Xuan Yuan Huang

Alvin Lim

Group 11

CS M152A

Lab 2

**Introduction**

For this lab, we were tasked with implementing a combinational converter (a module we named FPCVT) which takes 12-bit two’s complement linear encoding of integers and translates them into 8-bit floating point representations. Our simplified floating point representation has one sign bit, and then 3 bits for the exponent and 4 bits for the significand.

**FPCVT**

**D[11:0]**

**S[11:0]**

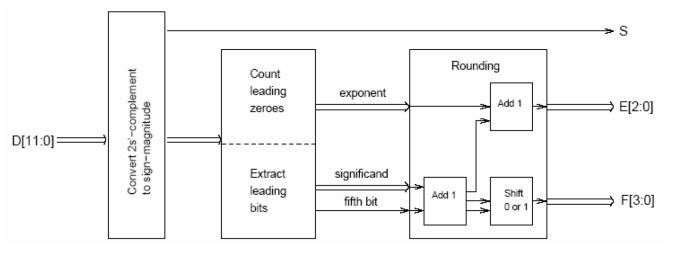
**E[2:0]**

**F[3:0]**

12-bit Input

8-bit Floating Point Representation

**Design**



**Inverter**

**Converter**

**Rounder**

**FPCVT**

We borrowed the schematic in the lab manual and implemented it as 3 submodules to the FPCVT module. Here is a breakdown of each of our 3 submodules:

* **Inverter**: “inverts” integer if necessary
  + ***Input:*** Original 12 bit integer
  + ***Output:*** **(a)** Sign bit **(b)** Absolute value of original integer
  + ***Implementation:***
    1. Save sign bit for output.
    2. If original number is negative, invert the input (~) and add 1.
       - If the input is INT\_MIN, invert it but don’t add 1 (because of overflow).
    3. Output the inversion (for negative inputs) or the original number.
* **Converter** does most of the work in converting from integer to floating point
  + ***Input:*** Absolute value of original integer (from Inverter)
  + ***Output:* (a)** Pre-rounded exponent **(b)** Pre-rounding significand **(c)** “Fifth bit” (used for rounding)
  + ***Implementation:***
    1. Priority encoder counts the number of leading zeros and calculates the corresponding exponent value.
    2. Shift the input by [exponent – 1] to acquire the first 5 bits after the leading zeroes. First 4 bits after leading zeroes becomes the significand and we keep the fifth bit for rounding.
       - If exponent is 0, don’t shift, take the last 4 bits as the significand and assume the fifth bit is zero.
* **Rounder** deals with rounding issues in the conversion
  + ***Input:* (a)** Pre-rounding exponent **(b)** Pre-rounding significand **(c)** “Fifth bit” (from Converter)
  + ***Output:*** Final, rounded **(a)** exponent and **(b)** significand
  + ***Implementation:***
    1. If the fifth bit is zero, we don’t need to round, so output the exponent and significand.
    2. If the fifth bit is one, increase the significand by 1 and output the significand and exponent.
       - If significand overflows when adding 1 to it, right shift the sum by one and increment the exponent by 1. Output the significand and exponent.
       - If the exponent overflows when incrementing by one, set all the bits of the significand and exponent as 1 (maximum representable floating point number in our representation).

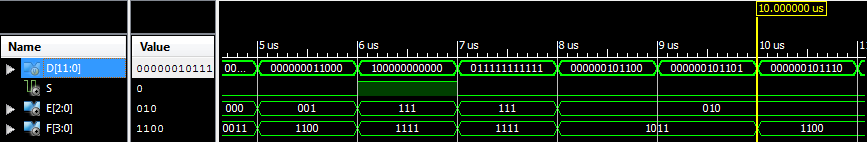
The FPCVT module then outputs the sign bit from the inverter along with the exponent and significand from the rounder as its own output.

**Simulation**

We used the following test cases on our module:

|  |  |  |
| --- | --- | --- |
| **Binary Input** | **Decimal Representation** | **Testing Purpose** |
| 12’b0 | 0 | Basic functionality |
| 12’b1 | 1 | Basic functionality |
| 12’b10 | 2 | Basic functionality |
| 12’b11 | 3 | Basic functionality |
| 12’b1\_1000 | 24 | Basic functionality |
| 12’b1111\_1111\_1111 | -1 | Negative number functionality |
| 12’b0111\_1111\_1111 | INT\_MAX | Edge case (Exponent overflow) |
| 12’b1000\_0000\_0000 | INT\_MIN | Corner case (Absolute value overflow) |
| 12’b0000\_0010\_1100 | 44 | Rounding |
| 12’b0000\_0010\_1101 | 45 | Rounding |
| 12’b0000\_0010\_1110 | 46 | Rounding |
| 12’b0000\_0010\_1111 | 47 | Rounding |

Here is a screenshot showing some of the waveforms showing the results of our test cases:



We encountered several bugs during testing:

* The FPCVT’s F and E outputs didn’t seem to update correctly most of the time. We realized that this issue occurred with numbers that had a 0 following their significand – we forgot an else clause in the rounder that would output the original value if rounding was not necessary.
* For small numbers, we were losing bits that should have stayed in the significand. We realized we were shifting too far when the exponent was only zero, so we implemented a conditional that would not shift in that case and simply assumed the fifth bit was zero.
* INT\_MIN is problematic in two’s complement, because the normal way of calculating absolute value simply returns INT\_MIN back. But because an integer as large as INT\_MIN can’t be represented in our 8 bit floating point regardless, we inverted *without* adding 1 before passing it on. While technically wrong, the inaccuracy is lost in conversion because it is exponentially smaller than the significand.

**Conclusion**

Overall, this lab was pretty straightforward, and we learned the basics of designing, implementing, and testing modules in Verilog. Our main struggles had to do with syntax, as well as edge cases and corner cases. We did feel that our experience with this lab made us wonder if it should have been the first assignment instead, to give us a gentler transition into hardware design and the Xilinx ISE.