CS M152A - Lab 1 Report

Adam Vuilleumier
UID: 305098071
Lab Section 6 (Rajas Mhaskar)

1. Introduction

In this laboratory, our goal was to program a virtual Field Programmable Gate-Array (FPGA) on the Xilinx ISE that converts a signed 13-bit binary string into a 9-bit floating point value (FPV). I used this 13-bit binary string as the input, D. The output was split into 3 registers: S, E, and F, which represent the sign bit, exponent, and mantissa of the floating point value, respectively. The decimal value of a floating point value can be calculated as follows:

$$V = (-1)^S * 2^E * F$$

A floating point value is quite different from a 2's complement binary string. Each section in an FPV (the 1-bit S, the 3-bit E, and the 5-bit mantissa) must be plugged into the above equation to calculate its decimal value. This allows for a wider range of values to be generated from fewer bits. However, a disadvantage of this format is that certain binary values cannot be represented as FPV's. Furthermore, certain decimal values can have multiple FPV representations, which can lead to design challenges that will be discussed later in this section.

The range of decimal numbers that can be represented by a 13-bit binary string are [-4096,4095]. The range of numbers represented by a 9-bit FPV are [-3968,3968]. Therefore, in this FPGA, any value that has a magnitude of 3968 or greater is represented by the maximum FPV: x_111_11111 (where x is the sign bit).

Additionally, because the mantissa is only 5 bits long, even decimal numbers within the specified FPV range might also be impossible to represent. Fortunately, this FPGA handles these special cases by outputting the closest FPV to the true value (which is usually accurate within ~1%).

Fig 1.1: True decimal value is 253, FPV represents 256

Finally, since multiple FPVs can map to the same decimal number, we use the normalized representation (where the MSB of the mantissa is 1). This ensures that the exponent is not needlessly high and that there is a 1:1 correspondence between each input and each combination of outputs.

2. Design Description

Using Verilog code within the Xilinx ISE, I implemented and simulated this FPGA using one module: FPCVT. The input, D, is a wire, but all other variables are of type reg, including the outputs, E and F. This means that all these values must be assigned within an *always* block.

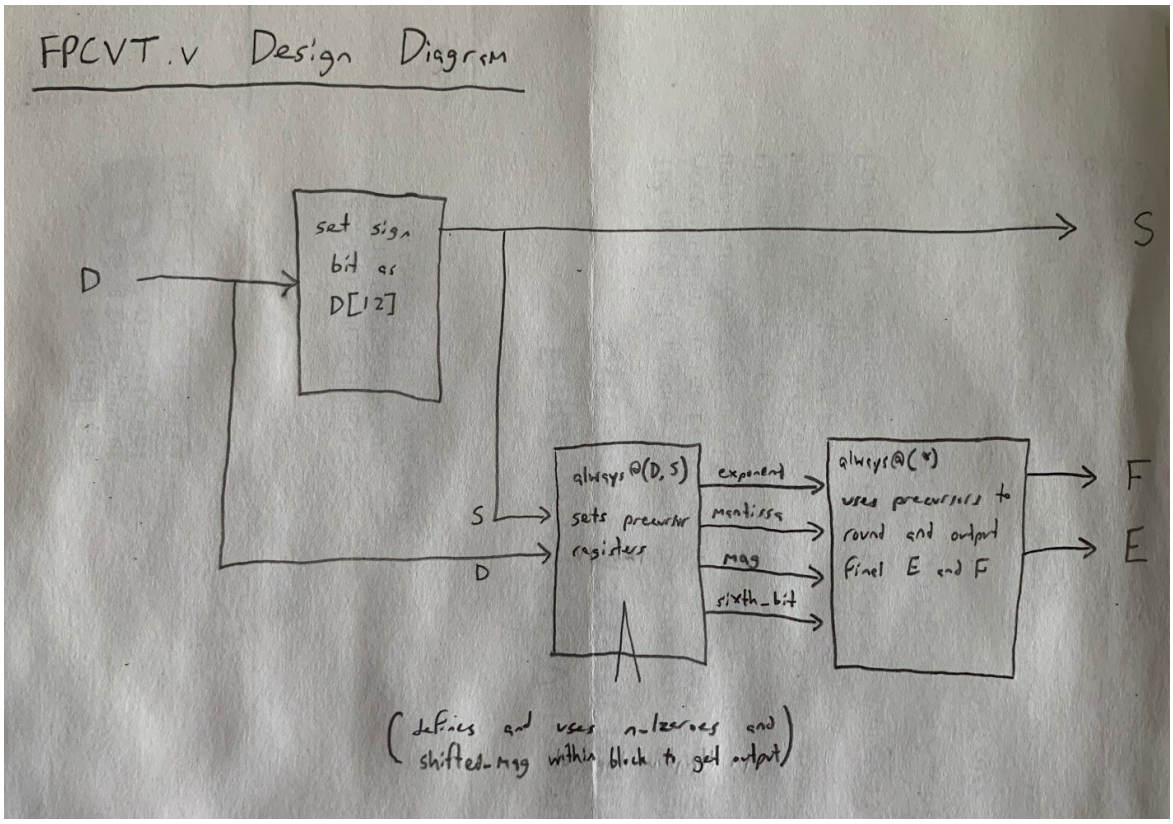

Fig 2.1: Schematic of my FPGA; each box represents an always block

There are three always blocks within my module. The first one sets the sign bit. Its sensitivity list depends on changes to D, and it sets S to D[12], the most significant (sign) bit of the 13-bit binary string, in all cases.

Once S has been set, the second always block uses this S and D to initialize the precursor variables (registers that are not the final outputs). This block first determines the magnitude of the 13-bit string and stores this value in register *mag*. Then, it counts the number of leading zeros in *mag* and stores that in *n_lzeroes.* In the normal case, we subtract this value from 7 to obtain

exponent. Edge cases (such as the instance when *n_lzeroes* > 7) are handled accordingly. *n_lzeroes* is also used to shift *mag* that number of spaces to the left, which results in *shifted_mag*. The most significant 5 bits of *shifted_mag* are assigned to *mantissa*. *Sixth_bit* is also obtained using *shifted_mag.*

The third always block runs whenever there is a change in the values of registers: *mag, mantissa, exponent*, or *sixth_bit*. Its main purpose is to assign final values to the output registers E and F. The registers *exponent* and *mantissa* correlate directly to these outputs, but they are unrounded. Once the edge cases in which *mag* is greater than or equal to the maximum FPV are handled, this third block rounds *mantissa* up by one bit or leaves it the same, depending on the value of *sixth_bit*. It will also adjust *exponent* if *mantissa* overflows its max value of 11111. Once the values of *mantissa* and *exponent* are finalized, they are assigned to F and E, respectively.

```
Design Summary
--------------
Number of errors:      0
Number of warnings:   25
Slice Logic Utilization:
  Number of Slice Registers:                    4 out of 126,800    1%
    Number used as Flip Flops:                  0
    Number used as Latches:                     4
    Number used as Latch-thrus:                 0
    Number used as AND/OR logics:               0
  Number of Slice LUTs:                        76 out of  63,400    1%
    Number used as logic:                      76 out of  63,400    1%
      Number using O6 output only:             57
      Number using O5 output only:              1
      Number using O5 and O6:                  18
      Number used as ROM:                       0
    Number used as Memory:                      0 out of  19,000    0%
    Number used exclusively as route-thrus:     0

Slice Logic Distribution:
  Number of occupied Slices:                   36 out of  15,850    1%
  Number of LUT Flip Flop pairs used:          77
    Number with an unused Flip Flop:           73 out of      77   94%
    Number with an unused LUT:                  1 out of      77    1%
    Number of fully used LUT-FF pairs:          3 out of      77    3%
    Number of unique control sets:              1
```

Fig 2.2: Map Report

According to the map report, there are 4 slice registers (all of which are used as latches), and 76 slice LUTs (all of which are used as logic) in the final FPGA. This report tells us what kinds of features we would need if this FPGA were to be physically assembled.

```
=========================================================================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 4
 12-bit adder                                         : 1
 3-bit adder                                          : 1
 3-bit subtractor                                     : 1
 5-bit adder                                          : 1
# Latches                                              : 7
 1-bit latch                                          : 7
# Comparators                                          : 2
 12-bit comparator greater                            : 1
 4-bit comparator lessequal                           : 1
# Multiplexers                                         : 58
 1-bit 2-to-1 multiplexer                             : 37
 12-bit 2-to-1 multiplexer                            : 2
 3-bit 2-to-1 multiplexer                             : 2
 4-bit 2-to-1 multiplexer                             : 12
 5-bit 2-to-1 multiplexer                             : 5
# Logic shifters                                       : 1
 12-bit shifter logical left                          : 1

=========================================================================
```

Fig 2.3: Synthesis Report

The synthesis report tells us all the specific adders, latches, muxes, and logic shifters needed to create the FPGA. Verilog code implements all of these elements using information from the three always loops described above.

3. Simulation Documentation

In order to test my FPGA, I simulated different inputs using a testbench file (shown below in **Fig 3.1**). In this file are different inputs for D that represent a wide range of sample cases. It is important to cover all the edge cases and exceptions that could occur when converting our 13-bit input into our three FPV outputs, as there are many instances where registers can overflow or cause unintended behavior.

```
D = 13'b0000000111010;   //basic case (easy 5 bit mantissa)
#100;

D = 13'b0111110000001; //case of FPV overflow (> 3968)
#100;

D = 13'b1000000000000; //case of FPV overflow (< -3968)
#100;

D = 13'b0000011111101; //case of rounding oversaturation of mantissa
//FPV rounding error: module gives 256 even though binary rep = 253
#100;

D = 13'b1110001111110; //case of rounding oversaturation and negative
#100;

D = 13'b0000000000010; //case of >8 leading 0's
#100;

D = 13'b1111111111100; //case of no leading 0's (small negative)
#100;

D = 13'b0000000000000; //case when input = 0
#100;
```

Fig 3.1: Annotated test inputs used in my testbench file

When the above test cases were simulated in Xilinx ISE, the FPGA produced the intended output for every case (shown in **Fig 3.2**). However, when D = 0000011111101, the output decimal value did not match the input decimal value. This was still the intended behavior though, as 253 (the input decimal value) cannot be represented as an FPV. Instead, the FPGA outputs the FPV corresponding to 256, which is only a 1.2% error from the correct value.

Fig 3.2: Output of the test cases shown in Fig 3.1

4. Conclusion

By the end of the lab, I had successfully programmed an FPGA that can convert a signed 13-bit binary number into a floating point value with a sign bit, a 3-bit exponent, and a 5-bit mantissa. I was also able to test this FPGA with a testbench file that included many edge cases (see **Fig 3.1**).

One challenge I faced during the course of this project stemmed from a misunderstanding I had about always blocks. I wrongly assumed that these blocks executed the code inside sequentially, and thus only had one always block for my entire FPGA. This caused problems because, at that time, I did not use the precursor registers *mantissa* and *exponent*. Instead, I was altering the values of E and F directly multiple times within the block. My logic depended on the code executing sequentially, but since always blocks are combinational, my choice had unintended consequences.

To solve this problem, I used the three always block configuration described in section 2. Since I was now using precursor variables, I could assign values to these variables once and then in the next always block, assign altered versions of these variables to my final outputs. This mimicked the sequential kind of logic that I needed for my algorithm.

One suggestion I have to improve this lab would be to give the students more freedom in choosing the representation of our output. I would have liked to design a FPGA that converts other types of inputs into other kinds of outputs (ie. unsigned binary to 8-bit FPV).

I would finally like to thank my TA, Rajas Mhaskar, for his assistance in explaining Verilog to our class as well as for his help in installing the Xilinx ISE when I was having errors with the license file.