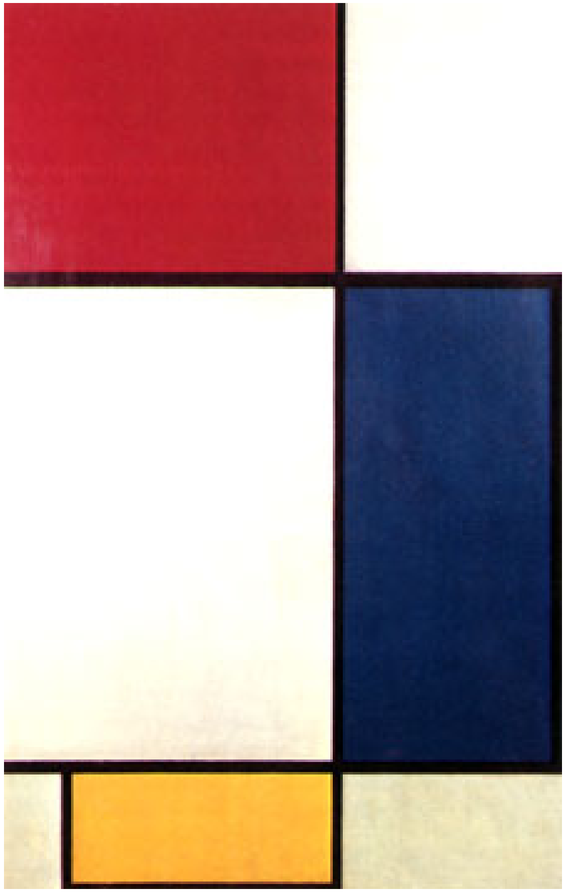
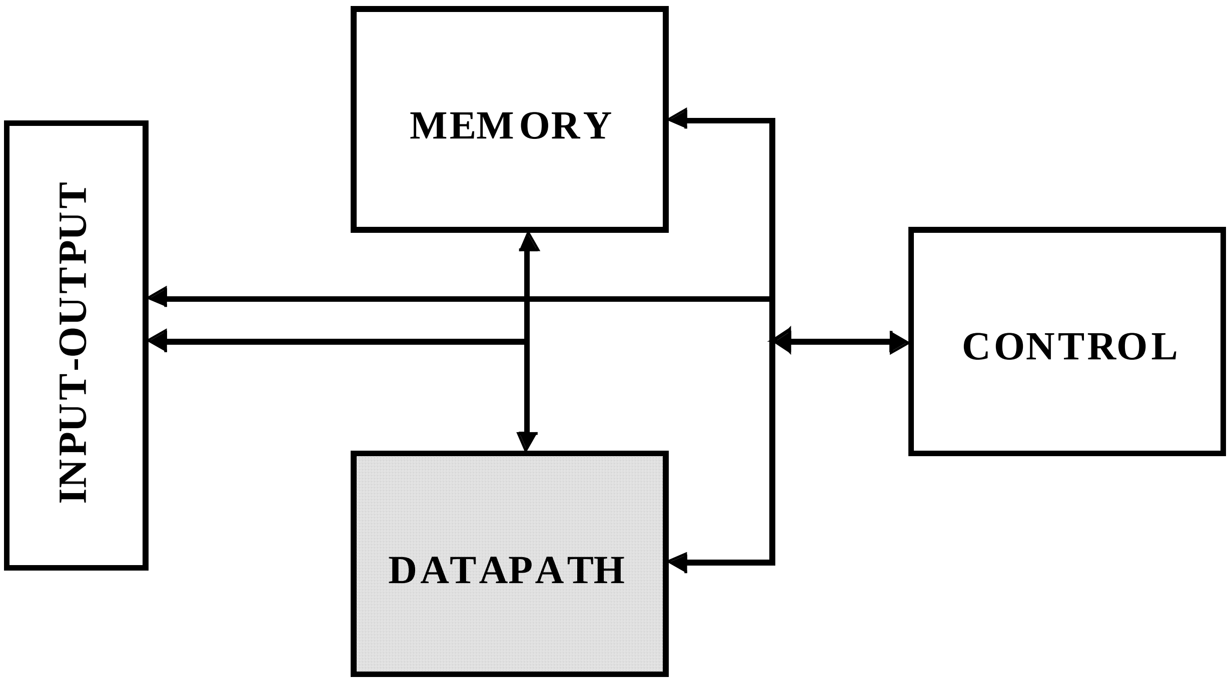
Arithmetic Building Blocks
Chapter 11 Rabaey



A Generic Digital Processor



Building Blocks for Digital Architectures

Datapath (Arithmetic Unit)

- Bit-sliced datapath (adder , multiplier, shifter, comparator, etc.)

Memory

- RAM, ROM, Buffers, Shift registers

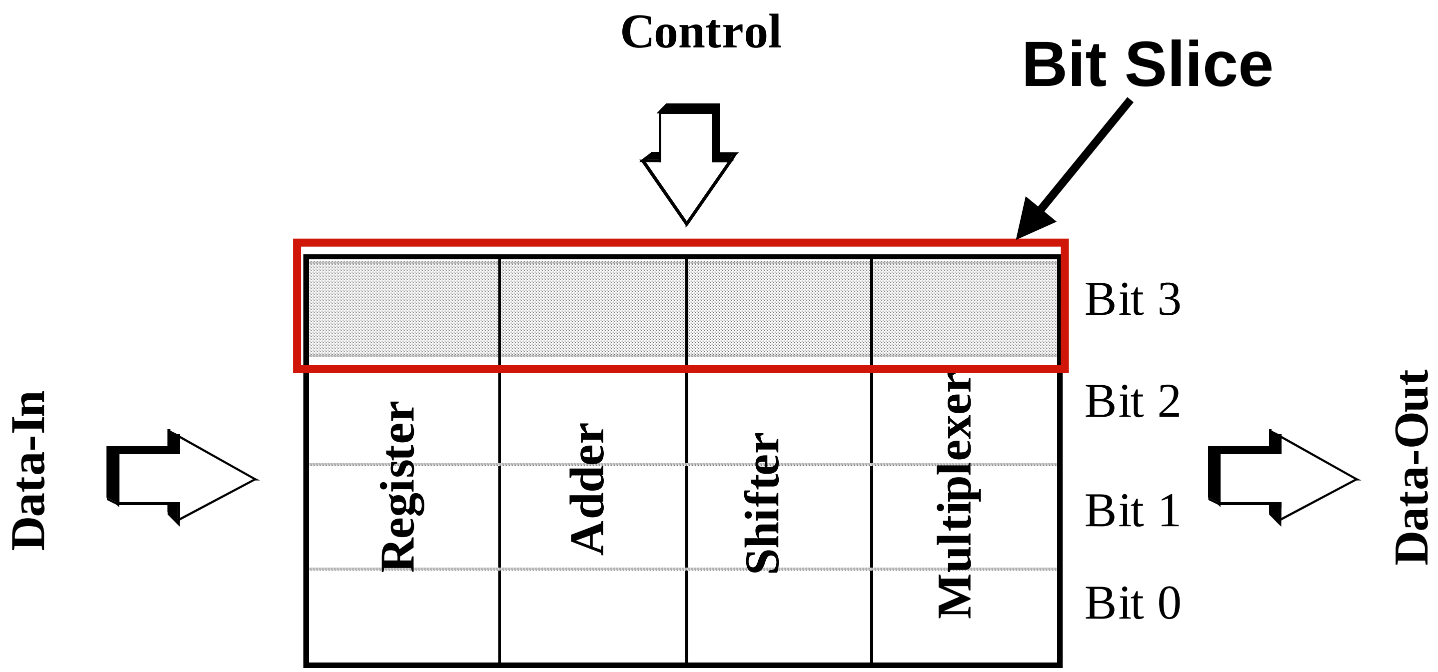
Control

- Finite state machine (PLA, random logic.)
- Counters

Interconnect

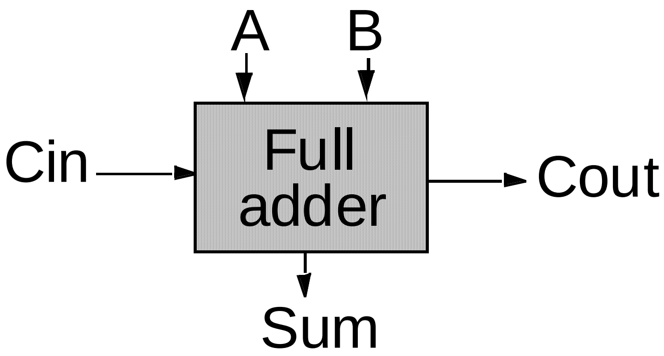
- Switches
- Arbiters
- Bus

Bit-Sliced Design



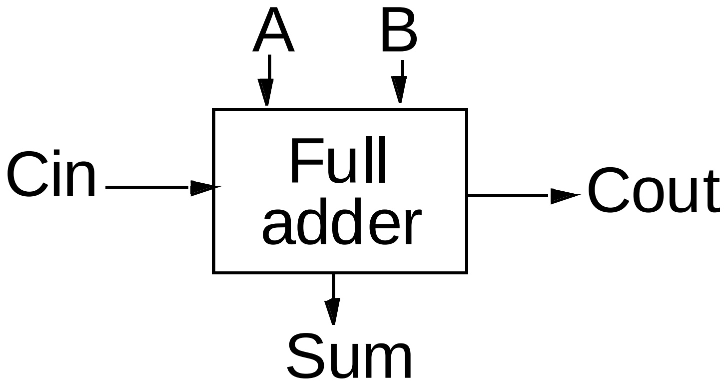
Tile identical processing elements

Full Adder



A	*B*	C_i	*S*	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

The Binary Adder

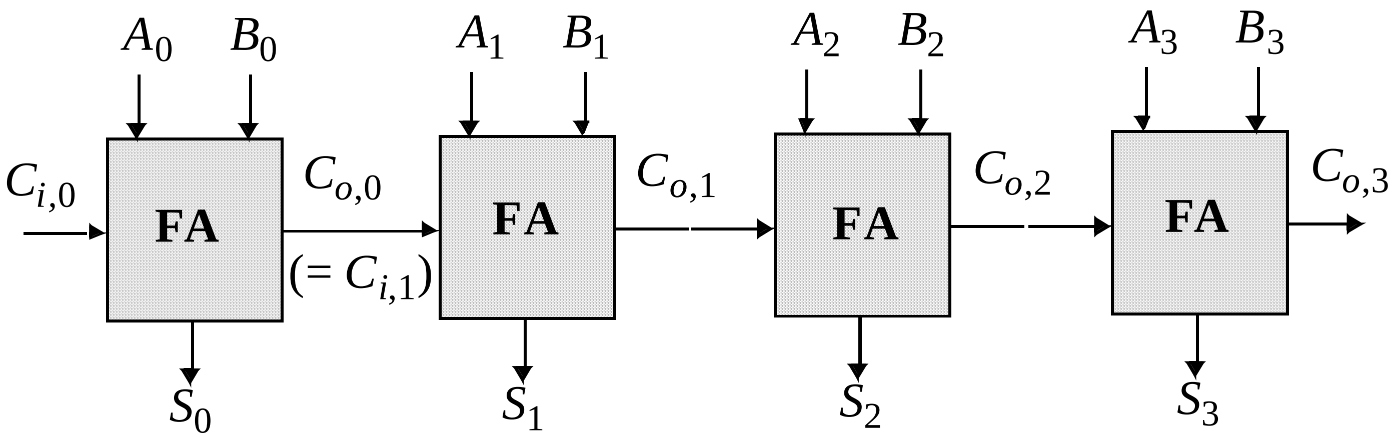


$$S = A \oplus B \oplus C_i$$

$$= A\bar{B}\bar{C}_i + \bar{A}B\bar{C}_i + \bar{A}\bar{B}C_i + ABC_i$$

$$C_o = AB + BC_i + AC_i$$

The Ripple-Carry Adder



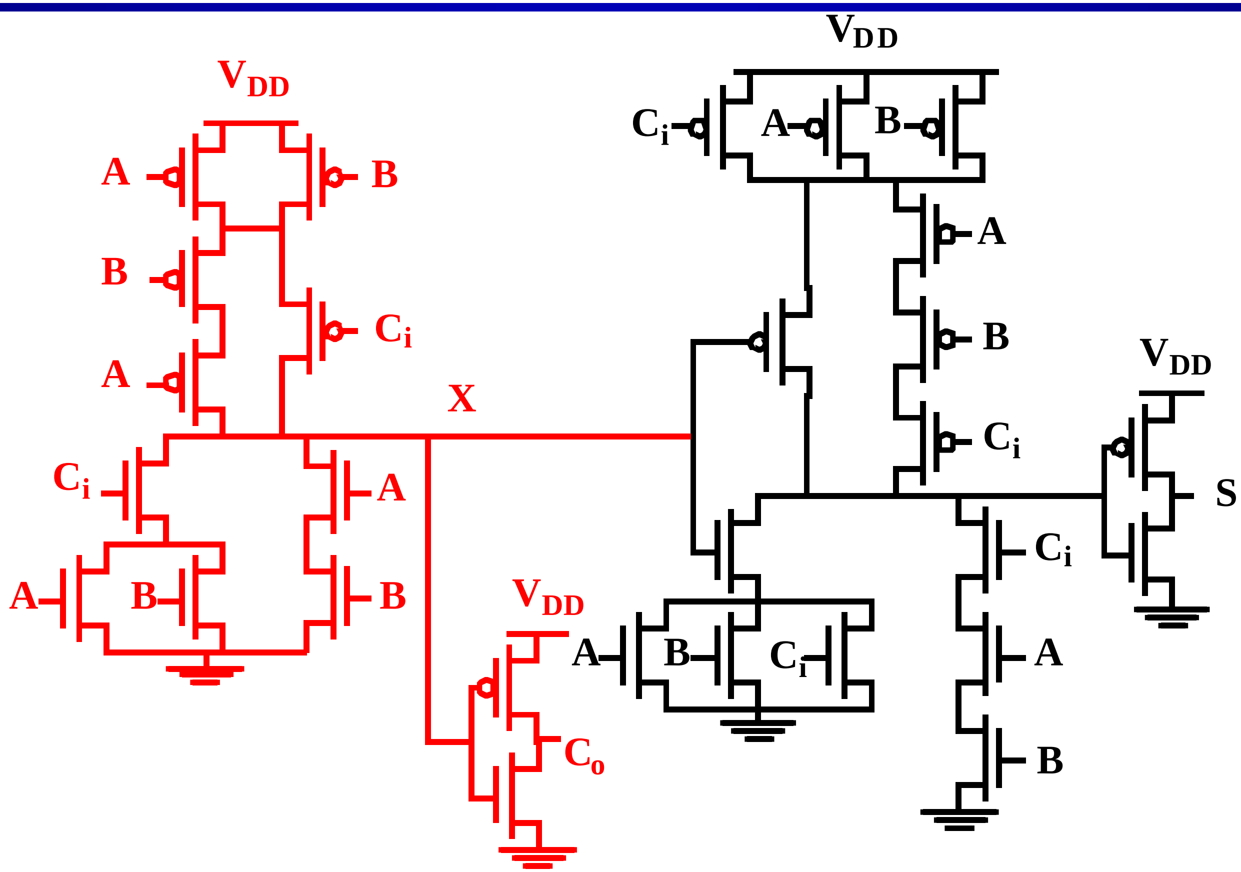
Worst case delay linear with the number of bits

$t_d = O(N)$

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

Goal: Make the fastest possible carry path circuit

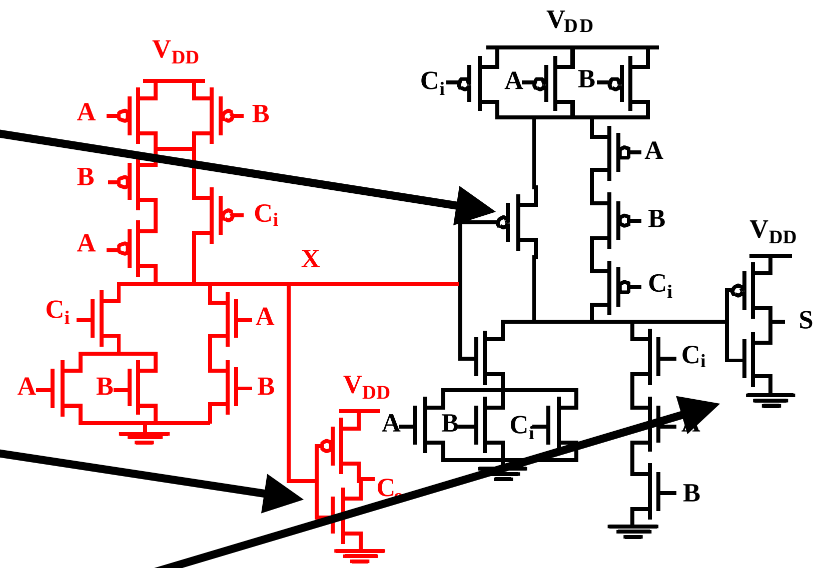
Complimentary Static CMOS Full Adder



28 Transistors

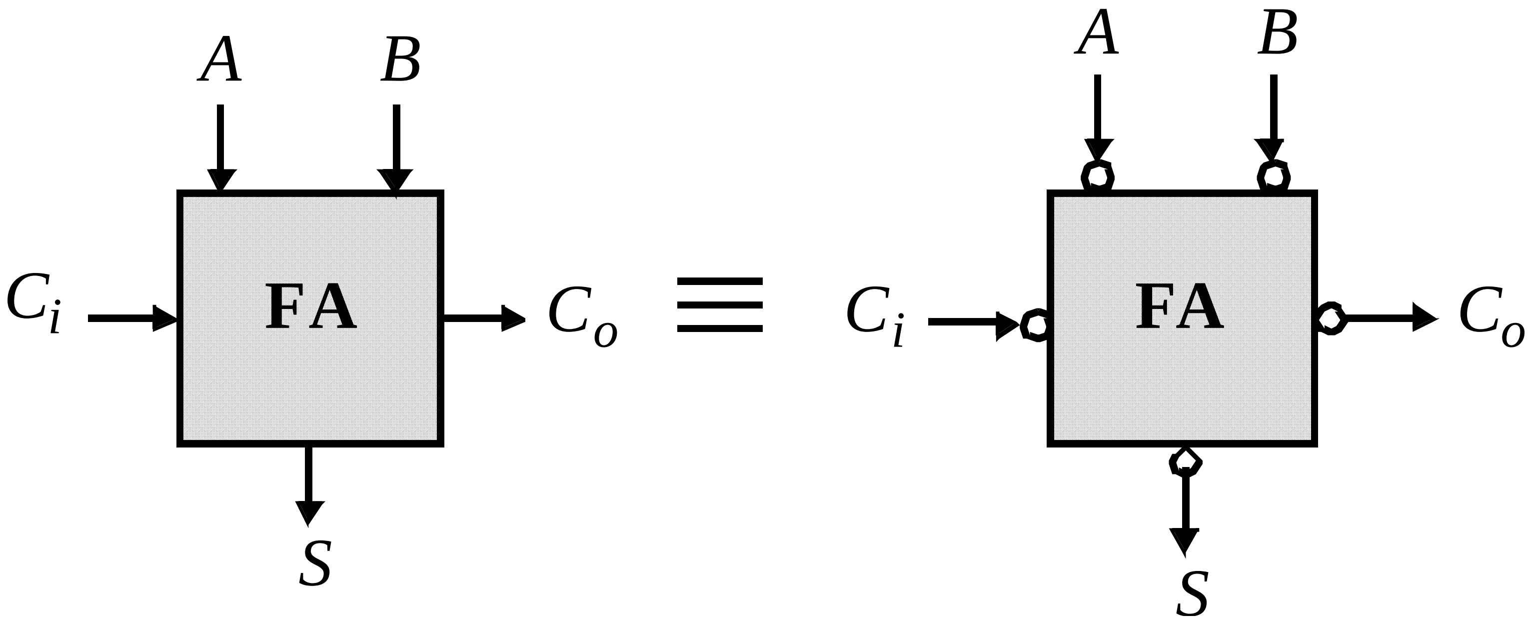
A Closer Look

- Drawbacks
  - » Tall PMOS Stack
    - – Slows down circuit
  - » C_o load is 2 diffusion and 6 gate capacitances
  - » Ci goes through the extra output inverter to Co
    - – Could optimize with next stage
  - » Sum generation has extra inverter on output
    - – Not the critical path
- Positive
  - » Ci closest to output node



28 Transistors

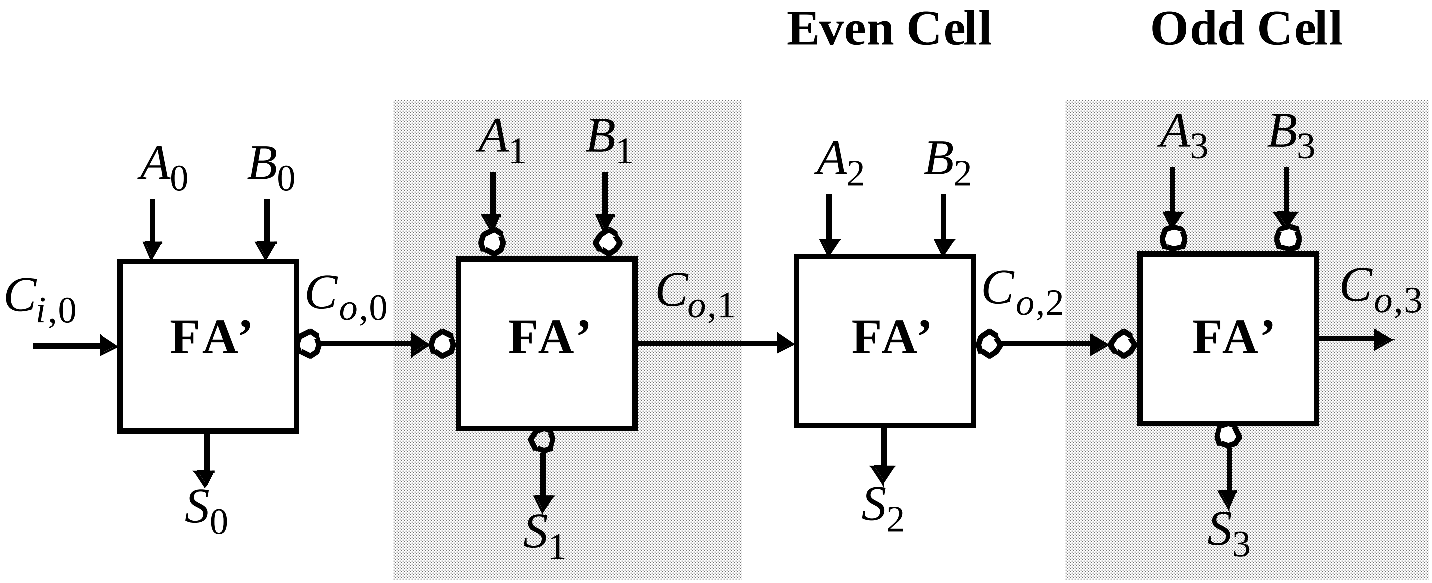
Inversion Property



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

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

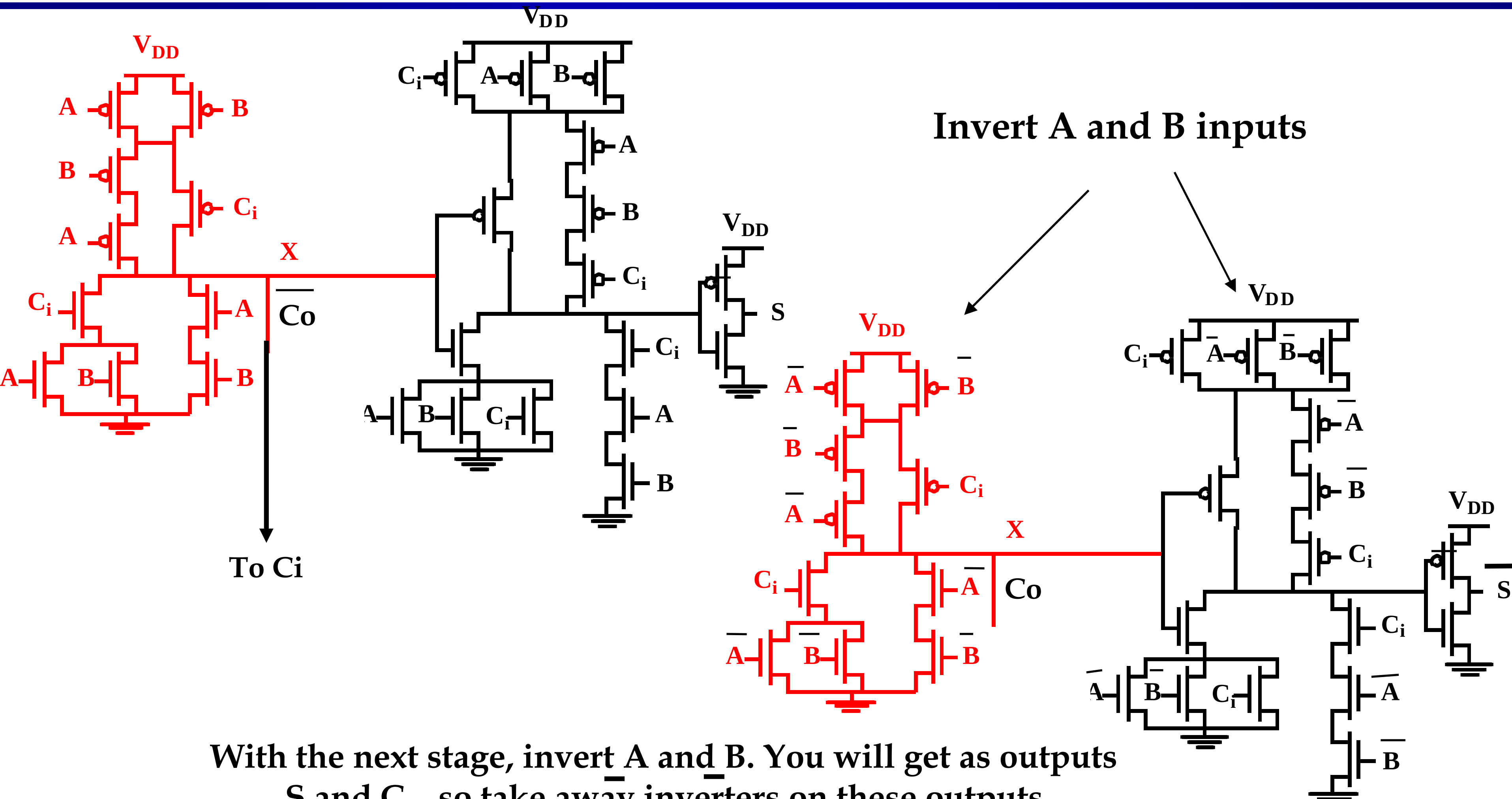
Minimize Critical Path by Reducing Inverting Stages



Exploit Inversion Property

Note: need 2 different types of cells

Applying Inversion Property



With the next stage, invert A and B. You will get as outputs
S and C…so take away inverters on these outputs.

Express Sum and Carry as Function of P, G, D

Define 3 new variable which ONLY depend on A, B

Generate (G) = AB $\mathbf{C_0 = 1 \text{ if } G = 1}$

Propagate (P) = A ⊕ B $\mathbf{C_0 = C_i \text{ if } P = 1}$

Delete = $\overline{A}\,\overline{B}$ $\mathbf{C_0 = 0 \text{ if } D = 1}$

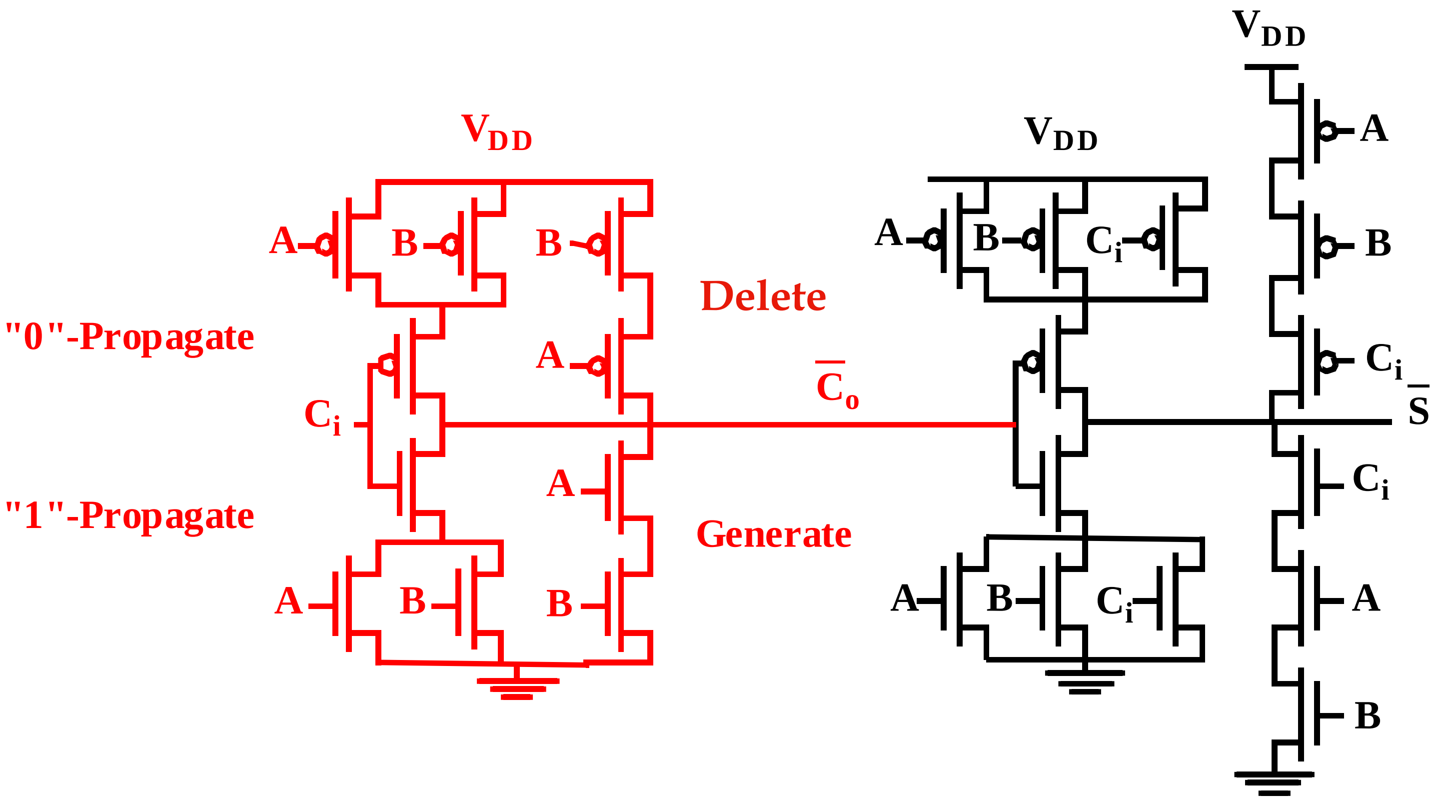
A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

$$C_o(G, P) = G + PC_i$$

$$S(G, P) = P \oplus C_i$$

Can also derive expressions for S and C_o based on D and P

A Better Structure: the Mirror Adder



24 transistors

The Mirror Adder I

- **The NMOS and PMOS chains are completely symmetrical. This guarantees identical rising and falling transitions if the NMOS and PMOS devices are properly sized. A maximum of two series transistors can be observed in the carry-generation circuitry.**
- **When laying out the cell, the most critical issue is the minimization of the capacitance at node C_o. The reduction of the diffusion capacitances is particularly important.**
- **The capacitance at node C_o is composed of four diffusion capacitances, two internal gate capacitances, and six gate capacitances in the connecting adder cell .**

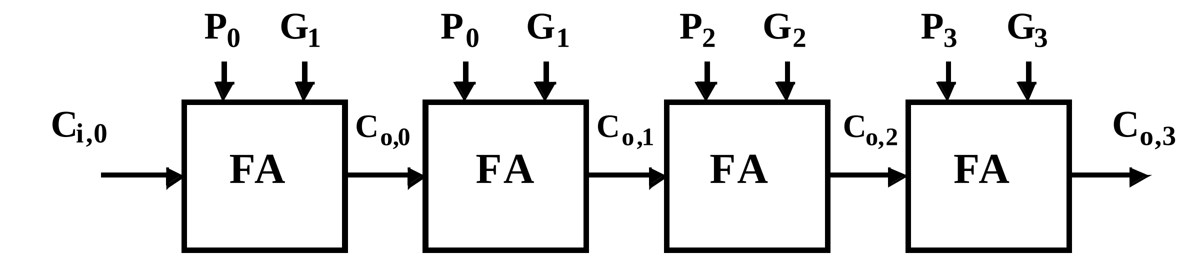
The Mirror Adder II

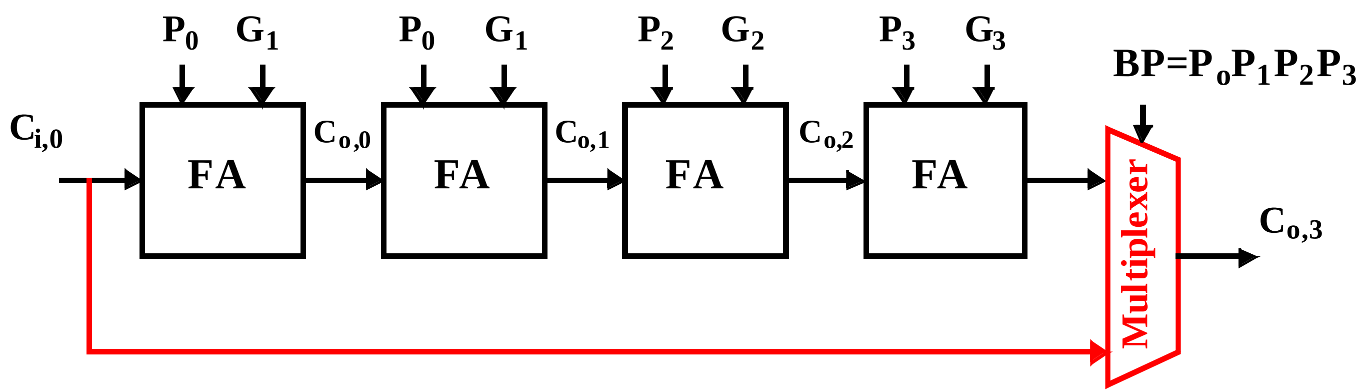
- **The transistors connected to C_i are placed closest to the output.**
  - **Fastest for late arriving inputs, C_i tends to arrive late**
- **Only the transistors in the carry stage have to be optimized for optimal speed. All transistors in the sum stage can be minimal size.**

Adder Architectures

- In addition to optimizing each full adder cell and exploiting inversion property, we can also reorganize the add computation to speed things up
- Basic idea is to overlap propagating the carry with computing the Propagate and Generate functions
- Discuss three basic architectures
  - Carry-Bypass
  - Carry-Select
  - Carry-Lookahead

Carry-Bypass Adder

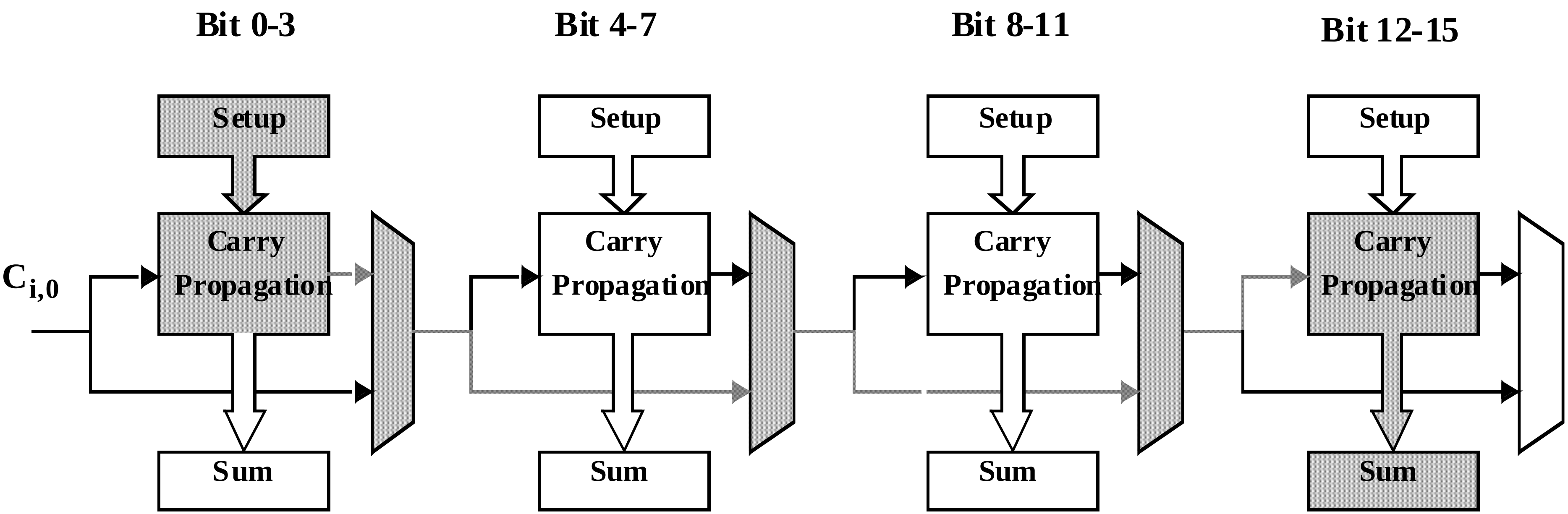




Idea: If (P0 and P1 and P2 and P3 = 1)

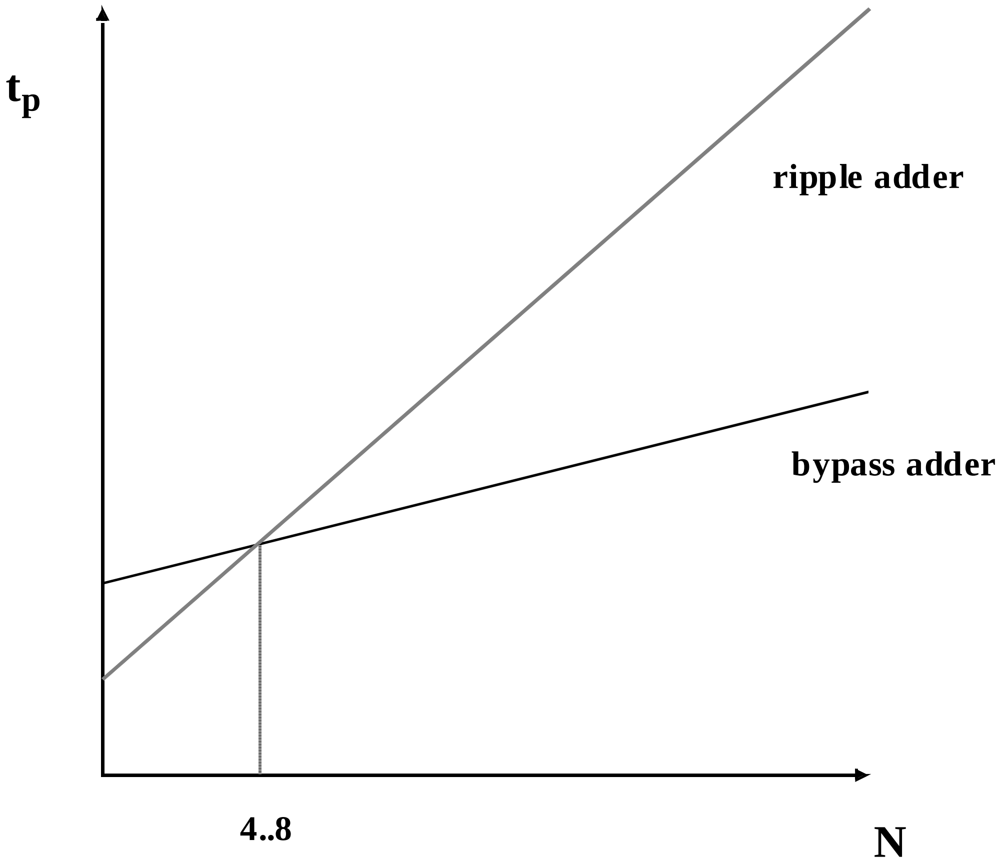
then $C_{o3} = C_0$, else “kill” or “generate”.

Carry-Bypass Adder (cont.)



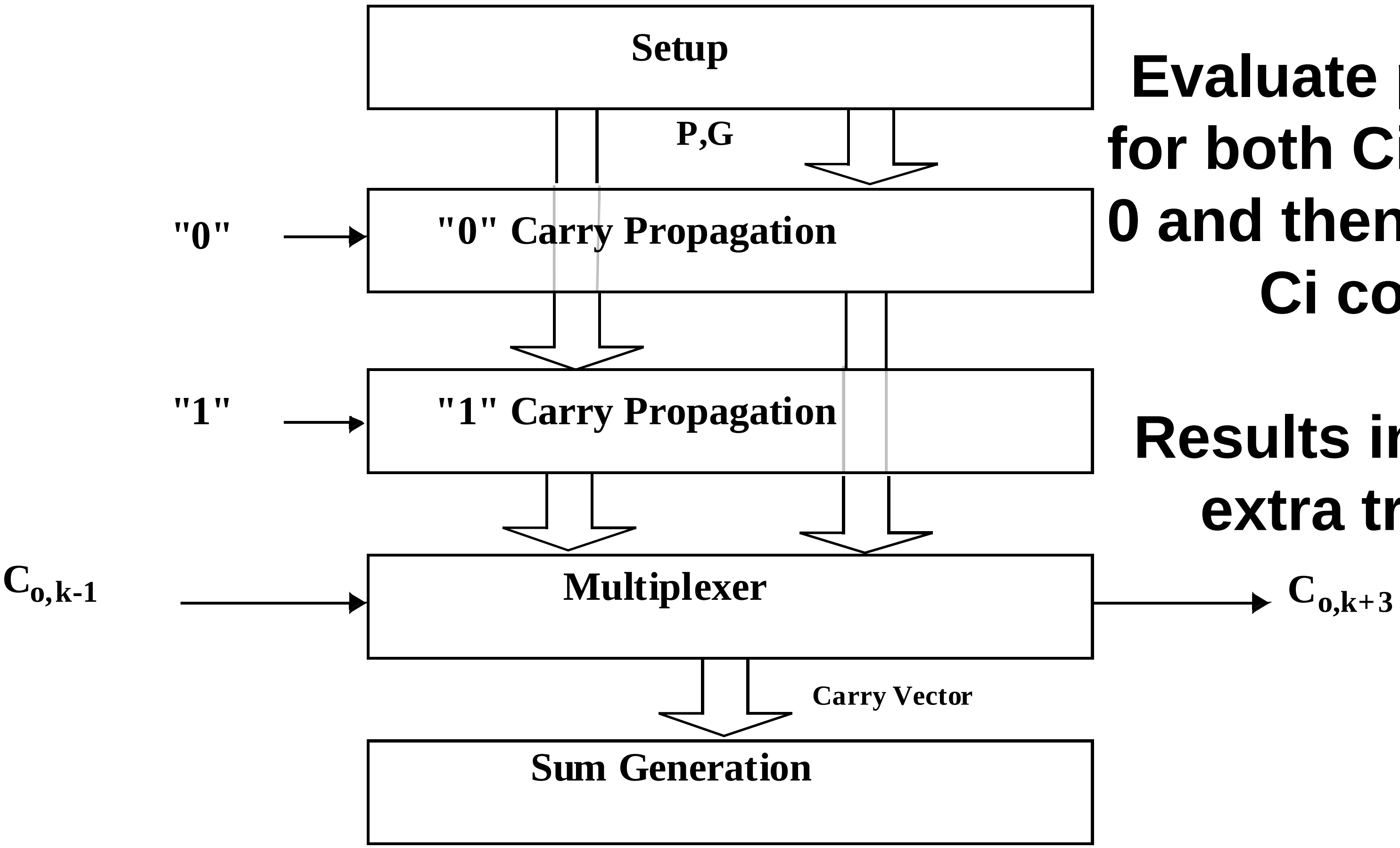
Note that this is done at the expense of a MUX in the carry delay path !!

Carry Ripple vs. Carry Bypass



Essentially greater than 4 bits is needed to overcome the overhead of the MUX

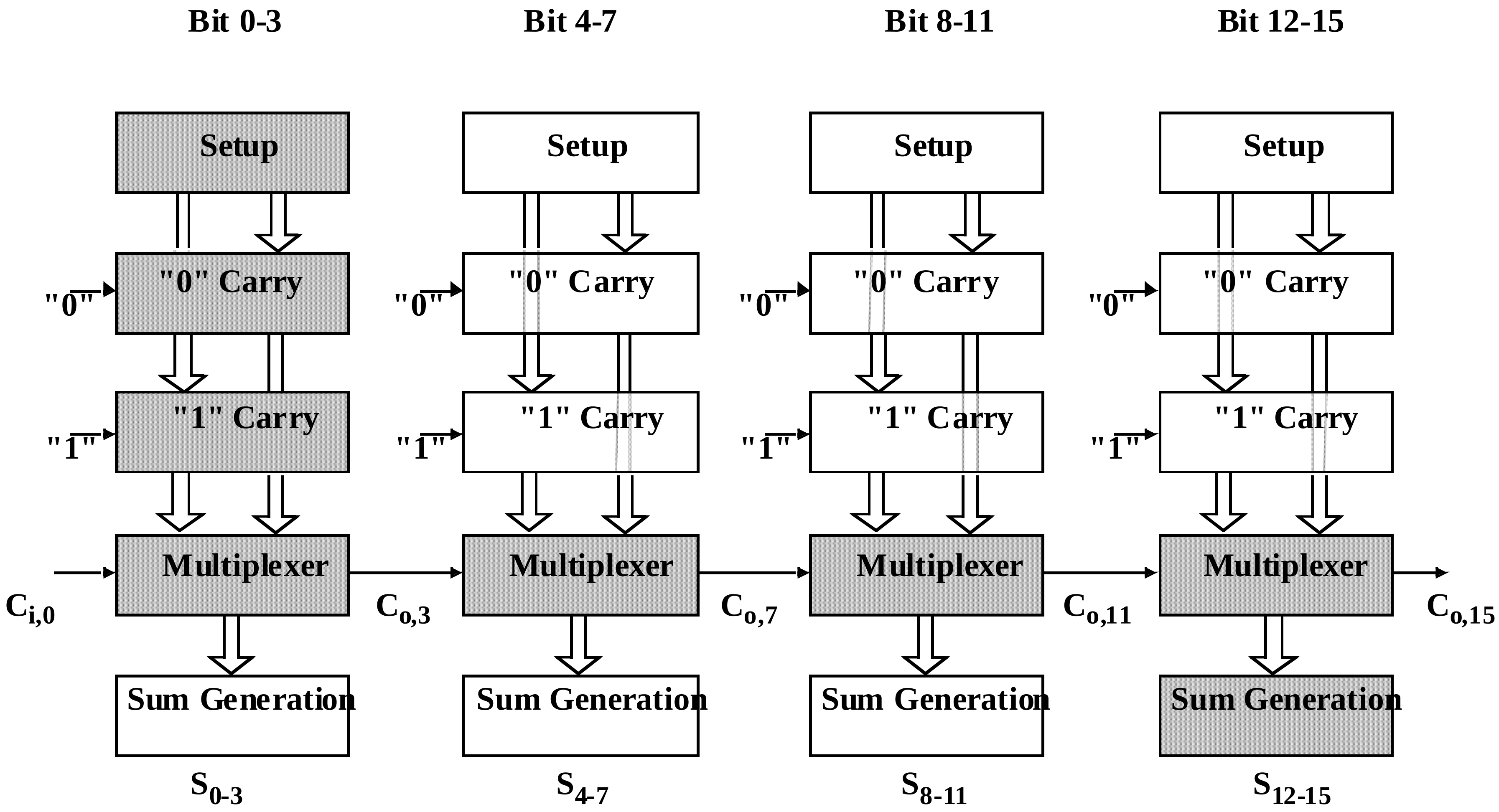
Carry-Select Adder



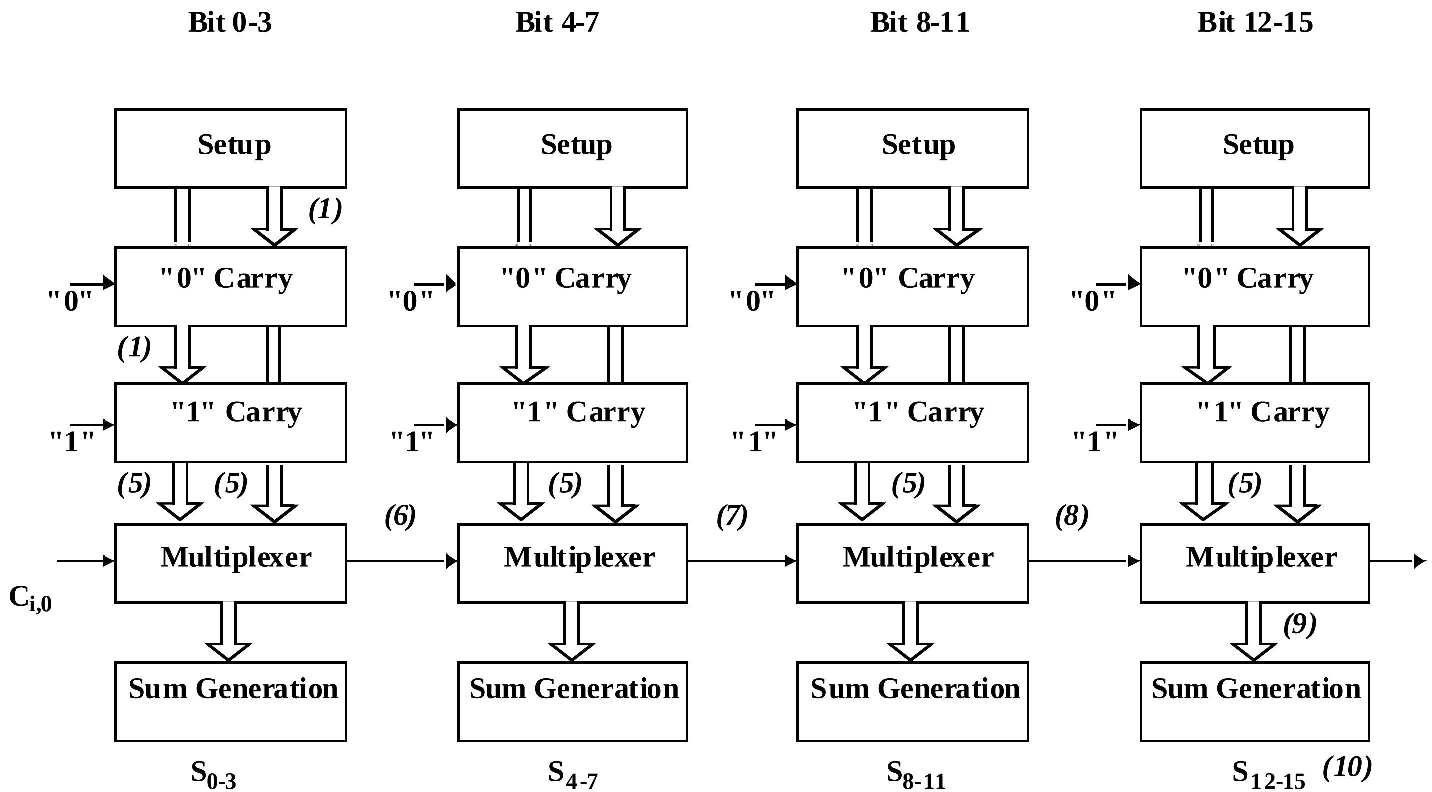
Evaluate possibilities for both Ci = 1 and Ci = 0 and then select when Ci comes in.

Results in about <u>30%</u> extra transistors

Carry Select Adder: Critical Path



Linear Carry Select

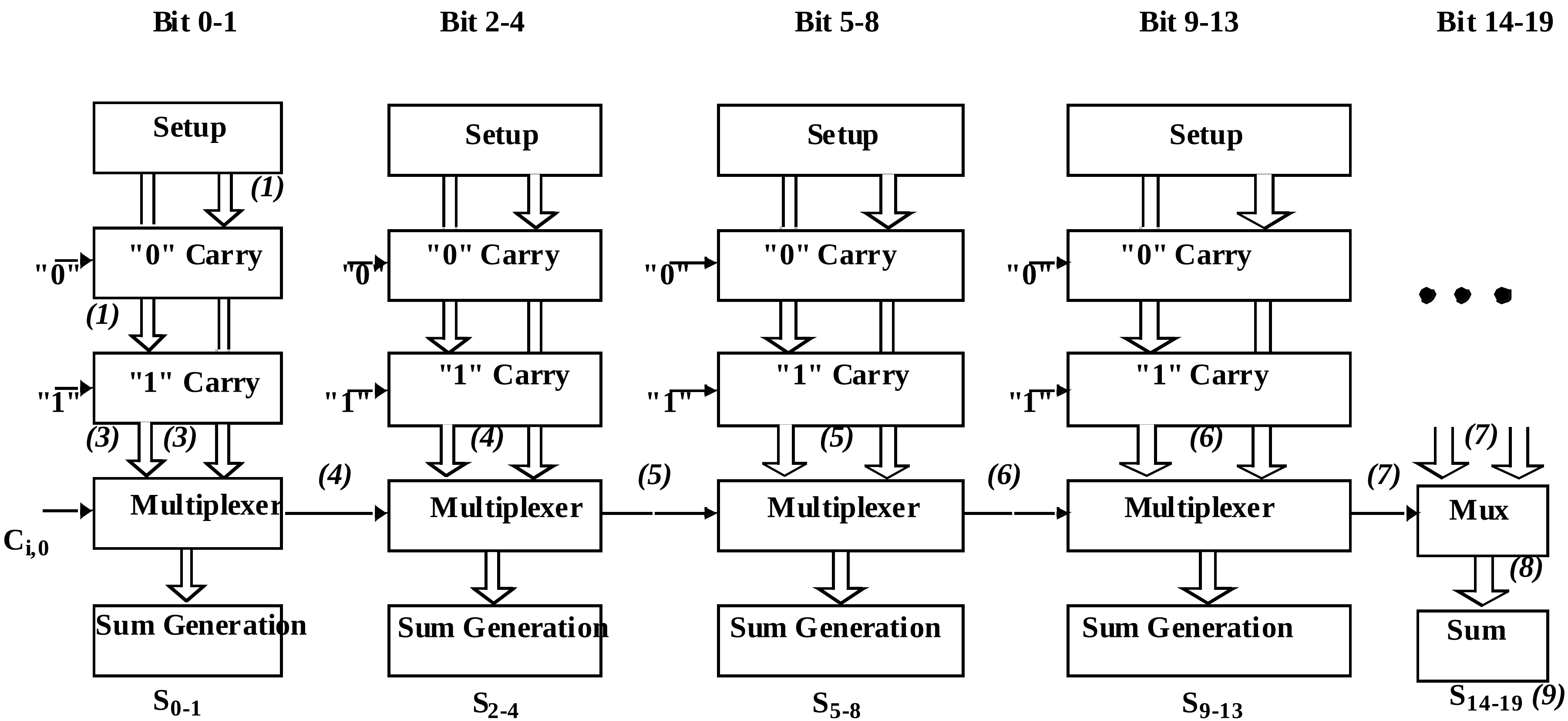


$$t_{add} = t_{setup} + \left(\frac{N}{M}\right) t_{carry} + M t_{mux} + t_{sum}$$

Carry-Select Adder Observations

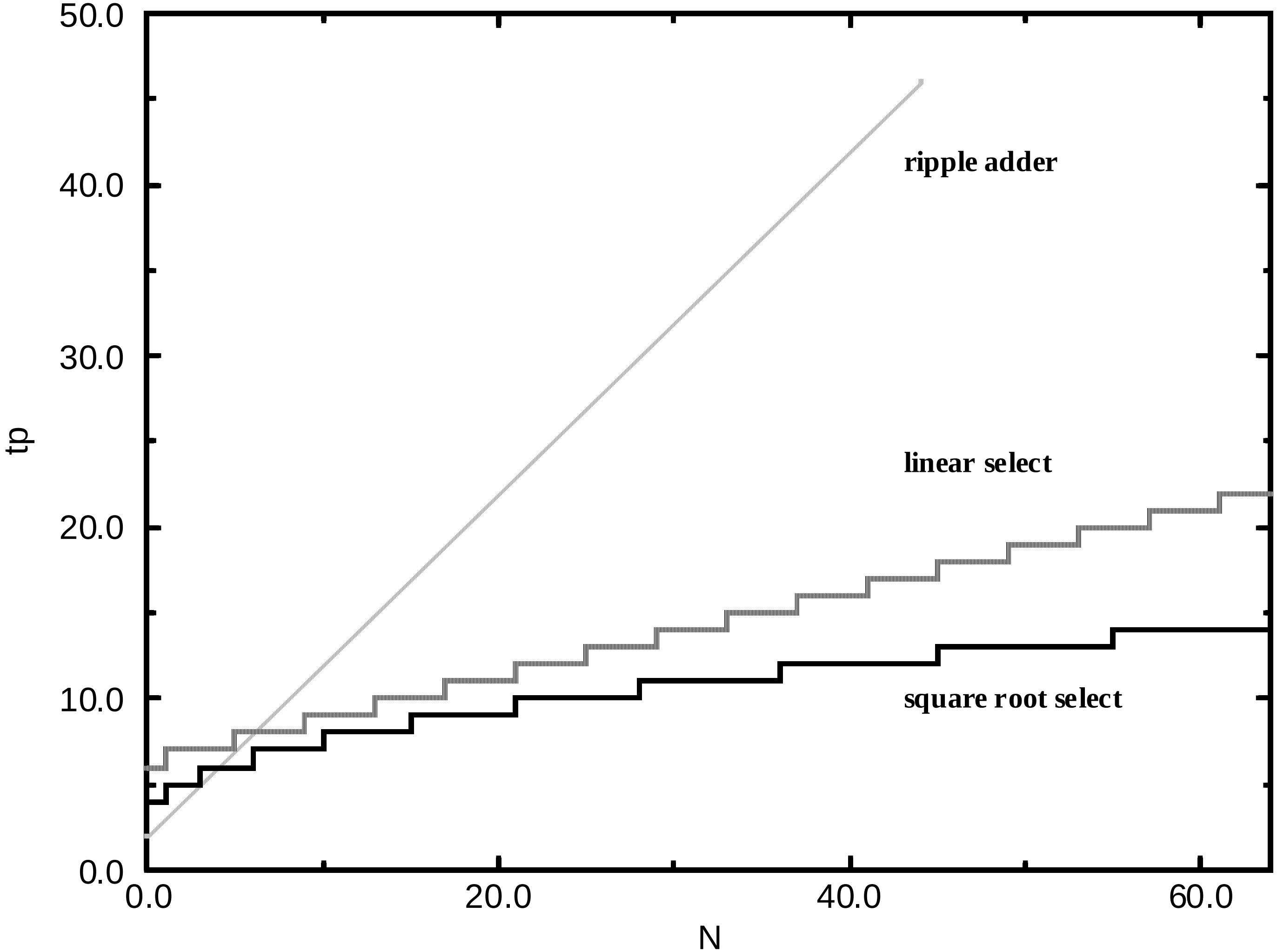
- The inputs to the final multiplexer are steady long before the Mux select (Ci) arrives
  - » Path is the same as is the number of bits
- Would be helpful to try and even out the delays so that the critical path is balanced between inputs and Mux select.
  - » Make logic simpler with the least significant bits by reducing the number of bits handled in the FA or half adder (HA). HA is FA without Ci (2 ins, 2 outs)
  - » Add bits progressively as you move to the MSB

Square Root Carry Select

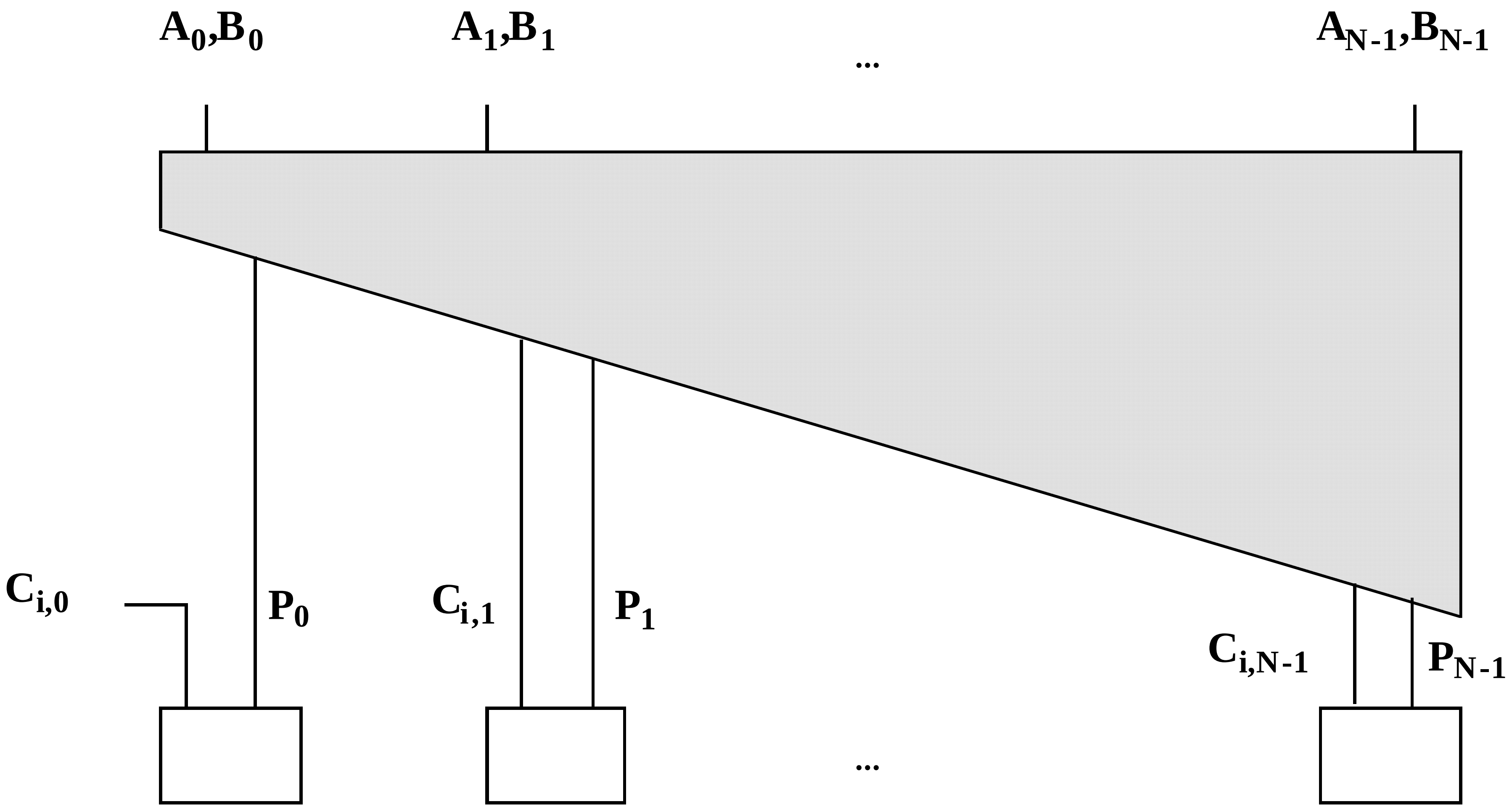


$$t_{add} = t_{setup} + P \cdot t_{carry} + (\sqrt{2N}) t_{mux} + t_{sum}$$

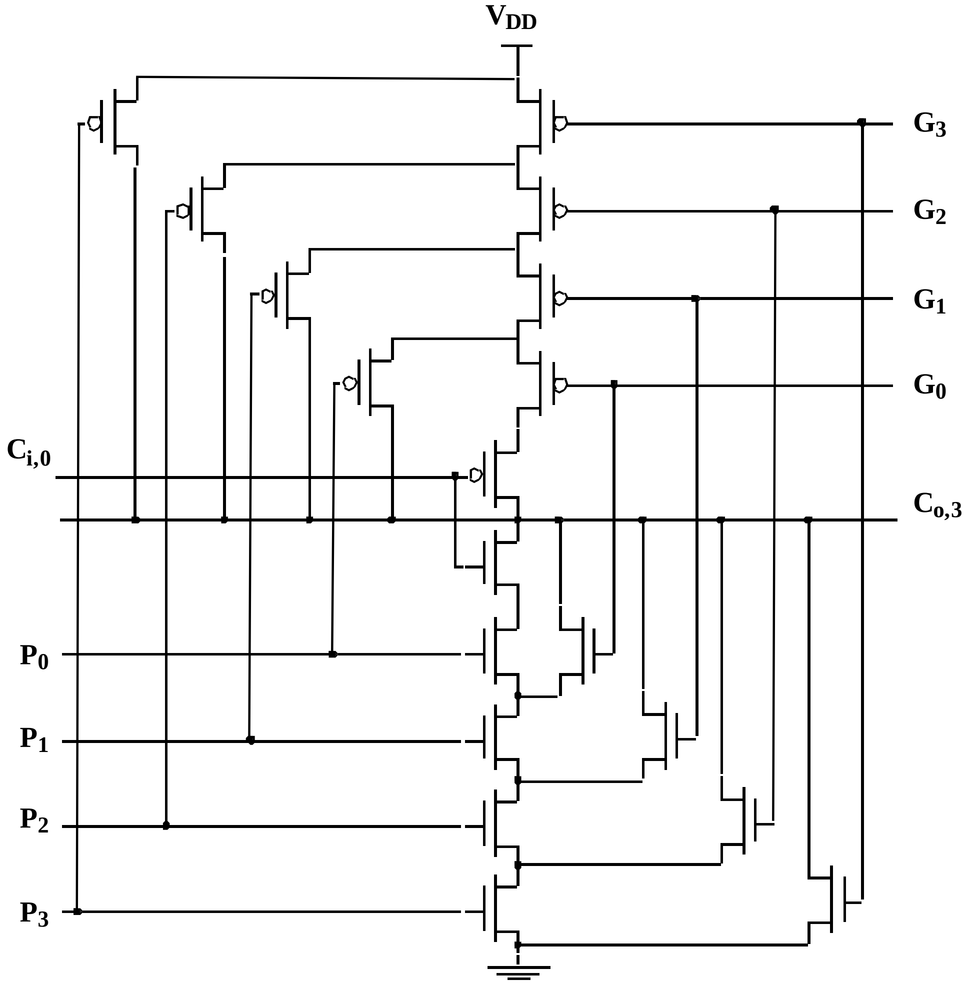
Adder Delays: Comparison



Carry Look Ahead: Basic Idea



Look-Ahead: Topology



- No more than N = 4 bits
  - Delay still increases linearly with number of bits
  - Capacitance, resistance too high for N > 4

Binary Multiplication

$$Z = X \times Y = \sum_{k=0}^{M+N-1} Z_k 2^k$$

$$= \left(\sum_{i=0}^{M-1} X_i 2^i\right)\left(\sum_{j=0}^{N-1} Y_j 2^j\right)$$

$$= \sum_{i=0}^{M-1} \left(\sum_{j=0}^{N-1} X_i Y_j 2^{i+j}\right)$$

with

$$X = \sum_{i=0}^{M-1} X_i 2^i$$

$$Y = \sum_{j=0}^{N-1} Y_j 2^j$$

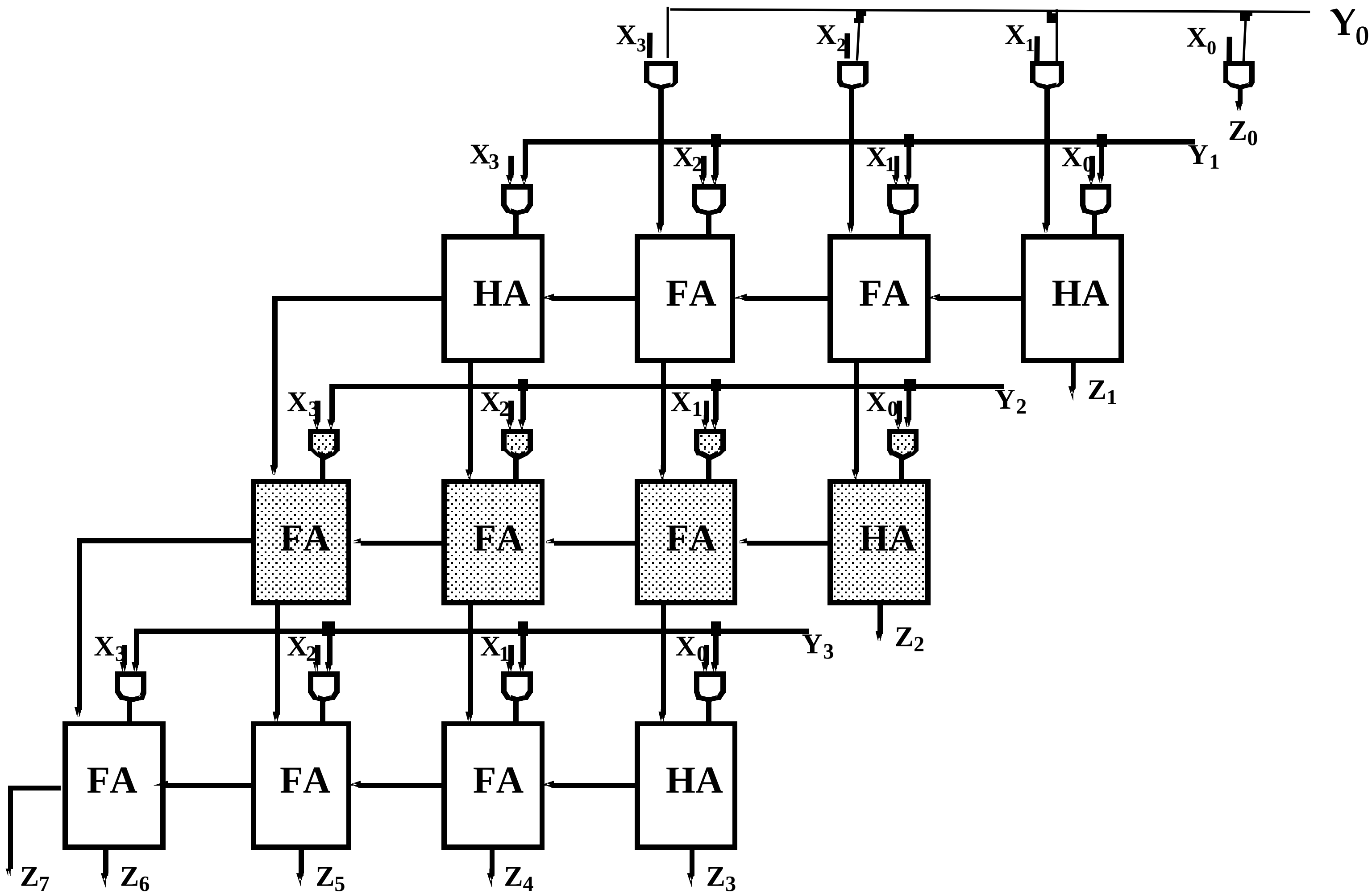
Binary Multiplication

```
            1 0 1 0 1 0
×               1 0 1 1
-----------------------
            1 0 1 0 1 0
          1 0 1 0 1 0
        0 0 0 0 0 0
+     1 0 1 0 1 0
-----------------------
      1 1 1 0 0 1 1 1 0
```

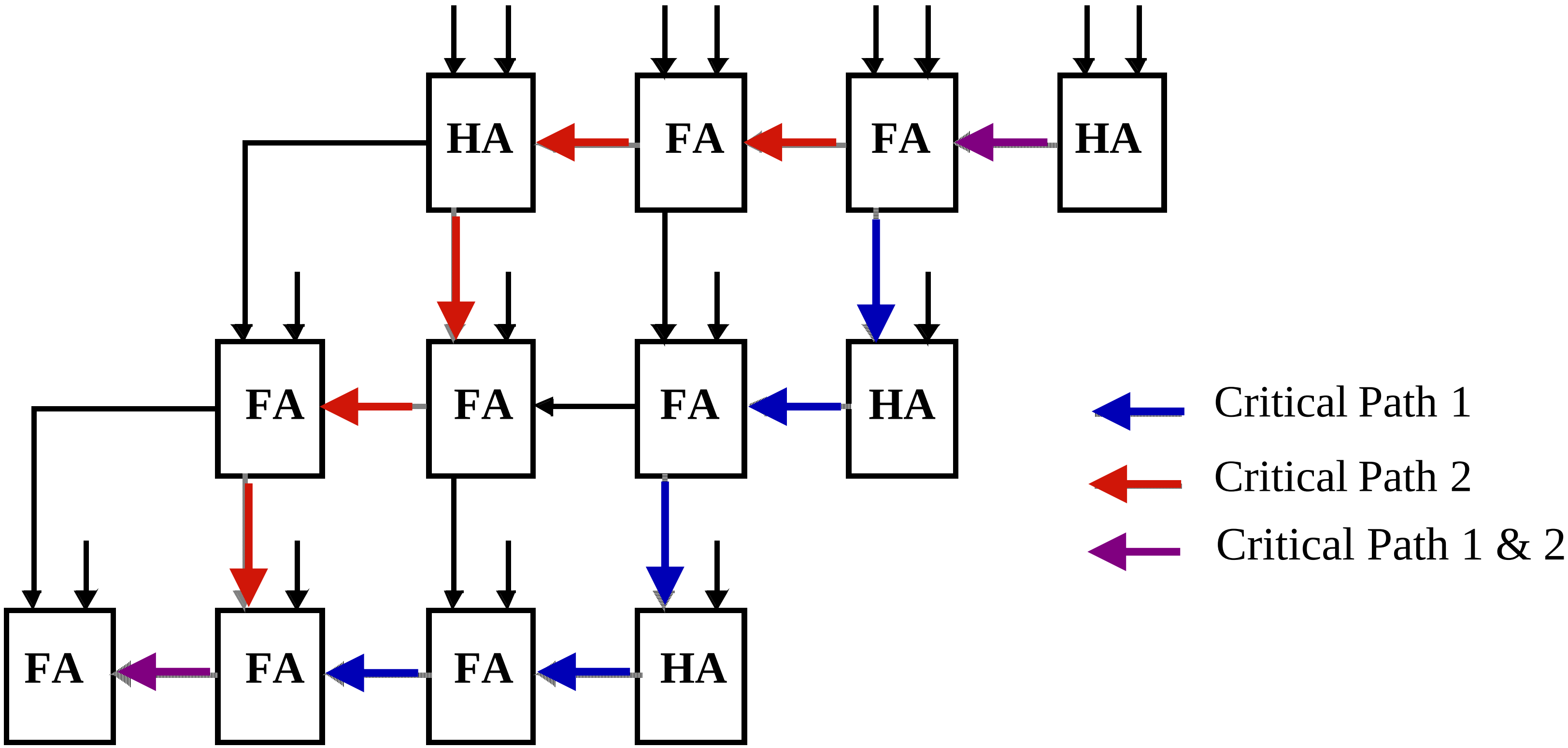
AND operation

Partial Products

The Array Multiplier

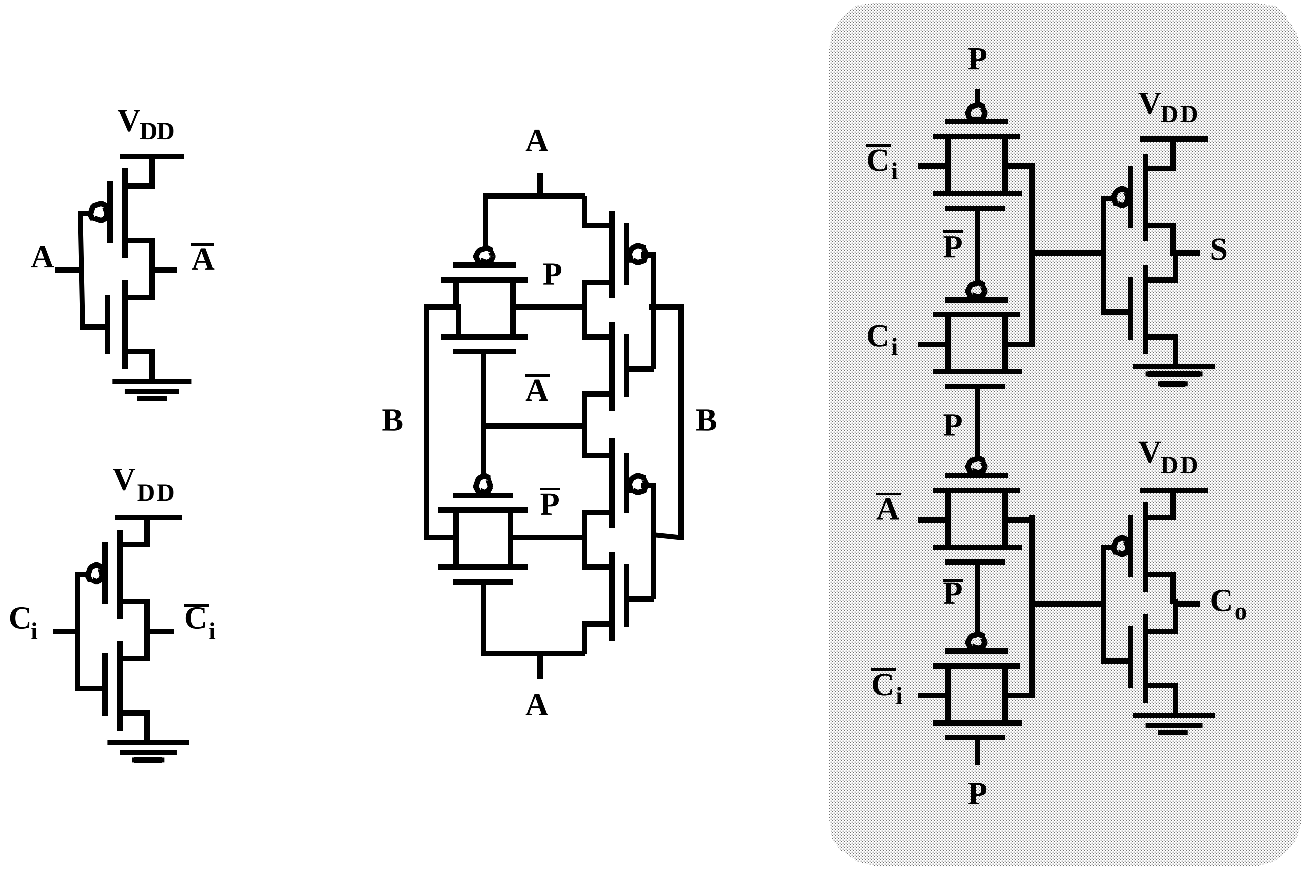


The MxN Array Multiplier: Critical Path



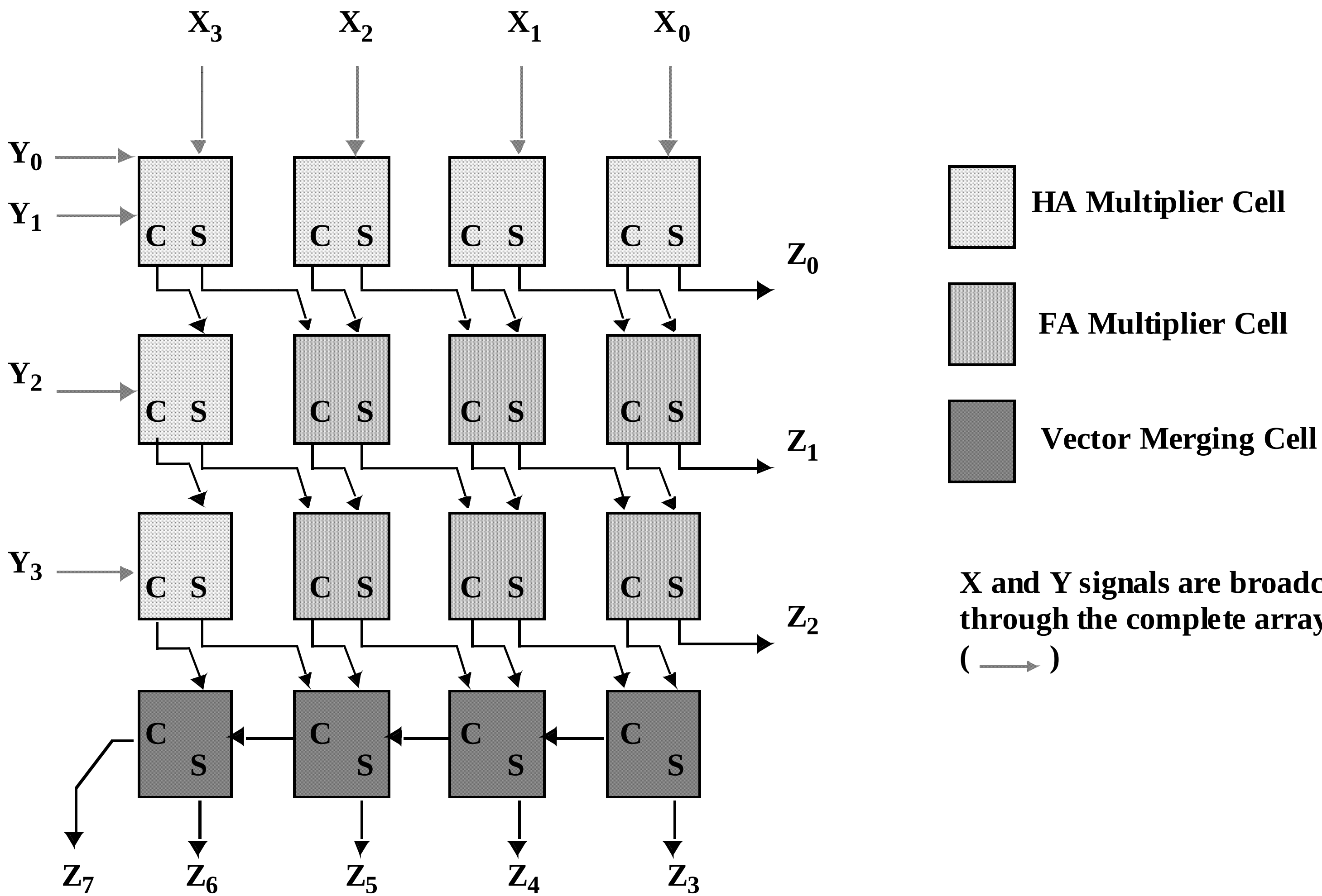
$$t_{mult} \approx [(M-1)+(N-2)]t_{carry} + (N-1)t_{sum} + t_{and}$$

Adder Cells in Array Multiplier



Identical Delays for Carry and Sum

Multiplier Floorplan

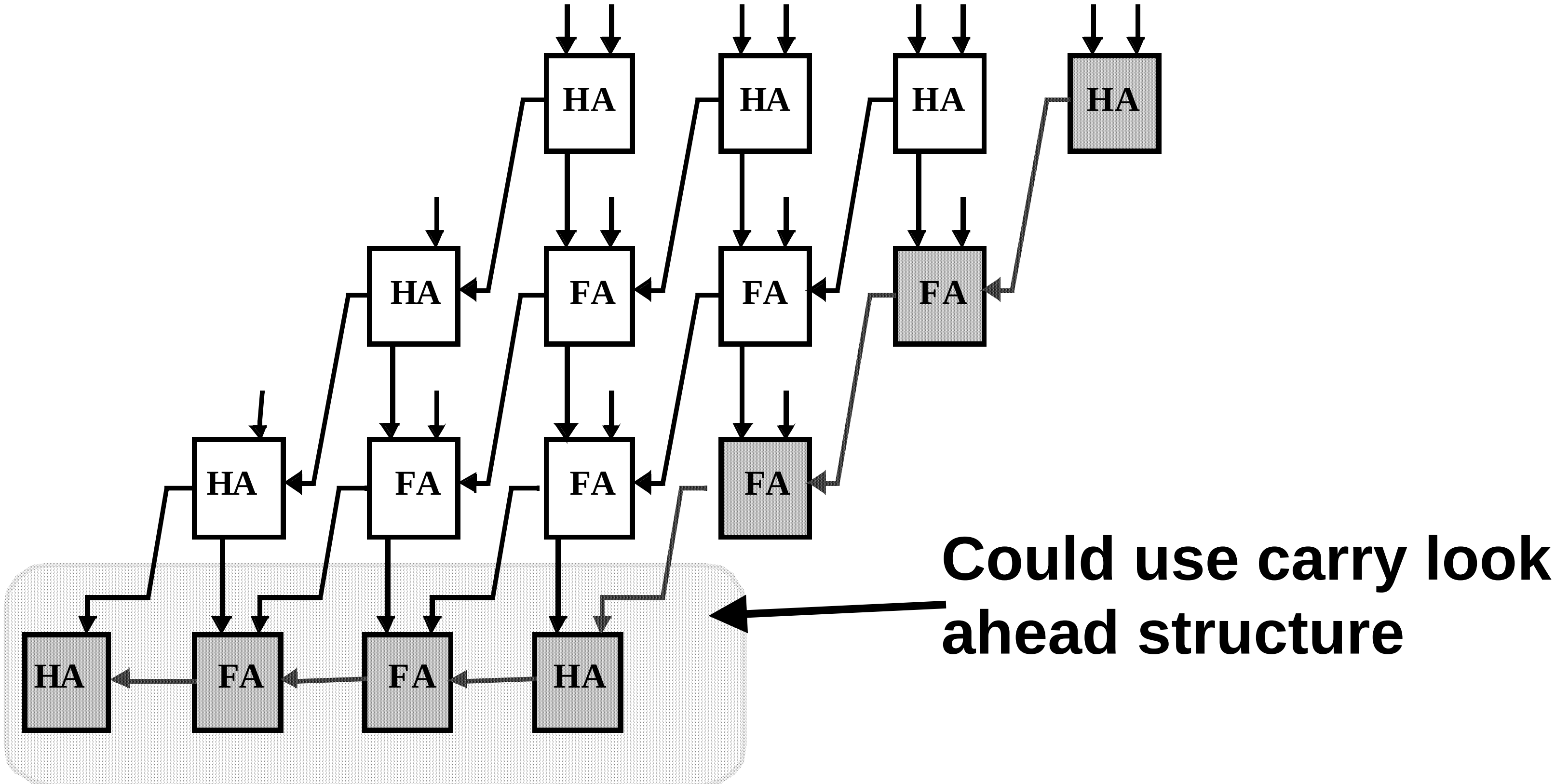


X and Y signals are broadcasted through the complete array. (→)

Array Multiplier Reflections

- Many equal critical paths
  - » Very hard to optimize by transistor sizing
- We could pass the carry bits diagonally down instead of across
  - » Output does not change
  - » Need to add an extra stage to accommodate this

Carry Save Multiplier

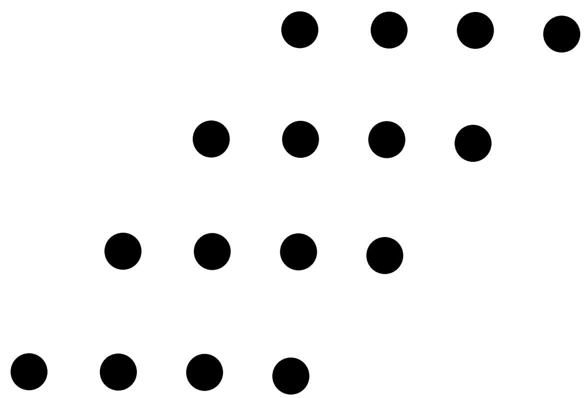


Vector Merging Adder

$$t_{mult} = (N-1)t_{carry} + t_{and} + t_{merge}$$

The Tree Multiplier

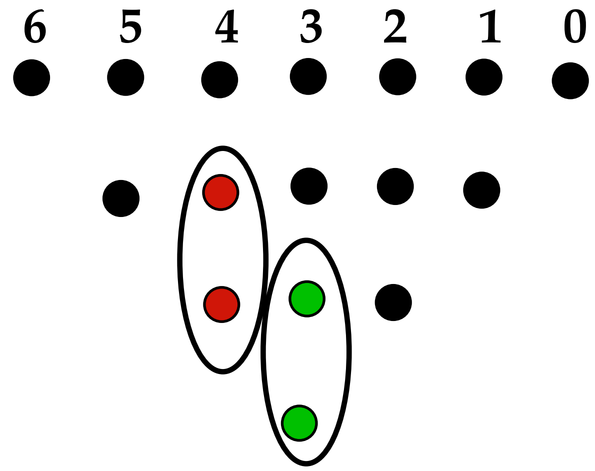
- Note that the partial products layout looks as follows:



- Note that we can rearrange and add the partial products differently
- Reduce number of adder circuits and logic depth
- FA compresses 3b to 2b, HA has 2b in and 2b out

Tree Multiplier

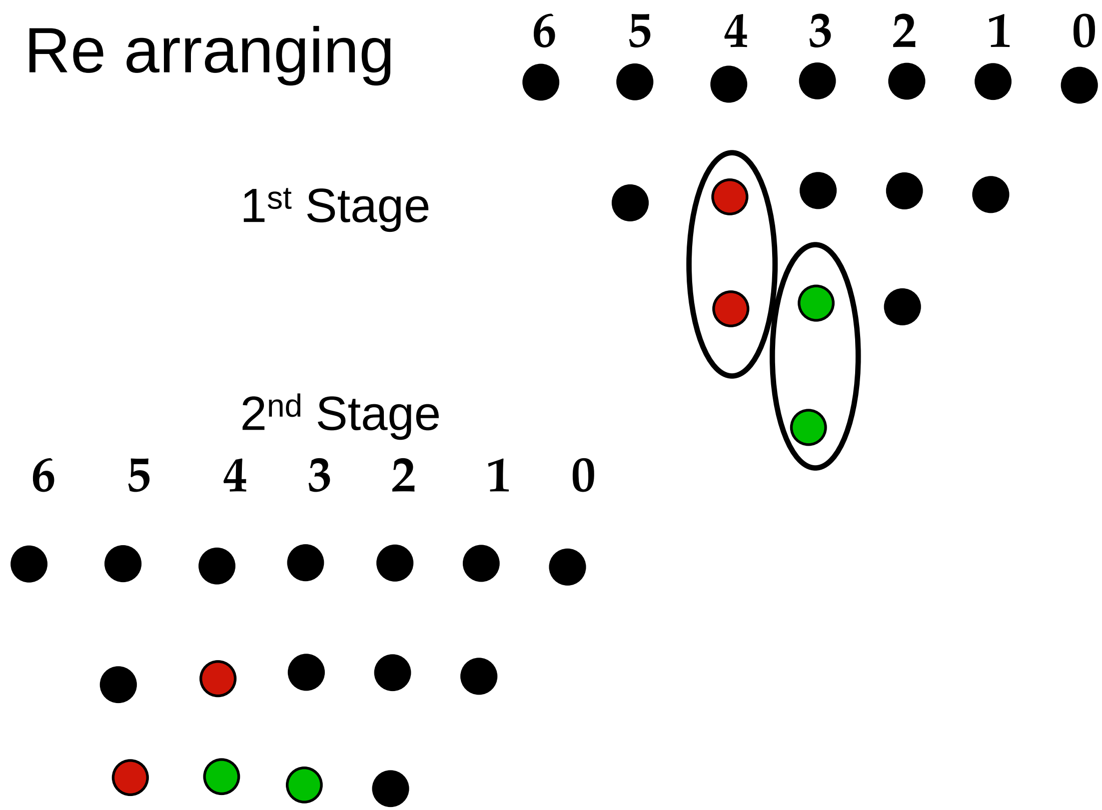
- Re arranging



1st Stage
Half Adders

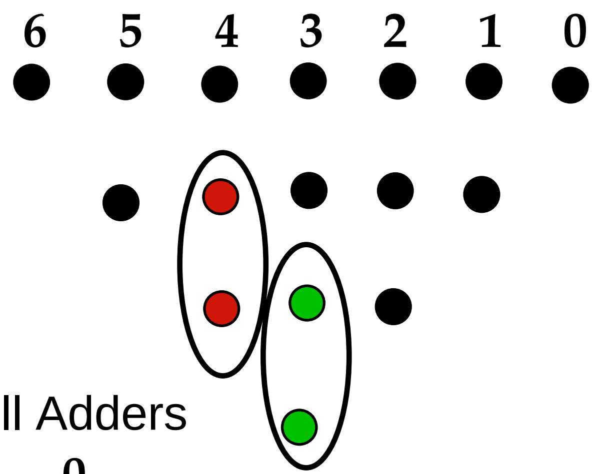
Tree Multiplier

- Re arranging



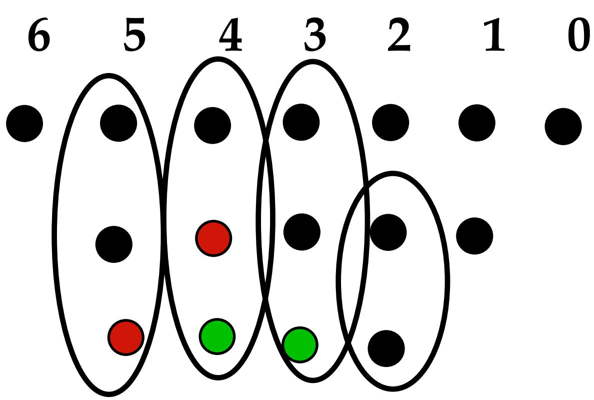
Tree Multiplier

- Re arranging



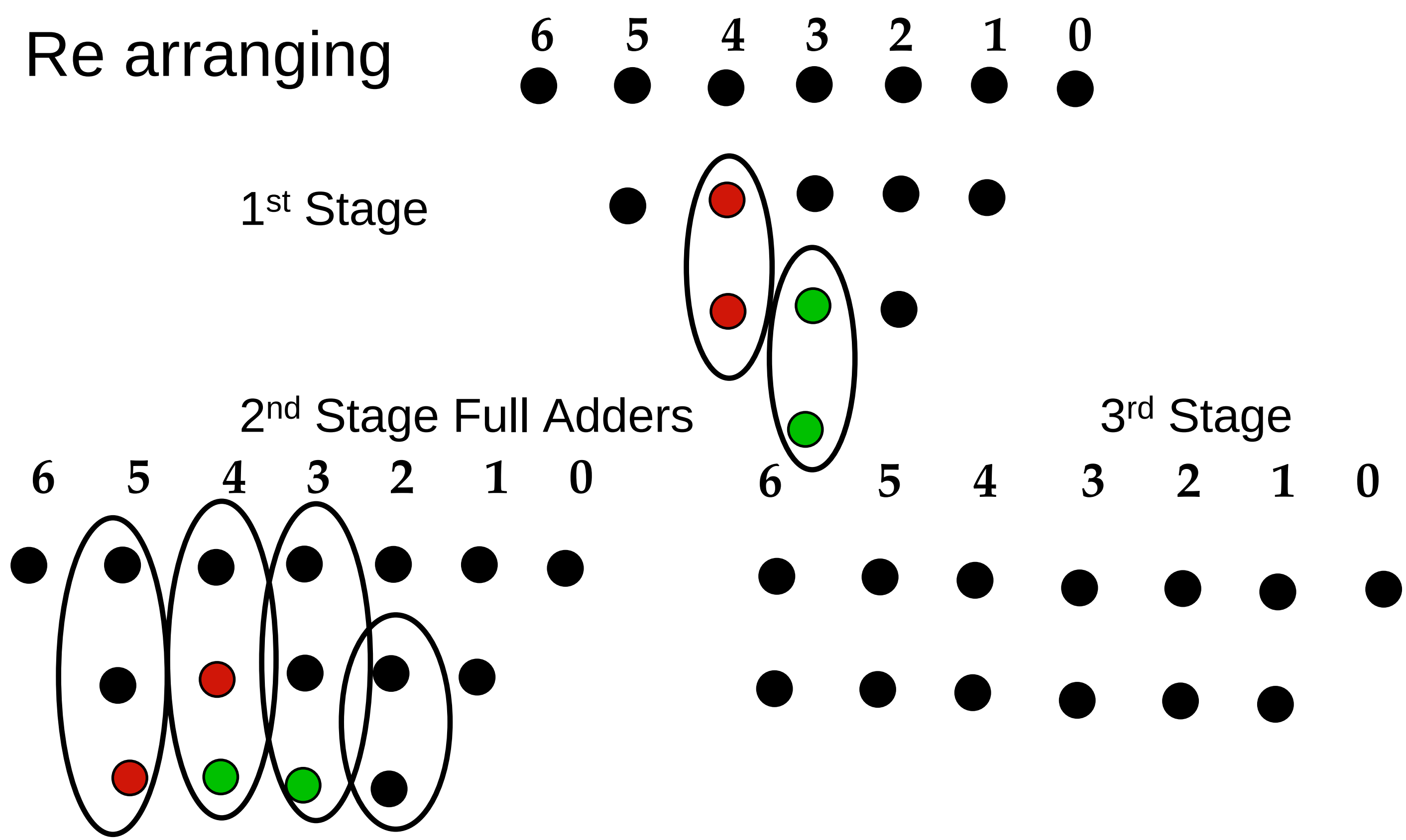
1st Stage

2nd Stage Full Adders



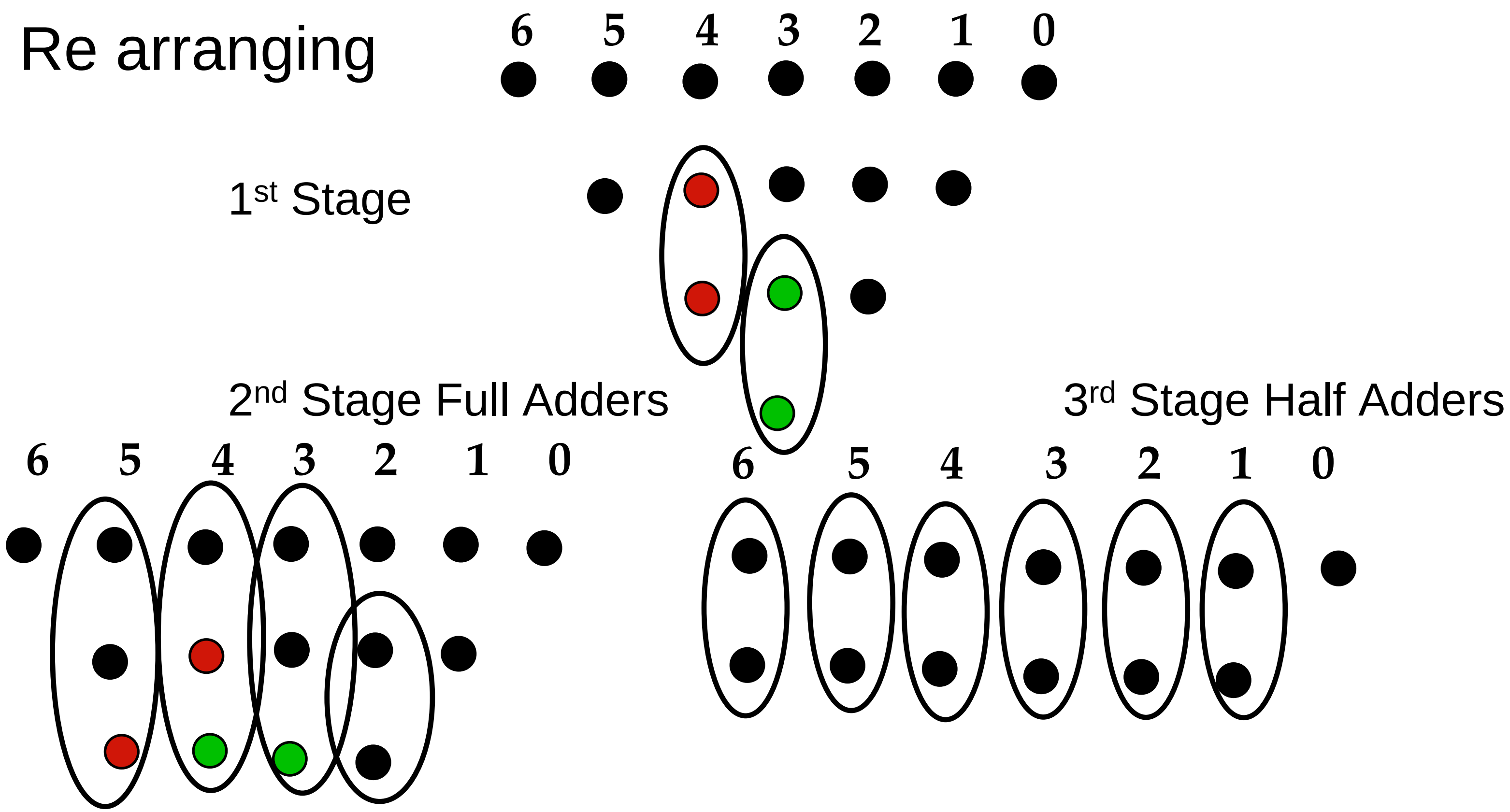
Tree Multiplier

- Re arranging

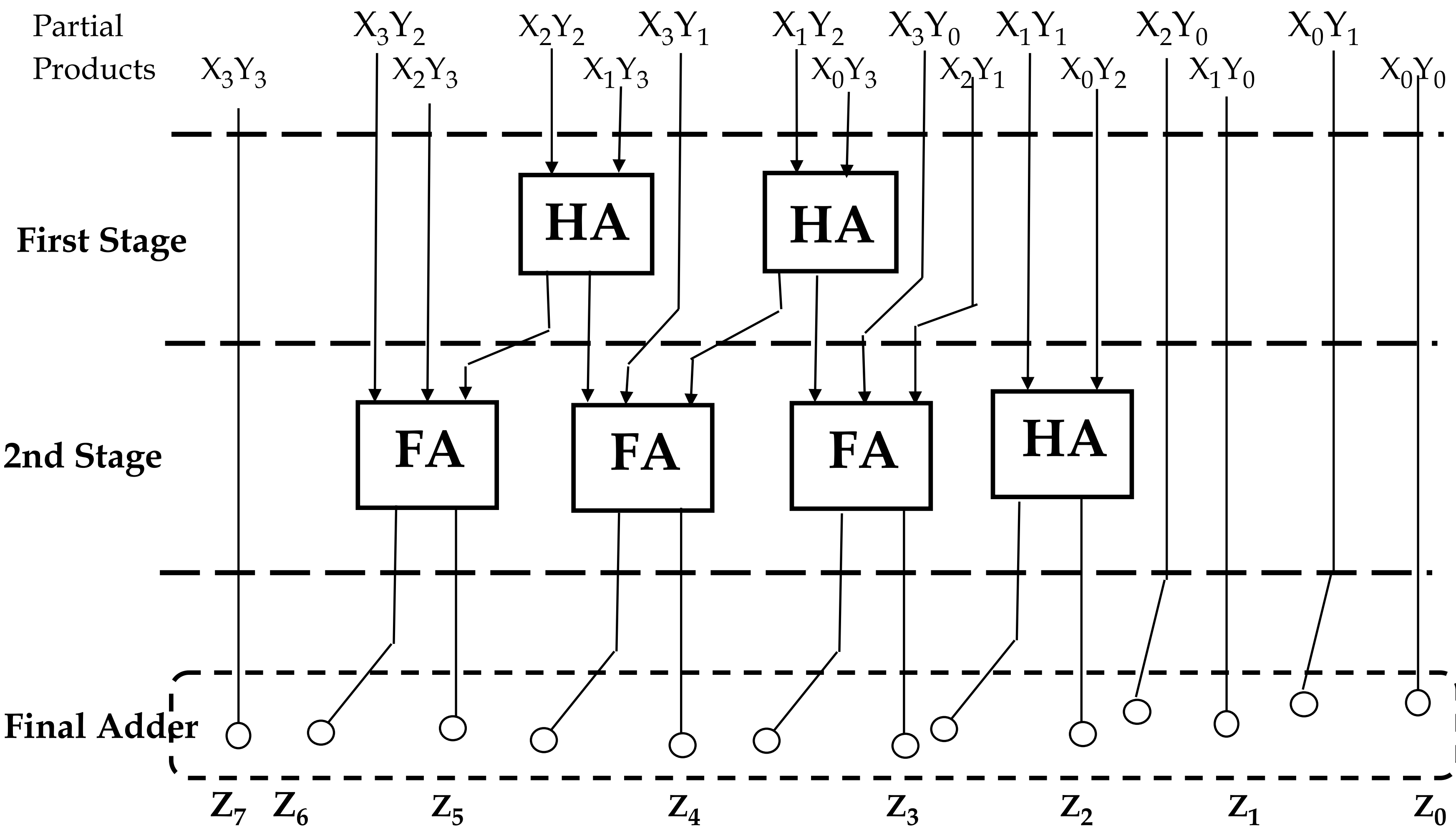


Tree Multiplier

- Re arranging



Wallace-Tree Multiplier



Multipliers: Summary

- Optimization goals different than Adder
  - » Identify critical path
  - » More system level optimization then individual cell optimization

Tree Multiplier

- Re arranging