University of Paderborn — Faculty EIM-I
Computer Engineering Group

Formally Verified Memory Access Monitors in Reconfigurable High-Performance Computing Systems

Master's Thesis
at the Department of Computer Engineering
Prof. Dr. M. Platzner

Department of Computer Science
Faculty for Electrical Engineering,
Computer Science and Mathematics
Paderborn University

presented by
Shoukath Ali Mohammad
on
21.12.2018

Supervisor:	Tobias Wiersema, M.Sc.
Examiners:	Prof. Dr. Sybille Hellebrand
	Prof. Dr. M. Platzner

Shoukath Ali Mohammad
Matriculation number: 6819945
AM Rippinger Weg 1
33098

Abstract

Heterogeneous systems often output high energy efficiency by adapting to needs of high performance and low power consumption for a given computational problem compared to conventional general-purpose computing units, by exploiting the heterogeneity in the system. One such heterogeneous systems are the High-Performance Computing (HPC) systems. HPC systems are aggressive machines which perform high-power computations. Re-configurable platforms on these HPC systems allow for custom hardware-accelerated computations drive with a low power input which is advantageous to a HPC system. These systems have shared resources, which are prone to security risks. One such shared resource, for instance, is globally shared memory among several nodes of a cluster in a HPC system. Only when all computing resources can access this memory, the system can perform at its full potential. When reconfigurable hardware are given direct access to the shared memory, the security risk of the system increases as a corrupt logic on the hardware can take down the entire system. Intellectual Property (IP) cores used in reconfigurable hardware such as FPGAs are designed by third parties which operate at different trust levels. To overcome this problem, the shared memory is isolated from the re-configurable hardware by introducing a run-time memory access monitor, monitoring all requests sent from hardware to access memory by enforcing a memory access policy. Formal verification of these monitors in such a dynamic system is challenging. Proof-Carrying Hardware (PCH), a formal verification method can be used to formally verify the access monitors PCH uses certificates as a proof of verification. A prototypical image processing system with meaningful memory access policies to show the feasibility of the approach.

Contents

List of Figures

List of Tables

1 Introduction

Heterogeneous systems in today's world have found their way into many disciplines. These systems are known for their high energy efficiency, high performance and low power consumption. HPC systems are one among these, which are superior to general purpose computing systems in handling high-power computing. A HPC system can be a cluster of several computers or can have specially built hardware to form a powerful machine. HPC systems may have shared resources at different levels like globally shared resources and local shared resources.

HPC systems with reconfigurable hardware get the benefits of both high-performance computing and fast, adaptive hardware accelerators. HPC systems use reconfigurable hardware like FPGAs to accelerate their computations. FPGAs are known for their massively parallel processing capabilities and low power consumption. They are configurable on demand for computational needs.

An FPGA uses Intellectual Properties (IPs) often delivered by third parties. IPs can be modified by a third party to take over the shared resources, undermining the security of the whole system. A monitoring scheme is introduced in the FPGAs of HPC systems to avoid such a scenario. The shared resources are isolated from the IPs on the FPGA using a memory access monitor. An access monitor monitors all the accesses to the shared resources by the IPs and takes action on suspicious accesses using memory access policy. The access monitor is formally verified using PCH, a formal verification technique to prove that it is trustworthy. PCH uses a proof in the form of certificate for the correctness of the IPs designed by a producer. The consumer can use the certificate to verify the trueness of the IP received.

1.1 Goals

- The first goal of this master's project is to research memory security in heterogeneous re-configurable systems, with a focus on the security of accesses to the shared memory.

- The second goal of the thesis is to survey the available HPC systems at the Paderborn University and to select a suitable environment for the prototype and shared resource to protect.

- The third goal of the thesis is to create a prototypical heterogeneous image processing system with a relevant shared resource access policy.

- Finally, the fourth goal of the project is to develop a verification/certification flow using Proof-Carrying Hardware to guarantee the accordance of a re-configurable heterogeneous HPC system to a shared resource access policy.

1.2 Approach

The approach of this thesis for accomplishing the goals described previously begins with a literature survey about memory security in the field of reconfigurable hardware. A survey of the research done so far on memory access monitoring on a reconfigurable system is conducted. Then a suitable HPC system is chosen for the monitoring approach. Later the need of formally verifying access monitors using PCH is described followed by the tool flow.

1.3 Thesis Outline

The rest of the thesis is as follows.

- **Chapter 2** Describes the research done so far on the memory security on reconfigurable hardware and PCH.
- **Chapter 3** Mentions all HPC systems available in Paderborn University along with all the parameters considered in selecting a suitable platform for the prototype.
- **Chapter 4** Explains the different possibilities of achieving memory security using memory access monitoring in a HPC system.
- **Chapter 5** Describes the implementation of a few example designs with the monitor protecting a shared memory.
- **Chapter 6** Gives an illustration of PCH verification tool flow to formally verify access monitors for HPC systems.
- **Chapter 7** Concludes the thesis with a review of the approach and its feasibility along with future improvements in this approach.

2 Background Information

This chapter gives details of related works on memory security on Reconfigurable Hardware, formal verification flow and basic information on AXI communication protocols.

Reconfigurable computing grew into a major part of many industries like defence, aerospace, automotive, medical, communications and many more. All these industries depend on Field Programmable Gate Arrays (FPGAs) to perform their respective computations at much lower costs compared to custom fabrications (e.g., ASIC). The logic on these FPGAs can be changed in the field during run-time. These advantages escalated their usage in critical systems. With the increase in usage of FPGAs, the need for securing it has also increased. In recent years, security in reconfigurable hardware has been an active research topic. However, in many recent projects, FPGAs were used for prototyping with a focus on security in multi-core processors from external and internal attacks. For example, Fiorin et al.[9] considered data protection techniques in a NoC based MPSoc by enforcing a protection architecture within a BRAM controller which monitors all the accesses based on the access rights stored in the lookup tables on an FPGA. Diguet et al.[6] also proposed a hierarchical security policy in protecting a reconfigurable NoC based SoC from external attacks. Basile et al.[2] considered software code execution on a trusted FPGA, in an untrusted embedded environment. The authors proposed a monitoring protocol to safely deliver the code to the desired target. According to the author, the dynamic update of the reconfigurable hardware reduces the risk of bitstream reversal. Cotret et al.[4] proposed bus-based security services for a multi-core system especially targeting the AXI based systems protecting the data exchange within the bus. Eckert et al.[8] introduced a watchdog to monitor the data of a memory bus and detect suspicious data. The watchdog concept can be adapted to shared buses on reconfigurable hardware. Crenne et al.[5] proposed a security core on an FPGA to securely load an application from an external interface to the main memory of a system and securely execute it. Zhang et al.[22] considered a vendor market perspective and surveyed various state-of-the-art defence mechanisms and trust issues of a reconfigurable system. Trimberger et al.[19] presented various security primitives adapted by the vendors in the latest FPGAs. Huffmire et al. [12] introduced a monitoring based approach using access policies to safeguard memory access in reconfigurable hardware. Their work can be adapted to memory access monitoring in a HPC context.

2.1 Huffmire's Work on Memory Access Monitoring

According to Huffmire et al., on a reconfigurable system, memory is always a primary source of concern. Any shared memory connected to or within the reconfigurable hardware is at a security risk. They introduced a memory access monitoring approach to protect the memory from malicious logic on reconfigurable hardware. In their approach to memory protection, they isolated the shared memory from all the modules accessing

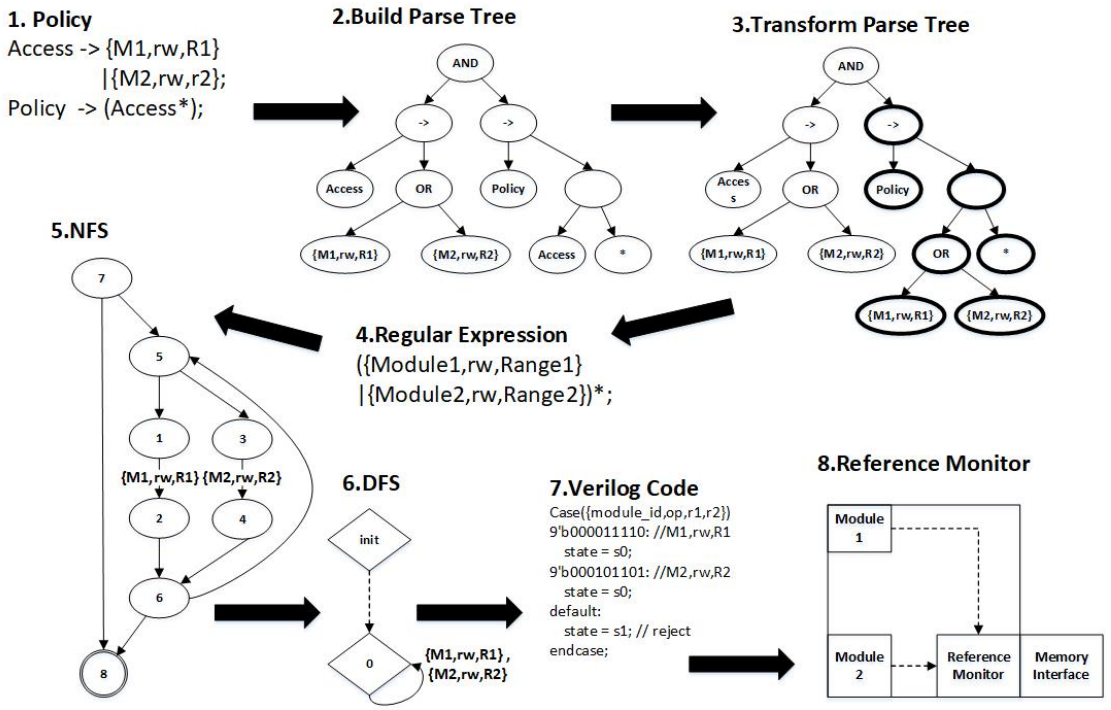


Figure 2.1: Policy compiler tool flow of a simple Compartmentalization policy[10].

the shared memory by placing a Reference Monitor (RM) between them. All other modules/cores have to route their memory accesses via the RM. They have a memory access policy according to which the cores are either granted or denied an access request. The memory access policies are user definable. Huffmire et al. introduced a formal language and a policy compiler to specify a memory access policy and generate them. Figure 2.1 depicts the tool flow of the policy compiler from a policy description of a simple compartmentalization policy to memory monitor implementation.

The policy generated from the policy compiler is a state machine which defines the memory accesses it allows. Each access is specified by a module id, an access type and a range. Module id denotes the cores/modules which requests the memory access with an access type, e.g., read, write, scrub. The range is a segment of memory which it wants to access. The outputs of the state machine are '1' and '0' to grant or deny access. The tool flow processes the range into a form of circuit which can perform a parallel search through all the addresses in the policy and decide if the requested address belongs to a specified range.

Huffmire et al. [12],[13] described various memory access policies in their formal language. In [12] Huffmire et al. provided two architectures for the placement of memory monitors as seen in Figure 2.2. To the left of the figure, the RM is placed between the shared bus and the shared memory. The RM monitors all the accesses to the shared memory. A core may be transferring sensitive data via the shared bus. An arbiter is used to make sure only one core gets access to the shared bus at a given time. Both the read and write accesses have to wait until the RM verifies the access legality. To the right of Figure 2.2, a buffer is placed between the shared bus and the shared memory. All the read accesses are allowed to reach the shared memory. The read access is stored in the buffer

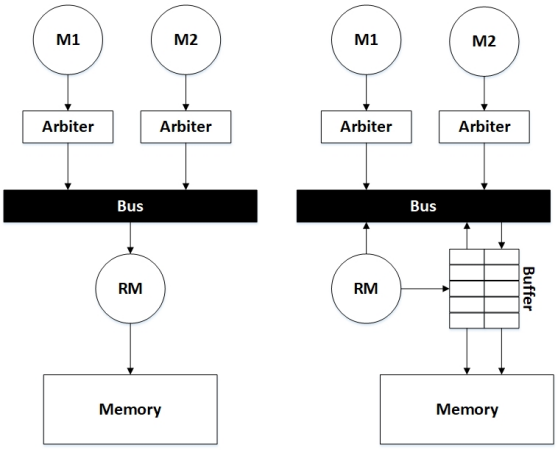


Figure 2.2: Alternative architectures of monitor mechanism[12].

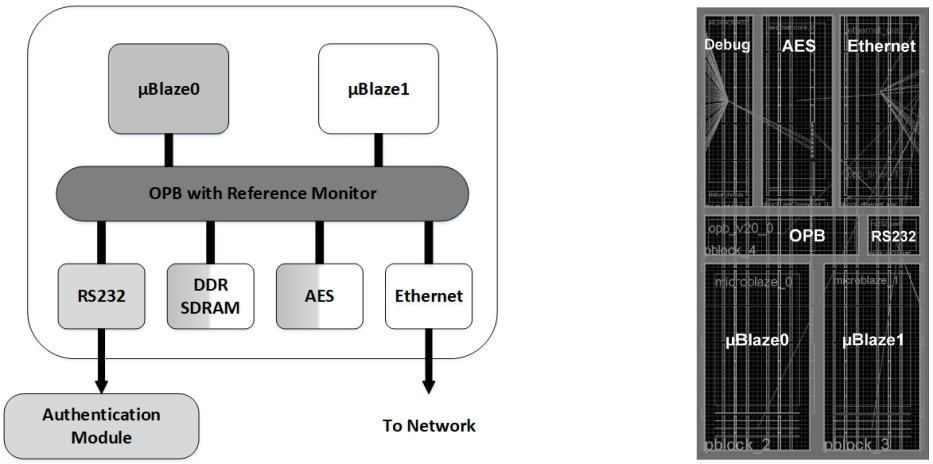
and RM parallelly checks its legality. The response from the shared memory reaches the core only if the access is legal. This architecture reduces the latency of the read access. The write access will reach the shared memory after the RM checks its legality. In [11] Huffmire et al. introduced physical isolation of the memory monitor at placement and route level to avoid any tampering with the monitors using Moats and Drawbridges. The moat size has a significant effect on the system performance and area overhead. Increase in moat size increases area overhead and decreases the performance of the system.

To verify the feasibility of the approach, Huffmire et al. [10] applied the concept of an access policy, moats and drawbridges and the reference monitor to a real-world system. A Red black system (Figure 2.3) set up on a single FPGA, was used for the prototype. Two μBlaze processors were used in the prototype One processor is for the red and the other for the black domain with an on-chip peripheral bus as the shared bus and various bus peripherals shared between the processors. The RM with a custom stateful policy was snooped onto the shared bus to monitor all the accesses to the memory using the shared bus.

From the evaluation of the system, the effect of physical isolation of monitor on area overhead increased with a decrease in performance for an increase in moat size. The authors concluded that the impact of the RM on the area overhead and the performance of the system was minimal.

2.2 Proof-Carrying Hardware

Necula[18] introduced the principle of Proof-Carrying Code (PCC) and further researched by Necula and Lee[17]. PCC was used as a defence mechanism against the malicious code received from untrusted sources. The producer and the consumer share the PCC tool flow. First, the code consumer sends a design specification to a code producer. The producer then generates the code along with the proof of code correctness. The code and the proof are submitted to the consumer. The consumer then verifies the proof and uses the code. The concepts of PCC were introduced to the hardware platform by Drzevitzky et al.[7] as Proof-Carrying Hardware (PCH) (Figure 2.4) using SAT-based technology. PCH was used to verify the correctness of a circuit implementation in the form of a proof/certificate provided by the producer to the consumer. PCH mainly



(a) Block diagram of the Red-Black system[10] (b) Floor planning view of Red-Black system[10]

Figure 2.3: A Red-Black system[10].

targeted FPGAs where a consumer keeps installing new modules and cannot afford long run-time in verifying the design. The tool flow provided by the author does not generate bitstreams for commercially available FPGAs. So, the work cannot be applied to the real world.

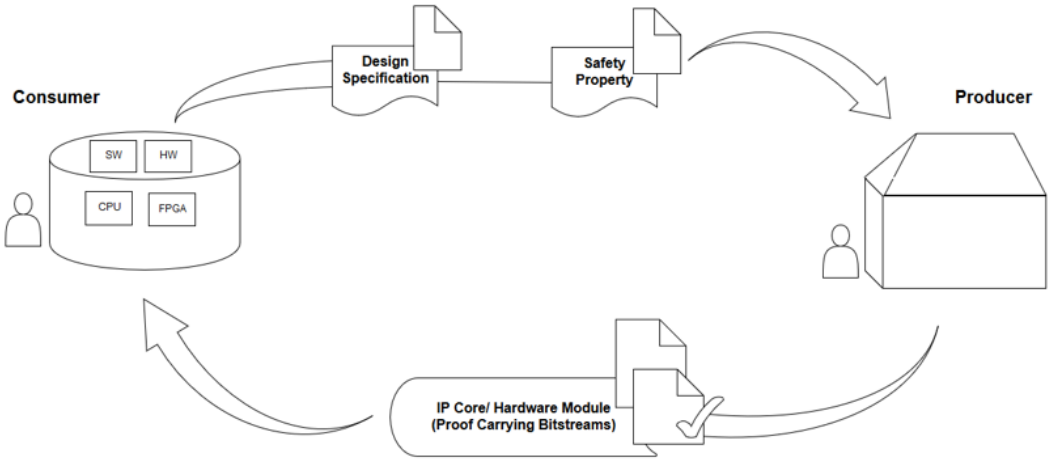


Figure 2.4: Proof-Carrying Hardware (PCH) basic flow

Wiersema et al.[20] combined the concepts of memory monitors and SAT-based PCH verification on an FPGA. They prototype a memory monitor on a virtual FPGA overlay (ZUMA) in a Zynq system. Their tool flow generated the bitstream for ZUMA. They used bounded sequential equivalence checker to unroll sequential circuits into combinational circuits. Wiersema et al. showed that the circuits lead to a large number of combinational circuits followed by large runtimes for verifying the proof. Later Isenberg et al.[16] introduced Induction Based PCH technology and compared it against SAT-based. PCH technology in various parameters considering both producer and consumer tool flow. They used IC3 a SAT-based algorithm which computes the inductive strengthening using

large number of small unsatisfiable queries. Dynamic policies lead to sequential Monitor circuits and Induction based PCH has advantages over SAT-based PCH for sequential circuits. The inductive strengthening when verified yields three small SAT problems which are solved by the consumer in relatively low time compared to the bounded unrolling method.

Understanding AXI4 protocols are essential to grasp the concept behind this Thesis. So we provide a brief introduction to AXI4 protocols.

2.3 AMBA AXI4 Protocols

The 'AXI' is a protocol used for point to point communication between different blocks of a Reconfigurable system. AXI4 interface is a more commonly used bus interface as it follows AMBA AXI protocols which is an industrial standard. Using the AXI4 interface to check the feasibility of the monitoring approach would give the advantage of the architecture is generic to any HPC system which has an AXI4 interface to access the shared memory.

The Advanced eXtensible Interface (AXI4) bus protocol was developed for the ARM Advanced Micro-controller Bus Architecture (AMBA)[21]. To use the AXI4 bus interface the bus peripherals should follow the AXI protocols. This can be achieved by configuring the user logic into AXI peripherals which are available in the Xilinx Vivado Design suite HLx used for this design.

"There are three types of AXI4 interfaces:

- AXI4-Full: For high-performance memory-mapped requirements.
- AXI4-Lite: For simple, low-throughput memory mapped communication.
- AXI4-Stream: For high-speed streaming data."[21]

The AXI4-Full specification provides various options like variable data width and address bus width for high bandwidth burst operations. It provides ID support to track the memory access requests. AXI4-Lite specification is similar to AXI4-Full but with no burst support and ID support. AXI4-Stream specification supports high-speed data transfer with no memory mapping. So, AXI4-Stream has no address support.

In this thesis, the monitoring approach uses the access policy generated by the policy compiler of Huffmire et al.[12]. The generated policy requires module ID and address as inputs and only AXI4-Full specification provides these inputs. So, in the next section AXI4-Full protocols are described briefly.

2.3.1 AXI4-Full Protocols

The AXI specification is always described between an AXI Master and an AXI Slave as seen in Figure 2.5. Any module which reads or writes data is an AXI Master and any module which responds to the read or write request should be an AXI Slave. They are connected using an AXI Interconnect block.

AXI4-Full has five different channels between the Master and the Slave as seen in Figure 2.6. All the AXI signals which should be understood for this thesis are explained in the next subsection.

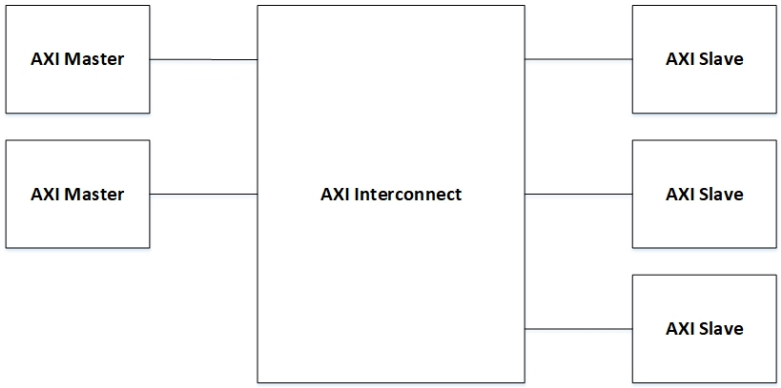


Figure 2.5: AXI Bus Interconnect [21]

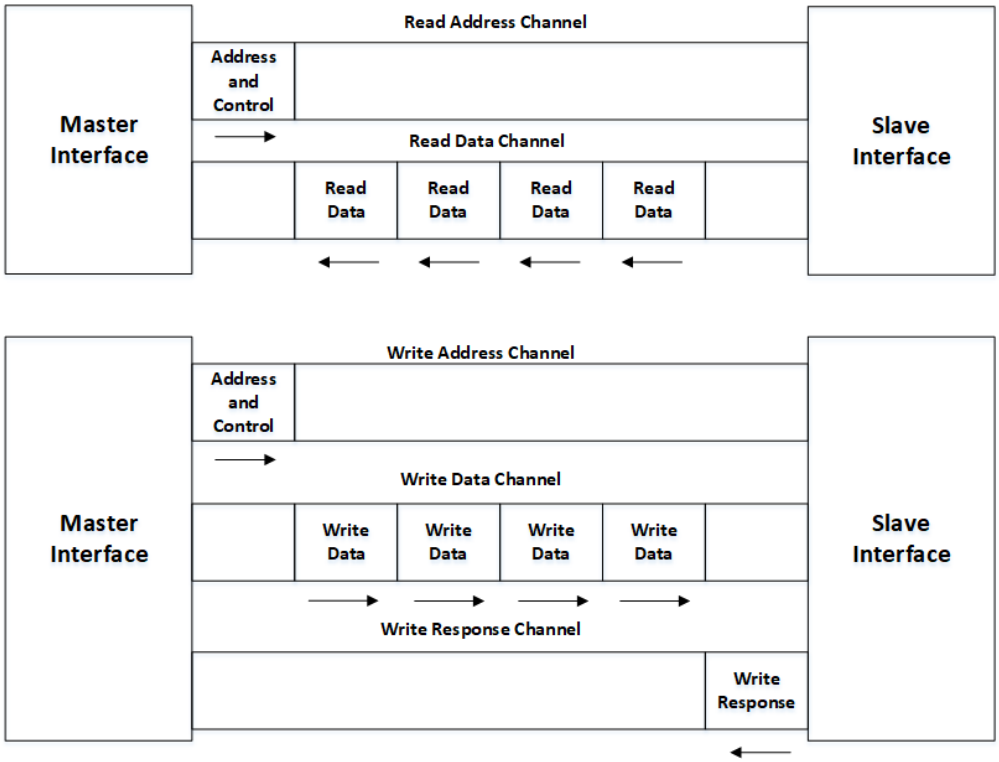


Figure 2.6: AXI-Full Interface [21]

2.3.2 Communication Between AXI Full Master and AXI Full Slave

All the five channels in Figure 2.6 have specific ports to communicate. Only the ports and their respective communications necessary to understand in the design of monitor are explained.

In a read state, AXI Master signal *AXI_ARVALID* goes high which means that it is ready to send the address to the AXI Slave. In response to it AXI Slave signal *AXI_ARREADY* goes high which means slave is ready to receive the address. When both the signals go high AXI Master sends the read address and the ID to the slave via *AXI_ARADDR* and *AXI_ARID*. After a successful transaction both *AXI_ARREADY*

and *AXI_ARVALID* go low. When the address is received by the slave signal *AXI_RVALID* goes high which means now the slave is ready to send the data in response to the address it received. AXI Master signal *AXI_RREADY* is set to high in response to *AXI_RVALID*. When both the signals go high the data is transferred through *AXI_RDATA* to the master along with the id via *AXI_RID* which can be used to keep track of the accesses made. When the last bit of the data is sent to the master *AXI_RLAST* is set to high by the slave and after a successful data transfer *AXI_RVALID*, *AXI_RREADY* and *AXI_RLAST* go low. When both the address and data transactions are successful AXI Master goes to write state.

In write state, first the AXI Master sets *AXI_AWVALID* high to let AXI slave know that it is ready to send the address and in response to it when AXI Slave sets the signal *AXI_AWREADY* master sends the write address and the ID via *AXI_AWADDR* and *AXI_AWID* and both the signals which were set to high during the address transfer go low. After sending the address the master set the signal *AXI_WVALID* to high to make the slave ready to receive the data. Now the slave responds by setting *AXI_WREADY* high to receive the data. When both the signals go high AXI Master transfers data via *AXI_WDATA* and when the last bit of data is sent *AXI_WLAST* is set to high. When the transaction is successful both *AXI_WVALID* and *AXI_WREADY* go low.

3 HPC Systems Survey

In this chapter we discuss all the FPGA equipped HPC systems operated by the Paderborn Center for Parallel Computing (PC^2) at the Paderborn University, their tool flow and select a suitable environment to check the feasibility of the memory monitoring approach. PC^2 is a scientific institute of Paderborn University with a focus on "advance interdisciplinary research in parallel and distributed computing with innovative computer systems"[15].

In recent years high-performance computing is growing considerably in the areas of science, engineering and technology, especially in industries. It has become a great hope for future application development which requires large quantities of computing resources. FPGAs are used nowadays as custom acceleration units in HPC systems. FPGAs reduce the burden of computation on the CPU by its parallel processing capabilities. A HPC system with an FPGA has an advantage of configuring the hardware for different application needs. An FPGA might perform part of a calculation of a given task, so the data stored in the shared memory should be protected from illegal accesses of the malicious logic in the FPGA.

The FPGA equipped HPC systems in PC^2 are

- Power 8 System
- Micron System
- HARP System
- XCL System
- Maxeler MPC-C System
- Convey HC-1 System
- Nallatech board System

The key factors to verify the monitoring approach are the control over implementation design, PCH verification support, a suitable shared resource to protect and also the possibility of using a memory access monitor. The above mentioned HPC systems are compared against the key factors and a suitable platform is chosen to demonstrate memory access monitoring and develop a formal verification tool flow.

3.1 Tool Flow

It is essential to choose a suitable tool flow when developing a prototype as a right tool flow can help to overcome many hurdles in the process of achieving memory access monitoring.

Tool Flow	HPC Systems	Pros	Cons
MaxJ (data flow language)	Maxeler MPC-C	Abstract level description Easy to code	Limited control over implementation Does not support PCH verification flow
Xilinx OpenCL Intel OpenCL	Micron HARP XCL Nallatech	Easy to code Abstract memory model Supports PCH verification flow	Limited control over implementation
Vivado HLS	Power8	Easy to code Supports PCH verification flow	Limited control over implementation
HDL (Verilog)	Power8 HARP Micron Convey-HC-1*	Complete control over implementation Supports PCH verification flow	Code complexity

'*' system is not currently available.

Table 3.1: Tool Flow comparison of HPC systems

Table 3.1 gives information about the tool flow used to code the user logic for an FPGA of the respective HPC systems to achieve memory access monitoring and formal verification.

MaxJ

The first tool flow in the table is MaxJ. It is a data flow language with abstraction level description which minimizes the code complexity of large circuits. MaxCompiler compiles MaxJ coded user logic and outputs an RTL file which describes the data flow and timing of a circuit. MaxJ gives limited control over the implementation design as the user logic is coded at a high-level language. MaxCompiler takes over the control from the user when compiling MaxJ to RTL. The tool flow also does not support PCH verification as it requires the design specification described in a Verilog which does not appear in MaxJ tool flow.

Xilinx OpenCL, Intel OpenCL

Xilinx and Intel have taken a step forward to use FPGAs for acceleration applications. So, they developed their variations of OpenCL to program FPGAs, GPUs and CPUs. As a high-level language, it reduces the code complexity. OpenCL uses an abstract memory model, common to all the platforms using OpenCL for programming. The compiler used

in OpenCL tools take over the implementation design, leaving the user with limited over the implementation design. The tools can generate Verilog from OpenCL. This Verilog description generated, can be used as design specification in PCH verification.

Vivado HLS

Vivado High-Level Synthesis (HLS) tool is used to code logic for an FPGA, using C, C++ or System C. All these are high-level languages, it is easier to code complex logic but lacking the control over the implementation design. Vivado HLS can be used to exploit the parallelism in FPGAs for acceleration applications. In a complex system with CPUs and FPGAs, HLS (any high-level language used to design in FPGA and not specifically Vivado HLS) can be used to get the best out of the code by running parts of it on software and hardware [1]. With the current tool available, Vivado HLS has restrictions over the code description, limiting the tools ability to understand the code and implement it efficiently. This tool generates a Verilog file allowing PCH verification.

HDL (Verilog)

Hardware Description Languages (HDLs) describe a digital logic circuit in the form of data flow, structural and behavioural models. They are implemented at Register Transfer Level (RTL) by the tools. A circuit described in HDL will closely resemble the actual implementation of a circuit. In a complex logic circuit, code complexity increases when using HDLs. Verilog used as design specification in PCH verification is an HDL. So, PCH verification is possible using this tool flow.

The access monitor not being an acceleration based design does not improve the scientific computation in a HPC system but thrives to secure the computation data on the shared memory. In the monitoring approach, tracking accesses and timing of decision-making on the access is essential, which requires control over the implementation design. To develop a formal verification using PCH, Verilog description of a design is needed. Hence, we use the Verilog (HDL) tool flow.

3.2 Available Shared Resources and Prototyping Possibilities

From the Table 3.1 HPC systems in PC^2 that support HDL tool flow are Power8, Micron, HARP and Convey HC-1. Out of these HPC systems Convey HC-1 is currently shut down and will not be active any soon. So, it is opted out from further analysis.

The following sections give a detailed explanation of HPC systems which support HDL tool flow and their shared resources (Table 3.2) followed by the possibilities of memory access monitoring.

3.2.1 Power8 HPC System

The Power8 HPC system [14] has a host RAM as the only global shared memory directly accessible from software and via the PCIe bus from FPGA. The main application runs on the host processor while the heavy computational functions run on an FPGA. Coherent Accelerator Interface Architecture (CAIA) combines the host CPU and the accelerator unit using the PCIe bus. This interface locally defines two functional components, the

HPC System	Shared Memory	Shared Bus	Shared Interface
Power8	Host RAM* PSL cache BRAM	PCIe bus*	Power Series Layer (PSL) Interface*
HARP	DRAM* FIU Cache Processor Last Level Cache*	QPI* PCIe 0* PCIe 1*	Core Cache Interface*
Micron	Host RAM* Hybrid Memory Cube* (HMC) BRAM	Pico bus* Stream bus*	Pico Multi-port Interface*

'*' indicates globally shared resources and the rest are local shared resources.

Table 3.2: Shared Resources of HPC systems which support HDL tool Flow

Power Series Layer (PSL) and Accelerator Functional Unit (AFU). The AFU contains acceleration specific functions on an FPGA. The PSL provides the base to all the communications between the host and the accelerator unit. PSL also provides a cache for the data used by the AFU. The ALU also uses local BRAMs to store data. The AFU, to get access to the host RAM, calculates the effective address and forwards it to the PSL. The PSL translates the effective address into a real address of the main memory. The PSL translates it only on request. Hence host RAM acts as a partially shared resource to the ALU.

The schematic view of the Power8 system from Figure 3.1 shows that the PSL interface is the only interface for the FPGA to interact with the host RAM via PCIe bus. Host RAM is protected by passing all the AFU accesses to PSL interface via an access monitor. This way ALU accesses have to pass through the monitor, therefore monitoring all the accesses to the host RAM. The PSL cache can also be protected using a memory access monitor.

3.2.2 The HARP cluster

HARP cluster is a HPC system developed by Intel. This system is briefly explained due to Intel confidentiality. Each node in the cluster has a global DRAM of that specific node. The FPGA consists of two parts namely interface unit and programmable unit. The interface unit uses three different buses namely, QPi, PCIe 0 and PCIe 1 to communicate with the Xeon processor. The programmable unit contains the Accelerator Unit (AU) used for scientific computations. Core Cache Interface (CCI) connects the programmable unit and the interface unit. CCI generates virtual addresses to access these buses on request by the AU. CCI provides the memory to the user logic on FPGA in a hierarchical form from Cache in interface unit (FPGA Interface Unit (FUI) cache), Last level cache in Xeon Processor to DRAM. An access monitor can monitor all the accesses to CCI to protect the shared memory.

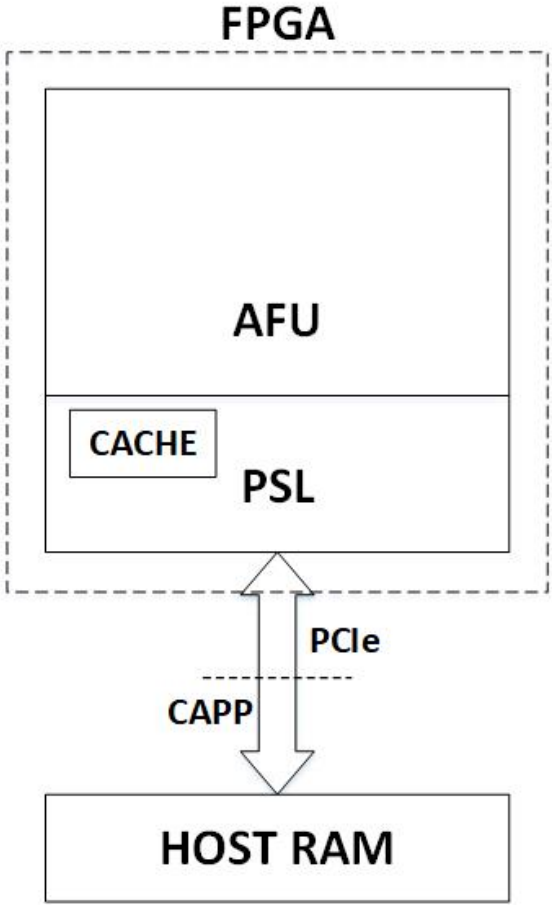


Figure 3.1: Schematic overview of the shared resources in a Power8 HPC system.

3.2.3 The Micron HPC system

Micron technology [3] developed the Micron HPC system (Figure 3.2) with Hybrid Memory Cube (HMC) technology. Pico Framework is the key for the Micron system to work as a unit. Pico Framework is a co-software, co-hardware interface. This framework handles all the interactions between Host CPU and FPGA. The address space of the host RAM is made visible by the Pico framework only on demand to the FPGA. So, host RAM is a partially shared resource. The Micron system has an off-chip global memory HMC accessed via the HMC controller from the FPGA.

The HMC is a 3-dimensional array of memory dies on a single logic die. It increases the performance of the system as each memory block on the HMC operates individually with a controller.

The Micron machine has two potential shared memories namely, the host RAM and the off-chip HMC. The Pico framework provides different interfaces for different communications. It provides a stream interface for the communicating the FPGA with the host RAM. The stream is a FIFO representation on hardware. The accesses to the stream can be passed through the access monitor to protect the host RAM. The Pico framework provides Pico bus for the host to communicate with the FPGA and also a path for the host to directly access FPGA. The Pico framework also provided Pico multi-port inter-

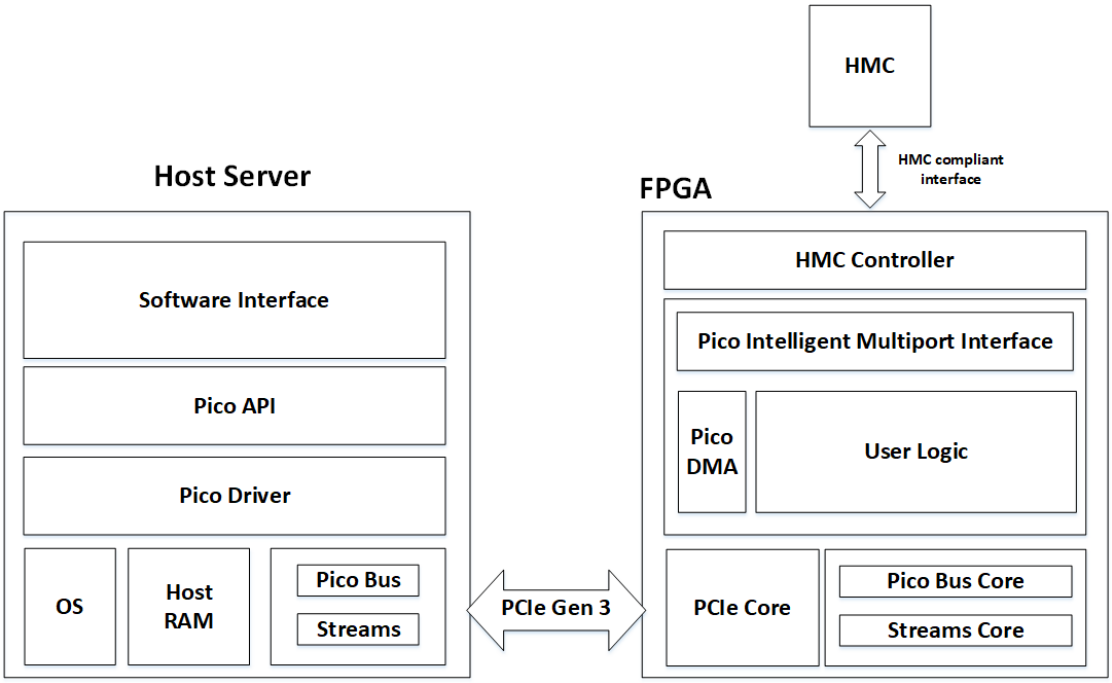


Figure 3.2: Block diagram of the Micron System

face for communication with the HMC controller. The Cores running on the FPGA have to use the same interface to access HMC. When we use Pico bus to communicate with FPGA its accesses can be extended to access HMC via FPGA. The accesses to the HMC via FPGA can be monitored using an access monitor to protect the data on HMC. Local BRAMs, when used by multiple cores, is protected similarly by an access monitor.

3.3 Platform for the memory access monitoring

From sections 3.1 and 3.2 the following conclusions are made in choosing a HPC system for prototyping.

In the Power8 system, the host RAM is a partial shared memory which makes it a less suitable resource to protect. The HARP cluster is a reasonable choice for the prototype as it has only one interface to communicate with all the shared memories. But due to the Intel confidentiality and lacking the freedom to share information, this system is not chosen for the prototype. The Micron HPC system stands out from other systems as it provides a unique opportunity to monitor both the host and FPGA accesses to the global shared memory HMC via Pico multi-port interface.

Verilog (HDL) as the tool flow and the HMC as the shared memory to protect, the Micron HMC system is chosen to test the concept of memory access monitoring.

4 Memory Access Monitoring on the Micron System

This chapter discusses different memory access monitoring architectures and its placement in different interfaces on the Micron system. First, the monitoring architectures are explained with their advantages and disadvantages in different scenarios. The best approach is selected for memory monitoring on the Micron system.

4.1 Shared Interfaces on the Micron System

There are three shared interfaces on Micron system, namely

- Pico Bus
- Streams
- Pico Intelligent Multiport Interface

From Figure 4.1 Pico Bus is used by the host to access HMC. Streams are used by the FPGA to interact with the Host RAM and other FPGAs without the host involvement and Pico multiport interface is used by the FPGA to access HMC.

4.1.1 Pico Intelligent MultiPort Interface

In the following section possibilities of memory access monitoring to a Pico multi-port interface is explained for different scenarios. Pico computing allows users to use the multi-port interface in four different ways.

- Full custom interface
- 640 bit-wide “native” interface
- Pico Computing’s high-performance interface
- AXI-4 like interface

The different interfaces available in the Micron system can be monitored using monitoring architectures introduced by Huffmire et al. (Figure 3.2). In the architecture to the left of Figure 3.2, the read and write accesses have to wait till the access legality is checked and enforced. In the architecture to the right of Figure 3.2, the read accesses are allowed to access the shared memory and at the same time access information is stored in buffers and read in parallel by the access monitor and the decision is enforced when the response is received from the shared memory. As the right architecture creates less latency compared to the left architecture, from this point we only talk about the architecture with buffer.

There is not much information provided for full custom interface either in the user manual or in the Pico framework website by Micron. We assume this interface to be fully customizable from the name given to it. This interface can be useful when the user requires to transfer 640 bit in one transaction. AXI-4 like interface in the Micron system is a custom built interface in Pico computing with a data bus width of 512 bits. AXI interface is useful when a user needs a single wide port of 512 bit for data transfer. Pico Computing's high-performance interface has five user ports each with a 128-bit user port. All the five ports follow round-robin arbitration. This interface is beneficial for more random accesses.

Full Custom Interface

This optional interface can be advantageous when other interfaces do not satisfy the needs of the user. The user is given a choice to customize the interface. The monitoring architectures as in Figure 2.2, can be applied to this interface due to its full customization capabilities. The software uses Pico bus to access FPGA. Hence a Pico to full custom interface bridge should be used to monitor software accesses.

640 bit-wide "native" interface

The name of the interface itself says that the interface has a data bus width of 640-bits. The interface is similar to a full custom interface. The only difference is that this interface has no customization capabilities. The monitoring architectures as in Figure 2.2 can be applied to this interface. The shared bus is the 640 bit-wide interface and the shared memory is the HMC. To monitor the software accesses, a Pico to 640 bit-wide interface bridge should be used and the accesses should be passed through the memory access monitor.

Pico Computing's high-performance interface

In this interface, there are five user ports with round robin arbitration. The monitoring architectures in Figure 2.2 can be applied to this interface. When multiple cores are assigned to each user port then each user port will have a memory access monitor and multiple cores access each user port using an arbitration scheme. Here the shared bus is the Pico Computing's high-performance interface and the shared memory is the HMC. To monitor the software accesses a Pico to Pico Computing's high-performance interface bridge should be used.

AXI4 like interface

The architectural concepts in Figure 2.2 can be applied to the AXI4 interface. All the cores or modules accessing the AXI interface have to follow the AXI protocols. The memory access monitor and cores accessing the AXI4 interface (shared bus) must be AXI peripherals (bus peripherals).

When the monitor is placed between AXI interconnect and the bus peripherals, all the accesses have to pass through the monitor. When using multiple cores, all cores and the monitor should be integrated into AXI Full peripheral as it provides ID support for the accesses by an individual core. When only a single core is used module id is not required. So, the monitor can be integrated into AXI Lite peripheral with a slight modification in

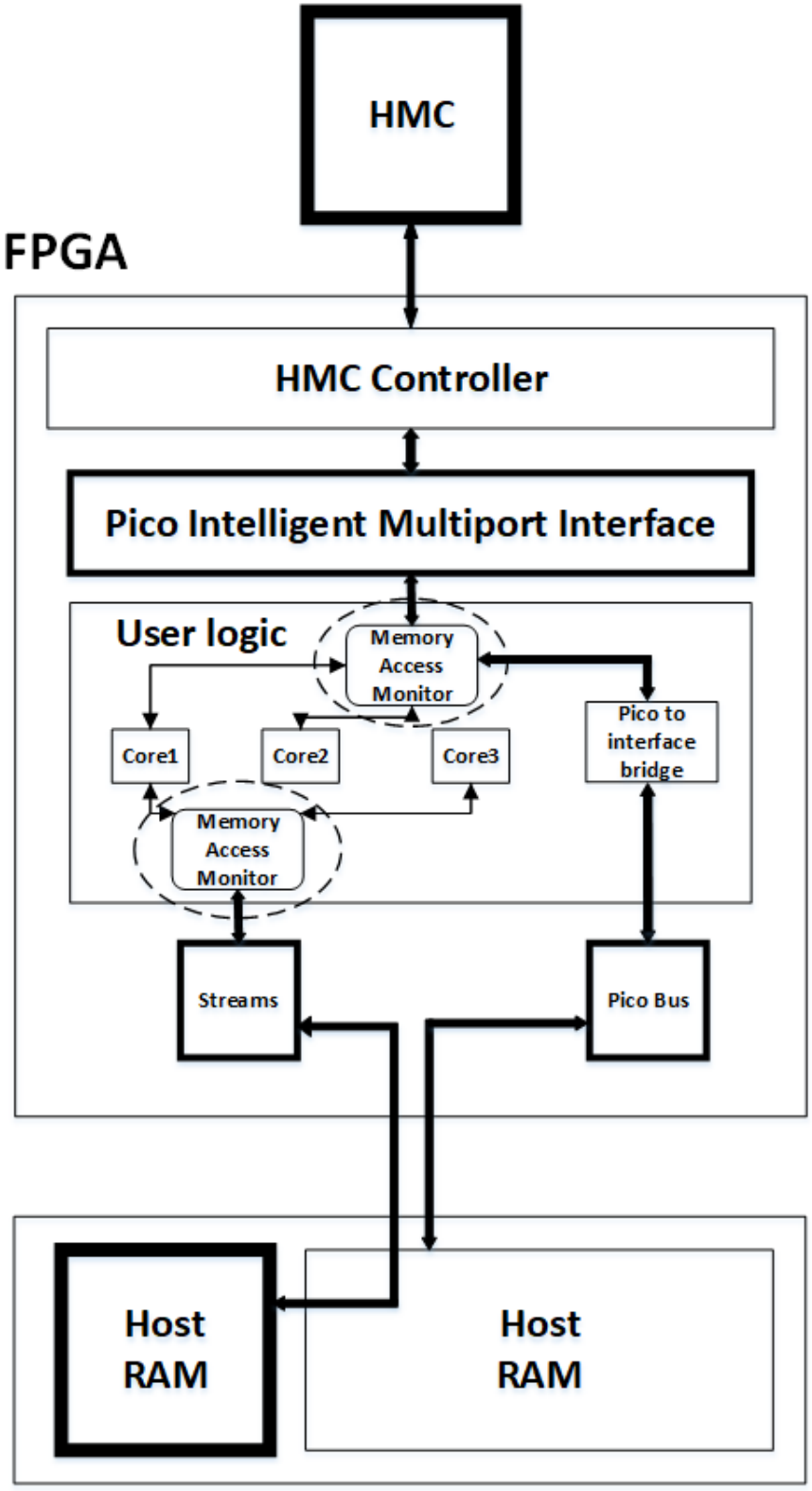


Figure 4.1: Possible locations of Memory Access Monitors in the Micron System.

the policy state machine by removing the module id input and having all the legal ranges of the peripheral in the policy state machine.

4.1.2 Pico Bus Interface

Pico Bus is used by the software to access FPGA. The accesses of the Pico Bus can be extended to access HMC by using a Pico to interface bridge and pass the accesses through the memory access monitor to monitor the software accesses.

4.1.3 Stream Interface

Stream interface is used by the FPGA to communicate with the host RAM. This interface is realized as a FIFO interface on hardware. There are two Stream interfaces in the Micron system, one as an input to the FPGA and the other as an output from the FPGA. This interface has no address parameter as data is sent or received in order. The policy state machine in the access monitor should be modified to contain a list of cores allowed to transfer data to the host RAM. The architecture in Figure 2.2 can be applied for one way communication.

From the above discussion it can be concluded that, out of all the available interfaces to access the shared memory in the Micron system, AXI4 interface is the only interface which can be available in other HPC systems and rest of the interfaces are specific to Micron system. So, we choose the AXI4 interface to access the globally shared HMC and apply the memory access monitoring approach.

4.2 Concept Design

In this section, we discuss the concept design of memory access monitoring using AXI4 interface to protect HMC.

The AXI4 like interface in the Micron system is similar to the AXI4 interconnect. As discussed above, all the AXI bus peripherals have to be AXI peripherals. So, the memory access monitor and all the cores or modules accessing the AXI4 interface must be AXI4 peripherals. As AXI4-Full interface has ID support used to tag each access with a unique ID, we assign one access ID specific to each core and common to all accesses made by that specific core. The ID of each AXI4-Full peripheral enables the policy state machine to distinguish the accesses of multiple cores.

The AXI protocols were previously introduced in section 2.3.1. The core which raises a request is the master and the core which responds to the request is a slave to the AXI interconnect. The HMC is the shared memory from which the data is read and written. So, it is an AXI slave to the AXI interface. All the cores request the data from and write data to HMC. So, they are AXI masters to the AXI interface. The memory access monitor is between the AXI interface and the cores. So, it has to act as AXI master to the AXI interface and as AXI slave to the cores. So, we design an AXI peripheral which has both AXI master and slave interfaces. In the AXI peripheral of the memory access monitor, the signals of slave interface are directly connected to the master interface. Slave interface acts as a pass-through interface and in the master interface, only the key signals which carry the important information are put to hold until the access legality is checked

and enforced. To arbiter the accesses of multiple cores in the design, we use AXI crossbar with round robin arbitration. The policy state machine generated by the policy compiler locks to a deny state when invalid access is made and does not allow any core to access the shared memory until the state machine is reset. This way all the accesses after invalid access are denied and no further requests to the shared memory is provided. Not giving access to the shared memory is a safe way to handle corrupt core. But this can give rise to a denial-of-service (DoS) attack where the corrupt core keeps raising invalid requests and effects the performance of the system. To overcome this attack whenever invalid access is made by a core, the access ID (specific to each core) of the core is raised as a read interrupt or write interrupt to a top module which then enforces a safety mechanism to shut down the core. When access is legal all the pins of the interrupt signals are set to low so that the top module ignores the interrupt signals.

4.2.1 Different ways to handle denial-of-service (DoS) attacks

Whenever an interrupt is raised by the memory access monitor, the top module receives the interrupt and decides what to do with the system. Few possible ways to handle DoS attacks are:

Cutting access to the AXI interface

By cutting the access to the AXI Interconnect all the accesses to the HMC can be avoided securing the data in the HMC. This can be achieved by either cutting the clock of the AXI interconnect or setting the reset of interconnect to low to disable it. In the Micron system, both the ways cannot be achieved from the hardware user logic. But the software can intervene and disable the AXI interconnect.

Shutting down the hardware system

In this case, the software has to interfere as the hardware user logic cannot shutdown the system. Whenever invalid access is made, the memory access monitor can raise an interrupt which the software receives and shuts down the system when interrupts are received.

Cutting the clock of the corrupt core

One graceful way to handle DoS attacks is to cut the clock of the corrupt core. For this approach in Xilinx Vivado, each core must be given a separate clock and the top module should be able to cut the clock of the corrupt core. The tool does not allow different clocks with same frequency for each AXI peripheral when connected to AXI crossbar and either a clock converter or an asynchronous FIFO must be added to each core for the tool to allow different clocks in the design. When multiple cores are used in the design this method increases the area of the design. If Vivado allows using multiple clocks when AXI peripherals are connecting to AXI crossbar, this approach is feasible.

Reset the corrupt core

Another graceful way to is to set the reset signal to low of the corrupt core when an invalid access is made. When an interrupt is raised by the memory access monitor, the

top module reacts to it by setting the reset signal to low of the corrupt core.

Depending upon the security needs of the system, one of the above mentioned methods can be applied. Out of all the above mentioned methods, cutting the access to AXI interface or shutting down the entire system are harsh ways to handle DoS attacks and are always better for the safety of the system. We consider handling the DoS attacks in the hardware and cutting the clock currently not being feasible, resetting the corrupt core method is enforced in the design.

The limitation of the state machine generated by the policy compiler is that when a core requests a burst of data or writes a burst of data with a starting address which is within its legal range and the end address is out of its legal range, the state machine fails to realize this and recognizes the access as legal. So, we modify the state machine to overcome this limitation by adding logic to consider the end address of a request depending on the burst size of data. AXI protocols support independent read and write channels. So, we use two state machines, one to monitor read accesses and the other to monitor write accesses. When a core makes invalid read access and valid write access or vice verse, logic is added in the design for the two state machines to communicate with each other such that once invalid access is made, any other legal access by the corrupt core is considered illegal. For an invalid read access, dummy data is sent to the core instead of actual data and in case of invalid write access, the request is not forwarded to the HMC. Consider cases where the starting address and the end address of a core are within the legal access range and the address range of other cores fall between the start and end address of the previous core. As the ranges are specified in the policy state machine, if the core is allowed to access those ranges it is considered as legal access. If the core is not allowed to access these ranges it is considered as illegal access.

The Pico bus does not follow AXI protocols. So, a Pico to AXI bridge is designed and all the software accesses are via the memory access monitor. The Pico bus works on Pico clock and the AXI erface works on the AXI clock. The signals from Pico bus have to cross two clock domains for which the AXI provides a clock converter peripheral and an asynchronous FIFO peripheral. The Pico clock and AXI clock have similar frequencies and the clock converter does not have 1:1 ratio for the clock conversion. So, an asynchronous FIFO peripheral with three stages (to avoid metastable state) is chosen for the clock domain crossover.

As mentioned in the early discussions, the top module is designed to handle interrupts. Whenever an interrupt is received, the top module compares the interrupt to the core IDs and disables the corrupt core. The Pico-AXI-Bridge is not disabled for invalid access because the firmware is a slave to software in the Micron system and interrupting software accesses may cause the system to hang. Further technical details of the design are explained in Section 5.1.

5 Implementation and Evaluation

This chapter explains the design implementation of the prototype and its formal verification tool flow. The feasibility of the memory access monitoring is supported by the implementation results and the functioning of the memory access monitoring is supported by the evaluation results. The current Micron HPC system is as follows: The Pico computing used in the system is version 6.0.0.21 running on a high performance AC-510 module combining a Kintex® Ultra-scale FPGA with a high bandwidth off-chip Hybrid Memory Cube (HMC) version 4.3.21 snapped onto an Ex-700 PCIe back-plane. The tools used are Xilinx Vivado design suite HLx 2017.2 on a CentOS platform.

5.1 Implementation of the memory access monitoring approach using the prototype

The prototype design is implemented as shown in Figure 5.1. To access the HMC we use image processing cores integrated into the AXI4-Full master peripherals. These peripheral's accesses to HMC are passed through the Memory Access Monitor (MAM) to check their legality. Each image processing AXI peripheral reads and writes 1Kb data in a burst of 256 transfers. In a single transaction, each core makes 4 bursts, reading and writing 4KB data in total via a 32-bit bus. We define the memory access policy in the formal language introduced by Huffmire et al. [12] and the policy compiler generates the policy state machine. We specify the peripheral IDs, their access ranges and the access rights of these ranges in the formal language as below.

For read and write policy:

$Range_1$ → [0x00010000,0x0001FFFF]; (Gaussian filter 1)
$Range_2$ → [0x00020000,0x0002FFFF]; (Gaussian filter 2)
$Range_3$ → [0x00030000,0x0003FFFF]; (Pico-AXI-bridge)
$Range_4$ → [0x00040000,0x0004FFFF]; (Gaussian filter 1)
$Range_5$ → [0x00050000,0x0005FFFF]; (Gaussian filter 2)
$Range_6$ → [0x00060000,0x0006FFFF]; (Pico-AXI-bridge)

Access → {{$Module_1$,rw,$Range_1$}|{$Module_2$,rw,$Range_2$}|{$Module_3$,rw,$Range_3$}|
{$Module_1$,rw,$Range_4$}|{$Module_1$,rw,$Range_5$}|{$Module_1$,rw,$Range_1$}};

Policy → (Access)*;

The policy-compiler generated policy state machine is summed up in Table 5.1. $Module_1$ has read and write access to $Range_1$ and $Range_4$. $Module_2$ has read and write access to $Range_2$ and $Range_5$. $Module_3$ has read and write access to $Range_3$ and $Range_6$. Policy state machine gives an output as one for legal accesses and zero for any invalid access.

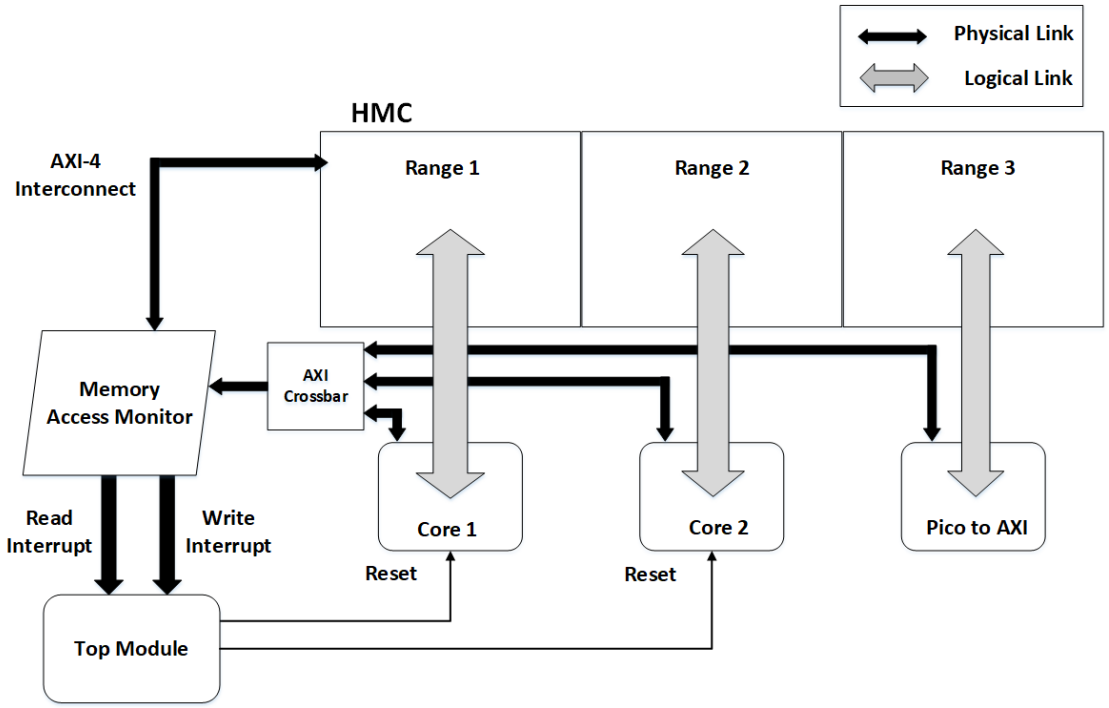
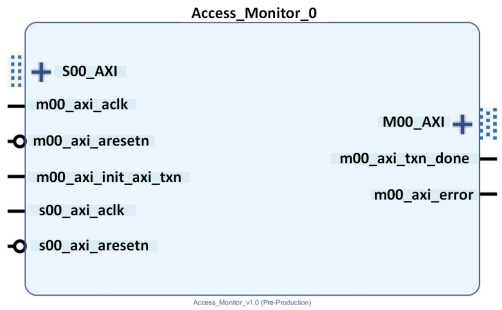


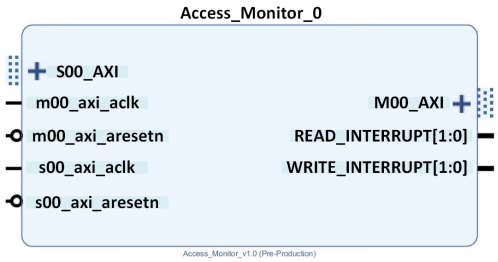
Figure 5.1: A Compartmentalization policy architecture. HMC stands for Hybrid Memory Cube.

Parameters	Module1	Module2	Module3
Functionality	Gaussian Filter	Gaussian Filter	Pico to AXI Translator
Module Id	01	10	11
Read and Write Range	{0x00010000, 0x0001FFFF} {0x00040000, 0x0004FFFF}	{0x00020000, 0x0002FFFF} {0x00050000, 0x0005FFFF}	{0x00030000, 0x0003FFFF} {0x00060000, 0x0006FFFF}

Table 5.1: Compartmentalization policy for the HMC



(a) AXI-4 Full peripheral with both Master and Slave interfaces



(b) Modified AXI-4 Full peripheral as a Memory Access monitor with Read and Write interrupts

Figure 5.2: AXI-4 Peripheral for the design of Memory Access Monitor

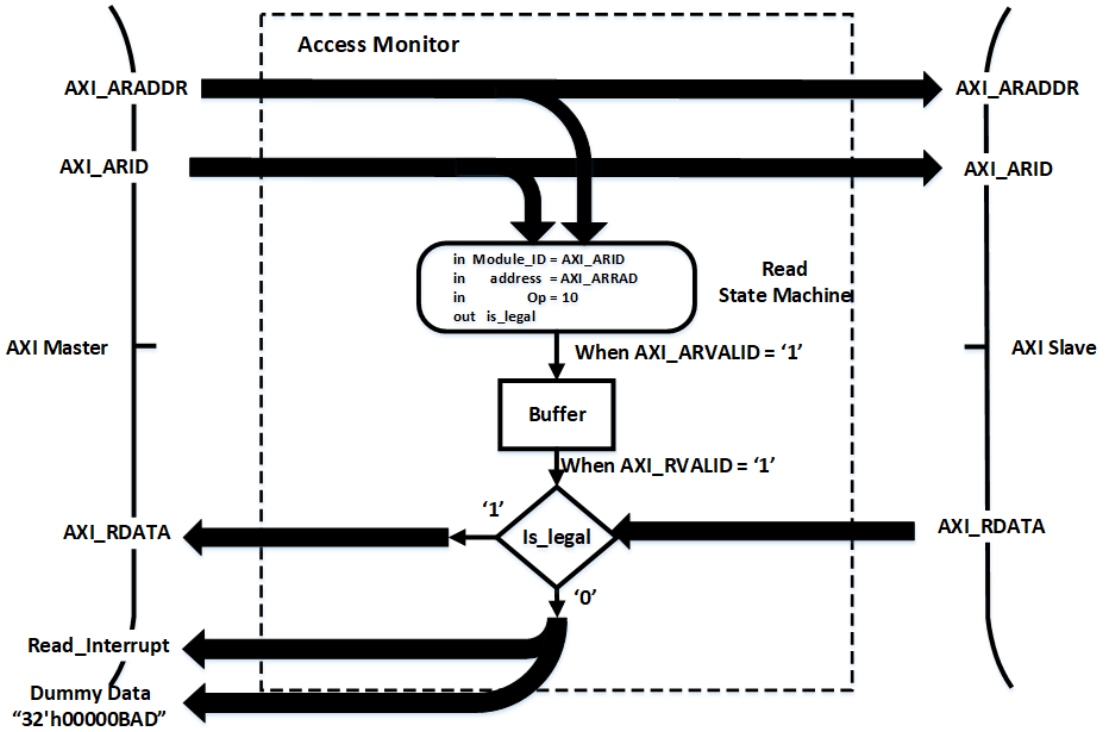
As explained in Section 4.2 an AXI-Full peripheral with both AXI-Full Master and AXI-Full Slave interface is used for the Access Monitor design (Figure 5.2). The AXI Interconnect can handle both read and write transactions at the same time as it supports independent read channel and write channels (Figure 2.6). So, two state machines are used in the monitor design, one for the read accesses and the other for the write accesses from the cores to the HMC. The monitor holds the read response and write requests until the legality of their access is checked and enforced.

The process in which the read accesses are handled by the monitor is depicted in Figure 5.3a. When a core is accessing the AXI interface to read from HMC, it provides the information of the address along with the ID of the core via *AXI_ARADDR* and *AXI_ARID* respectively. These signals are given as inputs to the read state machine. As it is a read state machine, the operand is always set to '10'. By the time HMC responds to the read request, read state machine checks the legality of the access. The state of the legality is saved in a buffer specific to each core when the *AXI_ARVALID* goes high. The data signal *AXI_RDATA* is forwarded only when the access was found to be legal by the read state machine and enforced when *AXI_RVALID* is high. *AXI_RVALID* is set to high by the slave when it is ready to send the data. When the access is found to be illegal, dummy data as "32'h00000BAD" is sent to the corrupt core and the ID of the corrupt core is generated as a read interrupt which is used by the top control module to disable the specific core. The interrupt signals handled by the top control module is shown in Table 5.3.

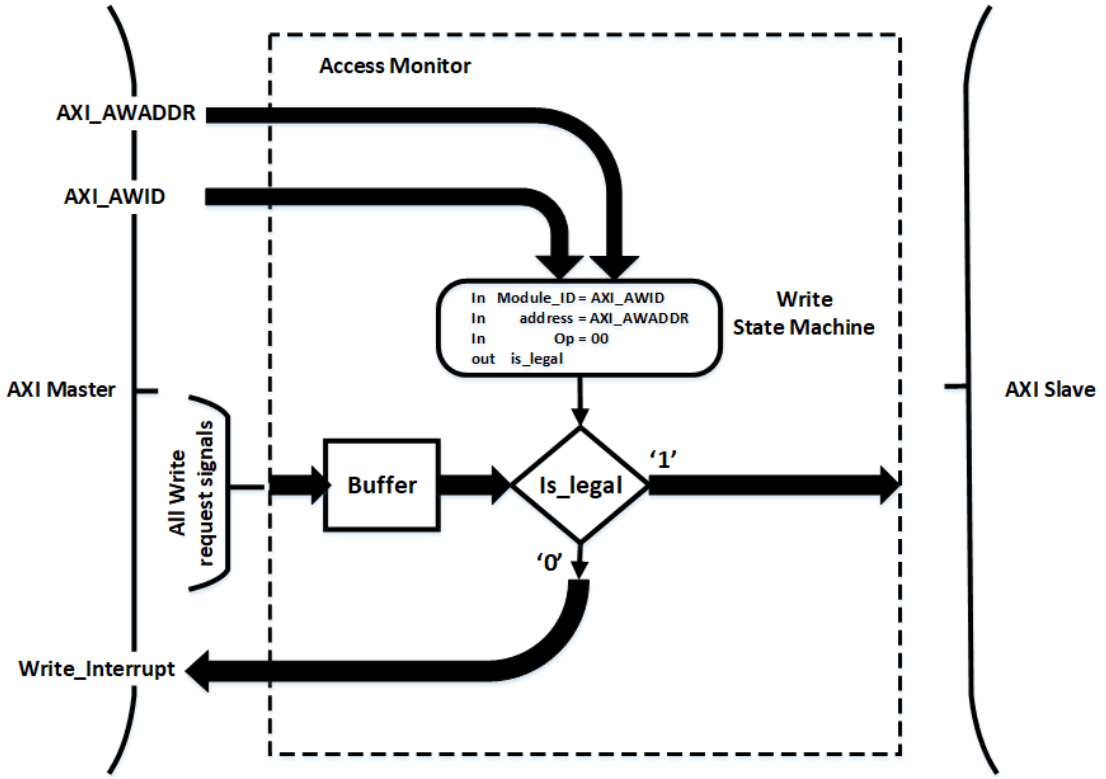
The write accesses are handled differently compared to the read accesses. The write channel flow via the Access Monitor can be seen in Figure 5.3b. In the write state, before the data is written to the HMC, write access legality is checked. All the write request signals are stored in a buffer. When a core requests to write data to the HMC, the write address and access ID of the respective core are communicated via *AXI_AWADDR* and *AXI_AWID* respectively. *AXI_AWADDR* is taken as input to the address port of the write state machine and *AXI_AWID* as the module ID. The operand of the write state machine is always set to '00'. All the write request signals are put to hold until the write state machine checks the access legality. The decision from the write state machine is enforced when *AXI_AWVALID* goes high. If the access is legal, the signals are forwarded to reach HMC. If the access turned out to be illegal the access ID of the core is generated as a write interrupt signal which is used by the top control module to disable the respective core.

One of the reasons to choose the Micron machine for the prototype design was the possibility to monitor the accesses from the software (Section 3.2.3). A Pico-AXI-Bridge is designed to monitor the software accesses. The IP design of the Pico to AXI translator can be seen in Figure 5.4. An AXI4-Full Master peripheral is modified to convert the Pico signals to AXI signals. An asynchronous FIFO with three-stage clock domain cross over is added in the next stage. All the data and address channels used by the AXI FIFO peripheral are set as a pass-through except read data channel. At least one channel must have a FIFO depth in the FIFO IP. So, the read data channel is set to the least FIFO depth available to reduce the area consumption by the core when implemented.

The image processing cores (firmware accesses) and the Pico-AXI-Bridge (software ac-



(a) Read channel flow inside Access Monitor



(b) Write channel flow inside Access Monitor

Figure 5.3: Read and Write channels handled by Access monitor

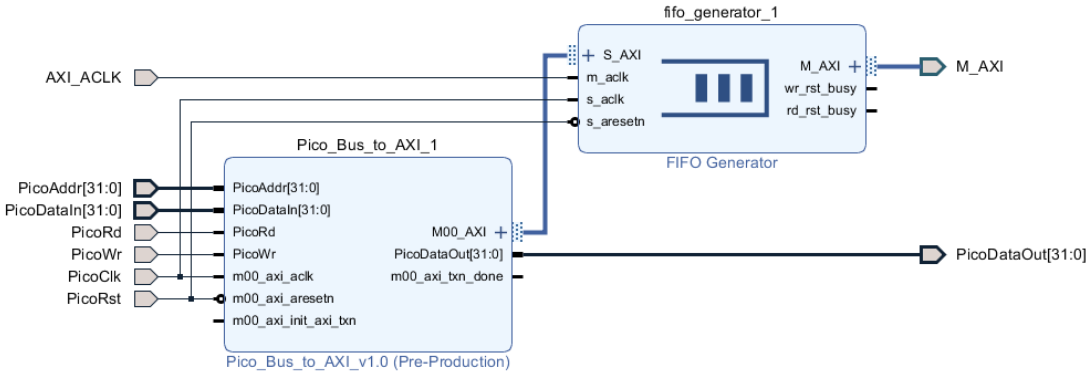


Figure 5.4: Pico to AXI translator IP with a three-stage asynchronous FIFO IP

Signals	Properties
PicoClk	250MHz free-running clock
PicoRst	active high reset signal
PicoAddr[31:0]	32-bit byte address (the address the host is reading from or writing to)
PicoDataOut[31:0]	[output] 32-bit data output (data read from the FPGA)
PicoRd	active high read strobe—asserted when the host is doing a read
PicoDataIn[31:0]	32-bit data input (data written to the FPGA)
PicoWr	active high write strobe—asserted when the host is doing a write

Table 5.2: Pico Bus Signals [3]

cesses) in the FPGA access the HMC. The memory access monitor checks the legality of all the accesses and the enforces the decision. The top control module disables the corrupted core except for Pico-AXI-Bridge.

After implementation, we observed that the disabling the corrupt core was causing the AXI interface to be stuck in a state where it is not available for use to other cores. So, to overcome this we decided to let the corrupt core write at a dummy address on the HMC specifically allocated for this purpose and then when the corrupt core leaves the AXI interface we disable it. Now other cores were able to access the AXI interface without any issues. The reason for making this choice is explained with simulation waveforms in 5.2.

5.1.1 Impact of Memory access monitor on the System Performance

FPGAs run on low frequencies. So, to achieve performance, many applications are designed to perform parallel computations. Such applications do not care about latency. For such applications memory, access monitors do not have a significant effect on the performance of the system. The HMC on the Micron system responds with variable latency depending on the load to the HMC. Hence latency cannot be accurately calculated from the implementation on FPGA. Hence latency was calculated from the post-implementation timing simulations which were expected to closely resemble the actual

Interrupts	Property
01	Core 1 made an invalid read or write access. AXI_ARESETN of core 1 is set to low.
10	Core 2 made an invalid read or write access. AXI_ARESETN of core 2 is set to low.
11	Pico-AXI-Bridge made an invalid read or write access. No action is taken.
00	Legal access. No action is taken

Table 5.3: Interrupt signals interpreted by the top control module

implementation. The maximum clock speed of the FPGA on the Micron system is 250 MHz resulting in a clock time of 4ns. We used the same clock frequency in the simulations. For read access, we counted the number of clock cycles taken from the point where data is requested by sending an address to the point where the last bit of data is received. For write access, we counted the number of clock cycles taken from the point where the write address and data are sent, to the point where the response from the Block RAM is received. For read access there was no delay with and without the monitor in the design. For write access the memory access monitor added one clock cycle latency to the design.

Comparison of the area and throughput of the system with and without the monitor can be seen in Table 5.4. To calculate throughput the number of clock cycles taken for an individual core to read and write 4KB data is counted. The clock count can be variable due to variable latency of the HMC. So, we counted the clock cycles for 10 different runs and divided it by 10 to get an average clock cycle count. The area overhead due to memory access monitor is only 2% of the design which is negligible.

Parameters	Without Access Monitor	With Access Monitor
Area (LUTs)	4202	4293
Average Throughput	1.753(GB/S)	1.740(GB/S)

Table 5.4: Area and performance effects of memory access monitors on the system

5.1.2 Formally Verified Memory Access Monitor

The induction based PCH technology and its tool flow by Isenberg et al. [16] is adapted to formally verify Access Monitor. The formal implementation is not implemented as the tool flow cannot produce the Xilinx FPGA bitstream. When the bitstream information is provided by Xilinx, this tool flow can be implemented.

Consumer: Memory Access Monitor Specification

The consumer describes the design functionality of the memory access monitor and submits it to the producer. The consumer will use the policy compiler by Huffmire et al. [12] to describe the memory access policy. The memory access policy and the memory access monitor are specified in behavioural Verilog. The only requisite is that the producer should be able to synthesize an implementation from the design specification, and the consumer should be able to derive a functional equivalence miter from the design specification. The security property employed is the functional equivalence, i.e., the

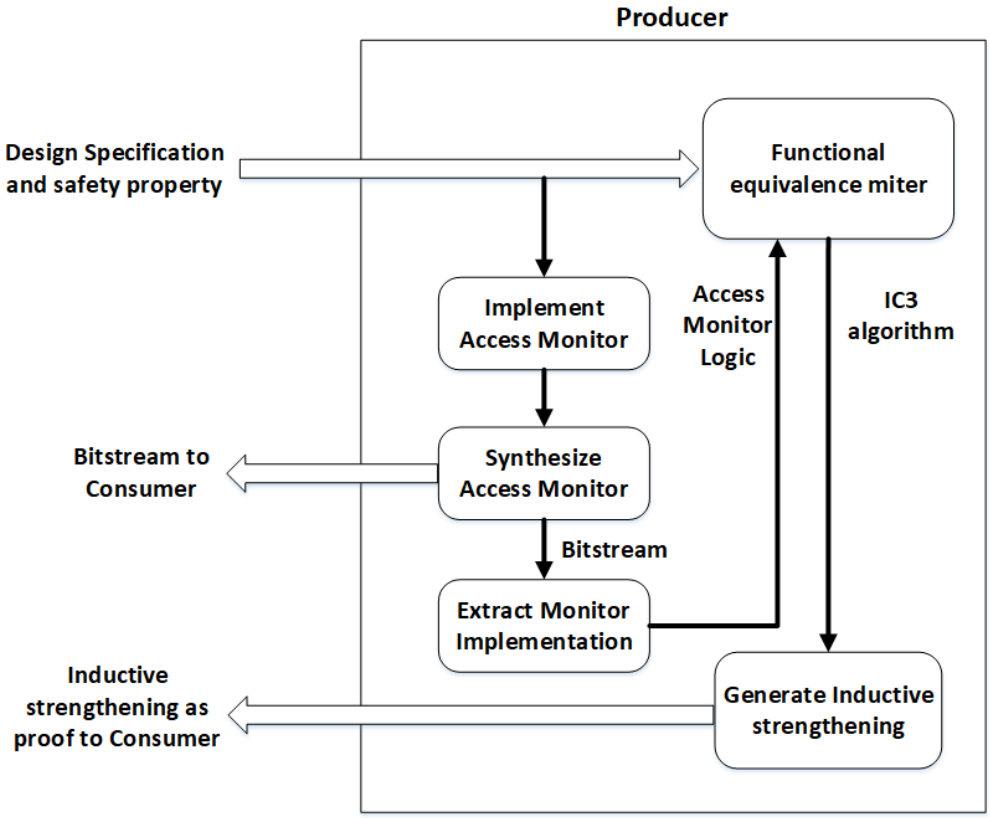


Figure 5.5: PCH Producer tool flow

inductive proof satisfies the functional equivalence of design specification to the implementation. The consumer tool flow is described in Figure 5.6. For a general scenario, a third party could specify the memory access policy and the safety property and order an implementation form the producer. This implies that the consumer will receive the design specification from the third party and the design implementation along with the inductive proof from the producer.

Producer: Implementation of Memory Access Monitor

The producer tool flow can be seen in Figure 5.5. The producer receives the design specification of the Memory Access monitor along with the safety property. The producer synthesizes the FPGA bitstream using the tool by Huffmre et al. and VTR for Verilog synthesize and place & route. The producer then re-extracts the logic from the bitstream and along with the design specification, generates the functional equivalence miter. The policy checker is then verified by IC3 algorithm to find the inductive strengthening of the safety property. The generated bitstream along with the proof of inductive strengthening is submitted to the consumer.

Consumer: Verification of Memory Access Monitor

A pictorial representation of Consumer tool flow can be seen in Figure 5.6. The consumer receives the access monitor bitstream and the proof of its functional correctness. The

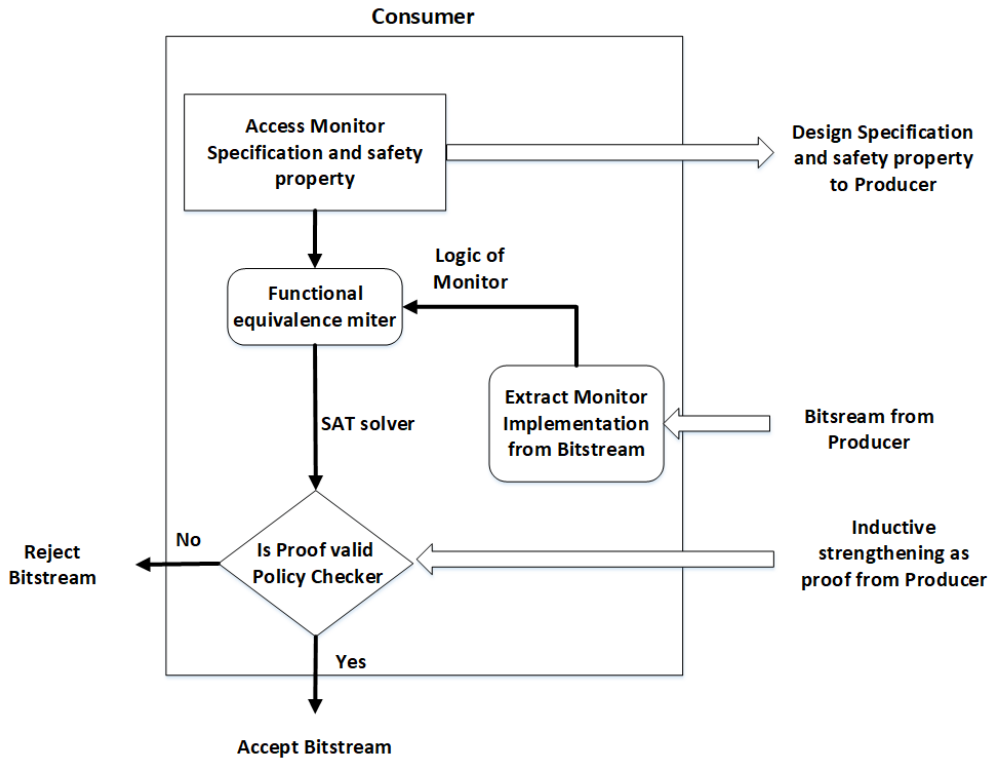


Figure 5.6: PCH Consumer tool flow

consumer then extracts the monitor logic from the bitstream and along with original design specification generates a policy checker. The consumer verifies the generated policy checker using the inductive strengthening received which yields three small SAT problems. These SAT problems when solved by an SAT solver will yield unsatisfiable and the functional equivalence holds for the design specification and the implementation. The bitstream is used by the consumer in the design.

If the implementation is faulty or the proof submitted is a different design then the inductive strengthening will yield satisfiable and the bitstream is rejected.

5.2 Evaluation of access monitoring approach using the prototype

To evaluate the memory monitoring approach the design setup is the same as Figure 5.1, only the HMC is replaced by a True Dual Port BRAM. The access range of all the cores remains the same as in shown in Table.

The effect of disabling the core on the system is as follows:

5.2.1 Case1: Invalid read access

From Figure 5.7 it can be observed that when core 1 makes invalid read access of 0X00020300 which is out of its legal range, the access monitor generates the read interrupt as '1' and the top module sets the reset signal of the core 1 and disables it.

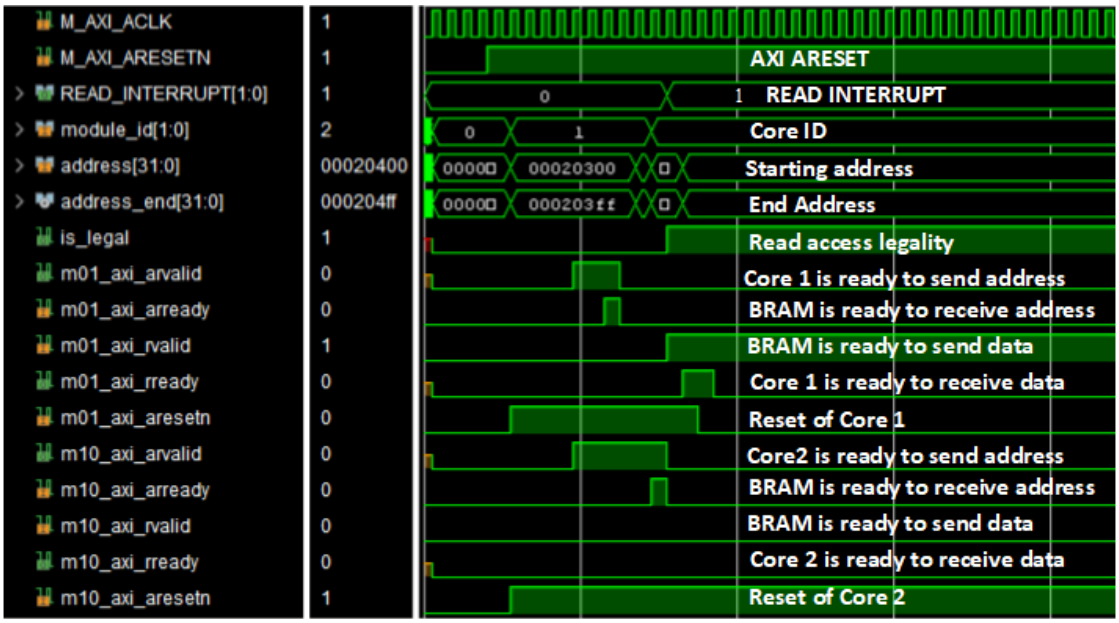


Figure 5.7: Simulation when a core is disabled when invalid read access is made

Core 1 and core 2 requested a transfer of 4 bursts and since the AXI crossbar used is round robin arbitrated, the slave must respond to core 2 after responding to core 1. This does not happen in the simulation. As seen in the simulation, core 2 never received the response from the BRAM that it is ready to send data. Theoretically, when a core is disabled the other cores should be able to access the HMC via AXI interconnect. But according to AXI protocols [21] the AXI interconnect has no time out for an access and waits for an infinite time until the transaction is complete. Until a transaction is complete between the master and slave interface, they are bound to the read channel. The slave is expecting to send data in 4 bursts and the core is disabled, the read channel is assigned to the corrupt core at that instance. So the read channel is not assigned to other cores as long as the previous transaction is complete. Hence the slave is stuck in the state. This can be attempted to solve in two ways.

Method 1

One way is to disable the corrupt core when it receives the dummy data and leaves the read channel. But as soon as the core receives the data, the state machine in the core changes its state to write state and may also immediately get hold of the write channel. So, the core must be disabled as soon as it leaves the read channel. It is quite challenging to achieve this timing as it takes just one clock cycle to change the state from read to write in the core and it also takes one clock cycle for the top module to receive a signal from the memory access monitor that the transaction is complete. The core must be disabled between these clock cycles. May be then the corrupt core leaves all the channels of AXI and safely disabled.

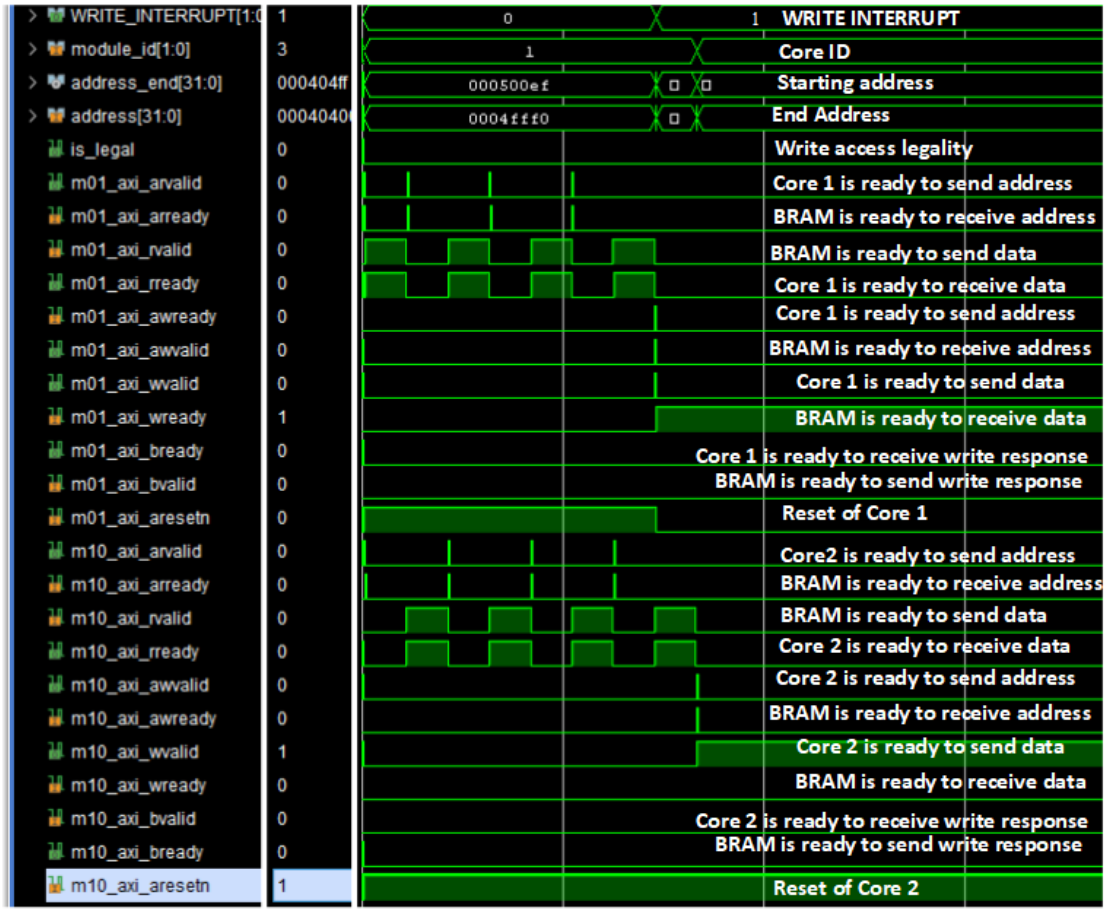


Figure 5.8: Simulation when a core is disabled when invalid write access is made

Method 2

Another way is to add a dummy master logic inside the memory access monitor. When invalid access is made, the dummy master logic switches the signals with the corrupt core and responds to the slave and completes the transaction so that the slave interface is not stuck in the read channel. This idea has to be tested and verified.

5.2.2 Case2: Invalid write access

From Figure 5.8, an access is made by core1 at 0X004FFF0 which is within its legal limits, but the access is of type burst, so the end address is 0x00500EF which is out of its legal access range. The access monitor raises a write interrupt as "1" which is the ID of core 1 in decimal format and the top module sets the reset of core 1 to low and disables core 1. As it is disabled the signals never reach the BRAM. Theoretically, the next core in line, core 2 in our case, should be able to communicate with the BRAM. From the figure, it can be seen that core 2 sends a signal that it is ready to send data but the BRAM never responds to the request. This can be attempted to solve in two ways.

Method 1

One way is to include a dummy slave interface in the access monitor which responds to invalid write access, receives all the data and the corrupt core is shut down after the transaction is complete. In this way, other cores may still be able to access the shared memory.

Method 2

Letting the corrupt core write the data to a dedicated dummy address. By this, the transaction is completed between the core and shared memory and the core can be disabled after the transaction.

From the above two cases, it can be concluded that shutting down the core after a complete transaction is desirable as it may let the system function as usual. This way, we let the corrupt core receive dummy data when invalid read access is made. As the core made invalid read access, the read state machine tells the write state machine that the specific core is corrupt. Even if the corrupt core makes valid write access, it is recognized as invalid access. Now the corrupt core's write data is written to a dummy address and after the transaction is complete according to AXI protocols, each core has to set a signal high which indicates that the transaction is complete. This signal can be used by the top module to disable the core.

From Figure 5.9 it can be observed that core 1 read access was found illegal and the read access legality from the read state machine is set to low. For valid accesses of core 2, it is set high. For the read transactions core 1 receives dummy data. This cannot be shown is simulation waveform as the data read is in kilobytes and cannot be seen unless extremely zoomed in. This is avoided for readability and will be shown later. Now the write state machine receives a valid write address request from core 1, recognizes it as invalid access and writes the data to a dummy address. After the transaction is complete, core 1 sets the transaction complete signal to high and at the same instance top module sets the reset of core1 to low, disabling it. From the figure, it can be seen that core 2 is able to access the bus as usual and the system works without halting in any state.

From Figure 5.11 it can be observed that core 1 makes valid read access followed by invalid write access and the write access invalid signal is set to low. As explained above for invalid write access the data is written to a dummy address. After the transaction is complete, core 1 sets the transaction complete signal to high and at the same instant top module sets the reset of core 1 to low, disabling it. From the figure, it can be seen that core 2 is able to access the bus as usual and the system works without halting in any state.

For readability of waveforms, we now scale down each core access to 32 bits per transaction in Figures 5.10, 5.12. In Figure 5.10 software makes invalid read access. The access monitor raises an interrupt as "11" with the module ID of the software core. Core 2 is also making invalid read access so "10" can also be seen as read interrupt. As we are not disabling the Pico to AXI core for invalid accesses it is allowed to make accesses. Since it is invalid read access, it receives dummy data "0X00000BAD" as a response on the PicoDataOut signal.

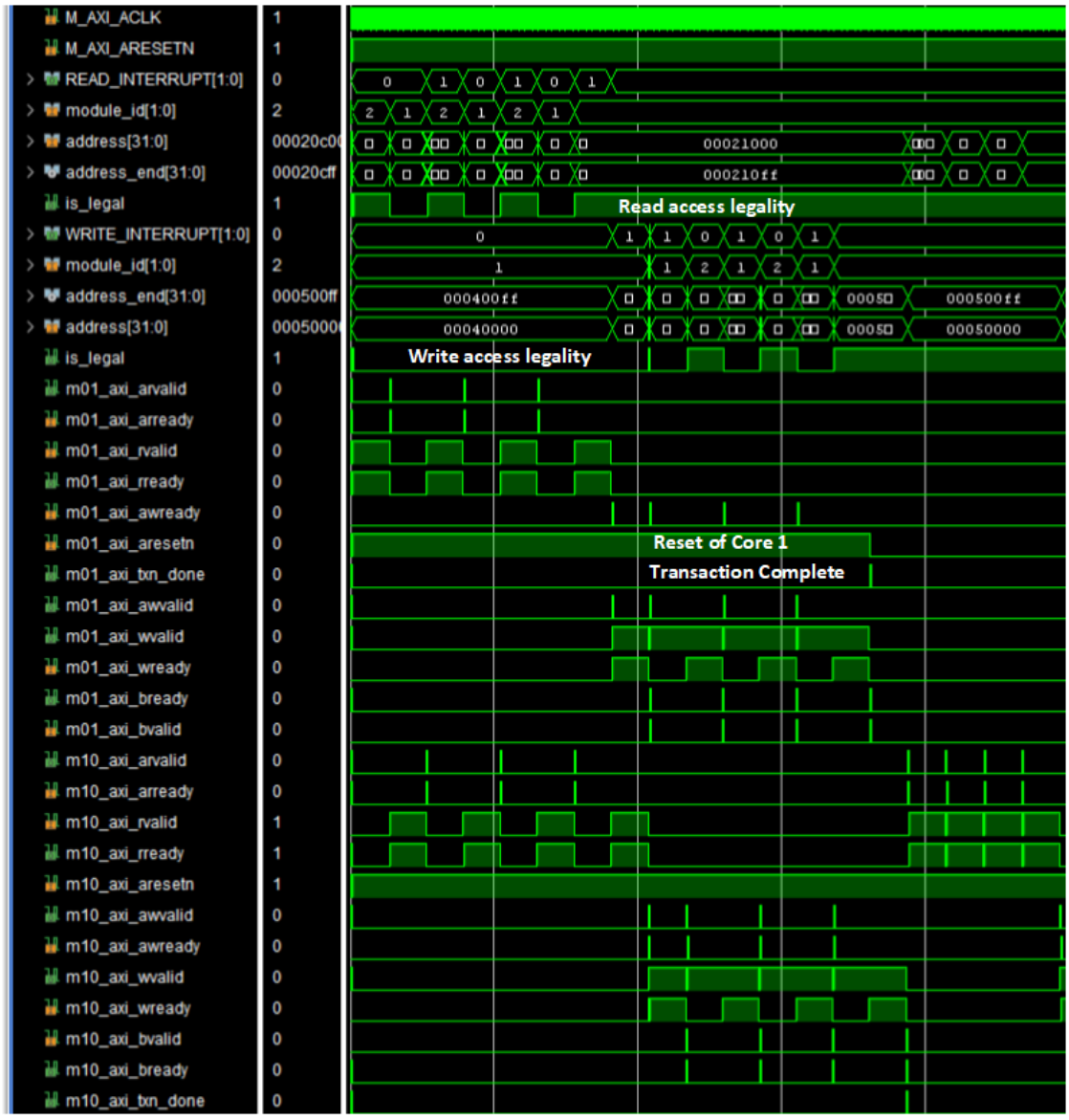


Figure 5.9: Simulation when core is disables after transaction for invalid read access

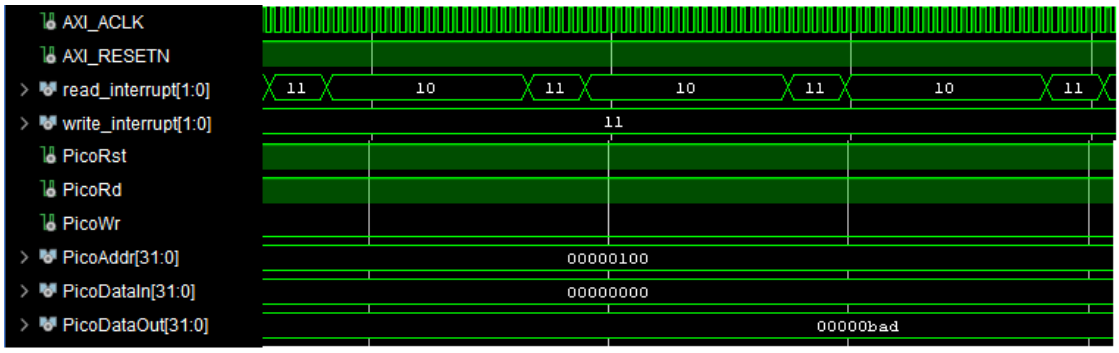


Figure 5.10: Simulation for software read invalid access

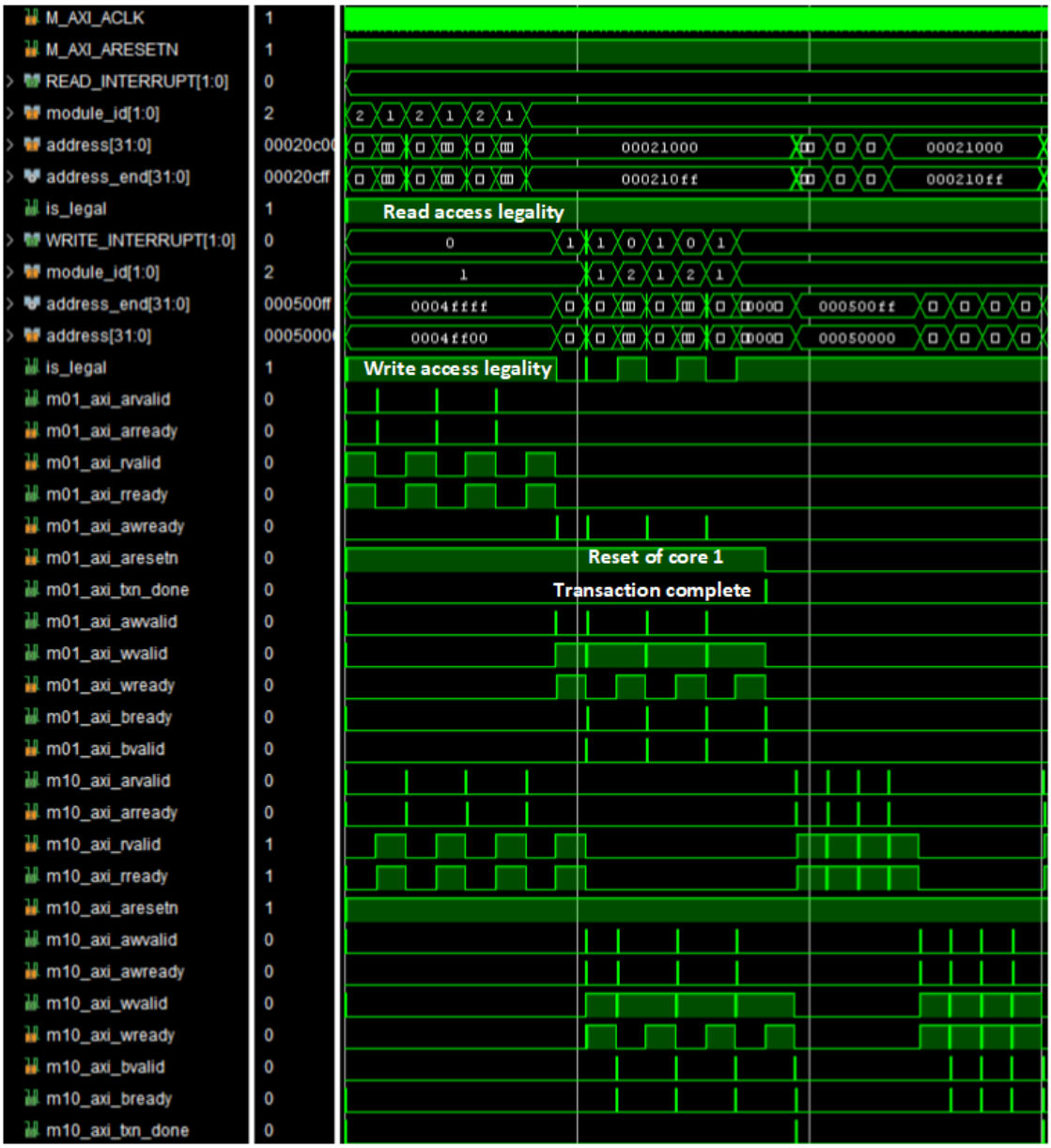


Figure 5.11: Simulation when core is disables after transaction for invalid write access

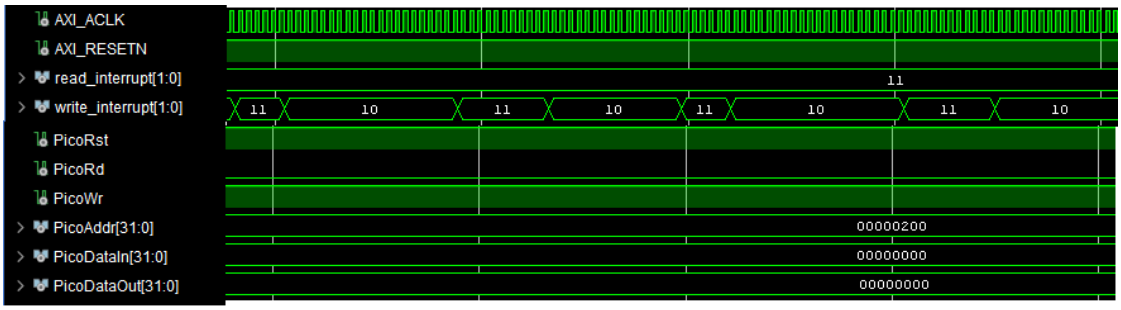


Figure 5.12: Simulation for software write invalid access

In Figure 5.12 software makes invalid write access and it can be seen that core id of Pico to AXI "11" is raised as an interrupt and the top module does not take any action on it and is allowed to make further accesses. The data is written to a dummy address when access is invalid.

5.2.3 Limitations of the Memory Access Monitor

Since a 32 bit data bus is used, the Monitor imposes a limitation on the throughput of the system. The data bus width can further be increased to 512 bits to obtain maximum throughput.

6 Conclusion and Future Work

The recent advances in science and technology demand reliable, efficient and swift systems like High Performance Computing (HPC) systems. These systems often have FPGAs to perform heavy computations. The programmable logic in these FPGAs often designed by third parties can be malicious in nature and mess with the computational data in the shared memory. The goal of this thesis was to check the feasibility of a Memory Access Monitoring System in a HPC system to protect a shared memory. We were successful in implementing this on a Micron HPC system at the University of Paderborn, chosen due to its ability to monitor memory access requests of both the Hardware and Software counterparts. We have successfully protected HMC (shared memory) form internal attracts of malicious programmable logic. We have extended the functionality of policy state machine of Huffmire at el. to check access legality for burst data is accessed. Depending on the system security needs our The measured parameters of the memory access monitor show negligible access delays and impact of area overhead and throughput is also significantly low. The memory access monitor is designed to be generic due to its inclusion into an AXI peripheral which is a near universal bus system, allowing it to be used on any HPC system supporting the AXI interface.

Following this, I presented a verification tool flow to verify if the memory access security intended by the consumer is met by the implemented Memory Access monitor, in a HPC system context. This was not achievable since the Xilinx does not disclose its bitstream properties required by the open source tool used for this verification.

Going further, the memory access monitoring can be improved to safely handle DoS attacks by using dummy read and write interfaces as a part of memory access monitor. A complete verification flow of the Access monitor using an induction based PCH tool flow can be performed when details of Xilinx FPGA bitstream are available. The memory access monitors can be isolated at placement and route level to ensure its safety from malicious logic.

Bibliography

[1] G.D. Bailey. *"The Advantages and Limitations of High Level Synthesis for FPGA Based Image Processing"*. In *Proceedings of the 9th International Conference on Distributed Smart Cameras*, ICDSC '15, pages 134–139. ACM, 2015.

[2] C. Basile, S. Di Carlo, and A. Scionti. *"FPGA-Based Remote-Code Integrity Verification of Programs in Distributed Embedded Systems"*. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(2):187–200, March 2012.

[3] Pico Computing. *"Pico computing with HMC"*. `https://picocomputing.zendesk.com/hc/en-us`. (Online; accessed 05-June-2018).

[4] P. Cotret, G. Gogniat, J. Diguet, and J. Crenne. *"Lightweight reconfiguration security services for AXI-based MPSoCs"*. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pages 655–658, Aug 2012.

[5] J. Crenne, R. Vaslin, G. Gogniat, J.P. Diguet, R. Tessier, and D. Unnikrishnan. *"Configurable Memory Security in Embedded Systems"*. *ACM Trans. Embed. Comput. Syst.*, 12(3):71:1–71:23, April 2013.

[6] J. Diguet, S. Evain, R. Vaslin, G. Gogniat, and E. Juin. *"NOC-centric Security of Reconfigurable SoC"*. In *First International Symposium on Networks-on-Chip (NOCS'07)*, pages 223–232, May 2007.

[7] S. Drzevitzky, U. Kastens, and M. Platzner. *"Proof-Carrying Hardware: Concept and Prototype Tool Flow for Online Verification"*. *International Journal of Reconfigurable Computing*, 2010.

[8] M. Eckert, I. Podebrad, and B. Klauer. *"Hardware Based Security Enhanced Direct Memory Access"*. In *Communications and Multimedia Security*, pages 145–151. Springer Berlin Heidelberg, 2013.

[9] L. Fiorin, S. Lukovic, G. Palermo, and P. di Milano. *"Implementation of a reconfigurable data protection module for NoC-based MPSoCs"*. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, April 2008.

[10] T. Huffmire, B. Brotherton, N. Callegari, J. Valamehr, J. White, R. Kastner, and T. Sherwood. *"Designing secure systems on reconfigurable hardware"*. 13, 07 2008.

[11] T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, T. Levin, T. Nguyen, and C. Irvine. *"Moats and Drawbridges: An Isolation Primitive for Reconfigurable Hardware Based Systems"*. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 281–295. IEEE Computer Society, 2007.

[12] T. Huffmire, S. Prasad, T. Sherwood, and R. Kastner. *"Policy-driven Memory Protection for Reconfigurable Hardware"*. In *Proceedings of the 11th European Conference on Research in Computer Security*, ESORICS'06, pages 461–478. Springer-Verlag, 2006.

[13] T. Huffmire, T. Sherwood, R. Kastner, and T. Levin. *"Enforcing memory policy specifications in reconfigurable hardware"*. *Computers Security*, 27(5):197 – 215, 2008.

[14] IBM. *"Power8 IBM"*. `http://openpowerfoundation.org/wp-content/uploads/resources/hwarch-caia-spec/content/index.html`, 2016 (accessed online 29-May-2018).

[15] IBM. *"Paderborn Center of Parallel Computing"*. `https://pc2.uni-paderborn.de/about-pc2/`, 2016 (accessed online 5-Nov-2018).

[16] T. Isenberg, M. Platzner, H. Wehrheim, and T. Wiersema. *"Proof-Carrying Hardware via Inductive Invariants"*. *ACM Trans. Des. Autom. Electron. Syst.*, 22(4):61:1–61:23, July 2017.

[17] G. Necula and P. Lee. *"Research on proof-carrying code for untrusted-code security"*. In *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*, page 204., May 1997.

[18] George C. Necula. *"Proof-carrying Code"*. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '97, pages 106–119. ACM, 1997.

[19] S. M. Trimberger and J. J. Moore. *"FPGA Security: Motivations, Features, and Applications"*. *Proceedings of the IEEE*, 102(8):1248–1265, Aug 2014.

[20] T. Wiersema, S. Drzevitzky, and M. Platzner. *"Memory security in reconfigurable computers: Combining formal verification with monitoring"*. In *2014 International Conference on Field-Programmable Technology (FPT)*, pages 167–174, Dec 2014.

[21] Xilinx. *"AMBA AXI-4 Protocols"*. `https://www.xilinx.com/products/intellectual-property/axi`, 2011.

[22] J. Zhang and G. Qu. *"A survey on security and trust of FPGA-based systems"*. In *2014 International Conference on Field-Programmable Technology (FPT)*, pages 147–152, Dec 2014.

Declaration of authorship

Hereby I assure on oath that I have written the present work independently and have used no other than the specified sources as an aid and made quotations indicated.

Place, Date Signature