Lab 6 - Design of a Simple General-Purpose Processor

Name: Reza Aablue

Student Number # : 500966944

Instructor: Dr. Reza Sedaghat

TA: Mr. Nauman Baig

Date Submitted: August 08, 2020

Submission Deadline: August 09, 2020

Term: Summer 2020

Course: Digital System (COE 328)

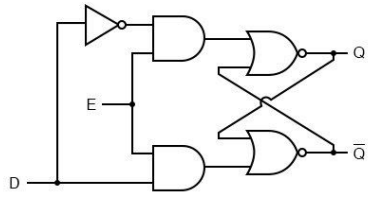
Table of Contents

1. Introduction

The objective of this laboratory experiment is to design and construct an Arithmetic and Logic Unit (ALU) in VHDL environment, which will be implemented three different problem sets. For the ALU, there are 5 components: 2 gated D Latches, 1 Moore Finite State Machine (FSM), 1 4-16 decoder, and 1 ALU core. In every problem set, all functions of the ALU will be designed and built using VHDL code and a final design Block diagram. Finally, the ALU for each problem set will be simulated, along with its components, using VHDL (based on functional simulation in Quartus Simulator.

2. Components: Latch1, Latch2

The Gated D Latch, also known as a D flip-flop, is a storage element that directly copies input to output, producing an output Q, and its complement, Q-complemented ($\overline{Q}$). Operating with a clock input, when the clock is set to “low-active”, the latch is not active and therefore remains in its previous state. When the clock is set to “high-active”, however, the latch will directly copy the input to output. For example, when clock=1 and D(data input)=0, Q(output)=0, and when clock=1 and D=1, Q=1. The two figures below represent the truth/characteristic table and the circuit diagram for the Gated D Latches. The only difference between the two is that for latch1, the input is variable A, and for latch2, the input is variable B.



E	D	Q	$\overline{Q}$
0	0	latch	latch
0	1	latch	latch
1	0	0	1
1	1	1	0

Table 1.1: The characteristic table and circuit diagram for a Gated D Latch. The E, or "Enable" input, works exactly as the clock input. The D, or Data input is the exact same as the A and B inputs in latch1 and latch2, respectively.

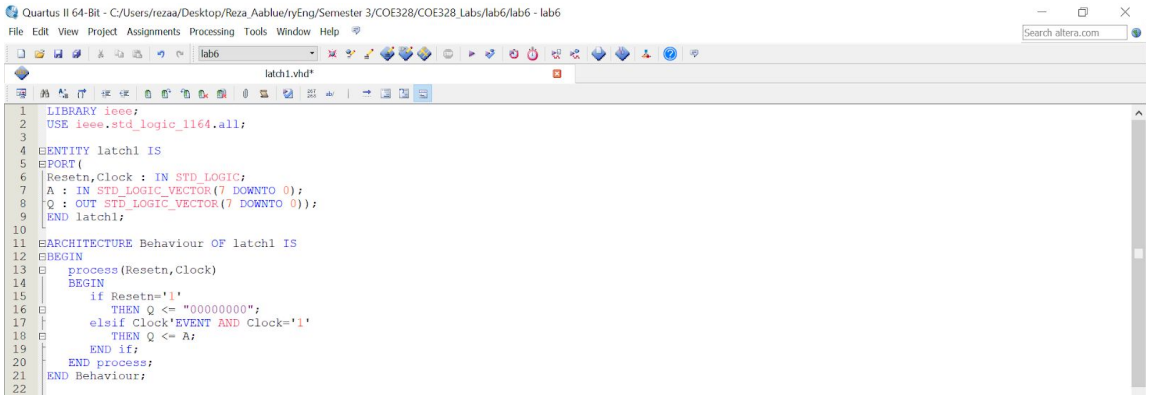


Figure 1.1: VHDL code for Gated D Latch #1, also known as latch1. Both latches have the exact same VHDL code and waveform, with the only difference being that variable B was used in latch2 instead of variable A.

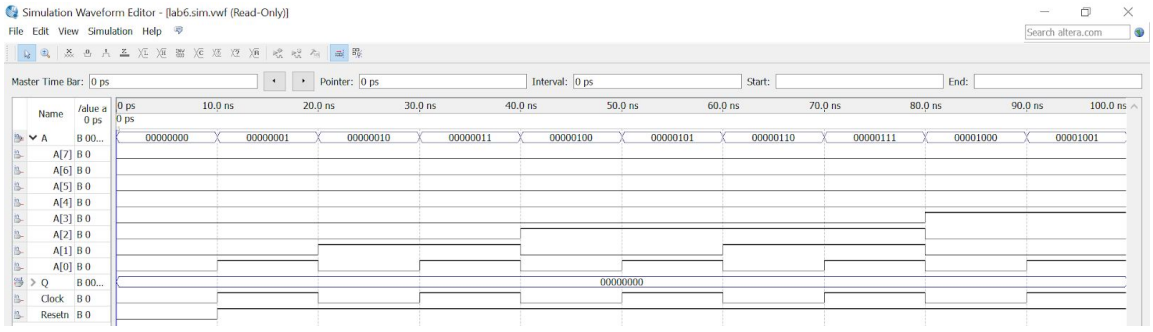


Figure 1.2: Read-only waveform for Gated D Latch #1, also known as latch1.

3. Components: Moore Finite State Machine (FSM)

The Moore Finite State Machine, also known as the Moore FSM, is a finite state machine where its output values are only determined by the current state. In the Moore FSM, there are inputs of Clock, data input and reset, and outputs of student number ID digits and the current state at each point where the "Clocking" happens during the waveform. This portion of the combined ALU schematic is responsible for determining the order of functions that the ALU will take and perform, as well as operating with the Clock to allow for a change of operations at every rising edge on the Clock.

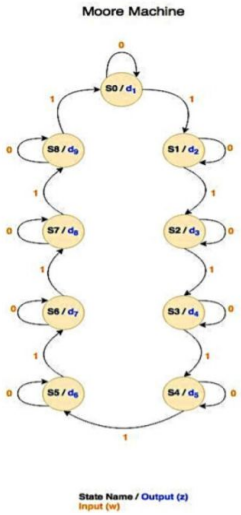
Table 1.2: State table for Moore FSM

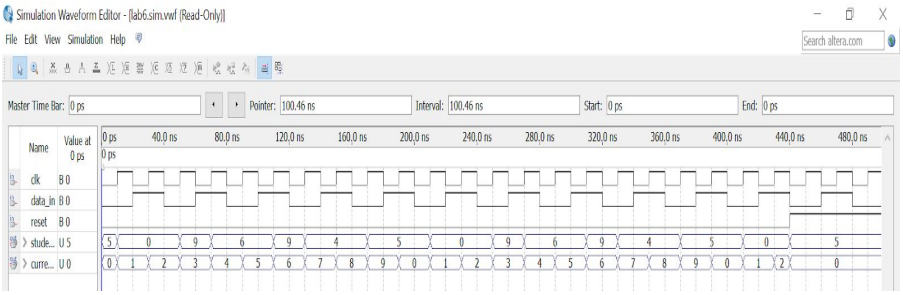
Current State	**Next State**		**Output (z)**
	w=0	**w=1**	
S0	S0	S1	d1=5
S1	S1	S2	d2=0
S2	S2	S3	d3=0
S3	S3	S4	d4=9
S4	S4	S5	d5=6
S5	S5	S6	d6=6
S6	S6	S7	d7=9
S7	S7	S8	d8=4
S8	S8	S9	d9=4
S9	S9	S0	d1=5

Table 1.3: State-assigned table for Moore FSM

Current State	Next State		Output (z)
	w=0	w=1	
0000	0000	0001	0101
0001	0001	0011	0000
0010	0011	0011	0000
0011	0011	0110	1001
0100	0110	0101	0110
0101	0101	0110	0110
0110	0110	0111	1001
0111	0111	1000	0100
1000	1000	1001	0100
1001	1001	0000	0101

Figures 1.3, 1.4: State Diagram and Read-only waveform for Moore FSM





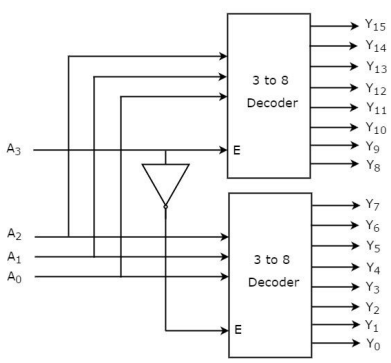
4. Components: 4-to-16 Decoder

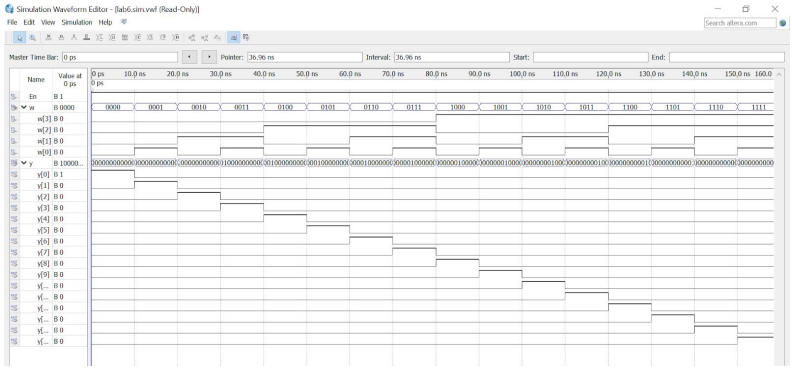
The 4-to-16 Decoder, also known as a 4:16 Decoder, is a circuit element which decodes 4-bit input to a 16-bit output with the help of an Enable input signal. When Enable is set to "low-active" or "0", the Decoder will not operate. However, when the Decoder is set to "high-active" or "1", the Decoder will operate and function, with the special case that for each valuation, only one bit can be set to "high-active" at a time. In the figures below, the specifications of a 4:16 Decoder will be further explained and demonstrated.

Table 1.4: Truth Table for a 4:16 Decoder

E	A	B	C	D	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11	Y12	Y13	Y14	Y15
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	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	1	1	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

Figures 1.5, 1.6: Circuit Diagram (using 2 3:8 Decoders) and Read-only waveform for the 4:16 Decoder





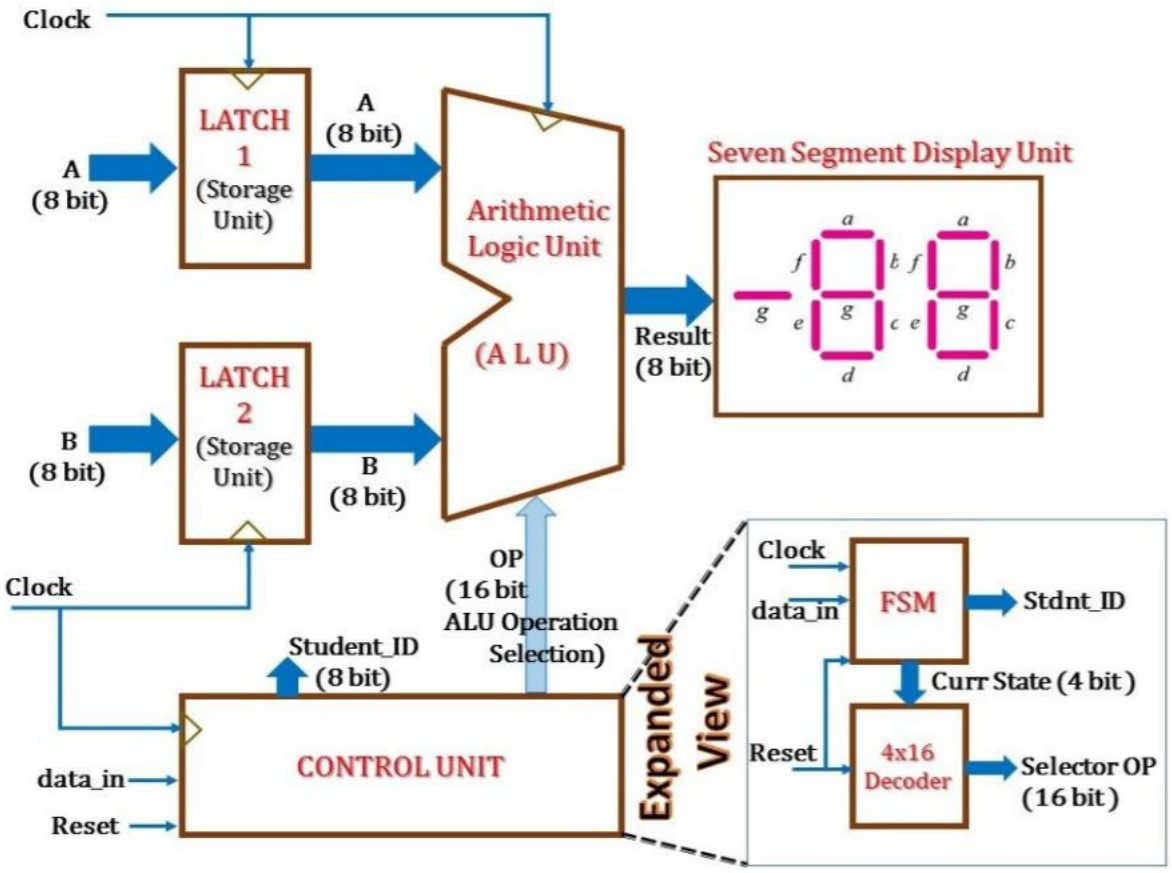
5. ALU_1 Problem Set 1 of the Lab 6 procedure

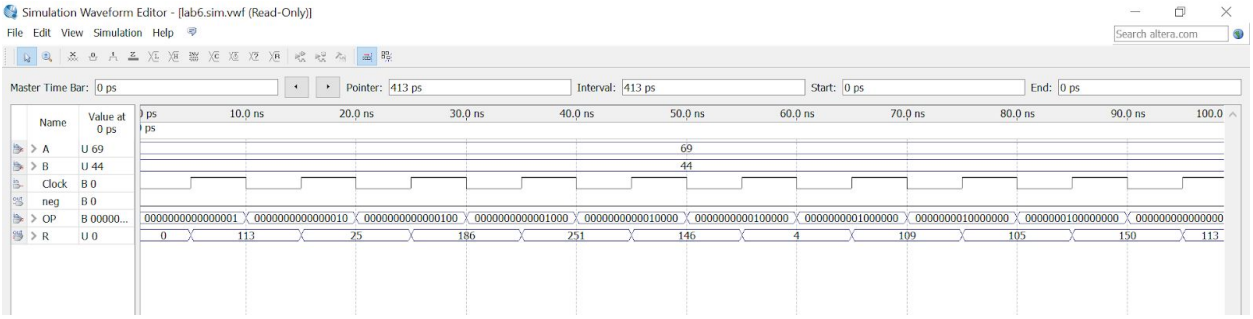
The first ALU, also known as ALU_1, is responsible for carrying out operations such as adding, subtracting, AND, OR, NAND, NOR, XOR, XNOR and the NOT logical operators. All of the 9 operators, or functions, are arranged in a particular order and all will occur one at a time with the rising edges of the Clock input. The ALU for problem 1 is synchronous in the sense that all blocks that require the Clock input are working together with the same Clock input. The ALU core for the schematic serves as an important block, which will allow for the change in operations so that all 9 functions are performed.

Table 1.5: Table of Microcodes for ALU_1 Problem Set 1

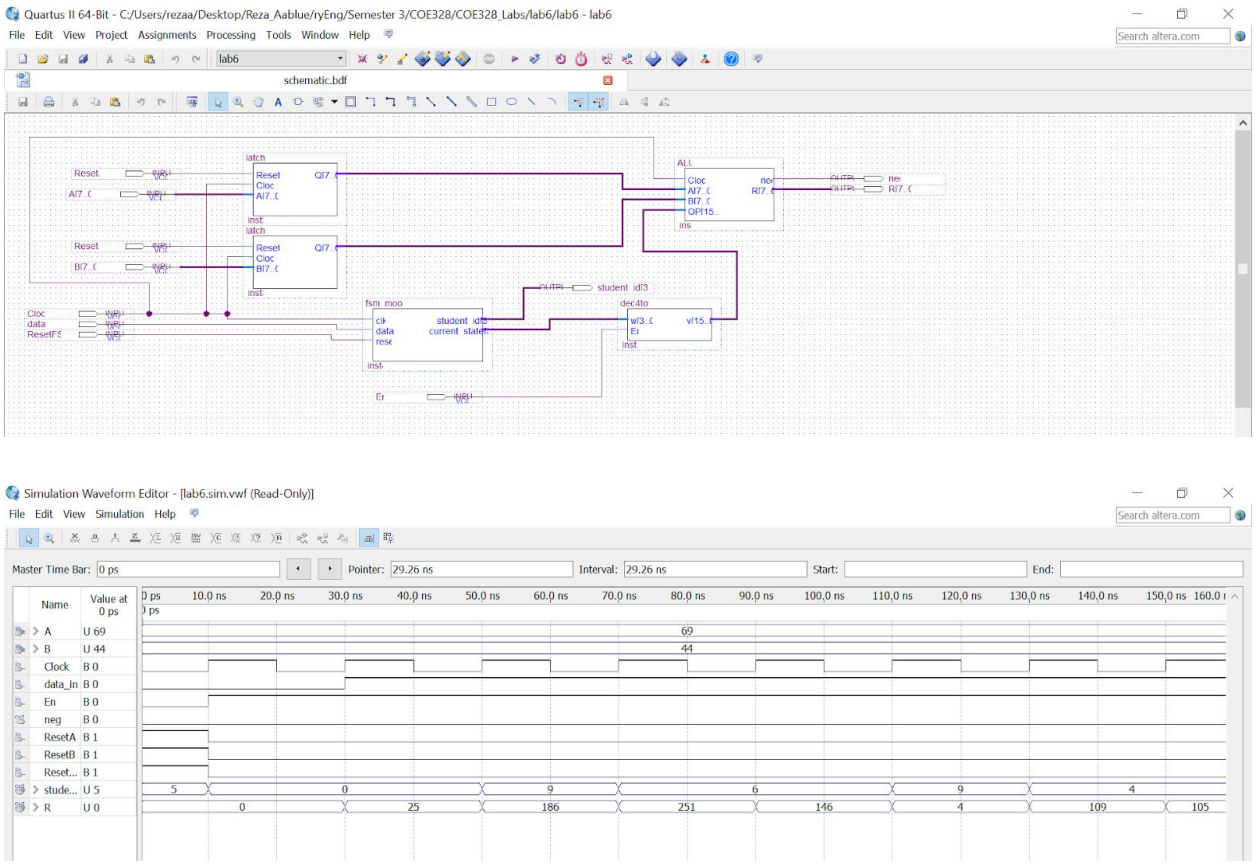
Function #	Microcode	Boolean Operation / Function
1	0000000000000001	sum(A, B)
2	0000000000000010	diff(A, B)
3	0000000000000100	$\overline{A}$
4	0000000000001000	$\overline{A \cdot B}$
5	0000000000010000	$\overline{A + B}$
6	0000000000100000	$A \cdot B$
7	0000000001000000	$A \oplus B$
8	0000000010000000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

Figures 1.7, 1.8: Circuit Diagram and Read-only waveform of ALU_1 Problem Set 1 and the ALU core





Figures 1.9, 1.10: Screenshot of the Block Schematic File (BDF) and the Read-only Waveform for the ALU_1 Problem Set 1



6. ALU_2 for Problem Set 2 of the Lab 6 procedure

The second ALU, also known as ALU_2, is responsible for carrying out operations like ALU_1, but now, there is a new list of operations to carry out. All of the 9 operators, or functions, are arranged in a particular order and all will occur one at a time with the rising edges of the Clock input. The ALU for problem 2 is also synchronous in the sense that all blocks that require the Clock input are working together with the same Clock input. The ALU core for the schematic serves as an important block, which will allow for the change in operations so that all 9 functions are performed.

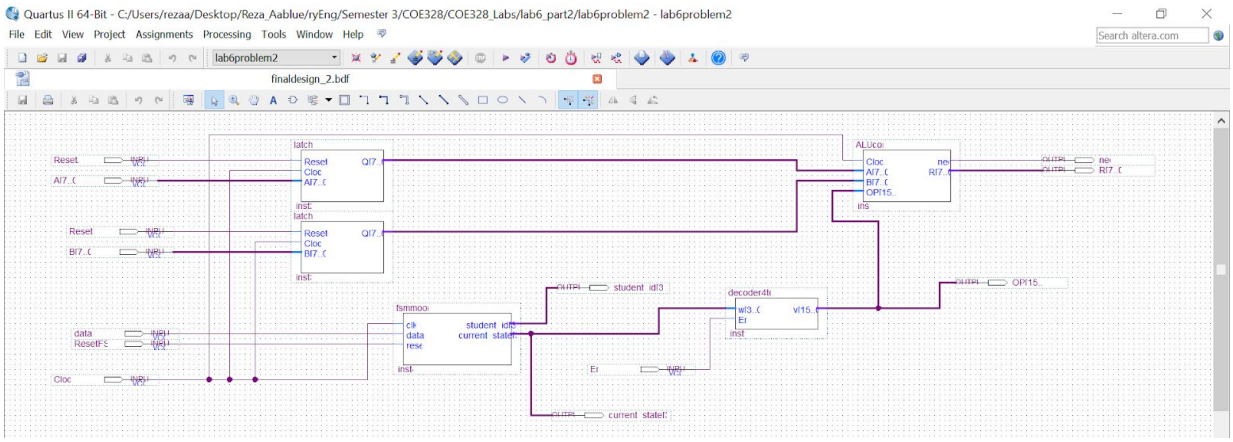
Name: Reza Aablue
Student ID: 500966944

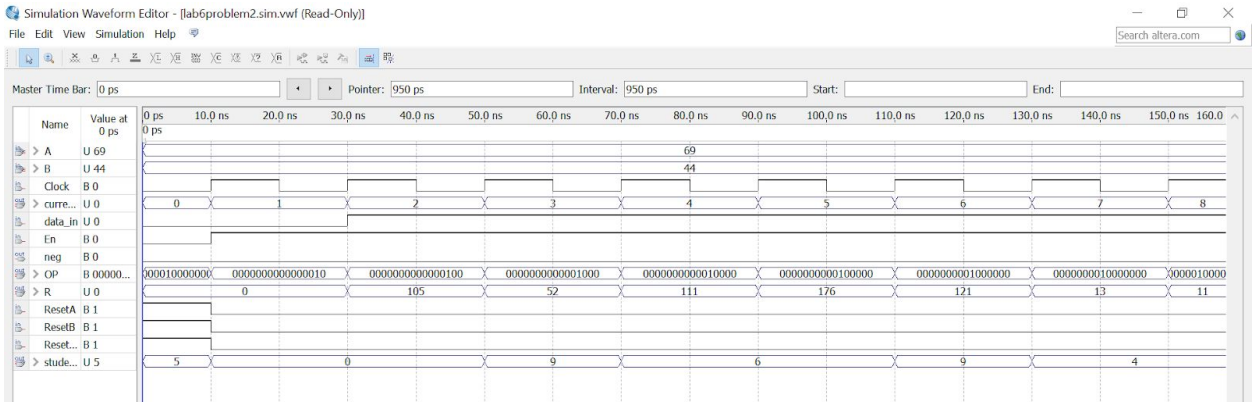
Lab 6: Design of a Simple General-Purpose Processor
Submission Deadline: August 09, 2020

Table 1.6, Figures 1.11, 1.12: Table of Microcodes for ALU_2 Problem Set 2, and Screenshot of the Block Schematic File (BDF), and the Read-only Waveform for the ALU_2 Problem Set 2

h)

Function #	Operation / Function
1	Rotate **A** to right by 4 bits (ROR)
2	Produce the result of XORing **A** and **B**
3	Invert the bit-significance order of **B**
4	Calculate the summation of **A** and **B** and decrease it by 2
5	Rotate **B** to left by 2 bits (ROL)
6	Invert the even bits of **B**
7	Swap the lower 4 bits of **B** with lower 4 bits of **A**
8	Shift **B** to right by 2 bits, input bit = 0 (SHR)
9	Invert lower four bits of **A**



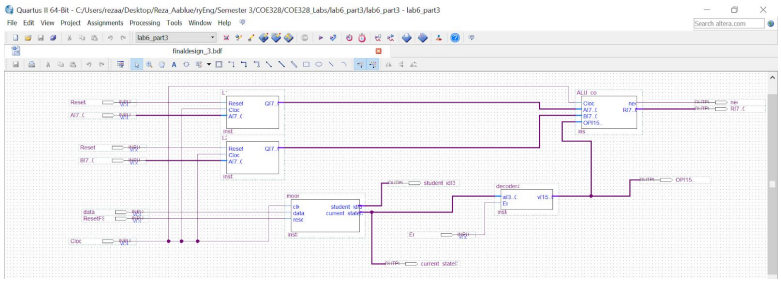


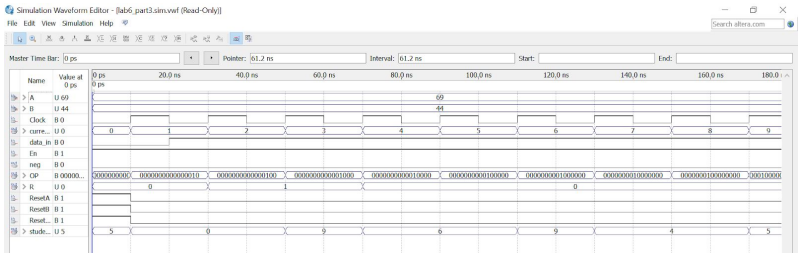
7. ALU_3 for Problem Set 3 of the Lab 6 procedure

The third ALU, also known as ALU_3, is responsible for carrying out just one operation, but for all 9 valuations. The purpose of ALU_3 is to check whether any of the 2 digits of number "B" or 44 are greater than the student ID output at each valuation. All of the 9 instances are arranged in a particular order and all will occur one at a time with the rising edges of the Clock input. The ALU for problem 3 is also synchronous in the sense that all blocks that require the Clock input are working together with the same Clock input. The ALU core for the schematic serves as an important block, which will allow for the change in operations so that all 9 instances are detected and analyzed.

Table 1.7, Figures 1.12, 1.13: Conditioning for ALU_3 Problem Set 3, and Screenshot of the Block Schematic File (BDF), and the Read-only Waveform for the ALU_3 Problem Set 3

h) For each microcode instruction, display 'y' if one of the 2 digits of B are greater than FSM output (**student_id**) and 'n' otherwise. Use the microcode instruction from part 1 of the lab.
i) For each microcode instruction, 'y' if one of the 2 digits of B are less than FSM output





8. Conclusion

In conclusion, this lab experiment proved to be quite lengthy, but quite interesting and challenging. The functions were successfully implemented using VHDL code and schematic design, despite the issues with the read-only waveforms. Therefore, the final conclusion about this experiment is that different types of ALUs can be implemented using VHDL and CAD design, and then to be successfully implemented on an FPGA Board.

9. Appendix

VHDL Code - Latch1,Latch2

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

ENTITY L1 IS
PORT(
Resetn,Clock : IN STD_LOGIC;
A : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END L1;

ARCHITECTURE Behaviour OF L1 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 Behaviour;
```

VHDL Code - Moore FSM

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

Entity moore IS
PORT (
clk, data_in, 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 moore IS
    type state_type IS (s0,s1,s2,s3,s4,s5,s6,s7,s8,s9);
    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 clk='0'
                THEN yfsm<=s0;
        else
                yfsm<=s1;
        END if;

WHEN s1=>
        if clk='0'
                THEN yfsm<=s1;
        else
                yfsm<=s2;
        END if;

WHEN s2=>
        if clk='0'
                THEN yfsm<=s2;
        else
                yfsm<=s3;
        END if;

WHEN s3=>
        if clk='0'
                THEN yfsm<=s3;
        else
                yfsm<=s4;
        END if;
```

```
WHEN s4=>
        if clk='0'
                THEN yfsm<=s4;
        else
                yfsm<=s5;
        END if;

WHEN s5=>
        if clk='0'
                THEN yfsm<=s5;
        else
                yfsm<=s6;
        END if;

WHEN s6=>
        if clk='0'
                THEN yfsm<=s6;
        else
                yfsm<=s7;
        END if;

WHEN s7=>
        if clk='0'
                THEN yfsm<=s7;
        else
                yfsm<=s8;
```

```
                    END if;


                WHEN s8=>
                    if clk='0'
                        THEN yfsm<=s8;
                    else
                        yfsm<=s9;
                    END if;


                WHEN s9=>
                    if clk='0'
                        THEN yfsm<=s9;
                    else
                        yfsm<=s0;
                    END if;
            END case;
        END if;
    END process;


    process(yfsm,data_in)
    BEGIN
        case yfsm is
            when s0=> student_id <="0101";


            when s1=> student_id <="0000";


            when s2=> student_id <="0000";
```

```
                when s3=> student_id <="1001";

                when s4=> student_id <="0110";

                when s5=> student_id <="0110";

                when s6=> student_id <="1001";

                when s7=> student_id <="0100";

                when s8=> student_id <="0100";

                when s9=> student_id <="0101";

END case;
END process;

process(yfsm)
BEGIN
        case yfsm is
                when s0=>
                        current_state<="0000";
                when s1=>
                        current_state<="0001";
                when s2=>
                        current_state<="0010";
```

```
                when s3=>
                        current_state<="0011";
                when s4=>
                        current_state<="0100";
                when s5=>
                        current_state<="0101";
                when s6=>
                        current_state<="0110";
                when s7=>
                        current_state<="0111";
                when s8=>
                        current_state<="1000";
                when s9=>
                        current_state<="1001";
        END case;
        END process;



END fsm;
```

VHDL Code - 4:16 Decoder

```
Library ieee;
Use ieee.std_logic_1164.all;


Entity decoder416 Is
        Port (w :In STD_LOGIC_VECTOR(3 DOWNTO 0);
                En: in Std_logic;
                y: Out std_LOGIC_VECTOR(15 DOWNTO 0));
```

```
        End decoder416;


Architecture Behavior of decoder416 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 others;

        End Behavior;
```

VHDL Code - ALU Core (Part 1)

```
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

USE IEEE.STD_LOGIC_UNSIGNED.ALL;

USE IEEE.NUMERIC_STD.ALL;


ENTITY ALU IS

        PORT( Clock : IN STD_LOGIC;

                        A, B : IN UNSIGNED(7 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(Clock, OP)
        BEGIN
            if(rising_edge(Clock))THEN
                CASE OP IS
                 WHEN "0000000000000001" =>
                 Result <= (Reg1 + Reg2);
                 neg <= '0';

                 WHEN "0000000000000010" =>
                 if (Reg1 > Reg2) then
                        neg <= '0';
                        Result <= (Reg1 - Reg2);

                 elsif (Reg1 < Reg2) then
                        Result <= ( NOT (Reg1 - Reg2)+1);
                        neg <= '1';
```

```
end if;

WHEN "0000000000000100" => -- inv
Result <= NOT (Reg1);
neg <= '0';

WHEN "0000000000001000" =>
Result <= (Reg1 NAND Reg2);
neg <= '0';

WHEN "0000000000010000" =>
Result <= (Reg1 NOR Reg2);
neg <= '0';

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

WHEN "0000000001000000" =>
Result <= (Reg1 OR Reg2);
neg <= '0';

WHEN "0000000010000000" =>
Result <= (Reg1 XOR Reg2);
neg <= '0';

WHEN "0000000100000000" =>
```

```
                    Result <= (Reg1 XNOR Reg2);
                    neg <= '0';

                    WHEN OTHERS =>
                    Result <= "00000000";

                END CASE;
            END IF;
        END PROCESS;
        R <= Result(7 DOWNTO 0);
END calculation;
```

VHDL Code - ALU Core (Part 2)

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

ENTITY ALUcore IS
    PORT( Clock : IN STD_LOGIC;
          A, B : IN UNSIGNED(7 DOWNTO 0);
          OP : IN UNSIGNED(15 DOWNTO 0);
          neg : OUT STD_LOGIC;
          R : OUT UNSIGNED(7 DOWNTO 0));
END ALUcore;

ARCHITECTURE calculation OF ALUcore IS
    SIGNAL Reg1, Reg2, Result : unsigned(7 downto 0):=(others =>'0');
```

```
        SIGNAL Reg4: UNSIGNED(0 TO 7);
        BEGIN
                Reg1 <= A;
                Reg2 <= B;
                Process(Clock, OP)
                BEGIN
                        if(rising_edge(Clock))THEN
                                CASE OP IS
                                 WHEN "0000000000000001" => --Rotate A to right by 4 bits (ROR) (DONE)
                                 Result <= Reg1 ror 4;
                                 neg <= '0';

                                 WHEN "0000000000000010" => --Produce the result of XORing A & B
(DONE)
                                 Result <= (Reg1 XOR Reg2);
                                 neg <= '0';

                                 WHEN "0000000000000100" => --Invert the bit-significance order of B
(DONE)
                                 Result <= (Reg2(0),Reg2(1),Reg2(2),
Reg2(3),Reg2(4),Reg2(5),Reg2(6),Reg2(7));
                                 neg <= '0';

                                 WHEN "0000000000001000" => --Calculate the summation of A & B &
decrease it by 2 (DONE)
                                 Result <= (Reg1 + Reg2) - "0010";
                                 neg <= '0';
```

```
                    WHEN "0000000000010000" => --Rotate B to left by 2 bits (ROL) (DONE)
                    Result <= Reg2(5 downto 0) & Reg2(7 downto 6);
                    neg <= '0';

                    WHEN "0000000000100000" => --Invert the even bits of B (DONE)
                    Result <= Reg2 xor "01010101";

                    WHEN "0000000001000000" => --Swap the lower 4 bits of B with lower 4 bits
of A (DONE)
        Result <= (Reg2 and "1100") or (Reg1 and "0101") ;
        neg <= '0';

                    WHEN "0000000010000000" => --Shift B to right by 2 bits, input bit = 0
(SHR) (DONE)
                    Result <= ("00" & Reg2(7 DOWNTO 2));
                    neg <= '0';

                    WHEN "0000000100000000" => --Invert lower four bits of A (DONE)
                    Result <= (NOT "00000101");
                    neg <= '0';

                    WHEN OTHERS =>
                    Result <= "00000000";

                   END CASE;
                END IF;
```

```
            END PROCESS;

            R <= Result(7 DOWNTO 0);

END calculation;
```

VHDL Code - ALU Core (Part 3)

```
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

USE IEEE.STD_LOGIC_UNSIGNED.ALL;

USE IEEE.NUMERIC_STD.ALL;


ENTITY ALU_core IS

      PORT( Clock : IN STD_LOGIC;

                  A, B : IN UNSIGNED(7 DOWNTO 0);

                  OP : IN UNSIGNED(15 DOWNTO 0);

                  neg : OUT STD_LOGIC;

                  R : OUT UNSIGNED(7 DOWNTO 0));

END ALU_core;


ARCHITECTURE calculation OF ALU_core IS

      SIGNAL Reg1, Reg2, Result : unsigned(7 downto 0):=(others =>'0');

      SIGNAL Reg4: UNSIGNED(0 TO 7);

      BEGIN

            Reg1 <= A;

            Reg2 <= B;


            Process(Clock, OP)

   BEGIN

      -- y = "00000001" and n = "00000000", both digits of B are 4. Number A is 69 and Number B is 44.
```

```
if(rising_edge(Clock))THEN
  CASE OP IS
   WHEN "0000000000000001" => -- d1: 5 (NO)
   Result <= "00000000";
   neg <= '0';
   WHEN "0000000000000010" => -- d2: 0 (YES)
   Result <= "00000001" ;
   neg <= '0';
   WHEN "0000000000000100" => -- d3: 0 (YES)
   Result <= "00000001" ;
   neg <= '0';
   WHEN "0000000000001000" => -- d4: 9 (NO)
   Result <= "00000000" ;
   neg <= '0';
   WHEN "0000000000010000" => -- d5: 6 (NO)
      Result <= "00000000";
   neg <= '0';
   WHEN "0000000000100000" => -- d6: 6 (NO)
      Result <= "00000000";
      neg <= '0';
   WHEN "0000000001000000" => -- d7: 9 (NO)
      Result <= "00000000";
   neg <= '0';
      WHEN "0000000010000000" => -- d8: 4 (NO)
   Result <= "00000000";
   neg <= '0';
   WHEN "0000000100000000" => -- d9: 4 (NO)
```

```
            Result <= "00000000";
                neg <= '0';
            WHEN OTHERS =>
            Result <= "00000000";
           END CASE;
                                    END if;
                END process;
        R <= Result(7 DOWNTO 0);
END calculation;
```