Veronica Gonzalez

CPE 166: Advanced Logic Design

Lab Session: 02

Lab Day: Wednesday 2:00 PM

Lab 3

VHDL Structural Design and Usage of Logic Analyzer

Lab Instructor: Jing Pang

Table of Contents

Objectives:

- Learn structural design strategy using VHDL
- Learn how to use logic analyzer for timing mode measurement
- Learn how to use logic analyzer for state mode measurement
- Learn finite state machine design using VHDL
- Learn how to simulate VHDL programs
- Learn how to download VHDL design to Spartan3E starter board

Materials:

- Spartan3E Starter Board
- +5VDC Power Supply
- USB Cable
- Xilinx ISE Tool
- Logic Analyzer

Part 1: 50 KHz BCD Counter Design and Logic Analyzer Measurement

Description: The clock signal is used to control timing of the digital design events on rising edge or falling edge of the clock. The digital Spartan3E board has 50 MHz crystal oscillator at pin "C9". The clock division circuit can be designed to implement different clock frequencies.

Purpose: The purpose of this lab is to analyze the clock frequency on the logic analyzer. Design a 50 KHz clock, and use it as "clk" signal for BCD counter circuit. The BCD counter can output binary values from "0000" to "1001". The BCD counter requires seven inputs: clk, load, up_down, sw1, sw2, sw3, and sw4.

Features:

1. If load = '1', the BCD counter should be loaded with 4-bit binary values set by four switches: sw1, sw2, sw3, and sw4.
2. If load = '0', and up_down = '1', the BCD counter will count up.
3. If load = '0', and up_down = '0', the BCD counter will count down.

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

entity logic_analyzer is
    port ( clk     : in std_logic;
           clkref  : out std_logic;
           din     : in std_logic_vector (3 downto 0); -- 4-bit input data
           load    : in std_logic;
           updown  : in std_logic;
           dout    : out std_logic_vector(3 downto 0) );

end logic_analyzer;

architecture beh of logic_analyzer is
signal  cnt:  std_logic_vector(999 downto 0); -- divide by 1000
signal  cnt2:  std_logic_vector(3 downto 0); -- count BCD numbers
signal clk2: std_logic;
begin

     process(clk)
     begin
        if (rising_edge(clk)) then
          if (cnt = 999) then           -- N-1
             cnt <= (others=>'0');
             clk2 <= '1';
          elsif( cnt < 499) then    -- N/2-1
             cnt <= cnt + 1;
             clk2 <= '1';
          else
             cnt <= cnt + 1;
             clk2 <= '0';
          end if;
        end if;

     end process;


     process(clk2, load, updown)
     begin
        if(load='1') then
            cnt2 <= din;
        elsif (rising_edge(clk2)) then
           if(updown='1') then
              if (cnt2 = 9) then
                 cnt2 <= (others=>'0');
             else
                cnt2 <= cnt2 + 1;
             end if;
          else
             if (cnt2 = 0) then
                cnt2 <= "1001";
             else
                cnt2 <= cnt2 - 1;
             end if;
          end if;
       end if;

     end process;


    clkref <= clk2;
    dout <= cnt2;

end beh;
```

Figure 1-1: Clock Division VHDL Code.

```
1  NET  "din<0>"   LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
2  NET  "din<1>"   LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;
3  NET  "din<2>"   LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;
4  NET  "din<3>"   LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;
5
6  NET  "load"     LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
7  NET  "updown"   LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
8
9  NET  "clk"      LOC = "C9"  | IOSTANDARD = LVCMOS33;
10
11 NET  "dout<0>"  LOC = "B4" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
12 NET  "dout<1>"  LOC = "A4" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
13 NET  "dout<2>"  LOC = "D5" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
14 NET  "dout<3>"  LOC = "C5" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
15
16 NET  "clkref"  LOC = "A6" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
17
```

Figure 1-2: Clock Division User Constraint File.

Demo:

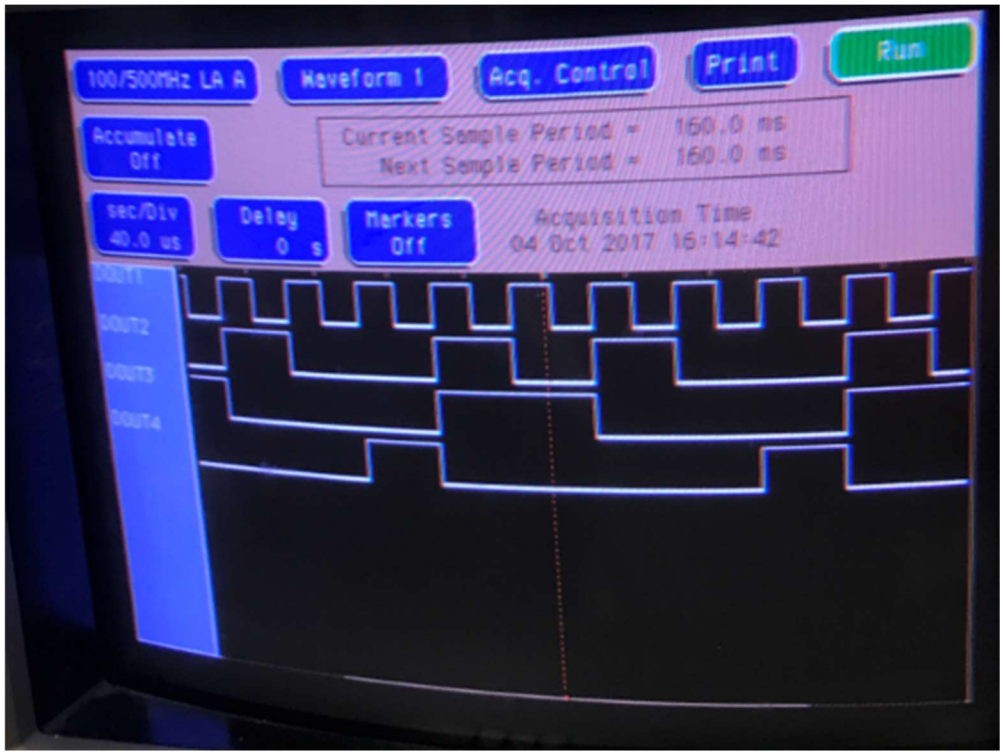

Figure1-3: Logic Analyzer Demo

Part 2: Pseudorandom Number Generator and Hamming Code Display on LEDs

Description: The (7, 4) Hamming Code is designed to detect and correct errors in 4 bit transmissions. For each 4-bit message transmission, 3 parity bits need to be attached to construct (7, 4) Hamming Code.

Procedure: The procedure for hamming code is as follows

1. The original 4-bit message is "1011" and it is noted as d4, d3, d2, d1.
2. Calculate parity bits p1, p2, p3 in three groups such as, d4 =1, d3 =0, d2 =1, and d1 = 1.
   P3 is the even parity bit of the group of data; d4 d3 d2, p3 => (101 p3) => p3 => 0
   P2 is the even parity bit of the group of data; d4 d3 d1, p2 => (101 p2) => p2 => 0
   P1 is the even parity bit of the group of data; d4 d2 d1, p1 => (111 p1) => p1 => 1
3. The final (7, 4) Hamming Code is arranged in the following format: d4 d3 d2 p3 d1 p2 p1.

   As a result, the final (7, 4) Hamming Code is: 1010101.

Design Specifications: Use $X^4 + X + 1$ linear feedback shift register (LFSR) as pseudorandom generator circuit shown in Figure2-1. The four D-Flip Flops used in LFSR circuit output 4-bit output values at 1 Hz in the free running mode. When sw1 switch is pressed, the finite state machine (FSM) will be in the idle state and the LCD display is blank. When the sw2 switch is pressed, the 4-bit LFSR message will be displayed on four LEDs. When SW3 is pressed, the (7, 4) hamming code will be displayed on seven LEDs.

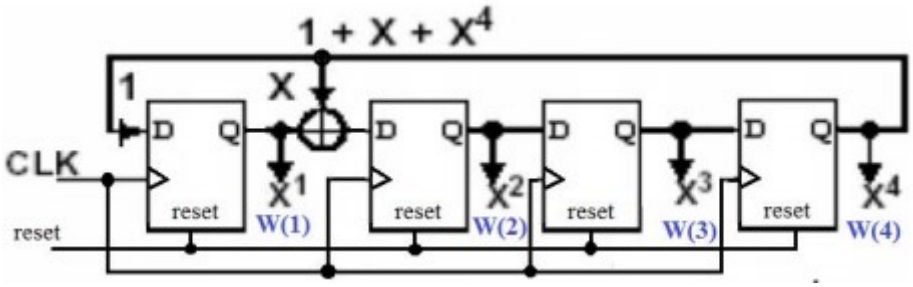

Figure 2-1: $X^4 + X + 1$ LFSR circuit

2-1: Clock Division

Description: Design clock divide by 50MHz down to 1 Hz.

```
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3  use ieee.std_logic_unsigned.all;
 4  use ieee.std_logic_arith.all;
 5
 6  entity clkdiv50 is
 7      port ( clk:  in std_logic;
 8             clk2: out std_logic );
 9
10  end clkdiv50;
11
12  architecture beh of clkdiv50 is
13  signal  cnt:  std_logic_vector(3 downto 0);
14  begin
15
16       process(clk)
17       begin
18          if (rising_edge(clk)) then
19            if (cnt = 49999999 ) then          -- N-1
20               cnt <= (others=>'0');
21               clk2 <= '1';
22            elsif( cnt < 24999999) then    -- N/2-1
23               cnt <= cnt + 1;
24               clk2 <= '1';
25            else
26               cnt <= cnt + 1;
27               clk2 <= '0';
28            end if;
29          end if;
30       end process;
31  end beh;
32
33
34
```

Figure 2-2: 1 Hz Clock Division VHDL Code.

2-2: LFSR

Description: LFSR is a linear feedback shift register that consists of registers and linear feedback from its previous register outputs.

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4     ENTITY lfsr IS
5     PORT
6     (
7        reset, clk: IN STD_LOGIC;
8        q : OUT STD_LOGIC_VECTOR(4 downto 1)
9     );
10 END lfsr;
11
12 ARCHITECTURE beh OF lfsr IS
13 signal W: std_logic_vector(4 downto 1);
14 BEGIN
15    process(clk, reset)
16    begin
17       if (reset='1') then
18          W <= ( 1=>'1', others => '0' );
19       elsif (rising_edge (clk)) then
20             W <= W(3 downto 1) & ( W(3) xor W(4));
21       end if;
22    end process;
23    q <= W;
24 END beh;
```

Figure2-3: LFSR VHDL Code.

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY lfsr_tb is
5  End lfsr_tb;
6
7  ARCHITECTURE beh OF lfsr_tb IS
8
9  Component lfsr
10    PORT
11    (
12       reset, clk: IN STD_LOGIC;
13       Q : OUT STD_LOGIC_VECTOR(4 downto 1)
14    );
15 END Component;
16
17 signal reset, clk: std_logic;
18 signal Q: std_logic_vector(4 downto 1);
19 BEGIN
20    uut: lfsr port map( reset=>reset, clk=>clk, Q=>Q);
21    Process
22    Begin
23       Clk <= '0';
24       Wait for 30 ns;
25       Clk <= '1';
26       Wait for 30 ns;
27    End process;
28    Process
29    Begin
30       reset <='1';
31       Wait for 18 ns;
32       reset <='0';
33       Wait for 20 ns;
34       Wait;
35    End process;
36 END beh;
```

Figure 2-4: LFSR VHDL Testbench Code.

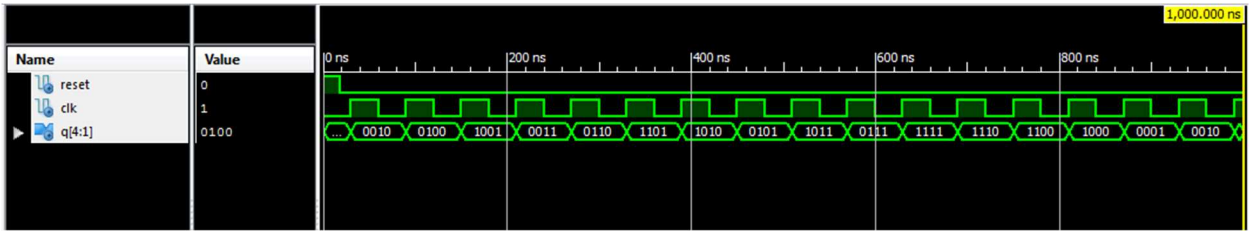

Figure 2-5: LFSR VHDL Testbench Waveform Simulation.

2-3: DFFS

Description: The purpose of the D-Flip Flops is to clear the data if clear is active high, or to load the data if load is active high. When load= '1' the data will be locked in the DFFs.

```
1  Library ieee;
2  Use ieee.std_logic_1164.all;
3
4  ENTITY dff IS
5   PORT
6   (
7           clk, reset, load: in std_logic;
8           d: in std_logic_vector(3 downto 0);
9           q: out std_logic_vector(3 downto 0)
10    );
11 END dff;
12
13 Architecture beh of dff is
14 begin
15     process(clk,load, reset )
16     begin
17        if (reset = '1') then
18          q <= "0000";
19        elsif (rising_edge (clk))then
20          if (load = '1') then
21          q <= d;
22          end if;
23        end if;
24     end process;
25 End beh;
```

Figure 2-6: DFF VHDL Code.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY dff_tb IS
END dff_tb;

ARCHITECTURE beh OF dff_tb IS
-- Component Declaration
        COMPONENT dff
        PORT(
                clk,reset,load : IN std_logic;
                d : IN std_logic_vector(3 downto 0);
                q : OUT std_logic_vector(3 downto 0)
                );
        END COMPONENT;

        SIGNAL clk,reset,load :  std_logic;
        SIGNAL d: std_logic_vector(3 downto 0);
        SIGNAL q :  std_logic_vector(3 downto 0);


BEGIN
-- Component Instantiation
 uut: dff PORT MAP(clk => clk,reset => reset, load=> load, d=>d,q=>q);

--  Test Bench Statements
   PROCESS
   BEGIN
       clk<='0';  Wait for 40 ns;
       clk<='1'; Wait for  40 ns;
   END PROCESS;

   PROCESS
   BEGIN

      d<="1011";
      reset<= '1'; load <= '0';
      Wait for 100 ns;
      reset<= '0'; load <='1';
      Wait for 80 ns;
      reset<= '0'; load<='0';
      wait;
   END PROCESS;

END beh;
```

Figure 2-7: DFF Testbench VHDL Code.

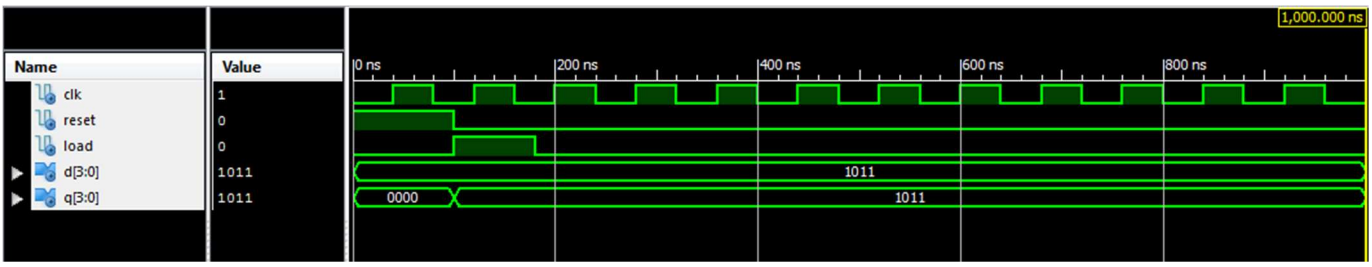

Figure 2-8: DFF Testbench Waveform Simulation.

2-4: Hamming

Description: The hamming design accepts 4-bit data inputs and then it will generate 7-bit hamming data. The figures below helped visualize hamming code.

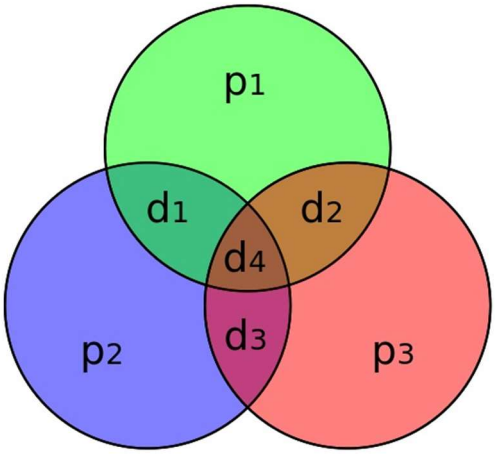

Figure 2-9: Data Input Position of d1, d2, d3, and d4.

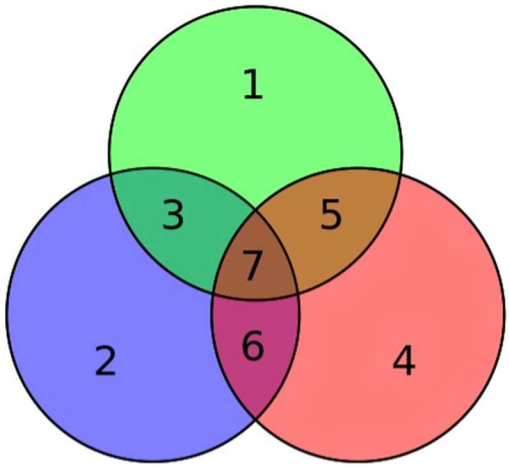

Figure 2-10: Bit Position of the Data

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4     ENTITY hamming IS
5     PORT
6     (
7        d: IN STD_LOGIC_VECTOR(3 downto 0);
8        dout : OUT STD_LOGIC_VECTOR(6 downto 0)
9     );
10 END hamming;
11
12
13  ARCHITECTURE beh OF hamming IS
14 signal p: std_logic_vector(2 downto 0);
15 BEGIN
16
17    p(0) <= d(3) xor d(1) xor d(0);  --p1 signal
18    p(1) <= d(3) xor d(2) xor d(0);  --p2 signal
19    p(2) <= d(3) xor d(2) xor d(1);  --p3 signal
20
21    dout <= d(3) & d(2) & d(1) & p(2) & d(0) & p(1) & p(0);
22
23 END beh;
```

Figure 2-11: Hamming VHDL Code.

```
1
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.ALL;
4
5   ENTITY hamming_tb IS
6   END hamming_tb;
7
8   ARCHITECTURE beh OF hamming_tb IS
9
10      -- Component Declaration for the Unit Under Test (UUT)
11
12      COMPONENT hamming
13      PORT(
14           d : IN  std_logic_vector(3 downto 0);
15           dout : OUT  std_logic_vector(6 downto 0)
16          );
17      END COMPONENT;
18
19     --Inputs
20     signal d : std_logic_vector(3 downto 0) := (others => '0');
21     --Outputs
22     signal dout : std_logic_vector(6 downto 0);
23
24  BEGIN
25
26     -- Instantiate the Unit Under Test (UUT)
27     uut: hamming PORT MAP (d => d,dout => dout);
28
29     process
30     begin
31        d<="1011";
32        wait for 400 ns;
33        d<="1101";
34        wait;
35     end process;
36
37  END beh;
```

Figure 2-12: Hamming Testbench VHDL Code.

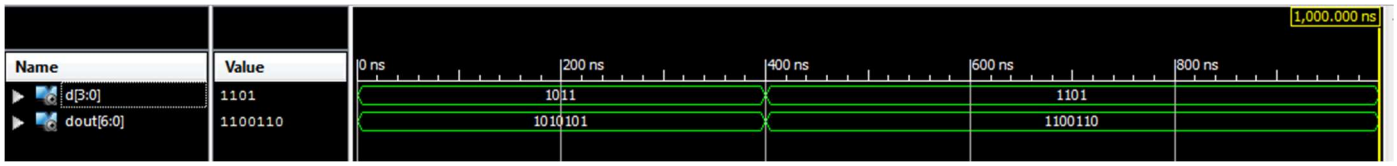

Figure 2-13: Hamming Testbench Waveform Simulation.

2-5: Mux

Description: This is a 4 to 1 mux but two of the selections are not used. There are two selections, one selection for to output hamming code and one selection to output LFSR message.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity mux is
   port (
      din: in std_logic_vector(3 downto 0);
      ham: in std_logic_vector (6 downto 0);
      sw2,sw3 : in std_logic;
      led : out std_logic_vector(6 downto 0)
   );
end mux;

architecture arch of mux is
signal temp: std_logic_vector(6 downto 0);
   begin

      temp <= "000" & din ;
      process (temp,ham,sw2,sw3)
      begin
      if (sw2 = '1') then
         led <= temp;
         elsif (sw3 = '1') then
         led <= ham;
         else
         led <= "0000000";
      end if;
   end process;
end arch;
```

Figure 2-14: Mux VHDL Code.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY mux_tb IS
END mux_tb;

ARCHITECTURE beh OF mux_tb IS

-- Component Declaration
        COMPONENT mux
        PORT(
               din: in std_logic_vector(3 downto 0);
               ham: in std_logic_vector (6 downto 0);
               sw2,sw3 : in std_logic;
               led : out std_logic_vector(6 downto 0)
        );
        END COMPONENT;
       --Inputs
        SIGNAL ham: std_logic_vector(6 downto 0);
        SIGNAL din    : std_logic_vector(3 downto 0);
        SIGNAL sw2,sw3   : std_logic;
        --Outputs
        SIGNAL led :  std_logic_vector(6 downto 0);


BEGIN

-- Component Instantiation
        uut: mux PORT MAP(din=>din,ham=>ham,sw2 => sw2,sw3 => sw3,led=>led);

--  Test Bench Statements
   PROCESS
   BEGIN
       ham<="1010101";din<="1011";sw2<= '0'; sw3<= '0';
       wait for 100 ns;
       sw2 <= '1'; sw3<= '0';
       wait for 100 ns;
       sw2 <= '0'; sw3<= '1';
       wait for 100 ns;
       sw2 <= '0'; sw3<= '0';
      wait; -- will wait forever
   END PROCESS;

END beh;
```

Figure 2-15: Mux Testbench VHDL Code.

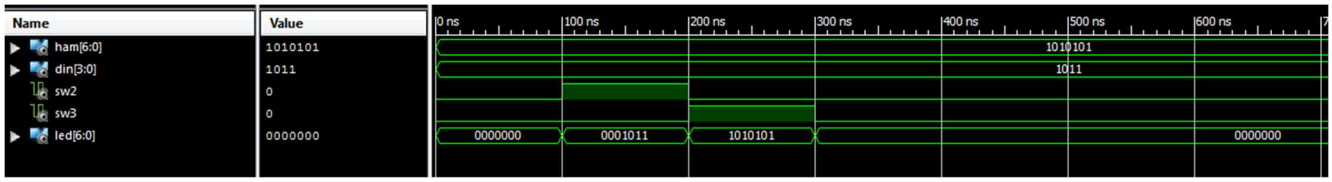

Figure 2-16: Mux Testbench Waveform Simulation.

2-6: FSM

Description: The purpose of the FSM is to control the signals to output the data of the top module selection.

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  ENTITY fsm IS
4     PORT
5     (   --sw1 is reset
6        clk,sw1,sw2,sw3,load : IN STD_LOGIC;
7        clear,sl,slfsr,sham: OUT STD_LOGIC);
8  END fsm;
9  ARCHITECTURE beh OF fsm IS
10 type state_type is (s0,s1,s2,s3);
11 signal cs, ns: state_type;
12
13 BEGIN
14    Process(sw1,clk) -- sw1 is reset
15    Begin
16       If (sw1 = '1') then
17          cs <= s0;
18       elsif (rising_edge(clk)) then
19          cs <= ns;
20       end if;
21    End process;
22
23    Process (cs,load,sw2,sw3) begin
24       Case (cs) is
25       When s0 =>  -- reset signals
26                   If(load = '1') then
27                      ns <= s1;
28                   elsif(sw2 = '1') then
29                      ns <= s2;
30                   elsif(sw3 = '1') then
31                      ns<= s3;
32                   else ns <= s0;
33                   end if;
34                   clear <= '1';sl <='0'; slfsr <= '0'; sham <= '0';
35       When s1 => --clr and load for dff
36                If(load = '1') then
37                      ns <= s1;
38                elsif(sw2 = '1') then
39                   ns <= s2;
40                elsif(sw3 = '1') then
41                   ns<= s3;
42                else ns <= s0;
43                end if;
44                clear <= '0'; sl <='1'; slfsr <= '0'; sham <= '0';
```

Figure 2-17: FSM VHDL Code.

Note: Continued on next page.

```
45      When s2 => -- signals to display lfsr
46
47              If(load = '1') then
48                 ns <= s1;
49              elsif(sw2 = '1') then
50                 ns <= s2;
51              elsif(sw3 = '1') then
52                 ns<= s3;
53              else ns <= s0;
54              end if;
55               clear <= '0'; sl<= '0'; slfsr <= '1'; sham <= '0';
56      When s3 => -- signals to display hamming
57
58              If(load = '1') then
59                 ns <= s1;
60              elsif(sw2 = '1') then
61                 ns <= s2;
62              elsif(sw3 = '1') then
63                 ns<= s3;
64              else ns <= s0;
65              end if;
66              clear <= '0'; sl<= '0'; slfsr <= '0'; sham <= '1';
67      When others=>
68              ns <= S0;
69      end case;
70    end process;
71  END beh;
```

Figure 2-17 (cont.): FSM VHDL Code.

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3
4   -- Uncomment the following library declaration if using
5   -- arithmetic functions with Signed or Unsigned values
6   --USE ieee.numeric_std.ALL;
7
8   ENTITY fsm_tb IS
9   END fsm_tb;
10
11  ARCHITECTURE behavior OF fsm_tb IS
12      COMPONENT fsm
13      PORT(
14           reset,clk : IN  std_logic;
15           clear,load: OUT STD_LOGIC;
16           sel: OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
17      END COMPONENT;
18
19     --Inputs
20     signal reset,clk : std_logic;
21     --Outputs
22     signal clear, load : std_logic;
23     signal sel: std_logic_vector(1 DOWNTO 0);
24
25  BEGIN
26     -- Instantiate the Unit Under Test (UUT)
27     uut: fsm PORT MAP (
28            reset => reset,clk => clk,clear => clear, load=>load, sel=>sel);
29
30     reset <= '1', '0' after 25 ns, '0' after 500 ns;
31     PROCESS
32       BEGIN
33           clk<='0';  Wait for 40 ns;
34           clk<='1'; Wait for  40 ns;
35       END PROCESS;
36
37  END;
```

Figure 2-18: FSM Testbench VHDL Code.

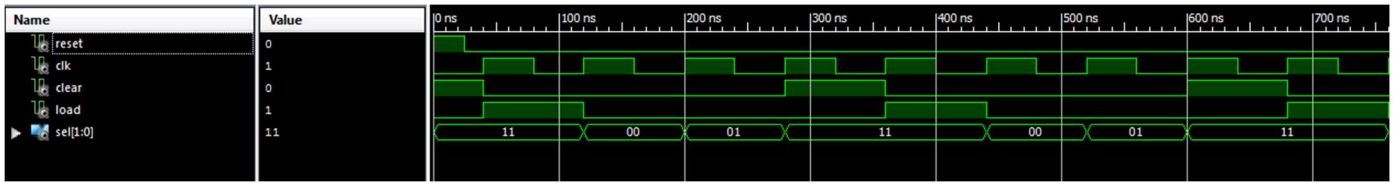

Figure 2-19: Mux Testbench Waveform Simulation.

2-7: Top Design

Description: The top module is the main module where each instance of each components are created and connected together to create one piece of virtual hardware.

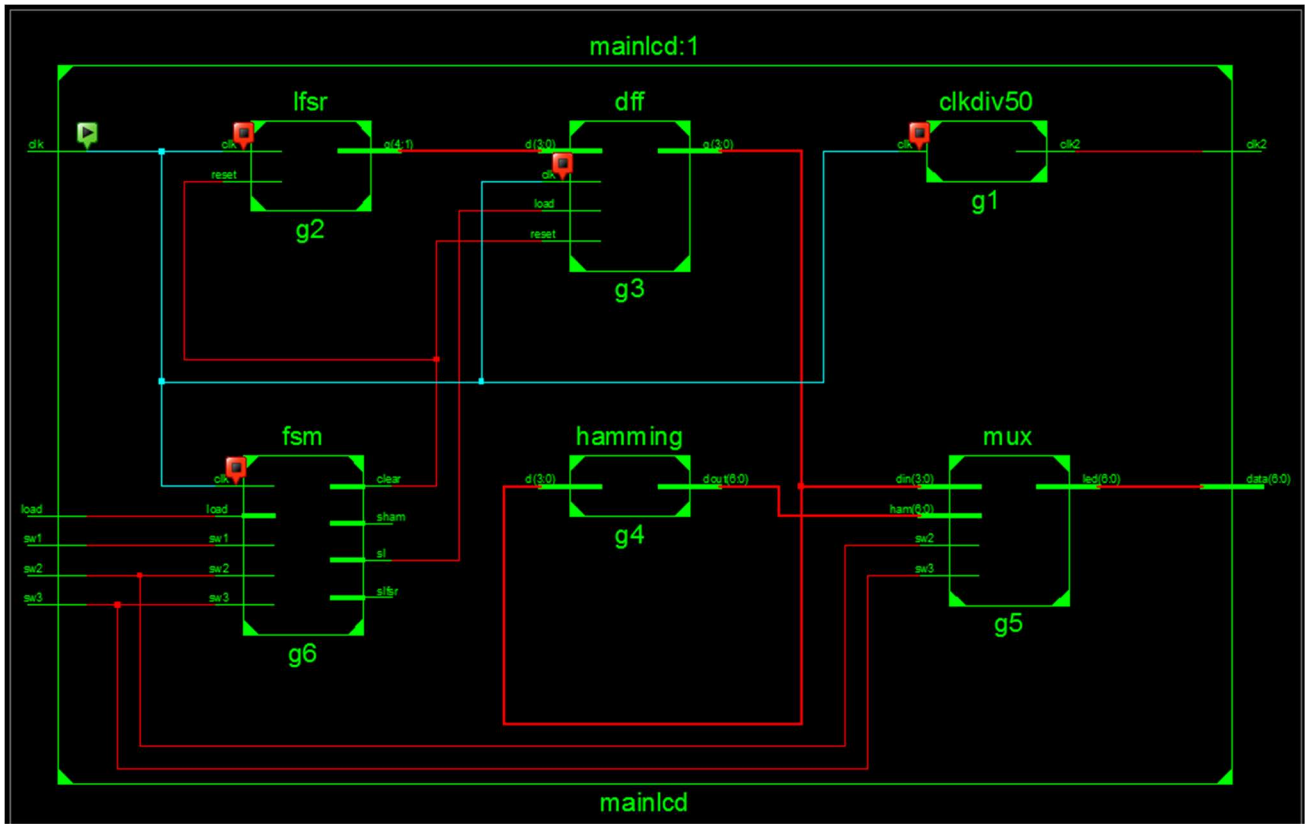

Figure 2-20: Top Design Schematic.

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

ENTITY mainlcd IS
PORT( --sw1 to button for reset,
       clk,load, sw1,sw2,sw3: IN  std_logic;
       --data_in: in std_logic(4 downto 1);
       data  : out std_logic_vector ( 6 downto 0);
       clk2: out std_logic
   );
END mainlcd;

ARCHITECTURE behavior OF mainlcd IS

COMPONENT clkdiv50
PORT ( clk:  in std_logic;
      clk2: out std_logic
      );
END COMPONENT;

COMPONENT lfsr
PORT(
      reset,clk: IN STD_LOGIC;
      --d: in std_logic_vector(4 downto 1);
      q : OUT STD_LOGIC_VECTOR(4 downto 1)
   );
END COMPONENT;

COMPONENT dff
PORT(
         clk, reset, load: in std_logic;
         d: in std_logic_vector(3 downto 0);
         q: out std_logic_vector(3 downto 0)
   );
END COMPONENT;

COMPONENT hamming
PORT(
      d: IN STD_LOGIC_VECTOR(3 downto 0);
      dout : OUT STD_LOGIC_VECTOR(6 downto 0)
   );
END COMPONENT;

COMPONENT mux
PORT(
      din: in std_logic_vector(3 downto 0);
      ham: in std_logic_vector (6 downto 0);
      sw2,sw3 : in std_logic;
      led : out std_logic_vector(6 downto 0)
      );
END COMPONENT;

COMPONENT fsm
PORT(
      clk,load,sw1,sw2,sw3: IN STD_LOGIC;
      clear,sl, slfsr,sham: OUT STD_LOGIC
   );
END COMPONENT;

signal dlfsr,dout : std_logic_vector(4 downto 1);
signal d, ledout : std_logic_vector(6 downto 0);
signal ld_data,clr,sl,sig1,sig2: std_logic;


```

Figure 2-21: Top Module Design in VHDL Code.

Note: Continued on next page.

```
66  BEGIN
67     data <= ledout;
68     clr <= sw1;
69     ld_data <=load;
70     sig1 <= sw2;
71     sig2 <= sw3;
72
73     g1: clkdiv50 PORT MAP (
74           clk => clk, clk2 => clk2
75        );
76
77     g2:  lfsr port map(
78          clk=> clk,reset=>clr, Q=>dlfsr
79        );
80
81      g3: dff port map(
82           clk => clk, reset => clr, load => sl , d=>dlfsr , q => dout
83        );
84
85     g4: hamming port map(
86           d => dout ,dout => d
87        );
88
89     g5: mux port map(
90           din => dout, ham => d , sw2 => sig1 , sw3 => sig2 , led => ledout
91        );
92
93     g6: fsm port map(
94           clk => clk,load => load, sw1 => sw1, sw2 => sw2, sw3 => sw3, sl => sl, clear => clr, slfsr=> sig1, sham => sig2
95        );
96  END;
```

Figure 2-21 (cont.): Top Module Design in VHDL Code.

```
 1
 2  LIBRARY ieee;
 3  USE ieee.std_logic_1164.ALL;
 4
 5  -- Uncomment the following library declaration if using
 6  -- arithmetic functions with Signed or Unsigned values
 7  --USE ieee.numeric_std.ALL;
 8
 9  ENTITY mainlcd_tb IS
10  END mainlcd_tb;
11
12  ARCHITECTURE behavior OF mainlcd_tb IS
13
14      -- Component Declaration for the Unit Under Test (UUT)
15
16      COMPONENT mainlcd
17      PORT(clk : IN  std_logic;
18           load: IN  std_logic;
19           sw1 : IN  std_logic;
20           sw2 : IN  std_logic;
21           sw3 : IN  std_logic;
22           --data_in: in std_logic_vector(4 downto 1);
23           data : OUT  std_logic_vector(6 downto 0);
24           clk2 : OUT  std_logic
25          );
26      END COMPONENT;
27
28     --Inputs
29     signal clk : std_logic := '0';
30     signal load: std_logic := '0';
31     signal sw1 : std_logic := '0';
32     signal sw2 : std_logic := '0';
33     signal sw3 : std_logic := '0';
34     --signal data_in : std_logic_vector(4 downto 1);
35
36     --BiDirs
37     signal data : std_logic_vector(6 downto 0);
38
39     --Outputs
40     signal clk2 : std_logic;
```

Figure 2-22: Top Module Testbench VHDL Code.

Note: Continued on next page.

```
41
42  BEGIN
43
44     -- Instantiate the Unit Under Test (UUT)
45     uut: mainlcd PORT MAP (
46            clk => clk,
47            load => load,
48            sw1 => sw1,
49            sw2 => sw2,
50            sw3 => sw3,
51            --data_in => data_in,
52            data => data,
53            clk2 => clk2
54          );
55
56     PROCESS BEGIN
57           clk<='0';  Wait for 20 ns;
58           clk<='1'; Wait for  20 ns;
59     END PROCESS;
60
61     process
62     begin
63
64        -- hold reset state for 100 ns.
65        sw1<= '1'; load <= '0';sw2<= '0'; sw3<= '0';
66        wait for 70 ns;
67        sw1<= '0'; load <= '1';sw2<= '0'; sw3<= '0';
68        wait for 70 ns;
69        sw1<= '0'; load <= '0';sw2<= '1'; sw3<= '0';
70        wait for 100 ns;
71        sw1<= '0'; load <= '0';sw2<= '0'; sw3<= '1';
72
73        wait;
74     end process;
75
76  END;
```

Figure 2-22 (cont.): Top Module Testbench VHDL Code.

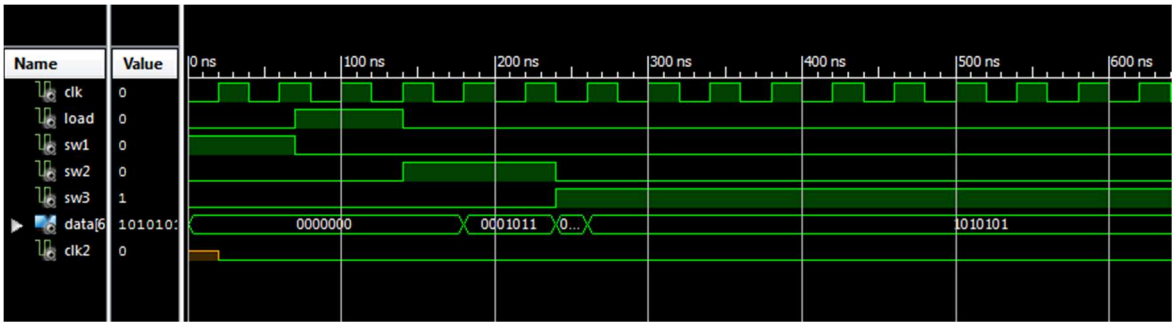

Figure 2-23: Top Module Testbench Waveform Simulation.

```
1   NET "sw1" LOC = "N17" | IOSTANDARD = LVTTL | PULLDOWN;
2   NET "load" LOC = "H18" | IOSTANDARD = LVTTL | PULLDOWN;
3   NET "sw2" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP;
4   NET "sw3" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;
5
6   NET "clk2" LOC = "D16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
7   NET "clk" LOC = "C9"     | IOSTANDARD = LVCMOS33;
8
9
10  NET "data<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
11  NET "data<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
12  NET "data<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
13  NET "data<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
14  NET "data<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
15  NET "data<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
16  NET "data<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
```

Figure 2-24: Top Module User Constraint File.

Part 3: Calculator LCD Display

Description: The calculator display is an implementation design based off of the design in part two. The purpose is to display data of the LFSR or output Calculator Functions depending on the input signals.

Procedure: The procedure for the Calculator LCD Display is as follows,

1. Reset the design using Button 1 (b1).
2. When Button 2 (b2) is logic high, the LFSR is enabled to run, and the values of q4, q3, q2, and q1 of the LFSR data will be displayed on four LEDs
3. When Button 1 (b2) is logic low, the LFSR is disabled to run, and the values of q4, q3, q2, and q1 wont change. Connect q4, q3, q2, and q1 with two different 2-bit vectors dout3 and dout4.
   Dout3 <= q2 & q1; dout4 <= q4 & q3;
4. Perform the calculator function according to the values of buttons b3 and b4 as show in Figure 3-1. Demo the calculator F on the LCD.

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

Figure 3-1: Calculation Function

Clock Division VHDL

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith.all;
5
6  entity clkdiv is
7  port(
8     osc: in std_logic;
9     b1 : in std_logic;
10    clk: out std_logic
11 );
12 end clkdiv;
13
14 architecture beh of clkdiv is
15 signal clkref: std_logic;
16 signal cnt : std_logic_vector(26 downto 0);
17 begin
18    clk <= clkref;
19
20    process(osc,b1)
21    begin
22       if(b1 = '1') then
23          cnt<=(others=>'0');
24          clkref<='0';
25       elsif( rising_edge(osc) ) then
26          if (cnt = 49999999 ) then
27             cnt <= ( others => '0' );
28             clkref <= '1';
29          elsif (cnt < 24999999) then
30             cnt <= cnt + 1;
31             clkref <= '1';
32          else
33             cnt <= cnt + 1;
34             clkref <= '0';
35          end if;
36       end if;
37    end process;
38
39 end beh;
```

Figure 3-2: Calculator Clock Division in VHDL

Finite State Machine VHDL

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith.all;
5
6  entity fsm is
7  port(
8     b1,b2,clk          : in std_logic;
9     clr,sig1,sig2 : out std_logic);
10 end fsm;
11
12 architecture Behavioral of fsm is
13 type state is (s1,s2,s3);
14 signal cs,ns : state;
15 begin
16    process(clk,b1)
17    begin
18       if(b1 ='1') then
19          cs <= s1;
20       elsif(rising_edge(clk)) then
21          cs <= ns;
22       end if;
23    end process;
24
25    process(cs,b1,b2)
26    begin
27       case (cs) is
28          when s1 => clr <= '1'; sig1 <= '0'; sig2 <= '0';
29             if (b1 = '1') then
30                ns <= s1;
31             elsif (b2 = '1') then
32                ns<=s2;
33             elsif(b2 = '0') then
34                ns <= s3;
35             end if;
36          when s2 => clr <= '0'; sig1 <= '1'; sig2 <= '0';
37             if (b1 = '1') then
38                ns <= s1;
39             elsif (b2 = '1') then
40                ns<=s2;
41             elsif(b2 = '0') then
42                ns <= s3;
43             end if;
44          when s3 => clr <= '0'; sig1 <= '0'; sig2 <= '1';
45             if (b1 = '1') then
46                ns <= s1;
47             elsif (b2 = '1') then
48                ns<=s2;
49             elsif(b2 = '0') then
50                ns <= s3;
51             end if;
52       end case;
53    end process;
54 end Behavioral;
```

Figure 3-3: Calculator FSM VHDL Code.

Linear Feedback Shift Register

```
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3  use ieee.std_logic_unsigned.all;
 4  use ieee.std_logic_unsigned.all;
 5  use ieee.std_logic_arith.all;
 6
 7  entity lfsr is
 8  port(
 9     b2,clk,reset,sig1,sig2: in std_logic;
10     q  : out std_logic_vector(4 downto 1)
11  );
12  end lfsr;
13
14  architecture beh of lfsr is
15  signal w: std_logic_vector(4 downto 1);
16  begin
17     q<=w;
18
19     process(clk,reset,sig1,sig2,b2)
20     begin
21        if(reset='1') then
22           w<="1111";
23        elsif(sig1 = '1') then
24           if(rising_edge(clk)) then
25              w(4)<=w(3);
26              w(3)<=w(2);
27              w(2)<=w(4) xor w(1);
28              w(1)<=w(4);
29           end if;
30        elsif(sig2 = '1') then
31              w(4)<=w(4);
32              w(3)<=w(3);
33              w(2)<=w(2);
34              w(1)<=w(1);
35        end if;
36     end process;
37
38  end beh;
```

Figure 3-4: Calculator LFSR VHD Code.

Data

Description: Data is the function of generating dout3 and dout4 for the calculator function.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith.all;
5
6  entity data is
7  port(
8     q : in std_logic_vector ( 4 downto 1);
9     dout3, dout4: out std_logic_vector(1 downto 0));
10 end data;
11
12 architecture Beh of data is
13 begin
14    process(q)
15    begin
16       dout3 <= q(2)&q(1);
17       dout4 <= q(4)&q(3);
18    end process;
19
20 end Beh;
```

Figure 3-5: Calculator Data VHD Code

Calculator 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
6  entity calculator is
7  port(
8     dout3, dout4 : in std_logic_vector(1 downto 0);
9     b1,b2,b3, b4 : in std_logic;
10    f            : out std_logic_vector(3 downto 1));
11 end calculator;
12
13 architecture Behavioral of calculator is
14 begin
15    process(b1,b2,b3,b4,dout3,dout4)
16    begin
17       if(b1='1') then
18          f<="000";
19       elsif(b3='0' and b4='0') then
20          if(b1/='1' and b2/='1') then
21             if(dout4 > dout3 ) then
22                f <= "001";
23             else
24                f <= "000";
25             end if;
26          end if;
27       elsif(b3='0' and b4='1') then
28          if(b1/='1' and b2/='1') then
29             f <= ("0"& dout4) + ("0"& dout3) ;
30          end if;
31       elsif(b3='1' and b4='0') then
32          if(b1/='1' and b2/='1') then
33             f <= ("0"& dout4) or ("0"& dout3);
34          end if;
35       elsif(b3='1' and b4='1') then
36          if(b1/='1' and b2/='1') then
37             f <= ("0"& dout4) and ("0"& dout3) ;
38          end if;
39       end if;
40    end process;
41
42 end Behavioral;
```

Figure 3-6: Calculator VHDL Code

LCD Display VHDL

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

entity lcd_display is
port(
   clk: in std_logic;
   f : in std_logic_vector(3 downto 1);
   lcd_e, lcd_rs, lcd_rw, lcd_7, lcd_6, lcd_5, lcd_4, sf_ce0: out std_logic);
end lcd_display;

architecture beh of lcd_display is
signal lcd_busy, lcd_stb: std_logic;
signal lcd_code         : std_logic_vector (5 downto 0);
signal lcd_stuff        : std_logic_vector (6 downto 0);
signal count            : std_logic_vector (24 downto 0);
signal lcd_rw_1         : std_logic;
begin
   process(clk)
   begin
      if (rising_edge (clk)) then
         lcd_busy <= '1';
         sf_ce0 <= '1';
         count <= count + 1;
         case count (24 downto 20) is
            when "00000" =>
               lcd_code <= "000011";
            when "00001" =>
               lcd_code <= "000011";
            when "00010" =>
               lcd_code <= "000011";
            when "00011" =>
               lcd_code <= "000010";
            when "00100" =>
               lcd_code <= "000010";
            when "00101" =>
               lcd_code <= "001000";
            when "00110" =>
               lcd_code <= "000000";
            when "00111" =>
               lcd_code <= "000110";
            when "01000" =>
               lcd_code <= "000000";
            when "01001" =>
               lcd_code <= "001100";
            when "01010" =>
               lcd_code <= "000000";
            when "01011" =>
               lcd_code <= "000001";
            when "01100" =>
               lcd_code <= "100011";
            when "01101" =>
               lcd_code <= "10000"&f(3);
            when "01110" =>
               lcd_code <= "100011";
            when "01111" =>
               lcd_code <= "10000"&f(2);
            when "10000" =>
               lcd_code <= "100011";
            when "10001" =>
               lcd_code <= "10000"&f(1);
            when others =>
               lcd_code <= "010000";
         end case;
         lcd_rw <= lcd_rw_1;
         lcd_stb <= (count(19) xor count(18)) and (not lcd_rw_1) and lcd_busy;
         lcd_stuff <= (lcd_stb) & lcd_code;
         (lcd_e, lcd_rs, lcd_rw_1, lcd_7, lcd_6, lcd_5, lcd_4) <= lcd_stuff;
      end if;
   end process;

end beh;
```

Figure 3-7: Calculator LCD Display VHDL Code

Top Design

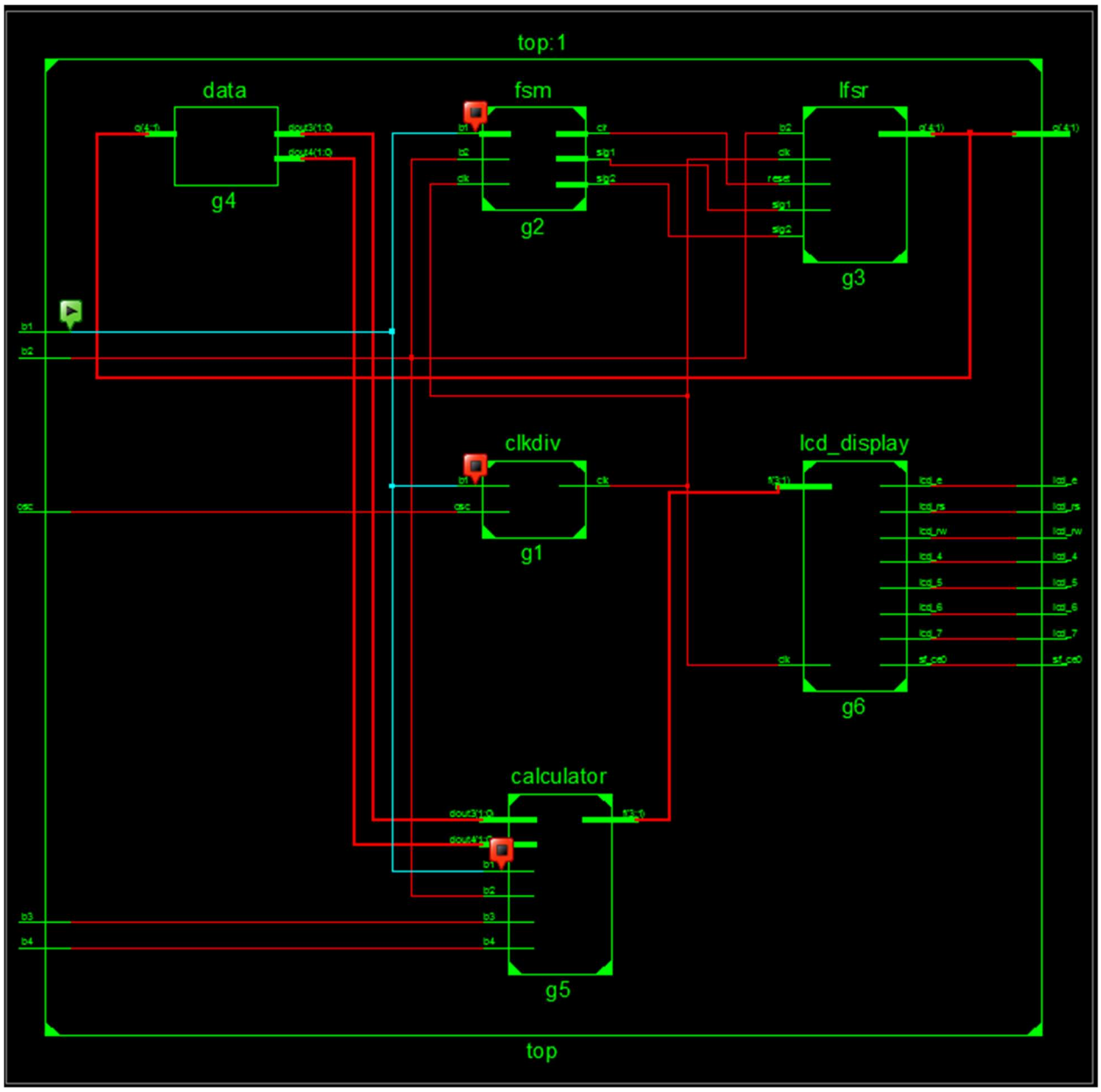

Figure 3-8: Calculator Top Design Schematic

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity top is
port (
   b1,b2,b3,b4,osc   : in std_logic;
   q  : inout std_logic_vector(4 downto 1);
   lcd_e, lcd_rs, lcd_rw, lcd_7, lcd_6, lcd_5, lcd_4, sf_ce0: out STD_LOGIC
);
end top;

architecture Behavioral of top is

component clkdiv is
port(
   osc: in std_logic;
   b1 : in std_logic;
   clk: out std_logic );
end component;

component fsm is
port(
   b1,b2,clk : in std_logic;
   clr,sig1,sig2  : out std_logic);
end component;

component lfsr is
port(
   b2,clk,reset,sig1,sig2  : in std_logic;
   q : out std_logic_vector(4 downto 1));
end component;

component data is
port(
   q : in std_logic_vector (4 downto 1);
   dout3,dout4 : out std_logic_vector (1 downto 0));
end component;

component calculator is
port(
   dout3, dout4   : in std_logic_vector(1 downto 0);
   b1,b2,b3, b4   : in std_logic;
   f              : out std_logic_vector(3 downto 1));
end component calculator;
```

Figure 3-9: Calculator Top Design VHDL Code.

Note: Continued on next page.

```
47
48  component lcd_display
49  port(
50     clk: in std_logic;
51     f  : in std_logic_vector(3 downto 1);
52     lcd_e, lcd_rs, lcd_rw, lcd_7, lcd_6, lcd_5, lcd_4, sf_ce0: out std_logic);
53  end component;
54
55  signal dout3,dout4   : std_logic_vector(1 downto 0);
56  signal clk,clr,sig1,sig2: std_logic;
57  signal f : std_logic_vector (3 downto 1);
58
59  begin
60     g1: clkdiv port map
61     (
62        osc =>osc ,b1 => b1,clk => clk
63     );
64     g2: fsm port map
65     (
66        b1 => b1,b2 => b2,clk => clk,clr => clr,sig1 => sig1 ,sig2 => sig2
67     );
68     g3: lfsr port map
69     (
70        b2 => b2,clk =>clk ,reset => clr,sig1 => sig1 ,sig2 => sig2, q =>q
71     );
72     g4: data port map
73     (
74        q => q,dout3 => dout3,dout4 => dout4
75     );
76     g5: calculator port map
77     (
78        dout3 => dout3, dout4 => dout4, b1 => b1,b2 => b2 ,b3 => b3, b4 => b4,f => f
79     );
80     g6: lcd_display port map
81     (
82        clk => clk,f =>f,lcd_e => lcd_e, lcd_rs => lcd_rs, lcd_rw => lcd_rw, lcd_7 => lcd_7 ,
83        lcd_6 => lcd_6, lcd_5 => lcd_5, lcd_4 => lcd_4, sf_ce0 => sf_ce0
84     );
85  end Behavioral;
```

Figure 3-9 (cont.): Calculator Top Design VHDL Code.

```
NET "q<1>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
NET "q<2>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
NET "q<3>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
NET "q<4>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;

NET "b1" LOC = "N17" | IOSTANDARD = LVTTL | PULLDOWN;
NET "b2" LOC = "H18" | IOSTANDARD = LVTTL | PULLDOWN;
NET "b3" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP;
NET "b4" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;

NET "sf_ce0" LOC = "D16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "osc" LOC = "C9"     | IOSTANDARD = LVCMOS33;

NET "lcd_e" LOC = "M18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "lcd_rs" LOC = "L18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "lcd_rw" LOC = "L17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;

NET "lcd_4" LOC = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "lcd_5" LOC = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "lcd_6" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "lcd_7" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
```

Figure 3-10: Calculator User Constraint File.

Part 4: Traffic Light Controller

Description: Suppose there is one road running east to west. There are East – West light controls with a red light (R), yellow light (Y) and green light (G). The behavior of such East – West light controls is shown in Figure 4-1. The "cnt" is used as counter. The counter is cleared to zeros when "reset" signal is logic high. The "reset" signal is used as asynchronous control signal for both the traffic light controller and the counter circuit.

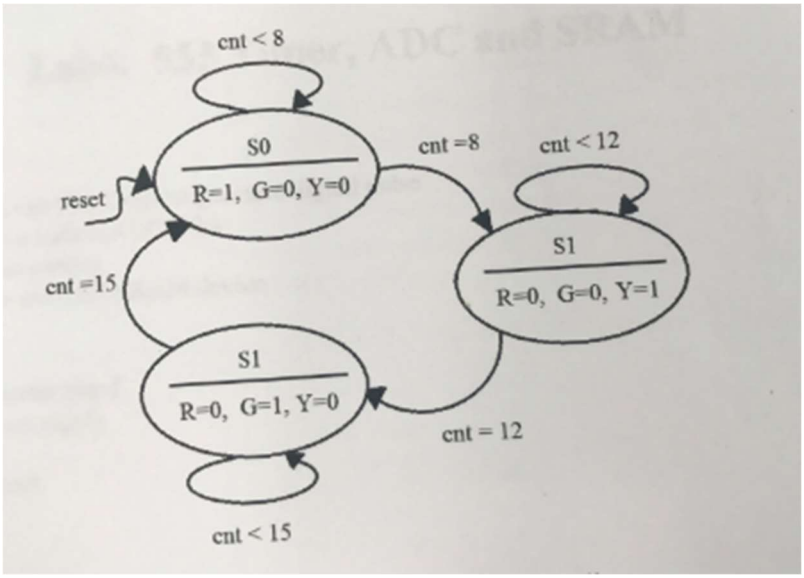

Figure 4-1: East – West traffic light controller

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

entity traffic_controller is

port ( clk, reset : IN std_logic;
       R,G,Y : OUT std_logic );

end traffic_controller;

architecture beh of traffic_controller is

type state_type is (s1,s2,s3);

signal state: state_type ;

signal count : STD_LOGIC_VECTOR(3 downto 0);
begin


process (clk,reset)
begin
if (reset ='1') then
   count <= "0000";
   state <=s1;
elsif (clk='1' and clk'event) then
  case state is
  when s1 =>
        count <= count + 1;
        if (count < 8) then
          state <= s1;
        elsif (count = 8) then
          state <= s2;
        end if;

  when s2 =>
        count <= count + 1;
        if (count < 12) then
          state <= s2;
        elsif (count = 12) then
          state <= s3;
        end if;
```

Figure 4-2: Traffic Light Controller FSM in VHDL.

Note: Continued on next page.

```
45    when s3 =>
46          count <= count + 1;
47          if (count < 15) then
48            state <= s3;
49          elsif (count = 15) then
50            state <= s1;
51          end if;
52    when others => state <= s1;
53    end case;
54
55  end if;
56
57  end process;
58
59  process(state)
60
61  begin
62
63  case state is
64
65    when s1 =>
66         R <='1';
67         G <='0';
68         Y <='0';
69    when s2 =>
70         R <='0';
71         G <='0';
72         Y <='1';
73    when s3 =>
74         R <='0';
75         G <='1';
76         Y <='0';
77    when others=>
78         R <='0';
79         G <='0';
80         Y <='0';
81
82  end case;
83
84  end process;
85
86  end beh;
87
```

Figure 4-2 (cont.): Traffic Light Controller FSM in VHDL.

```
1
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4
5  -- Uncomment the following library declaration if using
6  -- arithmetic functions with Signed or Unsigned values
7  --USE ieee.numeric_std.ALL;
8
9  ENTITY traffic_controller_tb IS
10 END traffic_controller_tb;
11
12 ARCHITECTURE behavior OF traffic_controller_tb IS
13
14     -- Component Declaration for the Unit Under Test (UUT)
15
16     COMPONENT traffic_controller
17     PORT(
18          clk : IN  std_logic;
19          reset : IN  std_logic;
20          R : OUT  std_logic;
21          G : OUT  std_logic;
22          Y : OUT  std_logic
23         );
24     END COMPONENT;
25
26
27    --Inputs
28    signal clk : std_logic := '0';
29    signal reset : std_logic := '0';
30
31    --Outputs
32    signal R : std_logic;
33    signal G : std_logic;
34    signal Y : std_logic;
35
36    -- Clock period definitions
37    constant clk_period : time := 10 ns;
```

Figure 4-3: Traffic Light Controller FSM VHDL Testbench Code.

Note: Continued on next page

```
39  BEGIN
40
41     -- Instantiate the Unit Under Test (UUT)
42     uut: traffic_controller PORT MAP (
43            clk => clk,
44            reset => reset,
45            R => R,
46            G => G,
47            Y => Y
48          );
49     reset <= '1', '0' after 5 ns, '0' after 100 ns;
50     -- Clock process definitions
51     clk_process :process
52     begin
53        clk <= '0';
54        wait for clk_period/2;
55        clk <= '1';
56        wait for clk_period/2;
57     end process;
58
59     -- Stimulus process
60     stim_proc: process
61     begin
62        -- hold reset state for 100 ns.
63        wait for 100 ns;
64
65        wait for clk_period*10;
66
67        -- insert stimulus here
68
69        wait;
70     end process;
71
72  END;
```

Figure 4-3 (cont.): Traffic Light Controller FSM VHDL Testbench Code.

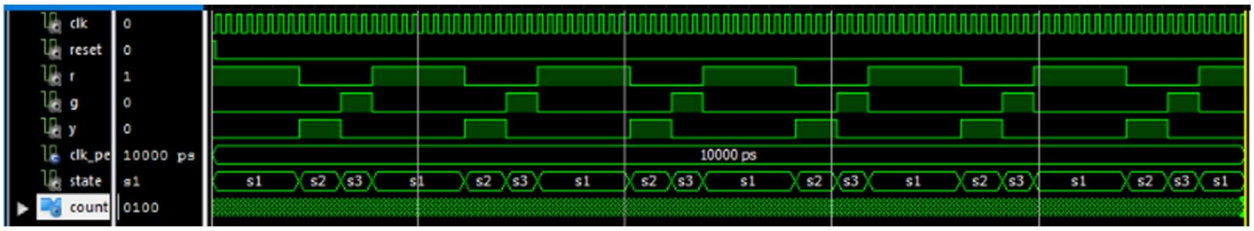

Figure 4-4: Traffic Light Controller FSM VHDL Waveform Simulation.

```
1
2  NET "Y" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
3  NET "R" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
4  NET "G" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
5
6  NET "reset" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN;
7
8  NET "clk" LOC = "C9"      | IOSTANDARD = LVCMOS33;
```

Figure 4-5: Traffic Light Controller User Constraint File

Part 5: RAM Design and Logic Analyzer

Description: In your RAM design, write 4'b1010 into 16 address location of the RAM. After a complete, writing, read data out of RAM.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity ram is
6  port (
7         address :in    std_logic_vector ( 3 downto 0);
8         data    :inout std_logic_vector ( 7 downto 0);
9         cs      :in    std_logic;
10        we      :in    std_logic;
11        oe      :in    std_logic
12     );
13 end ram;
14 architecture beh_ram of ram is
15
16 type memory is array (0 to 15)of std_logic_vector (7 downto 0);
17 signal mem : memory ;
18
19 begin
20
21
22       MEM_WRITE:
23       process (address, data, cs, we)
24      begin
25          if (cs = '1' and we = '1') then
26              mem(conv_integer(address)) <= data;
27          end if;
28       end process;
29
30
31       MEM_READ:
32      process (address, cs, we, oe, mem) begin
33           if (cs = '1' and we = '0' and oe = '1')  then
34       data <= mem(conv_integer(address));
35      else
36       data <=  (others => 'Z' );
37
38          end if;
39       end process;
40
41   end beh_ram;
```

Figure 5-1: RAM VHDL Code

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_unsigned.all;
4  use IEEE.std_logic_arith.all;
5
6  entity mem_fsm is
7    port ( clk, reset : IN std_logic;
8           address : OUT std_logic_vector(3 downto 0);
9           data:     INOUT std_logic_vector(7 downto 0);
10        cs, we, oe: OUT std_logic );
11 end mem_fsm;
12
13 architecture fsm_beh of mem_fsm is
14   type state_type is (idle, s1,s2,s3,s4);
15   signal state: state_type ;
16   signal cnt: std_logic_vector(3 downto 0);
17
18 begin
19
20   cs <= '1';
21   address <= cnt;
22
23   process (clk,reset)
24   begin
25     if (reset ='1') then
26       state   <= idle;
27       cnt     <= "0000";
28
29     elsif (clk='1' and clk'event) then
30       case state is
31    when idle =>
32       state <= s1;
33                cnt   <= "0000";
34
35         when s1 =>
36       state <= s2;
37                cnt   <= "0000";
38
39         when s2 =>
40       cnt   <=   cnt + 1;
41       if (cnt  < 15) then
42          state <= s2;
43       else
44          state <= s3;
45       end if;
```

Figure 5-2: RAM FSM VHDL Code

Note: Continued on next page.

```
46
47          when s3 =>   state <= s4;
48                   cnt  <= "0000";
49
50     when s4 =>
51        cnt  <= cnt + 1;
52        state <= s4;
53
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 <= "ZZZZZZZZ";
67          when s1 => we <= '1'; oe <= '0';  data <= "11000011";
68          when s2 => we <= '1'; oe <= '0';  data <= "11000011";
69          when s3 => we <= '0'; oe <= '1';  data <= "ZZZZZZZZ";
70          when s4 => we <= '0'; oe <= '1';  data <= "ZZZZZZZZ";
71     when others => we <= '0'; oe <= '0';   data <= "ZZZZZZZZ";
72          end case;
73    end process;
74
75  end fsm_beh;
```

Figure 5-3: RAM FSM VHDL Code

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY mem_top IS
    port(
       clk   : IN  std_logic;
       reset : IN  std_logic;
       data  : inout std_logic_vector ( 7 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 mem_top;

ARCHITECTURE behavior OF mem_top IS

    component ram
    port (
       address :in    std_logic_vector ( 3 downto 0);
       data    :inout std_logic_vector ( 7 downto 0);
       cs      :in    std_logic;
       we      :in    std_logic;
       oe      :in    std_logic
    );
    end component;

    COMPONENT mem_fsm
    PORT(
         clk   : IN  std_logic;
         reset : IN  std_logic;
         address : OUT  std_logic_vector(3 downto 0);
         data  : INOUT std_logic_vector(7 downto 0);
         cs : OUT  std_logic;
         we : OUT  std_logic;
         oe : OUT  std_logic
        );
    END COMPONENT;

    signal address :    std_logic_vector ( 3 downto 0);

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

Figure 5-4: RAM Top Module VHDL Code

Note: Continued on next page.

```
46
47  BEGIN
48
49      cs_out <= cs;
50      we_out <= we;
51      oe_out <= oe;
52      addr_out <= address;
53
54      g1: mem_fsm PORT MAP (
55              clk     => clk,
56              reset   => reset,
57              address => address,
58              data => data,
59              cs => cs,
60              we => we,
61              oe => oe
62           );
63
64      g2:  ram
65      port map(
66          address =>  address,
67          data    =>  data,
68          cs      => cs,
69          we      => we,
70          oe      => oe
71       );
72
73  END;
```

Figure 5-4 (cont.): RAM Top Module VHDL Code

```
1   NET  "address_out<0>"   LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
2   NET  "address_out<1>"   LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;
3   NET  "address_out<2>"   LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;
4   NET  "address_out<3>"   LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;
5
6   NET  "reset"     LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
7   NET  "clk"      LOC = "C9"  | IOSTANDARD = LVCMOS33;
8
9   NET  "data<0>"  LOC = "B4" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
10  NET  "data<1>"  LOC = "A4" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
11  NET  "dout<2>"  LOC = "D5" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
12  NET  "dout<3>"  LOC = "C5" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
13  NET  "data<4>"  LOC = "A6" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
14  NET  "data<5>"  LOC = "B6" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
15  NET  "dout<6>"  LOC = "E7" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
16  NET  "dout<7>"  LOC = "F7" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
17
18  NET  "oe_out"  LOC = "D7" | IOSTANDARD = LVTTL | SLEW = FAST|DRIVE = 8;
19  NET  "cs_out"  LOC = "C7" | IOSTANDARD = LVTTL | SLEW = FAST|DRIVE = 8;
20  NET  "we_out"  LOC = "F8" | IOSTANDARD = LVTTL | SLEW = FAST|DRIVE = 8;
```

Figure 5-5: RAM User Constraint File

Demo:

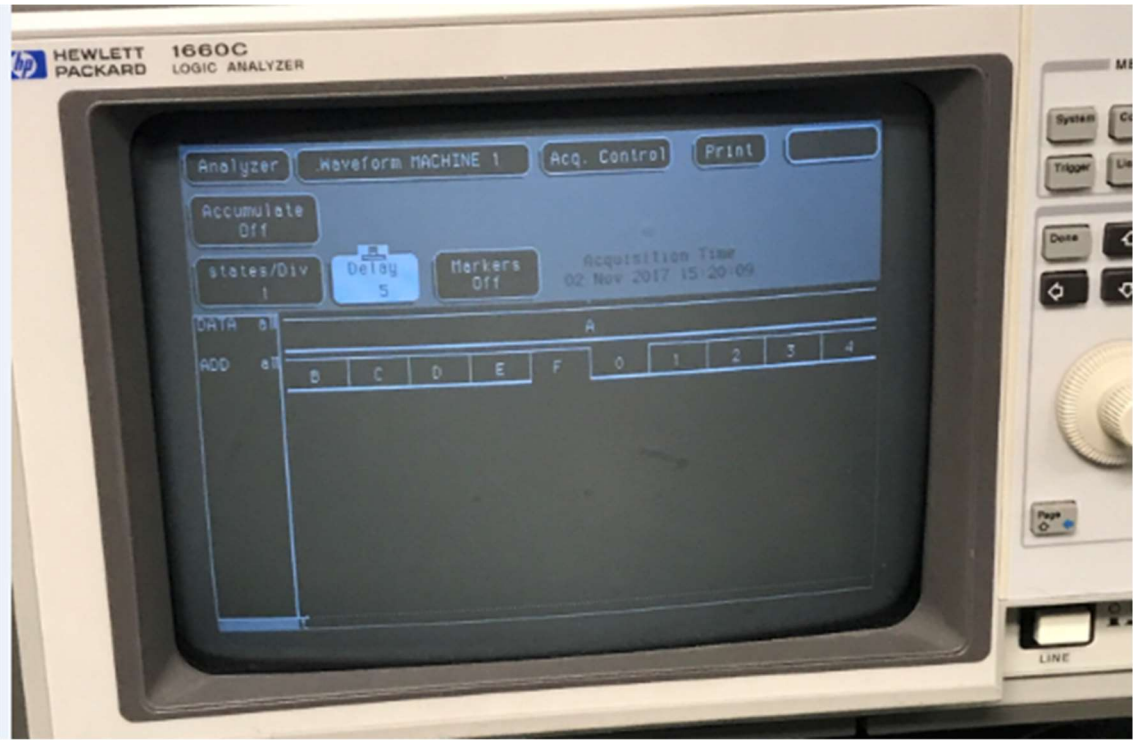

Figure 5-6: Logic Analyzer reading data out at 10KHz.

Conclusion: The purpose of this lab was to learn structural design strategy using VHDL, learn how to use logic analyzer for timing and state mode measurements, and learn finite state machine design using VHDL. After learning the tools to design and test the designs using VHDL testbench simulation then it is ready to be programmed to the Spartan3E board and we can verify that it works as anticipated. If for any reason it did not work then the testbench is the starting point to debug the design to find the problem. Learning how to use the Logic Analyzer is also a tool now available to test different designs requiring different clock frequencies.