CSM152A – Lab 2

Names: Jahan Kuruvilla Cherian; Kevin Xu

Group: Lab 6 – Group 9

UID: 104436427; 704303603

Date Performed: 4/19

TA: Babak Moatamed

1.0 INTRODUCTION

The aim of this lab was to demonstrate our ability to take a design specification and aptly convert it to a Verilog implementation. We were given the task to take a 12 bit two's complement encoded analog signal and convert it to a digital floating point representation of 8 bits. This was done by splitting the 12 bit input into a single *signed* bit (S), 4 bits for a *significand* (F) and 3 bit *exponent* (E), from which the representation was calculated using formula (1).

$$Floating\ Point\ Representation = (-1)^S * F * 2^E \tag{1}$$

This lab only had a testing and simulation of the design on Xilinx.

2.0 DESIGN IMPLEMENTATION

Figure 1 shows us the bit vector that represents the floating point conversion as required. We see the presence of the sign bit, significand and exponent.

7	6	5	4	3	2	1	0
S	E			F			

Figure 1: The resulting 8 bit vector for the floating point representation output by the final top module.

In order to follow good design principle, we broke the overall problem into three overall modules that were then connected using a top module, and tested using a testbench. The modules were broken up as per the block diagram given in the lab manual – shown in Figure 2.

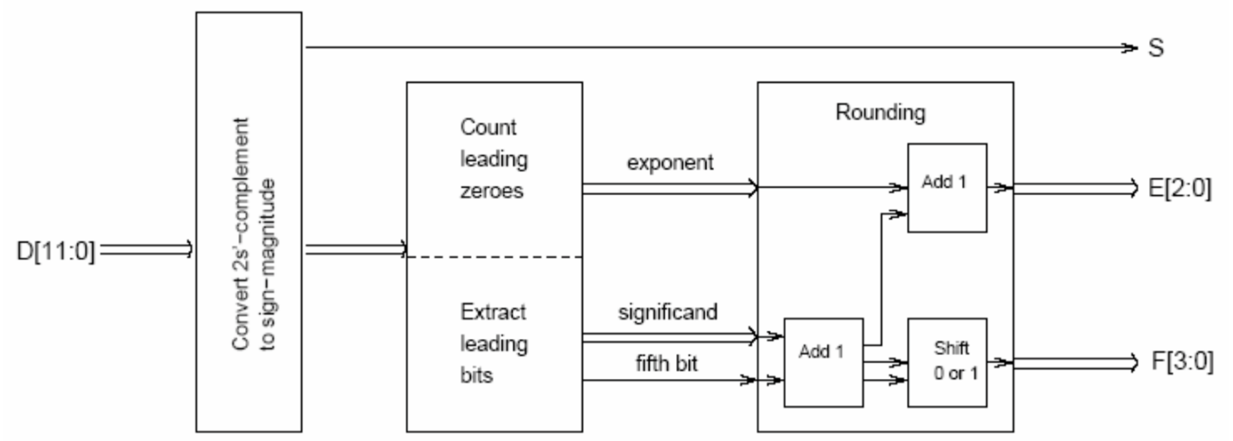

Figure 2: The overall design block diagram, off of which we based our design. Our final top module combines these three modules together.

The modules we separated into were the *twos_to_sign_magnitude, counting_and_extract* and *rounding* modules. The following sections explain more about them.

2.1 Twos to Sign Magnitude

This module converted the 12-bit two's complement vector into a sign magnitude representation. We did this by first extracting the most significant bit (which under Little Endian notation is the 11th position in the bit vector). Upon extracting this signed bit we have to determine whether the number is positive – in which case we continue to keep the magnitude the same as the input – or negative – in which case we use formula (2) to convert it to its magnitude representation.

$$-x = \sim x + 1 \tag{2}$$

Note that the exceptional case of the most negative number (-2048), doing so would lead to an overflow that our system could not handle, so we chose to output a series of all 1's as an indication of such an error.

Figure 3 represents our design of this module based off of how our Verilog implementation was written.

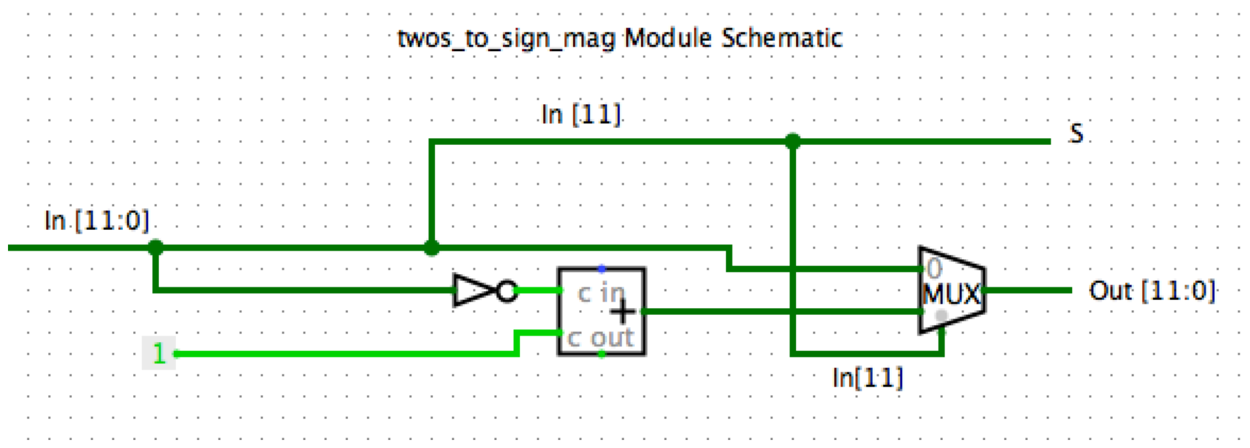

Figure 3: We output the input vector as our input into the next module just as is unless the significant bit is 1 (that is our number is negative) in which case we complement the number and add 1 to it as stated buy formula (2). The signed bit is immediately sent to the final output (S).

2.2 Count and Extract

This module takes in the output from the previous module and evaluates the exponent and significand, based on the number of leading zeroes given by Table 1.

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

Table 1: Represents the explicit exponent values for the various leading zeroes.

The significand is the very next 4 bits past the last leading zero. We also take into account the fifth bit after the significand which tells us whether the final floating point representation must be rounded up or down. The exceptional case of when we have the lest significant 4 bits as our significand, we set our fifth bit (Which doesn't technically exist) as 0.

Figure 4 represents the Logisim design of our Verilog implementation through the use of multiplexers to represent the if-else statements. From this we extract The output significand and exponent that is fed into our last module for rounding based on the fifth bit.

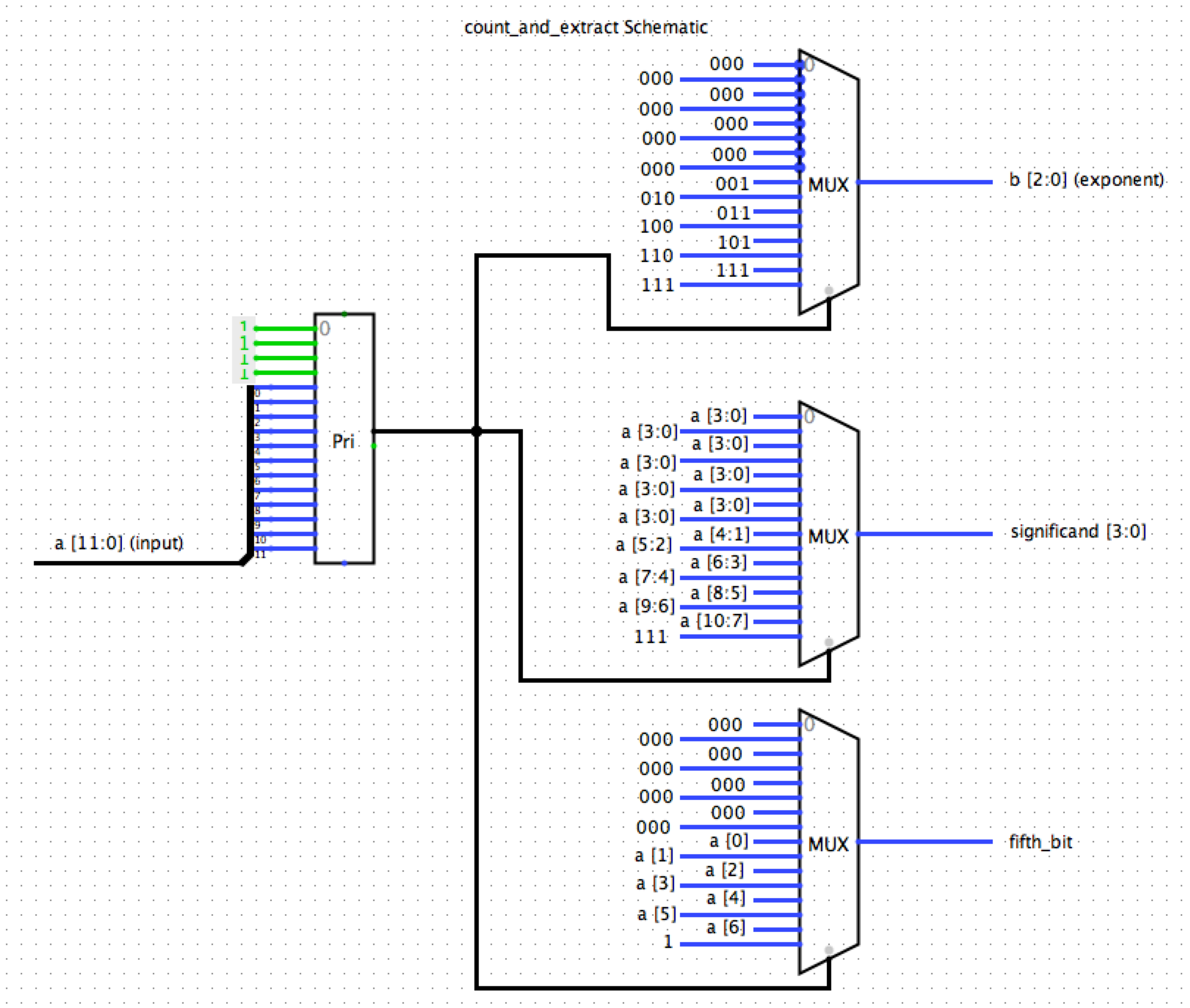

Figure 4: Logisim design schematic of our counting and extracting module. The priority encoder allows us to select the first 1 (i.e. tells us how many leading zeroes there are) and from there we extract the exponent by referencing Table 1, the significand and the fifth bit which are then fed as inputs into our final rounding module.

2.3 Rounding

If the fifth bit that comes into this module is 1 then we round up by adding one to the significand, and if the significand overflows we add 1 to the exponent field and we right shift the significand to form 1000.

If we are not rounding, we simply pass the result from the previous module as our final significand and exponent.

Figure 5 shows the Logisim design of our implementation of this module. As mentioned earlier for the case of overflow of both the exponent and significand, we convert the number to all 1's.

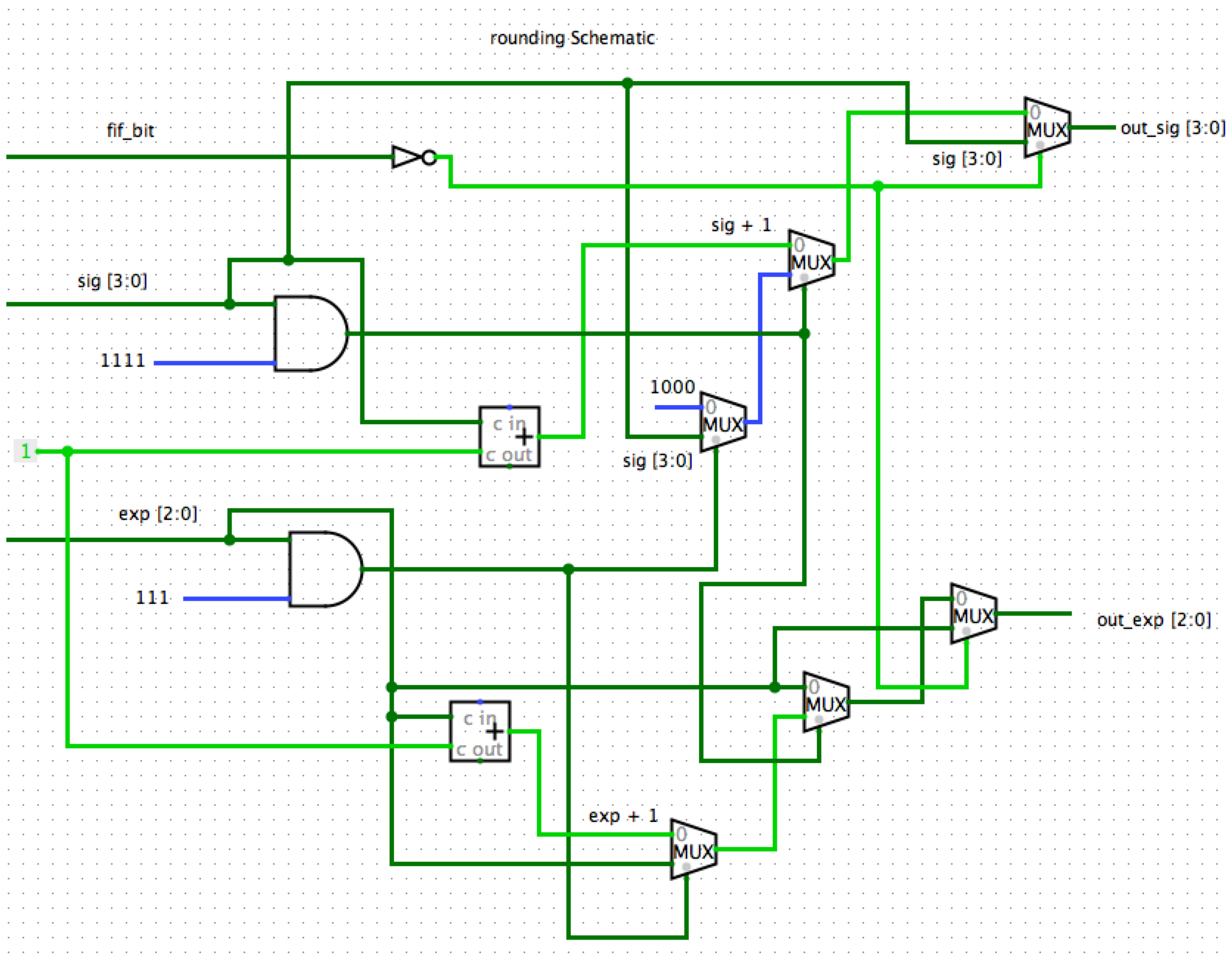

Figure 5: This Logisim design follows the flow of our basic Verilog implementation, using multiplexers as if-else statements. If the fifth bit is zero, then we require no rounding and so we output the same output from the previous module. If it is 1 however, we add one to the significand and select that. Upon the significand overflowing we add 1 to the exponent and right shift the significand to 1000 and output that instead. If both overflow the we send all 1's as the output.

2.4 Top Module

The final top module simply combines these three modules into one by passing each modules output as the input to the next module until we finally get the output from the final rounding module.

2.5 Test Bench

The test bench was a very simple one used to run a simulation of our implementation. The output of the testing values used are shown in the waveform represented in Figure 6.

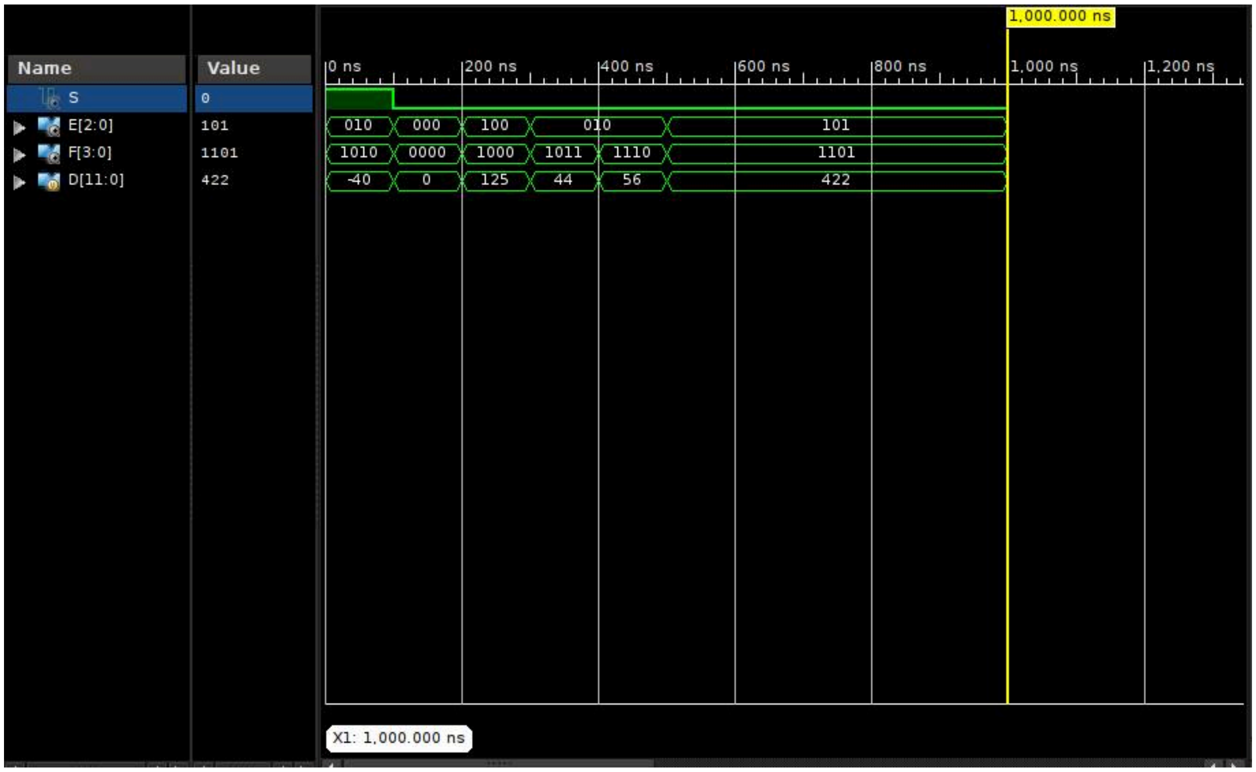

Figure 6: The waveform produced by the final top module running through the series of tests fed in. The values were verified through hand calculations and through the lab manual. This was used for demoing purposes.

3.0 CONCLUSION

This lab taught us how to build a basic combinational module in Xilinx ISE and build a full fledged testbench, from the initial block diagram design specification. We chose to modularize the problem and build a final overall testing module to combine the results.