**CS M152A / EE M116L**

**Lab 2: Floating Point Conversion**

**Scott Shi**

**Minh Le**

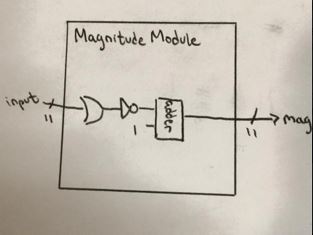
**I. INTRODUCTION AND REQUIREMENT**

**1.Floating Point Converter (Detailed Design Requirements)**

MODULES:

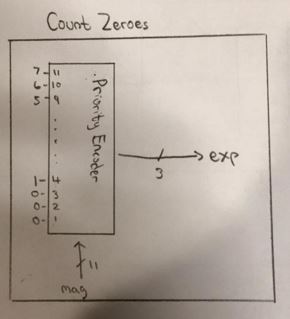
1. **Magnitude Module**

Magnitude module takes in the bottom 11 bits of the input and applies 2-complement to the module to extract the positive magnitude of the module. This is done with inverter gates and an adder.



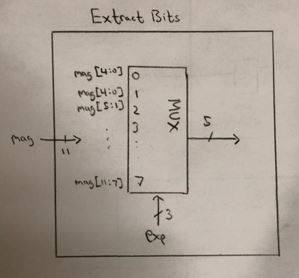
1. **Exponent Module**

Exponent module is a priority encoder that takes in magnitude and returns the corresponding value of exponent (hard wired as seen in diagram). Anything below 4 will have an exponent of 0.



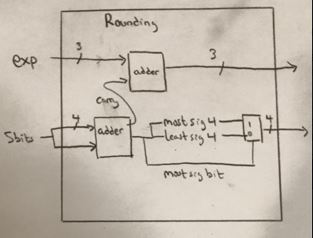
1. **Extract Leading Bits Module**

Extract leading bit takes in exponent values from previous module and extract the bits after the padded 0. This is possible because exponent value is related to number of zeros. In this design, we use a mux and hardcoded the input based on the case.



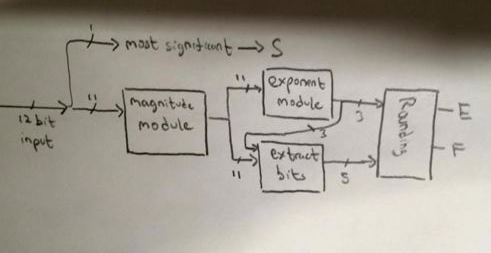
**IV. Rounding Module**

Rounding module rounds the 5 leading bits from leading bits module and adds the least significant bit to the 4 most significant bit using and adder. The carry signal will be inputted into an adder with the exponents bit from module II. Using a mux switched by most significant bit of result of the first adder, we will select either the most 4 significant or least 4 significant bit of the sum.



**2. Overall Flow**

The modules are connected as the diagram below.



**3. High Level Description (Background Information)**

Our design will take in a 12 bit decimal number, convert it into it’s floating point representation, and output the 8 bit representation (1 sign bit, 3 exponent bits, 4 float bits). The first eight bits of input will be from the switches, and the four other from a Pmod connector. The 8 output bits will be displayed on the LEDs. In the case where the magnitude of the number is too large to be represented, we just output all 1’s for the exponent and float bits.

**II. DESIGN DESCRIPTION**

**3. Code**

===

\*.v

===

`timescale 1ns / 1ps

module FPCVT(

D, //input - 12 bit 2 complement number

S, //output - signed bit

E, //output - 3 bit exponent

F //output - 4 bit significand

);

input [11:0] D;

output S;

output [2:0] E;

output [3:0] F;

assign S = D[11];

wire [11:0] mag; //magnitude

assign mag = (D[11] ? ~D + 1'b1 : D);

wire [2:0] temp\_exp;

assign temp\_exp =

mag[10]? 3'd7:

mag[9] ? 3'd6:

mag[8] ? 3'd5:

mag[7] ? 3'd4:

mag[6] ? 3'd3:

mag[5] ? 3'd2:

mag[4] ? 3'd1:

3'd0;

wire [4:0]five\_bits;

assign five\_bits = mag >> (|temp\_exp ? (temp\_exp-1'b1) : 0);

wire [4:0] add\_result;

assign add\_result = |temp\_exp ? five\_bits[4:1] + five\_bits[0] : five\_bits[3:0];

wire condition;

assign condition = (((&five\_bits) & (&temp\_exp))//used up all exp/sig bits

| D[11] & (&(~D[10:0]))); //in minimum

assign F = condition ? 4'b1111 : //special case

add\_result[4] ? add\_result[4:1] : //overflow

add\_result[3:0]; //no overflow

assign E = condition ? 3'b111 : //special case

add\_result[4] ? temp\_exp+1'b1 : //overflow

temp\_exp; //no overflow

endmodule

=====

\*TB.v

=====

`timescale 1ns / 1ps

module FPCVT\_TB;

// Inputs

reg [11:0] D;

// Outputs

wire S;

wire [2:0] E;

wire [3:0] F;

// Instantiate the Unit Under Test (UUT)

FPCVT uut (.D(D), .S(S), .E(E), .F(F));

initial begin

// Initialize Inputs

D = 0;

// Wait 100 ns for global reset to finish

#4096 $finish;

End

always begin

#1 D = D+1;

end

endmodule

====

\*.ucf

====

## Leds

NET "F<0>" LOC = "U16" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L2P\_CMPCLK, Sch name = LD0

NET "F<1>" LOC = "V16" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L2N\_CMPMOSI, Sch name = LD1

NET "F<2>" LOC = "U15" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L5P, Sch name = LD2

NET "F<3>" LOC = "V15" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L5N, Sch name = LD3

NET "E<0>" LOC = "M11" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L15P, Sch name = LD4

NET "E<1>" LOC = "N11" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L15N, Sch name = LD5

NET "E<2>" LOC = "R11" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L16P, Sch name = LD6

NET "S" LOC = "T11" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L16N\_VREF, Sch name = LD7

## Switches

NET "D<0>" LOC = "T10" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L29N\_GCLK2, Sch name = SW0

NET "D<1>" LOC = "T9" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L32P\_GCLK29, Sch name = SW1

NET "D<2>" LOC = "V9" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L32N\_GCLK28, Sch name = SW2

NET "D<3>" LOC = "M8" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L40P, Sch name = SW3

NET "D<4>" LOC = "N8" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L40N, Sch name = SW4

NET "D<5>" LOC = "U8" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L41P, Sch name = SW5

NET "D<6>" LOC = "V8" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L41N\_VREF, Sch name = SW6

NET "D<7>" LOC = "T5" | IOSTANDARD = "LVCMOS33"; #Bank = MISC, Pin name = IO\_L48N\_RDWR\_B\_VREF\_2, Sch name = SW7

## Buttons

#up

NET "D<8>" LOC = "A8" | IOSTANDARD = "LVCMOS33"; #Bank = 0, Pin name = IO\_L33N, Sch name = BTNU

#left

NET "D<9>" LOC = "C4" | IOSTANDARD = "LVCMOS33"; #Bank = 0, Pin name = IO\_L1N\_VREF, Sch name = BTNL

#down

NET "D<10>" LOC = "C9" | IOSTANDARD = "LVCMOS33"; #Bank = 0, Pin name = IO\_L34N\_GCLK18, Sch name = BTND

#right

NET "D<11>" LOC = "D9" | IOSTANDARD = "LVCMOS33"; #Bank = 0, Pin name = IO\_L34P\_GCLK19, Sch name = BTNR

**4. Synthesis Report**

========================================

\* Advanced Synthesis Report \*

========================================

Macro Statistics

# Adders/Subtractors : 4

12-bit adder : 1

3-bit adder : 1

4-bit subtractor : 1

5-bit adder : 1

# Multiplexers : 14

12-bit 2-to-1 multiplexer : 2

3-bit 2-to-1 multiplexer : 8

32-bit 2-to-1 multiplexer : 1

4-bit 2-to-1 multiplexer : 2

5-bit 2-to-1 multiplexer : 1

# Logic shifters : 1

12-bit shifter logical right : 1

========================================

\* Design Summary \*

========================================

Primitive and Black Box Usage:

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

# BELS : 84

# GND : 1

# INV : 11

# LUT1 : 1

# LUT3 : 13

# LUT4 : 2

# LUT5 : 9

# LUT6 : 23

# MUXCY : 11

# VCC : 1

# XORCY : 12

# IO Buffers : 20

# IBUF : 12

# OBUF : 8

Device utilization summary:

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

Selected Device : 6slx16csg324-3

Slice Logic Utilization:

Number of Slice LUTs: 59 out of 9112 0%

Number used as Logic: 59 out of 9112 0%

Slice Logic Distribution:

Number of LUT Flip Flop pairs used: 59

Number with an unused Flip Flop: 59 out of 59 100%

Number with an unused LUT: 0 out of 59 0%

Number of fully used LUT-FF pairs: 0 out of 59 0%

Number of unique control sets: 0

IO Utilization:

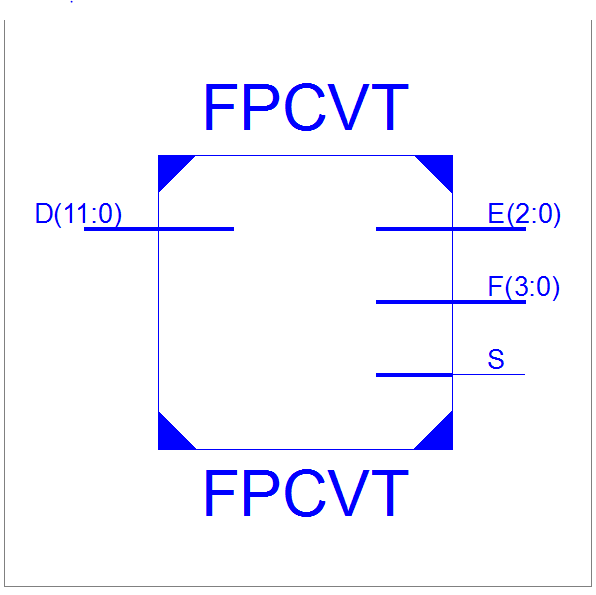
Number of IOs: 20

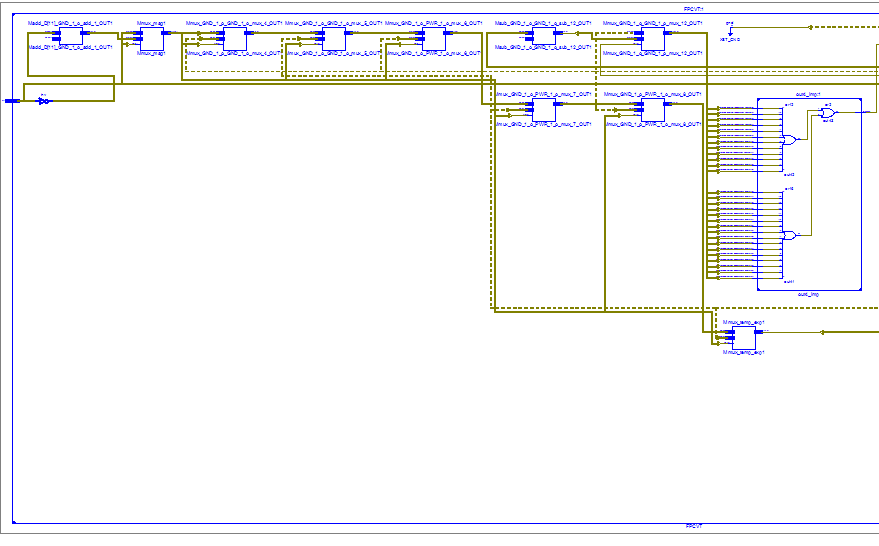
Number of bonded IOBs: 20 out of 232 8%

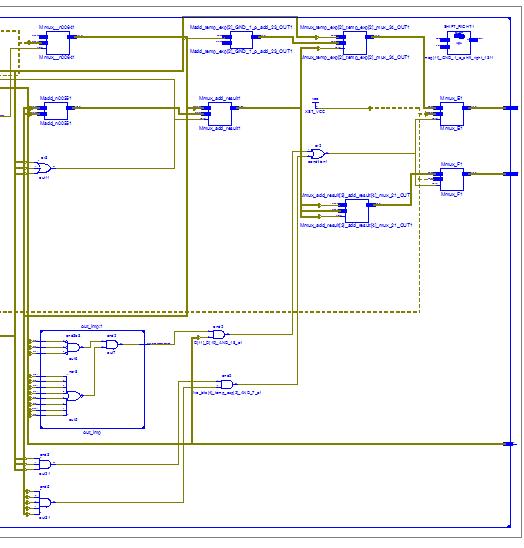
**5. Device Utilization Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| Slice Logic Utilization | Used | Available | Utilization |
| # of Slice LUTs | 50 | 9,112 | 1% |
| # used as logic | 50 | 9,112 | 1% |
| # using O6 output only | 30 |  |  |
| # using O5 output only | 11 |  |  |
| # using O5 and O6 | 9 |  |  |
| # of occupied Slices | 23 | 2,278 | 1% |
| # of MUXCYs used | 12 | 4,556 | 1% |
| # of LUT Flip Flop pair used | 50 |  |  |
| # with an unused Flip Flop | 50 | 50 | 100% |
| # of bonded IOBs | 20 | 232 | 8% |
| # of LOCed IOBs | 20 | 20 | 100% |
| Avg Fanout of Non-Clk Nets | 3.19 |  |  |

**6. High Level Schematics**



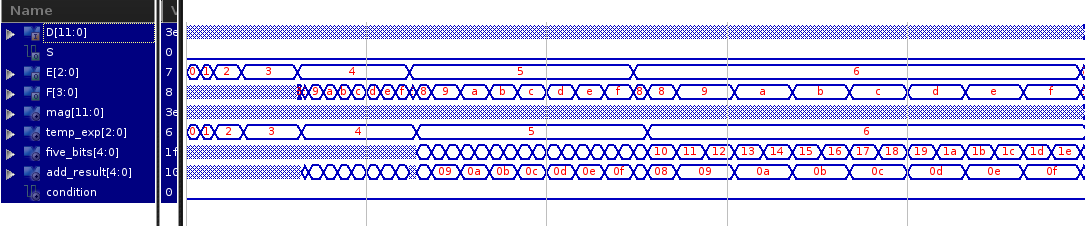




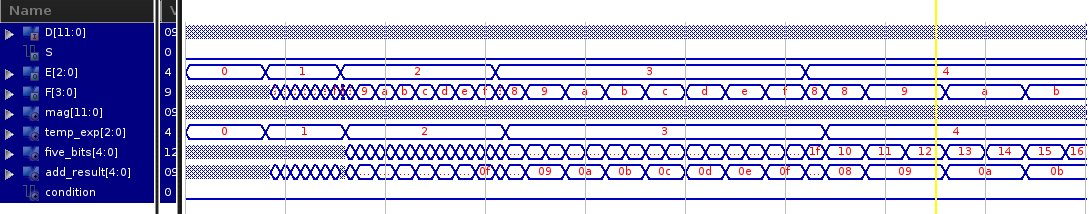
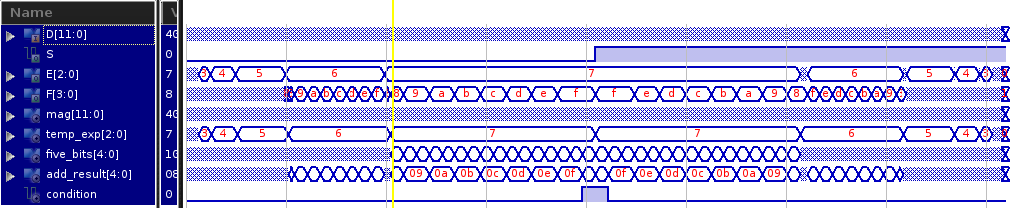
**7. Maximum Clock Frequency:** N/A

**III. SIMULATION DOCUMENTATION (TB is above)**

**8. Waveform**



\*zoomed in\*



**Test Cases:**

To test our code, we used the test cases provided in the lab specification. To do some general testing, we tried inputs 1-20 in decimal format. To do some more edge case testing, we pushed the buttons that corresponded to the more significant bits of the decimal representation input. Indeed, when the decimal number was unable to be represented in float format, all the LEDs lit up. Finally, we also tested our design against inputs with 1, 2, ..>= 8 leading zeros. One area of interest was when we did not have enough bits to represent the numbers. This occurred in the middle of the 2nd wave form where we see E is 0x7 and F is 0xf.

**IV. CONCLUSION**

Our design took in 12 bits of input (from the switches and 4 of the buttons) and treated it as an 12 bit decimal number. First we looked at the most significant bit and treated that as our sign bit for the output. If it was negative, we would set the sign bit to 1 and negate the input (~x + 1) then proceed in the process. Next, we looked at the number of leading zeros to calculate the exponent bits of our output using a long if/else chain and some other combination logic. With the combinational logic, we wanted to denote that if there’s 1 leading zero, the exponent bits will be 111 (7), 2 leading zeros, E will be 110 (6), and so on until there’s 8 or more leading zeros that E will be 000 (0). After that, the 4 bits after the final leading zero become our 4 output bits for F. Finally, we added some hard-coded cases for when there was overflow and numbers could not be represented. In that case, we just had all the output bits be 1’s.

A couple of challenges we ran into involved the edge cases, specifically int min and other inputs where we could not represent the number in float. To solve these challenges, we simply hard coded the output to be all ‘1s’ as the specification told us to do. Another big challenge we had was the lack of always loops since it was a combinational circuit. We were able to get around this problem by using long if/else chains.