Sam Lee
CPE 166 – 03
Advanced Logic Design Lab
Wednesday: 5PM -7:50 PM
Lecture Professor: Jing Pang
Lab Professor: Eric Carmi
Lab Report #3

Contents

Introduction

VHDL, is another type of hardware description language originally founded in 1983, the hardware description language is used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits. VHDL can also be used as a general-purpose parallel programming language. Like Verilog, it accomplishes the same tasks needed to create logical circuits; with a slightly higher level of description than Verilog. In this lab, there were four main portions, that would utilize VHDL in order to create certain circuit designs. For instance, in the first part, we were tasked with created a 1MHz Up-Down BCD Counter that would count down initially, but once a switch was "1" it would begin counting up. In the second part of the lab, we were to design a pseudo random generator using a Linear Feedback Shift Register. When SW1 switch is pressed, the finite state machine (FSM) will be in the idle state and the LED displays will be blank. When SW2 switch is pressed, the LFSR value at that moment will be displayed as 4-bit original message data on the LEDs. When SW3 switch is pressed, the (7, 4) hamming code will be constructed and displayed on the LEDs. Furthermore, for Part 3 of the lab we needed to build an LED calculator that would use maximum length sequence LFSR X^4 + X + 1 to generate pseudorandom sequence at 1 Hz clock rate. For the final part of this lab, we needed to generate a RAM design that would write binary data "1010" into 16 address locations of the RAM. After completely writing, read data out of RAM and program it to the NEXYS 4DDR. In simpler terms the main objectives for this lab are summarize in the following:

- Design 1MHz BCD Up-Down Counter
- Design a pseudo random generator using a Linear Feedback Shift Register with Hamming Code
- Design a LED calculator using the previous modules such as the LFSR and make it perform calculator function according to the values of button b3 and b4.
- Create a Ram Design that would write binary data "1010" into 16 address locations of the RAM. After completely writing, read data out of RAM and program it to the NEXYS 4DDR.

Part 1 – 1 MHz BCD Up-Down Counter

Design Purpose

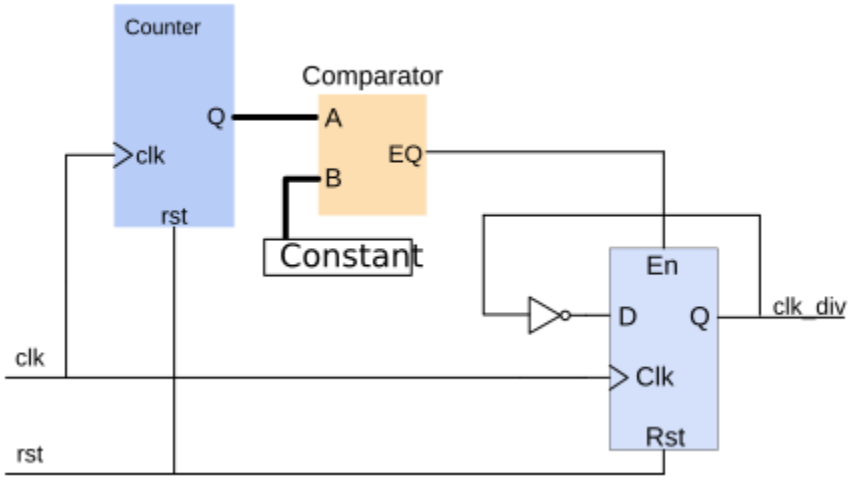
The primary objective of this part is to design a 1MHz BCD counter that would initially count down from 9 to 0. However, once a switch was enabled it will begin counting up. We needed to utilize clock division within our design in order to lower the native 100MHz frequency from the NEXYS 4DDR down to 1MHz; so that we could visible see the LEDs changing. Otherwise, they would be changing too fast for the naked eye to notice.

Engineering Data

BCD stands for binary coded decimal. It is used in this application because it makes it easier to display on the seven segment display. The inputs were the clock, a switch to control if the counter counted up or down, and the seven segment display. To display the number using the FPGA, a constraint file needed to be used to tell the FPGA which components would be used. This project consisted of one top level module which included the clock division process and the counting process. The clock division divided a 100 MHz clock down to a 1MHz clock so that the numbers changing could be seen from the human eye. The cnt_div variable is 27 bits because it would be enough bits to be able to count to 100M (100000000). The other count variable is used

to count between 0-9. A normal counter would just keep adding numbers until the program stops, but since we were only using one seven segment display, we could only show one decimal digit. This means the counter needed to reset once it reached 9 if it was counting up, and it would need to reset at 0 if it was counting down.

A frequency divider, also called a clock divider or scaler or prescaler, is a circuit that takes an input signal of a frequency and generates an output signal of a frequency usually smaller than the input frequency.



In this lab, it will essential to use a clock divider in order to bring down the 100MHz native frequency down to 1MHz. So, it should take 100000000 clock cycles before clk_div goes to '1' and returns to '0'. In another word, it takes 100000 clock cycles for clk_div to flip its value. So the constant we need to choose here is 100000.

Source Code

BCD Counter and Clk2 VHDL Code:

```
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;
  --load: in std_logic;
  updown: in std_logic;
  din: in std_logic_vector (3 downto 0);
  cntout: out std_logic_vector (3 downto 0);
  clkout: out std_logic
);
end bcdCounter;

architecture behavioral of bcdCounter is
```

```
--internal signals
signal cnt_div: std_logic_vector (9 downto 0);
signal cnt: std_logic_vector (3 downto 0);
signal clk2: std_logic;

begin

process(clk)
begin
if rising_edge (clk) then
  if(cnt_div = 99) then
    cnt_div <= (others => '0');
    clk2 <= '1';
  elsif (cnt_div < 49) then
    cnt_div <= cnt_div + 1;
    clk2 <= '1';
  else
    cnt_div <= cnt_div + 1;
    clk2 <= '0';
  end if;
end if;

end process;

process (clk2, updown) --process active on event of clk2,load, or updown
begin
--if(load = '1') then
 -- cnt <= din;
if rising_edge (clk2) then
if(updown = '1') then
  cnt <= cnt + 1;
else
  cnt <= cnt - 1;
end if;
end if;
end process;

cntout <= cnt; --output count
clkout <= clk2;--output clk
end behavioral;
```

User Constraint File

```
## Clock signal
set_property -dict {PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports clk]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]
```

```
#switch
set_property -dict {PACKAGE_PIN J15 IOSTANDARD LVCMOS33} [get_ports updown]

#leds
set_property -dict {PACKAGE_PIN H17 IOSTANDARD LVCMOS33} [get_ports {cntout[0]}]
set_property -dict {PACKAGE_PIN K15 IOSTANDARD LVCMOS33} [get_ports {cntout[1]}]
set_property -dict {PACKAGE_PIN J13 IOSTANDARD LVCMOS33} [get_ports {cntout[2]}]
set_property -dict {PACKAGE_PIN N14 IOSTANDARD LVCMOS33} [get_ports {cntout[3]}]
set_property -dict {PACKAGE_PIN V11 IOSTANDARD LVCMOS33} [get_ports clkout]

create_debug_core u_ila_0 ila
set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_EN_STRG_QUAL false [get_debug_cores u_ila_0]
set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
set_property port_width 1 [get_debug_ports u_ila_0/clk]
connect_debug_port u_ila_0/clk [get_nets [list clk_IBUF_BUFG]]
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe0]
set_property port_width 10 [get_debug_ports u_ila_0/probe0]
connect_debug_port u_ila_0/probe0 [get_nets [list {cnt_div_reg__0[0]} {cnt_div_reg__0[1]}
{cnt_div_reg__0[2]} {cnt_div_reg__0[3]} {cnt_div_reg__0[4]} {cnt_div_reg__0[5]}
{cnt_div_reg__0[6]} {cnt_div_reg__0[7]} {cnt_div_reg__0[8]} {cnt_div_reg__0[9]}]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe1]
set_property port_width 4 [get_debug_ports u_ila_0/probe1]
connect_debug_port u_ila_0/probe1 [get_nets [list {cntout_OBUF[0]} {cntout_OBUF[1]}
{cntout_OBUF[2]} {cntout_OBUF[3]}]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe2]
set_property port_width 8 [get_debug_ports u_ila_0/probe2]
connect_debug_port u_ila_0/probe2 [get_nets [list {plusOp[1]} {plusOp[3]} {plusOp[4]}
{plusOp[5]} {plusOp[6]} {plusOp[7]} {plusOp[8]} {plusOp[9]}]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe3]
set_property port_width 1 [get_debug_ports u_ila_0/probe3]
connect_debug_port u_ila_0/probe3 [get_nets [list clear]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe4]
set_property port_width 1 [get_debug_ports u_ila_0/probe4]
connect_debug_port u_ila_0/probe4 [get_nets [list clk2_i_2_n_0]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe5]
```

```
set_property port_width 1 [get_debug_ports u_ila_0/probe5]
connect_debug_port u_ila_0/probe5 [get_nets [list clk2_i_3_n_0]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe6]
set_property port_width 1 [get_debug_ports u_ila_0/probe6]
connect_debug_port u_ila_0/probe6 [get_nets [list clk2_i_4_n_0]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe7]
set_property port_width 1 [get_debug_ports u_ila_0/probe7]
connect_debug_port u_ila_0/probe7 [get_nets [list clk_IBUF]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe8]
set_property port_width 1 [get_debug_ports u_ila_0/probe8]
connect_debug_port u_ila_0/probe8 [get_nets [list clkout_OBUF]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe9]
set_property port_width 1 [get_debug_ports u_ila_0/probe9]
connect_debug_port u_ila_0/probe9 [get_nets [list {cnt[0]_i_1_n_0}]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe10]
set_property port_width 1 [get_debug_ports u_ila_0/probe10]
connect_debug_port u_ila_0/probe10 [get_nets [list {cnt[1]_i_1_n_0}]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe11]
set_property port_width 1 [get_debug_ports u_ila_0/probe11]
connect_debug_port u_ila_0/probe11 [get_nets [list {cnt[2]_i_1_n_0}]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe12]
set_property port_width 1 [get_debug_ports u_ila_0/probe12]
connect_debug_port u_ila_0/probe12 [get_nets [list {cnt[3]_i_1_n_0}]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe13]
set_property port_width 1 [get_debug_ports u_ila_0/probe13]
connect_debug_port u_ila_0/probe13 [get_nets [list {cnt_div[0]_i_1_n_0}]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe14]
set_property port_width 1 [get_debug_ports u_ila_0/probe14]
connect_debug_port u_ila_0/probe14 [get_nets [list {cnt_div[2]_i_1_n_0}]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe15]
set_property port_width 1 [get_debug_ports u_ila_0/probe15]
connect_debug_port u_ila_0/probe15 [get_nets [list {cnt_div[6]_i_2_n_0}]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe16]
set_property port_width 1 [get_debug_ports u_ila_0/probe16]
connect_debug_port u_ila_0/probe16 [get_nets [list {cnt_div[9]_i_2_n_0}]]
```

```
set_property C_CLK_INPUT_FREQ_HZ 300000000 [get_debug_cores dbg_hub]
set_property C_ENABLE_CLK_DIVIDER false [get_debug_cores dbg_hub]
set_property C_USER_SCAN_CHAIN 1 [get_debug_cores dbg_hub]
connect_debug_port dbg_hub/clk [get_nets clk_IBUF_BUFG]
```

Simulation Waveforms:

There was no simulation waveform, but we used the Integrated Logic Analyzer to display the outcome.

Figure 1: This figure shows the count up part of the demo. The waveform is counting up 0-9.

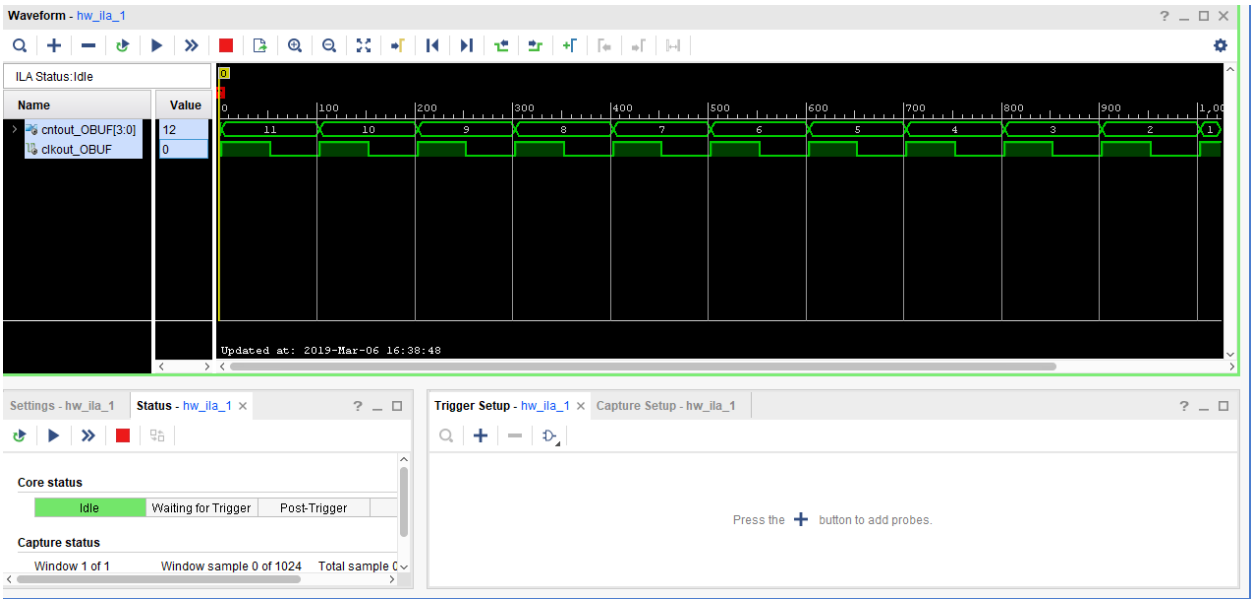


Figure 2: This figure shows the countdown part of the demo. The waveform is counting up 9-0.

Results Discussion

When the user would push the switch to high, the FPGA would count up from 0-9 and it would be displayed on the last 7 segment display. When the switch is low, the counter counts down

from 9-0. This lab was not difficult because the only piece of code not given to us was the exact clock divider module. The only thing we needed to change was the frequency of the clock that needed to be divided. Once that was figured out, all we needed to change was to determine what would be displayed.

Part 2 - Pseudorandom Number Generator and Hamming Code Displays on LEDs
Design Purpose:
The primary 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 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 LED displays will be blank. When SW2 switch is pressed, the LFSR value at that moment will be displayed as 4-bit original message data on the LEDs. When SW3 switch is pressed, the (7, 4) hamming code will be constructed and displayed on the LEDs.

Engineering Data:
The main components that we will utilize in this lab will be a linear feedback shift register, which is composed of d flip flops organized together. The size of the shift register depends on the bit size you want it to output. For instance, if we wanted an 8-bit number, we would need 8 flip flops for the LFSR etc. Moreover, the other main component of this part, will be a module that will calculate the hamming code of the values generated. Finally, using a FSM (finite state machine) we control the arithmetic and process of the design.
As explained before, at every clock cycle, the multiplier is shifted right by one bit and its value is tested. If it is a 0, then the zero value is added to the accumulator, and the result is shifted right by one bit. If the value is a 1, then the multiplicand is added to the accumulator, and the result is shifted right by one bit.

LFSR:
A linear-feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state. The most commonly used linear function of single bits is exclusive-or (XOR). Thus, an LFSR is most often a shift register whose input bit is driven by the XOR of some bits of the overall shift register value. Here is a picture of a basic block diagram of an LSFR.

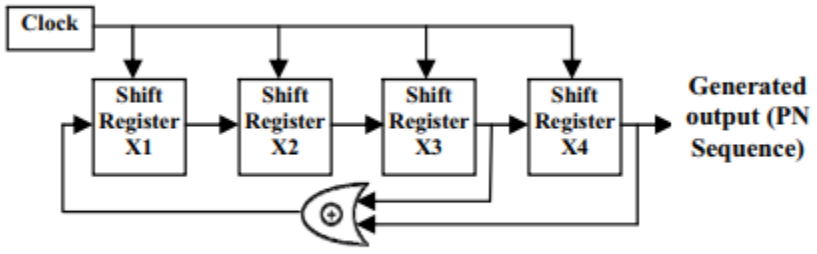


Figure 1. Basic block diagram of LFSR

A maximum-length LFSR produces a sequence unless it contains all zeros, in which case it will never change. The sequence of numbers generated by this method is random. The period of the

sequence is (2n - 1), where n is the number of shift registers used in the design. Here is a picture of the LSFR equation we used for this lab:

Design Specification

Use X^4 + X + 1 linear feedback shift register (LFSR) as pseudorandom generator circuit shown in Figure 3-1. The four D-Flip Flops used in LFSR circuit output 4-bit output values at 1 Hz in the free running mode.

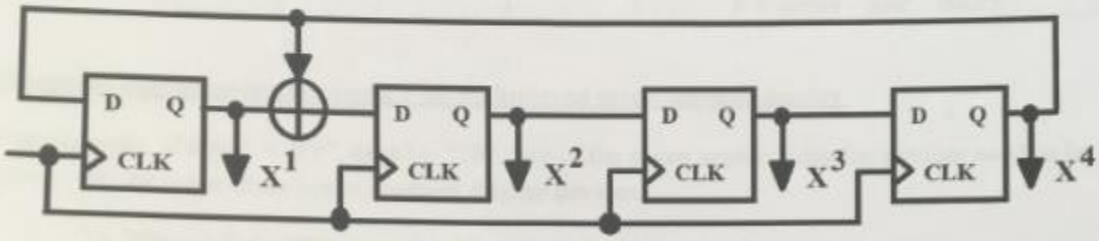


Figure 3-1. X^4 + X + 1 LFSR circuit

Hamming Code:

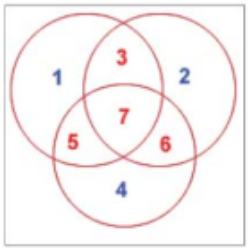
Hamming code is a set of error-correction codes that can be used to detect and correct the errors that can occur when the data is moved or stored from the sender to the receiver. It is technique developed by R.W. Hamming for error correction. The following figures demonstrate how to generate 7, 4 Hamming code.

Consider a message having four data bits (D) which is to be transmitted as a 7-bit codeword by adding three error control bits. This would be called a (7,4) code. The three bits to be added are three EVEN Parity bits (P), where the parity of each is computed on different subsets of the message bits as shown below.

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

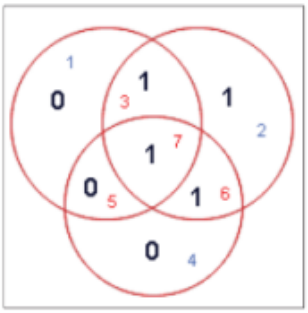
Why Those Bits? - The three parity bits (1,2,4) are related to the data bits (3,5,6,7) as shown at right. In this diagram, each overlapping circle corresponds to one parity bit and defines the four bits contributing to that parity computation.

For example, data bit 3 contributes to parity bits 1 and 2. Each circle (parity bit) encompasses a total of four bits, and each circle must have EVEN parity. Given four data bits, the three parity bits can easily be chosen to ensure this condition. It can be observed that changing any one bit numbered 1..7 uniquely affects the three parity bits. Changing bit 7 affects all three parity bits, while an error in bit 6 affects only parity bits 2 and 4, and an error in a parity bit affects only that bit. The location of any single bit error is determined directly upon checking the three parity circles.

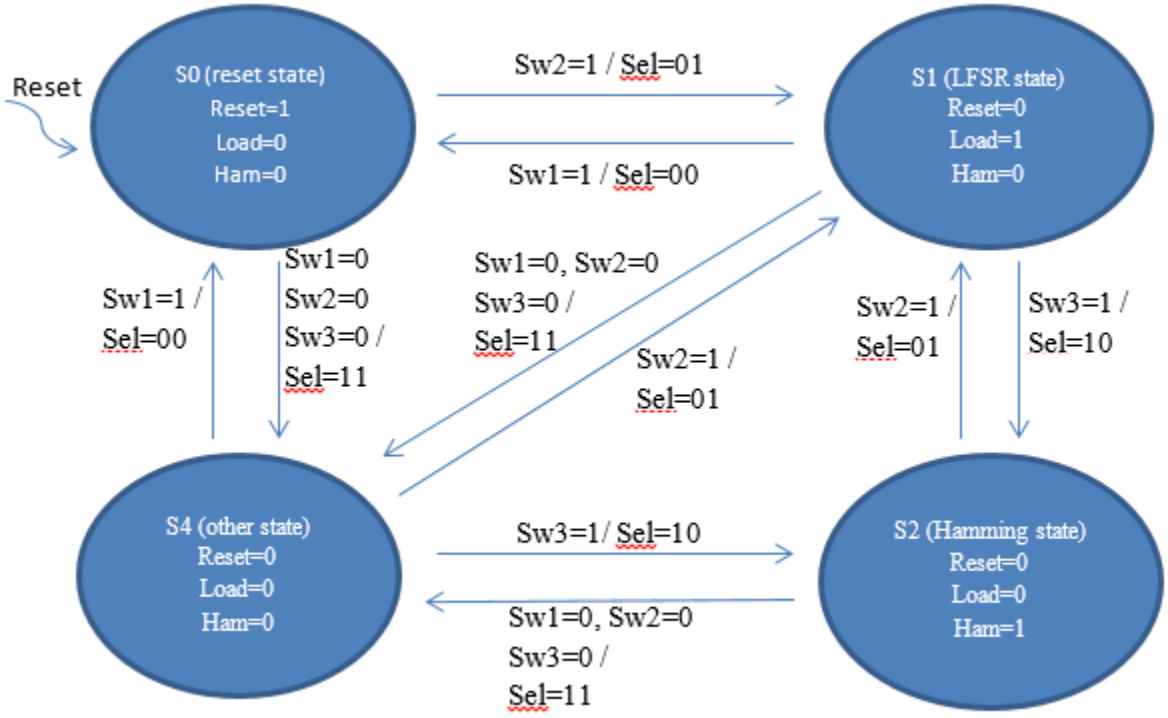


For example, the message 1101 would be sent as 1100110, since:

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)



When these seven bits are entered into the parity circles, it can be confirmed that the choice of these three parity bits ensures that the parity within each circle is EVEN, as shown here.



The final lower level module was the finite state machine to control all of the inputs to the other modules. The finite state machine took the inputs from the fpga and outputted them to the other modules. The state machine had four states: reset state, lfsr state, hamming code state, and other state. The other state was implemented so the lfsr would hold its last value because the load intput would be 0. The fsm was a mixture of Mealy and Moore outputs. The select that went to the multiplexor needed to be a Mealy output because it was dependent on the input and the current state. The other outputs that just controlled the reset and the load for the LFSR register only needed to be Moore outputs because they would only change when the state changed. When designing this state machine, I inititally only had three states. But when I tested on the fpga, the register never held its value. This was because there was never a state where every output from the state machine was zero. The state machine would just stay in the last value until the inputs were changed. This was fixed by adding a state where all inputs were zero, including the load, so the register would never display the new LFSR values. The lfsr did not have a load, so it was always running even if it was not being displayed. The register was the data being sent to the multiplexor, so it would change depending on the state machines inputs. Once the state machine was designed, a top level module was created to connect everything together. The top level consisted of the 100 MHz clock from the fpga, three switches (reset, sw1, sw2) and the output was to seven LEDs. Depending on what state the machine was in, it would display on all seven LEDs, only four, or zero LEDs.

Source Code

Hamming VHDL Code:

```
library ieee;
use ieee.std_logic_1164.all;
entity hamming is
port(
   d: in std_logic_vector(3 downto 0);
   ham_out : out std_logic_vector(6 downto 0)
);
end hamming;
architecture beh of hamming is
signal p: std_logic_vector(2 downto 0);
begin
p(0) <= d(3) xor d(1) xor d(0);
p(1) <= d(3) xor d(2) xor d(0);
p(2) <= d(3) xor d(2) xor d(1);
ham_out <= d(3) & d(2) & d(1) & p(2) & d(0) & p(1) & p(0);
end beh;
```

Finite State Machine VHDL Code:

```
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;
signal lfsr_cap:std_logic_vector(3 downto 0);
signal hamming_cap:std_logic_vector(6 downto 0);
begin
    state_proc:process (clk,sw1,sw2,sw3)
    begin
        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;
```

Clock Div VHDL Code:

```
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;
```

LSFR VHDL Code:

```
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);
BEGIN
   process( clk, rst )
   begin
   if (rst='1') then
      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;
```

TOP VHDL Code:

```
library ieee;
use ieee.std_logic_1164.all;
entity top is
port(
   clk, rst, 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(3 downto 0)
);
```

```
end component;
component hamming
port(
   d: in std_logic_vector(3 downto 0);
   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
   c1: clockDiv port map(clock => clk, clock2 => one_hz_clk);
   c2: 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);
   c3: hamming port map( d => lfsr_out, ham_out => hamming_out);
   c4: lfsr port map(clk => one_hz_clk, rst => rst, Q => lfsr_out);
end beh;
```

User Constraint File

```
set_property -dict {PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports clk]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]

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 R15 IOSTANDARD LVCMOS33} [get_ports rst]

set_property -dict {PACKAGE_PIN H17 IOSTANDARD LVCMOS33} [get_ports
{led_out[0]}]
set_property -dict {PACKAGE_PIN K15 IOSTANDARD LVCMOS33} [get_ports
{led_out[1]}]
set_property -dict {PACKAGE_PIN J13 IOSTANDARD LVCMOS33} [get_ports {led_out[2]}]
set_property -dict {PACKAGE_PIN N14 IOSTANDARD LVCMOS33} [get_ports
{led_out[3]}]
set_property -dict {PACKAGE_PIN R18 IOSTANDARD LVCMOS33} [get_ports
{led_out[4]}]
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports
{led_out[5]}]
set_property -dict {PACKAGE_PIN U17 IOSTANDARD LVCMOS33} [get_ports
{led_out[6]}]
```

```
create_debug_core u_ila_0 ila
set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_EN_STRG_QUAL false [get_debug_cores u_ila_0]
set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
set_property port_width 1 [get_debug_ports u_ila_0/clk]
connect_debug_port u_ila_0/clk [get_nets [list clk_IBUF_BUFG]]
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe0]
set_property port_width 3 [get_debug_ports u_ila_0/probe0]
connect_debug_port u_ila_0/probe0 [get_nets [list {hamming_out[0]} {hamming_out[1]}
{hamming_out[3]}]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe1]
set_property port_width 7 [get_debug_ports u_ila_0/probe1]
connect_debug_port u_ila_0/probe1 [get_nets [list {led_out_OBUF[0]} {led_out_OBUF[1]}
{led_out_OBUF[2]} {led_out_OBUF[3]} {led_out_OBUF[4]} {led_out_OBUF[5]}
{led_out_OBUF[6]}]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe2]
set_property port_width 4 [get_debug_ports u_ila_0/probe2]
connect_debug_port u_ila_0/probe2 [get_nets [list {lfsr_out[0]} {lfsr_out[1]} {lfsr_out[2]}
{lfsr_out[3]}]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe3]
set_property port_width 1 [get_debug_ports u_ila_0/probe3]
connect_debug_port u_ila_0/probe3 [get_nets [list clk_IBUF]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe4]
set_property port_width 1 [get_debug_ports u_ila_0/probe4]
connect_debug_port u_ila_0/probe4 [get_nets [list clock2]]
set_property C_CLK_INPUT_FREQ_HZ 300000000 [get_debug_cores dbg_hub]
set_property C_ENABLE_CLK_DIVIDER false [get_debug_cores dbg_hub]
set_property C_USER_SCAN_CHAIN 1 [get_debug_cores dbg_hub]
connect_debug_port dbg_hub/clk [get_nets clk_IBUF_BUFG]
```

Simulation Waveforms

For this lab there was no simulation since we displayed it on the FPGA board. The figures are below.

Figure 3: This figure represents the original 4-but message, which is “1011” and it is noted as d4, d3, d2, and d1.

Figure 4: This figure is the final (7, 4) hamming code, which is 1010101.

Results Discussion

Although I had some trouble designing the pseudo random generator through hierarchal design; after some careful thinking, I was able to formulate the correct modules to have the code successfully implemented onto the NEXYS 4DDR board. Once it was programmed onto the board I verified that the code worked correctly, displaying the LSFR as well as the Hamming code values.

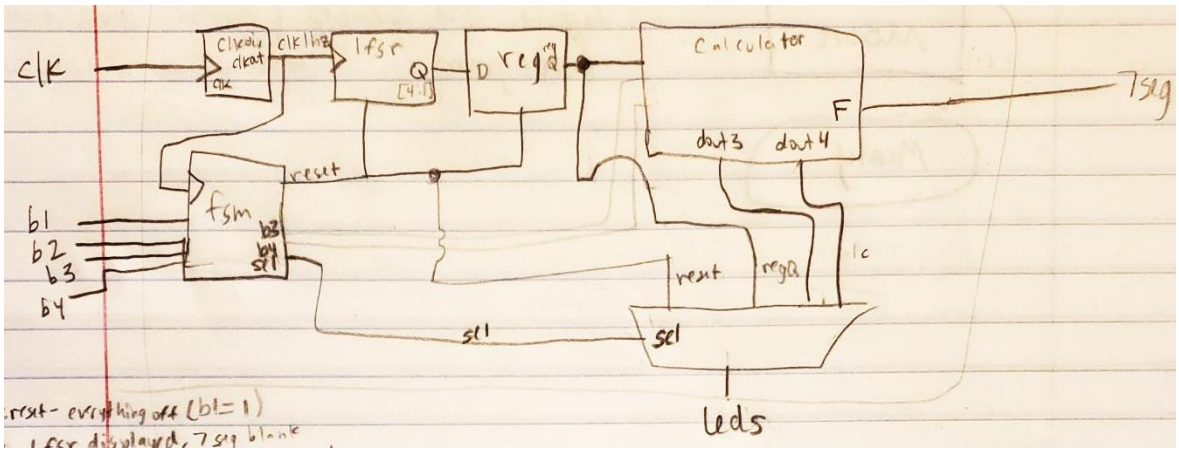
Part 3 – Calculator Design with Multiplexed Seven Segment Displays

Design Purpose

This lab was an introduction to an arithmetic logic unit. This calculator would use one 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. This calculator uses a lot of the modules from the previous parts of lab 3. The new modules included the calculator and the finite state machine. The design of the finite state machine was also somewhat inspired by the FSM used for the part 2.

Engineering Data

This calculator uses a lot of the modules from the previous parts of lab 3. The re-used modules included the clock division, LFSR, register, and the basic design of the multiplexer. The new modules included the calculator and the finite state machine. The design of the finite state machine was also somewhat inspired by the FSM used for the part 2. A schematic design of the calculator can be found with the image below:

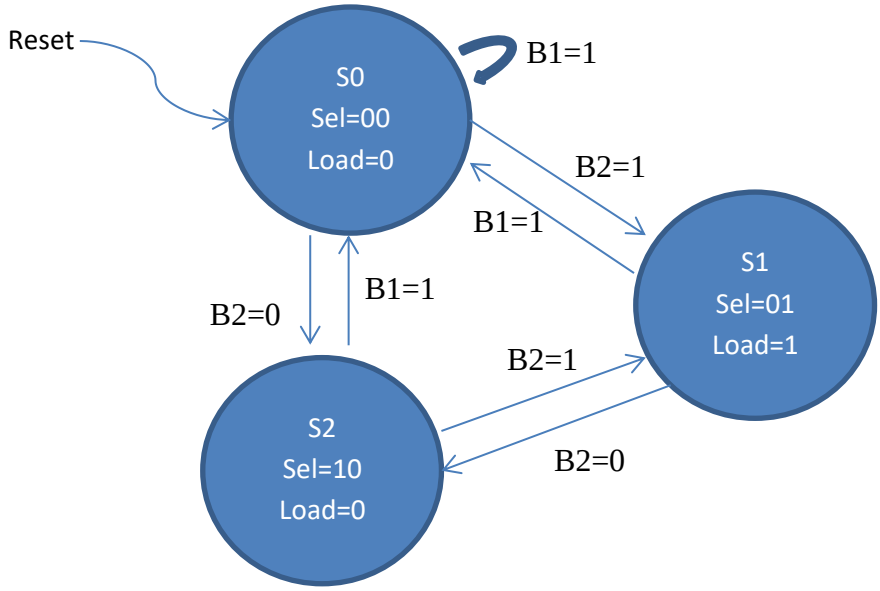


The calculator consisted of four functions, addition, subtraction, comparison, and NAND. The input from the register with the LFSR data needed to be split up into two 2-bit numbers. These numbers would be the numbers used to do the calculations. D3 was Q(2) and Q(1), while D4 was Q(4) and Q(3). The numbers were concatenated together using the and symbol. When creating the comparison d4>d3, I could not use the greater than symbol because it did not mean "greater than" in VHDL. To work around this, I needed to use a when else statement to do the comparison. Eventually, I changed the when else statement to an if statement so it would work within the process block. I also had difficulty when using a NAND gate. I am unsure if there was some other error and I just changed it and fixed the error some other way. Instead of using the keyword "NAND", I created by own NAND gate by using if statements. The addition and subtraction were just done by using the "+" and "-"operators. Here is an image of the inputs and outputs of the calculator results:

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

The next module was the finite state machine. It was designed using three states, the reset state, the LFSR state, and the calculator state. If button1 was high, it was in the reset state, if button 2 was high, it was in the LFSR state where the number displayed on the LEDs would change, and the final state was the calculator state which was reached when button 2 was low. Since this calculator used many of the same modules as the previous part, it was easier to design. The inputs to the state machine were two buttons, and the outputs were to four LEDs. The state diagram drawing is below.



The final module was the top level module which connected the finite state machine to the other modules. The top level module took the inputs from the 100 MHz clock, four switches, and the outputs were four LEDs and three 7-segment displays. There was no extra module to control what was displayed on the 7-segment displays because I thought it would be easier to control from the top module. This module connected all the modules together, and it also controlled the 7-segment displays. The calculator results were displayed on three 7-segment displays because the calculator result was a 3-bit binary number. This added a little bit of complexity compared to just displaying the decimal result on a single 7-segment display.

Source Code

Calculator VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity calculator is
  Port (din3, din4 : in STD_LOGIC_VECTOR (1 downto 0);
        y1, y2, y3, y4 : out STD_LOGIC_VECTOR (2 downto 0) );
end calculator;

architecture beh of calculator is
begin
    process (din3, din4)
    begin
        if (din4 > din3) then
            y1 <= "001";
        else
            y1 <= "000";
        end if;
```

```
        y2 <= ('0' & din4) + ('0' & din3);
        y3 <= ('0' & din4) or ('0' & din3);
        y4 <= ('0' & din4) and ('0' & din3);
    end process;

end beh;
```

Clock Div VHDL Code:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity clkdiv is
port(clk, rst : in STD_LOGIC;
     clock2 : out STD_LOGIC );
end clkdiv;
architecture beh of clkdiv is
signal cnt_div: std_logic_vector(25 downto 0);
signal tmp_clk : STD_LOGIC;
begin
    process(clk)
    begin
        if (rst = '1') then
            cnt_div <= (others => '0');
        elsif (rising_edge (clk)) then
            if (cnt_div = 99999999) then
                cnt_div <= (others => '0');
                tmp_clk <= '1';
            elsif (cnt_div < 49999999) then
                cnt_div <= cnt_div + 1;
                tmp_clk <= '1';
            else
                cnt_div <= cnt_div + 1;
                tmp_clk <= '0';
            end if;
        end if;
    end process;
    clock2 <= tmp_clk;
end beh;
```

Displayer VHDL Code:

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

```
entity displayer is
   Port ( clk: in std_logic;
   din, s: in STD_LOGIC_VECTOR(2 downto 0);
   seg: out std_logic_vector(7 downto 0);
   dig: out std_logic_vector(7 downto 0) );
end displayer;


architecture beh of displayer is
signal count: std_logic_vector(17 downto 0);
signal dd:    std_logic_vector(3 downto 0);
signal an:    std_logic_vector(7 downto 0);
signal dis1, dis2, dis3 : STD_LOGIC_VECTOR(6 downto 0);
begin

  process(clk)
  begin
   if(rising_edge(clk)) then
    count <= count + 1;
    case(count(17 downto 15)) is
      when "000" => dd  <= "0111";
                    an  <= x"FE";
      when "001" => dd  <= "0110";
                    an  <= x"FD";
      when "010" => dd  <= "0101";
                    an  <= x"FB";
      when "011" => dd  <= "0100";
                    an  <= x"F7";
      when "100" => dd  <= "0011";
                    an  <= x"EF";
      when "101" => dd  <= "0010";
                    an  <= x"DF";
      when "110" => dd  <= "0001";
                    an  <= x"AF";
      when "111" => dd  <= "0000";
                    an  <= x"7F";
      when others => dd  <= "0000";
                     an  <= x"7F";
     end case;
   end if;

  end process;

  dig <= an;
```

```
process (dd)
begin
  seg(7) <= '1';
  if ( s = "000") then
     dis1 <= "1111111";
     dis2 <= "1111111";
     dis3 <= "1111111";

  elsif (s = "001") then
     dis1 <= "1111111";
     dis2 <= "1111111";
     if (din(0) = '1') then
        dis3 <= "1111001";
     elsif(din(0) = '0') then
        dis3 <= "1000000";
     end if;
  elsif (s = "010") then
     if( din(2) = '1') then
        dis1 <= "1111001";
     elsif (din(2) = '0') then
        dis1 <= "1000000";
     end if;

     if (din(1) = '1') then
        dis2 <= "1111001";
     elsif (din(1) = '0') then
        dis2 <= "1000000";
     end if;

     if (din(0) = '1') then
        dis3 <= "1111001";
     elsif (din(0) = '0') then
        dis3 <= "1000000";
     end if;

  elsif (s = "011") then
     dis1 <= "1111111";

     if( din(1) = '1') then
        dis2 <= "1111001";
     elsif (din(1) = '0') then
        dis2 <= "1000000";
     end if;

     if( din(0) = '1') then
```

```
            dis3 <= "1111001";
        elsif (din(0) = '0') then
            dis3 <= "1000000";
        end if;
    elsif (s = "100") then
        dis1 <= "1111111";

        if( din(1) = '1') then
            dis2 <= "1111001";
        elsif (din(1) = '0') then
            dis2 <= "1000000";
        end if;

        if( din(0) = '1') then
            dis3 <= "1111001";
        elsif (din(0) = '0') then
            dis3 <= "1000000";
        end if;

    end if;


    case(dd) is
        When x"0" =>
            seg(6 downto 0) <= "1111111"; --dis1;
        When x"1" =>
            seg(6 downto 0) <= "1111111"; --dis2
        When x"2" =>
            seg(6 downto 0) <= "1111111";  --dis3

        --when x"0" => seg(6 downto 0) <= "0001110"; --to display F
        --when x"1" => seg(6 downto 0) <= "0001100"; --to display P
        --when x"2" => seg(6 downto 0) <= "0000010"; --to display G
        when x"3" => seg(6 downto 0) <= "1111111"; --to display A
        when x"4" => seg(6 downto 0) <= "1111111"; --to display -

        when x"5" => seg(6 downto 0) <= dis1; --to display F
        when x"6" => seg(6 downto 0) <= dis2; --to display u
        when x"7" => seg(6 downto 0) <= dis3; --to display n
        when others => seg(6 downto 0) <= "1111111"; --blank
  end case;
  end process;


end beh;
```

Finite State Machine VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity fsm is
  Port (sw1 , sw2, sw3, sw4, clk: in STD_LOGIC;
        RE, clr : out STD_LOGIC;
        muxS : out STD_LOGIC_VECTOR(2 downto 0) );
end fsm;

architecture beh of fsm is
type state_type is(S1, S2, S3, S4, S5, S6, S7);
signal cs, ns : state_type;

begin
    process(sw1, clk)
    begin
        if (sw1 = '1') then
            cs <= S1;
        elsif (rising_edge(clk)) then
            cs <= ns;
        end if;
    end process;

    process(cs, sw2, sw3, sw4)
    begin
        case(cs) is
            When S1 =>
                ns <= S2; clr <= '1'; RE <= '0'; muxS <= "000";

            When S2 =>
                clr <= '0'; RE <= '0'; muxS <= "000";
                if(sw2 = '1' and sw3 = '0' and sw4 = '0') then
                    ns <= S3;
                else
                    ns <= S2;
                end if;

            When S3 =>
                clr <= '0'; RE <= '1'; muxS <= "000";
                if (sw2 = '0' and sw3 = '0' and sw4 = '0') then
                    ns <= S4;
                else
                    ns <= S3;
                end if;

            When S4 =>
```

```
    clr <= '0'; RE <= '0'; muxS <= "001";
    if(sw2 = '0' and sw3 = '0' and sw4 = '0') then
       ns <= S4;
    elsif (sw2 ='0' and sw3 = '0' and sw4 = '1') then
       ns <= S5;
    elsif (sw2 ='0' and sw3 = '1' and sw4 = '0') then
       ns <= S6;
    elsif (sw2 ='0' and sw3 = '1' and sw4 = '1') then
       ns <= S7;
    elsif (sw2 = '1') then
       ns <= S3;
    else
       ns <= S4;
    end if;

When S5 =>
       clr <= '0'; RE <= '0'; muxS <= "010";
       if(sw2 = '1') then
          ns <= S3;
       elsif (sw2 ='0' and sw3 = '1' and sw4 = '0') then
          ns <= S6;
       elsif (sw2 ='0' and sw3 = '1' and sw4 = '1') then
          ns <= S7;
       elsif (sw2 ='0' and sw3 = '0' and sw4 = '0') then
          ns <= S4;
       elsif (sw2 ='0' and sw3 = '0' and sw4 = '1') then
          ns <= S5;
       else
          ns <= S5;
       end if;

When S6 =>
       clr <= '0'; RE <= '0'; muxS <= "011";
       if(sw2 = '1') then
          ns <= S3;
       elsif (sw2 ='0' and sw3 = '1' and sw4 = '0') then
          ns <= S6;
       elsif (sw2 ='0' and sw3 = '1' and sw4 = '1') then
          ns <= S7;
       elsif (sw2 ='0' and sw3 = '0' and sw4 = '0') then
          ns <= S4;
       elsif (sw2 ='0' and sw3 = '0' and sw4 = '1') then
          ns <= S5;
       else
          ns <= S6;
       end if;
```

```
        When S7 =>
            clr <= '0'; RE <= '0'; muxS <= "100";
            if(sw2 = '1') then
               ns <= S3;
            elsif (sw2 ='0' and sw3 = '1' and sw4 = '0') then
               ns <= S6;
            elsif (sw2 ='0' and sw3 = '1' and sw4 = '1') then
               ns <= S7;
            elsif (sw2 ='0' and sw3 = '0' and sw4 = '0') then
               ns <= S4;
            elsif (sw2 ='0' and sw3 = '0' and sw4 = '1') then
               ns <= S5;
            else
               ns <= S7;
            end if;
        When others =>
           ns <= S1; clr <= '1'; RE <= '0'; muxS <= "000";
     end case;
  end process;
end beh;
```

LSFR VHDL Code:

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

ENTITY lfsr IS
PORT
(
rst, clk, RE: IN STD_LOGIC;
Q1, Q2 : OUT STD_LOGIC_VECTOR(4 downto 1)
);
END lfsr;

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

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

```
end process;
Q1 <= W;
Q2 <= W;
END beh;
```

MUX VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux is
  Port (d0, d1, d2, d3, s : in STD_LOGIC_VECTOR(2 downto 0);
        y : out STD_LOGIC_VECTOR(2 downto 0) );
end mux;

architecture beh of mux is

begin
    process( s, d0, d1, d2, d3)
    begin
        if( s = "000") then
            y <= "000";
        elsif( s = "001") then
            y <= d0;
        elsif( s = "010") then
            y <= d1;
        elsif( s = "011") then
            y <= d2;
        elsif(s = "100") then
            y <= d3;
        end if;
    end process;

end beh;
```

TOP VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity top is
  Port (sw1, sw2, sw3, sw4, clk : in STD_LOGIC;
        numOut: out STD_LOGIC_VECTOR(3 downto 0);
        seg, dig : out STD_LOGIC_VECTOR(7 downto 0) );
end top;

architecture beh of top is
signal w1, w2, w4 : STD_LOGIC;
signal w3,w8,w9,w10,w11,w12 : STD_LOGIC_VECTOR(2 downto 0);
```

```
signal w7: STD_LOGIC_VECTOR(3 downto 0);
component lfsr
PORT
(rst, clk, RE: IN STD_LOGIC;
Q1, Q2 : OUT STD_LOGIC_VECTOR(4 downto 1));
end component;

component calculator
  Port (din3, din4 : in STD_LOGIC_VECTOR (1 downto 0);
        y1, y2, y3, y4 : out STD_LOGIC_VECTOR (2 downto 0) );
end component;

component clkdiv
port(clk, rst : in STD_LOGIC;
     clock2 : out STD_LOGIC );
end component;

component fsm
  Port (sw1 , sw2, sw3, sw4, clk: in STD_LOGIC;
        RE, clr : out STD_LOGIC;
        muxS : out STD_LOGIC_VECTOR(2 downto 0) );
end component;

component displayer
   Port ( clk: in std_logic;
   din, s: in STD_LOGIC_VECTOR(2 downto 0);
   seg: out std_logic_vector(7 downto 0);
   dig: out std_logic_vector(7 downto 0) );
end component;

component mux
  Port (d0, d1, d2, d3, s : in STD_LOGIC_VECTOR(2 downto 0);
        y : out STD_LOGIC_VECTOR(2 downto 0) );
end component;
begin
    U1: fsm port map (sw1 => sw1, sw2 => sw2, sw3 => sw3, sw4 => sw4, clk => clk, RE =>
w1, clr => w2, muxS => w3);
    U2: lfsr port map (rst => w2, RE => w1, clk => w4, Q1 => w7, Q2 => numOut);
    U3: calculator port map (din4 => w7(3 downto 2), din3 => w7(1 downto 0),y1 => w8, y2 =>
w9, y3 => w10,y4 => w11);
    U4: clkdiv port map (clk => clk, rst => w2, clock2 => w4);
    U5: displayer port map (clk => clk, din => w12, s => w3, seg => seg, dig => dig);
    U6: mux port map (d0 => w8, d1 => w9, d2 => w10,  d3 => w11, s => w3, y => w12);
end beh;
```

User Constraint File

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
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 { sw4 }];
#IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { sw3 }];
#IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { sw2 }];
#IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { sw1 }];
#IO_L13N_T2_MRCC_14 Sch=sw[3]
#set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { SW[4]
}]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
#set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { SW[5]
}]; #IO_L7N_T1_D10_14 Sch=sw[5]
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { SW[6]
}]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
#set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7]
}]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8]
}]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9]
}]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports {
SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports {
SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { SW[12]
}]; #IO_L24P_T3_35 Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports {
SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports {
SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports {
SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]


## LEDs
```

```
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports {
numOut[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports {
numOut[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports {
numOut[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports {
numOut[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
#set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports {
LED[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
#set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {
LED[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
#set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports {
LED[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
#set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {
LED[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {
LED[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports {
LED[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
#set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports {
LED[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports {
LED[11] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
#set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {
LED[12] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
#set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports {
LED[13] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
#set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports {
LED[14] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
#set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports {
LED[15] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

##7 segment display

set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { seg[0]
}]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { seg[1]
}]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { seg[2]
}]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { seg[3]
}]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { seg[4]
}]; #IO_L13P_T2_MRCC_14 Sch=ce
```

```
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { seg[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { seg[6] }]; #IO_L4P_T0_D04_14 Sch=cg

set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { seg[7] }]; #IO_L19N_T3_A21_VREF_15 Sch=dp

set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { dig[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { dig[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { dig[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { dig[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { dig[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { dig[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { dig[6] }]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { dig[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
```

Simulation Waveforms

There was no simulation waveforms but there are FPGA outputs.

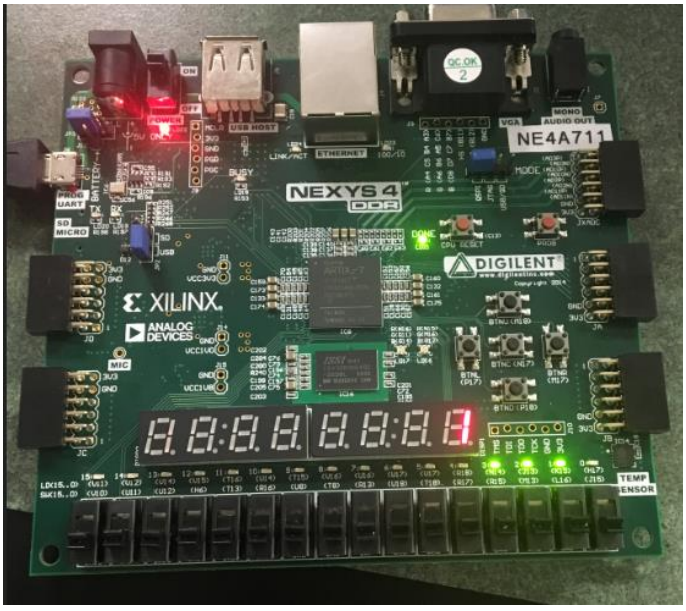


Figure 5: This figure displays b3b4 = “00”, and should then display blank, blank, 1.

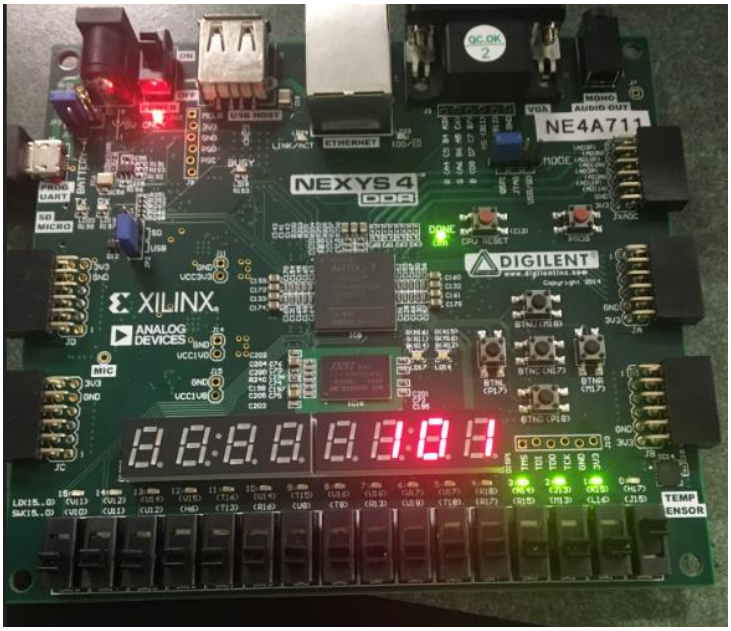


Figure 6: This figure displays b3b4 = “01”, and should then display 1, 0, 1.

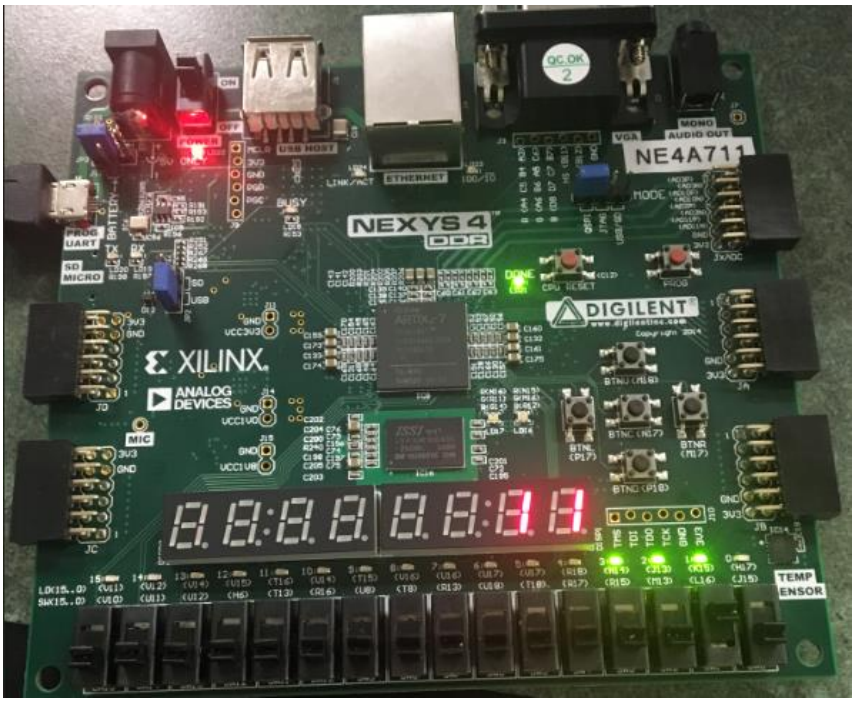


Figure 7: This figure displays b3b4 = “10”, and should then display blank, 1, 1.

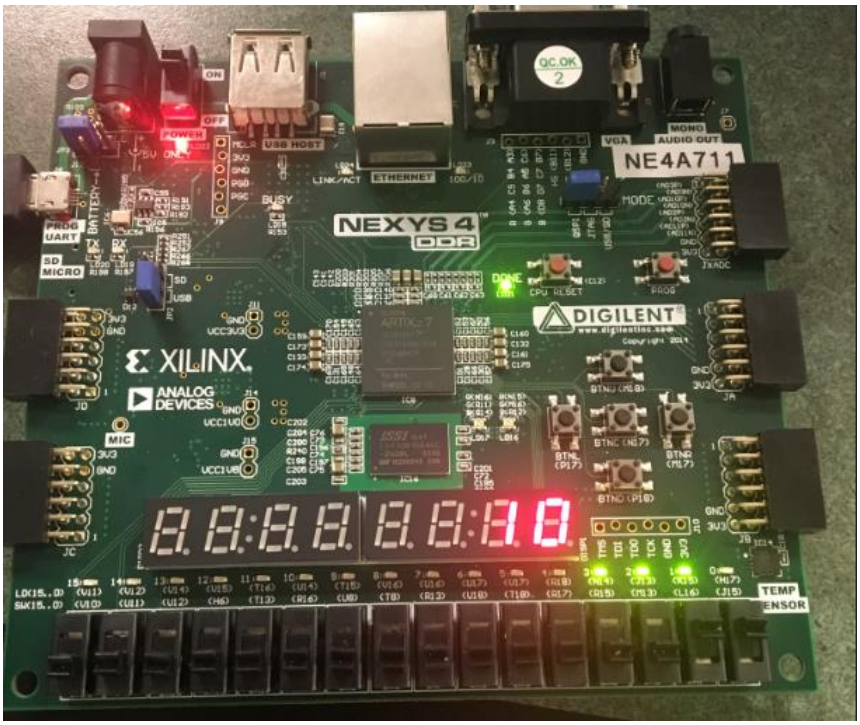


Figure 7: This figure displays b3b4 = “11”, and should then display blank, 1, 0.

Results Discussion

This lab was like the previous lab, so it was not so bad to design. It was difficult and frustrating to try and get the calculator to work when I did not know how to properly do the comparisons, but I was eventually able to figure it out. The calculator will perform the tasks it is supposed to do, the finite state machine goes into the correct state depending on the input, and the top-level module worked as expected. The final results were displayed on the FPGA, and they were also simulated. The calculator was able to add, subtract, compare, and NAND two numbers and display them in binary from on three different 7-segment displays.

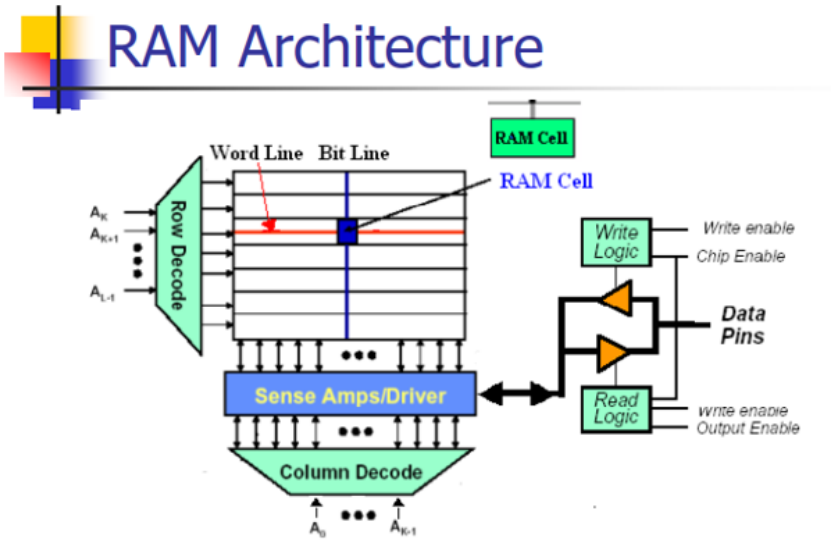
Part 4 - RAM Design and Logic Analyzer

Design Purpose

The primary objective of this part will be to use VHDL in order to design RAM. In our RAM design, we will need to write binary data "1010" into 16 address locations of the RAM. After complete writing, we need to read out that data out of RAM. In addition, we needed to demo the address and data bus results captured by Xilinx Integrated Logic Analyzer (ILA).

Engineering Data

Random access memory, or RAM, is one of the most important components of not only desktop PCs, but laptops, tablets, smartphones, and gaming consoles. Without it, doing just about anything on any system would be much, much slower. Static RAM provides faster access to data and is more expensive than DRAM. SRAM is used for a computer's cache memory and as part of the random-access memory digital-to-analog converter on a video card. The following picture shows the architecture of SRAM.



The RAM design was given to us and all we needed to change was the size of the data to be the 4-bit binary number "1010." The code given to us had an 8-bit number, so the size of the data needed to be changed to accommodate a smaller number. This needed to be changed in the top level file, the finite state machine, and the SRAM file. Once that was done, we needed to set up the debugger to make sure the clock domain for the data was correct. Once that was done, we created a constraint file to download the data to the FPGA. My constraint file consisted of the clock, a switch to reset it, and the LEDs would display the data. I believe to do this part, we only needed the clock, but I was unsure at the time, so I included the other inputs and outputs just in case they were necessary. Once the FPGA was programmed, the simulation showed my data in the addresses.

Source Code

SRAM VHDL Code:

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

entity sram is
port (
      address :in    std_logic_vector ( 3 downto 0);
      din     :in    std_logic_vector ( 3 downto 0);
      dout    :out   std_logic_vector ( 3 downto 0);
      cs      :in    std_logic;
      we      :in    std_logic;
      oe      :in    std_logic
   );
end sram;

architecture beh_sram of sram is

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

```
begin


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


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

     end if;
     end process;

  end beh_sram;
```

TOP_SRAM VHDL Test Bench Code:

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

entity top_sram is
 Port ( clk, reset: in std_logic;
       data:  inout std_logic_vector(3 downto 0)
 );
end top_sram;

architecture Behavioral of top_sram is

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

attribute mark_debug: string;
attribute keep: string ;
attribute mark_debug of address: signal is "true";
```

```
attribute mark_debug of wdata: signal is "true";

begin

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


g1:  entity sram ( beh_sram )
    port map ( address => address, din => data, dout => data,
            cs => cs, we => we,  oe => oe );

g2:  entity sram_fsm ( fsm_beh )
    port map ( clk=>clk, reset=>reset,
       address=> address,
             cs => cs, we => we, oe => oe );


end Behavioral;
```

TOP_FSM VHDL Code:

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

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

architecture fsm_beh of sram_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";

     when s2 =>
                cnt   <=   cnt + 1;
                if (cnt  < 15) then
                  state <= s2;
                else
                  state <= s3;
                end if;

     when s3 =>   state <= s4;
       cnt  <= "0000";

        when s4 =>
                cnt  <= cnt + 1;
                state <= s4;


  when others =>
                cnt     <= "0000";
                state   <= s1;

    end case;
  end if;
end process;

process(state)
begin
        case state is
                when idle => we <= '0'; oe <= '0';
     when s1 => we <= '1'; oe <= '0';
     when s2 => we <= '1'; oe <= '0';
     when s3 => we <= '0'; oe <= '1';
     when s4 => we <= '0'; oe <= '1';
          when others => we <= '0'; oe <= '0';
        end case;
end process;
```

```
end fsm_beh;
```

TOP Test Bench VHDL Code:

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

entity tb_top is
end tb_top ;
architecture Behavioral of tb_top is
signal clk,reset: std_logic;
signal data: std_logic_vector(3 downto 0);
component top_sram
port(
   clk,reset: in std_logic;
   data: inout std_logic_vector(3 downto 0)
);
end component;
constant clk_period: time:=10ns;
begin
uut: top_sram port map(clk,reset,data);
   clk_proc: process
   begin
   clk<='0';
   wait for clk_period/2;
   clk<='1';
   wait for clk_period/2;
   end process;

   stim_proc: process
   begin
   reset<='1';
   wait for 50ns;
   reset<='0';
   wait for 50ns;
   wait;
    end process;
end Behavioral;
```

User Constraint File

```
## This file is a general .xdc for the Nexys4 DDR Rev. C
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names
in the project
```

```
## Clock signal
set_property -dict {PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports clk]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]


##Switches

set_property -dict {PACKAGE_PIN J15 IOSTANDARD LVCMOS33} [get_ports reset]
##set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { start }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict {PACKAGE_PIN M13 IOSTANDARD LVCMOS33} [get_ports {data[0]}]
set_property -dict {PACKAGE_PIN R15 IOSTANDARD LVCMOS33} [get_ports {data[1]}]
set_property -dict {PACKAGE_PIN R17 IOSTANDARD LVCMOS33} [get_ports {data[2]}]
set_property -dict {PACKAGE_PIN T18 IOSTANDARD LVCMOS33} [get_ports {data[3]}]
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
#set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
##set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { data[0] }]; #IO_L24P_T3_35 Sch=sw[12]
##set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { data[1] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
##set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { data[2] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
##set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { data[3] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]


## LEDs

set_property -dict {PACKAGE_PIN H17 IOSTANDARD LVCMOS33} [get_ports {data[0]}]
set_property -dict {PACKAGE_PIN K15 IOSTANDARD LVCMOS33} [get_ports {data[1]}]
set_property -dict {PACKAGE_PIN J13 IOSTANDARD LVCMOS33} [get_ports {data[2]}]
set_property -dict {PACKAGE_PIN N14 IOSTANDARD LVCMOS33} [get_ports {data[3]}]
##set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { dout[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
##set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { dout[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
```

```
##set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports {
dout[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
##set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {
dout[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {
LED[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports {
LED[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
#set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports {
LED[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports {
LED[11] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
#set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {
LED[12] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
#set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports {
LED[13] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
#set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports {
LED[14] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
#set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports {
LED[15] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

#set_property -dict { PACKAGE_PIN R12   IOSTANDARD LVCMOS33 } [get_ports {
LED16_B }]; #IO_L5P_T0_D06_14 Sch=led16_b
#set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS33 } [get_ports {
LED16_G }]; #IO_L10P_T1_D14_14 Sch=led16_g
#set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports {
LED16_R }]; #IO_L11P_T1_SRCC_14 Sch=led16_r
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports {
LED17_B }]; #IO_L15N_T2_DQS_ADV_B_15 Sch=led17_b
#set_property -dict { PACKAGE_PIN R11   IOSTANDARD LVCMOS33 } [get_ports {
LED17_G }]; #IO_0_14 Sch=led17_g
#set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports {
LED17_R }]; #IO_L11N_T1_SRCC_14 Sch=led17_r


create_debug_core u_ila_0 ila
set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_EN_STRG_QUAL false [get_debug_cores u_ila_0]
set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
set_property port_width 1 [get_debug_ports u_ila_0/clk]
connect_debug_port u_ila_0/clk [get_nets [list clk_IBUF_BUFG]]
```

```
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe0]
set_property port_width 4 [get_debug_ports u_ila_0/probe0]
connect_debug_port u_ila_0/probe0 [get_nets [list {wdata[0]} {wdata[1]} {wdata[2]}
{wdata[3]}]]
create_debug_port u_ila_0 probe
set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe1]
set_property port_width 4 [get_debug_ports u_ila_0/probe1]
connect_debug_port u_ila_0/probe1 [get_nets [list {address[0]} {address[1]} {address[2]}
{address[3]}]]
set_property C_CLK_INPUT_FREQ_HZ 300000000 [get_debug_cores dbg_hub]
set_property C_ENABLE_CLK_DIVIDER false [get_debug_cores dbg_hub]
set_property C_USER_SCAN_CHAIN 1 [get_debug_cores dbg_hub]
connect_debug_port dbg_hub/clk [get_nets clk_IBUF_BUFG]
```

Simulation Waveforms

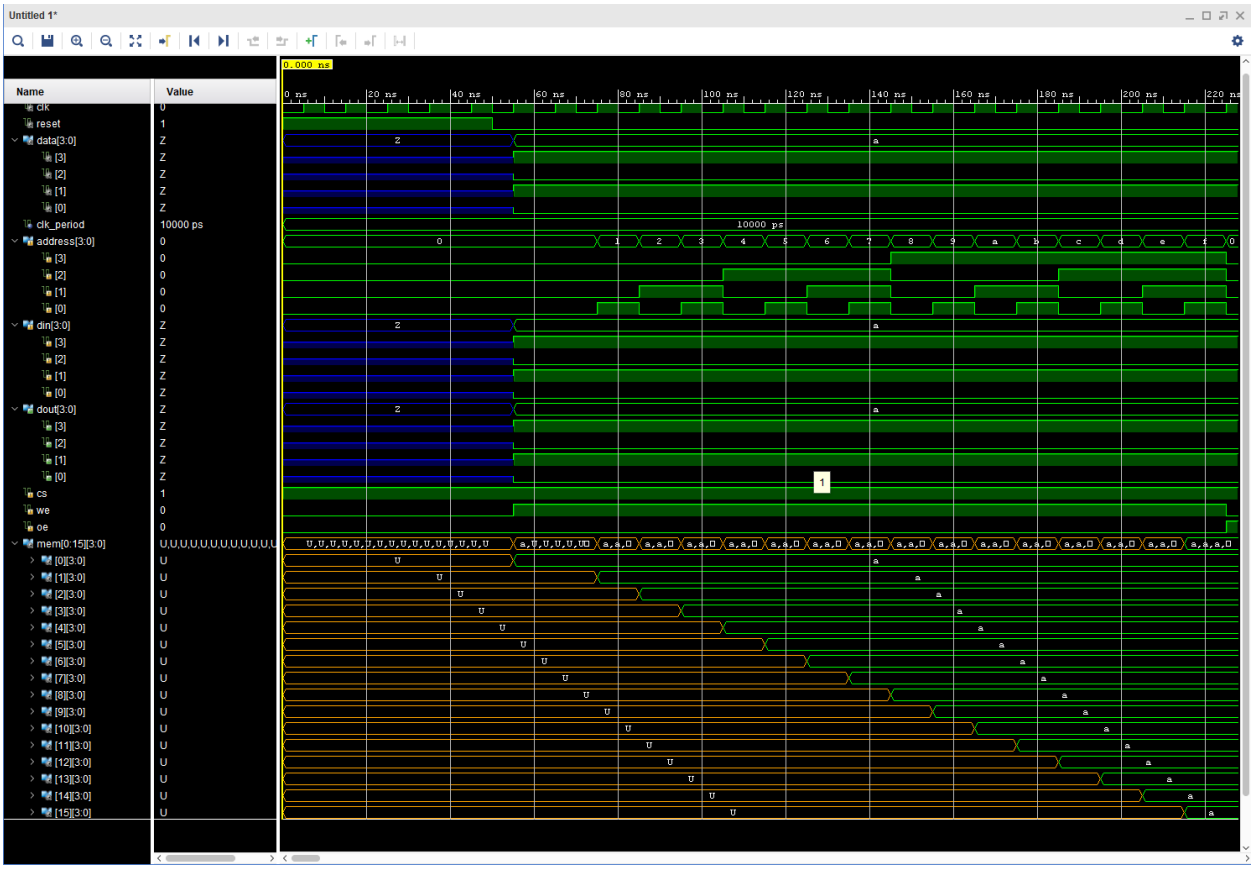


Figure 9: This figure shows the result of the Logic Analyzer after the bitstream was generated. As you can see the binary data “1010” was stored in 16 address lines.

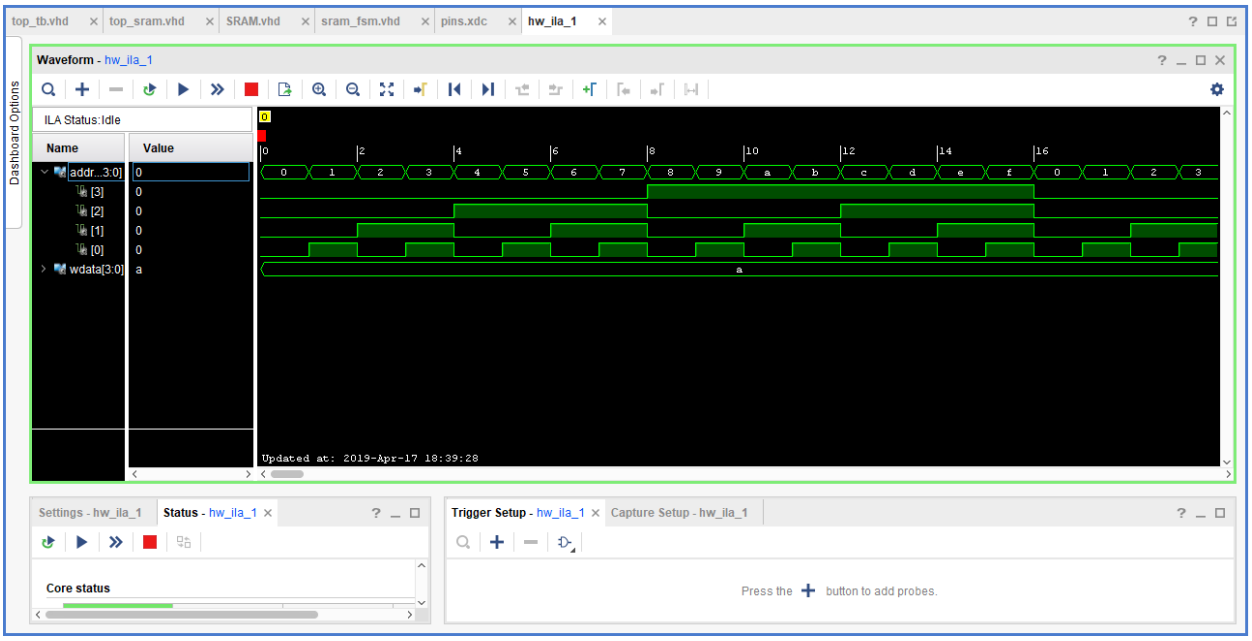


Figure 10: This figure is the simulation of the SRAM. It counts up to 15.

Results Discussion

I was able to effective write the 4 bit binary data into the 16 address lines of SRAM. Once written I went ahead and synthesized, Implemented and generated the bitstream in order to view the result of data. Clearly from the simulation waveform, one can see that 1010 was written into memory.

Conclusion

In this lab, I was able to learn how to implement various combinational and sequential circuits with finite state machines using VHDL. I was able to gain experience utilizing the concepts of hierarchal design models to structure my code in such a way that followed these specific models. Also, this lab included much more complex circuit designs such as a Linear Feedback Shift Register and Hamming code generation. Furthermore, in Part 2 of the lab, I was able to get a deeper understanding of how shift registers function and how to generate hamming code when need, as well lower the frequency of the clock by using clock division. In the final part, I learnt the feature of SRAM as well as how to write data into memory location. Once the data was written into memory; in order, to read out the data, I had to implement the code and generate a bitstream; and finally use Vivado's built in Logic Analyzer. Overall, I was able to complete most of the tasks needed by this lab.