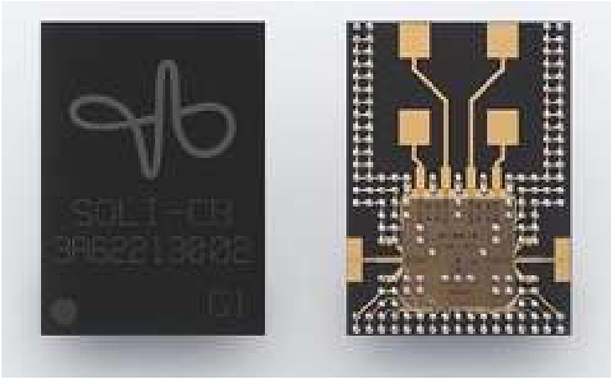
EECS151 : Introduction to Digital Design and ICs

Lecture 15 – Adders

Bora Nikolić and Sophia Shao

Infineon Flies Under the Radar on Google's Soli

Google last week rolled out its Pixel 4 smartphone, whose claim to fame is a radar-based technology that makes it the first smartphone featuring “Motion Sense” capabilities.



EE Times, 10/21/2019

Delay Optimization

- Wires contributes to delay and energy, especially in modern technology.
- We can use RC models to capture wire delay.
- Energy becomes an increasingly important optimization goal.
  - Dynamic Energy
  - Static Energy

Binary Adders

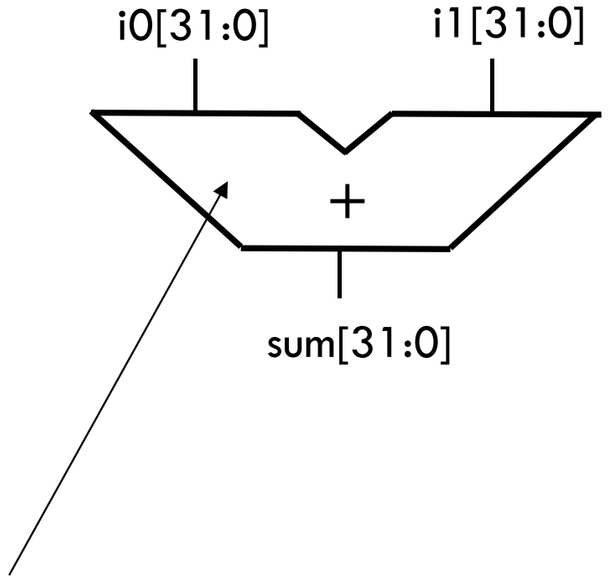
Binary Adder

- Adders

```
module add32(i0,i1,sum);
input [31:0] i0,i1;
output [31:0] sum;

assign sum = i0 + i1;

endmodule
```

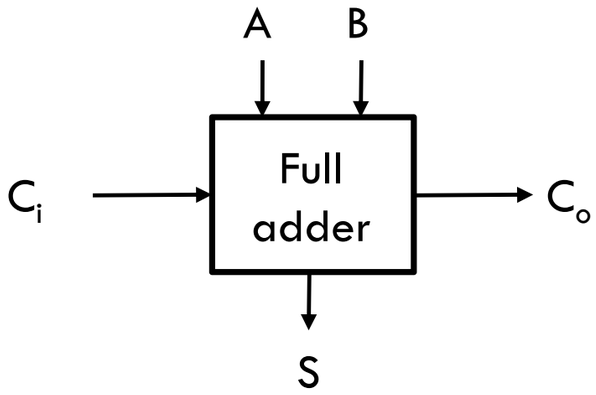


What's inside?

Depends on:

- Performance/power requirements
- Number of bits

Single-Bit Full-Adder

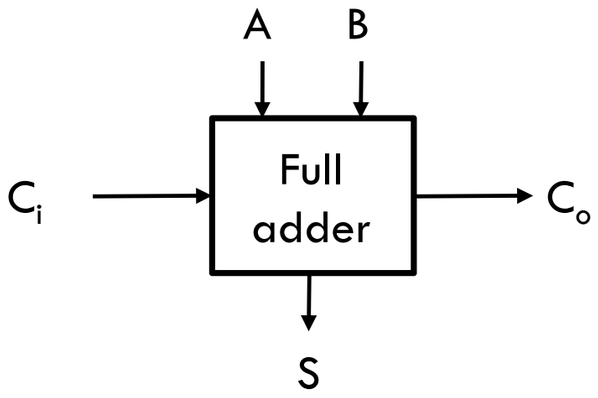


A	B	C_{in}	S	C_o	Carry Status
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Carry status = {generate, propagate, delete}

Single-Bit Full Adder

- Logic equations



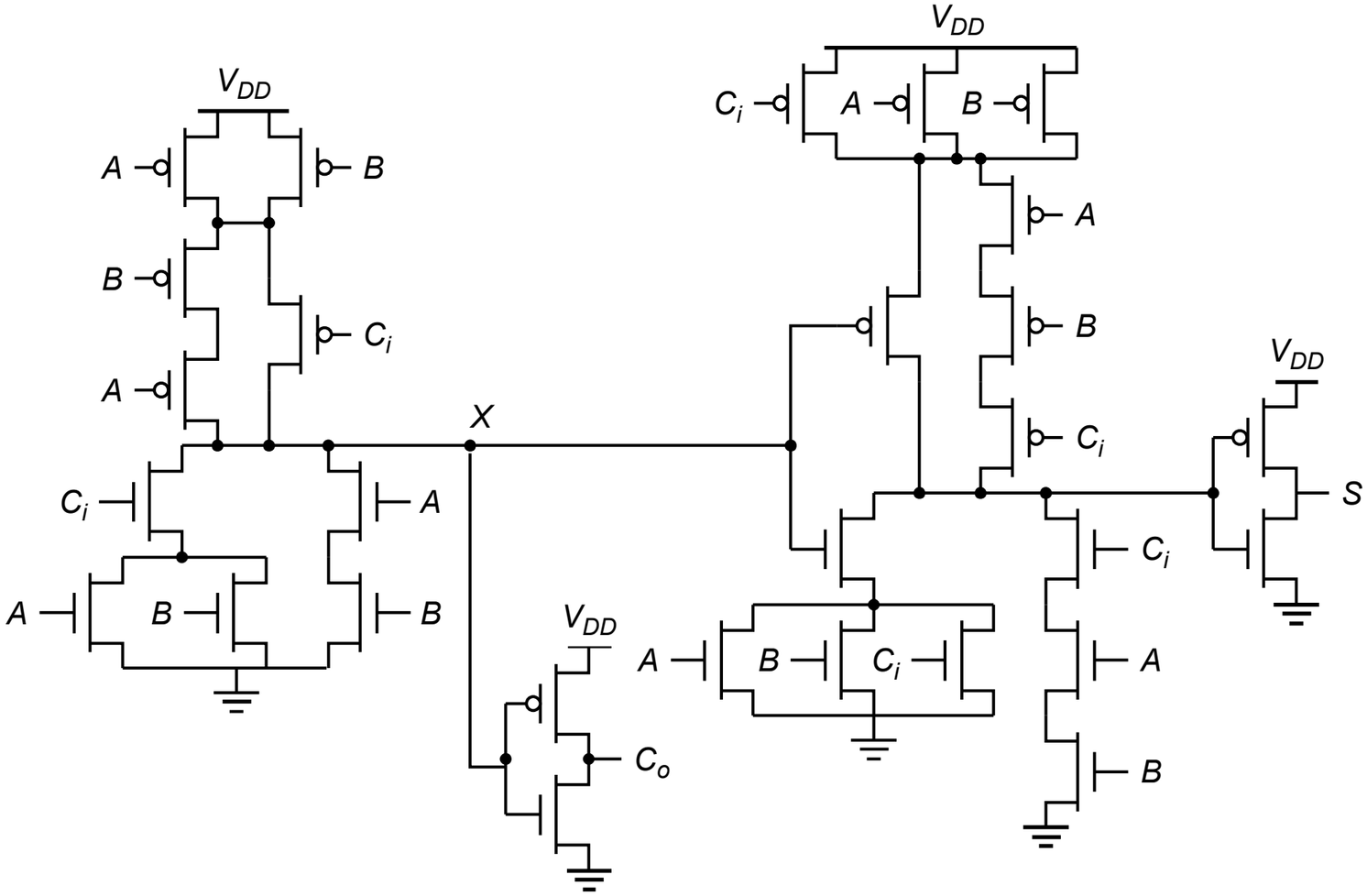
$$S = A \oplus B \oplus C_i$$

$$S = A\,\overline{B}\,\overline{C_i} + \overline{A}\,B\,\overline{C_i} + \overline{A}\,\overline{B}\,C_i + A\,B\,C_i$$

$$C_o = A\,B + B\,C_i + A\,C_i$$

Static CMOS Full Adder

- Direct mapping of logic equations



28 Transistors

Express Sum and Carry as a function of P, G, D

- Define generate, propagate and delete as functions of A, B
  - Will use two at a time

Generate (G) = AB

Propagate (P) = A + B (= A ⊕ B)

Delete = $\overline{A}\,\overline{B}$

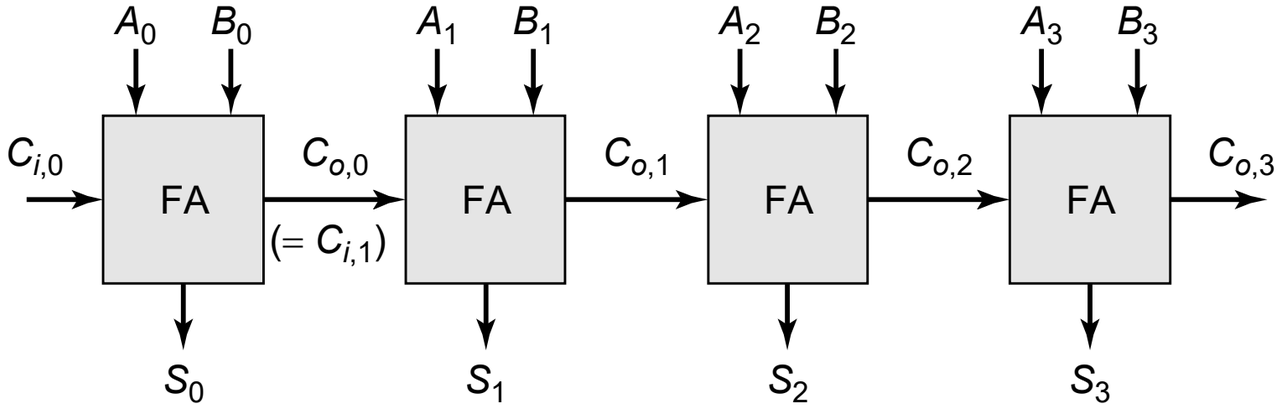
$$C_o = A\,B + B\,C_i + A\,C_i = G + P\,C_i$$

A	B	C_i	S	C_o	P	G
0	0	0	0	0	0	0
0	0	1	1	0	1	0
0	1	0	1	0	1	0
0	1	1	0	1	x	1
1	0	0	1	0	0	0
1	0	1	0	1	1	0
1	1	0	0	1	1	0
1	1	1	1	1	X	1

Can also derive expressions for C_o based on D and P

The Ripple-Carry Adder

- 4-bit adder



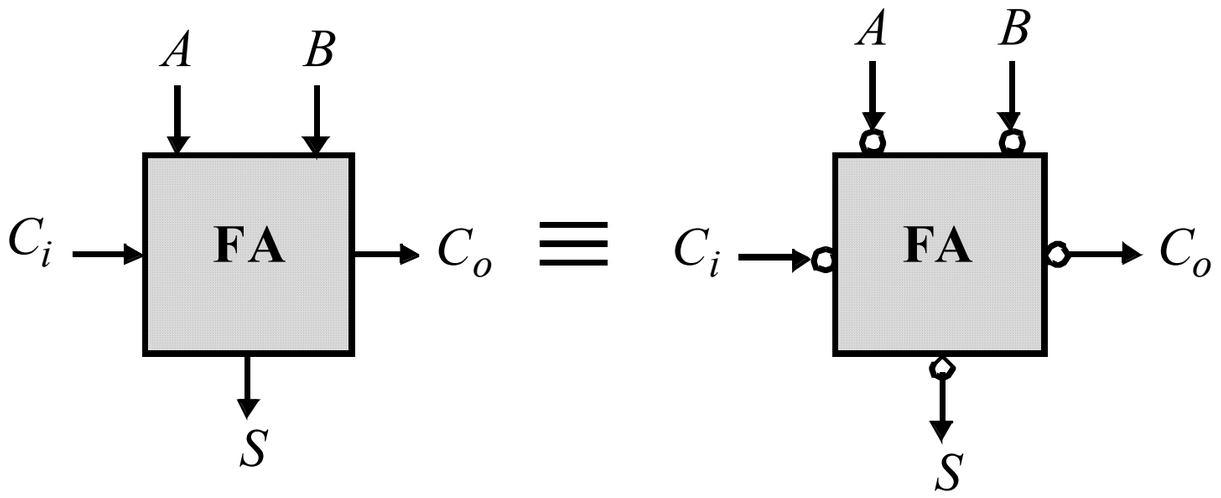
Worst case delay linear with the number of bits

$$t_d = O(N)$$

$$t_{adder} = (N\text{-}1)t_{carry} + t_{sum}$$

Goal: Make the fastest possible carry path circuit

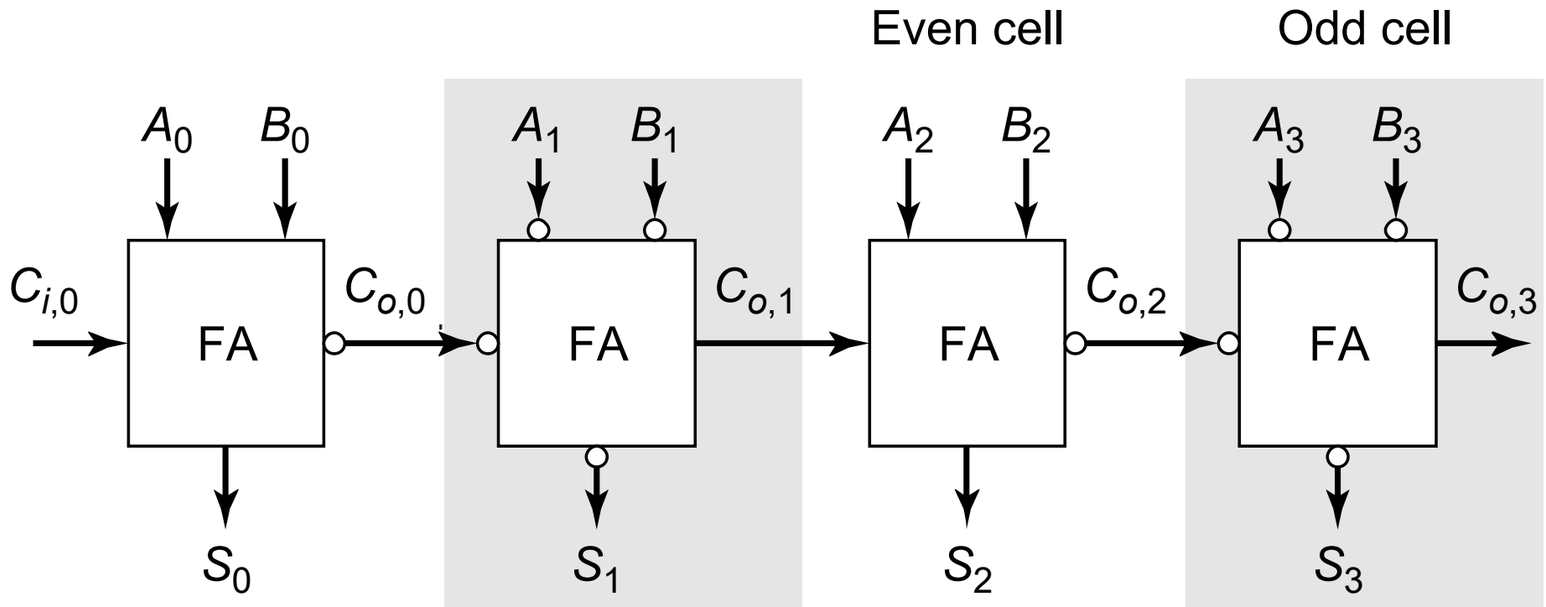
Inversion Property



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

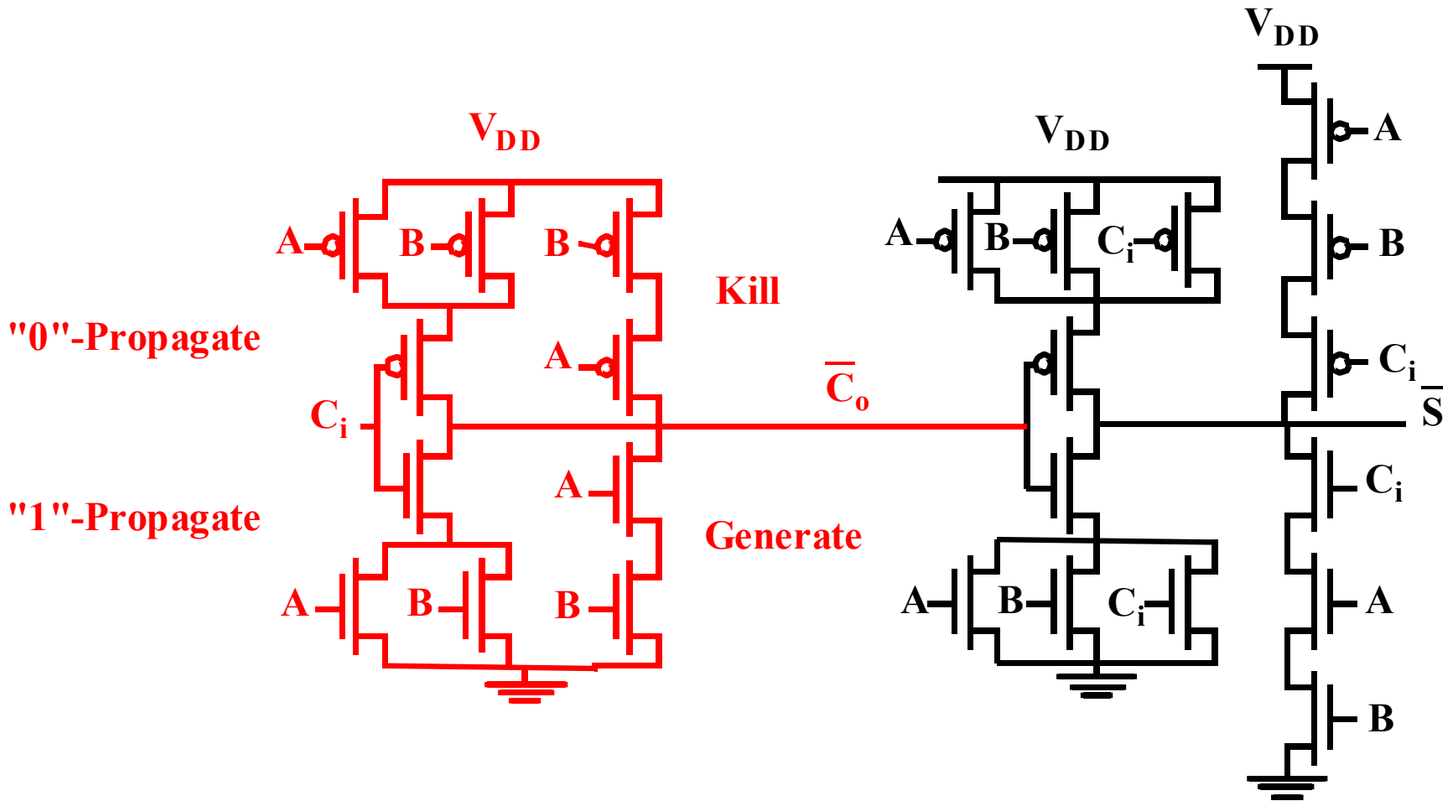
$$\overline{C_o}(A,B,C_i) = C_o(\bar{A},\bar{B},\bar{C}_i)$$

Minimize Critical Path by Reducing Inverting Stages



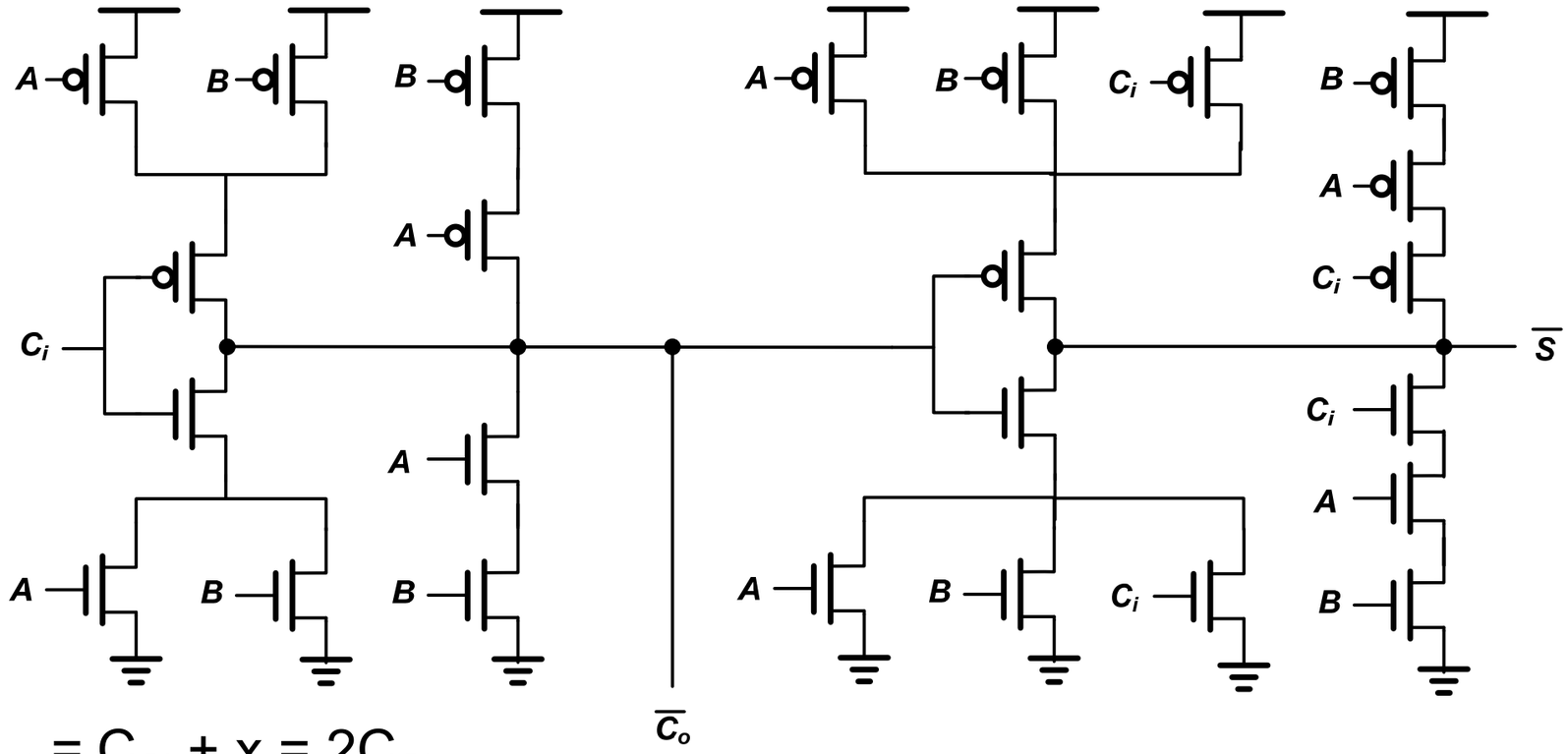
Exploit Inversion Property

A Better Structure: The Mirror Adder



24 transistors

Sizing the Mirror Adder



- $C_{load} = C_{Ci} + x = 2C_{Ci}$

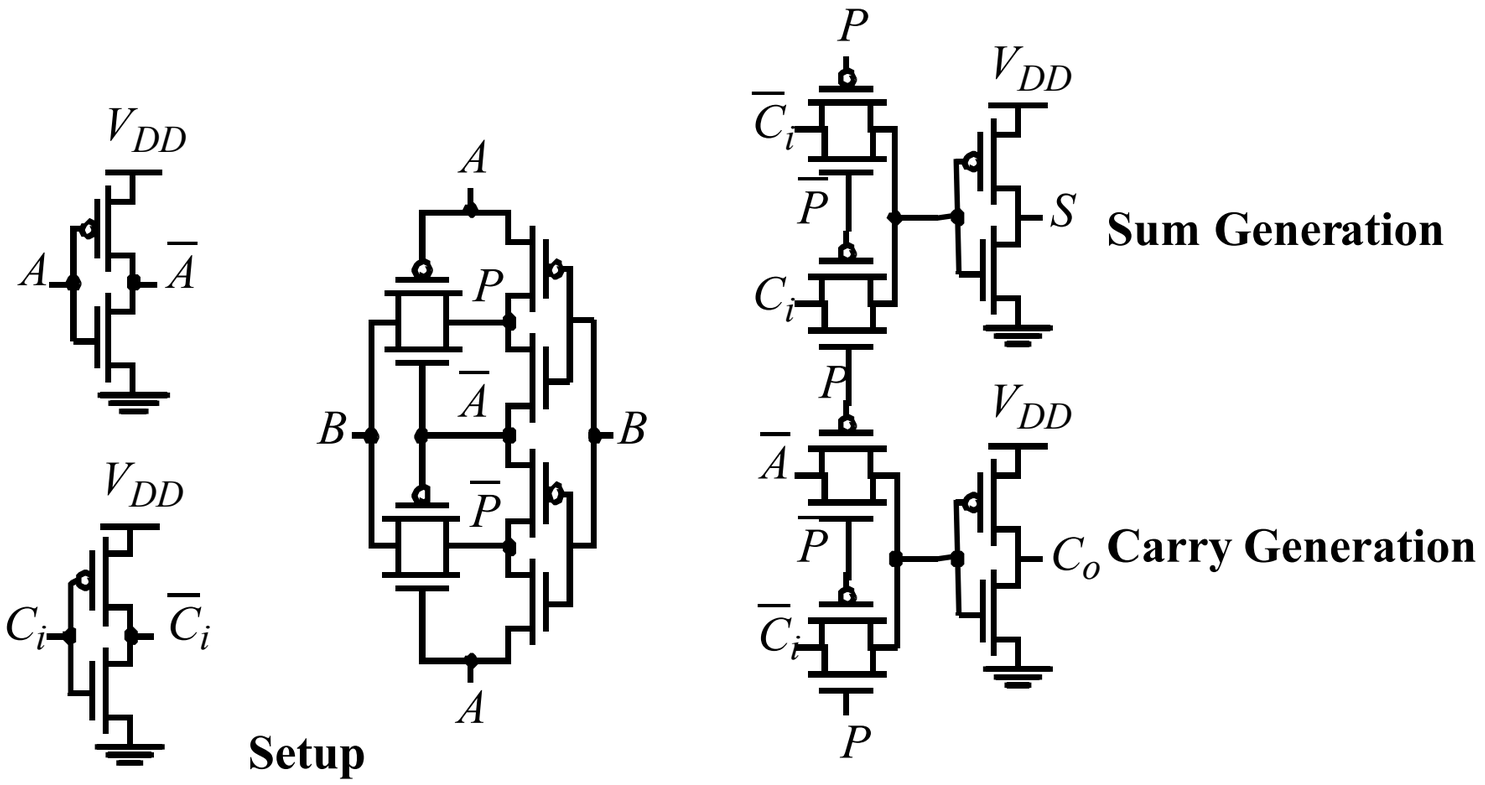
  → C_{Ci} =

- Reduce size of G and K stacks to reduce diffusion loading

The Mirror Adder

- **The NMOS and PMOS chains are completely symmetrical.**
  A maximum of two series transistors in the carry-generation stack.
- **Only the transistors in the carry stage have to be optimized for optimal speed. All transistors in the sum stage can be smaller.**
- **The transistors connected to C_i are placed closest to the output.**
- **Minimize the the capacitance at node C_o.**

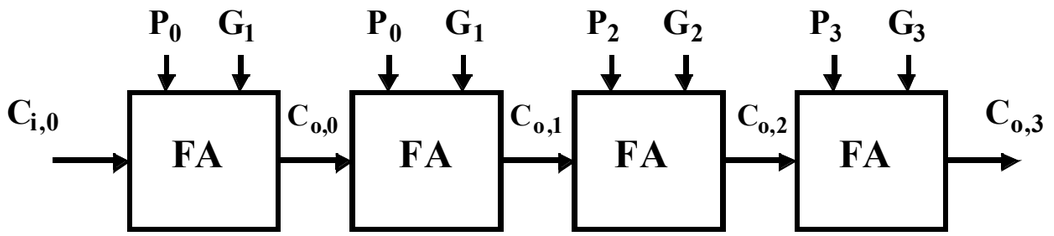
Transmission Gate Full Adder



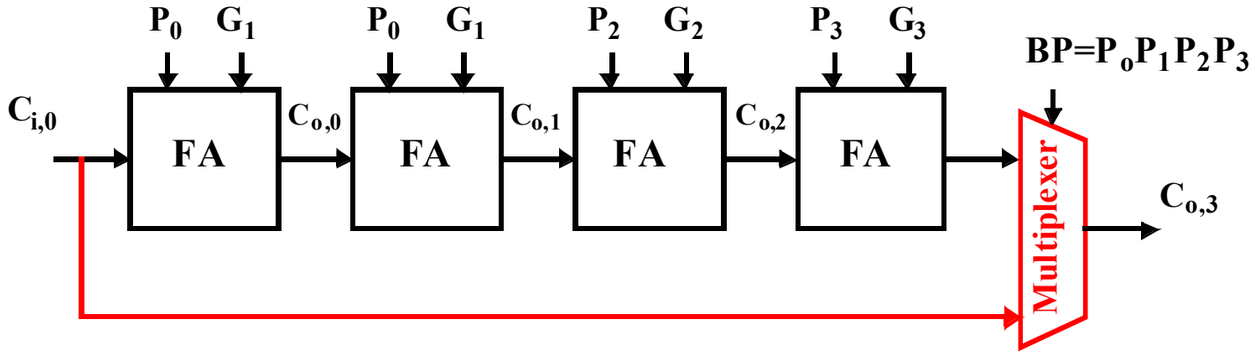
Carry Bypass Adders

Carry-Bypass Adder

- Also called ‘carry skip’

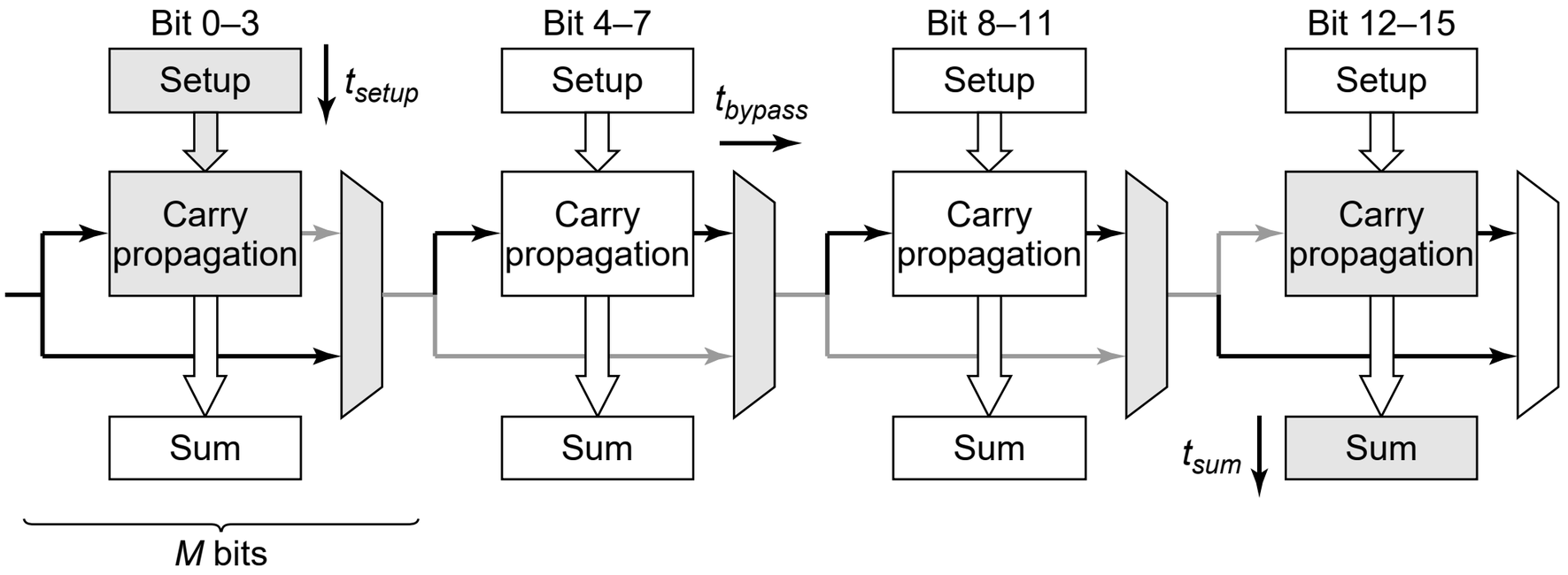


Also called Carry-Skip



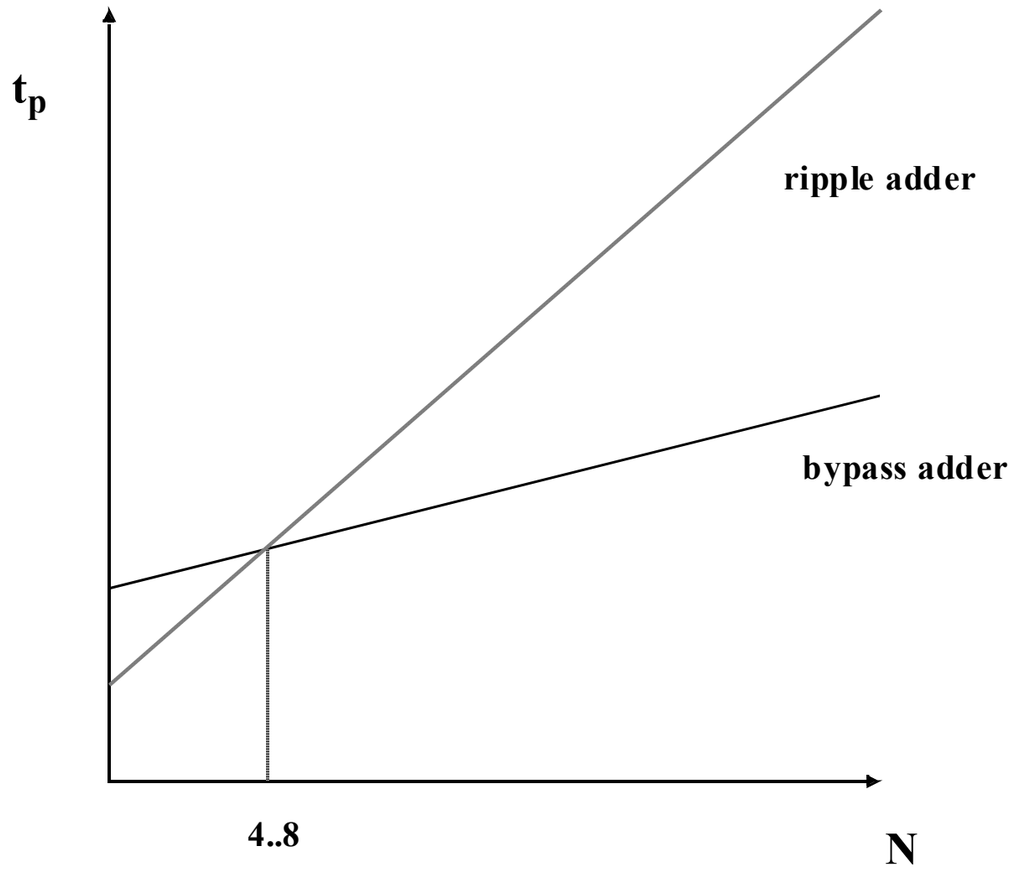
Idea: If (P0 and P1 and P2 and P3 = 1) then $C_{o3} = C_0$, else “kill” or “generate”.

Carry-Bypass Adder (cont.)



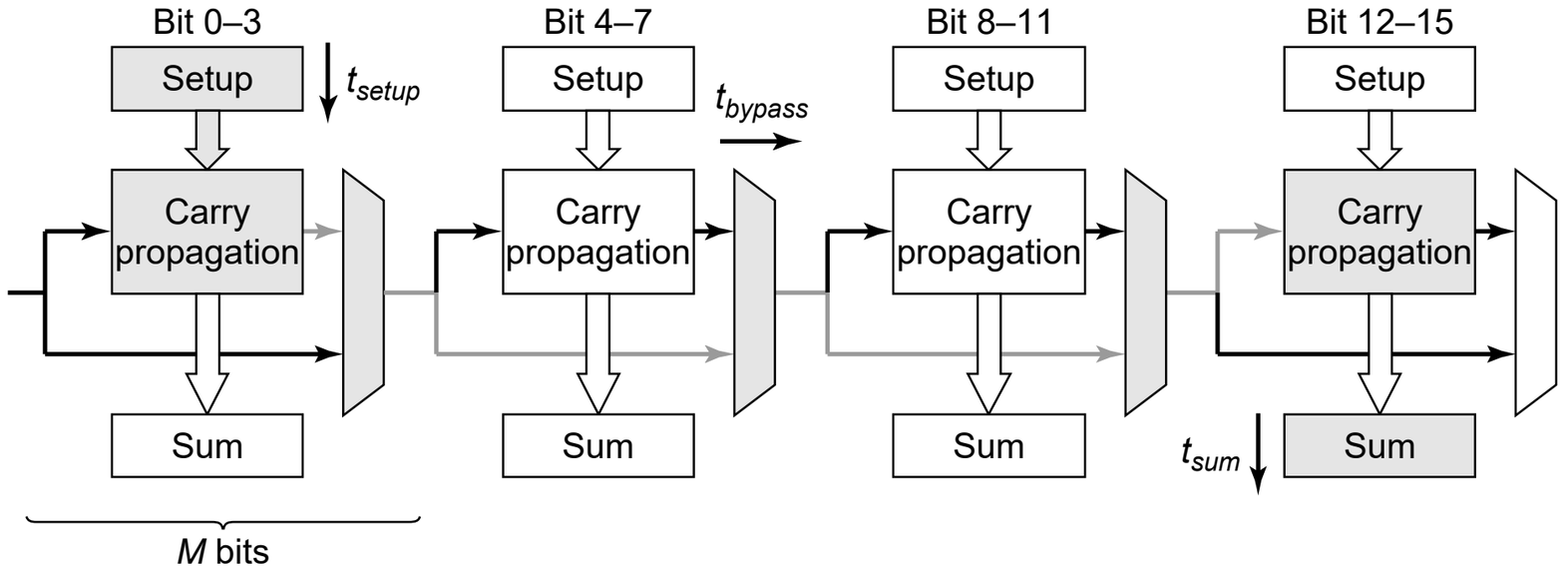
$$t_{adder} = t_{setup} + M_{tcarry} + (N/M-1)t_{bypass} + (M-1)t_{carry} + t_{sum}$$

Carry Ripple versus Carry Bypass



- Depends on technology, design constraints

To Design a Faster Carry-Bypass Adder



a) uniform groups of 4 are optimal

b) uniform groups >4 are optimal

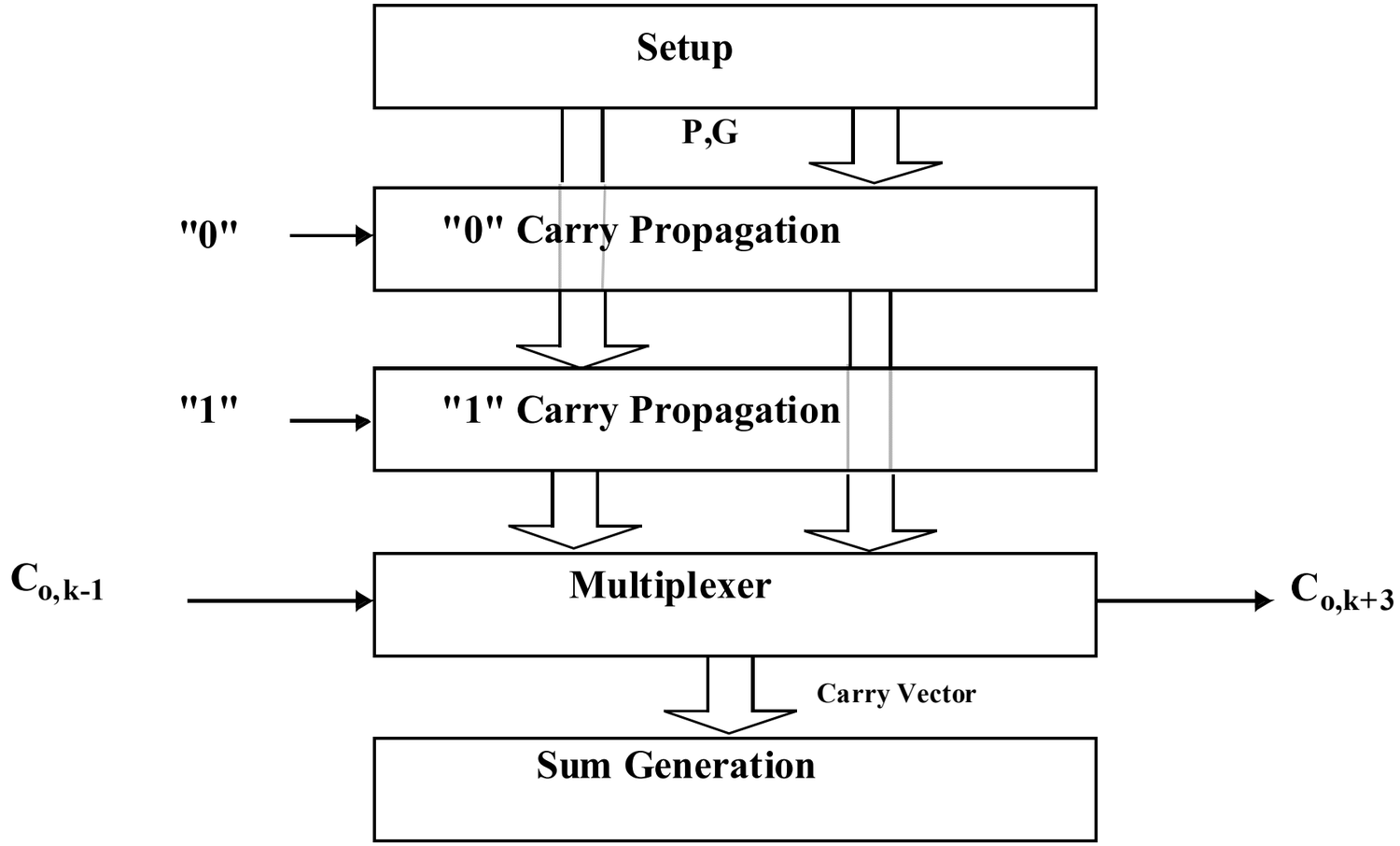
c) uniform groups >4 are optimal

d) increasing group size with higher bit position

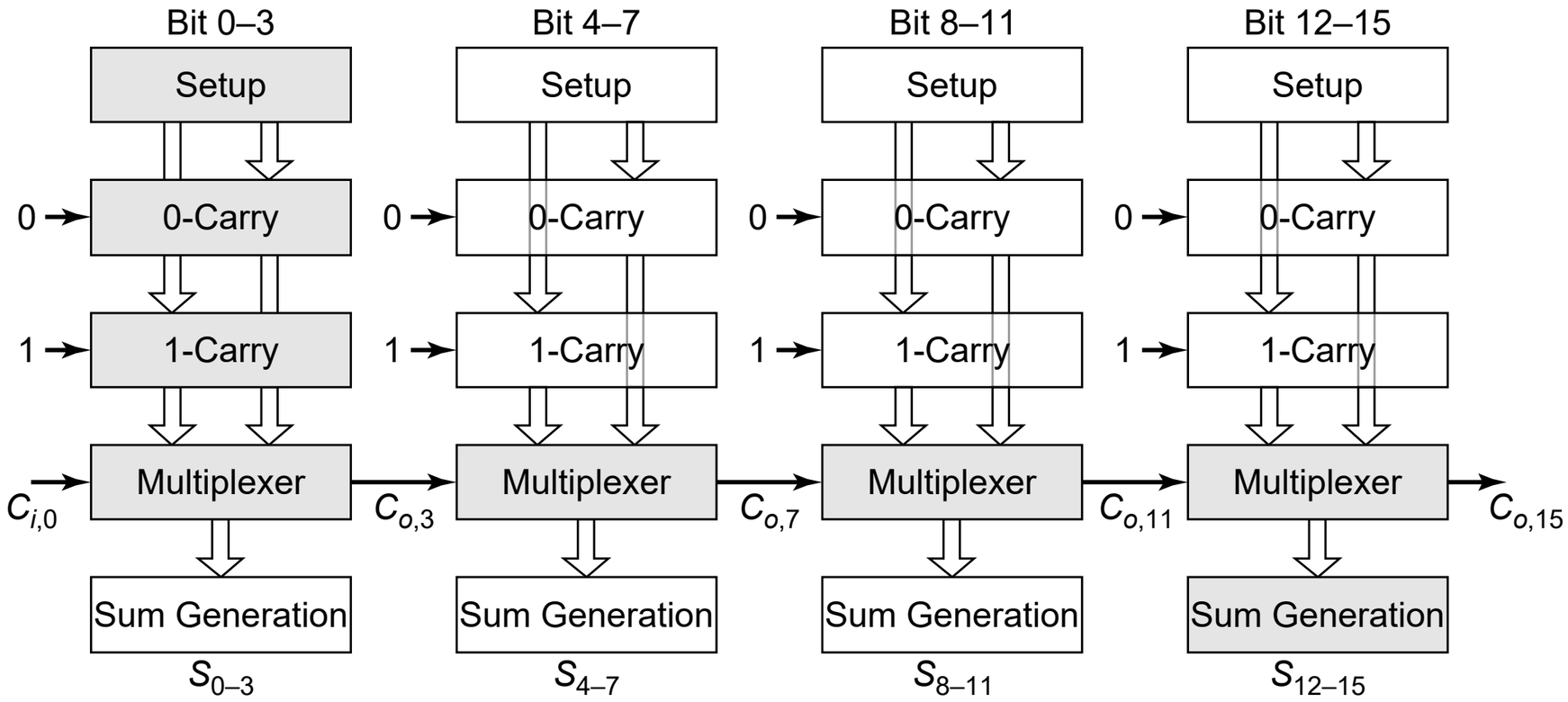
e) Wider groups around mid pit positions are optimal

Carry-Select Adders

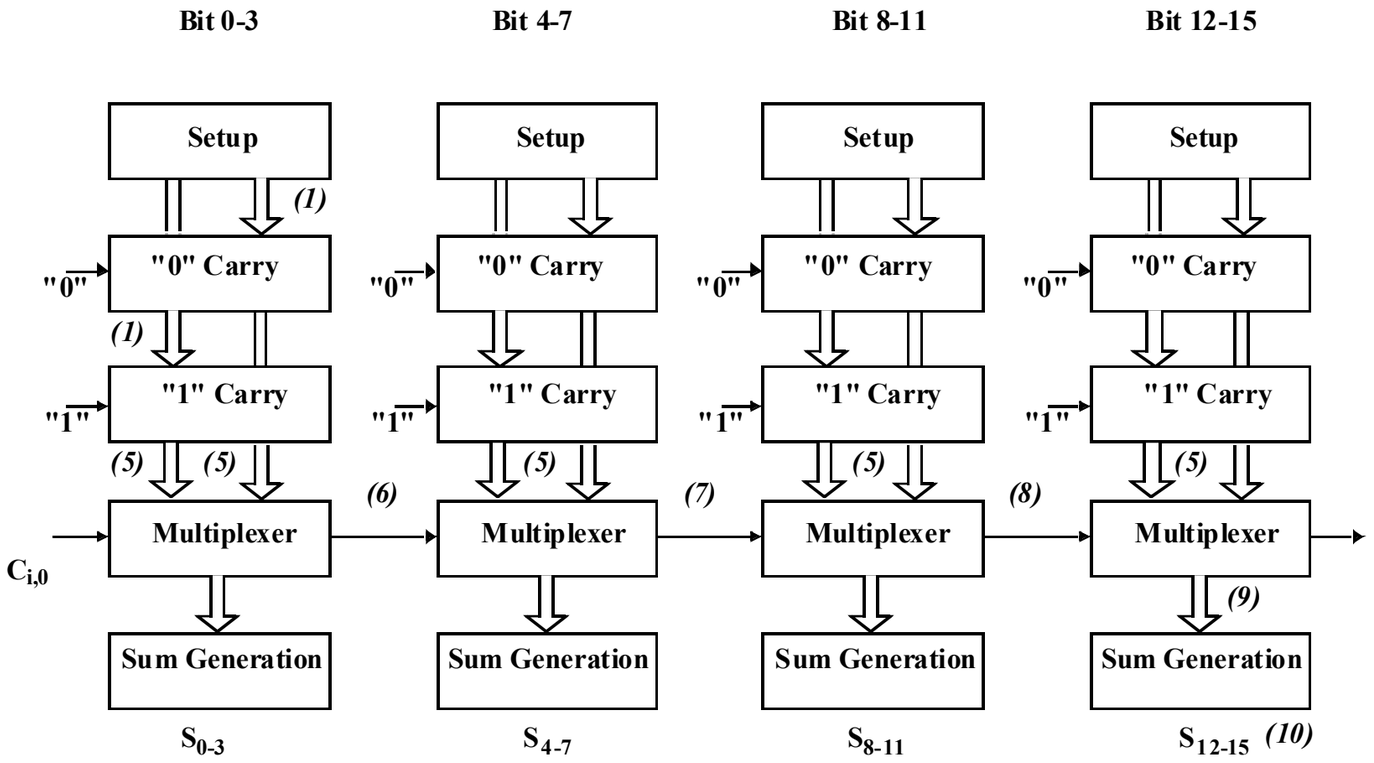
Carry-Select Adder



Carry Select Adder: Critical Path

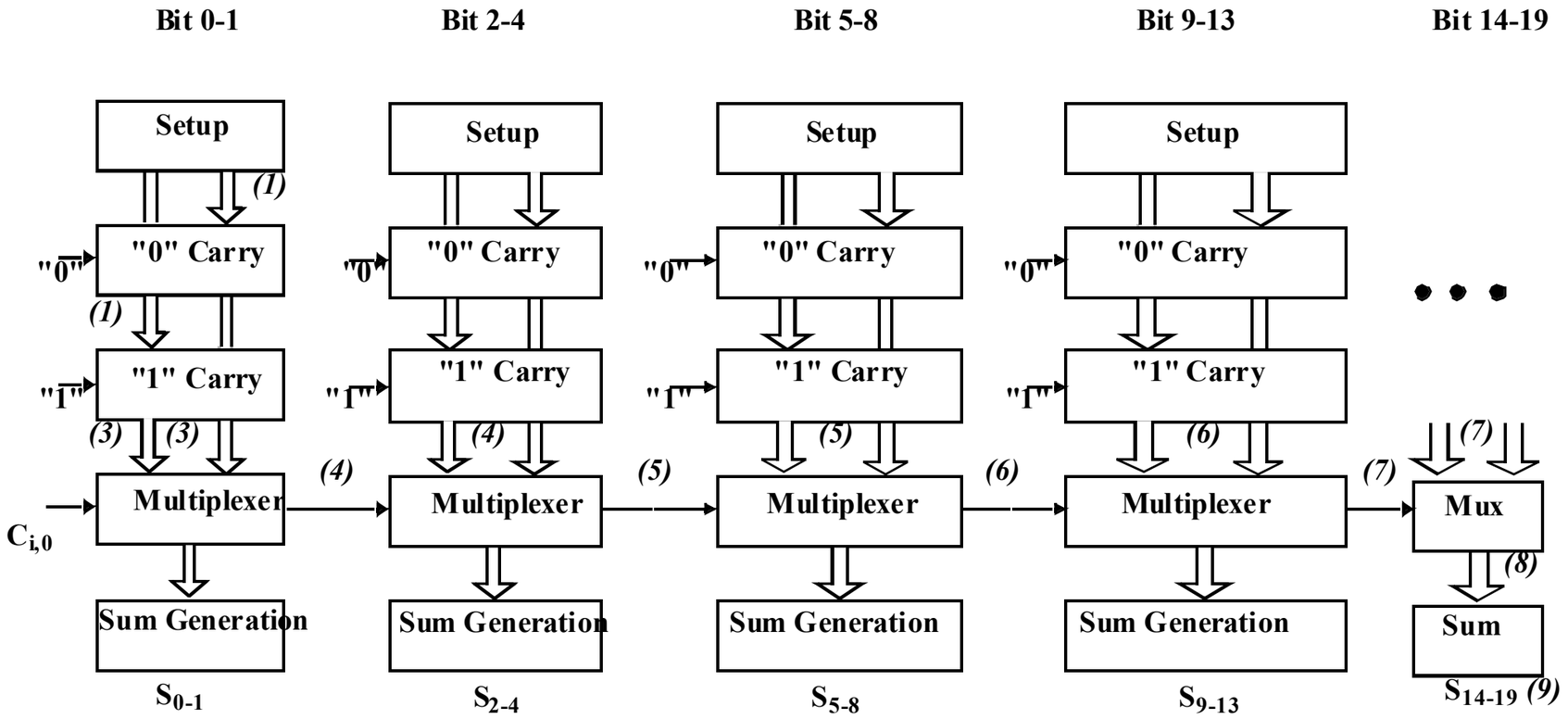


Linear Carry Select



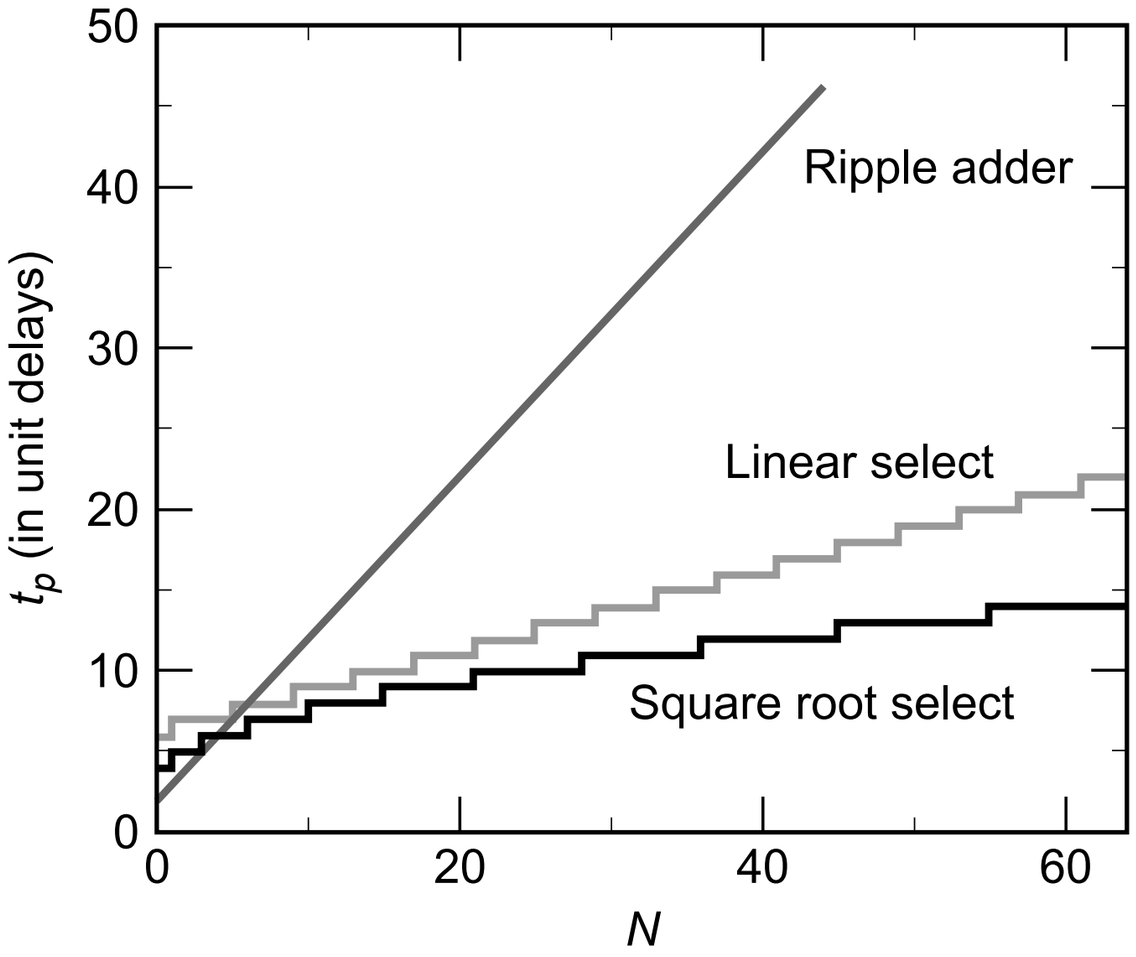
$$t_{add} = t_{setup} + \left(\frac{N}{M}\right) t_{carry} + M t_{mux} + t_{sum}$$

Square Root Carry Select



$$t_{add} = t_{setup} + P \cdot t_{carry} + (\sqrt{2N})t_{mux} + t_{sum}$$

Adder Delays - Comparison

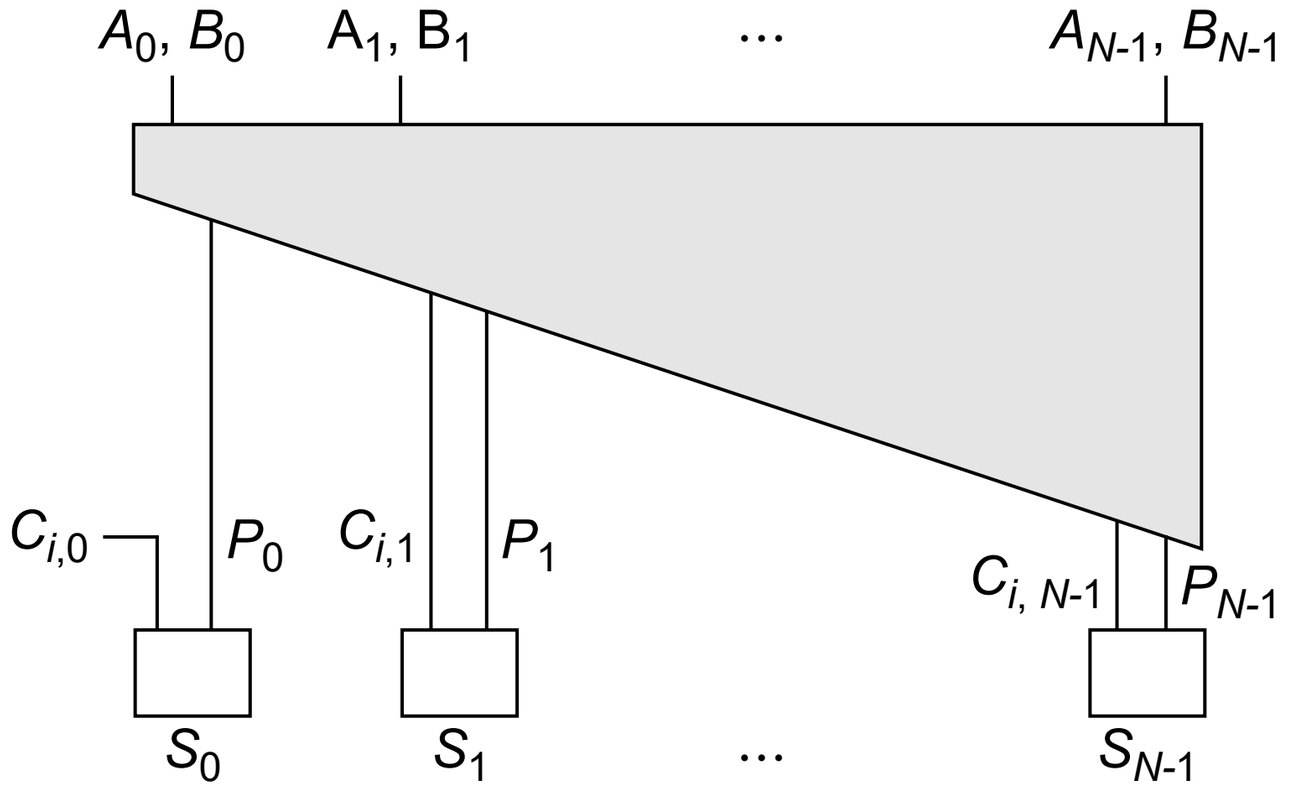


Administrivia

- Midterm 2 next week!
  - Covers material up to this Wednesday
- Power outages: We are still trying to keep lectures, homework, projects in sync
  - Please talk to us if you feel overwhelmed

Carry-Lookahead Adders

Lookahead - Basic Idea



$$C_{o,k} = f(A_k, B_k, C_{o,k-1}) = G_k + P_k C_{o,k-1}$$

Lookahead: Topology

Expanding lookahead equations:

$C_{o,1} = G_1 + P_1C_{i,1} = G_1 + P_1G_0 + P_1P_0C_{i,0}$

$C_o = A\,B + B\,C_i + A\,C_i = G + P\,C_i$

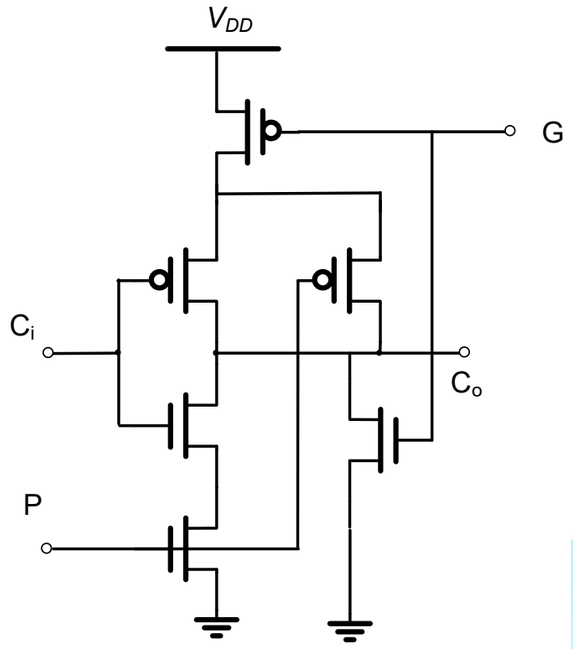
Carry at bit k:

$C_{o,k} = G_k + P_k(G_{k-1}+P_{k-1}C_{o,k-2})$

Expanding at bit k:

$C_{o,k} = G_k + P_k(G_{k-1}+P_{k-1}(\ldots+P_1(G_0+P_0C_{i,0})\ldots))$

Carry-lookahead gate grows at each bit position!



Carry Lookahead Trees

Build the carrylookahead tree as a hierarchy of gates

$C_{o,0} = G_0 + P_0C_{i,0}$

$C_{o,1} = G_1 + P_1C_{i,1} = G_1 + P_1G_0 + P_1P_0C_{i,0}$

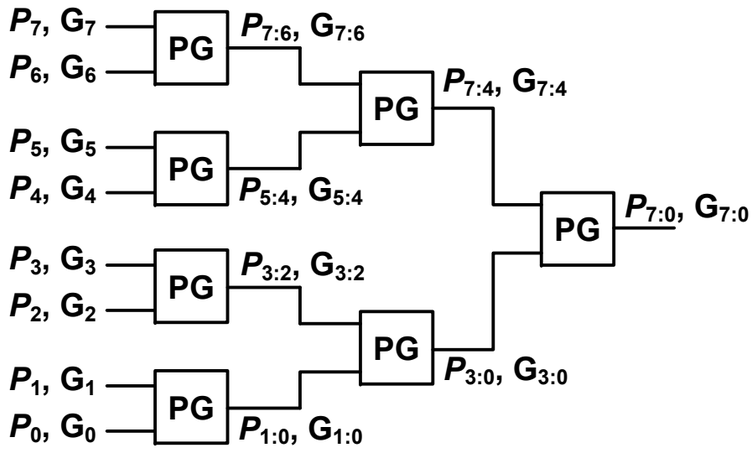
$C_{o,2} = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_{i,0}$
$= (G_2 + P_2G_1) + (P_2P_1)(G_0 + P_0C_{i,0}) = G_{2:1} + P_{2:1}C_{O,0}$

Can continue building the tree hierarchically.

Logarithmic (Tree) Adders – Idea

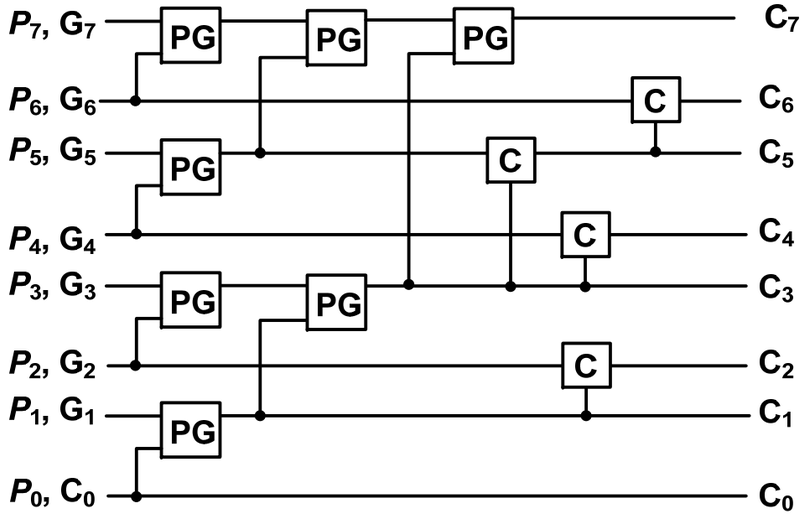
- "Look ahead" across groups of multiple bits to figure out the carry
  - Example with two bit groups:

    $P_{1:0} = P_1 \cdot P_0$, $G_{1:0} = G_1 + P_1 \cdot G_0$, → $C_{out1} = G_{1:0} + P_{1:0} \cdot C_{0,in}$
- Combine these groups in a tree structure:
  - Delay is now ~$\log_2(N)$



Rest of the Tree

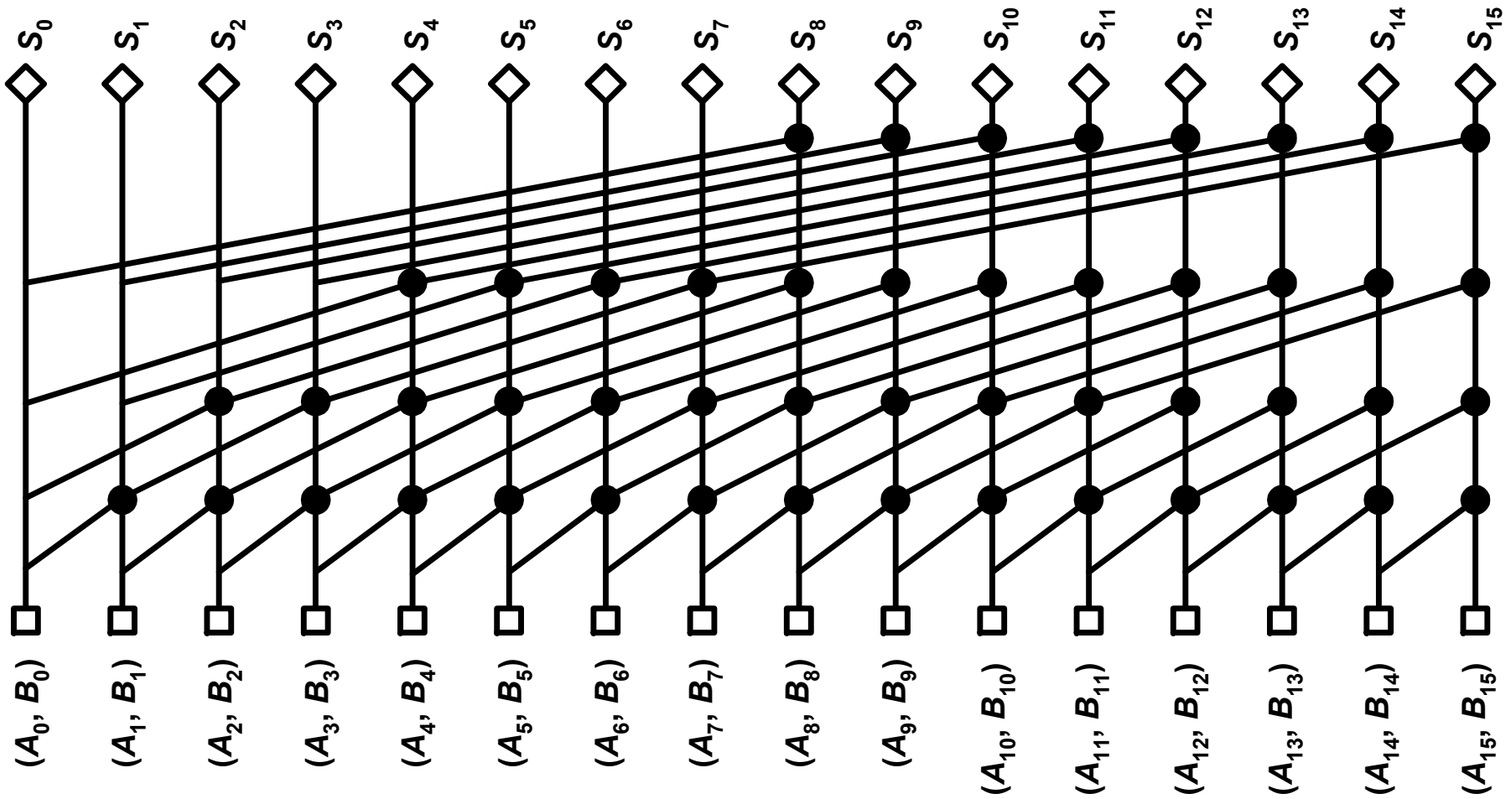
- Previous picture shows only half of the algorithm
  - Need to generate carries at individual bit positions too



Many Kinds of Tree Adders

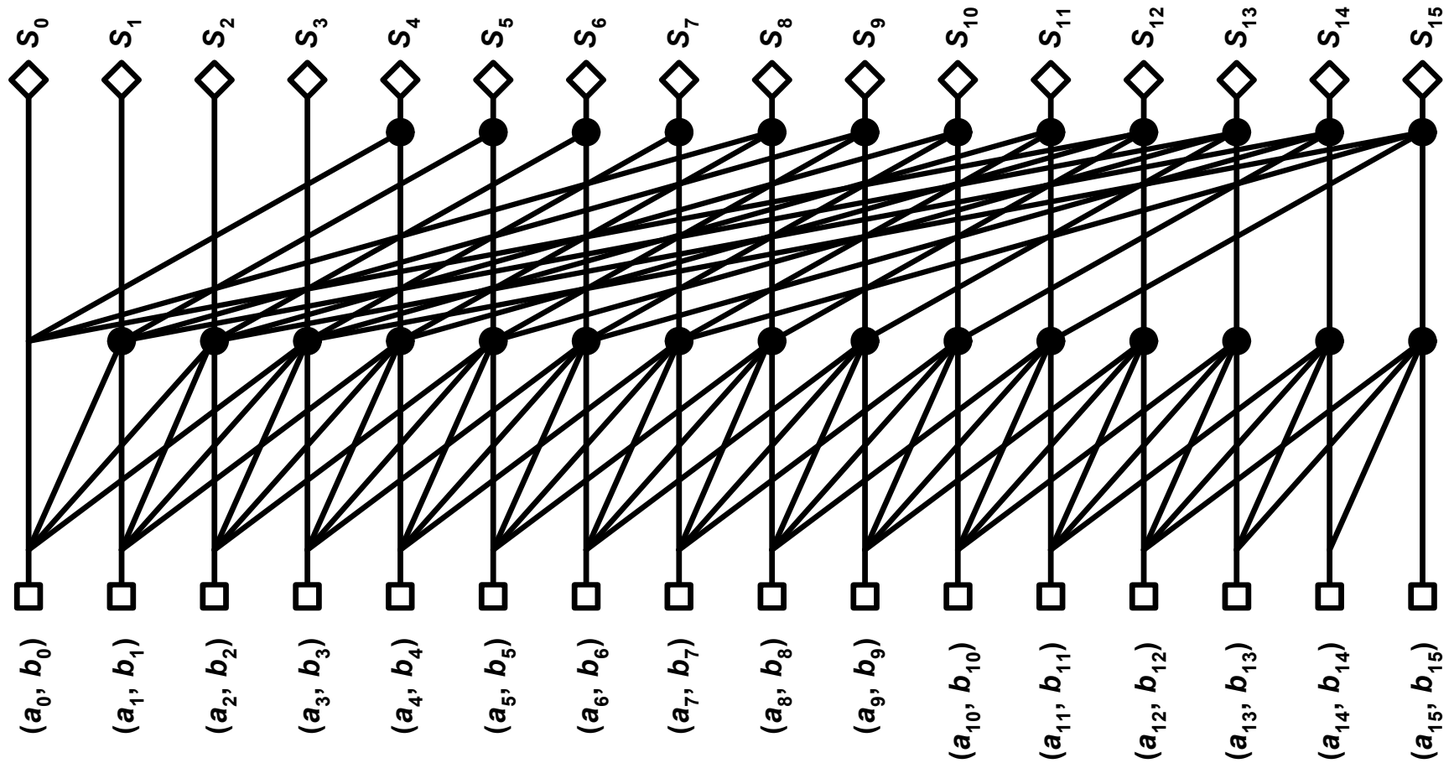
- Many ways to construct these tree (or “carry lookahead”) adders
  - Many of these variations named after the people who first came up with them
- Most of these vary three basic parameters:
  - Radix: how many bits are combined in each PG gate
    - Previous example was radix 2; often go up to radix 4
  - Tree depth: stages of logic to the final carry. Must be at least $\log_{Radix}(N)$
  - Sparseness

Tree Adders



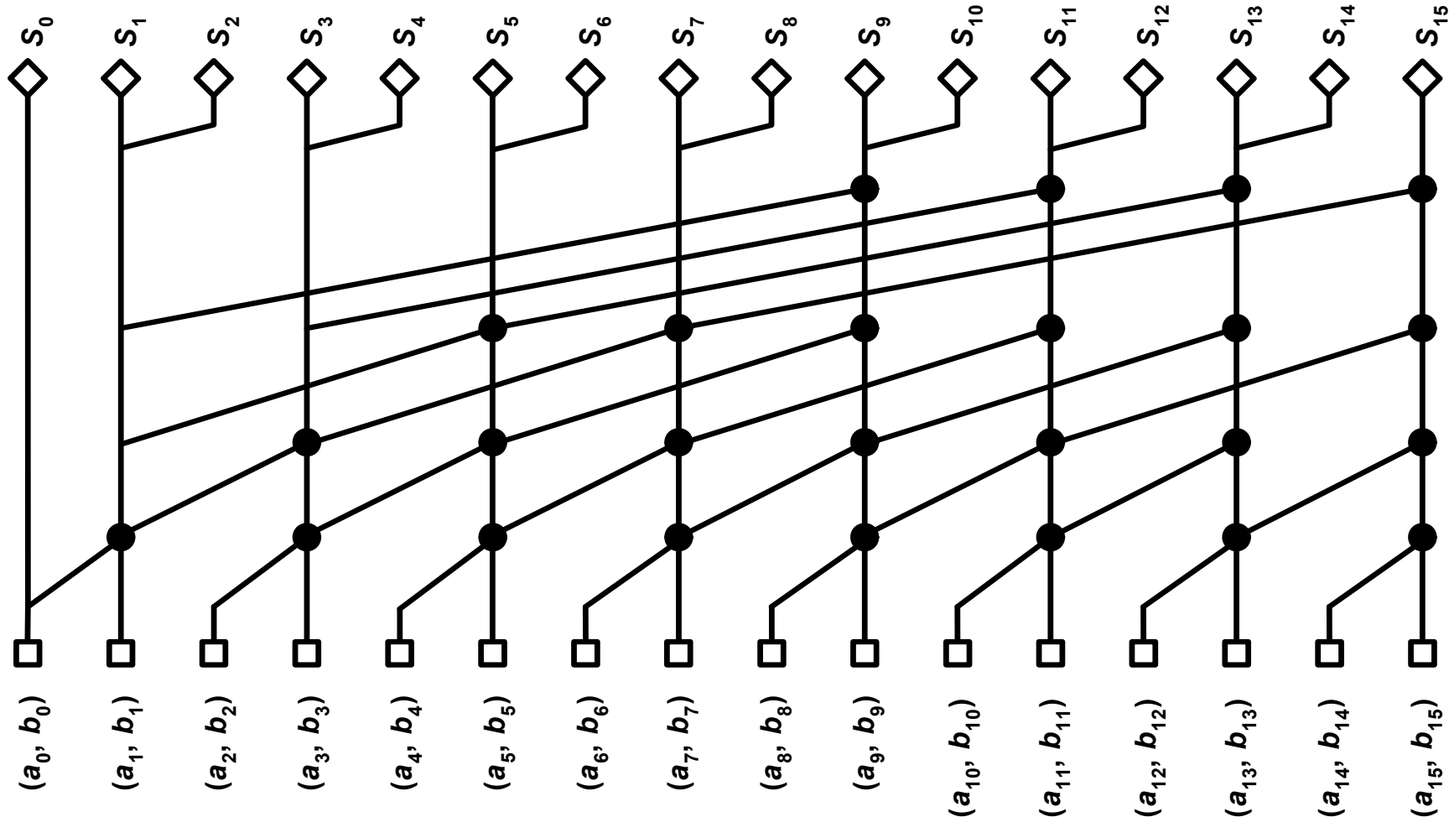
16-bit radix-2 Kogge-Stone tree

Tree Adders



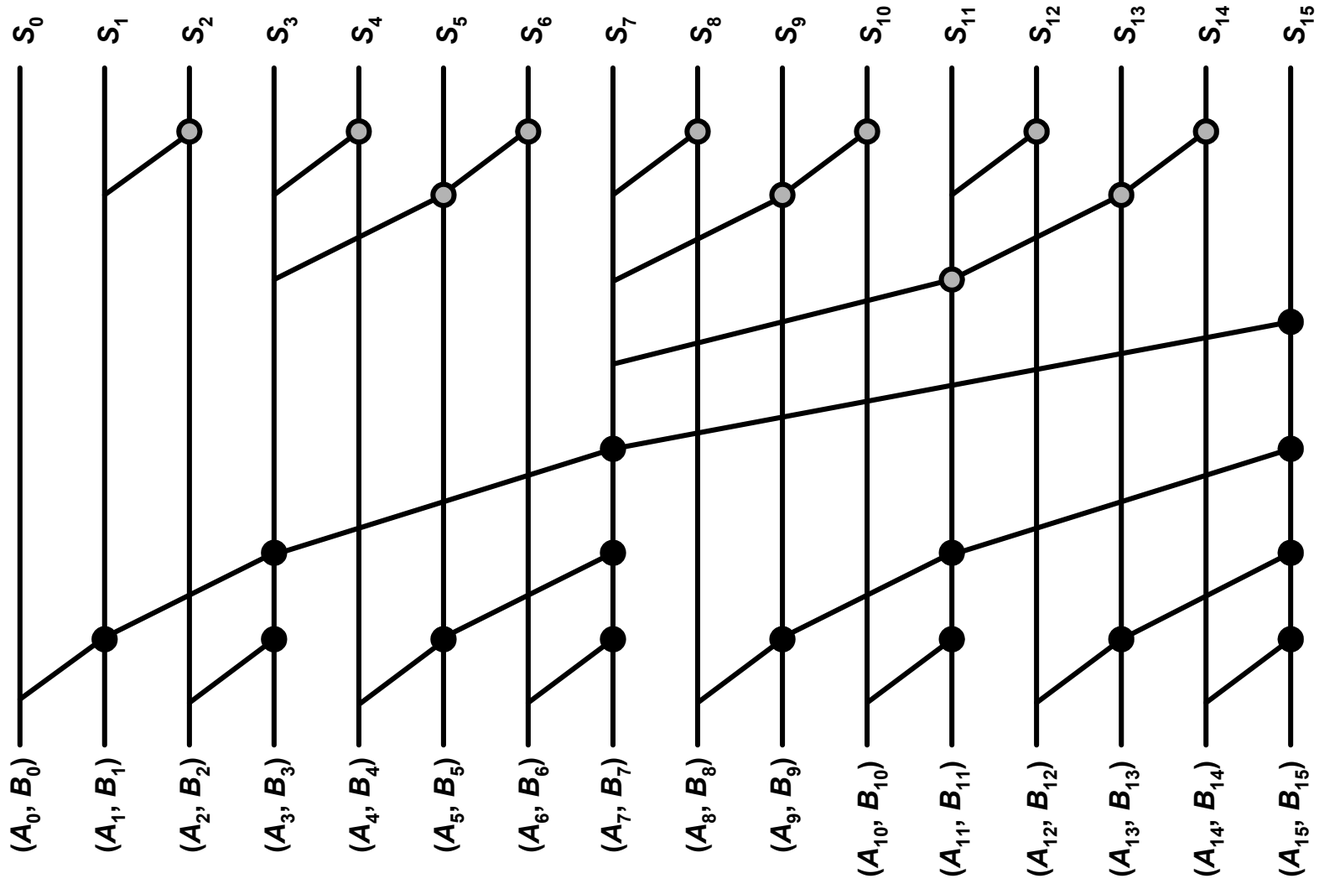
16-bit radix-4 Kogge-Stone Tree

Sparse Trees



16-bit radix-2 sparse tree with sparseness of 2

Tree Adders



Brent-Kung Tree

Summary

- Binary adders are a common building block of digital systems
- Carry is in the critical path
- Carry-bypass, carry-select are usually faster than ripple-carry for lengths > 8
- Carry-lookahead, O(~logN) is often the fastest adder with N > 16