CS 152A / EE M116L

Introductory Digital Lab Design

LAB #3

Floating Point Conversion

May 6, 2014

Grade:

Tianheng Tu

Name1: Nathan Tung

SID1: 004-059-195

Name2: Mark Iskandar

SID2: 704-050-889

INTRODUCTION

The purpose of Lab 3 is to design and implement a simulation of a combinatorial circuit using Xilinx ISE. The circuit we are building, called a Floating Point Converter (FPCVT), takes as input a 12-bit linear encoding of an analog signal and transforms it into a compounded 8-bit Floating Point encoding. In other words, it converts a 12-bit Two's Complement Representation into an 8-bit Floating Point Representation (or more specifically, a 1-bit sign, a 3-bit exponent, and a 4-bit mantissa) with some loss of data.

In order to build our system, we construct modules used in our design separately – modules responsible for converting 2's complement to sign-magnitude, determining the exponent, extracting the mantissa (or significand), rounding, and so on – and link them together to create a functional circuit. In this report, we will specify the role of each component module in our FPCVT and look at how everything functions together as a whole. The high-level schematic of our FPCVT can be found at the end of this report.

DESIGN COMPONENTS

Overall

As mentioned before, our FPCVT accepts a 12-bit 2's complement input. To convert it to floating-point, the procedure involves extracting the most significant bit (MSB) and outputting it as the sign-bit. Next, we convert the entire 12-bit input into a 12-bit sign-magnitude string (*Sign-Magnitude Converter*). Then we count the leading zeroes to determine the exponent (*Priority Encoder*) and pull out the next five bits (four bits if we reach the end) to use as our mantissa and rounding digit (*Mantissa Extractor*). Finally, if the fifth rounding bit is a 1, we need to round the mantissa up and account for overflow by incrementing the exponent, if applicable (*Rounder*).

Sign-Magnitude Converter

To convert 2's complement to sign-magnitude, we take the binary string, invert all the bits, and add 1. This module outputs the original MSB as the final sign-bit output. Then it inverts every bit in the 12-bit string and adds 1 by using three full adders in series to produce one possible output. Notice that a sign-bit of 0 means no work has to be done. The sign-bit goes into a MUX to select between the original 12-bit input (if it is positive and sign-bit is 0) or the new, sign-magnitude 12-bits (if original input is negative and sign-bit is 1).

Priority Encoder

After converting to sign-magnitude, we need to count the number of leading zeroes, which directly tells us the 3-bit exponent according to the table in the lab manual. A single leading zero (the lowest possible number of leading zeroes) gives a maximum exponent of 7, while 7 leading zeroes gives an exponent 1, and 8 or more leading zeroes gives an exponent of 0. To implement this, we wired our own priority encoder via NOT gates, AND gates, and OR gates. If the original input of FPCVT is D(11:0), then our priority encoder uses D10 through D4 in the logic.

Mantissa Extractor

This module accepts the 3-bit exponent and the 12-bit sign-magnitude string; its purpose is to extract the next five bits directly following the leading zeroes. (If possible, the fifth bit is

extracted to determine whether rounding needs to be done. If we reach the end of the 12-bit string before the fifth bit, then assume fifth bit is 0.) To select these five bits, we use five 8-bit MUXes and the exponent input as a selector. For example, an exponent of 5 means there are 3 leading zeroes, so with an input D(11:0), we output D8 down to D4, with D4 being the fifth bit.

To be more specific, the most significant MUX selects from D11 to D4, the next MUX selects from D10 to D3, and so on, until the fifth MUX selects from D7 to D0. Therefore, based on number of leading zeroes, we obtain the sequence of bits immediately following.

Rounder

FPCVT is not lossless, since we are compressing 12 bits of information into 8 bits. Some 12-bit numbers will inevitably map to the same 8-bit numbers, and rounding is necessarily to facilitate that mapping process. Our fifth rounding bit from before decides whether this step is necessary. If the rounding bit is 0, the mantissa does not need to be rounded up. However, if the rounding bit is 1 and the mantissa is 1111, not only does the mantissa change to 1000, but the exponent must be incremented as well due to overflow (the mantissa is shifted from 10000 to 1000, which necessitates an increase in exponent).

The rounding module uses a full adder to find the sum of the 4-bit mantissa and the fifth rounding bit. If the carry-out is 1, that means the pre-rounded mantissa is 1111, so the exponent must be incremented and the mantissa set to 1000. Thus the carry-out bit is added to the exponent via a full adder. It is also used as a selector in a MUX to set the MSB of the mantissa to 1 in case of overflow. No changes are made if the rounding bit is 0. This module outputs the final 3-bit exponent and the 4-bit mantissa that make up the Floating-Point Representation.

CONCLUSION

This report addresses the how our Floating Point Converter works on both a high-level scale (in terms of the design schematic) and in each individual module. Some of the biggest challenges we dealt with were figuring out how to use the Xilinx schematic tools (especially the bus taps) and determining where warnings and errors were being generated. In order to get a better feel for the software environment, our team started by making simple practice circuits mentioned in the tutorial. Debugging is much harder to solve; for us, it involved modular testing (that is, testing each module with sample data) until some error occurred or the output was incorrect. After much testing, we successfully built a FPCVT simulation that produced rewarding results.

Lab Contribution:

Nathan Tung: 50%

Mark Iskandar: 50%

DESIGN SCHEMATIC

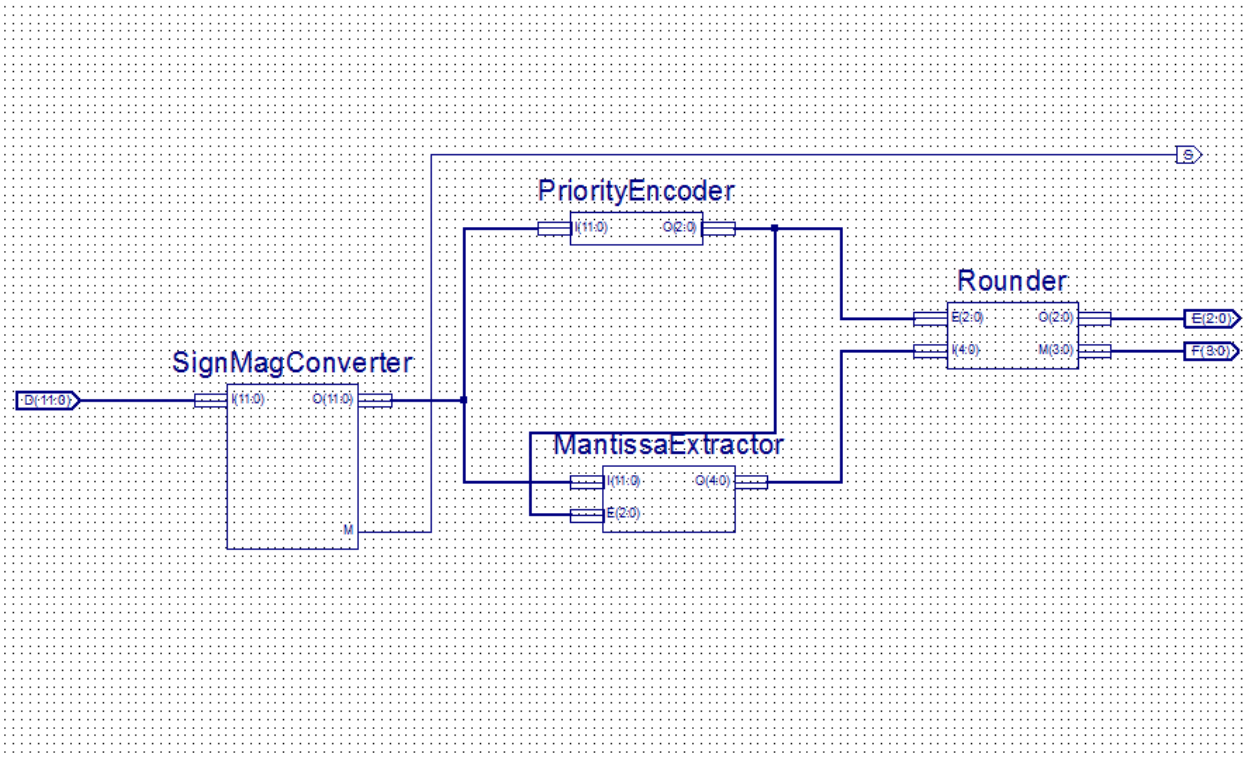

Figure 1: High-level schematic of the FPCVT combination circuit with custom modules.