ABSTRACT

This mini project focuses on the complete design and implementation of an **8-bit single-cycle Central Processing Unit (CPU)** using **Verilog Hardware Description Language (HDL)**. The primary objective of this work is to strengthen foundational knowledge in digital design and computer architecture by constructing a processor from the ground up—covering its datapath, control logic, instruction set architecture, and memory organization.

The CPU designed in this project implements a simple yet functional instruction set consisting of arithmetic, logical, memory access, and branch instructions. Each instruction is encoded in an 8-bit format, enabling compact representation and ease of decoding. The processor supports operations such as **ADD, SUB, AND, OR**, as well as **LOAD, STORE, JMP, and conditional jump (JZ)**. The architecture follows a **single-cycle execution model**, meaning each instruction is fetched, decoded, executed, and completed within one clock cycle. This approach not only simplifies the control logic but also provides a clear conceptual foundation for understanding how classical RISC processors operate internally.

Key hardware modules developed as part of this CPU include the **Program Counter (PC), Instruction Memory (IM), Register File (RF), Arithmetic Logic Unit (ALU), Data Memory (DM), and a combinational Control Unit**. All modules were coded in Verilog, simulated extensively, and verified through waveform analysis to ensure correctness of instruction execution. Additionally, a **Python-based assembler** was developed to convert human-readable assembly instructions into machine code compatible with the Verilog instruction memory initialization. This significantly improved testing efficiency and enabled rapid experimentation with different programs.

The project provides practical exposure to RTL design, modular hardware development, instruction encoding, debugging, and simulation workflows used in FPGA-based processor implementation. It reinforces crucial engineering concepts and serves as a strong foundation for more advanced designs, including pipelined CPUs, multi-cycle architectures, and embedded microcontrollers. Overall, this mini project demonstrates a complete working example of a custom-designed processor and highlights the educational value of implementing digital systems through Verilog HDL.

List of Figures

List of Tables

List of Symbols

S.No.	Symbol	Description
1	PC	Program Counter
2	IM	Instruction Memory
3	DM	Data Memory
4	Reg	Register
5	CPU	Central Processing Unit
6	ALU	Arithematic and Logic Unit
7	FPGA	Field Programmable Array Logic
8	ASIC	Application Specific Integrated Circuits
9	HDL	Hardware Description Language
10	LVS	Layout versus Schematic
11	DRC	Design Rules Check

Contents

……

Chapter 1: Introduction

The rapid advancement of digital systems, embedded controllers, and semiconductor technologies has made processor architecture a foundational subject in modern electronics and computer engineering. At the heart of every computing system lies the **Central Processing Unit (CPU)**, a digital circuit responsible for executing instructions, performing arithmetic and logical operations, and controlling data flow within the system. Understanding the internal working of a CPU—its datapath, control logic, memory hierarchy, and timing—is essential for engineers involved in fields such as **VLSI design, embedded systems, computer architecture, and digital logic design**.

This project involves the complete design and implementation of a **custom 8-bit single-cycle CPU**, beginning from Register-Transfer Level (RTL) modeling using **Verilog HDL**, followed by functional verification, FPGA prototyping, and finally an ASIC backend flow using **open-source EDA tools**. Unlike typical academic projects that end at simulation or FPGA-level execution, this work explores the entire pipeline of digital system development, including **RTL design, FPGA testing, synthesis, placement, routing, layout generation, Design Rule Checking (DRC), and Layout vs. Schematic (LVS) verification**, demonstrating readiness for real silicon fabrication.

The 8-bit CPU designed in this project supports a compact yet functional **Instruction Set Architecture (ISA)** consisting of arithmetic (ADD, SUB, AND, OR), data transfer (LOAD, STORE), and control flow instructions (JMP, JZ). The processor uses a **single-cycle execution model**, in which each instruction completes all stages—fetch, decode, execute, memory access, and write-back—within one clock cycle. This architecture simplifies the control unit and provides a clear understanding of data movement across the processor.

To validate real-time behavior, the CPU was synthesized and deployed on the **PYNQ-Z2 FPGA**, where internal registers and program counters were observed using LEDs and on-board debugging methods. Additionally, a **Python-based assembler** was developed to convert human-readable assembly instructions into machine code compatible with the Verilog-based instruction memory, enabling rapid testing of software programs on the custom CPU.

A major enhancement of this project is the successful execution of the entire **ASIC backend flow** using the open-source Qflow toolchain. This includes RTL synthesis (Yosys), standard-

cell placement (GrayWolf), detailed routing (QRouter), layout generation and DRC (Magic), and LVS verification (Netgen). The final layout passed both DRC and LVS checks after minor corrections, demonstrating that the CPU design is **fabrication-ready** and adheres to standard-cell technology constraints.

Overall, this project provides an end-to-end exploration of digital processor design—starting from conceptual architecture and RTL description, moving to FPGA prototyping for functional validation, and finally transitioning to ASIC-level implementation for physical verification. The work not only strengthens theoretical understanding but also offers substantial practical exposure to tools and methodologies used in modern semiconductor design flows.

Chapter 2: Literature Review (Background & Motivation)

The design and analysis of processor architectures have been central to the development of modern computing systems for more than five decades. Early microprocessors such as the **Intel 8080**, **MOS 6502**, and **Zilog Z80** laid the foundation for contemporary CPU design principles, influencing concepts such as register-based architecture, instruction pipelines, interrupt systems, and memory hierarchies. As computing evolved, academic and industry researchers emphasized simplicity, performance, and scalability—leading to the emergence of **Reduced Instruction Set Computer (RISC)** architectures. RISC principles, which prioritize efficient instruction execution and streamlined datapath design, serve as the conceptual basis for many educational processor models, including the one implemented in this project.

A significant body of literature also exists on **single-cycle processor architecture**, which is often introduced to students as the first step toward understanding more advanced multi-cycle and pipelined systems. Textbooks such as *"Computer Organization and Design"* by Patterson & Hennessy describe the single-cycle CPU as an ideal pedagogical model because each instruction completes in a single clock cycle, making the datapath straightforward and the control logic deterministic. This approach simplifies functional verification and provides learners with a transparent view of instruction execution, making it suitable for educational projects.

HDL-based processor implementation, particularly using **Verilog**, has become a standard practice in university-level digital design courses. Prior academic works highlight the importance of modular RTL design, separation of datapath and control, and cycle-accurate simulation. The use of FPGAs for verifying processor functionality is widely documented due to their reconfigurability, real-time execution capability, and compatibility with a variety of debugging techniques. Implementing a CPU on FPGA platforms such as the PYNQ-Z2 provides learners with practical exposure to clocking, I/O handling, resource utilization, and hardware-software co-design.

In parallel, advancements in open-source toolchains have enabled broader accessibility to **ASIC backend flows**, which were once limited to industry environments. Tools such as **Yosys** for synthesis, **GrayWolf** for placement, **QRouter** for detailed routing, **Magic** for layout and DRC, and **Netgen** for LVS verification have been used extensively in academia. The literature

demonstrates that open-source ASIC flow is instrumental for bridging theoretical digital design education with real-world semiconductor design practices.

The motivation behind this project originates from the desire to explore **end-to-end processor design**—from architectural conception and RTL modeling, to FPGA prototyping, and finally to ASIC implementation. This holistic approach provides a deeper understanding of:

- Instruction Set Architecture (ISA) design
- RTL coding discipline
- Datapath and control unit integration
- FPGA-based functional validation
- Physical design constraints in ASIC workflows
- The differences between FPGA logic mapping and ASIC standard-cell synthesis

By combining both FPGA and ASIC workflows, this project aims to deliver a comprehensive learning experience that reflects actual semiconductor industry processes. The opportunity to validate the same CPU through two different hardware realization platforms serves as a strong motivation to understand the full digital design pipeline, making this project academically enriching and practically relevant.

Chapter 3: List of Components and System Working

This chapter presents the complete set of hardware components, software tools, and RTL modules used in the development of the custom 8-bit single-cycle CPU. To provide a clear understanding of the system architecture, a high-level CPU block diagram is shown below. This diagram illustrates the core modules and the flow of instructions and data within the processor.

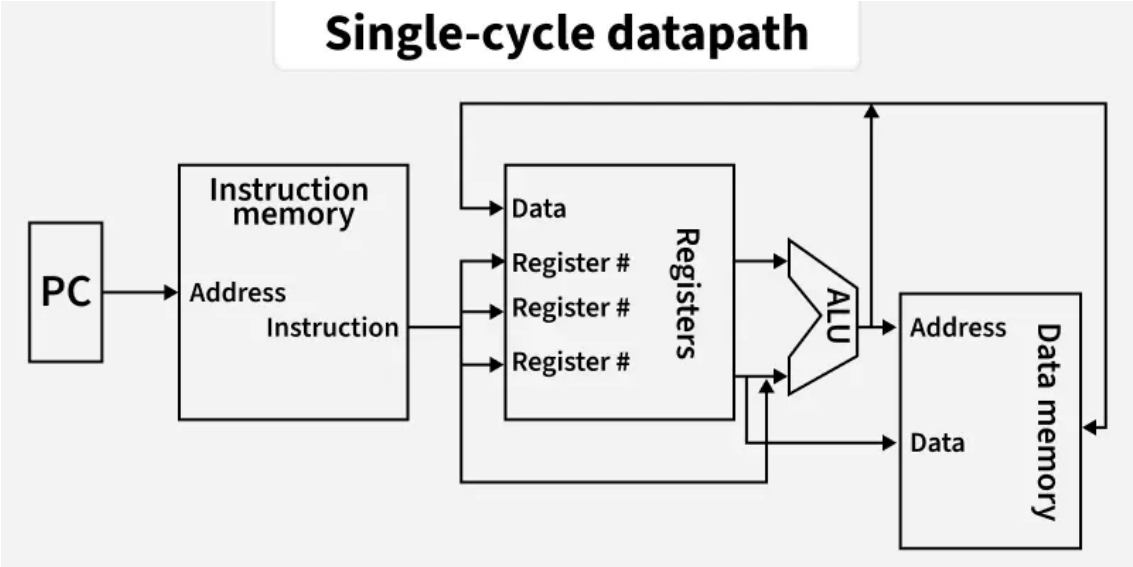

Figure 3.1: High-level architectural block diagram of the designed 8-bit single-cycle CPU.

The block diagram provides an overview of the major datapath components, including the Program Counter (PC), Instruction Memory, Register File, Arithmetic Logic Unit (ALU), Data Memory, and Control Unit, all interconnected to execute instructions in a single clock cycle.

3.1 Hardware Components Used

3.1.1 PYNQ-Z2 FPGA Development Board

The **PYNQ-Z2** is a powerful FPGA development platform designed for digital system design, embedded systems, and hardware–software co-design. It is built around the **Xilinx Zynq-7000 SoC**, which integrates a **dual-core ARM Cortex-A9 processor** with **programmable logic (PL)** on a single chip. This combination makes it suitable for both high-level application development and low-level hardware acceleration.

In this project, the PYNQ-Z2 board was used to **prototype and validate the custom 8-bit CPU** implemented in Verilog HDL. Internal CPU signals such as the Program Counter (PC),

register values (R0–R3), and Zero Flag were mapped to the board's **user LEDs** for real-time debugging and visualization. The board's hardware resources provided a convenient and reliable platform for testing the functional correctness of the CPU before performing ASIC backend implementation.

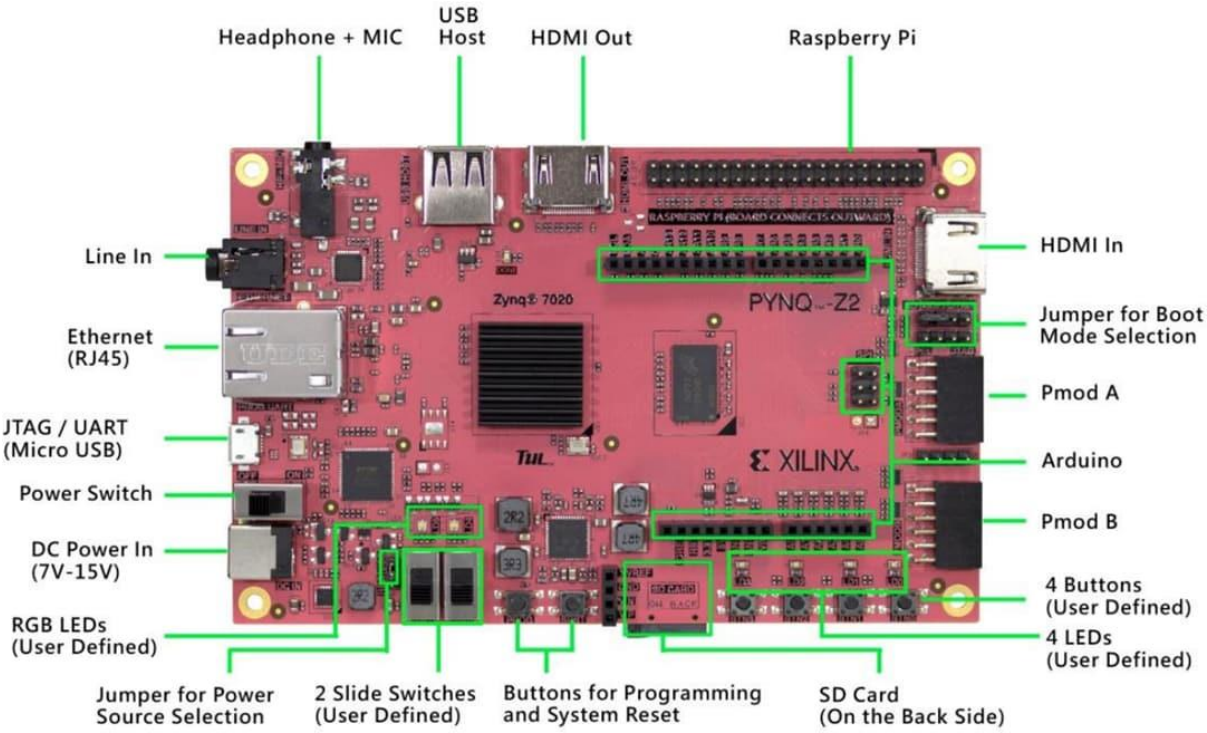

Figure 3.1.1.1: Pinout Description of PYNQ - Z2

Key Features of the PYNQ-Z2

Table 3.1.1: Description of PYNQ-Z2

Feature Category	Description
SoC/FPGA Device	Xilinx **Zynq-7020 (XC7Z020-1CLG400C)**
Processor (PS)	Dual-core **ARM Cortex-A9**, 650 MHz
Programmable Logic (PL)	Artix-7 FPGA fabric with 85K logic cells
Memory	512 MB DDR3 (connected to PS)
On-board Storage	MicroSD card slot (for booting PYNQ Linux)
Clock Sources	125 MHz system clock, 33.3 MHz oscillator

User I/O	2×6 user switches, **4 push buttons**, **4 RGB LEDs**, **4 LED indicators**
Pmod Interfaces	2 × 12-pin Pmod headers for external modules
Arduino-Compatible Header	Allows interfacing with Arduino shields
Audio Support	On-board audio codec (Line in/out, mic)
Video Interfaces	HDMI input & HDMI output ports
Communication Interfaces	USB-UART, Ethernet (10/100 Mbps)
Programming Interface	Micro-USB JTAG (via Digilent SMT2)
Power Options	12V DC jack or USB power
Supported Tools	Xilinx Vivado, PYNQ framework, Jupyter notebooks
Primary Use in This Project	FPGA prototyping of 8-bit CPU, mapping registers/PC/flags to LEDs

3.1.2 LED Indicators

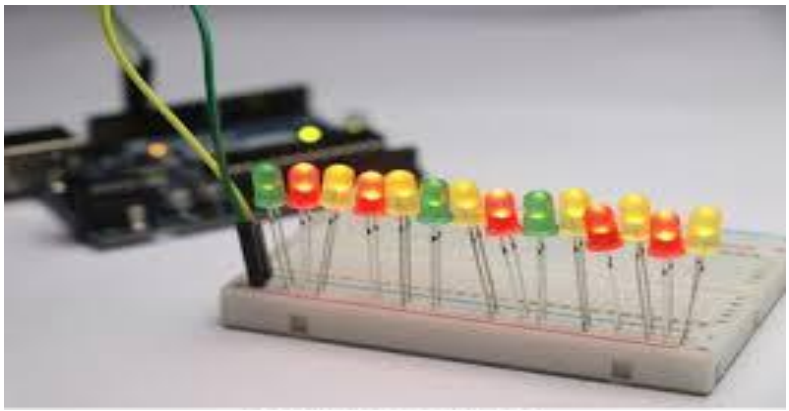

Figure 3.1.2.1: 2.5mm LEDs

In this project, **external 2.5 mm LEDs** were used as visual indicators to display internal CPU signals during hardware testing on the PYNQ-Z2 FPGA board. These LEDs were interfaced

through the board's GPIO pins and were driven by the output signals from the CPU, including a clear and intuitive way to observe changes in CPU state in real time, enabling step-by-step the **Program Counter (PC)**, **register values (R0–R3)**, and the **Zero Flag**. The LEDs provided debugging and verification of instruction execution.

The 2.5 mm LEDs were chosen due to their **low power consumption**, **sufficient brightness**, and **compatibility with the FPGA's 3.3V logic levels**. Appropriate current-limiting resistors (typically 220Ω–330Ω) were used to ensure safe operation. By mapping specific bits of internal CPU registers to individual LEDs, the team was able to visually verify operations such as arithmetic updates, branching decisions, memory load/store activity, and PC transitions on every clock cycle.

This simple yet effective hardware demonstration method greatly enhanced the understanding of CPU operation and allowed the verification of test programs without the need for complex debugging tools or external displays.

3.1.3 Micro-USB Cable & Power Supply

Figure 3.1.3.1: USB to type-B adapter

A Micro-USB cable and standard power supply were used as essential hardware components for programming and powering the PYNQ-Z2 FPGA development board during the implementation and testing of the 8-bit CPU. The Micro-USB interface serves two primary purposes:

1. FPGA Programming.
2. Power Delivery

3.2 Software & EDA Tools Used

This project utilizes multiple design tools spanning RTL simulation, FPGA development, and ASIC backend implementation.

3.2.1 Vivado Design Suite (Xilinx)

The **Vivado Design Suite** by Xilinx is the primary development environment used for synthesizing, implementing, and testing the Verilog-based 8-bit CPU on the **PYNQ-Z2 FPGA board**. Vivado provides a comprehensive toolchain for digital hardware development, offering integrated support for HDL simulation, constraint management, synthesis optimization, timing analysis, and FPGA bitstream generation.

In this project, Vivado played a crucial role in the **front-end design verification and physical implementation** of the CPU. All Verilog RTL modules—including the Program Counter, Register File, ALU, Control Unit, Data Memory, and top-level FPGA wrapper—were synthesized and validated using Vivado's logic synthesis engine. Following synthesis, the implementation stage mapped the design onto the programmable logic fabric of the PYNQ-Z2, ensuring that resource utilization, routing, and timing constraints were satisfied.

Vivado's **hardware manager** allowed seamless communication with the FPGA through the Micro-USB JTAG interface, enabling rapid downloading of the bitstream and real-time testing of the CPU. By mapping internal CPU signals (PC, R0–R3, Zero Flag) to on-board LEDs, Vivado facilitated easy visualization and debugging of processor functionality.

Key Roles of Vivado in This Project

- HDL synthesis of all CPU modules
- Implementation and routing on the PYNQ-Z2 FPGA
- Timing constraint verification
- Bitstream generation (.bit file)
- On-board testing using the Hardware Manager
- Debugging through LED/GPIO mapping

Vivado's rich feature set and tight integration with Xilinx FPGAs made it an essential tool for validating the CPU design prior to ASIC implementation. The use of Vivado ensured a robust

and repeatable development workflow, bridging the gap between simulation and real hardware behavior.

3.2.2 Qflow ASIC Toolchain

The **Qflow ASIC Toolchain** is an open-source digital design flow used to convert Verilog RTL code into a fully placed, routed, and verified ASIC layout. It integrates several specialized EDA tools that collectively implement the complete backend process required for standard-cell based chip design. In this project, Qflow was used to take the 8-bit CPU from a high-level RTL description to a fabrication-ready physical layout.

The Qflow pipeline consists of the following tools:

1. **Yosys - RTL Synthesis**

   Yosys performs logic synthesis by converting Verilog RTL into a gate-level netlist composed of standard cells from the OSU035 or similar library. It checks design hierarchy, resolves module dependencies, and generates a structural description optimized for area and logic depth.

   Role in this project:

- Synthesized all CPU modules (PC, ALU, RF, DM, control unit)
- Produced a technology-mapped gate-level netlist
- Ensured the RTL was ASIC-compatible

2. **GrayWolf -Standard Cell Placement**

   GrayWolf arranges synthesized standard cells within the chip's floorplan. It optimizes cell positions to minimize wire length and improve timing.

   Role in this project:

- Successfully placed 1000+ standard cells
- Generated .def and .pl placement files for routing

3. **QRouter-Signal Routing**

   QRouter performs detailed routing to connect all signal nets using multi-layer metal interconnect.

   Role in this project:

- Routed over 7000 nets with zero routing failures
- Generated .def and .rc routed layout files
- Ensured electrical connectivity and compliance with design rules

4. **Magic -Layout Generation & DRC**

Magic is used for both layout visualization and Design Rule Checking (DRC). It links placement and routing output to create a complete mask-level layout.

Role in this project:

- Produced the final layout file top_asic.mag
- Performed DRC checks
- Only 3 initial errors were found and fixed manually
- Final layout passed with 0 DRC violations

5. **Netgen — LVS Verification**

   Netgen performs Layout vs Schematic (LVS) checks to ensure that the physical layout matches the original synthesized schematic.

   Role in this project:

- Verified gate-level netlist vs. layout
- Final result: “Circuits Match”
- Confirmed correctness of ASIC implementation

3.3 RTL Modules

The Register-Transfer Level (RTL) modules form the core of the CPU architecture. Each module is designed in Verilog HDL using a modular, hierarchical approach. The CPU comprises multiple interacting RTL blocks, each responsible for a specific function in the instruction execution cycle. The major RTL modules used in this project include:

- Program Counter (PC)
- Instruction Memory (IM)
- Register File (RF)
- Arithmetic Logic Unit (ALU)
- Data Memory (DM)
- Control Unit
- Top-level Wrapper Modules (FPGA and ASIC versions)

These RTL modules collectively form the single-cycle datapath, enabling the CPU to fetch, decode, execute, and complete an instruction within one clock cycle.

3.3.1 Program Counter (PC)

The Program Counter (PC) is an **8-bit register** responsible for holding the address of the next instruction to be executed by the CPU. In each clock cycle, the PC either increments sequentially or loads a new address depending on the type of instruction.

Under normal operation, the PC updates as:

PC ← PC + 1, enabling continuous instruction flow.

For control-transfer instructions such as **JMP** (unconditional jump) and **JZ** (conditional jump if Zero Flag = 1), the PC loads the **target address** encoded within the instruction, allowing non-linear program execution.

The PC is reset to **0x00** during system initialization and updates synchronously with the clock. As a core part of the datapath, the Program Counter ensures correct instruction sequencing and serves as the driving element for the instruction fetch stage. It was fully verified through simulation, FPGA testing, and ASIC synthesis, confirming proper sequential and branch execution behavior.

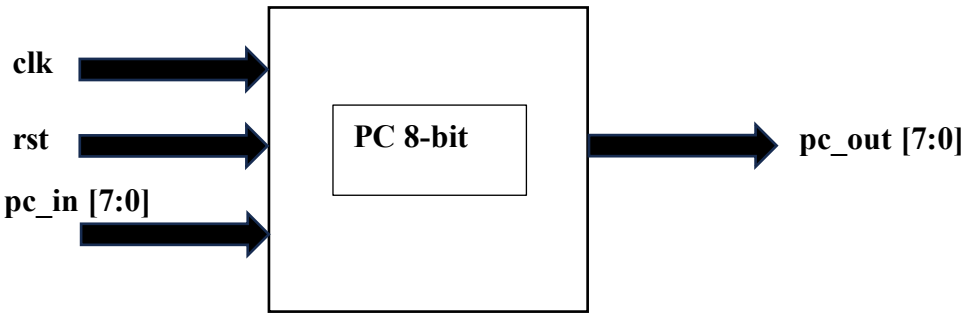

Figure 3.3.1.1: Block Diagram of Program Counter

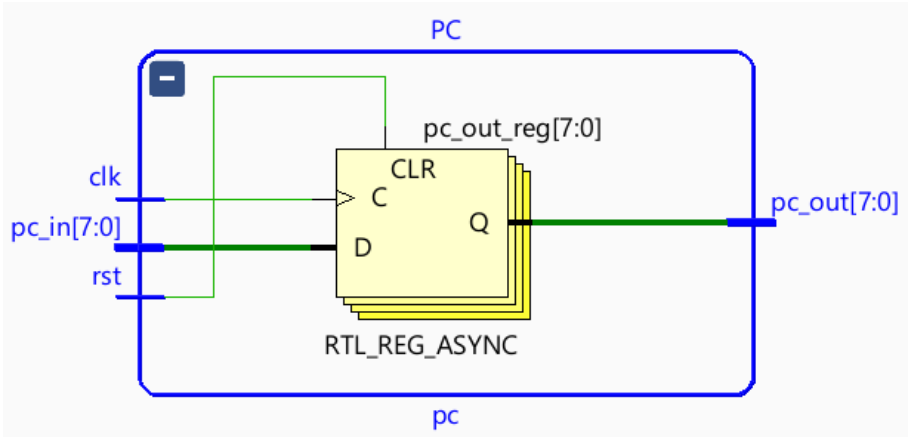

Figure 3.3.1.2: RTL Schematic of PC

3.3.2 Register File (RF)

The Register File is a compact storage unit consisting of **four 8-bit registers (R0–R3)** used for storing temporary data and operands. It provides **two combinational read ports** for accessing the source registers and a **synchronous write port** for updating the destination register during

the write-back stage. a central role in arithmetic, logical, and memory operations by supplying input values to the ALU. For FPGA debugging, the internal register values are also exposed to LEDs. The module was successfully verified during simulation, FPGA testing, and ASIC synthesis.

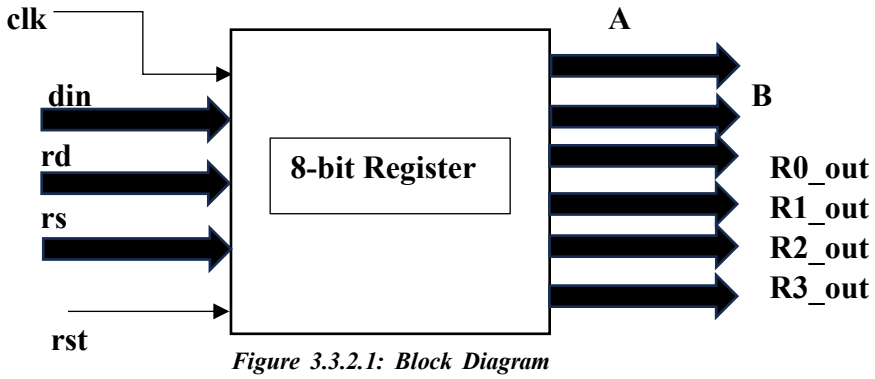

Figure 3.3.2.1: Block Diagram of 8-bit Register

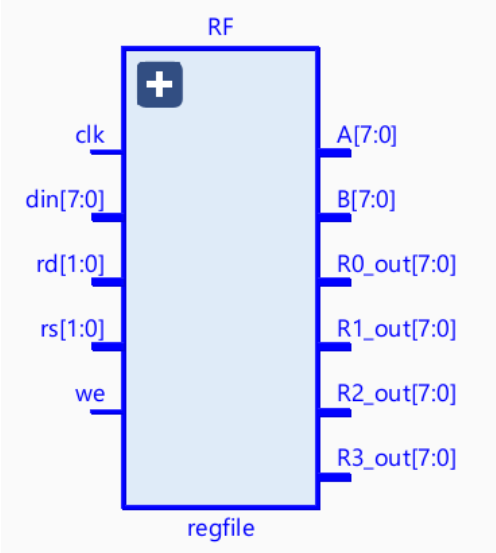

Figure 3.3.2.2: RTL Schematic of Register

3.3.3 Arithematic and Logic Unit (ALU)

The Arithmetic Logic Unit performs the core computational tasks of the CPU. It supports **four operations**: ADD, SUB, AND, and OR, selected using a 2-bit ALU control signal. The ALU outputs an 8-bit result and asserts a **Zero Flag** when the result is zero, enabling conditional branching. Being a purely combinational block, the ALU executes its operation within the same cycle and directly contributes to the single-cycle nature of the processor. It has been fully validated in simulation and hardware.

Figure 3.3.3.1: Block Diagram of 8-bit ALU

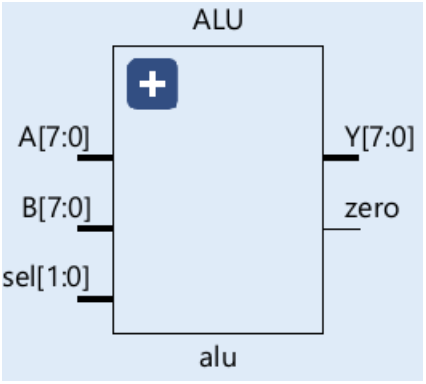

Figure 3.3.3.2: RTL Schematic of ALU

3.3.4 Instruction Memory (IM)

The Instruction Memory is an **8-bit wide, 256-location ROM** that stores the machine code program executed by the CPU. It is addressed by the Program Counter and returns the corresponding instruction during each fetch cycle. The memory is initialized using a HEX file generated by the custom Python assembler. As a read-only module, it ensures stable and consistent instruction flow throughout program execution.

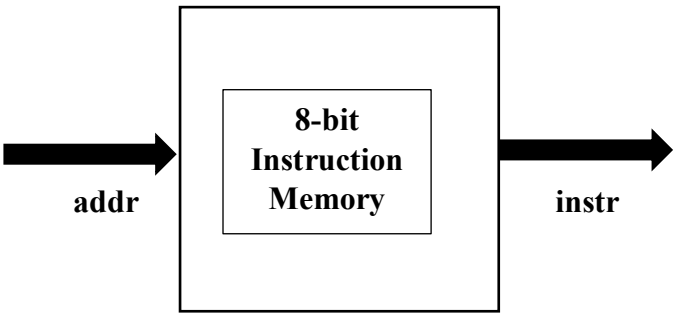

Figure 3.3.4.1: Block Diagram of Instruction Memory

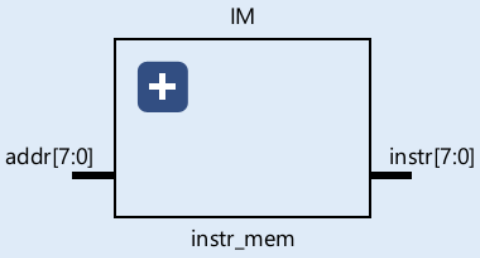

Figure 3.3.4.2: RTL Schematic of IM

3.3.5 Data Memory (DM)

The Data Memory is a small **16×8-bit RAM** used to support LOAD and STORE instructions. It receives a 4-bit address formed from instruction fields and performs either a read (for LOAD) or a write (for STORE). Reads are combinational while writes occur on the rising edge of the clock. The module enables basic data handling within the CPU and was validated through simulation and FPGA execution.

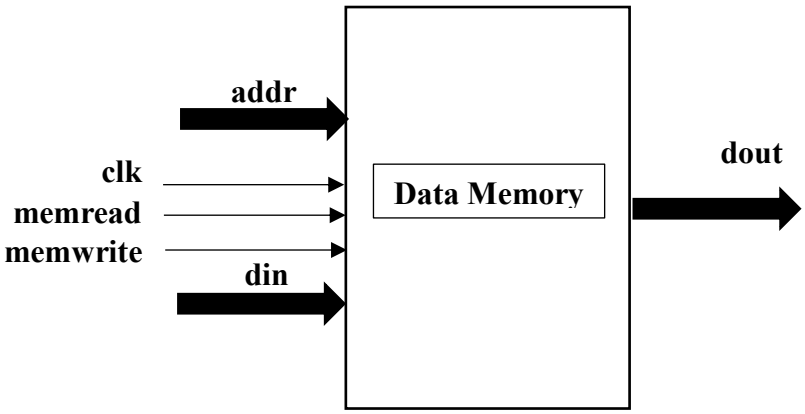

Figure 3.3.5.1: Block Diagram of Data Memory

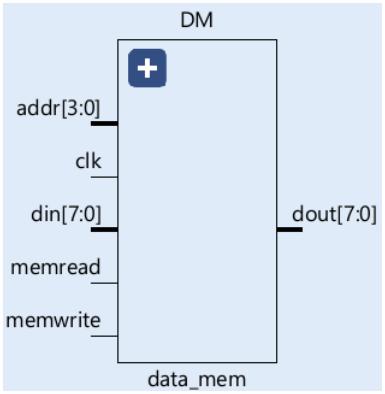

Figure 3.3.5.2: RTL Schematic of data memory

3.3.6 Control Unit (CU)

The Control Unit decodes the instruction fields—particularly the opcode—and generates the control signals required for correct CPU operation. These include register write enable, memory read/write enables, ALU operation select, and branch control signals. The CU ensures proper coordination among all modules, allowing the CPU to fetch, decode, execute, and commit results in a single cycle. Its correctness was confirmed through simulation, FPGA testing, and ASIC synthesis.

3.3.7 Clock Downscaler

The Clock Divider module is responsible for converting the **12 MHz input clock** of the FPGA into a much slower **1 Hz clock signal** suitable for human-perceptible observation of CPU behavior through LEDs and external components. This module uses a counter-based approach, where an internal register increments on every rising edge of the input clock. Once the counter reaches a predetermined terminal count corresponding to 12 million cycles, it toggles the output clock, effectively generating a 1 Hz signal.

By reducing the clock frequency, the Clock Divider enables each instruction cycle of the CPU to be visually monitored on the FPGA through LED updates, allowing clear demonstration and step-by-step debugging. This module is particularly useful for educational purposes, as it slows down instruction execution to a human-observable rate. The Clock Divider is fully synthesizable and compatible with both FPGA workflows and simulation environments.

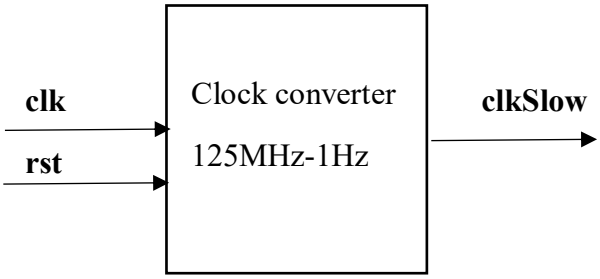

Figure 3.3.7.1: Block Diagram of Clock Converter

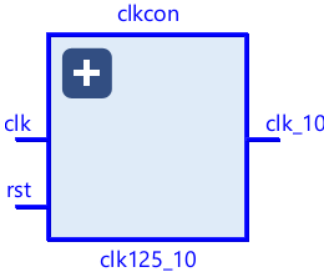

Figure 3.3.7.2: RTL Schematic of Clock Converter

3.3.8 CPU

The cpu.v module serves as the **top-level integration block** of the 8-bit single-cycle processor. It instantiates and interconnects all major RTL components, including the Program Counter, Instruction Memory, Register File, ALU, Data Memory, and Control Unit. This module defines the complete datapath and control flow required for instruction execution, coordinating how operands are fetched, processed, and written back within a single clock cycle.

The CPU module is responsible for generating all internal control signals, selecting ALU operations, forming memory addresses, managing conditional and unconditional branch operations, and determining the next Program Counter value. It also updates the Zero Flag based on ALU output results. In addition, the design exposes internal signals such as PC, register contents, and status flags for debugging on the FPGA platform.

The cpu.v module acts as the functional core of the entire processor system and was validated through simulation, FPGA implementation on the PYNQ-Z2 board, and ASIC synthesis using the Qflow toolchain. Its correct integration ensures reliable execution of the custom instruction set architecture designed for this project.

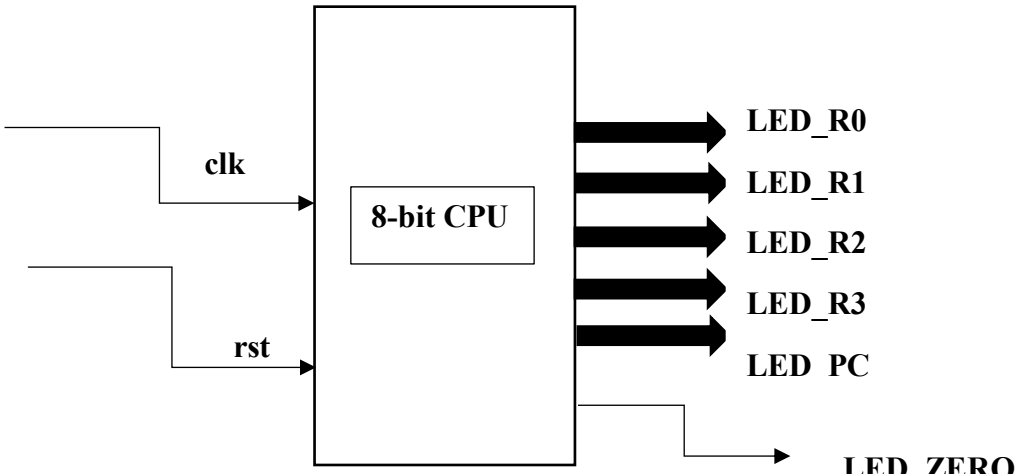

Figure 3.3.8.1: Block Diagram of CPU

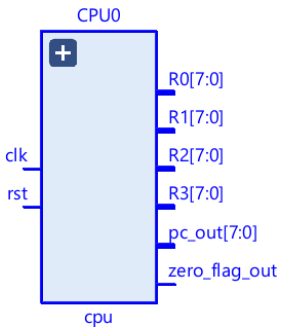

Figure 3.3.8.2: RTL Schematic of CPU

3.3.9 ASIC and FPGA Top-Level Wrappers

The FPGA wrapper *(top_fpga.v)* integrates the CPU with the hardware resources available on the **PYNQ-Z2 board**. It provides the necessary signal mappings to physical I/O components such as LEDs, push buttons, GPIOs, and the board clock. Internal CPU signals—including the Program Counter, register values, and Zero Flag—are routed to LEDs for real-time visualization and debugging.

This wrapper ensures compatibility with **Vivado**, enabling synthesis, place-and-route, bitstream generation, and on-board execution. The FPGA top module is primarily used for functional verification, educational demonstration, and step-by-step CPU behavior observation.

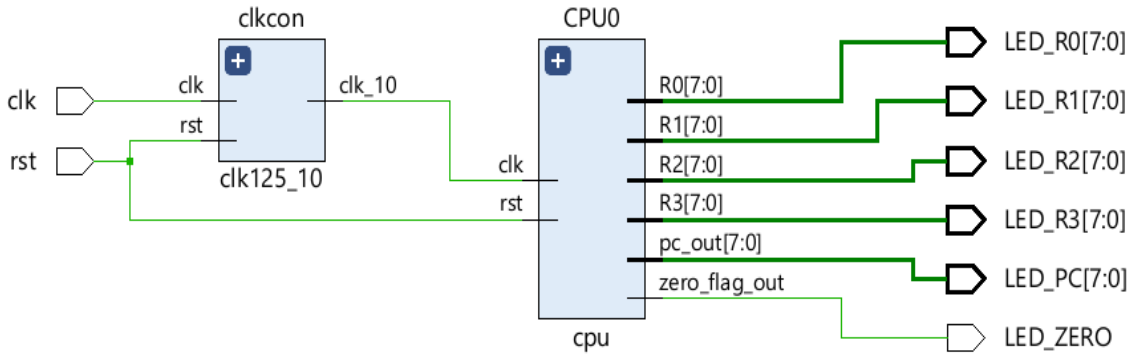

Figure 3.3.9.1: Top FPGA wrapper

The ASIC wrapper is designed specifically for the **Qflow open-source ASIC toolchain**, ensuring clean and fully synthesizable RTL for the backend process. It removes all FPGA-specific constructs, such as board-level I/O mappings, and exposes only the essential clock, reset, and CPU interfaces required by synthesis tools.

This wrapper enables smooth execution of the complete ASIC flow, including:

- **RTL Synthesis (Yosys)**
- **Standard-Cell Placement (GrayWolf)**
- **Routing (QRouter)**
- **Layout Generation & DRC (Magic)**
- **LVS Verification (Netgen)**

The ASIC wrapper forms the entry point for physical design and layout creation. Subsequent chapters will include the **placement results, routing diagrams, Magic layout screenshots, DRC reports, and LVS logs** generated using the top_asic.v module.

Figure 3.3.9.2: LVS log, Performed under Qflow

Figure 3.3.9.3: DRC log, on Qflow

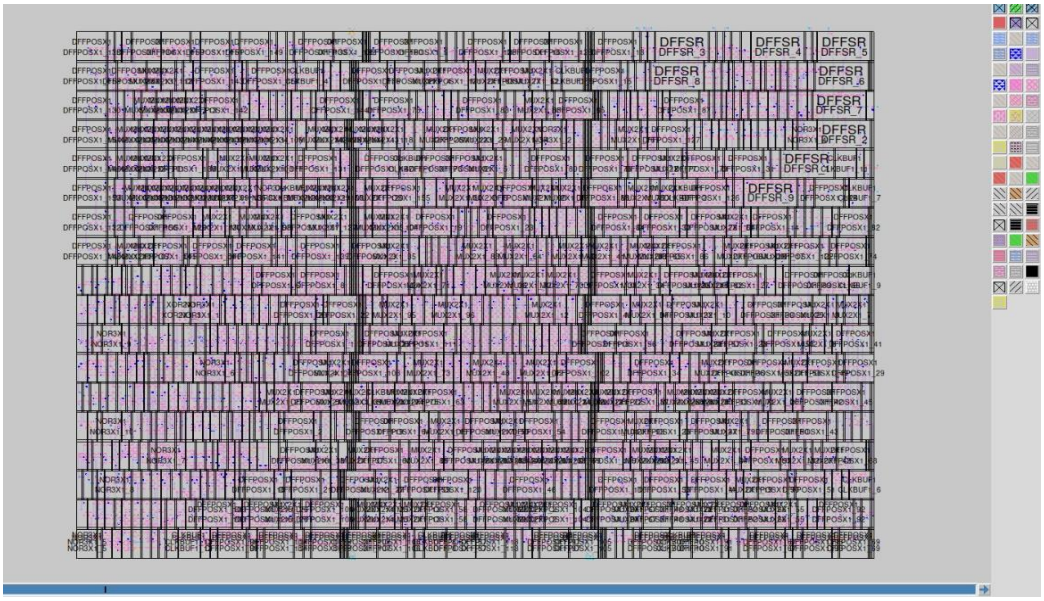

Figure 3.3.9.4: Layout of Top Module on QFlow

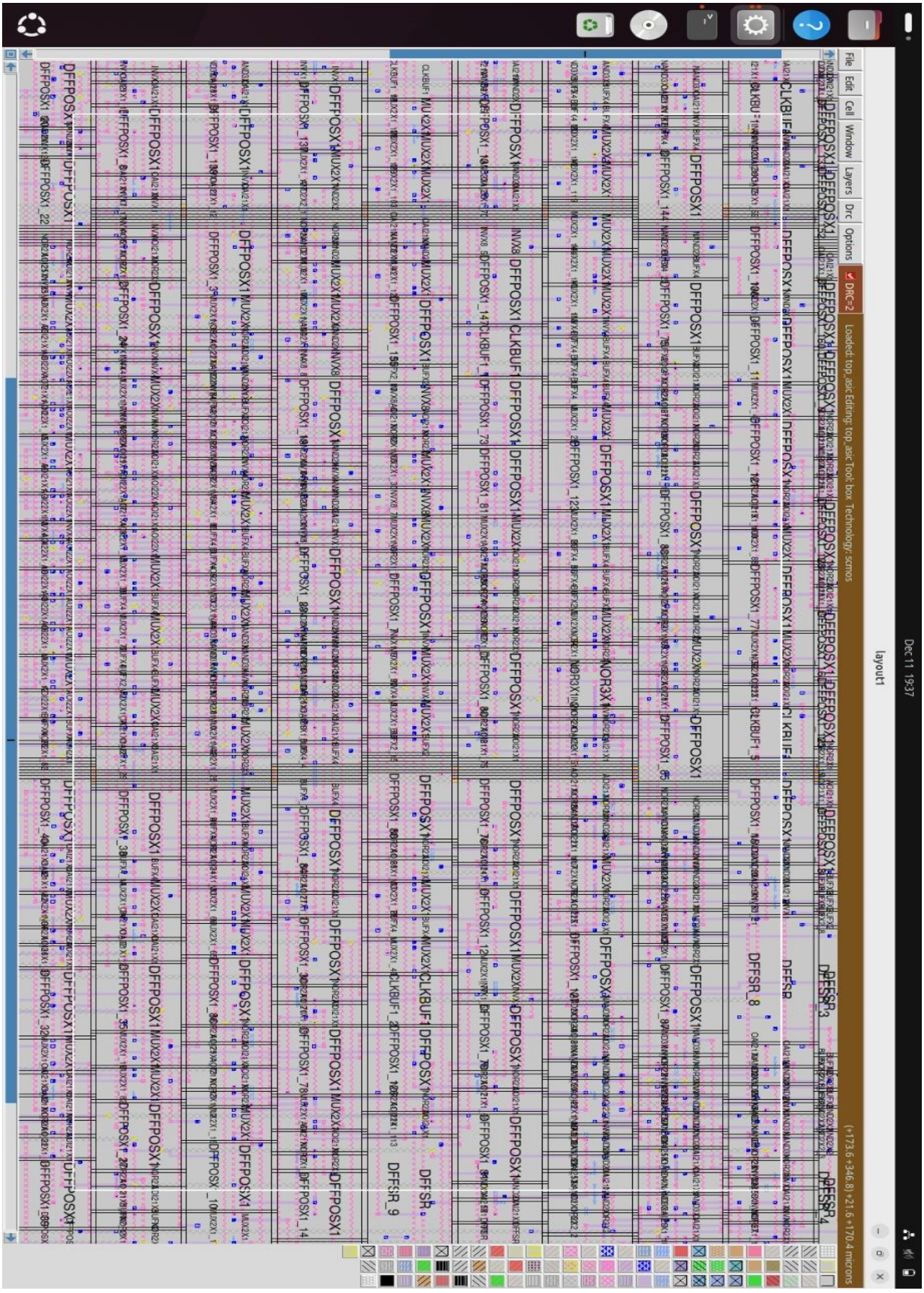

Figure 3.3.9.5: Zoom out View of Layout on Qflow

Chapter 4: Applications, Benefits to Society

The design and implementation of a custom 8-bit CPU using Verilog HDL, FPGA prototyping, and ASIC backend flow represents an important contribution to the fields of digital system design, embedded systems, and semiconductor technology. Although the processor developed in this project is educational in scale, it illustrates the fundamental principles behind modern computing devices and supports a wide range of academic, industrial, and societal applications.

4.1 Applications

4.1.1 Educational and Academic Use

The developed CPU serves as an effective teaching tool for students studying **digital electronics, computer architecture, VLSI design, and embedded systems**. Its modular design allows learners to understand core concepts such as datapath design, ALU operations, instruction decoding, memory access, and control logic. By implementing the design on FPGA and through ASIC tools, the project enables hands-on exposure to both front-end and back-end semiconductor workflows.

4.1.2 FPGA-Based Prototyping Systems

The CPU can be deployed on FPGA boards like the PYNQ-Z2 for rapid prototyping of embedded applications. Such small-scale processors are commonly used in low-power systems, testing environments, sensor data processing, and hardware-software co-design platforms. FPGA implementation allows quick modification of the instruction set and architecture for experimental research.

4.1.3 Custom Embedded Controllers

With minor extensions, the CPU can serve as the core of a **microcontroller** for simple automation tasks, robotics applications, IoT nodes, educational kits, and low-complexity signal processing systems. These applications do not require the performance of full-scale processors but benefit from customizable and resource-efficient architectures.

4.1.4 Research in Processor Design and Optimization

The CPU architecture provides a platform for exploring improvements such as pipelining, hazard detection, cache integration, and low-power design techniques. Such experimentation helps researchers understand the effects of architectural modifications on performance, area, and power.

4.1.5 ASIC Design Training and Fabrication Preparation

The transformation of the CPU into a layout-ready ASIC design using Qflow demonstrates its suitability for fabrication in academic multi-project wafer (MPW) programs. Students can use the flow to gain practical experience with standard-cell design, placement, routing, DRC, and LVS verification.

4.2 Benefits to Society

4.2.1 Strengthening Technical Education

The project enhances engineering education by bridging theory and practice. It equips students with skills relevant to **semiconductor design**, a rapidly growing industry critical to national technological development. By working through FPGA prototyping and ASIC backend tools, learners gain job-ready experience.

4.2.2 Promoting Self-Reliance in Semiconductor Technology

Hands-on exposure to processor design contributes to India's initiative towards **Atmanirbhar Bharat** in electronics and chip design. Training students in ASIC flows supports the long-term growth of domestic semiconductor capabilities.

4.3.3 Enabling Low-Cost Educational Tools

Custom CPUs like this can be integrated into affordable learning kits for schools, colleges, and skill development programs. Such tools demystify computing concepts and inspire the next generation of engineers.

4.3.4 Supporting Innovation in Embedded Systems

Understanding how processors work at the hardware level empowers students and developers to create innovative embedded solutions for healthcare, agriculture, automation, smart cities, and IoT devices. These sectors have a direct positive impact on societal development.

4.3.5 Encouraging Research and Development

This project fosters an environment of curiosity, experimentation, and problem-solving. Students trained in CPU and ASIC design contribute to the national pool of engineers capable of advancing cutting-edge technologies in AI accelerators, communication chips, and specialized processors.

Chapter 5: Conclusion & Future Scope

5.1 Conclusion

The design and implementation of an 8-bit single-cycle CPU using Verilog HDL, FPGA prototyping, and open-source ASIC backend tools has provided a comprehensive, end-to-end exploration of modern digital system design. Beginning with the formulation of a custom instruction set architecture (ISA), followed by the development of each RTL module—including the Program Counter, Register File, ALU, Data Memory, and Control Unit—the project successfully integrated all components into a coherent and fully functional processor.

Through simulation and FPGA deployment on the PYNQ-Z2 board, the CPU's behavior was validated in real hardware. Mapping internal signals to LEDs enabled intuitive observation of instruction execution, strengthening understanding of datapath operations and control flow. The development and use of a Python-based assembler further enhanced the testing workflow, enabling rapid program loading and verification.

A major highlight of the project is the successful execution of the complete **ASIC backend flow** using the Qflow toolchain. The processor was synthesized using Yosys, placed with GrayWolf, routed through QRouter, and verified for design rule compliance using Magic. The final Layout vs. Schematic (LVS) check using Netgen confirmed that the physical layout faithfully matches the synthesized schematic. This end-to-end process demonstrates that the CPU design is not only theoretically sound but also **fabrication-ready**, meeting core criteria for standard-cell ASIC design.

Overall, this project bridges the gap between academic study and practical engineering experience, offering valuable exposure to the complete lifecycle of processor development—from architectural conception to hardware realization and physical layout preparation. It highlights the feasibility, flexibility, and educational impact of designing custom processors using accessible open-source tools.

5.2 Future Scope

While the present CPU design is compact and optimized for learning and demonstration, it provides a strong foundation for future enhancements and research directions. Several avenues exist for extending its capabilities:

5.2.1 Multi-Cycle and Pipelined Architectures

The existing single-cycle design can be evolved into a multi-cycle or pipelined CPU to improve performance and introduce advanced architectural concepts such as hazard detection, forwarding, and pipeline control units.

5.2.2 Expanded Instruction Set Architecture (ISA)

Additional instructions—such as shift/rotate operations, compare instructions, stack operations, and immediate arithmetic—can broaden the functional capabilities of the CPU.

5.2.3 Integration of Interrupt Handling

Adding an interrupt controller would enable real-time event-driven processing, bringing the architecture closer to microcontroller-level capabilities.

5.2.4 Memory Expansion and Peripheral Interface

Connecting external RAM, ROM, UART, SPI, or I2C peripherals would enable the processor to interact with external devices, making it suitable for embedded applications.

5.2.5 Optimization for Low Power and Area

Physical design optimizations such as clock gating, logic minimization, and custom cell development could make the CPU more resource-efficient, enabling deployment in low-power IoT environments.

5.2.6 Fabrication Through MPW Programs

With the layout already DRC/LVS clean, the design can be submitted to university-level or open-source fabrication programs (such as Google SkyWater MPW), allowing students to create their own silicon chips.

5.2.7 High-Level System Integration

The CPU can be integrated into a larger SoC with memory controllers, accelerators, or programmable peripherals, providing experience with system-level VLSI design.

References

1. Patterson, D. A., & Hennessy, J. L., *Computer Organization and Design: The Hardware/Software Interface*, 5th Edition, Morgan Kaufmann, 2014.

2. Mano, M. M., & Ciletti, M. D., *Digital Design*, 5th Edition, Pearson Education, 2013.
3. Brown, S., & Vranesic, Z., *Fundamentals of Digital Logic with Verilog Design*, 3rd Edition, McGraw-Hill, 2014.

4. Bhasker, J., *Verilog HDL: A Guide to Digital Design and Synthesis*, 2nd Edition, Prentice Hall, 2001.

5. Xilinx Inc., *PYNQ-Z2 Board Reference Manual*, Digilent Inc., 2019

6. Xilinx Inc., *Vivado Design Suite User Guide*, Version 2020.2.

7. YosysHQ, *Yosys Open SYnthesis Suite Documentation*, Available at: https://yosyshq.net/yosys

8. Edwards, T., *Qflow: Open-Source VLSI Design Flow Documentation* (including GrayWolf, QRouter, Magic, Netgen), Open Circuit Design.

9. Edwards, T., *Magic VLSI Layout Tool Documentation*, Open Circuit Design, 2020.

10. Harris, D. M., & Harris, S. L., *Digital Design and Computer Architecture*, 2nd Edition, Morgan Kaufmann, 2012.

11. Weste, N. H. E., & Harris, D., *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th Edition, Pearson, 2010.

12. PYNQ Development Team, *PYNQ Documentation*, Available at: https://pynq.io

13. IEEE Standards Association, *IEEE Standard for Verilog Hardware Description Language*, IEEE Std 1364-2005.