Ryerson University

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

Course Title:	**Digital Systems**
Course Number:	**COE328**
Semester/Year	**F2023**

Instructor:	**Reza Sedaghat**

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

: Submission Date	**December 4, 2023**
Due Date:	**December 4, 2023**

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
AbdulRahman	**Syed**	**-**	**04**	**S.A**
Rabbani	**Mohammed Omar**	**-**	**04**	**Omar R.**

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pd

Table of Contents:

Introduction:

Student Number Used: 5011—
A-value: A = 78_{16} = $0111\ 1000_2$
B-value: B = 34_{16} = $0011\ 0100_2$

The ultimate objective of this lab was to generate a simple General Purpose Processor (GPP) by applying the knowledge of both combinational and sequential circuits obtained from all previous labs [1]. The Block Diagram files demonstrate this purpose of a GPP, processing and controlling the units. Two Latches A & B store 8-bit inputs and follow by passing this data onto a clock cycle [1]. The control unit houses both the Finite State Machine (FSM) as well as a 4x16 Decoder, which outputs the current state from 0-8 as a 4-bit signal for the FSM, while the decoder selects 1 of 16-bits of microcode output respectively [1]. The Arithmetic Logic Unit (ALU) operates as a direct result from the inputs of the two latches as well as the selected microcode input, producing an 8-bit outcome in the form of two 4-bit numbers that must be displayed accordingly [1]. To view the results of the above operations, Seven Segment Displays were utilized to observe the resulting hexadecimal output, as well as a negative bit instance of the seven segment in case of negative output when subtraction is performed [1].

Components:

Latch 1 & 2:

The purpose of the latch is that it acts as a storage element for the General Purpose Processor [1]. It can store 8-bits of input and feed this data into the Arithmetic Logic Unit. Two latches are used to realize two sets of 8-bit input data, named A and B respectively.

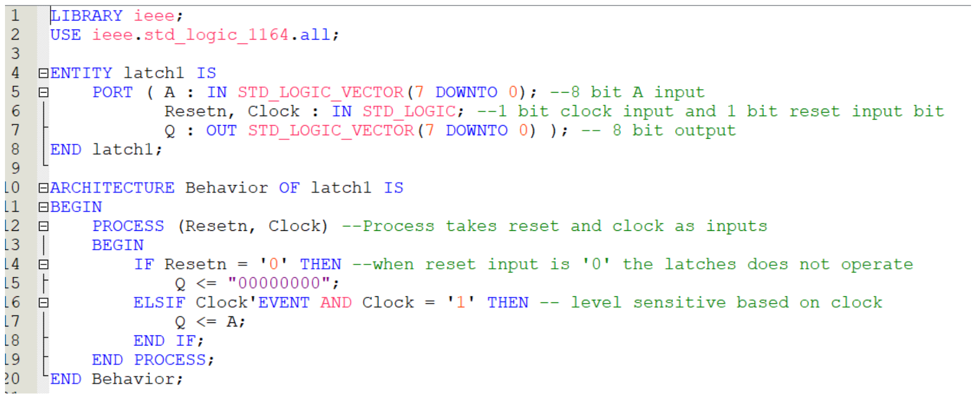

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

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

ARCHITECTURE Behavior OF latch1 IS
BEGIN
    PROCESS (Resetn, Clock) --Process takes reset and clock as inputs
    BEGIN
        IF Resetn = '0' THEN --when reset input is '0' the latches does not operate
            Q <= "00000000";
        ELSIF Clock'EVENT AND Clock = '1' THEN -- level sensitive based on clock
            Q <= A;
        END IF;
    END PROCESS;
END Behavior;
```

Figure 1: Latch 1 VHDL Code

The same code is used for latch2, the difference being, variable A is swapped with variable B, both storing values of 8-bit input.

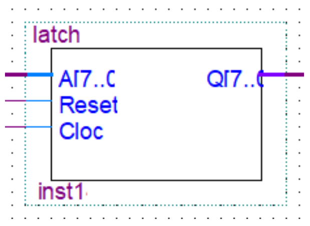


Figure 2: Latch 1 Block Symbol

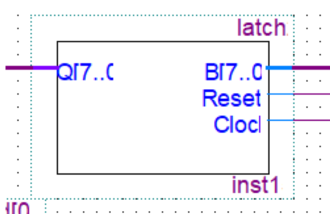


Figure 3: Latch 2 Block Symbol

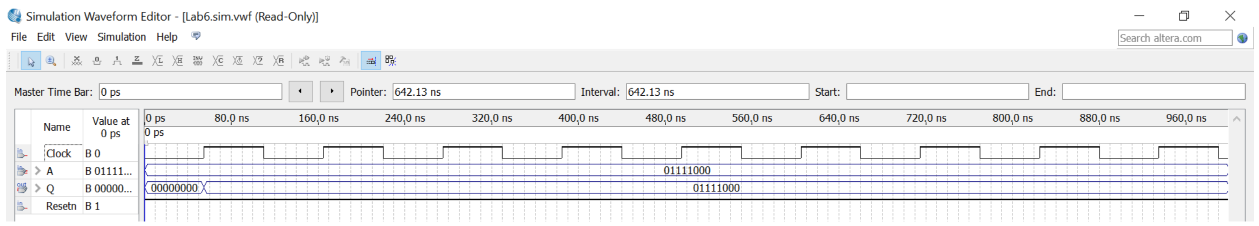


Figure 4: Latch 1 Waveform

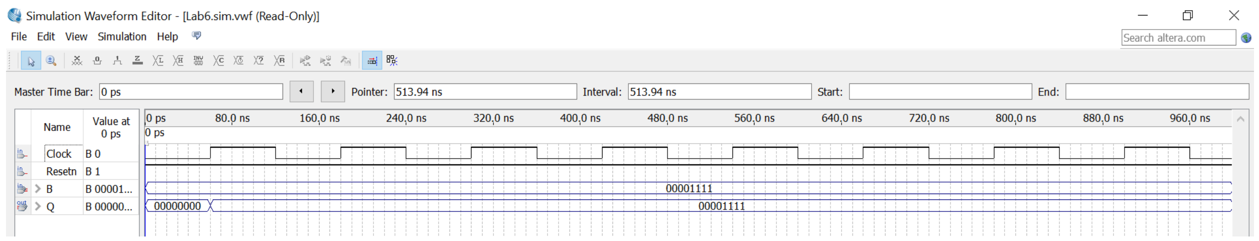


Figure 4: Latch 2 Waveform

Table 1. Truth table for both latches, with all instances of A[7..0] being swapped with B[7..0] for latch2.

Input	Input	Input	Output
Reset	Clock	A[7..0]	Q(t+1)
0	0	A[7..0]	0000 0000
0	1	A[7..0]	0000 0000
1	0	A[7..0]	Q(t)
1	1	A[7..0]	A[7..0]

4 to 16 Decoder:

The 4x16 decoder has been created in a way where it gets fed a 4-bit input from the Finite State Machine output, and produces a 16-bit microcode, all while the enable signal is set to active high. Every possible combination of input from the FSM is fed to a different microcode, allowing the decoder to serve a purpose as a selector for the ALU. Cycling through each state of the FSM causes the microcode output to be cycled as well, which results in the ALU also changing from one of its functions to the next [1].

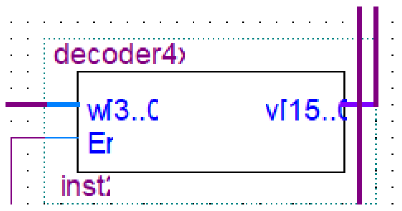
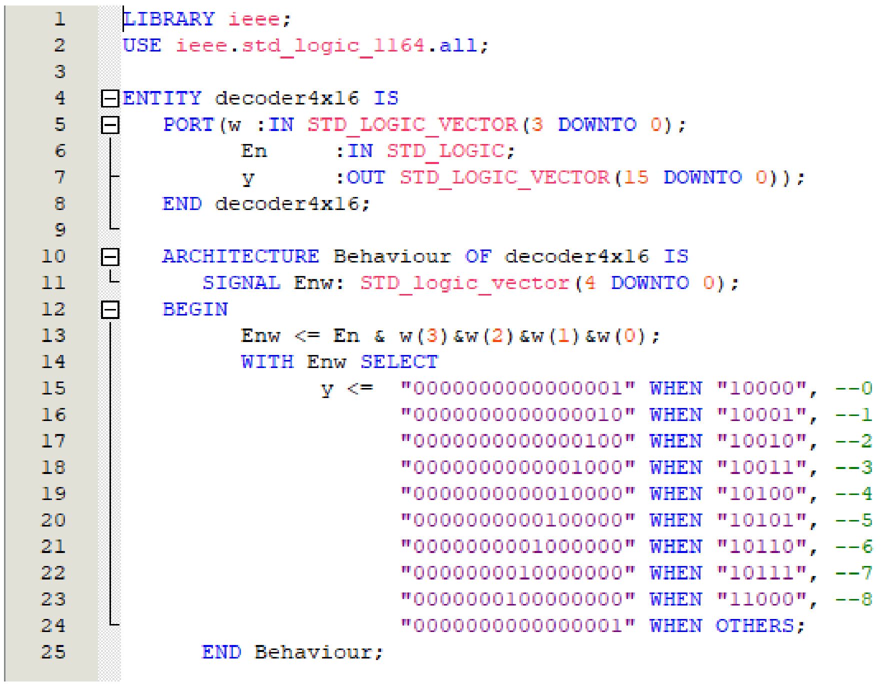


Figure 6: Latch 1 Block Symbol



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

ENTITY decoder4x16 IS
   PORT(w :IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         En     :IN STD_LOGIC;
         y      :OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
   END decoder4x16;

   ARCHITECTURE Behaviour OF decoder4x16 IS
      SIGNAL Enw: STD_logic_vector(4 DOWNTO 0);
   BEGIN
         Enw <= En & w(3)&w(2)&w(1)&w(0);
         WITH Enw SELECT
               y <=  "0000000000000001" WHEN "10000", --0
                     "0000000000000010" WHEN "10001", --1
                     "0000000000000100" WHEN "10010", --2
                     "0000000000001000" WHEN "10011", --3
                     "0000000000010000" WHEN "10100", --4
                     "0000000000100000" WHEN "10101", --5
                     "0000000001000000" WHEN "10110", --6
                     "0000000010000000" WHEN "10111", --7
                     "0000000100000000" WHEN "11000", --8
                     "0000000000000001" WHEN OTHERS;
      END Behaviour;
```

Figure 7: 4 to 16 decoder VHDL Code

Table 2. Truth table for the 4x16 decoder.

Inputs				Outputs															
w3	w2	w1	w0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11	Y12	Y13	Y14	Y15
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	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	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d

1	0	1	0	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d
1	0	1	1	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d
1	1	0	0	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d
1	1	0	1	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d
1	1	1	0	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d
1	1	1	1	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d

This decoder is programmed to activate one among sixteen potential outputs determined by a 4-bit input. The code is tailored to cater to only nine specific scenarios, addressing input sequences ranging from "0000" to "1000" to meet the lab's requirements. Outputs beyond these nine are assigned as dont cares, signifying their non-importance for the intended use-case.

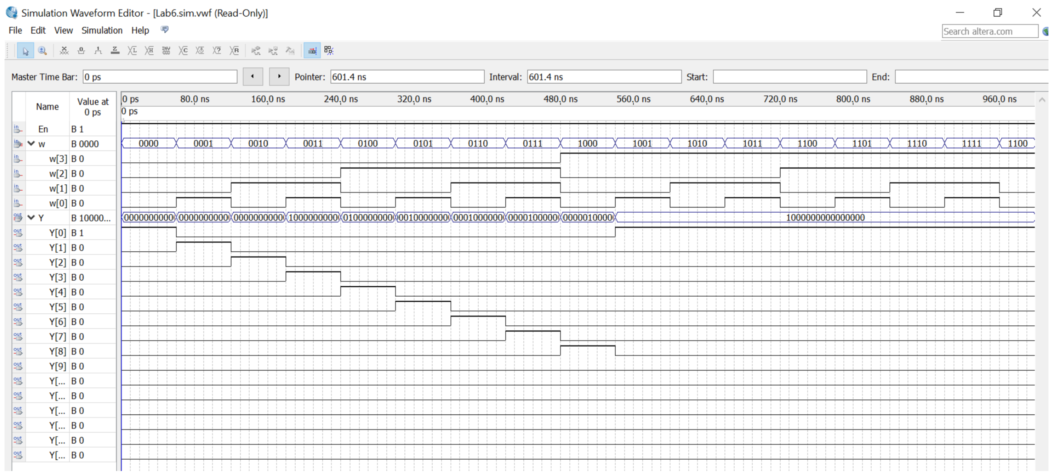


Figure 8: 4 to 16 decoder waveform

Finite State Machine (FSM):

The FSM is used to establish the process of an up counter in this lab. When a rising edge of the clock cycle is encountered, as long as the conditions of the enable being set to active high as well as the data_in being set to 1 are met, the FSM will count up starting from 0-8 [1]. These digits are representative of the states of the FSM, hence they can be sent to the 4x16 decoder as a 4-bit signal, while another 4-bit signal is sent to a seven segment display, representing the student ID number.

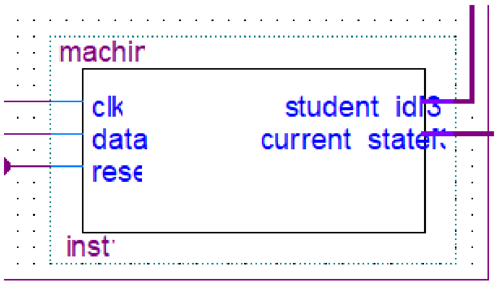
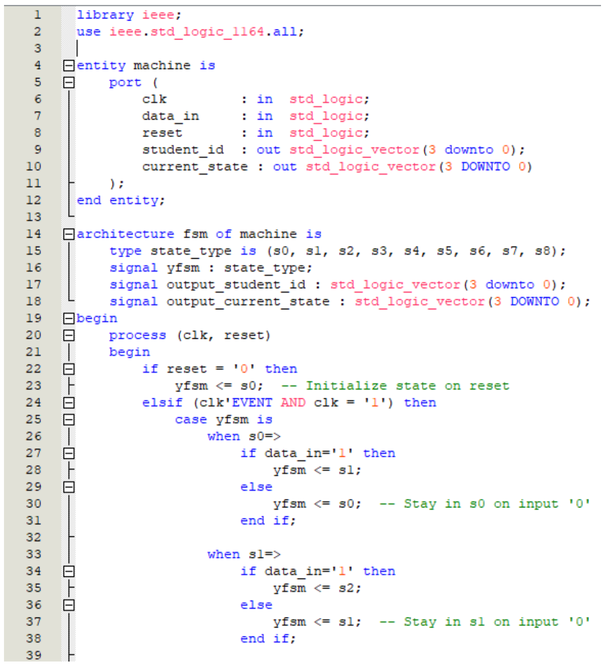
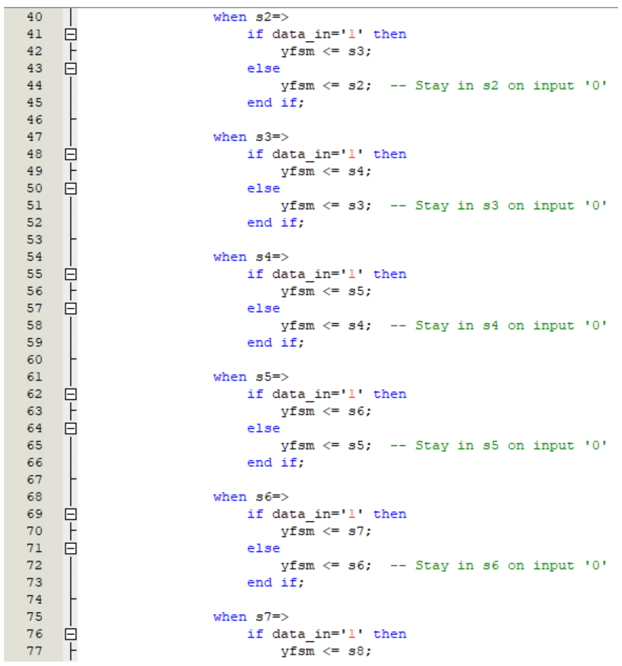


Figure 9: FSM Block Symbol



```
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 entity;

architecture fsm of machine is
    type state_type is (s0, s1, s2, s3, s4, s5, s6, s7, s8);
    signal yfsm : state_type;
    signal output_student_id : std_logic_vector(3 downto 0);
    signal output_current_state : std_logic_vector(3 DOWNTO 0);
begin
    process (clk, reset)
    begin
        if reset = '0' then
            yfsm <= s0;  -- Initialize state on reset
        elsif (clk'EVENT AND clk = '1') then
            case yfsm is
                when s0=>
                    if data_in='1' then
                        yfsm <= s1;
                    else
                        yfsm <= s0;  -- Stay in s0 on input '0'
                    end if;

                when s1=>
                    if data_in='1' then
                        yfsm <= s2;
                    else
                        yfsm <= s1;  -- Stay in s1 on input '0'
                    end if;

```



```
                when s2=>
                    if data_in='1' then
                        yfsm <= s3;
                    else
                        yfsm <= s2;  -- Stay in s2 on input '0'
                    end if;

                when s3=>
                    if data_in='1' then
                        yfsm <= s4;
                    else
                        yfsm <= s3;  -- Stay in s3 on input '0'
                    end if;

                when s4=>
                    if data_in='1' then
                        yfsm <= s5;
                    else
                        yfsm <= s4;  -- Stay in s4 on input '0'
                    end if;

                when s5=>
                    if data_in='1' then
                        yfsm <= s6;
                    else
                        yfsm <= s5;  -- Stay in s5 on input '0'
                    end if;

                when s6=>
                    if data_in='1' then
                        yfsm <= s7;
                    else
                        yfsm <= s6;  -- Stay in s6 on input '0'
                    end if;

                when s7=>
                    if data_in='1' then
                        yfsm <= s8;
```

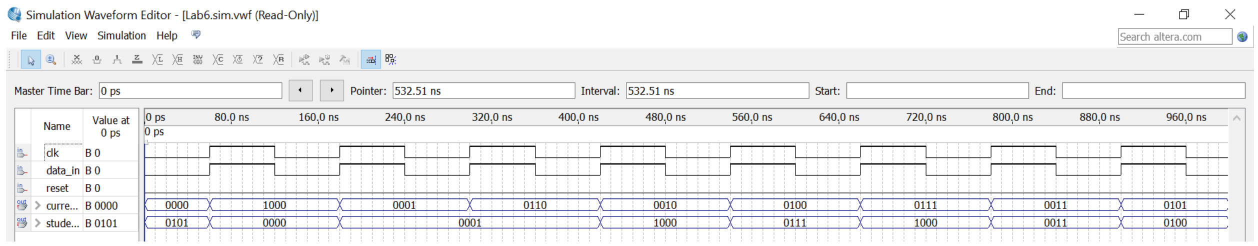
```
-- Implement the Moore or Mealy logic here
process (yfsm, data_in) -- data_in if reqd only
begin
    case yfsm is
        when s0=> --s5 points to s0
        student_id <= "0101"; --5
        current_state <= "0000";  -- current state s0

        when s1=> --s1 points to s8
        student_id <= "0000"; --0
        current_state <= "0001"; -- current state s1

        when s2=> --s8 points to s1
        student_id <= "0001"; --1
        current_state <= "0010"; -- current state s2

        when s3=> --s1 points to s6
        student_id <= "0001"; --1
        current_state <= "0011"; -- current state s3

        when s4=> --s6 points to s2
        student_id <= "1000"; --8
        current_state <= "0100"; -- current state s4
```

```
    when s5=> --s2 points to s4
    student_id <= "0111"; --7
    current_state <= "0101"; -- current state s5

    when s6=> --s4 points to s7
    student_id <= "1000"; --8
    current_state <= "0110"; -- current state s6

    when s7=> --s7 points to s3
    student_id <= "0011"; --3
    current_state <= "0111"; -- current state s7

    when s8=> --s3 points to s5
    student_id <= "0100"; --4
    current_state <= "1000"; -- current state s8

end case;
end process;
end architecture;
```

Figure 10: FSM VHDL Code

Table 3. State assignment table for the Finite State Machine. Assumes Reset is set to "1".

Present State	Next State		Student Id
	Data in=0	Data in=1	
$y_3y_2y_1y_0$	$Y_3Y_2Y_1Y_0$	$Y_3Y_2Y_1Y_0$	z
0000	0000	0001	5
0001	0001	0010	0
0010	0010	0011	1
0011	0011	0100	1
0100	0100	0101	8
0101	0101	0110	7
0110	0110	0111	8
0111	0111	1000	3
1000	1000	0000	4


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

Master Time Bar: 0 ps Pointer: 532.51 ns Interval: 532.51 ns Start: End:

Name	Value at 0 ps
clk	B 0
data_in	B 0
reset	B 0
curre...	B 0000
stude...	B 0101

File Edit View Simulation Help
Search altera.com
0 ps 80,0 ns 160,0 ns 240,0 ns 320,0 ns 400,0 ns 480,0 ns 560,0 ns 640,0 ns 720,0 ns 800,0 ns 880,0 ns 960,0 ns
0000 1000 0001 0110 0010 0100 0111 0011 0101
0101 0000 0001 1000 0111 1000 0011 0100

Figure 11: FSM waveform

ALU 1:

The objective of the Arithmetic Logic Unit is to calculate boolean functions based on two sets of 8-bit inputs, A and B [1]. Depending on the microcode input received from the 4x16 decoder, the ALU performs one of the 9 assigned functions [1]. Since it is known that the 4x16 decoder outputs a certain microcode based on input received from the FSM's current state output, each state of the FSM can be transitioned from one to the next when the positive edge of the clock is established. Hence , the ALU is a sequential circuit on its own due to its reliance on a clock cycle; it selects a function based on user input, controlled by the clock's position in an instance of time [1].

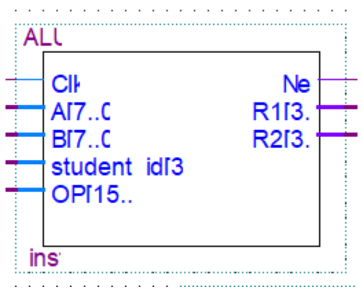
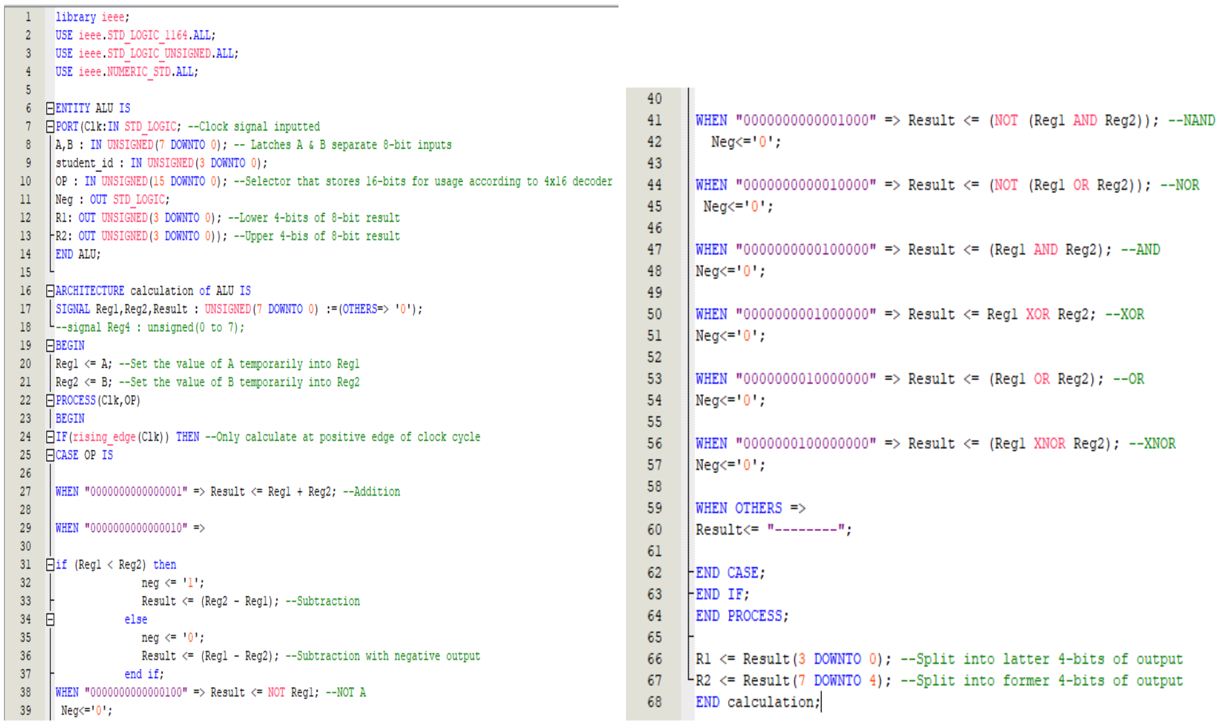


Figure 12: ALU 1 Block Symbol



```vhdl
library ieee;
USE ieee.STD_LOGIC_1164.ALL;
USE ieee.STD_LOGIC_UNSIGNED.ALL;
USE ieee.NUMERIC_STD.ALL;

ENTITY ALU IS
PORT(Clk:IN STD_LOGIC; --Clock signal inputted
A,B : IN UNSIGNED(7 DOWNTO 0); -- Latches A & B separate 8-bit inputs
student_id : IN UNSIGNED(3 DOWNTO 0);
OP : IN UNSIGNED(15 DOWNTO 0); --Selector that stores 16-bits for usage according to 4x16 decoder
Neg : OUT STD_LOGIC;
R1: OUT UNSIGNED(3 DOWNTO 0); --Lower 4-bits of 8-bit result
R2: OUT UNSIGNED(3 DOWNTO 0)); --Upper 4-bis of 8-bit result
END ALU;

ARCHITECTURE calculation of ALU IS
SIGNAL Reg1,Reg2,Result : UNSIGNED(7 DOWNTO 0) :=(OTHERS=> '0');
--signal Reg4 : unsigned(0 to 7);
BEGIN
Reg1 <= A; --Set the value of A temporarily into Reg1
Reg2 <= B; --Set the value of B temporarily into Reg2
PROCESS(Clk,OP)
BEGIN
IF(rising_edge(Clk)) THEN --Only calculate at positive edge of clock cycle
CASE OP IS

WHEN "0000000000000001" => Result <= Reg1 + Reg2; --Addition

WHEN "0000000000000010" =>

if (Reg1 < Reg2) then
                neg <= '1';
                Result <= (Reg2 - Reg1); --Subtraction
            else
                neg <= '0';
                Result <= (Reg1 - Reg2); --Subtraction with negative output
            end if;
WHEN "0000000000000100" => Result <= NOT Reg1; --NOT A
 Neg<='0';

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

WHEN "0000000000010000" => Result <= (NOT (Reg1 OR Reg2)); --NOR
 Neg<='0';

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

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

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

WHEN "0000000100000000" => Result <= (Reg1 XNOR Reg2); --XNOR
Neg<='0';

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

END CASE;
END IF;
END PROCESS;

R1 <= Result(3 DOWNTO 0); --Split into latter 4-bits of output
R2 <= Result(7 DOWNTO 4); --Split into former 4-bits of output
END calculation;
```

Figure 13: ALU1 VHDL Code

Table 4. The specific ALU functions used within Problem 1 [1].

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

Table 5. Purpose of all inputs and outputs.

Signal	Type	Purpose
Clk	Input	On positive rising edge of the signal, activates ALU functions
A[7..0]	Input	An 8-bit input that the ALU uses to carry out its functions
B[7..0]	Input	An 8-bit input that the ALU uses to carry out its functions
OP[15..0]	Input	A selector for the different functions used by the ALU
R1[3..0]	Output	Stores the last 4-bits of the output from the ALU function
R2[3..0]	Output	Stores the first 4-bits of the output from the ALU function
Neg	Output	Signals that the output of a function requires a negative value

Table 6. Determined Outputs for ALU1.

	Functions	Function Outputs	Hexadecimal	Student ID
1	A+B	1010 1100	AC	5
2	A-B	0010 0010	44	0
3	$\overline{A}$	1000 0111	87	1
4	$\overline{A \cdot B}$	1100 1111	CF	1
5	$\overline{A + B}$	1000 0011	83	8
6	$A \cdot B$	0011 0000	30	7
7	XOR	01001100	4C	8
8	$A + B$	0111 1100	7C	3
9	XNOR	10110011	B3	4

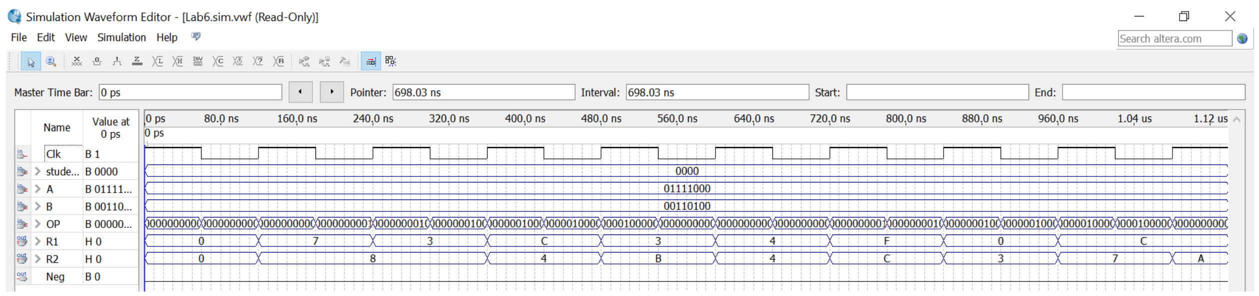


Figure 14: ALU 1 waveform

The waveforms show the signals for the clock , student ID, and the inputs A and B, as well as the operation code, and the resulting outputs R1, R2, and Neg. The changes in the output signals R1 and R2 occur after a certain delay following the rising edge of the Clk signal, indicating the processing time of the ALU to execute the given operation. The delay, also known as propagation delay, is the time taken for the inputs to be computed and reflected in the outputs. This delay is crucial for timing analysis and synchronization in digital circuits to ensure correct operation and data integrity throughout the system. One way to solve the delay is to set your clock cycle to a smaller degree instead of 60ns in the waveform setting to 10ns solves this issue. For example, taking the last iteration of the waveform whose output is "AC" its corresponding function is A+B because of the discussed delay, it's the reason why it's at the end of the waveform. Comparing the remaining waveform outputs to the VHDL Code, the outputs match the determined outputs.

Combined ALU 1

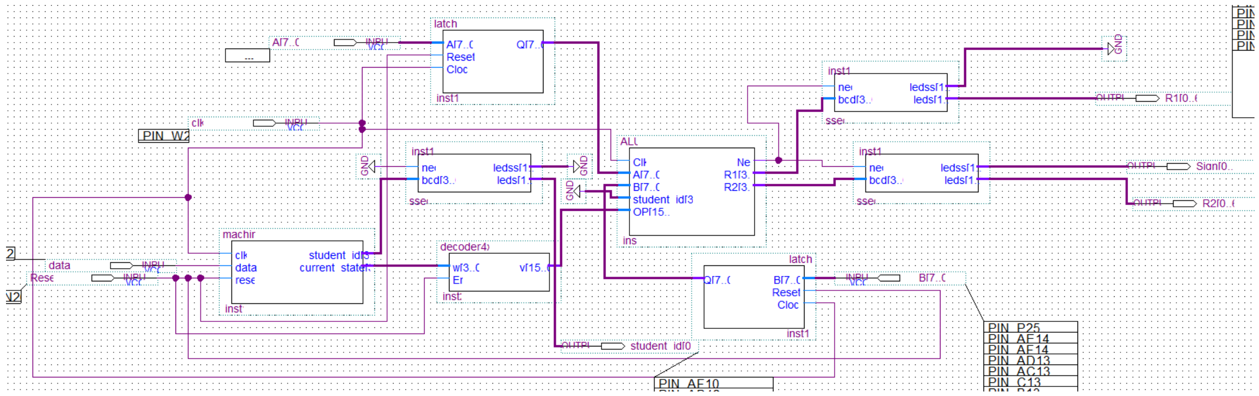
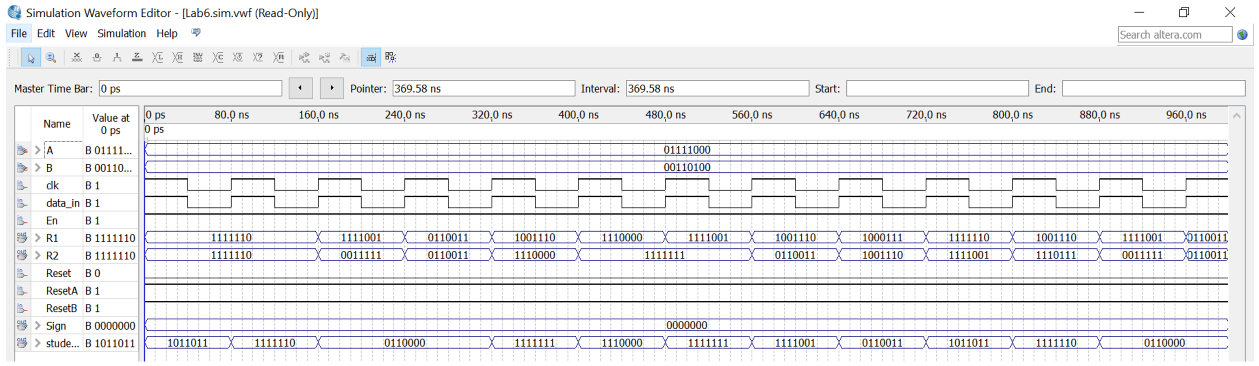


Figure 15: Combined ALU 1 Block Diagram


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

Master Time Bar: 0 ps   Pointer: 369.58 ns   Interval: 369.58 ns   Start: End:

Name	Value at 0 ps
A	B 01111...
B	B 00110...
clk	B 1
data_in	B 1
En	B 1
R1	B 1111110
R2	B 1111110
Reset	B 0
ResetA	B 1
ResetB	B 1
Sign	B 0000000
stude...	B 1011011

Figure 16: Combined ALU 1 waveform

This final block diagram file incorporates all the components mentioned above in the components section of this lab report. As a result, a fully working GPP is therefore established, which can be implemented onto an FPGA board for testing. The above circuit also functions as a sequential one, seeing as there is only one source for the clock.

Problem 2:

ALU2:

The objective of the Arithmetic Logic Unit (ALU) in this section is to execute a specific boolean function from a bank of 9 total functions, based on two 8-bit inputs, A and B. The only true difference here is that the 9 functions are different when compared to ALU1, as they follow the functions that are shown in Table 8 below. Everything else remains the same, from the I/O of the GPP circuit as well as all other block diagram components within the circuit itself.

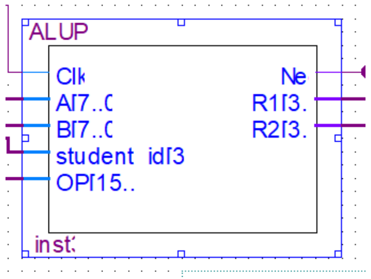
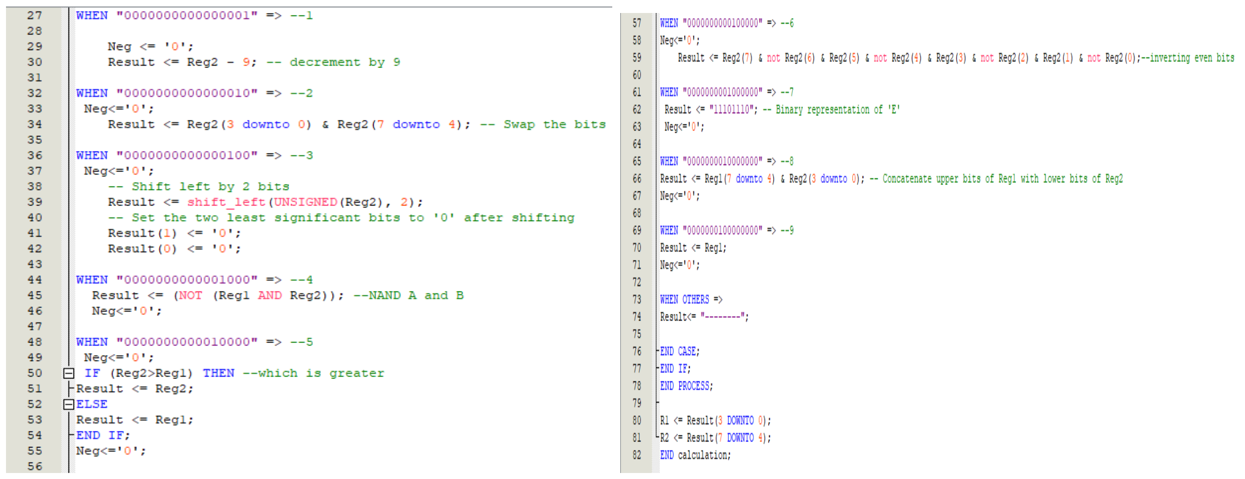


Figure 17: ALU 2 Block Symbol



```
WHEN "0000000000000001" => --1

    Neg <= '0';
    Result <= Reg2 - 9; -- decrement by 9

WHEN "0000000000000010" => --2
 Neg<='0';
    Result <= Reg2(3 downto 0) & Reg2(7 downto 4); -- Swap the bits

WHEN "0000000000000100" => --3
 Neg<='0';
    -- Shift left by 2 bits
    Result <= shift_left(UNSIGNED(Reg2), 2);
    -- Set the two least significant bits to '0' after shifting
    Result(1) <= '0';
    Result(0) <= '0';

WHEN "0000000000001000" => --4
  Result <= (NOT (Reg1 AND Reg2)); --NAND A and B
  Neg<='0';

WHEN "0000000000010000" => --5
 Neg<='0';
 IF (Reg2>Reg1) THEN --which is greater
Result <= Reg2;
ELSE
Result <= Reg1;
END IF;
Neg<='0';

WHEN "0000000000100000" => --6
Neg<='0';
   Result <= Reg2(7) & not Reg2(6) & Reg2(5) & not Reg2(4) & Reg2(3) & not Reg2(2) & Reg2(1) & not Reg2(0);--inverting even bits

WHEN "0000000001000000" => --7
 Result <= "11101110"; -- Binary representation of 'E'
 Neg<='0';

WHEN "0000000010000000" => --8
Result <= Reg1(7 downto 4) & Reg2(3 downto 0); -- Concatenate upper bits of Reg1 with lower bits of Reg2
Neg<='0';

WHEN "0000000100000000" => --9
Result <= Reg1;
Neg<='0';

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

END CASE;
END IF;
END PROCESS;

R1 <= Result(3 DOWNTO 0);
R2 <= Result(7 DOWNTO 4);
END calculation;
```

Figure 18: VHDL code for second instance of ALU component

Table 7. Purpose of all inputs and outputs.

Signal	Type	Purpose
Clk	Input	On positive rising edge of the signal, activates ALU functions
A[7..0]	Input	An 8-bit input that the ALU uses to carry out its functions
B[7..0]	Input	An 8-bit input that the ALU uses to carry out its functions
OP[15..0]	Input	A selector for the different functions used by the ALU
R1[3..0]	Output	Stores the last 4-bits of the output from the ALU function

R2[3..0]	Output	Stores the first 4-bits of the output from the ALU function
Neg	Output	Signals that the output of a function requires a negative value

Table 8. Determined Outputs for Problem 2 (F).

	Operation/ Function	Function Outputs	Hex	Student ID
1	1 Decrement B by 9	0010 1011	2B	5
2	Swap the lower and upper 4 bits of B	0100 0011	43	0
3	Shift A to left by 2 bits, input bit = 0 (SHL)	1110 0000	E0	1
4	Produce the result of NANDing A and B	1100 1111	CF	1
5	Find the greater value of A and B and produce the results (Max(A,B))	0111 1000	78	8
6	Invert the even bits of B	0110 0001	61	7
7	Produce null on the output	1110 1110	EE	8
8	Replace the upper four bits of B by upper four bits of A	0111 0100	74	3
9	Show A on the output	0111 1000	78	4

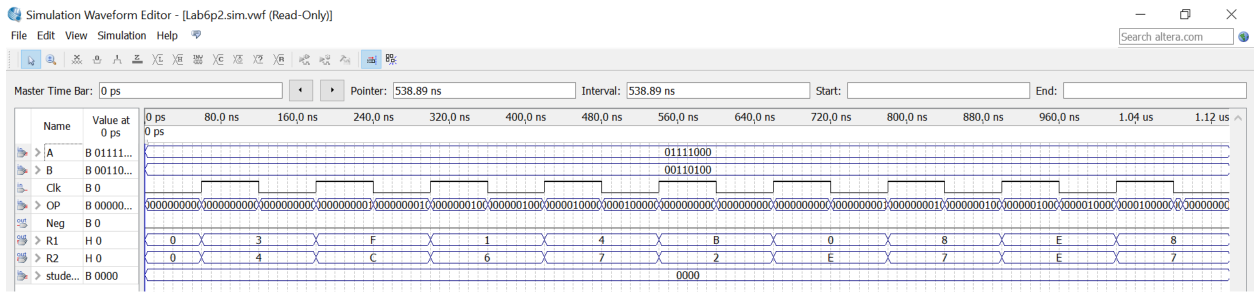

Simulation Waveform Editor - [Lab6p2.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 538.89 ns
Interval: 538.89 ns
Start:
End:
Name
Value at 0 ps
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns 1.04 us 1.12 us
A B 01111... 01111000
B B 00110... 00110100
Clk B 0
OP B 00000...
Neg B 0
R1 H 0 0 3 F 1 4 B 0 8 E 8
R2 H 0 0 4 C 6 7 2 E 7 E 7
stude... B 0000 0000

Figure 19: ALU 2 waveform

Similar to ALU 1 the delay in the ALU 2 waveform represents the propagation delay, which is the time needed for the ALU to process the inputs and produce outputs. This delay is a standard aspect of digital circuit operation, reflecting the time signals take to pass through the ALU's components. It's critical for maintaining data synchronization within the system. Comparing the waveform outputs to the VHDL Code, the outputs match the determined outputs.

Combined ALU 2

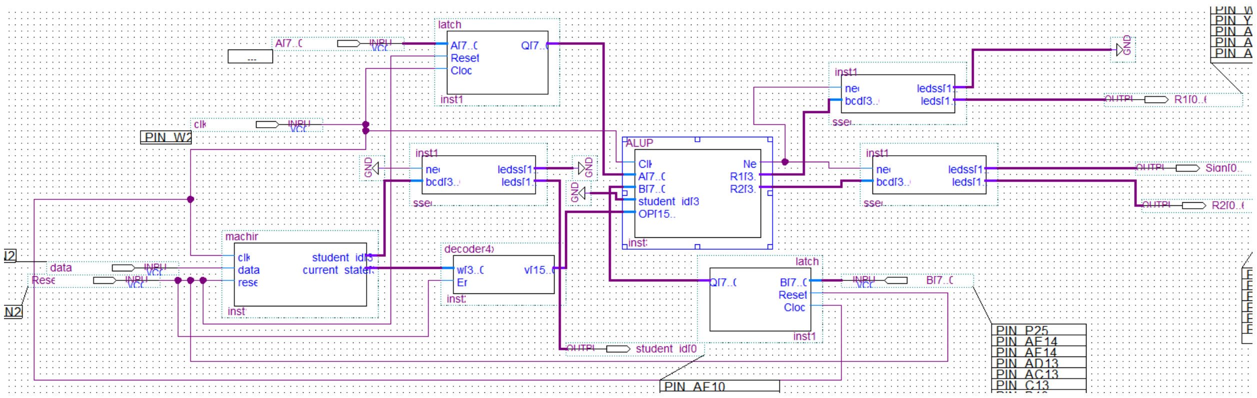
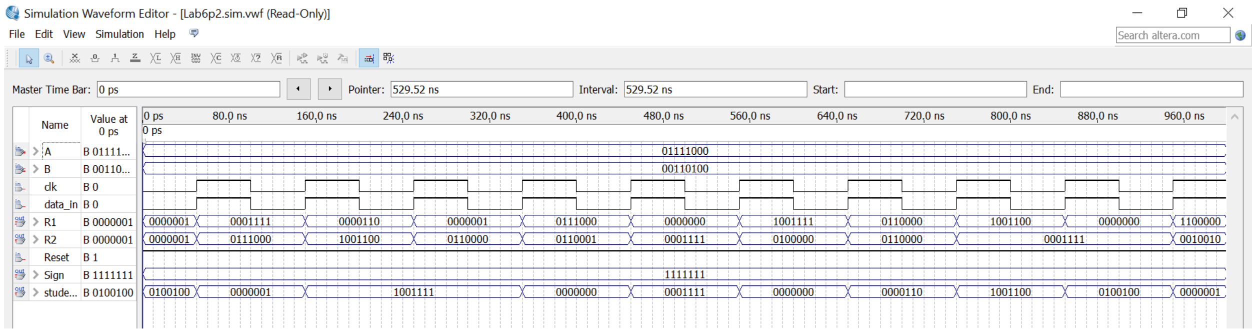


Figure 20: Combined ALU 2 Block Diagram


Simulation Waveform Editor - [Lab6p2.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com

Master Time Bar: 0 ps Pointer: 529.52 ns Interval: 529.52 ns Start: End:

```
Name     Value at 0 ps   0 ps  80.0 ns  160.0 ns  240.0 ns  320.0 ns  400.0 ns  480.0 ns  560.0 ns  640.0 ns  720.0 ns  800.0 ns  880.0 ns  960.0 ns
A        B 01111...      01111000
B        B 00110...      00110100
clk      B 0
data_in  B 0
R1       B 0000001       0000001 | 0001111 | 0000110 | 0000001 | 0111000 | 0000000 | 1001111 | 0110000 | 1001100 | 0000000 | 1100000
R2       B 0000001       0000001 | 0111000 | 1001100 | 0110000 | 0110001 | 0001111 | 0100000 | 0110000 | 0001111 | 0010010
Reset    B 1
Sign     B 1111111       1111111
stude... B 0100100       0100100 | 0000001 | 1001111 | 0000000 | 0001111 | 0000000 | 0000110 | 1001100 | 0100100 | 0000001
```

Figure 21: Combined ALU 2 waveform *Note outputs are inverted

When compared to the complete block diagram for ALU1, it can clearly be observed that the remainder of the circuit stays identical, with only the Arithmetic Logic Unit changing internally in terms of what functions it can cycle through.

Problem 3:

ALU3:

The objective of the Arithmetic Logic Unit (ALU) in this section has been changed slightly when compared to both ALU1 and ALU2. In order to analyze the student number digits from the FSM output, a modified seven segment will be used to show a “y” if one of the 2 digits of A is less than the current student number FSM output [1]. Otherwise, an “n” will be displayed on the modified seven segments [1]. This was accomplished by altering the input and output connections of the ALU, where R2 and Neg were grounded as they had no function in the ALU, hence only a singular output of R1 ended up being inserted into the new seven segment. By comparing the A-value of 78_{16} converted to binary with the binary equivalent of one single digit of the student_id at a time from the FSM, it was determined if the student_id was greater than either 7 or 8, or if its value was less, displaying the “y” or “n” respectively.

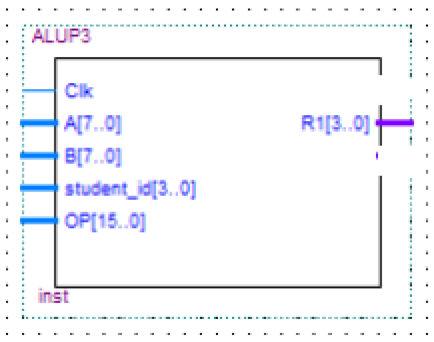
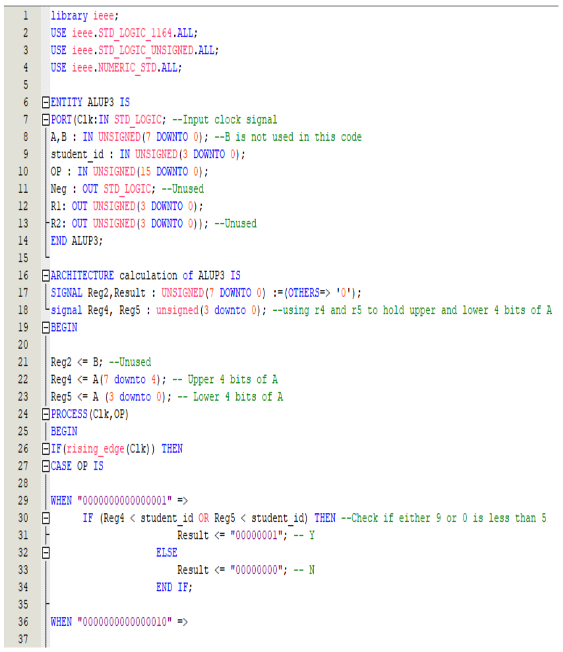
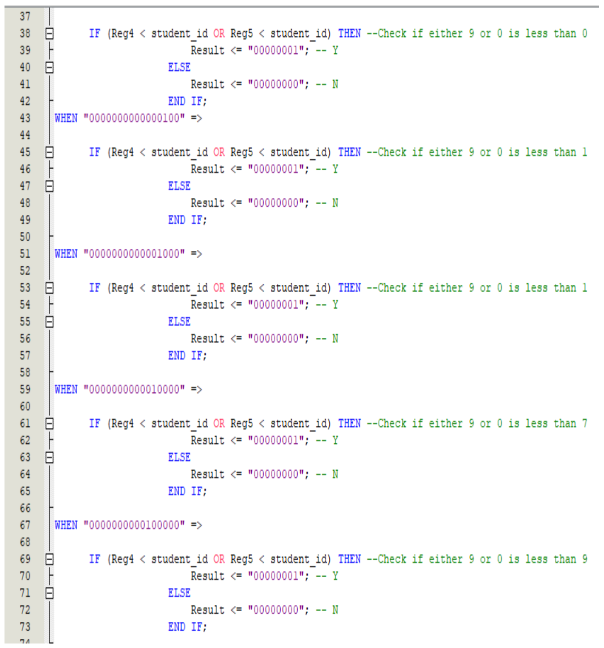


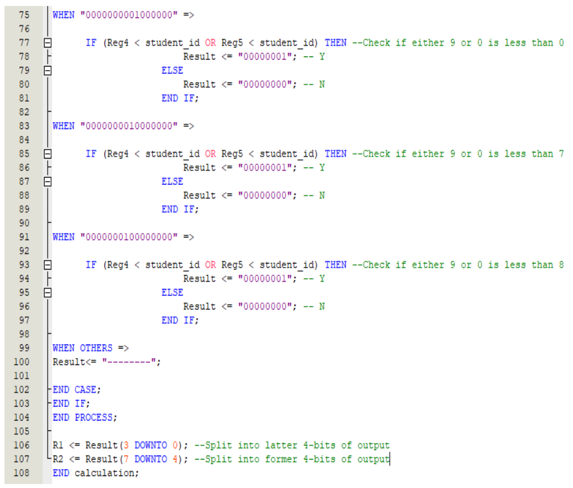
Figure 22: ALU 3 Block Symbol



```vhdl
library ieee;
USE ieee.STD_LOGIC_1164.ALL;
USE ieee.STD_LOGIC_UNSIGNED.ALL;
USE ieee.NUMERIC_STD.ALL;

ENTITY ALUP3 IS
PORT(Clk:IN STD_LOGIC; --Input clock signal
A,B : IN UNSIGNED(7 DOWNTO 0); --B is not used in this code
student_id : IN UNSIGNED(3 DOWNTO 0);
OP : IN UNSIGNED(15 DOWNTO 0);
Neg : OUT STD_LOGIC; --Unused
R1: OUT UNSIGNED(3 DOWNTO 0);
R2: OUT UNSIGNED(3 DOWNTO 0)); --Unused
END ALUP3;

ARCHITECTURE calculation of ALUP3 IS
SIGNAL Reg2,Result : UNSIGNED(7 DOWNTO 0) :=(OTHERS=> '0');
signal Reg4, Reg5 : unsigned(3 downto 0); --using r4 and r5 to hold upper and lower 4 bits of A
BEGIN

Reg2 <= B; --Unused
Reg4 <= A(7 downto 4); -- Upper 4 bits of A
Reg5 <= A (3 downto 0); -- Lower 4 bits of A
PROCESS(Clk,OP)
BEGIN
IF(rising_edge(Clk)) THEN
CASE OP IS

WHEN "0000000000000001" =>
      IF (Reg4 < student_id OR Reg5 < student_id) THEN --Check if either 9 or 0 is less than 5
                  Result <= "00000001"; -- Y
               ELSE
                  Result <= "00000000"; -- N
               END IF;

WHEN "0000000000000010" =>

```



```vhdl

      IF (Reg4 < student_id OR Reg5 < student_id) THEN --Check if either 9 or 0 is less than 0
                  Result <= "00000001"; -- Y
               ELSE
                  Result <= "00000000"; -- N
               END IF;
WHEN "0000000000000100" =>

      IF (Reg4 < student_id OR Reg5 < student_id) THEN --Check if either 9 or 0 is less than 1
                  Result <= "00000001"; -- Y
               ELSE
                  Result <= "00000000"; -- N
               END IF;

WHEN "0000000000001000" =>

      IF (Reg4 < student_id OR Reg5 < student_id) THEN --Check if either 9 or 0 is less than 1
                  Result <= "00000001"; -- Y
               ELSE
                  Result <= "00000000"; -- N
               END IF;

WHEN "0000000000010000" =>

      IF (Reg4 < student_id OR Reg5 < student_id) THEN --Check if either 9 or 0 is less than 7
                  Result <= "00000001"; -- Y
               ELSE
                  Result <= "00000000"; -- N
               END IF;

WHEN "0000000000100000" =>

      IF (Reg4 < student_id OR Reg5 < student_id) THEN --Check if either 9 or 0 is less than 9
                  Result <= "00000001"; -- Y
               ELSE
                  Result <= "00000000"; -- N
               END IF;
```



```vhdl
WHEN "0000000001000000" =>

      IF (Reg4 < student_id OR Reg5 < student_id) THEN --Check if either 9 or 0 is less than 0
                  Result <= "00000001"; -- Y
               ELSE
                  Result <= "00000000"; -- N
               END IF;

WHEN "0000000010000000" =>

      IF (Reg4 < student_id OR Reg5 < student_id) THEN --Check if either 9 or 0 is less than 7
                  Result <= "00000001"; -- Y
               ELSE
                  Result <= "00000000"; -- N
               END IF;

WHEN "0000000100000000" =>

      IF (Reg4 < student_id OR Reg5 < student_id) THEN --Check if either 9 or 0 is less than 8
                  Result <= "00000001"; -- Y
               ELSE
                  Result <= "00000000"; -- N
               END IF;

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

END CASE;
END IF;
END PROCESS;

R1 <= Result(3 DOWNTO 0); --Split into latter 4-bits of output
R2 <= Result(7 DOWNTO 4); --Split into former 4-bits of output
END calculation;
```

Figure 23: VHDL code for third instance of ALU component

Table 9. Determined Outputs for Problem 3 (F).

Yes or No	Student ID
n	5
n	0
n	1
n	1
y	8
n	7
y	8
n	3

Table 10. Purpose of all inputs and outputs.

Signal	Type	Purpose
Clk	Input	On positive rising edge of the signal, activates ALU functions
A[7..0]	Input	An 8-bit input that the ALU uses to carry out its functions
B[7..0]	Input	An 8-bit input that the ALU uses to carry out its functions
student_id[3..0]	Input	Retains the 4-bit binary value of the current student ID digit
OP[15..0]	Input	A selector for the different functions used by the ALU
R1[3..0]	Output	Generates "y" or "n" based on if condition of student ID < A is met

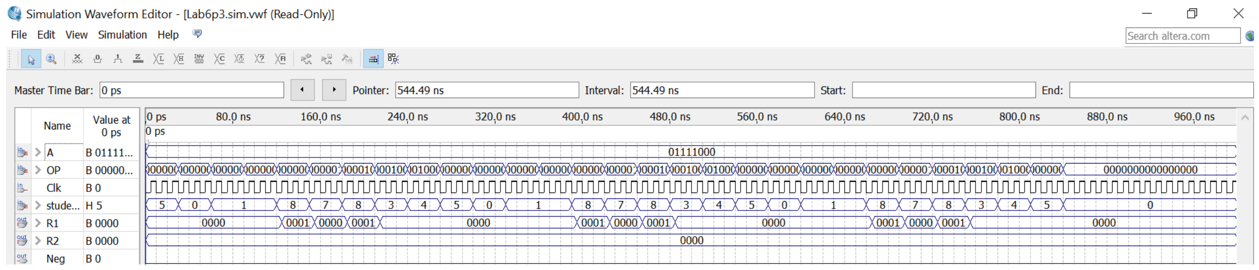

Simulation Waveform Editor - [Lab6p3.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 544.49 ns
Interval: 544.49 ns
Start:
End:
Name
Value at 0 ps
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
A B 01111... 01111000
OP B 00000... 0000000000000000
Clk B 0
stude... H 5 5 0 1 8 7 8 3 4 5 0 1 8 7 8 3 4 5 0 1 8 7 8 3 4 5 0
R1 B 0000 0000 0001 0000 0001 0000 0001 0000 0001 0000 0001 0000 0001 0000
R2 B 0000 0000
Neg B 0

Figure 14: ALU 3 waveform "0000" is "n" "0001" is "y"

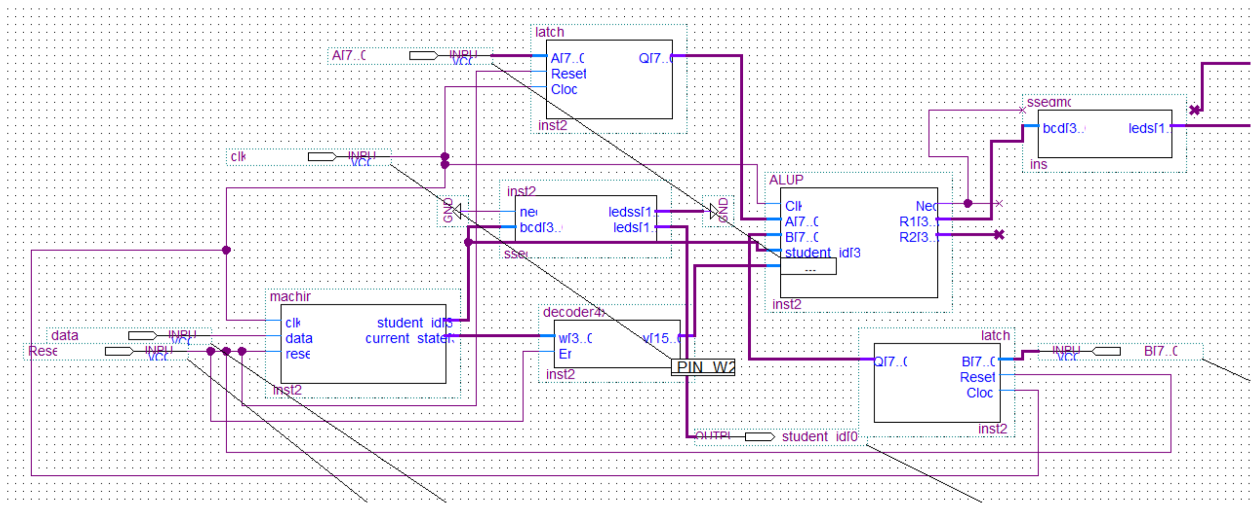
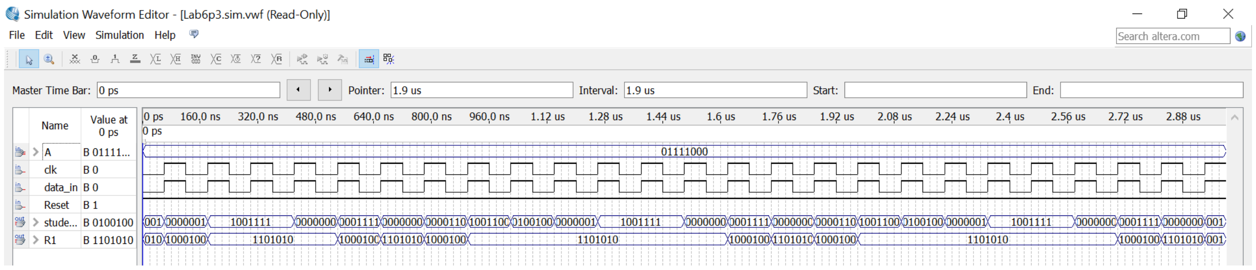


Figure 18: Combined ALU 3 Block Diagram



Simulation Waveform Editor - [Lab6p3.sim.vwf (Read-Only)]

```
Master Time Bar: 0 ps    Pointer: 1.9 us    Interval: 1.9 us    Start:    End:

Name       Value at 0 ps
A          B 01111...    01111000
clk        B 0
data_in    B 0
Reset      B 1
stude...   B 0100100     001 0000001 1001111 0000000 0001111 0000000 0000110 1001100 0100100 0000001 1001111 0000000 0001111 0000000 0000110 1001100 0100100 0000001 1001111 0000000 0001111 0000000 001
R1         B 1101010     010 1000100 1101010 1000100 1101010 1000100 1101010 1000100 1101010 1000100 1101010 1000100 1101010 001
```

Figure 14: Combined ALU 3 waveform *Note outputs are inverted

Comparing the waveform outputs to the VHDL Code, the outputs match the expected outputs.

Conclusion:

This lab project centered around the objective of creating a simple General-Purpose Processor circuit through the combination of arithmetic logic units, finite state machines, amongst other components that were implemented onto an FPGA board. Two inputs that were passed through latches A and B into the ALU were used to perform a certain function based on the selected function through the control unit's aid. The resulting output was at last displayed onto multiple seven segment displays. Overall, this lab was a vital learning chance to gain a thorough and in depth understanding of how sequential circuits work, through the reliance on one clock signal for all components. In the latter stages, small modifications were made to the circuit in order to satisfy the outputs needed for each respective part. As a result, this lab provided a useful review of other topics previously learned in digital systems, and opened up the option of furthering any interest in how microprocessors work and electronic systems as a whole.

References:

[1] "COE328 - Digital Systems," Department of Electrical and Computer Engineering, Toronto Metropolitan University. Available: https://www.ecb.torontomu.ca/~courses/coe328/. Accessed on: December 2, 2023.

Appendix:

Problem 1 Outputs:	Problem 2 Outputs:	Problem 3 Outputs: