CSE477
VLSI Digital Circuits
Fall 2002

Lecture 20: Adder Design

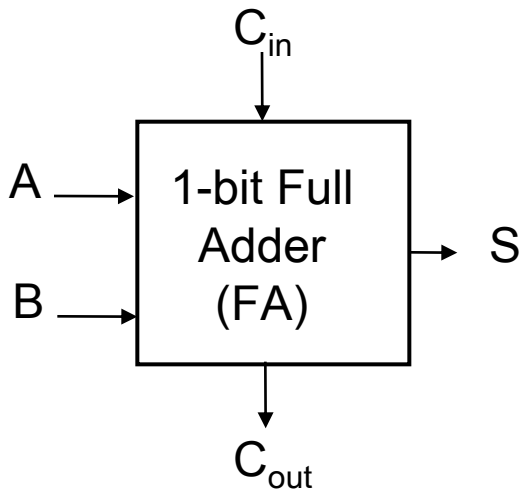
Mary Jane Irwin (www.cse.psu.edu/~mji)
www.cse.psu.edu/~cg477

[Adapted from Rabaey's *Digital Integrated Circuits*, ©2002, J. Rabaey et al.]

Review: Basic Building Blocks

- Datapath
  - Execution units
    - Adder, multiplier, divider, shifter, etc.
  - Register file and pipeline registers
  - Multiplexers, decoders
- Control
  - Finite state machines (PLA, ROM, random logic)
- Interconnect
  - Switches, arbiters, buses
- Memory
  - Caches (SRAMs), TLBs, DRAMs, buffers

The 1-bit Binary Adder



A	B	C_{in}	C_{out}	S	carry status
0	0	0	0	0	kill
0	0	1	0	1	kill
0	1	0	0	1	propagate
0	1	1	1	0	propagate
1	0	0	0	1	propagate
1	0	1	1	0	propagate
1	1	0	1	0	generate
1	1	1	1	1	generate

$G = A\&B$
$P = A \oplus B$
$K = !A \ \& \ !B$

$S = A \oplus B \oplus C_{in} = P \oplus C_{in}$

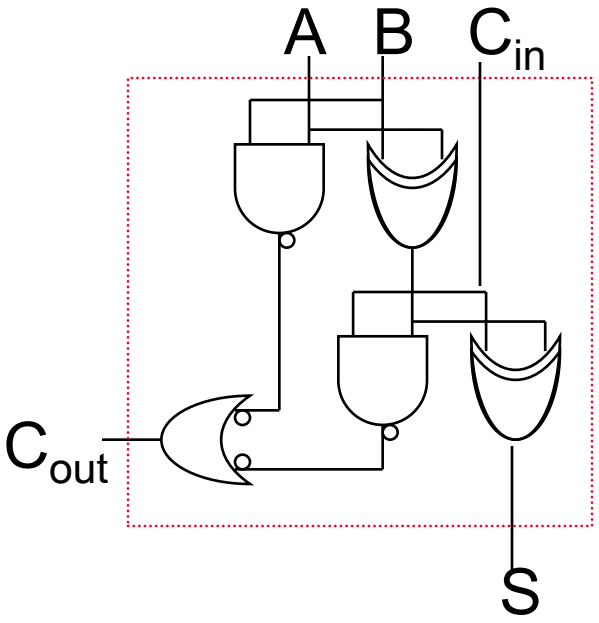
$C_{out} = A\&B \mid A\&C_{in} \mid B\&C_{in}$ (majority function)

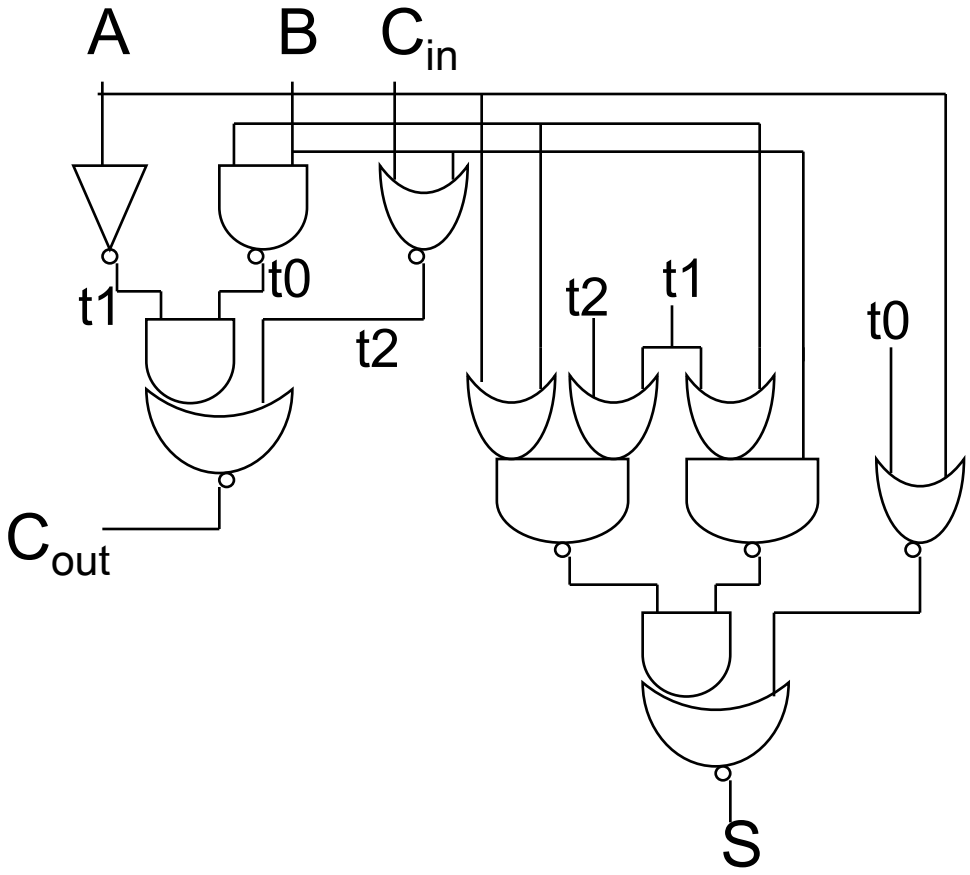
$= G \mid P\&C_{in}$

- How can we use it to build a 64-bit adder?
- How can we modify it easily to build an adder/subtractor?
- How can we make it better (faster, lower power, smaller)?

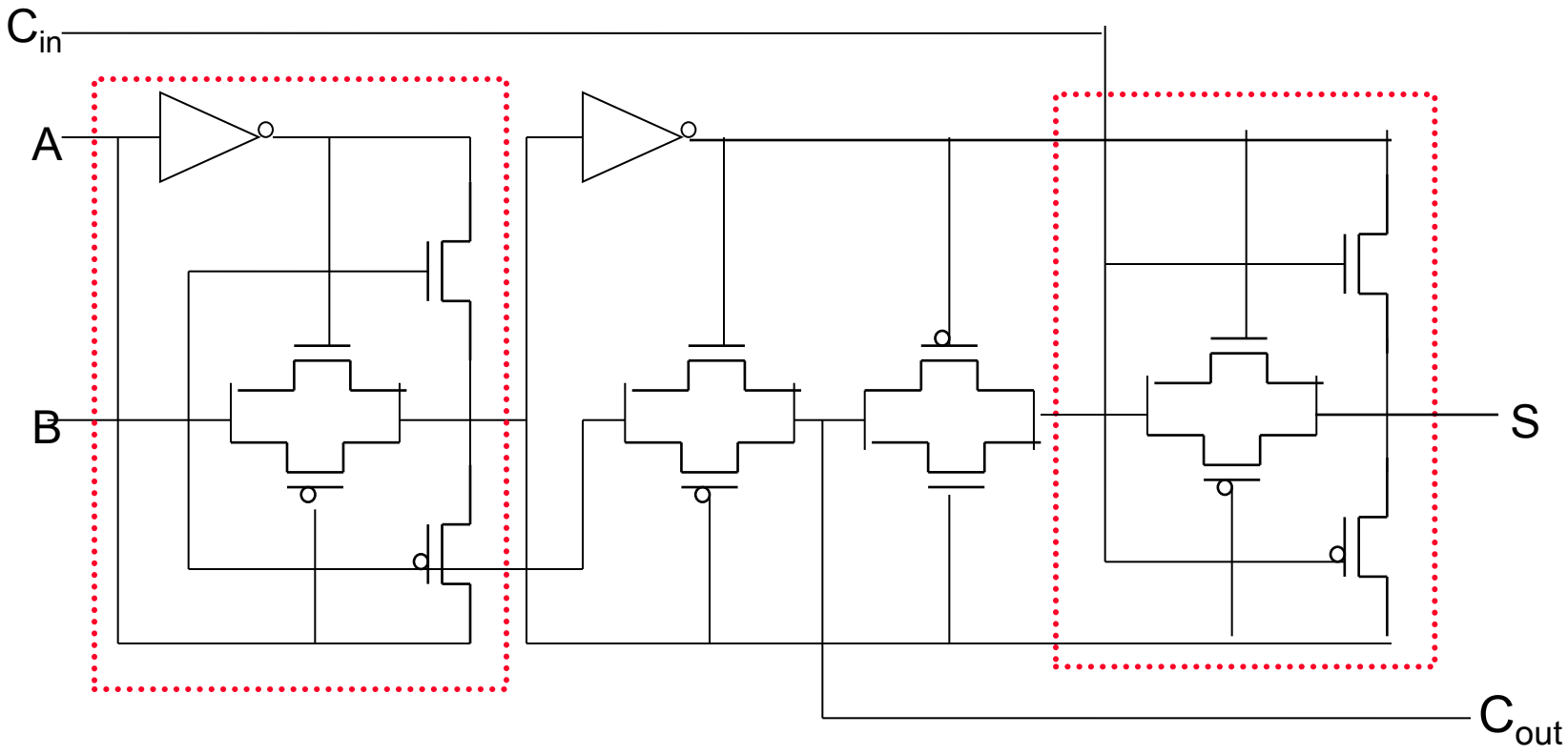
FA Gate Level Implementations

- The way you learned to design in CSE271 and CSE471



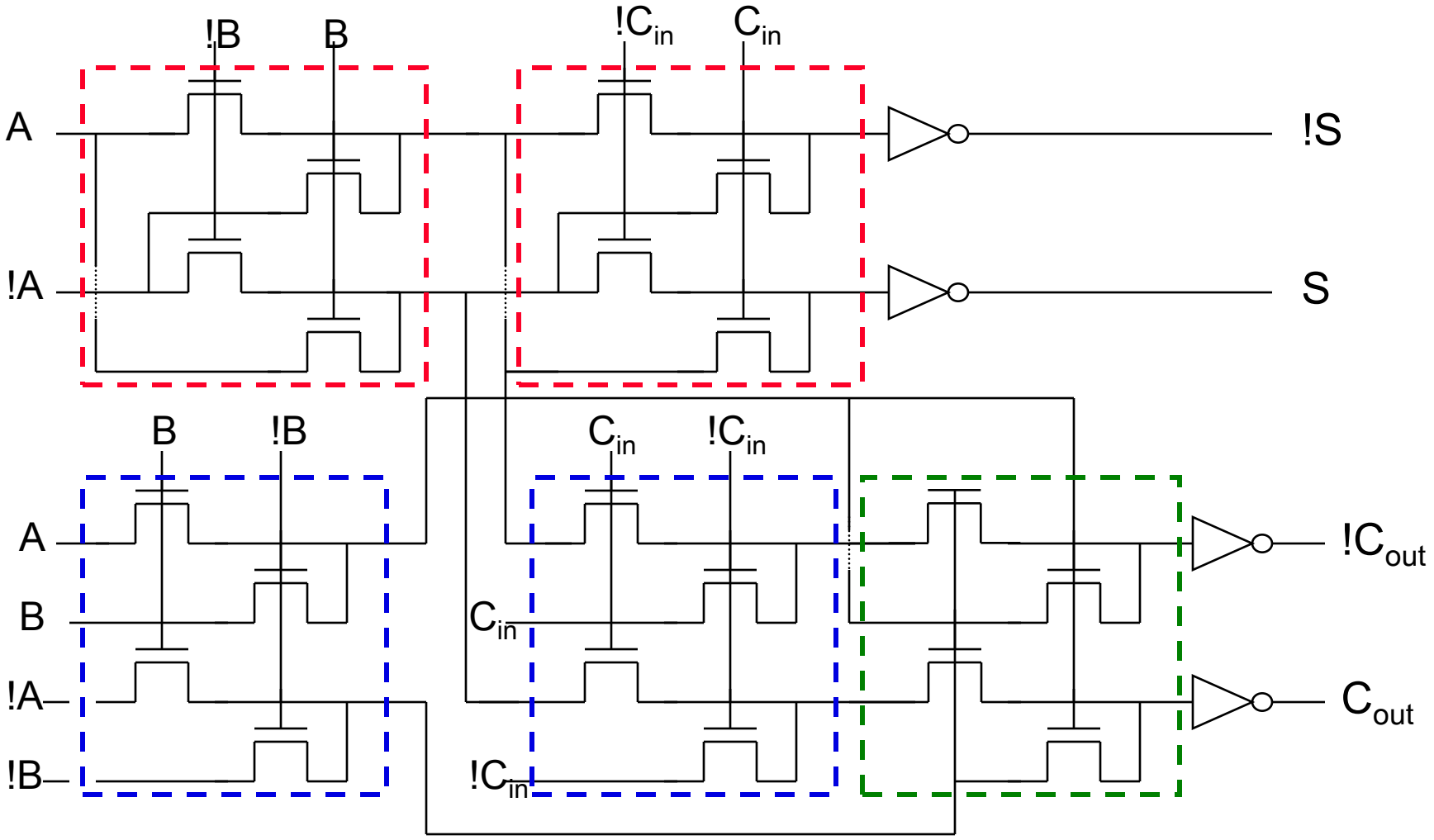


Review: XOR FA



16 transistors

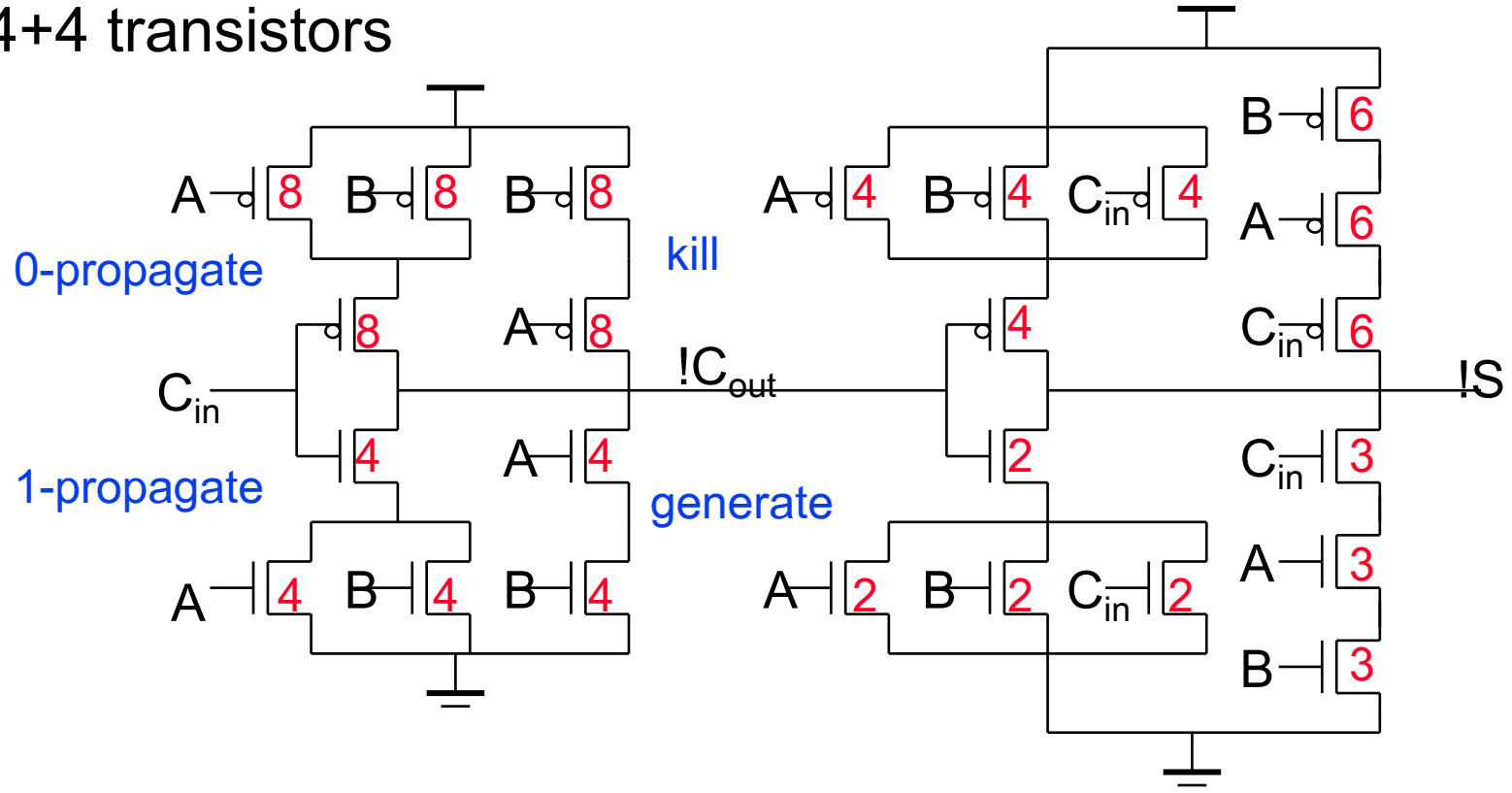
Review: CPL FA



20+8 transistors, dual rail – beware of threshold drops

Review: Mirror Adder

24+4 transistors



$C_{out} = A\&B \mid B\&C_{in} \mid A\&C_{in}$

$SUM = A\&B\&C_{in} \mid C_{OUT}\&(A \mid B \mid C_{in})$

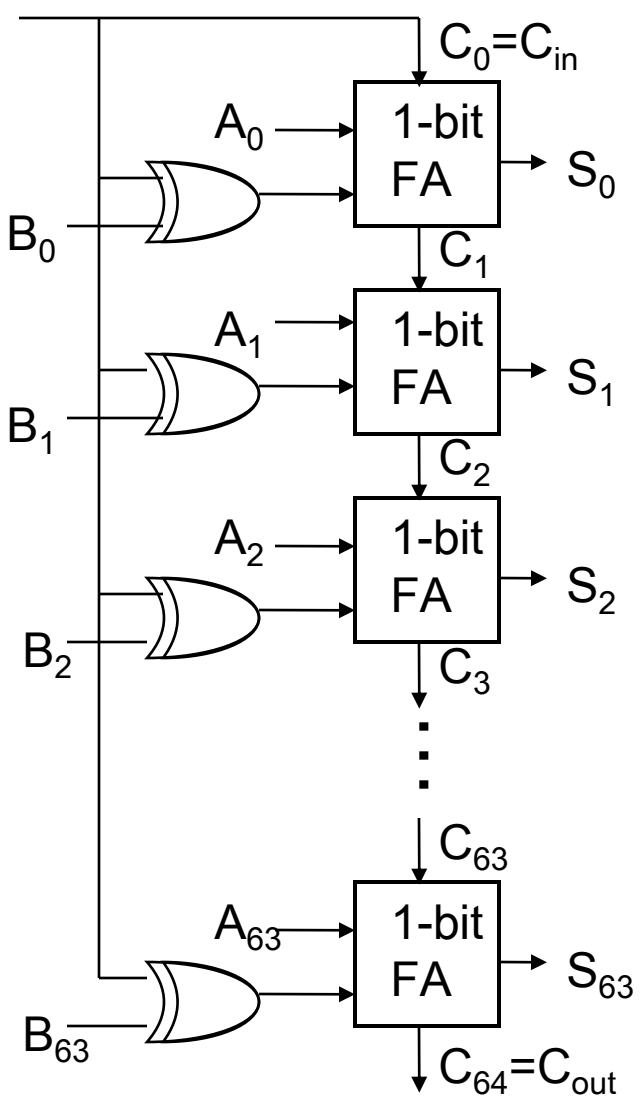
Sizing: Each input in the carry circuit has a logical effort of 2 so the optimal fan-out for each is also 2. Since $!C_{out}$ drives 2 internal and 2 inverter transistor gates (to form C_{in} for the nms bit adder) should oversize the carry circuit. PMOS/NMOS ratio of 2.

Mirror Adder Features

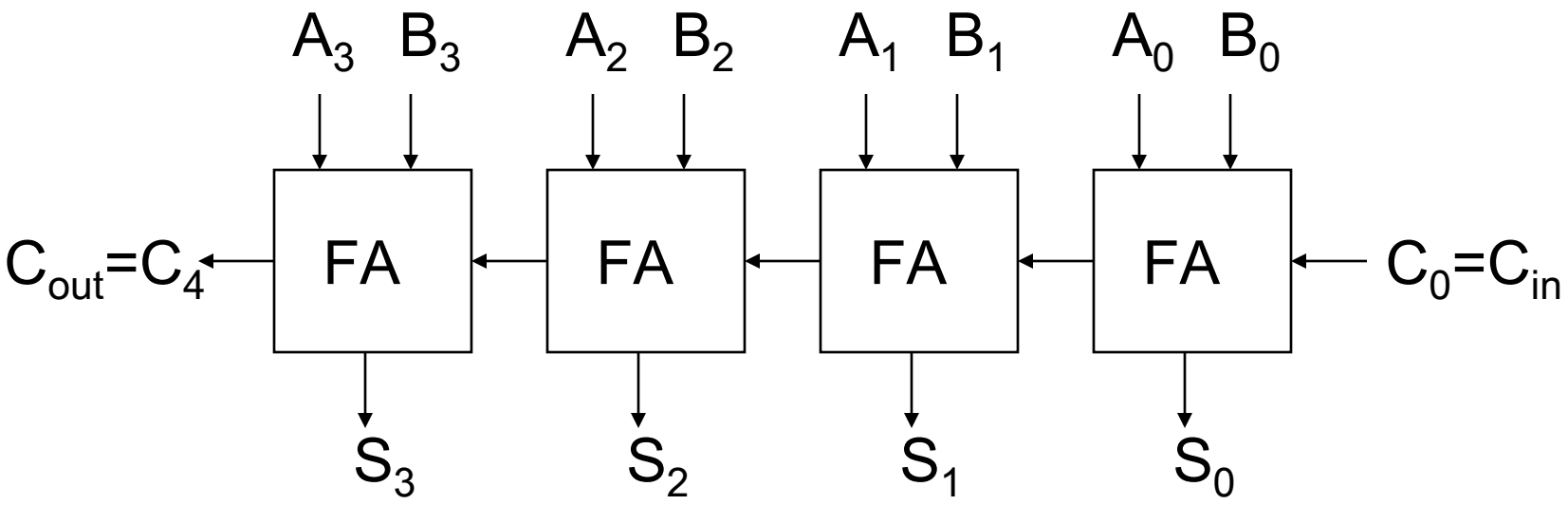
- The NMOS and PMOS chains are completely symmetrical with a maximum of two series transistors in the carry circuitry, guaranteeing identical rise and fall transitions if the NMOS and PMOS devices are properly sized.
- When laying out the cell, the most critical issue is the minimization of the capacitances at node $!C_{out}$ (four diffusion capacitances, two internal gate capacitances, and two inverter gate capacitances). Shared diffusions can reduce the stack node capacitances.
- The transistors connected to C_{in} are placed closest to the output.
- Only the transistors in the carry stage have to be optimized for optimal speed. All transistors in the sum stage can be minimal size.

A 64-bit Adder/Subtractor

- Ripple Carry Adder (RCA) built out of 64 FAs
- Subtraction – complement all subtrahend bits (xor gates) and set the low order carry-in
- RCA
  - advantage: simple logic, so small (low cost)
  - disadvantage: slow (O(N) for N bits) and lots of glitching (so lots of energy consumption)



Ripple Carry Adder (RCA)



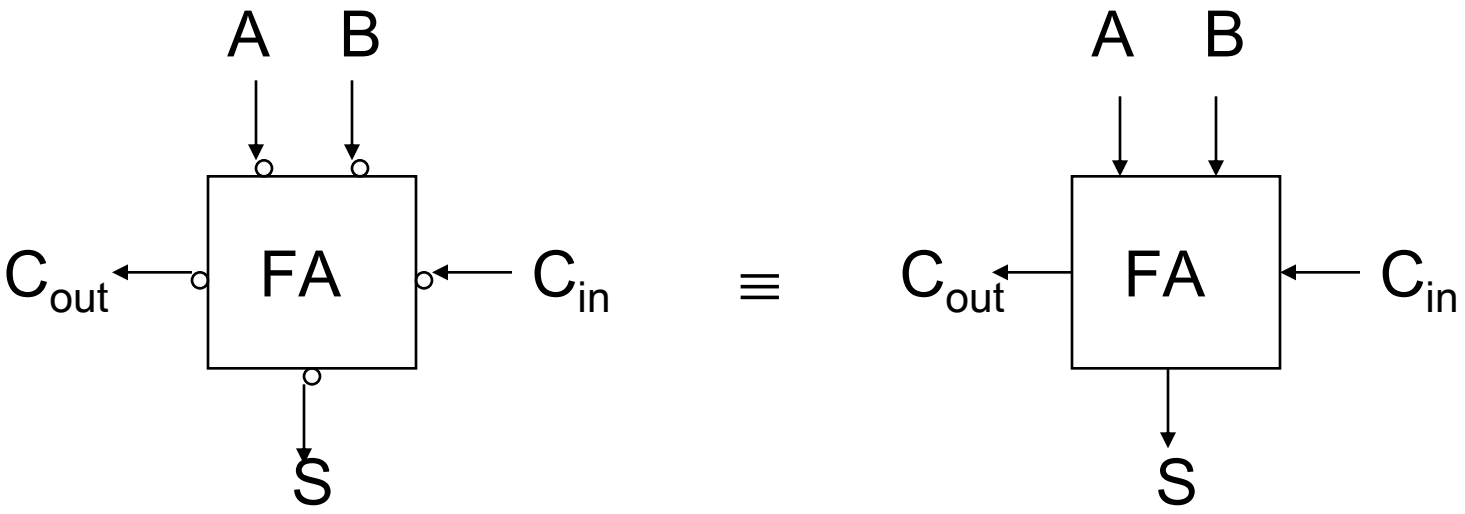
$$T_{adder} \approx T_{FA}(A,B \rightarrow C_{out}) + (N-2)T_{FA}(C_{in} \rightarrow C_{out}) + T_{FA}(C_{in} \rightarrow S)$$

T = O(N) worst case delay

Real Goal: Make the fastest possible carry path

Inversion Property

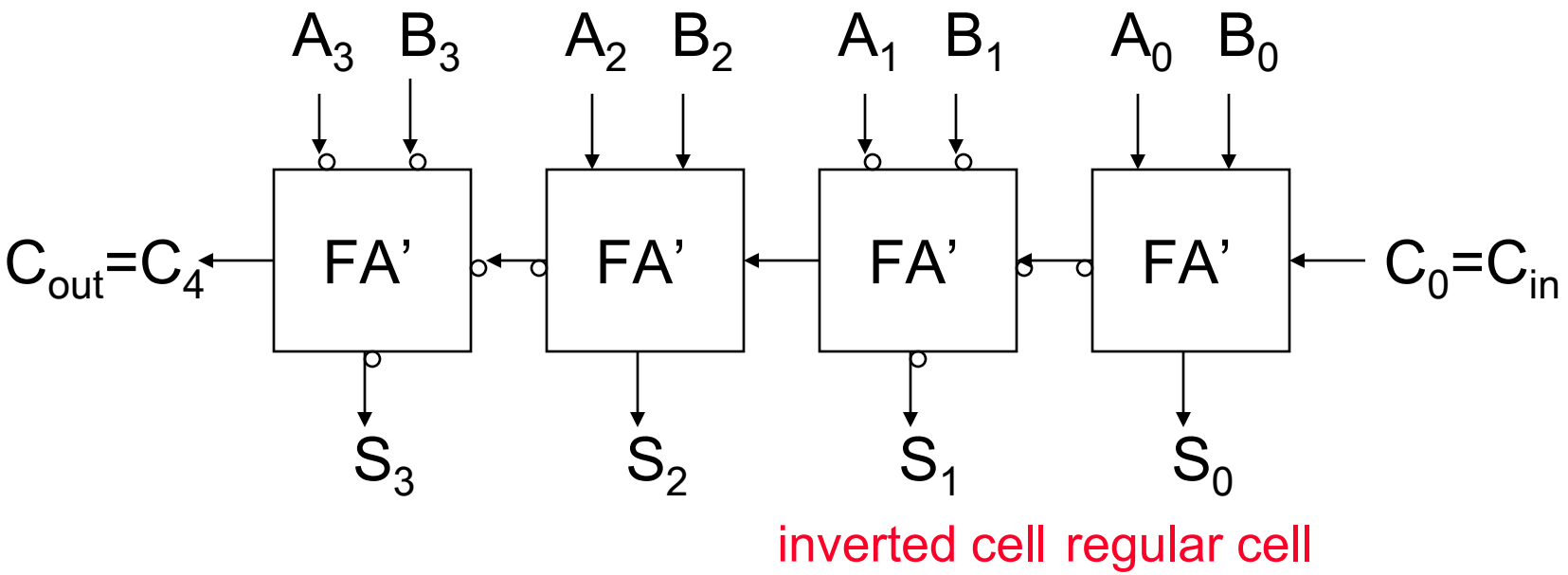
- Inverting all inputs to a FA results in inverted values for all outputs



$$!S\,(A, B, C_{in}) = S(!A, !B, !C_{in})$$

$$!C_{out}\,(A, B, C_{in}) = C_{out}\,(!A, !B, !C_{in})$$

Exploiting the Inversion Property



- Minimizes the critical path (the carry chain) by eliminating inverters between the FAs (will need to increase the transistor sizing on the carry chain portion of the mirror adder).

Now need two "flavors" of FAs

Fast Carry Chain Design

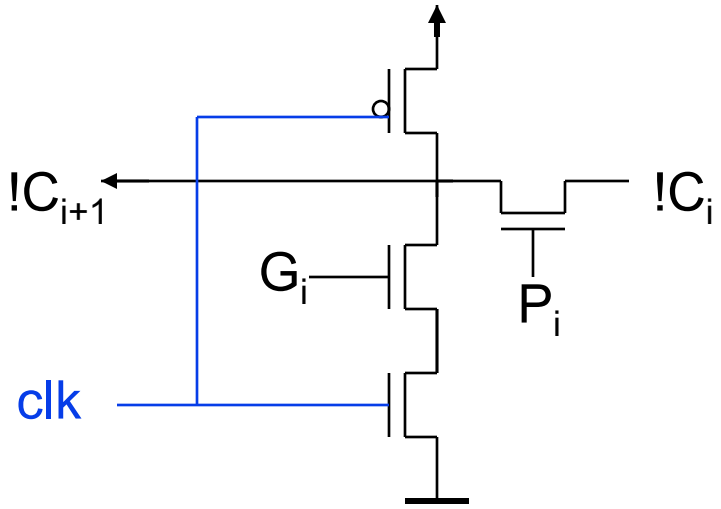
- The key to fast addition is a low latency carry network
- What matters is whether in a given position a carry is
  - generated $G_i = A_i \,\&\, B_i = A_iB_i$
  - propagated $P_i = A_i \oplus B_i$ (sometimes use $A_i \mid B_i$)
  - annihilated (killed) $K_i = !A_i \,\&\, !B_i$
- Giving a carry recurrence of

$$C_{i+1} = G_i \mid P_iC_i$$

$$C_1 = G_0 \mid P_0C_0$$
$$C_2 = G_1 \mid P_1G_0 \mid P_1P_0\,C_0$$
$$C_3 = G_2 \mid P_2G_1 \mid P_2P_1G_0 \mid P_2P_1P_0\,C_0$$
$$C_4 = G_3 \mid P_3G_2 \mid P_3P_2G_1 \mid P_3P_2P_1G_0 \mid P_3P_2P_1P_0\,C_0$$

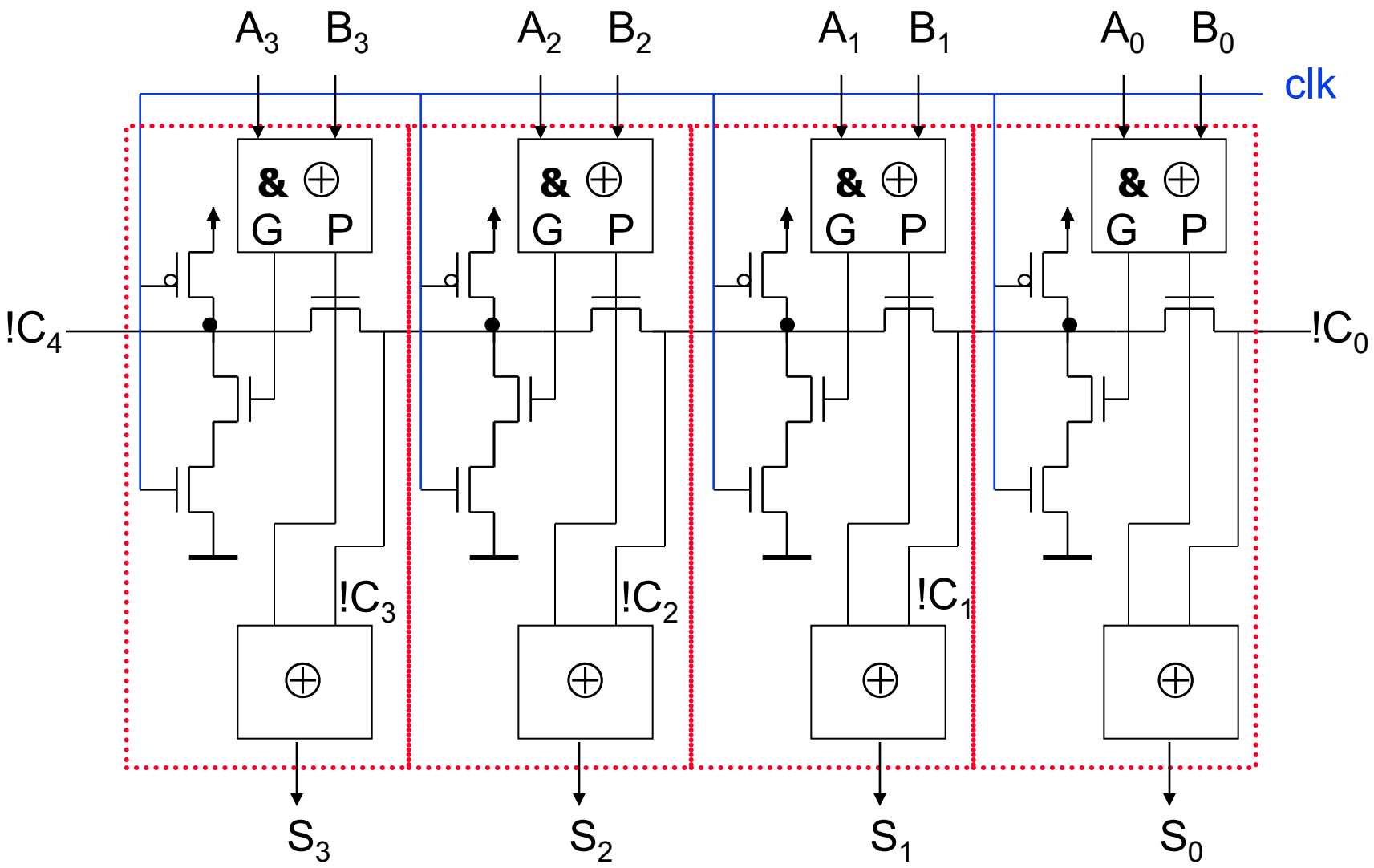
Manchester Carry Chain

- Switches controlled by G_i and P_i

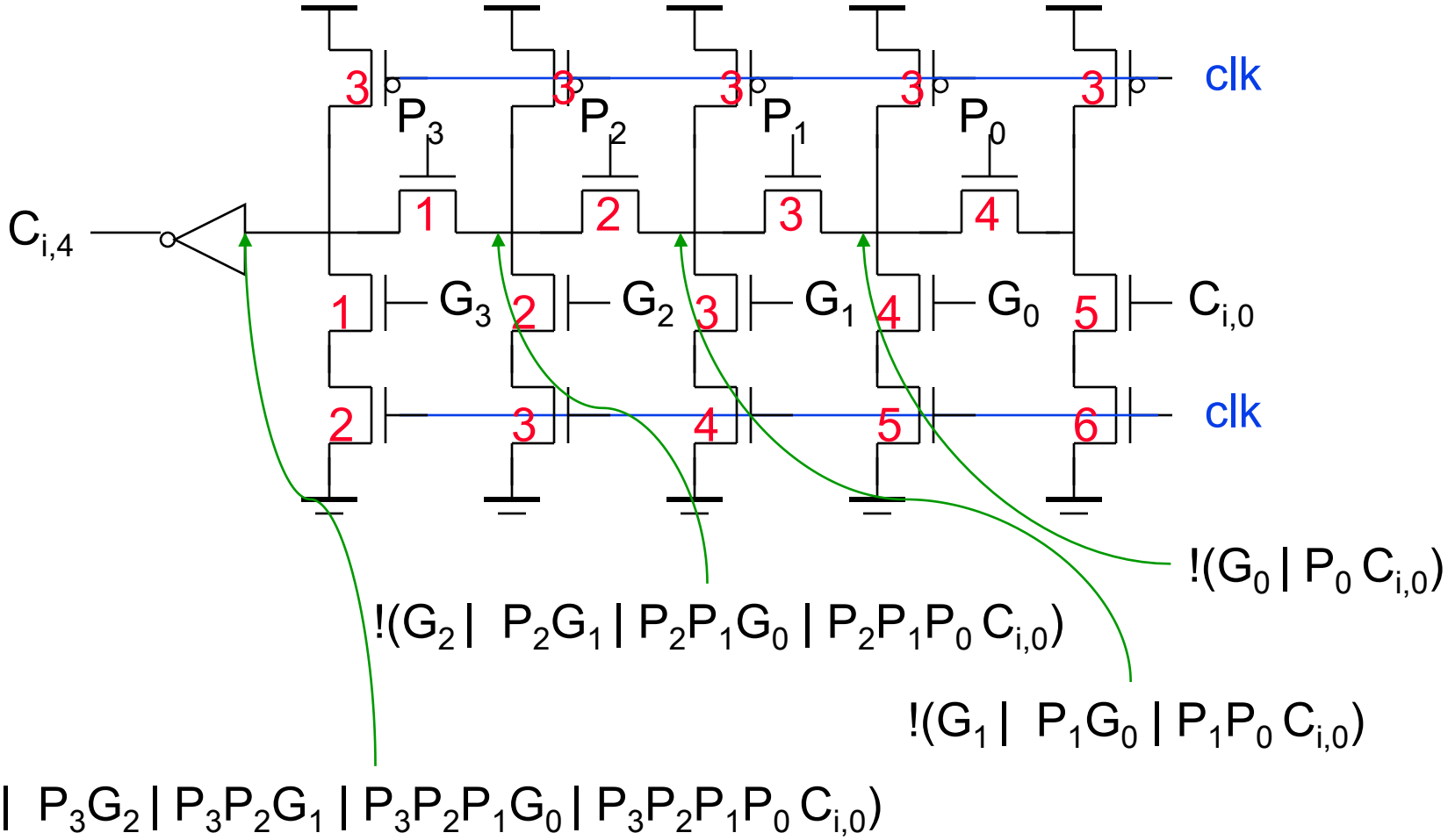


- Total delay of
  - time to form the switch control signals G_i and P_i
  - setup time for the switches
  - signal propagation delay through N switches in the worst case

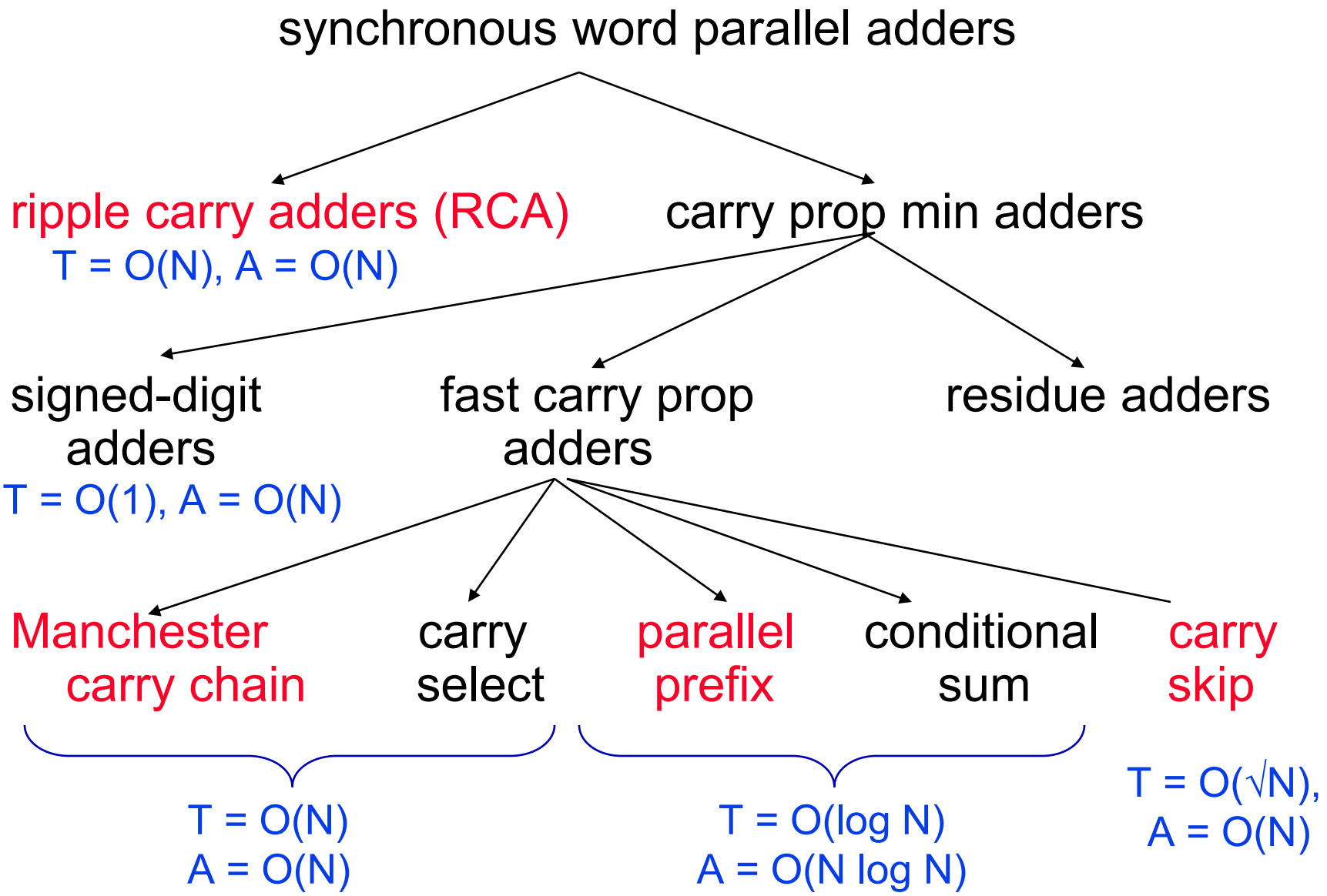
4-bit Sliced MCC Adder



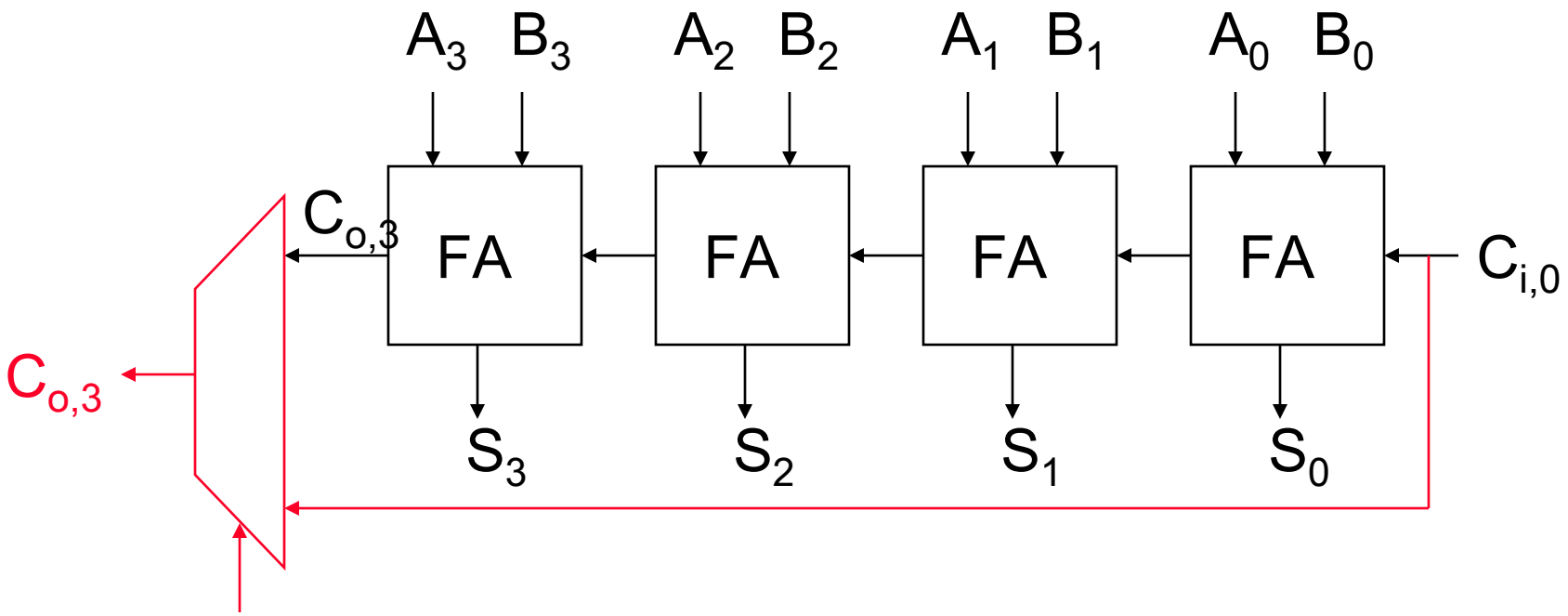
Domino Manchester Carry Chain Circuit



Binary Adder Landscape

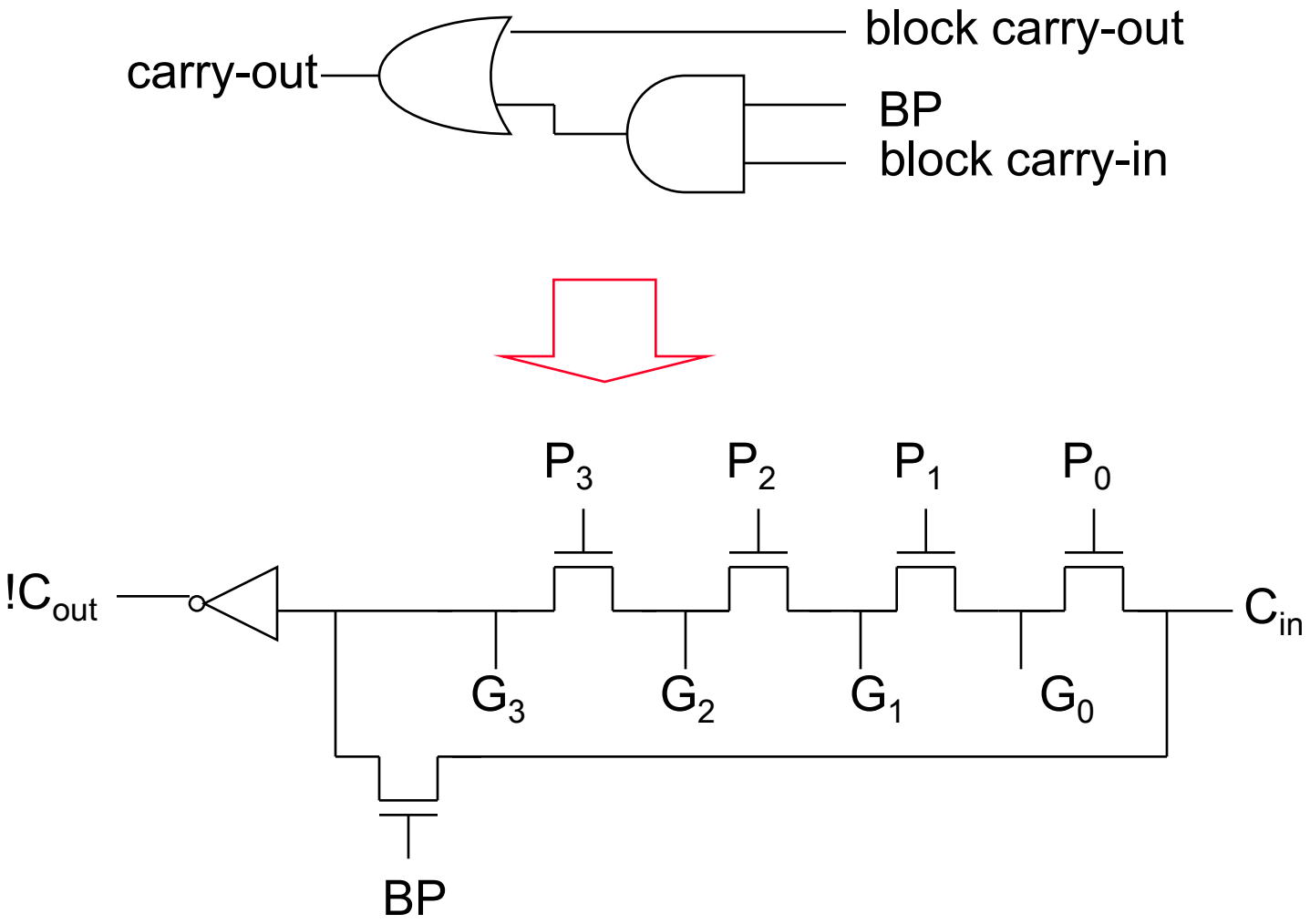


Carry-Skip (Carry-Bypass) Adder

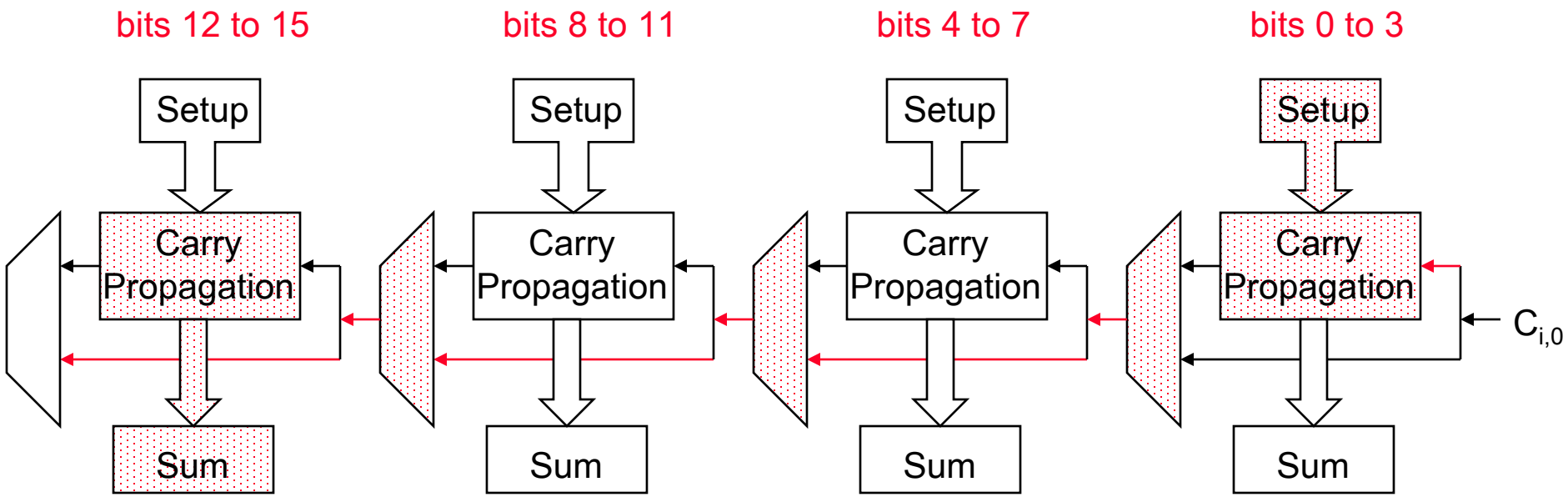


If (P_0 & P_1 & P_2 & P_3 = 1) then $C_{o,3} = C_{i,0}$ otherwise the block itself kills or generates the carry internally

Carry-Skip Chain Implementation



4-bit Block Carry-Skip Adder



Worst-case delay → carry from bit 0 to bit 15 = carry generated in bit 0, ripples through bits 1, 2, and 3, skips the middle two groups (B is the group size in bits), ripples in the last group from bit 12 to bit 15

$$T_{add} = t_{setup} + B\ t_{carry} + ((N/B) - 1)\ t_{skip} + B\ t_{carry} + t_{sum}$$

Optimal Block Size and Time

- Assuming one stage of ripple (t_{carry}) has the same delay as one skip logic stage (t_{skip}) and both are 1

$$T_{CSkA} = 1 + B + (N/B-1) + B + 1$$

t_{setup} — ripple in block 0 — skips — ripple in last block — t_{sum}

$$= 2B + N/B + 1$$

- So the optimal block size, B, is

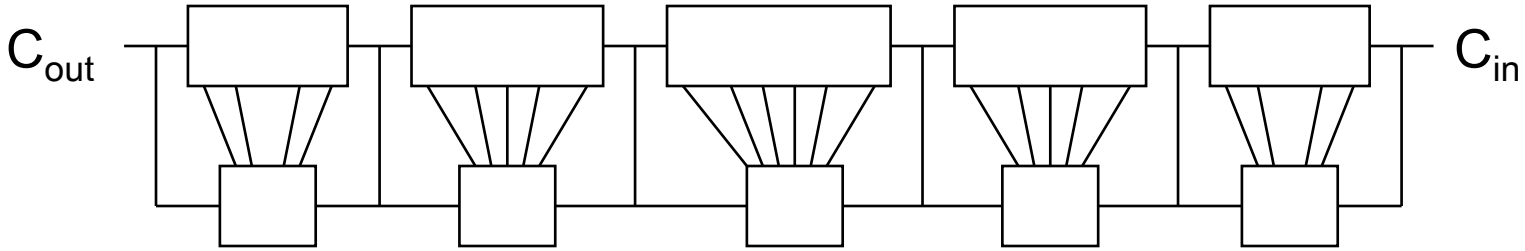
$$dT_{CSkA}/dB = 0 \Rightarrow \sqrt{(N/2)} = B^{opt}$$

- And the optimal time is

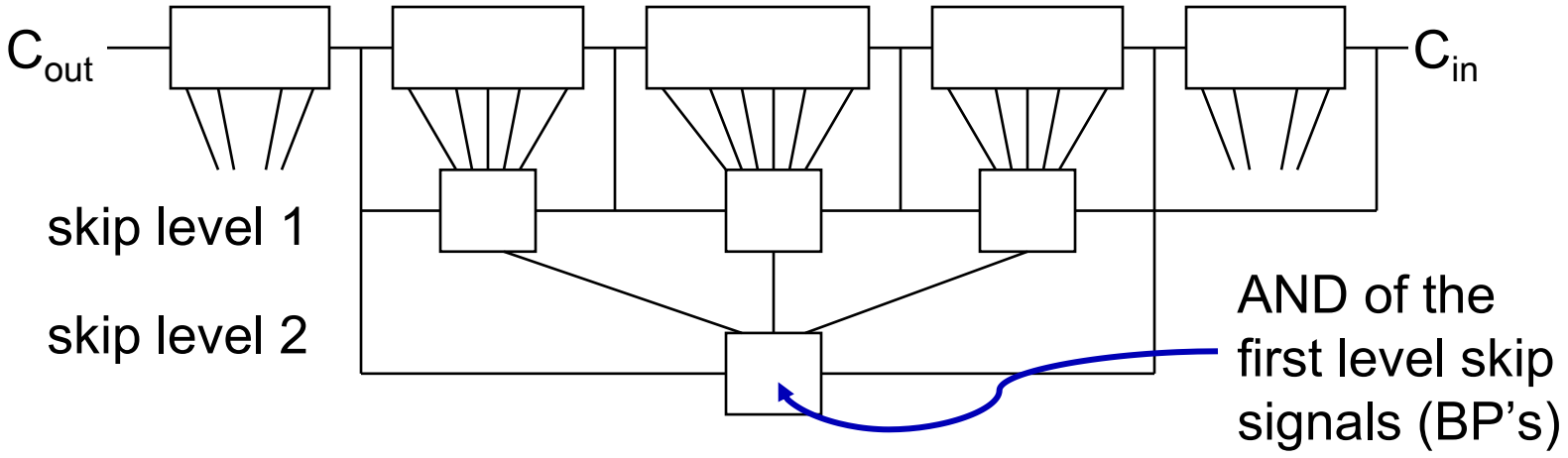
$$\text{Optimal } T_{CSkA} = 2(\sqrt{(2N)}) + 1$$

Carry-Skip Adder Extensions

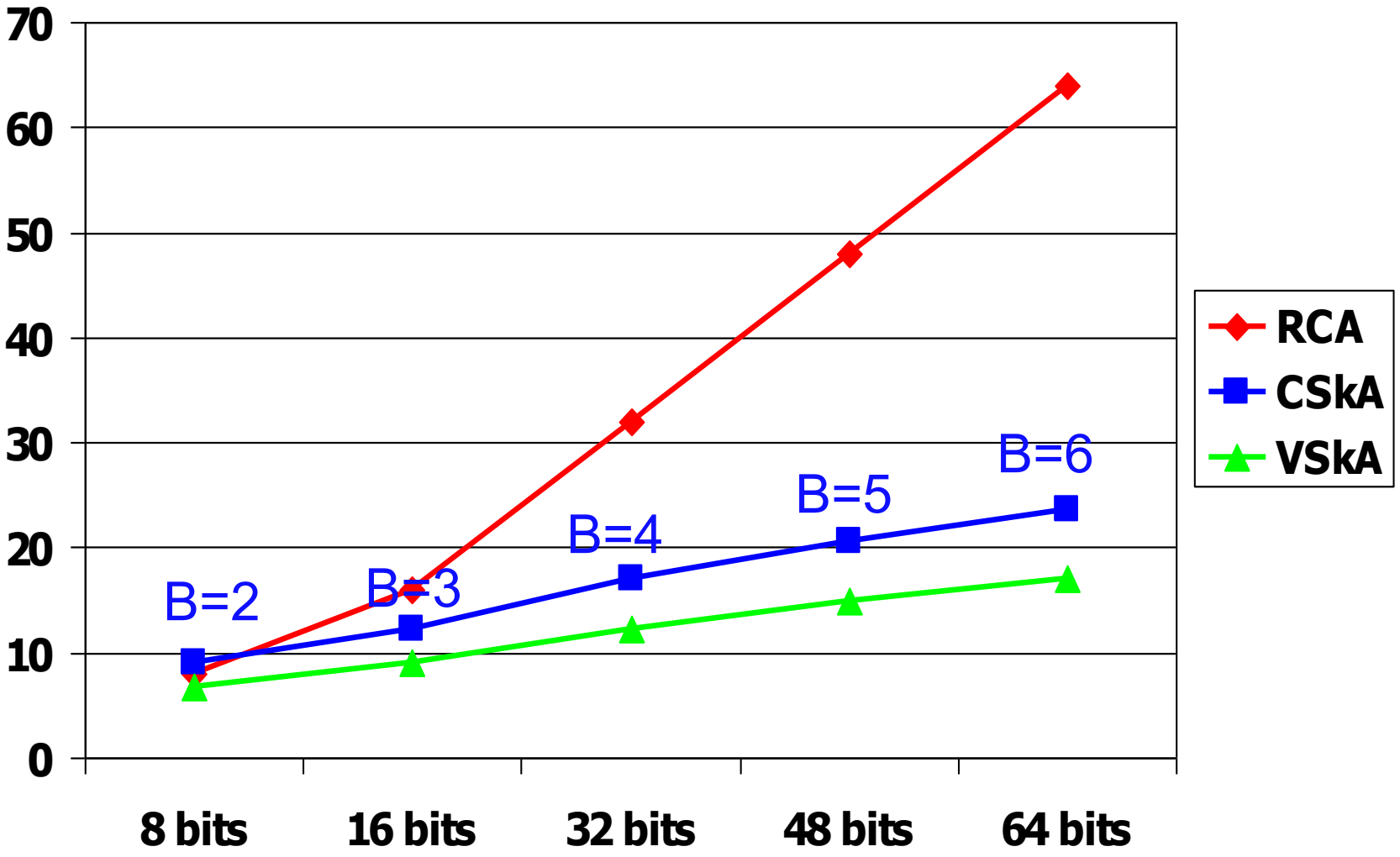
- Variable block sizes
  - A carry that is generated in, or absorbed by, one of the inner blocks travels a shorter distance through the skip blocks, so can have bigger blocks for the inner carries without increasing the overall delay



- Multiple levels of skip logic

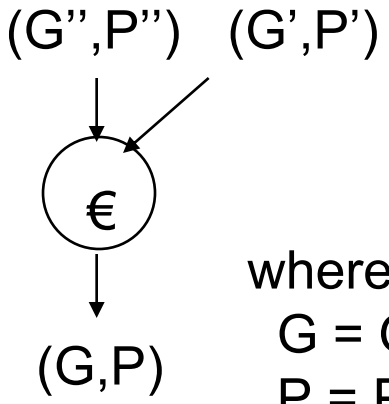


Carry-Skip Adder Comparisons



Parallel Prefix Adders (PPAs)

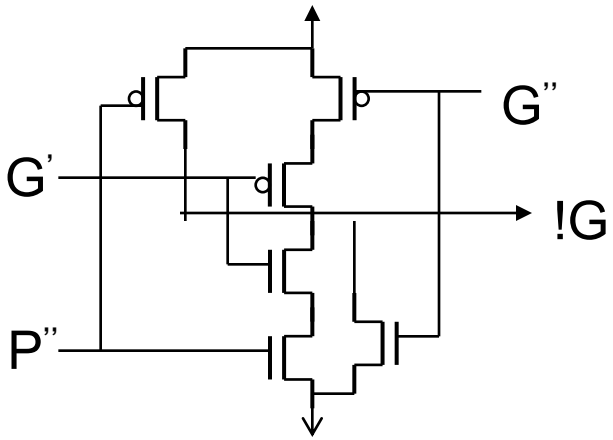
- Define carry operator € on (G,P) signal pairs



where

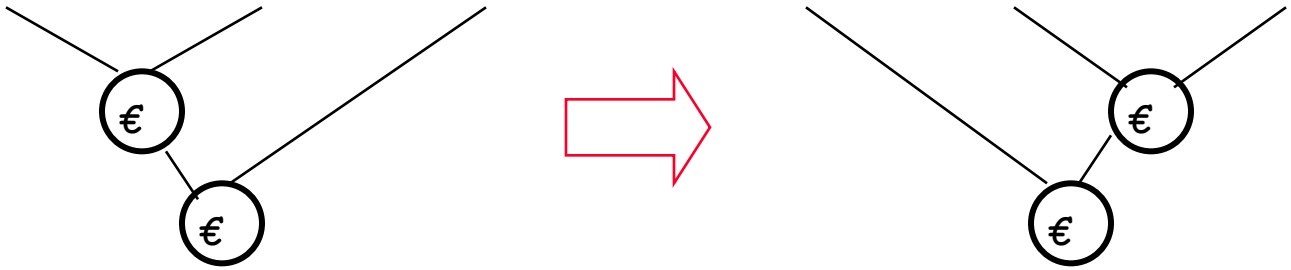
$G = G'' \vee P''G'$

$P = P''P'$



- € is associative, i.e.,

$[(g''',p''') \in (g'',p'')] \in (g',p') = (g''',p''') \in [(g'',p'') \in (g',p')]$

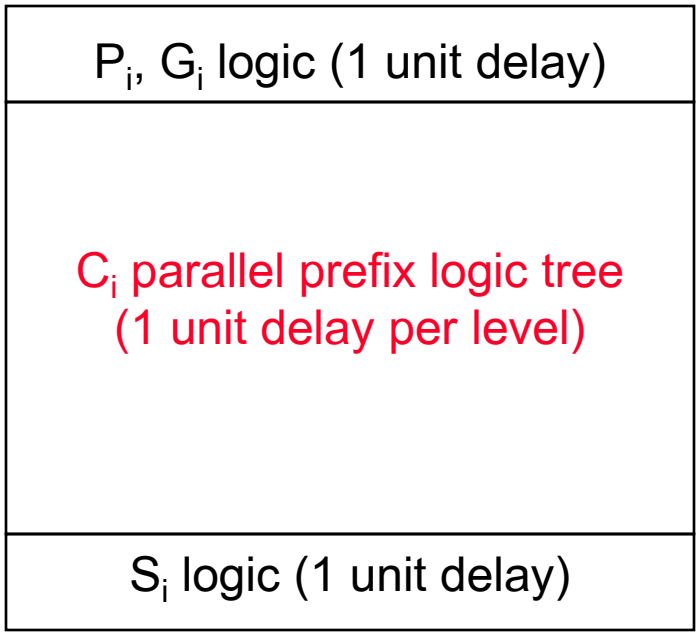


PPA General Structure

- Given P and G terms for each bit position, computing all the carries is equal to finding *all* the prefixes in parallel

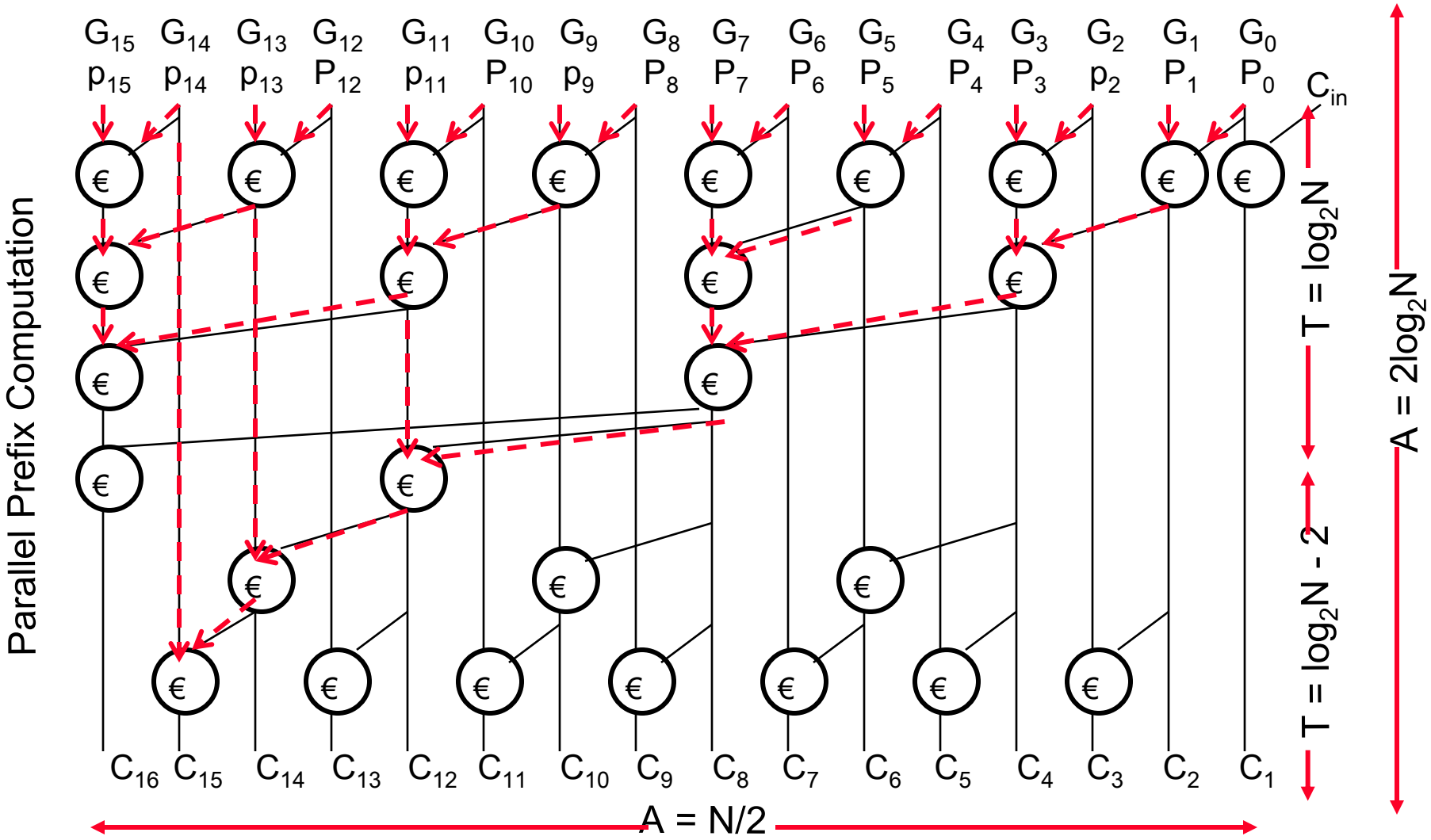
$$(G_0,P_0) \text{ € } (G_1,P_1) \text{ € } (G_2,P_2) \text{ € } \ldots \text{ € } (G_{N-2},P_{N-2}) \text{ € } (G_{N-1},P_{N-1})$$

- Since € is associative, we can group them in any order
  - but note that it is not commutative

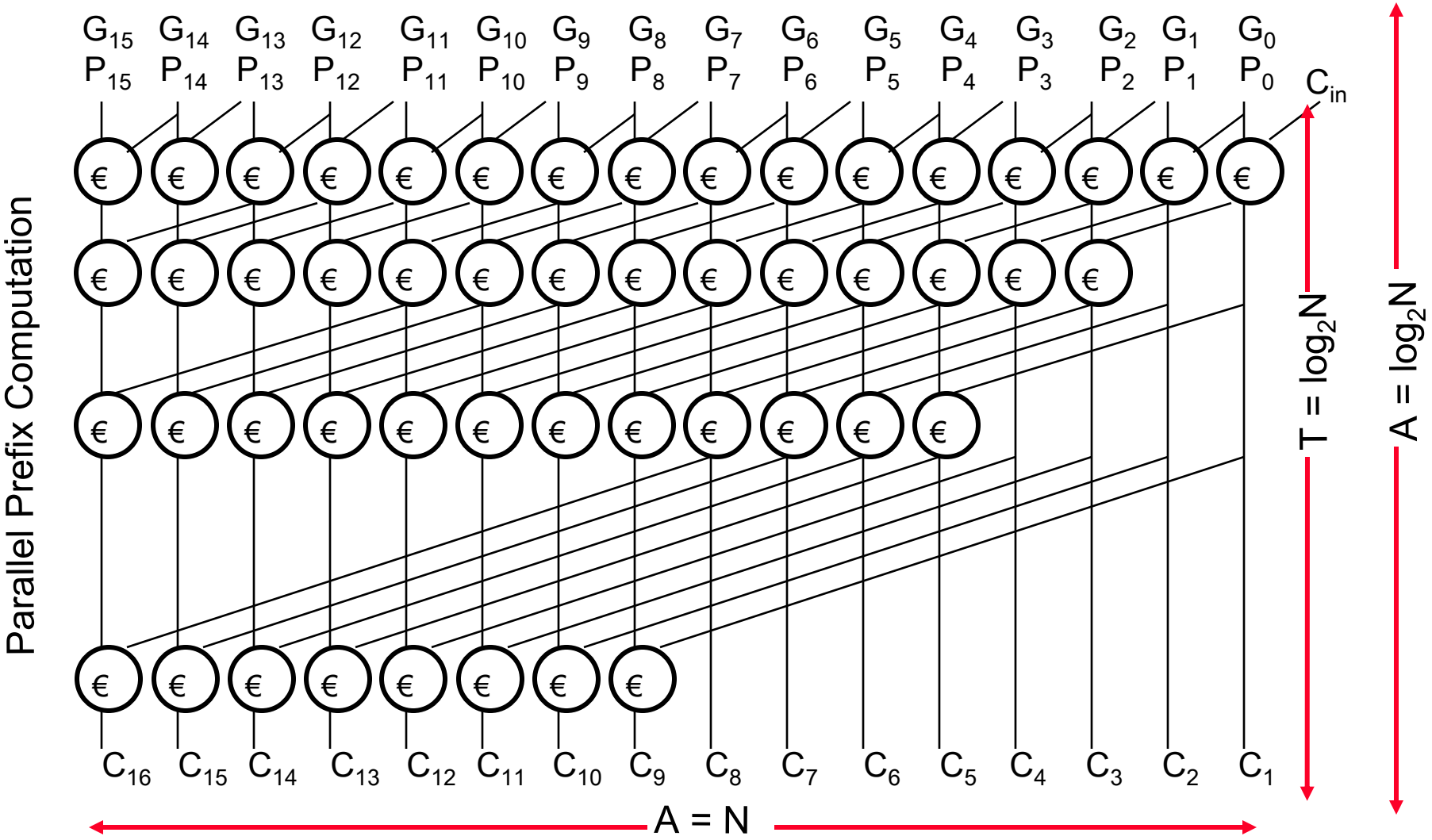


- Measures to consider
  - number of € cells
  - tree cell depth (time)
  - tree cell area
  - cell fan-in and fan-out
  - max wiring length
  - wiring congestion
  - delay path variation (glitching)

Brent-Kung PPA

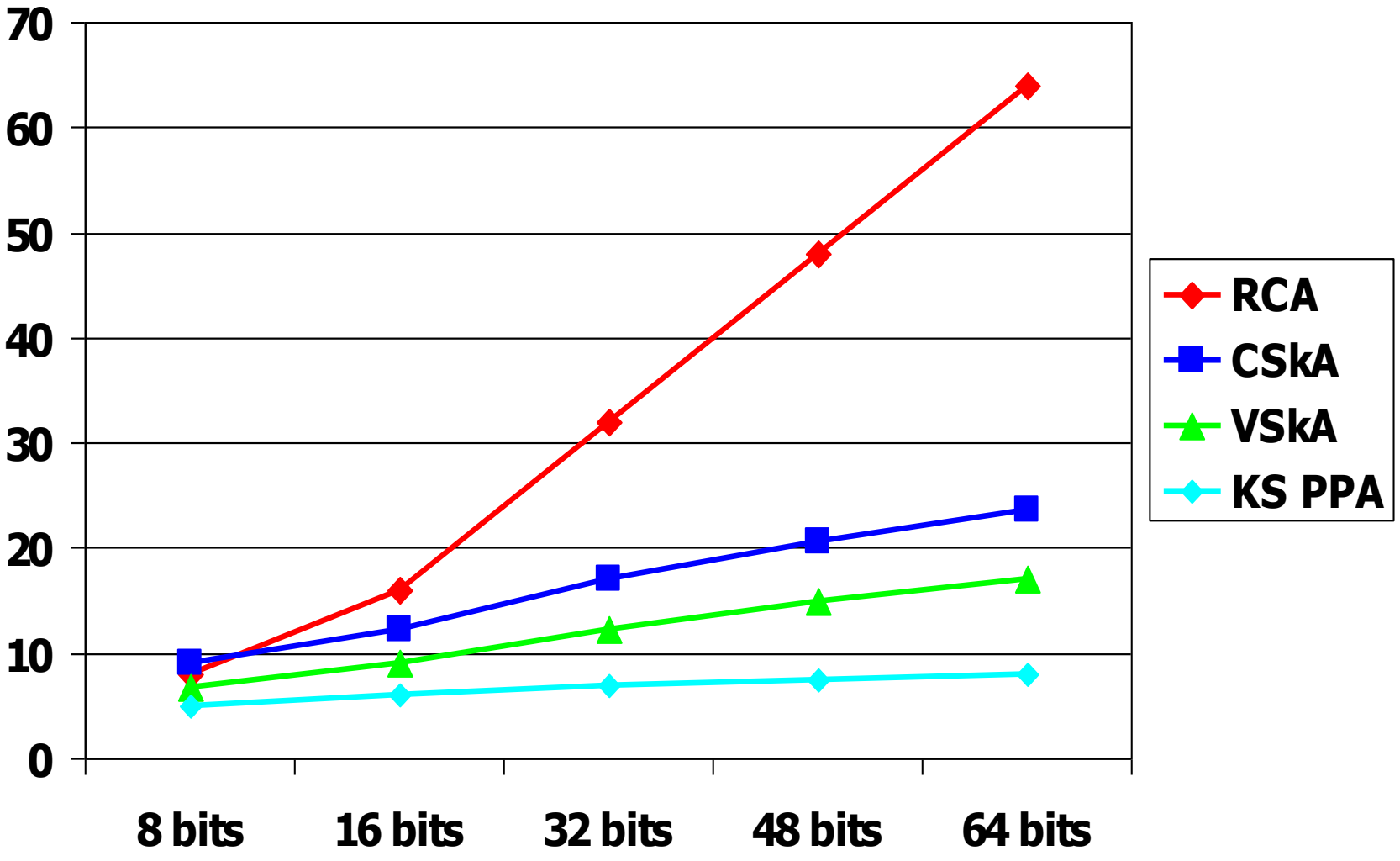


Kogge-Stone PPF Adder



$$T_{add} = t_{setup} + \log_2 N \; t_{€} + t_{sum}$$

More Adder Comparisons



Next Lecture and Reminders

- Next lecture
  - Multiplier Design
    - Reading assignment – Rabaey, et al, 11.4
- Reminders
  - Project final reports due December 5th
  - HW5 (last one!) due November 19th
  - Final grading negotiations/correction (except for the final exam) must be concluded by December 10th
  - Final exam scheduled
    - Monday, December 16th from 10:10 to noon in 118 and 121 Thomas