EECS151 : Introduction to Digital Design and ICs

Lecture 18 – Adders, Multipliers

Bora Nikolić

TSMC Details The Benefits of Its N3 Node

October 27, 2021, EETimes - TSMC, now chugging along with its N5 process node, said it will have its evolutionary N4 node ramped up to volume production this year. The N3 node, which will provide more of a technological leap than N4, is planned to go into volume production in the second half of 2022. N3 will indeed offer customers the kind of performance improvements they might hope for from a major node jump, though the speed improvement will be at the low-end of TSMC's projected aspirations from last year; the company also just missed its target for density improvement.

TSMC Node Comparisons

Metric	N7 to N5	N5 to N3 (2020 projection)	N5 to N3 (2021 actual)
Logic density improvement	1.87x	1.7x	1.6x
Speed improvement	15%	10-15%	11%
Power draw improvement	-20%	(n/a)	-27%

Source: TSMC, EE Times

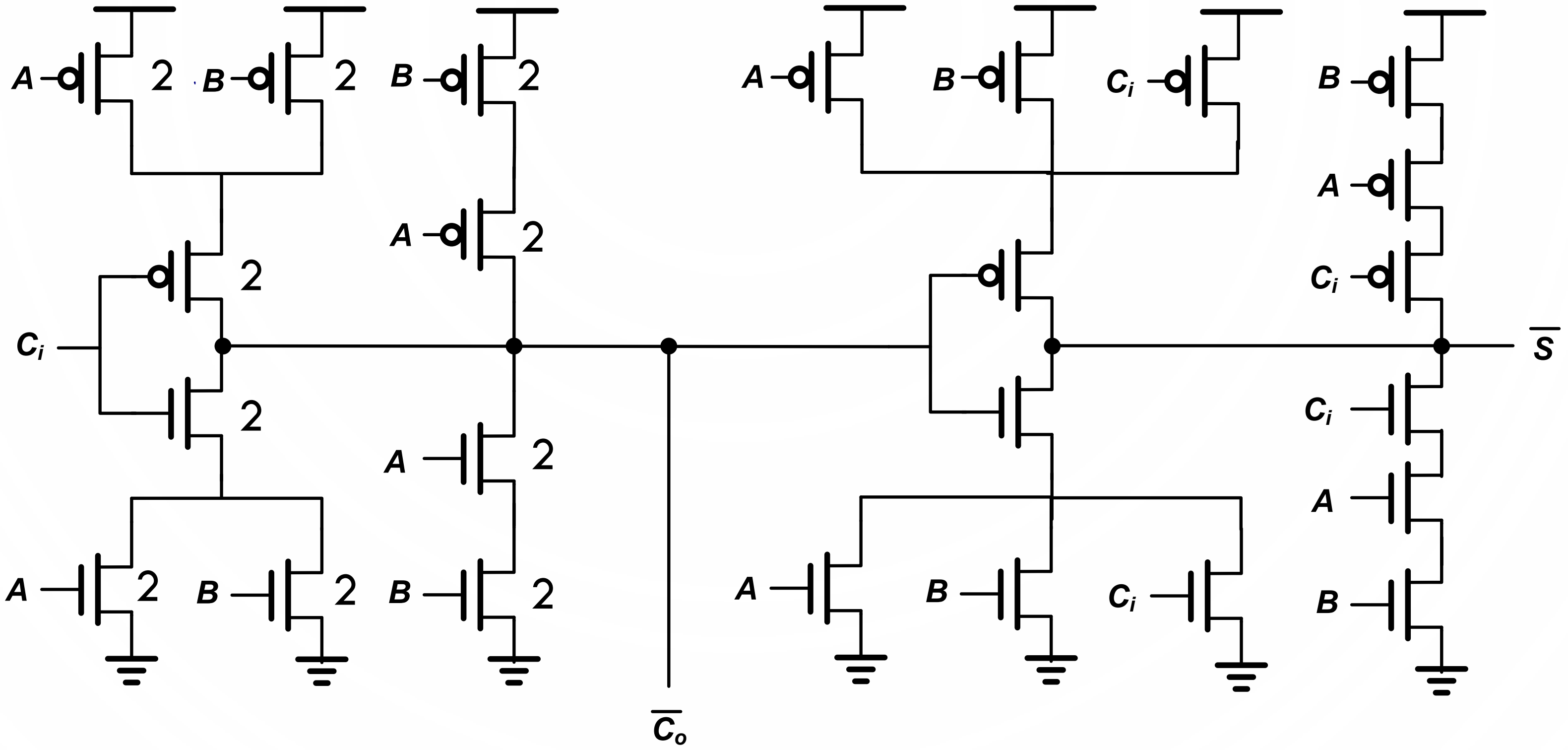
Review

- Binary adders are a common building block of digital systems
- Carry is in the critical path
- Mirror adders cells are commonly found in libraries
- Ripple-carry adder is the least complex, lowest energy

Ripple-Carry Adders

Sizing the Mirror Adder

$g_{Ci} = 2$

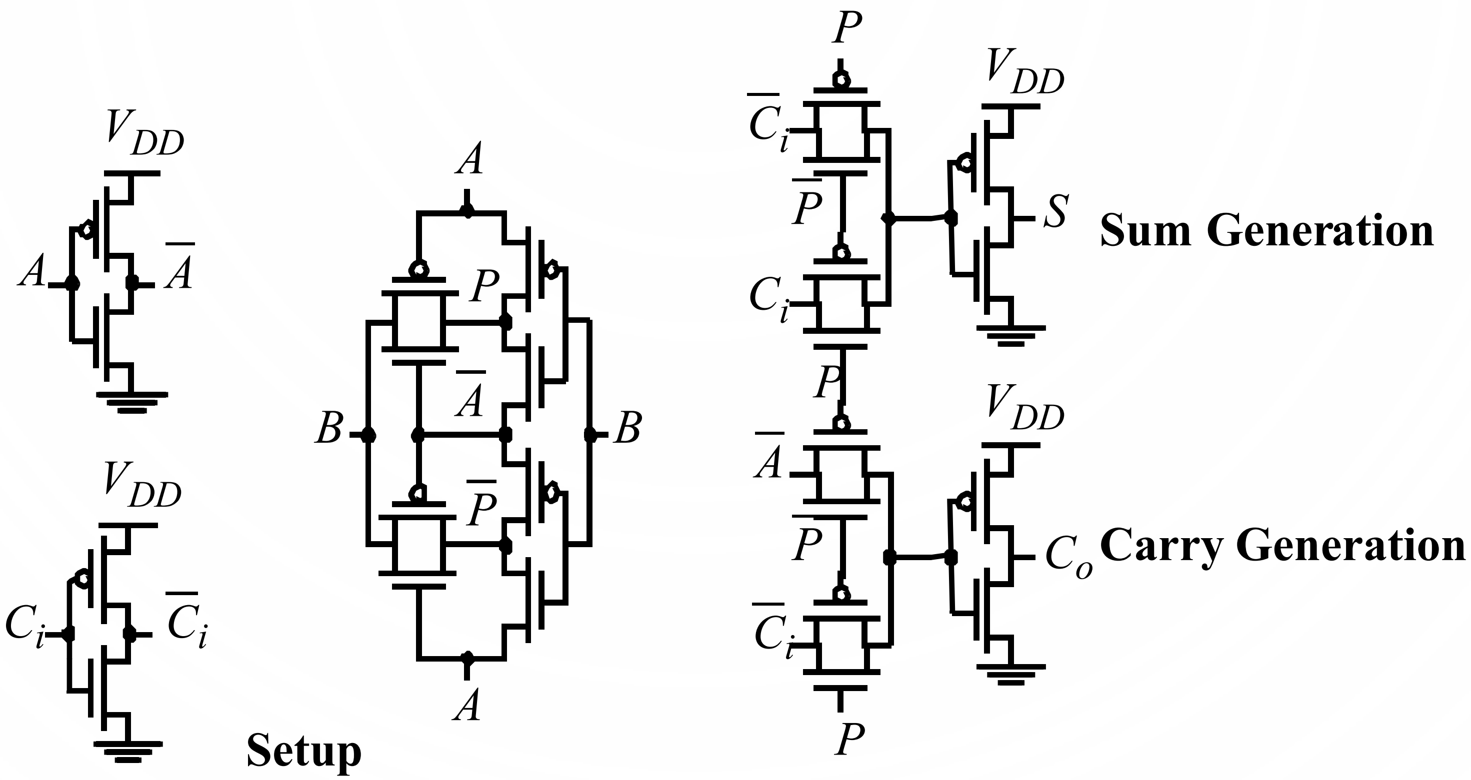


- Carry is in the critical path
- Optimal effort is 4, logical effort is 2
- Drives one carry and one sum input
  - Conveniently split fanout
- All stages equally sized

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 capacitance at node C_o.

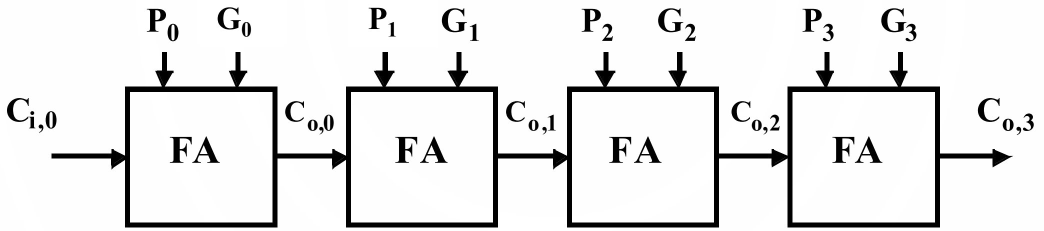
Transmission Gate Full Adder



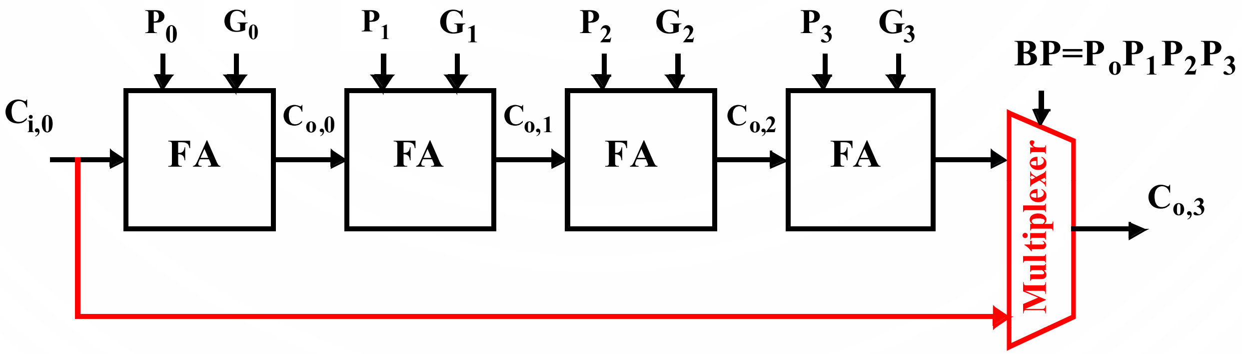
Carry Bypass Adders

Carry-Bypass Adder

- Also called 'carry skip'

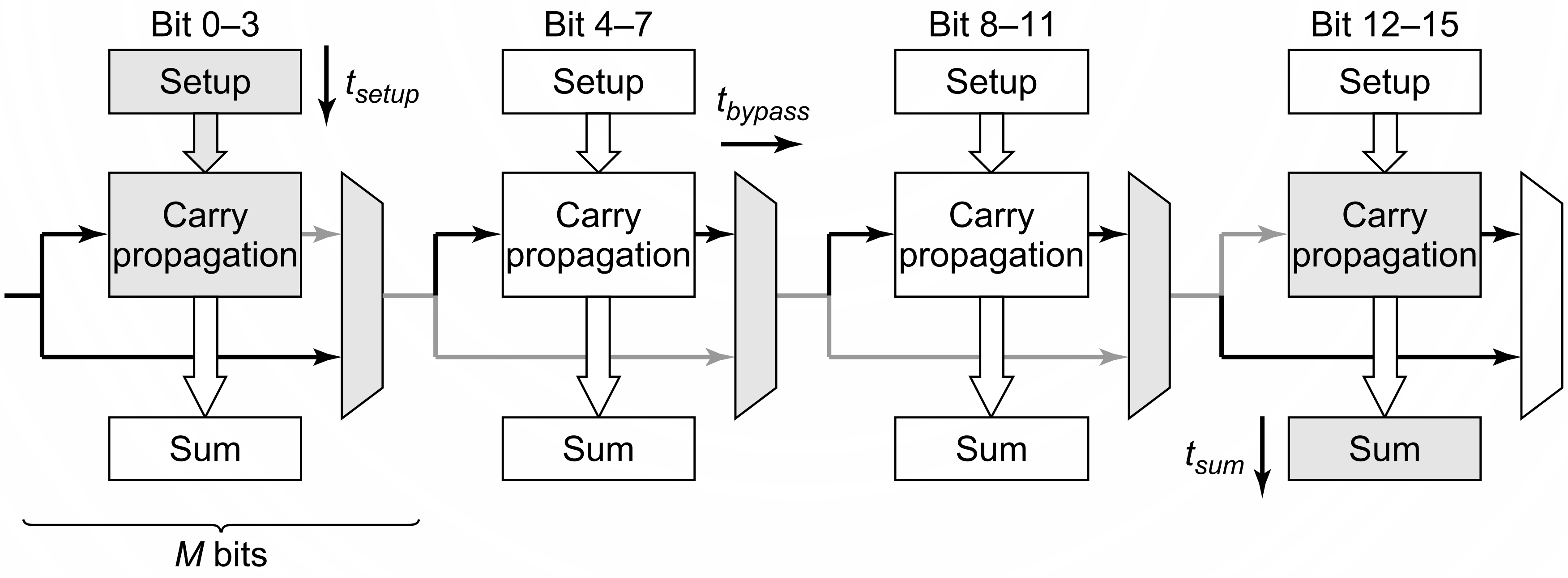


Also called Carry-Skip



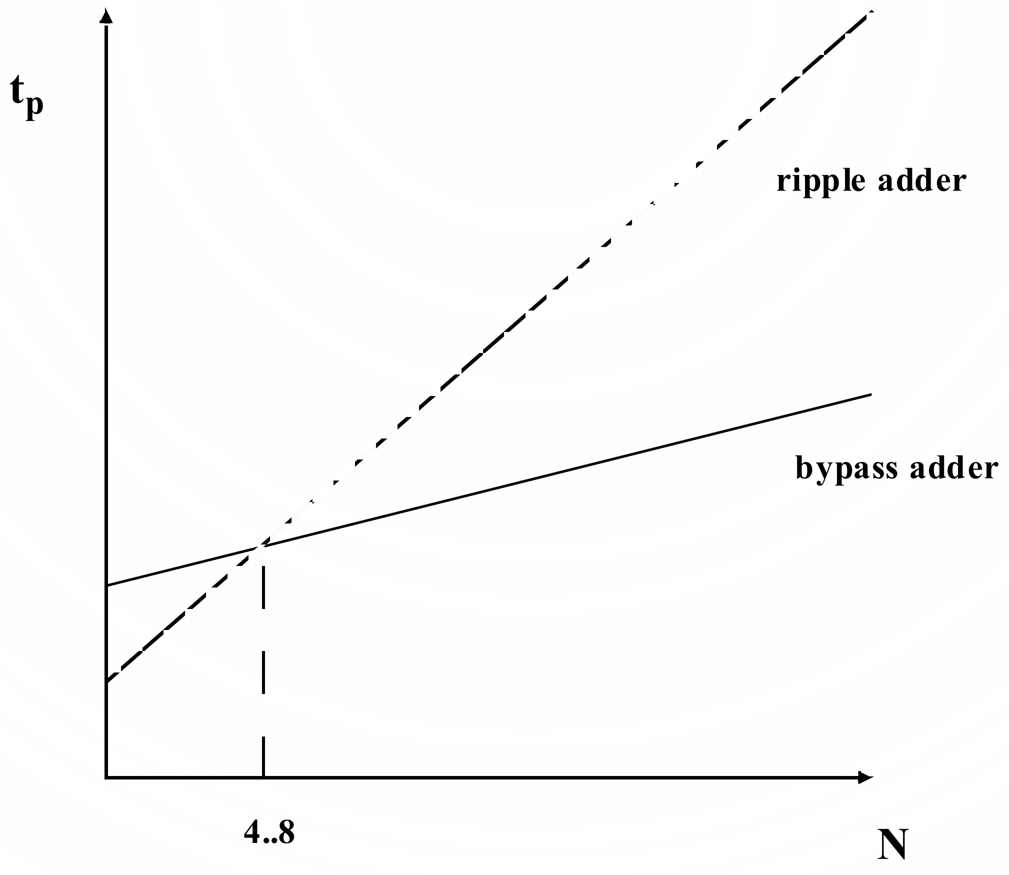
Idea: If (P0 and P1 and P2 and P3 = 1) then $C_{o3} = C_0$, else "kill" or "generate".

Carry-Bypass Adder (cont.)



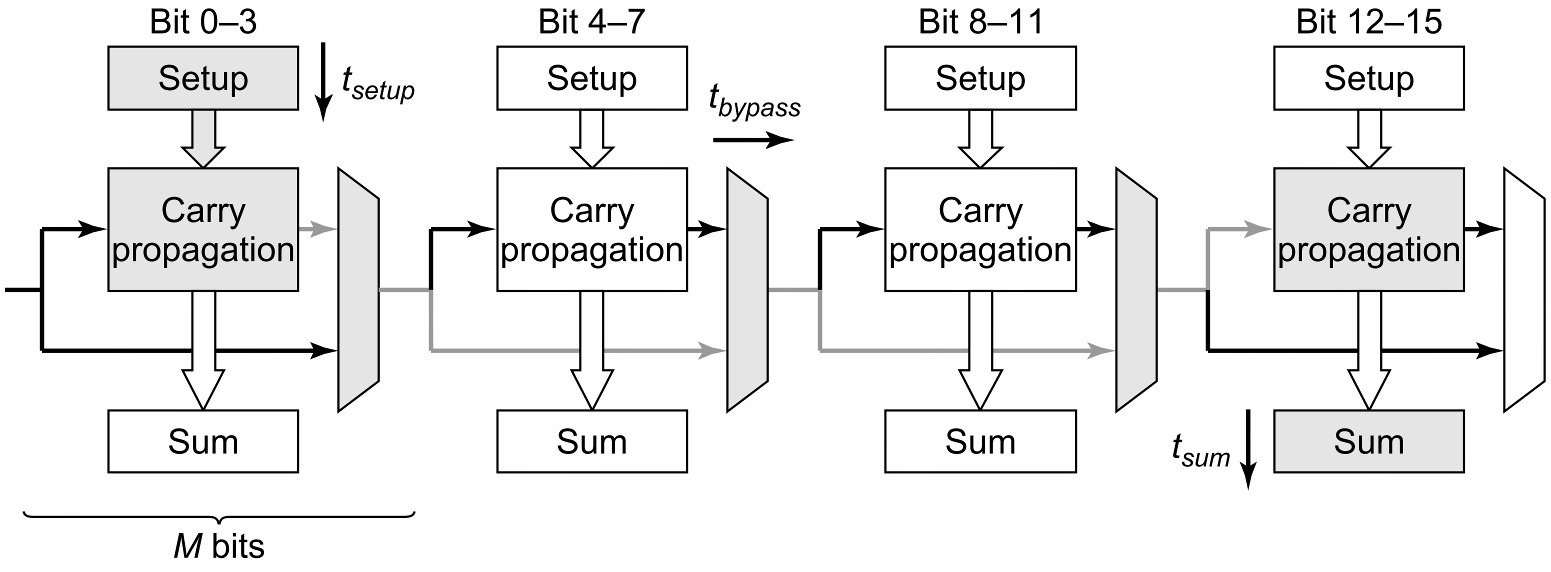
$$t_{adder} = t_{setup} + M_{tcarry} + (N/M-1)t_{bypass} + (M-1)t_{carry} + t_{sum}$$

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 bit positions are optimal

www.yellkey.com/ahead

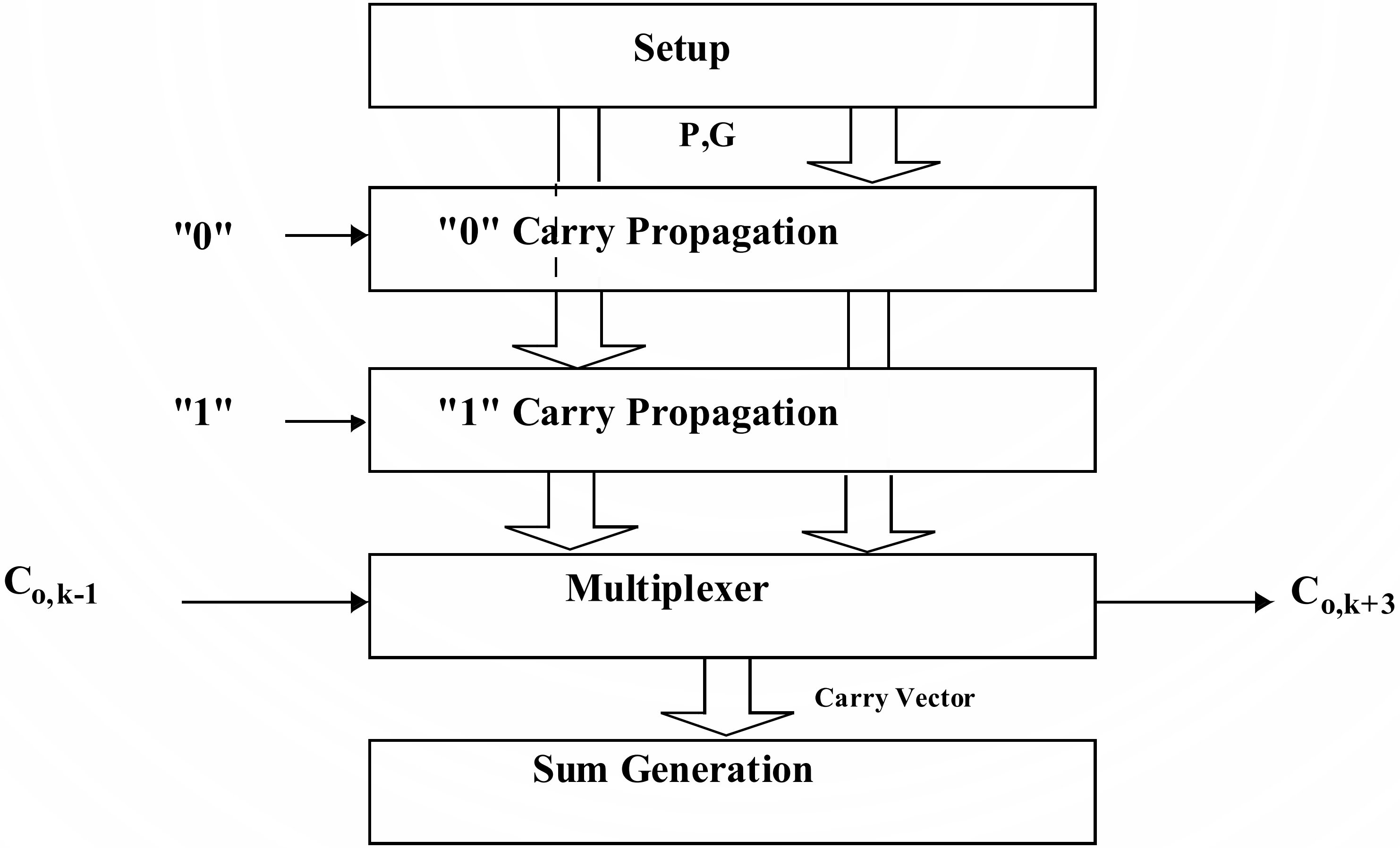
Faster Carry-Bypass

Administrivia

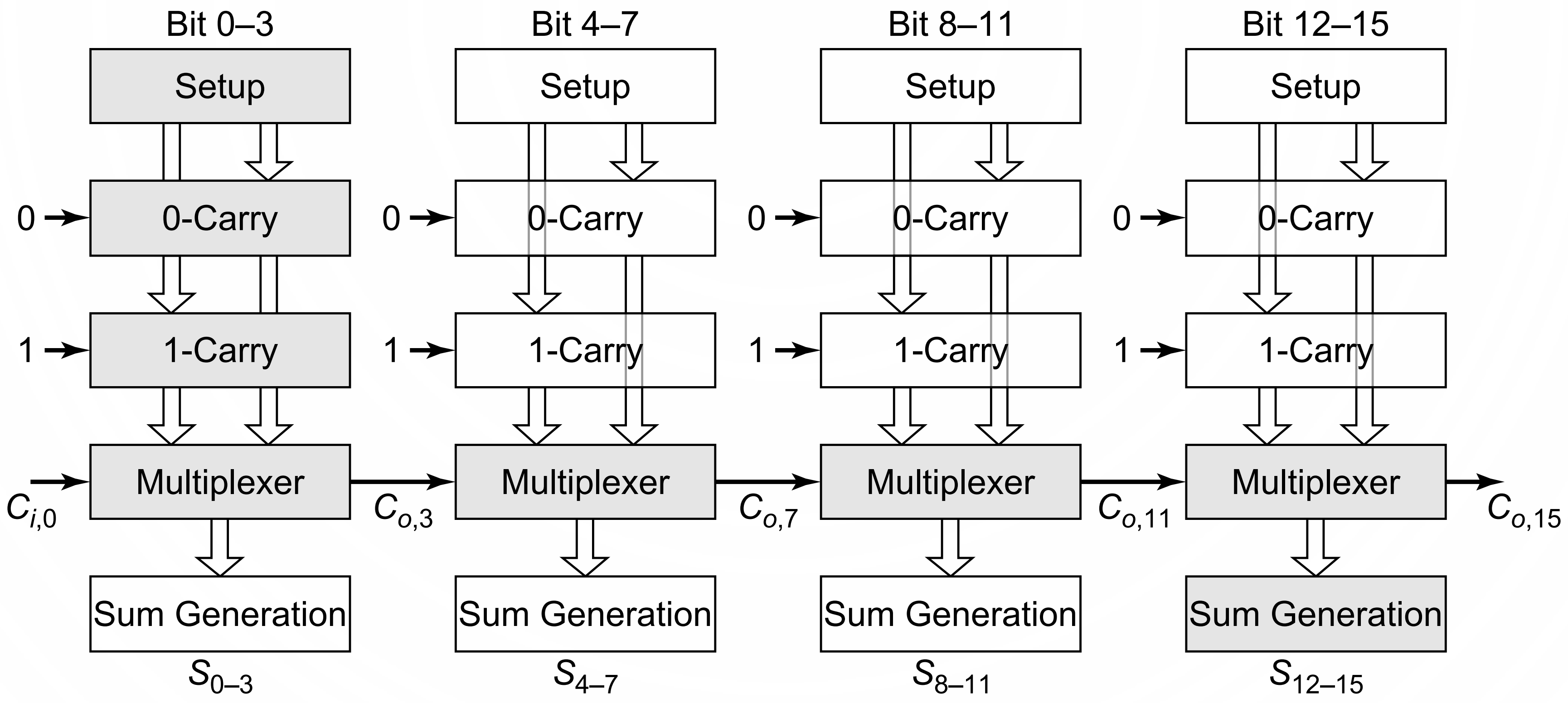
- Homework 7 due this week
- Homework 8 due next week
  - In scope for midterm
- All labs need to be checked off by this week!
- Projects (ASIC and FPGA) started, first check point this week
- Midterm 2 is on November 4 at 7pm
  - Review session tonight at 7pm

Carry-Select Adders

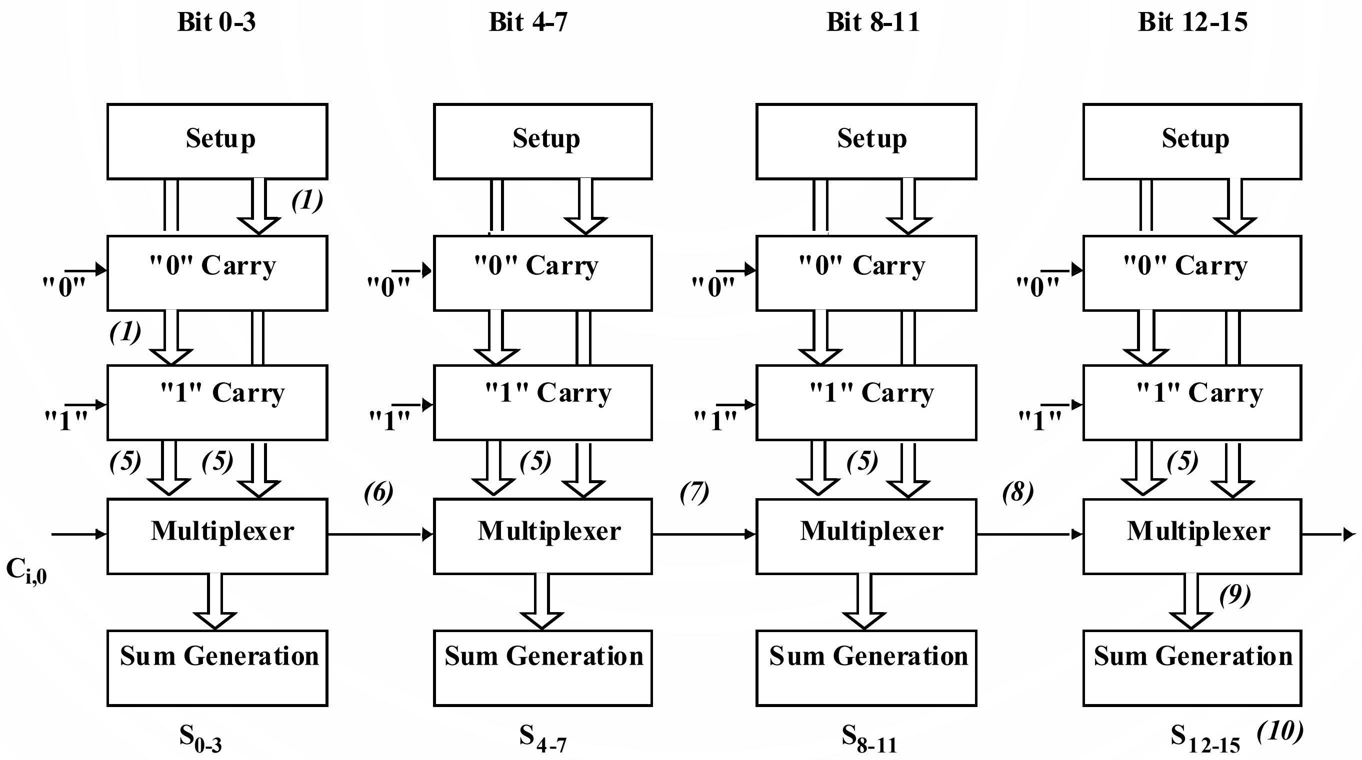
Carry-Select Adder



Carry Select Adder: Critical Path

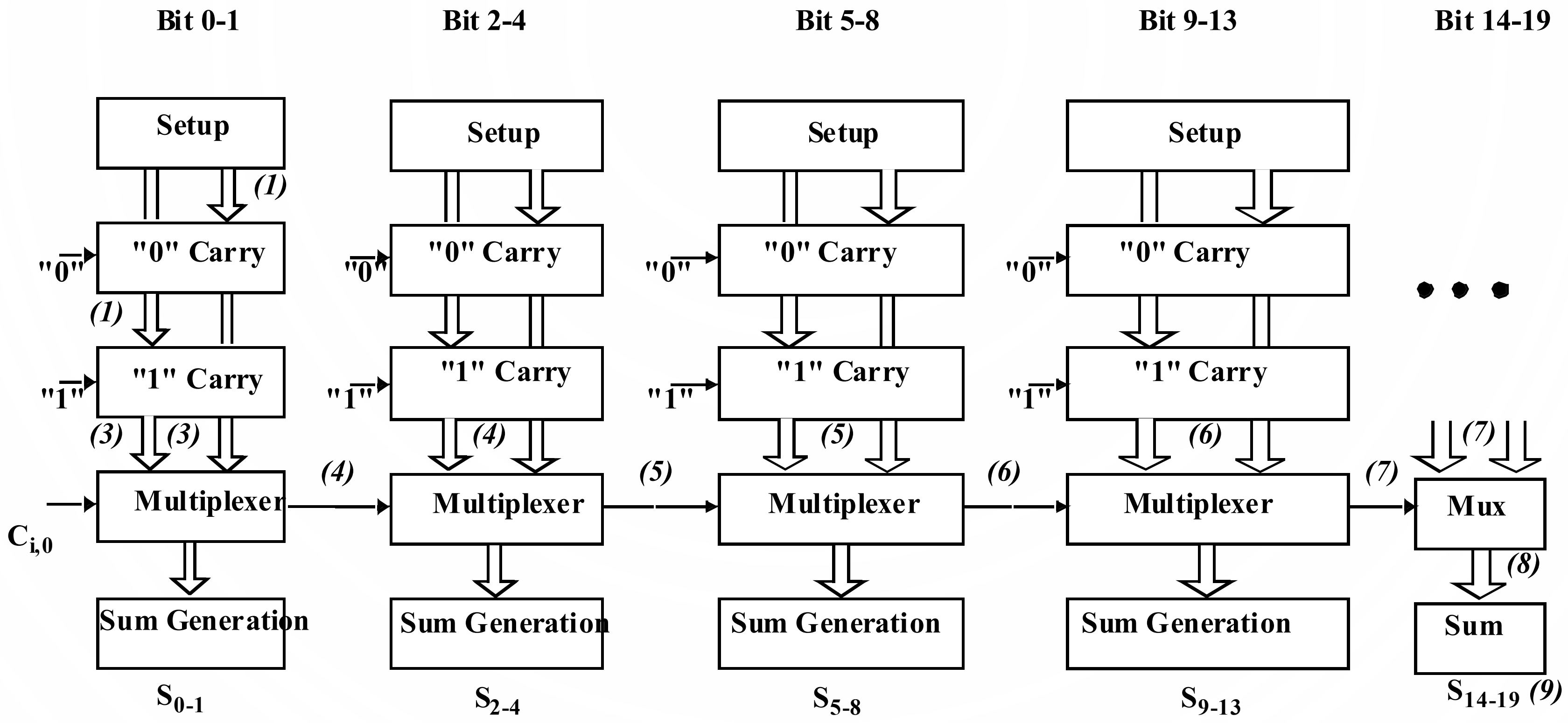


Linear Carry Select



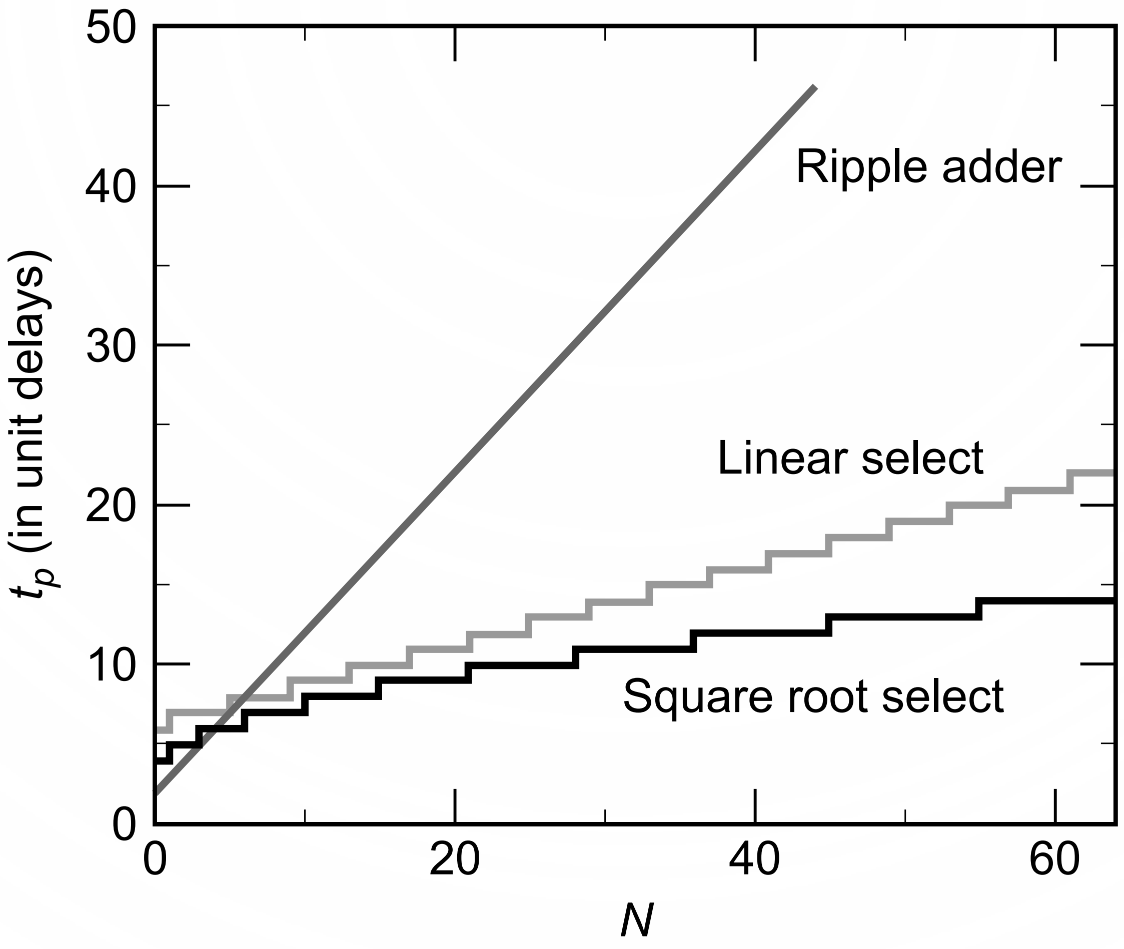
$$t_{add} = t_{setup} + \left(\frac{N}{M}\right) t_{carry} + M t_{mux} + t_{sum}$$

Square Root Carry Select



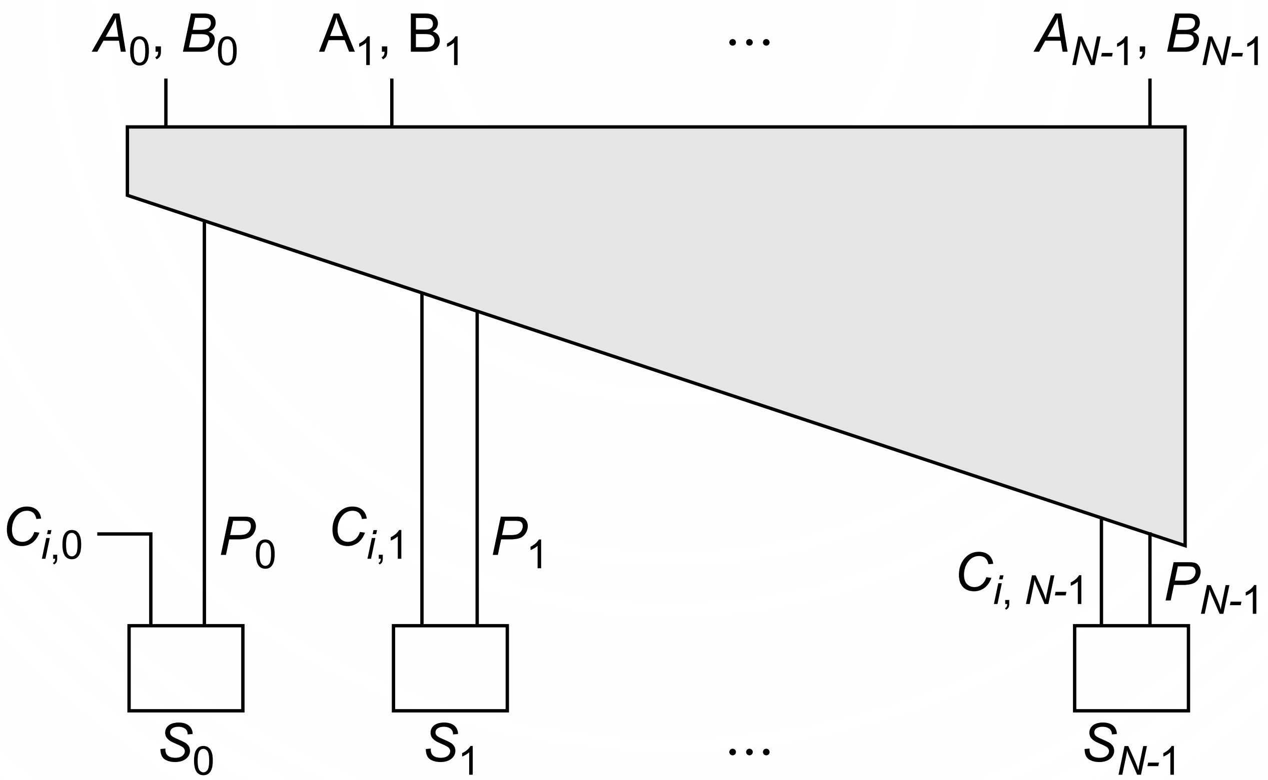
$$t_{add} = t_{setup} + P \cdot t_{carry} + (\sqrt{2N})t_{mux} + t_{sum}$$

Adder Delays - Comparison



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

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

Expanding lookahead equations:

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

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

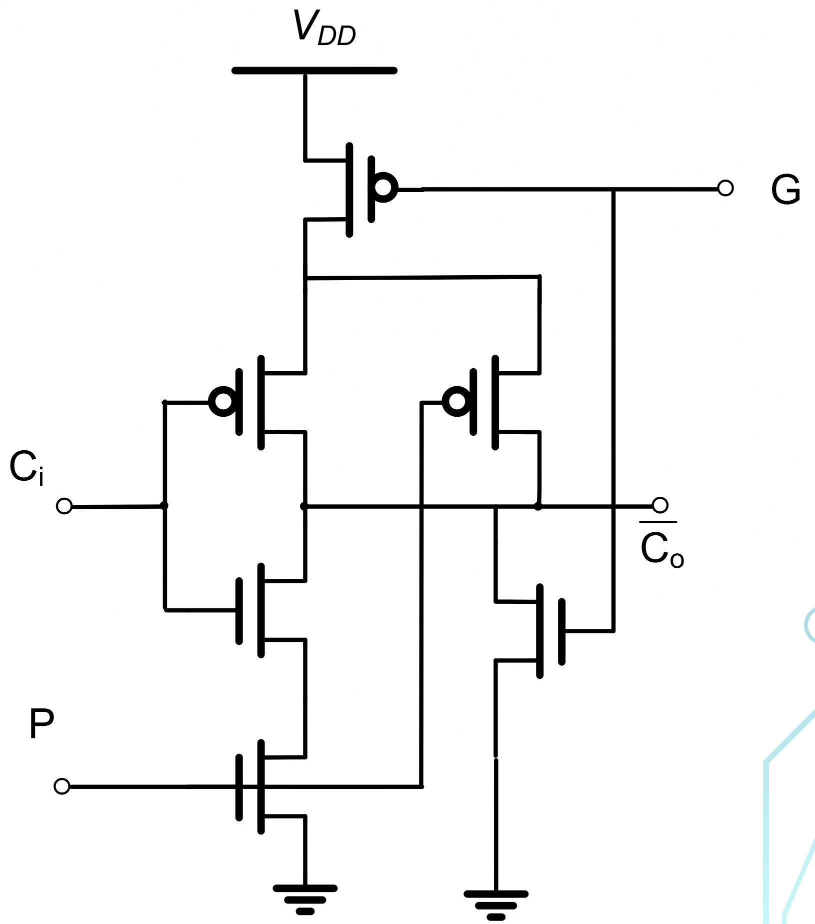
Carry at bit k:

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

Expanding at bit k:

$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}(\ldots + P_1(G_0 + P_0 C_{i,0})\ldots))$$

Carry-lookahead gate grows at each bit position!



Carry Lookahead Trees

Build the carrylookahead tree as a hierarchy of gates

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

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

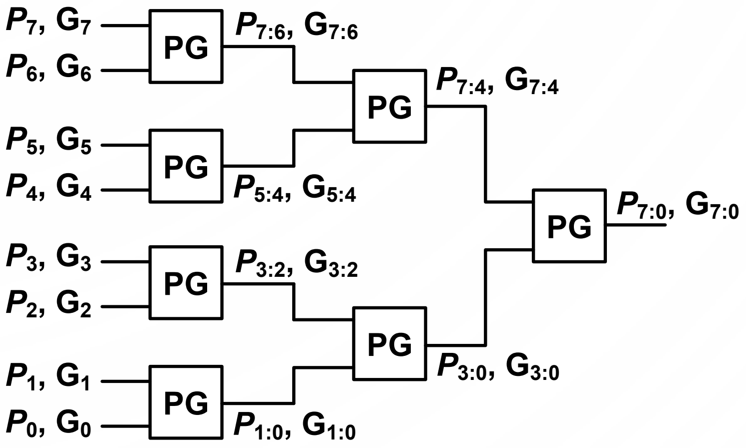
$C_{o,2} = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{i,0}$
$= (G_2 + P_2 G_1) + (P_2 P_1)(G_0 + P_0 C_{i,0}) = G_{2:1} + P_{2:1} C_{O,0}$

Can continue building the tree hierarchically.

Logarithmic (Tree) Adders – Idea

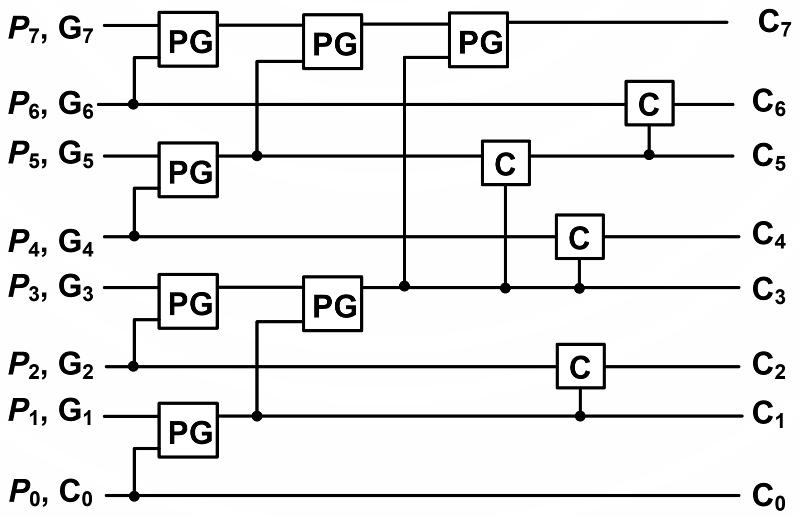
- "Look ahead" across groups of multiple bits to figure out the carry
  - Example with two bit groups:

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



Rest of the Tree

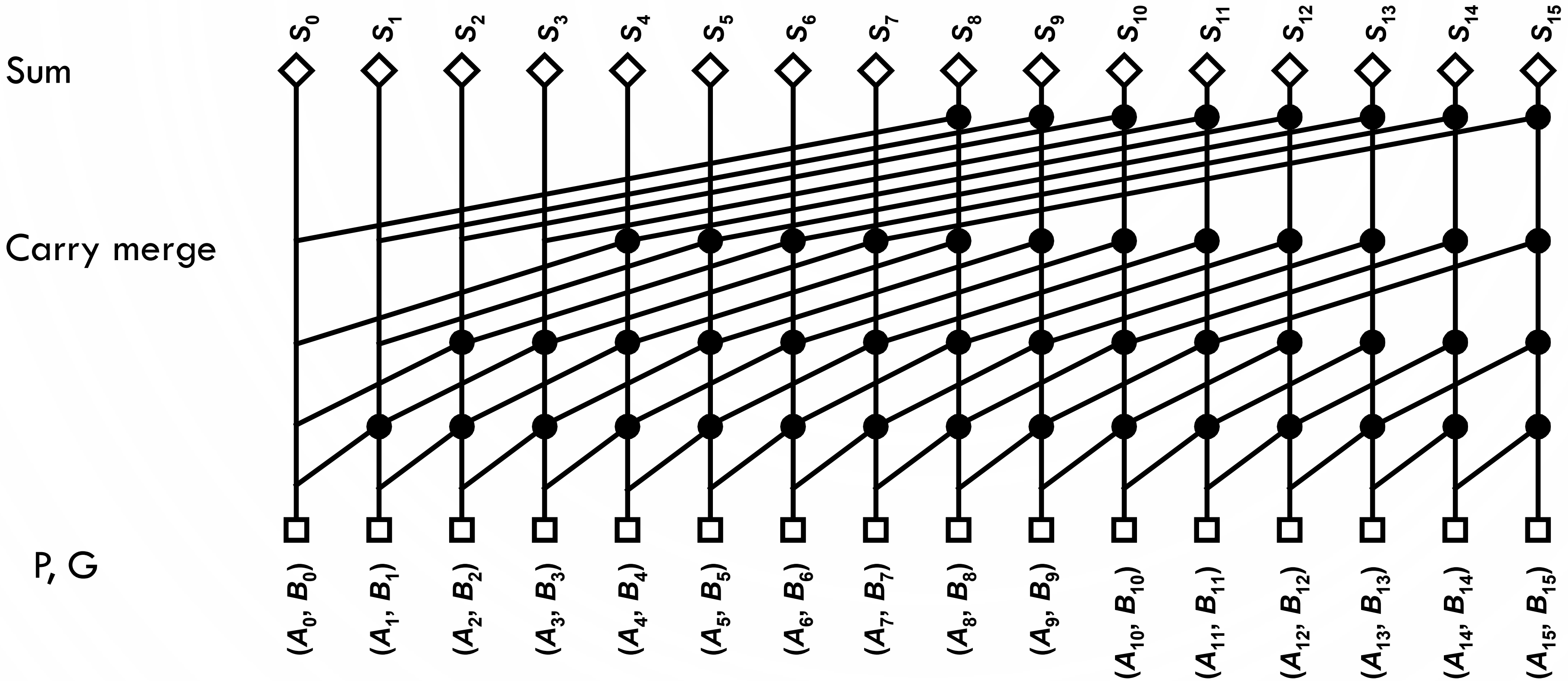
- Previous picture shows only half of the algorithm
  - Need to generate carries at individual bit positions too



Many Kinds of Tree Adders

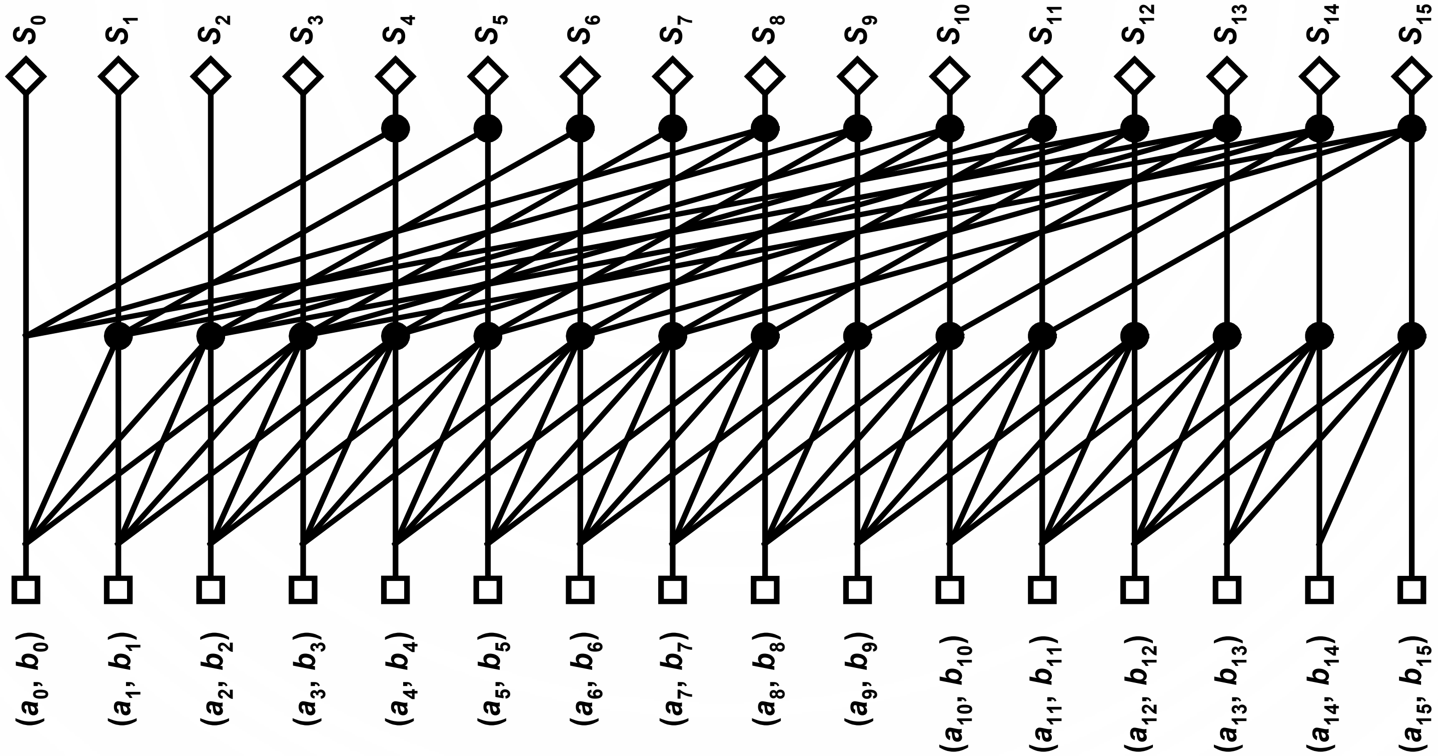
- Many ways to construct these tree (or “carry lookahead”) adders
  - Many of these variations named after the people who created them
- Most of these vary three basic parameters:
  - Radix: how many bits are combined in each PG gate
    - Previous example was radix 2; often go up to radix 4
  - Tree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

Tree Adders



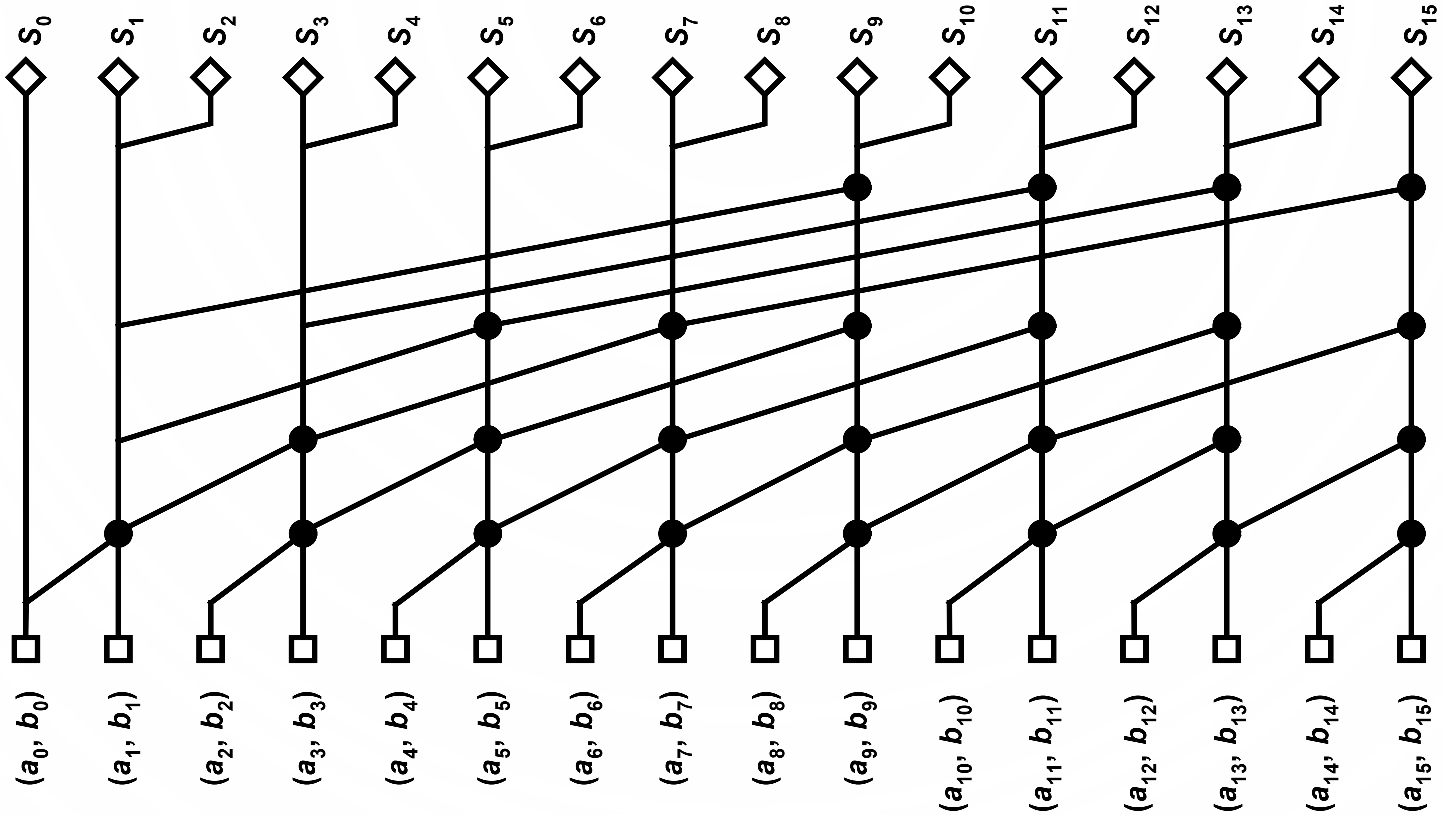
16-bit radix-2 Kogge-Stone tree

Tree Adders



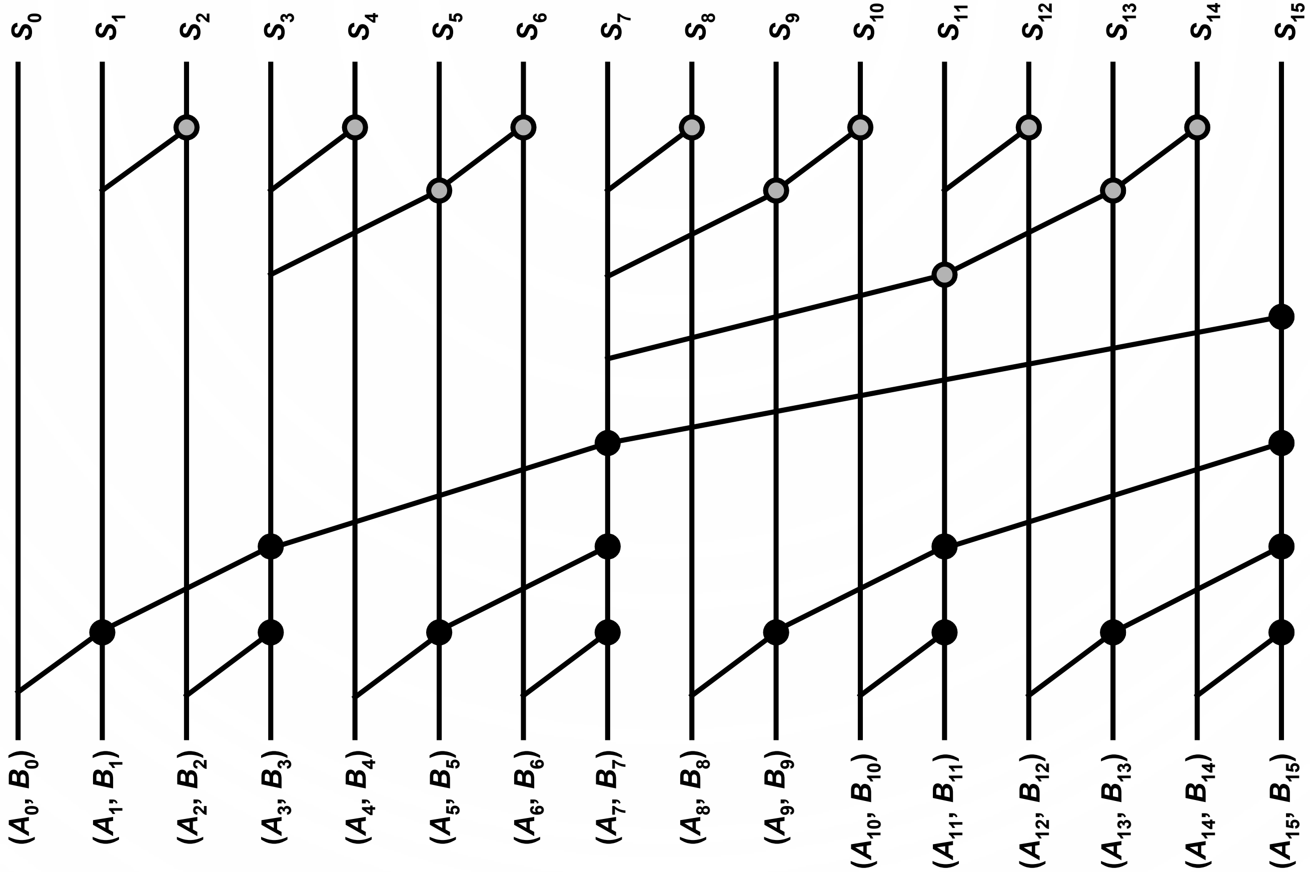
16-bit radix-4 Kogge-Stone Tree

Sparse Trees



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

Tree Adders



Brent-Kung Tree

Multipliers

Warmup

- Recall long multiplication of base-10 by hand:

$$\begin{array}{r} 12 \\ \times\ 56 \\ \hline \end{array}$$

- In base-2 (binary), we do the same thing:

$$\begin{array}{r} 011 \\ \times 101 \\ \hline \end{array}$$

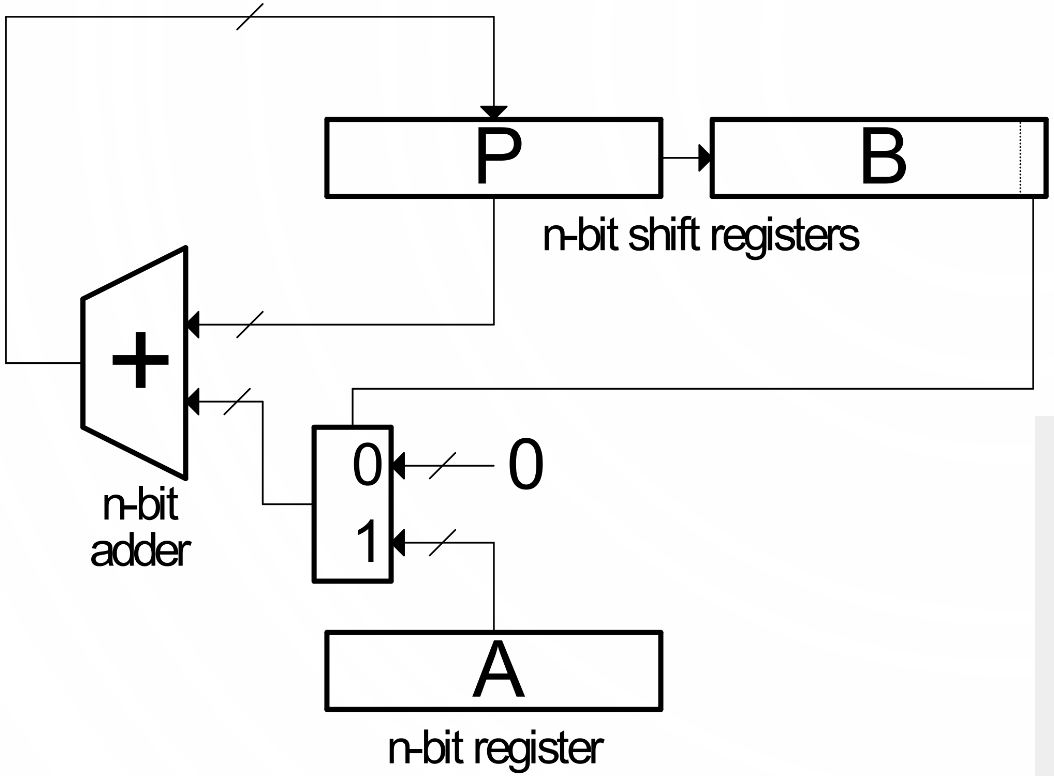
Multiplication

$$
\begin{array}{ccccccccl}
 & & & & a_3 & a_2 & a_1 & a_0 & \leftarrow \textit{Multiplicand} \\
 & & & \mathrm{X} & b_3 & b_2 & b_1 & b_0 & \leftarrow \textit{Multiplier} \\
\hline
 & & & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 & & \\
 & & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & & & \textit{Partial products} \\
 & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & & & \\
a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & & & \\
\hline
 & & & \ldots & & & a_1b_0+a_0b_1 & a_0b_0 & \leftarrow \textit{Product}
\end{array}
$$

Many different circuits exist for multiplication.

Each one has a different balance between speed (performance) and amount of logic (energy, cost).

"Shift and Add" Multiplier



- Sums each partial product, one at a time.
- In binary, each partial product is shifted versions of A or 0.

Control Algorithm:

1. P ← 0, A ← multiplicand, B ← multiplier
2. If LSB of B==1 then add A to P
   else add 0
3. Shift [P][B] right 1
4. Repeat steps 2 and 3 (n-1) more times.
5. [P][B] has product.

- Performance: N cycles of N-bit additions

"Shift and Add" Multiplier

Signed Multiplication:

Remember for 2's complement numbers MSB has negative weight:

$$X = \sum_{i=0}^{N-2} x_i 2^i - x_{n-1} 2^{n-1}$$

ex: $-6 = 11010_2 = 0•2^0 + 1•2^1 + 0•2^2 + 1•2^3 - 1•2^4$
$= 0 + 2 + 0 + 8 - 16 = -6$

- Therefore for multiplication:
  - a) subtract final partial product
  - b) sign-extend partial products
- Modifications to shift & add circuit:
  - a) adder/subtractor
  - b) sign-extender on P shifter register

Convince yourself

- What's -3 x 5?

$$\begin{array}{r} 1101 \\ \times\ 0101 \\ \hline \end{array}$$

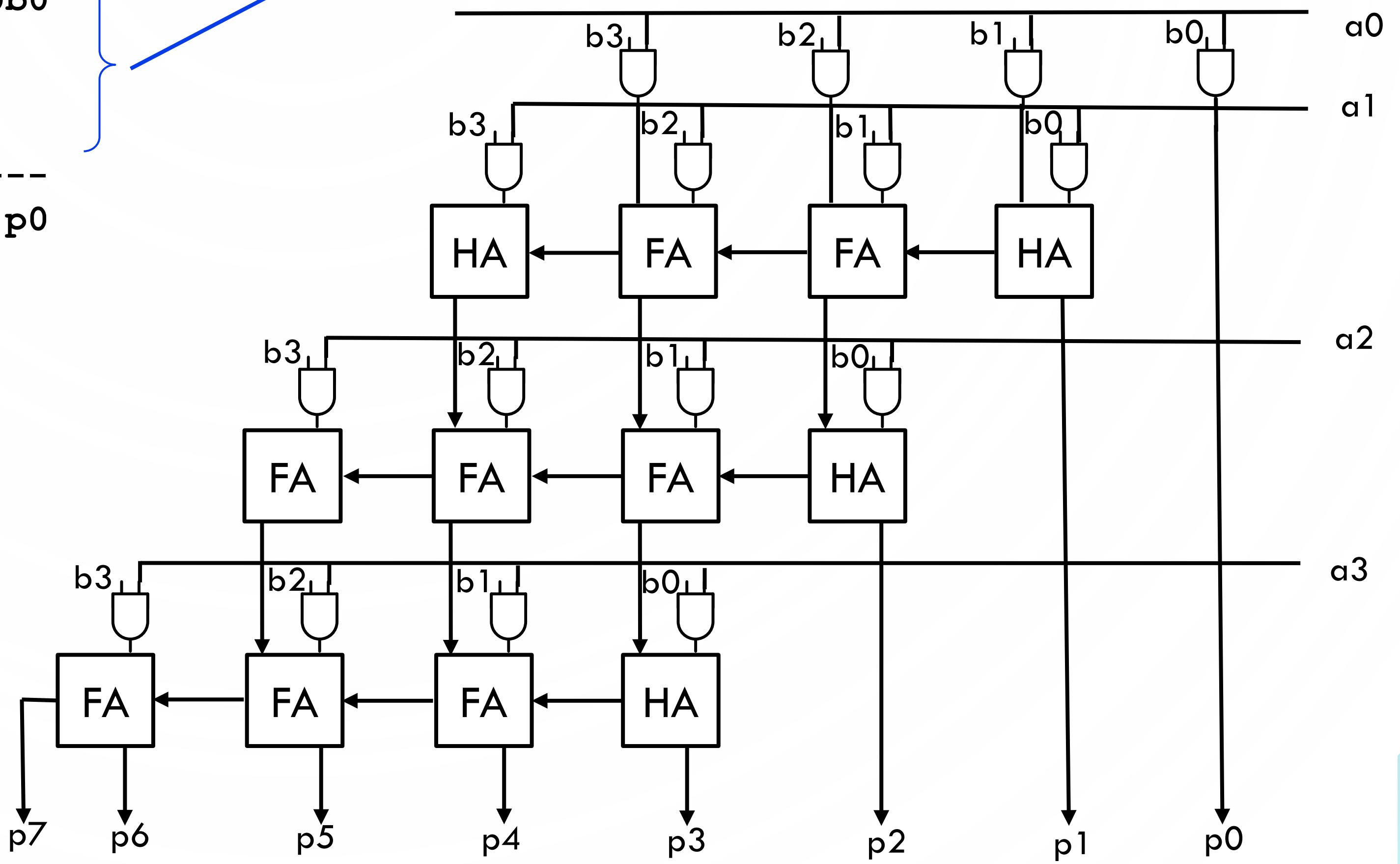
Unsigned Parallel Multiplier

Parallel (Array) Multiplier

```
                           a3   a2   a1   a0
                        *  b3   b2   b1   b0
                        --------------------
                         a3b0 a2b0 a1b0 a0b0
+                   a3b1 b2a1 a1b1 a0b1
+              a3b2 a2b2 a1b2 a0b2
+         a3b3 a2b3 a1b3 a0b3
-----------------------------------------
    p7   p6   p5   p4   p3   p2   p1   p0
```

multiplicand
multiplier

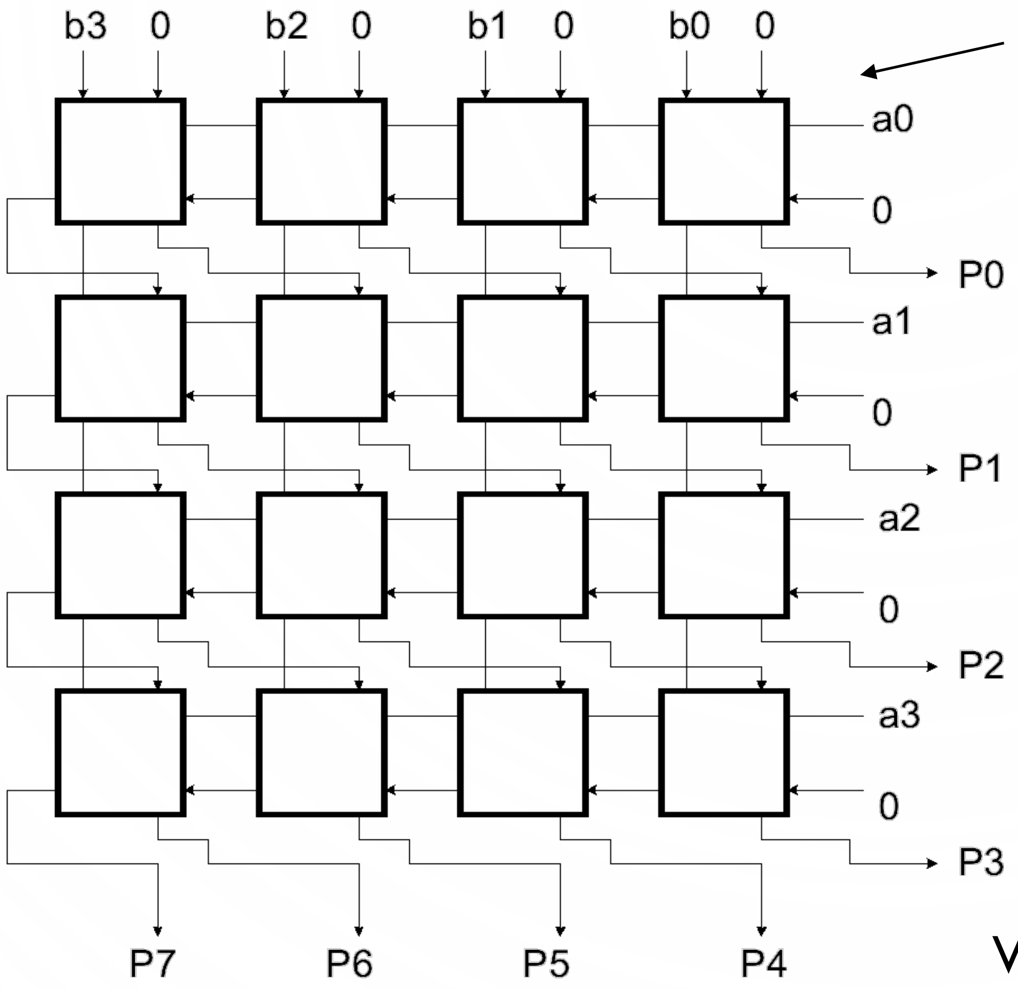
Partial products, one for each bit in multiplier
(each bit needs just one AND gate)



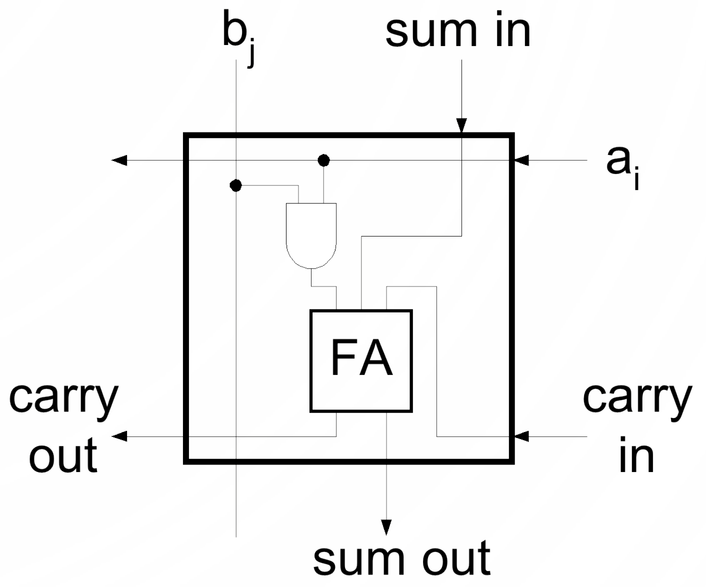
- Performance: What is the critical path?

Parallel (Array) Multiplier

Single cycle multiply: Generates all n partial products simultaneously.



Each row: n-bit adder with AND gates



What is the critical path?

Carry-Save Addition

- Speeding up multiplication is a matter of speeding up the summing of the partial products.
- "Carry-save" addition can help.
- Carry-save addition passes (saves) the carries to the output, rather than propagating them.

Example: sum three numbers,

3_{10} = 0011, 2_{10} = 0010, 3_{10} = 0011

$$
\begin{array}{rlcl}
3_{10} & 0011 & & \\
+\ 2_{10} & \underline{0010} & & \\
c & 0100 & = & 4_{10} \\
s & 0001 & = & 1_{10} \\
+\ 3_{10} & \underline{0011} & & \\
c & 0010 & = & 2_{10} \\
s & \underline{0110} & = & 6_{10} \\
 & 1000 & = & 8_{10}
\end{array}
$$

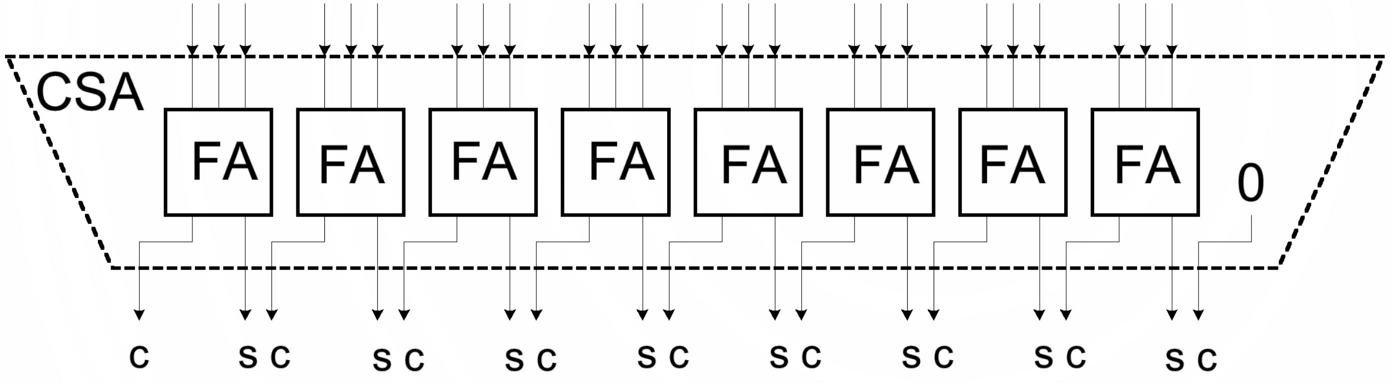
carry-save add

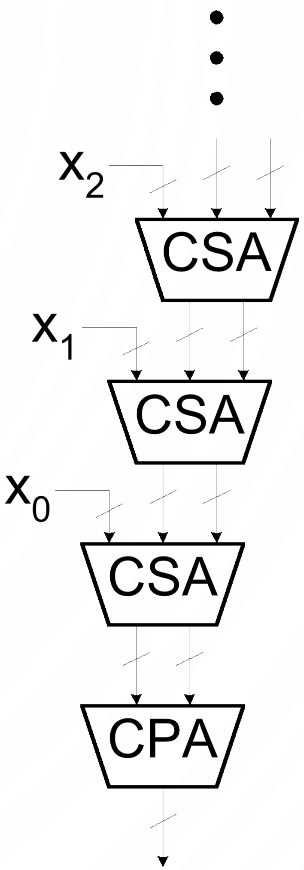
carry-save add

carry-propagate add

- In general, *carry-save* addition takes in 3 numbers and produces 2: "3:2 compressor":
- Whereas, *carry-propagate* takes 2 and produces 1.
- With this technique, we can avoid carry propagation until final addition

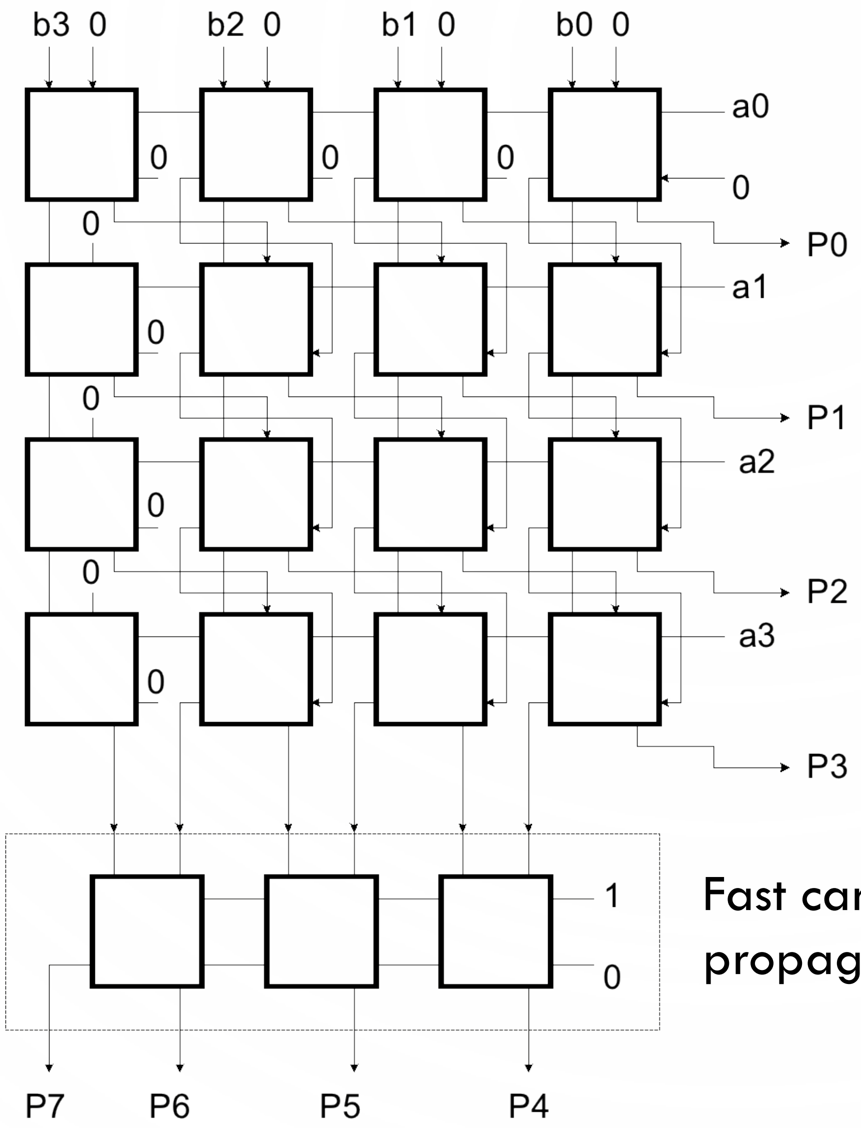
Carry-Save Circuits



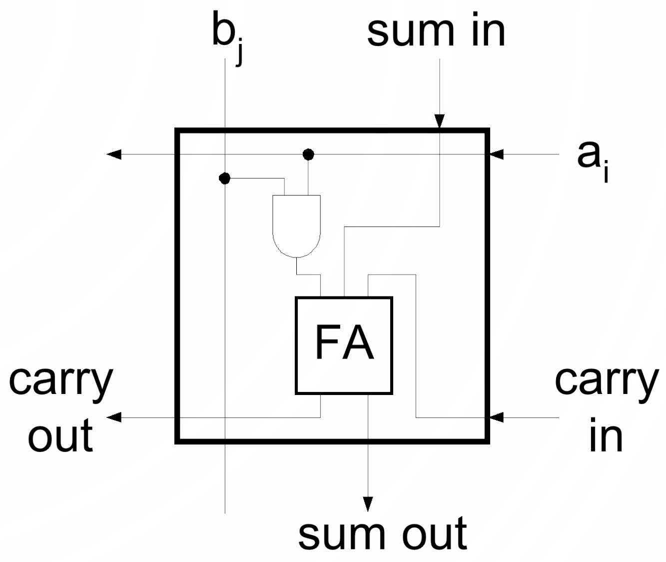


- When adding sets of numbers, carry-save can be used on all but the final sum.
- Standard adder (carry propagate) is used for final sum.
- Carry-save is fast (no carry propagation) and inexpensive (full adders)

Array Multiplier Using Carry-Save Addition



Fast carry-propagate adder

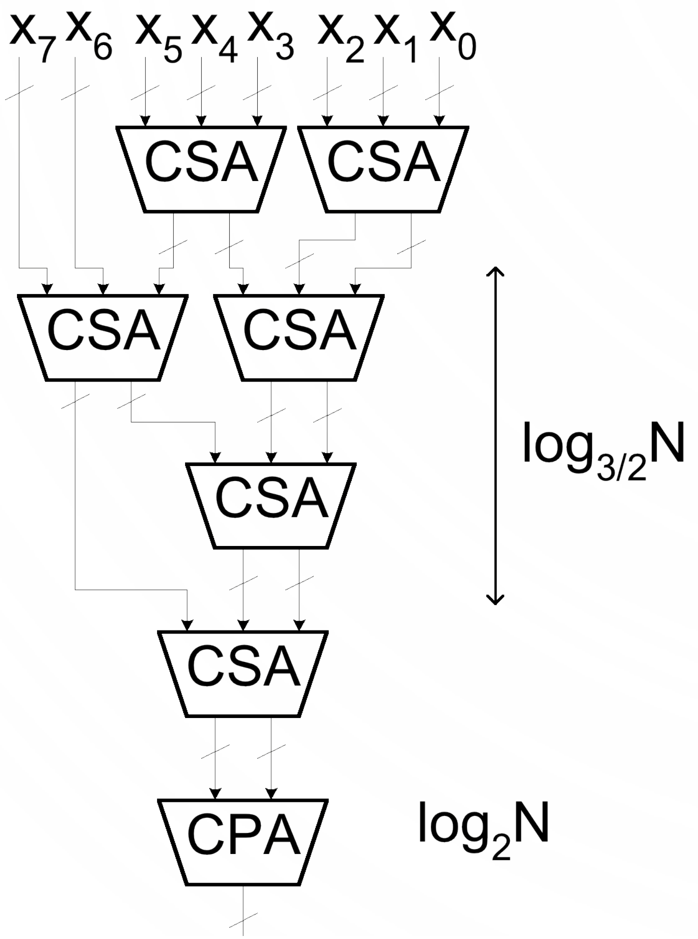


- What is the critical path?

Carry-Save Addition

CSA is associative and commutative. For example:

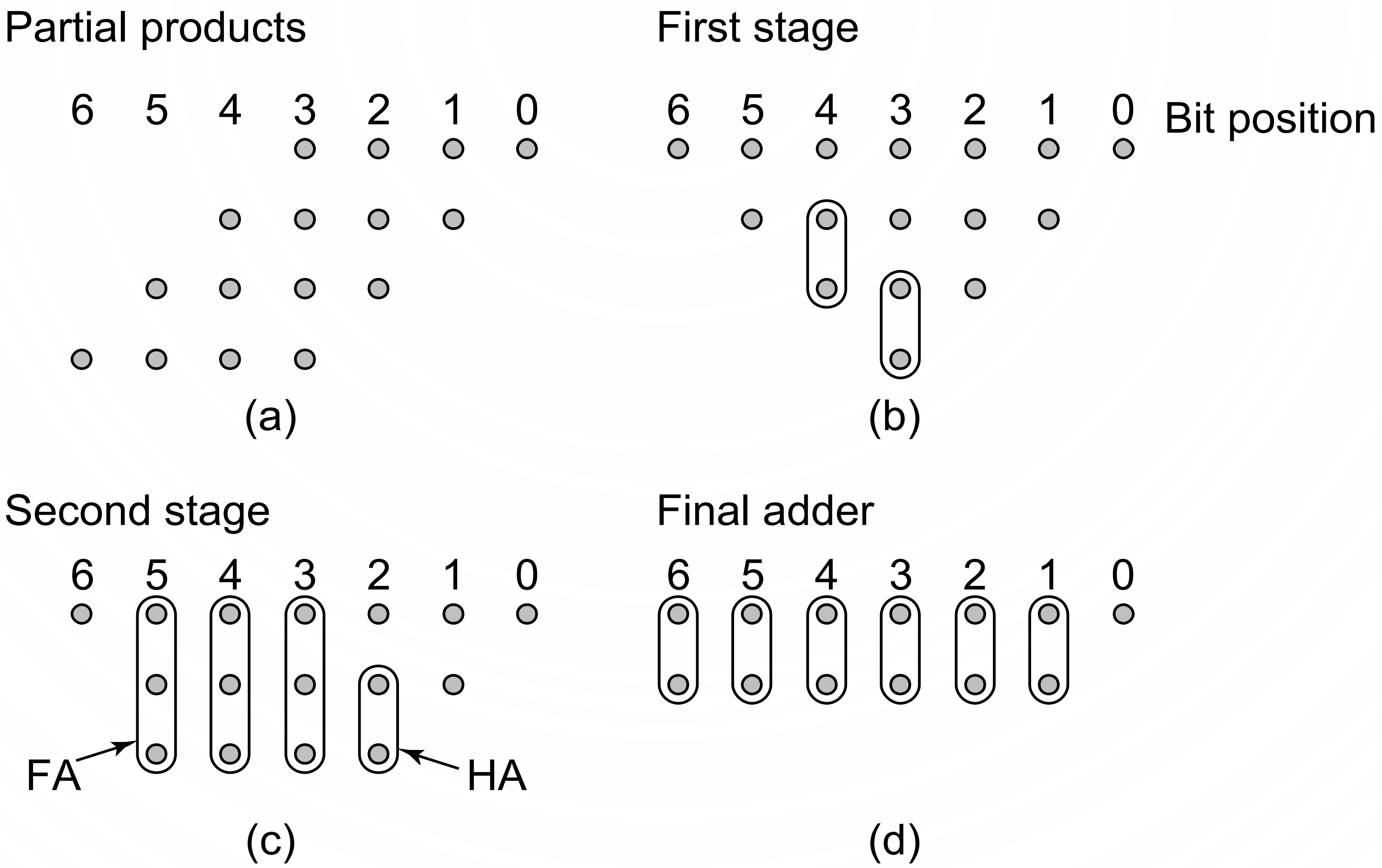
$$(((X_0 + X_1) + X_2) + X_3) = ((X_0 + X_1) + (X_2 + X_3))$$



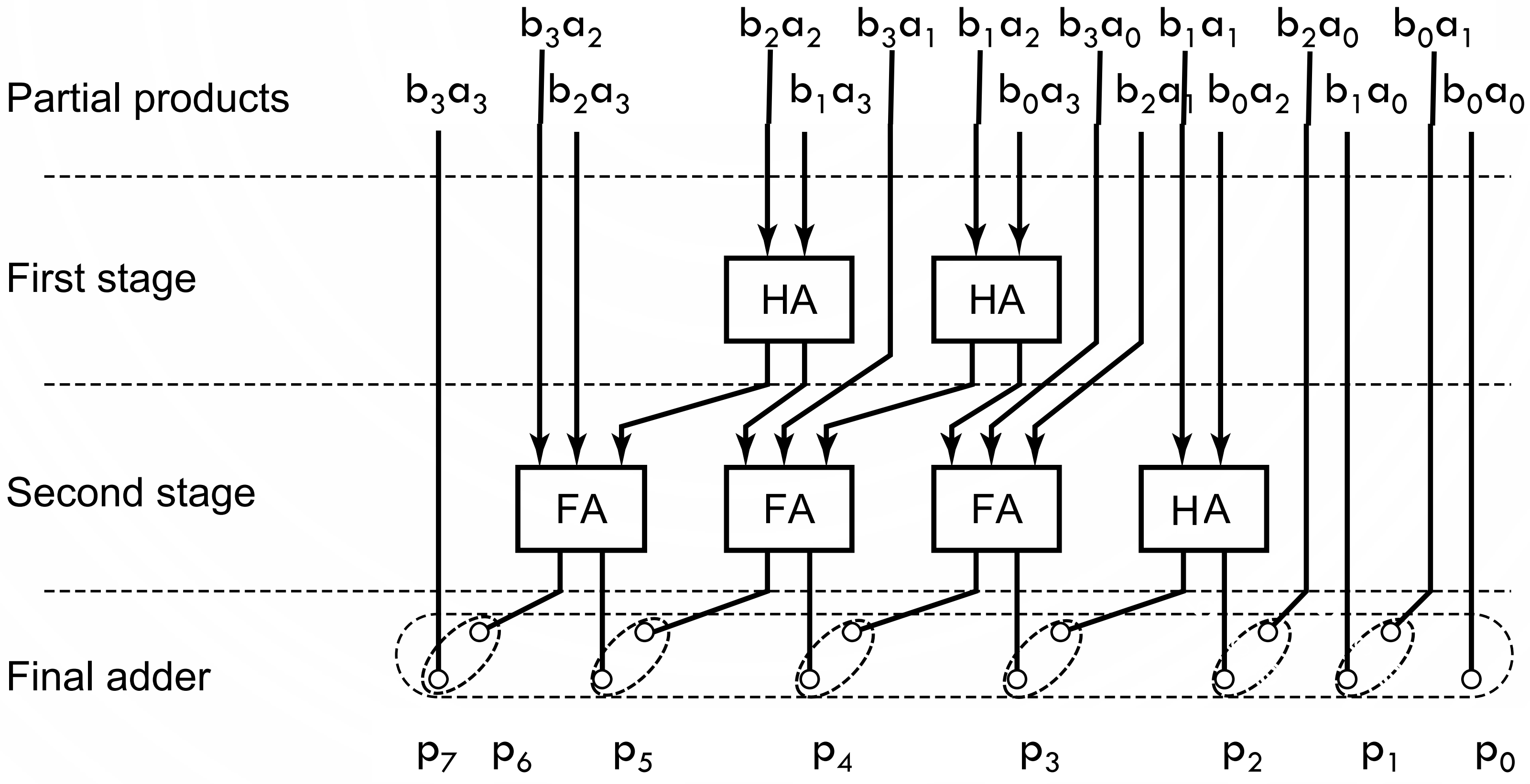
- A balanced tree can be used to reduce the logic delay
- It doesn't matter where you add the carries and sums, as long as you eventually do add them
- This structure is the basis of the ***Wallace Tree Multiplier***
- Partial products are summed with the CSA tree. Fast adder (ex: CLA) is used for final sum
- Multiplier delay α $\log_{3/2}N + \log_2 N$

Wallace-Tree Multiplier

- Reduce the partial products in logic stages – 4 x 4 example



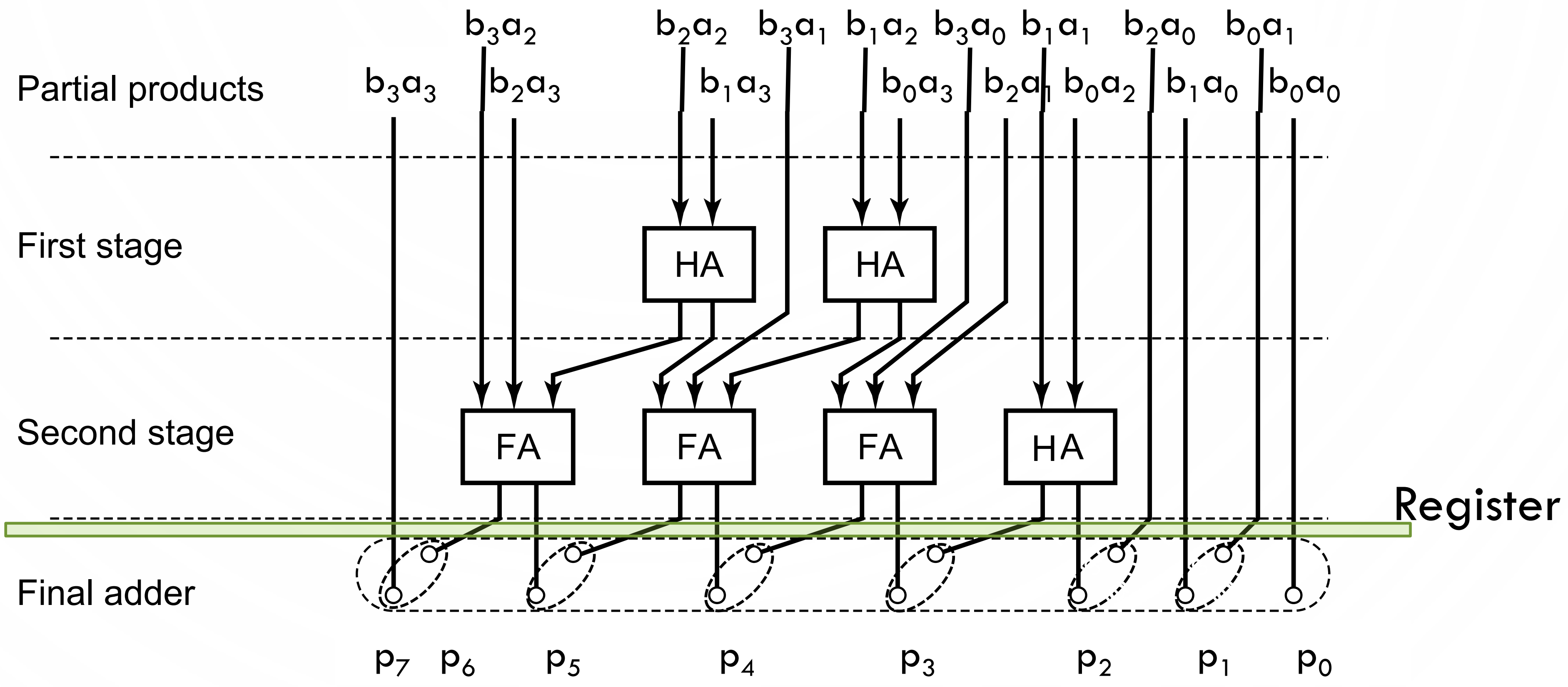
Wallace-Tree Multiplier



Note: Wallace tree is often slower than an array multiplier in FPGAs (which have optimized carry chains)

Increasing Throughput: Pipelining

- Multipliers have a long critical path: PP generation → reduction tree → final adder
  - Often pipelined before final adder (2x flip-flops for carry-save)



Summary

- Adders
  - Carry is in the adder critical path
  - Mirror adders cells are commonly found in libraries
  - Ripple-carry adder is the least complex, lowest energy
  - Carry-bypass, carry-select are usually faster than ripple-carry for bitwidths > 8
- Multipliers
  - Shift-and-add is the most compact
  - Parallel multipliers