Design of a Simple General- Purpose Processor

COE 328
Section 09
Arghavan Asad
Rayyan Faisal
501172024

Table of Contents

Toronto Metropolitan University

Introduction

In this lab a Simple Processor was made using VHDL code and implementing it on an FPGA board. 4 main parts were used to make the Processor, The Storage Unit, an ALU core, a Control Unit and Seven Segment Display units. In this lab the Storage Unit is built using two- 8 bit registers which in this case are Latches and are used to store two inputs that were named A and B for temporary time. The register reads these inputs and passes the input onto the output. The next vital part, the Control Unit, which contains the FSM and 4:16 decoder determines the microcode that will be sent to the ALU. The ALU then takes the input from the Control Unit and this performs an operation for A and B. The operation that was used for this lab will be further discussed later. Finally, the Seven Segment Display displays the output taken from the ALU which is the results of the operation for the inputs A and B. All of these components are combined together to make this simple processor work. The specifics of each component such as schematic, truth tables and waveforms will be further discussed in the lab report.

Components

Latch 1 and Latch 2:

This latch was one of the components that is part of the register. A latch is a type of digital circuit that can store one or more bits of information. It is used to maintain a data value of high or low as long as current is maintained in the circuit. In this case Latch 1 stores the 8-bit input A. The register reads the bit values on its input on the rising edge of the clock signal and passes those bit values to the output port on the next rising edge of the clock signal. To find the VHDL code of the Latch please refer to **Appendix A**. Both Latch 1 and 2 have the same code

Latch 2 has the same purpose but instead of storing the 8-bit input A it stores the 8-input B

Truth Tables for Latch 1 and 2:

Since a truth table for an 8-bit latch would have alot of possible solutions lets make the inputs my student number. My student number is 5011720244. Converting that to the inputs A and B we get

$$A = (20)_{16} = (00100000)_2$$
$$B = (24)_{16} = (00100100)_2$$

Clk (Clock)	**R (Reset)**	**A**	**Q (output)**
0	0	00100000	00000000 (if previous input did not exist, if it did it would be that previous input)
0	1	00100000	00000000
1	0	00100000	00100000
1	1	00100000	00000000

Table 1.1: The truth table for Latch 1 with input as A

For Latch 2 it is very similar with just input B instead of A.

Clk (Clock)	**R (Reset)**	**B**	**Q (output)**
0	0	00100100	00000000 (if previous input did not exist, if it did it would be that previous input)
0	1	00100100	00000000
1	0	00100100	00100100
1	1	00100100	00000000

Table 1.2: Truth table for Latch 2 with Input B

Block Diagram for Latch 1:

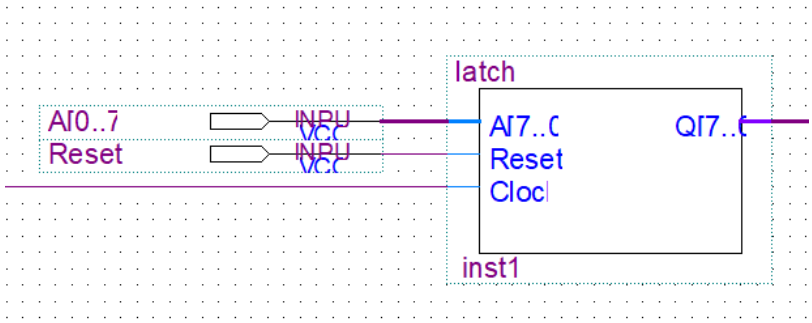

Figure 1.1: Block diagram for Latch 1

Block Diagram for Latch 2:

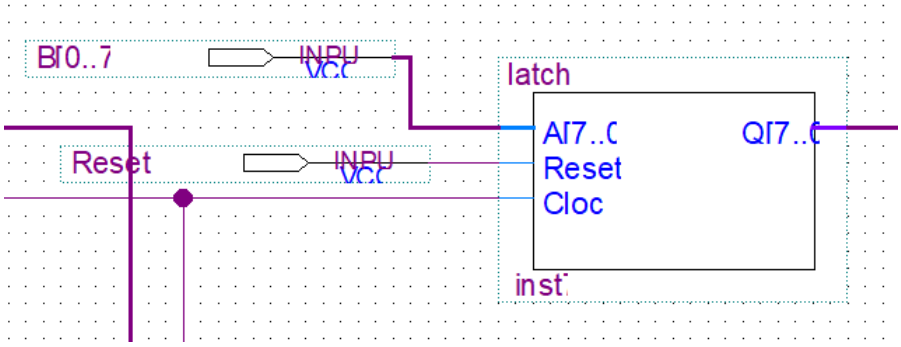

Figure 1.2: Block Diagram for Latch 2

Waveform:

Since the VHDL code is the same for each Latch and as seen above the block diagram is also the same. So the waveform for each Latch is also the same. Because of this only one waveform will be shown below and that waveform will have the input A as its input. However for the input B the waveform is the exact same but with the values in A changed to B

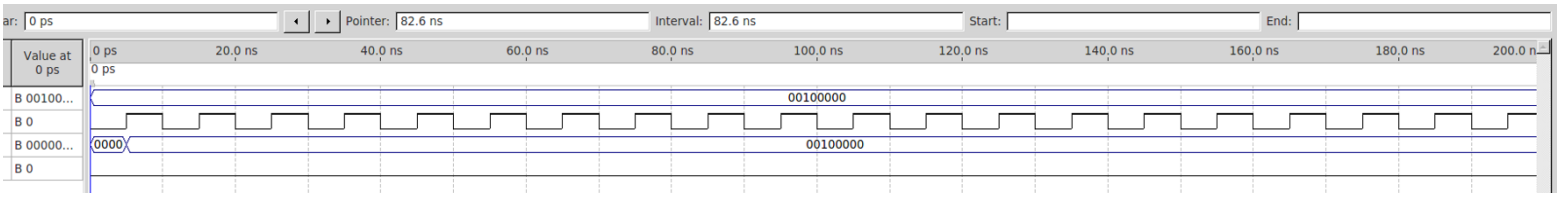

Figure 1.3: Waveform for Latch with input as A

State Machine (FSM):

` The FSM component decides the pattern of the controller sequence. Our FSM machine was designed to act as a Mealy machine and that was used to cycle through 9 different states using the clock signal. The current state of the FSM will then be passed along to the Decoder. This happens when the FSM takes the clock signal as the input, and produces the 4-bit output current_state and passes that to the decoder. The VHDL code can be found in the **Appendix Part A section**.

Truth Table for FSM:

Current State		Input	Next State		Outputs
A	B	I	Anext	Bnext	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	X	X	X
1	1	1	X	X	X

Figure 2.1: Simple Truth table for a FSM Machine

Block Diagram for FSM:

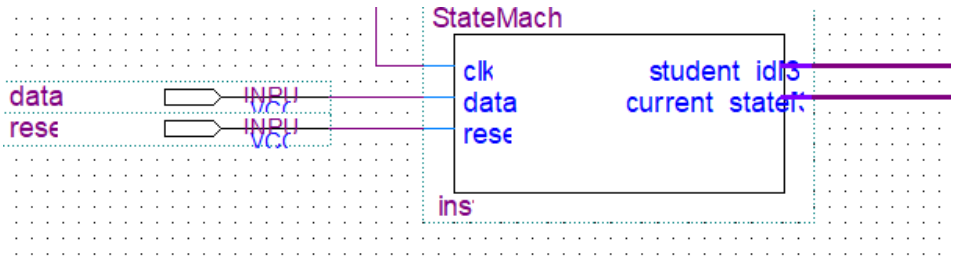

Figure 2.2: Block diagram of the FSM Machine

Waveform:

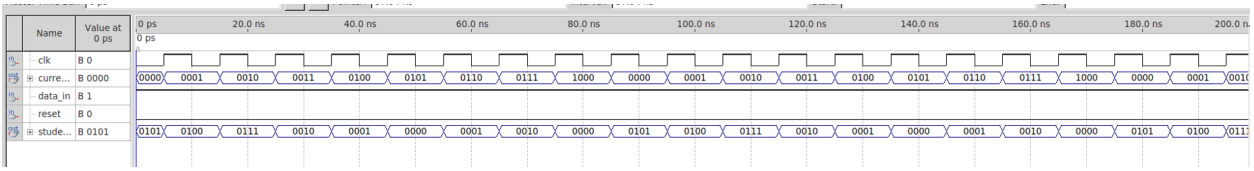

Figure 2.3: Waveform for the FSM Machine

4 to 16 Decoder:

A decoder is a digital circuit that converts a binary code into a set of output signals. A 4 to 16 decoder is a type of decoder that has 4 input lines and 16 output lines. In this lab it receives the signal current_state from FSM and decodes it to the operation-selector microcode. The decoder unit passes the signal OP to the ALU core, which will then translate to the operations selector for the ALU core and perform an operation based on a set of Tables. The VHDL code for this can be found in **Appendix Part A.**

E	D	C	B	A	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}
0	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 3.1: Truth table for a 4:16 Decoder

Since the signal that the decoder passes to the ALU is an OP signal the outputs in the truth table can be written as OP instead of y_0, y_1, y_2 etc… this gets you a truth table like the following

En	S_3	S_2	S_1	S_0	$Output$
1	0	0	0	0	OP0
1	0	0	0	1	OP1
1	0	0	1	0	OP2
1	0	0	1	1	OP3
1	0	1	0	0	OP4
1	0	1	0	1	OP5
1	0	1	1	0	OP6
1	0	1	1	1	OP7
1	1	0	0	0	OP8
1	1	0	0	1	OP9
1	1	0	1	0	OP10
1	1	0	1	1	OP11
1	1	1	0	0	OP12
1	1	1	0	1	OP13
1	1	1	1	0	OP14
1	1	1	1	1	OP15

Table 3.2: Truth table for a 4:16 Decoder when output is OP

Block Diagram for 4:16 Decoder:

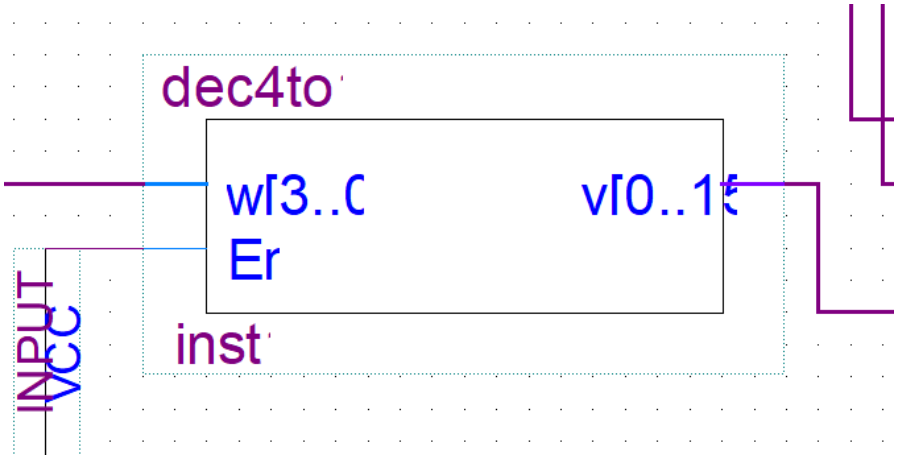

Figure 3.3: Block Diagram for 4:16 Decoder

Waveform:

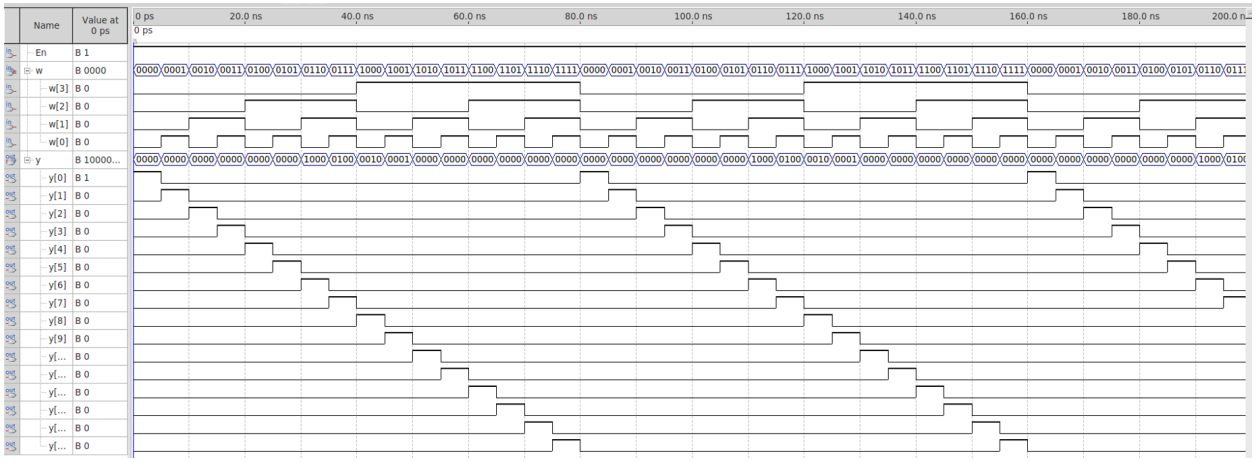

Figure 3.4: Waveform for a 4:16 Decoder

ALU 1- Problem Set 1

In ALU 1 all arithmetic and logical operations are to be implemented and applied. The ALU takes in two 8 bit inputs in this case A and B and a 16-bit input from the control unit. The microcode input from the controller unit is the operation selector signal which decides the operation that is to be applied on the inputs A and B from **Table 4.0**. Only 9 distinct operations are to be implemented which are in the following table. To find the code for ALU 1 pls visit **Appendix Part B**

The operations listed in Table 4.0 are the ones that are supposed to be implemented by the ALU and the result, which should be an 8-bit output, is to be displayed on two 7-segment displays or LEDS.

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 4.0: 9 different operations ALU 1 can do

ALU 1 has 4 inputs, which are the clock signal, input A, input B, and the microcode from the control unit. The inputs A and B are consistent and the operations implemented on these two vary based on whatever function it selects. Out of the 9 operations, the control unit choses one and then that operation is done on the inputs of A and B. These are then outputted to 3 different 7 segment displays, one that is in charge of the negative sign and two for the output display. On the first rising edge (when clock is on) the inputs A and B are sent to ALU 1, and when the control unit is turned on each number of the student id is displayed on one seven segment display, with the corresponding operation that A and B went through.

Block Schematic:

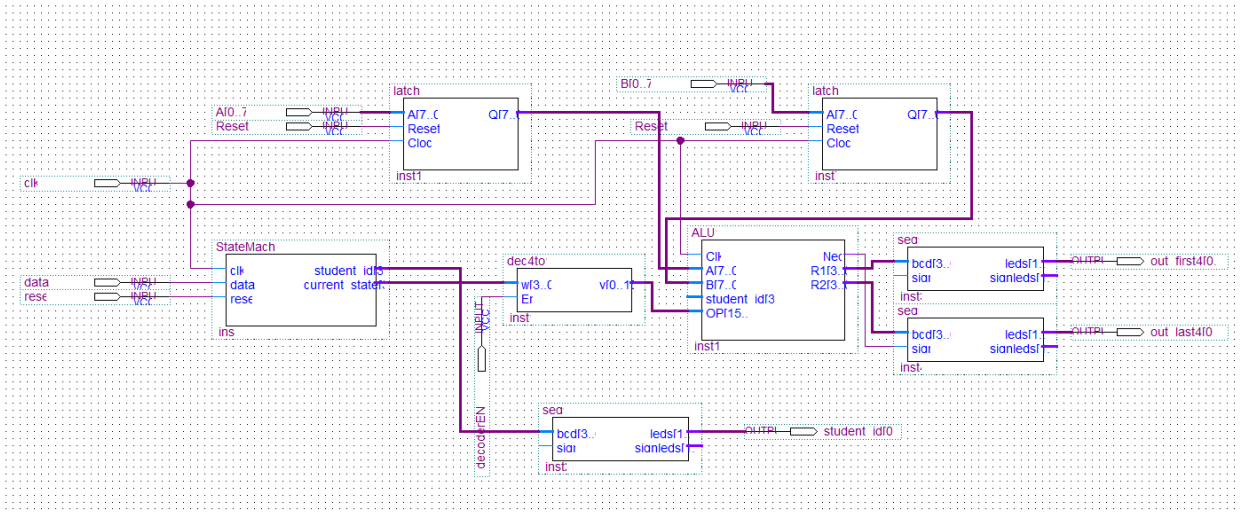

Figure 4.1: Block schematic for ALU 1

Table of Microcode

In order to check if the ALU actually did those operations on the waveform, a table of each Microcode was made so that we could compare the answers from the table with the waveform itself. Below in Table 4.2 are the manual calculations of input A = 00100000 and B= 00100100

Function	Microcode	Operation	R2 (binary)	R2 (Hex)	R1 (Binary)	R1 (Hex)
1- OP[0]	0000 0000 0000 0001	sum(**A,B**)	0100	4	0100	4
2- OP[1]	0000 0000 0000 0010	diff(**A,B**)	1111	F	1100	C
3- OP[2]	0000 0000 0000 0100	NOT(**A**)	1101	D	1111	F
4- OP[3]	0000 0000 0000 1000	NAND(**A,B**)	1101	D	1111	F
5-OP[4]	0000 0000 0001 0000	NOR(**A,B**)	1101	D	1011	B

6-OP[5]	0000 0000 0010 0000	AND(**A,B**)	0010	2	0000	0
7-OP[6]	0000 0000 0100 0000	XOR(**A,B**)	0000	0	0100	4
8-OP[7]	0000 0000 1000 0000	OR(**A,B**)	0010	2	0100	4
9-OP[8]	0000 0001 0000 0000	XNOR(**A,B**)	1111	F	1011	11

Table 4.2 Manual operations for ALU

Waveform:

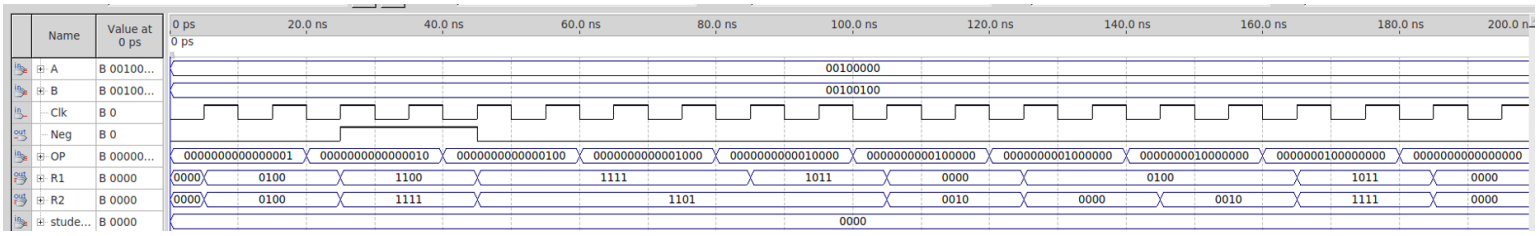

Figure 4.3: Time Diagram for ALU with my Student #

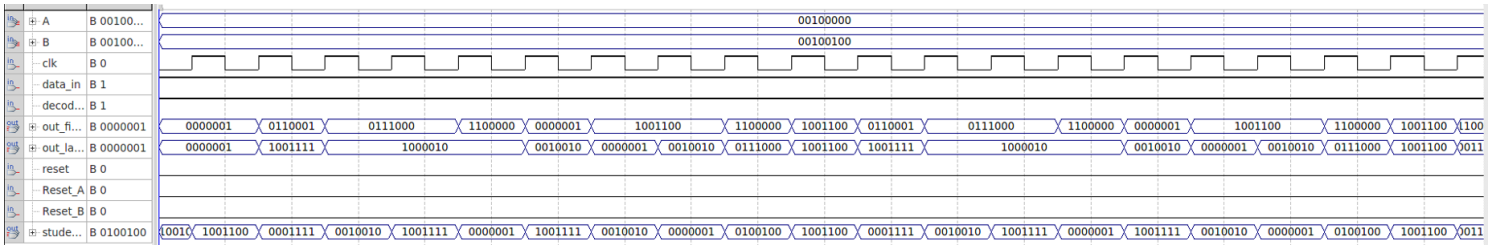

Figure 4.4: Timing Diagram for entire Problem set 1

ALU 2- Problem Set 2

In this problem set the ALU 1 from Problem set 1 was modified so that the ALU would choose a different operation from a different set of Operations. The ALU was modified such that it could do the following functions. To find the modified VHDL code pls go to **Appendix B.**

Function #	Operation / Function
1	Increment **A** by 2
2	Shift **B** to right by two bits, input bit = 0 (SHR)
3	Shift **A** to right by four bits, input bit = 1 (SHR)
4	Find the smaller value of **A** and **B** and produce the results (Min(**A**,**B**))
5	Rotate **A** to right by two bits (ROR)
6	Invert the bit-significance order of **B**
7	Produce the result of XORing **A** and **B**
8	Produce the summation of **A** and **B**, then decrease it by 4
9	Produce all high bits on the output

Table 5.0: Table of different set of operations ALU 2 must be able to do

This ALU has 4 inputs that affect the ALU 2. The 4 inputs are the clock signal, input A and B and the microcode from the control unit. This is the same as ALU 1, the inputs and the outputs are the same, the only difference is the functions that A and B go through because the functions in this ALU core are different than the ones from ALU 1.

Block Schematic:

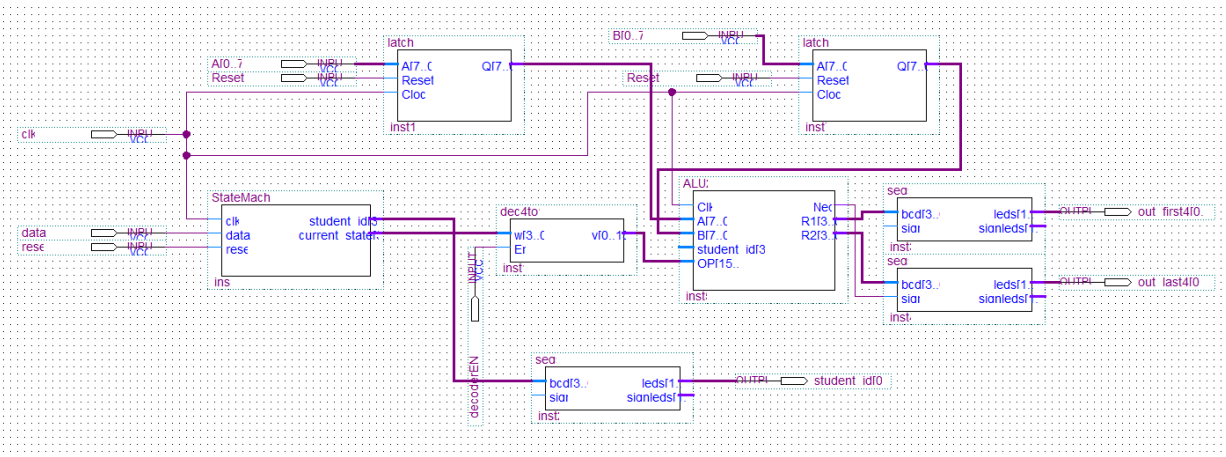

Figure 5.1: Block schematic for ALU 2

Table of Microcode

Below in the waveform the correct Output is shown, to ensure that is correct each operation was manually done on the inputs A and B and shown in the table below. Recall that A = 00100000 and B= 00100100

Function	Microcode	Operation	R2 (binary)	R2 (Hex)	R1 (Binary)	R1 (Hex)
1- OP[0]	0000 0000 0000 0001	Increment A by 2	0100	4	0000	0
2- OP[1]	0000 0000 0000 0010	Shift B to right by two bits, input bit = 0	0000	0	1001	9
3- OP[2]	0000 0000 0000 0100	Shift A to right by four bits, input bit = 1	1110	E	0000	0
4- OP[3]	0000 0000 0000 1000	Find the smaller value of A and B and produce the results (Min(A, B))	0010	2	0000	0
5-OP[4]	0000 0000 0001 0000	Rotate A to right by two bits (ROR)	0000	0	1000	8
6-OP[5]	0000 0000 0010 0000	Invert the bit-significance order of B	0010	2	0100	4
7-OP[6]	0000 0000 0100 0000	Produce the result of XORing A and B	0000	0	0100	4
8-OP[7]	0000 0000 1000 0000	Produce the summation of	0011	3	1100	C

		A and B, then decrease it by 4				
9-OP[8]	0000 0001 0000 0000	Produce all high bits on the output	1111	F	1111	F

Table 5.2: Manual Operations for ALU 2

Waveform:

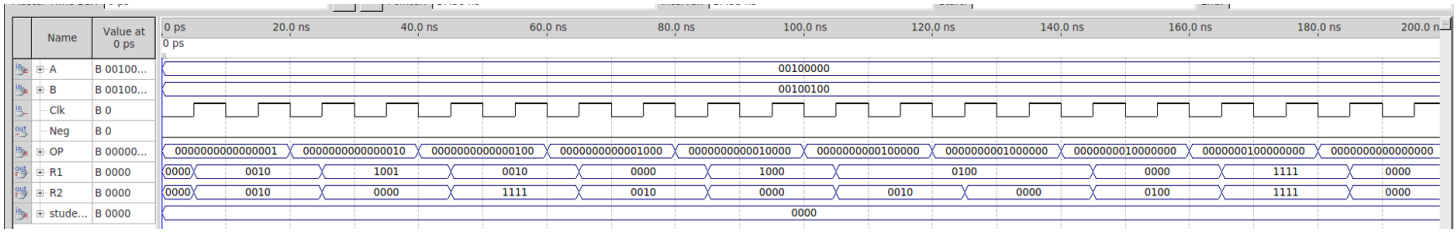

Figure 5.3: Time Diagram for ALU 2 with my Student #

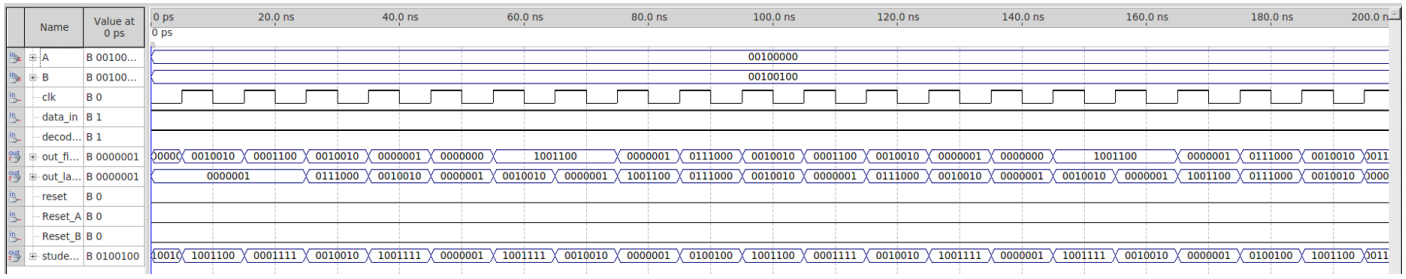

Figure 5.4: Timing Diagram for entire Problem set 2

ALU 3- Problem Set 3

The ALU 3 for problem set 3 is almost identical to the first problem set with very little modification. The biggest difference is that an additional Seven Segment Display is added with the outputs “y” or “n” depending on which number of the student number is displayed. For each microcode instruction, the ALU would output the previous outputs in Problem set 1 along with “y” if the FSM output is odd and “n” otherwise. Please refer to **Appendix Part B** to see the modified ALU code.

Similar to the previous two problem sets, the ALU 3 has the same inputs and they function the same way. The only modification for this problem set is that the student number outputted from the control unit is inputted in the ALU. The ALU checks if the student number is an odd or even number, and then sends an output of “1” if odd or “0” if even. It sends this output to a special 7-segment display which displays either a “y” or a “n.

Block Schematic:

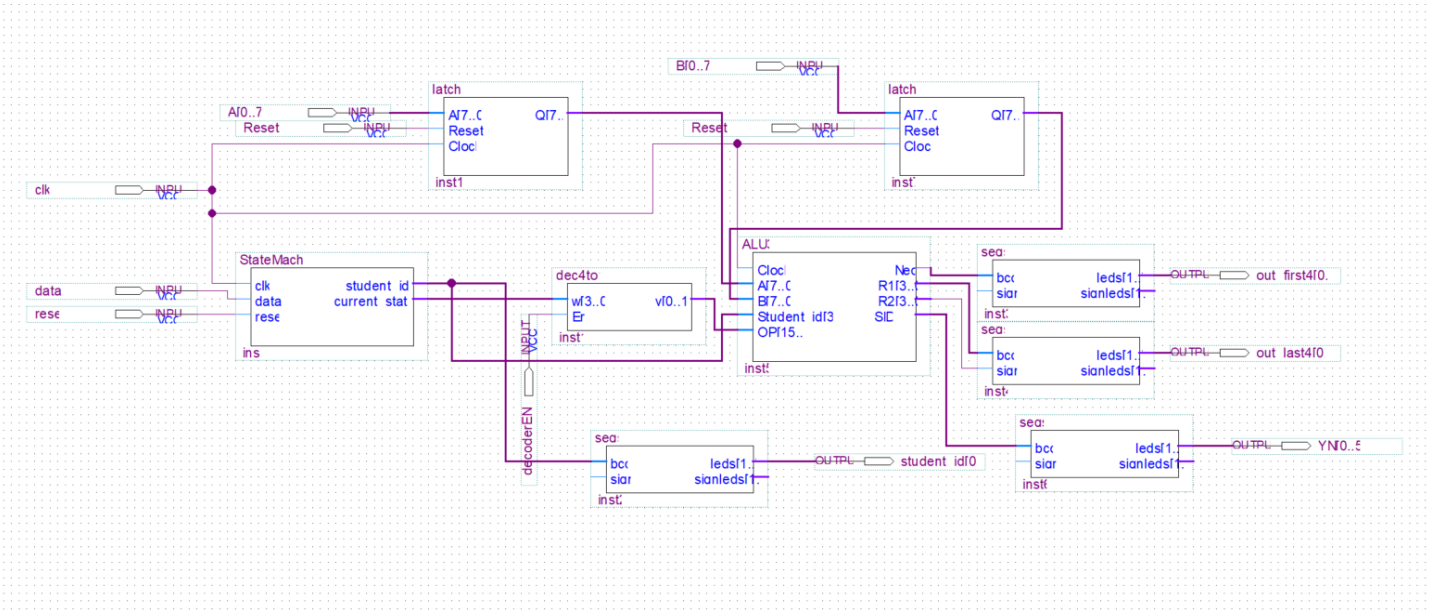

Figure 6.1: Block schematic for Problem Set 3

Table of Microcode

Student #	Microcode	Operation	R2 (binary)	R2 (Hex)	R1 (Binary)	R1 (Hex)	Odd/Even(ALU Output)
5	0000 0000 0000 0001	sum(**A,B**)	0100	4	0100	4	(1)y
0	0000 0000 0000 0010	diff(**A,B**)	1111	(-)F	1100	C	(0)n
1	0000 0000 0000 0100	NOT(**A**)	1101	D	1111	F	(1)y
1	0000 0000 0000 1000	NAND(**A,B**)	1101	D	1111	F	(1)y
7	0000 0000 0001 0000	NOR(**A,B**)	1101	D	1011	B	(1)y
2	0000 0000 0010 0000	AND(**A,B**)	0010	2	0000	0	(0)n
0	0000 0000 0100 0000	XOR(**A,B**)	0000	0	0100	4	(0)n
2	0000 0000 1000 0000	OR(**A,B**)	0010	2	0100	4	(0)n
4	0000 0001 0000 0000	XNOR(**A,B**)	1111	F	1011	11	(0)n

Figure 6.2: Table of the operations the SSEG will display based on weather the number is Odd

Waveform:

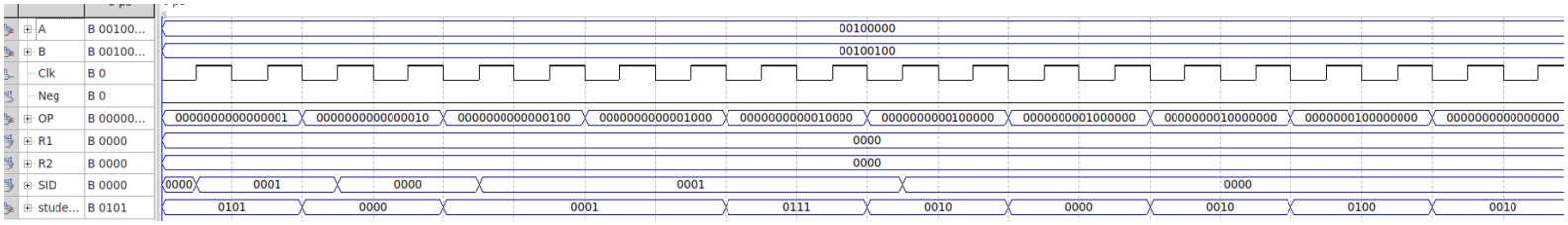

Figure 6.3 Waveform for ALU 3 with my Student #

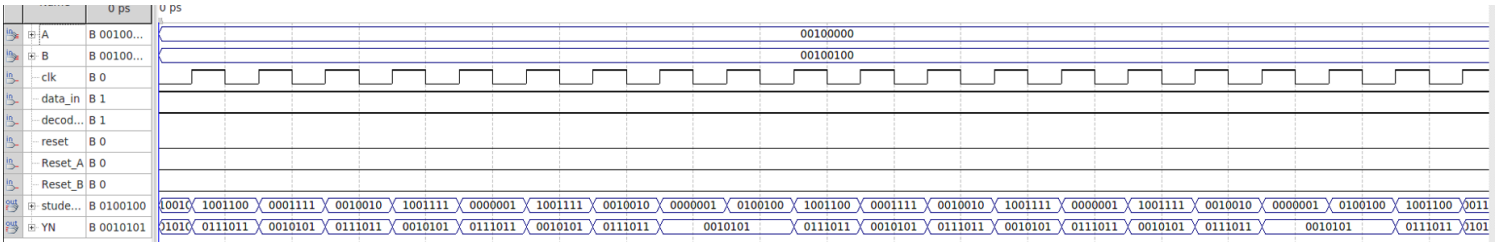

Figure 6.4: Waveform for entire problem 3 with my Student #

Conclusion

In conclusion this lab focused on the design and implementation of a Simple processor in a VHDL environment and using it on the FPGA board. The project contained 4 main components that ensured the processor works smoothly and properly. The Storage Unit with two Latches with 8-bit registries to temporarily store inputs A and B. The Control Unit, with an FSM and a 4:16 Decoder which determined the microcode delivered to the ALU core, which did a specific operation for the 8-bit input. Three sets of 9 operations were implemented through the three ALU Cores. The final component, the seven segment display unit, showed the output that was calculated from the ALU on inputs A and B, the student number, the sign, and for problem 3 whether a number was odd. To ensure this setup was done properly it was evaluated based on waveforms, circuit diagrams and a demonstration which was done on an FPGA board. In essence this was a very interesting lab and it provided a lot of necessary knowledge that will be needed in the future. One big issue in this lab was that despite the correct waveforms the FPGA board would not work; this could be due to the fact that they are super old and used by many students.

Toronto Metropolitan University

Appendix

VHDL Code for each Component:

Latch 1 and Latch 2:

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

ENTITY latch1 IS
        PORT ( A:IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                                Resetn, Clock : IN STD_LOGIC;
                                Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));

END latch1;
ARCHITECTURE Behavior OF latch1 IS
        BEGIN
                PROCESS ( Resetn, Clock )
                BEGIN
                        IF Resetn = '1' THEN
                                Q <= "00000000";
                        ELSIF Clock'EVENT AND Clock='1' THEN
                                Q <= A;
                        END IF;
                END PROCESS;
END Behavior;
```

State Machine (FSM):

```
library ieee;
use ieee.std_logic_1164.all;
entity machine is
        port
        (
                clk         : in   std_logic;
                data_in     : in   std_logic;
                reset       : in   std_logic;
                student_id  : out   std_logic_vector(3 downto 0);
```

```
                current_state : out std_logic_vector(3 DOWNTO 0)
        );
end entity;
architecture fsm of machine is
        -- Build an enumerated type with 9 states for the state machine (9 states for parsing 9 digits of student id)
   type state_type is (s0, s1, s2, s3, s4, s5, s6, s7, s8);
   -- Register to hold the current state
   signal yfsm : state_type;
begin
        process (clk, reset)
        begin
                if reset = '1' then
          yfsm <= s0;
      elsif (clk'EVENT AND clk = '1') then
                        -- Determine the next state synchronously, based on
                        -- the current state and the input
                        case yfsm is
                        when s0=>

                        if data_in='1' then
                                yfsm <= s2;
                        else yfsm <= s0;
                        end if;

                        when s2=>

                        if data_in='1' then
                                yfsm <= s4;
                        else yfsm <= s2;
                        end if;

                        when s4=>

                        if data_in='1' then
                                yfsm <= s6;
                        else yfsm <= s4;
                        end if;

                        when s6=>

                        if data_in='1' then
```

```
                    yfsm <= s8;
                else yfsm <= s6;
                end if;

                when s8=>

                if data_in='1' then
                    yfsm <= s7;
                else yfsm <= s8;
                end if;

                when s7=>

                if data_in='1' then
                    yfsm <= s5;
                else yfsm <= s7;
                end if;

                when s5=>

                if data_in='1' then
                    yfsm <= s3;
                else yfsm <= s5;
                end if;

                when s3=>

                if data_in='1' then
                    yfsm <= s1;
                else yfsm <= s3;
                end if;

                when s1=>

                if data_in='1' then
                    yfsm <= s0;
                else yfsm <= s1;
                end if;

        end case;
end if;
```

```
end process;
-- Implement the Moore or Mealy logic here
process (yfsm, data_in) -- data_in if reqd only
begin
        case yfsm is
                when s0=> --s1 points to s0
                student_id <= "0101"; --5
                current_state <= "0000";  -- current state s0

    when s2=> --s0 points to s2
                student_id <= "0000"; --0
                current_state <= "0010"; -- current state s2

    when s4=> --s2 points to s4
                student_id <= "0001"; --1
                current_state <= "0100"; -- current state s4

    when s6=> --s4 points to s6
                student_id <= "0001"; --1
                current_state <= "0110"; -- current state s6

    when s8=> --s6 points to s8
                student_id <= "1000"; --8
                current_state <= "1000"; -- current state s8

    when s7=> --s8 points to s7
                student_id <= "0010"; --2
                current_state <= "0111"; -- current state s7

    when s5=> --s7 points to s5
                student_id <= "0001"; --1
                current_state <= "0101"; -- current state s5

          when s3=> --s5 points to s3
                student_id <= "1000"; --8
                current_state <= "0011"; -- current state s3

    when s1=> --s3 points to s1
                student_id <= "0001"; --1
                current_state <= "0001"; -- current state s1
```

```
    end case;
        end process;
end architecture;
```

4 to 16 Decoder:

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

ENTITY dec4to16 IS
PORT ( w : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
En : IN STD_LOGIC ;
y : OUT STD_LOGIC_VECTOR(0 TO 15) ) ;
END dec4to16 ;

ARCHITECTURE Structure OF dec4to16 IS
COMPONENT dec2to4
PORT ( w : IN STD_LOGIC_VECTOR(1 DOWNTO 0) ;
En : IN STD_LOGIC ;
y : OUT STD_LOGIC_VECTOR(0 TO 3) ) ;
END COMPONENT ;
SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;
BEGIN
G1: FOR i IN 0 TO 3 GENERATE
Dec_ri: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(i), y(4*i TO 4*i+3) );
G2: IF i=3 GENERATE
Dec_left: dec2to4 PORT MAP ( w(i DOWNTO i-1), En, m ) ;
END GENERATE ;
END GENERATE ;
END Structure ;
```

VHDL Code for each ALU:

ALU 1 Problem set 1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ALU1 is
port(Clk:in std_logic;
```

```
A,B : in unsigned(7 downto 0);
student_id : in unsigned(3 downto 0);
OP : in unsigned(15 downto 0);
Neg : out std_logic;
R1: out unsigned(3 downto 0);
R2: out unsigned(3 downto 0));
end ALU1;
architecture calculation of ALU1 is
signal Reg1,Reg2,Result : unsigned(7 downto 0) :=(others=> '0');
begin
Reg1 <= A;
Reg2 <= B;
process(Clk,OP)
        begin
        if(rising_edge(Clk)) THEN
                case OP is

                        WHEN "0000000000000001" =>
                        Result <= Reg1 + Reg2;

                        WHEN "0000000000000010" =>
                        if (Reg2>Reg1) then
                        Result<= (Reg1+(NOT Reg2+1));
                        Neg<='1';
                        else
                        Result<= (Reg1-Reg2);
                        Neg<='0';
                        end if;

                        WHEN "0000000000000100" =>
                        Result <= NOT Reg1;
                        Neg<='0';

                        WHEN "0000000000001000" =>
                        Result <= (NOT (Reg1 AND Reg2));
                        Neg<='0';

                        WHEN "0000000000010000" =>
                        Result <= (NOT (Reg1 OR Reg2));
                        Neg<='0';
```

```
                    WHEN "0000000000100000" =>
                    Result <= (Reg1 AND Reg2);
                    Neg<='0';

                    WHEN "0000000001000000" =>
                    Result <= Reg1 XOR Reg2;
                    Neg<='0';

                    WHEN "0000000010000000" =>
                    Result <= (Reg1 OR Reg2);
                    Neg<='0';

                    WHEN "0000000100000000" =>
                    Result <= (Reg1 XNOR Reg2);
                    Neg<='0';

                    WHEN OTHERS =>
                    Result<= "--------";


                    end case;
                end if;
        end process;

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

ALU 2 Problem set 2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ALU2 is
port(Clk:in std_logic;
A,B : in unsigned(7 downto 0);
student_id : in unsigned(3 downto 0);
OP : in unsigned(15 downto 0);
Neg : out std_logic;
R1: out unsigned(3 downto 0);
```

```
R2: out unsigned(3 downto 0));
end ALU2;
architecture calculation of ALU2 is
signal Reg1,Reg2,Result : unsigned(7 downto 0) :=(others=> '0');
begin
Reg1 <= A;
Reg2 <= B;
process(Clk,OP)
        begin
        if(rising_edge(Clk)) THEN
                case OP is

                        WHEN "0000000000000001" =>
                        Result <= Reg1 + 2;
                        Neg<='0';

                        WHEN "0000000000000010" =>
                        Result <= shift_right(Reg2, 2);
                        Neg<='0';

                        WHEN "0000000000000100" =>
                        Result <= shift_right(Reg1, 4);
                        Result(7)<='1';
                        Result(6)<='1';
                        Result(5)<='1';
                        Result(4)<='1';

                        WHEN "0000000000001000" =>
                        if(Reg1>Reg2) THEN
                        Result <= Reg2;
                        else
                        Result <= Reg1;
                        end if;
                        Neg<='0';

                        WHEN "0000000000010000" =>
                        Result<=rotate_right(Reg1,2);
                        Neg<='0';

                        WHEN "0000000000100000" =>
                        Result(0)<=Reg2(7);
```

```
                    Result(1)<=Reg2(6);
                    Result(2)<=Reg2(5);
                    Result(3)<=Reg2(4);
                    Result(4)<=Reg2(3);
                    Result(5)<=Reg2(2);
                    Result(6)<=Reg2(1);
                    Result(7)<=Reg2(0);
                    Neg<='0';

                    WHEN "0000000001000000" =>
                    Result <= Reg1 XOR Reg2;
                    Neg<='0';

                    WHEN "0000000010000000" =>
                    Result <= Reg1 + Reg2 -4;
                    Neg<='0';

                    WHEN "0000000100000000" =>
                    Result(7)<='1';
                    Result(6)<='1';
                    Result(5)<='1';
                    Result(4)<='1';
                    Result(3)<='1';
                    Result(2)<='1';
                    Result(1)<='1';
                    Result(0)<='1';
                    Neg<='0';

                    WHEN OTHERS =>
                    Result<= "--------";


                    end case;
                end if;
        end process;

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

ALU 3 Problem set 3

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY ALU3 IS
   PORT (
      Clock : IN STD_LOGIC;      -- input clock signal
      A, B : IN UNSIGNED(7 DOWNTO 0);  -- 8-bit inputs from Latches A and B
      Student_id : IN UNSIGNED(3 DOWNTO 0);  -- 4-bit student id from FSM
      OP : IN UNSIGNED(15 DOWNTO 0);  -- 16-bit selector for Operation from Decoder
      Neg : OUT STD_LOGIC;  -- is the result negative? Set -ve bit output
      R1 : OUT UNSIGNED(3 DOWNTO 0);  -- lower 4-bits of 8-bit Result Output
      R2 : OUT UNSIGNED(3 DOWNTO 0);  -- higher 4-bits of 8-bit result output
      SID : OUT UNSIGNED(3 DOWNTO 0)  -- 4-bit SID output
   );
END ALU3;

ARCHITECTURE Behavior OF ALU3 IS
   SIGNAL Result : UNSIGNED(7 DOWNTO 0) := (OTHERS => '0');
BEGIN

   PROCESS (Clock, OP, Student_id)
   BEGIN
      IF (Clock = '1') THEN
         CASE OP IS
            WHEN "0000000000000001" =>  -- Even
               Neg <= '0';
               Result <= "00010001";
               SID <= Student_id;  -- Assign Student_id to SID
            WHEN "0000000000000010" =>  -- Odd
               Neg <= '0';
               IF Student_id = "1001" THEN
                  -- Display 'y' for student_id "501172024"
                  R1 <= "1111";
                  R2 <= "1001";
               ELSE
                  -- Display 'n' for other student_id values
                  R1 <= "1101";
                  R2 <= "1101";
               END IF;
```

```
            SID <= Student_id;  -- Assign Student_id to SID
          -- Add more cases as needed
          WHEN OTHERS =>  -- Don't care, do nothing
            NULL;
        END CASE;
      END IF;
    END PROCESS;
END Behavior;
```