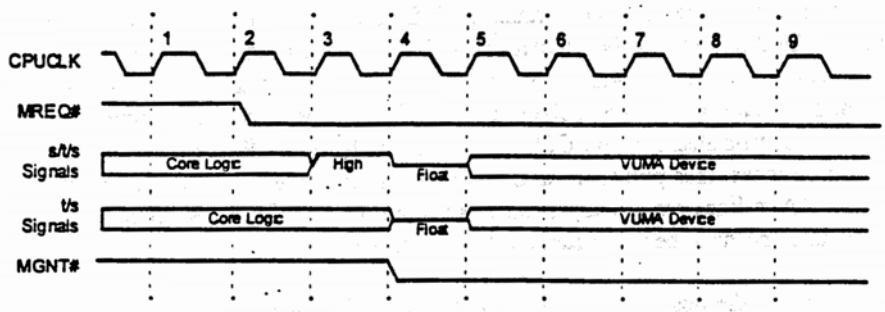
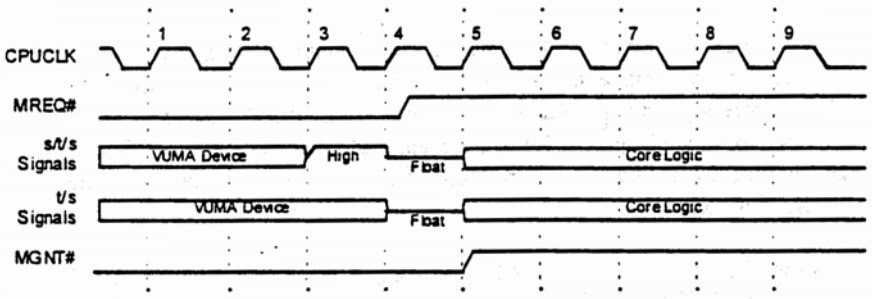


MREQ# is driven low from clock edge 2. Core logic samples it active on clock edge 3. Arbiter can give the bus right away, so core logic drives all s/t/s signals high from the same clock edge. Core logic tri-states all the shared signals (s/t/s and t/s) and drives MGNT# active from clock edge 4 . VUMA device samples MGNT# active at clock edge 5 and starts driving the bus from the same edge.

The shared DRAM signals are driven by VUMA device when it is the owner of the physical system memory bus. VUMA device relinquishes the physical system memory bus by de-asserting MREQ#. Bus Arbiter gives the bus back to core logic by de-asserting MGNT#. Also, as mentioned above, before core logic starts driving the bus, VUMA device should drive the s/t/s signals high for one CPUCLK clock and tri-state them. VUMA device should also tri-state all the shared t/s signals. The float condition on the bus should be for one CPUCLK clock, before core logic starts driving the bus. These activities are overlapped to improve performance as shown in Figure 5-6.

Figure 5-6 Bus hand off from VUMA device to core logic



VUMA device drives all s/t/s signals high from clock edge 3. VUMA device tri-states all shared signals (s/t/s and t/s) and de-asserts MREQ# from clock edge 4. Core logic samples MREQ# inactive on clock edge 5. Core logic drives all shared signals and deasserts MGNT# from clock edge 5.

5.3.2 DRAM Precharge

Petitioners HTC & LG - Exhibit 1002, p. 211

When the physical system memory bus is handed off from core logic to VUMA device or vice a versa, the DRAM needs to be precharged before the new master starts driving it. Part of this precharge can be hidden by overlapping with the arbitration protocol. As shown in Figure 5-5 and 5-6, all the DRAM control signals (including RAS# lines) are driven high for and tri-stated for one CPUCLK clock each. When RAS# lines are tri-stated, pull ups on those lines pull them to a logical high. Thus when a new master gets the control, the RAS# lines are already been precharged for two CPUCLK clocks. The rest of the precharge needs to be taken care by the new master.

1. VUMA device gets the bus from core logic - As shown in Figure 5-5, when VUMA device gets the physical system memory bus at clock edge 5, the DRAM has been precharged for two CPUCLK clocks. VUMA device needs to take care of the rest of the DRAM precharge. This precharge can be overlapped by VUMA device over some of its activity e.g. VUMA device may be running at a different clock than the CPUCLK clock and the precharge can be overlapped with the synchronization of MGNT# signal. VUMA device can calculate the number of clocks it needs to precharge the DRAM with the following formula:

No. of VUMA device clocks for DRAM precharge = {RAS# Precharge Time (tRP) - (2 * CPUCLK Clock Time Period)}/ VUMA device Clock Time Period

Example: CPU running at 66.66 MHz, VUMA device running at 50 MHz. 70ns Fast Page DRAM used.

No. of VUMA device clocks for DRAM precharge = {50ns - (2* 15ns)}/ 20ns
= {20ns}/ 20ns
= 1 clock

2. Core logic gets the bus from VUMA device - As shown in Figure 5-6, when core logic gets the physical system memory bus at clock edge 5, the DRAM has been precharged for two CPUCLK clocks. core logic needs to take care of the rest of the DRAM precharge. This precharge can be overlapped by core logic over some of its activity e.g. driving of new row address. Core logic can calculate the number of clocks it needs to precharge the DRAM with the following formula:

No. of CPU clocks for DRAM precharge = {RAS# Precharge Time (tRP) - (2 * CPUCLK Clock Time Period)}/ CPUCLK Clock Time Period

Example: CPU running at 66.66 MHz, VUMA device running at 50 MHz. 70ns Fast Page DRAM used.

No. of CPU clocks for DRAM precharge = {50ns - (2* 15ns)}/ 15ns
= {20ns}/ 15ns
= 2 clock

Petitioners HTC & LG - Exhibit 1002, p. 212

5.4 Synchronous DRAM

Synchronous DRAM support is optional for both core logic and VUMA device. Various Synchronous DRAM support scenarios are as follows:

1. Core logic does not support Synchronous DRAM - Since core logic does not support Synchronous DRAM, there would not be any Synchronous DRAM as the physical system memory and hence whether VUMA device supports Synchronous DRAM or not is irrelevant.

2. Core logic supports Synchronous DRAM - When core logic supports Synchronous DRAM, VUMA device may or may not be supporting it. Whether core logic and VUMA device support Synchronous DRAM or not should be transparent to the operating system and application programs. To achieve the transparency, system BIOS needs to find out if both core logic and VUMA device support this feature and set the system appropriately at boot. The following algorithm explains how it can be achieved. The algorithm is only included to explain the feature. Refer to the latest VUMA VESA BIOS Extensions for the most updated BIOS calls:

   a. Read <VUMA BIOS signature string (refer to VUMA VESA BIOS Extensions)>. Check if VUMA device supports Synchronous DRAM.
   b. If VUMA device does not support Synchronous DRAM, do not assign Synchronous DRAM banks for Main VUMA Memory. Assign Main VUMA Memory to Fast Page Mode or EDO bank. Also, if Auxiliary VUMA Memory is assigned by operating system to Synchronous DRAM banks, do not use it. Either repeat the request for Auxiliary VUMA Memory till it is assigned to Fast Page Mode or EDO bank or use some alternate method.
   c. If VUMA device supports Synchronous DRAM, read < VUMA BIOS signature string (refer to VUMA VESA BIOS Extensions)> to find out if VUMA device supports multiple banks access.
   d. If only single bank access supported on VUMA device, exit, as the Main VUMA Memory and Auxiliary VUMA Memory bank is fixed.
   e. If multiple banks access is supported and if the CS# for Synchronous DRAM bank is supported on VUMA device, assign the Main VUMA Memory to obtain the best possible system performance and exit.

5.4.1 Programmable Parameters

Synchronous DRAMs have various programmable parameters. Core logic programs Synchronous DRAM parameters to obtain the best possible results. The most efficient way for VUMA device to program its DRAM controller is to make a BIOS call to find

Petitioners HTC & LG - Exhibit 1002, p. 213

out the parameters core logic has decided and program its DRAM controller with the same parameters. Alternately, VUMA device could program its DRAM controller with one or all different parameters. If VUMA device programs its DRAM controller with any different parameters, it is VUMA device's responsibility to reprogram Synchronous DRAM back with the original parameters, before the physical system memory bus is handed off to core logic. In other words, VUMA device is free to change any or all of the parameters, but the change should be transparent to core logic.

How core logic programs various parameters and how VUMA device could inquire them is as follows:

1. Burst Length - Burst Length can be programmed as 1, 2 or 4. VUMA device needs to make a BIOS call <Return Memory Speed Type (refer to VUMA VESA BIOS Extensions)> to find out the Burst Length.

2. CAS Latency - As CAS latency depends on the speed of Synchronous DRAM used and the clock speed, this standard does not want to fix this parameter. Core logic programs this parameter to an appropriate value. VUMA device needs to make a BIOS call <Return Memory Speed Type (refer to VUMA VESA BIOS Extensions)> to find out the CAS latency.

3. Burst Ordering - Most efficient Burst Ordering depends upon the type of CPU used. VUMA device needs to make a BIOS call <Return Memory Speed Type (refer to VUMA VESA BIOS Extensions)> to find out the Burst Order.

5.4.2 Protocol Description and Timing

All the DRAM signals are shared by core logic and VUMA device. They are driven by current bus master. When core logic and VUMA device hand over the bus to each other, they must drive all the shared s/t/s signals high for one CPUCLK clock and then tri-state them. Also, they should tri-state all the shared t/s signals.

Synchronous DRAMs are precharged by precharge command. When the physical system memory bus is handed off from core logic to VUMA device or vice a versa, the DRAM precharge has two options:

1. Precharge both the internal banks before hand-off - This is a simple case where both the internal banks of the active synchronous DRAM bank are precharged and then the bus is handed off.
2. Requesting Master snoops the physical system memory bus and synchronous DRAM internal banks need not be precharged - In this case the requesting master snoops the DRAM address and control signals to track the open pages in the internal banks of the active synchronous DRAM bank. The internal banks of the active synchronous DRAM

Petitioners HTC & LG - Exhibit 1002, p. 214

are not precharged when the physical system memory bus is handed-off to the requesting master. If needed, the requesting master takes care of precharge after getting the physical system memory bus.

Both core logic and VUMA device have an option of either implementing or not implementing DRAM snoop feature. Whether core logic and VUMA device support DRAM snoop or not should be transparent to the operating system and application programs. To achieve the transparency, system BIOS and VUMA BIOS need to find out if both core logic and VUMA device support this feature and set the system appropriately at boot. The following algorithm explains how it can be achieved. The algorithm is only included to explain the feature. Refer to the latest VUMA VESA BIOS Extensions for the most updated BIOS calls:

1. System BIOS reads <VUMA BIOS signature string (refer to VUMA VESA BIOS Extensions)>, to find out if VUMA device can snoop the physical system memory bus.
2. If no, System BIOS programs core logic to precharge synchronous DRAM before bus hand-off.
3. If yes, System BIOS programs core logic not to precharge synchronous DRAM before bus hand-off.
4. VUMA BIOS makes a call, <Report VUMA - core logic capabilities (refer to VUMA VESA BIOS Extensions)>, to find out if core logic can snoop the physical system memory bus.
5. If no, VUMA BIOS programs VUMA device to precharge synchronous DRAM before bus hand-off.
6. If yes, VUMA BIOS programs VUMA device not to precharge synchronous DRAM before bus hand-off.

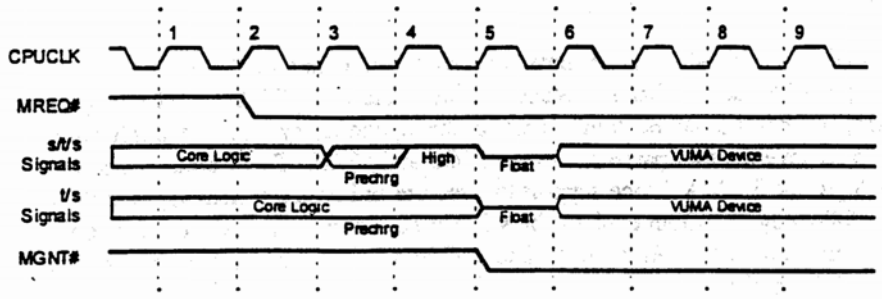
None, only one, or both of core logic and VUMA device can support this feature. When only one of them supports this feature memory precharge will be asymmetrical i.e. there will be precharge before hand-off one way and no precharge the other way.

5.4.2.1 Non-Snoop Cases

The shared DRAM signals are driven by core logic when it is the owner of the physical system memory bus. VUMA device requests the physical system memory bus by asserting MREQ#. Bus Arbiter grants the bus by asserting MGNT#. Also, before VUMA device starts driving the bus, core logic should drive all the shared s/t/s signals high for one CPUCLK clock and tri-state them. Core logic should also tri-state all the shared t/s signals. The tri-state condition on the bus should be for one CPUCLK clock, before VUMA device starts driving the bus. These activities are overlapped to improve performance as shown in Figure 5-7. Since VUMA device does not support DRAM snoop feature, DRAM is precharged before handing off the physical system memory bus as shown in Figure 5-7.

Petitioners HTC & LG - Exhibit 1002, p. 215

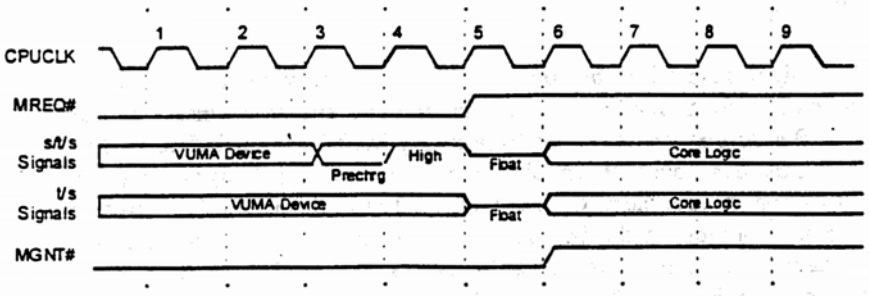
Figure 5-7 Bus hand off from Core Logic to VUMA device



MREQ# is driven low from clock edge 2. Core logic samples it active on clock edge 3. Arbiter can give the bus right away, so core logic gives precharge command to DRAM from the same clock edge. Core logic drives all the shared s/t/s signals high from clock edge 4. Core logic tri-states all the shared signals (s/t/s and t/s) and drives MGNT# active from clock edge 5. VUMA device samples MGNT# active at clock edge 6 and starts driving the bus from the same edge.

The shared DRAM signals are driven by VUMA device when it is the owner of the physical system memory bus. VUMA device relinquishes the physical system memory bus by de-asserting MREQ#. Bus Arbiter gives the bus back to core logic by de-asserting MGNT#. Also, as mentioned above, before core logic starts driving the bus, VUMA device should drive all the shared s/t/s signals high for one CPUCLK clock and tri-state them. VUMA device should also tri-state all the shared t/s signals. The float condition on the bus should be for one CPUCLK clock, before core logic starts driving the bus. These activities are overlapped to improve performance as shown in Figure 5-8. Since core logic does not support DRAM snoop feature, DRAM is precharged before handing off the physical system memory bus as shown in Figure 5-8.

Figure 5-8 Bus hand off from VUMA device to Core Logic



VUMA device gives precharge command from clock edge 3. It drives all shared s/t/s signals high from clock edge 4. It tri-states all shared signals (s/t/s and t/s) and de-asserts

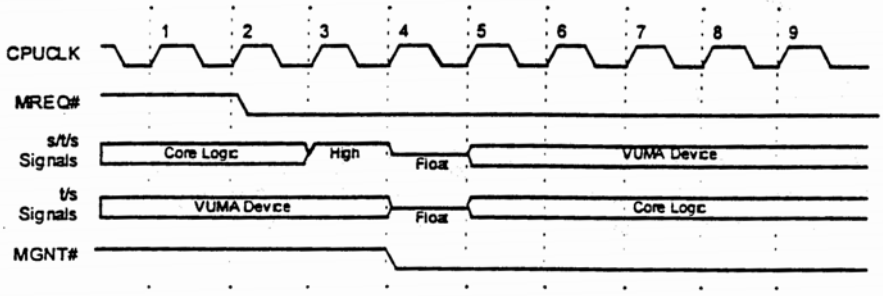
Petitioners HTC & LG - Exhibit 1002, p. 216

MREQ# from clock edge 5. Core logic samples MREQ# inactive on clock edge 6. Core logic drives all shared signals and deasserts MGNT# from clock edge 6.

5.4.2.2 Snoop Cases

The shared DRAM signals are driven by core logic when it is the owner of the physical system memory bus. VUMA device requests the physical system memory bus by asserting MREQ#. Bus Arbiter grants the bus by asserting MGNT#. Also, before VUMA device starts driving the bus, core logic should drive all the shared s/t/s signals high for one CPUCLK clock and tri-state them. Core logic should also tri-state all the shared t/s signals. The tri-state condition on the bus should be for one CPUCLK clock, before VUMA device starts driving the bus. These activities are overlapped to improve performance as shown in Figure 5-9. Since VUMA device supports DRAM snoop feature, core logic does not precharge DRAM before handing off the physical system memory bus as shown in Figure 5-9.

Figure 5-9 Bus hand off from core logic to VUMA device



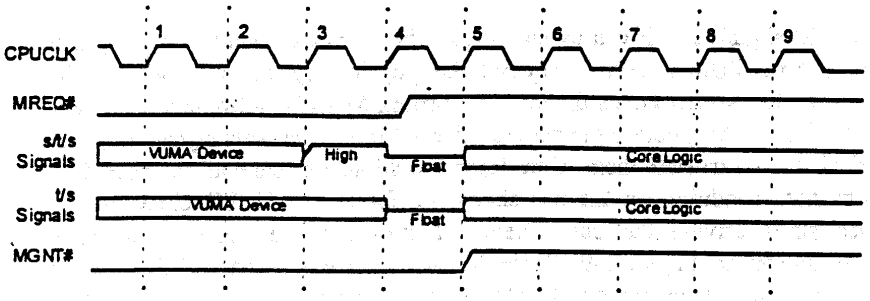
MREQ# is driven low from clock edge 2. Core logic samples it active on clock edge 3. Arbiter can give the bus right away and since VUMA device supports DRAM snoop feature, core logic drives all the shared s/t/s signals high from the same clock edge. Core logic tri-states all the shared signals (s/t/s and t/s) and drives MGNT# active from clock edge 4. VUMA device samples MGNT# active at clock edge 5 and starts driving the bus from the same edge.

The shared DRAM signals are driven by VUMA device when it is the owner of the physical system memory bus. VUMA device relinquishes the physical system memory bus by de-asserting MREQ#. Bus Arbiter gives the bus back to core logic by de-asserting MGNT#. Also, as mentioned above, before core logic starts driving the bus, VUMA device should drive all the shared s/t/s signals high for one CPUCLK clock and tri-state them. VUMA device should also tri-state all the shared t/s signals. The float condition on the bus should be for one CPUCLK clock, before core logic starts driving the bus. These activities are overlapped to improve performance as shown in Figure 5-10. Since core

Petitioners HTC & LG - Exhibit 1002, p. 217

logic supports DRAM snoop feature, VUMA device does not precharge DRAM before handing off the physical system memory bus as shown in Figure 5-10.

Figure 5-10 Bus hand off from VUMA device to core logic



VUMA device drives all shared s/t/s signals high from clock edge 3. It tri-states all shared signals (s/t/s and t/s) and de-asserts MREQ# from clock edge 4. Core logic samples MREQ# inactive on clock edge 5. Core logic drives all shared signals and deasserts MGNT# from clock edge 5.

5.5 Memory Parity support

Memory Parity support is optional on both core logic and VUMA device. If core logic supports parity it should be able to disable parity check for Main VUMA Memory and Auxiliary VUMA Memory areas while parity check on the rest of the physical system memory is enabled.

5.6 Memory Controller Pin Multiplexing

The logical interfaces for Fast Page, EDO and BEDO DRAMs are very similar but are significantly different than that of Synchronous DRAM. If mother board designers want to mix different DRAM technologies on the same mother board, core logic will have to multiplex DRAM control signals. The meaning of a multiplexed signal will depend on the type of DRAM core logic is accessing at a given time. If a VUMA device supports multiple banks access and mix of different DRAM technologies, it will also have to multiplex DRAM control signals. Both core logic and VUMA devices will have to have same multiplexing scheme. The appropriate JEDEC standard should be followed for multiplexing scheme.

6.0 Boot Protocol

Petitioners HTC & LG - Exhibit 1002, p. 218

6.1 Main VUMA Memory Access at Boot

In unified memory architecture, part of the physical system memory is assigned to Main VUMA Memory. The existing operating systems are not aware of unified memory architecture. Also, some of the existing operating systems size memory themselves. This poses a problem as the operating systems after sizing the total physical system memory, will assume that they could use all of the memory and might overwrite Main VUMA Memory. The solution to this problem is explained below:

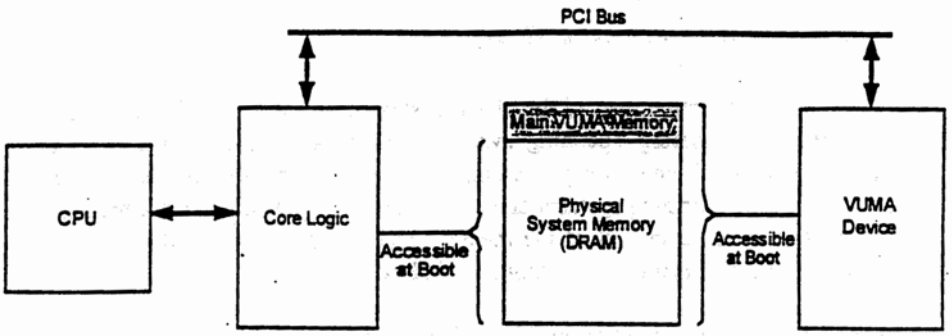
As shown in Figure 6-1, the solution to this problem is to disable core logic access to Main VUMA Memory area at boot time. In that case even if operating system, sizes the memory, it will find only (total physical system memory - Main VUMA Memory) and will not be aware of the Main VUMA Memory existence. This will avoid operating system ever writing to the Main VUMA Memory area. If VUMA device supports multiple banks access, it can access total physical system memory all the time. If VUMA device supports single bank access, it can access the bank of Main VUMA Memory all the time.

If VUMA device is a graphics controller, it needs a special consideration. Video screen is required during boot and since core logic can not access the Main VUMA Memory, it can not write to it. The problem is solved by programming the graphics controller into a pseudo legacy mode. In this mode graphics controller treats Main VUMA Memory exactly the same way as in non unified memory architecture situations i.e. as if it has its own separate frame buffer. So now, the total system looks just like a non unified memory architecture system and this mode is called as pseudo legacy mode. Core logic performs accesses to video through legacy video memory address space of A000:0 and B000:0. These accesses go on the PCI bus. Graphics controller claims these cycles. Graphics controller still needs to arbitrate for the physical system memory bus. After getting the bus, graphics controller performs reads/writes to Main VUMA Memory (frame buffer). After the system boots, it is still in the pseudo legacy mode. When operating system calls display driver, the driver programs core logic to allow access to Main VUMA Memory and switches the system from pseudo legacy mode to unified memory architecture.

In the case of other type of VUMA devices, device driver needs to program core logic to allow access to Main VUMA Memory.

Figure 6-1 Pseudo Legacy Mode

Petitioners HTC & LG - Exhibit 1002, p. 219



The following algorithm sums up the boot process in the case of VUMA device being a graphics controller:

1. System BIOS sizes the physical system memory.
2. System BIOS reads the size of Main VUMA Memory at previous boot (where this value is stored is System BIOS dependent, but needs to be in some sort of non volatile memory).
3. System BIOS programs its internal registers to reflect that total memory available is [total physical system memory(from step 1) - Main VUMA Memory at previous boot (from step 2)].
4. System boots and operating system calls display driver.
5. Display driver makes a System BIOS call, <Enable/Disable Main VUMA Memory (refer to VUMA VESA BIOS Extensions)>, to program core logic internal registers to reflect that it can access total physical system memory.
6. Display driver switches VUMA device to unified memory architecture mode.

Even though core logic can not access Main VUMA Memory till the time display driver enables it, core logic is responsible for Main VUMA Memory refresh.

VUMA device should claim PCI Master accesses to Main VUMA Memory till display driver enables core logic access to that area. Core logic should claim PCI Master accesses to Main VUMA Memory after display driver enables core logic access to that area.

6.2 Reset State

On power on reset, both core logic and VUMA device have their unified memory architecture capabilities disabled. MREQ# is de-asserted by VUMA device and MGNT# is de-asserted by core logic. System BIOS can detect if VUMA device supports unified memory architecture capabilities by reading <VUMA BIOS signature string (refer to VUMA VESA BIOS Extensions)>.

Petitioners HTC & LG - Exhibit 1002, p. 220

7.0 Electrical Specification

7.1 Signal Levels

This section describes the electrical signal levels for the arbitration signals only. DRAM signal levels depend on the type of DRAM used and hence can not be specified by the standard.

MREQ#	**output**	5v TTL or 3.3v LVTTL
	input	5v TTL for 5v buffer, 5v tolerant LVTTL for 3.3v buffer
MGNT#	**output**	5v TTL or 3.3v LVTTL
	input	5v TTL for 5v buffer, 5v tolerant LVTTL for 3.3v buffer
CPUCLK	**output**	5v TTL or 3.3v LVTTL
	input	5v TTL for 5v buffer, 5v tolerant LVTTL for 3.3v buffer

7.2 AC Timing

This section describes the AC timing parameters for the arbitration signals only. DRAM AC timing parameters depend on the type of DRAM used and hence can not be specified by the standard. Both MREQ# are MGNT# timing parameters are with respect to CPUCLK rising edge.

MREQ#	**output**	tClk to Out (max) - 10 ns
		tClk to Out (min) - 2 ns
	input	Set up time tSU (min) - 3 ns
		Hold time tH (min) - 0 ns
MGNT#	**output**	tClk to Out (max) - 10 ns
		tClk to Out (min) - 2 ns
	input	Set up time tSU (min) - 3 ns
		Hold time tH (min) - 0 ns
CPUCLK	**output**	clock frequency (max) - 66.66 MHz

7.2.1 Timing Budget

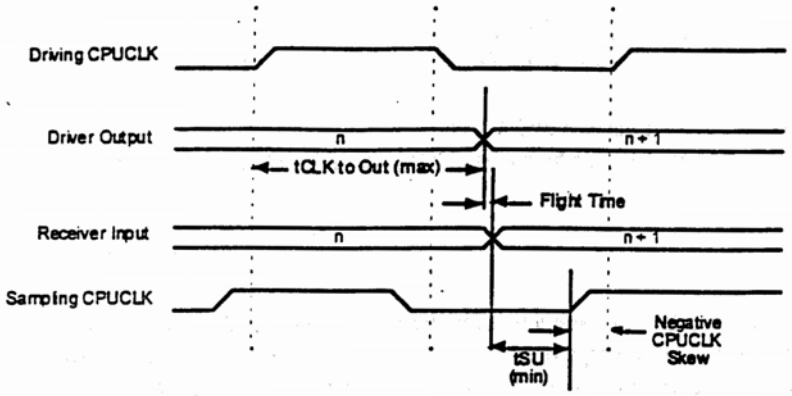
A margin for signal flight time and clock skew is added to the timing parameters. ± 2ns is allowed for the total of CPUCLK skew and signal flight time. Worst case timing budget calculations for setup and hold time are as follows:

Petitioners HTC & LG - Exhibit 1002, p. 221

7.2.1.1 Worst case for Setup time

Figure 7-1 shows the worst case for setup time. tClk to Out, flight time and clock skew have converged to reduce available setup time.

Figure 7-1 Worst case for setup time

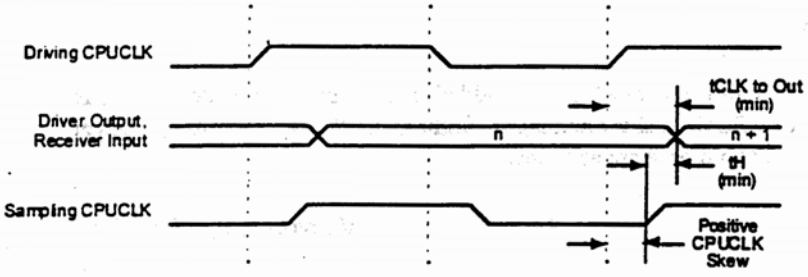


[tCLK to Out (max) + flight time + tSU (min) + negative CPUCLK skew] ≤ CPUCLK period i.e. 15ns @ 66.66 MHz.
[10ns + flight time + 3ns + negative CPUCLK skew] ≤ 15ns
[flight time + negative CPUCLK skew] ≤ 2ns

7.2.1.2 Worst case for Hold time

Figure 7-2 shows the worst case for hold time. tClk to Out and clock skew have converged to reduce available hold time. Positive flight time number helps in this case and hence it is assumed to be zero.

Figure 7-2 Worst case for hold time



Petitioners HTC & LG - Exhibit 1002, p. 222

[positive CPUCLK skew + tH (min)] ≤ tCLK to Out (min)
[positive CPUCLK skew + 0ns] ≤ 2ns
positive CPUCLK skew ≤ 2ns

7.3 Pullups

All s/t/s signals need pullups to sustain the inactive state until another agent drives them. Core logic has to provide pullups for all the s/t/s signals. VUMA device has as option of providing pullups on some of the s/t/s signals. All t/s signals need pulldowns. Core logic has to provide pulldowns for all the t/s signals. VUMA device has as option of providing pulldowns on the t/s signals. Pullups and pulldowns could either be internal to the chips or external on board.

DRAM Address - Core logic is responsible for pullups on DRAM Address lines.
DRAM control signals - Core logic is responsible for pullups on DRAM control signals. VUMA device has as option of providing pullups on them.
DRAM Data Bus - Core logic is responsible for pulldowns on DRAM data bus. VUMA device has as option of providing pulldowns on them.

Pullups and pulldowns are used to sustain the inactive state until another agent drives the signals and hence need to be weak. Recommended value for pullups and pulldowns is between 50 kohm and 80 kohm.

7.4 Straps

As some VUMA devices and core logic chips use DRAM data bus for straps, DRAM data bus needs to be assigned for straps for different controllers. The assignment of DRAM Data Bus for straps is as follows:

MD [0:19] VUMA device on Motherboard
MD [20:55] Reserved
MD [56:63] Core Logic

All the straps need to be pullups of 10 kohm.

7.5 DRAM Driver characteristics

Loading plays a critical role in DRAM access timing. In case of PC motherboards end users can expand the existing memory of a system by adding extra SIMMs. Hence, typically the total DRAM signal loading is not constant and could vary significantly. Both Core Logic and VUMA device must be able to drive the maximum load that the

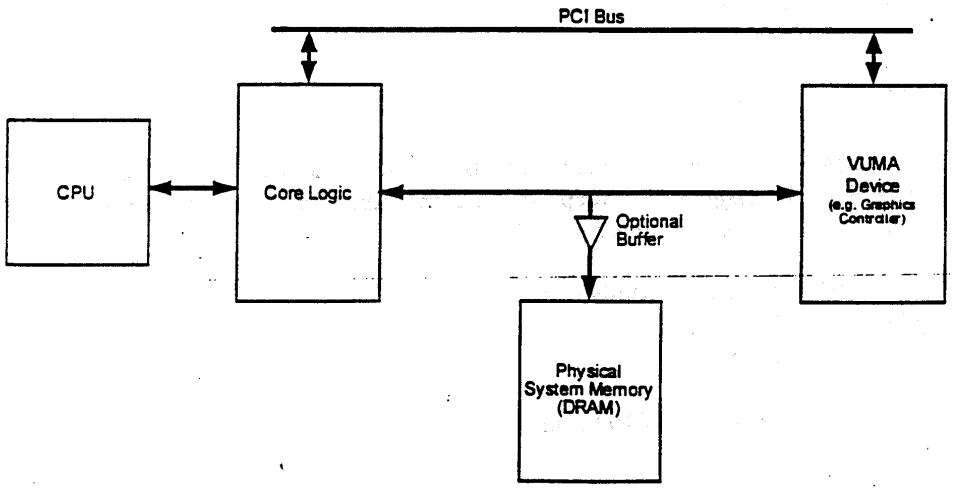
Petitioners HTC & LG - Exhibit 1002, p. 223

system motherboard is designed to accommodate. In typical motherboard designs DRAM signal loading can be excessive (on the order of 1000pF for some signals) and hence care must be taken for DRAM driver selection. Some general guide lines for DRAM driver design are as follows:

Slew-rate controlled drivers are recommended. Drivers with selectable current drive (such as 8/16 mA drivers) may be used. This can reduce overshoot and undershoot associated with over-driving lightly loaded signals and can prevent excessive rise and fall time delay due to not providing enough current drive on heavily loaded signals.

As shown in Figure 7-3, buffers may be placed on the system motherboard to reduce the per signal loading and/or provide larger drive strength capabilities. DRAM Write Enable and DRAM Address signals are typically the most heavily loaded signals. Column Address Strobe signals may also become overloaded when more than two DRAM banks are designed into a system. TTL or CMOS buffers (typically 244 type) may be used to isolate and duplicate heavily loaded signals on a per bank basis. 244 type buffers typically have very good drive characteristics as well and can be used to drive all of the heavily loaded DRAM control signals if the Core Logic and/or VUMA device has relatively weak drive characteristics. If external buffers are used, the buffer delays should be taken in to timing considerations.

Figure 7-3 Optional Buffers for DRAM Signals



Wider DRAM devices offer reduced system loading on some of the control signals. x4 DRAMs require four times the physical connections on RAS, MA (Address), and write enables as x16 DRAMs. The reduction in loading can be significant. If the designer has control over the DRAMs which will be used in the system, the DRAM width should be chosen to provide the least loading.

Petitioners HTC & LG - Exhibit 1002, p. 224

VESA®

VUMA Proposal

(Draft)

Video Electronics Standards Association

2150 North First Street, Suite 440
San Jose, CA 95131-2029

Phone: (408) 435-0333
FAX: (408) 435-8225

VESA Unified Memory Architecture
VESA BIOS Extensions (VUMA-SBE)
Proposal

Version: 1.0
Document Revision: 2.2p
November 1, 1995

Important Notice: This is a **draft** document from the Video Electronics Standards Association (VESA) Unified Memory Architecture Committee (VUMA). It is only for discussion purposes within the committee and with any other persons or organizations that the committee has determined should be invited to review or otherwise contribute to it. It has not been presented or ratified by the VESA general membership.

Purpose

To allow the video BIOS and other GUI specific software to control the VUMA hardware without specific knowledge or direct hardware access.

Summary

This document contains a specification for a system and video BIOS interface, VUMA-SVBE. The VUMA-SVBE interface will allow the video BIOS and other GUI specific software to control the VUMA hardware without specific knowledge or direct hardware access. The hardware protocol is described in VESA document VUMA 1.0.

Petitioners HTC & LG - Exhibit 1002, p. 225

Scope

Because this is a draft document, it cannot be considered complete or accurate in all respects although every effort has been made to minimize errors.

Intellectual Property

© Copyright 1995 – Video Electronics Standards Association. Duplication of this document within VESA member companies for review purposes is permitted. All other rights are reserved.

Trademarks

All trademarks used in this document are the property of their respective owners. VESA and VUMA are trademarks owned by the Video Electronics Standards Association.

Patents

The proposals and standards developed and adopted by VESA are intended to promote uniformity and economies of scale in the video electronics industry. VESA strives for standards that will benefit both the industry and end users of video electronics products. VESA cannot ensure that the adoption of a standard; the use of a method described as a standard: or the making, using, or selling of a product in compliance with the standard does not infringe upon the intellectual property rights (including patents, trademarks, and copyrights) of others. VESA, therefore, makes no warranties, expressed or implied, that products conforming to a VESA standard do not infringe on the intellectual property rights of others, and accepts no liability direct, indirect or consequential, for any such infringement.

Petitioners HTC & LG - Exhibit 1002, p. 226

Support For This Specification

If you have a product that incorporates VUMA™, you should ask the company that manufactured your product for assistance. If you are a manufacturer of the product, VESA can assist you with any clarification that you may require. All questions must be sent in writing to VESA via:

(The following list is the preferred order for contacting VESA.)

VESA World Wide Web Page:	*www.vesa.org*
Fax:	(408) 435-8225
Mail:	VESA 2150 North First Street Suite 440 San Jose, California 95131-2029

Acknowledgments

This document would not have been possible without the efforts of the members of the 1995 VESA Unified Memory Architecture Committee and the professional support of the VESA staff.

Work Group Members

Any industry standard requires information from many sources. The following list recognizes members of the VUMA Committee, which was responsible for combining all of the industry input into this proposal.

VUMA Chairperson

Rajesh Shakkarwar	OPTi

Software work group Members

Tim Crawford	Work group leader, Cirrus Logic
Phil Mummah	Phoenix Technologies
Josh Grossman	S3, Inc.
Christopher Rhodes	Award Software

Petitioners HTC & LG - Exhibit 1002, p. 227

Revision History

Initial Revision	Aug 14 '95
Rev .7	Aug 24 '95
Rev .8 Removed segment registers, modified Boot sequence wording.	Aug 28 '95
Rev .9 Added assumption F, modified issues and moved some to assumptions, remove references to main and aux VUMA memory, removed function for aux VUMA memory, modified speed/type function	Aug 28 '95
Rev 1.0 Added VUMA device DRAM support, SDRAM parameters in function 6	Sept 8 '95
Rev 1.1 Add items to Boot sequence Remove a function call and cleaned up others	Sept 11 '95
Rev 1.2 Add changes suggested in previous VESA meeting	Sept 22 '95
Rev 1.3 Added 32 bit I/F, added Aux functions, Added 16 bit protected mode I/F, Modified several functions	Sept 29 '95
Rev 1.4 Modified assumptions, issues, updated some functions.	Oct 6 '95
Rev 1.5 Modified the table of contents	Oct 6 '95
Rev 1.6 Minor modifications to assumptions	Oct 10 '95
Rev 1.7 Changes to assumptions, & goals. Made chages to some functions per discusion at last committee meeting.	Oct 16 '95
Rev 1.8 Made changes to how memory is reported, funcs 1 & 6. Modified ROM signature.	Oct 18 '95
Rev 1.9 Modified function 6 and assumptions. Other minor typos fixed.	Oct 18 '95
Rev 2.0 Modified assumptions C, J, L, N. Added assumptions O, P, Q, R. Removed issue 4. Added error code 5, 6. Changed unit of memory in ROM signature from 1K to 64K. Modified register definition in function 0, 1, 2, 6, and 7. Other minor typos fixed.	Oct 23 '95
Rev 2.1 Added assumption "s".	Oct 24 '95
Rev 2.2 Added requirement for 32 and 16 bit protected mode call stack selector and IO requirements. Removed issue section. Updated 32-bit and 16 bit interface section. Modified assumption l, o, s. Rewrote sec. 3.1, 3.2.. Modified sec.3.4, 3.5.0,3.5.1, 3.5.6,	Nov 1 '95

Petitioners HTC & LG - Exhibit 1002, p. 228

3.5.8.

Petitioners HTC & LG - Exhibit 1002, p. 229

TABLE OF CONTENTS

Petitioners HTC & LG - Exhibit 1002, p. 230

1.0 Introduction

This document contains specifications for VESA unified memory architecture system and video BIOS interface. The system BIOS VUMA-SBE (System BIOS Extensions) will allow the video BIOS and other GUI specific software to control the VUMA hardware without specific knowledge or direct hardware access. The video BIOS VUMA-VBE (Video BIOS Extensions) will allow the system BIOS and other GUI specific software to access the VUMA hardware without specific knowledge or direct hardware access. The hardware protocol is described in VESA document VUMA 1.0.

Readers of this document should already be familiar with the VESA BIOS extensions and programming at the BIOS level.

2.0 Goals and Assumptions

VUMA-SBE provides a hardware independent means for operating system and configuration utility software to control and get status from the VUMA hardware.

VUMA-SBE services need to be provided as part of the system and video BIOS ROMs since the functions need to be used during system boot up.

2.1 Goals

a. Allow system memory access to non system controller devices. These devices, called VUMA devices will have their own memory controller and access system memory directly. All of system memory is potentially accessible by VUMA devices.

b. Allow multiple devices. Although only one connector is allowed, multiple devices on the motherboard as well as multiple devices on the expansion board are allowed.

c. If a VUMA device that previously has requested memory is taken out of the system, the memory will be returned to the O/S on the next boot.

d. If a VUMA device is replaced by another VUMA device, the system will allocate the same amount of memory, if it meets the minimum requirements of the new board. Otherwise the memory allocated will be increased to the minimum required by the new board.

Petitioners HTC & LG - Exhibit 1002, p. 231

2.2 Assumptions

a. System BIOS will manage memory allocation requests from VUMA devices.
b. Memory must be sized, typed and contiguous before control is turned over to a VUMA device.
c. VUMA devices will test and initialize their own Main VUMA memory. (This is similar to the way they initialize and test their video memory on conventional VGA devices.)
e. The lowest PCI PFA number will have priority if more than one VGA device is plugged in. The manufacture can decide if the VUMA slot has the highest, lowest or middle PFA number.
f. The Video BIOS must be in shadow ram and writeable when control is passed to the video ROM as defined by the PCI SIG.
g. The values that BIOS reports in function 6 for current, voltage, and speed will be determined at build/compile time.
h. A device driver should insure that when requesting VUMA memory for the next boot that enough memory will be left for the O/S to boot.
i. The device driver should take into consideration memory bandwidth when requesting memory.
j. Memory is installed on the mother board in the bank or banks (RAS/CS) that the VUMA device can access. If a user moves memory to a bank (RAS/CS) that the VUMA controller can not access, the VUMA device will be disabled.
k. On a warm boot the sytem will reallocate VUMA memory for each device.
l. For a multi-function plug in board, only function 0 on the board may require a minimum amount of memory for booting. See section 3.4, point A. Set next boot size call (VUMA-SBE function 2) can only be made using the PFA of function 0 on the board.
m. If a plug in card has a bridge, only the first function of the first device behind the bridge may require a minimum amount of memory for booting. See section 3.4, point A.
n. System BIOS will insure that PCI addresses will not conflict with Main VUMA memory that is placed above system memory. Main VUMA memory could have addresses that are not contiguous with system memory. (See h/w spec.)
o Main VUMA memory that is contiguous to system memory must be disabled before OS boots.
p. VUMA device driver is responsible for enabling CPU access to Main VUMA memory. Note. all of Main VUMA memory access by the CPU is enabled when any part is enabled, i.e.. all or nothing. Disabling CPU access is not allowed at run time.
q. The Main VUMA memory must be contiguous, but it is not necessary to be contiguous with system memory.
r. If Main VUMA memory is not contiguous with system memory, CPU access does not need to be disabled prior to INT 19h.
s. When requesting Aux VUMA memory, if system memory is being cache by any type of cache. the cache must be cleared by an I/O instruction, not by reading memory. This is necessary since in protected mode a selector will not be available to the BIOS.

Petitioners HTC & LG - Exhibit 1002, p. 232

Petitioners HTC & LG - Exhibit 1002, p. 233

2.3 Boot Sequence

1. System BIOS sizes and configures (makes contiguous) all system memory.

2. System BIOS scans ROM space for VUMA devices and determine if there are any devices present that were not present at the last boot. If yes, then add the minimum amount of memory required by that device for booting, to VUMA memory.

3. Allocate VUMA memory. At this point all memory, including VUMA memory is enabled. If this is not possible to allocate all requested memory (possibly memory has been removed between boots) then the system will scan all VUMA ROMs and allocate the minimum necessary to boot.

4. Next, call the entry point to the VGA device. The VGA device tests and initializes it's memory at this time.

5. After the VGA device has initialized itself, control is given back to the system BIOS.

6. System BIOS then continues POST. During POST the system gives control to the other PCI devices (including VUMA devices). They then initialize themselves.

7. When the OS starts it's boot process, it will then load and execute the video driver. If necessary the video driver will then enable the CPU access to memory allocated to the VUMA device.

8. Any changes to the size of the memory allocated to the VUMA device will be requested by the video driver, O/S, or utility/properity sheet. These requests will then be implemented on the next boot.

Petitioners HTC & LG - Exhibit 1002, p. 234

3.0 VUMA VESA SYSTEM BIOS Extensions (VUMA-SBE)

The new system BIOS calls have that have been defined can be accessed via the following VUMA-SBE interfaces.

3.1 VUMA-SBE 32 bit interface.

Detecting the presence of the 32-bit interface for the VUMA-SBE functions is done using the BIOS32 Service Directory[1]. Use of the service directory involves 3 steps : locating the service directory, using the service directory to get the VUMA services entry point and finally calling the VUMA services to perform the desired function. The BIOS32 Service Directory is a contigous 16-byte data structure which begins on a 16-byte boundary somewhere in the physical address range 0E0000h - 0FFFFFh. It has the following format :

Offset	Size	Description
00h	4 bytes	ASCII Signature String _32_ This puts an underscore at offset 0, a '3' at offset 1, a '2' at offset 2 and another underscore offset 3.
04h	4 bytes	Entry point for BIOS32 Service directory This is a 32-bit physical address through which the service directory can be called.
08h	1 byte	Revision level The current revision level is 00h.
09h	1 byte	Length of data structure in paragraph (i.e., 16-byte) units. The data structure in this revision is 16 bytes long so this field has a value of 01h.
0Ah	1 byte	Checksum This field is a checksum of the complete data structure. It has a value such that when all of the bytes in the data structure are added together in a byte wide sum they add up to 00h.
0Bh	5 bytes	Reserved Must be 0

To locate the service directory a caller must scan 0E0000h to 0FFFFFh on 16-byte boundaries looking for the ASCII signature "_32_" and a valid checksummed data structure. If the service directory is NOT found then 32-bit VUMA support is not present in the BIOS.

[1] The BIOS32 Service Directory is an industry standard and is described by the document **Standard BIOS 32-bit Service Directory Proposal**, Revision 1.0 May24, 1993 available from Phoenix Technologies Ltd., Irvine, CA

Petitioners HTC & LG - Exhibit 1002, p. 235

To get the VUMA services entry point a CALL FAR to service directory is done using the value specified at offset 04h in the service directory data structure. The following is a list of the entry conditions and the return values when calling the service directory.

INPUT:

EAX	Service Identifier This is a 4 character string used to identify which 32-bit BIOS service is being sought. For VUMA is it "VUMA" where EAX = 414D5556h (NOTE: This corresponds to mov eax, 'VUMA')
EBX[31:8]	Reserved Must be set to 00h
EBX[7:0]	00h
CS	Code selector set up to encompass the physical page holding the entry point as well as the immediately following physical page. It MUST have the same base. CS is execute only.
DS	Data selector set up to encompass the physical page holding the entry point as well as the immediately following physical page. It MUST have the same base. DS is read-only.
SS	Stack selector must provide at least 1K of stack space and be 32-bit.
I/O	I/O Permissions must be provided so that the BIOS can perform any I/O necessary.

OUTPUT:

AL	Return Code 00h - Requested service is present. 80h - Requested service is NOT present. 81h - Unimplemented function specified in BL.
EBX	Physical address to use as the selector BASE for the service.
ECX	Value to use as the selector LIMIT for the service.
EDX	Entry point for the service relative to the BASE returned in EBX.

Once the VUMA entry point has been found the caller should create an execute-only CODE and read-only DATA selectors based on the values in EBX and ECX. The VUMA entry point can now be invoked using a CALL FAR with the created CODE selector and the offset in EDX. The following additional conditions must exist when calling the VUMA entry point :

Petitioners HTC & LG - Exhibit 1002, p. 236

- I/O permissions are such that the service can perform any I/O necessary
- The stack is a 32-bit stack and provides at least 1K of stack space
- The appropriate privilege is provided so that the service can enable/disable interrupts are needed

All other register settings are specified to the function being called.

3.2 VUMA-SBE 16 bit interface.

The 16 bit Interface is function-based and all parameters are passed in registers. If a register is not specified as an output parameter for a function, then it will be preserved. All flags are preserved. Function values are passed as input parameters in register BL. Return status is passed back in register AL. A return status of 00h indicates that the function was successful.

Prior to calling into the 16-bit interface in protected mode using the PUSHF / CALL sequence the following requirements must be met :

- CS is an execute-only selector with a BASE of 0F0000h and a LIMIT of 64K.
- DS is a read-only selector with a BASE of 0F0000h and a LIMIT of 64K.
- I/O permissions are such that the service can perform any I/O necessary
- The stack is a 16-bit stack and provides at least 1K of stack space
- The appropriate privilege is provided so that the service can enable/disable interrupts are needed

Entry to the 16 bit interface may be done one of two ways:

1. Entry point to the 16 bit services is F000:F859. To call these services:
   Set up the registers as indicated in the function description. Status information is returned in AX.

   ```
   PUSHF
   CALL FAR F000:F859
   Check results
   ```

2. The 16 bit interface may also be accessed through the INT 15h instruction. The value F401h is passed in the AX register, with the subfunction passed in BL. Status information is returned in AX.

Petitioners HTC & LG - Exhibit 1002, p. 237

3.3 Status Information for the calls

Every function returns information in the AX register. The format of the status word is as follows:

AL = F4h: Function is supported.
AL = FFh: No error, but function **NOT** supported.
AL = 00h: Function error or not completed yet, see error codes in AH.

AH = 00h: Function call successful.
AH != 00h: Function call failed.
= 01h: Unknown PFA. PFA does not match devices in system.
= 02h: Invalid Input Argument.
= 03h: Too many banks (RAS/CS lines) requested.
= 04h: Requested bank(s) RAS/CS line(s) not supported.
= 05h: Aux Memory not supported.
= 06h: Noncacheable/ write through cache area not available.
= 80h: Function needs to be called again to return additional information.
= FFh: Other unknown error.

3.4 ROM Signature

VUMA devices must have a ROM signature, within the first 1K, "_VUMA_XXxx" where XX is major version and xx is minor version. Following the minor version number:

A. 16 bit value with the minimum amount of memory necessary, in 64Kblocks, for booting. It is not a requirement to have all devices working to boot. Only devices essencial to bring up a system. such as VGA and a boot device (hard drive) are necessary. After booting, a device driver or utility may request additional memory for non-essential devices.

B. 16 bit value with bit map of memory banks (RAS/CS lines) supported. Bit 0 corresponds to bank number 0 etc. If a bit is set then the bank (RAS/CS line) is supported by the VUMA device.

C. 16 bit value for DRAM support. Bit set if supported.
Bit 0 = Fast Page
Bit 1 = EDOn
Bit 2 = SDRAM
Bit 3 = PN EDO (Burst EDO)
All other bit are reserved.

D. 8 bit value for features.
Bit 0 = Snooping supported by VUMA device if set. (See h/w spec for definition of snooping.

Petitioners HTC & LG - Exhibit 1002, p. 238

3.5 VUMA-SBE Functions

The following defined VUMA-SBE services are not included in the VBE standard documentation.

3.5.0 00h - Report VUMA Core Logic Capabilities

This function should be called before any other VUMA-SBE function is called to ensure that the VUMA system is present, and to inquire the core logic capabilities

Input: (AX is used only when being called by one of the two 16 bit interfaces.)
AH = F4h
AL = 01h
BL = 00h

Output:
AX = Status (see **section 3.3**)
BL = Major BIOS revision = 01h
BH = Minor BIOS revision = 00h
CX = Banks (RAS/CS) that are supported, could have memory installed, by the core logic controller
Bit 0 = bank 0
Bit 1 = bank 1
etc.
DX[3:0] = Core logic capabilities
0 = No special features
Bit 0 = 1 -> Controller supports non-cacheable regions
Bit 1 = 1 -> Controller supports write-thru cache regions
Bit 2 = 0 -> Cannot change at run time from cached to non-cached and back
1 -> Can change at run time from cached to non-cached and back
Bit 3 = 0 -> Cannot change at run time from non-write through to write through and back
1 -> Can change at run time from cached to non-write through and back
DX[4] = Core logic supports snooping, this item is relevant only when synchrounous DRAM is supported
0 = Snooping is NOT supported
1 = Snooping is supported
SI = Bank (RAS/CS) numbers with memory. Bit set if has memory.
Bit 0 = bank 0

Petitioners HTC & LG - Exhibit 1002, p. 239

Bit 1 = bank 1
etc.

DI = Bank (RAS/CS) numbers with memory and support VUMA.
Bit 0 = bank 0
Bit 1 = bank 1
etc.

Petitioners HTC & LG - Exhibit 1002, p. 240

3.5.1 01h - Request VUMA Main Memory capabilities

This function returns system controller capabilities.

Input: (AX is used only when being called by one of the two 16 bit interfaces.)
AH = F4h
AL = 01h
BL = 01h

Output:
AX = Status (see **section 3.3**)
BX = Minimum size can allocate in 64 K increments
CX = Maximum size can allocate in 64 K increments
SI = System memory noncacheable or write through area granularity in 64 K blocks. Minimum block size region in system memory that can have L2 cacheable, non-cacheable, or write through cache. This is a basis provided for rounding up Aux memory size request.
0 = Not defined
DI = VUMA main memory size increments from minimum size in 64K. When memory is disabled the CPU does not have access to it but refresh still occurs.
0 = Not defined

Petitioners HTC & LG - Exhibit 1002, p. 241

3.5.2 02h - Set (Request) VUMA Main Memory for Next Boot

This function sets the size of the Main VUMA memory for the next boot. An input parameter is the memory bank numbers (RAS/CS numbers) that can be accessed by the VUMA device. The banks supported (parameter passed in DX) **must** be the same as reported in the ROM signature as specified in section 3.4 of this document.

Input: (AX is used only when being called by one of the two 16 bit interfaces.)
AH = F4h
AL = 01
BL = 02
CX = PFA number.
CH = Bus Number (0 .. 255)
CL[7:3] = Device number
CL[2:0] = Function Number
DX = Banks (RAS/CS) that are supported by the calling device.
Bit 0 = bank 0
Bit 1 = bank 1
etc.
SI = Size in 64 Kbytes (Will be rounded up by the system BIOS if necessary)

Output:
AX = Status (see **section 3.3**)
DX = Actual size in 64 K bytes allocated.

Petitioners HTC & LG - Exhibit 1002, p. 242

3.5.3 03h - Get VUMA Main Memory Size for Next Boot for a Device

This function returns the size of Main VUMA memory to be set for the next boot for the selected controller.

Input: (AX is used only when being called by one of the two 16 bit interfaces.)
AH = F4h
AL = 01h
BL = 03h
CX = PFA number
CH = Bus Number (0 .. 255)
CL[7:3] = Device number
CL[2:0] = Function Number

Output:
AX = Status (see **section 3.3**)
DX = Size in 64 K bytes

Petitioners HTC & LG - Exhibit 1002, p. 243

3.5.4 04h - Get Memory Size for Next Boot for all MAIN VUMA Memory.

This function returns the size of Main VUMA memory to be set for the next boot.

Input: (AX is used only when being called by one of the two 16 bit interfaces.)
AH = F4h
AL = 01h
BL = 04h

Output:
AX = Status (see **section 3.3**)
DX = Size in 64 K bytes

Petitioners HTC & LG - Exhibit 1002, p. 244

3.5.5 05h - Get Current Memory Size for a Device or for all VUMA Memory

This function returns the size of Main VUMA memory for the selected controller. Note: Value returned in BH is for all of VUMA main memory since all main memory is either enabled or disabled. (Allows CPU access or does not allow CPU access.)

Input: (AX is used only when being called by one of the two 16 bit interfaces.)
AH = F4h
AL = 01h
BL = 05h
CX = PFA number
CH = Bus Number (0 .. 255)
CL[7:3] = Device number
CL[2:0] = Function Number
If CX = FF then return for all devices

Output:
AX = Status (see **section 3.3**)
BH[0] = Memory access for all of Main VUMA memory.
0 -> Memory is not enabled, not visible to the CPU
1 -> Memory is enabled, visible to the CPU
CX = Bit map of bank (RAS/CS) numbers used. Bit set if bank is used.
Bit 0 = bank 0
Bit 1 = bank 1
etc.
DX = Size in 64 Kbytes
SI = upper 16 bits of physical start address
DI = lower 16 bits of physical start address

Petitioners HTC & LG - Exhibit 1002, p. 245

3.5.6 06h - Return Memory Speed/Type/Location/Size

This function returns information about the type of memory installed for the selected bank (RAS/CS). Fractions greater than 0.5 are rounded up. Fractions 0.5 and lower are rounded down. If a bank is logically divided into more than one area, then the function needs to be called more than once. AX indicates whether the function is done or not. If more than 1 bit in CX is set then error code AH= 03H will be returned, therefore only one bit should be set in CX when calling this function. Note: If a bank has contigous memory, but part of the memory is system memory and part is VUMA memory, the information will also be returned in two steps. BX[14] will reflect the type of memory.

Input: (AX is used only when being called by one of the two 16 bit interfaces.)
AH = F4h
AL = 01h
BL = 06h
CX = Bank (RAS/CS) number
Bit 0 = bank 0
Bit 1 = bank 1
etc.
DX = Serial calling number

Output:
AX = Status (see **section 3.3,** if equal to 80, must call for more info)
BX[15] = Reserved.
BX[14] = MAIN VUMA memory.
0 = not main VUMA memory, is system memory
1 = main VUMA memory.
BX[13:7] = Speed of memory in nano-seconds
0 = undefined, else value.
BX[6:0] = Core logic controller speed in nano-seconds
0 = undefined, else value.
CX[3:0] = Type of memory
0000 = Undefined
0001 = Fast page mode
0010 = EDO
0011 = SDRAM
0100 = PN EDO (Burst EDO)
CX[5:4] = CAS latency
CX[7:6] = Burst
0 = 1
1 = 2
2 = 4
3 = undefined

Petitioners HTC & LG - Exhibit 1002, p. 246

CX[8] = Burst order
0 = Linear
1 = Sequential

CX[15:9] = Voltage of memory in tenths of a volt
0 = undefined
DX[7:0] = Size in Mega-bytes
SI = upper 16 bits of physical start address
DI = lower 16 bits of physical start address

Petitioners HTC & LG - Exhibit 1002, p. 247

3.5.7 07h - Enable/disable Main VUMA memory

The ability to enable/disable CPU access to Main VUMA memory is not required if Main VUMA memory is not contiguous with system memory. If supported, this function enables/disables CPU access to Main VUMA memory. When any device makes this call, all devices that have main VUMA memory will be affected. When CPU access to main VUMA memory is disabled, access to video memory may be done through the PCI bus.
Note: During run time (after Int 19) CPU access can not be disabled.

Input: (AX is used only when being called by one of the two 16 bit interfaces.)
AH = F4h
AL = 01h
BL = 07h
BH[0] = Memory access
0 -> Enable CPU access to VUMA Main memory.
1 -> Disable CPU access to VUMA Main memory. (Can not be done at run time.)

Output:
AX = Status (see **section 3.3**)

Petitioners HTC & LG - Exhibit 1002, p. 248

3.5.8 08h - Set (Request)/Free VUMA Aux Memory

This function sets (requests) the size of the aux memory for use at run time. See "How To Access Aux VUMA Memory" in the Appendix to be added at a later time. A physical starting address and size is passed in. This function will flush and then turn off caching for this area or change the area to write through cache.

Input: (AX is used only when being called by one of the two 16 bit interfaces.)
AH = F4h
AL = 01
BL = 08
BH[1:0] = Type of cache
Bit 0 set if non-cachable
Bit 1 set if write-Thru
CX = PFA number
CH = Bus Number (0 .. 255)
CL[7:3] = Device number
CL[2:0] = Function Number
DX = Size in K bytes, free VUMA Aux memory if set to 0
SI = upper 16 bits of physical address
DI = lower 16 bits of physical address

Output:
AX = Status (see **section 3.3**)
DX = Actual size in Kbytes allocated (rounded up by the system BIOS if necessary)

Petitioners HTC & LG - Exhibit 1002, p. 249

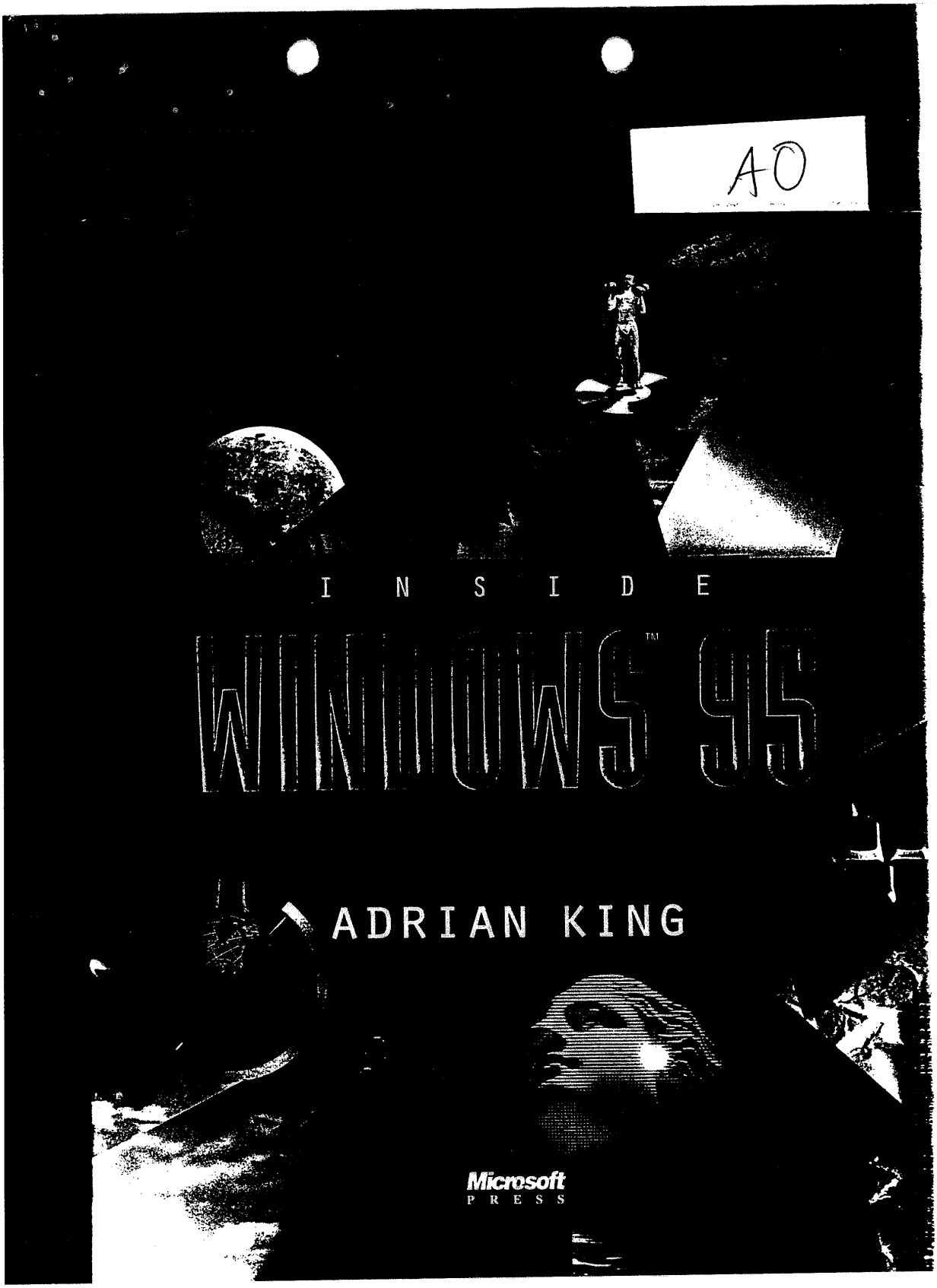
3.5.9 09h - Get VUMA Aux Memory Size

This function returns the size of the aux memory being used by a VUMA device.

Input: (AX is used only when being called by one of the two 16 bit interfaces.)
AH = F4h
AL = 01h
BL = 09h
CX = PFA number
CH = Bus Number (0 .. 255)
CL[7:3] = Device number
CL[2:0] = Function Number

Output:
AX = Status (see **section 3.3**)
DX = Size in Kbytes
SI = upper 16 bits of physical address
DI = lower 16 bits of physical address

Petitioners HTC & LG - Exhibit 1002, p. 250



Petitioners HTC & LG - Exhibit 1002, p. 251

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Copyright © 1994 by Adrian King

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Library of Congress Cataloging-in-Publication Data
King, Adrian, 1953-
Inside Windows 95 / Adrian King.
p. cm.
Includes index.
ISBN 1-55615-626-X
1. Windows (Computer programs) 2. Microsoft Windows (Computer file) I. Title.
QA76.76.W56K56 1994
005.4'469--dc20 93-48485
CIP

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 QMQM 9 8 7 6 5 4

Distributed to the book trade in Canada by Macmillan of Canada, a division of Canada Publishing Corporation.

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 (206) 936-7329.

PageMaker is a registered trademark of Aldus Corporation. Apple, AppleTalk, LaserWriter, Mac, Macintosh, and TrueType are registered trademarks of Apple Computer, Inc. LANtastic is a registered trademark of Artisoft, Inc. Banyan and Vines are registered trademarks of Banyan Systems, Inc. Compaq is a registered trademark of Compaq Computer Corporation. CompuServe is a registered trademark of CompuServe, Inc. Alpha AXP, DEC, and Pathworks are trademarks of Digital Equipment Corporation. LANstep is a trademark of Hayes Microcomputer Products, Inc. HP and LaserJet are registered trademarks of Hewlett-Packard Company. Intel is a registered trademark and EtherExpress, Pentium, and SX are trademarks of Intel Corporation. COMDEX is a registered trademark of Interface Group-Nevada, Inc. AS/400, IBM, Micro Channel, OS/2, and PS/2 are registered trademarks and PC/XT is a trademark of International Business Machines Corporation. 1-2-3, Lotus, and Notes are registered trademarks of Lotus Development Corporation. Microsoft, MS, MS-DOS, and XENIX are registered trademarks and ODBC, Win32s, Windows, Windows NT, and the Windows operating system logo are trademarks of Microsoft Corporation. MIPS is a registered trademark and R4000 is a trademark of MIPS Computer Systems, Inc. NetWare and Novell are registered trademarks of Novell, Inc. Soft-Ice/W is a registered trademark of Nu-Mega Technologies, Inc. DESQview is a registered trademark and Qemm is a trademark of Quarterdeck Office Systems. OpenGL is a trademark of Silicon Graphics, Inc. PC-NFS, Sun, and Sun Microsystems are registered trademarks of Sun Microsystems, Inc. TOPS is a registered trademark of TOPS, a Sun Microsystems company. UNIX is a registered trademark of UNIX Systems Laboratories.

Acquisitions Editor: Mike Halvorson
Project Editor: Erin O'Connor
Technical Editors: Seth McEvoy and Dail Magee, Jr.

Petitioners HTC & LG - Exhibit 1002, p. 252

was originally designed as a standardized format for 32-bit protected mode code modules. There is an API, internal to the base system, that VxDs can use.[11] Obviously, the scope of these functions is at a much lower level than the scope of the services called on directly by applications.

Memory Management

Memory management in Windows takes place at two different levels: a level seen by the application programmer and an entirely different view seen by the operating system. Over the course of different releases of Windows, the application programmer has seen little change in the available memory management APIs. Within the system, however, the memory management changes have been dramatic. Originally, Windows was severely constrained by real mode and 1 megabyte of memory. Then expanded memory provided a little breathing room, and currently the use of enhanced mode and extended memory relieves many of the original constraints. Windows 95 goes further yet and essentially removes all the remaining memory constraints.

Windows 95 continues to support all the API functions present in Windows 3.1, and you can still build and run applications that use the segmented addressing scheme of the 286 processor. However, if you look at the detailed documentation for the Windows 95 memory management API, you'll see that all of the API functions originally designed to allow careful management of a segmented address space are now marked "obsolete." The "obsolete" list includes, for example, all the functions related to selector management. The reason, of course, is the Windows 95 support for 32-bit linear memory and the planned obsolescence of the segmented memory functions—yet another unsubtle hint that the Win32 API is the API you should be using to write Windows applications.

Although use of the 32-bit flat memory model simplifies a lot of Windows programming issues, it would be misleading to say that Windows memory management has suddenly gotten easy.[12] Windows 95 actually has a number of new application-level memory management

11. The Windows Device Driver Kit is the best reference for detailed information on VxDs and the associated API functions.

12. The Windows 95 documentation lists 45 API functions under the heading "Memory Management." The "obsolete" list numbers 28 API functions.

Petitioners HTC & LG - Exhibit 1002, p. 253

capabilities. All of the functions relate to the management of memory within the application's *address space*, the private virtual memory allocated to the process. The systemwide management of memory is the responsibility of the base system, and the Windows API aims to hide many of the details of the system's lower-level functions.

Application Virtual Memory

Figure 3-3 illustrates the basic layout of a Win32 application's virtual memory. Every Win32 application has a similar memory map, and each such address space is unique. However, it is still not fully protected: the private memory allocated to one Win32 application can be addressed by another application. The Win32 application's private address space is also the region in which the system allocates memory to satisfy application requests at runtime.

The system address space is used to map the system DLLs into the application's address space. Calls to the system DLLs become calls into this region. Applications can also request the dynamic allocation of memory by means of virtual addresses mapped to the shared region. Having virtual addresses mapped to the shared address space caters to the need for controlled sharing of memory with other applications.

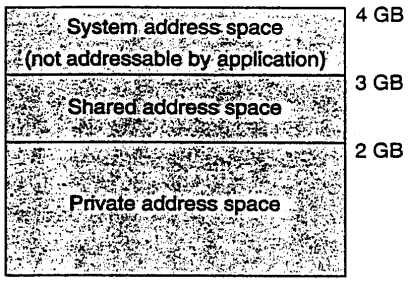


Figure 3-3.
Application virtual memory map.

Requests for memory at runtime fall into one of two categories: the application can make an explicit request for extra memory, or the system can respond to an implicit request for memory—that is, allocate memory to an application as a side effect of allocating some other resource. An implicit request occurs, for example, when an application

Petitioners HTC & LG - Exhibit 1002, p. 254

creates a new window on screen: the system must allocate memory for the data structures used to manage the window. Windows 95 claims memory for resource allocation from a large 32-bit linear region rather than from the restrictive 64K segment used in previous versions of Windows. An ongoing problem in versions through Windows 3.1, running out of memory during resource allocation, has been largely eradicated in Windows 95.

Heap Allocation

In Windows parlance, the term *heap* describes the region of memory used to satisfy application memory allocation requests. In Windows 3.1, the system maintains both a *local heap* and a *global heap*. The local heap is a memory region within the application's address space, and the global heap is a memory region belonging to the system. As an application makes requests for local memory, its address space is adjusted to encompass the newly allocated memory. The system resolves requests for global memory from the same system memory pool used for all applications. It's possible to run out of either or both resources, although the use of a 2-GB address space makes this highly unlikely. Exhaustion of the local heap affects only a single application. Exhaustion of the global heap has systemwide repercussions.

Windows 3.1 programmers have to consider a variety of factors as they decide how to satisfy an application's runtime memory requirements. Windows 3.1 also has a range of API functions for manipulating dynamically allocated segments, and the manipulation of these shifting regions is further complicated by the underlying segmented memory model. It isn't just a chunk of memory that must be allocated. The application also needs a selector so that it can address the memory correctly. Under Windows 95, the Win32 application model does away with all these considerations. Selectors are no longer required—it's simply a 32-bit address that identifies the new memory—and the local and global heaps are merged into a single heap. The API functions that deal with selectors and the manipulation of memory regions in a segmented model all become obsolete.

Windows 95 Application Memory Management

For a Windows programmer, the Win32 API greatly simplifies the most common dynamic memory allocation chores. Furthermore, the increased capability of the underlying 32-bit architecture allowed the Windows designers to add a number of new functions for application memory management.

Petitioners HTC & LG - Exhibit 1002, p. 255

- Windows 95 provides functions that support private heaps whereby an application can reserve a part of memory within its own address space. The application can create and use as many private heaps as it wishes and can direct the system to satisfy subsequent memory allocation calls from a specific private heap. An application might use the local heap functions to create several different memory pools that each contain data structures of the same type and size.
- Windows 95 provides functions that allow an application to reserve a specific region of its own virtual address space that once reserved won't be used to satisfy any other dynamic memory allocation requests. In a multithreaded application, the 32-bit pointer to this reserved region is a simple way to provide each thread with access to the same memory.
- Memory mapped files allow different applications to share data. An application can open a named file and map a region of the file into its virtual address space. The data in the file is then directly addressable by means of a single 32-bit memory address. Other applications can open the same file, map it into their private address spaces, and reference the same data by means of a single pointer.

System Memory Management

Regardless of changes in the details of application memory management, the Windows programming model has remained pretty consistent through the different product releases. Allocating blocks of memory at runtime, using a reference to a block to manipulate it, and ultimately returning the block to the system for re-use is the way in which Windows programmers have always dealt with dynamic memory requirements. Windows 95 is no different. What has changed, however, is the way in which the system realizes the application's requests for dynamic memory.

Starting with the Windows 3.0 enhanced mode and continuing with the Windows 95 Win32 application model, the Windows API manipulates only the application's virtual address space. This means that an application request for a block of memory will adjust the application's virtual address map but might do absolutely nothing to the system's physical memory. Remember that the 386 deals with physical

Petitioners HTC & LG - Exhibit 1002, p. 256

memory in pages each 4K in size. This page size is reflected in the virtual address space map of every Windows application. If an application requests 100K of memory, for example, its virtual address space will have 25 pages of memory added to it. The system will also adjust the data in its own control structures to reflect the application's new memory map.

However, at the time of allocation, Windows won't do anything to the physical memory in the system. It's only when the application starts to use the memory that the underlying system memory management kicks in and allocates physical memory pages to match the virtual memory references the application makes. If the application allocates but never references a region of its virtual memory space, the system might never allocate any physical memory to match the virtual memory. The ability of the 386 to allow physical memory pages to be used at different times within different virtual address spaces is the basis for the operating system's virtual memory capabilities.

Deep within the system are a range of memory management primitives available to device drivers and other system components that sometimes deal with virtual memory and sometimes force the system to commit actual physical memory pages. But these primitives are specific to the base operating system. Neither applications nor the Windows subsystem knows or cares about physical memory. Applications can force the system to allocate physical memory only by actually using the memory: namely, by reading from and writing to locations within a page. The separation of Windows memory management into the virtual and physical levels is a key aspect of the system. Applications and the Windows subsystems deal with defined APIs and virtual address spaces. The base system deals with physical memory as well as virtual address spaces.

Although physical memory is transparent to an application, its behavior can radically affect the performance of the system. For example, scanning through a two dimensional array of data row by row using C as the programming language will cause memory to be accessed from low to high virtual addresses because C stores two dimensional array data structures in *row major order*. As the memory sweep proceeds, the system will allocate physical memory pages to match the virtual memory accesses. Byte-at-a-time access will cause the system to allocate a new physical page every 4096 references. Other languages—FORTRAN, for example—store two dimensional arrays in *column major order*. Referencing the data row by row will generate memory references to widely scattered

Petitioners HTC & LG - Exhibit 1002, p. 257

memory locations, forcing a much higher frequency of physical page allocation and much-reduced application performance. So, although the programmer doesn't have to worry about matching virtual memory to physical memory, it is a good idea for the programmer to know something about how the underlying system primitives and hardware support the application.

Windows Device Support

The most important aspect of the Windows device driver architecture is its ability to *virtualize* devices. (Yes, it's that word again.) The greatest difference between the device drivers of Windows 95 and Windows 3.1 is the extensive use of protected mode drivers in Windows 95—in fact, it will be unusual if your system uses any real mode drivers at all after you install Windows 95. The use of protected mode for the drivers pays off in terms of both system performance and robustness. The manufacturers of disk devices can adopt a new driver architecture—borrowed from Windows NT—that almost guarantees the availability of a protected mode driver for every hard disk. In addition, new protected mode drivers for CD ROM devices, serial ports, and the mouse make the possibility of needing to support a device with a real mode driver quite remote.

Device Virtualization

The device virtualization capability allows Windows 95 to use the memory and I/O port protection capabilities of the 386 processor to share devices among the different virtual machines. Every MS-DOS VM believes it has full control over its host PC and is unaware of the fact that it might be sharing the screen with other MS-DOS VMs or with the Windows applications running in the System VM. For MS-DOS applications, the display drivers must reside in the lowest level of the operating system. Many MS-DOS applications, particularly those that use the display in a graphics mode or use serial ports, will address the hardware directly. Windows has to intercept all such direct access in order to bring order to a potentially chaotic situation. The MS-DOS application knows nothing of the need to cooperate with other applications and certainly doesn't depend on a system device driver to get the job done. With Windows applications, the system has a slightly easier task since device access is always

Petitioners HTC & LG - Exhibit 1002, p. 258

AP

SGS-THOMSON MICROELECTRONICS

TECHNICAL NOTE

MPEG VIDEO OVERVIEW

AN OVERVIEW OF THE MPEG COMPRESSION ALGORITHM

The MPEG standard was developed in response to industry needs for an efficient way of storing and retrieving video information on digital storage media. One inexpensive medium is the CD-ROM which can deliver data at approximately 1.2 Mbps, the MPEG standard was subsequently aimed at this data rate, in fact the data rate is variable and all decoders must be able to decode at rates up to 1.856 Mbps. Although the standard was developed with CD-ROM in mind, other storage and transmission media can include DAT, Winchester Disk, Optical Disk, ISDN and LAN.

Two other relevant international standards were also being developed during the work of the MPEG committee : H.261 by CCITT aimed at telecommunications applications and ISO 10918 by the ISO JPEG committee aimed at the coding of still pictures. Elements of both standards were incorporated into the MPEG standard, but subsequent development work by the committee resulted in coding elements found in neither.

Some of the participants in the MPEG committee include : Intel, Bellcore, DEC, IBM, JVC Corp, THOMSON CE, Philips CE, SGS-THOMSON, Sony Corp, NEC Corp and Matsushita EIC. These are not necessarily be the most important members of the committee but it gives an indication of the relevant importance of the MPEG standard.

Although the MPEG standard is quite flexible, the basic algorithms have been tuned to work well at data rates from 1 to 1.5 Mbps, at resolutions of about 350 by 250 Pixels at picture rates of up to 25 or 30 pictures per second. MPEG codes progressively-scanned images and does not recognise the concept of interlace; interlaced source video must be converted to a non interlace format prior to encoding. The format of the coded video allows forward play and pause, typical coding and decoding methods allow random access, fast forward and reverse play also, the requirements for these functions are very much application dependent and different encoding techniques will include varying levels of flexibility to account for these functions.

Compression of the digitised video comes from the use of several techniques : Sub sampling of the chroma information to match the human visual system, differential coding to exploit spatial redundancy, motion compensation to exploit temporal redundancy, Discrete Cosine Transform (DCT) to match typical image statistics, quantization, variable length coding, entropy coding and use of interpolated pictures.

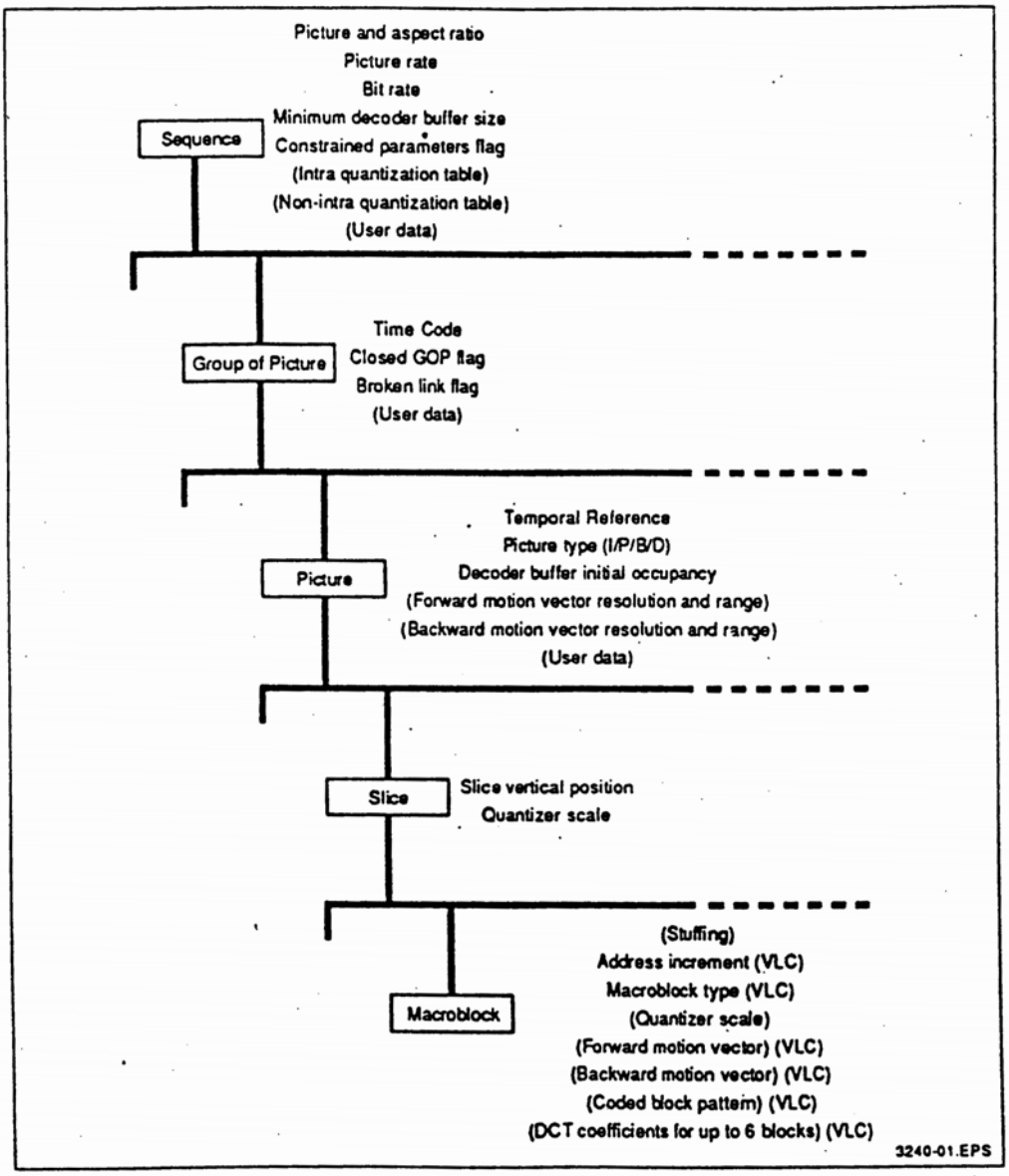
ALGORITHM STRUCTURE AND TERMINOLOGY

The MPEG hierarchy is arranged into layers (Figure 1). This layered structure is designed for flexibility and management efficiency, each layer is intended to support a specific function i.e. the sequence layer specifies sequence parameters such as picture size, aspect ratio, picture rate, bit rate etcetera , whereas the picture layer defines parameters such as the temporal reference and picture type.

This layered structure improves robustness and reduces susceptibility to data corruption.

Petitioners HTC & LG - Exhibit 1002, p. 259

Figure 1 : MPEG Bistream.Hierarchy



For convenience of coding, macroblocks are divided into six blocks of component Pixels - four luma and two chroma (Cr and Cb) (Figure 2).

Blocks are the basic coding unit and the DCT is applied at this block level. Each block contains 64 component Pixels arranged in an 8x8 array (Figure 3)

Petitioners HTC & LG - Exhibit 1002, p. 260

Figure 2 : Macroblock Structure

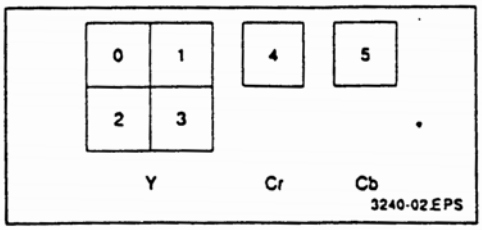
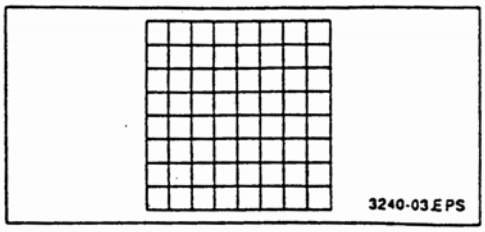


Figure 3 : Block Structure



There are four picture types : I pictures or INTRA pictures, which are coded without reference to any other pictures; P pictures or PREDICTED pictures which are coded using motion compensation from a previous picture; B pictures or BIDIRECTIONALLY predicted pictures which are coded using interpolation from a previous and a future picture and D pictures or DC pictures in which only the low frequency component is coded and which are only intended for fast forward search mode. B and P pictures are often called Inter pictures. Some other terminology that is often used are the terms M and N, M+1 represents the number of frames between successive I and P pictures whereas N+1 represents the number of frames between successive I pictures. M and N can be varied according to different applications and requirements such as fast random access. In Figure 4, M = 3 and N = 12.

A typical coding scheme will contain a mix of I,P and B pictures. A typical scheme will have an I picture every 10 to 15 pictures and two B pictures between succesive I and P pictures: refer to Figure 4.

MPEG COMPRESSION ALGORITHM

The MPEG algorithm is based around two key techniques : temporal compression and spatial compression. Temporal compression relies upon similarity between successive pictures using prediction and motion compensation whereas spatial compression relies upon redundancy within small areas of a picture and is based around the DCT transform, quantization and entropy coding techniques.

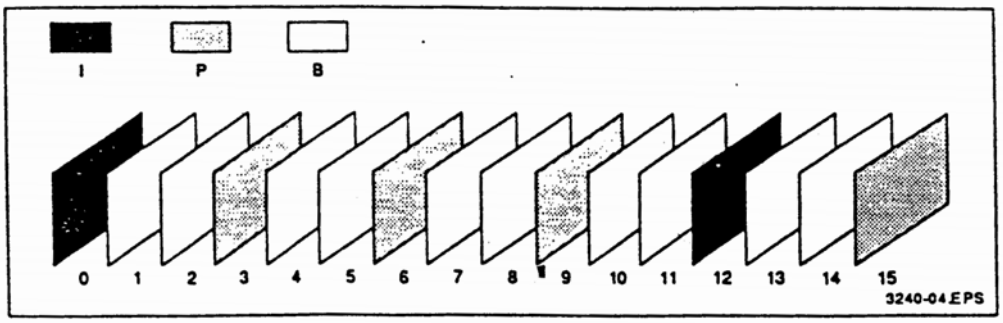
TEMPORAL COMPRESSION

Inter (B and P) pictures are coded using motion compensation, primarily prediction and interpolation.

Prediction

The predicted picture is the previous picture modified by motion compensation. Motion vectors are calculated for each macroblock. The motion vector is applied to all four luminance blocks in the macro block. The motion vector for both chrominance blocks is calculated from the luma vector. This technique relies upon the assumption that within a macroblock the difference between successive pictures can be represented simply as a vector transform (i.e. there is very little difference between successive pictures, the key difference being in position of the Pixels.).

Figure 4 : Typical sequence of pictures in display order



Petitioners HTC & LG - Exhibit 1002, p. 261

Interpolation

Interpolation (or bidirectional prediction) generates high compression in that the picture is represented simply as an interpolation between the past and future I or P pictures (again this is performed on a macroblock level).

Pictures are not transmitted in display order but in the order in which the decoder requires them to decode the bitstream (the decoder must of course have the reference picture(s) before any interpolated or predicted pictures can be decoded). The transmission order is shown in Figure 6.

Figure 5 : Make up of I, B and P pictures

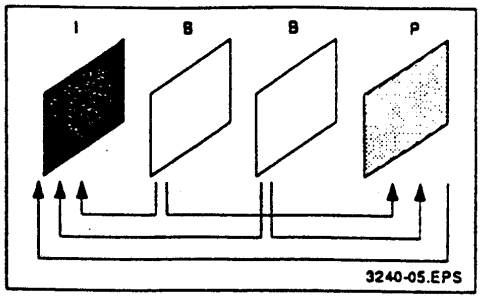
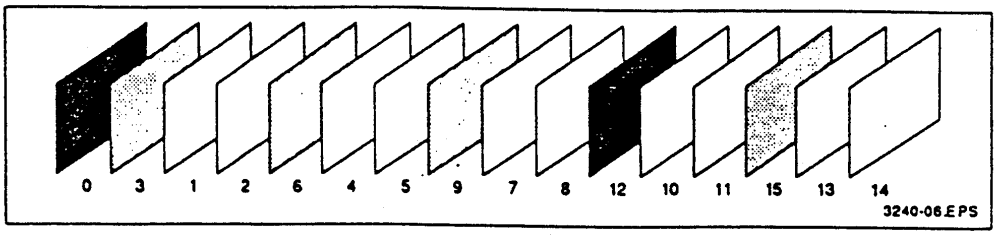


Figure 6 : Typical sequence of pictures in transmission order



SPATIAL COMPRESSION

The spatial compression techniques are similar to those of JPEG , DCT, Quantization and entropy coding. The compression algorithm takes advantage of the redundancy within each block (8 x 8 Pixels).

The resulting compressed datastream is made up of a combination of spatial and temporal compression techniques which best suit the type of picture being compressed. Decoding is controlled through the use of MPEG system codes which are put into the data stream explaining how to reconstruct specific areas of picture - as shown in Figure 1.

CONCLUSION

Through a combination of techniques, MPEG compression is designed to give good quality (typically similar or better quality to VHS) images from such storage media as CD-ROM. The quality is however, dependent upon the type of picture compressed and the level of redundancy within the sequence coded. Picture quality will also depend upon how well the sequence has been coded and which features are required - For Example : For fast random access, N will tend towards zero hence the quality of compression will deteriorate, if random access is not required then the number of P and B frames can increase, hence increasing the potential quality. The standard does not specify a method of compression but a syntax for the compressed data, this allows for differing compression techniques depending upon differing requirements. The decoding techniques are defined due to the nature of the compressed data stream.

This method allows for true flexibility in coding whilst retaining the format and hierarchy ensuring compatibility in the datastream and hence uniform readability.

Information furnished is believed to be accurate and reliable. However, SGS-THOMSON Microelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No licence is granted by implication or otherwise under any patent or patent rights of SGS-THOMSON Microelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. SGS-THOMSON Microelectronics products are not authorized for use as critical components in life support devices or systems without express written approval of SGS-THOMSON Microelectronics.

© 1992 SGS-THOMSON Microelectronics - All Rights Reserved

SGS-THOMSON Microelectronics GROUP OF COMPANIES

Australia - Brazil - China - France - Germany - Hong Kong - Italy - Japan - Korea - Malaysia - Malta - Morocco
The Netherlands - Singapore - Spain - Sweden - Switzerland - Taiwan - United Kingdom - U.S.A.

ORDER CODE :

Petitioners HTC & LG - Exhibit 1002, p. 262

F3.

On the Bus Arbitration for MPEG 2 Video Decoder

Chia-Hsing Lin and Chein-Wei Jen

Department of Electronics Engineering and Institute of Electronics
National Chiao Tung University

Abstract

A bus arbitration scheme for the MPEG-2 video decoder VLSI developed by NCTU is proposed in this paper. Compared to the traditional pure stochastic bus scheduling scheme, the internal buffer requirement and bus arbitration overheads are reduced due to the deterministic nature of this strategy. This bus arbitration scheme has been verified using Verilog simulator and will be implemented in the NCTU MPEG-2 decoder.

1. Introduction

ISO standard 13818[1] known as MPEG-2 (Moving Pictures Expert Group) have been adopted in many applications like TV set-top boxes, PC add-on cards and entertainment machines. To promote the success of this motion picture standard, it is attractive to develop a single chip decoder, accompanied by DRAMs, to establish a low cost decoding system. The cheapness of standard DRAMs is the main reason for MPEG-2 decoder VLSI to use as the temporal picture buffer. However, the decoder VLSI also has to contain several internal buffers, which will increase the cost of this decoder, to conquer the limited memory bandwidth provided by DRAM. Therefore, it is important to design a suitable bus arbitration scheme for memory access to utilize the bandwidth efficiently in order to reduce the amount of internal buffers.

In this paper, we propose a bus arbitration scheme for MPEG-2 decoder of main profile and main level (MP@ML). We will first give an architectural overview and functional description of NCTU MPEG-2 decoder in the next section. The bottleneck issue of memory access and the proposed bus arbitration scheme will then be presented in section 3. Section 4 demonstrates some simulation results using uninterpreted model[2] and Verilog simulator. Section 5 concludes the paper.

2. MPEG-2 Decoder Design

The architecture of the MPEG-2 video decoder developed by NCTU is shown in Fig.1. The system controller provides controls for other functional units. The decoding pipeline (including variable-length decoder, inverse quantizer, inverse discrete cosine transform unit and motion compensation unit) performs the main MPEG-2 decoding operations. A 64-bit memory data bus is used for the I/O transactions between functional units and external memory (which is used as the VBV buffer and reference picture buffer). The memory I/O transactions are managed by a memory controller. The video interface controls the display timing for video output and performs some post-processing operations like the output format conversion from 4:2:0 to 4:2:2.

To perform the decoding and display processes, the decoder first receives compressed bitstream from host interface to bitstream buffer (BBUF) and transfers them to the VBV buffer, which is located in the external memory. The decoder will then re-read the bitstream from VBV buffer to VLD buffer (VLD BUF) for the requirement of decoding pipeline. If the macroblock currently decoded is nonintra-coded, the decoder may also need to load the reference blocks from reference picture buffer, which is also in the external memory, to perform motion-compensation and interpolation. After adding the results from IDCT and MC units, the decoder will write back the sums to the reference buffer. Finally, at the time to display the previous decoded data, the decoder will read video data again from reference picture buffer to video output buffer (VBUF). Fig.2 shows the timing diagram for each functional unit in the MPEG-2 decoder.

VLSI Tech, Syst. and Appl., 1995 Symposium.

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 263

3. Bus Arbitration for Memory Access

3.1 The Problem of limited Memory Bandwidth

In order to reduce the number of DRAMs and the number of I/O pins, the VBV buffer and reference picture buffer share the same external memory port. A memory bandwidth problem occurs because of the several memory I/O transactions (including the bitstream data loading and storing, video output loading, reference picture loading, and predicted data storing) and DRAM refreshing cycles. Also, the overheads introduced by stochastic bus arbitration between different transaction requests will worsen the bus load. The traditional bus arbiter using fixed priority scheme[3] may cause functional units to starve without large internal memories for I/O buffering because of the heavy memory bus load in MPEG-2 with CCIR and higher resolution. In [4], Tatsuhiko, et al. proposed a sophisticated scheme to reduce the memory bottleneck. The basic idea of this scheme is a combination of priority assignment and polling (Fig.3). However, the extra FIFO and internal memories are still required to accommodate the stall of the decoding pipeline due to the stochastic nature of this scheme.

3.2 The Proposed Scheduling Scheme for Memory Access

Unlike the previous pure stochastic scheduling scheme, a "pseudo-deterministic" scheme to allocate the bandwidth for each I/O transaction is proposed in this paper. For each macroblock prediction mode, we analyze the worst case in data transferring and allocate the required duration for each memory I/O in one macroblock period according to the following criteria:

$$N_{video} + N_{load} + N_{store} + N_{btio} + N_{refresh} + N_{overhead} \le N_{MB} \le \frac{\text{clock rate}}{(\text{no. of MBs in a frame}) \times (\text{frame rate})} \quad (1)$$

$$\left(\frac{N_{MB}}{N_{ratio} \times N_{width} \times N_{dis}} + N_{ov} \right) \times N_{access} \times N_{dis} \le N_{video} \quad (2)$$

where

N_{MB} is the number of cycles to decode one macroblock,

N_{video} is the number of cycles to transfer video output data to display buffer,

N_{load} is the number of cycles to read reference blocks from reference picture buffer,

N_{store} is the number of cycles to write predicted macroblock to display buffer,

N_{btio} is the number of cycles to read from and write to VBV buffer,

$N_{refresh}$ is the number of cycles to refresh DRAM,

$N_{overhead}$ is the bus arbitration overhead,

N_{ratio} is the ratio of system clock and video output frequencies.

N_{width} is the width of memory bus,

N_{ov} is the number of DRAM page mode overhead,

N_{access} is the number of cycles to access one word from external memory in page mode, and

N_{dis} is the number of samples to display for one pixel.

Furthermore, to guarantee that the display process does not overrun the decoding process, the decoding rate must be larger than the display rate. Hence one more condition must be held:

$$\frac{\text{No of pixels in one picture}}{\text{No. of samples output in one macroblock time}} \ge \frac{\text{No of pixels in active region of a picture}}{\text{No. of samples in one macroblock}} \quad (3)$$

where

$$\text{No. of samples output in one macroblock time} = \left(\frac{N_{MB}}{N_{ratio} \times N_{width} \times N_{dis}} \right)$$

After we determine suitable time period for each I/O transaction, we can schedule them in the decoding time domain as the state diagram for bus arbitration shown in Fig. 4. The memory controller normally monitors the I/O requests (i.e., polling) to or from VBV buffer and perform the compressed bitstream input and output. While it is time for the transaction of any other I/O process, the bus will be allocated to that process until its transaction encounters end. The

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 264

memory controller will then return to the state to handle the memory access for VBV buffer input or output.

Fig.5 shows one example of the scheduling scheme for different macroblock prediction modes. Assume that the chip outputs one 8-bit video sample at 27 MHz for 4:2:2 format (converted from 4:2:0 encoded in MPEG-2:MP@ML, 720x480@30Hz). Also, the whole system operates at 27MHz that can access DRAM one word per 1.5 cycles in the fast page mode (cycle time 40ns). The decoder must output 480 bytes of previous decoded data for display and decode 384 bytes of data in one macroblock period (640 cycles@27MHz). While bi-directional-predicted macroblock is encountered (Fig. 5a and Fig. 5b), we will allocate more bus cycles for the loading of predicted blocks, which has relatively larger amount of data to be transferred. Although in this case we limit the bitstream I/O sustained rate to about 200Mbps, the rate is still far lager than the bit rate specified in MPEG-2:MP@ML (i.e., 15Mbps). For intra macroblock, on the other hand, more cycles will be allocated to bitstream I/O transactions because of the relatively lower compression ratio (Fig. 5c) in this type of macroblock. The display process will not overrun the decoding process because the criterion (3) is met:

$$\frac{858 \times 525}{480} = 938 \geq 900 = \frac{720 \times 480}{384} .$$

4. Simulation Results and Implementation

Fig. 6 shows the simulation results of the buffer occupancy using the fixed-priority dynamic scheduling and the proposed scheme. The test video sequence is "Flower garden" with bit rate 15Mbps and the simulation duration is one-frame time. Also, the simulation models are uninterpreted[2] to reduce the simulation time. Furthermore, we use intra-type data for bitstream input/output and inter-type data (frame picture and bidirectionally field-based prediction) for reference and predicted pictures. Although conditions with such heavy bus load for memory access could hardly occur in real case, it is useful to test the robustness of the arbitration scheme.

Here we fix the size of VLD buffer to 1kbit and observe the occupancy of bitstream buffer and video output buffer. Obviously, in the case adopting the fixed-priority dynamic scheduling scheme both of those buffer are larger than the ones in the case using the proposed scheme. Furthermore, although the buffer requirement in the latter case is smaller, the residual time for header decoding in a frame period is still larger than the one in the former case. It is because there exists less arbitration overhead with the proposed scheme. Table 1 summarizes the results.

The proposed bandwidth allocation scheme for memory access has been verified by Verilog simulation. We will implement this scheme in our MPEG-2 VLSI that is currently developed in NCTU.

5. Conclusion

Concluding, compared to the pure stochastic bus arbitration scheme, the proposed scheme reduces the required amount of internal I/O buffer and the overheads of bus arbitration for our MPEG-2 decoder. The only drawback is the little reduction in bitstream I/O sustained rate. We will implement this scheme in the NCTU MPEG-2 decoder.

6. Acknowledgment

This work was supported by the National Science Council under Grant NSC 79-0414-E009-008 and United Microelectronics Corporation.

7. References

[1] ISO/IEC CD 13818, "Generic Coding of Moving Pictures and Associated Audio," (MPEG-2).

[2] J. M. Schoem, editor, "Performance and Fault Modeling with VHDL," Prentice-Hall, 1992.

[3] Thierry Fautier, "VLSI Implementation of MPEG Decoders," in *Proc. IEEE ISCAS 94 Tutorials*, may 1994.

[4] T. Demura, T. Oto, K. Kitagaki, S. Ishiwata, G. Otomo, S. Michinaka, S. Suzuki, N. Goto, M. Matsui, H. Hara, T. Nagamatsu, K. Seta, T. Shimazawa, K. Maeguchi, T. Odaka, Y. Uetani, T. Oku, T. Yamakage and T. Sakurai, "A Single-Chip Video Decoder LSI," in *ISSCC*

Reproduced with permission of copyright owner. Further reproduction prohibited.

Page 265 of 280

Petitioners HTC & LG - Exhibit 1002, p. 265

Digest of Technical Papers, pp.72-72, Feb 1994.

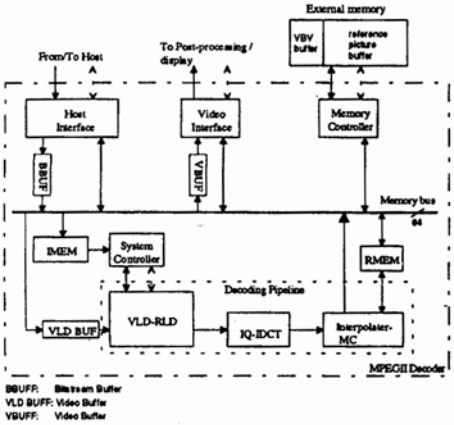


Fig.1 Architecture of the MPEG-2 decoder

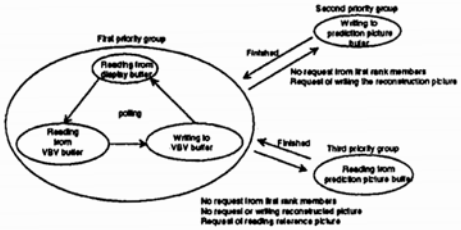


Fig. 3 The bus scheduling scheme proposed by Tatsuhiko Demura, et al.

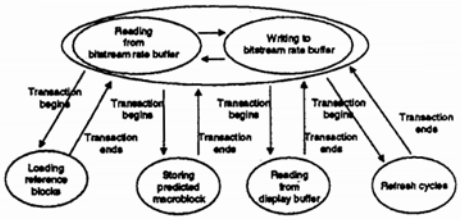


Fig. 4 The state diagram of proposed bus arbitration scheme.

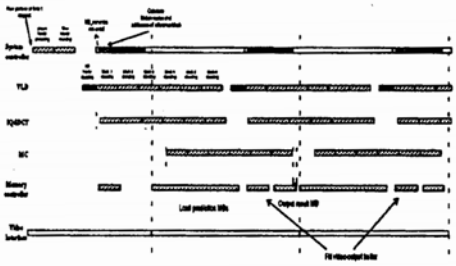


Fig.2 The decoding timing diagram of decoding pipeline (For frame picture, bidirectional field predicted macroblock).

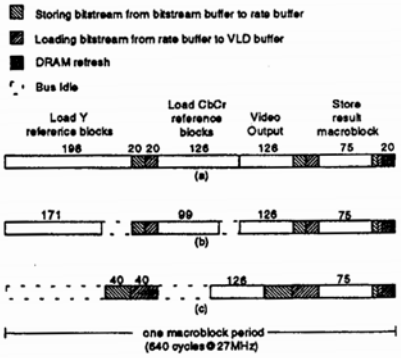


Fig.5 Examples using the proposed bus allocation scheme, (a) for frame picture, bidirectional field predicted macroblock (worst case), (b) for frame picture, bidirectional frame predicted macroblock, (c) for intra macroblock.

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 266

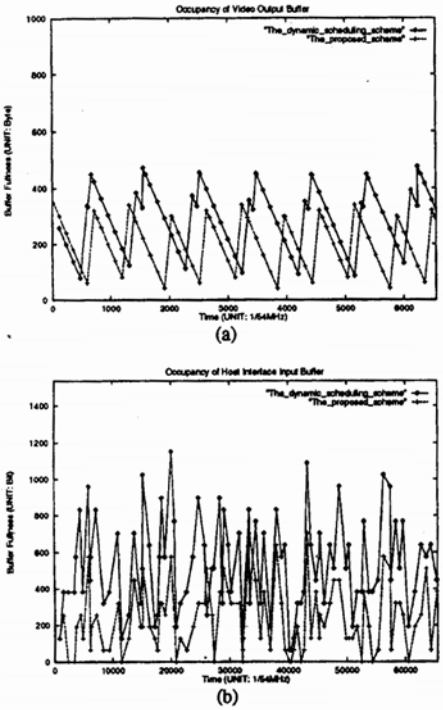


Fig. 6 Comparison of buffer occupancy in both scheduling schemes, (a) Video Output FIFO, (b) Host Input FIFO.

	The Fixed-Priority Dynamic scheduling Scheme	The Proposed Scheduling scheme	Saving
Size of Bitstream Buffer	1200bits	800bits	33%
Size of Video Output Buffer	480Bytes	360Bytes	25%
Residual Time for Header Decoding	20980cycles @27MHz	29610cycles @27MHz	4% (compared to one-frame time)

Table. 1 Comparison of buffer size and residual time for header decoding in both scheduling schemes.

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 267

A Low-Cost Graphics and Multimedia Workstation Chip Set

With just three VLSI parts, our latest workstation class lets designers optimize performance and cost at the system level. Its Hummingbird microprocessor features two-way superscalar execution incorporating two integer units, a floating-point unit, a 1-Kbyte internal instruction cache, an integrated external cache controller, an integrated memory and I/O controller, plus enhancements for little-endian and multimedia applications. Its Artist graphics controller integrates a graphical user interface accelerator, a frame buffer controller, and a video controller on a single chip.

Steve Undy

Mick Bass

Dave Hollenbeck

Wayne Kever

Larry Thayer

Hewlett-Packard

The computer system design approach known as disintegration spurns complex, highly integrated system components in favor of less complex, generic parts. A system house can have different design goals than its component supplier, leading to situations where the component vendor provides features on an integrated part not desired by the system house.

Using standard, off-the-shelf parts lets system designers pick and choose exactly the features they need without having to pay for unwanted ones. Component vendors may also charge premiums for integrated designs, cutting into the profits of the system house, which wants to add value to its products itself. Further, a system house may not want to depend on the availability of a vendor's complicated integrated design when shipping products to customers.

An alternative computer design approach is toward highly integrated parts and systems, a tack we have taken with the low-cost workstations we discuss here. As both a system house and a component vendor to itself, Hewlett-Packard can optimize a design for both performance and cost from a system perspective, giving it more flexibility in deciding where and how to place value-adding features. The system house thus can specify what features must be built into the components to precisely meet overall needs. Schedules, too, are now visible and their risks more controllable. Treating the end product—an entire workstation or server—as a whole rather than just a sum of its parts makes integration another degree of freedom in design optimization. In particular, the goal of the processor design team becomes overall system optimization rather than simply processor subsystem optimization.

A three-chip workstation system

Recently, we introduced a number of entry-level workstations and servers based on the Hummingbird PA7100LC processor. Figure 1 shows a block diagram of one of these computers, the HP 9000 Model 712/60 workstation. Because of integration, this design uses only three very large-scale integration parts. The Hummingbird processor chip connects directly to static cache RAMs and dynamic main memory RAMs. It also connects directly to the other two VLSI parts, named LASI and Artist, via a proprietary system bus. LASI, short for LAN (local-area network) and SCSI (Smaller Computer System Interface), provides a number of built-in I/O connections for the computer—RS-232, 16-bit stereo audio, and a parallel port among others—in addition to the two it was named for. The Artist chip is a graphics subsystem that connects directly to a color monitor.

Integrating so much onto the three VLSI parts was not an arbitrary choice. For example, integrating a memory controller onto the processor chip results in shorter cache miss penalties than

0272-1732/94/$04.00 © 1994 IEEE

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 268

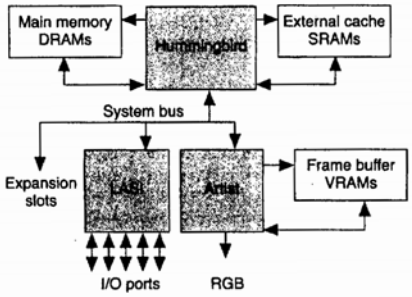


Figure 1. Model 712/60 block diagram.

a nonintegrated solution. This, in turn, enables performance improvements—especially for memory-intensive applications. It also allows the design of systems with smaller caches relative to nonintegrated systems, without compromising system performance. The 712/60 uses a fairly small 64-Kbyte external cache. Including a second integer execution unit on the processor also improves performance. The direct connection between the processor and the graphics controller allows for fast data transfers and increased graphics performance. The net effect of integration on performance is that last year's mid-range workstation performance is now available on this year's entry-level workstation. Figure 2 gives SPECint92 and SPECfp92 benchmark performance ratings for the 60-MHz 712/60 and estimated numbers for the 80-MHz Series 800 Model E45. Also shown is the estimated performance for a system running at 100 MHz.

Integration also reduces costs. Figure 3 shows the single processor board used in the 712/60. The 712/60 uses a fraction of the components used in systems built just a couple of years ago. The number of parts used in our processors has steadily decreased from the first CMOS design completed in 1988. Figure 4a (next page) shows this integration trend for processors, with each rectangle representing one VLSI part. The number of components used in graphics controllers has likewise decreased over the years, as Figure 4b shows. As an example, we have incorporated the RAM digital-analog converter, used to generate video signals, directly into the Artist chip, saving both the cost of an external component and the board area it would have occupied. The LASI chip replaces the many separate components needed to provide the I/O connections expected on a workstation.

Hummingbird integrated processor chip

Hummingbird is the fourth in a series of CMOS PA-RISC processors,[1-5] though in many ways, it is a departure from

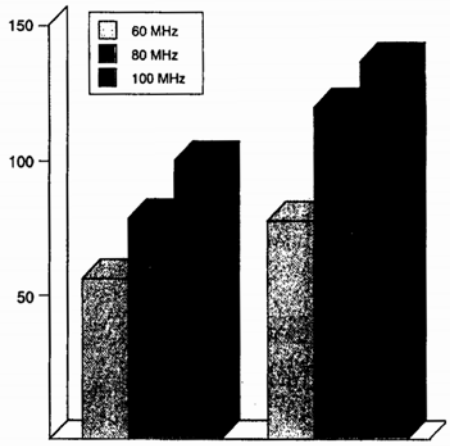


Figure 2. Benchmark performance.

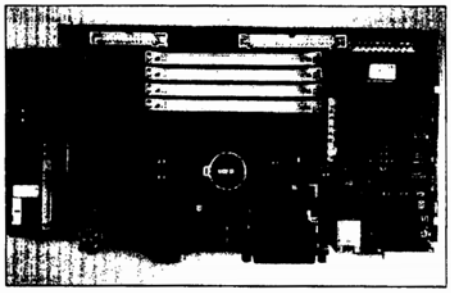


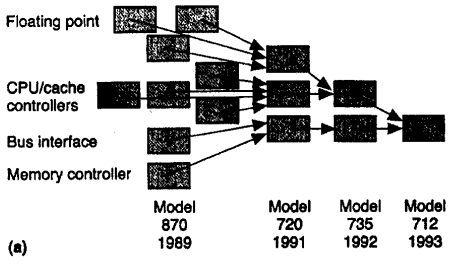
Figure 3. Processor board.

the earlier designs. Rather than concentrating on producing the most performance possible from a given piece of silicon, we designed Hummingbird to be the most cost-effective solution without compromising performance. Hummingbird also uniquely integrates the memory controller, I/O bus controller, and cache controller onto the processor chip.

Design goals. Hummingbird's several design objectives are not just isolated component goals, but are constraints we derived by carefully considering the needs of the entire computer system. For example, the choice to place the memory controller on the processor chip actually increases the cost

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 269



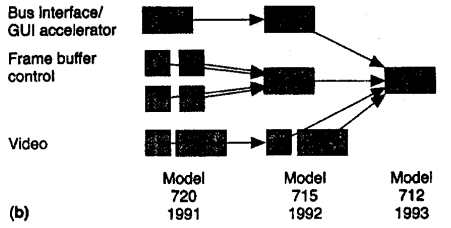


Figure 4. Integration trend: PA-RISC processor (a); graphics (b).

of the processor, but reduces the cost of the system. The goals include

- *Reduced cost—essential for competing in the very cost-sensitive, entry-level workstation market.* Integration of the memory controller helped lower system costs. Equally important, however, was the reduced cost cache organization that we implemented and support for industry-standard SRAMs, DRAMs, and memory SIMMs.
- *Uncompromised performance—considered system wide.* It was important that the cost objective not compromise processing power. Although Hummingbird boasts impressive integer and floating-point performance, it also has features to support high-performance graphics and multimedia applications. Included are a low-latency cache and memory system, new functional units and instructions, and an efficient system bus connection.
- *Inherently scalable—creating an easy upgrade path.* Hummingbird is scalable in clock rate, external cache sizes, main memory sizes, system bus clock ratios, and DRAM timing parameters.
- *Reduced power—achieved largely by using gated clocks and by eliminating dynamic circuit elements.*
- *Architecturally compliant—making Hummingbird completely compliant with the PA-RISC architecture.* Backward compatible with existing implementations, it includes extensions to improve the performance of little-endian and multimedia applications, and connection to standard I/O buses.
- *Improved manufacturability.* We wanted to reduce manufacturing costs and times by using standardized test methodologies and dedicated diagnostic circuitry.

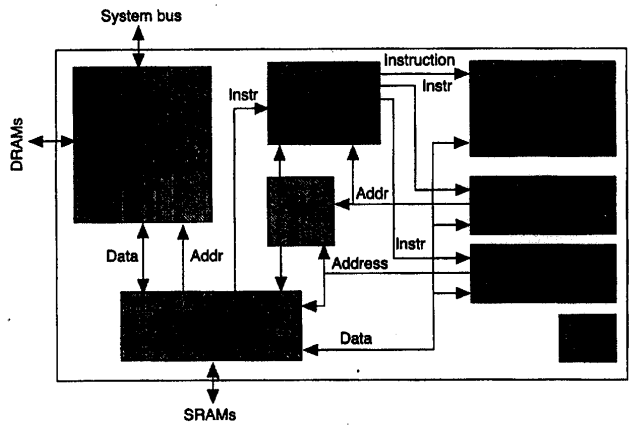


Figure 5. Hummingbird block diagram.

Features. The Hummingbird CPU design leveraged many of its core technologies and features from the PA7100. Thus it has a pipeline design very similar to that of the PA7100, although we made several minor changes. As Figure 5 shows, Hummingbird is a two-way superscalar implementation incorporating two integer execution units, a floating-point execution unit, an internal instruction cache, a controller for external cache, and a main memory and I/O controller. It interfaces directly to static cache RAMs, as well as to standard DRAMs.

Dual-integer superscalar execution. Hummingbird has three execution

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 270

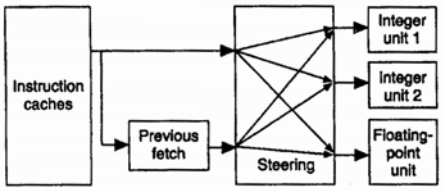


Figure 6. Instruction steering.

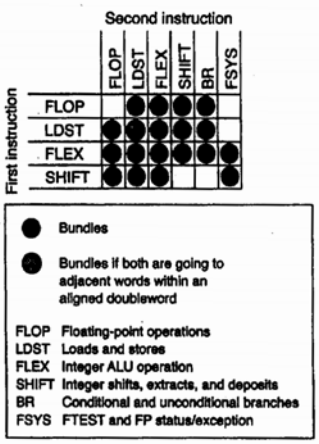


Figure 7. Superscalar bundling chart.

units. The first executes integer arithmetic, shift-type and branch instructions. The second executes integer arithmetic and memory reference instructions (both integer and floating-point). The third executes all floating-point arithmetic instructions.

The ability to execute two integer instructions simultaneously is a new feature for PA-RISC processors, requiring implementation of a second integer arithmetic logic unit. Through careful redesign of the integer datapath and by shrinking the translation look-aside buffer and other blocks from the PA7100, we made room for the second ALU in the chip floor plan. To keep costs down, we did not make the second execution unit as flexible as the first. Only the first execution unit, for instance, has the barrel shifter needed for shift and merge instructions. This relatively small investment in hardware lets us accelerate integer-only software by superscalar execution. The earlier PA7100 processor accelerated mostly floating-point applications by superscalar execution.

Every cycle, the instruction steering block (Figure 6) may issue an instruction to two of the three execution units. On each cycle, the instruction steering block fetches two instructions from the instruction cache. Depending on whether one or two instructions previously went to execution units, those instructions could be several instructions ahead of the program counter. Immediately after fetching instructions, the steering block examines them (along with any instructions from the previous fetch that have not yet executed) to determine which execution units they are to be directed to and whether two instructions may be bundled, that is, issued in the same cycle.

Several considerations arise for determining if two candidate instructions may be bundled. First is functional unit availability. For instance, with only one shifter implemented, only one shift instruction may issue per cycle. This does not tend to limit performance, as shifter use occurs less frequently than does ALU use. Figure 7 shows the combinations of instructions that can be bundled.

The second consideration concerns register dependencies. Even though the instruction steering block can bundle two addition instructions, it cannot do so if the second uses the result of the first. Also, it will not bundle some instructions, especially those that modify global resources such as the TLB or control registers, due to the complexity in determining dependencies. Branches are not bundled with the following instruction. The PA-RISC architecture[6] incorporates a concept called nullification, in which certain instructions can cause the following instruction to be nullified (not executed). The instruction steering block will not bundle instructions that can cause nullification with the following instruction. Like many architectures, PA-RISC uses delayed branching, where the processor fetches the instruction immediately following the branch, regardless of whether or not the branch is taken. We call this instruction the delay-slot instruction; it is not bundled.

Hummingbird has no address alignment constraints on bundles. It also allows the bundling of two load or store instructions referencing adjacent words in memory so long as they do not cross a double-word boundary. In this case, only a single double-word address—generated by the second integer execution unit—suffices for both instructions. Therefore, the two loads or stores may be bundled together. We designed special hardware to detect this case quickly enough to make the decision to bundle. Code that performs loads and stores to linear address ranges—especially procedure calls and context switches—will see an acceleration by this feature.

Cost-effective floating point. The floating-point unit (FPU) design for Hummingbird supports two goals: reduced system cost compared with PA7100-based systems and high performance for graphics. Floating-point performance is critical to

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 271

Table 1. Floating-point latencies and issue rates.

	Single precision			Double precision		
	Latency	Issue rate	Stalls	Latency	Issue rate	Stalls
Add/subtract	2	1	0	2	1	0
Multiply	2	1	0	3	2	1
Mpyadd/mpysub	2	1	0	3	2	1
Divide	8	8	7	15	15	14
Square root	8	8	7	15	15	14

graphics performance in PA-RISC systems because a significant amount of graphics processing takes place in the CPU. Fortunately, the PA7100 FPU provided an excellent starting point for performance, so we focused our design decisions on reducing cost without affecting performance for the targeted market. We wanted area reduction in the FPU to enable the integration of new features on the chip, such as the memory controller. We needed reduced power to minimize system power supply costs and cooling fan noise.

In graphics processing, only single-precision (32-bit) floating-point performance is critical. We could thus perhaps sacrifice some double-precision (64-bit) floating-point performance to make room on the chip for the memory controller. The PA7100 FPU has separate units for multiply, divide/square root, and ALU operations. Of these, only the multiplier architecture promised substantial area savings without a major redesign effort. For Hummingbird, we cut the array in half so that single-precision operations make one pass, while double-precision operations circulate their partial products through a second time before the final addition and rounding. Double-precision multiply is now a three-cycle operation, and a new operation can start every two cycles. This change reduced the multiplier power consumption because of the reduced amount of circuitry active on any cycle. Single-precision performance is unaffected and remains a two-cycle operation, where a new operation can start each cycle.

The change to the multiply latency and issue rate brought up an issue in the control logic. The two-cycle latency operations fit inside the normal five-stage pipeline of the CPU. Operations with longer latency require more control logic to avoid pipeline stalls. We elected to take an unconditional pipeline stall on any operation longer than two cycles. In practice, data dependencies often force these stalls anyway, so the performance impact is quite small, even for double-precision floating-point applications. By reducing the control logic we also save area. The number of register dependency comparators fell by 30 percent, and the random logic control core cell count dropped by 15 percent. Treating the long-latency operations in a simple, uniform way greatly simplified the controller design task. Table 1 summarizes latencies and issue rates.

The biggest opportunity for power savings in dynamic circuits comes from making evaluation conditional. This way, Hummingbird only draws current from the supply when precharging the logic following an evaluation cycle. The floating-point data path is composed almost entirely of dynamic logic to satisfy speed and area constraints. Given Hummingbird's lower frequency goal, we buffered the clocks into the three floating-point math units and qualified them with control signals. The three units have separate power switches. Whenever a valid floating-point operation begins, a power token gets passed along with the data, flowing through the pipeline and causing each stage to evaluate only on the cycle it is needed. With a continuous stream of flops, all the stages are active at the same time. However, whenever there are states on which new flops do not start, only those pipe stages with real work to do are active. Even in most floating-point benchmark programs, many states arise in which at least part of each math unit can remain inactive.

Cost- and performance-optimized caches and TLB. Like its predecessors, Hummingbird cycles its external cache at the processor frequency, allowing load instructions to execute every cycle without penalty. Unlike its predecessors, its external cache is combined, containing both instructions and data, and has a small (1-Kbyte) internal instruction cache. Even though we designed Hummingbird for low-cost systems, we had several reasons for retaining a single-cycle external cache. First, we felt that the silicon area on Hummingbird was better spent on other features (such as a second integer ALU and a memory controller) than on a relatively small data cache. Second, for low-cost systems running at moderate frequencies, our design does not require aggressive—costly—SRAM specifications. In fact, systems based on relatively slow 12-ns parts can run up to 66 MHz. The design required only 12 such parts. Lastly, the external cache organization allows for a greater degree of scalability and flexibility than a fixed-size internal cache.

We added the internal instruction cache to supply the needed instruction fetch bandwidth, as both instruction and data caches can be referenced in a single cycle. The caches are virtually indexed and physically tagged. The external cache has a 32-byte cache line size, while the internal cache has an 8-byte cache line size. Developers can configure the external cache size between 8 Kbytes and 2 Mbytes.

Hummingbird implements a two-level instruction cache hierarchy. The first level is the 1-Kbyte internal cache and the second level is one half of the external cache. The first level is a strict subset of the second. Both can provide two instructions every cycle. If a typical operation detects a first-level instruction cache miss, it forwards the instruction fetch to the second-level cache. If the second-level access hits, the cache controller forwards the double-word of instructions to

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 272

the instruction steering logic while also sending it to the first-level cache for insertion. If the second-level cache indicates a miss, the memory controller begins handling the miss.

Load and store instructions represent only approximately 40 percent of the total instruction mix for PA-RISC processors. Consequently, bandwidth is available to the external cache, which contains both the data cache and the second-level instruction cache. Taking advantage of this extra bandwidth is a prefetching machine that copies instructions from the second-level cache to the first (see Figure 8). Hummingbird will perform this prefetch every cycle that the external cache is not busy satisfying a data reference. The prefetch machine attempts to stay ahead of the program counter so that a first-level miss will not occur. At times, enough data references block the external cache that the prefetching machine cannot keep up with the program counter. If so, the prefetch machine advances to the current instruction fetch address to make future prefetches useful.

Prefetched instructions go into a two-entry queue of instructions to be written to the first-level cache. Writes into the first-level cache from this queue proceed in parallel with reads from the first-level cache. An instruction fetch may use an instruction out of this queue without penalty. If a first-level cache miss is detected at the same time a prefetch is in progress for that address, the instruction goes directly to the instruction-steering logic from the external cache, reducing the normal instruction miss penalty by one cycle. Branches take advantage of this feature by beginning a prefetch to the target of the branch immediately after issuing the target address to the first-level cache. After a branch is taken, the prefetch machine will begin prefetching from the new program counter location.

The data cache on Hummingbird is a conventional single-level external cache. Reads from the external cache require a single processor cycle—even at 100 MHz. Writes, however, require two consecutive cycles. Since store instructions generally must read the tag portion of the cache before writing the data portion, the design uses store pipelining. This optimization technique entails using separate address lines for the external tag and data SRAMs. This, in turn, allows the cache controller to read the tag for a given store at the same time it writes the data for the prior store. Thus store instructions effectively use only two cycles of cache bandwidth employing standard asynchronous SRAMs. The data cache uses another store optimization that involves only stalling the pipeline if a instruction bundle containing a store (which will begin a two-cycle cache sequence) immediately precedes a bundle containing a data reference. In this way, the data cache can effectively hide the extra cycle of cache bandwidth needed by a store instruction if the instructions executed on the following cycle do not need to access the cache.

The advantage of integrating a memory controller on the same chip as the CPU becomes apparent when second-level

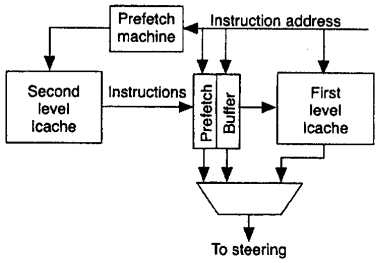


Figure 8. Instruction prefetching.

instruction or data cache misses occur. The cache controller is tightly coupled to the memory controller: the memory controller detects and begins handling a cache miss at the same time the CPU detects the miss. The cache controller uses several techniques to reduce the penalties associated with cache misses. It uses instruction streaming on second-level instruction cache misses, which allows the CPU to continue executing as soon as the first, or critical, double-word arrives from the memory controller. It writes the double-word to both levels of instruction cache while the CPU steps, or continues execution. This will occur for each double-word until all are written. Another feature, called stall-on-use, lets the CPU continue executing after it detects a data cache miss on a load instruction.

The cache controller can handle up to two outstanding cache misses at a time. Even though the CPU will stop stepping after detecting a second cache miss while a cache miss is in progress, it will resume stepping as soon as the cache line move-in for the first miss completes. This feature allows the memory controller to optimize misses to consecutive cache lines.

The virtual memory system for Hummingbird is essentially the same as that on the PA7100. We reduced the TLB from 120 entries to 64 to save area, although it remains fully associative. The TLB also contains eight block TLB entries for mapping large (512 Kbyte to 64 Mbyte) contiguous address ranges.

Tightly coupled memory system. The design of the memory system reflects the system-level design goals of low cost and power with high performance and scalability. We translated the system design goals into the following objectives for the memory system. The memory system should use the lowest cost commodity parts available at any given time. It should enable versatile system design by allowing a wide range of possible main memory sizes for scalability. The memory system should be capable of maintaining good performance levels, at the lowest possible cost, over a wide

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 273

Table 2. Memory system performance (at 60 MHz).

Transaction	Value
Miss occurrence to critical data	
Data cache miss	7 cycles
Instruction cache miss:	
Prefetch hit	4 cycles
Prefetch miss	7 cycles
Cache move-in bandwidth	
Page mode	160 Mbytes/s
Nonpage mode	107 Mbytes/s

range of system frequencies. The memory controller design should be simple, helping to achieve low development cost through first-time correctness, small area, and ease of testing.

Integrating the memory controller onto the same die as the CPU provides the memory controller with access to many important CPU internal resources. This enables performance gains that would not be possible in a nonintegrated solution. For example, the memory controller can eavesdrop on the real page number produced by the TLB. It can use this information to drive addresses to the DRAM before the occurrence of a miss is known. This speculative address issue saves a cycle on memory latency for cache misses. Integration also allows more effective use of the fast page mode of the DRAM than would otherwise be possible. Due to the early detection of cache misses, our design can in some cases avoid DRAM precharge penalties that a stand-alone memory controller could not. To further reduce miss penalties, the memory controller returns missing data to the cache in a critical-word-first fashion.

The memory controller implements an instruction prefetching algorithm. This prefetch mechanism occurs between memory and the instruction caches, and is in addition to the second-level cache to first-level cache prefetching described earlier. The algorithm very effectively reduces second-level instruction cache miss penalties, due to the proximity of the prefetch buffer to the CPU core. In the case of an instruction prefetch buffer hit, data can be sourced to the CPU and execution can continue within four CPU cycles of the detection of the second-level instruction cache miss.

The memory controller shares a four-entry transaction queue with the I/O controller. The transaction queue in many cases allows the CPU to continue execution, while the memory controller performs the queued transactions. When a cache miss occurs in which the cache line to be replaced has been modified, the memory controller queues the modified data while fetching the missing data from memory. Only then does it post the modified data to memory. Table 2 shows some performance characteristics of a typical Hummingbird memory system.

The memory controller is versatile enough to allow use of state-of-the-art commodity parts throughout the expected lifetime of the product. Industry-standard DRAM SIMMS form the system's main memory. The main memory data bus is 72-bits wide. Eight of the 72 bits serve for an error correcting code that can correct any single-bit error and detect any double-bit error. Since SIMMs of different types require different address bits to be multiplexed into the row and column addresses, the memory controller implements the address multiplexing function in a programmable fashion, memory card by memory card. This approach maximizes flexibility in the type of memory that may be installed in the system.

We built the memory controller with system scalability in mind. Systems may be built with as few as one, and as many as 16 SIMM slots, providing possible main memory sizes of 4 Mbytes to 2 Gbytes. Delays between DRAM address, control, and data edges are programmable, allowing for tailoring the speed (and cost) of the DRAM used for main memory to system requirements. The design supports DRAMs that implement an extended-data-out mode, providing superior page mode bandwidth at higher system frequencies.

Some systems may require buffering of some or all of the DRAM control lines. All DRAM control lines have programmable sense—active high versus active low—for this reason. The sense of each of the control lines may be programmed independently, allowing maximum system design flexibility.

Although the Hummingbird system caches are smaller than those in previous systems, the cycles per instruction contributions due to cache misses are on the same order as in systems with larger caches. By drastically reducing miss penalties through an integrated approach, our design maintains good performance at a lower system cost.

High-bandwidth I/O system. The I/O system uses a 32-bit bus onto which addresses and data are multiplexed. This substantially lowers the pin count and cost from a nonmultiplexed bus, thus allowing integration of the I/O controller onto the same die as the CPU and memory controller. Tight coupling between the I/O bus and the CPU and memory controller maintains performance, as does an efficient I/O protocol.

The I/O controller performs I/O reads and writes on behalf of the CPU, and direct-memory access on behalf of masters residing on the I/O bus. A transaction queue, shared with the memory controller, receives all CPU I/O requests, allowing the CPU to continue execution, in most cases, while the I/O transaction proceeds. DMA requests always insert directly into the head of the transaction queue. Addresses issue in a speculative manner to the DRAM address bus from the I/O bus when the I/O bus is not granted to the CPU. This not only benefits performance, but also allows the memory controller to handle DMA in the same way that it handles memory requests from the CPU, reducing the design complexity.

We paid particular attention to performance at the system level while designing the I/O system. For example, the

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 274

processor retains access to main memory while the I/O bus is granted to an external master that is performing DMA. The memory controller alternates between memory requests from the CPU and DMA requests from the I/O system.

The ability of the CPU to quickly move data from main memory to the I/O system is essential for good system graphics performance. The memory and I/O systems work together to allow overlapped execution of processor reads from memory and processor writes to I/O devices. The CPU can attain a bandwidth of 50 Mbytes/s from main memory to I/O by using this technique, without requiring special block move or DMA hardware.

The design of the I/O system also reflects system scalability. We structured the I/O bus to operate properly to a frequency of 40 MHz. The CPU-to-IO bus frequency ratio is programmable to either 2:1 or 3:1. For maximum system design flexibility, we left the system arbitration logic off chip.

New architectural extensions for flexibility and performance. The Hummingbird CPU is completely compliant with the PA-RISC 1.1 architecture.[6] Existing code will automatically be accelerated by the performance features we implemented, although newer compilers take better advantage of the superscalar abilities of the CPU. Besides being backwards compatible, Hummingbird also implements several new extensions to the architecture: little-endian addressing, uncachable memory pages, and multimedia-oriented instructions.

Hummingbird supports both big-endian addressing, which all previous PA-RISC processors implement, and little-endian addressing. The difference between the two modes specifically deals with the order of bytes within larger data quantities and can be conceptualized as whether the most significant, or leftmost, byte in a 4-byte register will be loaded from or stored to byte address 0 or 3. This may seem trivial, but many programs implicitly assume one byte order or the other, therefore representing a roadblock to porting software between computers having different byte-endian addressing. We wanted to tap into the large pool of software written for little-endian processors but still remain compatible with existing PA-RISC code. Thus we added a mode bit to the PA-RISC processor architecture that selects between big- and little-endian byte addressing. Called the E bit, we put it into the processor status word so that it can vary from process to process. A single workstation thus can run both big- and little-endian applications concurrently. The dynamic nature of this bit dictated that memory be one endianess or the other (chosen to be big-endian on Hummingbird) and that data quantities be either byte-swapped or not on transfers between the CPU's registers and memory. In this way, both big- and little-endian software consistently treat a datum correctly that they are processing.

Certain types of software can be better optimized if some memory pages never get loaded into the data cache, for example, a device driver that communicates with an I/O device by reading and writing messages in main memory locations. If memory is always cachable, the driver must execute time-consuming cache flushes to prevent memory writes caused by cache line replacements from corrupting the I/O device's messages. Hummingbird supports uncachable pages because we added another bit, called the U bit, to each TLB entry. This bit controls whether a data cache miss to memory space will cause a move-in of the target memory line or not.

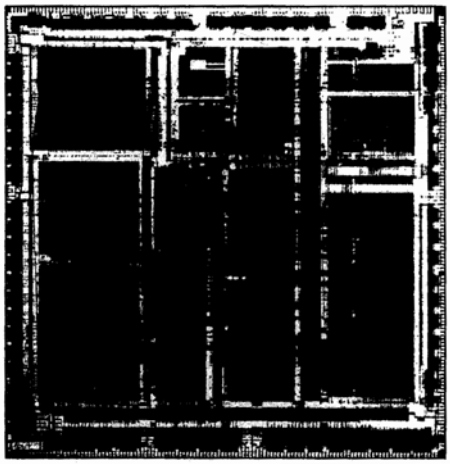


Figure 9. Hummingbird die.

An active area of multimedia research at Hewlett-Packard involves the algorithms used to decompress real-time audio and video information. Performance research suggested that many of the algorithms studied frequently used a few operations: addition and subtraction with either modular arithmetic or saturation, taking the average of two numbers, and multiplication by a small constant. Saturation clips the result to the largest value on positive overflow or clips to the smallest value on negative overflow. We have speeded up all these operations in Hummingbird. Each integer execution unit can execute two of these operations together, meaning that with the two integer units, four operations can occur simultaneously, thus accelerating the various multimedia algorithms substantially. These multimedia-motivated enhancements added insignificant (less than 0.2-percent) silicon area, while improving performance substantially and without requiring a dedicated multimedia accelerator chip.

Figure 9 shows a die photograph of Hummingbird; Table 3 (next page) gives some of the particulars about the chip design.

Reproduced with permission of copyright owner. Further reproduction prohibited.

Page 275 of 280

Petitioners HTC & LG - Exhibit 1002, p. 275

Table 3. Hummingbird details.

Parameter	Value
Transistors	900,000
Die size	14×14 mm
Metal layers	3
L_{eff}	0.61 µm (NFETs) 0.66 µm (PFETs)
Frequency	0-100 MHz
Power (60 MHz)	9.0W (worst case) 6.8W (typical)
Package	432-pin CPGA 1.8×1.8-inch
Supplies	5V, 3.3V

Artist integrated graphics chip

Coupled to the Hummingbird processor is a single-chip graphics system that complements the capabilities of the processor (see Figure 1).

Design goals. We designed the Artist graphics system to perform well in three areas:

- *Fast 2D graphical user interface.* Nearly every computer user has grown accustomed to running some sort of GUI. Speed is important for general user productivity.
- *Efficient 3D graphics.* HP's PowerShade software enables 3D graphics on even the least expensive workstation systems.
- *Digital video decompression.* Most of today's solutions require significant additional hardware. To meet the cost goals of the target workstation, we needed to provide this capability without additional hardware cost.

While it would be possible to design a graphics subsystem without considering other aspects of the system, the result would most likely be more expensive and slower than a system-oriented approach. Design and partitioning tradeoffs between GUI, 3D graphics, and decompression considerations let us place functionality where it can be provided most efficiently. In most cases, our graphics system design included performance margins to allow for the inevitable improvements in CPU speed.

Graphical user interface. Fast GUI performance is a good example of a system requirement involving hardware features in both the CPU and the graphics subsystem. GUIs use a number of low-level primitive routines that account for a majority of the time spent in typical user interactions. Accelerating these routines with a minimum of hardware to keep costs low presents the real problem.

Our criteria for including special features in the CPU were:[7]

- Is the proposed function best implemented in the CPU, or could the same function be implemented just as effectively in the graphics subsystem?
- Does the envisioned enhancement fit within the general CPU architecture in an economical fashion? (We never considered adding significant cost to the CPU.)
- Does the proposed enhancement provide a significant performance advantage?

One important operation in a GUI is passing data from main memory to the display for painting backgrounds or filling patterns. Graphics hardware cannot alone perform this operation: the CPU and memory system must also be involved. Our approach has the CPU/memory system providing a fast path from memory through the floating-point registers to the system bus (50 Mbytes/s) and the graphics hardware having adequate frame buffer write bandwidth (96 Mbytes/s) from the system bus.

The CPU also had to be able to send data quickly from CPU integer registers to graphics hardware. This sends low-level GUI primitives to the GUI accelerator. High bandwidth streams of CPU writes to I/O addresses have become standard in PA-RISC processors. Providing this capability involved reducing processor penalty cycles associated with I/O references and designing efficient mechanisms to transfer data between the CPU's connection to the memory and I/O controller and the system bus where the graphics controller resides.

Early investigations clearly showed that our cost constraints would not allow us to use hardware acceleration for all other GUI routines. Instead, we accelerated only those routines deemed most important: vectors, rectangles, screen-to-screen block moves, memory-to-screen block moves, text, cursor motion, and pixel formatting. Hardware support for vectors includes a vector drawing engine that can be loaded with a single word per connected polyline segment, while the CPU formats the word and performs the write-to-I/O space. The graphics hardware limit for vector drawing is over 2 million vectors/s.

Found in such areas as window backgrounds and boundaries, rectangle fill is another commonly used operation. Since VRAMs have a fast block mode to draw large, constant-color regions, we added hardware support for this function as well. Software specifies rectangles via a pair of writes. The hardware takes advantage of the four-column block mode to achieve a 425-million pixel/s peak rectangle fill rate with 2-Mbit VRAMs, and 850 million pixels/s peak with 4-Mbit VRAMs.

Since nearly all applications include text, user productivity demands fast text painting and scrolling. Artist lets the CPU provide just four words to define a 6×13 character, then optimizes the VRAM accesses to maximize text performance. An Artist chip can paint over one million characters/s.

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 276

Another aspect of fast text scrolling is the ability to move pixels quickly from one location on the display to another. We also use this capability to move entire windows on the display. Because moving pixels from one location to another would be inefficient if all the data had to go through the CPU, Artist includes hardware to handle this operation. With this support, Artist can achieve a block move rate of 47 million pixels/s within the frame buffer.

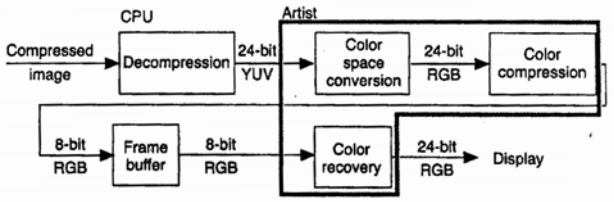


Figure 10. Image decompression pipeline. [Red-green-blue (RGB) and yellow-ultramarine-violet (YUV) are competing color schemes.]

To make all these features work seamlessly and efficiently for the GUI software drivers, we incorporated a number of addressing and data modes. These permit pixel accesses to be any of several pixel configurations (one, four, or 32 pixels per 32-bit word) with arbitrary frame buffer data alignment. Pixel replication can extend single-bit pixels to full depth; either ordered dithering or color compression can reduce 24-bit pixels to eight bits.

A hardware cursor maximizes GUI interactivity by allowing a cursor that does not affect the image bitmap. A hardware cursor can save many of the system CPU cycles spent on the GUI.

Three-dimensional graphics. Consistent with the system-design criteria used with GUI acceleration, Artist offers features to aid in the display of 3D data sets. These include a hardware vector rasterizer for accelerating wireframes and dithering and color compression for displaying 3D solids.

Fast memory-to-frame buffer writes help when double-buffering is required. Software can draw images to a virtual window in main memory, then quickly write them to the frame buffer when complete. The CPU's dual-integer ALUs help 3D graphics solids rendering into main memory in addition to aiding general-purpose processing. CPU floating-point enhancements for graphics, including fast clip checking and parallel multiplication and addition, allow very efficient vector vertex calculations.[7]

Multimedia. A new use of graphics hardware is for the display of images or image sequences that had previously been compressed. To make image retrieval interactive and real-time video sequencing possible, image restoration must be quick. Typically, this process includes variable-length decoding, inverse quantization, inverse discrete cosine transformation, and color-space conversion, as for example, in MPEG video decompression.

System-level design is especially helpful with digital image/video decompression. Since the CPU can do most of the full-motion digital video decoding with some instruction set tuning, dedicated hardware need not be added. Putting the last step of the decompression process (YUV-to-RGB color space conversion) in Artist further improves decompression performance. (See Figure 10.)

Figure 11. Dithering (top) versus color recovery (bottom).

Artist has circuitry to convert the color-space and to color compress the image into its 8-bit frame buffer. The colors are restored as part of the video refresh process; they are true color and appear to be 24-bits deep. The resultant images are much better that ones generated using dithering, a common technique (see Figure 11). The result provides real-time, decompressed, true-color video images on the display with only an entry-level, 8-bit hardware configuration.

Features. Bringing such advanced capabilities into widespread use requires a cost-effective solution. The graphics subsystem described here incorporates acceleration for GUIs, 3D graphics, and digital video with RAM control and video refresh in a single custom VLSI chip. When coupled with four or eight VRAMs (depending on the resolution of the display), this chip provides a complete workstation graphics hardware subsystem. (See Figure 12, next page, for its block diagram and Table 4 for its performance highlights.)

To minimize costs, we put the entire graphics system (except video RAM) on a single chip. Most graphics controller chips require external video clocks or digital-to-analog converters, but we included these in Artist to keep the parts count low. As Figure 13 shows, Artist consists of seven main blocks:

Bus interface/FIFO. Artist connects to the 32-bit multi-

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 277

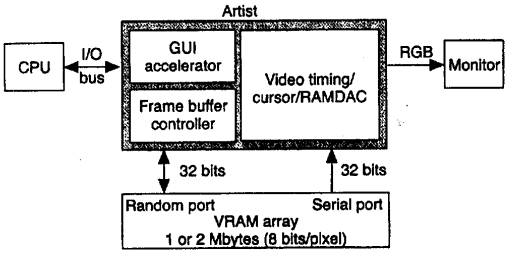


Figure 12. Graphics system block diagram.

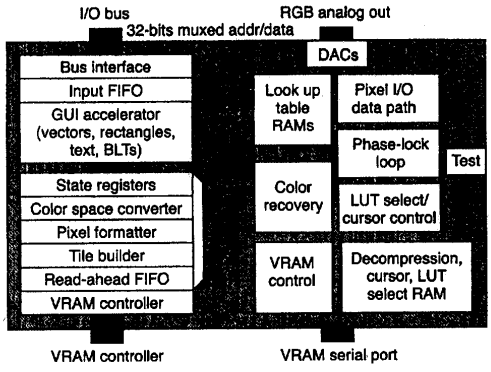


Figure 13. Artist chip block diagram.

plexed address/data system bus connecting the CPU, graphics, and I/O chip. Bus cycles run up to 40 MHz. Artist can accept either one or two data transfers per address cycle, making the peak available bandwidth over 100 Mbytes/s. A 32-deep first-in, first-out memory buffers transactions directed to various parts of the chip.

GUI accelerator. The GUI accelerator consists of an ALU connected to seven registers that manipulate display address and two registers that generate display data. Because these registers operate in a master/slave configuration, one operation can proceed while the next is being set up. Accelerated GUI functions include vector stepping, rectangle filling, text painting, pixel block moving, and lookup table writing.

Address/color formatter. At the output of the GUI accelerator is the address/color formatter that maps graphics data into the frame buffer. This process includes handling various pixel depths, plane masks, color spaces, and data alignments required by the software drivers. Contained in this block are the color converter, color compressor, dither unit, data barrel shifter, and lookup table and cursor data mapper.

Table 4. Artist performance.

Parameter	Value (per second)
Large rectangle fill (peak)	850M pixels
10-pixel, randomly oriented	2.1M vectors
10×10 rectangles	1.7M
Text characters (6×13 pixel)	1M
FB BitBit (unaligned)	47M pixels

Programmable VRAM controller. A VRAM controller at the output of the ACF accesses the random-access port of the VRAMS and initiates data-transfer cycles for updating the VRAM shift registers. Some of the timing parameters are programmable to maintain high levels of performance even when running at a slower clock frequency. Page mode cycles are 37.5 ns, with a clock frequency of 80 MHz. Making extensive use of block mode writes provides further performance optimization.

Video timing generator/PLL. A necessary part of any display controller is the video timing generator. The one built into Artist has programmable timing parameters, including the dot clock frequency itself. Artist reads in lookup-table select bits for each scan line during the horizontal blanking period prior to the display of that line. It supports a wide assortment of resolutions and refresh rates, from 640×480 pixels to 1,280×1,024 pixels with 72-Hz refresh.

Color recovery. Before the video refresh data reaches the lookup tables, it can pass through the color recovery unit. Whether the colors are recovered depends on whether the lookup-table selection bit matches the color recovery enable bit. The lookup tables make a small amount of correction to achieve the final image.

Lookup table/DACs. There are two lookup tables, each with a configuration of three 256 entries. Either all three RAMs can use a single 8-bit index in indexed (pseudocolor) mode or three separate indices can provide true-color decompression mode. Cursor data inserts into the video data stream after the lookup tables so it does not interfere with the images on the display. The DACs have a 75-ohm output so they can have a direct electrical connection to the display.

Chip details. All this circuitry fits on a die measuring 9.7×12 mm in a 208-pin package (See Figure 14). A digital flat-panel port requires a 240-pin package. Table 5 provides additional details about the Artist chip.

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 278

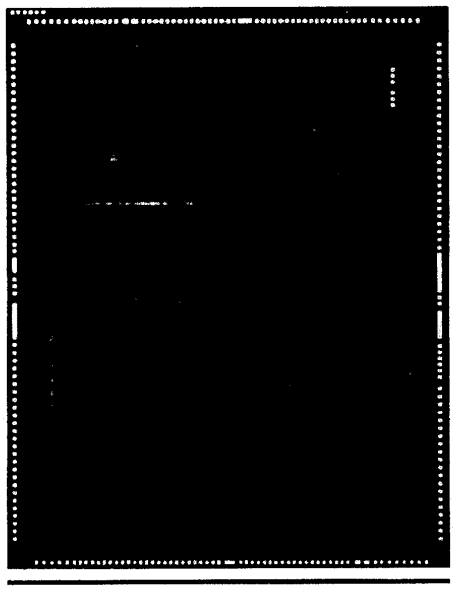


Figure 14. Artist die photograph.

Table 5. Artist details.

Parameter	Value
Transistors	525,000
Die size	9.7×12.1 mm
Metal layers	3
L_{eff}	0.61 µm (NFETs) 0.66 µm (PFETs)
Frequency	40-80 MHz (control) 25-135 MHz (video)
Power	3.5 W (worst case)
Package	208-pin QFP/240-pin QFP
Supplies	5V, 3.3V

Chip design methodology

Our low-cost goals drove several aspects of the chip designs for the initial systems. The cost of a high performance package becomes a significant portion of the delivered part cost. Reducing power dissipation was also a key consideration during the initial design phase.

The leveraged CPU design made heavy use of local two-phase nonoverlapping clock generators. We migrated this design to one that included a qualifier for each local clock phase, for idling circuits when not active, an especially important consideration for global bus drivers. Thus, for example, Hummingbird does not update or read registers unnecessarily. It uses a custom design approach in large, regularly structured blocks such as RAM, DACs, and most of the data path.

We also designed a special 432-pin ceramic pin-grid array for Hummingbird. Our power reduction strategies enabled us to use a package without bypass capacitors, reducing package and assembly costs. Our Artist packaging strategy involved using commonly available, inexpensive packaging.

For both chips, we synthesized control blocks from both behavioral descriptions and programmable logic array-style equations. We used a three-layer-over-the-top router for composing the artwork for these blocks, a departure from our previous PLA-style designs which markedly improved area efficiency. We used timing analysis and circuit simulation to find paths that needed optimization or custom circuits.

We incorporated an aggressive diagnostic capability into Hummingbird that involved piggybacking internal signals onto the system bus during its idle states. By presetting a signal group before running a test, the user can dump all the critical signals, including instruction and data addresses, instructions, and bundling information, virtual translation information, memory and I/O transaction information, and more. These diagnostic signals are driven transparently through the pin driver from their sources at twice Hummingbird's internal frequency.

Artist contains signature generators in several key positions to isolate failures to a single component. A signature in the bus interface verifies proper operation to the graphics system. Signatures on the VRAM random access and serial ports separate VRAM from Artist failures. A signature taken at the input to the DACs helps identify faults in the Artist video section. A crude ADC on the analog video port can identify major errors in the DAC output. Hummingbird also includes a signature generator to accelerate manufacturing tests of the internal instruction cache.

We also included IEEE 1149.1 compliance to help lower board test manufacturing cost. In addition, for Hummingbird, we merged our previous serial test methodology to allow sampling of all scanable nodes on the processor on a specific clock cycle and scanning the sampled values out of the chip while the system continues to run. This greatly aids diagnosis of failures on prototype systems.

WITH THE PA7100LC VLSI CHIP SET, Hewlett-Packard has pursued a path of high system integration to maximize both cost effectiveness and raw processing power. While performance continues to be an important factor, other design goals such as low cost and low power came into play

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 279

for this particular design. By performing system wide optimization, we both improved performance and lowered costs as we integrated an entire workstation system into three VLSI chips.

Acknowledgments

We only have space here to thank those who were directly involved with this article: Charlie Kohlhardt, Mark Forsyth, Craig Gleason, Doug Josephson, Joel Lamb, Tom Meyer, Leith Johnson, Pat Knebel, Paul Martin, Tony Barkans, John Wheeler, and Ruby Lee.

References

1. P. Knebel et al., "HP's PA7100LC: A Low-Cost Superscalar PA-RISC Processor," *Compcon Digest of Papers*, Feb. 1993, pp. 441-447.
2. T. Asprey et al., "Performance Features of the PA7100 Microprocessor," *IEEE Micro*, Vol. 13, No. 3, June 1993, pp. 22-35.
3. E. DeLano et al., "A High Speed Superscalar PA-RISC Processor," *Compcon Digest of Papers*, Feb. 1992, pp. 116-121.
4. M. Forsyth et al., "CMOS PA-RISC Processor for a New Family of Workstations," *Compcon Digest of Papers*, Feb. 1991, pp. 202-207.
5. D. Tanksalvala et al., "A 90-MHz CMOS RISC CPU Designed for Sustained Performance," *Digest of Tech. Papers, IEEE Solid-State Circuits Conf.*, Vol. 33, 1990, pp. 52-53.
6. R. Lee, "Precision Architecture," *Computer*, Vol. 22, No. 1, Jan. 1989, pp. 78-91.
7. C. Casey and L. Thayer, "Scalable Graphics Enhancements for PA-RISC Workstations," *Compcon Digest of Papers*, Feb. 1992, pp. 122-128.

Steve Undy is an engineer/scientist for Hewlett-Packard in Fort Collins, Colorado. He has contributed to the design and verification of six PA-RISC processors and was a key designer of the cache system on the Hummingbird processor. He presented the Hummingbird design at Hot Chips V.

Undy received a BS in electrical engineering and a BS in computer engineering from the University of Michigan and an MS in electrical engineering from Purdue University. He is a member of the IEEE Computer Society.

Mick Bass works at the Fort Collins site as a member of technical staff. He has contributed to CPU, memory controller, and I/O controller designs used in PA-RISC workstations. He has also worked on chip and system verification.

Bass received a BS degree in computer engineering from the University of Illinois at Urbana/Champaign. He is a member of the IEEE.

Dave Hollenbeck is a member of the technical staff at Fort Collins. He has been involved with CPU, memory controller, and system I/O chip designs for PA-RISC systems.

Hollenbeck earned an MSEE and BSEE from the University of South Florida.

Wayne Kever is a member of the technical staff at Fort Collins. He has been involved with the design of four PA-RISC CPU chip sets. His interests include high-speed circuit design, cost/performance trade-offs, and low-power design.

Kever received the BS degree from the University of Oklahoma and the MS degree from Stanford University, both in electrical engineering. He is a member of the IEEE and the IEEE Computer Society.

Larry Thayer is an engineer/scientist at Fort Collins where he has been involved with a number of chip designs for workstation graphics systems. He has published articles and papers in *Byte*, *IEEE Spectrum*, and the proceedings of ACM Siggraph and IEEE Compcon, among others.

Thayer received BS and MS degrees in electrical engineering from Ohio State University.

Direct any questions concerning this article to Steve Undy, Hewlett-Packard, 3404 E. Harmony Rd., Fort Collins, CO 80525; sru@fc.hp.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 153 Medium 154 High 155

Reproduced with permission of copyright owner. Further reproduction prohibited.

Petitioners HTC & LG - Exhibit 1002, p. 280