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 3](#_30j0zll)

[Part 1 – 1 MHz BCD Up-Down Counter 3](#_1fob9te)

[Design Purpose 3](#_3znysh7)

[Engineering Data 3](#_2et92p0)

[Source Code 4](#_tyjcwt)

[User Constraint File 5](#_3dy6vkm)

[Simulation Waveforms: 8](#_1t3h5sf)

[Results Discussion 8](#_4d34og8)

[Part 2 - Pseudorandom Number Generator and Hamming Code Displays on LEDs 9](#_2s8eyo1)

[Design Purpose: 9](#_17dp8vu)

[Engineering Data: 9](#_3rdcrjn)

[Source Code 12](#_26in1rg)

[User Constraint File 16](#_lnxbz9)

[Results Discussion 18](#_35nkun2)

[Part 3 – Calculator Design with Multiplexed Seven Segment Displays 18](#_1ksv4uv)

[Design Purpose 18](#_44sinio)

[Engineering Data 18](#_2jxsxqh)

[Source Code 20](#_z337ya)

[User Constraint File 30](#_3j2qqm3)

[Simulation Waveforms 32](#_1y810tw)

[Results Discussion 34](#_4i7ojhp)

[Part 4 - RAM Design and Logic Analyzer 34](#_2xcytpi)

[Design Purpose 34](#_1ci93xb)

[Engineering Data 34](#_3whwml4)

[Source Code 35](#_2bn6wsx)

[User Constraint File 39](#_qsh70q)

[Simulation Waveforms 42](#_3as4poj)

[Results Discussion 43](#_1pxezwc)

[Conclusion 43](#_49x2ik5)

## 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

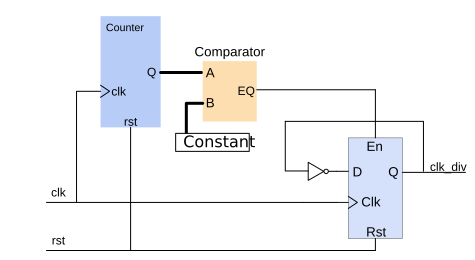
## Design Purpose

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.

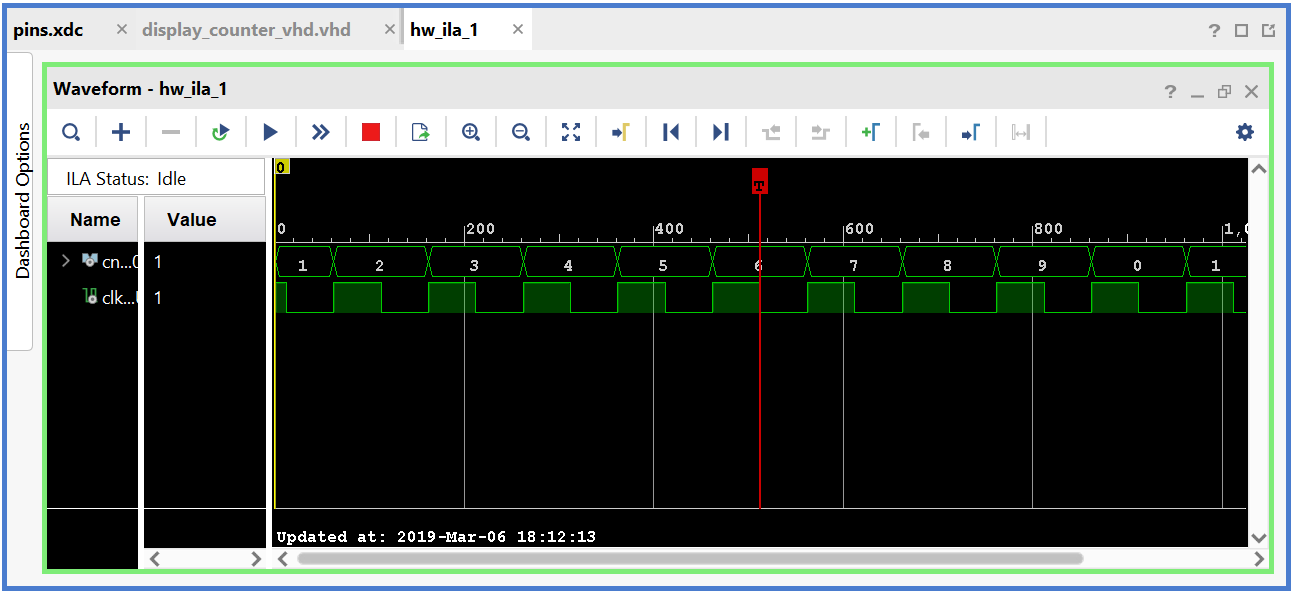


Figure 1: This figure shows the count up part of the demo. The waveform is counting up 0-9.

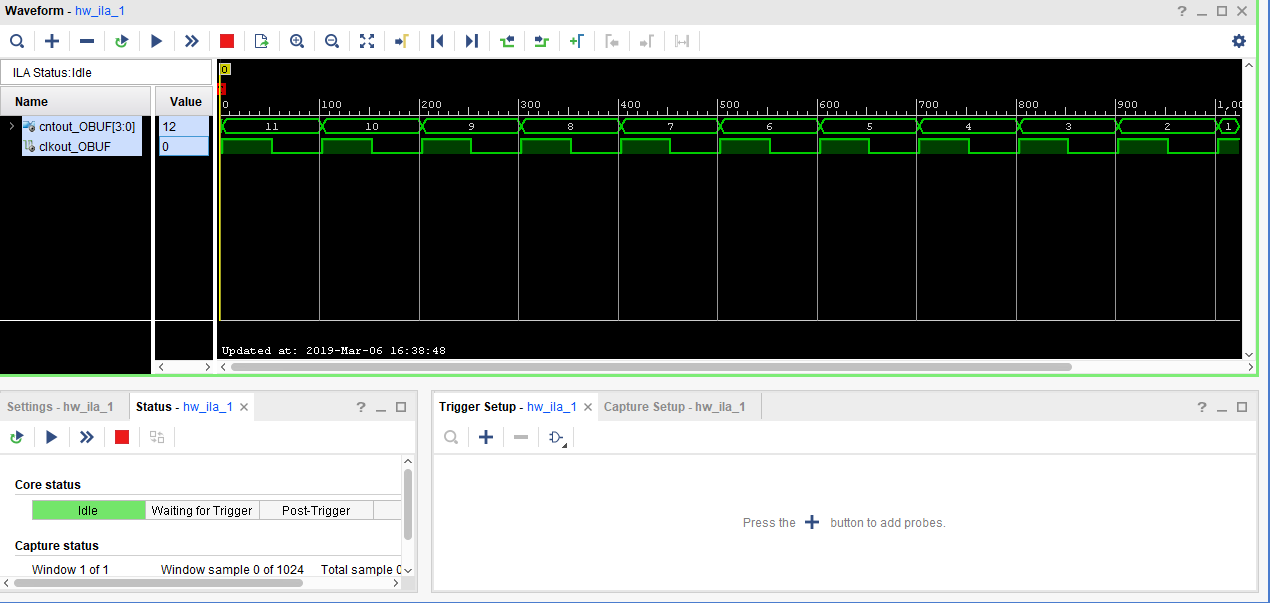


Figure 2: This figure shows the countdown part of the demo. The waveform is counting up 9-0.

## 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.

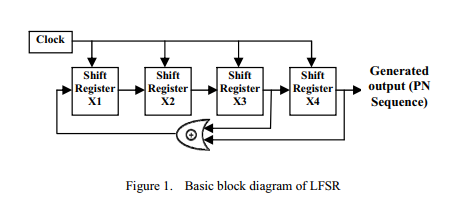
## Engineering Data:

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.

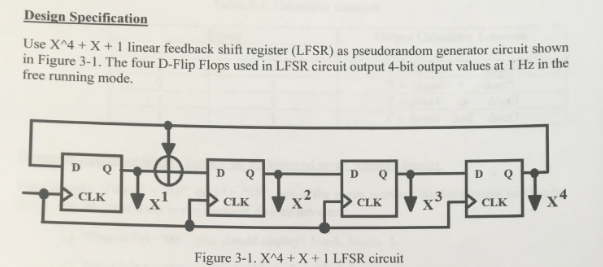
As explained before, at every clock cycle, the multiplier is shifted right by one bit and its value is tested. If it is a 0, then the zero value is added to the accumulator, and the result is shifted right by one bit. If the value is a 1, then the multiplicand is added to the accumulator, and the result 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.

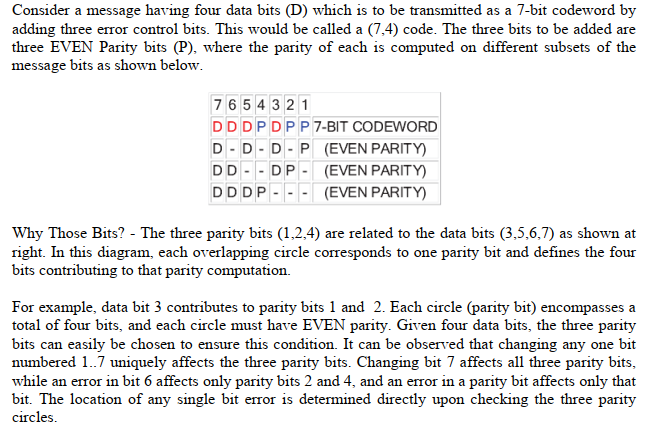


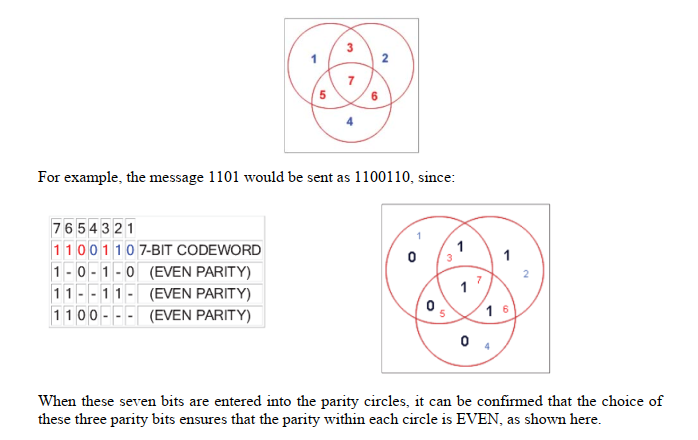
A maximum-length LFSR produces a sequence unless it contains all zeros, in which case it will never change. The sequence of numbers generated by this method is random. The period of the sequence is (2n - 1), where n is the number of shift registers used in the design. Here is a picture of the LSFR equation we used for this lab:

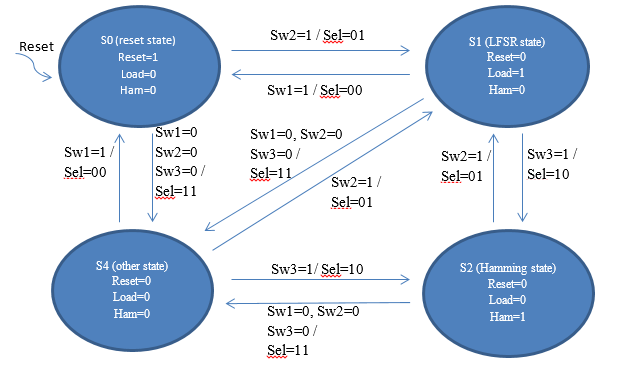


Hamming Code:

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.







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.

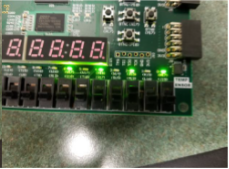


Figure 4: This figure is the final (7, 4) hamming code, which is 1010101.

## 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.

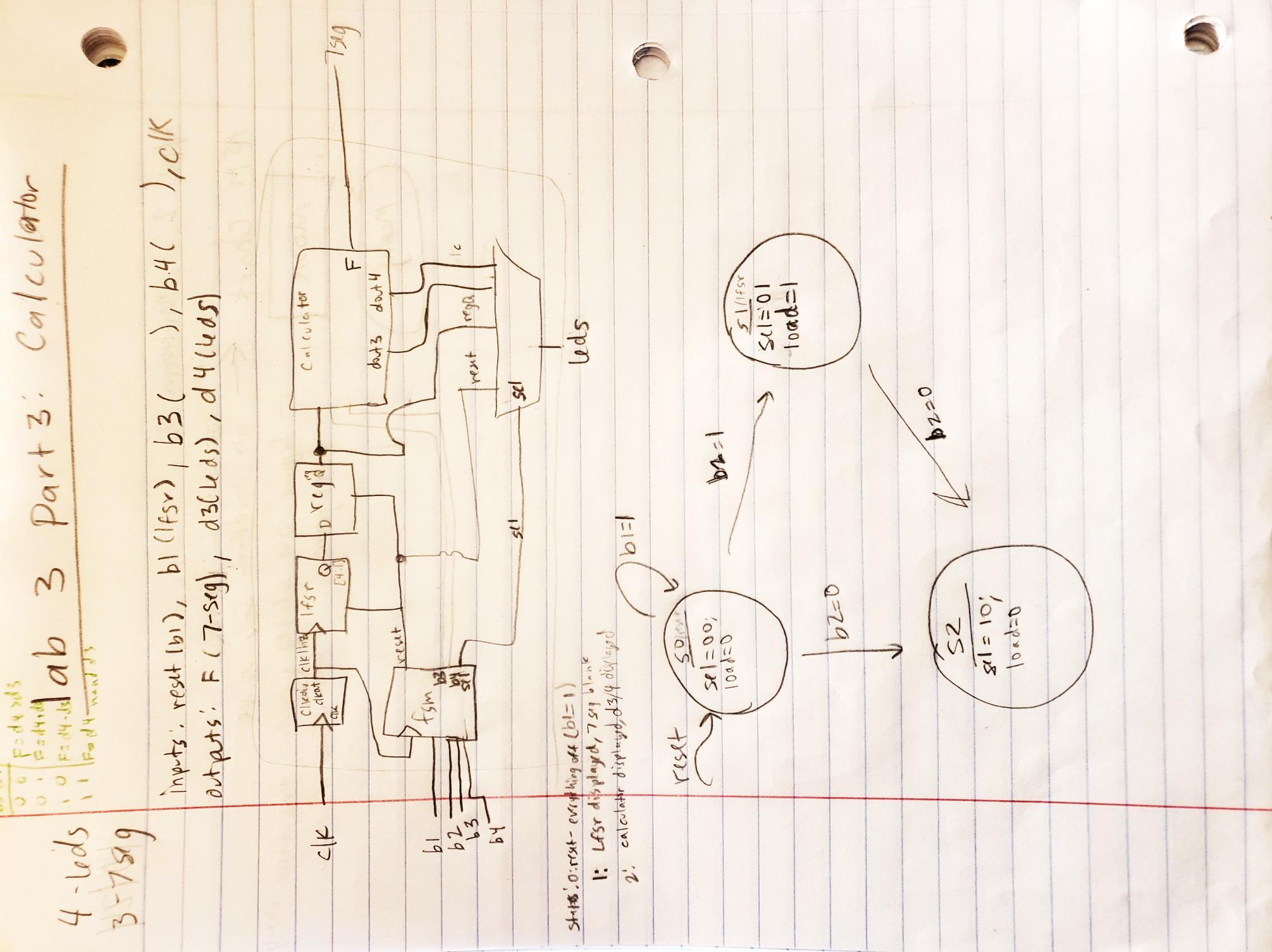
## Part 3 – Calculator Design with Multiplexed Seven Segment Displays

## Design Purpose

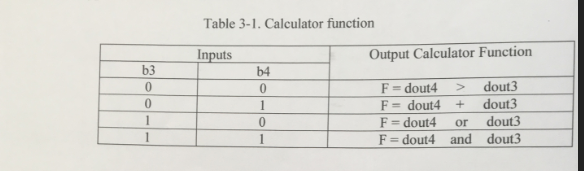
This lab was an introduction to an arithmetic logic unit. This calculator would use one 4-bit pseudo-randomly generated number from the LFSR, split it into two 2-bit numbers, and perform a calculation determined by the user’s input. The calculator performed four functions: add, subtract, compare, and 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:



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.

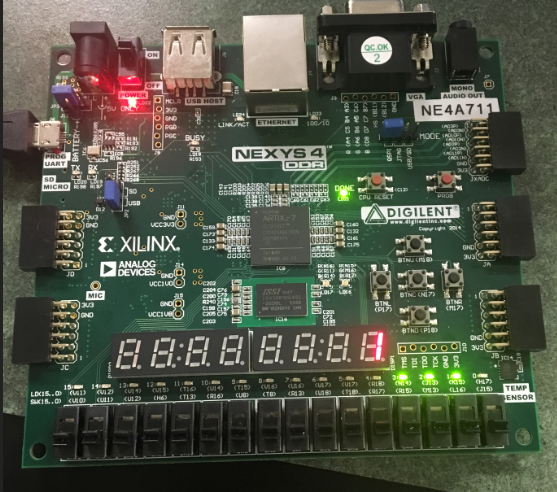


Figure 5: This figure displays b3b4 = “00”, and should then display blank, blank, 1.

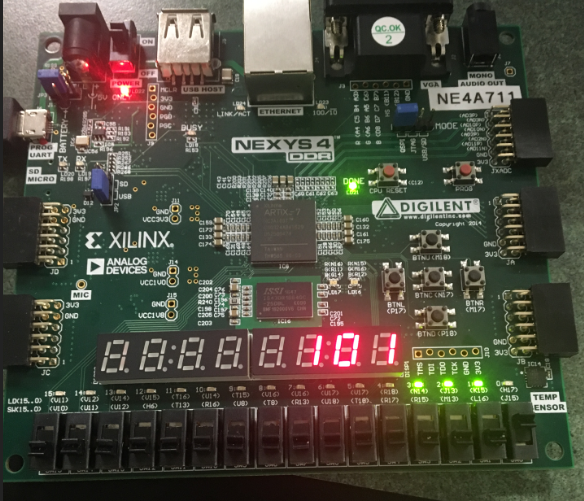


Figure 6: This figure displays b3b4 = “01”, and should then display 1, 0, 1.

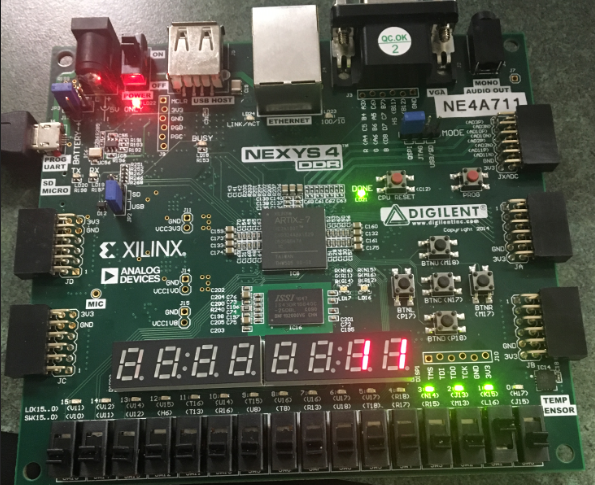


Figure 7: This figure displays b3b4 = “10”, and should then display blank, 1, 1.

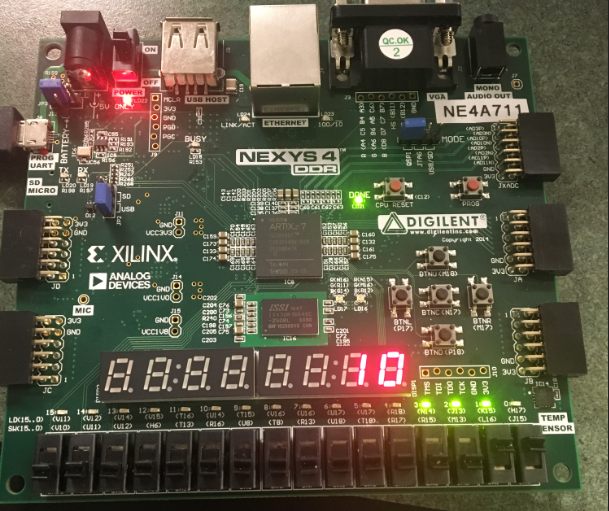


Figure 7: This figure displays b3b4 = “11”, and should then display blank, 1, 0.

## 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.

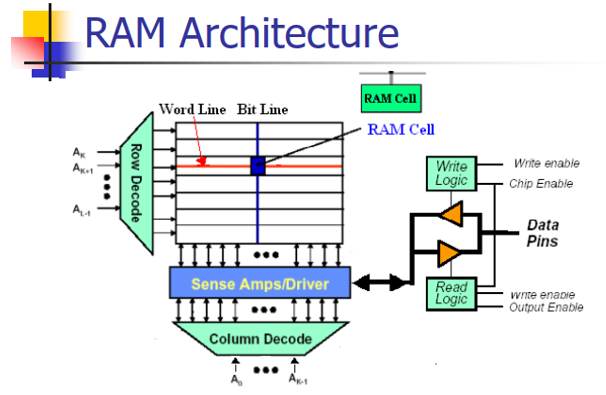
## Part 4 - RAM Design and Logic Analyzer

## 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

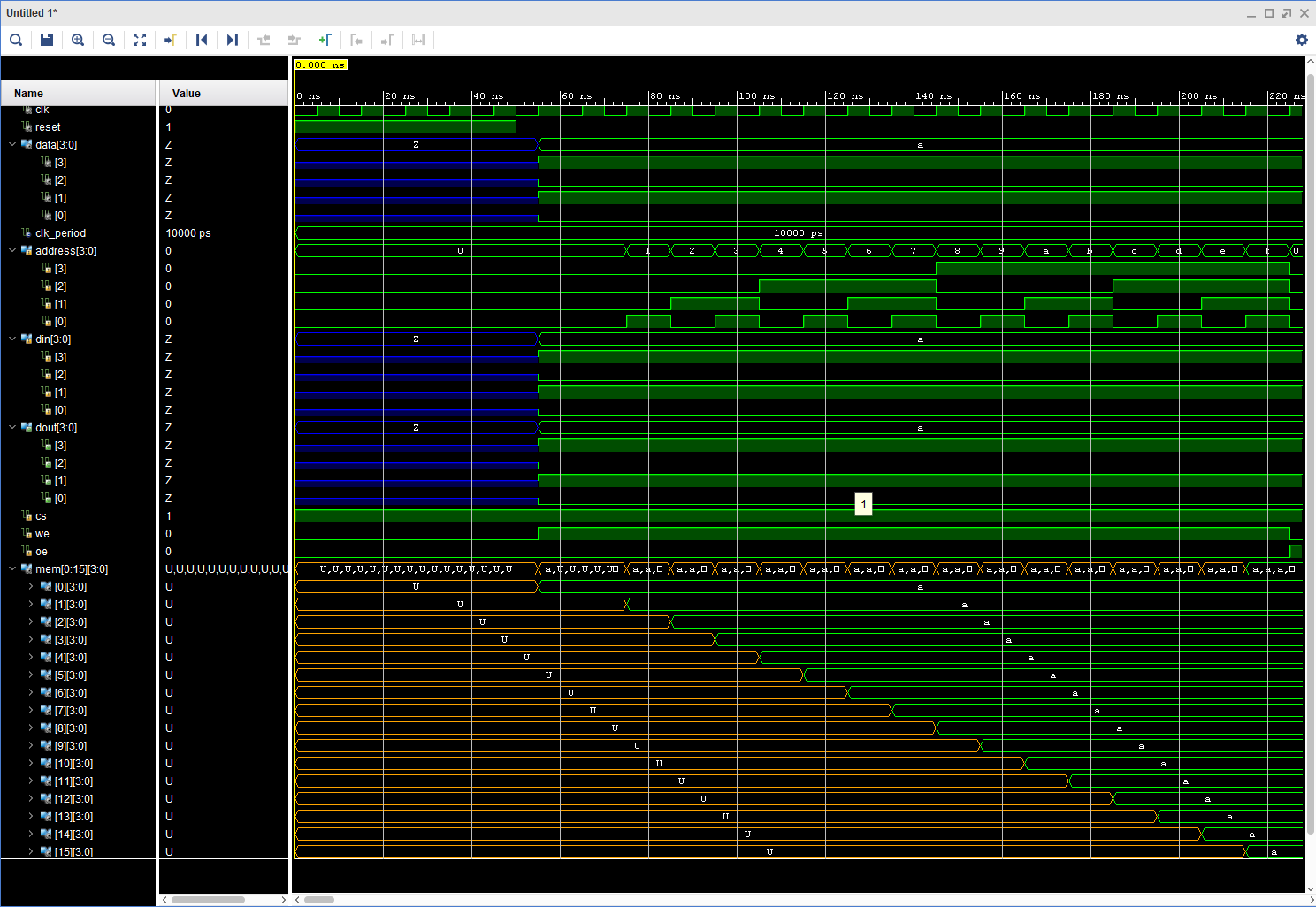


Figure 9: This figure shows the result of the Logic Analyzer after the bitstream was generated. As you can see the binary data “1010” was stored in 16 address lines.

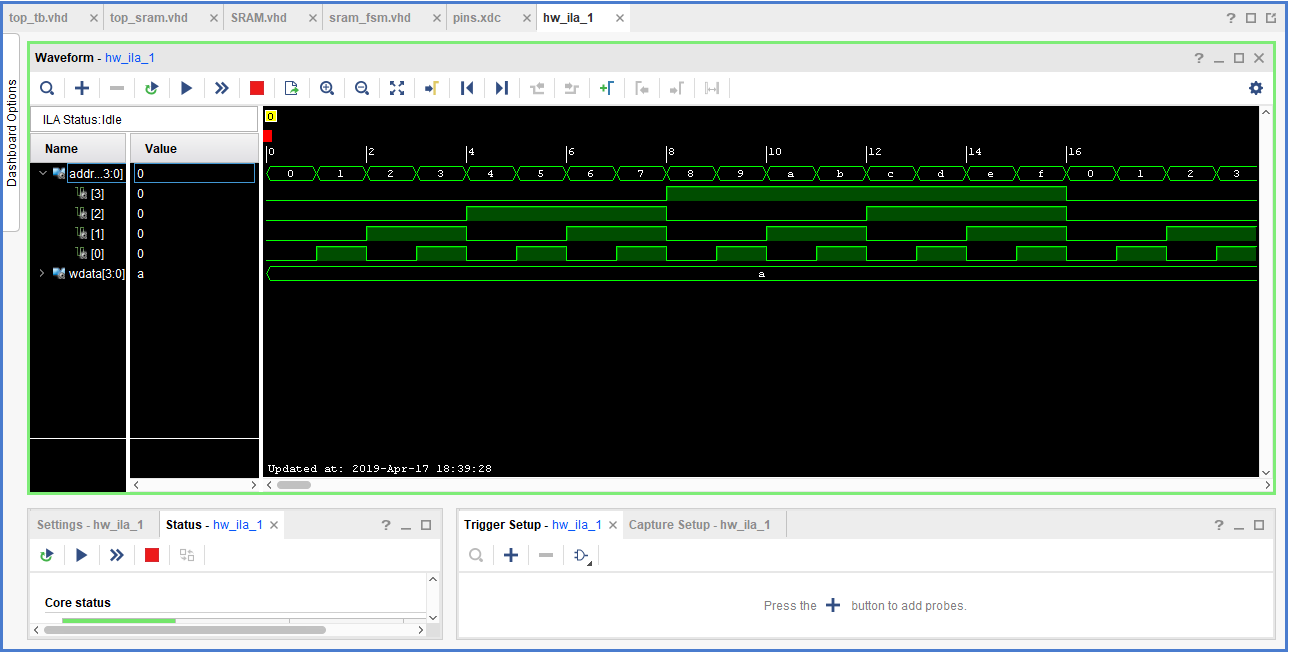


Figure 10: This figure is the simulation of the SRAM. It counts up to 15.

## 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.