Lab 03

Designing the Toy Processor Datapath

	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

February 1, 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
```

3 Experimental Results

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 it was relatively quite a bit of work. In the future, it may be best to assign a portion of the lab as prelab, so that it could be more spread out.