Homework 1: Design of Single-Precision Floating-Point Fused Multiply-Add (FMA)

Handout 2021/10/12

Due 2021/11/09

1. Given three IEEE 754 single-precision floating-point numbers A, B, C. the fused multiply-add (FMA) performs the computation of D= A*B+C. Design an arithmetic unit that computes the FMA. Note that your design should perform the alignment/normalization only once for the combined floating-point FMA. Hint: after extracting mantissa and exponents from A, B, C, perform the alignment to adjust the exponents of A*B and C so that they are the same. Then, perform the fixed-point operation, followed by normalization and rounding.
2. Since the numbers could be positive or negative, you should perform the multiplication and addition with operands represented in 2's complement format.
3. Assume the tuples of (sign, exponent, and mantissa) for the five numbers: A, B, C, P=A*B, and D are (sa, ea, ma), (sb, eb, mb), (sc, ec, mc), (sp, ep, mp), and (sd, ed, md) respectively. The format of the magnitude of the mantissa for A, B, C is Q1.23 with the hidden integer bit of 1. But after converting the sign-magnitude representation to 2's complement format, the new signed mantissa format for A, B, C become Q2.23 with two integer bits because the range of signed mantissa is between -2 and 2. Thus, the mantissa format for P=A*B is Q4.46 since the bit-width of the product is twice large and the mantissa range of the product is between -4 and 4. Also, you need to sign-extend mantissa of C to Q4.23 and sign-extend mantissa of P to Q4.46 before the mantissa addition to compute P+C in order to cover the possible increase of mantissa range during the addition, i.e., the range of the mantissa after the complete FMA operation is between -6 and 6. In summary, you need to consider the possible increase of mantissa range during the entire process of FMA computation, including conversion of sign-magnitude to 2's complement signed format, the product, and the addition.
4. First, design a non-pipelined (i.e., single-cycle) version for the FMA. That is, all the hardware is pure combinational logic without the use of any flip-flop. Write a testbench to verify your register transfer level (RTL) design.
5. Use Synopsys Design Compiler to synthesize the non-pipelined RTL design to gate-level netlist, and verify the gate-level design again. Note that you should synthesize your design with at least three different constraints: *delay-optimization*, *area-optimization*, and *in-between* (with reasonable delay constraint). In general, the curve of area versus delay looks look like a reciprocal curve.
6. What is the total area of the non-pipelined FMA for each synthesis (with a particular constraint)? What is the critical path delay of the non-pipelined FMA for each synthesis (with a particular constraint) ? Note that you have at least three results, one for each synthesis with the aforementioned constraints.
7. Then, design a pipelined FMA by partitioning it into the following pipelined stages. Note that each pipelined stage is modelled as a separate Verilog module.
   Stage 1: upack input A, B, C
   Stage 2: multiplication P=A*B
   Stage 3: alignment for P and C
   Stage 4: addition P+C
   Stage 5: conversion of 2's complement results back to sign-magnitude, normalization, and rounding
   Stage 6: pack result and generate D
8. Use Synopsys Design Compiler to synthesize the pipelined RTL design to gate-level netlist, and verify the gate-level design again. Note that you should synthesize your design with at least three

different constraints: delay-optimization, area-optimization, and in-between (with reasonable delay constraint).

9. What is the total area of the pipelined FMA design? What is the critical path delays of each pipelined stage in the pipelined FMA? What is the critical path delay of the entire pipelined FMA? Which pipelined stage is the critical stage, i.e., with the longest delay? Note that you should have at least three results, one for each different synthesis constraint.
10. Compare the area and delay of the floating-point FMA design with the fixed-point multiply-add design. You might refer to the following RTL code segment for fixed-point multiply-add.

```
wire signed [31:0] a, b, c, d, p;
wire signed [63:0] dp;

assign dp = a * b + {32{c[31]}, c}; // sign-extend c before addition with a*b
assign d = dp[63:32];    // take the 32 most significant bits (MSB) as the final output
```

11. You have several weeks to do this entire homework. But you need to submit intermediate reports every week. The weekly reports show your progress in each week. The complete report dues as shown above. The suggested progress report for each week is as follows.
    Week 1: Draw a block diagram of overall architecture and identify the major components in the architecture such as unpack, multiply, alignment, add, normalize, round, unpack. Give detailed description for each of the major components. Design and verify some of the components.
    Week 2: Design and verify the rest of the components. Then, synthesize the non-pipelined FMA and verify the gate-level netlists with various area/delay constraints
    Week 3: Design and Verify the pipelined FMA
12. Upload your report to NSYSU course website with proper file names showing your student ID and homework number. Example: ARITH HW1_1 M023040099 王小明. The attached compressed file should include all HDL codes, results of simulation, results of synthesis, final report, and all the supporting document.
13. The grading criterion are as follows:
    (40%) RTLcode, testbench, simulation results, and synthesis reports for the non-pipelined FMA**.**
    (40%) RTLcode, testbench, simulation results, and synthesis reports of the pipelined FMA
    (20%) Final report which contains the figures for the overall architectures, explanations of your designs and verification strategy, and your comments.