future internet

MDPI

Article

Secure and Dynamic Memory Management Architecture for Virtualization Technologies in IoT Devices

Jithin R * and Priya Chandran

Department of Computer Science and Engineering, National Institute of Technology, Calicut 673601, India; priya@nitc.ac.in

* Correspondence: jithinr550@gmail.com; Tel.: +91-9947545200

Received: 15 October 2018; Accepted: 28 November 2018; Published: 30 November 2018

Abstract: The introduction of the internet in embedded devices led to a new era of technology—the Internet of Things (IoT) era. The IoT technology-enabled device market is growing faster by the day, due to its complete acceptance in diverse areas such as domicile systems, the automobile industry, and beyond. The introduction of internet connectivity in objects that are frequently used in daily life raises the question of security—how secure is the information and the infrastructure handled by these devices when they are connected to the internet? Security enhancements through standard cryptographic techniques are not suitable due to the power and performance constraints of IoT devices. The introduction of virtualization technology into IoT devices is a recent development, meant for fulfilling security and performance needs. However, virtualization augments the vulnerability present in IoT devices, due to the addition of one more software layer—namely, the hypervisor, which enables the sharing of resources among different users. This article proposes the adaptation of ASMI (Architectural Support for Memory Isolation—a general architecture available in the literature for the improvement of the performance and security of virtualization technology) on the popular MIPS (Microprocessor without Interlocked Pipeline Stages) embedded virtualization platform, which could be adopted in embedded virtualization architectures for IoT devices. The article illustrates the performance enhancement achieved by the proposed architecture with the existing architectures.

Keywords: IoT security; dynamic memory allocation; memory isolation

1. Introduction

The internet can be defined as a vast network connecting computing devices that are distributed throughout different areas of the universe. Real-world objects such as cars, air conditioners, refrigerators, and others that connected to the internet with the help of internet-enabled embedded devices are called the Internet of Things (IoT). The internet communication infrastructure has evolved and grown immensely since its inception in the early 1990s, and we perceive its growth as a process with four marked stages as illustrated in Table 1.

Table 1. Stages of development of the internet communication infrastructure.

Stage	Devices	Connectivity
Mammoth Era	Desktops, Servers	Ethernet
Mobile Computers Era	Laptops, Notebooks	Wifi, Bluetooth
Smartphone Era	Mobiles, Smartphones	GPRS, 3G, 4G
Internet of Things Era	TVs, Fridges, Watches	Bluetooth, Zigbee, Wifi, 3G, 4G

The fast growth of the internet is popularly understood to have been caused by rapid advancements in two fields—networking technology and embedded device technology. In the first three stages of the development of networking, the diversity of devices connected to the internet was limited to servers, desktops, laptops, and mobile phones. Hence, the volume of data to be processed on the internet was limited and could be structured using standard databases.

The fourth growth stage of the internet led to the pervading of the web into day-to-day life objects, manifesting as the new era of technology called the Internet of Things (IoT) Era. Due to the evolution of the IoT, the diversity of objects on the internet grew from home appliances to automobiles, and consequently increased the volume of data to be processed on the internet. Moreover, the diverse nature of this information made it unsuitable for a structured database. With that, the vulnerability of the information and infrastructure multiplied manifold.

We start with our view of the IoT as a layered information flow infrastructure. According to our perception, the general structure of the IoT and the flow of information can be depicted as a six-layer architecture as shown in Figure 1. The first layer, *Information*, describes the real-time information about an object. For example, temperature, speed, humidity, sound, and similar parameters could be the types of information. Information can be real information or virtual information stored in devices like Radio-Frequency Identifications (RFIDs) and bar-codes. The information about the actual objects is fetched in the second layer of the IoT architecture, *Fetch*. Devices like sensors, RFID readers, and bar-code readers are used to carry the information, and comprise the Fetch stage. In a single IoT device, different information fetching devices can be available.

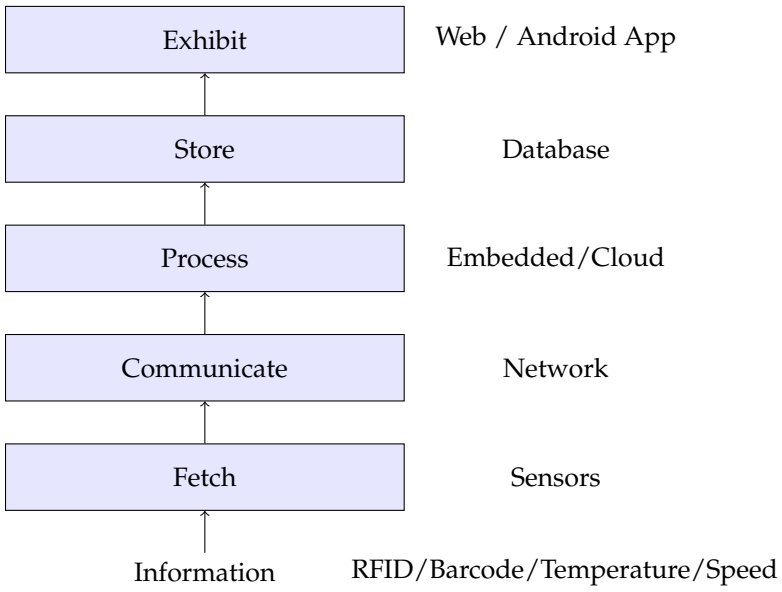

Figure 1. Information flow in the Internet of Things (IoT) architecture.

In the next layer, *Communicate*, this information is transferred to the processing unit. The information can be transferred between the fetching device and the processing device through wired or wireless medium. The communicated data are processed in the next level of architecture,

Process. Processing can be done either in an embedded device or a method available on the internet (i.e., Cloud). The fifth layer, *Store*, keeps all the data in a structured manner. The storage can be a database server or a cloud server. The sixth layer, *Exhibit*, displays this stored information to the end-user based on the requirements of the user through relevant applications. Applications can be web applications, Android applications, or can be based on real-time operating systems (RTOSs) such as RTLinux or something equivalent.

Unlike the devices available in the previous era, the impact of a security attack on IoT devices is enormous, as they directly connect to real-life objects like TVs, air conditioners, cars, etc. Recently, a specific style of terrorist attack has become prominent, where the vehicles are driven towards the crowd. We saw this style of attack in Paris and London. Terrorists can exploit IoT devices to carry out similar attacks. We also saw Mirai [1], a malware that was first identified in August 2016, which enters into non-secure IoT devices and implements a Distributed Denial of Service (DDoS) attack on the servers. Hence, basic security principles have to be ensured at each layer of the IoT architecture to protect the IoT devices. Standard cryptographic security solutions to ensure various security principles are not suitable due to the power and performance constraints in IoT devices [2]. The prime objective of this article is to present an architectural solution that improves security without compromising the performance of IoT systems.

The remaining sections of this article are organized as follows: Section 2 explores the various security concepts in IoT. Section 3 deals with virtualization technologies on the embedded platform and the security requirements of virtualization technologies in embedded platforms. Section 4 describes the MIPS embedded architecture, the virtualization technology in MIPS (Microprocessor without Interlocked Pipeline Stages) and the essential requirements for performance and security enhancements in MIPS virtualization. In Section 5, a secure and efficient ASMI (architectural support for memory isolation) memory architecture for the x86 architecture is adapted for the MIPS architecture and proposed as a memory isolation architecture for the MIPS architecture. Changes in hypervisor operation due to the incorporation of ASMI are explained in Section 6. Section 7 presents a comparison of the proposed architecture with the MIPS architecture and the MIPS virtualization architecture, with respect to the memory access operation using ModelSim and Xilinx software suites. Section 8 concludes the article with future directions for the work.

2. Security of IoT

IoT devices are widely used in the automobile industry for driver-assistance systems [3] and to achieve the industry goal of driverless cars. The information from IoT devices is used to control the acceleration, braking, and other functionalities of the car [3,4]. Attackers can exploit vulnerabilities in these types of IoT devices on the internet to carry out undesirable events on cars, such as increasing the speed of the car and braking or swerving suddenly, which may create dangerous situations on the road.

As pointed out in Section 1, IoT devices can widely differ in networking technology, and thereby differ significantly in terms of the nature of their security issues. Vulnerabilities in hardware design are also a significant challenge in consumer and industrial IoT devices [5]. Hence, the security of the IoT is an open research area where the security issues are heterogeneous. The richness of sensors and communications in embedded systems leads to threats that are far more prevalent [6] than those in normal systems. This observation motivates researchers in the area of security to work towards the security of IoT devices and systems.

The security of any system requires satisfying three necessary security conditions: confidentiality, integrity, and availability (CIA). These critical principles of security have to be accomplished for the information infrastructure at each layer of the IoT architecture to ensure the safety of the IoT system [4,7,8].

In a typical computing environment, confidentiality, integrity, and availability are ensured through the implementation of cryptographic methods like encryption, decryption, and hashing. However,

in an embedded environment, these conventional solutions are not suitable due to the power and performance constraints of embedded devices [2,9]. A detailed survey on the topic of Internet of things security [10] compared conventional system security and IoT system security and concluded that the traditional security mechanisms are not applicable in IoT systems, as the approaches are different in both systems. The survey included various types of IoT security solutions which are suitable for different kinds of IoT systems. The review also suggests finding a global security architecture for IoT systems instead of individual solutions.

Redesigning the system functionalities considering the security and performance requirements may be a viable solution to the security issues in embedded platforms [2]. The following paragraphs review the security solutions available in the literature, which are claimed to be better for embedded systems regarding performance compared to conventional security solutions available on conventional systems.

- Artificial intelligence (AI) is used in IoT devices in complex IoT environments for achieving better security [11]. AI-based methods use fewer resources than conventional methods. Neural networks, fuzzy logic and systems, and semantic computing are the methods used to meet the security requirements. Abnormal behavior analysis is an AI-based method to analyze the security of an entity. An AI profile is created for each entity, based on its normal behavior. Based on these AI profiles, entities are classified as secure and non-secure. This AI technique is used in smart home networks [2] for the protection of the sensor layer in the IoT architecture. Anomaly behavior analysis is executed on each sensor node and continuously evaluated to protect the operations of each node against threats.
- Miettinen et al. [12] describe an enhanced AI method for smart homes that classifies devices irrespective of the device type into three isolation levels—strict, restricted, and trusted. Each device is associated with a fingerprint. Fingerprints are generated from the packets generated by the devices [12]. A single classifier is trained for each device type. Each classifier provides a binary decision whether the input fingerprint matches the device-type or not.
- A lightweight hashing-based block-chain method described in [13] ensures confidentiality, integrity, and availability protection for IoT devices in smart homes. The advantage of this method was that it incurred less computing overhead for improved security, compared to the non-secure base model [13].
- A method based on a physical unclonable function (PUF), described in [14], is characterized by ultra-fast performance, ultra-low energy consumption, and small silicon footprint, and is hence suitable for low-energy devices. PUF ICs are embedded into each IoT device in the network to create a secure network protocol [14]. The network protocol relies on PUFs' challenge and response instead of cryptographic functions.
- Virtualization technology is considered as a security solution by itself in embedded devices [15,16] due to its ability to provide unique execution environments for each application. Another main benefit of virtualization technology is that it can provide a heterogeneous application environment on a single embedded platform with temporal and spatial separation. In other words, it can provide an isolated application environment for real-time applications, and another one for general-purpose applications, on the same embedded platform at the same time. Thus, virtualization technology helps users to load their applications on embedded devices without affecting the security or performance of machine-critical applications loaded in separate virtual environments [17].

An isolated heterogeneous application environment is a prime requirement in embedded devices like smartphones and smart TVs, where the original equipment manufacturer (OEM) applications and user applications can run simultaneously without affecting the security and performance of the device. This article focuses on virtualization technology security-based solutions, as they can provide an

isolated heterogeneous application environment for real-time applications. The next section discusses virtualization technologies in embedded devices and their challenges.

3. Virtualization on Embedded Devices

Embedded devices were initially used to run a single real-time application. The improvements in embedded hardware performance resulted in the ability to run multiple real-time applications on the same embedded platform. Sharing resources among multiple real-time applications causes security and performance issues. Providing separate execution environments (isolation) for real-time applications is a security and performance requirement in embedded platforms [17]. Virtualization technology can be used to fulfill these requirements. Virtual machine technology provides isolation at the middle layer (i.e., the process layer) of IoT information flow architecture.

Virtualization technology is mainly used in embedded devices to run normal and real-time operating systems (RTOSs) concurrently [17–20]. The difference between real-time operating systems and normal operating systems is that in a real-time operating system, the response time of instruction execution demands a guaranteed throughput rather than a high throughput [21] as required in a normal operating system. That means the response time or the time taken to complete the task can be predicted with 100% accuracy at the time of system design. Hence, the response time of embedded systems is also known as predicted time. That means the latency (delay over the predicted time) of execution of instructions should be less than or equal to zero for real-time operating systems. A latency greater than zero indicates failure of the real-time system [21].

The predicted time for instruction execution in the virtual real-time operating system should be the same as in a normal real-time operating system. Regular hypervisors are designed to create virtual machines running general operating systems. Embedded hypervisors [22] are required to build virtual machines that run real-time operating systems.

The embedded version of the renowned open source hypervisor Xen, called RT-Xen [22], is a real-time hypervisor for embedded platforms. ARINC 653 is another embedded hypervisor designed to provide spatial and temporal isolation for applications in the aviation industry. Several commercial RTOS products have adapted the ARNIC 653 design standards.

A virtual RTOS running in the QPlus Hypervisor is reported to have performance degradation due to the delay in interrupt execution. The interrupt execution delay is due to the additional time consumed for virtual CPU (VCPU) scheduling. If the VCPU is assigned to a particular CPU, the predicted time of instruction execution in virtual RTOS and normal RTOS would be similar, as CPU virtualization is a time-based demultiplexing method [19,23].

Hence, in real-time hypervisors, in order to achieve predicted time instruction execution on each virtual RTOS, the virtual CPU of each virtual RTOS should be directly assigned/pinned to the physical CPU. The SIGMA system is an example of such a system, designed such that a dedicated CPU core is assigned to each VCPU [19], to build a multi-OS environment with native performance. In this system, the interrupts to the CPU are rerouted to the corresponding virtual CPU with real-time performance [22].

In an embedded hypervisor, memory virtualization does not degrade performance, as memory is virtualized through spatial demultiplexing [19]. Additionally, the virtualization of Input/Ouput (I/O) device is done through memory [23]. Thus, the primary functionality of virtualization for an embedded hypervisor should be provided in the memory architecture. The embedded hypervisor architecture can be visualized as shown in Figure 2.

A trusted computing base/secure execution environment is considered to be a major security requirement of a virtualized IoT infrastructure [20,24–26]. Virtual machine isolation at different hardware levels (e.g., processor, memory, and networking) is essential for achieving a trusted computing base [27].

The ARM architecture uses ARM-TrustZone, a solution enabling two different zones to run on a single embedded platform [28]. One zone is considered as a secure zone, dedicated for running OEM

applications, and the other zone is a non-secure zone, intended for running user applications. The main drawback of ARM-TrustZone technology is that it is restricted to a single secure zone. That means only one secure zone is available at a time.

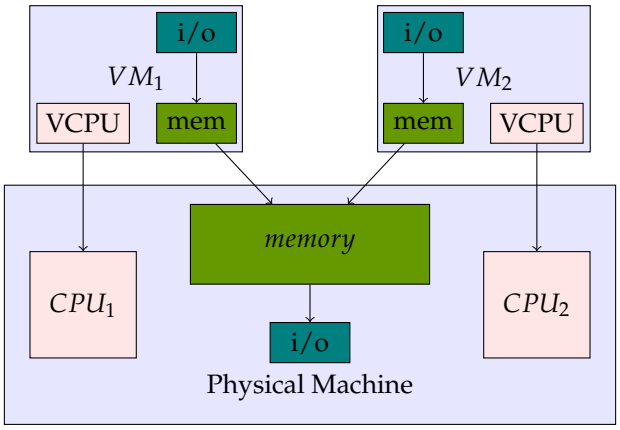

Figure 2. Architecture of embedded hypervisor.

According to the architecture developers in MIPS Technologies, Inc., the ARM-TrustZone model security approach is complex to implement [29]. MIPS architecture developers used a more scalable solution named MIPS multi-domain security [29]. The MIPS multi-domain technology provides multiple secure zones to each system function through the separation of memory spaces. MIPS multi-domain security is achieved through hardware-assisted full virtualization. Section 4 explains the full virtualization in MIPS.

In an embedded hypervisor architecture as shown in Figure 2, the VCPU is directly assigned to a specific CPU and a secure execution environment is achieved by securely partitioning the memory [17,26].

Embedded systems are usually based on microcontrollers (i.e., memory-integrated CPU), and hence are smaller in memory size. Running multiple applications on a single platform requires more memory than a single application running on the same platform. Therefore, the dynamic allocation of memory in an embedded hypervisor would be an added advantage towards the performance of virtual machines.

It is understood from the above observations that memory isolation and dynamic memory allocation are essential requirements of an embedded hypervisor for security and performance enhancements. Architecture support for memory isolation (ASMI) is a proposal for improving the security and performance of virtual machines in normal hypervisors [30]. The main features of the ASMI architecture are

- Dynamic memory allocation to virtual machines: Allocating memory to virtual machines on demand.
- Memory isolation among virtual machines (VMs): Separating the memory of each VM from others and also from the hypervisor for security and performance enhancements in VMs. This indicates that a memory region should be assigned to a single VM at a time.

As the features mentioned above of ASMI are the essential requirements of embedded hypervisor for security and performance enhancements in virtual machines, we propose ASMI on an embedded platform for performance and security enhancements. We decided to analyze the feasibility of ASMI on MIPS embedded architecture as the memory isolation feature of MIPS Multi-domain Security architecture matches with the feature of ASMI. The next section explains the MIPS architecture.

4. MIPS Architecture

The Microprocessor without Interlocked Pipeline Stages (MIPS) architecture has had a major share in the embedded processor market since 1990. The architecture has evolved and assimilated new technologies, including virtualization technologies. The MIPS architecture includes a virtualization module, which allows the hypervisor to run the fully virtualized guest OS. The provision of a feature to run a guest OS without modification is known as full virtualization. First, we describe the virtualization module of the MIPS architecture that forms the background on which ASMI is built for memory isolation.

To execute guest operating systems in full virtualization mode, additional registers, instructions, operating modes, and interrupts are required. The additionally required components in the MIPS architecture to enable full virtualization to guest operating systems are collectively called the virtualization module (MIPS VZ) [31]. The virtualization module helps to create separate operating environments for the guest OS and the hypervisor.

Operating Modes:

The virtualization module introduces a new operating mode named the *guest* mode in addition to the kernel, privileged, and user modes in the MIPS architecture. In a fully virtualized system, normal modes are accessible to the hypervisor, and the guest mode is accessible to the guest operating system.

Address Translation:

With virtualization arises the need for translation of the virtual guest address to an actual physical address. This is met with the help of an additional layer, as shown in Figure 3. The first address translation layer is used by the hypervisor and the second by the guest OS. The guest OS translates from a virtual address to a pseudo-physical address in the second layer and the hypervisor layer translates from a pseudo-physical address to a physical address in the first layer.

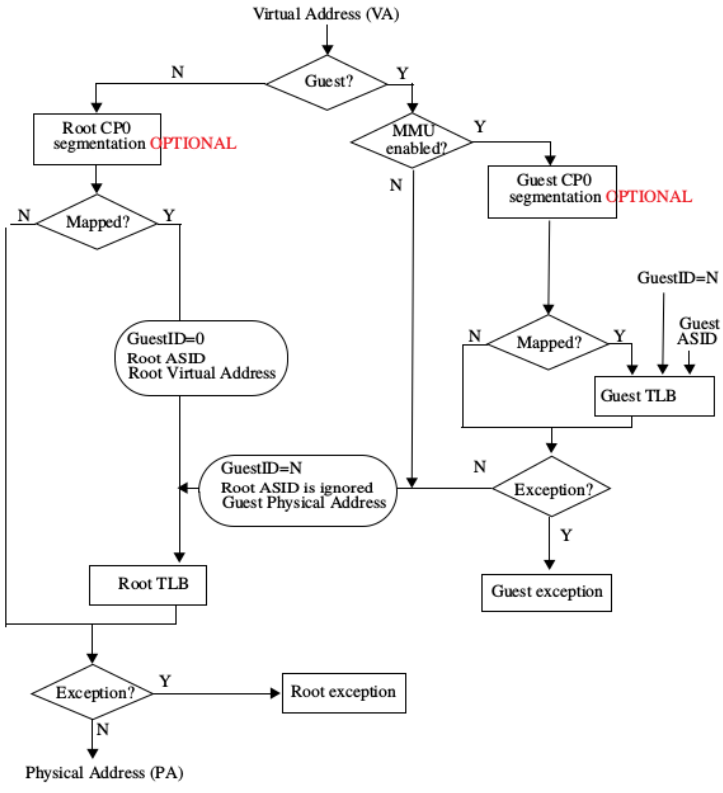

Figure 3. Two-level address translation in the MIPS virtualization mode [31].

CPU Registers:

The MIPS Privileged Resource Architecture (PRA) [31] controls the system resources. The CP0 registers [31] provide an interface between the instruction set architecture and the PRA. Different combinations of CP0 register values define different configurations of the MIPS architecture. The virtualization module adds new registers to CP0 to interact with PRA.

New Instructions:

Switching the mode of execution between root mode and guest mode is done using two new instructions: HYPCALL and ERET [31]. The HYPCALL instruction moves the control of execution from guest mode to root mode; and the ERET instruction, executed by the hypervisor, moves the control back to guest mode. HYPCALL and ERET instructions are analogous to the execution of SYSCALL and RET instructions in operating systems, to switch between user and kernel modes. These new instructions are part of the virtualization module which are configured through CP0 registers. Other than these instructions, some additional instructions (i.e., *DMFGC0, DMTGC0, MFGC0, and MTGC0*) are also introduced to move the data among guest OS and hypervisor [31].

When the CPU translates a virtual address, it checks whether the execution is in guest mode or not using the control bits in the *GuestCtl0* register, which is a part of the CP0 register set. The *Guest.Cause* register in the CP0 register set is used for guest interrupts. The guest OS can write to the *Guest.Cause* register to trigger interrupts. The external interrupts are rooted to the guest OS by the hypervisor. The $GuestCtl0_{PIP}$ field in the CP0 registers facilitates handling external interrupts for the guest OS and hypervisor.

If the CPU is in non-guest mode (root mode), the virtual address is searched for in the root TLB (Translation Lookaside Buffer). If the virtual address is mapped in the root TLB, the physical address is given back to the CPU. If the virtual address is not allocated in the TLB, an exception is raised and handled by the hypervisor. When the CPU is in guest mode, the virtual address is searched in the guest TLB. If the virtual address is found in guest TLB, the corresponding pseudo-physical address is returned, and if not found, an exception is raised. This pseudo-physical address is then converted, in root mode, to a real physical address using the root TLB, in the same pipeline stage [31].

This two-layer address translation mechanism is a bottleneck for achieving near-native performance in virtual RTOSs. Moreover, pre-assigned memory for virtual machines in an embedded hypervisor would cause memory starvation, since most of the devices are microcontrollers with small memory sizes.

ASMI is a hardware-assisted memory isolation technology in the literature that can provide memory isolation and dynamic memory allocation, and has a single-level address translation method. These features are essential requirements for security and performance enhancements in an embedded hypervisor. Hence, we propose ASMI on the MIPS embedded platform. The next section explains the proposal in detail.

5. ASMI-Enhanced MIPS

We propose ASMI on the MIPS embedded platform and present the details of the proposed architectural enhancements in the MIPS architecture and the MIPS virtualization module. The ASMI enhancements in MIPS are organized as follows.

Pro-mem:

ASMI introduces a new hardware unit (microcontroller) named the Pro-mem unit. The Pro-mem unit lies between the memory management unit and physical memory. The Pro-mem unit controls the entire physical memory operations. Pro-mem segregates the whole physical memory into many fixed-length physical segments. Each segment contains a fixed number of pages.

MPT:

A data-structure named the memory protection table (MPT) is introduced in the primary memory, which is accessible only to the Pro-mem unit. In a computer system, while booting, memory map information is passed to the hypervisor kernel from the BIOS. This memory map information explains architectural specifications like total available memory and restricted area. The memory map information in the BIOS has to be rewritten to subsume the MPT.

The remaining physical memory is divided into several fixed-length segments. It is proposed that the segment size is fixed after a detailed analysis of different segment sizes. Each segment has a corresponding entry in the MPT. Each entry consists of the segment number and the virtual machine ID. Virtual Machine ID, 0 (zero) is used for the hypervisor and privileged domain. Pro-mem assigns each segment to a single VM by storing the segment ID (SegID) and its corresponding virtual Machine ID (VMID) in the memory protection table (MPT).

Pipeline:

In the MIPS architecture, memory is handled using memory load and memory store operations. These load and store operations have to be executed through the Pro-mem unit. Hence, the required modifications in MIPS pipeline architecture shown in Figure 4, can be described as follows. Two changes are proposed in the MIPS pipeline. One is in the Instruction Fetch (IF) stage and another is in the Memory (MEM) stage, as shown in Figure 4.

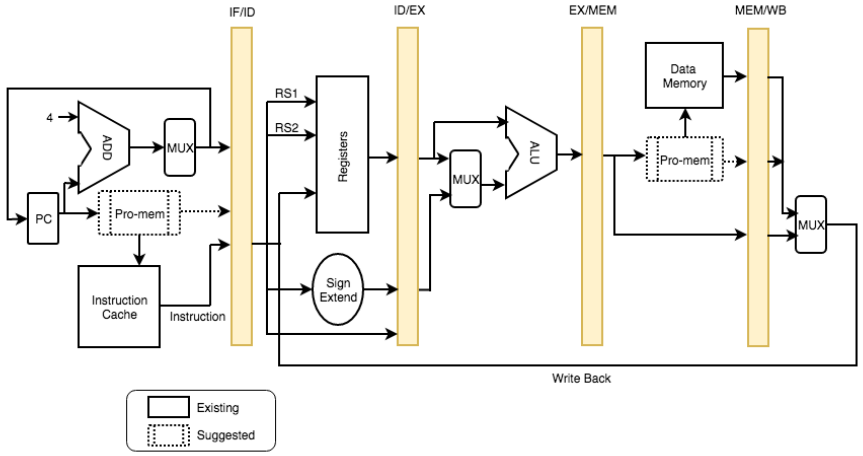

Figure 4. Proposed MIPS pipeline changes for ASMI.

In the first MIPS pipeline stage, an instruction is fetched from the instruction cache. ASMI proposes the verification of the instruction address from the MPT through the Pro-mem unit before fetching the instruction. Similarly, the read/write instruction reads/stores the data in memory at the fourth stage MEM. Before reading/writing the data to/from memory, ASMI proposes the address to be verified from the MPT through the Pro-mem unit. With these changes, all memory operations are ensured to be done through the Pro-mem unit.

These additional changes would make a delay in the IF and MEM stages of pipeline execution. Thus, to make the time difference in the pipeline stages approximately zero, we propose that the address verification be executed parallel to the cache controller, as the execution of data fetch and address verification functions are on separate hardware components. This is a challenge to be addressed in the ASMI development stage.

Cache Memory:

No change in the cache memory operations is required in light of the fact that as per the ASMI design, the information fetch operation in the cache memory at the pipeline stage is executed in parallel to the address verification task in the Pro-mem unit. However, a change is required in the cache controller to convey the fetched information from the cache memory to the processor in light of the address verification result. The alteration is to be addressed while implementing ASMI, based on the given number of cache levels in the processor architecture.

Instructions:

HYPCALL and ERET are the instructions that move the control of execution back and forth between the VM and the hypervisor. In the ASMI architecture, these instructions are modified to communicate with the Pro-mem unit about the movement of the control of execution among the VM and hypervisor. In addition to these instructions, two new instructions are required to inform the Pro-mem unit about the creation and deletion of the VM, named as the VMLAUNCH and VMHALT instructions.

TLB:

The MIPS VZ has a root TLB and a guest TLB. ASMI proposes a new field, Guest ID (GID), in the guest TLB to distinguish among the guest entries. Guest TLBs and the page tables in the guest OS store the virtual address and the corresponding real physical address. Table 2 shows the Guest TLB entry fields in the proposed ASMI-enabled MIPS architecture. In a Guest TLB entry, VPAGE indicates the virtual address bits, ASID indicates the process ID bits, GID indicates the Guest ID bits, PPAGE indicates the physical address bits, STAT indicates the status bits, and UND indicates the undefined bits. The size of each field is also mentioned in the table.

Table 2. Guest TLB entry in the proposed architecture.

VPAGE	ASID	GID	PPAGE	STAT	UND
20 bit	*6 bit*	*6 bit*	*20 bit*	*4 bit*	*8 bit*

Registers:

ASMI introduces three registers in the CP0 named SegMax, TOT, and SegLimit. These registers are updated with the help of VMLAUNCH and VMHALT instructions. The hypervisor calculates the total number of segments (SegLimit) at the time of hypervisor booting. SegLimit is constant until the hypervisor is rebooted. The SegLimit value is calculated at the hypervisor boot time, based on the segment size that the hypervisor is designed to work.

The register TOT is initially zero. When a hypervisor is loaded, TOT is updated to 1. TOT is incremented and decremented by 1 when a VM is created or deleted, respectively, with the help of VMLAUNCH and VMHALT instructions. Whenever TOT is updated, SegMax is updated. The value in SegMax is computed by dividing SegLimit with the TOT value.

ASMI has a VMIDR register in its design to store the currently running VMID. However, it is not required in MIPS as the CP0 already has a register to store the currently running Guest ID. With these additional components, we can incorporate the ASMI architecture in MIPS as shown in Figure 5.

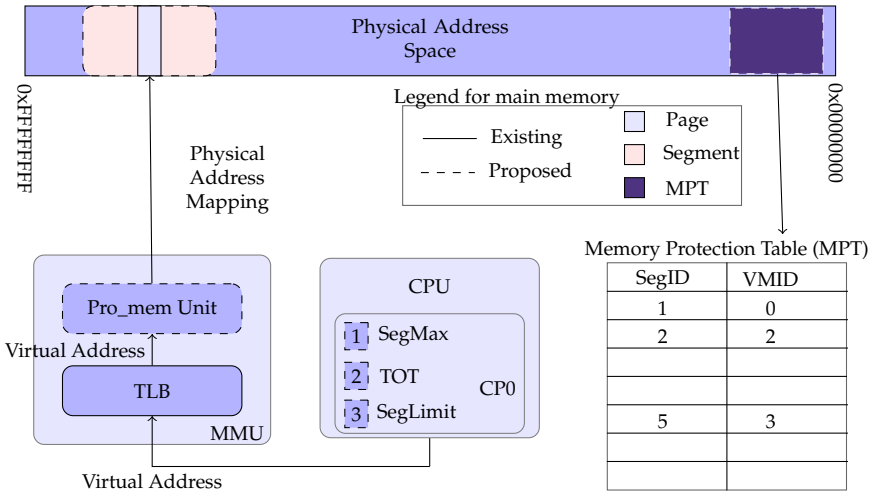

Figure 5. Proposed ASMI architecture for MIPS.

6. Operational Changes in the Proposed Architecture

The proposed modifications for employing ASMI in the MIPS architecture call for some changes in the embedded hypervisor operations. These modifications are in the initial booting, context switch, memory allocation, and memory access operations.

Booting:

The memory-map is a layout of the primary memory, passed on from the BIOS to inform the OS/hypervisor kernel about information such as the size of total memory, reserved regions, and other architectural details. When the hypervisor is booting, the memory map of the ASMI is transferred to the hypervisor from the BIOS, which restricts a portion of memory to the MPT table.

Initially, the MPT is empty. The MPT table has two fields. One is segmented ID, and another is virtual machine ID. The hypervisor memory request is passed to the Pro-mem unit, and the Pro-mem unit allocates the segments to the hypervisor by updating the MPT table with segment ID and the corresponding virtual machine ID. The virtual machine ID for the hypervisor and the privileged domain is zero. The hypervisor allocates these segments to its process by updating the root TLB with the physical address and corresponding virtual address. The initial register settings of the newly proposed registers are explained next.

The Pro-mem unit allocates a virtual machine ID zero to the guest's ID field in the CP0 register set while the hypervisor is booting. The SegLimit register is updated at boot time with a value equal to the total physical memory size divided by the segment size. SegMax is assigned with the SegLimit value, and the TOT value is set to 1 while booting the hypervisor.

The initial configurations of the architecture while booting are mentioned above. When a virtual machine is created or deleted by executing VMLAUNCH and VMHALT instructions, settings get updated as described.

Context Switch:

A hardware signal is sent to the Pro-mem unit to inform the execution of the VMLAUNCH instruction. The Pro-mem increments the TOT register by one. The SegMax register is updated by a value equal to SegLimit divided by TOT. Pro-mem assigns a random value to the CP0 Guest.ID register.

When a virtual machine is deleted by executing the VMHALT instruction from the hypervisor, settings get updated as described. A hardware signal is sent to the Pro-mem unit to inform the execution of the VMHALT instruction. Pro-mem decrements the TOT register by one. SegMax is updated by a value equal to SegLimit divided by TOT. Pro-mem resets all memory segment entries in the MPT of the corresponding VM with zero and fills those memory segments with zero.

Context switch between virtual machines and the hypervisor is done using HYPCALL and ERET instructions. Pro-mem stores the CP0 *Guest.ID* value to the initial address of the first memory segment of the VM and switches the CPU to root mode when a HYPCALL instruction is executed from the guest OS. Pro-mem loads the CP0 *ID* from the initial address of the first memory segment of the hypervisor and continues the execution in root mode.

When the ERET instruction is executed by the hypervisor, the CP0 *ID* is stored in the initial address of the first memory segment of the hypervisor and the CPU is switched to the guest mode. Pro-mem loads the CP0 *Guest.ID* from the initial address of the first memory segment of the VM and execution is continued in guest mode. Memory allocation and access operations in the ASMI architecture are explained next.

Memory Allocation:

Segments are allotted to virtual machines by storing the segment ID and virtual machine ID in the MPT table, and the assigned segment information is passed to the hypervisor. The hypervisor passes this information to the corresponding domain/virtual machine through *start_info_pages* and *shared_info_pages*. This information is used by the guest operating system to update the *free_pages* list. The memory allocation function in the guest OS assigns these *free_pages* to virtual address space and updates the page tables and TLB.

Memory Access:

Executing instructions in the CPU and fetching data require access to primary memory, which needs address translation. Address translation in ASMI is executed as shown in Figure 6, and can be described as follows. The processor/CPU passes the virtual address to the guest/root TLB. If the corresponding entry is identified from TLB, the physical address is given to the Pro-mem unit for validation.

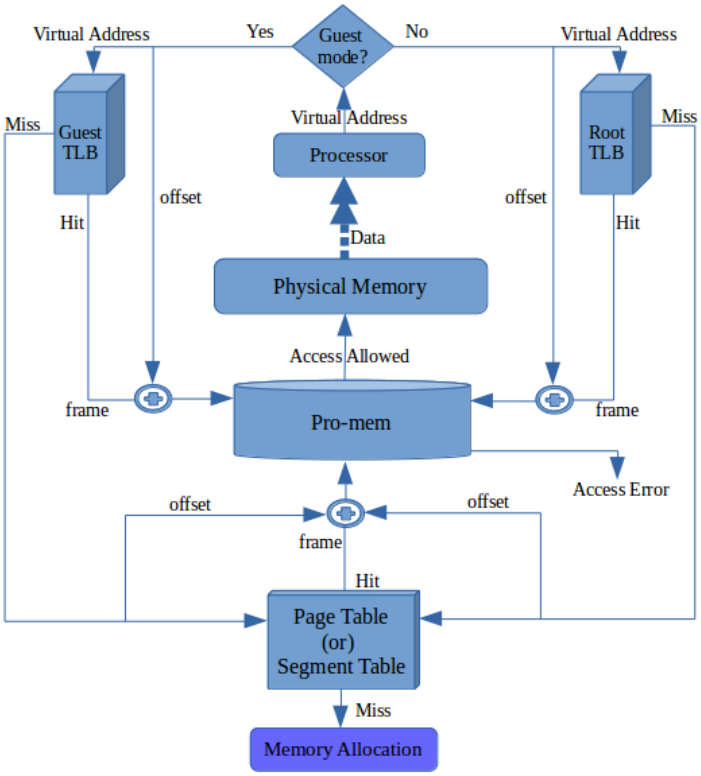

Figure 6. Address translation in ASMI.

If a TLB miss occurs, page tables are searched by the guest OS to determine the corresponding physical address. A page hit passes the physical address to the Pro-mem unit for validation. A page miss raises an exception that requests the OS to load the pages into memory through a memory allocation function.

Currently available simulators, which simulate the processors, memory, disk, and the entire system are written in high-level languages and run in the lowest privilege level. Since the instructions in the hardware-assisted virtualization environments operate at different privilege levels, such simulators are not suitable for the simulation and analysis of ASMI memory architecture [32]. Due to the lack of a proper simulation environment for virtualization-enabled processors, we decided to estimate the time taken for a memory access operation in an ideal memory module, simulated and synthesized using *ModelSim* and *Xilinx ISE* software. The following section explains the analysis process.

7. Performance Analysis

ModelSim is a multi-language HDL (Hardware Description Language) simulation environment for the simulation of electronic circuits. *Verilog*, a hardware description language available in *ModelSim* to model electronic systems, was used here for the simulation and verification of the correctness of the ASMI memory access operation. The following three Verilog modules constitute the memory access operations of ASMI in *ModelSim*.

- *calculate_seg_ID:* This module receives the physical address as input and calculates the segment ID.
- *data_memory:* This module fetches the data from memory
- *compare:* This module compares the given VMID value with the one in the VMIDR register.

Memory was organized as $(2^m) \times (n)$, where m is the address bus size and n is the data bus size. We used the m, n, and $segment\_ID$ sizes 8,8, and 4, respectively.

Xilinx, an integrated synthesis environment (ISE) used for the synthesis of electronic circuits, was used to obtain the delay of different electronic modules simulated using *ModelSim*. This delay is the time difference between the output obtained and the input provided for the respective modules. From the synthesis, the time taken for the operation of each module of ASMI was obtained as follows.

- *calculate_seg_ID*: 7.045 ns;
- *data_memory*: 5.835 ns;
- *compare*: 8.627 ns.

In normal memory access, only the *data_memory* module operation exists. For ASMI, however, the other two additional module operations (calculate_seg_ID & compare) are added to the *data_memory* operation. We analyzed the memory access operation in detail from the perspective of an operating system in three different environments. One was in a normal machine, another was in a virtual machine, and finally one in an ASMI-enabled virtual machine.

A memory access operation on any operating system consists of an address translation function and a data fetch operation from memory. In a regular machine, address translation is a paging mechanism, where each address translation constitutes four memory access operations [33]. In a virtual environment, the address translation mechanism is a nested page table mechanism, where each address translation constitutes 24 memory access operations [33].

The ASMI memory access operation replaces the nested page table mechanism with a standard page table mechanism with an additional verification function. The ASMI address translation function comprises four memory access operations. Additionally, the three modules mentioned above were used for verification. Hence, based on the simulation results, the time taken for memory access in normal, nested, and ASMI paging were as follows.

- Normal Paging = 4 × 5.835 ns = **23.34 ns**

- Nested Paging = 24 × 5.835 ns = **140.04 ns**
- ASMI Paging = 4 × 5.835 + (7.045 + 8.627 + 5.835) ns = **44.847 ns**

A graphical representation of the time taken by the three address translation mechanisms is shown in Figure 7. The ASMI memory architecture required less time for memory access compared to the nested paging memory access. This shows that ASMI architecture could provide better security and performance to virtual machines than the currently available architectures.

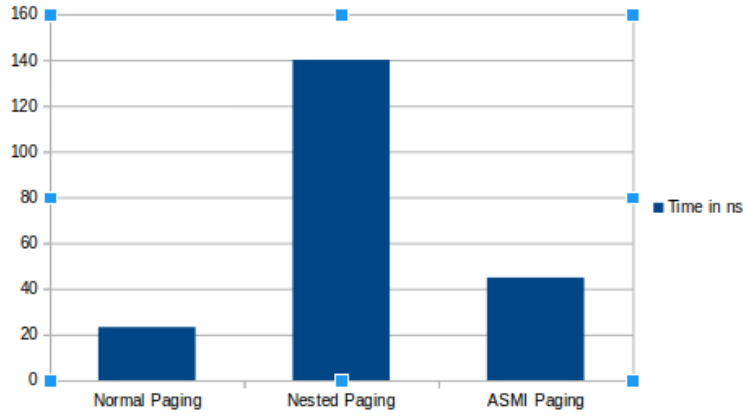

Figure 7. Time taken by a single memory access in different address translation mechanisms.

The results of this preliminary analysis motivates further study on the applicability of the ASMI architecture in embedded processors. Based on the positivity of the obtained results, it is proposed to conduct a full-fledged simulation study of the performance of real programs running on the ASMI-enhanced MIPS virtualization architecture.

8. Conclusions and Future Work

Internet-enabled embedded devices connect real-life objects called IoT devices to the Internet. The security of IoT devices is very important, as IoT vulnerability can profoundly affect human life. Available security solutions in the literature related to embedded platforms were surveyed, and it was inferred that typical security solutions are not suitable for embedded platforms. Virtualization technologies in embedded platforms using the MIPS architecture were surveyed, as virtualization technology is considered as a security solution to some of the security issues in IoT devices. Although the virtualization technology solves a portion of the presently existing security issues in IoT gadgets, it adds some new issues to the IoT systems. We identified the requirements for the enhancements of virtual machine security and performance in embedded hypervisors.

The feasibility of a security solution named ASMI on a MIPS-embedded platform was analyzed. The ASMI architecture was found suitable for embedded devices, as the requirements of virtualization in embedded devices match the features of the ASMI architecture. The required modifications to the MIPS architecture to successfully implement the ASMI architecture were also identified. The detailed operation of the ASMI architecture on the MIPS embedded platform was described. Simulation and synthesis of memory access operations in the ASMI architecture using *ModelSim* and *Xilinx ISE* software revealed that the ASMI architecture has the potential to improve the performance and security of virtual machines.

In the future, implementing ASMI on a proper simulation platform and analyzing the security and performance of the virtual machines running on the ASMI-enhanced hypervisor would clearly establish the practicality of the architecture.

Author Contributions: J.R wrote the paper and the works related to the article. P.C. has given the complete guidance for the work, and she corrected the logical and grammatical mistakes in the paper.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kolias, C.; Kambourakis, G.; Stavrou, A.; Voas, J. DDoS in the IoT: Mirai and other botnets. *Computer* **2017**, *50*, 80–84. [CrossRef]
2. Pacheco, J.; Hariri, S. IoT security framework for smart cyber infrastructures. In Proceedings of the 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W), Augsburg, Germany, 12–16 September 2016; pp. 242–247.
3. He, W.; Yan, G.; Da Xu, L. Developing vehicular data cloud services in the IoT environment. *IEEE Trans. Ind. Inform.* **2014**, *10*, 1587–1595. [CrossRef]
4. Lin, J.; Yu, W.; Zhang, N.; Yang, X.; Zhang, H.; Zhao, W. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet Things J.* **2017**, *4*, 1125–1142. [CrossRef]
5. Wurm, J.; Hoang, K.; Arias, O.; Sadeghi, A.R.; Jin, Y. Security analysis on consumer and industrial iot devices. In Proceedings of the 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), Macau, China, 25–28 January 2016; pp. 519–524.
6. Sherman, M. Attack Surfaces for Mobile Devices. In Proceedings of the 2Nd International Workshop on Software Development Lifecycle for Mobile, Hong Kong, China, 17 November 2014; ACM: New York, NY, USA, 2014; pp. 5–8. [CrossRef]
7. Minoli, D.; Sohraby, K.; Kouns, J. IoT security (IoTSec) considerations, requirements, and architectures. In Proceedings of the 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 8–11 January 2017; pp. 1006–1007.
8. Yang, Y.; Wu, L.; Yin, G.; Li, L.; Zhao, H. A survey on security and privacy issues in internet-of-things. *IEEE Internet Things J.* **2017**, *4*, 1250–1258. [CrossRef]
9. Moratelli, C.R. *A Lightweight Virtualization Layer with Hardware-Assistance for Embedded Systems*; Pontifícia Universidade Católica do Rio Grande do Sul: Porto Alegre, Brazil, 2016.
10. Alaba, F.A.; Othman, M.; Hashem, I.A.T.; Alotaibi, F. Internet of Things security: A survey. *J. Netw. Comput. Appl.* **2017**, *88*, 10–28. [CrossRef]
11. He, H.; Maple, C.; Watson, T.; Tiwari, A.; Mehnen, J.; Jin, Y.; Gabrys, B. The security challenges in the IoT enabled cyber-physical systems and opportunities for evolutionary computing & other computational intelligence. In Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, Canada, 24–29 July 2016; pp. 1015–1021.
12. Miettinen, M.; Marchal, S.; Hafeez, I.; Asokan, N.; Sadeghi, A.R.; Tarkoma, S. IoT Sentinel: Automated device-type identification for security enforcement in IoT. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 2177–2184.
13. Dorri, A.; Kanhere, S.S.; Jurdak, R.; Gauravaram, P. Blockchain for IoT security and privacy: The case study of a smart home. In Proceedings of the 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Kona, HI, USA, 13–17 March 2017; pp. 618–623.
14. Aman, M.N.; Chua, K.C.; Sikdar, B. Position Paper: Physical Unclonable Functions for IoT Security. In Proceedings of the 2Nd ACM International Workshop on IoT Privacy, Trust, and Security, Xi'an, China, 30 May 2016; ACM: New York, NY, USA, 2016; pp. 10–13. [CrossRef]
15. Moratelli, C.; Zampiva, S.; Hessel, F. Full-Virtualization on MIPS-based MPSOCs embedded platforms with real-time support. In Proceedings of the 2014 27th Symposium on Integrated Circuits and Systems Design (SBCCI), Aracaju, Brazil, 1–5 September 2014; pp. 1–7.
16. Heiser, G. Virtualizing embedded systems: Why bother? In Proceedings of the 48th Design Automation Conference, San Diego, CA, USA, 5–10 June 2011; ACM: New York, NY, USA, 2011; pp. 901–905.
17. Heiser, G. The role of virtualization in embedded systems. In Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems, Glasgow, Scotland, 1 April 2008; ACM: New York, NY, USA, 2008; pp. 11–16.
18. Gu, Z.; Zhao, Q. A state-of-the-art survey on real-time issues in embedded systems virtualization. *J. Softw. Eng. Appl.* **2012**, *5*, 277. [CrossRef]

19. Kanda, W.; Murata, Y.; Nakajima, T. SIGMA System: A Multi-OS Environment for Embedded Systems. *J. Signal Process. Syst.* **2010**, *59*, 33–43. [CrossRef]
20. Moratelli, C.; Johann, S.; Neves, M.; Hessel, F. Embedded virtualization for the design of secure IoT applications. In Proceedings of the 2016 International Symposium on Rapid System Prototyping (RSP), Pittsburgh, PA, USA, 1–7 October 2016; pp. 1–5.
21. Shin, K.G.; Ramanathan, P. Real-time computing: A new discipline of computer science and engineering. *Proc. IEEE* **1994**, *82*, 6–24. [CrossRef]
22. Xi, S.; Wilson, J.; Lu, C.; Gill, C. Rt-xen: Towards real-time hypervisor scheduling in xen. In Proceedings of the 2011 Proceedings of the International Conference on Embedded Software (EMSOFT), Taipei, Taiwan, 9–14 October 2011; pp. 39–48.
23. Smith, J.E.; Nair, R. *Virtual Machines: Versatile Platform for Systems and Processes*; Morgan Kaufmann: Burlington, MA, USA, 2006.
24. Pinto, S.; Pereira, J.; Gomes, T.; Ekpanyapong, M.; Tavares, A. Towards a trustzone-assisted hypervisor for real time embedded systems. *IEEE Comput. Arch. Lett.* **2017**, *16*, 158–161. [CrossRef]
25. Ukil, A.; Sen, J.; Koilakonda, S. Embedded security for Internet of Things. In Proceedings of the 2011 2nd National Conference on Emerging Trends and Applications in Computer Science (NCETACS), Shillong, India, 4–5 March 2011; pp. 1–6.
26. Pinto, S.; Oliveira, D.; Pereira, J.; Cardoso, N.; Ekpanyapong, M.; Cabral, J.; Tavares, A. Towards a lightweight embedded virtualization architecture exploiting arm trustzone. In Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), Barcelona, Spain, 16–19 September 2014; pp. 1–4.
27. Jithin, R.; Chandran, P. Virtual Machine Isolation. In *Recent Trends in Computer Networks and Distributed Systems Security, Proceedings of the Second International Conference on Security in Computer Networks and Distributed Systems (SNDS 2014), Trivandrum, India, 13–14 March 2014*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 91–102.
28. Security on ARM TrustZone. Available online: https://www.arm.com/products/security-on-arm/trustzone (accessed on 13 June 2018).
29. MIPS Multi-Domain Security. Available online: https://www.mips.com/products/technologies/mips-multi-domain-security (accessed on 13 June 2018).
30. Jithin, R.; Chandran, P. Dynamic partitioning of physical memory among virtual machines: ASMI: Architectural support for memory isolation. In Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, 4–8 April 2016; pp. 474–476.
31. MIPS64® Architecture for Programmers Volume IV-i: Virtualization Module of the MIPS64® Architecture. Available online: http://cdn2.imgtec.com/documentation/MD00847-2B-VZMIPS64-AFP-01.06.pdf (accessed on 10 May 2017).
32. Mhatre, S.C.; Chandran, P. On the Simulation of Processors Enhanced for Security in Virtualization. In Proceedings of the Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, Berlin, Germany, 9–13 April 2018; ACM: New York, NY, USA, 2018; pp. 51–52.
33. Ahn, J.; Jin, S.; Huh, J. Fast Two-level Address Translation for Virtualized Systems. *IEEE Trans. Comput.* **2015**, *64*, 3461–3474. [CrossRef]