Lab 02

4-bit ALU

	Ryan Cruz	**Zachary Davis**
Category	ryan.cruz25@uga.edu	zachdav@uga.edu

Pre-lab	50	50
In-lab Module & Testbench Design	50	50
In-lab Testbench Sim. & Analysis	50	50
In-lab FPGA Synthesis & Analysis	50	50
Lab Report Writing	50	50

January 25, 2018

Contents

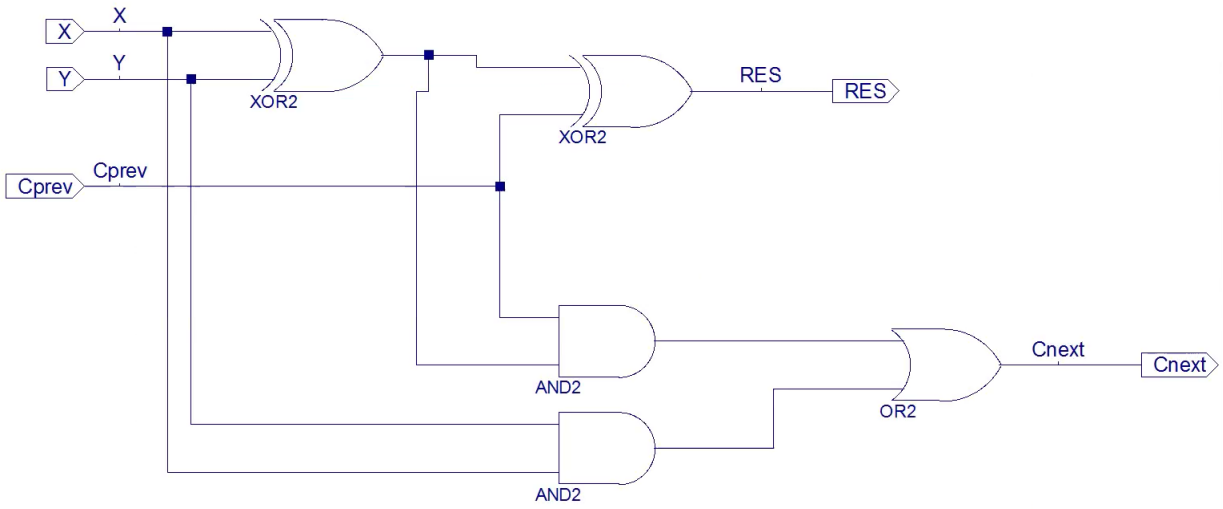
1 Lab Purpose

The purpose of this lab is to create a 4-bit ALU using the schematic method in Xilinx. This will be our first full project that involves creating multiple schematic modules that will eventually be compiled to create a top module that can be implemented on the board. We will design the basic parts of an ALU, including an adder/subtractor, a logic extender, an arithmetic extender, and eventually piece it all together with a UCF and seven segment display driver that will allow us to use this on the board.

2 Implementation Details

2.1 Part 0

We began by building a full adder, the first basic component of the ALU.

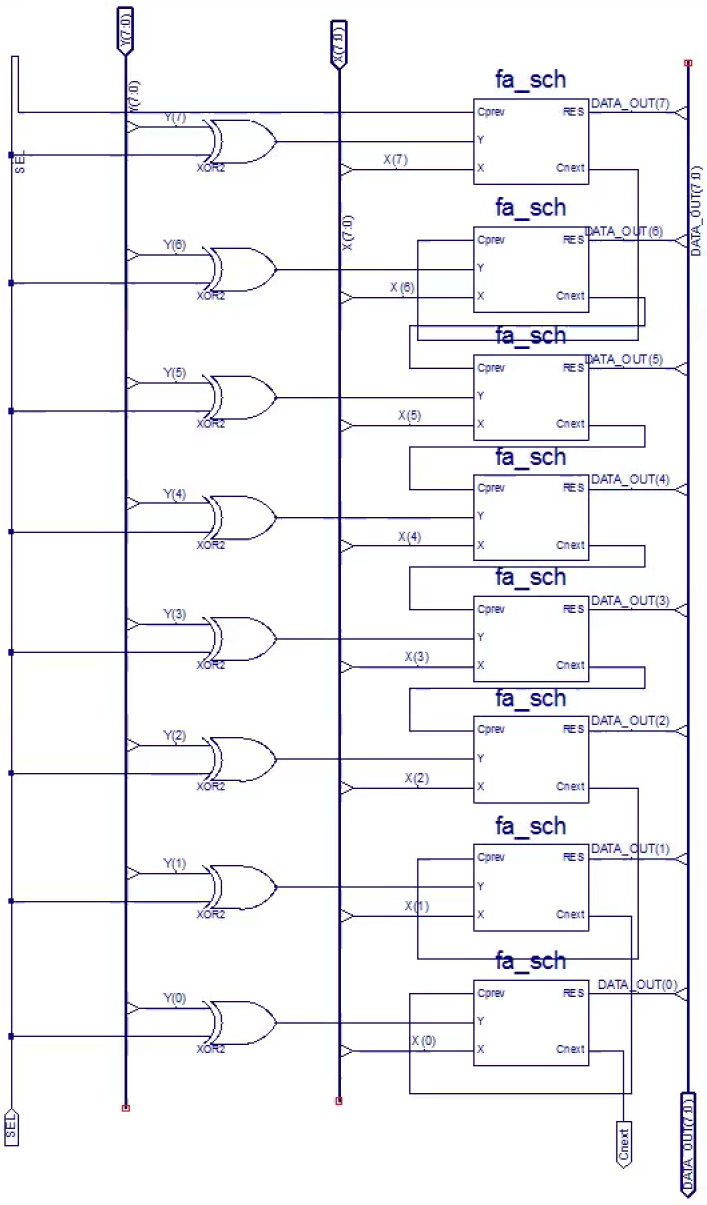



```
‘timescale 1ns/1ps

module fa_tbw_tb_0;
    reg Cprev = 1’b0;
    reg X = 1’b0;
    reg Y = 1’b0;
    wire Cnext;
    wire RES;

fa_sch UUT(
.Cprev(Cprev),
.X(X),
.Y(Y),
```

```
.Cnext(Cnext),
.RES(RES));

initial begin
#100;

//CASE 1
X=0;
Y=0;
Cprev=0;
#100;

//CASE 2
X=0;
Y=0;
Cprev=1;
#100;

//CASE 3
X=0;
Y=1;
Cprev=0;
#100;

//CASE 4
X=0;
Y=1;
Cprev=1;
#100;

//CASE 5
X=1;
Y=0;
Cprev=0;
#100;

//CASE 6
X=1;
Y=0;
Cprev=1;
#100;

//CASE 7
X=1;
Y=1;
Cprev=0;
#100;
```

```
//CASE 8
X=1;
Y=1;
Cprev=1;
#100;

end

endmodule
```

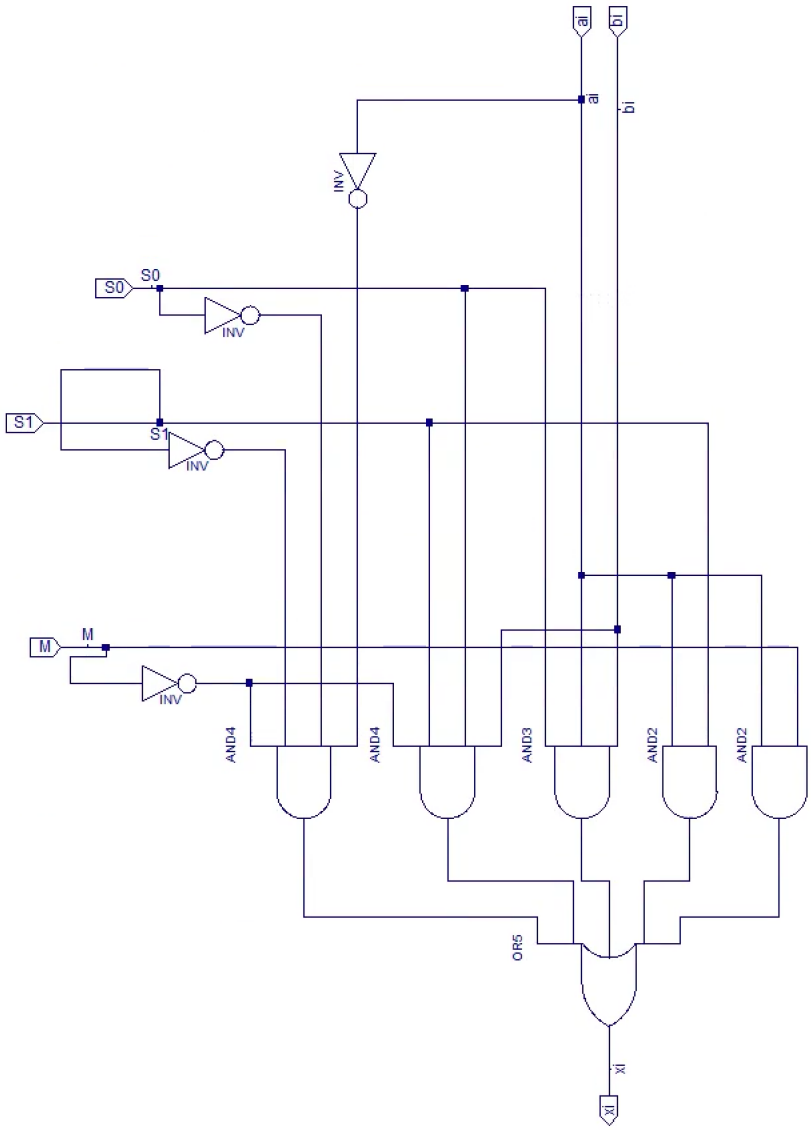
We then can add onto this by adding the ability to subtract and making it 8-bit, thus creating an 8-bit adder/subtractor



```
‘timescale 1ns / 1ps
//alu_sch_alu_sch_sch_tb
module alu_tbw_tb();

// Inputs
   reg [7:0] X = 8’b00000000;
   reg [7:0] Y = 8’b00000000;
   reg SEL = 1’b0;

// Output
   wire [7:0] DATA_OUT;
   wire Cnext;

// Instantiate the UUT
   alu_sch UUT (
                .X(X),
                .DATA_OUT(DATA_OUT),
                .Cnext(Cnext),
                .Y(Y),
                .SEL(SEL)
   );
// Initialize Inputs
initial begin
#100;   //Wait 100ns for initial inputs to settle.
for (i=0; i<max_count; i=i+1)
        begin
                {X,Y,SEL} = i;  //Cycle through all input combinations.
                #100;   //Wait 100ns between new inputs.
        end
end
endmodule
```

2.2 Part 1

Next, we built a logic extender so that we can output logic operations to the full adder.

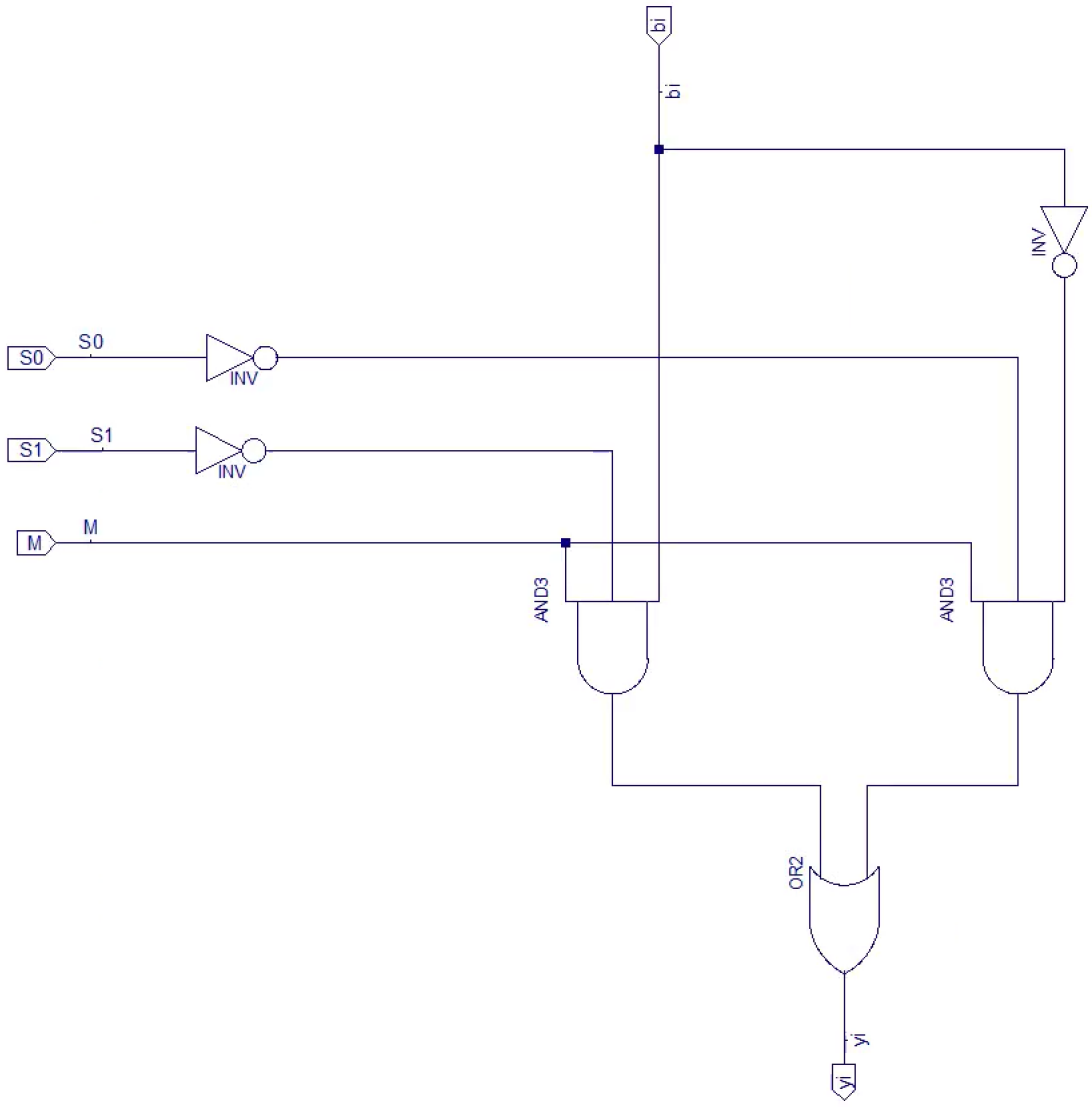



```
‘timescale 1ns / 1ps

module logic_ext_tbw_tb_0;

// Inputs
   reg ai;
   reg bi;
   reg S0;
   reg S1;
   reg M;
```

```
// Output
   wire xi;

        integer i = 0;
        parameter num_inputs = 5;
        parameter max_count = (1<<num_inputs);

// Instantiate the UUT
   logic_ext UUT (
                .xi(xi),
                .ai(ai),
                .bi(bi),
                .S0(S0),
                .S1(S1),
                .M(M)
   );
// Initialize Inputs
       initial begin
#100;
 for (i=0; i<max_count; i=i+1)
        begin {M,S1,S0,ai,bi} = i;
        #100;
        end
end
endmodule
```

2.3 Part 2

Similar to the Logic Extender, we will build an Arithmetic extender, which forwards arithmetic operations to the full adder rather than logic ones.

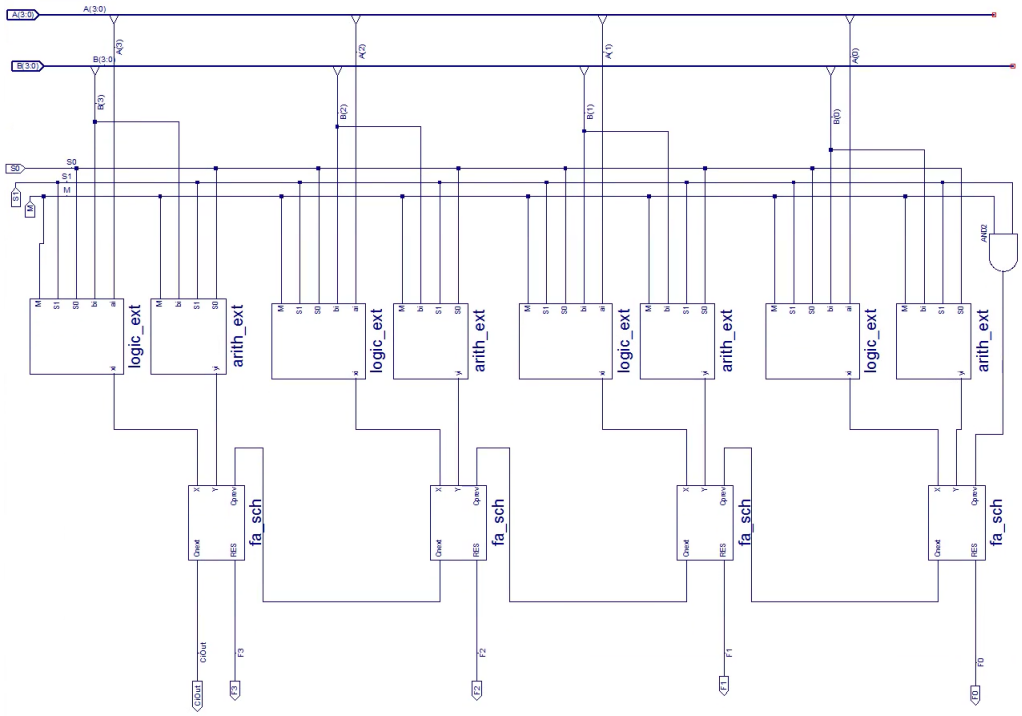



```
‘timescale 1ns/1ps
module arith_ext_tbw_tb_0;
reg bi = 1’b0;
reg M = 1’b0;
reg S0 = 1’b0;
reg S1 = 1’b0;
wire yi;
integer i = 0;
parameter num_inputs = 4;
parameter max_count = (1<<num_inputs);
```

```
arith_ext UUT (
.bi(bi),
.M(M),
.S0(S0),
.S1(S1),
.yi(yi));

initial begin
#100;   //Wait 100ns for initial inputs to settle.
for (i=0; i<max_count; i=i+1)
        begin
                {M,S1,S0,bi} = i;  //Cycle through all 4 input combinations.
                #100;   //Wait 100ns between new inputs.
        end
end

endmodule
```

2.4 Part 3

Now we can combine the previous parts into a working 4-bit ALU. In essence, we stack the Logic Extender and the Arithmetic Extender onto the Full Adder

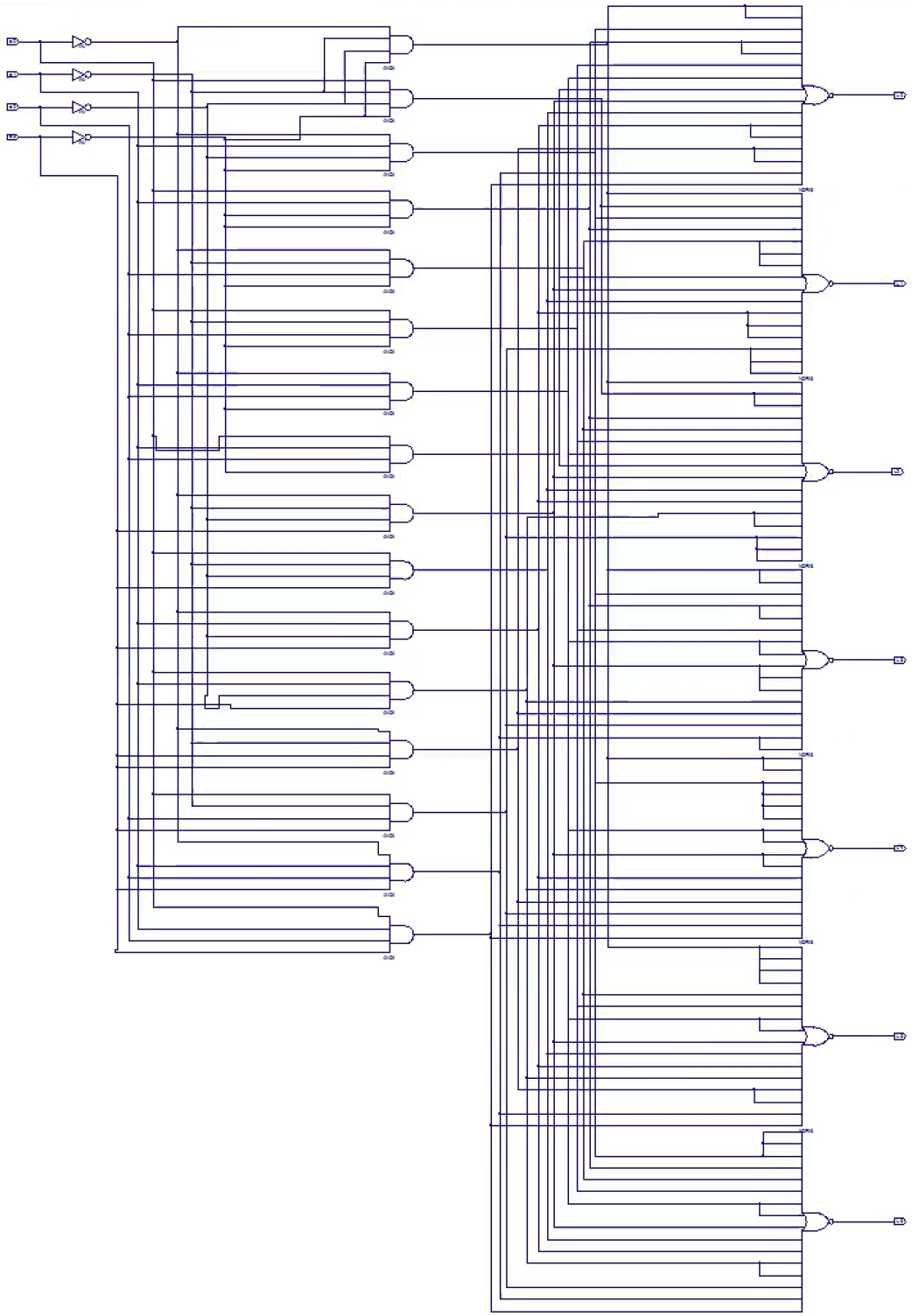


```
‘timescale 1ns / 1ps

module  alu4bit_tbw_tb_0;

// Inputs
   reg [3:0] A;
   reg [3:0] B;
   reg S0;
   reg S1;
   reg M;

// Output
   wire CiOut;
   wire F3;
   wire F2;
   wire F1;
   wire F0;

integer i =0;
parameter num_inputs =3;
parameter max_count = (1<<num_inputs);

// Instantiate the UUT
   alu4bit_sch UUT (
                .A(A),
                .B(B),
                .S0(S0),
                .S1(S1),
                .M(M),
                .CiOut(CiOut),
                .F3(F3),
                .F2(F2),
                .F1(F1),
                .F0(F0)
   );
// Initialize Inputs

initial begin
#100;
for(i=0; i<max_count;i=i+1)
                begin
                        {M,S1,S0}=i;
                        A=4’b0101;
                        B=4’b0100;
                        #100;
                end
        #100;
```

```
        for(i=0; i<max_count;i=i+1)
                begin
                        {M,S1,S0}=i;
                        A=4'b1010;
                        B=4'b0101;
                        #100;
                end
        end
endmodule
```

2.5 Part 4

During the lab period and for the demo, we used the the most updated version of our seven segment display driver from the previous class, and created a symbol of the verilog file, since an error in our schematic design gave us time constraint worries. We fixed the logic issue since then, and the schematic and test bench now work as expected.



```
‘timescale 1ns / 1ps
module Seven_Segment_Display_tb();

        reg [3:0] In;
        wire A_t, B_t,C_t, D_t, E_t, F_t, G_t;

        Seven_Segment_Display Seven_Segment_Display_1(
        In, A_t, B_t,C_t, D_t, E_t, F_t, G_t);

        initial
        begin

        //Case 0
        In <= 4’b0000;
        #1 $display("");
        #1 $display("Case 0: ");
        #1 $display("A_t = %b", A_t);
        #1 $display("B_t = %b", B_t);
        #1 $display("C_t = %b", C_t);
        #1 $display("D_t = %b", D_t);
        #1 $display("E_t = %b", E_t);
        #1 $display("F_t = %b", F_t);
        #1 $display("G_t = %b", G_t);

        //Case 1
In <= 4’b0001;
        #1 $display("");
        #1 $display("Case 1: ");
        #1 $display("A_t = %b", A_t);
        #1 $display("B_t = %b", B_t);
        #1 $display("C_t = %b", C_t);
        #1 $display("D_t = %b", D_t);
        #1 $display("E_t = %b", E_t);
        #1 $display("F_t = %b", F_t);
        #1 $display("G_t = %b", G_t);

        //Case 2
In <= 4’b0010;
        #1 $display("");
        #1 $display("Case 2: ");
        #1 $display("A_t = %b", A_t);
        #1 $display("B_t = %b", B_t);
        #1 $display("C_t = %b", C_t);
        #1 $display("D_t = %b", D_t);
        #1 $display("E_t = %b", E_t);
        #1 $display("F_t = %b", F_t);
        #1 $display("G_t = %b", G_t);
```

```
        //Case 3
In <= 4'b0011;
        #1 $display("");
        #1 $display("Case 3: ");
        #1 $display("A_t = %b", A_t);
        #1 $display("B_t = %b", B_t);
        #1 $display("C_t = %b", C_t);
        #1 $display("D_t = %b", D_t);
        #1 $display("E_t = %b", E_t);
        #1 $display("F_t = %b", F_t);
        #1 $display("G_t = %b", G_t);

        //Case 4
In <= 4'b0100;
        #1 $display("");
        #1 $display("Case 4: ");
        #1 $display("A_t = %b", A_t);
        #1 $display("B_t = %b", B_t);
        #1 $display("C_t = %b", C_t);
        #1 $display("D_t = %b", D_t);
        #1 $display("E_t = %b", E_t);
        #1 $display("F_t = %b", F_t);
        #1 $display("G_t = %b", G_t);

        //Case 5
In <= 4'b0101;
        #1 $display("");
        #1 $display("Case 5: ");
        #1 $display("A_t = %b", A_t);
        #1 $display("B_t = %b", B_t);
        #1 $display("C_t = %b", C_t);
        #1 $display("D_t = %b", D_t);
        #1 $display("E_t = %b", E_t);
        #1 $display("F_t = %b", F_t);
        #1 $display("G_t = %b", G_t);

        //Case 6
In <= 4'b0110;
        #1 $display("");
        #1 $display("Case 6: ");
        #1 $display("A_t = %b", A_t);
        #1 $display("B_t = %b", B_t);
        #1 $display("C_t = %b", C_t);
        #1 $display("D_t = %b", D_t);
        #1 $display("E_t = %b", E_t);
        #1 $display("F_t = %b", F_t);
```

```
        #1 $display("G_t = %b", G_t);

        //Case 7
In <= 4'b0111;
        #1 $display("");
        #1 $display("Case 7: ");
        #1 $display("A_t = %b", A_t);
        #1 $display("B_t = %b", B_t);
        #1 $display("C_t = %b", C_t);
        #1 $display("D_t = %b", D_t);
        #1 $display("E_t = %b", E_t);
        #1 $display("F_t = %b", F_t);
        #1 $display("G_t = %b", G_t);

        //Case 8
In <= 4'b1000;
        #1 $display("");
        #1 $display("Case 8: ");
        #1 $display("A_t = %b", A_t);
        #1 $display("B_t = %b", B_t);
        #1 $display("C_t = %b", C_t);
        #1 $display("D_t = %b", D_t);
        #1 $display("E_t = %b", E_t);
        #1 $display("F_t = %b", F_t);
        #1 $display("G_t = %b", G_t);

        //Case 9
        In <= 4'b1001;
        #1 $display("");
        #1 $display("Case 9: ");
        #1 $display("A_t = %b", A_t);
        #1 $display("B_t = %b", B_t);
        #1 $display("C_t = %b", C_t);
        #1 $display("D_t = %b", D_t);
        #1 $display("E_t = %b", E_t);
        #1 $display("F_t = %b", F_t);
        #1 $display("G_t = %b", G_t);

        //Case 10
        In <= 4'b1010;
        #1 $display("");
        #1 $display("Case 10: ");
        #1 $display("A_t = %b", A_t);
        #1 $display("B_t = %b", B_t);
        #1 $display("C_t = %b", C_t);
        #1 $display("D_t = %b", D_t);
        #1 $display("E_t = %b", E_t);
```

```
#1 $display("F_t = %b", F_t);
#1 $display("G_t = %b", G_t);

//Case 11
In <= 4'b1011;
#1 $display("");
#1 $display("Case 11: ");
#1 $display("A_t = %b", A_t);
#1 $display("B_t = %b", B_t);
#1 $display("C_t = %b", C_t);
#1 $display("D_t = %b", D_t);
#1 $display("E_t = %b", E_t);
#1 $display("F_t = %b", F_t);
#1 $display("G_t = %b", G_t);

//Case 12
In <= 4'b1100;
#1 $display("");
#1 $display("Case 12: ");
#1 $display("A_t = %b", A_t);
#1 $display("B_t = %b", B_t);
#1 $display("C_t = %b", C_t);
#1 $display("D_t = %b", D_t);
#1 $display("E_t = %b", E_t);
#1 $display("F_t = %b", F_t);
#1 $display("G_t = %b", G_t);

//Case 13
In <= 4'b1101;
#1 $display("");
#1 $display("Case 13: ");
#1 $display("A_t = %b", A_t);
#1 $display("B_t = %b", B_t);
#1 $display("C_t = %b", C_t);
#1 $display("D_t = %b", D_t);
#1 $display("E_t = %b", E_t);
#1 $display("F_t = %b", F_t);
#1 $display("G_t = %b", G_t);

//Case 14
In <= 4'b1110;
#1 $display("");
#1 $display("Case 14: ");
#1 $display("A_t = %b", A_t);
#1 $display("B_t = %b", B_t);
#1 $display("C_t = %b", C_t);
#1 $display("D_t = %b", D_t);
```

```
        #1 $display("E_t = %b", E_t);
        #1 $display("F_t = %b", F_t);
        #1 $display("G_t = %b", G_t);

        //Case 15
        In <= 4'b1111;
        #1 $display("");
        #1 $display("Case 15: ");
        #1 $display("A_t = %b", A_t);
        #1 $display("B_t = %b", B_t);
        #1 $display("C_t = %b", C_t);
        #1 $display("D_t = %b", D_t);
        #1 $display("E_t = %b", E_t);
        #1 $display("F_t = %b", F_t);
        #1 $display("G_t = %b", G_t);

        end
endmodule
```

2.6 Part 5

Put together all of the parts with instruction of the given schematic. Tested everything per usual with a test bench.

2.7 Part 6

With the schematics in place and tests ran, a UCF file can be created to implement on the board.

```
NET "A(3)" LOC = "N3";
NET "A(2)" LOC = "E2";
NET "A(1)" LOC = "F3";
NET "A(0)" LOC = "G3";

NET "B(3)" LOC = "B4";
NET "B(2)" LOC = "K3";
NET "B(1)" LOC = "L3";
NET "B(0)" LOC = "P11";

NET "M"   LOC = "A7";
NET "S1"  LOC = "M4";
NET "S0"  LOC = "C11";

NET "CiOut" LOC = "G1";
NET "CLK"  LOC = "B8";
```

```
NET "F3" LOC = "P6";
NET "F2" LOC = "P7";
NET "F1" LOC = "M11";
NET "F0" LOC = "M5";

NET "SS(0)" LOC = "L14";
NET "SS(1)" LOC = "H12";
NET "SS(2)" LOC = "N14";
NET "SS(3)" LOC = "N11";
NET "SS(4)" LOC = "P12";
NET "SS(5)" LOC = "L13";
NET "SS(6)" LOC = "M12";

NET "EN_L"  LOC = "K14";
NET "EN_ML" LOC = "M13";
NET "EN_MR" LOC = "J12";
NET "EN_R"  LOC = "F12";
```

2.8 Part 7

Programmed the board, implemented our program on it, and ran through every logic and arithmetic operation in demonstration. Everything worked perfectly after some debugging.

3 Experimental Results

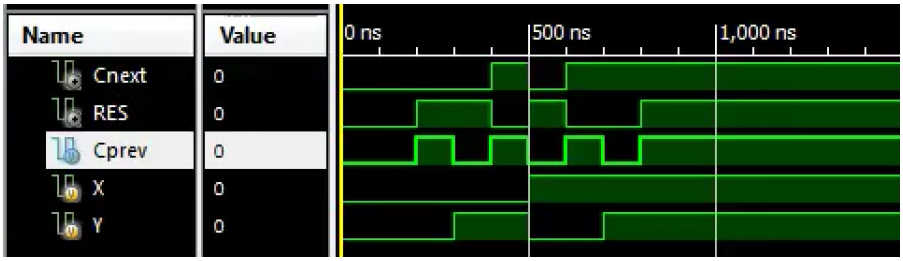


Figure 1: Waveform for the full adder schematic test bench. All test cases present for each value of X, Y, and Cprev, showing successful binary addition and carrying.

Figure 2: Waveform for the adder/subtractor. Used a loop to increment X,Y, and SEL in a manner that would cover all input cases.

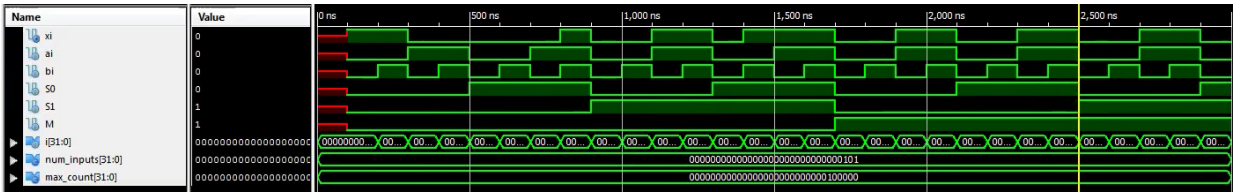


Figure 3: Waveform for the logic extender schematic test bench. All test cases match that of the table given, successfully performing all required logic operations. Used another loop to cycle through the input combinations.

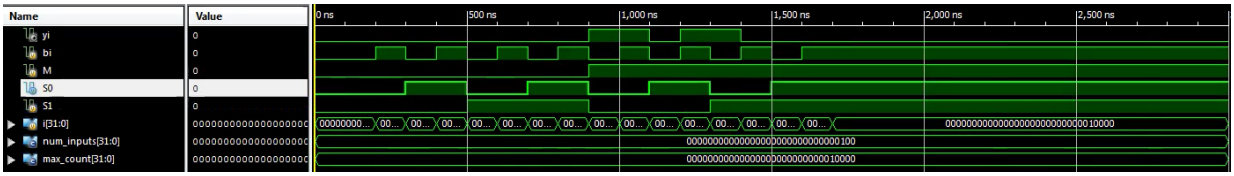


Figure 4: Waveform for the arithmetic extender schematic test bench. All test cases match that of the table given, successfully performing all required arithmetic operations

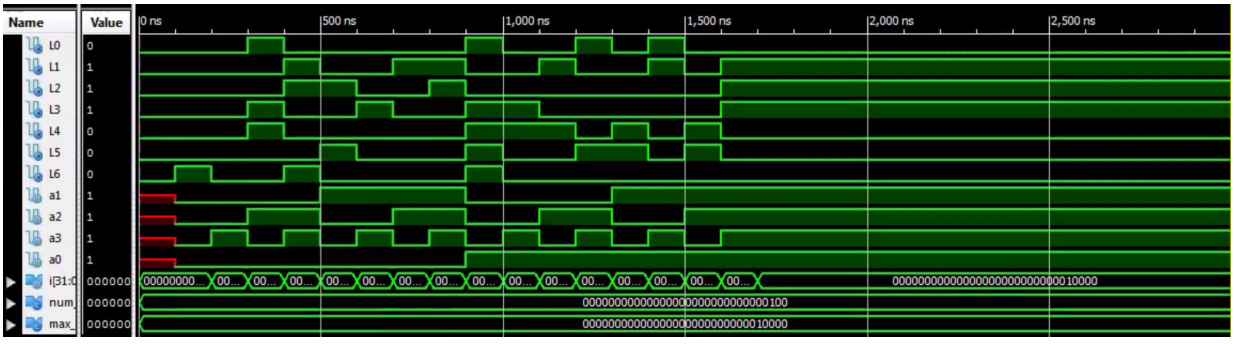


Figure 5: Waveform for the seven segment display driver, showing all possible combinations for the 4-bit binary input, and corresponding outputs.

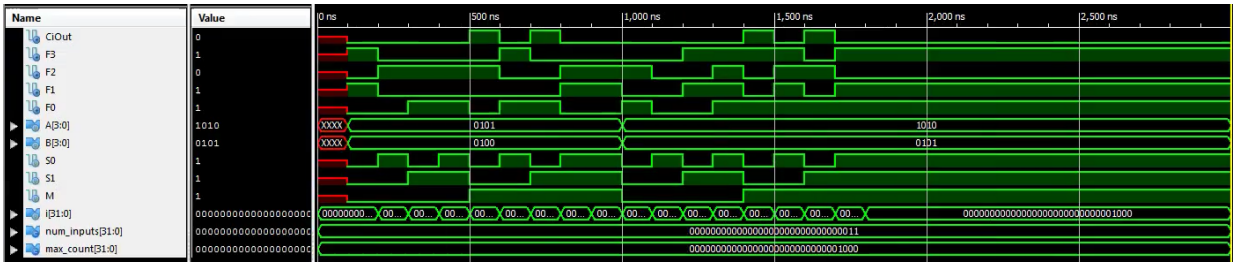


Figure 6: Waveform for the 4-bit ALU, testing again each possible input combination of S0, S1, and M.

4 Significance

This lab was the first comprehensive use of the Xilinx GUI schematic creator, and using it to create several modules that would eventually create a full project, and eventually exist on the board. While this did feel like a full, independent project of its own, in reality it is just the first piece of a much bigger puzzle that is a usable CPU. With the knowledge gained in this lab, piecing the parts together makes much more sense now.

5 Comments/Suggestions

Everything was rather straightforward, although, relatively, it was quite a bit of work. I know it was recommended to start out of class, but in the future, it may be best to assign a portion of the lab as prelab, so that it could be more spread out.