

Formalizing Process Protection in Application Adaptive Reconfigurable Architectures

Jay H. Byun and Andrew A. Chien

jaybyun@uiuc.edu, achien@cs.ucsd.edu

University of Illinois

University of California, San Diego

December 12, 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 conventional processor and operating system structure, identifying the few critical structures and operations of the hardware that enable multiprocess protection semantics of the operating system. We then introduce the concept of path constraints, a framework for reasoning about the permissible reconfigurability of these safety-critical structures and operations. Based on these analyses, we propose a protection architecture for the Morph/AMRM reconfigurable processor which enables most of the useful reconfigurability in the processor core while preserving the path constraints that 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 μ 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, and instead those will be dominated by interconnect performance and wiring density. 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 reconfigurability in the processor can be used to significant advantage [4,5], improving performance by factors of 10 to 100x for while avoiding the traditional disadvantages of custom computing approaches such as I/O coprocessor coupling and slower logic [6]. Owing to this fact, last few years have seen a tremendous increase in interest in reconfigurable architectures from the research community as well as the industry, resulting in a proliferation of researches that demonstrate a large potential for dramatic performance improvement over conventional processor architectures[7,8,9]. 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.

Reconfigurable, or application adaptive processors can utilize the integrated reconfigurable logic to tune the processor to better match each application. However, introducing application-dependent reconfigurability in the processor raises significant challenges for ensuring process isolation and protection (multiprocess isolation), a critical element of even desktops 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. Thus, it is very clear that multiprocess protection issue is fundamental to a widespread use of reconfigurable architectures in mainstream systems, but such an issue has not been addressed by the reconfigurable architecture community. In this paper, we formulate the problem and the solution of multiprocess protection in integrated reconfigurable processors and apply it to our on going project, the Morph/AMRM reconfigurable processor design.

To solve the multiprocess protection problem, we begin by examining the protection structures of conventional processors and OSs. Specifically studying the MIPS R10000[10] microprocessor, an exemplar of a system employing Unix/RISC protection architecture, we identify the hardware mechanisms and structures that are critical for enabling the Unix/RISC protection semantics. The key feature of the protection architecture is process isolation via address isolation and mediation. To enable such OS protection structure, the following hardware mechanisms and structures are critical:

- isolation of privileged data structures (processor control bits, TLB, kernel address space) through processor operating mode distinction,
- controlled address translation for all generated virtual addresses to enforce the OS managed address mapping,
- trap/interrupt mechanisms for returning control to the kernel for OS mediation.

Thus, to preserve this protection framework in reconfigurable processors, we cannot allow arbitrary reconfigurability to compromise these safety-critical mechanisms and structures. To formally reason about the permissible reconfigurability and multiprocess safety, we introduce the concept of path and path constraints. A path represents a flow of values through the components in the processor core. We propose a set of constraints that can be imposed on the path abstractions and show that the multiprocess protection

properties can be preserved in the reconfigurable processor by enforcing these constraints on the safety-critical paths.

In addition to preserving the conventional protection semantics, isolation and virtualization of the reconfigurable logic blocks belonging to each process must also be considered. Since we allow per process adaptation in the reconfigurable logic blocks, the configurations of them should be properly viewed as an extension of the application process' "virtual machine." This requires a coordinated isolation and context switching of reconfigurable logic blocks and other process resources. We can achieve this with the synchronous hardware context switching mechanism introduced in this paper.

Putting all the pieces together, we present the Morph/AMRM protection architecture as an example. By hardwiring critical parts in the processor core (satisfying path constraints), this architecture ensures multiprocess protection while retaining much of the useful reconfigurability. 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 our analysis of the software and hardware mechanisms central to the process protection in conventional processors and operating systems. Section 3 presents the implications of reconfigurability on process protection and the additional requirements that it induces. Path-based view of the protection architecture and the path constraints for reconfigurable paths are given in Section 4. In Section 5, we describe the Morph/AMRM system and a proposed protection architecture that meets the requirements set forth in Section 3 and 4. Section 6 analyzes the various tradeoffs related to reconfigurability and discusses alternate approaches. Section 7 summarizes future work and the material covered in this paper.

2 Process Protection in Conventional Processors

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.

2.1 OS Semantics for Process Protection

We first examine how a UNIX style operating system[11,12] ensures process isolation and thereby derive the hardware requirements it imposes. Then identify the corresponding support in the example RISC processor (MIPS R10000). The survey of conventional OS protection architecture shows that fundamentally, process protection is provided through process isolation via address isolation, resource access mediation, and safe transition of virtual machine states.

Application and Operating System Memory Isolation: The first entity to isolate and protect in order to provide process protection is the private memory space of each process and the kernel. Application and operating system memory isolation is achieved through controlled address translation (figure 1). 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). It follows that the following must be guaranteed to provide the memory space isolation,

1. The operating system controls and manages all address translation information (page tables, TLBs)
2. Application processes cannot alter the address translation information
3. All application accesses are subject to this translation

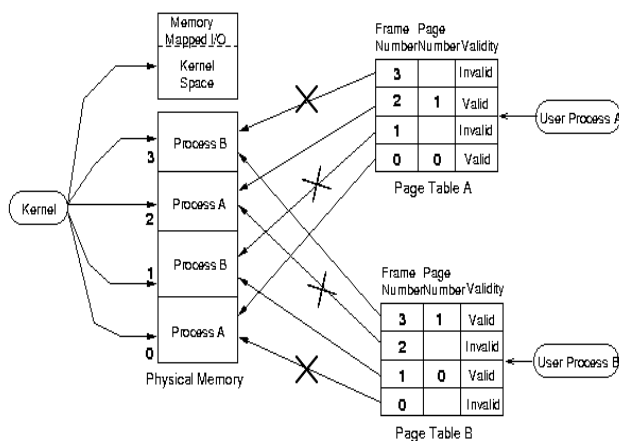


Figure 1: Multiprocess Protection based on Address Space Isolation

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. 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.

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 responsible for safe mode transition, context saving/restoring, and priority based servicing.

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. To do this, the OS and the hardware must capture the all the states required to describe the virtual machine and ensure safe transitions between virtual machine states (i.e. safe context switching). In UNIX for example, 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.

2.2 Hardware Guarantees for Process Protection

We can identify the following mechanisms that are implicitly provided by the hardware to enable the protection structure dictated by the OS.

Isolation of protection data structures - Execution modes and kernel address space: The protection structure described above required that protection data structure such as address translation only be allowed access by the OS. The processor supports this by distinguishing execution modes. The processor should at least provide two modes of execution, i.e. privileged execution(kernel) mode and user mode, so that the kernel data structure, special registers, and processor control bits can only be access and altered when executing in privileged mode. The data structure critical to process protection, e.g. TLB, the processor control bits (CP0 control processor registers in R10000) – which contains the processor mode bit itself, can only be accessed by a privileged instruction. Also, a kernel address space, which can only be accessed in privileged mode, is provided. The data structure critical to process protection such as page tables and process control blocks reside in this memory space. The processor faults when privileged accesses are attempted in user mode.

Trap/Interrupt generation and delivery: The OS relies on trap/interrupt generation and delivery mechanism provided by the processor in order to take control of the system. Resource mediation, context switching, fault handling which were all integral part of protection structure all relied on the trap/interrupt mechanism. The hardware must correctly generate of trap and interrupt signals from appropriate component of the hardware and correctly deliver it to the OS handler. The delivery entails switching to privileged mode with appropriate status bits and pointer to appropriate handler set.

Enforcement of controlled address translation in MMU: The memory management unit, which comprises the effective address calculation unit and the TLB, is placed between other parts of the processor and the physical memory. The processor must be designed so that no processor component can bypass the controlled address translation provided by the OS in the MMU. The MMU has the control logic and interface to allow OS to manage address translation information.

Arbitration hardware for shared resources: Another aspect of a multiprocess safe system is lockup freedom. Lockup freedom may not be a strict requirement for process isolation, but nonetheless important if we were to build a robust multiprocess system. Thus in addition to a lockup free resource

management by the OS, the hardware must provide an arbitrator logic implementing a lockup free protocol to control accesses to shared hardware resources(not visible to OS) such as the system bus.

3 Integrated Reconfigurable Processors and Process Protection

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 reconfigurable hardware to behave within the safe thresholds set by the behavior of safe and non-reconfigurable hardware.

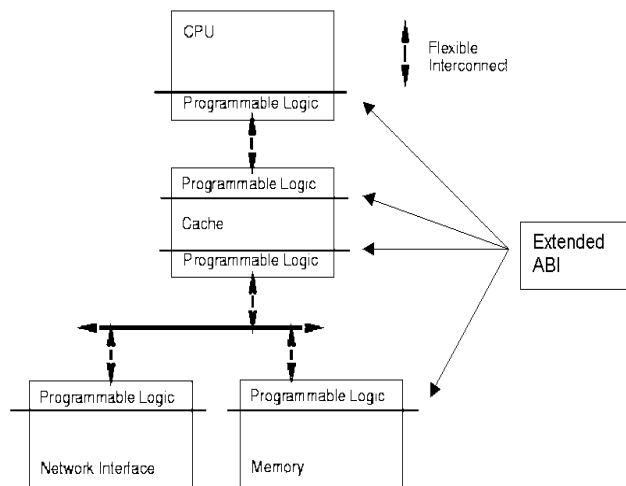


Figure 2: The canonical application-adaptive reconfigurable architecture, where elements of reconfigurable logic can in general be attached to all elements of interconnect, logic, and memory in the system.

3.1 Model of Integrated Reconfigurable Processor

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 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[3]. 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. In other words, the application adaptive reconfigurable processor can utilize the integrated reconfigurable logic to tune the processor to better match the application. This is different from a more traditional view of reconfigurable processor as a co-processor built entirely out of a reconfigurable logic block. To differentiate from the traditional reconfigurable processor, we will refer to this application adaptive processor as an integrated reconfigurable processor. This novel approach is embodied in the Morph/AMRM (Adaptive Memory Reconfiguration Management) architecture [4,5]. We characterize integrated 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 2. In this basic 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 CPO register or a range of TLB entries). In summary, in the most general case, the configurable hardware can be attached to or replace any part of the entire system, its actions can affect any part of the hardware system.

3.2 Implications of Reconfigurability on Process Protection

3.2.1 Reconfigurability and Conventional Hardware Guarantees for Process Protection

The generic OS protection structure relies on the hardware guarantees that were identified in section 2. However, in an integrated reconfigurable processor, where we allow reconfigurable hardware to augment or replace any part of the system, arbitrary reconfigurability can easily invalidate those guarantees. For instance, a reconfigurable TLB may compromise the guarantee of controlled address translation that the hardware must provide, undermining the OS protection structure based on address space isolation. Another example is where reconfigurable hardware may inject or divert trap/interrupt signals to/from various components of processor core and processor control logic. In fact, all of the hardware guarantees that were essential in implementing generic OS protection structure are no longer preserved. This results in a reconfigurable processor that has no provision for the OS to build even the most simple multiprocess environment. The later sections of this paper investigate the solutions to preserving those critical hardware guarantees and thereby producing a multiprocess safe integrated reconfigurable processor.

3.2.2 Extending the Concept of Process Protection

Per-application adaptation of reconfigurable logic throughout the hardware extends the conventional concept of a process. The conventional view of a process has three component: 1. an executable program, 2. The associated data needed by the program (variables, work space, buffers), 3. The execution context of the program (processor state, register values...). Process protection in the conventional sense meant isolating the program text and data residing in the physical memory space and safely swapping the execution context in and out. With the per-application adaptation of reconfigurable logic, the configuration of a reconfigurable logic block becomes the executable program, the associated data, and the context of the process for which the reconfigurable logic block is configured. This implies that the configuration of a reconfigurable logic block is also an entity that must be isolated and safely context switched as a part of the process.

As outlined above, 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 <-> reconfigurable hardware interactions
3. Reconfigurable hardware <-> reconfigurable hardware interactions

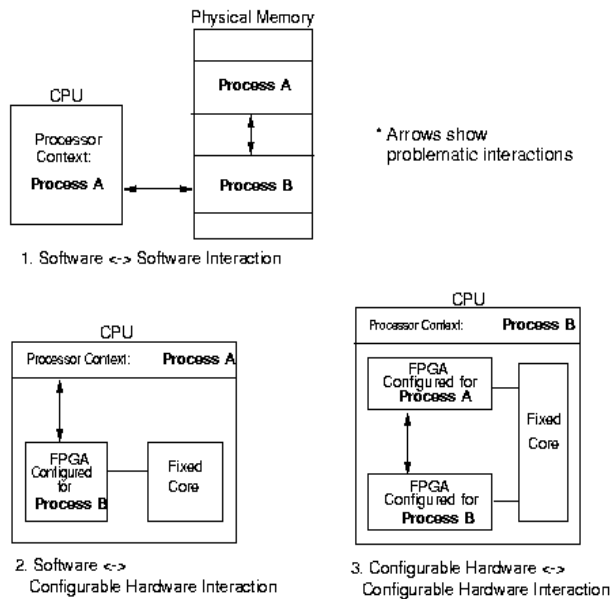


Figure 3 . Three types of interactions can cause protection compromises in application-adaptive configurable machines. Arrows show problematic interactions.

These cases are illustrated in Figure 3. 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.

3.2.3 Synchronous Hardware Context Switch Model

Requirements for process isolation in a configurable architecture extend the hardwired system requirements outlined in Section 2, requiring coordination across software and configurable hardware, and controlled access in all parts of the system that configurability is allowed. More specifically, the integrated reconfigurable processor requires a mechanism to synchronously switch processor state and all of the process' reconfigurable hardware throughout the system to eliminate newly introduced concerns of software \leftrightarrow reconfigurable hardware and reconfigurable hardware \leftrightarrow reconfigurable hardware interactions. To achieve this, the synchronous context switching mechanism must guarantee the following:

1. Reconfigurable logic not used by the active process must be disabled and/or bypassed.
2. Reconfigurable logic used by the active process must be enabled and loaded with the configuration for the active process. If not loaded with the configuration for the active process, swap out old configuration and swap in configuration for active process.
3. These actions must be carried out within the execution of software context switch, before the control is returned to the new active process.

This requires the following extension to the architecture:

- **Reconfigurable logic block enable/disable logic & bypass path:** to selectively disable and bypass RL

- **Configuration context register:** indicates which reconfigurable logic blocks are to be enabled for the active process. Swapped during context switch along with conventional context register.
- **Table of Configuration owners & match logic:** records which process's configuration is loaded in reconfigurable logic blocks. This bookkeeping is needed in order to swap configurations of reconfigurable logic blocks that does not match new active process.

In summary, to provide process protection in reconfigurable hardware, we must 1. preserve the conventional hardware requirements for process protection which may be compromised with integrated reconfigurable logic blocks, and 2. extend isolation and virtualization to include processes' reconfigurable logic blocks.

4. Path-based Analysis of Hardware

The formal framework for reasoning about the requirements and guarantees for process protection in integrated reconfigurable processors is provided by the path-based analysis of hardware at component level. A "path", which will be more formally defined later on, is an abstraction of the chain of components and the interconnections involved in a particular operation of the processor. As we have identified in the previous sections, there are several operations of the processor whose behavior must be correct to ensure process protection. For these critical operations, we would like to find out what constraints to impose on their path abstractions to enforce correctness of process protection in the presence of reconfigurable logic. We present general formal constraints of varying degrees, from which we will be able to derive the scope of flexibility and the level of guarantees.

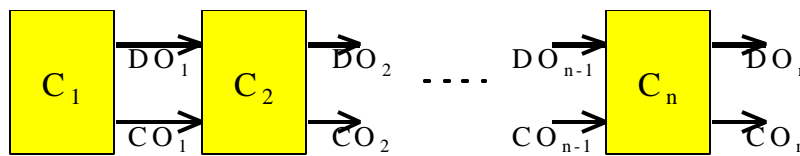
4.1 Definition of Path

A processor operation goes through a certain chain of components in the processor core and in the system board. Data represented as bits asserted on the output lines of these components are processed in each

component and then carried to the next component in the sequence. To reason about correctness at each component *without directly dictating what its detailed implementation should be*, we view correctness in terms of values asserted in the outputs of each component. In other words, we would like to represent the data flow of values during a specific processor operation as a path. To do so, we make an abstraction of the components as the control output lines and data output lines. The data output lines are the lines that carry some form of address information or data, for example the data bus and the address bus interconnecting the components. Others are control lines that carry control signals to other components, e.g. disable/enable, read/write, request/grant, command, mux select, etc. The control lines carry control signals used to sequence the components for an operation whereas the data lines carry pure data to be processed or transferred to the next component.

Definition: A *Path* is an ordered sequence of *Components*, C_i

where each *Component* is a tuple $\langle do_i, co_i \rangle$



Path of length n

and do_i = data outputs of C_i , co_i = control outputs of C_i

A path is an ordered and directed sequence, which means that the control and data flow oneway downstream from the head component to the last component in the path.

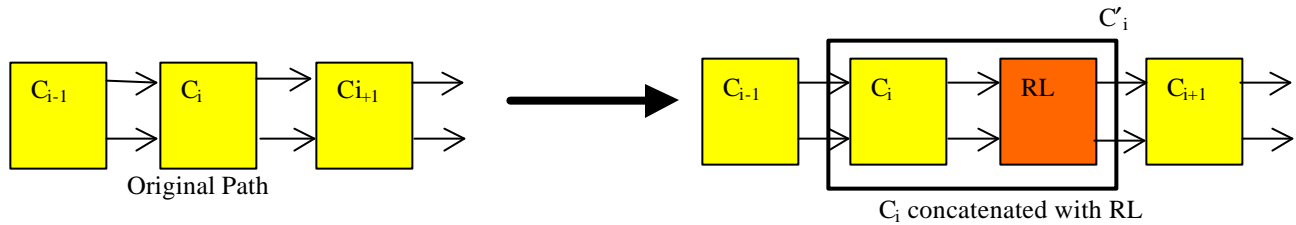
4.2 Reconfigurable Logic Augmented Path

In integrated reconfigurable processors, reconfigurable logic (RL) blocks can potentially replace or attach to any parts of the processor. Therefore, reconfigurable logic components can be added anywhere in along a path P , creating a new path P' . We constrain the reconfigurable logic augmented path by comparing its

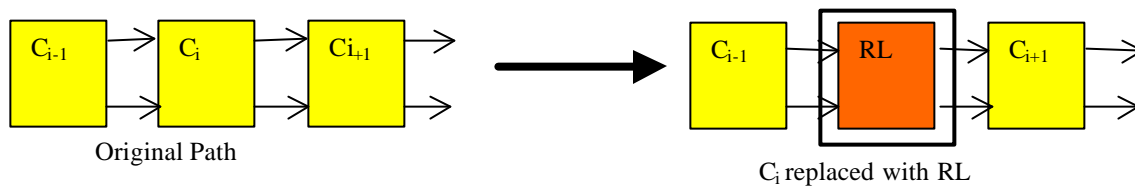
behavior to base path. To reason simply about the comparison, we structure the addition of configurable logic in the following ways:

1. **Concatenation:** RL can be **concatenated** with a component C_i to create C'_i in the i th position in the path.

The input ports and output ports of RL must be a superset of the ports of C_i .



2. **Replacement:** RL can **replace** a component C_i with a (reconfigurable) component C'_i . Again the input ports and output ports of C'_i must be a superset of the input ports and output ports of C_i .



These two mechanisms for path augmentation cover a wide variety of the useful possibilities, and define a new path P' . This allows component-by-component comparison of the original path and the reconfigurable logic augmented path.

4.3 Isolation Constraints on RL Augmented Paths

We express the correctness constraints on the values asserted on component outputs of the RL augmented path P' . We begin by considering the execution of a single process. The constraints are expressed with respect to the values asserted in the safe, base path. As discussed in Section 3, the base path presumes appropriate operating system behavior to ensure process isolation. If the constraints are met the RL augmented path combined with the same operating system behavior will ensure process isolation.

First, we need some definitions to express the constraints:

Definition: an *event* is the time point of the activity that caused new values to be asserted on output

lines of components.

Definition: $\text{value}(\text{Out}, e)$ is the value asserted on Out at event e

Definition: $\text{Domain}(\text{Out}, E) = \{ \text{value}(\text{out}, e) \mid e \in E \}$ where E is a set of events.

That is, $\text{Domain}(\text{Out}, E)$ is a set of all values that is asserted at the events in E

$\text{value}()$ and $\text{Domain}()$ are time series of values. Thus, we can compare all values generated over the lifetime of a process in both the base and RL augmented path. For convenience we introduce notation for the entire sequence of values produced by a process execution. If E_{proc} is a set of all events in the execution timeline of process **proc**, $\text{Domain}(\text{Out}, E_{\text{proc}})$ is set of all values that appear on **Out** during the execution of process **proc**.

Below are the general path constraints from the stricter to less strict order. Note that these are general constraints and only some of these may apply to some paths. The primed variables are from the RL augmented path, P' and non-primed variable are from safe, base path P . E_{proc} and E'_{proc} are a set of all events in P and P' respectively during the execution of process context, **proc**.

(i) Total Equivalence

$$E_{\text{proc}} = E'_{\text{proc}}$$

$$\text{and } \forall i, e \in E_{\text{proc}} (\text{value}(\text{do}_i, e) = \text{value}(\text{do}'_i, e) \text{ and } \text{value}(\text{co}_i, e) = \text{value}(\text{co}'_i, e))$$

This constraint ensures that when executing one process context, the sequence of values that appear on the data/address output lines and also control signals of the components are the same in the base path and the RL augmented path. This is the strictest requirement that also enforces functional equivalence of the components in the path.

(ii) Final Output Equivalence

$$E_{\text{proc}} = E'_{\text{proc}}$$

$$\text{and } \forall e \in E_{\text{proc}} (\text{value}(\text{do}_{\text{last}}, e) = \text{value}(\text{do}'_{\text{last}}, e) \text{ and } \text{value}(\text{co}_{\text{last}}, e) = \text{value}(\text{co}'_{\text{last}}, e))$$

where do_{last} , do'_{last} , co_{last} , and co'_{last} are from the last components of P and P'.

This less strict constraint requires only the output values of the last components match. The outputs of the last component serve as the output of the whole path. It is often the case that accesses to the entity of interest (e.g. the physical memory.) are made using these outputs. Then, this requirement imposes a constraint that the accesses made by the path should remain the same while the RL augmented path can be used as an “acceleration path”.

(iii) Final Output Subset

$$\text{Domain}(do'_{last}, E'_{proc}) \subseteq \text{Domain}(do_{last}, E_{proc})$$

Even if the outputs of the last component, which serves as the outputs of the whole path, are reordered and/or narrowed in their range, the process protection requirements could still be met for some paths.

(iv) Final Output Extension

$$\forall e \in E_{proc}, (\text{value}(do_{last}, e) = \text{value}(do'_{last}, e) \text{ and } \text{value}(co_{last}, e) = \text{value}(co'_{last}, e))$$

$$\text{and } \exists e' \in E'_{proc} \text{ and } e' \notin E_{proc} \Rightarrow \text{value}(do'_{last}, e') \in \text{EDO} \text{ and } \text{value}(co'_{last}, e') \in \text{ECO}$$

where EDO = set of allowable extended data output,

ECO = set of allowable extended control output.

This constraint states that if extra events are generated in path P', the final outputs must belong to a predefined allowable set of values. For some RL augmented path, it is more useful or sometimes even required to allow extra signals to be generated, e.g. interrupt signal from reconfigurable logic. Thus, we carefully predetermine a set of values that are allowable for these extra signals, not compromising the safety of the path.

4.3 Safety-critical Paths

As we pointed out, providing process protection in integrated reconfigurable processor means first of all preserving the hardware guarantees that the OS protection structure relies on. This subsection shows the path abstractions of those hardware guarantees and the preservation of the guarantees through path constraints.

4.3.1 Memory Access Path and Memory Space Isolation

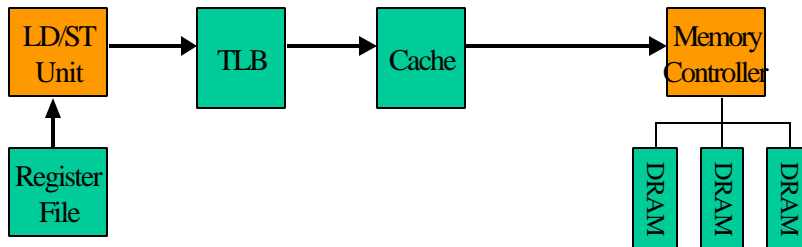


Figure 4. Example of a memory access path

The first step in reasoning about the process protection property of an integrated reconfigurable processor is to compare the processor state in terms of the physical memory state during the execution of processes in a non-reconfigurable, safe processor. In the memory access path, the head of the path is the component that is generating an effective address of the memory or memory mapped item. The end of the path is the component that accesses the physical DRAM chips with proper controls and addresses for DRAM, which is usually the memory controller. Any component that does appropriate address translation, caching, etc. can be placed in between those two endpoints. For example, following our path notation, the memory access path shown in figure 4 could be represented as:

$$\begin{aligned}
 P_{\text{memory_access}} &= \{ \text{LD/ST Unit, TLB, Cache, Memory Controller} \} \\
 &= \{ \langle \text{virtual_addr, (TLB_Lookup,R/W)} \rangle, \langle \text{virtual_addr,physical_addr}, \text{R/W} \rangle, \\
 &\quad \langle \text{physical_addr, R/W} \rangle, \langle \text{DRAM_addr, bank}, \text{R/W} \rangle \}
 \end{aligned}$$

Notice that for memory access path, we include only the outputs for addressing and control related to the memory address (i.e. virtual, physical, DRAM). The other elements and signals, though important for functional correctness, have no bearing on the protection.

As described in section 2, a memory access operation must preserve the memory space isolation property in order to maintain process protection. In other words, each physical address can only be accessed by the process that owns it at that time. If we define an **Owner()** function which maps a physical address to the process that owns it, the memory space isolation property for memory access path can be stated as follows:

$$\forall \text{addr} \in \text{domain}(\text{DO}_{\text{last}}, \text{T}_{\text{current_proc}}), \text{Owner}[\text{addr}] = \text{current_proc}$$

This means that the physical addresses asserted to the physical memory chips during the execution time of the current process all belong to the current process. The same must hold when the memory access path P' is reconfigurable :

$$\forall \text{addr}' \in \text{domain}(\text{DO}'_{\text{last}}, \text{T}_{\text{current_proc}}), \text{Owner}[\text{addr}'] = \text{current_proc}$$

Through proof by contradiction, it can be shown that the proposed constraints on RL augmented path do enforce the memory space isolation property above.

Theorem 4.1: For the memory access path, the constraints (i), (ii), or (iii) on the RL augmented path enforce memory space isolation.

Proof: Suppose that the memory space isolation property holds for the base path P but does not hold for RL augmented path P' ,

$$\forall \text{addr} \in \text{domain}(\text{DO}_{\text{last}}, \text{T}_{\text{current_proc}}), \text{Owner}[\text{addr}] = \text{current_proc}$$

$$\text{and not } (\forall \text{addr}' \in \text{domain}(\text{DO}'_{\text{last}}, \text{T}_{\text{current_proc}}), \text{Owner}[\text{addr}'] = \text{current_proc})$$

$$= \exists \text{addr}' \in \text{domain}(\text{DO}'_{\text{last}}, \text{T}_{\text{current_proc}}), \text{Owner}[\text{addr}'] \neq \text{current_proc}$$

In other words, there exists an address value asserted on the output line addr' of the RL augmented path P' that does not map to the address space owned by the current process. Then it follows that there exists addr' from P' that is not equal to some addr from P since all addr from P satisfies $\text{Owner}[\text{addr}] = \text{current_proc}$ by assumption. This contradicts with the constraints (i) and (ii), which

states that the final output values of the reconfigurable case(primed case) and base safe case must be equal,

$$\Rightarrow \exists (\text{addr}' \in \text{domain}(\text{DO}'_{\text{last}}, \text{T}_{\text{current_proc}}), \text{addr} \in \text{domain}(\text{DO}_{\text{last}}, \text{T}_{\text{current_proc}})), \text{addr}' \neq \text{addr}$$

contradicts with constraints (i) and (ii)

Furthermore, if there exists addr' of DO' and addr of DO that are not equal then the domains of the output lines DO' and DO are disjoint sets. This contradicts with the constraint (iii).

$$\Rightarrow \text{domain}(\text{DO}'_{\text{last}}, \text{T}_{\text{current_proc}}) \text{ and } \text{domain}(\text{DO}_{\text{last}}, \text{T}_{\text{current_proc}}) \text{ are disjoint sets}$$

$$\Rightarrow \text{domain}(\text{DO}'_{\text{last}}, \text{T}_{\text{current_proc}}) \not\subset \text{domain}(\text{DO}_{\text{last}}, \text{T}_{\text{current_proc}})$$

contradicts with constraint (iii)

Therefore, the proposed constraints (i), (ii), or (iii) preserve the memory space isolation property.

While we impose constraints on reconfigurable path in order to preserve memory space isolation, there is still a wide spectrum of possible memory access path adaptations. For example, even with the most strict constraint satisfied, we can have custom RL execution unit generating memory accesses as long as they are placed before the TLB and use virtual addresses. Also, the structure of the components such as the cache structure can be reconfigured if they are physically tagged. With the less strict constraint (iii), the custom memory access path can carry out remapping and narrowing of address space as in [20] or even a custom ‘preprocessing’ of data anywhere in the path from the memory to the register file. Another possible flexibility that is allowed by this constraint is that the order of memory operations can be changed in the RL augmented path. This might be useful for multiprocessor machines and for increasing ILP in superscalar machines, but the RL or the OS carries the burden of maintaining the desired memory semantic.

4.3.2 Interrupt/Trap Path

It is the control coprocessor's job to accept interrupt/trap signals and to jump to an appropriate handler. There are several components in the processor core that generate interrupt or trap signals to the control coprocessor as shown figure 5. These represent common interrupts/traps and the related components found in various implementations. Although the details of integrating interrupt/trap path into control logic of the processor may vary depending on implementations, the path abstraction sufficiently represents them as far as their delivery to the control logic is concerned.

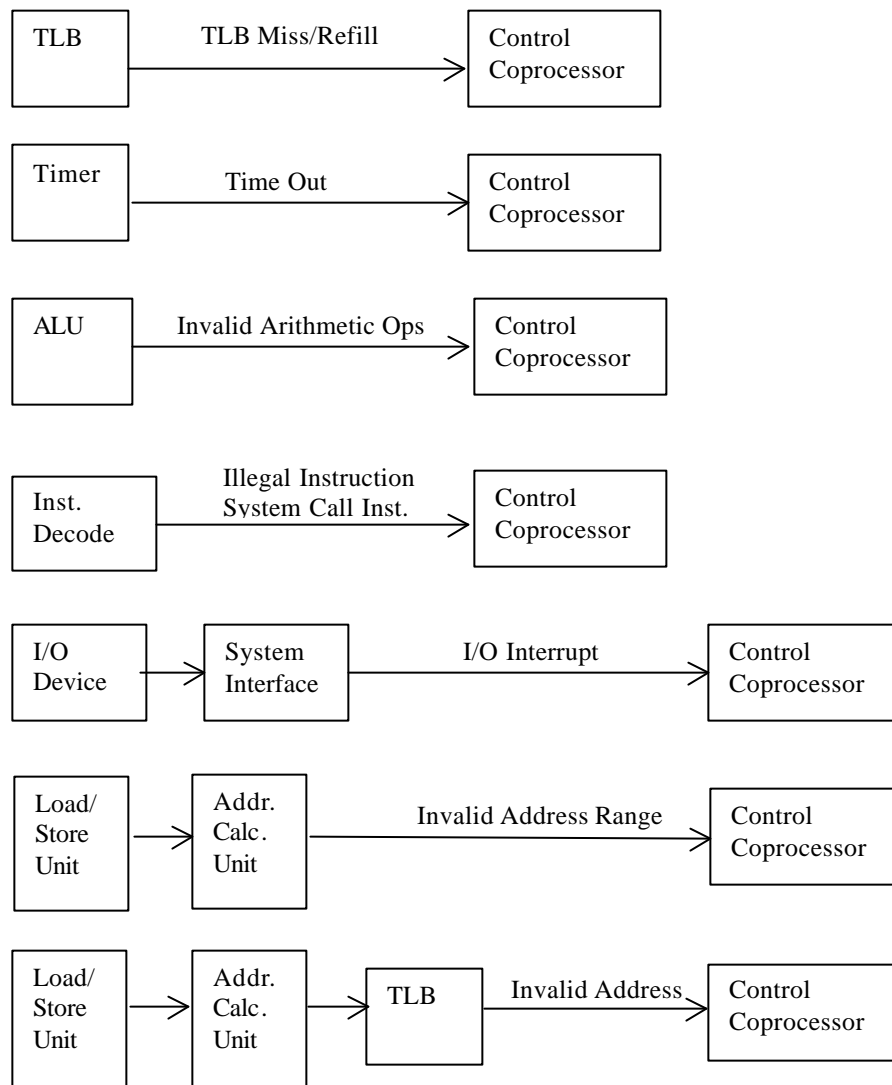


Figure 5: Various Interrupt/Trap Delivery Paths

For the safeness of the processor operation as well as correctness, it is critical that the interrupt/trap signals not be injected or diverted even when various component of the processor is augmented with reconfigurable logic. The non-injection and non-diversion requirement translates directly to the path

constraints that the sequences of interrupt/trap signals of base path and RL augmented path match for every interrupt/trap path. This is exactly what the constraint (i), Total Equivalence states. Also since it is the last component in the interrupt/trap path that actually delivers interrupt/trap signal to control coprocessor, constraint (ii), Final Output Equivalence satisfies non-injection and non-diversion properties. But constraints (i) and (ii) might be too restrictive for some reconfigurable paths that need to generate interrupt/trap signals specific to a configuration of the reconfigurable logic. For these paths, constraint (iv), Final Output Extension, allows safe extension of the required interrupt/trap signals.

Theorem 4.2: For interrupt/trap paths, the constraints (i), (ii), or (iv) ensure correct interrupt/trap delivery.

Any custom adaptation is possible on the components shown in figure 5 as long as they correctly deliver interrupts/traps as imposed by the constraints. For instance, a reconfigurable logic block can be used to build a custom instruction decode unit and a custom ALU to implement custom instruction and function. The proposed constraints allow such customizations as long as they generate and deliver correct interrupt/trap signals on exceptions such as execution of unprivileged instruction, execution of invalid operation, etc. Of course, if the components also belong to other critical paths, the custom adaptations must obey the constraints imposed from those paths, too.

4.3.3 Privileged State Access Path

Privileged states and data structures in the processor core such as control processor registers and the TLB contain safety-critical information and must be allowed access only by the OS. These accesses must remain unchanged even with active reconfigurable blocks present in the processor core. Altered accesses to these resources would indicate that sensitive protection data structures are compromised. Thus the reconfigurable path is required to observe constraints (i) or (ii) for these accesses.

Theorem 4.3: For privileged state access paths, the constraints (i) or (ii) preserve isolation of privileged state/data in the processor core.

4.3.4 Arbitrated Shared Resource Access Path

In order to build a multiprocess system with lockup freedom, we relied on a lockup-free resource management by the OS plus hardware arbitrators implementing a lockup-free protocol for certain hardware shared resource accesses (e.g. system bus) not visible to the OS. However, this access path can be replaced by a reconfigurable logic in an integrated reconfigurable processor. In such case, we need to verify that the application-customized arbitrator's protocol is lockup free. This requires a development of a protocol prover which is yet beyond the scope of this paper. However, we can trivially show that the path constraints (i) and (ii) are sufficient conditions of a lockup-free reconfigurable arbitration path.

Theorem 4.4: For arbitrated shared resource access paths, the constraints (i) or (ii) preserve lockup freedom.

Proof: From the assumption of a safe base path P , all of the events e generated by P are without lockup. Since E_{proc} and E'_{proc} , the set of all events in P and reconfigurable path P' during execution of $proc$ respectively, are same and have the same sequence by constraints (i) or (ii), we can see that the events generated by P' are also without lockup.

4.4 Constraints for Safe Context Switching

In the previous path examples, the proposed constraints preserve the properties required for process protection *during the execution of one process context*. That is, the safety is guaranteed by path constraints before and after a context switch but it is not enough to guarantee safe execution of context switching. The semantics of the context switching must not change if we want a safe context switching.

From the hardware point of view, we can observe that the following requirements should be met in an RL augmented processor to maintain safe context switching.

- **The trap/interrupt path to the control processor must satisfy the constraints (i),(ii), or (iv).**
- **Bypass/disable RL for LOAD/STORE operations**

The loading and saving of process context and hardware state is all done by generic LOAD/STORE or privileged LOAD/STORE, which means that the correctness of context switching relies on the functional correctness of memory operations. If we allow too much flexibility in an RL augmented memory operation path (such as in the least strict case above), the functional correctness of those operations are not guaranteed (although, memory space isolation is guaranteed). Thus for reconfigurable processors with such degree of flexibility, we need to bypass all RLs in the memory operation path. In our proposed model of integrated reconfigurable processor, a bypass/disable mechanism was already required to carry out synchronous hardware context switching, disabling reconfigurable logic blocks whose owner process was inactive. By executing context switching in the following order, reconfigurable logic can be bypass while doing LOAD/STORE.

1. Trap into context switch handler
2. Unschedule (disable) RL of previous process-- no RL enabled at this point.
3. Conventional context switching code using LOAD/STORE
4. Schedule (enable/swap in) RL for new active process
5. Trap out and relinquish control to new active process

4.5 Implementation of Constraints

There could be a couple of ways to enforce and/or check the proposed constraints. One way, as suggested for our Morph/AMRM protection architecture, is to have somewhat rigid design of the architecture so that critical components are strictly non-reconfigurable. For instance, the Total Equivalence constraint is satisfied for the memory access path in the proposed Morph/AMRM protection architecture, where the TLB is strictly non-configurable and is the only one that is allowed to do address translation.

Other more flexible approach would be to enforce it at the synthesis time of reconfigurable hardware or perform an online verification of reconfigurable hardware. For synthesis time verification/enforcement, *assertions* in hardware description language can be used to check that attached reconfigurable logic

preserves the values on critical path. (Example in VHDL below enforces Total Equivalence. Reconfigurable logic RL is attached at the output side of the component) Online verification can be performed by having a simple logic to compare against predefined set of outputs or outputs of simplified hardware mimicking the functionality of non-reconfigurable case. The online verification approach may be more useful for checking constraint (iii), the Final Output Subset constraint. For instance, constraint (iii) on the memory access path can be checked/enforced by simple duplicate hardware such as using a secondary TLB or a page-based check bits at the last stage.

```
entity RL is
    port(do, co:in BIT; do', co':out BIT);
end RL;

architecture RL_Assert of RL is
begin
    ...
    ...
    assert (do = do') and (co = co')
        report "May compromise system safety"
```

Figure 6: using assertions in VHDL to implement the constraint (i)

4.6 Path Abstraction vs. Real Processor Model

We have defined a path as an ordered and directed sequence, where the control and data flow oneway downstream from the head component to the last component in the path. Although this abstraction would not represent the precise register-transfer level model of the processor, it is possible and sufficient to extract one-way stream path for each operation of interest. For instance, we can extract an abstract path for a STORE operation with the head component being the LOAD/STORE unit that generates an effective virtual address and the last component being the memory controller generating actual access address and control signals for DRAM chips.

5 Putting it Together: Morph/AMRM Protection Architecture

The Morph/AMRM reconfigurable processor project is an ongoing effort to build an integrated reconfigurable processor. Following the basic concept of integrated reconfigurable processor outlined in

Section 3, the proposed Morph/AMRM processor integrates reconfigurable logic 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. This enables a highly flexible architecture but also makes virtually every part of the processor have reconfigurability and makes guaranteeing process protection harder.

To preserve process protection guarantees in the Morph/AMRM processor, it is designed to 1. Satisfy the path constraints in all protection related critical paths, 2. Provide mechanism for synchronous hardware context switch. In our initial design of the Morph/AMRM protection architecture, satisfying path constraints is achieved by rigid design rather than additional mechanisms such as online validation logic. That is, by assigning a few critical components to be strictly non-reconfigurable we are able to preserve process protection properties while enabling much of the useful reconfigurability. Some of these useful adaptations are custom 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.

5.1 Satisfying Path Constraints by Design

5.2.1 Memory Access Path

The basic idea for maintaining process protection in reconfigurable memory path is to make the TLB strictly non-reconfigurable and to control all accesses to memory or memory mapped item by checking/translating address in the hardwired TLB. Any reconfigurable component that accesses the system bus (for memory or other memory-mapped items) can be fully reconfigurable as long as they generate virtual addresses which is then checked by the hardwired TLB. Other components in the memory access path such as the cache (as long as they are physically tagged) can have fully reconfigurable policies and structures. The memory controller can be reconfigurable as long as they do not reshuffle the

addresses – we can check this by adding a secondary TLB (or even simpler page-based check bits) after the memory controller.

The rationale behind this design is that it will satisfy the Final Output Equivalence constraint and the actual accesses to the memory(main memory or physically-tagged caches) will be the same as if we had non-reconfigurable, safe processor. This is because the final output, which is in the form of physical addresses, must be checked and translated by a non-reconfigurable TLB whenever any component wants to access memory.

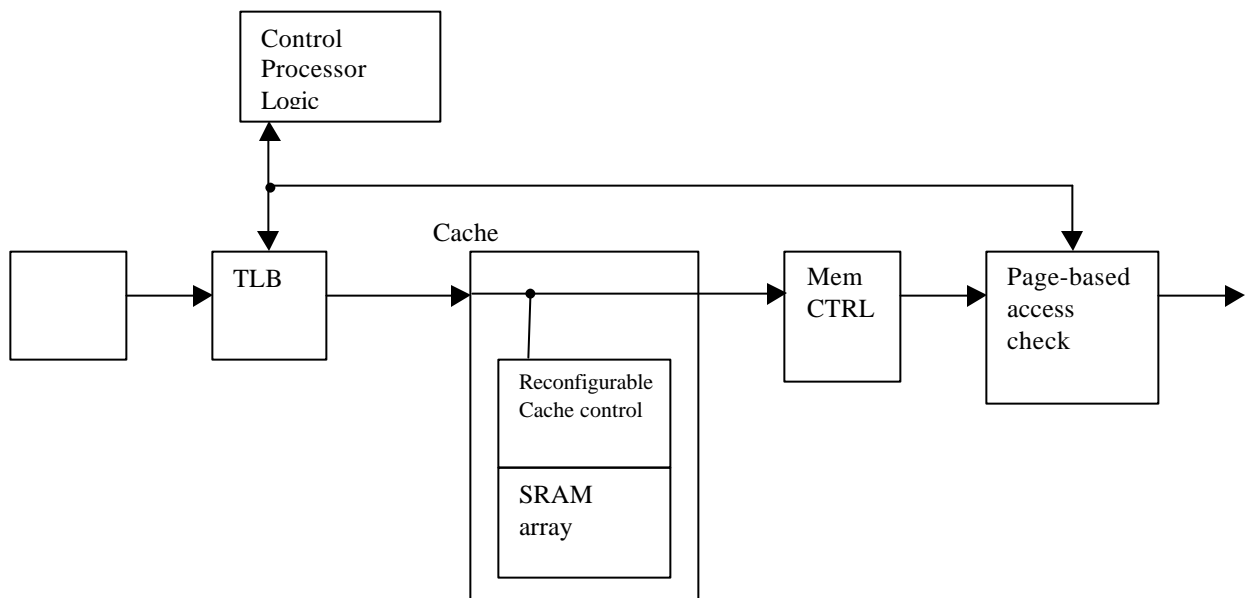


Figure 7 : memory access path design in Morph/AMRM. Notice that the controls to/fro the TLBs are strictly hardwired to the control processor logic. Shaded boxes are hardwired components.

5.2.2 Trap/Interrupt Path & Privileged State Access Path

In the general RISC processor organization, the control coprocessor(also known as CP0 in MIPS terminology) logic is central in providing mechanisms such as privileged/user mode transitions and exception/interrupt delivering/handling for OS mediation, which are required to guarantee process isolation. Thus by requiring that the control coprocessor logic and interface be strictly non-reconfigurable, we can avoid injection/diversion of trap/interrupt signals and compromise of privileged data structures. In addition to requiring that the interface between other components and CP0 be non-reconfigurable, the

following fault detection logic are hardwired to implement safe trap/interrupt and privileged state access paths.

- TLB: Address check, TLB miss/refill
- System Interface: I/O interrupt, reset, failure
- Instruction Decode: Illegal Inst. System Call Inst.

5.2.3 Arbitrated Shared Resource Access Path

In the current Morph/AMRM protection architecture design, all the arbitration logic for non-visible hardware shared resources are hardwired and implement a lockup-free protocol.

5.3 Support for Synchronous Hardware Context Switch

The support for synchronous hardware context switching is based on architectural extension suggested in Section 3.

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 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, corresponding reconfigured block will be selected and activated while other reconfigurable blocks not used by the process will be bypassed.

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 corresponding bit in the **Configuration Swap Register** is set. The **Configuration Swap Register** is checked in the context switch routine and new configuration is swapped in if necessary.

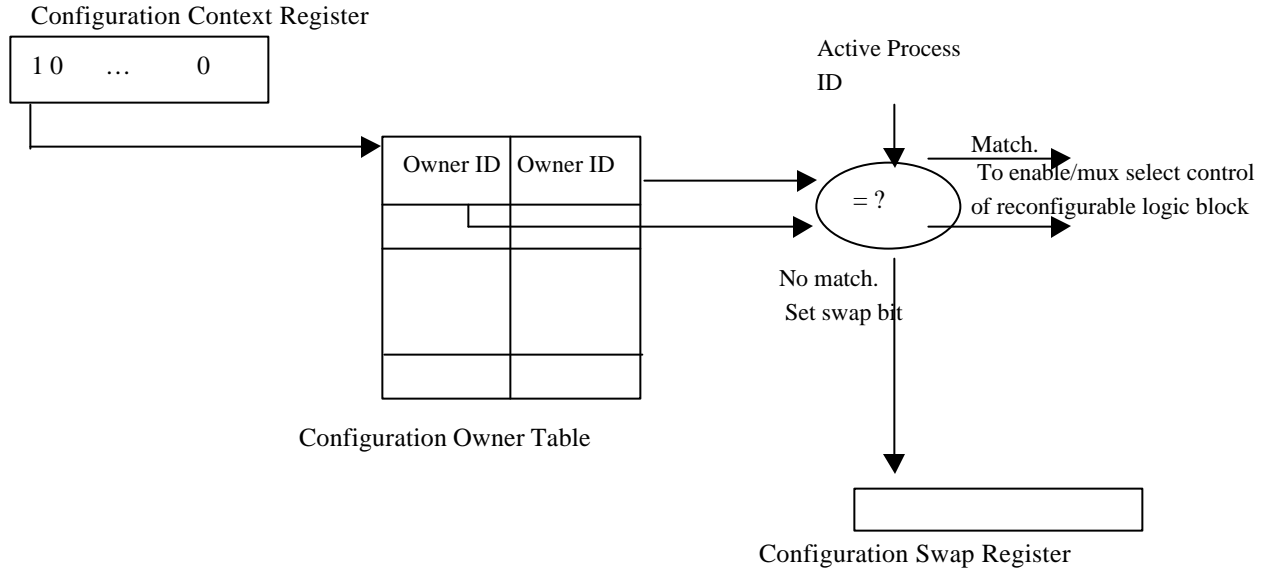


figure 8: Synchronous Hardware Context Switching

6 Discussions and Related Work

6.1 Flexibility of Multiprocess Safe Reconfigurable Processor

Our solution to providing multiprocess safety in integrated reconfigurable processors was by constraining the reconfigurability of components on safety-critical paths in the processor, trading off reconfigurability with safety to some extent. The more of the safety-critical paths we cover, the less the flexibility and the more the multiprocess safety guarantees. The flexibility of integrated reconfigurable processors also depends on how strict the constraints are and on how the enforcement of those constraints is implemented. All of these three factors determine the allowable reconfigurability of an integrated reconfigurable processor. However, the multiprocess safety is the highest order determinant of the allowable flexibility. In other words, the reconfigurability is first determined by the degree of safety that

we wish to provide and then by the strictness of the constraints and again by the implementation of the constraints. These tradeoffs are analyzed below:

1. Reconfigurability and Safety

If we cover all the multiprocess safety-critical paths in the reconfigurable hardware that are discussed in the main sections of the paper, i.e. all paths that has access to memory, all paths that has access to privileged processor state, all critical interrupt/trap paths, all paths to H/W arbitrated shared resources, the system will support safe multiprocess environment dictated by the OS protection semantics. However, putting constraints on all of those paths may be deemed too restrictive for some design goals. For example, a reconfigurable logic block implementing a custom DSP computation unit may forgo any hardware address translation and checking required for preserving the address space isolation property. It can be traded off with a more flexible and direct access to memory for larger bandwidth suitable for stream-based computations. Generally, it is possible to think of a spectrum of reconfigurable architectures that spans a range of flexibility and safety as classified below:

1. **Full Reconfigurability:** All processor components, including those on multiprocess safety-critical paths, are fully reconfigurable. This allows the maximum flexibility but provides no hardware mechanism to guarantee multiprocess protection.
2. **Traditional Coprocessor Reconfigurability:** Coprocessor devices are fully reconfigurable and their safety-critical paths are unchecked, e.g. reconfigurable coprocessor has direct access to memory and the system bus. Main processor is usually non-reconfigurable and multiprocess-safe, but the overall system is not. This approach is typically taken for FPGA-based coprocessor configurable designs.
3. **Reconfigurability with safety:** All processor components are reconfigurable but all memory access paths, all privileged register access paths, and all interrupt/trap paths are constrained. Other arbitrated access paths to shared resources such as busses and registers may not be constrained. This results in a system with process isolation guarantee but no guarantee of lockup freedom.

4. **Reconfigurability with full multiprocess support:** All memory access paths, all privileged register access paths, all interrupt/trap paths, and all other shared resources access paths satisfies appropriate path constraints for multiprocess safety. This is most restrictive but guarantees process protection and lockup freedom.

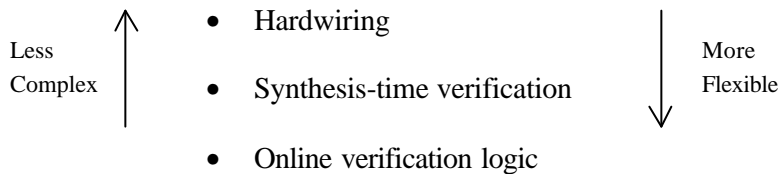
This classification lists the representative points in the wide spectrum of architectural classes that vary in their capabilities for customization to enhance application performance and the cost of providing multiprocess isolation guarantees. Architecture type #3 and #4 provides multiprocess safety guarantees while still allowing what one might consider to be a broad notion of useful reconfigurability, as discussed in the reconfigurability of Morph/AMRM processor. While Architecture type #3 provides a narrower notion of multiprocess isolation, architecture type #4 fully supports multiprocess environment by adding the guarantee that no process will arbitrarily lockup shared resources and thus lockup the whole system.

2. Reconfigurability and Constraints

The general constraints on paths with varying degree of strictness was presented in Section 4. Although all of these constraints guarantee process protection properties for the given paths, they provide different degrees of flexibility and functional correctness guarantees. The less strict the constraints are the more flexible the configuration of the reconfigurable path can be. However, less strict constraints provide weaker guarantee of the semantics of the operation being the same as the non-reconfigurable path. The burden of implementing desired semantics is then delegated to the reconfigurable logic and the program, whereas if the most strict constraint (i.e. total equivalence) is satisfied the reconfigurable logic is invisible to the program. That is, the different degrees of strictness in the constraints correspond to different balance points in the trade-off between flexibility vs. software complexity for maintaining correct semantics.

3. Reconfigurability and Constraint Enforcement

We have discussed the possible implementations for enforcement of constraints. Among them, hardwiring critical components (as in the initial Morph/AMRM design) is most rigid yet may take the least effort. The synthesis time verification of reconfigurable logic is more flexible but requires translating appropriate constraints into well-formulated specifications and the development of an automatic verification tool. Of course, this method could take much less effort once such translation and verification tools are built into the compiler for integrated reconfigurable systems. Lastly, the addition of online verification logic allows the most flexibility among the proposed implementation methods but also incurs more cost and complexity in terms of design and silicon.



6.2 Related Work and Multiprocess Safety

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 [13,14,15]. 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 fixed model for managing memory or external devices for these processors, which suggests that it should be classified as type #1 architecture above. 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.

To overcome these shortcomings that make full FPGA processors less than ideal for general-purpose computing, architectures that couple a general-purpose control processor with FPGA co-processors have been proposed [16]. FPGA is placed as a slave computational unit and is used to speed up what it can, while the main processor controls the whole execution and takes care of other computations. This falls in the architecture type #2. In [16], co-processor has its own memory interface and control logic, so it compromises multiprocess safety unless accesses through those components are checked to satisfy constraints for safety.

Reconfigurable processors with dynamic instruction sets [17] try to extend the application-specific computation capability of a general processor with a computational unit implemented in reconfigurable logic. 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 a hardwired 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 #4 (Reconfigurability w/ full multiprocess support) architecture. But the reconfigurability 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.

Recently proposed reconfigurable processors such as [18,19] as well as [4,5] tightly integrate reconfigurable logic block into existing processor core. These are what we defined as integrated reconfigurable processors. In [18], FPGA resources and memory are integrated into the processor's datapath to allow custom computation on FPGA. The FPGA component has fixed memory interface that maintains memory consistency and shares the fixed memory controller with other processor component. The fixed memory interface and shared fixed memory controller can provide protection guarantees of

architecture type #3 or type #4 but again has limited flexibility. Its reconfigurability is limited to providing custom computational unit that can share resources and work in parallel with the rest of the processor.

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 [20]. 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.

7 Summary and Future Work

The fundamental concept in an application-adaptive integrated reconfigurable processors is that the processor mechanisms, structures, and policies can be optimally adapted to each application by allowing arbitrary reconfigurability, which can compromise multiprocess safety. 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. 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. To formulate the constraints that may be imposed on reconfigurable logic to preserve those mechanisms, we introduced the concept of paths. The key hardware requirements for process protection can be preserved by enforcing the formal constraints on safety-critical paths, identified in the paper as memory access paths, interrupt/trap paths, privileged state access paths, and arbitrated shared resource access paths. We have also discussed the possible implementation for enforcing the constraints in reconfigurable processors, namely key component hardwiring, synthesis time verification, and online verification. In the near future, we are planning on developing an automatic HDL assertion generator from constraint specification and an automatic verification tool. This development will allow a synthesis time verification of path constraints and will also enable more flexible design.

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 13th 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] Mirsky, E. and DeHon, A. MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources, In *Proceedings Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, 1996, pp157-166
- [8] Waingold, E. et al, Baring it all to Software: The Raw Machine, In *IEEE Computer*, Sep 1997, pp86-93.
- [9] Goldstein, S. et al, PipeRench: A Coprocessor for Streaming Multimedia Acceleration, In *Proceedings of the 26th International Symposium on Computer Architecture (ISCA)*, May 1999, pp28-39

- [10] MIPS technologies, Inc. *MIPS R10000 Microprocessor User's Manual*, 1996.
- [11] Bach, M., *The Design of the UNIX Operating System*, Prentice-Hall, Englewood Cliffs, NJ, 1986.
- [12] Leffler, S., McKusick, M., Karels, M., Quarterman, J. *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley, Reading, MA, 1989.
- [13] Gokhale, M., Holmes, W., Kasper, 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.
- [14] Arnold, J., Buell, D., and Davis, E., Splash 2. In Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 316-324, June 1992.
- [15] 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.
- [16] 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.
- [17] 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).
- [18] Jacob, J. and Chow, P., Memory Interfacing and Instruction Specification for Reconfigurable Processors, In *Proceedings of ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '99)*, 1999, pp145-154.
- [19] Rupp, C. et al, The NAPA Adaptive Processing Architecture, *In Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '98)*, Apr 1998.
- [20] 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)*