

CS M152A: Introductory Digital Design

Laboratory

**Lab #1**

**Floating Point Conversion**

Name: Kenny Chan, Kenny Luu, Kyle Reidy

UID: 004769092, 104823244, 104839297

Lab Section: 5

Due Date: April 26, 2018

**Introduction**

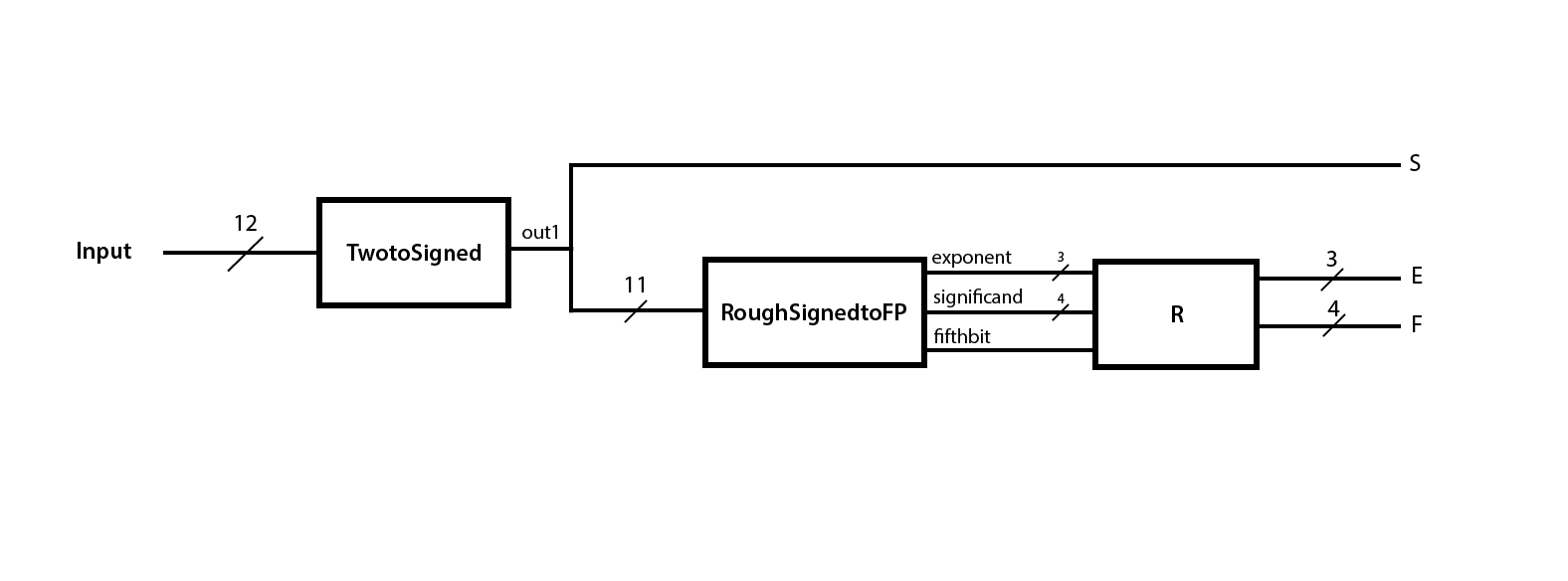
For this lab, our goal is to design a number conversion module (FPCVT) that converts a two’s complement binary number into its normalized floating point representation. This module takes in a twelve-bit binary number in two’s complement and outputs a sign bit (S), three exponent bits (E[3:0]), and four significand bits (F[3:0]). Mathematically, the floating point number can be computed using the formula:

**V = (-1)S x F x 2E**

and should be rounded as close as possible to the original two’s complement number.

To implement this, we will be using the ISE Design Suite to design the module through Verilog HDL. We will be simulating and debugging the behavior of this module through ISim Simulator. None of the code will be implemented into actual hardware. Instead, it will be simulated virtually.

**Design Description**

****

Our FPCVT module (Fig. 1) consists of three submodules: TwoToSigned, RoughSignedToFP, and R. The TwoToSigned module first converts the twelve-bit, two’s complement, representation of a number to its corresponding sign-magnitude representation. This sign-magnitude output is also twelve bits; the most significant bit is directly used as the final sign bit S and is not needed in the other two modules. The RoughSignedToFP submodule converts the magnitude of the number to its floating point representation, consisting of a three-bit exponent and four-bit significand. This submodule does no rounding, so it also outputs a “fifthbit,” which is usually the bit following the significand. Finally, the R submodule performs any rounding, based on the exponent, significand, and fifthbit from the previous module, and outputs the final E and F of the floating point number V.

**TwoToSigned**

The goal of the TwoToSigned submodule (Fig. 2) is to convert a twelve-bit two’s complement number to its sign-magnitude representation. The common method to finding the absolute value of a two’s complement number is to invert every bit except the most significant bit and add one if the number is negative (i.e. the sign bit is 1). Otherwise if the number is positive, do nothing. However, if the number is the negative minimum value (represented in binary by 1000 0000 0000), this method would fail due to overflow. So in particular, this case was hard-coded to output 1111 1111 1111 within our module. In the normal case, our submodule creates a mask consisting of eleven bits, all equal to the original sign bit of the two’s complement number. It performs an XOR operation with the other eleven bits of the two’s complement number and the created bitmask, and then adds the value of the sign bit. This works because an XOR with a mask of 0’s returns the original number while an XOR with a mask of 1’s inverts the number. The module outputs the unchanged sign bit as the final sign bit S and passes the eleven magnitude bits to the RoughSignedToFP module.

**RoughSignedToFP**

The RoughSignedToFP submodule (Fig. 3) converts the magnitude of the output generated by the TwoToSigned submodule to its rough representation by an exponent and a significand. The input of this module is the magnitude represented by the least significant 11 bits of the sign-magnitude number. Its output will be three exponent bits, four significand bits, and a fifthbit. Note that the exponent bits and significand bits may not represent the finalized floating point number because it may need to be rounded o a closer floating point representation.

The exponent value (exponent) is computed based on the number of leading zeros (Z) in the magnitude. To determine the number of leading zeros, we used if-else statements and read each bit sequentially, beginning with the most significant bit. If the value of Z is less than or equal to seven, the exponent should be equal to seven subtracted by Z. Otherwise, the number can be represented solely by the significand and the exponent value would be equal to zero.

After determining the number of leading zeros, the significand is simply equal to following four bits. As the name suggests, the fifthbit is the bit following the four bits of the significand. However, there is an exception when the exponent is zero. In this case, the significand is represented by the last four bits of the magnitude. Therefore, there is no fifth bit so we set the value of fifthbit to zero to prevent rounding.

**R**

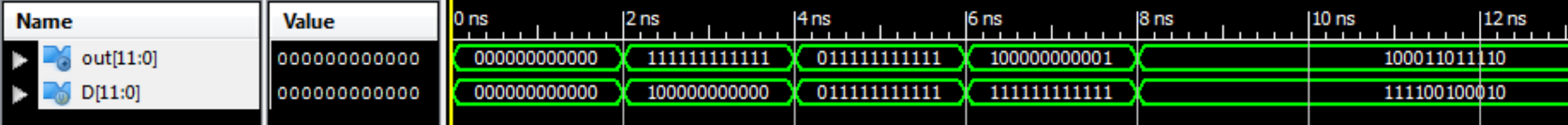
The final submodule, R, will use the fifthbit output from RoughSignedToFP to adjust the value of the significand and exponent, if necessary. Its inputs are the outputs of the RoughSignedToFP module: the exponent, significand and fifthbit. As a result, this module will output E and F, which correspond to the final floating point representation. In the general case, this submodule will just add the fifthbit to the significand which will account for rounding. In the special case where the significand is 1111, rounding up would cause an “overflow” to the value 10000, but our significand is only four bits. Hence, we set the significand to 1000 and increase the exponent by one to account for this “shift.” However, when both the significand and exponents are at their maximal values (1111 and 111, respectively), we simply return the original values because they represent the maximum magnitude that can be represented in floating point.

**Simulation Documentation**

**TwoToSigned**

To test this module, we used 0, Tmin, Tmax, -1, and an arbitrary value (-222). We chose these values because Tmin and Tmax would test the boundary conditions along with positive and negative numbers. We tested an arbitrary value to double check that the module was working as intended for intermediate values. With these tests, we assumed that a majority of intermediate values will also produce the correct results. This was justified when we passed the testbench provided by our instructor. We also tested 0, since this is a special value that can cause complications in signed magnitude. We had to hard-code the solution to -1, so we tested -1 specifically to make sure the behavior was correct.

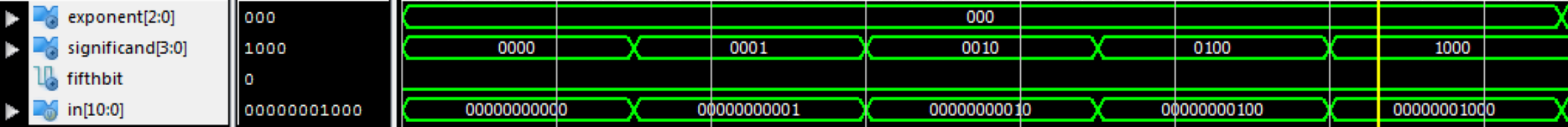
|  |  |  |
| --- | --- | --- |
| **Test Case (2’s Complement)** | **Result (One’s Complement)** | **Pass/No Pass** |
| D = 12’b0000 0000 0000 (0) | out = 12’b0000 0000 0000 (0) | Passed |
| D = 12’b1000 0000 0000 (Tmin) | out = 12’b1111 1111 1111 (-Smin) | Passed |
| D = 12’b0111 1111 1111 (Tmax) | out = 12’b0111 1111 1111 (Smax) | Passed |
| D = 12’b1111 1111 1111 (-1) | out = 12’b1000 0000 0001 (-1) | Passed |
| D = 12’b1111 0010 0010 (-222) | out = 12’b1000 1101 1110 (-222) | Passed |

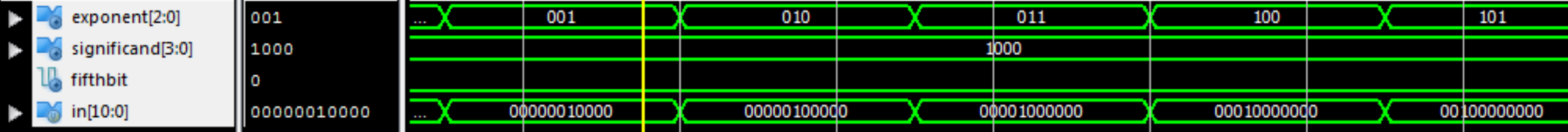


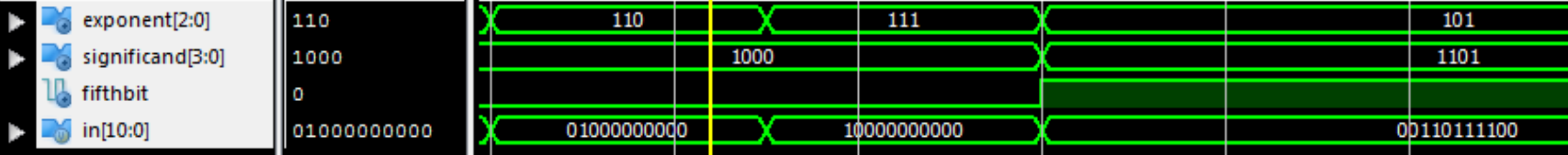
**RoughSignedToFP**

To test this module, we had to use more test cases then the others. However, they all test the same functionality of the module. Every test case except the last tests whether the module could count the number of leading zeros correctly to assign the right values to exponent[2:0] and significand[3:0]. The last test case was an arbitrary value and also checks whether the fifth bit will be set to 1.

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Result** | **Pass/No Pass** |
| in = 11’b000 0000 0000 | Exponent = 3’b000; Significand = 4’b0000;  Fifthbit is 0; | Passed |
| in = 11’b000 0000 0001 | Exponent = 3’b000; Significand = 4’b0001;  Fifthbit is 0; | Passed |
| in = 11’b000 0000 0010 | Exponent = 3’b000; Significand = 4’b0010;  Fifthbit is 0; | Passed |
| in = 11’b000 0000 0100 | Exponent = 3’b000; Significand = 4’b0100;  Fifthbit is 0; | Passed |
| in = 11’b000 0000 1000 | Exponent = 3’b000; Significand = 4’b1000;  Fifthbit is 0; | Passed |
| in = 11’b000 0001 0000 | Exponent = 3’b001; Significand = 4’b1000;  Fifthbit is 0; | Passed |
| in = 11’b000 0010 0000 | Exponent = 3’b010; Significand = 4’b1000;  Fifthbit is 0; | Passed |
| in = 11’b000 0100 0000 | Exponent = 3’b011; Significand = 4’b1000;  Fifthbit is 0; | Passed |
| in = 11’b000 1000 0000 | Exponent = 3’b100; Significand = 4’b1000;  Fifthbit is 0; | Passed |
| in = 11’b001 0000 0000 | Exponent = 3’b101; Significand = 4’b1000;  Fifthbit is 0; | Passed |
| in = 11’b010 0000 0000 | Exponent = 3’b110; Significand = 4’b1000;  Fifthbit is 0; | Passed |
| in = 11’b100 0000 0000 | Exponent = 3’b111; Significand = 4’b1000;  Fifthbit is 0; | Passed |
| in = 11’b001 1011 1100 | Exponent = 3’b101; Significand = 4’b1101;  Fifthbit is 1; | Passed |



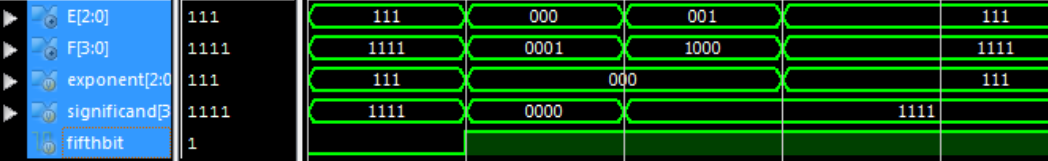




**Rounding**

To test this module, we used four test cases. The first one tests to ensure that no rounding occurs when the fifthbit is set to 0. The second test sets the fifthbit to 1 and checks to see whether the F[3:0] output is adjusted accordingly. The third test sets the fifthbit to 1 and checks to see whether a full F[3:0] output overflows into E[2:0]. The final test case sets the fifthbit to 1 and checks to see whether E[2:0] and F[3:0] are unchanged if they are both full.

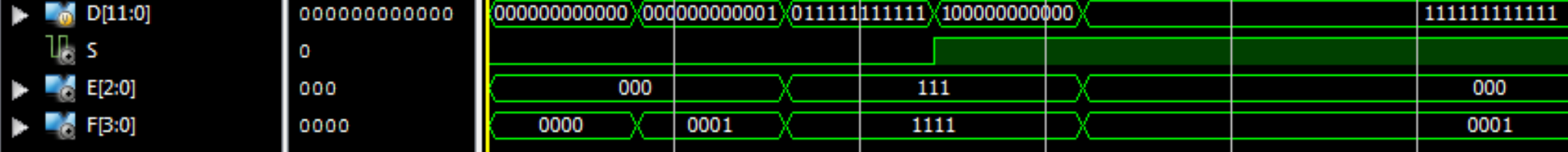
|  |  |  |
| --- | --- | --- |
| **Test Case** | **Result** | **Pass/No Pass** |
| exponent = 3’b111;  significand = 4’b1111;  fifthbit = 0; | E = 3’b111  F = 4’b1111 | Passed |
| exponent = 3’b000;  significand = 4’b0000;  fifthbit = 1; | E = 3’b000  F = 4’b0001 | Passed |
| exponent = 3’b000;  significand = 4’b1111;  fifthbit = 1 | E = 3’b001  F = 3’b1000 | Passed |
| exponent = 3’b111;  significand = 4’b1111;  fifthbit = 1; | E = 3’b111  F = 4’b1111 | Passed |



**FPCVT**

To test the entire module, we used the same methodology as testing the first submodule. We tested Tmin and Tmax and used the results to justify whether the intermediate values would pass or not. This was further justified when we passed all the test cases during the demo. We also tested -1, 0, and 1. We used 1 as a simple test to see whether the basic behavior was correct, and tested -1 and 0 since those were problematic numbers for the TwoToSigned module.

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Result** | **Pass/No Pass** |
| D = 12’b0000 0000 0000 (0) | S = 0  E = 3’b000  F = 4’b0000 | Passed |
| D = 12’b0000 0000 0001 (1) | S = 0  E = 3’b000  F = 4’b0001 | Passed |
| D = 12’b0111 1111 1111 (Tmax) | S = 0  E = 3’b111  F = 4’b1111 | Passed |
| D = 12’b1000 0000 0000 (Tmin) | S = 1  E = 3’b111  F = 4’b1111 | Passed |
| D = 12’b1111 1111 1111 (-1) | S = 1  E = 3’b000  F = 4’b0001 | Passed |



**Conclusion**

From this lab, we learned a good deal about designing circuits using Verilog and the other tools available to us. Many of the problems we ran into throughout the project were due to our lack of knowledge about the language and its intricacies. For example, figuring out what to declare as a wire as opposed to a register presented some problems, as did tracing code across different submodules and into simulations. Much of our debugging came through simple trial-and-error, and help was also provided by our TA. All things considered, this lab was a success, and we feel well-prepared for our future labs in this class.