EECS 151 Disc 11

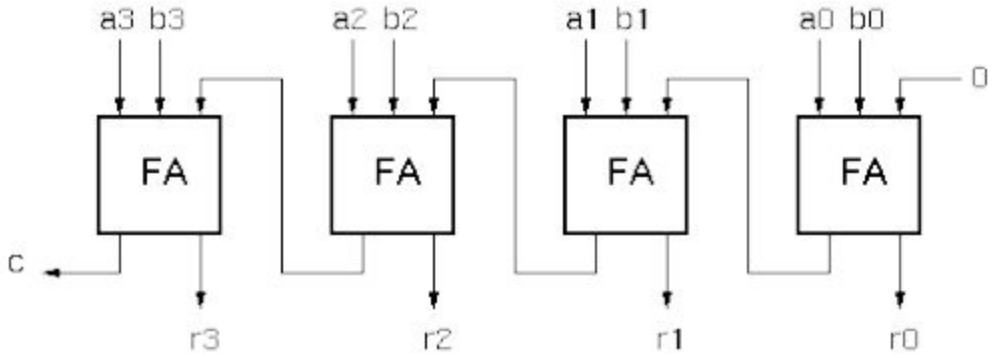
Rahul Kumar (session 1)
Yukio Miyasaka (session 2)

Berkeley
UNIVERSITY OF CALIFORNIA

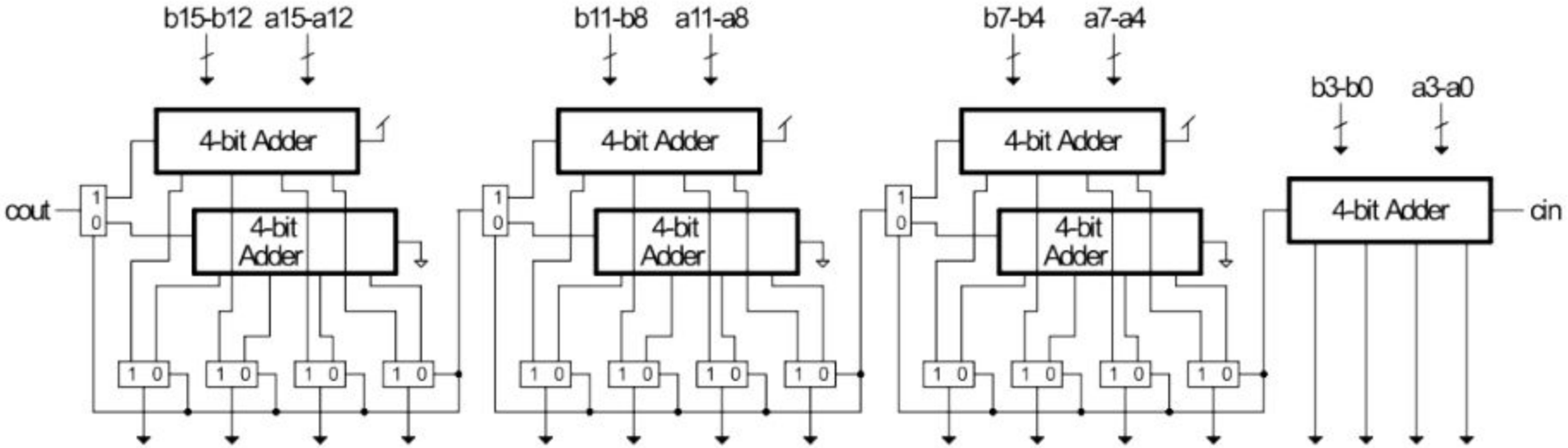
Contents

- Adder
- Multiplier

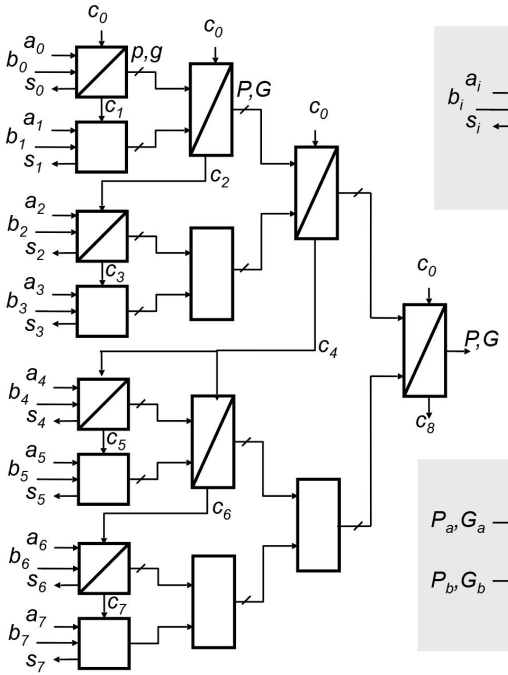
Ripple Carry Adder

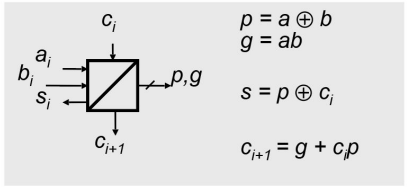


Carry Select Adder

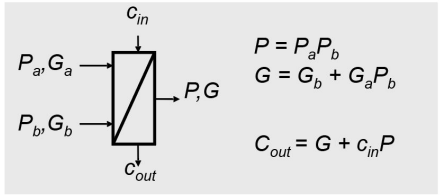


Carry Look-ahead Adder

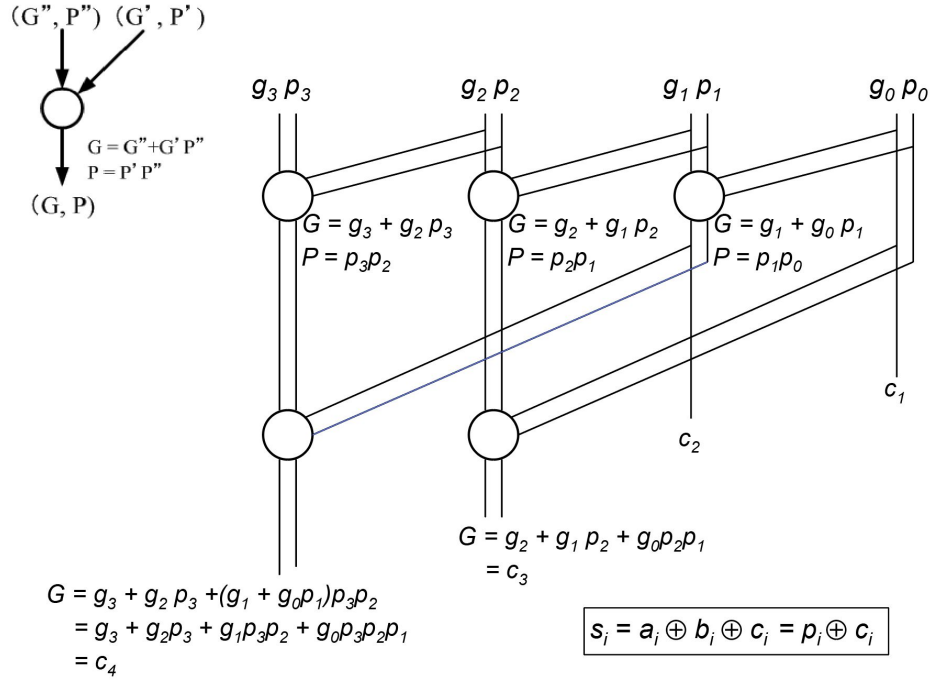




8-bit Carry Look-ahead Adder

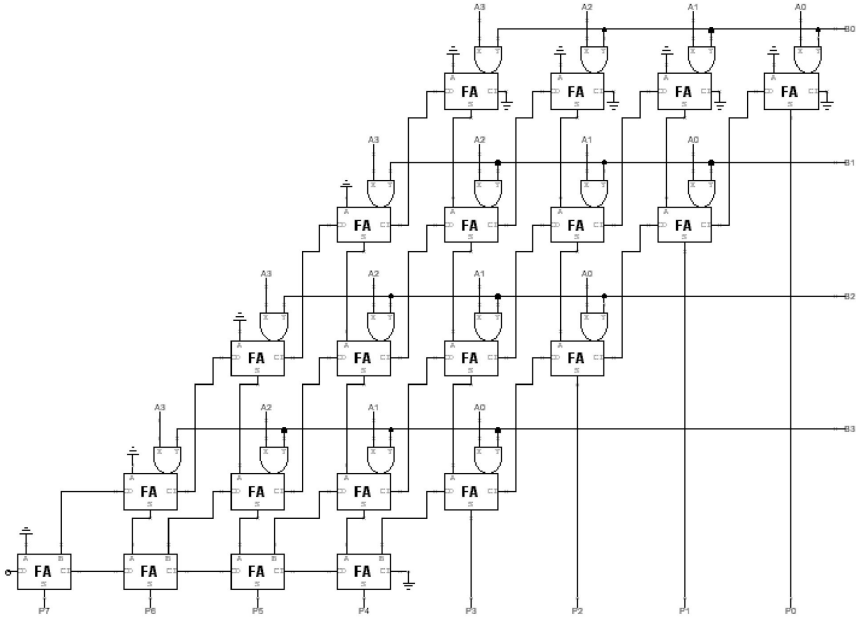


Parallel Prefix Adder

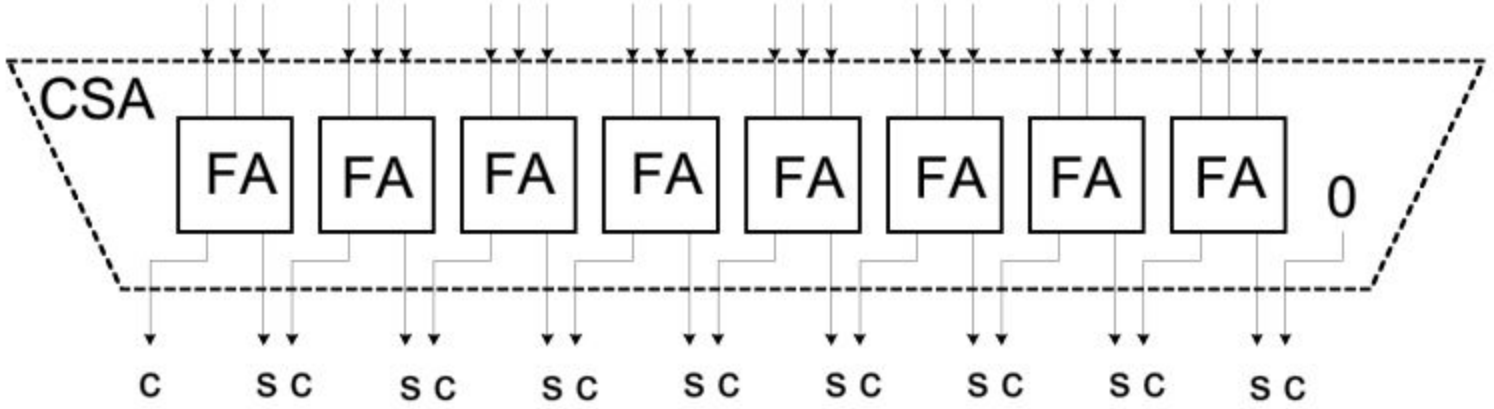


With carry-in,
g0 = cin, p0 = 0,
g1 = a0b0, p1 = a0^b0,
…

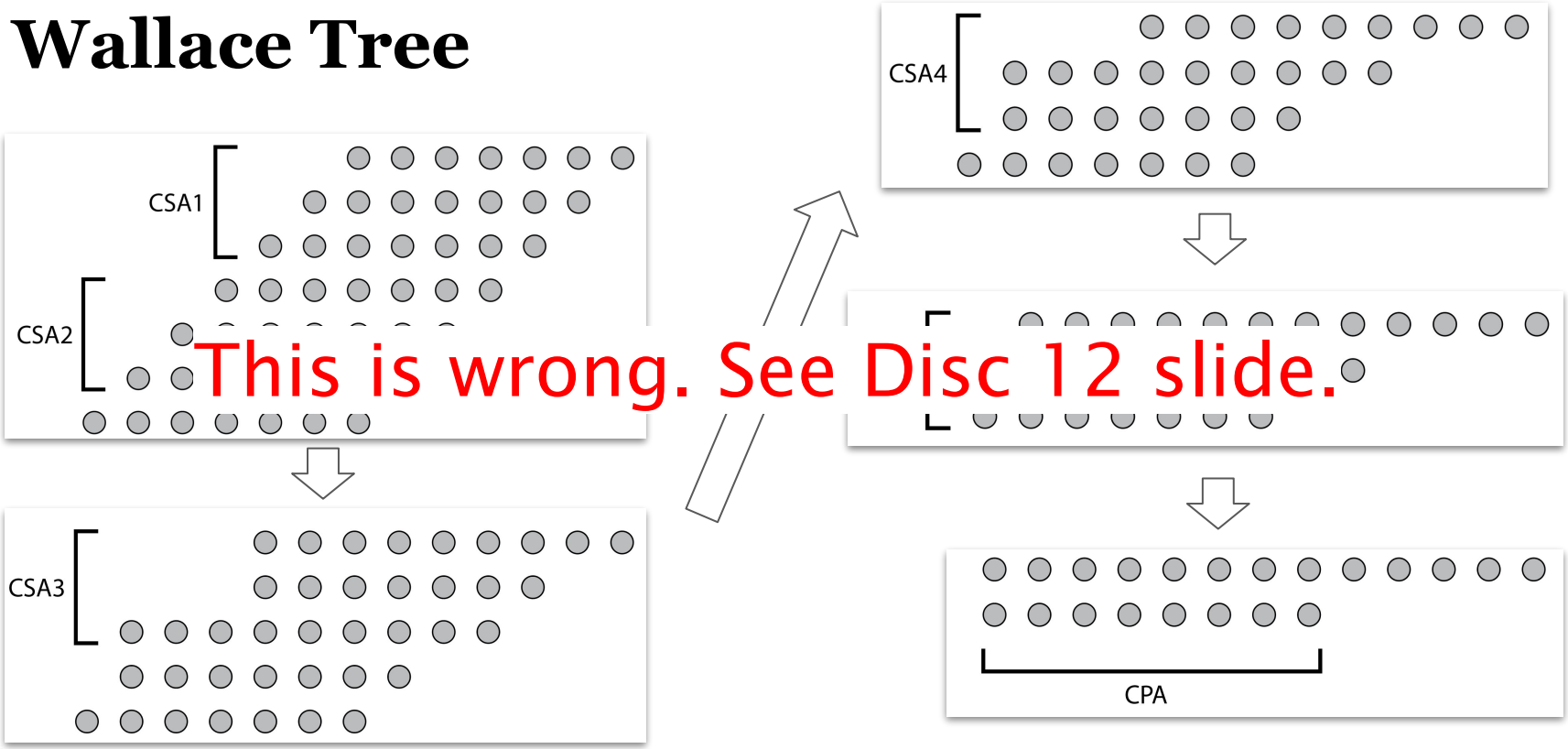
Array Multiplier



Carry Save Adder

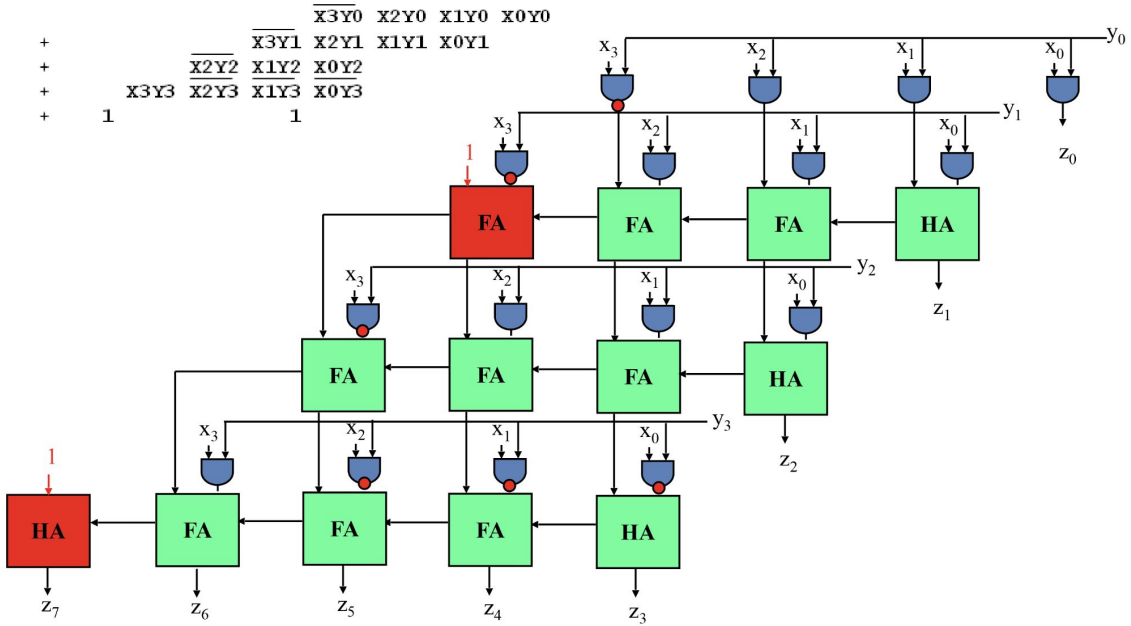


Wallace Tree



This is wrong. See Disc 12 slide.

Signed Multiplication



Booth Multiplier (Radix 4)

- Reduce #partial-products by looking at 2 bits (actually 3) at a time.
- We don't want to add A*3, so sub A and then add 4*A in the next partial product.
- We also need to sub 2*A instead of add 2*A to cancel the side-effect.
- Magically, Booth multiplier works for signed multiplication just by sign-extending the multiplier (B).
- Can use the optimization in the previous page for sign-extended partial products.

B_{K+1}	B_K	B_{K-1}	action
0	0	0	add 0
0	0	1	add A
0	1	0	add A
0	1	1	add 2*A
1	0	0	sub 2*A
1	0	1	sub A
1	1	0	sub A
1	1	1	add 0

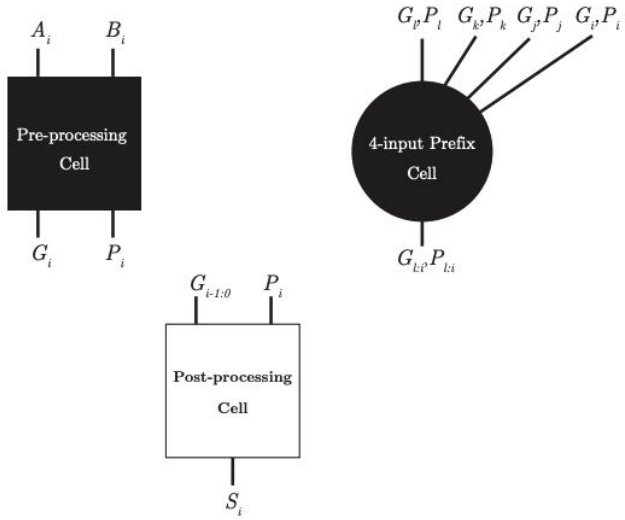
Practice Problems

(Taken from HW in previous semesters)

Radix-4 Kogge-Stone Adder

Recall from the lecture that Kogge–Stone adder is a parallel prefix form carry look-ahead adder (CLA). In this problem, we'd like to design and analyze a 16-bit radix-4 Kogge-Stone adder.

(a) Write down the Boolean functions of the gate implementation of following building blocks for the radix-4 Kogge-Stone adder. You may use AND, OR and XOR only. Also, how should you handle cases where certain 4-input prefix cells have fewer than 4 pairs of input?



Radix-4 Kogge-Stone Adder

Pre-processing: $G_i = A_i B_i, \quad P_i = A_i \oplus B_i$

4-input prefix: $G_{l:i} = G_l + (P_l(G_k + (P_k(G_j + P_j G_i))))$
$P_{l:i} = P_l P_k P_j P_i$

Post-processing: $S_i = P_i \oplus G_{i-1:0}$

Radix-4 Kogge-Stone Adder

(b) Which of the following are true for radix-4 Kogge-Stone adder?

____ Compared to Radix-2 adders, Radix-4 adders reduce the depth of the tree by a factor of 4 (excluding the input and output stages).

____ An N-bit Radix-4 adders has $\mathcal{O}(\sim log_4(N))$ in time.

____ It is possible to implement a 32-bit Kogge-Stone adder using only the building blocks in part (a).

Radix-4 Kogge-Stone Adder

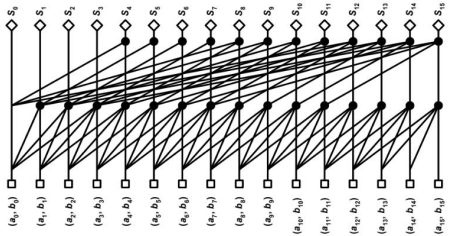
F. Radix-4 adders reduce the depth by a factor of 2.
T. But do keep in mind that each stage takes longer time.
T. You might need to leave some intermediate P,G outputs unconnected, though.

Radix-4 Kogge-Stone Adder

(c) Suppose given the following propagation delays:

$$t_{AND} = 5ps, \quad t_{OR} = 4ps, \quad t_{XOR} = 7ps$$

Derive the critical path of the 16-bit Kogge-Stone adder based on your implementation in part (a), ignoring the delays in routing. (Hint: If you are not sure about the topology, take a look at the slides in Lecture 19.)



16-bit radix-4 Kogge-Stone Tree

Radix-4 Kogge-Stone Adder

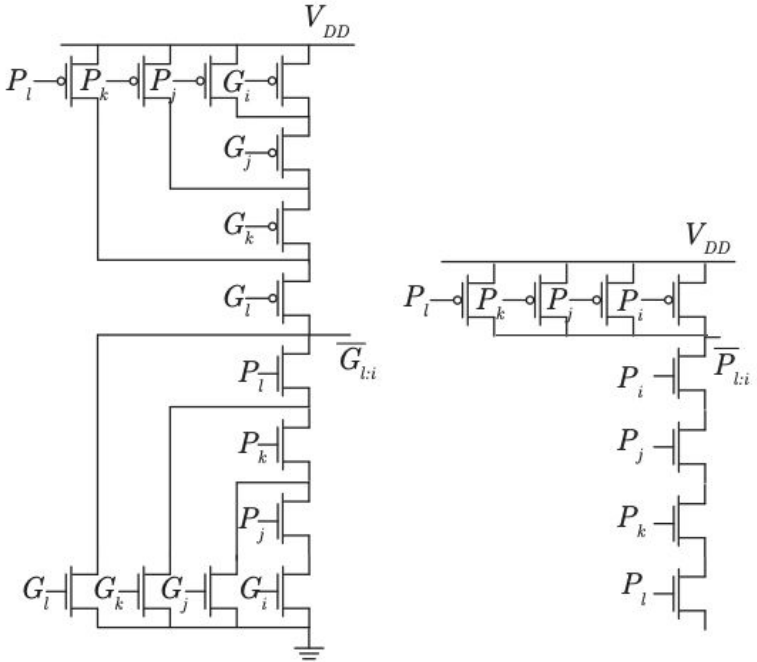
The longest path will go through: 1 pre-processing cell + 2 four-input prefix cells + 1 post-processing cell. So the total delay is:

$$
\begin{aligned}
t_{critical} &= max\{5ps, 7ps\} + 2 \times (5ps + 4ps + 5ps + 4ps + 5ps + 4ps) + 7ps \\
&= 7ps + 2 \times 27ps + 7ps \\
&= 68ps
\end{aligned}
$$

Radix-4 Kogge-Stone Adder

(e) **(251A Only)** In reality, those prefix cells will not be built using the basic 2-input AND, OR, XOR gates. Instead, they will be built as a big CMOS gate (and inverters). Draw the schematic of the 4-input prefix cell for $\bar{G}_{l:i}$ and $\bar{P}_{l:i}$ respectively with the minimum number of transistors.

Radix-4 Kogge-Stone Adder



Booth Multiplication

Refer to the following table of the behavior of Booth recoding.

B_{K+1}	B_K	B_{K-1}	Action
0	0	0	add 0
0	0	1	add A
0	1	0	add A
0	1	1	add 2A
1	0	0	sub 2A
1	0	1	sub A
1	1	0	sub A
1	1	1	add 0

Answer should be in the format of:

Suppose A=1010 B=1001
(B[1:-1]=010): add A,
(B[3: 1]=100): sub 2A,
...
result = A - (A<<2) + ...
= 0000(1010) - 00(1010)00 + ...
= ...

Write down the sequence of operation and the final result given the following unsigned two input numbers:

01100111 (A)
× 10110010 (B)

Booth Multiplication

(B[1:-1]=100): sub 2A,
(B[3: 1]=001): add A,
(B[5: 3]=110): sub A,
(B[7: 5]=101): sub A,
(B[9: 7]=001): add A

result = -(2A) + (A<<2) - (A<<4) - (A<<6) + (A<<8)
= - 0000000(01100111)0 + 000000(01100111)00
- 0000(01100111)0000 - 00(01100111)000000
+ (01100111)00000000
= 0100 0111 1001 1110 (18334 in decimal)