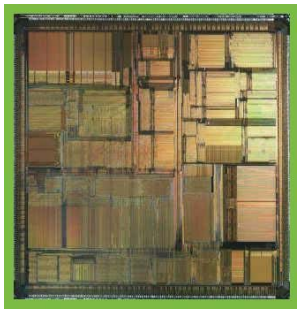
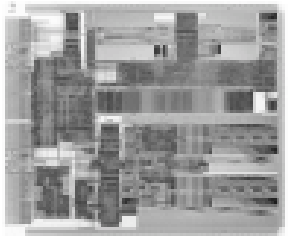
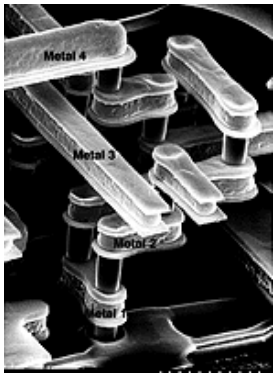




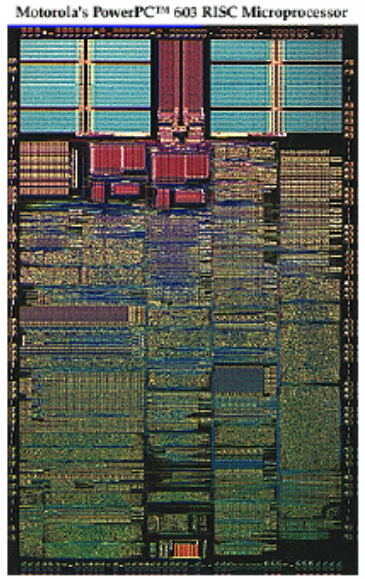
VLSI Arithmetic

Adders & Multipliers



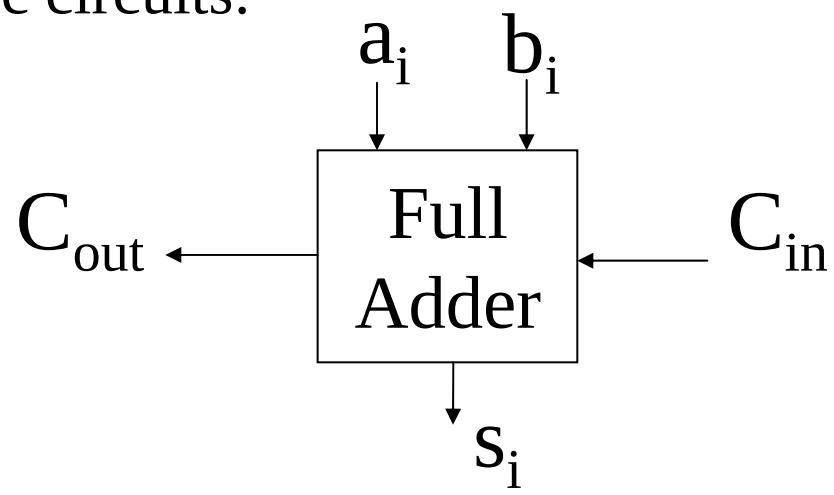
Prof. Vojin G. Oklobdzija
University of California

http://www.ece.ucdavis.edu/acsel



Addition of Binary Numbers

Full Adder. The full adder is the fundamental building block of most arithmetic circuits:

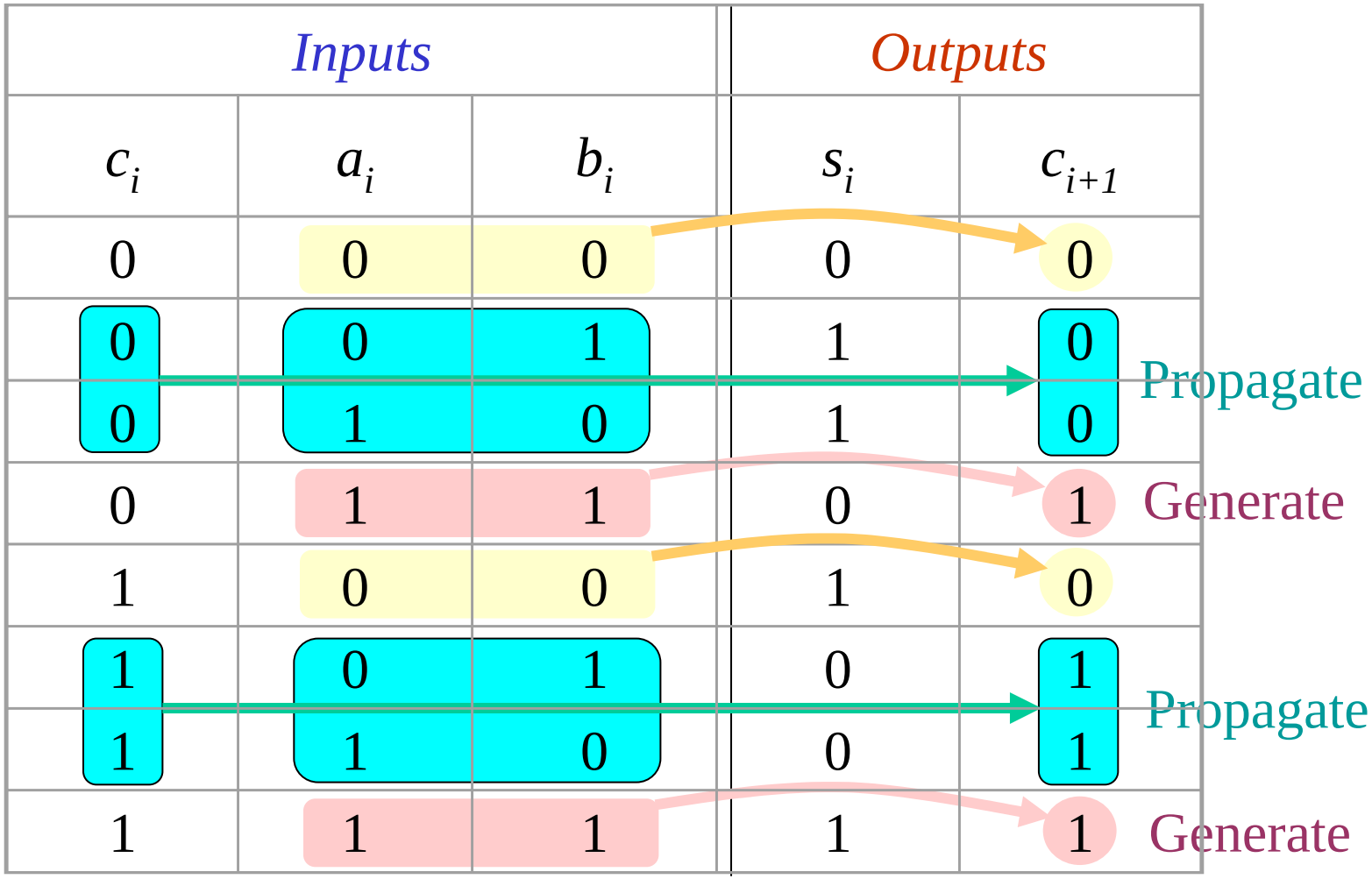


The sum and carry outputs are described as:

$$s_i = a_i\overline{b}_i\overline{c}_i + \overline{a}_i b_i\overline{c}_i + \overline{a}_i\overline{b}_i c_i + a_i b_i c_i$$

$$c_{i+1} = \overline{a}_i b_i c_i + a_i\overline{b}_i c_i + a_i b_i\overline{c}_i + a_i b_i c_i = a_i b_i + a_i c_i + b_i c_i$$

Addition of Binary Numbers



Full-Adder Implementation

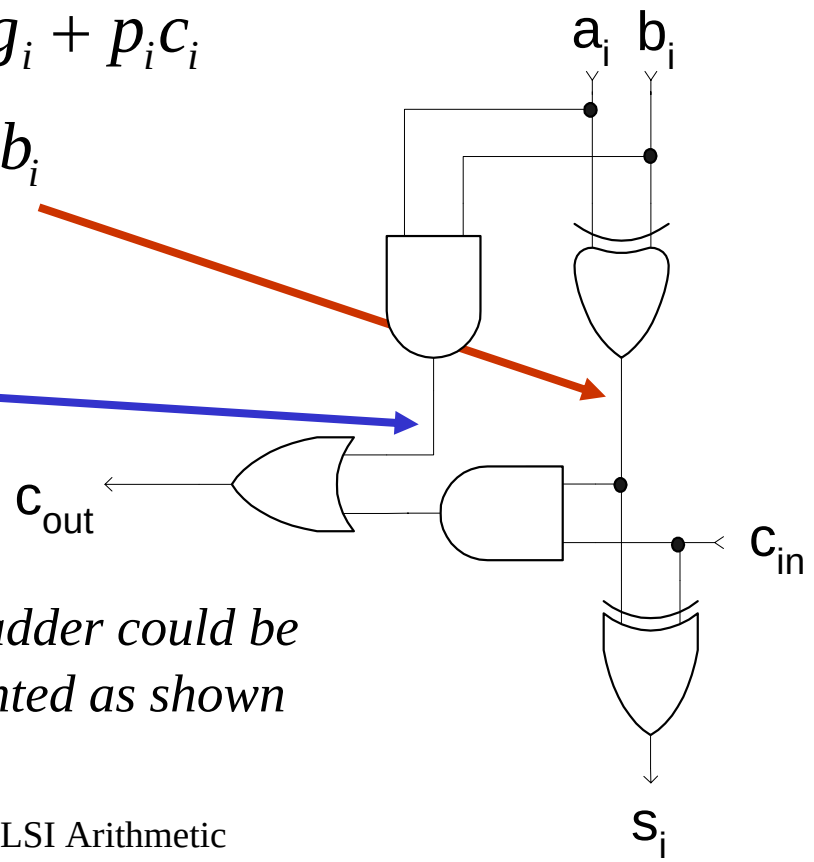
Full Adder operations is defined by equations:

$$s_i = a_i\bar{b}_i\bar{c}_i + \bar{a}_i b_i\bar{c}_i + \bar{a}_i\bar{b}_i c_i + a_i b_i c_i = a_i \oplus b_i \oplus c_i = p_i \oplus c_i$$

$$c_{i+1} = \bar{a}_i b_i c_i + a_i\bar{b}_i c_i + a_i b_i = g_i + p_i c_i$$

Carry-Propagate: $p_i = a_i \oplus b_i$
and Carry-Generate g_i

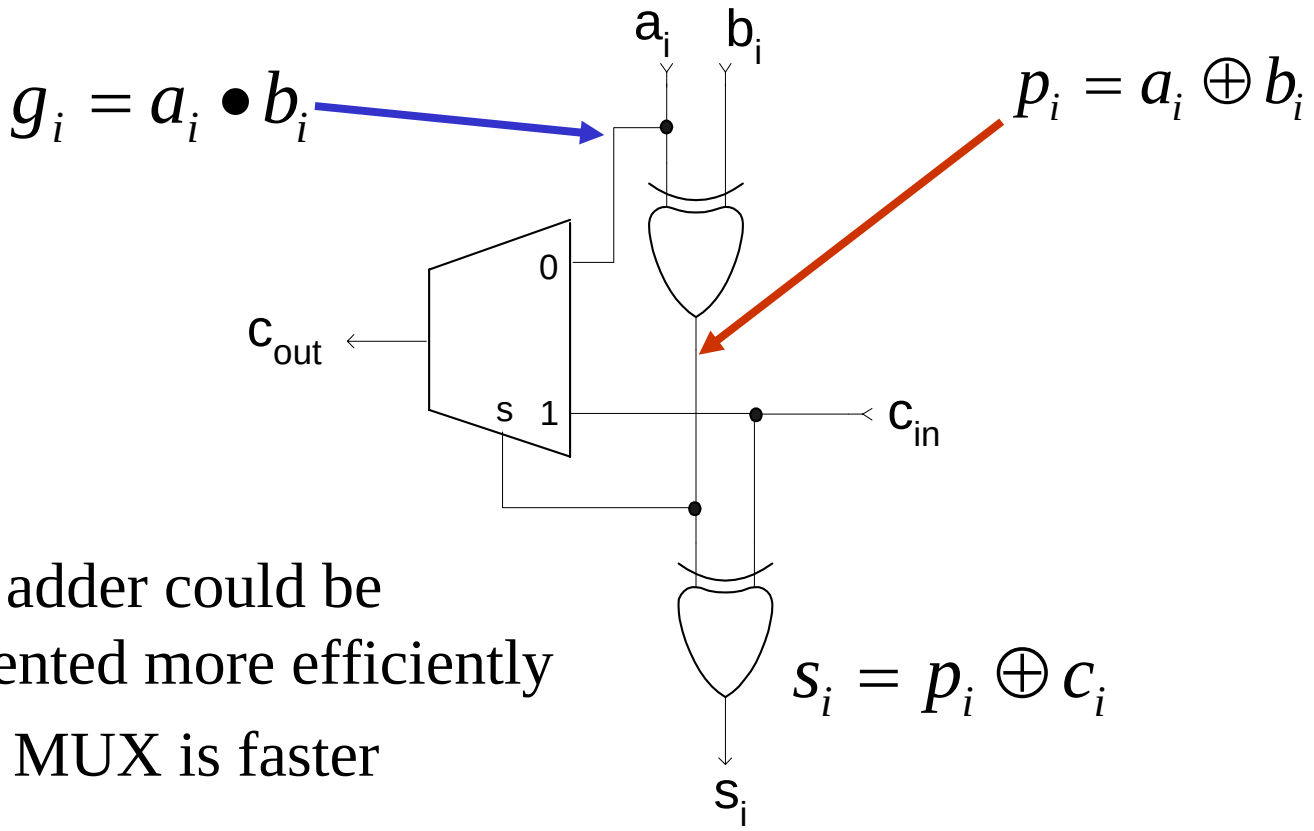
$$g_i = a_i \bullet b_i$$



One-bit adder could be implemented as shown

High-Speed Addition

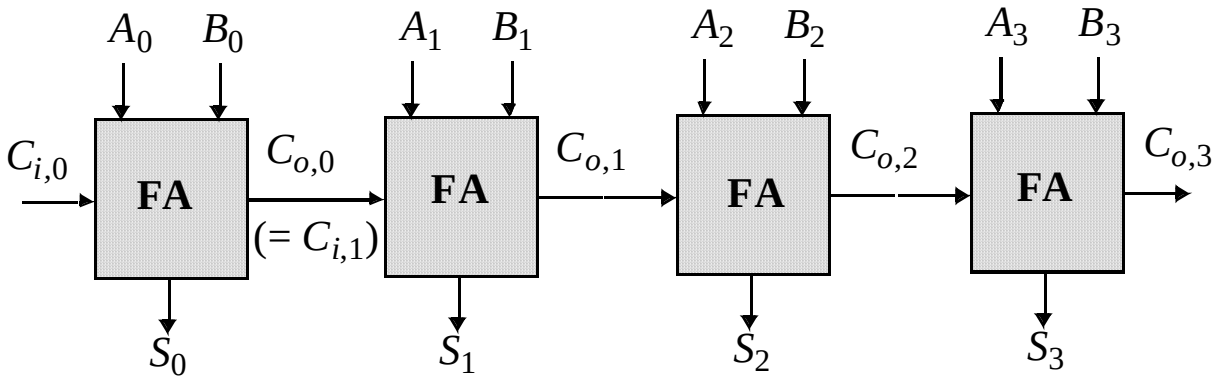
$$c_{i+1} = g_i + p_i c_i$$



One-bit adder could be implemented more efficiently because MUX is faster

The Ripple-Carry Adder

The Ripple-Carry Adder



Worst case delay linear with the number of bits

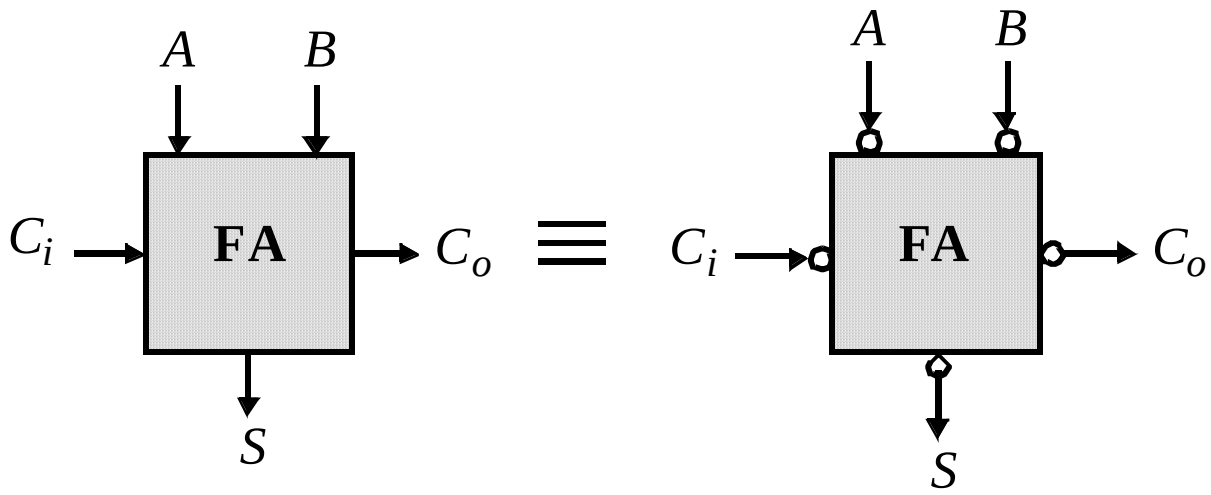
$t_d = O(N)$

$$\mathbf{t_{adder} \approx (N-1)t_{carry} + t_{sum}}$$

Goal: Make the fastest possible carry path circuit

From Rabaey

Inversion Property

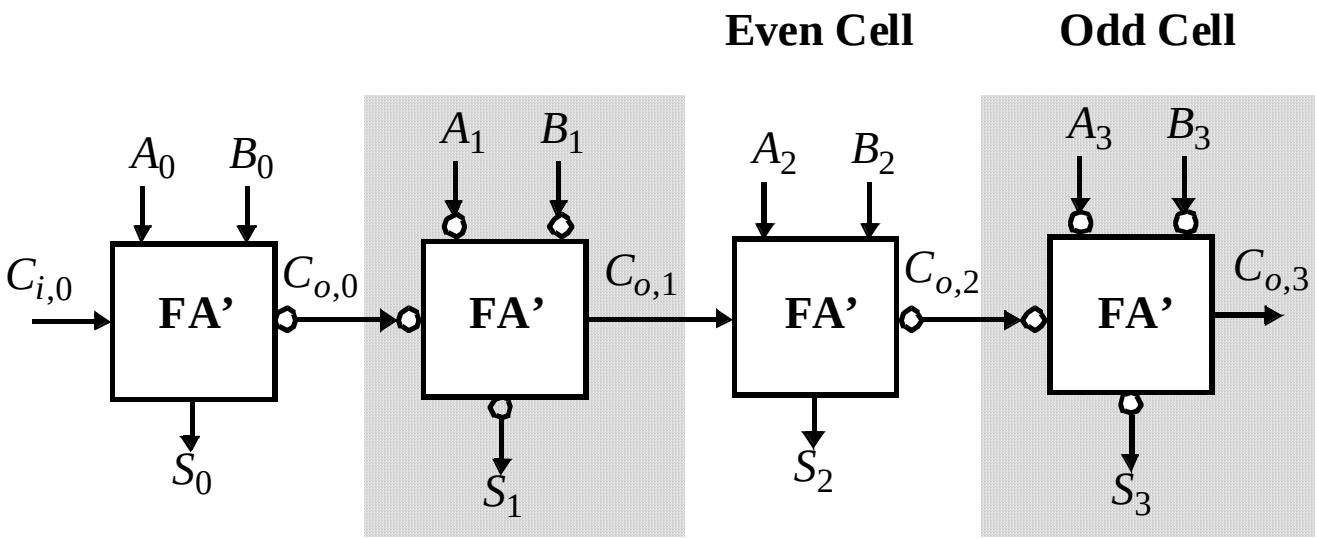


$$\bar{S}(A,B,C_i) = S(\bar{A},\bar{B},\bar{C}_i)$$

$$\overline{C_o}(A,B,C_i) = C_o(\bar{A},\bar{B},\bar{C}_i)$$

From Rabaey

Minimize Critical Path by Reducing Inverting Stages

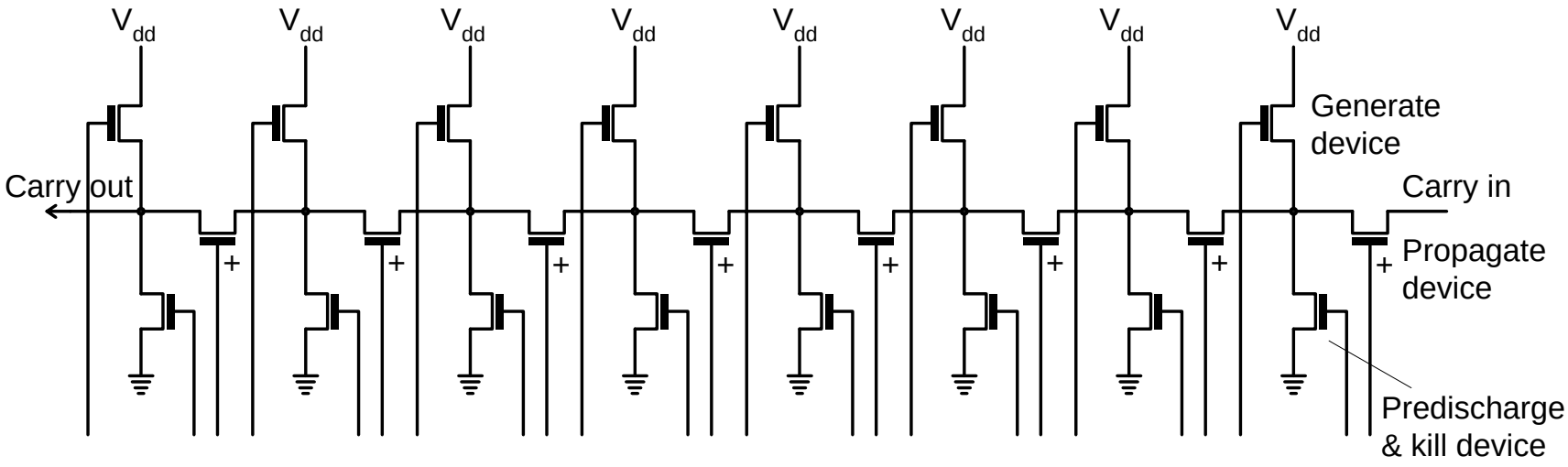


Exploit Inversion Property

From Rabaey **Note: need 2 different types of cells**

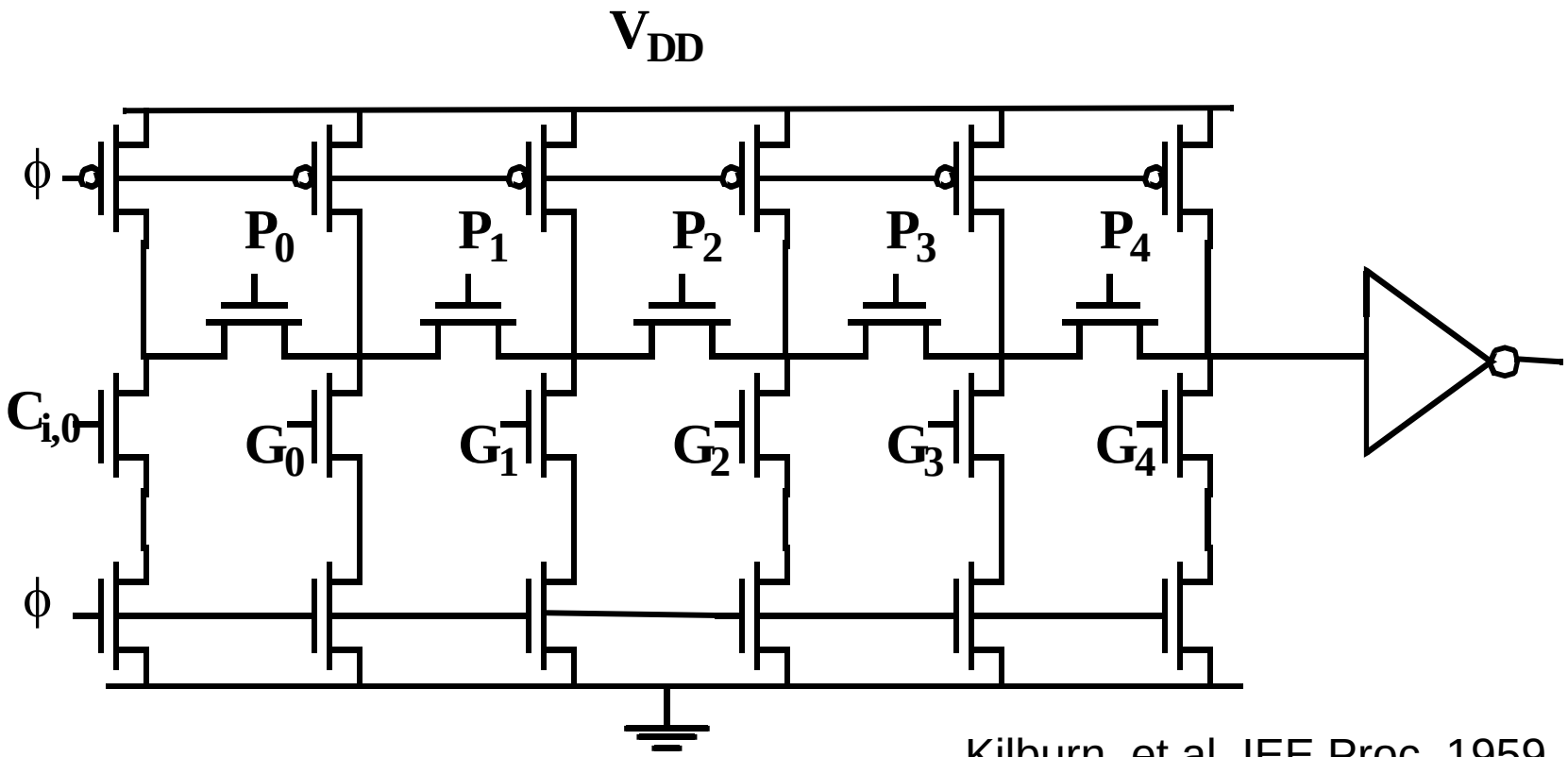
Manchester Carry-Chain Realization of the Carry Path

- Simple and very popular scheme for implementation of carry signal path



Manchester Carry Chain

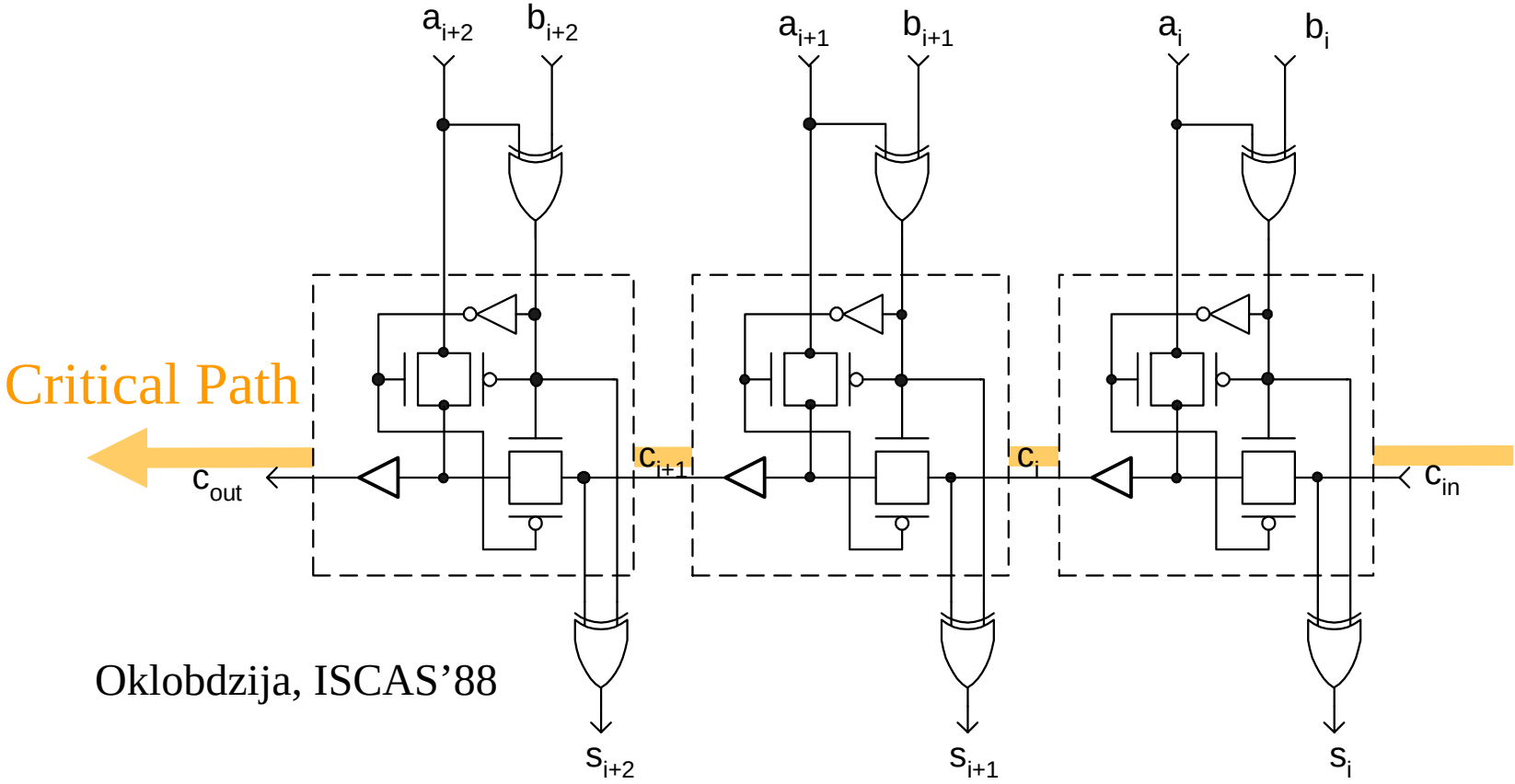
- Implement P with pass-transistors
- Implement G with pull-up, kill (delete) with pull-down
- Use dynamic logic to reduce the complexity and speed up



Kilburn, et al, IEE Proc, 1959.

Ripple Carry Adder

Carry-Chain of an RCA implemented using multiplexer from the standard cell library:



Oklobdzija, ISCAS'88

Carry-Lookahead Adder
(Weinberger and Smith)

Weinberger and J. L. Smith, “A Logic for High-Speed Addition”, National Bureau of Standards, Circ. 591, p.3-12, 1958.

Carry-Lookahead Adder

(Weinberger and Smith)

$$c_{i+1} = \bar{a}_i b_i c_i + a_i \bar{b}_i c_i + a_i b_i = g_i + p_i c_i$$

$$c_{i+2} = g_{i+1} + p_{i+1} c_{i+1}$$

$$= g_{i+1} + p_{i+1}(g_i + p_i c_1)$$

$$= g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_1$$

$$c_{i+3} = g_{i+2} + p_{i+2} c_{i+2}$$

$$= g_{i+2} + p_{i+2}(g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_i)$$

$$= g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i c_i$$

Carry-Lookahead Adder

$$G_j = g_{i+3} + p_{i+3}g_{i+2} + p_{i+3}p_{i+2}g_{i+1} + p_{i+3}p_{i+2}p_{i+1}c_j$$

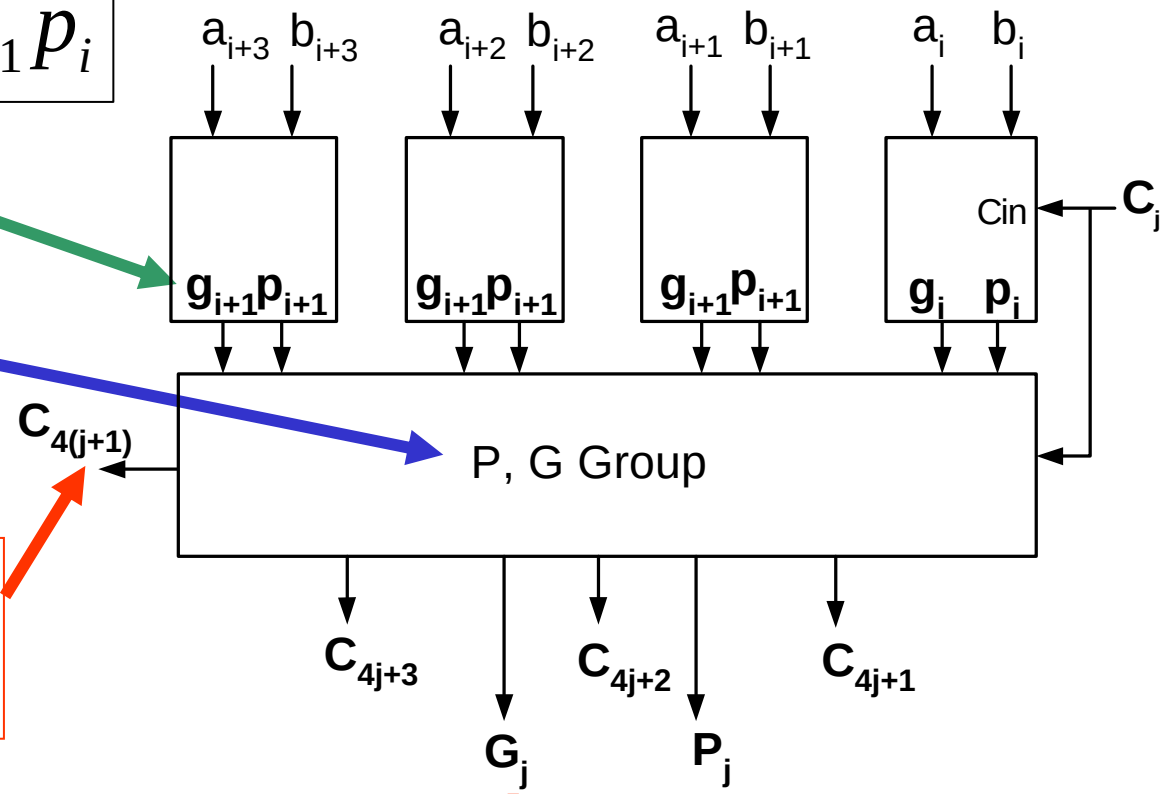
$$P_j = p_{i+3}p_{i+2}p_{i+1}p_i$$

One gate delay Δ to calculate p, g

One Δ to calculate P and two for G

Three gate delays To calculate $C_{4(j+1)}$

$$c_{4(j+1)} = G_i + P_i c_{4j}$$



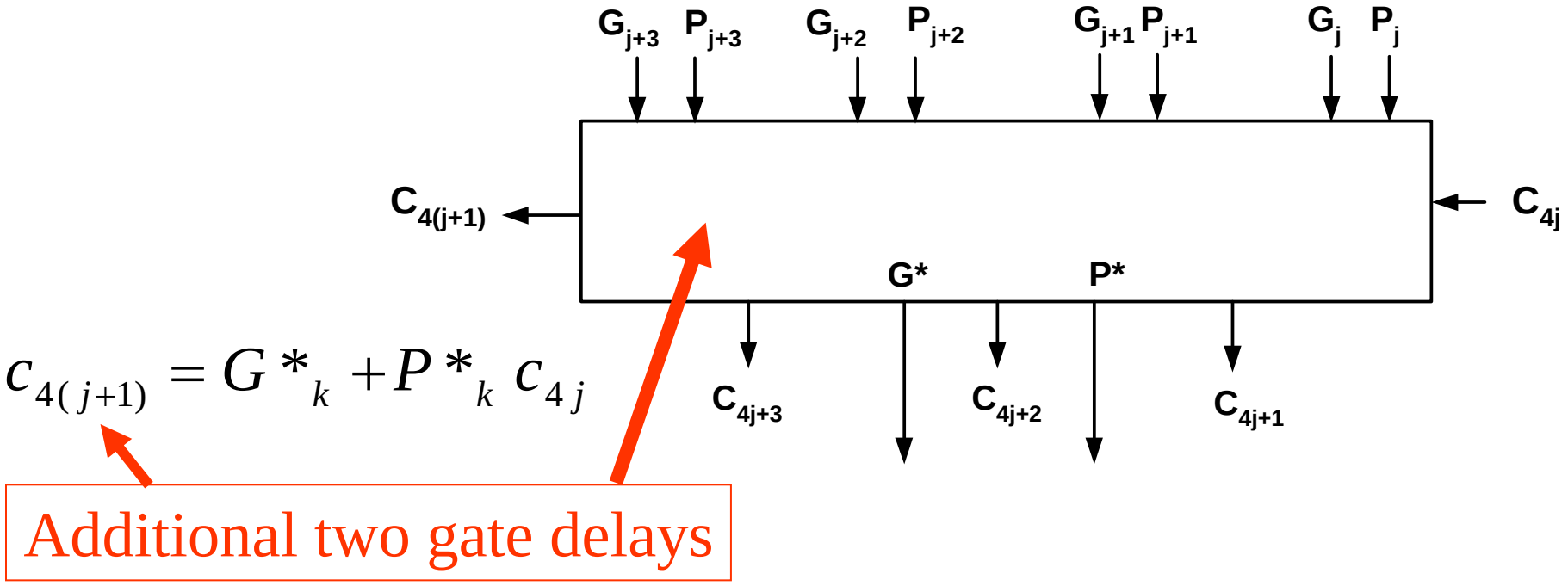
Compare that to 8 Δ in RCA !

Carry-Lookahead Adder

(Weinberger and Smith)

$$G^*_j = G_{i+3} + P_{i+3}G_{i+2} + P_{i+3}P_{i+2}G_{i+1} + P_{i+3}P_{i+2}P_{i+1}G_i$$

$$P^*_j = P_{i+3}P_{i+2}P_{i+1}P_i$$

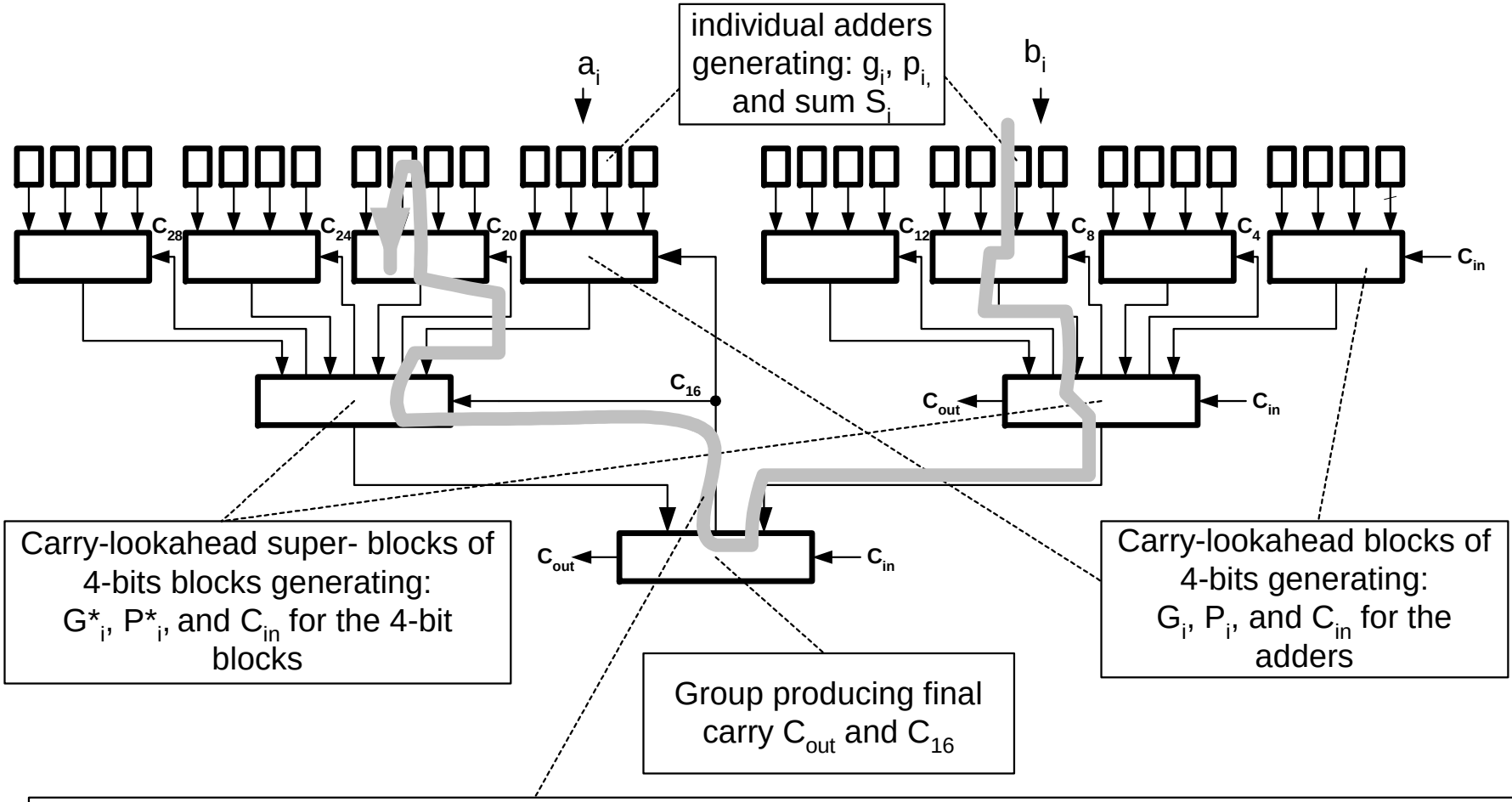


$$c_{4(j+1)} = G^*_k + P^*_k c_{4j}$$

Additional two gate delays

C_{16} will take a total of 5Δ vs. 32Δ for RCA !

32-bit Carry Lookahead Adder



Critical path delay = 1Δ (for gi,pi)+2x2Δ (for G,P)+3x2Δ (for Cin)+1XOR-Δ (for Sum) = appx. 12Δ of delay

Carry-Lookahead Adder

(Weinberger and Smith: original derivation)

$$C_1 = A_1B_1 + (A_1+B_1)C_0$$

$$C_2 = A_2B_2 + (A_2+B_2)C_1 = A_2B_2 + (A_2+B_2)A_1B_1 + (A_2+B_2)(A_1+B_1)C_0$$

$$C_3 = A_3B_3 + (A_3+B_3)C_2 = A_3B_3 + (A_3+B_3)A_2B_2 + (A_3+B_3)(A_2+B_2)A_1B_1 + (A_3+B_3)(A_2+B_2)(A_1+B_1)C_0$$

Carry-Lookahead Adder

(Weinberger and Smith: original derivation)

$$
\begin{aligned}
C_4 = \; & A_4B_4 \\
& +(A_4+B_4)C_3 \\
= \; & A_4B_4 \\
& +(A_4+B_4)A_3B_3 \\
& +(A_4+B_4)(A_3+B_3)A_2B_2 \\
& +(A_4+B_4)(A_3+B_3)(A_2+B_2)A_1B_1 \\
& +(A_4+B_4)(A_3+B_3)(A_2+B_2)(A_1+B_1)C_0 \\
= \; & A_4B_4 \\
& +(A_4+B_4)A_3B_3 \\
& +(A_4+B_4)(A_3+B_3)A_2B_2 \\
& +(A_4+B_4)(A_3+B_3)(A_2+B_2)(A_1+B_1)(A_1+C_0)(B_1+C_0).
\end{aligned}
$$

Carry-Lookahead Adder (Weinberger and Smith)

please notice the similarity with Parallel-Prefix Adders !

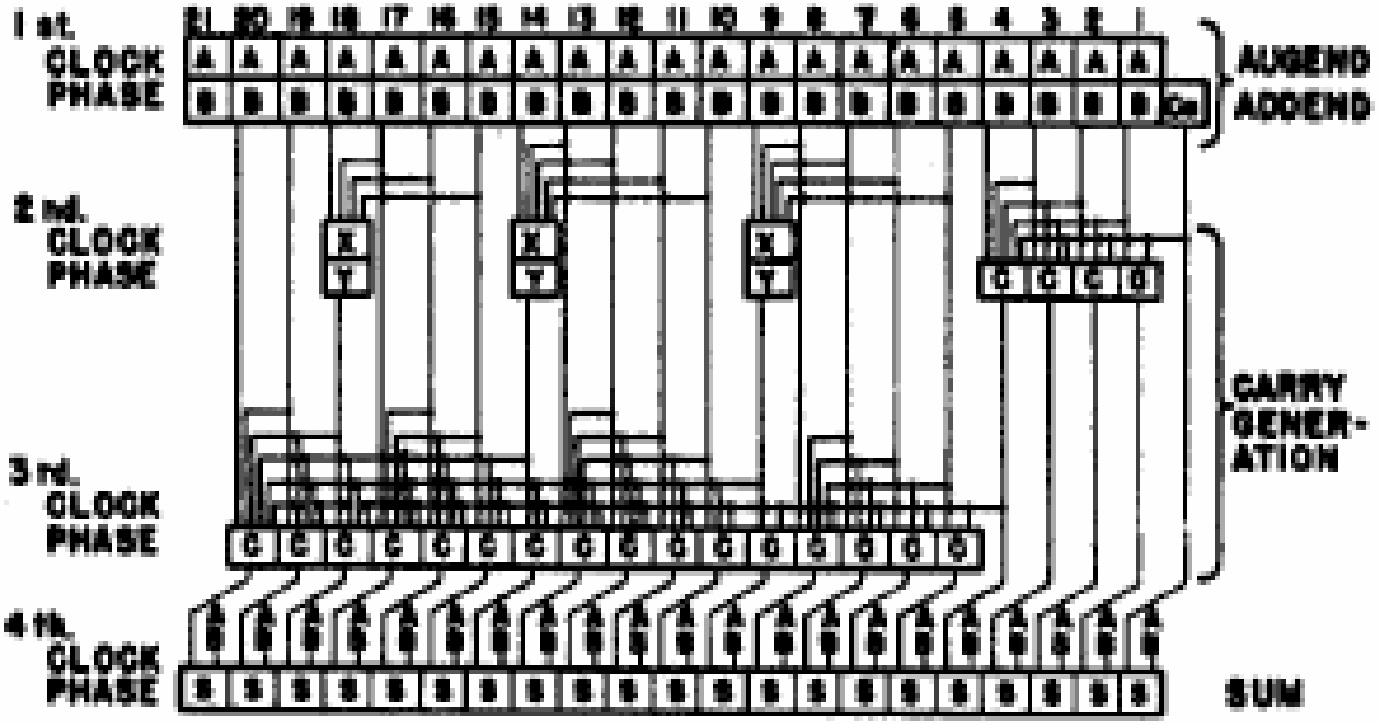


FIGURE 1.9. *Twenty-one-bit parallel binary adder*

Carry-Lookahead Adder (Weinberger and Smith)

please notice the similarity with Parallel-Prefix Adders !

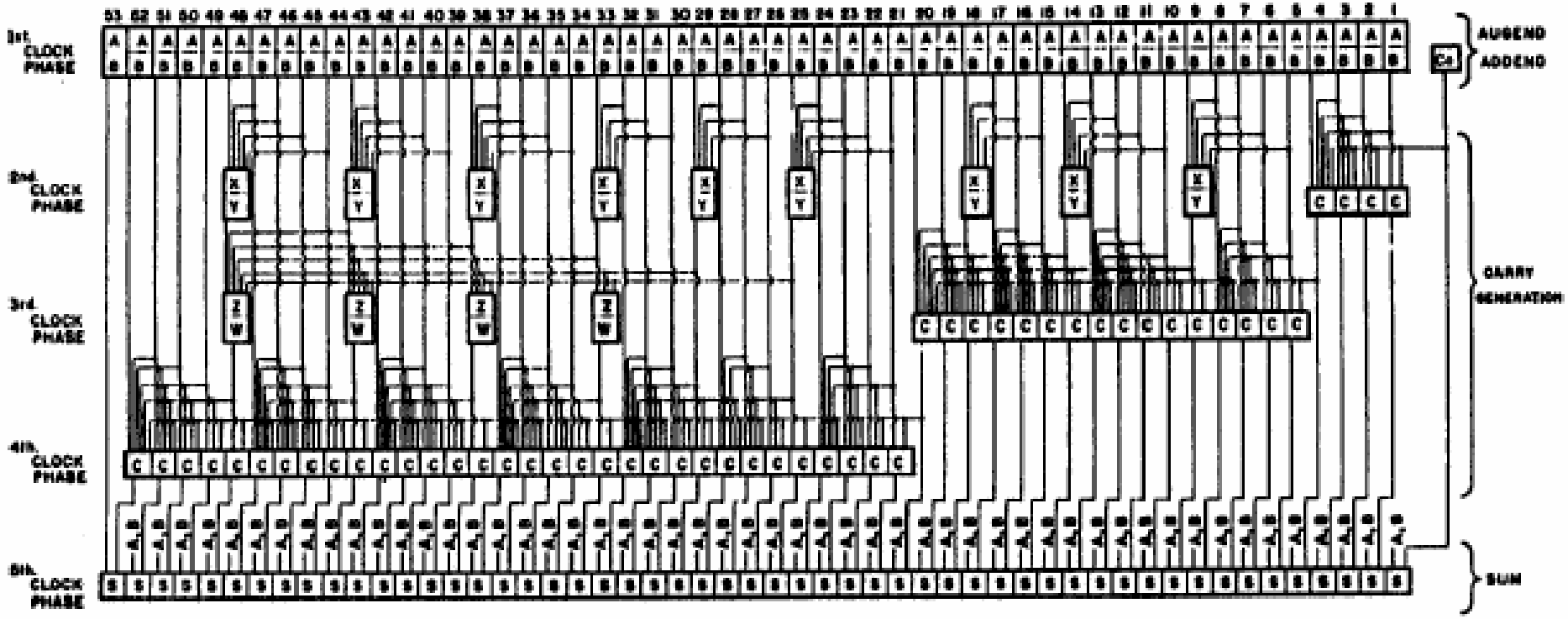


FIGURE 1.10. *Fifty-three-bit parallel binary adder.*