Table of Contents

Background:

The final laboratory experiment consists of the engineering of a simple general-purpose processor which can perform logical and arithmetic operations. The experiment was conducted with the Computer Aided Design software program, Quartus II by Altera [1]. This design software allows one to engineer, analyze, and test various digital systems with the use of its widely compatible features [1]. The objective of this laboratory experiment was to implement previously learned experimental and theoretical knowledge by designing an arithmetic and logical unit (ALU) on a computer aided software program [2]. This lab will demonstrate the development, compilation, and simulation of all functions of the ALU representing the student ID: 500960836. The general purpose processor will be structured with the use of VHDL code, logic equations, truth tables, primitive gates and schematic diagrams. [2]. The results are tested and verified through the simulation of the timing diagram as the results will be proven through the variation of binary numbers from the sequencing of clock cycles [3]. The entire engineering procedure will be displayed sequentially throughout the report and a thorough analysis regarding the entire general processing unit will be highlighted near the conclusion.

Introduction:

The general purpose processor (GPU) is composed of various components that were designed and enhanced over the COE 328 course period [2]. Each component is designated with crucial responsibilities that allow the GPU to function and produce the proper gathering, processing and display of binary data [2]. The GPU consists of 4 main components that allow the component to maintain its functionality: the storage unit, control unit, ALU core, and the display unit [2]. Without any one of these components will result in a lack of functionality of a portion of the digital system. The storage unit is represented as the temporary storage units that will procure the inputs of 8-bit binary data from the user to the ALU [2]. The Control unit is composed of the finite state machine and the 4x16 decoder which selects the proper operation for the ALU to perform [2]. The ALU core is the heart of the GPU as this component accepts all the inputs and performs the desired operation which will be decoded as an output [2]. The display unit consists of various seven segment displays (SSEGs) that decode the 4-bit output from the ALU and the control unit to display operation results and the sequencing of the student ID from the finite state machine [2]. As stated, the GPU will lose portions of its functionality if any of the components are removed for instance, if the SSEG was removed from the GPU then the operations will still perform, but there will be no display [3]. The components in this GPU are a combination of both sequential registers, ALU core, Mealy state machine, and combinational SSEGs, and the decoder [2]. The combination of each individual component showcases how modern electronics can be engineered as a system today [3].

Apparatus:

To conduct this experiment the following materials were utilized:

- Altera- Quartus II version 13.0 [1]
- Windows operating system
- Ryerson Lab Manual 5 for COE 328 - Digital Systems [4]
    - Figure 4 The code template for finite state machine
    - Figure 5 The block diagram for the finite state machine
- Ryerson Lab Manual 6 for COE 328 - Digital Systems [2]
    - Figure 1 The block diagram of the GPU (General Purpose Processor)
    - Figure 2 The code template for implementing the register storage unit
    - Figure 3 4x16 decoder diagram
    - Table 1 ALU (Arithmetic and Logical Unit) core operations for problem 1
    - Figure 4 The code template for ALU core
    - Table C core operations for problem 2
    - Assignment I core operation for problem 3
- Fundamentals of Logic with VHDL design by Brown, S. and Vranesic, Z. [3]
    - Figure 6.47 The code template for seven segment display
    - Figure 6.30 The code template for 2-4 decoder

Procedure:

1. Procure two 8-bit inputs ‘A’ and ‘B’ utilizing the last 4 digits of the assigned student ID in binary [2]
2. Design the VHDL code to implement a functional 8-bit input synchronous register using the code template provided in Figure 2 of Ryerson’s COE 328 laboratory manual 6 [2]
3. Implement the up counter with the synchronous output of student ID on the Mealy finite state machine [2]
4. Design the 4x16 decoder with the addition of several primitive gates to extend the previously implemented 3x8 decoder [2]
5. Design the microcode operator as the outputs from the decoder to act as a selector for the GPU [2]
6. Assemble both the Mealy machine and the constructed decoder to create the control unit for the GPU [2]
7. Utilize Table 1 from Ryerson’s COE 328 laboratory manual 6 to implement each logical and arithmetic operation on the provided code template of Figure 4 of Ryerson’s COE 328 laboratory manual 6 [2]

8. Import each component of the GPU to a schematic diagram (seven segment display, registers, ALU core, Mealy machine, decoder) with the included input, output and ground pins [2]
9. Identify each pin components and arrange the schematic components to showcase the connection of data buses that represent the proper signals [2]
10. Compile and simulate the schematic diagram and verify the functionality of each component and operation through the waveforms at every sequential clock cycle [2]
11. Utilize Table C of problem set 2 and modify the ALU core to represent the listed functions in the identical order of operations [2]
12. Compile and simulate the schematic diagram and verify the functionality of each component and operation through the waveforms at every sequential clock cycle [2]
13. Utilize the Assignment I to implement the proper functionality to solve problem 3 by altering the ALU and seven segment display components [2]
14. Compile and simulate the schematic diagram and verify the functionality of each component and operation through the waveforms at every sequential clock cycle [2]

Laboratory Preparation Data:

Student ID: 500960836
The provided binary vectors A and B represent the input values throughout laboratory experiment 6.
Input binary vectors (resembling last 4 digits of student ID):

$A = (0000\ 1000)_2$ (08) [2]

$B = (0011\ 0110)_2$ (36) [2]

Finite state machine utilized: Mealy
States: up counter from state 0 to state 8
Student ID output from Mealy machine: synchronous with up counter [2]
Assigned problem set 2: Table C [2]
Assigned problem set 3: Assignment I [2]
Pin declarations:
OP: 16-bit decoder output to ALU
R1: lower 4-bit representation of the ALU results
R2: higher 4-bit representation of the ALU results

Components:

Registers (Storage Unit)

The Register is an important part of the GPU circuit as it is the temporary storage unit for the input values as mentioned before [2]. The register is a series of flip-flops which are storage devices that operate at a rising edge of a clock input. The registers implemented in this laboratory experiment are known as D flip-flops as the data that is inputted is immediately output out of Q when the clock is at rising edge [3]. For the register to operate, the data must be set in as the clock transitions from 0 to 1 and this statement is recognized by the older statement code in VHDL: Clock ‘EVENT AND Clock= ‘1’ [3]. The registers represent a parallel in and parallel out synchronous circuit as all the bits that are input directly into the 8 flip-flops are output exactly at the same clock cycle. Since Quartus recognizes the amount of flip-flops needed to store the 8-bits the 8 parallel flip-flops are not individually coded in VHDL [3]. To present how the data was input in the truth table, table 1.1 emphasizes the ‘Q’ output for each particular data input. The truth tables are similar for both registers A and B aside from the assigned input data values.

$Clock$	D	Q	$\overline{Q}$	$Description$
0	x	Q	$\overline{Q}$	Memory no change
1	0	0	1	Reset data
1	1	1	0	Set data

Table 1.0 Truth table for single positive edge triggered D-flip-flop [5].

Clock	D7	D6	D5	D4	D3	D2	D1	D0	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
0	x	x	x	x	x	x	x	x	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0

Table 1.1 Truth table for 8-bit storage unit A. Only the outputs are shown to showcase material that is utilized in the waveform output. The input data represents 0 and 8 in binary [5].

Clock	D7	D6	D5	D4	D3	D2	D1	D0	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
0	x	x	x	x	x	x	x	x	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1	0	0	1	1	1	0	1	1	0

Table 1.2 Truth table for 8-bit storage unit B. Only the outputs are shown to showcase material that is utilized in the waveform output. The input data represents 3 and 6 in binary [5].

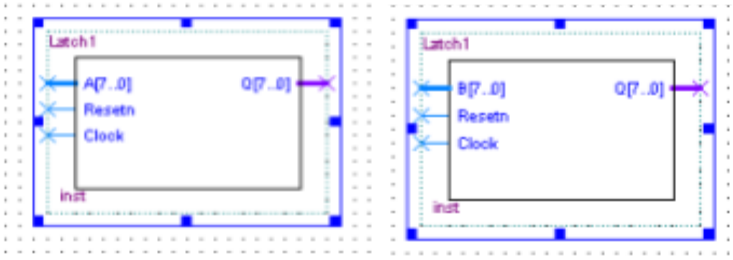

Figure 1.0 Block diagram components for registers A and B. The components are the same but named separately for A and B [1].

```
1   --Janakan Sivaloganathan
2   --500960836
3   --COE 328 DIGITAL SYSTEMS
4
5   LIBRARY ieee;
6   USE ieee.std_logic_1164.all;
7
8   ENTITY Latch1 IS --Although it says latch, this code represents a data input
    fli-flop register 8-bit output follows the 8-bit inputs
9      PORT( A: IN STD_LOGIC_VECTOR (7 DOWNTO 0); --8 bit A input (Similar to B input)
10           Resetn, Clock: IN STD_LOGIC; --1 bit clock input and 1 bit reset input bit
11           Q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)); --8 bit output
12  END Latch1;
13
14  ARCHITECTURE Behaviour OF Latch1 IS --Architecture displays what behaviours the
    varibles undergo
15  BEGIN
16     PROCESS(Resetn, Clock) --Process takes reset and clock as inputs it is ordered on
    a sensitivity list. Clock is prioritized
17     BEGIN
18        IF Resetn= '1' THEN --when reset input is '1' the latches does not operate (0)
    ouput
19              Q<= "00000000";
20        ELSIF Clock'EVENT AND Clock= '1' THEN  --True on the risig edge of the clock
    signal (changes values on the event change)
21                                              --Represents a positive edge signal
    since '1' is where the even transitions to
22           Q<= A; --Output can represent either A or B (A is used in this case)
23        END IF;
24     END PROCESS; --Sequential PROCESS statement must end
25  END Behaviour;
26
```

Figure 1.1 The VHDL code representation for 8 data flip-flops composed as a register. Two of the same components were created to represent the binary vectors A and B. The code showcases that all the flip-flops are synchronously input and output with the clock

signal statement thus, it is justifiable that the flip-flops are connected in parallel where the outputs are emitted in parallel as well [2].

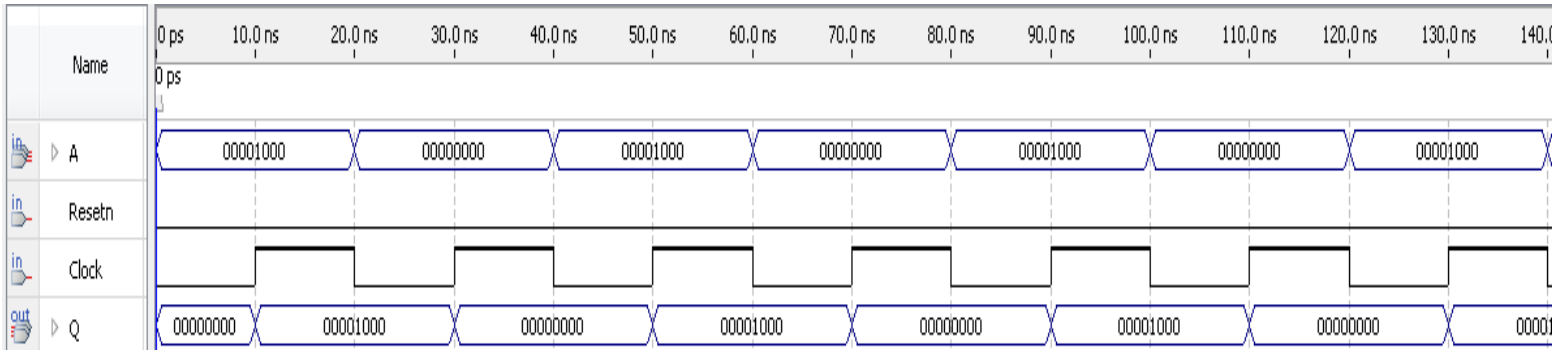

Figure 1.2 Timing diagram representation for the 8-bit register that operates at the rising edge of the clock. This diagram showcases a 10ns delay starting at 0ns as the data transitions and outputs for a clock cycle. The binary equivalent "08" is transitioned alongside "00" to showcase this propagation delay, otherwise the input would be set constantly to "00001000" [1].

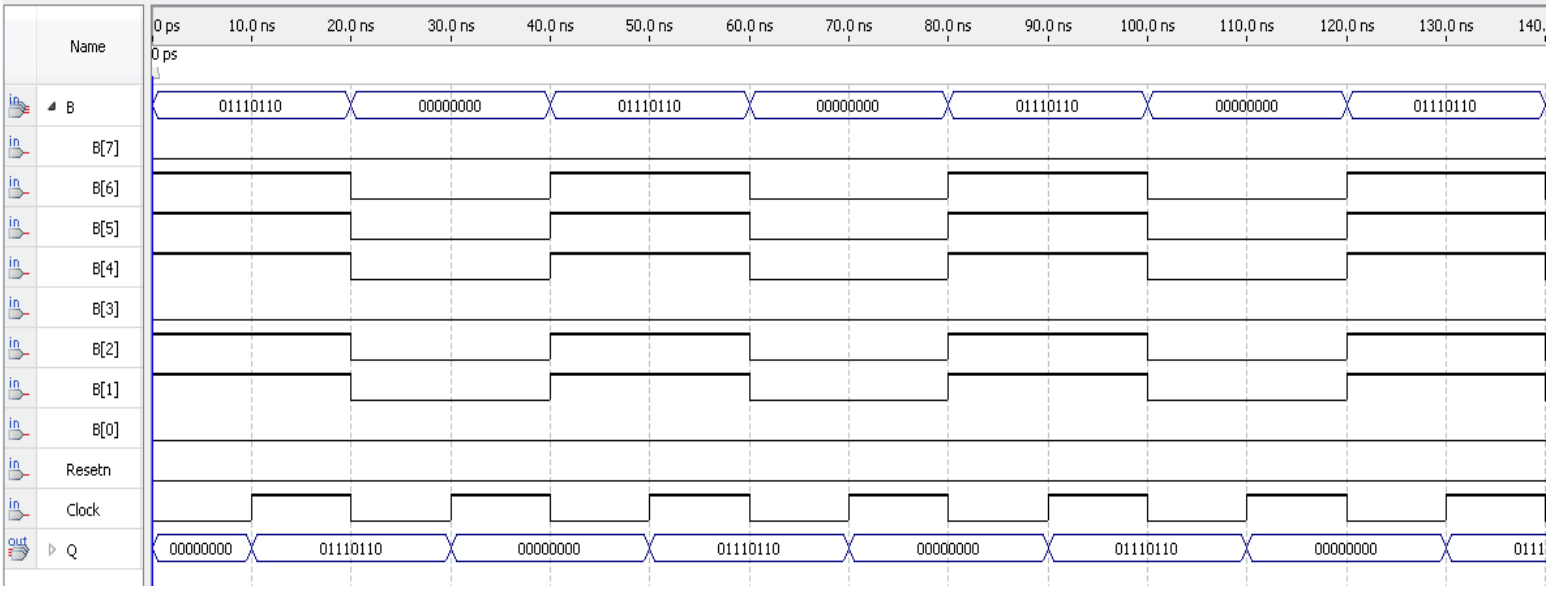

Figure 1.3 Timing diagram representation of the 8-bit register for signal B. Similar approach is used to showcase the propagation delay, the B decimal values "36" are transitioning on and off with "00". To display that the flip-flops within the register are edge triggered the input data is set for 20ns as the clock transition occurs. The "Q" output still follows the data from 10ns to 30ns clock cycle [2].

4x16 Decoder (First Portion of the Control Unit)

The 4x16 decoder was implemented with the use of the constructed 3x8 decoders from the previously implemented 2x4 decoders. The decoder prioritizes the decoding of the current states that are sequenced from the finite state machine which is then sent to the ALU core [2]. The

decoder acts as the selector for the various ALU operations that are depicted in Tables 1 and C provided by Ryerson's laboratory experiment 6 [2]. In order to represent the instructions for the operations, the output for the decoder must transition a '1' from the least significant bit (LSB) to the most significant bit (MSB) for 9 indexes [2]. Since only 9 instructions are current states are transitioned, to prevent other decoded outputs from occurring the "WHEN OTHERS" clause was utilized to represent a hexadecimal value of 0 output on the SSEG [3]. The 4x16 decoder passes the signal OP which is a 16 bit microcode to the ALU core everytime the clock transitions due to the sequencing of the finite state machine rising edge clock transitions [3].

En	w3	w2	w1	w0	y15	y14	y13	y12	y11	y10	y9	y8	y7	y6	y5	y4	y3	y2	y1	y0
0	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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

Table 2.0 Truth table for a 4 to 16-bit decoder. Only 9 instances were decoded since operations are only dependent on 8 different states from the Mealy machine [3].

```vhdl
--Janakan Sivaloganathan
--500960836
--COE 328 Digital Systems
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY decode IS
   PORT(w      :IN  STD_LOGIC_VECTOR(1 DOWNTO 0); --Input port
        En     :IN  STD_LOGIC; --Enable
        y      :OUT STD_LOGIC_VECTOR(0 TO 3)); --Decoded output
END decode;

ARCHITECTURE Behavior OF decode IS
   SIGNAL Enw: STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
   Enw<= En&w; --Concatenate the enable bit and the input bit to represent the truth table function
   WITH Enw SELECT
   y<= "1000" WHEN "100", --Decode representation of '0' in binary with the enable as '1'
       "0100" WHEN "101", --Decode representation of '1' in binary
       "0010" WHEN "110", --Decode representation of '2' in binary
       "0001" WHEN "111", --Decode representation of '3' in binary
       "0000" WHEN OTHERS; --If the input values are not as listen above then the decoder will be set to '0' Error catch
END Behavior;
```

Figure 2.0 The VHDL code representation for one of the 2x4 decoders utilized to construct the 4x16 decoder. In order to minimize errors or if the enable is turned off, default “0000” is set as output [3].

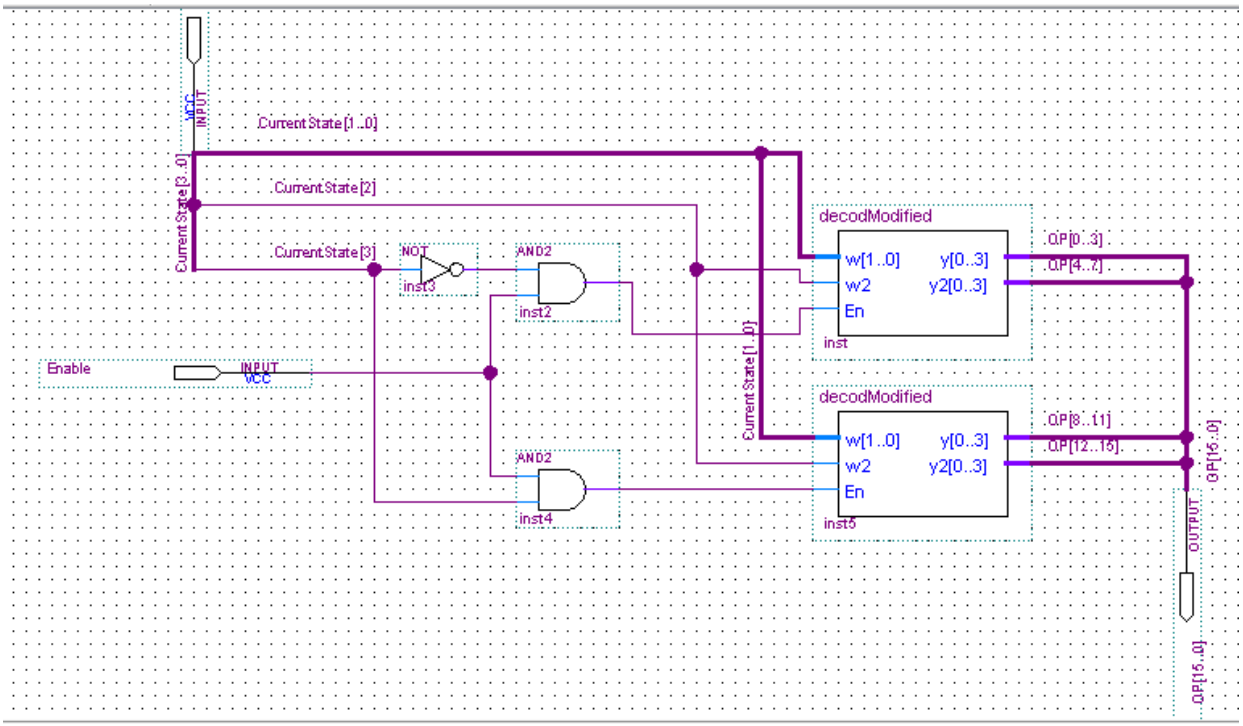

Figure 2.1 The schematic diagram of two 3x8 decoders that were enhanced to accept an additional input value. This diagram showcases the hierarchy of smaller decoders that

are used to construct the 4x16 decoder. In order to represent the OP output signal of 16 bits, all output line segments were joined starting from the most significant down to the least significant bits [2].

Figure 2.2 The block schematic diagram that is composed of the entire 4x16 function. This schematic is utilized in the main GPU for cleanliness [1].

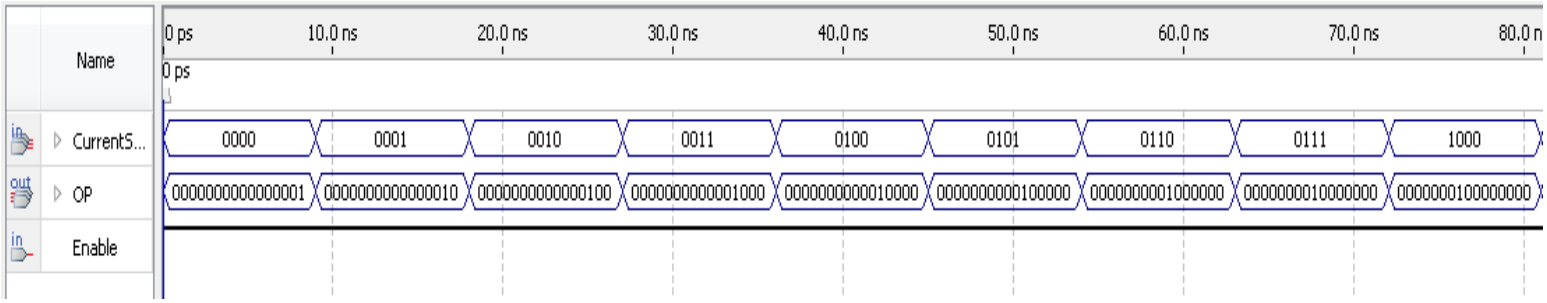

Figure 2.3 The timing diagram representation for the 4x16 decoder. This diagram showcases the 9 microcodes and the 8 states that the Mealy machine will output. Since the ALU is only composed of 9 operations, states above state 8 are listed as insignificant and the decoder will output "0000000000000000" to the ALU [1].

Mealy Finite State machine (Second Portion of the Control Unit)

The Mealy finite state machine is a sequential component which transitions through the 9 states from 0 to 8 corresponding to each student ID number. The Mealy finite state machine is a synchronous circuit where the next states are dependent on the current state and the input values [6]. One of the major differences that distinguishes the Mealy machine from the moore is that the output depends on the input values as well as the current states [6]. The Mealy machine is the primary source of the sequencing of the operations on the ALU core as it transitions current states as the data input is present and the clock transitions at the rising edge [6]. Since the Mealy machine displays the output concurrently as it transitions states, the student ID is ahead of the current state in the waveforms with respect to the propagation delay. It is important to recognize that since only 9 states are needed to represent the student ID, 2^4 flip-flops are needed to represent the circuit [6]. The remaining 7 flip-flops are stated as "don't cares" which is represented as 0's. The control unit thus represents the Mealy machine as the source of transitions and the decoder as the source of transmission to the ALU core to perform operations.

The state table and state assigned tables provided below represent the incrementing counter of current states alongside the corresponding student ID.

Present State	Next State		Output (z)	
	w= 0	w= 1	w= 0	w= 1
S0	S0	S1	5	0
S1	S1	S2	0	0
S2	S2	S3	0	9
S3	S3	S4	9	6
S4	S4	S5	6	0
S5	S5	S6	0	8
S6	S6	S7	8	3
S7	S7	S8	3	6
S8	S8	S9	6	5

Table 3.0 State table for an up counter Mealy machine that outputs the student ID: 500960836 [3].

Present State				Next State								Output (z)							
				w=0				w=1				w=0				w=1			
y3	y2	y1	y0	Y3	Y2	Y1	Y0	Y3	Y2	Y1	Y0	z3	z2	z1	z0	z3	z2	z1	z0
0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	0	1	0	0	1
0	0	1	1	0	0	1	1	0	1	0	0	1	0	0	1	0	1	1	0
0	1	0	0	0	1	0	0	0	1	0	1	0	1	1	0	0	0	0	0
0	1	0	1	0	1	0	1	0	1	1	0	0	0	0	0	1	0	0	0
0	1	1	0	0	1	1	0	0	1	1	1	1	0	0	0	0	1	1	1

0	1	1	1	0	1	1	1	1	0	0	0	0	1	1	1	0	1	1	0
1	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	1	0	1

Table 3.1 State assigned table representation of the Mealy machine current states, next states and outputs on the transition lines [3].

```
1   --Janakan Sivaloganathan
2   --500960836
3   --COE 328 DIGITAL SYSTEMS
4
5   library ieee;
6   use ieee.std_logic_1164.all;
7   entity Mealy is
8      port
9      (
10        clk    :in  std_logic;
11        data_in :in std_logic;
12        ResetFSM :in std_logic;
13        student_id :out std_logic_vector(3 downto 0);
14        current_state :out std_logic_vector(3 downto 0));
15  end entity;
16  architecture fsm of Mealy is
17     type state_type is (s0, s1, s2, s3, s4, s5, s6, s7, s8); --The states that are
    implemented on this finite state machine
18     signal yfsm : state_type;
19  begin
20  -- the clock is the only synchronous impact for the next states, if it is clocked
    and the data is either 0 or 1 it changes state
21  -- reset is the asynchronous switch to reset the state back to its original
22  -- implementing an upcounter (simultaneously outputting the student id in order)
23     process (clk, ResetFSM) --The clock is prioritized in the PROCESS statement
24     begin
25
26        if ResetFSM = '1' then --Set the machine will reset when it is 1
27           yfsm <= s0;
28        elsif (clk 'EVENT AND clk = '1') then
29        case yfsm is
30           when s0=>
31              IF data_in = '1' THEN
32              yfsm <= s1; --state 0 goes to state 1 if clocked with an input of '1'
33              ELSE
34              yfsm <= s0; --else stay the same
35              END IF;
36
37           when s1=>
38              IF data_in = '1' THEN
39              yfsm <= s2; --state 1 goes to state 2 if clocked with an input of '1'
40              ELSE
41              yfsm <= s1; --else stay the same
42              END IF;
43
44           when s2=>
45              IF data_in = '1' THEN
46              yfsm <= s3; --state 2 goes to state 3 if clocked with an input of '1'
47              ELSE
48              yfsm <= s2; --else stay the same
49              END IF;
50
51           when s3=>
52              IF data_in = '1' THEN
53              yfsm <= s4; --state 3 goes to state 4 if clocked with an input of '1'
54              ELSE
55              yfsm <= s3; --else stay the same
56              END IF;
57
```

```
58            when s4=>
59               IF data_in = '1' THEN
60               yfsm <= s5; --state 4 goes to state 5 if clocked with an input of '1'
61               ELSE
62               yfsm <= s4; --else stay the same
63               END IF;
64
65            when s5=>
66               IF data_in = '1' THEN
67               yfsm <= s6; --state 5 goes to state 6 if clocked with an input of '1'
68               ELSE
69               yfsm <= s5; --else stay the same
70               END IF;
71
72            when s6=>
73               IF data_in = '1' THEN
74               yfsm <= s7; --state 6 goes to state 7 if clocked with an input of '1'
75               ELSE
76               yfsm <= s6; --else stay the same
77               END IF;
78
79            when s7=>
80               IF data_in = '1' THEN
81               yfsm <= s8; --state 7 goes to state 8 if clocked with an input of '1'
82               ELSE
83               yfsm <= s7; --else stay the same
84               END IF;
85
86            when s8=>
87               IF data_in = '1' THEN
88               yfsm <= s0; --state 8 goes to state 0 if clocked with an input of '1'
89               ELSE
90               yfsm <= s8; --else stay the same
91               END IF;
92
93            when others =>
94               yfsm <= s0; --Redundant but still secure to catch any errors
95
96         end case;
97      end if;
98   end process;
99
100  --Since this is a Mealy Machine, the output depends on the primary input (w) or
     (data_in) to present a specific output
101  --The process statement determines which variables what the outputs will be
     sensistive to, in this case the state and primary input
102  --the state of the machine is not dependant of any other variable (current state) so
     the outputs are only represented by
103  --the current state and the input data (1 if statement to represent its change in
     transition line and next state)
104  --the mealy machine outputs the nextstate and nextstate output immediately as it
     outputs on its transition line
105  process (yfsm, data_in)
106  begin
107     case yfsm is
108        when s0=>
109           current_state<= "0000";
110           IF data_in = '1' THEN
111           student_id <= "0000"; --  0 Output when w= 1
```

```
112             ELSE
113             student_id <= "0101"; --  5 Output when w=0
114             END IF;
115
116         when s1=>
117             current_state<= "0001";
118             IF data_in = '1' THEN
119             student_id <= "0000"; -- 0 Output when w= 1
120             ELSE
121             student_id <= "0000"; -- 0 Output when w=0
122             END IF;
123
124         when s2=>
125             current_state<= "0010";
126             IF data_in = '1' THEN
127             student_id <= "1001"; -- 9 Output when w= 1
128             ELSE
129             student_id <= "0000"; -- 0 Output when w=0
130             END IF;
131
132         when s3=>
133             current_state<= "0011";
134             IF data_in = '1' THEN
135             student_id <= "0110"; -- 6 Output when w= 1
136             ELSE
137             student_id <= "1001"; -- 9 Output when w=0
138             END IF;
139
140         when s4=>
141             current_state<= "0100";
142             IF data_in = '1' THEN
143             student_id <= "0000"; -- 0 Output when w= 1
144             ELSE
145             student_id <= "0110"; -- 6 Output when w=0
146             END IF;
147
148         when s5=>
149             current_state<= "0101";
150             IF data_in = '1' THEN
151             student_id <= "1000"; -- 8 Output when w= 1
152             ELSE
153             student_id <= "0000"; -- 0 Output when w=0
154             END IF;
155
156         when s6=>
157             current_state<= "0110";
158             IF data_in = '1' THEN
159             student_id <= "0011"; -- 3 Output when w= 1
160             ELSE
161             student_id <= "1000"; -- 8 Output when w=0
162             END IF;
163
164         when s7=>
165             current_state<= "0111";
166             IF data_in = '1' THEN
167             student_id <= "0110"; -- 6 Output when w= 1
168             ELSE
169             student_id <= "0011"; -- 3 Output when w=0
170             END IF;
```

```
171
172         when s8=>
173            current_state<= "1000";
174            IF data_in = '1' THEN
175            student_id <= "0101"; -- 5 Output when w= 1
176            ELSE
177            student_id <= "0110"; -- 6 Output when w=0
178            END IF;
179
180         when OTHERS =>
181         current_state<= "----"; --represents the binary number 14 which is displayed
      as "null" where all the lights will be off, this is to minimize
      --error
182            student_id<= "----"; --error
183
184      end case;
185   end process;
186   end architecture;
```

Figure 3.0 The VHDL code representation for a Mealy machine that transitions current states as an up counter. The Mealy machine will then output the student ID if the data in is consistently '1' as: 096083650 [4].

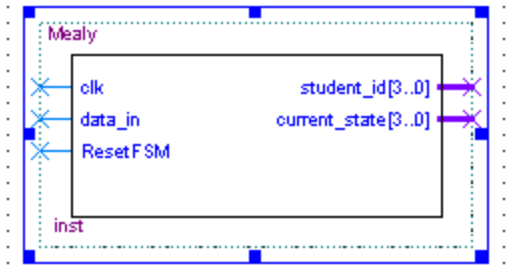

Figure 3.1 The block diagram schematic for the Mealy finite state machine component. This component is utilized in the GPU schematic diagram for cleanliness [1].

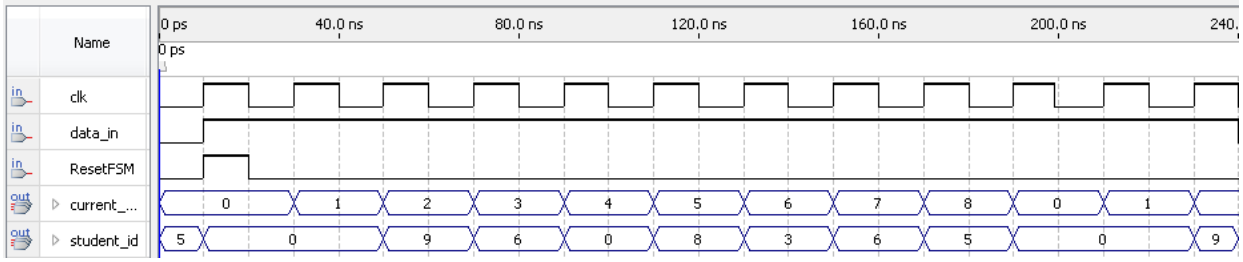

Figure 3.2 The timing diagram representation for the up counting Mealy machine. The radix is set to represent unsigned decimal numbers to showcase the sequencing of current states and student ID more clearly. Since the Mealy machine outputs immediately depending on the input data and clock, the student ID is thus ahead by 1 clock cycle (20ns period for this instance) [1].

Seven Segment Display

The seven segment display (SSEG) represents the display of the GPU. In order to display if the desired operations that are being performed, a source of display is needed [2]. For this instance

the display output to the Field Programmable Gate Array is not applicable so the most efficient way to display the output as a component is to implement it to an led. The seven segment display is similar to a decoder as it accepts a 4-bit binary vector and decodes it to its respective hexadecimal representations [3]. The SSEG also acts as a display for the negative signal for when the ALU core processes a negative the display will recognize it. The SSEG can be utilized for multiple scenarios for instance, if an overflow were to be produced by the ALU a conditional statement can be implemented to turn on a specific amount of leds when the overflow is recognized. For the provided GPU, the SSEG will be separated into two components as the 8 bit results from the ALU cannot be displayed on a single hexadecimal screen [2]. Two of the same SSEG components were utilized for the first two GPUs, the first one will represent the lower 4-bits and the negative, and the second SSEG will represent the higher 4-bits of the ALU output. To showcase if any errors occurred throughout the process, the "WHEN OTHERS" clause was implemented which will always display a blank screen or 0 if the decoder does not output correctly [3]. The use of the SSEG is merely to represent the values to a more readable format for many who are not familiar with binary value representations of decimal numbers [3].

```
1   --Janakan Sivaloganathan
2   --500960836
3   --COE 328 Digital Systems
4   LIBRARY ieee;
5   USE ieee.std_logic_1164.all;
6   USE ieee.std_logic_unsigned.all;
7
8   ENTITY SSEG IS
9      PORT (bcd    :IN   STD_LOGIC_VECTOR(3 DOWNTO 0);
10           neg    :IN   STD_LOGIC;
11  --neg value added to determine if the value from ASU (0,1) is positive or negative
12           leds   :OUT STD_LOGIC_VECTOR(0 TO 6);
13  --The values outputted from the ASU is represented through 4 bits, the BCD decoder
    will decode the 4 bits into an array of 7 bits
14  --representing the MSB, 0 to 6, LSB
15           ledss  :OUT STD_LOGIC_VECTOR(0 TO 6) );
16  END SSEG;
17
18  ARCHITECTURE Behaviour OF SSEG IS
19  BEGIN
20  --Sequential process of negative value and bcd with the "Case" statement
21  --The statements cyrcles from either '1' or not for neg
22     PROCESS (neg, bcd)
23     BEGIN
24
25        CASE neg IS
26           WHEN '1'     =>ledss<= "0000001";
27           --When the ASU provides a 1 the value is negative
28           --The sign is independant of the magnitude so it can range from (-F to +F)
29           WHEN OTHERS  =>ledss<= "0000000";
30           --When the ASU is positive, the output of the neg value will be equal to 0
    to represent the Hexadecimal value
31           --Can also be implemented as null value (output nothing), ("-------")
```

```
32          END CASE;
33          --The bcd is the magnitude of the ASU unit
34          CASE bcd IS             -- abcdefg
35          --Representations of the binary values in Hexidecimal
36          --Once the values are greater than 9 in magnitude, they will follow the
    alphabetical sequence until F
37              WHEN "0000"    =>leds<= "1111110"; --0
38              WHEN "0001"    =>leds<= "0110000"; --1
39              WHEN "0010"    =>leds<= "1101101"; --2
40              WHEN "0011"    =>leds<= "1111001"; --3
41              WHEN "0100"    =>leds<= "0110011"; --4
42              WHEN "0101"    =>leds<= "1011011"; --5
43              WHEN "0110"    =>leds<= "1011111"; --6
44              WHEN "0111"    =>leds<= "1110000"; --7
45              WHEN "1000"    =>leds<= "1111111"; --8
46              WHEN "1001"    =>leds<= "1111011"; --9
47              WHEN "1010"    =>leds<= "1110111"; --10 (outputs A)
48              WHEN "1011"    =>leds<= "0011111"; --11 (outputs b)
49              WHEN "1100"    =>leds<= "1001110"; --12 (outputs C)
50              WHEN "1101"    =>leds<= "0111101"; --13 (outputs D)
51              WHEN "1110"    =>leds<= "1001111"; --14 (outputs E)
52              WHEN "1111"    =>leds<= "1000111"; --15 (outputs F)
53              WHEN OTHERS    =>leds<= "0000000"; --(null)
54          END CASE;
55      END PROCESS;
56  END Behaviour;
```

Figure 4.0 The VHDL code representation for the common SSEG that is utilized in various locations of the GPU. This SSEG will showcase the negative display turned off if the value output on the display is “0000000”. The values are represented in hexadecimal format where if any error were to occur the output will also result in a null screen (all lights turned off) [3].

Seven Segment Display Modified

As mentioned before, the SSEG can be implemented in multiple different ways for various purposes and scenarios. The SSEG for problem set 3 requires the display of ‘y’ or ‘n’ to verify if the assignment of the problem is true or not [2]. The SSEG was modified to display “0010101” for ‘n’ and “0111011” for ‘y’ when any of the 4-bit split vectors from the ALU output is less than the student ID binary value. In order to minimize the errors, the “WHEN OTHERS” clause was utilized for whenever the input values to the SSEG is not ‘y’ then n will appear [3]. Although another statement can be used to represent error, for this implementation of the GPU and for simplicity the error was included with the ‘n’ output.

```
1   --Janakan Sivaloganathan
2   --500960836
3   --COE 328 Digital Systems
4   LIBRARY ieee;
5   USE ieee.std_logic_1164.all;
6   USE ieee.std_logic_unsigned.all;
7
8   ENTITY SSEG IS
9      PORT (bcd     :IN    STD_LOGIC_VECTOR(3 DOWNTO 0);
10  --neg value removed, it is redundant
11            leds    :OUT STD_LOGIC_VECTOR(0 TO 6));
12  --The only values needed is the output seven segment display and the input 4 bit
    value passed from the ALU
13  END SSEG;
```

```
14
15  ARCHITECTURE Behaviour OF SSEG IS
16  BEGIN
17  --Sequential process of negative value and bcd with the "Case" statement
18  --The statements cycles from either '1' or not for neg
19     PROCESS (bcd)
20     BEGIN
21
22        CASE bcd IS             -- abcdefg
23        --Representations of the binary values in Hexidecimal
24           WHEN "1111"    =>leds<= "0111011"; --y in hexidecimal, the bcd input from
    the ALU will signifiy if one of the bits are higher
25                                              --by outputting the value 1111, else any
    other value produced will give n
26                                              --the OTHERS statement reduces the errors
    as all values asside from the specified
27                                              --y value produces the 'n' output in
    hexidecimal
28           WHEN OTHERS    =>leds<= "0010101"; --n value (The WHEN OTHERS clause
    collects all unecessary outputs (errors) as well)
29        END CASE;
30     END PROCESS;
31  END Behaviour;
```

Figure 5.0 The VHDL code representation for the modified SSEG to represent the results from the ALU 3 of problem set 3. To represent the error, all unwanted values will result in 'n'. Although another value can be initialized to showcase error, 'n' was utilized for timing diagram consistency [3].

Problem Sets:

Arithmetic and Logical Unit 1:

The arithmetic and logical unit (ALU) is the core and the "brain" of the GPU. This component performs all the desired operations that are provided by the various problem sets according to the signal inputs that are received [2]. The ALU component is designed with the use of Figure 4 from Ryerson's COE 328 laboratory manual 6. In order to complete the design, various functionalities must be included to represent the operations that will be performed from the sequencing of the control unit outputs [2]. The first ALU core will represent the basic arithmetic operations such as summation and difference and the various logical operations in the following order: NOT, NAND, NOR, AND, XOR, OR, and XNOR [3]. The logical operations will consist of the operations among individual bits of the input binary vector provided by the A and B signals [3]. Further explanation of the results and the code implementation will be provided in the analysis section of the report.

The ALU core 1 requires 4 components to function properly: signals A, B, Clock, and the microcode from the control unit [2]. The 8-bit signals are required for the ALU to perform the operations between both A and B when necessary, multiple signals and bits can be altered as it is the engineer's discretion on how the ALU should be designed. This ALU will perform operations on both 8-bit vectors to represent the operations performed on 2 decimal student ID digits. The microcode input is necessary as it will act as it will select the desired operation, for this ALU the 16-bit output microcodes will cycle through the sequencing order of Table 1 [2].

The clock is extremely important as without the sequential input, the ALU will not operate. The ALU requires the clock since it will only perform the operations when a transition of the positive rising edge is detected in the timing diagram (from 0 to 1) [3]. The clock input can be also modified to be negative edge as well as it depends on the engineers desire for functionality [3]. The 3 outputs of the ALU consists of: the negative signal, “R1” and “R2”. Since the SSEG component is not capable of displaying two hexadecimal values on a single screen, the ALU requires 2 separate outputs to display the most significant bits and least significant bits of the result. In order to showcase clarification the pins in the GPU are named R1 LSB to showcase the lowest 4 bits of the result and R2 MSB to showcase the higher 4 bits of the result [2]. The negative symbol is implemented as another output to showcase if any of the operations among A and B result in a negative number (negative 2’s complement) [3]. Additionally, more outputs can be implemented to showcase significant instances of the ALU operations such as the overflow from addition or subtraction in the results, but for simplicity it was not implemented in this laboratory experiment.

Function #	Microcode	Boolean Operation/Function	Results (8-bit)
1	0000000000000001	$Summation\ (A,\ B)$	01111110
2	0000000000000010	$Difference\ (A,\ B)$	00101110
3	0000000000000100	$\overline{A}$	11110111
4	0000000000001000	$\overline{A \cdot B}$	11111111
5	0000000000010000	$\overline{A + B}$	11000001
6	0000000000100000	$A \cdot B$	00000000
7	0000000001000000	$A \oplus B$	00111110
8	0000000010000000	$A + B$	00111110
9	0000000100000000	$\overline{A \oplus B}$	11000001

Table 4.0 Table 1 operations and the correct 8-bit ALU output results for each microcode [2].

```
1   --Janakan Sivaloganathan
2   --500960836
3   --COE 328 Digital Systems
4   library IEEE;
5   use IEEE.STD_LOGIC_1164.ALL;
6   use IEEE.STD_LOGIC_UNSIGNED.ALL; --Library allows to use the various logical and
    arithmetic operations
7   use IEEE.NUMERIC_STD.ALL;
8   entity ALU is
9   port( Clock: in std_logic; --input clock signal
10        A,B: in unsigned (7 downto 0); --8-bit inputs from latches A and B
11        student_id: in unsigned (3 downto 0); --4 bit student id from FSM
12        OP: in unsigned (15 downto 0); --16-bit selector for Operation from Decoder
13        Neg: out std_logic; --if the result is negative Neg will be 1, else 0
14        R1: out unsigned (3 downto 0); --lower 4-bits of 8-bit result output
15        R2: out unsigned (3 downto 0)); --higher 4-bits of 8-bit Result output
16  end entity ALU;
17
18  architecture calculation of ALU is --temporary signal declaration
19  signal Reg1, Reg2, Result: unsigned(7 downto 0):=(others =>'0');
20  signal Reg4: unsigned (0 to 7); --Used as a placeholder and out put for other ALU
    modifications
21  begin
22  Reg1 <= A; --temporarily stores A in Reg1 local variable
23  Reg2 <= B; --temporarily stores B in Reg1 local variable
24
25  process(Clock, OP) --Clock is prioritized
26  begin
27     if(rising_edge(Clock))THEN --do the calculation at the postive edge of clock
    cycle (rising edge so when the clock goes from 0 to 1) new standard
28     Neg<= '0';
29        case OP is
30           WHEN "0000000000000001" =>
31                 Result<= (Reg1 + Reg2); --addition performed
32           WHEN "0000000000000010" =>
33           IF(Reg1 < Reg2) THEN
34              Result<= NOT(Reg1 - Reg2) + 1; --subtraction performed (difference)
35              Neg<= '1'; --negative if the first variable is less than second
36           ELSE
37              Result<= (Reg1 - Reg2);
38           END IF;
39           WHEN "0000000000000100" =>
40                 Result<= (NOT(Reg1)); --inverse operation
41           WHEN "0000000000001000" =>
42                 Result<= (Reg1 NAND Reg2); --Boolean NAND operation
43           WHEN "0000000000010000" => --Boolean NOR operation
44                 Result<= (Reg1 NOR Reg2);
45           WHEN "0000000000100000" => --Boolean AND operation
46                 Result<= (Reg1 AND Reg2);
47           WHEN "0000000001000000" => --Boolean OR operation
48                 Result<= (Reg1 XOR Reg2);
49           WHEN "0000000010000000" => --Boolean XOR operation
50                 Result<= (Reg1 OR Reg2);
51           WHEN "0000000100000000" => --Boolean XNOR operation
52                 Result<= (Reg1 XNOR Reg2);
53           WHEN OTHERS =>
54              Result<= "--------"; --"dont care do nothing" prevent errors from
    occuring and distrubing output waveform
55


56  end process;
57  R1 <= Result (3 downto 0); --Since the output seven segment can only retreive 4 bits
    we split the 2 decimals
58  R2 <= Result (7 downto 4); --R1 is the LSB and R2 is the MSB part of the Result
59  end calculation;
60
```

Figure 6.0 The VHDL code representation for ALU 1 operations. Each operation utilized the result output signal to showcase the final value. The final value is then split amongst two outputs R1 which is the 4 least significant bits of the result and R2 which is the greatest 4 significant bits of the result [2].

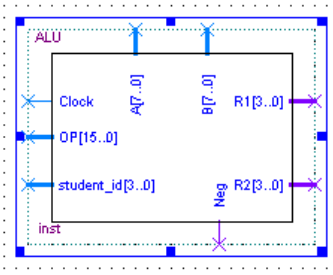

Figure 6.1 The block schematic diagram representation of the ALU 1. This component represents the core of the GPU which will accept the various inputs and output the desired results for the user. The student ID input is grounded as there is no use of it for ALU 1 [1].

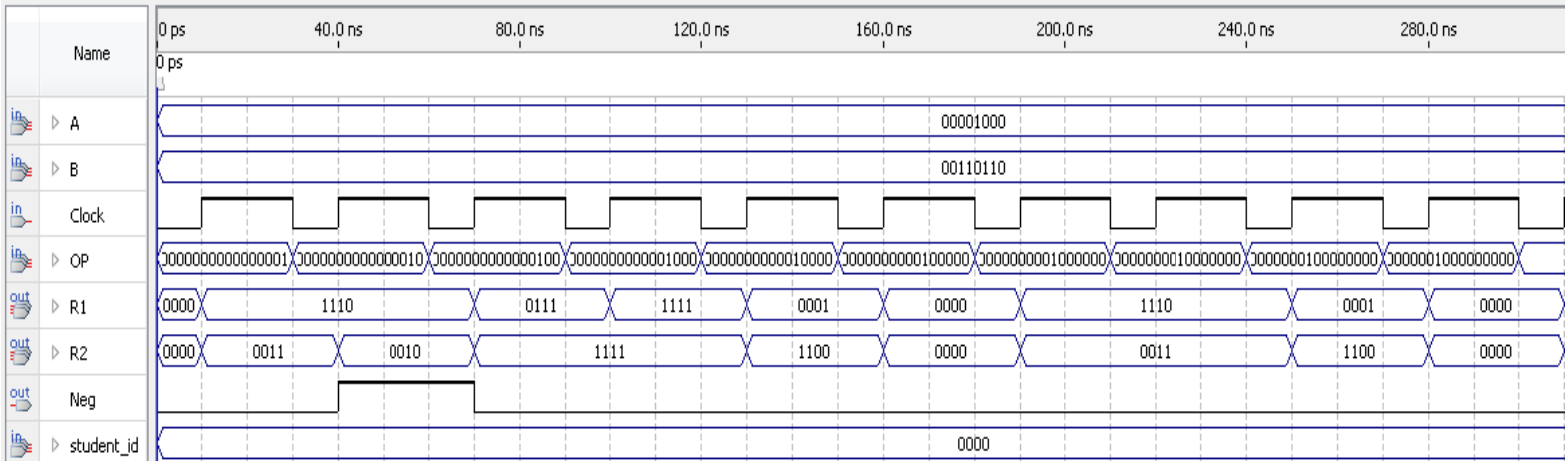

Figure 6.2 The timing diagram representation for the ALU 1 core. This timing diagram showcases the operations performed for a total time of 310ns for each 30ns clock period. Since no input is recognized for the single ALU the student ID is represented as “0”. The initial values at 10ns are disregarded as it represents a beginning offset where the input values are not directly input into the ALU. The timing diagram effectively showcases the transition of each operation as the microcode changes. The timing diagram verifies all the operations among “08” and “36” binary equivalents as it matches the expected outputs in Table 4.0 when R2 and R1 are combined. The negative output verifies the arithmetic operation where subtraction occurs (second operation) [1].

Arithmetic and Logical Unit 2:

The Arithmetic and Logical Unit 2 (ALU 2) has the similar purpose of functionalities as the ALU 1. The ALU 2 requires the modification of operations for each microcode that is output from the control unit. This problem set highlights how powerful computer aided design tools are since minor changes in the VHDL code in the ALU 2 project can completely change the GPU function and purpose. The only changes that are to be implemented are the modification to each instance of microcode in the VHDL code, this allows the inputs and the outputs to be similar to

the ALU 1. The negative output signal is still implemented to showcase if the difference of the first operation produces a negative in the result. Further explanation of the results and the code implementation will be provided in the analysis section of the report.

Function #	Microcode	Boolean Operation/Function	Results (8-bit)
1	0000000000000001	$Difference\ (A,\ B)$	00101110
2	0000000000000010	$2's\ complement\ (B)$	11001010
3	0000000000000100	$Swap\ the\ lower\ 4\ -\ bits\ of\ B\ wit$	00000110
4	0000000000001000	$Produce\ null\ on\ the\ output$	00000000
5	0000000000010000	$Decrement\ (B)\ by\ 5$	00110001
6	0000000000100000	$Invert\ the\ bit\ -\ significance\ order\ of\ A$	00010000
7	0000000001000000	$Shift\ B\ to\ the\ left\ by\ 3\ bits\ input\ the\ vacant\ bits\ with$ '1'	10110111
8	0000000010000000	$Increment\ A\ by\ 3$	00001011
9	0000000100000000	$Invert\ all\ bits\ of\ B$	11001001

Table 5.0 Table C operations and the correct 8-bit ALU output results for each microcode [2].

```
1   --Janakan Sivaloganathan
2   --500960836
3   --COE 328 Digital Systems
4   library IEEE;
5   use IEEE.STD_LOGIC_1164.ALL;
6   use IEEE.STD_LOGIC_UNSIGNED.ALL;
7   use IEEE.NUMERIC_STD.ALL;
8   entity ALU is --The same name is used for simplicity purposes, various projects for
    each ALU is used to differentiate each component
9   port( Clock: in std_logic;
10        A,B: in unsigned (7 downto 0); --input 8-bit values
11        student_id: in unsigned (3 downto 0); --not used (acts as a display to the
    SSEG for the time being
12        OP: in unsigned (15 downto 0); --The deocer output from the Mealy state machine
13        Neg: out std_logic; --negative signal, displays if there is a negative result
    in the SSEG or not
14        R1: out unsigned (3 downto 0);
15        R2: out unsigned (3 downto 0));
16  end entity ALU;
17
18  architecture calculation of ALU is
19  signal Reg1, Reg2, Result: unsigned(7 downto 0):=(others =>'0');
20  signal Reg4: unsigned (0 to 7); --Utilized Reg4 to represent the negative 2's
    complement of B since initially Reg4 is "00000000"
```

```
21    begin
22    Reg1 <= A; --temporary signals to hold input bits
23    Reg2 <= B; --temporary signals to hold input bits
24    process(Clock, OP) --sequential PROCESS statement clock is prioritized
25    begin
26       if(rising_edge(Clock))THEN --Sequential statement since the clock initiates the
      operations at rising edge
27       Neg<= '0'; --Negative is placed to represent '0' for all the operations that do
      not produce a negative number= 1
28                  --If negative then the operation will explicity state neg =1 which
      will signal the output else, it will stay 0
29          case OP is
30             WHEN "0000000000000001" =>          --Find the difference between A and B
31                   IF(Reg1 < Reg2) THEN          --If A is lower than B then we get a
      negative number
32                   Result<= NOT(Reg1 - Reg2) + 1; --The 2's complement is turned into
      the positive equivalent by conversion to 1's component
33                   Neg<= '1'; --If negative then the sign ouput is 1
34                   ELSE
35                   Result<= (Reg1 - Reg2); --Else the positive arithmetic operation is
      performed as the complements are disregarded
36                                           --if not included the negative will be on for
      the full run
37                   END IF;
38             WHEN "0000000000000010" => --Produces the 2's complement of B, the negative
      is removed since it does not explicity ask for signed decimal
39                   Result<= Reg4 - Reg2;
40             WHEN "0000000000000100" => --Swaps the lower 4 bits of A with the lower 4
      bits of B
41                   Result<= ((Reg1(7 downto 4)) & (Reg2(3 downto 0)));
42             WHEN "0000000000001000" => --Produces a null on the output
43                   Result<= "00000000";
44             WHEN "0000000000010000" => --Decrement B by 5 (subtract by 5) using the
      arithmetic operation
45                   IF(Reg2 < 5) THEN


46                   Result<= (Reg2 - 5);
47                   END IF;
48             WHEN "0000000000100000" => --Inverts all the bit-significance order of A
      (switch the MSB and LSB)
49                   Result<= (Reg1(0) & Reg1(1) & Reg1(2) & Reg1(3) & Reg1(4) & Reg1(5) &
      Reg1(6) & Reg1(7));
50             WHEN "0000000001000000" => --Shifts B to the left by 3 bits and then adds 3
      '1' inputs
51                   Result<= (shift_left(reg2,3)) + "111"; --Since the shift operator
      leaves 3 vacant indexes behind we can input the
52                                                          --1's by performing simple
      concantenation with the result of the shift
53                                                          --utilized the shift operator
      function provided by the IEEE.NUMERIC_STD.ALL, the
54             WHEN "0000000010000000" => --Increments A by 3 using arithmetic operation
      (individual bits are not being performed on)
55                   Result<= Reg1 + 3;
56             WHEN "0000000100000000" => --Inverts all the bits of B, all the 0's turn
      into 1 and all the 1's turn into 0's
57                   Result<= NOT Reg2;
58             WHEN OTHERS =>
59                Result<= "--------"; --"dont care do nothing if the microcode does not
      recieve a valid OP signal" (minimalize errors)
60
61          end case;
62       end if;
63    end process;
64    R1 <= Result (3 downto 0); --lower significant bits of the result
65    R2 <= Result (7 downto 4); --higher significant bits of the result
66    end calculation;
67
```

Figure 7.0 The VHDL code representation for ALU 2. The code features the same functionality of the ALU 1, but with different implementations for its operations [2].

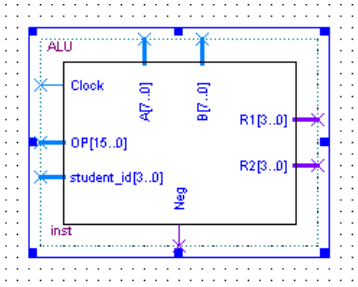

Figure 7.1 The block schematic diagram representation of the ALU 2. This component represents the core of the GPU which will accept the various inputs and output the desired results for the user. The student ID input is grounded as there is no use of it for ALU 2 [1].

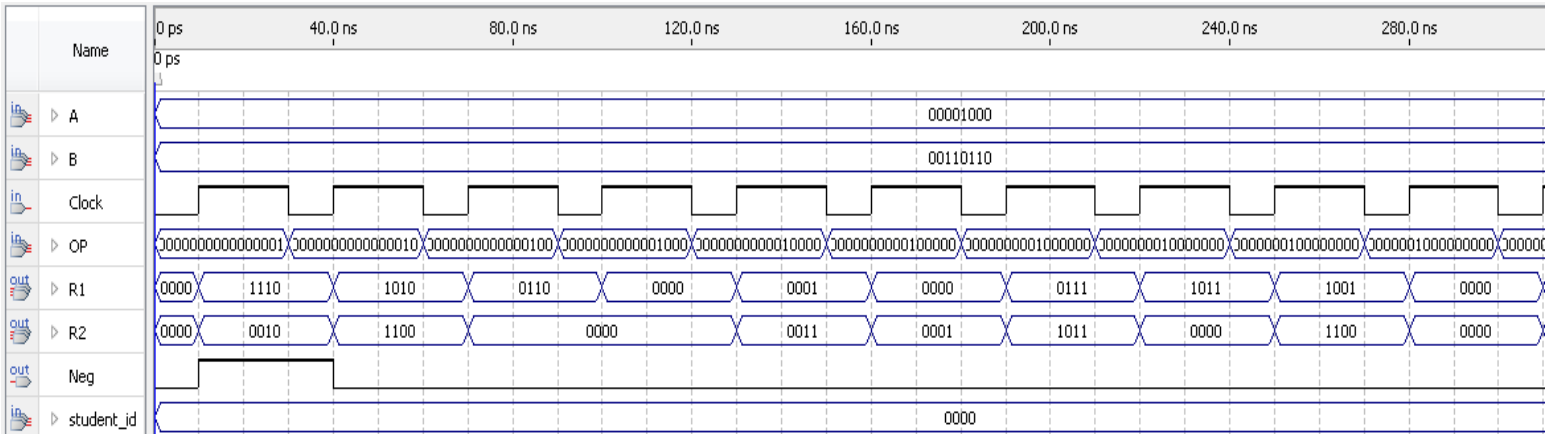

Figure 7.2 The timing diagram representation for the ALU 2 core. This timing diagram showcases the operations performed for a total time of 310ns for each 30ns clock period. Since no input is recognized for the single ALU the student ID is represented as "0". The initial values at 10ns are disregarded as it represents a beginning offset where the input values are not directly input into the ALU. The timing diagram verifies all the operations among "08" and "36" binary equivalents as it matches the expected outputs in Table 5.0 when R2 and R1 are combined. The negative signal represents the verification when the difference between A and B occurs in Table 5.0 (first operation) [1].

Arithmetic and Logical Unit 3:

The arithmetic and logical unit 3 (ALU 3) requires the modification to represent problem set 3 from the assignment I of Ryerson's COE 328 laboratory manual 6 [2]. The ALU 3 will perform the same functions as the ALU 1, but with a few enhancements. The ALU 3 must be modified so each operation that is conducted on signals A and B will output the corresponding 'y' for yes or 'n' for no for the particular assignment I [2]. Assignment I states of any of the decimal values of B are less than the student ID value, then output a yes [2]. In order for this problem to be solved, an additional student ID input must be designed to accept the 4-bit binary values that will be output from the Mealy machine from the control unit [2]. Since the Mealy machine

outputs each student ID on the transition line, at every clock rising edge an new input will be transitioned to the ALU 3 [6].

Assignment I requires comparison between the higher and lower bit values of B with the student ID [2]. Since each microcode instruction must present the correct output of the result, a sequential IF statement was utilized outside of the CASE statement of the ALU 3 operations [3]. The IF statement must be within the process statement because in VHDL the IF statement is known as a sequential conditional statement that will only proceed in the PROCESS order [3]. Due to the fact both the LSB and MSB portions of B represent different decimal equivalents, the IF statement will require either one of the values to verify a 'y' output [2]. If either the LSB or MSB 4-bit portion of the B signal is less in binary equivalency compared to the 4-bit student ID, then the single 4-bit result will select the 'y' output on the SSEG [2]. The resolution to problem set 3 will also produce the 'n' if none of the B decimal digits are less or equivalent to the student ID [3]. The implementation of this feature is applicable to all microcodes as the conditional statement is after all the beginning PROCESS have occurred. Only one output is implemented on ALU 3 since the output requires a single 4-bit value to represent a 'y' and a 'n' on the SSEG output [2]. The implemented output binary value to showcase the 'y' or 'n' on the SSEG is: "0000" for no and "1111" for yes which will then be decoded to present the 'y' and 'n' equivalent on the seven segment display. The negative is disregarded as the whole values are taken into consideration when comparing the student ID and the B values. In addition, it is clear to understand that the LSB and MSB binary portions of B are not affected by the operations thus, the register equivalents are truly compared with the student ID [3].

Function #	Microcode	Student ID	Binary Values for B	Results on SSEG (Any Values Less than Student ID?) 'y' or 'n'
1	0000000000000001	"0000" (0)	"0111" (3) "0110" (6)	"0010101" (n)
2	0000000000000010	"1001" (9)	"0111" (3) "0110" (6)	"0111011" (y)
3	0000000000000100	"0110" (6)	"0111" (3) "0110" (6)	"0111011" (y)
4	0000000000001000	"0000" (0)	"0111" (3) "0110" (6)	"0010101" (n)
5	0000000000010000	"1000" (8)	"0111" (3) "0110" (6)	"0111011" (y)
6	0000000000100000	"0111" (3)	"0111" (3) "0110" (6)	"0010101" (n)
7	0000000001000000	"0110" (6)	"0111" (3) "0110" (6)	"0111011" (y)
8	0000000010000000	"0101" (5)	"0111" (3) "0110" (6)	"0111011" (y)

9	0000000100000000	“0000” (0)	“0111” (3) “0110” (6)	“0010101” (n)

Table 6.0 Expected operations and the correct 8-bit ALU output results for each microcode [2].

```
1   --Janakan Sivaloganathan
2   --500960836
3   --COE 328 Digital Systems
4   --Same operations and variables of ALU except for the conditional statement that
    produces a different output "yes or no"
5
6   library IEEE;
7   use IEEE.STD_LOGIC_1164.ALL;
8   use IEEE.STD_LOGIC_UNSIGNED.ALL;
9   use IEEE.NUMERIC_STD.ALL;
10  entity ALU is
11  port( Clock: in std_logic;
12        A,B: in unsigned (7 downto 0);
13        student_id: in unsigned (3 downto 0);
14        OP: in unsigned (15 downto 0);
15        Sign: out unsigned (3 downto 0)); --Implemented a new output instead of Result
    since the result is being compared (more straight forward)
16  end entity ALU;
17
18  architecture calculation of ALU is
19  signal Reg1, Reg2, Result: unsigned(7 downto 0):=(others =>'0');
20  signal Reg4: unsigned (0 to 3); --The Reg4 in this project will be the output signal
    of the conditional statement that compares the student
21                                  --ID and the higher and lower significant bits of
    the B input bit vector
22  begin
23  Reg1 <= A;
24  Reg2 <= B;
25
26  process(Clock, OP)
27  begin
28     if(rising_edge(Clock))THEN
29        case OP is
30           WHEN "0000000000000001" =>
31                 Result<= (Reg1 + Reg2); --add
32           WHEN "0000000000000010" =>
33           IF(Reg1 < Reg2) THEN
34              Result<= NOT(Reg1 - Reg2) + 1; --subtract
35           ELSE
36              Result<= (Reg1 - Reg2);
37           END IF;
38           WHEN "0000000000000100" =>
39                 Result<= (NOT(Reg1));     --inverse
40           WHEN "0000000000001000" =>
41                 Result<= (Reg1 NAND Reg2); --nand
42           WHEN "0000000000010000" => --nor
43                 Result<= (Reg1 NOR Reg2);
44           WHEN "0000000000100000" => --and
45                 Result<= (Reg1 AND Reg2);
46           WHEN "0000000001000000" => --or
47                 Result<= (Reg1 XOR Reg2);
48           WHEN "0000000010000000" => --xor
49                 Result<= (Reg1 OR Reg2);
50           WHEN "0000000100000000" => --xnor
51                 Result<= (Reg1 XNOR Reg2);
52           WHEN OTHERS =>
53              Result<= "--------"; -- to prevent any errors from unwanted microcode
    outputs
54        end case;
```

```
55      end if;
56            IF((Reg2(7 downto 4) <student_id) or (Reg2(3 downto 0) <student_id)) THEN
    --Conditional statement to compare the 4-bit student ID
57
    --with the higher and lower significant 4-bit values of B
58            Reg4<= "1111"; --Implemented the "1111" to be recognized as the yes or 'y'
    if atleast one of the B decimal digits are less than student ID
59            ELSE
60            Reg4<= "0000"; --If the student ID is greater or if the values of B and
    student ID are only equal then the output will produce false "0000"
61                          --The ELSE statement is utlized to capture all possibilities
    that can be presented if it does not satisfy the conditional statement
62            END IF;       --thus, error will be minimalized
63    end process;
64    Sign<= Reg4; --the sign output will act as the selector which will tell the SSEG
    which display to output for the implemented 'y' and 'n'
65    end calculation;
```

Figure 8.0 The VHDL code representation for ALU 3. The ALU 3 utilizes the same operations as ALU 1, but it is composed of several enhancements to satisfy problem set 3. A single output signal "Sign" is utilized to represent a 'y' or a 'n' for the conditional statement provided on line 56. This portion of the problem set also requires an additional input of the student ID which will be output from the Mealy machine to compare the register values to the sequencing student ID values [2].

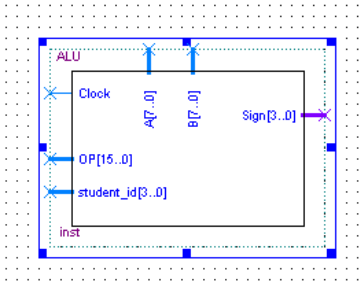

Figure 8.1 The block schematic diagram representation of the ALU 3. This component represents the core of the GPU which will accept the various inputs and output the desired results for the user. For this instance the student ID input is utilized [1].

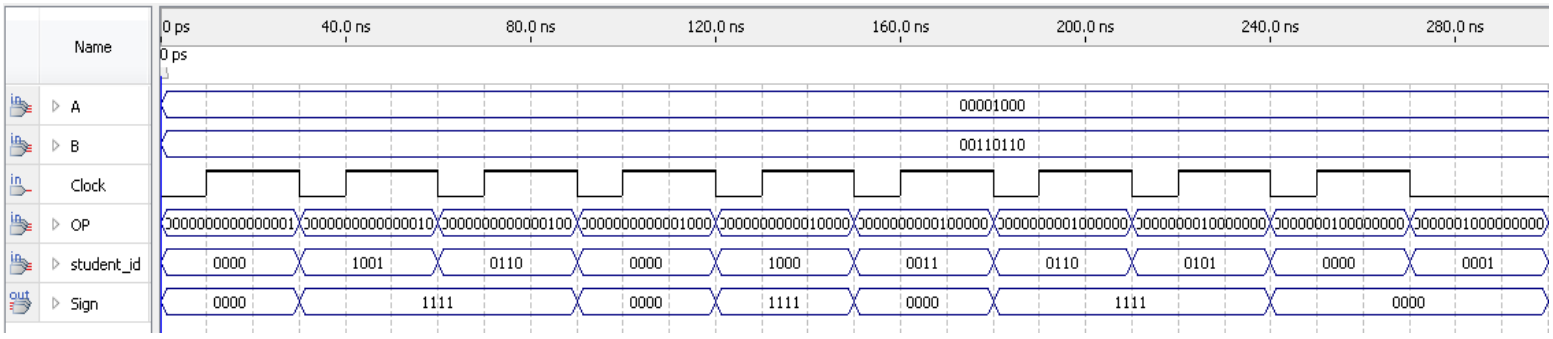

Figure 8.3 The timing diagram representation for the ALU 3. This timing diagram showcases the 9 operation cycle completed in 270ns without any propagation delay. It is also recognizable that for 10ns there will be a beginning offset, but it is disregarded as it

does not dramatically affect the timing diagram. The student Id is manually input to resemble the Mealy machine sequencing as it transitions on each clock rising edge. The timing diagram showcases the correct values as it matches the expected output listed in Table 6.0. To present error minimization within the timing diagram, when the clock is turned off at 280ns the output sign will still display a ‘n’ shown as “0000” [1].

Analysis:

General Processing Unit 1:

The General Processing Unit 1 is composed of the 4 different components: control unit, storage unit, display and the ALU 1 core [2]. As the clock transitions to the positive edge with the input data, the Mealy machine transitions to the nextstate and the output is displayed concurrently as it transitions [6]. This process will display the current states as an up counter transitioning from state 0 to state 8 [5]. The Mealy machine as well is composed of 4 flip-flops to store the 9 values of the student ID as it transitions through each output process [6]. It is also recognized that the output of the student ID on the waveform does not start from the digit ‘5’ as the Mealy machine always outputs the next value on the transition line. To maintain consistency throughout the waveform, all the reset values are set to initiate when a ‘1’ occurs. As well, the input data, enable and register signals are set as its appropriate values throughout to present consistent outputs. The 4x16 decoder will then decode the output from the finite state machine to a 16-bit input to the ALU which will perform operations from the inputs of the register units [2]. The ALU will then display the correct results on 2 different SSEGs in hexadecimal format to represent both the lower and high 4 significant bit values of the result [3].

The waveforms produced for the first ALU are presented as correct since the performed operation for each current state is aligned with Table 1.0 [2]. Although there is a propagation delay where the output is delayed by 1 clock period or (30ns period), the correct values are still verified [2]. The propagation delay is presented when various components are constructed together, this is proven as for previous single timing diagram representations do not represent the delays as effectively [3]. R LSB and R MSB present the hexadecimal representations of the outputs so whenever a ‘0’ is present the light of the seven segment display will be turned off. To represent if any fault where to happen during this process multiple error conditions are introduced in the VHDL code. The Mealy machine and the decoder includes the “WHEN OTHERS” clause to represent null on the output or “0000000” when values other than the desired outputs occur [3]. The timing diagram highlights this aspect as when the enable is turned off at 480ns the output at the 490ns clock transition represents a ‘0’ in hexadecimal format on the outputs. Majority of the arithmetic and boolean operations on the 8-bit operands were implemented with the key words hosted by the “std_logic_unsigned” library [3]. This

library Provides the ability to implement arithmetic and logical operations on unsigned binary vectors [3]. Logical operations perform operations on boolean values in bitwise format unlike arithmetic where the value operations are performed such as addition and subtraction amongst binary vectors [3]. This is useful as the negative display can easily represent the whole number of the result and prevent unnecessary VHDL to fix signed bit values. To enhance error minimization, a parity implementation can be conducted with an additional concatenation to the bit vectors that are being transmitted from each component [3]. For this laboratory, this idea was disregarded for simplicity since tangible hardware is not utilized.

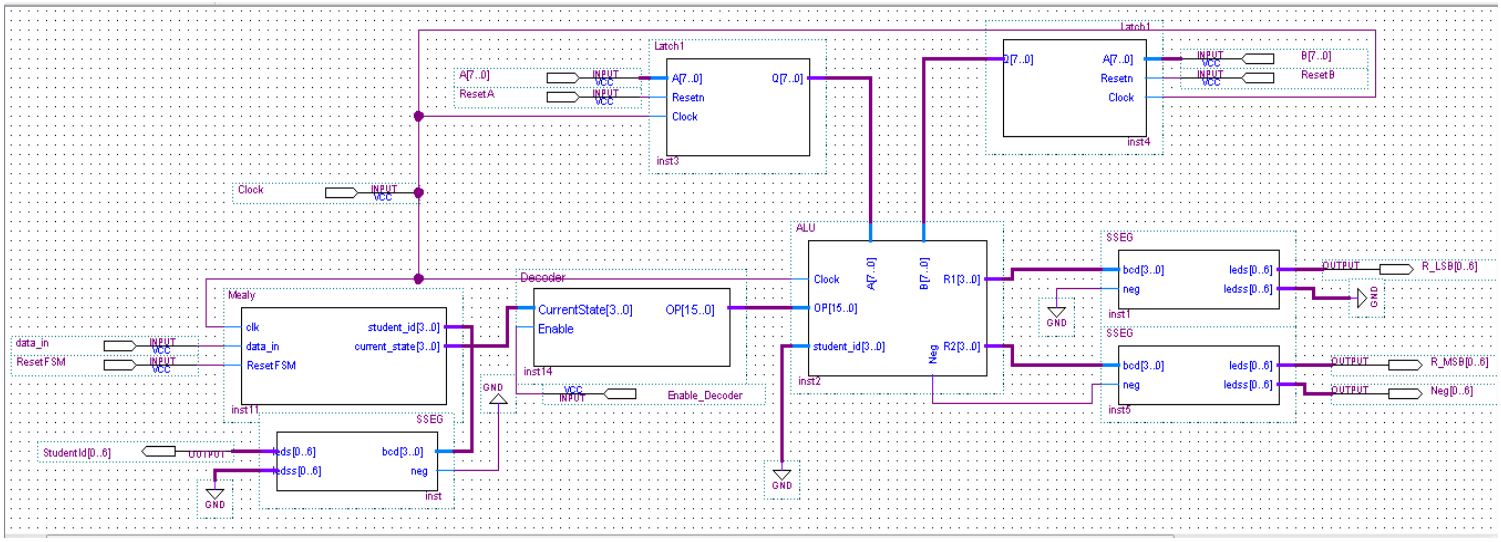

Figure 9.0 The schematic diagram representation of the general processing unit 1. This diagram showcases all the connections among the sequential and combinational circuits and Pins [2].

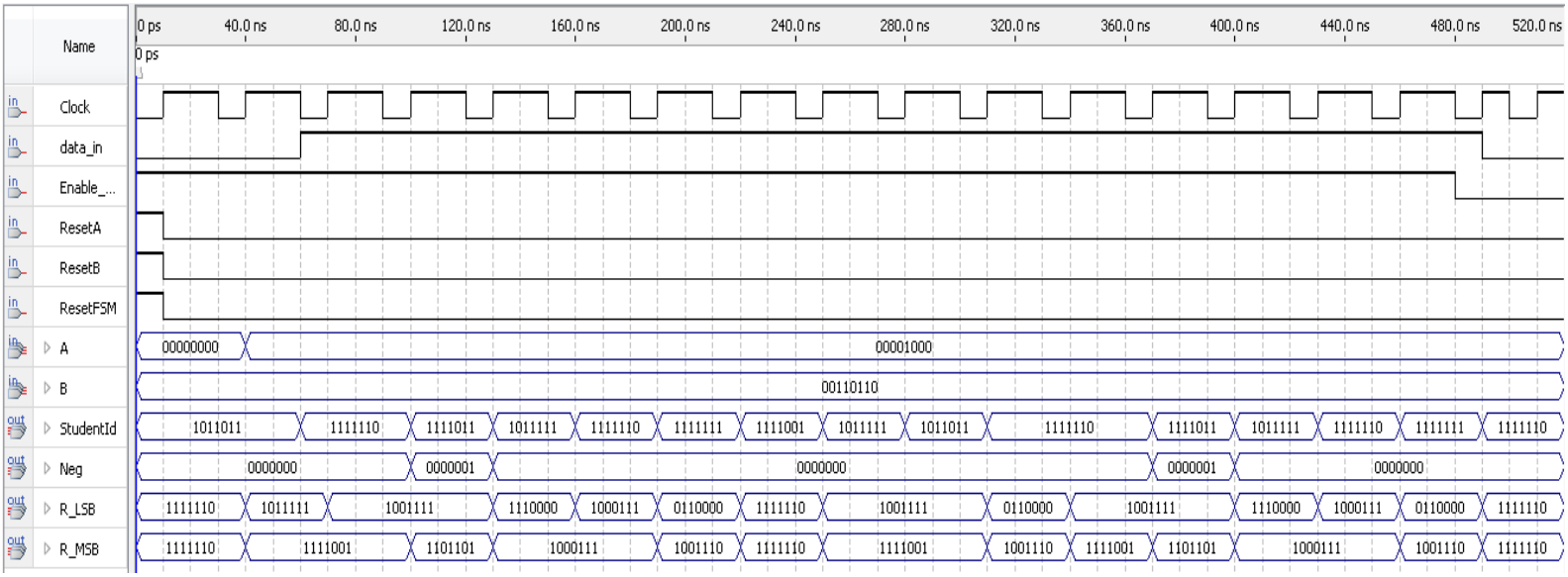

Figure 9.1 The presented timing diagram showcases a propagation delay on the output of the SSEG as every current state transition outputs the operation on the next clock. The output will be delayed by 30ns clock period since the operation is performed after with the input values prior to the 30ns. To showcase errors in the timing diagram, when the decoder is turned off at 490ns the resultant output also produces a '0' hexadecimal equivalent on the output. The correct student number starting from '5' can be displayed starting at 280ns due to the Mealy characteristics however, the student id is still delayed by 30ns due to the propagation delay [3].

General Processing Unit 2:

The general processing unit 2 follows the same characteristics as the GPU 1 except, the values of the ALU have changed to different operations [2]. The outputs are all verified in regards with the timing propagation as each SSEG display corresponds to the correct operation that is to be performed on the previous clock transition register inputs [3]. As stated before, in order to minimize errors during the GPU process, many additional codes have been included in VHDL to represent all the unknown output possibilities which will act as an indicator on the SSEG when such occurrences do occur [3]. An example of an error detection within the waveform is presented as when the decoder is turned off, the error will be catched and the output on the SSEG will result in '0'. The null output of the SSEG represents additional faults that may occur aside from the enable portion of the 4x16 decoder [3].

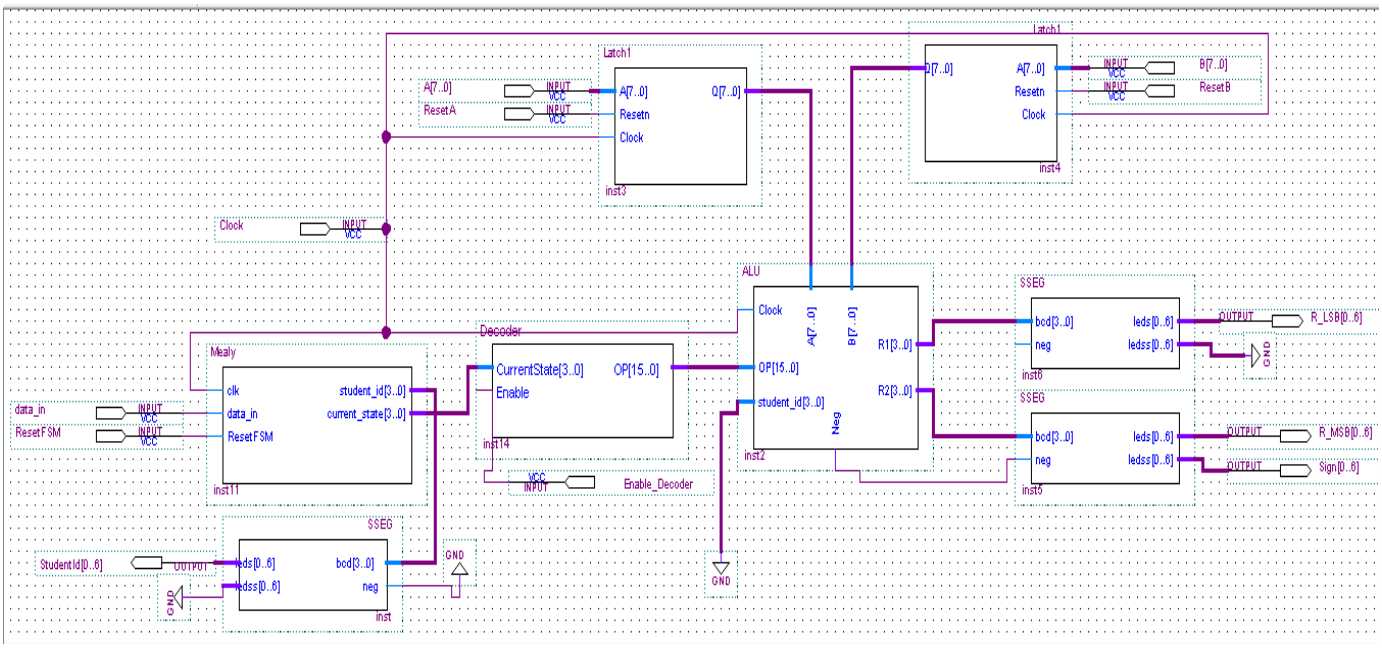

Figure 10.0 The schematic diagram representation of the general processing unit 2. This schematic diagram is similar to the GPU 1 as the differences among them are within the ALU [2].

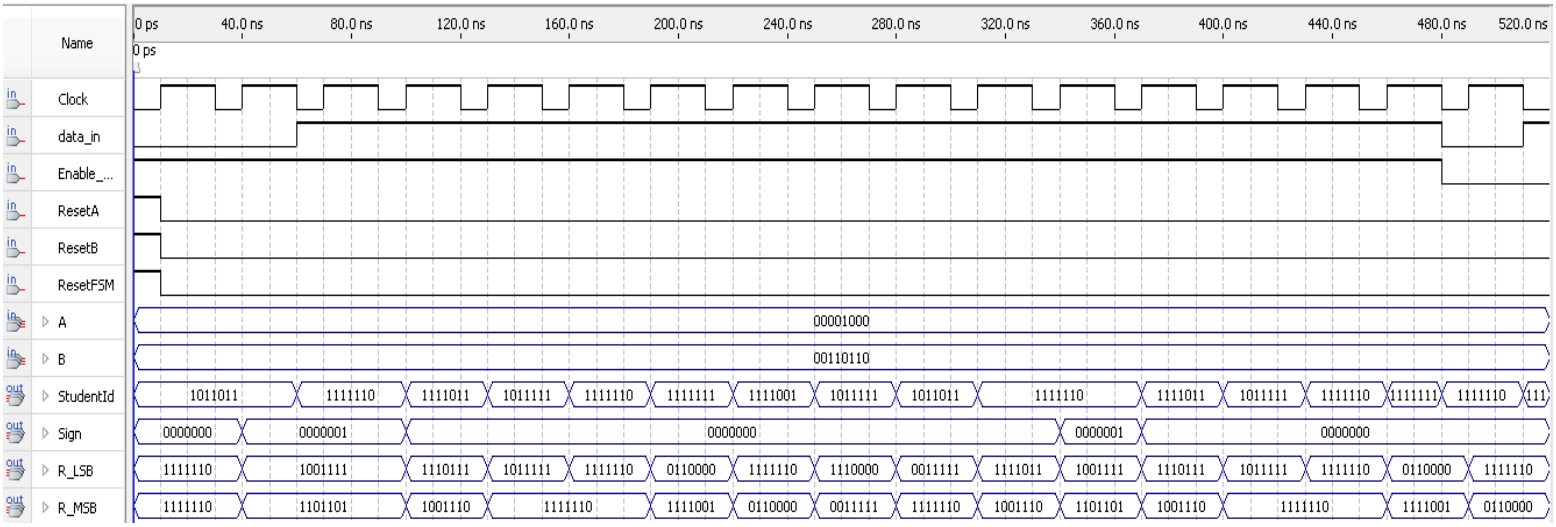

Figure 10.1 The timing diagram representing the correct outputs according to the seven segment display. The presented timing diagram showcases a propagation delay on the output of the SSEG as every current state transition outputs the operation on the next clock. The Sign for this processor is utilized to showcase if a negative has occurred. The output will be delayed by 30ns clock period since the operation is performed after with the input values prior to the 30ns. To showcase errors in the timing diagram, when the decoder is turned off at 490ns the resultant output also produces a "0" hexadecimal equivalent on the output. The correct student number starting from 5 can be displayed starting at 280ns due to the Mealy characteristics however, the student id is still delayed by 30ns due to the propagation delay [3].

General Processing Unit 3:

The final general processing unit requires simple changes compared to the GPU 1 and GPU 2. Since the desired output is a single hexadecimal representation confirmation for the assignment I, one of the SSEGs needs to be removed [2]. The timing diagram showcases the correct values for the assignment I because each output represents the corresponding yes or no valuation in the equivalent hexadecimal SSEG output compared to Table 6.0. The general processing Unit 3 utilizes the student ID input to procure data from the Mealy machine [2]. The Mealy machine will sequence through the student ID and current states sequentially for every rising edge of the clock and thus, the ALU will perform the conditional operations between them. Since the B value does not change within the ALU 1 operations, the timing diagram can be easily verified by comparing either decimal equivalents of the lower and higher bits of B compared to the student ID decimal equivalents [3]. The GPU 3 showcases that implemented systems can be modified and enhanced to a desired functionality. This can further be compared to the evolution of technology as CAD tools allow engineers to easily implement and enhance products constantly [3].

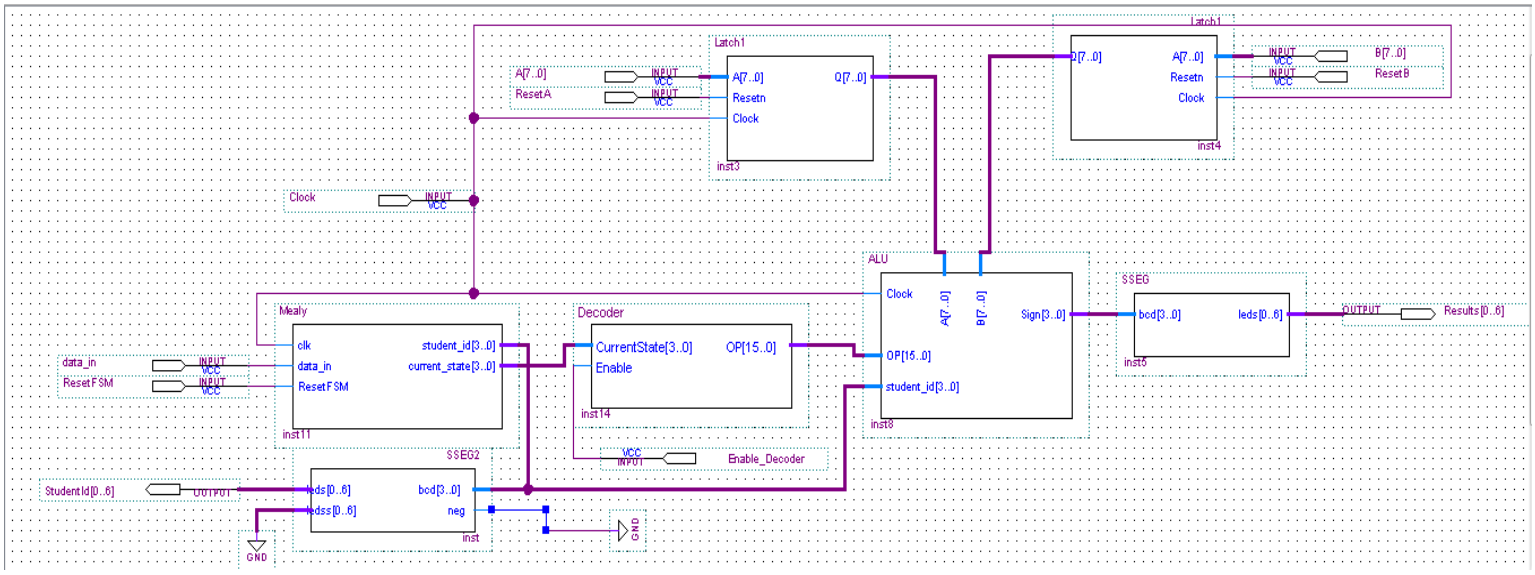

Figure 11.0 The schematic diagram representation of the general processing unit 3. This schematic diagram showcases slight differences among the GPU 1 and GPU 2. The GPU 3 only requires one SSEG to display the outputs. As well, the Mealy machine directly outputs the student ID on the external SSEG with a parallel connection to the ALU 3. The ALU 3 will then be able to perform the conditional operation amongst the storage values and the sequencing student ID's [2].

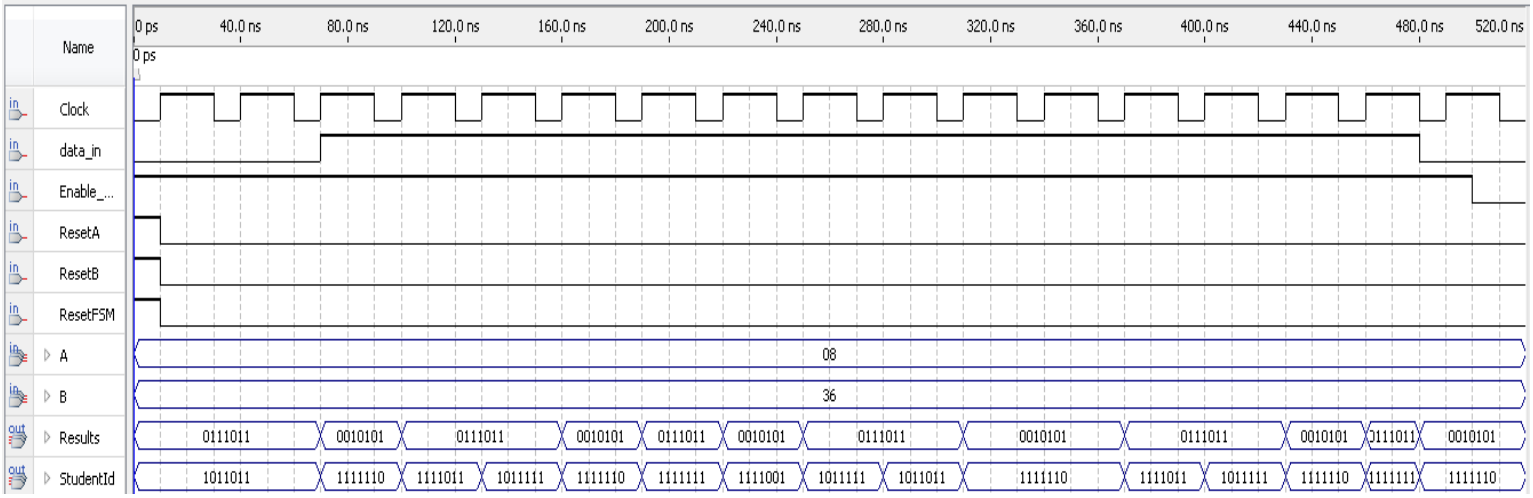

Figure 11.1 The timing diagram representing the correct outputs for GPU 3. The presented timing diagram showcases a propagation delay on the output of the SSEG. The input values were set as radixes of unsigned decimal numbers to showcase clarification when comparing the student ID from the Mealy machine to output results. The diagram also showcases a propagation delay as the output result is presented after a clock cycle. In order to showcase this the data set at 80ns was utilized. This was utilized for previous timing diagrams as well as an attempt to combat the delay. Error prevention is present at 480ns as the data value and enable fluctuates, the output then becomes 'n' [2].

Conclusion:

Overall, the final laboratory experiment of the general processing unit demonstrates the combination of material that was learned throughout the COE 328 course at Ryerson University [2]. The GPU consists of a combination of various primitive gates, sequential circuits, and combinational circuits [2]. Each component also demonstrates a sublevel circuit that demonstrates distinct complexities through its various functions such as the finite state machine [2]. The completed laboratory experiment showcases how crucial the cycle for engineering digital circuits is to develop a functional product without any defects [3]. The CAD tool Quartus by Altera demonstrates this process as the design process begins with a hardware code representation, a block diagram schematic representation and a testing of the project's functionality with the use of timing diagrams [1]. The timing diagram represents the functionality of the project as it tests whether any errors occurred or if components are not designed according to their expected functionality [3]. This can be translated into the GPU project as for many instances the control unit will not select the correct operations for the ALU. Timing diagrams also showcase propagation delays which may occur, in attempt to combat such issues additional changes to the waveform diagrams where presented. This is beneficial for the engineer because industry professionals can thoroughly test products before spending significant amounts of expenses and time manufacturing hardware that are faulty [3]. Nevertheless, the process of engineering the general process unit demonstrates how digital systems can be designed based upon abstract functional specifications.

References

[1] Z. Salcic, “Altera ® MAX + PLUS ® II 7.21 student edition programmable logic development software Altera university program.” .

[2] *COE328 - Digital Systems: Lab 6 - Design of a Simple General-Purpose Processor*, Ryerson University, Toronto, ON, Canada, 2019. Accessed on: Dec 1, 2020. [Online]. Available: https://www.ee.ryerson.ca/~courses/coe328/

[3] S. D. Brown and Z. G. Vranesic, *Fundamentals of Digital Logic with VHDL Design*. Boston: McGraw Hill Higher Education, 2009.

[4] *COE328 - Digital Systems: Lab 5 - VHDL for Sequential Circuits: Implementing a customized State Machine*, Ryerson University, Toronto, ON, Canada, 2019. Accessed on: Nov 16, 2020. [Online]. Available: https://www.ee.ryerson.ca/~courses/coe328/

[5] “D-type Flip Flop Counter or Delay Flip-flop,” *Basic Electronics Tutorials*, 29-Oct-2018. [Online]. Available: https://www.electronics-tutorials.ws/sequential/seq_4.html. [Accessed: 07-Dec-2020].

[6] P. Siddavaatam, "Lecture328_11_1" presented to Class, Ryerson University, Toronto, Ontario, Canada, November 10, 2020. [*PowerPoint*slides]. Available: https://courses.ryerson.ca/d2l/le/content/416908/viewContent/3102661/View