CS M152A Lab 1: Floating Point Conversion Report

Arnold Pfahnl

TA: Mohit Garg, Winter 2021

Contents

1 Introduction

In this lab we implement and test a combinational circuit that converts a 13-bit linear encoding into a 9-bit floating point representation. The module for floating point conversion must be called $FPCVT$, and there are specific requirements for input and output. D is the 13-bit input data in two's complement format. S is a single bit output that represents the sign. E is a 3-bit exponent output of the floating point representation. F is the 5-bit significand output of the floating point representation. The value represented by the floating point converter output is based on the following format:

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

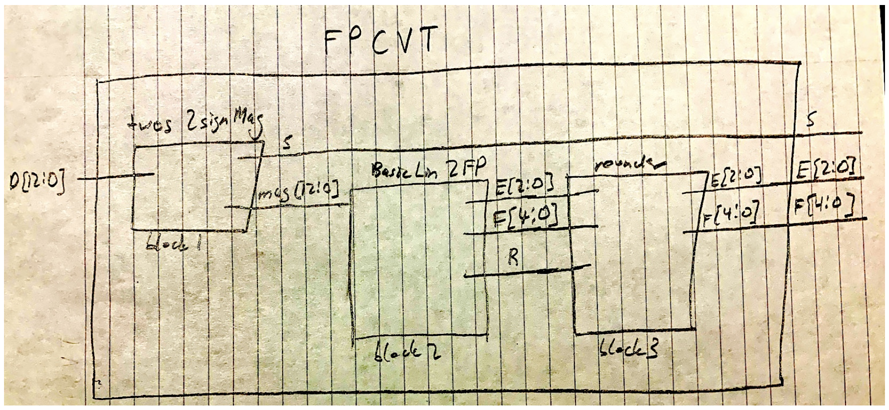
Since the precision of the 13-bit input can't be preserved with the 9-bit output, rounding is also a required feature.

2 Design: FPCVT Module and Submodules Implementation

I created the floating point conversion module, $FPCVT$, with three submodules:

- $twos2SignMag$: a module to convert the 2s' complement input into a sign bit and a magnitude. This is where the top level sign bit comes from.
- $basicLin2FP$: a module that performs a basic floating point conversion.
- $rounder$: a module that rounds to the correct value. This is where the top level exponent and significand outputs come from.

At the very top level, D is a 13-bit input, S is a one-bit sign output, E is a 3-bit exponent output, and F is a 5-bit significand output.



2.1 twos2SignMag

The *twos2SignMag* submodule, which implements *block*1 in the schematic, takes one input, the 13-bit input D to $FPCVT$. Since we are dealing with 2s' complement, the sign bit is the most significant bit (MSB), and the value of the MSB is assigned to the output sign bit S. If the sign bit isn't set, the magnitude is the same as D, but if the sign bit is set, the magnitude is D negated and incremented by 1.

For the corner case that the most negative value is encountered ('b1000000000000), after adding one, the magnitude value will overflow into the sign bit. To ensure that the floating point representation will maximize the exponent and significand, my implementation checks if the magnitude sign bit is set, and then sets all magnitude bits to 1.

2.2 basicLin2FP

The *basicLin2FP* submodule, which implements *block*2 in the schematic, takes one input, the 13-bit magnitude from *twos2SignMag*.

The first part of this submodule is counting the number of leading zeroes, which becomes the exponent. To do this, I used a priority encoder to count leading zeroes. The implementation is a series of if/else if statements that checks if a bit is set starting from the second-most significant bit (since the MSB is the sign bit, which is always zero for a magnitude) toward the least significant bit. The first set bit encountered tells us that the leading zeroes start from the next significant bit (the previous bit).

Once the exponent is determined, there are two more outputs: the significand, and the bit after the significand, R (if it exists). R is used as an input into the *rounder* submodule to determine the rounding needed. If the exponent is 0, the significand is just the first 5 bits of the magnitude, and R is set to 0. For all other exponents, the magnitude is first shifted right one less than the exponent (leading zeroes count) times. The first bit of the resulting value is stored as R, and then the significand is determined as the magnitude shifted the full exponent amount.

2.3 rounder

The *rounder* submodule, which implements *block*3 in the schematic, takes three inputs: the naïve exponent, naïve significand and R from *basicLin2FP*. If R is set, then we round. Rounding first begins by adding one to the naïve significand to produce the first rounded significand. If the rounded significand is less than the naïve significand, an overflow occurred. With a significand overflow, the rounded exponent becomes the naïve exponent incremented by one, and the rounded significand becomes the old rounded significand shifted right by one (with a one tacked on to the MSB). In the event that even the exponent overflows, the exponent and significand are set to their largest possible values (all bits set).

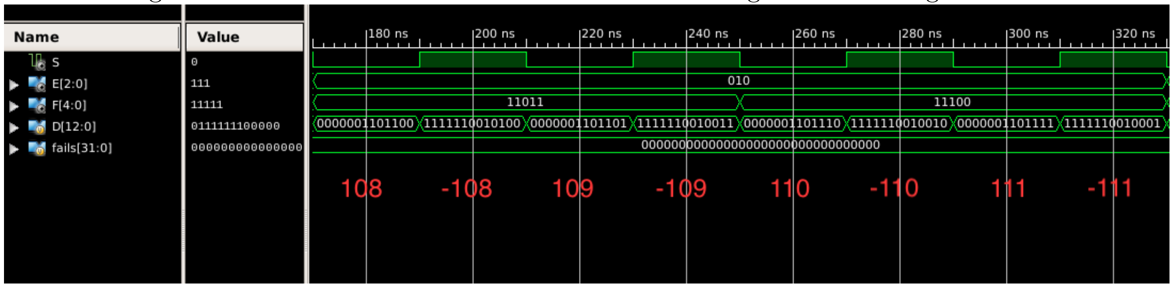
3 Simulation

To make testing easier, I created a Verilog task called *assert* that checks if the sign bit, exponent, and significand match up with their expected values for a given input D. If any mismatch occurs, the floating point bits are printed out for debugging. Additionally, my testbench counts the number of failures and reports the total at the end. With my final implementation of the $FPCVT$ module, all tests resulted in the expected output with no failures.

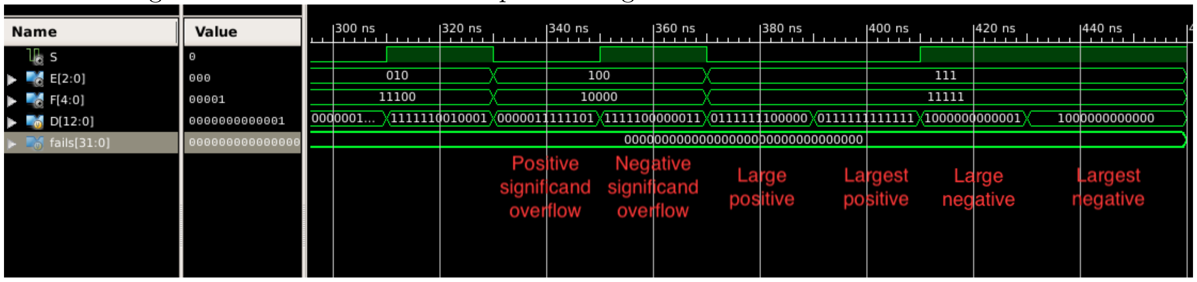
The actual groupings of test cases I used include,

1. Basic positive and negative values that don't get rounded. This checks the base functionality of the module.
2. Basic rounding examples with no overflow condition. This checks the basic rounding capabilities of the module.
3. Rounding with significand overflow. This checks one overflow corner case.
4. Rounding with significand and exponent overflow. This checks another overflow corner case. This also tests the most negative number.

The following is the waveform for some basic test cases including basic rounding:



The following is the waveform for some important edge cases:



4 ISE Design Summary

FPCVT Project Status (01/21/2021 - 04:15:23)			
Project File:	FPCVT.xise	**Parser Errors:**	No Errors
Module Name:	FPCVT	**Implementation State:**	Placed and Routed
Target Device:	xc6slx16-3csg324	• **Errors:**	No Errors
Product Version:	ISE 14.7	• **Warnings:**	No Warnings
Design Goal:	Balanced	• **Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	
Environment:	System Settings	• **Final Timing Score:**	0 (Timing Report)

Device Utilization Summary [-]				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	0	18,224	0%	
Number of Slice LUTs	50	9,112	1%	
Number used as logic	50	9,112	1%	
Number using O6 output only	34			
Number using O5 output only	12			
Number using O5 and O6	4			
Number used as ROM	0			
Number used as Memory	0	2,176	0%	
Number of occupied Slices	23	2,278	1%	
Number of MUXCYs used	16	4,556	1%	
Number of LUT Flip Flop pairs used	50			
Number with an unused Flip Flop	50	50	100%	
Number with an unused LUT	0	50	0%	
Number of fully used LUT-FF pairs	0	50	0%	
Number of slice register sites lost to control set restrictions	0	18,224	0%	
Number of bonded IOBs	22	232	9%	
Number of RAMB16BWERs	0	32	0%	
Number of RAMB8BWERs	0	64	0%	
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%	
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%	
Number of BUFG/BUFGMUXs	0	16	0%	
Number of DCM/DCM_CLKGENs	0	4	0%	
Number of ILOGIC2/ISERDES2s	0	248	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	248	0%	
Number of OLOGIC2/OSERDES2s	0	248	0%	
Number of BSCANs	0	4	0%	
Number of BUFHs	0	128	0%	

Number of BUFPLLs	0	8	0%	
Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	0	32	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCILOGICSEs	0	2	0%	
Number of PLL_ADVs	0	2	0%	
Number of PMVs	0	1	0%	
Number of STARTUPs	0	1	0%	
Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	3.09			

Performance Summary			**[-]**
Final Timing Score:	0 (Setup: 0, Hold: 0)	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:			

Detailed Reports **[-]**

Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Thu Jan 21 04:14:57 2021	0	0	0
Translation Report	Current	Thu Jan 21 04:15:03 2021	0	0	0
Map Report	Current	Thu Jan 21 04:15:10 2021	0	0	6 Infos (0 new)
Place and Route Report	Current	Thu Jan 21 04:15:17 2021	0	0	2 Infos (0 new)
Power Report					
Post-PAR Static Timing Report	Current	Thu Jan 21 04:15:21 2021	0	0	4 Infos (0 new)
Bitgen Report	Out of Date	Mon Jan 18 03:11:17 2021	0	0	0

Secondary Reports **[-]**

Report Name	**Status**	**Generated**
ISIM Simulator Log	Out of Date	Thu Jan 21 04:14:42 2021
WebTalk Report	Out of Date	Mon Jan 18 03:11:18 2021
WebTalk Log File	Out of Date	Mon Jan 18 03:11:19 2021

Date Generated: 01/21/2021 - 04:15:23

5 Conclusion

My floating point converter design follows roughly from the block diagram from the project specification. Not only does the module cover basic linear to floating point conversion, it also covers important cases such as rounding and overflow conditions that may occur. My testing was conducted to ensure that corner cases and general cases were covered.

The biggest difficulties I had with this lab were getting used to the syntax of Verilog, keeping track of the bits used for each operation to ensure proper outputs, and catching all the corner cases. In one of my earlier implementations of the $FPCVT$ module, I had encountered several warnings in the design summary that I wanted to eliminate. It turned out that there were cases where I was setting the value of a register to something with a longer bit length. To fix this, I either specified the bit length for a constant that was being assigned, or I created a temporary register from which the correct number of bits could be explicitly extracted.

In general, I thought this lab was well explained, and I have no suggestions for improvement. This was a good starter to the Verilog HDL and the Xilinx ISE, without adding too much complexity to overwhelm first-timers.