CPE166 Advanced Logic Design

Lab 3 Report

Student Name: Mario Palacios

Date: 05/01/2020

California State University, Sacramento

TABLE OF CONTENTS

1) Introduction:

VHDL is a hardware description language used a lot in the industry. It's functionality in electronic design automation makes it valuable to describe digital and mixed-signal systems; such as Field-Programmable Gate Arrays (FPGA) and integrated circuits. Both Verilog and VHDL can be used to create logical circuits, however VHDL uses a higher level of description than Verilog. In this lab we were assigned the task to create 1MHz BCD Up-Down Counter, Pseudo Generator, a RAM Design. With the purpose of learning structural design using VHDL and finite state machines using VHDL. We were also tasked to explore the Xilinx Logic Analyzer (ILA) to monitor internal design signals, but due to school closing we had no access to the NEXYS4 DDR FPGA board.

2) 1MHz BCD Up-Down Counter Design and Logic Analyzer – Part 1

2.1) Project 1 Design Purpose:

The NEXYS4 DDR FPGA board has a 100MHz crystal oscillator which is controlled by pin E3, however different applications may require different clock frequencies. For this part of the lab we are tasked to use clock division to make the initial FPGA clock frequency go down to 1MHz. This clock frequency value is needed for a Binary Coded Decimal (BCD) counter because 100MHz is too fast and will not allow us to visually see the LEDs change.

2.2) Engineering Data:

In order to see the LEDs on the FPGA board we need to lower the clock frequency by using a clock divider. The clock divider takes in an input signal of frequency and generates an output signal smaller than the input frequency. Once the clock frequency goes from the native 100MHz to 1MHz, we also needed another input of a switch named up_down that determines if the BCD Counter will cout up from 0 or count down from 9. The output will be a 4-bit variable named dout, and this will represent the 4-bit BCD counter value.

2.2.1) Design Source Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity myBCD is
  Port (
        clk: in std_logic;
        up_down: in std_logic;
        clk2: out std_logic;
        dout: out std_logic_vector(3 downto 0));
end myBCD;

architecture Behavioral of myBCD is
    signal tmp: std_logic_vector(3 downto 0);
    signal cnt_div: std_logic_vector(9 downto 0);
    signal clk_div: std_logic;
```

```
BEGIN
----------------------------Clock Divider----------------------------
   process(clk)
   begin
      if(rising_edge(clk)) then
         if(cnt_div = 99) then
            clk_div <= '1';
            cnt_div <= (others => '0');
         elsif(cnt_div < 49) then
            clk_div <= '1';
            cnt_div <= cnt_div + 1;
         else
            clk_div <= '0';
            cnt_div <= cnt_div + 1;
         end if;
      end if;
   end process;
   clk2 <= clk_div;
-------------------------BCD Counter Design-------------------------
   process(clk_div,up_down)
   begin
      tmp <= "0000";
      if(rising_edge(clk_div)) then
         if(up_down = '1') then
            tmp <= tmp + 1;
            if (tmp = 9) then
               tmp <= "0000";
            end if;
         end if;
         if (up_down = '0') then
            tmp <= tmp - 1;
            if (tmp = 0) then
               tmp <= "1001";
            end if;
         end if;
      end if;
   end process;
   dout <= tmp;
end Behavioral;
```

2.2.2) User Constraint File

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { clk }]

##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { up_down }];

## LEDs
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { dout[3] }];
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { dout[2] }];
set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { dout[1] }];
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { dout[0] }];
```

2.2.3) Waveform and Results Discussion

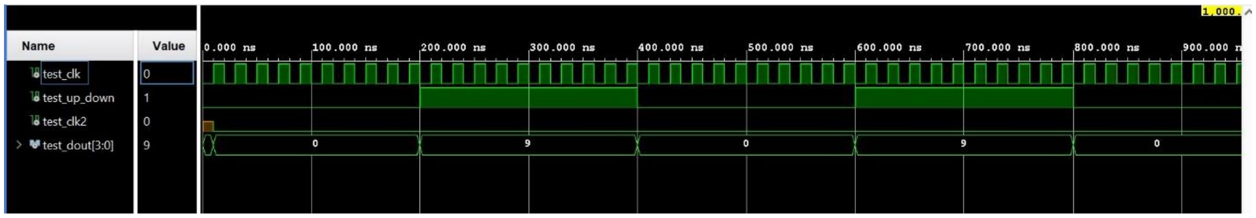


Figure 1. Waveform for BCD Counter

Results Discussion:
We were tasked to use the Logic Analyzer on the FPGA board, however I was not able to use due to course instruction being moved to online only. As a result we have the waveform as seen in Figure 1. The program is supposed to count down from which ever value it would be at if the switch is set to low, or count up from the current value it is at when the switch it set high. The count is set to the bounds of zero and nine. If the switch is set to low and reaches zero, instead of continuing to count down, past zero, it would reset the count to nine and continue to count down. The same concept goes for when the switch is set to high, once nine is reached the count resets at zero. This lab required me to convert a 100MHz clock to 1Hz, I included the code design in the source code section. I could not display it on the FPGA board due to the quarantine, but it can be see that there is a conversion happening in my code.

3) Pseudorandom Number Generator and Hamming Code – Part 2

3.1) Project Design Purpose

The objective of this lab is to design a pseudorandom number generator that uses a Linear Feedback Shift Register (LFSR) in the order of $X^4 + X + 1$, as shown in Figure 2. The LFSR consists of four D-Flip Flops that will then be used to output a 4-bit value at 1Hz. Accompanying the LFSR is a (7,4) Hamming Code, that will be constructed and displayed. In order to determine what which of the two will be displayed, a FSM will be implemented. The FSM will have state 0 as its IDLE state, state 1 as the LFSR, and state 2 as the (7,4) Hamming Code; for a total of three states.

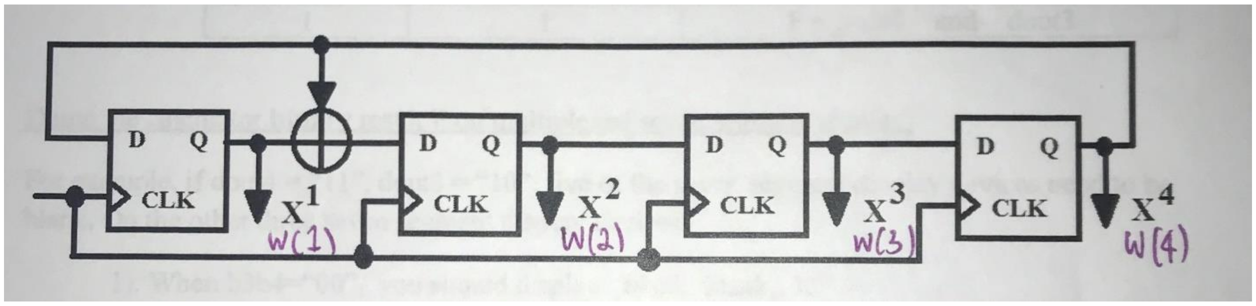


Figure 2. LFSR Circuit ($X^4 + X + 1$)

3.2) LFSR Source Code, Testbench, Simulation Waveform, and Results Discussion

Source Code:

```
------------------------------------ LFSR --------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity myLFSR is
  Port (
       rst,clk: IN STD_LOGIC;
       Q: OUT STD_LOGIC_VECTOR(4 downto 1));
end myLFSR;

architecture Behavioral of myLFSR is
    signal W: std_logic_vector(4 downto 1);
begin
    process(clk,rst)
    begin
        if(rst='1') then
            W <= ("0101");
        elsif(rising_edge(clk)) then
            W <= W(3 downto 2) & (W(1) xor W(4)) & W(4);
        end if;
    end process;
    Q <= W;
end Behavioral;

------------------------------- CLK DIVISION ---------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.all;
use IEEE.STD_LOGIC_unsigned.all;

entity clockDivision is
  Port (
       clk: IN STD_LOGIC;
       clk_div: OUT STD_LOGIC);
end clockDivision;

architecture Behavioral of clockDivision is
    signal cnt_div: std_logic_vector(25 downto 0);
    signal clk2: std_logic;

begin
    process(clk)
    begin
        if(rising_edge(clk)) then
```

```
            if(cnt_div = 49999999) then
                clk2 <= '1';
                cnt_div <= (others => '0');
             elsif(cnt_div < 24999999) then
                clk2 <= '1';
                cnt_div <= cnt_div + 1;
             else
                clk2 <='0';
                cnt_div <= cnt_div + 1;
             end if;
         end if;
     end process;
     clk_div <= clk2;
end Behavioral;
```

Testbench:

```
---------------------------THIS IS A TESTBENCH--------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity myLFSR_tb is
end myLFSR_tb;

architecture Behavioral of myLFSR_tb is
Component myLFSR
    PORT (
          rst,clk: IN STD_LOGIC;
          Q: OUT STD_LOGIC_VECTOR(4 downto 1));
END Component;

signal reset,clock: std_logic;
signal Q_out:       std_logic_vector(4 downto 1);

BEGIN
    uut: myLFSR port map(rst=>reset,clk=>clock,Q=>Q_out);

    Process
    begin
        clock<='0';
        wait for 100 ns;
        clock<='1';
        wait for 100 ns;
    end Process;

    Process
    begin
        reset<='1';
        wait for 60 ns;
        reset<='0';
        wait for 90 ns;
        wait;
    end Process;
end Behavioral;
```

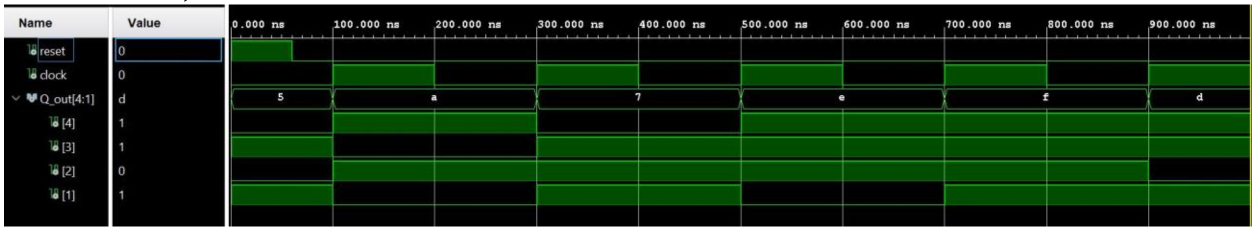


Figure 3. LFSR Simulation Waveform

Results Discussion:
A LFSR uses an exclusive-or (XOR) gate, as the most commonly used liner function of single bits. The sequence of numbers for this specific method is generated at random, as seen in the above Figure 3.

<u>3.3) (7, 4) Hamming Code Source Code, Testbench, Simulation Waveform and Result Discussion</u>

Source Code:

```
------------------------------------ HAMMING -------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity myHAM is
  Port (
        d: IN STD_LOGIC_VECTOR(3 downto 0);
        hamming: OUT STD_LOGIC_VECTOR(6 downto 0));
end myHAM;

architecture Behavioral of myHAM is
    signal p: std_logic_vector(2 downto 0);
begin
    p(2)<=d(3) xor d(2) xor d(1);
    p(1)<=d(3) xor d(2) xor d(0);
    p(0)<=d(3) xor d(1) xor d(0);
    hamming <= d(3) & d(2) & d(1) & p(2) & d(0) & p(1) & p(0);
end Behavioral;
```

Testbench:

```
---------------------------THIS IS A TESTBENCH--------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity myHam_tb is
end myHam_tb;

architecture Behavioral of myHam_tb is
```

```
    COMPONENT myHAM port (
        d: IN STD_LOGIC_VECTOR(3 downto 0);
        hamming: OUT STD_LOGIC_VECTOR(6 downto 0));
    end COMPONENT;

    signal test_d: std_logic_vector(3 downto 0);
    signal ham_out: std_logic_vector(6 downto 0);

BEGIN
    uut: myHAM port map (d => test_d, hamming => ham_out);

    Process
    begin
        test_d <= "0001";
        wait for 10 ns;
        test_d <= "1001";
        wait for 10 ns;
        test_d <= "0101";
        wait for 10 ns;
        wait;

    end Process;
end Behavioral;
```

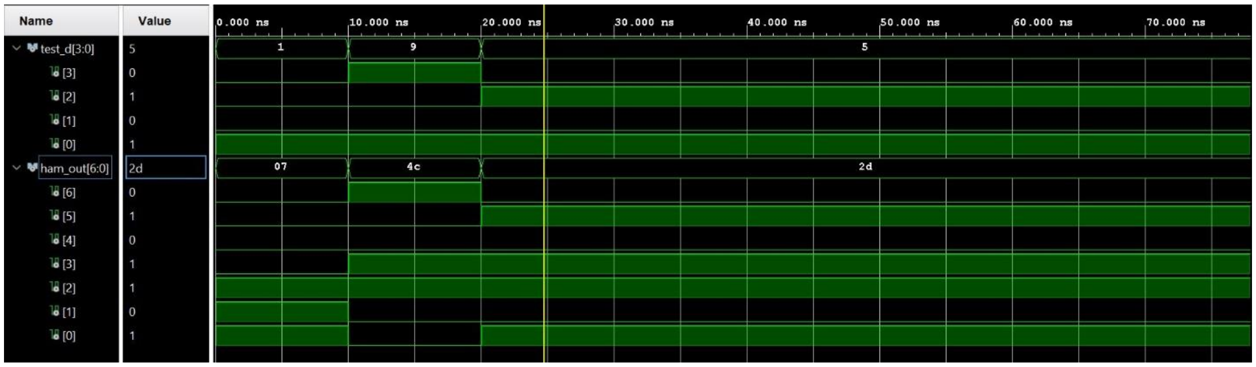
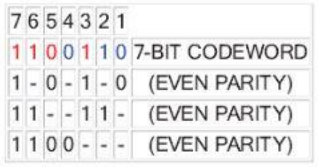


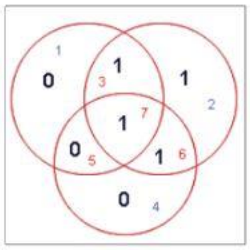
Figure 4. (7, 4) Hamming Code Simulation Waveform

Results Discussion:
The Hamming code is used to detect and correct errors that can occur during the movement or storage of data, from either the sender or receiver side. It has the ability to correct single bit errors at a cost which is less than sending an entire message twice. The simulation waveform in Figure 4, displays when given a value it will generate parity bits. Figure 5 below shows how when a transmitted message is received with a single bit error. It uses the parity bits to determine where the error is located.

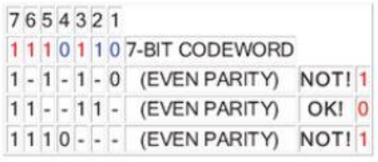
Transmitted:
1100110



7	6	5	4	3	2	1	
1	1	0	0	1	1	0	7-BIT CODEWORD
1	-	0	-	1	-	0	(EVEN PARITY)
1	1	-	-	1	1	-	(EVEN PARITY)
1	1	0	0	-	-	-	(EVEN PARITY)



Received:
111011



7	6	5	4	3	2	1			
1	1	1	0	1	1	0	7-BIT CODEWORD		
1	-	1	-	1	-	0	(EVEN PARITY)	NOT!	1
1	1	-	-	1	1	-	(EVEN PARITY)	OK!	0
1	1	1	0	-	-	-	(EVEN PARITY)	NOT!	1

Figure 5. Example of How Even Parity is Used to Correct a Single-Bitt Error

3.4) Finite State Machine Source Code, Testbench, Simulation Waveform and Result Discussion

Source Code:

```
--------------------------------------- FSM ------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity myFSM is
  Port (
        clk: IN STD_LOGIC;
        --Switch1 = IDLE; Switch2 = LFSR; Switch3 = HAMMING
        switch1,switch2,switch3: IN STD_LOGIC;
        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));
end myFSM;

architecture Behavioral of myFSM is
    type state_type is (s0,s1,s2);
    signal state: state_type;
    signal lfsr_cap: std_logic_vector(3 downto 0);
    signal hamming_cap: std_logic_vector(6 downto 0);

begin
    state_process: process (clk,switch1,switch2,switch3)
    begin
        if(switch1 = '0') and (switch2 = '0') and (switch3 = '0') and rising_edge(clk) then
            state <= s0;
        elsif (switch1 = '1') then
            state <= s0;
        elsif (rising_edge(clk)) then
            if (switch2 = '1') then
                lfsr_cap <= lfsr_in;
                hamming_cap <= hamming_in;
             end if;
        case state is
            when s0 =>
                if(switch2 = '1') then
                    state <= s1;
                elsif (switch3 = '1') then
                    state <= s2;
                else
                    state <= s0;
                end if;
            when s1 =>
                if(switch1 = '1') then
                    state <= s0;
                elsif(switch3 = '1') then
                    state <= s2;
                else
                    state <= s1;
                end if;
            when s2 =>
                if(switch1 = '1') then
                    state <= s0;
                elsif (switch2 = '1') then
                    state <= s1;
                else
                    state <= s2;
                end if;
            when others =>
                state <= s0;
        end case;
        end if;
    end process;

    output_process: 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 Behavioral;
```

Testbench:

```
---------------------------THIS IS A TESTBENCH---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity myFSM_tb is
end myFSM_tb;

architecture Behavioral of myFSM_tb is
```

```
  COMPONENT myFSM Port (
                  clk: IN STD_LOGIC;
                  --Switch1 = IDLE; Switch2 = LFSR; Switch3 = HAMMING
                  switch1,switch2,switch3: IN STD_LOGIC;
                  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));
  end COMPONENT;

  signal sw1,sw2,sw3,clock: std_logic;
  signal ham,fsm: std_logic_vector(6 downto 0);
  signal lfsr: std_logic_vector(3 downto 0);

BEGIN
  uut: myFSM Port map(clk => clock,switch1 => sw1,switch2 => sw2,
                              switch3 => sw3,hamming_in => ham,lfsr_in => lfsr,
                              Sfsm_out => fsm);

  Process
  begin
    clock <= '0';
    wait for 10 ns;
    clock <= '1';
    wait for 10ns;
  end Process;

  Process
  begin
    sw1 <= '0';
    sw2 <= '0';
    sw3 <= '0';
    wait for 30 ns;

    sw1 <= '1';
    wait for 40 ns;

    sw2 <= '1';
    sw1 <= '0';
    wait for 40 ns;

    sw3 <= '1';
    sw2 <= '0';
    wait for 40 ns;

    sw3 <= '0';
    sw1 <= '1';
    wait for 40 ns;

    sw1 <= '0';
    wait for 40 ns;
    wait;
    end Process;
end Behavioral;
```

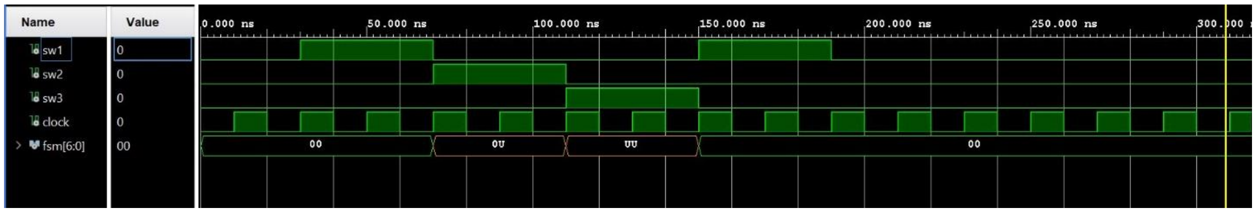


Figure 6. FSM Simulation Waveform

Results Discussion:
I tested out if the switches would take us to the correct state, when one is set to high. As you can see there are some undefined variables and that is because I have not included their code in my FSM Hierarchy. This is because it will be included in the next section labeled TOP. However when switch 1 is set to high it goes to the IDLE state, where the FSM output will be zero. When switch 2 is set high it goes to state 2, that incorporates the LFSR, and the FSM output contain four-bits of zero and the last four bits for the random pseudonumber. Lastly when switch 3 is set high the (7, 4) Hamming Code is generated, that will include an 8-bit generated number.

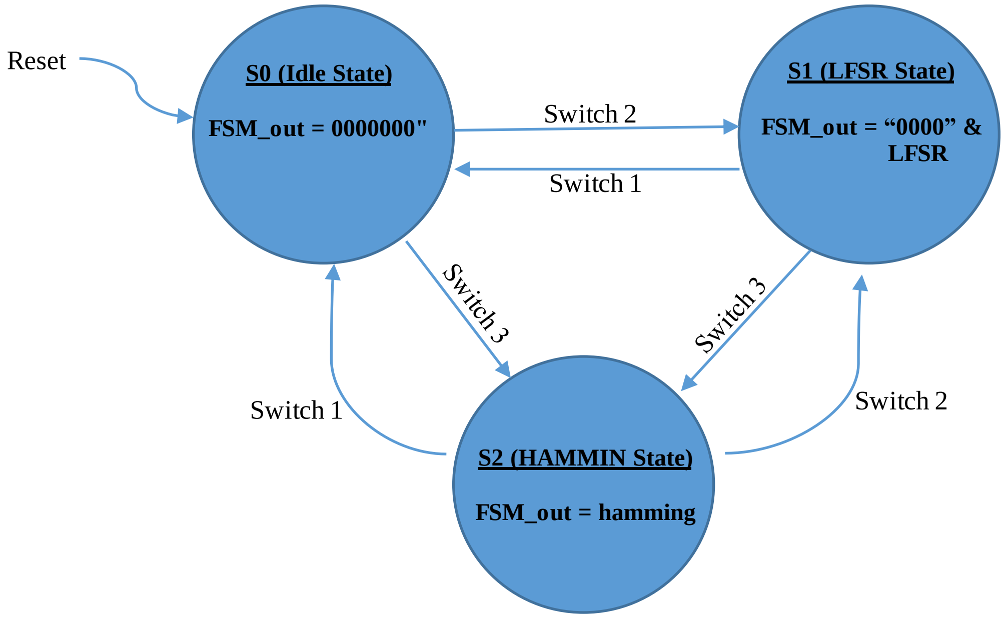


Figure 7. FSM Diagram

3.5 Top Level Design, Testbench, Constraints, Simulation Waveform and Result Discussion

Source Code:

```
----------------------------------- TOP -----------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TOP is
  Port (
        clock,reset,sw1,sw2,sw3: IN STD_LOGIC;
        led_out: OUT STD_LOGIC_VECTOR(6 downto 0));
end TOP;

architecture Behavioral of TOP is
    signal clk_oneHZ: std_logic;
    signal lfsrTOP_out: std_logic_vector(3 downto 0);
    signal hammingTOP_out: std_logic_vector(6 downto 0);

    component clockDivision
    port (
      --   clk: IN STD_LOGIC;
         clk_div: OUT STD_LOGIC);
    end component;
    component myLFSR
    port (
          rst,clk: IN STD_LOGIC;
          Q: OUT STD_LOGIC_VECTOR(4 downto 1));
    end component;
    component myHAM
    port (
        d: IN STD_LOGIC_VECTOR(3 downto 0);
        hamming: OUT STD_LOGIC_VECTOR(6 downto 0));
    end component;
    component myFSM
    port (
        clk: IN STD_LOGIC;
        --Switch1 = IDLE; Switch2 = LFSR; Switch3 = HAMMING
        switch1,switch2,switch3: IN STD_LOGIC;
        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));
    end component;
begin
    --g1: clockDivision port map(clk => clock, clk_div => clk_oneHZ);
    g2: myFSM port map(clk => clock,switch1 => sw1,switch2 => sw2,switch3 => sw3,
                    hamming_in => hammingTOP_out,lfsr_in => lfsrTOP_out,fsm_out => led_out);
    g3: myHAM port map(d => lfsrTOP_out,hamming => hammingTOP_out);
    g4: myLFSR port map(rst => reset,clk => clock,Q => lfsrTOP_out);
end Behavioral;
```

Testbench:

```
---------------------------THIS IS A TESTBENCH---------------------------
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 tb_beh of TOP_tb is
   component TOP
   port (
      clock,reset,sw1,sw2,sw3: IN STD_LOGIC;
      led_out: OUT STD_LOGIC_VECTOR(6 downto 0));
   end component;

   signal test_clock: std_logic;
   signal test_reset: std_logic;
   signal test_sw1: std_logic;
   signal test_sw2: std_logic;
   signal test_sw3: std_logic;
   signal test_led_out: std_logic_vector(6 downto 0);

begin
   -- Circuit Under Test
   uut: TOP port map (clock=>test_clock,reset=>test_reset,sw1=>test_sw1,
                              sw2=>test_sw2,sw3=>test_sw3,led_out=>test_led_out);

   -- SET CLOCK
   Process
   begin
      test_clock<='0';
      wait for 1 ns;
      test_clock<='1';
      wait for 1 ns;
   end Process;

   -- SET RESET
   Process
   begin
      test_reset<='1';
      wait for 2 ns;
      test_reset<='0';
      wait for 5 ns;
      wait;
   end Process;

   --SET INPUTS
   process
   begin
```

```
        test_sw1 <= '0';
        test_sw2 <= '0';
        test_sw3 <= '0';

        for i in 0 to 4 loop
            test_sw1 <= '1';
            wait for 2 ns;
            test_sw1 <= '0';
            wait for 5 ns;

            test_sw2 <= '1';
            wait for 2 ns;
            test_sw2 <= '0';
            wait for 5 ns;

            test_sw3 <= '1';
            wait for 2 ns;
            test_sw3 <= '0';
            wait for 5 ns;
        end loop;
        wait;
    end process;
end tb_beh;
```

Constraints:

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clock }];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { clock }]

##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { sw1 }];
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { sw2 }];
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { sw3 }];
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { reset }];

## LEDs
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { led_out[6] }];
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { led_out[5] }];
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { led_out[4] }];
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { led_out[3] }];
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { led_out[2] }];
set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { led_out[1] }];
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { led_out[0] }];
```

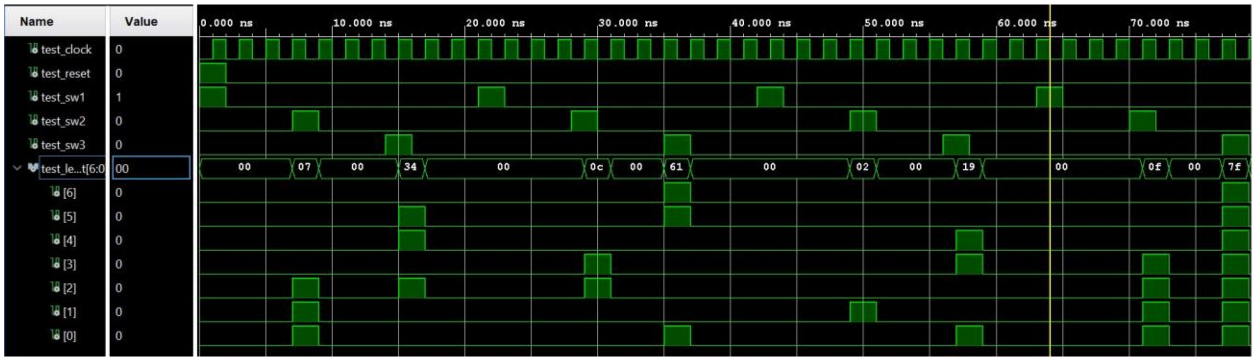


Figure 9. TOP Simulation Waveform

Results Discussion:
The final module created was named TOP because it contains all the modules into one concise module. The finite state machine takes the input from the FPGA and then determines to either use the LFSR or (7, 4) Hamming to display the output on LED lights. Due to the quarantine I was not able to display this on the FPGA board, and I tried my best to create a constraint file. Overall it can be seen in Figure 9, that when a switch is changed the proper output is being displayed.

4) RAM Design and Logic Analyzer - Part 3

4.1) Design Purpose

In the part of the lab we were tasked to create a RAM design in VHDL. In our RAM design we will write 4’b1010 into 16 memory address locations of the RAM. Since we could not use the FPGA board on the

4.2) 4 by 4 RAM Design Code, and Testbench

------------------------------------ *4 BY 4 RAM* ------------------------------------
Source Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity mySRAM is
  Port (
        address: in STD_LOGIC_VECTOR(3 downto 0);
        cs: in STD_LOGIC;
        we: in STD_LOGIC;
        oe: in STD_LOGIC;
        data: inout STD_LOGIC_VECTOR(3 downto 0));
end mySRAM;

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

BEGIN
    MEM_WRITIE:
```

```
    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 Behavioral;
```

Testbench:

```
----------------------------THIS IS A TESTBENCH---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RAM_tb is
end RAM_tb;

architecture Behavioral of RAM_tb is
    COMPONENT mySRAM  Port (
                        address: in STD_LOGIC_VECTOR(3 downto 0);
                        cs: in STD_LOGIC;
                        we: in STD_LOGIC;
                        oe: in STD_LOGIC;
                        data_in: inout STD_LOGIC_VECTOR(3 downto 0);
                        data_out: out STD_LOGIC_VECTOR(3 downto 0));
    end COMPONENT;

    signal addy: std_logic_vector (3 downto 0);
    signal din: std_logic_vector(3 downto 0);
    signal dout: std_logic_vector(3 downto 0);
    signal cs,we,oe: std_logic;

BEGIN
    uut: mySRAM port map    (address => addy,cs => cs, we => we, oe => oe,
                         data_in => din,data_out => dout);

    Process
    begin
        addy <= "0100";
        din <= "0101";
        cs <= '1';
        we <= '1';
        oe <= '0';
        wait for 5 ns;

        addy <= "1001";
        din <= "0110";
        wait for 5 ns;

        addy <= "0001";
        din <= "1100";
        wait for 5 ns;

        addy <= "0100";
        we <= '0';
        oe <= '1';
        wait for 5 ns;

        addy <= "1001";
        wait for 5 ns;

        addy <= "0001";
        wait for 5 ns;
        wait;
    end Process;
end Behavioral;
```

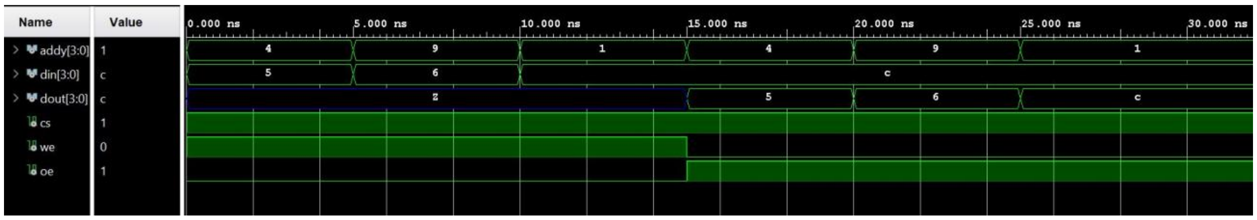


Figure 10. Four by Four RAM Design Simulation Waveform

4.3 RAM Controller Design Code and Testbench

Source Code:

```
-------------------------------------- FSM ------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity myFSM_SRAM is
    Port (
          clk,rst: in std_logic;
          cs,we,oe: out std_logic;
          addr: out std_logic_vector(3 downto 0));
```

```
end myFSM_SRAM;

architecture Behavioral of myFSM_SRAM is
    type state_type is (idle,s0,s1,s2,s3);
    signal state: state_type;
    signal cnt: std_logic_vector(3 downto 0);
BEGIN
    cs <= '1';
    addr <= cnt;

    process(clk,rst)
    begin
        if(rst = '1') then
            state <= idle;
            cnt   <= "0000";

        elsif (clk = '1' and clk'event) then
            case state is
                when idle =>
                    state <= s0;
                    cnt   <= "0000";
                when s0 =>
                    state <= s1;
                    cnt   <= "0000";
                when s1 =>
                    cnt <= cnt + 1;
                    if (cnt < 15) then
                        state <= s1;
                    else
                        state <= s2;
                    end if;
                when s2 =>
                    state <= s3;
                    cnt   <= "0000";
                when s3 =>
                    cnt   <= cnt + 1;
                    state <= s3;
                when others =>
                    state <= s0;
                    cnt   <= "0000";
             end case;
         end if;
    end process;

    process(state)
    begin
        case state is
            when idle =>
                we <= '0'; oe <= '0';
            when s0 =>
                we <= '1'; oe <= '0';
            when s1 =>
                we <= '1'; oe <= '0';
            when s2 =>
                we <= '0'; oe <= '1';
            when s3 =>
                we <= '0'; oe <= '1';
            when others =>
                we <= '0'; oe <= '0';
        end case;
    end process;
end Behavioral;
```

Testbench:

```
----------------------------THIS IS A TESTBENCH---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FSM_tb is
end FSM_tb;

architecture Behavioral of FSM_tb is
    COMPONENT myFSM_SRAM Port (
                            clk,rst: in std_logic;
                            cs,we,oe: out std_logic;
                            addr: out std_logic_vector(3 downto 0));
    end COMPONENT;

    signal cs,we,oe,clock,reset: std_logic;
    signal addy: std_logic_vector(3 downto 0);

BEGIN
-- CIRCUIT UNDER TEST --
    uut: myFSM_SRAM port map (clk => clock,rst => reset,cs => cs,
                                                 we => we,oe => oe,addr => addy);

-- SET CLOCK --
    process
    begin
        clock <= '0';
        wait for 10 ns;
        clock <= '1';
        wait for 10 ns;
    end process;

-- SET RESET --
    process
    begin
        reset <= '1';
        wait for 20 ns;
        reset <= '0';
        wait for 50 ns;
        wait;
    end process;
end Behavioral;
```

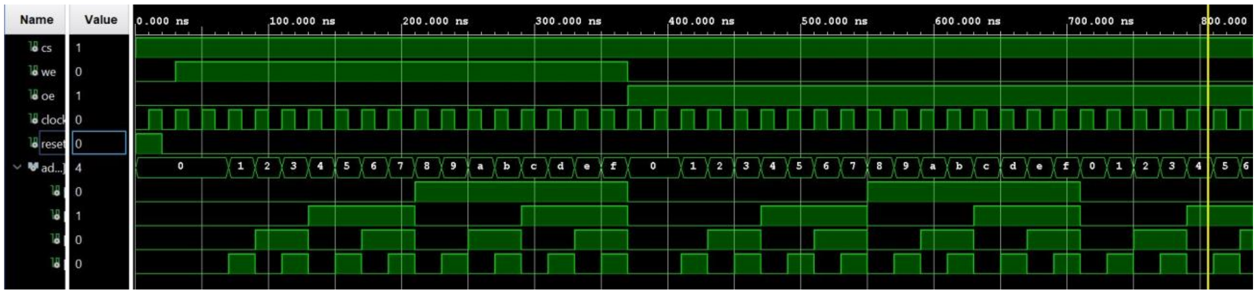


Figure 11. RAM Controller Design Simulation Waveform

<u>4.4) Top Level Design Code, Testbench, Simulation Waveform and Result Discussion</u>

Source Code:

```
------------------------------------ TOP ------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity top_SRAM is
    Port (
          clk, rst: in STD_LOGIC;
          data: inout STD_LOGIC_VECTOR(3 downto 0));
end top_SRAM;

architecture Behavioral of top_SRAM is
    COMPONENT mySRAM port (
                          address: in STD_LOGIC_VECTOR(3 downto 0);
                          cs: in STD_LOGIC;
                          we: in STD_LOGIC;
                          oe: in STD_LOGIC;
                          data_in: in STD_LOGIC_VECTOR(3 downto 0);
                          data_out: out STD_LOGIC_VECTOR(3 downto 0));
    end COMPONENT;

    COMPONENT myFSM_SRAM port (
                              clk,rst: in std_logic;
                              cs,we,oe: out std_logic;
                              addr: out std_logic_vector(3 downto 0));
    end COMPONENT;

    signal cs,we,oe: std_logic;
    signal address: std_logic_vector(3 downto 0);
    signal d: std_logic_vector(3 downto 0);

BEGIN
    data <= "1010" when (we = '1' and oe = '0') else "ZZZZ";
    d <= data;
```

```
   g1: mySRAM port map(
                  address => address,
                  cs => cs,
                  we => we,
                  oe => oe,
                  data_in => data,
                  data_out => data);

   g2: myFSM_SRAM port map(
                     clk => clk,
                     rst => rst,
                     cs => cs,
                     we => we,
                     oe => oe,
                     addr => address);

end Behavioral;
```

Testbench:

```
---------------------------THIS IS A TESTBENCH--------------------------
library IEEE;
use IEEE.STDLOGIC_1164.ALL;

entity mySRAM_tb is
end mySRAM_tb;

architecture Behavioral of mySRAM_tb is
   COMPONENT top_SRAM port (
                                    clk, rst: in STD_LOGIC;
                                    data: inout STD_LOGIC_VECTOR(3 downto 0));
   end COMPONENT;

   signal clock,reset: std_logic;
   signal data_test: std_logic_vector(3 downto 0);

BEGIN
-- CIRCUIT UNDER TEST --
   uut: top_SRAM port map (clk => clock,rst => reset,data => data_test);

-- SET CLOCK --
   process
   begin
      clock <= '0';
      wait for 10 ns;
      clock <= '1';
      wait for 10 ns;
   end process;

-- SET RESET --
   process
```

```
    begin
        data_test <= "1010";
        reset <= '1';
        wait for 20 ns;
        reset <= '0';
        wait for 50 ns;
        wait;
    end process;
end Behavioral;
```

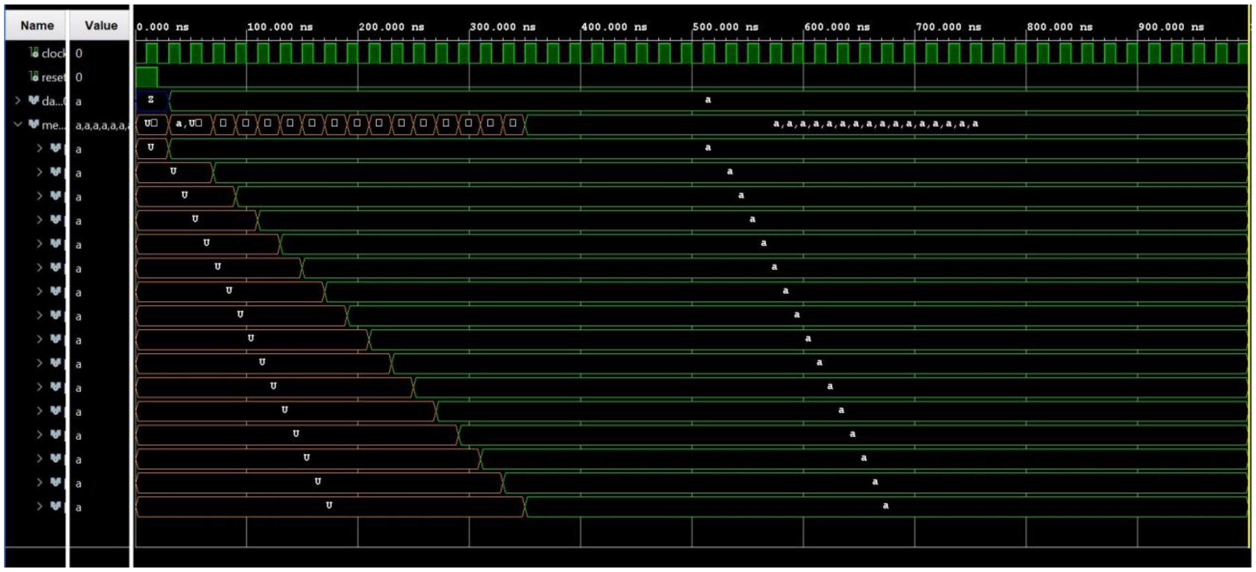


Figure 12. TOP Level Design Simulation Waveform Displaying "1010" in 16 Memory Locations

Results Discussion:
The results displayed in the Figure 12, prove I was able to meet this lab part's objective, of displaying "1010" (a in hex), in 16 memory locations. Due to the quarinteine I was not able to display this on an FPGA board, and I did not attempt to create a constraint file because I had no FPGA board. Overall this part of the lab was the most enjoyable and I had little issues creating it.

Conclusion:

In this lab, I learned and demonstrated various combination and sequential circuits using VHDL, while also implementing a Finite State Machine in VHDL for the first time. I gained more experience in creating hierarchical designs models to structure my code. I was able emphasis on how shift registers and hamming codes operate, while also using clock division in order to display values on an FPGA board. I was introduced to RAM architecture, with a focus on SRAM, and how it is able to write and read data in order to generate a bit stream. In the end this lab was enjoyable and I was able to complete the tasks assigned to me.