LAB Assignment 4

Floating Point Multiplier

Dr. Ahmed Elhossini

Introduction

In this lab assignment, you will implement a simple floating point multiplier that uses non-standard 23 bits long format. The purpose of this lab is get familiar with various aspects that must be considered during the implementation of floating point arithmetic circuits. In this implementation, some abnormal values/operations will be ignored for simplicity.

Pre-requisites

1. Knowledge of VHDL.

Objectives:

1. Implement 23 x 23-bits floating point multiplier using VHDL.
2. Simulate the multiplier using Xilinx ISIM.
3. Show performance evaluation - critical path delay, and number of lookup tables required

Equipment and Tools

1. Xilinx ISE tool chain.

Lab Time:

From January 25th to Feb 22th 2017.

Introduction: Floating Point Multiplier

The floating-point multiplication circuit is shown in Figure 1. As discussed in the lecture the packing and packing modules are responsible for data validation. Unsigned multiplier is used to multiply the significand of both numbers while exponents are added. After multiplication, normalization and rounding stages are required. Normalization is performed by shifting (in this case to right 1 position), while rounding can be implemented using truncation. In this lab assignment, you will implement this multiplier circuit.

Floating Point Format

The floating-point number format is 23 bits composed of 3 parts as shown in Figure 2:

1. 1-bit for the sign
2. 8 bits for the exponent biased by 127.
3. 14 bits for the significand (15 with the hidden 1)

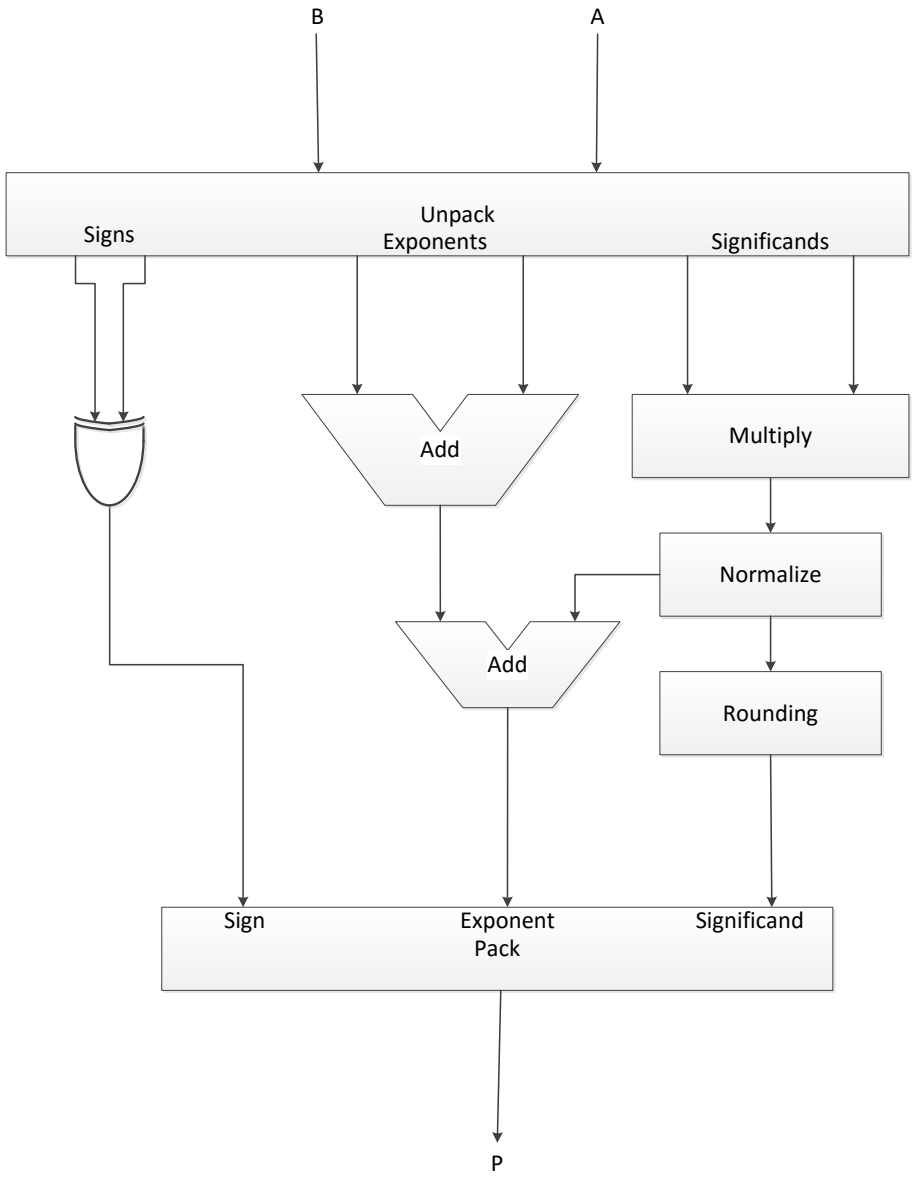

Figure 1 Floating Point Number Multiplier

+/-	Exponent 8-bits	Significand 14-bits

Figure 2 Floating Point Format

Lab Steps

Step 1. Radix-4 Booth multiplier

Use radix-4 booth multiplier implemented in Lab-assignment 3 (Figure 3) to multiply the significand part. The significand part is only 14-bits; you have to append 1 so the input of the multiplier is 15x15. The extra bit should be set to '0'. The result is 30 bits, 2 bits integer and 28 fractions. Normalization will shift 1 bit to the right, which rounding will remove all extra bits (the first 15 bits only).

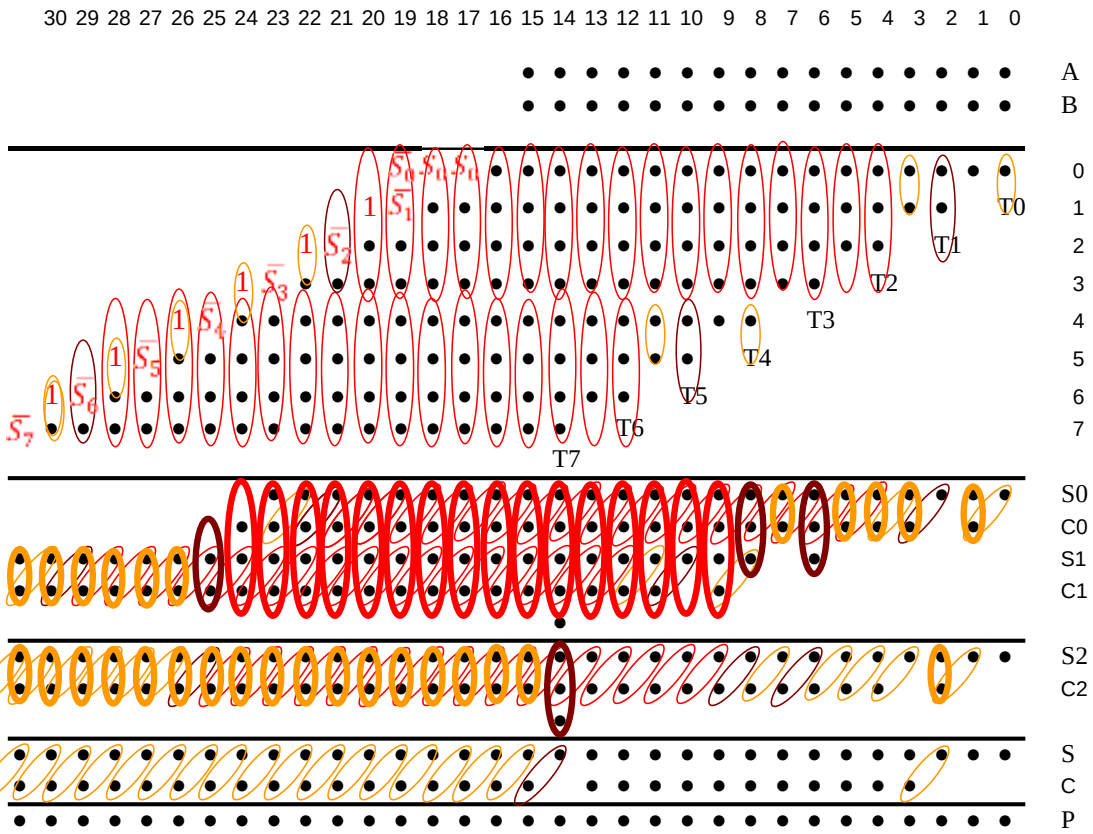

Figure 3 Dot Diagram of Radix 4-booth multiplier. 4-2 counters are used in the first two stages.

Step 2. Exponent Addition

Use the 8-bits conditional sum adder implemented in lab-assignment 1 to add the exponents. Note that, an extra adder is required to adjust the exponent after normalization. As multiplication result may be normalized by shifting 1 bit to the right, the exponent may be adjusted by adding '1'. This can be done during the exponent addition by setting the carry-in input to '1' during addition. Exponent bias should be done here by subtracting 127 from the result.

Step 3. Normalization and Rounding

Normalization and Rounding can be combined if we ignore the abnormal values (when exponent =0). This does not require an actual circuit. Rounding and normalization, using this approach can be performed by taking only bits (29 to 15) or (28 to 14) from the multiplication results, depending on the most significant bit. This will form the required 15 bits result that will be passed to the output.

Step 4. Sign Generation, Unpacking and Packing

The output sign is generated by XORing the sign of both operands. In the unpacking stage the inputs of the adder and multiplier should be prepared. At this stage, the exponent and significand values for both operands are checked for 0. If both are equal to 0, the operand is equal to 0, and the multiplication result should be 0 as well. Multiplication should not take place at this time. In our simplified implementation, this is the only abnormal value that will be checked. The logic required to check the 0 value is discussed in the lectures.

At the output stage, the value of the exponent (the adder output) should be checked for overflow (underflow) and the 0 value. Flags should be generated at this stage:

1. Zero flag
2. Overflow/underflow flag
3. Sign flag

Deliverables:

1. VHDL code for all modules implemented.
2. Simulation results for the floating point multiplier: Waveform for the final circuit, for both behavioral and timing simulations (Post place and route).
3. Performance analysis
    a. Report the maximum frequency, and critical path delay.
    b. Area: Number of LUT and other resources.