CS M152A Lab 2
Floating Point Conversion

HUAN-SONY NGUYEN

Ryan Trihernawan (904063131)
Walter Qian (204301927)

Introduction

The purpose of Lab 3 is to design a circuit that converts a 12-bit linear encoding to into a compounded 8-bit Floating Point Representation with some loss of data. More specifically our combinatorial circuit, called FPCVT, will take the 12-bit input and output a sign bit, a 3-bit exponent and a 4-bit mantissa (also called significand).

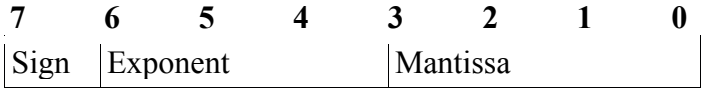

Figure 1: 8-bit Floating Point Figure

$$V = (-1)^S \times F \times 2^E$$
Figure 2: Floating Point Formula

The first figure represents the structure of the 8-bit floating point. The second figure is the floating point formula. S is the sign bit. F is the Mantissa. E is the exponent. In order to convert the 12-bit two's complement we create a circuit with 3 modules. The first module transforms the two's complement into sign magnitude. The second module then converts it to our floating point. The final module is responsible for rounding the 8-bits to the final answer. This report will go over each of the modules in detail.

Design Description

The overall circuit has a 12-bit two's complement as input and outputs a sign bit, a 3-bit exponent and a 4-bit mantissa. The first module is responsible for extracting the sign bit and converting the two's complement to sign magnitude. The second module determines the exponent from leading zeros, the mantissa from the next 4 bits and a fifth bit used for rounding. Extracting the exponent and mantissa will give us the correct floating point about half the time. Using the fifth bit guarantees we have the most accurate floating point representation. The third and final module will use the fifth bit to round the exponent and mantissa. The three modules will be explained in detail and the circuit schematic can be seen in the drawing below.

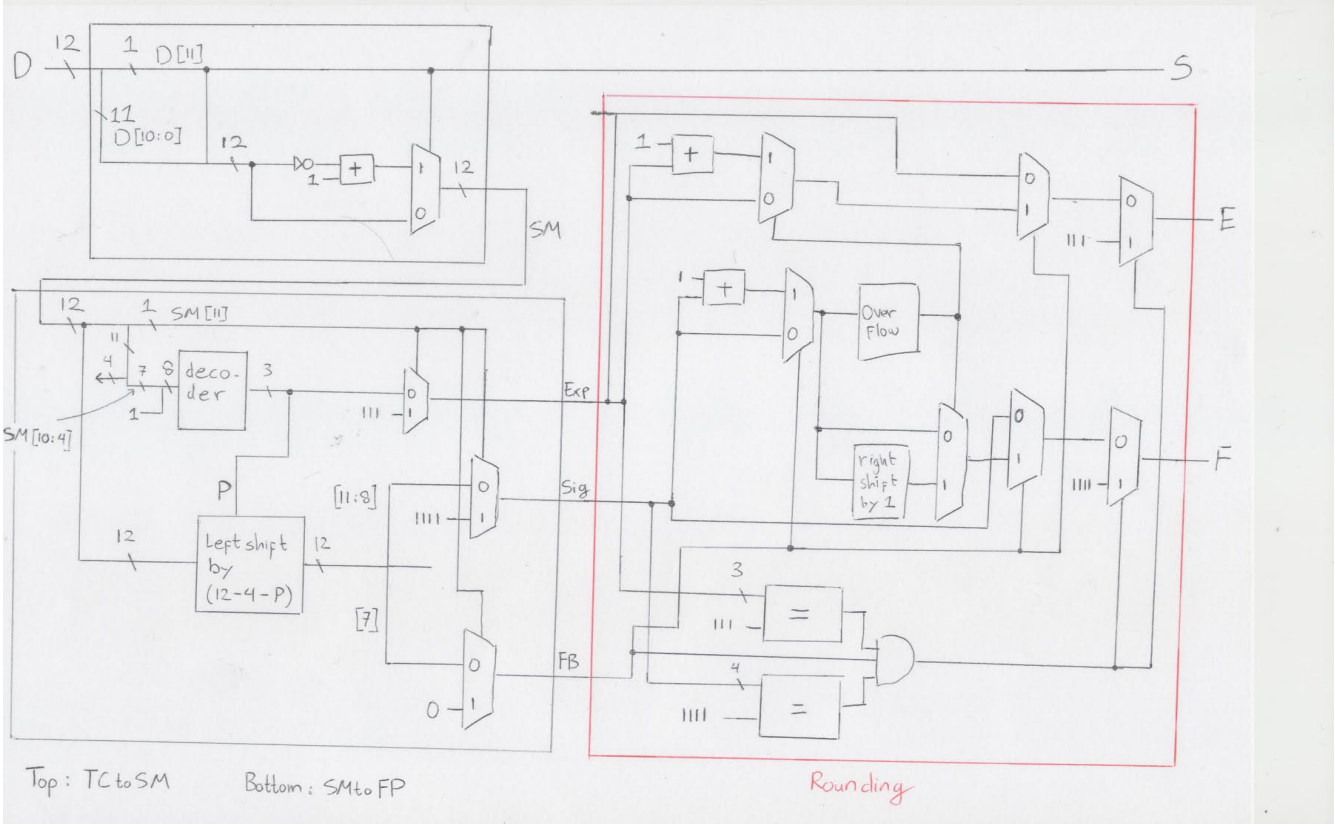

Figure 3: Circuit Schematic

The first block takes a 12-bit two's complement as input and outputs a 12-bit sign magnitude and sign bit. It is illustrated in the top left of the schematic. We take the sign bit of the input and enter it into a MUX. If the sign bit is 1 (negative) then we invert all the bits and add 1. Otherwise if the sign bit is 0, the input remains unchanged. The sign bit is then passed directly to the output of the entire circuit.

The next module takes the 12-bit sign magnitude and extracts the 8-bit floating point and is the bottom left of the schematic. The first step is to count the number of leading zeros, which gives us the exponent. The smaller the number of leading zeros, the greater the exponent. With one leading zero we get a maximum of 7 for the exponent. For every increment in leading zeros we reduce the exponent by 1. At 8 or more leading zeros the resulting exponent is 0. After we get the exponent we can get the 4-bits needed for the Mantissa. The Mantissa consists of the 4-bits immediately following the last leading zero. We also extract the fifth bit if possible for rounding purposes. We also take care of a special case for when the input is the greatest negative number (-2048). We use MUXes to check if the first bit of the input is 1 and set the values for exponent and Mantissa to 111 and 1111 respectively for this special case.

The final module in our circuit takes the 3-bit exponent, the 4-bit mantissa and the fifth bit as inputs and outputs the rounded 3-bit exponent and 4-bit mantissa. It is the right half of the schematic. We use MUXes with the fifth bit as input to determine if rounding is required. If the fifth bit is 0 then we do not need to round. However when it is 1 we add 1 to the Mantissa. If this causes the Mantissa to overflow we increment the exponent to account for the overflow. This occurs when the Mantissa is 1111, which would normally result in a 10000 mantissa. Since we only have 4 bits we increment the exponent by one and shift the mantissa to 1000. If the exponent is already 111 then we do not need to

round since it is already the largest floating point representation. Finally we output the rounded exponent and mantissa.

Simulation Documentation

We will give test cases and discuss expected outcomes and special cases for the circuit and each of the three modules.

Initial Value	12-bit Linear	8-bit Floating Point	Output Value	Percent Error
1120	[0100 0110 0000]	[0 111 1001]	1152	2.86%
-2048	[1000 0000 0000]	[1 111 1111]	-1920	6.25%
46	[0000 0010 1110]	[0 010 1100]	48	4.35%
-422	[1110 0101 1010]	[1 101 1101]	-416	1.42%
128	[0000 0111 1101]	[0 100 1000]	128	2.40%

Table 1: Circuit Inputs and Outputs

Table 1 above illustrates the test cases for the overall circuit. There are five test cases and initial values and 12-bit linear encodings are provided. The 8-bit floating point is taken from the modules below and converted to a decimal value to see the percent error. Each of the outputs is as expected, as they are the closest possible 8-bit floating point value. The second test case of -2048 is a special case and is represented by the smallest possible 8-bit floating point value of -1920. We will look at the modules more carefully now. The modules use the same test cases.

Two's Complement (Input)	Sign Magnitude (Output)	Sign Bit (Output)
[0100 0110 0000]	[0100 0110 0000]	0
[1000 0000 0000]	[1000 0000 0000]	1
[0000 0010 1110]	[0000 0010 1110]	0
[1110 0101 1010]	[0001 1010 0110]	1
[0000 0111 1101]	[0000 0111 1101]	0

Table 2: Two's complement to sign magnitude module input and outputs

The first module converts two's complement to sign magnitude. The sign bit is the first input bit. All the outputs are as expected. The second case is the special case as -2048 remains the same. We have a special case for it in the second module. All the other test cases are correctly converted to the sign magnitude. The sign magnitude output is passed to the second module while the sign bit is directly sent to the circuit output.

Sign Magnitude (Input)	Exponent (Output)	Mantissa (Output)	Fifth Bit (Output)
[0100 0110 0000]	[111]	[1000]	1
[1000 0000 0000]	[111]	[1111]	0
[0000 0010 1110]	[010]	[1011]	1

[0001 1010 0110]	[101]	[1101]	0
[0000 0111 1101]	[011]	[1111]	1

Table 3: Sign magnitude to floating point module inputs and outputs

Table 3 shows the test cases for the second module. The Mantissa is simply the first four bits after the leading zeros and the fifth bit is used for rounding. The exponent is taken from the number of leading zeros. All the outputs are as expected. The second case is the special case. We use MUXes to determine if the sign magnitude is -2048. We correctly convert the exponent and mantissa to all ones as it is the closest approximation in 8-bit floating point. All the outputs are sent to the third module.

Exponent (Input)	Mantissa (Input)	Fifth Bit (Input)	Exponent (Output)	Mantissa (Output)
[111]	[1000]	1	[111]	[1001]
[111]	[1111]	0	[111]	[1111]
[010]	[1011]	1	[010]	[1100]
[101]	[1101]	0	[101]	[1101]
[011]	[1111]	1	[100]	[1000]

Table 4: Rounding module inputs and outputs

Above are the test cases for the final module. The fifth bit determines if rounding is required. In the case of rounding the mantissa is incremented first and in the case of overflow the exponent is incremented as well. All the outputs are as expected. The fifth case demonstrates the case of mantissa overflow. The mantissa is initially 1111 and the fifth bit is 0. Thus the exponent increases from 3 to 4 and the mantissa becomes 1000. Another special case is when the exponent is 111 and the mantissa is 1111. This is already the largest possible floating point so we do not round.

Conclusion

The floating point converter successfully converts a 12-bit input into its 8-bit floating point counterpart in the form of a sign bit, 3-bit exponent, and 4-bit mantissa as specified in the lab manual. Some difficulties were encountered during the design of the three modules that make up the circuit. The first module design is straightforward because it consists of only an adder and a multiplexer. The second module design is not as straightforward as the first one due to the need to handle the special case for the 12-bit input when the most significant bit is 1 and the remaining bits are 0. The complement of this input is still a negative number and thus it needs to be handled specially as if the module produces exponent bits, mantissa bits, and fifth bit that correspond to the most negative floating point number. Programming the special case was more intuitive than designing the schematic because Verilog “if statements” are more easily visualized than multiplexers are. Hence, the “if statements” were coded before the schematic of the multiplexers to handle the special case was completed. The third and last module is the most challenging one due to the multiple interpretations of the schematic into a program. This is another case where programming the logic was more intuitive than designing the schematic due to the necessity of utilizing multiple multiplexers to handle different special cases. The program was designed to closely resemble its schematic counterpart, hence the use of explicit “right shifts” (>>). In conclusion, the major confusion that often arose throughout the lab is the need to fully design a schematic before programming the design in Verilog.