

7686214



# THE UNITED STATES OF AMERICA

**TO ALL TO WHOM THESE PRESENTS SHALL COME:**

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

*July 12, 2018*

**THIS IS TO CERTIFY THAT ANNEXED IS A TRUE COPY FROM THE RECORDS OF THIS OFFICE OF THE FILE WRAPPER AND CONTENTS OF:**

**APPLICATION NUMBER:** *10/869,200*  
**FILING DATE:** *June 16, 2004*  
**PATENT NUMBER:** *7149867*  
**ISSUE DATE:** *December 12, 2006*



Certified by

*Andres Ibarra*

Under Secretary of Commerce  
for Intellectual Property  
and Director of the United States  
Patent and Trademark Office

061604

16834 U.S. PTO

<b>UTILITY PATENT APPLICATION TRANSMITTAL</b> <small>(Only for new nonprovisional applications under 37 CFR 1.53(b))</small>	Attorney Docket No.	SRC028
	First Inventor	Daniel Poznanovic et al.
	Title	SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY BANDWIDTH IN RECONFIGURABLE HARDWARE
	Express Mail Label No.	EV331755319US

16834 U.S. PTO  
10/869200  
061604

APPLICATION ELEMENTS	Commissioner for Patents P.O. Box 1450 Alexandria, VA 22313-1450
----------------------	--

- |   |  |
|---|--|
| 1. <input checked="" type="checkbox"/> Fee Transmittal Form<br><small>(submit an original and a duplicate for fee processing)</small><br>2. <input type="checkbox"/> Applicant claims small entity status.<br><small>See 37 CFR 1.27</small><br>3. <input checked="" type="checkbox"/> Specification [ total pages <u>26</u> ]<br><small>(preferred Arrangement set forth below)</small><br>- Descriptive title of the Invention<br>- Cross References to Related Applications<br>- Statement Regarding Fed sponsored R&D<br>- Reference to sequence listing, a table, or a computer program listing appendix<br>- Background of the Invention<br>- Brief Summary of the Invention<br>- Brief Description of the Drawings<br>- Detailed Description<br>- Claim(s)<br>- Abstract of the Disclosure<br>4. <input checked="" type="checkbox"/> Drawing(s) [ total sheets <u>12</u> ]<br>5. <input checked="" type="checkbox"/> Oath or Declaration [ total pages <u>3</u> ]<br>a. <input checked="" type="checkbox"/> Newly executed (original or copy)<br>b. <input type="checkbox"/> Copy from prior appl. (37 C.F.R. § 1.63(d))<br><small>(for continuation/divisional with Box 18 completed)</small><br>i. <input type="checkbox"/> <b>DELETION OF INVENTOR(S)</b><br><small>Signed statement attached deleting inventor(s) named in prior application, see 37 C.F.R. §§ 1.63(d)(2) and 1.33(b).</small> | 6. <input type="checkbox"/> Application Data Sheet. (See 37 CFR 1.76)<br>7. <input type="checkbox"/> CD-ROM or CD-R in duplicate, large table or Computer Program (Appendix)<br>8. Nucleotide and/or Amino Acid Sequence Submission (if applicable, all necessary)<br>a. <input type="checkbox"/> Computer Readable Form<br>b. <input type="checkbox"/> Specification Sequence Listing on:<br>i. <input type="checkbox"/> CD-ROM or CD-R (2 copies); or<br>ii. <input type="checkbox"/> paper<br>c. <input type="checkbox"/> Statements verifying identity of above copies |
|---|--|

ACCOMPANYING APPLICATION PARTS	
9. <input checked="" type="checkbox"/> Assignment Papers (coversheet/document(s))	
10. <input type="checkbox"/> 37 CFR. 3.73(b) Statement <input checked="" type="checkbox"/> Power of Attorney <small>(when there is an assignee)</small>	
11. <input type="checkbox"/> English Translation Document	
12. <input type="checkbox"/> IDS & Form PTO/SB/08A <input type="checkbox"/> Copies of IDS Citations	
13. <input type="checkbox"/> Preliminary Amendment	
14. <input checked="" type="checkbox"/> Return Receipt Postcard (MPEP 503)	
15. <input type="checkbox"/> Certified Copy of Priority Document(s)	
16. <input type="checkbox"/> Nonpublication Request Under 35 USC 122(b)(2)(B)(i). Applicant must attach form PTO/SB/35	
17. <input checked="" type="checkbox"/> Other: Certificate of Mailing by Express Mail	

18. If a CONTINUING APPLICATION, check appropriate box, and supply the requisite information below and in a preliminary amendment, or in an Application Data Sheet under 37 CFR 1.76:

Continuation  Divisional  Continuation-in-part (CIP) of prior application No.:   /  

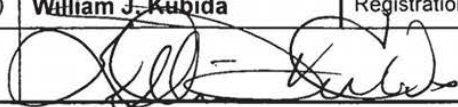
Prior application information: Examiner:                      Group/Art Unit:                     

FOR CONTINUATION OR DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 5b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

19. CORRESPONDENCE ADDRESS

Customer Number **25235** or  Correspondence address below

Name			
Address			
City	State	ZIP	
Country	Telephone	Fax	

Name (Print/Type)	<b>William J. Kubida</b>	Registration No.	<b>29,664</b>
(Signature)		Date	<u>16 Aug 2004</u>

EXPRESS MAIL NO. EV331755319US  
Attorney Docket No. SRC028  
Client/Matter No. 80404.0033.001

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:

Daniel Poznanovic, David E. Caliga,  
and Jeffrey Hammes

Serial No. NEW

Filed: Herewith

For: SYSTEM AND METHOD OF  
ENHANCING EFFICIENCY AND  
UTILIZATION OF MEMORY  
BANDWIDTH IN RECONFIGURABLE  
HARDWARE

CERTIFICATE OF MAILING BY EXPRESS MAIL

Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

Sir:

The undersigned hereby certifies that the following documents:

1. Utility Patent Application Transmittal;
2. Fee Transmittal and \$928 filing fee;
3. Utility Patent Application- 22 pgs. Spec, 3 pgs. Claims, 1 pg. Abstract;
4. Executed Declaration for Utility Patent Application;
5. 12 sheets of drawings;
6. Recordation Form Cover Sheet PTO 1595 with Executed Assignment and Recording Fee of \$40.00;
7. Return postcard; and
8. Certificate of Mailing By Express Mail

relating to the above application, were deposited as "Express Mail", Mailing Label No. EV331755319US, with the United States Postal Service, addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

16 June 2004

Date

16 June 2004

Date

Mailer



William J. Kubida, Reg. No. 29,664  
HOGAN & HARTSON<sup>LLP</sup>  
One Tabor Center  
1200 17th Street, Suite 1500  
Denver, Colorado 80202  
(719) 448-5909 Tel  
(303) 899-7333 Fax

061604

**FEE TRANSMITTAL for FY 2004**  
 Effective 10/01/2003. Patent fees are subject to annual revision

Applicant claims small entity status. See 37 CFR 1.27

**TOTAL AMOUNT OF PAYMENT (\$)** **968.00**

**Complete if Known**

Application Number: -----  
 Filing Date: Herewith  
 First Named Inventor: Daniel Poznanovic et al.  
 Examiner Name: \_\_\_\_\_  
 Group / Art Unit: \_\_\_\_\_  
 Attorney Docket No.: SRC028

**METHOD OF PAYMENT (check all that apply)**

check  credit card  money order  other  none  
 Deposit Account

Deposit Account Number: **50-1123**

Deposit Account Name: **Hogan & Hartson L.L.P.**

The Director is authorized to: (check all that apply)  
 Charge fee(s) indicated below  Credit any overpayments  
 Charge any additional fee(s) or any underpayment of fee(s)  
 Charge fee(s) indicated below, except for the filing fee to the above-identified deposit account

**3. ADDITIONAL FEES**

Large Entity Fee (\$)	Small Entity Fee (\$)	Fee Description	Fee Paid
130	65	Surcharge - late filing fee or oath	
50	25	Surcharge - late provisional filing fee or cover sheet	
130	130	Non-English specification	
2,520	2,520	For filing a request for ex parte reexamination	
920*	920*	Requesting publication of SIR prior to Examiner action	
1,840*	1,840*	Requesting publication of SRI after Examiner action	
110	55	Extension for reply within first month	
420	210	Extension for reply within second month	
950	475	Extension for reply within third month	
1,480	740	Extension for reply within fourth month	
2,010	1,005	Extension for reply within fifth month	
330	165	Notice of Appeal	
330	165	Filing a brief in support of an appeal	
290	145	Request for oral hearing	
1,510	1,510	Petition to institute a public use proceeding	
110	55	Petition to revive - unavoidable	
1,330	665	Petition to revive - unintentional	
1,330	664	Utility issue fee (or reissue)	
480	240	Design issue fee	
640	320	Plant issue fee	
130	130	Petitions to the Commissioner	
50	50	Processing fee under 37 CFR 1.17(q)	
180	180	Submission of Info Disclosure Stmt	
40	40	Recording each patent assignment per property (times number of properties)	40.00
770	385	Filing a submission after final rejection (37 CFR § 1.129(a))	
770	385	For each additional invention to be examined (37 CFR §1.129(b))	
770	385	Request for Continued Examination	
900	900	Request for expedited examination of a design application	
Other fee (specify) .....			
*Reduced by Basic Filing Fee Paid		<b>SUBTOTAL (3)</b>	<b>(\$)</b> <b>40.00</b>

**FEE CALCULATION**

**1. BASIC FILING FEE**

Large Entity Fee (\$)	Small Entity Fee (\$)	Fee Description	Fee Paid
770	385	Utility Filing Fee	770.00
340	170	Design filing fee	
530	265	Plant filing fee	
770	385	Reissue filing fee	
160	80	Provisional filing fee	
<b>SUBTOTAL (1)</b>			<b>(\$)</b> <b>770.00</b>

**2. EXTRA CLAIM FEES FOR UTILITY AND REISSUE**

Total Claims: 24 -20\*\*= 4 X 18 = 72.00  
 Independent Claims: 4 -3\*\*= 1 X 86 = 86.00  
 Multiple Dependent: \_\_\_\_\_ = \_\_\_\_\_

\*\*or number previously paid, if greater; For Reissues, see below

Large Entity Fee (\$)	Small Entity Fee (\$)	Fee Description	Fee Paid
18	9	Claims in excess of 20	
86	43	Independent claims in excess of 3	
290	145	Multiple dependent claim, if not paid	
86	43	**Reissue independent claims over original patent	
18	9	**Reissue claims in excess of 20 and over original patent	
<b>SUBTOTAL (2)</b>			<b>(\$)</b> <b>158.00</b>

**SUBMITTED BY** Complete (if applicable)  
 Name (Print/Type): **William J. Kubida**  
 Signature: *William J. Kubida*  
 Registration No. (Attorney/Agent): **29,664**  
 Telephone: **(719) 448-5900**  
 Date: **16 June 2004**

**SYSTEM AND METHOD OF ENHANCING EFFICIENCY  
AND UTILIZATION OF MEMORY BANDWIDTH IN  
RECONFIGURABLE HARDWARE**

**1. Related Applications.**

**[0001]** The present invention claims the benefit of U.S. Provisional Patent application Serial No. 60/479,339 filed on June 18, 2003, which is incorporated herein by reference in its entirety.

**BACKGROUND OF THE INVENTION**

**1. Field of the Invention.**

**[0002]** The present invention relates, in general, to enhancing the efficiency and utilization of memory bandwidth in reconfigurable hardware. More specifically, the invention relates to implementing explicit memory hierarchies in reconfigurable processors that make efficient use of off-board, on-board, on-chip storage and available algorithm locality. These explicit memory hierarchies avoid many of the tradeoffs and complexities found in the traditional memory hierarchies of microprocessors.

**2. Relevant Background.**

**[0003]** Over the past 30 years, microprocessors have enjoyed annual performance gains averaging about 50% per year. Most of the gains can be attributed to higher processor clock speeds, more memory bandwidth and increasing utilization of instruction level parallelism (ILP) at execution time.

**[0004]** As microprocessors and other dense logic devices (DLDs) consume data at ever-increasing rates it becomes more of a challenge to design memory hierarchies that can keep up. Two measures of the gap between the microprocessor and memory hierarchy are bandwidth efficiency and bandwidth

utilization. Bandwidth efficiency refers to the ability to exploit available locality in a program or algorithm. In the ideal situation, when there is maximum bandwidth efficiency, all available locality is utilized. Bandwidth utilization refers to the amount of memory bandwidth that is utilized during a calculation. Maximum bandwidth utilization occurs when all available memory bandwidth is utilized.

**[0005]** Potential performance gains from using a faster microprocessor can be reduced or even negated by a corresponding drop in bandwidth efficiency and bandwidth utilization. Thus, there has been significant effort spent on the development of memory hierarchies that can maintain high bandwidth efficiency and utilization with faster microprocessors.

**[0006]** One approach to improving bandwidth efficiency and utilization in memory hierarchies has been to develop ever more powerful processor caches. These caches are high-speed memories (typically SRAM) in close proximity to the microprocessor that try to keep copies of instructions and data the microprocessor may soon need. The microprocessor can store and retrieve data from the cache at a much higher rate than from a slower, more distant main memory.

**[0007]** In designing cache memories, there are a number of considerations to take into account. One consideration is the width of the cache line. Caches are arranged in lines to help hide memory latency and exploit spatial locality. When a load suffers a cache miss, a new cache line is loaded from main memory into the cache. The assumption is that a program being executed by the microprocessor has a high degree of spatial locality, making it likely that other memory locations in the cache line will also be required.

**[0008]** For programs with a high degree of spatial locality (e.g., stride-one access), wide cache lines are more efficient since they reduce the number of times a processor has to suffer the latency of a memory access. However, for programs with lower levels of spatial locality, or random access, narrow lines

are best as they reduce the wasted bandwidth from the unused neighbors in the cache line. Caches designed with wide cache lines perform well with programs that have a high degree of spatial locality, but generally have poor gather/scatter performance. Likewise, caches with short cache lines have good gather/scatter performance, but loose efficiency executing programs with high spatial locality because of the additional runs to the main memory.

**[0009]** Another consideration in cache design is cache associativity, which refers to the mapping between locations in main memory and cache sectors. At one extreme of cache associativity is a direct-mapped cache, while at another extreme is a fully associative cache. In a direct mapped-cache, a specific memory location can be mapped to only a single cache line. Direct-mapped caches have the advantage of being fast and easy to construct in logic. The disadvantage is that they suffer the maximum number of cache conflicts. At the other extreme, a fully associative cache allows a specific location in memory to be mapped to any cache line. Fully associative caches tend to be slower and more complex due to the large amount of comparison logic they need, but suffer no cache conflict misses. Oftentimes, caches fall between the extremes of direct-mapped and fully associative caches. A design point between the extremes is a k-set associative cache, where each memory location can map to k cache sectors. These caches generally have less overhead than fully associative caches, and reduce cache conflicts by increasing the value of k.

**[0010]** Another consideration in cache design is how cache lines are replaced due to a capacity or conflict miss. In a direct-mapped cache, there is only one possible cache line that can be replaced due to a miss. However, in caches with higher levels of associativity, cache lines can be replaced in more than one way. The way the cache lines are replaced is referred to as the replacement policy.

**[0011]** Options for the replacement policy include least recently used (LRU), random replacement, and first in—first out (FIFO). LRU is used in the majority of circumstances where the temporal locality set is smaller than the cache size, but it is normally more expensive to build in hardware than a random replacement cache. An LRU policy can also quickly degrade depending on the working set size. For example, consider an iterative application with a matrix size of  $N$  bytes running through a LRU cache of size  $M$  bytes. If  $N$  is less than  $M$ , then the policy has the desired behavior of 100% cache hits, however, if  $N$  is only slightly larger than  $M$ , the LRU policy results in 0% cache hits as lines are removed just as they are needed.

**[0012]** Another consideration is deciding on a write policy for the cache. Write-through caches send data through the cache hierarchy to main memory. This policy reduces cache coherency issues for multiple processor systems and is best suited for data that will not be re-read by the processor in the immediate future. In contrast, write-back caches place a copy of the data in the cache, but does not immediately update main memory. This type of caching works best when a data just written to the cache is quickly requested again by the processor.

**[0013]** In addition to write-through and write-back caches, another kind of write policy is implemented in a write-allocate cache where a cache line is allocated on a write that misses in cache. Write-allocate caches improve performance when the microprocessor exhibits a lot of write followed by read behavior. However, when writes are not subsequently read, a write-allocate cache has a number of disadvantages: When a cache line is allocated, it is necessary to read the remaining values from main memory to complete the cache line. This adds unnecessary memory read traffic during store operations. Also, when the data is not read again, potentially useful data in the cache is displaced by the unused data.



**[0014]** Another consideration is made between the size and the speed of the cache: small caches are typically much faster than larger caches, but store less data and fewer instructions. Less data means a greater chance the cache will not have data the microprocessor is requesting (i.e., a cache miss) which can slow everything down while the data is being retrieved from the main memory.

**[0015]** Newer cache designs reduce the frequency of cache misses by trying to predict in advance the data that the microprocessor will request. An example of this type of cache is one that supports speculative execution and branch prediction. Speculative execution allows instructions that likely will be executed to start early based on branch prediction. Results are stored in a cache called a reorder buffer and retired if the branch was correctly predicted. Of course, when mis-predictions occur instruction and data bandwidth are wasted.

**[0016]** There are additional considerations and tradeoffs in cache design, but it should be apparent from the considerations described hereinbefore that it is very difficult to design a single cache structure that is optimized for many different programs. This makes cache design particularly challenging for a multipurpose microprocessor that executes a wide variety of programs. Cache designers try to derive the program behavior of "average" program constructed from several actual programs that run on the microprocessor. The cache is optimized for the average program, but no actual program behaves exactly like the average program. As a result, the designed cache ends up being sub-optimal for nearly every program actually executed by the microprocessor. Thus, there is a need for memory hierarchies that have data storage and retrieval characteristics that are optimized for actual programs executed by a processor.

**[0017]** Designers trying to develop ever more efficient caches optimized for a variety of actual programs also face another problem: as caches add additional features, the overhead needed to implement the added features also grows.

Caches today have so much overhead that microprocessor performance may be reaching a point of diminishing returns as the overhead starts to cut into performance. In the Intel Pentium III processor for example, more than half of the 10 million transistors are dedicated to instruction cache, branch prediction, out-of-order execution and superscalar logic. The situation has prompted predictions that as microprocessors grow to a billion transistors per chip, performance increases will drop to about 20% per year. Such a prediction, if borne out, could have a significant impact on technology growth and the computer business.

**[0018]** Thus, there is a growing need to develop improved memory hierarchies that limit the overhead of a memory hierarchy without also reducing bandwidth efficiency and utilization.

#### **SUMMARY OF THE INVENTION**

**[0019]** Accordingly, an embodiment of the invention includes a reconfigurable processor that includes a computational unit and a data access unit coupled to the computational unit, where the data access unit retrieves data from an on-processor memory and supplies the data to the computational unit, and where the computational unit and the data access unit are configured by a program.

**[0020]** The present invention also involves a reconfigurable processor that includes a first memory of a first type and a data prefetch unit coupled to the memory, where the data prefetch unit retrieves data from a second memory of a second type different from the first type, and the first and second memory types and the data prefetch unit are configured by a program.

**[0021]** Another embodiment of the invention includes a reconfigurable hardware system that includes a common memory, also referred to as external memory, and one or more reconfigurable processors coupled to the common memory, where at least one of the reconfigurable processors includes a data prefetch unit to read and write data between the unit and the common memory,

and where the data prefetch unit is configured by a program executed on the system.

**[0022]** Another embodiment of the invention includes a method of transferring data that includes transferring data between a memory and a data prefetch unit in a reconfigurable processor, transferring data between the prefetch unit and a data access unit, and transferring the data between a computational unit and the data access unit, where the computational unit, data access unit and the data prefetch unit are configured by a program.

**[0023]** Additional embodiments of the invention are set forth in part in the description that follows, and in part will become apparent to those skilled in the art upon examination of the following specification, or may be learned by the practice of the invention. The advantages of the invention may be realized and attained by means of the instrumentalities, combinations, compositions, and methods particularly pointed out in the appended claims.

#### **BRIEF DESCRIPTION OF THE DRAWINGS**

**[0024]** Figure 1 shows a reconfigurable processor in which the present invention may be implemented;

**[0025]** Figure 2 shows computational logic as might be loaded into a reconfigurable processor;

**[0026]** Figure 3 shows a reconfigurable processor as in Figure 1, but with the addition of data access units;

**[0027]** Figure 4 shows a reconfigurable processor as in Figure 3, but with the addition of data prefetch units;

**[0028]** Figure 5 shows reconfigurable processor with the inclusion of external memory;

**[0029]** Figure 6 shows reconfigurable processors with external memory and with an intelligent memory controller;

**[0030]** Figure 7 shows a reconfigurable processor having a combination of data prefetch units and data access units feeding computational logic;

**[0031]** Figure 8 shows the bandwidth efficiency and utilization gains obtained when utilizing a data prefetch unit and an intelligent memory controller to perform strided memory references;

**[0032]** Figure 9A and Figure 9B show the bandwidth efficiency and utilization gains obtained when utilizing a data prefetch unit and an intelligent memory controller to perform subset memory references in X-Y plane;

**[0033]** Figure 10A and Figure 10B show the bandwidth efficiency and utilization gains obtained when utilizing a data prefetch unit and an intelligent memory controller to perform subset memory references in X-Z plane;

**[0034]** Figure 11A and Figure 11B show the bandwidth efficiency and utilization gains obtained when utilizing a data prefetch unit and an intelligent memory controller to perform subset memory references in Y-Z plane;

**[0035]** Figure 12A and Figure 12B show the bandwidth efficiency and utilization gains obtained when utilizing a data prefetch unit and an intelligent memory controller to perform subset memory references in a mini-cube;

**[0036]** Figure 13 shows the bandwidth efficiency and utilization gains obtained when utilizing a data prefetch unit and an intelligent memory controller to perform indirect memory references;

**[0037]** Figure 14 shows the bandwidth efficiency and utilization gains obtained when utilizing a data prefetch unit and an intelligent memory controller to perform strided memory reference together with computation.

## **DETAILED DESCRIPTION**

### **1. Definitions:**

**[0038] Direct execution logic (DEL)** - is an assemblage of dynamically reconfigurable functional elements that enables a program to establish an optimized interconnection among selected functional units in order to implement a desired computational, data prefetch and/or data access functionality for maximizing the parallelism inherent in the particular code.

**[0039] Reconfigurable Processor** – is a computing device that contains reconfigurable components such as FPGAs and can, through reconfiguration, instantiate an algorithm as hardware.

**[0040] Reconfigurable Logic** – is composed of an interconnection of functional units, control, and storage that implements an algorithm and can be loaded into a Reconfigurable Processor.

**[0041] Functional Unit** – is a set of logic that performs a specific operation. The operation may for example be arithmetic, logical, control, or data movement. Functional units are used as building blocks of reconfigurable logic.

**[0042] Macro** – is another name for a functional unit.

**[0043] Memory Hierarchy** – is a collection of memories

**[0044] Data prefetch Unit** – is a functional unit that moves data between members of a memory hierarchy. The movement may be as simple as a copy, or as complex as an indirect indexed strided copy into a unit stride memory.

**[0045] Data access Unit** – is a functional unit that accesses a component of a memory hierarchy, and delivers data directly to computational logic.

**[0046] Intelligent Memory Control Unit** – is a control unit that has the ability to select data from its storage according to a variety of algorithms that can be selected by a data requestor, such as a data prefetch unit.

**[0047] Bandwidth Efficiency** – is defined as the percentage of contributory data transferred between two points. Contributory data is data that actually participates in the recipients processing.

**[0048] Bandwidth Utilization** – is defined as the percentage of maximum bandwidth between two points that is actually used to pass contributory data.

## 2. Description

**[0049]** A reconfigurable processor (RP) 100 implements direct executable logic (DEL) to perform computation, as well a memory hierarchy for maintaining input data and computational results. DEL is an assemblage of dynamically reconfigurable functional elements that enables a program to establish an optimized interconnection among selected functional units in order to implement a desired computational, data prefetch and/or data access functionality for maximizing the parallelism inherent in the particular code. The term DEL may also be used to refer to the set of constructs such as code, data, configuration variables, and the like that can be loaded into RP 100 to cause RP 100 to implement a particular assemblage of functional elements.

**[0050]** Figure 1 presents an RP 100, which may be implemented using field programmable gate arrays (FPGAs) or other reconfigurable logic devices, that can be configured and reconfigured to contain functional units and interconnecting circuits, and a memory hierarchy comprising on-board memory banks 104, on-chip block RAM 106, registers wires, and a connection 108 to external memory. On-chip reconfigurable components 102 create memory structures such as registers, FIFOs, wires and arrays using block RAM. Dual-ported memory 106 is shared between on-chip reconfigurable components 102. The reconfigurable processor 100 also implements user-defined computational

logic (e.g., such as DEL 200 shown in Figure 2) constructed by programming an FPGA to implement a particular interconnection of computational functional units. In a particular implementation, a number of RPs 100 are implemented within a memory subsystem of a conventional computer, such as on devices that are physically installed in dual inline memory module (DIMM) sockets of a computer. In this manner the RPs 100 can be accessed by memory operations and so coexist well with a more conventional hardware platform. It should be noted that, although the exemplary implementation of the present invention illustrated includes six banks of dual ported memory 104 and two reconfigurable components 102, any number of memory banks and/or reconfigurable components may be used depending upon the particular implementation or application.

**[0051]** Any computer program, including complex graphics processing programs, word processing programs, database programs and the like, is a collection of algorithms that interact to implement desired functionality. In the common case in which static computing hardware resources are used (e.g., a conventional microprocessor), the computer program is compiled into a set of executable code (i.e., object code) units that are linked together to implement the computer program on the particular hardware resources. The executable code is generated specifically for a particular hardware platform. In this manner, the computer program is adapted to conform to the limitations of the static hardware platform. However, the compilation process makes many compromises based on the limitations of the static hardware platform.

**[0052]** Alternatively, an algorithm can be defined in a high level language then compiled into DEL. DEL can be produced via a compiler from high level programming languages such as C or FORTRAN or may be designed using a hardware definition language such as Verilog, VHDL or a schematic capture tool. Computation is performed by reconfiguring a reconfigurable processor with the DEL and flowing data through the computation. In this manner, the

hardware resources are essentially adapted to conform to the program rather than the program being adapted to conform to the hardware resources.

**[0053]** For purposes of this description a single reconfigurable processor will be presented first. A sample of computational logic 201 is shown in Figure 2. This simple assemblage of functional units performs computation of two results ("A+B" and "A+B-(B\*C)") from three input variables or operands "A", "B" and "C". In practice, computational units 201 can be implemented to perform very simple or arbitrarily complex computations. The input variables (operands) and output or result variables may be of any size necessary for a particular application. Theoretically, any number of operands and result variables may be used/generated by a particular DEL. Great complexity of computation can be supported by adding additional reconfigurable chips and processors.

**[0054]** For greatest performance the DEL 200 is constructed as parallel pipelined logic blocks composed of computational functional units capable of taking data and producing results with each clock pulse. The highest possible performance that can be achieved is computation of a set of results with each clock pulse. To achieve this, data should be available at the same rate the computation can consume the data. The rate at which data can be supplied to DEL 200 is determined, at least in significant part, by the memory bandwidth utilization and efficiency. Maximal computational performance can be achieved with parallel and pipelined DEL together with maximizing the memory bandwidth utilization and efficiency. Unlike conventional static hardware platforms, however, the memory hierarchy provided in a RP 100 is reconfigurable. In accordance with the present invention, through the use of data access units and associated memory hierarchy components, computational demands and memory bandwidth can be matched.

**[0055]** High memory bandwidth efficiency is achieved when only data required for computation is moved within the memory hierarchy. Figure 3 shows a simple logic block 300 comprising computational functional units 301, control



(not shown), and data access functional units 303. The data access unit 303 presents data directly to the computational logic 301. In this manner, data is moved from a memory device 305 to the computational logic and from the computational logic back into a memory device 305 or block RAM memory 307 within an RP 100.

**[0056]** Figure 4 illustrates the logic block 300 with an addition of a data prefetch unit 401. The data prefetch unit 401 moves data from one member of the memory hierarchy 305 to another 308. Data prefetch unit 401 operates independently of other functional units 301, 302 and 303 and can therefore operate prior to, in parallel with, or after computational logic. This independence of operation permits hiding the latency associated with obtaining data for use in computation. The data prefetch unit deposits data into the memory hierarchy within RP 100, where computational logic 301, 302 and 303 can access it through data access units. In the example of Figure 4, prefetch unit 401 is configured to deposit data into block RAM memory 308. Hence, the prefetch units 401 may be operated independently of logic block 300 that uses prefetched data.

**[0057]** An important feature of the present invention is that many types of data prefetch units can be defined so that the prefetch hardware can be configured to conform to the needs of the algorithms currently implemented by the computational logic. The specific characteristics of the prefetch can be matched with the needs of the computational logic and the format and location of data in the memory hierarchy. For example, Figure 9A and Figure 9B show an external memory that is organized in a 128 byte (16 word) block structure. This organization is optimized for stride 1 access of cache based computers. A stride 128 access can result in a very inefficient use of bandwidth from the memory, since an extra 120 bytes of data is moved for every 8 bytes of requested data yielding a 6.25% bandwidth efficiency.

**[0058]** Figure 5 shows an example of data prefetch in which there are no bandwidth gains since all data fetched from external memory blocks is also transferred and used in computational units 301 through memory bank access units 303. However, bandwidth utilization is increased due to the ability of the data prefetch units 501 to initiate a data transfer in advance of the requirement for data by computational logic.

**[0059]** In accordance with an embodiment of the present invention, data prefetch units 601 are configured to communicate with an intelligent memory controller 603 in Figure 6 and can extract only the desired 8 bytes of data, discard the remainder of the memory block, and transmit to the data prefetch unit only the requested portion of the stride 128 data. The prefetch units 601 then delivers that data to the appropriate memory components within the memory hierarchy of the logic block 300.

**[0060]** Figure 6 shows the prefetch units 601 delivering data to the RP's onboard memory banks 305. An onboard memory bank data access unit 303 then delivers the data to computational logic 301 when required. The data prefetch units 501 couple with an intelligent memory controller 601 in the implementation of Figure 6 that supports a strided reference pattern, which yields a 100% bandwidth efficiency in contrast to the 6.25% efficiency. Although illustrated as a single block of external memory, multiple numbers of external memories may be employed as well.

**[0061]** In Figure 7, the combination of data prefetch units 701 and data access units 703 feeding computational logic 301 such that bandwidth efficiency and utilization are maximized is shown in Figure 7. In this example strided data prefetch units 701 fetch only the required data words from external memory. Figure 8 demonstrates the efficiency gains enabled by this combination. Prefetch units 701 deliver the data into stream memory components 705 that is accessed by stream data access units 703. The stream data access units 703 fetch data from the stream based on valid data bits that are provided to the

stream by the data prefetch units 701 as data is presented to the stream. Use of the stream data access unit allows computational logic to be activated upon initiation of the data prefetch operation. This, in turn, allows computation to start with the arrival of the first data item, signaled by valid data bits. Computational logic 301 does not have to await arrival of a complete buffer of data in order to proceed. This elimination of latency increases the bandwidth utilization, by allowing data transfer to continue uninterrupted and in parallel with computation.

**[0062]** Figure 8 illustrates the efficiency gains enabled by the configuration of Figure 7. Figure 8 shows a plurality of memory blocks 800 in which only one memory element 801 exists in each memory block 800. The configuration of Figure 7 allows the desired portions 801 of each memory block 800 to be compacted into a transfer buffer 805. The desired data elements 801 are compacted in order. Since only the contents of the transfer buffer 805 need be transferred to the computational logic, a significant increase in transfer efficiency can be realized.

**[0063]** Figures 9A/9B, 10A/10B, 11A/11B and 12A/12B show bandwidth efficiency gains that are achieved in various situations when a subset of stored data is required for computation. Applications store data in a specific order in memory. However it is often the case that the actual reference pattern required during computation is different from the ordering of data in memory. Figures 9A/9B, 10A/10B, 11A/11B and 12A/12B show an example of a X,Y,Z coordinate oriented data which is stored such that striding though the X axis is the most efficient for retrieving blocked data.

**[0064]** Coupling data prefetch units in the RP 100 with an intelligent memory controller 601 in the external memory yields a significant improvement in bandwidth efficiency and utilization. Four examples are presented in the Figures 9A/9B, 10A/10B, 11A/11B and 12A/12B in which the shaded memory locations indicate desired data. The Figures illustrate an intelligent memory

controller's response to each of four different data prefetch unit's requests for data. Again, an important feature of the present invention is the ability to implement various kinds or styles of prefetch units to meet the needs of a particular algorithm being implemented by computational elements 301. For ease of illustration, each example shows the same set of computational logic, however, in most cases the function being implemented by components 301 would change and therefore alter the decision as to which prefetch strategy is most appropriate. In accordance with the present invention, the prefetch units are implemented in a manner that is optimized for the implemented computational logic.

**[0065]** Figure 9A/9B shows response to a request from an XY-slice data prefetch unit. Figure 10A/10B shows response to a XZ-slice data prefetch unit request. Figure 11A/11B shows response to a YZ-slice data prefetch unit request. Figure 12A/12D shows the response to a SubCube data prefetch unit request. In each of these examples the data prefetch units are configured to pass information to the intelligent memory controller 601 to identify the type of request that is being made, as well as a data address and parameters, in this case, defining the slice size or sub-cube size.

**[0066]** One of the largest bandwidth efficiency and utilization gains can be seen in the case of a Gather data prefetch unit working in cooperation with an intelligent memory controller 601. Figure 13 illustrates the activity in the external memory controller 601. In this example an index array 1301 and a data array 1303 reside in memory. A gather data prefetch unit in an RP 100 requests a gather by specifying the access type as "gather", and providing a pointer to index array 1301, and another pointer to the data array 1303. The memory controller uses the index array 1301 to select desired data elements, indicated by shading, and then delivers an in order stream of data to the prefetch unit. Gains are made by delivering only requested data from transfer buffer 1305 (not the remainder of a data block as in cache line oriented systems) by eliminating the need to transfer an index array either to the

processor or to the memory controller, and by eliminating the start/stop time required when the data is not streamed to the requestor.

**[0067]** A further bandwidth efficiency and utilization gain is made when coupling a data prefetch unit with memory controller capable of computation. Figure 14 illustrates activity in a cooperating memory controller having a computational component 1407 in response to a data prefetch unit. Here the prefetch unit requests a "strided compute", providing parameters for an operator, and addresses, and strides for data to be operated upon. In Figure 14, the data to be operated on comprises "X" data 1401 and "Y" data 1403. The data 1401 and 1403 are processed by computational component 1407 to generate a resultant value that is a specified function of X and Y as indicated by  $F(X,Y)$  in Figure 14. The resultant values are then passed to the requesting prefetch unit via transfer buffer 1405. In this case only computed results are passed and no operand data need to be transferred. Accordingly, where the desired data, indicated by shading in Figure 14, resides across multiple blocks, efficiency is achieved not only by avoiding transfer of the undesired data surrounding the desired data, but also because only the result is transferred, not the original data 1401/1403.

### **EXAMPLES**

**[0068]** Some programming examples utilizing the memory hierarchy of the present invention will now be illustrated. The first example illustrates how a computational intensive matrix multiplication problem may be handled by the explicitly parallel and addressable storage of the present invention.

#### **1. Example 1: Explicit Parallel and Addressable Storage**

**[0069]** Consider the matrix multiplication  $C = A \times B$ , where:

A is a matrix of size M rows by 64 columns;

B is a matrix of size 64 rows by N columns; and

C is a matrix of size M rows by N columns.

The size and shape of this problem typically arises in the context of LU decomposition in linear algebra libraries (e.g., LAPACK). The operation count for this problem would be  $2 * M * N * 64$ , and the total data necessary to transport would be  $(M * 64 + N * 64 + M * N)$ , making the problem quite computationally intensive.

**[0070]** The dot-product formulation of the matrix multiplication may be represented as the following a triple-nested loop:

```
for (i = 0; i < m; i++) {  
    for (j = 0; j < n; j++) {  
        sum = 0;  
        for (k = 0; k < 64; k++) {  
            sum += A[k*m+i] * B[j*64+k];  
        }  
        C1[i+j*mm] = sum;  
    }  
}
```

**[0071]** On a conventional microprocessor with static execution resources, these loops would be arranged to give stride-one data access where possible and also block or tile these uses to facilitate data cache hits on the B and A matrices, which are read many times. With the configurable memory hierarchy of the present invention, matrix B may be stored in on-board BRAM memory 307 and rows of matrix A in registers.

**[0072]** The rows of matrix B may be stored in independently, locally declared BRAM arrays (B0, B1, . . . B63). The rows are stored as independent memory structures, and may be accessed in parallel. Rows of matrix A may be stored in 64 registers described with scalar variables. With these explicit data structures, the following pseudo code can describe the matrix multiplication:

```
Load B into BRAM;

for (i = 0; i < m; i++) {

Load ith Row of A into registers A00 to A63;

For (j = 0; j < n; j++) {

    C[i+j+m] +=

A00 * b0[j] +

A01 * b1[j] +

A02 * b2[j] + //inner loop produces

A03 * b3[j] + //128 results per

A04 * b4[j] + //clock cycle. 64 rows

A05 * b5[j] + //of B are read in

A06 * b6[j] + //parallel

. . .

A63 * b63[j];
```

**[0073]** The code is designed to minimize the amount of data motion. The A and B matrices are read once and the C matrix is written just once at it is produced. When computational resources permit, the i loop could also be

unrolled to process multiple rows of matrix A against matrix B in the inner loop. Processing two rows of A, for example, would produce 256 computational results per clock cycle.

## 2. Example 2: Irregular Memory Access

**[0074]** Benchmarks have been developed for measuring the ability of a computer system to perform indirect updates. An indirect update, written in the C programming language, looks like:

```
for (I = 0; I < N; I++) {  
  
    A[Index[I]] = A[Index[I]] + B[I];  
  
}
```

Typically, A is a large array, and Index has an unpredictable distribution. The benchmark generally forces memory references to miss in cache, and for entire cache lines to be brought in for single-word updates. The problem gets worse as memories get further away from processors and cache lines become wider.

**[0075]** In this example, the arrays have 64-bit data. To complete one iteration of this loop, 24 bytes of information is required from memory and 8 bytes are written back for a total of 32 bytes of memory motion per iteration. On an implicit architecture with cache-lines of width W bytes, each iteration results in the following memory bus traffic:

1. Index[I]: 8 bytes per iteration due to stride-1 nature;
2. B[I]: 8 bytes per iteration due to stride-1 nature; and
3. A[Index[I]]: W bytes read and written per iteration.

The total amount of bus traffic is  $2*W + 16$  bytes per iteration. On an average microprocessor today,  $W = 128$  so an iteration of this loop results in 272 bytes



of memory traffic when only 32 bytes is algorithmically required, making only 12% of the data moved as being useful for the problem.

**[0076]** In addition, because microprocessors rely on wide cache lines and hardware pre-fetching strategies to amortize the long latency to main memory, only a small number of outstanding cache-line misses are typically tolerated. Because of the irregular nature of this example, hardware pre-fetching provides little benefit, making it difficult to keep the memory bus saturated, even with the large amount of wasted memory traffic. Bus utilization on the microprocessor processing only consumes about 700 MB/sec of the 3.2 GB/sec available, or 22%. Combining the poor bus utilization with the relatively small amount of data that is useful results in the microprocessor executing at about 2.5% of peak.

**[0077]** The memory hierarchy of the present invention does not require that memory traffic be organized in a cache-line structure, permitting loop iteration to be accomplished with the minimum number of bytes (in this case 32 bytes of memory traffic). In addition, data pre-fetch functional units may be fully pipelined, allowing full use of available memory bus bandwidth. Data storing may be handled in a similar pipelined fashion. An example of the pseudo code that performs the random update in the memory hierarchy looks like:

```
for (i=0; i < N-Gather_size; i=i+Gather_size) {
    gather ( A, Index, i, A_local, Gather_size)
    for (j=0; j < Gather_size; j++) {
        A_local[j] = A_local[j] + B[j];
    }
    scatter (A_local, Index, &A[i], Gather_size);
}
```

**[0078]** This loop will pipeline safely as described by the pseudo code provided that the index vector has no repeated values within each Gather\_size segment. If repeats are present, then logic within the gather unit can preprocess the Index vector and B vector into safe sub-lists that can be safely pipelined with little or no overhead.

### **Conclusion**

**[0079]** It should be apparent that the scaleable, programmable memory mechanisms enabled by the present invention are available to exploit available algorithm locality and thereby achieve up to 100% bandwidth efficiency. In addition, the scaleable computational resources can be leveraged to attain 100% bandwidth utilization. As a result, the present invention provides a programmable computational system that delivers the maximum possible performance for any memory bus speed. This combination of efficiency and utilization yields orders of magnitude performance benefit compared with implicit models when using an equivalent memory bus.

**[0080]** Although the invention has been described and illustrated with a certain degree of particularity, it is understood that the present disclosure has been made only by way of example, and that numerous changes in the combination and arrangement of parts can be resorted to by those skilled in the art without departing from the spirit and scope of the invention, as hereinafter claimed.

**WE CLAIM:**

1. A reconfigurable processor comprising:  
a first memory having a first characteristic memory type; and  
a data prefetch unit coupled to the memory, wherein the data prefetch  
5 unit retrieves data from a second memory of second characteristic memory  
type and wherein the memory types and data prefetch unit are configured by  
a program.
2. The reconfigurable processor of claim 1, wherein the processor  
does not have a cache to store data from the memory.
- 10 3. The reconfigurable processor of claim 1, wherein the data retrieved  
from the memory is not a cache line-sized unit of contiguous data.
4. The reconfigurable processor of claim 1, wherein the data prefetch  
unit is coupled to a memory controller that controls the transfer of the data  
between the memory and the data prefetch unit.
- 15 5. The reconfigurable processor of claim 1, wherein the data prefetch  
unit receives processed data from on-processor memory and writes the  
processed data to an external off-processor memory memory.
6. The reconfigurable processor of claim 1, wherein the data prefetch  
unit comprises at least one register from the reconfigurable processor.
- 20 7. The reconfigurable processor of claim 1, wherein the data prefetch  
unit is disassembled when another program is executed on the reconfigurable  
processor.
8. The reconfigurable processor of claim 1 wherein said prefetch  
unit is operative to retrieve data from a processor memory.
- 25 9. The reconfigurable processor of claim 8 wherein said processor  
memory is a microprocessor memory.

10. The reconfigurable processor of claim 8, wherein said processor memory is a reconfigurable processor memory.

11. A reconfigurable hardware system, comprising:  
a common memory; and

5 one or more reconfigurable processors coupled to the common memory, wherein at least one of the reconfigurable processors includes a data prefetch unit to read and write data between the unit and the common memory, and wherein the data prefetch unit is configured by a program executed on the system.

10 12. The reconfigurable hardware system of claim 11, comprising a memory controller coupled to the common memory and the data prefetch unit.

13. The reconfigurable hardware system of claim 11, wherein the reconfigurable processor is not coupled to a cache.

15 14. The reconfigurable hardware system of claim 11, wherein the data written and read between the data prefetch unit and the common memory is not a cache line-sized unit of contiguous data.

15 15. The reconfigurable hardware system of claim 11, wherein the at least of the reconfigurable processors also includes a computational unit  
20 coupled to the data access unit.

16. The reconfigurable hardware system of claim 15, wherein the computational unit is supplied the data by the data access unit.

17. A method of transferring data comprising:

25 transferring data between a memory and a data prefetch unit in a reconfigurable processor; and

transferring the data between a computational unit and the data access unit, wherein the computational unit and the data access unit, and the data prefetch unit are configured by a program.

18. The method of claim 17, wherein the data is written to the memory, said method comprising:

transferring the data from the computational unit to the data access unit; and

5 writing the data to the memory from the data prefetch unit.

19. The method of claim 17, wherein the data is read from the memory, said method comprising:

transferring the data from the memory to the data prefetch unit; and

10 reading the data directly from the data prefetch unit to the computational unit through a data access unit.

20. The method of claim 19, wherein all the data transferred from the memory to the data prefetch unit is processed by the computational unit.

21. The method of claim 19, wherein the data is selected by the data prefetch unit based on an explicit request from the computational unit.

15 22. The method of claim 17, wherein the data transferred between the memory and the data prefetch unit is not a complete cache line.

23. The method of claim 17, wherein a memory controller coupled to the memory and the data prefetch unit, controls the transfer of the data between the memory and the data prefetch unit.

20 24. A reconfigurable processor comprising:  
a computational unit; and

a data access unit coupled to the computational unit, wherein the data access unit retrieves data from memory and supplies the data to the computational unit, and wherein the computational unit and the data access  
25 unit are configured by a program.

## **ABSTRACT OF THE DISCLOSURE**

**[0081]** A reconfigurable processor that includes a computational unit and a data prefetch unit coupled to the computational unit, where the data prefetch unit retrieves data from a memory and supplies the data to the computational unit through memory and a data access unit, and where the data prefetch unit, memory, and data access unit is configured by a program. Also, a reconfigurable hardware system that includes a common memory; and one or more reconfigurable processors coupled to the common memory, where at least one of the reconfigurable processors includes a data prefetch unit to read and write data between the unit and the common memory, and where the data prefetch unit is configured by a program executed on the system. In addition, a method of transferring data that includes transferring data between a memory and a data prefetch unit in a reconfigurable processor; and transferring the data between a computational unit and the data prefetch unit.

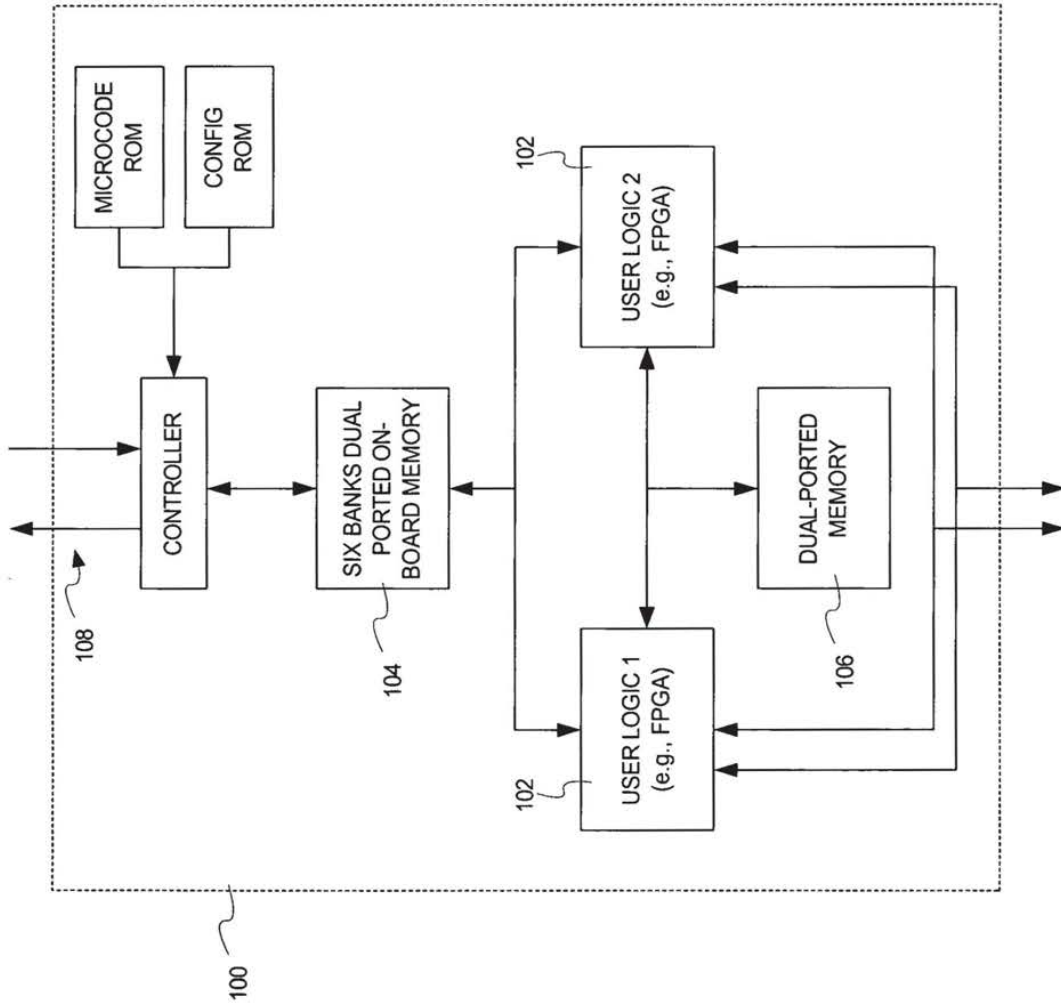


FIG. 1

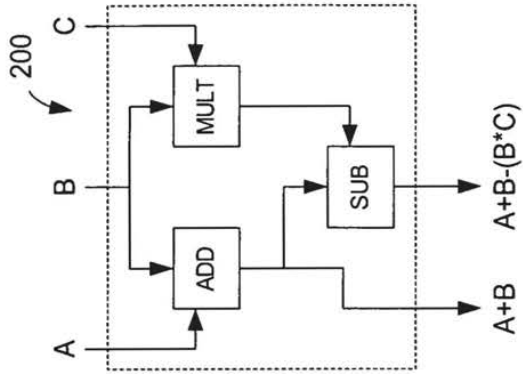


FIG. 2



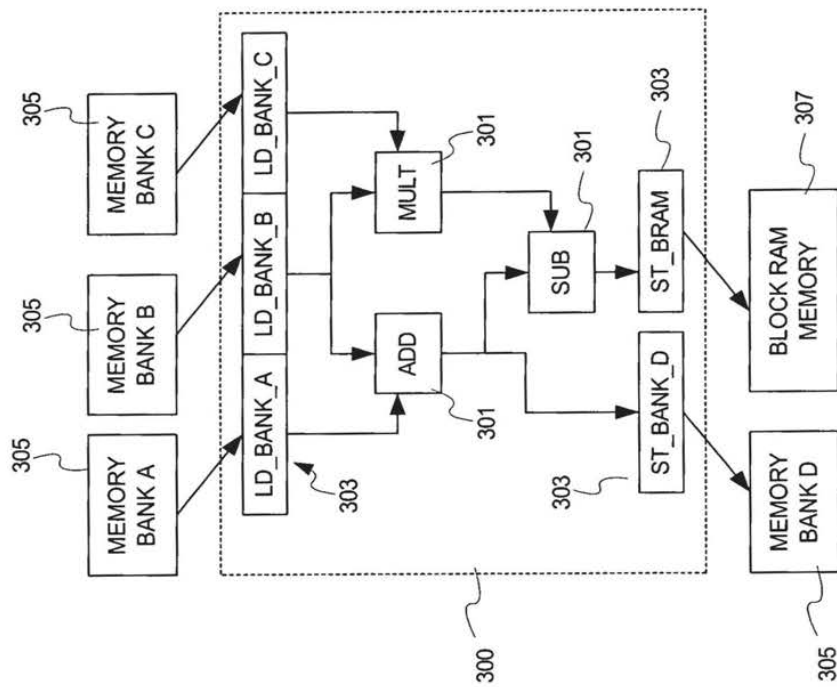
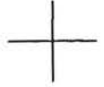


FIG. 3





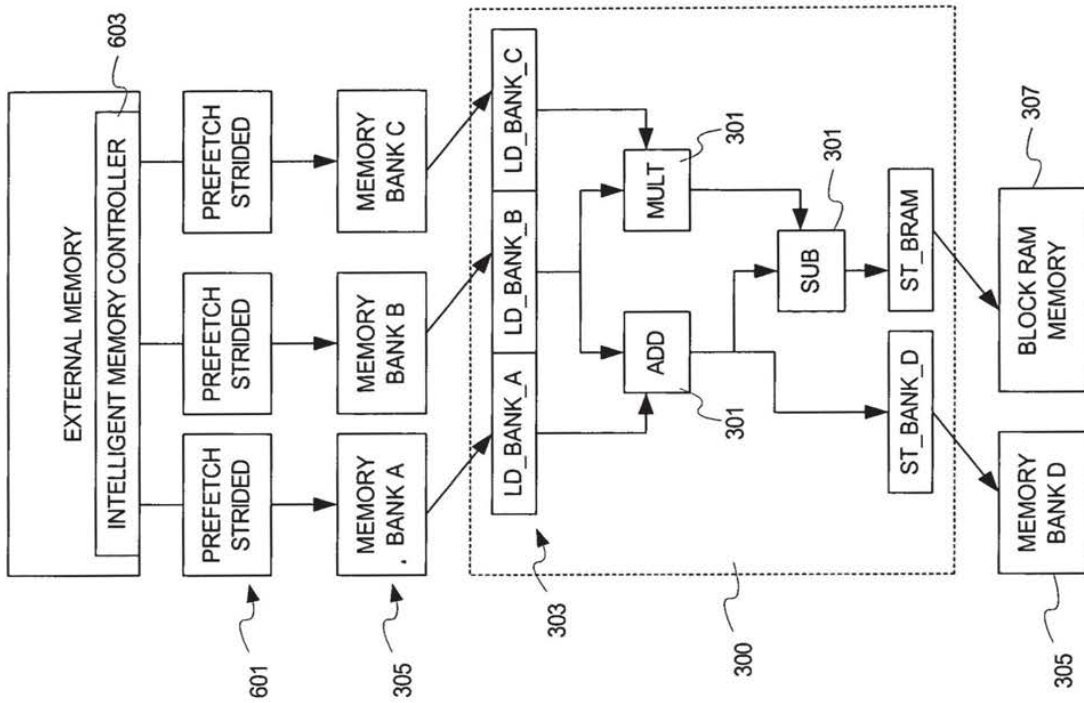


FIG. 6

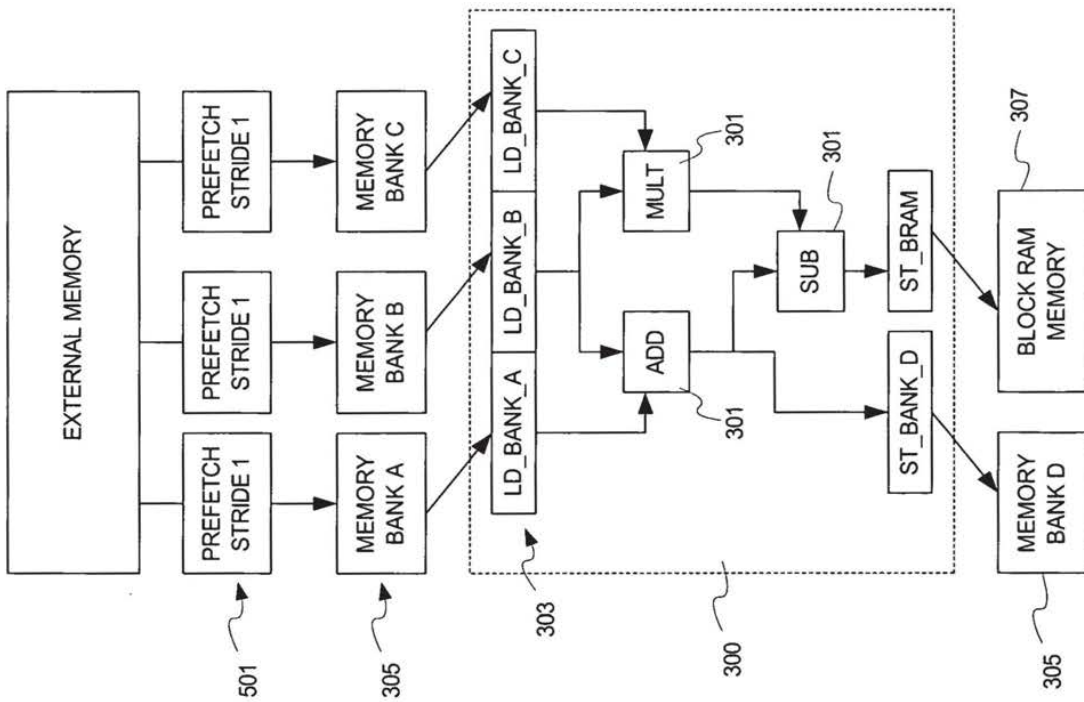


FIG. 5

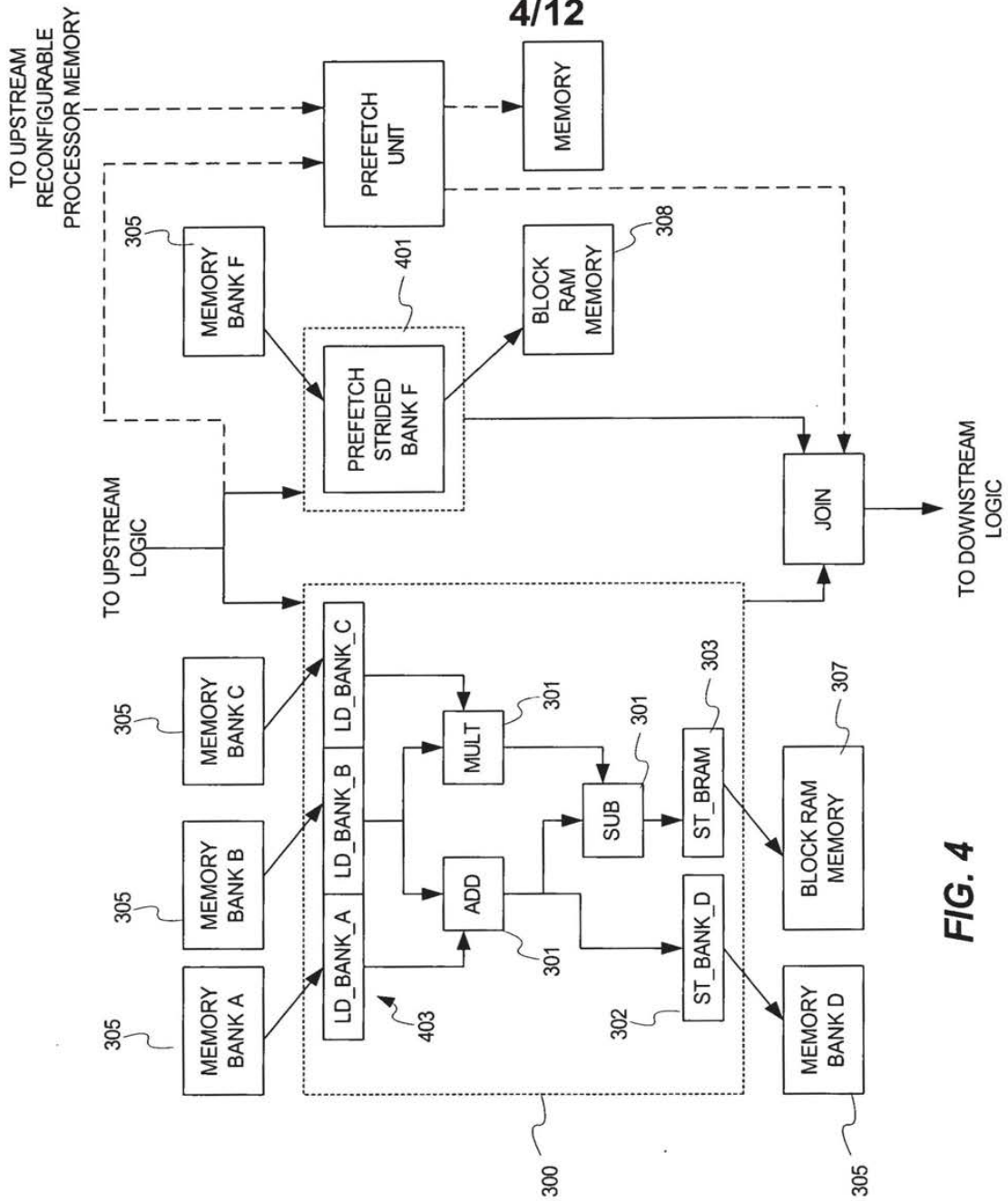


FIG. 4

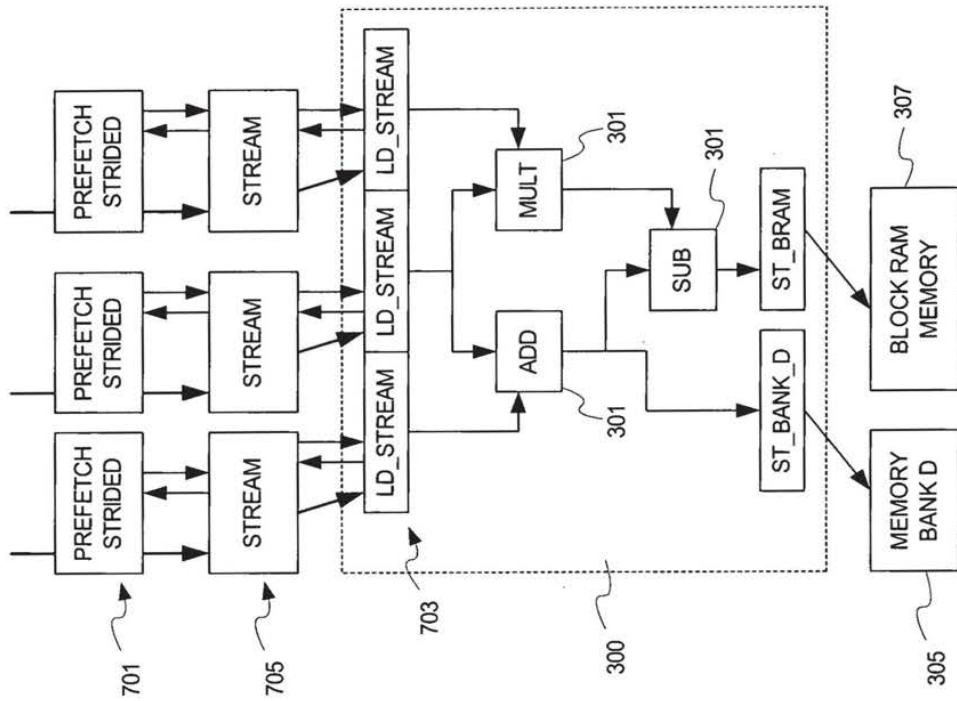


FIG. 7

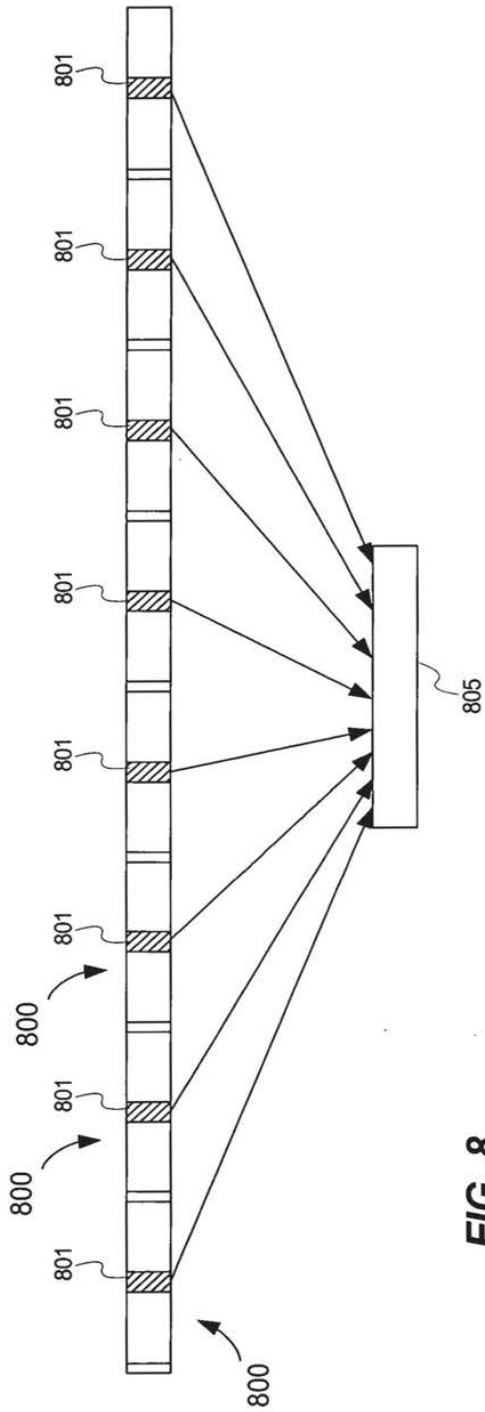
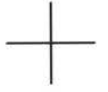


FIG. 8



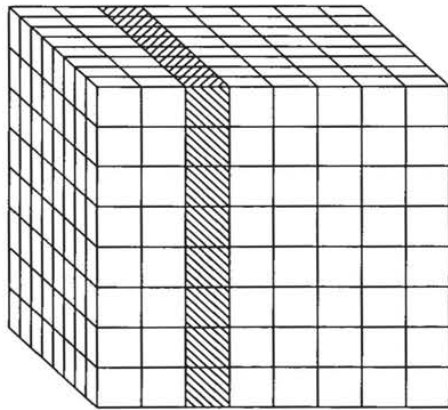


FIG. 9A

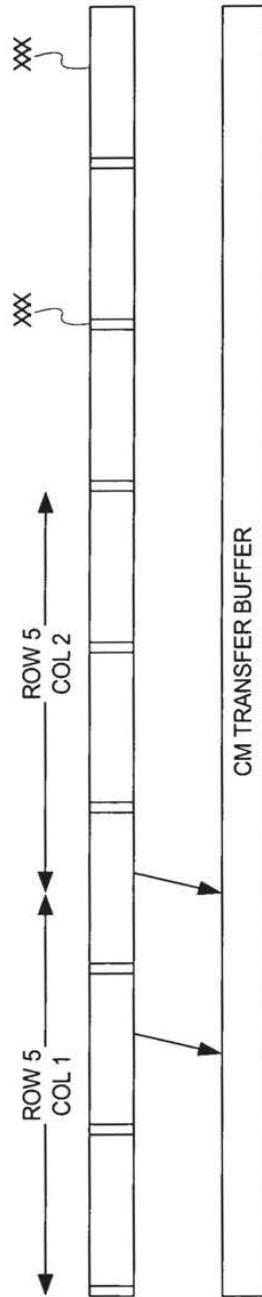


FIG. 9B



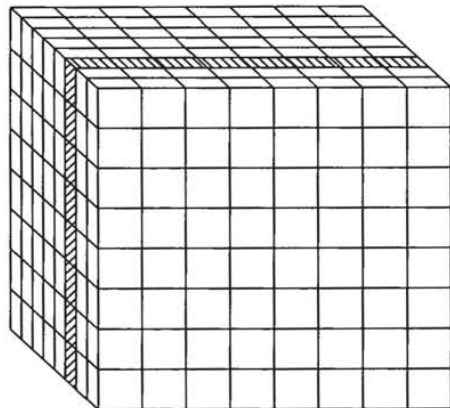


FIG. 10A

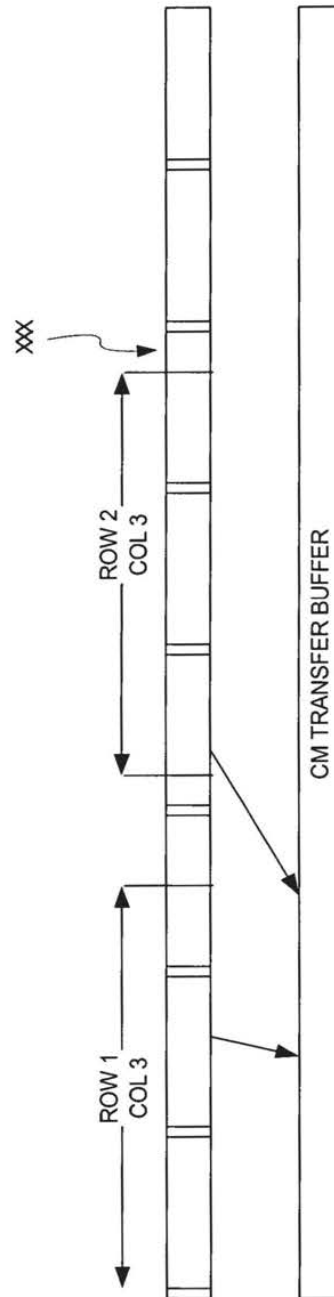


FIG. 10B



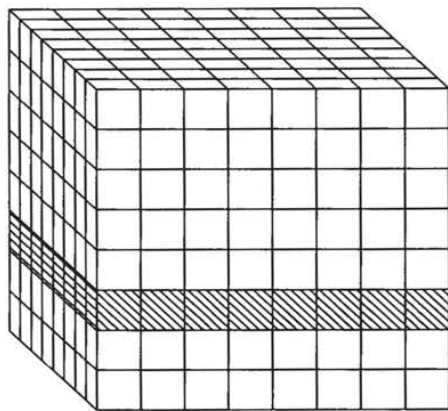
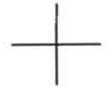


FIG. 11A

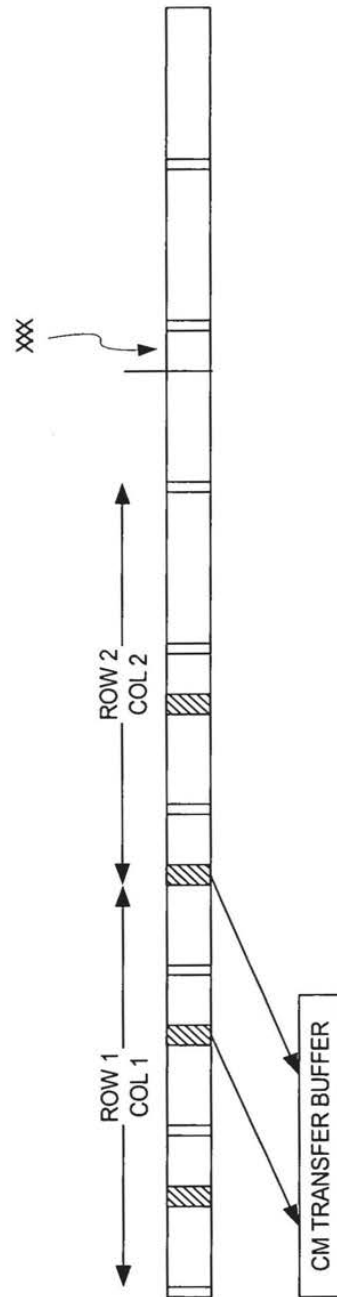


FIG. 11B



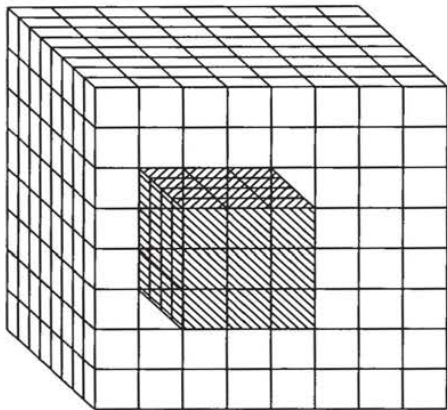


FIG. 12A

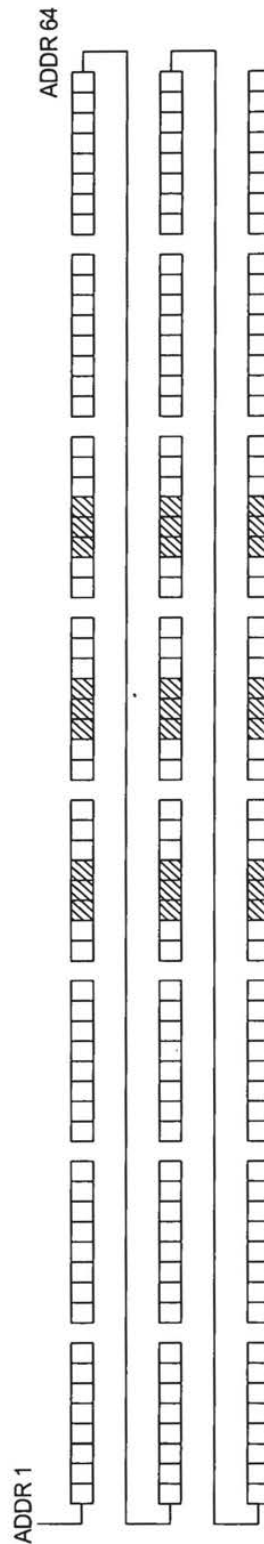


FIG. 12B





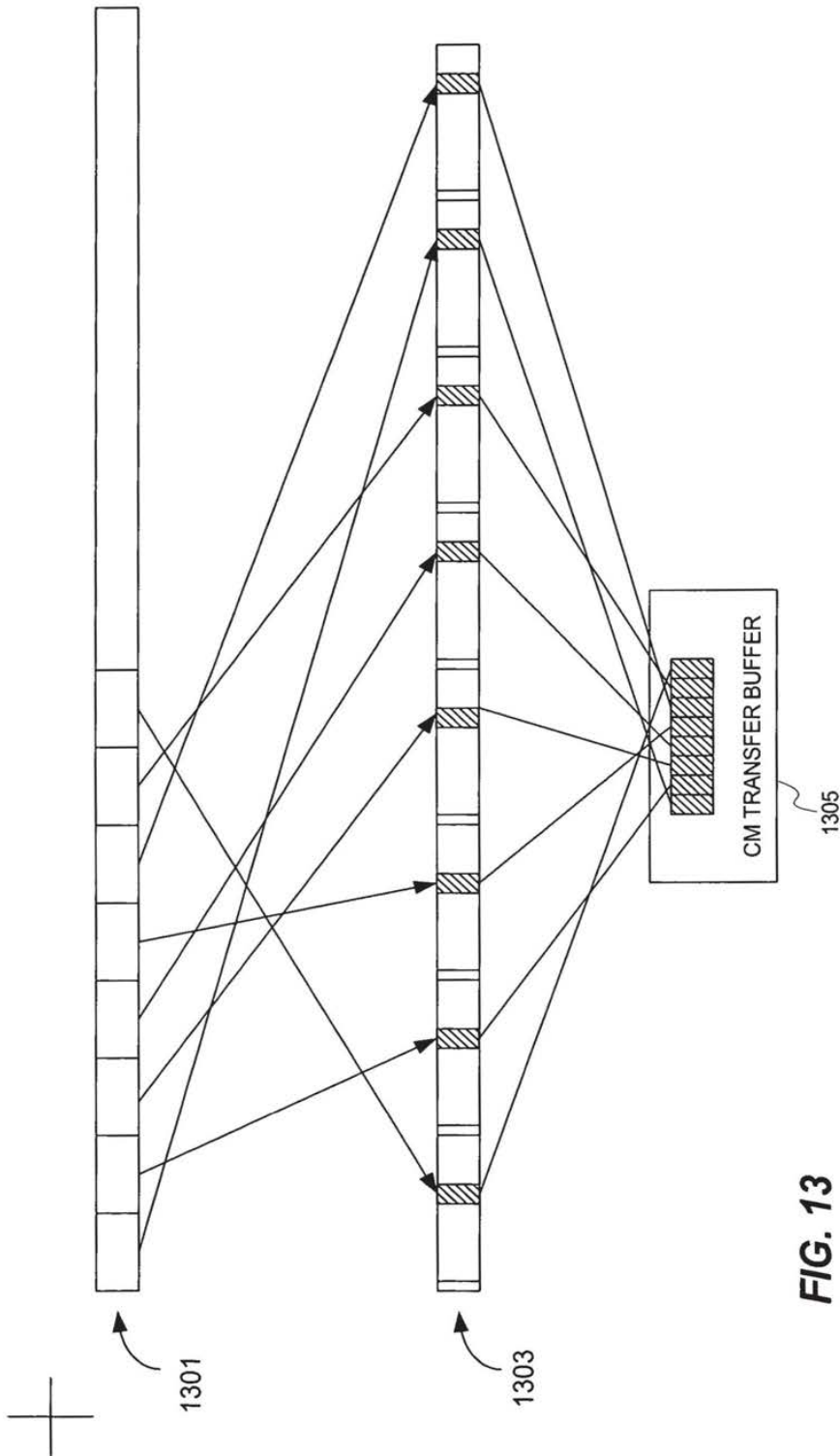


FIG. 13

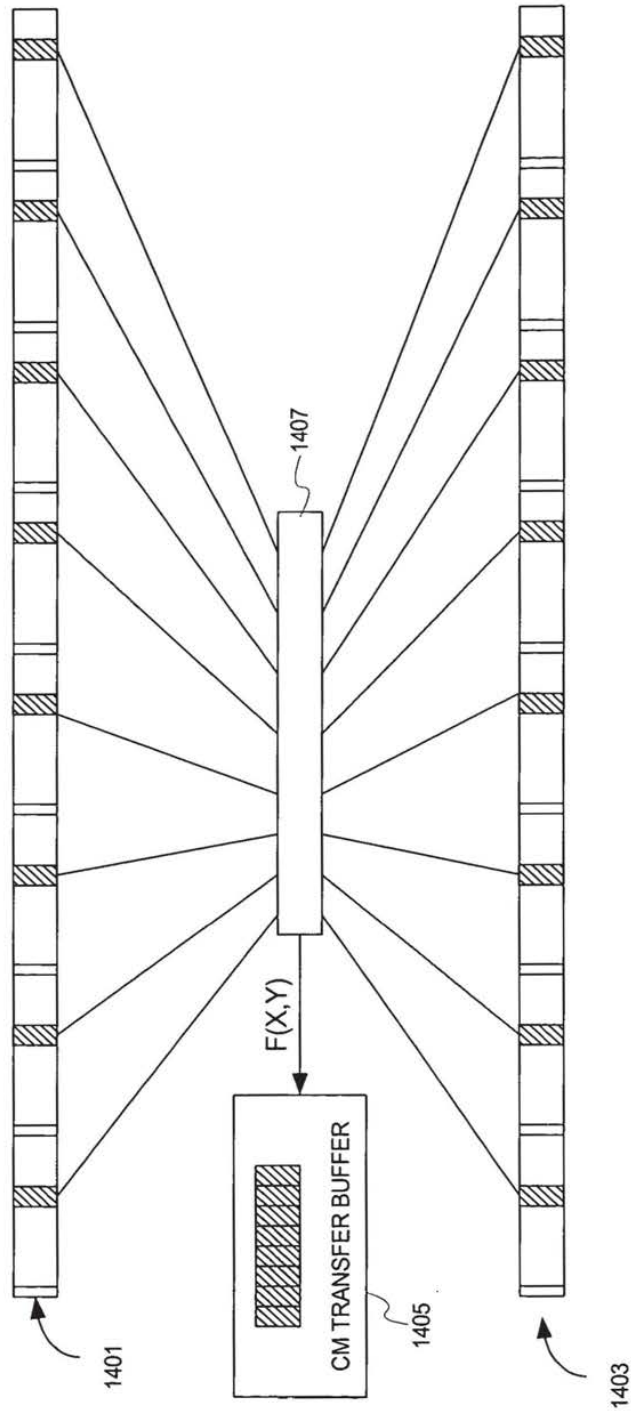
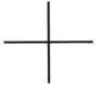


FIG. 14



<b>DECLARATION FOR UTILITY OR DESIGN PATENT APPLICATION (37 CFR 1.63)</b>	<b>Attorney Docket No.</b>	SRC028
	<b>First Named Inventor</b>	Daniel Poznanovic et al.
	<i>COMPLETE IF KNOWN</i>	
	Application Number	-----
<input checked="" type="checkbox"/> Declaration Submitted with Initial Filing    OR <input type="checkbox"/> Declaration Submitted after Initial Filing--surcharge 37 CFR 1.16(e) required	Filing Date	Herewith
	Art Unit	
	Examiner Name	

I hereby declare that:

Each inventor's residence, mailing address, and citizenship are as stated below next to their name.

I believe the inventor(s) named below to be the original and first inventor(s) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY  
BANDWIDTH IN RECONFIGURABLE HARDWARE

the specification of which

is attached hereto

OR

was filed on  
(MM/DD/YYYY)

as U.S. Application No. or  
PCT International Application No.

and was amended on  
(MM/DD/YYYY)

(if applicable)

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment specifically referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in 37 CFR 1.56, including for continuation-in-part applications, material information which became available between the filing date of the prior application and the national or PCT international filing date of the continuation-in-part application.

I hereby claim foreign priority benefits under 35 U.S.C § 119(a)-(d) or (f), or 365(b) of any foreign application(s) for patent or inventor's or plant breeder's rights certificate(s), or § 365(a) of any PCT international application which designated at least one country other than the United States of America, listed below and have also identified below, by checking the box, any foreign application for patent or inventor's or plant breeder's rights certificate(s), or any PCT international application having a filing date before that of the application on which priority is claimed.

Prior Foreign Appl. No.(s)	Country	Foreign Filing Date (MM/DD/YYYY)	Priority Not Claimed	Certified Copy Attached?	
				Yes	No
			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Additional foreign application nos. are listed on a supplemental priority data sheet PTO/SB/02B attached hereto:

I hereby claim the benefit under 35 U.S.C. § 119(e) of any United States provisional application(s) listed below.

Application Number(s)      Filing Date (MM/DD/YYYY)

**60/479,339**

**06/18/2003**

## DECLARATION – Utility or Design Patent Application

I hereby claim the benefit under 35 U.S.C. 120 of any U.S. application(s) or 365(c) of any PCT international application designating the United States of America, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT international application in the manner provided by the first paragraph of 35 U.S.C. 112, I acknowledge the duty to disclose information which is material to patentability as defined in 37 CFR 1.56 which became available between the filing date of the prior application and the national or PCT international filing date of this application

U.S. Parent Application or PCT Parent No.	Parent Filing Date (MM/DD/YY)	Parent Patent No. (if applicable)

Additional U.S. or PCT international application nos. listed on PTO/SB/02B attached hereto.

As a named inventor, I hereby appoint the following registered practitioner(s) to prosecute this application and to transact all business in the Patent Trademark Office connected therewith:

- Customer Number **25235**  
OR  
 Registered practitioner(s) name/registration number listed below

Name	Registration Number	Name	Registration Number

Additional registered practitioner(s) named on supplemental sheet PTO/SB/02C attached hereto.

Direct all correspondence to:  Customer Number **25235** OR  Correspondence address below

Name					
Address					
City		State		ZIP	
Country		Telephone		Fax	

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. 1001 and such willful false statements may jeopardize the validity of the application or any patent issued thereon.

**Name of Sole or First Inventor:**  A petition has been filed for this unsigned inventor.

Given Name (first and middle [if any]) \_\_\_\_\_ Family Name or Surname \_\_\_\_\_

**Daniel** \_\_\_\_\_ **Poznanovic** \_\_\_\_\_

Inventor's Signature  Date **6-16-04**

Residence City **Colorado Springs** State **Colorado** Country **USA** Citizenship **USA**

Mailing Address **1136 Middle Creek Parkway**

City **Colorado Springs** State **Colorado** ZIP **80921** Country **USA**

Additional inventors or a legal representative are being named on the   1   supplemental additional inventor(s) sheet(s) PTO/SB/02A or 02LR attached hereto.

<b>DECLARATION</b>	<b>ADDITIONAL INVENTOR(S)</b> Supplemental Sheet Page <u>  1  </u> of <u>  1  </u>
--------------------	--

Name of Additional Joint Inventor, if any:		<input type="checkbox"/> A petition has been filed for this unsigned inventor					
Given Name (first and middle [if any])		Family Name or Surname					
<b>David E.</b>		<b>Caliga</b>					
Inventor's Signature	<i>David E. Caliga</i>				Date	6/16/2004	
Residence: City	<b>Colorado Springs</b>	State	<b>CO</b>	Country	<b>USA</b>	Citizenship	<b>USA</b>
Mailing Address	<b>8445 Lauralwood Lane</b>						
City	<b>Colorado Springs</b>	State	<b>CO</b>	ZIP	<b>80919</b>	Country	<b>USA</b>
Name of Additional Joint Inventor, if any:		<input type="checkbox"/> A petition has been filed for this unsigned inventor					
Given Name (first and middle [if any])		Family Name or Surname					
<b>Jeffrey</b>		<b>Hammes</b>					
Inventor's Signature	<i>Jeffrey Hammes</i>				Date	6-16-04	
Residence: City	<b>Colorado Springs</b>	State	<b>CO</b>	Country	<b>USA</b>	Citizenship	<b>USA</b>
Mailing Address	<b>870 Vindicator Dr., #311</b>						
City	<b>Colorado Springs</b>	State	<b>CO</b>	ZIP	<b>80919</b>	Country	<b>USA</b>
Name of Additional Joint Inventor, if any:		<input type="checkbox"/> A petition has been filed for this unsigned inventor					
Given Name (first and middle [if any])		Family Name or Surname					
Inventor's Signature					Date		
Residence: City		State		Country		Citizenship	
Mailing Address							
City		State		ZIP		Country	

PATENT APPLICATION SERIAL NO. \_\_\_\_\_

U.S. DEPARTMENT OF COMMERCE  
PATENT AND TRADEMARK OFFICE  
FEE RECORD SHEET

06/21/2004 HVUONG1 00000046 10869200

01 FC:1001	770.00	OP
02 FC:1202	72.00	OP
03 FC:1201	86.00	OP

PTO-1556  
(5/87)

\*U.S. Government Printing Office: 2002 — 489-267/69033

**PATENT APPLICATION FEE DETERMINATION RECORD**  
Effective October 1, 2003

Application or Docket Number

10869 200

**CLAIMS AS FILED - PART I**

	(Column 1)	(Column 2)
TOTAL CLAIMS	24	
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	24 minus 20 =	4
INDEPENDENT CLAIMS	4 minus 3 =	1
MULTIPLE DEPENDENT CLAIM PRESENT		<input type="checkbox"/>

SMALL ENTITY TYPE

OR OTHER THAN SMALL ENTITY

RATE	FEE
BASIC FEE	385.00
X\$ 9=	
X43=	
+145=	
TOTAL	

RATE	FEE
BASIC FEE	770.00
X\$18=	72
X86=	80
+290=	-
TOTAL	928

\* If the difference in column 1 is less than zero, enter "0" in column 2

**CLAIMS AS AMENDED - PART II**

AMENDMENT A	(Column 1)	(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total	*	Minus	**
Independent	*	Minus	***
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM			<input type="checkbox"/>

SMALL ENTITY OR

OTHER THAN SMALL ENTITY

RATE	ADDITIONAL FEE
X\$ 9=	
X43=	
+145=	
TOTAL ADDIT. FEE	

RATE	ADDITIONAL FEE
X\$18=	
X86=	
+290=	
TOTAL ADDIT. FEE	

AMENDMENT B	(Column 1)	(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total	*	Minus	**
Independent	*	Minus	***
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM			<input type="checkbox"/>

RATE	ADDITIONAL FEE
X\$ 9=	
X43=	
+145=	
TOTAL ADDIT. FEE	

RATE	ADDITIONAL FEE
X\$18=	
X86=	
+290=	
TOTAL ADDIT. FEE	

AMENDMENT C	(Column 1)	(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total	*	Minus	**
Independent	*	Minus	***
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM			<input type="checkbox"/>

RATE	ADDITIONAL FEE
X\$ 9=	
X43=	
+145=	
TOTAL ADDIT. FEE	

RATE	ADDITIONAL FEE
X\$18=	
X86=	
+290=	
TOTAL ADDIT. FEE	

\* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.  
 \*\* If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."  
 \*\*\* If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."  
 The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
S1	103	reconfigur\$3 adj (processor micro-processor CPU processor)	US-PGPUB; USPAT	OR	ON	2004/12/11 17:39
S2	125	reconfigur\$3 adj (processor micro-processor CPU microprocessor)	US-PGPUB; USPAT	OR	ON	2004/11/26 15:49
S3	6	reconfigur\$3 adj (processor micro-processor CPU microprocessor) and "711".clas.	US-PGPUB; USPAT	OR	ON	2004/11/26 15:50
S4	0	reconfigur\$3 adj (processor micro-processor CPU microprocessor) and prefetch	US-PGPUB; USPAT	OR	ON	2004/11/26 15:50
S5	11	711/170-173.ccls. and dynamic near3 logic	US-PGPUB; USPAT	OR	ON	2004/12/02 16:34
S6	847	smc.as.	US-PGPUB; USPAT	OR	ON	2004/12/02 16:34
S7	0	smc.as. and "711".clas.	US-PGPUB; USPAT	OR	ON	2004/12/02 16:34
S8	0	smc.as. and "712".clas.	US-PGPUB; USPAT	OR	ON	2004/12/02 16:34
S9	0	(smc and computers) .as. and "712".clas.	US-PGPUB; USPAT	OR	ON	2004/12/02 16:35
S10	0	(smc and computers) .as.	US-PGPUB; USPAT	OR	ON	2004/12/02 16:35
S11	0	(smc and computers).as.	US-PGPUB; USPAT	OR	ON	2004/12/02 16:35
S12	9	(src and computers).as.	US-PGPUB; USPAT	OR	ON	2004/12/02 16:45
S13	72	711/170.ccls. and dynamic\$4 near3 configur\$5	US-PGPUB; USPAT	OR	ON	2004/12/02 16:39
S14	2	711/170.ccls. and dynamic\$4 near3 configur\$5 with cache	US-PGPUB; USPAT	OR	ON	2004/12/02 16:39
S15	196	reconfigurable adj processor	US-PGPUB; USPAT	OR	ON	2004/12/02 17:32
S16	4	"206189".ap.	US-PGPUB; USPAT	OR	ON	2004/12/02 17:34
S17	1	"5024031".pn.	US-PGPUB; USPAT	OR	ON	2004/12/03 11:14
S18	5	"869200".ap.	US-PGPUB; USPAT	OR	ON	2004/12/03 15:30
S19	1401	711/170.ccls.	US-PGPUB; USPAT	OR	ON	2004/12/03 15:30
S20	376	711/170.ccls. and (reconfigur\$5 rearrang\$4 application adj specific)	US-PGPUB; USPAT	OR	ON	2004/12/03 18:33

Search History 1/10/05 7:53:03 AM Page 1  
C:\APPS\EAST\Workspaces\10869200.wsp



S21	93	711/170.ccls. and matrix	US-PGPUB; USPAT	OR	ON	2004/12/03 15:54
S22	50	711/170.ccls. and fpga	US-PGPUB; USPAT	OR	ON	2004/12/03 16:40
S23	186	712/15.ccls.	US-PGPUB; USPAT	OR	ON	2004/12/03 16:41
S24	230	711/170.ccls. and (application near2 specific application-specific)	US-PGPUB; USPAT	OR	ON	2004/12/03 18:18
S25	196	reconfigurable adj processor	US-PGPUB; USPAT	OR	ON	2004/12/03 18:18
S26	129	S25 and fpga	US-PGPUB; USPAT	OR	ON	2004/12/13 11:21
S27	6	S26 and memory with reconfiguring	US-PGPUB; USPAT	OR	ON	2004/12/03 18:30
S28	34	711/170.ccls. and ((reconfigur\$5 rearrang\$4) and application adj specific)	US-PGPUB; USPAT	OR	ON	2004/12/03 18:34
S29	50	711/170.ccls. and FPGA	US-PGPUB; USPAT	OR	ON	2004/12/13 16:42
S30	208	711/170.ccls. and reconfig\$7	US-PGPUB; USPAT	OR	ON	2004/12/14 15:49
S31	1	"6779131".pn.	US-PGPUB; USPAT	OR	ON	2004/12/13 11:25
S32	0	("6779131").URPN.	USPAT	OR	ON	2004/12/13 11:26
S33	9	("5892896"   "6060339"   "6081463"   "6154851"   "6204562"   "6363502"   "6405324"   "6483755"   "6530005").PN.	US-PGPUB; USPAT; USOCR	OR	ON	2004/12/13 11:42
S34	12	direct adj execution adj logic	US-PGPUB; USPAT; USOCR	OR	ON	2004/12/13 11:47
S35	4	711/170.ccls. and programmable adj logic adj blocks	US-PGPUB; USPAT; USOCR	OR	ON	2004/12/13 12:16
S36	5	"869200".ap.	US-PGPUB; USPAT; USOCR	OR	ON	2004/12/13 12:16
S37	3	711/171-172.ccls. and FPGA	US-PGPUB; USPAT	OR	ON	2004/12/13 16:42
S38	50	711/170.ccls. and FPGA	US-PGPUB; USPAT	OR	ON	2004/12/13 16:42
S39	50	711/170.ccls. and FPGA	US-PGPUB; USPAT; USOCR	OR	ON	2004/12/13 16:42

S40	3	711/171-172.ccls. and FPGA	US-PGPUB; USPAT; USOCR	OR	ON	2004/12/13 16:43
S41	8	711/173.ccls. and FPGA	US-PGPUB; USPAT; USOCR	OR	ON	2004/12/13 16:43
S42	78	711/170.ccls. and reprogram\$5	US-PGPUB; USPAT	OR	ON	2004/12/14 15:49
S43	78	711/171-172.ccls. and (reprogram\$5 reconfig\$6)	US-PGPUB; USPAT	OR	ON	2004/12/14 15:50
S44	70	S43 not S42	US-PGPUB; USPAT	OR	ON	2004/12/14 15:51
S45	346	711/170-172.ccls. and ((configur\$5).ti. (configur\$6).ab.)	US-PGPUB; USPAT	OR	ON	2004/12/14 15:52
S46	14	711/170-172.ccls. and ((configur\$5).ti. (configur\$6).ab.) and prefetch	US-PGPUB; USPAT	OR	ON	2004/12/14 15:52
S47	73	711/170-172.ccls. and ((configur\$5).ti. (configur\$6).ab.) and bandwidth	US-PGPUB; USPAT	OR	ON	2004/12/14 15:52
S48	6	711/170-172.ccls. and ((configur\$5).ti. (configur\$6).ab.) and vhdl	US-PGPUB; USPAT	OR	ON	2004/12/14 15:52
S49	35	711/170-172.ccls. and ((configur\$5).ti. (configur\$6).ab.) and matrix	US-PGPUB; USPAT	OR	ON	2004/12/14 15:53
S50	12	711/170-172.ccls. and ((configur\$5).ti. (configur\$6).ab.) and parallelism	US-PGPUB; USPAT	OR	ON	2004/12/14 15:53
S51	1	"6553477".pn.	US-PGPUB; USPAT	OR	ON	2004/12/17 11:03
S52	3	711/170-173.ccls. and reconfigurable adj processor	US-PGPUB; USPAT; JPO	OR	ON	2004/12/30 20:10
S53	9	("20030046530"   "5737524"   "5872919"   "5915104"   "5953512"   "6000014"   "6104415"   "6216219"   "6339819").PN.	US-PGPUB; USPAT; USOCR	OR	ON	2004/12/30 19:28
S54	207	reconfigurable adj processor	US-PGPUB; USPAT; JPO	OR	ON	2004/12/30 19:29
S55	515	reconfigurable adj2 processor	US-PGPUB; USPAT; JPO	OR	ON	2004/12/30 19:29
S56	308	S55 not S54	US-PGPUB; USPAT; JPO	OR	ON	2004/12/30 19:39

S57	104	S55 and ("711" "713").clas.	US-PGPUB; USPAT; JPO	OR	ON	2004/12/30 19:42
S58	7	(adaptive adj processor) and ("711" "713").clas.	US-PGPUB; USPAT; JPO	OR	ON	2004/12/30 19:43
S59	3	S58 not S57	US-PGPUB; USPAT; JPO	OR	ON	2004/12/30 19:43
S60	0	"008128".pa.	US-PGPUB; USPAT; JPO	OR	ON	2004/12/30 20:10
S61	5	"008128".ap.	US-PGPUB; USPAT; JPO	OR	ON	2004/12/30 20:14
S62	34	src adj computers	US-PGPUB; USPAT; JPO	OR	ON	2004/12/30 20:15
S63	15	711/118.ccls. and reconfigurable near3 (memory cache RAM random adj access adj memory processor)	US-PGPUB; USPAT	OR	ON	2005/01/03 13:19
S64	6	711/117.ccls. and reconfigurable near3 (memory cache RAM random adj access adj memory processor)	US-PGPUB; USPAT	OR	ON	2005/01/03 11:58
S65	4	"859051".ap.	US-PGPUB; USPAT	OR	ON	2005/01/03 12:06
S66	1	"6678790".pn.	US-PGPUB; USPAT	OR	ON	2005/01/03 12:29
S67	2	"021492".ap.	US-PGPUB; USPAT	OR	ON	2005/01/10 07:41

52



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/869,200	06/16/2004	Daniel Poznanovic	SRC028	5929
25235	7590	01/14/2005	EXAMINER THOMAS, SHANE M	
HOGAN & HARTSON LLP ONE TABOR CENTER, SUITE 1500 1200 SEVENTEENTH ST DENVER, CO 80202			ART UNIT      PAPER NUMBER 2186	

DATE MAILED: 01/14/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

<b>Office Action Summary</b>	Application No. 10/869,200	Applicant(s) POZNANOVIC ET AL.	
	Examiner Shane M Thomas	Art Unit 2186	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --  
**Period for Reply**

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

**Status**

- 1)  Responsive to communication(s) filed on 16 June 2004.
- 2a)  This action is FINAL.                      2b)  This action is non-final.
- 3)  Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

**Disposition of Claims**

- 4)  Claim(s) 1-24 is/are pending in the application.
  - 4a) Of the above claim(s) \_\_\_\_\_ is/are withdrawn from consideration.
- 5)  Claim(s) \_\_\_\_\_ is/are allowed.
- 6)  Claim(s) 1-24 is/are rejected.
- 7)  Claim(s) \_\_\_\_\_ is/are objected to.
- 8)  Claim(s) \_\_\_\_\_ are subject to restriction and/or election requirement.

**Application Papers**

- 9)  The specification is objected to by the Examiner.
- 10)  The drawing(s) filed on 16 June 2004 is/are: a)  accepted or b)  objected to by the Examiner.
  - Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
  - Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11)  The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

**Priority under 35 U.S.C. § 119**

- 12)  Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
  - a)  All    b)  Some \*    c)  None of:
    - 1.  Certified copies of the priority documents have been received.
    - 2.  Certified copies of the priority documents have been received in Application No. \_\_\_\_\_.
    - 3.  Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

\* See the attached detailed Office action for a list of the certified copies not received.

**Attachment(s)**

- 1)  Notice of References Cited (PTO-892)
- 2)  Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3)  Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)  
Paper No(s)/Mail Date \_\_\_\_\_
- 4)  Interview Summary (PTO-413)  
Paper No(s)/Mail Date \_\_\_\_\_
- 5)  Notice of Informal Patent Application (PTO-152)
- 6)  Other: \_\_\_\_\_

### DETAILED ACTION

This Office action is responsive to the application filed 6/16/2004. Claims 1-24 are presented for examination.

The examiner requests, in response to this Office action, any reference(s) known to qualify as prior art under 35 U.S.C. sections 102 or 103 with respect to the invention as defined by the independent and dependent claims. That is, any prior art (including any products for sale) similar to the claimed invention that could reasonably be used in a 102 or 103 rejection. This request does not require applicant to perform a search. This request is not intended to interfere with or go beyond that required under 37 C.F.R. 1.56 or 1.105.

The request may be fulfilled by asking the attorney(s) of record handling prosecution and the inventor(s)/assignee for references qualifying as prior art. A simple statement that the query has been made and no prior art found is sufficient to fulfill the request. Otherwise, the fee and certification requirements of 37 CFR section 1.97 are waived for those documents submitted in reply to this request. This waiver extends only to those documents within the scope of this request that are included in the application's first complete communication responding to this requirement. Any supplemental replies subsequent to the first communication responding to this request and any information disclosures beyond the scope of this are subject to the fee and certification requirements of 37 CFR section 1.97.

In the event prior art documentation is submitted, a discussion of relevant passages, figs. etc. with respect to the claims is requested. The examiner is looking for specific references to 102/103 prior art that identify independent and dependent claim limitations. Since applicant is

most knowledgeable of the present invention and submitted art, his/her discussion of the reference(s) with respect to the instant claims is essential. **A response to this inquiry is greatly appreciated.**

**The examiner also requests**, in response to this Office action, that support be shown for language added to any original claims on amendment and any new claims. That is, indicate support for newly added claim language by specifically pointing to page(s) and line no(s) in the specification and/or drawing figure(s). This will assist the examiner in prosecuting the application.

#### *Drawings*

The element --computation logic 201-- of paragraph 53 should be corrected to 200 as per figure 2.

#### *Claim Objections*

Claims 1-23 are objected to because of the following informalities:

As per claim 1, the term -- the memory-- of line 3 should be amended to read --the *first* memory-- since --*the* memory-- has not been previously defined. Appropriate correction is required.

As per claim 2, the term --the processor-- should be amended to --the *reconfigurable* processor since the term --*the* processor-- has not been previously defined in the claims.

As per claim 5, line 3, the term --memory-- has been mistakenly duplicated.

Art Unit: 2186

As per claim 8, the term --prefetch unit-- should be amended to --*data* prefetch unit-- since the term --prefetch unit-- has not been previously defined in the claims.

As per claim 11, the term --the unit-- should be amended to --the *data* prefetch unit-- since the term --the unit-- has not been previously defined in the claim.

As per claim 15, the term --at least of the-- of line 2 should be corrected to read --at least *one* of--.

As per claim 17, the term --the data access unit-- of lines 4-5 should be amended to --*a* data access unit-- since the term --*the* data access unit-- has not been previously defined in the claim.

Claims 3,4,6,7,9,10, 12-14,16, and 18-23, are objected to as being dependent on objected claims.

### ***Claim Rejections - 35 USC § 112***

The following is a quotation of the first paragraph of 35 U.S.C. 112:

The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

Claims 1-10, 13, and 14, are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the written description requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention.



Art Unit: 2186

As per claim 1, the terms --first characteristic type-- and --second characteristic type-- are not clearly defined in the Applicant's specification. Applicant is reminded of 37 C.F.R. 1.75 (d)(1) which states that the claim or claims must conform to the invention as set forth in the remainder of the specification and the terms and phrases used in the claims must find clear support or antecedent basis in the description so that the meaning of the terms in the claims may be ascertainable by reference to the description. (See 1.58(a).) The phrases --first characteristic type-- and --second characteristic type-- are not terms of art; nonetheless, for the purposes of examination, the Examiner shall regard the terms as meaning any type of memory (e.g. a SRAM, Flash Rom, DRAM, hard disk, etc.).

As per claims 2 and 13, the Applicant's disclosure does not explicitly mention that the reconfigurable processors cannot have a cache. The disclosure mentions in the Background section, and specifically in paragraphs 16-17, the drawbacks of having a hard-wired cache in a system; however, the Detailed Description does not explicitly state that the reconfigurable processor as taught by the Applicant *cannot* contain a cache. It appears to the Examiner that no specific (hard-wired) cache memory is included in the reconfigurable processor as taught in the disclosure; rather an on-board memory and user-logic can be configured based on a program (paragraph 52). Therefore, for the purposes of examination, the Examiner shall interpret the claim such that the reconfigurable processor of claim 1 does not contain a *hard-wired* (specific) cache.

As per claims 3 and 14, it follows from the rejection for claims 2 and 13, that since Applicant's disclosure does not explicitly state that a reconfigurable processor *cannot* have a cache, the disclose further does not explicitly teach that the reconfigurable processor cannot have

a cache line-sized unit of contiguous data. For the purposes of examination and based on the discussion of claim 2 above, the Examiner shall interpret the limitation of claim 3 such that the reconfigurable processor of claim 1 does not have a *hard-wired* (specific) cache line-sized unit of contiguous data being retrieved from the [second] memory.

As per claim 4, it is not clear to which memory the term --the memory-- refers as --the memory lacks antecedent basis--. For the purposes of examination, the Examiner shall interpret the term --the memory-- to indicate the --second memory-- of claim 1.

As per claim 7, the term --disassembled-- is not known to be a term of art, and further, not specifically defined in the Applicant's specification. Nonetheless, for the purposes of examination, the Examiner shall regard the term --disassembled-- with the broadest reasonable interpretation. Refer to **37 C.F.R. 1.75 (d)(1)**.

Claims 5, 6, and 8-10, are rejected as being dependent on rejected base claim 1.

The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

Claims 2-4, 8-10, and 15-23 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

As per claim 2, it is not clear which memory (first or second memory) the term --the memory-- is referring to since --the memory-- lacks antecedent basis. The Examiner recommends amending the term --the memory-- to overcome this rejection. Nonetheless, for the purposes of examination, the Examiner shall interpret the claim as --the first memory--.

Art Unit: 2186

As per claim 3, it is not clear which memory (first or second memory) the term --the memory-- is referring to since --the memory-- lacks antecedent basis. The Examiner recommends amending the term --the memory-- to overcome this rejection. Nonetheless, for the purposes of examination, the Examiner shall interpret the claim as --the second memory--.

As per claim 4, it is not clear which memory (first or second memory) the term --the memory-- is referring to since --the memory-- lacks antecedent basis. The Examiner recommends amending the term --the memory-- to overcome this rejection. Nonetheless for the purposes of examination, the Examiner shall interpret that claim as --the second memory--.

As per claim 8, it is not clear whether the processor memory is the same as the second memory or if the processor memory is a separate (third) memory since the data prefetch unit is claimed as retrieving data from both a second memory and a processor memory. The Examiner shall interpret the second memory as being a processor memory.

As per claims 15 and 17, it is not clear if the term --the data access unit-- is referring to --the data prefetch unit-- or is a new entity being defined by the claim since the term --the data access unit-- lacks antecedent basis. Nonetheless, for the purposes of examination, the Examiner shall regard the term --the data access unit-- to be a separate entity based in part from the Applicant descriptions of the drawings on page 8 showing that the data prefetch unit and data access unit are distinct entities.

As per claim 19, it is not clear whether the term --a data access unit-- is the same data access unit that has been defined in claim 17 or the --a data access unit-- is a different data access unit that performs the limitation of claim 19 and does not perform the limitation of the data

Art Unit: 2186

access unit of claim 17. For the purposes of examination, the Examiner shall interpret the --a data access unit-- as --*the* data access unit-- [of claim 17].

As per claims 9-10 and 16-23, the claims are rejected as being dependent on rejected claims.

***Claim Rejections - 35 USC § 102***

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

Claims 1-24 are rejected under 35 U.S.C. 102(e) as being anticipated by Paulraj (U.S. Patent Application Publication No. 2003/0084244).

As per claim 1, Paulraj shows a reconfigurable processor in figure 6 and a first memory (L1) having a first characteristic memory type (line size, blocking factor, associativity, etc.) and a second memory (L2) having a second characteristic memory type (line size, blocking factor, associativity, etc.). Refer to paragraph 23. Paulraj further teaches a functional unit 102 that executes applications using the memories L1 and L2 (paragraph 9). As is known in the art, a cache memory controller is often used to access and move data between a memory hierarchy. The Examiner is considering a data prefetch unit to be the logic associated with the moving, and only the moving, of data between the first and second memories (L1 and L2) since Paulraj shows

Art Unit: 2186

a connection between the levels of cache in figure 6. This logic as well as the first and second memory types (L1 and L2) are configured by a program – refer to paragraphs 23-24. The data prefetch unit as defined by the Examiner must be configured as well by the program when moving data since the cache line size and blocking factor can change, so different amounts of data can be exchanged for the same access when different programs run.

As per claims 2 and 13, as taught in paragraphs 23 and 29 of Paulraj, no specific cache is present in the system of Paulraj. Rather, an FPGA is utilized as representing a caching hierarchy and is optimized based on the memory needs of a specific program running on the reconfigurable processor.

As per claims 3 and 14, Paulraj teaches in paragraph 23 that a specific cache line size of contiguous data is not retrieved since the data line size is optimized based on the memory needs of the program when executing on the reconfigurable processor. Refer also to paragraph 29.

As per claim 4, Paulraj teaches that a load/store unit is used to access the caches (L1-L3) in order to determine if cache data is present in the cache hierarchy (paragraph 6). Since the functional unit 102 (figure 6) is responsible for accessing the programmable memory unit 104, the Examiner is therefore considering the load/store unit logic of the programmable memory unit that is responsible for for accessing the L1 and L2 caches (first and second memory types) to be a memory controller. It can be seen that the memory controller, as defined by the Examiner, controls the transfer of data between the memory (assuming second memory L2) and the data prefetch unit, since the memory controller (load/store unit logic) is responsible for retrieving the data from the cache if a hit occurs (paragraph 4).

As per claim 5, as taught in paragraph 1, an external memory (element 18, figure 1) is generally coupled to a microprocessor and holds data to be used by the microcontroller during program execution. The Examiner is considering the process of writing data back to the external memory from the FPGA memory 104 containing the caches (on-board memory), such as during a write-back scheme as known in the art, to be performed by the data prefetch unit portion of the functional logic as defined above by the Examiner. The data prefetch logic, as defined above, is responsible for all of the transfer of data into, out of, and between the FPGA memory 104.

As per claim 6, the Examiner is regarding a --register-- in its broadest reasonable sense and it thus considering it be to be a unit of logic. Therefore, the portion of the function logic that is responsible for the movement of data (as defined above to be the data prefetch unit) is being considered by the Examiner as containing a --register-- portion of the reconfigurable processor since, for instance, the blocking factor and line size of the programmable memory 112 can change, a --register-- or portion of the reconfigurable processor must be set in order to indicate the current line size and blocking factor when a given application is being run on the reconfigurable processor at a given point in time. Refer to paragraph 23.

As per claim 7, the Examiner is considering the process of --disassembling the data prefetch unit-- as modifying the data prefetch unit logic of the function logic 102 every time the program being executed by the reconfigurable processor changes. It can be seen that the data prefetch unit changes during these intervals since the cache line size, blocking factor, and associativity of the FPGA changes when optimal for the next program to be executed (refer to paragraph 23). Thus it can be seen that the data prefetch unit logic is --disassembled-- when another program is executed by the reconfigurable processor of Paulraj.

As per claim 8, as can be seen that the FPGA memory 112, that comprises the first and second memories (L1 and L2) and which is accessed by the data prefetch unit of the functional unit 102 as discussed above, is a --processor memory-- (part of cpu 110). Therefore, since the data prefetch unit can access the L2 cache as discussed above in the rejection of claim 1, the data prefetch unit can retrieve data from the L2 portion of --processor memory-- 112.

As per claim 9, as shown in figure 1 and taught in paragraph 1 of Paulraj, the system 10 is actually a microprocessor, which contains a memory controller 14. The main difference between the prior art of figure 1 and the invention of Paulraj in figure 6 is that the memory hierarchy is configurable and accessed by a functional unit in lieu of a separate memory controller logic (paragraph 9). Therefore, since the memory controller logic for accessing the cache hierarchy is still contained within cpu 110 of figure 6, it can be seen that the cpu 110 is actually a microprocessor. It follows that the --processor memory-- 112 is therefore a --microprocessor memory--.

As per claim 10, since the cpu 110 of figure 6 is a reconfigurable processor (able to reconfigure its memory hierarchy to match the needs of the application it is currently running), it can be seen that the cpu memory 112 is a reconfigurable processor memory.

As per claim 11, Paulraj depicts a reconfigurable hardware system in figure 6. Paulraj further teaches in paragraph 26 that when a particular application is to be run by the reconfigurable processor 110, a configuration vector is retrieved to program the programmable memory 112 (figure 6). As shown in figure 6, the step of accessing the configuration vector is executed outside of the reconfigurable processor 110. Therefore, the Examiner is considering the memory that contains the configuration vectors to be a--common memory-- and a data

Art Unit: 2186

prefetch unit (reconfiguration unit 106 executing on the reconfigurable processor 110) accessing the common memory in order to determine how to program the memory 112 (paragraph 29).

The data prefetch unit 106 is --configured-- by an application to be executed on the system 110 since when a new application is to be executed, the data prefetch unit is called upon (or configured) to access the configuration vector for the particular application.

As per claim 12, the Examiner is considering a --memory controller-- to be the system portion utilized when creating a new configuration vector for an application. Such a process occurs in figure 5 and taught in paragraphs 23-25 of Paulraj. When a new configuration vector is created by analyzing performance information that has been collected for the application. The Examiner is thereby considering the --memory controller-- to be the element of the reconfigurable hardware system that is associated with storing the new configuration vector into the common memory so that the vector can be accessed later when the same application is run again.

As per claim 15, the Examiner is considering the reconfiguration module 106 of the reconfigurable processor 110, as comprising two distinct elements: a --computational unit-- and a --data access unit--. The data access unit is the element that is responsible for accessing the configuration vector as taught in paragraph 29 of Paulraj; or in other words, the Examiner is considering the --data access unit-- to be the same as the --memory controller-- defined in the rejection of claim 12. The Examiner is further considering the --computational unit-- of the reconfiguration module 106 to be the element that sets up the programmable memory module 104 using the configuration vector that was accessed by the --data access unit-- (paragraph 29).



As per claim 16, as taught by Paulraj in paragraph 29, the --data access unit-- supplies the configuration vector to the --computational unit-- in order to set up the programmable memory 104 as required by the application to be run on the reconfigurable processor 110.

As per claim 17, the Examiner is considering a --data prefetch unit-- to be the reconfiguration unit 106 of reconfigurable processor 110 (figure 6). As taught in paragraph 26 and 29 of Paulraj, the --data prefetch unit-- accesses a memory in order to determine if a configuration vector is known for a given application, and if so, the vector is retrieved (from the memory). If this --data-- (configuration vector) is not known then a simulation is performed with the application in order to collect performance information. The Examiner is considering the element that executes and collects the performance data as being a --computational unit-- and the element of Paulraj that stores the configuration vector, once determined, to be a --data access unit-- since it stores the vector into the --memory-- from which it can be later retrieved (step 212 of figure 5). The --computational unit-- , --data access unit-- , and the --data prefetch unit-- are all --configured-- by a program (application) since (1) a new application configures the computational unit portion of the reconfiguration unit to perform a simulation in order to determine the optimal memory hierarchy organization; (2) the new application configures the --data access unit-- to store and retrieve (step 212) the configuration vector for that particular application; and (3) the --data prefetch unit-- is configured by the application to determine if a configuration file exists for the application and if so, the data prefetch unit is configured by the program the programmable memory 112 in order to optimize the programmable memory for that particular application.

As per claim 18, the --data-- (configuration vector) is transferred from the

Art Unit: 2186

--computational unit-- to the --data access unit-- when the configuration unit has created a configuration vector (step 208 of figure 5). The --data-- is written to the memory --from-- the --data prefetch unit-- since the data prefetch unit (reconfiguration unit 106) is the element that executed the beginning of the configuration vector creation process (step 200 of figure 5). Refer to paragraph 26. Thus the Examiner is considering the data as being written --from-- the data prefetch unit.

As per claim 19, as taught in paragraph 26, if the configuration vector is known, the vector is retrieved from the memory to the data prefetch unit (reconfiguration unit 106). The data is read directly from the data prefetch unit when a request to create a configuration vector is made for a new application as shown in figure 6 since the data prefetch unit is responsible for being the vector creation process. The data is directed from the data prefetch unit (reconfigure logic) to be read from the memory by the data access unit to the computational unit where it is processed to produce a configuration vector.

As per claim 20, as stated above, the configuration vector (--data--) is created by the computational unit via acquired simulation data. The configuration vector is the resultant product that is transferred from the memory to the data prefetch unit when it is determined that the configuration vector for the application is available (paragraph 26). Thus --all-- of the data that is transferred is processed by the computational unit (albeit before the transfer occurs) since the data prefetch unit required the entire configuration vector in order to set up the programmable memory 112.

As per claim 21, Paulraj shows in paragraph 26 that an explicit request for the configuration vector for the current application results in the data (if it exists) selected for the optimal configuration of the programmable memory 112 for that application.

As per claim 22, the Examiner is not considering the data (configuration vector) to be the size of a complete cache line since the data is used to create a cache hierarchy. In other words, the caches (L1-L3) of the programmable memory 112 are not programmed when the data is transferred from the memory to the data prefetch unit; therefore, the data cannot be a complete cache line.

As per claim 23, since the Examiner defined the portion of the reconfiguration unit that accesses the configuration file (data) from the memory, the Examiner is defining the logic that controls the actual transfer of that data to the data prefetch unit (portion of the reconfiguration unit that executes the fetch of the configuration vector and then programs the programmable memory 112) to be a --memory controller--. Thus the data access unit determines whether a configuration vector exists for an application and if so, the memory controller sends that data to the data prefetch unit.

As per claim 24, The Examiner is considering the element that executes and collects the performance data as being a --computational unit-- and the element of Paulraj that stores and retrieves the configuration vector, once determined, to be a --data access unit-- since it stores the vector into the --memory-- from which it can be later retrieved (step 212 of figure 5). The --computational unit-- and --data access unit -- are --configured-- by a program (application) since (1) a new application causes in the configuration of the computational unit portion of the reconfiguration unit to perform a simulation in order to determine the optimal memory hierarchy

organization for the application and (2) the new application causes the configuration of the --data access unit-- to store and retrieve (step 212) the configuration vector for that particular application. Refer to paragraphs 25-27.

### *Conclusion*

The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

Poznanovic (U.S. Patent Application Publication No. 2003/0046530) teaches a reconfigurable processor (figure 2) which can be reprogrammed based on a program.

Vondran (U.S. Patent No. 6,243,791) illustrates an example of the operation of a cache controller in a cache hierarchy (column 1, lines 54-67).

Otterness (U.S. Patent No. 6,460,122) further teaches common operation of a cache controller in column 21, lines 1-16.

Darling (U.S. Patent No. 6,714,041) teaches a reconfigurable system (figure 5) that is able to be reprogrammed based on a program.

Burton (U.S. Patent Application Publication No. 2003/0088737) teaches uncached device operations in a reconfigurable processor system.

Gschwind et al. (U.S. Patent Application Publication No. 2003/0046492) teaches a reconfigurable memory array which can be operated as a cache or a non-cache memory.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Shane M Thomas whose telephone number is (703) 605-0725.

Art Unit: 2186

Please note: the aforementioned number will change to (571) 272-4188 effective October 19, 2004. The examiner can normally be reached M-F 8:30 - 5:30.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Matt M Kim can be reached on (703) 305-3821, which will change to (571) 272-4182 effective October 19, 2004. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).



Shane M. Thomas



**MATTHEW ANDERSON**  
**PRIMARY EXAMINER**  
**GROUP 2/00**

<b>Notice of References Cited</b>	Application/Control No. 10/869,200	Applicant(s)/Patent Under Reexamination POZNAOVIC ET AL.	
	Examiner Shane M Thomas	Art Unit 2186	Page 1 of 1

**U.S. PATENT DOCUMENTS**

*	Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
A	US-2003/0084244 A1	05-2003	Paulraj, Dominic	711/118
B	US-2003/0046530 A1	03-2003	Poznanovic, Daniel	713/100
C	US-6,243,791	06-2001	Vondran, Jr., Gary Lee	711/120
D	US-6,460,122	10-2002	Otterness et al.	711/154
E	US-6,714,041	03-2004	Darling et al.	326/38
F	US-2003/0088737	05-2003	Burton, Lee	711/118
G	US-2003/0046492 A1	03-2003	Gschwind et al.	711/118
H	US-			
I	US-			
J	US-			
K	US-			
L	US-			
M	US-			

**FOREIGN PATENT DOCUMENTS**

*	Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
N					
O					
P					
Q					
R					
S					
T					

**NON-PATENT DOCUMENTS**

*	Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
U	"Summary: The Cache Read/Write Process," The PC Guide, 2001, www.pcguide.com/ref/mbsys/cache/func.htm.
V	Chien et al., "Safe and Protected Execution for the Morph/AMRM Reconfigurable Processor," IEEE, 1999, pp 1-13.
W	
X	

\*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)  
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Get the Next Generation in IM Smileys

[Click here!](#)



**NOTE:** Using robot software to mass-download the site degrades the server and is prohibited. [See here for more.](#)

Find The PC Guide helpful? Please consider a donation to [The PC Guide Tip Jar](#). Visa/MC/Paypal accepted. Interested in discussing the latest news, current events and other topics of general interest? Check out [CurEvents.com!](#)

[ [The PC Guide](#) | [Systems and Components Reference Guide](#) | [Motherboard and System Devices](#) | [System Cache](#) | [Function and Operation of the System Cache](#) ]

### **Summary: The Cache Read/Write Process**

Having looked at all the parts and design factors that make up a cache, in this section the actual process is described that is followed when the processor reads or writes from the system memory. This example is the same as in the other sections on this page: 64 MB memory, 512 KB cache, 32 byte cache lines. I will assume a direct mapped cache, since that is the simplest to explain (and is in fact most common for level 2 cache):

1. The processor begins a read/write from/to the system memory.
2. Simultaneously, the cache controller begins to check if the information requested is in the cache, and the memory controller begins the process of either reading or writing from the system RAM. This is done so that we don't lose any time at all in the event of a cache miss; if we have a cache hit, the system will cancel the partially-completed request from RAM, if appropriate. If we are doing a write on a write-through cache, the write to memory always proceeds.
3. The cache controller checks for a hit by looking at the address sent by the processor. The lowest five bits (A0 to A4) are ignored, because these differentiate between the 32 different bytes in the cache line. We aren't concerned with that because the cache will always return the whole 32 bytes and let the processor decide which one it wants. The next 14 lines (A5 to A18) represent the line in the cache that we need to check (notice that  $2^{14}$  is 16,384).
4. The cache controller reads the tag RAM at the address indicated by the 14 address lines A5 to A18. So if those 14 bits say address 13,714, the controller will examine the contents of tag RAM entry #13,714. It compares the 7 bits that it reads from the tag RAM at this location to the 7 address bits A19 to A25 that it gets from the processor. If they are identical, then the controller knows that the entry in the cache at that line address is the one the processor wanted; we have a hit. If the tag RAM doesn't match, then we have a miss.
5. If we do have a hit, then for a read, the cache controller reads the 32-byte contents of the cache data store at the same line address indicated by bits A5 to A18 (13,714), and sends them to the

Google Search

Web The PC Guide

Ads by Goooooogle

#### **Cache Upgrade**

Huge Selection of Merchants Compare & Find the Lowest Rate!  
[www.bottomdollar.com](http://www.bottomdollar.com)

#### **Cache Upgrade**

Merchants compete for your business Compare instant bottom-line prices!  
[www.pricerabber.com](http://www.pricerabber.com)

#### **Computer Cache Memory**

Compare Prices, Read Reviews & More Find the Lowest Prices at Smarter  
[www.smarter.com](http://www.smarter.com)

#### **Cache cpu memory**

Find the best components prices Compare products, stores & reviews  
[cpus.motherboards.nextag.com](http://cpus.motherboards.nextag.com)

#### **Cache Upgrade**


New & Used Cache Upgrade. aff Check out the deals now!  
[www.ebay.com](http://www.ebay.com)

**BEST AVAILABLE COPY**

processor. The read that was started to the system RAM is canceled. The process is complete. For a write, the cache controller writes 32 bytes to the data store at that same cache line location referenced by bits A5 to A18. Then, if we are using a write-through cache the write to memory proceeds; if we are using a write-back cache, the write to memory is canceled, and the dirty bit for this cache line is set to 1 to indicate that the cache was updated but the memory was not.

6. If we have a miss and we were doing a read, the read of system RAM that we started earlier carries on, with 32 bytes being read from memory at the location specified by bits A5 to A25. These bytes are fed to the processor, which uses the lowest five bits (A0 to A4) to decide which of the 32 bytes it wanted. While this is happening the cache also must perform the work of storing these bytes that were just read from memory into the cache so they will be there for the next time this location is wanted. If we are using a write-through cache, the 32 bytes are just placed into the data store at the address indicated by bits A5 to A18. The contents of bits A19 to A25 are saved in the tag RAM at the same 14-bit address, A5 to A18. The entry is now ready for any future request by the processor. If we are using a write-back cache, then before overwriting the old contents of the cache line, we must check the line's dirty bit. If it is set (1) then we must first write back the contents of the cache line to memory, and then clear the dirty bit. If it is clear (0) then the memory isn't stale and we continue without the write cycle.
7. If we have a cache miss and we were doing a write, interestingly, the cache doesn't do much at all, because most caches don't update the cache line on a write miss. They just leave the entry that was there alone, and write to memory, bypassing the cache entirely. There are some caches that put all writes into the appropriate cache line whenever a write is done. They make the general assumption that anything the processor has just written, it is likely to read back again at some point in the near future. Therefore, they treat *every* write as a hit, by definition. This means there is no check for a hit on a write; in essence, the cache line that is used by the address just written is always replaced by the data that was just put out by the processor. It also means that on a write miss the cache controller must update the cache, including checking the dirty bit on the entry that was there before the write, exactly the same as what happens for a read miss.

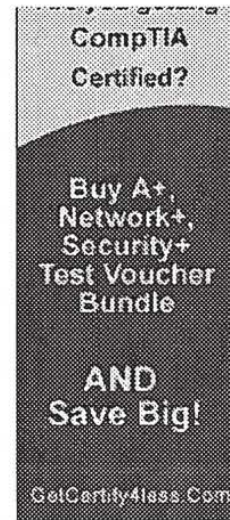
As complex as it already is :^) this example would of course be even more complex if we used a set associative or fully associative cache. Then we would have a search to do when checking for a hit, and we would also have the matter of deciding which cache line to update on a cache miss.

 Next: [Cache Characteristics](#)

[Home](#) - [Search](#) - [Topics](#) - [Up](#)

The PC Guide (<http://www.PCGuide.com>)  
Site Version: 2.2.0 - Version Date: April 17, 2001

**BEST AVAILABLE COPY**



CompTIA  
Certified?

Buy A+,  
Network+,  
Security+  
Test Voucher  
Bundle

AND  
Save Big!

GetCertify4less.Com



# Safe and Protected Execution for the Morph/AMRM Reconfigurable Processor

Andrew A. Chien  
Department of Computer Science and Engineering  
University of California, San Diego  
[achien@cs.ucsd.edu](mailto:achien@cs.ucsd.edu)

Jay H. Byun  
Department of Computer Science  
University of Illinois at Urbana-Champaign  
[jaybyun@cs.uiuc.edu](mailto:jaybyun@cs.uiuc.edu)

April 1, 1999

## Abstract

Technology scaling of CMOS processes brings relatively faster transistors (gates) and slower interconnects (wires), making viable the addition of reconfigurability to increase performance. In the Morph/AMRM system, we are exploring the addition of reconfigurable logic, deeply integrated with the processor core, employing the reconfigurability to manage the cache, datapath, and pipeline resources more effectively. However, integration of reconfigurable logic introduces significant protection and safety challenges for multiprocess execution. We analyze the protection structures in a state of the art microprocessor core (R10000), identifying the few critical logic blocks and demonstrating that the majority of the logic in the processor core can be safely reconfigured. Subsequently, we propose a protection architecture for the Morph/AMRM reconfigurable processor which enable nearly the full range of power of reconfigurability in the processor core while requiring only a small number of fixed logic features which to ensure safe, protected multiprocess execution.

## 1. Introduction

Trends in semiconductor technology suggest that the use of reconfigurable logic blocks within the processor will be desirable in the future. Projections from Semiconductor Industry Association(SIA) for the year 2007 indicate advanced semiconductor processes using 0.1 micron feature sizes [1]. However, this feature size, as measured by transistor channel length, is of decreasing importance to logic and circuit as well as processor speed. In systems of that era, logic density, logic speed, and processor speed will be dominated by interconnect performance and wiring density. For 2007, the SIA projects pitch for the finest interconnect at 0.4-0.6 microns. Between logic blocks, average interconnect lengths typically range from from 1,000x to 10,000x pitch -- up to 6mm of intra-chip interconnect length. For such an interconnect, the achievable global clock speed would be limited to approximately 1 nanosecond. Within a few technology generations, a crossover will occur, and the average interconnect delay will surpass logic block delays -- projections indicate that by the year 2007, average interconnect delay can be equivalent to five gate delays.

Once past the cross-over point, dynamic interconnect (reconfigurable interconnect or logic) can be introduced at modest impact even on critical timing paths[2]. In such systems, the dynamic configurability in the processor can be used to significant advantage [4, 5], improving performance by factors of 10 to 100x for computational kernels while avoiding the traditional disadvantages of custom computing approaches such as I/O coprocessor coupling and slower logic [6]. In these systems, reprogrammable logic blocks will replace static interconnects in the processor core, paving the way for a new class of architectures which are customized to the application, delivering more robust and higher performance.

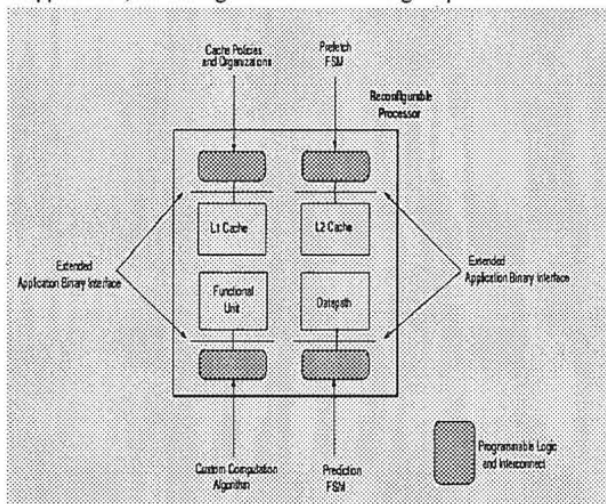


Figure 1. Reconfigurability in the processor core and the extended application to fixed hardware interface

Reconfigurable, or application adaptive processors allow customization of mechanisms, bindings, and policies on a per application basis. While current microprocessors implement a number of aggressive architectural techniques such as speculative execution, branch prediction, block prefetching, multi-level caching, etc. to achieve higher execution speeds, these mechanisms and policies are tuned

BEST AVAILABLE COPY

for a broad suite of applications (e.g. SPEC), and thus cannot be tightly matched to the needs of a particular application, procedure, or even loop in an application. For example, the cache block size and organization is chosen to maximize performance over a suite of applications, but may not give best performance on any particular application. Similar constraints apply to other performance critical aspects such as value prediction, branch prediction, and data movement. In contrast, a processor incorporating reconfigurability can adopt optimal policies (and in some cases better mechanisms) for the application, enabling increased execution efficiency. Thus, the reconfigurable logic can be used to tune the processor to better match the application, rather than the more traditional view of thinking of it as an add-on coprocessor. One example of this per-application basis tuning would be to adapt the cache line size to maximize performance for that application[3]. This approach is embodied in the Morph/AMRM (Adaptive Memory Reconfiguration Management) architecture [4, 5], and the basic change in perspective is that the reconfigurable hardware is an extension of the application program, extending the application -- fixed hardware interface to enable more efficient execution. The fixed hardware then has a somewhat richer (and in parts lower level) interface as shown in Figure 1. Studies of Morph/AMRM have demonstrated that performance increases of ten to 100 times are possible [5]. In essence, this is an extension of the application binary interface (so-called ABI), but need not be a non-portable extension of the application programming interface (API) if appropriate CAD support is available. This approach is similar to that which has recently gained popularity in the software design community as "open implementations" [7] in which software architects recognize the need to open the implementation for customization for particular application uses in order to achieve adequate performance.

Introducing application-controlled reconfigurability in the processor raises significant challenges for ensuring process isolation and protection (multiprocess isolation), a critical element of robust desktop and to an increasing degree, embedded computing systems. Multiprocess isolation is an essential modularity element in software systems: without the guarantee of safely isolated and protected processes, the system can never be robust since software faults cannot be contained and the system cannot be safely extended. It is essential for robust reconfigurable computing that an application's customization only affect its computation, not that of other applications. For example, if application-defined hardware were allowed to control hardware addressing, it could allow unauthorized corruption of operating system data or even the data of other application processes. If an application-defined hardware were allowed to control data prefetching, it could swamp the memory system with spurious requests. If

application-defined hardware were allowed to control privilege mode changes, it could compromise all traditional protection structures.

Our study examines the protection structures of traditional processors and operating systems, and based on these lessons, proposes a safe multiprocess execution architecture for reconfigurable systems. We analyze in detail the software and hardware mechanisms central to the process protection in conventional processors and OS, specifically studying the MIPS R10000 [8] microprocessor, an exemplar of a system employing Unix/RISC protection architecture. This study elucidates the key mechanisms and architectural features for Unix style two mode protection, and addressing based isolation. The key feature of this protection architecture is process isolation via address isolation and mediation. Specifically,

1. All access to hardware devices is mediated by the operating system,
2. The operating system manages address translation to isolate processes,
3. Application processes cannot change the address translation information,
4. Application processes cannot substitute other translation information,
5. All application accesses are subject to this translation, and
6. The hardware ensures these guarantees

We subsequently describe the Morph/AMRM architecture, outlining the dimensions of configurability and the hazards for multiprocess protection they induce. For the Morph/AMRM system, we then describe the protection architecture, describing in detail how each of the key properties of the operating system / processor protection architecture are provided. The key elements of this protection architecture are:

1. A hardwired control processor which controls instruction sequence and privilege mode transitions
2. A hardwired control processor to TLB control for address translation and TLB entry management
3. A requirement for all other configurable elements (system chip sets, input/output devices, memory controllers) must deal in virtual addresses, and their accesses are checked by local TLBs
4. Controlled access to key shared interconnects such as the system bus are controlled by hardwired arbiters which are not changed, system reserves highest priority to allow preemption for these resources

This architecture enables configurability in the processor complex because it can ensure multiprocess protection (safe configuration). We also believe it enables much of the useful configurability in the processor complex, notably policies for improving efficient management of resources and even the addition of instructions, special functional

units, or even processor state. The model provided to application programs is a private, configurable, virtual machine which enables rich application customization. These applications (and their customizations) are cleanly isolated.

The remainder of the paper is organized as follows. Section 2 describes the basic problem of protected execution and process isolation in computer systems. Section 3 describes our analysis of the software and hardware mechanisms central to the process protection in conventional processors and operating systems. Section 4 discusses the implications of reconfigurability on process protection and identifies the key requirement for safe process isolation in reconfigurable processors. In Section 5, we describe the Morph/AMRM system and a proposed protection architecture that meets these requirements set forth in Section 3. Section 6 discusses alternate approaches and the limitations on configurability imposed by the Morph/AMRM protection architecture. Sections 7 summarizes future work and the material covered in this paper.

## 2. Process Isolation: the Problem

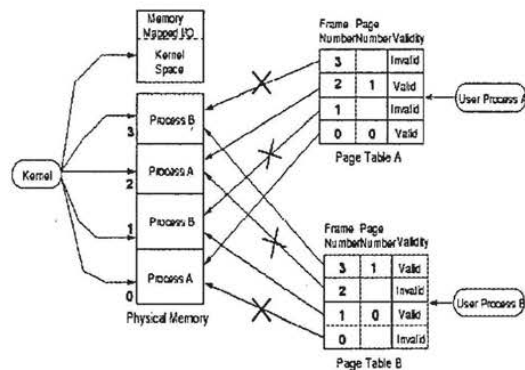


Figure 2: Multiprocess Protection based on Address Space Isolation

To understand the challenges of multiprocess isolation, it is instructive to first consider the possible modalities in which multiprocess isolation can be compromised. In the simplest mode, an application corrupts the data of another, causing it to fail or compute incorrectly. In a more complex mode, the application somehow locks up the machine, so no other application state is damaged, but neither can the machine make progress. One example of this would be jamming the memory bus or defeating the timer interrupt which ensures preemption. A more serious failure mode is to corrupt the operating system's data, which can lead to a machine crash in which all applications have data corruption. Finally, an application could also corrupt

input/output device state, confounding the operating system, the device (leading to data loss or misdirection), or application data itself. In all of these cases, the failure is the result of allowing an application action which can affect the machine hardware state, other application memory state, or operating system state.

The key issue in safe multiprocess execution is to control access to hardware resources, ensuring that these accesses are non-interfering. In general, access to main memory, as well as other architecturally visible state (processor data registers, control registers), system chip registers, and input/output device state must be controlled. Traditional approaches partition memory access, virtualize resources such as processor data resources with multitasking, and use operating system calls to mediate operations which require access to control registers, system chip sets, input/output device state, etc. Note that isolation and virtualization must apply to any resource at any level that a process can claim its ownership. The final piece of the puzzle is that in order to support the virtualization and multitasking, transitions between the different entities must be carefully controlled to prevent compromise.

## 3. Process Isolation in the MIPS R10000

The key issue in maintaining a safe multiprocessing environment is ensuring process isolation: the processor and the OS must prevent independent processes from interfering with the data and memory of each other and of the operating system kernel. They must also prevent a malicious process from taking over the processor and locking up the system.

Through a detailed analysis of the R10000 architecture and operating system, we identify the hardware mechanisms and OS software structures that are central to process isolation. We chose the MIPS R10000 processor as an exemplar of a modern RISC processor that supports a relatively simple UNIX style protection structure [9]. We first examine how a UNIX style operating system ensures process isolation and thereby derive the hardware requirements it imposes. Then identify the corresponding support in the R10000 processor. In the following discussion, we assume that the address translation is on a simple paging system. Most of today's systems actually employ multiple-paging or segmented paging but the address translation mechanism is fundamentally the same as a simple paging system.

### 3.1 Operating System-based Process Protection

#### 3.1.1 Application and Operating System Memory Isolation

Application and operating system memory isolation is achieved through controlled address translation. The physical memory of each process is isolated by having process's virtual address space pages map to its own physical memory frames only. To protect processes from modification by other processes, the memory-management hardware and the OS must prevent programs from changing their own address mappings. The UNIX kernel, for example, runs in a privileged mode (kernel mode or system mode) in which memory mapping may be controlled, whereas application processes run in an unprivileged mode (user mode). The page tables, mapping information for each process reside in the memory space of the kernel so that they can only be modified by the OS running in kernel mode. This address translation control to ensure isolation is achieved through the following mechanisms in UNIX [9, 10].

1. Locating correct translation information for each process. By using a special page table base register(*ptbr*) which is set from the process control block(*PCB*) on each process switch, the OS can correctly locate the page table for the executing process. Then the index portion of the virtual address is added to the address pointed to by the *ptbr* to locate the appropriate page table entry (*PTE*).
2. Distinguishing valid and invalid entries in page tables. Notice that the page table can contain entries that are not used by the process. These unused entries correspond to the pages that are not in the process's logical address space and thus compromise process isolation. The OS uses *valid-invalid* bits to distinguish these entries. Alternatively, the page table can also be implemented to contain only the entries that are actually used by the process. This implementation will require a special register containing the length of the process's page table, usually called *page-table length register(PTLR)*. *PTLR* can be used to check if page index portion of the virtual address is in the range and therefore is not accessing illegal translation information.
3. Controlling access types  
While the address translation to physical memory frames can be valid, the access to those physical memory frames are unlimited; the process can read, write, and execute them. It will be safer and more efficient if we can control the type of access to them. The protection bit field in the *PTE* provides this access control information. At the same time that the physical address is being computed, the protection bits can be checked to verify that no accesses not granted are being made. These bits usually indicate whether the process can read/write, read-only, or execute-only. The type and the number of the protection bits provided are dependent on the underlying processor.

#### 4. Managing TLB consistency.

The translation information, namely the *PTE*, is cached in the processor's TLB to avoid extra memory access to the page table. Using special privileged instructions, OS updates the TLB with consistent mapping information when a miss occurs. But notice that after a context switch, although the new page table is pointed to by the new process's *ptbr*, the TLB would contain entries that are left over from the previous processes. Therefore, to ensure process isolation, we need to invalidate or distinguish the entries in the TLB that does not belong to the executing process. This can be done by allowing the OS to flush the TLB by a special privileged instruction after a context switch or by tagging the TLB entries with the process ID's and valid-invalid bits.

### 3.1.2 Resource Protection through Operating System Mediation

Not only the memory but also all resources that can be shared by processes must be isolated and virtualized. The OS provides protected resource access through mediation. The most fundamental role of the operating system is to mediate process's accesses to system resources. Processes are provided with a system call interface to the operating system kernel, and all accesses to the resources must go through the system calls to the kernel hence protecting the resources from illegal accesses of processes. The operating system can enforce this through the following features of the OS and the hardware:

1. System trap instruction and system call handler:  
System call invocation is made through special trap instruction that changes the mode to kernel mode and jumps to system call handler location predefined by the OS. This system call handler is responsible for all system call processing in kernel, such as saving/restoring process context, selecting appropriate kernel function through system call dispatch vector, transferring control back to user process in user mode. The system call handler is one of the most fundamental routines in the kernel and is written very carefully to ensure safety.
2. Interrupt architecture:  
The interrupt architecture in the processor and the OS guarantees correct invocation and handling of interrupts and provides priority management mechanisms. The interrupt handler is one of the most fundamental and carefully-written kernel routines and is responsible for safe mode transition, context saving/restoring, and priority based servicing.

**For general I/O resources, the following features of the OS and the hardware ensures protection.**

3. Privileged I/O instructions:

I/O address space is separate from main memory space(e.g. x86 processors) and can only be accessed through privilege mode instructions(e.g. inb, outb in x86)

4. Memory mapped I/O in protected memory space:

I/O accesses are made by memory access instructions to main memory space, but this space can only be accessed by kernel.

5. I/O buffers in protected memory space:

Buffers for I/O operations reside in kernel space or space private to each process and thus protected from other processes.

**For CPU resources,**

6. Preemptive time-quota based scheduling:

A process must relinquish the CPU when its time-quota expires. The scheduler is designed carefully to avoid starvation of low-priority processes. Timer interrupt has a very high priority, second only to power-failure interrupt.

**3.1.3 Machine State Virtualization and Safe Transitions (context switch)**

Multiprocess isolation in a computer system can be considered as providing to each process a private and isolated virtual machine. The OS captures the state of each virtual machine provided and ensures safe transitions between virtual machine states (i.e. safe context switching). In UNIX, the virtual machine state is captured in the *Process Control Block(PCB)*. It contains a snapshot of general-purpose registers, memory context, and special registers, etc. Context switching involves a series of low-level privileged instructions to switching these states and performing many hardware-specific tasks in order to ensure safe transition to a new virtual hardware state. These hardware-specific tasks include flushing the data, instruction, address translation(TLB) caches, and flushing the execution pipeline.

The OS process isolation mechanisms that are identified in this section can be distilled into two key elements in the hardware which enable process isolation:

1. Processor execution modes and kernel address space
2. Control of address translation and TLB management

**3.2 Hardware Support for Process Protection**

The two key elements in the hardware to support process protection can be further classified into a range of features that must be provided by the hardware to enable process protection mechanisms dictated by the OS:

**Execution modes and kernel address space:** The processor should at least provide two modes of execution, i.e. privileged execution mode and user mode, so that the

kernel data structure, special registers, and processor control bits can only be access and altered through special privileged instructions. The virtual address space should contain a kernel address space that can only be accessed in privileged execution mode. This is where the kernel data structure resides.

**Context register:** This register identifies the current process and is used to select appropriate page tables for controlling memory access.

**Valid-Invalid bit, PTLR:** The processor should be able to recognize the valid-invalid bit for PTEs which identifies those that do not map to a valid physical address. Optionally, process can have a Page Table Length Register to set the bound on the page table.

**Protection bits:** The process should be able to recognize a protection bit or some set of them to allow a finer level of access control on the pages.

**TLB tagging or flush mechanism:** The processor should enable the OS to distinguish the TLB entries that belong to the executing process by having TLB entries tagged with process ID's or allow the OS to flush out the TLB by supplying a special TLB-flush instruction.

In this section, we discuss how these features are implemented in the R10000 processor, a typical superscalar RISC microprocessor from the MIPS RISC family.

**3.2.1 The System Control Processor (CP0)**

The central part of R10000's protection architecture is the CP0 processor. The CP0 controls execution mode switch, TLB management and control, exception catching and dispatching, cache control, and the TLB where the protection checking is carried out. CP0's states and registers can only be altered by CP0 instructions, which are privileged MOVE instructions(*MFC0*, *MTC0*, etc). These instructions can only be executed in kernel mode. Thus the system is protected from non-privileged processes which attempt to alter the CP0 processor state including the processor operation mode

**3.2.2 Processor Modes**

Processor operating modes for the R10000 include Kernel mode, Supervisor mode, and User mode. The current mode is indicated in the CP0 registers, and that mode can only be changed in two ways:

1. CP0 status register's *KSU* field is changed explicitly by CP0 MOVE instructions executed in kernel or supervisor mode.
2. The processor is handling an error (*ERL* bit in CP0 is set) or an exception(*EXL* bit in CP0 is set) and is forced into kernel mode. This mechanism is used to implement guarded transitions such as those used by system calls.

The current operating mode also determines access to the kernel address space (or a respective application address space) as described in the next section.

When the system starts up, the system is in Error level (*ERL* bit is set) so the processor is in the kernel mode. The cold-reset exception handler boot straps the operating system which then runs in kernel mode.

### 3.2.3 Kernel Address Space

To enable the operating system kernel to mediate access to all hardware resources as well as interprocess communication, it must have access to all memory. While in kernel mode, the processor is allowed to access all kernel segments (*kseg0*, *kseg1*, *kseg3*) and user segments, allowing access to all of the system resources.

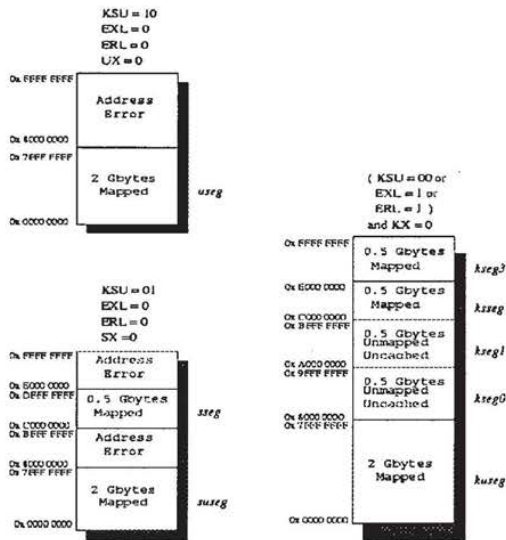


Figure 3. Kernel and User Address Spaces in the R10000

While in user mode, the processor can only access a subset of the memory space as determined by the address mappings for that application process. This is typically a subset of the address space, and is illustrated in Figure 3. The accessible address space for a user process is determined by the TLB entries whose address space identifier (*ASID*) tags match the ID of the process. If an application process attempts to reference an address not mapped by its TLB entry or attempts to reference an address in kernel address space, an Address Error Exception will occur. As with all exceptions, this is handled by an operating system installed exception handler and generally results in a fatal signal for the application process and its termination, thus protecting the system data and other processes' data from unauthorized application access.

### 3.2.4 Control of Address Translation and TLB Management

The control of address translation, namely checking validity and access type control, is supported in R10000 by moving the PTE to CP0 registers and then performing corresponding checks and raising appropriate exceptions in the CP0 control processor core. The *EntryHi* and *EntryLo* CP0 registers are always loaded with the TLB entry or the page table entry (if the TLB misses) that corresponds to the virtual address. The address translation, as well as the required checking and exception raising, is done using the contents of these registers. The *EntryHi* and *EntryLo* CP0 registers are loaded only through *TLBP*, *TLBWI*, *TLBWR* CP0 instructions (in case of TLB hit) or generic *move to CP0* instructions (in case of TLB miss) so that these registers cannot be altered by user processes.

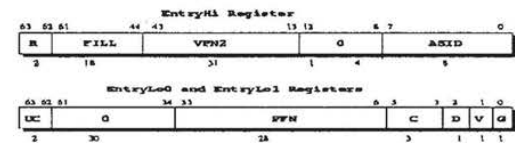


Figure 4. The Address Translation Control registers in the R10000.

As shown in Figures 4, the TLB entry and the corresponding *EntryHi* and *EntryLo* CP0 registers have validity bits *V* and *D* that are recognized by the CP0 and used as the invalid-valid bit and the protection bit that were described in the section on the OS protection scheme.

The CP0 has *Context* and *Xcontext* registers which point to the base of the page tables so that the page table for the executing process is located safely and quickly after a context switch.

The R10000 provides to the OS the means to manage and maintain consistent TLB entries. The TLB entries are not flushed after every context switch in R10000. Instead, R10000 allows the TLB entries loaded for different processes to be distinguished. Support for this can be found in the 8-bit *ASID* (Address Space ID) field in the TLB entry. *ASID* is a unique id that is assigned to each process. After a context switch, a new value is loaded into the *ASID* field of the *EntryHi* register. Only the TLB entries whose *ASID* tag matches this ID or set to global are enabled. In this way, it is guaranteed that only the pages that belong to the executing process or the pages that are globally shared are translated and accessed. The TLB entries are written with the contents of the *EntryHi* and *EntryLo* CP0 registers only through *TLBP*, *TLBWI*, *TLBWR* CP0 instructions so that the user processes cannot alter the TLB directly.

The L1 caches in R10000 are virtually indexed and physically tagged. It is virtually indexed in order to reduce access time to the cache by allowing finding set/reading tag and address translation for tag to occur concurrently. It is

important to note that because it is still physically tagged, accesses to the cache cannot bypass the TLB, where most of memory protection scheme is implemented, even though it is virtually indexed

#### 4. Process Isolation in Reconfigurable Hardware

Because multiprocess decomposition is a critical element of modularity and fault isolation in software systems, providing a safe multiprocess execution is a critical requirement for reconfigurable processors to achieve widespread use. We have described the basic multiprocess protection problem in Section 2, and outlined the possible failure modes. In reconfigurable systems, these failure modes are largely the same, but can occur via the actions of both the software application program and the application-adapted configurable hardware. As we will see, a key aspect of a protection architecture for reconfigurable systems is to restrict the capabilities of the configurable hardware for unchecked access to architecturally visible and invisible system state.

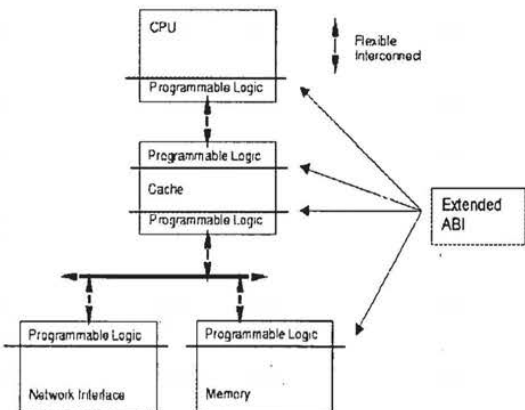


Figure 5: The canonical application-adaptive reconfigurable architecture, where elements of reconfigurable logic can in general be attached to all elements of interconnect, logic, and memory in the system.

#### 4.1 Reconfigurable Architecture Model

We characterize reconfigurable processors as a new class of processors with a fraction of the silicon area dedicated to reconfigurable logic blocks on which application-customized mechanisms or computations can be built. This basic architecture is characterized in Figure 5. In this basic architectural framework, reconfigurable elements can be attached to all elements of interconnect, logic, and memory, enabling any conceivable augmentation

of the hardwired system. This is the most general model, and is the starting point for our analysis of process isolation. As examples of the type of configurability that can be achieved, major functional blocks in these processors can also be reconnected, replaced, or have their communication mediated. Elements of state can be altered, arbitration protocols can be changed, finite-state machines can be replaced and interconnect resources can be added (or diverted) to speed (or slow) particular data movement operations. All of these changes can be integrated into the functional operation of the processor (e.g. change the meaning of an instruction) as well as its protection structure (e.g. allow a non-privileged instruction to change a CP0 register or a range of TLB entries). In summary, in the most general case, the configurable hardware can be attached to any part of the entire system, its actions can affect any part of the hardware system.

#### 4.2 Implications of Reconfigurability on Process Protection

For safe protected execution in reconfigurable machines, we need the guarantee of process isolation that the rigid process isolation mechanisms provide while allowing a certain degree of freedom in reconfigurability of the hardware. Reconfigurability adds to the conventional concerns of controlling the software <-> software interactions of processes that share the processor, resulting in the following range of concerns:

1. Software <-> software interactions
2. Software <-> configurable hardware interactions
3. Configurable hardware <-> configurable hardware interactions

These cases are illustrated in Figure 6. The first case corresponds to the traditional process protection problem. In the second case, as the processor is context switched amongst the application processes, the surrounding configurable hardware may or may not be switched synchronously. In fact in some cases, it may be clearly advantageous for the configurable hardware to continue execution while the corresponding application process is context switched out. Because the configurable hardware is properly viewed as an extension of the application process' "virtual machine", care must be taken to ensure that inappropriate interactions do not occur. For example, one such reconfiguration might involve permuting data in the memory between phases of execution in an application program. While it might be advantageous to allow this permutation to go on while the application is not scheduled on the processor, process isolation dictates that the customization of the memory controller must not affect the functional behavior of the system for other processes (e.g. other applications or even the kernel). Finally, the third

case involves interactions of the configurable hardware with shared system (hardwired) resources which cause either compromises of data or more basic aspects of the system. For example, if application #1 reconfigures the addressing interface from the processor to the memory bus, and application #2 customizes the addressing interface of the memory controller, allowing direct interaction could cause inappropriate data access or corruption. In short, the reconfigured logic as well as the processes must be safely isolated to achieve a robust and extensible system.

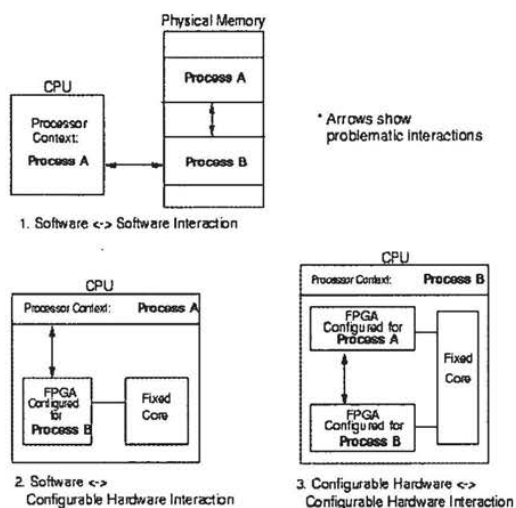


Figure 6. Three types of interactions can cause protection compromises in application-adaptive configurable machines. Arrows show problematic interactions.

Requirements for process isolation in a configurable architecture extend the hardwired system requirements outlined in Section 3, requiring coordination across software and configurable hardware, and controlled access in all parts of the system that configurability is allowed.

Reconfigurable architectures with process protection guarantees as well as existing reconfigurable architectures can be classified according to the customizability and the safety guarantee they provide. One possible range of configurable architecture classes spans a range of flexibility and safety as below:

1. **Full Configurability:** All processor components fully reconfigurable, all memory accesses checked and translated by a hardwired TLB which enforces OS mappings. All other elements of system configurable.
2. **Aggressive Configurability with safety:** All processor components excepting privilege mode changes and privileged operations fully reconfigurable, all memory accesses checked and translated by a

- hardwired TLB which enforces OS mappings. All other elements of system configurable, but accesses to registers, shared resources such as busses, and memories all controlled via hardwired access checking.
3. **Moderate Configurability with safety:** All processor components excepting privilege mode changes and privileged operations fully reconfigurable, all memory accesses checked and translated by a hardwired TLB which enforces OS mappings. Other devices which generate addresses are configurable, and have accesses checked by a shared (or multiple) TLB's. Configuration of accesses to registers, shared resources such as busses, and memories not allowed.
4. **Traditional Coprocessor Configurability:** Only coprocessor devices are configurable and their accesses to shared resources are unchecked (these could be checked by a hardwired TLB at the I/O interface). No address translation or shuffling in the MMU. This approach is typically taken for FPGA-based coprocessor configurable designs.
5. **Processor Configurability with safety:** All processor components excepting privilege mode changes and privileged operations fully reconfigurable, all memory accesses checked and translated by a hardwired TLB which enforces OS mappings. Other parts of the system are not configurable.

These architectures vary widely in their capabilities for customization to enhance application performance and the cost of providing multiprocess isolation guarantees. Because the issues are complex, and a detailed analysis of even one of these architectures is a topic for an entire technical paper, we merely point out that #1 allows the greatest flexibility, but cannot ensure that any isolation is guaranteed.

Architecture types #2 and #3 allow what one might consider to be a broad notion of useful configurability, leaving only the protection core, TLB checks, and a few key arbitration resources fixed. By maintaining minimal structures and mechanisms that have been identified as essential in satisfying process protection requirements, we believe that process protection can be guaranteed while allowing a certain degree of freedom in reconfigurability of the hardware. Within the scope of types #2 and #3 alone, there is a wide range of architectural space to explore.

Architecture type #4 is the traditional configurable coprocessor protection model where the configurable hardware is viewed as an extension of the system hardware, and dealt with by the operating system as an input/output device. This is dangerous, as the configurable logic can easily compromise system integrity. At a minimum, address checking (and interrupt capability) should be controlled.

Finally, architecture type #5 is the complement of #4, providing processor side configurability but no coprocessor



configurability. This type allows customization of data movement and computation around the primary locus of computation, and the tight coupling this makes customization significantly more powerful than in coprocessor systems. In type #5, process isolation is easily maintained by a hardwired TLB and checking all processor references.

## 5. Process Isolation in the Morph/AMRM Architecture

In the Morph/AMRM reconfigurable processor [4, 5], we propose that reconfigurable logic can be integrated into various components of the processor core to allow per applications adoption of optimal policies and/or custom mechanisms for data movement, memory hierarchy management, value prediction, branch prediction, etc. Rather than a more traditional approach of having a reconfigurable functional unit for custom computations, we are attaching reconfigurable logic to every component of the processor that is needed in optimizing various performance critical mechanisms and policies. This enables a highly flexible architecture but also makes virtually every part of the processor have reconfigurability.

The design of the Morph/AMRM protection architecture follows the protection model described in Section 3 by depending on memory addressing control and a privilege mode structure for ensuring that control and providing a virtual machine and system services for each process. However, because Morph/AMRM incorporates configurable logic deep internal to the processor core, careful engineering of exactly what must be hardwired is required. The Morph/AMRM architecture enables configurability in the processor complex with safe multiprocess protection. We also believe it enables much of the useful configurability in the processor complex, notably policies for improving efficient management of resources and even the addition of instructions, special functional units, or even processor states. The model provided to application programs is a private, configurable, virtual machine which enables rich application customization. These applications (and their customizations) are cleanly isolated.

The Morph/AMRM protection architecture is a type #3 configurable architecture (Moderate Configurability with Safety), preserving strong process isolation guarantees. The key idea is to have a few parts of the system be hardwired (unchangeable) and to also limit the connectivity to other resources (ensuring mediated access to those resources). Together, these two approaches ensure that key processor resources can be protected and recovered.

The basic model uses a context switching mechanism which synchronously switches processor state and all of the process' configurable hardware throughout the system simultaneously. Thus, Morph/AMRM eliminates concerns

of software<->configurable hardware and configurable hardware<->configurable hardware interactions for unrelated programs. This leaves the main issues of ensuring secure context switching and strict address isolation.

The first two hardware features ensure virtualized execution and secure process switching. The latter five mechanisms enable process isolation.

### 1. Hardwired CP0 core:

The control processor is central in providing mechanisms such as privileged/user mode transitions and exception and interrupt delivering and handling for OS mediation, which are required to guarantee process isolation. The control processor core cannot be reconfigured in our design.

### 2. Hardwired instruction pipeline:

Controls and the structure of the execution pipeline are fixed for instruction sequencing. However, Customizable functional unit provided for custom instruction.

### 3. Hardwired CP0 to the TLB control for address translation and TLB entry management:

As pointed out in section 3, controlling/checking address translation in the TLB and TLB entry management is another central hardware requirement for process isolation. We have hardwired the TLB and the control from CP0 to TLB to guarantee correct isolation.

### 4. The remainder of the datapath, processor, and caches can all be configurable and connected in arbitrary ways for maximum flexibility.

### 5. TLB controlled accesses for other configurable elements accessing system bus:

Components such as system chip sets, I/O controllers, memory controllers that access the system bus(for memory or other memory-mapped items) can be fully configurable as long as they generate virtual addresses which is then checked by hardwired local TLBs.

### 6. Hardwired arbiters for controlling accesses to key shared resources:

Access to key shared interconnects such as the system bus are controlled by hardwired arbiters which are not changed, system reserves highest priority to allow preemption for these resources, configurable hardware can be redundant interconnects to these to accelerate, but cannot compromise the arbitration of these key resources.

### 7. Context switching and multiplexing/bypassing reconfigurable blocks to isolate and virtualize reconfigured logic blocks for different processes.

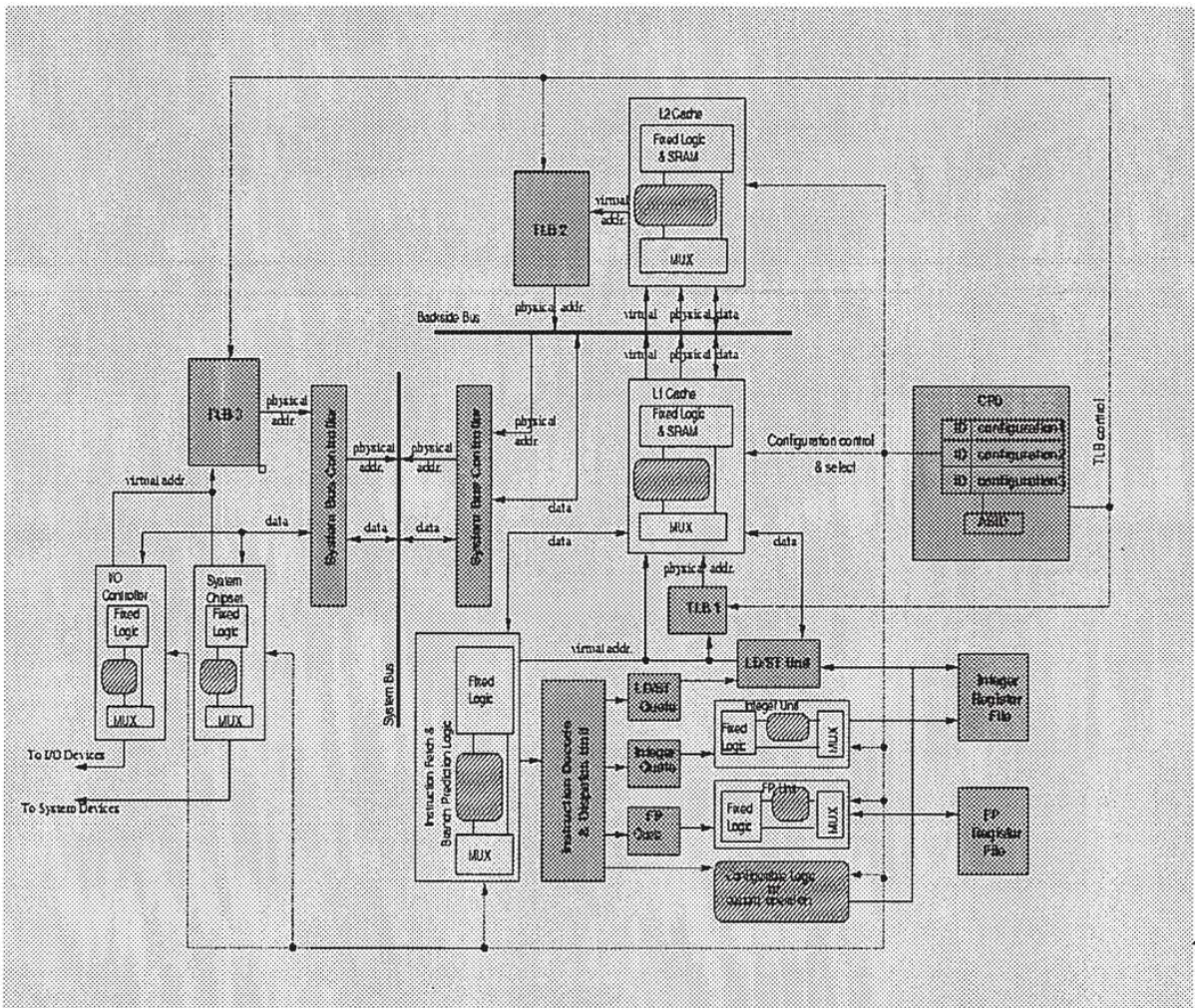


Figure 7. Morph/AMRM Protection Architecture. Rounded stripe box denotes reconfigurable logic block. Shaded box denotes fixed logic.

This is intentionally a simple model that provides most of the power of configurability and incurs a rather significant overhead of process isolation. The model provided to application programs is a private, configurable, virtual machine which enables rich application customization. These applications (and their customizations) are cleanly isolated. The schematic diagram of the protection architecture with the considerations listed above is given in Figure 7.

As explained earlier, these key features of the protection architecture are realized by restricting configurability of the components that are identified (in the previous sections) to be critical in maintaining classical process protection guarantees. The diagram in figure 7 shows which components are configurable or which are not (shaded boxes in the diagram). System chip set, I/O controller,

memory controllers also generate virtual addresses and have them checked and translated by a shared or local hardwired TLB, which also has hardwired control from CP0.

In addition, to eliminate newly introduced concerns of software->configurable hardware and configurable hardware->configurable hardware interactions, our basic model incorporates mechanisms to synchronously switch processor state and all of the process' configurable hardware throughout the system simultaneously. The outline of this mechanism, namely the synchronized hardware context switching, is briefly explained below:

**Configuration owner table and configuration context register in CP0:** The reconfigurable logic blocks are isolated, with multiplexers to control the inputs and outputs

to each block. Controls to these multiplexers come from CP0, which maintains a table of the owner processes of each reconfigured block. The configuration context register in CP0 indicates which reconfigurable blocks are to be used for the current process (see fig. 8). Notice that there are entries for two banks in each entry of the configuration owner table and that each reconfigurable logic block is divided into 2 banks. This can allow two different configurations of that block for two different processes to be switched without having to swap in the new configuration at each context switch.

**Configuration selection/bypassing:** If the entries in reconfigured block owner table for these blocks match the current process ID (different from process ID in OS. ASID used in TLB can be used here.), corresponding reconfigured block will be selected and activated while other reconfigurable blocks not used by the process will be bypassed. A more radical approach could even "suspend" the clock for this logic/memory to completely disable when the owner process is inactive in order to ensure that no interference from unscheduled process' reconfigured logic.

**Configuration swapping:** If a block is to be used by the current process but does not match any of the two process ID fields (i.e. not configured for this process), an exception is raised and new configuration is swapped in.

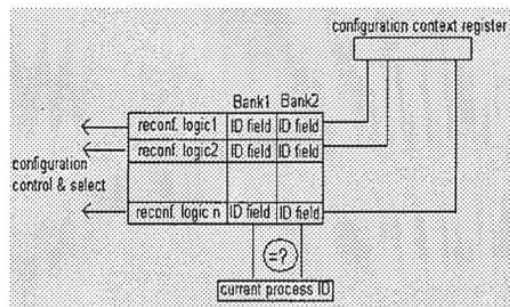


figure 8. Configuration owner table and configuration context register

## 6. Discussion and Related Work

The last decade has seen a proliferation of reconfigurable computing machines based on programmable logic blocks. In this section, we present some of these efforts and discuss the multiprocess protection issues in these alternate approaches in comparison with the Morph/AMRM architecture. We also discuss the limitations of our current Morph/AMRM protection architecture proposal.

FPGA processors, or processors built entirely out of FPGAs, account for the majority of reconfigurable

computing machines that have been proposed [11, 12, 13]. Like ASIC hardware, these processors perform special computations that are specific to the task that systems with these processors are to carry out. The difference between ASIC design and FPGA processors is that FPGA processors can have a few contexts so that these processors can carry out different operations in different stages of the task. Also, unlike the rigid ASIC designs, they can also re-tune themselves for better performance in response to the data that they are computing. Since most of these could not work as a stand-alone processor or only implemented as an experimental testbed, it is inappropriate to discuss multiprocess protection issues for these processors. There is no clear model for managing memory or external devices for these processors, which suggests that it will be difficult to, if not impossible, to design a stand-alone FPGA processor supporting safe multiprocess environment. Therefore, their use is usually limited to specific process engine used in domain-specific embedded systems.

While impressive performances have been reported for FPGA-based processors [11, 12, 13], these machines also have other shortcomings such as no instruction sequencing, long configuration time due to limit I/O, and slower implementation of standard functions. To overcome these shortcomings that make them less than ideal for general-purpose computing, architectures that couple a general-purpose control processor with FPGA co-processors have been proposed [14]. FPGA is placed as a slave computational unit on the same die as the processor and is used to speed up what it can, while the main processor controls the whole execution and takes care of other computations. Only some regular portions in the program such as loops and subroutines that can be programmed in reconfigurable logic to obtain speed-up are carried out in the reconfigurable part. This falls in the architecture type #4 as classified in section 4. The FPGA co-processor has its own memory interface and control logic, so it compromises multiprocess safety unless the system is extended so that the access is controlled by hardwired (local) TLB, which in turn is controlled through a hardwired control processor core. In Morph/AMRM, other configurable devices which generate addresses (e.g. system chip sets, input/output devices, memory controllers) must generate virtual address and is checked by local TLBs with fixed control from the control processor. Maintaining cache coherency is another problem to be solved for reconfigurable architectures of this kind.

Reconfigurable processors with dynamic instruction sets [15] try to extend the application-specific computation capability of a general processor with a computational unit implemented in reconfigurable logic. Again, these efforts have not explicitly addressed multiprocess protection. The overall architecture of the processor and the instruction execution cycle is similar to a general purpose processor but they have an extensible instruction set that can carry out

custom instructions as needed. As with other proposed works, this architecture is yet implemented only as an experimental testbed to demonstrate potential performance gain and thus lacks details in mechanisms to support real multiprocess environment. The existence of global controller in charge of interface to memory, registers, and processor status suggest that this architecture could be extended to provides protection guarantee of type #5 architecture. But the configurability is simply limited to providing configurable functional unit(implementing custom instructions) whereas in Morph/AMRM, configurability extends to other processor and system components to improve utilization of performance critical resources while providing each process with a private, configurable virtual machine by ensuring isolation and lock-up freedom. The reconfigurable logic blocks in the core enables customization of hardware granularity, memory system management, and bindings between resources, which is driven by the application.

The proposed Morph/AMRM protection architecture provides isolated, customizable virtual machine to each process, and pays a price in limiting configurability. While new instructions can be added, the parts of the instruction decoder and control that access the privilege control parts of the system must be hardwired. This still allows execution pipeline configurability and a wide range of optimizations, should they be performance sensible. The Morph/AMRM architecture also requires that all memory accesses be checked by hardwired TLBs and that there be no other address translation or shuffling beyond that. This restriction precludes adaptations that dynamically remap memory addresses at the translation level to implement scatter/gather technique and to increase the reach of TLBs [16]. However, such adaptations are not inherently safe, and depend on the values put into the translation tables. As such, they cannot be proven correct as a system attribute, but must depend on software to enforce some restrictions on use to ensure correctness and multiprocess isolation. Our Morph/AMRM architecture can be extended to include such a notion.

We have not completed the actual design and implementation of a prototype processor yet, but we are aware of the possibility that adding extra switches and muxes to isolate and context-switch customized logics that are spread about in the processor may incur considerable overhead in terms of clock rate, silicon area, and design complexity.

## 7. Summary and Future Work

Introducing application-controlled reconfigurability in the processor raises significant challenges in ensuring process isolation and protection (i.e. multiprocess isolations), a critical element of robust computing systems. In this paper, we have analyzed the implications of

hardware reconfigurability on a multi-process environment and proposed architectural requirements for safe and protected execution for reconfigurable processors classified according to the protection guarantees and the level of reconfigurability they provide. Our study began by examining the protection structures of traditional processors and operating systems, identifying the key mechanisms of this protection architecture that is based on process isolation via address space isolation and mediation. This served as the starting point of our analysis and design of the protection architecture for reconfigurable processors.

Based on observations made through the analysis and classification, we have presented an architectural design incorporating a protection architecture that is best suited for our MORPH/AMRM reconfigurable processor. The key elements of this protection architecture were: 1. hardwired control processor for privilege mode transitions and instruction sequencing , 2. hardwired control to TLB for address translation and TLB entry management, 3. All other reconfigurable elements that generate addresses must deal in virtual address, and their accesses checked by local/shared TLBs, 4. Controlled access to key shared interconnects are controlled by hardwired arbiters. With these features, the OS and hardware mechanisms required for process process protection are well preserved and protected accesses are strictly checked and controlled by these mechanisms, while allowing all other components to be reconfigurable for better flexibility. This architectural design will thus provide most of the power of configurability and at the same time provide to each process a private, configurable, virtual machine which enables rich per-application basis adaptation. We plan to carry out simulations to verify our design and refine it in parallel with the development of our prototype evaluation board of the MORPH/AMRM processor. The simulations that we are planning on will be capable of revealing realistic OS – processor interaction, and is likely to be based on SimOS[17].

Making reconfigurable processors multiprocess-safe isn't the only requirement for a robust reconfigurable system, however. Dynamically validating and correcting the reconfigured logic is needed to find hardware faults and possibly to contain them. In the near future, we will study the issues concerning online validation/hardware fault containment and give a complete solution to building a reliable and robust reconfigurable system.

### Acknowledgements

We would like to express our gratitude to Prof. Rajesh Gupta, Prof. Alex Nicolau, Prof. Alexander Veidenbaum, Louis Giannini, Ali Dasdan, Ben Zhang, and Martin Schulz for their comments and contributions. The Morph/AMRM project is supported by DARPA/ITO under contract number DABT63-98-C-0045 and by NSF Award number ASC-96-34947.

## References

- [1] Semiconductor Industry Association. National Technology Roadmap for Semiconductors(NTRS), 1997.
- [2] Satapathy, R., Gupta, R. Analysis of Technology Trends: Making a Case for Architectural Adaptation in Custom Data-paths, 1997.
- [3] A. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, X. Ji, Adapting Cache Line Size to Application Behavior. To appear in *Proceedings of the 13<sup>th</sup> ACM International Conference on Supercomputing, 1999(ICS '99)*.
- [4] Chien, A. and Gupta, R. MORPH: A system Architecture for Robust High Performance Using Customization. In *Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation (Frontiers '96)*(Oct. 1996), pp. 336-345.
- [5] Zhang, X., Dasdan, A., Schulz, M., Gupta, R., and Chien, A. Architectural Adaptation for Application-Specific Locality Optimization. In *Proceedings of the International Conference on Computer Design* (Oct. 1997)
- [6] DeHon, A. *Reconfigurable Architectures for General-Purpose Computing*, Ph.D. thesis, Massachusetts Institute of Technology, 1996
- [7] Kiczles, G., et Al. Open Implementation Design Guidelines. In *Proceedings of the 19<sup>th</sup> International Conference on Software Engineering*, 1997.
- [8] MIPS technologies, Inc. *MIPS R10000 Microprocessor User's Manual*, 1996.
- [9] Bach, M., *The Design of the UNIX Operating System*, Prentice-Hall, Englewood Cliffs, NJ, 1986.
- [10] Leffler, S., McKusick, M., Karels, M., Quarterman, J. *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley, Reading, MA, 1989.
- [11] Gokhale, M., Holmes, W., Kospers, A., Lucas, S., Minnich, R., Sweely, D., and Lopresti, D. Building and Using a Highly Programmable Logic Array. *IEEE Computer*, 24(1):pp. 81-89, Jan. 1991.
- [12] Arnold, J., Buell, D., and Davis, E., Splash 2. In *Proceedings of the 4<sup>th</sup> Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 316-324, June 1992.
- [13] Vuillemin, J., Bertin, P., Roncin, D., Shand, M., Touati, H., and Boucard, P. Programmable Active Memories: Reconfigurable Systems Come of Age. *IEEE Transactions on VLSI Systems*, 4(1):pp.56-69, Mar. 1996.
- [14] Hauser, J. and Wawrzynek, J. Garp: A MIPS Processor with a Reconfigurable Coprocessor. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, 1997.
- [15] Wirthlin, J. and Hutchings, B. DISC: The dynamic instruction set computer. In *Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing*, John Schewel, Editor, Proc. SPIE 2607, pp. 92-103 (1995).
- [16] J. Carter, W. Hsieh, L. Stoller, M. Swanson, L. Zhang, E. Brunvand, A. Davis, Chen-Chi Kuo, R. Kuramkote, M. Parker, L. Schaelicke, and T. Tateyama. Impulse: Building a Smarter Memory Controller. To appear in the *Proceedings of IEEE Fifth International Symposium on High Performance Computer Architecture (HPCA-5)*
- [17] Mendel Rosenblum, Stephen A. Herrod, Emmett Witchel, and Anoop Gupta. Complete Computer Simulation: The SimOS Approach. In *IEEE Parallel and Distributed Technology*, Fall 1995.

**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

**BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- BLACK BORDERS
- IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT OR DRAWING
- BLURRED OR ILLEGIBLE TEXT OR DRAWING
- SKEWED/SLANTED IMAGES
- COLOR OR BLACK AND WHITE PHOTOGRAPHS
- GRAY SCALE DOCUMENTS
- LINES OR MARKS ON ORIGINAL DOCUMENT
- REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- OTHER: \_\_\_\_\_

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.**

**Index of Claims**



Application No.

10/869,200

Examiner

Shane M Thomas

Applicant(s)

POZNANOVIC ET AL.

Art Unit

2186

√	Rejected
≡	Allowed

-	(Through numeral) Cancelled
+	Restricted

N	Non-Elected
I	Interference

A	Appeal
O	Objected

Claim		Date	
Final	Original		
		1/10/05	
1	✓		
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24	✓		
25			
26			
27			
28			
29			
30			
31			
32			
33			
34			
35			
36			
37			
38			
39			
40			
41			
42			
43			
44			
45			
46			
47			
48			
49			
50			

Claim		Date	
Final	Original		
51			
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			
69			
70			
71			
72			
73			
74			
75			
76			
77			
78			
79			
80			
81			
82			
83			
84			
85			
86			
87			
88			
89			
90			
91			
92			
93			
94			
95			
96			
97			
98			
99			
100			

Claim		Date	
Final	Original		
101			
102			
103			
104			
105			
106			
107			
108			
109			
110			
111			
112			
113			
114			
115			
116			
117			
118			
119			
120			
121			
122			
123			
124			
125			
126			
127			
128			
129			
130			
131			
132			
133			
134			
135			
136			
137			
138			
139			
140			
141			
142			
143			
144			
145			
146			
147			
148			
149			
150			





4-12-05

IFW



Client Matter No. 80404.0033.001  
Express Mail No.: EV330612115US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Serial No. 10/869,200	Confirmation No.: 5929
Application of: POZNANOVIC	Customer No.: <b>25235</b>
Filed: June 16, 2004	
Art Unit: 2186	
Examiner: THOMAS, Shane M	
Attorney Docket No. SRC028	
For: SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY BANDWIDTH IN RECONFIGURABLE HARDWARE	

AMENDMENT AND RESPONSE PURSUANT TO OFFICE ACTION  
DATED JANUARY 14, 2005

MAIL STOP AMENDMENT  
Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

Sir:

In response to the office communication mailed January 14, 2005 please amend the above-identified application as follows:

**Amendments to the Claims** are reflected in the listing of claims which begins on page 3 of this paper.

**Amendments to the Drawings** begin on page 7 of this paper and include both an attached replacement sheet and an annotated sheet showing changes.

**Remarks/Arguments** begin on page 8 of this paper.

\\BDO - 80404/0033 - 175820 v1

Appl. No: 10/869,200  
Amdt. Dated April 11, 2005  
Reply to Office action of January 14, 2005

An **Appendix** including 1 sheet of amended drawing figures is attached following page 8 of this paper.

**A. Amendments to the Claims:**

This listing of claims will replace all prior versions and listings of claims in the application:

**Listing of Claims:**

1. (Currently Amended) A reconfigurable processor comprising:  
a first memory having a first characteristic memory bandwidth and/or memory utilization type; and  
a data prefetch unit coupled to the first memory, wherein the data prefetch unit retrieves data from a second memory of second characteristic memory bandwidth and/or memory utilization and place the retrieved data in the first memory type and wherein at least the first the memory types and data prefetch unit are configured by a program.
2. (Currently Amended) The reconfigurable processor of claim 1, wherein the reconfigurable processor does not have a cache to store data from the first memory.
3. (Currently Amended) The reconfigurable processor of claim 1, wherein the second memory has a characteristic line size and the data retrieved from the second memory is not a cache line-sized unit of contiguous data.
4. (Currently Amended) The reconfigurable processor of claim 1, wherein the data prefetch unit is coupled to a memory controller that controls the transfer of the data between the second memory and the data prefetch unit.
5. (Currently Amended) The reconfigurable processor of claim 1, wherein the data prefetch unit receives processed data from on-processor memory and writes the processed data to an external off-processor ~~memory~~ memory.

6. (Original) The reconfigurable processor of claim 1, wherein the data prefetch unit comprises at least one register from the reconfigurable processor.

7. (Original) The reconfigurable processor of claim 1, wherein the data prefetch unit is disassembled when another program is executed on the reconfigurable processor.

8. (Currently Amended) The reconfigurable processor of claim 1 wherein said second memory comprises a processor memory and said data prefetch unit is operative to retrieve data from [[a]] the processor memory.

9. (Original) The reconfigurable processor of claim 8 wherein said processor memory is a microprocessor memory.

10. (Original) The reconfigurable processor of claim 8 wherein said processor memory is a reconfigurable processor memory.

11. (Currently Amended) A reconfigurable hardware system, comprising:

a common memory; and

one or more reconfigurable processors coupled to the common memory, wherein at least one of the reconfigurable processors includes a data prefetch unit to read and write data between the data prefetch unit and the common memory, and wherein the data prefetch unit is configured by a program executed on the system.

12. (Original) The reconfigurable hardware system of claim 11, comprising a memory controller coupled to the common memory and the data prefetch unit.

13. (Currently Amended) The reconfigurable hardware system of claim 11, wherein the one or more reconfigurable processors are [[is]] not coupled to a cache.

14. (Currently Amended) The reconfigurable hardware system of claim 11, wherein the common memory has a characteristic line size and the data written and read between the data prefetch unit and the common memory is not a cache line-sized unit of contiguous data.

15. (Currently Amended) The reconfigurable hardware system of claim 11, wherein the at least one of the reconfigurable processors also includes a computational unit coupled to the a data access unit.

16. (Original) The reconfigurable hardware system of claim 15, wherein the computational unit is supplied the data by the data access unit.

17. (Currently Amended) A method of transferring data comprising:  
transferring data between a memory and a data prefetch unit in a reconfigurable processor; and  
transferring the data between a computational unit and the a data access unit, wherein the computational unit and the data access unit, and the data prefetch unit are configured by a program.

18. (Original) The method of claim 17, wherein the data is written to the memory, said method comprising:  
transferring the data from the computational unit to the data access unit;  
and  
writing the data to the memory from the data prefetch unit.

19. (Currently Amended) The method of claim 17, wherein the data is read from the memory, said method comprising:  
transferring the data from the memory to the data prefetch unit; and  
reading the data directly from the data prefetch unit to the computational unit through [[a]] the data access unit.

20. (Original) The method of claim 19, wherein all the data transferred from the memory to the data prefetch unit is processed by the computational unit.

21. (Original) The method of claim 19, wherein the data is selected by the data prefetch unit based on an explicit request from the computational unit.

22. (Original) The method of claim 17, wherein the data transferred between the memory and the data prefetch unit is not a complete cache line.

23. (Original) The method of claim 17, wherein a memory controller coupled to the memory and the data prefetch unit, controls the transfer of the data between the memory and the data prefetch unit.

24. (Original) A reconfigurable processor comprising:  
a computational unit; and  
a data access unit coupled to the computational unit, wherein the data access unit retrieves data from memory and supplies the data to the computational unit, and wherein the computational unit and the data access unit are configured by a program.

### **REMARKS/ARGUMENTS**

Claims 1-24 remain in the application. Claims 1, 2, 5, 8, 11, 15 and 17 are amended to address informalities noted in the Office action. No new matter is added by these amendments.

#### **A. Drawings.**

The correction made to Fig. 2 is believed to overcome the objection to the drawings.

#### **B. Claim Objections**

Claims 1, 2, 5, 8, 11, 15 and 17 are amended to overcome the objections stated in the office action. It is respectfully requested that the objections to claims 1-23 be withdrawn.

#### **C. Rejections under 35 U.S.C. 112.**

Claims 1-10, 13 and 14 were rejected under 35 U.S.C. 112. This rejection is respectfully traversed.

Specifically, the Office action questions the reference to a first characteristic memory type and a second characteristic memory type in claim 1. This is illustrated, for example, in Fig. 3 in which a logic block 300 moves data from a first memory 305 having a first characteristic memory type to a second memory 307 having a second characteristic memory type. As set out in the paragraphs [0007]-[0016] of the specification, for example, the memory characteristics may include one or more of the following characteristics: line size, associativity, replacement policy, write policy, and cache size, all of which provide varying memory bandwidth efficiency and/or memory bandwidth utilization. The amendment to claim 1 is believed to clarify this feature of the invention and overcome the objections raised in the Office action.

With respect to claims 2 and 13, the examiner's interpretation that claims 2 and 13 do not require a hard-wired cache is accurate. It is noted that these limitations appear in claims 2 and 13, not claim 1.

The amendments to claims 3, 4 and 14 are believed to clarify the questions raised in the Office action.

Claims 2-4, 8-10 and 15-23 were rejected under 35 U.S.C. 112 as indefinite. The amendments to claims 2, 3, 4, 8, 15 and 17 are believed to overcome the rejections.

**D. Rejections under 35 U.S.C. 102.**

Claims 1-24 were rejected under 35 U.S.C. 102 based upon Paulraj. This rejection is respectfully traversed.

Independent claim 1 calls for a reconfigurable processor. As set out in Applicant's specification at paragraph [0039], a reconfigurable processor is a computing device that instantiates an algorithm as hardware. Although the reference show a reconfigurable cache, Paulraj does not show or suggest a reconfigurable processor that instantiates an algorithm as hardware. Moreover, nothing in Paulraj would suggest the rather significant changes required to replace the CPU with a reconfigurable processor. For at least these reasons claim 1 is not anticipated nor made obvious by Paulraj.

Claims 2-10 that depend from claim 1 are allowable over Paulraj for at least the same reasons as claim 1 as well as the limitations that are presented in those claims.

Claim 11 calls for a reconfigurable hardware system comprising one or more reconfigurable processors. As noted above with respect to claim 1, Paulraj does not show or suggest even one reconfigurable processor. For at least these reasons claim 11 and claims 12-16 that depend from claim 11 are believed to be allowable over Paulraj.

Independent claim 17 calls for, among other things, transferring data between a memory and a data prefetch unit in a reconfigurable processor. As noted above, Paulraj does not show or suggest a reconfigurable processor, nor transferring data between a memory and a data prefetch unit in a reconfigurable



processor. For at least these reasons claim 17 and claims 18-23 that depend from claim 17 are allowable over Paulraj.

Claim 24 calls for a reconfigurable processor having a computational unit and a data access unit that are configured by a program. Paulraj does not show a reconfigurable processor. Moreover, the element of Paulraj that stores and retrieves the configuration vector is not configurable by a program. Similarly, the element that executes and collects performance data is not configurable by a program. Paulraj does not suggest making these elements configurable.

**E. Conclusion.**

The references that were cited but not relied upon are no more relevant than the references that were relied upon. In view of all of the above, the claims are now believed to be allowable and the case in condition for allowance which action is respectfully requested. Should the Examiner be of the opinion that a telephone conference would expedite the prosecution of this case, the Examiner is requested to contact Applicants' attorney at the telephone number listed below.

Any fee deficiency associated with this submittal may be charged to Deposit Account No. 50-1123.

Respectfully submitted,

April 11, 2005

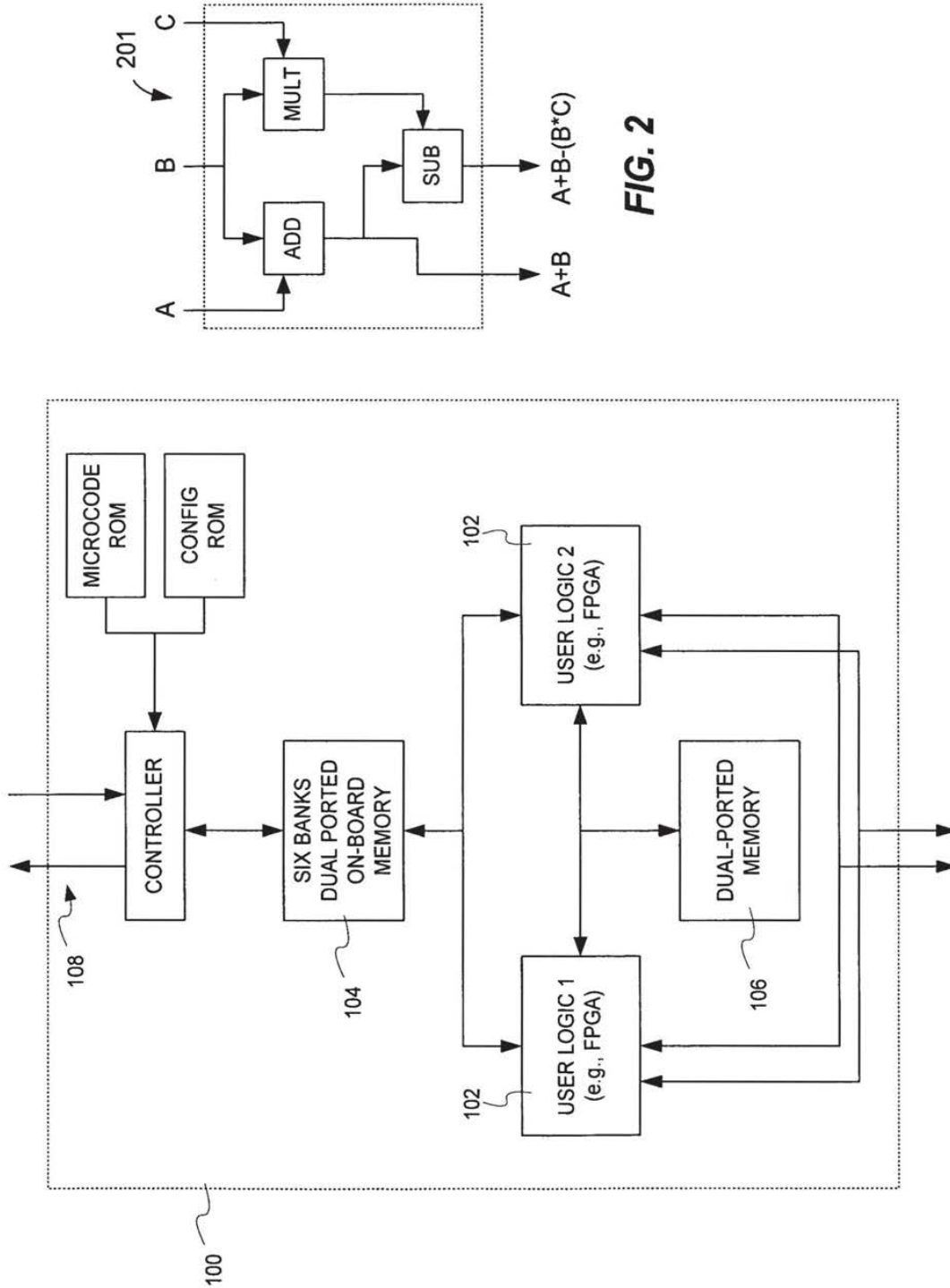
  
\_\_\_\_\_  
Stuart T. Langley, Reg. No. 33,940  
Hogan & Hartson LLP  
One Tabor Center  
1200 17th Street, Suite 1500  
Denver, Colorado 80202  
(720) 406-5335 Tel  
(303) 899-7333 Fax

Appl. No: 10/869,200  
Amdt. Dated April 11, 2005  
Reply to Office action of January 14, 2005

**B. Amendments to the Drawings:**

The attached sheet of drawings includes changes to Fig. 2. This sheet which includes Figs. 1-2 replaces the original sheet including Fig. 1-2. In Figure 2, element 201 is correctly identified.

Attachment:            Replacement Sheet  
                              Annotated Sheet Showing Changes





Client Matter No. 80404.0033.001  
Express Mail No.: EV330612115US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Serial No. 10/869,200	Confirmation No.: 5929
Application of: POZNANOVIC	Customer No.: <b>25235</b>
Filed: June 16, 2004	
Art Unit: 2186	
Examiner: THOMAS, Shane M	
Attorney Docket No. SRC028	
For: SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY BANDWIDTH IN RECONFIGURABLE HARDWARE	

CERTIFICATE OF MAILING BY EXPRESS MAIL

Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

Sir:

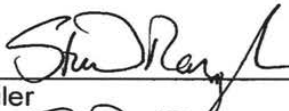
The undersigned hereby certifies that the following documents:

- Amendment and Response Pursuant to Office Action( 10 pages);
- Replacement drawing sheet (1 sheet);
- Information Disclosure Statement and copies of 3 references;
- Certificate of Mailing by Express Mail (1 page); and
- Return Receipt Postcard

relating to the above application, were deposited as "Express Mail", Mailing Label No. EV330612115US with the United States Postal Service, addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on April 11, 2005.

April 11, 2005  
Date

April 11, 2005  
Date

  
\_\_\_\_\_  
Mailer

  
\_\_\_\_\_  
Stuart T. Langley, Reg. No. 33,940  
HOGAN & HARTSON<sub>LLP</sub>  
One Tabor Center  
1200 17th Street, Suite 1500  
Denver, Colorado 80202  
(720) 406-5335 Tel  
(303) 899-7333 Fax



Express Mail No. EV330612115US  
Attorney Docket No. SRC028  
Client/Matter No. 80404.0033.001

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:

Daniel Poznanovic, David E. Caliga, and Jeffrey Hammes

Serial No. 10/809,200

Filed: June 16, 2004

For: SYSTEM AND METHOD OF ENHANCING  
EFFICIENCY AND UTILIZATION OF MEMORY  
BANDWIDTH IN RECONFIGURABLE HARDWARE

Group Art Unit: 2186

Examiner: Thomas, Shane M.

Confirmation No.: 5929

INFORMATION DISCLOSURE STATEMENT  
UNDER 37 C.F.R. 1.97

Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

Sir:

Applicant hereby submits for filing under 37 CFR 1.97 a disclosure statement. In submitting these references, no representation is made or implied that the references are or are not material to the examination of this application. The patents, publications or other information of which Applicant is presently aware are listed in Form PTO/SB/08A submitted herewith and copies of all such patents and publications are attached hereto.

No fee is believed due for this submittal pursuant Examiner's request for references in the Office Action dated January 14, 2005. However, any fee deficiency associated with this submittal may be charged to Deposit Account No. 50-1123.

Respectfully submitted,

4/11/05  
\_\_\_\_\_  
Date

  
\_\_\_\_\_  
Stuart T. Langley, Reg. No. 33,940  
HOGAN & HARTSON LLP  
One Tabor Center  
1200 17th Street, Suite 1500  
Denver, Colorado 80202  
(720) 406-5335 Tel  
(303) 899-7333 Fax



<b>INFORMATION DISCLOSURE          STATEMENT BY APPLICANT</b>  <i>(Use as many sheets as necessary)</i>		Application Number	<b>10/809,200</b>
		Filing Date	<b>June 16, 2004</b>
		First Named Inventor	<b>Daniel Poznanovic et al.</b>
		Art Unit	<b>2186</b>
		Examiner Name	<b>Thomas, Shane M.</b>
Sheet	1 of 2	Attorney Docket No.	<b>SRC028</b>

U.S. PATENT DOCUMENTS					
Examiner Initials	Cite No. <sup>1</sup>	Document No. No. – Kind Code <sup>2</sup>	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Doc	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
		US-6,076,152	06/13/2000	Huppenthal et al.	
		US-6,247,110	06/12/2001	Huppenthal et al.	
		US-6,356,983	03/12/2002	Parks	
		US-6,594,736	06/15/2003	Parks	
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			

FOREIGN PATENT DOCUMENTS							
Examiner Initials	Cite No. <sup>1</sup>	Foreign Patent Document		Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Doc	Pages, Columns, Lines Where Relevant Passages or Relevant Figures Appear	T <sup>6</sup>
		Country Code <sup>2</sup>	Number <sup>3</sup> Kind Code <sup>5</sup>				

<b>EXAMINER SIGNATURE</b>	<b>DATE CONSIDERED</b>
<p>EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant. <sup>1</sup> Applicant's unique citation designation number (optional). <sup>2</sup> See Kinds Codes of USPTO Patent Documents at <a href="http://www.uspto.gov">www.uspto.gov</a> or MPEP 901.04. <sup>3</sup> Enter Office that issued the document, by the two-letter code (WIPO Standard ST.3). <sup>4</sup> For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. <sup>5</sup> Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST. 16 if possible. <sup>6</sup> Applicant is to place a check mark here if English language Translation is attached.</p> <p>This collection of information is required by 37 CFR 1.97 and 1.98. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) and application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 2 hours to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.</p>	



## 4 CODE DEVELOPMENT AND PORTING ISSUES

### 4.1 ALLOCATING MAP RESOURCES

When a program begins execution on the SRC-6E system, the only resources available to it are the local memory and processor assigned to it by the operating system at program start-up. The user must allocate MAP resources needed prior to calling a function, which will execute on the MAP. The function, which allocates and initializes MAP resources, is `map_allocate`. The prototype for this function is:

```
int map_allocate (int nmaps);
```

where `nmaps` is an input parameter indicating the number of MAPs to allocate for the job.

1 <= `nmaps` <= `MAXMAPS` (SRC6-E system only supports one MAP to be allocated per program executable)

The function's return value indicates if the allocation was successful (0) or not (<>0).

**NOTE:** MAPs are identified by their MAP ID number. If `n` MAPs are currently allocated, their ID numbers are 0 thru `n-1`.

### 4.2 RELEASING MAP RESOURCES

It is necessary to explicitly release any MAP resources allocated during the execution of a program prior to termination of the program when executing on a MAP or in emulation mode. The function that frees MAP resources is `map_free`. The prototype for this function is:

```
int map_free (int nmaps);
```

where `nmaps` is an input parameter indicating the number of MAPs to release from the job.

1 <= `nmaps` <= `MAXMAPS` (SRC6-E system only supports one MAP to be allocated per program executable)

The Function's return value indicates success (0) or failure (<>0) of the release.

**NOTE:** `map_free` releases the MAPs with the highest MAP ID's first. For example, consider an application that has seven MAPs allocated to it, MAP IDs 0, 1, 2, 3, 4, 5 and 6. After the execution of the statement:

```
stat = mapfree( 2 );
```

there are five MAPs remaining allocated to the job with MAP IDs 0, 1, 2, 3, and 4.





## 4.3 APPLICATION MODIFICATIONS FOR MAP EXECUTION

To take full advantage of the MAP hardware, it is necessary for the user to make some modifications to the application. As the compiler technology matures and more optimizations can be automatically performed that target the unique characteristics of MAP hardware, it will become less important for the users to understand the behavior of the hardware and modify their code accordingly. The next sections describe required application modifications for MAP execution. The modifications include partitioning the code for MAP execution into a new file, restructuring the partitioned code, inserting MAP resource management library calls, and inserting calls to the function compiled for MAP execution in the code that executes on the Intel processor.

### 4.3.1 Partitioning The Code

The first step in porting an application to the SRC-6E system utilizing the MAP hardware is to identify some portion of the application such that, when that portion is compiled for the MAP, overall performance will improve. Loops are often good candidates for execution on the MAP. Loop nests and their associated loop bodies, that can be pipelined, have shown execution speed-up. Another potential improvement could be in the process of manipulating single bits from within a long bitstream of data.

**NOTE:** Once a section of code has been identified for MAP, it must be placed in its own separate function and be the only function in that file.

The function name will be the base name of all the components generated as a result of compilation. Similarly, the filename that contains the function to be compiled, minus .c or .C suffix, becomes the base name for all files generated as a result of compilation. The function has the computational portion of code that will be executed on the MAP. It must also be modified to manage the data movement to and from the MAP's On-Board Memory (OBM). Data movement functions must be inserted to manage data movement between the System Common Memory (SCM) and OBM. At the beginning of program execution, all data (memory) resides in SCM. Any array data needed by a function executing on the MAP must be moved to OBM explicitly, and any array result data computed during MAP execution must be explicitly moved back to SCM. The compiler automatically moves scalar formal parameters of the MAP function to and from the MAP as needed. Array data resides in OBM structures during MAP execution. The OBM data movement functions are described in the next section.

In addition to isolating the MAP code in its own function, and the insertion of data movement function calls, other code modifications may be required for MAP execution. Performance gains using the MAP will generally come through the efficient execution of loops. The MAP C Compiler attempts to pipeline loops, where a loop's iterations are fired one per clock. The MAP C Compiler has restrictions on the kinds of loops it can pipeline, and ongoing work on the compiler at SRC Computers, Inc. is aimed toward reducing these restrictions.



Loops may not have "breaks" in them, since a break creates a multiple-exit loop body. Because of C's rules regarding the evaluation of the '&&' and '||' logical operators, in which the right-hand expression is evaluated only if the left-hand side does not resolve the operator's result, the use of these operators in a loop termination expression will produce a multiple-exit loop and a compiler error such as:

```
error (martello, #32): can't pipeline a multiple-exit loop
```

To avoid this, use the bitwise operators '&' and '|' instead. Here is an example, where an array is searched for the value '42':

```
for (i=0, a=0; (i<m)&(a!=42); i++) {  
    a = AL[i];  
    idx = i;  
}
```

In general, a MAP function may have only one lexical reference, read or write, to each bank of OBM. The exception to this is that a pipelined loop may contain up to eight reads from a given bank. When multiple reads occur, the loop code generated by the compiler is throttled so as to use N clocks per iteration, where N is the maximum number of reads to any bank from within the loop. This is because the MAP hardware can reference only one OBM word per clock.

A structural subtlety arises with regard to the restriction that multiple code blocks cannot reference the same bank of OBM in a MAP function. Because the MAP C Compiler creates bottom-test loops in its dataflow graphs, a 'while' loop is implemented by creating a zero-trip test outside of the loop, and then converting the loop to a bottom-test. This means that a test such as:

```
while (A[i] != 42)
```

will result in two array references to 'A', and compilation will produce the error:

```
error (martello, #25): multiple reference to bank 'a'
```

In general, bottom-test loops will produce slightly more space-efficient MAP implementations since the zero-trip test does not need to be created.

Occasionally a user might wish to prevent loop pipelining. The most common reason would be an inner loop that contains a call to an external macro. Since external macros don't pipeline, the MAP C Compiler will issue an error when it sees the external macro. To tell the compiler not to pipeline, use the '-nf' option.

All data referenced or defined during execution of the MAP function is explicitly passed through the formal procedures of the function. No global data (externs) can be accessed by the MAP function, with the exception of the specifically named structures associated with the OBM. These structures are described in the following section.



Array parameters must be declared with the square bracket notation ( [ ] ) rather than as pointers to data. Pointer arithmetic is not supported in MAP functions.

The last formal parameter in the MAP function's argument list is a default int mapid. mapid is the number that indicates the MAP on which the function is to execute. The MAPs allocated to a job are identified by an integer in the range of 0 thru n-1 where n is the number of MAPs currently allocated to the job. (Refer to section 4.1 for more information about MAP allocation.)

**NOTE:** Only one MAP may be allocated to a user's job, and thus, the mapid specified for a MAP function to execute on should be 0.

The function must not contain any external calls or external function references except those that will be linked to either SRC defined or user defined macros (Refer to section 5). In this sense, functions must be wholly contained on the MAP. No I/O, system calls, or other runtime functions such as memory allocation that require operating system intervention are allowed.

Data types int, long, and long long are supported. The intrinsic operators +, -, \*, .==, !=, >, >=, <, and <= are fully supported for these types. The intrinsic operator / (division) is supported only for 32-bit integers at this time, as is the math function SQRT. The bitwise operators &, |, and ! are provided for signed and unsigned int and long long (32 and 64-bit) data types. Left and right shift operators, << and >>, are supported for signed and unsigned int and long long types. Logical operators &&, !, || are also supported.

Floating-point and complex data types are supported for MAP functions; however, operations on floating-point and complex data are not supported. Logical data types and operations are not yet supported.

**Table 4. Supported Operations**

Type=	Integer				Real		Boolean	Complex	
	char	short	int	long long	float/ double	long double		complex	long double complex
Addition			Y	Y					
Subtraction			Y	Y					
Multiplication			Y	Y					
Division			Y						
==, !=, <=, >=, <, >			Y	Y					
!, &,  , ^			Y	Y					
&&,   , !, ^			Y	Y					

The following is a list of criteria that must be met with regards to data in a MAP function:

- Except as parameters to data movement functions, function array formal parameters must not be referenced or defined.
- Function array formal parameters must be declared using square brackets ( [ ] ) to distinguish them from formal parameters that are pointers to scalars.
- Formal parameter arrays must be of either signed or unsigned long long type (64-bit integer data).
- Lexically, each OBM bank can be read from only one code block, and written by only one code block. If a bank is both read and written, then the read and write must be from separate code blocks. The number of writes to a bank is limited to one. The number of reads from a bank is limited to eight, and the reads must all occur in the same code block.



### 4.3.2 Local Scalars And Arrays

The MAP Compiler supports local arrays, of one and two dimensions, which are allocated to Block RAM within the user chip. These arrays are available in 32-bit and 64-bit widths, and in size increments of 512 up to 8192 words. Any number of arrays can be allocated up to the limit of the Block RAM space in the user chip (144 512x32 RAM units). The user may specify any size up to 8192; the compiler will allocate a size to the next increment of 512. For example:

```
int T0[1000];
```

The compiler will allocate the 1024x32 BRAM to the array 'T0'. Each local array is subject to the same read/write restrictions that exist for OBM accesses; lexically, each array can be read from one code block and written from one code block. The read and write must be from separate code blocks. The number of writes to an array is limited to one. The number of reads is limited to eight, and they must occur in the same code block.

32-Bit	64-Bit
512X32	512X64
1024X32	1024X64
1536X32	1536X64
2048X32	2048X64
2560X32	2560X64
3072X32	3072X64
3584X32	3584X64
4096X32	4096X64
4608X32	4608X64
5120X32	5120X64
5632X32	5632X64
6144X32	6144X64
6656X32	6656X64
7168X32	7168X64
7680X32	7680X64
8192X32	8192X64

### 4.3.3 Data Alignment And Movement Functions

The user must correctly declare the data needed for the MAP computation. All arrays, passed as arguments to a MAP function, must be cache aligned (aligned on multiple 32-byte cache line boundaries) in order for correct movement between SCM and MAP OBM. This can be achieved using setting pointers to a cache aligned address within an array that has been padded in size to allow alignment. To facilitate this alignment, the function `addr32` exists in the MAP library (`libmap`). An example of the use of this function is given in the example. The prototype for this function is:

```
void *addr32 (void *addr);
```

where "addr" is the address of an array that has been declared with padding for alignment, and the function result is the address of the first cache aligned word within that array.

Alternately, cache alignment of data can be achieved in C by using a memory allocation function that provides cache-aligned buffers. The prototype for the cache-aligned allocation function is:

```
char *Cache_Aligned_Allocate(int size);
```

where "size" is the requested buffer size in bytes. The function returns a pointer to the requested buffer.

Cache aligned buffers can be freed using the `Cache_Aligned_Free` function. The prototype for this function is:

```
void Cache_Aligned_Free(char *buffer);
```

where "buffer" is a pointer to the buffer that is to be freed.

Special functions handle the data movement between SCM and MAP OBM. Since OBM is actually comprised of 6 banks, each with separately addressable memory, SRC developed a syntax for users that identifies where data is to be moved when transferring it from local memory to OBM. This syntax involves utilizing structures whose names specifically correspond to the six banks of OBM. The banks or structures are named `banka`, `bankb`, ..., `bankf`. Each structure may only contain one array member and no other members. All array data involved in MAP computation must reside in one of these six specifically named structures.

For example:

```
#define size 10000
struct {
    uint64_t data_array [size];
} banka;
```

Rather than referencing the array through a structure dereference, a pointer may be created:

```
uint64_t *data_array = banka.data_array ;
```

If such a pointer is used, the name of the pointer must be identical to the structure member array it points to.

To simplify the declaration of the structures, member arrays, and pointers, the user may wish to define macros for doing so. The macros may be placed in a header file and be included in all the users MAP functions. An example of a macro for allocating a single dimensioned array in OBM banka may look like:

```
#define BANK_A_ALLOC(_name, _type, _size) \
    struct {\
        _type _name[_size];\
    } banka;\
    _type *_name = banka._name;
```

Assuming this macro is placed in the user header file `my_macros`, the declarations above of the structure `banka` with a member `data_array` of 10000 elements of type `int` and the associated pointer to that array would be replaced with:

```
#include my_macros
#define size 10000
BANK_A_ALLOC(data_array, uint64_t, size)
```

Multiple arrays may be allocated in an OBM bank. For example, the following declarations allocate two arrays in `banka`:

```
struct {
    int64_t AL0[32];
    int64_t AL1[32];
} banka;
int64_t *AL0 = banka.AL0;
int64_t *AL1 = banka.AL1;
```

However, because multiple reads to an OBM bank can occur only from within one pipelined loop, these two arrays can be referenced only from within the same pipelined loop.

To transfer data from SCM to OBM, six functions are provided, one for each OBM bank. To transfer data from OBM to SCM there are also six functions, one for each OBM bank. Finally, there are six functions to synchronize completion of transfers to or from each OBM bank. The function prototypes are:

```
void cm2obm_x(void *obm_addr, void *cm_addr, int length);
void obm2cm_x(void *cm_addr, void *obm_addr, int length);
void wait_server_x();
```

where "`x`" is the name of the OBM banka thru `bankf`; "`cm_addr`" is the first word address in SCM; "`obm_addr`" is the first word address in OBM (see `OBM_stripe` function below); and "`length`" is the number of bytes to transfer. Using these functions, it is only possible to read or write a given bank within a MAP function. For example, the use of `cm2obm_a` and `obm2cm_a` within the same function is not permitted because OBM banka is both read and written.

In some cases, it is desirable to stripe data from SCM across multiple banks of OBM. Addition functions are provided to stripe into and out of OBM. The prototypes for these functions are:

```
void cm2obm_svn(void *obm_addr, void *cm_addr, long long obm_stride,
               long long cm_stride, int length);
void obm2cm_svn(void *cm_addr, void *obm_addr, long long cm_stride,
               long long obm_stride, int length);
void wait_server_svn();
```

where "n" is a digit in the range of 0-5; "obm\_addr" is the first word address in OBM; "cm\_addr" is the first word address in SCM; "obm\_stride" is a constant which represents the stripe/stride pattern in OBM; "cm\_stride" is the SCM stride; and "length" is the transfer size in bytes.

**NOTE:** See the *SCR-6E MAP<sup>®</sup> Hardware Guide* for further information on use of the stripe and stride transfers.

A function, `OBM_stripe`, is provided which creates the `obm_stride` argument to the above routines. This is an integer constant which represents the stripe pattern across the OBM banks and the stride in the banks. The function prototype is:

```
int64_t OBM_stripe (int stride, char *stripes, int *err);
```

where "stride" is the stride between elements in an OBM bank; "err" is the error return code of the function (0 = successful; <0 = an error occurred); and "stripes" is a character string representing the OBM bank stripe pattern of the transfer. Stripes is comprised of one to sixteen characters A-F or X, all upper case, each separated by a comma. The characters A-F represent the banks in which to place (take) the next value transferred, and X indicates a skipped value. For example, `stride = 1`, and `stripes = A,C,A,E,X`, `OBM_stripe` returns a value, which if used as the `obm_stride` argument to `cm2obm_svn`, results in values coming from SCM to be placed in OBM as follows. The first value goes to bank A, the second to bank C, the third to bank A, the fourth to bank E, the fifth is discarded (ignored). If more values remain to be transferred, the pattern is repeated until all values are transferred. In the case of a transfer from OBM to SCM (`obm2cm_svn`) the value 0 is transferred to SCM for each bank designated as "X" in the string.

It is possible to DMA to and DMA from the same OBM bank using the `cm2obm_svn` and the `obm2cm_svn` functions. For example, a MAP function can have an input array that is both an input, and an output to the function. A `cm2obm_svn` function may be used with the `obm_stride` value set to write to only a single bank to transfer input values to OBM. After MAP computation is complete, a `obm2cm_svn` function may be used with the `obm_stride` value set to read from only the same bank to copy the array out of OBM.

There are a total of six OBM servers, which perform the memory transfers between SCM and OBM. These servers are shared by the bank specific transfer functions (those ending with a specific bank identifier A-F) and the stripe transfer functions (those ending with a server number 0-5). Each OBM server may be used only once per MAP function. The functions sharing a given OBM server are:

```
Server 0: cm2obm_a, obm2cm_a, cm2obm_sv0, obm2cm_sv0
Server 1: cm2obm_b, obm2cm_b, cm2obm_sv1, obm2cm_sv1
Server 2: cm2obm_c, obm2cm_c, cm2obm_sv2, obm2cm_sv2
Server 3: cm2obm_d, obm2cm_d, cm2obm_sv3, obm2cm_sv3
Server 4: cm2obm_e, obm2cm_e, cm2obm_sv4, obm2cm_sv4
Server 5: cm2obm_f, obm2cm_f, cm2obm_sv5, obm2cm_sv5
```

Given this sharing, a MAP function may not call both `cm2obm_d` and `obm2cm_sv3` as both of these functions share OBM server 3.

The data transfers between SCM and OBM are asynchronous. The wait server functions wait for the transfers to complete before execution continues.

The concepts of the previous section are demonstrated in the following example.



The original code to be modified for MAP execution is as follows:

```
#include <stdio.h>
#include <sys/types.h>

#define SIZE 64

int main () {

    int i,n;
    int s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10;
    long long a[SIZE + 8], b[SIZE + 8];

    for (i = 0; i < SIZE; i++) {
        b[i] = 6 - (i % 12);
    }

    /*****
    *   10th degree horner's rule polynomial evaluation.
    *
    *   n       - Vector length
    *   a       - Input vector (addr)
    *   b       - Output vector (addr)
    *
    *****/

    /* Coefficients */
    s0 = 1;
    s1 = 2;
    s2 = 3;
    s3 = 4;
    s4 = 5;
    s5 = 6;
    s6 = 7;
    s7 = 8;
    s8 = 9;
    s9 = 10;
    s10 = 11;

    /* calculation loop */

    for (i = 0; i < n; i++) {
        a[i] = s0 + b[i] * (s1 + b[i] * (s2 + b[i] * (s3 + b[i]
            * (s4 + b[i] * (s5 + b[i] * (s6 + b[i]
            * (s7 + b[i] * (s8 + b[i] * (s9 + b[i]
            * s10))))))));
    }

    /* Print Results */

    for (i = 0; i < SIZE; i++) {
        printf ("b[%d]: %lld \t a[%d]: %lld \n",
            i,b[i],i,a[i]);
    }
}
```

When modifying code for MAP execution, the computational portion of the code that will benefit from MAP execution is first identified. In this example, the for-loop on the "db" array is what is targeted. The loop