Design and Implementation of a 10-bit FSM Processor Using Verilog: AvionCPU

Ege Potur
Istanbul Türkiye
e-mail: {ege.potur@marun.edu.tr}

***Öz* — Bu proje, Verilog HDL kullanan Von Neumann mimarisine dayalı basit bir işlemci olan AvionCPU'nun RTL tasarımını ve uygulamasını sunar. İşlemci, bellek erişimi, aritmetik, kontrol akışı ve sistem düzeyindeki talimatlar dahil olmak üzere dokuz işlemi destekleyen 10 bitlik bir talimat seti mimarisine sahiptir. CPU, talimat getirme, kod çözme, işlenen işleme ve yürütmeyi koordine eden beş ayrı durumda çalışan bir sonlu durum makinesi (FSM) olarak uygulanır. İşlevselliği doğrulamak için simülasyonda üç test programı yürütüldü: ikisi temel aritmetik kullanılarak ve biri MUL talimatını kullanmadan tekrarlanan toplama yoluyla çarpmayı uygulayarak kontrol akışını ve döngü mantığını gösterdi. İşlemci tüm test durumlarını başarıyla geçerek veri yolu ve kontrol mantığının doğru entegrasyonunu doğruladı. Bu proje, düşük seviyeli CPU mimarisinin, talimat döngüsü tasarımının ve donanım tanımlama dillerinde dijital sistem modellemesinin pratik bir gösterimi olarak hizmet eder.**

***Anahtar Kelimeler* — FPGA, CPU**

***Abstract* — This project explains the design and simulation of a simple processor called AvionCPU, which was built using Verilog and follows the Von Neumann architecture. The processor works with 10-bit instructions and supports nine operations, including loading and storing data, arithmetic calculations, jumping in the code, and system commands. The CPU is controlled by a finite state machine (FSM) with five steps that manage the instruction cycle from fetching to execution. To test the processor, three example programs were run in simulation. Two of them used simple arithmetic, and one tested multiplication using a loop and repeated addition instead of the built-in MUL command. This confirmed that the processor handles both arithmetic and control flow instructions correctly. All tests passed successfully, showing that the CPU works as expected. This project helped demonstrate how a basic CPU operates and how digital systems can be modeled and tested using hardware description languages like Verilog.**

***Keywords* — FPGA, CPU.**

I. Introduction

This project focuses on the design and implementation of a basic processor named AvionCPU which is developed in Verilog HDL. The processor is built on the Von Neumann architecture, where both instructions and data use the same memory space. AvionCPU is designed to execute 10-bit wide instructions that consist of a 4-bit operation code and a 6-bit address field. It can execute nine operations: memory load and store (LOD, STO), arithmetic operations (ADD, SUB, MUL, DIV), control flow (JMP, JMZ), and system-level commands (NOP, HLT).

The main objective of the project is to demonstrate how a simple CPU fetches, decodes, and executes instructions using a finite state machine (FSM) structure. Through a five-state control model, the processor sequentially processes each instruction and performs corresponding memory or ALU operations. The implementation is verified through simulation by executing multiple test programs stored in memory, including one that performs multiplication through repeated addition using a loop structure, thereby validating correct functionality of control flow and branching logic. In the end, the project helps students better understand how a CPU works, how instructions are handled, and how to design digital systems at the RTL (register-transfer level).

II. System Architecture

The AvionCPU processor was designed and tested using Verilog, a hardware description language. All development and simulation were done on EDA Playground, an online tool that allows users to write, compile, and run Verilog code without installing any software. It uses open-source simulators like Icarus Verilog and provides a simple way to test hardware designs in the browser.

AvionCPU is built using the Von Neumann architecture, where both instructions and data are stored in the same memory.

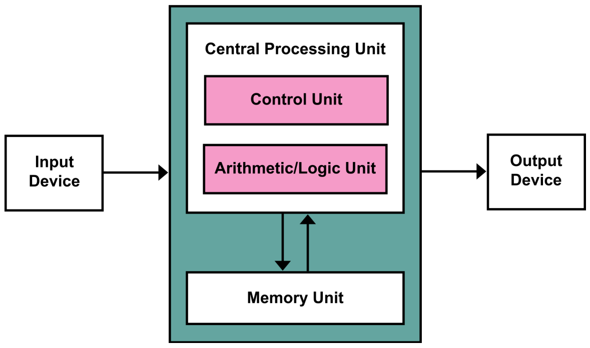
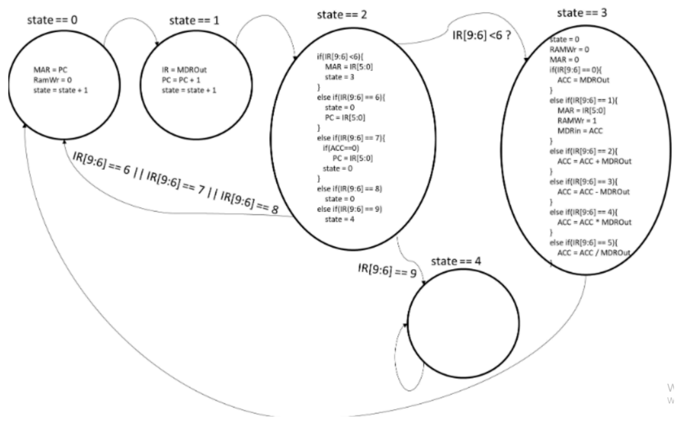


Fig 1. Von Neumann Architecture

The processor uses a small memory (called block RAM or BRAM) with 64 addresses, and each memory location is 10 bits wide. It works as a finite state machine (FSM) with five steps (states), going through each one to process instructions. The design includes four key parts:

- Program Counter (PC): Keeps the address of the current instruction.
- Instruction Register (IR): Stores the instruction being processed.
- Accumulator (ACC): A temporary storage for arithmetic and logic operations.
- Control Unit / FSM: Manages the CPU steps: fetch, decode, operand handling, and execution.

The memory module (**`blram`**) is used for both instructions and data. It is created outside the CPU and filled with test programs at the start using Verilog's **`initial`** blocks. The testbench (**`tb_avion_cpu`**) simulates how the CPU runs by giving it clock and reset signals, running test cases, and checking the final results.



```
MAR = PC
RamWr = 0
state = state + 1
```

```
IR = MDROut
PC = PC + 1
state = state + 1
```

```
if(IR[9:6] <6){
  MAR = IR[5:0]
  state = 3
}
else if(IR[9:6] == 6){
  state = 0
  PC = IR[5:0]
}
else if(IR[9:6] == 7){
 if(ACC==0)
    PC = IR[5:0]
 state = 0
}
else if(IR[9:6] == 8)
  state = 0
else if(IR[9:6] == 9)
  state = 4
```

```
state = 0
RAMWr = 0
MAR = 0
if(IR[9:6] == 0){
  ACC = MDROut
}
else if(IR[9:6] == 1){
  MAR = IR[5:0]
  RAMWr = 1
  MDRin = ACC
}
else if(IR[9:6] == 2){
  ACC = ACC + MDROut
}
else if(IR[9:6] == 3){
  ACC = ACC - MDROut
}
else if(IR[9:6] == 4){
  ACC = ACC * MDROut
}
else if(IR[9:6] == 5){
  ACC = ACC / MDROut
}
```

Fig. 2 State diagram

An architecture diagram is also used to show how each part of the processor connects and works together. It shows how the CPU reads from and writes to memory, how data flows between modules, and how control signals are handled by the FSM.

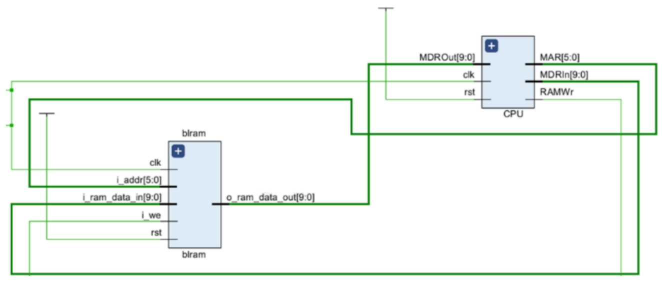


Fig. 3 Project design diagram

III. Software Used

In order to test the AvionCPU, a testbench was implemented in Verilog and provided as part of the project. It includes two main parts: the testbench module called **`tb_avion_cpu`**, which sends the clock and reset signals and checks the results and the RAM module called **`blram`**, which holds the instructions and data used during the tests.

There are three test cases, each written using AvionCPU's 10-bit machine language. These programs check different features of the processor:

- Test Case 1: Adds the values in memory addresses 50 and 51 and saves the result in address 52.
- Test Case 2: Multiplies the values in addresses 50 and 51 using the MUL instruction, then stores the result in address 52.
- Test Case 3: Multiplies the same values as Test Case 2, but without using MUL. Instead, it adds the number in address 50 many times using a loop. This tests the jump instructions (JMP, JMZ) and control logic.

The programs are loaded into memory automatically at the start of the simulation using Verilog's **`initial`** blocks. After the CPU runs the program, the testbench checks if the final result (for example, the value in address 52) matches what it should be. If not, it shows an error.

IV. Results

The AvionCPU runs using a five-state finite state machine (FSM), which handles every step of the instruction cycle: fetching, decoding, executing, and interacting with memory. In simulations, all three test programs ran correctly, and the final results matched the expected values. This confirms that the CPU design and control logic were implemented correctly.

By completing this project, we learned how a CPU works internally, how a finite state machine controls each instruction step, and how instructions are handled in hardware. We also saw how loops and conditionals can be built at the hardware level without using any high-level programming. Overall, the project gave valuable hands-on experience in building a working processor, improved our understanding of digital systems, and showed how control logic is used in real embedded system design.

Project team

Ege Potur: Ege is a software engineer and a MSc. student at Marmara University. He specialized in developing C++ and Qt applications. His main interests are computer vision and machine learning.