An FPGA-Based RTL Design Using Verilog

Yusuf Sait ERDEM

Bilgisayar Mühendisliği

Fen Bilimleri Enstitüsü, Marmara Üniversitesi

e-mail: yusufsaiterdem@gmail.com

*Abstract* — This project presents a processor design, implemented in Verilog at the Register-Transfer Level (RTL), and demonstrates its usage. It shows the processor's capability to execute machine language code snippets by performing operations such as block RAM read/write, arithmetic (add, multiply), and control flow (conditional jump). The project illustrates the functionality of key processor components such as Memory, Control Unit and Registers by executing machine language instructions.

Keywords — FPGA, CPU

# Introductıon

In the rapidly evolving landscape of digital electronics, the demand for high-performance yet cost-effective hardware solutions for various tasks is very important. This project focuses on designing and demonstrating some parts of a custom processor named AvionCPU which is structured using the Von Neumann architecture and is capable of executing simplified machine language instructions.

Our approach leverages techniques like Register-Transfer Level (RTL) design, a powerful abstraction for describing digital circuits, which is implemented using the Verilog Hardware Description Language [2]. Furthermore, the project utilizes Field-Programmable Gate Arrays (FPGAs) as the target hardware platform, enabling flexible and rapid prototyping [1], in conjunction with industry-standard Electronic Design Automation (EDA) tools such as Vivado [3] for synthesis and optimization.

The processor supports fundamental operations such as arithmetic computation (addition, subtraction, multiplication), memory access (load/store), and control flow (jumps and conditional jumps). These capabilities are implemented through a state machine-based architecture that coordinates key processor components—namely the Control Unit, Memory, Registers, and Arithmetic Logic Unit (ALU).

This project serves both educational and engineering purposes. It offers a perspective on how low-level hardware modules interact and operate together to execute programs.

# System Archıtecture

The AvionCPU processor's architecture is fundamentally based on the Von Neumann Architecture, a design principle where both program instructions and data are stored in the same memory space. This unified memory access simplifies the processor's design by allowing a single bus for both instructions and data. The core components of the AvionCPU, as with typical Von Neumann designs, include the Control Unit, responsible for orchestrating operations; Memory (specifically, a Block RAM module for storing both code and data); and Registers (including the Accumulator and Program Counter) for temporary data storage and address tracking. The Arithmetic Logic Unit (ALU) performs all arithmetic and logical operations, such as addition, subtraction, and multiplication.

The entire processor design is conceived at RTL, a high level of abstraction that describes the flow of data between hardware registers and the logical operations performed on that data. This RTL design is meticulously captured using Verilog, a widely adopted Hardware Description Language (HDL), which allows for precise behavioral and structural modeling of digital circuits.

For physical realization and rapid prototyping, Field-Programmable Gate Arrays (FPGAs) serve as the target hardware platform. FPGAs provide the flexibility to configure custom digital circuits, making them ideal for processor development. The entire design flow, from RTL code to FPGA bitstream generation can be managed using Vivado, Xilinx's comprehensive Electronic Design Automation suite [3]. Vivado facilitates tasks such as synthesis, implementation, and verification of the Verilog code onto the chosen FPGA device.

# Software Used

To support the development phase and educational aspects of this project, an online hardware and software simulation platform, www.avionchip.com, is utilized. This platform allows for virtual execution and observation of the designed CPU, enabling effective debugging and validation of the designed architecture without requiring physical hardware.

The final design has four fundamental units that are; registers, memory, aritmetic logic unit (ALU) and control unit.

**Registers**  
  
AvionCPU utilizes several internal registers for control and data handling:

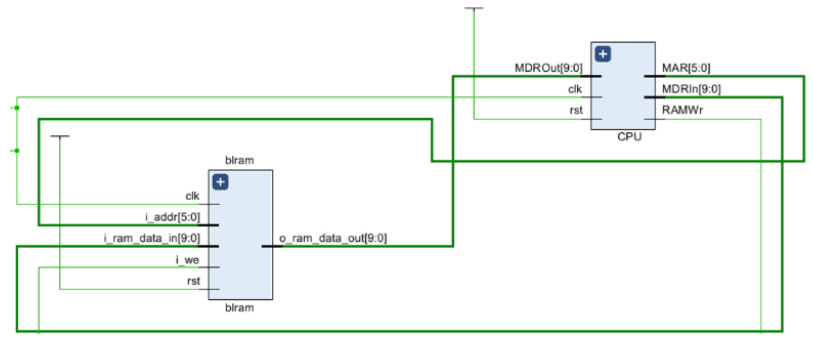
* PC (Program Counter): Points to the current instruction address in RAM.
* IR (Instruction Register): Holds the fetched instruction.
* ACC (Accumulator): Serves as the main arithmetic working register.
* state: Indicates the current stage in the finite state machine (FSM).

Additionally, the design internally uses:

* MAR (Memory Address Register, 6-bit): Addresses RAM for read/write operations.
* MDRIn and MDROut (10-bit): Handle input and output data to/from RAM.
* RAMWr (1-bit): Enables write operations in RAM.

**Memory**

Block RAM (BRAM) is implemented to serve as storage for both program instructions and data [4]. BRAM addresses are 6 bits wide, enabling access to 64 memory locations, each 10 bits wide. Data transfers between BRAM and other units are mediated through MDR and MAR registers. The below diagram shows how the BRAM is integrated in the system:

Figure 1: Integration of Block RAM (BRAM) with the CPU, showing signal connections for address, data, control, and synchronization.

**Processing Unit (ALU)**

The Arithmetic Logic Unit performs the main computation tasks:

* ADD: Adds a memory operand to ACC.
* SUB: Subtracts a memory operand from ACC.
* MUL: Multiplies ACC by a memory operand.

These operations are executed depending on the opcode of the current instruction.

**Control Unit**

The Control Unit orchestrates instruction fetch, decode, and execution via a 5-state finite state machine:

* State 0 – Instruction fetch from RAM.
* State 1 – Instruction decode and PC increment.
* State 2 – Load operand from memory if required.
* State 3 – Perform computation or control action.
* State 4 – (If required) finalize actions like writing to memory.

The Control Unit ensures synchronization and proper signal generation based on the current instruction and state.

**Instruction Set Architecture (ISA)**

The processor supports 9 instructions encoded in a 10-bit format:

* LOD, STO, ADD, SUB, MUL: Arithmetic and memory instructions.
* JMP, JMZ: Control flow instructions.
* NOP: No operation.
* HLT: Halt processor.

Instruction format: [9:6] Opcode | [5:0] Address

Each instruction is executed step-by-step according to the FSM state transitions.

# Results

The designed AvionCPU supports the following operations:

* Memory Access: Loading from and storing to specific RAM addresses.
* Arithmetic Operations: Performing addition, subtraction, and multiplication using RAM-stored operands.
* Control Flow: Jumping unconditionally or conditionally based on the value of ACC.

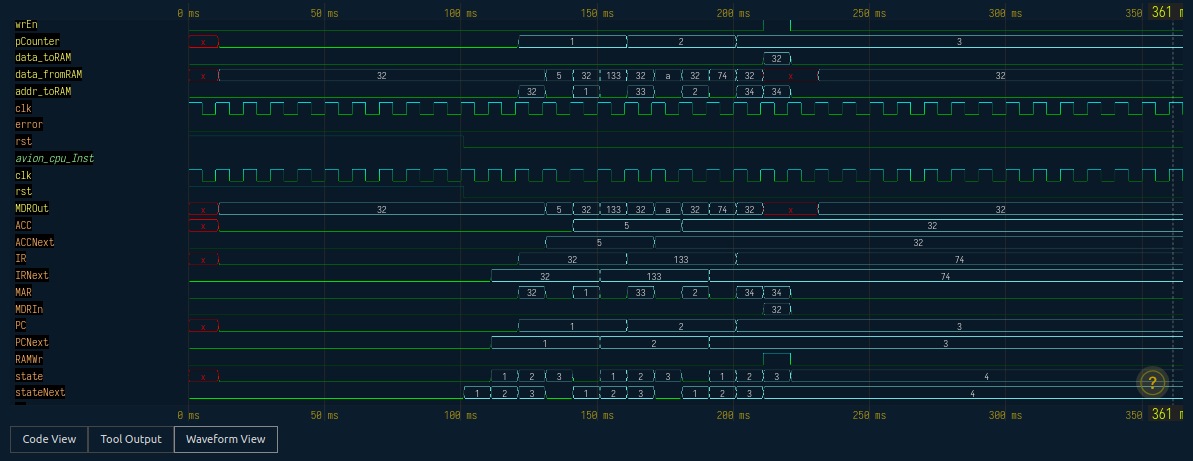
These operations were tested using several machine language code snippets, illustrating real use cases such as summing or multiplying numbers stored in memory.

Test Example 1 demonstrated memory read, addition, and store operations.

Test Example 2 tested multiplication using the MUL instruction.

Test Example 3 emulated multiplication via repeated addition, testing more complex control flow with loops and conditional jumps.

The following figure shows values of the registers on the AvionCPU while running the test example 2 in the waveform:

Figure 2: Image of the behavior of AvionCPU registers during execution of Test Example 2 (multiplication), including changes in PC, IR, ACC. The image is acquired from simulation tool in www.avionchip.com.

##### Project team

Yusuf Sait ERDEM - Computer Engineer (M.Sc.) with a background in data science and embedded systems. His experience includes designing and implementing neural networks, real-time computer vision systems, and embedded software design. He holds a Master's degree in Electrical and Electronic Engineering from İzmir Demokrasi University and is currently pursuing a Doctoral degree in Computer Engineering at Marmara University.

##### Reference Files

Github link: https://github.com/YusufSait/An-FPGA-Based-RTL-Design-Using-Verilog/

##### References

1. Madariaga, Ander, et al. "Review of electronic design automation tools for high-level synthesis." *2010 International Conference on Applied Electronics*. IEEE, 2010.
2. Wilson, Peter. *Design recipes for FPGAs: using Verilog and VHDL*. Newnes, 2015.
3. Xilinx, “Vivado Design Suite User Guide,” vol. 895, 2016.
4. Le Roux, Rikus, George Van Schoory, and Pieter Van Vuuren. "Block RAM-based architecture for real-time reconfiguration using Xilinx® FPGAs." *South African computer journal* 56.1 (2015): 22-32.