Course Number	COE 328
Course Title	Digital Systems
Semester and Year	Fall 2023
Instructor	Dr. Sedaghat
Teacher's Assistant	Leandra Budau

Lab/Tutorial Number	6
Lab/Tutorial Title	Design of a SImple General-Purpose Processor

Section Number	02
Submission Date	December 4, Monday, 11:58 PM
Due Date	December 4, Monday, 11:59 PM

Student Name	Student Number	Student Signature
Hasib Bhuiyan	501169402	

Table Of Contents

Introduction:

In lab 6, the concept of the Arithmetic and Logic Unit (ALU) is introduced, as well as its implementation and the main function. It is crucial to understand that the ALU comprises four main components; that is the Procuring Input Data, Storage Unit(Register), Control Unit, and the ALU Core. The storage unit consists of the latches which behave as a register, storing all the obtained and previously procured input data from the last four digits of the Student's ID number. The control unit regulates the flow of the process, it performs the fetching of instructions or signals which are made readable for later use by the ALU Core. The role of the control unit is done by the Finite State Machine(FSM) and the 4 to 16 decoder; here the FSM has the clock signal which carries out the cycle through each state or more specifically each student number. The 4 to 16 decoder converts the 4-bit input from the FSM into a 16-bit value that is readable by the ALU Core. Now the ALU core is the most critical part of the ALU itself, as it is directly involved in processing and executing the arithmetic and logical operations on the 16-bit input as required by the project's microcode and operations table. The final unit would be the seven segment display unit, which converts the 4-bit value into the BCD format which can be interpreted through the FPGA board. All these components and their units are made into distinct symbols in the final block diagram as they are crucial in the implementation of the properly functioning ALU circuit unit.

Components: Latch1, Latch2. 4:16 Decoder, and FSM:

Component Description:

The following elements and units can be found in the first few main components of the ALU. From the Procuring Input Data and Storage Unit, we have the two latches and from the Control Unit, we have the 4:16 decoder and the FSM. The latches store the obtained and procured input data that is A and B from the last four digits of the student ID (B being the last two digits, and the A being the two digits before that). The control unit's FSM controls the flow of the 4-bit output selection that is to be passed to the decoder; the FSM utilizes the clock signal cycles through the states from 0 to 8, each state representing a digit in the student's ID number through the 4 bits. This value is then converted by the decoder into a 16-bit output that can be identified by a microcode, which is further processed by the ALU Core.

Latch 1 and Latch 2:

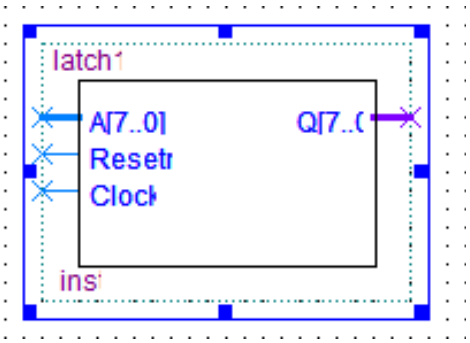


Figure 1: Block Diagram for the Latch A symbol

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

ENTITY latch1 IS
   PORT ( A:IN STD_LOGIC_VECTOR(7 DOWNTO 0); --8 bit A input
          Resetn, Clock : IN STD_LOGIC;--1 bit clock input and 1 bit reset input bit
          Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));--8 bit output

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

Figure 2: VHDL code for Latches

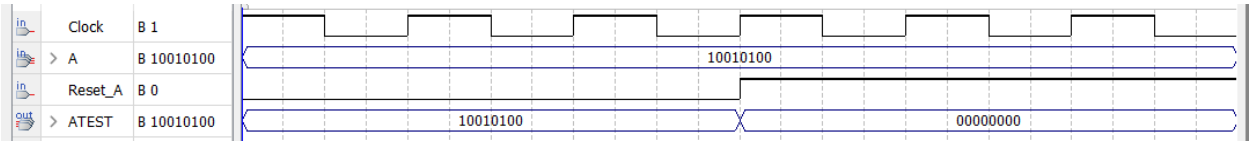


Figure 3: Waveform for Latch A

Table 1: Latch Truth Table

Input	Reset		Input(8-bit)	Output(Reset=0)	Output(Reset=1)
A	0	1	10010100	00000000	10010100
B	0	1	00000010	00000000	00000010

Finite State Machine(FSM):

Table 2: FSM Truth Table

Present State	Next State		Output	
	w = 0	w = 1	w = 0	w = 1
0000	0000	0001	0101	0000
0001	0001	0010	0000	0001
0010	0010	0011	0001	0001
0011	0011	0100	0001	0110
0100	0100	0101	0110	1001
0101	0101	0110	1001	0100
0110	0110	0111	0100	0000
0111	0111	1000	0000	0010
1000	1000	0000	0010	0101

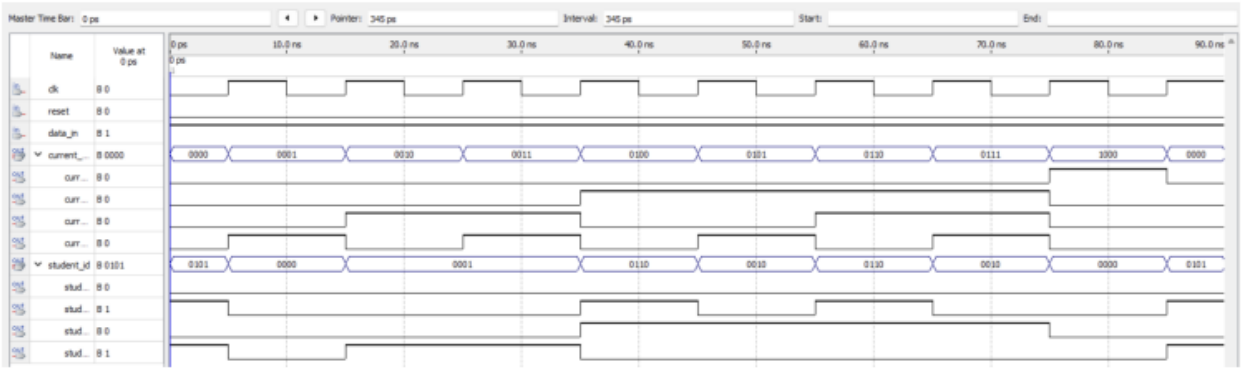


Figure 4: Waveform for FSM

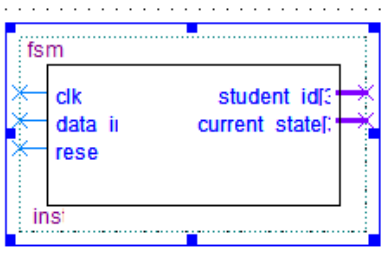


Figure 5: Block Diagram Symbol for FSM

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity fsm 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 fsm is
    type state_type is (s0, s1, s2, s3, s4, s5, s6, s7, s8);

    signal yfsm: state_type;
  begin
    process(clk, reset)
    begin
      if reset = '1' then -- resets the states to initial states
        yfsm <= s0;
      elsif(clk 'EVENT AND clk = '1') then
        case yfsm is

             when s0 =>
               if (data_in = '1') then
                 yfsm <= s1;
               else
                 yfsm <= s0;
               end if;

             when s1 =>
               if(data_in = '1')then
                 yfsm <= s2;
               else
                 yfsm <= s1;
               end if;

              when s2 =>
                 if(data_in = '1')then
                    yfsm <= s3;
                 else
                    yfsm <= s2;
                 end if;

              when s3 =>
                 if(data_in = '1')then
                    yfsm <= s4;
                 else
                    yfsm <= s3;
                 end if;

              when s4 =>
                 if(data_in = '1')then
                    yfsm <= s5;
                 else yfsm <= s4;
                 end if;

              when s5 =>
                 if(data_in = '1')then
                    yfsm <= s6;
                 else
                    yfsm <= s5;
                 end if;

              when s6 =>
                 if(data_in = '1')then
                    yfsm <= s7;
                 else
                    yfsm <= s6;
                 end if;

              when s7 =>
                 if(data_in = '1')then
                    yfsm <= s8;
                 else
                    yfsm <= s7;
                 end if;

              when s8 =>
                 if(data_in = '1')then
                    yfsm <= s0;
                 else
                    yfsm <= s8;
                 end if;
           end case;
        end if;
     end process;

     process(yfsm, data_in)
     begin --Student Number: 501169402
     case yfsm is
     when s0 =>
        current_state <= "0000";

            student_id <= "0101";


     when s1 =>
        current_state <= "0001";

            student_id <= "0000";


     when s2 =>
        current_state <= "0010";

            student_id <= "0001";

      when s3 =>
         current_state <= "0011";

              student_id <= "0001";


      when s4 =>
         current_state <= "0100";

              student_id <= "0110";

      when s5 =>
         current_state <= "0101";

              student_id <= "1001";


      when s6 =>
         current_state <="0110";

              student_id <= "0100";


      when s7 =>
         current_state <= "0111";

              student_id <= "0000";


      when s8 =>
         current_state <= "1000";

              student_id <= "0010";

    end case;
end process;
 end Architecture;
```

Figure 6: VHDL Code for FSM

4 To 16 Decoder:

Table 3: 4:16 Decoder Truth Table

Input(w[w0, w1, w2, w3])	En(Enable)	Output(16-bit)
0000	1	0000000000000001
0001	1	0000000000000010
0010	1	0000000000000100
0011	1	0000000000001000
0100	1	0000000000010000
0101	1	0000000000100000
0110	1	0000000001000000
0111	1	0000000010000000
1000	1	0000000100000000
1111	1	0000000000000000(Others)

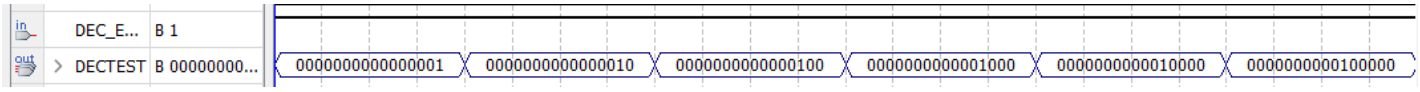


Figure 7: Waveform for 4 to 16 Decoder

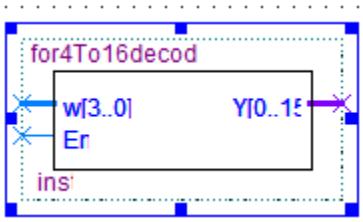


Figure 8: Block Diagram Symbol for 4:16 Decoder

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3
4   ENTITY for4To16decoder IS
5      PORT (w  : IN  STD_LOGIC_VECTOR(3 DOWNTO 0) ;
6            En : IN  STD_LOGIC ;
7            Y:  OUT STD_LOGIC_VECTOR(0 TO 15))  ;
8
9   END for4To16decoder ;
10
11  ARCHITECTURE Behavior OF for4To16decoder IS
12     SIGNAL Enw : STD_LOGIC_VECTOR(4 DOWNTO 0) ;
13  BEGIN
14     Enw <= En & w(3)&w(2)&w(1)&w(0);
15     WITH Enw SELECT --[Enw + y] => ["Enw"+"0000"]
16     Y <=
17       "0000000000000001" WHEN "10000",
18       "0000000000000010" WHEN "10001",
19       "0000000000000100" WHEN "10010",
20       "0000000000001000" WHEN "10011",
21       "0000000000010000" WHEN "10100",
22       "0000000000100000" WHEN "10101",
23       "0000000001000000" WHEN "10110",
24       "0000000010000000" WHEN "10111",
25       "0000000100000000" WHEN "11000",
26       "0000000000000001" WHEN OTHERS;
27
28
29  END Behavior ;
```

Figure 9: VHDL Code for 4 to 16 Decoder

ALU_1 for Problem Set 1 of the Lab 6 procedure:

Description:

The ALU core is the most crucial component to the general processor, it performs the main task of the processor, that is to execute operations and carry out the arithmetic and logic process to give the final results to the seven segment display, which will display it. The ALU core receives instructions and selected operation by the control unit, and using its given inputs of A and B from the latches, the ALU core will carry out an operation. There are 9 operations that can be done; addition, subtraction, NOT, NAND, NOR, AND, XOR, OR, XNOR.

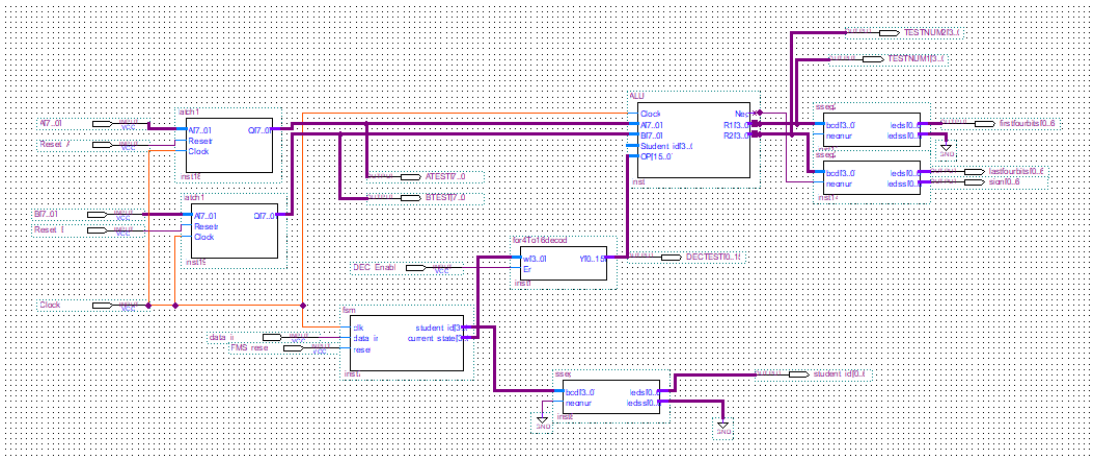


Figure 1: Block Diagram of ALU_1

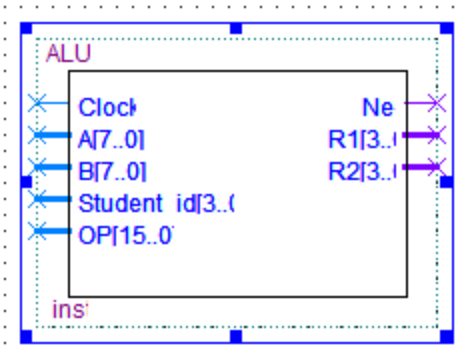


Figure 2: Block Diagram Symbol for ALU1

Inputs and Outputs:

- The A and B input values are from the last four digits of the student ID, they are the values stored in the latches, and operations are carried out on these values.
- The clock input allows the ALU, FSM, and the Latches to go through a flow, and change values accordingly to a cycle; this is especially useful for the control unit inorder to regulate selections and output operations
- The student_id input allows direct outputs and connections for the student ID number for other functions.
- OP is the 16-bit input given by the control unit to the ALU, the OP acts as instructions determining which operation to select for the ALU.
- Neg determines if the value of the output is negative(‘1’) or positive(‘0’)
- The reset inputs(Reset_A, Reset_B, FMS_Reset) allows the functions to access the reset option that will return them to the beginning; this allows control to the flow and changes made.
- R1 and R2 are the 4-bit outputs executed by the ALU core; this is the final result(8-bit) broken into two parts(fistfourbits, lastfourbits).
- Enablers(data_in, DEC_Enable) are simple inputs which allow input to be processed when it's ‘1’, and block inputs when it’s ‘0’.

Table 1: ALU_1 Microcode and Operations

Function #	Microde	Boolean Operation/Function
1	**0000000000000001**	**sum(A, B)**
2	**0000000000000010**	**diff(A, B)**
3	**0000000000000100**	**not(A)**
4	**0000000000001000**	**nand(A, B)**
5	**0000000000010000**	**nor(A, B)**
6	**0000000000100000**	**and (A, B)**
7	**0000000001000000**	**xor(A, B)**
8	**0000000010000000**	**or(A, B)**
9	**0000000100000000**	**xnor(A, B)**

Table 2: Result Value for student number 501169402

Student ID	Output(TESTNUM1 and TESNUM 2)
5	96
0	92
1	6B
1	FF
6	69
9	00
4	96
0	96
2	69

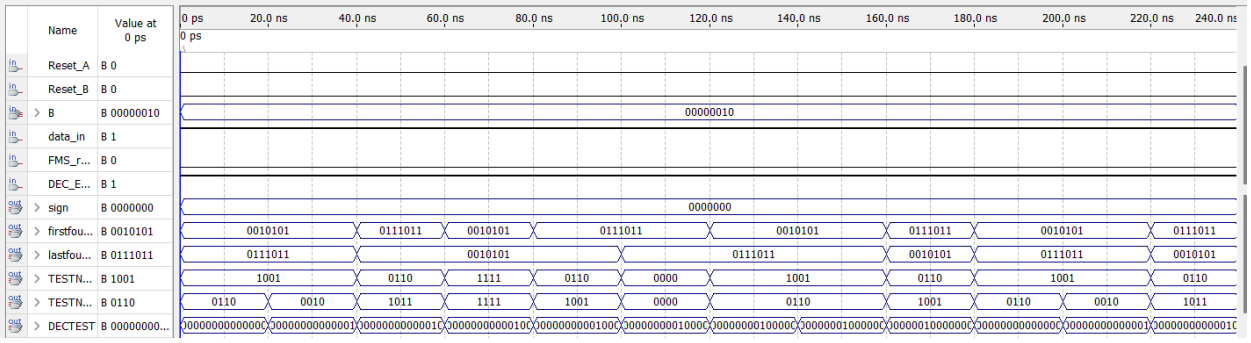


Figure 3: Waveform for ALU_1

ALU_2 for Problem Set2 of the Lab 6 procedure:

Description:

The second version of the ALU(ALU_2) is very much the same structure as the first one. They are the same, except that this time the ALU performs different functions. The instructions and selections are still given by the FSM, the values are still stored in latches A and B, and they are also displayed in the seven segment display like previously. However, this time the 9 operations are: incrementing A by two, shifting B to the right by two bits, shifting A to the right by four bits, finding the minimum between A and B, rotating A by two bits, Inverting the bit-significance order of B, performing XOR on A and B together, adding A and B then subtracting 4, and producing all high bits to the output.

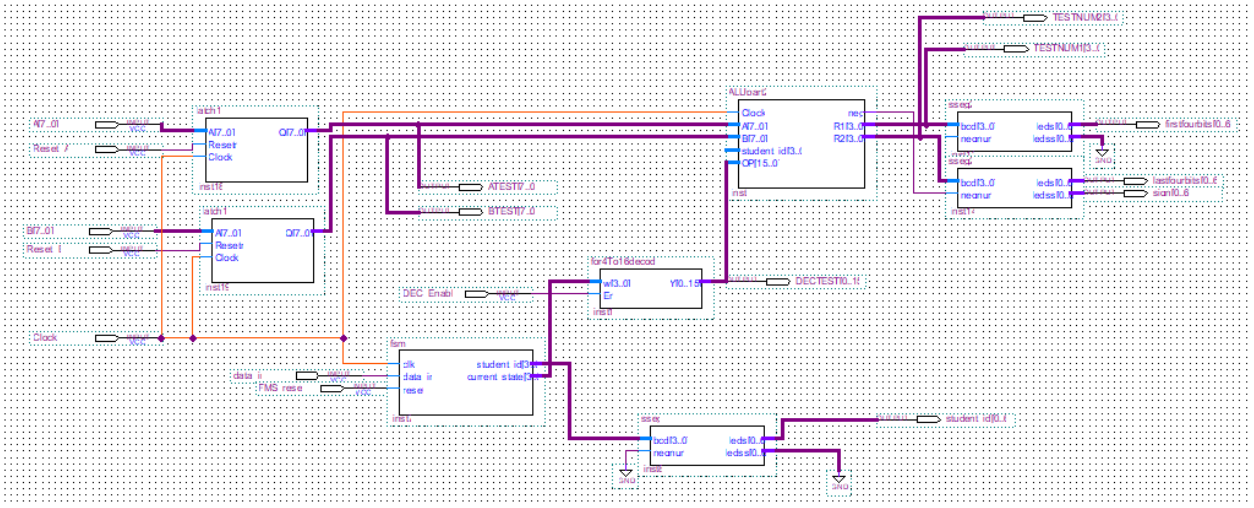


FIgure 1: Block Diagram of ALU_2

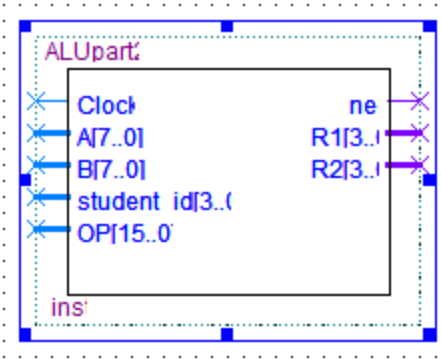


Figure 2: Block Diagram Symbol for ALU2

Inputs and Outputs:

- The A and B input values are from the last four digits of the student ID, they are the values stored in the latches, and operations are carried out on these values.
- The clock input allows the ALU, FSM, and the Latches to go through a flow, and change values accordingly to a cycle.
- The student_id input allows direct outputs and connections for the student ID number for other functions.
- OP is the 16-bit input given by the control unit to the ALU, the OP acts as instructions determining which operation to select for the ALU.
- Neg determines if the value of the output is negative(‘1’) or positive(‘0’)
- The reset inputs(Reset_A, Reset_B, FMS_Reset) allows the functions to access the reset option that will return them to the beginning; this allows control to the flow and changes made.
- R1 and R2 are the 4-bit outputs executed by the ALU core; this is the final result(8-bit) broken into two parts(fistfourbits, lastfourbits), then it is displayed.
- Enablers(data_in, DEC_Enable) are simple inputs which allow input to be processed when it's ‘1’, and block inputs when it’s ‘0’.

Table 1: ALU_2 Microcode and Operations

Function #	Microde	Boolean Operation/Function
1	**0000000000000001**	**Increment A by 2**
2	**0000000000000010**	**Shift B to the right by two bits, input bit = 0 (SHR)**
3	**0000000000000100**	**Shift A to the right by four bits, input bit = 1(SHR)**
4	**0000000000001000**	**FInd the smaller value of A and B and produce the results(Min(A, B))**
5	**0000000000010000**	**Rotate A to the right by two bits(ROR)**
6	**0000000000100000**	**Invert the bit-significance order of B**
7	**0000000001000000**	**Produce the result of XORing A and B**
8	**0000000010000000**	**Produce the summation of A and B, then decrease it by 4**
9	**0000000100000000**	**Produce all high bits on the output**

Table 2: Result Value for student number 501169402

Student ID	Output(TESTNUM1 and TESNUM 2)
5	96
0	00
1	F9
1	02
6	25
9	40
4	96
0	92
2	FF

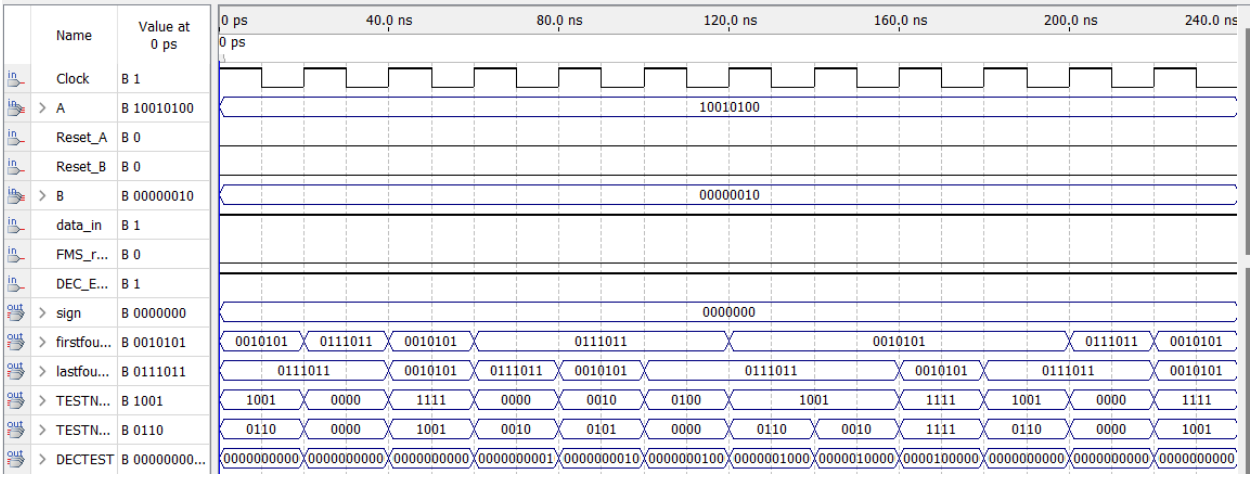


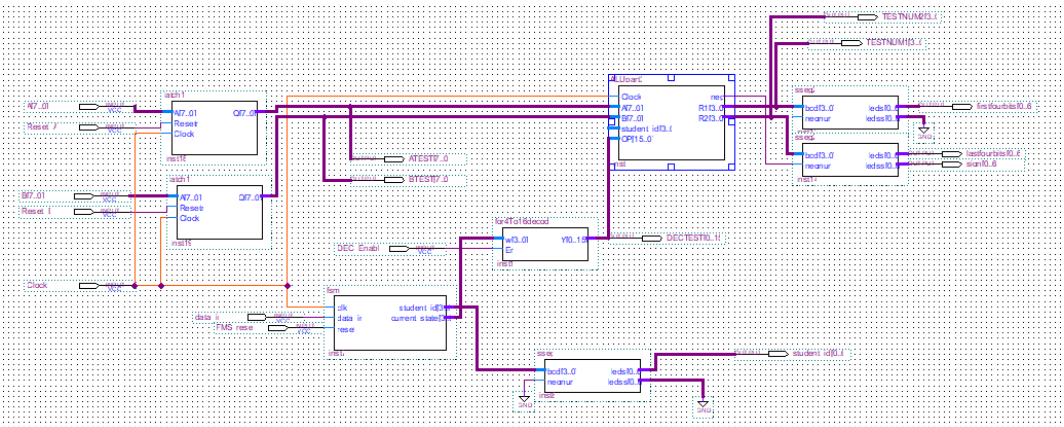
Figure 2: Block Diagram of ALU_3

ALU_3 for Problem Set3 of the Lab 6 procedure:

Description:

Similarly to the previous ALUs, this ALU is no different; it performs the Arithmetic and logical operations and produces a result. However, for this ALU_3, the operations are a bit different; with the control unit selection given, and the latches values of A and B given, the ALU takes the FSM state values(from 0 to 8 states) and determines if it's a odd or even state; when the ALU determines it, instead of giving a number representing result, it gives a result of either 'YES'(0000), or 'NO'(0001). Note that R1 and R2 are the same values, the same 4-bits are repeated for both values; for the seven segment display, they both display a yes or a no. To make this happen, the seven segment display must also change in order to show just a yes or a no.

Figure 1: Block Diagram for ALU_3



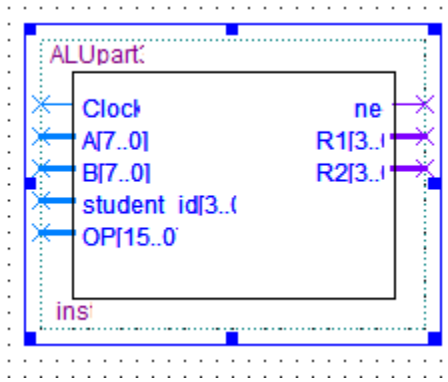


Figure 2: Block Diagram Symbol for ALU3

Inputs and Outputs:

- The A and B input values are from the last four digits of the student ID, they are the values stored in the latches, and operations are carried out on these values.
- The clock input allows the ALU, FSM, and the Latches to go through a flow, and change values accordingly to a cycle.
- The student_id input allows direct outputs and connections for the student ID number for other functions.
- OP is the 16-bit input given by the control unit to the ALU, the OP acts as instructions determining which operation to select for the ALU.
- Neg determines if the value of the output is negative(‘1’) or positive(‘0’)
- The reset inputs(Reset_A, Reset_B, FMS_Reset) allows the functions to access the reset option that will return them to the beginning; this allows control to the flow and changes made.
- R1 and R2 are the 4-bit outputs executed by the ALU core; this is the final result(8-bit) broken into two parts(fistfourbits, lastfourbits), in this case R1 and R2 are the same, since they represent either a ‘yes’ or a ‘no’ that is repeated twice when it is displayed.
- Enablers(data_in, DEC_Enable) are simple inputs which allow input to be processed when it's ‘1’, and block inputs when it’s ‘0’.

Table : ALU_3 Microcode and Operations

Function #	Microde	Boolean Operation/Function
1	**0000000000000001**	**Output a “n” for no[Even]**
2	**0000000000000010**	**Output a “y” yes[odd]**
3	**0000000000000100**	**Output a “n” for no[Even]**
4	**0000000000001000**	**Output a “y” yes[odd]**
5	**0000000000010000**	**Output a “n” for no[Even]**
6	**0000000000100000**	**Output a “y” yes[odd]**
7	**0000000001000000**	**Output a “n” for no[Even]**
8	**0000000010000000**	**Output a “y” yes[odd]**
9	**0000000100000000**	**Output a “n” for no[Even]**

Table 2: Result Value for student number 501169402

Student ID	Output(TESTNUM1 and TESNUM 2)
5	0001
0	0000
1	0001
1	0000
6	0001
9	0000
4	0001
0	0000
2	0001

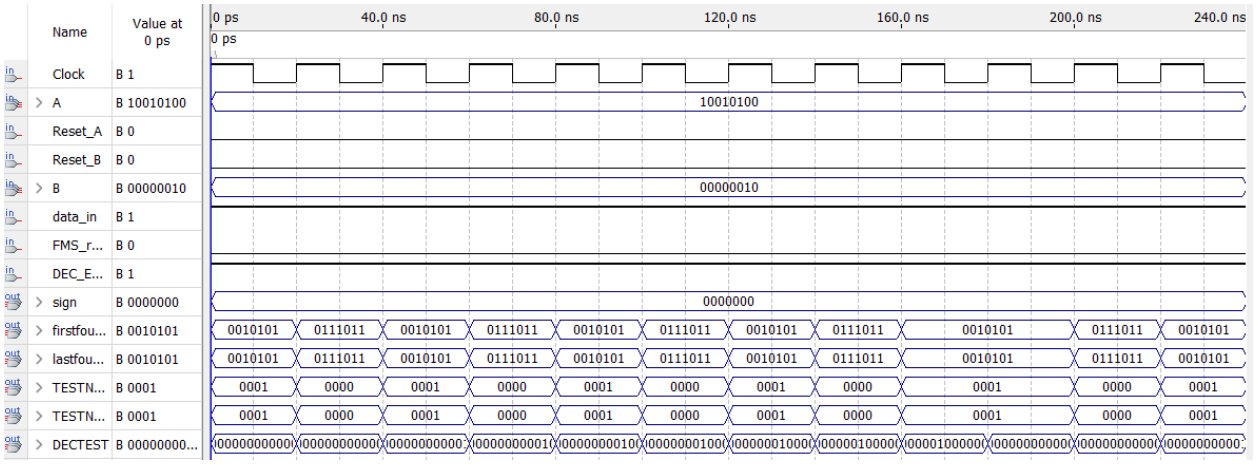


Figure 3: Waveform for ALU_3

Conclusion:

In conclusion, through lab 6 the very concept of the ALU was shown through practical means. By completing the lab, the implementation of a simple general processor unit was made. As it could be seen, for all the different versions of ALU(ALU_1, ALU_2, ALU_3), the waveforms did certainly match with the expected results from the tables; the implementation of the ALU and it's codes were definitely correct, and it complemented well with the theoretical concepts of this lab. Although the concept of ALU was new, the implementation and functionality of the ALU was not something that was new, it was something that was built up to since the first lab; the ALU is just combination of all the labs and it clearly shows how a processing unit is not just made up of one or two functions and units, it is made up many different components all working in different ways, and merging together to make a functional unit.