Evaluation of Softprocessor RISCV for Edge Computing Applications

Guillermo Montesdeoca [*], Víctor Asanza [†], Rebeca Estrada [*] Cristian Ramírez [‡] Jonathan Cagua [*]

[*]Escuela Superior Politécnica del Litoral, Espol (FIEC & CTI)
Guayaquil, Ecuador
{guianmon,restrada,jcagua}@espol.edu.ec
[†]SDAS Research Group, Ben Guerir 43150, Morocco
victor.asanza@sdas-group.com
[‡]SDAS Research Universitat Politècnica de València, Valencia, Spain
crirabe@posgrado.upv.es

***Abstract*—In this paper, a performance evaluation between RISCV and other processors is conducted by estimating the runtime of the Fibonacci sequence algorithm while systematically increasing the number of iterations. To ensure a fair comparison, RISCV is implemented on an FPGA Zynq-7000 SoC from Xilinx, using the same clock frequency as other processors to eliminate potential overclocking effects. Numerical results demonstrate that RISCV outperforms the ATMEGA328P AVR processor while exhibiting remarkable power efficiency. In addition, it requires fewer instructions than its counterparts for Fibonacci algorithms, Matrix Multiplication and RGB to HLB Conversion. These results position RISCV as an attractive option for Edge applications where energy conservation is essential, highlighting its superior performance in both energy efficiency and instruction efficiency.**

***Index Terms*—RISC-V; FPGA; SoC; Soft-Core Processor; VHDL**

I. Introduction

There is now a critical need to develop compact, customized, energy-efficient processors with real-time processing capabilities for multiple applications, from ubiquitous smart mobile devices to intricate radar systems and embedded computer systems in modern vehicles. These embedded systems play a key role, combining necessary hardware components such as microprocessors, memory modules and displays, tailored to specific applications. Their increasing Their increasing trend is beingdriven by the quest to optimize performance-to-size ratios, a goal that is intensifying every year [1].

Contemporary electronics design principles focus on consolidating entire systems onto a single microchip, integrating not just individual components but entire ecosystems, including microprocessors, memories, input and output peripherals, Analog-to-Digital Converters (ADCs) and specialized features tailored to specific applications [2]. This paradigm shift allows the creation of multifaceted systems in condensed forms that improve efficiency and enable miniaturization.

These embedded systems incorporate specialized features, from hardware accelerators for tasks such as artificial intelligence [3] and cryptography to specific interfaces [2]. Traditional microprocessors face challenges due to their fixed specifications when etched in silicon, making adaptability difficult in a rapidly evolving technology landscape. To address this rigidity, System on a Chip (SoC) have emerged as dynamic solutions, offering various configurations meticulously designed to meet specialized requirements in different industries and applications, reflecting a responsive approach to market demands [4].

The softcore processor approach offers a significant advantage through its higher level of abstraction, enabling quick and efficient modifications to processor architecture and functionality. This method allows for the creation of tailored Systems on a Chip (SoC) systems, precisely matched to specific application needs. Notable examples like Xilinx's MicroBlaze [5] and ARM Cortex-M1 demonstrate the approach's versatility [6], as they can be configured within dedicated development environments, allowing designers to customize hardware attributes. These processors can then be seamlessly integrated into Field-Programmable Gate Arrays (FPGAs), resulting in highly adaptable solutions. What sets this approach apart is its transformative impact on Intellectual Property (IP) block configuration, granting designers the ability to dynamically shape hardware-level characteristics of chips. This newfound flexibility revolutionizes processor fine-tuning, allowing for specific adjustments in performance, power, or features.

Softcore processors, configured at the Hardware Description Language (HDL) level, offer improved versatility and adaptability, allowing designers to tailor them to specific application requirements, making them ideal for scenarios requiring rapid prototyping or frequent iterations. These processors operate based on a predefined set of instructions, covering vital functions such as memory operations and arithmetic. Often, designers use commercially available Intellectual Property (IP) blocks from companies such as ARM as fundamental building blocks, providing a reliable starting point for development. However, it is crucial to consider the potential royalty payments associated with the use of proprietary IP cores.

Nevertheless, the advent of RISC-V signals a departure from this model, representing a free and open Industrial Standard Architecture (ISA) with no royalties or non-disclosure agreements. The transparency of RISC-V democratizes processor design, enabling a broader community of developers and companies to innovate collaboratively. This open access fosters innovation and accelerates advances in processor technology, removing financial barriers and cultivating a dynamic environment for progress and collaboration.

RISC-V International is a non-profit organization that oversees and promotes the RISC-V open instruction set architecture[1]. This organization is responsible of the development, promotion and ongoing standardization of the RISC-V architecture.

The performance offered by these processors is intrinsically related to the complexities of the systems in which they are embedded, whether it is the low-power design of the ARM Cortex-M0+ (Raspberry Pi Pico) or the role of the ARM Cortex-A9 (Zynq-7000) within sophisticated mobile devices. Therefore, the comparative analysis presented in [7] aims to unravel the subtle strengths and weaknesses of these processors within their specific contexts, shedding light on their performance, energy efficiency and adaptability.

Since we have introduced a design of the first Ecuadorian open-source software softprocessor called RISCV-EC in a prior work [8], this paper aims to evaluate our proposed RISC-V by a comprehensive comparison with other established processors such as the 8-bit AVR microcontroller (Microchip ATmega 328p), ARM Cortex-M0+ (Raspberry Pi Pico) and ARM Cortex-A9 (Zynq-7000). These processors, each with diverse capabilities and design philosophies, provide a robust framework for exploration. The comparison is carried out using three algorithms, namely Fibonacci, Matrix Multiplication and RGB to HLB Conversion algorithms.

II. Related Work

In the field of processor design and implementation, a large body of academic work has been focused on analyzing the differences between hard processors and soft processors, by establishing the differences and trade-offs that define the landscape of processor architectures, drawing insights from various fields of computer science and engineering. For instance, [9, 10] present a glimpse into this extensive body of work, highlighting the adaptability of soft processors and acknowledging the efficient performance of hard processors.

Several studies in the field of microprocessor selection have addressed the task of choosing the most suitable microprocessor based on specific requirements and processor characteristics. Within this area of research, academics have devised methodologies and frameworks to guide decision making process. For example, [1] provides valuable insights into the microprocessor selection process, especially in aligning the technical requirements of the project with appropriate microprocessor attributes.

[1]https://riscv.org)

Researchers have embraced the use of FPGAs as a versatile platform for implementing digital systems, moving away from the limitations of traditional microprocessors. This paradigm shift introduces a wealth of possibilities, including the creation of custom architectures tailored to specific applications. Key advantages of this FPGA-centric approach include the ability to design custom hardware accelerators for optimized computational performance, leverage parallel processing for tasks requiring real-time data handling and complex computations such as vector processors [11]. Other work has focused on exercising detailed control over low-level hardware design, accelerating rapid prototyping and iterative development, achieving energy efficiency in power-sensitive applications, real-time processing, low latency and high throughput are paramount, as explained in references [12] and [13].

Selecting a suitable processor goes beyond raw computational power, considering factors such as energy efficiency, memory capacity, input/output interfaces, scalability, security features and long-term support. Academics such as Jim Ledin and Dave Farley delve into the intricacies of designing Soft-Core Processors, detailed in reference [14], offering a comprehensive guide to their design. Their work explores the complexities of processors, the differences in instruction set architectures (including RISC-V), and the operating principles of devices such as smartphones that rely on ARM-based processors.

Other studies focused on the comparison between hard processors and soft-core processors, remarking their advantages, tradeoffs, and practical implications on performance such as [2], where the authors evaluate the performance of a digital control application on an FPGA using both hard and soft-core processors, providing valuable insights into the real-world implications of choosing between these approaches. In addition, the reference [4] performs a comprehensive comparative analysis of Intellectual Properties (IPs) of commercially available processors, including the influential Xilinx Microblaze, serving in decision making when selecting processor IPs for specific projects or applications.

In particular, [15] presents a comprehensive evaluation of a commercial RISC-V core, assessing processor reliability and error resilience in practical applications. Their research analyzed errors, measuring their frequencies during fundamental operations both with and without an operating system. This research has significant relevance, where processors are fundamental to applications ranging from critical systems to consumer electronic devices, contributing to the broader debate on processor quality, performance and robustness.

Gray Research LLC introduced an innovative approach focused on designing a RISC-V core with optimal component efficiency, aiming to maximize the number of cores accommodated within a FPGA framework [16]. This strategy aimed to harness parallelism's power by densely populating FPGAs with numerous cores, enhancing computational throughput and system performance, especially in applications with real-time data processing and high-performance computing requirements. This initiative reflects a broader trend in leveraging

FPGA technology for diverse applications, highlighting RISC-V's adaptability and potential to address hardware design challenges and unlock the full potential of FPGA-based parallelism.

In our prior work [8], we proposed a design of the first Ecuadorian open-source software softprocessor called RISCV-EC, which is based on a RISC-V single core architecture and compared to other processors such as AVR ATMEGA328P, ARM Cortex M1 and ARM Cortex A9 Zynq-7000 . The results showed that the RISCV-EC softprocessor has a better performance than the ATMEGA328P AVR processor for any given number of iterations.

LowRISC[2] is a non-profit organization considered a key player in collaborative efforts aimed at accelerating RISC-V technology, forging partnerships with universities, research centers and industry giants such as Google. Among its contributions to the RISC-V ecosystem is the development of a RISC-V soft core known as IBEX [17].

III. Microarchitecture

Here we present the architecture of the proposed RISC-V softprocessor, emphasizing the Medium Scale Integration (MSI) blocks that constitute its core. Each part of a microcontroller is detailed, encompassing design considerations and practical implementation within a Xilinx FPGA environment.This comprehensive narrative aims to provide readers with an in-depth comprehension of the softprocessor's inner workings, from architectural components to real-world implementation, facilitating a holistic grasp of its design and practical deployment.

A. Softprocessor Design

The architectural implementation was carried out using VHDL and is visually represented in Figure 1. All the source code for individual blocks and the complete architecture has been made publicly available on the OpenCores platform [3]. OpenCores stands as a preeminent online community dedicated to the development of gateway Intellectual Properties (IP Cores). The RISCV softprocessor aligns with the architectural principles of a RISC-V processor, as meticulously delineated in ISO documentation.

- **Arithmetic Logic Unit (ALU):** This integral component assumes the pivotal role of executing arithmetic operations on integers and facilitating comparisons. Its operation is governed by two 32-bit inputs, one featuring a 4-bit Operation Code (Opcode) designated as a communication flag for interactions between the control unit and the ALU (refer to Table I). Furthermore, the ALU generates a 32-bit output to convey the computed results. Notably, in this context, there's no need for a carry mechanism, as the numbers are subjected to addition as signed integers, ensuring that the sums remain within the confines of the 32-bit range. In VHDL, a case statement is employed to dynamically switch between operations based on the Opcode, allowing the ALU to adapt its functionality accordingly.
- **Register Bank:** The register bank serves as a 32-bit by 32-bit memory unit, exhibiting a multifaceted functionality. It encompasses three distinct 5-bit read inputs, a 32-bit data input, a pivotal Write/Read enable signal (WE) that dictates data transfer, a clock input to synchronize operations, and two 32-bit outputs. The memory's operation is noteworthy for its asynchronous reading, meaning it responds swiftly to read requests, while writing operations are synchronous, ensuring data integrity and coherence. This synchronization is achieved through the inference of the memory block's behavior, allowing the synthesis tool to seamlessly orchestrate the required operations. In essence, the register bank primarily comprises flip-flops, underpinning its efficient and reliable performance.
- **Instruction Memory:** The register bank functions as a versatile memory unit with a size of 32 bits by 32 bits. It includes three separate 5-bit read inputs, a 32-bit data input, a crucial Write/Read enable signal (WE) controlling data transfer, a clock input for synchronization, and two 32-bit outputs. The memory operates asynchronously during reads, promptly responding to requests, and synchronously during writes, guaranteeing data integrity and coherence. This synchronization is achieved through the inference of the memory block's behavior, allowing the synthesis tool to seamlessly orchestrate the required operations. In essence, the register bank primarily comprises flip-flops, underpinning its efficient and reliable performance.
- **Program Counter (PC):** This integral MSI block functions as a register responsible for storing the current program line's value, denoting the instruction being executed by the processor. Its operation is synchronized with the clock signal, incrementing by 4 during each clock cycle. This incrementation is achieved through a well-coordinated interplay between an adder and a multiplexer, ensuring precise program flow control and efficient execution of instructions.
- **Instruction Decoder:** Within this critical section, the 32-

TABLE I: Instructions that can be executed by the instruction set processor RISCV

Operations between registers	Operations with immediate	Flow clontrol operations	Memory operations
add	addi	beq	lb
sub	slti	bne	lh
sll	sltiu	blt	lw
slt	xori	bge	lb
sltu	ori	bltu	lhu
xor	andi	bgeu	sb
srl	sli		sh
sra	srli		sw
and	sli		sh

[2]https://lowrisc.org

[3]https://github.com/RISCV-EC/RISCV-EC

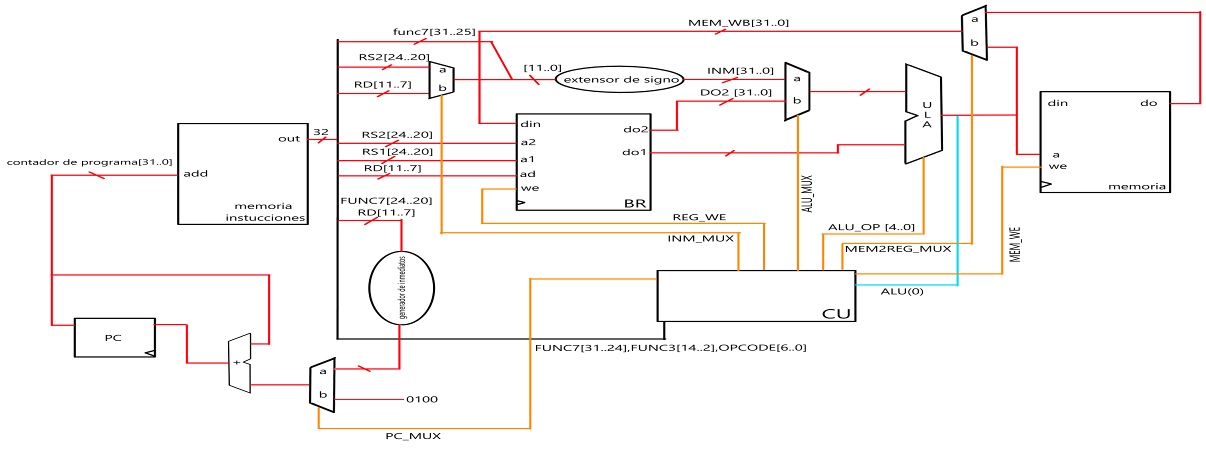
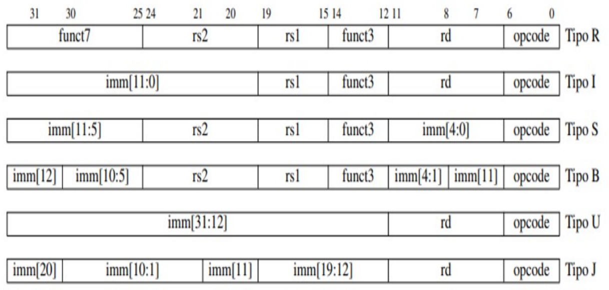


Fig. 1: *Proposed architecture*

bit instruction is dissected into distinct sets that hold essential relevance for the processor's operation. The division adheres to the format established by R-type instructions. Specifically, the first 7 bits are dedicated to Func7, followed by designated bits for rs1 and rs2, each assigned their respective bit fields. An allocation of 3 bits is made for Func3, while RD claims 5 bits for its designation. Lastly, the opcode is allocated 7 bits. This systematic segmentation, depicted in Figure **??**, facilitates the precise extraction of crucial information from the instruction, streamlining the processor's execution of diverse commands.



31	30 25	24 21 20	19 15	14 12	11 8	7	6 0	
funct7		rs2	rs1	funct3	rd		opcode	Tipo R
imm[11:0]			rs1	funct3	rd		opcode	Tipo I
imm[11:5]		rs2	rs1	funct3	imm[4:0]		opcode	Tipo S
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	Tipo B
imm[31:12]					rd		opcode	Tipo U
imm[20]	imm[10:1]	imm[11]	imm[19:12]		rd		opcode	Tipo J

Fig. 2: Instruction Format

- **Multiplexors:** The Multiplexer, a fundamental MSI block, adeptly alters its outputs by strategically selecting from among its inputs. In the context of our architecture, a total of four multiplexers collaboratively contribute to the system's operation. The selection of inputs for each of these multiplexers is meticulously governed by the control unit, which generates and dispatches the requisite control signals. This orchestrated utilization of multiplexers plays a pivotal role in facilitating dynamic data routing and decision-making within the processor's operations, enhancing its overall efficiency and adaptability.
- **Sign extender:** Within our MSI architecture, the Sign Extender assumes a crucial role, particularly in processing immediate instructions characterized by a 12-bit integer value encoded within the instruction. To seamlessly integrate this value with the ALU, which operates on 32-bit inputs, the 12 bits must undergo an extension to match the ALU's data width. Remarkably, this extension is executed without any alteration to the magnitude or sign of the original value, ensuring that precision and consistency are upheld throughout the computational processes.
- **Control Unit:** The pivotal Control Unit, a critical MSI block within our architecture, operates as a Moore Finite State Machine (FSM) responsible for the generation of intricate control signals, thereby orchestrating the execution of instructions by the processor. It takes as input the opcode, func3, and func7 fields from the RISC-V instruction format, specifically for R-type instructions. Through a systematic evaluation facilitated by a case statement, the Control Unit categorizes instructions into their respective types, including R, I, S, B, U, or J. Following this classification, an additional nested case statement is employed to pinpoint the specific instruction, leveraging the func3 field. In instances where Opcode and func3 overlap, such as in addition and subtraction instructions, an if-else or when-else statement effectively resolves the selection between the two, ensuring precise instruction execution within the processor's operations. This comprehensive control mechanism is instrumental in maintaining order and accuracy throughout the execution of diverse instructions.

B. Instructions Type

The instructions in our architecture are categorized into five distinct types: Type-R, Type-I, Type-S, Type-B, and Type-J. These types correspond to Record, Immediate, Save, Branch, and Jump instructions, each featuring its unique format. The RISCV Softprocessor possesses the inherent capability to differentiate between these instruction types and execute them sequentially. In the ensuing sections, we elucidate the inner workings of each instruction type within the processor, commencing with R-type instructions. This classification and differentiation of instruction types play a fundamental role in facilitating the orderly execution of a diverse range of commands within the processor's operations.

- **Type R Instructions:** In the RISCV Softprocessor architecture, as depicted in Fig. 1, the processing of Type R instructions initiates with the instructions being fetched from the instruction memory and subsequently routed to a single instruction decoder. Within this decoder, the opcode, func7, and func3 fields are extracted and dispatched to the control unit. This control unit plays a pivotal role in generating activation signals that orchestrate the behavior of various elements, including the ALU, contingent on the instruction type, as delineated in Table 1. Notably, $INMMUX$ serves as the signal governing the multiplexer responsible for selecting the relevant field for immediate generation. In the context of Type R instructions, this signal effectively becomes a "don't care" since immediates are not employed in these instructions.
  The $REG-WE$ signal assumes a value of 1, signifying that the output of the ALU must be stored within a register. Conversely, both $PC-MUX$ and $MEN-WE$ remain at 0 since Type R instructions do not pertain to memory operations or branch instructions. Finally, $ALU-MUX$ and $MEN2REG-MUX$ are set to 1, indicating the necessity to select the appropriate portion of the register bank for ALU operations, with the output of the ALU being directed to the din port of the register bank. This comprehensive configuration ensures precise and coordinated execution of Type R instructions within the processor's operations.
- **Type I Instructions:** In the context of Type I instructions, the activation signals closely resemble those of the previous instruction type. The ALU receives the corresponding opcode for the operation from the Control Unit (CU). Notably, $REG-WE$ and $MEN2REG-MUX$ both assume a value of 1. The former signifies that the results of the ALU operation should be written to the register block, while the latter is responsible for directing the ALU's output to the data input of the register bank.
  Conversely, the remaining control signals are set to 0. In this specific instruction type, the bits spanning from 31 to 20 within the instruction are consolidated and extended using the sign extender block, ultimately arriving at one of the ALU ports. The resultant output from the ALU operation is then stored within the register bank, completing the execution of Type I instructions in a coordinated and precise manner.
- **Type S Instructions:** In the case of Type S instructions, specific control signals come into play, namely $INM-MUX$ and $MEM-WE$, while the remainder remain inactive. The process begins with the creation of an immediate value by concatenating bits 31 to 25 and 11 to 7. This 11-bit immediate is then extended while preserving its sign through the sign extender block.
  Subsequently, this extended immediate value is added to the contents of the register indicated by the address in RS1. The result of this addition becomes the memory address where the data will be stored. The data itself is sourced from the RS2 register, but it is important to note that it is not subjected to ALU operations because the $ALU-MUX$ signal remains deactivated in this scenario. This streamlined execution path ensures the efficient handling of Type S instructions within the RISCV softprocessor architecture.
- **Type L Instructions:** For Type L instructions, the $REG-WE$ signal is activated, while the other control signals remain deactivated. In this context, the values stored in RS1 are subjected to addition with an immediate value within the ALU. The outcome of this operation serves as the memory address from which data will be read. This data is then directly transmitted to the Data In (din) port of the register block, ready to be stored in the address specified by RD. This straightforward process ensures efficient execution of Type L instructions within the RISCV softprocessor architecture.
- **Type B Instructions:** For Type B instructions, the operation of the ALU depends on the specific instruction being executed, as determined by the operations outlined in Table 1. Additionally, the control signals $PC-MUX$ and $ALU-MUX$ are set to 1, while the rest remain at 0. The functioning of Type B instructions involves a comparison between an immediate value and the content of RS2. The outcome of this comparison serves as an input to the Control Unit (CU), signaling the initiation of a branch operation. Consequently, the CU activates the $PC-MUX$, which in turn alters the input of an adder responsible for modifying the program counter. The degree of modification to the program counter is decoded based on the values in the funct and rd fields of the instruction. This decoding process is facilitated by a specialized block that rearranges the bits, generating the necessary immediate value for the branch instruction. This mechanism ensures the correct execution of branching operations within the RISCV softprocessor architecture.

The Register Bank plays a crucial role in the RISCV softprocessor architecture, receiving signals from RS_2, RS_1, and RD as specified in the instruction encoding. RS_2 and RS_1 are directly connected to the read ports of the register bank, and the data stored in those addresses are immediately accessible through Do_2 and Do_1. Meanwhile, RD is directed to the ad port, which serves as the write address. The Write Enable (WE) signal coordinates with the clock to synchronize the storage of data arriving through d_{in}. The register bank's outputs are then fed into the ALU, with do_1 going directly to port 2, and do_1 connected to a multiplexer. This multiplexer is responsible for allowing the selection between using immediate values or performing operations between registers, ensuring flexibility and adaptability in executing various instructions within the processor.

IV. Numerical Results

For the evaluation of the architectures, we used three algorithms, which are available at the following link https://github.com/RISCV-EC/RISCV-EC/tree/main/Algorithms. The description of each of the three algorithms is is presented below.

Fibonacci Series: The program calculates and stores the Fibonacci sequence in memory, starting from 1 and 2, until the value reaches or exceeds 100. The code uses specific registers and arithmetic and bit manipulation operations to perform the necessary operations. The program begins by initializing a series of registers and setting the stack pointer before entering a loop where the Fibonacci sequence is calculated and stored. Once the limit of 100 is reached, the program finishes and restores the stack pointer to the original state. The Fibonacci sequence is stored in the program memory for later use. We employed Algorithm 1 to assess and compare the performance of the various development boards utilized in this study. Additionally, a significant parameter under consideration was the utilization of logic elements during the synthesis of the VHDL code on the FPGA.

Algorithm 1: Fibonacci series

```
Output: Array of integers $fib$ of size 100 containing
        the first 100 Fibonacci numbers
Data: Integers: $fib[100]$, $i$
for $i \leftarrow 2$ to $99$ do
    $fib[i] \leftarrow fib[i-1] + fib[i-2]$
end
return $fib$
```

Matrix Multiplication: The program implements the multiplication of two matrices. A function is defined *matrix_multiply*, which takes two matrices stored in memory, performs the necessary multiplication and accumulation operations, stores the result in the first matrixand and uses specific registers and arithmetic operations to manipulate the data in the matrices. The main code *main* initializes the matrices, calls the function *matrix_multiply* and stores the result, restoring the state of the stack pointer before exiting. The code also includes data sections that store the values of the arrays and details about the size of the stack used by the functions, thus showing their structure and functionality. We employed Algorithm 2 to assess and compare the performance of the various development boards utilized in this study. Additionally, a significant parameter under consideration was the utilization of logic elements during the synthesis of the VHDL code on the FPGA.

Algorithm 2: Matrix Multiplication

```
Input: Matrix $A$ and $B$
Output: Matrix $result$
for $i \leftarrow 0$ to $2$ do
    for $j \leftarrow 0$ to $2$ do
        $result[i][j] \leftarrow 0$
        for $k \leftarrow 0$ to $2$ do
            $result[i][j] \leftarrow result[i][j] + A[i][k] \times B[k][j]$
        end
    end
end
return $result$
```

RGB to HLB Converter: The program performs the conversion of a color from Red Green and Blue (RGB) format to Hue Saturation Lightness (HSL). It utilizes two structures, *RGBColor* and *HSLColor*, to represent colors in RGB and HSL formats, respectively. The *RGBtoHSL* function takes a color in RGB format and calculates its corresponding values in HSL format. It performs the necessary mathematical calculations to determine the hue, saturation, and lightness of the provided color. In the main program (*main*), a pure red color (255 in red, 0 in green, and blue) is created, converted to HSL format using the *RGBtoHSL* function, and the resulting values are stored in an *HSLColor* structure. This code demonstrates a basic example of color conversion, which can be useful in graphical applications and image processing. We employed Algorithm 3 to assess and compare the performance of the various development boards utilized in this study. Additionally, a significant parameter under consideration was the utilization of logic elements during the synthesis of the VHDL code on the FPGA.

A. Features of the RISCV processor

The main features of the RISCV processor are listed below:

- **RISC-V Architecture:** The processor aligns with RISC-V architectural principles, following a Reduced Instruction Set Computing approach for efficient instruction execution.
- **32 Registers:** The processor includes a set of 32 registers, with the initial register (X0) always set to zero due to its physical grounding connection.
- **Arithmetic Logic Unit (ALU):** The ALU performs arithmetic operations on integers and facilitates comparisons. It operates on two 32-bit inputs using a 4-bit Operation

Algorithm 3: RGB to HLB Converter

```
Input: Color RGB: $rgb = (r, g, b)$
Output: Color HSL: $hsl = (h, s, l)$
$[0, 1] \leftarrow \texttt{Normalize}(r, g, b)$
$M \leftarrow \max(r, g, b)$
$m \leftarrow \min(r, g, b)$
$l \leftarrow (M + m)/2$
if $M = m$ then
    $s \leftarrow 0$
    $h \leftarrow 0$                          // Undefined, but set to 0
else
    $d \leftarrow M - m$
    if $l > 0.5$ then
        $s \leftarrow d/(2 - M - m)$
    else
        $s \leftarrow d/(M + m)$
    end
    if $M = r$ then
        $h \leftarrow (g - b)/d + (g < b?6 : 0)$
    else
        if $M = g$ then
            $h \leftarrow (b - r)/d + 2$
        else
            $h \leftarrow (r - g)/d + 4$
        end
    end
    $h \leftarrow h/6$
end
return $hsl = (h, s, l)$
```

Code (Opcode) and produces a 32-bit output. The ALU handles addition of signed integers within the 32-bit range without a carry mechanism.

- **Register Bank:** This 32-bit by 32-bit memory unit includes read inputs, a data input, Write/Read enable signal (WE), and clock input. It supports asynchronous reading and synchronous writing, ensuring data integrity. Flip-flops are utilized for efficient performance.
- **Instruction Memory:** Similar to the register bank, this 32-bit by 32-bit memory unit supports asynchronous reading and synchronous writing. It plays a crucial role in storing instructions for the processor to execute.
- **Program Counter (PC):** This integral component stores the current program line's value, indicating the instruction being executed. It increments by 4 during each clock cycle, ensuring precise program flow control.
- **Instruction Decoder:** The 32-bit instruction is dissected into distinct sets, following the format of R-type instructions. It involves decoding Func7, rs1, rs2, Func3, RD, and opcode fields to extract essential information, streamlining diverse command execution.
- **Multiplexers:** Four multiplexers facilitate dynamic data routing and decision-making within the processor's operations. Inputs are selected based on control signals generated by the Control Unit, enhancing efficiency and adaptability.
- **Sign Extender:** This component extends 12-bit integer values from immediate instructions to match the ALU's 32-bit data width. It ensures precision and consistency in computational processes.
- **Control Unit:** Operating as a Moore Finite State Machine (FSM), the Control Unit generates intricate control signals based on opcode, func3, and func7 fields. It categorizes instructions into types (R, I, S, B, U) and further refines instruction selection using nested case statements and conditional statements, ensuring accurate instruction execution.

TABLE II: FPGA resources Usage for synthesizing

Resources	FPGA capacity	Usage
Slice LUTs	53200	3.69%
Slice Registers	106400	2.36%
F7 Muxes	26600	0.77%
F8 Muxes	13300	0.24%
Slice	13300	7.03%
LUT as Logic	53200	3.20%
LUT as Memory	17400	1.48%
Block RAM Tile	140	1.07%
Bonded IOB	125	0.80%
BUFGCTRL	32	9.38%
BSCANE2	4	25.00%

B. Resource Usage

The use of programmable logic elements within the FPGA following the synthesis of the RISCV design, as outlined in Table II, demonstrates significant improvement. These findings indicate that the consumption of Logic Elements (LE) is notably minimal. Consequently, the envisioned processor can be readily employed in alternative FPGAs with more modest logic element resources available.

C. Power consumption

Finally, we present a comparison of processor power consumption while running the Fibonacci Series algorithm. For this comparison, each processor worked with its respective clock frequency. The clock frequencies per processor are as follows:

- RISCV operates at 50MHz.
- 8-bit AVR Microcontroller (Microchip ATmega 328p) runs at 16MHz.
- ARM Cortex-M0+ (Raspberry Pi Pico) operates at 125MHz.
- ARM Cortex-A9 (Zynq-7000) runs at 650MHz.

Figure 3 shows the results of static, dynamic and total power consumption. These values were obtained experimentally for each of the processors on their respective development boards: RISCV on the Zynq-7000, the 8bit AVR Microcontroller (Microchip ATmega 328p) on the Arduino UNO, the ARM Cortex-M0+ on the Raspberry Pi Pico and ARM Cortex-A9 on the Zynq-7000. The definition of each of these consumption metrics is as follows:

- **Static Power:** It is the power consumption in Watts (W) by the development board with the processor in IDLE mode and it was easured experimentally,.
- **Dynamic Power:** It is the power in Watts (W) solely from the processor executing the Fibonacci Series algorithm [18]. It was calculated by subtracting static power from total power.

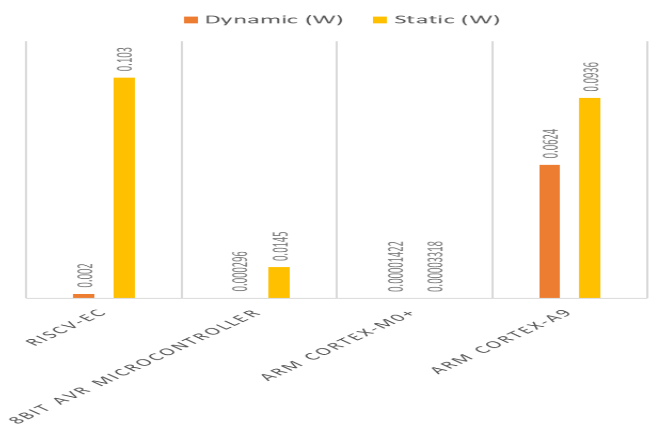


Fig. 3: Power consumption Comparison

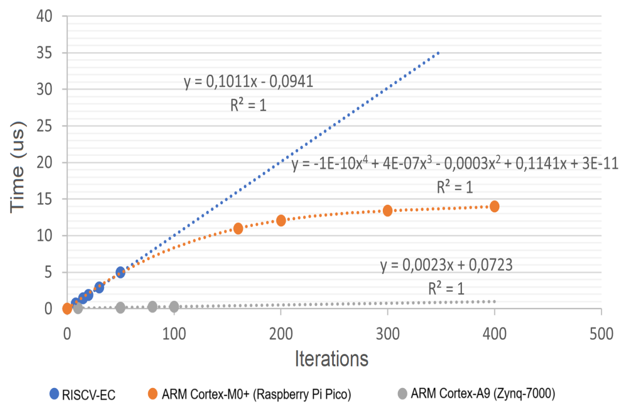


Fig. 4: *Running Time Comparison among RISCV, ARM Cortex-M0+ (Raspberry Pi Pico) and ARM Cortex-A9 (Zynq-7000) processors*

Numerical results demonstrates that the proposed RISCV processor consumes 2mW when executing the Fibonacci Series algorithm while the single-core ARM Cortex-M0+ processor (Raspberry Pi Pico) consumes 14.22uW, and the 8-bit AVR Microcontroller (Microchip ATmega 328p) consumes 0.296mW. In particular, ARM Cortex-A9 processor (Zynq-7000) uses 62.4mW due to its higher performance capabilities and dual-core configuration integrated with 28nm Artix technology.

D. *Performance Comparison*

In Section IV, the performance evaluation of each processor was conducted using Algorithm 1, considering their respective implementations and disparities in the instruction set. Figure 4 illustrates a comparative analysis among the proposed processor, the ARM Cortex-M0+ (Raspberry Pi Pico), and the ARM Cortex-A9 (Zynq-7000) processor, shedding light on their relative performance characteristics.

All processors were programmed in assembly language, utilizing registers for mathematical operations rather than memory. The observed performance order, from slowest to fastest, was as follows: RISCV, ARM Cortex-M0+ (Raspberry Pi Pico), and ARM Cortex-A9 (Zynq-7000). This performance discrepancy can be attributed to the distinct clock frequencies of these processors (specifically, RISCV, ARM Cortex-M0+, and ARM Cortex-A9 operate at 50MHz, 125MHz, and 650MHz, respectively). The absence of a pipeline in the RISCV hampers its maximum achievable frequency. The Raspberry Pi Pico, powered by the ARM Cortex-M0+ core, is designed for efficiency and employs a short 2-stage pipeline, resulting in a lower maximum clock frequency. In contrast, the PYNQ-Z1's SOC boasts two ARM Cortex-A9 cores, created for a 45nm manufacturing process and featuring an 8-stage pipeline. While it holds the highest frequency among the processors in this comparison, it remains relatively low by contemporary standards. In summary, a higher clock frequency correlates with increased performance, with pipeline stages playing a pivotal role in boosting a processor's frequency. It's essential to note that while this experiment allows us to evaluate each processor's implementation, it doesn't provide a comprehensive assessment of the differences in execution time attributable to variations in instruction sets, primarily due to the substantial differences in clock frequencies.

To facilitate a comparison between the proposed processor and one with a similar clock frequency, we employed 8bit AVR Microcontroller (Microchip ATmega 328p) on the Arduino Uno development board, which operates at a lower frequency of 16 MHz and features a 1-stage pipeline. The outcomes of this comparative analysis are depicted in Fig. 5. It's noteworthy to acknowledge that there exists a disparity in power consumption between these two processors; however, it's crucial to emphasize that this assessment provides only an approximate estimate, precluding us from drawing concrete conclusions in the realm of power consumption.

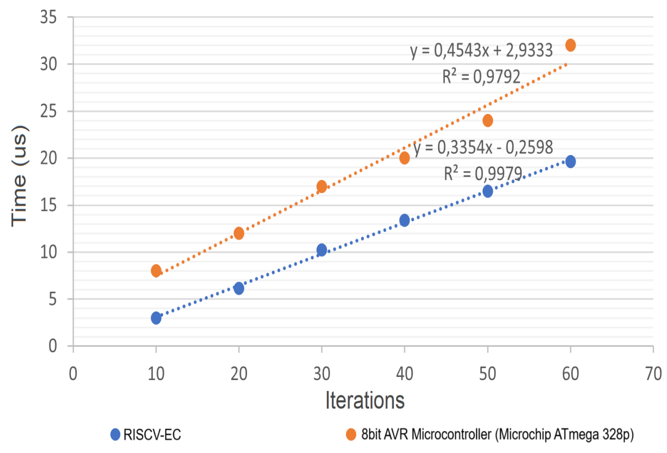


Fig. 5: *Performance comparison RISCV (16MHz) vs 8bit AVR Microcontroller - Microchip ATmega 328p (16MHz) with lower clock frequency.*

As anticipated, the RISCV exhibits significantly better performance compared to the Arduino. This performance gap can be attributed to several factors. Firstly, the RISCV's instruction set allows for more efficient code execution. Secondly, the RISCV benefits from a 32-bit bus, which enables faster data transfer and processing, whereas the Arduino operates with an 8-bit bus, limiting its computational capabilities. These architectural distinctions underscore the advantages of the RISCV softprocessor in terms of speed and efficiency.

In Figure 6, we observe the outcomes of a comparative analysis between the RISCV processor and the ARM Cortex-M0+ processor found in the Raspberry Pi Pico. To ensure a fair comparison, the operating frequency of the RISCV was adjusted to match the 125MHz frequency of the ARM Cortex-M0+. Notably, the RISCV outperforms the ARM Cortex-M0+ when the number of interactions exceeds 300. Conversely, for interaction counts lower than 300, the ARM Cortex-M0+ processor on the Raspberry Pi Pico exhibits superior performance when compared to the 32-bit RISCV processor. These findings underscore the nuanced performance dynamics influenced by factors such as operating frequency and computational workload.

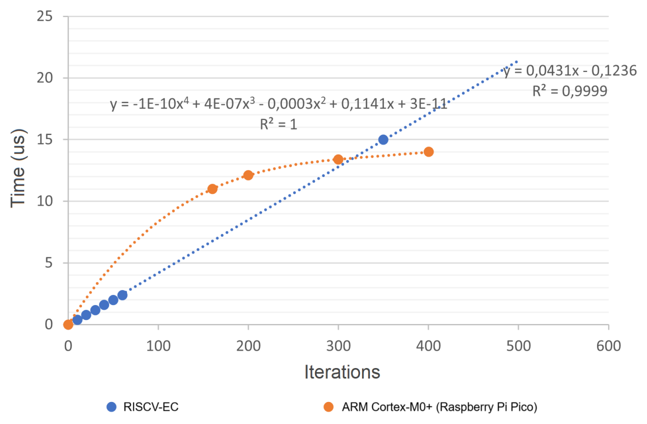


Fig. 6: *Comparison RISCV vs ARM Cortex-M0+ (Raspberry Pi Pico) with similar operating frequency (125MHz).*

For a comprehensive and equitable comparison between the processors, we conducted simulations that enabled us to configure the RISCV with a higher operating frequency. This simulation-based approach was necessary as the physical limitations of the experimental RISCV processor precluded it from achieving higher clock frequencies. Figure 7 illustrates the comparative results between the RISCV and the ARM Cortex-A9 (Zynq-7000), both operating at identical clock frequencies (650MHz). Notably, even with matched clock frequencies, the ARM Cortex-A9 in the Zynq 7000 outperforms the RISCV, reaffirming the supremacy of the ARM Cortex-A9 in terms of raw processing power. This analysis provides valuable insights into the influence of architectural differences on processor performance under similar operating conditions.

Another comparison is based on the number of instructions required by each processor to execute the algorithms: Fibonacci Series, Matrix Multiplication and RGB to HLB Converter, shown in Figure 8. In this comparison, we take advantage of the architecture of each processor to implement

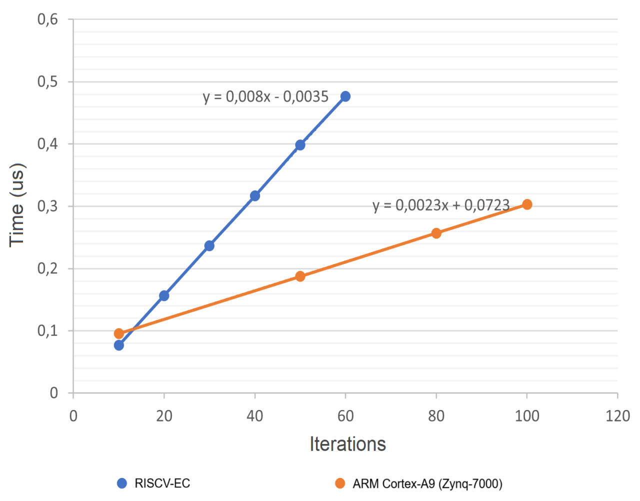


Fig. 7: *Simulated Performance of RISCV vs ARM Cortex-A9 (Zynq-7000) with same clock frequency (650MHz).*

in assembly language the proposed algorithms. That is to say, those processors with a better architecture will need a lower number of instructions to execute the same algorithm.

On the one hand, the results show that the ARM Cortex-A9 architecture (Zynq-7000) requires fewer instructions to execute the three algorithms. On the other hand, the RISCV processor is the second processor that requires fewer instructions to execute the Fibonacci Series algorithm. When running the Matrix Multiplication algorithm, the second best processor is the ARM Cortex-M0+ (Raspberry Pi Pico). Finally, when running the RGB to HLB Converter algorithm, the second best processor requiring the least number of instructions is the ARM Cortex-M0+ (Raspberry Pi Pico).

Comparing the RISCV processor, we can notice that it is a robust device that when executing the Serial Fibonacci algorithm, requires fewer instructions than the 8bit AVR Microcontroller (Microchip ATmega 328p) and ARM Cortex-M0+ (Raspberry Pi Pico) processors. Likewise, the RISCV processor, when executing the Matrix Multiplication algorithm, requires fewer instructions than the 8bit AVR Microcontroller processor (Microchip ATmega 328p). Finally, the proposed RISCV processor, when executing the RGB to HLB Converter algorithm, requires less number of instructions than the 8bit AVR Microcontroller (Microchip ATmega 328p). That is, the proposed RISCV processor requires fewer instructions when executing serial-based algorithms than the ARM Cortex-M0+ and 8bit AVR Microcontroller processors. But when executing algorithms requiring matrix operations it requires fewer instructions than the 8bit AVR Microcontroller.

V. Conclusions

This paper evaluates a novel design of the first Ecuadorian open-source softprocessor, based on the RISC-V single-core architecture. In addition, we conducted a thorough performance comparison with various well-known processors, such as AVR ATMEGA328P microcontroller, ARM Cortex M0,

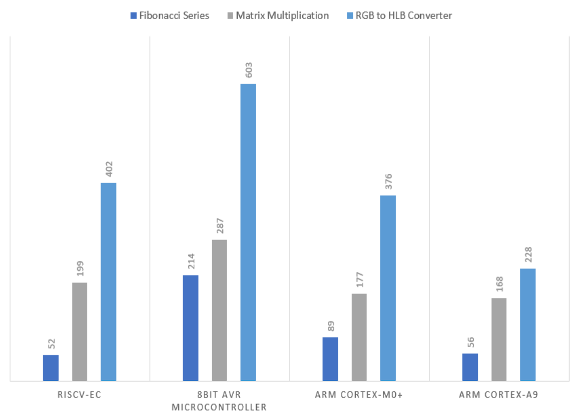


Fig. 8: *Number of instructions per algorithms*

and A9. For convenience, the RISCV softprocessor's FPGA configuration was adjusted to operate at flexible clock frequencies of 16MHz, 125MHz, and 650MHz, which allows us to have fair comparisons with other processors. Numerical results shows that RISCV softprocessor consistently outperformed the 8bit AVR Microcontroller processor across a range of Fibonacci series iterations and also presented a better performance than the ARM Cortex-M0+ processor after 300 iterations. In scenarios with less than 18 iterations, the RISCV also outperformed the ARM Cortex-A9 processor. In addition, RISCV processor shows remarkable energy efficiency by consuming only 2mW during the execution of the Fibonacci Series algorithm, which is higher to the energy consumed by the single-core ARM Cortex-M0+ processor and the 8-bit AVR microcontroller (i.e 14.22muW and 0.296mW respectively). RISCV is an attractive option in applications where energy conservation is essential, when compared the dual-core ARM Cortex-A9 processor that consumed 62.4mW due to its higher power.

Regarding the number of instructions, the RISCV processor requires fewer instructions than the 8-bit AVR and ARM Cortex-M0+ microcontrollers when executing the Fibonacci Series algorithm. Moreover, when executing the Array Multiplication algorithm, the RISCV processor needs fewer instructions than the 8-bit AVR microcontroller. Finally, for the RGB to HLB Conversion algorithm, the RISCV processor requires fewer instructions than the 8-bit AVR microcontroller. In summary, the proposed processor outperforms both the ARM Cortex-M0+ and the 8-bit AVR microcontroller in serial-based algorithms, while outperforming the 8-bit AVR microcontroller in algorithms involving matrix operations. As future work, we propose to explore other strategies to improve the energy efficiency of the processor, such as the parallelization of J-type instructions to reduce the number of instructions required in matrix operations and conversion algorithms.

References

[1] Jason G Tong, Ian DL Anderson, and Mohammed AS Khalid. Soft-core processors for embedded systems. In *2006 International Conference on Microelectronics*, pages 170–173. IEEE, 2006.

[2] GSK Gayatri Devi and G Kumara Swamy. An overview of microcontroller unit: from proper selection to specific application. *Journal of Critical Reviews*, 3(1):2016.

[3] Behzad Salami, Osman S. Unsal, and Adrian Cristal Kestelman. On the resilience of rtl nn accelerators: Fault characterization and mitigation. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 322–329, 2018.

[4] Ahmed Karim Ben Salem, Slim Ben Othman, and Slim Ben Saoud. Hard and soft-core implementation of embedded control application using rtos. pages 1896–1901, 2008.

[5] Tian Zheng, Gang Cai, and Zhihong Huang. A soft risc-v processor ip with high-performance and low-resource consumption for fpga. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2538–2541, 2022.

[6] Vx00ED;ctor Asanza, Rebeca Estrada, Jocelyn Miranda, Leiber Rivas, and Danny Torres. Performance comparison of database server based on soc fpga and arm processor. In *2021 IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–6, 2021.

[7] Louise H Crockett, Ross A Elliot, Martin A Enderwitz, and Robert W Stewart. *The Zynq book: embedded processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 all programmable SoC*. Strathclyde Academic Media, 2014.

[8] Guillermo Montesdeoca, Víctor Asanza, Rebeca Estrada, Irving Valeriano, and MA Muneeb. Softprocessor riscv-ec for edge computing applications. In *International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 209–220. Springer, 2023.

[9] Vx00ED;ctor Asanza, Rebeca Estrada, Jocelyn Miranda, Leiber Rivas, and Danny Torres. Performance comparison of database server based on soc fpga and arm processor. In *2021 IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–6, 2021.

[10] Shaodong Qin and Mladen Berekovic. A comparison of high-level design tools for soc-fpga on disparity map calculation example. *arXiv preprint arXiv:1509.00036*, 2015.

[11] Francesco Minervini, Oscar Palomar, Osman Unsal, Enrico Reggiani, Josue Quiroga, Joan Marimon, Carlos Rojas, Roger Figueras, Abraham Ruiz, Alberto Gonzalez, et al. Vitruvius+: an area-efficient risc-v decoupled vector coprocessor for high performance computing applications. *ACM Transactions on Architecture and Code Optimization*, 20(2):1–25, 2023.

[12] Victor Asanza, Rebeca Estrada Pico, Danny Torres, Steven Santillan, and Juan Cadena. Fpga based meteorological monitoring station. In *2021 IEEE Sensors Applications Symposium (SAS)*, pages 1–6, 2021.

[13] Guillermo Montesdeoca, Víctor Asanza, Kevin Chica, and Diego H. Peluffo-Ordóñez. Analysis of sorting algorithms using a wsn and environmental pollution data based on fpga. In *2022 International Conference on Applied Electronics (AE)*, pages 1–4, 2022.

[14] Jim Ledin and Dave Farley. *Modern Computer Architecture and Organization: Learn x86, ARM, and RISC-V architectures and the design of smartphones, PCs, and cloud servers*. Packt Publishing Ltd, 2022.

[15] Imran Wali, Alfonso Sánchez-Macián, Alexis Ramos, and Juan Antonio Maestro. Analyzing the impact of the operating system on the reliability of a risc-v fpga implementation. In *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–4. IEEE, 2020.

[16] Jan Gray. Grvi phalanx: A massively parallel risc-v fpga accelerator accelerator. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 17–20. IEEE, 2016.

[17] Mathijs De Kremer, Marco Brohet, Subhadeep Banik, Roberto Avanzi, and Francesco Regazzoni. Resource-constrained encryption: Extending ibex with a qarma hardware accelerator. In *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 147–155, 2023.

[18] Cristian Ramírez, Adrián Castelló, and Enrique S Quintana-Orti. A blislike matrix multiplication for machine learning in the risc-v isa-based gap8 processor. *The Journal of Supercomputing*, 78(16):18051–18060, 2022.