Adam Cole
UID: [redacted]
Project 2 Lab Report

1. Introduction and Requirements

For Project 2, our goal was to implement a 13-bit linear encoding of an analog signal into a compounded 9-bit floating point (FP) representation. The 13-bit data comes into our combinatorial circuit as a Decimal, in two's complement representation. In this schematic, the most significant bit represents the sign of the number, whether positive or negative, and the remainder of the bits become the magnitude of the number. In Floating Point Representation, the 13-bit input must be changed into the following 9-bit format:

8	7	6	5	4	3	2	1	0
S	E			F				

In this representation, the most significant bit represents the sign of the float, and the rest of the binary digits evaluate to the value of the number with the formula below:

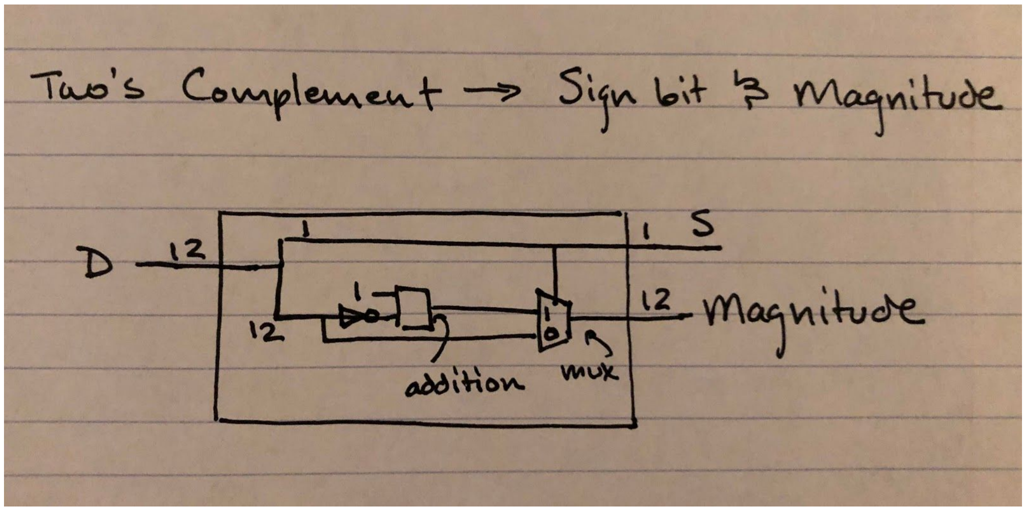
$$\mathbf{V = (-1)^S \times F \times 2^E}$$

As we can see, the three "E" bits are used as an exponent, and the 5 "F" bits are used as the fraction that is multiplied against 2 raised to the E power. To convert the decimal in two's complement into floating point, we created a combinatorial circuit in Xilinx ISE software. Designing and simulating this circuit, we were able to create the converter using three submodules.

2. Design Description

2.1 Two's Complement to Sign Bit and Magnitude

Our first submodule extracted the sign bit from the two's complement number and then updated the input based on the sign. If the most significant bit was 0, the number is positive and the input is left alone, passed on to the next submodule. If the most significant bit was 1, the input was complemented and added to 1, changing the value to be positive. In this way, we are able to retrieve the magnitude from the negative representation of the number. The schematic of our first submodule is shown below:



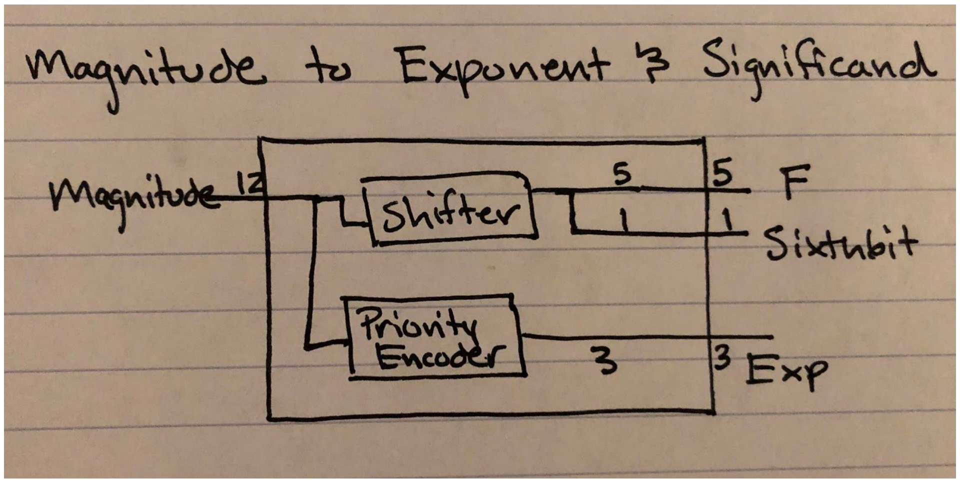
As you can see, the two's complement number is passed into the module, and the most significant bit is checked. If one, the value is complemented and added to one, otherwise the magnitude stays the same.

2.2 Magnitude to Exponent and Significand

The second submodule I created took the magnitude output from the first submodule and split it into three "E" bits and five "F" bits to be used in the floating point representation. A third output was used, however, in order to help in our next module for correctly rounding. The exponent of the number is derived from the number of leading 0's in the magnitude. For example, the chart below gives the conversions:

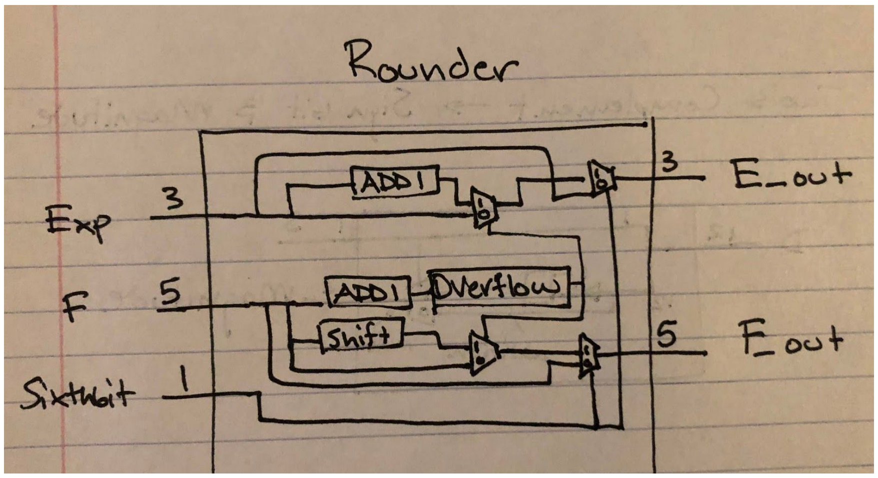
Leading Zeroes	Exponent
1	7
2	6
3	5
4	4
5	3
6	2
7	1
≥8	0

The next output, the 5 "F" bits, are the five bits that immediately follow the last leading 0 in the magnitude representation. Therefore, this third output called SixthBit is the sixth bit after the last leading 0 - and it gives our converted valuable information if to round our converted float up or not. The schematic of our second module is given below:



2.3 Rounding

Our third and final submodule is responsible for rounding the output float to be as accurate as possible to our two's complement representation. This submodule takes in the second submodule's E and F bits, as well as SixthBit. The outputs are our final E and F bits after they have been adjusted to rounding. If the SixthBit is one, then we round our float up, and if it is zero, we round our float down (and keep it as it is). We implement rounding by adding the sixth bit to our F bits. If this addition causes our F bits to take up 6 bits instead of 5, this overflow is pushed into the exponent. In this case, we right shift F by 1 bit, truncating the F bits back into 5 bits, and add 1 to the exponent bits. As our third submodule outputs our rounded E and F bits, we have finished calculating our floating point value. The schematic of our third module is given below:



2.4 All Together - Decimal to Floating Point Converter

After developing the three submodules, the only step left is to piece them together. From the first submodule, we retrieved the sign bit as well as the magnitude. Using this magnitude, we split it up into exponential bits and the “F” significand bits. Our third module accurately rounds our split, and our float is reached.

3. Simulation Documentation

3.1 Two’s Complement to Sign Bit and Magnitude

Sample Test Cases:

D (Input)	Mag (Output)	S (Output)
0000000000000	0000000000000	0
0111111111111	0111111111111	0
1111111111111	0000000000001	1
1000000000000	1000000000000	1

While simulating our first submodule, we noticed that when testing -4096, the magnitude result appears equivalent to +4096. I note this and handle it in future submodules.

3.2 Magnitude to Exponent and Significand

Sample Test Cases:

Mag (Input)	E (Output)	F (Output)	SixthBit(output)
0000000000001	000	0001	0
0001101111101	101	11011	1
1111111111111	111	11111	0
1000000000000 *	111	1111	0

3.3 Rounding

Sample Test Cases Below. When the rounding tries to overflow the floating point, our module caps the floating point value at `FLOAT_MAX`.

E_in	F_in	SixthBit	E_out	F_out
000	00000	1	000	00001
111	11111	1	111	11111
001	11111	0	001	11111
100	11111	1	101	10000

3.4 Decimal to Floating Point Converter

Combining all the above pieces, we were able to successfully convert any Two's Complement decimal number into an equivalent Floating Point number. Some of the many test cases are included below:

#	D (input)	S (output)	E (output)	F (output)
1	0000000000000	0	000	00000
2	1000000000000	1	111	11111
3	0111111111111	0	111	11111
4	1111110000001	1	011	10000
5	0000000101101	0	001	10111

These 5 test cases cover all functionality of our three submodules, including cases where rounding is necessary, negative numbers (test case #4).

4. Conclusion and Synthesis

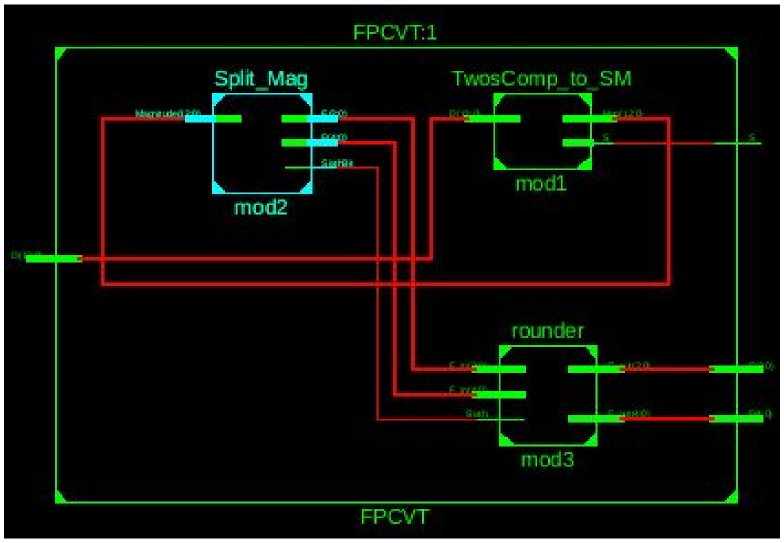
Our design successfully converts 13-bit two's complement numbers into 9-bit floating point representations. After running the synthesis on our high level module, the following output summary was given:

```
*                          Design Summary                          *
==================================================================

Top Level Output File Name         : FPCVT.ngc

Primitive and Black Box Usage:
------------------------------
# BELS                             : 91
#      GND                         : 1
#      INV                         : 12
#      LUT1                        : 1
#      LUT2                        : 1
#      LUT3                        : 12
#      LUT4                        : 3
#      LUT5                        : 9
#      LUT6                        : 26
#      MUXCY                       : 12
#      VCC                         : 1
#      XORCY                       : 13
# FlipFlops/Latches                : 6
#      LDC_1                       : 1
#      LDP_1                       : 5
# IO Buffers                       : 22
#      IBUF                        : 13
#      OBUF                        : 9
```

And when the RTL Schematic was created through Synthesis, the schematic matched our high level design:


FPCVT:1
Split_Mag
mod2
TwosComp_to_SM
S
mod1
rounder
mod3
FPCVT