Lab 2:
Floating Point Conversion

Matthew Wong 704569674
Wenlong Xiong 204407085
Vincent Jin 604464719

CS M152A Winter 2017
Monday - Wednesday, 10 - 12

Introduction

For this lab, we designed a combinational circuit that converts a 12-bit two's-complement representation into 8-bit floating point representation. This floating point representation consists of a 1-bit sign representation, a 3-bit exponent, and a 4-bit significand. The value represented by these 8 bits is:

$$V = (-1)^S \times F \times 2^E$$

Since our combinational circuit will be converting 12 input bits into 8 output bits, our circuit will be performing compression. Thus, it is possible for multiple unique 12-bit inputs to produce the same 8-bit output. Also, it is quite likely that a value in 12-bit two's-complement cannot be exactly expressed in 8-bit floating point. In this case, the 12-bit value will be rounded to the closest possible 8-bit floating point value.

Our implementation will be tested and run on the Xilinx simulation (not the FPGA board). Simulation tests are listed below in the simulation documentation section.

Design Description

The overall design of the combinational circuit will consist of a top module that contains three submodules. As seen in Figure 1, the top module, called FPCVT, simply directs the 12-bit input into the first submodule, connects the outputs and inputs of the submodules, and condenses the desired 8-bit output from the submodules.

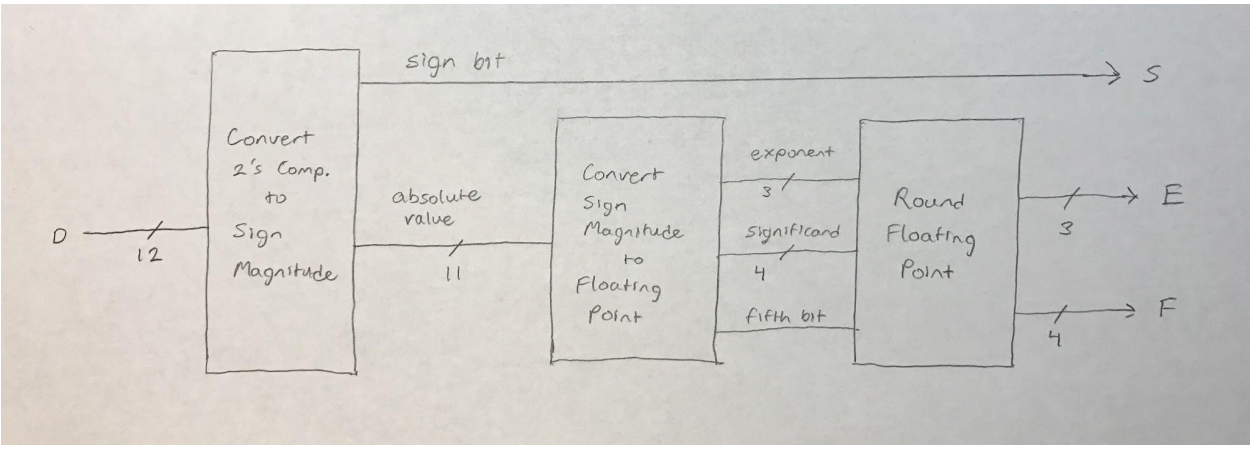

Figure 1: **High Level Design of the Top Module.**

The first submodule is responsible for converting the 12-bit two's-complement input into a 12-bit output in sign magnitude representation. The submodule checks most significant bit to determine the sign of the input. If the input is positive (the most significant bit is 0), then the other 11 bits are left unchanged. Otherwise, the input is negative and the lower 11 bits are inverted and incremented by 1. One problem occurs with 12'b100000000000, which is -2048. When the invert-increment is applied to -2048, the result is still 12'b100000000000, which is 0 in sign magnitude. Our module checks for this case. If the input equals -2048, the module will output -2047, which is 12 bits that are all 1.

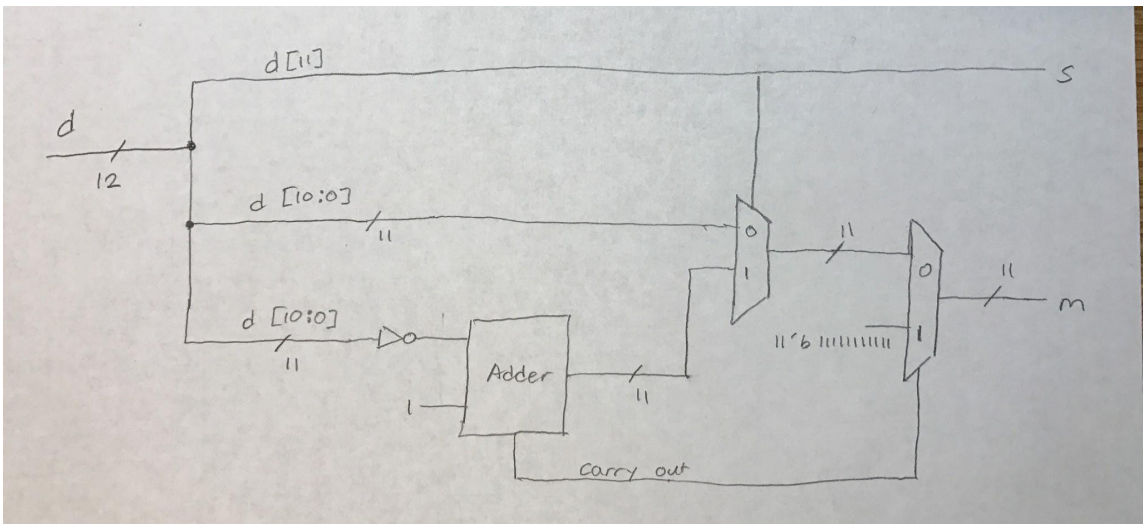

Figure 2: **Design of Conversion from Two's-Complement to Sign Magnitude.** The most significant bit becomes the sign bit output and selects whether to invert-increment the other 11 bits. The carry-out from the adder is used to catch the special case of the input being -2048.

The second submodule is responsible for converting the 12-bit sign magnitude into floating point representation. The submodule uses the lower 11 bits of the sign magnitude input to determine the output of the 3-bit exponent, 4-bit significand, and 1-bit fifth bit that the next submodule will use for rounding. To explain, the submodule checks the number of leading zero-bits in the lower 11 bits of the input. If there are no leading zero-bits, the exponent will be 7. If there is one leading zero-bit, the exponent will be 6. If there are two leading zero-bits, the exponent will be 5, and the pattern continues. The significand is the next 4 bits immediately following the last leading zero-bit. The fifth bit is the bit following the 4 bits used for the significand. If there are at least 8 leading zero-bits, then the significand is automatically set to the lowest 4 bits and the fifth bit is set to 0.

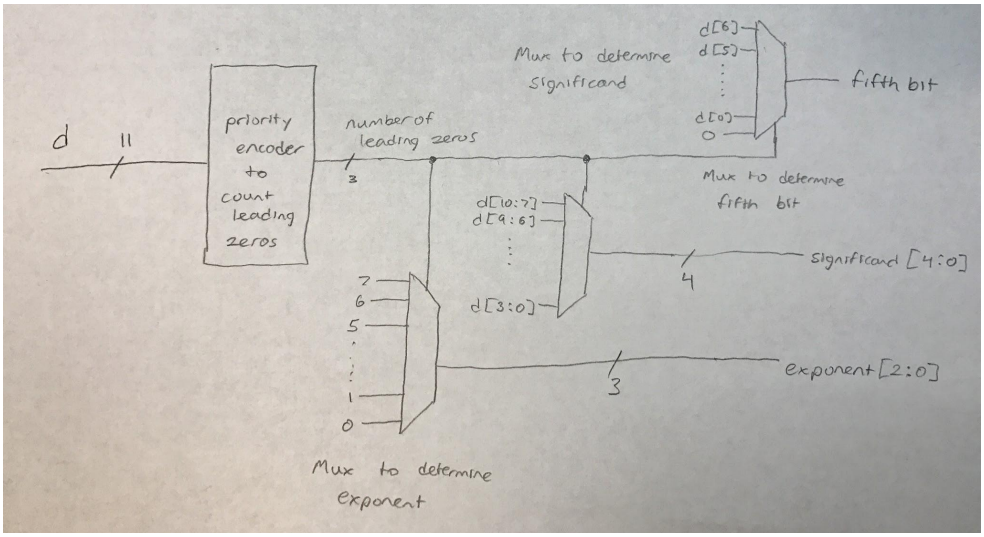

Figure 3: **Design of Conversion from Sign Magnitude to Floating Point.** The number of leading zeros is counted to determine which bits will be used for the output of the 3-bit exponent, 4-bit significand, and fifth bit.

The third submodule is responsible for rounding the exponent and significand based on the value of the fifth bit. The submodule takes in a 3-bit exponent, a 4-bit significand, and the fifth bit as inputs. It will output the rounded 3-bit exponent and the rounded 4-bit significand. To explain, if the fifth bit is 0, the exponent and significand are left unchanged. Otherwise, if the fifth bit is 1, the 4-bit significand is incremented by 1. If the significand overflows, then the exponent is incremented by 1 and the significand is set to 4'b1000. There is a special case to be considered when the exponent is 3'b111, the significand is 4'b1111, and the fifth bit is a 1. Normally, the significand and exponent would both overflow when incremented, which is undesirable. Thus, the submodule will leave the exponent and significand unchanged in this case.

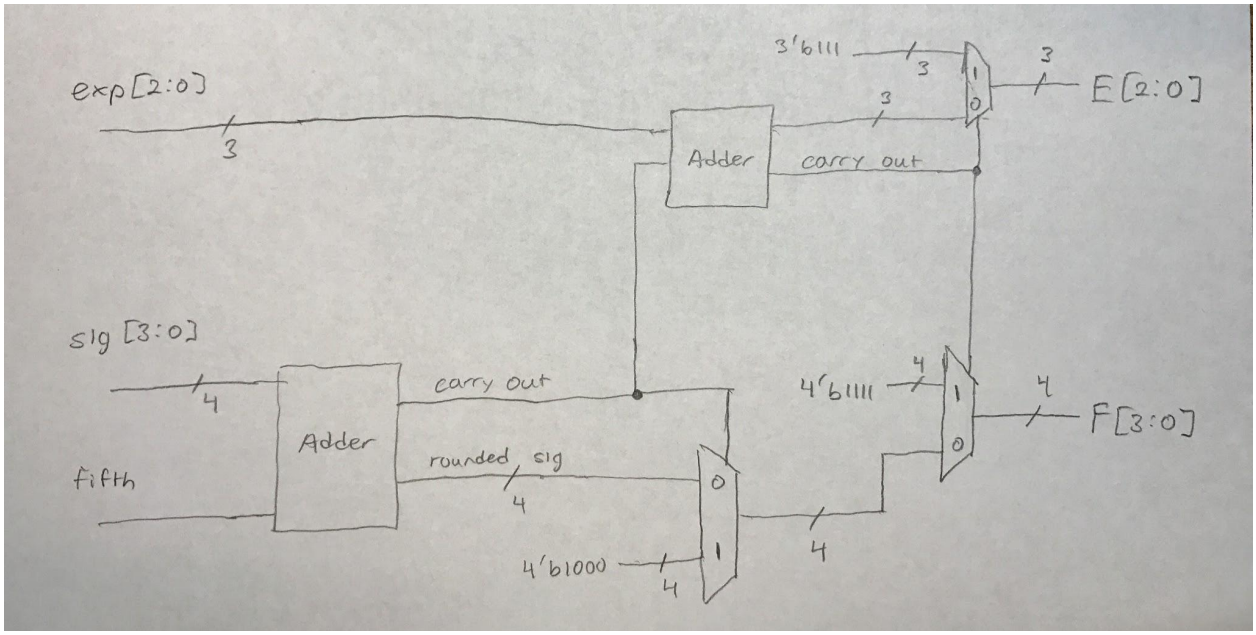

Figure 4: **Design of Rounding the Floating Point.** The significand and the fifth bit are added to determine the rounded significand. If the addition overflows, the exponent is incremented and the significand is set to 4'b1000. If the exponent addition overflows, the exponent is set to 3'b111 and the significand is set to 4'b1111.

Simulation Documentation

convert2toSM module:

Test Case	Result	Pass/No Pass
D = 12'b001000110010	Testing a normal positive number: S = 0 M = 01000110010	Passed
D = 12'b100001100000	Testing a normal negative number: S = 1 M = 11110100000	Passed
D = 12'b111111111111	Testing -1: S = 1 M = 00000000001	Passed
D = 12'b000000000000	Testing 0: S = 0 M = 00000000000	Passed
D = 12'b011111111111	Testing the maximum positive number: S = 0 M = 11111111111	Passed
D = 12'b100000000000	Testing the minimum negative number: S = 1 M = 11111111111	Passed

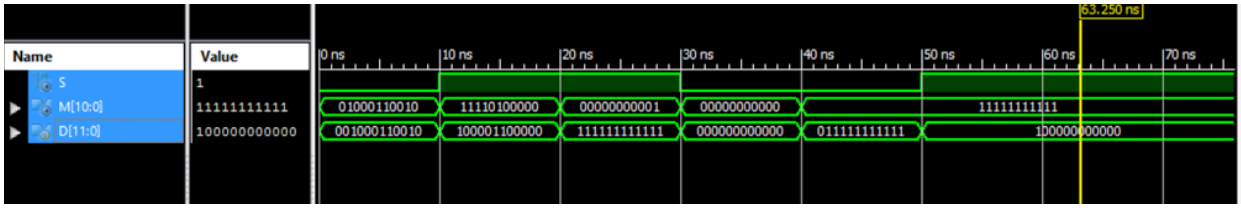

Figure 5: **Waveform of Test Cases for Conversion of Two's-Complement to Sign Magnitude**

count_leading_zeros module (convert SM to FP):

Test Case	Result	Pass/No Pass
D = 11'b10100110101	1 Leading Zero exp = 111, sig = 1010, fifth = 0	Passed
D = 11'b01100110101	2 Leading Zeros exp = 110, sig = 1100, fifth = 1	Passed
D = 11'b00100110101	3 Leading Zeros exp = 101, sig = 1001, fifth = 1	Passed
D = 11'b00010110101	4 Leading Zeros exp = 100, sig = 1011, fifth = 0	Passed
D = 11'b00001010101	5 Leading Zeros exp = 011, sig = 1010, fifth = 1	Passed
D = 11'b00000110101	6 Leading Zeros exp = 010, sig = 1101, fifth = 0	Passed
D = 11'b00000010101	7 Leading Zeros exp = 001, sig = 1010, fifth = 1	Passed
D = 11'b00000001101	8 Leading Zeros exp = 000, sig = 1101, fifth = 0	Passed

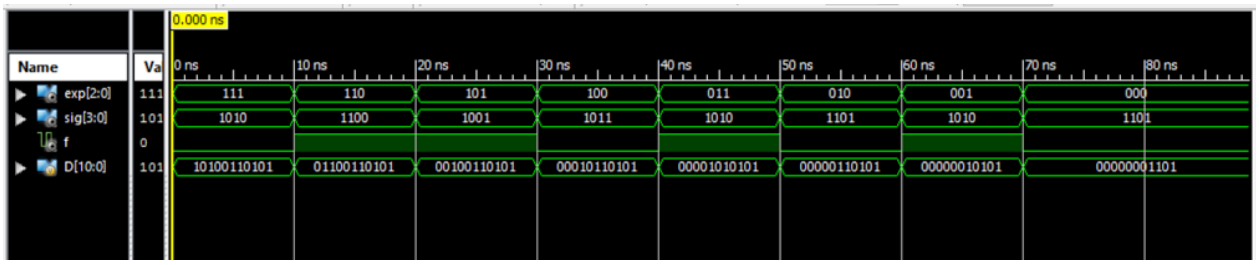

Figure 6: **Waveform of Test Cases for Conversion from Sign Magnitude to Floating Point**

rounding module:

Test Case	Result	Pass/No Pass
Exp =101 Sig =1010 Fifth = 0	No Rounding E = 101 F = 1010	Passed
Exp =101 Sig =1010 Fifth = 1	Rounding w/o Significand Overflow E = 101 F = 1011	Passed
Exp =101 Sig =1111 Fifth = 1	Rounding w/ Significand Overflow E = 110 F = 1000	Passed
Exp =111 Sig =1111 Fifth = 1	Rounding w/ Exp and Sig Overflow E = 111 F = 1111	Passed

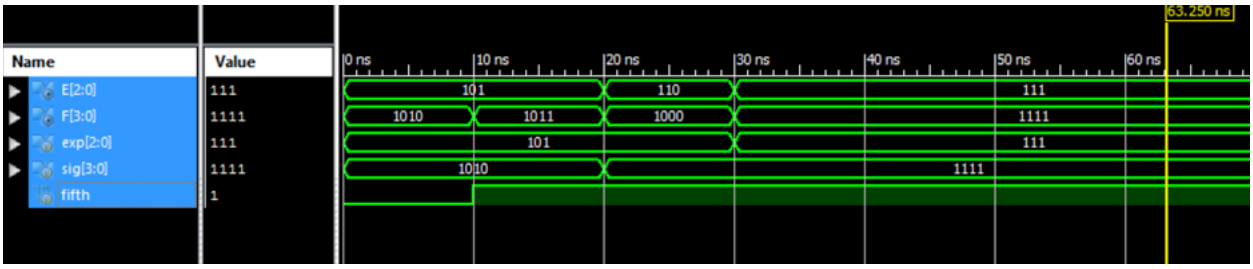

Figure 7: **Waveform of Test Cases for Rounding of Floating Point**

FPCVT module (top module):

Test Case	Result	Pass/No Pass
D = 12'b001000110010	Testing a normal positive number: S = 0 E = 110 F = 1001	Passed
D = 12'b111001100000	Testing a normal negative number: S = 1 E = 101 F = 1101	Passed
D = 12'b111111111111	Testing -1: S = 1 E = 000 F = 0001	Passed
D = 12'b000000000000	Testing 0: S = 0 E = 000 F = 0000	Passed
D = 12'b011111111111	Testing the maximum positive number: S = 0 E = 111 F = 1111	Passed
D = 12'b100000000000	Testing the minimum negative number: S = 1 E = 111 F = 1111	Passed

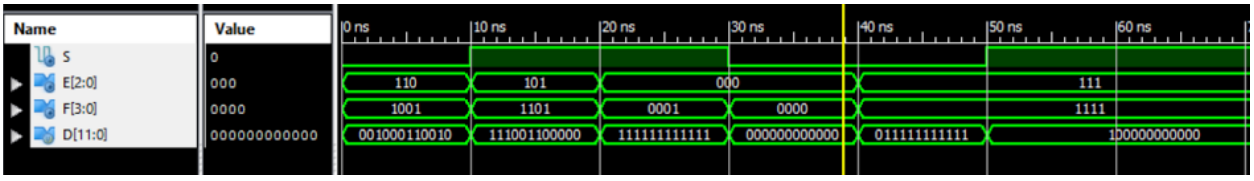

Figure 8: **Waveform of Test Cases for Top Module FPCVT**

Conclusion

The combinational circuit we have created can convert a 12-bit two's-complement number into its corresponding 8-bit floating point number. There are three main submodules within an encapsulating top module. The first submodule converts the two's-complement number into its sign magnitude number. The second submodule converts the sign magnitude number into its floating point exponent and significand. The third submodule rounds the exponent and significand to the appropriate 8-bit floating point value.

Our design works for every 12-bit two's-complement number except for one special case, the minimum value of 12-bit two's-complement which is -2048. We included code to handle this special case in the submodule that converts two's-complement to sign magnitude. Also, there was a special case that we had to consider for the third submodule that rounds the floating point value. If the exponent, significand, and fifth bit were all 1's, then the resulting increments to the significand and exponent would cause the exponent to overflow to 000. We prevent this from happening by including code that sets the significand to 1111 and the exponent to 111 in the case of exponent overflow.