



**Department of Electrical,
Computer, & Biomedical Engineering**
Faculty of Engineering & Architectural Science

Course Title:	COE
Course Number:	328
Semester/Year (e.g.F2016)	F2024

Instructor:	Arghavan Asad
--------------------	----------------------

<i>Assignment/Lab Number:</i>	6
<i>Assignment/Lab Title:</i>	<i>Design of A Simple Central Processing Unit</i>

<i>Submission Date</i> :	2024-12-12
<i>Due Date:</i>	<i>2024-12-12</i>

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Nimalraaj	Sharllene	501180933	21	Sharllene Nimalraaj

Table of Contents

1. Introduction

- a. Description of the overall lab and the components

2. Components

- a. Description of components: Latch1, Latch2, 4:16 Decoder and FSM

3. Problem Set 1 ALU

- a. Description of the problem
- b. Purpose of inputs and outputs
- c. Table of Microcodes
- d. Block schematic and waveform

4. Problem Set 2 ALU

- a. Description of the problem
- b. Purpose of inputs and outputs
- c. Table of Microcodes
- d. Block schematic and waveform

5. Problem Set 3 ALU

- a. Description of the problem
- b. Purpose of inputs and outputs
- c. Table of Microcodes
- d. Block schematic and waveform

6. Conclusion

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.

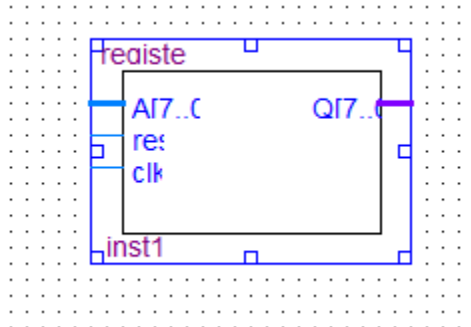
Components:

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

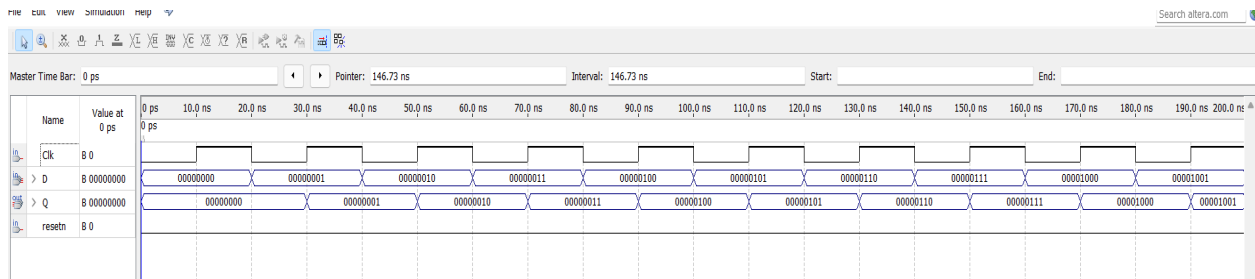
Basic Latch (Latch 1)

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:



Truth Table for Latch1 (and Latch2):

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

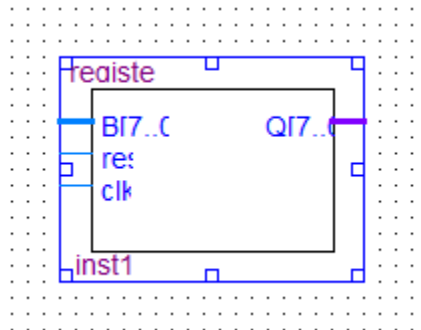
Latch 2

Description:

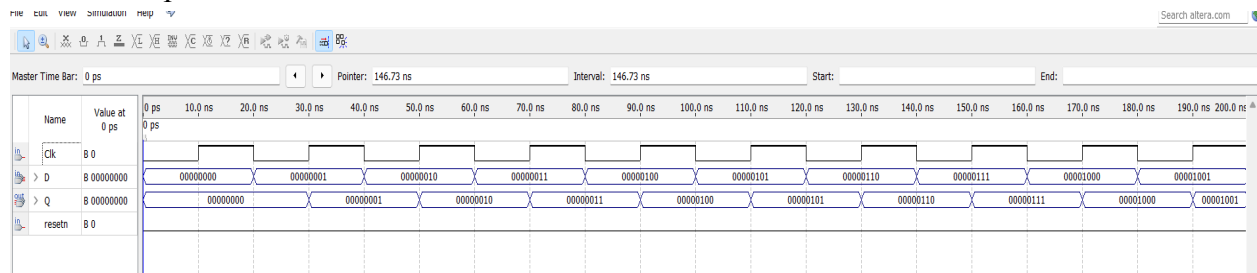
Latch 2 functions similarly to Latch 1 but is used for a different input channel. This modular design allows the system to process multiple data streams concurrently. The circuit

diagram and truth table illustrate its role in the CPU, and the waveform file confirms its operation under clocked conditions.

Component's Circuit Diagram:



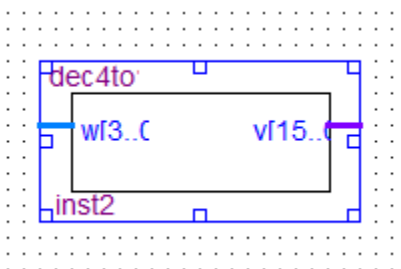
Component Waveform:



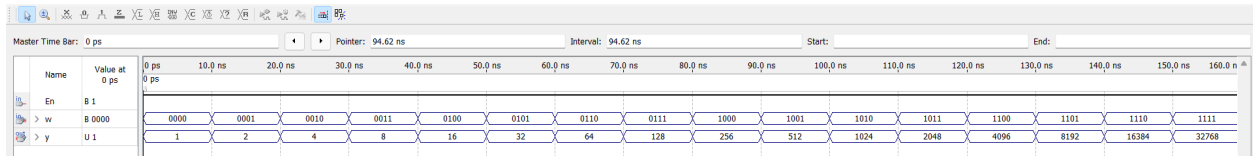
4:16 Decoder

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:



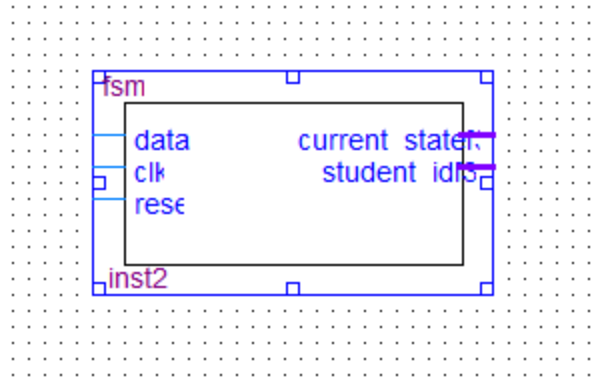
Truth Table for Component:

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

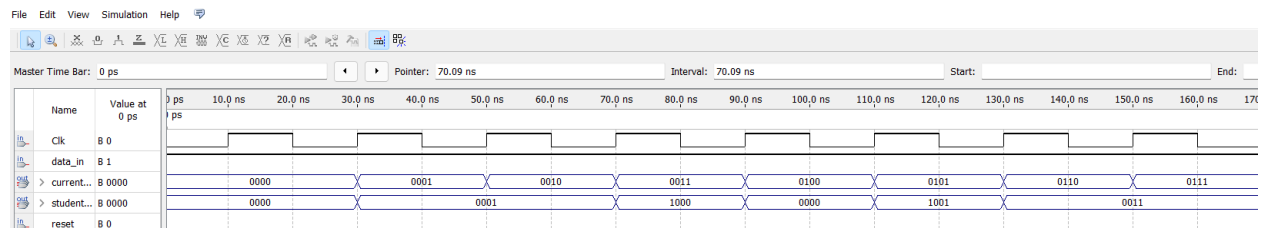
Finite State Machine (FSM)

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:



Truth Table for Component:

Current State		Input I	Next State		Outputs Y	Flip Flop Inputs	
A	B		A _{next}	B _{next}		D _A	D _B
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.

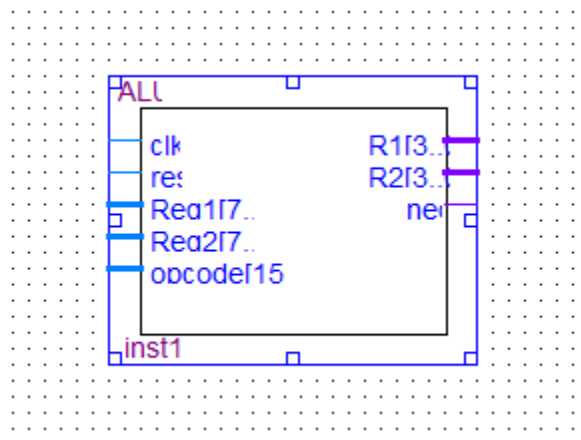
In this application of the ALU, **student_id** is not used.

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.

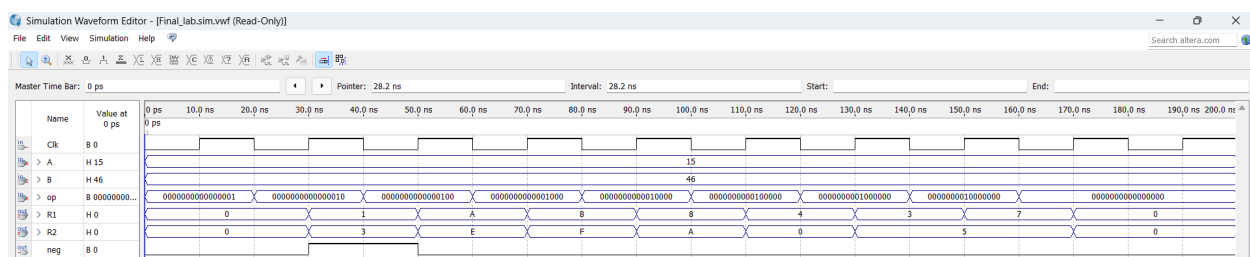
BDF:



Microcode, and Waveform

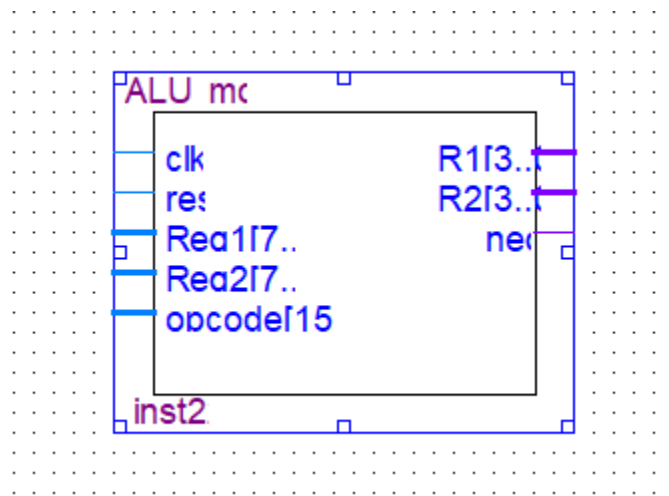
Function #	Microcode	Boolean Operation / Function
1	0000000000000001	$\text{sum}(A, B)$
2	0000000000000010	$\text{diff}(A, B)$
3	0000000000000100	\overline{A}
4	0000000000001000	$\overline{A \cdot B}$
5	0000000000010000	$\overline{A + B}$
6	0000000000100000	$A \cdot B$
7	0000000001000000	$A \oplus B$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

Table 3.1: ALU problem set 1 microcode.



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

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.std_logic_unsigned.all;
4  USE ieee.numeric_std.all;
5
6  entity ALU_mod is -- ALU unit includes Reg. 3
7  port (
8      clk, res : in std_logic;
9      Reg1, Reg2 : in unsigned(7 downto 0); -- 8-bit inputs A & B from Reg. 1 & Reg. 2
10     opcode : in unsigned(15 downto 0); -- 8-bit opcode from Decoder
11     R1, R2 : out unsigned(3 downto 0);
12     neg : out std_logic
13 );
14 end ALU_mod;
15
16 architecture calculation of ALU_mod is
17     signal result : unsigned(7 downto 0); -- 8-bit Result
18     begin
19     process (clk, res, opcode)
20     begin
21         if res = '1' then
22             result <= (others => '0');
23             neg <= '0';
24         end if;
25         --elsif (clk'EVENT AND clk = '1') then
26         case opcode is
27             -- "0000000000000001" =>
28             -- Produce the difference between A and B
29             if (Reg1 >= Reg2) then
30                 result <= Reg1 - Reg2;
31                 neg <= '0';
32             else
33                 result <= Reg2 - Reg1;
34                 neg <= '1';
35             end if;
36
37             when "0000000000000010" =>
38                 -- Produce the 2's complement of B
39                 result <= not(Reg2) + 1; -- 2's complement
40                 neg <= '0';
41
42             when "0000000000000100" =>
43                 -- Swap the lower 4 bits of A with lower 4 bits of B
44                 result <= Reg2(3 downto 0) & Reg1(7 downto 4);
45                 neg <= '0';
46
47             when "0000000000001000" =>
48                 -- Produce null on the output
49                 result <= (others => '0');
50                 neg <= '0';

```

```

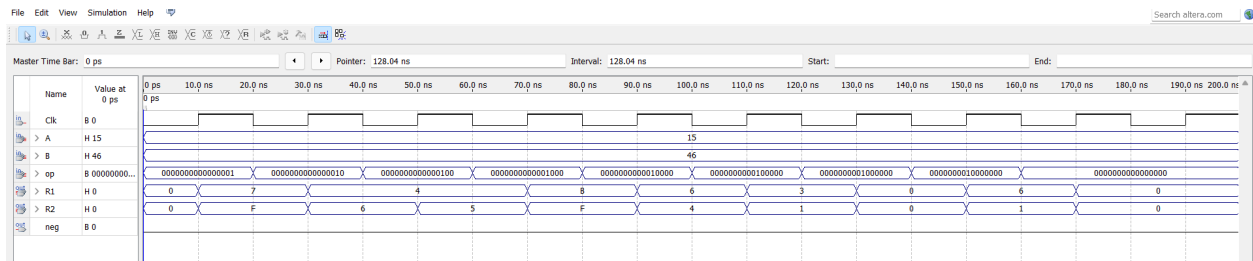
52 when "000000000010000" =>
53     -- Decrement B by 5
54     if (Reg2 >= 5) then
55         result <= Reg2 - 5;
56         neg <= '0';
57     else
58         result <= (others => '0'); -- Set to 0 if underflow
59         neg <= '1';
60     end if;
61
62 when "000000000100000" =>
63     -- Invert the bit-significance order of A
64     result <= Reg1(0) & Reg1(1) & Reg1(2) & Reg1(3) &
65         Reg1(4) & Reg1(5) & Reg1(6) & Reg1(7); -- Reverse bits
66     neg <= '0';
67
68 when "000000001000000" =>
69     -- Shift B to the left by 3 bits, input bit = 1 (SHL)
70     result <= (Reg2 sll 3) or "00000111"; -- SHL by 3 and fill with 1
71     neg <= '0';
72
73 when "0000000010000000" =>
74     -- Increment A by 3
75     result <= Reg1 + 3;
76     neg <= '0';
77

```

```

64     result <= Reg1(0) & Reg1(1) & Reg1(2) & Reg1(3) &
65         Reg1(4) & Reg1(5) & Reg1(6) & Reg1(7); -- Reverse bits
66     neg <= '0';
67
68 when "000000001000000" =>
69     -- Shift B to the left by 3 bits, input bit = 1 (SHL)
70     result <= (Reg2 sll 3) or "00000111"; -- SHL by 3 and fill with 1
71     neg <= '0';
72
73 when "0000000010000000" =>
74     -- Increment A by 3
75     result <= Reg1 + 3;
76     neg <= '0';
77
78 when others =>
79     -- Don't care, do nothing
80     result <= (others => '-'); -- Undefined
81     neg <= '0';
82 end case;
83 --end if;
84 end process;
85
86 R1 <= result(3 downto 0);
87 R2 <= result(7 downto 4);
88 end calculation;
89

```



ALU 3:

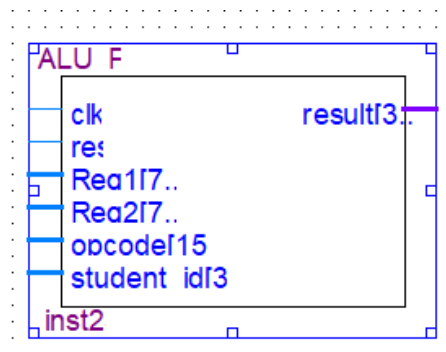
ALU from Problem 1 is used to incorporate an additional data input — the **student_id** — which is provided as an output from the FSM component of the Control Unit. The updated ALU3 processes three inputs: **Reg1**, **Reg2**, and **student_id**.

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:

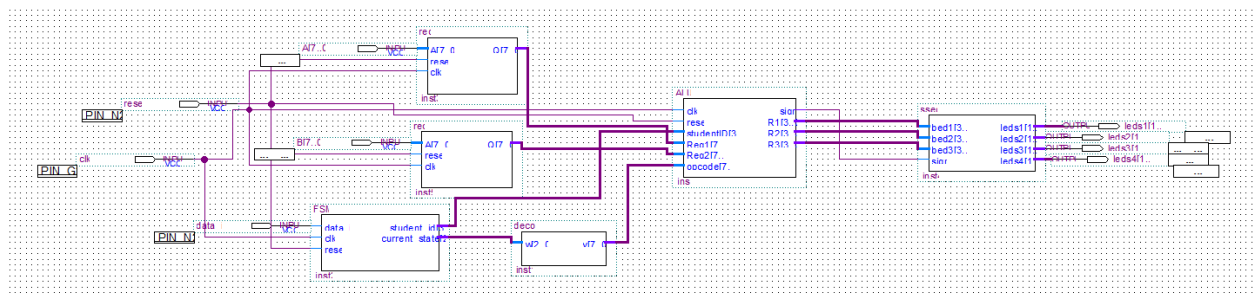


```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.numeric_std.all;
4
5 entity ALU_P3 is -- ALU_P3 unit includes Reg. 3
6 port (
7     clk, res : in std_logic;
8     Reg1, Reg2 : in unsigned(7 downto 0); -- 8-bit inputs A & B from Reg. 1 & Reg. 2
9     opcode : in unsigned(15 downto 0); -- 8-bit opcode from Decoder
10    result : out unsigned(3 downto 0);
11    student_id : in unsigned(3 downto 0) -- 4-bit student_id
12);
13 end ALU_P3;
14
15 architecture calculation of ALU_P3 is
16 -- Check parity of student_id and set result accordingly
17 signal parity_bit : std_logic; -- Signal to store parity
18 begin
19     -- Calculate parity of student_id (odd parity)
20     parity_bit <= '1' when (student_id(0) xor student_id(1) xor student_id(2) xor student_id(3)) = '1' else '0';
21
22     process (clk, res, opcode)
23     begin
24         if res = '1' then
25             result <= (others => '0');
26             --elsif (clk'EVENT AND clk = '1') then
27
28                 process (clk, res, opcode)
29                 begin
30                     if res = '1' then
31                         result <= (others => '0');
32                         --elsif (clk'EVENT AND clk = '1') then
33
34                             if parity_bit = '1' then
35                                 -- Set result to 'y' (binary for 'y')
36                                 result <= "1111"; -- 'y'
37
38                             else
39                                 -- Set result to 'n' (binary for 'n')
40                                 result <= "0000"; -- 'n'
41
42                             end if;
43                         end if;
44                     end process;
45                 end calculation;
46             end if;
47         end process;
48     end calculation;
49 end

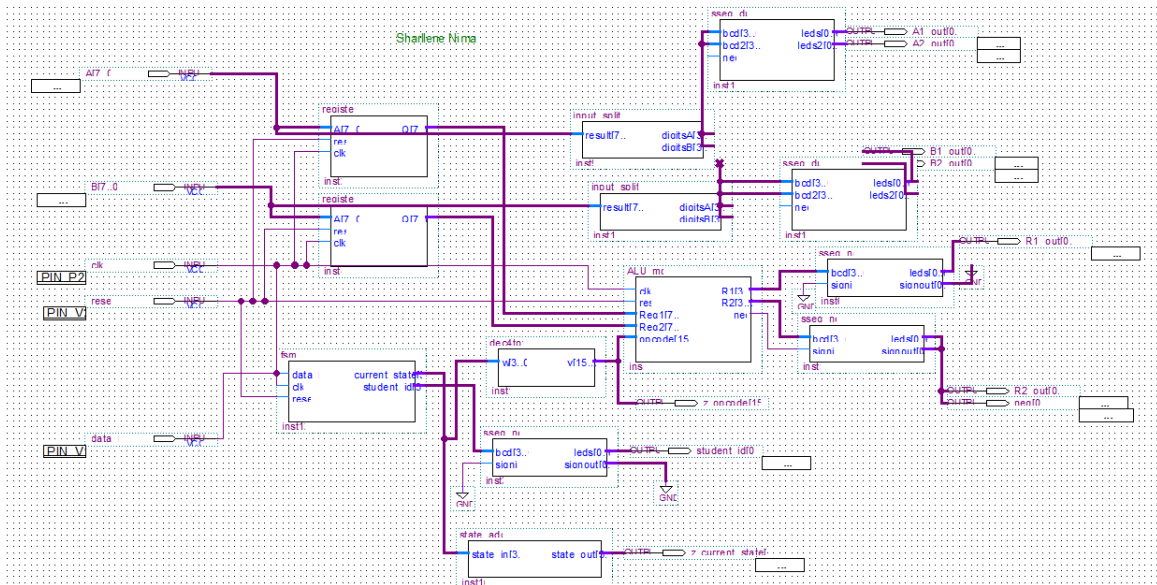
```

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.



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.