

ECE 368—Spring 2009, Instructor: Prof. Shantanu Dutt

Lab 6 : Part I: Due Tue, April 9, Part II: Due Tue, April 16

Purpose

The purpose of this lab is designing a datapath (an add-and-shift Booth's multiplier, in particular) and an FSM to control the datapath using techniques discussed in class.

The lab is divided into two parts: (i) the datapath (multiplier) that needs to be structurally described in VHDL and simulated with different control and data inputs to check if different operations in it (e.g., add, subtract, shift right, etc.) perform correctly; (ii) the controller FSM described behaviorally in VHDL, connected to the datapath component, and simulation of the entire multiplication process for different random inputs.

Design Problem – Part I

Assume that the delay of a k -input AND/OR/NAND/NOR/NOT gate is $2k$ ns, and that of a k -input XOR/XNOR gate is $3k$ ns.

- (1) Give a datapath, i.e., block diagram, (of the type given in the lecture notes for regular A&S multiplication) for performing Booth's multiplication for 16-bit input numbers. Also show clearly **all** the **labeled** control signals going to the various components like registers and shift registers to control various functions (e.g., loading, shifting, reset, etc.). The control signals are generated by a controller (basically an FSM) at appropriate clock cycles. The input to the controller are called *status signals* and include for e.g., the current and previous bits of the Q register. Show the controller as a rectangular box (without any inside details) and show all the status signals going to it. **100**

For consistency use the following self-explanatory control signal labels going to the various registers and other logic (e.g., 2's complement logic):

- AC_wr : = 1 for enabling write of the AC register;
- AC_shr : = 1 for SHR of AC, = 0 for load of AC from external bus; Thus to load AC from an external bus, we need: $AC_wr = 1$, $AC_shr = 0$, and to SHR AC we need: $AC_wr = 1$, $AC_shr = 1$.
- AC_clr : = 1 means AC is reset to 0 (asynchronously)
- Q_wr : = 1 for enabling write of Q;

- Q_shr : = 1 for SHR of Q, = 0 for load of AC from external bus;
- Q_clr : = 1 means Q is reset to 0 (asynchronously)
- M_wr : = 1 for loading M;
- M_clr : = 1 means M is reset to 0 (asynchronously)
- $Cout_reg_wr$: = 1 for loading the 1 bit Cout_reg (with the carry out of the adder);
- $Cout_reg_clr$: = 1 means Cout_reg is reset to 0 (asynchronously)
- add_sub : = 0 for add, =1 for subtract (obtains negative of multiplicand Y)
- cnt_up : = 1 for counting up (happens in the next cc, just like a load)
- cnt_clr : = 1 means counter is reset to 0 (asynchronously).

For any other control signals not listed above, you can use your own labels.

For status signals use:

- $q0, q_prev$: LSB and previous LSB of Q; note that the q_prev bit needs to be stored in a 1-bit register, say Q_prev , that the LSB of Q is shifted into when AC-Q is shifted right. Thus there should be a clear signal for Q_prev (and any other control signal you might need).
- $count$: value/output of the counter.

For any other status signals not listed above, you can use your own labels.

- (2) Using registers, shift registers (both of these need to be defined structurally using component D-FFs, Muxes, etc.), Muxes, a counter (the latter two may be described completely behaviorally), and your 16-bit CLA adder (or RCA if you did not get your CLA adder working) as components, and other *random* or *glue* logic (e.g., AC[n] logic, inverter logic for obtaining 2's complement of the multiplicand Y) give a *structural* VHDL architecture description of the datapath (block diagram) obtained above for performing 16-bit multiplication using Booth's algorithm. **250**

Note: The component D-FFs of registers need to be described behaviorally with a load control signal, an asynchronous reset/clear signal and the usual positive edge-triggering. The shift signals are not signals to the D-FFs but are signals used in the shift registers (AC, Q, etc.), e.g., to MUXes used

Test Bench – Part I

Instantiate the 16-bit Booth's multiplier datapath and test all operations in it by a combination of different data and control inputs: **150**

1. Loading of AC, Q, M registers
2. Resetting of AC register
3. $AC \leftarrow AC + M$ – show for 4 different data inputs to the adder
4. $AC \leftarrow AC - M$ – show for 4 different data inputs to the adder
5. Correct AC[n] logic generation and storing in FF for overflow and no-overflow conditions (these conditions can be generated by loading AC and M registers with appropriate values and performing addition or subtraction on them). Show these for 4 different AC, M data for each condition (overflow and no overflow).
6. Right shift of AC[n]-AC-Q register combination (show for 8 different data inputs, four with and four without overflows)

NOTE: You can combine the last 4 items in one simulation set so that they are all accomplished with one set of 8 data inputs loaded to AC and M registers.

Design Problem – Part II

- (1) Assuming that your clock period is such that loading, shifting and resetting registers take 1 cc while add/subtract takes 3 cc, design the Moore FSM for the controller for 16-bit Booth's algorithm, show all output signals generated in it in each state and also show near each state using a register-transfer notation (RTL) (e.g., $AC \leftarrow P_i + Y$) what is accomplished in each state. **150**
- (2) Describe the controller FSM designed above behaviorally in VHDL using one or more processes in an architecture in which the 16-bit Booth multiplication datapath of Part I is instantiated. **200**

Test Bench – Part II

Instantiate a 16-bit Booth's multiplier from part (3) in a test bench that does the following:

1. A process that generates 25 random test inputs.
2. A process that generates a clock with period T_{clk} such that $T_{clk} \approx 1.1 \times \lceil cla_delay/3 \rceil$, where cla_delay is the maximum estimated (analytical) delay of your 16-bit CLA adder. Specify the delay calculation of cla_delay clearly from a past lab.
3. In a **separate** process from the one that generates the random inputs, demonstrate the correctness of the Booth's multiplier using an automated checking scheme that converts the standard logic vectors of the two inputs A, B (16 bits each) and the output P (32 bits) into integers $Aint, Bint, Pint$ respectively, after an appropriate delay for each input-pair that allows the final output to appear, and checks if P is the correct product using **assert** and **report** statements (the assert statement should verify if $Pint = Aint * Bint$; note the the inbuilt $*$ operator in VHDL is being used here to verify correctness of your designed Booth's multiplier). **200**
4. Determine from the graphical interface, the worst-case *simulation* delay, over some 10 random inputs, of the product P . What is the worst-case simulation delay in cc's? For each of the 10 inputs, also determine in terms of cc's (assuming the adder takes 3cc's, etc. as specified above) determine how many cc's a regular A&S 2's complement multiplier will take (you need to design the FSM for this multiplier for this purpose—this should not take too much time as you can use a modified and simplified version of the Booth's multiplier FSM for this purpose) and note the corresponding simulation delays (in cc's) of the Booth multiplier in a table. Determine the % average delay improvement of the Booth multiplier over the regular A&S multiplier computed as $[(\text{Total regular A\&S delay for 10 inputs}) - (\text{Total Booth delay for 10 inputs})] \times 100 / (\text{Total regular A\&S delay for 16 inputs})$ **300**