Ben-Gurion University of the Negev

School of Electrical and Computer Engineering

Preparation Report

Implementing RISC-V Soft-core on an FPGA using Open-Source Tools Only

Project Number:	**p-2025-033**		
Student:	**Guy Cohen**	**207881004**	**guyap@post.bgu.ac.il**
Supervisor:	**Dr. Guy Tel-zur**		
Submitting Date:	**15.12.2024**		

תוכן עניינים

Implementing RISC-V Soft-core on an FPGA using Open Source Tools Only

1.Abstract

I. English version

This project aims to implement a RISC-V soft-core processor on the Ice Breaker FPGA (Lattice FPGA) using only open-source tools. A soft-core processor is a processor implemented using the FPGA fabric, in contrast to a hard-core processor, which is physically implemented as a structure within the silicon. The primary goal is to demonstrate that a fully functional processor can be developed without the need for expensive tools and commercial licenses. Open-source software, such as Yosys for synthesis and Gtkwave for simulation, will be utilized throughout the design and implementation process. After the processor is implemented, it will be tested by running encryption code (or a short code, depending on hardware constraints) to ensure its functionality. This project will showcase the feasibility of cost-effective FPGA development by utilizing only free, open-source tools for every stage, from simulation to debugging. The use of FPGAs for processor implementation offers a highly customizable and efficient platform, providing both educational and practical advantages in hardware design. Ultimately, the project will prove that open-source tools and FPGA technology can be effectively combined to develop a fully operational processor, offering a low-cost alternative to traditional development methods.

Keywords: RISC-V, FPGA, open-source tools, GTKWAVE, Yosys, Nextpnr, openocd, IceStorm, Icestudio, soft-core processor, encryption code, Amaranth HDL.

II. Hebrew version

פרויקט זה בא לממש מעבד RISC V בעל ליבה רכה על גבי רכיב FPGA של חברת LATTICE (דגם icebreaker) באמצעות שימוש בכלים חופשיים בלבד. מעבד בעל ליבה רכה הוא מעבד הממומש באמצעות שימוש בשערים הלוגיים הנמצאים בתוך שבב ה FPGA, בניגוד למעבד בעל ליבה קשה, המיושם פיזית כמבנה בתוך הסיליקון. מטרת הפרויקט היא להדגים כי ניתן לפתח מעבד אשר עובד במתכונת מלאה מבלי להשתמש בכלים יקרי ערך. כלים חופשיים כגון Yosys המשמש לסינתזה ו-Gtkwave המשמש לסימולציה של המערכת, יבואו לידי ביטוי במהלך בניית הפרויקט. לאחר סיום פיתוח המעבד, הוא ייבחן על ידי הרצת קוד הצפנה (או קוד קצר, תלוי באילוצי החומרה) אשר יבטיח את פעולתו התקינה. פרויקט זה יראה את היתכנות הפיתוח המודל של FPGA תוך שימוש אך ורק בכלי קוד פתוח בחלקים השונים של הפרויקט, משלב הסימולציה ועד שלב הצריבה. השימוש ב FPGA למימוש מעבדים מציע פלטפורמה גמישה ויעילה מאוד, ומספק יתרונות לימודיים ופרקטיים בעיצוב חומרה. בסופו של דבר, הפרויקט יוכיח שניתן לשלב בצורה אפקטיבית בין כלים פתוחים וטכנולוגיית FPGAכדי לפתח מעבד אשר פועל בצורה מלאה, מה שמציע אלטרנטיבה זולה לשיטות פיתוח מסורתיות.

מילות מפתח: RISC-V, FPGA, כלים חופשיים בלבד, GTKWAVE, Yosys, Nextpnr,IceStorm , Openocd, מעבד בעל ליבה רכה, קוד הצפנה, Icestudio, Amaranth HDL.

2.Problem Definition

The high costs associated with commercial tools for FPGA (a chip that includes cell set that can be configured in different ways[2]) development have created barriers for many individuals and institutions in experimenting with processor design and hardware development. In commercial FPGA environments, software licenses for synthesis, simulation, and debugging often come with significant costs, limiting accessibility for students and researchers. For example, the basic license for Xilinx Vivado, one of the popular FPGA synthesis tools, costs $2,995 [1]. However, a new and complete toolchain for FPGAs with its associated open tools has recently emerged from the open-source community [2].

The challenge is to implement a processor on an FPGA using only free, open-source tools for all stages of development. By doing so, this project aims to address the limitations imposed by high-cost commercial software and demonstrate that functional processors can be designed and implemented using accessible, open-source tools without compromising quality or functionality.

3.Project Objective

The objective of this project is to successfully implement a RISC-V soft-core processor on an FPGA using only open-source tools, specifically on the Ice Breaker FPGA (a Lattice FPGA). A primary goal is to reduce the development cost to zero, meaning a completely free development process. The project will utilize tools such as Yosys for synthesis and Gtkwave for simulation, ensuring that every step from design through debugging is achieved with accessible, open-source software. This approach aims to highlight the potential of open-source tools in reducing development costs, offering a low-cost alternative to traditional, license-based methods.

The project also leverages the RISC-V architecture, which stands out for its revolutionary open Instruction Set Architecture (ISA). Unlike proprietary ISAs, the RISC-V ISA is available freely, allowing developers not only to use open-source environments but also to access complete processor implementations without licensing fees. This aligns with the project's objective to create a truly open-source solution from development environment to processor core, demonstrating a cost-effective and accessible pathway for FPGA-based processor development.

4.Project Contents

In this project, our goal is to create a functional RISC-V processor with the RV32I instruction set architecture (ISA) on an FPGA platform. The project is divided into two main components: the hardware, which involves the FPGA platform, and the software, which consists of the RISC-V processor code written in Verilog.

Note: If I manage to complete the project ahead of the planned schedule, I will consider adding or innovating additional features to the project. For example:

1. **Interrupt Controller** – Adding interrupt handling capability to the processor to manage external events and I/O requests.
2. **Processor Architecture Enhancement** – Transitioning from a Pipelined processor to a Pipelined processor with performance improvements.
3. **Peripheral Devices** – Implementing auxiliary devices like timers, PWM controllers for motor control, or an RTC (Real-Time Clock).
4. **Expanding the Instruction Set (ISA)** – Adding custom instructions to enhance performance or enable special use cases.
5. **GPIO Interface** – Integrating basic input/output interfaces such as LEDs and buttons to demonstrate the processor's response to real-time inputs.

I. Hardware

The hardware component of the project consists of the FPGA platform, specifically the Ice Breaker FPGA board. This will serve as the physical environment for implementing the RISC-V processor. The FPGA will house the logic gates and memory elements that make up the processor, allowing us to configure and implement the processor's functionality.

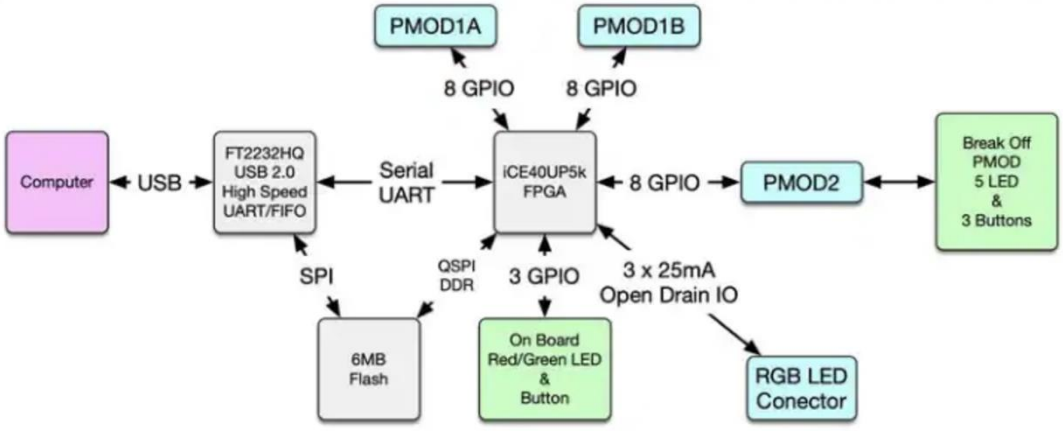


Figure 1 : Ice Breaker FPGA Block Diagram

II. Software

The software component consists of the RISC-V processor design, which is written in Verilog. The Verilog code defines the functionality of the RV32I processor, including its various components such as the ALU (Arithmetic Logic Unit), registers, control unit, and memory. Additionally, the software tools used for synthesis, simulation, and testing, such as Yosys, GTKWave, and Nextpnr, will be employed to verify the processor's operation on the FPGA platform. After the processor is implemented, I will also write assembly code to run on top of it, enabling the execution of programs and demonstrating the processor's capabilities.

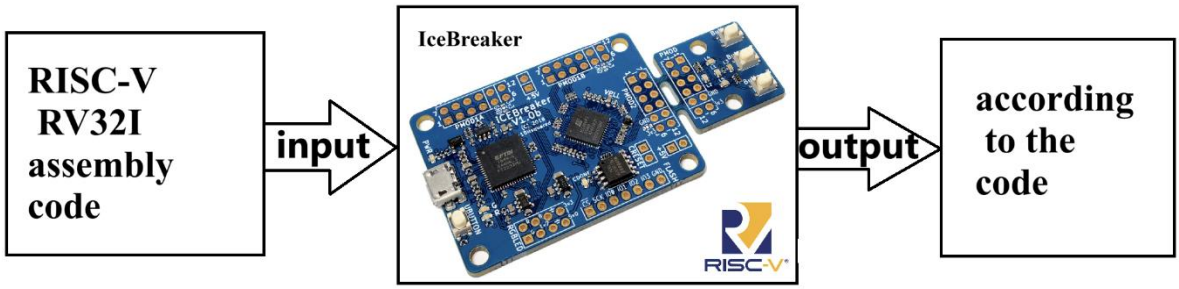


Figure 2: Project's Block Diagram

III. Final Testing

The final phase of the project focuses on rigorous testing to ensure the functionality and correctness of the implemented RISC-V processor. This phase consists of the following steps:

1. **Testing Multiple RISC-V Architectures**:
   During the project, I will implement two different types of RISC-V processor architectures, and potentially a third, depending on time constraints. Each architecture will be thoroughly tested to validate its functionality and performance.
2. **Running Initial Light Tests**:
   After implementing each processor architecture, I will execute a light test by writing and running a simple assembly program, such as printing "Hello, World!" to the screen via UART. This test will confirm basic functionality, including communication with external interfaces and proper code execution.
3. **Comprehensive ISA Validation**:
   As a final check, I will run a program that utilizes all possible instructions defined by the RV32I instruction set architecture. This comprehensive test ensures that every aspect of the processor, including the ALU, control unit, and memory, operates correctly.
4. **Execution of Complex Algorithms**:
   Depending on the hardware constraints of the FPGA platform, I plan to run a more

demanding program, such as an AES-128 encryption algorithm. This test will stress the processor's capabilities and validate its performance under heavier computational loads.

These tests collectively ensure that the processor operates as intended and demonstrates its robustness and versatility. The successful completion of these tests will confirm the functionality and reliability of the implemented RISC-V processor.

5.Proposed Methods

I. Constraints

During the preparation we identified few constraints that we will have to act accordingly –

- **Timeframe** – the time frame of this project is defined to be 8 months, from 03/11/2024 until **20/07/2025** (final submission).
- **Hardware capabilities** – As our project is aimed specifically to be done on the RISCV softcore running on the, we are limited to the capabilities of the board [11]: (More details about the chip will be provided in another section of the report)
    - 5280 logic cells
    - Internal RAM (bits): 120k + 1024k
    - Flash: 128 Mbit QSPI DDR

II. Open Source Tools

During the project, I must use free tools only. Below is a list of the tools:

- **GTKWave:** GTKWave is a fully featured GTK+ based wave viewer for Unix, Win32, and Mac OSX, capable of reading LXT, LXT2, VZT, FST, and GHW files, as well as standard Verilog VCD/EVCD files, allowing for easy viewing and analysis [3]. During the project, I will use GTKWave for simulation to verify the processor's functionality as it is implemented(see figure 1).

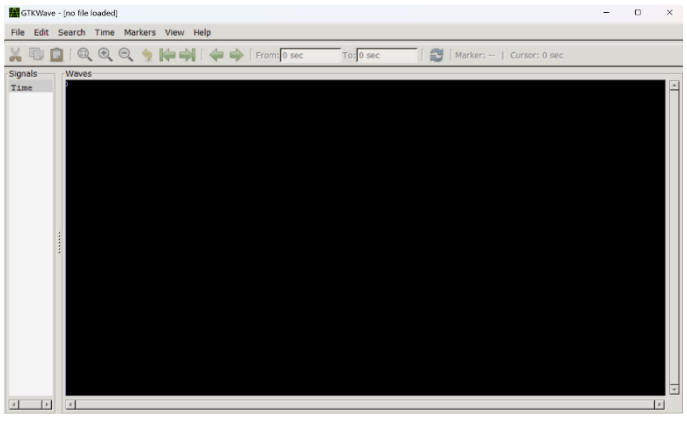


Figure 3: GTKWave home screen (without uploaded file)

- **Yosys:** Yosys is a framework for RTL synthesis and more. It currently has extensive Verilog-2005 support and provides a basic set of synthesis algorithms for various application domains. Yosys is the core component of most of our implementation and verification flows [4]. During the project, I will use Yosys for synthesizing and programming the processor code onto the FPGA chip.
- **Nextpnr:** Nextpnr is an open-source, timing-driven place-and-route (PnR) framework designed for FPGA development, supporting Linux, Windows, and macOS. Unlike traditional tools that use flat file formats, Nextpnr employs an API-based architecture, allowing it to accommodate the complexities of modern FPGAs. It provides flexibility for custom packing, routing, and bitstream generation, optimizing performance by avoiding virtual function overhead and enabling compile-time optimizations. Nextpnr uses a JSON-based format for netlist exchange and includes architecture-specific features, like timing-driven placement and iterative routing. It supports a range of hard-IP components and integrates with open-source projects like Icestorm and Trellis. Notably, it implements database deduplication to reduce disk and memory usage, improving efficiency [5].

- **OpenOCD:** OpenOCD tools are used to read out the content of memory cells and modify it [6]. In order to run code on the FPGA, I will use this tool.

- **Icestudio**: Visual editor for open FPGA boards, Icestudio generates Verilog files from visual circuits, build the Bitstream automatically and upload it to the FPGA board. This process is done using only Opensource Tools[7] (see figure 2).

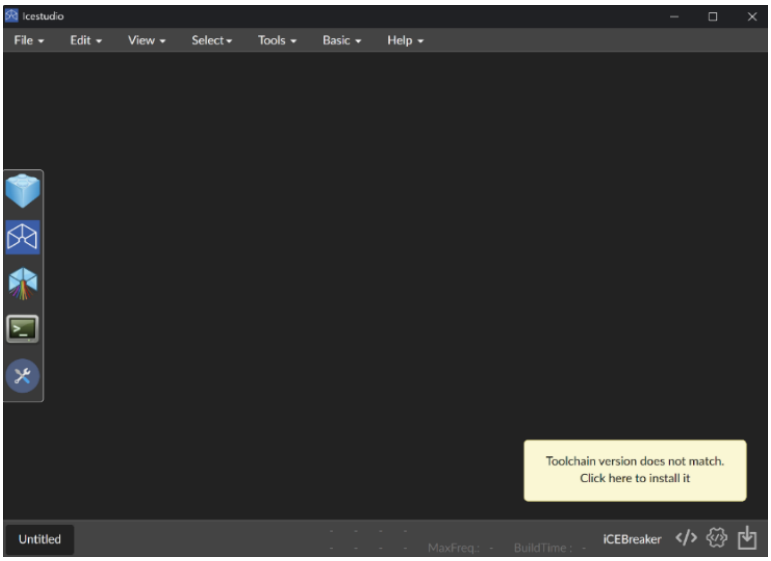


Figure 4: Icestudio home screen

- **Iverilog**: Icarus Verilog (Iverilog) is a free compiler for the IEEE-1364 Verilog hardware description language. This tool is used for simulation in conjunction with GTKWave.

Figure 5: Iverilog symbol

- **IceStorm**: IceStorm aims at documenting the bitstream format of Lattice iCE40 FPGAs and providing simple tools for analyzing and creating bitstream files. The IceStorm flow is a fully open-source Verilog-to-Bitstream flow for iCE40 FPGAs [4]. The Icestudio visual editor based on IceStorm tool.
- **MCY(Mutation Cover with Yosys)**: MCY is a tool designed to assist digital designers and project managers in assessing and enhancing testbench coverage. By introducing mutations to the design and analyzing testbench responses, MCY helps identify weaknesses in testing, allowing for more comprehensive and effective verification. This tool integrates with Yosys, an open-source RTL synthesis framework, to ensure improved test coverage in digital design projects [4].
- **Amaranth**: The Amaranth project is an open-source toolchain for hardware development, using Python to design synchronous digital logic [4]. It includes resources like a hardware definition language, a standard library, a simulator, and a build system, covering all stages of FPGA development. Amaranth aims to simplify complex hardware design, reduce coding errors, and promote reusability. Additionally, it supports flexible integration with other industry-standard tools and languages, allowing designers to combine Amaranth with Verilog or VHDL as needed.
- **APIO**: Apio is a user-friendly toolkit designed to simplify working with FPGAs. It provides pre-built tools for verifying, synthesizing, simulating, and uploading Verilog designs to supported FPGA boards.

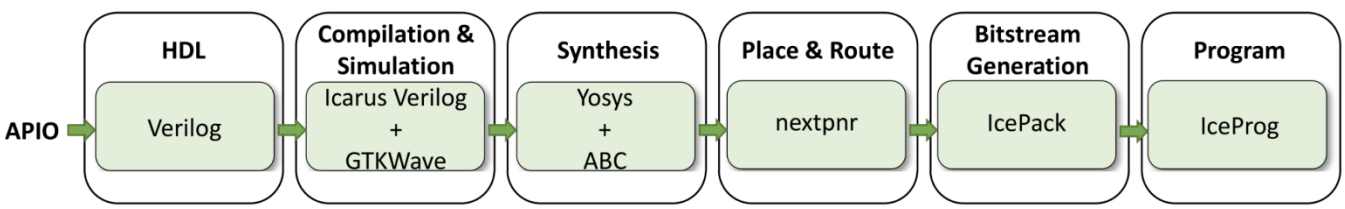


Figure 6: APIO Digital Design Workflow

III. Project Assumptions

To ensure the successful completion of the project, several assumptions were made regarding the hardware, software, and overall implementation process:

- **FPGA Platform Compatibility**:
  It is assumed that the Icebreaker FPGA board is sufficient for hosting the selected design of the RISC-V processor (see more details in literature review chapter) with the RV32I instruction set architecture. The board's resources, including logic cells, memory, and DSPs, are expected to meet the design requirements.
- **Open-Source Toolchain**:
  The project assumes the stability and reliability of the open-source toolchain. These tools are expected to integrate seamlessly with the FPGA and handle the complexities of the design.
- **Documentation and Support**:
  It is assumed that available documentation, tutorials, and resources for the iCEBreaker FPGA and associated tools will provide adequate guidance for implementation and troubleshooting.

IV. Project Risks

Despite careful planning, several risks could impact the project's progress and success. These risks are categorized below:

- **FPGA Resource Limitations:** The selected FPGA board (Icebreaker with iCE40UP5k) offers 5280 logic cells, limited internal memory, and support for specific modules. There is a risk that the project's design might exceed the available resources.
  What can we do?
  - Optimize the design by simplifying logic.
  - Explore alternate FPGA boards with larger resource capacity if necessary.
  - Add an external memory.
- **Incompatibility with Toolchains:** Potential issues with the compatibility of open-source tools with the design specifications may delay development.
  What can we do?
  - Validate all tools in a preliminary setup before proceeding with full-scale design.
  - Use well-documented and widely used open-source tools to minimize unexpected issues.

- **Project Timeline Overrun:** Unexpected design challenges, debugging, or resource limitations could result in exceeding the project timeline.

  What can we do?

  - Develop a detailed project plan with milestones and deadlines.
  - Dynamically update the project goals based on encountered challenges.

6.Theory

I. RISC-V

The objective of RISC-V is to provide a free and open ISA for industry implementations, supporting computer architecture research and education, and general-purpose software development. The RISC-V ISA is designed to be extensible, with a small base integer ISA and optional standard extensions for general-purpose software development. The base ISA is carefully restricted to a minimal set of instructions sufficient to provide a reasonable target for compilers, assemblers, linkers, and operating systems. Commercial ISAs can become outdated or lose popularity, making it difficult for third parties to continue supporting them. An open ISA can be continued and developed by interested parties even if it loses popularity [8].

Figure 7: RISCV symbol

II. RV32I (Base Integer Instruction Set)

The RISC-V ISA has 31 general-purpose registers x1-x31, which hold integer values, and one additional user-visible register: the program counter pc. The base ISA has four core instruction formats (R/I/S/U), which are a fixed 32 bits in length and must be aligned on a four-byte boundary in memory. RV32I (Base Integer Instruction Set) includes [8]:

- R-type operations, including ADD, SUB, SLT, SLTU, AND, OR, and XOR.

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	ADD/SLT/SLTU	dest	OP
0000000	src2	src1	AND/OR/XOR	dest	OP
0000000	src2	src1	SLL/SRL	dest	OP
0100000	src2	src1	SUB/SRA	dest	OP

Figure 8: Integer Register-Register Operations

- Integer Register-Immediate Instructions:

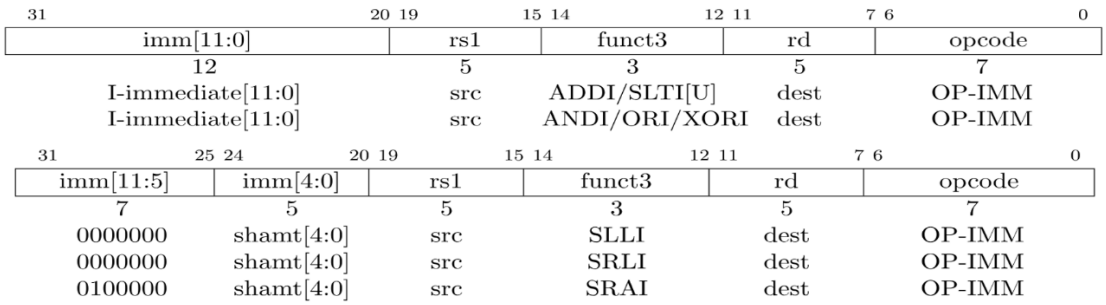


Figure 9: Integer Register-Immediate Instructions structure

- The NOP instruction does not change any user-visible state, except for advancing the pc.

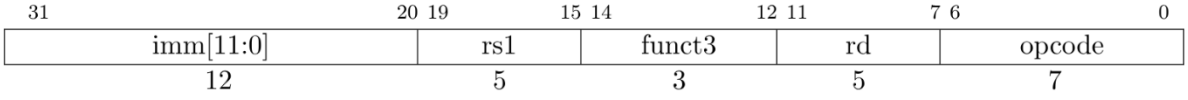


Figure 10: NOP instruction structure

- LUI (load upper immediate) and AUIPC (add upper immediate to pc):

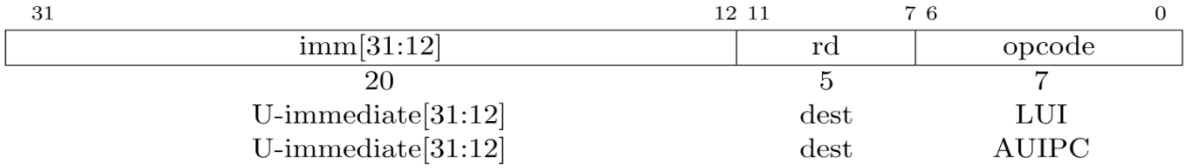


Figure 11: LUI and AUIPC instructions structure

- JAL expands to jal x1, offset [11:1] and writes the address of the instruction following the jump (pc+2) to the link register, x1.

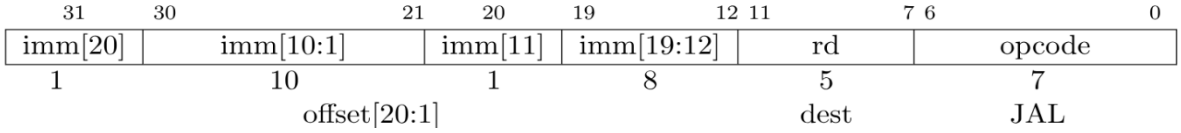


Figure 12: JAL instruction structure

- JALR performs the same operation as C.JR, but additionally writes the address of the instruction following the jump (pc+2) to the link register, x1, and expands to **jalr x1, rs1, 0**.

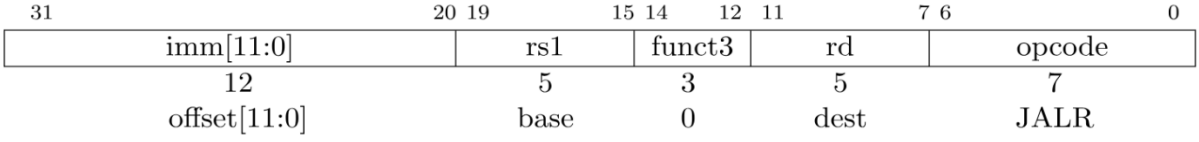


Figure 13: JALR instruction structure

- Branch instructions compare two registers, All branch instructions use the B-type instruction format.

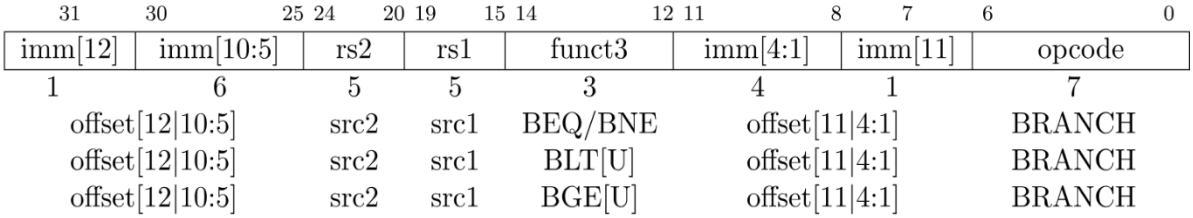


Figure 14: Branch instructions structure

- Load and Store instructions, Loads and stores are encoded in I-type and S-type formats, respectively.

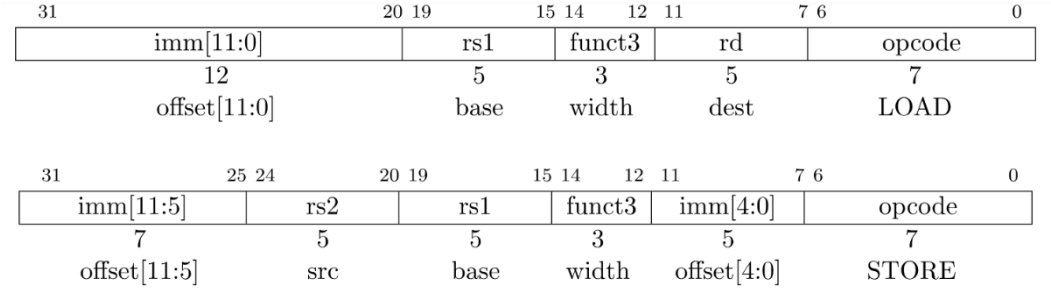


Figure 15: Load and Store instructions structure

III. A sequential 5-stages pipeline

Pipelining is an implementation technique that overlaps the execution of multiple instructions, taking advantage of parallelism among the actions needed to execute an instruction. The throughput of an instruction pipeline is determined by how often an instruction exits the pipeline. The time required between moving an instruction one step down the pipeline is a processor cycle, which is usually 1 clock cycle [9]. The pipeline structure includes a series of registers placed between consecutive pipeline stages, ensuring smooth transfer of data and control signals from one stage to the next. These registers act as intermediaries, holding values temporarily as they move through the pipeline. The program counter (PC) is implemented as an edge-triggered register, updated at the end of each clock cycle. This design prevents race conditions during PC updates, maintaining synchronization and reliable operation [9].

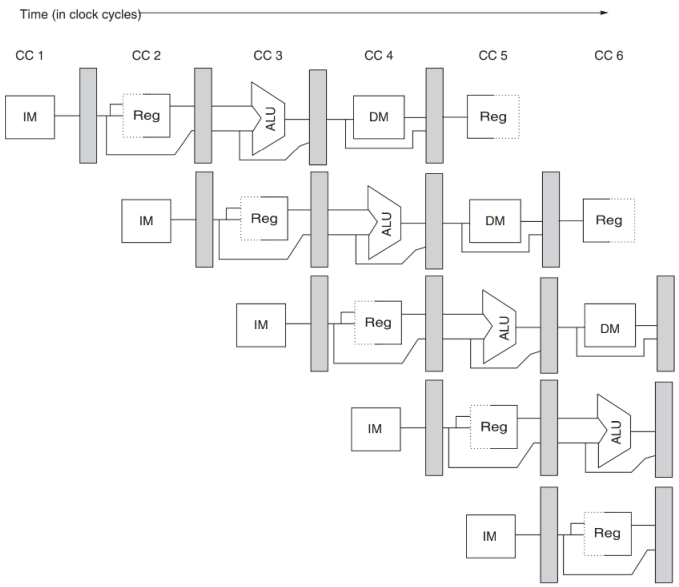


Figure 16: pipeline registers between successive pipeline stages

IV. Lattice iCE40UP5k FPGA

The iCEBreaker FPGA board is a versatile and open-source development tool designed to facilitate digital logic design and exploration, particularly in educational settings. It is fully compatible with the latest open-source FPGA development tools and is ideal for working with next-generation open CPU architectures. The board includes three standard Pmod connectors, which allow for easy expansion with a wide variety of third-party modules, as well as custom-designed Pmods. This makes it an excellent choice for projects that require additional peripherals, such as a seven-segment display, DIP switches, or HDMI output. The iCEBreaker board comes with a breakaway Pmod featuring three pushbuttons and five LEDs, enabling users to start experimenting immediately [10]. Additionally, the Lattice iCE40UP5k FPGA on the board is powered by three power supplies: VCORE, which provides 1.2V for the internal FPGA components, and VCCIO0&1 and VCCIO2, which supply 3.3V for the I/O pins [11]. Its open-source nature, combined with ample documentation and support from tools like Yosys, Arachne-pnr, and IceStorm, makes the iCEBreaker an excellent platform for students and educators to learn about FPGA design and development.

- Features of the chip [12]:
    - 5280 logic cells (4-LUT + Carry + FF)
    - 120Kbit dual-port block RAM
    - 1Mbit (128KByte) single-port RAM

    - PLL, 2x SPI, 2x I^2C hard IPs
    - Two internal oscillators (10kHz and 48MHz) for simple designs
    - 8x DSP multiplier blocks for signal processing such as audio synthesis and even software-defined radio
    - Low power consumption is ideal for battery-powered applications
    - 3x 24mA drive and 3x hard IP PWM (can drive RGB LEDs and small motors)
- I/O options 3x pins (header) for RGB LED
    - 2x onboard LEDs
    - UART, RX pin, and TX pin accessible via virtual (USB) serial port
    - One push button
    - 2x available Pmod connectors (16 x pins total)
    - Breakaway Pmod (8x pins)
- Storage
    - 128Mbit (16MB) quad SPI double data rate (QSPI-DDR) flash
- Prewired breakaway Pmod module
    - Input and output user accessible
    - 5x LEDs in a star pattern
    - 3x pushbuttons
- Onboard FPGA programmer and USB-to-serial adapter
    - Compatible with IceStorm iceprog tool
    - Driverless connection as a serial device to host computer
- USB high speed
    - Onboard FT2232 USB chip
    - Up to 480Mbit/s interface to the host computer

7. Literature review

The RISC-V architecture has been implemented in numerous processors, each designed for various levels of performance and complexity. These implementations exhibit diverse characteristics, with each processor offering unique features tailored to specific use cases and applications. The modularity and flexibility of the RISC-V Instruction Set Architecture (ISA) enable developers to adapt designs to their requirements, ranging from minimalistic embedded processors to high-performance computing systems. In this chapter, the implementation of difference architectures will be discussed.

I. FemtoRV

This project draws inspiration from Bruno Levy's work on implementing RISC-V processors, specifically the FemtoRV design. Levy's FemtoRV is a minimalist RISC-V implementation designed to be both educational and practical. Its most basic variant, quark, is a straightforward RV32I core, written in just 400 lines of documented Verilog code. Despite its simplicity, the processor is capable of executing compiled C programs and is fully compliant with the RISC-V ISA.

One of the notable aspects of FemtoRV is its modular design philosophy, which emphasizes incremental learning. Starting with the basic RV32I core, developers can progressively extend the processor by incorporating features such as pipelining, additional instruction sets, or even multi-core functionality. This step-by-step approach makes it suitable for educational purposes and allows developers to explore the trade-offs between simplicity and performance.

While the FemtoRV processors are not optimized for high performance, they are functional and versatile. For instance, the larger petitbateau core supports RV32IMFC and demonstrates the capability to run complex applications like the game DOOM, provided sufficient hardware resources are available. Levy also provides a detailed tutorial that guides users through the hardware and software development process, making the FemtoRV project a valuable resource for learning processor design and RISC-V programming.

By adapting the principles and methodologies used in FemtoRV, our project aims to implement a custom RISC-V processor with a focus on education and modularity, while exploring enhancements such as pipelining to improve performance. This iterative development approach aligns with the modular and scalable philosophy of the RISC-V architecture [13].

II. PicoRV32

PicoRV32 is a compact and highly optimized RISC-V processor that implements the RV32IMC instruction set. It is designed to be small, highly efficient, and suitable for integration into FPGA and ASIC designs. Notably, it can be configured as various core types, including RV32E, RV32I, RV32IC, RV32IM, and RV32IMC. The processor is available as open-source hardware, licensed under the ISC license, similar to the MIT and BSD licenses[14].

- **Key Features**:
    - **Small Footprint**: Requires between 750-2000 LUTs (Look-Up Tables) on 7-Series Xilinx FPGAs.
    - **High Frequency**: Capable of achieving a high maximum frequency (250-450 MHz) on 7-Series FPGAs.
    - **Interrupt and Co-Processor Support**: Optional interrupt handling (IRQ) and co-processor interface (PCPI) to extend functionality.
- **Performance**: The PicoRV32 processor is optimized for size and maximum frequency rather than raw computational performance. Its cycles per instruction (CPI) can vary depending on the instruction mix, but the typical CPI is around 4.0, with specific instructions such as memory loads and stores requiring 5-6 cycles. This makes the processor highly efficient for embedded systems where size and clock speed are critical, though not necessarily the highest in computational throughput.
- **Applications**: Due to its small size and high fmax, the PicoRV32 is suitable for embedded systems, FPGA designs, and ASIC applications, particularly as an auxiliary processor in larger systems where it can perform specific tasks or handle interrupts without compromising the overall system's timing.

III. SERV - The Smallest RISC-V CPU

The SERV RISC-V CPU is an award-winning, highly compact processor core designed to minimize silicon area while maintaining compliance with the RISC-V ISA. SERV emphasizes simplicity and minimalism, making it particularly suitable for embedded systems where area constraints are critical.

- **Key Features**:
    - **ISA**: Implements RV32IZifencei with optional extensions (C, M, Zicsr).
    - **Bit-Serial Architecture**: Processes one bit per clock cycle, a design choice that allows for extreme area reduction.

  - **Minimal Silicon Area**: The smallest RISC-V core available, requiring only 198 LUTs and 164 flip-flops in Lattice iCE40 FPGA architecture.
  - **Optional Features**: Supports timer interrupts, custom extensions, and integration of a multiplication/division unit (MDU).
  - **Software Support**: Compatible with standard RISC-V toolchains and supports the Zephyr 3.7 operating system.
  - **License**: Open-source under the ISC license, with optional commercial licenses.

- **Design Philosophy and Applications**:
  SERV's bit-serial design processes one bit per clock cycle, trading off performance for a drastically smaller silicon area. While this architecture inherently limits execution speed, it excels in applications where area, power consumption, and simplicity are prioritized over computational throughput. Examples include low-power IoT devices, basic embedded systems, and educational tools for learning processor architecture.

- **Extension Interface**:
  To enhance flexibility, SERV provides an extension interface for integrating custom accelerators. For example, by setting the MDU parameter, an external multiplication/division unit can implement the M ISA extension. Other custom accelerators may require modifications to the decoder but can be integrated seamlessly.

- **Comparison to Other Implementations**:
  SERV distinguishes itself from processors like FemtoRV and PicoRV32 by focusing almost exclusively on minimizing silicon area. While FemtoRV emphasizes modularity and educational use, and PicoRV32 balances size with high clock frequencies, SERV's bit-serial design achieves the smallest possible implementation, making it ideal for scenarios with extreme area constraints. However, its bit-serial approach results in higher cycle counts for executing instructions compared to the parallel architectures of FemtoRV and PicoRV32[15].

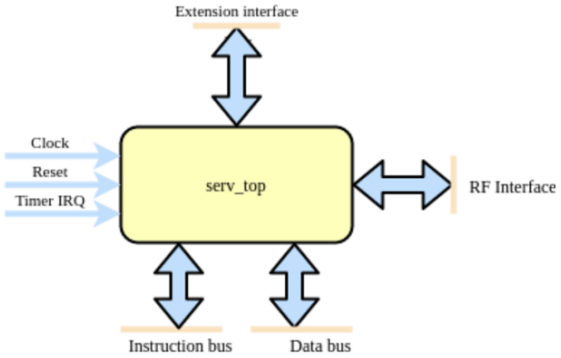


Figure 17: SERV RISC-V top entity

IV. ASTIRV32I

The ASTIRV32I processor represents one of the most fundamental implementations of the RISC-V architecture, specifically targeting the RV32I instruction set. This baseline version works within a 32-bit address space and includes basic computational capabilities such as integer arithmetic, loads, stores, and control flow instructions. Its minimalist design adheres to the essential requirements of the RISC-V specification while emphasizing simplicity and modularity [16].

- **Core Architecture**:

  ASTIRV32I utilizes a Von Neumann architecture, in which instructions are processed sequentially. This approach simplifies the design, resulting in a distributed control unit without a centralized controller.

  The modular design leverages multiplexers for internal interconnections, enabling flexibility in system modifications. The addition or removal of components can be achieved by adjusting multiplexer inputs, which facilitates extensibility.

- **Memory and Interface Design**:

  The processor architecture incorporates an external RAM module, configurable through generic parameters to create a customized memory map. This modularity ensures compatibility with various applications by supporting byte-addressable memory.

  A UART interface was implemented to streamline communication and validation processes. This interface is managed by a finite state machine (FSM) comprising six states: IDLE, DECODE, RESET, WRITE MEM, START ASTIRV32I, and READ MEM.

- **Resource Utilization and Performance:** ASTIRV32I was implemented on the Lattice iCE40-HX8K FPGA. The processor demonstrates efficient use of logic cells,

LUTs, and flip-flops. While the standalone ASTIRV32I core consumes minimal resources, integrating the UART interface and RAM increases utilization. The design's parameterized nature ensures flexibility, enabling tailored resource allocation for specific applications.

- **Logic Cells**: 4320 (56.25% Usage)
- **LUT**: 3086 (40.18% Usage)
- **Flip-Flops**: 1135 (14.77% Usage)

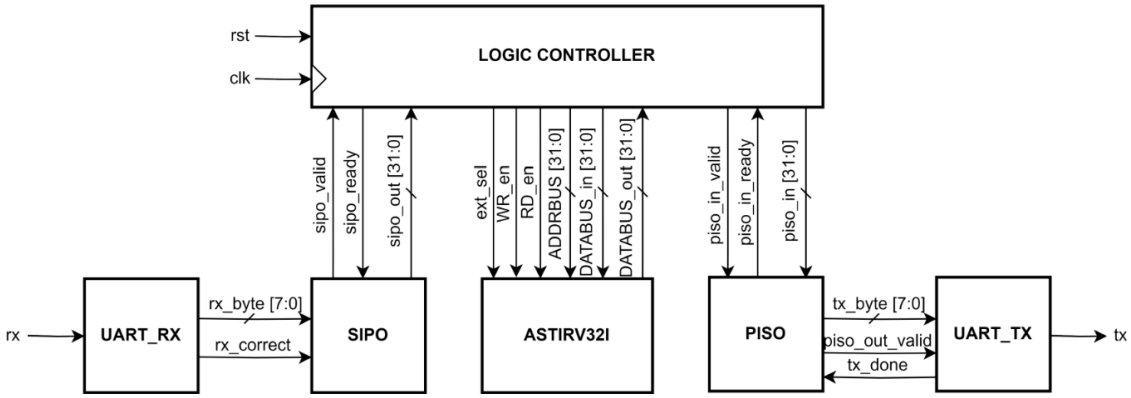


Figure 18: ASTIRV32I Interface Block Diagram

V. Comparison

The RISC-V processors FemtoRV, PicoRV32, SERV, and ASTIRV32I each represent unique approaches to implementing the RISC-V Instruction Set Architecture (ISA). Below is a comparison highlighting their key attributes, strengths, and trade-offs:

Aspect	FemtoRV	PicoRV32	SERV	ASTIRV32I
Design Philosophy	Modular, educational. Incremental learning	Compact, optimized for size and frequency.	Minimalistic, prioritizes area reduction.	Minimalistic, modular, baseline RV32I.
Target Applications	Education, experimentation.	Embedded systems, FPGAs, ASICs.	Low-power IoT, area-constrained devices.	Customizable embedded systems.
ISA Support	RV32I (quark variant) with optional extensions (e.g.,	RV32I, RV32IC, RV32IM,	RV32IZifencei with optional extensions (C, M, Zicsr).	RV32I (integer arithmetic, loads, stores,

	M, F, C for larger variants).	RV32IMC, and RV32E.		and control flow).
Footprint (LUTs)	400 (quark core), larger for extensions.	750–2000 on Xilinx 7-Series FPGAs.	198 on Lattice iCE40 FPGAs.	3086 on Lattice iCE40-HX8K FPGA.
Modularity	High: Incremental upgrades (pipelining).	Moderate: Supports interrupts and PCPI.	Moderate: Allows for accelerators.	High: Extensible with multiplexers.
Max Frequency (fmax)	Limited, not optimized for high speed.	250–450 MHz on Xilinx 7-Series FPGAs.	Low due to bit-serial design.	Not explicitly optimized for speed.
CPI	~1.8	~4.0, 5–6 for memory operations.	High (1 bit processed per clock).	Sequential execution, modest CPI.

Findings:

Diverse Applications and Design Approaches:

- **FemtoRV**: Primarily suited for educational purposes, focusing on incremental learning with high modularity and potential extensions like pipelining and multi-core designs.
- **PicoRV32**: Optimized for embedded systems and ASICs, offering a good balance between size, performance, and clock frequency, but with limitations in computational performance.
- **SERV**: Focuses on minimizing silicon area, making it ideal for low-power IoT systems but with very low performance.
- **ASTIRV32I**: A minimalistic and modular design, well-suited for customized embedded systems, but not optimized for speed.

Suitability for Complex Tasks:

- **FemtoRV** and **PicoRV32** are better suited for computational tasks like AES128 encryption, thanks to potential extensions.

- **SERV** is not suitable for complex tasks like AES128 due to its extremely low performance (processing one bit per clock cycle).
- **ASTIRV32I** could be adapted with external resources or acceleration (e.g., co-processor), but it has limitations due to its basic design.

Analysis for AES128 Encryption:

- **FemtoRV** and **PicoRV32** could be suitable for AES128 implementation if extensions are used(It is likely that additional memory will be required to run this encryption code).
- **SERV** is less suitable for heavy computational tasks like AES128 due to its bit-serial design.
- **ASTIRV32I** would require customization or external acceleration to improve performance.

VI. Conclusion

FemtoRV is the ideal choice for implementing AES128 encryption due to its modular and educational design, which allows for incremental enhancements as needed. With its base RV32I core, FemtoRV is lightweight and simple, yet it can be expanded to accommodate more complex tasks like encryption. The modularity of FemtoRV enables easy integration of extensions such as pipelining or additional instruction sets, which can enhance performance for computationally intensive tasks like AES128. Furthermore, FemtoRV's compatibility with the RISC-V ISA ensures it can support the necessary operations and software, making it well-suited for both educational purposes and practical implementations. While not optimized for high performance out of the box, its extensibility and scalability offer a path to achieve the necessary capabilities for AES128 encryption with the addition of more resources, After implementing the FemtoRV core, and based on various considerations, it will be decided whether additional adjustments are necessary for running the encryption algorithm, or if we should run less resource-intensive code instead.

8. Budget Estimation

I. Equipment

Item	Cost	Purchase/ lab equipment
Icebreaker (Lattice) board model: iCE40UP5K[9]	65 USD	Purchase
Private computer: Asus ZenBook Pro 15 UX535LI-H2170T	₪6879	Purchase

II. Software and license

Name	Cost
Open-source tools only	0 USD

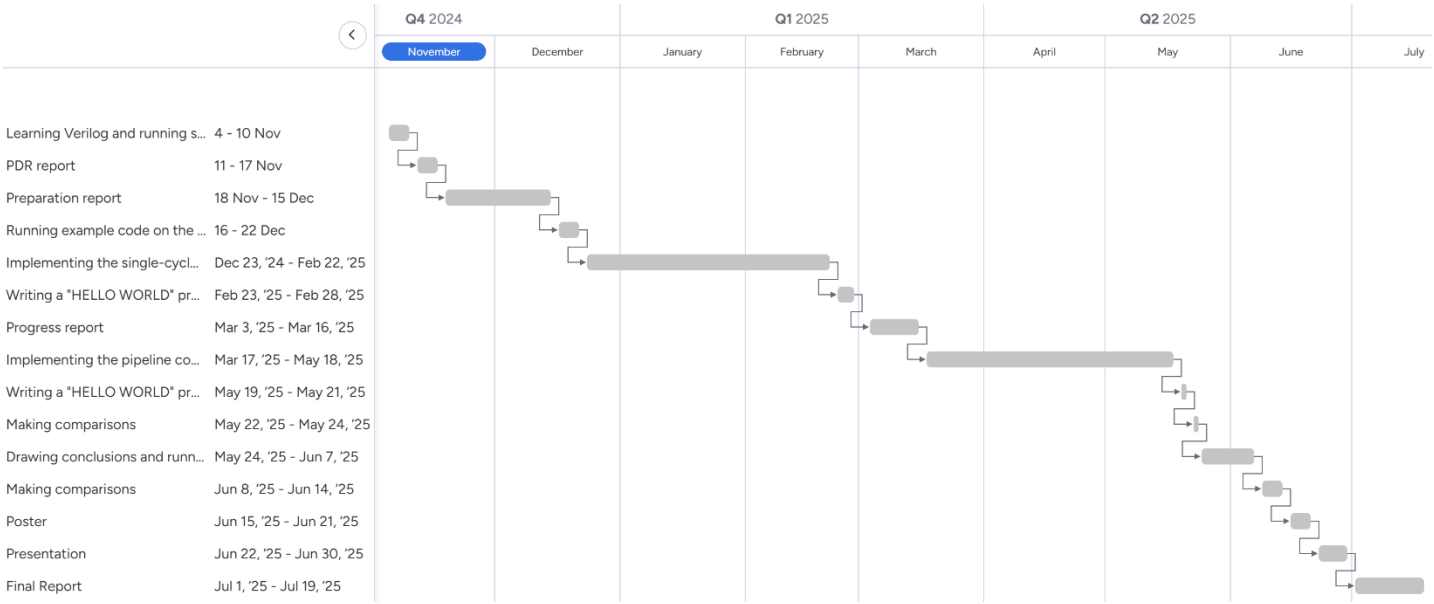
III. Manpower

Name	Hours approximation	Cost
Student	720 Hours	$720 \times 30\$_{per\ hour} = 21600\$$
Supervisor	72 Hours	$72 \times 100\$_{per\ hour} = 7200\$$

- **Total:** in practice we do not plan to have any expenses, as all necessary tools must be open source only.

9. Schedule

I. Gantt



II. Table

Task	Status	Timeline
Learning Verilog and running simulations	Done	4 - 10 Nov
PDR report	Done	11 - 17 Nov
Preparation report	Working on it	18 Nov - 15 D...
Running example code on the chip	Not Started	16 - 22 Dec
Implementing the single-cycle configuration of the Femto-RV	Not Started	Dec 23, '24 - F...
Writing a "HELLO WORLD" program in C and running it	Not Started	Feb 23, '25 - F...
Progress report	Not Started	Mar 3, '25 - M...
Implementing the pipeline configuration of the FemtoRV	Not Started	Mar 17, '25 - ...
Writing a "HELLO WORLD" program in C and running it	Not Started	May 19, '25 - ...
Making comparisons	Not Started	May 22, '25 - ...
Drawing conclusions and running AES128 code on both cores (hardware constraints permitting) or simpler code.	Not Started	May 24, '25 - ...
Making comparisons	Not Started	Jun 8, '25 - Ju...
Poster	Not Started	Jun 15, '25 - J...
Presentation	Not Started	Jun 22, '25 - J...
Final Report	Not Started	Jul 1, '25 - Jul ...

Note:

If I manage to complete tasks ahead of the planned schedule, I will consider adding extra features or introducing new innovations to the project.

10.Bibliography

[1] AMD official website, https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado/vivado-buy.html

[2] José M. Cañas, Jesús Fernández-Conde, Julio Vega and Juan Ordóñez (2022) Reconfigurable Computing for Reactive Robotics Using Open-Source FPGAs, Spain.

[3] GTKWave official website, https://gtkwave.sourceforge.net.

[4] YosysHQ official website, https://www.yosyshq.com/open-source.

[5] David Shah, Eddie Hung, Clifford Wolf, Serge Bazanski, Dan Gisselquist and Miodrag Milanovic (2019) Yosys+nextpnr: an Open Source Framework from Verilog to Bitstream for Commercial FPGAs

[6] E V Tabakov, A I Zinina and D S Kireev (2020) Debugging method for spacecraft's equipment based on ARM processors, Russia.

[7] https://github.com/FPGAwars/icestudio/wiki

[8] Andrew Waterman1 , Krste Asanovi´c1,2 (2017) The RISC-V Instruction Set Manual Volume I: User-Level ISA, CS Division, EECS Department, University of California, Berkeley.

[9] John L. Hennesy, David A. Patterson (2012) Computer Architecture A Quantitative Approach (5th edition), Stanford University, University of California, Berkeley.

[10] CROWSSUPPLY website, https://www.crowdsupply.com/1bitsquared/icebreaker-fpga.

[11] Luca Martis, Gianluca Leone, Luigi Raffo, Paolo Meloni (2024) Low-Power FPGA-based Spiking Neural Networks for Real-Time Decoding of Intracortical Neural Activity, IEEE SENSORS JOURNAL

[12] MOUSER ELECTRONICS website, https://www.mouser.co.il/new/1bitsquared/1bitsquared-icebreaker-fpga-dev-boards/

[13] https://github.com/BrunoLevy/learn-fpga?tab=readme-ov-file

[14] https://github.com/YosysHQ/picorv32

[15] https://github.com/olofk/serv?tab=readme-ov-file

[16] Pablo Navarro-Torrero, Macarena C. Mart´ınez-Rodr´ıguez, Angel Barriga-Barros, Piedad Brox, Full Open-Source Implementation of an Academic RISC-V on FPGA, Instituto de Microelectronica de Sevilla, (IMSE-CNM), CSIC / Universidad de Sevilla, Sevilla, Spain

11. Recommended grade

המלצת ציון (ע"י מנחה אקדמי) לדו"ח מכין

מספר הפרויקט: p-2025-033

שם הפרויקט: מימוש מעבד RISCV בעל ליבה רכה על גבי FPGA באמצעות כלים חופשיים בלבד.

שם המנחה מהמחלקה: גיא תל-צור

שם הסטודנט: גיא כהן ת.ז.: 207881004

%		חלש	בינוני	טוב	ט"מ	מצוין	
		55-64	65-74	75-84	85-94	95-100	
15	הבנת הנושא הצורך וסביבת היישום						
15	חיפוש מקורות והבנת עבודות דומות						
15	שלמות דף מפרט (הצעת מחקר)						
15	הצעת תכנון ותכנון הבדיקות הסופיות						
10	גילוי יוזמה וחריצות						
20	פתרון בעיות, מקוריות ותרומה אישית (מעבר למילוי ההנחיות)						
10	הערכת תקציב, לו"ז וחלוקת עבודה, ציון מקורות ושלמות כללית						

הערכת רמת הקושי של הפרויקט: קל מאוד / קל / בינוני / קשה / קשה מאוד

הערות: