Implementation of a PMP unit in a RISC-V commercial core

Ferran Rubert Sánchez

February 15, 2024

Abstract– Implementation, integration and verification of physical memory protection (PMP) unit in Semidynamics commercial RISC-V core Atrevido423. PMP is an optional standard feature used for memory isolation in security critical systems that provides per-hardware-thread machine mode control registers that specify the access privileges for physical memory regions. An overview of the company, state-of-art on RISC-V cores, description of RISC-V architecture and privilege specification, is provided. An explanation of the implementation is given using the PMP unit of PULP core cva6 as an example. Finally, the integration with the core and various methods of verification used are described.

Keywords– RISC-V, PMP

Resum– Implementació, integració i verificació de la unitat de protecció de memòria física (PMP) en el nucli comercial de Semidynamics RISC-V Atrevido423. PMP és una característica estàndard opcional utilitzada per a l'aïllament de memòria en sistemes crítics de seguretat. Proporciona registres de control de mode màquina per maquinari que especifiquen els privilegis d'accés per a les regions de memòria física. Es realitza una visió general sobre l'empresa, l'estat de l'art sobre els nuclis RISC-V, la descripció de l'arquitectura RISC-V i l'especificació de privilegis, l'explicació de les implementacions posant com a exemple la unitat PMP del nucli PULP cva6. Per finalitzar, la integració amb el nucli i diversos mètodes de verificació que he fet servir.

Paraules clau– RISC-V, PMP

1 Introduction

In the landscape of modern computing, the choice of a processor architecture plays a key role in shaping the technological advancements of diverse applications. From the sprawling domain of edge computing, where resource-efficient devices power the Internet of Things (IoT) ecosystem, to the vast realm of data centers orchestrating complex computations for artificial intelligence, the selection of an architecture reverberates through the entirety of the computing spectrum. The RISC-V architecture, characterized by its simplicity, modularity, and open standard, has gained significant traction in the industry. In the fast-paced realm of RISC-V core development, safeguarding sensitive data is a top priority. The goal of this project is to implement and verify a Physical Memory Protection (PMP) unit within Semidynamics Atrevido 423, a high-performance RISC-V core.

The RISC-V architecture is an open standard instruction set architecture (ISA) based on established reduced instruction set computing principles. This means that anyone can access the ISA specification and build their own RISC-V processor without paying licenses or royalties. For example, Atrevido 423 is a RISC-V commercial processor. This means that the company has total control over the configuration and is fully customizable. Due to the proprietary nature of the core and the private work undertaken by the company, this project will provide an overview, focusing on the PMP implementation.

Protecting accesses to main memory is critically important. Computer systems usually run more than one process and are used by different users, each having different privileges and distrusting the others. Thus, some form of

• E-mail de contacte: 1461144@uab.cat
• Mencio realitzada: Enginyeria de Computadors
• Treball tutoritzat per: Lluís Terés Terés (DPTMISE, IMB-CNM), José María Arnau (Semidynamics)
• Curs 2023/24

memory protection is required to guarantee the integrity and confidentiality of the program code and data. In addition to assigning memory exclusively to just a single process (or user), it may also be shared among a group of processes or users. This can be used to make data available to others or even allow others to manipulate the data to create a communication channel.

As mentioned the PMP unit would be integrated on a RISC-V commercial core Atrevido 423 from Semidynamics. Semidynamics, a Barcelona-based company, is an European supplier of RISC-V IP cores, specialising in high-bandwidth high-performance cores with vector units targeted at machine learning and artificial intelligence applications [1].

2 Background

This section introduces some basic concepts required to understand the development of this project, such as a brief explanation of the core Atrevido 423 and architecture used.

2.1 RISC-V Cores

RISC-V is an open standard ISA developed by the Parallel Computing Laboratory at UC Berkeley. Unlike proprietary ISAs, such as x86 by Intel and ARM by ARM Holdings, RISC-V is open-source and freely available for anyone to use, modify, and build their own core. In addition, RISC-V can be customized with non-standard extensions. This work draws its foundation from the RISC-V Instruction Set Manual Privileged [2]. The focus will be on exploring key elements such as Control Status Registers (CSRs), Privilege modes, Physical Memory Attributes (PMA), and, notably, of Physical Memory Protection (PMP).

First and foremost, let's do a brief review on memory management. Memory management is the process of managing the physical memory resources of a machine in order to share the physical memory among multiple processes. Each process has its own virtual memory space, which is managed by the Operating System (OS). A process can access its own memory space using virtual addresses. The OS maps virtual addresses to physical addresses, reserving a certain amount of physical memory for one or more specific processes. This mapping is used for virtual address translation, which can be done through software or hardware support. With this management, each process has its own view of memory space, making memory isolation even more granular. The OS controls memory accesses from each process, so each process can only access its reserved physical space. In addition, application programming becomes easier as the programmer does not have to deal with physical memory fragmentation, physical memory hierarchy management, memory reallocation, and other issues.

In the RISC-V architecture, CSRs are special-purpose registers that control and monitor the processor's behavior. These registers are essential for system and application-level control, providing a standardized interface for privileged operations and status monitoring. CSRs are used to configure processor behavior, handle exceptions, and enable features like interrupts.

TABLE 1: Codification of the privileges modes

Nominal privilege code	Privilege mode	Abbreviation
00	User	U-mode
01	Supervisor	S-mode
11	Machine	M-mode

As shown in Table 1, there are 3 privilege modes with their own encoding. Nominal privilege code determines not only the current privilege mode, but also the privilege access required for each CSR. Each CSR is identified by an address (@CSR). By convention [2], this address belongs to a 12-bit encoding space that is divided into several parts.

- **Access type:** The bits @CSR[11:10] encode the access type. With a value of 11 it is set as read only, which means that the value of this register cannot be modified by an access. Otherwise, the register is set as read/write, which means that the value of the register can be modified or not.
- **Privilege:** The bits @CSR[9:8] encode the lowest privilege required to access the register, as shown in Table 1.
- **Address:** The @CSR[7:0] bits identify the register. In other words, these bits are the register address within the CSR system. As noted in the RISC-V documentation [2], there are reserved address spaces for either standard or custom implementations.

Machine CSRs are controlled by M-mode, which has access to all CSRs. M-mode is the highest privilege mode, along with debug mode, which is used for testing and verification purposes. M-mode is the first mode entered when the core is started or reset. Machine CSRs are already implemented in the core used. However, the Machine Memory Protection CSRs, pmpcfg0-pmpcfg15 and pmpaddr0-pmpaddr63 should be added. While detailing every CSR is beyond the paper's scope, one holds particular significance, the Machine Status register (mstatus). It tracks and controls the current operating state of the hardware thread. Two important bits in the mstatus register are **MPP (Machine Previous Privilege)** and **MPRV (Modify Privilege)**. MPP indicates the privilege mode that was in effect before the current machine-mode interrupt or exception was taken. MPRV controls how certain loads and stores access virtual or physical memory.

Privilege modes are environments used to isolate different executions. In this way, a security layer is added to prevent unwanted applications from managing sensitive data or taking control of the entire core or system. These privileges are managed by hardware and software. Usually there is a privileged mode that has full access to the system. For example, in the RISC-V specification, there is a machine privilege mode that has full access to the entire state of the core, even has access to other privileged modes. In other words,

the machine mode manages the behavior of the processor and the system. The other privilege modes are supervisor and user. Typically, the operating system uses the supervisor mode privilege to manage some functions of the system and resources. The least privileged is the user mode, which is used to run user applications.

The core used in this project is called Atrevido 423. It is a RISC-V processor developed by Semidynamics. The Atrevido 423 is a 64-bit customisable 4-way out-of-order processor that supports multiple extensions like bit manipulation, crypto and vector, complete MMU support, Linux-ready. Atrevido supports multiprocessing environments and can be configured as a coherent core with a CHI NoC or as a simpler, incoherent core connected via an AXI interface. Furthermore, with an improved TLB and MMU and support for SV39 and SV48, the core is well suited for running applications with large memory footprints using Linux. On the security side the accesses to physical addresses are checked in parallel using the PMA and the PMP. Customers can also optionally choose to protect the Data cache with ECC and the Instruction cache with parity, if required for their target markets. Furthermore, the Atrevido core is fully compliant with the latest RVA22 RISC-V profile [3].

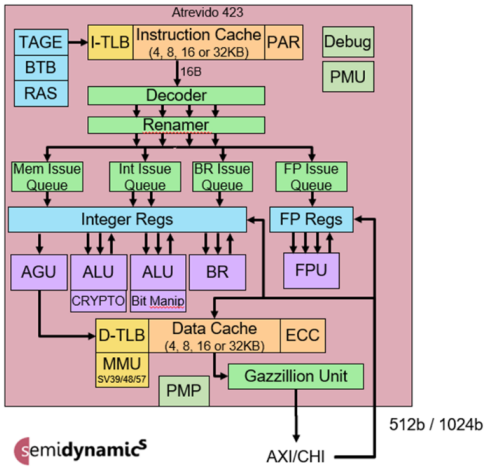

Fig. 1: Atrevido 423 block scheme.

The figure 1 shows the block scheme as a general overview of the entire core. Later on, there will be a more detailed block regarding the PMP integration to the core.

2.2 Memory Protection in RISC-V

Unlike some architectures, such as ARM with TrustZone [4], traditional RISC-V processors operate without dedicated isolation technologies. In reality, RISC-V incorporates two security mechanisms in consonance with the security layer that privilege modes add: the PMA and the PMP.

The physical memory map for a complete system includes various address ranges, some corresponding to memory regions, some to memory-mapped control registers, and come to vacant holes in the address space. Some memory regions might not support reads, writes, or execution; some might not support cache coherence or might have different memory models. In RISC-V systems, these properties and capabilities of each region of the machine's physical address space are termed physical memory attributes [2]. In short, PMA allows specifying attributes for different regions of physical memory. These attributes include settings such as readable, writable, executable, cacheable and other memory-related properties. Unlike PMP values, which will be described below, PMAs do not vary by execution context. The PMAs of some memory regions are fixed during the design of the chip. The specification also supports a programmable PMA which can be configured at run time to support different uses that imply different PMAs. Most systems will require that at least some PMAs are dynamically checked in hardware later in the execution pipeline after the physical address is known, as some operations will not be supported at all physical memory addresses. PMAs are checked for any access to physical memory, including accesses that have undergone virtual to physical memory translation. To aid in system debugging, it is strongly recommend that, where possible, RISCV processors precisely trap physical memory accesses that fail PMA checks. Precisely trapped PMA violations manifest as instruction, load, or store access-fault exceptions, distinct from virtual memory page-fault exceptions [2].

To support secure processing and contain faults, it is desirable to limit the physical addresses accessible by software running on a hart. A hart in the context of RISC-V architecture refers to a hardware thread. An optional PMP unit provides per-hart machine-mode CSRs to allow physical memory access privileges (read, write, execute) to be specified for each physical memory region, with the maximum allowance of 64 regions. Machine mode is the highest privilege level and by default has read, write, and execute permissions across the entire memory map of the device. However, privilege levels below machine mode do not have read, write, or execute permissions to any region of the device memory map unless it is specifically allowed by the PMP. For the lower privilege levels, the PMP may grant permissions to specific regions of the device's memory map, but it can also revoke permissions when in machine mode. PMP checks are applied to all accesses whose effective privilege level is supervisor (S) or user (U), including instruction fetches in S and U mode when the MPRV bit in the mstatus register is clear (mstatus.MPRV=0), and data accesses in any mode when the MPRV bit in mstatus is set (mstatus.MPRV=1) and MPP field in mstatus contains S (mstatus.MPP=0x1) or U (mstatus.MPP=0x0).

Each PMP region consists of an 8-bit *pmpXcfg* field and a 64-bit *pmpaddrX* register, which specifies the base address of the protected region. The extent of each region depends on the Addressing (A) mode. These 8-bit *pmpXcfg* fields are located within the 64-bit *pmpcfgY* CSRs. Each *pmpXcfg* field includes bits for read, write, and execute permissions, a two-bit address-matching field (A), and a Lock bit (L) shown in Figure 2. It is worth noting that overlapping regions are allowed, but in cases of overlap, the region defined by the lowest-numbered PMP entry takes precedence. Address matching bits (A) referes to the way boundaries of memory regions are calculated. The possible address matching modes are:

- **OFF:** PMP Entry disabled. No PMP protection ap-

plied for any privilege level.

- **TOR:** Top of the range. Two adjacent PMP address registers are used. The upper limit of this region is set by the value in pmpaddrX, and the lower limit (or base) is defined by the value in *pmpaddr(X-1)*. To check if a specific address *a* is within this region, we verify that *a* is greater than or equal to *pmpaddr(X-1)* and less than *pmpaddrX*.
- **NA4:** Naturally aligned four-byte. Supports only a four-byte region with four byte granularity.
- **NAPOT:** Naturally aligned power-of-2. When this configuration is set, the lower bits of the *pmpaddrX* register represent the size of the region, while the upper bits represent the base address right-shifted by two. In between these bits, there's a zero bit, often called the "least significant zero bit" (LSZB).

PMP allows for region locking, which means once a region is locked, you cannot change its configuration or address settings unless you perform a system reset. You lock a PMP entry by setting the "L" bit in the *pmpXcfg* register. If an entry is locked, writes to both its configuration (*pmpicfg*) and its address (*pmpaddri*) registers are ignored. The "L" bit also influences whether R/W/X permissions are enforced in M-mode (Machine mode). When "L" is set, these permissions are enforced for all privilege modes. When "L" is cleared, permissions only apply to S and U modes.

For machine mode, PMP checks do not occur unless the lock bit (L) is set in the *pmpcfgY* CSR for a particular region. For virtual address translation, PMP checks are also applied to page table accesses in supervisor mode. PMP violations are always trapped precisely at the processor. In effect, PMP can grant permissions to S and U modes, which by default have none, and can revoke permissions from M-mode, which by default has full permissions [2].

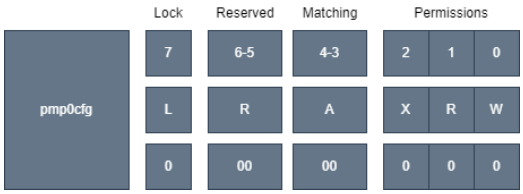

Fig. 2: PMP configuration register format.

The PMP system is designed to work alongside page-based virtual memory systems [2]. When virtual memory paging is enabled, instructions that access virtual memory may lead to multiple physical memory accesses, including references to page tables. PMP checks are applied to all of these accesses. The effective privilege mode for these implicit page-table accesses is supervisor mode.

Implementations with virtual memory systems may perform address translations speculatively and earlier than necessary for explicit virtual-memory access. PMP settings for the resulting physical address can be checked at any point between address translation and the explicit virtual-memory access. If there is a mis-predicted branch to a non-executable address range, it doesn't trigger an exception. Therefore, when PMP settings are changed in a way that affects either the physical memory containing page tables or the physical memory to which the page tables point, M-mode software must synchronize the PMP settings with the virtual memory system. This synchronization is achieved by executing an SFENCE.VMA instruction with rs1=x0 and rs2=x0 after writing to the PMP CSRs. In cases where a non-reserved section of the memory map doesn't have PMP permissions defined, supervisor or user mode accesses will be denied by default. However, machine mode access will be permitted. For access to reserved areas within a device's memory map, reading from these areas will return 0x0, and write attempts will be disregarded.

The Atrevido423 PMP supports 16 regions and an implementation of 16 PMP CSRs. Access to each region is controlled by an 8-bit configuration register *pmpXcfg* field and one address register *pmpaddrX*. The PMP unit implements the architecturally defined *pmpcfgY* CSRs *pmpcfg0* and *pmpcfg2*, supporting the 16 regions and the other *pmpcfgY* up to *pmpcfg14* are implemented, but hardwired to zero. The PMP values are checked in parallel with the PMA checks [2]. Since the PMAs are static and non-configurable, the PMP can only revoke read, write, or execute permissions to the PMA regions if those permissions already apply statically.

2.3 Related Work

RISC-V processors equipped with PMP support are already available in the market. The public SiFive E31 core complex manual provides insights into their PMP unit [5]. Notably, during RISC-V Summits, companies showcase projects and advancements related to their work and some of them are related to security and PMP usage. For instance, in 2018, Silicon Labs presented an embedded Real-Time Operating System (RTOS) that utilized PMP for achieving process separation and isolation [6]. Several companies have bolstered the security of their cores through the implementation of PMP.

In academic research, there is a notable paper on the verification of a PMP conducted by PhD students at the University of California, Berkeley [7]. The PULP organization's open-source research core, Ariane, incorporates PMP implementation. Additionally, other cores such as Lagarto I leverage PMP from the OpenPiton + Ariane core collaboration [8]. This project involves working on the Register-Transfer Level (RTL) implementation of PMP within the open-source core from the PULP platform, namely cva6 [9]. While the overarching idea is shared, the implementation has been adjusted to suit specific requirements of Atrevido423 core. It is important to note that details of the implementation cannot be disclosed due to confidentiality, but the discussion will touch upon common concepts. It is important to emphasize that this project operates independently, and there is no collaboration with or reliance on the implementation of the PULP platform.

3 Implementation of a PMP Unit

3.1 RTL Development of PMP Unit

The Verilog module presented encapsulates the functionality of a PMP unit, designed with flexibility through configurable parameters. These parameters include `PLEN` for the physical address length (defaulted to 34 bits for RV32), `PMP_LEN` representing the PMP configuration length (defaulted to 32 bits), and `NR_ENTRIES` specifying the number of PMP entries (defaulted to 4). Inputs to the module consist of the physical address to be examined (`addr_i`), the type of memory access (`access_type_i`), and the privilege level (`priv_lvl_i`). Additionally, configuration inputs (`conf_addr_i` and `conf_i`) define PMP settings. Within the module, a loop generates PMP entries based on the specified number, utilizing a submodule (`pmp_entry`) for comparison with the input address. The core logic for determining access permission is housed in an `always_comb` block, iterating through PMP entries and checking for matches with the input address. Access is granted based on configured security settings and privilege levels, considering whether the privilege level is not in Machine mode or if the configuration is locked (which applies in Machine mode as well). If no entry matches the address, the module allows all accesses in Machine mode and disallows accesses in other modes. Notably, if there are no PMP entries (`NR_ENTRIES == 0`), the module defaults to allowing all accesses. This comprehensive design ensures effective PMP-based memory access control, accommodating various configurations and providing security in accordance with RISC-V privileged architecture specifications.

The Verilog module titled `pmp_entry` constitutes a vital component in the implementation of the PMP, contributing to the assessment of whether a given physical address aligns with the specifications defined for a particular PMP entry. With configurable parameters, such as `PLEN` and `PMP_LEN`, the module takes inputs including the physical address (`addr_i`) and PMP configuration details (`conf_addr_i`, `conf_addr_prev_i`, `conf_addr_mode_i`). Outputs are provided through `match_o`, signaling whether the input address corresponds to the configured PMP settings. The module intricately employs bitwise operations and logical checks based on the specified address modes (e.g., TOR, NA4, NAPOT, OFF). Additionally, the implementation incorporates assertions for runtime verification, ensuring the accuracy of size extraction, overflow prevention, and other conditions in adherence to the chosen PMP address mode. This modular and versatile design plays a crucial role in evaluating and enforcing PMP-based memory protection within the broader context of RISC-V architecture.

In conjunction with the `pmp_entry` module, the effective implementation of PMP within the cva6 architecture relies on the utilization of the CSR file (`csr_regfile`). As detailed in the preceding sections, the `csr_regfile` serves as a central component, managing the various control and status registers that dictate the behavior of the processor. These registers include configurations related to PMP, among other critical functionalities.

Furthermore, it's noteworthy that within the cva6 architecture, the logic governing exceptions, a crucial aspect in memory protection and system stability, is centralized in the Memory Management Unit (MMU) module. This design choice consolidates exception handling within the MMU, providing a coherent and centralized approach to manage events such as page faults, access violations, and other memory-related exceptions. In this architecture, the MMU module plays a key role not only in memory translation but also in maintaining the integrity and reliability of the system by handling exceptions seamlessly.

3.2 Integration of PMP Unit in a Modern RISC-V Core

The integration with the core Atrevido takes place within the Load Store Unit (LSU). In the figure 3, an approximation of the core-memory interface is depicted.

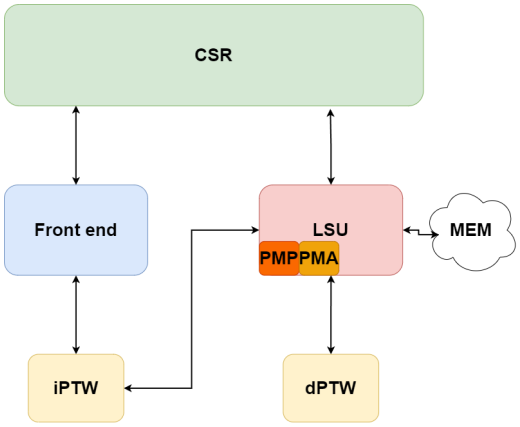

Fig. 3: Atrevido423 block scheme of the "core-memory" interface.

The Load Store Unit (LSU) is where all security checks are performed in parallel with the PMA. The LSU is responsible for executing all memory related instructions, that include scalar loads and stores, atomics instructions, CMO instructions, and vector load and store operations for all flavors. The Front-End is the part of Atrevido responsible for providing instructions to the Execution Units. The main blocks of the Front-End are the Branch Predictor, the Instruction Memory System and the Decoder. As being said Atrevido is a fully operative core and MMU, DTLB ready. Both the PMA and PMP modules receive the physical address, and subsequently, the signals are flopped to an exception logic. This exception logic is bifurcated into exception detection and exception reporting components. The exception logic plays a key role in determining the presence of an error and identifying the specific nature of the error. Additionally, the PMP module relies on information from the CSR file, as previously elucidated. This collaborative design, depicted in the following picture 4, ensures that security checks, address validation, and exception handling are seamlessly coordinated within the LSU, contributing to the robustness and reliability of the overall system architecture. This implies that modifications to the LSU and the CSR have ripple effects on other interconnected modules to maintain the core's seamless operation without introducing disruptions. As illustrated in the figure 3, this requires adjustments to the front-end as well. Specifically, the incorporation of the new PMP illegal signal prompts the need for changes in the front-end to appropriately handle the PMP

protection signal, which indicates the legality of memory access. To look at how PMP generally operates, let us suppose that a program is being executed by a RISC-V core and the next instruction in line is an addition instruction – ADD. For simplicity's sake, we can also assume that all relevant instructions and addresses are present in the instruction cache. In a conventional implementation, the core increments the program counter and fetches the instruction in a process of reading the content of a memory cell at the PC address. Typically the instruction is passed to the decoder logic, which is capable of discerning between different types of instructions, e.g. addition, subtraction, multiplication, etc. PMP however, when enabled, makes a check before decoding the instruction, to ensure that its execution is allowed.

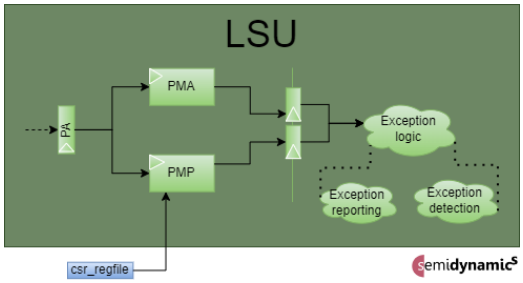

Fig. 4: PMP block scheme on the LSU of Atrevido423 block scheme.

3.3 Verification of the PMP Unit

It is important to note that any modified part of the core must be tested after design and before its implementation. So new test programs were implemented to ensure that these hardware modifications work as expected. The testing phase is ongoing. Integrating the PMP in a modern processor was a challenging task that took more time than expected. Although the verification is in progress and the PMP is working for many tests, a commercial processor requires a more extensive verification. Considering the intricate nature of the design and the fact that it involves a commercial core, the verification process demands meticulous attention. The emphasis is on achieving extensive coverage and ensuring a bug-free implementation. Recognizing the complexity and criticality of the task at hand, the testing efforts are geared towards meeting the stringent standards required for a commercial-grade core.

The following languages were used to develop and verify the PMP unit:

- **SystemVerilog:** based on Verilog, is a HDL that can be used to design, test, implement and do verification process for electronic systems such as processors or embedded systems. For this project, this language was used to design the PMP unit, as well as to modify and implement all the changes in the RTL, also is used in a propertary tool for functional verification.
- **C/C++:** Object oriented programming language (C++) which can use high level and low level structures. It is used to modify and implement a unit level testbench to test the new features.
- **RISC-V Assembler:** Low level language used to implement programs to test the new core's modifications.
- **Python:** A versatile programming language known for its simplicity and readability. Primarily utilized for general scripting.

After completing the RTL code, the next phase involves meticulous testing of all changes at the block level. To validate the new implementation of the CSRs, we design and execute assembly tests, incorporating read and write operations on the new PMP registers. These tests are compiled and subsequently simulated using Verilator, an open-source Verilog simulator renowned for its rapid simulation capabilities [10]. Additionally, we leverage licenses for Synopsys VCS, a proprietary Verilog simulator acclaimed for its industry-standard performance and rich feature set [11]. To ensure the accuracy of our implementation, we compare results with Spike, a RISC-V architecture simulator that inherently incorporates the PMP unit. While Spike serves as a reliable ground truth, occasional discrepancies are addressed through careful scrutiny. The initial phase of crafting RISC-V assembly tests proved challenging, demanding considerable time for test generation, debugging, and error resolution.

During the testing phase, the company acquired licenses for a tool that facilitates functional testing using SystemVerilog. This eliminates the need for manually generating RISC-V assembly tests and simulating with Spike, as the tool automates the verification process. Despite its time-consuming nature, this approach streamlines testing procedures and enhances efficiency.

The Universal Verification Methodology (UVM) serves as an industry-wide standard for developing testbenches for hardware modules implemented in Verilog. In our approach, we enhance an existing SystemC UVM testbench for the LSU by incorporating the new pmp features. This testbench includes an interpreter capable of reading pseudoinstructions and simulating the behavior of the RTL code within the core.

The primary testing scenario involves configuring the mode and address regions, followed by executing an arbitrary number of memory accesses with different types. Thorough validation of the PMP behavior requires considering various angles. One approach is to configure the system to intentionally violate rules and observe the raised exceptions. Conversely, we can set up the system to expect successful operations.

Certain features of the configuration space require to independent testing. For instance, a directed test can be designed specifically to assess the lock mechanism. Additionally, the execute permission's relevance during instruction fetches and the read and write permissions checked during loads and stores allow for separate unit tests. Integration tests then combine these features to ensure comprehensive validation of the PMP functionality.

One of the first tasks in testbench design is to define stimulus patterns, which will match a use case of the device as closely as possible. While it is possible to manually craft tests targeting certain features, our predominant approach involves the generation of extensive tests, from which we subsequently extract the relevant sections. To achieve this, I employed two distinct methods:

- **Genetic algorithms:** emulate natural selection processes to evolve a population of potential test inputs over multiple generations. These inputs are evaluated based on predefined criteria, such as coverage, and undergo processes like selection, crossover, and mutation to refine the test set.
- **Fuzzer:** AFL (American Fuzzy Lop) instruments the target program to monitor code coverage during execution and uses seed inputs to generate new test cases. It prioritizes inputs leading to increased code coverage, facilitating the discovery of vulnerabilities [12].

While genetic algorithms provide flexibility for diverse testing objectives, AFL is particularly effective in security testing, uncovering vulnerabilities through the exploration of different code paths. The choice between these approaches depends on specific testing goals and the nature of the system under test. Genetic algorithms offer flexibility, whereas AFL excels in security-focused fuzz testing. Both approaches have been employed, generating random tests for PMP using both AFL and genetic algorithms.

In Figure 5, the test generation process is developed through two methods. Subsequently, the generated tests undergo simulation using Verilator, enabling the extraction of toggle information and code coverage metrics. Furthermore, for a detailed analysis, functional coverage is pursued through the creation of covergroups and coverpoints in SystemVerilog. These covergroups and coverpoints are meticulously crafted to capture specific scenarios of interest, enabling a comprehensive assessment of the PMP logic.

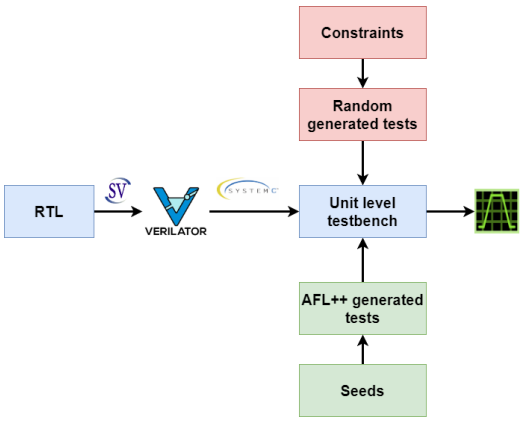

Fig. 5: Test generation flow.

4 Evaluation Methodology

4.1 Simulation Environment

The core has been simulated using the Semidynamics simulator, Spike but with a lot of changes. It needs specific versions of GCC compiler, Verilator and Linux to work properly.

The diagram in Figure 6 shows how the core is simulated and tested. A RISC-V compiler compiles the test programs using some linker files already provided by Semidynamics. A compiled file, such as a binary file, contains the program instructions that the simulated core will execute.

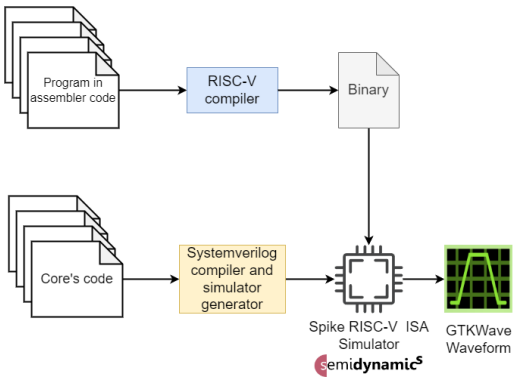

Fig. 6: Core simulation environment.

The register transfer language (RTL) code is compiled using other RISC-V compiler. The result is a simulator that emulates the behavior of the implemented core. Once the simulator executes a program, it produces a waveform of the multiple signals that the core has. This waveform shows the values of the signals at each clock cycle until the executed program ends. The analysis of these waveforms was conducted using gtkwave, a program renowned for its waveform analysis capabilities [13].

4.2 FPGA Platform

Semidynamics is currently using Vivado 2020.2. The Xilinx VCU128 serves as the Field Programmable Gate Array (FPGA) development board for our project. In our verification process, we initiate Linux on this board and execute a series of tests and benchmarks. To facilitate this verification, we utilize a tracer designed to function with the FPGA. This tracer operates by executing the same set of operations on Spike, a RISC-V architecture simulator [14], enabling a comparison to ensure that both the FPGA and Spike yield consistent results. This dual verification approach, combining real hardware with simulation, enhances the reliability and accuracy of our system evaluation on the FPGA.

5 Conclusions

This thesis presents a professional perspective on the sequential steps involved in integrating a new module into an existing operational commercial core. Specifically, for security enhancements, the incorporation of the PMP module is explored. The study delves into a notable open-source repository, the cva6 core, a well-regarded resource within the community. This open-source initiative serves as a valuable starting point, offering insights and support to individuals seeking experience in this domain. The initial community-driven effort sets the foundation for integrating the module into a commercial core, necessitating adaptability to make the integration seamless. Subsequently, thorough testing and verification become imperative, as reliability and trust are paramount in the industry. Utilizing advanced industry tools, tests are generated to achieve coverage, a critical metric in establishing the core's reliability and trustworthiness.

Although the concepts were straightforward, the implementation was not as simple due to the abundance of corner

cases that a complete implementation must deal with. Manipulating a commercial core of such complexity adds an extra layer of challenge, turning seemingly straightforward tasks into intricate endeavors, particularly for those without extensive expertise in the field. Fortunately, the guidance from the tutors and the assistance from Semidynamics colleagues proved indispensable during moments of confusion.

Two key points stand out: The first emphasizes the vast RISC-V community, teeming with dedicated professionals tirelessly working each day to craft an open-source ISA accessible to all. It is crucial to underscore that, as a private company, our work remains proprietary, and we do not actively engage with the open-source projects like for example the mentioned cva6. Nonetheless, emphasizing collaborative initiatives is crucial. The second point highlights the significant disparity between an experimental core and a commercial one, the latter necessitating clear, detailed steps and placing immense importance on verification. Verification, at times underrated, is a pivotal stage in commercializing a processor, as confidence in flawless functionality is paramount.

6 Future work

Even though the PMP implementation is complete, there's perpetual room for enhancement. Moreover, the development of new test programs is essential to verify diverse execution behaviors. Future endeavors will involve refining the testbench and continually testing and enhancing coverage, ensuring that new core features are seamlessly integrated without introducing issues or PMP-related bugs. In the dynamic landscape of a commercial core, where multiple engineering teams contribute changes, the work is ongoing. The continuous cycle of improvements and adaptations to evolving requirements ensures that there's always something to be addressed, particularly concerning features one has previously worked on. Continuos updating-integration is a key strategy for a continuous and seamelessly evolution.

Acknowledgments

Firstly, I want to sincerely thank my supervisor Lluís for all his work and guidance during this project. I greatly appreciate my family for their unconditional support. I want to express my deepest gratitude to my girlfriend, Clara, for her constant support and help at all times. Lastly, I want to thank my colleagues at Semidynamics for treating me so well and specially to José María for his supervision and help to achieve succesfully this project.

References

[1] Semidynamics. *About Us*. Semidynamics website. URL: https://semidynamics.com/about-us.

[2] *The RISC-V Instruction Set Manual Volume II: Privileged Architecture*. RISC-V Specifications. URL: https://riscv.org/technical/specifications/.

[3] Semidynamics. *Semidynamics Announces Fully Customizable, 4-way Atrevido 423 RISC-V Core for Big Data Applications*. Press Release. Launches at RISC-V Summit China on booth B2, August 23-25, 2023. 2023.

[4] ARM. *Arm TrustZone for Cortex-M*. Arm TrustZone technology page. URL: https://www.arm.com/technologies/trustzone-for-cortex-m#:~:text=Arm%20TrustZone%20technology%20is%20used,Learn%20More.

[5] *E31 Core Complex Manual*. StarFive Technology. URL: https://starfivetech.com/uploads/e31_core_complex_manual_21G1.pdf.

[6] Jean J. Labrosse. "Using the RISC-V PMP (Physical Memory Protection) with an Embedded RTOS to Achieve Process Separation and Isolation". In: *Software Architect* (2018).

[7] Kevin Cheang et al. "Verifying RISC-V Physical Memory Protection". In: *Department of Electrical Engineering and Computer Sciences, University of California, Berkeley* (202x). Email: {kcheang, crasmussen, dayeol, dkohlbre, krste, sseshia}@berkeley.edu.

[8] Neiel I. Leyva-Santes et al. "Lagarto I RISC-V Multi-core: Research Challenges to Build and Integrate a Network-on-Chip". In: *Supercomputing*. Ed. by Moisés Torres y Jaime Klapp. Cham: Springer International Publishing, 2019, pp. 237–248. ISBN: 978-3-030-38043-4.

[9] *PULP Platform - cva6 Repository*. GitHub repository. URL: https://github.com/pulp-platform/cva6/blob/pulp.

[10] Wilson Snyder. *Verilator*. Official Verilator website. 2003-2023. URL: https://verilator.org.

[11] Inc. Synopsys. *Synopsys VCS*. Official Synopsys product page. 2023. URL: https://www.synopsys.com/verification/simulation/vcs.html.

[12] Andrea Fioraldi et al. "AFL++: Combining Incremental Steps of Fuzzing Research". In: *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association, Aug. 2020.

[13] GTKWave Developers. *GTKWave*. Official GTKWave website. Year of the version or last update. URL: https://gtkwave.sourceforge.net/.

[14] *RISC-V ISA Simulator*. GitHub repository. URL: https://github.com/riscv-software-src/riscv-isa-sim.