Lab 2: Binary Combinational Array Multiplier Design, Sequential Multiplier Design

CPE 166/EEE 270 Advanced Logic Design Lab

Lab Session: Wednesday 2:00 – 4:40

Section: 02

Daniel Komac

Table of Contents

[**Introduction:**](#_jnn0xx68vuq7) **3**

[**Part 1: Combinational Logic Multiplier**](#_tumun8oc51ph) **4**

[Source Code:Half Adder](#_nt23gxf0etvs) 4

[Source Code:Full Adder](#_rcwh9vmohrab) 6

[Four by Four Multiplier:](#_7h13mswrs4oo) 8

[**Part 2: 4 by 4 Binary Sequential Multiplier Design**](#_6h21ptpjvs27) **13**

[Source Code:Adder](#_bdmuwk2acisg) 13

[Source Code:D Flip-Flop A](#_5vak3nbmk4sv) 13

[Source Code: D Flip-Flop B](#_vzoldigbgvg1) 15

[Source Code: FSM](#_y2ot6ftgdwhy) 16

[Source code: Mult (Multiplier)](#_46puae7f7axe) 23

[Source code: Mux](#_8owa8jskys2e) 24

[Source code: Top Module](#_f1e92y8kljlz) 26

[**Part 3: LCD**](#_yc7hhxy0djce) **28**

[Source Code: LCD](#_lbz4oqu5tbbd) 28

[**Conclusion:**](#_fyx4meh500ex) **32**

# 

# Introduction:

This lab explores two different ways to create a 4 by 4-binary multiplier in Verilog. Broken into two parts, Part 1 will use combinational logic while Part 2 will use sequential logic to create the multiplier. A time-independent digital circuit implemented using logic gates whose output depends solely on its inputs may be considered to be using combinational logic. A digital circuit using sequential logic will have an output that is dependent on the inputs as well as what state the circuit is in. Both programs can take two 4 bit binary numbers and return an 8-bit number that is the result of the two numbers multiplied together. A hierarchical design will be used to implement both programs. A top-level method will be made up of various lower level methods to create the multipliers. Every method will be created and tested individually, where they will eventually be linked together by a higher-level module. Using a hierarchical design, the top level module will be neat and easy understand; given two 4-bit inputs, the 8-bit output will be the two inputs multiplied. Then at the very end, the lap explores on how to display letters on the LCD from using verilog code. The lab serves as an introduction for writing modules in Verilog as well as creating test benches in Verilog to verify the modules are working through simulation. The waveforms from the simulations will be analyzed to show the module is working as intended.

# Part 1: Combinational Logic Multiplier

## Source Code:Half Adder

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: HalfAdder.v

--Purpose of Code:Show the half adder logic.

--Project Part Number: Part 1 – A

module HalfAdder(a,b,c\_out,s);

input a,b;

output c\_out,s;

assign s = a^b;

assign c\_out = a&b;

endmodule

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: halfAdder\_tb.v

--Purpose of Code:Test the half adder logic.

--Project Part Number: Part 1 – A

module halfadder\_tb; //test batch

reg ta, tb;

wire tcout, tsum;

halfAdder u1 (.a(ta), .b(tb), .c\_out(tc\_out), .s(ts)); //name association

initial

begin

begin ta = 0; tb = 0;

#10; ta = 0; tb = 1;

#10; ta = 1; tb = 0;

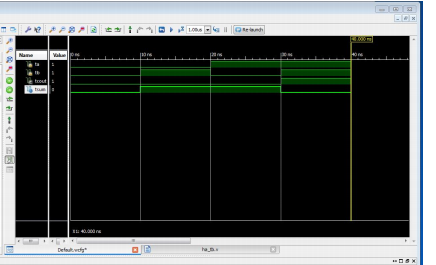
#10; ta = 1; tb = 1;

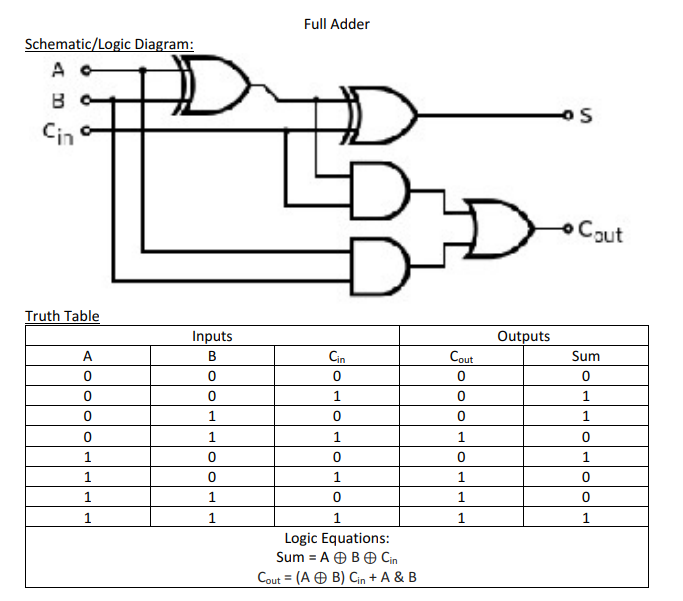
#10 $stop;

end

endmodule

Simulation Waveform:





## Source Code:Full Adder

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: FullAdder.v

--Purpose of Code:Show the Full adder logic.

--Project Part Number: Part 1 – B

module FullAdder(a,b,cin,cout,s);

input a,b,c\_in;

wire m,n,p;

output s,c\_out;

HalfAdder i1(.a(a),.b(b),.cout(n),.s(m));

HalfAdder i2(.a(cin),.b(m),.cout(p),.s(s));

assign c\_out = n|p;

endmodule

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: FullAdder\_tb.v

--Purpose of Code:Test the Full adder logic.

--Project Part Number: Part 1 – B

Module fulladder\_tb;

Reg a, b, cin; wire cout, sum;

FullAdder u3(.a(a),.b(b),.cin(cin),.cout(cout), .s(s));

initial

begin

a=0;b=0;cin=0;

#10 a=0;b=0;cin=1;

#10;a=0;b=1;cin=0;

#10;a=0;b=1;cin=1;

#10;a=1;b=0;cin=0;

#10;a=1;b=0;cin=1;

#10;a=1;b=1;cin=0;

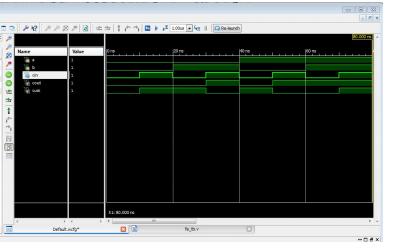
#10;a= 1;b=1;cin=1;

#10$stop;

end

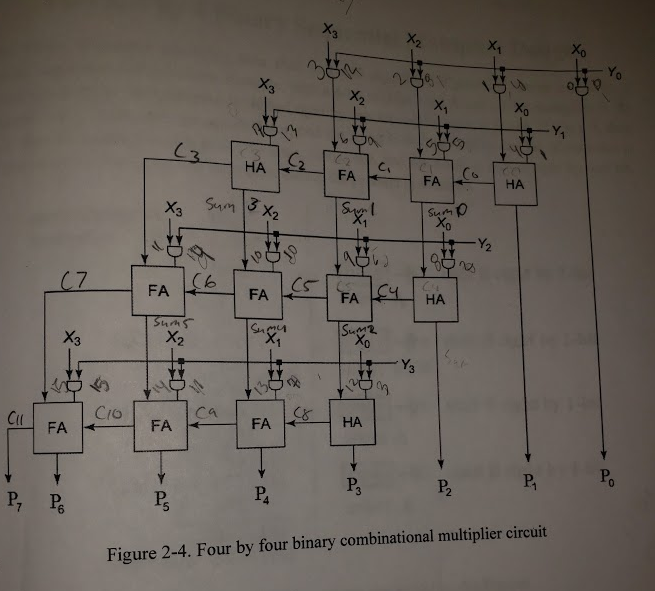
endmodule

Simulation Waveform:

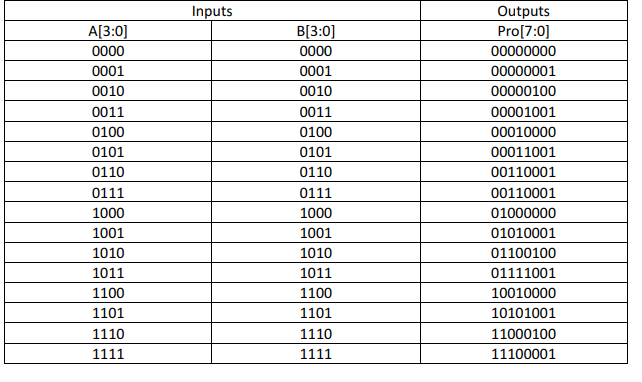


## Four by Four Multiplier:

Schematic / Logic Diagram:



Truth Table:



Source Code:

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: myMulti.v

--Purpose of Code:Show the four by four combinational multiplexer logic

--Project Part Number: Part 1 – C

module myMulti(x,y,p);

input [3:0] x,y;

output [7:0] p;

wire [15:0] andRes;

wire [10:0] c;

wire [5:0] sum;

assign andRes[0] = y[0] & x[0];

assign andRes[1] = y[1] & x[0];

assign andRes[2] = y[2] & x[0];

assign andRes[3] = y[3] & x[0];

assign andRes[4] = y[0] & x[1];

assign andRes[5] = y[1] & x[1];

assign andRes[6] = y[2] & x[1];

assign andRes[7] = y[3] & x[1];

assign andRes[8] = y[0] & x[2];

assign andRes[9] = y[1] & x[2];

assign andRes[10] = y[2] & x[2];

assign andRes[11] = y[3] & x[2];

assign andRes[12] = y[0] & x[3];

assign andRes[13] = y[1] & x[3];

assign andRes[14] = y[2] & x[3];

assign andRes[15] = y[3] & x[3];

//First place output

assign p[0] = andRes[0];

//Second place output

HalfAdder add1(.a(andRes[1]), .b(andRes[4]),.c\_out(c[0]),.s(p[1]));

//Third place output

FullAdder fAdd1(.a(andRes[8]),.b(andRes[5]),.c\_in(c[0]),.c\_out(c[1]),.s(sum[0]));

HalfAdder add2(.a(sum[0]),.b(andRes[2]),.c\_out(c[4]),.s(p[2]));

//Fourth Place output

FullAdder fAdd2(.a(andRes[9]),.b(andRes[12]),.c\_in(c[1]),.c\_out(c[2]),.s(sum[1]));

FullAdder fAdd3(.a(sum[1]),.b(andRes[6]),.c\_in(c[4]),.c\_out(c[5]),.s(sum[2]));

HalfAdder add3(.a(andRes[3]),.b(sum[2]),.c\_out(c[8]),.s(p[3]));

//Fifth Place output

HalfAdder add4(.a(andRes[13]),.b(c[2]),.c\_out(c[3]),.s(sum[3]));

FullAdder fAdd4(.a(sum[3]),.b(andRes[10]),.c\_in(c[5]),.c\_out(c[6]),.s(sum[4]));

FullAdder fAdd5(.a(sum[4]),.b(andRes[13]),.c\_in(c[8]),.c\_out(c[9]),.s(p[4]));

//Sixth Place output

FullAdder fAdd6(.a(andRes[14]),.b(c[3]),.c\_in(c[6]),.c\_out(c[7]),.s(sum[5]));

FullAdder fAdd7(.a(andRes[11]),.b(c[9]),.c\_in(sum[5]),.c\_out(c[10]),.s(p[5]));

//Seventh Place output && Eighth Place output

FullAdder fAdd8(.a(andRes[15]),.b(c[7]),.c\_in(c[10]),.c\_out(p[7]),.s(p[6]));

endmodule

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: multi\_tb.v

--Purpose of Code:test the four by four combinational multiplexer logic

--Project Part Number: Part 1 – C

module multi\_tb;

reg [3:0] x,y;

wire [7:0] p;

//myMulti t1(.x(x),.y(y),.p(p));

initial

begin

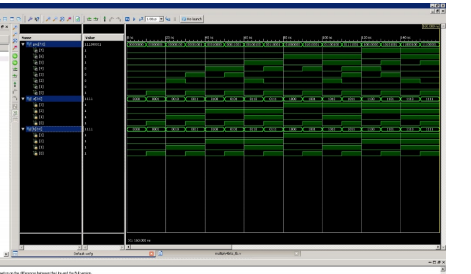
x=4'b1011;

y=4'b1101;

#20 $stop;

end

endmodule



User Constraints File:

##Switches

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports {x[0]}]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports {x[1]}]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports {x[2]}]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports {x[3]}]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { y[0] }]; #IO\_L12N\_T1\_MRCC\_14 Sch=sw[4]

set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { y[1] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { y[2] }]; #IO\_L17N\_T2\_A13\_D29\_14 Sch=sw[6]

set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { y[3] }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

## LEDs

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { p[0] }]; #IO\_L18P\_T2\_A24\_15 Sch=led[0]

set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { p[1] }]; #IO\_L24P\_T3\_RS1\_15 Sch=led[1]

set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { p[2] }]; #IO\_L17N\_T2\_A25\_15 Sch=led[2]

set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { p[3] }]; #IO\_L8P\_T1\_D11\_14 Sch=led[3]

set\_property -dict { PACKAGE\_PIN R18 IOSTANDARD LVCMOS33 } [get\_ports { p[4] }]; #IO\_L7P\_T1\_D09\_14 Sch=led[4]

set\_property -dict { PACKAGE\_PIN V17 IOSTANDARD LVCMOS33 } [get\_ports { p[5] }]; #IO\_L18N\_T2\_A11\_D27\_14 Sch=led[5]

set\_property -dict { PACKAGE\_PIN U17 IOSTANDARD LVCMOS33 } [get\_ports { p[6] }]; #IO\_L17P\_T2\_A14\_D30\_14 Sch=led[6]

set\_property -dict { PACKAGE\_PIN U16 IOSTANDARD LVCMOS33 } [get\_ports { p[7] }]; #IO\_L18P\_T2\_A12\_D28\_14 Sch=led[7]

Results:

In this part of the lab, we were able to get our expected values from the 4 by 4 combination multiplier design. In this part there were not too much trouble except that we had to work slow and methodically. As long as we drew out and wrote on the schematic of what we need we were able to accomplish our goals. In this part, we finished quickly but ended up getting stuck for hours because of a small error in our code, in which after many gruely tries were able to find out what was bugging our code.

# Part 2: 4 by 4 Binary Sequential Multiplier Design

## Source Code:Adder

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: adder.v

--Purpose of Code:Adds register B and A together

--Project Part Number: Part 2a

module adder( a, b, cout, s);

input [3:0] a, b;

output cout;

output [3:0] s;

assign {cout, s} = a+b;

endmodule

## Source Code:D Flip-Flop A

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: dffa.v

--Purpose of Code:Load a number into A register

--Project Part Number: Part 2b

module dffa (clk,clr, load, da, qa);

input clk, clr, load;

input [3:0] da;

output [3:0] qa;

reg [3:0] qa;

always@(posedge clr or posedge clk)

begin

if(clr) qa <= 0;

else if (load)

qa <= da;

end

endmodule

Source Code: D Flip-Flop A Testbench

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: dffa\_tb.v

--Purpose of Code:Test flip flop

--Project Part Number: Part 2b

`timescale 1ns/1ps

module dffa\_tb;

reg clk, clr, load;

reg [3:0] da;

wire [3:0] qa;

dffa uut (.clk(clk), .clr(clr), .load(load), .da(da), .qa(qa));

// dffa uut ( clk, clr, load, da, qa );

initial clk = 0;

always #10 clk = ~ clk;

initial

begin

clr = 1; load = 0;

da = 4'b1011;

#24 clr = 0; load = 1;

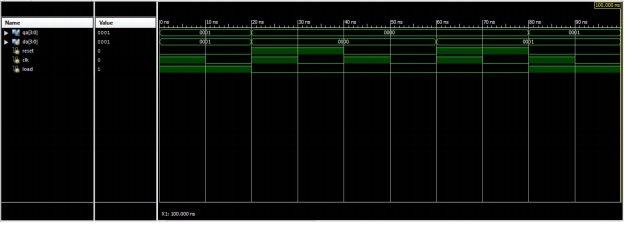
#20 load = 0;

#60 $stop;

end

endmodule

Simulation Waveform:



## Source Code: D Flip-Flop B

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: dffb.v

--Purpose of Code:D flip flop to shift register to the right one bit.

--Project Part Number: Part 2c

module dffb (clk, clr, load, sft, db, qb);

input clk, clr, load, sft;

input [4:0] db;

output [4:0] qb;

reg [4:0] qb;

always@(posedge clr or posedge clk)

begin

if(clr) qb <= 0;

else if (load)

qb <= db;

else if (sft)

//qb <= { 1'b0, qb[3:1] };

begin

qb[5]<= 1'b0;

qb[3] <= qb[4];

qb[2] <= qb[3];

qb[1] <= qb[2];

qb[0] <= qb[1];

end

end

endmodule

## Source Code: FSM

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: fsm.v

--Purpose of Code: Shows the finite state machine and the steps to take in order to load shift add, and clear registers for the sequential multiplier.

--Project Part Number: Part 2d

module fsm(reset, clk, clr, load\_ab, load\_p, shf\_b, shf\_p, en\_add);

input reset, clk;

output clr, load\_ab, load\_p, shf\_b, shf\_p, en\_add;

reg clr, load\_ab, load\_p, shf\_b, shf\_p, en\_add;

reg[4:0] cs, nexts;

parameter s\_clr = 5'b00000,

s\_load\_ab = 5'b00001,

s\_en\_add1 = 5'b00010,

s\_load\_p1 = 5'b00011,

s\_shf\_p1 = 5'b00100,

s\_shf\_b1 = 5'b00101,

s\_en\_add2 = 5'b00110,

s\_load\_p2 = 5'b00111,

s\_shf\_p2 = 5'b01000,

s\_shf\_b2 = 5'b01001,

s\_en\_add3 = 5'b01010,

s\_load\_p3 = 5'b01011,

s\_shf\_p3 = 5'b01100,

s\_shf\_b3 = 5'b01101,

s\_en\_add4 = 5'b01110,

s\_load\_p4 = 5'b01111,

s\_shf\_p4 = 5'b10000,

s\_wait = 5'b10001;

always @ (posedge clk or posedge reset)

begin

if(reset) cs <= s\_clr;

else cs <= nexts;

end

always @ (cs)

begin

case(cs)

s\_clr : nexts = s\_load\_ab;

s\_load\_ab : nexts = s\_en\_add1;

s\_en\_add1 : nexts = s\_load\_p1;

s\_load\_p1 : nexts = s\_shf\_p1;

s\_shf\_p1 : nexts = s\_shf\_b1;

s\_shf\_b1 : nexts = s\_en\_add2;

s\_en\_add2 : nexts = s\_load\_p2;

s\_load\_p2 : nexts = s\_shf\_p2;

s\_shf\_p2 : nexts = s\_shf\_b2;

s\_shf\_b2 : nexts = s\_en\_add3;

s\_en\_add3 : nexts = s\_load\_p3;

s\_load\_p3 : nexts = s\_shf\_p3;

s\_shf\_p3 : nexts = s\_shf\_b3;

s\_shf\_b3 : nexts = s\_en\_add4;

s\_en\_add4 : nexts = s\_load\_p4;

s\_load\_p4 : nexts = s\_shf\_p4;

s\_shf\_p4: nexts = s\_wait;

s\_wait: nexts = s\_wait;

default : nexts = s\_clr;

endcase

end

always @ (cs)

begin

case(cs)

s\_clr:

begin

clr = 1;

load\_ab = 0;

load\_p = 0;

shf\_b = 0;

shf\_p = 0;

en\_add = 0;

end

s\_load\_ab:

begin

clr = 0;

load\_ab = 1;

load\_p = 0;

shf\_b = 0;

shf\_p = 0;

en\_add = 0;

end

s\_en\_add1:

begin

clr = 0;

load\_ab = 0;

load\_p = 0;

shf\_b = 0;

shf\_p = 0;

en\_add = 1;

end

s\_load\_p1:

begin

clr = 0;

load\_ab = 0;

load\_p = 1;

shf\_b = 0;

shf\_p = 0;

en\_add = 0;

end

s\_shf\_p1:

begin

clr = 0;

load\_ab = 0;

load\_p = 0;

shf\_b = 0;

shf\_p = 1;

en\_add = 0;

end

s\_shf\_b1:

begin

clr = 0;

load\_ab = 0;

load\_p = 0;

shf\_b = 1;

shf\_p = 0;

en\_add = 0;

end

s\_en\_add2:

begin

clr = 0;

load\_ab = 0;

load\_p = 0;

shf\_b = 0;

shf\_p = 0;

en\_add = 1;

end

s\_load\_p2:

begin

clr = 0;

load\_ab = 0;

load\_p = 1;

shf\_b = 0;

shf\_p = 0;

en\_add = 0;

end

s\_shf\_p2:

begin

clr = 0;

load\_ab = 0;

load\_p = 0;

shf\_b = 0;

shf\_p = 1;

en\_add = 0;

end

s\_shf\_b2:

begin

clr = 0;

load\_ab = 0;

load\_p = 0;

shf\_b = 1;

shf\_p = 0;

en\_add = 0;

end

s\_en\_add3:

begin

clr = 0;

load\_ab = 0;

load\_p = 0;

shf\_b = 0;

shf\_p = 0;

en\_add = 1;

end

s\_load\_p3:

begin

clr = 0;

load\_ab = 0;

load\_p = 1;

shf\_b = 0;

shf\_p = 0;

en\_add = 0;

end

s\_shf\_p3:

begin

clr = 0;

load\_ab = 0;

load\_p = 0;

shf\_b = 0;

shf\_p = 1;

en\_add = 0;

end

s\_shf\_b3:

begin

clr = 0;

load\_ab = 0;

load\_p = 0;

shf\_b = 1;

shf\_p = 0;

en\_add = 0;

end

s\_en\_add4:

begin

clr = 0;

load\_ab = 0;

load\_p = 0;

shf\_b = 0;

shf\_p = 0;

en\_add = 1;

end

s\_load\_p4:

begin

clr = 0;

load\_ab = 0;

load\_p = 1;

shf\_b = 0;

shf\_p = 0;

en\_add = 0;

end

s\_shf\_p4:

begin

clr = 0;

load\_ab = 0;

load\_p = 0;

shf\_b = 0;

shf\_p = 1;

en\_add = 0;

end

s\_wait:

begin

clr = 0;

load\_ab = 0;

load\_p = 0;

shf\_b = 0;

shf\_p = 0;

en\_add = 0;

end

default:

begin

clr = 0;

load\_ab = 0;

load\_p = 0;

shf\_b = 0;

shf\_p = 0;

en\_add = 0;

end

endcase

end

endmodule

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: fsm\_tb.v

--Purpose of Code: test fsm.v

--Project Part Number: Part 2d

module fsm\_tb;

// Inputs

reg reset;

reg clk;

// Outputs

wire clr;

wire load\_ab;

wire load\_p;

wire shf\_b;

wire shf\_p;

wire en\_add;

// Instantiate the Unit Under Test (UUT)

fsm uut (

.reset(reset),

.clk(clk),

.clr(clr),

.load\_ab(load\_ab),

.load\_p(load\_p),

.shf\_b(shf\_b),

.shf\_p(shf\_p),

.en\_add(en\_add)

);

initial clk = 0;

always #10 clk = ~clk;

initial

begin

reset = 1;

#10;

reset = 0;

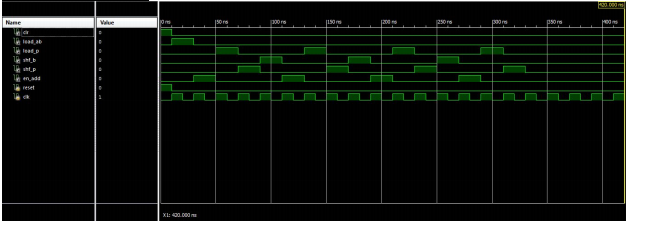
#10;

#400 $stop;

end

endmodule

Simulation Waveform:



## Source code: Mult (Multiplier)

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: mult.v

--Purpose of Code: The design that connects dffa, dffb, mux, adder, and preregister.

--Project Part Number: Part 2e

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 02/14/2018 02:32:01 PM

// Design Name:

// Module Name: mult

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module mult(display,multicand,multiplier,RST,P, CLK,shft\_con,ld1,ld2,loadP,dispA,dispB,shftP,dispMulti,dispMultc);

output [3:0] dispA,dispB,dispMulti,dispMultc;

input [3:0] multicand;

input [4:0] multiplier;

input RST, CLK, shft\_con, ld1,ld2,loadP,shftP;

wire [3:0] j,l,n;

wire [4:0] k;

wire [3:0] m;

wire s;

output [7:0]display;

//assign l = 4'b0111;

assign l = P[7:4];

output [7:0] P;

dffa load1(.clk(CLK),.clr(RST),.da(multicand),.load(ld1),.qa(j));

dffb load2(.clk(CLK),.clr(RST),.db(multiplier),.qb(k),.sft(shft\_con),.load(ld2));

mux choose1(.s(k[0]),.d1(4'b0000),.d0(j),.y(n));

adder add1(.a(l),.b(n),.cout(s),.s(m));

shiftp shift1(.RST(RST),.CLK(CLK),.mostsig(s),.data\_in(m),.shift\_sig(shftP),.loadP(loadP),.prod(P));

assign dispMultc = j;

assign dispMulti = k;

assign dispA = n;

assign dispB = m;

assign display = l;

Endmodule

## Source code: Mux

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: mux.v

--Purpose of Code: Chooses the value for y depending on whether the value is 1 or a 0.

--Project Part Number: Part 2f

module mux ( a, b, y);

input b;

input[3:0] a;

wire[3:0] z;

assign z = 4'b0000;

output[3:0] y;

reg[3:0] y;

always @ (a or b or z)

begin

case(b)

0 : y = z;

1 : y = a;

endcase

end

endmodule

Source code: P-Register

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: preg.v

--Purpose of Code: The p-register stores the value of the multiplier.

--Project Part Number: Part 2g

module preg(clk, reset, load, shift, cin, din\_p, p, ph);

input clk, reset, load, shift, cin;

input[3:0] din\_p;

output[7:0] p;

output[3:0] ph;

reg[3:0] ph, pl;

always @ (posedge clk or posedge reset)

begin

if (reset)

ph <= 4'h0;

else if (load)

ph <= din\_p;

else if (shift)

ph <= { cin , ph[3:1] };

end

always @ (posedge clk or posedge reset)

begin

if (reset)

pl <= 4'h0;

else if (shift)

pl <= { ph[0] , pl[3:1] };

end

assign p = {ph, pl};

endmodule

## Source code: Top Module

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: top.v

--Purpose of Code: The top layer that combines all the code into one top module.

--Project Part Number: Part 2h

module top(a, b, p, clk, clr);

/\*

testx, testy, testm, testn, testz, testr

\*/

input[3:0] a, b;

input clk, clr;

output[7:0] p;

wire j, k, l, m, n, o, q;

//create one instance for finite state machine

fsm f1(.reset(clr), .clk(clk), .clr(q), .load\_ab(j), .load\_p(k), .shf\_b(m), .shf\_p(n), .en\_add(o));

//create one instance for multiplier logic circuit

mult m1(.a(a), .b(b), .p(p), .reset(q), .clk(clk), .load\_ab(j), .load\_p(k), .shf\_b(m), .shf\_p(n), .en\_add(o) );

endmodule

User Constraints File: 4 by 4 Binary Sequential Multiplier Design

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clk }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { a[0]}]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { a[1]}]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { a[2] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { a[3] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { p[0] }]; #IO\_L18P\_T2\_A24\_15 Sch=led[0]  
set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { p[1]}]; #IO\_L24P\_T3\_RS1\_15 Sch=led[1]  
set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { p[2] }]; #IO\_L17N\_T2\_A25\_15 Sch=led[2]  
set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { p[3] }]; #IO\_L8P\_T1\_D11\_14 Sch=led[3]  
set\_property -dict { PACKAGE\_PIN R18 IOSTANDARD LVCMOS33 } [get\_ports { p[4]}]; #IO\_L7P\_T1\_D09\_14 Sch=led[4]  
set\_property -dict { PACKAGE\_PIN V17 IOSTANDARD LVCMOS33 } [get\_ports { p[5] }]; #IO\_L18N\_T2\_A11\_D27\_14 Sch=led[5]  
set\_property -dict { PACKAGE\_PIN U17 IOSTANDARD LVCMOS33 } [get\_ports { p[6] }]; #IO\_L17P\_T2\_A14\_D30\_14 Sch=led[6]  
set\_property -dict { PACKAGE\_PIN U16 IOSTANDARD LVCMOS33 } [get\_ports { p[7] }]; #IO\_L18P\_T2\_A12\_D28\_14 Sch=led[7]

set\_property -dict { PACKAGE\_PIN N17 IOSTANDARD LVCMOS33 } [get\_ports { clr }]; #IO\_L9P\_T1\_DQS\_14 Sch=btnc

Results:

By far our most frustrating and longest part of the lab. Because we got stuck on this lab, it took us almost the whole semester to complete. With this part of the lab, we got very close but just couldn’t find what was wrong with it. Because it is a multi layer design, troubleshooting our code was extremely tedious. We were even getting errors in areas even from codes that were provided to us from the professor. In part 2, we were having trouble with the sequential multiplier not registering our values correctly. We had the shift registers and everything working great but were being off one or two values. We found that it had to do with pre registering the values inside of P. To overcome this problem, we made a separate design and incorporated it into our top level to pre register our values.

# Part 3: LCD

## Source Code: LCD

--Author: Xue Eric Lee

--Date: 4/10/2018

--File Name: fpga\_proj.v

--Purpose of Code: The design gives a signal to the LCD to perform the display of cpe 166.

--Project Part Number: Part 3

module fpga\_proj( clk, seg, dig);

input clk;

output [7:0] seg;

output [7:0] dig;

parameter N = 18;

reg [N-1:0] count;

reg [3:0] dd;

reg [7:0] seg;

reg [7:0] an;

always @ (posedge clk)

begin

count <= count + 1;

case(count[N-1:N-3])

3'b000 :

begin

dd = 4'd7;

an = 8'b11111110;

end

3'b001:

begin

dd = 4'd6;

an = 8'b11111101;

end

3'b010:

begin

dd = 4'd5;

an = 8'b11111011;

end

3'b011:

begin

dd = 4'd4;

an = 8'b11110111;

end

3'b100 :

begin

dd = 4'd3;

an = 8'b11101111;

end

3'b101:

begin

dd = 4'd2;

an = 8'b11011111;

end

3'b110:

begin

dd = 4'd1;

an = 8'b10111111;

end

3'b111:

begin

dd = 4'd0;

an = 8'b01111111;

end

endcase

end

assign dig = an;

always @ (dd)

begin

seg[7] = 1'b1;

case(dd)

4'd0 : seg[6:0] = 7'b1000110; //to display C

4'd1 : seg[6:0] = 7'b0001100; //to display P

4'd2 : seg[6:0] = 7'b0000110; //to display E

4'd3 : seg[6:0] = 7'b1111001; //to display 1

4'd4 : seg[6:0] = 7'b0000010; //to display 6

4'd5 : seg[6:0] = 7'b0000010; //to display 6

4'd6 : seg[6:0] = 7'b0001001; //to display X

4'd7 : seg[6:0] = 7'b1000111; //to display L

default : seg[6:0] = 7'b1111111; //blank

endcase

end

endmodule

User Constraints File: LCD

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 }];

## 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]

Results:

In this part of the lab, we had to display CPE 166 onto the LCD board. We were able to achieve with not very much problems as we were supplied a template from the professor and we just had to change up a few things and then write the constraints file for the board. It was actually pretty easy and did not take much time to accomplish this part of the lab.

# Conclusion:

This lab overall served as an introduction to the Verilog programming language. The code required making logic gate assignments, creating time dependent modules such as d flip flops, linking modules together in hierarchical design and using case statements to create a finite state machine. Test benches for all the modules also needed to be designed in Verilog. Inputs and outputs needed to be declared in a certain way, the tested module needed to be called in a certain fashion. For the sequential multiplier, a clock needed to be made for the simulation, as well as learning how to make the simulation wait certain periods before assigning more inputs. The concepts of the combinational design versus sequential were also explored. The advantage of a combinational design is that is does not require a clock, but they can also be messy to organize. The advantage of a sequential design is that while they require a clock, managing the connected modules is an easier task.