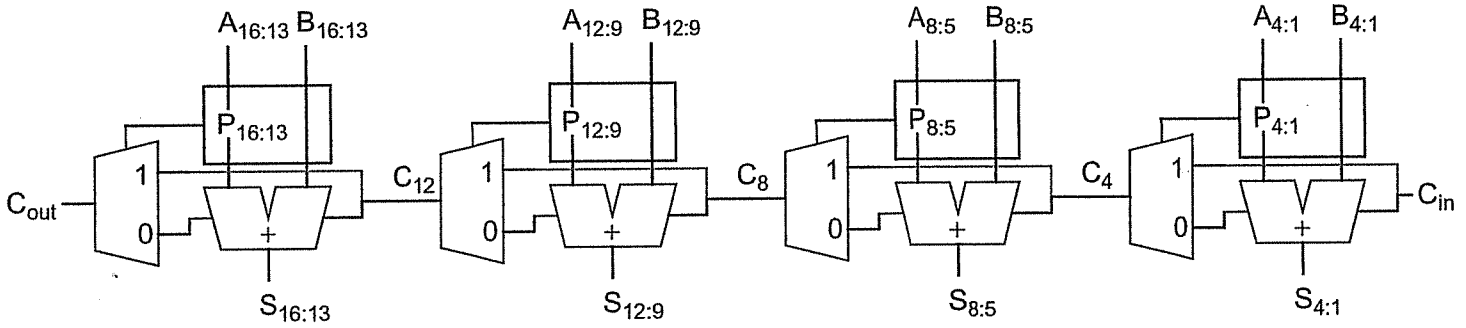
12.4 Other High-Speed Adders

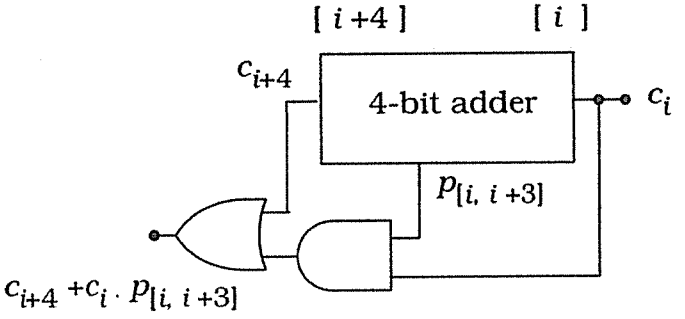
12.4.1 Carry-Skip Circuits

Carry-skip adders speed wide adders by allowing the propagation of a carry-bit around an entire portion of the adder if the group propagate signal would just propagate the carry through that section:



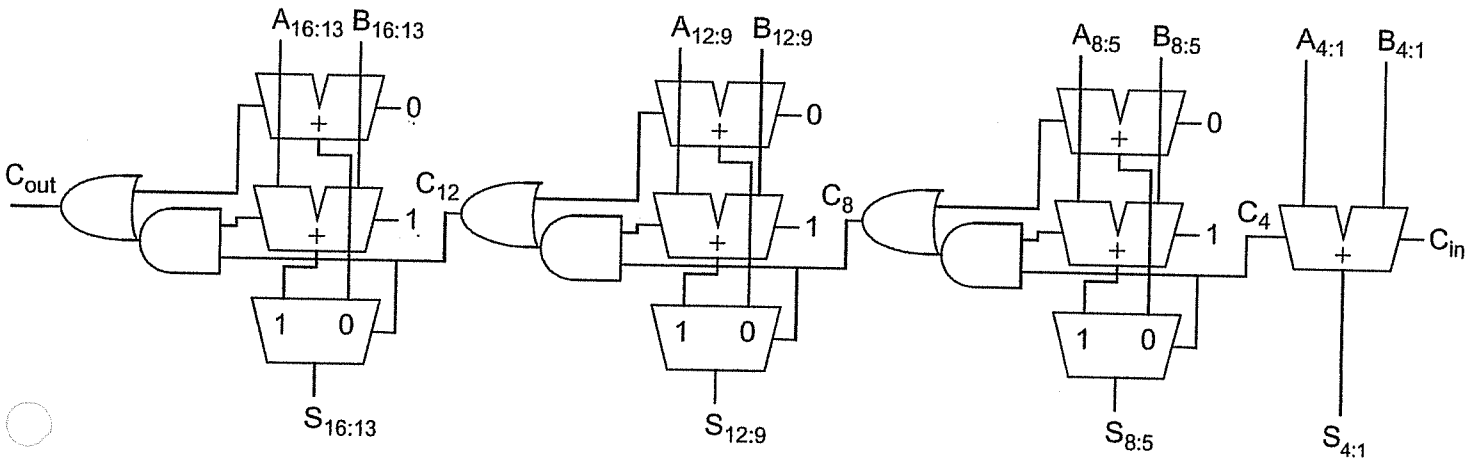
The hardware cost is the same as a simple carry-ripple adder plus k multiplexers and k n-input AND gates. It is attractive when ripple-carry adders are too slow and hardware costs must still be kept low.

Note: your book shows an implementation using an AND-OR logic combination rather than a MUX:



12.4.2 Carry-Select Adders

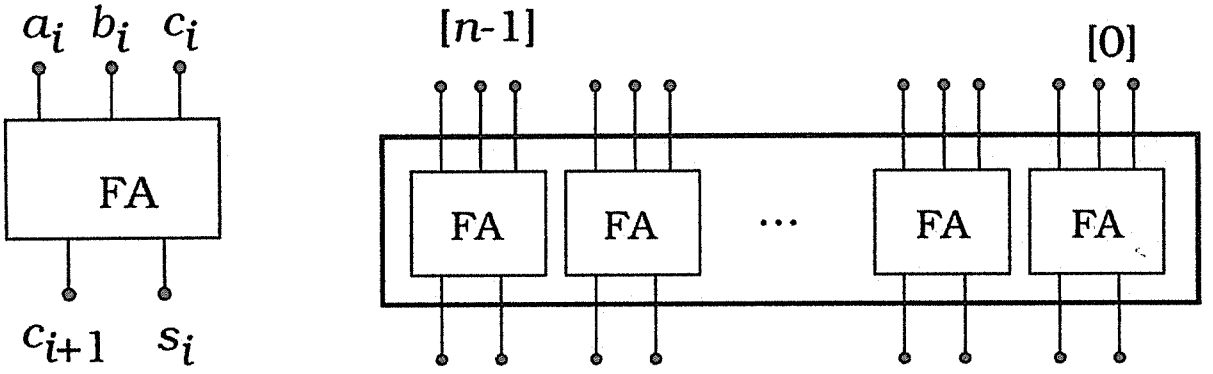
Carry-select adders note that there are only two possible C_{in} inputs to an adder or block of adders, 1 and 0, and so computes the C_{out} value for both situations, then multiplexes the result based on the actual C_{in} value:

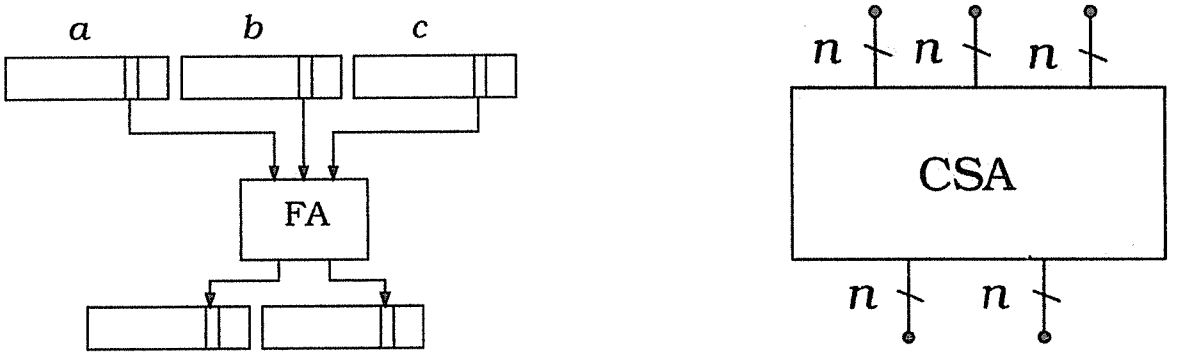


A more compact design that factors out some of the common logic in the two adders is sometimes called a **carry-increment adder**, and has about the same critical delay as the carry-select adder, but the area is smaller.

12.4.3 Carry-Save Adders

Carry-Save adders treat the operation of a full adder as a 3-input, 2-output device, and "saves" the C_{out} bits as a n-bit carry-out word, rather than immediately summing it with the adjacent adder:

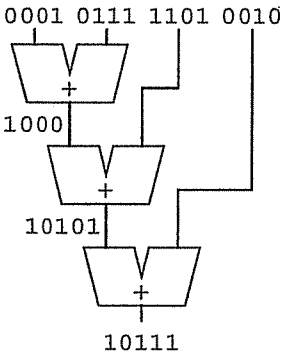




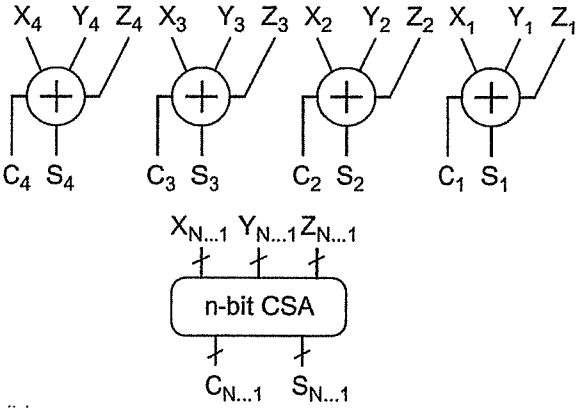
This particular implementation is especially useful for summing more than one number, and is therefore very useful in applications like multiplication.

The final sum of several numbers requires CPAs (carry-propagate adders) at the final step to sum the results.

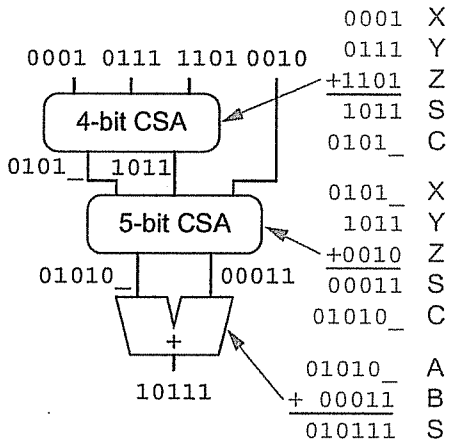
Consider the addition of four binary numbers. The most intuitive method would be to sum them individually in three stages of CPAs:



The carry-propagate paths through all of the CPAs give a slow critical path.

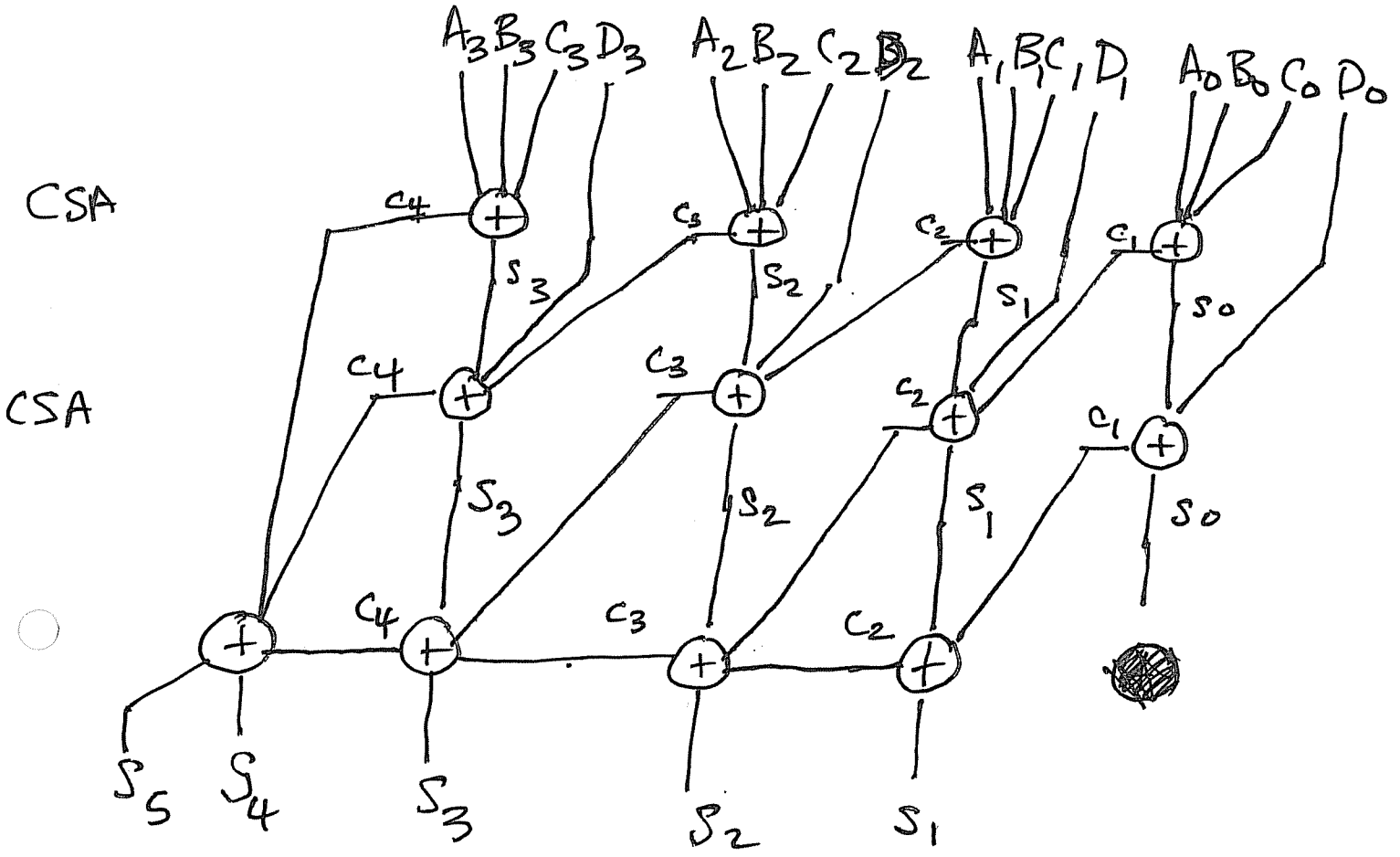


A carry-save adder (CSA) on the other hand, does not sum the carries, and so the critical path is for the sum computation, since all inputs process in parallel.



This can be applied to the four input addition with 2 levels of CSAs, and a final sum computed by a CPA

Example: Draw the connections for 4-input, 4-bit addition using CSA architecture



12.4 – Other High Speed Adders
(not in your book)

Tree Adders – for wide adders (>16 bits) the delay of passing a carry through the look-ahead stages dominates the adder worst path delay. This delay can be reduced by "looking ahead across the look-ahead blocks." This leads to a multilevel tree of look-ahead structures. These types of multi-level lookahead adders are therefore used in modern architectures with wider bitwidths (though detailed study is outside the scope of this course)

Summary of Adders:

Your Final Design Project requires a Manchester Carry Look Ahead architecture. When the adder type is not specified, how does a VLSI designer choose an architecture?
When ripple-carry adders meet timing constraints they are the best choice because they use the least energy and are easy to build. Carry-skip and carry-increment architectures work well for 8-16 bit lengths. For 32 or especially 64 bits, tree adders are much faster.

12.5 Multipliers

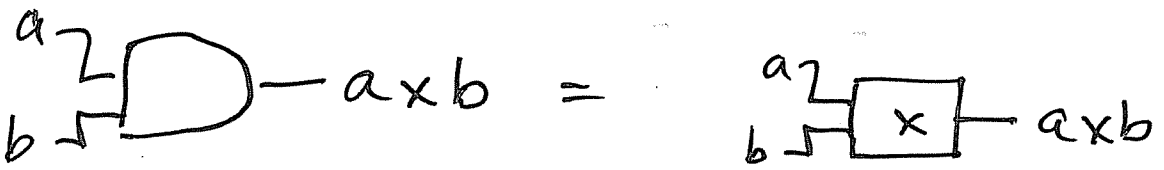
Binary multiplication:

$$0 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 0 = 0$$
$$1 \times 1 = 1$$

a	b	axb
0	0	0
0	1	0
1	0	0
1	1	1

} AND

So the bit-level multiplication is trivial:



Consider the complication of multiplying :

$$a_3a_2a_1a_0 \times b_3b_2b_1b_0 = p_7p_6p_5p_4p_3p_2p_1p_0$$

				a_3	a_2	a_1	a_0	multiplicand
				$\times$ b_3	b_2	b_1	b_0	multiplier
				$a_3\,b_0$	$a_2\,b_0$	$a_1\,b_0$	$a_0\,b_0$	
			+ $a_3\,b_1$	$a_2\,b_1$	$a_1\,b_1$	$a_0\,b_1$		
		+ $a_3\,b_2$	$a_2\,b_2$	$a_1\,b_2$	$a_0\,b_2$			
	+ $a_3\,b_3$	$a_2\,b_3$	$a_1\,b_3$	$a_0\,b_3$				
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	product

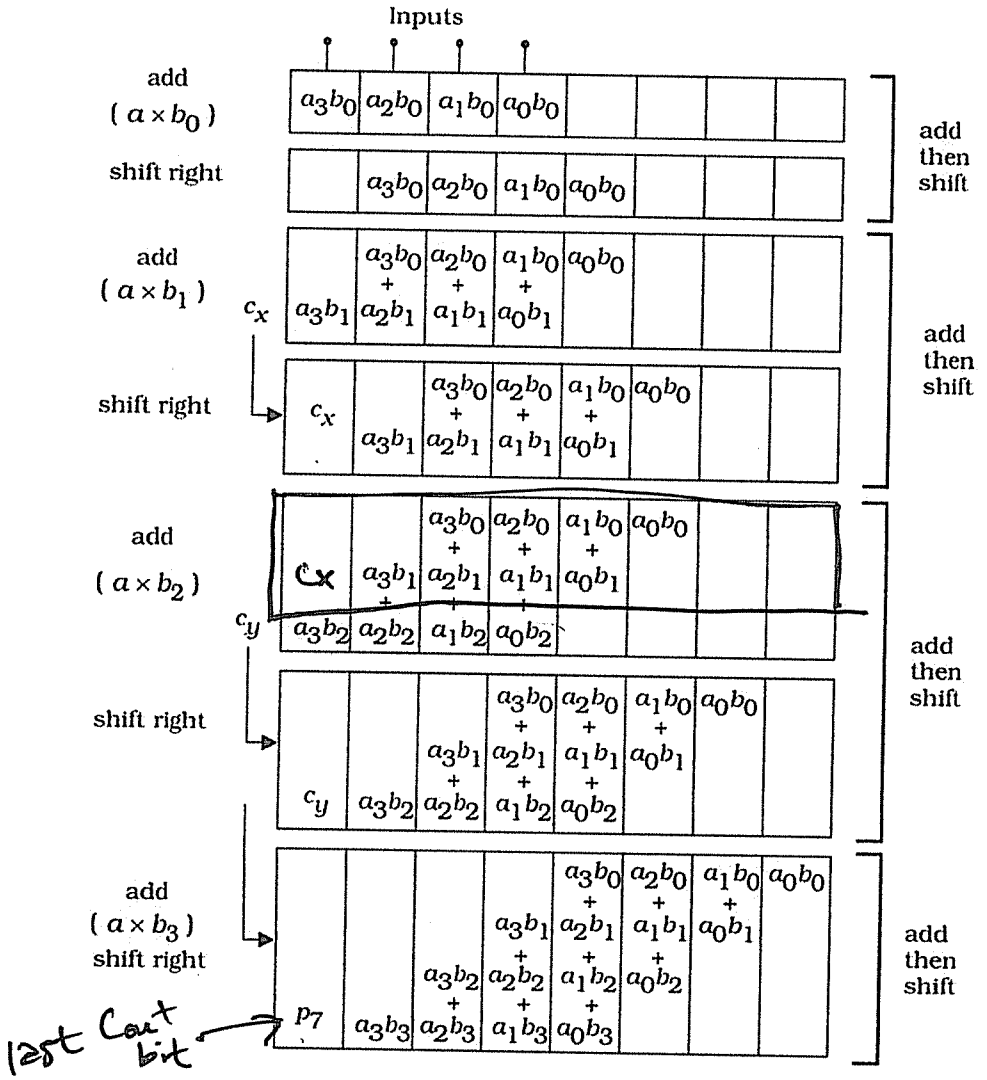
A special case of binary multiplication is multiplication by a factor 2^m, for example multiplying by $b = 00000010 = 2^1$ is equivalent to shifting the multiplicand a by 2 to the left, *i.e.* a LSL 1. In general multiplying by 2^m is equivalent to a LSL m.

Division by 2^k can be accomplished with an arithmetic shift right by k, however this complicates the masking circuitry in the shifter, since the mask must preserve the sign of the shifted 2'sC number for arithmetic right shifts.

One implementation of a multiplier treats the multiplication as each bit of b_n multiplying the entire word $a_3a_2a_1a_0$, with the bits shifted by a factor of n, *i.e.* multiplied by 2^n:

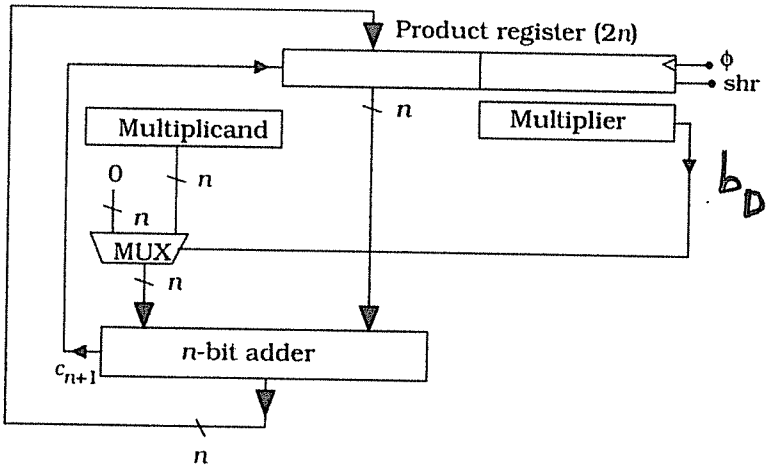
				a_3	a_2	a_1	a_0		multiplicand
			$\times$	b_3	b_2	b_1	b_0		multiplier
				$(a_3$	a_2	a_1	$a_0)\times b_0$		$(a\times b_0)\,2^0$
			$(a_3$	a_2	a_1	$a_0)\times b_1$			$(a\times b_1)\,2^1$
		$(a_3$	a_2	a_1	$a_0)\times b_2$				$(a\times b_2)\,2^2$
+	$(a_3$	a_2	a_1	$a_0)\times b_3$					$(a\times b_3)\,2^3$
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0		product

This method uses a **product register** to temporarily hold the product, building it one b bit at a time with subsequent adds and shifts.



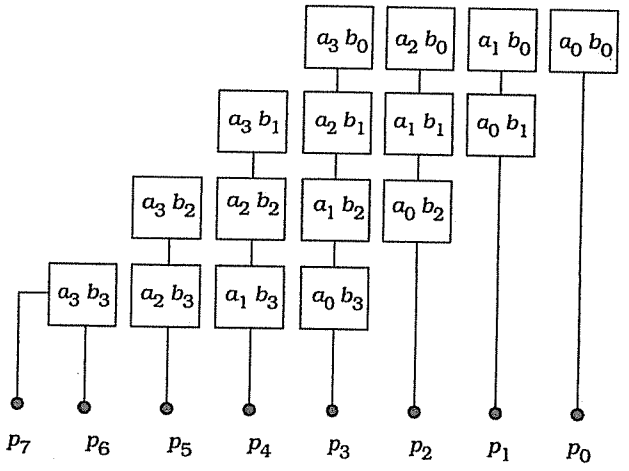
NOTE: the carry-out bits are tracked by shifting them into the left register as the appended bit.

This design can be used in the register-based network:

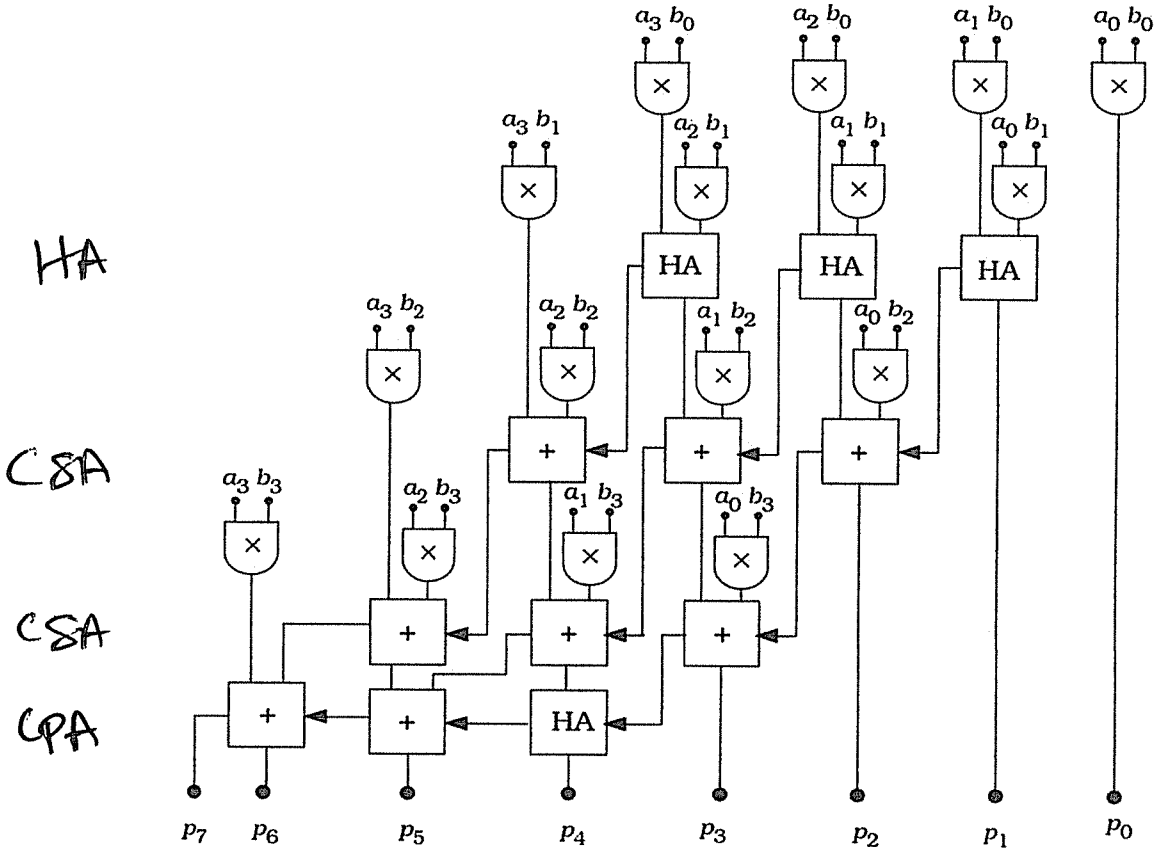


12.5.1 Array Multipliers

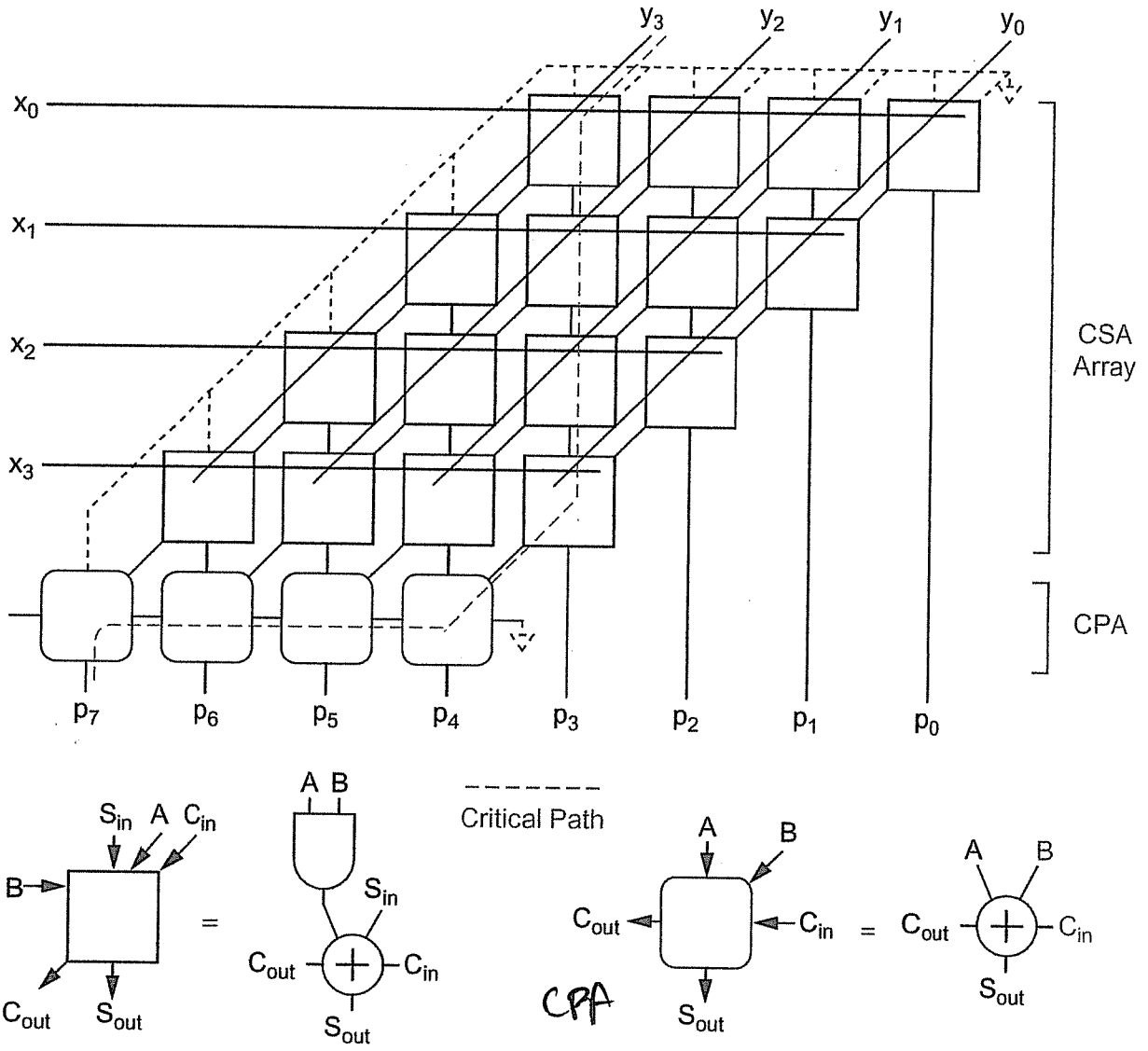
Array multipliers accept the multiplier and multiplicand and produces each bit of the product in a parallel manner:



This can be efficiently implemented using carry-save adders, (and half-adders where only 2 inputs are needed), with CPA architecture to sum the final row:



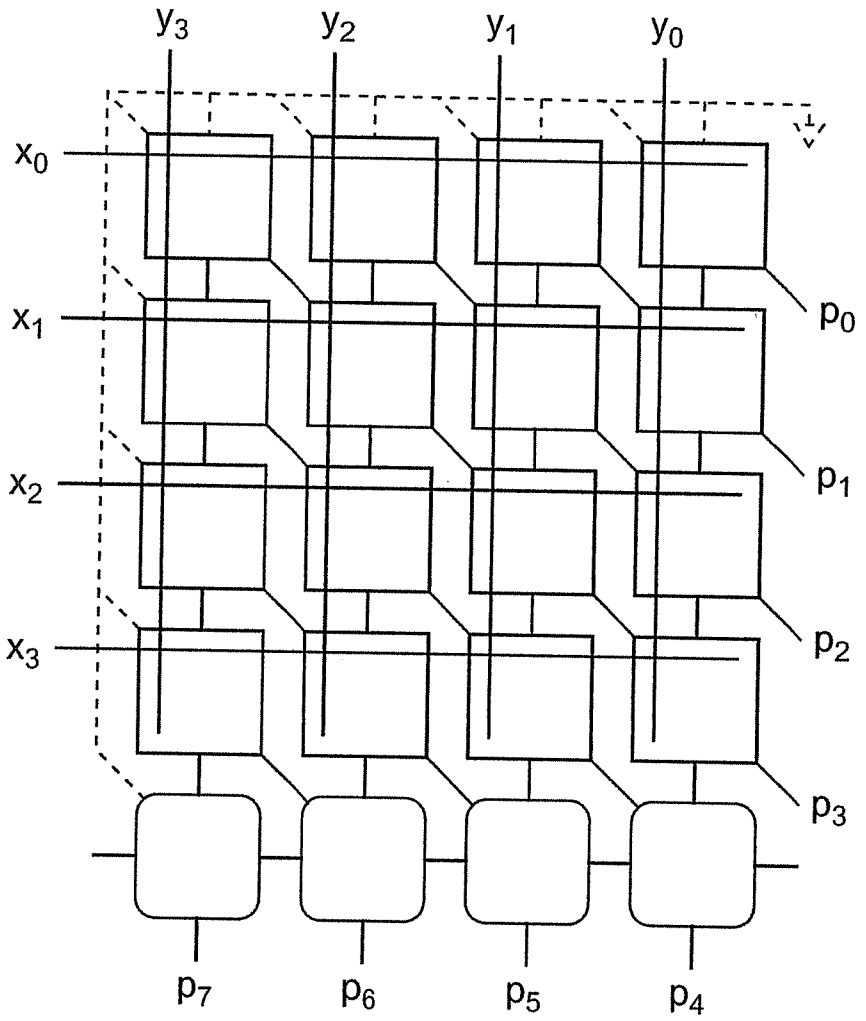
This can be made into a regular array:



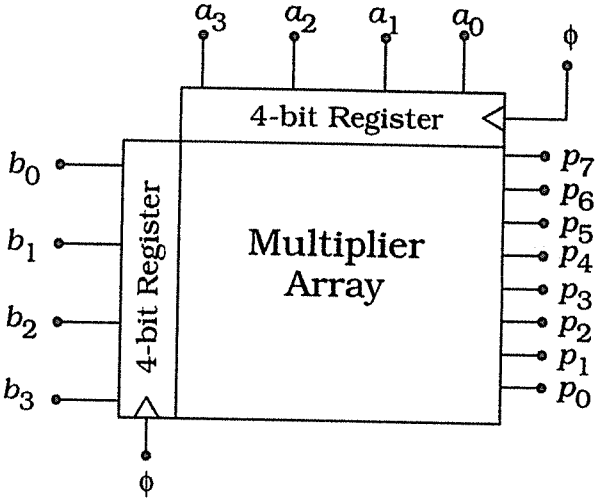
CSA

The critical path follows 4 sum outputs in the CSA array, and the entire carry-propagate chain in the CPAs. A parallelogram isn't a great shape for floorplanning a chip, so a regular square array would be preferable.

The layout can be "squashed" into a rectangular array:



Input registers (and/or an output register) may be added to synchronize the dataflow:



Reading: Finish Ch 12, Finish Ch 13 by next Wed.

Check-In: Draw the layout for an SRAM cell