r0 or
      clip_sm_state_indx_r0 or
      clip_src_vertex_indx_r0 or
      clip_dst_vertex_indx_r0 or
      clip_src_vertex_type_r0)
  begin
    ve_veoc_vector_back_re    = clip_xfc_r0;
    ve_veoc_vector_back_raddr = 5'b00000;

    case (clip_sm_state_indx_r0)

      SMC_OUTPUT_FIRST_BARYC_0,
      SMC_OUTPUT_FIRST_BARYC_1,
      SMC_OUTPUT_FIRST_BARYC_2,
      SMC_OUTPUT_REST_BARYC,
      SMC_T_BLEND_PREV_ABC_0,
      SMC_T_BLEND_CURR_ABC_0,
      SMC_T_BLEND_PREV_ABC_1,
      SMC_T_BLEND_CURR_ABC_1 : begin
        ve_veoc_vector_back_re    = clip_xfc_r0;
        // GetBarycCoord subroutine
        ve_veoc_vector_back_raddr =  { 2'b00,
```



```
                          clip_src_vertex_indx_r0[3:0] };
    end

    SMC_OUTPUT_FIRST_CLIP_POS_0,
    SMC_OUTPUT_FIRST_CLIP_POS_1,
    SMC_OUTPUT_FIRST_CLIP_POS_2,
    SMC_OUTPUT_REST_CLIP_POS,
    SMC_T_BLEND_PREV_POS_0,
    SMC_T_BLEND_CURR_POS_0,
    SMC_T_BLEND_PREV_POS_1,
    SMC_T_BLEND_CURR_POS_1,
    SMC_CLIP_DIST_VV,
    SMC_CLIP_DIST_UCP : begin
      ve_veoc_vector_back_re = clip_xfc_r0;
      // GetPosition subroutine
      if(clip_src_vertex_type_r0==1'b1) begin
        ve_veoc_vector_back_raddr =  { 2'b01,
                            clip_src_vertex_indx_r0[3:0] };
      end else begin
        case (clip_src_vertex_indx_r0[1:0])
          PS_VERT_UL : begin
            ve_veoc_vector_back_raddr = VEOC_PS_POS_VERT_UL;
          end
          PS_VERT_UR : begin
            ve_veoc_vector_back_raddr = VEOC_PS_POS_VERT_UR;
          end
          PS_VERT_LL : begin
            ve_veoc_vector_back_raddr = VEOC_PS_POS_VERT_LL;
          end
          PS_VERT_LR : begin
            ve_veoc_vector_back_raddr = VEOC_PS_POS_VERT_LR;
          end
          2'bxx : begin
            ve_veoc_vector_back_raddr = 6'bxxxxxx;
          end
        endcase
      end
    end

    SMC_EDGE_DISTANCE_0,
    SMC_EDGE_DISTANCE_1,
    SMC_CLIP_T_FACTOR_PREV_0,
    SMC_CLIP_T_FACTOR_CURR_0,
    SMC_CLIP_T_FACTOR_PREV_1,
    SMC_CLIP_T_FACTOR_CURR_1 : begin
      // GetClipDist subroutine
      ve_veoc_vector_back_re    = clip_xfc_r0;
      if(clip_src_vertex_type_r0==1'b1) begin
        ve_veoc_vector_back_raddr =  { 2'b00,
                                      clip_src_vertex_indx_r0[3:0] };
```

```
      end else begin
        ve_veoc_vector_back_raddr = VEOC_CLIP_DIST_ORIG;
      end
      // end GetClipDist
    end

    SMC_PS_UCP_DIST_UL : begin
      ve_veoc_vector_back_raddr = VEOC_PS_POS_VERT_UL;
    end

    SMC_PS_UCP_DIST_UR : begin
      ve_veoc_vector_back_raddr = VEOC_PS_POS_VERT_UR;
    end

    SMC_PS_UCP_DIST_LL : begin
      ve_veoc_vector_back_raddr = VEOC_PS_POS_VERT_LL;
    end

    SMC_PS_UCP_DIST_LR : begin
      ve_veoc_vector_back_raddr = VEOC_PS_POS_VERT_LR;
    end

    SMC_PS_ENGH_TEST : begin
      ve_veoc_vector_back_raddr = VEOC_CLIP_DIST_ORIG;
    end
```

```

    endcase
  end

// CCG_DECODE
always @(ccg_state_var_indx_r0 or
         ccg_ve_ucp_indx_r0 or
         ccg_ve_cc_valid_r0 or
         ccg_vertex_store_indx_r0 or
         ccg_sm_state_indx_r0 or
         ccg_xfc_r0 or
         dx_clip_space_def)
  begin

    ccg_vte_opcode   = VTE_NO_OP;
    ccg_ve_opcode    = VECTOR_NO_OP;
    ccg_ve_a_memsel  = VEA_MEMSEL_VEOC_VE;
    ccg_ve_b_memsel  = VEB_MEMSEL_STVE_VE;
    ccg_ve_ax_select = SRC_SELECT_FORCE_0;
    ccg_ve_ay_select = SRC_SELECT_FORCE_0;
    ccg_ve_az_select = SRC_SELECT_FORCE_0;
    ccg_ve_aw_select = SRC_SELECT_FORCE_0;
    ccg_ve_bx_select = SRC_SELECT_FORCE_0;
    ccg_ve_by_select = SRC_SELECT_FORCE_0;
    ccg_ve_bz_select = SRC_SELECT_FORCE_0;
```

```
    ccg_ve_bw_select = SRC_SELECT_FORCE_0;
    ccg_ve_a_is_wwww    = 1'b0;
    ccg_ve_broadcast_x  = 1'b0;
    ccg_ve_abs_a        = 1'b0;
    ccg_ve_abs_b        = 1'b0;
    ccg_ve_abs_c        = 1'b0;
    ccg_ve_ax_negate    = 1'b0;
    ccg_ve_ay_negate    = 1'b0;
    ccg_ve_az_negate    = 1'b0;
    ccg_ve_aw_negate    = 1'b0;
    ccg_ve_bx_negate    = 1'b0;
    ccg_ve_by_negate    = 1'b0;
    ccg_ve_bz_negate    = 1'b0;
    ccg_ve_bw_negate    = 1'b0;
    ccg_ve_cx_negate    = 1'b0;
    ccg_ve_cy_negate    = 1'b0;
    ccg_ve_cz_negate    = 1'b0;
    ccg_ve_cw_negate    = 1'b0;
    ccg_ve_bcc_flat_tst = 1'b0;
    ccg_ve_out_mem_sel  = 3'b0;
    ccg_ve_out_addr     = 5'b00000;
    ccg_ve_out_we       = 4'b0000;
    ccg_ve_accum_sel    = 1'b0;
    ccg_ve_pre_acc_we   = 4'b0000;

    ccg_agve_dly_vertex_store_indx = ccg_vertex_store_indx_r0;
    ccg_agve_dly_valid_bit_set     = 2'b0;
    ccg_agve_dly_user_clip_indx    = 4'b0;
    ccg_agve_dly_vv_cc_test        = 1'b0;
    ccg_agve_dly_ucp_cc_test       = 1'b0;
    ccg_agve_dly_bcc_cc_test       = 1'b0;
    ccg_agve_dly_ps_ucp_cc_test    = 1'b0;
    ccg_agve_dly_ps_engh_test      = 1'b0;

    ccg_vte_st_indx = ccg_state_var_indx_r0;

    if (ccg_xfc_r0 == 1'b1) begin
      case (ccg_sm_state_indx_r0)
        SMCC_EMPTY : begin
          if (dx_clip_space_def) begin
            ccg_ve_opcode = VE_CLIP_CODE_VV_Z0W;
          end else begin
            ccg_ve_opcode = VE_CLIP_CODE_VV_ZWW;
          end
          ccg_vte_opcode   = VTE_ORIG_POS;
          ccg_ve_a_is_wwww        = 1'b1;
          ccg_agve_dly_vv_cc_test = 1'b1;
          if(ccg_ve_cc_valid_r0) begin
            ccg_agve_dly_valid_bit_set = CC_VALID;
          end
```

```
          ccg_ve_a_memsel  = VEA_MEMSEL_POS_BUF_VE;
          ccg_ve_ax_select = SRC_SELECT_X;
          ccg_ve_ay_select = SRC_SELECT_Y;
          ccg_ve_az_select = SRC_SELECT_Z;
          ccg_ve_aw_select = SRC_SELECT_W;
          ccg_ve_b_memsel  = VEB_MEMSEL_STVE_VE;
          ccg_ve_bx_select = SRC_SELECT_X;
          ccg_ve_by_select = SRC_SELECT_Y;
          ccg_ve_bz_select = SRC_SELECT_Z;
          ccg_ve_bw_select = SRC_SELECT_W;
        end

        SMCC_CLIP_CODE_UCP: begin
          ccg_ve_opcode    = VE_DOT_PRODUCT;
          ccg_ve_a_memsel  = VEA_MEMSEL_POS_BUF_VE;
          ccg_ve_ax_select = SRC_SELECT_X;
          ccg_ve_ay_select = SRC_SELECT_Y;
          ccg_ve_az_select = SRC_SELECT_Z;
          ccg_ve_aw_select = SRC_SELECT_W;
          ccg_ve_b_memsel  = VEB_MEMSEL_STVE_VE;
          ccg_ve_bx_select = SRC_SELECT_X;
          ccg_ve_by_select = SRC_SELECT_Y;
          ccg_ve_bz_select = SRC_SELECT_Z;
          ccg_ve_bw_select = SRC_SELECT_W;
          if(ccg_ve_cc_valid_r0) begin
```

```
            ccg_agve_dly_valid_bit_set = CC_VALID;
          end
          ccg_agve_dly_user_clip_indx = ccg_ve_ucp_indx_r0;
          ccg_agve_dly_ucp_cc_test    = 1'b1;
        end

        default;
      endcase
    end
  end

//CLIP_DECODE
always @(clip_plane_indx_r0 or
        clip_dst_vertex_indx_r0 or
        clip_src_vertex_indx_r0 or
        clip_src_vertex_type_r0 or
        clip_ve_ucp_valid_r0 or
        clip_sm_state_indx_r0 or
        clip_state_var_indx_r0 or
        dx_clip_space_def)
  begin

    clip_vte_opcode   = VTE_NO_OP;
    clip_vte_st_indx  = 2'b0;
    clip_ve_opcode    = VECTOR_NO_OP;
```

```
    clip_ve_a_memsel  = VEA_MEMSEL_VEOC_VE;
    clip_ve_b_memsel  = VEB_MEMSEL_STVE_VE;
    clip_ve_ax_select = SRC_SELECT_FORCE_0;
    clip_ve_ay_select = SRC_SELECT_FORCE_0;
    clip_ve_az_select = SRC_SELECT_FORCE_0;
    clip_ve_aw_select = SRC_SELECT_FORCE_0;
    clip_ve_bx_select = SRC_SELECT_FORCE_0;
    clip_ve_by_select = SRC_SELECT_FORCE_0;
    clip_ve_bz_select = SRC_SELECT_FORCE_0;
    clip_ve_bw_select = SRC_SELECT_FORCE_0;
    clip_ve_a_is_wwww    = 1'b0;
    clip_ve_broadcast_x  = 1'b0;
    clip_ve_abs_a        = 1'b0;
    clip_ve_abs_b        = 1'b0;
    clip_ve_abs_c        = 1'b0;
    clip_ve_ax_negate    = 1'b0;
    clip_ve_ay_negate    = 1'b0;
    clip_ve_az_negate    = 1'b0;
    clip_ve_aw_negate    = 1'b0;
    clip_ve_bx_negate    = 1'b0;
    clip_ve_by_negate    = 1'b0;
    clip_ve_bz_negate    = 1'b0;
    clip_ve_bw_negate    = 1'b0;
    clip_ve_cx_negate    = 1'b0;
    clip_ve_cy_negate    = 1'b0;
```

```
clip_ve_cz_negate    = 1'b0;
clip_ve_cw_negate    = 1'b0;
clip_ve_bcc_flat_tst = 1'b0;
clip_ve_out_mem_sel  = 3'b0;
clip_ve_out_addr     = 5'b00000;
clip_ve_out_we       = 4'b0000;
clip_ve_accum_sel    = 1'b0;
clip_ve_pre_acc_we   = 4'b0000;

clip_agve_dly_vertex_store_indx = 6'b000000;
clip_agve_dly_valid_bit_set     = 2'b0;
clip_agve_dly_user_clip_indx    = 4'b0;
clip_agve_dly_vv_cc_test        = 1'b0;
clip_agve_dly_ucp_cc_test       = 1'b0;
clip_agve_dly_bcc_cc_test       = 1'b0;
clip_agve_dly_ps_ucp_cc_test    = 1'b0;
clip_agve_dly_ps_engh_test      = 1'b0;

// GetBarycCoord subroutine
get_baryc_a_memsel  = VEA_MEMSEL_ZERO_FLT;
get_baryc_ax_select = SRC_SELECT_FORCE_0;
get_baryc_ay_select = SRC_SELECT_FORCE_0;
get_baryc_az_select = SRC_SELECT_FORCE_0;
get_baryc_ax_negate = 1'b0;
get_baryc_cx_negate = 1'b0;
```

```

if(clip_src_vertex_type_r0==1'b1) begin
   get_baryc_ax_select = SRC_SELECT_X;
   get_baryc_ay_select = SRC_SELECT_Y;
   get_baryc_az_select = SRC_SELECT_Z;
   get_baryc_a_memsel  = VEA_MEMSEL_VEOC_VE;
end else if(clip_src_vertex_type_r0==1'b0) begin
   case(clip_src_vertex_indx_r0[1:0])
     2'b00: begin
        get_baryc_ax_select = SRC_SELECT_FORCE_1;
        get_baryc_ay_select = SRC_SELECT_FORCE_0;
        get_baryc_az_select = SRC_SELECT_FORCE_0;
     end
     2'b01: begin
        get_baryc_ax_select = SRC_SELECT_FORCE_0;
        get_baryc_ay_select = SRC_SELECT_FORCE_1;
        get_baryc_az_select = SRC_SELECT_FORCE_0;
     end
     2'b10: begin
        get_baryc_ax_select = SRC_SELECT_FORCE_0;
        get_baryc_ay_select = SRC_SELECT_FORCE_0;
        get_baryc_az_select = SRC_SELECT_FORCE_1;
     end
     2'b11: begin
        get_baryc_ax_select = SRC_SELECT_FORCE_1;
```

```
        get_baryc_ax_negate = 1'b1;
        get_baryc_cx_negate = 1'b1;
        get_baryc_ay_select = SRC_SELECT_FORCE_1;
        get_baryc_az_select = SRC_SELECT_FORCE_1;
     end
     default: begin
        get_baryc_a_memsel  = 2'bxx;
        get_baryc_ax_select = 3'bxxx;
        get_baryc_ay_select = 3'bxxx;
        get_baryc_az_select = 3'bxxx;
        get_baryc_ax_negate = 1'bx;
        get_baryc_cx_negate = 1'bx;
     end
   endcase
end else begin
   get_baryc_a_memsel  = 2'bxx;
   get_baryc_ax_select = 3'bxxx;
   get_baryc_ay_select = 3'bxxx;
   get_baryc_az_select = 3'bxxx;
   get_baryc_ax_negate = 1'bx;
   get_baryc_cx_negate = 1'bx;
end
// end GetBarycCoord subroutine

// GetPosition subroutine
```

```
if((clip_src_vertex_type_r0==1'b1) ||
   (clip_src_vertex_indx_r0[6]==1'b1)) begin
   get_pos_a_memsel = VEA_MEMSEL_VEOC_VE;
end else if((clip_src_vertex_type_r0==1'b0) &&
           (clip_src_vertex_indx_r0[6]==1'b0)) begin
   get_pos_a_memsel = VEA_MEMSEL_POS_BUF_VE;
end else begin
   get_pos_a_memsel = 2'bxx;
end
// end GetPosition subroutine

// GetClipDist subroutine
get_clipdist_a_memsel = VEA_MEMSEL_VEOC_VE;
if(clip_src_vertex_type_r0==1'b1) begin
   get_clipdist_a_select = SRC_SELECT_W;
end else if(clip_src_vertex_type_r0==1'b0) begin
   case(clip_src_vertex_indx_r0[1:0])
     2'b00 : begin
        get_clipdist_a_select = SRC_SELECT_X;
     end
     2'b01 : begin
        get_clipdist_a_select = SRC_SELECT_Y;
     end
     2'b10 : begin
        get_clipdist_a_select = SRC_SELECT_Z;
        end
        default : begin
          get_clipdist_a_select = 3'bxxx;
        end
      endcase
    end else begin
      get_clipdist_a_select = 3'bxxx;
    end
    // end GetPosition subroutine

    case (clip_sm_state_indx_r0)

      SMC_OUTPUT_FIRST_BARYC_0,
      SMC_OUTPUT_FIRST_BARYC_1,
      SMC_OUTPUT_FIRST_BARYC_2,
      SMC_OUTPUT_REST_BARYC : begin
        if (clip_ve_ucp_valid_r0) begin
          clip_vte_opcode = VTE_BC_NO_W;
        end else begin
          clip_vte_opcode = VTE_BC_MULT_W;
        end
        // GetBarycCoord
        clip_ve_a_memsel  = get_baryc_a_memsel;
        clip_ve_ax_select = get_baryc_ax_select;
        clip_ve_ay_select = get_baryc_ay_select;
```

```
        clip_ve_az_select = get_baryc_az_select;
        clip_ve_ax_negate = get_baryc_ax_negate;
        clip_ve_cx_negate = get_baryc_cx_negate;
        // end GetBarycCoord
        clip_agve_dly_vertex_store_indx = clip_dst_vertex_indx_r0;
      end

      SMC_OUTPUT_FIRST_CLIP_POS_0,
      SMC_OUTPUT_FIRST_CLIP_POS_1,
      SMC_OUTPUT_FIRST_CLIP_POS_2,
      SMC_OUTPUT_REST_CLIP_POS : begin
        clip_vte_opcode   = VTE_CLIP_POS;
        clip_ve_ax_select = SRC_SELECT_X;
        clip_ve_ay_select = SRC_SELECT_Y;
        clip_ve_az_select = SRC_SELECT_Z;
        clip_ve_aw_select = SRC_SELECT_W;
        clip_ve_a_memsel  = get_pos_a_memsel;
        clip_agve_dly_vertex_store_indx = clip_dst_vertex_indx_r0;
      end

      SMC_T_BLEND_PREV_ABC_0: begin
        clip_ve_opcode     = VE_MULTIPLY;
        clip_ve_pre_acc_we = 4'b0111;
        // GetBarycCoord
        clip_ve_a_memsel  = get_baryc_a_memsel;
```

```
        clip_ve_ax_select = get_baryc_ax_select;
        clip_ve_ay_select = get_baryc_ay_select;
        clip_ve_az_select = get_baryc_az_select;
        clip_ve_ax_negate = get_baryc_ax_negate;
        clip_ve_cx_negate = get_baryc_cx_negate;
        // end GetBarycCoord
        clip_ve_bx_select = SRC_SELECT_X;
        clip_ve_by_select = SRC_SELECT_X;
        clip_ve_bz_select = SRC_SELECT_X;
        clip_ve_bw_select = SRC_SELECT_FORCE_0;
        clip_ve_b_memsel  = VEB_MEMSEL_CLIPTEMP_VEC;
      end

      SMC_T_BLEND_CURR_ABC_0: begin
        clip_ve_opcode       = VE_MULTIPLY_ADD;
        clip_ve_out_we       = 4'b0111;
        clip_ve_accum_sel    = ACCUM_SEL_PRE_ACCUM;
        clip_ve_bcc_flat_tst = 1'b1;
        // GetBarycCoord
        clip_ve_a_memsel  = get_baryc_a_memsel;
        clip_ve_ax_select = get_baryc_ax_select;
        clip_ve_ay_select = get_baryc_ay_select;
        clip_ve_az_select = get_baryc_az_select;
        clip_ve_ax_negate = get_baryc_ax_negate;
        clip_ve_cx_negate = get_baryc_cx_negate;
```

```
        // end GetBarycCoord
        clip_ve_bx_select   = SRC_SELECT_Y;
        clip_ve_by_select   = SRC_SELECT_Y;
        clip_ve_bz_select   = SRC_SELECT_Y;
        clip_ve_bw_select   = SRC_SELECT_FORCE_0;
        clip_ve_out_mem_sel = VE_OUT_VEOC_VECTOR_BACK;
        clip_ve_out_addr    = clip_dst_vertex_indx_r0;
        clip_ve_b_memsel    = VEB_MEMSEL_CLIPTEMP_VEC;
      end

      SMC_T_BLEND_PREV_ABC_1: begin
        clip_ve_opcode     = VE_MULTIPLY;
        clip_ve_pre_acc_we = 4'b0111;
        // GetBarycCoord
        clip_ve_a_memsel  = get_baryc_a_memsel;
        clip_ve_ax_select = get_baryc_ax_select;
        clip_ve_ay_select = get_baryc_ay_select;
        clip_ve_az_select = get_baryc_az_select;
        clip_ve_ax_negate = get_baryc_ax_negate;
        clip_ve_cx_negate = get_baryc_cx_negate;
        // end GetBarycCoord
        clip_ve_bx_select = SRC_SELECT_Z;
        clip_ve_by_select = SRC_SELECT_Z;
        clip_ve_bz_select = SRC_SELECT_Z;
        clip_ve_bw_select = SRC_SELECT_FORCE_0;
```

```
    clip_ve_b_memsel  = VEB_MEMSEL_CLIPTEMP_VEC;
  end

  SMC_T_BLEND_CURR_ABC_1: begin
    clip_ve_opcode      = VE_MULTIPLY_ADD;
    clip_ve_out_we      = 4'b0111;
    clip_ve_accum_sel   = ACCUM_SEL_PRE_ACCUM;
    clip_ve_bcc_flat_tst = 1'b1;
    // GetBarycCoord
    clip_ve_a_memsel  = get_baryc_a_memsel;
    clip_ve_ax_select = get_baryc_ax_select;
    clip_ve_ay_select = get_baryc_ay_select;
    clip_ve_az_select = get_baryc_az_select;
    clip_ve_ax_negate = get_baryc_ax_negate;
    clip_ve_cx_negate = get_baryc_cx_negate;
    // end GetBarycCoord
    clip_ve_bx_select   = SRC_SELECT_W;
    clip_ve_by_select   = SRC_SELECT_W;
    clip_ve_bz_select   = SRC_SELECT_W;
    clip_ve_bw_select   = SRC_SELECT_FORCE_0;
    clip_ve_out_mem_sel = VE_OUT_VEOC_VECTOR_BACK;
    clip_ve_out_addr    = clip_dst_vertex_indx_r0;
    clip_ve_b_memsel    = VEB_MEMSEL_CLIPTEMP_VEC;
  end

```

```
  SMC_T_BLEND_PREV_POS_0: begin
    clip_ve_opcode     = VE_MULTIPLY;
    clip_ve_pre_acc_we = 4'b1111;
    clip_ve_ax_select  = SRC_SELECT_X;
    clip_ve_ay_select  = SRC_SELECT_Y;
    clip_ve_az_select  = SRC_SELECT_Z;
    clip_ve_aw_select  = SRC_SELECT_W;
    // GetPosition
    clip_ve_a_memsel   = get_pos_a_memsel;
    // end GetPosition
    clip_ve_bx_select  = SRC_SELECT_X;
    clip_ve_by_select  = SRC_SELECT_X;
    clip_ve_bz_select  = SRC_SELECT_X;
    clip_ve_bw_select  = SRC_SELECT_X;
    clip_ve_b_memsel   = VEB_MEMSEL_CLIPTEMP_VEC;
  end

  SMC_T_BLEND_CURR_POS_0: begin
    clip_ve_opcode      = VE_MULTIPLY_ADD;
    clip_ve_out_we      = 4'b1111;
    clip_ve_accum_sel   = ACCUM_SEL_PRE_ACCUM;
    clip_ve_bcc_flat_tst = 1'b1;
    clip_ve_ax_select   = SRC_SELECT_X;
    clip_ve_ay_select   = SRC_SELECT_Y;
    clip_ve_az_select   = SRC_SELECT_Z;
```

```
    clip_ve_aw_select   = SRC_SELECT_W;
    // GetPosition
    clip_ve_a_memsel    = get_pos_a_memsel;
    // end GetPosition
    clip_ve_bx_select   = SRC_SELECT_Y;
    clip_ve_by_select   = SRC_SELECT_Y;
    clip_ve_bz_select   = SRC_SELECT_Y;
    clip_ve_bw_select   = SRC_SELECT_Y;
    clip_ve_out_mem_sel  = VE_OUT_VEOC_VECTOR_BACK;
    clip_ve_out_addr    = { 2'b01,
                            clip_dst_vertex_indx_r0[3:0] };
    clip_ve_b_memsel    = VEB_MEMSEL_CLIPTEMP_VEC;
  end

  SMC_T_BLEND_PREV_POS_1: begin
    clip_ve_opcode     = VE_MULTIPLY;
    clip_ve_pre_acc_we = 4'b1111;
    clip_ve_ax_select  = SRC_SELECT_X;
    clip_ve_ay_select  = SRC_SELECT_Y;
    clip_ve_az_select  = SRC_SELECT_Z;
    clip_ve_aw_select  = SRC_SELECT_W;
    // GetPosition
    clip_ve_a_memsel   = get_pos_a_memsel;
    // end GetPosition
    clip_ve_bx_select  = SRC_SELECT_Z;
```

```
    clip_ve_by_select  = SRC_SELECT_Z;
    clip_ve_bz_select  = SRC_SELECT_Z;
    clip_ve_bw_select  = SRC_SELECT_Z;
    clip_ve_b_memsel   = VEB_MEMSEL_CLIPTEMP_VEC;
  end

  SMC_T_BLEND_CURR_POS_1: begin
    clip_ve_opcode      = VE_MULTIPLY_ADD;
    clip_ve_out_we      = 4'b1111;
    clip_ve_accum_sel   = ACCUM_SEL_PRE_ACCUM;
    clip_ve_bcc_flat_tst = 1'b1;
    clip_ve_ax_select   = SRC_SELECT_X;
    clip_ve_ay_select   = SRC_SELECT_Y;
    clip_ve_az_select   = SRC_SELECT_Z;
    clip_ve_aw_select   = SRC_SELECT_W;
    // GetPosition
    clip_ve_a_memsel    = get_pos_a_memsel;
    // end GetPosition
    clip_ve_bx_select   = SRC_SELECT_W;
    clip_ve_by_select   = SRC_SELECT_W;
    clip_ve_bz_select   = SRC_SELECT_W;
    clip_ve_bw_select   = SRC_SELECT_W;
    clip_ve_bw_select   = SRC_SELECT_W;
    clip_ve_out_mem_sel  = VE_OUT_VEOC_VECTOR_BACK;
    clip_ve_out_addr    = { 2'b01,
                      clip_dst_vertex_indx_r0[3:0] };
    clip_ve_b_memsel     = VEB_MEMSEL_CLIPTEMP_VEC;
  end

  SMC_EDGE_DISTANCE_0: begin
    clip_ve_opcode       = VE_MULTIPLY;
    clip_ve_pre_acc_we   = 4'b0001;
    clip_ve_abs_a        = 1'b1;
    // GetClipDist
    clip_ve_ax_select    = get_clipdist_a_select;
    clip_ve_a_memsel     = get_clipdist_a_memsel;
    // end GetClipDist
    clip_ve_bx_select    = SRC_SELECT_FORCE_1;
  end

  SMC_EDGE_DISTANCE_1: begin
    clip_ve_opcode       = VE_MULTIPLY_ADD;
    clip_ve_out_mem_sel  = VE_OUT_INVERSE;
    clip_ve_out_we       = 4'b0001;
    clip_ve_accum_sel    = ACCUM_SEL_PRE_ACCUM;
    clip_ve_abs_a        = 1'b1;
    // GetClipDist
    clip_ve_ax_select    = get_clipdist_a_select;
    clip_ve_a_memsel     = get_clipdist_a_memsel;
    // end GetClipDist
```

```
    clip_ve_bx_select    = SRC_SELECT_FORCE_1;
  end

  SMC_CLIP_T_FACTOR_PREV_0: begin
    clip_ve_opcode       = VE_MULTIPLY;
    clip_ve_out_mem_sel  = VE_OUT_CLIPTEMP_VECTOR;
    clip_ve_out_we       = 4'b0001;
    clip_ve_abs_a        = 1'b1;
    // GetClipDist
    clip_ve_ax_select    = get_clipdist_a_select;
    clip_ve_a_memsel     = get_clipdist_a_memsel;
    // end GetClipDist
    clip_ve_bx_select    = SRC_SELECT_X;
    clip_ve_b_memsel     = VEB_MEMSEL_INV_RET_SC;
  end

  SMC_CLIP_T_FACTOR_CURR_0: begin
    clip_ve_opcode       = VE_MULTIPLY;
    clip_ve_out_mem_sel  = VE_OUT_CLIPTEMP_VECTOR;
    clip_ve_out_we       = 4'b0010;
    clip_ve_abs_a        = 1'b1;
    // GetClipDist
    clip_ve_ay_select    = get_clipdist_a_select;
    clip_ve_a_memsel     = get_clipdist_a_memsel;
    // end GetClipDist
```

```
    clip_ve_by_select    = SRC_SELECT_X;
    clip_ve_b_memsel     = VEB_MEMSEL_INV_RET_SC;
  end

  SMC_CLIP_T_FACTOR_PREV_1: begin
    clip_ve_opcode       = VE_MULTIPLY;
    clip_ve_out_mem_sel  = VE_OUT_CLIPTEMP_VECTOR;
    clip_ve_out_we       = 4'b0100;
    clip_ve_abs_a        = 1'b1;
    // GetClipDist
    clip_ve_az_select    = get_clipdist_a_select;
    clip_ve_a_memsel     = get_clipdist_a_memsel;
    // end GetClipDist
    clip_ve_bz_select    = SRC_SELECT_Y;
    clip_ve_b_memsel     = VEB_MEMSEL_INV_RET_SC;
  end

  SMC_CLIP_T_FACTOR_CURR_1: begin
    clip_ve_opcode       = VE_MULTIPLY;
    clip_ve_out_mem_sel  = VE_OUT_CLIPTEMP_VECTOR;
    clip_ve_out_we       = 4'b1000;
    clip_ve_abs_a        = 1'b1;
    // GetClipDist
    clip_ve_aw_select    = get_clipdist_a_select;
    clip_ve_a_memsel     = get_clipdist_a_memsel;
```

```
    // end GetClipDist
    clip_ve_bw_select    = SRC_SELECT_Y;
    clip_ve_b_memsel     = VEB_MEMSEL_INV_RET_SC;
  end

  SMC_CLIP_DIST_VV: begin
    clip_ve_opcode       = VE_MULTIPLY_ADD;
    clip_ve_a_is_wwww    = 1'b1;
    clip_ve_broadcast_x = 1'b1;
    clip_ve_out_mem_sel = VE_OUT_VEOC_VECTOR_BACK;
    if (clip_src_vertex_type_r0) begin
      clip_ve_out_we                  = 4'b1000;
      clip_agve_dly_bcc_cc_test       = 1'b1;
      clip_agve_dly_vertex_store_indx = clip_dst_vertex_indx_r0;
      clip_ve_out_addr = { 2'b00,
                           clip_dst_vertex_indx_r0[3:0] };
    end else begin
      case(clip_dst_vertex_indx_r0[1:0])
        2'b00 : begin
          clip_ve_out_we = 4'b0001;
        end
        2'b01 : begin
          clip_ve_out_we = 4'b0010;
        end
        2'b10 : begin
```

```
       clip_ve_out_we = 4'b0100;
     end
     default : begin
       clip_ve_out_we = 4'bxxxx;
     end
    endcase
    clip_ve_out_addr = VEOC_CLIP_DIST_ORIG;
  end
  case(clip_plane_indx_r0[2:0])
   CC_CLIP_FAR,
   CC_CLIP_RIGHT,
   CC_CLIP_TOP : begin
     clip_ve_cx_negate = 1'b1;
   end
   CC_CLIP_NEAR,
   CC_CLIP_LEFT,
   CC_CLIP_BOTTOM : begin
     clip_ve_cx_negate = 1'b0;
   end
   default : begin
     clip_ve_cx_negate = 1'bx;
   end
  endcase
  case(clip_plane_indx_r0[2:0])
   CC_CLIP_NEAR : begin
```

```
     if(dx_clip_space_def) begin
       clip_ve_bx_select = SRC_SELECT_FORCE_0;
     end else begin
       clip_ve_bx_select = SRC_SELECT_FORCE_1;
     end
     clip_ve_ax_select = SRC_SELECT_Z;
   end
   CC_CLIP_FAR : begin
     clip_ve_bx_select = SRC_SELECT_FORCE_1;
     clip_ve_ax_select = SRC_SELECT_Z;
   end
   CC_CLIP_LEFT,
   CC_CLIP_RIGHT : begin
     clip_ve_bx_select = SRC_SELECT_X;
     clip_ve_ax_select = SRC_SELECT_X;
   end
   CC_CLIP_BOTTOM,
   CC_CLIP_TOP : begin
     clip_ve_bx_select = SRC_SELECT_Y;
     clip_ve_ax_select = SRC_SELECT_Y;
   end
   default : begin
     clip_ve_bx_select = 3'bxxx;
     clip_ve_ax_select = 3'bxxx;
   end
```

```
  endcase
  clip_ve_aw_select = SRC_SELECT_W;
  // GetPosition
  clip_ve_a_memsel  = get_pos_a_memsel;
  // end GetPosition
  clip_ve_b_memsel  = VEB_MEMSEL_STVE_VE;
end

SMC_CLIP_DIST_UCP: begin
  clip_ve_opcode      = VE_DOT_PRODUCT;
  clip_ve_out_mem_sel = VE_OUT_VEOC_VECTOR_BACK;
  if (clip_src_vertex_type_r0) begin
    clip_ve_out_we                 = 4'b1000;
    clip_agve_dly_bcc_cc_test      = 1'b1;
    clip_agve_dly_vertex_store_indx = clip_dst_vertex_indx_r0;
    clip_ve_out_addr = { 2'b00,
                    clip_dst_vertex_indx_r0[3:0] };
  end else begin
    case(clip_dst_vertex_indx_r0[1:0])
     2'b00 : begin
       clip_ve_out_we = 4'b0001;
     end
     2'b01 : begin
       clip_ve_out_we = 4'b0010;
     end
```

```
     2'b10 : begin
       clip_ve_out_we = 4'b0100;
     end
     default : begin
       clip_ve_out_we = 4'bxxxx;
     end
    endcase
    clip_ve_out_addr = VEOC_CLIP_DIST_ORIG;
  end
  clip_ve_ax_select = SRC_SELECT_X;
  clip_ve_ay_select = SRC_SELECT_Y;
  clip_ve_az_select = SRC_SELECT_Z;
  clip_ve_aw_select = SRC_SELECT_W;
  clip_ve_bx_select = SRC_SELECT_X;
  clip_ve_by_select = SRC_SELECT_Y;
  clip_ve_bz_select = SRC_SELECT_Z;
  clip_ve_bw_select = SRC_SELECT_W;
  clip_agve_dly_user_clip_indx = clip_plane_indx_r0;
  // GetPosition
  clip_ve_a_memsel  = get_pos_a_memsel;
  // end GetPosition
  clip_ve_b_memsel  = VEB_MEMSEL_STVE_VE;
end

SMC_PS_CULL_RADIUS_VPORT: begin
```

```
    clip_ve_opcode       = VE_MULTIPLY;
    clip_ve_out_mem_sel  = VE_OUT_CLIPTEMP_VECTOR;
    clip_ve_out_we       = 4'b0010;
    clip_ve_a_memsel     = VEA_MEMSEL_POS_BUF_VE;
    clip_ve_b_memsel     = VEB_MEMSEL_STVE_VE;
    clip_ve_ay_select    = SRC_SELECT_W;
    clip_ve_by_select    = SRC_SELECT_W;
  end

  SMC_PS_CULL_RADIUS_W: begin
    clip_ve_opcode       = VE_MULTIPLY;
    clip_ve_out_mem_sel  = VE_OUT_VEOC_VECTOR_BACK;
    clip_ve_out_addr     = VEOC_CLIP_DIST_ORIG;
    clip_ve_out_we       = 4'b0010;
    clip_ve_a_memsel     = VEA_MEMSEL_POINT_BUF_SC;
    clip_ve_b_memsel     = VEB_MEMSEL_CLIPTEMP_VEC;
    clip_ve_ay_select    = SRC_SELECT_X;
    clip_ve_by_select    = SRC_SELECT_Y;
  end

  SMC_PS_CLIP_DIST_UCP: begin
    clip_ve_opcode       = VE_DOT_PRODUCT;
    clip_ve_ax_select    = SRC_SELECT_X;
    clip_ve_ay_select    = SRC_SELECT_Y;
    clip_ve_az_select    = SRC_SELECT_Z;
```

```
    clip_ve_aw_select    = SRC_SELECT_W;
    clip_ve_bx_select    = SRC_SELECT_X;
    clip_ve_by_select    = SRC_SELECT_Y;
    clip_ve_bz_select    = SRC_SELECT_Z;
    clip_ve_bw_select    = SRC_SELECT_W;
    clip_ve_a_memsel     = VEA_MEMSEL_POS_BUF_VE;
    clip_ve_b_memsel     = VEB_MEMSEL_STVE_VE;
    clip_ve_out_mem_sel  = VE_OUT_VEOC_VECTOR_BACK;
    clip_ve_out_addr     = VEOC_CLIP_DIST_ORIG;
    clip_ve_out_we       = 4'b0001;
  end

  SMC_PS_XY_RADIUS_VPORT: begin
    clip_ve_opcode       = VE_MULTIPLY;
    clip_ve_a_memsel     = VEA_MEMSEL_POS_BUF_VE;
    clip_ve_b_memsel     = VEB_MEMSEL_STVE_VE;
    clip_ve_ax_select    = SRC_SELECT_W;
    clip_ve_ay_select    = SRC_SELECT_W;
    clip_ve_bx_select    = SRC_SELECT_X;
    clip_ve_by_select    = SRC_SELECT_Y;
    clip_ve_out_mem_sel  = VE_OUT_CLIPTEMP_VECTOR;
    clip_ve_out_we       = 4'b0011;
  end

  SMC_PS_XY_RADIUS_W: begin
```

```
    clip_ve_opcode       = VE_MULTIPLY;
    clip_ve_a_memsel     = VEA_MEMSEL_POINT_BUF_SC;
    clip_ve_b_memsel     = VEB_MEMSEL_CLIPTEMP_VEC;
    clip_ve_ax_select    = SRC_SELECT_X;
    clip_ve_ay_select    = SRC_SELECT_X;
    clip_ve_bx_select    = SRC_SELECT_X;
    clip_ve_by_select    = SRC_SELECT_Y;
    clip_ve_out_mem_sel  = VE_OUT_CLIPTEMP_VECTOR;
    clip_ve_out_we       = 4'b0011;
  end

  SMC_PS_POS_UL: begin
    clip_ve_opcode       = VE_ADD;
    clip_ve_a_memsel     = VEA_MEMSEL_POS_BUF_VE;
    clip_ve_b_memsel     = VEB_MEMSEL_CLIPTEMP_VEC;
    clip_ve_ax_select    = SRC_SELECT_X;
    clip_ve_ay_select    = SRC_SELECT_Y;
    clip_ve_az_select    = SRC_SELECT_Z;
    clip_ve_aw_select    = SRC_SELECT_W;
    clip_ve_bx_select    = SRC_SELECT_X;
    clip_ve_by_select    = SRC_SELECT_Y;
    clip_ve_bx_negate    = 1'b1;
    clip_ve_out_mem_sel  = VE_OUT_VEOC_VECTOR_BACK;
    clip_ve_out_addr     = VEOC_PS_POS_VERT_UL;
    clip_ve_out_we       = 4'b1111;
```

```
  end

  SMC_PS_POS_UR: begin
    clip_ve_opcode       = VE_ADD;
    clip_ve_a_memsel     = VEA_MEMSEL_POS_BUF_VE;
    clip_ve_b_memsel     = VEB_MEMSEL_CLIPTEMP_VEC;
    clip_ve_ax_select    = SRC_SELECT_X;
    clip_ve_ay_select    = SRC_SELECT_Y;
    clip_ve_az_select    = SRC_SELECT_Z;
    clip_ve_aw_select    = SRC_SELECT_W;
    clip_ve_bx_select    = SRC_SELECT_X;
    clip_ve_by_select    = SRC_SELECT_Y;
    clip_ve_out_mem_sel  = VE_OUT_VEOC_VECTOR_BACK;
    clip_ve_out_addr     = VEOC_PS_POS_VERT_UR;
    clip_ve_out_we       = 4'b1111;
  end

  SMC_PS_POS_LR: begin
    clip_ve_opcode       = VE_ADD;
    clip_ve_a_memsel     = VEA_MEMSEL_POS_BUF_VE;
    clip_ve_b_memsel     = VEB_MEMSEL_CLIPTEMP_VEC;
    clip_ve_ax_select    = SRC_SELECT_X;
    clip_ve_ay_select    = SRC_SELECT_Y;
    clip_ve_az_select    = SRC_SELECT_Z;
    clip_ve_aw_select    = SRC_SELECT_W;
    clip_ve_bx_select    = SRC_SELECT_X;
    clip_ve_by_select    = SRC_SELECT_Y;
    clip_ve_by_negate    = 1'b1;
    clip_ve_out_mem_sel  = VE_OUT_VEOC_VECTOR_BACK;
    clip_ve_out_addr     = VEOC_PS_POS_VERT_LR;
    clip_ve_out_we       = 4'b1111;
  end

  SMC_PS_POS_LL: begin
    clip_ve_opcode       = VE_ADD;
    clip_ve_a_memsel     = VEA_MEMSEL_POS_BUF_VE;
    clip_ve_b_memsel     = VEB_MEMSEL_CLIPTEMP_VEC;
    clip_ve_ax_select    = SRC_SELECT_X;
    clip_ve_ay_select    = SRC_SELECT_Y;
    clip_ve_az_select    = SRC_SELECT_Z;
    clip_ve_aw_select    = SRC_SELECT_W;
    clip_ve_bx_select    = SRC_SELECT_X;
    clip_ve_by_select    = SRC_SELECT_Y;
    clip_ve_bx_negate    = 1'b1;
    clip_ve_by_negate    = 1'b1;
    clip_ve_out_mem_sel  = VE_OUT_VEOC_VECTOR_BACK;
    clip_ve_out_addr     = VEOC_PS_POS_VERT_LL;
    clip_ve_out_we       = 4'b1111;
  end

```

```
  SMC_PS_UCP_DIST_UL: begin
    clip_ve_opcode       = VE_DOT_PRODUCT;
    clip_ve_a_memsel     = VEA_MEMSEL_VEOC_VE;
    clip_ve_b_memsel     = VEB_MEMSEL_STVE_VE;
    clip_ve_ax_select    = SRC_SELECT_X;
    clip_ve_ay_select    = SRC_SELECT_Y;
    clip_ve_az_select    = SRC_SELECT_Z;
    clip_ve_aw_select    = SRC_SELECT_W;
    clip_ve_bx_select    = SRC_SELECT_X;
    clip_ve_by_select    = SRC_SELECT_Y;
    clip_ve_bz_select    = SRC_SELECT_Z;
    clip_ve_bw_select    = SRC_SELECT_W;
    clip_agve_dly_ps_ucp_cc_test    = 1'b1;
    clip_agve_dly_vertex_store_indx = { 4'b0000,
                                        PS_VERT_UL };
    clip_agve_dly_user_clip_indx    = clip_plane_indx_r0;
  end

  SMC_PS_UCP_DIST_UR: begin
    clip_ve_opcode       = VE_DOT_PRODUCT;
    clip_ve_a_memsel     = VEA_MEMSEL_VEOC_VE;
    clip_ve_b_memsel     = VEB_MEMSEL_STVE_VE;
    clip_ve_ax_select    = SRC_SELECT_X;
    clip_ve_ay_select    = SRC_SELECT_Y;
    clip_ve_az_select    = SRC_SELECT_Z;
```

```
    clip_ve_aw_select    = SRC_SELECT_W;
    clip_ve_bx_select    = SRC_SELECT_X;
    clip_ve_by_select    = SRC_SELECT_Y;
    clip_ve_bz_select    = SRC_SELECT_Z;
    clip_ve_bw_select    = SRC_SELECT_W;
    clip_agve_dly_ps_ucp_cc_test    = 1'b1;
    clip_agve_dly_vertex_store_indx = { 4'b0000,
                                        PS_VERT_UR };
    clip_agve_dly_user_clip_indx    = clip_plane_indx_r0;
  end

  SMC_PS_UCP_DIST_LR: begin
    clip_ve_opcode       = VE_DOT_PRODUCT;
    clip_ve_a_memsel     = VEA_MEMSEL_VEOC_VE;
    clip_ve_b_memsel     = VEB_MEMSEL_STVE_VE;
    clip_ve_ax_select    = SRC_SELECT_X;
    clip_ve_ay_select    = SRC_SELECT_Y;
    clip_ve_az_select    = SRC_SELECT_Z;
    clip_ve_aw_select    = SRC_SELECT_W;
    clip_ve_bx_select    = SRC_SELECT_X;
    clip_ve_by_select    = SRC_SELECT_Y;
    clip_ve_bz_select    = SRC_SELECT_Z;
    clip_ve_bw_select    = SRC_SELECT_W;
    clip_agve_dly_ps_ucp_cc_test    = 1'b1;
    clip_agve_dly_vertex_store_indx = { 4'b0000,
```

```
                                        PS_VERT_LR };
    clip_agve_dly_user_clip_indx    = clip_plane_indx_r0;
  end

  SMC_PS_UCP_DIST_LL: begin
    clip_ve_opcode       = VE_DOT_PRODUCT;
    clip_ve_a_memsel     = VEA_MEMSEL_VEOC_VE;
    clip_ve_b_memsel     = VEB_MEMSEL_STVE_VE;
    clip_ve_ax_select    = SRC_SELECT_X;
    clip_ve_ay_select    = SRC_SELECT_Y;
    clip_ve_az_select    = SRC_SELECT_Z;
    clip_ve_aw_select    = SRC_SELECT_W;
    clip_ve_bx_select    = SRC_SELECT_X;
    clip_ve_by_select    = SRC_SELECT_Y;
    clip_ve_bz_select    = SRC_SELECT_Z;
    clip_ve_bw_select    = SRC_SELECT_W;
    clip_agve_dly_ps_ucp_cc_test    = 1'b1;
    clip_agve_dly_vertex_store_indx = { 4'b0000,
                                        PS_VERT_LL };
    clip_agve_dly_user_clip_indx    = clip_plane_indx_r0;
    if (clip_ve_ucp_valid_r0) begin
      clip_agve_dly_valid_bit_set  = VE_PS_UCP_VALID;
    end
  end

```

```
      SMC_PS_ENGH_TEST: begin
        clip_ve_opcode                = VE_CLIP_CODE_VV_ZWW;
        clip_ve_a_is_wwww             = 1'b1;
        clip_agve_dly_ps_engh_test     = 1'b1;
        clip_agve_dly_user_clip_indx    = clip_plane_indx_r0;
        clip_agve_dly_vertex_store_indx = clip_src_vertex_indx_r0;
        clip_ve_az_select             = SRC_SELECT_X;
        clip_ve_aw_select              = SRC_SELECT_Y;
        clip_ve_a_memsel               = VEA_MEMSEL_VEOC_VE;
        if (clip_ve_ucp_valid_r0) begin
          clip_agve_dly_valid_bit_set  = VE_PS_ENGH_VALID;
        end
      end

      default;
    endcase
  end

//Arbiter selection of CCG or CLIP command
always @(
        ccg_vte_opcode_r1 or
          ccg_vte_st_indx_r1 or
        ccg_ve_opcode_r1 or
        ccg_ve_a_memsel_r1 or
        ccg_ve_b_memsel_r1 or
```

```
        ccg_ve_ax_select_r1 or
        ccg_ve_ay_select_r1 or
        ccg_ve_az_select_r1 or
        ccg_ve_aw_select_r1 or
        ccg_ve_bx_select_r1 or
        ccg_ve_by_select_r1 or
        ccg_ve_bz_select_r1 or
        ccg_ve_bw_select_r1 or
        ccg_ve_a_is_wwww_r1 or
        ccg_ve_broadcast_x_r1 or
        ccg_ve_abs_a_r1 or
        ccg_ve_abs_b_r1 or
        ccg_ve_abs_c_r1 or
        ccg_ve_ax_negate_r1 or
        ccg_ve_ay_negate_r1 or
        ccg_ve_az_negate_r1 or
        ccg_ve_aw_negate_r1 or
        ccg_ve_bx_negate_r1 or
        ccg_ve_by_negate_r1 or
        ccg_ve_bz_negate_r1 or
        ccg_ve_bw_negate_r1 or
        ccg_ve_cx_negate_r1 or
        ccg_ve_cy_negate_r1 or
        ccg_ve_cz_negate_r1 or
        ccg_ve_cw_negate_r1 or
```

```
        ccg_ve_bcc_flat_tst_r1 or
        ccg_ve_out_mem_sel_r1 or
        ccg_ve_out_addr_r1 or
        ccg_ve_out_we_r1 or
        ccg_ve_accum_sel_r1 or
        ccg_ve_pre_acc_we_r1 or
        ccg_agve_dly_vertex_store_indx_r1 or
        ccg_agve_dly_valid_bit_set_r1 or
        ccg_agve_dly_user_clip_indx_r1 or
        ccg_agve_dly_vv_cc_test_r1 or
        ccg_agve_dly_ucp_cc_test_r1 or
        ccg_agve_dly_bcc_cc_test_r1 or
        ccg_agve_dly_ps_ucp_cc_test_r1 or
        ccg_agve_dly_ps_engh_test_r1 or

        clip_xfc_r1 or
        clip_vte_opcode_r1 or
        clip_vte_st_indx_r1 or
        clip_ve_opcode_r1 or
        clip_ve_a_memsel_r1 or
        clip_ve_b_memsel_r1 or
        clip_ve_ax_select_r1 or
        clip_ve_ay_select_r1 or
        clip_ve_az_select_r1 or
        clip_ve_aw_select_r1 or
```

```
        clip_ve_bx_select_r1 or
        clip_ve_by_select_r1 or
        clip_ve_bz_select_r1 or
        clip_ve_bw_select_r1 or
        clip_ve_a_is_wwww_r1 or
        clip_ve_broadcast_x_r1 or
        clip_ve_abs_a_r1 or
        clip_ve_abs_b_r1 or
        clip_ve_abs_c_r1 or
        clip_ve_ax_negate_r1 or
        clip_ve_ay_negate_r1 or
        clip_ve_az_negate_r1 or
        clip_ve_aw_negate_r1 or
        clip_ve_bx_negate_r1 or
        clip_ve_by_negate_r1 or
        clip_ve_bz_negate_r1 or
        clip_ve_bw_negate_r1 or
        clip_ve_cx_negate_r1 or
        clip_ve_cy_negate_r1 or
        clip_ve_cz_negate_r1 or
        clip_ve_cw_negate_r1 or
        clip_ve_bcc_flat_tst_r1 or
        clip_ve_out_mem_sel_r1 or
        clip_ve_out_addr_r1 or
        clip_ve_out_we_r1 or
        clip_ve_accum_sel_r1 or
        clip_ve_pre_acc_we_r1 or
        clip_agve_dly_vertex_store_indx_r1 or
        clip_agve_dly_valid_bit_set_r1 or
        clip_agve_dly_user_clip_indx_r1 or
        clip_agve_dly_vv_cc_test_r1 or
        clip_agve_dly_ucp_cc_test_r1 or
        clip_agve_dly_bcc_cc_test_r1 or
        clip_agve_dly_ps_ucp_cc_test_r1 or
        clip_agve_dly_ps_engh_test_r1
       )
   begin

     arbsel_vte_opcode              <= ccg_vte_opcode_r1;
     arbsel_vte_st_indx             <= ccg_vte_st_indx_r1;
     arbsel_ve_opcode               <= ccg_ve_opcode_r1;
     arbsel_ve_a_memsel             <= ccg_ve_a_memsel_r1;
     arbsel_ve_b_memsel             <= ccg_ve_b_memsel_r1;
     arbsel_ve_ax_select            <= ccg_ve_ax_select_r1;
     arbsel_ve_ay_select            <= ccg_ve_ay_select_r1;
     arbsel_ve_az_select            <= ccg_ve_az_select_r1;
     arbsel_ve_aw_select            <= ccg_ve_aw_select_r1;
     arbsel_ve_bx_select            <= ccg_ve_bx_select_r1;
     arbsel_ve_by_select            <= ccg_ve_by_select_r1;
     arbsel_ve_bz_select            <= ccg_ve_bz_select_r1;
```

```
     arbsel_ve_bw_select            <= ccg_ve_bw_select_r1;
     arbsel_ve_a_is_wwww            <= ccg_ve_a_is_wwww_r1;
     arbsel_ve_broadcast_x          <= ccg_ve_broadcast_x_r1;
     arbsel_ve_abs_a                <= ccg_ve_abs_a_r1;
     arbsel_ve_abs_b                <= ccg_ve_abs_b_r1;
     arbsel_ve_abs_c                <= ccg_ve_abs_c_r1;
     arbsel_ve_ax_negate            <= ccg_ve_ax_negate_r1;
     arbsel_ve_ay_negate            <= ccg_ve_ay_negate_r1;
     arbsel_ve_az_negate            <= ccg_ve_az_negate_r1;
     arbsel_ve_aw_negate            <= ccg_ve_aw_negate_r1;
     arbsel_ve_bx_negate            <= ccg_ve_bx_negate_r1;
     arbsel_ve_by_negate            <= ccg_ve_by_negate_r1;
     arbsel_ve_bz_negate            <= ccg_ve_bz_negate_r1;
     arbsel_ve_bw_negate            <= ccg_ve_bw_negate_r1;
     arbsel_ve_cx_negate            <= ccg_ve_cx_negate_r1;
     arbsel_ve_cy_negate            <= ccg_ve_cy_negate_r1;
     arbsel_ve_cz_negate            <= ccg_ve_cz_negate_r1;
     arbsel_ve_cw_negate            <= ccg_ve_cw_negate_r1;
     arbsel_ve_bcc_flat_tst         <= ccg_ve_bcc_flat_tst_r1;
     arbsel_ve_out_mem_sel          <= ccg_ve_out_mem_sel_r1;
     arbsel_ve_out_addr             <= ccg_ve_out_addr_r1;
     arbsel_ve_out_we               <= ccg_ve_out_we_r1;
     arbsel_ve_accum_sel            <= ccg_ve_accum_sel_r1;
     arbsel_ve_pre_acc_we           <= ccg_ve_pre_acc_we_r1;
     arbsel_agve_dly_vertex_store_indx <= ccg_agve_dly_vertex_store_indx_r1;
```

```
     arbsel_agve_dly_valid_bit_set      <= ccg_agve_dly_valid_bit_set_r1;
     arbsel_agve_dly_user_clip_indx     <= ccg_agve_dly_user_clip_indx_r1;
     arbsel_agve_dly_vv_cc_test         <= ccg_agve_dly_vv_cc_test_r1;
     arbsel_agve_dly_ucp_cc_test        <= ccg_agve_dly_ucp_cc_test_r1;
     arbsel_agve_dly_bcc_cc_test        <= ccg_agve_dly_bcc_cc_test_r1;
     arbsel_agve_dly_ps_ucp_cc_test     <= ccg_agve_dly_ps_ucp_cc_test_r1;
     arbsel_agve_dly_ps_engh_test       <= ccg_agve_dly_ps_engh_test_r1;
     if (clip_xfc_r1) begin
       arbsel_vte_opcode              <= clip_vte_opcode_r1;
       arbsel_vte_st_indx             <= clip_vte_st_indx_r1;
       arbsel_ve_opcode               <= clip_ve_opcode_r1;
       arbsel_ve_a_memsel             <= clip_ve_a_memsel_r1;
       arbsel_ve_b_memsel             <= clip_ve_b_memsel_r1;
       arbsel_ve_ax_select            <= clip_ve_ax_select_r1;
       arbsel_ve_ay_select            <= clip_ve_ay_select_r1;
       arbsel_ve_az_select            <= clip_ve_az_select_r1;
       arbsel_ve_aw_select            <= clip_ve_aw_select_r1;
       arbsel_ve_bx_select            <= clip_ve_bx_select_r1;
       arbsel_ve_by_select            <= clip_ve_by_select_r1;
       arbsel_ve_bz_select            <= clip_ve_bz_select_r1;
       arbsel_ve_bw_select            <= clip_ve_bw_select_r1;
       arbsel_ve_a_is_wwww            <= clip_ve_a_is_wwww_r1;
       arbsel_ve_broadcast_x          <= clip_ve_broadcast_x_r1;
       arbsel_ve_abs_a                <= clip_ve_abs_a_r1;
       arbsel_ve_abs_b                <= clip_ve_abs_b_r1;
```

```
       arbsel_ve_abs_c                <= clip_ve_abs_c_r1;
       arbsel_ve_ax_negate            <= clip_ve_ax_negate_r1;
       arbsel_ve_ay_negate            <= clip_ve_ay_negate_r1;
       arbsel_ve_az_negate            <= clip_ve_az_negate_r1;
       arbsel_ve_aw_negate            <= clip_ve_aw_negate_r1;
       arbsel_ve_bx_negate            <= clip_ve_bx_negate_r1;
       arbsel_ve_by_negate            <= clip_ve_by_negate_r1;
       arbsel_ve_bz_negate            <= clip_ve_bz_negate_r1;
       arbsel_ve_bw_negate            <= clip_ve_bw_negate_r1;
       arbsel_ve_cx_negate            <= clip_ve_cx_negate_r1;
       arbsel_ve_cy_negate            <= clip_ve_cy_negate_r1;
       arbsel_ve_cz_negate            <= clip_ve_cz_negate_r1;
       arbsel_ve_cw_negate            <= clip_ve_cw_negate_r1;
       arbsel_ve_bcc_flat_tst         <= clip_ve_bcc_flat_tst_r1;
       arbsel_ve_out_mem_sel          <= clip_ve_out_mem_sel_r1;
       arbsel_ve_out_addr             <= clip_ve_out_addr_r1;
       arbsel_ve_out_we               <= clip_ve_out_we_r1;
       arbsel_ve_accum_sel            <= clip_ve_accum_sel_r1;
       arbsel_ve_pre_acc_we           <= clip_ve_pre_acc_we_r1;
       arbsel_agve_dly_vertex_store_indx <= clip_agve_dly_vertex_store_indx_r1;
       arbsel_agve_dly_valid_bit_set      <= clip_agve_dly_valid_bit_set_r1;
       arbsel_agve_dly_user_clip_indx     <= clip_agve_dly_user_clip_indx_r1;
       arbsel_agve_dly_vv_cc_test         <= clip_agve_dly_vv_cc_test_r1;
       arbsel_agve_dly_ucp_cc_test        <= clip_agve_dly_ucp_cc_test_r1;
       arbsel_agve_dly_bcc_cc_test        <= clip_agve_dly_bcc_cc_test_r1;
```

```
      arbsel_agve_dly_ps_ucp_cc_test    <= clip_agve_dly_ps_ucp_cc_test_r1;
      arbsel_agve_dly_ps_engh_test      <= clip_agve_dly_ps_engh_test_r1;
    end
  end

always @(ve_a_memsel_r2 or
        ve_veoc_vector_back_rdata_r2 or
        pos_rdata_r2 or
        pntsz_rdata_r2)
  begin
    amem_sel_data = 128'h0;

    case(ve_a_memsel_r2)
      VEA_MEMSEL_VEOC_VE: begin
        amem_sel_data = ve_veoc_vector_back_rdata_r2;
      end
      VEA_MEMSEL_POS_BUF_VE: begin
        amem_sel_data = pos_rdata_r2;
      end
      VEA_MEMSEL_POINT_BUF_SC: begin
        amem_sel_data = pntsz_rdata_r2;
      end
      VEA_MEMSEL_ZERO_FLT: begin
        amem_sel_data = 128'h0;
      end
```

```
      default;
    endcase
  end

always @(ve_b_memsel_r2 or
        stve_rdata_r2 or
        ve_cliptemp_vec or
        inv_ret_sc_data)
  begin
    bmem_sel_data = 128'h0;

    case(ve_b_memsel_r2)
      VEB_MEMSEL_STVE_VE: begin
        bmem_sel_data = stve_rdata_r2;
      end
      VEB_MEMSEL_CLIPTEMP_VEC: begin
        bmem_sel_data = ve_cliptemp_vec;
      end
      VEB_MEMSEL_INV_RET_SC: begin
        bmem_sel_data = {64'h0,inv_ret_sc_data};
      end
      VEB_MEMSEL_ZERO_FLT: begin
        bmem_sel_data = 128'h0;
      end
      default;
```

```
    endcase
  end

//swizzle data
assign agswz_ve_in_a0 = swizzle(ve_ax_select_r2, amem_sel_data);
assign agswz_ve_in_a1 = swizzle(ve_ay_select_r2, amem_sel_data);
assign agswz_ve_in_a2 = swizzle(ve_az_select_r2, amem_sel_data);
assign agswz_ve_in_a3 = swizzle(ve_aw_select_r2, amem_sel_data);
assign agswz_ve_in_b0 = swizzle(ve_bx_select_r2, bmem_sel_data);
assign agswz_ve_in_b1 = swizzle(ve_by_select_r2, bmem_sel_data);
assign agswz_ve_in_b2 = swizzle(ve_bz_select_r2, bmem_sel_data);
assign agswz_ve_in_b3 = swizzle(ve_bw_select_r2, bmem_sel_data);

always @(ve_opcode_r2)
  begin
    agve_valid_op = (ve_opcode_r2!=0);
  end


//*********************************************************************
// Synchronous Section
//*********************************************************************

//register
always @(posedge sclk)
```

```
  begin
    clip_xfc_r0  <= arb_clip_xfc;

    //register ccg values for decode purposes
    ccg_xfc_r0               <= arb_ccg_xfc;
    ccg_state_var_indx_r0    <= ccg_state_var_indx;
    ccg_ve_ucp_indx_r0       <= ccg_ve_ucp_indx;
    ccg_ve_cc_valid_r0       <= ccg_ve_cc_valid;
    ccg_vertex_store_indx_r0 <= ccg_vertex_store_indx;
    ccg_sm_state_indx_r0     <= ccg_sm_state_indx;

    clip_plane_indx_r0       <= clip_plane_indx;
    clip_dst_vertex_indx_r0  <= clip_dst_vertex_indx;
    clip_src_vertex_indx_r0  <= clip_src_vertex_indx;
    clip_src_vertex_type_r0  <= clip_src_vertex_type;
    clip_ve_ucp_valid_r0     <= clip_ve_ucp_valid;
    clip_sm_state_indx_r0    <= clip_sm_state_indx;
    clip_state_var_indx_r0   <= clip_state_var_indx;

    arb_state_var_indx_r0    <= nxt_arb_state_var_indx;
    stve_re_r0               <= nxt_stve_re;
    state_type_r0            <= nxt_state_type;

    //Implement the ve_cliptemp_vector register
    if (ve_cliptemp_vector_we[0] == 1'b1) begin
```

```
   ve_cliptemp_vec[31:0] <= ve_wdata[31:0];
end
if (ve_cliptemp_vector_we[1] == 1'b1) begin
   ve_cliptemp_vec[63:32] <= ve_wdata[63:32];
end
if (ve_cliptemp_vector_we[2] == 1'b1) begin
   ve_cliptemp_vec[95:64] <= ve_wdata[95:64];
end
if (ve_cliptemp_vector_we[3] == 1'b1) begin
   ve_cliptemp_vec[127:96] <= ve_wdata[127:96];
end

//maintian state management ptr's and dirty bits
ucp0_write_ptr        <= nxt_ucp0_write_ptr;
ucp0_write_after_cpy  <= nxt_ucp0_write_after_cpy;
ucp1_write_ptr        <= nxt_ucp1_write_ptr;
ucp1_write_after_cpy  <= nxt_ucp1_write_after_cpy;
ucp2_write_ptr        <= nxt_ucp2_write_ptr;
ucp2_write_after_cpy  <= nxt_ucp2_write_after_cpy;
ucp3_write_ptr        <= nxt_ucp3_write_ptr;
ucp3_write_after_cpy  <= nxt_ucp3_write_after_cpy;
ucp4_write_ptr        <= nxt_ucp4_write_ptr;
ucp4_write_after_cpy  <= nxt_ucp4_write_after_cpy;
ucp5_write_ptr        <= nxt_ucp5_write_ptr;
ucp5_write_after_cpy  <= nxt_ucp5_write_after_cpy;
```

```
gb_write_ptr          <= nxt_gb_write_ptr;
gb_write_after_cpy    <= nxt_gb_write_after_cpy;
pntsz_write_ptr       <= nxt_pntsz_write_ptr;
pntsz_write_after_cpy <= nxt_pntsz_write_after_cpy;

//AG_R1 register
clip_xfc_r1       <= clip_xfc_r0;
clip_ve_opcode_r1 <= clip_ve_opcode;

ccg_xfc_r1        <= ccg_xfc_r0;
ccg_ve_opcode_r1  <= ccg_ve_opcode;
//CCG Decode R1 Delay
ccg_vte_opcode_r1              <= ccg_vte_opcode;
ccg_vte_st_indx_r1             <= ccg_vte_st_indx;
ccg_ve_opcode_r1               <= ccg_ve_opcode;
ccg_ve_a_memsel_r1             <= ccg_ve_a_memsel;
ccg_ve_b_memsel_r1             <= ccg_ve_b_memsel;
ccg_ve_ax_select_r1            <= ccg_ve_ax_select;
ccg_ve_ay_select_r1            <= ccg_ve_ay_select;
ccg_ve_az_select_r1            <= ccg_ve_az_select;
ccg_ve_aw_select_r1            <= ccg_ve_aw_select;
ccg_ve_bx_select_r1            <= ccg_ve_bx_select;
ccg_ve_by_select_r1            <= ccg_ve_by_select;
ccg_ve_bz_select_r1            <= ccg_ve_bz_select;
ccg_ve_bw_select_r1            <= ccg_ve_bw_select;
```

```
ccg_ve_a_is_wwww_r1            <= ccg_ve_a_is_wwww;
ccg_ve_broadcast_x_r1          <= ccg_ve_broadcast_x;
ccg_ve_abs_a_r1                <= ccg_ve_abs_a;
ccg_ve_abs_b_r1                <= ccg_ve_abs_b;
ccg_ve_abs_c_r1                <= ccg_ve_abs_c;
ccg_ve_ax_negate_r1            <= ccg_ve_ax_negate;
ccg_ve_ay_negate_r1            <= ccg_ve_ay_negate;
ccg_ve_az_negate_r1            <= ccg_ve_az_negate;
ccg_ve_aw_negate_r1            <= ccg_ve_aw_negate;
ccg_ve_bx_negate_r1            <= ccg_ve_bx_negate;
ccg_ve_by_negate_r1            <= ccg_ve_by_negate;
ccg_ve_bz_negate_r1            <= ccg_ve_bz_negate;
ccg_ve_bw_negate_r1            <= ccg_ve_bw_negate;
ccg_ve_cx_negate_r1            <= ccg_ve_cx_negate;
ccg_ve_cy_negate_r1            <= ccg_ve_cy_negate;
ccg_ve_cz_negate_r1            <= ccg_ve_cz_negate;
ccg_ve_cw_negate_r1            <= ccg_ve_cw_negate;
ccg_ve_bcc_flat_tst_r1         <= ccg_ve_bcc_flat_tst;
ccg_ve_out_mem_sel_r1          <= ccg_ve_out_mem_sel;
ccg_ve_out_addr_r1             <= ccg_ve_out_addr;
ccg_ve_out_we_r1               <= ccg_ve_out_we;
ccg_ve_accum_sel_r1            <= ccg_ve_accum_sel;
ccg_ve_pre_acc_we_r1           <= ccg_ve_pre_acc_we;
ccg_agve_dly_vertex_store_indx_r1 <= ccg_agve_dly_vertex_store_indx;
ccg_agve_dly_valid_bit_set_r1     <= ccg_agve_dly_valid_bit_set;
```

```
ccg_agve_dly_user_clip_indx_r1    <= ccg_agve_dly_user_clip_indx;
ccg_agve_dly_vv_cc_test_r1        <= ccg_agve_dly_vv_cc_test;
ccg_agve_dly_ucp_cc_test_r1       <= ccg_agve_dly_ucp_cc_test;
ccg_agve_dly_bcc_cc_test_r1       <= ccg_agve_dly_bcc_cc_test;
ccg_agve_dly_ps_ucp_cc_test_r1    <= ccg_agve_dly_ps_ucp_cc_test;
ccg_agve_dly_ps_engh_test_r1      <= ccg_agve_dly_ps_engh_test;
//CLIPPER Decode R1 Delay
clip_vte_opcode_r1             <= clip_vte_opcode;
clip_vte_st_indx_r1            <= clip_vte_st_indx;
clip_ve_opcode_r1              <= clip_ve_opcode;
clip_ve_a_memsel_r1            <= clip_ve_a_memsel;
clip_ve_b_memsel_r1            <= clip_ve_b_memsel;
clip_ve_ax_select_r1           <= clip_ve_ax_select;
clip_ve_ay_select_r1           <= clip_ve_ay_select;
clip_ve_az_select_r1           <= clip_ve_az_select;
clip_ve_aw_select_r1           <= clip_ve_aw_select;
clip_ve_bx_select_r1           <= clip_ve_bx_select;
clip_ve_by_select_r1           <= clip_ve_by_select;
clip_ve_bz_select_r1           <= clip_ve_bz_select;
clip_ve_bw_select_r1           <= clip_ve_bw_select;
clip_ve_a_is_wwww_r1           <= clip_ve_a_is_wwww;
clip_ve_broadcast_x_r1         <= clip_ve_broadcast_x;
clip_ve_abs_a_r1               <= clip_ve_abs_a;
clip_ve_abs_b_r1               <= clip_ve_abs_b;
clip_ve_abs_c_r1               <= clip_ve_abs_c;
```

```
clip_ve_ax_negate_r1             <= clip_ve_ax_negate;
clip_ve_ay_negate_r1             <= clip_ve_ay_negate;
clip_ve_az_negate_r1             <= clip_ve_az_negate;
clip_ve_aw_negate_r1             <= clip_ve_aw_negate;
clip_ve_bx_negate_r1             <= clip_ve_bx_negate;
clip_ve_by_negate_r1             <= clip_ve_by_negate;
clip_ve_bz_negate_r1             <= clip_ve_bz_negate;
clip_ve_bw_negate_r1             <= clip_ve_bw_negate;
clip_ve_cx_negate_r1             <= clip_ve_cx_negate;
clip_ve_cy_negate_r1             <= clip_ve_cy_negate;
clip_ve_cz_negate_r1             <= clip_ve_cz_negate;
clip_ve_cw_negate_r1             <= clip_ve_cw_negate;
clip_ve_bcc_flat_tst_r1          <= clip_ve_bcc_flat_tst;
clip_ve_out_mem_sel_r1           <= clip_ve_out_mem_sel;
clip_ve_out_addr_r1              <= clip_ve_out_addr;
clip_ve_out_we_r1                <= clip_ve_out_we;
clip_ve_accum_sel_r1             <= clip_ve_accum_sel;
clip_ve_pre_acc_we_r1            <= clip_ve_pre_acc_we;
clip_agve_dly_vertex_store_indx_r1 <= clip_agve_dly_vertex_store_indx;
clip_agve_dly_valid_bit_set_r1    <= clip_agve_dly_valid_bit_set;
clip_agve_dly_user_clip_indx_r1   <= clip_agve_dly_user_clip_indx;
clip_agve_dly_vv_cc_test_r1       <= clip_agve_dly_vv_cc_test;
clip_agve_dly_ucp_cc_test_r1      <= clip_agve_dly_ucp_cc_test;
clip_agve_dly_bcc_cc_test_r1      <= clip_agve_dly_bcc_cc_test;
clip_agve_dly_ps_ucp_cc_test_r1   <= clip_agve_dly_ps_ucp_cc_test;
```

```
clip_agve_dly_ps_engh_test_r1     <= clip_agve_dly_ps_engh_test;

pos_re_r1                        <= pos_re;
pntsz_re_r1                      <= pntsz_re;
ve_veoc_vector_back_re_r1        <= ve_veoc_vector_back_re;
stve_re_r1                       <= stve_re_r0;

//AG_R2 register
vte_opcode_r2               <= arbsel_vte_opcode;
vte_st_indx_r2              <= arbsel_vte_st_indx;
ve_opcode_r2                <= arbsel_ve_opcode;
ve_a_memsel_r2              <= arbsel_ve_a_memsel;
ve_b_memsel_r2              <= arbsel_ve_b_memsel;
ve_ax_select_r2             <= arbsel_ve_ax_select;
ve_ay_select_r2             <= arbsel_ve_ay_select;
ve_az_select_r2             <= arbsel_ve_az_select;
ve_aw_select_r2             <= arbsel_ve_aw_select;
ve_bx_select_r2             <= arbsel_ve_bx_select;
ve_by_select_r2             <= arbsel_ve_by_select;
ve_bz_select_r2             <= arbsel_ve_bz_select;
ve_bw_select_r2             <= arbsel_ve_bw_select;
ve_a_is_wwww_r2             <= arbsel_ve_a_is_wwww;
ve_broadcast_x_r2           <= arbsel_ve_broadcast_x;
ve_abs_a_r2                 <= arbsel_ve_abs_a;
ve_abs_b_r2                 <= arbsel_ve_abs_b;
```

```
ve_abs_c_r2                 <= arbsel_ve_abs_c;
ve_ax_negate_r2             <= arbsel_ve_ax_negate;
ve_ay_negate_r2             <= arbsel_ve_ay_negate;
ve_az_negate_r2             <= arbsel_ve_az_negate;
ve_aw_negate_r2             <= arbsel_ve_aw_negate;
ve_bx_negate_r2             <= arbsel_ve_bx_negate;
ve_by_negate_r2             <= arbsel_ve_by_negate;
ve_bz_negate_r2             <= arbsel_ve_bz_negate;
ve_bw_negate_r2             <= arbsel_ve_bw_negate;
ve_cx_negate_r2             <= arbsel_ve_cx_negate;
ve_cy_negate_r2             <= arbsel_ve_cy_negate;
ve_cz_negate_r2             <= arbsel_ve_cz_negate;
ve_cw_negate_r2             <= arbsel_ve_cw_negate;
ve_bcc_flat_tst_r2          <= arbsel_ve_bcc_flat_tst;
ve_out_mem_sel_r2           <= arbsel_ve_out_mem_sel;
ve_out_addr_r2              <= arbsel_ve_out_addr;
ve_out_we_r2                <= arbsel_ve_out_we;
ve_accum_sel_r2             <= arbsel_ve_accum_sel;
ve_pre_acc_we_r2            <= arbsel_ve_pre_acc_we;
agve_dly_vertex_store_indx_r2 <= arbsel_agve_dly_vertex_store_indx;
agve_dly_valid_bit_set_r2    <= arbsel_agve_dly_valid_bit_set;
agve_dly_user_clip_indx_r2   <= arbsel_agve_dly_user_clip_indx;
agve_dly_vv_cc_test_r2       <= arbsel_agve_dly_vv_cc_test;
agve_dly_ucp_cc_test_r2      <= arbsel_agve_dly_ucp_cc_test;
agve_dly_bcc_cc_test_r2      <= arbsel_agve_dly_bcc_cc_test;
```

```
agve_dly_ps_ucp_cc_test_r2    <= arbsel_agve_dly_ps_ucp_cc_test;
agve_dly_ps_engh_test_r2      <= arbsel_agve_dly_ps_engh_test;

if (pos_re_r1) begin
   pos_rdata_r2 <= pos_rdata;
end
if(pntsz_re_r1) begin
   pntsz_rdata_r2 <= pntsz_rdata;
end
if(ve_veoc_vector_back_re_r1) begin
   ve_veoc_vector_back_rdata_r2 <= ve_veoc_vector_back_rdata;
end
if(stve_re_r1) begin
   stve_rdata_r2 <= stve_rdata;
end


//AG_R3 outup register
ag_vte_opcode             <= vte_opcode_r2;
ag_vte_st_indx            <= vte_st_indx_r2;
ag_vte_vertex_store_indx   <= agve_dly_vertex_store_indx_r2[1:0];

ag_ve_opcode              <= ve_opcode_r2;

ag_ve_in_a0               <= agswz_ve_in_a0;
```

```
    ag_ve_in_a1              <= agswz_ve_in_a1;
    ag_ve_in_a2              <= agswz_ve_in_a2;
    ag_ve_in_a3              <= agswz_ve_in_a3;
    ag_ve_in_b0              <= agswz_ve_in_b0;
    ag_ve_in_b1              <= agswz_ve_in_b1;
    ag_ve_in_b2              <= agswz_ve_in_b2;
    ag_ve_in_b3              <= agswz_ve_in_b3;

    ag_ve_a_is_wwww          <= ve_a_is_wwww_r2;
    ag_ve_broadcast_x        <= ve_broadcast_x_r2;
    ag_ve_abs_a              <= ve_abs_a_r2;
    ag_ve_abs_b              <= ve_abs_b_r2;
    ag_ve_abs_c              <= ve_abs_c_r2;
    ag_ve_ax_negate          <= ve_ax_negate_r2;
    ag_ve_ay_negate          <= ve_ay_negate_r2;
    ag_ve_az_negate          <= ve_az_negate_r2;
    ag_ve_aw_negate          <= ve_aw_negate_r2;
    ag_ve_bx_negate          <= ve_bx_negate_r2;
    ag_ve_by_negate          <= ve_by_negate_r2;
    ag_ve_bz_negate          <= ve_bz_negate_r2;
    ag_ve_bw_negate          <= ve_bw_negate_r2;
    ag_ve_cx_negate          <= ve_cx_negate_r2;
    ag_ve_cy_negate          <= ve_cy_negate_r2;
    ag_ve_cz_negate          <= ve_cz_negate_r2;
    ag_ve_cw_negate          <= ve_cw_negate_r2;
```

```
    ag_ve_bcc_flat_tst       <= ve_bcc_flat_tst_r2;
    ag_ve_out_mem_sel        <= ve_out_mem_sel_r2;
    ag_ve_out_addr           <= ve_out_addr_r2;
    ag_ve_out_we             <= ve_out_we_r2;
    ag_ve_accum_sel          <= ve_accum_sel_r2;
    ag_ve_pre_accum_we       <= ve_pre_acc_we_r2;

    agve_dly_valid_op_r3         <= agve_valid_op;
    agve_dly_vertex_store_indx_r3 <= agve_dly_vertex_store_indx_r2;
    agve_dly_valid_bit_set_r3    <= agve_dly_valid_bit_set_r2;
    agve_dly_user_clip_indx_r3   <= agve_dly_user_clip_indx_r2;
    agve_dly_vv_cc_test_r3       <= agve_dly_vv_cc_test_r2;
    agve_dly_ucp_cc_test_r3      <= agve_dly_ucp_cc_test_r2;
    agve_dly_bcc_cc_test_r3      <= agve_dly_bcc_cc_test_r2;
    agve_dly_ps_ucp_cc_test_r3   <= agve_dly_ps_ucp_cc_test_r2;
    agve_dly_ps_engh_test_r3     <= agve_dly_ps_engh_test_r2;


    agve_dly0 <= { agve_dly_valid_op_r3,
                   agve_dly_vertex_store_indx_r3,
                   agve_dly_valid_bit_set_r3,
                   agve_dly_user_clip_indx_r3,
                   agve_dly_vv_cc_test_r3,
                   agve_dly_ucp_cc_test_r3,
                   agve_dly_bcc_cc_test_r3,
```

```
                   agve_dly_ps_ucp_cc_test_r3,
                   agve_dly_ps_engh_test_r3 };

    agve_dly1 <= agve_dly0;
    agve_dly2 <= agve_dly1;
    agve_dly3 <= agve_dly2;
    agve_dly4 <= agve_dly3;
    agve_dly5 <= agve_dly4;

//add latencey matching stages here

    { agve_dly_valid_op,
      agve_dly_vertex_store_indx,
      agve_dly_valid_bit_set,
      agve_dly_user_clip_indx,
      agve_dly_vv_cc_test,
      agve_dly_ucp_cc_test,
      agve_dly_bcc_cc_test,
      agve_dly_ps_ucp_cc_test,
      agve_dly_ps_engh_test } <= agve_dly5;
 end

assign ag_to_clip_point_size = pntsz_rdata_r2;


```

```

//*********************************************************************
// Functions
//*********************************************************************

// Function implementation of arb
function [1:0] arb;
   input srst;
   input arb_clip_ve_valid;
   input arb_ccg_ve_valid;
   reg arb_nxt_clip_xfc;
   reg arb_nxt_ccg_xfc;
  begin
    if ((arb_clip_ve_valid == 1'b1) && (srst == 1'b0)) begin
       arb_nxt_ccg_xfc   = 1'b0;
       arb_nxt_clip_xfc  = 1'b1;
    end else if ((arb_ccg_ve_valid == 1'b1) && (srst == 1'b0)) begin
       arb_nxt_ccg_xfc   = 1'b1;
       arb_nxt_clip_xfc  = 1'b0;
    end else begin
       arb_nxt_ccg_xfc   = 1'b0;
       arb_nxt_clip_xfc  = 1'b0;
    end
    arb = {arb_nxt_clip_xfc, arb_nxt_ccg_xfc};
  end
```

```
endfunction

// Function implementation of dirty bit
function [0:0] dirty;
   input srst;
   input rbiu_cpy;
   input [3:0] sel;
   reg nxt_write_after_cpy;
  begin
    if (srst || rbiu_cpy) begin
       nxt_write_after_cpy  = 1'b0;
    end else if (|sel) begin
       nxt_write_after_cpy  = 1'b1;
    end
    dirty = nxt_write_after_cpy;
  end
endfunction

// Function implementation of state write ptr stwtptr
function [2:0] stwtptr;
   input srst;
   input rbiu_cpy;
   input write_after_cpy;
   input [2:0] cur_write_ptr;
   reg   [2:0] nxt_write_ptr;
```

```
  begin
    if (srst) begin
       nxt_write_ptr  = 3'b000;
    end else if (write_after_cpy && rbiu_cpy) begin
       nxt_write_ptr  = cur_write_ptr + 3'b001;
    end else begin
       nxt_write_ptr  = cur_write_ptr;
    end
    stwtptr = nxt_write_ptr;
  end
endfunction

function [31:0] swizzle;
   input [2:0] ve_swizzle_select;
   input [127:0] mem_sel_data;
   reg [31:0] swizzled_data;
  begin
    swizzled_data= 32'h0;

    case( ve_swizzle_select)
      SRC_SELECT_X: begin
         swizzled_data = mem_sel_data[31:0];
      end
      SRC_SELECT_Y: begin
         swizzled_data = mem_sel_data[63:32];
```

```
      end
      SRC_SELECT_Z: begin
         swizzled_data = mem_sel_data[95:64];
      end
      SRC_SELECT_W: begin
         swizzled_data = mem_sel_data[127:96];
      end
      SRC_SELECT_FORCE_0: begin
         swizzled_data = 32'h0;
      end
      SRC_SELECT_FORCE_1: begin
         swizzled_data = 32'h3f800000;
      end
      default;
    endcase
    swizzle = swizzled_data;
  end
endfunction

//*****************************************************************************
***
// State Storage Instantiation
//*****************************************************************************
***
ati_1rp_state_storage #(1)      ustate_storage_reg_dx_clip (
            .w_data(rbiu_wd[19]),
```

```
            .r_data(agrd_dx_clip_space_def),
            .st_data(dx_clip_space_def),
            .st_indx(arb_state_var_indx_r0),
            .w_addr(rbiu_wa),  //context
        .r_addr(rbiu_wd[2:0]),
            .we(rbiu_we),
            .re(rbiu_re),
            .sel(rbiu_ag_dx_clip_sp_def_sel),
            .cpy(rbiu_cpy),
            .clk(sclk));

ati_1rp_state_storage #(3) ustate_storage_reg_ucp0 (
            .w_data(ucp0_write_ptr),
            .r_data(agrd_ucp0_write_ptr),
            .st_data(ucp0_rd_off),
            .st_indx(arb_state_var_indx_r0),
            .w_addr(rbiu_wa),  //context
        .r_addr(rbiu_wd[2:0]),
            .we(rbiu_we),
            .re(rbiu_re),
            .sel(|rbiu_ag_ucp0_sel),
            .cpy(rbiu_cpy),
            .clk(sclk));

ati_1rp_state_storage #(3) ustate_storage_reg_ucp1 (
           .w_data(ucp1_write_ptr),
           .r_data(agrd_ucp1_write_ptr),
           .st_data(ucp1_rd_off),
           .st_indx(arb_state_var_indx_r0),
           .w_addr(rbiu_wa),  //context
       .r_addr(rbiu_wd[2:0]),
           .we(rbiu_we),
           .re(rbiu_re),
           .sel(|rbiu_ag_ucp1_sel),
           .cpy(rbiu_cpy),
           .clk(sclk));

ati_1rp_state_storage #(3) ustate_storage_reg_ucp2 (
           .w_data(ucp2_write_ptr),
           .r_data(agrd_ucp2_write_ptr),
           .st_data(ucp2_rd_off),
           .st_indx(arb_state_var_indx_r0),
           .w_addr(rbiu_wa),  //context
       .r_addr(rbiu_wd[2:0]),
           .we(rbiu_we),
           .re(rbiu_re),
           .sel(|rbiu_ag_ucp2_sel),
           .cpy(rbiu_cpy),
           .clk(sclk));

```

```
ati_1rp_state_storage #(3) ustate_storage_reg_ucp3 (
           .w_data(ucp3_write_ptr),
           .r_data(agrd_ucp3_write_ptr),
           .st_data(ucp3_rd_off),
           .st_indx(arb_state_var_indx_r0),
           .w_addr(rbiu_wa),  //context
       .r_addr(rbiu_wd[2:0]),
           .we(rbiu_we),
           .re(rbiu_re),
           .sel(|rbiu_ag_ucp3_sel),
           .cpy(rbiu_cpy),
           .clk(sclk));

ati_1rp_state_storage #(3) ustate_storage_reg_ucp4 (
           .w_data(ucp4_write_ptr),
           .r_data(agrd_ucp4_write_ptr),
           .st_data(ucp4_rd_off),
           .st_indx(arb_state_var_indx_r0),
           .w_addr(rbiu_wa),  //context
       .r_addr(rbiu_wd[2:0]),
           .we(rbiu_we),
           .re(rbiu_re),
           .sel(|rbiu_ag_ucp4_sel),
           .cpy(rbiu_cpy),
           .clk(sclk));
```

```

ati_1rp_state_storage #(3) ustate_storage_reg_ucp5 (
           .w_data(ucp5_write_ptr),
           .r_data(agrd_ucp5_write_ptr),
           .st_data(ucp5_rd_off),
           .st_indx(arb_state_var_indx_r0),
           .w_addr(rbiu_wa),  //context
       .r_addr(rbiu_wd[2:0]),
           .we(rbiu_we),
           .re(rbiu_re),
           .sel(|rbiu_ag_ucp5_sel),
           .cpy(rbiu_cpy),
           .clk(sclk));

ati_1rp_state_storage #(3) ustate_storage_reg_gb (
           .w_data(gb_write_ptr),
           .r_data(agrd_gb_write_ptr),
           .st_data(gb_rd_off),
           .st_indx(arb_state_var_indx_r0),
           .w_addr(rbiu_wa),  //context
       .r_addr(rbiu_wd[2:0]),
           .we(rbiu_we),
           .re(rbiu_re),
           .sel(|rbiu_ag_gb_sel),
           .cpy(rbiu_cpy),
```

```
           .clk(sclk));

ati_1rp_state_storage #(3) ustate_storage_reg_pntsz (
           .w_data(pntsz_write_ptr),
           .r_data(agrd_pntsz_write_ptr),
           .st_data(pntsz_rd_off),
           .st_indx(arb_state_var_indx_r0),
           .w_addr(rbiu_wa),  //context
       .r_addr(rbiu_wd[2:0]),
           .we(rbiu_we),
           .re(rbiu_re),
           .sel(|rbiu_ag_pntsz_sel),
           .cpy(rbiu_cpy),
           .clk(sclk));



//************************************************************************
***
// Memory Instantiation
//************************************************************************
***
`ifdef USE_BEHAVE_MEM

// Instantiate the cliptemp memory as
// 4 35d by 32w to create 35d x 128w
```

```
dum_mem_p2 #(
u0_clptmp_ADDR_WIDTH ,
u0_clptmp_DATA_WIDTH ,
u0_clptmp_WORDS      ,
u0_clptmp_DEBUG
)
u0_clptmp_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(ve_veoc_vector_back_re),
        .iMEW(ve_veoc_vector_back_we[0]),
        .iWEN(ve_veoc_vector_back_we[0]),
        .iRADR(ve_veoc_vector_back_raddr),
        .iWADR(ve_waddr),
        .iD(ve_wdata[31:0]),
        .oQ(ve_veoc_vector_back_rdata[31:0]));

dum_mem_p2 #(
u1_clptmp_ADDR_WIDTH ,
u1_clptmp_DATA_WIDTH ,
u1_clptmp_WORDS      ,
u1_clptmp_DEBUG
)
u1_clptmp_dum_mem_p2 (
        .iRCLK(sclk),
```

```
        .iWCLK(sclk),
        .iMER(ve_veoc_vector_back_re),
        .iMEW(ve_veoc_vector_back_we[1]),
        .iWEN(ve_veoc_vector_back_we[1]),
        .iRADR(ve_veoc_vector_back_raddr),
        .iWADR(ve_waddr),
        .iD(ve_wdata[63:32]),
        .oQ(ve_veoc_vector_back_rdata[63:32]));

dum_mem_p2 #(
u2_clptmp_ADDR_WIDTH ,
u2_clptmp_DATA_WIDTH ,
u2_clptmp_WORDS      ,
u2_clptmp_DEBUG
)
u2_clptmp_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(ve_veoc_vector_back_re),
        .iMEW(ve_veoc_vector_back_we[2]),
        .iWEN(ve_veoc_vector_back_we[2]),
        .iRADR(ve_veoc_vector_back_raddr),
        .iWADR(ve_waddr),
        .iD(ve_wdata[95:64]),
        .oQ(ve_veoc_vector_back_rdata[95:64]));
```

```

dum_mem_p2 #(
u3_clptmp_ADDR_WIDTH ,
u3_clptmp_DATA_WIDTH ,
u3_clptmp_WORDS      ,
u3_clptmp_DEBUG
)
u3_clptmp_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(ve_veoc_vector_back_re),
        .iMEW(ve_veoc_vector_back_we[3]),
        .iWEN(ve_veoc_vector_back_we[3]),
        .iRADR(ve_veoc_vector_back_raddr),
        .iWADR(ve_waddr),
        .iD(ve_wdata[127:96]),
        .oQ(ve_veoc_vector_back_rdata[127:96]));

// Instantiate the pntsz memory
// 1 64d x 32w
dum_mem_p2 #(
u_pntsz_ADDR_WIDTH ,
u_pntsz_DATA_WIDTH ,
u_pntsz_WORDS      ,
u_pntsz_DEBUG
```

```
)
u_pntsz_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(pntsz_re),
        .iMEW(pntsz_mem_we),
        .iWEN(pntsz_mem_we),
        .iRADR(pntsz_raddr),
        .iWADR(pntsz_mem_waddr),
        .iD(pos_pntsz_ag_mem_data[31:0]),
        .oQ(pntsz_rdata));

// Instantiate the position memory
// 1 64d x 128w
dum_mem_p2 #(
u_pos_ADDR_WIDTH ,
u_pos_DATA_WIDTH ,
u_pos_WORDS      ,
u_pos_DEBUG
)
u_pos_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(pos_re),
        .iMEW(pos_mem_we),
        .iWEN(pos_mem_we),
        .iRADR(pos_raddr),
        .iWADR(pos_mem_waddr),
        .iD(pos_pntsz_ag_mem_data),
        .oQ(pos_rdata));

// Instantiate the state for ucp,gb,pntsz memory as
// 4 64d by 32w to create 64d x 128w
dum_mem_p2 #(
u0_stve_ADDR_WIDTH ,
u0_stve_DATA_WIDTH ,
u0_stve_WORDS     ,
u0_stve_DEBUG
)
u0_stve_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(stve_re_r0),
        .iMEW(stve_we[0]),
        .iWEN(stve_we[0]),
        .iRADR(stve_raddr),
        .iWADR(stve_wa),
        .iD(rbiu_wd),
        .oQ(stve_rdata[31:0]));

```

```
dum_mem_p2 #(
u1_stve_ADDR_WIDTH ,
u1_stve_DATA_WIDTH ,
u1_stve_WORDS     ,
u1_stve_DEBUG
)
u1_stve_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(stve_re_r0),
        .iMEW(stve_we[1]),
        .iWEN(stve_we[1]),
        .iRADR(stve_raddr),
        .iWADR(stve_wa),
        .iD(rbiu_wd),
        .oQ(stve_rdata[63:32]));

dum_mem_p2 #(
u2_stve_ADDR_WIDTH ,
u2_stve_DATA_WIDTH ,
u2_stve_WORDS     ,
u2_stve_DEBUG
)
u2_stve_dum_mem_p2 (
        .iRCLK(sclk),
```

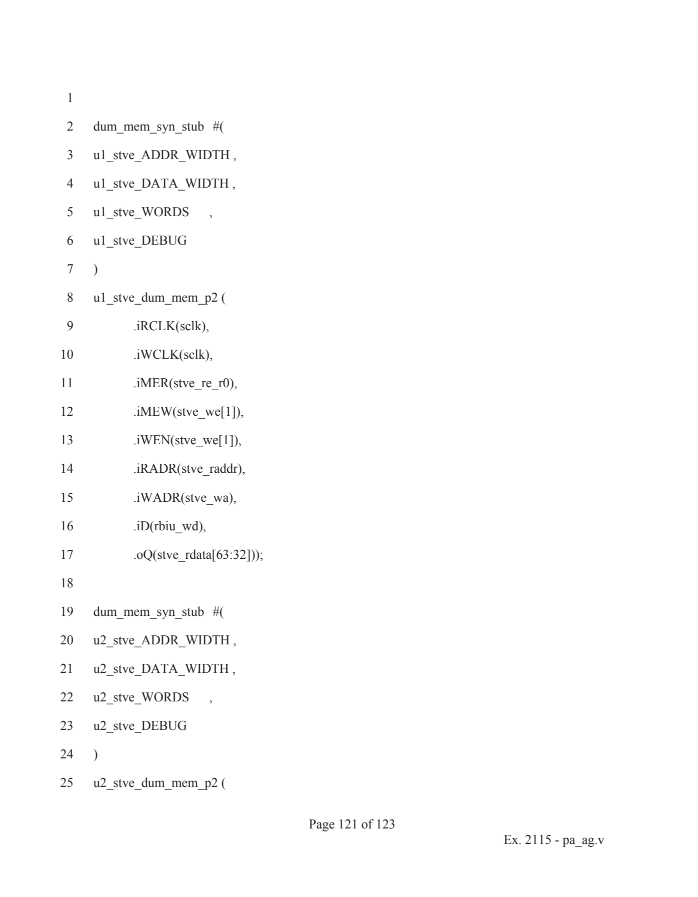
```
        .iWCLK(sclk),
        .iMER(stve_re_r0),
        .iMEW(stve_we[2]),
        .iWEN(stve_we[2]),
        .iRADR(stve_raddr),
        .iWADR(stve_wa),
        .iD(rbiu_wd),
        .oQ(stve_rdata[95:64]));

dum_mem_p2 #(
u3_stve_ADDR_WIDTH ,
u3_stve_DATA_WIDTH ,
u3_stve_WORDS     ,
u3_stve_DEBUG
)
u3_stve_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(stve_re_r0),
        .iMEW(stve_we[3]),
        .iWEN(stve_we[3]),
        .iRADR(stve_raddr),
        .iWADR(stve_wa),
        .iD(rbiu_wd),
        .oQ(stve_rdata[127:96]));
```

```
`else
dum_mem_syn_stub  #(
u0_clptmp_ADDR_WIDTH ,
u0_clptmp_DATA_WIDTH ,
u0_clptmp_WORDS     ,
u0_clptmp_DEBUG
)
u0_clptmp_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(ve_veoc_vector_back_re),
        .iMEW(ve_veoc_vector_back_we[0]),
        .iWEN(ve_veoc_vector_back_we[0]),
        .iRADR(ve_veoc_vector_back_raddr),
        .iWADR(ve_waddr),
        .iD(ve_wdata[31:0]),
        .oQ(ve_veoc_vector_back_rdata[31:0]));

dum_mem_syn_stub  #(
u1_clptmp_ADDR_WIDTH ,
u1_clptmp_DATA_WIDTH ,
u1_clptmp_WORDS     ,
u1_clptmp_DEBUG
)
u1_clptmp_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(ve_veoc_vector_back_re),
        .iMEW(ve_veoc_vector_back_we[1]),
        .iWEN(ve_veoc_vector_back_we[1]),
        .iRADR(ve_veoc_vector_back_raddr),
        .iWADR(ve_waddr),
        .iD(ve_wdata[63:32]),
        .oQ(ve_veoc_vector_back_rdata[63:32]));

dum_mem_syn_stub  #(
u2_clptmp_ADDR_WIDTH ,
u2_clptmp_DATA_WIDTH ,
u2_clptmp_WORDS      ,
u2_clptmp_DEBUG
)
u2_clptmp_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(ve_veoc_vector_back_re),
        .iMEW(ve_veoc_vector_back_we[2]),
        .iWEN(ve_veoc_vector_back_we[2]),
        .iRADR(ve_veoc_vector_back_raddr),
        .iWADR(ve_waddr),
        .iD(ve_wdata[95:64]),
```

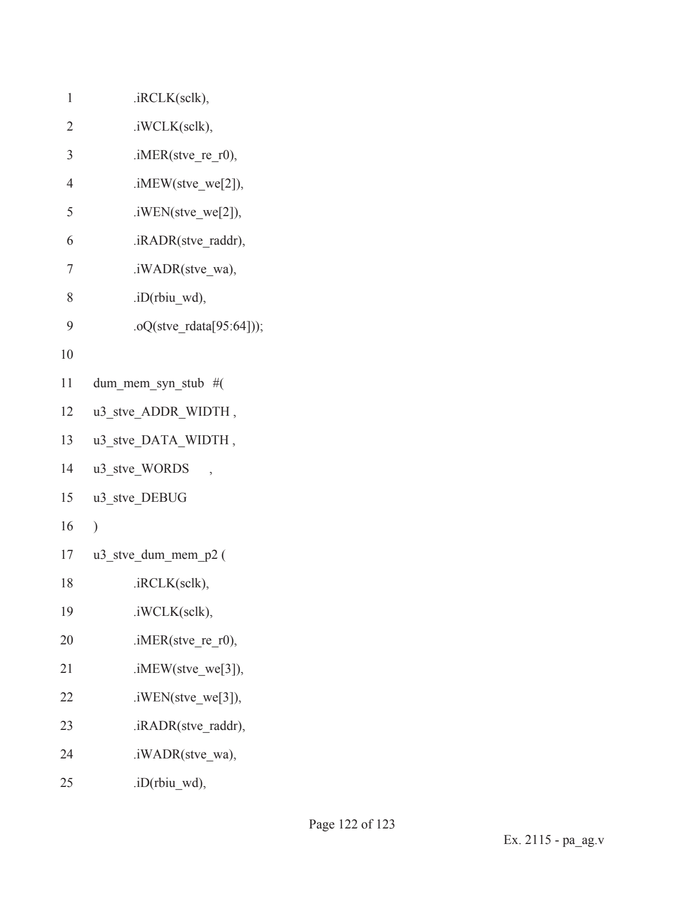
```
        .oQ(ve_veoc_vector_back_rdata[95:64]));

dum_mem_syn_stub  #(
u3_clptmp_ADDR_WIDTH ,
u3_clptmp_DATA_WIDTH ,
u3_clptmp_WORDS      ,
u3_clptmp_DEBUG
)
u3_clptmp_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(ve_veoc_vector_back_re),
        .iMEW(ve_veoc_vector_back_we[3]),
        .iWEN(ve_veoc_vector_back_we[3]),
        .iRADR(ve_veoc_vector_back_raddr),
        .iWADR(ve_waddr),
        .iD(ve_wdata[127:96]),
        .oQ(ve_veoc_vector_back_rdata[127:96]));

// Instantiate the pntsz memory
// 1 64d x 32w
dum_mem_syn_stub  #(
u_pntsz_ADDR_WIDTH ,
u_pntsz_DATA_WIDTH ,
u_pntsz_WORDS      ,
```

```
u_pntsz_DEBUG
)
u_pntsz_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(pntsz_re),
        .iMEW(pntsz_mem_we),
        .iWEN(pntsz_mem_we),
        .iRADR(pntsz_raddr),
        .iWADR(pntsz_mem_waddr),
        .iD(pos_pntsz_ag_mem_data[31:0]),
        .oQ(pntsz_rdata));

// Instantiate the position memory
// 1 64d x 128w
dum_mem_syn_stub  #(
u_pos_ADDR_WIDTH ,
u_pos_DATA_WIDTH ,
u_pos_WORDS      ,
u_pos_DEBUG
)
u_pos_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(pos_re),
```

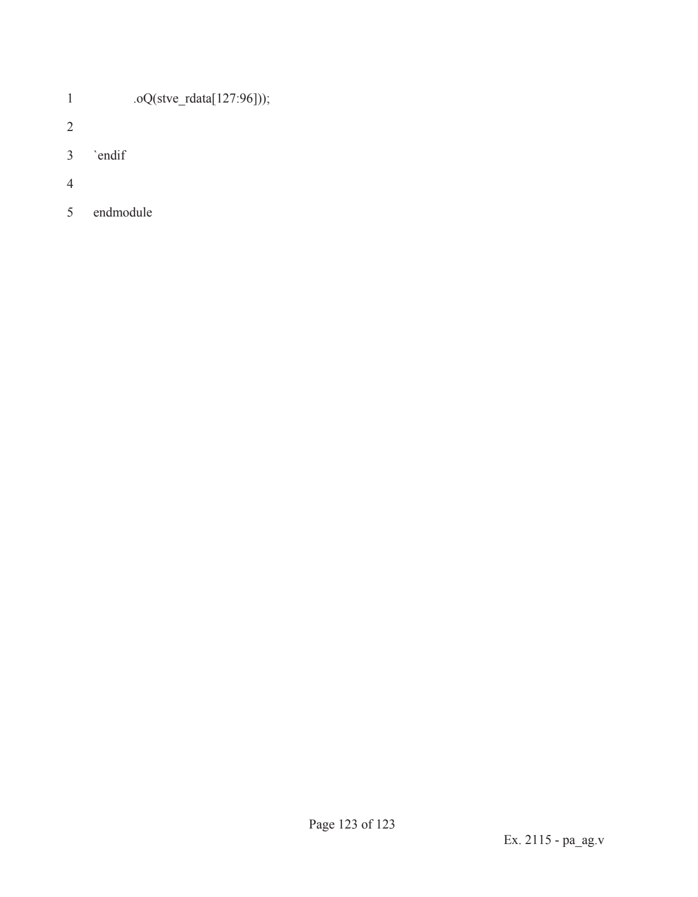
```
        .iMEW(pos_mem_we),
        .iWEN(pos_mem_we),
        .iRADR(pos_raddr),
        .iWADR(pos_mem_waddr),
        .iD(pos_pntsz_ag_mem_data),
        .oQ(pos_rdata));

// Instantiate the state for ucp,gb,pntsz memory as
// 4 64d by 32w to create 64d x 128w
dum_mem_syn_stub  #(
u0_stve_ADDR_WIDTH ,
u0_stve_DATA_WIDTH ,
u0_stve_WORDS      ,
u0_stve_DEBUG
)
u0_stve_dum_mem_p2 (
        .iRCLK(sclk),
        .iWCLK(sclk),
        .iMER(stve_re_r0),
        .iMEW(stve_we[0]),
        .iWEN(stve_we[0]),
        .iRADR(stve_raddr),
        .iWADR(stve_wa),
        .iD(rbiu_wd),
        .oQ(stve_rdata[31:0]));
```



```

dum_mem_syn_stub  #(
u1_stve_ADDR_WIDTH ,
u1_stve_DATA_WIDTH ,
u1_stve_WORDS      ,
u1_stve_DEBUG
)
u1_stve_dum_mem_p2 (
          .iRCLK(sclk),
          .iWCLK(sclk),
          .iMER(stve_re_r0),
          .iMEW(stve_we[1]),
          .iWEN(stve_we[1]),
          .iRADR(stve_raddr),
          .iWADR(stve_wa),
          .iD(rbiu_wd),
          .oQ(stve_rdata[63:32]));

dum_mem_syn_stub  #(
u2_stve_ADDR_WIDTH ,
u2_stve_DATA_WIDTH ,
u2_stve_WORDS      ,
u2_stve_DEBUG
)
u2_stve_dum_mem_p2 (
```



```
          .iRCLK(sclk),
          .iWCLK(sclk),
          .iMER(stve_re_r0),
          .iMEW(stve_we[2]),
          .iWEN(stve_we[2]),
          .iRADR(stve_raddr),
          .iWADR(stve_wa),
          .iD(rbiu_wd),
          .oQ(stve_rdata[95:64]));

dum_mem_syn_stub  #(
u3_stve_ADDR_WIDTH ,
u3_stve_DATA_WIDTH ,
u3_stve_WORDS      ,
u3_stve_DEBUG
)
u3_stve_dum_mem_p2 (
          .iRCLK(sclk),
          .iWCLK(sclk),
          .iMER(stve_re_r0),
          .iMEW(stve_we[3]),
          .iWEN(stve_we[3]),
          .iRADR(stve_raddr),
          .iWADR(stve_wa),
          .iD(rbiu_wd),
```



```
          .oQ(stve_rdata[127:96]));

`endif

endmodule
```

```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//   Project: R400
//     File: pa_sxifccg.v
//
//   Description:
//     sxif/ccg top level
//
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//
//  Trade secret of ATI Technologies, Inc.
//  Copyright 2002, ATI Technologies, Inc., (unpublished)
//
//  All rights reserved.  This notice is intended as a precaution against
//  inadvertent publication and does not imply publication or any waiver
//  of confidentiality.  The year included in the foregoing notice is the
//  year of creation of the work.
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

`include "header.v"

module
pa_sxifccg
 (
```

```
////////////////////////////////////////////////////////////////
// INPUTS
////////////////////////////////////////////////////////////////
// global
clk,
reset,

// ati_state_storage (sxif)
isxif_st_w_data,
isxif_st_w_addr,
isxif_st_we,
isxif_st_r_addr,
isxif_st_re,
isxif_st_sel,
isxif_st_cpy,

// vgt_to_ccgen fifo
ivgt_to_ccgen_fifo_write,
ivgt_to_ccgen_fifo_active_verts,
ivgt_to_ccgen_fifo_state_var_indx,

// sx0, receive
isx0_receive_fifo_write,
isx0_receive_fifo_wrdata,

```

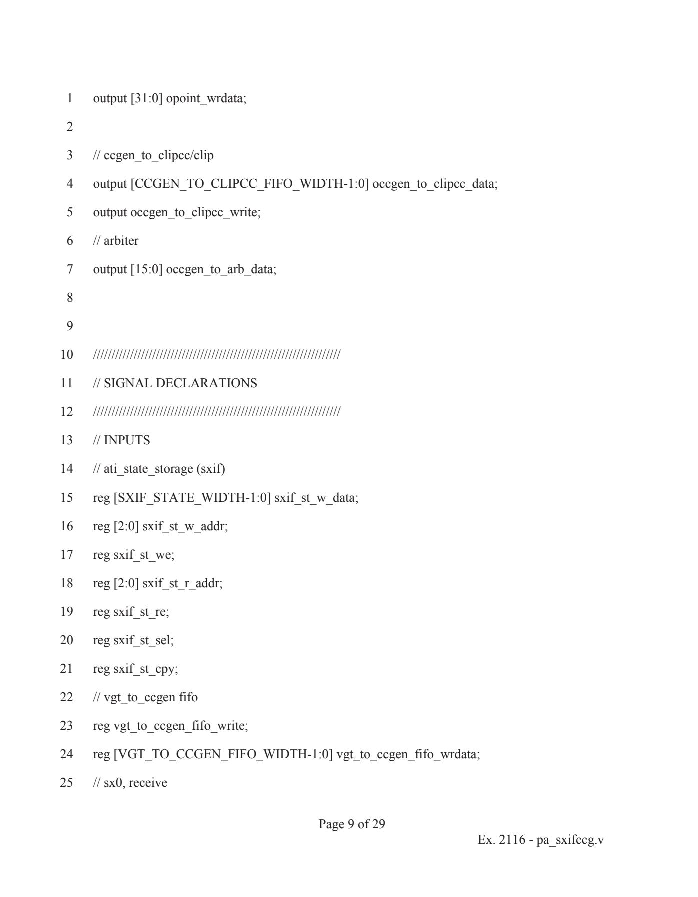
```
 // sx1, receive
 isx1_receive_fifo_write,
 isx1_receive_fifo_wrdata,

 // ccg state
 iccg_state0,
 iccg_state1,
 iccg_state2,
 iccg_state3,
 iccg_state4,
 iccg_state5,
 iccg_state6,
 iccg_state7,

 // ccgen_to_clipcc/clip
 ioutsm_clr_orig_vertices,
 iccgen_to_clipcc_fifo_full,

 // arbiter
 iarb_to_ccgen_xfc,


 ////////////////////////////////////////////////////////////////
 // OUTPUTS
 ////////////////////////////////////////////////////////////////
```

```
// state
osxif_st_r_data,

// state to clipper
osxif_state0,
osxif_state1,
osxif_state2,
osxif_state3,
osxif_state4,
osxif_state5,
osxif_state6,
osxif_state7,

// vgt_to_ccgen fifo
ovgt_to_ccgen_fifo_notfull,

// sx0, request
opa_to_sx0_req,
opa_to_sx0_sp_id,
opa_to_sx0_offset,
opa_to_sx0_aux,
opa_to_sx0_last,

// sx1, request
opa_to_sx1_req,
  opa_to_sx1_sp_id,
  opa_to_sx1_offset,
  opa_to_sx1_aux,
  opa_to_sx1_last,

  // position memory
  oposition_write,
  oposition_wraddr,
  oposition_wrdata,

  // point memory
  opoint_write,
  opoint_wraddr,
  opoint_wrdata,

  // ccgen_to_clipcc/clip
  occgen_to_clipcc_data,
  occgen_to_clipcc_write,

  // arbiter
  occgen_to_arb_data
);

`include "pa_clip_pkg.v"

```

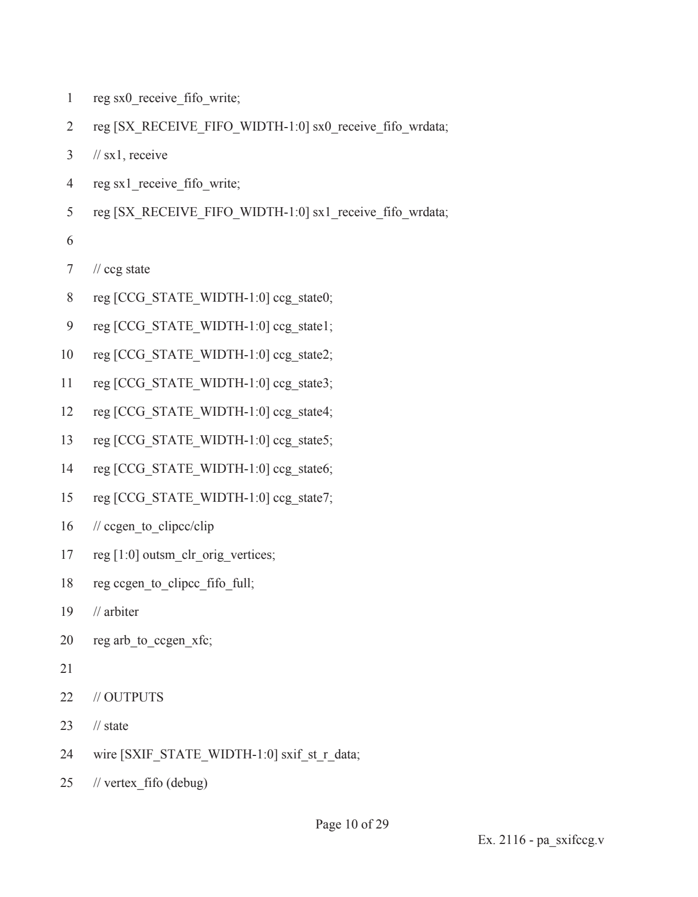
```
////////////////////////////////////////////////////////////////
// INPUT DECLARATIONS
////////////////////////////////////////////////////////////////
// global
input clk;
input reset;
// ati_state_storage (sxif)
input [SXIF_STATE_WIDTH-1:0] isxif_st_w_data;
input [2:0] isxif_st_w_addr;
input isxif_st_we;
input [2:0] isxif_st_r_addr;
input isxif_st_re;
input isxif_st_sel;
input isxif_st_cpy;
// vgt_to_ccgen fifo
input ivgt_to_ccgen_fifo_write;
input [5:0] ivgt_to_ccgen_fifo_active_verts;
input [2:0] ivgt_to_ccgen_fifo_state_var_indx;
// sx0, receive
input isx0_receive_fifo_write;
input [SX_RECEIVE_FIFO_WIDTH-1:0] isx0_receive_fifo_wrdata;
// sx1, receive
input isx1_receive_fifo_write;
input [SX_RECEIVE_FIFO_WIDTH-1:0] isx1_receive_fifo_wrdata;

```

```
// ccg state
input [CCG_STATE_WIDTH-1:0] iccg_state0;
input [CCG_STATE_WIDTH-1:0] iccg_state1;
input [CCG_STATE_WIDTH-1:0] iccg_state2;
input [CCG_STATE_WIDTH-1:0] iccg_state3;
input [CCG_STATE_WIDTH-1:0] iccg_state4;
input [CCG_STATE_WIDTH-1:0] iccg_state5;
input [CCG_STATE_WIDTH-1:0] iccg_state6;
input [CCG_STATE_WIDTH-1:0] iccg_state7;
// ccgen_to_clipcc/clip
input [1:0] ioutsm_clr_orig_vertices;
input iccgen_to_clipcc_fifo_full;
// arbiter
input iarb_to_ccgen_xfc;

////////////////////////////////////////////////////////////////
// OUTPUT DECLARATIONS
////////////////////////////////////////////////////////////////
// state
output [SXIF_STATE_WIDTH-1:0] osxif_st_r_data;
// state to clipper
output [SXIF_STATE_WIDTH-1:0] osxif_state0;
output [SXIF_STATE_WIDTH-1:0] osxif_state1;
output [SXIF_STATE_WIDTH-1:0] osxif_state2;
output [SXIF_STATE_WIDTH-1:0] osxif_state3;
```

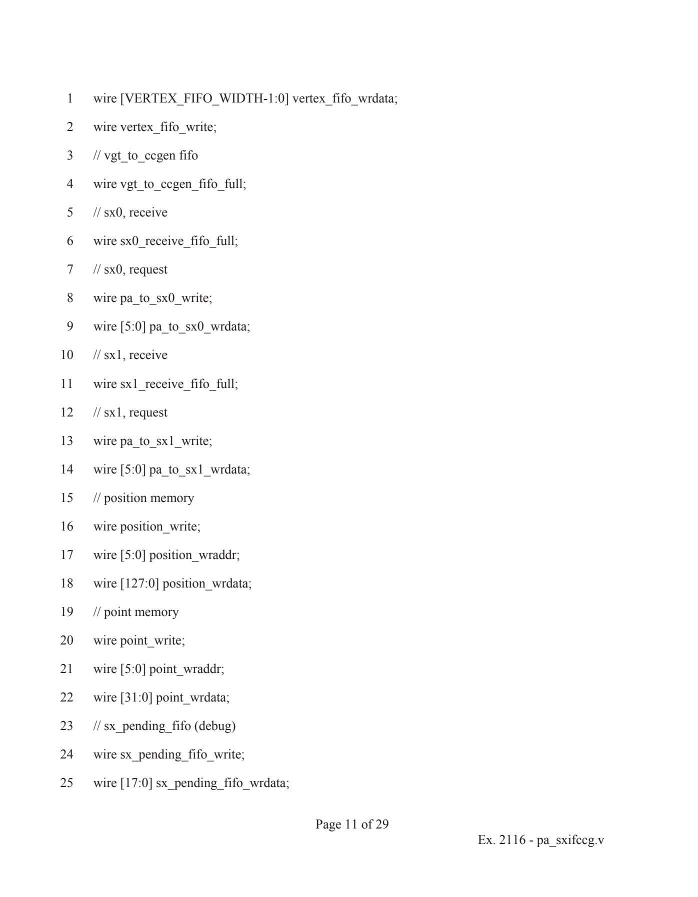
```
output [SXIF_STATE_WIDTH-1:0] osxif_state4;
output [SXIF_STATE_WIDTH-1:0] osxif_state5;
output [SXIF_STATE_WIDTH-1:0] osxif_state6;
output [SXIF_STATE_WIDTH-1:0] osxif_state7;
// vgt_to_ccgen fifo
output ovgt_to_ccgen_fifo_notfull;
// sx0, request
output opa_to_sx0_req;
output opa_to_sx0_sp_id;
output [1:0] opa_to_sx0_offset;
output opa_to_sx0_aux;
output opa_to_sx0_last;
// sx1, request
output opa_to_sx1_req;
output opa_to_sx1_sp_id;
output [1:0] opa_to_sx1_offset;
output opa_to_sx1_aux;
output opa_to_sx1_last;
// position memory
output oposition_write;
output [5:0] oposition_wraddr;
output [127:0] oposition_wrdata;
// point memory
output opoint_write;
output [5:0] opoint_wraddr;
```



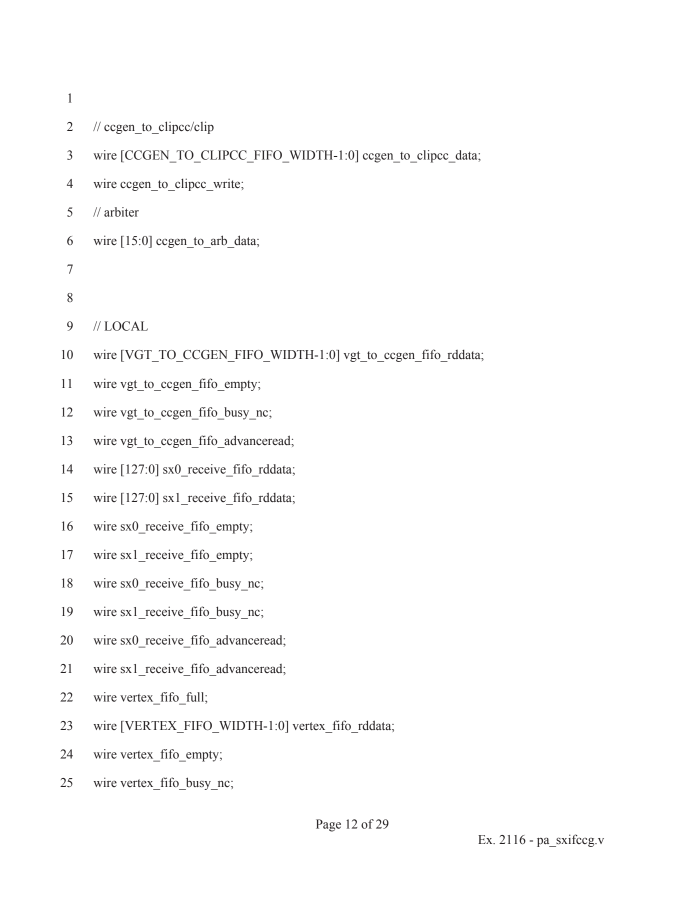
```
output [31:0] opoint_wrdata;

// ccgen_to_clipcc/clip
output [CCGEN_TO_CLIPCC_FIFO_WIDTH-1:0] occgen_to_clipcc_data;
output occgen_to_clipcc_write;
// arbiter
output [15:0] occgen_to_arb_data;


////////////////////////////////////////////////////////////////
// SIGNAL DECLARATIONS
////////////////////////////////////////////////////////////////
// INPUTS
// ati_state_storage (sxif)
reg [SXIF_STATE_WIDTH-1:0] sxif_st_w_data;
reg [2:0] sxif_st_w_addr;
reg sxif_st_we;
reg [2:0] sxif_st_r_addr;
reg sxif_st_re;
reg sxif_st_sel;
reg sxif_st_cpy;
// vgt_to_ccgen fifo
reg vgt_to_ccgen_fifo_write;
reg [VGT_TO_CCGEN_FIFO_WIDTH-1:0] vgt_to_ccgen_fifo_wrdata;
// sx0, receive
```



```
reg sx0_receive_fifo_write;
reg [SX_RECEIVE_FIFO_WIDTH-1:0] sx0_receive_fifo_wrdata;
// sx1, receive
reg sx1_receive_fifo_write;
reg [SX_RECEIVE_FIFO_WIDTH-1:0] sx1_receive_fifo_wrdata;

// ccg state
reg [CCG_STATE_WIDTH-1:0] ccg_state0;
reg [CCG_STATE_WIDTH-1:0] ccg_state1;
reg [CCG_STATE_WIDTH-1:0] ccg_state2;
reg [CCG_STATE_WIDTH-1:0] ccg_state3;
reg [CCG_STATE_WIDTH-1:0] ccg_state4;
reg [CCG_STATE_WIDTH-1:0] ccg_state5;
reg [CCG_STATE_WIDTH-1:0] ccg_state6;
reg [CCG_STATE_WIDTH-1:0] ccg_state7;
// ccgen_to_clipcc/clip
reg [1:0] outsm_clr_orig_vertices;
reg ccgen_to_clipcc_fifo_full;
// arbiter
reg arb_to_ccgen_xfc;

// OUTPUTS
// state
wire [SXIF_STATE_WIDTH-1:0] sxif_st_r_data;
// vertex_fifo (debug)
```



```
wire [VERTEX_FIFO_WIDTH-1:0] vertex_fifo_wrdata;
wire vertex_fifo_write;
// vgt_to_ccgen fifo
wire vgt_to_ccgen_fifo_full;
// sx0, receive
wire sx0_receive_fifo_full;
// sx0, request
wire pa_to_sx0_write;
wire [5:0] pa_to_sx0_wrdata;
// sx1, receive
wire sx1_receive_fifo_full;
// sx1, request
wire pa_to_sx1_write;
wire [5:0] pa_to_sx1_wrdata;
// position memory
wire position_write;
wire [5:0] position_wraddr;
wire [127:0] position_wrdata;
// point memory
wire point_write;
wire [5:0] point_wraddr;
wire [31:0] point_wrdata;
// sx_pending_fifo (debug)
wire sx_pending_fifo_write;
wire [17:0] sx_pending_fifo_wrdata;
```



```

// ccgen_to_clipcc/clip
wire [CCGEN_TO_CLIPCC_FIFO_WIDTH-1:0] ccgen_to_clipcc_data;
wire ccgen_to_clipcc_write;
// arbiter
wire [15:0] ccgen_to_arb_data;


// LOCAL
wire [VGT_TO_CCGEN_FIFO_WIDTH-1:0] vgt_to_ccgen_fifo_rddata;
wire vgt_to_ccgen_fifo_empty;
wire vgt_to_ccgen_fifo_busy_nc;
wire vgt_to_ccgen_fifo_advanceread;
wire [127:0] sx0_receive_fifo_rddata;
wire [127:0] sx1_receive_fifo_rddata;
wire sx0_receive_fifo_empty;
wire sx1_receive_fifo_empty;
wire sx0_receive_fifo_busy_nc;
wire sx1_receive_fifo_busy_nc;
wire sx0_receive_fifo_advanceread;
wire sx1_receive_fifo_advanceread;
wire vertex_fifo_full;
wire [VERTEX_FIFO_WIDTH-1:0] vertex_fifo_rddata;
wire vertex_fifo_empty;
wire vertex_fifo_busy_nc;
wire vertex_fifo_advanceread;
wire [6:0] available_positions;
wire decrement_available_positions;
wire [SXIF_STATE_WIDTH-1:0] sxif_st_data0;
wire [SXIF_STATE_WIDTH-1:0] sxif_st_data1;
wire [SXIF_STATE_WIDTH-1:0] sxif_st_data2;
wire [SXIF_STATE_WIDTH-1:0] sxif_st_data3;
wire [SXIF_STATE_WIDTH-1:0] sxif_st_data4;
wire [SXIF_STATE_WIDTH-1:0] sxif_st_data5;
wire [SXIF_STATE_WIDTH-1:0] sxif_st_data6;
wire [SXIF_STATE_WIDTH-1:0] sxif_st_data7;

////////////////////////////////////////////////////////////////////
// MAP INPUTS
////////////////////////////////////////////////////////////////////
always @(
        // ati_state_storage (sxif)
        isxif_st_w_data or
        isxif_st_w_addr or
        isxif_st_we or
        isxif_st_r_addr or
        isxif_st_re or
        isxif_st_sel or
        isxif_st_cpy or
        // vgt_to_ccgen fifo
```

```
        ivgt_to_ccgen_fifo_write or
        ivgt_to_ccgen_fifo_active_verts or
        ivgt_to_ccgen_fifo_state_var_indx or
        // sx0
        isx0_receive_fifo_write or
        isx0_receive_fifo_wrdata or
        // sx1
        isx1_receive_fifo_write or
        isx1_receive_fifo_wrdata or
        // vertex fifo/ccg

        // ccg state
        iccg_state0 or
        iccg_state1 or
        iccg_state2 or
        iccg_state3 or
        iccg_state4 or
        iccg_state5 or
        iccg_state6 or
        iccg_state7 or

        // ccgen_to_clipcc/clip
        ioutsm_clr_orig_vertices or
        iccgen_to_clipcc_fifo_full or

```

```
        // arbiter
        iarb_to_ccgen_xfc
       ) begin : proc000
  // ati_state_storage
  sxif_st_w_data = isxif_st_w_data;
  sxif_st_w_addr = isxif_st_w_addr;
  sxif_st_we = isxif_st_we;
  sxif_st_r_addr = isxif_st_r_addr;
  sxif_st_re = isxif_st_re;
  sxif_st_sel = isxif_st_sel;
  sxif_st_cpy = isxif_st_cpy;

  // vgt_to_ccgen fifo
  vgt_to_ccgen_fifo_write = ivgt_to_ccgen_fifo_write;
  vgt_to_ccgen_fifo_wrdata = {ivgt_to_ccgen_fifo_active_verts,
                              ivgt_to_ccgen_fifo_state_var_indx};
  // sx0
  sx0_receive_fifo_write = isx0_receive_fifo_write;
  sx0_receive_fifo_wrdata = isx0_receive_fifo_wrdata;
  // sx1
  sx1_receive_fifo_write = isx1_receive_fifo_write;
  sx1_receive_fifo_wrdata = isx1_receive_fifo_wrdata;

  // ccg state
  ccg_state0 = iccg_state0;
```

```
  ccg_state1 = iccg_state1;
  ccg_state2 = iccg_state2;
  ccg_state3 = iccg_state3;
  ccg_state4 = iccg_state4;
  ccg_state5 = iccg_state5;
  ccg_state6 = iccg_state6;
  ccg_state7 = iccg_state7;

  // ccgen_to_clipcc/clip
  outsm_clr_orig_vertices = ioutsm_clr_orig_vertices;
  ccgen_to_clipcc_fifo_full = iccgen_to_clipcc_fifo_full;

  // arbiter
  arb_to_ccgen_xfc = iarb_to_ccgen_xfc;
end


////////////////////////////////////////////////////////////////////
// MAP OUTPUTS
////////////////////////////////////////////////////////////////////
// state
assign osxif_st_r_data = sxif_st_r_data;

// state to clipper
assign osxif_state0 = sxif_st_data0;
assign osxif_state1 = sxif_st_data1;
assign osxif_state2 = sxif_st_data2;
assign osxif_state3 = sxif_st_data3;
assign osxif_state4 = sxif_st_data4;
assign osxif_state5 = sxif_st_data5;
assign osxif_state6 = sxif_st_data6;
assign osxif_state7 = sxif_st_data7;

// vgt_to_ccgen fifo
assign ovgt_to_ccgen_fifo_notfull = ~vgt_to_ccgen_fifo_full;
// sx0, request
assign {opa_to_sx0_req,
        opa_to_sx0_sp_id,
        opa_to_sx0_offset,
        opa_to_sx0_aux,
        opa_to_sx0_last} = pa_to_sx0_wrdata;
// sx1, request
assign {opa_to_sx1_req,
        opa_to_sx1_sp_id,
        opa_to_sx1_offset,
        opa_to_sx1_aux,
        opa_to_sx1_last} = pa_to_sx1_wrdata;
// position memory
assign oposition_write = position_write;
assign oposition_wraddr = position_wraddr;
```

```
assign oposition_wrdata = position_wrdata;
// point memory
assign opoint_write = point_write;
assign opoint_wraddr = point_wraddr;
assign opoint_wrdata = point_wrdata;

// ccgen_to_clipcc/clip
assign occgen_to_clipcc_data = ccgen_to_clipcc_data;
assign occgen_to_clipcc_write = ccgen_to_clipcc_write;
// arbiter
assign occgen_to_arb_data = ccgen_to_arb_data;


////////////////////////////////////////////////////////////////////////
// vgt_to_ccgen fifo
////////////////////////////////////////////////////////////////////////
ati_fifo_top
#(
  VGT_TO_CCGEN_FIFO_WIDTH,
  VGT_TO_CCGEN_FIFO_POINTER_SIZE,
  VGT_TO_CCGEN_FIFO_DEPTH,
  VGT_TO_CCGEN_FIFO_SKID_WORDS
 )
uvgt_to_ccgen_fifo
 (
```

```
  .write_data (vgt_to_ccgen_fifo_wrdata),
  .we         (vgt_to_ccgen_fifo_write),
  .full       (vgt_to_ccgen_fifo_full),
  .read_data  (vgt_to_ccgen_fifo_rddata),
  .empty      (vgt_to_ccgen_fifo_empty),
  .busy       (vgt_to_ccgen_fifo_busy_nc),
  .re         (vgt_to_ccgen_fifo_advanceread),
  .clk        (clk),
  .reset      (reset)
 );


////////////////////////////////////////////////////////////////////////
// sx0 receive fifo
////////////////////////////////////////////////////////////////////////
ati_fifo_top
#(
  SX_RECEIVE_FIFO_WIDTH,
  SX_RECEIVE_FIFO_POINTER_SIZE,
  SX_RECEIVE_FIFO_DEPTH,
  SX_RECEIVE_FIFO_SKID_WORDS
 )
usx0_receive_fifo
 (
  .write_data (sx0_receive_fifo_wrdata),
```

```
  .we         (sx0_receive_fifo_write),
  .full       (sx0_receive_fifo_full),
  .read_data  (sx0_receive_fifo_rddata),
  .empty      (sx0_receive_fifo_empty),
  .busy       (sx0_receive_fifo_busy_nc),
  .re         (sx0_receive_fifo_advanceread),
  .clk        (clk),
  .reset      (reset)
 );


////////////////////////////////////////////////////////////////////////
// sx1 receive fifo
////////////////////////////////////////////////////////////////////////
ati_fifo_top
#(
  SX_RECEIVE_FIFO_WIDTH,
  SX_RECEIVE_FIFO_POINTER_SIZE,
  SX_RECEIVE_FIFO_DEPTH,
  SX_RECEIVE_FIFO_SKID_WORDS
 )
usx1_receive_fifo
 (
  .write_data (sx1_receive_fifo_wrdata),
  .we         (sx1_receive_fifo_write),
  .full       (sx1_receive_fifo_full),
  .read_data  (sx1_receive_fifo_rddata),
  .empty      (sx1_receive_fifo_empty),
  .busy       (sx1_receive_fifo_busy_nc),
  .re         (sx1_receive_fifo_advanceread),
  .clk        (clk),
  .reset      (reset)
 );


/////////////////////////////////////////////////////////////////
// vertex_fifo fifo
/////////////////////////////////////////////////////////////////
ati_fifo_top
#(
  VERTEX_FIFO_WIDTH,
  VERTEX_FIFO_POINTER_SIZE,
  VERTEX_FIFO_DEPTH,
  VERTEX_FIFO_SKID_WORDS
 )
uvertex_fifo
 (
  .write_data (vertex_fifo_wrdata),
  .we         (vertex_fifo_write),
  .full       (vertex_fifo_full),
```

```
  .read_data  (vertex_fifo_rddata),
  .empty      (vertex_fifo_empty),
  .busy       (vertex_fifo_busy_nc),
  .re         (vertex_fifo_advanceread),
  .clk        (clk),
  .reset      (reset)
 );

/////////////////////////////////////////////////////////////////
// ati_8rp_state_storage
/////////////////////////////////////////////////////////////////
ati_8rp_state_storage
#(
  SXIF_STATE_WIDTH,
  SXIF_STATES
 )
uati_8rp_state_storage
 (
  .st_data0     (sxif_st_data0),
  .st_data1     (sxif_st_data1),
  .st_data2     (sxif_st_data2),
  .st_data3     (sxif_st_data3),
  .st_data4     (sxif_st_data4),
  .st_data5     (sxif_st_data5),
  .st_data6     (sxif_st_data6),
```

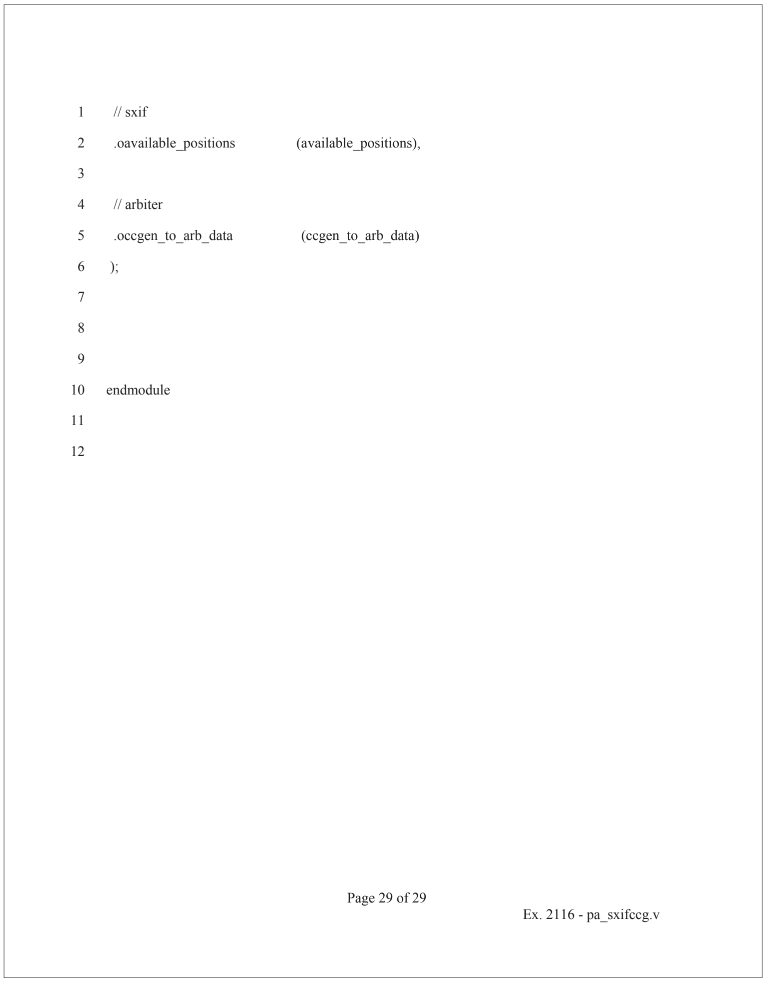
```
  .st_data7     (sxif_st_data7),
  .r_data       (sxif_st_r_data),
  .w_data       (sxif_st_w_data),
  .w_addr       (sxif_st_w_addr),
  .r_addr       (sxif_st_r_addr),
  .we           (sxif_st_we),
  .re           (sxif_st_re),
  .sel          (sxif_st_sel),
  .cpy          (sxif_st_cpy),
  .clk          (clk)
 );



/////////////////////////////////////////////////////////////////
// sxif state machine
/////////////////////////////////////////////////////////////////
pa_ccg_sxifsm
upa_ccg_sxifsm
 (
  /////////////////////////////////////////////////////////////
  // GLOBAL SIGNALS
  /////////////////////////////////////////////////////////////
  .clk                              (clk),
  .reset                            (reset),
```

```

  /////////////////////////////////////////////////////////////
  // INPUTS
  /////////////////////////////////////////////////////////////
  // vgt_to_ccgen fifo
  .ivgt_to_ccgen_fifo_empty          (vgt_to_ccgen_fifo_empty),
  .ivgt_to_ccgen_fifo_rddata         (vgt_to_ccgen_fifo_rddata),

  // state
  .isxif_state0                      (sxif_st_data0),
  .isxif_state1                      (sxif_st_data1),
  .isxif_state2                      (sxif_st_data2),
  .isxif_state3                      (sxif_st_data3),
  .isxif_state4                      (sxif_st_data4),
  .isxif_state5                      (sxif_st_data5),
  .isxif_state6                      (sxif_st_data6),
  .isxif_state7                      (sxif_st_data7),

  // vertex_fifo
  .ivertex_fifo_full                 (vertex_fifo_full),
  .ivertex_fifo_advanceread          (vertex_fifo_advanceread),

  // ccg
  .iavailable_positions              (available_positions),

// sx_to_pa_0
.isx_to_pa_empty_0              (sx0_receive_fifo_empty),
.isx_to_pa_vector_0             (sx0_receive_fifo_rddata),

// sx_to_pa_1
.isx_to_pa_empty_1              (sx1_receive_fifo_empty),
.isx_to_pa_vector_1             (sx1_receive_fifo_rddata),


//////////////////////////////////////////////////////////
// OUTPUTS
//////////////////////////////////////////////////////////
// vgt_to_ccgen fifo
.ovgt_to_ccgen_fifo_advanceread     (vgt_to_ccgen_fifo_advanceread),

// pa_to_sx0
.opa_to_sx0_write               (pa_to_sx0_write),
.opa_to_sx0_wrdata              (pa_to_sx0_wrdata),

// pa_to_sx1
.opa_to_sx1_write               (pa_to_sx1_write),
.opa_to_sx1_wrdata              (pa_to_sx1_wrdata),

// position memory
.omem_position_write            (position_write),
```

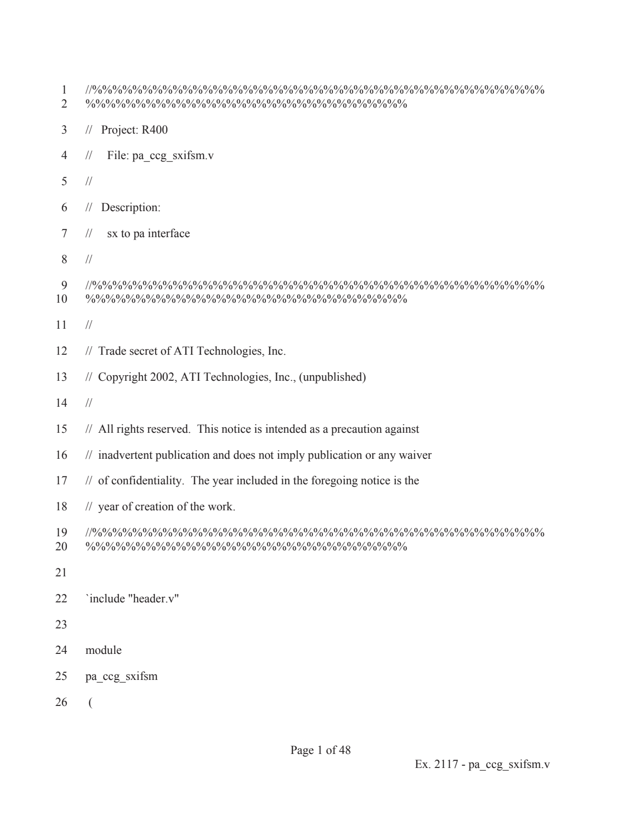
```
.omem_position_wraddr           (position_wraddr),
.omem_position_wrdata           (position_wrdata),

// point memory
.omem_point_write               (point_write),
.omem_point_wraddr              (point_wraddr),
.omem_point_wrdata              (point_wrdata),

// ccg
.odecrement_available_positions     (decrement_available_positions),

// vertex_fifo
.overtex_fifo_write             (vertex_fifo_write),
.overtex_fifo_wrdata            (vertex_fifo_wrdata),

// sx_to_pa_0
.osx_to_pa_advanceread_0        (sx0_receive_fifo_advanceread),

// sx_to_pa_1
.osx_to_pa_advanceread_1        (sx1_receive_fifo_advanceread),

// debug only
.osx_pending_fifo_write         (sx_pending_fifo_write),
.osx_pending_fifo_wrdata        (sx_pending_fifo_wrdata)
);
```

```


pa_ccg_ccgsm
upa_ccg_ccgsm
(
//////////////////////////////////////////////////////////////
// COMMON
//////////////////////////////////////////////////////////////
.clk                            (clk),
.reset                          (reset),

//////////////////////////////////////////////////////////////
// INPUTS
//////////////////////////////////////////////////////////////
// vertex fifo
.ivertex_fifo_empty             (vertex_fifo_empty),
.ivertex_fifo_rd_data           (vertex_fifo_rddata),

// state
.iccg_state0                    (ccg_state0),
.iccg_state1                    (ccg_state1),
.iccg_state2                    (ccg_state2),
.iccg_state3                    (ccg_state3),
.iccg_state4                    (ccg_state4),
.iccg_state5                    (ccg_state5),
```

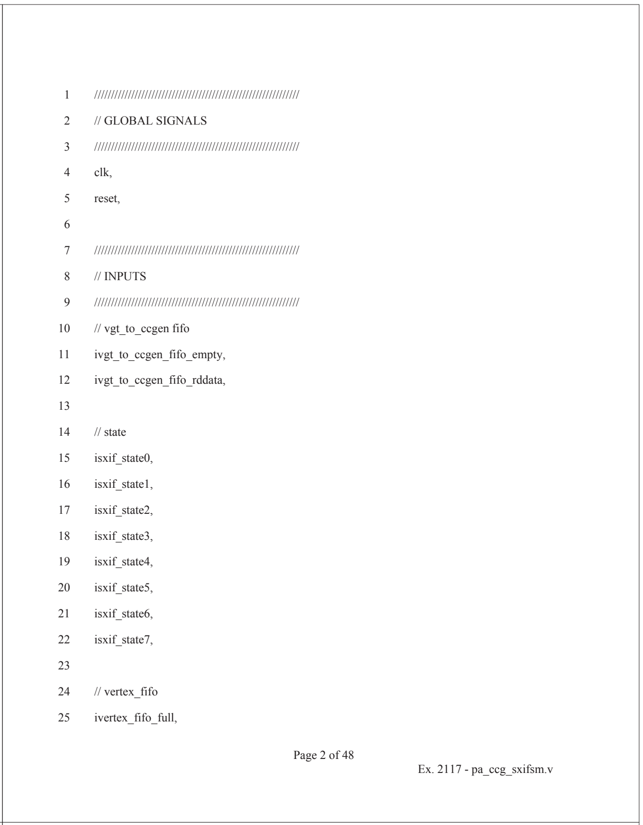
```
.iccg_state6                    (ccg_state6),
.iccg_state7                    (ccg_state7),

// clipper
.ioutsm_clr_orig_vertices       (outsm_clr_orig_vertices),

// arbiter
.iarb_to_ccgen_xfc              (arb_to_ccgen_xfc),

// ccgen_to_clipcc
.iccgen_to_clipcc_fifo_full     (ccgen_to_clipcc_fifo_full),

// sxif
.idecrement_available_positions     (decrement_available_positions),

//////////////////////////////////////////////////////////////
// OUTPUTS
//////////////////////////////////////////////////////////////
// vertex fifo
.overtex_fifo_advanceread       (vertex_fifo_advanceread),

// ccgen_to_clipcc
.occgen_to_clipcc_data          (ccgen_to_clipcc_data),
.occgen_to_clipcc_write         (ccgen_to_clipcc_write),

```



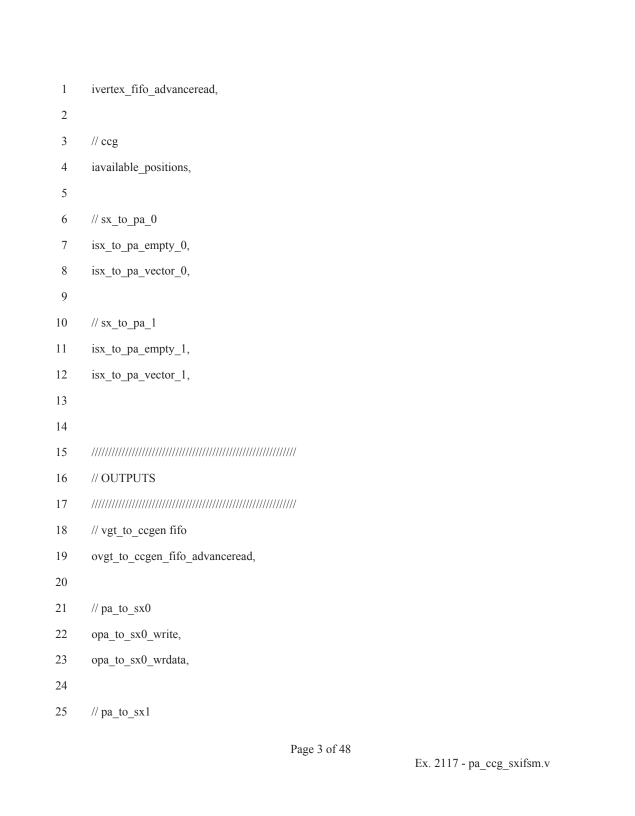
```
  // sxif
  .oavailable_positions          (available_positions),

  // arbiter
  .occgen_to_arb_data            (ccgen_to_arb_data)
 );



endmodule
```



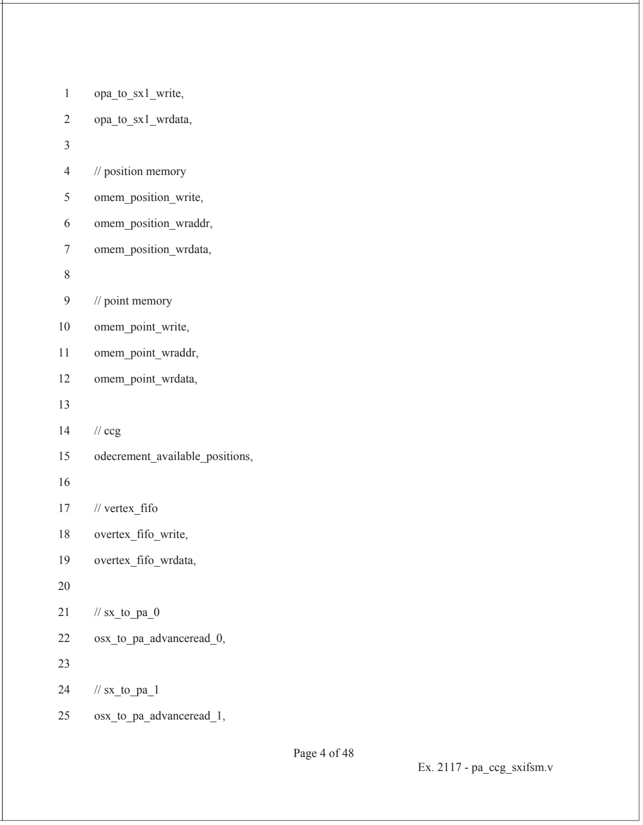
```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//   Project: R400
//     File: pa_ccg_sxifsm.v
//
//   Description:
//     sx to pa interface
//
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//
//  Trade secret of ATI Technologies, Inc.
//  Copyright 2002, ATI Technologies, Inc., (unpublished)
//
//  All rights reserved.  This notice is intended as a precaution against
//  inadvertent publication and does not imply publication or any waiver
//  of confidentiality.  The year included in the foregoing notice is the
//  year of creation of the work.
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

`include "header.v"

module
pa_ccg_sxifsm
 (
```



```
////////////////////////////////////////////////////////////
// GLOBAL SIGNALS
////////////////////////////////////////////////////////////
clk,
reset,

////////////////////////////////////////////////////////////
// INPUTS
////////////////////////////////////////////////////////////
// vgt_to_ccgen fifo
ivgt_to_ccgen_fifo_empty,
ivgt_to_ccgen_fifo_rddata,

// state
isxif_state0,
isxif_state1,
isxif_state2,
isxif_state3,
isxif_state4,
isxif_state5,
isxif_state6,
isxif_state7,

// vertex_fifo
ivertex_fifo_full,
```



```
 ivertex_fifo_advanceread,

 // ccg
 iavailable_positions,

 // sx_to_pa_0
 isx_to_pa_empty_0,
 isx_to_pa_vector_0,

 // sx_to_pa_1
 isx_to_pa_empty_1,
 isx_to_pa_vector_1,


 ////////////////////////////////////////////////////////////
 // OUTPUTS
 ////////////////////////////////////////////////////////////
 // vgt_to_ccgen fifo
 ovgt_to_ccgen_fifo_advanceread,

 // pa_to_sx0
 opa_to_sx0_write,
 opa_to_sx0_wrdata,

 // pa_to_sx1
```



```
opa_to_sx1_write,
opa_to_sx1_wrdata,

// position memory
omem_position_write,
omem_position_wraddr,
omem_position_wrdata,

// point memory
omem_point_write,
omem_point_wraddr,
omem_point_wrdata,

// ccg
odecrement_available_positions,

// vertex_fifo
overtex_fifo_write,
overtex_fifo_wrdata,

// sx_to_pa_0
osx_to_pa_advanceread_0,

// sx_to_pa_1
osx_to_pa_advanceread_1,

   // debug only
   osx_pending_fifo_write,
   osx_pending_fifo_wrdata
  );

`include "pa_clip_pkg.v"

//////////////////////////////////////////////////////////////////////
// INPUT DECLARATIONS
//////////////////////////////////////////////////////////////////////
// global
input clk;
input reset;

// vgt_to_ccgen fifo
input ivgt_to_ccgen_fifo_empty;
input [VGT_TO_CCGEN_FIFO_WIDTH-1:0] ivgt_to_ccgen_fifo_rddata;

// state
input [SXIF_STATE_WIDTH-1:0] isxif_state0;
input [SXIF_STATE_WIDTH-1:0] isxif_state1;
input [SXIF_STATE_WIDTH-1:0] isxif_state2;
input [SXIF_STATE_WIDTH-1:0] isxif_state3;
input [SXIF_STATE_WIDTH-1:0] isxif_state4;
```

```
input [SXIF_STATE_WIDTH-1:0] isxif_state5;
input [SXIF_STATE_WIDTH-1:0] isxif_state6;
input [SXIF_STATE_WIDTH-1:0] isxif_state7;

// vertex_fifo
input ivertex_fifo_full;
input ivertex_fifo_advanceread;

// ccg
input [6:0] iavailable_positions;

// sx_to_pa_0
input isx_to_pa_empty_0;
input [127:0] isx_to_pa_vector_0;

// sx_to_pa_1
input isx_to_pa_empty_1;
input [127:0] isx_to_pa_vector_1;

//////////////////////////////////////////////////////////////////////
// OUTPUT DECLARATIONS
//////////////////////////////////////////////////////////////////////
// vgt_to_ccgen fifo
output ovgt_to_ccgen_fifo_advanceread;

```

```
// pa_to_sx_0
output opa_to_sx0_write;
output [5:0] opa_to_sx0_wrdata;

// pa_to_sx_1
output opa_to_sx1_write;
output [5:0] opa_to_sx1_wrdata;

// position memory
output omem_position_write;
output [5:0] omem_position_wraddr;
output [127:0] omem_position_wrdata;

// point memory
output omem_point_write;
output [5:0] omem_point_wraddr;
output [31:0] omem_point_wrdata;

// ccg
output odecrement_available_positions;

// vertex_fifo
output overtex_fifo_write;
output [VERTEX_FIFO_WIDTH-1:0] overtex_fifo_wrdata;

```

```
// sx_to_pa_0
output osx_to_pa_advanceread_0;

// sx_to_pa_1
output osx_to_pa_advanceread_1;

// debug only
output osx_pending_fifo_write;
output [17:0] osx_pending_fifo_wrdata;

//////////////////////////////////////////////////////////////////////
// SIGNAL DECLARATIONS
//////////////////////////////////////////////////////////////////////
// inputs
// vgt_to_ccgen fifo
reg vgt_to_ccgen_fifo_empty;
reg [6:0] vgt_to_ccgen_active_verts;
reg [2:0] vgt_to_ccgen_state_var_indx;
// state
reg [SXIF_STATE_WIDTH-1:0] sxif_state0;
reg [SXIF_STATE_WIDTH-1:0] sxif_state1;
reg [SXIF_STATE_WIDTH-1:0] sxif_state2;
reg [SXIF_STATE_WIDTH-1:0] sxif_state3;
reg [SXIF_STATE_WIDTH-1:0] sxif_state4;
reg [SXIF_STATE_WIDTH-1:0] sxif_state5;
reg [SXIF_STATE_WIDTH-1:0] sxif_state6;
reg [SXIF_STATE_WIDTH-1:0] sxif_state7;
// vertex_fifo
reg vertex_fifo_full;
reg vertex_fifo_advanceread;
// ccg
reg [6:0] available_positions;
// sx_to_pa
reg [1:0] sx_to_pa_empty;
reg [127:0] sx_to_pa_vector[0:1];

// outputs
// vgt_to_ccgen fifo
reg vgt_to_ccgen_fifo_advanceread;
// pa_to_sx
reg pa_to_sx_write[0:1];
reg pa_to_sx_req[0:1];
reg pa_to_sx_sp_id[0:1];
reg [1:0] pa_to_sx_offset[0:1];
reg pa_to_sx_aux[0:1];
reg pa_to_sx_last[0:1];
// position memory
reg tcl_scratch_mem_position_write;
reg [5:0] position_address;
reg [127:0] tcl_scratch_mem_position_data;
```

```
// point memory
reg tcl_scratch_mem_point_write;
reg [5:0] point_address;
reg [31:0] tcl_scratch_mem_point_data;
// ccg
reg decrement_available_positions;
// vertex_fifo
reg vertex_fifo_write;
reg [10:0] vertex_fifo_wr_param_cache_indx;
reg [2:0] vertex_fifo_wr_state_var_indx;
reg vertex_fifo_wr_edge_flag;
reg vertex_fifo_wr_kill_flag;
// sx_to_pa
reg [1:0] sx_to_pa_advanceread;
// debug only
reg sx_pending_fifo_write;
reg [17:0] sx_pending_fifo_wrdata;

// local
reg [4:0] statevar_bits_vert_param_cache_size;
reg statevar_bits_use_vtx_point_size;
reg statevar_bits_sxpa_aux_vector;
reg statevar_bits_use_vtx_edge_flag;
reg statevar_bits_use_vtx_kill_flag;
reg [5:0] next_sx_request_indx;
```

```
reg [5:0] sx_request_indx;
reg [1:0] pasx_req_cnt[0:1];
reg [1:0] next_sx_sent;
reg [1:0] sx_sent;
reg [1:0] increment_pasx_req_cnt;
reg [1:0] decrement_pasx_req_cnt;
reg sx_pending_wr_sx_sel;
reg sx_pending_wr_sp_id;
reg sx_pending_wr_aux_sel;
reg sx_pending_wr_aux_inc;
reg [6:0] sx_pending_wr_pci;
reg [3:0] sx_pending_wr_req_mask;
reg [2:0] sx_pending_wr_state_var_indx;
reg [1:0] sx_pending_fifo_wraddr;
reg [17:0] sx_pending_fifo_rddata;
reg [1:0] sx_pending_fifo_rdaddr;
reg [2:0] sx_pending_fifo_contents;
reg [17:0] sx_pending_fifo[0:3];
reg sx_pending_fifo_empty;
reg sx_pending_fifo_full;
reg sx_pending_fifo_advanceread;
reg next_sx_pending_rd_sx_sel;
reg next_sx_pending_rd_sp_id;
reg next_sx_pending_rd_aux_sel;
reg next_sx_pending_rd_aux_inc;
```

```
reg [6:0] next_sx_pending_rd_pci;
reg [3:0] next_sx_pending_rd_req_mask;
reg [2:0] next_sx_pending_rd_state_var_indx;
reg initial_sx_pending_rd_sx_sel;
reg initial_sx_pending_rd_sp_id;
reg initial_sx_pending_rd_aux_sel;
reg initial_sx_pending_rd_aux_inc;
reg [6:0] initial_sx_pending_rd_pci;
reg [3:0] initial_sx_pending_rd_req_mask;
reg [2:0] initial_sx_pending_rd_state_var_indx;
reg sx_pending_rd_sx_sel;
reg sx_pending_rd_sp_id;
reg sx_pending_rd_aux_sel;
reg sx_pending_rd_aux_inc;
reg [6:0] sx_pending_rd_pci;
reg [3:0] sx_pending_rd_req_mask;
reg [2:0] sx_pending_rd_state_var_indx;
reg next_aux_sel;
reg aux_sel;
reg [6:0] next_param_cache_base;
reg [6:0] param_cache_base;
reg next_sx_aux;
reg sx_aux;
reg [2:0] next_sx_receive_indx;
reg [2:0] sx_receive_indx;
reg next_sx_pending_advance;
reg sx_pending_advance;
reg [5:0] next_point_address;
reg [5:0] next_position_address;
reg [3:0] vertex_fifo_entriesavailable;
reg next_pa_to_sx_write[0:1];
reg next_pa_to_sx_req[0:1];
reg next_pa_to_sx_sp_id[0:1];
reg [1:0] next_pa_to_sx_offset[0:1];
reg next_pa_to_sx_aux[0:1];
reg next_pa_to_sx_last[0:1];
reg [2:0] request_side_vs_export_mode;
reg [3:0] request_side_vs_export_count;
reg [2:0] receive_side_vs_export_mode;

// local variables
reg [SXIF_STATE_WIDTH-1:0] var80_sxif_state;
reg [SXIF_STATE_WIDTH-1:0] var85_sxif_state;
reg [5:0] var00_vgt_to_ccgen_active_verts;
reg var100_sx_sel;
reg var100_sp_id;
reg [1:0] var100_offset;
reg [6:0] var100_pci;
reg [7:0] var100_remaining_positions;
reg var100_sx_all_sent;
```

```
reg [2:0] var300_sx_pending_fifo_contents;
reg [2:0] var400_next_sx_receive_indx;
reg var400_ignore_this_cycle;
reg var400_sx_sel;
reg var400_vector_write;
reg var400_first_vector_write;
reg var400_last_vector_write;
reg [2:0] var400_vector_wr_cnt;
reg var400_next_sx_pending_rd_sx_sel;
reg var400_next_sx_pending_rd_sp_id;
reg var400_next_sx_pending_rd_aux_sel;
reg var400_next_sx_pending_rd_aux_inc;
reg [6:0] var400_next_sx_pending_rd_pci;
reg [3:0] var400_next_sx_pending_rd_req_mask;
reg [2:0] var400_next_sx_pending_rd_state_var_indx;
reg [1:0] var480_next_pasx_req_cnt_0;
reg [1:0] var480_next_pasx_req_cnt_1;
reg [3:0] var600_vertex_fifo_entriesavailable;
reg [127:0] var400_sx_to_pa_vector_var400_sx_sel;
reg [31:0] var400_sx_to_pa_vector_y_var400_sx_sel;
reg [31:0] var400_sx_to_pa_vector_z_var400_sx_sel;

// for-loop variables
integer indx100;
integer indx400;
```

```
integer indx477;

////////////////////////////////////////////////////////////////////////////
// map inputs
////////////////////////////////////////////////////////////////////////////
always @(
        ivgt_to_ccgen_fifo_empty or
        ivgt_to_ccgen_fifo_rddata or
        isxif_state0 or
        isxif_state1 or
        isxif_state2 or
        isxif_state3 or
        isxif_state4 or
        isxif_state5 or
        isxif_state6 or
        isxif_state7 or
        ivertex_fifo_full or
        ivertex_fifo_advanceread or
        iavailable_positions or
        isx_to_pa_empty_0 or
        isx_to_pa_empty_1 or
        isx_to_pa_vector_0 or
        isx_to_pa_vector_1 or
        var00_vgt_to_ccgen_active_verts
       ) begin : proc00
```

```

  // vgt_to_ccgen fifo
  vgt_to_ccgen_fifo_empty = ivgt_to_ccgen_fifo_empty;
 {var00_vgt_to_ccgen_active_verts,
  vgt_to_ccgen_state_var_indx} = ivgt_to_ccgen_fifo_rddata;
  vgt_to_ccgen_active_verts = {1'b0, var00_vgt_to_ccgen_active_verts};
  if (var00_vgt_to_ccgen_active_verts == 'h0) begin
    vgt_to_ccgen_active_verts = VERTICES_PER_SLOT;
  end

  // state
  sxif_state0 = isxif_state0;
  sxif_state1 = isxif_state1;
  sxif_state2 = isxif_state2;
  sxif_state3 = isxif_state3;
  sxif_state4 = isxif_state4;
  sxif_state5 = isxif_state5;
  sxif_state6 = isxif_state6;
  sxif_state7 = isxif_state7;

  // vertex_fifo
  vertex_fifo_full = ivertex_fifo_full;
  vertex_fifo_advanceread = ivertex_fifo_advanceread;

  // ccg
  available_positions = iavailable_positions;

  // sx_to_pa_0
  sx_to_pa_empty[0] = isx_to_pa_empty_0;
  sx_to_pa_vector[0] = isx_to_pa_vector_0;

  // sx_to_pa_1
  sx_to_pa_empty[1] = isx_to_pa_empty_1;
  sx_to_pa_vector[1] = isx_to_pa_vector_1;
end


////////////////////////////////////////////////////////////////////
// map outputs
////////////////////////////////////////////////////////////////////
// vgt_to_ccgen fifo
assign ovgt_to_ccgen_fifo_advanceread = vgt_to_ccgen_fifo_advanceread;

// pa_to_sx 0
assign opa_to_sx0_write = pa_to_sx_write[0];
assign opa_to_sx0_wrdata = {pa_to_sx_req[0],
                            pa_to_sx_sp_id[0],
                            pa_to_sx_offset[0],
                            pa_to_sx_aux[0],
                            pa_to_sx_last[0]};
```

```

// pa_to_sx 1
assign opa_to_sx1_write = pa_to_sx_write[1];
assign opa_to_sx1_wrdata = {pa_to_sx_req[1],
                            pa_to_sx_sp_id[1],
                            pa_to_sx_offset[1],
                            pa_to_sx_aux[1],
                            pa_to_sx_last[1]};

// position memory
assign omem_position_write = tcl_scratch_mem_position_write;
assign omem_position_wraddr = position_address;
assign omem_position_wrdata = tcl_scratch_mem_position_data;

// point memory
assign omem_point_write = tcl_scratch_mem_point_write;
assign omem_point_wraddr = point_address;
assign omem_point_wrdata = tcl_scratch_mem_point_data;

// ccg
assign odecrement_available_positions = decrement_available_positions;

// vertex_fifo
assign overtex_fifo_write = vertex_fifo_write;
assign overtex_fifo_wrdata = {vertex_fifo_wr_param_cache_indx,
```

```
                              vertex_fifo_wr_state_var_indx,
                              vertex_fifo_wr_edge_flag,
                              vertex_fifo_wr_kill_flag};

// sx_to_pa
assign osx_to_pa_advanceread_0 = sx_to_pa_advanceread[0];
assign osx_to_pa_advanceread_1 = sx_to_pa_advanceread[1];

// debug only
assign osx_pending_fifo_write = sx_pending_fifo_write;
assign osx_pending_fifo_wrdata = sx_pending_fifo_wrdata;

////////////////////////////////////////////////////////////////////
// state mux, request side
////////////////////////////////////////////////////////////////////
always @(
        sxif_state0 or
        sxif_state1 or
        sxif_state2 or
        sxif_state3 or
        sxif_state4 or
        sxif_state5 or
        sxif_state6 or
        sxif_state7 or
        vgt_to_ccgen_state_var_indx
```

```
        ) begin : proc80
  case(vgt_to_ccgen_state_var_indx)
    0 : var80_sxif_state = sxif_state0;
    1 : var80_sxif_state = sxif_state1;
    2 : var80_sxif_state = sxif_state2;
    3 : var80_sxif_state = sxif_state3;
    4 : var80_sxif_state = sxif_state4;
    5 : var80_sxif_state = sxif_state5;
    6 : var80_sxif_state = sxif_state6;
    7 : var80_sxif_state = sxif_state7;
    default : var80_sxif_state = sxif_state0;
  endcase

  request_side_vs_export_mode = var80_sxif_state[6:4];
  request_side_vs_export_count = var80_sxif_state[3:0];
end

////////////////////////////////////////////////////////////////////
// state mux, receive side
////////////////////////////////////////////////////////////////////
always @(
        sxif_state0 or
        sxif_state1 or
        sxif_state2 or
        sxif_state3 or
```

```
          sxif_state4 or
          sxif_state5 or
          sxif_state6 or
          sxif_state7 or
          next_sx_pending_rd_state_var_indx
        ) begin : proc85
  case(next_sx_pending_rd_state_var_indx)
    0 : var85_sxif_state = sxif_state0;
    1 : var85_sxif_state = sxif_state1;
    2 : var85_sxif_state = sxif_state2;
    3 : var85_sxif_state = sxif_state3;
    4 : var85_sxif_state = sxif_state4;
    5 : var85_sxif_state = sxif_state5;
    6 : var85_sxif_state = sxif_state6;
    7 : var85_sxif_state = sxif_state7;
    default : var85_sxif_state = sxif_state0;
  endcase

  receive_side_vs_export_mode = var85_sxif_state[6:4];
end

//////////////////////////////////////////////////////////////////////
// decode request side state
//////////////////////////////////////////////////////////////////////
always @(
```

```
          request_side_vs_export_mode or
          request_side_vs_export_count
        ) begin : proc90

  statevar_bits_vert_param_cache_size = request_side_vs_export_count + 1;

  if ((request_side_vs_export_mode != 0) &&
      (request_side_vs_export_mode != 7)) begin
    statevar_bits_sxpa_aux_vector = 'h1;
    end
  else begin
    statevar_bits_sxpa_aux_vector = 'h0;
  end
end

//////////////////////////////////////////////////////////////////////
// decode receive side state
//////////////////////////////////////////////////////////////////////
always @(
          receive_side_vs_export_mode
        ) begin : proc95
  if ((receive_side_vs_export_mode == 2) ||
      (receive_side_vs_export_mode == 5)) begin
    statevar_bits_use_vtx_point_size = 'h1;
    end
```

```
  else begin
    statevar_bits_use_vtx_point_size = 'h0;
  end

  if ((receive_side_vs_export_mode == 3) ||
      (receive_side_vs_export_mode == 6)) begin
    statevar_bits_use_vtx_edge_flag = 'h1;
    end
  else begin
    statevar_bits_use_vtx_edge_flag = 'h0;
  end

  if ((receive_side_vs_export_mode == 4) ||
      (receive_side_vs_export_mode == 5) ||
      (receive_side_vs_export_mode == 6)) begin
    statevar_bits_use_vtx_kill_flag = 'h1;
    end
  else begin
    statevar_bits_use_vtx_kill_flag = 'h0;
  end
end

//////////////////////////////////////////////////////////////////////
// PaSxRequest
//////////////////////////////////////////////////////////////////////
```

```
always @(
          aux_sel or
          next_sx_request_indx or
          pa_to_sx_last[0] or
          pa_to_sx_last[1] or
          param_cache_base or
          pasx_req_cnt[0] or
          pasx_req_cnt[1] or
          statevar_bits_sxpa_aux_vector or
          statevar_bits_vert_param_cache_size or
          sx_aux or
          sx_request_indx or
          sx_sent or
          var100_offset or
          var100_pci or
          var100_remaining_positions or
          var100_sp_id or
          var100_sx_sel or
          vgt_to_ccgen_active_verts or
          vgt_to_ccgen_fifo_empty or
          vgt_to_ccgen_state_var_indx
        ) begin : proc100
  // init variables
  var100_sx_sel = 'h0;
  var100_sp_id = 'h0;
```

```
var100_offset = 'h0;
var100_pci = param_cache_base + (var100_offset * statevar_bits_vert_param_cache_size);
var100_remaining_positions = 'h0;
var100_sx_all_sent = 'h1;

// defaults
increment_pasx_req_cnt[0] = 'h0;
increment_pasx_req_cnt[1] = 'h0;
next_pa_to_sx_req[0] = 'h0;
next_pa_to_sx_sp_id[0] = 'h0;
next_pa_to_sx_offset[0] = 'h0;
next_pa_to_sx_aux[0] = 'h0;
next_pa_to_sx_last[0] = 'h0;
next_pa_to_sx_write[0] = 'h0;
next_pa_to_sx_req[1] = 'h0;
next_pa_to_sx_sp_id[1] = 'h0;
next_pa_to_sx_offset[1] = 'h0;
next_pa_to_sx_aux[1] = 'h0;
next_pa_to_sx_last[1] = 'h0;
next_pa_to_sx_write[1] = 'h0;
next_aux_sel = aux_sel;
vgt_to_ccgen_fifo_advanceread = 'h0;
next_sx_request_indx = sx_request_indx;
next_param_cache_base = param_cache_base;
next_sx_aux = sx_aux;
```

```
sx_pending_fifo_write = 'h0;
sx_pending_wr_sx_sel = 'h0;
sx_pending_wr_sp_id = 'h0;
sx_pending_wr_aux_sel = 'h0;
sx_pending_wr_aux_inc = 'h0;
sx_pending_wr_pci = 'h0;
sx_pending_wr_req_mask = 'hf;
sx_pending_wr_state_var_indx = vgt_to_ccgen_state_var_indx;
next_sx_sent[0] = sx_sent[0];
next_sx_sent[1] = sx_sent[1];

if (vgt_to_ccgen_fifo_empty == 'h0) begin
  var100_sx_sel = sx_request_indx[SX_SEL_BIT];
  var100_sp_id = sx_request_indx[SP_ID_BIT];
  var100_offset = sx_request_indx[OFFSET_HI_BIT:OFFSET_LO_BIT];

  var100_pci = param_cache_base + (var100_offset * statevar_bits_vert_param_cache_size);

  if (pasx_req_cnt[var100_sx_sel] < ACTIVE_PASX_REQUESTS) begin
    increment_pasx_req_cnt[var100_sx_sel] = 'h1;

    var100_remaining_positions = vgt_to_ccgen_active_verts - sx_request_indx;
    if (var100_remaining_positions[7] == 'h1) begin
      var100_remaining_positions = 'h0;
    end
```

```

    sx_pending_wr_sx_sel  = var100_sx_sel;
    sx_pending_wr_sp_id   = var100_sp_id;
    sx_pending_wr_aux_sel = aux_sel;

    sx_pending_wr_aux_inc = 'h0;
    if (statevar_bits_sxpa_aux_vector == 'h0) begin
      sx_pending_wr_aux_inc = 'h1;
    end

    sx_pending_wr_pci = var100_pci;

    case(var100_remaining_positions)
      0 : begin
        sx_pending_wr_req_mask = 'h0;
      end

      1 : begin
        sx_pending_wr_req_mask = 'h1;
      end

      2 : begin
        sx_pending_wr_req_mask = 'h3;
      end

```

```
      3 : begin
        sx_pending_wr_req_mask = 'h7;
      end

      default : begin
        sx_pending_wr_req_mask = 'hf;
      end
    endcase

    sx_pending_wr_state_var_indx = vgt_to_ccgen_state_var_indx;

    sx_pending_fifo_write = 'h1;

    next_pa_to_sx_req[var100_sx_sel]    = 'h1;
    next_pa_to_sx_sp_id[var100_sx_sel]  = var100_sp_id;
    next_pa_to_sx_offset[var100_sx_sel] = var100_offset;
    next_pa_to_sx_aux[var100_sx_sel]    = sx_aux;

    next_pa_to_sx_last[var100_sx_sel]   = 'h0;
    if (var100_remaining_positions <= VECTORS_PER_SX_REQUEST_SET) begin
      next_pa_to_sx_last[var100_sx_sel]   = 'h1;
    end

    next_pa_to_sx_write[var100_sx_sel] = 'h1;

```

```
   var100_sx_all_sent = 'h1;
   for (indx100 = 0; indx100 < PASX_INTERFACES; indx100 = indx100 + 1) begin
     var100_sx_all_sent = var100_sx_all_sent & ((var100_sx_sel == indx100) |
(sx_sent[indx100] == 'h1));
   end

   if ((aux_sel == 'h1)                  ||
       (statevar_bits_sxpa_aux_vector == 'h0))   begin
     next_aux_sel = 'h0;

     if ((var100_remaining_positions <= VECTORS_PER_SX_REQUEST) &&
(var100_sx_all_sent == 'h1)) begin
       vgt_to_ccgen_fifo_advanceread = 'h1;

       next_sx_request_indx = 'h0;

       next_param_cache_base = param_cache_base +
                           (statevar_bits_vert_param_cache_size * US_ALU_VE_MEMORIES);

       for (indx100 = 0; indx100 < PASX_INTERFACES; indx100 = indx100 + 1) begin
         next_sx_sent[indx100] = 'h0;
       end
       end
     else begin
       next_sx_request_indx = next_sx_request_indx + VECTORS_PER_SX_REQUEST;
       next_sx_sent[var100_sx_sel] = 'h1;
```

```
         end
         end
       else begin
         next_aux_sel = 'h1;
       end

       if (next_pa_to_sx_last[var100_sx_sel] == 'h1) begin
         next_sx_aux = 'h0;
         end
       else if (statevar_bits_sxpa_aux_vector == 'h1) begin
         next_sx_aux = ~sx_aux;
       end
     end
   end
end

////////////////////////////////////////////////////////////////////
// sx_pending_fifo_wrdata
////////////////////////////////////////////////////////////////////
always @(
          sx_pending_wr_sx_sel or
          sx_pending_wr_sp_id or
          sx_pending_wr_aux_sel or
          sx_pending_wr_aux_inc or
          sx_pending_wr_pci or
```

```
          sx_pending_wr_req_mask or
          sx_pending_wr_state_var_indx
         ) begin : proc200
  sx_pending_fifo_wrdata = {
                           sx_pending_wr_sx_sel,
                           sx_pending_wr_sp_id,
                           sx_pending_wr_aux_sel,
                           sx_pending_wr_aux_inc,
                           sx_pending_wr_pci,
                           sx_pending_wr_req_mask,
                           sx_pending_wr_state_var_indx
                          };
end


////////////////////////////////////////////////////////////////////
// sx_pending fifo
////////////////////////////////////////////////////////////////////
always @(posedge clk) begin : proc300
  if (reset == 'h1) begin
    sx_pending_fifo_wraddr <= 'h0;
    sx_pending_fifo_rdaddr <= 'h0;
    sx_pending_fifo_contents <= 'h0;
    end
  else begin
```

```
    var300_sx_pending_fifo_contents = sx_pending_fifo_contents;

    if (sx_pending_fifo_write == 'h1) begin
      sx_pending_fifo[sx_pending_fifo_wraddr] <= sx_pending_fifo_wrdata;
      sx_pending_fifo_wraddr <= sx_pending_fifo_wraddr + 'h1;
      var300_sx_pending_fifo_contents = var300_sx_pending_fifo_contents + 'h1;
    end

    if (sx_pending_fifo_advanceread == 'h1) begin
      sx_pending_fifo_rdaddr <= sx_pending_fifo_rdaddr + 'h1;
      var300_sx_pending_fifo_contents = var300_sx_pending_fifo_contents - 'h1;
    end

    if ((var300_sx_pending_fifo_contents == (SX_PENDING_FIFO_DEPTH-1)) ||
        (var300_sx_pending_fifo_contents == SX_PENDING_FIFO_DEPTH))     begin
      sx_pending_fifo_full <= 'h1;
      end
    else begin
      sx_pending_fifo_full <= 'h0;
    end

    if (var300_sx_pending_fifo_contents == 'h0) begin
      sx_pending_fifo_empty <= 'h1;
      end
    else begin
```

```
   sx_pending_fifo_empty <= 'h0;
  end

  sx_pending_fifo_contents <= var300_sx_pending_fifo_contents;
 end
end


////////////////////////////////////////////////////////////////////////
// sx_pending fifo read mux
////////////////////////////////////////////////////////////////////////
always @(
        sx_pending_fifo[0] or
        sx_pending_fifo[1] or
        sx_pending_fifo[2] or
        sx_pending_fifo[3] or
        sx_pending_fifo_rdaddr
       ) begin : proc340
 sx_pending_fifo_rddata = sx_pending_fifo[sx_pending_fifo_rdaddr];
end

////////////////////////////////////////////////////////////////////////
// sx_pending_wr_sx_sel
// sx_pending_wr_sp_id
// sx_pending_wr_aux_sel
```

```
// sx_pending_wr_aux_inc
// sx_pending_wr_pci
// sx_pending_wr_req_mask
// sx_pending_wr_state_var_indx
////////////////////////////////////////////////////////////////////////
always @(
        sx_pending_fifo_rddata
       ) begin : proc360
 {initial_sx_pending_rd_sx_sel,
  initial_sx_pending_rd_sp_id,
  initial_sx_pending_rd_aux_sel,
  initial_sx_pending_rd_aux_inc,
  initial_sx_pending_rd_pci,
  initial_sx_pending_rd_req_mask,
  initial_sx_pending_rd_state_var_indx} = sx_pending_fifo_rddata;
end

////////////////////////////////////////////////////////////////////////
// PaSxReceive
////////////////////////////////////////////////////////////////////////
always @(
        initial_sx_pending_rd_sx_sel or
        initial_sx_pending_rd_sp_id or
        initial_sx_pending_rd_aux_sel or
        initial_sx_pending_rd_aux_inc or
```

```
        initial_sx_pending_rd_pci or
        initial_sx_pending_rd_req_mask or
        initial_sx_pending_rd_state_var_indx or
        next_point_address or
        next_sx_pending_advance or
        next_sx_pending_rd_req_mask or
        next_sx_receive_indx or
        point_address or
        position_address or
        statevar_bits_use_vtx_point_size or
        statevar_bits_use_vtx_edge_flag or
        statevar_bits_use_vtx_kill_flag or
        sx_pending_advance or
        sx_pending_fifo_empty or
        sx_pending_rd_aux_inc or
        sx_pending_rd_aux_sel or
        sx_pending_rd_pci or
        sx_pending_rd_req_mask or
        sx_pending_rd_sp_id or
        sx_pending_rd_state_var_indx or
        sx_pending_rd_sx_sel or
        sx_receive_indx or
        sx_to_pa_empty or
        sx_to_pa_vector[0] or
        sx_to_pa_vector[1] or
```

```
        tcl_scratch_mem_position_data or
        available_positions or
        var400_first_vector_write or
        var400_last_vector_write or
        var400_sx_sel or
        var400_vector_wr_cnt or
        var400_vector_write or
        vertex_fifo_entriesavailable or
        vertex_fifo_full
       ) begin : proc400

  // init variables
  var400_sx_sel = 'h0;
  var400_vector_write = 'h0;
  var400_first_vector_write = 'h0;
  var400_last_vector_write = 'h0;
  var400_vector_wr_cnt = 'h0;
  var400_next_sx_pending_rd_sx_sel = sx_pending_rd_sx_sel;
  var400_next_sx_pending_rd_sp_id = sx_pending_rd_sp_id;
  var400_next_sx_pending_rd_aux_sel = sx_pending_rd_aux_sel;
  var400_next_sx_pending_rd_aux_inc = sx_pending_rd_aux_inc;
  var400_next_sx_pending_rd_pci = sx_pending_rd_pci;
  var400_next_sx_pending_rd_req_mask = sx_pending_rd_req_mask;
  var400_next_sx_pending_rd_state_var_indx = sx_pending_rd_state_var_indx;
  var400_ignore_this_cycle = 'h0;
  var400_next_sx_receive_indx = sx_receive_indx;

  // defaults
  next_sx_receive_indx = sx_receive_indx;
  next_sx_pending_advance = sx_pending_advance;
  sx_pending_fifo_advanceread = 'h0;
  decrement_pasx_req_cnt[0] = 'h0;
  decrement_pasx_req_cnt[1] = 'h0;
  tcl_scratch_mem_point_write = 'h0;
  tcl_scratch_mem_position_write = 'h0;
  next_position_address = position_address;
  next_point_address = point_address;
  decrement_available_positions = 'h0;
  vertex_fifo_write = 'h0;
  vertex_fifo_wr_param_cache_indx = 'h0;
  vertex_fifo_wr_state_var_indx = 'h0;
  vertex_fifo_wr_edge_flag = 'h0;
  vertex_fifo_wr_kill_flag = 'h0;
  sx_to_pa_advanceread[0] = 'h0;
  sx_to_pa_advanceread[1] = 'h0;
  //tcl_scratch_mem_position_data = sx_to_pa_vector[0];
  //tcl_scratch_mem_point_data = tcl_scratch_mem_position_data[VECTORX_HIGH:VECTORX_LOW];
  tcl_scratch_mem_position_data = 'h0;
  tcl_scratch_mem_point_data = 'h0;
  next_sx_pending_rd_sx_sel = sx_pending_rd_sx_sel;
```

```
  next_sx_pending_rd_sp_id = sx_pending_rd_sp_id;
  next_sx_pending_rd_aux_sel = sx_pending_rd_aux_sel;
  next_sx_pending_rd_aux_inc = sx_pending_rd_aux_inc;
  next_sx_pending_rd_pci = sx_pending_rd_pci;
  next_sx_pending_rd_req_mask = sx_pending_rd_req_mask;
  next_sx_pending_rd_state_var_indx = sx_pending_rd_state_var_indx;

  var400_ignore_this_cycle = 'h0;
  if (sx_receive_indx == VECTORS_PER_SX_REQUEST) begin
    if (sx_pending_fifo_empty == 'h0) begin
      var400_next_sx_pending_rd_sx_sel = initial_sx_pending_rd_sx_sel;
      var400_next_sx_pending_rd_sp_id = initial_sx_pending_rd_sp_id;
      var400_next_sx_pending_rd_aux_sel = initial_sx_pending_rd_aux_sel;
      var400_next_sx_pending_rd_aux_inc = initial_sx_pending_rd_aux_inc;
      var400_next_sx_pending_rd_pci = initial_sx_pending_rd_pci;
      var400_next_sx_pending_rd_req_mask = initial_sx_pending_rd_req_mask;
      var400_next_sx_pending_rd_state_var_indx = initial_sx_pending_rd_state_var_indx;

      var400_next_sx_receive_indx = 'h0;
      next_sx_pending_advance = 'h1;
      end
    else begin
      var400_ignore_this_cycle = 'h1;
    end
  end
```

```

  if (var400_ignore_this_cycle == 'h0) begin
    var400_sx_sel = var400_next_sx_pending_rd_sx_sel;

    var400_vector_write = var400_next_sx_pending_rd_req_mask[var400_next_sx_receive_indx[1:0]];

    var400_first_vector_write = var400_vector_write & ~var400_next_sx_pending_rd_aux_sel;
    var400_last_vector_write = var400_vector_write &
                           (var400_next_sx_pending_rd_aux_sel | var400_next_sx_pending_rd_aux_inc);

    if (next_sx_pending_advance == 'h1) begin
      var400_vector_wr_cnt = 'h0;
      for (indx400 = 0; indx400 < VECTORS_PER_SX_REQUEST; indx400=indx400+1) begin
        if (var400_next_sx_pending_rd_req_mask[indx400] == 'h1) begin
          var400_vector_wr_cnt = var400_vector_wr_cnt + 'h1;
        end
      end

      if ((sx_to_pa_empty[var400_sx_sel] == 'h0) &&
          ((var400_first_vector_write == 'h0) || (var400_vector_wr_cnt <= available_positions)) &&
          ((var400_last_vector_write == 'h0) || (var400_vector_wr_cnt <= vertex_fifo_entriesavailable))) begin
        sx_pending_fifo_advanceread = 'h1;
        decrement_pasx_req_cnt[var400_sx_sel] = 'h1;
```

```
          next_sx_pending_advance = 'h0;
        end
      end

      if ((sx_to_pa_empty[var400_sx_sel] == 'h1) ||
         ((var400_first_vector_write == 'h1) && (available_positions == 'h0)) ||
         ((var400_last_vector_write == 'h1) && (vertex_fifo_full == 'h1))) begin
        // do nothing this clock
        end
      else begin
        sx_to_pa_advanceread[var400_sx_sel] = 'h1;

        if (var400_vector_write == 'h1) begin


          vertex_fifo_wr_edge_flag = 'h0;
          vertex_fifo_wr_kill_flag = 'h0;

          if (var400_next_sx_pending_rd_aux_sel == 'h1) begin
            if (statevar_bits_use_vtx_point_size == 'h1) begin
              tcl_scratch_mem_point_write = 'h1;
              tcl_scratch_mem_position_data = sx_to_pa_vector[var400_sx_sel];
              tcl_scratch_mem_point_data = tcl_scratch_mem_position_data[VECTORX_HIGH:VECTORX_LOW];
            end

            var400_sx_to_pa_vector_var400_sx_sel = sx_to_pa_vector[var400_sx_sel];
```

```
     var400_sx_to_pa_vector_y_var400_sx_sel = var400_sx_to_pa_vector_var400_sx_sel[VECTORY_HIGH:VECTORY_LOW];
     var400_sx_to_pa_vector_z_var400_sx_sel = var400_sx_to_pa_vector_var400_sx_sel[VECTORZ_HIGH:VECTORZ_LOW];

     if (statevar_bits_use_vtx_edge_flag == 'h1) begin
      vertex_fifo_wr_edge_flag = var400_sx_to_pa_vector_y_var400_sx_sel[0];
     end

     vertex_fifo_wr_kill_flag = 'h0;
     if (statevar_bits_use_vtx_kill_flag == 'h1) begin
      if (var400_sx_to_pa_vector_z_var400_sx_sel[30:0] != 'h0) begin
       vertex_fifo_wr_kill_flag = 'h1;
      end
     end

     end
    else begin
     tcl_scratch_mem_position_write = 'h1;
     next_position_address = next_position_address + 'h1;
     tcl_scratch_mem_position_data = sx_to_pa_vector[var400_sx_sel];
     decrement_available_positions = 'h1;
    end

    if ((var400_next_sx_pending_rd_aux_sel == 'h1) ||
      (var400_next_sx_pending_rd_aux_inc == 'h1)) begin
```

```
     next_point_address = next_point_address + 'h1;

     vertex_fifo_wr_param_cache_indx = {var400_next_sx_pending_rd_sp_id,
                      var400_sx_sel,
                      var400_next_sx_receive_indx[1:0],
                      var400_next_sx_pending_rd_pci};
     vertex_fifo_wr_state_var_indx = var400_next_sx_pending_rd_state_var_indx;
     vertex_fifo_write = 'h1;
    end

    var400_next_sx_pending_rd_req_mask[var400_next_sx_receive_indx[1:0]] = 'h0;
   end

   var400_next_sx_receive_indx = var400_next_sx_receive_indx + 'h1;
  end
 end

 next_sx_pending_rd_sx_sel = var400_next_sx_pending_rd_sx_sel;
 next_sx_pending_rd_sp_id = var400_next_sx_pending_rd_sp_id;
 next_sx_pending_rd_aux_sel = var400_next_sx_pending_rd_aux_sel;
 next_sx_pending_rd_aux_inc = var400_next_sx_pending_rd_aux_inc;
 next_sx_pending_rd_pci = var400_next_sx_pending_rd_pci;
 next_sx_pending_rd_req_mask = var400_next_sx_pending_rd_req_mask;
 next_sx_pending_rd_state_var_indx = var400_next_sx_pending_rd_state_var_indx;
 next_sx_receive_indx = var400_next_sx_receive_indx;
```

```
end


//////////////////////////////////////////////////////////////////////
// sx_pending_rd
//////////////////////////////////////////////////////////////////////
always @(posedge clk) begin : proc475
 if (reset == 'h1) begin
  sx_pending_rd_sx_sel = 'h0;
  sx_pending_rd_sp_id = 'h0;
  sx_pending_rd_aux_sel = 'h0;
  sx_pending_rd_aux_inc = 'h0;
  sx_pending_rd_pci = 'h0;
  sx_pending_rd_req_mask = 'h0;
  sx_pending_rd_state_var_indx = 'h0;
  end
 else begin
  sx_pending_rd_sx_sel = next_sx_pending_rd_sx_sel;
  sx_pending_rd_sp_id = next_sx_pending_rd_sp_id;
  sx_pending_rd_aux_sel = next_sx_pending_rd_aux_sel;
  sx_pending_rd_aux_inc = next_sx_pending_rd_aux_inc;
  sx_pending_rd_pci = next_sx_pending_rd_pci;
  sx_pending_rd_req_mask = next_sx_pending_rd_req_mask;
  sx_pending_rd_state_var_indx = next_sx_pending_rd_state_var_indx;
 end
```

```
end

//////////////////////////////////////////////////////////////////////
// pa_to_sx output register
//////////////////////////////////////////////////////////////////////
always @(posedge clk) begin : proc477
 if (reset == 'h1) begin
  for (indx477 = 0; indx477 < PASX_INTERFACES; indx477 = indx477 + 1) begin
   pa_to_sx_write[indx477] <= 'h0;
   pa_to_sx_req[indx477] <= 'h0;
   pa_to_sx_sp_id[indx477] <= 'h0;
   pa_to_sx_offset[indx477] <= 'h0;
   pa_to_sx_aux[indx477] <= 'h0;
   pa_to_sx_last[indx477] <= 'h0;
  end
  end
 else begin
  for (indx477 = 0; indx477 < PASX_INTERFACES; indx477 = indx477 + 1) begin
   pa_to_sx_write[indx477] <= next_pa_to_sx_write[indx477];
   pa_to_sx_req[indx477] <= next_pa_to_sx_req[indx477];
   pa_to_sx_sp_id[indx477] <= next_pa_to_sx_sp_id[indx477];
   pa_to_sx_offset[indx477] <= next_pa_to_sx_offset[indx477];
   pa_to_sx_aux[indx477] <= next_pa_to_sx_aux[indx477];
   pa_to_sx_last[indx477] <= next_pa_to_sx_last[indx477];
  end
  end
end

//////////////////////////////////////////////////////////////////////
// pasx_req_cnt
//////////////////////////////////////////////////////////////////////
always @(posedge clk) begin : proc480
  if (reset == 'h1) begin
    pasx_req_cnt[0] <= 'h0;
    pasx_req_cnt[1] <= 'h0;
    end
  else begin
    var480_next_pasx_req_cnt_0 = pasx_req_cnt[0];
    var480_next_pasx_req_cnt_1 = pasx_req_cnt[1];

    if (increment_pasx_req_cnt[0] == 'h1) begin
      var480_next_pasx_req_cnt_0 = var480_next_pasx_req_cnt_0 + 'h1;
    end

    if (decrement_pasx_req_cnt[0] == 'h1) begin
      var480_next_pasx_req_cnt_0 = var480_next_pasx_req_cnt_0 - 'h1;
    end

    if (increment_pasx_req_cnt[1] == 'h1) begin
      var480_next_pasx_req_cnt_1 = var480_next_pasx_req_cnt_1 + 'h1;
```

```
    end

    if (decrement_pasx_req_cnt[1] == 'h1) begin
      var480_next_pasx_req_cnt_1 = var480_next_pasx_req_cnt_1 - 'h1;
    end

    pasx_req_cnt[0] <= var480_next_pasx_req_cnt_0;
    pasx_req_cnt[1] <= var480_next_pasx_req_cnt_1;
  end
end

//////////////////////////////////////////////////////////////////////
// registers
//////////////////////////////////////////////////////////////////////
always @(posedge clk) begin : proc500
  if (reset == 'h1) begin
    sx_request_indx <= 'h0;
    aux_sel <= 'h0;
    param_cache_base <= 'h0;
    sx_aux <= 'h0;
    sx_sent[0] <= 'h0;
    sx_sent[1] <= 'h0;

    sx_receive_indx <= VECTORS_PER_SX_REQUEST;
    sx_pending_advance <= 'h0;
```

```
    position_address <= 'h0;
    point_address <= 'h0;
    end
  else begin
    sx_request_indx <= next_sx_request_indx;
    aux_sel <= next_aux_sel;
    param_cache_base <= next_param_cache_base;
    sx_aux <= next_sx_aux;
    sx_sent[0] <= next_sx_sent[0];
    sx_sent[1] <= next_sx_sent[1];

    sx_receive_indx <= next_sx_receive_indx;
    sx_pending_advance <= next_sx_pending_advance;
    position_address <= next_position_address;
    point_address <= next_point_address;
  end
end

//////////////////////////////////////////////////////////////////////
// copy part of the control to the vertex fifo to get
// the number of entries available
//////////////////////////////////////////////////////////////////////
always @(posedge clk) begin : proc600
  if (reset == 'h1) begin
    vertex_fifo_entriesavailable <= VERTEX_FIFO_DEPTH + 1;
```

```
    end
  else begin
    var600_vertex_fifo_entriesavailable = vertex_fifo_entriesavailable;

    if (vertex_fifo_write == 'h1) begin
      var600_vertex_fifo_entriesavailable = var600_vertex_fifo_entriesavailable - 'h1;
    end

    if (vertex_fifo_advanceread == 'h1) begin
      var600_vertex_fifo_entriesavailable = var600_vertex_fifo_entriesavailable + 'h1;
    end

    vertex_fifo_entriesavailable <= var600_vertex_fifo_entriesavailable;
  end
end

endmodule

```

```
`include "header.v"
`include "sc_header.v"
//                              -*- Mode: Verilog -*-
// Filename        : sc.v
// Description     : SC top block
// Author          : Mike  Mantor
// Created On      : Sunday March 17 2002
// Last Modified By: .
// Last Modified On: .
// Update Count    : 0
// Status          :  Initial

//------------------------------------------------------------------------
//
// $Id: //depot/r400/devel/parts_lib/src/gfx/sc/sc.v#50 $
//
// $Change: 43702 $
//
//
// Copyright:   Trade secret of ATI Technologies, Inc.
//                      © Copyright 2001-2002, ATI Technologies, Inc., (unpublished)
//
//                      All rights reserved.  This notice is intended as a precaution against
//                      inadvertent publication and does not imply publication or any waiver
//                      of confidentiality.  The year included in the foregoing notice is the
```

```
//                      year of creation of the work.
//
//------------------------------------------------------------------------
module sc (
   // -------------------------------------------
   // Chip Signals
   // -------------------------------------------
   sclk_global,
   srst,
   RBBM_SC_soft_reset,
   ROM_SP0_disable,
   ROM_SP1_disable,
   ROM_SP2_disable,
   ROM_SP3_disable,

   CG_SC_pm_enb,
   RBBM_regclk_active,         // Clock gating intiator

   // -------------------------------------------
   // Interface to the Global Register Bus (RBBM)
   // -------------------------------------------
   RBBM_a,                    // address
   RBBM_we,                   // write enable
   RBBM_wd,                   // write data
   RBBM_re,                   // read enable
```

```
   RBB_rs_in,                 // read strobe daisy chain in
   RBB_rs_out,                // read strobe daisy chain out
   RBB_rd_in,                 // read data daisy chain in
   RBB_rd_out,                // read data daisy chain out


   SC_RBBM_cntx0_busy,            //
   SC_RBBM_cntx17_busy,           //

   SC_a,                    // address
   SC_we,                   // write enable
   SC_wd,                   // write data
   SC_re,                   // read enable
   // -------------------------------------------
   // Interface to the CP
   // -------------------------------------------
   CP_SC_wc_inc,             //Increment write confirm counter
   SC_CP_wc_dec,             //Decrement write confirm counter
   SC_CP_vq_snd,             //VisQuerry send visibility flag
   SC_CP_vq_index,           //VisQuerry index to identify one of 64 VQ flags
   SC_CP_vq_discard,         //VisQuerry 0=>keep, 1=>Discard
   SC_CP_mp_snd,             //Sc is sending the multi-pass command now
   SC_CP_mp_loop,            //0=>Continue 1=>Loop
   // -------------------------------------------
   // Interface to the PA Setup Unit
```

```
   // -------------------------------------------
   PA_SC_p0,
   PA_SC_p1,
   PA_SC_p2,
   PA_SC_p3,
   PA_SC_p4,
   PA_SC_xy0,
   PA_SC_xy1,
   PA_SC_xy2,
   PA_SC_zminmax,
   PA_SC_cntl,
   PA_SC_phase,
   PA_SC_v0_indx,
   PA_SC_valid,
   SC_PA_earlyfrz,
   // -------------------------------------------
   // Interface to the render central coarse tile
   // -------------------------------------------
   SC_RC_coarse_event,  //need to determine what field will carry id when its an event
   SC_RC_coarse_tilex,
   SC_RC_coarse_tiley,
   SC_RC_coarse_minz,
   SC_RC_coarse_maxz,
   SC_RC_coarse_zplane,
   SC_RC_coarse_mask,
```

ATI 2118
LG v. ATI
IPR2015-00326

```
SC_RC_coarse_back,
SC_RC_coarse_state,
SC_RC_coarse_send,
RC_SC_coarse_rtr,
SC_RC_coarse_covered,
// -------------------------------------------
// Interface to the render central hier mask
// -------------------------------------------
RC_SC_hier_mask,
RC_SC_hier_rb_id,
RC_SC_hier_split,
RC_SC_hier_send,
SC_RC_hier_rtr,
// -------------------------------------------
// Interface to the render central detailed mask
// -------------------------------------------
SC_RC_detail_mask,
SC_RC_detail_send,
RC_SC_detail_rtr,
// -------------------------------------------
// Interface to the shader export block SX0
// -------------------------------------------
u0_SC_SX_quad_x,
u0_SC_SX_quad_y,
u0_SC_SX_quad_mask,
```

```
u0_SC_SX_quad_tilex,
u0_SC_SX_quad_tiley,
u0_SC_SX_quad_rb_id,
u0_SC_SX_quad_split,
u0_SC_SX_quad_send,
u0_SX_SC_quad_rtr,
// -------------------------------------------
// Interface to the shader export block SX1
// -------------------------------------------
u1_SC_SX_quad_x,
u1_SC_SX_quad_y,
u1_SC_SX_quad_mask,
u1_SC_SX_quad_tilex,
u1_SC_SX_quad_tiley,
u1_SC_SX_quad_rb_id,
u1_SC_SX_quad_split,
u1_SC_SX_quad_send,
u1_SX_SC_quad_rtr,
// -------------------------------------------
// Interface to the shader sequencer
// -------------------------------------------
SC_SQ_data,        //
SC_SQ_valid,
SQ_SC_free_buff,    //
SQ_SC_dec_cntr_cnt,   //
```

```

// -------------------------------------------
// Interface to shader pipes
// -------------------------------------------
u0_SC_SP_data,
u0_SC_SP_type,
u0_SC_SP_last_quad,
u0_SC_SP_valid,

u1_SC_SP_data,
u1_SC_SP_type,
u1_SC_SP_last_quad,
u1_SC_SP_valid,

u2_SC_SP_data,
u2_SC_SP_type,
u2_SC_SP_last_quad,
u2_SC_SP_valid,

u3_SC_SP_data,
u3_SC_SP_type,
u3_SC_SP_last_quad,
u3_SC_SP_valid

);
```

```

// ***********************************
// I/O Definitions
// ***********************************
// Chip Signals
input         sclk_global;
input         srst;
input         RBBM_SC_soft_reset;
input         ROM_SP0_disable;
input         ROM_SP1_disable;
input         ROM_SP2_disable;
input         ROM_SP3_disable;

input         CG_SC_pm_enb;
input         RBBM_regclk_active;       // Clock gating intiator

// Interface to the Register Bus (RBBM)
input   [16:2] RBBM_a;
input         RBBM_we;
input   [31:0] RBBM_wd;
input         RBBM_re;
input         RBB_rs_in;
output         RBB_rs_out;
input   [31:0] RBB_rd_in;
output  [31:0] RBB_rd_out;

output        SC_RBBM_cntx0_busy;          //
output        SC_RBBM_cntx17_busy;         //

// Registered out for possible daisy chaining of Rbbm bus
output  [16:2] SC_a;
output        SC_we;
output  [31:0] SC_wd;
output        SC_re;
// --------------------------------------------
// Interface to the CP
// --------------------------------------------
input         CP_SC_wc_inc;            //Increment write confirm counter
output        SC_CP_wc_dec;            //Decrement write confirm counter
output        SC_CP_vq_snd;            //VisQuerry send visibility flag
output   [5:0] SC_CP_vq_index;         //VisQuerry index to identify one of 64 VQ flags
output        SC_CP_vq_discard;        //VisQuerry 0=>keep, 1=>Discard
output        SC_CP_mp_snd;            //Sc is sending the multi-pass command now
output        SC_CP_mp_loop;           //0=>Continue 1=>Loop
// --------------------------------------------
// Interface to the PA Setup Unit
// --------------------------------------------
input  [17:0]  PA_SC_xy0;
input  [17:0]  PA_SC_xy1;
input  [17:0]  PA_SC_xy2;
```

```
input  [31:0]  PA_SC_p0;
input  [39:0]  PA_SC_p1;
input  [31:0]  PA_SC_p2;
input  [31:0]  PA_SC_p3;
input  [31:0]  PA_SC_p4;
input  [13:0]  PA_SC_zminmax;
input  [29:0]  PA_SC_cntl;
input   [1:0]  PA_SC_phase;
input   [1:0]  PA_SC_v0_indx;
input          PA_SC_valid;
output         SC_PA_earlyfrz;
// --------------------------------------------
// Interface to Render Central Coarse Tile
// --------------------------------------------
output        SC_RC_coarse_event;
output [9:0]  SC_RC_coarse_tilex;
output [9:0]  SC_RC_coarse_tiley;
output [13:0] SC_RC_coarse_minz;
output [13:0] SC_RC_coarse_maxz;
output [95:0] SC_RC_coarse_zplane;
output [15:0] SC_RC_coarse_mask;
output        SC_RC_coarse_back;
output [2:0]  SC_RC_coarse_state;
output        SC_RC_coarse_send;
output        SC_RC_coarse_covered;
```

```
input        RC_SC_coarse_rtr;
//--------------------------------------------
// Interface to Render Central Hier Kill
//--------------------------------------------
input  [15:0] RC_SC_hier_mask;
input   [1:0] RC_SC_hier_rb_id;
input        RC_SC_hier_split;
input        RC_SC_hier_send;
output        SC_RC_hier_rtr;
//--------------------------------------------
// Interface to Render Central Detail Quad Mask
//--------------------------------------------
output [15:0] SC_RC_detail_mask;
output        SC_RC_detail_send;
input        RC_SC_detail_rtr;
// --------------------------------------------
// Interface to the shader export block SX0
// --------------------------------------------
output [1:0]  u0_SC_SX_quad_x;
output [1:0]  u0_SC_SX_quad_y;
output [31:0] u0_SC_SX_quad_mask;
output [1:0]  u0_SC_SX_quad_tilex;
output        u0_SC_SX_quad_tiley;
output [1:0]  u0_SC_SX_quad_rb_id;
output        u0_SC_SX_quad_split;
```

```
output        u0_SC_SX_quad_send;
input        u0_SX_SC_quad_rtr;
// --------------------------------------------
// Interface to the shader export block SX1
// --------------------------------------------
output [1:0]  u1_SC_SX_quad_x;
output [1:0]  u1_SC_SX_quad_y;
output [31:0] u1_SC_SX_quad_mask;
output [1:0]  u1_SC_SX_quad_tilex;
output        u1_SC_SX_quad_tiley;
output [1:0]  u1_SC_SX_quad_rb_id;
output        u1_SC_SX_quad_split;
output        u1_SC_SX_quad_send;
input        u1_SX_SC_quad_rtr;
// --------------------------------------------
// Interface to the shader sequencer
// --------------------------------------------
input         SQ_SC_free_buff;   //used to synchronize SP and SQ interface
input         SQ_SC_dec_cntr_cnt;  //to prevent SQ request queue from overflowing
output  [`SC_SQ_DATA_WIDTH-1:0] SC_SQ_data;        //This data bus has a 1st clk and
2nd clock definition
output          SC_SQ_valid;        //Data at the output is valid and assumed sampled at next
clock
// --------------------------------------------
// Interface to the shader pipe SP0
// --------------------------------------------
```

```
output  [99:0] u0_SC_SP_data;
output  [1:0]  u0_SC_SP_type;
output         u0_SC_SP_last_quad;
output         u0_SC_SP_valid;
// -------------------------------------------
// Interface to the shader pipe SP1
// -------------------------------------------
output  [99:0] u1_SC_SP_data;
output  [1:0]  u1_SC_SP_type;
output         u1_SC_SP_last_quad;
output         u1_SC_SP_valid;
// -------------------------------------------
// Interface to the shader pipe SP2
// -------------------------------------------
output  [99:0] u2_SC_SP_data;
output  [1:0]  u2_SC_SP_type;
output         u2_SC_SP_last_quad;
output         u2_SC_SP_valid;
// -------------------------------------------
// Interface to the shader pipe SP3
// -------------------------------------------
output  [99:0] u3_SC_SP_data;
output  [1:0]  u3_SC_SP_type;
output         u3_SC_SP_last_quad;
output         u3_SC_SP_valid;
```

```


//**********************************************
// Internal Signal definitions
//**********************************************


wire           sclk;
wire           sclk_reg;
wire           sclk_sc;

wire           sc_srst;
wire           sc_hard_srst;
wire           sc_soft_srst;

wire           cg_blk_gated_clk_override;

wire           regclk_active;
wire           reg_clk_en;

wire           sc_clk_en;

wire    [16:2] reg_a;
wire           reg_we;
wire    [31:0] reg_wd;
```

```
wire           reg_re;

wire           sc_rs;
wire    [31:0] sc_rd;

//wire           su_earlyfrz;


wire           st_msaa_enable;
wire     [3:0] st_msaa_num_samples;
wire           st_scissor_en;
wire           st_draw_zero_length_line;
wire           st_window_offset_disable;
wire    [13:0] st_window_scissor_x_min;
wire    [13:0] st_window_scissor_x_max;
wire    [13:0] st_window_scissor_y_min;
wire    [13:0] st_window_scissor_y_max;
wire    [14:0] st_x_offset;
wire    [14:0] st_y_offset;
wire           st_bres_cntl_en;
wire     [7:0] st_bres_cntl_reg;

wire    [31:0] st_aa_mask;
wire           st_output_screen_xy;
wire           st_line_stipple_enable;
```

```
wire           st_jss_enable;
wire     [1:0] st_jss_x_dim;
wire     [1:0] st_jss_y_dim;
wire     [3:0] st_max_sample_dist;
wire     [3:0] st_jss_sample0_sel;
wire     [3:0] st_jss_sample1_sel;
wire     [3:0] st_jss_sample2_sel;
wire     [3:0] st_jss_sample3_sel;
wire     [3:0] st_jss_sample4_sel;
wire     [3:0] st_jss_sample5_sel;
wire     [3:0] st_jss_sample6_sel;
wire     [3:0] st_jss_sample7_sel;
wire     [3:0] st_jss_sample8_sel;
wire     [3:0] st_jss_sample9_sel;
wire     [3:0] st_jss_sample10_sel;
wire     [3:0] st_jss_sample11_sel;
wire     [3:0] st_jss_sample12_sel;
wire     [3:0] st_jss_sample13_sel;
wire     [3:0] st_jss_sample14_sel;
wire     [3:0] st_jss_sample15_sel;
wire     [3:0] st_msaa_urc_samp_offset_x;
wire     [3:0] st_msaa_urc_samp_offset_y;
wire     [3:0] st_msaa_llc_samp_offset_x;
wire     [3:0] st_msaa_llc_samp_offset_y;
wire     [3:0] st_msaa_lrc_samp_offset_x;
```

```
    wire     [3:0] st_msaa_lrc_samp_offset_y;
    wire     [3:0] st_sample_0_x;
    wire     [3:0] st_sample_0_y;
    wire     [3:0] st_sample_1_x;
    wire     [3:0] st_sample_1_y;
    wire     [3:0] st_sample_2_x;
    wire     [3:0] st_sample_2_y;
    wire     [3:0] st_sample_3_x;
    wire     [3:0] st_sample_3_y;
    wire     [3:0] st_sample_4_x;
    wire     [3:0] st_sample_4_y;
    wire     [3:0] st_sample_5_x;
    wire     [3:0] st_sample_5_y;
    wire     [3:0] st_sample_6_x;
    wire     [3:0] st_sample_6_y;
    wire     [3:0] st_sample_7_x;
    wire     [3:0] st_sample_7_y;
    wire     [3:0] st_sample_8_x;
    wire     [3:0] st_sample_8_y;
    wire     [3:0] st_sample_9_x;
    wire     [3:0] st_sample_9_y;
    wire     [3:0] st_sample_10_x;
    wire     [3:0] st_sample_10_y;
    wire     [3:0] st_sample_11_x;
    wire     [3:0] st_sample_11_y;
```

```
    wire     [3:0] st_sample_12_x;
    wire     [3:0] st_sample_12_y;
    wire     [3:0] st_sample_13_x;
    wire     [3:0] st_sample_13_y;
    wire     [3:0] st_sample_14_x;
    wire     [3:0] st_sample_14_y;
    wire     [3:0] st_sample_15_x;
    wire     [3:0] st_sample_15_y;
//    wire   [127:0] st_aa_offset_tbl;
    wire    [15:0] st_line_pattern;
    wire     [7:0] st_repeat_count;
    wire     [3:0] st_pattern_start;
    wire           st_pattern_bit_order;
    wire           st_auto_reset_enable;
    wire     [3:0] st_current_ptr;
    wire     [7:0] st_current_count;
    wire           st_cliprect_enable;
    wire    [13:0] st_cliprect_0_x_min;
    wire    [13:0] st_cliprect_0_y_min;
    wire    [13:0] st_cliprect_0_x_max;
    wire    [13:0] st_cliprect_0_y_max;
    wire    [13:0] st_cliprect_1_x_min;
    wire    [13:0] st_cliprect_1_y_min;
    wire    [13:0] st_cliprect_1_x_max;
    wire    [13:0] st_cliprect_1_y_max;
```

```
    wire    [13:0] st_cliprect_2_x_min;
    wire    [13:0] st_cliprect_2_y_min;
    wire    [13:0] st_cliprect_2_x_max;
    wire    [13:0] st_cliprect_2_y_max;
    wire    [13:0] st_cliprect_3_x_min;
    wire    [13:0] st_cliprect_3_y_min;
    wire    [13:0] st_cliprect_3_x_max;
    wire    [13:0] st_cliprect_3_y_max;
    wire    [15:0] st_clip_rule;
    wire           st_poly_offset_front_enable;
    wire           st_poly_offset_back_enable;
    wire           st_poly_offset_para_enable;
    wire    [31:0] st_poly_offset_front_offset;
    wire    [31:0] st_poly_offset_back_offset;
    wire    [31:0] st_poly_offset_front_scale;
    wire    [31:0] st_poly_offset_back_scale;

    wire           st_iter_msaa_enable;
    wire    [3:0]  st_iter_msaa_num_samples;
    wire           st_iter_jss_enable;

    wire   [270:0] pa_sc_inputs;
    wire   [270:0] pa_sc_inputs_reg;

    wire    [29:0] cntl_in;
```

```
    wire    [13:0] zminmax_in;
    wire    [31:0] p0_in;
    wire    [39:0] p1_in;
    wire    [31:0] p2_in;
    wire    [31:0] p3_in;
    wire    [31:0] p4_in;
    wire           valid_in;
    wire     [1:0] v0_indx_in;
    wire     [1:0] phase_in;
    wire           event_in;
    wire     [3:0] event_id_in;
    wire    [17:0] xy0_in;
    wire    [17:0] xy1_in;
    wire    [17:0] xy2_in;

    wire    [31:0] RBIU_wdata; //   Register Write Data
    wire           RBIU_we;
    wire           RBIU_re;
    wire     [2:0] RBIU_waddr;
    wire     [2:0] RBIU_raddr;
    wire           RBIU_cpy;
    wire           RBIU_PIPE_susc_cntl_sel;
    wire           RBIU_PIPE_sq_context_misc_sel;
    wire           RBIU_PIPE_window_offset_sel;
    wire           RBIU_PIPE_aa_config_sel;
```

```
wire         RBIU_PIPE_aa_mask_sel;
wire         RBIU_PIPE_jss_sample_sel_0_sel;
wire         RBIU_PIPE_jss_sample_sel_1_sel;
//   wire         RBIU_PIPE_msaa_2x2_offset_sel;
//   wire         RBIU_PIPE_aa_offset_tbl_0_sel;
//   wire         RBIU_PIPE_aa_offset_tbl_1_sel;
//   wire         RBIU_PIPE_aa_offset_tbl_2_sel;
//   wire         RBIU_PIPE_aa_offset_tbl_3_sel;
wire         RBIU_PIPE_line_stipple_sel;
wire         RBIU_PIPE_line_stipple_state_sel;
wire         RBIU_PIPE_line_cntl_sel;
wire         RBIU_PIPE_window_scissor_tl_sel;
wire         RBIU_PIPE_window_scissor_br_sel;
wire         RBIU_PIPE_screen_scissor_tl_sel;
wire         RBIU_PIPE_screen_scissor_br_sel;
wire         RBIU_PIPE_cliprect_0_tl_sel;
wire         RBIU_PIPE_cliprect_0_br_sel;
wire         RBIU_PIPE_cliprect_1_tl_sel;
wire         RBIU_PIPE_cliprect_1_br_sel;
wire         RBIU_PIPE_cliprect_2_tl_sel;
wire         RBIU_PIPE_cliprect_2_br_sel;
wire         RBIU_PIPE_cliprect_3_tl_sel;
wire         RBIU_PIPE_cliprect_3_br_sel;
wire         RBIU_PIPE_cliprect_rule_sel;
wire         RBIU_PIPE_poly_offset_front_scale_sel;
```

```
wire         RBIU_PIPE_poly_offset_front_offset_sel;
wire         RBIU_PIPE_poly_offset_back_scale_sel;
wire         RBIU_PIPE_poly_offset_back_offset_sel;
wire   [31:0] PIPE_RBIU_rdata;

wire         sr_prim_we;
wire         sr_z_we;
wire         sr_pipe_valid;
wire         sr_event;
wire   [3:0] sr_event_id;
wire         sr_null_prim;
wire   [2:0] sr_dealloc_slot;
wire         sr_first_prim_of_slot;
wire         sr_end_of_pkt;
wire         sr_back_face;
wire   [1:0] sr_provoking_vertex;
wire         sr_x_major;
wire         sr_start_in_diamond;
wire         sr_end_in_diamond;
wire   [2:0] sr_prim_type;
wire   [1:0] sr_phase;
wire   [2:0] sr_state_var_indx;
wire   [2:0] qpp_state_var_indx;
wire   [17:0] sr_v0;
wire   [17:0] sr_v1;
```

```
wire   [17:0] sr_v2;
wire   [17:0] sr_ref_x;
wire   [17:0] sr_ref_y;
wire   [31:0] sr_i0;
wire   [31:0] sr_ix;
wire   [31:0] sr_iy;
wire   [31:0] sr_j0;
wire   [31:0] sr_jx;
wire   [31:0] sr_jy;
wire   [31:0] sr_w0;
wire   [31:0] sr_wx;
wire   [31:0] sr_wy;
wire   [10:0] sr_param_cache_indx0;
wire   [10:0] sr_param_cache_indx1;
wire   [10:0] sr_param_cache_indx2;
wire          sr_null_prim_zff;
wire          sr_zy_max_zff;
wire          sr_back_face_zff;
wire    [2:0] sr_prim_type_zff;
wire          sr_polymode_zff;
wire    [2:0] sr_state_var_indx_zff;
wire   [17:0] sr_ref_x_zff;
wire   [17:0] sr_ref_y_zff;
wire   [31:0] sr_z0_zff;
wire   [39:0] sr_zx_zff;
```

```
wire   [39:0] sr_zy_zff;
wire   [13:0] sr_z_min_zff;
wire   [13:0] sr_z_max_zff;
wire         sr_cntx0_busy;
wire         sr_cntx1to7_busy;

wire         prim_ff_re;
wire         prim_ff_full;
wire         prim_ff_empty;
wire   [`SC_PRIM_INTERP_WIDTH -1:0] prim_ff_wr_data;
wire   [`SC_PRIM_INTERP_WIDTH -1:0] prim_ff_rd_data;
wire         z_ff_re;
wire         z_ff_full;
wire         z_ff_empty;
wire   [`SC_ZDATA_WIDTH -1:0] z_ff_wr_data;
wire   [`SC_ZDATA_WIDTH -1:0] z_ff_rd_data;
wire         tile_ff_re;
wire         tile_ff_full;
wire         tile_ff_empty;
wire   [`SC_TILEDATA_WIDTH -1:0] tile_ff_wr_data;
//   wire   [171:0] tile_ff_rd_data;

wire         pipe_rts;
wire   [1:0] pipe_phase;
wire         event_flag;
wire    [3:0] event_id;
wire   [36:0] e0;
wire   [31:0] e0_y;
wire   [18:0] e0_dx;
wire   [18:0] e0_dy;
wire   [36:0] e1;
wire   [31:0] e1_x;
wire   [18:0] e1_dx;
wire   [18:0] e1_dy;
wire   [36:0] e2;
wire   [31:0] e2_x;
wire   [18:0] e2_dx;
wire   [18:0] e2_dy;
wire          x_dir;
wire   [12:0] x_start;
wire   [12:0] x_end;
wire          y_dir;
wire   [12:0] y_start;
wire   [12:0] y_end;
wire    [3:0] bb_fract_bits;
wire          pass_empty_prim;
wire          cw_event;
wire    [3:0] cw_event_id;
wire    [3:0] cw_bb_fract_bits;
wire          cw_pass_empty_prim;
```

```
wire   [36:0] cw_e0;
wire   [31:0] cw_e0y;
wire   [18:0] cw_dxe0;
wire   [18:0] cw_dye0;
wire   [36:0] cw_e1;
wire   [31:0] cw_e1x;
wire   [18:0] cw_dxe1;
wire   [18:0] cw_dye1;
wire   [36:0] cw_e2;
wire   [31:0] cw_e2x;
wire   [18:0] cw_dxe2;
wire   [18:0] cw_dye2;
wire          cw_xdir;
wire    [9:0] cw_tilex;
wire    [2:0] cw_xmin;
wire    [2:0] cw_xmax;
wire          cw_ydir;
wire          cw_xmajor ;
wire    [9:0] cw_tiley;
wire    [2:0] cw_ymin;
wire    [2:0] cw_ymax;
wire          cw_last_tile;
wire          cw_tile_valid;
wire          pipe_freeze_b_early;
wire          pipe_freeze_b_dly;
```

```
wire          pipe_freeze_b_dly1;
wire          qmsk_z_freeze_b;
wire          qm_last_tile;
wire          qm_event;
wire    [3:0] qm_event_id;
wire          qm_xdir;
wire          qm_ydir;
wire    [9:0] qm_tilex;
wire    [9:0] qm_tiley;
wire    [1:0] qm_tilex_m3;
wire    [1:0] qm_tiley_m3;
wire    [2:0] qm_xmin;
wire    [2:0] qm_xmax;
wire    [2:0] qm_ymin;
wire    [2:0] qm_ymax;
wire    [3:0] qm_bb_fract_bits;
wire          qm_z_mask_needed;
wire   [36:0] qm_e0;
wire   [36:0] qm_e1;
wire   [36:0] qm_e2;
wire   [18:0] qm_dxe0;
wire   [18:0] qm_dye0;
wire   [18:0] qm_dxe1;
wire   [18:0] qm_dye1;
wire   [18:0] qm_dxe2;
```

```
wire   [18:0] qm_dye2;
wire   [15:0] qm_quadmask;
wire          qm_quadmask_valid;

wire [`SC_PRIM_INTERP_WIDTH -1:0] qpp_prim_data;
wire          qpp_fpos_early;
wire          qpp_last_qdpair_of_prim;
wire          qpp_q0_last_of_tile;
wire          qpp_q0_zmask_needed;
wire          qpp_q0_qhit;
wire    [9:0] qpp_q0_tilex;
wire    [9:0] qpp_q0_tiley;
wire    [1:0] qpp_q0_quadx;
wire    [1:0] qpp_q0_quady;
wire    [1:0] qpp_q0_rb_id;
wire          qpp_q0_split;
wire    [7:0] qpp_q0_ulc_sample_mask;
wire    [7:0] qpp_q0_urc_sample_mask;
wire    [7:0] qpp_q0_llc_sample_mask;
wire    [7:0] qpp_q0_lrc_sample_mask;
wire    [2:0] qpp_q0_ulc_cntrmost_sample_id;
wire    [2:0] qpp_q0_urc_cntrmost_sample_id;
wire    [2:0] qpp_q0_llc_cntrmost_sample_id;
wire    [2:0] qpp_q0_lrc_cntrmost_sample_id;
wire          qpp_q1_last_of_tile;
wire         qpp_q1_zmask_needed;
wire         qpp_q1_qhit;
wire   [9:0] qpp_q1_tilex;
wire   [9:0] qpp_q1_tiley;
wire   [1:0] qpp_q1_quadx;
wire   [1:0] qpp_q1_rb_id;
wire         qpp_q1_split;
wire   [1:0] qpp_q1_quady;
wire   [7:0] qpp_q1_ulc_sample_mask;
wire   [7:0] qpp_q1_urc_sample_mask;
wire   [7:0] qpp_q1_llc_sample_mask;
wire   [7:0] qpp_q1_lrc_sample_mask;
wire   [2:0] qpp_q1_ulc_cntrmost_sample_id;
wire   [2:0] qpp_q1_urc_cntrmost_sample_id;
wire   [2:0] qpp_q1_llc_cntrmost_sample_id;
wire   [2:0] qpp_q1_lrc_cntrmost_sample_id;

wire         zff_null_prim;
wire         zff_zy_max;
wire         zff_back_face;
wire   [2:0] zff_prim_type;
wire         zff_polymode;
wire   [2:0] zff_state_var_indx;
wire  [17:0] zff_ref_x;
wire  [17:0] zff_ref_y;
```

```
wire   [13:0] zff_z_min;
wire   [13:0] zff_z_max;
wire   [31:0] zff_z0;
wire   [39:0] zff_zx;
wire   [39:0] zff_zy;

wire          rc_event;
wire    [9:0] rc_tilex;
wire    [9:0] rc_tiley;
wire   [13:0] rc_minz;
wire   [13:0] rc_maxz;
wire   [95:0] rc_zplane;
wire   [15:0] rc_mask;
wire          rc_back;
wire    [2:0] rc_state;
wire          rc_send;
wire          rc_covered;
wire          rc_rtr;
wire          rc_hier_rtr;
wire   [15:0] rc_in_hier_mask;
wire    [1:0] rc_in_rb_id;
wire          rc_in_split;
wire          rc_in_hier_send;

wire          qdpkr_in_fz;
```

```

//Signals for the sc_rc detailed z interface
wire         detail_mask_accum_rdy;
wire         detail_hit_0;
wire         detail_lqt_0;
wire         detail_hit_1;
wire         detail_lqt_1;
wire  [15:0] detail_mask;
wire         detail_mask_valid;
wire         rdy_for_detail_mask;

//Signals between the packer and iterator
wire [`SC_QD_DATA_WIDTH-1:0] pkr_qd0;
wire [`SC_QD_DATA_WIDTH-1:0] pkr_qd1;
wire [`SC_QD_DATA_WIDTH-1:0] pkr_qd2;
wire [`SC_QD_DATA_WIDTH-1:0] pkr_qd3;
wire         pkr_qdhit0;
wire         pkr_qdhit1;
wire         pkr_qdhit2;
wire         pkr_qdhit3;
wire [`SC_PRIM_INTERP_WIDTH -1:0] pkr_primdata;
wire         pkr_ds_one_clk_command;
wire         pkr_ds_end_of_prim;
wire         pkr_ds_end_of_vector;
wire         pkr_send_row;
```

```
wire [2:0]     pkr_sv_indx;

wire          pkr_cntx0_busy;
wire          pkr_cntx1to7_busy;
wire          pkr_iter_cntx0_busy;
wire          pkr_iter_cntx1to7_busy;

wire          iterator_input_fz;
wire    [2:0] iterator_sv_indx;
wire    [1:0] iterator_SX0_quad_x;
wire    [1:0] iterator_SX0_quad_y;
wire   [31:0] iterator_SX0_quad_mask;
wire    [1:0] iterator_SX0_quad_tilex;
wire          iterator_SX0_quad_tiley;
wire    [1:0] iterator_SX0_quad_rb_id;
wire          iterator_SX0_quad_split;
wire          iterator_SX0_quad_send;
wire          SX0_iterator_quad_rtr;
wire    [1:0] iterator_SX1_quad_x;
wire    [1:0] iterator_SX1_quad_y;
wire   [31:0] iterator_SX1_quad_mask;
wire    [1:0] iterator_SX1_quad_tilex;
wire          iterator_SX1_quad_tiley;
wire    [1:0] iterator_SX1_quad_rb_id;
wire          iterator_SX1_quad_split;
wire         iterator_SX1_quad_send;
wire         SX1_iterator_quad_rtr;
wire         SQ_iterator_free_buff;
wire         SQ_iterator_dec_cntr_cnt;
wire  [`SC_SQ_DATA_WIDTH-1:0] iterator_SQ_data;
wire         iterator_SQ_valid;
wire  [99:0] iterator_SP0_data;
wire   [1:0] iterator_SP0_type;
wire         iterator_SP0_last_quad;
wire         iterator_SP0_valid;
wire  [99:0] iterator_SP1_data;
wire   [1:0] iterator_SP1_type;
wire         iterator_SP1_last_quad;
wire         iterator_SP1_valid;
wire  [99:0] iterator_SP2_data;
wire   [1:0] iterator_SP2_type;
wire         iterator_SP2_last_quad;
wire         iterator_SP2_valid;
wire  [99:0] iterator_SP3_data;
wire   [1:0] iterator_SP3_type;
wire         iterator_SP3_last_quad;

wire         rt_set_cntx0_busy;
wire         cntx0_decr;
wire         cntx1to7_decr;
```

```





//create sclk
ati_master_clock_permanent uati_master_clock_permanent(
   .clk_in(sclk_global),
   .clk_out(sclk)
);

ati_dff_in #(1) uati_dff_in_pm_en(
   .clk(sclk),
   .d(CG_SC_pm_enb),
   .q(cg_blk_gated_clk_override)
);

//create clock enable signals based on active signals
ati_dff_in #(1) uati_dff_in_regclk_active(
   .clk(sclk),
   .d(RBBM_regclk_active),
   .q(regclk_active)
);

//This enable would be intiated by rbbm_regclk_active and held high in the
```

```
//block as long as necessary to ensure all data could be read
assign reg_clk_en = regclk_active | SC_RBBM_cntx0_busy | SC_RBBM_cntx17_busy;

//This active signal would be a collection of request from external blocks that require
//the block clocks to be enabled along with internal busy signals that require the clocks
//to stay on
assign sc_clk_en = !cg_blk_gated_clk_override | RBBM_regclk_active |
SC_RBBM_cntx0_busy | SC_RBBM_cntx17_busy;



//Generate the sclk_reg clock tree
ati_master_clock_gater uati_master_clock_gater_sclk_reg (
   .clk_in(sclk_global),
   .clk(sclk),
   .en(reg_clk_en),
   .pm_enb(cg_blk_gated_clk_override),
   .clk_out(sclk_reg)
);

//Generate sclk_sc clock tree
ati_master_clock_gater uati_master_clock_gater_sc_clk (
   .clk_in(sclk_global),
   .clk(sclk),
   .en(sc_clk_en),
   .pm_enb(cg_blk_gated_clk_override),
   .clk_out(sclk_sc)
```

```
);

//RBBM Interface register
ati_rbbm_intf uati_rbbm_intf(
   .sclk_reg(sclk_reg),
   .rbbm_we(RBBM_we),
   .rbbm_re(RBBM_re),
   .rbbm_a(RBBM_a),
   .rbbm_wd(RBBM_wd),
   .reg_we(reg_we),
   .reg_re(reg_re),
   .reg_a(reg_a),
   .reg_wd(reg_wd),
   .pipe_we(SC_we),
   .pipe_re(SC_re),
   .pipe_a(SC_a),
   .pipe_wd(SC_wd),
   .rbbm_rs_in(RBB_rs_in),
   .rbbm_rd_in(RBB_rd_in),
   .block_rs(sc_rs),
   .block_rd(sc_rd),
   .rbbm_rs_out(RBB_rs_out),
   .rbbm_rd_out(RBB_rd_out)
);

```

```

//register input reset
ati_dff_in #(1) uati_dff_in_sc_hard_srst(
  .clk(sclk),
  .d(srst),
  .q(sc_hard_srst)
);

//register input soft resets
ati_dff_in #(1) uati_dff_in_sc_soft_srst(
  .clk(sclk),
  .d(RBBM_SC_soft_reset),
  .q(sc_soft_srst)
);

//use this in the block for the srst everywhere except
//state storage that can only get reset by a hard reset
assign sc_srst = sc_soft_srst | sc_hard_srst;


//register block outputs using the ati_dff_out or ati_dff_en_out
ati_dff_out #(1) uati_dff_out_earlyfrz(
  .clk(sclk_sc),
  //.d(su_earlyfrz),
  .d(pipe_freeze_b_early),
```

```
    .q(SC_PA_earlyfrz)
  );





//--------------------------------------------------
// Register inputs
//--------------------------------------------------

  sc_interface_regs usc_interface_regs(
    .clk(sclk_sc),
    .en(pipe_freeze_b_dly),
    .pa_sc_inputs(pa_sc_inputs),
    .pa_sc_inputs_reg(pa_sc_inputs_reg),
    .RC_SC_coarse_rtr(RC_SC_coarse_rtr),
    .rc_rtr(rc_rtr),
    .RC_SC_hier_send(RC_SC_hier_send),
    .rc_in_hier_send(rc_in_hier_send),
    .RC_SC_hier_mask(RC_SC_hier_mask),
    .rc_in_hier_mask(rc_in_hier_mask),
    .RC_SC_hier_rb_id(RC_SC_hier_rb_id),
    .rc_in_rb_id(rc_in_rb_id),
    .RC_SC_hier_split(RC_SC_hier_split),
```

```
  .rc_in_split(rc_in_split),
  .u0_SX_SC_quad_rtr(u0_SX_SC_quad_rtr),
  .SX0_iterator_quad_rtr(SX0_iterator_quad_rtr),
  .u1_SX_SC_quad_rtr(u1_SX_SC_quad_rtr),
  .SX1_iterator_quad_rtr(SX1_iterator_quad_rtr),
  .SQ_SC_free_buff(SQ_SC_free_buff),
  .SQ_iterator_free_buff(SQ_iterator_free_buff),
  .SQ_SC_dec_cntr_cnt(SQ_SC_dec_cntr_cnt),
  .SQ_iterator_dec_cntr_cnt(SQ_iterator_dec_cntr_cnt),
  .rc_event(rc_event),
  .SC_RC_coarse_event(SC_RC_coarse_event),
  .rc_tilex(rc_tilex),
  .SC_RC_coarse_tilex(SC_RC_coarse_tilex),
  .rc_tiley(rc_tiley),
  .SC_RC_coarse_tiley(SC_RC_coarse_tiley),
  .rc_minz(rc_minz),
  .SC_RC_coarse_minz(SC_RC_coarse_minz),
  .rc_maxz(rc_maxz),
  .SC_RC_coarse_maxz(SC_RC_coarse_maxz),
  .rc_zplane(rc_zplane),
  .SC_RC_coarse_zplane(SC_RC_coarse_zplane),
  .rc_mask(rc_mask),
  .SC_RC_coarse_mask(SC_RC_coarse_mask),
  .rc_back(rc_back),
  .SC_RC_coarse_back(SC_RC_coarse_back),
```

```
    .rc_state(rc_state),
    .SC_RC_coarse_state(SC_RC_coarse_state),
    .rc_covered(rc_covered),
    .SC_RC_coarse_covered(SC_RC_coarse_covered),
    .rc_send(rc_send),
    .SC_RC_coarse_send(SC_RC_coarse_send),
    .rc_hier_rtr(rc_hier_rtr),
    .SC_RC_hier_rtr(SC_RC_hier_rtr),
    .detail_mask_valid(detail_mask_valid),
    .SC_RC_detail_send(SC_RC_detail_send),
    .detail_mask(detail_mask),
    .SC_RC_detail_mask(SC_RC_detail_mask),
    .iterator_SX0_quad_x(iterator_SX0_quad_x),
    .u0_SC_SX_quad_x(u0_SC_SX_quad_x),
    .iterator_SX0_quad_y(iterator_SX0_quad_y),
    .u0_SC_SX_quad_y(u0_SC_SX_quad_y),
    .iterator_SX0_quad_mask(iterator_SX0_quad_mask),
    .u0_SC_SX_quad_mask(u0_SC_SX_quad_mask),
    .iterator_SX0_quad_tilex(iterator_SX0_quad_tilex),
    .u0_SC_SX_quad_tilex(u0_SC_SX_quad_tilex),
    .iterator_SX0_quad_tiley(iterator_SX0_quad_tiley),
    .u0_SC_SX_quad_tiley(u0_SC_SX_quad_tiley),

    .iterator_SX0_quad_rb_id(iterator_SX0_quad_rb_id),
    .u0_SC_SX_quad_rb_id(u0_SC_SX_quad_rb_id),
        .iterator_SX0_quad_split(iterator_SX0_quad_split),
        .u0_SC_SX_quad_split(u0_SC_SX_quad_split),

        .iterator_SX0_quad_send(iterator_SX0_quad_send),
        .u0_SC_SX_quad_send(u0_SC_SX_quad_send),
        .iterator_SX1_quad_x(iterator_SX1_quad_x),
        .u1_SC_SX_quad_x(u1_SC_SX_quad_x),
        .iterator_SX1_quad_y(iterator_SX1_quad_y),
        .u1_SC_SX_quad_y(u1_SC_SX_quad_y),
        .iterator_SX1_quad_mask(iterator_SX1_quad_mask),
        .u1_SC_SX_quad_mask(u1_SC_SX_quad_mask),
        .iterator_SX1_quad_tilex(iterator_SX1_quad_tilex),
        .u1_SC_SX_quad_tilex(u1_SC_SX_quad_tilex),
        .iterator_SX1_quad_tiley(iterator_SX1_quad_tiley),
        .u1_SC_SX_quad_tiley(u1_SC_SX_quad_tiley),

        .iterator_SX1_quad_rb_id(iterator_SX1_quad_rb_id),
        .u1_SC_SX_quad_rb_id(u1_SC_SX_quad_rb_id),
        .iterator_SX1_quad_split(iterator_SX1_quad_split),
        .u1_SC_SX_quad_split(u1_SC_SX_quad_split),

        .iterator_SX1_quad_send(iterator_SX1_quad_send),
        .u1_SC_SX_quad_send(u1_SC_SX_quad_send),
        .iterator_SQ_data(iterator_SQ_data),
        .SC_SQ_data(SC_SQ_data),
```

```
        .iterator_SQ_valid(iterator_SQ_valid),
        .SC_SQ_valid(SC_SQ_valid),
        .iterator_SP0_data(iterator_SP0_data),
        .u0_SC_SP_data(u0_SC_SP_data),
        .iterator_SP0_type(iterator_SP0_type),
        .u0_SC_SP_type(u0_SC_SP_type),
        .iterator_SP0_last_quad(iterator_SP0_last_quad),
        .u0_SC_SP_last_quad(u0_SC_SP_last_quad),
        .iterator_SP0_valid(iterator_SP0_valid),
        .u0_SC_SP_valid(u0_SC_SP_valid),
        .iterator_SP1_data(iterator_SP1_data),
        .u1_SC_SP_data(u1_SC_SP_data),
        .iterator_SP1_type(iterator_SP1_type),
        .u1_SC_SP_type(u1_SC_SP_type),
        .iterator_SP1_last_quad(iterator_SP1_last_quad),
        .u1_SC_SP_last_quad(u1_SC_SP_last_quad),
        .iterator_SP1_valid(iterator_SP1_valid),
        .u1_SC_SP_valid(u1_SC_SP_valid),
        .iterator_SP2_data(iterator_SP2_data),
        .u2_SC_SP_data(u2_SC_SP_data),
        .iterator_SP2_type(iterator_SP2_type),
        .u2_SC_SP_type(u2_SC_SP_type),
        .iterator_SP2_last_quad(iterator_SP2_last_quad),
        .u2_SC_SP_last_quad(u2_SC_SP_last_quad),
        .iterator_SP2_valid(iterator_SP2_valid),
```

```
        .u2_SC_SP_valid(u2_SC_SP_valid),
        .iterator_SP3_data(iterator_SP3_data),
        .u3_SC_SP_data(u3_SC_SP_data),
        .iterator_SP3_type(iterator_SP3_type),
        .u3_SC_SP_type(u3_SC_SP_type),
        .iterator_SP3_last_quad(iterator_SP3_last_quad),
        .u3_SC_SP_last_quad(u3_SC_SP_last_quad),
        .iterator_SP3_valid(iterator_SP3_valid),
        .u3_SC_SP_valid(u3_SC_SP_valid),
        .sr_cntx0_busy(sr_cntx0_busy),
        .SC_RBBM_cntx0_busy(SC_RBBM_cntx0_busy),
        .sr_cntx1to7_busy(sr_cntx1to7_busy),
        .SC_RBBM_cntx17_busy(SC_RBBM_cntx17_busy),
        .RC_SC_detail_rtr(RC_SC_detail_rtr),
        .rdy_for_detail_mask(rdy_for_detail_mask)
    );

//--------------------------------------------------
// RBBM Interface
//--------------------------------------------------
    sc_rbiu usc_rbiu (
        // Chip Signals
        .iSCLK_REG(sclk_reg),
        // Interface to the Global Register Bus (RBBM)
        .iRBBM_a_q1(reg_a),                // address
```

```
        .iRBBM_we_q1(reg_we),              // write enable
        .iRBBM_wd_q1(reg_wd),              // write data
        .iRBBM_re_q1(reg_re),          // read enable
        // RBBM read data daisy chain
        .oRBIU_block_rs(sc_rs),            // read strobe daisy chain out
        .oRBIU_block_rd(sc_rd),            // read data daisy chain out
        // Interface to sc_pipe
        .oRBIU_we(RBIU_we),
        .oRBIU_re(RBIU_re),
        .oRBIU_waddr(RBIU_waddr),
        .oRBIU_raddr(RBIU_raddr),
        .oRBIU_wdata(RBIU_wdata),
        .iPIPE_RBIU_rdata(PIPE_RBIU_rdata),
        .oRBIU_cpy(RBIU_cpy),
        .oRBIU_PIPE_susc_cntl_sel(RBIU_PIPE_susc_cntl_sel),
        .oRBIU_PIPE_sq_context_misc(RBIU_PIPE_sq_context_misc_sel),
        .oRBIU_PIPE_window_offset_sel(RBIU_PIPE_window_offset_sel),
        .oRBIU_PIPE_aa_config_sel(RBIU_PIPE_aa_config_sel),
        .oRBIU_PIPE_aa_mask_sel(RBIU_PIPE_aa_mask_sel),
        .oRBIU_PIPE_jss_sample_sel_0_sel(RBIU_PIPE_jss_sample_sel_0_sel),
        .oRBIU_PIPE_jss_sample_sel_1_sel(RBIU_PIPE_jss_sample_sel_1_sel),
//      .oRBIU_PIPE_msaa_2x2_offset_sel(RBIU_PIPE_msaa_2x2_offset_sel),
//      .oRBIU_PIPE_aa_offset_tbl_0_sel(RBIU_PIPE_aa_offset_tbl_0_sel),
//      .oRBIU_PIPE_aa_offset_tbl_1_sel(RBIU_PIPE_aa_offset_tbl_1_sel),
//      .oRBIU_PIPE_aa_offset_tbl_2_sel(RBIU_PIPE_aa_offset_tbl_2_sel),
//        .oRBIU_PIPE_aa_offset_tbl_3_sel(RBIU_PIPE_aa_offset_tbl_3_sel),
        .oRBIU_PIPE_line_stipple_sel(RBIU_PIPE_line_stipple_sel),
        .oRBIU_PIPE_line_stipple_state_sel(RBIU_PIPE_line_stipple_state_sel),
        .oRBIU_PIPE_line_cntl_sel(RBIU_PIPE_line_cntl_sel),
        .oRBIU_PIPE_window_scissor_tl_sel(RBIU_PIPE_window_scissor_tl_sel),
        .oRBIU_PIPE_window_scissor_br_sel(RBIU_PIPE_window_scissor_br_sel),
        .oRBIU_PIPE_screen_scissor_tl_sel(RBIU_PIPE_screen_scissor_tl_sel),
        .oRBIU_PIPE_screen_scissor_br_sel(RBIU_PIPE_screen_scissor_br_sel),
        .oRBIU_PIPE_cliprect_0_tl_sel(RBIU_PIPE_cliprect_0_tl_sel),
        .oRBIU_PIPE_cliprect_0_br_sel(RBIU_PIPE_cliprect_0_br_sel),
        .oRBIU_PIPE_cliprect_1_tl_sel(RBIU_PIPE_cliprect_1_tl_sel),
        .oRBIU_PIPE_cliprect_1_br_sel(RBIU_PIPE_cliprect_1_br_sel),
        .oRBIU_PIPE_cliprect_2_tl_sel(RBIU_PIPE_cliprect_2_tl_sel),
        .oRBIU_PIPE_cliprect_2_br_sel(RBIU_PIPE_cliprect_2_br_sel),
        .oRBIU_PIPE_cliprect_3_tl_sel(RBIU_PIPE_cliprect_3_tl_sel),
        .oRBIU_PIPE_cliprect_3_br_sel(RBIU_PIPE_cliprect_3_br_sel),
        .oRBIU_PIPE_cliprect_rule_sel(RBIU_PIPE_cliprect_rule_sel),
        .oRBIU_PIPE_poly_offset_front_scale_sel(RBIU_PIPE_poly_offset_front_scale_sel),
        .oRBIU_PIPE_poly_offset_front_offset_sel(RBIU_PIPE_poly_offset_front_offset_sel),
        .oRBIU_PIPE_poly_offset_back_scale_sel(RBIU_PIPE_poly_offset_back_scale_sel),
        .oRBIU_PIPE_poly_offset_back_offset_sel(RBIU_PIPE_poly_offset_back_offset_sel),
        .oRBIU_PIPE_rt_set_cntx0_busy(rt_set_cntx0_busy)
    );

//---------------------------------------------------
```

```
// Instantiate blocks
//---------------------------------------------------

    sc_state usc_state (
        .iSRST(sc_srst),
        .iSCLK(sclk_sc),
        .iRBIU_wdata(RBIU_wdata),
        .iRBIU_we(RBIU_we),
        .iRBIU_re(RBIU_re),
        .iRBIU_waddr(RBIU_waddr),
        .iRBIU_raddr(RBIU_raddr),
        .iRBIU_cpy(RBIU_cpy),
        .iRBIU_PIPE_susc_cntl_sel(RBIU_PIPE_susc_cntl_sel),
        .iRBIU_PIPE_sq_context_misc_sel(RBIU_PIPE_sq_context_misc_sel),
        .iRBIU_PIPE_window_offset_sel(RBIU_PIPE_window_offset_sel),
        .iRBIU_PIPE_aa_config_sel(RBIU_PIPE_aa_config_sel),
        .iRBIU_PIPE_aa_mask_sel(RBIU_PIPE_aa_mask_sel),
        .iRBIU_PIPE_jss_sample_sel_0_sel(RBIU_PIPE_jss_sample_sel_0_sel),
        .iRBIU_PIPE_jss_sample_sel_1_sel(RBIU_PIPE_jss_sample_sel_1_sel),
//        .iRBIU_PIPE_msaa_2x2_offset_sel(RBIU_PIPE_msaa_2x2_offset_sel),
//        .iRBIU_PIPE_aa_offset_tbl_0_sel(RBIU_PIPE_aa_offset_tbl_0_sel),
//        .iRBIU_PIPE_aa_offset_tbl_1_sel(RBIU_PIPE_aa_offset_tbl_1_sel),
//        .iRBIU_PIPE_aa_offset_tbl_2_sel(RBIU_PIPE_aa_offset_tbl_2_sel),
//        .iRBIU_PIPE_aa_offset_tbl_3_sel(RBIU_PIPE_aa_offset_tbl_3_sel),
        .iRBIU_PIPE_line_stipple_sel(RBIU_PIPE_line_stipple_sel),
```

```
        .iRBIU_PIPE_line_stipple_state_sel(RBIU_PIPE_line_stipple_state_sel),
        .iRBIU_PIPE_line_cntl_sel(RBIU_PIPE_line_cntl_sel),
        .iRBIU_PIPE_window_scissor_tl_sel(RBIU_PIPE_window_scissor_tl_sel),
        .iRBIU_PIPE_window_scissor_br_sel(RBIU_PIPE_window_scissor_br_sel),
        .iRBIU_PIPE_screen_scissor_tl_sel(RBIU_PIPE_screen_scissor_tl_sel),
        .iRBIU_PIPE_screen_scissor_br_sel(RBIU_PIPE_screen_scissor_br_sel),
        .iRBIU_PIPE_cliprect_0_tl_sel(RBIU_PIPE_cliprect_0_tl_sel),
        .iRBIU_PIPE_cliprect_0_br_sel(RBIU_PIPE_cliprect_0_br_sel),
        .iRBIU_PIPE_cliprect_1_tl_sel(RBIU_PIPE_cliprect_1_tl_sel),
        .iRBIU_PIPE_cliprect_1_br_sel(RBIU_PIPE_cliprect_1_br_sel),
        .iRBIU_PIPE_cliprect_2_tl_sel(RBIU_PIPE_cliprect_2_tl_sel),
        .iRBIU_PIPE_cliprect_2_br_sel(RBIU_PIPE_cliprect_2_br_sel),
        .iRBIU_PIPE_cliprect_3_tl_sel(RBIU_PIPE_cliprect_3_tl_sel),
        .iRBIU_PIPE_cliprect_3_br_sel(RBIU_PIPE_cliprect_3_br_sel),
        .iRBIU_PIPE_cliprect_rule_sel(RBIU_PIPE_cliprect_rule_sel),
        .iRBIU_PIPE_poly_offset_front_scale_sel(RBIU_PIPE_poly_offset_front_scale_sel),
        .iRBIU_PIPE_poly_offset_front_offset_sel(RBIU_PIPE_poly_offset_front_offset_sel),
        .iRBIU_PIPE_poly_offset_back_scale_sel(RBIU_PIPE_poly_offset_back_scale_sel),
        .iRBIU_PIPE_poly_offset_back_offset_sel(RBIU_PIPE_poly_offset_back_offset_sel),
        .iSTATE_VAR_INDX(sr_state_var_indx),
        .iSTATE_VAR_INDX_DLY(3'b000),
        .iZ_SV_INDX(zff_state_var_indx),
        .iQPP_SV_INDX(qpp_state_var_indx),
        .iPKR_SV_INDX(pkr_sv_indx),
        .iITER_SV_INDX(iterator_sv_indx),
```

```

        .oPIPE_RBIU_rdata(PIPE_RBIU_rdata),
        .oST_MSAA_ENABLE(st_msaa_enable),
        .oST_ITER_MSAA_ENABLE(st_iter_msaa_enable),
        .oST_AA_MASK(st_aa_mask),
        .oST_MSAA_NUM_SAMPLES(st_msaa_num_samples),
        .oST_ITER_MSAA_NUM_SAMPLES(st_iter_msaa_num_samples),
//        .oST_SCISSOR_EN(st_scissor_en),
//        .oST_DRAW_ZERO_LENGTH_LINE(st_draw_zero_length_line),
        .oST_WINDOW_OFFSET_DISABLE(st_window_offset_disable),
        .oST_WINDOW_SCISSOR_X_MIN(st_window_scissor_x_min),
        .oST_WINDOW_SCISSOR_X_MAX(st_window_scissor_x_max),
        .oST_WINDOW_SCISSOR_Y_MIN(st_window_scissor_y_min),
        .oST_WINDOW_SCISSOR_Y_MAX(st_window_scissor_y_max),
        .oST_SCREEN_SCISSOR_X_MIN(),
        .oST_SCREEN_SCISSOR_X_MAX(),
        .oST_SCREEN_SCISSOR_Y_MIN(),
        .oST_SCREEN_SCISSOR_Y_MAX(),
        .oST_X_OFFSET(st_x_offset),
        .oST_Y_OFFSET(st_y_offset),
        .oST_BRES_CNTL_EN(st_bres_cntl_en),
        .oST_BRES_CNTL_REG(st_bres_cntl_reg),

        .oST_OUTPUT_SCREEN_XY(st_output_screen_xy),
        .oST_LINE_STIPPLE_ENABLE(st_line_stipple_enable),
    .oST_JSS_ENABLE(st_jss_enable),
    .oST_ITER_JSS_ENABLE(st_iter_jss_enable),
    .oST_JSS_X_DIM(st_jss_x_dim),
    .oST_JSS_Y_DIM(st_jss_y_dim),
    .oST_MAX_SAMPLE_DIST(st_max_sample_dist),
    .oST_JSS_SAMPLE0_SEL(st_jss_sample0_sel),
    .oST_JSS_SAMPLE1_SEL(st_jss_sample1_sel),
    .oST_JSS_SAMPLE2_SEL(st_jss_sample2_sel),
    .oST_JSS_SAMPLE3_SEL(st_jss_sample3_sel),
    .oST_JSS_SAMPLE4_SEL(st_jss_sample4_sel),
    .oST_JSS_SAMPLE5_SEL(st_jss_sample5_sel),
    .oST_JSS_SAMPLE6_SEL(st_jss_sample6_sel),
    .oST_JSS_SAMPLE7_SEL(st_jss_sample7_sel),
    .oST_JSS_SAMPLE8_SEL(st_jss_sample8_sel),
    .oST_JSS_SAMPLE9_SEL(st_jss_sample9_sel),
    .oST_JSS_SAMPLE10_SEL(st_jss_sample10_sel),
    .oST_JSS_SAMPLE11_SEL(st_jss_sample11_sel),
    .oST_JSS_SAMPLE12_SEL(st_jss_sample12_sel),
    .oST_JSS_SAMPLE13_SEL(st_jss_sample13_sel),
    .oST_JSS_SAMPLE14_SEL(st_jss_sample14_sel),
    .oST_JSS_SAMPLE15_SEL(st_jss_sample15_sel),
//    .oST_MSAA_URC_SAMP_OFFSET_X(st_msaa_urc_samp_offset_x),
//    .oST_MSAA_URC_SAMP_OFFSET_Y(st_msaa_urc_samp_offset_y),
//    .oST_MSAA_LLC_SAMP_OFFSET_X(st_msaa_llc_samp_offset_x),
//    .oST_MSAA_LLC_SAMP_OFFSET_Y(st_msaa_llc_samp_offset_y),
```

```
//    .oST_MSAA_LRC_SAMP_OFFSET_X(st_msaa_lrc_samp_offset_x),
//    .oST_MSAA_LRC_SAMP_OFFSET_Y(st_msaa_lrc_samp_offset_y),
    .oST_SAMPLE_0_X(st_sample_0_x),
    .oST_SAMPLE_0_Y(st_sample_0_y),
    .oST_SAMPLE_1_X(st_sample_1_x),
    .oST_SAMPLE_1_Y(st_sample_1_y),
    .oST_SAMPLE_2_X(st_sample_2_x),
    .oST_SAMPLE_2_Y(st_sample_2_y),
    .oST_SAMPLE_3_X(st_sample_3_x),
    .oST_SAMPLE_3_Y(st_sample_3_y),
    .oST_SAMPLE_4_X(st_sample_4_x),
    .oST_SAMPLE_4_Y(st_sample_4_y),
    .oST_SAMPLE_5_X(st_sample_5_x),
    .oST_SAMPLE_5_Y(st_sample_5_y),
    .oST_SAMPLE_6_X(st_sample_6_x),
    .oST_SAMPLE_6_Y(st_sample_6_y),
    .oST_SAMPLE_7_X(st_sample_7_x),
    .oST_SAMPLE_7_Y(st_sample_7_y),
    .oST_SAMPLE_8_X(st_sample_8_x),
    .oST_SAMPLE_8_Y(st_sample_8_y),
    .oST_SAMPLE_9_X(st_sample_9_x),
    .oST_SAMPLE_9_Y(st_sample_9_y),
    .oST_SAMPLE_10_X(st_sample_10_x),
    .oST_SAMPLE_10_Y(st_sample_10_y),
    .oST_SAMPLE_11_X(st_sample_11_x),
```

```
    .oST_SAMPLE_11_Y(st_sample_11_y),
    .oST_SAMPLE_12_X(st_sample_12_x),
    .oST_SAMPLE_12_Y(st_sample_12_y),
    .oST_SAMPLE_13_X(st_sample_13_x),
    .oST_SAMPLE_13_Y(st_sample_13_y),
    .oST_SAMPLE_14_X(st_sample_14_x),
    .oST_SAMPLE_14_Y(st_sample_14_y),
    .oST_SAMPLE_15_X(st_sample_15_x),
    .oST_SAMPLE_15_Y(st_sample_15_y),
    .oST_LINE_PATTERN(st_line_pattern),
    .oST_REPEAT_COUNT(st_repeat_count),
//    .oST_PATTERN_START(st_pattern_start),
    .oST_PATTERN_BIT_ORDER(st_pattern_bit_order),
    .oST_AUTO_RESET_ENABLE(st_auto_reset_enable),
    .oST_CURRENT_PTR(st_current_ptr),
    .oST_CURRENT_COUNT(st_current_count),
    .oST_CLIPRECT_ENABLE(st_cliprect_enable),
    .oST_CLIPRECT_0_X_MIN(st_cliprect_0_x_min),
    .oST_CLIPRECT_0_Y_MIN(st_cliprect_0_y_min),
    .oST_CLIPRECT_0_X_MAX(st_cliprect_0_x_max),
    .oST_CLIPRECT_0_Y_MAX(st_cliprect_0_y_max),
    .oST_CLIPRECT_1_X_MIN(st_cliprect_1_x_min),
    .oST_CLIPRECT_1_Y_MIN(st_cliprect_1_y_min),
    .oST_CLIPRECT_1_X_MAX(st_cliprect_1_x_max),
    .oST_CLIPRECT_1_Y_MAX(st_cliprect_1_y_max),
```

```
    .oST_CLIPRECT_2_X_MIN(st_cliprect_2_x_min),
    .oST_CLIPRECT_2_Y_MIN(st_cliprect_2_y_min),
    .oST_CLIPRECT_2_X_MAX(st_cliprect_2_x_max),
    .oST_CLIPRECT_2_Y_MAX(st_cliprect_2_y_max),
    .oST_CLIPRECT_3_X_MIN(st_cliprect_3_x_min),
    .oST_CLIPRECT_3_Y_MIN(st_cliprect_3_y_min),
    .oST_CLIPRECT_3_X_MAX(st_cliprect_3_x_max),
    .oST_CLIPRECT_3_Y_MAX(st_cliprect_3_y_max),
    .oST_CLIP_RULE(st_clip_rule),
    .oST_POLY_OFFSET_PARA_ENABLE(st_poly_offset_para_enable),
    .oST_POLY_OFFSET_BACK_ENABLE(st_poly_offset_back_enable),
    .oST_POLY_OFFSET_FRONT_ENABLE(st_poly_offset_front_enable),
    .oST_POLY_OFFSET_FRONT_SCALE(st_poly_offset_front_scale),
    .oST_POLY_OFFSET_FRONT_OFFSET(st_poly_offset_front_offset),
    .oST_POLY_OFFSET_BACK_SCALE(st_poly_offset_back_scale),
    .oST_POLY_OFFSET_BACK_OFFSET(st_poly_offset_back_offset),
    .oST_SEND_CENTERS(st_send_centers),
    .oST_SEND_CENTROIDS(st_send_centroids)
  );

  sc_stage_reg usc_stage_reg(
   .iSRST(sc_srst),
   .iSCLK(sclk_sc),
   .iFREEZE_B(pipe_freeze_b_dly),
   .iFREEZE1_B(pipe_freeze_b_dly1),
  .iVALID(valid_in),
  .iPHASE(phase_in),
  .iP0(p0_in),
  .iP1(p1_in),
  .iP2(p2_in),
  .iP3(p3_in),
  .iP4(p4_in),
  .iXY0(xy0_in),
  .iXY1(xy1_in),
  .iXY2(xy2_in),
  .iZMINMAX(zminmax_in),
  .iCNTL(cntl_in),
  .iV0_INDX(v0_indx_in),
  .iRT_SET_CNTX0_BUSY(rt_set_cntx0_busy),
  .iCNTX0_DECR(cntx0_decr),
  .iCNTX1TO7_DECR(cntx1to7_decr),
  .iPKR_ITER_CNTX0_BUSY(pkr_iter_cntx0_busy),
  .iPKR_ITER_CNTX1TO7_BUSY(pkr_iter_cntx1to7_busy),

  .oPRIM_WE(sr_prim_we),
  .oZ_WE(sr_z_we),
  .oPIPE_VALID(sr_pipe_valid),
  .oEVENT(sr_event),
  .oEVENT_ID(sr_event_id),
  .oNULL_PRIM(sr_null_prim),
```

```
  .oDEALLOC_SLOT(sr_dealloc_slot),
  .oFIRST_PRIM_OF_SLOT(sr_first_prim_of_slot),
  .oEND_OF_PKT(sr_end_of_pkt),
  .oBACK_FACE(sr_back_face),
  .oPROVOKING_VERTEX(sr_provoking_vertex),
  .oX_MAJOR(sr_x_major),
  .oSTART_IN_DIAMOND(sr_start_in_diamond),
  .oEND_IN_DIAMOND(sr_end_in_diamond),
  .oPRIM_TYPE(sr_prim_type),
  .oPHASE(sr_phase),
  .oSTATE_VAR_INDX(sr_state_var_indx),
  .oV0(sr_v0),
  .oV1(sr_v1),
  .oV2(sr_v2),
  .oREF_X(sr_ref_x),
  .oREF_Y(sr_ref_y),
  .oI0(sr_i0),
  .oIX(sr_ix),
  .oIY(sr_iy),
  .oJ0(sr_j0),
  .oJX(sr_jx),
  .oJY(sr_jy),
  .oW0(sr_w0),
  .oWX(sr_wx),
  .oWY(sr_wy),
```

```
  .oPARAM_CACHE_INDX0(sr_param_cache_indx0),
  .oPARAM_CACHE_INDX1(sr_param_cache_indx1),
  .oPARAM_CACHE_INDX2(sr_param_cache_indx2),
  .oNULL_PRIM_ZFF(sr_null_prim_zff),
  .oZY_MAX_ZFF(sr_zy_max_zff),
  .oBACK_FACE_ZFF(sr_back_face_zff),
  .oPRIM_TYPE_ZFF(sr_prim_type_zff),
  .oPOLYMODE_ZFF(sr_polymode_zff),
  .oSTATE_VAR_INDX_ZFF(sr_state_var_indx_zff),
  .oREF_X_ZFF(sr_ref_x_zff),
  .oREF_Y_ZFF(sr_ref_y_zff),
  .oZ0_ZFF(sr_z0_zff),
  .oZX_ZFF(sr_zx_zff),
  .oZY_ZFF(sr_zy_zff),
  .oZ_MIN_ZFF(sr_z_min_zff),
  .oZ_MAX_ZFF(sr_z_max_zff),
  .oCNTX0_BUSY(sr_cntx0_busy),
  .oCNTX1TO7_BUSY(sr_cntx1to7_busy)
);


sc_primfifo usc_primfifo(
  .we(sr_prim_we),
  .re(prim_ff_re),
  .full(prim_ff_full),
```

```
  .empty(prim_ff_empty),
  .busy(),
  .write_data(prim_ff_wr_data),
  .read_data(prim_ff_rd_data),
  .clk(sclk_sc),
  .reset(sc_srst)
);


sc_zfifo usc_zfifo(
  .we(sr_z_we),
  .re(z_ff_re),
  .full(z_ff_full),
  .empty(z_ff_empty),
  .busy(),
  .write_data(z_ff_wr_data),
  .read_data(z_ff_rd_data),
  .clk(sclk_sc),
  .reset(sc_srst)
);


assign st_scissor_en = 1'b1;

sc_pipe usc_pipe(
 .iPIPE_FREEZE_B(pipe_freeze_b_dly),
 .iSCLK(sclk_sc),
 .iSRST(sc_srst),
 .iPIPE_RTS(sr_pipe_valid),
 .iPIPE_PHASE(sr_phase),
 .iEVENT(sr_event),
 .iEVENT_ID(sr_event_id),
 .iX_MAJOR(sr_x_major),
 .iSTART_IN_DIAMOND(sr_start_in_diamond),
 .iEND_IN_DIAMOND(sr_end_in_diamond),
 .iNULL_PRIM(sr_null_prim),
 .iPRIM_TYPE(sr_prim_type),
 .iV0(sr_v0),
 .iV1(sr_v1),
 .iV2(sr_v2),
 .iV0_EARLY(xy0_in),
 .iV1_EARLY(xy1_in),
 .iST_MSAA_ENABLE(st_msaa_enable),
 .iST_JSS_ENABLE(st_jss_enable),
 .iST_MAX_SAMPLE_DIST(st_max_sample_dist),
 .iST_LINE_STIPPLE_ENABLE(st_line_stipple_enable),
 .iST_SCISSOR_EN(st_scissor_en),
 .iST_DRAW_ZERO_LENGTH_LINE(st_draw_zero_length_line),
 .iST_X_MIN(st_window_scissor_x_min),
 .iST_X_MAX(st_window_scissor_x_max),
```

```
 .iST_Y_MIN(st_window_scissor_y_min),
 .iST_Y_MAX(st_window_scissor_y_max),
 .iST_X_OFFSET(st_x_offset),
 .iST_Y_OFFSET(st_y_offset),
 .iST_BRES_CNTL_EN(st_bres_cntl_en),
 .iST_BRES_CNTL_REG(st_bres_cntl_reg),

 .oPIPE_RTS(pipe_rts),
 .oPIPE_PHASE(pipe_phase),
 .oEVENT(event_flag),
 .oEVENT_ID(event_id),
 .oE0(e0),
 .oE0_Y(e0_y),
 .oE0_DX(e0_dx),
 .oE0_DY(e0_dy),
 .oE1(e1),
 .oE1_X(e1_x),
 .oE1_DX(e1_dx),
 .oE1_DY(e1_dy),
 .oE2(e2),
 .oE2_X(e2_x),
 .oE2_DX(e2_dx),
 .oE2_DY(e2_dy),
 .oX_DIR(x_dir),
 .oX_START(x_start),
```

```
 .oX_END(x_end),
 .oY_DIR(y_dir),
 .oY_START(y_start),
 .oY_END(y_end),
 .oBB_FRACT_BITS(bb_fract_bits),
 .oPASS_EMPTY_PRIM(pass_empty_prim)
);


sc_coarse_walker usc_coarse_walker(
 .iSCLK(sclk_sc),
 .iSRST(sc_srst),
 .iPIPE_RTS(pipe_rts),
 .iPIPE_PHASE(pipe_phase),
 .iEVENT(event_flag),
 .iEVENT_ID(event_id),
 .iBB_FRACT_BITS(bb_fract_bits),
 .iPASS_EMPTY_PRIM(pass_empty_prim),
 .iE0(e0),
 .iE0_Y(e0_y),
 .iE0_DX(e0_dx),
 .iE0_DY(e0_dy),
 .iE1(e1),
 .iE1_X(e1_x),
 .iE1_DX(e1_dx),
```

```
 .iE1_DY(e1_dy),
 .iE2(e2),
 .iE2_X(e2_x),
 .iE2_DX(e2_dx),
 .iE2_DY(e2_dy),
 .iX_DIR(x_dir),
 .iX_START(x_start),
 .iX_END(x_end),
 .iY_DIR(y_dir),
 .iY_START(y_start),
 .iY_END(y_end),
 .iQMASK_FF_ALM_FULL(tile_ff_full),
 .iRC_RTR(rc_rtr),

 .oEVENT(cw_event),
 .oEVENT_ID(cw_event_id),
 .oBB_FRACT_BITS(cw_bb_fract_bits),
 .oPASS_EMPTY_PRIM(cw_pass_empty_prim),
 .oE0(cw_e0),
 .oE0_Y(cw_e0y),
 .oE0_DX(cw_dxe0),
 .oE0_DY(cw_dye0),
 .oE1(cw_e1),
 .oE1_X(cw_e1x),
 .oE1_DX(cw_dxe1),
  .oE1_DY(cw_dye1),
  .oE2(cw_e2),
  .oE2_X(cw_e2x),
  .oE2_DX(cw_dxe2),
  .oE2_DY(cw_dye2),
  .oX_DIR(cw_xdir),
  .oX_CURR(cw_tilex),
  .oX_MIN(cw_xmin),
  .oX_MAX(cw_xmax),
  .oY_DIR(cw_ydir),
  .oY_CURR(cw_tiley),
  .oY_MIN(cw_ymin),
  .oY_MAX(cw_ymax),
  .oLAST_TILE_OF_PRIM(cw_last_tile),
  .oZ_FF_RD_EN(z_ff_re),
  .oPRIM_RTS(cw_tile_valid),
  .oPIPE_FREEZE_B_EARLY(pipe_freeze_b_early),
  .oPIPE_FREEZE_B_DLY(pipe_freeze_b_dly),
  .oPIPE_FREEZE_B_DLY1(pipe_freeze_b_dly1),
  .oQMSK_Z_FREEZE_B(qmsk_z_freeze_b)
);

// rwr - tmp drivers until sigs are driven
assign cw_xmajor = 1'b0;

```

```
sc_quadmask usc_quadmask(
  .iSCLK(sclk_sc),
  .freezeb(qmsk_z_freeze_b),
  .tile_valid(cw_tile_valid),
  .event_in(cw_event),
  .event_id_in(cw_event_id),
  .last_tile_in(cw_last_tile),
  .pass_empty_prim_in(cw_pass_empty_prim),
  .xdir_in(cw_xdir),
  .ydir_in(cw_ydir),
  .xmajor_in(cw_xmajor),
  .tilex_in(cw_tilex),
  .tiley_in(cw_tiley),
  .xmin_in(cw_xmin),
  .xmax_in(cw_xmax),
  .ymin_in(cw_ymin),
  .ymax_in(cw_ymax),
  .bb_fract_bits_in(cw_bb_fract_bits),
  .e0y(cw_e0y),
  .e1x(cw_e1x),
  .e2x(cw_e2x),
  .dxe0_in(cw_dxe0),
  .dye0_in(cw_dye0),
  .dxe1_in(cw_dxe1),
  .dye1_in(cw_dye1),
```

```
  .dxe2_in(cw_dxe2),
  .dye2_in(cw_dye2),
  .e0_in(cw_e0),
  .e1_in(cw_e1),
  .e2_in(cw_e2),

  .last_tile_out(qm_last_tile),
  .event_out(qm_event),
  .event_id_out(qm_event_id),
  .xdir_out(qm_xdir),
  .ydir_out(qm_ydir),
  .xmajor_out(qm_xmajor),
  .tilex_out(qm_tilex),
  .tiley_out(qm_tiley),
  .tilex_m3_out(qm_tilex_m3),
  .tiley_m3_out(qm_tiley_m3),
  .xmin_out(qm_xmin),
  .xmax_out(qm_xmax),
  .ymin_out(qm_ymin),
  .ymax_out(qm_ymax),
  .bb_fract_bits_out(qm_bb_fract_bits),
  .z_mask_needed_out(qm_z_mask_needed),
  .e0_out(qm_e0),
  .e1_out(qm_e1),
  .e2_out(qm_e2),
```

```
  .dxe0_out(qm_dxe0),
  .dye0_out(qm_dye0),
  .dxe1_out(qm_dxe1),
  .dye1_out(qm_dye1),
  .dxe2_out(qm_dxe2),
  .dye2_out(qm_dye2),
  .quadmask_out(qm_quadmask),
  .quadmask_valid(qm_quadmask_valid)
);


sc_z_interp usc_z_interp(
  .iSCLK(sclk_sc),
  .iSRST(sc_srst),
  .iFREEZE_B(qmsk_z_freeze_b),
  .iZ0(zff_z0),
  .iZX(zff_zx),
  .iZY(zff_zy),
  .iZ_MIN(zff_z_min),
  .iZ_MAX(zff_z_max),
  .iREF_X(zff_ref_x),
  .iREF_Y(zff_ref_y),
  .iZY_MAX(zff_zy_max),
  .iBACK_FACE(zff_back_face),
  .iPRIM_TYPE(zff_prim_type),
  .iPOLYMODE(zff_polymode),
  .iNULL_PRIM(zff_null_prim),
  .iSTATE_ID(zff_state_var_indx),
  .iST_OFFSET_FRONT_EN(st_poly_offset_front_enable),
  .iST_OFFSET_BACK_EN(st_poly_offset_back_enable),
  .iST_OFFSET_PARA_EN(st_poly_offset_para_enable),
  .iST_OFFSET_FRONT(st_poly_offset_front_offset),
  .iST_OFFSET_BACK(st_poly_offset_back_offset),
  .iST_SCALE_FRONT(st_poly_offset_front_scale),
  .iST_SCALE_BACK(st_poly_offset_back_scale),


  .iEVENT(qm_event),
  .iEVENT_ID(qm_event_id),
  .iTILEX(qm_tilex),
  .iTILEY(qm_tiley),
  .iMASK(qm_quadmask),
  .iQM_VALID(qm_quadmask_valid),
  .iZ_MASK_NEEDED(qm_z_mask_needed),

  .oEVENT(rc_event),
  .oTILEX(rc_tilex),
  .oTILEY(rc_tiley),
  .oMINZ(rc_minz),
  .oMAXZ(rc_maxz),
```

```
  .oZPLANE(rc_zplane),
  .oMASK(rc_mask),
  .oBACK(rc_back),
  .oSTATE_ID(rc_state),
  .oCOVERED(rc_covered),
  .oVALID(rc_send)
);

// !!!! rwr - tmp drivers until test can be updated
wire [13:0] tmp_st_cliprect_0_x_min = 'b0;
wire [13:0] tmp_st_cliprect_0_y_min = 'b0;
wire [13:0] tmp_st_cliprect_0_x_max = 'b0;
wire [13:0] tmp_st_cliprect_0_y_max = 'b0;
wire [13:0] tmp_st_cliprect_1_x_min = 'b0;
wire [13:0] tmp_st_cliprect_1_y_min = 'b0;
wire [13:0] tmp_st_cliprect_1_x_max = 'b0;
wire [13:0] tmp_st_cliprect_1_y_max = 'b0;
wire [13:0] tmp_st_cliprect_2_x_min = 'b0;
wire [13:0] tmp_st_cliprect_2_y_min = 'b0;
wire [13:0] tmp_st_cliprect_2_x_max = 'b0;
wire [13:0] tmp_st_cliprect_2_y_max = 'b0;
wire [13:0] tmp_st_cliprect_3_x_min = 'b0;
wire [13:0] tmp_st_cliprect_3_y_min = 'b0;
wire [13:0] tmp_st_cliprect_3_x_max = 'b0;
wire [13:0] tmp_st_cliprect_3_y_max = 'b0;
```

```

wire  [2:0] tmp_st_msaa_urc_samp_offset_x = 'b0;
wire  [2:0] tmp_st_msaa_urc_samp_offset_y = 'b0;
wire  [2:0] tmp_st_msaa_llc_samp_offset_x = 'b0;
wire  [2:0] tmp_st_msaa_llc_samp_offset_y = 'b0;
wire  [2:0] tmp_st_msaa_lrc_samp_offset_x = 'b0;
wire  [2:0] tmp_st_msaa_lrc_samp_offset_y = 'b0;

wire [127:0] tmp_st_aa_offset_tbl = 'b0;


sc_qdpr_proc usc_qdpr_proc(
  .iSCLK(sclk_sc),
  .iSRST(sc_srst),

  .iPRIM_FIFO_EMPTY(prim_ff_empty),
  .iINTERP_PRIM_DATA(prim_ff_rd_data),
  .oPRIM_FIFO_ADVANCE(prim_ff_re),
  .oCNTX0_DECR(cntx0_decr),
  .oCNTX1TO7_DECR(cntx1to7_decr),

  .iQM_QUADMASK_VALID(qm_quadmask_valid),
  .iQM_DATA(tile_ff_wr_data),
  .oTILEDATA_FIFO_FULL(tile_ff_full),

```

```
  .iRC_SC_HIER_MASK(rc_in_hier_mask),
  .iRC_SC_RB_ID(rc_in_rb_id),
  .iRC_SC_SPLIT(rc_in_split),
  .iRC_SC_HIER_SEND(rc_in_hier_send),
  .oSC_RC_HIER_RTR(rc_hier_rtr),

  .iFREEZE(qdpkr_in_fz),

  .oQPP_SV_INDX(qpp_state_var_indx),
  .iSV_MSAA_ENABLE(st_msaa_enable),
  .iSV_JSS_ENABLE(st_jss_enable),
  .iSV_MSAA_NUM_SAMPLES(st_msaa_num_samples[2:0]),
  .iSV_AA_MASK(st_aa_mask),
  .iSV_JSS_X_DIM(st_jss_x_dim),
  .iSV_JSS_Y_DIM(st_jss_y_dim),
  .iSV_JSS_SAMPLE_SEL({st_jss_sample0_sel[2:0],
                       st_jss_sample1_sel[2:0],
                       st_jss_sample2_sel[2:0],
                       st_jss_sample3_sel[2:0],
                       st_jss_sample4_sel[2:0],
                       st_jss_sample5_sel[2:0],
                       st_jss_sample6_sel[2:0],
                       st_jss_sample7_sel[2:0],
                       st_jss_sample8_sel[2:0],
                       st_jss_sample9_sel[2:0],
                    st_jss_sample10_sel[2:0],
                    st_jss_sample11_sel[2:0],
                    st_jss_sample12_sel[2:0],
                    st_jss_sample13_sel[2:0],
                    st_jss_sample14_sel[2:0],
                    st_jss_sample15_sel[2:0]}),
        .iSV_CLIPRECT_ENABLE(st_cliprect_enable),
        .iSV_CLIPRECT_RULE(st_clip_rule),
        .iSV_CLIPRECT_0_XMIN(tmp_st_cliprect_0_x_min),
        .iSV_CLIPRECT_0_YMIN(tmp_st_cliprect_0_y_min),
        .iSV_CLIPRECT_0_XMAX(tmp_st_cliprect_0_x_max),
        .iSV_CLIPRECT_0_YMAX(tmp_st_cliprect_0_y_max),
        .iSV_CLIPRECT_1_XMIN(tmp_st_cliprect_1_x_min),
        .iSV_CLIPRECT_1_YMIN(tmp_st_cliprect_1_y_min),
        .iSV_CLIPRECT_1_XMAX(tmp_st_cliprect_1_x_max),
        .iSV_CLIPRECT_1_YMAX(tmp_st_cliprect_1_y_max),
        .iSV_CLIPRECT_2_XMIN(tmp_st_cliprect_2_x_min),
        .iSV_CLIPRECT_2_YMIN(tmp_st_cliprect_2_y_min),
        .iSV_CLIPRECT_2_XMAX(tmp_st_cliprect_2_x_max),
        .iSV_CLIPRECT_2_YMAX(tmp_st_cliprect_2_y_max),
        .iSV_CLIPRECT_3_XMIN(tmp_st_cliprect_3_x_min),
        .iSV_CLIPRECT_3_YMIN(tmp_st_cliprect_3_y_min),
        .iSV_CLIPRECT_3_XMAX(tmp_st_cliprect_3_x_max),
        .iSV_CLIPRECT_3_YMAX(tmp_st_cliprect_3_y_max),

```

```
        .oQPP_PRIM_DATA(qpp_prim_data),
        .oQPP_FPOS_EARLY(qpp_fpos_early),

        .oQPP_LAST_QDPAIR_OF_PRIM(qpp_last_qdpair_of_prim),

        .oQPP_Q0_LAST_OF_TILE(qpp_q0_last_of_tile),
        .oQPP_Q0_ZMASK_NEEDED(qpp_q0_zmask_needed),
        .oQPP_Q0_QHIT(qpp_q0_qhit),
        .oQPP_Q0_TILEX(qpp_q0_tilex),
        .oQPP_Q0_TILEY(qpp_q0_tiley),
        .oQPP_Q0_QUADX(qpp_q0_quadx),
        .oQPP_Q0_QUADY(qpp_q0_quady),
        .oQPP_Q0_RB_ID(qpp_q0_rb_id),
        .oQPP_Q0_SPLIT(qpp_q0_split),
        .oQPP_Q0_ULC_SAMPLE_MASK(qpp_q0_ulc_sample_mask),
        .oQPP_Q0_URC_SAMPLE_MASK(qpp_q0_urc_sample_mask),
        .oQPP_Q0_LLC_SAMPLE_MASK(qpp_q0_llc_sample_mask),
        .oQPP_Q0_LRC_SAMPLE_MASK(qpp_q0_lrc_sample_mask),
        .oQPP_Q0_ULC_CNTRMOST_SAMPLE_ID(qpp_q0_ulc_cntrmost_sample_id),
        .oQPP_Q0_URC_CNTRMOST_SAMPLE_ID(qpp_q0_urc_cntrmost_sample_id),
        .oQPP_Q0_LLC_CNTRMOST_SAMPLE_ID(qpp_q0_llc_cntrmost_sample_id),
        .oQPP_Q0_LRC_CNTRMOST_SAMPLE_ID(qpp_q0_lrc_cntrmost_sample_id),

        .oQPP_Q1_LAST_OF_TILE(qpp_q1_last_of_tile),
        .oQPP_Q1_ZMASK_NEEDED(qpp_q1_zmask_needed),
```

```
        .oQPP_Q1_QHIT(qpp_q1_qhit),
        .oQPP_Q1_TILEX(qpp_q1_tilex),
        .oQPP_Q1_TILEY(qpp_q1_tiley),
        .oQPP_Q1_QUADX(qpp_q1_quadx),
        .oQPP_Q1_QUADY(qpp_q1_quady),
        .oQPP_Q1_RB_ID(qpp_q1_rb_id),
        .oQPP_Q1_SPLIT(qpp_q1_split),
        .oQPP_Q1_ULC_SAMPLE_MASK(qpp_q1_ulc_sample_mask),
        .oQPP_Q1_URC_SAMPLE_MASK(qpp_q1_urc_sample_mask),
        .oQPP_Q1_LLC_SAMPLE_MASK(qpp_q1_llc_sample_mask),
        .oQPP_Q1_LRC_SAMPLE_MASK(qpp_q1_lrc_sample_mask),
        .oQPP_Q1_ULC_CNTRMOST_SAMPLE_ID(qpp_q1_ulc_cntrmost_sample_id),
        .oQPP_Q1_URC_CNTRMOST_SAMPLE_ID(qpp_q1_urc_cntrmost_sample_id),
        .oQPP_Q1_LLC_CNTRMOST_SAMPLE_ID(qpp_q1_llc_cntrmost_sample_id),
        .oQPP_Q1_LRC_CNTRMOST_SAMPLE_ID(qpp_q1_lrc_cntrmost_sample_id)
    );

    wire [`SC_QD_DATA_WIDTH-1:0]  qd0_data;
    wire [`SC_QD_DATA_WIDTH-1:0]  qd1_data;

    assign qd0_data[`SC_QD_TILEX]          = qpp_q0_tilex;
    assign qd0_data[`SC_QD_TILEY]          = qpp_q0_tiley;
    assign qd0_data[`SC_QD_QUADX]          = qpp_q0_quadx;
    assign qd0_data[`SC_QD_QUADY]          = qpp_q0_quady;
    assign qd0_data[`SC_QD_RB_ID]          = qpp_q0_rb_id;
```

```
    assign qd0_data[`SC_QD_SPLIT]              = qpp_q0_split;
    assign qd0_data[`SC_QD_ULC_SAMPLE_MASK]    = qpp_q0_ulc_sample_mask;
    assign qd0_data[`SC_QD_URC_SAMPLE_MASK]    = qpp_q0_urc_sample_mask;
    assign qd0_data[`SC_QD_LLC_SAMPLE_MASK]    = qpp_q0_llc_sample_mask;
    assign qd0_data[`SC_QD_LRC_SAMPLE_MASK]    = qpp_q0_lrc_sample_mask;
    assign   qd0_data[`SC_QD_ULC_CNTRMOST_SAMPLE_ID]                    =
qpp_q0_ulc_cntrmost_sample_id;
    assign   qd0_data[`SC_QD_URC_CNTRMOST_SAMPLE_ID]                    =
qpp_q0_urc_cntrmost_sample_id;
    assign   qd0_data[`SC_QD_LLC_CNTRMOST_SAMPLE_ID]                    =
qpp_q0_llc_cntrmost_sample_id;
    assign   qd0_data[`SC_QD_LRC_CNTRMOST_SAMPLE_ID]                    =
qpp_q0_lrc_cntrmost_sample_id;

    assign qd1_data[`SC_QD_TILEX]              = qpp_q1_tilex;
    assign qd1_data[`SC_QD_TILEY]              = qpp_q1_tiley;
    assign qd1_data[`SC_QD_QUADX]              = qpp_q1_quadx;
    assign qd1_data[`SC_QD_QUADY]              = qpp_q1_quady;
    assign qd1_data[`SC_QD_RB_ID]              = qpp_q1_rb_id;
    assign qd1_data[`SC_QD_SPLIT]              = qpp_q1_split;
    assign qd1_data[`SC_QD_ULC_SAMPLE_MASK]    = qpp_q1_ulc_sample_mask;
    assign qd1_data[`SC_QD_URC_SAMPLE_MASK]    = qpp_q1_urc_sample_mask;
    assign qd1_data[`SC_QD_LLC_SAMPLE_MASK]    = qpp_q1_llc_sample_mask;
    assign qd1_data[`SC_QD_LRC_SAMPLE_MASK]    = qpp_q1_lrc_sample_mask;
    assign   qd1_data[`SC_QD_ULC_CNTRMOST_SAMPLE_ID]                    =
qpp_q1_ulc_cntrmost_sample_id;
    assign   qd1_data[`SC_QD_URC_CNTRMOST_SAMPLE_ID]                    =
qpp_q1_urc_cntrmost_sample_id;
```

```
   assign qd1_data[`SC_QD_LLC_CNTRMOST_SAMPLE_ID] = qpp_q1_llc_cntrmost_sample_id;
   assign qd1_data[`SC_QD_LRC_CNTRMOST_SAMPLE_ID] = qpp_q1_lrc_cntrmost_sample_id;


   sc_packer usc_packer(
     .sclk(sclk_sc),
     .srst(sc_srst),

     .sp0_disable(ROM_SP0_disable),
     .sp1_disable(ROM_SP1_disable),
     .sp2_disable(ROM_SP2_disable),
     .sp3_disable(ROM_SP3_disable),

     .qd0_hit(qpp_q0_qhit),
     .qd0_lqt(qpp_q0_last_of_tile),
     .qd0_zneeded(qpp_q0_zmask_needed),
     .qd0_data(qd0_data),

     .qd1_hit(qpp_q1_qhit),
     .qd1_lqt(qpp_q1_last_of_tile),
     .qd1_zneeded(qpp_q1_zmask_needed),
     .qd1_data(qd1_data),

     .last_qdpair_of_prim(qpp_last_qdpair_of_prim),
```

```
     .early_fpos(qpp_fpos_early),
     .primdata(qpp_prim_data),

     .iterator_input_fz(iterator_input_fz),
     .detail_mask_accum_rdy(detail_mask_accum_rdy),

    //outputs
     .qdpkr_in_fz(qdpkr_in_fz),
     .pkr_qd0(pkr_qd0),
     .pkr_qd1(pkr_qd1),
     .pkr_qd2(pkr_qd2),
     .pkr_qd3(pkr_qd3),
     .pkr_qdhit0(pkr_qdhit0),
     .pkr_qdhit1(pkr_qdhit1),
     .pkr_qdhit2(pkr_qdhit2),
     .pkr_qdhit3(pkr_qdhit3),
     .pkr_primdata(pkr_primdata),
     .pkr_ds_one_clk_command(pkr_ds_one_clk_command),
     .pkr_ds_end_of_prim(pkr_ds_end_of_prim),
     .pkr_ds_end_of_vector(pkr_ds_end_of_vector),
     .pkr_send_row(pkr_send_row),

     .pkr_sv_indx(pkr_sv_indx),
     .pkr_cntx0_busy(pkr_cntx0_busy),
     .pkr_cntx1to7_busy(pkr_cntx1to7_busy),
```

```

     //control signals to the detail mask accumulater
     .detail_hit_0(detail_hit_0),
     .detail_lqt_0(detail_lqt_0),
     .detail_hit_1(detail_hit_1),
     .detail_lqt_1(detail_lqt_1)
   );


   sc_detail_mask_accum usc_detail_mask_accum (
     .iSCLK(sclk_sc),
     .iSRST(sc_srst),

     .oRDY_FOR_TILE_DATA(detail_mask_accum_rdy),

     .iX_0(qd0_data[`SC_QD_QUADX]),
     .iY_0(qd0_data[`SC_QD_QUADY]),
     .iHIT_0(detail_hit_0),
     .iEOT_0(detail_lqt_0),

     .iX_1(qd1_data[`SC_QD_QUADX]),
     .iY_1(qd1_data[`SC_QD_QUADY]),
     .iHIT_1(detail_hit_1),
     .iEOT_1(detail_lqt_1),

```

```
     .oDETAIL_MASK(detail_mask),

     .oDETAIL_MASK_VALID(detail_mask_valid),
     .iRDY_FOR_DETAIL_MASK(rdy_for_detail_mask)
   );

   sc_iter usc_iter (
     .sclk(sclk_sc),
     .srst(sc_srst),
     .st_send_centers(st_send_centers),
     .st_send_centroids(st_send_centroids),
     .st_output_screen_xy(st_output_screen_xy),
     .st_iter_msaa_enable(st_iter_msaa_enable),
     .st_iter_msaa_num_samples(st_iter_msaa_num_samples),
     .st_iter_jss_enable(st_iter_jss_enable),

     .pkr_qd0(pkr_qd0),
     .pkr_qd1(pkr_qd1),
     .pkr_qd2(pkr_qd2),
     .pkr_qd3(pkr_qd3),
     .pkr_qdhit0(pkr_qdhit0),
     .pkr_qdhit1(pkr_qdhit1),
     .pkr_qdhit2(pkr_qdhit2),
     .pkr_qdhit3(pkr_qdhit3),
     .pkr_primdata(pkr_primdata),
    .pkr_ds_one_clk_command(pkr_ds_one_clk_command),
    .pkr_ds_end_of_prim(pkr_ds_end_of_prim),
    .pkr_ds_end_of_vector(pkr_ds_end_of_vector),
    .pkr_send_row(pkr_send_row),

    .pkr_cntx0_busy(pkr_cntx0_busy),
    .pkr_cntx1to7_busy(pkr_cntx1to7_busy),
    .pkr_iter_cntx0_busy(pkr_iter_cntx0_busy),
    .pkr_iter_cntx1to7_busy(pkr_iter_cntx1to7_busy),

    .iterator_sv_indx(iterator_sv_indx),
    .iterator_input_fz(iterator_input_fz),
    .iterator_SX0_quad_x(iterator_SX0_quad_x),
    .iterator_SX0_quad_y(iterator_SX0_quad_y),
    .iterator_SX0_quad_mask(iterator_SX0_quad_mask),
    .iterator_SX0_quad_tilex(iterator_SX0_quad_tilex),
    .iterator_SX0_quad_tiley(iterator_SX0_quad_tiley),
    .iterator_SX0_quad_rb_id(iterator_SX0_quad_rb_id),
    .iterator_SX0_quad_split(iterator_SX0_quad_split),
    .iterator_SX0_quad_send(iterator_SX0_quad_send),
    .SX0_iterator_quad_rtr(SX0_iterator_quad_rtr),

    .iterator_SX1_quad_x(iterator_SX1_quad_x),
    .iterator_SX1_quad_y(iterator_SX1_quad_y),
    .iterator_SX1_quad_mask(iterator_SX1_quad_mask),
```

```
    .iterator_SX1_quad_tilex(iterator_SX1_quad_tilex),
    .iterator_SX1_quad_tiley(iterator_SX1_quad_tiley),
    .iterator_SX1_quad_rb_id(iterator_SX1_quad_rb_id),
    .iterator_SX1_quad_split(iterator_SX1_quad_split),
    .iterator_SX1_quad_send(iterator_SX1_quad_send),
    .SX1_iterator_quad_rtr(SX1_iterator_quad_rtr),

    .iterator_SQ_data(iterator_SQ_data),
    .iterator_SQ_valid(iterator_SQ_valid),
    .SQ_iterator_free_buff(SQ_iterator_free_buff),
    .SQ_iterator_dec_cntr_cnt(SQ_iterator_dec_cntr_cnt),

    .iterator_SP0_data(iterator_SP0_data),
    .iterator_SP0_type(iterator_SP0_type),
    .iterator_SP0_last_quad(iterator_SP0_last_quad),
    .iterator_SP0_valid(iterator_SP0_valid),

    .iterator_SP1_data(iterator_SP1_data),
    .iterator_SP1_type(iterator_SP1_type),
    .iterator_SP1_last_quad(iterator_SP1_last_quad),
    .iterator_SP1_valid(iterator_SP1_valid),

    .iterator_SP2_data(iterator_SP2_data),
    .iterator_SP2_type(iterator_SP2_type),
    .iterator_SP2_last_quad(iterator_SP2_last_quad),
```

```
    .iterator_SP2_valid(iterator_SP2_valid),

    .iterator_SP3_data(iterator_SP3_data),
    .iterator_SP3_type(iterator_SP3_type),
    .iterator_SP3_last_quad(iterator_SP3_last_quad),
    .iterator_SP3_valid(iterator_SP3_valid)
  );

//--------------------------------------------------
// Assign statements
//--------------------------------------------------

// Concatenate inputs to register them
  assign pa_sc_inputs = {PA_SC_valid,     //    [270]
                         PA_SC_v0_indx,   // [269:268]
                         PA_SC_cntl,      // [267:238]
                         PA_SC_phase,     // [237:236]
                         PA_SC_zminmax,   // [235:222]
                         PA_SC_xy0,       // [221:204]
                         PA_SC_xy1,       // [203:186]
                         PA_SC_xy2,       // [185:168]
                         PA_SC_p0,        // [167:136]
                         PA_SC_p1,        //  [135:96]
                         PA_SC_p2,        //   [95:64]
                         PA_SC_p3,        //   [63:32]
```

```
                         PA_SC_p4};       //    [31:0]

// Assign registered inputs for use in sc_stage_reg
  assign valid_in = pa_sc_inputs_reg[270];
  assign v0_indx_in = pa_sc_inputs_reg[269:268];
  assign cntl_in = pa_sc_inputs_reg[267:238];
  assign phase_in = pa_sc_inputs_reg[237:236];
  assign zminmax_in = pa_sc_inputs_reg[235:222];
  assign xy0_in = pa_sc_inputs_reg[221:204];
  assign xy1_in = pa_sc_inputs_reg[203:186];
  assign xy2_in = pa_sc_inputs_reg[185:168];
  assign p0_in = pa_sc_inputs_reg[167:136];
  assign p1_in = pa_sc_inputs_reg[135:96];
  assign p2_in = pa_sc_inputs_reg[95:64];
  assign p3_in = pa_sc_inputs_reg[63:32];
  assign p4_in = pa_sc_inputs_reg[31:0];

// Concatenate some of outputs of sc_stage_reg to create write data for
// primitive fifo.
  assign prim_ff_wr_data[`SC_PI_EVENT]            = sr_event;
  assign prim_ff_wr_data[`SC_PI_EVENTID]          = sr_event_id;
  assign prim_ff_wr_data[`SC_PI_DEALLOCATE_SLOT]  = sr_dealloc_slot;
  assign prim_ff_wr_data[`SC_PI_FIRST_PRIM_OF_SLOT] = sr_first_prim_of_slot;
  assign prim_ff_wr_data[`SC_PI_END_OF_PACKET]    = sr_end_of_pkt;
  assign prim_ff_wr_data[`SC_PI_BACK_FACE]        = sr_back_face;
```

```
    assign prim_ff_wr_data[`SC_PI_PROVOKING_VERT]     = sr_provoking_vertex;
    assign prim_ff_wr_data[`SC_PI_STATE]              = sr_state_var_indx;
    assign prim_ff_wr_data[`SC_PI_TYPE]               = sr_prim_type;
    assign prim_ff_wr_data[`SC_PI_V0X]                = sr_ref_x;
    assign prim_ff_wr_data[`SC_PI_V0Y]                = sr_ref_y;
    assign prim_ff_wr_data[`SC_PI_PC_PTR0]            = sr_param_cache_indx0;
    assign prim_ff_wr_data[`SC_PI_PC_PTR1]            = sr_param_cache_indx1;
    assign prim_ff_wr_data[`SC_PI_PC_PTR2]            = sr_param_cache_indx2;
    assign prim_ff_wr_data[`SC_PI_IW0]                = sr_i0;
    assign prim_ff_wr_data[`SC_PI_IW_DX]              = sr_ix;
    assign prim_ff_wr_data[`SC_PI_IW_DY]              = sr_iy;
    assign prim_ff_wr_data[`SC_PI_JW0]                = sr_j0;
    assign prim_ff_wr_data[`SC_PI_JW_DX]              = sr_jx;
    assign prim_ff_wr_data[`SC_PI_JW_DY]              = sr_jy;
    assign prim_ff_wr_data[`SC_PI_INVW0]              = sr_w0;
    assign prim_ff_wr_data[`SC_PI_INVW_DX]            = sr_wx;
    assign prim_ff_wr_data[`SC_PI_INVW_DY]            = sr_wy;

//    assign prim_ff_wr_data = {sr_event_id,              // [372:369]  4
//                              sr_dealloc_slot,          //     [368]  1
//                              sr_first_prim_of_slot,    //     [367]  1
//                              sr_end_of_pkt,            //     [366]  1
//                              sr_back_face,             //     [365]  1
//                              sr_provoking_vertex,      // [364:363]  2
//                              sr_state_var_indx,        // [362:360]  3
```

```
//                              sr_prim_type,             // [359:357]  3
//                              sr_ref_x,                 // [356:339] 18
//                              sr_ref_y,                 // [338:321] 18
//                              sr_param_cache_indx0,     // [320:310] 11
//                              sr_param_cache_indx1,     // [309:297] 11
//                              sr_param_cache_indx2,     // [298:288] 11
//                              sr_i0,                    // [287:256] 32
//                              sr_ix,                    // [255:224] 32
//                              sr_iy,                    // [223:192] 32
//                              sr_j0,                    // [191:160] 32
//                              sr_jx,                    // [159:128] 32
//                              sr_jy,                    //  [127:96] 32
//                              sr_w0,                    //   [95:64] 32
//                              sr_wx,                    //   [63:32] 32
//                              sr_wy};                   //    [31:0] 32
                                                          // total = 373 bits

// Concatenate some of outputs of sc_stage_reg to create write data for Z fifo.
    assign z_ff_wr_data[`SC_ZD_NULL_PRIM] = sr_null_prim_zff;
    assign z_ff_wr_data[`SC_ZD_ZY_MAX] = sr_zy_max_zff;
    assign z_ff_wr_data[`SC_ZD_BACK_FACE] = sr_back_face_zff;
    assign z_ff_wr_data[`SC_ZD_PRIM_TYPE] = sr_prim_type_zff;
    assign z_ff_wr_data[`SC_ZD_POLYMODE] = sr_polymode_zff;
    assign z_ff_wr_data[`SC_ZD_STATE_INDX] = sr_state_var_indx_zff;
    assign z_ff_wr_data[`SC_ZD_REF_X] = sr_ref_x_zff;
```

```
    assign z_ff_wr_data[`SC_ZD_REF_Y] = sr_ref_y_zff;
    assign z_ff_wr_data[`SC_ZD_Z_MIN] = sr_z_min_zff;
    assign z_ff_wr_data[`SC_ZD_Z_MAX] = sr_z_max_zff;
    assign z_ff_wr_data[`SC_ZD_Z0] = sr_z0_zff;
    assign z_ff_wr_data[`SC_ZD_ZX] = sr_zx_zff;
    assign z_ff_wr_data[`SC_ZD_ZY] = sr_zy_zff;

//    assign z_ff_wr_data = {sr_polymode,              //     [171]  1
//                           sr_state_var_indx,        // [170:168]  3
//                           sr_ref_x,                 // [167:150] 18
//                           sr_ref_y,                 // [149:132] 18
//                           sr_z_min,                 // [131:118] 14
//                           sr_z_max,                 // [117:104] 14
//                           sr_z0,                    //  [103:72] 32
//                           sr_zx,                    //   [71:36] 36
//                           sr_zy};                   //    [35:0] 36
                                                       // total = 172 bits

// Concatenate outputs of sc_quadmask to create write data for tile fifo.
    assign tile_ff_wr_data[`SC_TD_LAST_TILE]     = qm_last_tile;
    assign tile_ff_wr_data[`SC_TD_ZMASK_NEEDED] = qm_z_mask_needed;
    assign tile_ff_wr_data[`SC_TD_EVENT]         = qm_event;
    assign tile_ff_wr_data[`SC_TD_XMIN]          = qm_xmin;
    assign tile_ff_wr_data[`SC_TD_XMAX]          = qm_xmax;
    assign tile_ff_wr_data[`SC_TD_YMIN]          = qm_ymin;
```

```
    assign tile_ff_wr_data[`SC_TD_YMAX]          = qm_ymax;
    assign tile_ff_wr_data[`SC_TD_BBFRACTBITS]  = qm_bb_fract_bits;
    assign tile_ff_wr_data[`SC_TD_XDIR]          = qm_xdir;
    assign tile_ff_wr_data[`SC_TD_YDIR]          = qm_ydir;
    assign tile_ff_wr_data[`SC_TD_TILEX]         = qm_tilex;
    assign tile_ff_wr_data[`SC_TD_TILEY]         = qm_tiley;
    assign tile_ff_wr_data[`SC_TD_TILEX_M3]      = qm_tilex_m3;
    assign tile_ff_wr_data[`SC_TD_TILEY_M3]      = qm_tiley_m3;
    assign tile_ff_wr_data[`SC_TD_XMAJOR]        = qm_xmajor;
    assign tile_ff_wr_data[`SC_TD_E0_SAMPLE]     = qm_e0;
    assign tile_ff_wr_data[`SC_TD_E1_SAMPLE]     = qm_e1;
    assign tile_ff_wr_data[`SC_TD_E2_SAMPLE]     = qm_e2;
    assign tile_ff_wr_data[`SC_TD_E0_DX]         = qm_dxe0;
    assign tile_ff_wr_data[`SC_TD_E0_DY]         = qm_dye0;
    assign tile_ff_wr_data[`SC_TD_E1_DX]         = qm_dxe1;
    assign tile_ff_wr_data[`SC_TD_E1_DY]         = qm_dye1;
    assign tile_ff_wr_data[`SC_TD_E2_DX]         = qm_dxe2;
    assign tile_ff_wr_data[`SC_TD_E2_DY]         = qm_dye2;
    assign tile_ff_wr_data[`SC_TD_STIPPLE_MASK] = 8'b11111111;

//    assign tile_ff_wr_data = {qm_last_tile,            //     [265]  1
//                              qm_event,                //     [264]  1
//                              qm_xdir,                 //     [263]  1
//                              qm_ydir,                 //     [262]  1
//                              qm_z_mask_needed,        //     [261]  1
//                qm_bb_fract_bits,      // [260:257]  4
//                qm_tilex,              // [256:247] 10
//                qm_tiley,              // [246:237] 10
//                qm_xmin,               // [236:234]  3
//                qm_xmax,               // [233:231]  3
//                qm_ymin,               // [230:228]  3
//                qm_ymax,               // [227:225]  3
//                qm_e0,                 // [224:188] 37
//                qm_e1,                 // [187:151] 37
//                qm_e2,                 // [150:114] 37
//                qm_dxe0,               //  [113:95] 19
//                qm_dye0,               //   [94:76] 19
//                qm_dxe1,               //   [75:57] 19
//                qm_dye1,               //   [56:38] 19
//                qm_dxe2,               //   [37:19] 19
//                qm_dye2};              //    [18:0] 19
                                         // total = 266 bits
// Assign read data of z fifo
  assign zff_null_prim = z_ff_rd_data[`SC_ZD_NULL_PRIM];
  assign zff_zy_max = z_ff_rd_data[`SC_ZD_ZY_MAX];
  assign zff_back_face = z_ff_rd_data[`SC_ZD_BACK_FACE];
  assign zff_prim_type = z_ff_rd_data[`SC_ZD_PRIM_TYPE];
  assign zff_polymode = z_ff_rd_data[`SC_ZD_POLYMODE];
  assign zff_state_var_indx = z_ff_rd_data[`SC_ZD_STATE_INDX];
  assign zff_ref_x = z_ff_rd_data[`SC_ZD_REF_X];
```

```
  assign zff_ref_y = z_ff_rd_data[`SC_ZD_REF_Y];
  assign zff_z_min = z_ff_rd_data[`SC_ZD_Z_MIN];
  assign zff_z_max = z_ff_rd_data[`SC_ZD_Z_MAX];
  assign zff_z0 = z_ff_rd_data[`SC_ZD_Z0];
  assign zff_zx = z_ff_rd_data[`SC_ZD_ZX];
  assign zff_zy = z_ff_rd_data[`SC_ZD_ZY];



// Group aa sample offsets
//   assign st_aa_offset_tbl = {
//         st_sample_15_y,
//         st_sample_15_x,
//         st_sample_14_y,
//         st_sample_14_x,
//         st_sample_13_y,
//         st_sample_13_x,
//         st_sample_12_y,
//         st_sample_12_x,
//         st_sample_11_y,
//         st_sample_11_x,
//         st_sample_10_y,
//         st_sample_10_x,
//         st_sample_9_y,
//         st_sample_9_x,
```

```
//         st_sample_8_y,
//         st_sample_8_x,
//         st_sample_7_y,
//         st_sample_7_x,
//         st_sample_6_y,
//         st_sample_6_x,
//         st_sample_5_y,
//         st_sample_5_x,
//         st_sample_4_y,
//         st_sample_4_x,
//         st_sample_3_y,
//         st_sample_3_x,
//         st_sample_2_y,
//         st_sample_2_x,
//         st_sample_1_y,
//         st_sample_1_x,
//         st_sample_0_y,
//         st_sample_0_x
//   };

endmodule  // sc

```