CPE166 Advanced Logic Design

Lab 3 Report:

VHDL Structural Design and Usage of Logic Analyzer

Student Name: Andy Cha

Date: 5-01-2020

California State University, Sacramento

TABLE OF CONTENTS

Section	Page
Title Page …….....	1
Table Of Contents …….....	2-3
Introduction …….....	4
Part 1: MHz BCD Counter Design and Logic Analyzer ……...	5
Design Purpose ……...	5
Design Experiments …….	5
Design Source Code ……..	5-6
Simulation Waveform of clk2 and BCD Counter Waveforms …….	7
Result Discussion ……	8
Part 2: Pseudorandom Number Generator and Hamming Code Displays on LEDs ……..	8
Design Purpose ……...	8
Design Experiments …….	9
Design Source Code ……..	9-13
Simulation Waveform ……..…….	13-14
Result Discussion ……	14
Part 3: Calculator Design with Multiplexed Seven Segment Displays …….	14
Design Purpose ……...	14
Design Experiments …….	14-15
Design Source Code ……..	15-20
Simulation Waveform ……..…….	20
Result Discussion ……	21

Part 4: RAM Design and Logic Analyzer …… 21

Design Purpose ……... 21

Design Experiments …… 21

Design Source Code …….. 22-25

Simulation Waveform ……...……. 25

Result Discussion …… 25

Conclusion ……. 26

Introduction

The objective of this lab is to learn structural design using VHDL and learn finite state machine design. Then we will write testbenches in VHDL and learn to simulate our program. Since campus is closed and lab sessions are online, we will not be using the FPGA with Xilinx to monitor our design signals. There will be 4 parts to this lab. The first part of this lab is to design a 1MHz BCD Up-Down Counter. Here we will familiarize ourselves with VHDL and learn the importance of a clock divider. The second part, we will be designing a pseudo random generator using a Linear Feedback Shift Register with Hamming Code. The third part of this lab is to design a LED calculator using the previous modules such as the LFSR and make it perform a calculator function according to the values of button b3 and b4. The final part of this lab is to create a Ram Design that would write binary data "1010" into 16 address locations of the RAM.

Part 1: MHz BCD Counter Design and Logic Analyzer

Design Purpose:

The objective of this part is to design a 1MHz BCD counter that would count from 0 to 9. If the switch is high impedance, then it will begin counting up from 0. If the counter is at 9 and counts up, it will loop to 0. It will also loop back from 0 to 9 if low. We also need to utilize clock division within our design in order to lower the native 100MHz frequency from the FPGA down to 1MHz so that we could visually see the changes happening. If we don't then they will be changing too fast for us to see. So it would take the clock to go 100000000 clock cycles before the clock division becomes '1'.

Design Experiments:

We only need 2 files: bcdcounter(clk div & bcdcounter) and the testbench bcdcounter_tb.

Design Source Code:

bcdcounter.vhdl

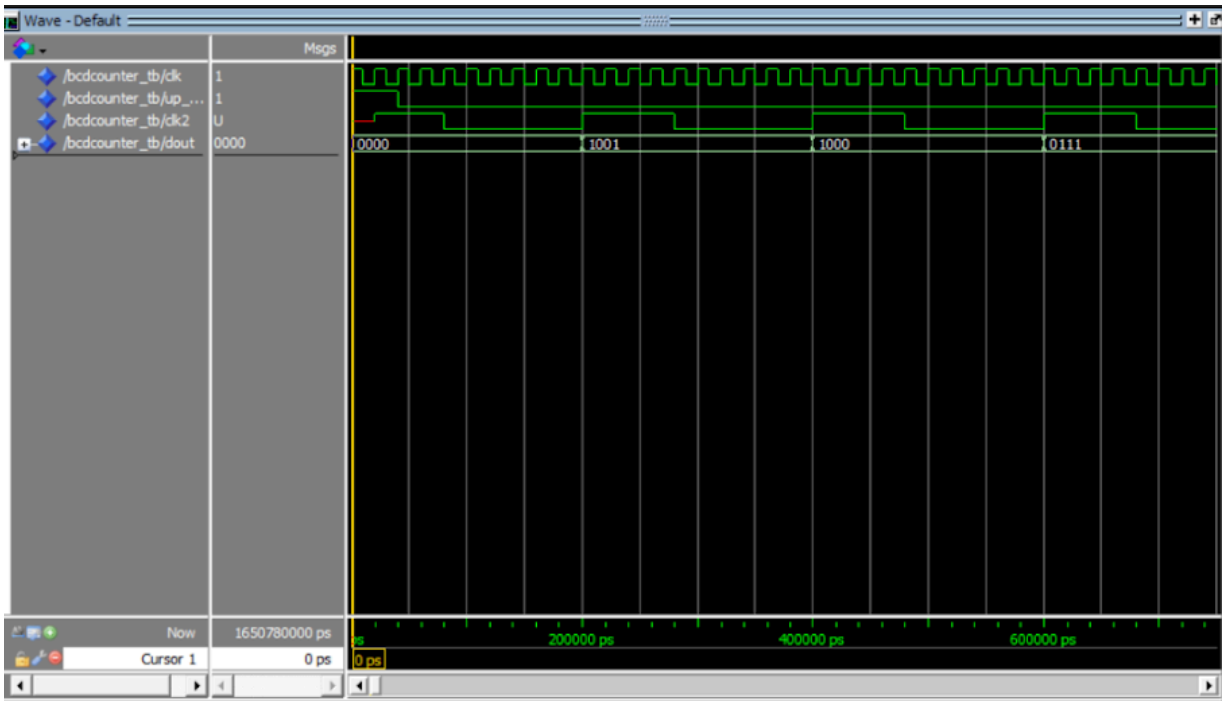
```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY bcdcounter IS

PORT
(
  clk : IN STD_LOGIC;
  up_down: IN STD_LOGIC;
  clk2: OUT STD_LOGIC;
  dout: OUT STD_LOGIC_VECTOR(3 downto 0)
);

END bcdcounter;

ARCHITECTURE beh OF bcdcounter IS

signal cnt_div: std_logic_vector(3 downto 0) := "0000";
signal count: std_logic_vector(3 downto 0) := "0000";
signal tmp: std_logic;

begin

 process (clk)
 begin
 if (rising_edge(clk)) then
    if (cnt_div = 9) then
       tmp <= '1';
       cnt_div <= (others => '0');
    elsif(cnt_div < 3) then
       tmp <= '1';
       cnt_div <= cnt_div + 1;
    else
       tmp <= '0';
       cnt_div <= cnt_div + 1;
    end if;
 end if;
 end process;

 process(tmp, up_down)
 begin
 if rising_edge(tmp) then
    if(up_down = '1') then
       count <= count + 1;
       if (count = "1001") then
          count <= "0000";
       end if;
    else
       count <= count - 1;
       if (count = "0000") then
          count <= "1001";
       end if;
    end if;
 end if;
 end process;

dout <= count;
clk2 <= tmp;

END beh;
```

Testbench:

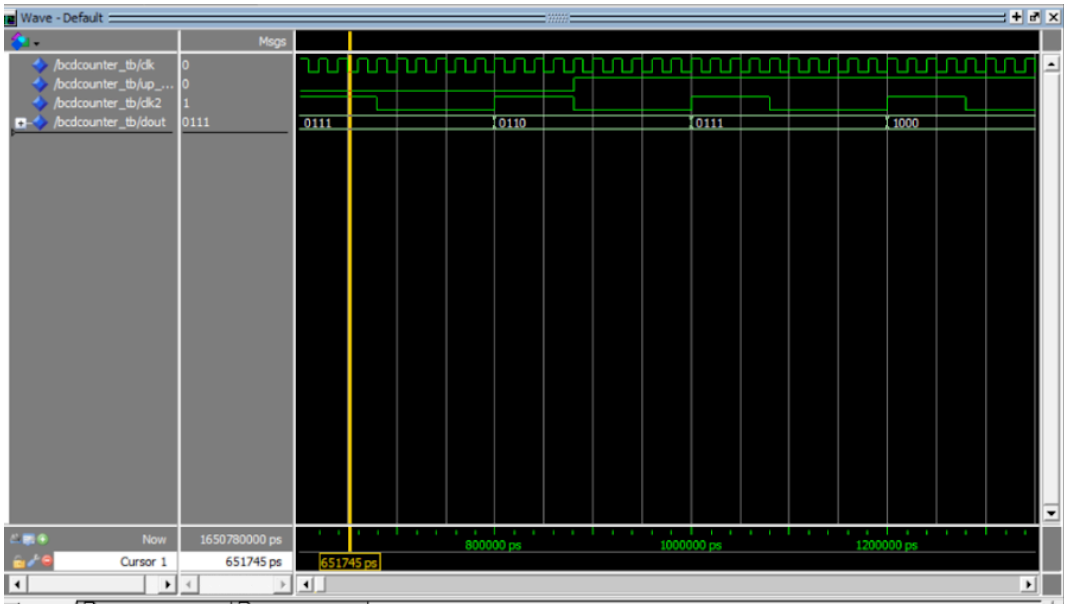
bcdcounter_tb.vhdl

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

ENTITY bcdcounter_tb IS
END bcdcounter_tb;

ARCHITECTURE sim OF bcdcounter_tb IS
component bcdcounter
 PORT(
  clk : IN STD_LOGIC;
  up_down: IN STD_LOGIC;
  clk2: OUT STD_LOGIC;
  dout: OUT STD_LOGIC_VECTOR(3 downto 0)
 );
end component;

signal clk : std_logic := '1';
signal up_down : std_logic := '1';
signal clk2: STD_LOGIC;
signal dout: STD_LOGIC_VECTOR(3 downto 0);

BEGIN
    dev_to_test:  bcdcounter
    port map(clk, up_down, clk2, dout);

    clk_stimulus:  process
    begin
        wait for 10 ns;
        clk <= not clk;
    end process clk_stimulus;

    up_down_stimulus: process
    begin
        wait for 40 ns;
        up_down <= not up_down;
        wait for 800 ns;
    end process up_down_stimulus;

END sim;
```

Simulation Waveform of clk2 and BCD Counter Waveforms:



The waveform above we can see that clk2 is a slower version of clk so we know that the clock division is working. Next we see that ‘dout’ starts as ‘0000’ but because the up_down flip is ‘0’ for the next 3 clock cycles, it loops back to 9 (1001), 8(1000), and then 7(0111).



In this picture we see the ‘up_down’ switch becomes ‘1’ therefore ‘dout’ starts to increment up for each rising clock edge of clk2.

Result Discussion:

When the user would push the ‘up_down’ switch to high, the output ‘dout’ would count up from 0-9. When the switch is low, the counter counts down from 9-0. We can also see that clk2 is a slower version of clk so we know that the clock division works. This part wasn’t too bad because the bcd-counter and clock division code was given to us on the course webpage. We just had to modify the code a little bit to make it work.

Part 2: Pseudorandom Number Generator & Hamming Code Displays on LEDs

Design Purpose:

The objective of part was to design a pseudo random number generator that used a LFSR (linear feedback shift register) of the order X^4 + X + 1. The four D-Flip Flops used in LFSR circuit output 4-bit values. When switch 1 (SW1) is pressed, the finite state machine (FSM) will be in the idle state displaying “0000”. When switch 2 (SW2) is pressed, the LFSR value will be displayed as a 4-bit message. When switch 3 (SW3) is pressed, the (7, 4) hamming code will be constructed and displayed on the LEDs. If the original LFSR 4 bit message is ‘1011’ then the Hamming code would be ‘1010101’ because the Hamming code would be arranged as d4 d3 d2 p3 d1 p2 p1. ‘d’ would be the data and the ‘p’ would be the parity bits.

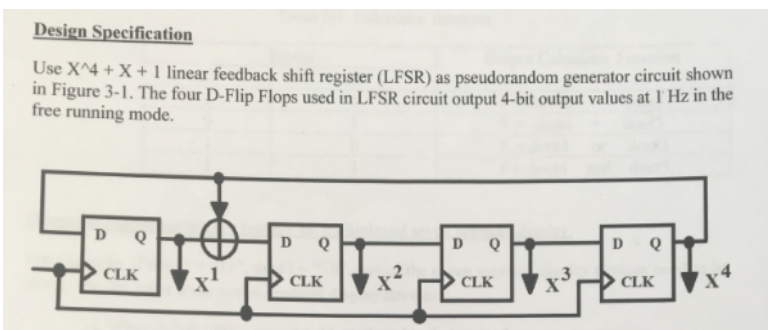


Figure 3-1. X^4 + X + 1 LFSR circuit

Design Experiments:

For this part of the lab, we only need 6 files:

clockDiv - this file will change the original clk cycle to a 1Hz clock cycle

lfsr - this file will do the arithmetic math of the 4 D-flip-flops for the LFSR

hamming - this file will take the 4-bit LFSR and give the Hamming code by determining the data bits(d) and parity bits(p) and output the (7,4) Hamming code arrangement.

fsm - This will determine what dout will be by looking at inputs. The FSM will have 3 states.

Top - Connects clockDiv, lfsr, Hamming, and fsm file to create the circuit design as shown.

Top_tb - testbench for the top file.

Design Source Code:

clockDiv.vhdl

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

ENTITY clockDiv IS
port(
        clock: in std_logic;
        clock2: out std_logic
);
end clockDiv;

architecture beh of clockDiv is
signal cnt_div: std_logic_vector(25 downto 0);

begin

    process(clock)
    begin
        if (rising_edge (clock)) then
            if (cnt_div = 49999999) then
                cnt_div <= (others => '0');
                clock2 <= '1';
            elsif (cnt_div < 24999999) then
                cnt_div <= cnt_div + 1;
                clock2 <= '1';
            else
                cnt_div <= cnt_div + 1;
                clock2 <= '0';
            end if;
        end if;
    end process;
end beh;
```

lfsr.vhdl

```
library ieee;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY lfsr IS
PORT(
    rst, clk: IN STD_LOGIC;
    Q : OUT STD_LOGIC_VECTOR(4 downto 1)
);
END lfsr;

ARCHITECTURE beh OF lfsr IS
signal W: std_logic_vector(4 downto 1) := "1011";

BEGIN
    process( clk, rst )
    begin
    if (rst='1') then
        W <= "1011";
    --W <= ( 1=>'1', others => '0' );
    elsif (rising_edge (clk)) then
        W <= W(3 downto 2) & ( W(1) xor W(4) ) & W(4);
    end if;
    end process;
    Q <= W;
END beh;
```

hamming.vhdl

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

ENTITY hamming IS
port(
        d: in std_logic_vector(4 downto 1);
        ham_out : out std_logic_vector(6 downto 0)
);
end hamming;

architecture beh of hamming is
signal p: std_logic_vector(3 downto 1);

begin

p(1) <= d(4) xor d(2) xor d(1);
p(2) <= d(4) xor d(3) xor d(1);
p(3) <= d(4) xor d(3) xor d(2);
ham_out <= d(4) & d(3) & d(2) & p(3) & d(1) & p(2) & p(1);

end beh;
```

fsm.vhdl

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

entity fsm is
port(
    hamming_in: in std_logic_vector(6 downto 0);
    lfsr_in: in std_logic_vector(3 downto 0);
    fsm_out: out std_logic_vector(6 downto 0);
    clk: in std_logic;
    sw1,sw2,sw3: in std_logic
);
end fsm;

architecture beh of fsm is
type state_type is (s0,s1,s2);
signal state: state_type;

--4bit code for switch 2
signal lfsr_cap:std_logic_vector(3 downto 0);
--8bit code for switch 3
signal hamming_cap:std_logic_vector(6 downto 0);

begin
    state_proc:process (clk,sw1,sw2,sw3)
    begin
        --sw1 blank display
        if(sw1 = '1') then
            state <= s0;

        elsif(rising_edge(clk)) then
            if (sw2 = '1') then
                lfsr_cap <= lfsr_in;
                hamming_cap <= hamming_in;
            end if;
```

```
        case state is
            when s0 =>
                    if (sw2 = '1') then
                            state <= s1;
            else
                            state <= s0;
                    end if;
            when s1 =>
                    if (sw3 = '1') then
                            state <= s2;
                    elsif (sw3 = '0') then
                            state <= s1;
                    end if;
            when s2 =>
                    if (sw2 = '1') then
                            state <= s1;
                    else
                  state <= s2;
                    end if;
            when others =>
                            state <= s0;
        end case;
        end if;
        end process;

output_proc:process(state, lfsr_in, hamming_in)
begin
case state is
    when s0 =>
        fsm_out <= "0000000";
    when s1 =>
        fsm_out <= "000" & lfsr_cap;
    when s2 =>
       fsm_out <= hamming_cap;
end case;
end process;
end beh;
```

top.vhdl

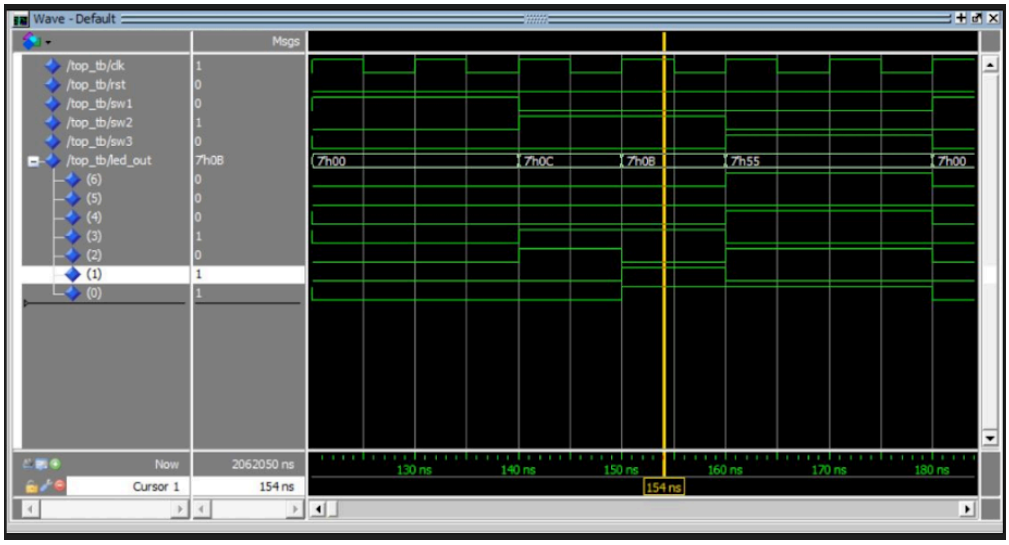
```
library ieee;
use ieee.std_logic_1164.all;

entity top is
port(
    clk: in std_logic;
    rst: in std_logic;
    sw1, sw2, sw3: in std_logic;
    led_out: out std_logic_vector(6 downto 0)
);
end top;

architecture beh of top is
signal one_hz_clk: std_logic;
signal lfsr_out: std_logic_vector(3 downto 0);
signal hamming_out: std_logic_vector(6 downto 0);

component clockDiv
port(
    clock: in std_logic;
    clock2: out std_logic
);
end component;

component lfsr
port(
    clk, rst: in std_logic;
    Q: out std_logic_vector(4 downto 1)
);
end component;

component hamming
port(
    d: in std_logic_vector(4 downto 1);
    ham_out : out std_logic_vector(6 downto 0)
);
end component;

component fsm
port(
    hamming_in:in std_logic_vector(6 downto 0);
    lfsr_in:in std_logic_vector(3 downto 0);
    fsm_out:out std_logic_vector(6 downto 0);
    clk:in std_logic;
    sw1,sw2,sw3:in std_logic
);
end component;

begin
    g1: clockDiv port map(clock => clk, clock2 => one_hz_clk);
    g2: fsm port map( hamming_in => hamming_out, lfsr_in => lfsr_out, fsm_out => led_out, clk => one_hz_clk, sw1 => sw1, sw2 => sw2, sw3 => sw3);
    g3: hamming port map( d => lfsr_out, ham_out => hamming_out);
    g4: lfsr port map(clk => one_hz_clk, rst => rst, Q => lfsr_out);
end beh;
```

top_tb.vhdl

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

ENTITY top_tb IS
END top_tb;

ARCHITECTURE sim OF top_tb IS
component top
 PORT(
   clk : IN STD_LOGIC;
   rst: IN STD_LOGIC;
   sw1, sw2, sw3: IN STD_LOGIC;
   led_out: OUT STD_LOGIC_VECTOR(6 downto 0)
 );
end component;

signal clk: std_logic := '0';
signal rst: std_logic;
signal sw1, sw2, sw3: std_logic;
signal led_out: STD_LOGIC_VECTOR(6 downto 0);

BEGIN
    dev_to_test: top
    port map(clk, rst, sw1, sw2, sw3, led_out);

    clk_stimulus: process
    begin
        clk <= not clk;
        wait for 5 ns;
    end process clk_stimulus;

```

```
    switch_stimulus: process
    begin
    sw1 <= '0'; sw2 <= '0'; sw3 <= '0'; rst <= '1';
    Wait for 20 ns;
    sw1 <= '1'; sw2 <= '0'; sw3 <= '0'; rst <= '0';
    Wait for 50 ns;
    sw1 <= '0'; sw2 <= '1'; sw3 <= '0'; rst <= '0';
    Wait for 50 ns;
    sw1 <= '0'; sw2 <= '0'; sw3 <= '1'; rst <= '0';
    Wait for 50 ns;
    end process switch_stimulus;

END sim;
```

Simulation Waveform:



In the waveform above, we can see that when rst and SW1 is ‘1’ at the start, led_out displays ‘0000’. When SW2 is ‘1’ we see that led_out displays the LFSR message ‘0001011’. When SW3 is ‘1’ led_out displays the hamming code ‘1010101’.

Result Discussion:

The waveform shows that the LFSR and Hamming code works correctly. The hamming code is looking at the even parity to determine if there is an error in the 4 bit message. The LFSR and hamming code was explained and examples were given on the course webpage to help. The hard part was creating the FSM and the top file. The FSM was not producing the correct output and the top file was hard to wire all the files together correctly. Creating the testbench was also a challenge but this part gives good experience with hierarchy design.

Part 3: Calculator Design with Multiplexed Seven Segment Displays

Design Purpose:

This lab introduces an arithmetic logic unit. We also have to change the initial sequence period of 15 clock cycles to a sequence at 1 Hz. This design has 4 switches: switch 1(SW1), switch 2(SW2), switch b3, and switch b4. SW1 will reset the design. When SW2 is high, it enables the LFSR to run so it displays the 4 bit data. When SW2 is low, the LFSR is disabled. b3 and b4 determine the calculator function. This calculator would use a 4-bit pseudo-randomly generated number from the LFSR, split it into two 2-bit numbers, and perform a calculation determined by the user’s input. The calculator performed four functions: add, subtract, compare, and NAND.

Table 3-1. Calculator function

Inputs		Output Calculator Function
b3	b4	
0	0	F = dout4 > dout3
0	1	F = dout4 + dout3
1	0	F = dout4 or dout3
1	1	F = dout4 and dout3

Design Experiments:

This part of the lab we have 7 files:

clock_div - this file will change the original clk cycle to a 1Hz clock cycle

input - this file will determine if the output will be loaded to the fsm or resetted

lfsr - this file will do the arithmetic math of the 4 D-flip-flops for the LFSR

calculator - this will determine the calculator function to perform depending on b3 and b4

fsm - fsm will have 3 states determining the output: clear, display LFSR, or calc. function

top - puts clock_div, input, lfsr, calculator, and fsm file to create the final circuit.

top_tb - testbench for the top file to demonstrate waveform

Design Source Code:

clock_div.vhdl

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

ENTITY clock_div IS
PORT(
      clk: IN STD_LOGIC;
      clkout: OUT STD_LOGIC);
END clock_div;

ARCHITECTURE behavior OF clock_div IS

SIGNAL tmp: integer := 0;

BEGIN
process (clk)
     constant meg100 : integer := 100000000;
     begin
         if (rising_edge(clk)) then
             if (tmp = meg100) then
                 clkout <= '1';
                 tmp <= 0;
             elsif(tmp<(meg100/2 - 1)) then
                 clkout <= '1';
                 tmp <= tmp + 10000000;
             else
                 clkout <= '0';
                 tmp <= tmp + 10000000;
             end if;
         end if;
end process;

END behavior;
```

input.vhdl

```
library IEEE;
use IEEE.std_logic_1164.all;

entity input is

port (
        load, clear: IN std_logic;
        clk: IN std_logic;
        din : IN std_logic_vector(3 downto 0);
        qout : OUT std_logic_vector(3 downto 0)
       );

end input;

architecture beh of input is

begin
process(clk)
     begin
        if (rising_edge(clk)) then
           if (clear = '1') then
              qout <= "0000";
           end if;
           if (load = '1') then
              qout <= din;
           end if;
        end if;
end process;

end beh;
```

lfsr.vhdl

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

ENTITY lfsr IS
PORT(
       clk: IN STD_LOGIC;
       rst: IN std_logic;
       lfsr_out: OUT STD_LOGIC_VECTOR(4 downto 1)
      );
end entity lfsr;

ARCHITECTURE beh OF lfsr IS
signal W: std_logic_vector(4 downto 1) := "0001";

begin
     lfsr_out(1) <= W(4);
     lfsr_out(2) <= W(3);
     lfsr_out(3) <= W(2);
     lfsr_out(4) <= W(1);

process(clk)
begin
     if rising_edge(clk) then
         if (rst ='1') then
                W <= "1000";
             else
                W(1) <= W(4);
                W(2) <= W(1) XOR W(4);
                W(3) <= W(2);
                W(4) <= W(3);
            end if;
     end if;
end process;

end beh;
```

calculator.vhdl

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

ENTITY calculator IS
PORT(
      B3, B4: IN STD_LOGIC;
      dout3, dout4: IN STD_LOGIC_vector(1 downto 0);
      F: OUT STD_LOGIC_vector(1 downto 0)
    );
END calculator;

ARCHITECTURE behavior OF calculator IS

signal tmp : std_logic_vector(1 downto 0);

BEGIN
tmp <= B3 & B4;
process (tmp, dout3, dout4)
    begin
    case (tmp) is
        when "00" => if (dout4 > dout3) then
                        F <= "01";
                    else
                        F <= "00";
                    end if;
        when "01" => F <= (dout4 + dout3);
        when "10" => F <= (dout4 or dout3);
        when "11" => F <= (dout4 and dout3);
        when others => F <= "00";
    end case;
end process;

END behavior;
```

fsm.vhdl

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY fsm IS
PORT(
      SW1, SW2, clk: in std_logic;
      clr, load: out std_logic
    );
END fsm;

ARCHITECTURE beh OF fsm IS

TYPE state IS (s0, s1, s2);
signal cs, ns: state := s0;

BEGIN

process (cs, SW1, SW2)
begin
  case cs is
    when s0 =>
        if (SW2 = '1') then
            ns <= s1;
        else
            ns <= s0;
             end if;

    when s1 =>
        if (SW2 = '0') then
            ns <= s2;
        else
                ns <= s1;
        end if;

    when s2 =>
             ns <= s2;

    when others => ns <= s0;
  end case;
end process;
```

```
41
42  process (clk)
43  begin
44  if(rising_edge(clk)) then
45      if (SW1 = '1') then
46          cs <= s0;
47      else
48          cs <= ns;
49      end if;
50  end if;
51  end process;
52
53  process (cs)
54  begin
55     case cs is
56      when s0 =>
57          clr <= '1';
58          load  <= '0';
59
60      when s1 =>
61          clr <= '0';
62          load  <= '1';
63
64      when s2 =>
65          clr <= '0';
66          load  <= '0';
67
68      when others =>
69          clr <= '1';
70          load  <= '0';
71
72     end case;
73  end process;
74
75  END beh;
```

top.vhdl

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.STD_LOGIC_ARITH.ALL;
4   USE IEEE.STD_LOGIC_UNSIGNED.ALL;
5   use IEEE.NUMERIC_STD.all;
6
7   entity top is
8   port(
9       clock: in std_logic;
10      switch1, switch2, switch3, switch4: in std_logic;
11      lfsr_display : out std_logic_vector(3 downto 0);
12      calc_display : out std_logic_vector(1 downto 0));
13  end top;
14
15  architecture beh of top is
16
17  signal clk2: std_logic;
18  signal reset: std_logic;
19  signal ld: std_logic;
20  signal lfsr_o: std_logic_vector(3 downto 0);
21  signal dff_out: std_logic_vector(3 downto 0);
22
23  component input is
24  port (
25      load, clear: IN std_logic;
26      clk: IN std_logic;
27      din: IN std_logic_vector(3 downto 0);
28      qout: OUT std_logic_vector(3 downto 0)
29     );
30  end component;
31
32  component clock_div IS
33  PORT(
34      clk: IN STD_LOGIC;
35      clkout: OUT STD_LOGIC
36     );
37  END component;
```

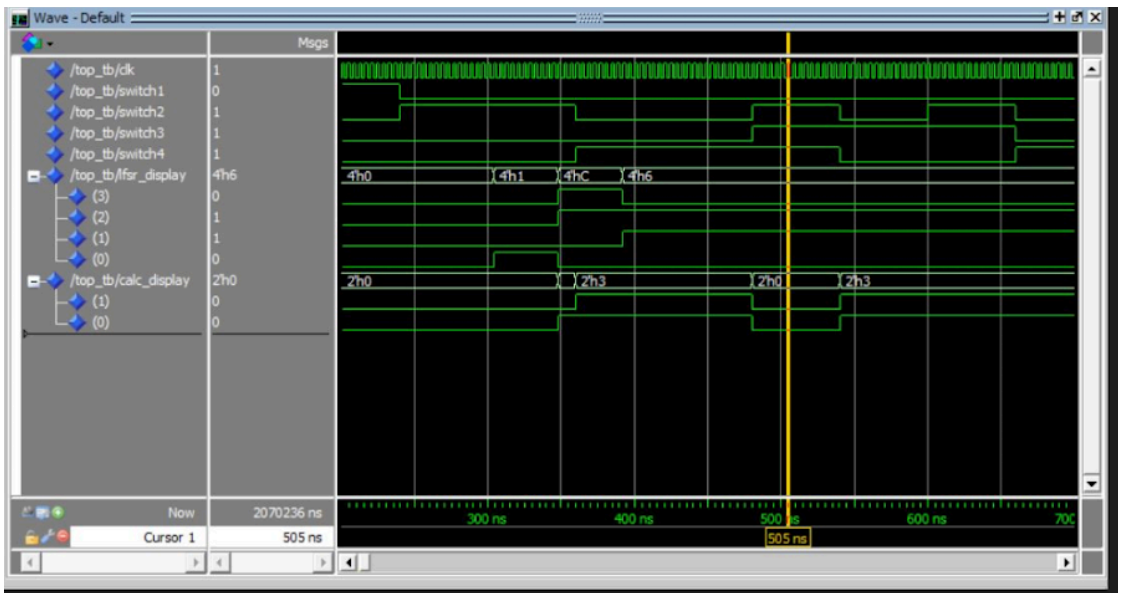
```
component fsm IS
PORT(
      SW1, SW2: in std_logic;
      clk: in std_logic;
      clr, load: out std_logic
     );
END component;

component lfsr IS
PORT(
      clk: IN STD_LOGIC;
      rst: in std_logic;
      lfsr_out: OUT STD_LOGIC_VECTOR(4 downto 1)
     );
end component;

component calculator IS
PORT(
      B3, B4: IN STD_LOGIC;
      dout3, dout4: IN STD_LOGIC_vector(1 downto 0);
      F: OUT STD_LOGIC_vector(1 downto 0)
     );
END component;

begin
lfsr_display <= dff_out;

u1: input port map(load => ld, clear => reset, clk => clk2, din => lfsr_o, qout => dff_out);
u2: clock_div port map(clk => clock, clkout => clk2);
u3: fsm port map(SW1 => switch1, SW2 => switch2, clk => clk2, clr => reset, load => ld);
u4: lfsr port map(clk => clk2, rst => reset, lfsr_out => lfsr_o);
u5: calculator port map(B3 => switch3,B4 => switch4, dout3 => dff_out(1 downto 0), dout4 => dff_out(3 downto 2), F => calc_display);

end beh;
```

top_tb.vhdl

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.all;

ENTITY top_tb IS
END top_tb;

ARCHITECTURE test OF top_tb IS

component top is port(
                clock: in std_logic;
                switch1, switch2, switch3, switch4: in std_logic;
                lfsr_display: out std_logic_vector(3 downto 0);
                    calc_display: out std_logic_vector(1 downto 0)
                      );
end component;

signal clk: std_logic := '0';
signal switch1: std_logic;
signal switch2: std_logic;
signal switch3: std_logic;
signal switch4: std_logic;
signal lfsr_display: std_logic_vector(3 downto 0);
signal calc_display: std_logic_vector(1 downto 0);

BEGIN

uut: top port map (clk, switch1, switch2, switch3, switch4, lfsr_display, calc_display);

clk_stimulus: process
Begin
    CLK <= not CLK;
    wait for 2 ns;
End process clk_stimulus;
```

```
switch_stimulus: process
Begin
switch1 <= '0'; switch2 <= '0'; switch3 <= '0'; switch4 <= '0';
Wait for 120 ns;
switch1 <= '1'; switch2 <= '0'; switch3 <= '0'; switch4 <= '0';
Wait for 120 ns;
switch1 <= '0'; switch2 <= '1'; switch3 <= '0'; switch4 <= '0';
Wait for 120 ns;
switch1 <= '0'; switch2 <= '0'; switch3 <= '0'; switch4 <= '1';
Wait for 120 ns;
switch1 <= '0'; switch2 <= '1'; switch3 <= '1'; switch4 <= '1';
Wait for 60 ns;
switch1 <= '0'; switch2 <= '0'; switch3 <= '1'; switch4 <= '0';
Wait for 60 ns;
switch1 <= '0'; switch2 <= '1'; switch3 <= '1'; switch4 <= '0';
Wait for 60 ns;
switch1 <= '0'; switch2 <= '0'; switch3 <= '0'; switch4 <= '1';
wait;
End process switch_stimulus;

END test;
```

Simulation Waveform:



The waveform above shows the clk, the 4 inputs, and the 2 outputs. We see that when SW2 is on, the LFSR is activated. When SW3 is on we see that the calc_display displays the correct function by looking at switch 3 and switch 4 which is an AND operation because both switches are ‘1’. To see if it is correct, we split the 4-bit LFSR ‘0110’ into 2 bits: ‘01’ and ‘10’. The output is ‘00’ so it’s correct. If we look elsewhere we can see that the other operations are performed as well.

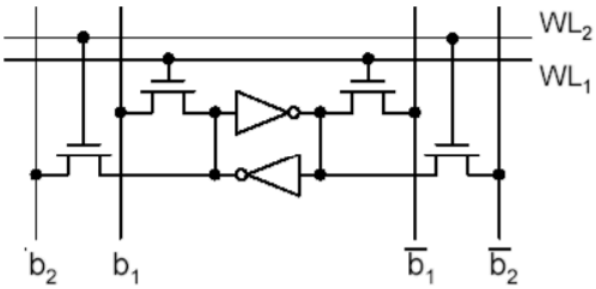
Result Discussion:

The calculator will perform the 4 tasks it is supposed to do depending on the finite state machine. The FSM has 3 states and depending on the input will determine whether or not to clear, display LFSR, or calculator function. This lab used hierarchy design which is always a challenge to put everything together. The final results were simulated correctly. The calculator was able to add, subtract, compare, and NAND two numbers.

Part 4: RAM Design and Logic Analyzer

Design Purpose:

The objective of this part will be to design a RAM. In our RAM design, we will need to write binary data 4'b1010 into 16 address locations of the RAM. After completing writing, we need to read data out of the RAM as well. We will run simulations on the design to show the result.



Truth Table

$\overline{WE}$	$\overline{CS}$	$\overline{OE}$	I/O	Function
X	H	X	High-Z	Standby
H	L	H	High-Z	Output Disabled
H	L	L	Dout	Read Data
L	L	X	DIN	Write Data

Design Experiments:

For this part of the lab, we only need 4 files:

ram - this will determine the function from the truth table with: we, cs, oe

mem_fsm - there will be 5 states to determine the output depending on inputs

mem_top - puts ram and mem_fsm file together to create final circuit

top_tb - testbench for mem_top to display results of ram

Design Source Code:

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

entity ram is

port (
      address: in std_logic_vector (3 downto 0);
      data: inout std_logic_vector (3 downto 0);
      cs: in std_logic;
      we: in std_logic;
      oe: in std_logic;
     );

end ram;

architecture beh_ram of ram is

type memory is array (0 to 15) of std_logic_vector (3 downto 0);
signal mem: memory;

begin

   MEM_WRITE:
      process(address, data, cs, we)
      begin
    if (cs = '1' and we = '1') then
       mem(conv_integer(address)) <= data;
    end if;
      end process;

   MEM_READ:
      process (address, cs, we, oe, mem)
      begin
        if (cs = '1' and we = '0' and oe = '1')  then
       data <= mem(conv_integer(address));
    else
       data <=  (others => 'Z' );
    end if;
      end process;

end beh_ram;
```

mem_fsm.vhdl

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity mem_fsm is
  port (
     clk,reset: IN std_logic;
         address: OUT std_logic_vector(3 downto 0);
     data:INOUT std_logic_vector(3 downto 0);
     cs, we, oe: OUT std_logic
        );
end mem_fsm;

architecture fsm_beh of mem_fsm is

type state_type is (idle, s1, s2, s3, s4);
signal state: state_type;
signal cnt: std_logic_vector(3 downto 0);

begin
     cs <= '1';
         address <= cnt;

  process (clk, reset)
  begin
    if (reset ='1') then
      state <= idle;
      cnt <= "0000";
    elsif (clk = '1' and clk'event) then
      case state is
    when idle =>
        state <= s1;
                cnt   <= "0000";

        when s1 =>
        state <= s2;
                cnt   <= "0000";
```

```
40          when s2 =>
41          cnt <= cnt + 1;
42          if (cnt < 15) then
43             state <= s2;
44          else
45             state <= s3;
46          end if;
47
48          when s3 => state <= s4;
49                 cnt  <= "0000";
50
51       when s4 =>
52          cnt <= cnt + 1;
53          state <= s4;
54
55          when others =>
56          cnt <= "0000";
57          state <= s1;
58
59        end case;
60      end if;
61    end process;
62
63    process(state)
64    begin
65      case state is
66          when idle => we <= '0'; oe <= '0';  data <= "ZZZZ";
67              when s1 => we <= '1'; oe <= '0';  data <= "1010";
68              when s2 => we <= '1'; oe <= '0';  data <= "1010";
69              when s3 => we <= '0'; oe <= '1';  data <= "ZZZZ";
70              when s4 => we <= '0'; oe <= '1';  data <= "ZZZZ";
71          when others => we <= '0'; oe <= '0';   data <= "ZZZZ";
72          end case;
73    end process;
74
75  end fsm_beh;
```

mem_top.vhdl

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3
4   ENTITY mem_top IS
5       port(
6        clk: IN std_logic;
7        reset: IN std_logic;
8        data: inout std_logic_vector ( 3 downto 0);
9        addr_out: OUT std_logic_vector(3 downto 0);
10       cs_out: OUT std_logic;
11       we_out: OUT std_logic;
12       oe_out: OUT std_logic
13      );
14  END mem_top;
15
16  ARCHITECTURE behavior OF mem_top IS
17
18      COMPONENT ram
19      PORT(
20       address: in std_logic_vector ( 3 downto 0);
21       data: inout std_logic_vector ( 3 downto 0);
22       cs:in std_logic;
23       we:in std_logic;
24       oe:in std_logic
25      );
26      end component;
27
28      COMPONENT mem_fsm
29      PORT(
30           clk: IN  std_logic;
31           reset: IN  std_logic;
32           address: OUT  std_logic_vector(3 downto 0);
33       data: INOUT std_logic_vector(3 downto 0);
34           cs: OUT  std_logic;
35           we: OUT  std_logic;
36           oe: OUT  std_logic
37          );
38      END COMPONENT;
39
40  signal address: std_logic_vector (3 downto 0);
41
```

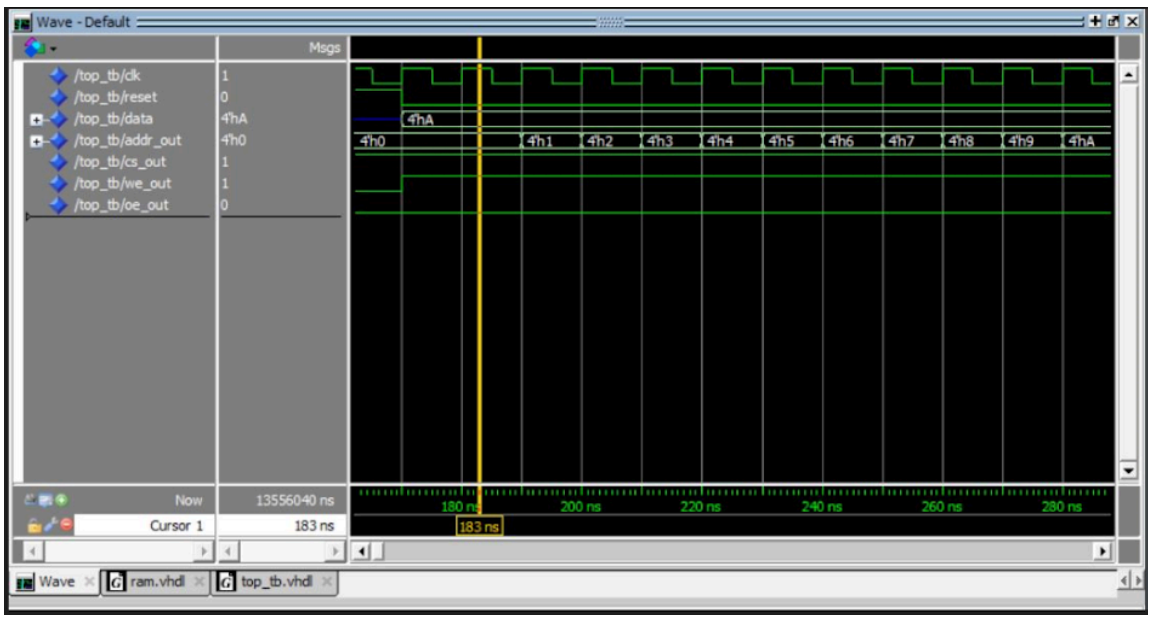
```
signal cs: std_logic;
signal we: std_logic;
signal oe: std_logic;

BEGIN
        cs_out <= cs;
    we_out <= we;
    oe_out <= oe;
    addr_out <= address;

g1: mem_fsm PORT MAP(
        clk => clk,
        reset => reset,
        address => address,
        data => data,
        cs => cs,
        we => we,
        oe => oe
        );

g2:  ram PORT MAP(
        address => address,
        data => data,
        cs => cs,
        we => we,
        oe => oe
        );

END;
```

top_tb.vhdl

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

ENTITY top_tb IS
END top_tb;

ARCHITECTURE test OF top_tb IS

component mem_top IS
    port(
    clk: IN  std_logic;
        reset: IN  std_logic;
    data: inout std_logic_vector ( 3 downto 0);
        addr_out: OUT  std_logic_vector(3 downto 0);
        cs_out: OUT  std_logic;
        we_out: OUT  std_logic;
        oe_out: OUT  std_logic

    );
END component;

signal clk: std_logic := '0';
signal reset: std_logic := '0';
signal data: std_logic_vector(3 downto 0);
signal addr_out: std_logic_vector(3 downto 0);
signal cs_out: std_logic;
signal we_out: std_logic;
signal oe_out: std_logic;

BEGIN
uut: mem_top port map (
    clk =>  clk,
        reset => reset,
    data => data,
        addr_out => addr_out,
        cs_out => cs_out,
        we_out => we_out,
        oe_out => oe_out
);
```

```
clock stimulus: process
Begin
CLK <= not CLK;
wait for 5 ns;
End process clock_stimulus;

Process
Begin
reset <= '1';
wait for 170 ns;
reset <= '0';
wait for 170 ns;
reset <= '1';
wait;
End process;

END test;
```

Simulation Waveform:



Result Discussion:

The 4 bit binary data ‘1010’ was stored successfully into the 16 address lines of SRAM. This part of the lab was easy because the mem_top, ram, and mem_fsm was given by the professor on the course webpage. The only thing we had to write was the testbench for the top file. This part was a good introduction to how SRAM functions and works.

Conclusion:

In this lab, I was able to learn how to implement various circuits with finite state machines using VHDL. We were also introduced to how to create a clock division so that we can see the results. I was able to gain experience utilizing the concepts of hierarchical design to structure my code in such a way that it is easy to understand the circuit piece by piece. We learned how to build a Linear Feedback Shift Register (LFSR) and how to generate a hamming code. The final part of this lab introduced the features of SRAM and how to write data into memory location. Overall, I was able to complete all of the sections in this lab.