EECS151/251A
Introduction to Digital Design and ICs

Lecture 19: Energy & Adders

Sophia Shao

ACM TURING AWARD HONORS JACK J. DONGARRA FOR PIONEERING CONCEPTS AND METHODS WHICH HAVE RESULTED IN WORLD-CHANGING COMPUTATIONS

Dongarra's Algorithms and Software Fueled the Growth of High-Performance Computing and Had Significant Impacts in Many Areas of Computational Science from AI to Computer Graphics

https://amturing.acm.org/

Summary

- Wire also contributes to delay, especially in modern technology.
- We can use RC model to capture wire delay as well.
- Energy becomes an increasingly important optimization goal.
  - Dynamic Energy
  - Static Energy

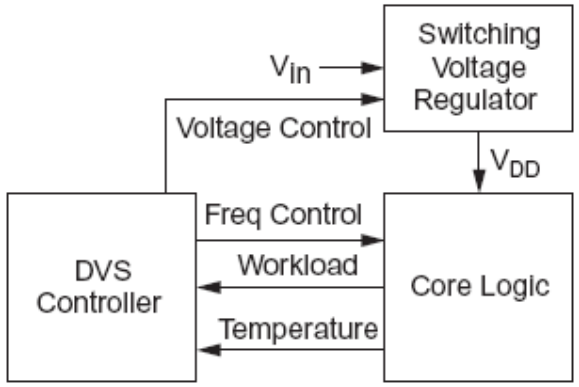
- Energy
  - Overview
  - Dynamic & Static
- Adders
  - Single-Bit Full Adder
  - Ripple-Carry Adder
  - Carry-Bypass Adder
  - Carry-Lookahead Adder

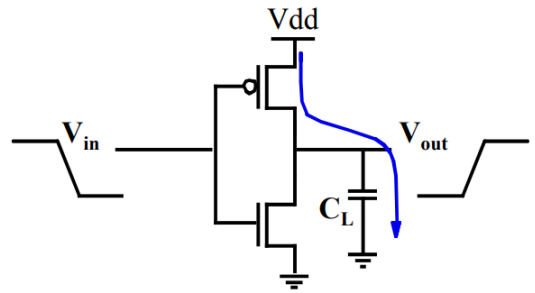
Where does power go in CMOS?

- Dynamic Power Consumption
  - Charging and discharging capacitors
  - Short-Circuit Currents
    - Short-circuit path between supply rails during switching
- Leakage (Static) Power
  - Leaky transistors

#1: Dynamic Power Consumption

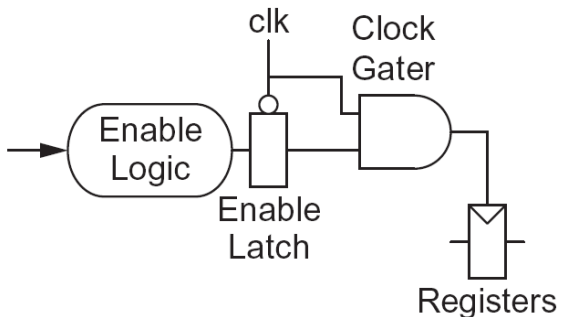
- Voltage and frequency scaling (lower Vdd, f)





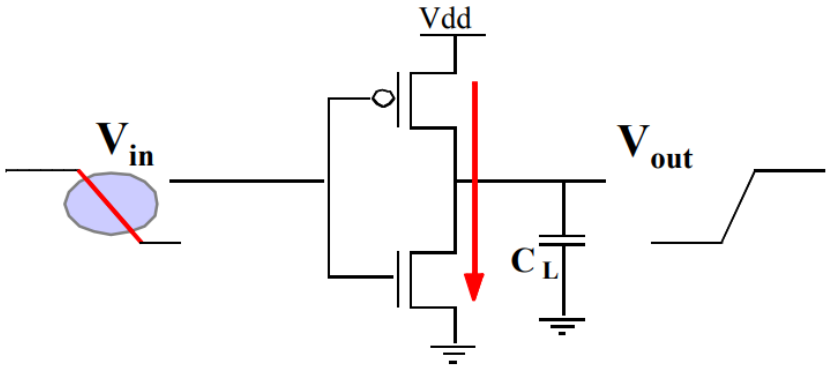
$$Power = \alpha * C_L * V_{dd}^2 * f_{clk}$$

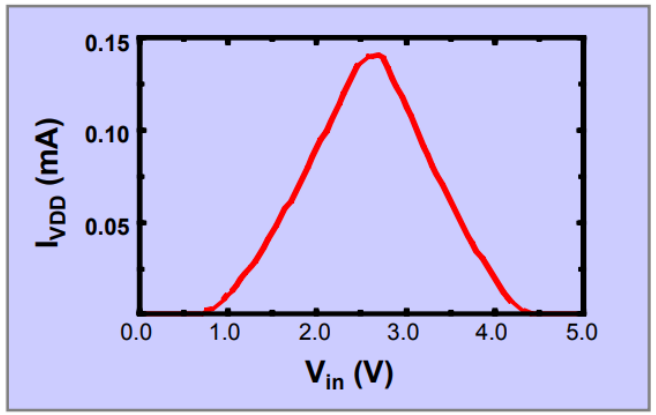
- Reduce capacitance (lower C_L)
  - Gate: minimize device sizes w/o significantly hurting perf.
  - Wire: shorten long wires
- Reduce activity factor (lower α)
  - Clock gating



#2: Short-circuit Current

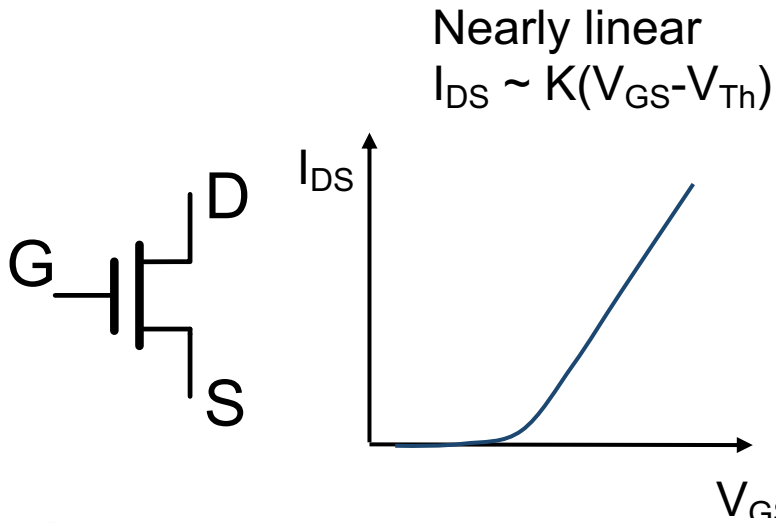
- During transition, both PUD and PDN are on.
- Similar to dynamic power, related to switching activity.
- Typically small, <10% of total power consumption
- Becomes less important in advanced technology
  - Threshold voltages do not scale as fast as supply voltage.
  - If $V_{dd} < V_{thn} + |V_{thp}|$, no short circuit.

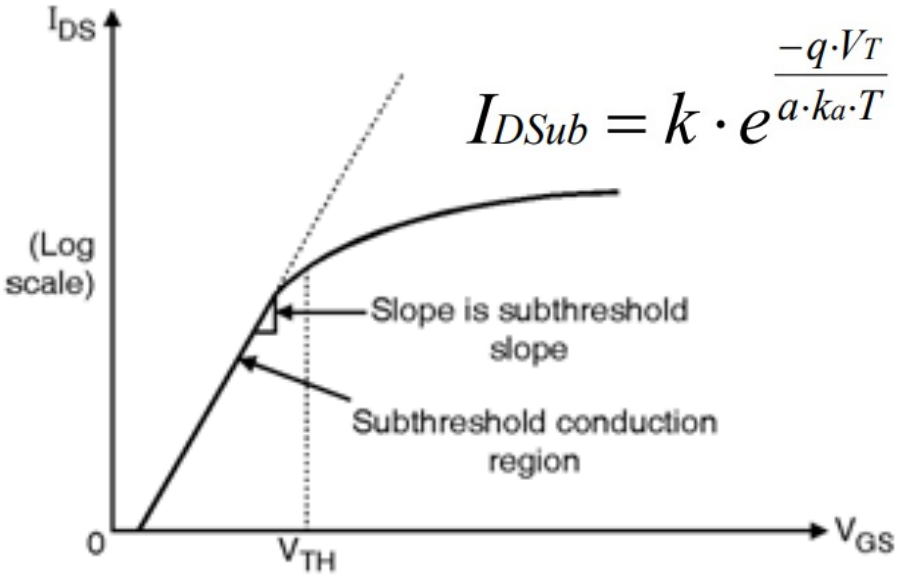




#3 Leakage (Static) Power

- Power is consumed with a chip is not switching.
  - "Off" is not really off.
- Subthreshold leakage currents grow exponentially with increases in temperature and decreases in threshold voltage.
  - But threshold voltage scaling is key to circuit performance.
- To reduce leakage power
  - Power gating: Turn OFF power to blocks when they are idle to save leakage
  - Use high-Vt cells.





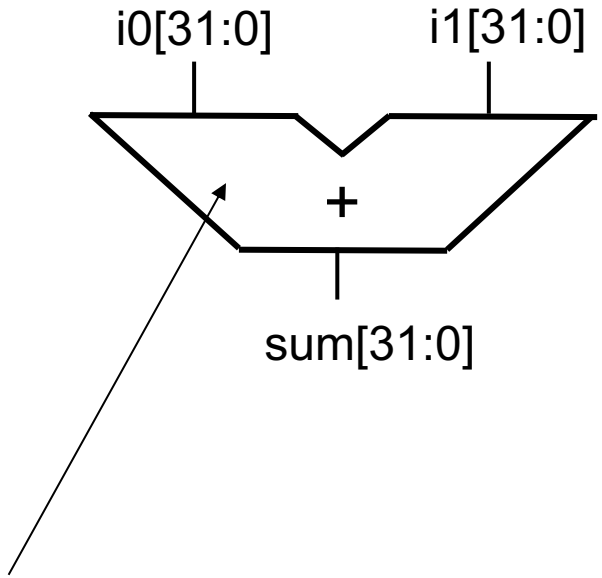
I_{DS} Vs V_{GS} characteristics in log scale

- Energy
  - Overview
  - Dynamic & Static
- **Adders**
  - **Single-Bit Full Adder**
  - **Carry-Ripple Adder**
  - **Carry-Bypass Adder**
  - **Carry-Lookahead Adder**

Binary Adder

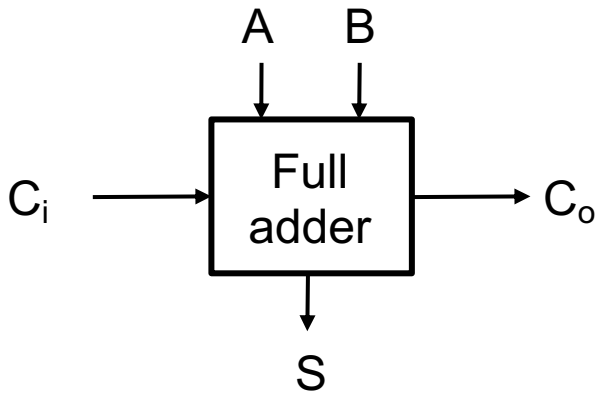
- Adders

```
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

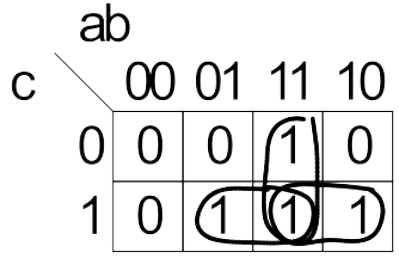


A	B	C_{in}	C_o	S
0	0	0	0	0
		1	0	1
0	1	0	0	1
		1	1	0
1	0	0	0	1
		1	1	0
1	1	0	1	0
		1	1	1

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

$$S = A \oplus B \oplus Ci$$

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



cout = ab + bc + ac

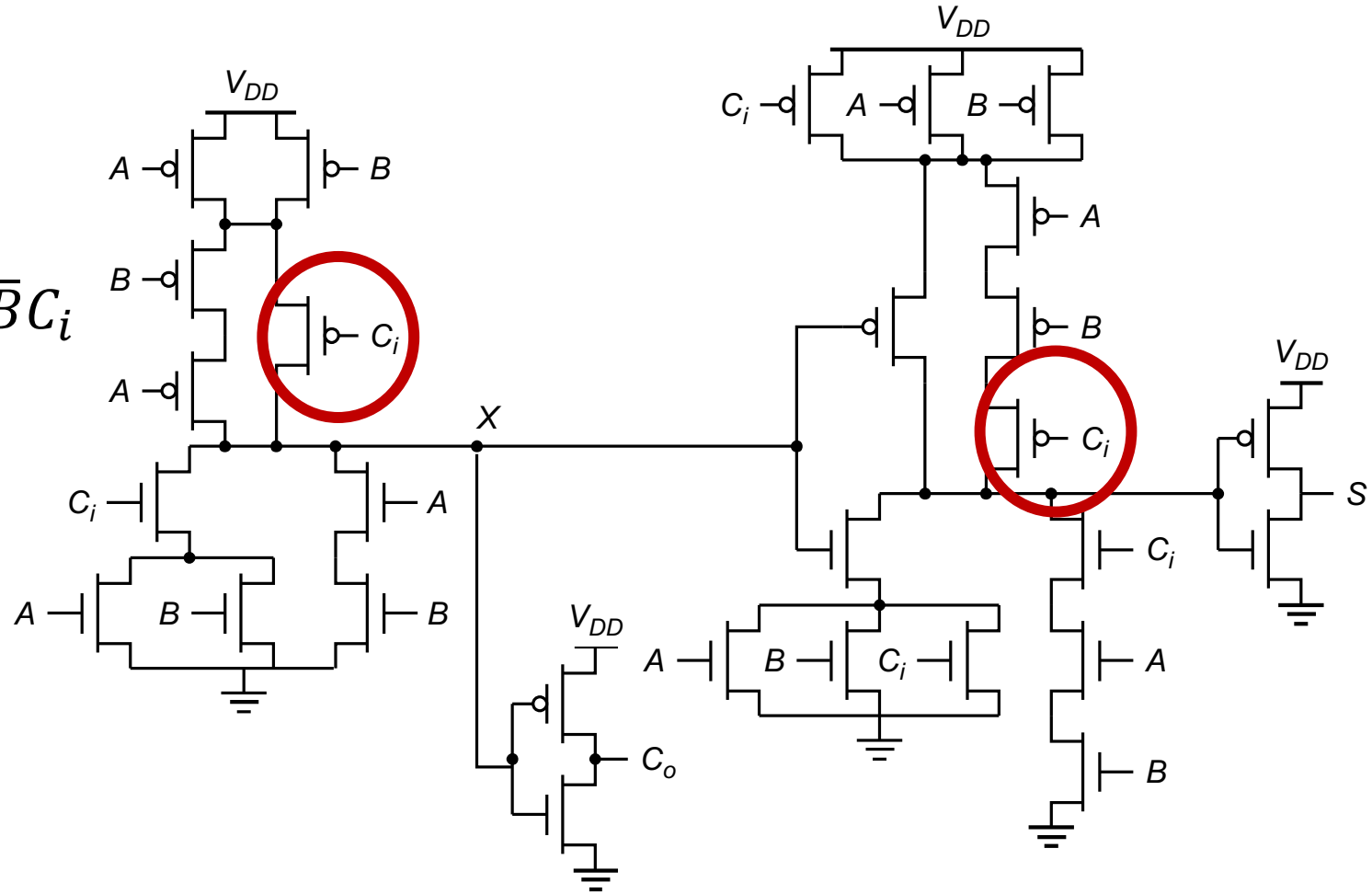
Static CMOS Full Adder

- Direct mapping of logic equations

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

$$\overline{C_o} = \overline{AB + C_i(A+B)}$$

$$S = ABC_i + A\bar{B}\bar{C_i} + \bar{A}B\bar{C_i} + \bar{A}\bar{B}C_i$$
$$= ABC_i + (A+B+C_i)\,\overline{C_o}$$



28 Transistors

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

- Define generate, propagate and kill as functions of inputs
  - Will use two at a time

Generate (G) = AB

Propagate (P) = A ⊕ B, close to (A+B)

Kill (K) = $\overline{A}\,\overline{B}$

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

$S = P \oplus Ci$

A	B	C_{in}	G	P	K	C_o	S
0	0	0	0	0	1	0	0
		1				0	1
0	1	0	0	1	0	0	1
		1				1	0
1	0	0	0	1	0	0	1
		1				1	0
1	1	0	1	0	0	1	0
		1				1	1

Can also derive expressions for C_o based on K and P

Why Generate and Propagate?

- Ripple-carry adders have to wait for the carry of each bit pair to be computed before getting the result.
  - This can take a long time!
- G and P depend only on A and B, not on C (the carry)
  - Thus, we can compute G and P for all bits in parallel
  - Then, the idea is that we can more easily compute the sum without having to wait for carries

Generate (G) = AB

Propagate (P) = A $\oplus$ B, close to (A+B)

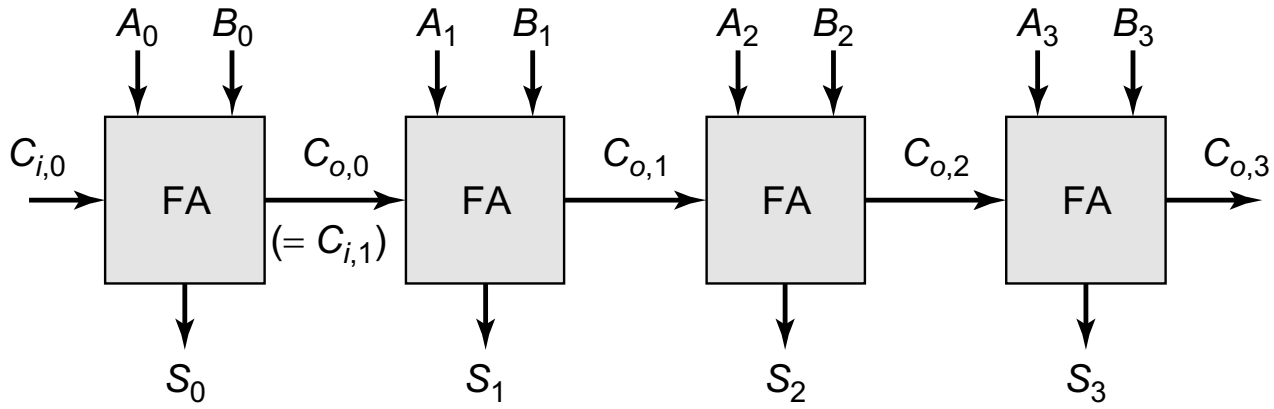
Announcement

- Project starts!
  - Checkpoint 1 this week.
  - Wrap up your labs if you haven’t done so already.
- Homework 7 out this week.

- Energy
  - Overview
  - Dynamic & Static
- Adders
  - Single-Bit Full Adder
  - **Carry-Ripple Adder**
  - **Carry-Bypass Adder**
  - **Carry-Lookahead Adder**

The Ripple-Carry Adder

- 4-bit adder



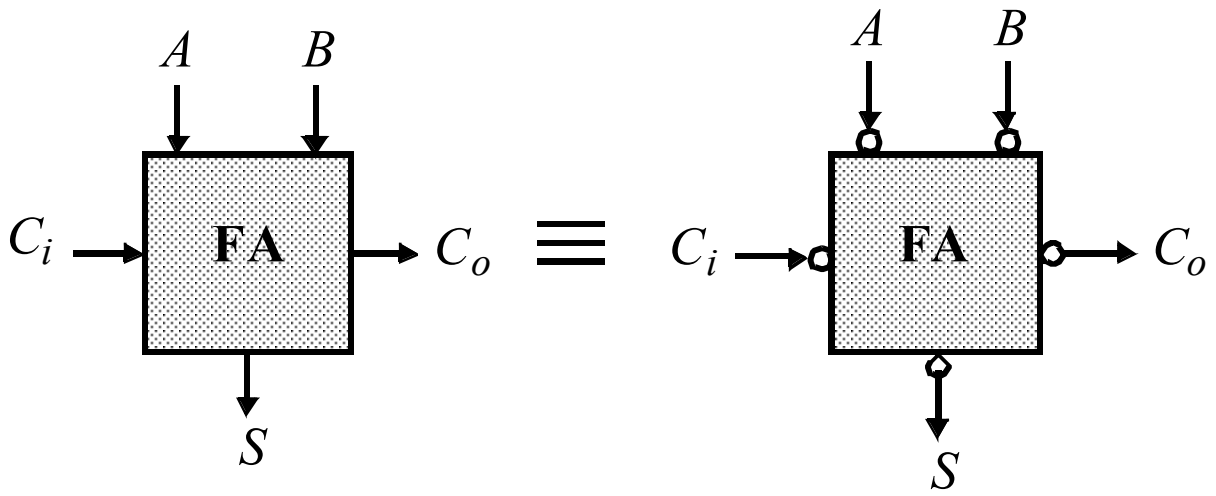
Worst case delay linear with the number of bits

$$t_d = O(N)$$

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

Goal: Make the fastest possible carry path circuit

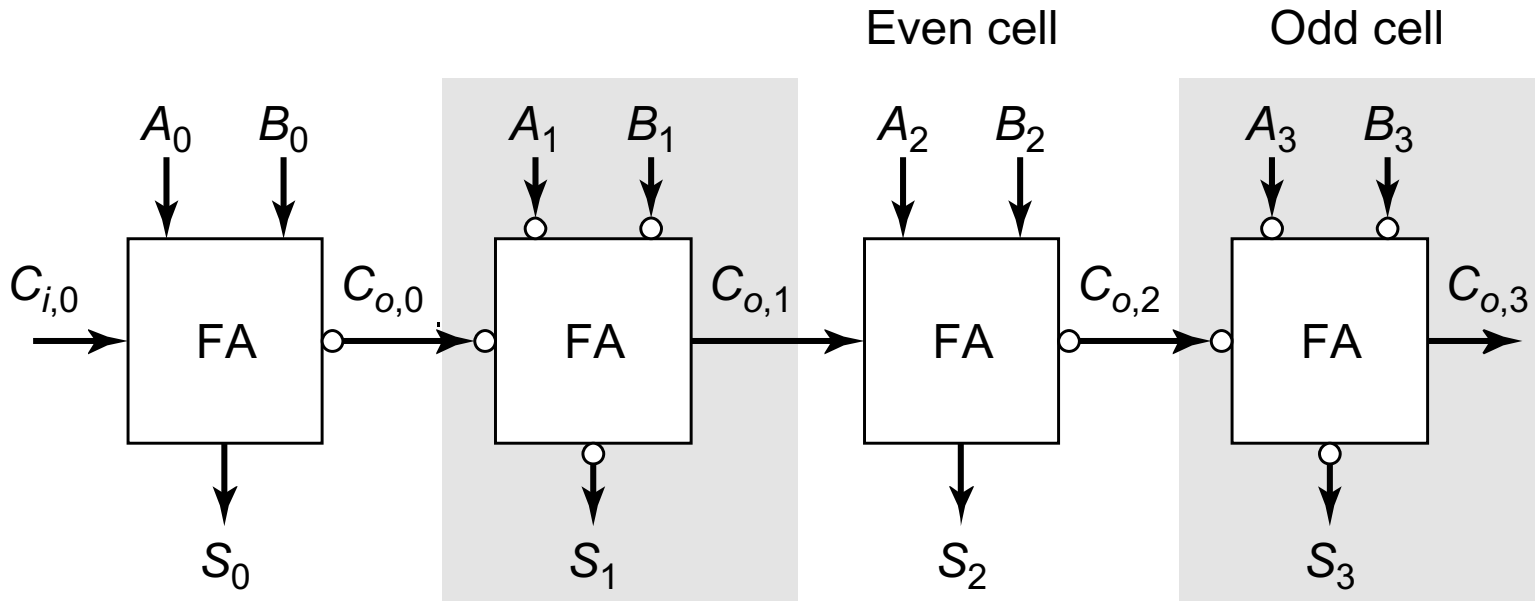
Inversion Property



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

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

Minimize Critical Path by Reducing Inverting Stages

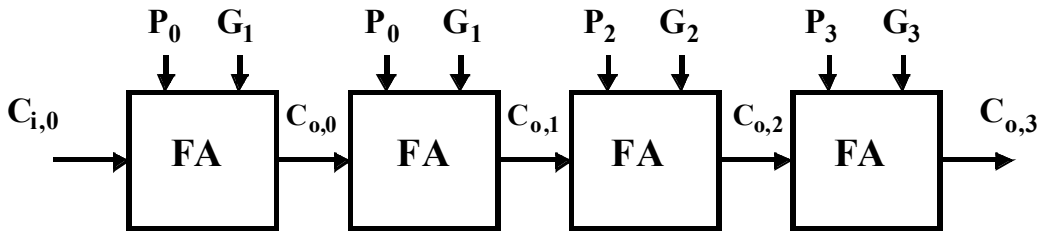


Exploit Inversion Property

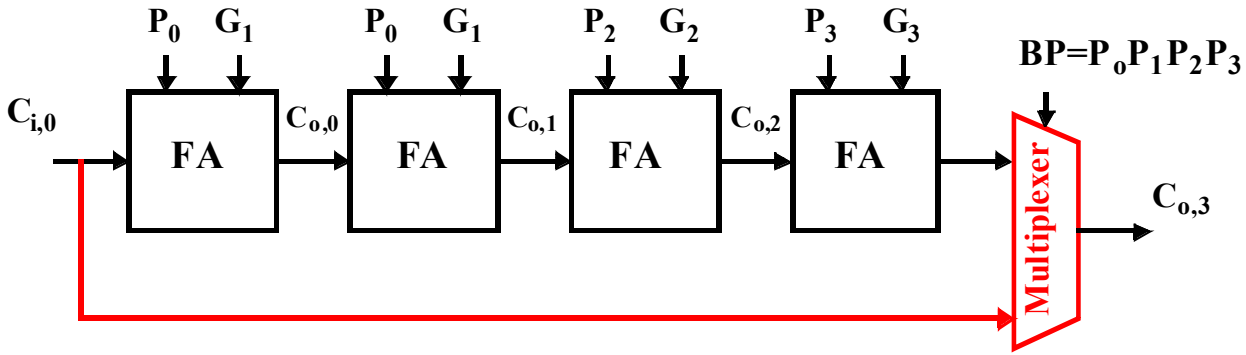
- Energy
  - Overview
  - Dynamic & Static
- Adders
  - Single-Bit Full Adder
  - Carry-Ripple Adder
  - **Carry-Bypass Adder**
  - **Carry-Lookahead Adder**

Carry-Bypass Adder

- Also called ‘carry skip’

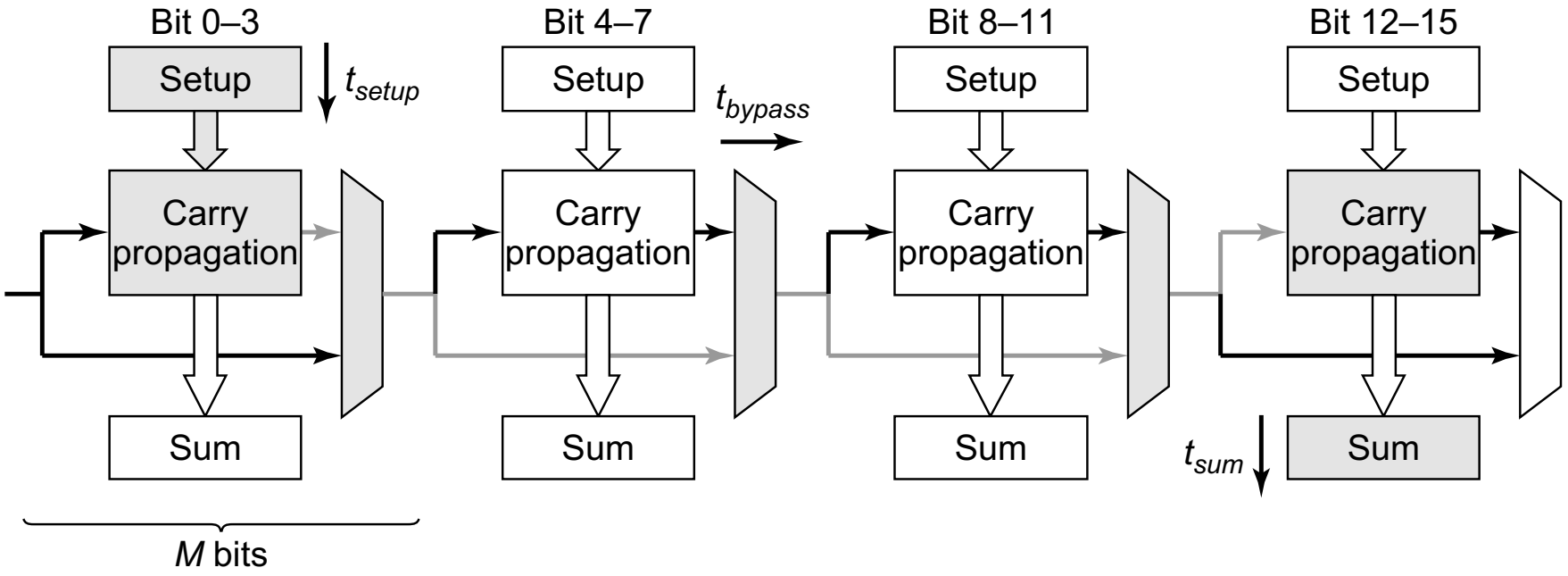


Also called Carry-Skip



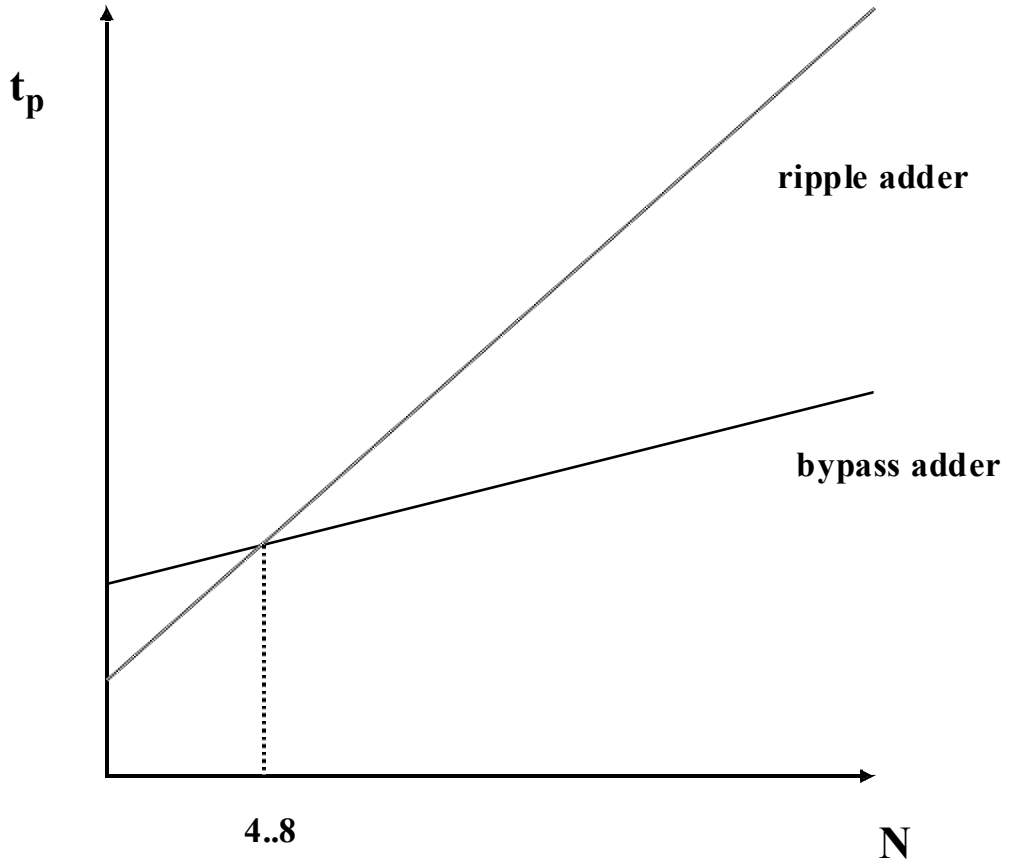
Idea: If (P0 and P1 and P2 and P3 = 1) then $C_{o3} = C_0$, else “kill” or “generate”.

Carry-Bypass Adder (cont.)



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

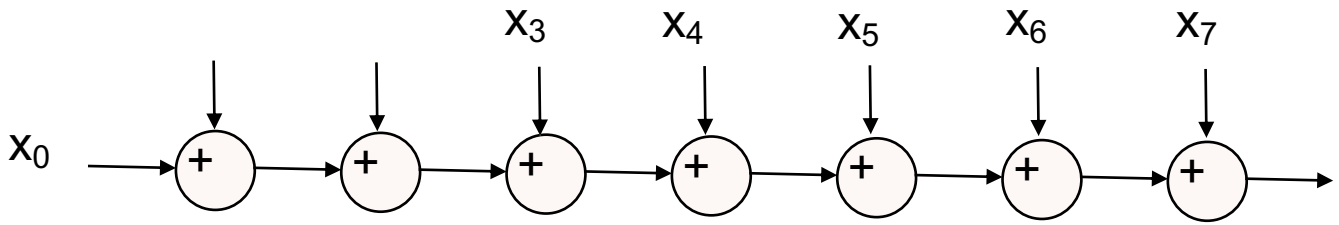
Carry Ripple versus Carry Bypass



- Depends on technology, design constraints

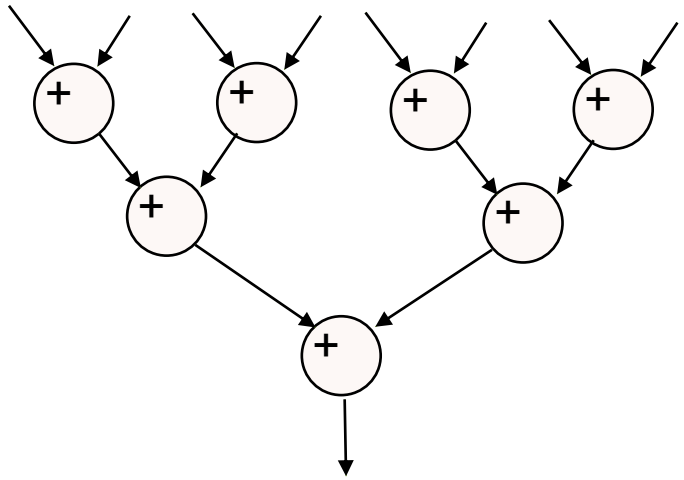
- Energy
  - Overview
  - Dynamic & Static
- Adders
  - Single-Bit Full Adder
  - Carry-Ripple Adder
  - Carry-Bypass Adder
  - **Carry-Lookahead Adder**

Trees for optimization



T = O(N)

$$((((((x_0 + x_1) + x_2) + x_3) + x_4) + x_5) + x_6) + x_7$$



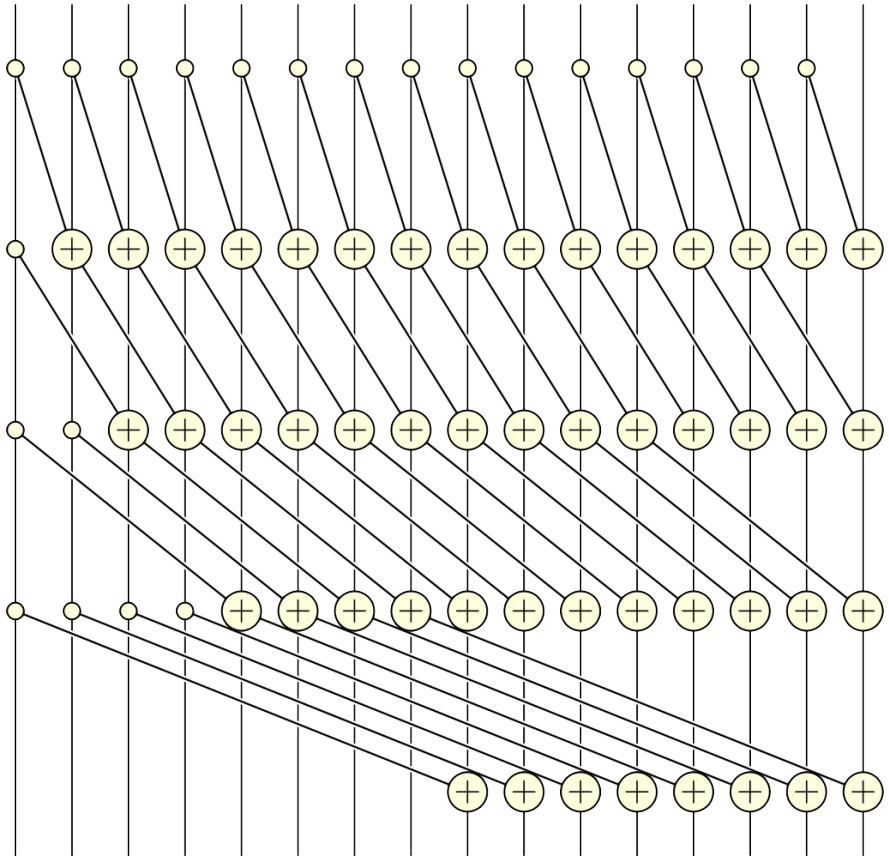
T = O(log N)

$$((x_0 + x_1) + (x_2 + x_3)) + ((x_4 + x_5) + (x_6 + x_7))$$

- "+" is associative.

Parallel Prefix, or “Scan”

- If “+” is an associative operator, and $x_0,\ldots,x_{p-1}$ are input data then *parallel prefix* operation computes: $y_j = x_0 + x_1 + \ldots + x_j$ **for j=0,1,…,p-1**



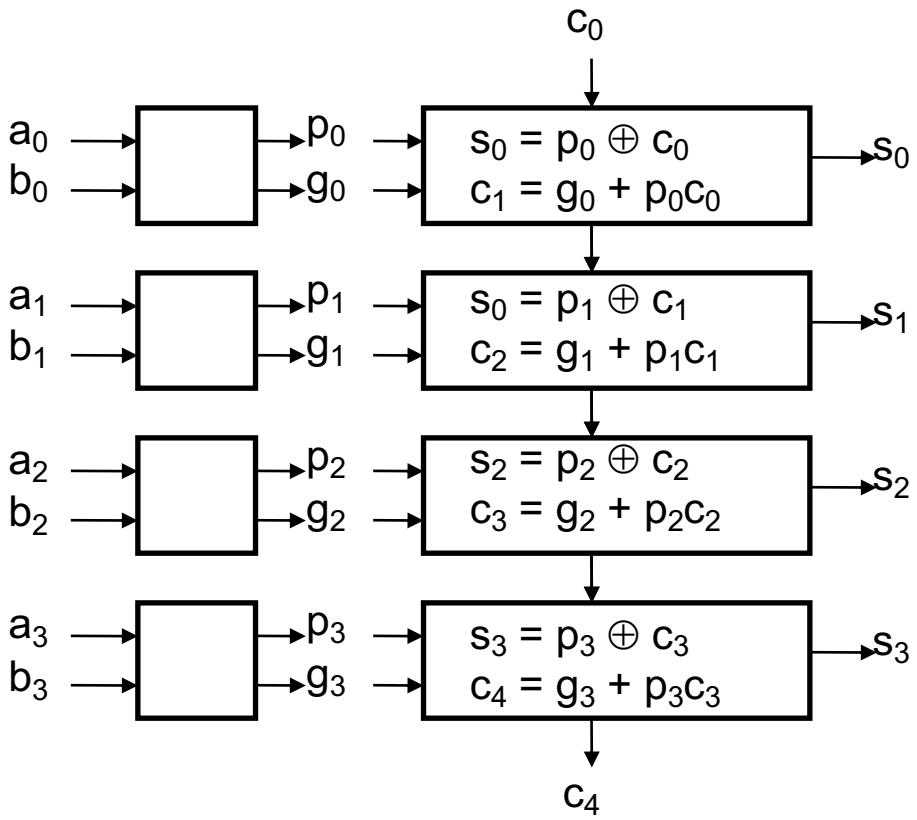
Lookahead - Basic Idea

Generate (G) = AB

Propagate (P) = A ⊕ B, close to (A+B)

Kill (K) = $\overline{A}\,\overline{B}$

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



- So far, no advantage over ripple adder: T α N

Lookahead - Basic Idea

Expanding lookahead equations:

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

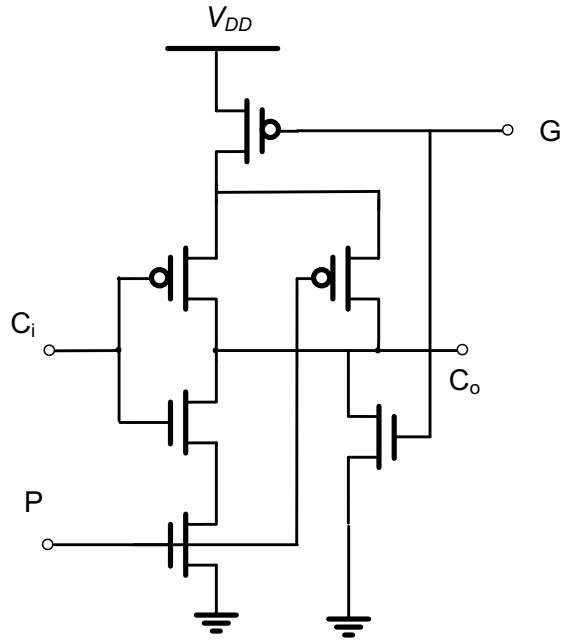
$C_o = A\,B + B\,Ci + A\,Ci = G + P\,C_i$

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:

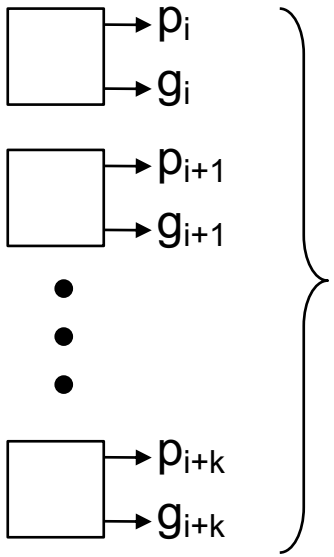
$C_{o,k} = G_k + P_k(G_{k-1}+P_{k-1}(\ldots+P_1(G_0+P_0C_{i,0})\ldots))$



Carry-lookahead gate grows at each bit position!

Lookahead - Basic Idea

- "Group" propagate and generate signals:
- P true if the group as a whole propagates a carry to c_{out}
- G true if the group as a whole generates a carry
- Group P and G can be generated hierarchically



c_{in}

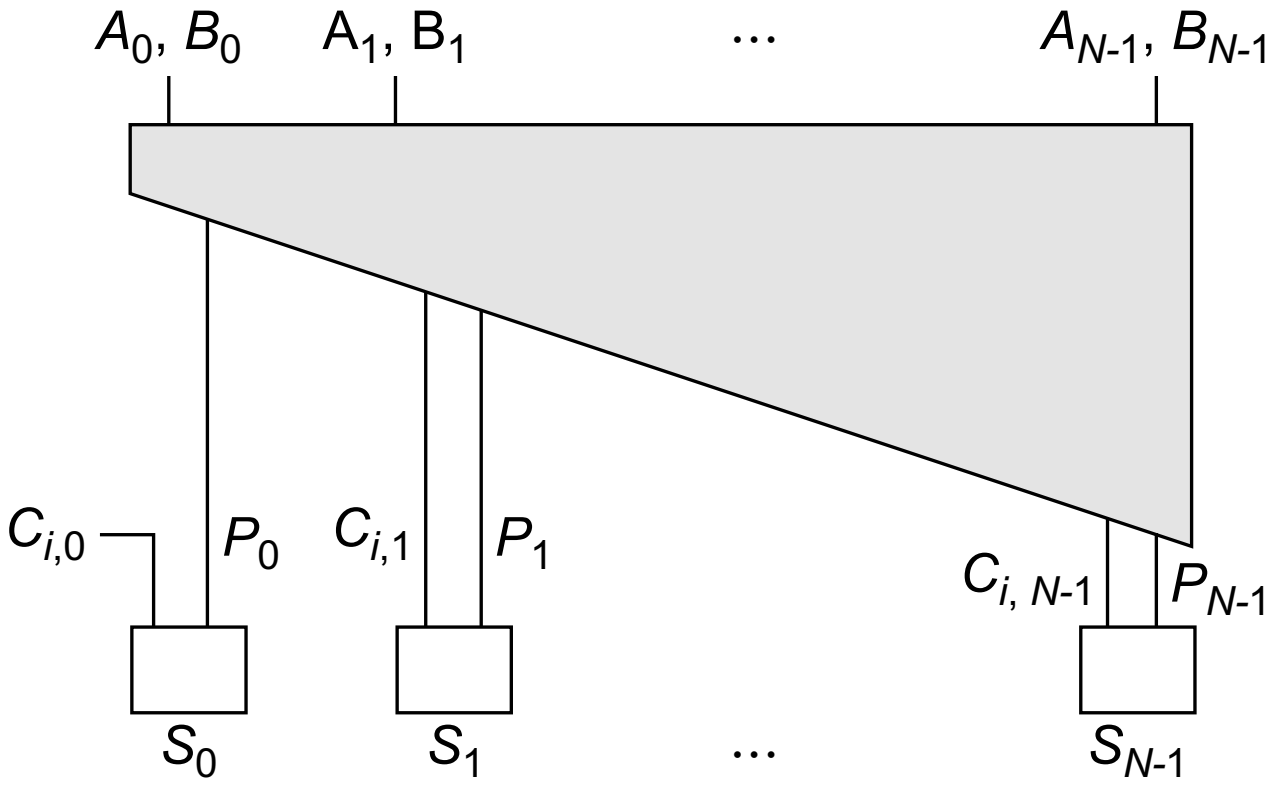
$P = p_i\, p_{i+1} \ldots p_{i+k}$

$G = g_{i+k} + p_{i+k}g_{i+k-1} + \ldots + (p_{i+1}p_{i+2} \ldots p_{i+k})g_i$

c_{out}

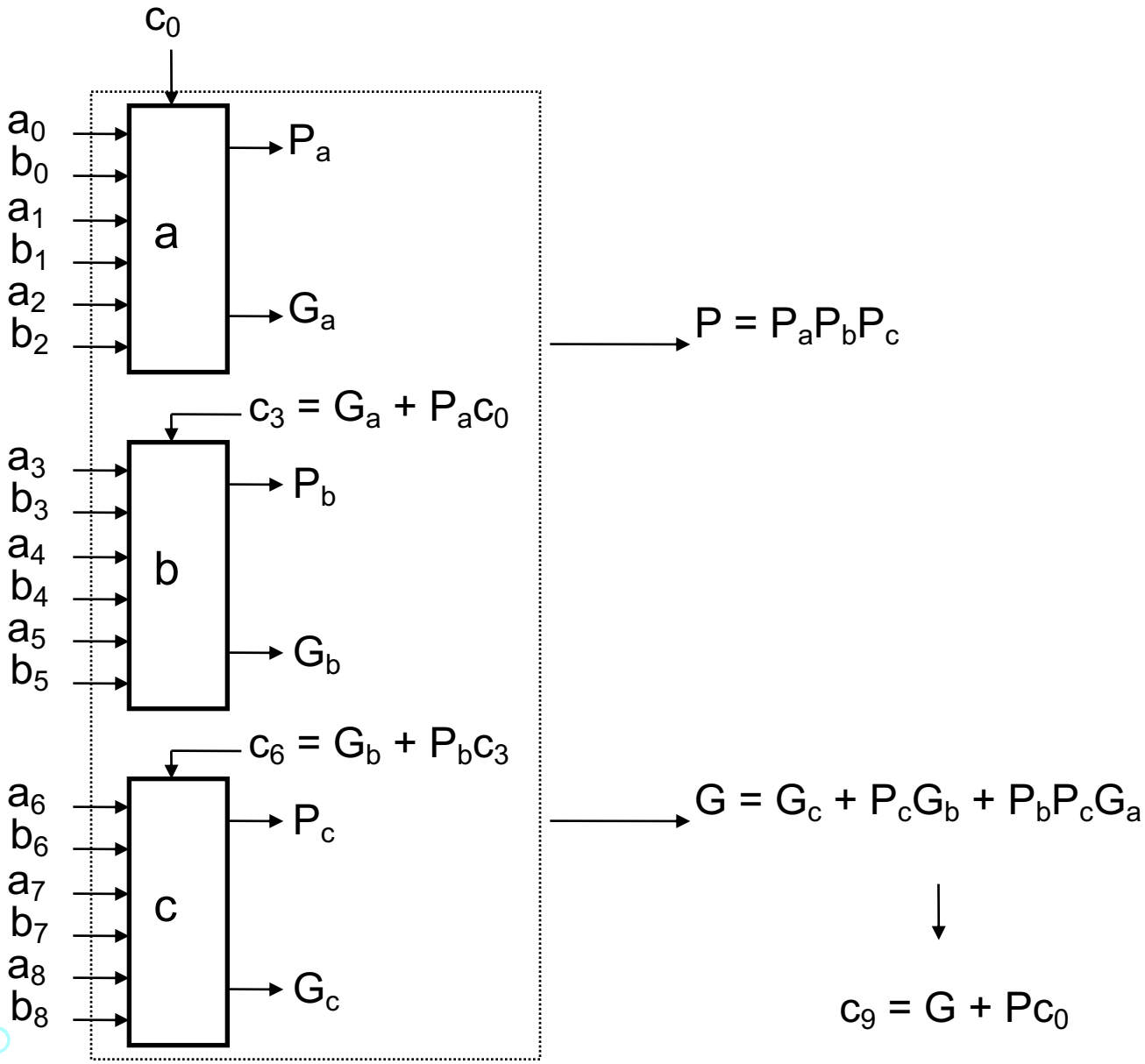
$$c_{out} = G + Pc_{in}$$

Carry Look-ahead Adders



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

Carry Look-ahead Adders



9-bit Example of hierarchically generated P and G signals:

Carry Lookahead Trees

Build the carry-lookahead tree as a hierarchy of gates

$$C_{o,0} = G_0 + P_0 C_{i,0}$$

$$C_{o,1} = G_1 + P_1 C_{i,1} = G_1 + P_1 G_0 + P_1 P_0 C_{i,0}$$

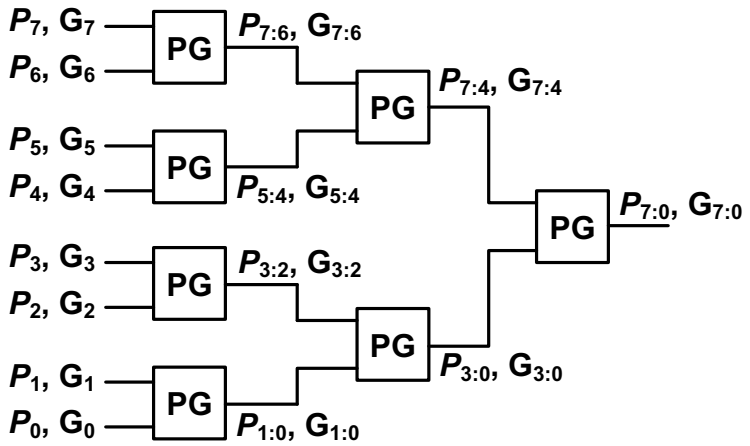
$$C_{o,2} = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{i,0}$$
$$= (G_2 + P_2 G_1) + (P_2 P_1)(G_0 + P_0 C_{i,0}) = G_{2:1} + P_{2:1} C_{O,0}$$

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:

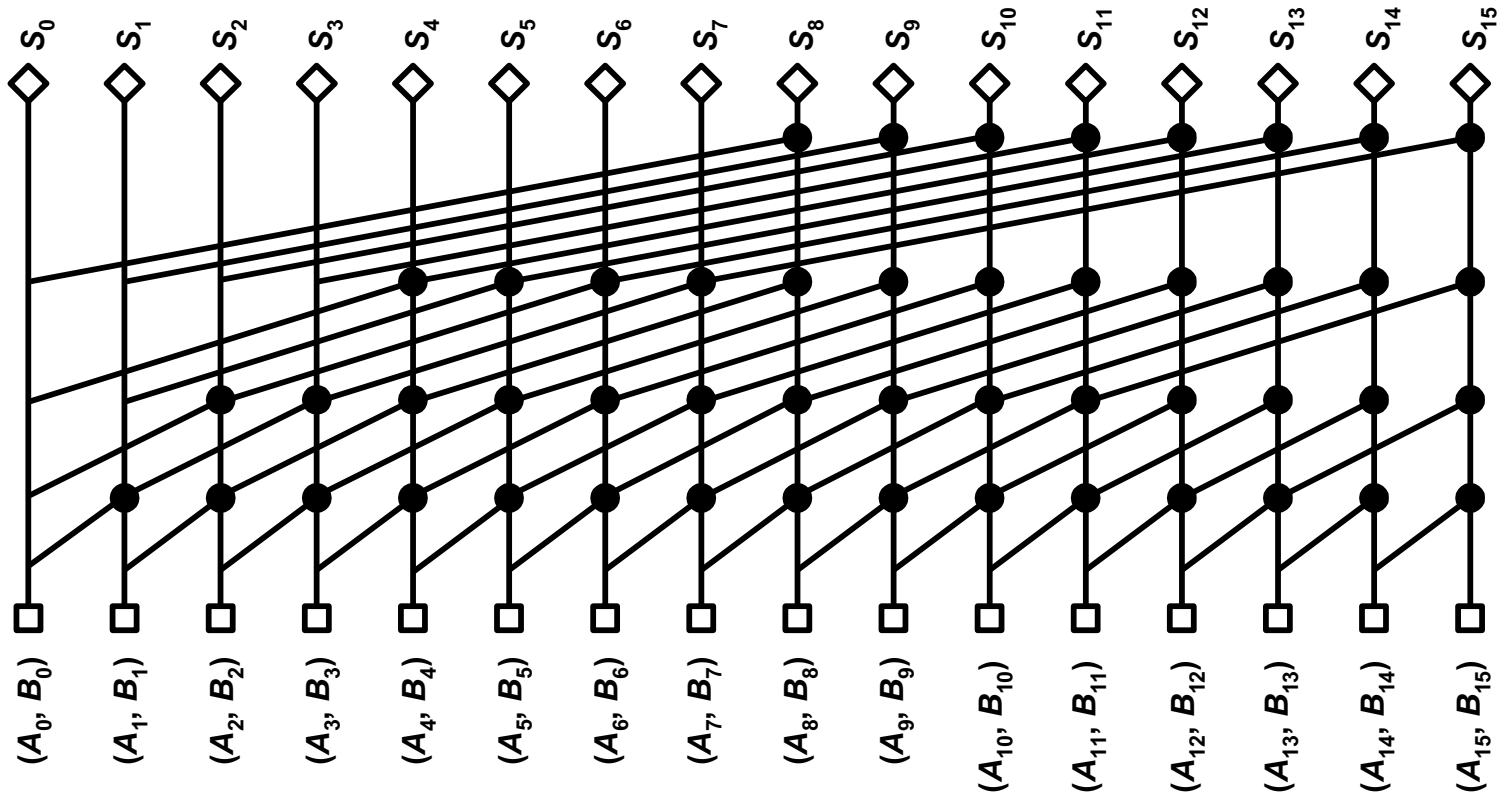
    $P_{1:0} = P_1 \cdot P_0$, $G_{1:0} = G_1 + P_1 \cdot G_0$, → $C_{out1} = G_{1:0} + P_{1:0} \cdot C_{0,in}$
- Combine these groups in a tree structure:
  - Delay is now ~$\log_2(N)$



Many Kinds of Tree Adders

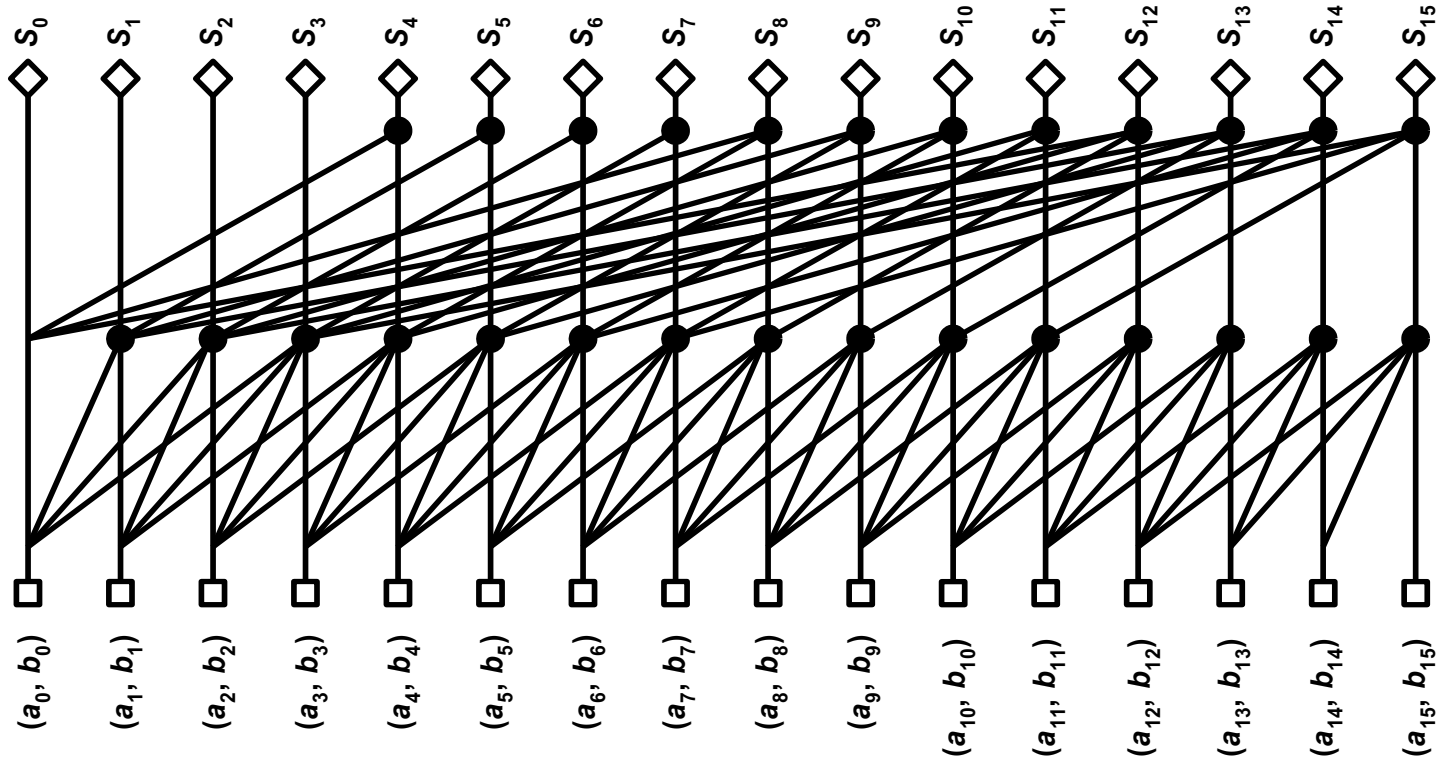
- Many ways to construct these tree (or “carry lookahead”) adders
  - Many of these variations named after the people who first came up with them
- Most of these vary three basic parameters:
  - Radix: how many bits are combined in each PG gate
    - Previous example was radix 2; often go up to radix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

Tree Adders



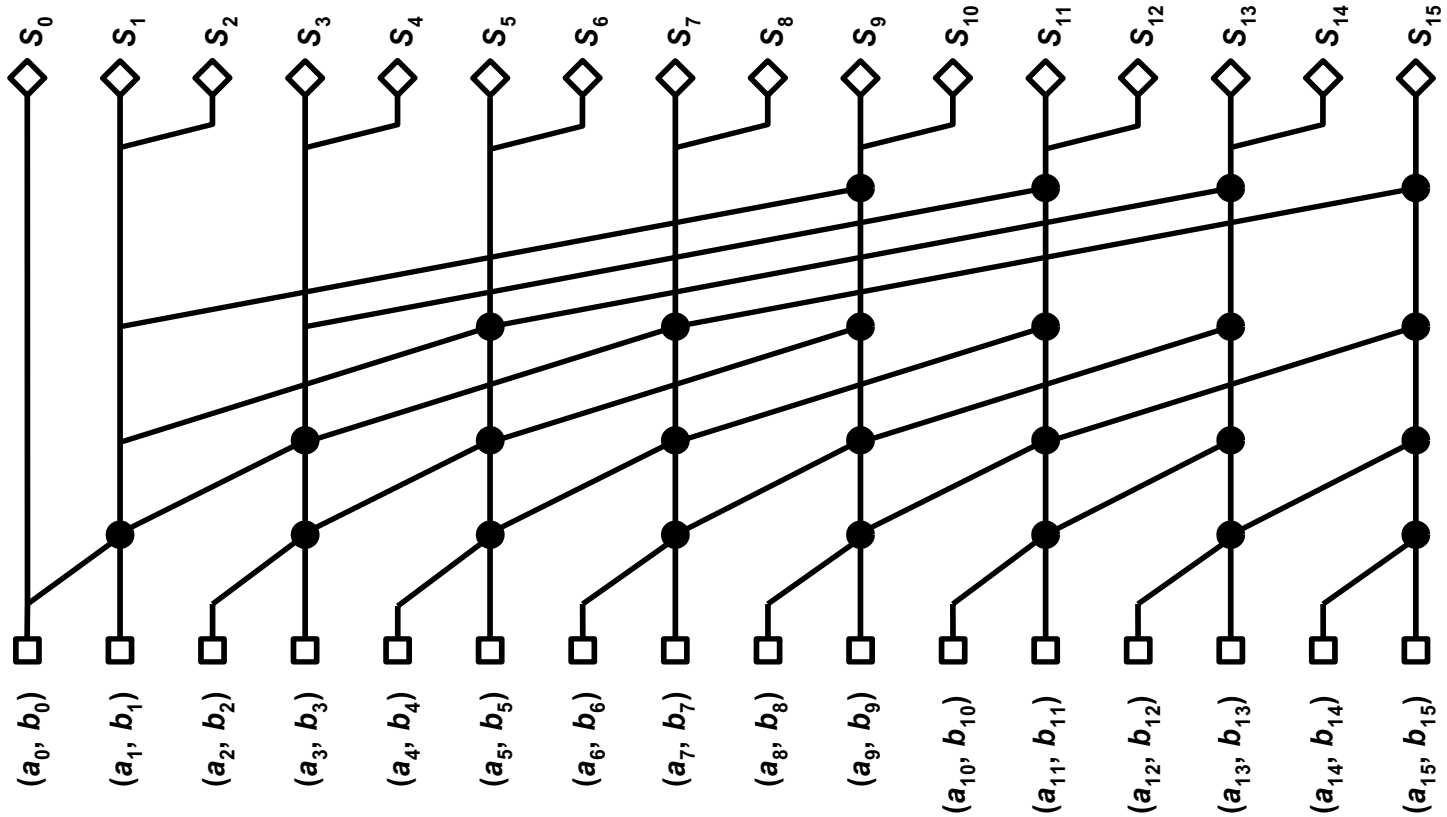
16-bit radix-2 Kogge-Stone tree

Tree Adders



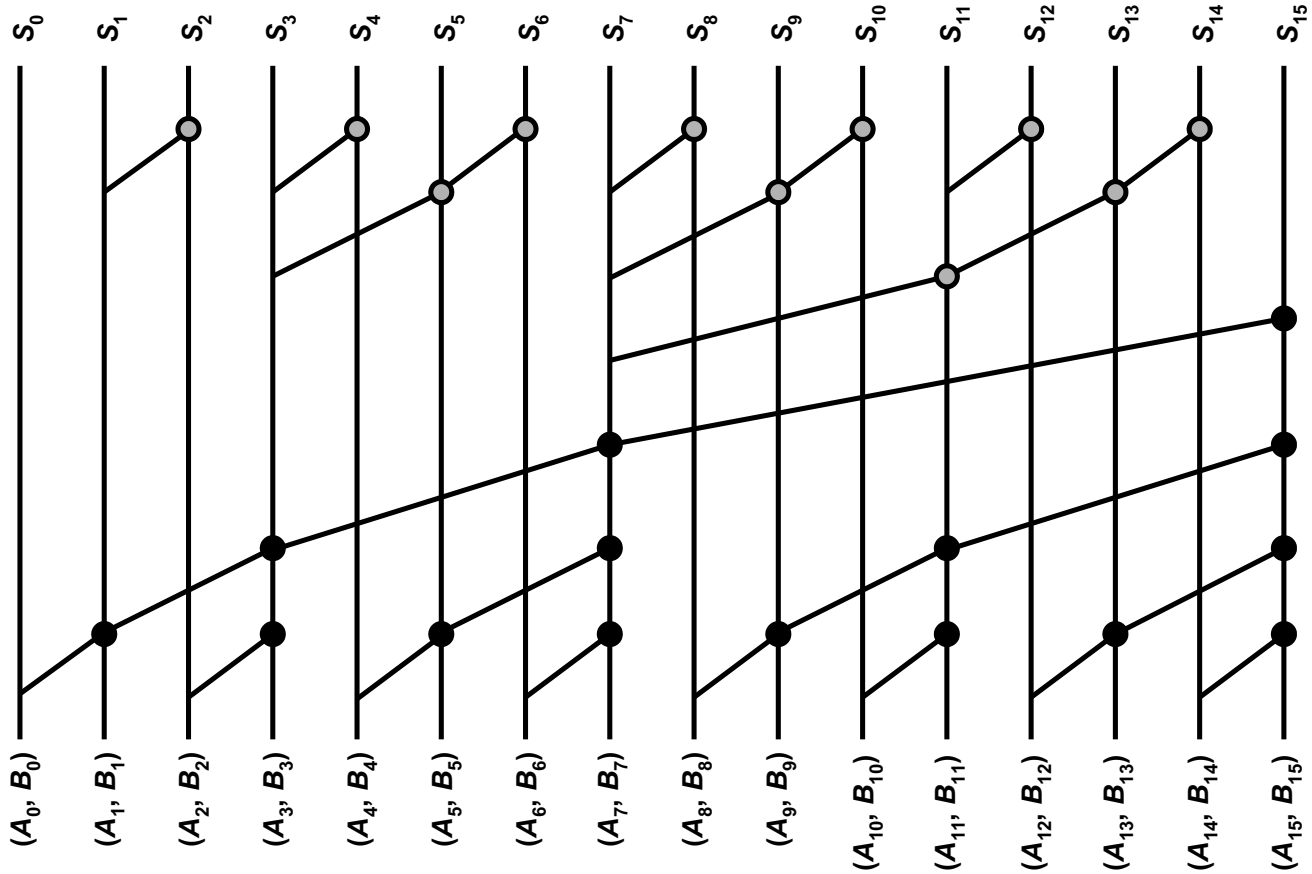
16-bit radix-4 Kogge-Stone Tree

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 is usually faster than ripple-carry for lengths > 8
- Carry-lookahead, O(~logN) is often the fastest adder with N > 16