RYERSON UNIVERSITY

Faculty of Engineering, Architecture and Science

Department of Electrical and Computer Engineering

Course Number	328
Course Title	COE
Semester/Year	F 2020

Instructor	Prof. Vadim

ASSIGNMENT No.

Assignment Title	LAB 6

Submission Date	
Due Date	

Student Name	Apurva Patel
Student ID	500876938
Signature*	A.P

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

Table of Contents :

Introduction :

The purpose of this Lab experiment is to design and construct an Arithmetic and Logic Unit (ALU) in VHDL and implement it. The goal was to create a GPU (General Processor Unit) which contains two storage units. A control unit and an ALU. This will be done with the design and built of all functions of the ALU. In addition, VHDL will be used to design and simulate the ALU based on Functional Simulation in the Quartus Simulator. Latch is used which is essentially a storage unit that temporarily stores the input its given. In addition, Moore FSM is used to pass current state to the decoder and it basically serves the purpose of being a control unit. Furthermore, a 4-16 decoder is used which gets the signal from Moore FSM so that it can pass it down to the ALU core. The ALU core then does all the Arithmetic and Logical operations. Different parts would come together for the creation of a functional ALU. Quartus II version 13.0 was used for this lab.

Components :

a) **Latches :**

2 Latches were used with the purpose of being storage elements. They take an 8 bits input and then send an 8 bits output. In Table 1, the truth table for Latch is displayed. At the bottom, the latches' VHDL code is displayed and after that, Is the latch's schematic block diagram and then it's resulting waveform after compilation.

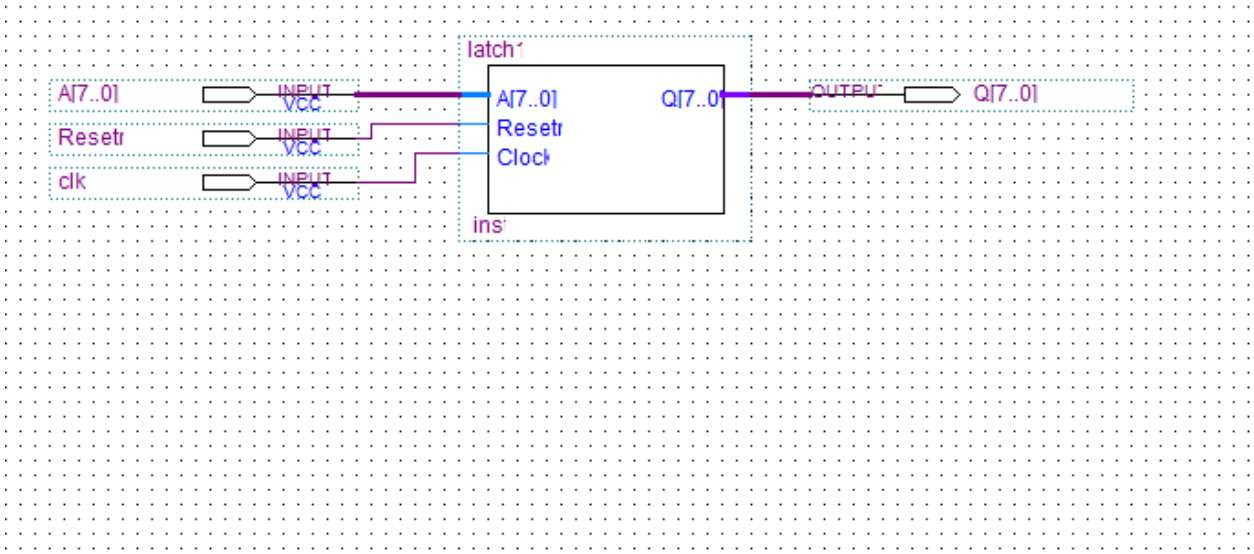
Reset	Clock	A	Q
1	↑	0000 0001	0000 0000
1	↑	0000 0010	0000 0001
1	↑	0000 0011	0000 0010
1	↑	0000 0100	0000 0011
1	↑	0000 0101	0000 0100
1	↑	0000 0110	0000 0101
1	↑	0000 0111	0000 0110
0	↑	0000 1000	0000 0000
0	↑	0000 1001	0000 0000

Table 1 : TruthTable for Latch

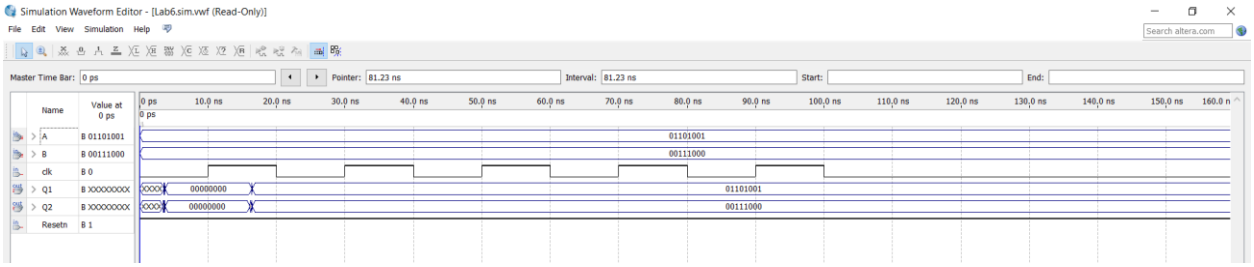
VHDL Code for Latches :

```
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);
6             Resetn, Clock: IN STD_LOGIC;
7             Q: OUT STD_LOGIC_VECTOR(7 downto 0));
8   END latch1;
9
10  Architecture Behavior of latch1 is
11  Begin
12      Process (Resetn, Clock)
13      Begin
14          IF Resetn = '1' THEN
15          Q <= "00000000";
16          ELSIF Clock'EVENT AND Clock = '1' THEN
17          Q <= A;
18          END IF;
19      End Process;
20  End Behavior;
```

BDF for Latches :



Waveform for Latches :



A = 69 = 01101001

B = 38 = 00111000

****Quartus does not generate proper waveforms for my Inputs in my system****

FSM Moore Machine :

Another part of the control unit is an FSM. Below is the state diagram and state table for a Moore FSM. It initially starts at state 0, then for instance, if reset = 1, it moves to the next state if w = 1. If w = 0 however, it stays in the current state. In the FSM VHDL Code, student id 500876938 is used and then displayed in the following waveform. **UPCOUNTER USED FOR THIS PROJECT**

VHDL Code :

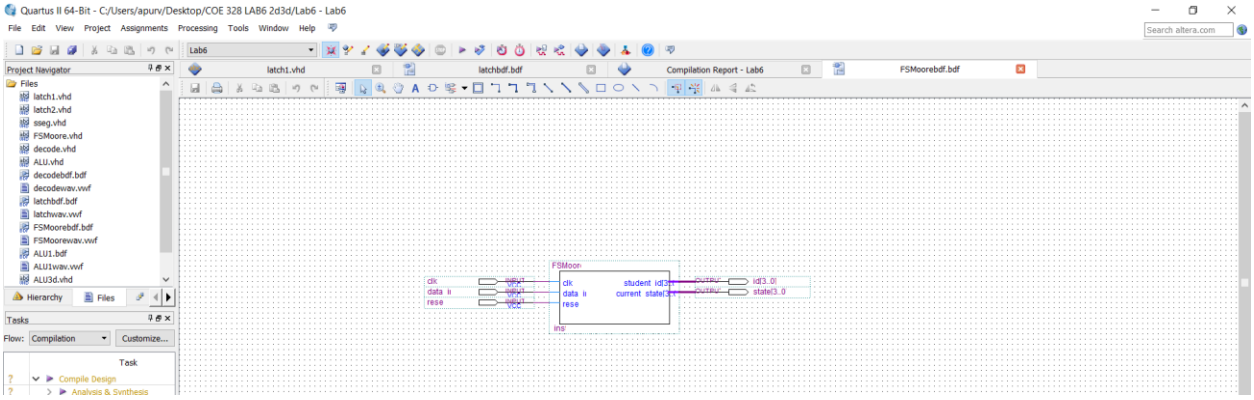
```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
ENTITY FSMoore is
 port (
            clk: IN std_logic;
       data_in : IN std_logic;
         reset : IN std_logic;
    student_id: out std_logic_vector(3 downto 0);
 current_state: out std_logic_vector(3 downto 0));
END ENTITY ;

ARCHITECTURE fsm OF FSMoore 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=>
                  if data_in='0' then
                     yfsm<=s0;
                  else
                     yfsm<=s1;
                  end if;
```

```
               when s1=>
                  if data_in='0' then
                     yfsm<=s1;
                  else
                     yfsm<=s2;
                  end if;
               when s2=>
                  if data_in='0' then
                     yfsm<=s2;
                  else
                     yfsm<=s3;
                  end if;
               when s3=>
                  if data_in='0' then
                     yfsm<=s3;
                  else
                     yfsm<=s4;
                  end if;
               when s4=>
                  if data_in='0' then
                     yfsm<=s4;
                  else
                     yfsm<=s5;
                  end if;
               when s5=>
                  if data_in='0' then
                     yfsm<=s5;
                  else
                     yfsm<=s6;
                  end if;
```

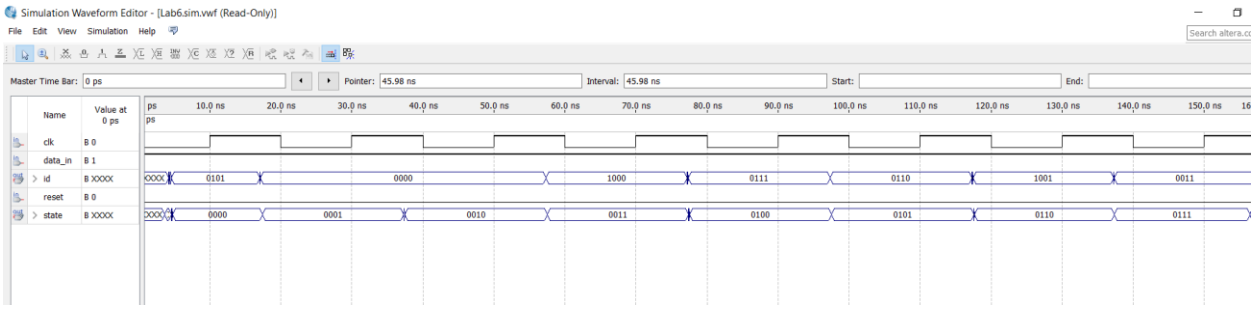
```
59                      when s6=>
60                         if data_in='0' then
61                            yfsm<=s6;
62                         else
63                            yfsm<=s7;
64                         end if;
65                      when s7=>
66                         if data_in='0' then
67                            yfsm<=s7;
68                         else
69                            yfsm<=s8;
70                         end if;
71                      when s8=>
72                         if data_in='0' then
73                            yfsm<=s8;
74                         else
75                            yfsm<=s0;
76                         end if;
77                      end case;
78                   end if;
79                end process;
80                process(yfsm)
81                begin
82                      case yfsm is
83                         when s0=>
84                            student_id<= "0101";
85                            current_state<="0000";
86                         when s1=>
87                            student_id<= "0000";
88                            current_state<="0001";
89                         when s2=>
90                            student_id<= "0000";
91                            current_state<="0011";
```

```
92                          when s3=>
93                              student_id<= "1000";
94                              current_state<="0101";
95                          when s4=>
96                              student_id<= "0111";
97                              current_state<="0111";
98                          when s5=>
99                              student_id<= "0110";
100                             current_state<="1000";
101                         when s6=>
102                             student_id<= "1001";
103                             current_state<="0110";
104                         when s7=>
105                             student_id<= "0011";
106                             current_state<="0100";
107                         when s8=>
108                             student_id<= "1000";
109                             current_state<="0010";
110                     end case;
111                 end process;
112 end fsm;
```

BDF for FSM Moore :



Waveform for FSM :



Decode :

A 4 to 16 decoder was used to get the signal from FSM to ALU core.

VHDL Code :

Quartus II 64-Bit - C:/Users/apurv/Desktop/COE 328 LAB6 2d3d/Lab6 - Lab6

File Edit View Project Assignments Processing Tools Window Help

Project Navigator

Files
- latch1.vhd
- latch2.vhd
- sseg.vhd
- FSMoore.vhd
- decode.vhd
- ALU.vhd
- decodebdf.bdf
- decodewav.vwf
- latchbdf.bdf
- latchwav.vwf
- FSMoorebdf.bdf
- FSMoorewav.vwf
- ALU1.bdf
- ALU1wav.vwf
- ALU3d.vhd

Hierarchy | Files

Tasks

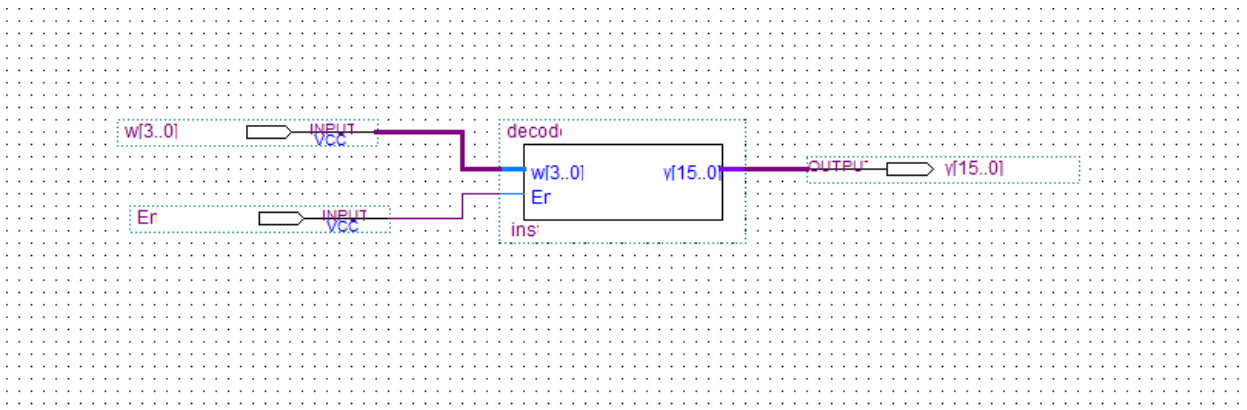
Flow: Compilation | Customize...

Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate progr
TimeQuest Timing Analysis

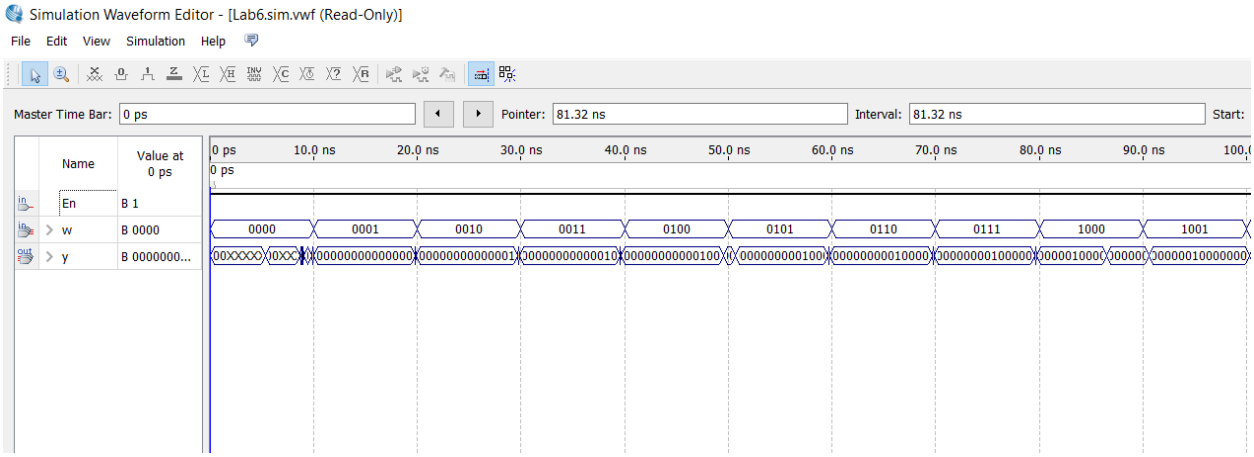
Lab6 | decode.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY decode IS
5      PORT(w   :IN STD_LOGIC_VECTOR(3 DOWNTO 0);
6           En  :IN STD_LOGIC;
7           y   :OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
8  END decode;
9
10 ARCHITECTURE Behavior OF decode IS
11     SIGNAL Enw: STD_LOGIC_VECTOR(4 DOWNTO 0);
12 BEGIN
13     Enw <= En & w;
14     WITH Enw SELECT
15        y <=  "0000000000000001" WHEN "10000",
16              "0000000000000010" WHEN "10001",
17              "0000000000000100" WHEN "10010",
18              "0000000000001000" WHEN "10011",
19              "0000000000010000" WHEN "10100",
20              "0000000000100000" WHEN "10101",
21              "0000000001000000" WHEN "10110",
22              "0000000010000000" WHEN "10111",
23              "0000000100000000" WHEN "11000",
24              "0000000000000000" WHEN OTHERS;
25 END Behavior;
```

BDF for Decode :



Waveform for Decode :



ALU Problem 1 :

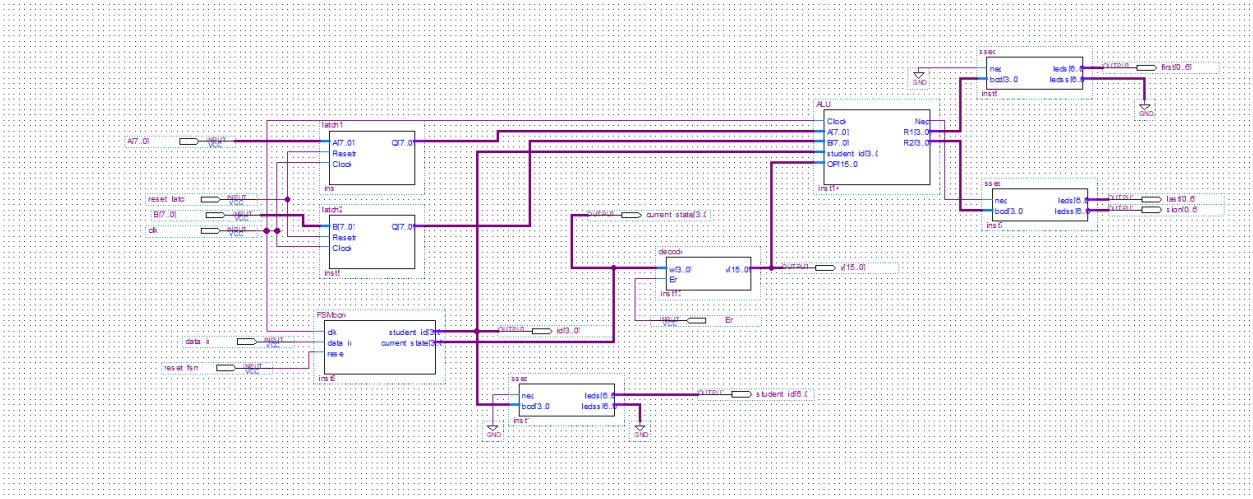
Below is ALU Core VHDL Code. This was used to perform all Logical and Arithmetic operations given to it through the input from the decoder. FSM was used to insert my student number 500898317 and ALU and latch are used. Underneath the VHDL code is the block schematic and waveform with the desired result.

VHDL Code :

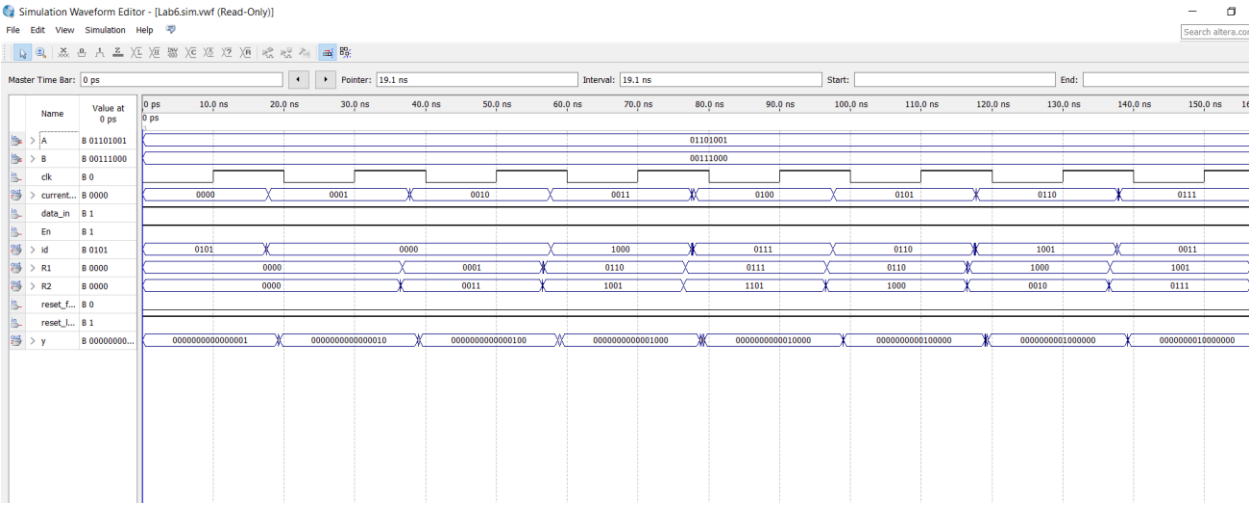
```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
4  USE IEEE.NUMERIC_STD.ALL;
5
6  ENTITY ALU is
7  port ( Clock: in std_logic;
8         A,B: in unsigned(7 downto 0);
9         student_id: in unsigned (3 downto 0);
10        OP: in unsigned (15 downto 0);
11        Neg: out std_logic;
12
13        R1: out unsigned(3 downto 0);
14        R2: out unsigned(3 downto 0));
15 end ALU;
16
17 Architecture calculation of ALU is
18 Signal Reg1,Reg2,Result: unsigned (7 downto 0):=(others => '0');
19 Signal Reg4: unsigned(0 to 7);
20 Begin
21 Reg1 <= A;
22 Reg2 <= B;
23 Process(Clock, OP)
24 Begin
25       if(rising_edge(Clock)) THEN
26          case OP is
27             WHEN "0000000000000001" => Result <=(Reg1 + Reg2);
28             WHEN "0000000000000010" => if (Reg1 > Reg2)then
29                                        Result <= (Reg1 - Reg2);
30                                        Else
31                                        Result <= (Reg2 - Reg1);
32                                        end if;
33             WHEN "0000000000000100" => Result <=(NOT Reg1);
34             WHEN "0000000000001000" => Result <=(NOT(Reg1 AND Reg2));
```

```
34              WHEN "0000000000001000" => Result <=(NOT(Reg1 AND Reg2));
35              WHEN "0000000000010000" => Result <=(NOT(Reg1 OR Reg2));
36              WHEN "0000000000100000" => Result <=(Reg1 AND Reg2);
37              WHEN "0000000001000000" => Result <=(Reg1 OR Reg2);
38              WHEN "0000000010000000" => Result <=(Reg1 XOR Reg2);
39              WHEN "0000000100000000" => Result <=(Reg1 XNOR Reg2);
40              WHEN OTHERS =>
41          end case;
42      end if;
43  end Process;
44  R1 <= Result (3 downto 0);
45  R2 <= Result (7 downto 4);
46  end calculation;
47
```

BDF for ALU 1 :



Waveform for ALU 1 : ALL MY FUNCTIONS ARE WORKING



ALU Problem 2 :

ALU core and its functionalities were modified according to the function underneath. Apart from a modified ALU, same storage elements and control unit is used. The schematic block diagram is displayed underneath and then successful compilation of the VHDL code shows the desired waveform also displayed underneath.

I COULD NOT GET MY ALU 2 VHDL CODE TO WORK BECAUSE OF A SMALL ERROR (WHICH I CANNOT FIGURE OUT). I AM SURE, MY FUNCTIONS IN THE VHDL CODE ARE RIGHT.

Given function **2d** :

d)

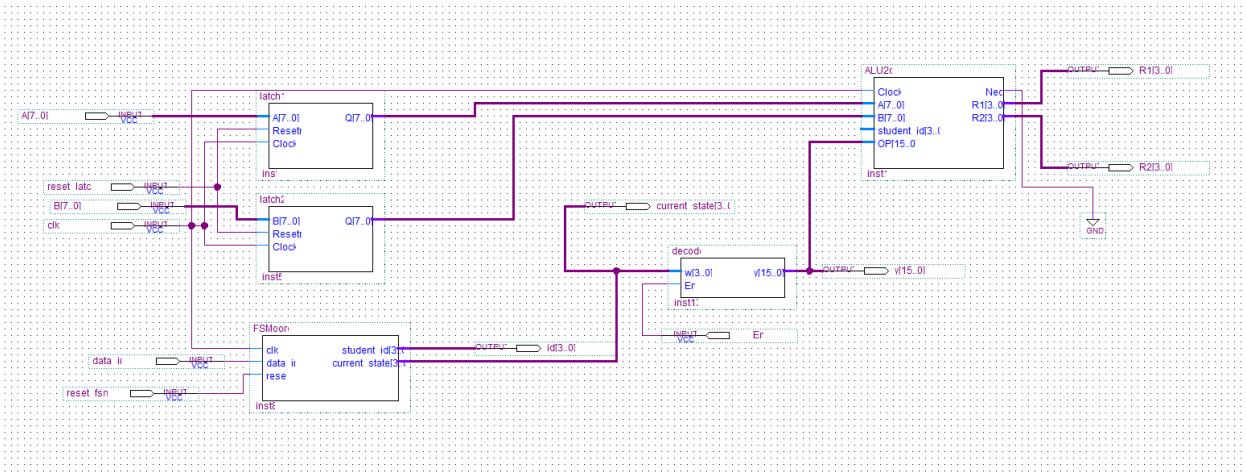
Function #	Operation / Function
1	Shift **A** to right by two bits, input bit = 1 (SHR)
2	Produce the difference of **A** and **B** and then increment by 4
3	Find the greater value of **A** and **B** and produce the results (Max(**A**,**B**))
4	Swap the upper 4 bits of **A** by the lower 4 bits of **B**
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

VHDL for ALU 2 :

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.NUMERIC_STD.ALL;
4
5   ENTITY ALU2d is
6   port ( Clock: in std_logic;
7          A,B: in unsigned(7 downto 0);
8          student_id: in unsigned (3 downto 0);
9          OP: in unsigned (15 downto 0);
10         Neg: out std_logic;
11         R1: out unsigned(3 downto 0);
12         R2: out unsigned(3 downto 0));
13  end ALU2d;
14
15  Architecture calculation of ALU2d is
16     Signal Reg1,Reg2,Result: unsigned (7 downto 0):=(others => '0');
17     Signal Reg4: unsigned(0 to 7);
18     Begin
19     Reg1<= A;
20     Reg2<= B;
21        Process(Clock, OP)
22        Begin
23           if(rising_edge(Clock)) THEN
24              case OP is
25                 WHEN "0000000000000001" => Result<= Reg1 SRL 2;
26                 WHEN "0000000000000010" => if Reg1>Reg2 THEN
27                                               Result<= (Reg1 - Reg2) + 4;
28                                               Neg<= '0';
29                                            else
30                                               Result<= (Reg2 - Reg1) + 4;
31                                               Neg<= '1';
32                                            end if;
```

```
33                WHEN "0000000000000100" => if Reg1>Reg2 THEN
34                                              Result<= Reg1;
35                                           else
36                                              Result<= Reg2;
37                                           end if;
38                WHEN "0000000000001000" => Reg1(7)<= Reg2(3);
39                                           Reg1(6)<= Reg2(2);
40                                           Reg1(5)<= Reg2(1);
41                                           Reg1(4)<= Reg2(0);
42                WHEN "0000000000010000" => Result<= Reg1 + 1;
43                WHEN "0000000000100000" => Result<= (Reg1 AND Reg2);
44                WHEN "0000000001000000" => Reg1(7)<= NOT Reg1(7);
45                                           Reg1(6)<= NOT Reg1(6);
46                                           Reg1(5)<= NOT Reg1(5);
47                                           Reg1(4)<= NOT Reg1(4);
48                WHEN "0000000010000000" => Result<= Reg2 ROL 3;
49                WHEN "0000000100000000" => Result<= "00000000";
50                WHEN OTHERS => Result<= "00000000";
51            end case;
52        end if;
53    end Process;
54    R1 <= Result (3 downto 0);
55    R2 <= Result (7 downto 4);
56end calculation;
57
```

BDF for ALU 2 :



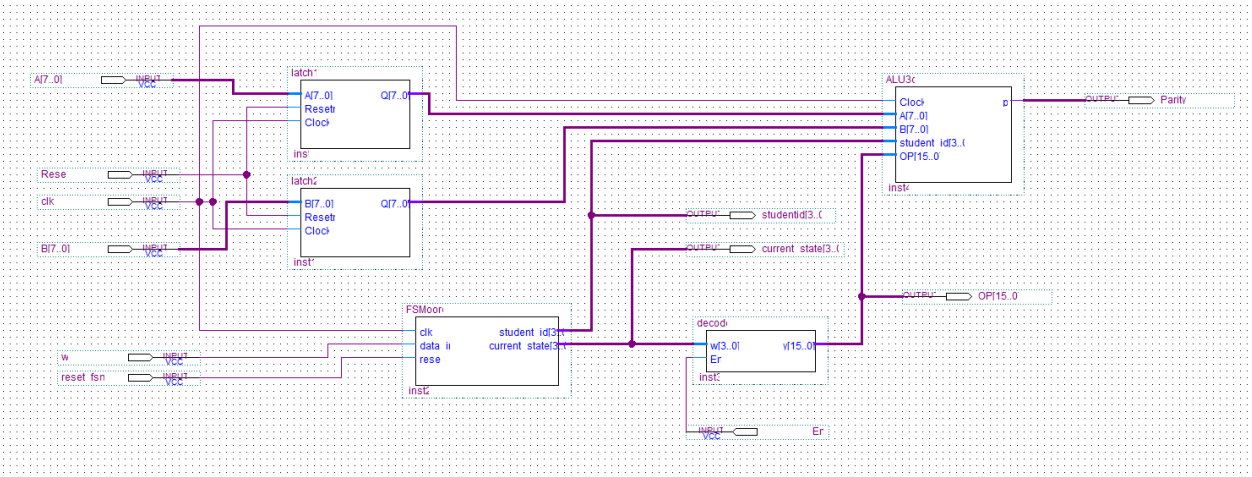
ALU Problem 3 :

For problem 3,the VHDL code for the ALU was modified again to carry out a different function (Function 3D). The assigned function is “For each microcode instruction, display 'y' if the FSM output (student_id) has an even parity and 'n' otherwise”. **For this part 7 segment display is not used. Instead, an output pin is used, which displays ‘1’ if the digit has even parity and ‘0’ otherwise.**

VHDL code for ALU 3d :

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY ALU3d 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);
       p : out std_logic);
end ALU3d ;

Architecture calculation of ALU3d is
Signal id: unsigned (3 downto 0):=(others => '0');
Begin
id <= student_id;
Process(Clock, OP)
Begin
      if(rising_edge(Clock)) THEN
         case OP is
            WHEN "0000000000000001" => p <= ((id(3)) XOR (id(2)) XOR (id(1)) XOR (id(0)));
            WHEN "0000000000000010" => p <= ((id(3)) XOR (id(2)) XOR (id(1)) XOR (id(0)));
            WHEN "0000000000000100" => p <= ((id(3)) XOR (id(2)) XOR (id(1)) XOR (id(0)));
            WHEN "0000000000001000" => p <= ((id(3)) XOR (id(2)) XOR (id(1)) XOR (id(0)));
            WHEN "0000000000010000" => p <= ((id(3)) XOR (id(2)) XOR (id(1)) XOR (id(0)));
            WHEN "0000000000100000" => p <= ((id(3)) XOR (id(2)) XOR (id(1)) XOR (id(0)));
            WHEN "0000000001000000" => p <= ((id(3)) XOR (id(2)) XOR (id(1)) XOR (id(0)));
            WHEN "0000000010000000" => p <= ((id(3)) XOR (id(2)) XOR (id(1)) XOR (id(0)));
            WHEN "0000000100000000" => p <= ((id(3)) XOR (id(2)) XOR (id(1)) XOR (id(0)));
            WHEN OTHERS => p<='0';
         end case;
      end if;
end Process;
end calculation;
```

BDF for ALU 3d :



Waveform for ALU 3d : My system does not give proper waveform, it is disturbed for some reason, but it works correctly on other systems.

Conclusion

From this lab it can be concluded that using VHDL and CAD Tools we can create different Combinational Circuits and Storage Elements which can be used to design a General Processor Unit (GPU). By adjusting the inputs of the combinational circuit (4 to 16 decoder and ALU), we can obtain different combinations of output, which are then used by the ALU to carry out various logic functions. By connecting the inputs of the decoder to the FSM, a particular output can be achieved again after the state machine completes a full cycle. Thus, the ALU can carry out a specific micro-function after the state machine completes a full cycle. The latches used act as storage elements. Thus, they read and store the input during the rising edge and return the input as output values in the following rising edge of the clock cycle. Due to this the ALU produces the required output, but after the clock reaches high value. If the input pins were connected directly to the input ports of the ALU, rather than the 2 latches which are connected to the ALU, there would be no delay, but the input data would not be stored. Furthermore, the ALU can be modified according to any user specific requirements/schematic in order to carry out different logic functions.