**CSM152A-LAB 2**

Lab 5

Yichen Lyu

UID: 004940413

May 2, 2020

**1. INTRODUCTION**

This lab will explore how to design, implement and test using Xilinx a combinational circuit that converts an analog 13-bit 2s complement representation to a compounded 9-bit floating point representation. The floating point value flt[8:0] is composed by 3 parts: a 1-bit sign S (flt[8]), a 3-bit exponent E (flt[7:5]), and a 5-bit significand F (flt[4:0]). The overall value of the floating-point representation is calculated as below:

Through 3 combinational circuit modules, the overall design should round the analog signal to its closet floating-point value.

**2. DESIGN DESCRIPTION**

The overall combinational circuit will accept a 13-bit 2s complement input, D[12:0], and output a 1-bit sign value S, a 3-bit exponent value E[2:0], and a 5-bit significand F[4:0]. The floating-point representation is the combination of S, E, and F, shown in the figure below.

A picture containing object, clock

Description automatically generated

Figure 1: Floating Point Representation

The overall circuit is broken up into 3 modules: convert\_2s\_to\_sign, count\_extract, and round. They follow the same design given in the lab specification.

A screenshot of a cell phone

Description automatically generated

Figure 2: Overall Combinational Circuit Design

**2.1 convert\_2s\_to\_sign**

This module coverts a 2s-complement to a sign magnitude. 2s-complement stores all positive values in binary forms. When dealing with a negative value, it flips its absolute value and then add 1. Sign-magnitude has its MSB representing the sign and the rest as the absolute value. 2s-complement and sign magnitude have the same representation when the value stored is positive. When it is negative, the MSB of sign magnitude is set to one, and the rest of it stores the absolute value of the 2s-complement. This requires the following implementation: we need to flip all the bits in 2s-complement and then add 1. When the input is -4096, however, it will overflow if we try to represent it using sign-magnitude. Thus, I choose to round it as -4095.

The implementation is described in the circuit design below. D[12] sets S directly, and D[11:0]’s complement is calculated by flipping it and adding 1. Then, a mux chooses to put out either D[11:0] without change or its complement. The select signal is D[12]. The code implements the mux using if-else statement.

A close up of text on a white background

Description automatically generated

Figure 3: convert\_2s\_to\_sign Circuit Design

**2.2 count\_extract**

This module counts the leading zeros to decide the value of exponent and then extract the first 5 bits starting from the first non-0 bit as a value of significand. Both exponent and significand are values before rounding. It takes in a sign magnitude temp\_sign[12:0] and outputs the preliminary floating point E and F field: exponent[2:0] and significand[4:0]. It also outputs a 1-bit sixth\_bit that is used for rounding in the next module.

The exponent is calculated according to the chart given in the lab specification. The less leading zero the sign magnitude has, the larger its value is, and the larger the exponent needs to be. The leading zeros directly determines exponent [2:0].

A screenshot of a cell phone

Description automatically generated

Figure 4: Relationship between Leading Zeros and Exponent

We then extract the next 5 bits right after the leading zeros. They are outputted as significand[4:0]. The sixth bit is outputted as sixth\_bit for rounding. If there are exactly or less than 5 bits left, significand[4:0] outputs the last 5 bits, and sixth\_bit is set to 1’b0.

The diagram below shows the circuit design. In coding, it is implemented by if-else statements.

A picture containing text, map

Description automatically generated

Figure 5: count\_extract Circuit Design

**2.3 round**

This module rounds the preliminary floating point exponent and significand field produced by the last module to a value closest to the analog signal. It takes 3 inputs: exponent[2:0], significand[4:0], and sixth\_bit. It outputs 2 values: E[2:0] and F[4:0] to the overall circuit.

When sixth\_bit is 1’b0, there is no need for rounding. exponent will become E and significand will become F. If sixth\_bit is 1’b1, we need to round the significand. significand will be incremented by 1, as long as it does not cause overflow. If it overflows, exponent is incremented by 1 and outputted to be E. Otherwise, it remains the same. There is one case where the exponent overflows as well: D = 4’d4095. When this happens, it is rounded to the largest floating value: 2’b0 111 11111. That is, the exponent and significand are unchanged and outputted directly to E and F.

The circuit design is shown below and is implemented using if-else statements.

A close up of a map

Description automatically generated

Figure 6: round Circuit Design

**2.4 FPCVT**

This module combines all 3 modules together according to the overall design circuit (figure 2). It takes in D[12:0], whose 13th bit becomes output S. After going through the 3 modules described above, it outputs the rounded E and F.

**2.5 testbench**

The testbench checks several critical D inputs: 4095, -4095, 4096, -255, 511, -1023, 100, 102.

Case 1: 4095 is used to check maximum floating point conversion. It also checks the condition of both exponent and significand overflow.

Case 2: -4095 is used to check if the 2s complement conversion is done correctly. It should only differ with case 1 by sign.

Case 3: -4096 is used to check minimum floating representation. There is no way for sign magnitude to present this value, so it is rounded to -4095. The output of case 2 and 3 should be the same.

Case 4, 5, 6: They are used to check significand overflow. All their significands are the same, and their exponents differ by 1. Their significands should all over flow to 5’b10000, and their exponents should increase by 1.

Case 7: 100 is a normal case that has neither overflow nor rounding.

Case 8: 102 is the same as 100, except that its significand rounded up by 1. This is checking normal rounding.

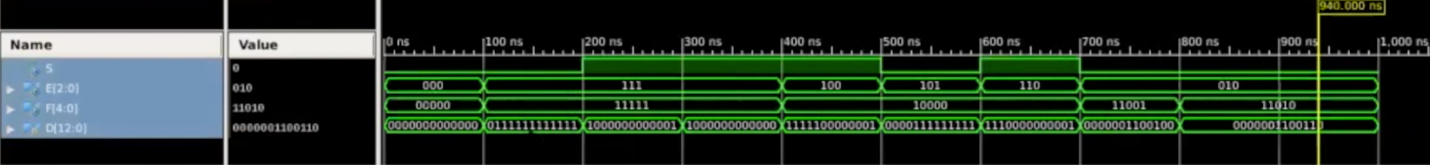


Figure 7: Implementation Waveform

**4. CONCLUSION**

This lab allows us to explore how to build a combinational circuit using multiple modules in Xilinx ISE. It helps us learn how to modularize a larger design to smaller pieces and implement it with higher efficiency.

**5. APPENDIX: DESIGN SUMMARY REPORT**A screenshot of a cell phone

Description automatically generatedA close up of a piece of paper

Description automatically generated

A screenshot of a computer

Description automatically generated