Introduction:

This lab focuses on the implementation and design of a simple CPU in a VHDL environment, which will be implemented on an FPGA board. The CPU consists of an arithmetic and logic unit (ALU), a control unit, a memory and an input/output unit. The ALU performs various arithmetic and logical operations on 2 8-bit inputs A and B and produces an output. The control unit then supplies the opcodes to the ALU, which defines the operations to be performed. The memory is simulated by a finite-state machine (FSM), which generates the current state signal that controls the ALU's operation.

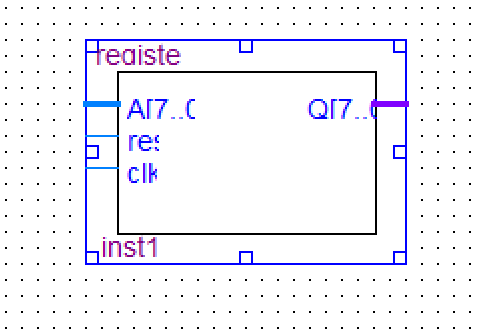
Components:

During this lab the following components were used, Latch1 and 2, FSM, and a 4to16 decoder.

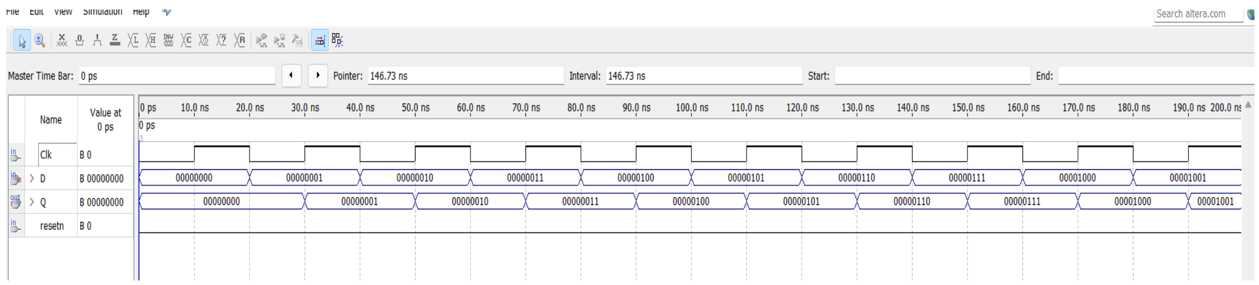
Basic Latch (Latch 1)

The basic latch outlined in this lab is the main component of the storage unit. It takes in a binary number as an input and gives it out as an output for each cycle.
The circuit uses 2 basic latches for each binary number.

Circuit Diagram for Latch:



Waveform for Latch:



Master Time Bar: 0 ps Pointer: 146.73 ns Interval: 146.73 ns Start: End:

```
Name     Value at 0 ps   0 ps  10.0 ns  20.0 ns  30.0 ns  40.0 ns  50.0 ns  60.0 ns  70.0 ns  80.0 ns  90.0 ns  100.0 ns  110.0 ns  120.0 ns  130.0 ns  140.0 ns  150.0 ns  160.0 ns  170.0 ns  180.0 ns  190.0 ns  200.0 ns
Clk      B 0
D        B 00000000      00000000  00000001  00000010  00000011  00000100  00000101  00000110  00000111  00001000  00001001
Q        B 00000000      00000000  00000001  00000010  00000011  00000100  00000101  00000110  00000111  00001000  00001001
resetn   B 0
```

Truth Table for Latch1 (and Latch2):

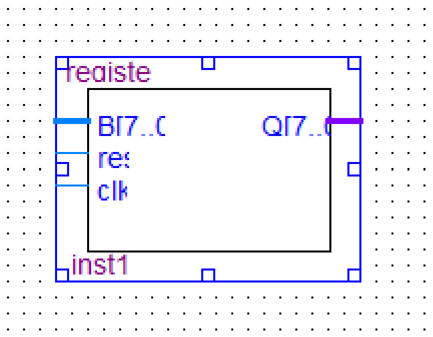
CLK	D	Q(t+1)
0	X	Q(t)
1	0	0
1	1	1

Latch 2

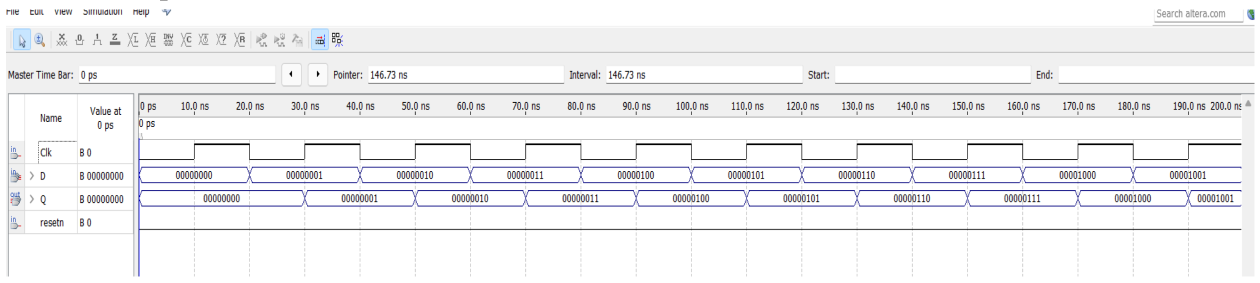
Description:

Latch 2 functions similarly to Latch 1 but is used for a different input channel. This modular design allows the system to process multiple data streams concurrently. The circuit diagram and truth table illustrate its role in the CPU, and the waveform file confirms its operation under clocked conditions.

Component's Circuit Diagram:



Component Waveform:

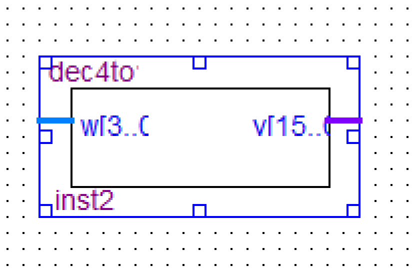


```
Master Time Bar: 0 ps      Pointer: 146.73 ns      Interval: 146.73 ns      Start:      End:

Name     Value at 0 ps   0 ps  10.0 ns  20.0 ns  30.0 ns  40.0 ns  50.0 ns  60.0 ns  70.0 ns  80.0 ns  90.0 ns  100.0 ns  110.0 ns  120.0 ns  130.0 ns  140.0 ns  150.0 ns  160.0 ns  170.0 ns  180.0 ns  190.0 ns  200.0 ns
Clk      B 0
D        B 00000000      00000000  00000001  00000010  00000011  00000100  00000101  00000110  00000111  00001000  00001001
Q        B 00000000      00000000  00000001  00000010  00000011  00000100  00000101  00000110  00000111  00001000  00001001
resetn   B 0
```

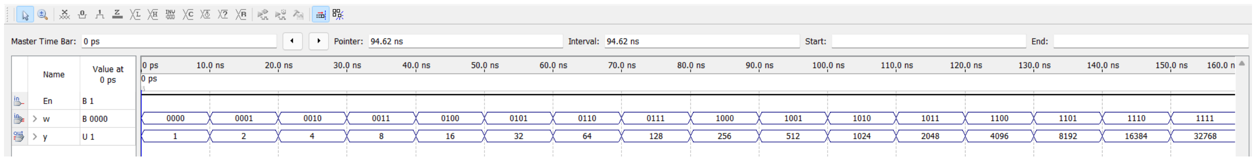
4:16 Decoder

The 4:16 Decoder was a component designed to take in the 4-bit state output of the FSM and give out a unique 16-bit output for each state, which selects one operation for the ALU. The mapping facilitates the control of multiple ALU functions using a compact opcode format. The decoder's truth table, block diagram, and simulation waveforms demonstrate its role in the control unit.

Component's Circuit Diagram:

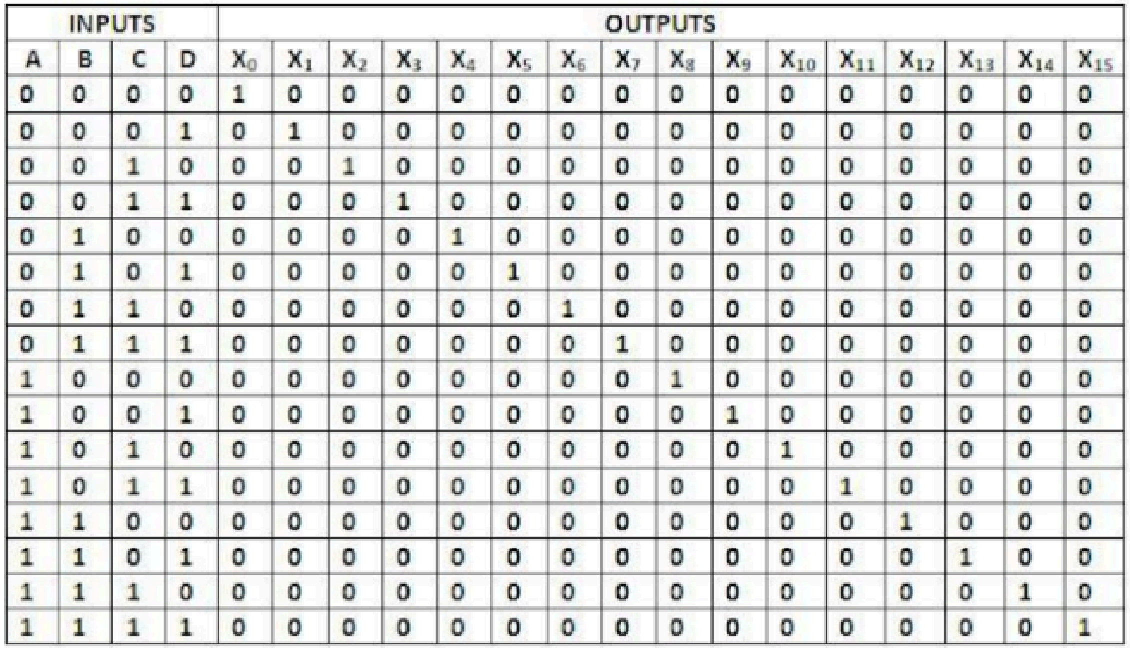


Waveform:



```
Master Time Bar: 0 ps      Pointer: 94.62 ns      Interval: 94.62 ns      Start:      End:

Name   Value at 0 ps   0 ps   10.0 ns  20.0 ns  30.0 ns  40.0 ns  50.0 ns  60.0 ns  70.0 ns  80.0 ns  90.0 ns  100.0 ns  110.0 ns  120.0 ns  130.0 ns  140.0 ns  150.0 ns  160.0 n
En     B 1
> w    B 0000          0000   0001     0010     0011     0100     0101     0110     0111     1000     1001     1010      1011      1100      1101      1110      1111
> y    U 1             1      2        4        8        16       32       64       128      256      512      1024      2048      4096      8192      16384     32768
```

Truth Table for Component:

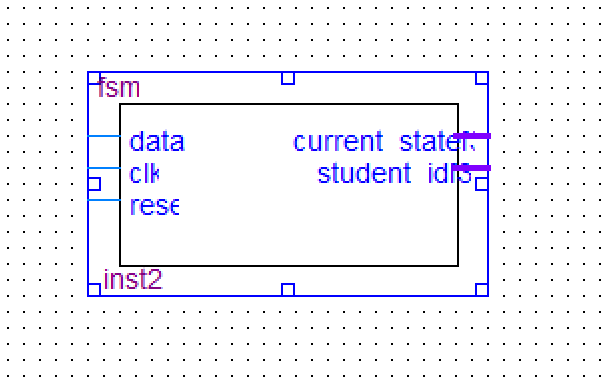


INPUTS				OUTPUTS															
A	B	C	D	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

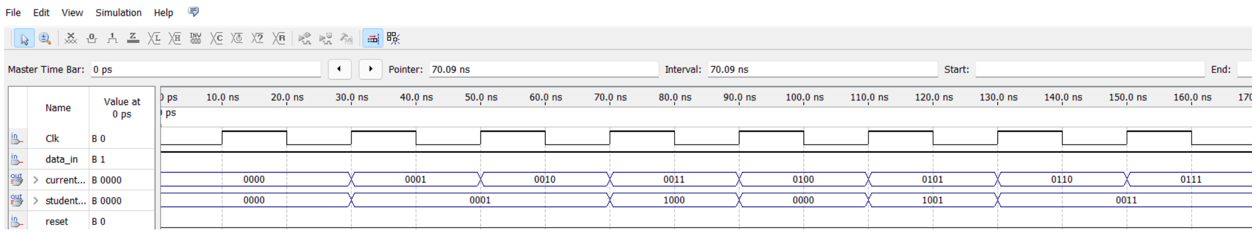
Finite State Machine (FSM)

The FSM serves as a program counter, cycling through predefined states to sequence operations. Each state corresponds to an opcode fed into the decoder. The FSM ensures orderly execution of instructions, with a state transition diagram and simulation results highlighting its operation. Modifications were made to adapt the FSM to our student numbers.

Circuit Diagram:



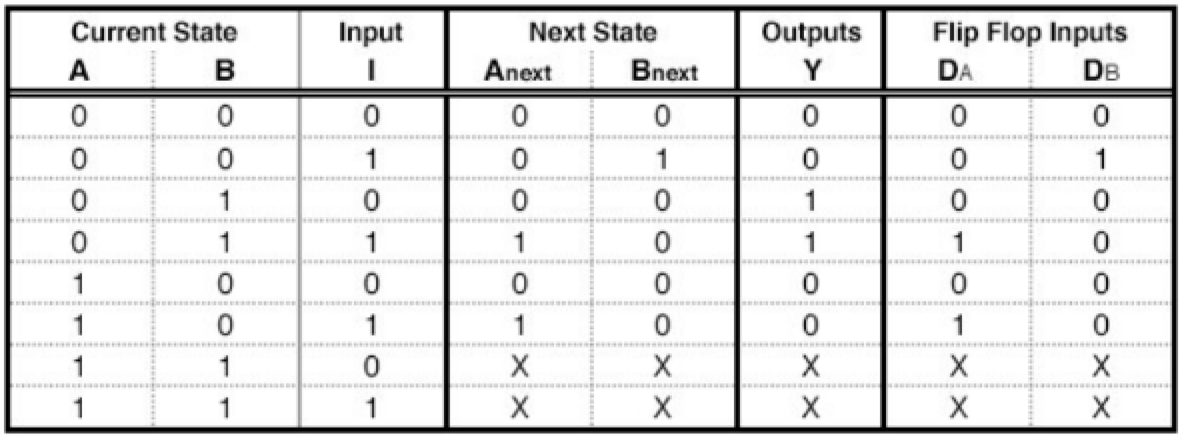
Waveform:


File Edit View Simulation Help
0 ps 10.0 ns 20.0 ns 30.0 ns 40.0 ns 50.0 ns 60.0 ns 70.0 ns 80.0 ns 90.0 ns 100.0 ns 110.0 ns 120.0 ns 130.0 ns 140.0 ns 150.0 ns 160.0 ns
0000 0001 0010 0011 0100 0101 0110 0111
0000 0001 1000 0000 1001 0011

Master Time Bar: 0 ps   Pointer: 70.09 ns   Interval: 70.09 ns   Start: End:

Name	Value at 0 ps
Clk	B 0
data_in	B 1
current...	B 0000
student...	B 0000
reset	B 0

Truth Table for Component:



Current State		Input	Next State		Outputs	Flip Flop Inputs	
A	B	I	Anext	Bnext	Y	DA	DB
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1
0	1	0	0	0	1	0	0
0	1	1	1	0	1	1	0
1	0	0	0	0	0	0	0
1	0	1	1	0	0	1	0
1	1	0	X	X	X	X	X
1	1	1	X	X	X	X	X

Arithmetic Logical Unit (ALU)

ALU 1:

The ALU (Arithmetic Logic Unit) serves as the processor that determines the operation to perform on the inputs based on the state of the FSM (Finite State Machine). It has five inputs:

1. **Clock**
2. **A** (binary input)
3. **B** (binary input)
4. **student_id**
5. **OP** (operation code)

Inputs **A** and **B** are binary numbers that act as the inputs to the storage unit, provided that **data_in** is set to 1. The **Clock** input toggles between 0 and 1. On a rising edge (when the clock transitions from 0 to 1), the ALU reads the value of the **OP** input, which is produced by the decoder. The ALU then executes a switch statement that matches the value of **OP** to a corresponding case, performing the specified operation on **A** and **B** as outlined in the table provided in the lab manual.

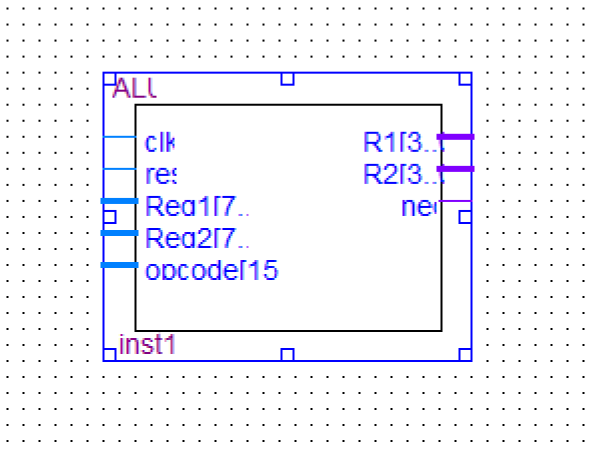
In this application of the ALU, student_id is not used.

There are 3 outputs for the ALU;

1. Neg
2. R1
3. R2

Neg outputs 1 if the number is negative and lights up an LED on the 7-segment display. R1 and R2 are 4-bit outputs, which can give an 8-bit output when combined. Each bit goes to the 7-segment display to output the result in hexadecimal.

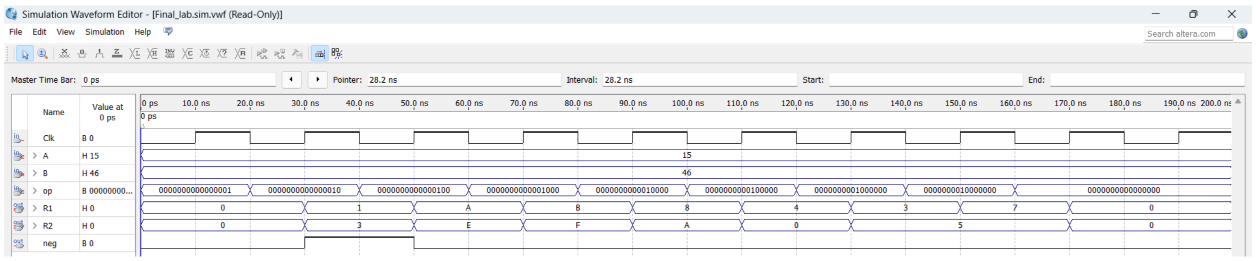
BDF:



Microcode, and Waveform

Function #	Microcode	Boolean Operation / Function
1	0000000000000001	sum(A, B)
2	0000000000000010	diff(A, B)
3	0000000000000100	$\overline{A}$
4	0000000000001000	$\overline{A \cdot B}$
5	0000000000010000	$\overline{A + B}$
6	0000000000100000	$A \cdot B$
7	0000000001000000	$A \oplus B$
8	0000000010000000	$A + B$
9	0000000100000000	$\overline{A \oplus B}$

Table 3.1: ALU problem set 1 microcode.


Simulation Waveform Editor - [Final_lab.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 28.2 ns
Interval: 28.2 ns
Start:
End:
Name
Value at 0 ps
0 ps
0 ps 10.0 ns 20.0 ns 30.0 ns 40.0 ns 50.0 ns 60.0 ns 70.0 ns 80.0 ns 90.0 ns 100.0 ns 110.0 ns 120.0 ns 130.0 ns 140.0 ns 150.0 ns 160.0 ns 170.0 ns 180.0 ns 190.0 ns 200.0 ns
Clk B 0
A H 15 15
B H 46 46
op B 00000000... 0000000000000001 0000000000000010 0000000000000100 0000000000001000 0000000000010000 0000000000100000 0000000001000000 0000000010000000 0000000000000000
R1 H 0 0 1 A B 8 4 3 7 0
R2 H 0 0 3 E F A 0 5 0
neg B 0

ALU 2:

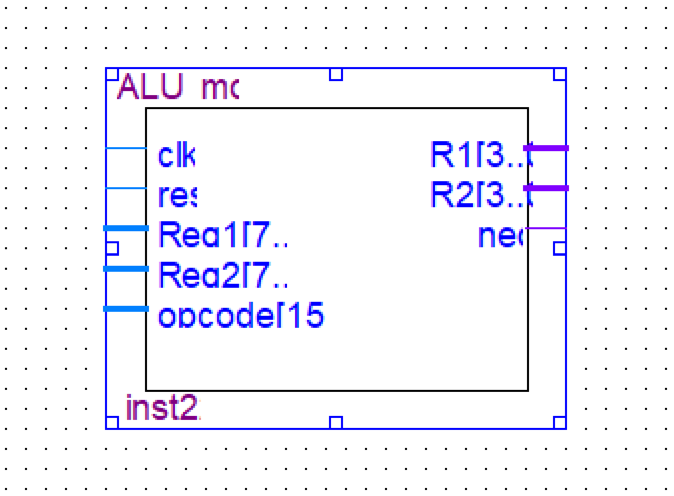
The Arithmetic Logic Unit (ALU) is the component responsible for determining the operation to perform on the inputs, guided by the state of the Finite State Machine (FSM). It accepts the following five inputs:

1. **Clock**
2. **A**
3. **B**
4. **student_id**
5. **OP**

Inputs **A** and **B** are binary numbers that serve as the inputs to the storage unit whenever **data_in** is set to 1. The **Clock** input alternates between 0 and 1. On the rising edge (when the Clock transitions from 0 to 1), the ALU reads the value of the **OP** input, which is the output of the decoder. It then executes a corresponding operation on **A** and **B** using a switch statement, where each case corresponds to a specific operation as outlined in the lab manual.

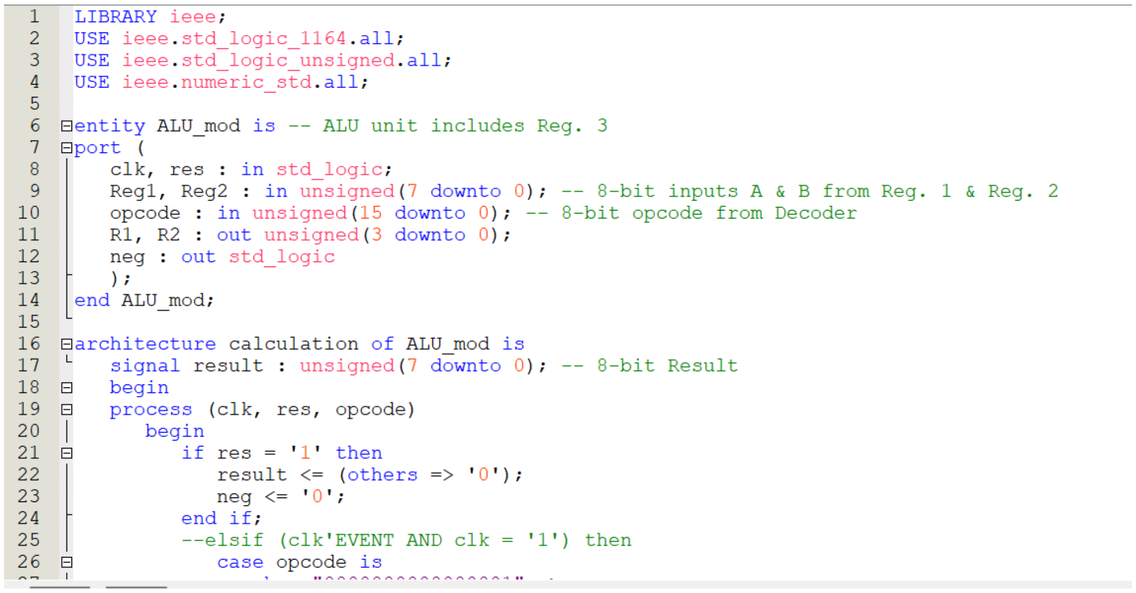
The **student_id** input serves no functional purpose in this part.

BDF:

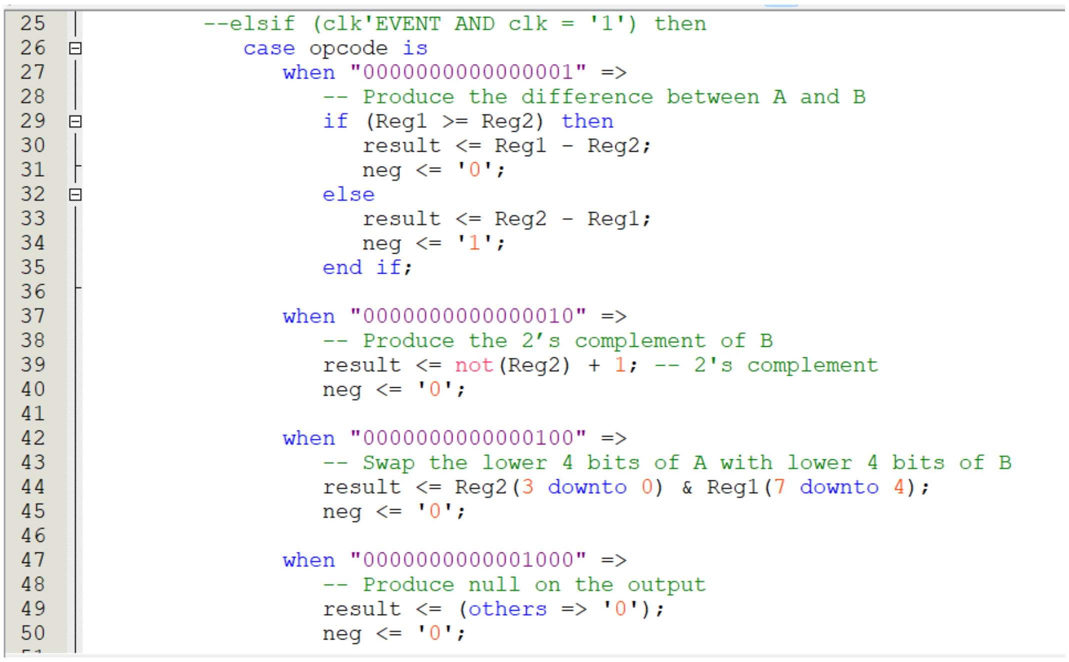


Microcode:

Function #	Operation / Function
1	Produce the difference between **A** and **B**
2	Produce the 2's complement of **B**
3	Swap the lower 4 bits of **A** with lower 4 bits of **B**
4	Produce null on the output
5	Decrement **B** by 5
6	Invert the bit-significance order of **A**
7	Shift **B** to left by three bits, input bit = 1 (SHL)
8	Increment **A** by 3

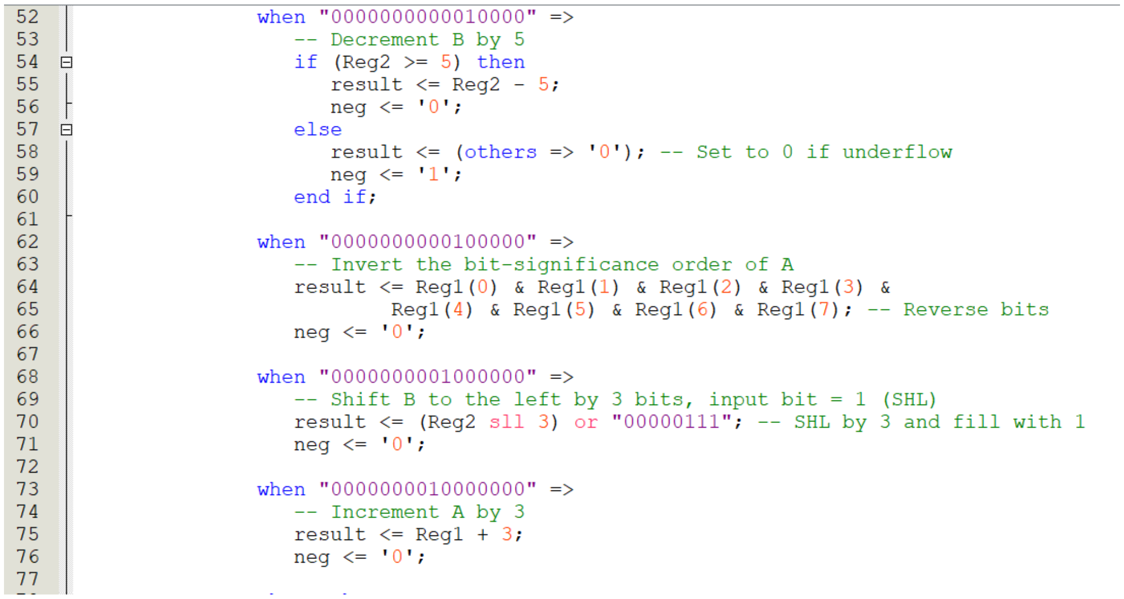


```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

entity ALU_mod is -- ALU unit includes Reg. 3
port (
    clk, res : in std_logic;
    Reg1, Reg2 : in unsigned(7 downto 0); -- 8-bit inputs A & B from Reg. 1 & Reg. 2
    opcode : in unsigned(15 downto 0); -- 8-bit opcode from Decoder
    R1, R2 : out unsigned(3 downto 0);
    neg : out std_logic
    );
end ALU_mod;

architecture calculation of ALU_mod is
    signal result : unsigned(7 downto 0); -- 8-bit Result
    begin
    process (clk, res, opcode)
        begin
            if res = '1' then
                result <= (others => '0');
                neg <= '0';
            end if;
            --elsif (clk'EVENT AND clk = '1') then
                case opcode is
```

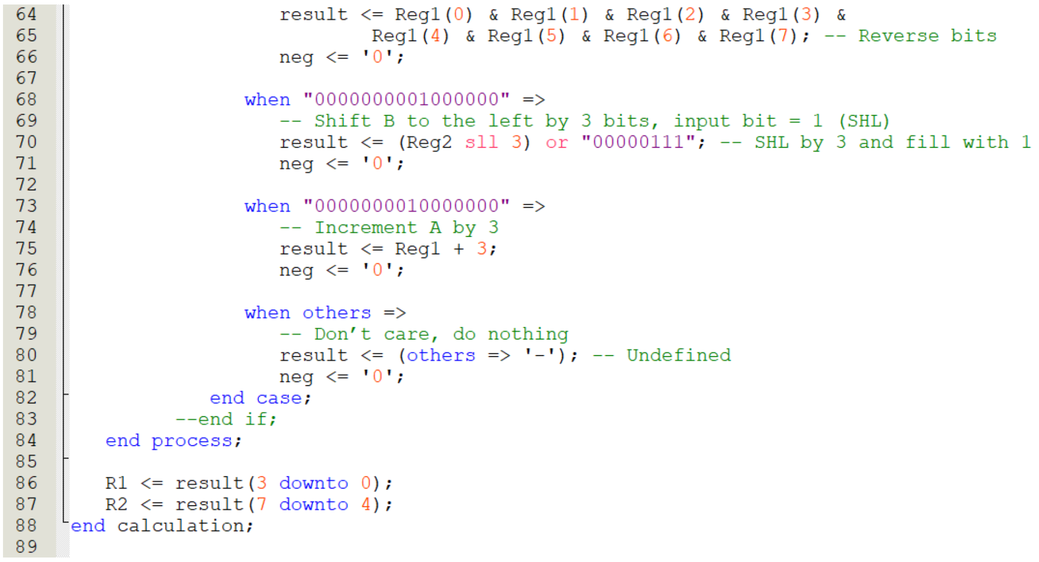


```vhdl
            --elsif (clk'EVENT AND clk = '1') then
                case opcode is
```

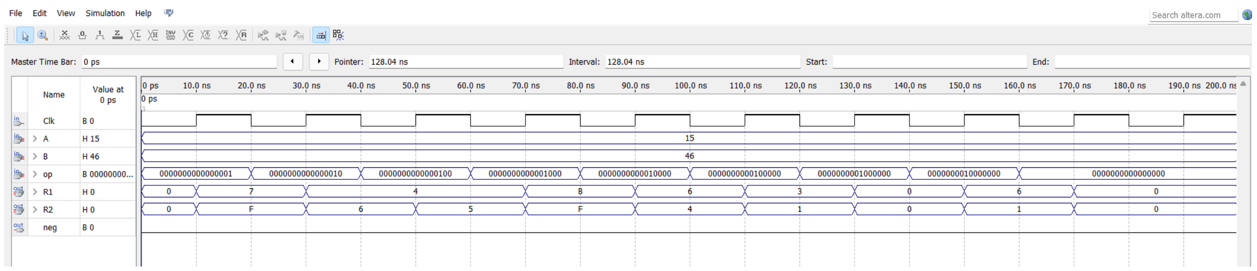
```vhdl
                    when "0000000000000001" =>
                        -- Produce the difference between A and B
                        if (Reg1 >= Reg2) then
                            result <= Reg1 - Reg2;
                            neg <= '0';
                        else
                            result <= Reg2 - Reg1;
                            neg <= '1';
                        end if;

                    when "0000000000000010" =>
                        -- Produce the 2's complement of B
                        result <= not(Reg2) + 1; -- 2's complement
                        neg <= '0';

                    when "0000000000000100" =>
                        -- Swap the lower 4 bits of A with lower 4 bits of B
                        result <= Reg2(3 downto 0) & Reg1(7 downto 4);
                        neg <= '0';

                    when "0000000000001000" =>
                        -- Produce null on the output
                        result <= (others => '0');
                        neg <= '0';
```



```vhdl
               when "0000000000010000" =>
                  -- Decrement B by 5
                  if (Reg2 >= 5) then
                     result <= Reg2 - 5;
                     neg <= '0';
                  else
                     result <= (others => '0'); -- Set to 0 if underflow
                     neg <= '1';
                  end if;

               when "0000000000100000" =>
                  -- Invert the bit-significance order of A
                  result <= Reg1(0) & Reg1(1) & Reg1(2) & Reg1(3) &
                          Reg1(4) & Reg1(5) & Reg1(6) & Reg1(7); -- Reverse bits
                  neg <= '0';

               when "0000000001000000" =>
                  -- Shift B to the left by 3 bits, input bit = 1 (SHL)
                  result <= (Reg2 sll 3) or "00000111"; -- SHL by 3 and fill with 1
                  neg <= '0';

               when "0000000010000000" =>
                  -- Increment A by 3
                  result <= Reg1 + 3;
                  neg <= '0';
```



```vhdl
                  result <= Reg1(0) & Reg1(1) & Reg1(2) & Reg1(3) &
                          Reg1(4) & Reg1(5) & Reg1(6) & Reg1(7); -- Reverse bits
                  neg <= '0';

               when "0000000001000000" =>
                  -- Shift B to the left by 3 bits, input bit = 1 (SHL)
                  result <= (Reg2 sll 3) or "00000111"; -- SHL by 3 and fill with 1
                  neg <= '0';

               when "0000000010000000" =>
                  -- Increment A by 3
                  result <= Reg1 + 3;
                  neg <= '0';

               when others =>
                  -- Don't care, do nothing
                  result <= (others => '-'); -- Undefined
                  neg <= '0';
            end case;
         --end if;
   end process;

   R1 <= result(3 downto 0);
   R2 <= result(7 downto 4);
end calculation;
```



Master Time Bar: 0 ps
Pointer: 128.04 ns
Interval: 128.04 ns
Start:
End:

Name	Value at 0 ps
Clk	B 0
A	H 15
B	H 46
op	B 00000000...
R1	H 0
R2	H 0
neg	B 0

ALU 3:

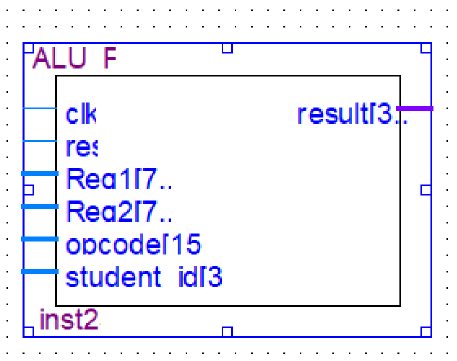
ALU from Problem 1 is used to incorporate an additional data input — the **student_id** — which is provided as an output from the FSM component of the Control Unit. The updated ALU3 processes three inputs: **Reg1**, **Reg2**, and **student_id**.

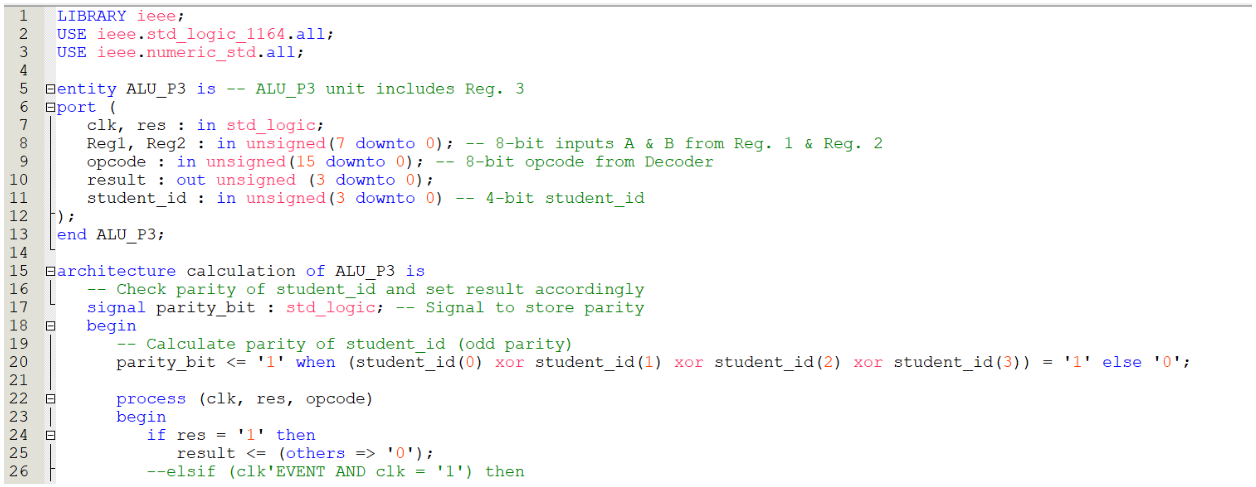
In this part, the **student_id** input is utilized by the ALU to determine if its least significant bit (LSB) is 1 or 0. Since an LSB of 1 indicates an odd number and an LSB of 0 indicates an even number, the ALU assigns a new output, **E**, which outputs 1 for odd numbers and 0 for even numbers. This **E** signal is then sent to the 7-segment display, where an **if statement** controls the display to show **"n"** for odd numbers and **"y"** for even numbers.

Function that was chosen: **FUNCTION 3 (below)**

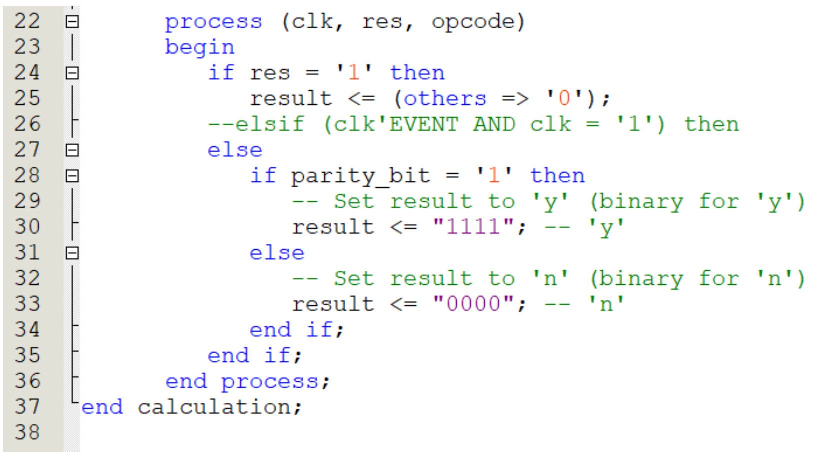
c) For each opcode submitted to the ALU, display 'y' if the **student_id** signal has an odd parity and 'n' otherwise

BDF Screenshot:





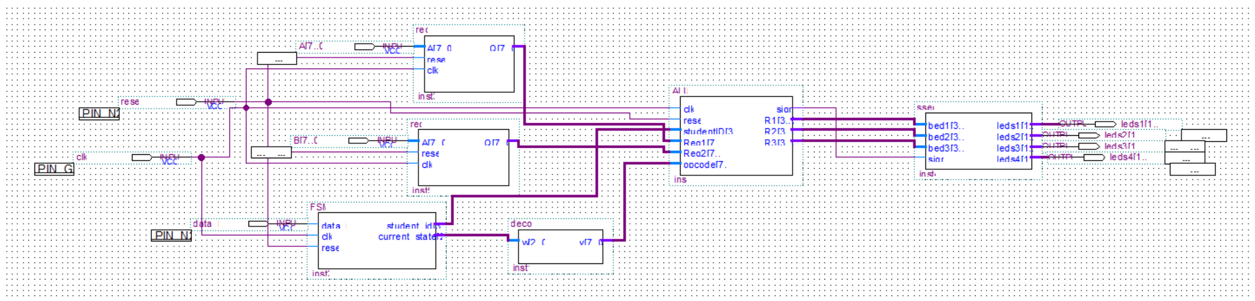
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity ALU_P3 is -- ALU_P3 unit includes Reg. 3
port (
    clk, res : in std_logic;
    Reg1, Reg2 : in unsigned(7 downto 0); -- 8-bit inputs A & B from Reg. 1 & Reg. 2
    opcode : in unsigned(15 downto 0); -- 8-bit opcode from Decoder
    result : out unsigned (3 downto 0);
    student_id : in unsigned(3 downto 0) -- 4-bit student_id
);
end ALU_P3;

architecture calculation of ALU_P3 is
    -- Check parity of student_id and set result accordingly
    signal parity_bit : std_logic; -- Signal to store parity
    begin
        -- Calculate parity of student_id (odd parity)
        parity_bit <= '1' when (student_id(0) xor student_id(1) xor student_id(2) xor student_id(3)) = '1' else '0';

        process (clk, res, opcode)
        begin
            if res = '1' then
                result <= (others => '0');
            --elsif (clk'EVENT AND clk = '1') then
```



```
        process (clk, res, opcode)
        begin
            if res = '1' then
                result <= (others => '0');
            --elsif (clk'EVENT AND clk = '1') then
```

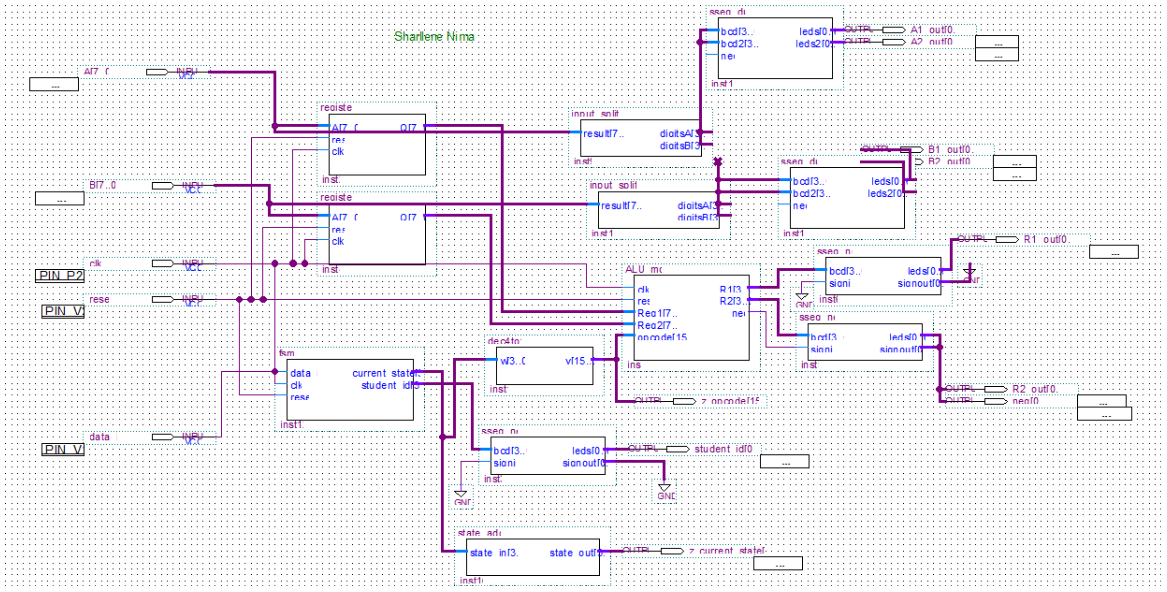
```
            else
                if parity_bit = '1' then
                    -- Set result to 'y' (binary for 'y')
                    result <= "1111"; -- 'y'
                else
                    -- Set result to 'n' (binary for 'n')
                    result <= "0000"; -- 'n'
                end if;
            end if;
        end process;
end calculation;
```

Circuit Diagrams:



Circuit Version 1 with ALU 1

After realizing that the circuit diagram needed to be changed for ALU2 and 3, it was completely redesigned to the one below.



Circuit Diagram for ALU 2

References

Brown, S. D., & Vranesic, Z. G. (2009). *Fundamentals of Digital Logic with VHDL Design*. New York, United States: McGraw-Hill Education.

Conclusion:

The successful design and implementation of a simple Central Processing Unit (CPU) provided valuable insight into the fundamental components and operation of modern computer architecture. This lab emphasized the integration and coordination of key subcomponents, including the Arithmetic Logic Unit (ALU), Control Unit (incorporating a finite state machine and decoder), registers, and input/output displays.

Through the design and synthesis process, concepts such as data storage, control sequencing, and functional simulation were applied. By using VHDL for hardware description, the CPU's operational logic was effectively translated into a functional hardware model, capable of executing arithmetic and logical operations. The structured approach of developing each subcomponent independently before integrating them into a complete system reinforced the modular nature of digital design.

Additionally, functional testing and simulation using Quartus software allowed for the verification and validation of the CPU's performance. The waveform analysis confirmed that the CPU correctly executed the specified operations, cycling through control states and producing accurate results on the 7-segment displays.

This lab reinforced key principles of digital system design, including modular design, synchronization, and binary arithmetic. The hands-on experience with FPGA implementation bridged the gap between theoretical knowledge and practical application. The ability to design, simulate, and test a functioning CPU lays a strong foundation for more complex digital design projects in the future.