Lab 3: VHDL Structural Design and Usage of Logic Analyzer

CPE 166 Advanced Logic Design Lab

Lab Session: Wednesday 2PM-445PM

Talal Jawaid

5/6/2018

Table of Contents

[Introduction 3](#Introduction)

[Part 1: 50KHz BCD Counter Design 4](#Part1)

[Part 2: Pseudorandom Number Generator and Hamming Code Display on LEDs 9](#Part2)

[Part 3: Calculator LCD Display 17](#part3)

[Part 4: Traffic Light Controller 40](#part3)

[Part 5: SRAM 59](#part3)

[Conclusion 64](#conclusion)

**Purpose:**

The purpose of this lab is to teach students how to design and implement code in VHDL. We learn how to use VHDL in the same way that we have been using Verilog HDL. We also used the FPGA to simulate the hardware to test this on. This lab consists of five parts. The first part is a 50 KHz BCD counter design so that we can base all of our events on the timing of the clock. The second part of the lab is to use a pseudorandom number generator and display hamming code on the FPGA leds. The third part of the lab is to create a calculator to run on the FPGA. Part 4 of the lab is a traffic light controller. Part 5 of the lab is to write a code to the SRAM. For most parts of the lab, we were provided guidance by the professor, or we had the code explicitly given to us by her so that we could simply run it on our FPGAs or run test benches.

**Part 1: 50 KHz BCD Counter Design**

Design Purpose:

The purpose of the BCD counter is so that it can control the timing of events based on the rising or falling edge of the clock. It can count up to ten (output hex values from 0 to F) on the application of the clock signal. The Artix 7 board only has 50MHz. So in order to measure a 50 KHz clock signal, we had to divide the count in half.

Engineering Data:

Source code:

--Talal Jawaid

--CPE 166

-- PURPOSE: bcd counter

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity counter is

port(

C, CLR, UP\_DOWN : in std\_logic;

Q: out std\_logic\_vector(3 downto 0)

);

end counter;

architecture archi of counter is

signal tmp: std\_logic\_vector(3 downto 0);

begin

process (C, CLR)

begin

if (CLR='1') then

tmp <= "0000";

elsif (C'event and C='1') then

if (UP\_DOWN='1') then

tmp <= tmp + 1;

else

tmp <= tmp - 1;

end if;

end if;

end process;

Q <= tmp;

end archi;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity clkdiv8 is

port(clk1 : in std\_logic;

clk2 : out std\_logic);

end clkdiv8;

architecture beh of clkdiv8 is

signal counter: std\_logic\_vector(2 downto 0);

begin

process (clk1)

begin

if (rising\_edge(clk1)) then

if (counter=7) then

clk2 <= '1';

counter <= (others=>'0');

elsif(counter<3) then

clk2 <= '1';

counter <= counter + 1;

else

clk2 <= '0';

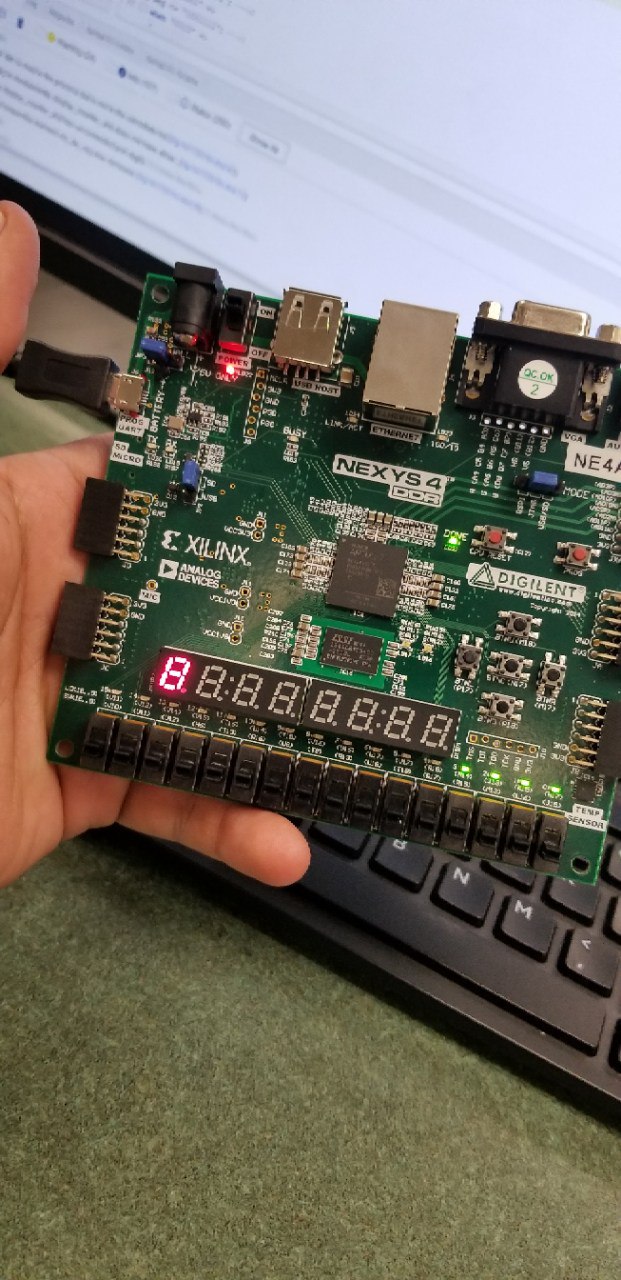
counter <= counter + 1;

end if;

end if;

end process;

end beh;



Results:

We could observe from the blinking led on the FPGA that we correctly implemented the code and wrote the constraints file accordingly. According to our lab instructor, if it wasn’t for the clock divider, the led would be blinking faster than the eye could see, resulting in the led seemingly being “on.” The fact that it was blinking at a rate visible to the human eye showed us that the clock divider worked correctly. In this way we learned that we can assign events to the edges or levels of a signal other than the “official” clock. This allows us to slow down events as we prefer.

Part 2:

Design Purpose:

The purpose of this part of the lab was to introduce us to the pseudorandom number generator so that we could understand how to use the random number generator in VHDL similar to how you would call the random() function in C++ or Java. This part of the lab also introduced us to the hamming code, where we are given the 4 bits and are asked to construct the parity bits. This is done so that a software can check using the first 4 bits whether the corresponding parity bits are correct. If they aren’t, then the software knows that there was an error in transmission.

To generate the random number, we used the LSFR to change four bits, which were our inputs, every divided clock cycle. We also had a reset button which reset the fsm. On press of the second button, the 4 bit LSFR showed up on the 4 leds. When the third button was pressed, the full hamming code was displayed on the leds.

Engineering Data:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity mux is

port (

a, b, c, d : in std\_logic;

s : in std\_logic\_vector (1 downto 0);

o : out std\_logic

);

end mux;

architecture beh of mux is

begin

process (a, b, c, d, s)

begin

case s is

when "00" => o <= a;

when "01" => o <= b;

when "10" => o <= c;

when others => o <= d;

end case;

end process;

end beh;

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 04/04/2018 03:23:05 PM

-- Design Name:

-- Module Name: DFFS - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity DFFS is

port(d, rst, clk : in std\_logic;

dffs\_out : out std\_logic);

end DFFS;

architecture Behavioral of DFFS is

begin

process(clk)

begin

if (clk'event and clk='1') then

if rst='1' then

dffs\_out <= '0';

else

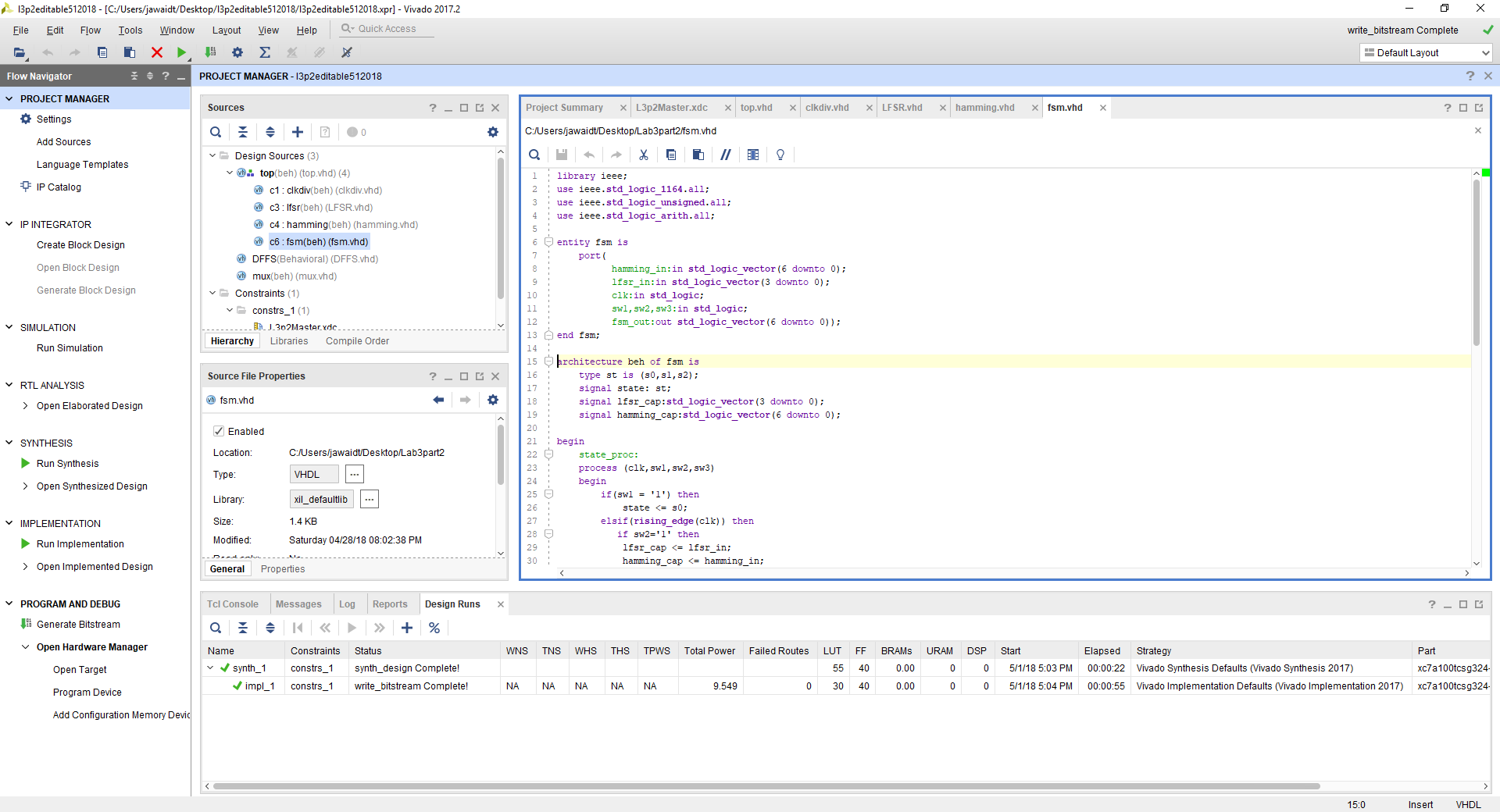
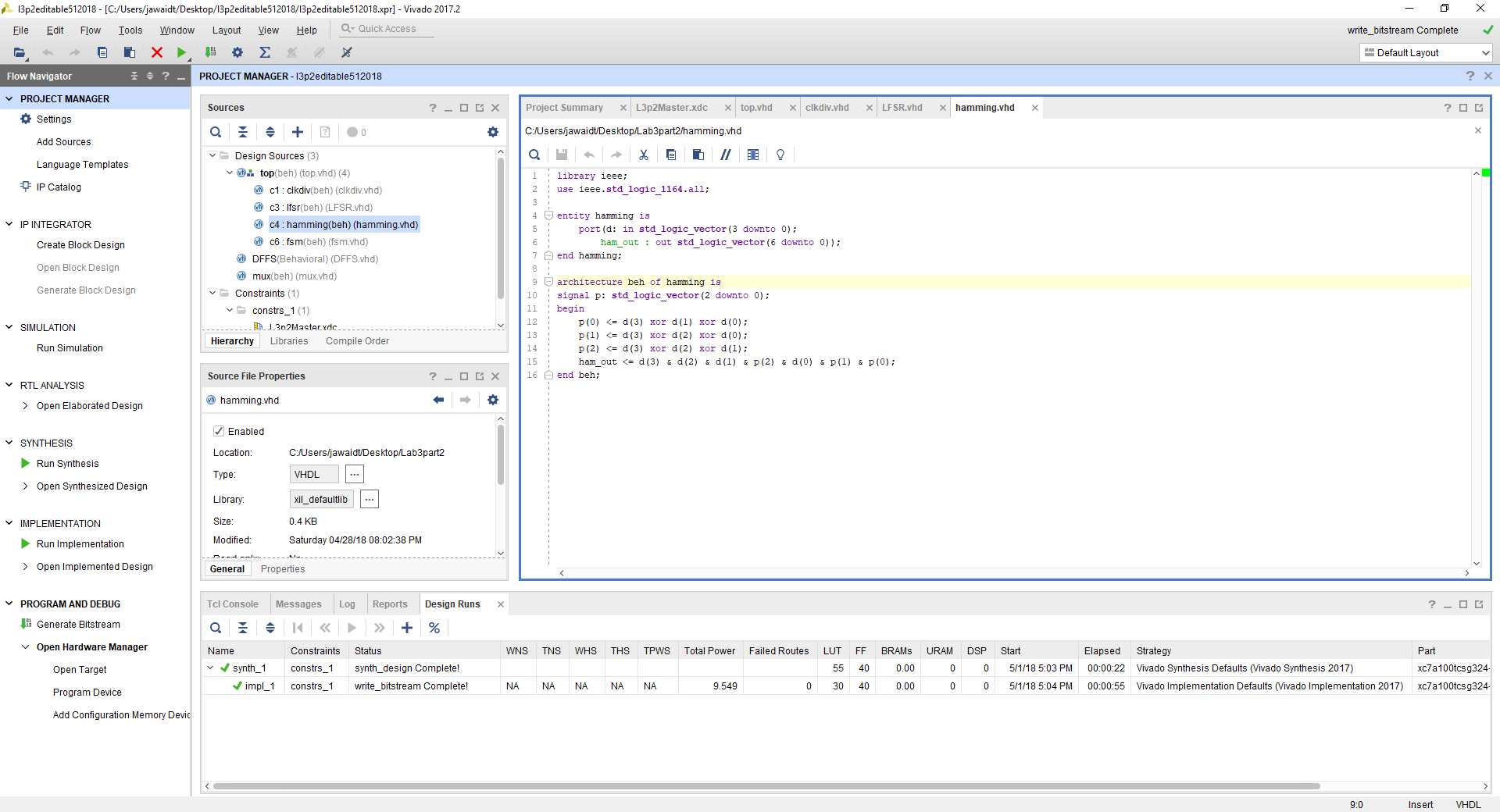
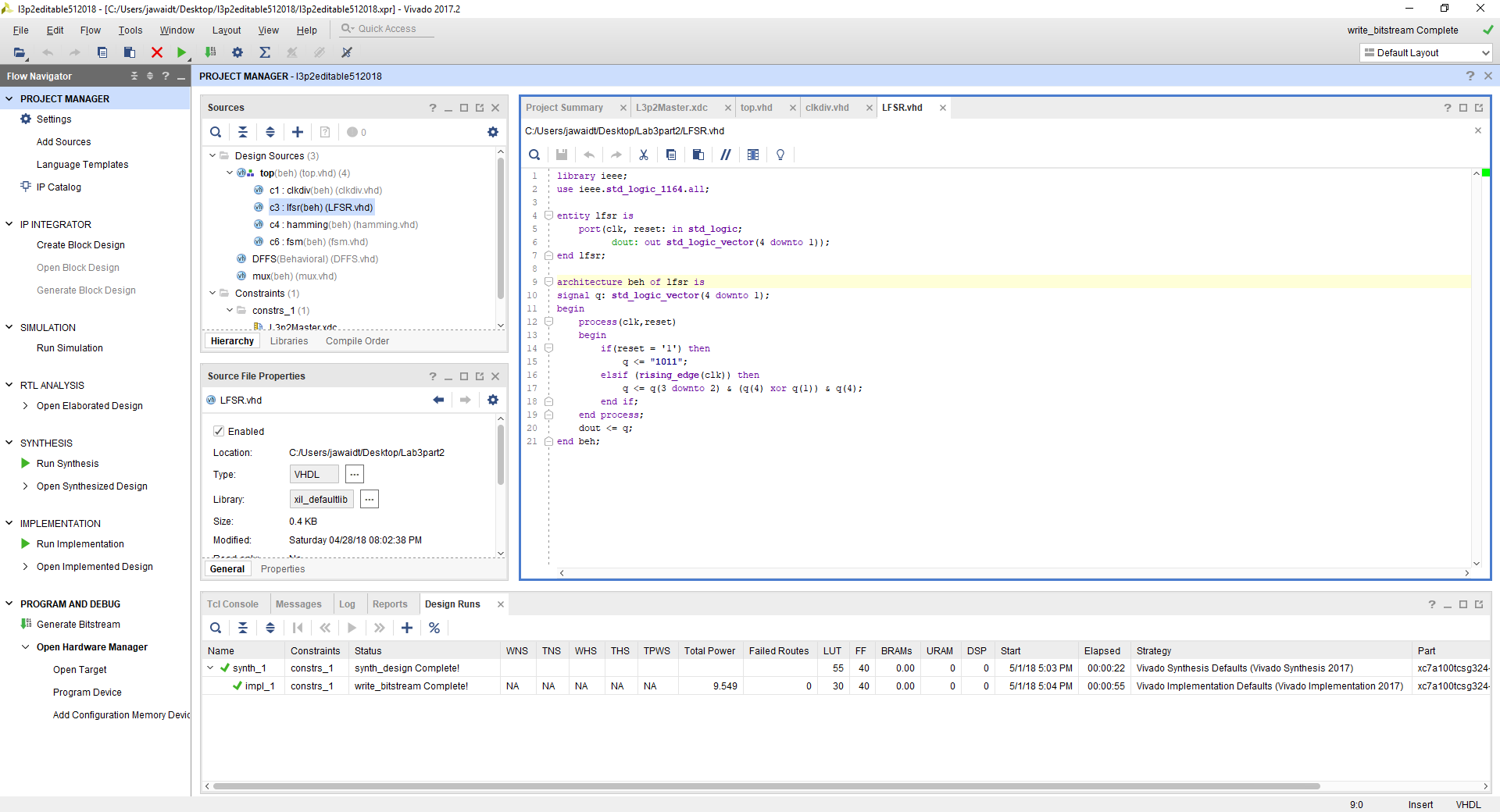
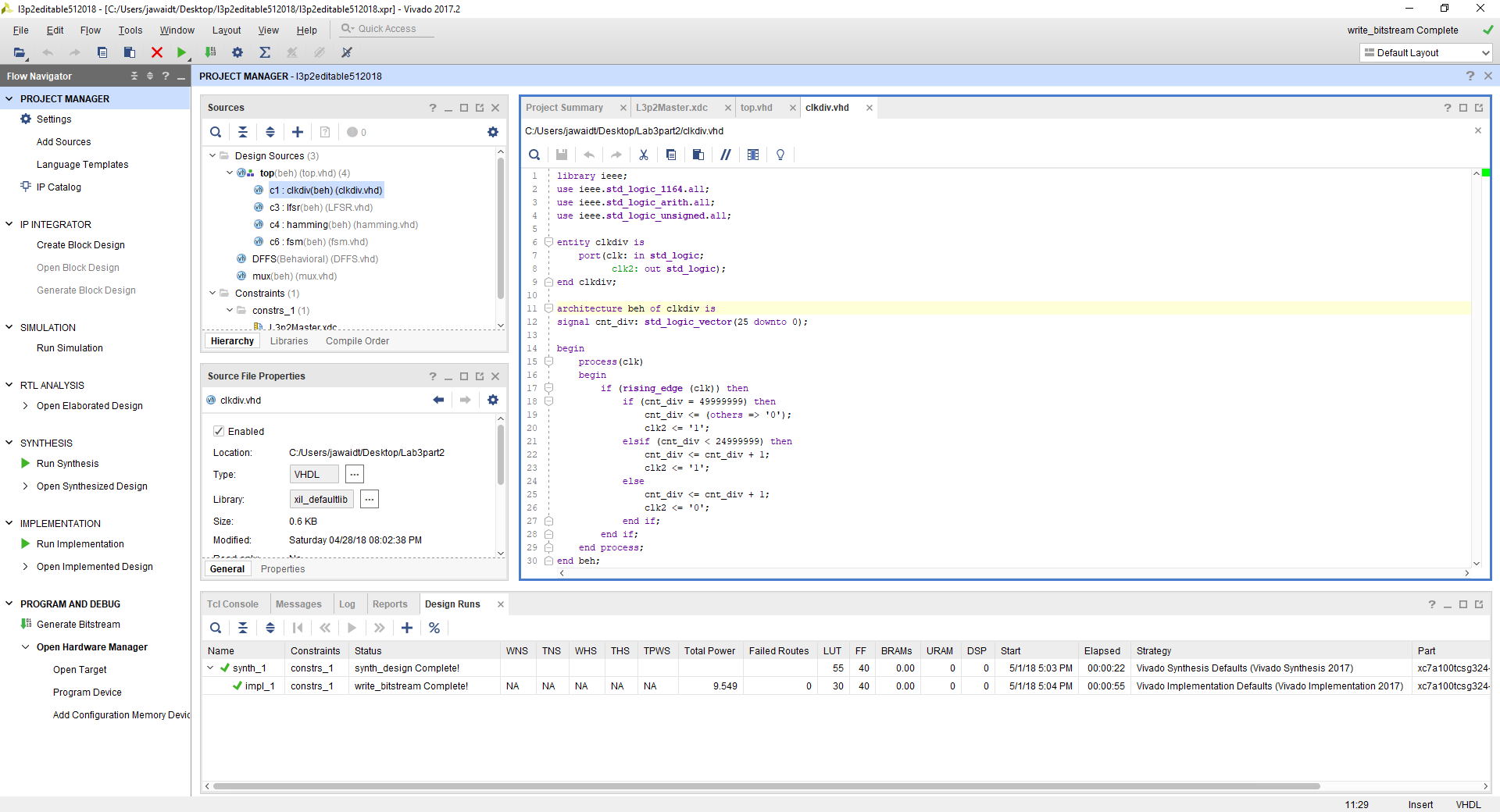
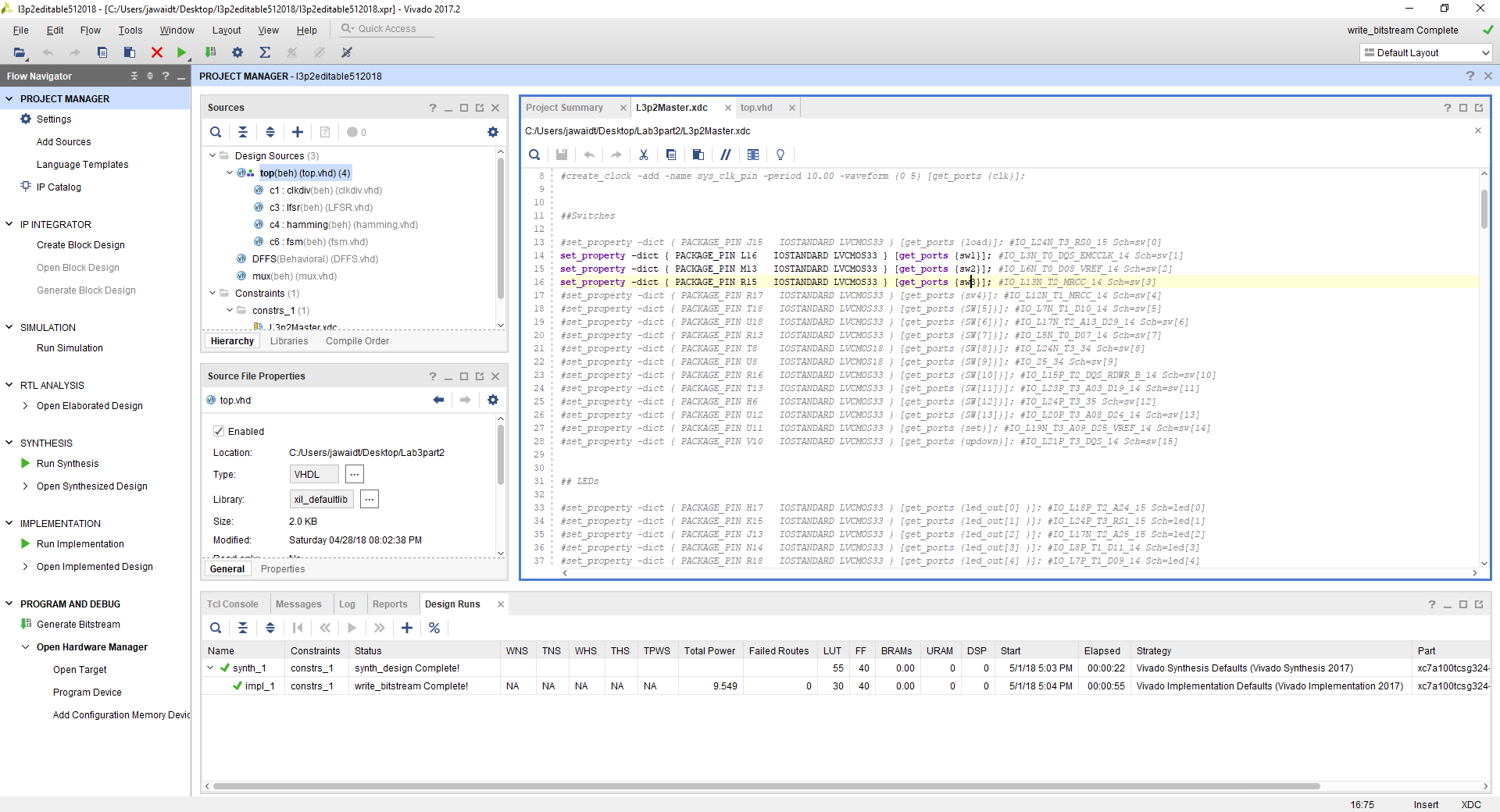
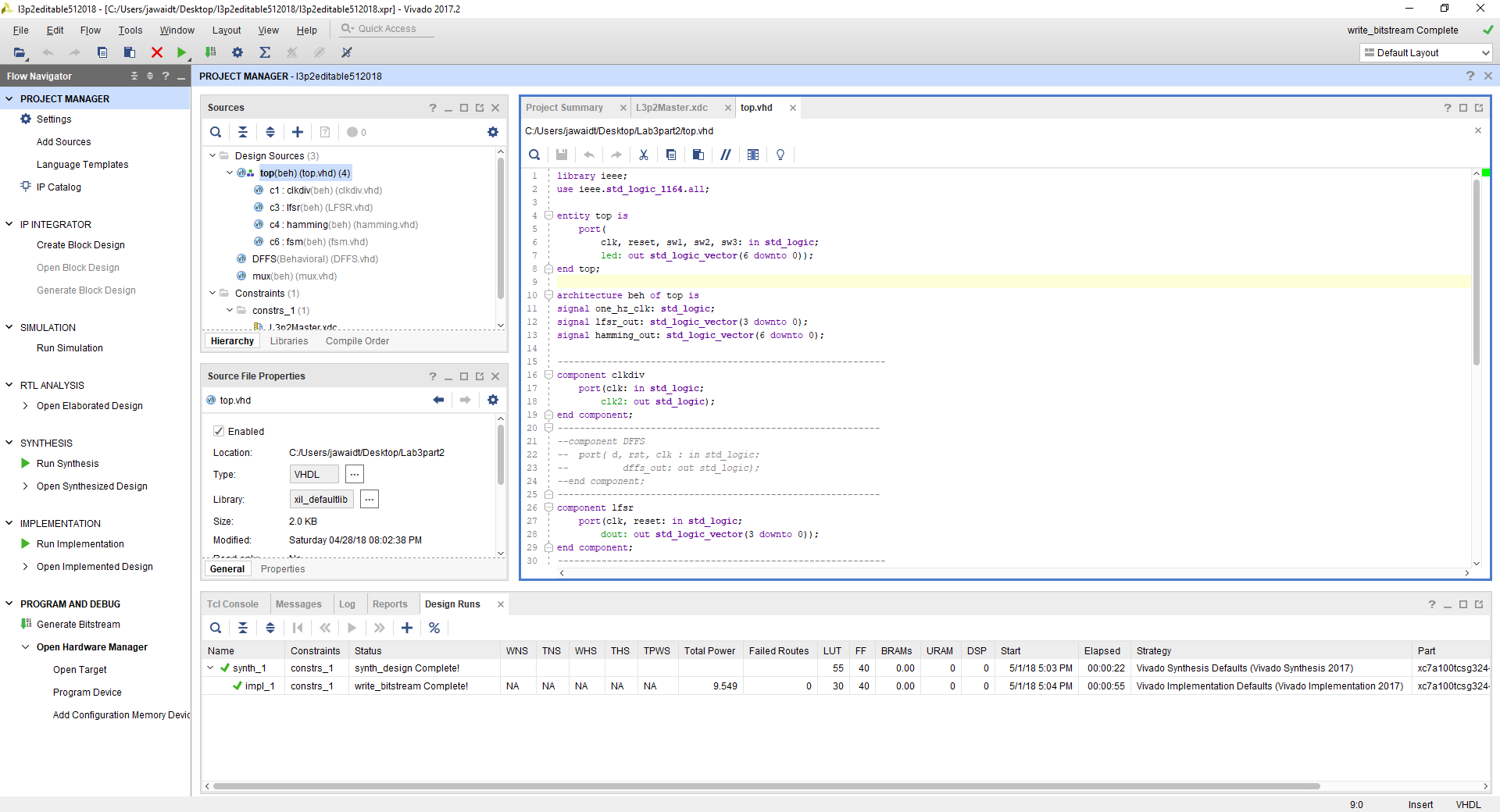
dffs\_out <= d;

end if;

end if;

end process;

end Behavioral;

**LAB 3 PART 2**

Result:

The pseudorandom number generator worked as expected. It outputted the correct hamming code as well. We had major issues implementing the professors code, due to the incomplete code as well as the lack of documentation provided. It was tough but we managed to figure out what was going on in her code.

**Part 3:**

Design Purpose:

The purpose of this part is to create a calculator to display onto the FPGA. This calculator follows the truth table provided to us in the lab manual.

Engineering Data:



library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_arith.all;

use IEEE.STD\_LOGIC\_unsigned.all;

entity cal is

port(

dout3, dout4 : in std\_logic\_vector(1 downto 0);

b1,b2,b3, b4 : in std\_logic;

f : out std\_logic\_vector(3 downto 1));

end cal;

architecture Behavioral of cal is

begin

process(b1,b2,b3,b4,dout3,dout4)

begin

if(b1='1') then

f <= "000";

elsif(b3='0' and b4='0') then

if(b1/='1' and b2/='1') then

if(dout4 > dout3 ) then

f <= "001";

else

f <= "000";

end if;

end if;

elsif(b3='0' and b4='1') then

if(b1/='1' and b2/='1') then

f <= ("0"& dout4) + ("0"& dout3) ;

end if;

elsif(b3='1' and b4='0') then

if(b1/='1' and b2/='1') then

f <= ("0"& dout4) or ("0"& dout3);

end if;

elsif(b3='1' and b4='1') then

if(b1/='1' and b2/='1') then

f <= ("0"& dout4) and ("0"& dout3) ;

end if;

end if;

end process;

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity calculator is

port (

b1,b2,b3,b4,osc : in std\_logic;

q : inout std\_logic\_vector(4 downto 1);

lcd\_e, lcd\_rs, lcd\_rw, lcd\_7, lcd\_6, lcd\_5, lcd\_4, sf\_ce0: out STD\_LOGIC

);

end calculator;

architecture Behavioral of calculator is

signal dout3,dout4 : std\_logic\_vector(1 downto 0);

signal clk,res : std\_logic;

signal lfsgen : std\_logic;

signal lfshold : std\_logic;

signal f3\_l : STD\_LOGIC\_VECTOR (5 downto 0);

signal f3\_u : STD\_LOGIC\_VECTOR (5 downto 0);

signal f2\_l : STD\_LOGIC\_VECTOR (5 downto 0);

signal f2\_u : STD\_LOGIC\_VECTOR (5 downto 0);

signal f1\_l : STD\_LOGIC\_VECTOR (5 downto 0);

signal f1\_u : STD\_LOGIC\_VECTOR (5 downto 0);

signal f : std\_logic\_vector (3 downto 1);

component counter is

port(

osc : in std\_logic;

b1 : in std\_logic;

clk : out std\_logic );

end component;

component fsm is

port(

b1,b2,clk : in std\_logic;

res,lfsgen,lfshold : out std\_logic);

end component;

component lfsr is

port(

b2,clk,res,lfsgen,lfshold : in std\_logic;

q : out std\_logic\_vector(4 downto 1));

end component;

component doutt is

port(

q : in std\_logic\_vector (4 downto 1);

dout3,dout4 : out std\_logic\_vector (1 downto 0));

end component;

component cal is

port(

dout3, dout4 : in std\_logic\_vector(1 downto 0);

b1,b2,b3, b4 : in std\_logic;

f : out std\_logic\_vector(3 downto 1));

end component cal;

component lcdvalue

port(

f : in std\_logic\_vector(3 downto 1);

clk : in std\_logic;

f3\_l : out STD\_LOGIC\_VECTOR (5 downto 0);

f3\_u : out STD\_LOGIC\_VECTOR (5 downto 0);

f2\_l : out STD\_LOGIC\_VECTOR (5 downto 0);

f2\_u : out STD\_LOGIC\_VECTOR (5 downto 0);

f1\_l : out STD\_LOGIC\_VECTOR (5 downto 0);

f1\_u : out STD\_LOGIC\_VECTOR (5 downto 0) );

end component;

component lcd\_display

port(

clk : in std\_logic;

f : in std\_logic\_vector(3 downto 1);

lcd\_e, lcd\_rs, lcd\_rw, lcd\_7, lcd\_6, lcd\_5, lcd\_4, sf\_ce0 : out std\_logic);

end component;

begin

q0 : counter port map(osc,b1,clk);

q1 : fsm port map( b1,b2,clk,res,lfsgen,lfshold);

q2 : lfsr port map(b2,clk,res,lfsgen,lfshold,q);

q3 : doutt port map(q,dout3,dout4);

q4 : cal port map(dout3, dout4, b1,b2,b3, b4,f);

q6 : lcd\_display port map(osc,f,lcd\_e, lcd\_rs, lcd\_rw, lcd\_7, lcd\_6, lcd\_5, lcd\_4, sf\_ce0);

end Behavioral;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity counter is

port(

osc : in std\_logic;

b1 : in std\_logic;

clk : out std\_logic

);

end counter;

architecture beh of counter is

signal clkref : std\_logic;

signal cnt10 : std\_logic\_vector(26 downto 0);

begin

clk <= clkref;

process(osc,b1)

begin

if(b1 = '1') then

cnt10 <= (others=>'0');

clkref <='0';

elsif( rising\_edge(osc) ) then

if (cnt10 = 49999999 ) then

cnt10 <= ( others => '0' );

clkref <= '1';

elsif (cnt10 < 24999999) then

cnt10 <= cnt10 + 1;

clkref <= '1';

else

cnt10 <= cnt10 + 1;

clkref <= '0';

end if;

end if;

end process;

end beh;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity doutt is

port(

q : in std\_logic\_vector ( 4 downto 1);

dout3, dout4 : out std\_logic\_vector(1 downto 0));

end doutt;

architecture Behavioral of doutt is

begin

process(q)

begin

dout3 <= q(2)&q(1);

dout4 <= q(4)&q(3);

end process;

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity fsm is

port(

b1,b2,clk : in std\_logic;

res,lfsgen,lfshold : out std\_logic);

end fsm;

architecture Behavioral of fsm is

type state is (s1,s2,s3);

signal cs,ns : state;

begin

process(clk,b1)

begin

if(b1 ='1') then

cs <= s1;

elsif(rising\_edge(clk)) then

cs <= ns;

end if;

end process;

process(cs,b1,b2)

begin

case (cs) is

when s1 =>

if (b1 = '1') then

ns <= s1;

elsif (b2 = '1') then

ns<=s2;

elsif(b2 = '0') then

ns <= s3;

end if;

when s2 =>

if (b1 = '1') then

ns <= s1;

elsif (b2 = '1') then

ns<=s2;

elsif(b2 = '0') then

ns <= s3;

end if;

when s3 =>

if (b1 = '1') then

ns <= s1;

elsif (b2 = '1') then

ns<=s2;

elsif(b2 = '0') then

ns <= s3;

end if;

end case;

end process;

process(cs)

begin

case (cs) is

when s1 =>

res <= '1';

lfsgen <= '0';

lfshold <= '0';

when s2 =>

res <= '0';

lfsgen <= '1';

lfshold <= '0';

when s3 =>

res <= '0';

lfsgen <= '0';

lfshold <= '1';

end case;

end process;

end Behavioral;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity lcd\_display is

port(

clk : in std\_logic;

f : in std\_logic\_vector(3 downto 1);

lcd\_e, lcd\_rs, lcd\_rw, lcd\_7, lcd\_6, lcd\_5, lcd\_4, sf\_ce0 : out std\_logic);

end lcd\_display;

architecture beh of lcd\_display is

signal lcd\_busy, lcd\_stb : std\_logic;

signal lcd\_code : std\_logic\_vector (5 downto 0);

signal lcd\_stuff : std\_logic\_vector (6 downto 0);

signal count : std\_logic\_vector (24 downto 0);

signal lcd\_rw\_1 : std\_logic;

begin

process(clk)

begin

if (rising\_edge (clk)) then

lcd\_busy <= '1';

sf\_ce0 <= '1';

count <= count + 1;

case count (24 downto 20) is

when "00000" =>

lcd\_code <= "000011";

when "00001" =>

lcd\_code <= "000011";

when "00010" =>

lcd\_code <= "000011";

when "00011" =>

lcd\_code <= "000010";

when "00100" =>

lcd\_code <= "000010";

when "00101" =>

lcd\_code <= "001000";

when "00110" =>

lcd\_code <= "000000";

when "00111" =>

lcd\_code <= "000110";

when "01000" =>

lcd\_code <= "000000";

when "01001" =>

lcd\_code <= "001100";

when "01010" =>

lcd\_code <= "000000";

when "01011" =>

lcd\_code <= "000001";

when "01100" =>

lcd\_code <= "100011";

when "01101" =>

lcd\_code <= "10000"&f(3); --f3

when "01110" =>

lcd\_code <= "100011";

when "01111" =>

lcd\_code <= "10000"&f(2); --f2

when "10000" =>

lcd\_code <= "100011";

when "10001" =>

lcd\_code <= "10000"&f(1); --f1

when others =>

lcd\_code <= "010000";

end case;

lcd\_rw <= lcd\_rw\_1;

lcd\_stb <= (count(19) xor count(18)) and (not lcd\_rw\_1) and lcd\_busy;

lcd\_stuff <= (lcd\_stb) & lcd\_code;

(lcd\_e, lcd\_rs, lcd\_rw\_1, lcd\_7, lcd\_6, lcd\_5, lcd\_4) <= lcd\_stuff;

end if;

end process;

end beh;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity lfsr is

port(

b2,clk,res,lfsgen,lfshold : in std\_logic;

q : out std\_logic\_vector(4 downto 1)

);

end lfsr;

architecture beh of lfsr is

signal cnt10: std\_logic\_vector(4 downto 1);

begin

q <= cnt10;

process(clk,res,lfsgen,lfshold,b2)

begin

if(res='1') then

cnt10<="1111";

elsif(lfsgen = '1') then

if(rising\_edge(clk)) then

cnt10(4)<=cnt10(3);

cnt10(3)<=cnt10(2);

cnt10(2)<=cnt10(4) xor cnt10(1);

cnt10(1)<=cnt10(4);

end if;

elsif(lfshold = '1') then

cnt10(4)<=cnt10(4);

cnt10(3)<=cnt10(3);

cnt10(2)<=cnt10(2);

cnt10(1)<=cnt10(1);

end if;

end process;

end beh;

Results:

We were able to get the calculator working as according to the truth table. However, we were unable to completely understand the code provided to us by the professor. The only reason the code works is because it was provided to us. Even when we made modifications to the code, the output still remained the same, so we are still confused as to how the code actually works.

Part 4:

Design Purpose:

This part of the lab, we had to implement a traffic light in VHDL. This part of the lab was simple as all we had to do was take the code that was given to us and add some states to the given state machine. The state machine consisted of each state that a traffic light can consist of. This includes, red, yellow, green, and a default state.

Engineering Data:

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.std\_logic\_unsigned.all;

use IEEE.std\_logic\_arith.all;

entity fsm is

port ( clk, reset : IN std\_logic;

R,G,Y : OUT std\_logic );

end fsm;

architecture beh of fsm is

type state\_type is (s1,s2,s3);

signal state: state\_type ;

signal count : STD\_LOGIC\_VECTOR(3 downto 0);

begin

process (clk,reset)

begin

if (reset ='1') then

count <= "0000";

state <=s1;

elsif (clk='1' and clk'event) then

case state is

when s1 =>

count <= count + 1;

if (count < 8) then

state <= s1;

-- count <= count+1;

elsif (count = 8) then

state <= s2;

count <= "0000";

end if;

when s2 =>

count <= count + 1;

if (count < 8) then

state <= s2;

--count <= count+1;

elsif (count = 8) then

state <= s3;

count <="0000";

end if;

when s3 =>

count <= count + 1;

if (count < 8) then

state <= s3;

elsif (count = 8) then

state <= s1;

end if;

when others => state <= s1;

end case;

end if;

end process;

process(state)

begin

case state is

when s1 =>

R <='1';

G <='0';

Y <='0';

when s2 =>

R <='0';

G <='0';

Y <='1';

when s3 =>

R <='0';

G <='1';

Y <='0';

when others=>

R <='0';

G <='0';

Y <='0';

end case;

end process;

end beh;

Test bench:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.numeric\_std.all;

entity fsm\_tb is

end;

architecture test of fsm\_tb is

component fsm

port (

clk: in std\_logic;

reset: in std\_logic;

R: out std\_logic;

G: out std\_logic;

Y: out std\_logic);

end component;

signal clk : std\_logic := '0';

signal reset : std\_logic := '1';

signal R : std\_logic;

signal G : std\_logic;

signal Y : std\_logic;

begin

uut: fsm PORT MAP(

clk => clk,

reset => reset,

R=>R,

G=>G,

Y=>Y

);

--port map(clk,reset,R,G,Y);

clk\_stimulus: process

begin

wait for 10 ns;

clk <= not clk;

end process;

data\_stimulus: process

begin

reset <= '0';

wait for 300 ns;

end process;

end test;

Results:

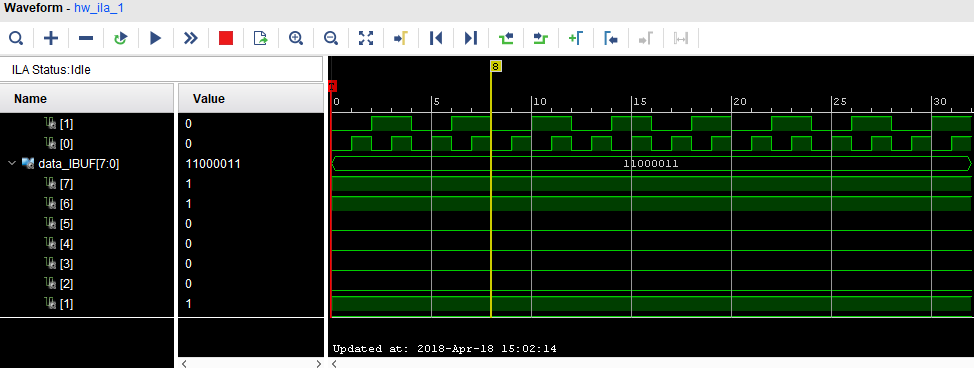
Overall, this part of the lab was very simple. It helped us understand how to implement state machines in VHDL similar to how we implemented them in Verilog in earlier labs and semesters. If I could make a recommendation for future semesters, I would make this the first part of lab 3 as it is the easiest to understand in terms of the code. Also, any changes you make are easily reproducible on the FPGA as well as the simulation, so it does not require much extra work. After the first part of lab 3, we chose to skip ahead and do part 4, which prepared us well for the remaining parts of the lab.

Part 5:

Design Purpose:

The purpose of this part of the lab was to write a code to the SRAM. This was a very difficult part of the lab and we don’t truly know how we implemented it. The code provided to us was for writing a code to the RAM but according to our lab instructor, we had to write the code to the SRAM instead. This required understanding the code that was provided to us and then rewriting the code so that it would fit the requirements of the lab.

Engineering Data:



library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity clkdiv is

port ( Osc: in std\_logic;

clk: out std\_logic);

end clkdiv;

architecture Behavioral of clkdiv is

signal clk\_10kh: std\_logic;

signal cnt\_div: std\_logic\_vector(12 downto 0);

begin

process(Osc)

begin

if (rising\_edge (Osc)) then

if (cnt\_div = 4999) then

cnt\_div<=(others => '0');

clk\_10kh <= '1';

elsif (cnt\_div < 2499) then

cnt\_div <= cnt\_div + 1;

clk\_10kh <= '1';

else

cnt\_div <= cnt\_div + 1;

clk\_10kh <= '0';

end if;

end if;

end process;

clk <= clk\_10kh;

end Behavioral;

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.std\_logic\_unsigned.all;

use IEEE.std\_logic\_arith.all;

entity mem\_fsm is

port ( clk, reset : IN std\_logic;

address : OUT std\_logic\_vector(3 downto 0);

data: INOUT std\_logic\_vector(7 downto 0);

cs, we, oe: OUT std\_logic );

end mem\_fsm;

architecture fsm\_beh of mem\_fsm is

type state\_type is (idle, s1,s2,s3,s4);

signal state: state\_type ;

signal cnt: std\_logic\_vector(3 downto 0);

begin

cs <= '1';

address <= cnt;

process (clk,reset)

begin

if (reset ='1') then

state <= idle;

cnt <= "0000";

elsif (clk='1' and clk'event) then

case state is

when idle =>

state <= s1;

cnt <= "0000";

when s1 =>

state <= s2;

cnt <= "0000";

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'; data <= "ZZZZZZZZ";

when s1 => we <= '1'; oe <= '0'; data <= "11000011";

when s2 => we <= '1'; oe <= '0'; data <= "11000011";

when s3 => we <= '0'; oe <= '1'; data <= "ZZZZZZZZ";

when s4 => we <= '0'; oe <= '1'; data <= "ZZZZZZZZ";

when others => we <= '0'; oe <= '0'; data <= "ZZZZZZZZ";

end case;

end process;

end fsm\_beh;

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY mem\_top IS

port(

clk : IN std\_logic;

reset : IN std\_logic;

data : inout std\_logic\_vector ( 7 downto 0);

addr\_out : OUT std\_logic\_vector(3 downto 0);

cs\_out : OUT std\_logic;

we\_out : OUT std\_logic;

oe\_out : OUT std\_logic

);

END mem\_top;

ARCHITECTURE behavior OF mem\_top IS

component ram

port (

address :in std\_logic\_vector ( 3 downto 0);

data :inout std\_logic\_vector ( 7 downto 0);

cs :in std\_logic;

we :in std\_logic;

oe :in std\_logic

);

end component;

COMPONENT mem\_fsm

PORT(

clk : IN std\_logic;

reset : IN std\_logic;

address : OUT std\_logic\_vector(3 downto 0);

data : INOUT std\_logic\_vector(7 downto 0);

cs : OUT std\_logic;

we : OUT std\_logic;

oe : OUT std\_logic

);

END COMPONENT;

signal address : std\_logic\_vector ( 3 downto 0);

signal cs : std\_logic;

signal we : std\_logic;

signal oe : std\_logic;

BEGIN

cs\_out <= cs;

we\_out <= we;

oe\_out <= oe;

addr\_out <= address;

g1: mem\_fsm PORT MAP (

clk => clk,

reset => reset,

address => address,

data => data,

cs => cs,

we => we,

oe => oe

);

g2: ram

port map(

address => address,

data => data,

cs => cs,

we => we,

oe => oe

);

END;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity ram is

port (

address :in std\_logic\_vector ( 3 downto 0);

data :inout std\_logic\_vector ( 7 downto 0);

cs :in std\_logic;

we :in std\_logic;

oe :in std\_logic

);

end ram;

architecture beh\_ram of ram is

type memory is array (0 to 15)of std\_logic\_vector (7 downto 0);

signal mem : memory ;

begin

MEM\_WRITE:

process (address, data, cs, we)

begin

if (cs = '1' and we = '1') then

mem(conv\_integer(address)) <= data;

end if;

end process;

MEM\_READ:

process (address, cs, we, oe, mem) begin

if (cs = '1' and we = '0' and oe = '1') then

data <= mem(conv\_integer(address));

else

data <= (others => 'Z' );

end if;

end process;

end beh\_ram;

Results:

We were able to get the code working after several attempts at rewriting the code provided to us. It was difficult to understand and we do not truly understand how the SRAM works or how we are able to write to it using VHDL. Without being given the code by the professor, we don’t believe we would have been able to complete this part of the lab. Luckily this part of the lab was simple conceptually, so all we had to do was write the given code to the SRAM and it was successful.

**Conclusion**

Through the five parts of this lab, we learned how to implement a BCD counter, a pseudorandom generator using a LSFR, a way to display hamming code to the FPGA, and how to implement a calculator. This lab took multiple months to complete so we believe that it is easily the most difficult of the labs. The most difficult part of this lab was implementing the SRAM instead of the RAM. We would really hope that in the future semesters, this lab could be reorganized with lab 3 part 4 as the last part of the lab. Also if lab 3 was the last lab instead of the third lab, it would help students a lot in terms of time management for the current lab four and lab five.