Design of 8-BIT Harvard CPU
Team 3
Date

1. Team members

Name	Matriculation Number	Email Address
Mofifoluwa Akinwande	2220059	Mofifoluwa-ipadeola.akinwande@stud.hshl.de
Rubayet Kamal	2219943	rubayet.kamal@stud.hshl.de
Moiz Zaheer Malik	2220074	Moiz-zaheer.malik@stud.hshl.de

2. Concept description

The purpose of the lab project is to have a good understanding of Digital Design and Computer-Aided Design of FPGAs. Digital Design parts are to be realized in VHDL and the hardware such as the PCB where the FPGA along with other components sit is to be designed in KiCAD.

It has been decided among the group mates after discussion with Prof. Dr. Hayek that an 8-BIT Harvard CPU will be developed. The inspiration comes from first semester's Computer Science I course where different architectures such as Harvard (separate bus for data and instructions) and Von Neumann (single bus for data and instruction) were introduced.

The initial project conception looked as can be seen in Figure 1. Instructions are to be pre-loaded into the module known as Instruction Unit after which the CPU will perform arithmetic or logical computations to show the output on a 7 Segment Display of the development board. Figure 2 shows the state transitions that take place within the CPU. Starting from a reset state, the CPU always follow the state transitions shown. The execution depends upon the decoded instruction. An example is the instruction '10000001' means load the data that is stored in address '00001'. As the output from CPU will be in binary and it is easier to program 7 Segment Displays with the input being in BCD form, a separate decoder is also designed which takes the output from the CPU as its input.

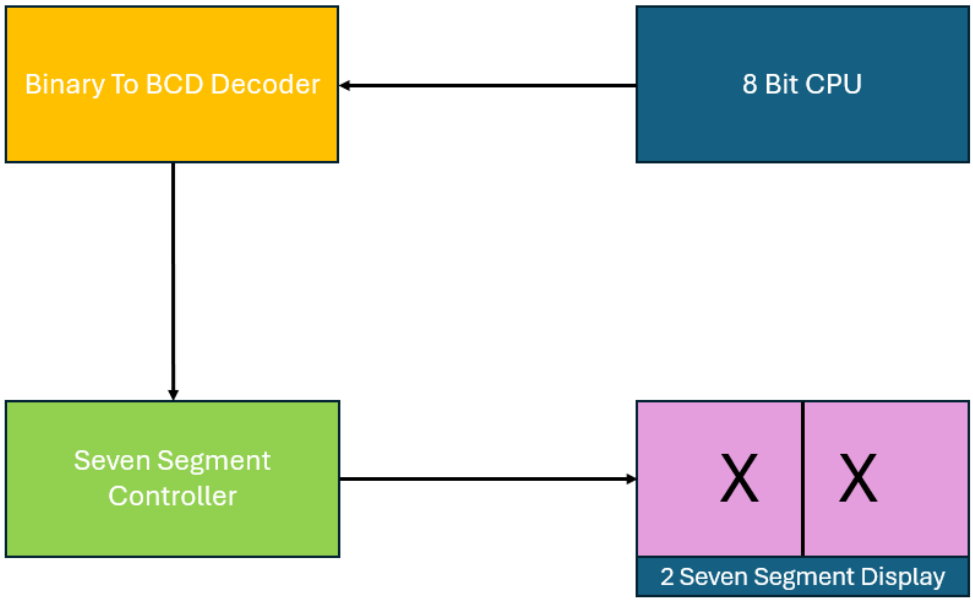


Figure 1: Top Level Block Diagram for 8 BIT CPU.

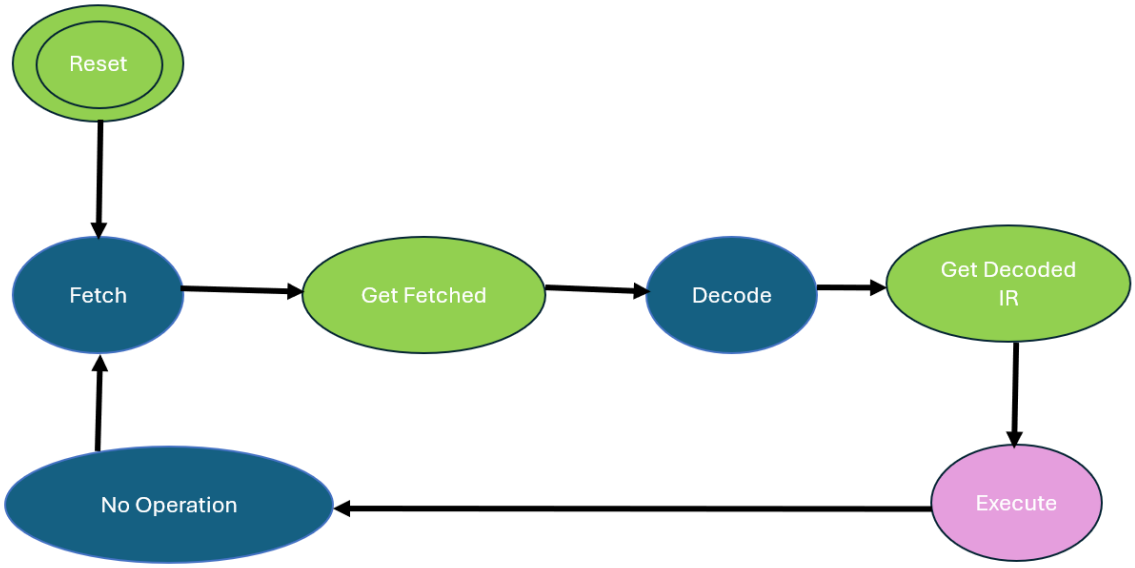


Figure 2: State diagram showing transitions.

Table 1: Opcodes for implemented instructions

Instruction	Opcode
No Operation	000
STORE	001
SUM	010
SUB	011

LOAD	100
OR	101
JUMP If No Carry Flag	110
AND	111

The purpose of this documentation is to highlight complete implementation of the project. Section 3 will discuss about task division along with project management method employed. Furthermore, Section 4 will bring forth the technological approaches implemented on FPGA and with PCB. Section 5 discusses the implementation of digital design in VHDL with blocked diagrams. This section also will highlight results with testbenches. Section 6 takes over to discuss A-Z of PCB Design where the FPGA along with other components are designed in KiCAD. Section 7 discusses the results of the whole project along with the conclusion drawn.

3. Project/Team management

This section is divided as follows: Subsection 3.1 shows the project management method employed and Subsection 3.2. tabulates the division of tasks among the 3 group members.

3.1. Project Management Method: This section is divided into two subsections. Subsection 3.1.1 outlines the design flow for the VHDL part whereas subsection 3.1.2 outlines the flow for PCB Design.

3.1.1 Digital Design:

As can be seen in Figure 3, a specification diagram showing the requirements was first conceptualized, thereafter a block diagram was prepared for the functional design. Once the functional design was set up and ready, VHDL code was written for every component and their respective function. In order guarantee that the VHDL code worked, verification was done via testbench in Modelsim. Finally, the working code was synthesized, implemented onto the FPGA before creating the Bitfile.

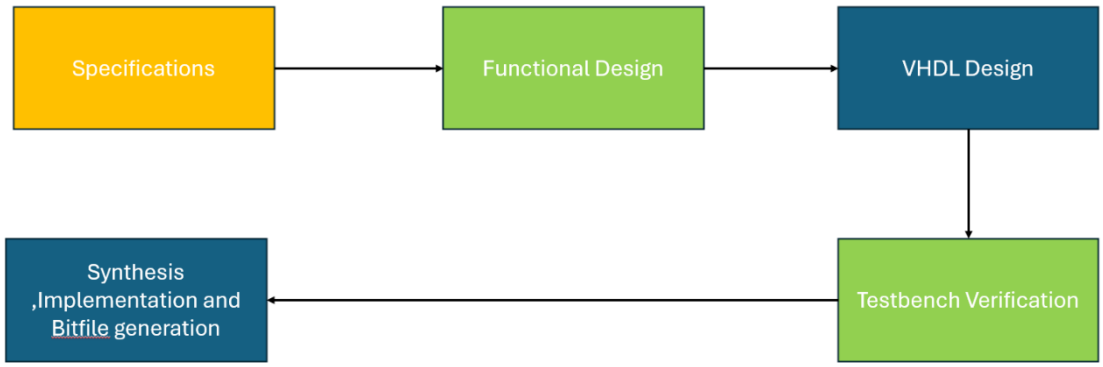


Figure 3: Design flow diagram for Digital Design.

3.1.2 PCB Design:

This flowchart outlines the complete design process of the custom FPGA development board. It starts with setting project objectives, followed by schematic design and selecting

components. The design is organized using hierarchical sheets, then laid out as a PCB. After validating the design with ERC/DRC checks, the final step is generating Gerber files for manufacturing.

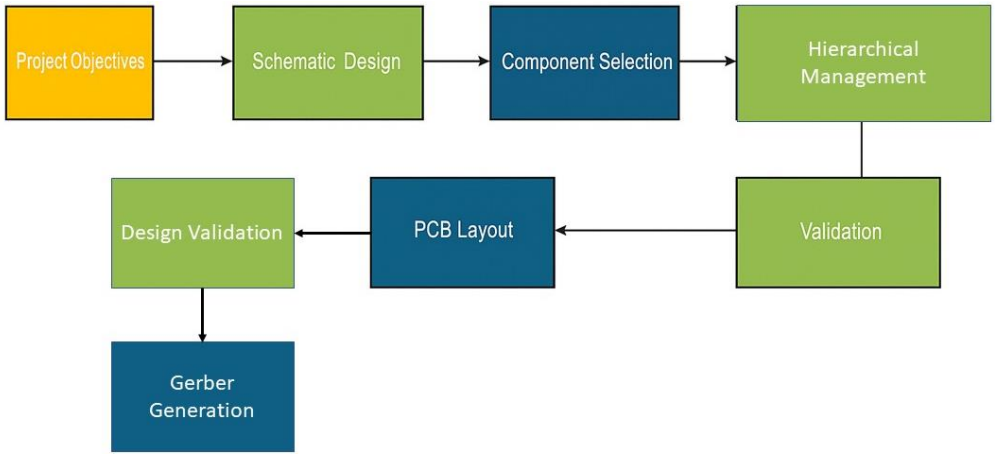


Figure 4: Design flow diagram for PCB Design.

3.2. Task Division

Table 1 below shows the division of task among group mates. Mofifoluwa and Rubayet were primarily in charge of VHDL. The CPU has different parts such as Accumulator or Data Unit and these were distributed among them in a fair manner. On the other hand, the PCB Design department was completely overseen by Moiz.

Table 2: Task division among group mates

Name	Tasks
Mofifoluwa Akinwande	1) VHDL Modelling of Accumulator, Data Unit, Instruction Unit and Program Counter. 2) Testbench creation and verification of individual modules. 3) Preparation of state diagrams.
Rubayet Kamal	1) VHDL Designing of 8-BIT ALU, Instruction Register, Control Unit, BinaryTo7Segement Converter and Seven Segment Controller. 2) Testbench creation of Top-Level View for instruction verification. 3) Synthesizing, Implementation and Bit file creation. 4) Preparation of state diagrams.

Name	Tasks
Moiz Zaheer Malik	1) Design and development of a custom FPGA development board using the iCE40UP5K FPGA. 2) Schematic creation, hierarchical sheet structuring, and component selection in KiCAD. 3) PCB layout, manual routing of critical nets, and validation using ERC/DRC checks. 1) Documentation of the full design process, including design objectives, tools, and future improvements.

4. Implemented and Integrated Technologies:

4.2. **Very High-Speed Integrated Hardware Description Language (VHDL):** VHDL is a language used to describe the behaviour and structure of electronic systems, specially in digital circuits. In this project, VHDL is used to write code that defines the logic of the digital system such as finite state machine of the control unit as shown in Figure 2. VHDL allowed for writing synthesizable code that was later mapped onto FPGA. Libraries such as IEEE libraries provided standardized packages and definitions essential for writing reliable, portable and synthesizable hardware description. IEEE.STD_LOGIC_1164 is an example that defined standard logic types such as std_logic and std_logic_vector.

4.3. **Field Programmable Gate Array (FPGA):** FPGA is a reconfigurable hardware platform used to implement digital logic. In this project, FPGA is used to physically implement the VHDL code. Therefore, the FPGA served as the hardware platform where the VHDL designed digital circuits were implemented. The FPGA board used in this project is the Nexys A7 board which is a complete, ready-to-use digital circuit development platform based on the latest Artix-7™ Field Programmable Gate Array (FPGA) from Xilinx®. With its large, high-capacity FPGA, generous external memories, and collection of USB, Ethernet, and other ports, the Nexys A7 can host designs ranging from introductory combinational circuits to powerful embedded processors.

4.4. **KiCAD:** KiCAD is an open-source PCB Design Software which allowed us to design custom PCB's that connect to our FPGA board and other system components. KiCAD is used to design custom PCBs for interfacing external components such as sensors or user interfaces with the FPGA. This allowed for a more compact and robust hardware implementation.

4.5. **Github:** Github was used as version control for the project. It allowed for tracking changes to code and also acted as a central storage for project files and their version history.

5. VHDL and FPGA Implementation

In this section, the complete digital design in VHDL of modules composed of the CPU will be discussed. The modules are shown in Figure 4. Subsection 5.1 describes the design of each module and their purpose by showing their corresponding block diagram generated in Xilinx Vivado. Subsection 5.2 shows testbench and result of the top-level CPU for two different instructions allowing the verification before synthesizing, implementation and bit file generation.

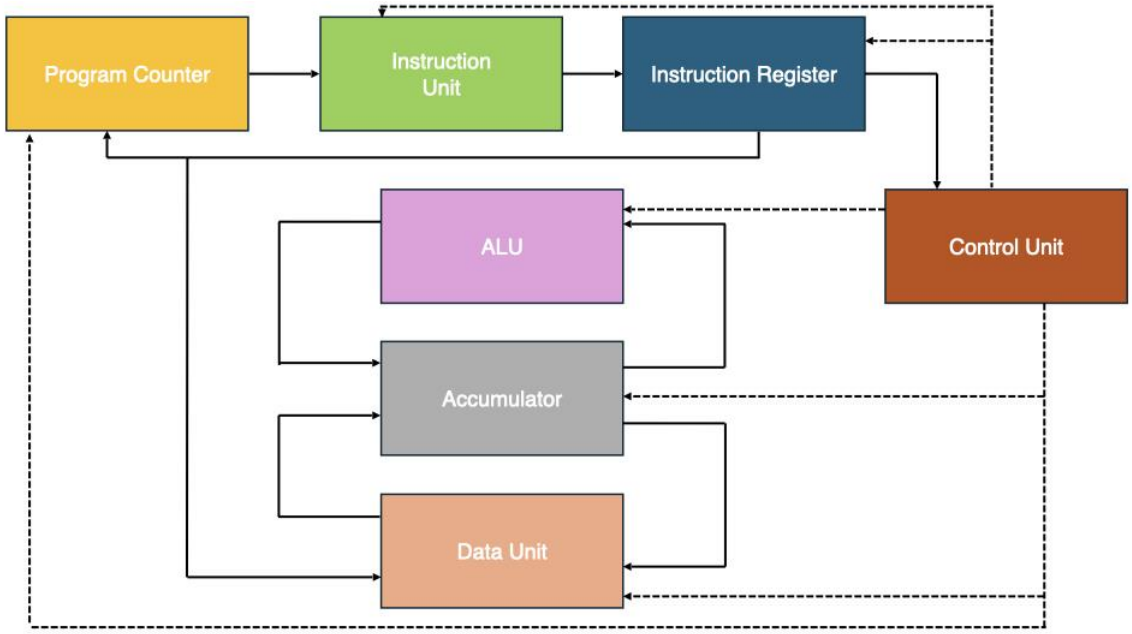


Figure 4: Modules of CPU showing their connections.

5.2. Components designed in VHDL:

5.1.1. Program Counter: The role of this module is to point to the next instruction in Instruction Unit. A MUX is used to allow for either an incrementation of 1 to point to the next instruction, or for a jump instruction or to reset in order to start from the beginning. All of this depends on the first 3 bits of the instruction in instruction unit as opcode of “110” leads to Jump which is listed in Table 1. An 8-bit register made of 8 D Flip-flops points to the next instruction every clock cycle when the flip flops are enabled. The eight-bit full adder keeps adding 1 to its output for every next instruction the program counter points to. The block diagram of the program counter with all the blocks discussed can be seen in Figure 5.

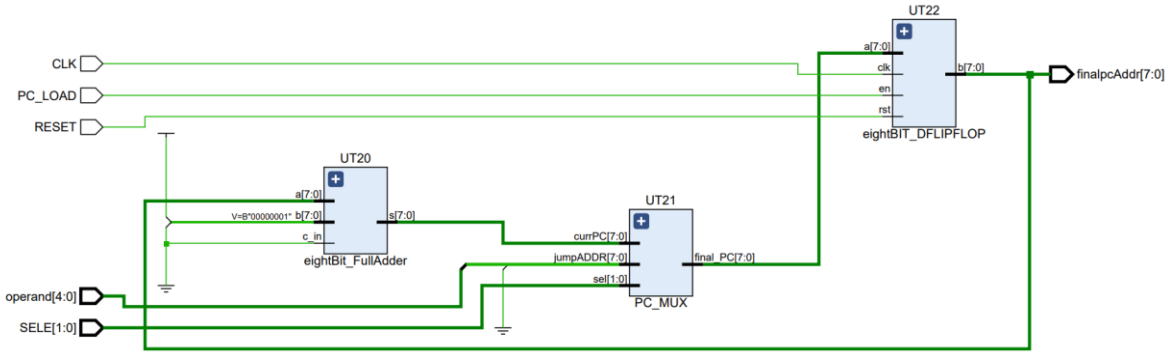


Figure 5: Block Diagram for Program Counter.

5.1.2. Instruction Unit: The role of instruction unit is to give the address of the instructions preloaded in the instruction unit. The output address depends upon the program counter output as program counter points to the address of the next instruction. This is shown in Figure 6. The output address goes to the Instruction Register which is discussed in 5.1.3.

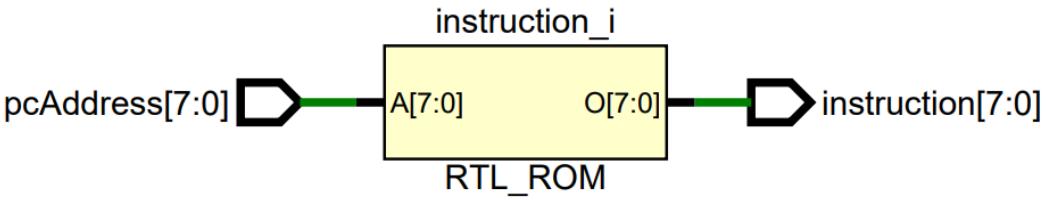


Figure 6: Block Diagram for Instruction Unit.

5.1.3. Instruction Register: The instruction register has a simple role. It receives the address of the instructions from instruction unit as an input and divides that address into portions: opcode and operand. Opcode refers to the instruction type and operand refers to the data on which the operation will be performed. For example: an instruction of “10000001” is divided into “100” and “00001” meaning load the data which is stored in 00001. This is shown in Figure 7.

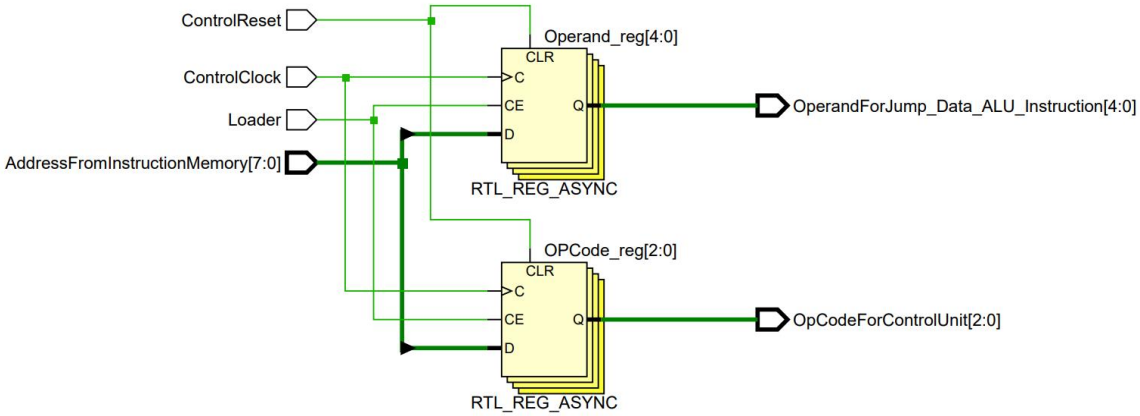


Figure 7: Block Diagram for Instruction Register.

5.1.4. Accumulator: The accumulator is one of the most important parts of the CPU. It loads data from memory unit, stores data into memory unit and acts as one of the operands in ALU mathematical or logical calculations. The accumulator can be said to be divided into three parts as shown in Figure 8: Mux, Register, demux. A MUX which oversees selecting whether to take the data from ALU or Data Unit. After ALU operations, the data from ALU is needed to be stored in the accumulator. On the other hand, when the goal is to load data from memory unit, the data is taken from data unit. The purpose of the register is to reload the accumulator with the new data. Finally, the role of demux is to decide to which module should the output be sent to. For mathematical or logical operations overseen by ALU, the selector is setup for ALU and on the other hand for instructions such as Store a data into memory unit, the selector is setup to choose the other bus as the output.

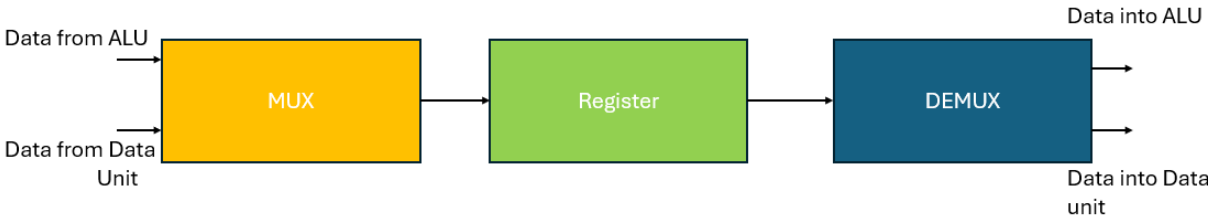


Figure 8: Block Diagram for Accumulator.

5.1.5. Arithmetic Logical Unit (ALU): An 8-Bit ALU is basically designed in a structural method using 8 1-bit ALUs as shown in Figure 9. The following functions is implemented by ALU in the CPU: Addition, Subtraction, AND, OR. The two operands of the ALU are data from the memory unit and data from accumulator. These two operands must be enabled to perform the operations. This is overseen by the control unit which is talked about later.

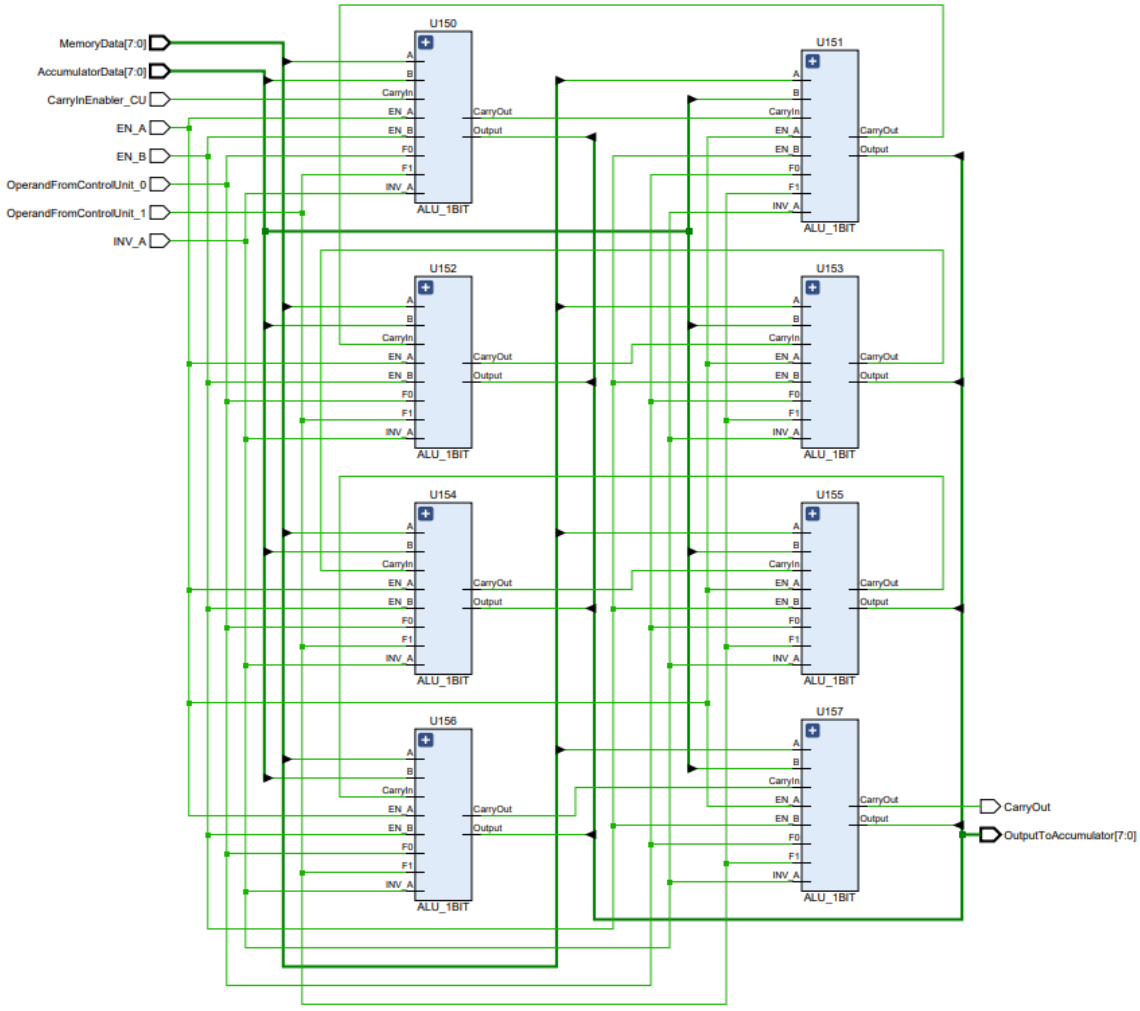


Figure 9: Block Diagram for 8-bit ALU.

5.1.6. Data Unit: The data unit is like a storage unit. It has data stored addresses. As the operand is 5 bits wide, a maximum number of 32 different addresses can be used to store data. The data unit is only accessible for read and write when MEM_READ and MEM_WRITE are enabled which is overseen by the control unit depending on whether LOAD and STORED are decoded respectively as instruction or not. Figure 10 shows this.

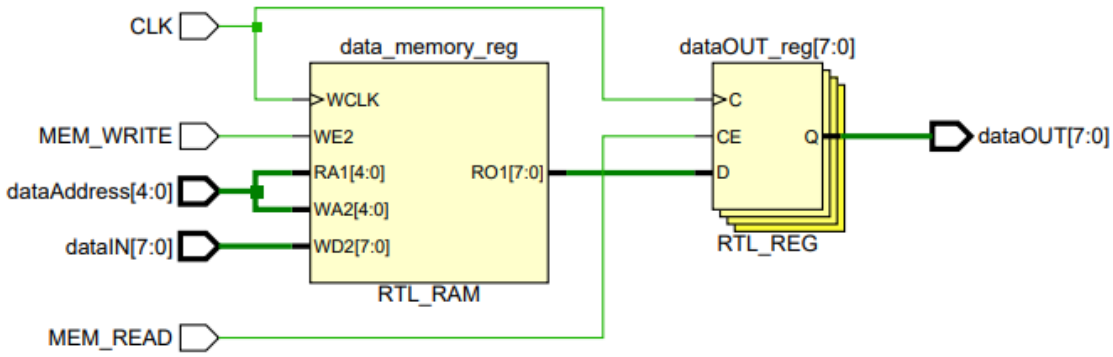


Figure 10: Block Diagram for Data Unit.

5.1.7. Control Unit: The Control Unit is the brain of the CPU or in other words, the orchestrator. It can be seen in Figure 2 how the CPU transitions through different states. The execution of each instruction depends upon the opcode which are the first 3 bits of the instruction as shown in Table 1. These 3 bits are fed into the control unit where the opcode is decoded. The complete state machine is implemented in Control Unit. A simplified version of the block diagram of Control Unit can be seen in Figure 11.

Figure 11: Block Diagram for Control Unit.

5.1.8 Top Level: Figure 12 shows the block diagram from a top level or a bird view for the complete system. It resembles the block diagram of Figure 1 which was formed when coming up with concept. As the output of the PCB will be in binary, a binary to BCD decoder is designed to transform the output to BCD.

For each of the four seven segment displays to appear bright and continuously illuminated, all eight digits should be driven once every 1 to 16ms, for a refresh frequency of about 1 KHz to 60Hz. For example, in a 62.5Hz refresh scheme, the entire display would be refreshed once every 16ms, and each digit would be illuminated for 1/8 of the refresh cycle, or 2ms. The controller must drive low the cathodes with the correct pattern when the corresponding anode signal is driven high. This is the role of the seven-segment controller to ensure that the clock drives each display at a time which the human eye won't be able to notice.

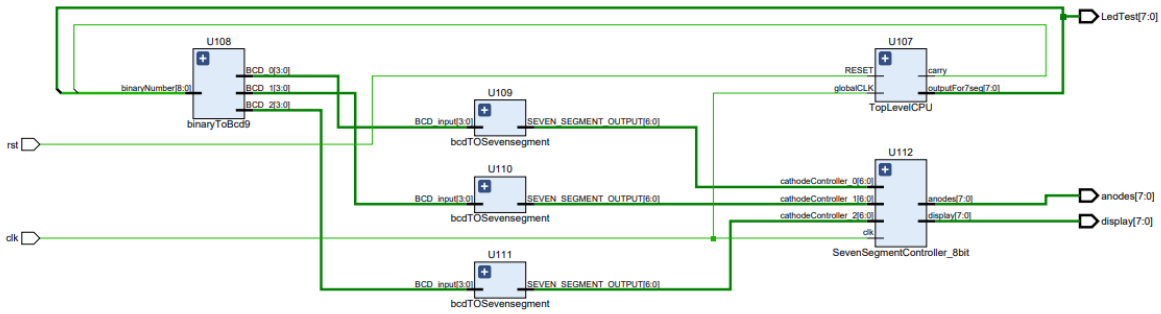


Figure 12: Block Diagram for Bird View.

5.2 Testbench verification:

Almost all components were verified by Testbenches. In this documentation only the top level testbench is shown as an example. This one testbench should suffice to get a full picture of the CPU.

The following instruction set is preloaded into the instruction unit as shown in Figure 13. To make it easier to understand, the data unit, which as 32 possible addresses, has data matching the address. Meaning address 0 has Data 0, address 1 has data 1…. Address 31 has data 31. The instructions are as follows: Load 4, Store 4 in address 3 which originally had 3 as the data, and now load the data stored in address 3 to show in Seven Segment Display. The successful operation should show 4 as the final output, instead of 3 or any other data.

```
CASE(pcAddress) IS
                WHEN "00000000" => instruction<="00000000";
                WHEN "00000001" => instruction<="10000100";          --Load
                WHEN "00000010" => instruction<="00100011";          --Store
                WHEN "00000011" => instruction<="10000011";          --Load
                WHEN OTHERS =>     instruction <= "00000000"; -- No Operation
END CASE;
```

Figure 13: Example of executed instruction.

The testbench as simulated in ModelSim looks as shown in Figure 14. The second to last signal shows 4 is the output for the Seven Segment Display. Load caused DataMemoryRead_TB to be HIGH whereas Store made DataMemoryRead_TB to be LOW. The state transitions can also be verified as the following sequence is maintained: ProgramCounterLoader_TB, InstructionRegisterLoad_TB and AccumulatorLoader_TB.

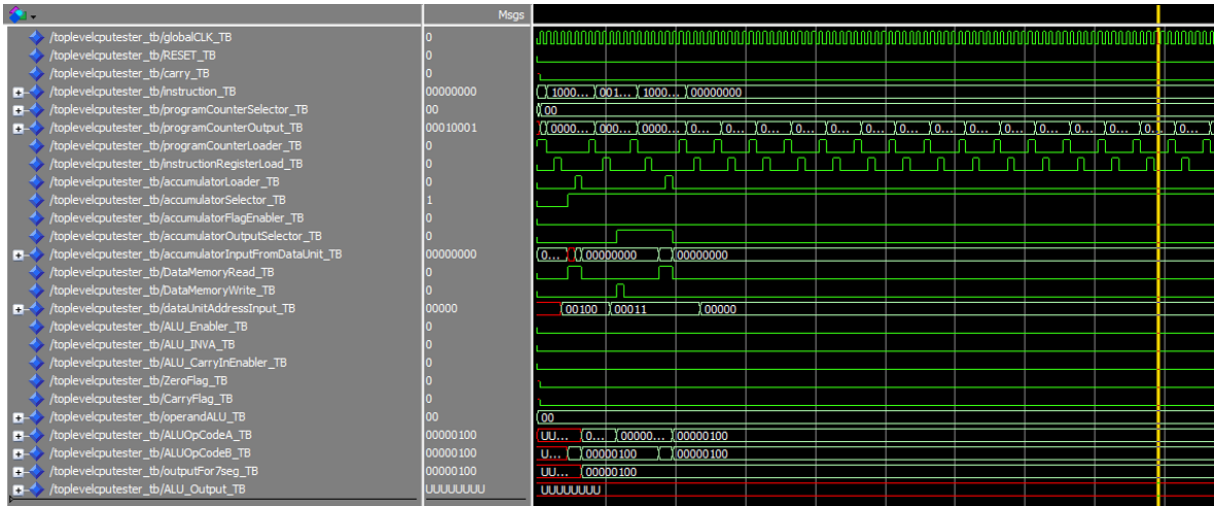


Figure 14: Wave for testbench verification of Top Level.

6.PCB Design

FPGAs are widely used in modern digital systems due to their flexibility and speed. However, most commercial development boards are either expensive or include too many features for simple use cases. This project addresses that by designing a beginner friendly, low-cost board that focuses only on core FPGA functionalities. The Lattice iCE40UP5K was chosen because it is low power, compact, and compatible with open-source toolchains. The entire board can be powered and programmed via a single micro-USB port.
The key goals were to create a compact, USB-powered FPGA board which supports a basic FPGA programming and UART communication including essential feedback via LEDs and a 7-segment display. Another goal was to use to source component to keep it simple enough for anyone to replicate. Essential features involved a USB USB interface (JTAG + UART via FT2232HL) alongside a SPI Flash (W25Q128JV) for storing bitstreams. A Voltage regulation of 3.3V and 1.2V from USB 5V is provided in parallel to a 4-digit 7 segment display (HDSP-B09G). Of course, CDONE indicator LED and Reset button too were added for testing and resetting.
The following tools were used: **KiCAD 7** for schematic and PCB layout design, **SnapEDA** for part libraries (symbols and footprints) and **FreeRouting**: assisted in auto routing some non-critical nets which assisted in auto routing some non-critical nets. Table 3 shows the components alongside their purpose in the project and the part number used.

Table 3: Key component along their purpose and part number.

Component	Purpose	Part Number
FPGA	Main chip	iCE40UP5K-SG48ITR50
USB interface	JTAG & UART	FT2232HL
Flash	Bitstream storage	W25Q128JV
LDO Regulator	3.3V I/O	LDO40LPU33RY
LDO Regulator	1.2V core	LD1117V

Display	Output	HDSP-B09G
Oscillator	Clock source	16 MHz XO
Button	Manual reset	CRESET_B
LED	CDONE indicator	Green SMD LED

6.1 Schematic Overview

Figure 15 shows the schematic design from the top level. The schematic is organized into several hierarchical sheets to keep the design modular and easy to troubleshoot. Each sheet handles a specific function such as power regulation, communication, configuration, and display control.

The FT2232HL chip provides dual USB channels one for JTAG programming and one for UART communication. This dual functionality over a single USB connection eliminates the need for separate power or programming tools. To ensure proper power-up behavior of the SPI Flash, 10kΩ pull-up resistors are added to the CS, WP, and HOLD pins. A 5V Schottky diode was used for USB input protection. Decoupling capacitors were placed next to every power pin on the FPGA and major ICs.

The 4-digit HDSP-B09G display is connected to the FPGA through current limiting resistors. Its signal lines are routed carefully to reduce crosstalk and maintain uniform brightness across digits.

Two LEDs are also included: one for CDONE status to indicate successful FPGA configuration, and another user-controlled LED for basic output testing. **FT2232HL**: USB communication (UART + JTAG)

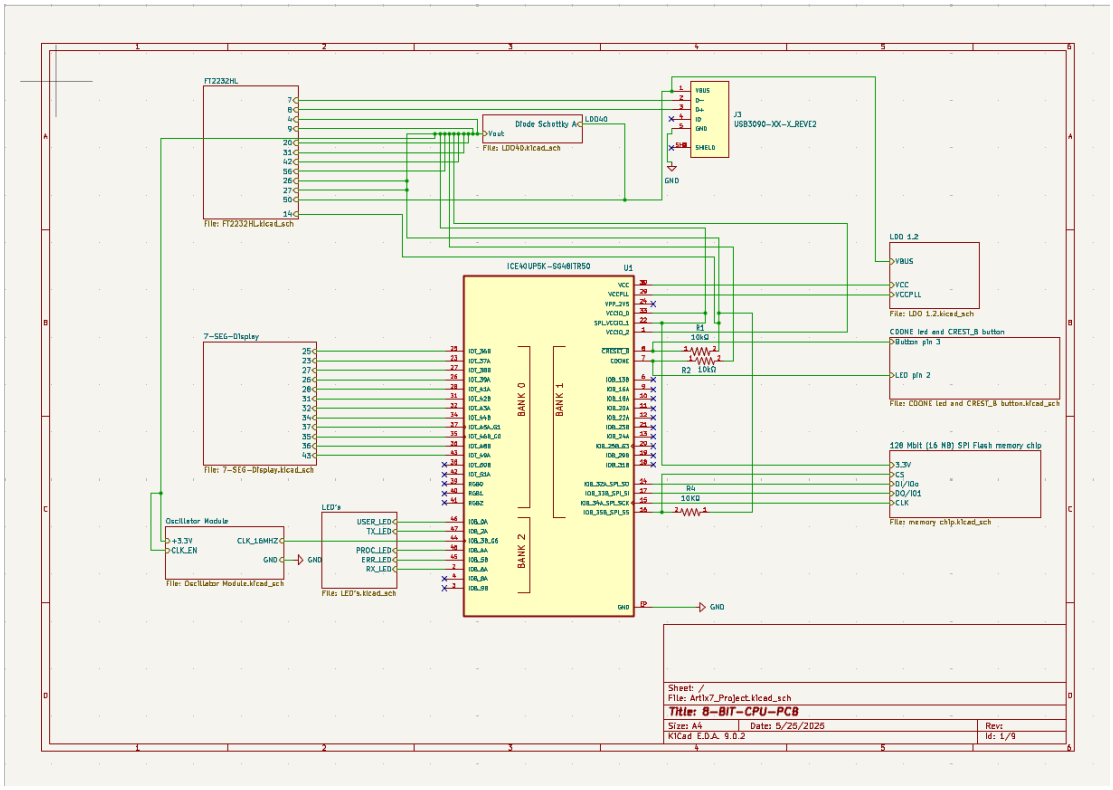


Figure 15: Schematic.

6.2 PCB Layout and Component Placement

6.2.1 Layout Strategy

The FPGA development board uses a standard two-layer PCB stack up: the top layer is used for component placement and routing, while the bottom layer acts as a ground plane to improve signal integrity and reduce noise.

Key components are placed with attention to power flow and signal proximity. The FPGA is centered to allow efficient routing to peripherals. The USB port and FT2232HL are placed along the top edge for easy cable access, while voltage regulators are located near the FPGA's power pins to minimize trace lengths.

Critical signal paths such as USB data lines, SPI connections, and the oscillator clock were routed manually to ensure signal quality and avoid EMI. Less sensitive connections were routed using Free Routing and then manually refined.

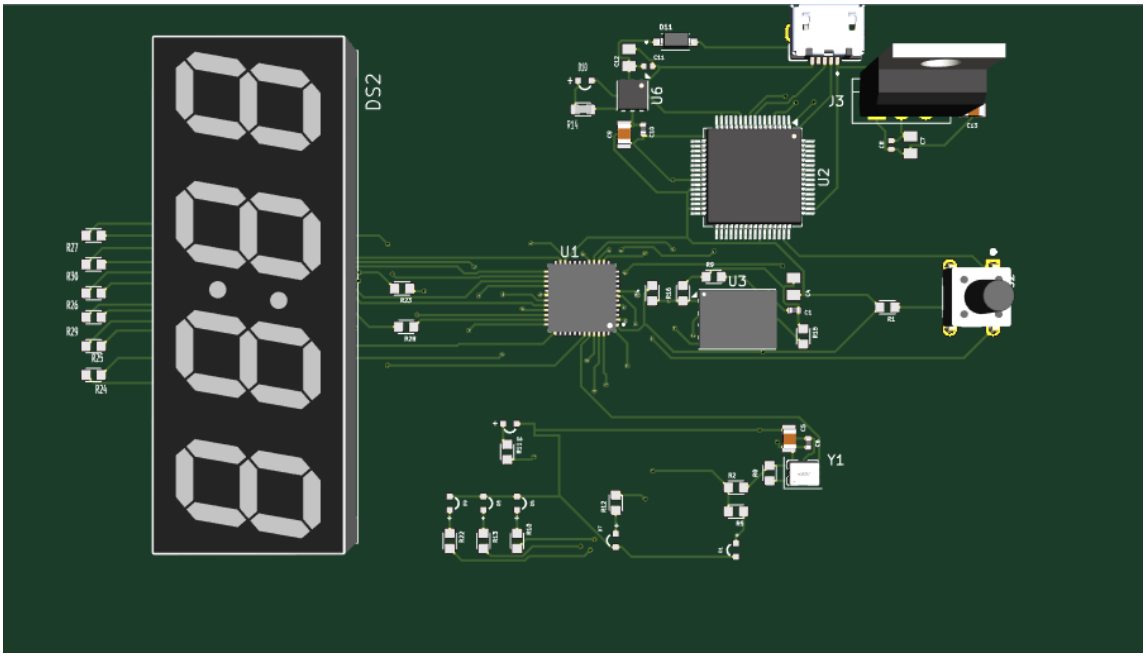
A solid copper fills on the bottom layer serves as the ground plane, with multiple vias connecting component grounds for low impedance. Four mounting holes are included at the corners for mechanical support in enclosures.

6.3 Validation and Manufacturing Files

Before exporting for manufacturing, the standard ERC and DRC checks were done and once they successfully passed, the gerber and drill files needed for manufacturing were generated.

7. Results

This section discusses the results of the project starting from the output seen in FPGA board's seven segment display to the PCB layout designed.



The

Figure 17: 3D View of PCB with components.

FPGA was able to show output as expected in all instruction cases. However, there was one issue when trying to store data into the same address from which data is already loaded from. For example, loading 5 from address 0b05 and storing a data into address 0b05 after some computation such as Addition or Subtraction resulted in all anodes and cathodes in display to be low which resulted in them being illuminated. This means reading from and writing to the same address even in different clock cycles wasn't sustainable. However, testbench verification showed that the digital design was correct. This is likely a storage error or a clock synchronization error. One workaround that was thought of but could not implemented due to limited semester time was to use the physical ROM of the FPGA to store the data.

Figure 17 shows the PCB Layout in 3D View along with the components. As can be seen all routing rules were maintained.

8. References and Sources

[1] Andrew S. Tanenbaum, Todd Austin, STRUCTURED COMPUTER ORGANIZATION, 1996.
[2] ModelSim-Intel® FPGAs Standard Edition Software Version 20.1.1
[3] Vivado 2024.2.
[4] KiCad 9.0.3.
[5] Github Repository.