IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 2 April 2024, accepted 2 May 2024, date of publication 9 May 2024, date of current version 28 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3399601

RESEARCH ARTICLE

HSP-V: Hypervisor-Less Static Partitioning for RISC-V COTS Platforms

JOÃO SOUSA, JOSÉ MARTINS, TIAGO GOMES, AND SANDRO PINTO
Centro ALGORITMI/LASI, Universidade do Minho, 4800-058 Guimaraes, Portugal

Corresponding author: João Sousa (id10564@alunos.uminho.pt)

This work was supported in part by FCT-*Fundação para a Ciência e Tecnologia* within the Research and Development Units Project Scope under Grant UIDB/00319/2020, Grant SFRH/BD/00297/2023, and Grant SFRH/BD/138660/2018; and in part by European Union's Horizon Europe Research and Innovation Program under the Project Cross-Platform Open Security Stack for Connected Devices (CROSSCON) under Grant 101070537.

ABSTRACT Virtualization technology has played a pivotal role in consolidating Mixed-Criticality Systems (MCS) onto a single computing platform. However, not all RISC-V processors present in Commercial Off-The-Shelf (COTS) platforms feature the so called Hypervisor extension, which poses a significant challenge in offering hardware virtualization capabilities in existing RISC-V silicon. This paper introduces HSP-V, a ready-to-run low-level software stack to provide static partitioning on RISC-V COTS platforms lacking hardware virtualization support. HSP-V leverages the Domain feature of the RISC-V Open Source Supervisor Binary Interface (OpenSBI) reference implementation to define partitions protected by the Physical Memory Protection (PMP) unit. Additionally, it provides other capabilities such as interrupt partitioning, direct interrupt injection, cache partitioning, and platform-level isolation for DMA-capable devices. The conducted evaluation assesses the impact of HSP-V on different empirical metrics, including domain boot time, interrupt latency, code size, and execution performance using micro and application benchmarks (LMBench and MiBench, respectively). HSP-V achieves highly deterministic interrupt latency with an average execution time of 457 ns (with a standard deviation of only 22 ns), with essentially zero traps in the Domain execution. In scenarios with cache interference, the HSP-V keeps the performance overhead as low as 0.39% for the best case scenario. Finally, all work described in this article is publicly available and open-sourced for the community to further evolve, port, and evaluate HSP-V in other hardware platforms.

INDEX TERMS Mixed-criticality systems, virtualization, static-partitioning, RISC-V, OpenSBI.

I. INTRODUCTION

Cyber-physical systems have evolved significantly in the past few decades [1], transitioning from single-purpose devices with limited communications and simple interfaces to power- and compute-hungry general-purpose systems with multiple functionalities and complex interactions [2]. To meet the demands for reduced size, weight, power, and cost (SWaP-C), there has been a paradigm shift towards the deployment of mixed-criticality systems (MCS), which integrate and consolidate different applications with distinct levels of criticality into a single hardware platform [3], [4], [5]. This approach requires spatial, temporal, and fault isolation among all subsystems, ensuring that subsystems with lower-criticality do not compromise the timing, functionality, or performance of the safety-critical ones. Virtualization stands out as the key enabler technology for consolidating MCSs, focusing on workload consolidation and isolation between different computing environments, e.g., operating systems (OSes), on a single hardware platform.

Hypervisors have been extensively used for virtualization, providing the ability to efficiently share resources, support different workloads according to the criticality level of the applications, and guarantee strong isolation between all instances. Currently, Hypervisors can span from minimalist approaches optimized for safety and security, e.g., static partitioning Hypervisors (SPHs) [6], to more feature-rich and resource-efficient solutions, such as Xen [7] and

The associate editor coordinating the review of this manuscript and approving it for publication was Ali Kashif Bashir.

KVM [8]. SPHs allocate fixed and dedicated resources to each virtual machine (VM) at design time, including central processing unit (CPU) cores, memory, devices, and interrupts [6], [9], [10], [11]. This minimalistic approach, specially tailored for MCS, guarantees the effective isolation and allocation of resources, which is mainly possible by leveraging instruction-set architecture (ISA) virtualization extensions [10], [12]. Such extensions can already be found in well-established computer architectures such as Arm (since Armv7-A [13]), Intel (introduced with Intel VT [14]), and more recently in RISC-V with the Hypervisor extension specification [12].

The RISC-V ISA [15] has been gaining traction across a wide range of computing domains, including MCS [12], [16], [17]. In comparison to other architectures, RISC-V is highly modular, reserving part of the encoding to custom extensions, enabling highly specialized implementations that can scale from simple microcontrollers to supercomputers. Regarding virtualization support, the Hypervisor extension [18], ratified in Q4 2021, is defined as part of the privileged ISA. Despite the extension being already supported in QEMU and several open-source soft-core RISC-V processors deployed in field-programmable gate array (FPGA) [9], [10], [19], its support in commercial off-the-shelf (COTS) platforms remains scarce, where only one silicon-based implementation (in the form of a test-chip) is known to be available [20], limiting the widespread utilization of Hypervisors with this architecture.

Targeting RISC-V hardware platforms lacking hardware virtualization support, this paper presents HSP-V, a Hypervisor-less static partitioning solution that allows the deployment of MCS systems in scenarios where using a Hypervisor is not feasible. HSP-V is based on OpenSBI,[1] the *de facto* Supervisor Binary Interface (SBI) firmware implementation for RISC-V. By leveraging the OpenSBI Domain feature, a system-level partition of underlying hardware having dedicated memory regions and harts (i.e., hardware threads, essentially cores in RISC-V lingo), HSP-V can run at the highest privilege mode, i.e., machine mode (M-mode), while keeping each partition running at the supervisor/user modes. Partitioning is achieved using memory isolation primitives widely available on RISC-V processors, such as the Physical Memory Protection (PMP). Although OpenSBI domains already embody the core requirements of a static partitioning system, some features are still missing, hampering the fully deployment of the functionalities provided by an SPH: (i) interrupt partitioning and domain assignment; (ii) inter-VM interference mitigation; and (iii) platform-level memory isolation for direct memory access (DMA) devices.[2]

The main contributions of this article are summarized as follows:

- The introduction of a Hypervisor-less static partitioning solution based on the OpenSBI reference implementation, specially designed for RISC-V COTS hardware platforms that lack the Hypervisor extension;
- Several contributions to the OpenSBI reference implementation, such as: (i) interrupt partitioning by mediating the access to the platform-level interrupt controller (PLIC); (ii) shared cache partitioning; and (iii) assignment of DMA-capable devices to different domains by using the platform-specific I/O memory protection Unit (IOMPU);
- A comprehensive evaluation of the HSP-V regarding code size, boot time and performance overhead, interference, and interrupt latency.

II. BACKGROUND

A. PARTITIONING TECHNOLOGIES

Partitioning technologies, such as TEEs and SPHs, play a pivotal role in modern computing systems. While a TEE provides a secure and isolated environment for sensitive operations and data with high levels of confidentiality and integrity, virtualization can be leveraged for workload consolidation and isolation between different computing environments on a single hardware platform.

1) TRUSTED EXECUTION ENVIRONMENTS (TEES)

A TEE involves splitting the system into two distinct worlds [22], i.e., a non-secure world mainly used for rich-OS support and applications, and a secure world that is commonly responsible for executing critical functionalities such as data encryption, fingerprint authentication, monetary transaction services, etc. Such secure services usually execute under a trusted application (TA) supported by a trusted OS. The main goal is to ensure that applications running in the secure world are protected and isolated from any interaction by any other component present in the system. Examples of TEE implementations include Intel SGX [23] and Arm TrustZone [24]. Intel SGX provides hardware-assisted trusted execution, creating secure enclaves to protect application code and data from any access from other software component, even those with root privileges. Similarly, Arm TrustZone offers hardware-based access control by enabling a processor to run in two isolated execution environments, i.e., the secure and non-secure world. TrustZone is widely used in mobile devices and ARM-based servers [25], hosting secure kernels such as Trustonic,[3] Qualcomm's QSEE,[4] and Linaro's OP-TEE,[5] ensuring the protection of security-critical data, and facilitating the deployment of various TAs with distinct functionalities.

[1] OpenSBI: https://github.com/riscv-software-src/opensbi

[2] OpenSBI support for technologies such as the I/O physical memory protection (IOPMP) [21] is not yet available, but it is on the project's roadmap.

[3] Trustonic:https://www.trustonic.com/

[4] Qualcomm: https://www.qualcomm.com/products/snap-dragon/security

[5] OP-TEE: https://github.com/OP-TEE/

2) STATIC PARTITIONING HYPERVISORS (SPHS)

Modern SPHs include Jailhouse [26], Xen Dom0-less [27], and Bao [10]. They all follow a minimalist implementation, on the order of a few thousand Source Lines of Code (SLoC), and they mainly perform the partitioning and assignment (with no sharing) of platform hardware resources, such as CPU, memory, devices, and interrupts, among the existing VMs. Since each virtual CPU (vCPU) is pinned to a single physical CPU, SPHs do not include a scheduler as part of the Hypervisor internals, achieving reduced size and complexity. All these features are mainly possible by leveraging dedicated ISA virtualization extensions [10], [12]. For instance, Arm's hardware virtualization provides a new higher privilege mode for the Hypervisor, offering support for interrupt virtualization, I/O memory management unit (IOMMU) that securely allows VMs to directly control DMA-capable devices [21], [28], [29], and the two-stage memory address translation. Nonetheless, this latter may represent a potential challenge to the security and real-time requirements of MCS.

Besides logical space and temporal isolation, MCS-oriented Hypervisors must also take into account the shared micro-architectural resources present in complex memory hierarchies of modern multi-core platforms, e.g., last-level caches (LLC), interconnects, and memory controllers, as critical subsystems can be sensitive to the timing variations resulting from contention on such components [30], [31], [32]. To mitigate inter-core interference at the Hypervisor level [33], several techniques have been proposed by the real-time research community, such as cache coloring [34], [35], [36], DRAM bank coloring [37], memory throttling [35], [38], [39], and I/O regulation [40], [41].

B. RISC-V

RISC-V is an open-standard ISA created as a research project in 2010 at the University of California, Berkeley [15], and currently managed by the non-profit RISC-V International. With a highly permissive license that allows for both open and proprietary implementations, its highly flexible and modular design enables different features, e.g., floating point, atomic and vector instructions, etc., to be added as extensions on top of the base integer instruction set (both on 32-bit and 64-bit instructions). Within the scope of this article, there are some key components that, working together, are essential to provide a flexible and secure computing environment that allows the deployment of the HSP-V architecture.

1) RISC-V PRIVILEGED MODES

The RISC-V privileged architecture specification [18] defines three base privilege modes (from higher to lower privilege): Machine mode (M-mode), Supervisor mode (S-mode), and User mode (U-mode). The only mandatory privileged level is the M-mode, commonly intended for hosting the firmware and that operates only with physical addresses, i.e., without virtual address translation. The S-mode and the U-mode are used to run OSes and applications, respectively. Typically, microcontrollers implement only the U-mode, while application class processors also implement the S-mode, which provides support for virtual memory and enables the execution of Unix-like OSes. In addition to these privileged levels, the privileged spec defines the Hypervisor extension, which introduces the concept of supervisor virtualization mode by adding two orthogonal, but less privileged, modes: the Virtual-Supervisor (VS) and the Virtual-User (VU). Furthermore, the S-mode is extended with Hypervisor functionalities such as control over two-stage translation and renamed Hypervisor-extended Supervisor mode (HS-mode) [12], [42].

TABLE 1. RISC-V Linux-capable platforms without the Hypervisor extension.

Platform	SoC	Security Features	Interrupt Controller
PolarFire SoC Icicle Kit	PolarFire SoC FPGA	PMP, MMU, IOMPU, Way-Locking	PLIC
BeagleV-Ahead	Alibaba T-Head TH1520 SoC	PMP, MMU, OTP, TEE System	PLIC
SiFive HiFive Unleashed	SiFive Freedom U540 SoC	PMP, MMU, OTP, Way-Locking	PLIC
SiFive Unmatched	SiFive Freedom U740 SoC	PMP, MMU, OTP, Way-Locking	PLIC
Nezha	Allwinner D1 SoC	PMP, MMU, IOMMU	PLIC
VisionFive	StartFive JH7100 64-bit Soc	PMP, MMU, OTP, TRNG	PLIC
VisionFive 2	StartFive JH7110 64-bit Soc	PMP, MMU	PLIC
Lichee RV Dock	Allwinner D1 SoC	PMP, MMU	PLIC

While QEMU and several open-source soft-core RISC-V processors deployed in FPGA already support the Hypervisor extension, its adoption in COTS platforms is currently limited [20]. Table 1 summarizes the landscape of widely used linux-capable RISC-V COTS platforms that lack the Hypervisor extension. Nonetheless, several security features are still supported, e.g., PMP, MMU, IOMPU (available in the PolarFire SoC Icicle Kit for protecting DMA-capable devices), Way-Locking (available in the PolarFire SoC Icicle Kit, in the Sifive Hifive Unleashed and in the Sifive Unmatched for cache partitioning) and others. For the interrupt controller, they all implement PLIC, which provides no interrupt partitioning or virtualization support.

2) PHYSICAL MEMORY PROTECTION (PMP)

The RISC-V provides a memory protection mechanism called PMP [18] that is capable of limiting supervisor and user (and optionally machine) mode accesses to the physical memory address space. For this reason, when virtual memory is present (enabled by the MMU), and a translation is needed, the PMP takes only effect after the memory translation. PMP is controlled from M-mode, allowing the definition of a whitelist for address space regions, each with different access rights (i.e., read, write, and execute) by configuring a set of Control and Status Registers (CSRs). Depending on the implementation, the PMP unit can use either 16 or 64 CSRs, thus limiting the maximum number of currently accessible memory regions. An access to a memory address not included in the whitelisted memory regions, or that violates its permissions, will cause an access fault and subsequent trap the system execution to M-mode.

3) INTERRUPTS AND PLIC

The RISC-V architecture includes three main classes of interrupts: (i) software interrupts, comparable to inter-processor interrupt (IPI); (ii) timer-based interrupts; and (iii) external interrupts. While software- and timer-based interrupts are considered local interrupts and managed by per-hart interrupt controllers, such as the Core-local Interrupt (CLINT[6]) or the Core-local Interrupt Controller (CLIC[7]), external interrupts are platform-wide and shared among all harts. The platform-level interrupt controller (PLIC[8]) is responsible for routing and multiplexing peripheral interrupts to all harts in the system, depending on how it is configured through an MMIO interface. The PLIC is able to multiplex up to 1023 distinct external interrupts to one or more hart contexts, i.e., a combination of a hart and its associated privilege level. As only M- and S-mode can receive interrupts, the PLIC typically has two contexts per-hart.

There are two main MMIO regions in the PLIC: (i) a global configuration region; and (ii) a per-context region for defining the harts priority mask and handling interrupts. Although the PLIC has been the standard interrupt controller for RISC-V since its inception, the new Advanced Interrupt Architecture (AIA)[9] [43] is now the reference interrupt controller that will supersede the PLIC. The AIA controller includes a redesigned PLIC, called the advanced PLIC (APLIC), which despite including the very same functionalities, it does not provide backward compatibility, i.e., systems or applications designed to work with the original PLIC are not compatible with the APLIC. Despite the emergence of AIA, current Linux-capable RISC-V COTS platforms still rely on the PLIC, as seen in Table 1.

C. OPENSBI

The RISC-V non-ISA SBI specification aims at providing an abstraction over low-level, implementation-defined, and platform-level components and mechanisms to ease the implementation and porting of supervisory software. It defines a number of run-time services meant to be provided by M-mode firmware, such as hart-state management, IPI-issuing, TLB invalidation, and shootdown. The OpenSBI project is an open-source reference implementation of the RISC-V SBI designed to be highly modular and easily adaptable to a wide range of RISC-V platforms, supporting different ISA extensions and non-ISA components. It can be directly used as the run-time firmware (or as a library included in external firmware or bootloaders), supporting the handling of misaligned memory accesses and the emulation at M-mode of extensions that are not implemented (e.g., Legacy and IPI Extension [44]), required by supervisor or user software.

[6]CLINT: https://github.com/pulp-platform/clint
[7]CLIC: https://github.com/riscv/riscv-fast-interrupt
[8]PLIC: https://github.com/riscv/riscv-plic-spec
[9]AIA:https://github.com/riscv/riscv-aia

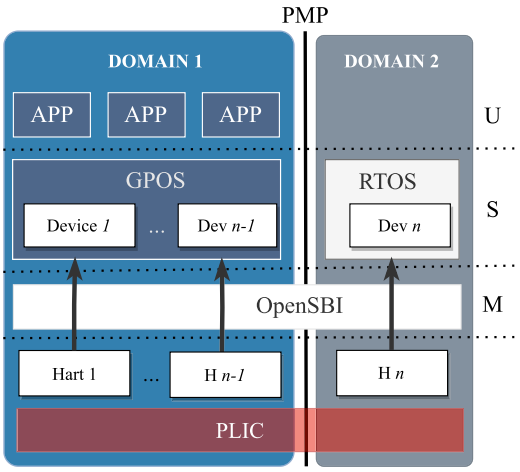


FIGURE 1. GPOS and RTOS configuration with vanilla OpenSBI.

The OpenSBI Domain system, is capable of partitioning the underlying hardware by assigning dedicated memory regions to one or more harts. Besides partitioning harts and memory/MMIO regions, OpenSBI also restricts the effect of the services it provides to the invoking domain's harts. For example, it will deny requests to send IPIs to harts which are part of other calling hart's domain. Figure 1 depicts an example OpenSBI domain system configuration comprising two domain instances running in S-mode: (i) one with a Unix-like OS with its applications running in U-mode; and (ii) the other with an RTOS configuration (FreeRTOS).

The initial boot stages are responsible for loading both OpenSBI's and the domains' images to the main memory, being the configuration passed in the form of a Device Tree (DT) node following a custom binding. If this node is not present, OpenSBI assumes a single "root" domain containing all harts, devices, and memory (excluding its own memory). Next, the OpenSBI domain instances are created with their harts and memory regions with respective access permissions, followed by some sanity checks to avoid any user misconfiguration such as domains' memory overlapping. Finally, it assigns each memory to its domains through the PMP entry setup and jumps to a pre-configured address in the domain's boot hart. The other domain's harts may be later woken up via the hart power-state management service.

D. CHALLENGES OF STATIC PARTITIONING WITHOUT VIRTUALIZATION EXTENSIONS ON RISC-V COTS

Designing a static partitioning solution without relying on virtualization extensions is not directly possible on current available RISC-V COTS platforms. Despite OpenSBI already implementing the core functionalities towards the goal of static partitioning with the domains system, the most important challenges still need to be addressed:

- **Interrupt partitioning by mediating a domain's access to the PLIC:** Some PLIC registers include the

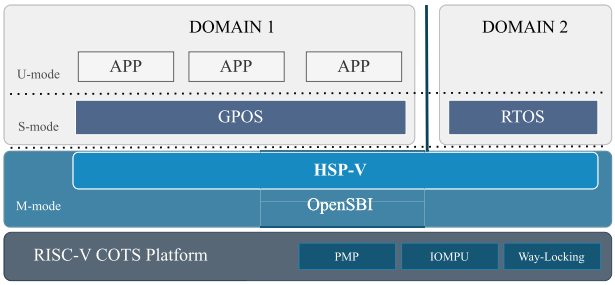


FIGURE 2. HSP-V architecture overview for the Microchip's PolarFire SoC FPGA Icicle kit.

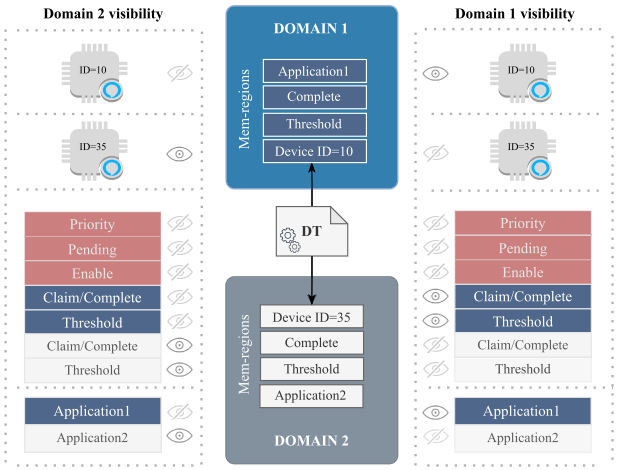


FIGURE 3. HSP-V system configured for two domains and their respective memory access permissions.

- configuration of multiple interrupts and contexts, with a few registers being shared between both domains. The shared PLIC address space across domains, as depicted in Figure 1, highlights the challenge of preventing one domain from interfering with the interrupts of adjacent domains. Additionally, it is critical to provide mechanisms to allow the execution of OSes with unmodified PLIC drivers (e.g., with trap-and-emulate).
- **Assignment of DMA-capable devices to different domains via the IOMPU:** It is mandatory to provide memory isolation at the system-level, i.e., including DMA devices in isolated domains.
- **Shared cache partitioning:** It is necessary to deploy mechanisms to mitigate inter-hart interference, namely contention for shared cache lines.

III. HSP-V IMPLEMENTATION

HSP-V is a static partitioning solution for RISC-V COTS platforms based on the OpenSBI implementation that defines partitions leveraging the RISC-V PMP. HSP-V provides other key features such as interrupt partitioning, direct interrupt injection, cache partitioning, and platform-level isolation for DMA-capable devices. Figure 2 depicts the high-level architecture of the HSP-V targeting the Microchip's PolarFire SoC FPGA Icicle Kit [45], previously introduced in Table 1. HSP-V uses the custom IOMPU to provide the I/O memory isolation features, and the available way-locking mechanism for cache partitioning.

A. OPENSBI DOMAIN CONFIGURATION ENHANCEMENTS

Domain configuration is done by adding an *opensbi-domains* node under the *chosen* node to the platform's hardware description DT file. Listing 1 includes the settings for the configuration illustrated in Figure 3, which is composed of two bare-metal application domains, each statically pinned to a single CPU and to a single device. The custom binding for this node includes two types of subnodes: *memory regions* and *domain instances*. A ***memory region*** node essentially defines a base address (*base*) and a size (*order*) which may refer to actual memory size or MMIO regions, while a ***domain instance*** defines a domain's configuration with four important properties: (i) ***possible-harts***, (ii) ***regions***, (iii) ***possible-devices***, and (iv) ***cache-partitions***, which are detailed below. Other *domain instance* node properties include (i) the *boot-hart*, (ii) the *next-mode* (next privileged level), and (iii) the *next-addr* (entry point address) for the domain.[10]

possible-harts: this property contains the pointer handle (*phandle*) for each hart assigned to the domain. In this specific configuration, Domain1 is assigned to *cpu1* and has total access (read, write, and execute permissions) to its application memory region (*Dom1MainMem*), while Domain2 is assigned to *cpu3* and has total access to its distinct application memory region (*Dom2MainMem*).

regions: this property defines the physical address space carvings assigned to the domain with an array of tuples, each containing a *phandle* to the *memory region* node, plus a bitmap for the assigned permissions (read, write, execute).

possible-devices: this property contains an array of *phandle* to devices assigned to that domain instance. This property was added to the vanilla OpenSBI configuration since a domain instance lacks device interrupt details and DMA-capable device's information. In the default configuration, to assign a device to a domain instance, it was necessary to create a dedicated memory region for that device. However, this information can be directly retrieved from the device node *reg* property. Thus, in the new configuration, the domain instantiation is able to retrieve the memory region for the device and its associated interrupts (assigning them to the partition as discussed in Section III-B), and to identify DMA-capable devices and their respective IOMPU ID (as discussed in Section III-D). This optimized device MMIO region assignment mechanism is able to streamline and simplify the configuration of domains, reducing the possibility of user misconfigurations.

[10]Vanilla OpenSBI Domain configuration: https://github.com/riscv-software-src/opensbi/blob/master/docs/domain_support.md

```
chosen{
    opensbi-domains {
    Dom1MainMem: Dom1MainMem {
        compatible = "opensbi,domain,memregion";
        base = <0x0 0x8020000>;
        order = <20>;
    };
    Dom2MainMem: Dom2MainMem {
        compatible = "opensbi,domain,memregion";
        base = <0x0 0x8010000>;
        order = <20>;
    };
    Dom1instance: Dom1instance {
        compatible = "opensbi,domain,instance";
        possible-harts = <&cpu1>;
        regions = <&Dom1MainMem 0x7>;
        possible-devices = <&periph_with_id10>;
        cache- partitions = <0x0F>;
        ...
    };
    Dom2instance: Dom2instance {
        compatible = "opensbi,domain,instance";
        possible-harts = <&cpu3>;
        regions = <&Dom2MainMem 0x7>;
        possible-devices = <&periph_with_id35>;
        cache- partitions = <0xF0>;
        ...
};};};
```

Listing 1. HSP-V configuration for the system configuration depicted in Figure 2 and Figure 3.

cache-partitions: This property represents a bitmap for the cache partitions assigned to a given domain, independently of the method used to partition the cache. For this specific configuration, the cache partitioning is done through domains by assigning four cache ways to Domain1 (*cache-partitions* property with $0x0F$), and different four cache ways for Domain2 (*cache-partitions* property with $0xF0$).

B. INTERRUPT PARTITIONING

When devices are assigned to domains, these must access the PLIC to configure and handle respective device's interrupts. However, concurrent and unsynchronized accesses to PLIC registers might result in unpredictable behaviour for the involved domains. Even if different domains cooperate to perform such accesses, this would still be a major security/safety threat/attack vector, as a malicious domain could intentionally interfere and tamper with other domains' interrupts. To prevent this, we implemented a partitioning mechanism for the PLIC into OpenSBI. The approach is based on the principle that (as explained in Section II-B) PLIC context MMIO regions are specific to a given hart, while the global configuration regions must be shared among all domains.

The implemented mechanism starts by verifying if no PLIC MMIO regions are defined in the *regions* properties, followed by configuring the PMP to allow domain access to its harts supervisor context MMIO region. As for the global configuration register, and for a realistic number of device interrupts and due to the limited number of PMP CSRs, it would be impossible to configure the PMP to grant access to the registers which pertain only to those interrupts. Furthermore, some registers configure multiple interrupts simultaneously on a per-bit basis - thus protection granularity cannot be enforced by the PMP. In light of these arguments, to protect and mediate access to this critical PLIC region (memory regions represented with color red in Figure 3, i.e., the *Priority*, *Pending* and *Enable* PLIC regions), the proposed mechanism uses the classical trap-and-emulate technique.

Since the RISC-V access control faults generate precise exceptions, when a domain tries to access the global PLIC region it traps to M-mode. OpenSBI uses the exception information CSRs (e.g., *mcause*, *mtval*, *mepc*) to read the fault instruction and decode the access to retrieve key information such as the access type (load or store), the destination/source register, and the access width. Since the exception program counter (*mepc*) carries information about the virtual address, it is required to set the *mstatus.MPRV* bit to read the instruction. When this bit is enabled, M-mode memory accesses are executed as if they were coming from the privilege level that generated the trap. Hence, when the domain has virtual memory enabled, the access is subject to address translation using the domain's page tables.

For the accessed address available through *mtval*, if virtual memory is enabled, it is necessary to perform a manual page-table walk to retrieve the actual physical address. Then, if this accessed physical address is indeed part of the critical PLIC region, the OpenSBI uses the PLIC emulation support. Based on the target address, the type of PLIC register being accessed is decoded. This access is then passthrough (or ignored) based on the accessing domain contexts and assigned interrupts. At the end of this process, the execution returns to the previous execution context and resumes from its last instruction. Nonetheless, trap-and-emulating this PLIC region will only result in significant overheads when configuring interrupts. These MMIO registers are not on the critical path for the interrupt handling, as they are typically only accessed during domain's initialization. The interrupts are still delivered directly to S-mode, and the PLIC registers touched during interrupt handling are directly accessible to domains without any traps.

C. CACHE PARTITIONING

In the realm of MCS, to comply with security and real-time requirements, minimizing deviations in the execution time is crucial for maintaining a deterministic behavior. However, contention at inter-hart (therefore at inter-domain) memory hierarchy could result in significant and unpredictable execution time, i.e., at shared micro-architectural resources. Specifically, regarding shared LLC, a given domain executing a memory intensive workload might inadvertently evict the cache lines of a critical domain, incurring in high memory access latency and low memory bandwidth, and potentially resulting in missing the execution deadlines. In a worse case, a malicious domain might intentionally evict such lines to perform Denial-of-Service attacks (DoS) [46], [47]

or even apply cache-side timing channel techniques (e.g., Prime+Probe [48], [49]) to retrieve information on the victim domain's data or execution flow.

The Microchip's Polarfire SoC features a Physically-Index/Physically-Tagged (PIPT) 2 MB shared and a unified L2 LLC following a 16-way set-associative topology [45]. Besides allowing the use of carve-outs directly as scratchpad memories, with essentially constant access times, it also provides a mechanism to lock cache ways with a per-master granularity, where each hart has two masters, i.e., one for the instruction cache and another for the data cache. A cache controller interface provides a *WayMask* register for each master, where each bit in the mask corresponds to one of the cache ways [45]. When a bit is clear in a master's mask, it indicates that this specific way cannot be evicted by that master. Nonetheless, this mechanism does not provide any logical isolation, as a hart can still read its masked ways. For the isolation requirements, the PMP unit is always needed. Thus, this locking mechanism is used to partition the LLC among the multiple domains according to the *cache-partitions* property of the domain's DT binding, i.e., a bitmap representation of the assigned cache partitions to that domain. Given the way-locking mechanism available in the Microchip's Polarfire SoC, each of the 16 least-significant bits in the *cache-partitions* property represents eviction rights over one of the cache ways.

At initialization time, and for each domain, the *WayMask* registers are set for the assigned harts instruction and data caches with the value of *cache-partitions*. The exception is the case when *cache-partitions* is not set in the domain's configuration. Hence, it is assumed that all cache ways are assigned to all domain's harts and consequently shared among them. Despite the advantages of including the *WayMask* register in the cache controller interface, regarding the DMA devices it essentially groups the different DMA channels into a single *WayMask* master. As a result, it is not possible to achieve a fully partitioned cache for certain device assignment combinations, i.e., locking ways for a specific DMA channel will also lock the same ways for other channels allocated to adjacent domains.

D. DMA PROTECTION

Unmediated access to the main memory by a domain's non-CPU bus master (i.e., a DMA-capable device) can result in data corruption (and thus the state and/or sensitive information) of other domains. To avoid this, the Microchip's Polarfire SoC includes a built-in IOMPU for each of these masters, including Ethernet, eMMC, and USB peripherals [45]. The IOMPU configuration registers essentially follow the same structure as the PMP CSRs. However, the number of regions for each IOMPU device varies from 2 to 16, e.g., 4 configuration entries for the MMC master block and 8 entries for Ethernet master blocks. As a result, since a domain is a set of harts and memories and each hart supports a maximum of 16 regions (i.e., 16 PMP entries), the number of regions for a given block master might be less than the number of regions assigned to its domain. With that in mind, the HSP-V approach configures the memory regions of these masters accordingly to the domains that will leverage the device. If there are still not enough registers to configure the domain's region, a fault is triggered and the system is fully halted before starting any domain. At runtime, in case a peripheral tries to access a region not present in its IOMPU regions, an interrupt is issued to OpenSBI, which acts by halting all harts belonging to a device's domain.

TABLE 2. SLoC and binary size (bytes).

		SLoC			Size (bytes)			
		c	asm	Total	.text	.data	.bss	Total
OpenSBI	utils	6236	27	6263	41597	1240	13224	56061
	platform	373	0	373	2247	192	544	2983
	sbi	8507	229	8736	52010	816	128376	181202
	firmware	33	860	893	49695	120	0	49815
	Total	15149	1116	**16265**	145549	2368	142144	**290061**
HSP-V	utils	7159	27	7186	44532	1240	15624	61396
	platform	480	0	480	2273	320	40	2633
	sbi	9130	229	9359	54680	824	128376	183880
	firmware	33	863	896	49691	120	0	49811
	Total	16802 (+10.9%)	1119 (+0%)	**17921** (+10.2%)	151176 (+3.8%)	2504 (+6%)	144040 (+1.3%)	**297720** (+2%)

IV. EVALUATION

The evaluation of the HSP-V was conducted on a Microchip PolarFire SoC Icicle Kit board [45], which features a SiFive E51 platform management hart, a quad-core U54 application cluster with per-core 32 MB L1 data and instruction caches, and a 2 MB shared L2 cache. The performed tests include HSP-V code size, boot overhead, execution performance and inter-domain interference, and interrupt latency.

A. CODE SIZE

This work extends the OpenSBI v1.0, adding significant features provided by HSP-V while maintaining the original code structure. Table 2 presents the SLoC and the final binary size of subsystem for both the vanilla OpenSBI and HSP-V, retrieved with the compiler optimizations set to -O2. The HSP-V adds about 1656 SLoC to the 16265 SLoC of the vanilla OpenSBI, which corresponds to an increase of around 10%. Most of the additional SLoC are in (i) the *utils* directory, specifically, in the DT parsing logic; (ii) the *platform*-dependent code with the driver implementation for applying the cache partitioning and the IOMPU register setup; and (iii) the *sbi* core that was enhanced with the PLIC trap-and-emulation code. On the other hand, the final binary file with the features added by the HSP-V is around 298 MB, which corresponds to an additional 8 MB (mostly on the *.text* section) to the original OpenSBI binary file (290 MB). Despite not completely negligible, the modifications required by the HSP-V dot not significantly impact the system's Trusted Code Base (TCB).

B. BOOT OVERHEAD

This evaluation consists in measuring the total boot time of a configuration with two single-hart domains that follows the boot sequence illustrated by Figure 4. Domain (i) consists of

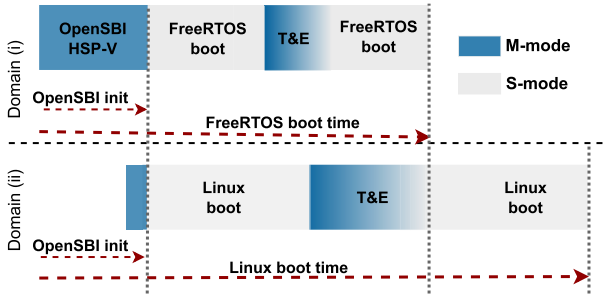


FIGURE 4. Boot sequence for a configuration with two domains under the same platform.

TABLE 3. Boot time (ms) of vanilla OpenSBI and HSP-V.

	Scenario	OpenSBI init. time (ms)		Total boot time (ms)	
		avg	std-dev	avg	std-dev
OpenSBI	freertos	80.134	0.303	94.140	0.748
	linux			6734.653	5.026
	freertos lock	80.202	0.200	94.049	0.200
	linux lock			6800.564	6.043
HSP-V	freertos	112.051 (+40%)	0.197	154.517 (+63%)	0.438
	linux			8375.438 (+24%)	5.510
	freertos lock	112.102	0.205	154.385	0.207
	linux lock			8417.998	5.358

a FreeRTOS configuration with a binary size of 57 MB, while Domain (ii) includes a Linux-based system with an image size 104 MB. The measurements include the total boot time (label: *OpenSBI init*) of both the HSP-V with the enhanced version of the OpenSBI, and the vanilla OpenSBI, as well as the boot execution time (labels *FreeRTOS boot time* and *Linux boot time*) of each domain, both represented by red arrows. Additionally, it was measured the execution time of the trap-and-emulation (label T&E) mechanism, which corresponds to the PLIC accesses for interrupt partitioning after setting up the domains, as well as the influence of enabling the cache partitioning feature. To carry out these measures (in clock cycles) the *rdcycle* pseudo-instruction was used, and the collected results are summarized in the Table 3.

Regarding the initialization time without the cache lock, the vanilla OpenSBI takes on average 80.134 ms to complete, while the OpenSBI with the HSP-V requires 112.051 ms to finish the initialization, corresponding to a boot time overhead of around 40%. With the cache locking mechanism enabled, these values further increase to 80.202 ms for the vanilla OpenSBI, and 112.102 ms for the HSP-V. For the total boot time without cache locking, the HSP-V requires around 154.517 ms for booting the FreeRTOS, and 8375.438 ms for booting the Linux system. These values, when compared to the native versions of both domains, correspond to an overhead of nearly 63% and 24%, respectively. This is mainly due to the trap-and-emulating operations required by the HSP-V for configuring the PLIC MMIO regions, which are not required in the vanilla OpenSBI.

C. PERFORMANCE OVERHEAD AND INTERFERENCE

To assess the performance overhead and the inter-hart/inter-domain interference, it was used the MiBench embedded benchmark suite's automotive subset, a reference benchmark widely used in the evaluation of MCS [6], [10], [12], and the LMbench [50], a suite of portable micro-benchmarks designed to measure various aspects of a computing system's performance.

1) MIBENCH

This benchmark suite consist of six different tests that execute in a single-hart Linux-based domain, containing four memory-intensive algorithms susceptible to interference caused by the LLC and memory contention, such as qsort, susan corners, and susan edges. The interference between harts/domains is introduced by a bare-metal application that runs on other three harts and executes a memory-intensive workload that continuously performs sequential writes to a 1.5 MB array with a stride equal to the cache line size (64 bytes). Each benchmark executed for four different system configurations: (i) hosted execution (*solo*), (ii) *solo* with cache locking enable (*solo-lock*), (iii) hosted execution under interference from multiple domains (*interf*), and (iv) *interf* with cache locking enable (*interf-lock*). For the tests including the cache locking mechanism, four cache ways (512 MB) were allocated to the bare-metal application, and eight cache ways (1 MB) to the Linux-based domain. The last four remaining cache ways (512 MB) are reserved to be used as scratchpad memory by OpenSBI. Figure 5 depicts the performance results using the *solo* configuration as the baseline, where each bar represents the average execution time of 1000 samples.

By enabling the cache partitioning (*solo-lock*), the overall performance decreases when compared with the *solo* configuration, which can be explained by the decreasing of the amount of available cache memory that is allocated to each domain. When stressing the system with interference (*interf*) caused by the bare-metal application running on the three remaining harts, the performance starts decreasing, especially in the memory-intensive benchmarks, i.e., the *qsort small* takes around 95.50 ms to complete (+50%), the *susan corners small* requires around 20.82 ms (+79.73%), and the *susan edges small* takes nearly 22.80 ms (+71.27%) to finish. With the *interf-lock* configuration, the cache partitioning mechanism mitigates the effect of this interference, which reduces the execution time of the previously mentioned memory-intensive benchmarks, i.e., the *qsort small* takes now around 78.90 ms to complete (+25.72%), the *susan corners small* requires now around 15.16 ms (+30.87%), and the *susan edges small* takes nearly 16.89 ms (+26.86%) to finish. Overall, the benchmarks handling smaller data sets (*-small*) are more susceptible to cache interference than the large versions.

2) LMBENCH

This benchmark suite targets UNIX systems and aims at measuring various aspects of a computer system's performance, such as memory latency and bandwidth, context switching,

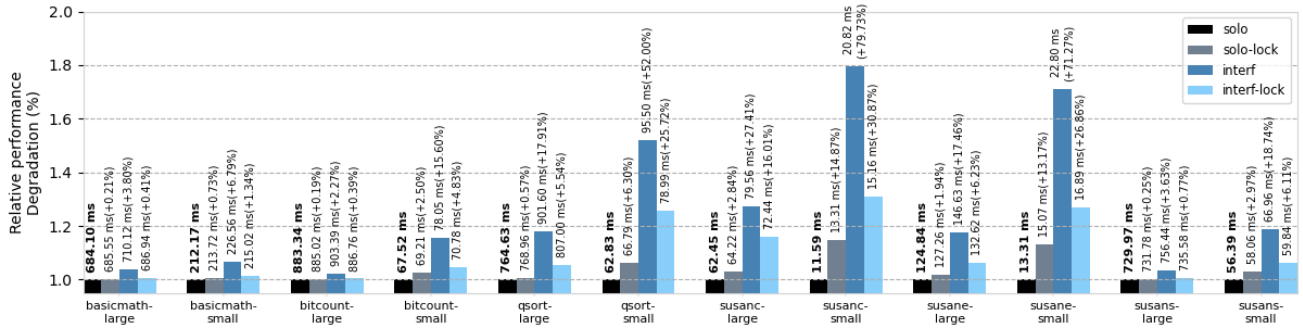


FIGURE 5. MiBench automotive benchmark suite performance results.

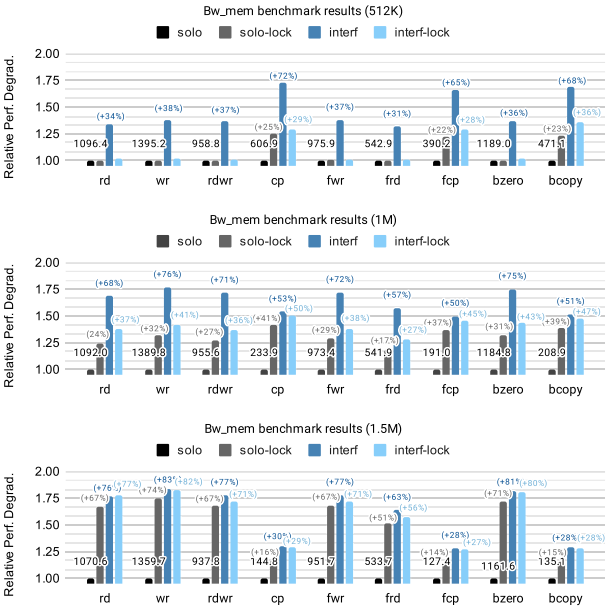


FIGURE 6. LMbench automotive benchmark suite performance results.

file system operations, and inter-process communication, among others. This evaluation only uses the *bw_mem* benchmark, which was used to evaluate memory operations bandwidth for different block sizes, i.e., 512 KB, 1 MB, and 1.5 MB, executed for the same system configurations as MiBench, i.e., for *solo*, *solo-lock*, *interf*, and *interf-lock*. The interference was caused by the same bare-metal (for *interf* configurations), and the cache locking mechanism followed the same way allocation as for MiBench, i.e., four cache ways to the bare-metal application and eight cache ways to the Linux-based domain. Figure 6 depicts the performance results using the *solo* configuration as the baseline, where each bar represents an average memory bandwidth in megabytes per second (MB/s) of 100 samples. For each sample, the micro-benchmark was configured with 10 warm-ups and 1000 repetitions (–W 10 –N 1000), encompassing 100000 samples (per bar).

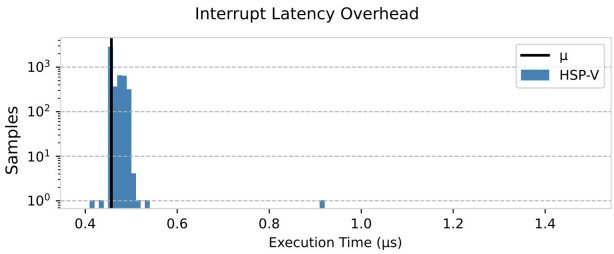
LMBench results reinforce the same conclusions as MiBench, with the behaviour of *solo*, *solo-lock*, *interf* and *interf-lock* configurations following the same pattern. Nevertheless, the results show that the relative performance of the system decreases with the increase of the workload (except for the copy operations). For the copy operations (*cp*), the relative performance degradation can be explained by the workload being equal (512 KB) or higher (1 and 1.5 MB) than the available LLC cache for each domain. As the *cp* operation uses two buffers (the source and destination buffers are cacheable), the size of necessary memory doubles (e.g., the workload of *cp* is two times the size of *wr*), making it the most memory-intensive micro-benchmark. For 512 MB workload, the memory bandwidth rates are 606 MB/s in *cp*, 390 MB/s in *fcp*, and 471 MB/s in *bcopy*; for 1 MB workload the memory bandwidth rates are 233 MB/s in *cp*, 191 MB/s in *fcp*, and 208 MB/s in *bcopy*; and for 1.5 MB workload the memory bandwidth rates are 144 MB/s in *cp*, 127 MB/s in *fcp*, and 135 MB/s in *bcopy*. Other experiments were performed with bigger memory workloads (ranging from 256 KB to 2 MB). However, the achieved results followed the same pattern.

D. INTERRUPT LATENCY

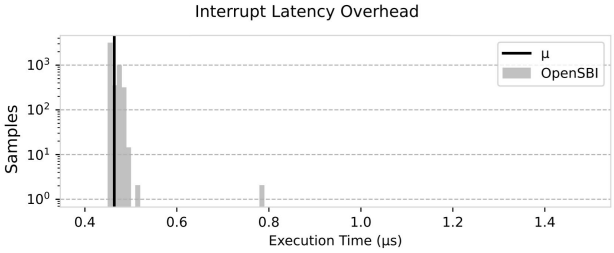
To measure the interrupt latency, a crafted minimal bare-metal benchmark application leverages an external timer peripheral configured in decrement mode with a 10 ms auto-reload period, to both trigger the interrupts and measure their respective delay. All measurements were taken with cold L1 caches, which, between each measurement, are invalidated with the *ifence* instruction and the data flushed by reading the content of a dummy array with the size of the cache. Figure 7 depicts the results in the form of 5000 samples histogram for two configurations: (i) the interrupt latency of the vanilla OpenSBI without PLIC partitioning (Figure 7a); and (ii) the interrupt latency in the HSP-V with PLIC partitioning (Figure 7b). The obtained results show that the HSP-V do not impact the interrupt latency, displaying a standard deviation of only 22 ns, with an average execution time of 457 ns. Such results correlate with what was previously explained in Section III-B, showing that the PLIC partitioning only causes traps to OpenSBI on interrupt configuration and not

TABLE 4. Comparison between HSP-V and similar systems: ● indicates the feature is present, and ○ otherwise.

		Partitioning Technologies			Security Design					Software Licensing
VE	System	TEE	Hypervisor	Other	Memory IO	Memory Core	Interference Mitigation Mechanisms	PLIC Part.	TCB	Open-Source
YES	Bao [10]	○	●	○	IOMMU	MMU (2^{nd} stage), MMU, Trap & Emulate	● Cache-coloring	●	<10 K SLoC	●
	Jailhouse [26]	○	●	○	WiP (IOMMU)	MMU (2^{nd} stage), MMU, Trap&Emulate	● Cache-coloring	○	10 K SLoC	●
	XtratuM [51]	○	●	○	IOMMU	MMU (2^{nd} stage), MMU, ARINC 653 standard	● SafeSU	○	n.a.	●
	Dom0-less (Xen) [27]	○	●	○	WiP (IOMMU)	WiP	○ n.a.	○	n.a.	●
NO	Multizone [30]	●	○	○	n.a.	PMP, Trap & emulate	○ n.a.	○	n.a.	○
	Keystone [52]	●	○	○	IOPMP	PMP, MMU	● Way-locking	○	~15 K SLoC	●
	VOSySmonitoRV [16]	○	○	●	n.a.	PMP, MMU	○ n.a.	○	n.a.	○
	HSP-V	○	○	●	**IOMPU**	**PMP, MMU, Trap&Emulate**	● **Way-locking**	●	**~15 K SLoC**	●



(a) HSP-V interrupt latency (with PLIC partitioning).



(b) OpenSBI interrupt latency (without PLIC partitioning).

FIGURE 7. Interrupt latency overhead.

on interrupt handling, as external interrupts continue to be directly delegated to the S-mode in the *mideleg* CSR.

V. HSP-V IN PERSPECTIVE WITH RELATED WORK

This section provides an overview of existing RISC-V static partitioning systems, such as Bao [10], Jailhouse [26], XtratuM [51], Dom0-less (Xen) [27], Multizone [30], Keystone [52], and VOSySmonitoRV [16], putting them in perspective with the HSP-V solution. Table 4 highlights their differences considering the following features: (i) hardware virtualization support; (ii) the partition technology adopted, i.e., TEEs, Hypervisors, or other approaches exploring hardware RISC-V security primitives to statically isolate resources across several environments; (iii) the security design features; (iv) and the software license.

A. HYPERVISOR TECHNOLOGIES

The most prominent open-source SPHs supporting (or working towards support) the RISC-V architecture (thanks to soft-core implementations such as Rocket [12], CVA6 [42], and NOEL-V [53], deployed in FPGA) are Bao [10], Jailhouse [26], Xen (Dom0-less) [27], and XtratuM [51]. Their static partitioning design defines CPU and IO memory accesses among all existing VMs. It adopts a 1-1 mapping of virtual to physical CPUs, with no need for a scheduler to mediate CPU allocation and ensure deterministic performance for each VM. Memory resources are statically assigned to VMs using two-stage address translation, where the second stage efficiently remaps each guest's physical memory to the corresponding Hypervisor's physical memory.

Considering Bao, the CPU and IO memory operate on a pass-through-only basis, i.e., with the IOMMU mediating DMA-related operations and MMU remapping the CPU accesses from each VM, resulting in no Hypervisor intervention for the VM execution. Bao explores an interference mitigation technique based on cache-coloring and supports PLIC partitioning through the deployment of a virtualization extension to the PLIC specification. In addition to all these features, Bao requires a TCB size of less than 10 K of SLoC. Regarding the Jailhouse, the current project's roadmap aims to include the IOMMU for direct device assignment to guests [9]. In terms of memory core management, Jailhouse uses the MMU second stage for page table walks, aiming to eliminate the Hypervisor intervention. However, it does not support PLIC partitioning, forcing the Hypervisor to apply trap and emulation mechanisms in all accesses to PLIC. Despite having access to the interrupt controller, the Hypervisor must manually inject interrupts into VMs, which heavily affects interrupt latency and real-time guarantees. Lastly, it deploys a cache-coloring mechanism for interference mitigation, and the TCB is as low as 10 K SLoC.

The XtratuM Hypervisor [51] implements a partitioning architecture based on the ARINC 653 standard, currently benefiting from MMU and IOMMU to perform spatial partitioning and a cyclic scheduling policy for temporal partitioning. Nonetheless, the latter may increase the frequency of translations, increasing the pressure on TLB and, consequently, introducing overheads due to TLB invalidation during context switches. Regarding the microarchitecture

partitioning, it introduces the Safe Statistics Unit (SafeSU), tackling inter-core interference and mitigating contention on shared buses. Regarding the Xen Dom0-less [27], despite supporting Arm-based architectures, its deployment for RISC-V-based cores is still underway, and only scarce information is available at the moment of the writing of this article.

B. TEE SOLUTIONS

Multizone [30] and Keystone [52] are among the most well-known TEE solutions currently supporting RISC-V. The commercially-available Multizone [30] operates on a two-operation mode infrastructure, encompassing M- and U-mode, and enables the execution of multiple isolated zones. Regarding the security design, it leverages a configuration file to specify read, write, and execute memory access control policies, and to map other physical resources such as I/O devices and their interrupts to each zone. PMP guarantee the core memory protection, while I/O protection, specifically the DMA accesses are controlled via software. Additionally, to offer seamless support for unmodified binaries and fully isolate the interrupt handling process between zones, it relies on the trap and emulation technique. Lastly, it features a formally verifiable code base implemented in assembly code with a small TCB size.

Regarding the Keystone [52] framework, it is an open-source project dedicated to build customizable TEEs based on RISC-V. The Keystone security design involves a secure monitor executing in M-mode, leveraging PMP for CPU memory protection. Additionally, it can benefit from various RISC-V implementations to enhance security in IO operations, i.e., non-standard IOPMP. The framework implements a cache partitioning mechanism based on real platform specifications, effectively minimizing micro-architectural interference through a way-locking technique. Notwithstanding, it's important to highlight that Keystone, unlike some counterparts, delegates interrupts to M-Mode, not providing enclaves with the capability to receive their interrupts.

C. OTHER STATIC PARTITIONING SOLUTIONS

Among the RISC-V static partition solutions that do not make use of virtualization extensions, VOSySmonitoRV [16] emerges as the system most closely comparable to HSP-V. VOSySmonitoRV operates at M-Mode, with a design tailored for MCS. The isolation is performed by assigning MMIO devices and CPUs to different partitions and offering memory access isolation via PMP and MMU. At the time of this writing, no information is available regarding PLIC, interrupt partitioning and micro-architecture mitigation mechanisms.

HSP-V stands out from previous solutions in two main aspects: (i) it is the first to implementing memory IO isolation relying on an IOMPU platform-specific feature, and (ii) it is the only solution that performs the partitioning of the native PLIC without relying on hardware virtualization support. This is achieved by enabling partitions to directly access specific PLIC registers without the need for trap-and-emulate mechanisms, and by allowing direct interrupt injection with with essentially zero traps in the Domain execution. The HSP-V implementation requires a higher number of SLoC when compared to other SPH included in Table 4. However this is mainly a consequence of building atop the vanilla OpenSBI code.

VI. DISCUSSION

HSP-V provides SPH features independently of hardware virtualization primitives. By leveraging a static partitioning design, HSP-V facilitates the consolidation of different criticality systems on top of the same hardware platform while fulfilling the requirements of MCSs in broader areas such as automotive, industrial control, aerospace, and medical fields. For instance, in the automotive field, we can easily find systems with different Safety Integrity Levels (SILs), where non-critical Quality Management (QM) and ASIL-A systems (e.g., infotainment and exterior lighting) coexist with critical ASIL-D systems (e.g., airbags, anti-lock brakes, and power steering). Furthermore, by leveraging the basic ISA security primitives ready-available in all RISC-V platforms (i.e., PMP), HSP-V can operate across various RISC-V platforms.

A. TRADE-OFFS BETWEEN HSP-V AND SPHS

Despite both SPHs and HSP-V relying on the same partitioning principles to consolidate different workloads onto the same platform, there are trade-offs in terms of performance overhead, interrupt latency, and scalability. Regarding performance, SPHs typically leverage 2-stage address translation, resulting in performance degradation of up to 7% when using 4KB of pages. Notwithstanding, SPHs can achieve minimal performance overhead levels (<1%) by utilizing large contiguous memory regions of 2MB (i.e., superpages), reducing the TLB misses. However, the granularity of superpages prevents the use of cache coloring mechanisms, which is typically used to mitigate interference in LLCs by up to 40%. In HSP-V, interference mitigation is obtained through cache-locking mechanisms, which reduce interferences by up to 50% while avoiding the penalty of performance degradation imposed by 2-stage address translation. Regarding interrupt latency, SPHs typically have a 4x increase in interrupt latency compared to the baseline despite using interrupt direct injection [6]. In contrast, HSP-V maintains native interrupt latency without introducing overhead during domain runtime execution, thus enhancing system execution performance and determinism, making it well-suited for MCSs. Lastly, scalability presents the main drawback for HSP-V. Its static partitioning design based on PMP is typically restricted to 16 entries, limiting its scalability. Workloads (akin VMs) running in partitions also need to cooperate in the (physical) address space, due to the lack of virtual memory support, hampering the use of legacy, pre-compiled binaries. Moreover, the cache partitioning mechanism is constrained by custom hardware

support, unlike the flexible coloring approach utilized by SPHs.

B. HSP-V UTILIZING PMP ACROSS DIVERSE PLATFORMS

The utilization of the PMP unit allows the HSP-V to be deployed in platforms with custom isolation primitives [25], or resource-constrained microcontrollers, for the consolidation of critical workloads [54]. The PMP primitive used for domain isolation can be less scalable and flexible than the virtual memory infrastructure leveraged by Hypervisors. For example, the limited number of PMP regions available in some systems, might not be enough for a specific target configuration, which is not an issue when using virtual memory.

C. USING IOMPU AS HSP-V'S IOPMP

HSP-V prevents data corruption from DMA-capable devices by leveraging the IOMPU platform-specific feature. However, only Polarfire and Nezha platforms include IO protection features, i.e., IOMPU and IOMMU, respectively, as summarized in Table 1. Notwithstanding, as part of future work, HSP-V could leverage the trap-and-emulate technique to control DMA-capable devices on platforms which not include IO protection. Additionally, HSP-V could also be adapted to support the future IOPMP RISC-V specification. Currently, the RISC-V community has been working towards the ratification of a standard named IOPMP [55], which shall mediate and manage device accesses to memory.

D. FUTURE PROSPECTS OF CACHE PARTITIONING

HSP-V incorporates cache partitioning through platform-specific features, i.e., the cache-locking mechanism. The presence of this mechanism extends beyond the Icicle board, encompassing other platforms such as those equipped with SiFive Freedom U540 and U740 SoCs. Given its unavailability on some platforms, the RISC-V community has been working towards establishing a standardized interface for seamless control over cache partitioning.

VII. CONCLUSION

This work presented and discussed the HSP-V, a Hypervisor-less static partitioning solution for the consolidation of MCS on commercially-available COTS RISC-V platforms lacking Hypervisor extensions. Based on the OpenSBI project, it enhances the configuration bindings by adding a interrupt partition mechanism through the mediation of the platform's interrupt controller. Furthermore, it includes a cache partitioning mechanism to split the last-level cache among Domains by mediating accesses using platform-level MPUs. For the demonstration and evaluation of HSP-V, it was used the Microchip PolarFire SoC Icicle Kit board, a widely-used COTS RISC-V platform. The collected results show that: (i) due to the latency imposed by the trap-and-emulation, HSP-V causes some overhead to the boot time; (ii) the overall performance is not affected by HSP-V; (iii) cache partitioning is a good approach for reducing the inter-domain interference; and (iv) the interrupt latency is not affected.

Hereafter, future steps will encompass: (i) the evaluation of HSP-V on other RISC-V-based COTS platforms; and (ii) the implementation of system recovery mechanisms on platforms with IOMPU features. This latter will enhance the HSP-V design (currently it only halts a domain when a DMA-access violation is detected) by forwarding the IOMPU error interrupt to the respective domain, which enables the possibility for the domain to take the appropriate recovery measures.

REFERENCES

[1] M. K. Habib and C. Chimsom I, "CPS: Role, characteristics, architectures and future potentials," *Proc. Comput. Sci.*, vol. 200, pp. 1347–1358, Jan. 2022.

[2] G. Heiser, "The role of virtualization in embedded systems," in *Proc. 1st Workshop Isolation Integr. Embedded Syst.*, Apr. 2008, pp. 11–16.

[3] A. Burns and R. I. Davis, *Mixed Criticality Systems—A Review*, 13th ed., Feb. 2022.

[4] M. Hassan, "Heterogeneous MPSoCs for mixed-criticality systems: Challenges and opportunities," *IEEE Des. Test.*, vol. 35, no. 4, pp. 47–55, Aug. 2018.

[5] Z. Jiang, N. Audsley, P. Dong, N. Guan, X. Dai, and L. Wei, "MCS-IOV: Real-time I/O virtualization for mixed-criticality systems," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2019, pp. 326–338.

[6] J. Martins and S. Pinto, "Shedding light on static partitioning hypervisors for arm-based mixed-criticality systems," in *Proc. IEEE 29th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, May 2023, pp. 40–53.

[7] J.-Y. Hwang, S.-B. Suh, S.-K. Heo, C.-J. Park, J.-M. Ryu, S.-Y. Park, and C.-R. Kim, "Xen on ARM: System virtualization using xen hypervisor for ARM-based secure mobile phones," in *Proc. 5th IEEE Consum. Commun. Netw. Conf.*, Jan. 2008, pp. 257–261.

[8] C. Dall and J. Nieh, "KVM/ARM: The design and implementation of the Linux ARM hypervisor," in *Proc. 19th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2014, pp. 333–348.

[9] R. Ramsauer, S. Huber, K. Schwarz, J. Kiszka, and W. Mauerer, "Static hardware partitioning on RISC-V: Shortcomings, limitations, and prospects," in *Proc. IEEE 8th World Forum Internet Things (WF-IoT)*, Oct. 2022, pp. 1–6.

[10] J. Martins, A. Tavares, M. Solieri, M. Bertogna, and S. Pinto, "Bao: A lightweight static partitioning hypervisor for modern multi-core embedded systems," in *Workshop Next Gener. Real-Time Embedded Syst.*, ser. OASIcs, vol. 77, 2020.

[11] Y. Shen, L. Wang, Y. Liang, S. Li, and B. Jiang, "Shyper: An embedded hypervisor applying hierarchical resource isolation strategies for mixed-criticality systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 1287–1292.

[12] B. Sá, J. Martins, and S. Pinto, "A first look at RISC-V virtualization from an embedded systems perspective," *IEEE Trans. Comput.*, vol. 71, no. 9, pp. 2177–2190, Sep. 2022.

[13] M. Dam, R. Guanciale, and H. Nemati, "Machine code verification of a tiny ARM hypervisor," in *Proc. 3rd Int. workshop Trustworthy Embedded Devices*, Nov. 2013, pp. 3–12.

[14] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith, "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48–56, May 2005.

[15] K. Asanovi and D. A. Patterson, "Instruction sets should be free: The case for RISC-V," EECS Dept., Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2014-146, Aug. 2014.

[16] F. Caforio, P. Iannicelli, M. Paolino, and D. Raho, "VOSySmonitoRV: A mixed-criticality solution on linux-capable RISC-V platforms," in *Proc. 10th Medit. Conf. Embedded Comput. (MECO)*, Jun. 2021, pp. 1–4.

[17] F. Cosimi, F. Tronci, S. Saponara, and P. Gai, "Analysis, hardware specification and design of a programmable performance monitoring unit (PPMU) for RISC-V ECUs," in *Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP)*, Jun. 2022, pp. 213–218.

[18] A. Waterman, K. Asanović, and J. Hauser, *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture*, document version 20211203, RISC-V Int., Dec. 2021.

[19] H. Lu and F. Zhang, "Raven: A novel kernel debugging tool on RISC-V," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, Jul. 2022.

[20] L. Valente, A. Veeran, M. Sinigaglia, Y. Tortorella, A. Nadalini, N. Wistoff, B. Sá, A. Garofalo, R. Psiakis, M. Tolba, A. Kulmala, N. Limaye, O. Sinanoglu, S. Pinto, D. Palossi, L. Benini, B. Mohammad, and D. Rossi, "Shaheen: An open, secure, and scalable RV64 SoC for autonomous nano-UAVs," in *Proc. IEEE Hot Chips 35 Symp. (HCS)*. Los Alamitos, CA, USA: IEEE Computer Society, Aug. 2023, pp. 1–12.

[21] A. T. Markettos, C. Rothwell, B. F. Gutstein, A. Pearce, P. G. Neumann, S. W. Moore, and R. N. M. Watson, "Thunderclap: Exploring vulnerabilities in operating system IOMMU protection via DMA from untrustworthy peripherals," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019.

[22] J. Van Bulck, D. Oswald, E. Marin, A. Aldoseri, F. D. Garcia, and F. Piessens, "A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1741–1758.

[23] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 86, Jan. 2016.

[24] S. Pinto and N. Santos, "Demystifying arm TrustZone: A comprehensive survey," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–36, Jan. 2019.

[25] D. Cerdeira, J. Martins, N. Santos, and S. Pinto, "ReZone: Disarming TrustZone with TEE privilege reduction," in *Proc. 31st USENIX Secur. Symp. (USENIX Secur.)* Boston, MA, USA: USENIX Assoc., Aug. 2022, pp. 2261–2279.

[26] R. Ramsauer, J. Kiszka, D. Lohmann, and W. Mauerer, "Look mum, no VM exits! (Almost)," 2017, *arXiv:1705.06932*.

[27] S. Stabellini, "Static partitioning made simple," in *Proc. Embedded Linux Conf.*, 2019. [Online]. Available: https://www.youtube.com/watch?v=UfiP9eAV0WA

[28] X. Li, X. Li, C. Dall, R. Gu, J. Nieh, Y. Sait, and G. Stockwell, "Design and verification of the arm confidential compute architecture," in *Proc. 16th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, Carlsbad, CA, USA, Jul. 2022, pp. 465–484.

[29] A. Holdings, "ARM system memory management unit architecture specification—SMMU architecture version 2.0," Tech. Rep., 2013.

[30] C. Garlati and S. Pinto, "A clean slate approach to Linux security RISC-V enclaves," in *Proc. Embedded World Conf.*, 2020.

[31] P. Burgio, M. Bertogna, I. S. Olmedo, P. Gai, A. Marongiu, and M. Sojka, "A software stack for next-generation automotive systems on many-core heterogeneous platforms," in *Proc. Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2016, pp. 55–59.

[32] M. Xu, L. T. X. Phan, H.-Y. Choi, Y. Lin, H. Li, C. Lu, and I. Lee, "Holistic resource allocation for multicore real-time systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2019, pp. 345–356.

[33] D. Costa, L. Cuomo, D. Oliveira, I. M. Savino, B. Morelli, J. Martins, A. Biasci, and S. Pinto, "IRQ coloring and the subtle art of mitigating interrupt-generated interference," in *Proc. IEEE 29th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*. Los Alamitos, CA, USA: IEEE Computer Society, Aug. 2023, pp. 47–56.

[34] T. Kloda, M. Solieri, R. Mancuso, N. Capodieci, P. Valente, and M. Bertogna, "Deterministic memory hierarchy and virtualization for modern multi-core embedded systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2019, pp. 1–14.

[35] P. Modica, A. Biondi, G. Buttazzo, and A. Patel, "Supporting temporal and spatial isolation in a hypervisor for ARM multicore platforms," in *Proc. IEEE Int. Conf. Ind. Technol. (ICIT)*, Feb. 2018, pp. 1651–1657.

[36] H. Kim and R. Rajkumar, "Predictable shared cache management for multi-core real-time virtualization," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 1, pp. 1–27, Dec. 2017.

[37] H. Yun, R. Mancuso, Z.-P. Wu, and R. Pellizzoni, "PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*. Los Alamitos, CA, USA: IEEE Computer Society, Apr. 2014, pp. 155–166.

[38] F. Farshchi, Q. Huang, and H. Yun, "BRU: Bandwidth regulation unit for real-time multicore processors," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2020, pp. 364–375.

[39] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory access control in multiprocessor for real-time systems with mixed criticality," in *Proc. 24th Euromicro Conf. Real-Time Syst.*, Jul. 2012, pp. 299–308.

[40] M. Zini, G. Cicero, D. Casini, and A. Biondi, "Profiling and controlling I/O-related memory contention in COTS heterogeneous platforms," *Softw., Pract. Exper.*, vol. 52, no. 5, pp. 1095–1113, May 2022.

[41] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, A. Schuster, and D. Tsafrir, "ELI: Bare-metal performance for I/O virtualization," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 411–422, Mar. 2012.

[42] B. Sá, L. Valente, J. Martins, D. Rossi, L. Benini, and S. Pinto, "CVA6 RISC-V virtualization: Architecture, microarchitecture, and design space exploration," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 31, no. 11, pp. 1713–1726, Nov. 2023.

[43] F. Costa, M. Rodríguez, B. Sá, and S. Pinto. (2023). *Open Source RISC-V Advanced Interrupt Architecture (AIA) IP*. [Online]. Available: https://api.semanticscholar.org/CorpusID:261050864

[44] P. Dabbelt and A. Patra. (2022). *RISC-V Supervisor Binary Interface Specification*. [Online]. Available: https://wiki.riscv.org/display/HOME/RISC-V+Technical+Specifications

[45] (2022). *Microsemi, Microchip Technology, PolarFire SoC MSS Technical Reference Manual*. [Online]. Available: https://onlinedocs.microchip.com

[46] M. Bechtel and H. Yun, "Denial-of-service attacks on shared cache in multicore: Analysis and prevention," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*. Los Alamitos, CA, USA: IEEE Computer Society, Apr. 2019, pp. 357–367.

[47] M. Bechtel and H. Yun, "Exploiting DRAM bank mapping and HugePages for effective denial-of-service attacks on shared cache in multicore," in *Proc. 7th Symp. Hot Topics Sci. Secur.*, Sep. 2020.

[48] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 605–622.

[49] M. Sabbagh, Y. Fei, T. Wahl, and A. A. Ding, "SCADET: A side-channel attack detection tool for tracking prime-probe," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2018, pp. 1–8.

[50] L. McVoy and C. Staelin, "Lmbench: Portable tools for performance analysis," in *Proc. USENIX éAnnu. Tech. Conf. (USENIX ATC 96)*. San Diego, CA, USA: USENIX Assoc., Jan. 1996.

[51] N.-J. Wessman, F. Malatesta, J. Andersson, P. Gomez, M. Masmano, V. Nicolau, J. L. Rhun, G. Cabo, F. Bas, R. Lorenzo, O. Sala, D. Trilla, and J. Abella, "De-RISC: The first RISC-V space-grade platform for safety-critical systems," in *Proc. IEEE Space Comput. Conf. (SCC)*. Los Alamitos, CA, USA: IEEE Computer Society, Aug. 2021, pp. 17–26.

[52] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proc. 15th Eur. Conf. Comput. Syst.*, Apr. 2020.

[53] J. Andersson, "Development of a NOEL-V RISC-V SoC targeting space applications," in *Proc. 50th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2020, pp. 66–67.

[54] D. Oliveira, T. Gomes, and S. Pinto, "UTango: An open-source TEE for IoT devices," *IEEE Access*, vol. 10, pp. 23913–23930, 2022.

[55] IOPMP Task Group. *RISC-V IOPMP Architecture Specification*. Accessed: Mar. 12, 2024. [Online]. Available: https://github.com/riscv-non-isa/iopmp-spec

JOÃO SOUSA received the master's in electronics and computer engineering. During the master's degree, he was a Visiting Student with the University of Würzburg, Germany. He is currently pursuing the Ph.D. degree with the Embedded Systems Research Group, University of Minho, Portugal. His main research interests include virtualization, operating systems, computer architectures, and the IoT systems.

JOSÉ MARTINS received the master's degree in electronics and computer engineering. During the master's degree, he was a Visiting Student with the University of Würzburg, Germany. He is currently pursuing the Ph.D. degree and a Teaching Assistant with the Embedded Systems Research Group, University of Minho, Portugal. He has a significant background in operating systems and virtualization for embedded systems. Over the last few years, he has also been involved in several projects on the aerospace, automotive, and video industries. He is the main author of the Bao hypervisor.

TIAGO GOMES received the master's degree in telecommunications engineering and the Ph.D. degree in electronics and computers engineering from the University of Minho. He is currently an Assistant Professor with the University of Minho, and an Auxiliary Researcher with ALGORITMI Research Centre, University of Minho. He counts with more than 50 scientific publications, focusing on the topics of hardware/software co-design for resource-constrained the Internet of Things devices, security in (reconfigurable) the IoT devices, and embedded hardware acceleration for multi-sensor perception systems targeting safety-critical automotive applications.

SANDRO PINTO received the Ph.D. degree in electronics and computer engineering. During the Ph.D. degree, he was a Visiting Researcher with Asian Institute of Technology, Thailand, University of Wurzburg, Germany, and Jilin University, China. He is currently an Associate Research Professor with the University of Minho, Portugal. He has a deep academic background and several years of industry collaboration, focusing on operating systems, virtualization, and security for embedded, CPS, and the IoT systems. He has published more than 70 peer-reviewed articles and is a skilled presenter with speaking experience in several academic and industrial conferences.