Design of Simple General-Purpose Processor

Thiveyan Nadeswaran
COE 328 - 07
December 3, 2022

Table of Contents

Introduction

The main purpose of this lab was to design and construct a general-purpose processor using the knowledge from the completion of previous lab experiments. 4 distinct components that are constructed in this lab experiment include: an Arithmetic and Logical Unit (ALU), 2 latches, and a control unit which consists of finite state machine (FSM) and 4:16 Decoder. The FSM was implemented with Mealy logic. The results are displayed through the connection of 7-segment displays, which could be viewed in the waveforms.

Components

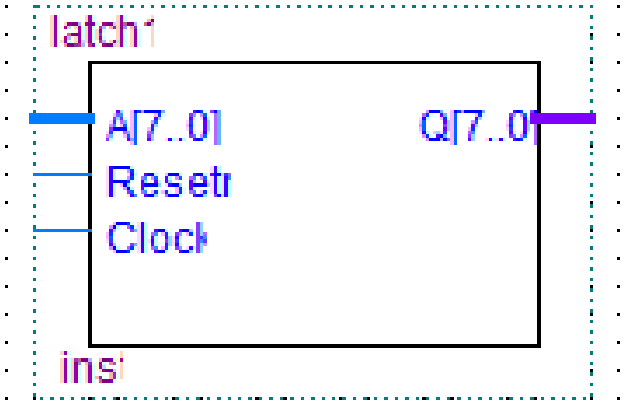
1. **Latch (Latch 1 and Latch 2)**

   The latch takes in a binary number for input and outputs it each cycle, as it is the main component of the storage unit. 2 basic latches are utilized in the circuit, one for each binary number from the two inputs ‘A’ and ‘B’. Both latches are D flip-flops since a clock transition controls them.

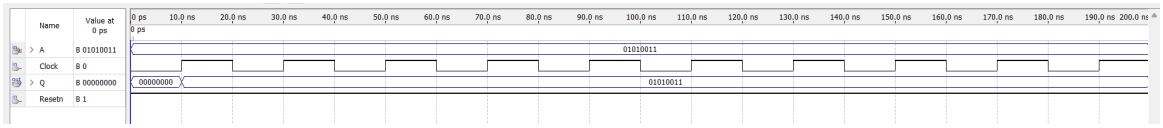
   VHDL code:

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY latch1 IS
5  PORT(A:IN STD_LOGIC_VECTOR(7 DOWNTO 0); --8 bit A input
6  Resetn, Clock:IN STD_LOGIC; --1 bit clock input and 1 bit reset input bit
7  Q: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));--8 bit output
8  END latch1;
9  ARCHITECTURE Behavior OF latch1 IS
10 BEGIN
11 PROCESS (Resetn, Clock)--Process takes reset and clock as inputs
12 Begin
13 IF Resetn = '0' THEN --when reset input is '0' the latches does not operate
14 Q <= "00000000";
15 ELSIF Clock'EVENT AND Clock='1' THEN -- level sensitive based on clock
16 Q <= A;
17 END IF;
18 END PROCESS;
19 END Behavior;
```

   Circuit Diagram:



Waveform for Latch:



In this waveform, the input 53 in 8-bit binary was used. When the clock is 0, the output(Q) is 0. In contrast, when the clock is 1, the output would match A, which would be 53 in 8-bit binary. The truth table for the latch would be shown below:

Truth table for Latch:

Clock	D	Q(t+1)
0	x	Q(t)
1	0	0
1	1	1

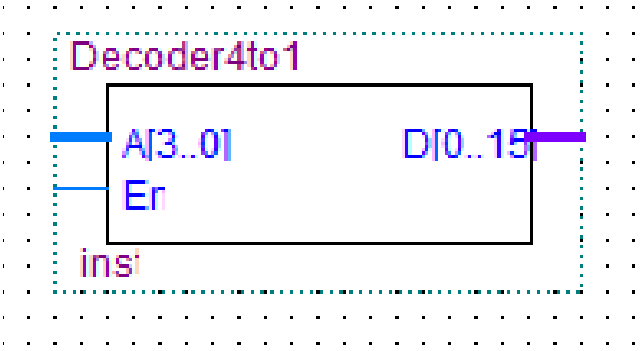
2. 4:16 Decoder

The purpose of the 4:16 Decoder is to take in a 4-bit input from the current state signal in the FSM and output a unique 16-bit binary into the OP signal in ALU. The 4:16 decoder and the FSM are sub-components that make up the control unit of the processor.

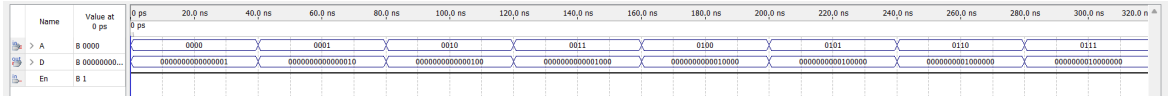
VHDL Code:

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

ENTITY Decoder4to16 IS
PORT (A : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
 En : IN STD_LOGIC ; --enable
 D : OUT STD_LOGIC_VECTOR(0 TO 15) ) ;
 END Decoder4to16;

ARCHITECTURE Behavior OF Decoder4to16 IS
 SIGNAL EnA : STD_LOGIC_VECTOR(4 DOWNTO 0) ;
BEGIN
 EnA <= En & A;
 WITH EnA SELECT
 D <= "0000000000000001" WHEN "10000",
 "0000000000000010" WHEN "10001",
 "0000000000000100" WHEN "10010",
 "0000000000001000" WHEN "10011",
 "0000000000010000" WHEN "10100",
 "0000000000100000" WHEN "10101",
 "0000000001000000" WHEN "10110",
 "0000000010000000" WHEN "10111",
 "0000000100000000" WHEN "11000",
 "0000001000000000" WHEN "11001",
 "0000010000000000" WHEN "11010",
 "0000100000000000" WHEN "11011",
 "0001000000000000" WHEN "11100",
 "0010000000000000" WHEN "11101",
 "0100000000000000" WHEN "11110",
 "1000000000000000" WHEN "11111",
 "0000000000000000" WHEN OTHERS ;
END Behavior ;
```

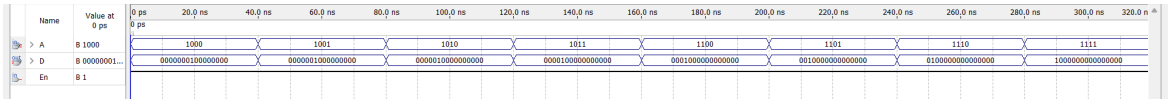
Circuit Diagram of 4:16 Decoder:



Waveform of 4:16 Decoder:
Inputs 0-7:



Inputs 8-15:



Truth table for Decoder:

A(4-bit input)	D(16-bit output)
0000	0000000000000001
0001	0000000000000010
0010	0000000000000100
0011	0000000000001000
0100	0000000000010000
0101	0000000000100000
0110	0000000001000000
0111	0000000010000000
1000	0000000100000000
1001	0000001000000000
1010	0000010000000000
1011	0000100000000000
1100	0001000000000000
1101	0010000000000000
1110	0100000000000000
1111	1000000000000000

3. FSM

The pattern of the controller sequence is decided by the FSM component of the control unit. The FSM acts as an up-counter, where it cycles through states 0 to 8 consecutively and back to state 0 after state 8, as the student_id is displayed on a 7-digit segment. The FSM takes the clock signal as an input and produces 4-bit output current state and it is passed to the decoder sub-component.

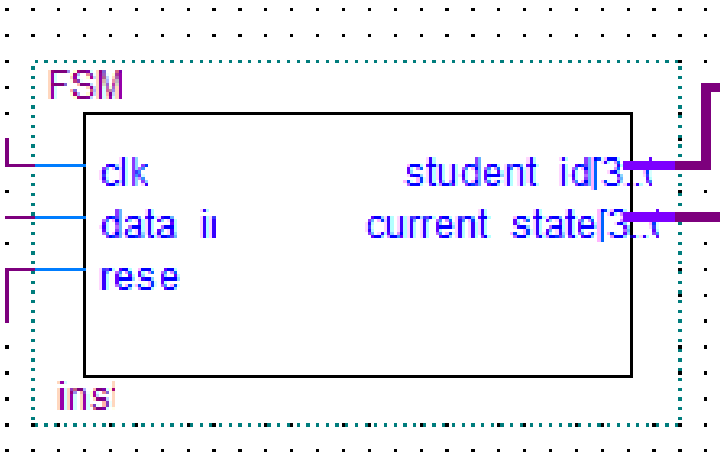
VHDL:

```
library ieee;
use ieee.std_logic_1164.all;
entity FSM 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 Behavior of FSM is
type state_type is(s0, s1, s2, s3, s4, s5, s6, s7, s8);
signal y: state_type;
begin
process(clk, reset)
begin
if reset = '1' then
y <= s0;
elsif(clk'EVENT AND clk = '1') then
case y is
when s0=>
case data_in is
when '0' => y <= s0;
when '1' => y <= s1;
when others =>y<= s0;
end case;
when s1=>
case data_in is
when '0' => y <= s1;
when '1' => y <= s2;
when others =>y<= s1;
end case;
when s2=>
case data_in is
when '0' => y <= s2;
when '1' => y <= s3;
when others =>y<= s2;
end case;
when s3=>
case data_in is
when '0' => y <= s3;
when '1' => y <= s4;
when others =>y<= s3;
end case;
when s4=>
case data_in is
when '0' => y <= s4;
when '1' => y <= s5;
when others =>y<= s4;
end case;
when s5=>
case data_in is
when '0' => y <= s5;
when '1' => y <= s6;
when others =>y<= s5;
end case;
when s6=>
case data_in is
when '0' => y <= s6;
when '1' => y <= s7;
when others =>y<= s6;
end case;
when s7=>
case data_in is
when '0' => y <= s7;
when '1' => y <= s8;
```

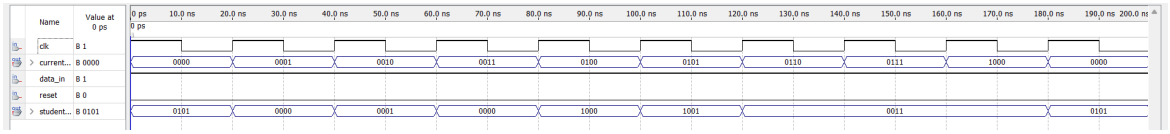
```
when others =>y<= s7;
end case;
when s8=>
case data_in is
when '0' => y <= s8;
when '1' => y <= s0;
when others =>y<= s8;
end case;
end case;
end if;
end process;

process (y, data_in)
begin
case y
is
when s0=>
student_id <= "0101";
current_state <= "0000";
when s1=>
student_id <= "0000";
current_state <= "0001";
when s2=>
student_id <= "0001";
current_state <= "0010";
when s3=>
student_id <= "0000";
current_state <= "0011";
when s4=>
student_id <= "1000";
current_state <= "0100";
```

```
when s5=>
student_id <= "1001";
current_state <= "0101";
when s6=>
student_id <= "0011";
current_state <= "0110";
when s7=>
student_id <= "0011";
current_state <= "0111";
when s8=>
student_id <= "0011";
current_state <= "1000";
end case;

end process;
end Behavior;
```

Circuit Diagram:



Waveform of FSM:



Truth table for FSM:

Present State	Next State		Output Student_id
	data_in = 0	data_in = 1	
0000	0000	0001	0101
0001	0001	0010	0000
0010	0010	0011	0001
0011	0011	0100	0000
0100	0100	0101	1000
0101	0101	0110	0101
0110	0110	0111	0011
0111	0111	1000	0011
1000	1000	0000	0011

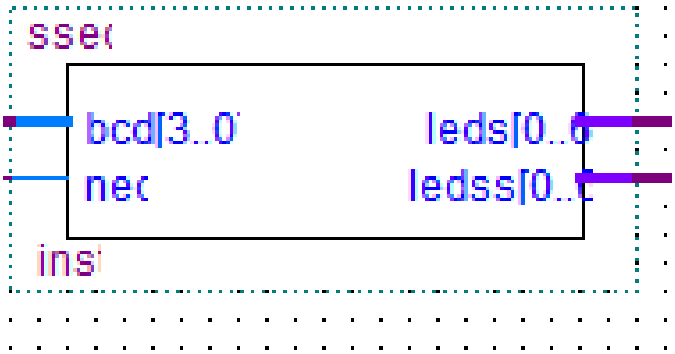
4. **7-segment**

The 7-segment code was the code used in lab experiment 3 that successfully worked. The values 0-15 are displayed in hexadecimal code and the resultant output is a negative value. When neg is set to high (1), the output of ledss is 0000001, otherwise it would be 0000000. The 7-segment was used in all three parts of the ALU, where it was used to display the student_id from the FSM. Additionally, it was used to display the 7-segment of R1 and R2 from the ALU in part 1 and part 2.

VHDL Code:

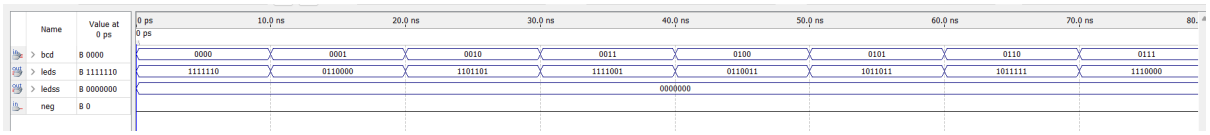
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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

END sseg ;

ARCHITECTURE Behavior OF sseg IS
BEGIN
PROCESS ( bcd )
BEGIN
CASE bcd IS -- abcdefg
WHEN "0000" => leds <= "1111110" ;
WHEN "0001" => leds <= "0110000" ;
WHEN "0010" => leds <= "1101101" ;
WHEN "0011" => leds <= "1111001" ;
WHEN "0100" => leds <= "0110011" ;
WHEN "0101" => leds <= "1011011" ;
WHEN "0110" => leds <= "1011111" ;
WHEN "0111" => leds <= "1110000" ;
WHEN "1000" => leds <= "1111111" ;
WHEN "1001" => leds <= "1110011" ;
--Sequence A-F Below--
WHEN "1010" => leds <= "1110111" ;
WHEN "1011" => leds <= "0011111" ;
WHEN "1100" => leds <= "1001110" ;
WHEN "1101" => leds <= "0111101" ;
WHEN "1110" => leds <= "1001111" ;
WHEN "1111" => leds <= "1000111" ;
WHEN OTHERS => leds <= "-------" ;

END CASE ;
END PROCESS ;

PROCESS (neg)
BEGIN
IF (neg = '1') then
ledss <= "0000001";
ELSE
ledss <= "0000000";
END IF;
END PROCESS;
END Behavior ;
```

Circuit Diagram:

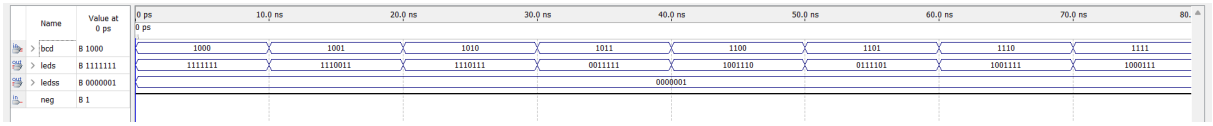


Waveform:

From 0 to 7:



From 8 to F:



Truth Table of 7-segment for positive inputs:

bcd(input)	leds(abcdefg)
0000	1111110
0001	0110000
0010	1101101
0011	1111001
0100	0110011
0101	1011011
0110	1011111
0111	1110000
1000	1111111

1001	1110011
1010	1110111 (A)
1011	0011111 (b)
1100	1001110 (C)
1101	0111101 (d)
1110	1001111 (E)
1111	1000111 (F)

7-segment truth table for negative input:

neg	ledss(abcdefg)
0	0000000
1	0000001

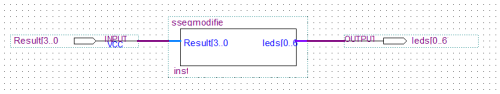
5. Modified 7-Segment (used in ALU part 3)

The main change of the modified 7-segment display is that the new input and output variables would determine whether the displayed output is ‘y’ or ‘n’. In the waveform when Result is set to 1, it would output ‘y’, while it would output ‘n’ if it is 0. The modified 7-segment is only used in ALU part 3.

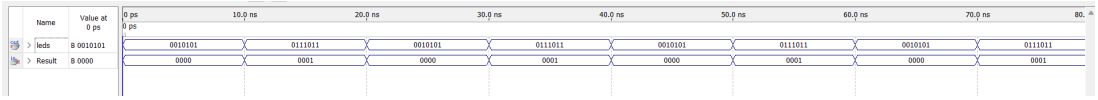
VHDL code:

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity ssegmodified IS
    PORT(
        Result: IN STD_LOGIC_VECTOR(3 downto 0);
        leds: OUT STD_LOGIC_VECTOR(0 to 6));
END ssegmodified;
ARCHITECTURE Behavior OF ssegmodified IS
BEGIN
PROCESS(Result)
    BEGIN
    CASE Result IS
        When "0001" => leds <="0111011"; --yes(y)
        When "0000" => leds <="0010101"; --no(n)
        When OTHERS => leds <="-------";
    END CASE;
    END PROCESS;
END Behavior;
```

Circuit Diagram:



Waveform of ssegmodified:



Truth Table of ssegmodified:

Result(input)	leds(output)
0001	0111011
0000	0010101

Arithmetic Logical Unit (ALU) - Part 1

ALU core is the “heart of every GPA unit”, since all arithmetic and logical operations are implemented and applied as required. The operations that are performed in the ALU are based on the states of the FSM. The ALU has five inputs, which include:

1. Clock
2. A
3. B
4. student_id
5. OP

Besides the 7-segment display and 4:16 decoder, the clock input is shared between all components since they are sequential circuits. The clock alternates between 0 and 1, and ALU will check the OP input when rising edge occurs (input changed to 1 from 0). The inputs used for the ALU parts are based on the last four digit of one’s student number. In this case it would be: 5333. Hence, input A would be 53, while input B would be 33. The student_id does not serve any purpose for ALU.

The ALU has three outputs:

1. Neg
2. R1
3. R2

Neg refers to the negative output of the 7-segment display, where the value would be a negative number when neg equals 1. The 8-bit is split into two 4-bit outputs, R1 and R2 respectively, since the seven digit segment requires 4-bit input.

Block Diagram of the GPU from the lab manual:

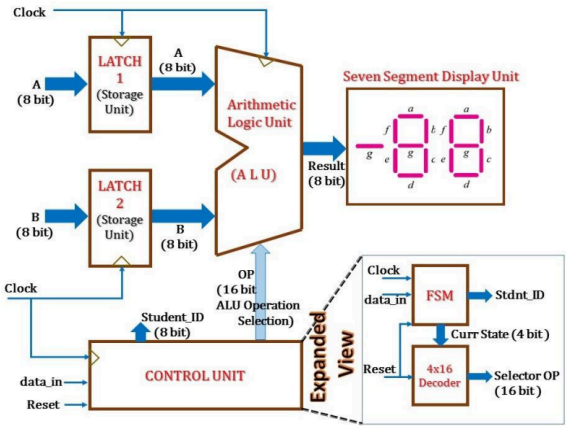


Figure 1. The Block Diagram of the GPU

ALU core operations for Problem 1:

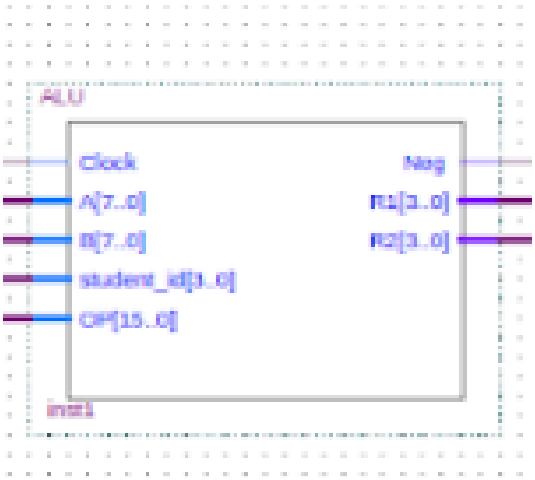
Function #	Microcode	Boolean Operation / Function
1	0000000000000001	sum(A, B)
2	0000000000000010	diff(A, B)
3	0000000000000100	$\overline{A}$
4	0000000000001000	$\overline{A \cdot B}$
5	0000000000010000	$\overline{A + B}$
6	0000000000100000	$A \cdot B$
7	0000000001000000	$A \oplus B$
8	0000000010000000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

VHDL Code:

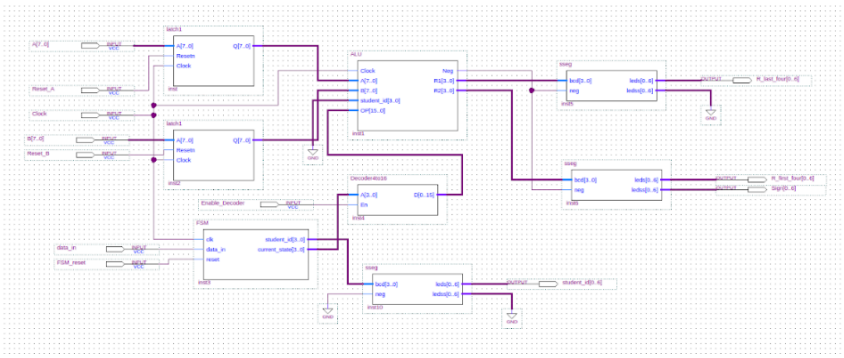
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
entity ALU is
   port(Clock: in std_logic;--input clock signal
      A,B:in unsigned(7 downto 0); --8 bit inputs from latches A and B
      student_id: in unsigned(3 downto 0); --4 bit student id 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 bit output
      R1: out unsigned(3 downto 0); --lower 4-bits of 8-bit Result Output
      R2: out unsigned (3 downto 0)); --higher 4-bits of 8-bit Result Output
end ALU;
architecture calculation of ALU is --temporary signal declaration
   signal Reg1,Reg2,Result:unsigned(7 downto 0):=(others=>'0');
   signal Reg4: unsigned(0 to 7);
begin
   Reg1 <= A; --temporarily store A in Reg1 local variable
   Reg2 <= B; --temporarily store B in Reg2 local variable
process(Clock,OP)
   begin
if(rising_edge(Clock)) THEN --Do the calculation @ positive edge of clock cycle.
   case OP is
   When "0000000000000001" =>
   Result <= (Reg1 + Reg2); --addition
   When "0000000000000010" =>
      if(Reg2>Reg1) then
   Result<=(Reg1+(NOT Reg2+1));
      Neg<='1'; --negative bit
   else
      Result<=(Reg1-Reg2);
      Neg<='0';
   end if;
```

```
   When "0000000000000100" =>
      Result <= (NOT Reg1); --inverse
   When "0000000000001000" =>
      Result <= (Reg1 NAND Reg2); --boolean NAND
   When "0000000000010000" =>
      Result <= (Reg1 NOR Reg2); --boolean NOR
   When "0000000000100000" =>
      Result <= (Reg1 AND Reg2); --boolean AND
   When "0000000001000000" =>
      Result <= (Reg1 XOR Reg2); --boolean XOR
   When "0000000010000000" =>
      Result <= (Reg1 OR Reg2); --boolean OR
   When "0000000100000000" =>
      Result <= (Reg1 XNOR Reg2); --boolean XNOR
   When OTHERS =>
      Result <= "--------";
   end case;
   end if;
   end process;
   R1 <= Result(3 downto 0); --Since the output seven segments can
   R2 <= Result(7 downto 4); --only 4-bits, split the 8-bit to two 4-bits.
end calculation;
```

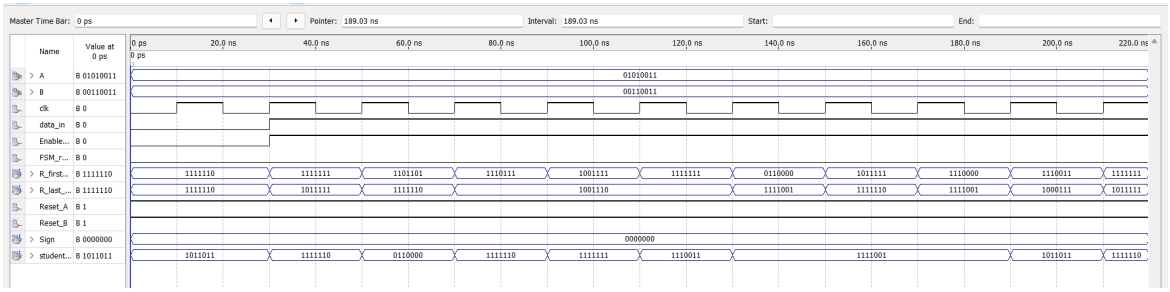
Circuit Diagram of ALU component (part 1):



Circuit Diagram of ALU part 1:



Waveform of ALU part 1:



(Note: R_first and R_last are shifted by one block)

Arithmetic Logical Unit (ALU) - Part 2

The main purpose of the ALU in part 2 is the same as part 1. The ALU would perform different operations based on the state of the FSM and would receive the same input of A(53) and B(33). Problem (a) is selected for part 2.

ALU core operations for Part 2 problem (a):

Function #	Operation / Function
1	Increment **A** by 2
2	Shift **B** to right by two bits, input bit = 0 (SHR)
3	Shift **A** to right by four bits, input bit = 1 (SHR)
4	Find the smaller value of **A** and **B** and produce the results (Min(**A**,**B**))
5	Rotate **A** to right by two bits (ROR)
6	Invert the bit-significance order of **B**
7	Produce the result of XORing **A** and **B**
8	Produce the summation of **A** and **B**, then decrease it by 4
9	Produce all high bits on the output

The new functions would create new results in the waveform. The inputs and outputs of the ALU in part 2 would be the exact same as part 1 as shown below:

- Inputs
    - Clock
    - A
    - B
    - Student_id
    - OP
- Outputs
    - Neg
    - R1
    - R2

Besides the 7-segment display and 4:16 decoder, the clock input is shared between all components since they are sequential circuits. The clock alternates between 0 and 1, and ALU will check the OP input when rising edge occurs (input changed to 1 from 0). The inputs used for the ALU parts are based on the last four digit of one's student number. In this case it would be: 5333. Hence, input A would be 53, while input B would be 33. The student_id does not serve any purpose for ALU.

Neg refers to the negative output of the 7-segment display, where the value would be a negative number when neg equals 1. The 8-bit is split into two 4-bit outputs, R1 and R2 respectively, since the seven digit segment requires 4-bit input.

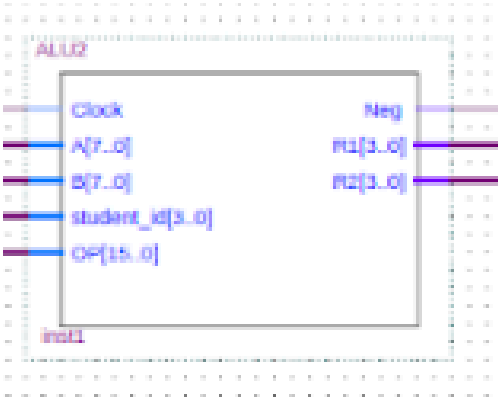
VHDL Code:

```
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;--input clock signal
A,B:in unsigned(7 downto 0); --8 bit inputs from latches A and B
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 bit output
R1: out unsigned(3 downto 0); --lower 4-bits of 8-bit Result Output
R2: out unsigned (3 downto 0)); --higher 4-bits of 8-bit Result Output
end ALU2;
architecture calculation of ALU2 is --temporary signal declaration
signal Reg1,Reg2,Result:unsigned(7 downto 0):=(others=>'0');
signal Reg4: unsigned(0 to 7);
begin
Reg1 <= A; --temporarily store A in Reg1 local variable
Reg2 <= B; --temporarily store B in Reg2 local variable
process(Clock,OP)
begin
if(rising_edge(Clock)) THEN --Do the calculation @ positive edge of clock cycle.
case OP is
When "0000000000000001" => --1
Result <= 2+Reg1; --increment by 2
When "0000000000000010" => --2
--input bit = 0
Result <= shift_right(unsigned(Reg2),2); --shift to the right by 2
When "0000000000000100" => --3
Result <= shift_left(unsigned(Reg1),4); --shift to the right by 4
Result(7) <= '1'; --input bit = 1
Result(6) <= '1';
Result(5) <= '1';
Result(4) <= '1';
```

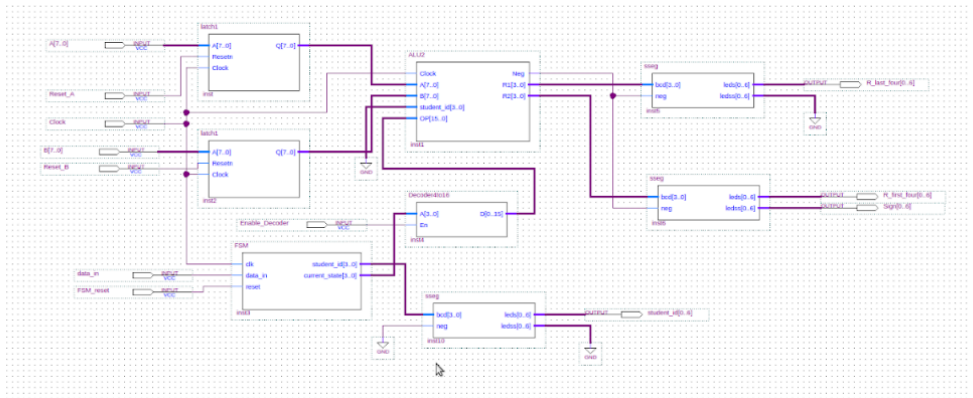
```
When "0000000000001000" => --4
if(Reg2>Reg1) TheN
Result <= Reg1;
else
Result <= Reg2;
end if;
Neg<='0';
When "0000000000010000" => --5
Result <= Reg4; --ROR A by 2
Result(7) <= Reg1(1);
Result(6) <= Reg1(0);
Result(5) <= Reg1(7);
Result(4) <= Reg1(6);
Result(3) <= Reg1(5);
Result(2) <= Reg1(4);
Result(1) <= Reg1(3);
Result(0) <= Reg1(2);
Neg<='0';
When "0000000000100000" => --6
Result <= Reg4; --invert bit significance
Result(7) <= Reg2(0);
Result(6) <= Reg2(1);
Result(5) <= Reg2(2);
Result(4) <= Reg2(3);
Result(3) <= Reg2(4);
Result(2) <= Reg2(5);
Result(1) <= Reg2(6);
Result(0) <= Reg2(7);
Neg<='0';
When "0000000001000000" => --7
Result <= (Reg1 XOR Reg2); --boolean XOR
Neg<='0';
When "0000000010000000" => --8
Result <= ((Reg1 + Reg2)-4); --summation of A and B minus 4
```

```
When "0000000100000000" => --9
Result (7)<= Reg1(7); --all values are set to high bit
Result (6)<= Reg1(7);
Result (5)<= Reg1(7);
Result (4)<= Reg1(7);
Result (3)<= Reg1(7);
Result (2)<= Reg1(7);
Result (1)<= Reg1(7);
Result (0)<= Reg1(7);
Neg<='0';
When OTHERS =>
--don't care, do nothing
Result <= "--------";
end case;
end if;
end process;
R1 <= Result(3 downto 0); --Since the output seven segments can
R2 <= Result(7 downto 4); --only 4-bits, split the 8-bit to two 4-bits.
end calculation;
```

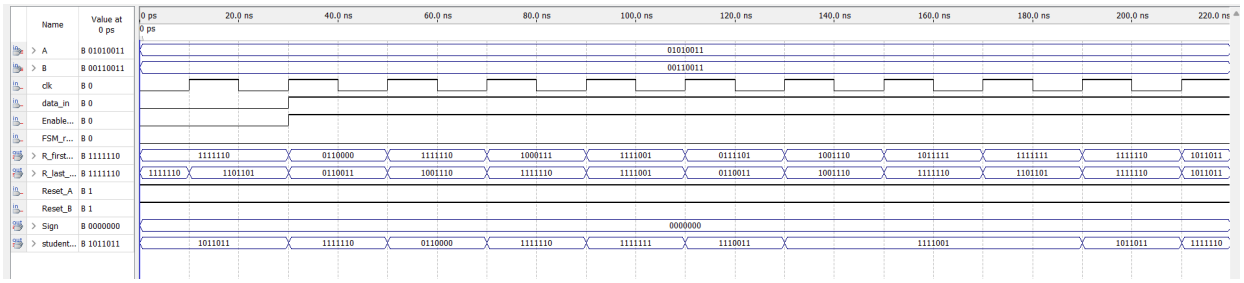
Circuit Diagram of ALU component (part 2):



Circuit Diagram of ALU part 2:



Waveform:



(Note: R_first and R_last are shifted by one block)

Arithmetic Logical Unit (ALU) - Part 3

The main purpose of the ALU in part 3 is to perform required operation based on a problem. There would be modifications in the VHDL code and block diagram for this part. Problem (a) is selected for part 3.

Part 3 problem (a):

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

The final output would either be displayed as ‘y’ (yes) or ‘n’ (no). The inputs and outputs of the ALU in part 3 is different from the previous parts:

- Inputs
    - Clock
    - A
    - B
    - student_id
    - OP

- Output
    - Result

The purpose of the clock, two inputs (A and B), and OP are the same as the previous parts. The student_id serves a purpose in this part. The ALU would check whether the student_id is odd for each state, and would output “0001” if it is odd, and “0000” if it is not odd. The result would be an input for the modified 7-segment code, where the 4-bit would be converted into the 7-segment display for ‘y’ and ‘n’.

ALU core operations for problem 3:

Function	Operation	Student #	Expected Results
1	If student # is odd, ‘y’; else ‘n’	5	y
2	If student # is odd, ‘y’; else ‘n’	0	n
3	If student # is odd, ‘y’; else ‘n’	1	y
4	If student # is odd, ‘y’; else ‘n’	0	n
5	If student # is odd, ‘y’; else ‘n’	8	n
6	If student # is odd, ‘y’; else ‘n’	5	y

7	If student # is odd, ‘y’; else ‘n’	3	y
8	If student # is odd, ‘y’; else ‘n’	3	y
9	If student # is odd, ‘y’; else ‘n’	3	y

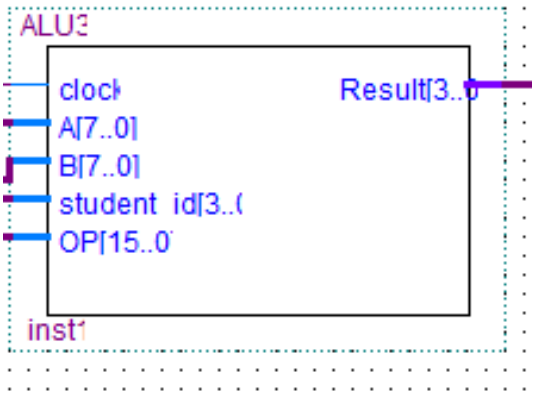
VHDL Code:

```
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;
A,B: in unsigned(7 downto 0);
student_id : in unsigned(3 downto 0);
OP: in unsigned(15 downto 0);
Result : out unsigned(3 downto 0));
end ALU3;
architecture calculation of ALU3 is
signal Res: unsigned(7 downto 0):=(others=>'0');
signal Reg4 : unsigned(0 to 7);

begin
process(clock,OP)
begin
if(rising_edge(clock)) then
Case OP is
when "0000000000000001"=>
if (student_id(0) = NOT'0') then
Res <= ("00000001");
else
Res <= "00000000";
end if;

when "0000000000000010"=>
if (student_id(0) = NOT'0') then
Res <= "00000001";
else
Res <= "00000000";
end if;
```

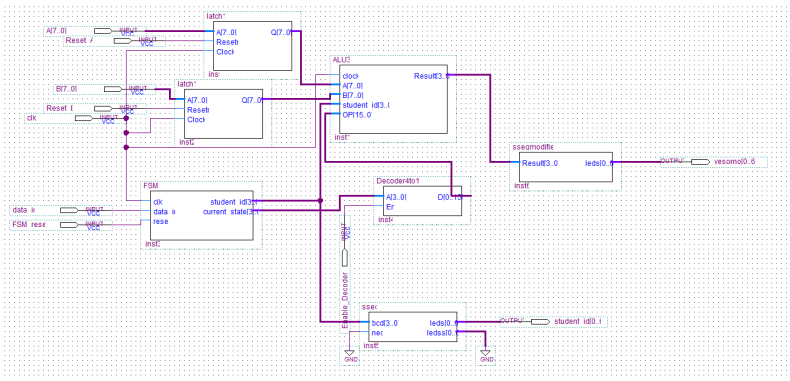
```
when "0000000000000100"=>
if (student_id(0) = NOT'0') then
Res <= "00000001";
else
Res <= "00000000";
end if;

when "0000000000001000"=>--complete 4 if one is greater than the other
if (student_id(0) = NOT'0') then
Res <= "00000001";
else
Res <= "00000000";
end if;

when "0000000000010000"=>
if (student_id(0) = NOT'0') then
Res <= "00000001";
else
Res <= "00000000";
end if;

when "0000000000100000"=>
if (student_id(0) = NOT'0') then
Res <= "00000001";
else
Res <= "00000000";
end if;

when "0000000001000000"=>
if (student_id(0) = NOT'0') then
Res <= "00000001";
else
Res <= "00000000";
```

```
end if;

when "0000000010000000"=>
if (student_id(0) = NOT'0') then
Res <= "00000001";
else
Res <= "00000000";
end if;

when "0000000100000000"=>
if (student_id(0) = NOT '0') then
Res <= "00000001";
else
Res <= "00000000";
end if;

when Others=> Result <= "----";
end case;
end if;
end process;
Result <= Res(3 downto 0);
end calculation;
```

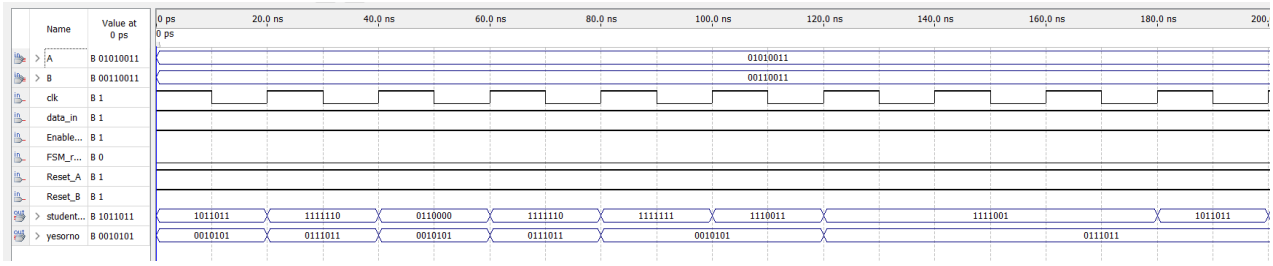
Circuit Diagram of ALU component (part 3):



Circuit Diagram of ALU part 3:



Waveform:



(Note: R_first and R_last are shifted by one block)

Conclusion

In conclusion, the main objective of lab 6 was to create three variations of ALU with the knowledge of previous lab experiments. Each individual components were able to work together to build a processor. The VHDL codes, block diagram, and the waveforms were successfully created and displayed correct values, other than the ALU result being shifted by one block. Hence, the lab was completed successfully.