Building a General-Purpose Microprocessor

Table of Contents

Introduction and Background

Introduction: The main purpose of this lab was to design and construct an Arithmetic and Logic Unit in VHDL environment and implement it on an FPGA board. This was achieved by combining the three components that make up this circuit – D Latches that store information from two data inputs – A and B; a Control unit that comprises of two elements, the first being a Finite State Moore Machine that can change states based on its current state and the primary inputs, and outputs each digit of the Student ID that corresponds to each state, sequentially. The second element of the Control Unit is a 4-to-16 decoder, which uses the current state of the FSM as its 4-bit input, and outputs a 16-bit code, which is then used as one of the inputs for the final component of this Lab. The Arithmetic Logic Unit performs addition, subtraction, and logic functions. It is fed by the stored inputs A and B and depending on the unique 16-bit code supplied by the decoder, a function corresponding to that code is carried out on the two operands. All three components – The latches, the Control Unit and the ALU are controlled by a Clock. Once combined, compiled, and checked for errors using waveforms, this circuitry was modified to match the assigned problem sets. There were many concepts that will be talked about in the theory section, and the errors that were made and fixed eventually, will be explained in the discussion section. - 501034331

Components: <u>Gated D Latch – Latch1 & Latch2:</u>

A Gated D latch, which acts a superior SR latch, is a very useful storage element that requires only one data input 'D'. This D latch makes it impossible to create a troublesome situation where the 'S' and 'R' branches both have a signal of 1, and thus this avoids the outputs Q and $\bar{Q}$ from having unknown values as shown in Table 6.1. In this lab, these latches act as a temporary storage unit in which each data input A and B are stored, before getting clocked into the ALU for further usage. On the first positive edge of the clock, the Data input value is stored in the Latch if the reset is set to 1, otherwise the stored value is cleared to "00000000" and then on the next positive edge, it is transmitted to the ALU core.

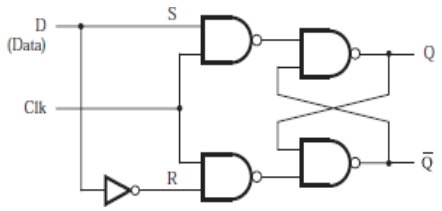


Figure 6.1a– Gated D Latch Circuit as seen in the Textbook

Table 6.1 – D Latch Truth Table

Clock	D	Q(t+1)
0	x	Q(t) = No Change
1	0	0
1	1	1

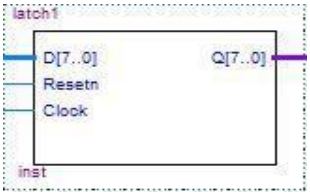


Figure 6.1b – Gated D Latch Block/Symbol

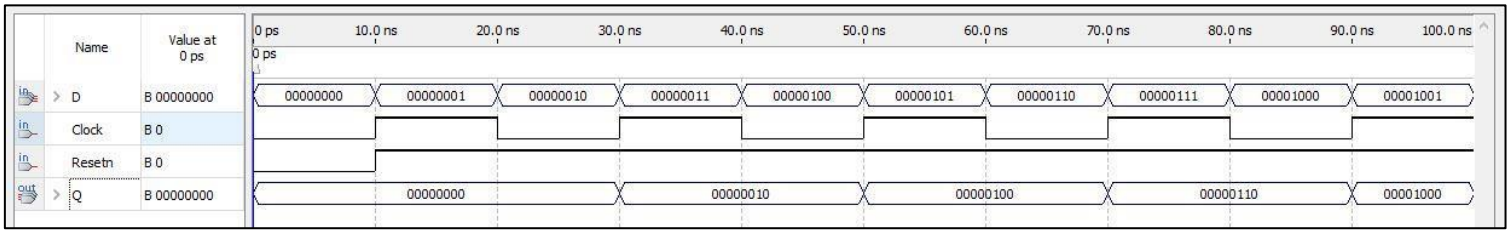


Figure 6.1c – Gated D Latch Waveform Simulation

As seen in Figure 6.i and 6.1c, the Reset pin is set to low-active meaning that the reset input must be 0 to reset the components and there were two latches – Latch1 and Latch2, that store A and B respectively, and both function identically (VHDL Code shown in Figure 6.i - Appendix)

Control Unit - Component 1:

4:16 Decoder:

A Decoder is used to decode encoded information. It takes in a small number of data inputs and outputs a larger, unique binary code that represents that particular valuation of data inputs inserted. A decoder takes in n inputs and produces 2^n outputs (as shown in Figure 6.2a) depending on the signal supplied by the Enable input – 1 allows the decoder to produce outputs, while a 0 signal stops the decoder from functioning. The task of the 4:16 decoder in this lab is to take in the 4-bit binary valuation of the current state of the FSM and output a corresponding 16-bit code that is "one-hot encoded" as shown in Table 6.2. This 16-bit code is aimed to act as a unique operation call within the ALU core (Figure 6.ii).

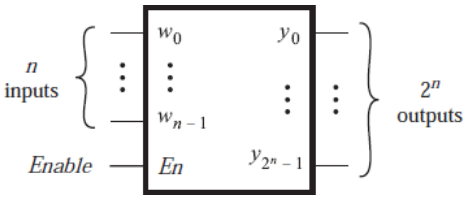


Figure 6.2a– A generic n-to-2n Decoder Block Schematic as seen in the Textbook

Table 6.2 – 4:16 Decoder Truth Table

En	w_3	w_2	w_1	w_0	y_{15}	y_{14}	y_{13}	y_{12}	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
1	0	0	0	0	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	0	**1**	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
1	0	0	1	1	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	**1**	0	0	0	0
1	0	1	0	1	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	**1**	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

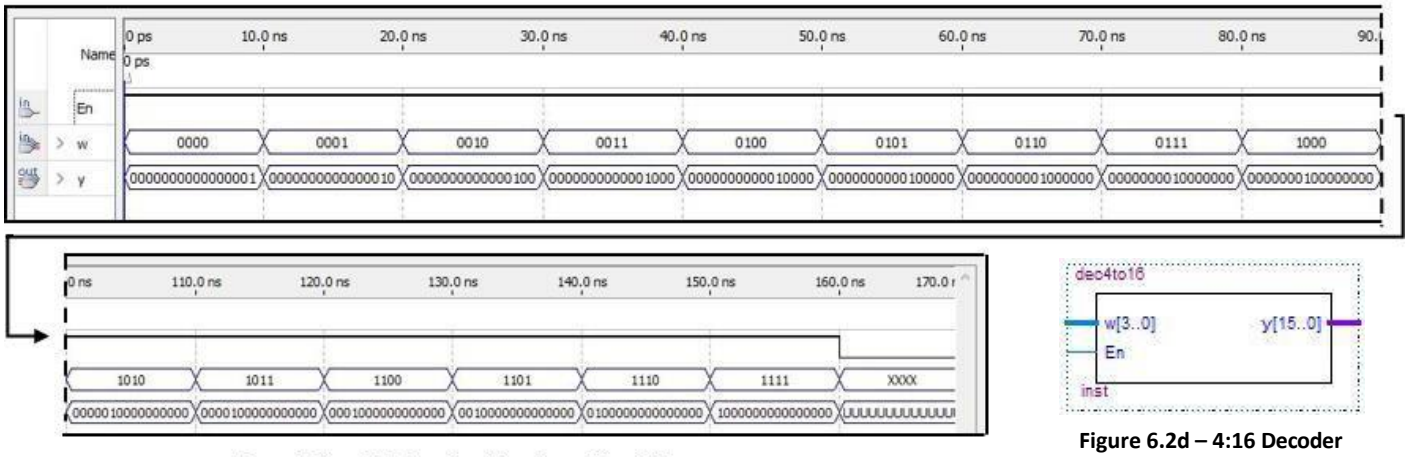


Figure 6.2b - 4:16 Decoder Waveform Simulation

Figure 6.2d – 4:16 Decoder Block/Symbol

Control Unit - Component 2:

Finite State Machine (Moore):

A Finite State Machine or sequential circuit is where the outputs depend on the past behavior, as well as on the present values of inputs. They consist of a combinational circuit and storage elements made of flip-flops. There are two types of edge-triggered FSMs both of which have their next state variables depending on the present state of the circuit and the primary outputs. However, in Moore machine, the primary outputs only depend on the current state, while in Mealy, they depend on the primary inputs as well. In this lab, a Moore machine was used to change states from s0 to s8 sequentially, and as they change states, each corresponding digit in the student ID is outputted, while the valuation of the current state is passed onto the decoder to be used as an indirect operation call.

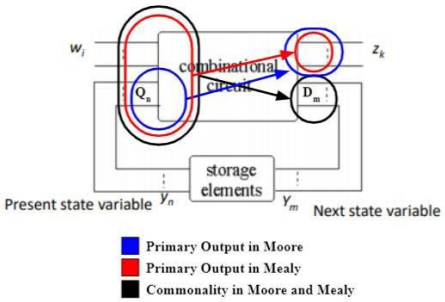


Figure 6.3a– FSM as seen in the PowerPoints

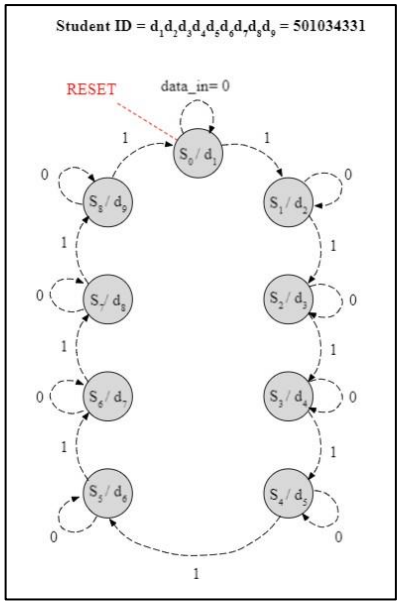


Figure 6.3b– Moore State Diagram

Table 6.3 – State-Assigned Characteristic Table for Moore

Current State	Next State		Output (z)
	w = 0	w = 1	
S0 = 0000	S0 = 0000	S1 = 0001	d1 = 5 = 0101
S1 = 0001	S1 = 0001	S2 = 0010	d2 = 0 = 0000
S2 = 0010	S2 = 0010	S3 = 0011	d3 = 1 = 0001
S3 = 0011	S3 = 0011	S4 = 0100	d4 = 0 = 0000
S4 = 0100	S4 = 0100	S5 = 0101	d5 = 3 =0011
S5 = 0101	S5 = 0101	S6 = 0110	d6 = 4 = 0100
S6 = 0110	S6 = 0110	S7 = 0111	d7 = 3 = 0011
S7 = 0111	S7 = 0111	S8 = 1000	d8 = 3 = 0011
S8 = 1000	S8 = 1000	S0 = 0000	d9 = 1 = 0001

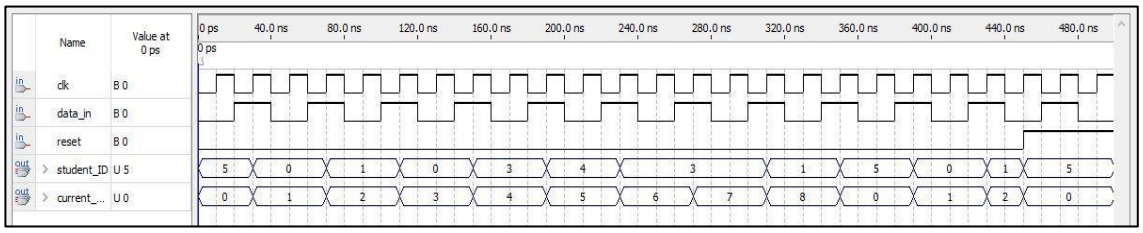


Figure 6.3d – FSM Waveform Simulation

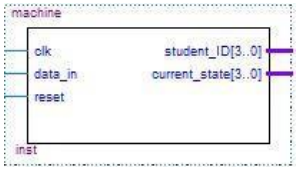


Figure 6.3e – FSM Block/Symbol

The VHDL code for the FSM is shown in Figure 6.iii in the Appendix. Figure 6.3d shows the waveform of the FSM and Figure 6.2e shows the block diagram. The student ID is an independent output while the current state output is then fed into the 4:16 decoder to determine the operation that needs to be used in the ALU.

ALU – Problem 1: The Arithmetic Logic Unit is used to carry out basic arithmetic and logic functions. It must have a set of inputs and a signal input that the ALU recognizes as a unique call for a unique function with the two operands being the data inputs in this case. The values of A and B stored in Latches 1 and 2 respectively, are inputted into the ALU, while the 16-bit code which was initially decoded from the current state of the FSM, is used as the Operation Call. In this Lab, the ALU was supposed to follow the functions according to the microcode from Table 1 in the Lab Manual, or Table 6.4 from this report. However, the inputs are only clocked at the positive edge of the clock. Once the operations are carried out, depending on the type of function, the outputs are produced and the signal for the negative sign is outputted if possible – 0 meaning no negative sign, and 1 meaning the existence of a negative sign.

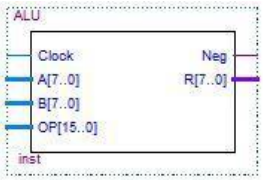


Figure 6.4a – ALU Block/Symbol

Table 6.4 – Microcode for ALU_1

Function #	Microcode	Boolean Operation / Function
1	0000000000000001	$sum(A,B)$
2	0000000000000010	$diff(A,B)$
3	0000000000000100	$\bar{A}$
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	$\overline{A \oplus B}$

Function #1 is a regular addition function where A and B are added together, similar to an ASU. For Function #2, the difference between A and B is counted by doing the operation A-B. If A > B, then the operation A-B is done and the negative signal = 0. If A < B, the operation B-A is done and the negative signal = 1. For Functions 3-9, each bit of both inputs is applied to the logic operation the symbols represent. For example, '⊕' means XOR, and so, $A \oplus B$ = A XOR B. The VHDL Code for ALU_1 is shown in Figure 6.iv in the Appendix, the block diagram is shown in Figure 6.4c and the waveform in Figure 6.4d.

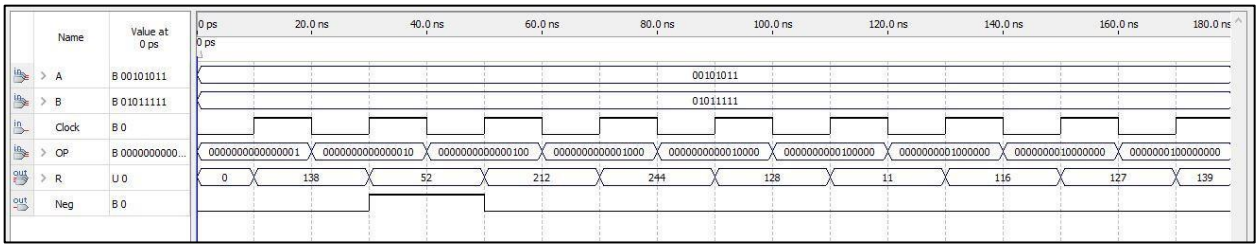


Figure 6.4b – ALU 1 Core Waveform Simulation with inputs being 43, and 95

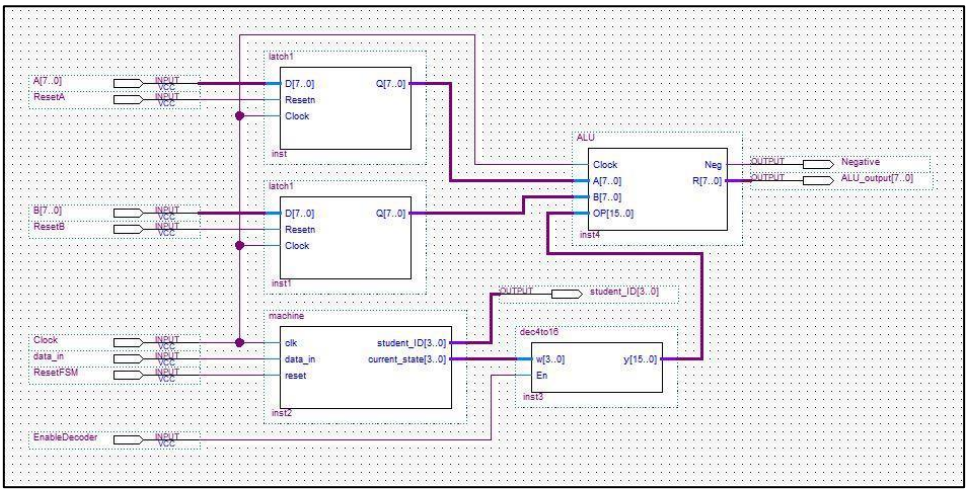


Figure 6.4c – Processor Problem 1 Block Schematic Diagram

Figure 6.4d – Final Problem 1 - ALU_1 Combined Waveform Simulation with inputs being 43 and 31

ALU – Problem 2: This Arithmetic Logic Unit had the same inputs and outputs as that of Problem 1, thus it also has the same symbol. The formation of the ALU in Problem 2 was based on the assigned problem sets given to each person. In each problem set, the arithmetic and logic operations correspond to a given set of microcodes and so, instead of using the default operations in Problem 1, these were customized for each person. The Problem set assigned was '**E**' and so, the functions that are called for the same microcodes as in Problem 1, were different. According to the problem set, the microcodes for ALU_2 is shown in Table 6.5.

Table 6.5 – Microcode for ALU_2

Function #	Microcode	Operation / Function
1	0000000000000001	Replace the odd bits of A with odd bits of B
2	0000000000000010	Produce the result of NANDing A and B
3	0000000000000100	Calculate the summation of A and B and decrease it by 5
4	0000000000001000	Produce the 2's complement of B
5	0000000000010000	Invert the even bits of B
6	0000000000100000	Shift A to left by 2 bits, input bit = 1 (SHL)
7	0000000001000000	Produce null on the output
8	0000000010000000	Produce 2's complement of A
9	0000000100000000	Rotate B to right by 2 bits (ROR)

In Function #1, the replacement of odd bits of A with the odd bits of B can be done by first establishing that the final output that the user should see is the new modified A. As shown in Figure 6.v, the final output Result should show the new version of A. Now that this has been established, all the existing even bits of A are assigned to even bits of Result. Then, all odd bits of B are assigned to the odd bits of Result. This is then deemed the final version of A, with its even bits staying as even bits of A and its odd bits becoming that of B. Function #2 is similar to Function #4 of Problem 1. Function #3 is similar to Function #1 of Problem 1, except that in the end, 5 is deducted from the sum. Function #4 is where the two's complement is found for B. In Function #5, the off bits of B are assigned to Result, while the even bits of B are inverted and then assigned to Result. For Function#6, a special function is used to shift A by 2 bits. Here, the places where the numbers are shifted from as supposed to have anew value of 0. In Function #7, all bits of Result are equal to 0. Function #8 is similar to Function 4 except that A is used. In Function #9, a special function is used to rotate values of B by 2 bits to the right. In this function, when shifted to the right, the rightmost bits are getting carried onto the left side. The waveform in Figure 6.5a shows the functionality of ALU individually, for Problem 2. The values that were used to confirm the waveform was the same inputs the Lab instructor used for the purpose of accuracy.

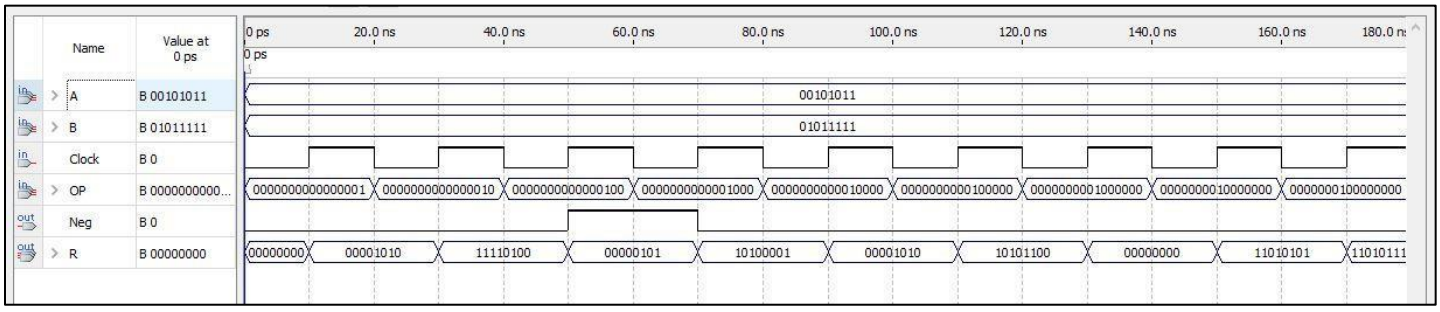


Figure 6.5a – ALU 2 Core Waveform Simulation with inputs 43 and 95

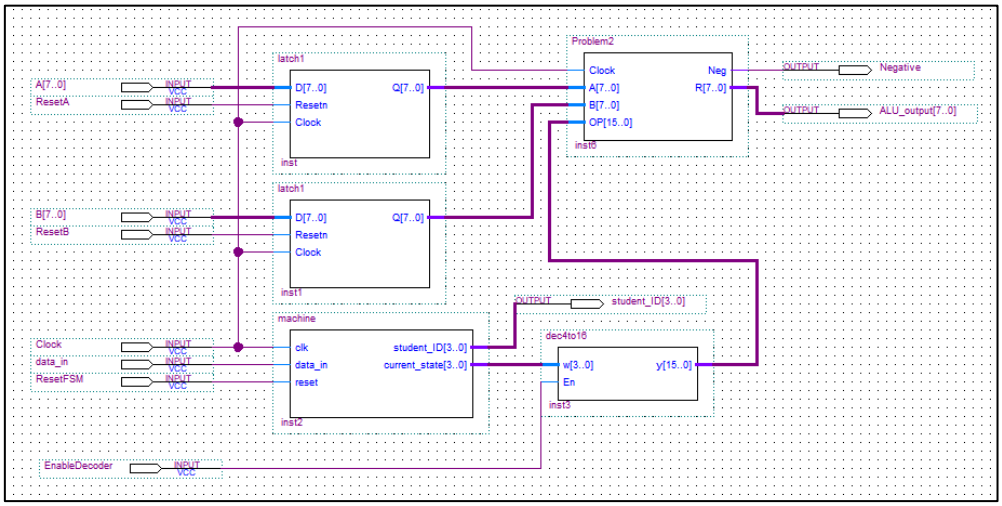


Figure 6.5b – Processor Problem 2 Block Schematic Diagram

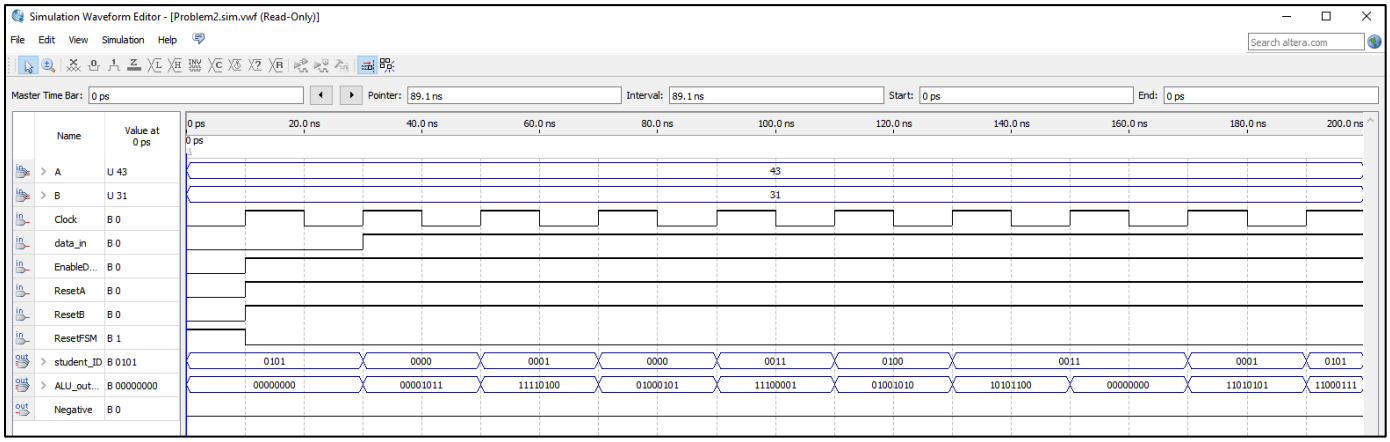


Figure 6.5c – Final Problem 2 - ALU_2 Combined Waveform Simulation with inputs 43 and 31

ALU – Problem 3: The Arithmetic Logic Unit created in this part of the lab was required to carry out a specific function that was assigned. The problem assigned was '**E**' and so, the task was to display a 'y' if one of the 2 digits of the A input is greater than FSM output (student_ID) and 'n' otherwise. Since a display was not used, specific logic values were assigned to 'y' and 'n'. Therefore, it is assumed that a logic value of 1 represents 'y' and 0 represents 'n'. In this section of the lab it is important to note that the input value for A must be entered in Hexadecimal and not its Binary equivalent like in Problems 1 and 2. This is because, when two-digit inputs are entered, it is easier to represent each digit as its own 4-bit hexadecimal number than doing BCD correction. Thus for example, if 43 is used, its Hexadecimal equivalent is 0100 0011 and not $00101011_{(2)}$. Thus, when these values are entered, each digit is recognized by splicing the 8-bit binary number into two 4-bit numbers as shown in Figure 6.vi. Once this is established, the comparing of values is the next step.

The “If-else” statements compare the bigger digit in the A input with the student number for each case being the output of the decoder. In each case, the condition for the student ID is a one-hot encoded representation of the student number that it represents earlier from when the decoder converted the 4-bit valuation into a 16-bit code. Thus, the binary representation of the digit of the student number at that state is compared to the larger digit of A, which then determines if that digit is larger or smaller. If it is smaller, it results to the YN value being 0, and 1 otherwise.

Table 6.6 – Microcode for ALU_3

Function #	Microcode	Operation / Function		
		Reg2 = A(7 DOWNTO 4)	Reg1 = A(3 DOWNTO 0)	Otherwise
1	0000000000000001	If Reg1>(d1 = 5), Y=1	If Reg2>(d1 = 5), Y=1	Y = 0
2	0000000000000010	If Reg1>(d2 = 0), Y=1	If Reg2>(d2 = 0), Y=1	Y = 0
3	0000000000000100	If Reg1>(d3 = 1), Y=1	If Reg2>(d3 = 1), Y=1	Y = 0
4	0000000000001000	If Reg1>(d4 = 0), Y=1	If Reg2>(d4 = 0), Y=1	Y = 0
5	0000000000010000	If Reg1>(d5 = 3), Y=1	If Reg2>(d5 = 3), Y=1	Y = 0
6	0000000000100000	If Reg1>(d6 = 4), Y=1	If Reg2>(d6 = 4), Y=1	Y = 0
7	0000000001000000	If Reg1>(d7 = 3), Y=1	If Reg2>(d7 = 3), Y=1	Y = 0
8	0000000010000000	If Reg1>(d8 = 3), Y=1	If Reg2>(d8 = 3), Y=1	Y = 0
9	0000000100000000	If Reg1>(d9 = 1), Y=1	If Reg2>(d9 = 1), Y=1	Y = 0

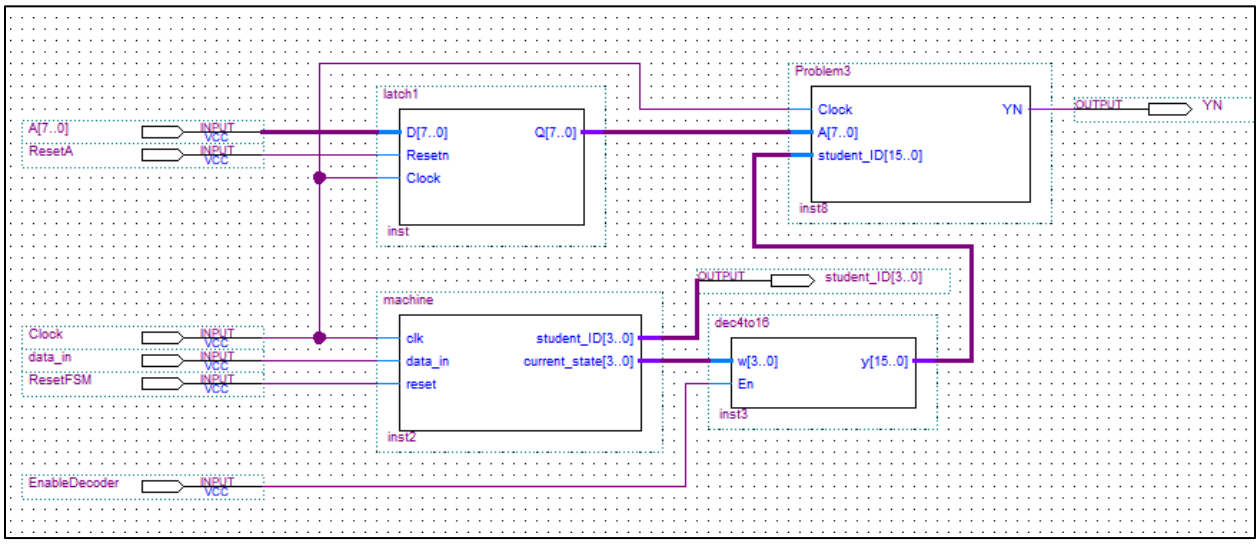


Figure 6.6a – Processor Problem 3 Block Schematic Diagram

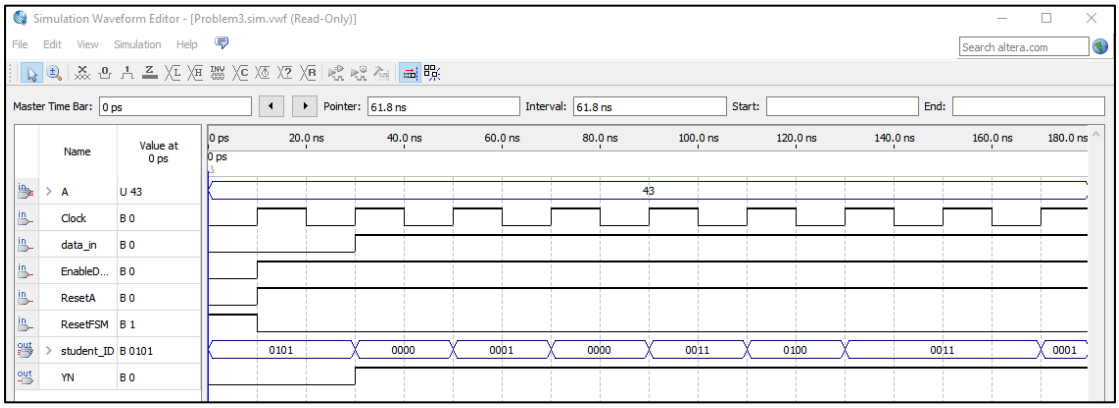


Figure 6.6b - Final ALU_3 Waveform Simulation

Discussion and Conclusion

Flaws and Troubleshooting:

All sections in this lab were conducted with maximum precision in mind. They were checked and tested multiple times before the lab was finished, but on the way, these are the few errors and flaws that were encountered:

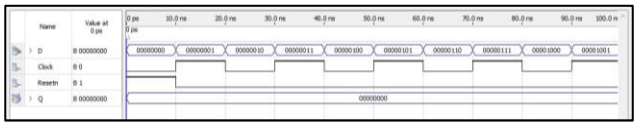


Figure 6.a – Error #1

1. **Usage of Reset input incorrectly:**
   The VHDL code in Figure 6.i for the latches that the Q value would be set to "0000000" if the reset signal is set to 0, regardless of the clock signal. However, it wasn't realized for a decent portion of this lab that when following the waveform shown in the lab session to check for correctness, the reset value that the instructor used to reset the circuit was 1, which caused many failures in compilation. However, the code used here required the reset value to be 0 to clear the Q.
2. **Creation of multiple files within one project:**
   Although it was pointed out in the lab manual that different projects should be made to simulate waveforms for separate components, it became necessary to make multiple waveform files to debug some of the codes. However, this was detrimental as some files became corrupted on Quartus. The occurrences of these software errors became frequent, which caused more failures like outputting the same values for the new set of inputs as the previous set of inputs. This was eventually fixed with a fresh new project file.
3. **Assigning values to a variable:**
   It was realized in the lab demonstration that when new variables for certain inputs in the customized problem were used, they stopped the code from performing its function. For example, for Problem 2 – Function 3, the A and B values were added and then assigned to a new variable that was made. Then the 5 was subtracted. However, due to some inconsistencies in Quartus, the new variable was set to 0 and thus, the final output became -5 instead of A+B -5. This was solved by reverting back to and modifying the given templates.
4. **Syntax:**
   It is important to notice the minor differences in format when copying a template from the textbook. These minor differences can make a code uncompilable and so they must be addressed. Many of these components went through constant debugging due to this error.

Discussion:

Through this lab, a Simple General Microprocessor was built using the several concepts learned in the course. However, using the data and waveforms shown, the functionality of the microprocessor can be better understood. As seen in Figures 6.4c and 6.5b, the Latches temporarily store each input, which in this case were A and B. Simultaneously, the data_in input controls and changes the current state of the FSM directly. Each state corresponds to a certain digit in the student number, which is outputted independently. Each state of the circuit has a specific valuation assigned to it. Every clock cycle, the valuation is passed into the decoder to be converted into a unique 16-bit code. This 16-bit code serves as an operation call in the ALU. Using this operation call, the function is performed on the two operands A and B, initially stored in the Latches. Each one of these components is controlled by a clock signal and changes occur at the positive edge along with their own respective reset signals acting as secondary controls.

Qualitative analysis shows that the waveforms for the Latches, FSM and the decoder for Parts A and B are identical to the waveform shown in the Lab session. This means that these parts were carried out successfully. However, the combined waveforms for Problems 1 through 3 must be carefully analyzed. The waveform in Figure 6.4d for Problem 1 uses 43 and 31 as its inputs and outputs the correct results if the values are manually calculated and checked. It can be seen that the digits of the student number correspond to their particular states along with the operation for each state of the circuit. This can be checked with Table 6.4.

For Problem 2, according to the Microcode instructions shown in Table 6.6 and as explained previously, the operations were formed under each case of unique operation calls. As mentioned in the errors, the reset signal was inverted from before for the circuit to function correctly as planned. Considering the special functions such as shifting of bits or rotating, are all utilized. Overall, if checked manually, it can be said that this part of the lab was also successful.

For Problem 3, the waveform shown in Figure 6.6b works flawlessly as the function of that ALU was to check if any one of the digits in A is greater than the student number at that time and make y = 1 if yes and 0 otherwise. It can be seen that wherever the student number reaches a number higher than 4, which is 5, the y = 0, and y = 1 otherwise. This is shown throughout the waveform as y = 0 only when the student number is 5.

Conclusion:

This lab successfully helped realize a fully functioning General-Purpose Processor by designing and constructing an Arithmetic and Logic Unit (ALU) in a VHDL environment. The wide scope of all the operations that an Arithmetic Logic Unit uses, were introduced, and used according to specifications. In addition, major components learned throughout the duration of this course – Storage elements such as Latches (Chapter 7), A Finite State machine (Chapter 8), A decoder (Chapter 6), and the ALU (Including the ASU and entire basic foundation of logic circuits) – were all utilized. Although there were quite a few errors, they were handled and fixed eventually. Many of the concepts were cleared through this lab and it allowed the cumulation of almost all types of VHDL codes used in this course with basic but crucial functions. Customized functions and problem sets raised the difficulty of this lab by challenging the way logic circuits must be handled. Overall, this lab was a success as it completely fulfilled its main purpose.

Appendix

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

ENTITY latch1 IS
   PORT (D              :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 <= D;
      END IF;
   END PROCESS;
END Behavior;
```

Figure 6.i– Gated D Latch VHDL code

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

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

ARCHITECTURE Behavior OF dec4to16 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;
```

Figure 6.ii– 4:16 Decoder VHDL code

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

ENTITY machine IS
   PORT (clk            :IN STD_LOGIC;
         data_in        :IN STD_LOGIC;
         reset          :IN STD_LOGIC;
         student_ID     :OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
         current_state  :OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END machine;

ARCHITECTURE fsm OF machine 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
            yfsm <= s0;
         ELSIF (clk'EVENT AND clk = '1') THEN
            CASE yfsm IS
               WHEN s0 =>
                  IF data_in = '0' THEN
                     yfsm <= s0;
                  ELSE
                     yfsm <= s1;
                  END IF;
               WHEN s1 =>
                  IF data_in = '0' THEN
                     yfsm <= s1;
                  ELSE
                     yfsm <= s2;
                  END IF;
               WHEN s2 =>
                  IF data_in = '0' THEN
                     yfsm <= s2;
                  ELSE
                     yfsm <= s3;
                  END IF;
               WHEN s3 =>
                  IF data_in = '0' THEN
                     yfsm <= s3;
                  ELSE
                     yfsm <= s4;
                  END IF;
               WHEN s4 =>
                  IF data_in = '0' THEN
                     yfsm <= s4;
                  ELSE
                     yfsm <= s5;
                  END IF;
               WHEN s5 =>
                  IF data_in = '0' THEN
                     yfsm <= s5;
                  ELSE
                     yfsm <= s6;
                  END IF;
               WHEN s6 =>
                  IF data_in = '0' THEN
                     yfsm <= s6;
                  ELSE
                     yfsm <= s7;
                  END IF;
               WHEN s7 =>
                  IF data_in = '0' THEN
                     yfsm <= s7;
                  ELSE
                     yfsm <= s8;
                  END IF;
               WHEN s8 =>
                  IF data_in = '0' THEN
                     yfsm <= s8;
                  ELSE
                     yfsm <= s0;
                  END IF;
            END CASE;
         END IF;
      END PROCESS;

      PROCESS(yfsm, data_in)
      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 <= "0001";
            WHEN s3 =>
               current_state <= "0011";
               student_ID <= "0000";
            WHEN s4 =>
               current_state <= "0100";
               student_ID <= "0011";
            WHEN s5 =>
               current_state <= "0101";
               student_ID <= "0100";
            WHEN s6 =>
               current_state <= "0110";
               student_ID <= "0011";
            WHEN s7 =>
               current_state <= "0111";
               student_ID <= "0011";
            WHEN s8 =>
               current_state <= "1000";
               student_ID <= "0001";
         END CASE;
      END PROCESS;
END fsm;
```

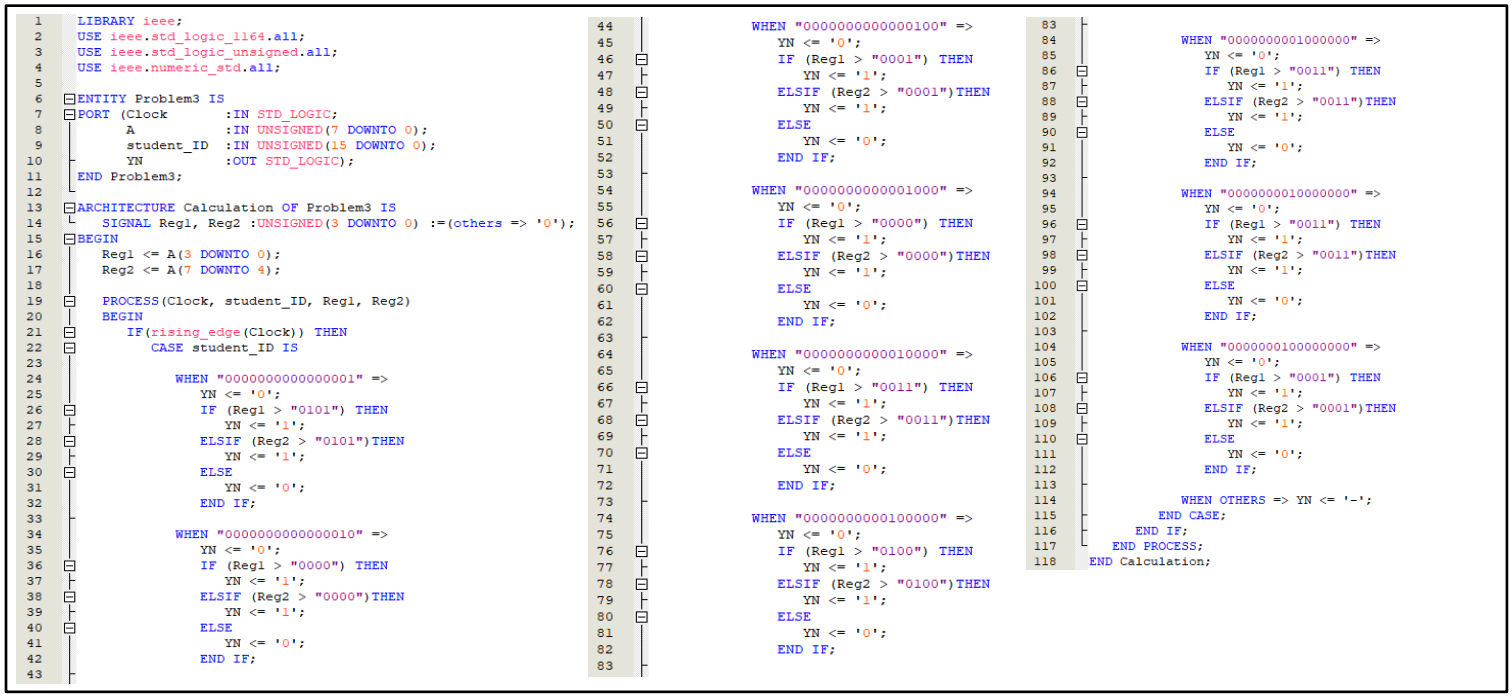
Figure 6.iii– FSM VHDL Code

```vhdl
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');
BEGIN
   Reg1 <= A;
   Reg2 <= B;
   PROCESS(Clock, OP)
   BEGIN
      IF(rising_edge(Clock)) THEN
         CASE OP IS
            WHEN "0000000000000001" =>
               Neg <= '0';
               Result <= (Reg1 + Reg2);

            WHEN "0000000000000010" =>
               IF Reg1 > Reg2 THEN
                  Result <= (Reg1 - Reg2);
                  Neg <= '0';
               ELSE
                  Result <= (Reg2 - Reg1);
                  Neg <= '1';
               END IF;

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

            WHEN "0000000000001000" =>
               Neg <= '0';
               Result <= NOT (Reg1 AND Reg2);
            WHEN "0000000000010000" =>
               Neg <= '0';
               Result <= NOT (Reg1 OR Reg2);

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

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

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

            WHEN "0000000100000000" =>
               Neg <= '0';
               Result <= NOT (Reg1 XOR Reg2);

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

         END CASE;
      END IF;
   END PROCESS;

   R <= Result(7 DOWNTO 0);

END Calculation;
```

Figure 6.iv– ALU Problem 1 VHDL Code

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

ENTITY Problem2 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 Problem2;

ARCHITECTURE Calculation OF Problem2 IS
   SIGNAL Reg1, Reg2, Result  :UNSIGNED(7 DOWNTO 0) :=(others => '0');
BEGIN
   Reg1 <= A;
   Reg2 <= B;
   PROCESS(Clock, OP)
   BEGIN
      IF(rising_edge(Clock)) THEN
         CASE OP IS
            WHEN "0000000000000001" =>
               Neg <= '0';
               Result(7) <= Reg2(7);
               Result(6) <= Reg1(6);
               Result(5) <= Reg2(5);
               Result(4) <= Reg1(4);
               Result(3) <= Reg2(3);
               Result(2) <= Reg1(2);
               Result(1) <= Reg2(1);
               Result(0) <= Reg1(0);

            WHEN "0000000000000010" =>
               Neg <= '0';
               Result <= (Reg1 NAND Reg2);

            WHEN "0000000000000100" =>
               Result <= (Reg1 + Reg2) - 5;

            WHEN "0000000000001000" =>
               Result <= 0 - Reg2;
               Neg <= Result (7);

            WHEN "0000000000010000" =>
               Neg <= '0';
               Result(7) <= Reg2(7);
               Result(6) <= NOT Reg2(6);
               Result(5) <= Reg2(5);
               Result(4) <= NOT Reg2(4);
               Result(3) <= Reg2(3);
               Result(2) <= NOT Reg2(2);
               Result(1) <= Reg2(1);
               Result(0) <= NOT Reg2(0);

            WHEN "0000000000100000" =>
               Neg <= '0';
               Result <= (Reg1 SLL 2);

            WHEN "0000000001000000" =>
               Neg <= '0';
               Result <= "00000000";

            WHEN "0000000010000000" =>
               Result <= 0 - Reg1;
               Neg <= Result (7);

            WHEN "0000000100000000" =>
               Neg <= '0';
               Result <= Reg2 ROR 2;

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

         END CASE;
      END IF;
   END PROCESS;

   R <= Result(7 DOWNTO 0);

END Calculation;
```

Figure 6.v– ALU Problem 2 VHDL Code



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

ENTITY Problem3 IS
PORT (Clock       :IN STD_LOGIC;
      A           :IN UNSIGNED(7 DOWNTO 0);
      student_ID  :IN UNSIGNED(15 DOWNTO 0);
      YN          :OUT STD_LOGIC);
END Problem3;

ARCHITECTURE Calculation OF Problem3 IS
   SIGNAL Reg1, Reg2 :UNSIGNED(3 DOWNTO 0) :=(others => '0');
BEGIN
   Reg1 <= A(3 DOWNTO 0);
   Reg2 <= A(7 DOWNTO 4);

   PROCESS(Clock, student_ID, Reg1, Reg2)
   BEGIN
      IF(rising_edge(Clock)) THEN
         CASE student_ID IS

            WHEN "0000000000000001" =>
               YN <= '0';
               IF (Reg1 > "0101") THEN
                  YN <= '1';
               ELSIF (Reg2 > "0101")THEN
                  YN <= '1';
               ELSE
                  YN <= '0';
               END IF;

            WHEN "0000000000000010" =>
               YN <= '0';
               IF (Reg1 > "0000") THEN
                  YN <= '1';
               ELSIF (Reg2 > "0000")THEN
                  YN <= '1';
               ELSE
                  YN <= '0';
               END IF;

            WHEN "0000000000000100" =>
               YN <= '0';
               IF (Reg1 > "0001") THEN
                  YN <= '1';
               ELSIF (Reg2 > "0001")THEN
                  YN <= '1';
               ELSE
                  YN <= '0';
               END IF;

            WHEN "0000000000001000" =>
               YN <= '0';
               IF (Reg1 > "0000") THEN
                  YN <= '1';
               ELSIF (Reg2 > "0000")THEN
                  YN <= '1';
               ELSE
                  YN <= '0';
               END IF;

            WHEN "0000000000010000" =>
               YN <= '0';
               IF (Reg1 > "0011") THEN
                  YN <= '1';
               ELSIF (Reg2 > "0011")THEN
                  YN <= '1';
               ELSE
                  YN <= '0';
               END IF;

            WHEN "0000000000100000" =>
               YN <= '0';
               IF (Reg1 > "0100") THEN
                  YN <= '1';
               ELSIF (Reg2 > "0100")THEN
                  YN <= '1';
               ELSE
                  YN <= '0';
               END IF;

            WHEN "0000000001000000" =>
               YN <= '0';
               IF (Reg1 > "0011") THEN
                  YN <= '1';
               ELSIF (Reg2 > "0011")THEN
                  YN <= '1';
               ELSE
                  YN <= '0';
               END IF;

            WHEN "0000000010000000" =>
               YN <= '0';
               IF (Reg1 > "0011") THEN
                  YN <= '1';
               ELSIF (Reg2 > "0011")THEN
                  YN <= '1';
               ELSE
                  YN <= '0';
               END IF;

            WHEN "0000000100000000" =>
               YN <= '0';
               IF (Reg1 > "0001") THEN
                  YN <= '1';
               ELSIF (Reg2 > "0001")THEN
                  YN <= '1';
               ELSE
                  YN <= '0';
               END IF;

            WHEN OTHERS => YN <= '-';
         END CASE;
      END IF;
   END PROCESS;
END Calculation;
```

Figure 6.vi– ALU Problem 3 VHDL Code

References:

- Brown, S. and Vranesic, Z. Fundamentals of Digital Logic with VHDL Design, Third Edition, McGraw-Hill, 2009.
- COE 328 Homepage. Tutorial; Lecture Notes; Lab Manual. Available: https://www.ee.ryerson.ca/~courses/coe328/.
- Quartus II 13.0 Software Download. Available: https://drive.google.com/file/d/1gk_DXx_EMd9KWnmCq7FayTjwcodLiu-Y/view
- Sedaghat, Reza - Digital Systems COE 328 Lecture PowerPoint Presentations