Lab 2

Russell Jew (ID: 604477316)
Jeffrey Jiang (ID: 904255069)

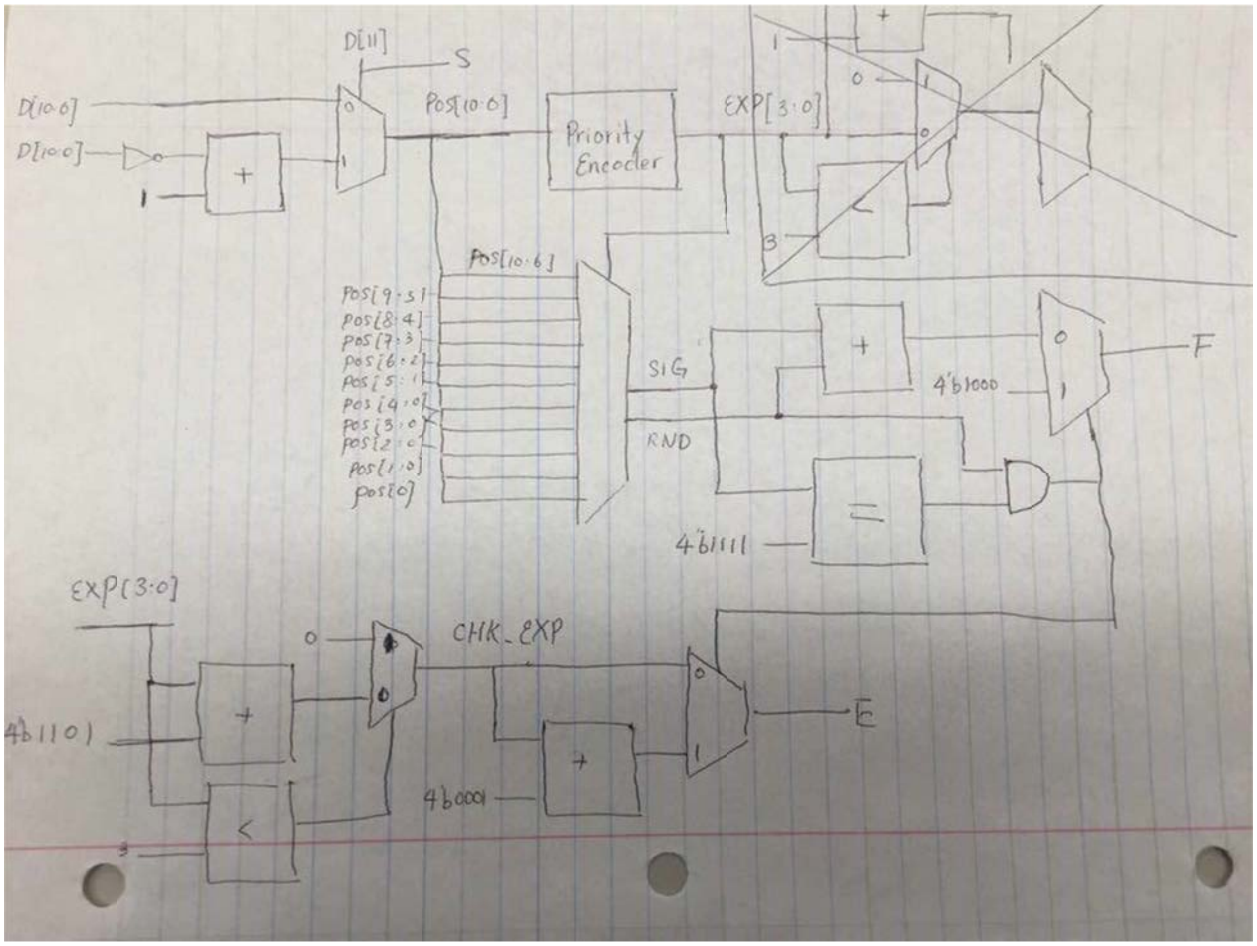
Introduction

In Project 2, we are given the task to implement a floating-point converter. The idea is to design a schematic that will allow a 12 bit 2's-complement number to be converted into an 8-bit floating point number, with a 4 bit significand and 3 bit exponent.

As background information, a 2's-complement representation of an integer is based on a simple linear encoding for the most part. The most significant bit represents the sign bit of the number and the other bits represents the sum of powers of 2. The negative form of a positive number essentially is the inverse + 1, which creates a representation where the negative number and the positive number will sum up to 0, making it so that addition and subtraction are easily implemented in 2's-complement. Floating point numbers, on the other hand, follow a more "scientific format" rule, where the bits are split up into a sign bit s, significand bits F, and exponent bits E. This creates a floating point number where the value is $(-1)^s \times F \times 2^E$. An additional complication of this floating-point converter is that because the conversion goes from a higher number of bits to a lower number of bits, there must be a possible loss of information, leading into various edge cases and rounding cases that complicate the converter.

This project essentially serves a first project for understanding the design of a logic circuit and the translation of the schematic to Verilog code, allowing for familiarization of Verilog syntax and use of simulations.

Design



The overall design of the floating-point converter is shown above. The diagram that we drew above does ignore certain edge cases, which will be discussed below.

Inputs:
D[11:0] - Input data in two's complement to be converted to floating point.

Outputs:
S - Sign bit of the floating point representation
E[2:0] - Exponent of the floating point representation
F[3:0] - Significand of the floating point representation

The first step to convert from two's complement to floating point representation is to convert the two's complement to positive if it is negative. So in the top left part of the our diagram, we use a mux with D[11], the sign bit, as the selector bit. The inputs to the mux are D[10:0] and D[10:0] inverted plus 1. D[11] selects the inverted input if it is 1 (input is negative) or selects the non-inverted input if it is 0 (input is positive). D[11] goes on to become S (sign bit of floating point).

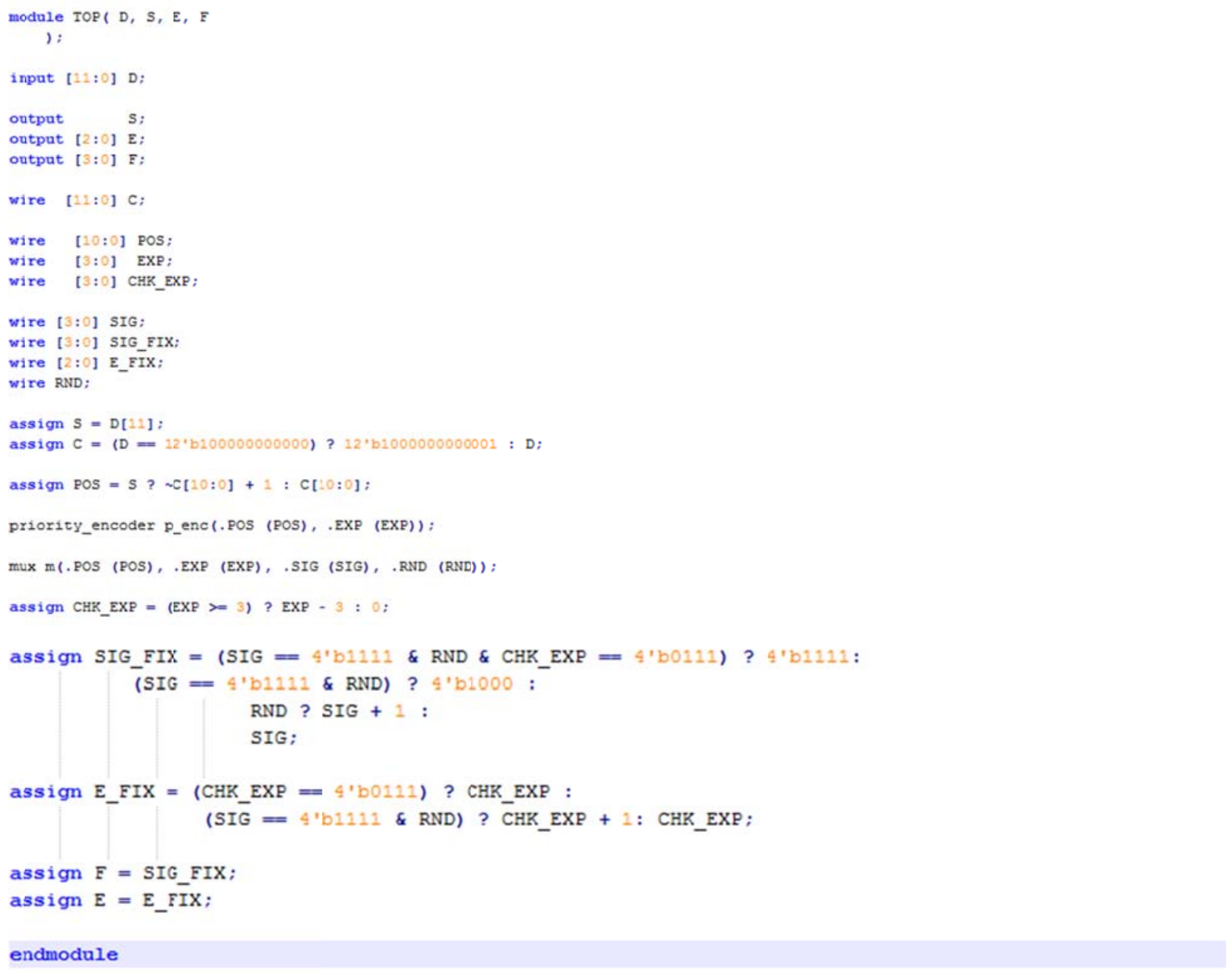
Next step is to run the output of the mux to a priority encoder to get the position of the most significant one. With the position of the most significant one, we can determine the significand and the exponent. The significand will the most significant one and the three bits following it. The exponent will be the position of the bit after the significand.

To get the significand, we can use the output of the priority encoder (position of the most significant one) as a selector bit into a mux. The inputs to the mux are the possible significands along with the next bit as the round bit. The output of the priority encoder will select the correct significand based on the value it is.

To get the exponent, we also use the output of the priority encoder (EXP [3:0]). The value of the exponent will be EXP - 3 if it is greater than 3, otherwise the exponent will just be zero. To accomplish this, we use a comparator between EXP and 3 as the selector bit for a mux. The inputs to the mux are 0 and EXP - 3. The mux outputs 0 if EXP < 3, else it outputs EXP - 3. The output of the mux will be the value of the exponent.

However, there is another problem that needs to be accounted for, which is rounding and overflow. So for rounding we would just add 1 to the significand if the bit after the significand was 1. But the problem is that adding 1 to significand could cause it to overflow. So to account for this, we check the value of the significand. If adding one to the significand will cause it to overflow (significand = 4'b1111) , then we output 4'b1000 for the significand and add 1 to the exponent as a shift. To do this, we use an AND gate to check for the condition (significand = 4'b1111 AND RND = 1) and the output of the AND gate goes into a mux. The mux will output the correct significand based on the AND gate's check, which will become the value for F[3:0]. The output of the AND gate is also run through another mux for the exponent, and will select the corrected exponent if the condition holds (significand = 4'b1111 AND RND = 1) , which will become the value for E[2:0]. One final case that occurs when EXP is already at the maximum of 7 (= 3'b111) In theory this causes the exponent to overflow as well, which following this pattern would reset the exponent bits back to 0. Instead, we want to round to the closest possible represented value, which would be the maximum positive number that can be represented using an 8 bit floating-point value, which is 8'b01111111, so we make sure that we set the exponent value to 7, which is not a change in the exponent value. Similarly, the negative case would set the exponent value to 7. As a result, we only set the value of E to EXP + 1 when EXP did NOT equal 4'b0111 AND significand = 4'b1111 AND RND = 1.

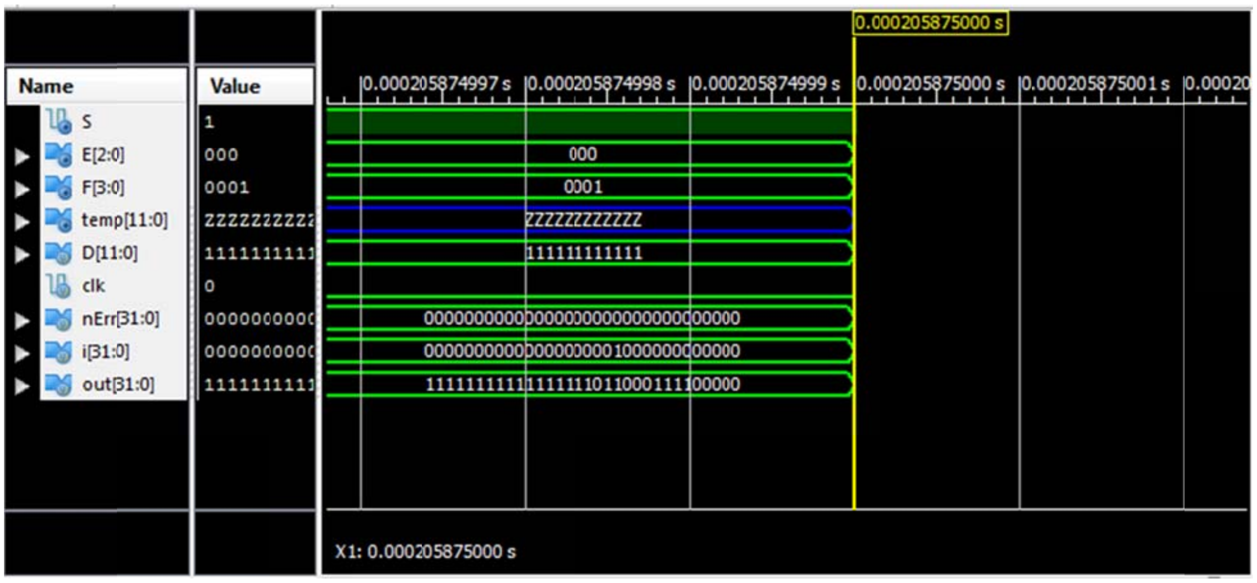
Finally, one last case that was important to consider was D = 0 and D = MIN = -2048 = 12'b100000000000. The most important aspect of these two numbers is that the two's complement reversal leads to the same number. For the zero case, this makes sense since 0 = -0. However, 2048 != -2048 normally, so this presents problems. Based on the schematic we drew, other than the sign bit, these two numbers would have been handled in the exact same way, since we extracted the positive number out of the 2's-complement representation. However, this means that the floating point representation would end up being 0 and -0, which are both zero, and this is not expected for D = MIN. Thus, the only thing we could do was to set up a special case where D = MIN to account for this special case where 2's-complement inversion is not exactly accurate.

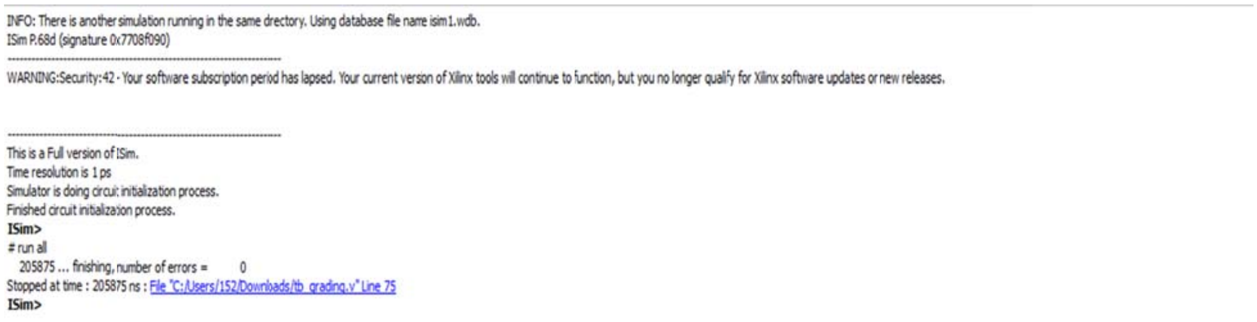


```
module TOP( D, S, E, F
    );

input [11:0] D;

output       S;
output [2:0] E;
output [3:0] F;

wire  [11:0] C;

wire   [10:0] POS;
wire   [3:0]  EXP;
wire   [3:0] CHK_EXP;

wire [3:0] SIG;
wire [3:0] SIG_FIX;
wire [2:0] E_FIX;
wire RND;

assign S = D[11];
assign C = (D == 12'b100000000000) ? 12'b1000000000001 : D;

assign POS = S ? ~C[10:0] + 1 : C[10:0];

priority_encoder p_enc(.POS (POS), .EXP (EXP));

mux m(.POS (POS), .EXP (EXP), .SIG (SIG), .RND (RND));

assign CHK_EXP = (EXP >= 3) ? EXP - 3 : 0;

assign SIG_FIX = (SIG == 4'b1111 & RND & CHK_EXP == 4'b0111) ? 4'b1111:
        (SIG == 4'b1111 & RND) ? 4'b1000 :
                RND ? SIG + 1 :
                SIG;

assign E_FIX = (CHK_EXP == 4'b0111) ? CHK_EXP :
            (SIG == 4'b1111 & RND) ? CHK_EXP + 1: CHK_EXP;

assign F = SIG_FIX;
assign E = E_FIX;

endmodule
```

Verilog code for the floating point converter. TOP is the main module which uses the modules of priority encoder and mux to implement the design. The verilog code accounts for the edge cases discussed above.

Simulation





```
INFO: There is another simulation running in the same drectory. Using database file name isim1.wdb.
ISim P.68d (signature 0x7708f090)
----------------------------------------------------------------
WARNING:Security:42 - Your software subscription period has lapsed. Your current verson of Xilinx tools will continue to function, but you no longer qualify for Xilinx software updates or new releases.


----------------------------------------------------------------
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
# run all
   205875 ... finishing, number of errors =        0
Stopped at time : 205875 ns : File "C:/Users/152/Downloads/tb_grading.v" Line 75
ISim>
```

The simulation of this code was simply an exhaustive list of all 4096 different 12 bit 2's-complement values possible from -2048 to 2047. Each case had an expected output that could be tested very simply because of the combinational nature of this circuit. This code for the testbench was provided which indicated three special cases which were exponent and significand overflow, no rounding, and significand overflow. The testbench would output error messages for unexpected outputs and the number of errors that occurred.

One of the problems we ran into when running the simulation was the fact that the simulation included examples explained to be ignored on the lab manual. This case was explained to an extent in the design portion: the overflow of the exponent to 8. This bug, when found, forced us to add an additional consideration to the MUX that we had in place and we added an additional stage in order

Conclusion

The purpose of this lab was to design a two's complement to floating point converter. The overall design was to make the two's complement number positive, use a priority encoder to get the position of the most significant one which could be used to get the exponent and significand, and account for rounding and overflow errors. Some difficulties that we encountered were dealing with the edge cases where overflows got complicated and the MIN case.

As an improvement to this lab, we would like to mainly suggest just making the lab manual clearer, which the addition of just using the maximum value as the output of the high value cases and perhaps giving a little bit of a head-start with skeleton code snippets. Other than this, we believe that this lab was a good introductory lab into creating a Verilog module that was truly useful while also being rather simplistic in terms of coding.