Grace Pope

CPE 166 – 03
Advanced Logic Design Lab
Monday: 2 PM - 4:40 PM
Lab 2 Report

Contents

Introduction

Lab 2 was a three part lab that served as an introduction to hierarchical design in Verilog. We were tasked with designing a multiplier, a 4-bit multiplier, and using the 7-segment displays to display a message. The first two parts were designing a multiplier using two different methods. The final part was an introduction to uploading the code onto the fpga. It was a simple tutorial where we did not need to change many parts so we could learn how to use the Xilinx Vivado program and the fpga.

Part 1 – Binary Combinational Multiplier

Design Purpose

This lab was designed to make a multiplying module that could consist of many full adders and half adders and other combinational logic. The diagram was given to us and we needed to design the individual modules.

Engineering Data

For this lab, I designed the multiplier according to the schematic seen below. It involved using 16 and gates, 4 half adders, and 8 full adders. Both the full adders and the half adders had carry out and sum values that needed to be wires in the Verilog design. This multiplier was not very complex, but it is made up of many different components. The half adder is composed of one and gate and one xor gate. The outputs are the sum and the carry. The full adder connects two half adders and ors the answer to get the sum. The outputs of the full adder module are also the sum and the carry.

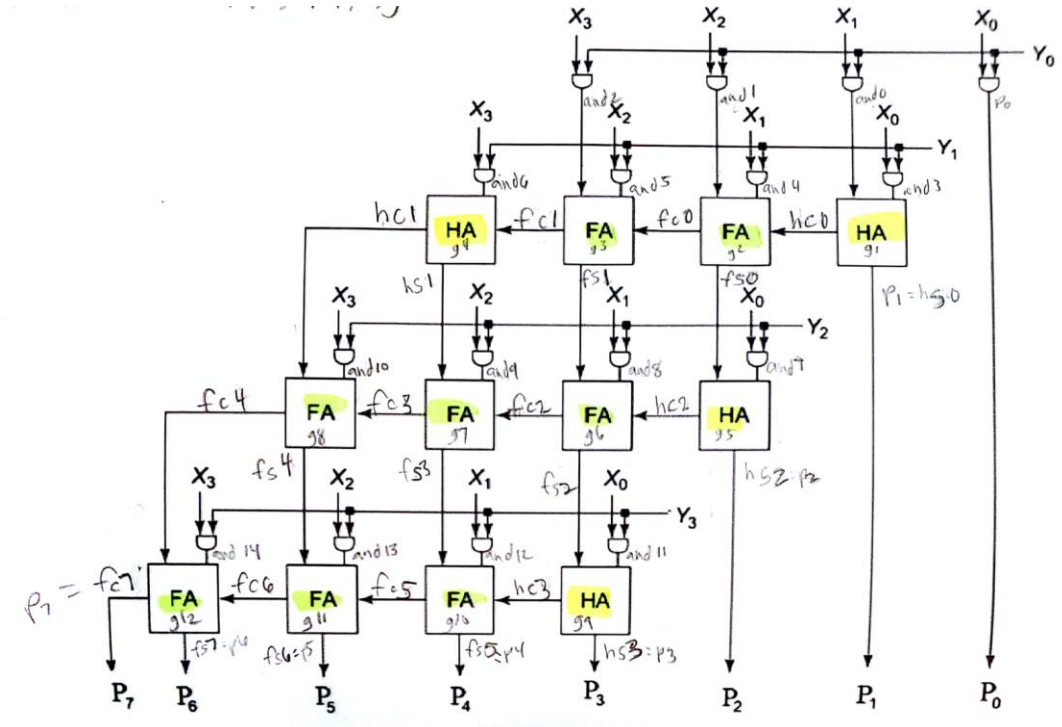

Figure 1: Diagram used to design the combinational multiplier

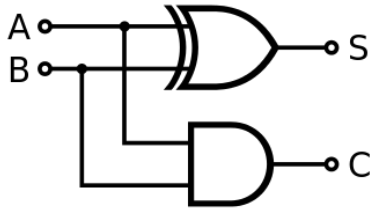

Figure 2: Half Adder schematic

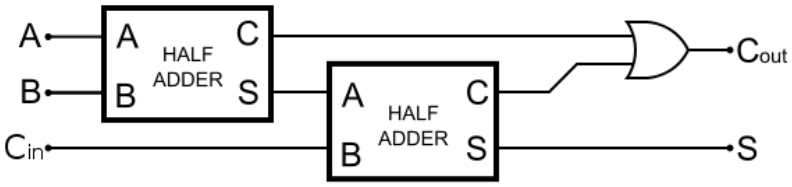

Figure 3: Full adder schematic using half adder modules

Source Code

```
//----------------------------------------------------
//-- Author : Grace Pope
//-- Date : 9/10/18
//-- File Name :multiply.v
//-- Purpose of Code : multiply two 4 bit number
//-- Project Part Number : Lab 3 part 1
//----------------------------------------------------
`timescale 1ns / 1ps
module multiply(x,y,p);
input [3:0] x, y;
output [7:0] p;
wire [14:0] ad;
wire [7:0] fs, fc;
wire [7:0] hs, hc;
assign p[0]  = x[0] & y[0];
assign ad[0] = x[1] & y[0];
assign ad[1] = x[2] & y[0];
assign ad[2] = x[3] & y[0];
assign ad[3] = x[0] & y[1];
assign ad[4] = x[1] & y[1];
assign ad[5] = x[2] & y[1];
assign ad[6] = x[3] & y[1];
assign ad[7]  = x[0] & y[2];
assign ad[8]  = x[1] & y[2];
assign ad[9]  = x[2] & y[2];
assign ad[10] = x[3] & y[2];
assign ad[11] = x[0] & y[3];
assign ad[12] = x[1] & y[3];
assign ad[13] = x[2] & y[3];
assign ad[14] = x[3] & y[3];
ha g1(ad[3], ad[0], hc[0], hs[0]);
assign p[1] = hs[0];
fa g2(ad[4], ad[1], hc[0], fs[0], fc[0]);
fa g3(ad[5], ad[2], fc[0], fs[1], fc[1]);
ha g4(ad[6], fc[1], hc[1], hs[1]);
ha g5(ad[7], fs[0], hc[2], hs[2]);
assign p[2] = hs[2];
fa g6(ad[8], fs[1], hc[2], fs[2], fc[2]);
```

```
//----------------------------------------------------
//-- Author : Grace Pope
//-- Date : 9/10/18
//-- File Name :multiply_tb.v
//-- Purpose of Code : test bench for multiply
module
//-- Project Part Number : Lab 3 part 1
//----------------------------------------------------
`timescale 1ns / 1ps
module multiply_tb;
reg [3:0] x;
reg [3:0] y;
wire [7:0] p;
multiply uut(x,y,p);
initial
begin
x  = 4'b0000; y = 4'b1111;
        #20 x  = 4'b0001; y = 4'b1000;
        #20 x  = 4'b0001; y = 4'b1100;
        #20 x  = 4'b0010; y = 4'b0100;
        #20 x  = 4'b1111; y = 4'b1100;
        #20 x  = 4'b1011; y = 4'b1111;
        #20 x  = 4'b1111; y = 4'b1111;
  #20 $stop;
end
endmodule
```

```
fa g7(ad[9], hs[1], fc[2], fs[3], fc[3]);
fa g8(ad[10], hc[1], fc[3], fs[4], fc[4]);
ha g9(ad[11], fs[2], hc[3], hs[3]);
assign p[3] = hs[3];
fa g10(ad[12], fs[3], hc[3], fs[5], fc[5]);
assign p[4] = fs[5];
fa g11(ad[13], fs[4], fc[5], fs[6], fc[6]);
assign p[5] = fs[6];
fa g12(ad[14], fc[4], fc[6], fs[7], fc[7]);
assign p[6] = fs[7];
assign p[7] = fc[7];
endmodule
```

```
//-------------------------------------------------
//-- Author : Grace Pope
//-- Date :9/10/18
//-- File Name :ha.v
//-- Purpose of Code : half adder
//-- Project Part Number : Lab 3 part 1
//-------------------------------------------------
module ha(a,b,c,s);
  input a,b;
  output c,s;
  assign c=a&b;
  assign s=a^b;
endmodule
```

```
//-------------------------------------------------
//-- Author : Grace Pope
//-- Date :9/10/18
//-- File Name :ha_tb.v
//-- Purpose of Code : half adder test bench
//-- Project Part Number : Lab 3 part 1
//-------------------------------------------------
`timescale 1ns / 1ps
module ha_tb
   reg a,b;
   wire c,s;
   ha uut(a,b,c,s);
   initial
   begin
   a=0; b=0;
   #20 a=0; b=1;
   #20 a=1; b=0;
   #20 a=1; b=1;
   #20
   $stop;
   end
endmodule
```

```
//-------------------------------------------------
//-- Author : Grace Pope
//-- Date : 9/10/18
//-- File Name :fa.v
//-- Purpose of Code : full adder with carry out
and sum
//-- Project Part Number : Lab 3 part 1
//-------------------------------------------------
module fa(a,b,cin,s,cout);
  input a,b,cin;
  output cout,s;
  wire m,n,p;
  ha g1(a,b,n,m);
  ha g2(.a(cin), .b(m), .s(s), .c(p));
```

```
//-------------------------------------------------
//-- Author : Grace Pope
//-- Date : 9/10/18
//-- File Name :fa_tb.v
//-- Purpose of Code : full adder test bench
//-- Project Part Number : Lab 3 part 1
//-------------------------------------------------
`timescale 1ns / 1ps
module fa_tb( );
  reg a,b,cin;
  wire s,cout;
  fa uut(a,b,cin,s,cout);
  initial
  begin
```

```
  assign cout=p|n;
endmodule
```

```
    a=1'b0; b=1'b0; cin=1'b0;
    #20 a=1'b0; b=1'b0; cin=1'b1;
    #20 a=1'b0; b=1'b1; cin=1'b0;
    #20 a=1'b0; b=1'b1; cin=1'b1;
    #20 a=1'b1; b=1'b0; cin=1'b0;
    #20 a=1'b1; b=1'b0; cin=1'b1;
    #20 a=1'b1; b=1'b1; cin=1'b0;
    #20 a=1'b1; b=1'b1; cin=1'b1;
    #20
    $stop;
  end
endmodule
```

Simulation Waveforms

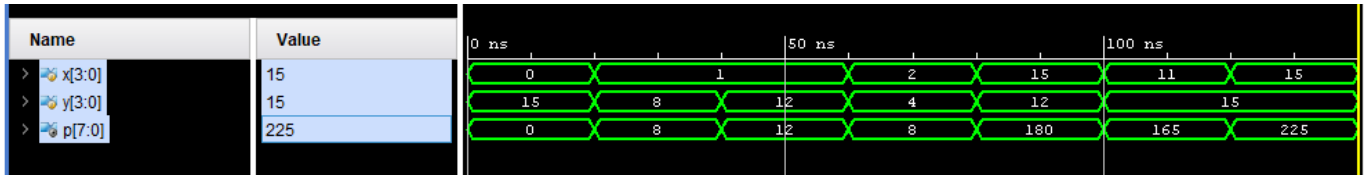

Figure 4: Combinational Multiplier Simulation results

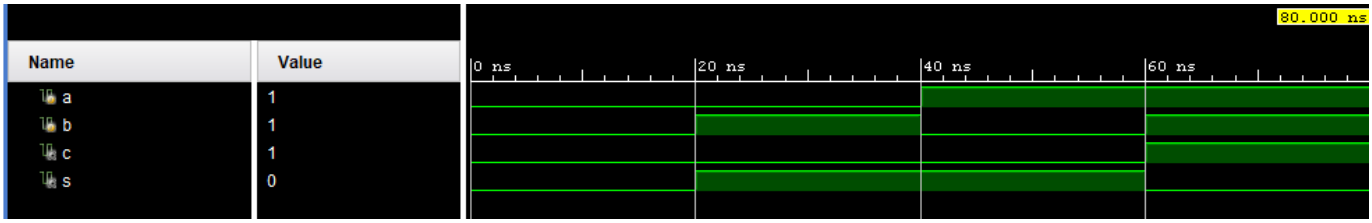

Figure 5: Half adder simulation results

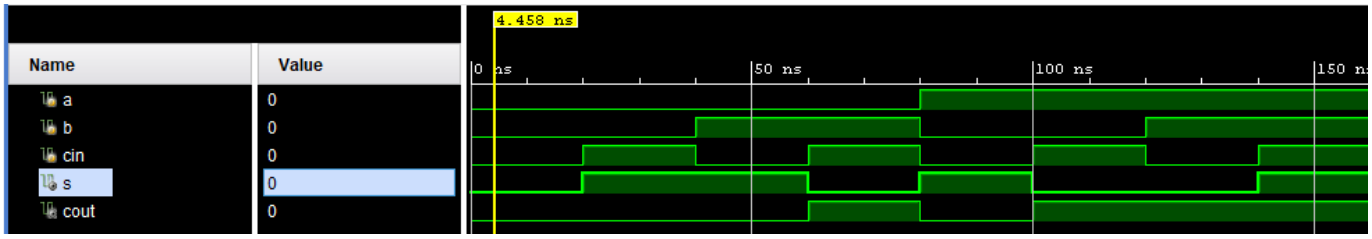

Figure 6: Full adder simulation results

Results Discussion

For this part, I was able to demonstrate two numbers being multiplied. In the simulation, the two numbers being multiplied are immediately shown on the output. It works as expected. The most difficult part was making sure all of the wire names were declared and making sure they were connected to the right node on the top level module.

Part 2 – Binary Sequential Multiplier

Design Purpose

This multiplier was to be designed using a shift multiplier. This was much more difficult than doing the sequential multiplier because it required the use of a state machine to control the inputs, and because of the shifting required to create the multiplier.

Engineering Data

This multiplier required many different components that were connected all together in the top level module. The top module consisted of the finite state machine for the controller and the multiplier which consisted of all of the other modules. The multiplier module was designed using figure 7 as a reference, and it consisted of the register, shift register, multiplexor, multiplier, and the full adder. Together, these modules will shift and multiply the inputs. This module is controlled by the state machine outputs, and the state machine is controlled by the top level module. The top level module is connected to the multiplier module which will do the multiplication of the two numbers. The full adder, multiplexor, register, and shift register were given to us from previous projects, so the only new modules we needed to design were the multiplier and finite state machine.

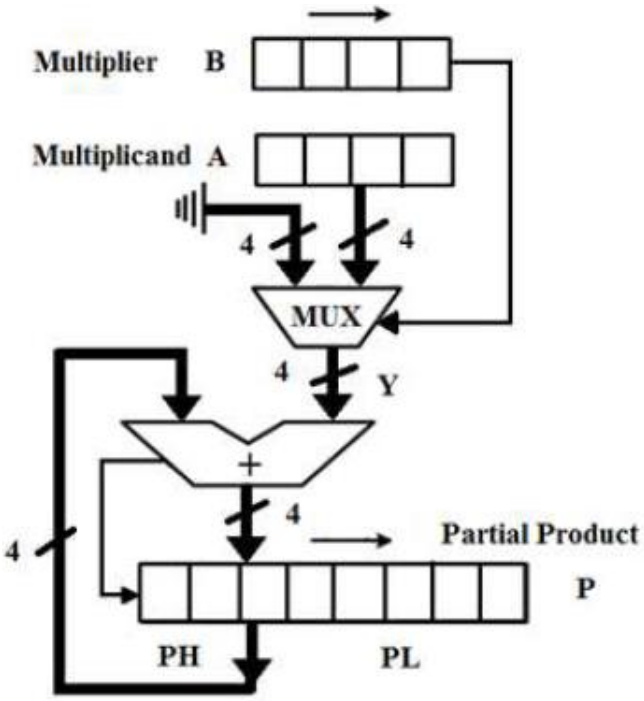

Figure 7: Schematic for sequential multiplier

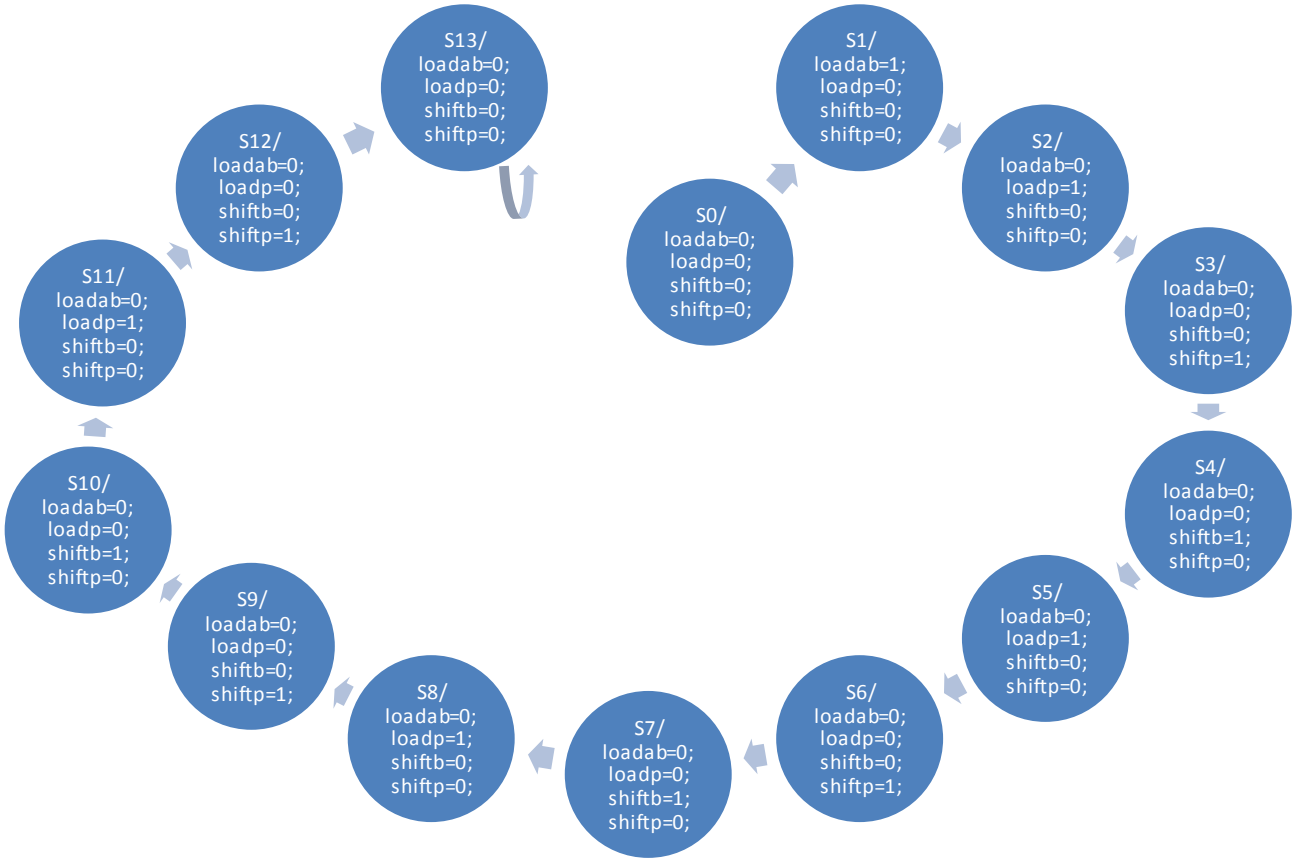

Figure 8: State diagram for lab 2 part 2

Source Code

```
//----------------------------------------------------
//-- Author : Grace Pope
//-- Date : 10/31/18
//-- File Name :top.v
//-- Purpose of Code : top level module with inputs and outputs for fpga
//-- Project Part Number : Lab 3 part 2
//--Hardware FPGA/CPLD Device : xc7a100tcsg324-1
//----------------------------------------------------
`timescale 1ns / 1ps
module top(a,b,reset,clk,p);
   input reset,clk;
   input [3:0] a,b;
   output [7:0] p;
   wire loadab, loadp, shiftb, shiftp;
   fsm uut(.clk(clk), .reset(reset), .loadab(loadab), .loadp(loadp), .shiftb(shiftb), .shiftp(shiftp));
   multiplier uut2(.clk(clk), .reset(reset), .loadab(loadab), .loadp(loadp), .shiftb(shiftb), .shift p(shiftp),
.a(a), .b(b), .p(p));
  endmodule
//----------------------------------------------------
```

```
//-- Author : Grace Pope
//-- Date : 10/31/18
//-- File Name :fsm.v
//-- Purpose of Code : finite state machine
//-- Project Part Number : Lab 3 part 2
//--Hardware FPGA/CPLD Device : xc7a100tcsg324-1
//----------------------------------------------------
module fsm(clk, reset, loadab, loadp, shiftb, shiftp);
   input reset, clk;
   output reg loadab, loadp, shiftb,shiftp;
   wire y;
   reg [3:0] cs, ns;
   parameter s0=4'b0000, s1=4'b0001, s2=4'b0010, s3=4'b0011,
          s4=4'b0100, s5=4'b0101, s6=4'b0110, s7=4'b0111,
          s8=4'b1000, s9=4'b1001, s10=4'b1010, s11=4'b1011,
          s12=4'b1100, s13=4'b1101;
  always @(posedge reset or posedge clk)
  begin
   if(reset)
       cs<=s0;
   else
       cs<=ns;
  end
  always @ (*)
    begin
     case(cs)
      s0:
         begin
         loadab=0; loadp =0; shiftb=0; shiftp=0;
         end
      s1:
         begin
         loadab=1; loadp= 0; shiftb=0; shiftp=0;
         end
      s2:
         begin
         loadab=0; loadp= 1; shiftb=0; shiftp=0;
         end
       s3:
         begin
         loadab=0; loadp=0; shiftb=0; shiftp=1;
         end
       s4:
         begin
         loadab=0; loadp= 0; shiftb=1; shiftp=0;
         end
       s5:
         begin
```

```
        loadab=0; loadp= 1; shiftb=0; shiftp=0;
        end
      s6:
        begin
        loadab=0; loadp= 0; shiftb=0; shiftp=1;
        end
      s7:
        begin
        loadab=0; loadp= 0; shiftb=1; shiftp=0;
        end
      s8:
        begin
        loadab=0; loadp= 1; shiftb=0; shiftp=0;
        end
      s9:
        begin
        loadab=0; loadp= 0; shiftb=0; shiftp=1;
        end
      s10:
        begin
        loadab=0; loadp= 0; shiftb=1; shiftp=0;
        end
      s11:
        begin
        loadab=0; loadp= 1; shiftb=0; shiftp=0;
        end
      s12:
        begin
        loadab=0; loadp= 0; shiftb=0; shiftp=1;
       end
      s13:
        begin
        loadab=0; loadp= 0; shiftb=0; shiftp=0;
        end
     default:
        begin
        loadab=0; loadp= 0; shiftb=0; shiftp=0;
        end
    endcase
   end
always @ (*)
begin
 case(cs)
    s0: ns<=s1;
    s1: ns<=s2;
    s2: ns<=s3;
    s3: ns<=s4;
    s4: ns<=s5;
```

```
      s5: ns<=s6;
      s6: ns<=s7;
      s7: ns<=s8;
      s8: ns<=s9;
      s9: ns<=s10;
      s10: ns<=s11;
      s11: ns<=s12;
      s12: ns<=s13;
      s13: ns<=s13;
      default: ns<=s0;
   endcase
  end
endmodule
```

```
//----------------------------------------------------
//-- Author : Grace Pope
//-- Date : 10/31/18
//-- File Name :multiplier.v
//-- Purpose of Code : secondary level module to multiply two 4 bit numbers
//-- Project Part Number : Lab 3 part 2
//--Hardware FPGA/CPLD Device : xc7a100tcsg324-1
//----------------------------------------------------
module multiplier(clk, reset, loadab, loadp, shiftb, shiftp, a, b, p);
   input reset, clk, loadab, loadp, shiftb, shiftp;
   input [3:0] a,b;
   output [7:0] p;
   wire  cout;
   wire [3:0] ph, pl, d1,  y, qa, da, db, qb, datain;
   dffa uut (.clk(clk), .clr(reset), .load(loadab), .da(a), .qa(qa));
   dffb uut2(.clk(clk), .clr(reset), .load(loadab), .sft(shiftb), .db(b), .qb(qb));
   mux  uut3(.s(qb[0]), .d1(qa), .y(y) );//mux ( s, d1, y );
   mult uut4(.clk(clk), .reset(reset), .load(loadp), .shift(shiftp), .datain(datain), .p(p), .ph(ph), .cout(cout));
   fa   uut5(.a(y), .b(ph), .cout(cout), .sum(datain));
endmodule
```

```
//----------------------------------------------------
//-- Author : Grace Pope
//-- Date : 10/31/18
//-- File Name :dffa.v
//-- Purpose of Code : 4 bit d flip flop with clear
//-- Project Part Number : Lab 3 part 2
//--Hardware FPGA/CPLD Device : xc7a100tcsg324-1
//----------------------------------------------------
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
```

```
//----------------------------------------------------
//-- Author : Grace Pope
//-- Date : 10/31/18
//-- File Name :dffb.v
//-- Purpose of Code : 4 bit shift register with load and clear
//-- Project Part Number : Lab 3 part 2
//--Hardware FPGA/CPLD Device : xc7a100tcsg324-1
//----------------------------------------------------
module dffb (clk, clr, load, sft, db, qb);
input       clk, clr, load, sft;
input [3:0]  db;
output [3:0] qb;
reg    [3:0] 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] };
end
endmodule
```

```
//----------------------------------------------------
//-- Author : Grace Pope
//-- Date : 10/31/18
//-- File Name :mux.v
//-- Purpose of Code : Multiplexor
//-- Project Part Number : Lab 3 part 2
//--Hardware FPGA/CPLD Device : xc7a100tcsg324-1
//----------------------------------------------------
module mux ( s, d1,  y );
input [3:0] d1;
input       s;
output [3:0] y;
reg    [3:0] y;
always@( s or d1 )
begin
  if (s)//(s=1)
        y = d1;
  else
        y <= 0;
end
endmodule
```

```
//----------------------------------------------------
//-- Author : Grace Pope
//-- Date : 10/31/18
//-- File Name :mult.v
//-- Purpose of Code : gets data and shifts it
//-- Project Part Number : Lab 3 part 2
//--Hardware FPGA/CPLD Device : xc7a100tcsg324-1
//----------------------------------------------------
module mult(clk,reset,load,shift,datain,p,ph,cout);
	input clk,reset,load,shift,cout;
	input [3:0] datain;
	output [7:0] p;
	output[3:0] ph;
	reg [8:0] tmp;
	always @(posedge reset or posedge clk)
	begin
		if(reset) tmp<=9'b000000000;
		else if(load)
		begin
			tmp[7:4]<=datain;
			tmp[8]<=cout;
		end
		else if(shift) tmp<={tmp[8], tmp[7:1]};
	end
	assign p=tmp[7:0];
  assign ph=tmp[7:4];
endmodule
```

```
//----------------------------------------------------
//-- Author : Grace Pope
//-- Date : 10/31/18
//-- File Name :fa.v
//-- Purpose of Code : adds two numbers
//-- Project Part Number : Lab 3 part 2
//--Hardware FPGA/CPLD Device : xc7a100tcsg324-1
//----------------------------------------------------
module fa(a, b, cout, sum);
input [3:0] a, b;
output     cout;
output [3:0] sum;
assign {cout, sum} = a + b;
endmodule
```

```
//----------------------------------------------------
//-- Author : Grace Pope
//-- Date : 10/31/18
//-- File Name :top_test.v
//-- Purpose of Code : test bench for the top level module
//-- Project Part Number : Lab 3 part 2
//--Hardware FPGA/CPLD Device : xc7a100tcsg324-1
```

```
//-------------------------------------------------
`timescale 1ns / 1ps
module top_test;
reg reset, clk;
reg [3:0] a,b;
wire [7:0] p;
top uut (a,b,reset,clk,p);
initial  clk = 0;
always  #5 clk = ~ clk;
initial
begin
        reset = 0; a = 4'b1011;  b = 4'b1001;
  #20   reset = 1;
  #20   reset = 0; a = 4'b0010;  b = 4'b0011;
  #140  reset = 1;
  #20   reset = 0; a = 4'b1010;  b = 4'b0110;
  #140  reset = 1;
  #20   reset = 0; a = 4'b1111;  b = 4'b1110;
  #200  $stop;
end
endmodule
```

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 { reset }];
#IO_L24N_T3_RS0_15  Sch=sw[0]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { b[0] }]; #IO_L24N_T3_34
Sch=sw[8]
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { b[1] }]; #IO_25_34
Sch=sw[9]
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { b[2] }];
#IO_L15P_T2_DQS_RDWR_B_14  Sch=sw[10]
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { b[3] }];
#IO_L23P_T3_A03_D19_14  Sch=sw[11]
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { a[0] }]; #IO_L24P_T3_35
Sch=sw[12]
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { a[1] }];
#IO_L20P_T3_A08_D24_14  Sch=sw[13]
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { a[2] }];
#IO_L19N_T3_A09_D25_VREF_14  Sch=sw[14]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { a[3] }];
```

```
#IO_L21P_T3_DQS_14  Sch=sw[15]

## 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]
```

Simulation Waveforms

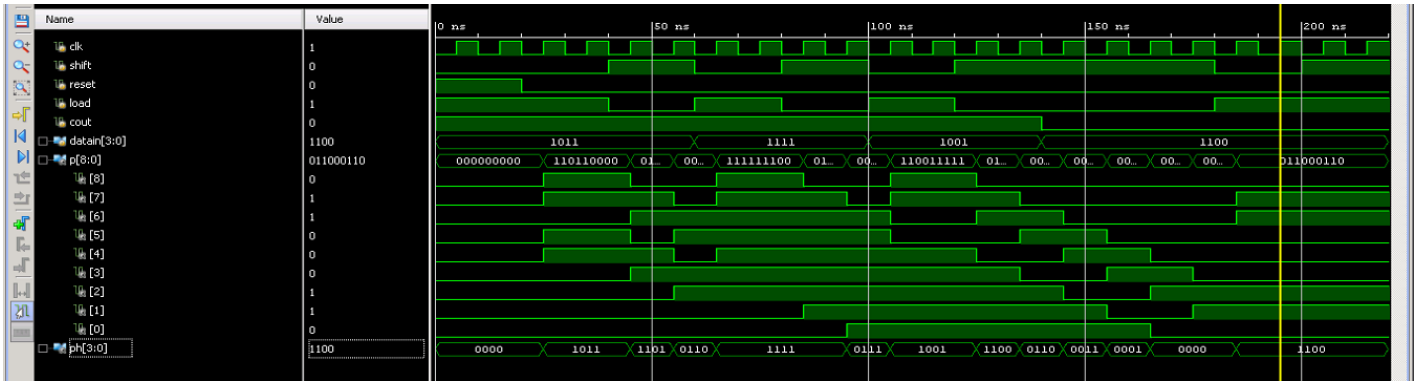

Figure 9: Simulation results from Multiplier module

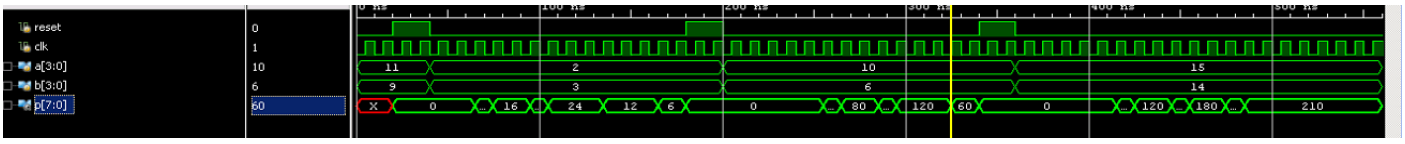

Figure 10: Simulation results from top module

Results Discussion

This lab was difficult because it involved many different modules working together to get the product of two numbers. The module works as expected, and all of the modules within the top module also work as expected. One of the hardest parts of this lab was figuring out how to do the shifting and how it

would be added to the other modules. It was also difficult to determine the number of states needed and how they would interact with the multiplier module.

Part 3 – Display Message using 7 Segment Display

Design Purpose

This part of the lab was designed for us to learn how to display things on the seven segment displays and to learn how to upload a program to the FPGA.

Engineering Data

This lab was mostly given to us. The code to display the numbers and letters was given to us and all we needed to change were what letters were used. The binary representation of the numbers and letters were determined using the picture in figure 11. To upload to the fpga, we needed to generate a bitstream and program the device with the bit stream we just generated. Once it is downloaded, the fpga is programmed with the program we just created.

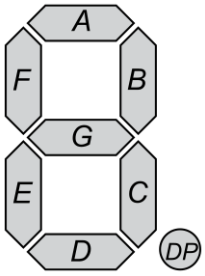

Figure 11: Individual segments of a 7 segment display

Source Code

```
//----------------------------------------------------
//-- Author : Grace Pope
//-- Date : 9/24/18
//-- File Name :demo.v
//-- Purpose of Code : display CPE166GP on Fpga
//-- Project Part Number : Lab 3 part 3
//--Hardware FPGA/CPLD Device : xc7a100tcsg324-1
//----------------------------------------------------
`timescale 1ns / 1ps
module demo( 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'b0000010;  //to display G
   4'd7 : seg[6:0] = 7'b0001100;  //to display P
   default : seg[6:0] = 7'b1111111;  //blank
  endcase
 end
```

User Constraint File

```
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 }];

##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]
```

Results Discussion

During this lab, I was able to display CPE1666P on the FPGA board. That basically translates to CPE 166 GP, which is the class name and my initials. I forgot to take a picture of my board when it was still displaying my initials, but the person who I shared the FPGA board with, Danny Yang, took a picture, displayed below. This lab had no other parts, other than displaying words and numbers on the seven segments, so it was simple compared to the earlier parts of the lab.

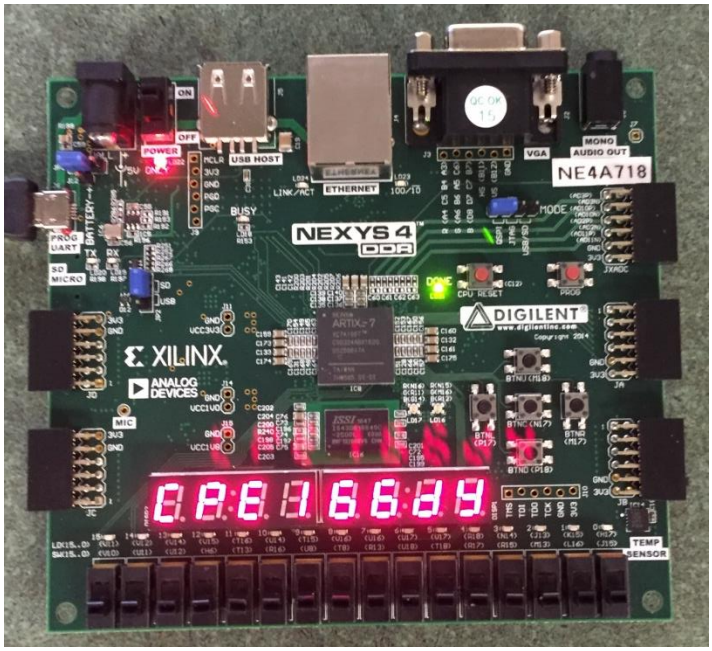

Figure 12: FPGA displaying CPE166dY

Conclusion

Overall, this lab was simple, but difficult. The first and third parts were easy because the diagrams or the Verilog code was given to us. But the second part of this lab was very difficult because shift registers are a harder concept for me to understand. I know there are better ways to solve part 2 of this lab, but this way seemed the easiest for me to do. This was a good lab because it showed us how to design using sequential logic and combinational logic to do the same function. The combinational multiplier was easier to design, but the sequential multiplier seems like it would be easier to scale to use larger numbers. The most enjoyable part of this lab was the third part because it allowed me to see my name on the fpga.