Lab 2: Multipliers

Introduction

In this lab you will build two multiplier circuits and then simulate them using *ModelSim* software with your own testbench. Please read the accompanying documents *Testbench Tutorial* and `Creating Generic Hardware`.

Part I: Carry Save Multiplier

A simple multiplier performs multiplication in hardware in a manner similar to performing it by hand. This type of multiplier is called an *Array Multiplier*. However, array multipliers are quite slow. Faster multipliers can compute a product of two numbers more quickly. For Part I you will design an **unsigned** 8x8 *Carry Save Multiplier* (CSM). Figure 1 shows an example of 4-bit carry save multiplier architecture.

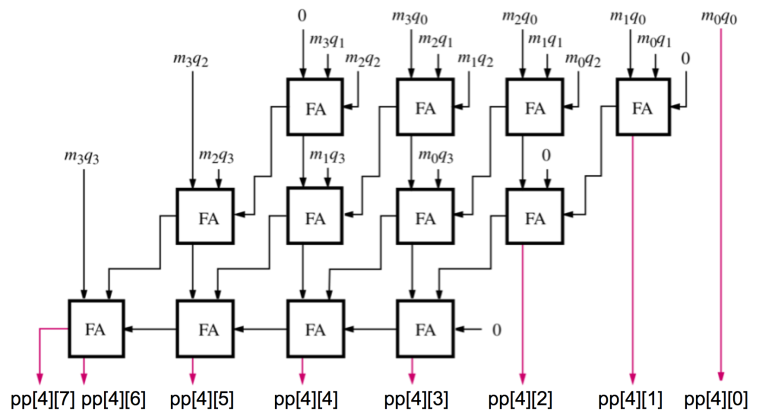

Figure 1: 4-bit Multiplier Carry Save Array (Figure from pg. 354, "Computer Organization and Embedded Systems", 6th ed., Hamacher, Vranesic, Zaky and Manjikian)

Figure 1 shows the multiplication of the quotient q0-q3 by the multiplicand m0-m3 to produce the 8-bit result, p0-p7. This 4x4 multiplier uses 3 rows of full-adders (FA) while your multiplier should consist of 7 rows of 8 full-adders each. The input to and output from each adder is refereed to as a 'partial product (pp)'. For the 4x4 multiplier shown in Figure 1, the final output is given as pp[4][7:0].

Structural coding, by instantiating several rows, is encouraged. You may find the `generate` statement helpful for instantiating many copies of your sub-modules. See the accompanying document *Creating Generic Hardware* for a tutorial.

You must not use the * and + Verilog operators in the design of your circuit. To ensure this, the values of each row of your multiplier will be declared as outputs. **The automarker will test the partial products of each row (i.e. pp[1], pp[2], etc.)**.

The Testbench

After you've created your 8x8 unsigned CSM module, the next thing you should do is to simulate it to make sure it is correct. For this, you will write a testbench to test every possible combination of inputs (all 65536 of them), and compare the result of your multiplier to the value returned by the Verilog multiplication operator *. Please see the accompanying document *Testbench Tutorial*, which describes a very similar testbench to the one required for this lab, except for testing an adder rather than a multiplier.
If a mismatch occurs between the expected product and your module's output, you should print a message to the ModelSim console using the `$display` command indicating which test case failed. This will help you locate the bug. If all test cases pass, print a message indicating this as well. Run your testbench in ModelSim to verify that your multiplier is correct. If the testbench finds errors in some test cases, debug and fix your multiplier until all results pass. Don't forget that you can set the radix of the displayed signals to *Decimal* in the wave window.

Part II: Wallace Tree Multiplier

In part II you must design an even faster multiplier than the CSM, namely the Wallace Tree Multiplier (WTM). The WTM takes uses the fact that the full-adder blocks used in multipliers have 3 inputs (namely A, B and Cin), but only two numbers are added in most multiplier designs. This results in the WSM having a smaller delay, particularly for larger multipliers.

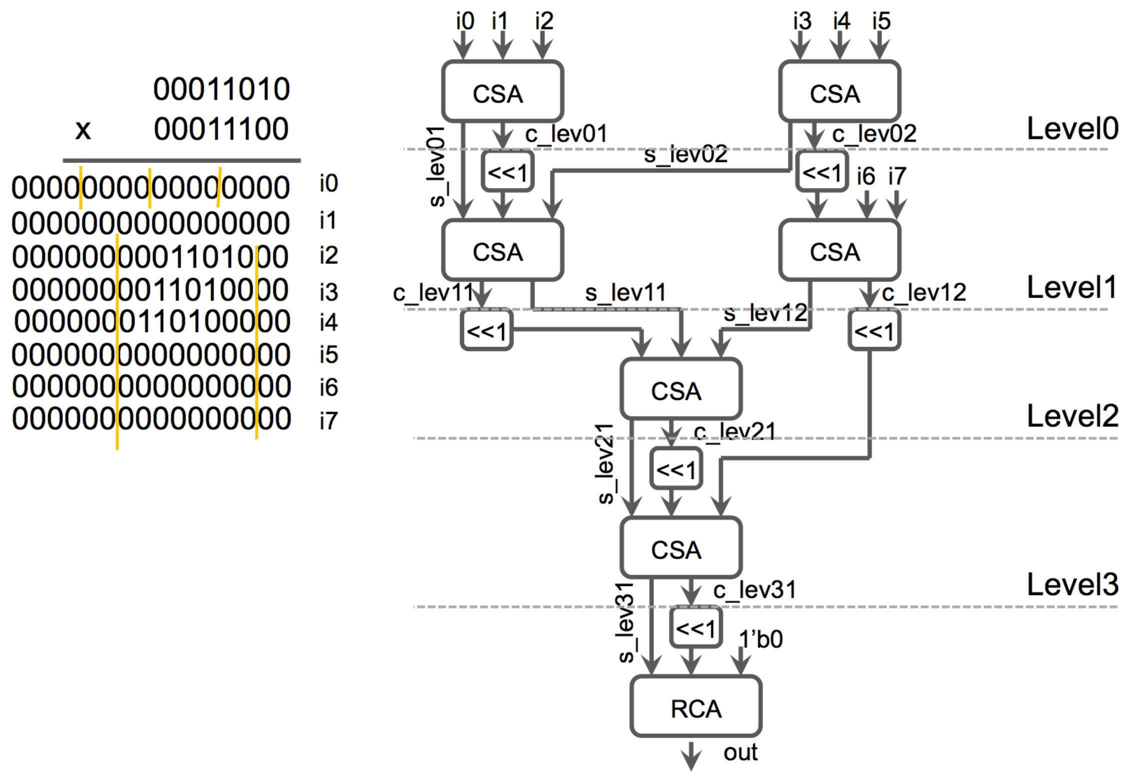

Figure 2: 8-bit Wallace Tree Multiplier Architecture

Figure 2 shows the block diagram of a 8x8 WTM. You must design an **unsigned** 8x8 WTM for Part II. The implementation is based on a pipeline of carry save adders (CSA) followed by a ripple carry adder (RCA) as the final stage. The inputs to the CSAs at each level are also shifted left by 1 bit before being input. The

calculation on the left shows the values for i0 - i7, which are the inputs to the CSAs in level 0 and level 1. To help you, the provided lab kit includes parameterized code for the CSA and RCA. Once again, to ensure you are performing all the steps correctly, The automarker will test the carry and save outputs of Level2 and Level3.
Please be warned that the WTM is a conceptually difficult multiplier so take your time to understand the design clearly before attempting to implement it. You are encouraged to learn more about the WTM from online resources as well.
You must repeat the steps provided in Part I to implement a WSM, test it using ModelSim using your own testbench. **Once again, you must not use the * and + Verilog operators in the design of your circuit.**

Preparation

- Familiarize yourself with the Generate statement by reading the `Creating Generic Hardware` tutorial.
- Download the starter kit for this lab. Separate files are provided for parts 1 and 2.
- Name the finished code part1.sv and part2.sv respectively.

In-Lab

Go through the provided tester for both part1.sv and part2.sv to make sure that your code is able to be recognized by the automarker

Submission

Your should submit part1.sv and part2.sv using the instructions provided in the submission document.