Mini-CPU Project

Implementation of an 8-Bit RISC CPU (45 nm CMOS Technology)

Course: EE4620/6620 — VLSI Design | Instructor: Dr. Saiyu Ren | October 2025
Keerthi Patil, Dept. of Electrical Engineering, Wright State University, patil.115@wright.edu

---

I. INTRODUCTION

Our project focuses on implementing an 8-bit RISC CPU using the FreePDK45 45 nm CMOS technology node. The CPU is the heart of a computing system because it executes instructions, processes data, and coordinates the operation of other hardware components. RISC architectures are widely used in the semiconductor industry due to their simple instruction format, efficiency, and scalability. Many leading companies, including NVIDIA, Arm, Intel, AMD, and Apple, rely on RISC-based processors to achieve high performance with lower design complexity. Unlike CISC architectures, which depend on large and complex instruction sets, RISC uses a smaller, fixed-length instruction format. This simplifies control logic, supports faster execution, and provides a clear framework for understanding processor fundamentals.

Our implementation adopts an accumulator-based architecture with a minimal instruction set, offering a focused and practical design scope without the overhead of a more complex processor. This approach helps reinforce core concepts in datapath and control design. The CPU is implemented at the transistor level rather than using automated synthesis tools or standard cell libraries. This allows for a deeper understanding of how circuit behavior, timing, and architectural decisions interact in real designs. The processor includes essential components such as an ALU, accumulator, instruction register, program counter, decoder, memory, multiplexer, I/O buffer, and a ring oscillator for clock generation. The project team is divided into three roles: Rob Pierce serves as the Architect and ISA Owner, Vadde Ramya Jyothi focuses on verification and tools, and I, Keerthi Patil, am responsible for RTL design and datapath implementation.

II. 8-BIT RISC PROCESSOR ARCHITECTURE

The processor is an 8-bit single-cycle RISC CPU intended to be designed using FreePDK45 45 nm CMOS technology. The objective of the design is to keep the control logic simple, minimize hardware usage, and maintain a clear instruction flow. The architecture is based on a fixed instruction format, a single accumulator, and a basic bus structure for data transfer. All instructions are planned to execute within a single oscillator cycle, making the overall design straightforward to implement, verify, and analyze.

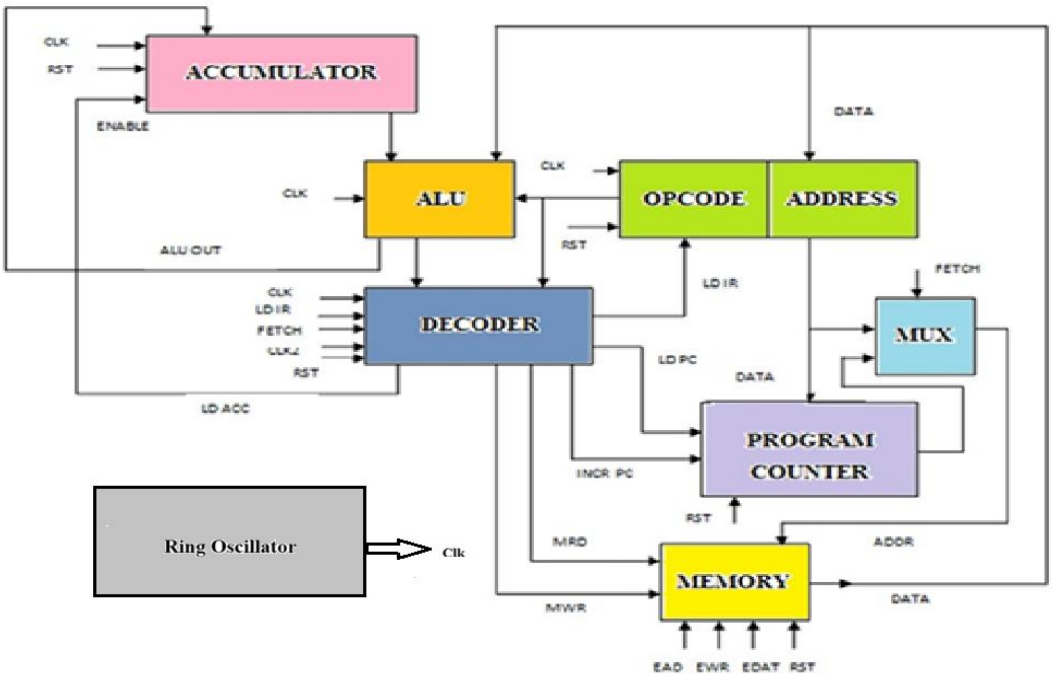

Figure 1: Block diagram of the 8-bit RISC CPU

1. ISA (INSTRUCTION SET ARCHITECTURE):

This processor follows the Von Neumann architecture, where both instructions and data share the same memory and bus. This approach simplifies the design and makes it more efficient for a small RISC CPU.

1. ISA Width: The instruction size is 8 bits, which is enough to define operations and address memory.

2. Instruction Format:

Opcode (3 bits): Defines the operation.
Operand (5 bits): Holds the memory address or data.
This supports 8 instructions and direct addressing of 32 memory locations.

3. Registers:

Accumulator(A) (8-bit) is used for all arithmetic and logic operations.
Zero Flag (ZR) is set when the ALU result is zero and is used for conditional instructions.

Table -1: Operation Based on Opcode

Opcode	Instruction	Operation Description
000	HALT	Stop program execution
001	SKIP ZERO	Skip next instruction if accumulator =0
010	ADD	Add memory operand to accumulator
011	AND	Logical AND with accumulator
100	XOR	Logical XOR with accumulator
101	LOAD	Load accumulator from memory
110	STORE	Store accumulator to memory
111	JUMP	Jump to specified address

4. Addressing Modes:

The processor uses direct addressing. The 5-bit operand directly points to the memory location. The accumulator acts as the main register for most operations.

5. Operation Based on Opcode:

The 3-bit opcode defines the operation carried out by the processor. It includes basic arithmetic, logic, memory access, and control instructions such as HALT, SKZ, ADD, AND, XOR, LOAD, STORE, and JUMP. These opcodes control the execution steps and direct data movement between blocks.

2. MICROARCHITECTURE

1.Execution Model

The CPU works on a single-cycle execution model. In one oscillator cycle, the instruction is fetched, decoded, and executed. A ring oscillator is used to generate the clock signal inside the chip, so there is no need for an external clock. This makes the timing easier to control.

2.Bus Architecture

All the blocks in the CPU are connected using a common bus.

Address Bus (5-bit): Carries the instruction or data address from PC/IR to memory.

Data Bus (8-bit): Transfers data between ALU, accumulator, memory, and I/O.

Control Bus: Sends timing and control signals to make all blocks work together.

3.Functional Blocks:

The CPU is built with a modular design and consists of nine core functional blocks.

i. Ring Oscillator – Acts as the internal clock source, providing precise timing signals that keep all CPU operations synchronized.
ii. Instruction Register (IR) – Temporarily holds the fetched instruction and splits it into the opcode and address for proper decoding.
iii. Accumulator – Serves as a working register that stores intermediate values and final results, helping reduce repeated memory access.
iv. Arithmetic Logic Unit (ALU) – Executes arithmetic and logic operations and updates the zero flag to support conditional branching.
v. Memory – A compact 8-bit, 32-word memory space that stores both instructions and data required for processing.
vi. Program Counter (PC) – Keeps track of the next instruction's address and updates automatically after each cycle or during jumps.
vii. Decoder & Control Unit – Interprets the opcode and generates control signals to coordinate the actions of all components.
viii. Multiplexer (MUX) – Chooses between PC and IR address outputs, controlling how data flows during instruction fetch and execution.
ix. I/O Buffer – Provides temporary data storage to manage smooth transfers between the CPU and external input/output devices.

4.Fetch–Decode–Execute Cycle

Fetch: The program counter (PC) sends the current address to memory, and that instruction is placed into the instruction register (IR).
Decode: The decoder reads the opcode bits and prepares the required control signals for the next stage.
Execute: The ALU carries out the arithmetic or logic operation and places the result back into the accumulator or memory.
PC Update: After execution, the PC either increases to the next instruction or changes to a new address if a jump occurs.

5.Single-Cycle Implementation

All instructions finish in one oscillator cycle. Fetch, decode, and execute happen one after another in the same cycle. This keeps the design simple and easy to test.

3. MEMORY MODEL

The memory used in the 8-bit RISC processor is a small on-chip SRAM, designed to store both instructions and data in a single block. It is 8 bits wide with 32 addressable locations, accessed through a 5-bit address bus. An 8-bit data bus connects the memory with other CPU components such as the accumulator, ALU, and instruction register. This shared memory structure simplifies the design and supports single-cycle operation.

The memory supports three main functions - read, write, and reset - controlled by the signals mrd (read), mwr (write), and rst (reset). When the read signal is active, data from the addressed location is placed on the data bus and sent to the CPU. When the write signal is active, data from the accumulator or ALU is stored into memory. When reset is active, all memory locations are cleared. SRAM is chosen instead of DRAM because it offers faster access and does not require refresh cycles, which is ideal for single-cycle execution.

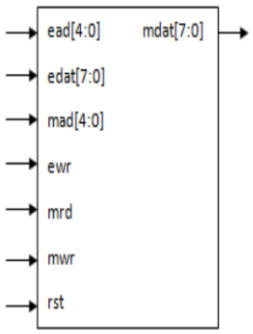

Figure 2: Block diagram of Memory

- Memory Signals
  - ead[4:0] – external address input
  - edat[7:0] – external data input
  - mad[4:0] – internal address input
  - mdat[7:0] – internal data output
  - ewr, mrd, mwr, rst – control signals for write, read, and reset
- Flow of Operation
  - When reset is active, memory contents are cleared.
  - If ewr = 0, the system remains idle.
  - If mwr = 1, data is written into memory.
  - If mrd = 1, data is read from the selected address and sent to the internal bus.
  - If no operation is active, the output goes to high impedance (Z) state.

This SRAM block works closely with the Program Counter, Decoder, and Instruction Register, allowing both instruction fetch and data transfer to happen in a single oscillator cycle. This helps achieve a compact, fast, and power-efficient design.

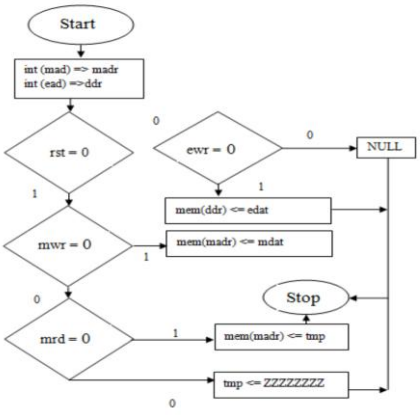

Fig-3: Flow chart of Memory

4. TARGETS

The 8-bit RISC CPU is designed using FreePDK45 (45 nm CMOS) in Cadence Virtuoso and ADE Explorer. The target operating frequency is around 300 MHz, with an expected stable range between 250 MHz and 400 MHz. The actual frequency will be determined from the critical path delay using $f_{max} = 1/T_{pd}$. The estimated gate count is approximately 2,000 or more, covering the ALU, accumulator, program counter, instruction register, decoder, control unit, and memory interface. The total power is expected to be around 13–14 mW at 1.0 V, including both static and dynamic components. Logical effort analysis will be used to support the frequency target. Using logical, branching, and electrical effort, the total path effort (F) will be calculated to validate the design's performance.

5. FREEPDK45

FreePDK45 is a 45 nm CMOS technology that I have been using both in my lab work and in this project. It provides all the required device models, technology files, and design rules needed to design and simulate circuits at the transistor level in Cadence Virtuoso. Since it is an open-source PDK, it is widely used for academic and research purposes and gives students exposure to advanced technology nodes.

In my VLSI Design lab, I used this PDK to design and verify basic digital circuits like inverters, logic gates, half adders, and full adders. This helped me build a solid foundation in circuit design, power analysis, and timing verification. For this 8-bit RISC processor project, I am using the same PDK to implement and simulate the complete processor. This makes the work more realistic and allows me to apply what I learned in the lab directly to a larger and more complex architecture.

6. Technology and Tools

Design: Cadence Virtuoso
Synthesis: 45 nm FreePDK45 CMOS Technology
Verification: Behavioral and Structural Simulation (ADE Explorer, Cadence Virtuoso)

III. CONCLUSION:

This project focuses on the design of an 8-bit RISC CPU using FreePDK45 (45 nm CMOS) technology in Cadence Virtuoso and ADE Explorer. The processor uses a single-cycle architecture with a fixed 8-bit instruction format and an accumulator-based datapath to keep the design simple and efficient. The target operating frequency is 300 MHz, with an expected stable range between 250 MHz and 400 MHz. Logical effort delay analysis is applied to validate timing feasibility. The estimated gate count is around 2,000 and the total power budget is 13–14 mW at a nominal supply voltage of 1.0 V.

The next step is to implement each module at the transistor level and integrate them to build the complete CPU. Functional, timing, and power simulations will be carried out to verify the design and compare the results with the estimated targets. This project provides a strong foundation for understanding the interaction between architecture, circuit design, and technology, and sets the stage for future improvements such as pipelining and optimization.

IV. REFERENCES:

[1] *IJRASET*, "Design of a 16-Bit Harvard Structure RISC Processor in Cadence 45 nm Technology." https://www.ijera.com/papers/Vol2_issue2/H22053058.pdf
[2] *A. H. S. Aishwarya and S. Hiremath, "Design of Low Power High Speed 8-Bit RISC Processor," International Research Journal of Engineering and Technology (IRJET), vol. 7, no. 6, June 2020.* https://www.irjet.net/archives/V7/i6/IRJET-V7I6194.pdf
[3] Cadence Design Systems, "Cadence University Program — Digital Design and Verification Training Modules,"2025.
[4] P. Magnusson and A. Hirano, *The RISC-V Reader: An Open Architecture Atlas*, 2nd ed., Kindle edition, 2018. https://www.cs.sfu.ca/~ashriram/Courses/CS295/assets/books/HandP_RISCV.pdf