University of California, Davis
Department of Electrical and Computer Engineering

EEC180B DIGITAL SYSTEMS II Spring Quarter 2014

LAB 5: Booth Radix-4 Multiplier Design

Objective: In this lab, you will design a Booth radix-4 multiplier and compare its performance with your Booth multiplier from Lab 3. You will perform timing simulation and power analysis on both designs in order to determine which one is more energy-efficient.

This is a one-week lab but you are allowed to work with a partner. The partner should be from your lab section. Both partners will get the same score.

I. Booth Radix-4 Multiplier

In the standard shift-and-add algorithm for multiplication, you examine the multiplier one bit at a time starting with the LSB in order to generate partial products. As we saw in Lab 3, the Booth algorithm examines the multiplier *two* bits at a time in order to determine the operation – add-shift, subtract-shift, or just shift. The goal of Booth's algorithm was to reduce the number of addition operations by avoiding addition when there were strings of 0s and 1s. In Booth's day, shifting was faster than adding so this technique could improve multiplier performance. For some multiplier bit patterns, Booth's algorithm is faster than standard multiplication, but not for all.

In the radix-4 Booth algorithm, you will consider multiplier bits in blocks of three, such that each block overlaps the previous block by one bit. The blocks start at the LSB, although the first block assumes a previous bit of 0 and only uses the two least significant bits of the multiplier. The following table illustrates the operations associated with each pattern of three bits:

Block	Operation
000	Do Nothing
001	Add multiplicand
010	Add multiplicand
011	Add 2*multiplicand
100	Subtract 2*multiplicand
101	Subtract multiplicand
110	Subtract multiplicand
111	Do Nothing

After examining the current block and performing the necessary operation, the partial product and multiplier are shifted right **twice**. Note that this must be an ***arithmetic* right-shift** in order to preserve the sign bit of the partial product.

Design Requirements

1. Your multiplier design must be *scalable*.
   Write your code so that your multiplier can be easily scaled to different sizes by simply changing a bit-width parameter. You should be able to change your design from an 8-bit multiplier to a 16-bit or 24-bit multiplier by changing a parameter. (Note: for simplicity, you can assume an even bit-width will always be used.)

2. Your multiplier design must be *synthesizable*.
   You should be able to synthesize and download an 8-bit version of your radix-4 multiplier and test it in exactly the same way that you tested your radix-2 Booth multiplier in Lab 3.

3. Your multiplier design must work with your Lab 3 test bench.
   Your port definition for the radix-4 multiplier must match the port definition for your radix-2 multiplier.

4. Your multiplier hardware should be very similar to the hardware design in Lab 3, using an accumulator (A), register B to hold the multiplier, and register C to hold the multiplicand. The final product will reside in registers A and B. In addition, your finite state machine for controlling your multiplier will be very similar to the one used in Lab 3. This will help to make your timing and power comparisons more meaningful. The number of state transitions (hence clock cyles) required for a radix-4 multiplication, however, should be much lower than the number for a radix-2 multiplication.

   Note: you will need to be careful about keeping the sign correct for partial products. Be especially careful of the case when the multiplicand is the smallest negative number (1 followed by *n-1* 0s).

Design Procedure

Perform the following steps:

1. Using the radix-4 Booth algorithm, multiply -9 by -13 *by hand* similar to the example in Lab 3. Choose the smallest value for n that will work. Do at least one other test case by hand to verify that your algorithm works correctly.

2. Write the Verilog code for an 8-bit radix-4 Booth multiplier (n=8).

3. Simulate your Verilog design with the same test bench and test vectors that you used in Lab 3. Verify that you get the correct results for all cases.

4. (Optional) Download your 8-bit radix-4 Booth multiplier to the DE2 board and verify that it works correctly.

5. Change the bit-width to 16 (n=16), synthesize and perform a timing simulation. Compare the results to a 16-bit radix-2 Booth multiplier. Also, perform a power analysis and compare the results with your 16-bit radix-2 multiplier from Lab 3.

6. Demonstrate your simulations to your TA and have him sign a verification sheet. Print your simulation waveforms for a single multiplication.

III. Power and Energy

In a CMOS circuit, there are two sources of power consumption – dynamic power that is proportional to the clock frequency and square of the supply voltage and static power that is largely independent of the design in the case of an FPGA. Dynamic power itself has three sources – (a) I/O pads (b) FPGA core and (c) clocking network.

Power is the average energy per unit time is reported in watts (joules/sec). One could reduce the power by performing the same operation very slowly that is spending lot of time to do the same operation. However, that is not a good measure for the *efficiency of a circuit*, because in general we are interested in improving both power consumption and performance of a circuit. Also, in a battery-operated device, we are interested in prolonging the battery life, which means "energy" is a better metric for the efficiency of a circuit.

In general, we are interested in energy per operation, the operation being 16x16 multiplication in this lab exercise. You can compute this by multiplying the average power (reported by the Power Play tool) with the time required to complete a 16x16 mulitply operation. The latter depends on your specific implementation.

IV. Lab Report

For your lab report, include the following:

- Lab Cover Sheet with signed TA verification for successful timing simulation.
- Complete Verilog source code for radix-4 multiplier.
- Simulation waveforms of your timing simulations.
- Hand-execution of radix-4 algorithm for two test cases.
- State transition diagram for your multiplier controller state machine.
- Resource report indicating how many FPGA resources were required for each the designs.
- Comparison of energy-efficiency of radix-4 vs. radix-2 multiplier designs.
- ***Prove*** that the radix-4 Booth algorithm is correct.

V. Grading Guidelines

- Part I Timing Simulation 25 points
- Part I Power Analysis 25 points
- Lab Report 50 points