# Design of a Simple Processor

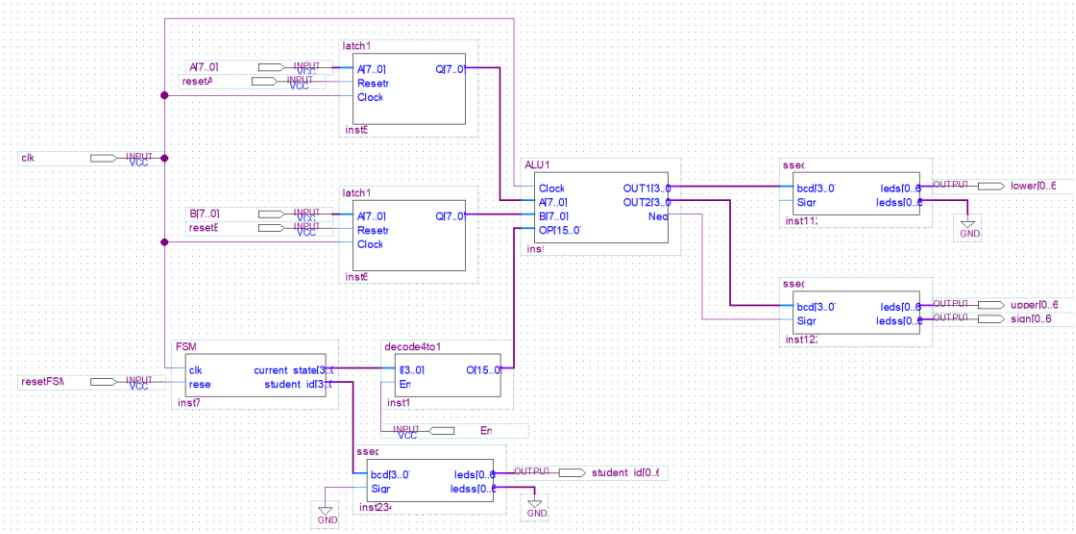
COE 328-022 Pei Yang

Dec 3 2022

# Table of Contents:

# Introduction:

**Part 1:**




Components used:

2 x Latch

1 x decoder (4:16)

1 x FSM

1 x ALU (Version 1)

3 x Sseg

The latch is used to store the value of the input, until a HIGH clock value is observed by the block.

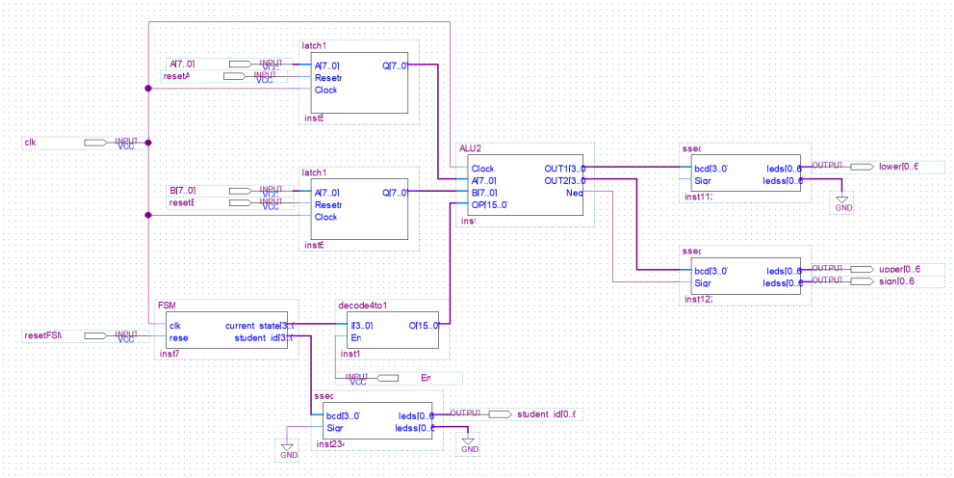
The decoder (4:16) is used to interpret the 4-bit FSM output and convert it into a 16-bit, which will be fed into the ASU unit as input. En is used to control when the block operates, and must be 1 in order for O[15..0] to output data.

The FSM is used to determine the current state and student id, by cycling through the programmed list of states and state values, and student id numbers.

The ALU accepts two numbers from the latches and performs an arithmetic/logical operation depending on the 16-bit input from decoder.

The Ssegs is used to convert the 8-bit input received into a 7-bit output which would be used with a seven segment LED display, to display the result in hexadecimal.

**Part 2:**



Components used:

2 x Latch

1 x decoder (4:16)

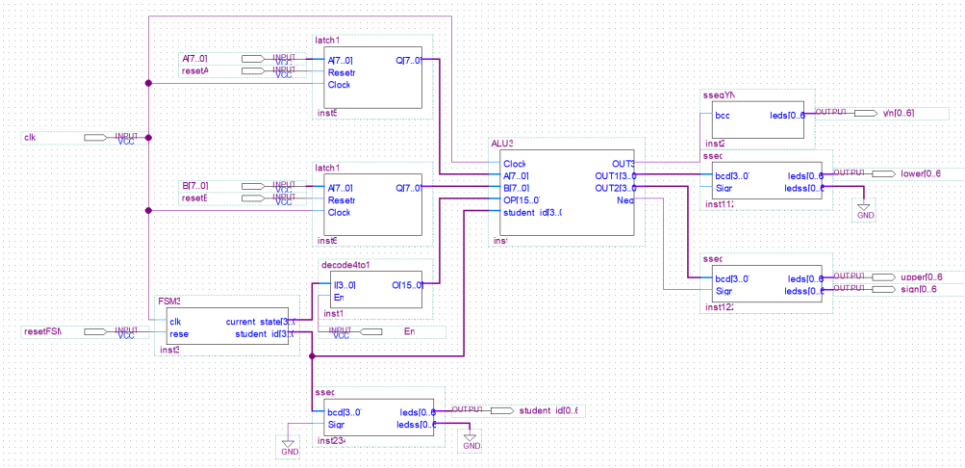
1 x FSM

1 x ALU (Version 2)

3 x Sseg

All components act the same as the ones in part 1.

**Part 3:**



Components used:

2 x Latch

1 x decoder (4:16)

1 x FSM

1 x ALU (Version 3)

3 x Sseg

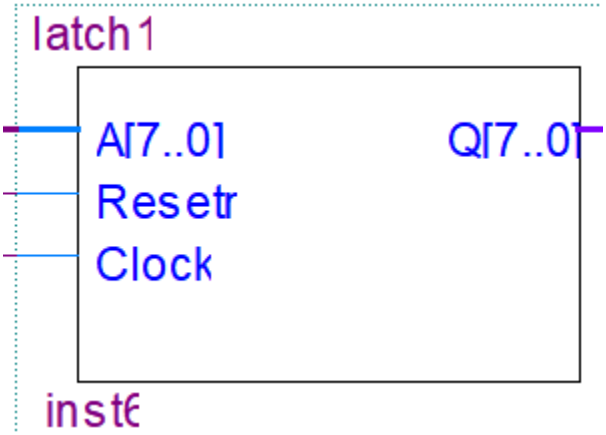
1 x Special Sseg

ALU3 contains 1 more input and output compared to the previous ALU blocks. The special Sseg is a modified version of the original Sseg, that only contains 1 input and output.

## Components:

### Latch

The inputs are A[7..0], Resetn, Clock. The output is Q[7..0]. Clock is used to sync the operations of all blocks in the diagram, so that all operations are performed at a clock value of '1'. Resetn is used to set the output value to 0 if needed. A[7..0] is the manual input that is output through Q[7..0] if Resetn and Clock have a value of 1. The latch block stores the input value until the next HIGH clock. In this case, it stores the value input in A[7..0] until a value of 1 is fed into the Clock input.

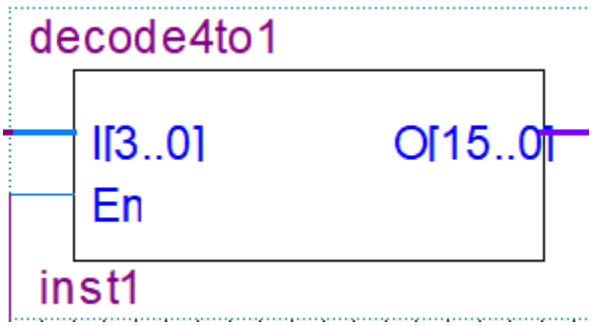



VHDL:

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

ENTITY latch1 IS
      PORT ( A:IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                        Resetn, Clock : IN STD_LOGIC;
                        Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));

END latch1;
ARCHITECTURE Behavior OF latch1 IS
      BEGIN
            PROCESS ( Resetn, Clock )
            BEGIN
                  IF Resetn = '0' THEN
                        Q <= "00000000";
                  ELSIF Clock'EVENT AND Clock='1' THEN
                        Q <= A;
                  END IF;
            END PROCESS;
END Behavior;
```

## Decoder

The inputs are I[3..0], En. The output is O[15..0]. I[3..0] is the 4-bit input taken from FSM. En controls the output of the block, which will only exist if En has a value of 1. The 16-bit output O[15..0] is determined by the combined En and I[3..0]. The decoder interprets the 4-bit input from FSM and converts it into a 16 bit output, which is used as input for the ALU block. Only an output value of 1-9 were coded in the VHDL code, since the ALU selection is from 1-9. Other possible values are caught in the WHEN OTHERS function and are given an 16-bit output value of 0.



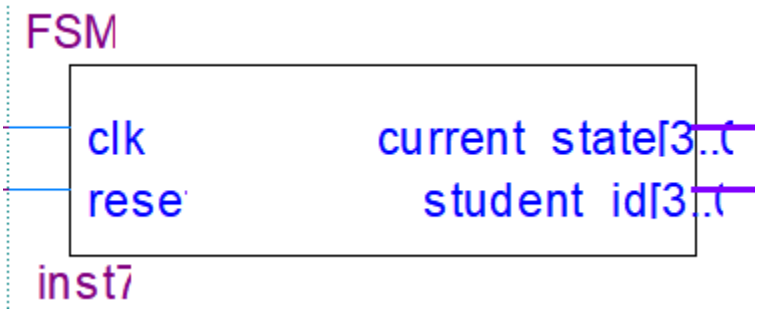

VHDL:

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY decoder IS
PORT ( w : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
En : IN STD_LOGIC ;
y : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) ) ;
END decoder ;
ARCHITECTURE Behavior OF decoder IS
SIGNAL Enw : STD_LOGIC_VECTOR(4 DOWNTO 0) ;
BEGIN
Enw <= En & w ;
WITH Enw SELECT
y <= "0000000000000001" WHEN "10000",
      "0000000000000010" WHEN "10001",
     "0000000000000100" WHEN "10010",
      "0000000000001000" WHEN "10011",
      "0000000000010000" WHEN "10100",
      "0000000000100000" WHEN "10101",
      "0000000001000000" WHEN "10110",
      "0000000010000000" WHEN "10111",
      "0000000100000000" WHEN "11000",
      "0000000000000000" WHEN OTHERS ;
END Behavior ;
```

## FSM

The inputs are clk (clock) and resetFSM. The outputs are current_state[3..0] and student_id[3..0]. Clk is used to sync all block operations. When resetFSM is HIGH, the state will be reset to first state. The 4-bit output current_state[3..0] is the current function state number in binary, which is fed into the decoder

(4:16). The student_id[3..0] 4-bit output goes into an Sseg block for part 1 and 2, which is directly displayed in the final waveform. In part 3, student_id[3..0] is used by the ALU3 to perform certain calculations. The FSM cycles through 9 total states, outputting one number in the student ID and the state value.



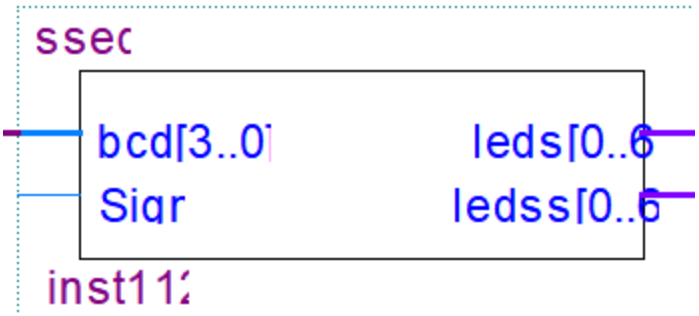

VHDL:

```
library ieee;
use ieee.std_logic_1164.all;

entity FSM is
port(clk,reset:in std_logic;
      current_state,student_id:out std_logic_vector(3 downto 0));
      end FSM;

      architecture FSM of FSM is
      type state_type is (s0,s1,s2,s3,s4,s5,s6,s7,s8);
      signal yfsm:state_type;
      begin
      process(clk,reset)
      begin
            if reset = '1' then
            yfsm<=s0;
            elsif(clk'event and clk='1')then

                  case yfsm is
                  when s0 => yfsm <=s1;
                  when s1 => yfsm <=s2;
                  when s2 => yfsm <=s3;
                  when s3 => yfsm <=s4;
                  when s4 => yfsm <=s5;
                  when s5 => yfsm <=s6;
                  when s6 => yfsm <=s7;
                  when s7 => yfsm <=s8;
                  when s8 => yfsm <=s0;
                  when others => yfsm <=s0;
                  end case;
                  end if;
                  end process;
```

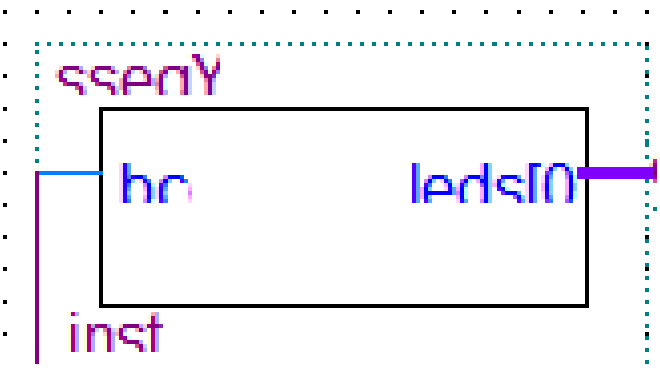
```
process (yfsm)
begin
      case yfsm is
            when s0 =>current_state <="0000";
                  student_id<="0101"; -- 5

            when s1=>current_state<="0001";
                  student_id<="0000"; -- 0

            when s2=>current_state<="0010";
                  student_id<="0001"; -- 1

            when s3=>current_state<="0011";
                  student_id<="0001"; -- 1

            when s4=>current_state<="0100";
                  student_id<="0011"; -- 3

            when s5=>current_state<="0101";
                  student_id<="0111"; -- 7

            when s6=>current_state<="0110";
                  student_id<="0110"; -- 6

            when s7=>current_state<="0111";
                  student_id<="0101"; -- 5

            when s8=>current_state<="1000";
                  student_id<="1001"; -- 9

            when others =>current_state <="1111";
                        student_id <="1111";

      end case;
end process;
end FSM;
```

## SSEG

The inputs are bcd[3..0] and Sign. The outputs are leds[0..6] and ledss[0..6]. bcd[3..0] inputs the 4-bit value that will be displayed on the 7-segment display. The Sign inputs determines if the display segment denoting negative(ledss[0..6]) will be turned on. leds[0..6] outputs the value as it would appear on a 7-segment display. This block converts the 4-bit input into a data that could be read by a 7-segment display.




The Special Sseg (ssegYN) has the same purpose as sseg. The only different is this block only has 1 input and output, which is bcd and leds[0..6]. Depending on the value of bcd, leds[0..6] will output a “y” or “n” as drawn on a 7-segment display.




VHDL:

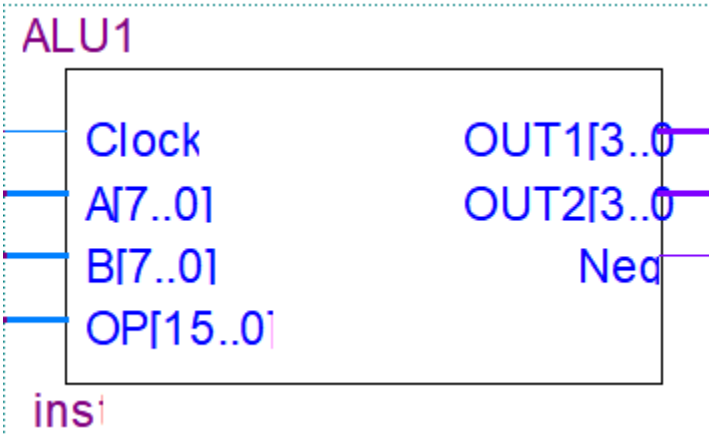
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;


ENTITY sseg IS
    PORT (bcd   : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        leds     : OUT STD_LOGIC_VECTOR(0 TO 6);
               ledss             : OUT STD_LOGIC_VECTOR(0 TO 6);
          Sign     : IN STD_LOGIC);
END sseg;

ARCHITECTURE Behavior OF sseg IS
BEGIN
    PROCESS (bcd, Sign)
    BEGIN

        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 <= "1011011";5
            WHEN "0110" => leds <= "1011111";6
            WHEN "0111" => leds <= "1110000";7
```

```
            WHEN "1000" => leds <= "1111111";8
            WHEN "1001" => leds <= "1110011";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;

          IF (Sign = '1') then
                ledss <= "0000001";
                          else
                                ledss <= "0000000";
          END IF;

    END PROCESS;
END BEHAVIOR;
```

VHDL (Special Sseg):

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


ENTITY ssegYN IS
    PORT (bcd   : IN STD_LOGIC;
          leds     : OUT STD_LOGIC_VECTOR(0 TO 6));
END ssegYN;

ARCHITECTURE Behavior OF ssegYN IS
BEGIN
    PROCESS (bcd)
    BEGIN

        CASE bcd IS
            WHEN '0' => leds <= "0010101"; -- N
            WHEN OTHERS => leds <= "0111011"; -- Y
        END CASE;

    END PROCESS;
END BEHAVIOR;
```

## ALU



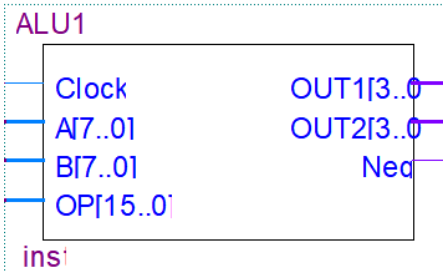

ex. Diagram of ALU1

The ALU block differs between all three parts of lab 6, however all blocks perform arithmetic/logical operations on the 8-bit inputs A[7..0] and B[7..0], depending on the 16-bit input OP[15..0]. All blocks will split the final into two 4-bit sections, which will be output through OUT1[3..0] and OUT2[3..0]. The VHDL codes for each ALU block can be found under the respective parts.

## Note:

Due to the way the clock and reset values are set, the first state of the first cycle will always display an invalid output. Therefore, the value of the first state of the second cycle will be used instead. This applies to all ALU blocks.

## Part 1:



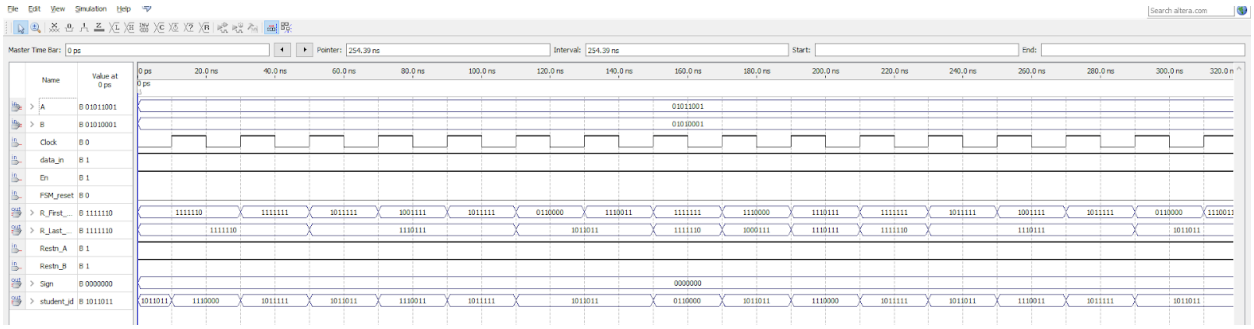

The ALU in part one performs arithmetic and logical operations depending on the inputs provided. Each arithmetic/logical function has a unique selection microcode, which is determined by the input OP. The functions and their microcodes were predetermined by the table below, which was provided in the manual for lab 6:

| 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 | $\overline{A \oplus B}$ |

The input clock is used by the ALU to determine when the perform an operation, and syncs all blocks in the circuit so a change in state (and therefore in ALU operation) occurs at the same time. The A[7..0] and B[7..0] 8-bit inputs accept the values to be used in the operations. The OP[15..0] 16-bit input accepts the 16-bit output from the decoder, which is used to determine the operation to perform using the available inputs.

The OUT1[3..0] and OUT2[3..0] 4-bit outputs each contain a hexadecimal digit that represents the result of the operation performed. The Neg bit output passes either a 0 or 1 to the Sseg, which will be used to display the negative sign on the Sseg if needed.

The student ID displayed is a mix of mine and my partner's ID, 76596551.



## Expected output:

| | #1 | #2 | Hex | Sseg #1 | Sseg #2 |
|---|---|---|---|---|---|
| sum | 1010 | 1010 | AA → | 1110111 | 1110111 ↓ |
| diff | 1000 | 0000 | 8 → | 1111111 | 1111110 |
| inverse | 0110 | 1010 | A6 → | 1011111 | 1110111 |
| NAND | 1110 | 1010 | AE → | 1001111 | 1110111 |
| AND | 0110 | 1010 | A6 → | 1011111 | 1110111 |
| OR | 0001 | 0101 | 51 → | 0110000 | 1011011 |
| XOR | 1001 | 0101 | 59 → | 1110011 | 1011011 |
| XNOR | 1000 | 0000 | 8 → | 1111111 | 1111110 |

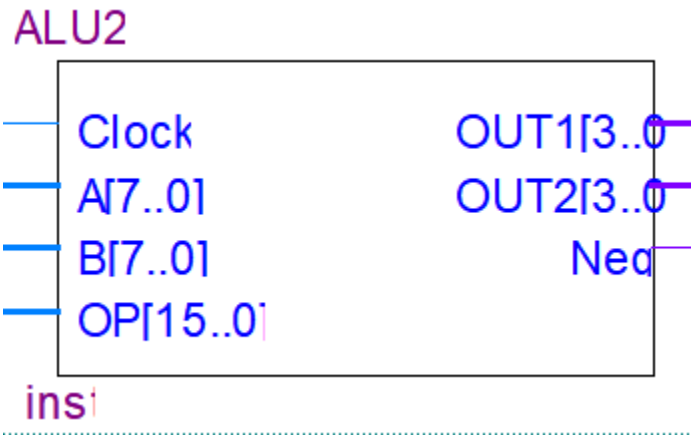
Ignoring the first state in cycle 1, the waveform matches the expected output.

## ALU1 VHDL:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
entity ALU1 is
port( Clock : In std_logic;
        A,B : in unsigned(7 downto 0);
        student_id: in unsigned(3 downto 0);
        OP: in unsigned(15 downto 0);
        Neg: out std_logic;
        R1: out unsigned(3 downto 0);
        R2: out unsigned(3 downto 0));
end ALU1;
architecture calculation of ALU1 is
      signal Reg1,Reg2,Result: unsigned(7 downto 0):=(others =>'0');
signal reg4: unsigned(0 to 7);
begin
```

```
      Reg1 <= A;
      Reg2 <=B;
process(Clock,OP)
begin
    if(rising_edge(Clock)) then
        case OP is
            when "0000000000000001"=>--add
            Result <= Reg1 + Reg2;
            when "0000000000000010"=>--sub
            Result <= Reg1 - Reg2;
            if reg1 > reg2 then
                neg <= '0';
            else
                neg <= '1';
            end if;

            when "0000000000000100"=>--inverse
            Result <= NOT Reg1;
            when "0000000000001000"=>--boolean NAND
            Result <= Reg1 NAND Reg2;
            when "0000000000010000"=>--boolean NOR
            Result <= Reg1 NOR Reg2;
            when "0000000000100000"=>--Boolean AND
            Result <= Reg1 AND Reg2;
            when "0000000001000000"=>--Boolean OR
            Result <= Reg1 OR Reg2;
            when "0000000010000000"=>--Boolean XOR
            Result <= Reg1 XOR Reg2;
            when "0000000100000000"=>--Boolean XNOR
            Result <= Reg1 XNOR Reg2;
            when others =>
            Result <= "00000000";
        end case;
    end if;
end process;
R1 <= Result(3 downto 0);
R2 <= Result (7 downto 4);
end calculation;
```

## Part 2:



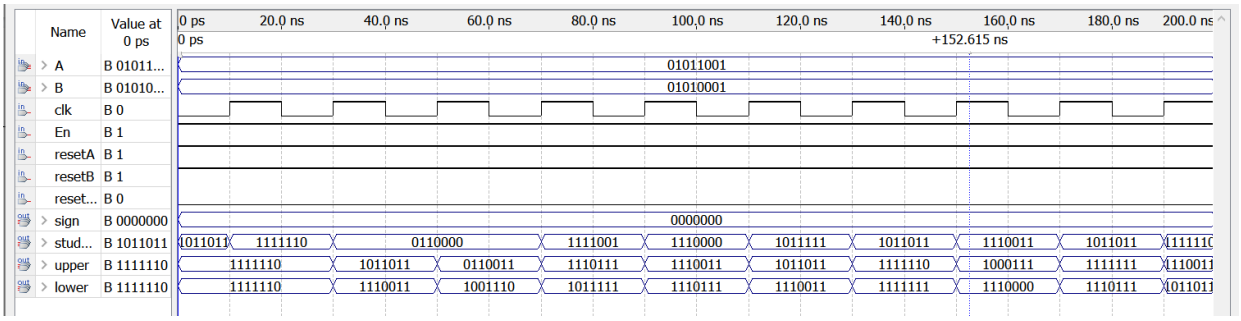

The ALU in part two performs arithmetic and logical operations depending on the inputs provided. Each arithmetic/logical function has a unique selection microcode, which is determined by the input OP. The functions were determined by the table below, which was one of eight available function tables in the manual for lab 6:

**b)**

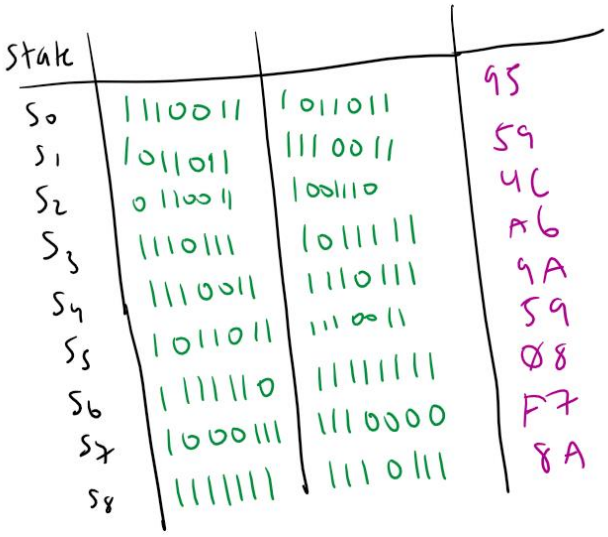
| Function # | Operation / Function |
|---|---|
| 1 | Swap the lower and upper 4 bits of **A** |
| 2 | Produce the result of ORing **A** and **B** |
| 3 | Decrement **B** by 5 |
| 4 | Invert all bits of **A** |
| 5 | Invert the bit-significance order of **A** |
| 6 | Find the greater value of **A** and **B** and produce the results ( Max(**A**,**B**) ) |
| 7 | Produce the difference between **A** and **B** |
| 8 | Produce the result of XNORing **A** and **B** |
| 9 | Rotate **B** to left by three bits (ROL) |

All inputs and outputs of ALU2 function the same way as ALU1. The only change between ALU1 and ALU2 were in the VHDL code that controlled the logical/arithmetic operations performed by the ALU block. The student ID output in this waveform is my student number, not the mix from part 1.

## Output:



## Expected output:




Caluculations:

1. 01011001
Results: 1001 0101
9 5

2. 01011001 OR
01010001
01011001
5 9

3. 01011001
- 00000101
01010000
0101 = 5
4 C

4. 01011001
10100110
A 6

5. 01011001
msb lsb
1 2 3 4 5 6 7 8
1001 1010
9 A

6. Max (A,B)
= A
01011001
5 9

7. (A-B)
01011001 - 01010001
01011001
+ 10101111
X 0 0 0 0 1 0 0 0
Over Flow ϕ 8

10101110
+ 1
10101111

8. A XNOR B

XNOR

| A | B | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

01011001
⊙ 01010001
11110111
F 7

9. Rotate B 3 bits
01010001
DIII
10001010
8 A

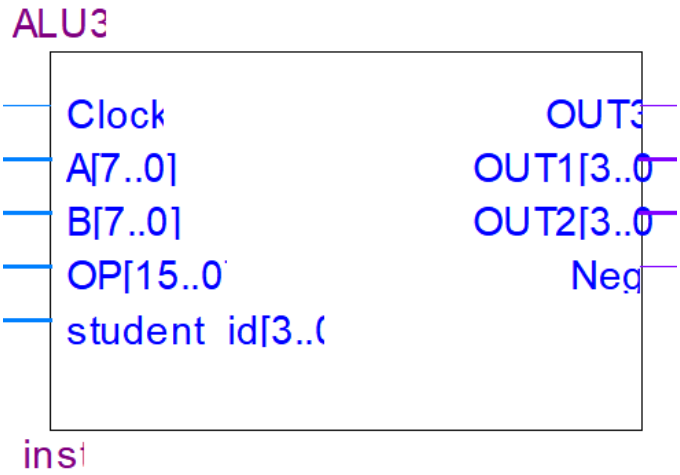
The waveform matches the calculated output.

### ALU2 VHDL:

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

ENTITY ALU2 IS
PORT (Clock : in STD_LOGIC; -- in clock
      A, B : in unsigned(7 DOWNTO 0); -- 8 bit in from A B
      OP : in unsigned(15 DOWNTO 0); -- 16 bit in from decode
      OUT1 : out unsigned(3 DOWNTO 0); -- lower 4 bit/8 bit out
      OUT2 : out unsigned(3 DOWNTO 0); -- higher 4 bit/8 bit out
      Neg : out std_logic); -- dawg yo numba is neg or nah

END ALU2;

ARCHITECTURE calc of ALU2 IS -- temp signal
      SIGNAL Reg1, Reg2, Result : unsigned(7 DOWNTO 0) := (others => '0');
      SIGNAL Reg4 : unsigned (0 TO 7);
      BEGIN
            Reg1 <= A; -- store A in Reg1
            Reg2 <= B; -- store B in Reg2

process(Clock,OP)
BEGIN
      IF(rising_edge(Clock)) THEN -- do calc @ pos clock
            CASE OP IS
                  WHEN "0000000000000001" => -- swap upper lower of A
                        Result <= Reg1(3 downto 0) & Reg1(7 downto 4);
                  WHEN "0000000000000010" => -- A OR B
                        Result <= (Reg1 OR Reg2);
                  WHEN "0000000000000100" => -- B-5
                        IF (Reg2 < ("00000101")) THEN
                              Neg <= '1';
                        ELSE
                              Neg <= '0';
                        END IF;
                        Result <= (Reg2-"00000101"); -- (-5)

                  WHEN "0000000000001000" => -- invert A
                        Result <= not(A);
                  WHEN "0000000000010000" => -- invert bit sig A
                        Result <= Reg1(0)& Reg1(1)& Reg1(2)& Reg1(3)&
Reg1(4)& Reg1(5)& Reg1(6)& Reg1(7);
                  WHEN "0000000000100000" => -- max A,b
                        if (Reg1 > Reg2) then
```

```
                Result <= Reg1;
            else
                Result <= Reg2;
            end if;
            Neg <= '0';
                  WHEN "0000000001000000" => -- diff A b
                        Result <= Reg1 - Reg2;
            if reg1 > reg2 then
                neg <= '0';
            else
                neg <= '1';
            end if;

                  WHEN "0000000010000000" => -- XNOR
                        Result <= (Reg1 XNOR Reg2);
                        Neg <= '0';
                  WHEN "0000000100000000" => -- rotate left
                        Result <= ROTATE_LEFT (Reg2, 3);
                        Neg <= '0';
                  WHEN OTHERS =>
                        Result <= "11111111";
            END CASE;
      END IF;
END PROCESS;
OUT1 <= Result(3 downto 0);
OUT2 <= Result (7 downto 4);
END calc;
```

## Part 3:

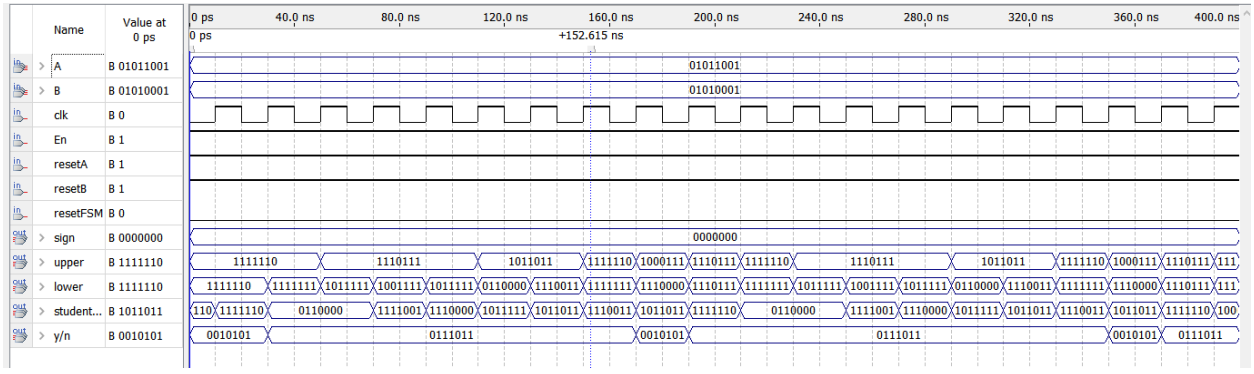



The ALU in part three is a duplicate of the ALU in part one, with the addition that it performs the following instructions:

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

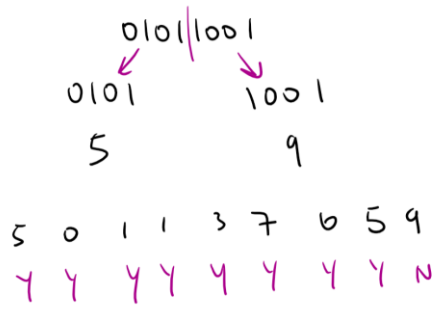
ALU3 maintains the same inputs and outputs as ALU1, with the addition of a student_id[3..0] 4-bit input, and a OUT3 bit input. The student_id[3..0] input is compared with the first and second digit of A by ALU3, to determine the output of OUT3. If the student_id[3..0] of the current state is larger than the first or second digit, the OUT3 will carry a value of 1. If smaller than, the OUT3 will carry a value of 0. The OUT3 bit output serves as the bit input the special Sseg (ssegYN), which will output a 'y' or 'n' on the 7-segment display if the input given is 1 or 0 respectively.

## Output:




## Expected output:

Other than y/n, all other values should match the waveform from part 1.




Aside from the state 1 in cycle 1 (as expected), the waveform matches the expected output.

## ALU3 VHDL:

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

ENTITY ALU3 IS
PORT (Clock : in STD_LOGIC; -- in clock
      A, B : in unsigned(7 DOWNTO 0); -- 8 bit in from A B
      OP : in unsigned(15 DOWNTO 0); -- 16 bit in from decode
      student_id : in unsigned(3 downto 0); -- student in

      OUT3 : out STD_LOGIC; -- y or n
      OUT1 : out unsigned(3 DOWNTO 0); -- lower 4 bit/8 bit out
      OUT2 : out unsigned(3 DOWNTO 0); -- higher 4 bit/8 bit out
      Neg : out std_logic); -- dawg yo numba is neg or nah
END ALU3;

ARCHITECTURE calc of ALU3 IS -- temp signal
SIGNAL Reg1, Reg2, Result : unsigned(7 DOWNTO 0) := (others => '0');
BEGIN
      Reg1 <= A; -- store A in Reg1
      Reg2 <= B; -- store B in Reg2

PROCESS(Clock, OP, Reg1)
BEGIN
      IF(rising_edge(Clock)) THEN -- do calc @ pos clock
            CASE OP IS
                  WHEN "0000000000000001" => -- Reg1 + Reg2
                        Result <= (Reg1 + Reg2);
                        Neg <= '0';
                  WHEN "0000000000000010" => -- Reg1 - Reg 2, neg bit if
need
                        Result <= ( Reg1 - Reg2 );
                        IF (Reg1 > Reg2) THEN
```

```
                              Neg <= '0';
                        ELSE
                              Neg <= '1';
                    end if ;
                  WHEN "0000000000000100" => -- inverse
                        Result <= (not(Reg1));
                        Neg <= '0';
                  WHEN "0000000000001000" => -- NAND
                        Result <= (Reg1 NAND Reg2);
                        Neg <= '0';
                  WHEN "0000000000010000" => -- NOR
                        Result <= (Reg1 NOR Reg2);
                        Neg <= '0';
                  WHEN "0000000000100000" => -- AND
                        Result <= (Reg1 AND Reg2);
                        Neg <= '0';
                  WHEN "0000000001000000" => -- OR
                        Result <= (Reg1 OR Reg2);
                        Neg <= '0';
                  WHEN "0000000010000000" => -- XOR
                        Result <=(Reg1 XOR Reg2);
                        Neg <= '0';
                  WHEN "0000000100000000" =>-- XNOR
                        Result <= (Reg1 XNOR Reg2);
                        Neg <= '0';
                  WHEN OTHERS =>
                        Result <= "--------";
            END CASE;


            IF ((Reg1(7 downto 4) > student_id) OR (Reg1(3 downto 0) >
student_id)) THEN -- if digit larger id (y)
                  OUT3 <= '1';
            ELSE
                  OUT3 <= '0';
            END IF;

      END IF;
END PROCESS;
OUT1 <= Result(3 DOWNTO 0); -- first 4 bit out
OUT2 <= Result(7 DOWNTO 4); -- last 4 bit out

END calc;
```

## Conclusion:

There was a lot of difficulty during part 1, however that was due to calculation error. Instead of calculating the numbers by hand, I decided to use a calculator. The numbers calculated were incorrect and an inappropriate amount of time was wasted trying to figure out why the waveform values did not match the calculated values. Once that issue was resolved, the rest of the lab was relatively easy to solve.