Configurable RISC-V softcore processor for FPGA implementation

João Filipe Monteiro Rodrigues, Instituto Superior Técnico, Universidade de Lisboa

***Abstract*—Over the past years, the processor market has been dominated by proprietary architectures that implement instruction sets that require licensing and the payment of fees to receive permission so they can be used. ARM is an example of one of those companies that sell its microarchitectures to the manufactures so they can implement them into their own products, and it does not allow the use of its instruction set (ISA) in other implementations without licensing. The RISC-V instruction set appeared proposing the hardware and software development without costs, through the creation of an open-source ISA. This way, it is possible that any project that implements the RISC-V ISA can be made available open-source or even implemented in commercial products. However, the RISC-V solutions that have been developed do not present the needed requirements so they can be included in projects, especially the research projects, because they offer poor documentation, and their performances are not suitable. With this work, the main goal was the development of a RISC-V processor that has as characteristics the adaptability, flexibility, and features that are not yet present in the current solutions. As the base of this work, it was used a non-RISC-V processor, the MB-Lite, that was modified to implement this ISA, and it was improved with new functionalities, such as caches, data transfer through the PCIe bus, a serial communication module (UART), counters and timers, and multi-cycle functional units. The proposed solution was implemented and tested on an FPGA in order to validate the system's correct operation and to obtain its experimental characterization.**

***Index Terms*—RISC-V, instruction set (ISA), softcore architecture, FPGA.**

I. Introduction

OVER the years, the CPU market has been dominated by INTEL and AMD, in what concerns their x86 architectures, and by ARM, which is not a manufacturer but develops processor's designs mainly for mobile devices and microcontrollers. Those ARM designs are licensed to the manufacturers who integrate them into their products or use them to develop and sell their own System on Chips (SoCs). These processors are typically closed-source, and even their instruction sets cannot be used to develop third-party architectures without permission and the payment of royalty fees. All these limitations make the development of new processors more expensive and increase the difficulties to have a significant market share.

In 2010, the development of a new Instruction Set Architecture (ISA), called RISC-V, was driven at the University of California, Berkeley. RISC-V has the purpose of creating an extensible and open-source instruction set, not only for academic use but also for commercial products. Under this assumption, it is sometimes referred to as the "Linux" of processors. As a consequence, RISC-V has received significant support from the open-source community, with the adaptation and development of several programming tools. The RISC-V Foundation controls the RISC-V evolution, and its members are responsible for promoting the adoption of RISC-V and participating in the development of the new ISA. In the list of members are big companies like Google, NVIDIA, Western Digital, Samsung, or Qualcomm.

The main goal of this work is the development of a RISC-V softcore processor to be implemented in an FPGA, using a non-RISC-V core as the base of this architecture. The proposed solution is focused on solving the problems and limitations identified in the other RISC-V cores that were analyzed in this thesis, especially in terms of the adaptability and flexibility, allowing future modifications according to the applications needs and requirements. The software support is also taken into consideration, and several tools and libraries were developed to make the software development easier.

This document is organized as follows: Section II presents the RISC-V ISA, the analysis of several RISC-V cores, and a reference to other non-RISC-V cores is also done. Section III describes the development process related to the proposed architecture and its features. Section IV presents the developed software tools to interact with the processor. Section V includes the implementation process of the developed system on an Field Programmable Gate Array (FPGA), and it also presents the set of experimental results that were obtained. Finally, Section VI contains the conclusions.

II. RISC-V specification and implementations

In this section, the analysis of the RISC-V ISA specification is presented, focusing on the base integer instruction set and the existing extensions.

Then, several existing RISC-V cores are studied and compared. It is essential to understand what are the functionalities and drawbacks of each one to understand what is missing in the existing solutions and what is important to implement in a new solution. This analysis is essentially focused on the FPGA support, the documentation quality, and how easily they can be adapted and modified.

Finally, as a consequence of the current state of the existing RISC-V processors, other non-RISC-V cores are referenced.

A. *RISC-V ISA*

RISC-V includes a base integer ISA with 32, 64, and 128-bit variants (RV32, RV64, and RV128, respectively). Additionally, it supports optional extensions giving the flexibility to use what is necessary to implement in a specific application.

1) Base Integer Instruction Set

Currently, the RV32I (version 2.1) instruction set features 40 instructions [1], and it was designed to reduce the hardware resources, providing support for small implementations. At the same time, this ISA is recognized as a compilation target and is supported by several operating systems. The set divides the instruction into the following five groups: 21 Integer Computational Instructions; 8 Load and Store Instructions; 1 Ordering Instruction; 8 Control Transfer Instructions; and 2 Environment Call and Breakpoints instructions.

2) Extensions

One of the RISC-V main goals is to be suitable for all types of applications, starting from low-end applications, with hardware restrictions, where the base integer instruction set is enough, but also reaching high-performance processors which might require, as an example, floating-point operands or support for division and multiplication operations. This goal can be achieved by defining extensions to the instruction set, which can be implemented or not, according to the requirements and needs of each architecture.

By taking in consideration the main objectives of this thesis, a particular attention is given to the Integer Multiplication and Division Extension. This extension allows multiplication and division between two operands stored in two registers. In particular, we focus only on the 32-bit instructions (RV32M), since there are also instructions for 64-bit operands, but they were not used in this work. The instructions which comprise this extension are present in Table I.

Multiplication Instructions:

Four multiplication instructions are available, one of which is used to obtain the lower 32 bits of the result and the remaining three allow to obtain the upper 32 bits, according to the sign of the operands. If both lower and upper bits are required, the instructions MULH[S][U] and MUL should be executed sequentially, assuring that the operands registers are preserved.

Division Instructions:

There are two instructions for division, signed and unsigned, and two instructions to obtain the remainder. In these instructions, rs1 stores the dividend and rs2 stores the divisor. The division should round the quotient towards zero, and the remainder should have the dividend's sign, according to the C99 standard.

Table I: Integer Multiplication and Division instructions [1].

Instruction	**Meaning**
mul rd, rs1, rs2	rs1 x rs2 (lower 32 bits)
mulh rd, rs1, rs2	rs1 x rs2 (signed x signed, upper 32 bits)
mulhsu rd, rs1, rs2	rs1 x rs2 (signed x unsigned, upper 32 bits)
mulhu rd, rs1, rs2	rs1 x rs2 (unsigned x unsigned, upper 32 bits)
div rd, rs1, rs2	rs1 / rs2 signed division
divu rd, rs1, rs2	rs1 / rs2 unsigned division
rem rd, rs1, rs2	Remainder of *div*
remu rd, rs1, rs2	Remainder of *divu*

B. RISC-V Cores

The number of processors based on the RISC-V ISA has increased over time. Currently, there are processors to be used as simple microcontrollers, coprocessors, more advanced SoC with the capability to boot Linux and even more advanced implementations with multicore support and integration of customized accelerators. Several of them were selected to be studied compared based on their extensibility/customization, FPGA implementation support, and available documentation. These are the RISC-V cores that will be analyzed and compared: PicoRV32, Rocket Chip, ORCA, Potato, PULPino, VexRiscv, and SweRV.

Extensibility/customization:

From the set of processors reviewed, the ORCA and the VexRiscv support three different interfaces: AXI, Intel Avalon, and Wishbone. This way, these two processors are compatible with different interfaces used by different vendors. The PicoRV32 supports AXI and Wishbone, but it is not compatible with Intel Avalon. The Rocket Chip, the PULPino, and the SweRV are only compatible with AXI. The Potato only offers the Wishbone interface. These interfaces can be used to connect to external memories, DMAs, accelerators, or other modules like UART, I2C, and SPI as it is done in the PULPino.

In terms of the architecture, the Rocket Chip is the most extensible and modifiable processor because it supports multicore implementations with different types of cores, caches, and accelerators. Due to the existing interconnections, which allow the development of the SoC in an object-oriented way, each component uses a specific interface and can be easily attached and modified.

FPGA implementation:

Almost half of the analyzed processors have maximum operating frequencies under 100 MHz (the Potato, the PULPino, and the existing implementations of the Rocket Chip). On the other hand, the PicoRV32 is the processor with the highest frequency, 714 MHz in a specific Xilinx UltraScale+ device and a value between 250 and 450 MHz on the 7-Series Xilinx FPGAs. The VexRiscv achieves a maximum operating frequency that can range between 193 and 336 MHz, according to the configuration, when implemented on the Xilinx Artix 7. Finally, the ORCA processor, which was developed to support multiple vendors, is limited to 125 MHz on the Altera Cyclone IV FPGA. The SweRV does not offer any implementation for FPGA.

Documentation and Support:

The documentation of a processor is essential to understand how it was developed, the internal architecture, its features, its purpose, and how it can be used. From the processors referred above, the SweRV from Western Digital has the complete documentation certainly because it will be used in commercial products. The PULPino is also well documented, having documentation for the SoC and both cores are available.

Conclusion:

The platform with the best documentation is the SweRV, followed by the PULPino. However, when an existing FPGA implementation is a requirement, the SweRV does not offer support, and PULPino is not able to run on high frequencies. The cores with the best FPGA support are the PicoRV32 and the Vexriscv. Nevertheless, PicoRV32 is not pipelined, and both are not well documented. The Rocket Chip is the best solution when it comes to the extensibility, but it is not targeted to FPGAs, and the documentation about the SoC is poor.

Hence, it is reasonable to conclude that, the existing RISC-V softcores are not suitable to be used in research projects, since it was not possible to find a solution that reasonably meets all of these three topics.

C. Other non-RISC-V softcores

Currently, there are also available multiple non-RISC-V open-source processors that were developed in the past decade. Several of them have what is missing in the current RISC-V cores: proper documentation, FPGA support with high operating speeds and low resources usage, and a design which can be easily modified and adapted. Consequently, if we select one of these processors and modify the architecture to implement the RISC-V ISA, in the end, it should be theoretically possible to obtain similar results in terms of speed and area.

The MB-Lite [2], [3] softcore represents a promising possibility when focusing on the implementation of the requirements initially proposed and following the two-process design methodology [4], by J. Gaisler. Furthermore, it proved to be a reliable alternative in the field of the open-source softcores. It has a small size, but at the same time, it can achieve similar frequencies as its competitors. The way the code is organized and commented, the modularity and the interfaces it offers, along with the provided documentation and good practices followed during its development contribute to make this processor an excellent option to use in research projects.

The MB-Lite softcore was developed as part of T. Kranenburg's Master Thesis, at Delft University of Technology. It implements the Xilinx's 32-bit RISC MicroBlaze ISA, and it can execute programs compiled by the standard compiler without modifications. However, not all the MicroBlaze instructions were implemented to simplify the architecture.

The MB-Lite architecture, represented in Figure 1, is based on the MIPS processors, featuring a 5-stage pipeline with a modular implementation design, and two separated instruction and data memories. Each stage was carefully developed and described in separated components, following the same hardware description methodologies and signals naming. The exception is the Write-Back stage, which was implemented in the same component as the Instruction Decode, due to its connection to the Register File, but in this schematic, they were represented separately to show the traditional view of the pipeline.

III. Proposed Architecture

This section presents the architecture development of the proposed processor. The main focus is on the modifications that were introduced to the MB-Lite architecture and the implementation of the RISC-V ISA, as well as the new features, and the taken decisions related to the FPGA implementation. This presentation is introduced by an architecture overview of the processor, where the main structural changes are highlighted. In the following subsections, each component is explained in more detail.

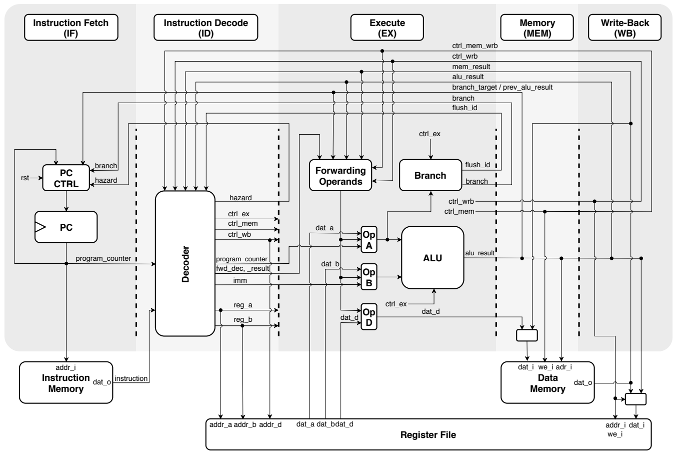

Figure 1: The MB-Lite architecture. The pipeline consists of 5 stages: Instruction Fetch, Instruction Decode, Execute, Memory, and Write-Back. Some signals and logic are not represented to simplify the figure.

A. Architecture Overview

By analyzing the MB-Lite original architecture, described in the previous section, one realizes that it offers fewer features than the existing RISC-V cores. Stands out the support for multi-cycle functional units, caches, and pre-built peripherals, such as UART, timers, and counters. Based on this observation, the solution that was developed during this thesis was focused not only on the implementation of the RISC-V ISA but also on keeping the MB-Lite characteristics, namely the methodologies used and its reliability. The new features of the architecture beyond the support for the RISC-V ISA include:

- Development of a multi-cycle divider and a multiplier and providing support for additional functional units.
- Support for the execution of some instructions out-of-order, after the ID stage.
- Development of a specialized unit for conflict detection and handling, caused by the out-of-order execution and other structural hazards.
- Creation of an extra writing port on the Register File to write-back the non-memory instructions directly after the EX stage, avoiding an extra stage.
- Development of peripherals (UART module, counter, timer, and data cache).
- Addition of a buffer that works as a waiting queue between the EX and the MEM stage, to store memory request instructions until they can enter to the MEM stage, in order to keep the issuing of instructions.

The processor architecture, represented in Figure 2, shows the changes that were introduced on the MB-Lite pipeline. The new signals and units are represented in orange. The EX stage now has support for a multi-cycle pipelined divider and multiplier, with configurable latencies, allowing out-of-order execution in the remaining stages. To detect and solve the hazards introduced by these modifications, it was necessary to develop and include a Dependency Handler unit on the ID stage. This unit is responsible for analyzing the current instruction type, the operands dependencies, and the state of the functional units, in order to issue or stall the processor whenever required.

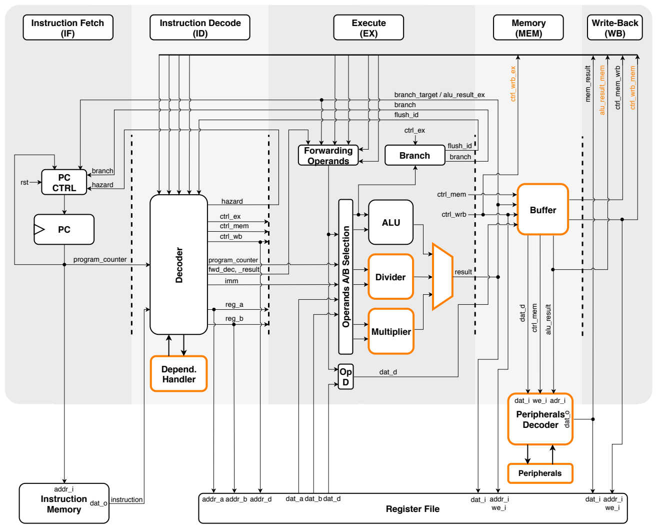

Figure 2: Processor Architecture schematic with simplifications. The new units and signals are represented in orange. A dependency handler unit was included on the ID stage to detect hazards. The EX stage features a multi-cycle pipelined divider and a multiplier. The Register File was modified to support two write ports. The MEM stage features a buffer to store instructions that are waiting for their request to be processed.

The MEM stage was modified to support memory requests with unknown latency. This requisite was necessary to support the interconnection of a data cache. By using an address decoder, it is possible to set the address map and connect the cache and other peripherals using the same interface.

To avoid stalls on the processor's pipeline every time an instruction is waiting on the MEM stage until the memory request is complete, different ways to continue the execution of instructions (without dependencies) were identified. The first modification consists in the implementation of a new write port on the RF and forwarding non-memory request instructions directly from the EX stage to the WB. The second improvement was the creation of an instructions buffer (Figure 3) with a configurable size on the entrance of the MEM stage to allow the issue of new instructions without dependencies, even when there is one instruction waiting on the MEM stage. If there is an empty space inside the buffer, the ID stage will continue issuing memory instructions.

B. RISC-V support

As previously referred, the MB-Lite processor is an open-source implementation of the MicroBlaze ISA [5]. To implement the RISC-V ISA, the required changes were mainly focused on the instruction decoder because it was necessary to change the parsing of each instruction due to differences in the formats.

The MicroBlaze ISA has only two types of instructions, Type A (used for register-register instructions) and Type B (used for register-immediate operations). On the other hand, the RISC-V ISA has six different types of instructions. New entries were added inside the decoder to parse the new types of instructions. Each instruction type is identified with a specific opcode since instructions of the same type share the same opcode. Then, after the type has been identified, the specific instruction is discovered and the operands are retrieved.

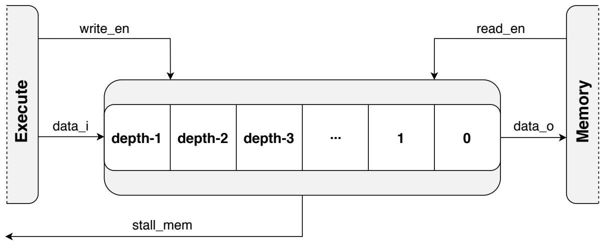

Figure 3: Memory stage instructions buffer. Based on the occupation of this queue, the processor will continue issuing new memory instructions in case of having free space.

C. Memory Structure

In a Harvard architecture, instructions and data are stored in different memories. This model allows a simultaneous access to data and instructions, but it is not possible to mix the program and data addressing space, for instance, to load a new program as data before the execution. Consequently, this processor is designed to be used as a slave, because each program needs to be previously transferred before its execution.

The initial goal was the implementation of two caches, one for instructions and the other for data. Due to time limitations, only one was implemented and the priority was given to the data cache because, the data part has tendency to be larger than the instructions part.

Beyond the instructions and data memories, the processor supports peripherals that are part of the address space. They are accessible via memory instructions, which are handled by the address decoder and forwarded according to the address map. The default address space represented in Table II features 16KB for instructions, 256B for peripherals, and nearly 4GB for data, allowing a large configuration space according to the user's preferences.

Table II: Address Map.

Region	Start Address	End Address	Size
Instructions	0x0000 0000	0x0000 3FFF	16KB
Data (cached)	0x0000 4000	0xFFFF FEFF	4GB
Peripherals	0xFFFF FF00	0xFFFF FFFF	256B

1) Instruction memory

Since the instruction cache was not implemented, the program instructions are stored inside BRAMs. The memory should have one port to receive and send data across the Peripheral Component Interconnect Express (PCIe) interface and a second port connected to the processor to fetch the instructions. Xilinx offers a BRAM generator [6] through the Vivado's IP Catalog, which supports several types of RAMs with configurable sizes. According to the architecture requirements, the True Dual-Port BRAM is the right option because it provides two read and write ports for different addresses simultaneously and two independent clocks. This way, it is possible to connect one of the ports to the receive the data that is sent via PCIe, but due to implementation requirements, this interface must be compatible with AXI4. So, we need to use the Xilinx's AXI Block RAM Controller IP to make the conversion between both interfaces, because BRAMs are not compatible with AXI.

The connections between the AXI Block RAM Controller and the instruction memory, implemented using a True Dual-Port BRAM, are represented in Figure 4, where is also represented the core connected to the other port. It is represented the two independent clock signals, one from the PCIe logic and the dedicated core clock.

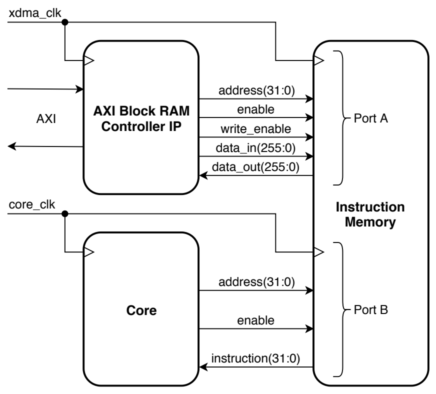

Figure 4: Connections between the Core, the Instruction Memory (implemented with a True-Dual Port RAM IP), and the AXI Block RAM Controller IP.

2) Data memory and cache implementation

The adopted cache uses an existing implementation as its base structure, requiring several modifications to support 32-bit words, lines with a larger dimension, a configurable number of lines, and data invalidation when the processor is reset. The cache can be configured to use two different configurations: direct-mapped or 2-way set associative with the Least Recently Used (LRU) data replacement policy. Both types use the Write-Back policy, where the date is only written to the main memory when the cache line needs to be replaced by a new loaded line. This write policy allows reducing the number of memory accesses when compared to the Write-Through policy, since this last policy keeps the memory always updated. Figure 5 shows the structure of the default cache with 2 ways of associativity and 16 lines, each with 256 bits, featuring one valid and one dirty bit per line. This cache has a total capacity of 1KB, corresponding to 256 32-bit words.

Depending on the program type and the amount of data needed in the beginning and during its execution, the main data memory could require a storage space higher than the available BRAMs can provide. The solution goes through the use of DRAMs present on the board to store the processor's execution data.

The connection between the design implemented on the FPGA and the DRAM is made through the Xilinx Memory

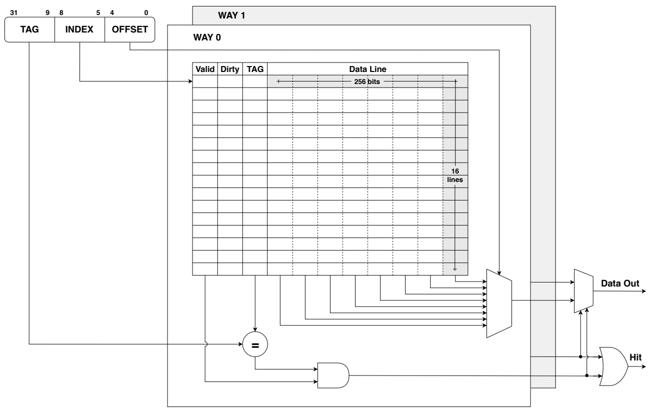

Figure 5: 2-way set associative cache structure. Each way is comprised of 16 lines with 256 bits each, the tag, a valid bit, and a dirty bit.

Interface Generator (MIG) [7]. This IP is responsible for generating memory controllers and interfaces, simplifying the design process by providing the constraints and Verilog or VHDL design implementations automatically.

D. Multi-cycle Functional Units

With the addition of support for the RISC-V Integer Multiplication and Division extension, the core requires the implementation of a multiplier and a divider. Typically the multiplication and division are too complex to be executed on a single clock cycle without reducing the frequency. The multi-cycle functional units solve this problem because they need several clock cycles to generate the result. By using the Xilinx IPs, a pipelined multiplier unit and a pipelined divider with configurable latency were developed to be implemented in the Execute stage in parallel with the ALU. The multiplier IP is configured with a latency of 6 clock cycles, and the divider latency was set to 30 clock cycles. It is higher due to the complexity of the division algorithms.

1) Integer Multiplier

The integer multiplier unit developed for this processor includes a multi-cycle pipelined Xilinx IP multiplier [8]. The IP settings allow the configuration of the type of the operands (signed or unsigned), its sizes, and the latency required to compute the result. The RISC-V multiplication instructions (see Table I) use 32-bit operands, signed or unsigned. Since the multiplier unit must support both types of operands, but the IP, after generated, only support one type, it was decided to define the operands always as signed. Then, by using an extra bit, the sign can be extended when the operand is signed, but otherwise it has the value zero. This selection is made by the control signals a_sign and b_sign, which are dependent on the operation type. Thus, the multiplier IP inputs are 33-bit wide, and the output result is 66-bit wide (2×33 bits), but the two most significant bits are discarded. According to the multiplication instruction, the output value can be the lower or the higher 32 bits of the result. There is a control signal (low_high) that chooses the desired part. Figure 6 shows the structure of this unit.

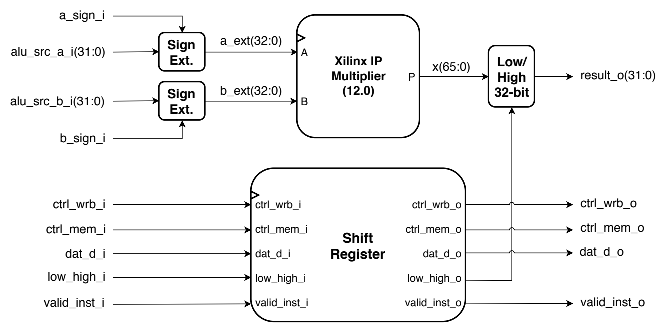

Figure 6: The Multiplier Unit. Since the multiplier is pipelined, it is necessary to have a shift register to store the control signals of each operation.

2) Integer Divider

The development of the integer divider unit, represented in Figure 7, followed the same principles of the multiplier. This time it was used a multi-cycle Xilinx IP divider, with different division implementations: LutMult, Radix-2, and High Radix. According to the IP's manual [9], the LutMult solution limits the dimension of the dividend and the divisor to 17 and 12 bits, respectively; the High Radix only supports fractional remainders; the Radix-2 allows dividends and divisors up to 64 bits, fractional and integer remainders and it is recommended for applications which require high throughput. Therefore, Radix-2 is the only solution compliant with the ISA requirements.

The divider IP always computes the quotient and the remainder, and both are available in its outputs. Depending on the instruction, the desired result can be the quotient or the remainder. This selection is made by the signal div_rem, generated in the Decode stage. According to the C99 standard, the quotient is rounded towards zero, while the remainder's signal is equal to the dividend's.

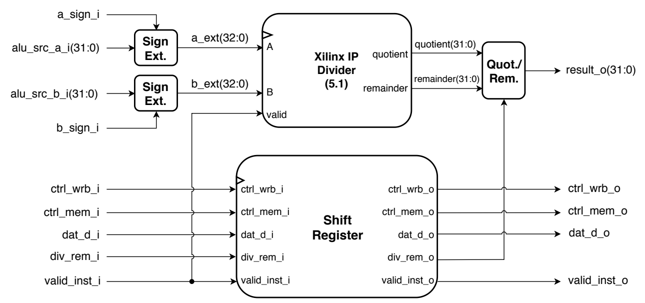

Figure 7: The Divider Unit. A shift register with the same latency is used to store the control signals during the division operations.

E. Dependencies Handling

The modifications made in the processor's pipeline, specifically the introduction of a data cache and the multi-cycle functional units, caused some new hazards that had to be solved. The need to have a control mechanism capable of identifying and handling the dependencies between instructions and the usage of multiple functional units arise. The considered unit was inspired by the Scoreboard concept, introduced in the CDC 6600 computer [10], in 1963. Due to the similarities behind the goal of both units, the name "Scoreboard" was also used in this unit.

Inside this unit, represented in Figure 8, there are 3 main blocks. The top one is a set of shift registers responsible for storing the expected latency of each register operand when an instruction inside a functional unit makes use of that same register as its destination. The verification of dependencies occurs as soon as the instruction enters the ID stage. By decoding its used registers identification, it is verified if the corresponding position on the shift register is set to '1'. If so, the scoreboard indicates that a hazard exists on its output and the processor stalls. For memory request instructions, there is a specific bit that is set to '1' when a load instruction is issued. Consequently, instructions dependent on that register are stalled. As soon as the load is finished, that bit is cleared, and instructions waiting on the ID stage are ready to be issued if no other hazards exist.

The other 2 blocks of the scoreboard (see Figure 8) are responsible for storing the position of each instruction inside the multiplier and the divider. It is necessary to provide the ID stage with information about when it should stall, in order to avoid structural hazards, which happen when one of these units has an instruction in its penultimate stage, and the instruction about to be issued uses the ALU. Since the ALU execution only takes 1 clock cycle, a collision in the EX output would occur. Another situation occurs when the instruction present on the ID stage requires the multiplier, but inside the divider there is an instruction in a position of its meta-pipeline that would finish simultaneously. The solution goes through the introduction of a 1 cycle stall instead of issuing the instruction.

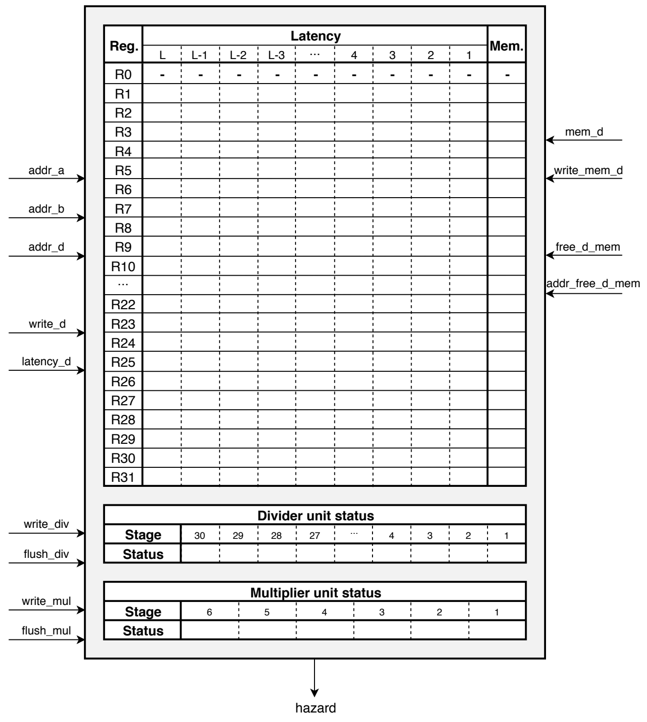

Figure 8: Representation of the scoreboard unit.

F. Peripherals

Peripherals are used to expand the processors' capabilities to interact with its external environment, according to the application's requirements. This subsection presents three memory-mapped peripherals that were developed for this processor. The first peripheral is a UART module, to be used as standard input and output, together with a set of software functions. The remaining peripherals are a cycle counter and a timer, useful to measure time on programs and evaluate the processor's performance.

1) UART for Standard Input/Output

The architecture of the considered UART module is present in Figure 9. The Address Decoder is responsible for controlling which FIFO signal is going to be read or written, according to the memory map. The TX and RX modules, available on [11], are responsible for transmitting and receiving the bits from the FPGA's lines, and the TX Controller initiates the transmission when the TX FIFO has content, by setting the TX_VALID signal to high.

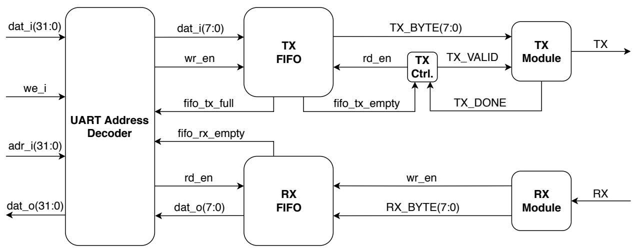

Figure 9: UART module.

2) Cycle Counter

Counters are used in processors to count events or the number of clock cycles, and they are very usefully for basic performance analysis allowing the comparison of benchmarks across different processors with different operating frequencies.

To tackle this need, it was developed a 64-bit cycle counter compatible with the same memory interface as the other peripherals. The counter features a 64-bit accumulator that is always counting when the processor is enabled, and it is reset when the processor is also reset. The choice of a 64-bit counter (instead of a 32-bit) is to reduce the chance of a overflow while the count is being considered. The selection of the output part is easily made internally using the input address, as represented in Figure 10.

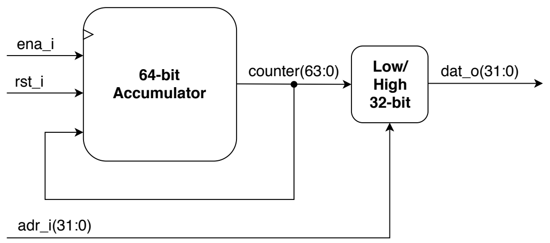

Figure 10: The cycle counter block diagram.

3) Timer

Timers are used to measure how much time a specific task needs to be executed, being a valuable tool to evaluate the processor's performance. Its hardware is the same used in the cycle counter, but it requires a dedicated clock with a well-known frequency instead of the clock used in the processor. Despite being set by the user before the synthesis and implementation, the processor's clock can suffer minor changes during the synthesis process, which will compromise the precision of the results, since the time results are calculated using the relation between the number of clocks counted by the accumulator and the frequency.

G. Data Transfer (PCI Express)

Xilinx provides an IP [12] that allows the connection between the user logic and the PCIe bus through the FPGA PCIe interface. The DMA/Bridge Subsystem for PCI Express, also referred as XDMA, can be configured as a DMA data mover or as Bridge between PCIe and AXI memory. The DMA data mover is ideal for moving blocks of data, and it can act as an AXI memory mapped interface or as an AXI streaming interface to allow direct connection to RTL logic. The PCIe bridge is used to convert PCIe packets into AXI traffic and vice versa without offering support for data streaming or a DMA. In this design, the DMA data mover configuration was used, acting as an AXI memory-mapped interface, since the main goal is the data transference between the host and the FPGA to specific addresses.

The data flow between the host and the processor initiates with the binary transference from the host memory to the XDMA on the FPGA via PCIe. Depending on the destination address, the data that is being received is sent to the DRAM (via MIG IP) or the BRAM (via AXI to BRAM IP) using the AXI4 protocol. The core uses simple interfaces (write, enable, address, data in, data out) to access data on the instruction memory or the data cache. The data cache communicates with the MIG via AXI4 when it needs to access the DRAM to read or store data.

IV. Software Tools

To complement the proposed architecture, a set of software tools is required to allow the development of programs and the interaction between the core (on the FPGA) and the host machine (personal computer). This section focuses on explaining which tools should be installed on the machine to allow the software development and the access to the PCIe bus, to transfer the binary and the data. Beyond that, several other tools were developed, including a Board Support Package, a makefile, libraries to interact with the processor's peripherals, and scripts to transfer the binaries and read or clear the processor's memories, according to the user intention.

A. Development workflow

The development workflow includes the software installed on the host machine to compile programs (RISC-V GCC compiler), the required drivers to enable the connection between the host and the XDMA through the PCIe bus, the support files created to allow the software development (Makefile and Board Support Package) and scripts to transfer data between the host and the core.

1) Software Requirements

RISC-V offers an official GNU toolchain with support for the GCC compiler, GDB, newlib, and glibc [13]. Due to some bugs on the official tools, it was decided to use as an alternative the xPack GNU RISC-V Embedded GCC [14], which is a pre-compiled modified version specially adapted to produce bare-metal applications and can be easily installed on different operating systems. According to the xPack GNU RISC-V Embedded GCC release notes [14], it is fully compatible with the official specifications.

To allow the detection and data transfers between the host and the XDMA on the FPGA, using the PCIe bus, Xilinx provides specific Windows and Linux drivers [15]. Along with the drivers, Xilinx also offers test scripts to validate the correct operation of the implemented design. Some of the test scripts were modified to allow the transference of any binary file with any arbitrary size to execute or store on the core, with the possibility to set the desired memory address.

B. Developed Libraries

1) Standard Input and Output

The set of developed functions that are herein presented have the purpose of making the development process easier when the access to the standard input and output is required. They act as a software layer between the user code and the hardware implementation of the UART module.

The user calls the printf() function as if he was using the standard printf() from the stdio.h library. The printf() function (and others not represented here) are responsible for creating the output string and sending each char to the putchar() function. The putchar() function calls the uart_putchar() function that is responsible for writing the char to the UART TX FIFO memory address, always verifying if the TX FIFO is full or not.

To read data from the UART RX line, the user should call the gets() function, passing as argument a pointer to a string and its maximum size. Then, each char stored in the RX FIFO is read (provided that the RX FIFO is not empty). The read process ends when the user writes a new line. All the content written to the UART RX is replicated in the terminal by simply writing it back to the TX line.

2) Cycle Counter and Timer

Both the Cycle Counter and the Timer hardware implementation are similar, so the corresponding functions to access both peripherals only differ on the memory addresses that are accessed. The developed functions getcounter() and printcounter(), are used to obtain the current 64-bit counter value and print it to the terminal, respectively. The code detects if the counter overflows between the reading of both parts, and if this happens, it will try again. Similar functions are used for the timer (gettimer() and printtimer()).

V. Implementation and Experimental Results

To test and evaluate the developed processor, different system configurations were implemented and prototyped in an FPGA, to be compared and evaluated in terms of resources, operating frequency, and power requirements.

Moreover, some software tests were executed to verify the correct system operation and to obtain quantitative evaluation from the execution of algorithms that can be used as benchmarks for comparing its performance with other softcores.

Based on the obtained results and taking into account all the implemented features, a discussion is presented, at the end of the section, to compare the proposed solution with the other RISC-V cores presented in Section II.

A. Prototyping framework

The developed processor was implemented on a Xilinx Virtex Ultrascale+ VCU1525 device, featuring an XCVU9P FPGA. This board is mainly targeted for computationally intensive applications due to the amount of resources offered by its FPGA and the available communication interfaces. Looking for its technical features, stand out the Gen3 x16 / Gen4 x8 PCIe interface, offering high transfer rates, four DDR4 DIMM slots, each one with 16GB memories, and finally, a UART interface via USB. Regarding the FPGA resources, the number of available LUTs, FFs, BRAMs, LUT Random-Access Memories (LUTRAMs), and Digital Signal Processing (DSP) units are listed in Table III [16].

Table III: XCVU9P FPGA resources.

LUT	1182240
FF	2364480
BRAM	2160
LUTRAM	591840
DSP	6840

During the development phase, several versions of the softcore were implemented on the FPGA, as new features were added. This process started with the PCIe XDMA implementation, followed by modifications on the architecture to add multi-cycle functional units support, hazards handling, peripherals, and cache support. The last step was the connection to the MIG, to start using the DRAM as data memory. The complete solution was implemented according to what was initially purposed and idealized. The simplified system diagram, presented in Figure 11, illustrates the major components and the connections between them.

The XDMA IP is responsible for connecting the host processor and the core's memory system, by converting the PCIe data packets received from the host into AXI requests. Inside the FPGA, the data is transferred, according to the address map, via AXI to the data memory (and placed in the external DDR4 memory through the MIG IP) or to the instruction memory, implemented with BRAMs. The data cache, the UART module, and other peripherals are connected to the core data bus through the address decoder. Hence, this address decoder is responsible for forwarding the memory requests according to the memory address. Finally, the user has access to the core's standard input and output by using the implemented UART, in particular, by connecting the host to the board's USB port, since it provides UART over USB conversion.

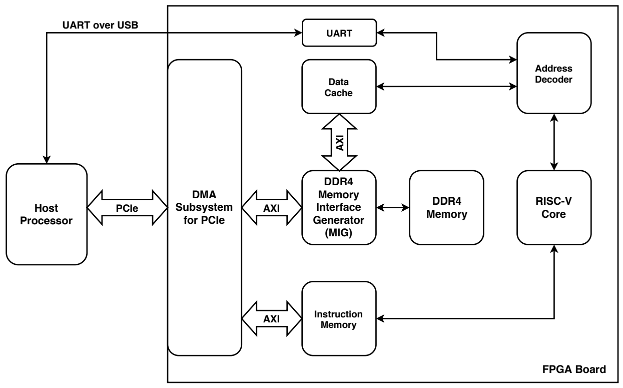

Figure 11: Simplified diagram of the proposed system with the connections between the main blocks inside the FPGA and the host processor.

To make the implementation process more manageable, the core and the data cache were packaged into custom IPs. The complete version was implemented with the help of Vivado IP Integrator design automation tool, which is responsible for the auto-configuration of some of the IPs (such as, the XDMA, and the MIG) and establishing connections between them. Those tools also add extra logic when the IPs cannot be directly connected; for instance, when they are driven by different clocks, the AXIs interfaces have different configurations, or when it is necessary to connect multiple AXI blocks to the same bus.

The following variants in the developed softcore were implemented:

- Core: this is the simplest implementation, that only includes the core with the modifications in the architecture. Different versions with and without the multiplier and the divider were also implemented to evaluate the resources used by these units.
- Core + XDMA: base version including the XDMA IP.
- Core + XDMA + Cache: base version including the XDMA IP and the data cache.
- Core + XDMA + Cache + DRAM: this version is the complete system, featuring the connection to the DRAM using the MIG IP.

1) Area and timing constraints analysis

By using the Xilinx Vivado 2019.1 tool, the proposed architecture was submitted to the implementation process to understand how the modifications impacted the resources usage and operating frequency. The small version of the implemented RISC-V core is a minimal implementation with all the changes in the pipeline, and optionally, the divider and the multiplier. However, the extra IPs, such as the XDMA, the MIG, the data cache, and the peripherals, were removed. Looking at Table IV, we can observe that the minimal implementation, as expected, uses less resources than the others and it supports a maximum operating frequency of 250MHz. The multiplier has low logic usage since it is implemented with DSPs. However,

Table IV: Resources usage and operating frequency of the different configurations of the developed processor.

Design	Resources					Freq. [MHz]
	LUT	LUT RAM	FF	BRAM	DSP	
Core	5800	2407	2346	0	0	250
Core (+MUL)	5921	2434	2439	0	4	250
Core (+DIV)	7234	2419	5230	0	0	250
Core (+MUL+DIV)	7336	2446	5307	0	4	250
Core + XDMA	28856	2367	31931	75	4	200
Core + XDMA + Cache	33137	3699	33872	59	4	100
Core + XDMA + Cache + DRAM	94343	9924	102055	110.5	7	100

the divider increases the area with significant impact in the FFs and LUTs usage because the Radix-2 algorithm exploits the FPGA logic, in opposition to the High Radix algorithm that uses DSPs.

With the addition of the XDMA, the used resources increased significantly. The XDMA is a complex IP that requires a large FPGA area, especially in terms of LUTs and Flip-flops. The frequency was reduced from 250MHz to 200MHz. An explanation for that can be the fact that the logic placement is near the PCIe pins on the FPGA, but the four clock pins are relatively far, which increases the critical path and forces higher clock periods. This implementation also includes a data memory implemented with BRAMs, which explains the use of 75 BRAMs when compared to the base core.

When the pre-developed cache integrated into a custom IP was added to the processor, the clock frequency was reduced to 100 MHz. The main reason for this reduction is the fact that the considered cache implementation is not optimized, leading to an increase in the critical path. According to information reported by Vivado, the critical path starts on the logic responsible for selecting the processor output data and ends on the Write-Back stage memory data input.

The last implementation presented in Table IV corresponds to the complete system, which includes the connection to the DRAM using the MIG IP. The operating frequency was kept in 100MHz, but the used area increased about three times, mainly due to the resources required by the MIG implementation.

B. Power analysis

Table V presents the results of the power analysis estimations that were provided by Vivado tool after the post-implementation process. The resources used and the routing, as well as other circuits, affect the power results. More resources typically mean more energy consumed.

Looking at these the power results, it can be seen that the developed core's small implementation require less power than the others, since it has the lowest area occupancy. On the other hand, the two implementations with the XDMA IP, as expected, have higher power values. The complete system, with the DRAM connection, uses three times more resources than the previous implementation, causing an increase of about three times in the power consumption.

Table V: Power results for each implementation.

Design	Dynamic Power [W]
Core	0.27
Core (+ MUL + DIV)	0.30
Core + XDMA	1.31
Core + XDMA + Cache	1.32
Core + XDMA + Cache + DRAM	3.91

C. Benchmarks results

To evaluate the processor performance and reliability, a set of benchmark tests were executed. These tests, written in C, implement simple algorithms that require some processor effort to finish the tasks. The number of clock cycles and the corresponding execution time were obtained for each test.

The set of benchmark tests includes the Tower of Hanoi, Fibonacci sequence element calculation, and vector-vector division and multiplication. The first two tests implement recursive algorithms that can be used to confirm the processor's reliability, Due to these algorithms' execution grow rates, the processor runs during millions of clock cycles without failing. The others are useful for testing data memory accesses, by enabling the verification of hardware resources, such as the data cache system operation, the main memory accesses communication, and the implemented divider and multiplier.

The advantage of using such algorithms is because they are simple, and the obtained results can be easily verified, being useful to validate the correct operation.

D. Discussion

The comparison of this solution with other softcores is not an easy task. As it was previously said, the other RISC-V cores presented in this work have several particularities. Some are not suitable for FPGAs, so they cannot be implemented and tested under the same conditions. Others, announced as suitable for FPGAs, are difficult to implement due to the lack of documentation or are attached to specific boards. The obtained results that were achieved with this processor in terms of frequency and resources usage also cannot be directly compared, because all of them were implemented in different FPGAs, leading to different implementation results.

What it is possible to compare are the functionalities and the characteristics each one offers. Looking to the analyzed RISC-V cores, we can observe that the developed architecture is the first one offering support for data transfer via PCIe, it integrates a data cache, supports the Multiplication and Division Extension, AXI connection to the external DRAM memory, several peripherals, including a UART module, being prepared to support more due to a well-defined interface.

Beyond that, due to its robust architecture inherited from the MB-Lite softcore, the proposed solution keeps its adaptability and flexibility. Most of the modifications were made thinking in future customizations, such as the addition of more functional units with variable latencies, the addition of new peripherals, or other types of cache structures.

VI. Conclusion

With this work, a new RISC-V processor was developed with the purpose of creating an alternative to the current RISC-V solutions.

The RISC-V ISA was implemented on a non-RISC-V processor, the MB-Lite, due to its solid architecture and good design. After that, the architecture started to be modified, and it was created hardware support for the RISC-V Integer Multiplication and Division Extension through the implementation of a multi-cycle multiplier and a divider, support for caches and external memories, and data transfer via PCIe.

Three peripherals were also developed (UART module, timer, and counter) to give the user more ways to interact with the processor, also providing a set of libraries to make the software integration easier.

The developed system was implemented in a Xilinx Virtex Ultrascale+ VCU1525 FPGA, and the implementation results were evaluated in terms of timing requirements, power efficiency, and area.

References

[1] A. Waterman and K. Asanovic, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20190608-Base-Ratified," RISC-V Foundation, Tech. Rep., March 2019.

[2] T. Kranenburg, "Design of a portable and customizable microprocessor for rapid system prototyping," 2009.

[3] T. Kranenburg and R. Van Leuken, "MB-LITE: A robust, light-weight soft-core implementation of the MicroBlaze architecture," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 997–1000.

[4] J. Gaisler, "Fault-tolerant microprocessors for space applications," *Gaisler Research*, pp. 41–50, 2012.

[5] Xilinx, "MicroBlaze Processor Reference Guide," Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug984-vivado-microblaze-ref.pdf, 2018.

[6] ——, "Block Memory Generator," Available: https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v8_3/pg058-blk-mem-gen.pdf, 2017, [Online].

[7] ——, "UltraScale Architecture-Based FPGAs Memory IP," Available: https://www.xilinx.com/support/documentation/ip_documentation/ultrascale_memory_ip/v1_4/pg150-ultrascale-memory-ip.pdf, 2019, [Online].

[8] ——, "Multiplier," Available: https://www.xilinx.com/support/documentation/ip_documentation/mult_gen/v12_0/pg108-mult-gen.pdf, 2015, [Online].

[9] ——, "Divider Generator," Available: https://www.xilinx.com/support/documentation/ip_documentation/div_gen/v5_1/pg151-div-gen.pdf, 2016, [Online].

[10] J. E. Thornton, "Parallel operation in the control data 6600," in *Proceedings of the October 27-29, 1964, fall joint computer conference, part II: very high speed computer systems*. ACM, 1964, pp. 33–40.

[11] nandland, "UART, Serial Port, RS-232 Interface," Available: https://www.nandland.com/vhdl/modules/module-uart-serial-port-rs232.html, [Online].

[12] Xilinx, "DMA/Bridge Subsystem for PCI Express," Available: https://www.xilinx.com/support/documentation/ip_documentation/xdma/v4_1/pg195-pcie-dma.pdf, 2019, [Online].

[13] RISC-V, "GNU toolchain for RISC-V, including GCC," Available: https://github.com/riscv/riscv-gnu-toolchain, 2019, [Online].

[14] T. xPack Project, "xPack GNU RISC-V Embedded GCC," Available: https://xpack.github.io/blog/2019/07/31/riscv-none-embed-gcc-v8-2-0-3-1-released/, 2019, [Online; accessed 10-October-2019].

[15] Xilinx, "Xilinx PCI Express DMA Drivers and Software Guide," Available: https://www.xilinx.com/support/answers/65444.html, 2019, [Online; accessed 01-October-2019].

[16] ——, "Xilinx Virtex UltraScale+ FPGA VCU1525 Acceleration Development Kit," Available: https://www.xilinx.com/products/boards-and-kits/vcu1525-a.html, 2019, [Online].