Digital Object Identifier

Design and Implementation of a Single-Cycle RISC-V Processor for FPGA

HENRIQUE CARDOZO DAS NEVES[1], PROF. DR. ANDRÉ INÁCIO REIS[2]
[1]UFRGS Informatics Institute, Porto Alegre, BR (e-mail: henrique.cardozo@ufrgs.br)
[2]UFRGS Informatics Institute, Porto Alegre, BR (e-mail: andreis@inf.ufrgs.br)

This work was supported by UFRGS, Softex and CI Innovator.”

ABSTRACT This paper presents the design and implementation of a RISC-V single-cycle processor in FPGA (Field Programmable Gate Array), highlighting the open and modular instruction set architecture (ISA) of RISC-V. The single-cycle processor is used for educational applications due to its simple and lean design. The circuit was implemented using the Verilog language in conjunction with the Quartus II and Questasim tools for processor design and testing.

INDEX TERMS Processor, FPGA, RISC-V, Verilog, Quartus, Questasim.

I. INTRODUCTION

A RISC-V processor is based on the RISC-V architecture, which is an open-source, royalty-free instruction set computing architecture (ISA). The term "RISC" stands for "Reduced Instruction Set Computing". In recent years, the RISC-V architecture has gained prominence in the processor landscape due to its open and flexible nature, allowing developers to create and customize their own implementations, as it offers advantages in terms of simplicity, scalability, and energy efficiency, which makes it an attractive option for many modern computing applications. An FPGA (Field Programmable Gate Array) is a type of integrated circuit that can be programmed and reprogrammed by the user after its manufacture. Unlike other chips, such as ASICs (Application Specific Integrated Circuits), which are designed for a single, fixed function, FPGAs offer high flexibility and can be configured to perform many specific tasks through hardware description languages, such as VHDL or Verilog. The ability to design and implement processors on Field-Programmable Gate Arrays (FPGAs) offers an ideal platform for rapid prototyping and testing of new architectures, as well as providing great flexibility in terms of customization and optimization for specific applications.

In this context, the development of single-cycle processors, which execute one instruction per clock cycle, is particularly interesting due to their simplicity and educational applications. The combination of the RISC-V architecture with the single-cycle implementation in FPGAs can result in systems for learning and understanding computational concepts, computer architecture to support the development of processors with a higher level of complexity, such as multicycle and pipeline.

This paper presents the design and implementation of a single-cycle RISC-V processor for FPGAs, exploring the advantages and challenges associated with this approach. Through a detailed description of the development process and the results obtained.

A. MOTIVATION

My curiosity to understand how a processor works as a whole inspired me to develop the project described in this paper. Based on the lessons learned in the Innovative CI course taught by the Federal University of Rio Grande do Sul (UFRGS), I was successful in the research and development activities of the project. I can also highlight the computational power and flexibility that FPGAs have, which can help students, educational institutions and startups to bring their prototypes to life, enabling technical flexibility aligned with a business vision to achieve their specific goals. I also see the need to inspire a new generation of scientists, hobbyists and engineers to build a more inclusive, fair, technological and scientific future for the new generations.

B. CONTRIBUTION

The purpose of this paper is to expand the technical possibilities in the development and prototyping of computational circuits. The ability to develop several prototypes of complex computational circuits with the help of FPGAs is a powerful option to align speed, quality and simplicity in the development of technological projects. To demonstrate how an open instruction set (RISC-V) aligned with the use of FPGAs can impact the time to launch a technological application on the market (prototyping time), teach cutting-edge technologies in educational institutions, promote the production of open hardware projects and the learning of hardware description languages such as Verilog.

II. RELATED WORKS

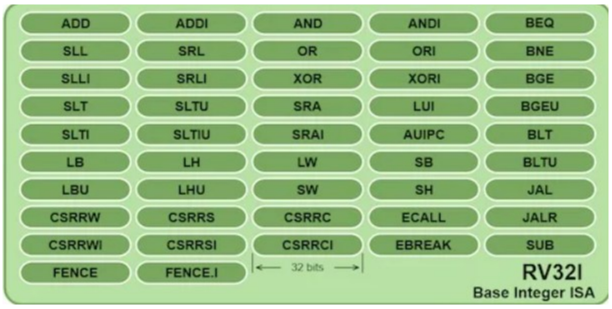
The foundation during the Innovative CI course was essential for learning digital circuits and related technologies, such as Verilog, testbenches, FPGA flow, Quartus II and QuestaSim.

Single-cycle processors are used for educational purposes due to their technical simplicity. The design of a single-cycle processor was the subject of a RISC-V book widely used in digital systems disciplines, which demonstrates the entire fundamental basis for the development of a processor, from the introduction to the RISC-V instruction set, architecture components, development and application of RISC-V CPUs, and reports on research and education in computer architecture.

III. METHODOLOGY

A. ARCHITECTURAL DEFINITION

The architecture is based on the RV32I subset of the RISC-V ISA, prioritizing basic integer instructions (lw, sw, and, or, add, subtract, beq) to simplify the single-cycle design. The RISC-V RV32I architecture is a base subset of the RISC-V ISA designed for 32-bit systems, characterized by simplicity, modularity, and efficiency. It serves as the foundation for hardware implementations such as FPGAs, ASICs, and microcontrollers, with a focus on low power consumption and high performance.



ADD	ADDI	AND	ANDI	BEQ
SLL	SRL	OR	ORI	BNE
SLLI	SRLI	XOR	XORI	BGE
SLT	SLTU	SRA	LUI	BGEU
SLTI	SLTIU	SRAI	AUIPC	BLT
LB	LH	LW	SB	BLTU
LBU	LHU	SW	SH	JAL
CSRRW	CSRRS	CSRRC	ECALL	JALR
CSRRWI	CSRRSI	CSRRCI	EBREAK	SUB
FENCE	FENCE.I	← 32 bits →		RV32I Base Integer ISA

Figure:1 Base Integer ISA RV32I

B. INSTRUCTION SET

Seven basic integer instructions (lw, sw, and, or, add, subtract, beq) were chosen for the processor implementation, which operate as follows:

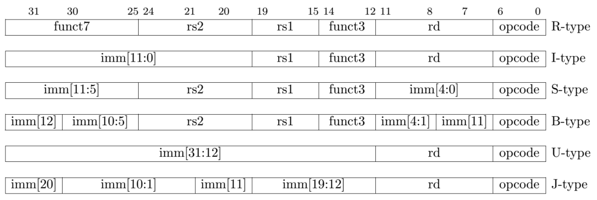
lw (Load Word): Loads a word (32 bits) from memory to a register. Uses an address calculated as base + offset, where the base is a register and the offset is an immediate 12-bit value.

sw (Store Word): Stores a word (32 bits) from a register in memory. The address is calculated as base + offset, similar to lw.

and/or (Logic): Performs bitwise logical operations (and or or) between two registers and stores the result in a third.

add/sub (Arithmetic): Performs the addition or subtraction of values stored in the 32-bit registers where the operations are to be performed.

beq (Branch if Equal): Compares two registers and branch the program flow if they are equal. The offset is an immediate value multiplied by 2 (for instruction alignment).



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

Figure:2 Instruction Set

C. EXECUTING INSTRUCTIONS

The execution of an instruction in a single-cycle RISC-V processor involves 5 steps that occur within a single clock cycle. Each step of the process follows a function defined for the execution of a specific instruction:

Instruction Fetch and PC Increment: The first step is to fetch the instruction from memory. The processor uses the current value of the Program Counter (PC) to access memory and retrieve the instruction. After the fetch, the PC is incremented to point to the next instruction in the sequence.

Instruction Decoding: The instruction is decoded to determine the type of operation that should be performed. This involves identifying the fields of the instruction, such as the opcode, source registers, and destination registers.

Reading Operands: Depending on the type of instruction, the required operands are read from the register bank. For example, for an R-type instruction (such as ADD), the values of the source registers are read.

Operation Execution: The operation specified by the instruction is executed.

This may involve arithmetic, logical, or shift operations, performed by the Arithmetic Logic Unit (ALU). For load or store instructions, the memory address is calculated and the read or write operation is performed.

Result Writing: If the instruction produces a result, it is written to the destination register or to memory, depending on the type of instruction.

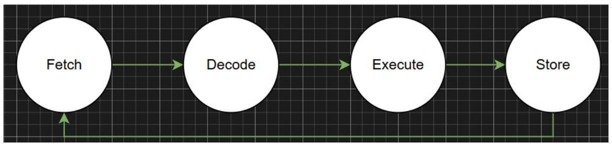


Figure:3 Instruction Cycle

D. HARDWARE DESCRIPTION LANGUAGE (HDL)

Verilog is a hardware description language (HDL) widely used for designing and simulating digital electronic systems. The Verilog hardware description language plays a crucial role in processor development, especially in the case of single-cycle RISC-V architectures.

Verilog allows the description of systems at different levels of abstraction, which is essential for processor development.

Behavioral: At this level, the focus is on the behavior of the system, without worrying about the physical implementation. This facilitates initial modeling and simulation of the processor, allowing engineers to test the functionality before moving on to detailed implementation.

RTL (Register Transfer Level): This level combines combinational and sequential logic, allowing a more detailed description of the data flow between registers. It is essential for the implementation of instructions in a RISC-V processor.

Gate Level: The system is described in terms of logic gates, which is useful for final circuit synthesis. Although more detailed, this level is less common for developing complex processors, as synthesis is usually performed automatically from RTL descriptions.

Verilog's simulation capabilities are crucial for processor development. Verilog allows you to create testbenches that provide input stimuli to test the processor's behavior in different scenarios, identifying and correcting errors before physical implementation. After simulation and testing, Verilog can be used to synthesize the processor design into a form that can be implemented on devices such as FPGAs (Field-Programmable Gate Arrays). This allows the processor to be reprogrammed multiple times, making development more flexible and cost-effective.

Verilog promotes modularity by allowing code to be organized into reusable modules. This is especially useful in the development of complex processors, where components such as ALUs, registers, and controllers can be designed and tested separately before being integrated into the complete system. In summary, Verilog is an essential tool for the development of single-cycle RISC-V processors due to its capabilities for modeling at different levels of abstraction, simulation, synthesis, and modularity.

E. SOFTWARE QUARTUS II

Quartus II is a digital systems development tool from Altera (now part of Intel) that is used to describe, compile, simulate, and program digital circuits in programmable logic devices such as Field-Programmable Gate Arrays (FPGAs).

Processor Description in HDL: To develop a single-cycle RISC-V processor using Quartus II, you must begin by describing the processor in an HDL language such as VHDL. This involves defining all of the processor modules.

Project Setup in Quartus II: Start Quartus II and create a new project using the project creation wizard. Choose the type of FPGA device you want to use (for example, Cyclone II or Cyclone III).

Compile: Compile the project to verify that there are no syntactical or logic errors. Quartus II generates detailed reports during compilation to help identify problems.

F. QUESTASIM SIMULATOR

QuestaSim is a cutting-edge simulation software developed by Siemens that offers performance and capability for simulating hardware designs. It is widely used in the development and verification of integrated circuits, including FPGA (Field-Programmable Gate Arrays).

Compilation and Simulation: QuestaSim allows you to compile designs in traditional modes or through an organized project, facilitating the management of large sets of files and libraries.

Coverage and Verification: QuestaSim provides tools to measure code and functional coverage, ensuring that all aspects of the design are adequately tested. This includes coverage of declarations, expressions, conditions, toggles, and finite state machines (FSM).

I used Questasim software to simulate the testbenches for each processor module, testing them separately to verify their operation. Then, I connected all the processor modules to test their operation as a whole.

G. DEVELOPMENT AND MODULARIZATION

The process defined to develop the 14 modules that make up the processor was:

Specification: Definition of the functionalities and requirements of each module.

Logical Design: Implementation of the modules using the Verilog hardware description language (HDL).

Simulation: Modularized tests to ensure that each module works separately and integrated into the processor as an organism.

Synthesis: Conversion of the HDL code into a bitstream, the format to be implemented in an FPGA.

The development of the 14 modules of the single-cycle RISC-V processor for FPGAs was developed using the modularization technique, where each block is developed separately from the rest of the modules.

H. MODULES

PC (Program Counter): The Program Counter (PC) is a register that stores the address of the next instruction to be executed. In the design of a single-cycle RISC-V processor, the PC is initially configured to begin execution at a specific address in the instruction memory. It is incremented each clock cycle to point to the next instruction, usually adding 4 bytes to its current value, since RISC-V instructions are 32-bit (4 bytes).

Adder (PC + 4): The Adder (PC + 4) is a circuit that adds 4 to the current value of the PC. This is necessary to advance to the next instruction in memory. This circuit is simply an adder that performs the operation PC + 4, ensuring that the PC is updated correctly after each instruction.

Multiplexer (PC): The Multiplexer (PC) is used to select between different sources of addresses for the PC. In a single-cycle processor, it can choose between the current value of the PC incremented (for normal instructions) or a new address (in the case of jumps or branches). This allows the processor to change the flow of execution as needed.

Instruction Memory: The Instruction Memory stores all the instructions that the processor will execute. In a RISC-V single-cycle processor, this memory is accessed using the PC value to read the next instruction. The instruction memory is typically implemented as a ROM.

Register File: The Register File is a set of registers that stores temporary data during the execution of instructions. In RISC-V, there are 32 general-purpose registers. It is responsible for reading and writing data from these registers as instructions are executed.

Control Unit: The Control Unit interprets the instructions read from memory and controls the flow of data between the different components of the processor. It determines which operations should be performed by the ALU, which registers should be read or written, and how the multiplexers should be configured.

Immediate Generator (Imm Gen): The Immediate Generator is responsible for handling the immediate values present in instructions. In RISC-V, many instructions include immediate values that need to be adjusted in order to be used correctly. For example, immediate values may need to be flagged or shifted to fit the word size of the processor.

Arithmetic Logic Unit (ALU): The Arithmetic Logic Unit (ALU) performs the arithmetic and logical operations required for instructions. This includes operations such as addition, subtraction, AND, OR. The ALU is a critical component of the processor, as it performs the fundamental operations that allow the processor to perform its tasks.

Control: Arithmetic Logic Unit (ALU): Control of the ALU is done by the Control Unit, which determines which operation the ALU should perform based on the current instruction. This involves setting the ALU control signals to select the correct operation.

Multiplexer (ALU): The Multiplexer (ALU) is used to select the correct operands for operations performed by the ALU. Depending on the instruction, the operands may come from different sources, such as registers or immediate values.

Memory: Memory is used to store data that is not currently in the registers. In a single-cycle processor, memory is accessed during load and store operations.

Multiplexer (Memory): The Multiplexer (Memory) selects between different data sources for memory operations. This may include choosing between data coming from the registers or from a load operation.

Branch: Branch is an operation that alters the flow of execution of the processor by redirecting it to a new address in the instruction memory. In a single-cycle processor, branching is implemented by using the PC multiplexer to select the new address instead of the incremented PC value.

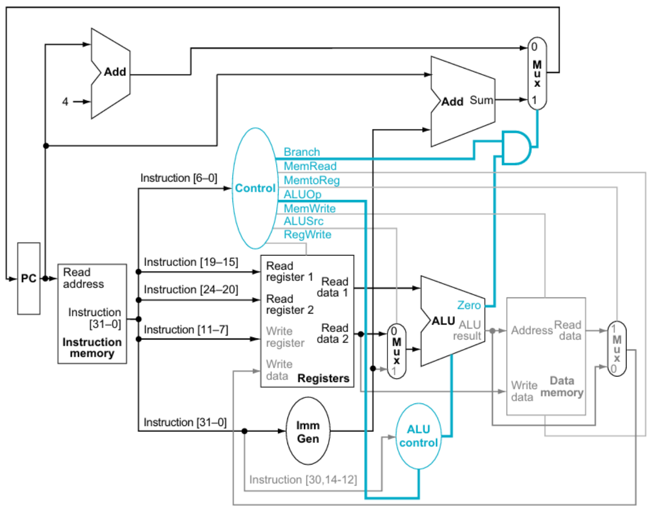


Figure:4 RISC-V Single Cycle Processor

I. PHYSICAL TESTS

After the synthesis is complete, we can proceed with the physical tests with an FPGA board by following the process:

Simulation and Compilation of the Project: Before loading the code into the FPGA, it is important to perform a simulation to verify the circuit's operation. After the simulation, compile the project again after any changes to the code or pin configuration.

File Generation (Bitstream): After successful compilation, Quartus II generates a .sof file (SRAM Object File), which is the configuration file for the FPGAs.

Hardware Configuration Setup: Connection with a USB-Blaster cable to connect the FPGA board to the computer

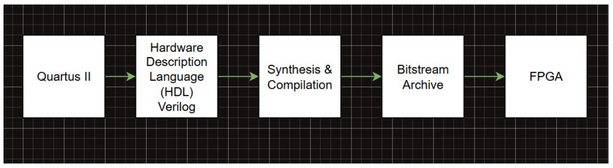


Figure:5 FPGA Flux

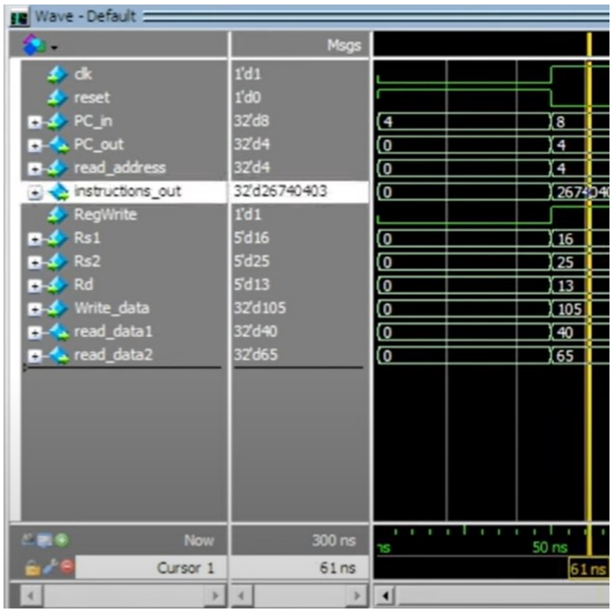
IV. RESULTS

A. INSTRUCTIONS SIMULATION

In this section, we will discuss the results after the design, testing, and synthesis of the processor were completed. To perform the tests shown below, the registers were pre-set with fixed decimal values.

To perform the instruction testing, assembly code was developed to perform each operation with the pre-set values of the registers.

B. ADD INSTRUCTION

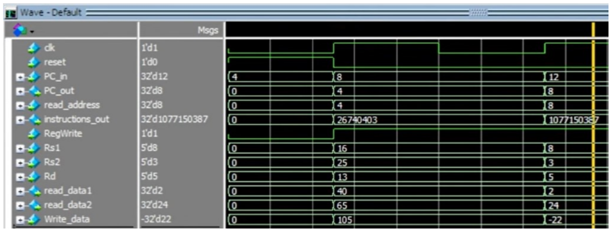


```
Wave - Default
                     Msgs
clk                  1'd1
reset                1'd0
PC_in                32'd8         4  8
PC_out               32'd4         0  4
read_address         32'd4         0  4
instructions_out     32'd26740403  0  267404
RegWrite             1'd1
Rs1                  5'd16         0  16
Rs2                  5'd25         0  25
Rd                   5'd13         0  13
Write_data           32'd105       0  105
read_data1           32'd40        0  40
read_data2           32'd65        0  65
Now                  300 ns              50 ns
Cursor 1             61 ns               61 ns
```

Figure:6 Add Instruction

```
add x13, x16, x25
```

C. SUB INSTRUCTION

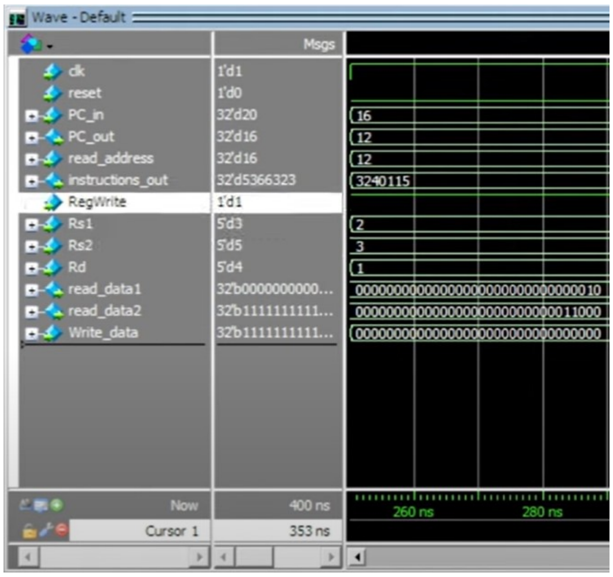


```
Wave - Default
                     Msgs
clk                  1'd1
reset                1'd0
PC_in                32'd12          4  8         12
PC_out               32'd8           0  4         8
read_address         32'd8           0  4         8
instructions_out     32'd1077150387  0  26740403  1077150387
RegWrite             1'd1
Rs1                  5'd8            0  16        8
Rs2                  5'd3            0  25        3
Rd                   5'd5            0  13        5
read_data1           32'd2           0  40        2
read_data2           32'd24          0  65        24
Write_data           -32'd22         0  105       -22
```

Figure:7 Sub Instruction

```
sub x5, x8, x3
```

D. AND INSTRUCTION

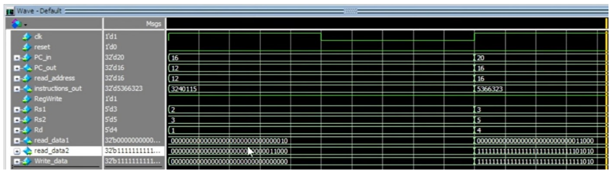


```
Wave - Default
                     Msgs
clk                  1'd1
reset                1'd0
PC_in                32'd20            16
PC_out               32'd16            12
read_address         32'd16            12
instructions_out     32'd5366323       3240115
RegWrite             1'd1
Rs1                  5'd3              2
Rs2                  5'd5              3
Rd                   5'd4              1
read_data1           32'b0000000000... 00000000000000000000000000000010
read_data2           32'b1111111111... 00000000000000000000000000011000
Write_data           32'b1111111111... 00000000000000000000000000000000
Now                  400 ns            260 ns   280 ns
Cursor 1             353 ns
```

Figure:8 And Instruction

```
addi x22, x21, 3
```

E. OR INSTRUCTION

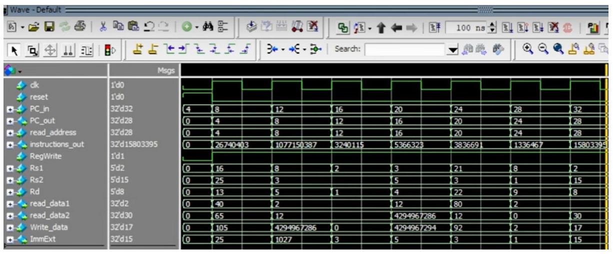


```
Wave - Default
                     Msgs
clk                  1'd1
reset                1'd0
PC_in                32'd20            16                                 20
PC_out               32'd16            12                                 16
read_address         32'd16            12                                 16
instructions_out     32'd5366323       3240115                            5366323
RegWrite             1'd1
Rs1                  5'd3              2                                  3
Rs2                  5'd5              3                                  5
Rd                   5'd4              1                                  4
read_data1           32'b0000000000... 00000000000000000000000000000010   00000000000000000000000000011000
read_data2           32'b1111111111... 00000000000000000000000000011000   11111111111111111111111111101010
Write_data           32'b1111111111... 00000000000000000000000000000000   11111111111111111111111111111010
```

Figure:9 Or Instruction

```
ori x9, x8, 1
```

F. LW INSTRUCTION

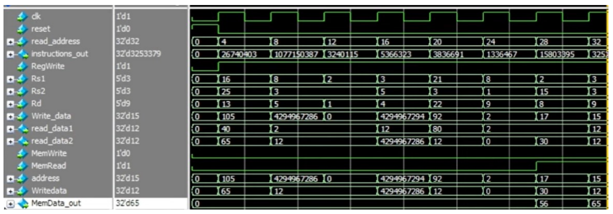


```
Wave - Default
                     Msgs
clk                  1'd0
reset                1'd0
PC_in                32'd32          4  8         12          16       20          24       28       32
PC_out               32'd28          0  4         8           12       16          20       24       28
read_address         32'd28          0  4         8           12       16          20       24       28
instructions_out     32'd15803395    0  26740403  1077150387  3240115  5366323     3836691  1336467  15803395
RegWrite             1'd1
Rs1                  5'd2            0  16        8           2        3           21       8        2
Rs2                  5'd15           0  25        3                    5           3        1        15
Rd                   5'd8            0  13        5           1        4           22       9        8
read_data1           32'd2           0  40        2                    12          80       2
read_data2           32'd30          0  65        12                   4294967286  12       0        30
Write_data           32'd17          0  105       4294967286  0        4294967294  92       2        17
ImmExt               32'd15          0  25        1027        3        5           3        1        15
```

Figure:10 Load Store Instruction

```
lw x8, 15(x5)
```

G. SW INSTRUCTION



```
clk                  1'd1
reset                1'd0
read_address         32'd32        0  4         8           12       16          20       24       28        32
instructions_out     32'd3253379   0  26740403  1077150387  3240115  5366323     3836691  1336467  15803395  325
RegWrite             1'd1
Rs1                  5'd3          0  16        8           2        3           21       8        2         3
Rs2                  5'd3          0  25        3                    5           3        1        15        3
Rd                   5'd9          0  13        5           1        4           22       9        8         9
Write_data           32'd15        0  105       4294967286  0        4294967294  92       2        17        15
read_data1           32'd12        0  40        2                    12          80       2                  12
read_data2           32'd12        0  65        12                   4294967286  12       0        30        12
MemWrite             1'd0
MemRead              1'd1
address              32'd15        0  105       4294967286  0        4294967294  92       2        17        15
Writedata            32'd12        0  65        12                   4294967286  12       0        30        12
MemData_out          32'd65        0                                                              56        65
```

Figure:11 Store Word Instruction

```
sw x15, 12(x5)
```

H. SYNTHESIZED RTL - RTL VIEWER (MODULES)

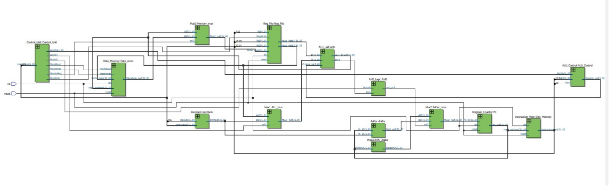


Figure:12 RTL Viewer - Modules

I. SYNTHESIZED RTL - RTL VIEWER (FULL)

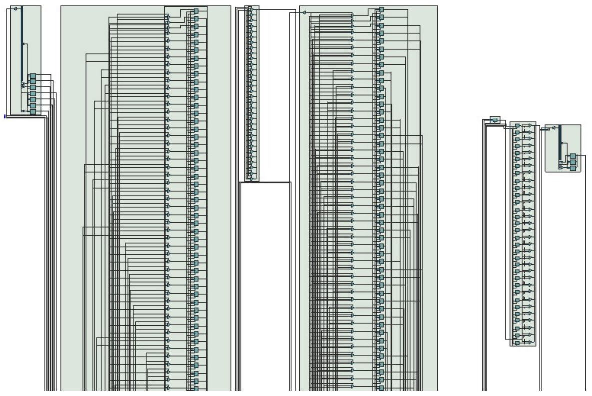


Figure:13 RTL Viewer - Full View

V. CONCLUSION

The development and implementation of a single-cycle RISC-V processor for FPGAs represents an important milestone in the field of computer architecture. This project demonstrated the feasibility of designing and implementing a highly efficient and compact RISC-V processor with 7 instructions (lw, sw, and, or, add, subtract, beq), optimized for execution on FPGAs.

The single-cycle approach adopted allows all instructions to be executed in a single clock cycle, which results in a significant reduction in complexity and improves performance predictability. In addition, the choice of the RISC-V architecture ensures flexibility and compatibility with a wide range of applications, thanks to its open source license and the growing community of developers.

The implementation on FPGAs also allows rapid prototyping and testing, facilitating the development of new applications and continuous improvements in processor design.

This work paves the way for future investigations and optimizations, such as the development of new instructions, exploration with other types of processors (multicycle or pipeline), and the exploration of new FPGA technologies.

In summary, the design and implementation of a single-cycle RISC-V processor for FPGAs demonstrates the potential of combining efficient architectures with reconfigurable programming technologies, offering an innovative and scalable solution for a variety of computing applications.

ACKNOWLEDGMENT

I would like to thank the opportunity to have participated in the first phase of the CI Inovador technology residency taught by UFRGS in partnership with Softex. I would like to thank the professors, monitors, and collaborators of UFRGS who, with great patience and perseverance, managed to transmit their knowledge to the students. I would like to thank my fellow residents for their mutual support and motivation during difficult times. I would like to thank my advisor, Prof. André Reis, for choosing to guide me from the beginning of this journey, and the group (Leonardo, Douglas, Marcelo, Ester, and Ítala) that we formed during the project development process.

REFERENCES

[1] *Computer organization and design RISC-V edition:The hardware software interface, Hennessy, John L and Patterson, David A, 2017, Elsevier Science & Technology Books*

[2] *Computer organization and design RISC-V edition:The hardware software interface, Hennessy, John L and Patterson, David A, 2017, Elsevier Science & Technology Books*

[3] *Design and Implementation of RISC-V ISA (RV32IM) on FPGA, Singh, Anmol and Kumar, Arpit and Singh, Abhishek and Reddy, R Anirudh and Pushpalatha, KN, SSRG International Journal of VLSI & Signal Processing, 2023*

[4] *Single cycle RISC-V micro architecture processor and its FPGA prototype, Dennis, Don Kurian and Priyam, Ayushi and Virk, Sukhpreet Singh and Agrawal, Sajal and Sharma, Tanuj and Mondal, Arijit and Ray, Kailash Chandra, 2017 7th International Symposium on Embedded Computing and System Design (ISED)*

[5] *RVfpga: Using a RISC-V Core Targeted to an FPGA in Computer Architecture Education, Harris, Sarah L. and Chaver, Daniel and Piñuel, Luis and Gomez-Perez, J.I. and Liaqat, M. Hamza and Kakakhel, Zubair L. and Kindgren, Olof and Owen, Robert, 2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*

[6] *FPGA Programming for Beginners: Bring your ideas to life by creating hardware designs and electronic circuits with SystemVerilog, Bruno, Frank, 2021, Packt Publishing Ltd*

[7] *Verilog HDL: a guide to digital design and synthesis, Palnitkar, Samir. 2003, Prentice Hall Professional*

[8] *Getting hands on altera quartus II software, Parab, Jivan S and Gad, Rajendra S and Naik, GM and Parab, Jivan S and Gad, Rajendra S and Naik, GM, Hands-on Experience with Altera FPGA Development Boards, pages 19–37, 2018, Springer*

[9] *Profiling Techniques in QuestaSim and Design of a Profiler Hardware Module, Mishra, Ashish and Kumar, Suresh and others, IUP Journal of Telecommunications, 2017*

[10] *A High-Level Synthesis Approach for a RISC-V RV32I-Based System on Chip and Its FPGA Implementation, Toker, On, Engineering Proceedings, 2023, MDPI*