University of Pennsylvania
Department of Electrical and System Engineering
Computer Organization

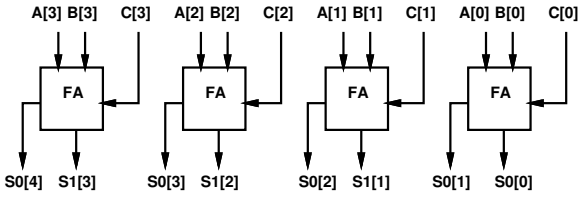
ESE534, Spring 2012 — Assignment 2: Space-Time Multiply — Monday, Jan. 23

Due: Monday, January 30, 12:00PM

We saw in lecture how to build various adders. In this problem, We're asking you to review or develop various techniques for building multipliers.

- Give latency and area in terms of the operand bitwidth, w. (we'll take asymptotic analysis, or you can use symbolic constants in terms of primitive gates such as $T_{fulladderslice}$, T_{and2}, A_{and2}, $A_{registerbit}$, A_{mux2})
- When asked to draw an implementation, show the $w = 4$ case (except for 3(e) where we ask you to show $w = 8$). You may use hierarchical schematics.

1. Consider a spatial multiplier built out of simple, ripple-carry adders.
   (a) Show a 4×4 multiplier.
   (b) What is the area and latency for this multiplier? (function of w)
2. Let's consider an alternate technique that uses the same full adder bitslice as in the previous ripple-carry adder design, but which wires up the carries differently. [This technique is known as **delayed addition**.]



Here, A and B will be your normal two inputs to the adder. $S0$ and $S1$ together store the sum.

   (a) What is the latency of a single w-bit delayed addition?
   (b) How can the C input to the delayed adder be used?
   (c) Use these delayed-addition adders to build a spatial multipler. The two input-operands to the multiplier are in standard form. Output values are represented as two numbers (*i.e.* S0, S1 form shown above). Show the resulting, spatial multiplier which starts with numbers in standard form, but uses these delayed adders internally. (show $w = 4$ case.)

(d) What do you need to do to the multiplier output to convert the result back into normal form?
- Remember, S0 and S1 jointly encode the final result. The normal form output should be a single binary-encoded word.
- We would like this final conversion to have minimum latency. Be specific about how we implement the operation to minimize the latency it contributes.

(e) What is the final area and latency of this multiplier? (function of w)

3. Continuing to the use full-adder bitslice used above, wire them up as an associative reduce tree to compute the result of the multiplication from all the bit-wise partial-products ($a_i \wedge b_j$).

(a) How many partial-product bits do you start with? (function of w)
(b) How many bits are outputs from the first stage of full-adder bitslices?
(c) What reduction do you get with a single stage of full adders? (*e.g.* from part (a) to (b))
(d) Continuing to use stages of full-adder bitslices to reduce the number of bits, how deep is the full reduce tree? (function of w)
(e) Show the resulting multipler. ($w = 8$ case)
(f) What is the final area and latency of this multiplier? (function of w)

4. Using a datapath with only w full-adder bitslices, w AND's, multiplexers, and registers, develop an FSMD to compute the product of two w-bit numbers in the least amount of total time. You may want to continue to use the basic delayed addition multiplication strategy from 3.

(a) Identify the state registers you will need (what do they hold? how big will they be as a function of w?).
(b) Show the resulting multipler:
- datapath (show $w = 4$ case)
- how the datapath and FSM interact (control signals between them)
- state machine diagram for the FSM

(c) What is the latency of each cycle? (this probably demands you think about the gate-level implementation of the FSM and the datapath; can you guarantee this is **not** a function of w for any w? (assuming delay is in the gates not the wires))
(d) How many cycles does it take to complete the multiply? (function of w)
(e) What is the final area and latency of this multiplier? (function of w)

5. Fillin the following table from your area/latency answers to the problems above (all functions of w):

Design	**Area**	**Latency**
P1: Ripple-Carry Based		
P2: Delayed-Addition Based		
P3: Associative Reduce Delayed-Addition		
P4: Delayed-Addition FSMD		