Ryerson University

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

Course Title:	Digital Systems
Course Number:	COE 328
Semester/Year (e.g.F2016)	F2021

Instructor:	Dr.Patrick Siddavattam

Assignment/Lab Number:	LAB REPORT #6
Assignment/Lab Title:	Design of a General-Purpose Processor Unit (GPU)

Submission Date :	12/08/2021
Due Date:	12/09/2021

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Patel	Astha	501040209	14	A.P.

TABLE OF CONTENTS Page

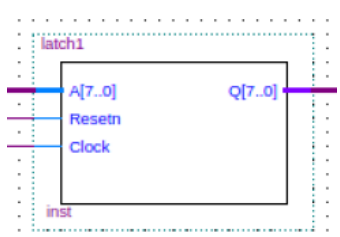
A BRIEF DESCRIPTION OF THE COMPONENTS

The objective of this lab experiment is to design a simple General-purpose Processing Unit (GPU). Multiple components will be utilized to build the GPU system, these components include Latches; a Finite-State Machine (FSM); a 4-to-16 Decoder; Arithmetic & Logic Unit (ALU); Seven-Segment Display (SSEG). A latch is a level sensitive storage element that temporarily stores the input. Next, an FSM works as a counter that could be useful when a certain order of outputs is required. Furthermore, a 4-to-16 decoder utilizes multiple inputs to execute a particular operation. An ALU machine combines information from different components to execute the desired output. Finally, the seven-segment displays a binary or a hexadecimal value from the final output. For this lab experiment will utilize all the components described above to design a final GPU. Within a computer system, a GPU executes mathematical operations for the Central Processing Unit (CPU). The goal of this lab is to experiment with different mathematical operations using the GPU design.

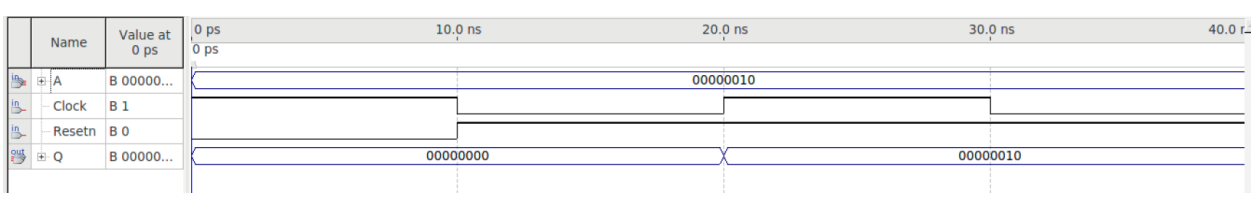
COMPONENTS:

2.1 Latch #1:

Latches are level sensitive storage elements; they execute operations when the machine is clocked. A latch has pins Clock; Reset; Input; output. Latch machines are level sensitive; they change or stay in the same state depending on the clock signal. The Clock pin allows the latch to discard the present data and store the new incoming data. The Reset pin resets the values to a user-customized value; the latch stays at the reset state until clocked again. The latch will store an 8-bit binary number and send its information to the ALU. The first 8-bit binary representation is the 6th and 7th digit of my student ID (0000 and 0010, respectively).



2.2.1: Block symbol A : A symbol for latch #1 machine.



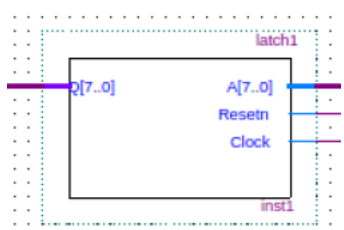
2.1.2: Waveform A: the waveform of a latch machine.

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

ENTITY latch1 IS
   PORT (A              : IN STD_LOGIC_VECTOR(7 DOWNTO 0); --8 bit 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 Behaviour OF latch1 IS
   BEGIN
      PROCESS (Resetn,Clock)--Process takes reset and clock as inputs
      BEGIN
         IF Resetn = '0' THEN --when reset inout 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 Behaviour;
```

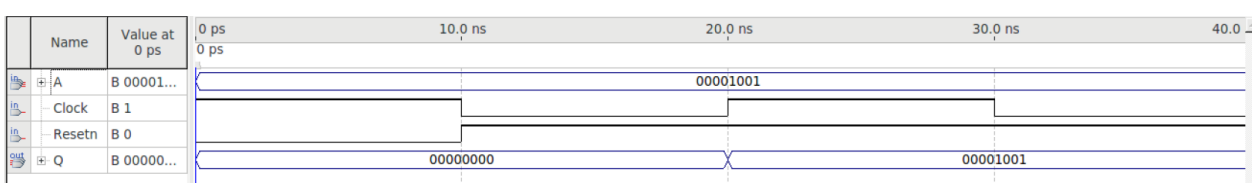
2.1.3: VHDL code A : A VHDL code for a Latch machine.

2.2 Latch #2:

This latch machine is identical to the previous machine, this time, this machine will store the 8th and the 9th digit (0000 and 1001, respectively) of my student ID number. As described above, latches are level sensitive storage elements; they execute operations when the machine is clocked. A latch has pins Clock; Reset; Input; output. The Clock pin allows the latch to discard the present data and store the new incoming data. The Reset pin resets the values to a user-customized value; the latch stays at the reset state until clocked again. The latch will store an 8-bit binary number and send its information to the ALU.



2.2.1: Block symbol B: A block diagram of Latch #2.



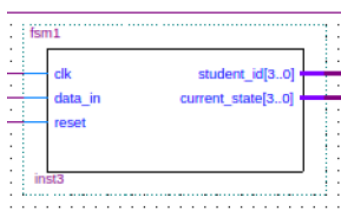
2.2.2: Waveform B: A waveform representation of the second latch symbol.

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

ENTITY latch1 IS
   PORT (A              : IN STD_LOGIC_VECTOR(7 DOWNTO 0); --8 bit 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 Behaviour OF latch1 IS
   BEGIN
      PROCESS (Resetn,Clock)--Process takes reset and clock as inputs
      BEGIN
         IF Resetn = '0' THEN --when reset inout 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 Behaviour;
```

2.2.3: VHDL code B: A VHDL code for a Latch machine.

2.3 Finite-State Machine (FSM):

An FSM is a machine that could represent a particular state at any given time, from a finite set of possible states. This machine will represent one state at a time, each time it is clocked. A Mealy FSM is utilized for this experiment. A Mealy machine is dependent on the primary inputs, the present state, and the primary outputs. Depending on the primary inputs, the machine will represent a state when clocked. The Mealy machine in this lab experiment represents the student ID as the states change from 0 to 8.



2.3.1: Block symbol C: A block symbol of the FSM machine.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;

ENTITY fsm1 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 fsm1;
ARCHITECTURE fsm OF fsm1 IS

--Build an enumerated type with 9 states for the state machine (9 states
--for parsing 9 digits of student ID)
   TYPE state_type IS (s0,s1,s2,s3,s4,s5,s6,s7,s8);

 --register to hold the current state
 SIGNAL yfsm : state_type;

BEGIN
   PROCESS(clk,reset)
   BEGIN
      IF reset = '1' THEN
         yfsm <= s0;
```

```
      ELSE IF (clk 'EVENT AND clk ='1') THEN
--Determine the next state synchronously,based on
--the current state and input.
         CASE yfsm IS

            WHEN s0 =>
               IF data_in = '0' THEN
                  current_state <= "0000";
                  yfsm <= s0;
               ELSE
                  current_state <= "0001";
                  yfsm <= s1;
               END IF;
            WHEN s1 =>
               IF data_in = '0' THEN
               current_state <= "0001";
                  yfsm <= s1;
               ELSE
                  current_state <= "0010";
                  yfsm <= s2;
               END IF;
            WHEN s2 =>
               IF data_in = '0' THEN
                  current_state <= "0010";
                  yfsm <= s2;
               ELSE
                  current_state <= "0011";
```

```
                yfsm <= s3;
            END IF;
        WHEN s3 =>
            IF data_in = '0' THEN
                current_state <= "0011";
                yfsm <= s3;
            ELSE
                current_state <= "0100";
                yfsm <= s4;
            END IF;
        WHEN s4 =>
            IF data_in = '0' THEN
                current_state <= "0100";
                yfsm <= s4;
            ELSE
                current_state <= "0101";
                yfsm <= s5;
            END IF;
        WHEN s5 =>
            IF data_in = '0' THEN
                current_state <= "0101";
                yfsm <= s5;
            ELSE
                current_state <= "0110";
                yfsm <= s6;
            END IF;
        WHEN s6 =>
```

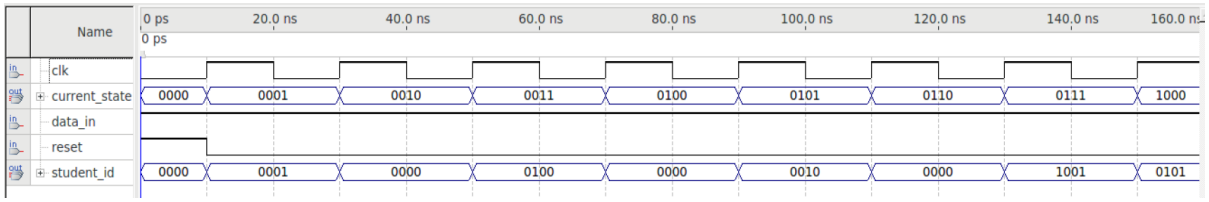
```
            IF data_in = '0' THEN
                current_state <= "0110";
                yfsm <= s6;
            ELSE
                current_state <= "0111";
                yfsm <= s7;
            END IF;
        WHEN s7 =>
            IF data_in = '0' THEN
                current_state <= "0111";
                yfsm <= s7;
            ELSE
                current_state <= "1000";
                yfsm <= s8;
            END IF;
        WHEN s8 =>
            IF data_in = '0' THEN
                current_state <= "1000";
                yfsm <= s1;
            ELSE
                current_state <= "0000";
                yfsm <= s0;
            END IF;

        END CASE;
    END IF;
    END IF;
```

```
  END PROCESS;

--implement the moore or mealy logic here

  PROCESS (yfsm,data_in)--IF REQUIRED
   BEGIN

      CASE yfsm IS

        WHEN s0 =>
           IF    data_in = '0' THEN
                 student_id <= "0101";
           ELSE
                 student_id <= "0000";
           END IF;
        WHEN s1 =>
           IF    data_in = '0' THEN
                 student_id <= "0000";
           ELSE
                 student_id <= "0001";
           END IF;
        WHEN s2 =>
           IF     data_in = '0' THEN
                 student_id <= "0001";
           ELSE
                 student_id <= "0000";
           END IF;
```

```
        WHEN s3 =>
           IF    data_in = '0' THEN
                 student_id <= "0000";
           ELSE
                 student_id <= "0100";
           END IF;
        WHEN s4 =>
           IF    data_in = '0' THEN
                 student_id <= "0100";
           ELSE
                 student_id <= "0000";
           END IF;
        WHEN s5 =>
           IF    data_in = '0' THEN
                 student_id <= "0000";
           ELSE
                 student_id <= "0010";
           END IF;
        WHEN s6 =>
           IF    data_in = '0' THEN
                 student_id <= "0010";
           ELSE
                 student_id <= "0000";
           END IF;
        WHEN s7 =>
           IF    data_in = '0' THEN
                 student_id <= "0000";
```

```
           ELSE
                 student_id <= "1001";
           END IF;
        WHEN s8 =>
           IF    data_in = '0' THEN
                 student_id <= "1001";
           ELSE
                 student_id <= "0101";
           END IF;
      END CASE;
   END PROCESS;
END fsm;
```

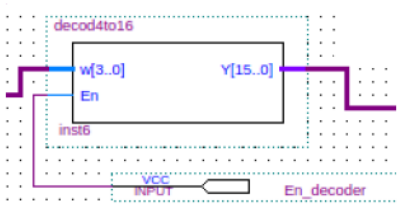
2.3.2: VHDL code C: The VHDL code for the required FSM machine that is designed using the Mealy FSM. The states change from 0 to 8.



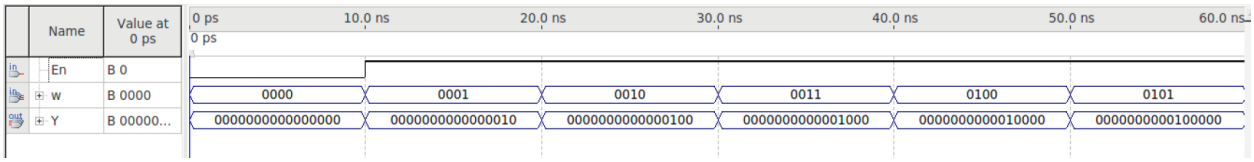
2.3.3: Waveform C : A waveform of the Mealy FSM. An example of the above mealy FSM design. The data_in = 1, the mealy machine refers to the next state, which is s1 and the student digit is s1 = 0 when data_in =1.

2.4 4-to-16 Decoder:

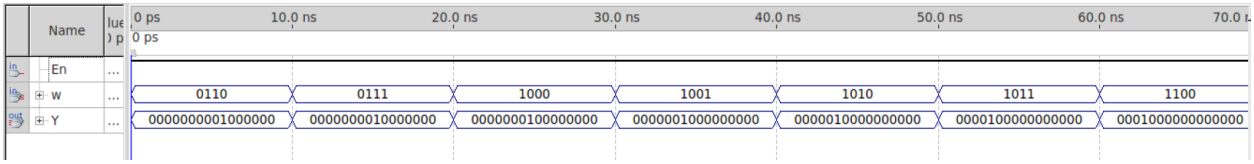
A 4-to-16 decoder receives a 4-bit binary number that is utilized to output one piece of information, a 16-bit binary number for this case. There are sixteen possible outputs that are executed one at a time depending on the 4-bit input. For this experiment, these inputs represent an output which is then reported to the ALU that executes the operations. The 4-to-16 decoder will receive a state that the FSM is representing at a particular time, the decoder will then output a 16-bit binary number that is then sent to the ALU to manage. An Enable pin enables the decoder to be turned on and off. The decoder will only output information if the Enable is on.



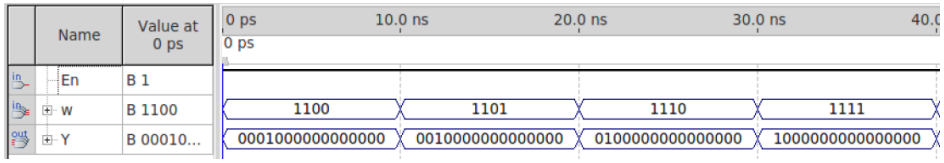
2.4.1: Block symbol D: A block symbol of a 4-to-16 decoder.



2.4.2: Waveform D part 1: A waveform of a 4-to-16 decoder for the first 6 macrocodes.



2.4.2: Waveform D part 2: A waveform of a 4-to-16 decoder for the next 7 macrocodes.



2.4.2: Waveform D part 3: A waveform of a 4-to-15 decoder for the last 4 macrocodes.

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

ENTITY decod4to16 IS
   PORT (w  : IN  STD_LOGIC_VECTOR(3 DOWNTO 0) ;
         En : IN  STD_LOGIC ;
         Y  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0))  ;

END decod4to16 ;

ARCHITECTURE Behavior OF decod4to16 IS
   SIGNAL Enw : STD_LOGIC_VECTOR(4 DOWNTO 0) ;
BEGIN
   Enw <= En & w ;
   WITH Enw SELECT
   Y <=    "0000000000000001" WHEN "10000",
           "0000000000000010" WHEN "10001",
           "0000000000000100" WHEN "10010",
           "0000000000001000" WHEN "10011",
           "0000000000010000" WHEN "10100",
           "0000000000100000" WHEN "10101",
           "0000000001000000" WHEN "10110",
           "0000000010000000" WHEN "11000",
           "0000000100000000" WHEN "11001",
           "0000001000000000" WHEN "11010",
           "0000010000000000" WHEN "11011",
           "0000100000000000" WHEN "11100",
           "0001000000000000" WHEN "11101",
           "0010000000000000" WHEN "11110",
           "0100000000000000" WHEN "11111",

           "0000000000000000" WHEN OTHERS;

END Behavior ;
```

2.4.3: VHDL code D : a VHDL code for a 4-to-16 decoder.

ALU_1

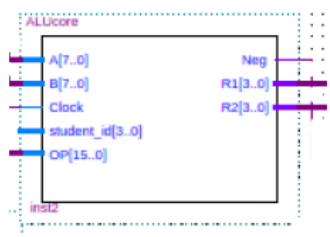
An ALU is a processing unit that handles mathematical calculations. For this lab experiment, the ALU component receives two types of information, first, two 8-bit numbers from the two latches, pin A[7..0] and pin B[7..0], and second, a 16-bit microcode from the 4-to-16 decoder. The decoder will receive a value from the FSM and send the 16-bit microcode command associated with the current state to the ALU. Inside the ALU code prompt, there are operations defined under each microcode; the operation is executed each time a microcode is received. Furthermore, once the ALU receives a microcode, the operation is applied to the two 8-bit numbers from the latches. The result, also an 8-bit number, stores the number inside two arrays, R1[3..0] and R2[3..0]; R1 stores the lower 4-

bits and R2 stores the higher 4-bits of the result. Regarding the pins on the block diagram: the input pins A[7..0] and B[7..0] are the two 8-bit numbers received from the two latches, pin OP[15..0] is the 16-bit number sent from the decoder, and the Clock pin allows ALU to move onto the next cycle of the microcode , and the output pins, Neg will be utilized to display a negative sign if the result is a negative number, and finally, pins R1[3..0] and R2[3..0] will store the 8-bit result executed from the ALU operation. This result is then sent to the seven-segment display that will display the binary or hexadecimal number on the display, observed on the waveform.

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

Table 1. ALU Core Operations for Problem 1

3.1.1: Problem 1



3.1.2: Block symbol 1.1 : a block symbol of the ALU component with operations integrated concerning Table 1.

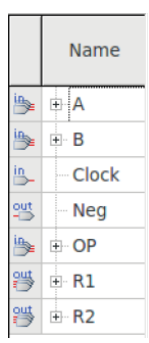
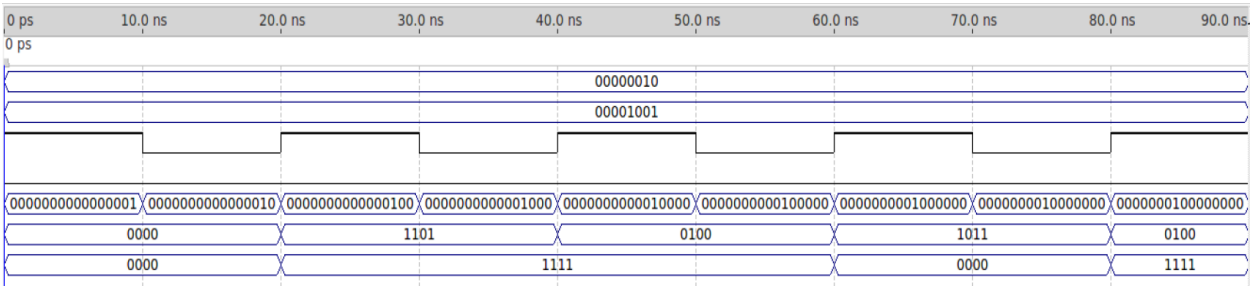


Figure 1: a table of the pins utilized on the waveform in the next figure.



3.1.3: Waveform 1.1: A waveform of the ALU component only.

```
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.all;
USE ieee.STD_LOGIC_UNSIGNED.all;
USE ieee.NUMERIC_STD.all;

ENTITY Problem2d IS
PORT(A,B        : IN UNSIGNED(7 DOWNTO 0);-- 8 bit inputs from latches A and B
     Clock      : IN STD_LOGIC; -- input clock signal
     student_id : IN UNSIGNED(3 DOWNTO 0);--4 bit student ID from FSM
     OP         : IN UNSIGNED(15 DOWNTO 0); --16-bit selector for Operation from Decoder
     Neg        : OUT STD_LOGIC; -- is the result negative? set -ve output
     R1         : OUT UNSIGNED(3 DOWNTO 0);--lower 4-bits of 8-bit Result Outout
     R2         : OUT UNSIGNED(3 DOWNTO 0)); --higher 4-bits of 8-bits Result Output

END Problem2d;

ARCHITECTURE calculation OF Problem2d IS -- temporary signal declarations

SIGNAL Reg1,Reg2,Result : UNSIGNED(7 DOWNTO 0) := (OTHERS => '0');
SIGNAL Reg4 :UNSIGNED(0 TO 7);

BEGIN

Reg1 <= A; --temporary store A in Reg1 local variable
Reg2 <= B; --temporary store B in Reg2 local variable

PROCESS(Clock,OP)
BEGIN
```

```
IF(rising_edge(Clock)) THEN --Do calculation @ positive edge of clock cycle
   CASE OP IS

      WHEN "0000000000000001" =>
      --Shift A to right by two bits, input bit =1 (SHR)
         Result <= Reg1 srl 2;

      WHEN "0000000000000010" =>
      --Produce the difference of A and B and then increment by 4
            Result <= (Reg1 - Reg2) + 4;
         IF (Result < 0) THEN
            Neg <= '1';
         ELSE Neg <= '0';
         END IF;

      WHEN "0000000000000100" =>
      --Find the greater value of A and B and produce the results (Max(A,B))
         IF(Reg1 > Reg2) THEN
            Result <= Reg1;
         ELSE Result <= Reg2;
         END IF;

      WHEN "0000000000001000" =>
      --Swap the upper 4 bits of A by the lower 4 bits of B
         Result(7 DOWNTO 4) <= Reg2(3 DOWNTO 0);
         Result(3 DOWNTO 0) <= Reg1(3 DOWNTO 0);

      --Do boolean NOR
         Result <= Reg1 NOR Reg2;

      WHEN "0000000000100000" =>
      --Do boolean AND
         Result <= Reg1 AND Reg2;

      WHEN "0000000001000000" =>
      --Do boolean XOR
         Result <= Reg1 XOR Reg2;

      WHEN "0000000010000000" =>
         --Do boolean OR
         Result <= Reg1 OR Reg2;

      WHEN "0000000100000000" =>
      --Do boolean XNOR
         Result <= Reg1 XNOR Reg2;

      WHEN OTHERS =>
      --Don't care ,do nothing

   END CASE;
 END IF;
END PROCESS;

R1 <= Result(3 DOWNTO 0); --Since the output seven segment can

R2 <= Result(7 DOWNTO 4); -- Only 4-bits, split the 8-bit two 4-bits

END calculation;
```

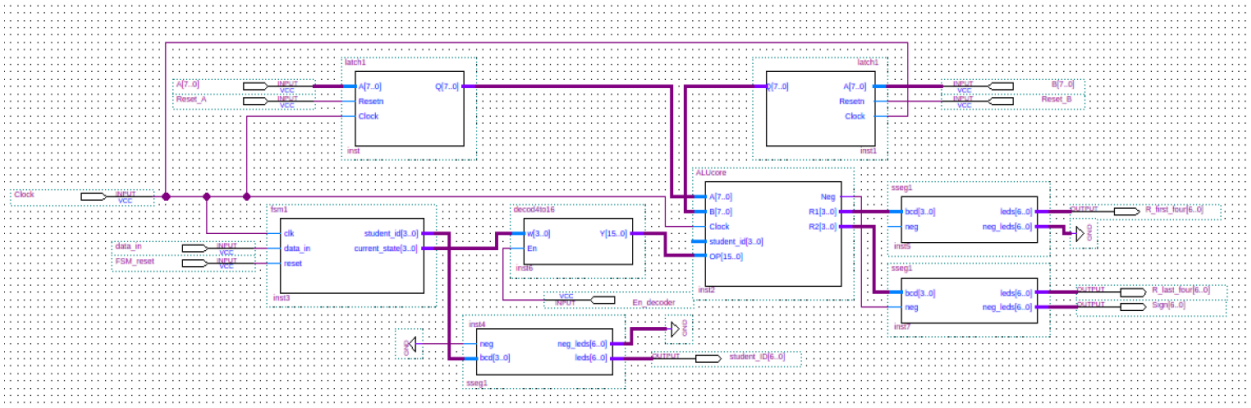
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY sseg1 IS
   PORT ( bcd :IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          leds :OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
          neg  : IN STD_LOGIC;
          neg_leds : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
         );

end sseg1;

ARCHITECTURE Behavior OF sseg1 IS
BEGIN
   PROCESS (bcd,neg)
   BEGIN
   IF (neg = '0') THEN
      neg_leds <= "0000000";
      CASE bcd IS                --abcdefg
         WHEN "0000" => leds <= "1111110";--0
         WHEN "0001" => leds <= "0110000";--1
         WHEN "0010" => leds <= "1101101";--2
         WHEN "0011" => leds <= "1111001";--3
         WHEN "0100" => leds <= "0110011";--4
         WHEN "0101" => leds <= "1011011";--5
         WHEN "0110" => leds <= "1011111";--6
         WHEN "0111" => leds <= "1110000";--7
         WHEN "1000" => leds <= "1111111";--8
```

2.4.4: VHDL code 1.1: A VHDL code for the ALU design.

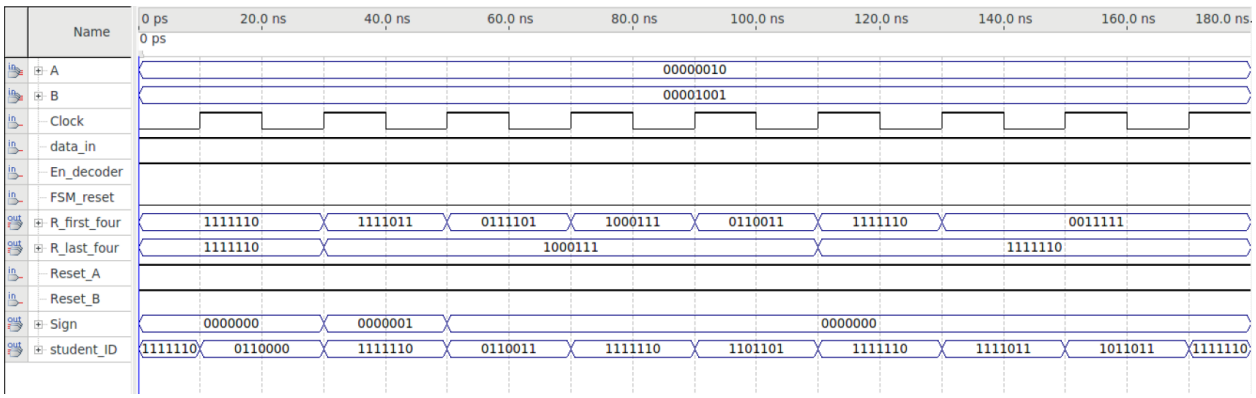
```
         WHEN "1001" => leds <= "1111011";--9
         WHEN "1010" => leds <= "1110111";--A
         WHEN "1011" => leds <= "0011111";--B
         WHEN "1100" => leds <= "1001110";--C
         WHEN "1101" => leds <= "0111101";--D
         WHEN "1110" => leds <= "1001111";--E
         WHEN "1111" => leds <= "1000111";--F
         WHEN OTHERS => leds <= "-------";
      END CASE;

   ELSE
                     neg_leds <= "0000001";
      CASE bcd IS
         WHEN "0000" => leds <= "1111110";--0
         WHEN "0001" => leds <= "0110000";--1
         WHEN "0010" => leds <= "1101101";--2
         WHEN "0011" => leds <= "1111001";--3
         WHEN "0100" => leds <= "0110011";--4
         WHEN "0101" => leds <= "1001011";--5
         WHEN "0110" => leds <= "1011111";--6
         WHEN "0111" => leds <= "1110000";--7
         WHEN "1000" => leds <= "1111111";--8
         WHEN "1001" => leds <= "1111011";--9
         WHEN "1010" => leds <= "1110111";--A
         WHEN "1011" => leds <= "0011111";--B
         WHEN "1100" => leds <= "1001110";--C
         WHEN "1101" => leds <= "0111101";--D
```

```
         WHEN "1110" => leds <= "1001111";--E
         WHEN "1111" => leds <= "1000111";--F
         WHEN OTHERS => leds <= "-------";
         END CASE;
   END IF;
END PROCESS;
END Behavior;
```

3.1.5: VHDL code 1.2: A VHDL code for the seven-segment display.



3.1.6: Block Diagram 1: The final block diagram of the GPU design.



3.1.7: Waveform 1.2: A waveform of the final GPU design.

ALU_2

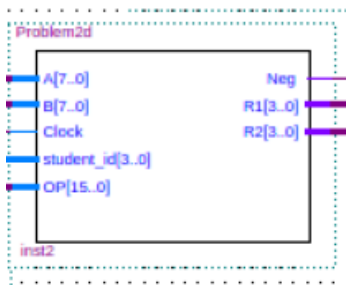
The second ALU design executes mathematic operations based on a 16-bit input as well. This ALU2 design will receive two 8-bit inputs from the latches as well and a 16-bit microcode from the 4-to-16 decoder. The decoder receives a current state from the FSM and the decoder associates that current state binary value with the 16-bit microcode inside the decoder code. The decoder will send this microcode to the ALU to handle the rest of the execution process. Once the ALU receives the microcode, it will run the mathematical operation linked to that microcode. The operation will be applied to the two 8-bit binary numbers that are temporarily stores in the latches. The result will be an 8-bit number that will be split into two arrays. The pins containing the ALU block symbol are: the input pins A[7..0] and B[7..0] are the two 8-bit numbers obtained from the latches, pin OP[15..0] will send the microcode from the decoder, and the Clock symbol will allow the ALU to move onto the next cycle, and the output pins, R1[3..0] and R2[3..0] will contain the result that is split into two arrays. R1[3..0] contains the lower 4-bits of the result and R2[3..0] contain the

higher 4-bits. This result will be displayed on the seven-segment display as either a binary or a hexadecimal representation of the student ID number, observed on the waveform.

d)

Function #	Operation / Function
1	Shift **A** to right by two bits, input bit = 1 (SHR)
2	Produce the difference of **A** and **B** and then increment by 4
3	Find the greater value of **A** and **B** and produce the results (Max(**A**,**B**))
4	Swap the upper 4 bits of **A** by the lower 4 bits of **B**
5	Increment **A** by 1
6	Produce the result of ANDing **A** and **B**
7	Invert the upper four bits of **A**
8	Rotate **B** to left by 3 bits (ROL)
9	Show null on the output

3.2.1: Table 2: a table of operations integrated for this ALU design.



3.2.2: Block symbol 2.1: A block symbol of the ALU component for problem 2.

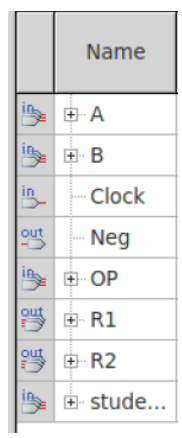
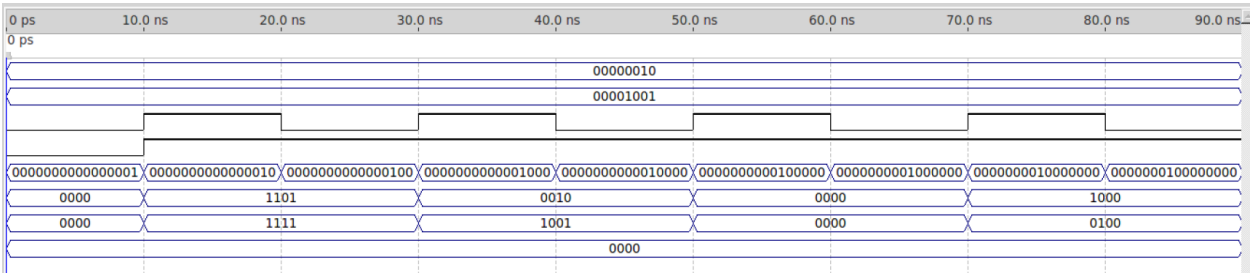


Figure 2.1: A table of the pins utilized on the waveform of ALU.



3.2.3: Waveform 2.1: A waveform of the ALU design concerning Table 2.

```
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.all;
USE ieee.STD_LOGIC_UNSIGNED.all;
USE ieee.NUMERIC_STD.all;

ENTITY Problem2d IS
PORT(A,B        : IN UNSIGNED(7 DOWNTO 0);-- 8 bit inputs from latches A and B
     Clock      : IN STD_LOGIC; -- input clock signal
     student_id : IN UNSIGNED(3 DOWNTO 0);--4 bit student ID from FSM
     OP         : IN UNSIGNED(15 DOWNTO 0); --16-bit selector for Operation from Decoder
     Neg        : OUT STD_LOGIC; -- is the result negative? set -ve output
     R1         : OUT UNSIGNED(3 DOWNTO 0);--lower 4-bits of 8-bit Result Outout
     R2         : OUT UNSIGNED(3 DOWNTO 0)); --higher 4-bits of 8-bits Result Output

END Problem2d;

ARCHITECTURE calculation OF Problem2d IS -- temporary signal declarations

SIGNAL Reg1,Reg2,Result : UNSIGNED(7 DOWNTO 0) := (OTHERS => '0');
SIGNAL Reg4 :UNSIGNED(0 TO 7);

BEGIN

Reg1 <= A; --temporary store A in Reg1 local variable
Reg2 <= B; --temporary store B in Reg2 local variable

PROCESS(Clock,OP)
BEGIN

   IF(rising_edge(Clock)) THEN --Do calculation @ positive edge of clock cycle
      CASE OP IS

         WHEN "0000000000000001" =>
         --Shift A to right by two bits, input bit =1 (SHR)
            Result <= Reg1 srl 2;

         WHEN "0000000000000010" =>
         --Produce the difference of A and B and then increment by 4
         IF(Reg1 < Reg2) THEN
            Result <= (Reg1 - Reg2) + 4;
            Neg <= '1';
         ELSE
            Result <= (Reg1 + Reg2 ) + 4;
            Neg <= '0';
         END IF;

         WHEN "0000000000000100" =>
         --Find the greater value of A and B and produce the results (Max(A,B))
            IF Reg1 > Reg2 THEN
            Result <= Reg1;
            ELSE Result <= Reg2;
            END IF;
            Neg <='0';

         WHEN "0000000000001000" =>
         --Swap the upper 4 bits of A by the lower 4 bits of B
```

```
            WHEN "0000000000010000" =>
            --Increment A by 1
                  Result <= Reg1 + 1;

            WHEN "0000000000100000" =>
            --Produce the results of ANDing A and B
               Result <= Reg1 AND Reg2;

            WHEN "0000000001000000" =>
            --Invert the upper four bits of A
               Result(0) <= Reg1(0);
               Result(1) <= Reg1(1);
               Result(2) <= Reg1(2);
               Result(3) <= Reg1(3);
               Result(4) <= NOT Reg1(4);
               Result(5) <= NOT Reg2(5);
               Result(6) <= NOT Reg2(6);
               Result(7) <= NOT Reg2(7);

            WHEN "0000000010000000" =>
               --Rotate B to left by 3 bits (ROL)
               Result <= Reg2 ROL 3;

            WHEN "0000000100000000" =>
            --Show null on the output
               Result <= NULL;

             WHEN OTHERS =>
             --Don't care ,do nothing

         END CASE;
      END IF;
   END PROCESS;

   R1 <= Result(3 DOWNTO 0); --Since the output seven segment can
   R2 <= Result(7 DOWNTO 4); -- Only 4-bits, split the 8-bit two 4-bits

END calculation;
```

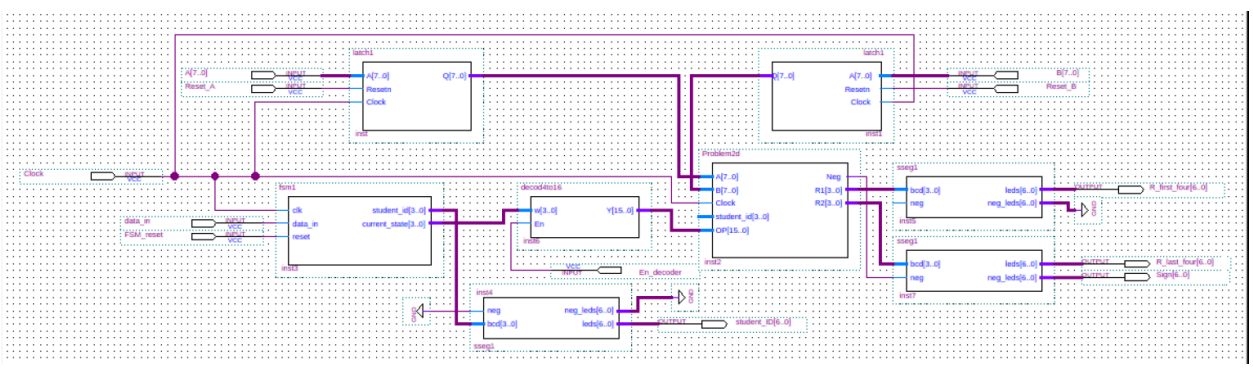
3.2.4: VHDL code 2.1: A VHDL code for the ALU design concerning problem 2.

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

ENTITY sseg1 IS
   PORT ( bcd :IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          leds :OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
          neg  : IN STD_LOGIC;
          neg_leds : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
         );

end sseg1;

ARCHITECTURE Behavior OF sseg1 IS
BEGIN
   PROCESS (bcd,neg)
   BEGIN
   IF (neg = '0') THEN
      neg_leds <= "0000000";
      CASE bcd IS                --abcdefg
         WHEN "0000" => leds <= "1111110";--0
         WHEN "0001" => leds <= "0110000";--1
         WHEN "0010" => leds <= "1101101";--2
         WHEN "0011" => leds <= "1111001";--3
         WHEN "0100" => leds <= "0110011";--4
         WHEN "0101" => leds <= "1011011";--5
         WHEN "0110" => leds <= "1011111";--6
         WHEN "0111" => leds <= "1110000";--7
         WHEN "1000" => leds <= "1111111";--8
```

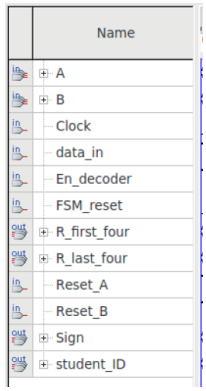
```
            WHEN "1001" => leds <= "1111011";--9
            WHEN "1010" => leds <= "1110111";--A
            WHEN "1011" => leds <= "0011111";--B
            WHEN "1100" => leds <= "1001110";--C
            WHEN "1101" => leds <= "0111101";--D
            WHEN "1110" => leds <= "1001111";--E
            WHEN "1111" => leds <= "1000111";--F
            WHEN OTHERS => leds <= "-------";
         END CASE;

      ELSE
                        neg_leds <= "0000001";
         CASE bcd IS
            WHEN "0000" => leds <= "1111110";--0
            WHEN "0001" => leds <= "0110000";--1
            WHEN "0010" => leds <= "1101101";--2
            WHEN "0011" => leds <= "1111001";--3
            WHEN "0100" => leds <= "0110011";--4
            WHEN "0101" => leds <= "1001011";--5
            WHEN "0110" => leds <= "1011111";--6
            WHEN "0111" => leds <= "1110000";--7
            WHEN "1000" => leds <= "1111111";--8
            WHEN "1001" => leds <= "1111011";--9
            WHEN "1010" => leds <= "1110111";--A
            WHEN "1011" => leds <= "0011111";--B
            WHEN "1100" => leds <= "1001110";--C
            WHEN "1101" => leds <= "0111101";--D
```

```
            WHEN "1110" => leds <= "1001111";--E
            WHEN "1111" => leds <= "1000111";--F
            WHEN OTHERS => leds <= "-------";
            END CASE;
      END IF;
   END PROCESS;
END Behavior;
```

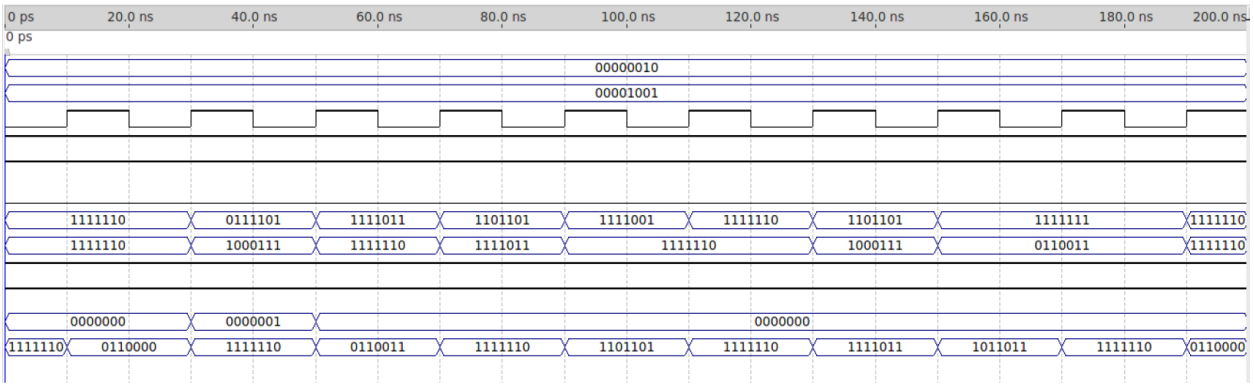
3.2.5: VHDL code 2.2: A VHDL code for the seven-segment display.



3.2.6: Block diagram 2: A block diagram of the second GPU design.



3.2.7 Figure 2.2: A table of the input/output pins utilized for the next waveform.



3.2.8 : Waveform 2.2: A waveform of the second GPU design.

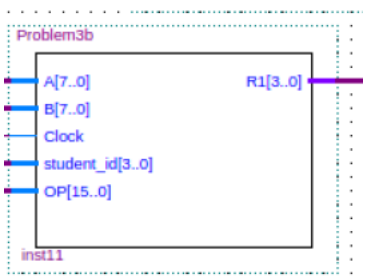
ALU_3

The following ALU design is dependent on the input student ID number. The ALU receives a 16-bit microcode from the 4-to-16 decoder and an 8-bit student ID number; this student ID is obtained from the FSM component. The 16-bit microcode decides which operation will be executed by the ALU. For this specific case of the ALU design, each microcode will assess the same condition for each microcode received. If the student ID is odd, then the output will be ‘y’ and if the student ID is even then the output will be ‘n’. The even or odd number is determined via a modulus operator, if the remainder of any number divided by an even number is 0 then the input is an even number and odd when remainder equals 1. The seven-segment will display this output as either ‘y’ or ‘n’. Furthermore, ALU will receive a microcode each time the component is clocked, the decoder will receive the state that the FSM is in and send the microcode to the ALU to execute the operations. The result will be either ‘0000 0000’ if the ID is odd or ‘0000 0001’ if the ID is even. The result will be stored in an array of 4-bits. This is

because the higher 4-bits are the same for both results, it would not be useful to store those values, thus, only the lower 4-bits will be stored inside an array, pin R1[3..0]. This result will be assessed by the seven-segment display and the final output will be displayed on the seven-segment display as either ‘y’ or ‘n’. The pins utilized for this ALU design ; the input pins are student_id[3..0] which is obtained from the FSM (resides in the control unit), pin OP[15..0] is the microcode received from the 4-to-16 bit decoder(resides in the control unit), and Clock will allow the ALU to move onto the next cycle, and the only output pin utilized is R1[3..0] that contains the lower 4-bits of the result. Contrary to the previous two ALU designs that utilized two arrays to display the output, only the lower 4-bits from the result will be assessed further. The higher 4-bits could be ignored as they are identical. In fact, only the lower 2-bits are assessed in this experiment but keeping the first array (R1[3..0]) is useful to prevent any extra adjustments to the seven-segment code.

b) For each microcode instruction, display 'y' if the FSM output (**student_id**) is even and 'n' otherwise

3.3.1: Figure 3.1: Problem 3.



3.3.2: Block Symbol 3: A block symbol of the third ALU component.

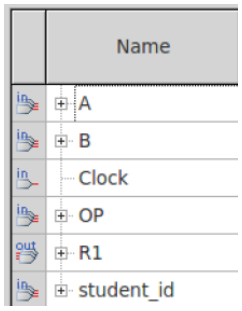
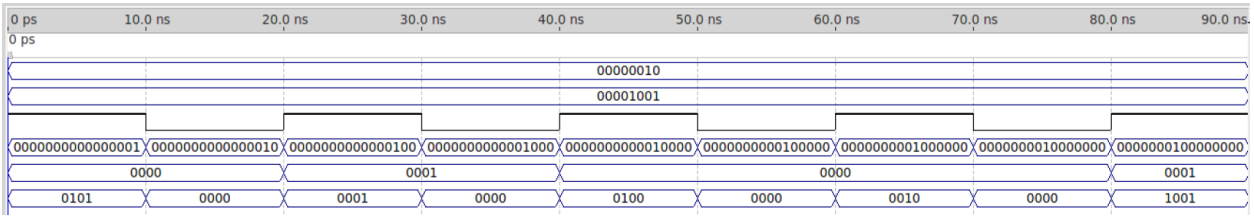


Table 3.1 : A table of pins utilized for the ALU waveform.



3.3.3: Waveform 3.1 : A waveform for the ALU design concerned with Problem 3.

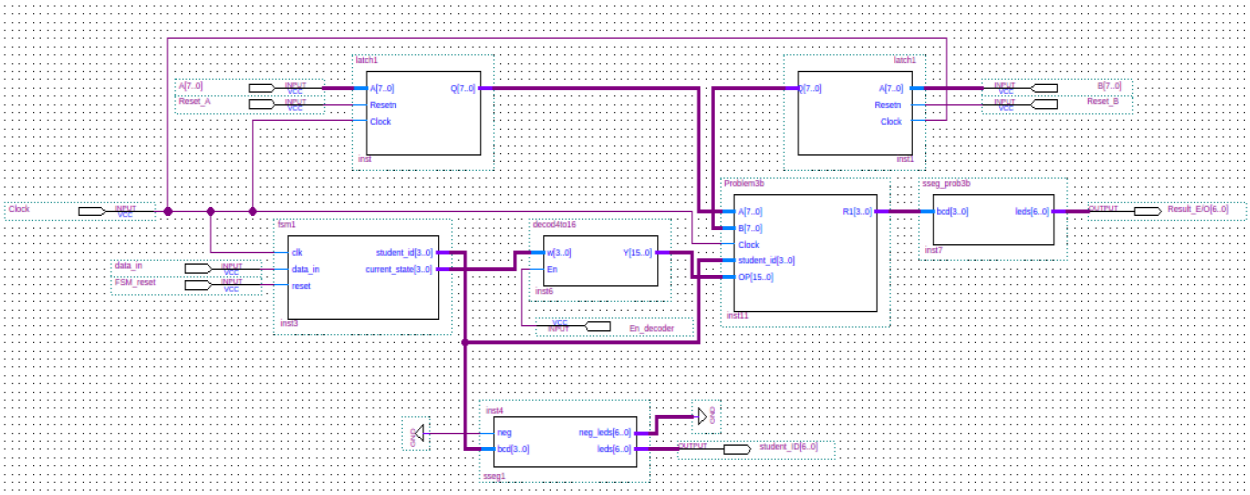
```
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.all;
USE ieee.STD_LOGIC_UNSIGNED.all;
USE ieee.NUMERIC_STD.all;

ENTITY Problem3b IS
PORT(A,B        : IN UNSIGNED(7 DOWNTO 0);-- 8 bit inputs from latches A and B
     Clock      : IN STD_LOGIC; -- input clock signal
     student_id : IN UNSIGNED(3 DOWNTO 0);--4 bit student ID from FSM
     OP         : IN UNSIGNED(15 DOWNTO 0); --16-bit selector for Operation from Decoder
     R1         : OUT UNSIGNED(3 DOWNTO 0));--lower 4-bits of 8-bit Result Outout


END Problem3b;

ARCHITECTURE calculation OF Problem3b IS -- temporary signal declarations

SIGNAL Reg1,Reg2,Result : UNSIGNED(7 DOWNTO 0) := (OTHERS => '0');
SIGNAL Reg4 :UNSIGNED(0 TO 7);

BEGIN

PROCESS(Clock,OP)
BEGIN
   IF(rising_edge(Clock)) THEN --Do calculation @ positive edge of clock cycle
      CASE OP IS

         WHEN "0000000000000001" =>

            IF (student_id MOD 2) = 0 THEN
               Result <= "00000000"; --Y

            ELSE
               Result <= "00000001"; --N
            END IF;

         WHEN "0000000000000010" =>
            IF (student_id MOD 2) = 0 THEN
               Result <= "00000000"; --Y

            ELSE
               Result <= "00000001"; --N
            END IF;

         WHEN "0000000000000100" =>
            IF (student_id MOD 2) = 0 THEN
               Result <= "00000000"; --Y

            ELSE
               Result <= "00000001"; --N
            END IF;

         WHEN "0000000000001000" =>
            IF (student_id MOD 2) = 0 THEN
               Result <= "00000000"; --Y

```

```
            ELSE
               Result <= "00000001"; --N
            END IF;

         WHEN "0000000000010000" =>
            IF (student_id MOD 2) = 0 THEN
               Result <= "00000000"; --Y

            ELSE
               Result <= "00000001"; --N
            END IF;

         WHEN "0000000000100000" =>
            IF (student_id MOD 2) = 0 THEN
               Result <= "00000000"; --Y

            ELSE
               Result <= "00000001"; --N
            END IF;

         WHEN "0000000001000000" =>
            IF (student_id MOD 2) = 0 THEN
               Result <= "00000000"; --Y

            ELSE
               Result <= "00000001"; --N
            END IF;

         WHEN "0000000010000000" =>
            IF (student_id MOD 2) = 0 THEN
               Result <= "00000000"; --Y

            ELSE
               Result <= "00000001"; --N
            END IF;

         WHEN "0000000100000000" =>
            IF (student_id MOD 2) = 0 THEN
               Result <= "00000000"; --Y

            ELSE
               Result <= "00000001"; --N
            END IF;

         WHEN OTHERS =>
         --Don't care ,do nothing

      END CASE;
   END IF;
END PROCESS;

R1 <= Result(3 DOWNTO 0);


END calculation;
```

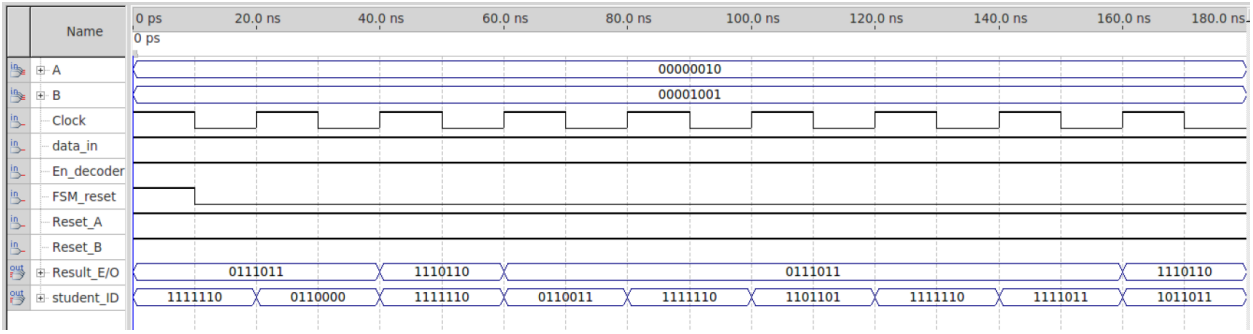
3.3.4: VHDL code 3.1: A VHDL code for the ALU component regarding problem 3.

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

ENTITY sseg_prob3b IS
   PORT ( bcd :IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          leds :OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
         );

end sseg_prob3b;

ARCHITECTURE Behavior OF sseg_prob3b IS
BEGIN
   PROCESS (bcd)
   BEGIN

      CASE bcd IS                --abcdefg
         WHEN "0000" => leds <= "0111011"; --y
         WHEN "0001" => leds <= "1110110"; --N
         WHEN OTHERS => leds <= "-------"; --
   END CASE;
 END PROCESS;
END Behavior;
```

3.3.5: VHDL code 3.3: A VHDL code for the seven-segment display regarding Problem 3.



3.3.6: Block Diagram 3.1: A block diagram of the GPU design concerned with Problem 3.



3.3.7: Waveform 3.2: A waveform of the GPU design.

CONCLUSION

A Central Processing Unit (CPU) is a key component within every computer system. It is known as the brain of a computer system. A CPU is a computer's control unit that receives information, chooses operations to be executed related to that information and sends the result of that operation as an output. Within the CPU, there is a General-Purpose Processing Unit (GPU) that handles all mathematical operations for the CPU. The GPU receives information which it then associates with a microcode that executes a particular operation and outputs a result. There are three different ALU designs involved in this experiment, the 1st and the 2nd ALU design involves mathematical operation being applied to two 8-bit binary numbers and the result outputs an 8-bit result, the 3rd ALU design checks whether the input 8-bit number is even or odd and the result determined using a mathematical operation as well. First the two 8-bit numbers are stored in a storage element, a latch for this experiment, which is sent to the ALU, the ALU receives a second input from the control unit, a microcode. This microcode decides which operation is to be executed by the ALU design. The result is sent to the seven-segment display, which is then associated with a binary-to-decimal or hexadecimal representation of the binary output result. The result of the performed operation is then displayed on a seven-segment display.

The goal of designing a simple GPU design was successful as the waveforms of all ALU designs display the expected output on the waveforms. All waveforms display a seven-segment display output represents the decimal or hexadecimal representation of binary representation of the result.