CS M152A Lab 2 Report

Group 8 - Jackson Bae and Ryan Yang

Introduction:

The goal of this lab was to design a combinational circuit in Verilog to compress a 12-bit two's complement input into an 8-bit floating-point output. This floating-point format consists of a 1-bit sign, a 3-bit exponent, and a 4-bit significand, allowing for greater range than an 8-bit linear representation and making it suitable for applications such as audio signal processing. The output represents values in the form $V = (-1)^S \times F \times 2^E$, where S is the sign bit, E is the exponent, and F is the significand.

Design Description:

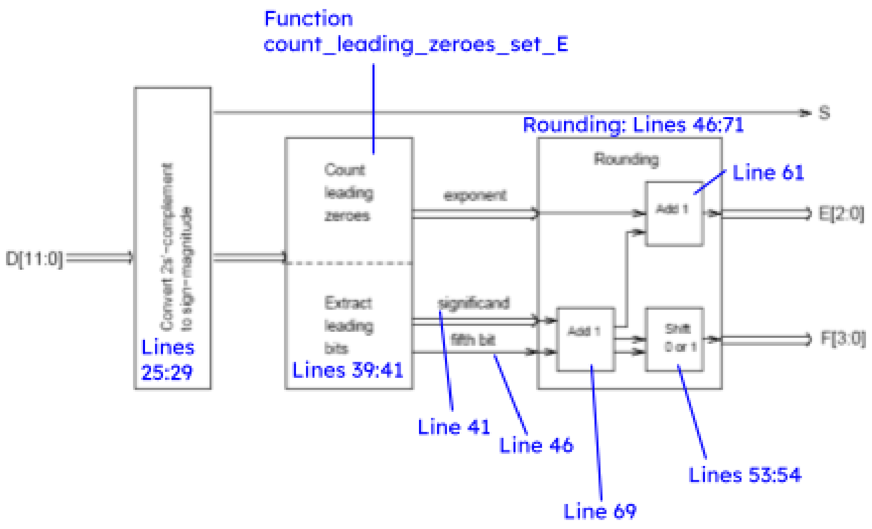

Figure 1: *Schematic of FPCVT.v module with code annotations*

The logic for the floating-point converter is fully contained within the FPCVT.v module. The design is entirely combinational and structured within a single `always @(*)` block. Most of the control flow is organized using if-else statements to cover various conditions, including special cases like the minimum negative input and rounding overflow.

First, the sign bit S is recorded directly from the most significant bit (MSB) of the 12-bit input D. Next, the design checks for the special case where D equals -2048, which cannot be properly normalized through standard logic and therefore has its output hardcoded to the maximum representable magnitude (E = 111, F = 1111). For all other inputs, the magnitude is computed: if D is positive, it is used directly; if D is negative, its two's complement is taken.

After computing the magnitude, the design checks whether the input is zero. If it is, the exponent and significand are hardcoded to zero. Otherwise, a helper function, `count_leading_zeroes_set_E`, is used to determine the exponent based on the number of leading zeros in the magnitude.

Once the exponent is determined, the magnitude is shifted left to remove the leading zeros, aligning the most significant non-zero bits to the top. The significand F is then assigned from the highest four bits of the shifted value. To implement rounding, the fifth bit following the significand is captured. If this bit is 1, the design rounds up. Special care is taken in the rounding logic: if rounding would overflow the significand (i.e., if F is 1111), the significand is set to 1000 and the exponent is incremented. If the exponent is already at its maximum value (111), it remains capped and the significand is set to 1111 to represent the maximum possible floating-point value.

Simulation and Testing:

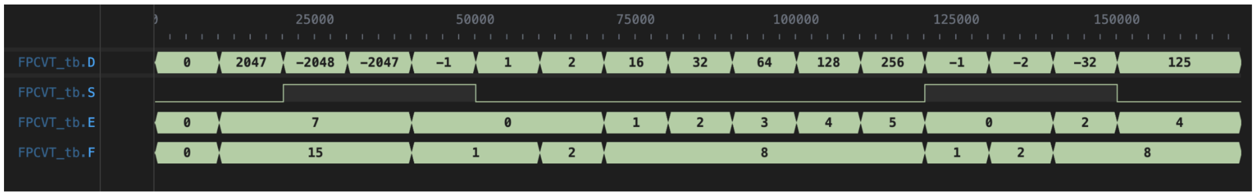

Figure 2: *Simulation waveforms for select test cases*

To verify the correctness of the FPCVT module, we developed a Verilog testbench that systematically applied a range of input values to exercise both typical and edge-case behavior of the circuit. The testbench applied each input value to the 12-bit input D and observed the corresponding sign (S), exponent (E), and significand (F) outputs. Simulation waveforms were captured using $dumpvars and examined to confirm that the outputs match the expected behavior for each test case.

We began by testing key boundary values: 0, the maximum positive value (+2047), the minimum negative value (–2048), and –1 (all 1's in 2's complement form). These cover critical corner cases including full positive/negative scale and correct two's complement interpretation. Next, we tested a sequence of small positive and negative values (such as +1, +2, +16, +32, etc.), which are designed to fit cleanly into the floating-point representation without requiring rounding. These test whether the converter produces correct normalized outputs and encodes simple powers of two cleanly into the exponent/significand format.

To test rounding behavior, we included values near rounding thresholds, including those that require rounding up to the next representable floating-point value. These cases also tested for rounding overflow—such as a significand exceeding its bit-width (e.g., 10000)—and confirmed that the circuit correctly increments the exponent and resets the significand when needed. Finally, we included inputs like +56 and +16, which can have multiple valid floating-point representations, to ensure the module consistently normalizes values according to specification.

Conclusion:

In this lab, we designed a combinational floating-point converter that transforms a 12-bit two's complement input into an 8-bit floating-point representation. Our implementation followed a step-by-step process to meet the lab specifications accurately and handle all required edge cases. The key steps in our design were:

1. Recording the sign bit
2. Detect -2048 edge case

3. Capture magnitude of E
4. Count leading 0’s to determine E and shift amount
5. Set the significand based on the shifted value
6. Round F and E based on the 5th bit, being careful of overflow edge cases

We encountered several challenges, primarily in understanding the floating-point and normalized representations. These were mostly resolved by re-reading the specification multiple times and working through example values to clarify the logic. Implementing the logic in code was occasionally confusing, especially since we had to rely on if-else statements rather than returning early after handling edge cases. Creating a helper function helped improve organization, but we could have made the code even more manageable by isolating and handling edge cases at the beginning, rather than integrating them into the main logic flow.

One area of potential improvement for the lab is the organization of the specification. Some important implementation details are scattered across different sections, which can make the task harder to follow at first. For example, the section labeled “Input Format” begins with a table mapping the number of leading zeros to the exponent value—information that is more relevant to encoding logic than to input formatting. Reorganizing the specification so that related concepts are grouped more clearly would help students better understand the task and reduce confusion during implementation.