Analysis of Virtualization through Co-Processor

M.Tech. Thesis

By
RIYA VERMA

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2021

ANALYSIS OF VIRTUALIZATION THROUGH CO-PROCESSOR

A THESIS

Submitted in partial fulfillment of the
requirements for the award of the degree
of
Master of Technology

by

RIYA VERMA

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2021

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **ANALYSIS OF VIRTUALIZATION THROUGH CO-PROCESSOR** in the partial fulfillment of the requirements for the award of the degree of **MASTER OF TECHNOLOGY** and submitted in the **DEPARTMENT OF ELECTRICAL ENGINEERING , Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from JULY 2019 to JUNE 2021 under the supervision of Dr. Vimal Bhatia, Professor, IIT Indore and Sreekanth Modaikkal, Lead Engineer, Sr., QUALCOMM India Private Limited

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

07/06/21
Signature of the student with date
(RIYA VERMA)

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

07.06.2021
Signature of the Supervisor of M.Tech. thesis (with date)

(Dr. Vimal Bhatia**)**

07/06/21
Signature of the Supervisor of M.Tech. thesis (with date)

(Sreekanth Modaikkal**)**

RIYA VERMA has successfully given his/her M.Tech. Oral Examination held on **07/06/2021**.

07.06.2021
07/06/21
Signature(s) of Supervisor(s) of M.Tech. thesis
Date:

Convener, DPGC
Date:07/06/2021

Signature of PSPC Member #1
Date:07 June 2021

Signature of PSPC Member #2
Date: 7/06/2021

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis supervisors Dr. Vimal Bhatia and Mr. Sreekanth Modaikkal for their constant support, patience, encouragement, and immense knowledge. Their guidance helped me in conducting the research and writing of this M.Tech dissertation.

I am thankful to the members of my thesis committee: Dr. Vivek Kanhangad and Dr. Mukesh Kumar for their encouragement and insightful comments. Their questions and suggestions helped me to widen my knowledge from various perspectives.

I am thankful to all the team members at QUALCOMM for their support and enthusiasm. Lastly, I also thank my family for their unceasing encouragement and moral support throughout this journey.

Riya Verma

M.Tech. (Communication and Signal Processing)

1902102002

Department of Electrical Engineering, IIT Indore

Abstract

Virtualization technology was created as a reliable time-sharing option in the early years. It is the most effective strategy to minimize IT costs while enhancing efficiency, speed, flexibility, and scalability in today's world. It is the process of producing a virtual (rather than real or physical) version of something in the field of computing, which encompasses virtual computer hardware platforms, computer network resources and storage devices.

We use Full Hardware virtualization in our work, which is the virtualization of computer system as whole HW platforms, logical abstractions of its componentry, or just the functionality required to run various operating systems. Any software or operating system capable of running on the underlying physical hardware can be run in a virtualized environment (VM). Users can run many guest operating systems at the same time. In this case, the VM simulates enough underlying hardware to run an unmodified guest OS in isolation. Full virtualization provides the finest isolation and security for virtual machines (VMs), as well as making migration and portability easier.

We are using a co-processor to enable Full Hardware virtualization and increase system performance by offloading processor-intensive operations from the main core processor.

The main objective of the work is to utilize Full Virtualization technology to enable secured multi-threading and efficient resource utilization among the VMs using a single hardware i.e., the Co-processor. This is done by allowing separation of resources on 4KB boundary between VMs and designing a hardware block logic inside the co-processor which helps the hypervisor to lock down the data and control path to avoid VMs from impacting one another. We are also adding virtualized interrupts and thus reducing interrupt latency time to process interrupt. We observe in this work that Co-processor in a virtualized environment is reducing CPU overhead and in turn AHB bus bandwidth reduction. As this is hardware controlled rather than software, thus we get better performance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

VM	Virtual Machine
OS	Operating System
HW	Hardware
SW	Software
IT	Information Technology
FPGA	Filed Programmable Gate Arrays
PCI	Peripheral Component Interconnect
GPU	Graphics Processing Unit
HPC	High Performance Computing
SPMD	Single Program Multiple Data
VMM	Virtual Machine Monitor
CPU	Central processing Unit
DMA	Direct Memory Access
I/O	Input Output
RAM	Random Access Memory
CP	Control Path
PTBR	Page Table Base Register
KB	KiloByte
MB	MegaByte
AHB	Advanced High-Performance Bus

MSI	Message Signalled Interrupt
FSM	Finite State Machine
RTL	Register Transfer Level
EDA	Electronic Design Automation

Chapter 1

Introduction

Virtualization technology is thought to have begun in the late 1960s and early 1970s, when IBM spent a significant amount of time and effort building a reliable time-sharing system.

Virtualization has arisen as a software solution that allows many functioning frameworks and programs to run on the same server at the same time. It had a significant impact on the IT technology.

Virtualization is a technique that uses software to construct an abstraction layer over computer hardware, allowing the hardware components of a computer, such as processors, memory, and storage, to be separated into several virtual computers, or VMs. Even though it is only running on a piece of the actual underlying hardware, each VM runs its own OS and behaves like an independent machine. It means that virtualization allows for more efficient use of physical computer hardware and a higher return on the hardware investment made by the industry.

Virtualization is now a common trend in corporate IT infrastructure. It is also the technology that drives the economics of cloud computing. Virtualization can boost IT speed, flexibility, and scalability while lowering costs significantly. Virtualization's benefits include performance, greater workload mobility and resource availability, all of which make it easier to administer and maintain with less cost and ownership.

A co-processor is a computer processor that works in conjunction with the principal processor, the CPU. It helps accelerate system performance by lowering software overhead by offloading processor-intensive tasks off the primary core processor.

But in a multi VM environment, there can be problems faced in the virtualized environment relating to resource allocation and abstraction and thus making it insecure or other VMs to work in the environment and impacting their performance and leading to failure of transaction or collapse of VM.

Therefore, effective management of virtualized environments gives new requirements such as effective resource allocation, de-allocation and secure operation in VM in response to changing application demands. In our work, we are discussing our approach of utilization of full virtualization technology to enable secured multi-threading and efficient resource utilization among the virtual machines using a single hardware i.e., Co-processor.

1.1 Organization of the Thesis

This chapter has given basic introduction to virtualization technology, co-processor and the objective of the work in brief.

The remaining contents are organized as follows:

- Chapter 2 : This chapter includes the literature review of past works and research done in the field of virtualization technology. It also discusses the problem faced in resource allocation in virtualized environment and the motivation behind our work.

- Chapter 3: This chapter includes details about the fundamentals used further in the thesis. It provides detailed knowledge of virtualization technology basics in Section 3.1 and Section 3.2 covers the basics of co-processor.

- Chapter 4 : This chapter provides a detailed description of our proposed approach. It includes the high level diagram and functioning of hardware module, the finite state machine logic, memory interaction.

- Chapter 5 : This chapter includes the Simulation results as Waves from the RTL design.

- Chapter 6: In this chapter, conclusions are made, and a discussion on the possibility of future work is presented.

Chapter 2

Review of Past work and Problem Formulation

This chapter includes a literature review of the Virtualization technology and past work done in the field of virtualization through co-processor. Section 2.2 includes the challenges faced and the motivation behind the work done.

2.1 Literature review

The Virtualization technology has been a widely researched subject with great success, this gives the basis for the our work.

In [1] system's performance of 3 basic hypervisors with one supporting paravirtualization, other container virtualization and last one supporting Full virtualization is compared and evaluated based on various tests conducted by the benchmarking tools. The results obtained showed that different hypervisors show different performances in different tests conducted. Also, it showed that the hypervisor with full virtualization support has comparatively higher system performance than the hypervisors supporting Para virtualization and Container virtualization.

AbdElRahem *et al*. (2016) gives an overview of cloud computing module was given, focusing on virtualization which is one of its enabling technologies, surveying its vulnerabilities and threats, that may be an obstacle in front of further adopting cloud computing, and illustrating some of the countermeasures that are used against these threats and vulnerabilities.

Goel *et al*. (2016) determines the evolution of virtualization during the past decade. It gives a systematic review of virtualization process, development and applications, through a survey of literature and the classification of articles.

Jain and Choudhary (2016) discusses about virtualization, before and after virtualization, its role in cloud computing, briefly about hypervisor, different types of virtualization and its benefits. It concludes that one of the most cost and energy saving and hardware minimizing technique used by cloud providers is virtualization.
In [5] introduction to virtualization of hardware on reconfigurable devices is showed using three main approaches- temporal partitioning, virtualized execution, and virtual machine.
In [6], the aim is to find out the advantages and limitations of current virtualization techniques, analyze their cost and performance. It also depicts which hardware virtualization techniques will able to provide efficient solutions for multiprocessor operating systems. Finally, it presents the current aspects of hardware virtualization which will help large organizations to take effective decision while choosing server with virtualization support.
Perez *et al*. (2008) examines emerging hardware and software virtualization technologies in the context of modern computing environments and requirements. It shows that acceleration of security and virtualization features is expected in the near term. It includes examples showing the acceleration of exits and the propagation of page table entry changes when a guest operating system running in a VM modifies access bits to maintain consistency between VM and hypervisor page table entries.
Vu *et al*. (2014) presents a hardware software co-designed co-processor virtualization support for FPGA-based coprocessors, which are mixed-criticality multicore systems and connected via PCI Express to an Intel multicore platform. It shows an approach that enables the efficient sharing of coprocessors between several VMs by providing each VM its own virtual interface to the coprocessor and moving the handling of parallel existing function accesses into the hardware virtualization layer.
Li *et al.* (2011) proposes a virtualization concept which enables efficient sharing of GPU resources among microprocessors in High Performance Computing (HPC) systems under Single Program Multiple Data (SPMD) execution model. To achieve the desired objective of making each microprocessor effectively utilize shared resources, it investigates the concurrency and overlapping potentials that can be

exploited on the GPU device level. It also analyzes the performance and overheads of direct GPU access and sharing from multiple microprocessors as a comparison baseline.

Huang and Hsiung (2009) proposes a virtual hardware mechanism to further raise the utilization of reconfigurable hardware functions. Furthermore, we also propose a unified communication mechanism to further enhance the scalability of the design, without losing the generality in accessing hardware designs. Its result system performance can be also further improved when the virtual hardware mechanism is

Chaoub *et al.* (2021) discusses the challenges faced while designing and deploying network constraints for providing broadband connectivity to rural areas and proposes novel approaches and solutions to bridge this digital gap. In the service provisioning for rural regions section, it discusses the possibility of adoption of open/virtualized/cloud-based solutions for rural network operators.

2.2 Motivation/Objective

Figure 2.1 depicts the generalised high level diagram of the virtualized environment without the co-processor. In a virtualized environment, we have multiple VMs, each VM having its own underlying OS with applications supported by it. These VMs are seperated from the underlying hardware by a layer of Hypervisor, which helps co-ordinate between them.

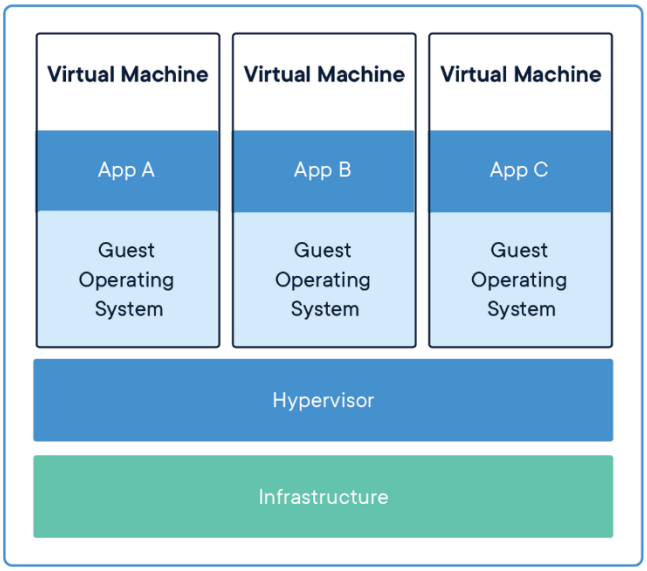

Figure 2.1 : High level diagram of generic Virtualized environment[22]

Figure 2.2 shows the high level diagram of virtualized environment with co-processor assissting the Hypervisor. In Full Virtualization, these VMs are independent of one another, having each of its own intended functionality. These VMs are going to access the same main core processor at the same time asking for the hardware resources like control and data paths, to perform their functions from the instruction set.

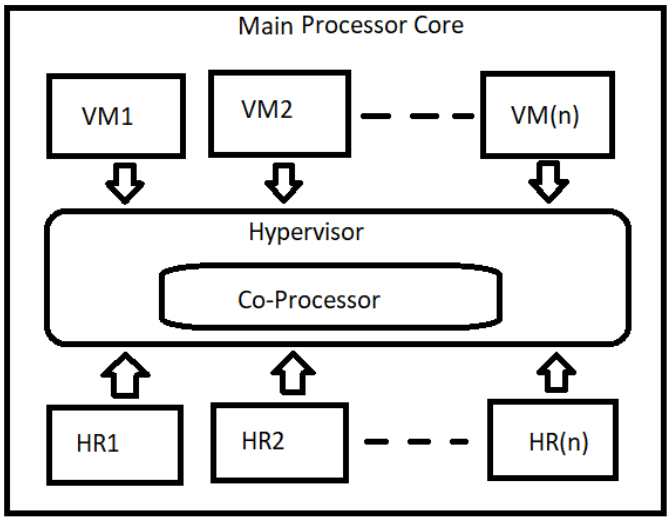

Figure 2.2: High level diagram of Virtualized environment with co-processor

The problem arises in this process as each VM in full virtualized environment thinks that it is the master and have access to all the underlyig hardware and compute cycles and is unaware of the need of other VMs. So, a scenario may arise when these VMs can interfere with other VMs and impinge their compute cycles or the harware resources they are using. In a normal scenario, there is only the Hypervisor present that acts as the interface between the underlying hardware and VMs, giving them the resources they need and co-ordinating between them. In virtualization through co-processor scenario, the co-processor i.e the harware is assissting the hypervisor software in this process of resource allocation, de-allocation and monitoring. So, while sharing these resources we cannot tolerate this interference as this violates the inteded function operation and also impacts the safey and security in certain applications.

This challenge to enable secured transaction and efficient resource allocation in virtulization, where VMs do not impact or interefere with one another gives the motivation to design a hardware module in the co-processor that helps in faster and secure operation, further discussed in Chapter 4.

Chapter 3

Background

In this chapter, a detailed discussion about some important topics is done that will help to understand further sections.

3.1 Virtualization

Virtualization is the technology or process of creating a software-based, or virtual (rather than actual) representation of something, such as virtual applications, servers, storage, and networks. Virtualization technology utilizes software to create an abstraction layer over computer hardware that allows the hardware elements of a computer such as processors, memory, storage etc. to be divided into various virtual computers, commonly called virtual machines (VMs).

Before learning more about virtualization, we should know the difference between traditional physical architecture and virtual architecture. Figure 3.1 shows the differences between a traditional and virtual architecture difference at a high-level.

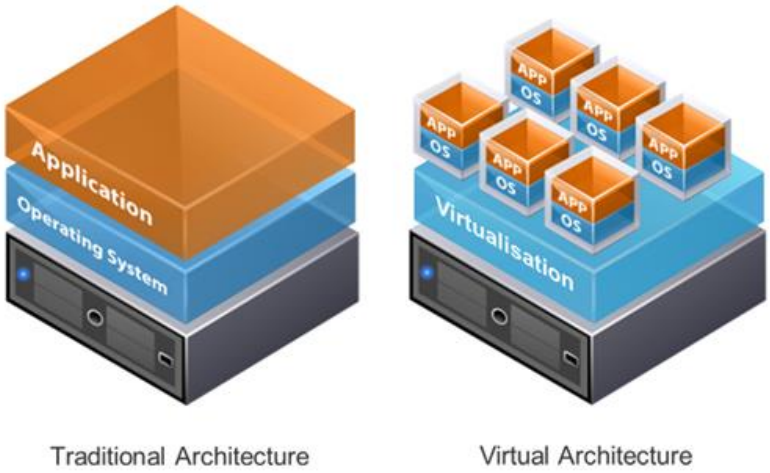

Figure 3.1: Difference between traditional and virtual architecture [25]

In traditional architecture, the operating system is placed directly on hardware components, such as a rack-mount server or a blade server. The OS supports a wide range of apps. Only the physical CPU and memory resources can be allocated. It uses a physical network adapter to send and receive data. If the administrator wants to allocate extra physical resources, such as the CPU core, RAM, network adapters, and so on, the hardware must be upgraded.

The OS is loaded on the hardware in Virtual architecture via a thin layer of software called the virtualization layer or hypervisor. Each virtual machine's physical hardware resources are dynamically allocated by the hypervisor. It is not required to upgrade the hardware if the administrator wants to allocate more memory resources to the virtual machine. To sum up, virtual architecture is more flexible than traditional architecture.

3.1.1 Motivation for virtualization

Due to the limitations of x86 servers, many IT organizations have to deploy multiple servers, each operating at a fraction of their capacity, to keep pace with today's high storage and processing demands. This results in huge inefficiencies and excessive operating costs.

This necessitates the use of virtualization. Virtualization uses software to build a virtual computer system by simulating hardware functions. IT businesses can now host several virtual systems, as well as numerous operating systems and applications, on a single server. Economies of scale and increased efficiency are among the advantages that result.

3.1.2 Virtual Machines

A virtual machine (VM) is a software-based operating system (OS) or application environment that imitates dedicated hardware. On a physical hardware system, it has its own virtual CPU, memory, network interface, and storage. They are software-based virtual environments that replicate a physical computer. They are composed of

files that hold the VM's setup, storage for the virtual hard disc, and snapshots of the VM's state at a certain point in time.

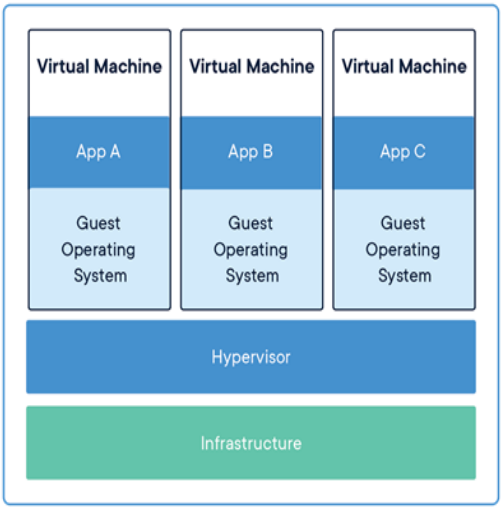

Figure 3.2: Virtual machines on virtualized environment [24]

Virtual Machines' Key Features:

• Partitioning – Run several operating systems on a single physical computer and distribute system resources amongst VMs.

• Isolation—At the hardware level, it provides fault and security isolation.

• Encapsulation- Save the full state of a virtual machine to files and move or transfer them as easily as files.

• Hardware Independence: Any VM can be provisioned or moved to any physical server.

3.1.3 Hypervisor

A hypervisor, also known as Virtual Machine Monitor (VMM), is the software layer that coordinates VMs. It serves as an interface between the VM and the underlying physical hardware, ensuring that each has access to the physical resources it needs to execute. It also ensures that the VMs do not interfere with each other by impinging on each other's memory space or compute cycles.

There are two types of hypervisors:

- **Type 1 hypervisors, often known as "bare-metal" hypervisors**, run directly on the physical hardware of the underlying computer, interfacing directly with its CPU, memory, and physical storage, and do away with the traditional operating system entirely. That's why they're known as bare-metal hypervisors. They are quite efficient, but they require a separate system to manage and administer the several virtual machines as well operate the host hardware.

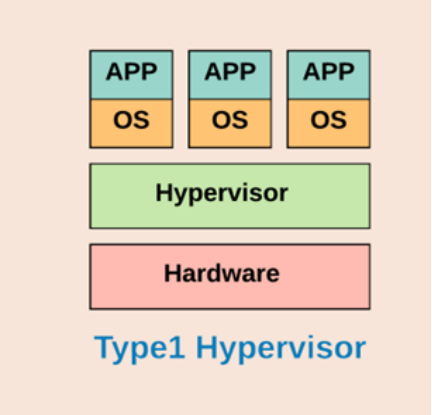

Figure 3.3: Type1 Hypervisor [23]

- **Type 2 hypervisors** run as an OS application rather than directly on the underlying hardware. They increase end-user productivity by allowing quick and easy access to a secondary guest OS in addition to the primary one running on the host system. However, there are difficulties with latency, performance, and other potential security risks.

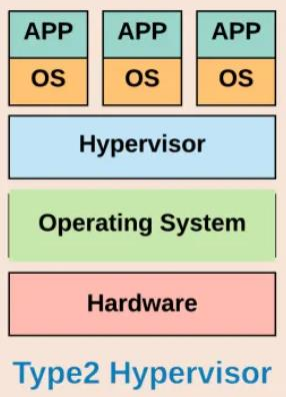

Figure 3.4: Type2 Hypervisor [23]

3.1.4 Virtualization working

Physical resources and virtual environments are separated by hypervisor software. Hypervisors can be deployed directly on hardware or on top of an operating system. Hypervisors divide up your physical resources so that virtual environments can access them.

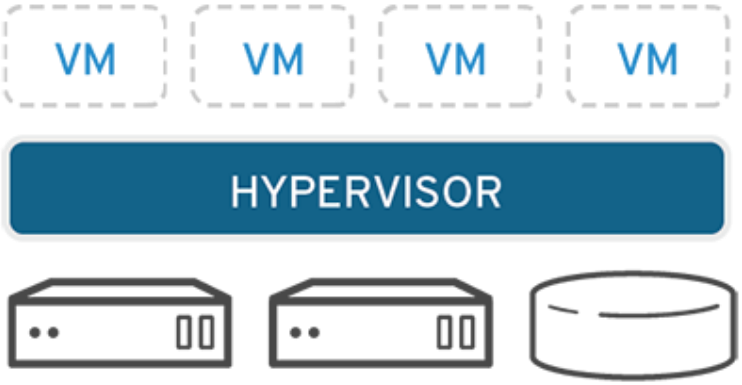

Figure 3.5: Virtualized environment [20]

The physical environment's resources are partitioned to the several virtual environments as needed. Within the VM, users interact with it and conduct computations. The virtual machine is a single data file that runs. When a user or program sends an order that demands additional resources from the physical environment while the virtual environment is running, the hypervisor communicates the request to the physical system and implements the adjustments, all at near-native speed.

3.1.5 Types of Virtualization

Depending on the type of application and hardware used, virtualization can take many different shapes. The following are the most common types:

- **Hardware Virtualization**

  Virtualization of computers as whole hardware platforms, logical abstractions of its componentry, or just the functionality required to run various operating systems is known as hardware virtualization.

- **Desktop virtualization**

  Desktop virtualization allows numerous desktop operating systems to operate on the same computer, each in its own virtual machine. It allows users to view a desktop remotely from a connected device by simulating a workstation load. This distinguishes the desktop environment and its programmes from the device that is used to access them.

- **Network virtualization**

  Network virtualization use software to generate a "view" of the network that allows an administrator to control the network from a single console. It abstracts hardware pieces and operations (for example, connections, switches, and routers) into software that runs on a hypervisor.

- **Server Virtualization**

  Server virtualization is the technique of dividing a single server's resources into many virtual servers. These virtual servers can function independently. Server virtualization enables enterprises to use a single (host) server to run numerous independent operating systems (guests or virtual) with varied configurations. The technique also saves organizations money on the hardware costs of maintaining a large number of physical servers, allowing them to optimize their server architecture.

- **Storage virtualization**

  Storage virtualization allows all storage devices on a network to be accessed and managed as if they were only one. Storage virtualization,

in particular, consolidates all storage blocks into a single shared pool from which they can be allocated to any VM on the network as needed. Storage virtualization simplifies the provisioning of storage for virtual machines and makes the most of all available storage.

- **Data virtualization**

  Between the applications that access the data and the systems that store it, data virtualization tools build a software layer. The layer converts a data request or query from an application as needed and returns results that can span several systems. Any program can access all of that data, thanks to data virtualization, regardless of its source, format, or location.

- **CPU virtualization**

  CPU virtualization divides a single CPU into many virtual CPUs that can be used by numerous VMs.

- **Cloud virtualization**

  Virtualization is key to the cloud computing model. Cloud computing companies can offer a variety of services to consumers by virtualizing servers, storage, and other physical data centre resources. Virtualized server, storage, and network infrastructure as a service (IaaS). Virtualized development tools, databases, and other cloud-based services are available as part of the Platform as a Service (PaaS) model. Software as a service (SaaS)-software application used on the cloud.

3.1.6 Virtualization Techniques

- **Full Virtualization**

  The term "full virtualization" refers to a virtualization technology that creates an environment that totally mimics the underlying hardware. Any

software that can run on physical hardware can be run in the VM, and any OS that the underlying hardware supports can be run in each individual VM. Users can run many guest operating systems at the same time. In addition, the VM simulates sufficient hardware to run an unmodified guest OS in isolation.

Full Virtualization is achieved through a mixture of binary translation and direct execution, in which the guest OS is totally isolated (totally detached) from the underlying hardware via the virtualization layer. Because the guest OS is unaware that it is being virtualized, no changes are required.

Because the same guest OS instance can run on virtualized or native hardware, it provides the highest isolation and security for VMs and simplifies migration and portability.

Virtualization products from VMware and Microsoft Virtual Server are two examples.

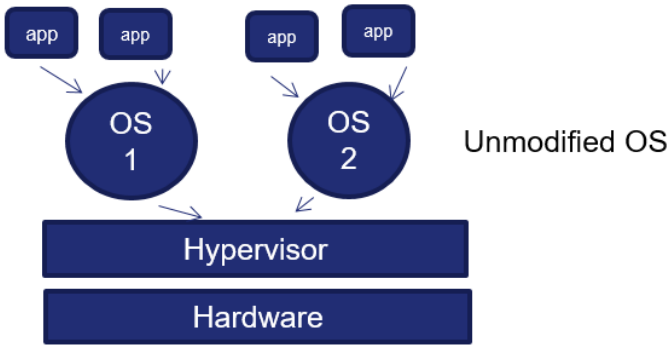

Figure 3.6: Full virtualization

- **Para Virtualization**

Paravirtualization is different from full virtualization, where the unmodified OS does not know it is virtualized and sensitive OS calls are trapped using binary translation. It entails altering the OS kernel, as seen in Figure 3.5. Depending on the workload, the performance advantage of paravirtualization versus full virtualization varies.

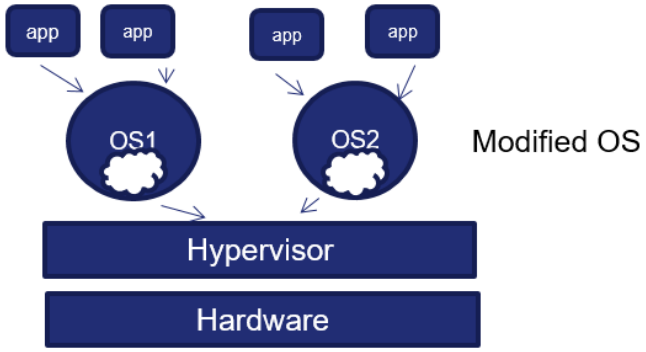

Figure 3.7: Para virtualization

- **Hardware-assisted Virtualization**

  It is a virtualization method that uses hardware capabilities, primarily the host processors, to enable efficient complete virtualization. It takes use of the physical components of a computer to support the software VMM, which is used to manage virtual machines. This strategy boosts performance by minimizing software workload.

3.1.7 Advantages and disadvantages of virtualization

Virtualization gives several benefits in the field of computing, such as:

- **Resource optimization:** Prior to virtualization, each application server had its own dedicated physical CPU, which was also underutilized. Virtualization, on the other hand, allows you to run many apps on a single physical machine. This allows for the most efficient use of the computational capabilities of the physical hardware.

- **Consolidation:** If various applications only require a limited amount of computing power, the administrator can combine several computers into a single server that runs several virtual machines (VMs). As a result, costs will be lowered because there will be less floor and rack space, as well as less heat and power use.

- **Minimal downtime:** Crashing the operating system or an application can result in downtime and impair the user's productivity. Admins can operate multiple redundant VMs in parallel and swap between them if a problem emerges.

- **Easy Migration:** In virtualization, if the guest system's hardware fails, it can be relocated to a healthy server with minimal downtime.

- **Security benefits**: Infected VMs can be rolled back to a moment in time when they were clean and stable, known as a snapshot.

Disadvantages of Virtualization

Virtualization also presents some challenges:

- Type 2 hypervisor running on a host OS is vulnerable to attack. If a hypervisor is compromised, all virtual machines and guest operating systems are potentially theirs. Because hypervisors can allow virtual machines to connect without using the physical network, it can be difficult to view their traffic and hence detect suspicious activity.
- Malware is typically firmly interwoven into the core components of the OS, persisting beyond system rollbacks, therefore disinfecting a non-virtualized OS is not always possible.
- Between the OS and the HW, virtualization creates some performance latency.

3.2 Co-processor

A coprocessor is a computer processor that supports the primary processor's functions (the CPU). Floating point arithmetic, graphics, signal processing, text processing, cryptography, and I/O interfacing with peripheral devices are examples of operations performed by the coprocessor. Coprocessors can improve system performance by offloading processor-intensive activities from the primary core processor. They have varying degrees of autonomy. Some of them rely on coprocessor instructions contained in the CPU's instruction stream for direct control. Others are asynchronous processors that can work independently. Direct memory access (DMA) is commonly used, with the host CPU constructing a command list. A coprocessor is typically connected to a core processor and enhances the processing capabilities of the core by expanding the instruction set or providing customizable registers.

Chapter 4

Proposed Approach

In this Chapter, we provide the details of our proposed approach while doing the RTL design of the HW secure block. In the proposed method, we discuss the current co-processor design, hardware module placement, its high level diagram, the FSM and its interaction with the RAM. In the end, we discuss the access legality and failure.

4.1 Co-processor and placement of Hardware secure Block

We have proposed the design of a Hardware secure block module in the Co-processor. This Hardware secure block module will provide means for hypervisor for allocation and deallocation of hardware resources to VM. It will enable locking down the resources for VMs so that do not interfere one another. Thus, giving us faster and secure operation.

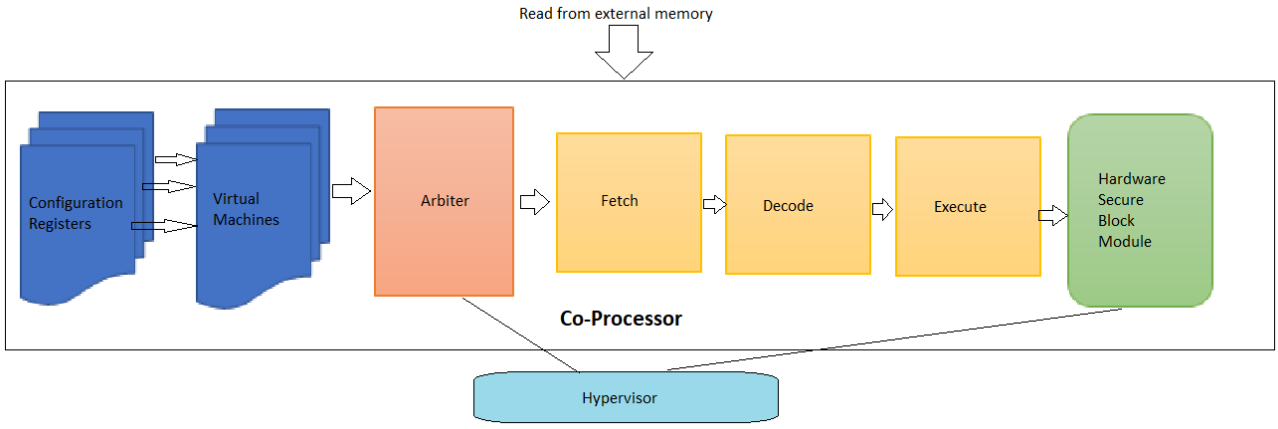

Figure 4.1: Top level architecture of Co-Processor with Hypervisor

Figure 4.1 shows the top-level architecture of the co-processor with the hypervisor registers. In a multi VM environment, each VM has an intended function and are independent from one another. These VMs need to access the same main core processor for its operation. The Co-processor helps the main processor to supplement the applicable tasks for the VMs. The software pushes the data in the configurable registers for each VM by software register programming. These registers have control and data path information for the VM to perform its operation. The VMs in our design has multi -threading operation for parallel and faster operation. The arbiter then generates arbitration winner among threads for each VM based on priority-based arbitration scheme. The Fetch module will fetch the command data like address, opcode, size etc. for the arbitration winner thread VM. This data is given to Decode module which decodes the opcode and abstracts data to send to Execute module to execute the intended operation of updating destination interface registers. Arbiter and Hardware secure block module are controlled by Hypervisor software registers.

Figure 4.1 also shows the placement of the Hardware Secure block in the existing Co-processor architecture. It is placed after the execute module, to act as an additional process between co-processor and outer modules.

4.2 Hardware secure block module

To enable secured multi-threading operation via Hardware secure block module, separation of VM programming registers on separate 4KB boundaries to allow for VM separation is done. The hardware resource of each type is on 4KB boundaries to allow for resource separation. Logic in the hardware secure block module is added to control the co-processor access to different register spaces (needed to be controlled for virtualization) that enables/disables access to 4KB boundary. Similarly, it will provide means for hypervisor to lock down the data path to avoid guest VMs from impacting one another i.e., limit or locking the resources the VM has access to.

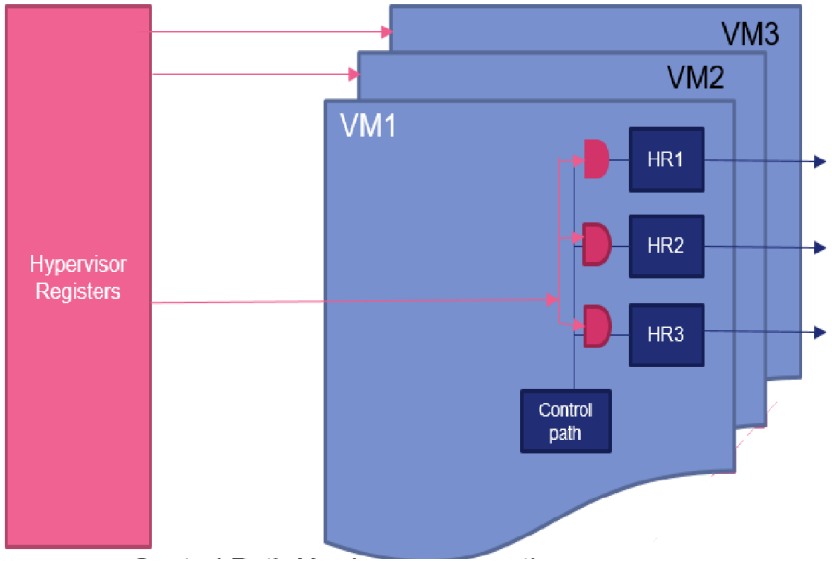

Figure 4.2: Control path hardware allocation

Figure 4.2 shows that each VM say VM1, VM2 so on has control path allocated to it by software and then a separate set of registers programmed by hypervisor would allow each VM to reserve the hardware resources for the appropriate data path.

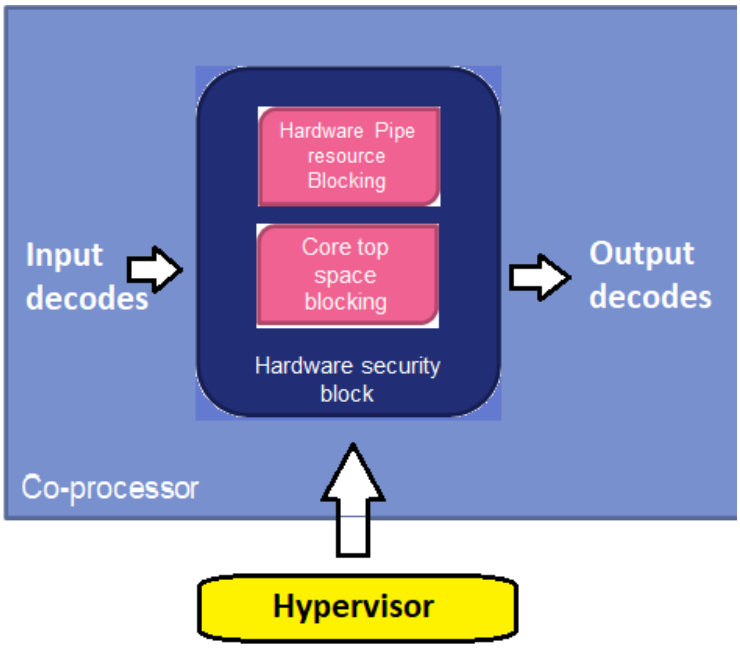

Figure 4.3: Top level diagram of Hardware secure block

Figure 4.3 shows the top-level diagram of Hardware secure block module. It is designed to work on two levels- Hardware Pipe resource blocking and Processor core top space blocking. It takes input decodes of the hardware resources and register spaces form the co- processor and hardware resource allocation/de-allocation and top core access permission data from the Hypervisor. The output of the block is final decode signals for the various resources to the output interface.

4.2.1 Hardware Pipe resource blocking

In our design we have pipe hardware resources to perform data processing at the source and destination level. At this level of Hardware secure block module, we are reserving or flushing the pipe hardware resource for the VM.

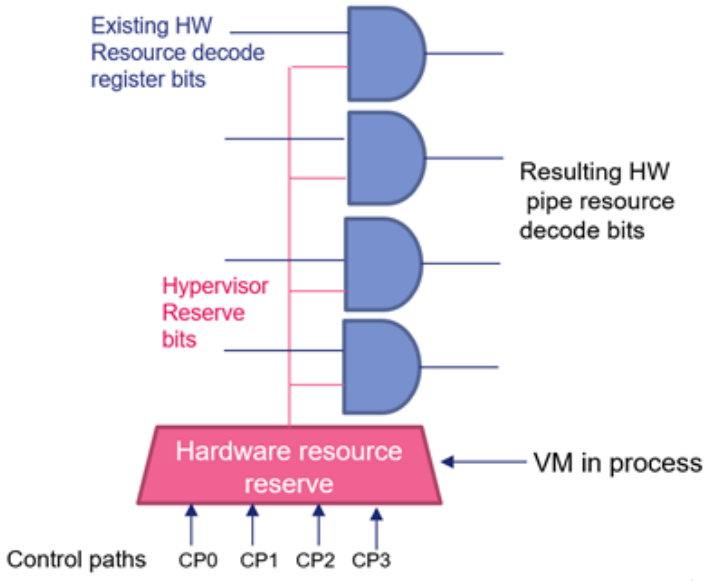

Figure 4.4: Hardware pipe resource blocking logic

Figure 4.4 shows the Hardware pipe resource blocking logic we have designed in the Hardware secure block module. We logically AND the existing hardware decode register bits coming from inside the co-processor with the Reserves bits from the

Hypervisor register for the respective multiplexed control path of the VM in process. The software maps the appropriate control path to the corresponding VM. Say, VM0 is given CP0 and has reservation for hardware resource pipe 1 and 2. With this logic, the VM can use the hardware resource allocated to it. Any attempt to use any other resource reserved for other VM's control path will be suppressed/ blocked by the hardware. This logic is implemented in the RTL design of the module.

4.2.2 Processor core top space blocking

The hypervisor will control the Co-processor access to Processor Core space for each VM via configuration registers. Each bit in the register defines permission to corresponding 4KB address space i.e. access permission bits.

Figure 4.5 shows the RAM structure of the access permission table of the VMs. For 1MB address space, we require 8x32 bit registers. To save area cost the HW implementation will use 40x32 RAM with 8 address off-set for each VM's Control path. Each bit of the RAM will correspond to permission for Co-processor to access the 4KB page.

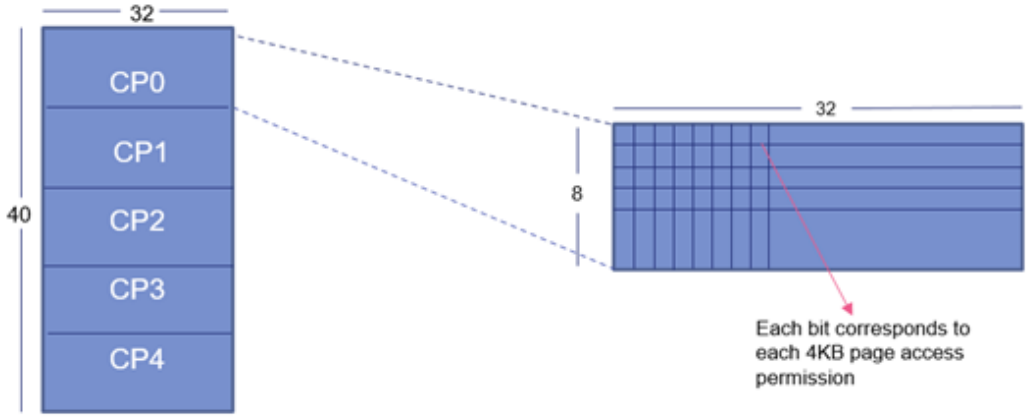

Figure 4.5: Processor core top access permission table RAM

4.2.2.1 RAM size calculation

We have main address space of 1MB. All the resources are placed at 4KB boundaries. So, the No. of 4KB pages = 1MB ÷ 4KB = 256 pages. No. of

address location for 32-bit data write enable RAM=256 ÷ 32= 8 address locations. So, for each VM's CP, we need 8x32 bit register. The No. of VM we have are 5. So, RAM size= 40x32 (since 8x5=40). We have generated a RAM of size 40x32 using the memory compiler.

4.2.2.2 Working

For incoming transaction from the VM in process, the HW secure block will take either 1 or 2 clock stages to decide on access permission-

- In first stage, from the incoming transaction address, it will use Address bits [19:17] + CP Offset to go to the RAM location for the VM and read the access permission table content data.
  If the Address bits [19:17] + CP Offset of incoming transaction is same as previous transaction, i.e., that VM's permission bits have already been read, then this stage will be bypassed.
- In second stage, it will use Address bits [16:12] to select that 4KB page permission from the 32-bit access permission table data, that were read from previous stage. Based on permission bits, HW will decide whether to block or grant the permission. Figure 4.6 shows the above explained working of incoming transaction with the HW secure block.

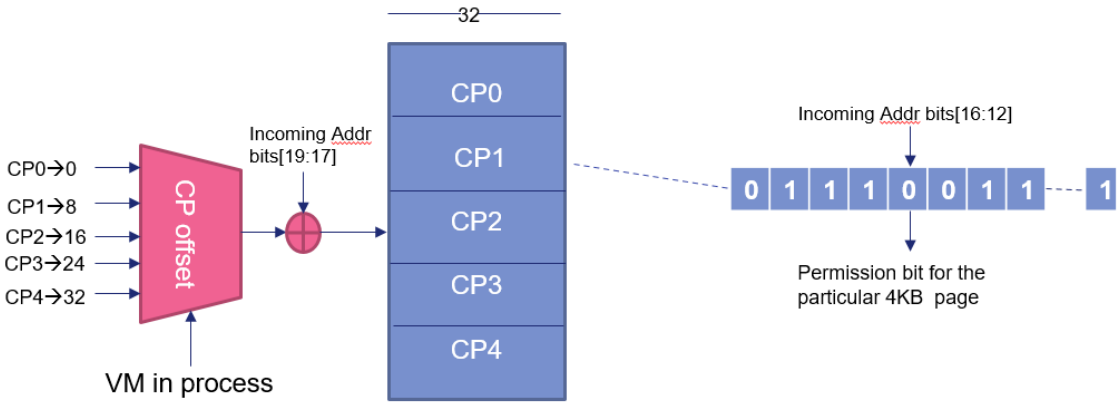

Figure 4.6: Working of HW secure block in top core space blocking

4.2.2.3 Paging hardware resource access

We have a 32-bit incoming transaction address from the core processor for the VM. The hardware resource uses pages virtual memory of 4KB pages. It is of 2 parts- Page Number and Page offset. Page Number specifies the specific page of the process from which Core processor wants to read the data. Page Offset specifies the specific word on the page that Core processor wants to read.

Table 4.1 shows the number of bits for page number and the page offset. For 4KB page, the offset field contains 12 bits (2^{12} =4K) . The remaining 20 bits are page number bits.

Page number	Page offset
20	12

Table 4.1: Decomposition of incoming address

Page table is used by virtual memory address translation. It is structured array in memory and indexed by page number. It concatenates the frame number with the offset part of incoming address to form physical address to access main memory. The base address for the current process's page table, which is handled by the OS, is stored in PTBR. Figure 4.7 shows paging to access HW resource page in main memory.

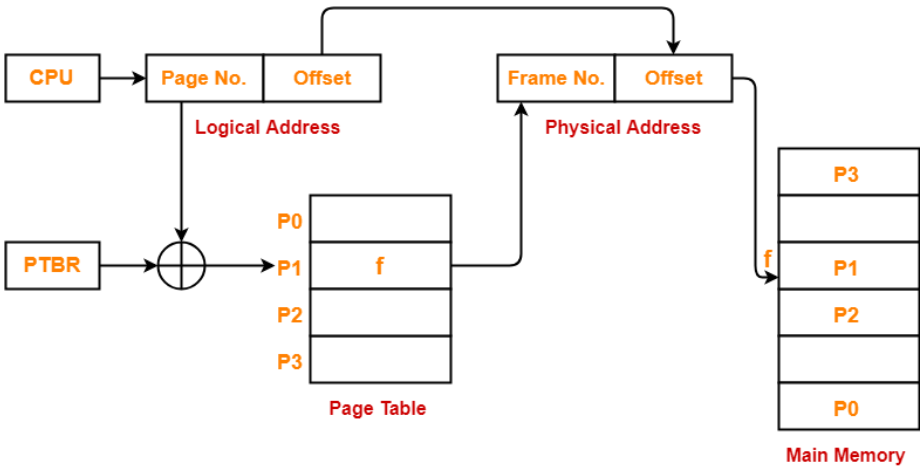

Figure 4.7: Paging to HW resource in main memory [11]

4.2.2.4 Access legality

The bits that control access legality are defined to the co-processor and written by the Hypervisor. The Co-processor does not modify these bits. It just reads them to decide if it should attempt an access. The valid bit indicates if the Co-processor should trap to the operating system when the page is accessed. The access permission bits controls access to the page depending on the attempted co-processor operation. Access is allowed only if the valid bit is 1 and the appropriate access permission bit is 1.

- Access permission Bit = 0x0 → Block access to corresponding 4KB resource page
- Access permission Bit = 0x1 → Grant access to corresponding 4KB resource page

4.2.2.5 Processor Core space blocking Finite State Machine

We have designed a Finite State machine to implement the operation of the HW secure block in processor core space blocking. Table 4.2 shows the state table of FSM.

	FSM state	Description
00	IDLE	Idle state
01	WRITE	RAM single location write
10	PREFILL	Prefill permission table
11	READ	Read permission table

Table 4.2: FSM state table

State 00 – IDLE is the state when machine is in ide state and not doing any operation.

State 01- WRITE is the state when the Hypervisor software writes to a single location in the permission table with the reserve bits.

State 10- PREFILL is when prefill trigger is received and the machine jumps to this state and fills the permission table (8x32 bits) of the VM in the RAM with pre-decided value.

State 11- READ is the machine state when it must read the permission bits from the particular location in the RAM according to address bits of incoming transaction.

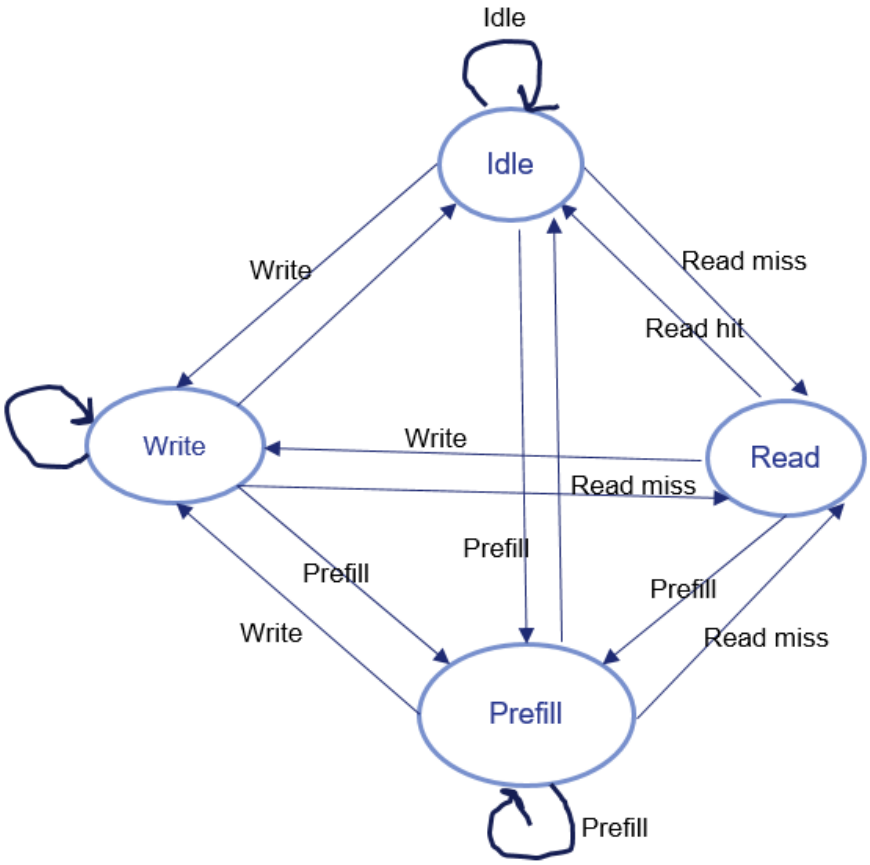

Figure 4.8: State diagram of FSM

Figure 4.8 shows the State diagram of the Finite State Machine. In the beginning, the machine is in IDLE state and will continue to be in it if no input condition for other state is met. The highest priority state is WRITE. If the input signals satisfy the write condition, then the FSM goes to WRITE state and writes the 32 bit data received from the hypervisor to the single location in the RAM as permission bits. The next priority state is PREFILL. As soon as the prefill trigger from the VM in process is received and condition is met, the FSM jumps to PREFILL state. It will prefill the predefined data in the 8x32 bit register location in the RAM till the prefill counter = 111. When done signal is generated it moves to next state. If the condition for read miss is met i.e., when FSM hasn't read for a particular address, then the FSM goes to READ state and reads the 32-bit data from

that location of RAM. After if no other condition is met then moves to IDLE and generates read hit signal for that address. If the same address comes and Read hit signal is already done then no need to read from that location and bypass that state and goes to IDLE.

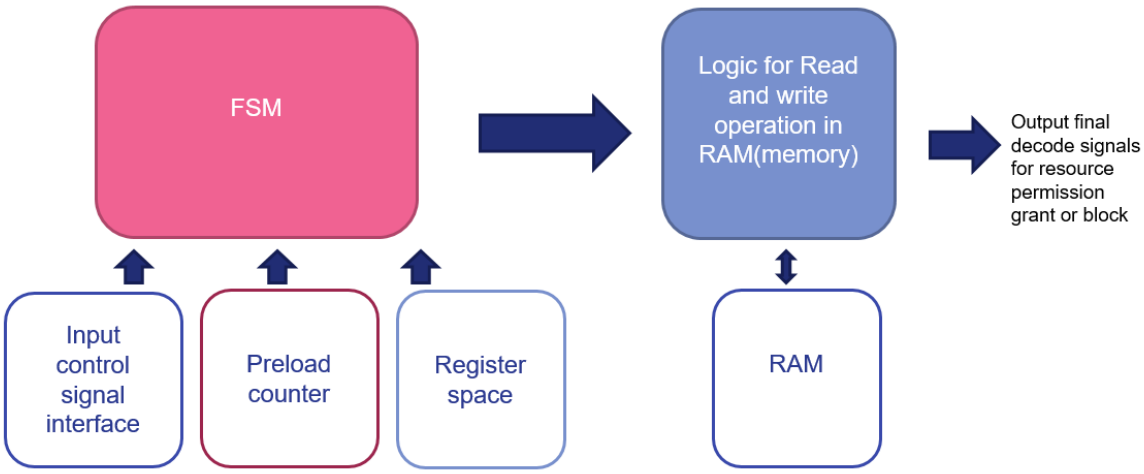

Figure 4.9: FSM logic interaction

Figure 4.9 shows FSM logic interaction with all the interfaces. The Finite State Machine takes input from the control signal interface, where it gets VM in process, incoming transaction address and read/write trigger information. Also, it interacts with the Register space to get prefill trigger and prefill value. It synchronizes with the prefill operation with the prefill counter. The FSM generates control signals for each state to perform state specific task and given to combinational logic for read/write operation to interact with the RAM.

4.2.2.6 RAM memory interaction logic

The FSM will generate control signal to perform each RAM interaction operation of write, read, and prefill through combinational logic design in RTL.

- Write control signal from FSM will perform the following interaction in RAM-

Ram write enable =1’b1

Ram write address=Incoming transaction Address bits [19:17] + CP Offset

Mem[Ram write address]= input 32-bit data

- Prefill control signal from FSM will perform the following interaction in RAM-

  Ram write enable =1’b1

  Initial Prefill count=1’b0

  Ram write address=Incoming transaction Address bits [19:17] + CP Offset

  Mem[Ram write address]= 32-bit prefill value copy

  Prefill count=Prefill count+3’d1;

  Prefill done=(Prefill count==3'b111)?1'b1:1'b0

- Read control signal from FSM will perform the following interaction in RAM-

  Ram read enable=1’b1

  Ram read address= Incoming transaction Address bits [19:17] + CP Offset

  Output read data=mem[Ram read address]

  Page select= Address bits [16:12]

  Final output decode signal to decide whether permission to access is received or blocked is obtained by AND of the output read data from RAM and page select bits and ORing the vector.

  Output access grant or fail=|(Output read data & (32’d1 << page select))

4.2.3 Access failure

If an access is not allowed based on the access legality bits then the co-processor just traps to the operating system. The operating system invokes its

handler for memory access failures. The handler determines whether the error was due to an illegal address in the currently executing program or a page that is not currently in memory. For debug purpose, the address and hardware resource space information about permission to access fail transaction will captured by HW and interrupt will be generated.

Chapter 5

Results and Discussion

The simulated result waveforms of the RTL design of the hardware module are shown in this chapter. Also the impact on speed, latency and Bandwidth is discussed.

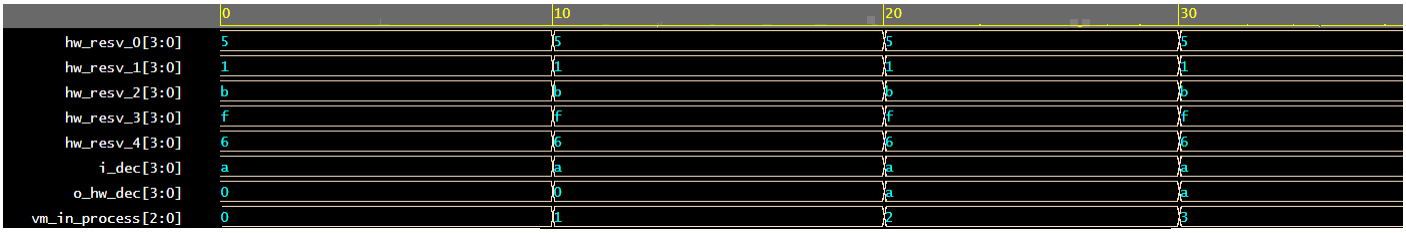

Fig 5.1: Simulation result waves for Hardware Pipe Resource Blocking

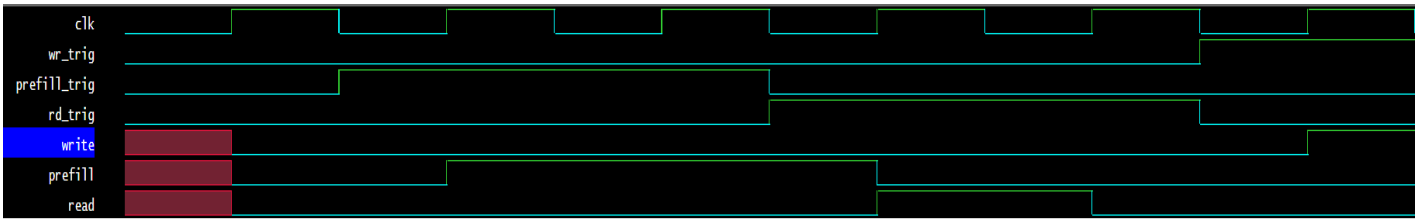

Fig 5.2: Simulation result waves for Processor Top Core Access Finite State Machine

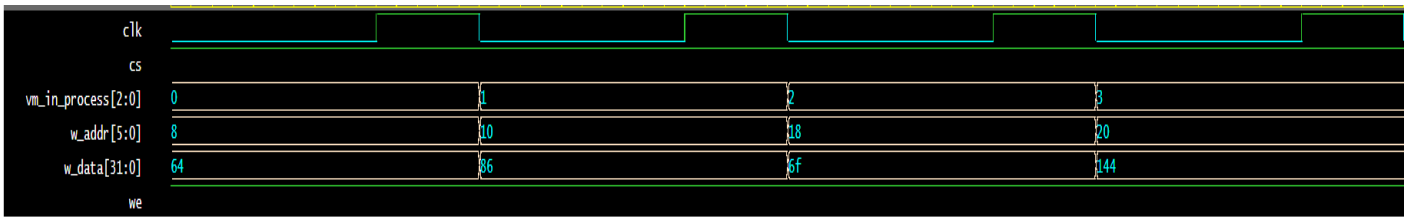

Fig 5.3: Simulation result waves for WRITE State in FSM

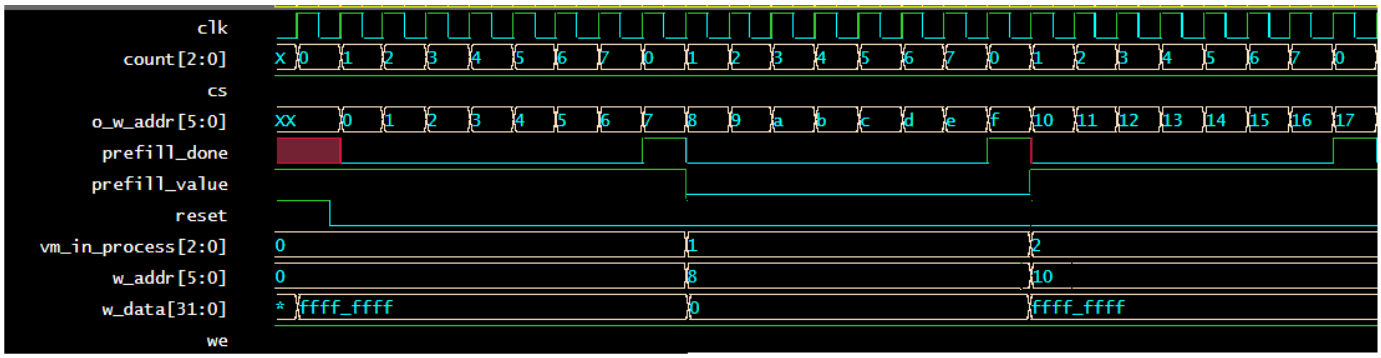

Fig 5.4: Simulation result waves for PREFILL State in FSM

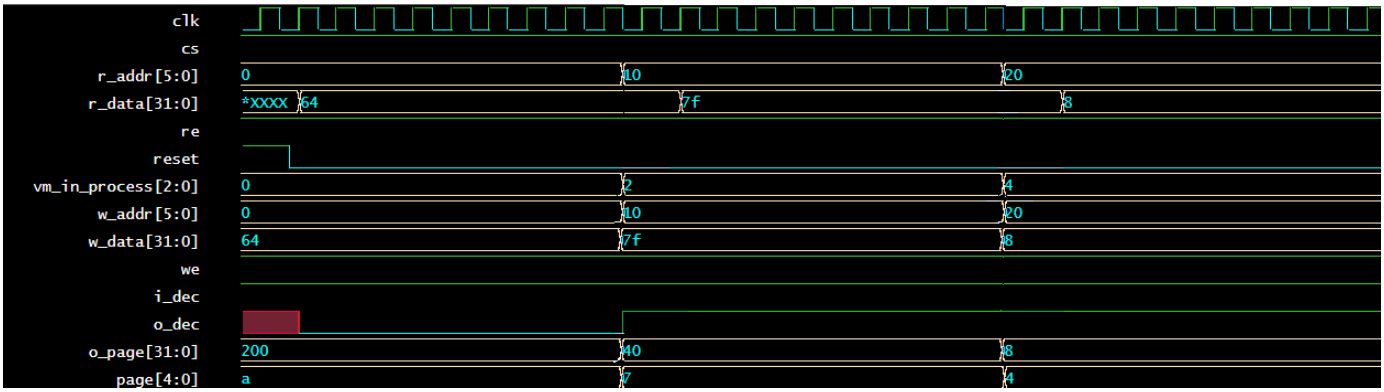

Fig 5.5: Simulation result waves for READ State in FSM and final output decode

The main idea of using a coprocessor in our design is to assign a computing intensive task to it, so as to unload the CPU. The advantage of this approach decreases significantly if the main processor CPU has to check the status of co-processor computations is done or not. To avoid this situation, the use of virtualized interrupts is done in our design. Distinguishable or unique interrupts are assigned to each of virtual machine interfaces, since the co-processor is shared among the VMs. By doing this, we can identify the connection of a particular interrupt and the assigned Virtual machine interface. In case this is not done, the interrupt handler module has to ask the co-processor about the chosen VM, to process the interrupt. This extra step will increase the total latency of the process. Thus , this is undesirable.

In our design we use Message-Signaled Interrupt (MSI) mechanism. It is an alternative to line-based interrupts. Instead of using a particular dedicated pin to trigger an interrupt, MSI requires devices to trigger an interrupt by writing a data to a software-controlled memory address. In our design, we will use this data to differentiate between the different interrupts. These data is contained for each of the VM interface in the hypervisor software. To generate separate unique interrupts for each VM interface, the MSI data register of device is extended to the core register set that stores other MSI data also. These registers are written by the Hypervisor software. Along with this, we have additional HW called interrupt generator or

controller. When the interrupt has to be generated, the control unit will output the interrupt generation request along with the SID of the chosen VM to the Interrupt controller. It will generate the interrupt message and the hypervisor will determine which VM it is addressed to and forward the interrupt to that VM. Without Virtualization, a single Interrupt controller sits in main processor but in our design the interrupt controller is encapsulated inside the co-processor, shared between VMs and thus reduces interrupt latency time for interrupt process and gives better VM to VM communication.

In our work, latency measurements are done to show the performance advantages of the proposed virtualization approach with respect to memory mapped I/O operations. In each of the accesses issued by software running on the CPU with clk frequency of 1.6 GHz, 32 bits of data are to be written to / read from a 32 bit wide register of the coprocessor. The register resides in the control unit and is accessible via the memory-mapped interface in an only Software-based Virtualization system with shared resources and respective virtual interface in Hardware Supported Virtualization System with shared Resources, both in multi-VM environment.

It is observed that virtualization process using a completely software-based approach causes an overhead of 57.14% for read accesses and 72% for write accesses compared to the Hardware supported Virtualization technique. So, in the Hardware supported virtualization, overheads are saved with respect to software virtualization. This in turn is saving clock cycles and give us the speed up factor of 2.33 for read access and 3.57 for write access.

In a case where co-processor hardware is not used, when multiple VM is running, then whenever in between transaction, resource allocation change is detected, then the software must go and program for each resource, control and data path allocation for each VM. But when using a Co-processor in a virtualized environment, only the Resource permission table's valid bits for the multiple VM must be programmed and the rest of functioning is the same. Thus, a Co-processor in a virtualized environment is reducing CPU overhead and in turn AHB bus bandwidth reduction. As a result, we save clock cycles by using the Hardware support in virtualization, thus reducing

latency. As this is hardware controlled rather than software, overhead on the software (hypervisor) is reduced thus we get better performance in terms of speed.

Chapter 6

Conclusion and Scope for future work

In this work, the RTL design of HW secure block to enable secured multi-threading and efficient resource allocation among VMs using Co-processor is done. Our approach enables to have secure and faster operation with efficient resource allocation in a multi-VM environment using a hardware secure logic in Co-processor. The design is compiled free of errors from the Synopsys SpyGlass lint tool and successfully verified with complete regression and validated. The simulated waveforms snips from the EDA tool are added in previous chapter.

The goal of virtualization and security in VM is to build secure foundations of computing by integrating isolated and trusted technologies to enable verifiable confinement and trust features across large, dispersed settings. Hardware provides the foundation for these properties in such situations. Acceleration of security and virtualization features in Heterogenous multicore processors is predicted to be in demand in the future at the processor and chip level. Future development in this area could support a growing number of peripherals. So, future work might be done with the following goals in mind: how to grow the computer system's scale and increase utilization and real-time performance of resource system.

REFERENCES

[1] A. B. S., H. M.J., J. P. Martin, S. Cherian and Y. Sastri, System Performance Evaluation of Para Virtualization, Container Virtualization, and Full Virtualization Using Xen, OpenVZ, and XenServer, 2014 Fourth International Conference on Advances in Computing and Communications, Cochin, 2014, pp. 247-250, doi: 10.1109/ICACC.2014.66.

[2] O. AbdElRahem, A. M. Bahaa-Eldin and A. Taha, Virtualization security: A survey, 2016 11th International Conference on Computer Engineering & Systems (ICCES), Cairo, 2016, pp. 32-40, doi: 10.1109/ICCES.2016.7821971.

[3] N. Goel, A. Gupta and S. N. Singh, A study report on virtualization technique, 2016 International Conference on Computing, Communication and Automation (ICCCA), Noida, 2016, pp. 1250-1255, doi: 10.1109/CCAA.2016.7813908.

[4] N. Jain and S. Choudhary, Overview of virtualization in cloud computing, 2016 Symposium on Colossal Data Analysis and Networking (CDAN), Indore, 2016, pp. 1-4, doi: 10.1109/CDAN.2016.7570950.

[5] Virtualization of Hardware – Introduction and Survey Christian Plessl and Marco Platzner Computer Engineering & Networks Lab Swiss Federal Institute of Technology (ETH) Zurich, Switzerland plessl@tik.ee.ethz.ch

[6] HARDWARE VIRTUALIZATION SUPPORT IN INTEL, AMD AND IBM POWER PROCESSORS

[7] R. Perez, L. van Doorn and R. Sailer, Virtualization and Hardware-Based Security, in IEEE Security & Privacy, vol. 6, no. 5, pp. 24-31, Sept.-Oct. 2008, doi: 10.1109/MSP.2008.135

[8] D. V. Vu, T. Sandmann, S. Baehr, O. Sander and J. Becker, Virtualization Support for FPGA-Based Coprocessors Connected via PCI Express to an Intel Multicore Platform, 2014 IEEE International Parallel & Distributed Processing Symposium Workshops, Phoenix, AZ, USA, 2014, pp. 305-310, doi: 10.1109/IPDPSW.2014.42.

[9] T. Li, V. K. Narayana, E. El-Araby and T. El-Ghazawi, GPU Resource Sharing and Virtualization on High Performance Computing Systems, 2011 International Conference on Parallel Processing, 2011, pp. 733-742, doi: 10.1109/ICPP.2011.88.

[10] C. Huang and P. Hsiung, Hardware Resource Virtualization for Dynamically Partially Reconfigurable Systems, in IEEE Embedded Systems Letters, vol. 1, no. 1, pp. 19-23, May 2009, doi: 10.1109/LES.2009.2028039.

[11] S. Kan and J. Dworak, Systematic Test Generation for Secure Hardware Supported Virtualization, 2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech), 2017, pp. 550-556, doi: 10.1109/DASC-PICom-DataCom-CyberSciTec.2017.100.

[12] S. Jin, J. Ahn, S. Cha and J. Huh, Architectural support for secure virtualization under a vulnerable hypervisor, 2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2011, pp. 272-283.

[13] A. Chaoub, M. Giordani, B. Lall, V. Bhatia, A. Kliks, L. Mendes, K. Rabie, H. Saarnisaari , A. Singhal, N. Zhang, S. Dixit, M. Zorzi, 6G for Bridging the Digital Divide: Wireless Connectivity to Remote Areas, IEEE Wireless Communications Magazine, 2021

[14] Address Translation, https://www.d.umn.edu/~gshute/os/address-translation.xhtml

[15] Paging, https://www.gatevidyalay.com/paging-memory-management-operating-system/

[16] VMware, http://www.vmware.com/

[17] Virtualization, https://www.ibm.com/cloud/learn/virtualization-a-complete-guide

[18] Motivation for Virtualization, https://www.vmware.com/in/solutions/virtualization.html

[19] Virtualization Techniques, https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf

[20] Working of Virtualization, https://www.redhat.com/en/topics/virtualization/what-is-virtualization

[21] History of Virtualization, https://docs.oracle.com/cd/E26996_01/E18549/html/VMUSG1010.html

[22] https://medium.com/devops-mojo/difference-between-docker-container-and-virtual-machine-comparision-between-docker-and-vm-697330d0761c

[23] https://www.mycloudwiki.com/cloud/fundamentals/virtualization-hypervisor-basics/

[24] https://medium.com/devops-mojo/difference-between-docker-container-and-virtual-machine-comparision-between-docker-and-vm

[25] https://www.itfuel.com/blog/9/virtualisation-in-laymans-terms-for-the-smaller-organisation/

[26] https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/msg-signaled-interrupts-paper.pdf