CS M152A Project 1 Report: Floating Point Conversion

Belle Lerdworatawee, Lab 1
TA: Ananya Ravikumar
Jan 24, 2021

Introduction and Requirement:

The purpose of this lab was to introduce students to Xilinx ISE software and Verilog by designing and testing a combinational circuit. The circuit compresses a 13-bit binary encoding of an analog signal to a 9-bit Floating Point Representation (FPR). The project requires us to use a simplified FPR that consists of 1-bit Sign Bit Representation, a 3-bit Exponent, and a 5-bit Significand. The conversion from the FPR back to the 13-bit linear encoding is as follows: $V = (-1)^S \times F \times 2^E$. While typical FPRs require the significand to be normalized (most significant bit [MSB] must be 1), the project did not ask us to account for this. Instead, we needed to focus on accurately representing numbers in the FPR that can't be fully represented by 5 bits as well as treating edge cases correctly. After designing the compression module, we write a test bench and simulate the module with various inputs to test the accuracy and reliability of our module.

Design Description

For the design, I roughly followed the sample block diagram provided in the project description.

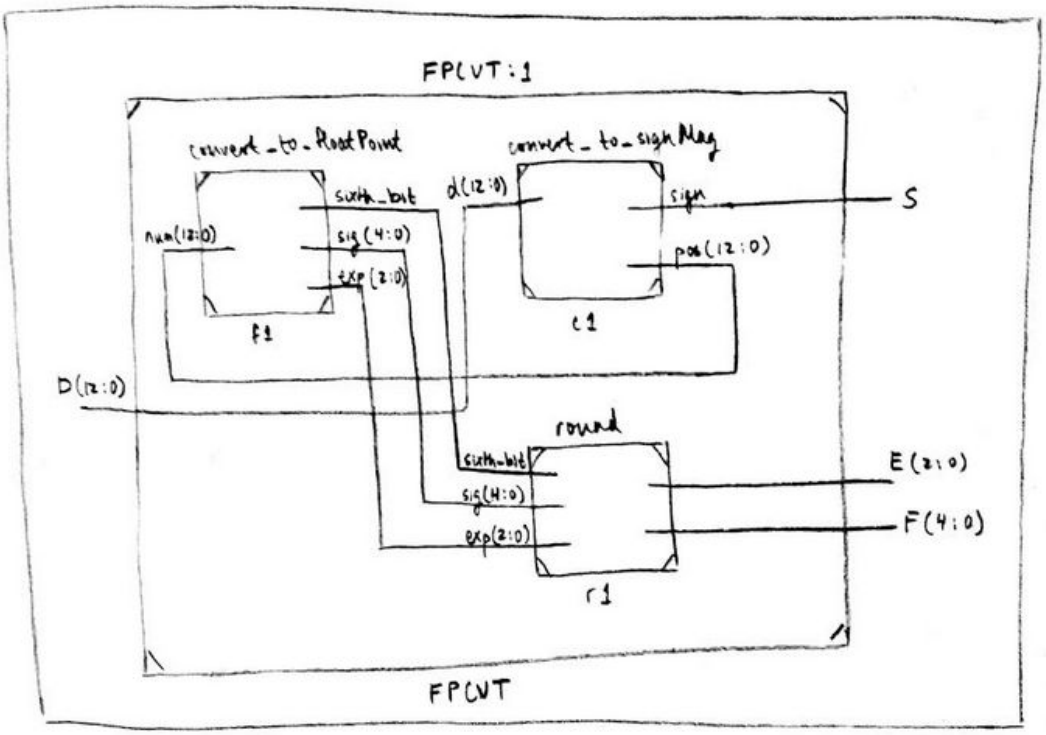


Figure 1: hand drawn overview of the top module, FPCVT

I created 3 submodules—convert_to_signMag, convert_to_floatPoint, and round—and placed them within the FPCVT module. As an overview, the overarching module relies entirely on the submodules to convert the linear encoding to FPR. The FPCVT takes in one input, 13-bit D, and outputs 3 things: a 1-bit Sign Bit, a 3-bit Exponent, and a 5-bit Significand. It feeds its input D into the convert_to_signMag in order to first transform any negative numbers to their absolute value, as well as handle any edge cases. Then it sets the 1-bit Sign Bit and sends the converted linear encoding to the convert_to_floatPoint module. This module then builds the exponent and significand from the converted linear encoding. It outputs those two pieces of information and an additional "sixth_bit" to the last module, round. Round then determines based on the sixth_bit whether or not the significand needs to be rounded and does so accordingly. It sets the final E and F of the FPCVT module.

Submodules:

1. convert_to_signMag — As the name suggests, the purpose of this module is to convert numbers from two's complement to signed magnitude representation. This module takes in one 13-bit input, D, and outputs a 1-bit Sign Bit, and a 13-bit number. The input binary strings with an MSB of 1 are treated as negative numbers. This module converts negative numbers to their absolute value while leaving positive numbers untouched. I used an if statement in an always block to check if the input is positive or negative. Then I use another if statement in the same block to check for the edge case of the input being -4096. If it is, the output is set to the largest positive number 4095.
2. convert_to_floatPoint — This module extracts the exponent and significand from the now positive number from convert_to_signMag. It takes in a 13-bit input and outputs a 5-bit significand, 3-bit exponent, and 1-bit flag. From the project spec, the significand starts from the first bit on the left that is a 1 (provided that the linear encoding has less than 8 leading zeros). The bit right after the significand potion (the sixth one) in the linear encoding determines if the significand will be rounded up or not. I used a case statement in an always block to implement a priority encoder that counts the leading zeros. Based on which case a number falls into, I also right-shift the input accordingly to get the correct bits for the significand, and save the last bit for the flag. The default case handles latch issues by dealing with numbers with more than 8 leading zeros. It simply captures the last 6 bits of the linear encoding and sets the exponent to 0.
3. round — Lastly, this module finishes the compression by rounding the significand if necessary. It takes in a 5-bit significand, 3-bit exponent, and 1-bit flag (sixth_bit), and outputs the 5-bit significand F and 3-bit exponent E of the FPCVT module. A high flag bit indicates that the significand needs to be rounded up by adding one. I implemented this through nested if statements in an always block. The outermost if statement checks if the flag is high or not. If the flag is high, then I have another if statement inside that

checks for potential overflow by adding 1 to the significand and exponent. In cases of overflow, I hardcode the exponent and significant accordingly. If the significand overflows, I add 1 to the exponent and set the significand to 1_0000. I handle the edge case where both overflow by setting the exponent to 111 and the significand to 1_1111. otherwise, I add one to the significand or leave it be if the flag was low.

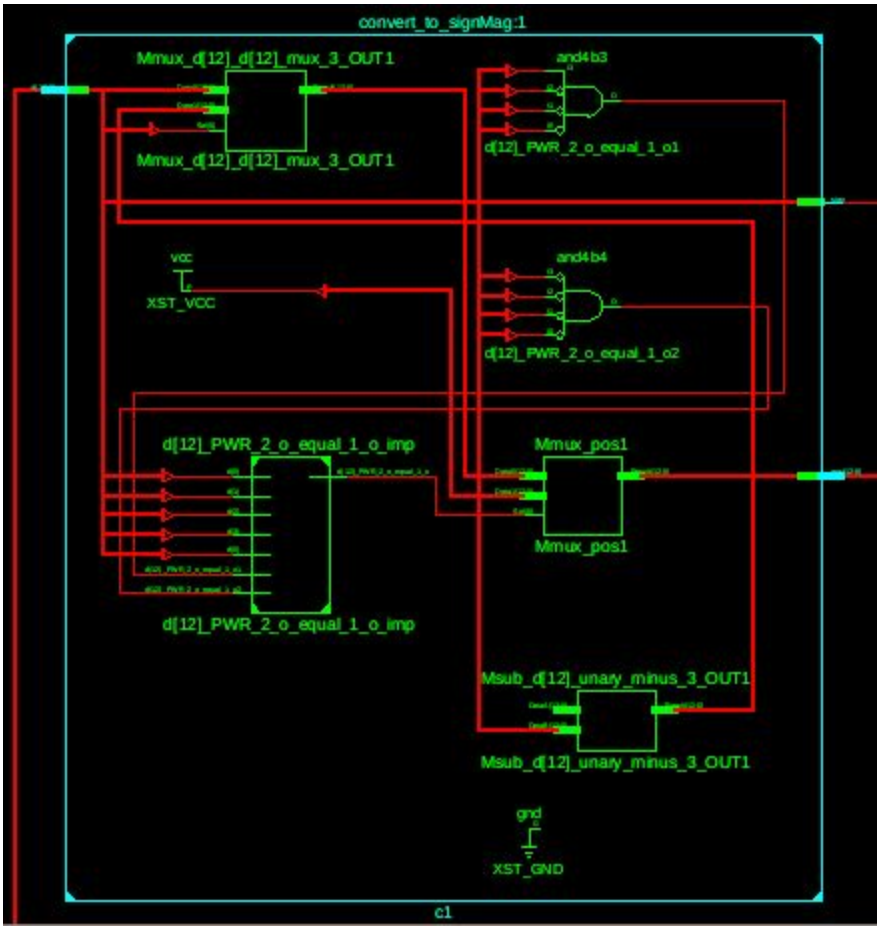


Figure 2: ISE generated schematic of the convert_to_signMag module

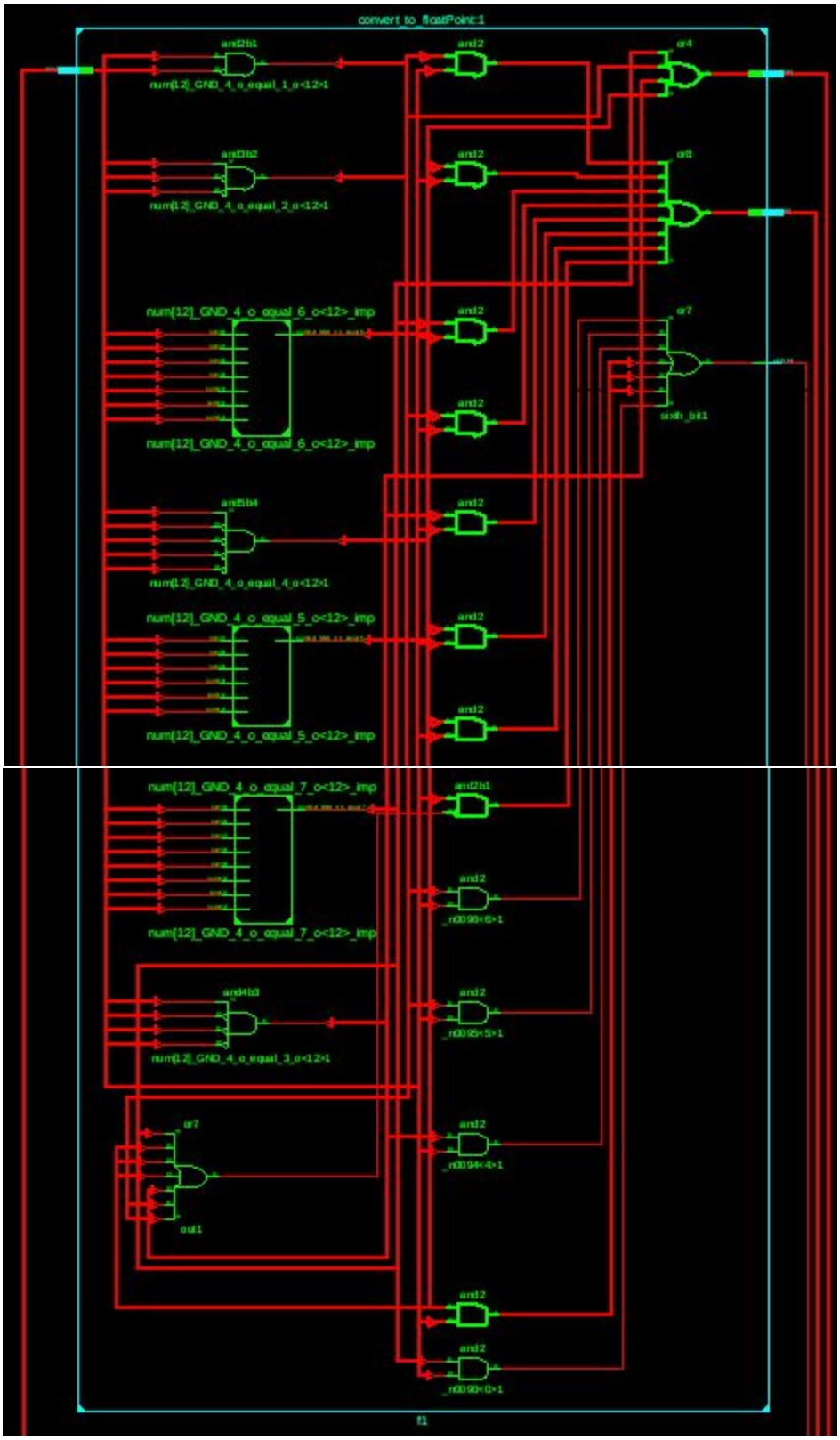


Figure 3: simplified hand drawn convert_to_floatPoint mod

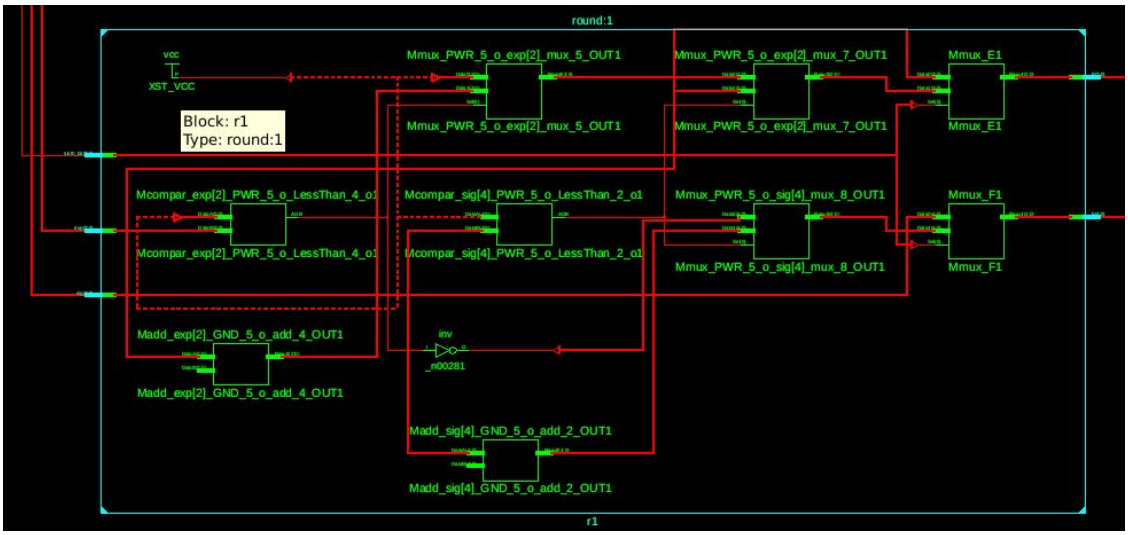
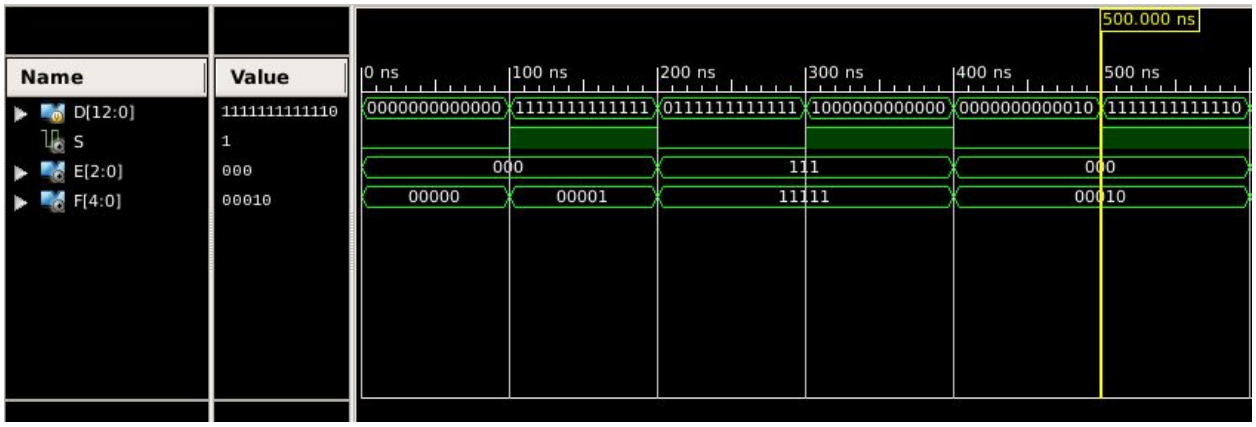


Figure 3: simplified hand drawn round module

Simulation Documentation

In order to determine if the FPCVT module converted the 13-bit linear encodings to FPR accurately, I tested it with different, regular inputs first to see if it behaved as expected. Then I fed the module various edge cases to test if those were handled correctly. For the different inputs of D, I observed the changes in the outputs S, E, and F. I tested for the most negative and positive numbers, more than 8 leading zeros, numbers that required rounding in the significand or the exponent or both, as well as regular positive and negative numbers.



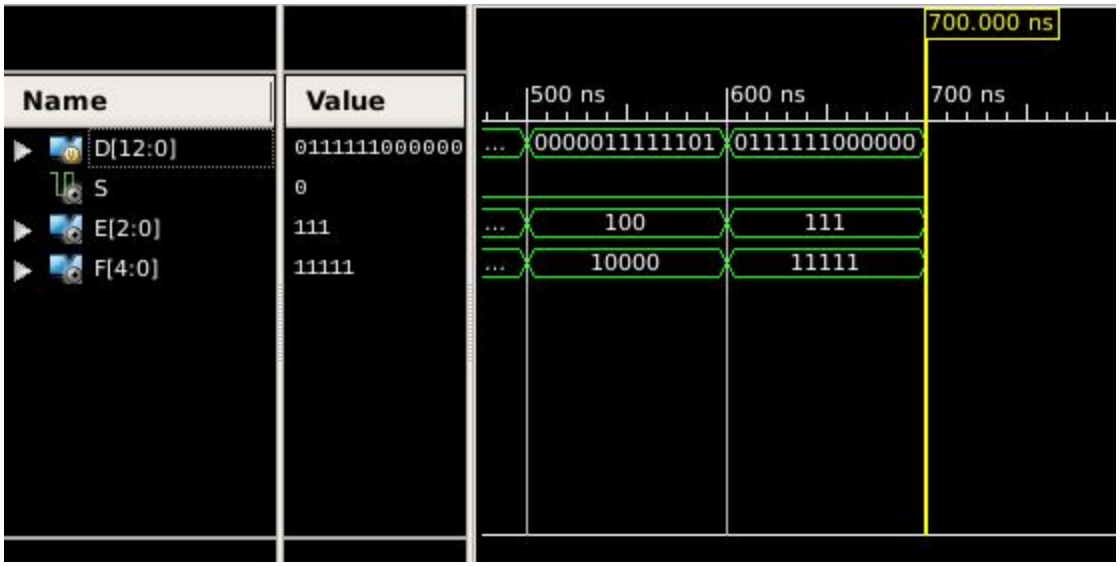


Figure 5: Simulation Waveform for FPCVT

First and foremost, S is the easiest output to test. I tried both negative and positive numbers to see if S would correctly be 1 for negative and positive. I tried 4 different inputs (0, -4096, 4095, -0) to see if S was high for -4096 and -0 and low otherwise, in which it was. Even though -0 isn't a valid input, I was just using it to see if S was set properly. Next, we can analyze the input at 600ns as an example. D was 0_1111_1100_0000 (4032); this test case will have an overflow in both the significand and the exponent. Since the number is positive, there's no need for conversion to sign magnitude. Hence from observation, the significand will be 1_1111 and the sixth_bit flag is a 1 which signals a need for rounding. There's only 1 leading zero which means the exponent should be 111. We can observe from the waveform that S is 0, E is 111, and F is 1_1111 as expected.

A bug that I found during simulation was originally F and E were a series of don't cares. The error lay in my implementation of the convert_to_floatPoint module. I had originally left the assignment of the significand and the sixth_bit flag after the case statement block. That was due to my unfamiliarity with Verilog and my misconception that Verilog instructions are executed sequentially. I fixed this by assigning the significand and sixth_bit inside each case statement.

FPCVT Project Status (01/24/2021 - 19:29:08)			
Project File:	Project1.xise	**Parser Errors:**	No Errors
Module Name:	FPCVT	**Implementation State:**	Synthesized
Target Device:	xc6slx16-3csg324	**• Errors:**	No Errors
Product Version:	ISE 14.7	**• Warnings:**	2 Warnings (0 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	67	9112	0%
Number of fully used LUT-FF pairs	0	67	0%
Number of bonded IOBs	22	232	9%

Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Sun Jan 24 19:30:05 2021	0	2 Warnings (0 new)	0
Translation Report	Out of Date	Sat Jan 23 23:02:29 2021	0	0	0
Map Report	Out of Date	Sat Jan 23 23:02:55 2021	0	1 Warning (1 new)	6 Infos (6 new)
Place and Route Report	Out of Date	Sat Jan 23 23:03:12 2021	0	0	2 Infos (2 new)
Power Report					
Post-PAR Static Timing Report	Out of Date	Sat Jan 23 23:03:25 2021	0	0	4 Infos (4 new)
Bitgen Report					

Secondary Reports			[-]
Report Name	**Status**	**Generated**	
ISIM Simulator Log	Out of Date	Sun Jan 24 19:30:41 2021	

Date Generated: 01/24/2021 - 20:57:39

Figure 6: ISE Design Overview Summary Report

There are no errors with my design. The two warnings make note about a truncation error during the rounding step. However I take care of this by writing an if statement to filter out the

cases where the significand or exponent could overflow and thus be truncated. I also hardcode the significand and exponent in these cases so truncation is not an issue.

Conclusion

In summary, I designed a module that compresses 13-bit linear encodings to 9-bit FPR's of that encoding. I split the larger problem into smaller subproblems that are handled by submodules. This made it easier to isolate issues and fix them accordingly. I used the Lab and Q&A sessions to figure out what steps I needed to take to get started. The main difficulties I encountered were learning how to handle common syntax errors and what generates them. I also needed to develop an intuition behind when to use wires and when to use registers. These I dealt with by reading Verilog tutorials and reviewing the Verilog slides on CCLE. I have just one suggestion: it would help to provide a concrete list of items needed on our report. I know that the project description and syllabus both have a paragraph explaining the requirements, but sometimes there would be tidbits said during Lab or Office Hours that were different from the written requirements. This would just help provide further clarification! Other than that, I learned the fundamentals of Verilog from this lab and gained the experience of troubleshooting and debugging common Verilog bugs.