Ryerson University
Department of Electrical & Computer Engineering
Faculty of Engineering & Architectural Science

Course Title:	DIGITAL SYSTEMS
Course Number:	COE328
Semester/Year (e.g.F2016)	FALL2020

Instructor:	Nauman Baig

Assignment/Lab Number:	6
Assignment/Lab Title:	Design of a Simple General- Purpose Processor

Submission Date:	04/12/2020
Due Date:	04/12/2020

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Vinayak	Unnati	500983169	16	

Reset Form

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

TABLE OF CONTENTS:

INTRODUCTION & GOALS:

The main purpose of this lab experiment is to design and built a GPU (General Processing Unit). This lab required knowledge of components from previous 5 labs. The ALU would be the core and is responsible for doing logical and arithmetic calculations, based on the 2 input values. The output is dependent on the main control unit, which consists of the 4-16 decoder and a FSM (Function State Machine).

This lab was split into 3 main parts, with the first part being required to write four VHDL codes: latch1, decoder, FSM, and the final ALU code. We were required to implement a final circuit combining the VHDL code blocks for all the codes. A final waveform for the circuit was created. The second part of this lab required us to modify the ALU code from part 1 based on the assigned function. From there, we had to implement the same circuit from part 1, with the difference being that the ALU block from part 1 would be replaced with the ALU block diagram from part 2. A final waveform was also created for this circuit. For part 3, we had to modify the ALU code once again based on the assigned function. A circuit for part 3 was created by taking the previous circuit and replacing the ALU block with the ALU block modified for part 3. A final waveform was also created for this circuit.

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
9	Invert all bits of **B**

Assigned functions for part II: These functions had to be implemented in the ALU code for part II of the lab experiment.

c) For each microcode instruction, display 'y' if the FSM output (**student_id**) had an odd parity and 'n' otherwise

Assigned function for part III: This function had to be implemented in the ALU code for part III of the lab experiment.

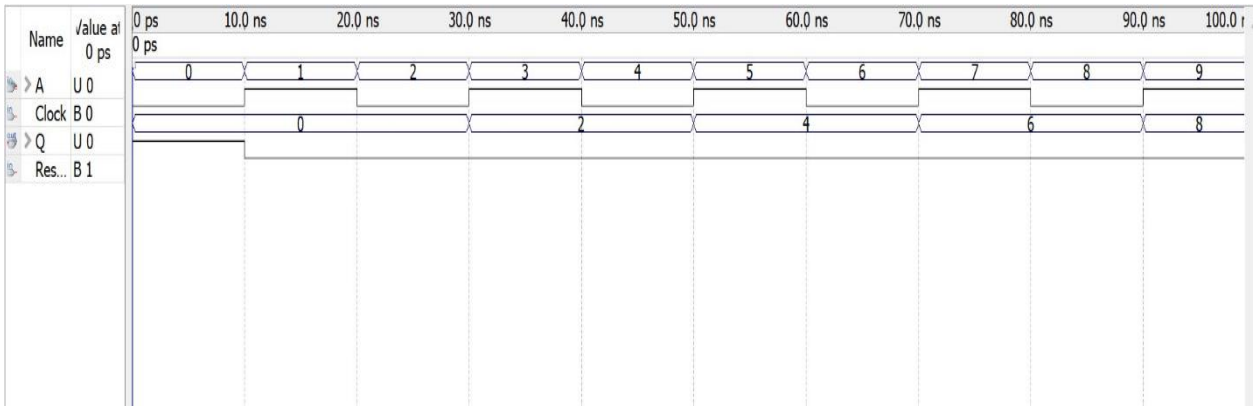
LAB PROCEDURE:

COMPONENTS:

Latch:

This code helps create the circuit element, which acts like a memory storage device. The VHDL code for latch along with its corresponding waveform can be seen below.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY latch1 IS
5
6     PORT (A : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
7           Resetn, Clock: IN STD_LOGIC;
8           Q: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
9
10 END latch1;
11
12 ARCHITECTURE Behavior OF latch1 IS
13 BEGIN
14    PROCESS (Resetn, Clock)
15    BEGIN
16       IF Resetn = '1' THEN
17          Q <= "00000000";
18       ELSIF Clock'EVENT AND Clock = '1' THEN
19          Q <= A;
20       END IF;
21    END PROCESS;
22 END Behavior;
```



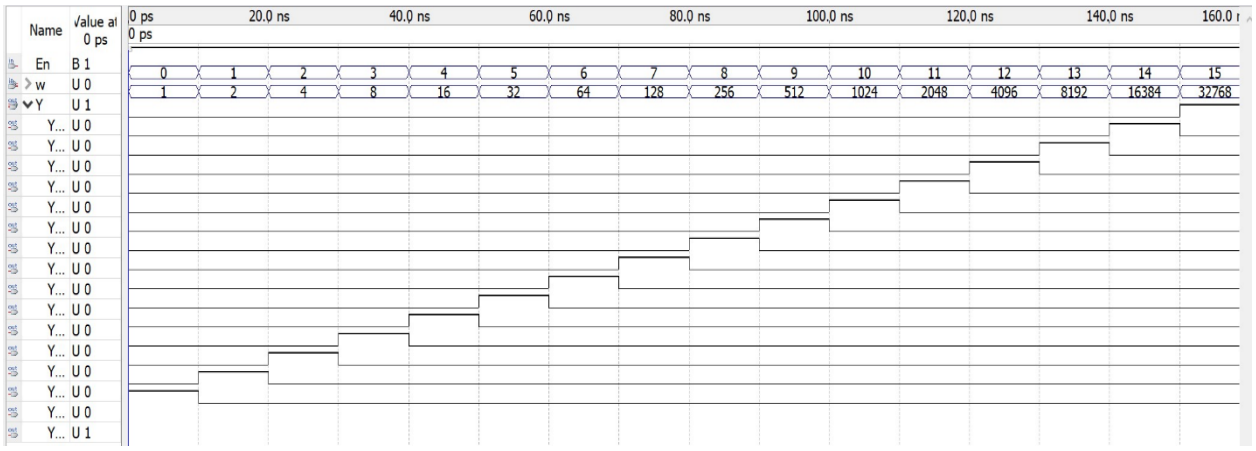
Reset	Clock	A	Q
1	↑	0000 0001	0000 0000
1	↑	0000 0010	0000 0001
1	↑	0000 0011	0000 0010
1	↑	0000 0100	0000 0011
1	↑	0000 0101	0000 0100
1	↑	0000 0110	0000 0101
1	↑	0000 0111	0000 0110
0	↑	0000 1000	0000 0000
0	↑	0000 1001	0000 0000

Table 1 : TruthTable for Latch

Decoder:

This code is used so it can take the 4-bit input of the states of the FSM and convert them to an output of 16-bit. There output would correspond to the microcode for the ALU code. We also had to write a VHDL code for lab 4; that code was taken and changed so that it converts from 4-bit to 16-bit. The code for the decoder, the waveform, along with the truth table is given below:

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

ENTITY decod_4to16 IS
PORT (w: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      En: IN STD_LOGIC;
      Y: OUT STD_LOGIC_VECTOR(0 TO 15));
END decod_4to16;

ARCHITECTURE Behavior OF decod_4to16 IS
SIGNAL Enw: STD_LOGIC_VECTOR(4 DOWNTO 0);
BEGIN
    Enw <= En & w;
    WITH Enw Select

    y <= "0000000000000001" WHEN "10000",
         "0000000000000010" WHEN "10001",
         "0000000000000100" WHEN "10010",
         "0000000000001000" WHEN "10011",
         "0000000000010000" WHEN "10100",
         "0000000000100000" WHEN "10101",
         "0000000001000000" WHEN "10110",
         "0000000010000000" WHEN "10111",
         "0000000100000000" WHEN "11000",
         "0000001000000000" WHEN "11001",
         "0000010000000000" WHEN "11010",
         "0000100000000000" WHEN "11011",
         "0001000000000000" WHEN "11100",
         "0010000000000000" WHEN "11101",
         "0100000000000000" WHEN "11110",
         "1000000000000000" WHEN "11111",
         "0000000000000000" WHEN OTHERS;
END Behavior;
```



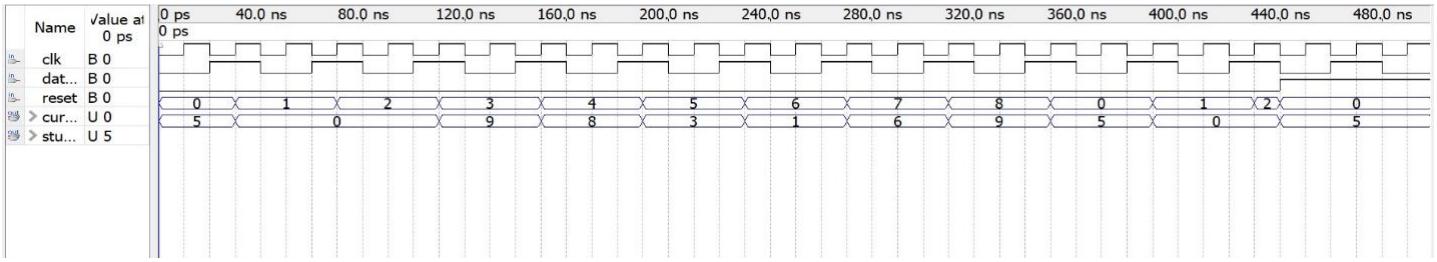
W	X	Y	Z	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
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

FSM:

The FSM (finite state machine) has 3 inputs; clk, data_in, and reset. These input variables control the transition from state to state. The output of the FSM are the student_id and current, which are 4-bits each.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY fsm IS
5
6  PORT (clk, data_in, reset: IN STD_LOGIC;
7        student_id: OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
8        current: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
9  END ENTITY;
10
11 ARCHITECTURE Behavior OF fsm IS
12 type state_type is (s0, s1, s2, s3, s4, s5, s6, s7, s8);
15 BEGIN
16
17    p1: process(clk, reset)
18    BEGIN
19
20    if reset = '1' then
21    yfsm <= s0;
22    else if (clk'EVENT AND clk = '1') THEN
23    CASE yfsm is
24
25    when s0 => if data_in = '1' then yfsm <=s1; else yfsm <= s0; end if;
26    when s1 => if data_in = '1' then yfsm <=s2; else yfsm <= s1; end if;
27    when s2 => if data_in = '1' then yfsm <=s3; else yfsm <= s2; end if;
28    when s3 => if data_in = '1' then yfsm <=s4; else yfsm <= s3; end if;
29    when s4 => if data_in = '1' then yfsm <=s5; else yfsm <= s4; end if;
30    when s5 => if data_in = '1' then yfsm <=s6; else yfsm <= s5; end if;
31    when s6 => if data_in = '1' then yfsm <=s7; else yfsm <= s6; end if;
32    when s7 => if data_in = '1' then yfsm <=s8; else yfsm <= s7; end if;
33    when s8 => if data_in = '1' then yfsm <=s0; else yfsm <= s8; end if;
34
35    end case;
36    end if;
37    end if;
38    end process p1;
```

```
40  process (yfsm, data_in)
41
42  BEGIN
43
44  case yfsm is
45
46  when s0=>
47  current <= "0000";
48  student_id <= ("0101");
49
50  when s1=>
51  current <= "0001";
52  student_id <= ("0000");
53
54  when s2=>
55  current <= "0010";
56  student_id <= ("0000");
57
58  when s3=>
59  current <= "0011";
60  student_id <= ("1001");
61
62  when s4=>
63  current <= "0100";
64  student_id <= ("1000");
66  when s5=>
67  current <= "0101";
68  student_id <= ("0011");
69
70  when s6=>
71  current <= "0110";
72  student_id <= ("0001");
73
74  when s7=>
75  current <= "0111";
76  student_id <= ("0110");
77
78  when s8=>
79  current <= "1000";
80  student_id <= ("1001");
81
82  end case;
83  end process;
84  END Behavior;
```

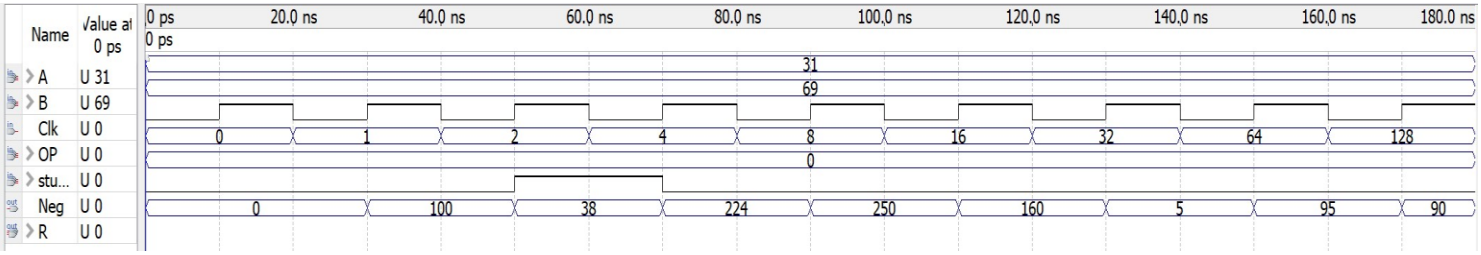


ALU:

The main purpose of the ALU (Arithmetic & Logic Unit) is to perform many calculations of many functions on two input variables; in this case, these input variables were A and B. The Boolean function that would be calculated from functions 1-9 would depend on the microcode. The table below shows the Boolean operation/function that would correspond to each of the microcode values.

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	$\overline{A \oplus B}$

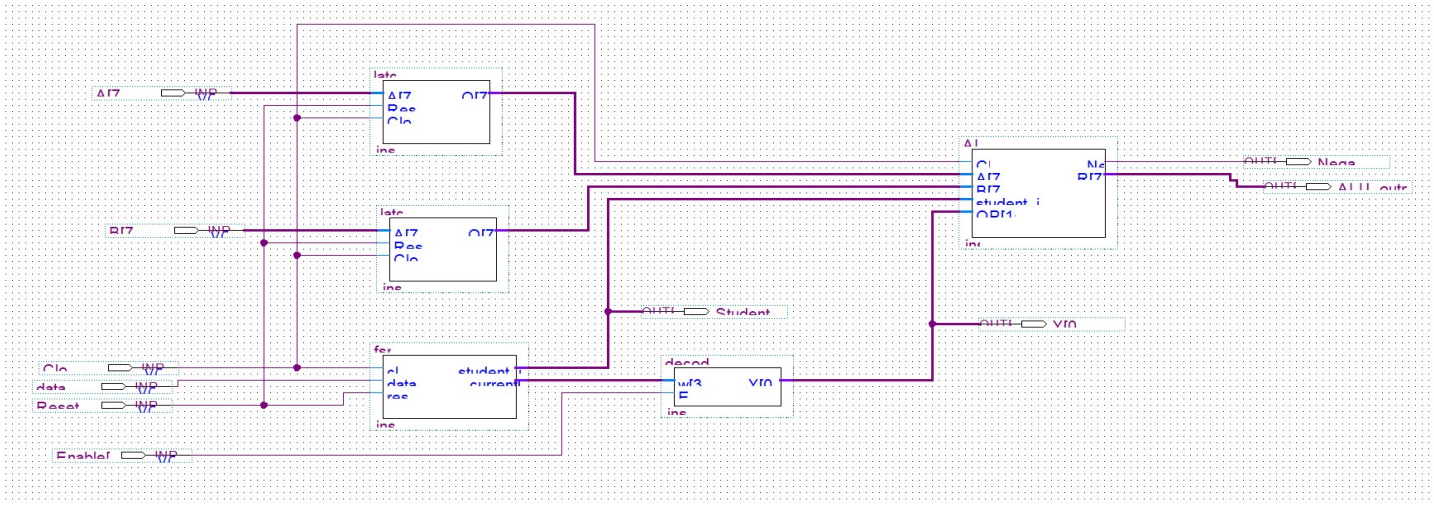
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

ENTITY ALU 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;
      R :OUT UNSIGNED(7 DOWNTO 0));

END ALU;

ARCHITECTURE calculation of ALU IS
signal Reg1, Reg2, Result: unsigned(7 DOWNTO 0):=(others => '0');
signal Reg4: unsigned(0 TO 7);
BEGIN

Reg1 <= A;
Reg2 <= B;

PROCESS (Clk, OP)

BEGIN

if(rising_edge(Clk)) THEN
case OP is
    WHEN "0000000000000001" --Function 1

        => Result <= (Reg1 + Reg2);

    WHEN "0000000000000010" --Function 2

        => if Reg1 < Reg2 THEN
        neg <= '1';
        Result <= Reg2-Reg1;
        else
        Result <= Reg1-Reg2;
        neg <= '0';
        end if;

    WHEN "0000000000000100" --Function 3

        => Result <= NOT(Reg1);
        neg <= '0';

    WHEN "0000000000001000"

        => Result <= (Reg1 NAND Reg2); --Function 4

    WHEN "0000000000010000"

        => Result <= (Reg1 NOR Reg2); --Function 5

    WHEN "0000000000100000"

        => Result <= (Reg1 AND Reg2); -- Function 6

    WHEN "0000000001000000"

        => Result <= (Reg1 OR Reg2); --Function 7

    WHEN "0000000010000000"

        => Result <= (Reg1 XOR Reg2); --Function 8

    WHEN "0000000100000000"

        => Result <= (Reg1 XNOR Reg2); --Function 9

    WHEN OTHERS
        => NULL;

END CASE;
END IF;
END PROCESS;

R <= Result(7 DOWNTO 0);

END calculation;
```

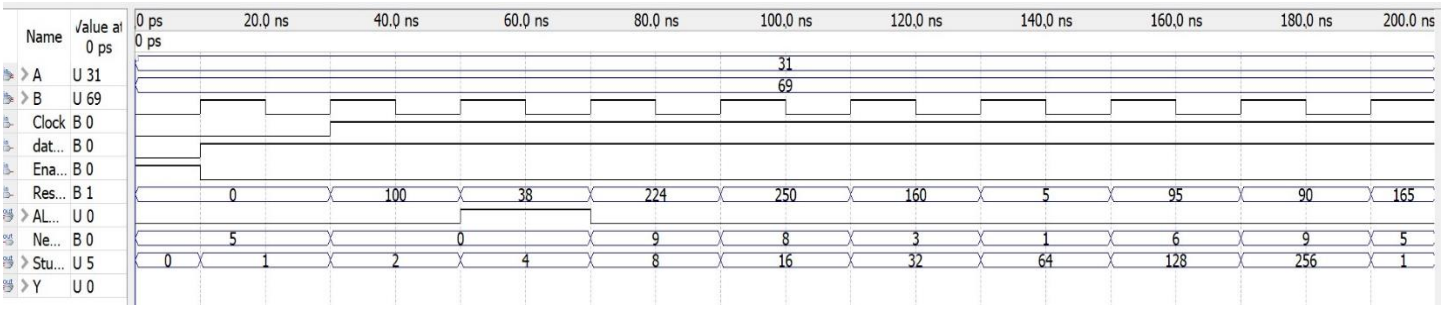


The last two digits of my student number are 3169, so A was set equal to 31 and B was set equal to 69.

PART I:

For this part, we were required to implement a final circuit, which combined all the VHDL codes and their corresponding block diagrams, which were discussed in the components section of the lab report.





We can confirm that this waveform is correct as the output of the operations make sense. For example, A and B are 31 and 69 respectively. We see that when we add then together the output is 100 and when we subtract them, we get 38, with a 1 rise, which corresponds to -38.

PART II:

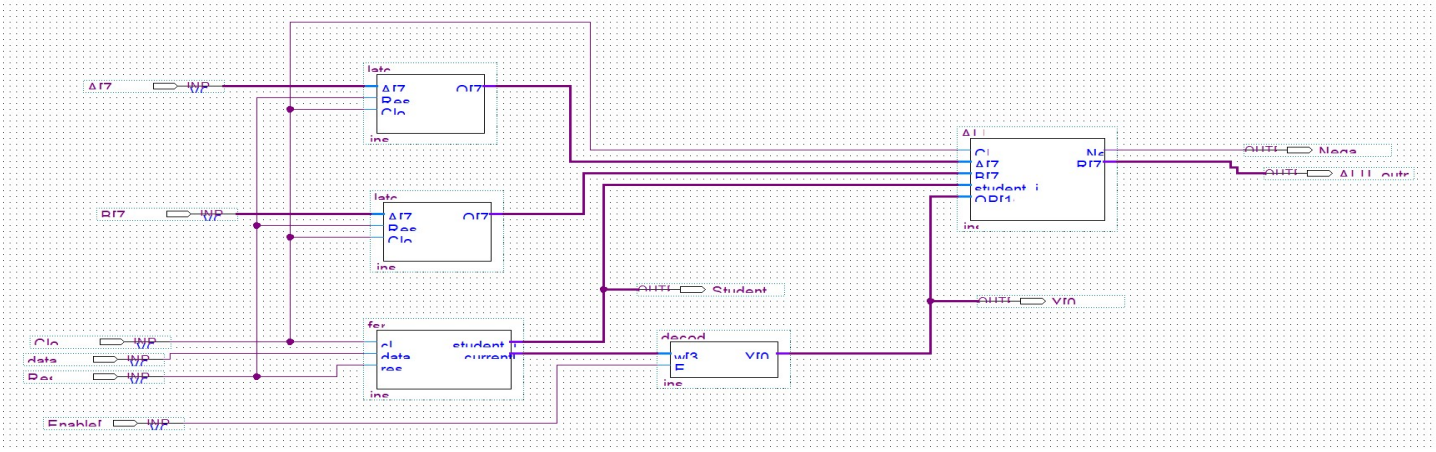
This part required us to implement the same final circuit from part I, with the difference being that the ALU code from part I had to be modified to perform the assigned functions. I was assigned table C, which is shown below.

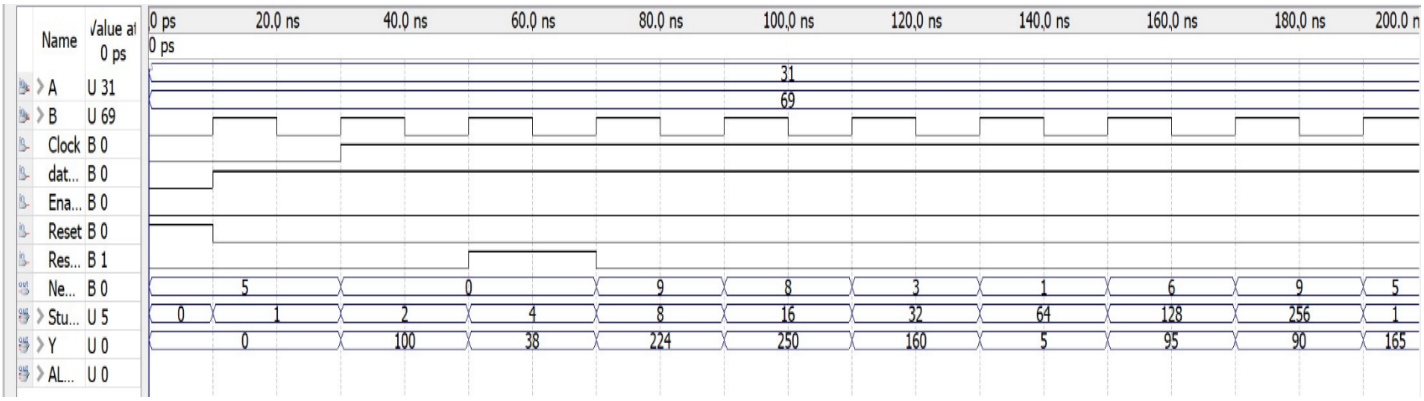
Function #	Operation / Function
1	Produce the difference between **A** and **B**
2	Produce the 2's complement of **B**
3	Swap the lower 4 bits of **A** with lower 4 bits of **B**
4	Produce null on the output
5	Decrement **B** by 5
6	Invert the bit-significance order of **A**
7	Shift **B** to left by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

Functions 1 asked to perform the subtraction operation between the variables A and B. Function 2 asked to produce the 2's complement of B. This was done by taking the not of B and adding 1 to the result. Function 3 asked to switch the 4 lower bits of A with the 4 lower bits of B. Function 4 was simple and asked to produce a null as the output. Function 5 was also simple and required to decrease the value of B by 5. Function 6 required for the bit significance order of A to be inverted. I was required to shift B to the left by 3 bits for function 7. Function was simple and required for A to be increased by 3. Function 9 asked for all bits of B to be inverted.

The modified ALU code for all these operations is shown below. Each operation is labeled. The final circuit is also shown, along with its corresponding waveform. The circuit is similar to the one from part I. The difference is that the ALU block from part I is replaced with the ALU block created from the modified ALU code for part II.

```vhdl
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;
      R :OUT UNSIGNED(7 DOWNTO 0));

END ALU2;

ARCHITECTURE calculation of ALU2 IS
signal Reg1, Reg2, Result: unsigned(7 DOWNTO 0):=(others => '0');
signal Reg4: unsigned(0 TO 7);
BEGIN

Reg1 <= A;
Reg2 <= B;
PROCESS (Clk, OP)

BEGIN

if(rising_edge(Clk)) THEN
case OP is
   WHEN "0000000000000001" --Function 1

      => if Reg1 < Reg2 THEN
      neg <= '1';
      Result <= Reg2-Reg1;
      else
      Result <= Reg1-Reg2;
      neg <= '0';
      end if;

   WHEN "0000000000000010" --Function 2

      => Result <= ((not(Reg2)) + 1);
      neg <= '0';


   WHEN "0000000000000100" --Function 3

      => Result <= (Reg1(7 downto 4) & Reg2(3 downto 0));
   WHEN "0000000000001000" => NULL;  --Function 4


   WHEN "0000000000010000" --Function 5

      => Result <= Reg2 - 5;

   WHEN "0000000000100000" --Function 6

      => Result <= (Reg2(0)&Reg2(1)&Reg2(2)&Reg2(3)&Reg2(4)&Reg2(5)&Reg2(6)&Reg2(7

   WHEN "0000000001000000" --Function 7

      => Result <= ( Reg2(4 downto 0) & "111");

   WHEN "0000000010000000" --Function 8

      => Result <= Reg1 + "00000011";

   WHEN "0000000100000000" --Function 9

      => Result <= (NOT Reg2);

   WHEN OTHERS => NULL;

END CASE;
END IF;
END PROCESS;
```



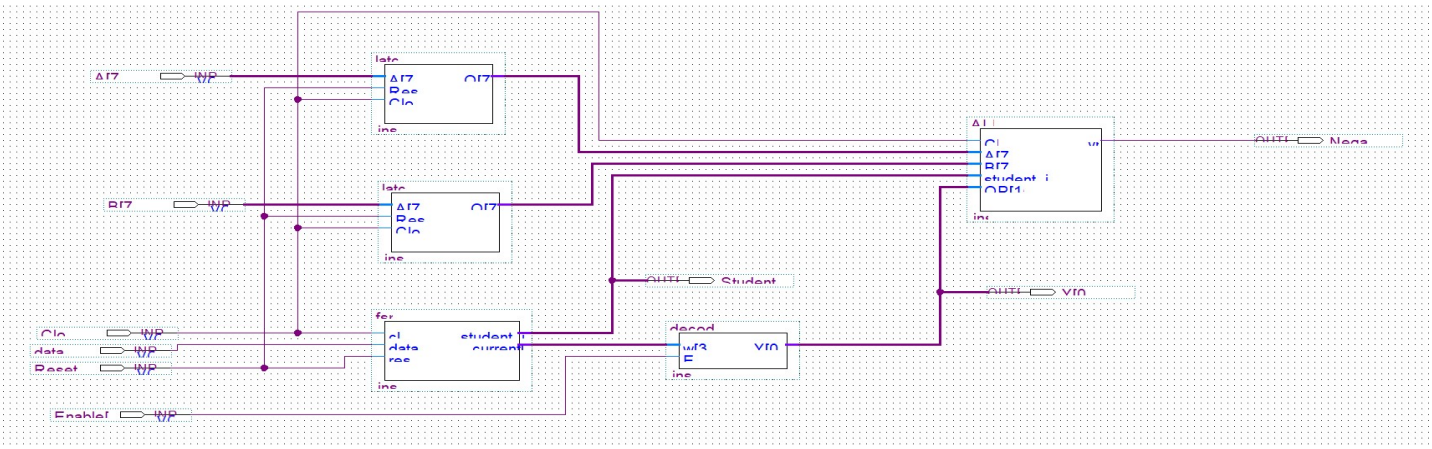


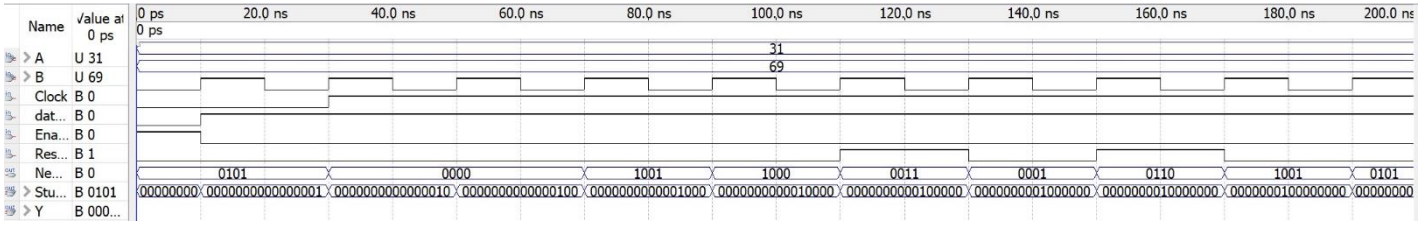
To confirm if the final waveform was correct, A and B were set equal to 31 and 69 respectively. Each of the operations were performed on them to see what the answer would be. Online calculators were used to confirm the calculations done by hand.

PART III:

This part is like part I and II in that you are once again required to modify the ALU code to perform certain functions with the input variables being A and B. For part I, the variables were A and B, which the functions were the same for everyone. In part II, each person was assigned a different set of functions. The ALU had to be modified such that for each microcode, I had to check for the parity (even/odd) of my student number.

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

ENTITY ALU3 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);
      yn :OUT STD_LOGIC);

END ALU3;

ARCHITECTURE calculation of ALU3 IS
signal student: std_logic_vector(3 DOWNTO 0);
signal parity : std_logic;

BEGIN

student <= std_logic_vector(student_id);
parity  <= student(0)XOR student(1) XOR student (2) XOR student(3);

PROCESS (Clk, OP)

BEGIN

if(rising_edge(Clk)) THEN
case OP is

   WHEN "0000000000000001" =>

      if parity ='1'
      then yn <= '1';
      else yn <= '0';
      end if;

   WHEN "0000000000000010" =>

      if parity ='1'
      then yn <= '1';
      else yn <= '0';
      end if;

   WHEN "0000000000000100" =>

      if parity ='1'
      then yn <= '1';
      else yn <= '0';
      end if;

   WHEN "0000000000001000" =>

      if parity ='1'
      then yn <= '1';
      else yn <= '0';
      end if;

   WHEN "0000000000010000" =>

      if parity ='1'
      then yn <= '1';
      else yn <= '0';
      end if;

   WHEN "0000000000100000" =>

      if parity ='1'
      then yn <= '1';
      else yn <= '0';
      end if;

   WHEN "0000000001000000" =>

      if parity ='1'
      then yn <= '1';
      else yn <= '0';
      end if;

   WHEN "0000000010000000" =>

      if parity ='1'
      then yn <= '1';
      else yn <= '0';
      end if;

   WHEN "0000000100000000" =>

      if parity ='1'
      then yn <= '1';
      else yn <= '0';
      end if;

   WHEN OTHERS
      => NULL;
```





The waveform for this part has a decay, which is why the output is shifted to the right.

CONCLUSION:

This lab experiment helped me understand that various circuits and VHDL codes can be reused to create a larger circuit which combines all the other VHDL code block elements. I learned that the input and outputs can by manipulated to implement something different from what was originally used. After creating a circuit that acts like a simple GPU, I learned that each block element has its own function. The ALU block was created from the ALU code; ALU carries out different logical operations until the machine (Moore in this case) has completed its cycle. The latches block in the circuit was created from the latches code; this part of the GPU is responsible for reading and storing data/memory. This stored data/memory call be used by other components in the circuits as needed. After understanding the contribution of each part, it can be said that a GPU that works well must contain a storage unit, control unit, and the main ALU that performs the operations.