Lab 5

A 32-bit Floating Point Arithmetic Unit

In this lab, you are required to design and implement a 32-bit floating point arithmetic unit that performs addition, subtraction, multiplication and division with 32-bit floating point numbers.

A floating-point number consists of 3 parts: 1-bit *sign*, 8-bit *exponent* and 23-bit *mantissa*. You are going to work with IEEE single precision floating point standard representation which requires 32 bits. The first bit is the *sign* bit $\mathtt{s}$, the next 8 bits are the *exponent* bits $\mathtt{e}$, and the final 23 bits are the *mantissa* bits $\mathtt{m}$. The *sign* bit $\mathtt{s}$ is 0 and 1 for unsigned and signed numbers, respectively. The base of the *exponent* is 2. Instead of the signed *exponent* $\mathtt{e}$, the value stored is an unsigned 8-bit integer $\bar{\mathtt{e}}$=$\mathtt{e}$+127, called the *excess-127* format. Therefore, $\bar{\mathtt{e}}$ is between 0 and 255. For example, a number with *exponent* 2 is represented with $\bar{\mathtt{e}}$=2+127=129 in floating-point format. There is also an implied bit of *mantissa* which is always 1. However, it is not shown in *mantissa*. For example, *mantissa* $(01010000001000000000000)_2$ actually represents the number $(1.01010000001000000000000)_2$. Therefore, *mantissa* can take a value between 1 and 2. Therefore, a number V given in 32-bit floating point format with $\mathtt{s}$, $\bar{\mathtt{e}}$ and $\mathtt{m}$ is equal to

$$V = (-1)^{\mathtt{s}} \times (2)^{\bar{\mathtt{e}}-127} \times (1.\mathtt{m})_2$$

For example, $(91.34375)_{10}$ is represented as

$$(2)^6 \times (1.01101101011)_2 = (-1)^0 \times (2)^{133-127} \times (1.01101101011000000000000)_2$$

Therefore, $(91.34375)_{10}$ is represented with $\mathtt{m} = (01101101011000000000000)_2$, $\mathtt{s} = 0$ and $\bar{\mathtt{e}} = (133)_{10} = (10000101)_2$ in 32-bit floating point format as

01000010101101101011000000000000

where bits with red, blue and green color represent *sign*, *excess-127* and *mantissa*, respectively. This is the format you will use for providing input to your hardware, $\{\mathtt{s},\bar{\mathtt{e}},\mathtt{m}\}$.

Your hardware should take two inputs in 32-bit floating point format and produce one output in the same format as its inputs (a number in 32-bit floating point format). You can assume that the input numbers will be given such that there won't be overflow after addition, subtraction, multiplication and division operations.

Your hardware should be able to perform 4 different operations on two inputs in 32-bit floating point format: addition, subtraction, multiplication and division. Along with input numbers, your hardware should take selection signal *sel* as input for determining the operation type as shown in Table 1.

There are some special representations you need to take care of:

- If $\bar{\mathtt{e}} = 0$, $\mathtt{m} = 0$, then the number V is 0.

Lab 5

sel	Operation
2'b00	Addition
2'b01	Subtraction
2'b10	Multiplication
2'b11	Division

Table 1: Operation Selection Signal

- If $\mathtt{s} = 0$, $\overline{\mathtt{e}} = 255$, $\mathtt{m} = 0$, then the number V is $+\infty$.
- If $\mathtt{s} = 1$, $\overline{\mathtt{e}} = 255$, $\mathtt{m} = 0$, then the number V is $-\infty$.
- If $\overline{\mathtt{e}} = 255$, $\mathtt{m} \neq 0$, then the number V is `NaN` (Not a Number). Please note that if one of the input numbers is `NaN`, the output should be `NaN`.
- If $\overline{\mathtt{e}} = 0$, $\mathtt{m} \neq 0$, then the number V is defined as $(-1)^{\mathtt{s}} \times (2)^{-126} \times (\mathtt{0.m})_2$.

You will be given two Python codes, `fixed_to_float.py` and `float_to_fixed.py`. The code `fixed_to_float.py` converts numbers in fixed point format into floating point format. You can use this code for generating test inputs for your design. The code `float_to_fixed.py` converts numbers in floating point format into fixed point format. You can use this code for checking the correctness of the output of your design. You can again use the `Google Colab` for running the codes.

Your hardware should work with the following I/O protocol (please note that *rst* signal is not shown for simplicity):

1. Hardware accepts new inputs if *busy* signal is 0.
2. For providing input, *start* signal should be 1 for one clock cycle. Then, in the same cycle, input numbers and selection signal should be entered using *numin1*, *numin2* and *sel* signals respectively. Your hardware will not accept any more input until it finishes its operation with current inputs and outputs the result.
3. As *start* signal is set as 1, *busy* signal should be set as 1 (and remains as 1 until the operation is finished.).
4. When your hardware finishes its operation, *finish* signal will be 1 for a single clock cycle and ,in the same clock cycle, resulting number will be at the output *numout*. Then, *busy* signal should be brought to 0 and the hardware should accept new input. An example timing diagram for input and output operations is shown in Figure 1.

Write a behavioral Verilog model for your hardware design. In your Verilog model, your Verilog module should have the following interface:

Lab 5

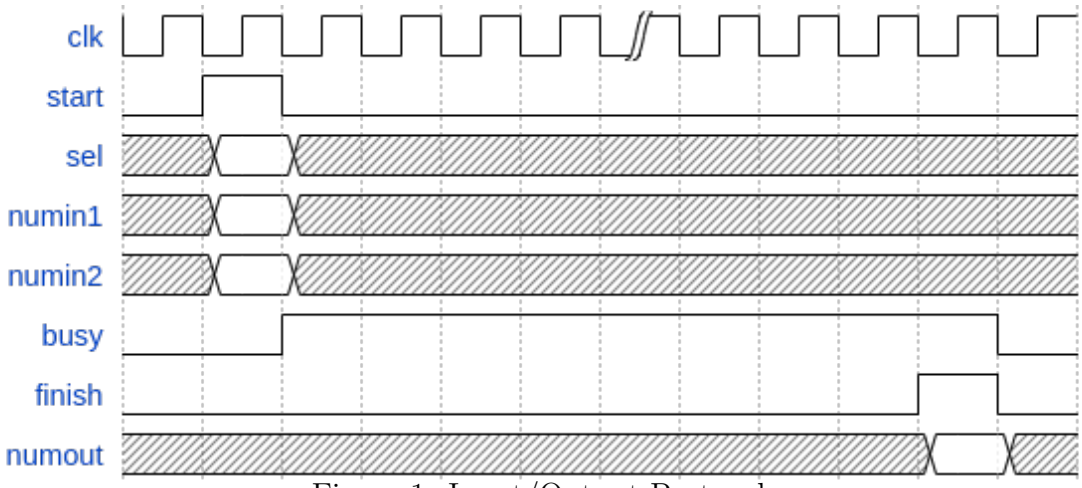

Figure 1: Input/Output Protocol

```
module FloatingPtAU(clk,rst,start,sel,numin1,numin2,
                    busy,finish,numout);
    input clk,rst;
    input start;
    input [1:0] sel;
    input [31:0] numin1,numin2;
    output reg busy, finish;
    output reg [31:0] numout;

    // ...

endmodule
```

You should also write a Verilog testbench that verify the correctness of your Verilog model. In the testbench, you should apply input values to your model and compare its outputs with the expected results. Make sure that you test your design for all four operation types. Please note that you should use your own test cases. Using your friend's test cases will be considered as plagiarism. After verifying the correctness of your design, synthesize (not implement) your design for Artix-7 FPGA (Family:Artix 7, Device:XC7A100T, Package:CSG324, Speed:-3) using Xilinx ISE WebPack 14.7 (**You will not implement your design on an FPGA board. You will just report synthesis results.**). Then, you should write a lab report (in .pdf format) with *readable* English and good structure (dumping many figures and raw synthesis/implementation reports into a pdf file is not a lab report.). Your report should include the following information:

1. Write a summary of the assignment and your design.
2. What is the hardware area-cost (slices, register, LUTs, DSPs, BRAMs, etc.) of your design? What is the maximum achievable frequency of your design?

3. How many clock cycles does it take **on average** to finish addition, subtraction, multiplication, division operations?

Finally, put all of your Verilog project directory and your report into one ".zip" file named Lab5_username(s).zip (e.g. Lab5_berkea_erdinco.zip) and submit this zip file using EE310 SUCourse website. Please note that we will use a plagiarism detection software for your lab assignments.