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

Course Title:	**Digital Systems**
Course Number:	**COE 328**
Semester/Year (e.g.F2016)	**F2020**

Instructor:	Vadim Guerkov

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

Submission Date:	**7/12/2020**
Due Date:	**7/12/2020**

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
Rahman	Taskin	500973553	11	N.R

Table of Contents

Introduction

The objective of this lab experiment is to design and build Arithmetic Logic Unit (ALU) using VHDL code, and to implement that code to block diagram schematic with multiple components which work together to function as a unit (Figure 1). These components consist of 2 latches (memory devices), a control unit, which consists of a Finite State Machine (FSM) and a 4-16 Decoder, and finally a seven-segment display for the output. The ALU will be performing a set of arithmetical and logical functions with two 8-bit inputs. The purpose of the *latches* is to store these two 8-bit binary values in memory and provides them as input to the main component of the ALU, called the core which performs arithmetical and logical operations. The *control* unit is responsible for giving instructions and signals to the core that performs tasks. This is done with an FSM and a 4-16 Decoder which work in unison to provide a command to the core, such as, "display the sum of the two inputs", using the assigned microcode values. Lastly the output is fed into a binary to seven segment display, to display the output.

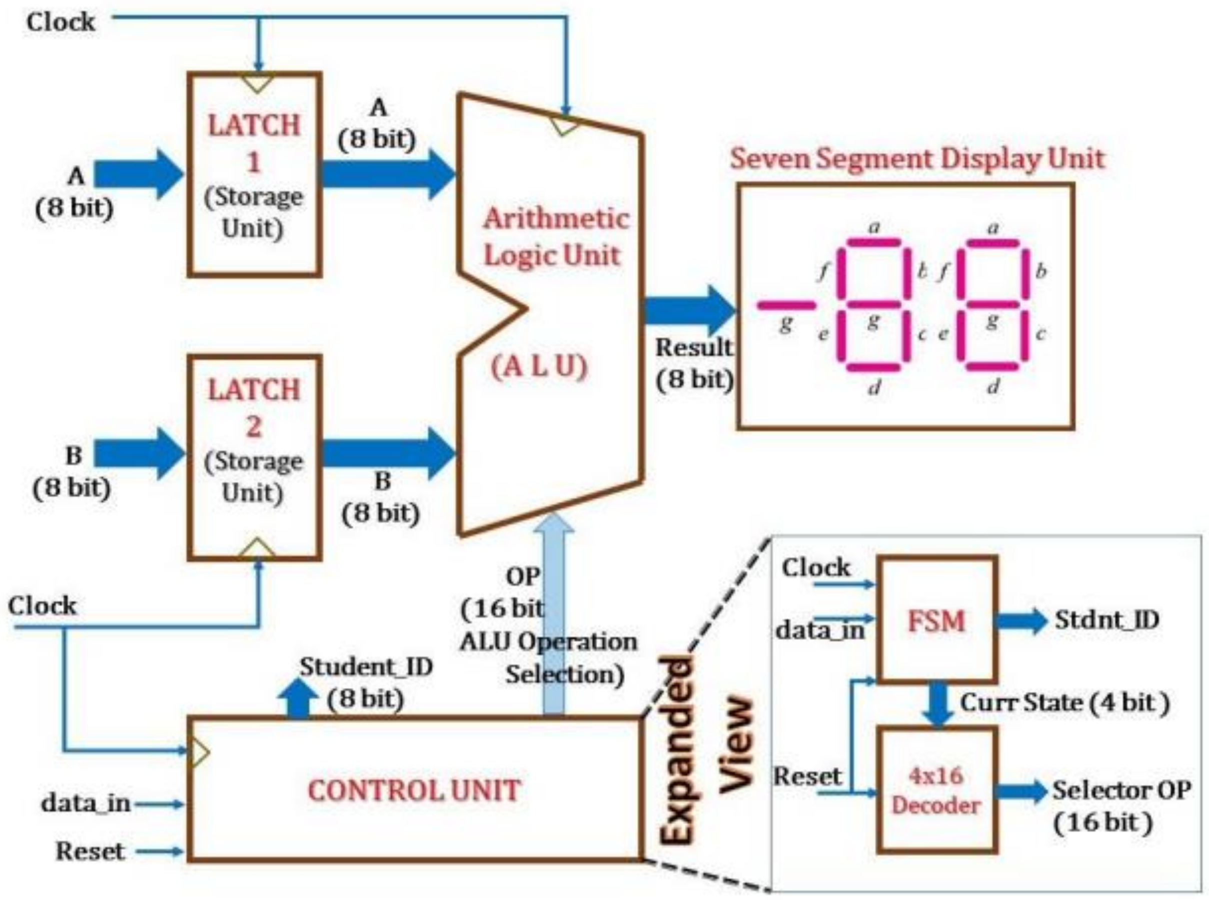


Figure 1. General diagram of the ALU showing the interconnection between all components

Components

The components (besides the ALU core) as already mentioned in include the following:

- Latch 1
- Latch 2
- FSM (Finite State Machine)
- 4-to-16 Decoder
- Seven-segment display

The latches store memory of the inputs to the ALU, and the FSM and Decoder signal the instructions to the core, the ALU core then carries out the task assigned from the decoder, finally the seven-segment displays the output of the ALU operation.

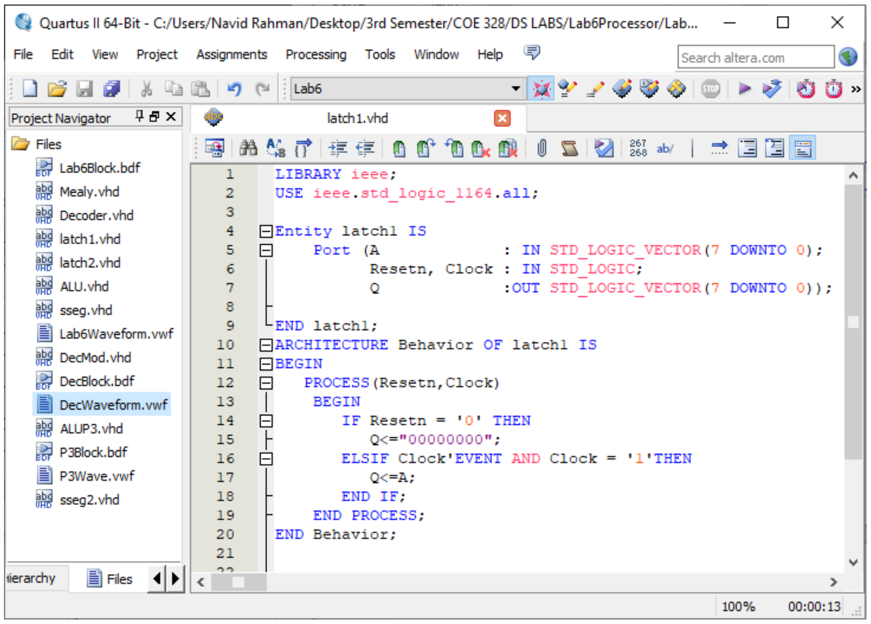
Latch 1

The first latch stores the one of the inputs, which we can call input A, which is an 8-bit binary number. It stores this values in an 8-bit register, the register reads the 8-bit input on its input terminal and passes it to the output port, but this happens only at the rising-edge of the clock. Since the latch is edge-triggered, the input is passed to the output only when the rising-edge of the clock input signal goes into the latch. The latch also has a reset input, where, if it is a binary 0 it passes the 8-bit binary value for 0 to the output, and if 1 it passes the input value to the output. The reset input is independent of the clock, which means this reset is an asynchronous reset.

Truth table

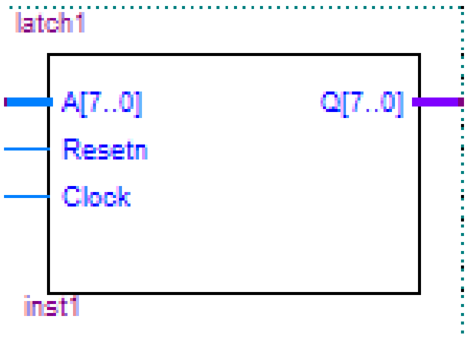
Reset	Clock (Rising edge)	A (Any 8-Bit Input)	Q (8-Bit output)
0	**0**	**A**	00000000
0	**1**	**A**	00000000
1	**0**	**A**	Latch (Previous Output)
1	**1**	**A**	A

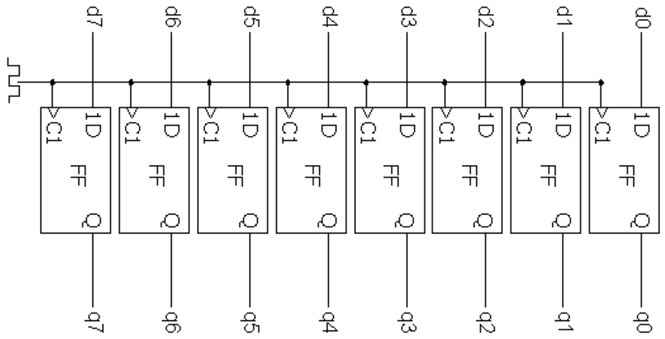
VHDL code



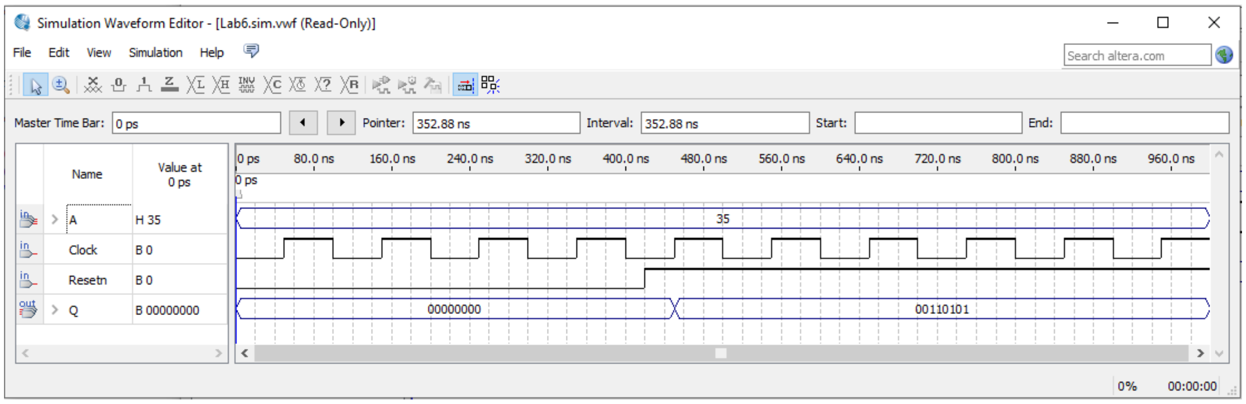
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

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 = '0' THEN
          Q<="00000000";
       ELSIF Clock'EVENT AND Clock = '1'THEN
          Q<=A;
       END IF;
    END PROCESS;
END Behavior;
```

Block Diagram/Schematic





Waveform


Simulation Waveform Editor - [Lab6.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com

Master Time Bar: 0 ps | Pointer: 352.88 ns | Interval: 352.88 ns | Start: | End:

Name	Value at 0 ps
A	H 35
Clock	B 0
Resetn	B 0
Q	B 00000000

0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns

A: 35

Q: 00000000 → 00110101

0% 00:00:00

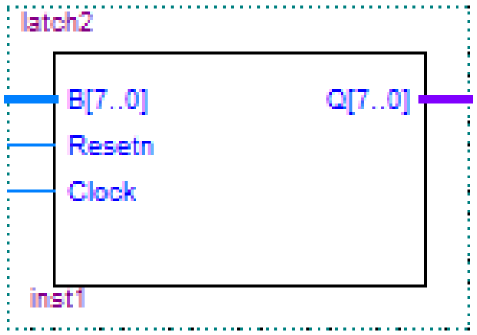
Latch 2

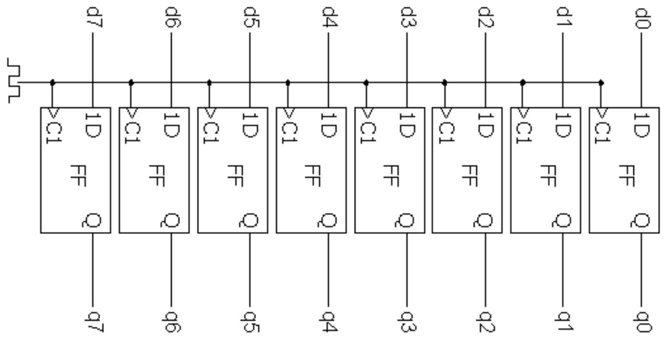
This latch is the exact same as the first latch mentioned above the only difference being that this latch stores the second 8-bit input for the ALU, like the first latch this latch operates with an 8-bit register, with the same clock and reset characteristics.

Truth table

Reset	Clock (Rising edge)	B (Any 8-Bit Input)	Q (8-Bit output)
0	**0**	**B**	00000000
0	**1**	**B**	00000000
1	**0**	**B**	Latch (Previous Output)
1	**1**	**B**	B

Block Diagram/Schematic

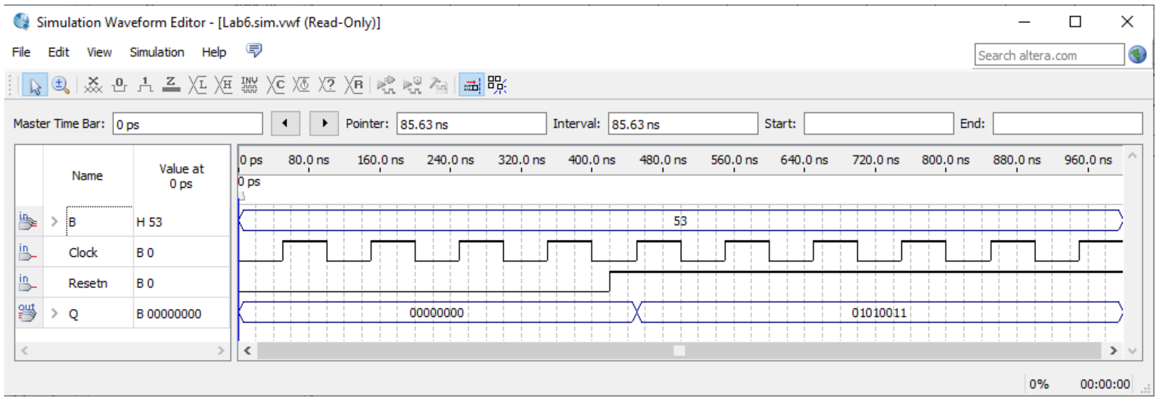




VHDL Code

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

Entity latch2 IS
    Port (B : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          Resetn, Clock : IN STD_LOGIC;
          Q:  OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END latch2;

ARCHITECTURE Behavior OF latch2 IS
BEGIN
   PROCESS(Resetn,Clock)
    BEGIN
       IF Resetn = '0' THEN
            Q<="00000000";
      ELSIF Clock'EVENT AND Clock = '1'THEN
          Q<=B;
        END IF;
    END PROCESS;
END Behavior;
```

Waveform



Master Time Bar: 0 ps Pointer: 85.63 ns Interval: 85.63 ns Start: End:

Name	Value at 0 ps
B	H 53
Clock	B 0
Resetn	B 0
Q	B 00000000

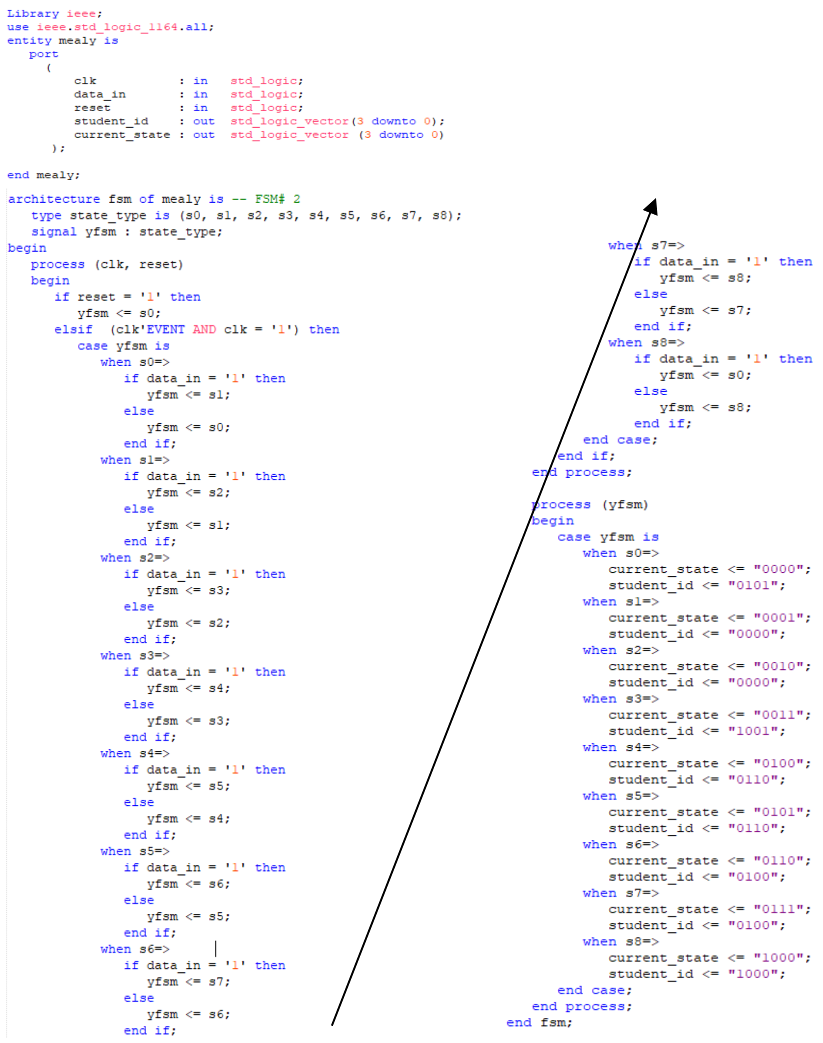
FSM (Finite-State Machine)

The FSM is a device that consists of an n-bit shift register, which is comprised of n flip-flops and 2 combinational circuits and a switch. One combinational circuit which is combined with the switch, is used to determine the next input to the register, the other combinational circuit is used to determine the output. The machine starts at a default state and shifts to the next state at the rising edge of the clock and repeats this process until it reaches the last state in cycle, then, it repeats this process as long as there is a clock rising edge, there is also a reset input which is used to reset the machine back to its default state (starting position). The next state is determined by the next input to the shift register, which is determined using a combinational circuit using the previous inputs and a switch, and the final output is determined using a combinational circuit which uses the output of the shift register to determine the final output. The Mealy FSM is a type of finite state machine in which the final output depends on the state of the switch, this is type used in this lab experiment. Finally, a diagram showing the process of the FSM of shifting from one state to another can be seen in **Figure 2**, the 1 or 0 beside the arrows refers to the state of the switch, and the output in blue represents the final output.

Truth table

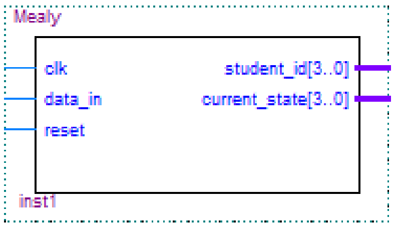
Reset	Clock	Data-In	Student-Id	Current-State	State
1	1	1	0101	0000	1
1	1	1	0000	0001	2
1	1	1	0000	0010	3
1	1	1	1001	0011	4
1	1	1	0110	0100	5
1	1	1	0110	0101	6
1	1	1	0100	0110	7
1	1	1	0100	0111	8
1	1	1	1000	1000	9

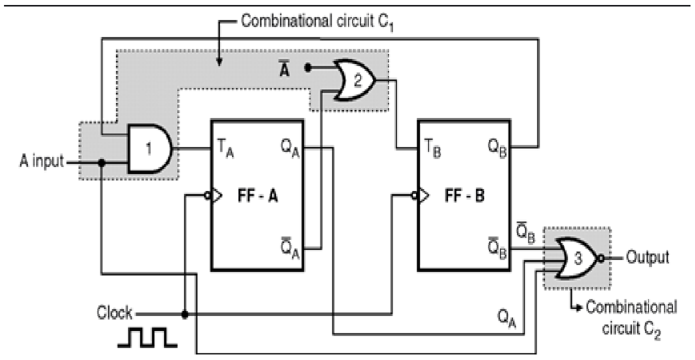
VHDL code



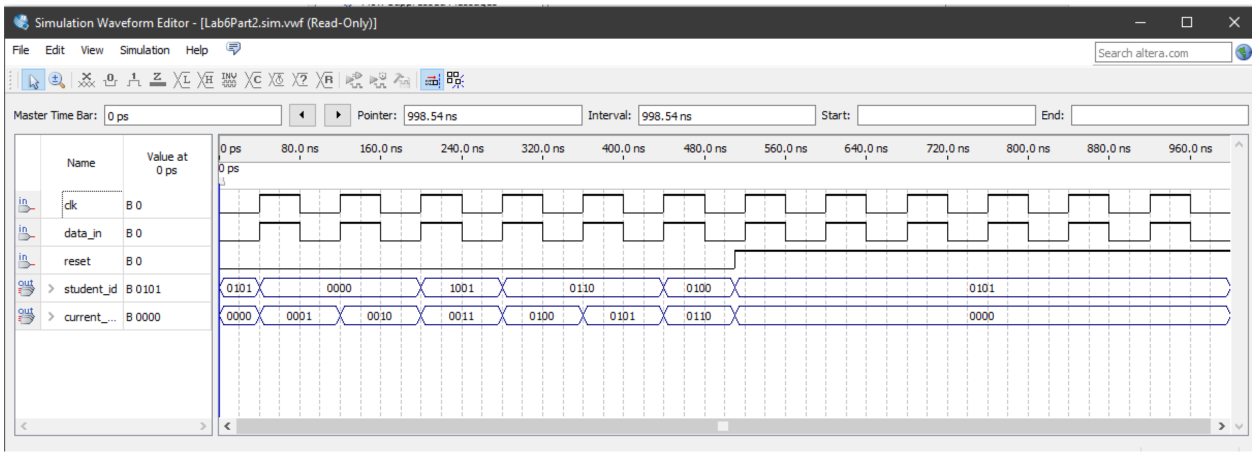
```
Library ieee;
use ieee.std_logic_1164.all;
entity mealy is
   port
     (
         clk           : in   std_logic;
         data_in       : in   std_logic;
         reset         : in   std_logic;
         student_id    : out  std_logic_vector(3 downto 0);
         current_state : out  std_logic_vector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

end mealy;

architecture fsm of mealy is -- FSM# 2
   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
         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)
   begin
      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 <= "0000";
         when s3=>
            current_state <= "0011";
            student_id <= "1001";
         when s4=>
            current_state <= "0100";
            student_id <= "0110";
         when s5=>
            current_state <= "0101";
            student_id <= "0110";
         when s6=>
            current_state <= "0110";
            student_id <= "0100";
         when s7=>
            current_state <= "0111";
            student_id <= "0100";
         when s8=>
            current_state <= "1000";
            student_id <= "1000";
      end case;
   end process;
end fsm;
```

Block Diagram/Schematic





Waveform


Simulation Waveform Editor - [Lab6Part2.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 998.54 ns
Interval: 998.54 ns
Start:
End:
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns

Name	Value at 0 ps
clk	B 0
data_in	B 0
reset	B 0
student_id	B 0101
current_...	B 0000

```
student_id: 0101 | 0000 | 1001 | 0110 | 0100 | 0101
current_...: 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0000
```

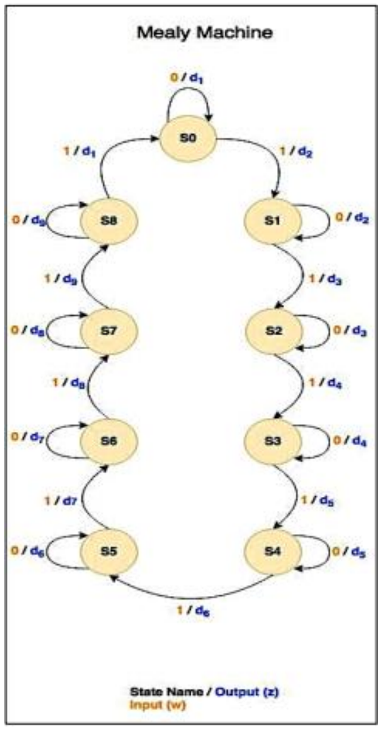


Figure 2: Mealy machine state diagram

4-to-16 Decoder

The 4-to-16 Decoder, provides the intructions to the ALU core, with a 16-bit binary output, which indicates the task that the core should perform. The Decoder takes the 4-bit binary output from Mealy machine, and decodes it into the 16-bit binary value that corresponds to the 4-bit FSM value. The Decoder has an enable input, which either diables or enable the device, if the Decoder is disabled it outputs the 16-bit binary value that corresponds to 0. (Note: The output of the Decoder is reversed for practicality purposes).

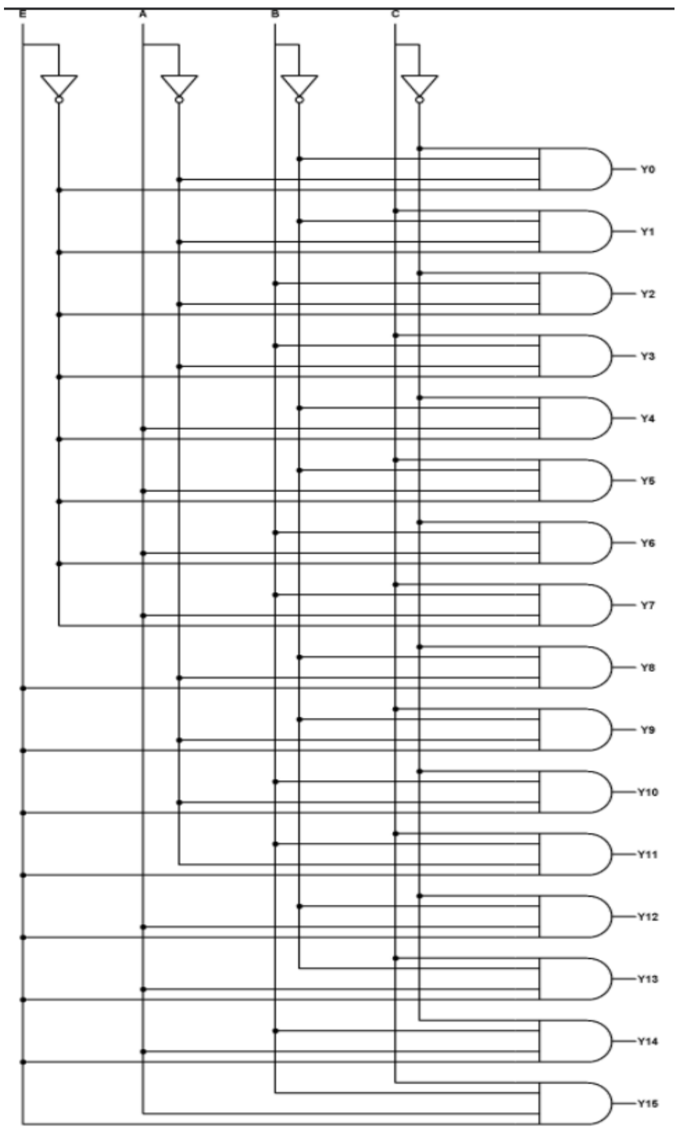
Truth table

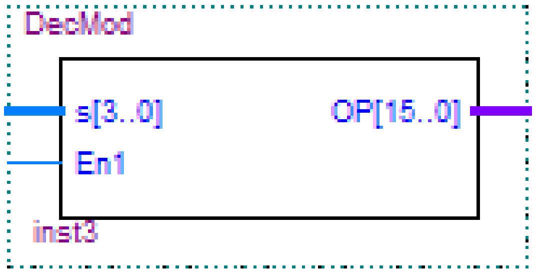
En	S (4-Bit input)	OP (16-Bit output)	OP-Reversed
0	**X**	0000000000000000	0000000000000000
1	**0000**	1000000000000000	0000000000000001
1	**0001**	0100000000000000	0000000000000010
1	**0010**	0010000000000000	0000000000000100
1	**0011**	0001000000000000	0000000000001000
1	**0100**	0000100000000000	0000000000010000
1	**0101**	0000010000000000	0000000000100000
1	**0110**	0000001000000000	0000000001000000
1	**0111**	0000000100000000	0000000010000000
1	**1000**	0000000010000000	0000000100000000
1	**1001**	0000000001000000	0000001000000000
1	**1010**	0000000000100000	0000010000000000
1	**1011**	0000000000010000	0000100000000000
1	**1100**	0000000000001000	0001000000000000
1	**1101**	0000000000000100	0010000000000000
1	**1110**	0000000000000010	0100000000000000
1	**1111**	0000000000000001	1000000000000000

VHDL code

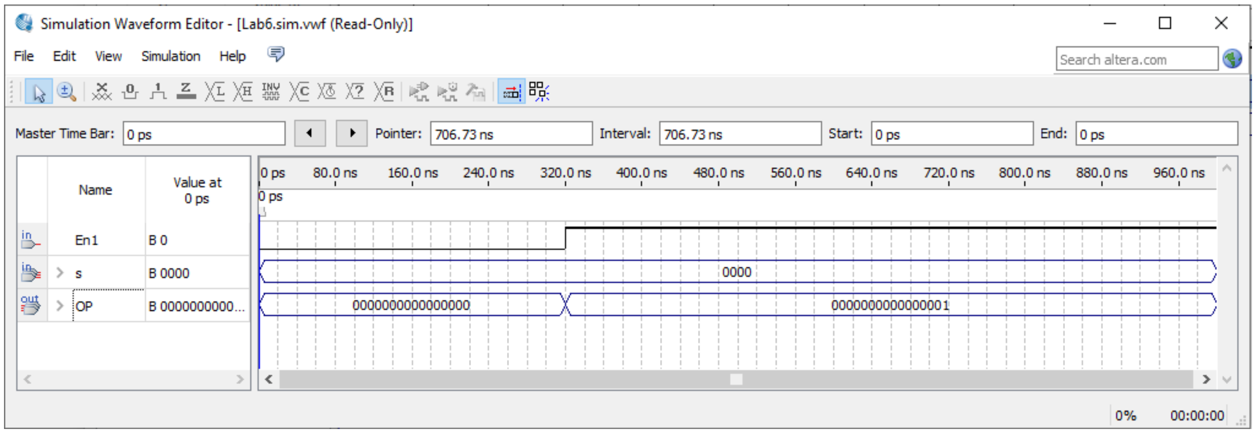
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY work;
USE work.Decoder_package.all;

ENTITY decode4to16 IS
   PORT(s    : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        Enl  : IN STD_LOGIC;
        OP   : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
   END decode4to16;

ARCHITECTURE Structure OF decode4to16 IS
   SIGNAL m: STD_LOGIC_VECTOR(0 TO 1);
BEGIN
   m(0) <=(NOT s(3) AND Enl);
   m(1)<=(s(3) AND Enl);
    Dec1: Decoder PORT MAP
       (s(2 DOWNTO 0), m(0), OP(7 DOWNTO 0));
    Dec2: Decoder PORT MAP
       (s(2 DOWNTO 0), m(1),OP(15 DOWNTO 8));
END STRUCTURE;
```

Block Diagram/Schematic





Waveform


Simulation Waveform Editor - [Lab6.sim.vwf (Read-Only)]

Master Time Bar: 0 ps Pointer: 706.73 ns Interval: 706.73 ns Start: 0 ps End: 0 ps

Name	Value at 0 ps
En1	B 0
s	B 0000
OP	B 0000000000...

s: 0000
OP: 0000000000000000, 0000000000000001

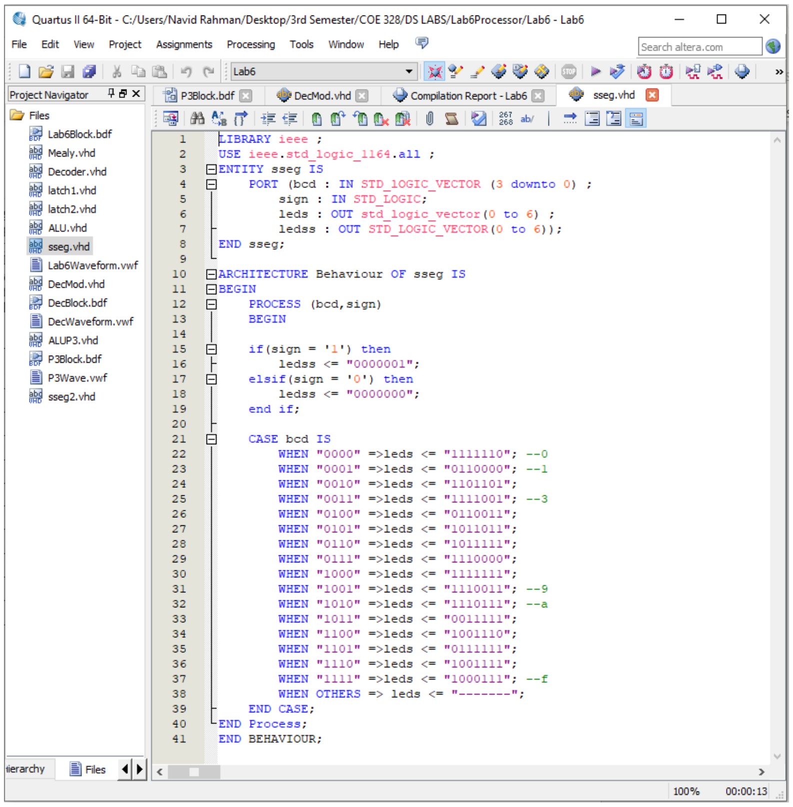
Seven-Segment Display

The seven-segment display takes the output of the ALU core, which is an 8-bit binary values and converts it to its corresponding seven-segment value, which is a 7-bit binary value. The original 8-bit binary value is converted by separating the 8-bit value to two 4-bit binary values, one for the first four bits and one for the last four. These 4-bit values are then converted to its corresponding seven-segment binary value. The 1's in this new seven-segment binary value indicate which of the 7 led lit segments of the display are active, hence the name. The binary bit has a corresponding letter assigned to it, going to the letter a to the letter g from the most significant bit to the least, this can be seen in the truth table. Also, the output will have an associated sign value to indicate whether the output from the ALU is positive or negative.

Truth table

Decimal	Binary DCBA	7 Segment Code a b c d e f g
0	0000	1 1 1 1 1 1 0
1	0001	0 1 1 0 0 0 0
2	0010	1 1 0 1 1 0 1
3	0011	1 1 1 1 0 0 1
4	0100	0 1 1 0 0 1 1
5	0101	1 0 1 1 0 1 1
6	0110	0 0 1 1 1 1 1
7	0111	1 1 1 0 0 0 0
8	1000	1 1 1 1 1 1 1
9	1001	1 1 1 0 0 1 1

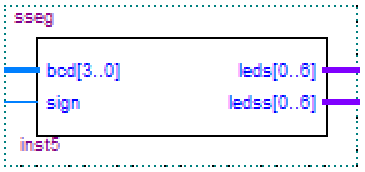
VHDL code

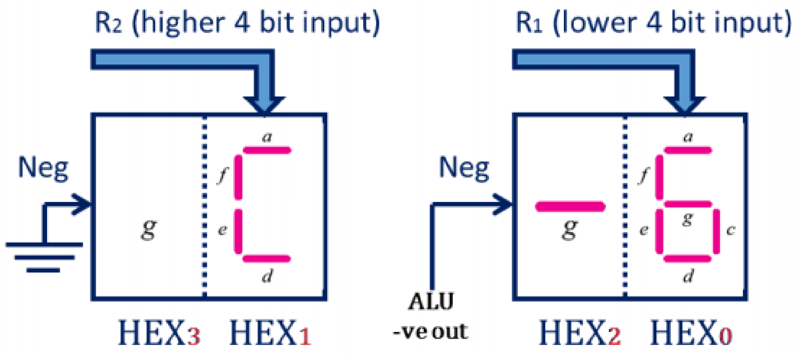


Quartus II 64-Bit - C:/Users/Navid Rahman/Desktop/3rd Semester/COE 328/DS LABS/Lab6Processor/Lab6 - Lab6

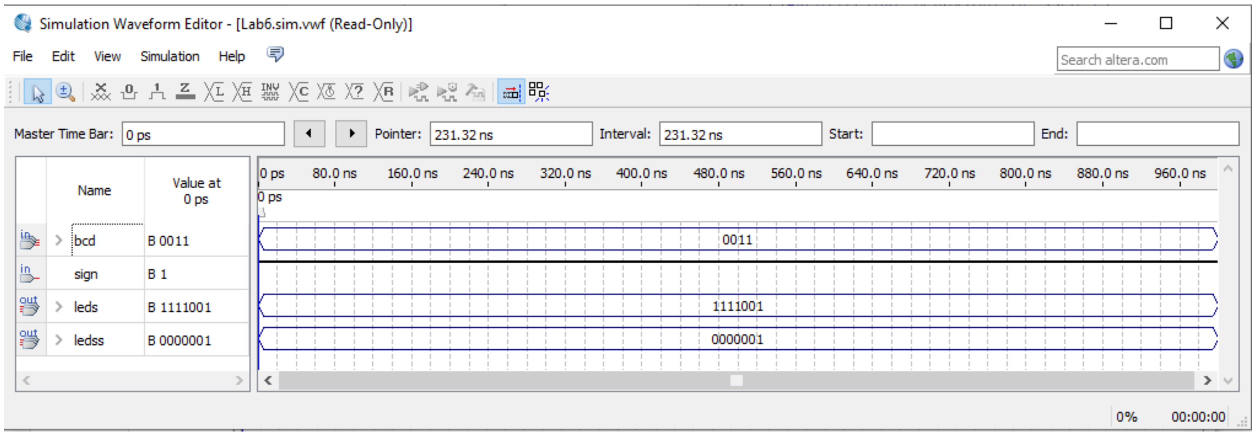
```vhdl
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY sseg IS
    PORT (bcd : IN STD_LOGIC_VECTOR (3 downto 0) ;
        sign : IN STD_LOGIC;
        leds : OUT std_logic_vector(0 to 6) ;
        ledss : OUT STD_LOGIC_VECTOR(0 to 6));
END sseg;

ARCHITECTURE Behaviour OF sseg IS
BEGIN
    PROCESS (bcd,sign)
    BEGIN

    if(sign = '1') then
        ledss <= "0000001";
    elsif(sign = '0') then
        ledss <= "0000000";
    end if;

    CASE bcd IS
        WHEN "0000" =>leds <= "1111110"; --0
        WHEN "0001" =>leds <= "0110000"; --1
        WHEN "0010" =>leds <= "1101101";
        WHEN "0011" =>leds <= "1111001"; --3
        WHEN "0100" =>leds <= "0110011";
        WHEN "0101" =>leds <= "1011011";
        WHEN "0110" =>leds <= "1011111";
        WHEN "0111" =>leds <= "1110000";
        WHEN "1000" =>leds <= "1111111";
        WHEN "1001" =>leds <= "1110011"; --9
        WHEN "1010" =>leds <= "1110111"; --a
        WHEN "1011" =>leds <= "0011111";
        WHEN "1100" =>leds <= "1001110";
        WHEN "1101" =>leds <= "0111111";
        WHEN "1110" =>leds <= "1001111";
        WHEN "1111" =>leds <= "1000111"; --f
        WHEN OTHERS => leds <= "-------";
    END CASE;
END Process;
END BEHAVIOUR;
```

Block Diagram/Schematic





Waveform



Simulation Waveform Editor - [Lab6.sim.vwf (Read-Only)]

Master Time Bar: 0 ps   Pointer: 231.32 ns   Interval: 231.32 ns   Start: End:

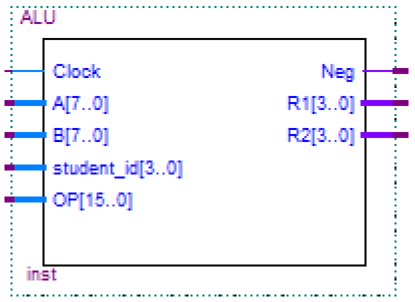
Name	Value at 0 ps	Waveform
bcd	B 0011	0011
sign	B 1	
leds	B 1111001	1111001
ledss	B 0000001	0000001

ALU_1: Problem Set 1

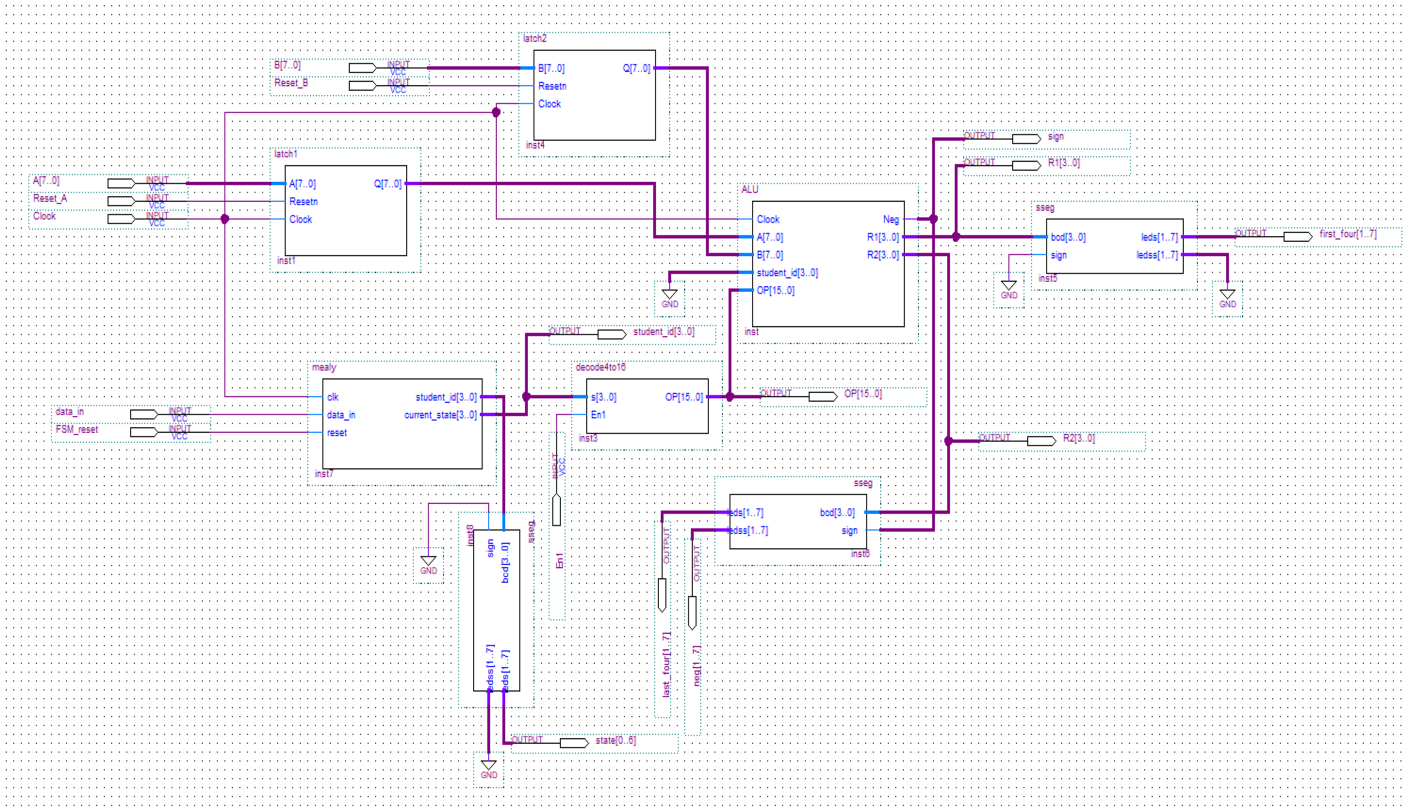
The first part of the experiment was to design an ALU core which performs the first set of tasks for the assigned microcode values. For the second and third problem sets the tasks performed by the core is simply changed for the assigned microcode. The tasks to be performed by the ALU core for the first problem set are as follows:

1. Sum of the two inputs, A and B.
2. Difference of the two inputs, A and B.
3. Boolean NOT of input A.
4. Boolean NAND of the two inputs, A and B.
5. Boolean NOR of A and B.
6. Boolean AND of A and B.
7. Boolean OR of A and B.
8. Boolean XOR of A and B.
9. Boolean XNOR of A and B.

Block Diagram



ALU core



ALU Unit

Inputs/Outputs

- **Clock** – The clock input determines when the ALU will read the instruction sent to it by the 4-to-16 Decoder. At the rising edge of the clock, the ALU core reads the 16-bit binary value output from the Decoder, and then uses is to determine what task to perform.
- **A[7..0]** (8-bit) – As mentioned before it is one of the two inputs stored in one of the latches and is used to perform arithmetic and logic function on.
- **B[7..0]** (8-bit) – This is the other input used to perform calculations along with the first input (Input A).
- **Student_id[3..0]** – The student id input is a 4-bit binary value provided by the FSM's combinational circuit, it represents the particular digit of our student id in binary, that corresponds

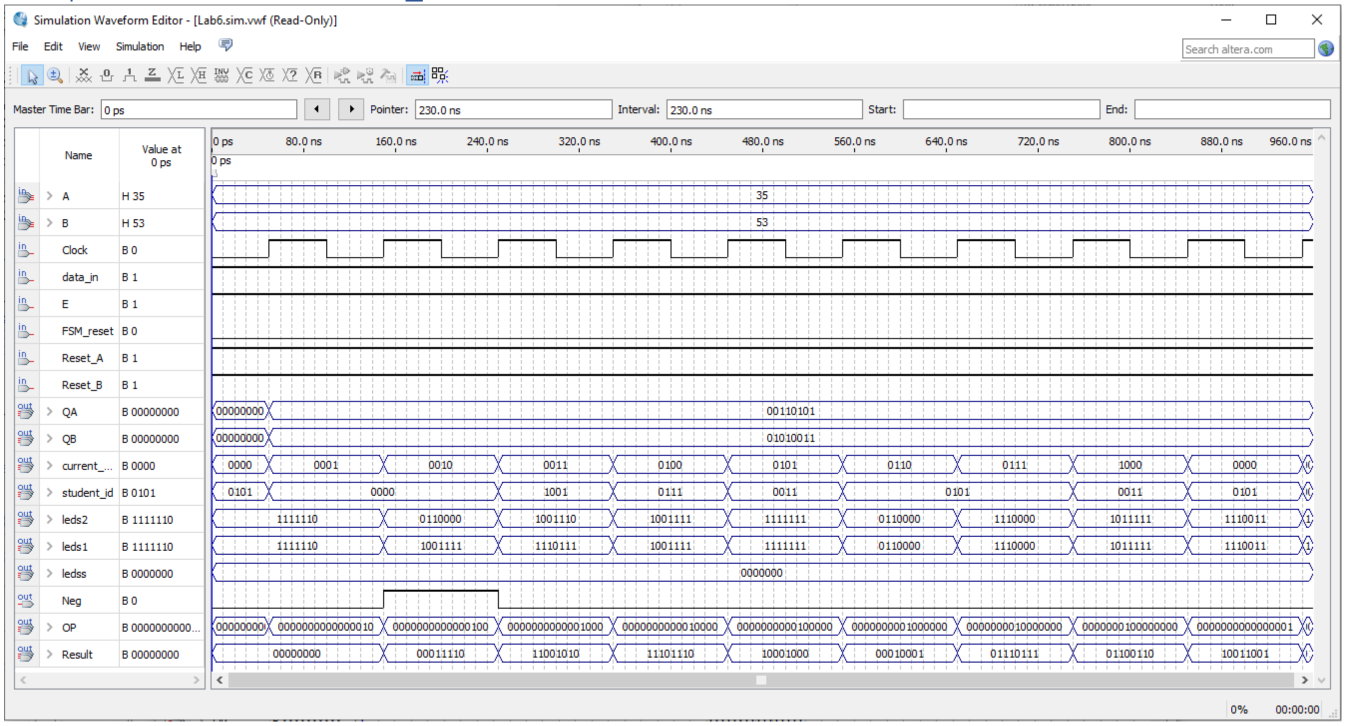
to the state that it is assigned to. The first digit is outputted during the first state of the FSM's state sequence, and the second digit with the second state, etc.

- **OP[15..0]** – The OP input represents the 16-bit decoded value from the Decoder, and it is used to tell the ALU core, what tasks to perform. The core is designed to read a specific 16-bit (microcode) value and perform a task associated with that value. Ex. (0000000000000001 → Sum of the two inputs, A and B).
- **Neg** – This output tells us if the final output value of the ALU is positive or negative and is displayed using a seven-segment display.
- **R1[3..0]** – This output represents the first four bits of the final output.
- **R2[3..0]** – This output represents the last four bits of the final output.

Table of Microcode's Generated by Decoder for ALU_1

Number of Functions	**OP**	**Instructions/Functions (Arithmetic or Logical)**
1	000000000000000**1**	Addition
2	00000000000000**1**0	Subtraction (With Sign)
3	0000000000000**1**00	NOT (Inverter)
4	000000000000**1**000	NAND
5	00000000000**1**0000	NOR
6	0000000000**1**00000	AND
7	000000000**1**000000	OR
8	00000000**1**0000000	XOR
9	0000000**1**00000000	XNOR
10	000000**1**000000000	Previous Output
11	00000**1**0000000000	Previous Output
12	0000**1**00000000000	Previous Output
13	000**1**000000000000	Previous Output
14	00**1**0000000000000	Previous Output
15	0**1**00000000000000	Previous Output
16	**1**000000000000000	Previous Output

Complete Waveform for ALU_1



Simulation Waveform Editor - [Lab6.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
0%
00:00:00

Master Time Bar: 0 ps Pointer: 230.0 ns Interval: 230.0 ns Start: End:

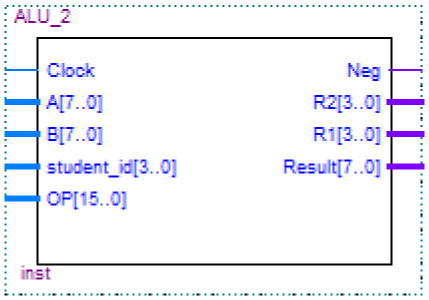
Name	Value at 0 ps	Waveform
A	H 35	35
B	H 53	53
Clock	B 0	
data_in	B 1	
E	B 1	
FSM_reset	B 0	
Reset_A	B 1	
Reset_B	B 1	
QA	B 00000000	00000000, 00110101
QB	B 00000000	00000000, 01010011
current_...	B 0000	0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 0000
student_id	B 0101	0101, 0000, 1001, 0111, 0011, 0101, 0011, 0101
leds2	B 1111110	1111110, 0110000, 1001110, 1001111, 1111111, 0110000, 1110000, 1011111, 1110011
leds1	B 1111110	1111110, 1001111, 1110111, 1001111, 1111111, 0110000, 1110000, 1011111, 1110011
ledss	B 0000000	0000000
Neg	B 0	
OP	B 0000000000...	00000000, 0000000000000010, 0000000000000100, 0000000000001000, 0000000000010000, 0000000000100000, 0000000001000000, 0000000010000000, 0000000100000000, 0000000000000001
Result	B 00000000	00000000, 00011110, 11001010, 11101110, 10001000, 00010001, 01110111, 01100110, 10011001

ALU_2: Problem Set 2

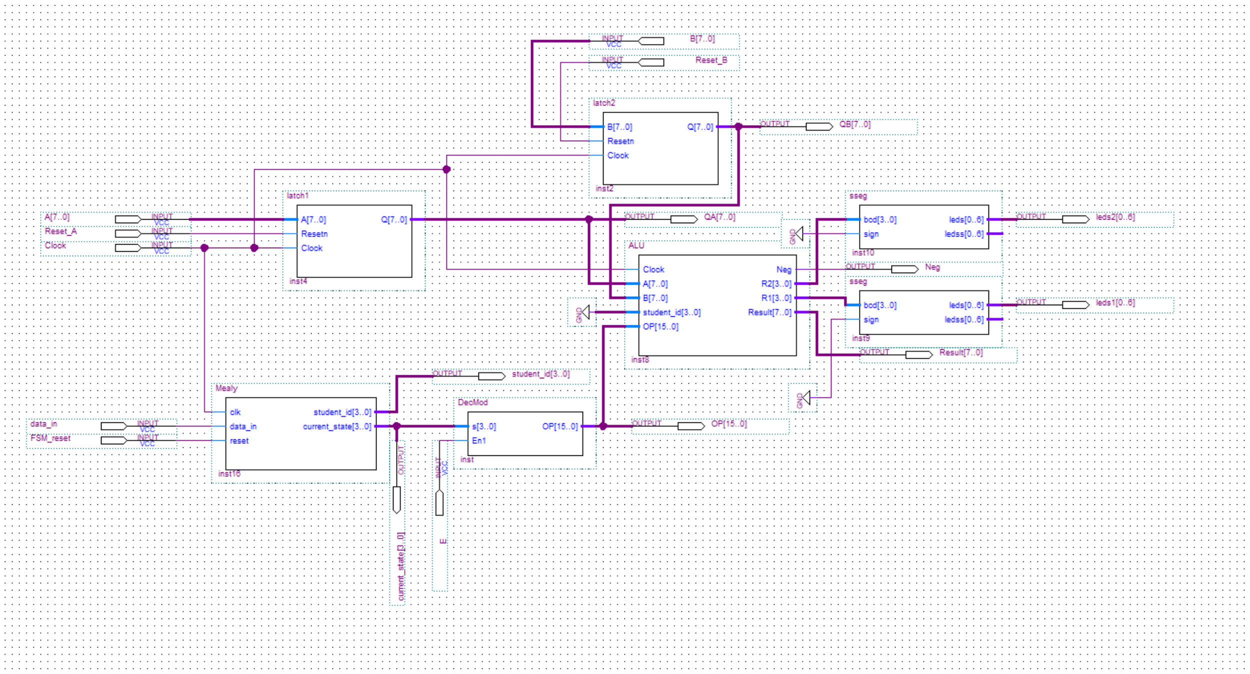
For problem set 2, the only things that changed from the first problem set are the tasks to be performed by the ALU, for each microcode value. The set of tasks for this problem set are as follows:

1. Decrement the input B by 5.
2. Swap the lower and upper 4-bits of input B.
3. Shift input A to the left by 2 bits, input bit = 0 (SHL).
4. Produce the result of NANDing A and B.
5. Find the greater value between inputs A and B and produce the results (Max(A, B))
6. Invert the even bits of input B
7. Produce null on the output
8. Replace the upper four bits of B by the upper four bits of A
9. Show A on the output

Block Diagram/Schematic



ALU_2 core



ALU_2 Unit

Inputs/Outputs

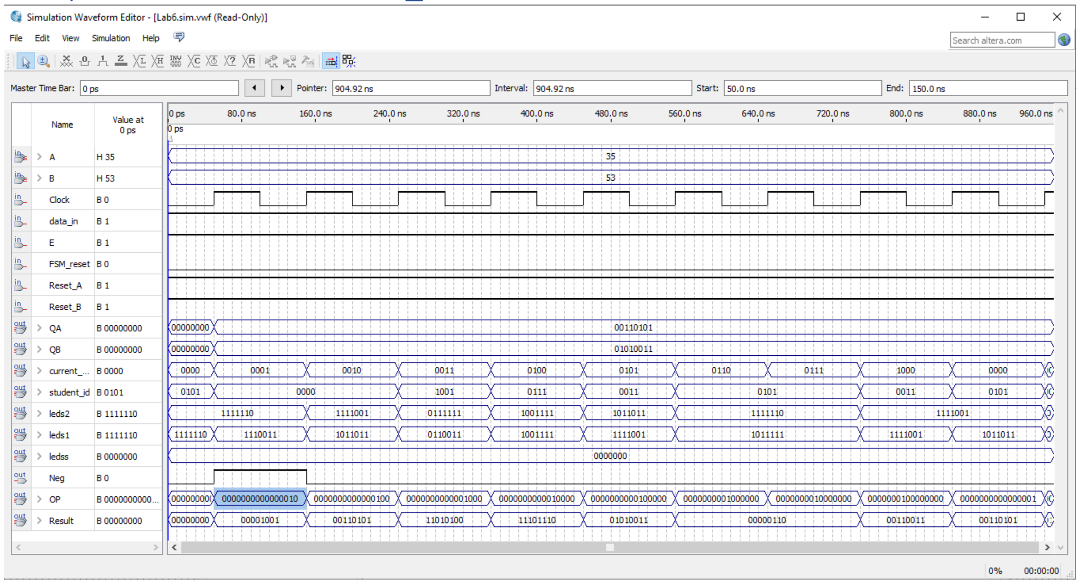
- **Clock** – The clock input determines when the ALU will read the instruction sent to it by the 4-to-16 Decoder. At the rising edge of the clock, the ALU core reads the 16-bit binary value output from the Decoder, and then uses is to determine what task to perform.
- **A[7..0]** (8-bit) – As mentioned before it is one of the two inputs stored in one of the latches and is used to perform arithmetic and logic function on.
- **B[7..0]** (8-bit) – This is the other input used to perform calculations along with the first input (Input A).
- **Student_id[3..0]** – The student id input is a 4-bit binary value provided by the FSM's combinational circuit, it represents the particular digit of our student id in binary, that corresponds to the state that it is assigned to. The first digit is outputted during the first state of the FSM's state sequence, and the second digit with the second state, etc.

- **OP[15..0]** – The OP input represents the 16-bit decoded value from the Decoder, and it is used to tell the ALU core, what tasks to perform. The core is designed to read a specific 16-bit (microcode) value and perform a task associated with that value. Ex. (0000000000000001 → Sum of the two inputs, A and B).
- **Neg** – This output tells us if the final output value of the ALU is positive or negative and is displayed using a seven-segment display.
- **R1[3..0]** – This output represents the first four bits of the final output.
- **R2[3..0]** – This output represents the last four bits of the final output.

Table of Microcode's Generated by Decoder for ALU_2

Function Number	**OP**	**Instructions/Functions (Arithmetic or Logical)**
1	000000000000000**1**	Decrement B by 9
2	00000000000000**1**0	Swap the lower and upper 4-bits of B
3	0000000000000**1**00	Shift A to left by 2-bits, input bit = 0 (SHL)
4	000000000000**1**000	Produce the result of NANDing A and B
5	00000000000**1**0000	Find the greater value of A and B and produce the results (Max(A, B))
6	0000000000**1**00000	Invert the even bits of B
7	000000000**1**000000	Produce null on the output
8	00000000**1**0000000	Replace the upper four bits of B by the upper four bits of A
9	0000000**1**00000000	Show A on the output
10	000000**1**000000000	Previous Output
11	00000**1**0000000000	Previous Output
12	0000**1**00000000000	Previous Output
13	000**1**000000000000	Previous Output
14	00**1**0000000000000	Previous Output
15	0**1**00000000000000	Previous Output
16	**1**000000000000000	Previous Output

Complete Waveform for ALU_2



```
Simulation Waveform Editor - [Lab6.sim.vwf (Read-Only)]
Master Time Bar: 0 ps   Pointer: 904.92 ns   Interval: 904.92 ns   Start: 50.0 ns   End: 150.0 ns

Name        Value at 0 ps     0 ps  80.0 ns  160.0 ns  240.0 ns  320.0 ns  400.0 ns  480.0 ns  560.0 ns  640.0 ns  720.0 ns  800.0 ns  880.0 ns  960.0 ns
A           H 35              35
B           H 53              53
Clock       B 0
data_in     B 1
E           B 1
FSM_reset   B 0
Reset_A     B 1
Reset_B     B 1
QA          B 00000000        00000000 | 00110101
QB          B 00000000        00000000 | 01010011
current_... B 0000            0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 0000
student_id  B 0101            0101 | 0000 | 1001 | 0111 | 0011 | 0101 | 0011 | 0101
leds2       B 1111110         1111110 | 1111001 | 0111111 | 1001111 | 1011011 | 1111110 | 1111001
leds1       B 1111110         1111110 | 1110011 | 1011011 | 0110011 | 1001111 | 1111001 | 1011111 | 1111001 | 1011011
ledss       B 0000000         0000000
Neg         B 0
OP          B 0000000000...   00000000 | 0000000000000010 | 0000000000000100 | 0000000000001000 | 0000000000010000 | 0000000000100000 | 0000000001000000 | 0000000010000000 | 0000000100000000 | 0000000000000001
Result      B 00000000        00000000 | 00001001 | 00110101 | 11010100 | 11101110 | 01010011 | 00000110 | 00110011 | 00110101
```

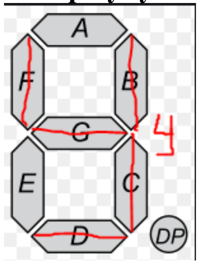
ALU_3: Problem Set 3

For problem set 3, the task of the ALU was to check if one of the digits from the 3rd last and 4th last digits of the student number 50097**35**53, was equal to the student id digit outputted from the FSM. If the condition is true the ALU should display 'y' using seven-segment display, and if not, it should display 'n'. The task assigned for this problem set is as follows:

1. Display 'y' if one of the two digits of input A (first 2 digits from the last four digits of the student number) are equal to the FSM Output (Student_id).
2. Display 'y' if none of the two digits of input A (first 2 digits from the last four digits of the student number) are equal to the FSM Output (Student_id).

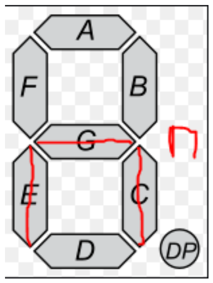
The output should look like the following:

Display 'y':



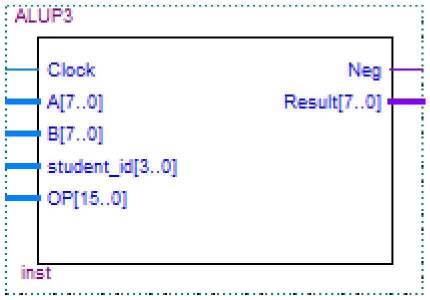
BCDFG = '1'

Display 'n':

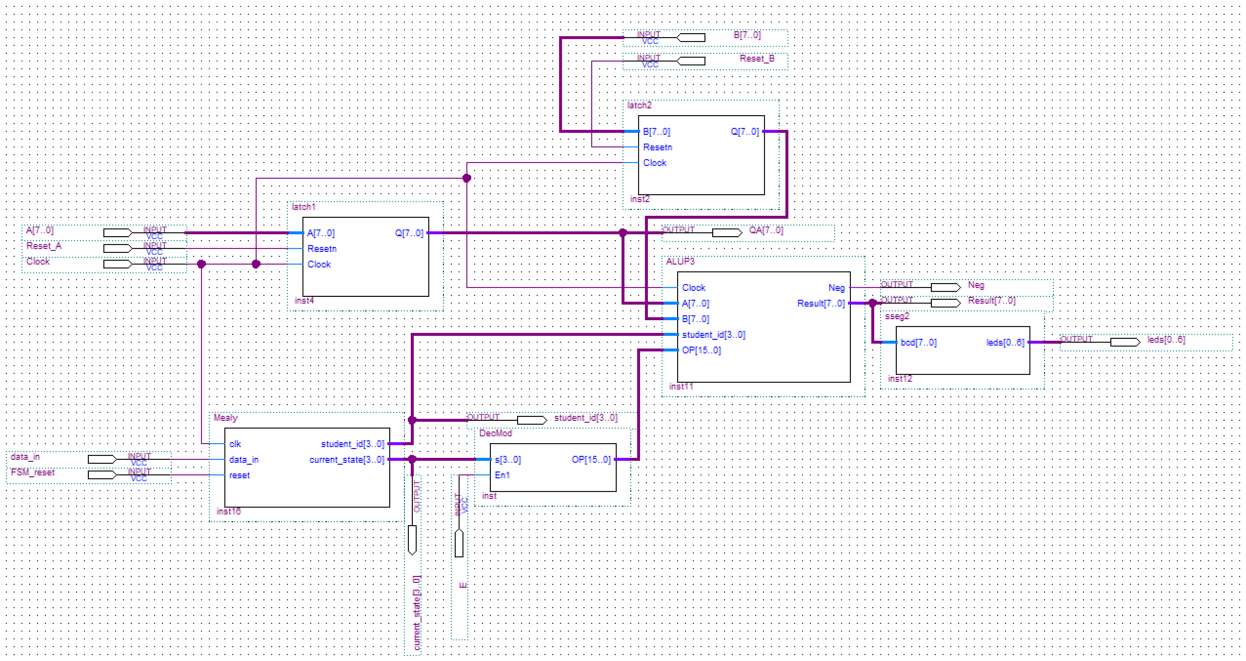


CEG = ‘1’

Block Diagram/Schematic



ALU_3 core



ALU_3 Unit

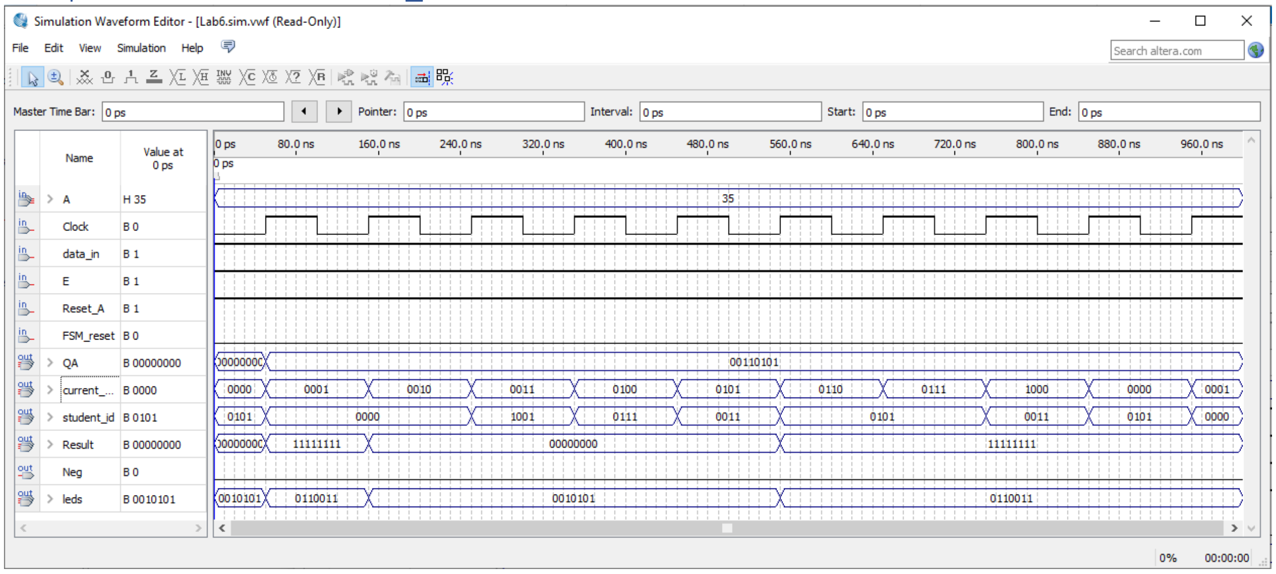
Inputs/Outputs

- **Clock** – The clock input determines when the ALU will read the instruction sent to it by the 4-to-16 Decoder. At the rising edge of the clock, the ALU core reads the 16-bit binary value output from the Decoder, and then uses is to determine what task to perform.
- **A[7..0]** (8-bit) – As mentioned before it is one of the two inputs stored in one of the latches and is used to perform arithmetic and logic function on.
- **B[7..0]** (8-bit) – This is the other input used to perform calculations along with the first input (Input A).
- **Student_id[3..0]** – The student id input is a 4-bit binary value provided by the FSM's combinational circuit, it represents the particular digit of our student id in binary, that corresponds to the state that it is assigned to. The first digit is outputted during the first state of the FSM's state sequence, and the second digit with the second state, etc.
- **OP[15..0]** – The OP input represents the 16-bit decoded value from the Decoder, and it is used to tell the ALU core, what tasks to perform. The core is designed to read a specific 16-bit (microcode) value and perform a task associated with that value. Ex. (0000000000000001 → Sum of the two inputs, A and B).
- **Neg** – This output tells us if the final output value of the ALU is positive or negative and is displayed using a seven-segment display.
- **Result[7..0]** – The final output is being displayed with this output, and it is an 8-bit binary value, and it is either all 1's or all 0's to denote a true or false value for the state of the condition being checked. This output will then be displayed by a conversion to a seven-segment value for 'y', or 'n'.

Table of Microcode's Generated by Decoder for ALU_3

Function Number	Student_id	Input B (35 in Hex to Binary)	Result	SSEG
1	0101	00110101	11111111	0111011 (yes)
2	0000	00110101	00000000	0010101 (no)
3	0000	00110101	00000000	0010101 (no)
4	1001	00110101	00000000	0010101 (no)
5	0111	00110101	00000000	0010101 (no)
6	0011	00110101	11111111	0111011 (yes)
7	0101	00110101	11111111	0111011 (yes)
8	0101	00110101	11111111	0111011 (yes)
9	0011	00110101	11111111	0111011 (yes)

Complete Waveform for ALU_3



Conclusion

An ALU (Arithmetic Logic Unit) was designed and implemented using first designing and building multiple components, then combining all components to assemble the ALU. These components were the, latches, FSM, Decoder, ALU core, and the seven-segment display. The components were built by first coding the function using VHDL code, then implemented by connecting them with all other components using the block diagram schematic. There were three parts to this lab experiment, the difference between each being the tasks that are to be performed by the ALU. From this lab I really learned to analyze waveform simulations, as certain outputs are greatly affected by inputs that are connected to other components and vice-versa. For example, the clock determines when all components activate, which means, since some components inputs (ALU core) are reliant on the output of other components there is a delay between the desired output its corresponding input. This is a result of the clock input going into all components at the exact same time, which causes the ALU core to receive the correct input with a delay of one clock cycle. In summary I learned how a simple ALU works, with its multiple components, and the purpose of each component, and I learned step by step process of ALU, from the input being received to the output being displayed.