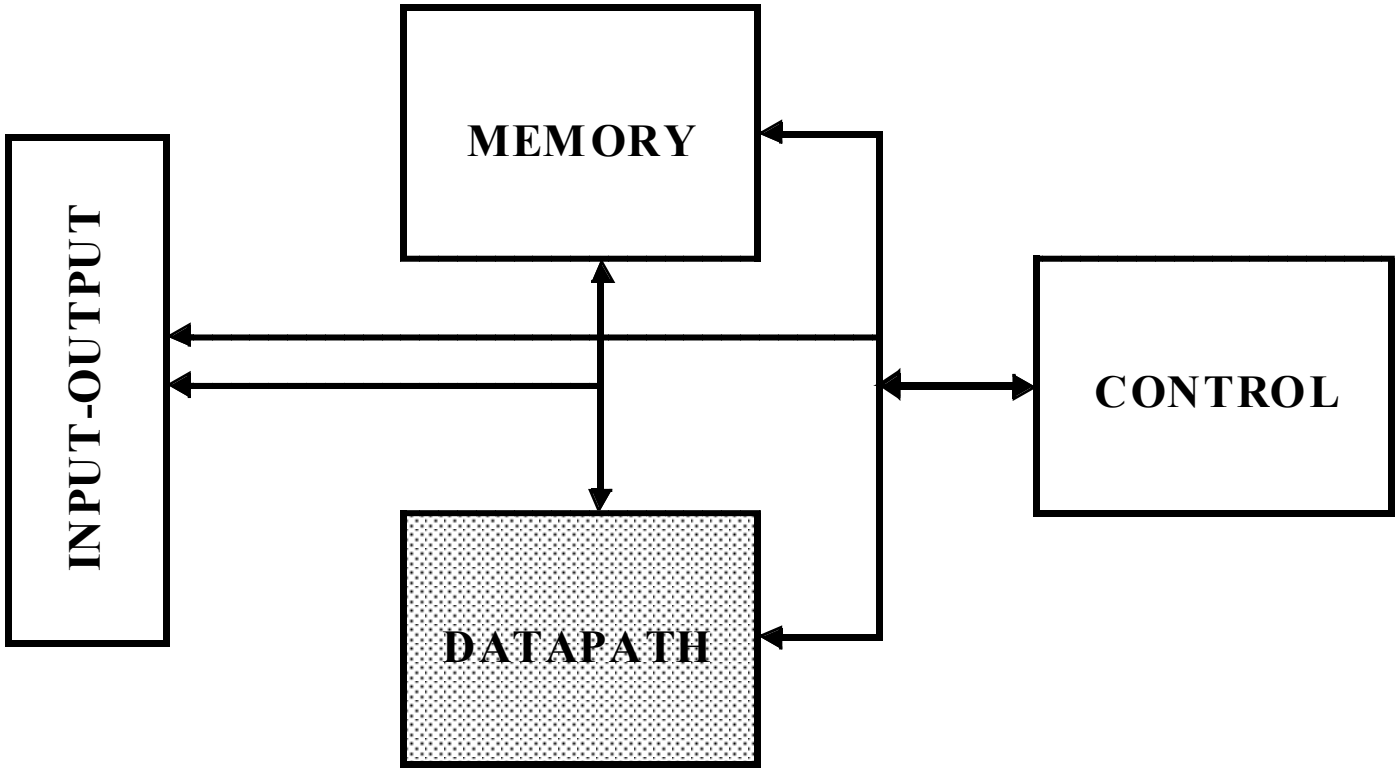
Digital Integrated Circuits
A Design Perspective

Jan M. Rabaey
Anantha Chandrakasan
Borivoje Nikolic

Arithmetic Circuits

A Generic Digital Processor



Building Blocks for Digital Architectures

Arithmetic unit

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

Memory

- RAM, ROM, Buffers, Shift registers

Control

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

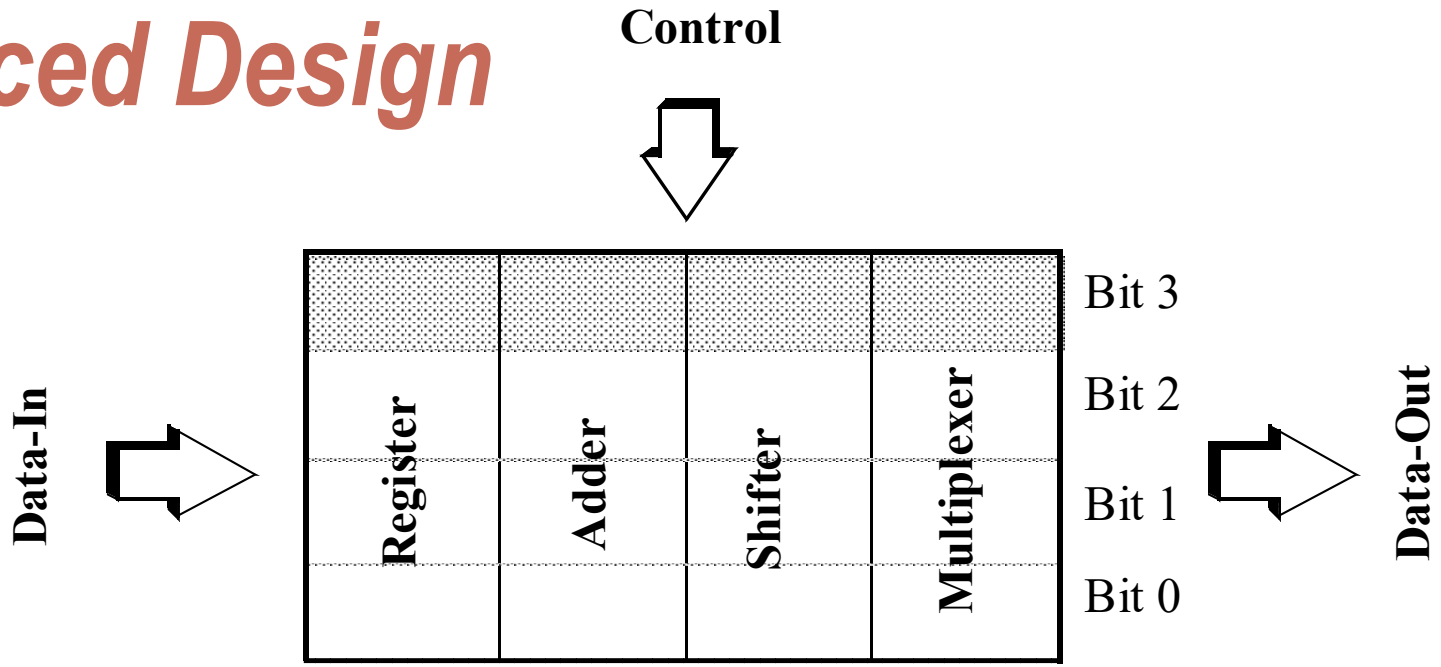
Interconnect

- Switches
- Arbiters
- Bus

Arithmetic building blocks

- Speed and power of arithmetic components often dominates the overall system performance
- For each module, multiple topologies and ways of design exists, with each of them has its own advantages
- A global picture is of crucial importance. A designer focus their attention on gates or transistors that have the largest impact on their goal function. Non-critical components can be developed routinely.
- Typically two optimization process: logic optimization (re-arrange Boolean equations so that a faster or small circuit could be obtained) and circuit optimization (manipulate circuit topology and transistor sizes to optimize speed)

Bit-Sliced Design

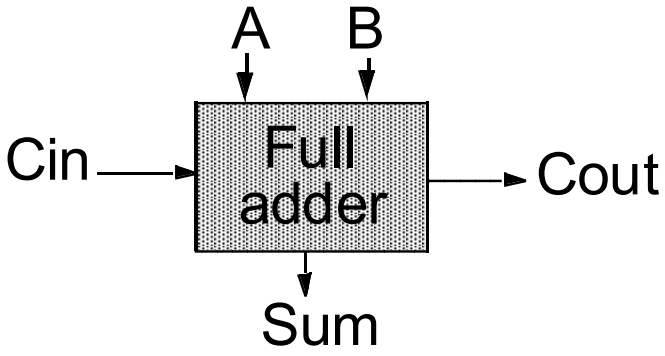


Tile identical processing elements

Since the same operation has to be performed on each bit of a data word, the data path can consist of the number of bit slices (equal to the word length), each operating on a single bit – hence the term ***bit-sliced***

Adders

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

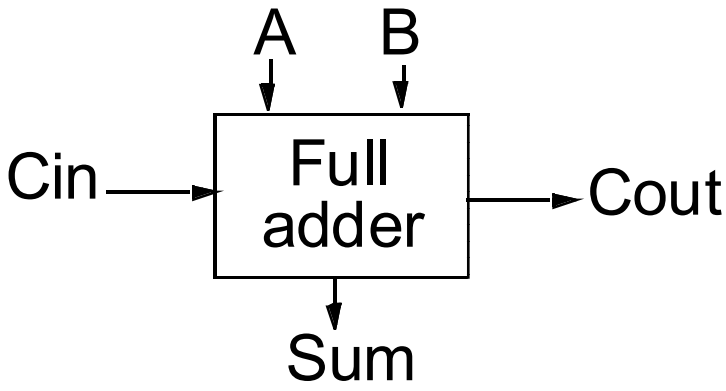
Generate (G) = AB

Propagate (P) = $A \oplus B$

Delete (D) = $\overline{A}\,\overline{B}$

G,D, ensures a carry bit will be generated or deleted at Co independent of Ci, While P guarantees that Ci will propagate to Co.

The Binary Adder

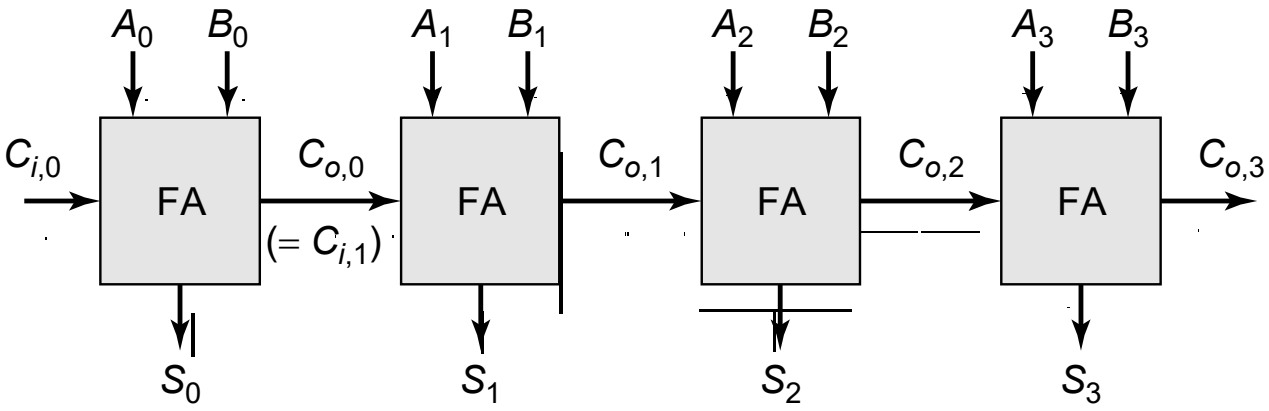


$$S = A \oplus B \oplus C_i$$

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

$$C_o = AB + BC_i + AC_i$$

The Ripple-Carry Adder



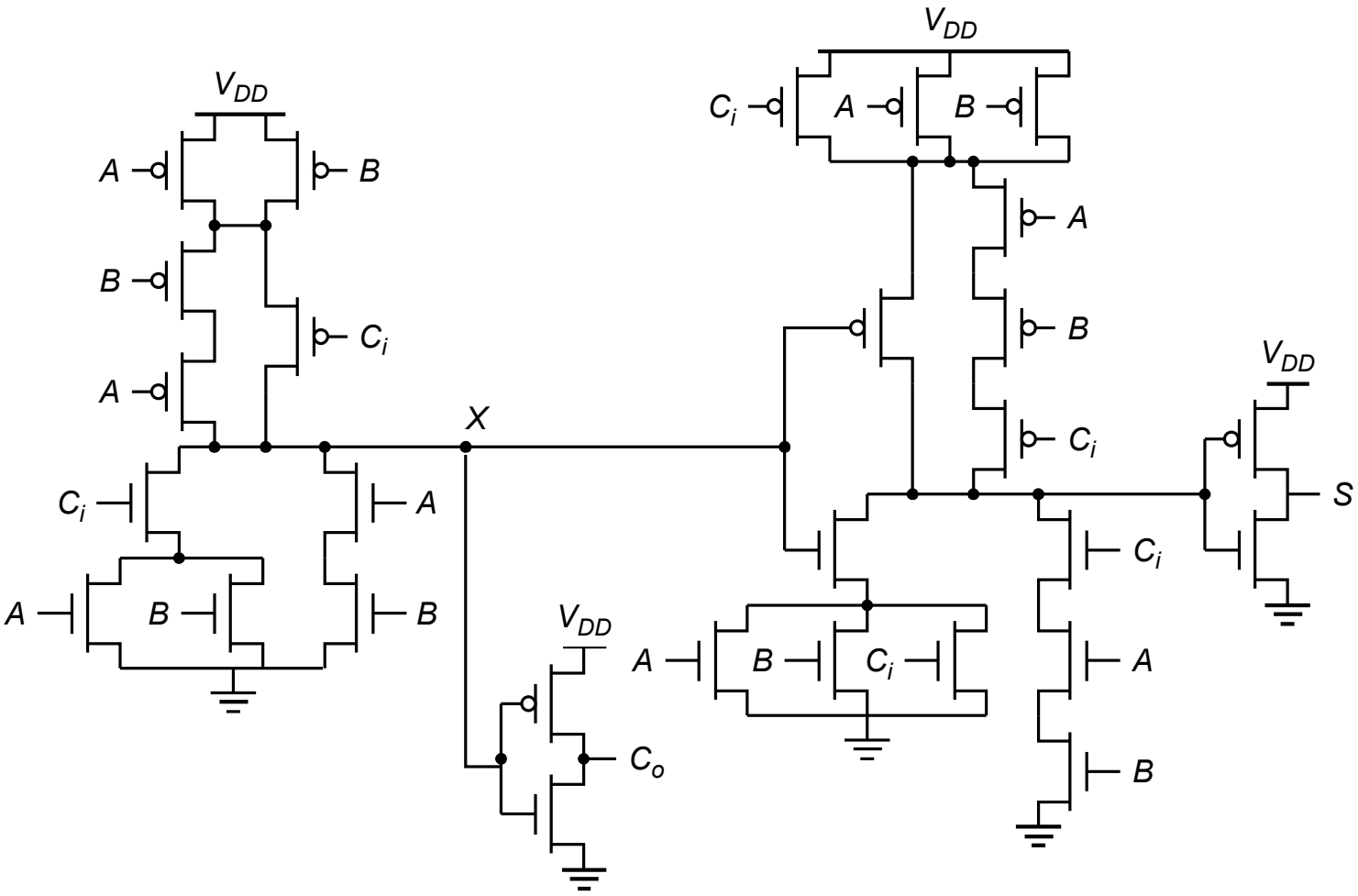
Worst case delay linear with the number of bits

$$t_d = \mathrm{O}(N)$$

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

Goal: Make the fastest possible carry path circuit

Complimentary Static CMOS Full Adder



28 Transistors

Complimentary Static CMOS Full Adder

- Large PMOS stacks are present in both carry and sum generation circuits
- Intrinsic load capacitance of C_o signal is large and consists of eight capacitance components
- There is one more inverter delay for carry and sum (worse when the load capacitance is large)
- Note that critical signal C_i closer to the output node

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

Define 3 new variable which ONLY depend on A, B

Generate (G) = AB

Propagate (P) = A ⊕ B

Delete (D) = $\overline{A}\,\overline{B}$

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

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

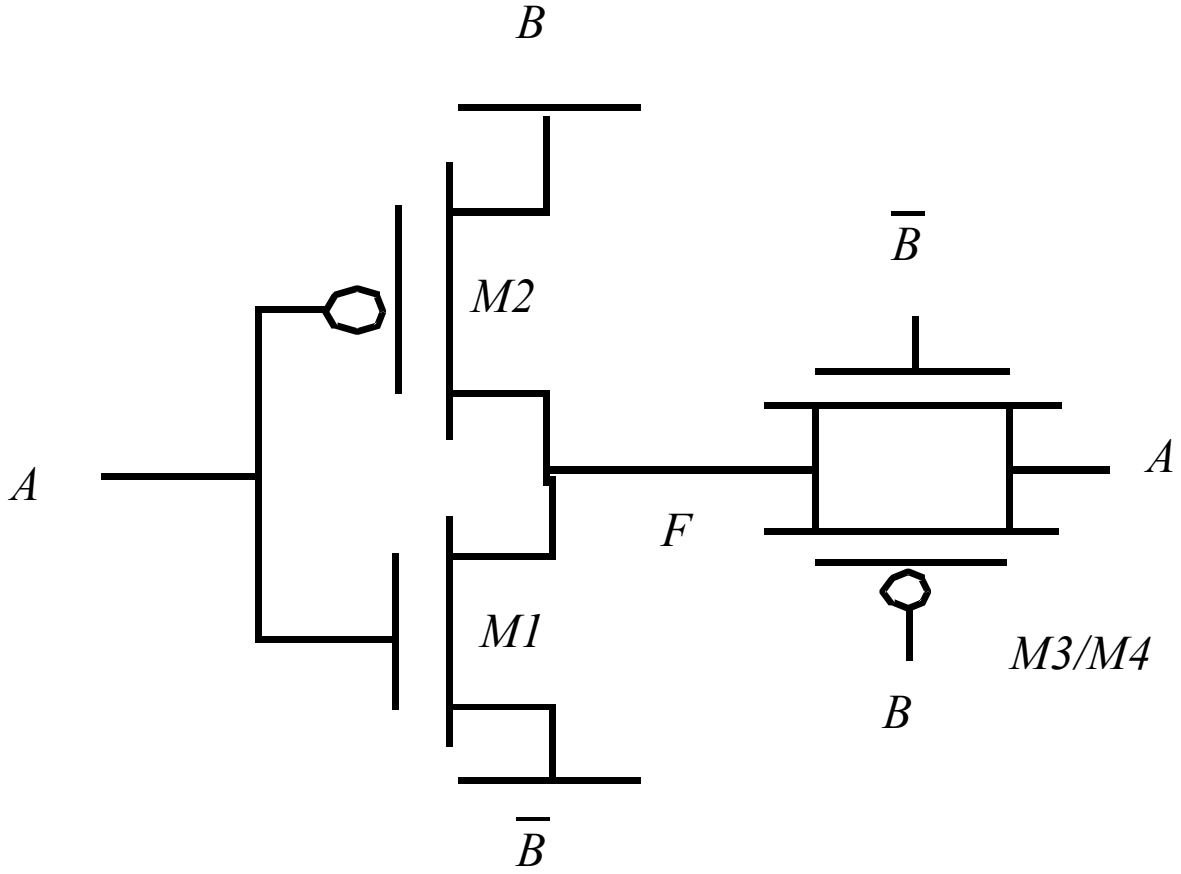
Can also derive expressions for *S* and C_o based on *D* and *P*

Note that we will be sometimes using an alternate definition for

Propagate (P) = A + B

Transmission Gate XOR

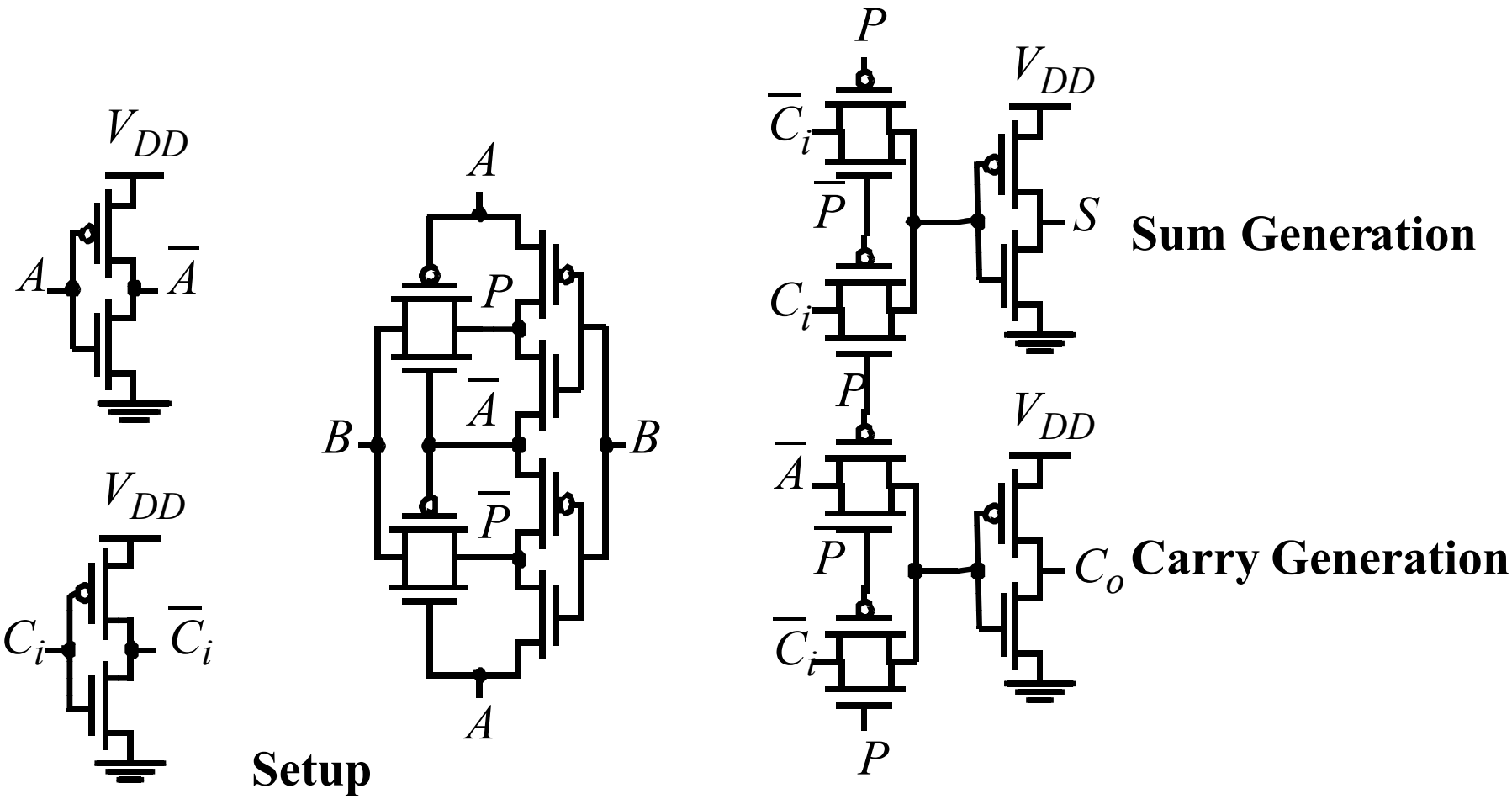
$F = (\overline{A} \bullet B + A \bullet \overline{B})$, 12 transistors for complementary implementation



When B=1, M1/M2 inverter, M3/M4 off, so F=$\overline{A}$B

When B=0, M1/M2 off, M3/M4 transmission gate, so F=A$\overline{B}$

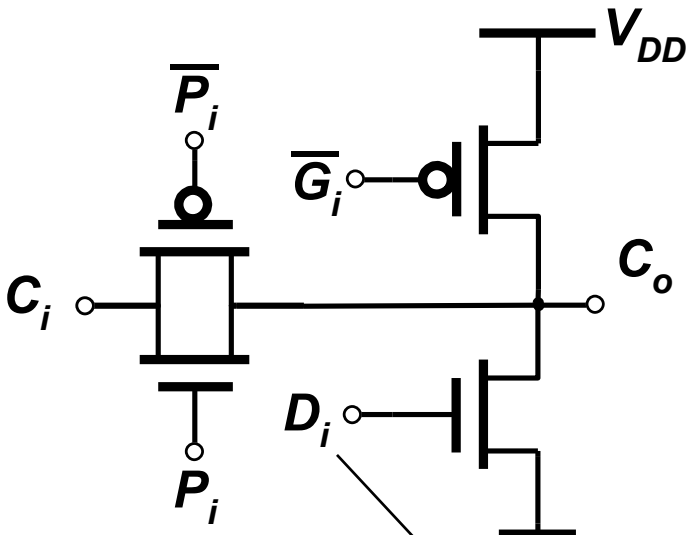
Transmission Gate Full Adder

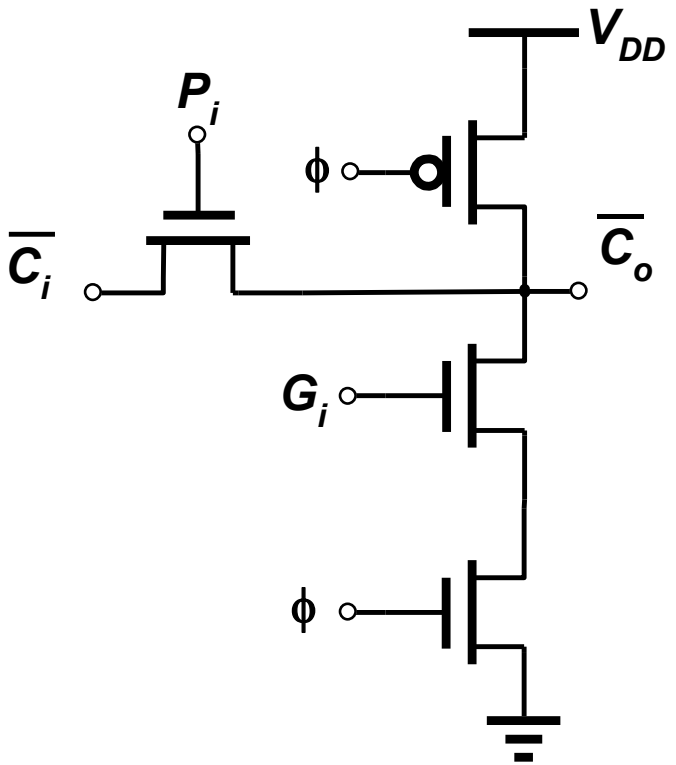


Propagate (P) = A ⊕ B

$$C_o(G,P) = G + PC_i$$
$$S(G,P) = P \oplus C_i$$

Manchester Carry Chain





Generate (G) = AB

Propagate (P) = $A \oplus B$

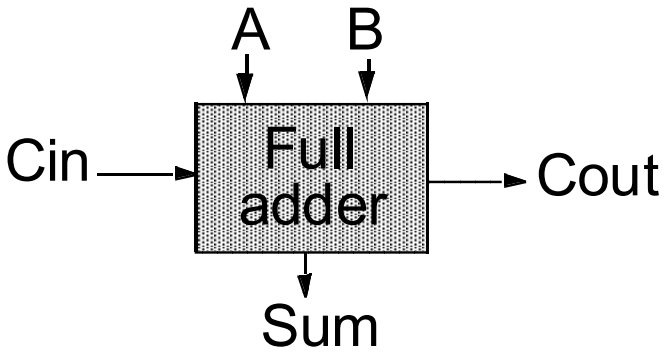
Delete = $\overline{A}\,\overline{B}$

Prevent floating C_o

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

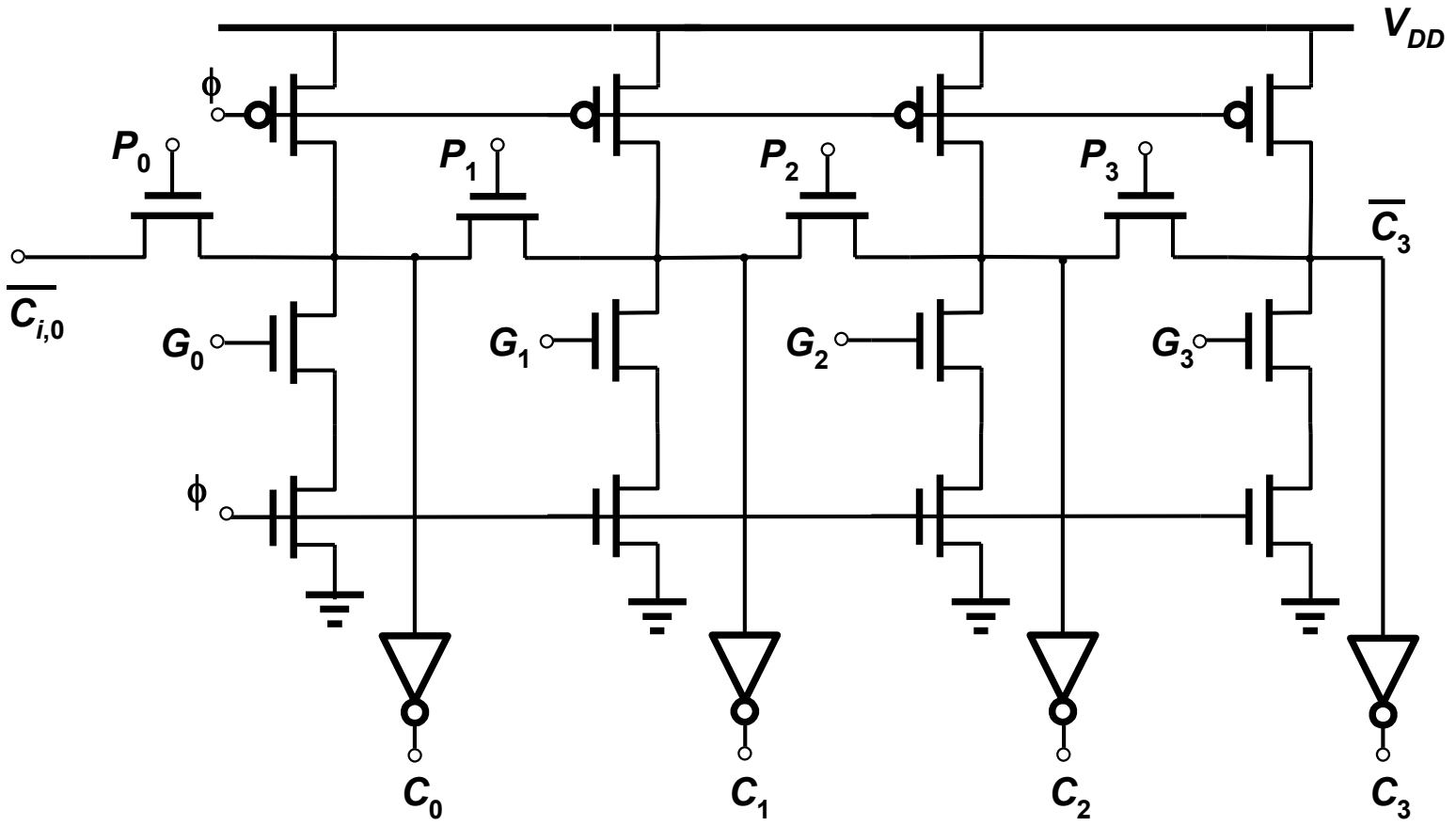
$$S(G,P) = P \oplus C_i$$

Full-Adder



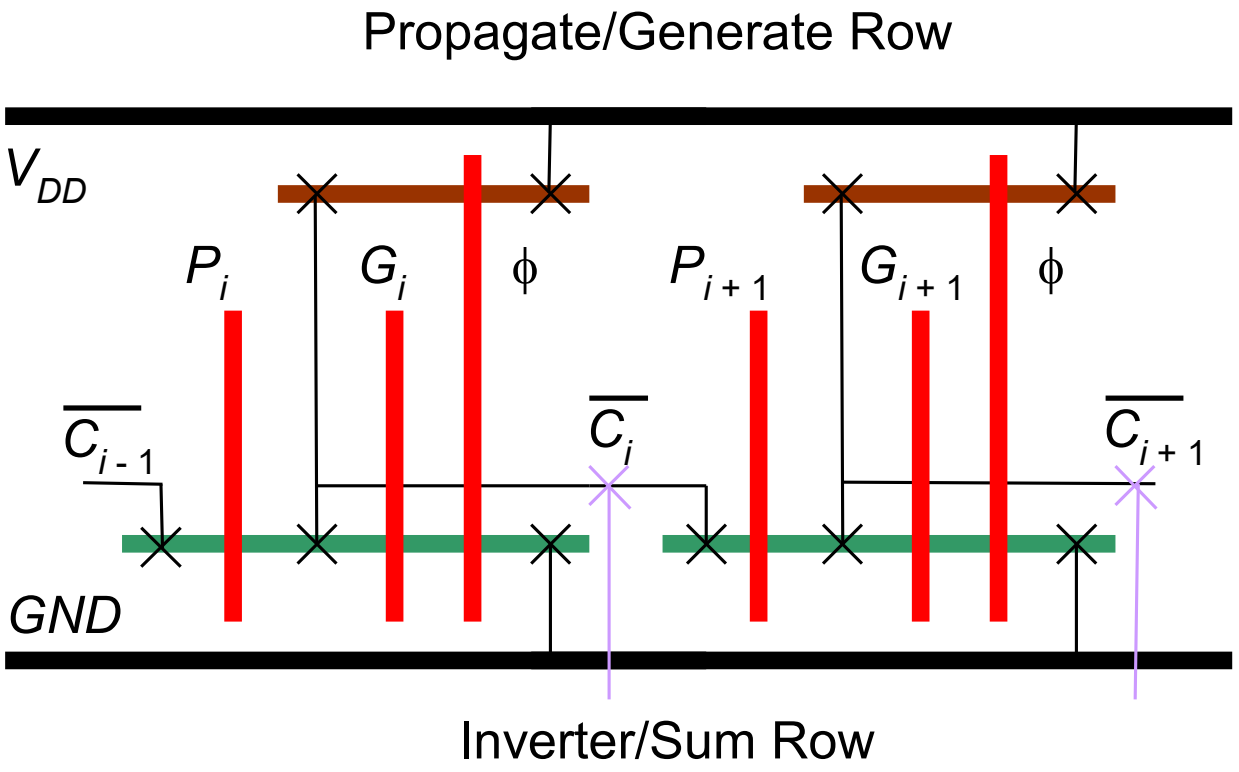
A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

Manchester Carry Chain



Manchester Carry Chain

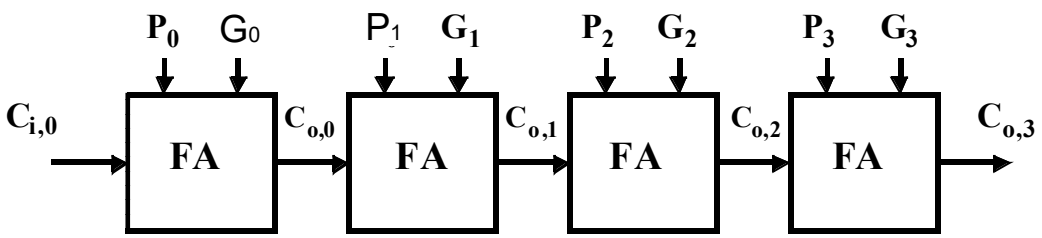
Stick Diagram



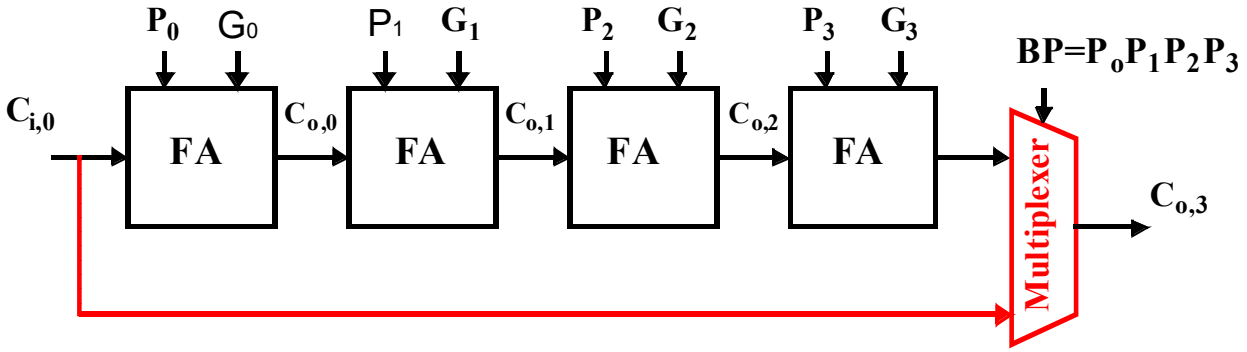
Manchester Carry Chain

- Delay for the Manchester Carry Chain can be modeled similar to a linearized RC network as in transmission-gates
- This means the propagation delay is quadratic in the number of bits *N* (but does not imply the delay will be larger than the ripple carry adder)
- It might be necessary to insert signal buffering inverters.
- Still a ripple carry adder, typically only good for small word length (<8/16 bits)
- We need faster adders for computer and multimedia applications with word length 32-128 bits

Carry-Bypass Adder



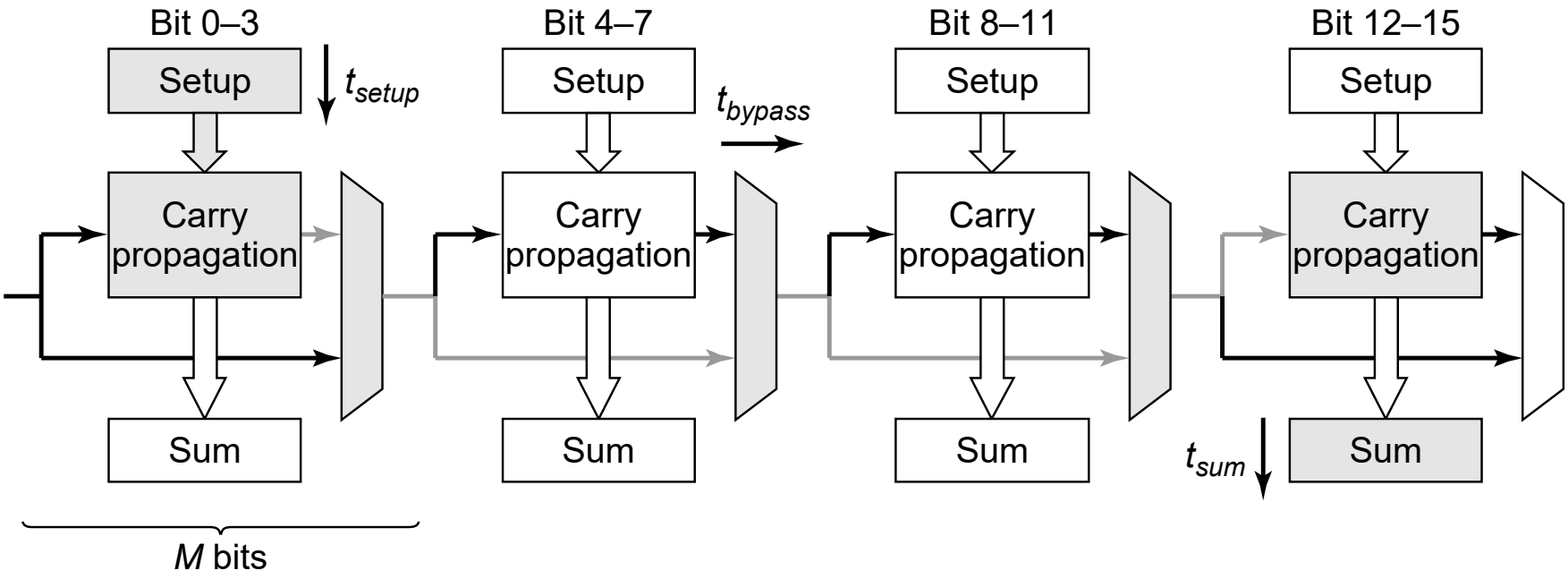
Also called Carry-Skip



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

Break the bit-slice organization

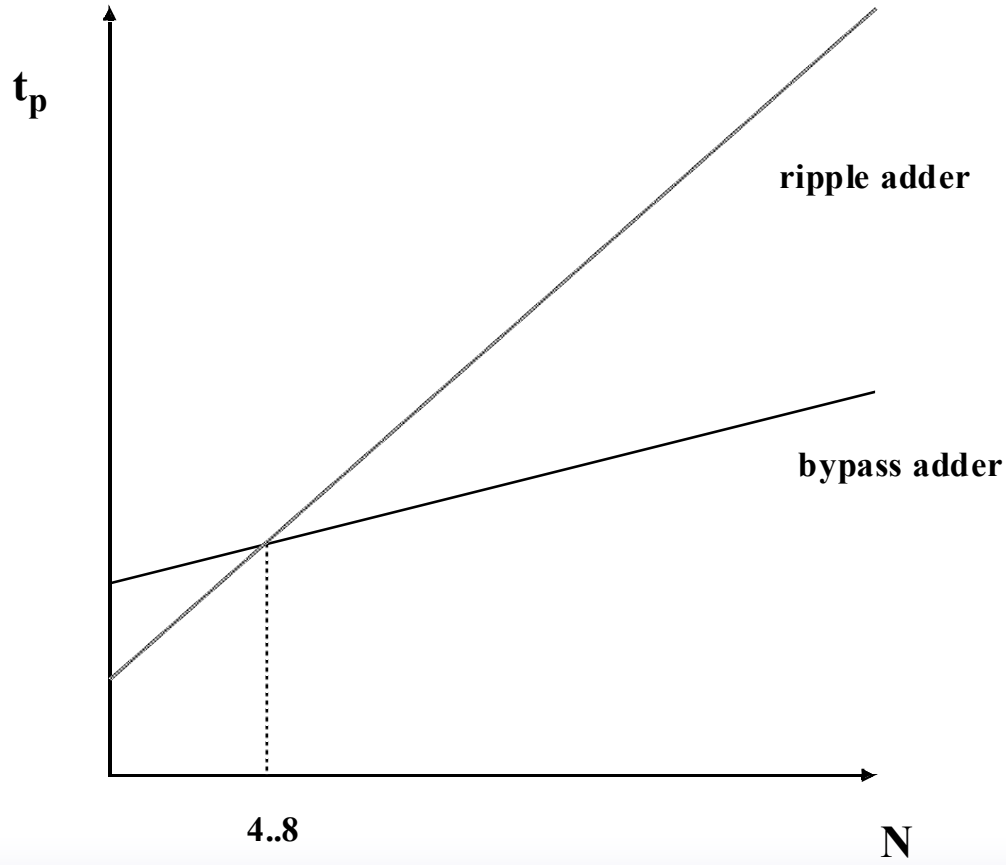
Carry-Bypass Adder (cont.)



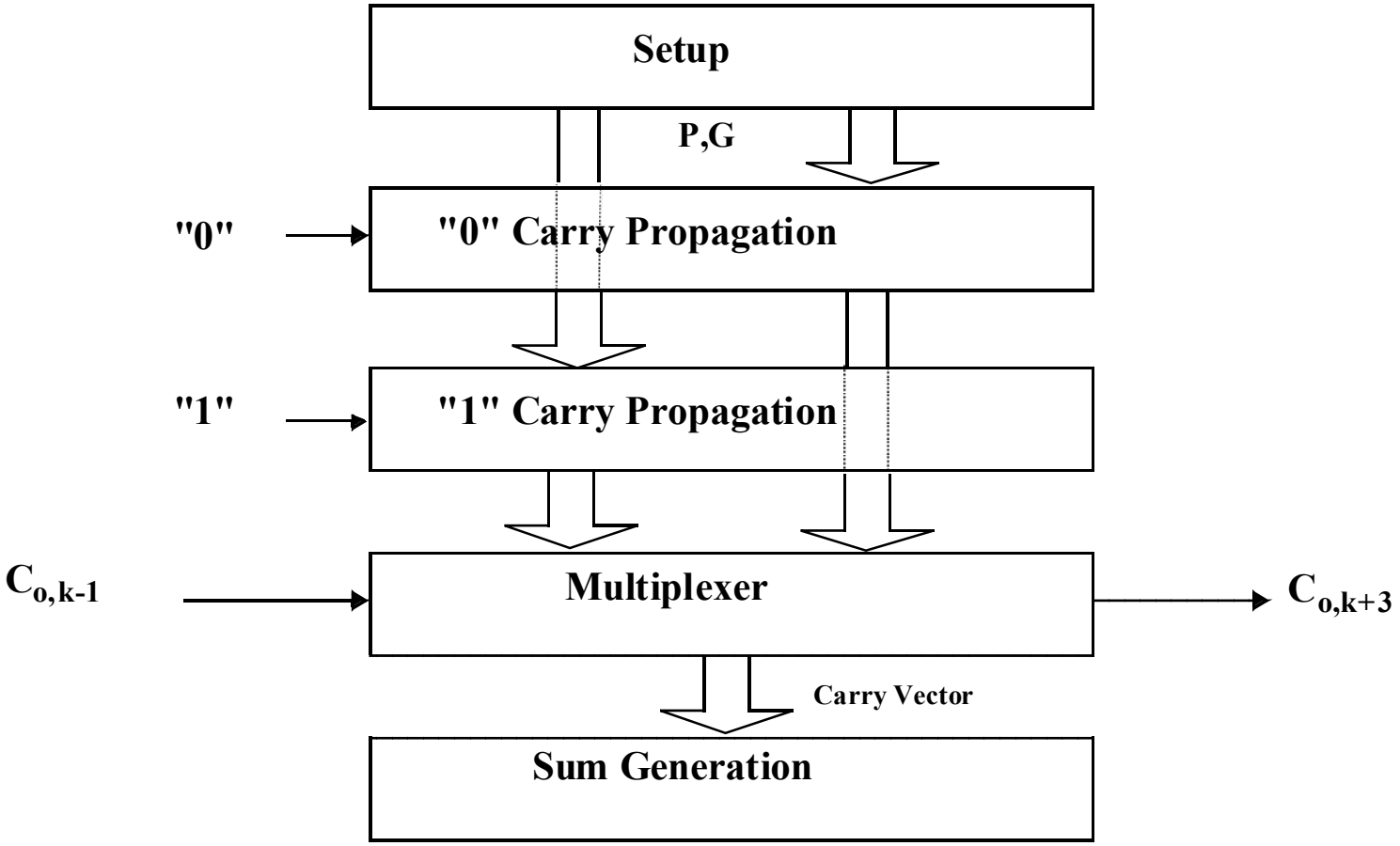
$$t_{adder} = t_{setup} + Mt_{carry} + (N/M\text{-}1)t_{bypass} + (M\text{-}1)t_{carry} + t_{sum} \quad \textit{(worst case)}$$

T_{setup}: *overhead time to create G, P, D signals*

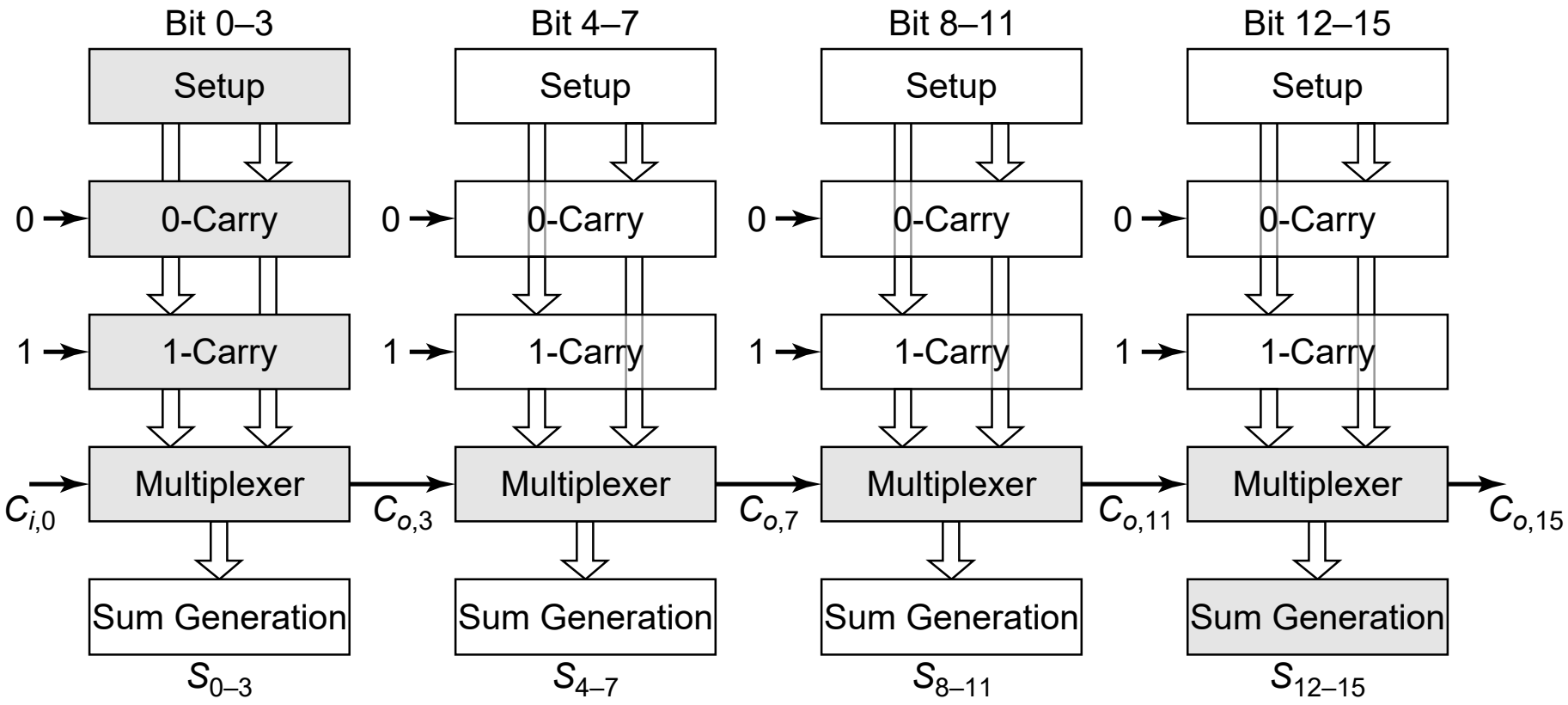
Carry Ripple versus Carry Bypass (both still linear)



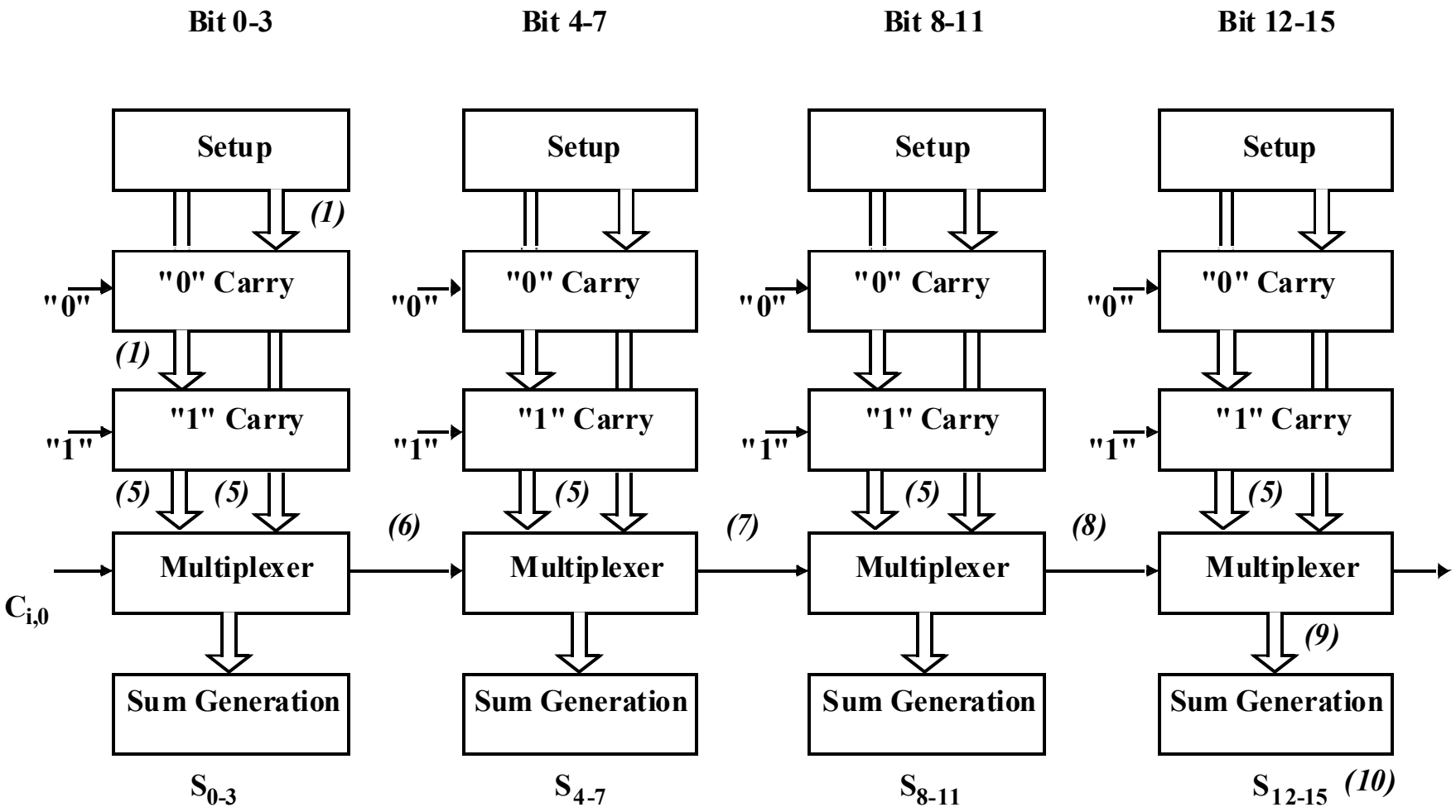
Carry-Select Adder



Carry Select Adder: Critical Path

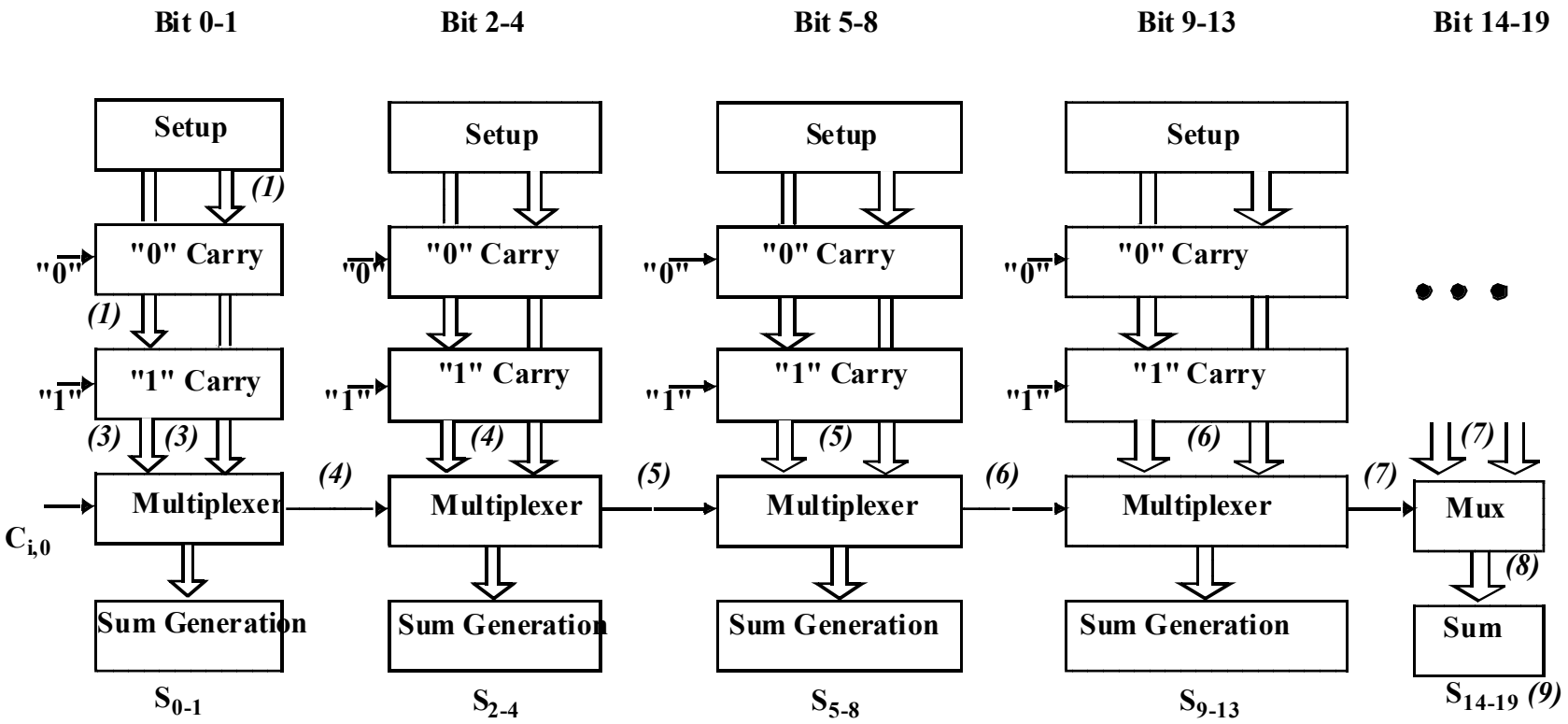


Linear Carry Select



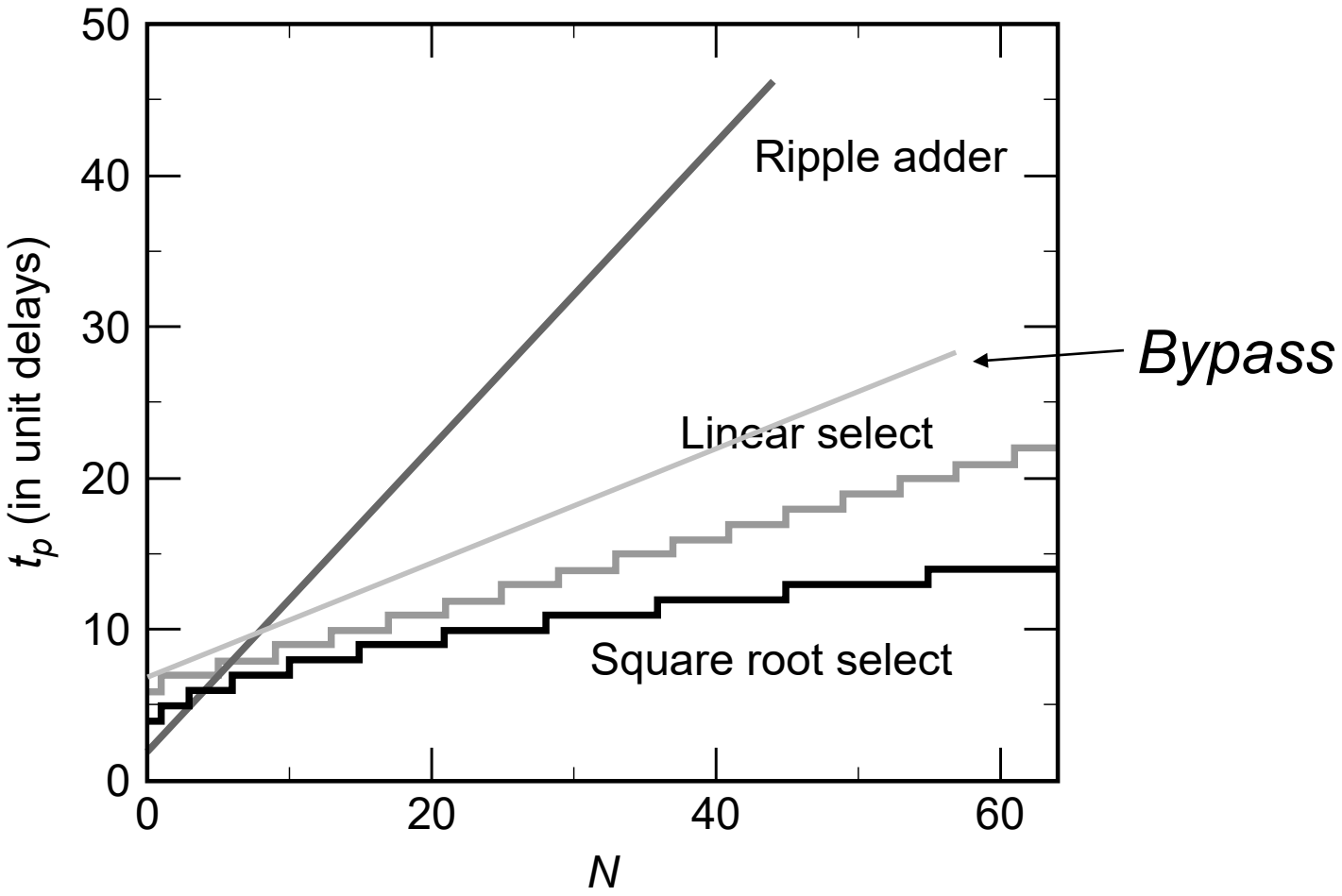
$$t_{adder} = t_{setup} + Mt_{carry} + (N/M)t_{mux} + t_{sum}$$

Square Root Carry Select

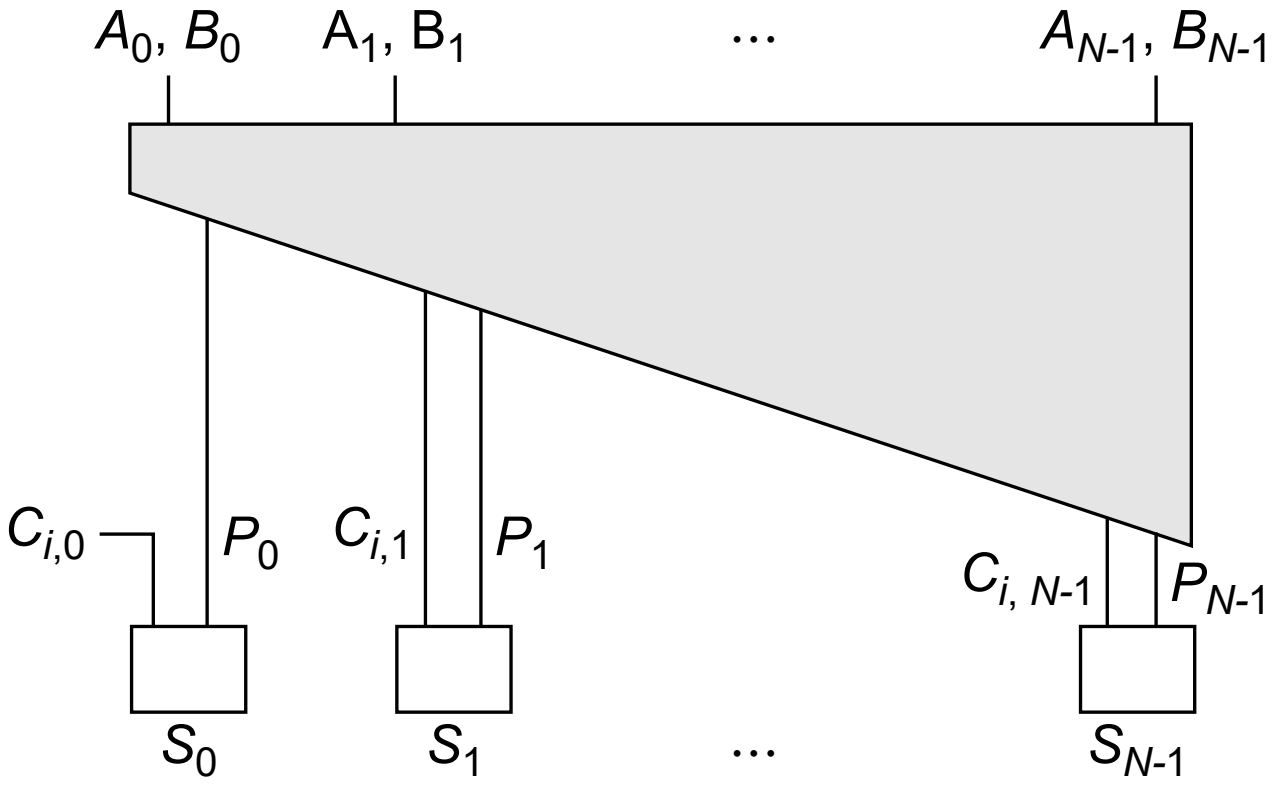


$$t_{add} = t_{setup} + Mt_{carry} + (\sqrt{2N})t_{mux} + t_{sum}$$

Adder Delays - Comparison



LookAhead - Basic Idea



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

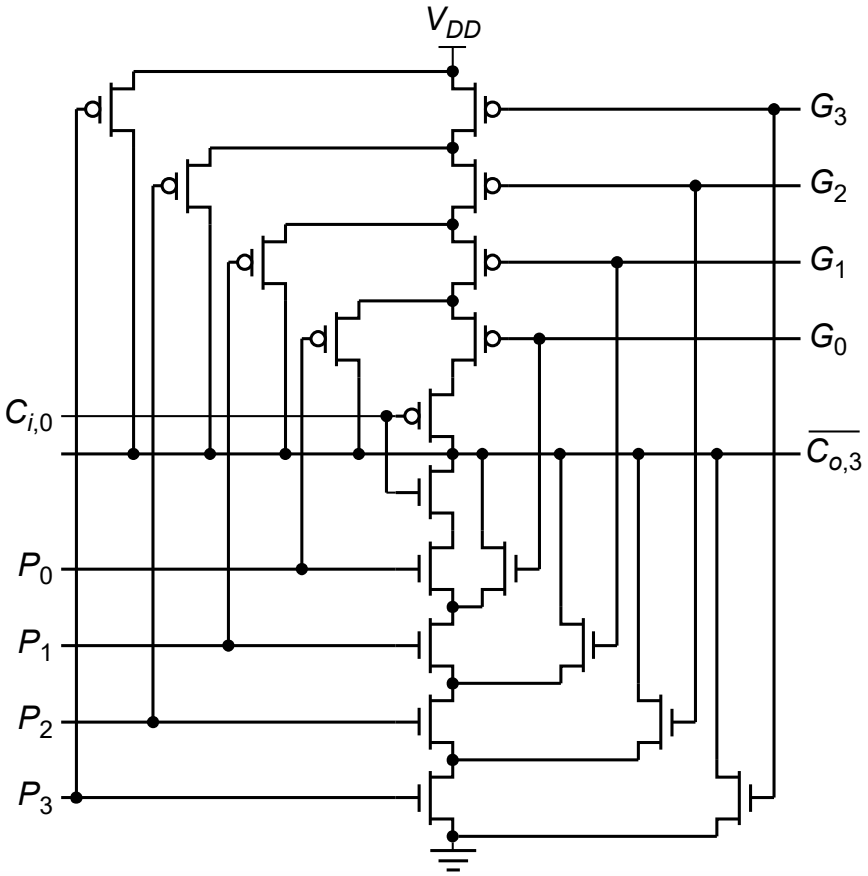
Look-Ahead: Topology

Expanding Lookahead equations:

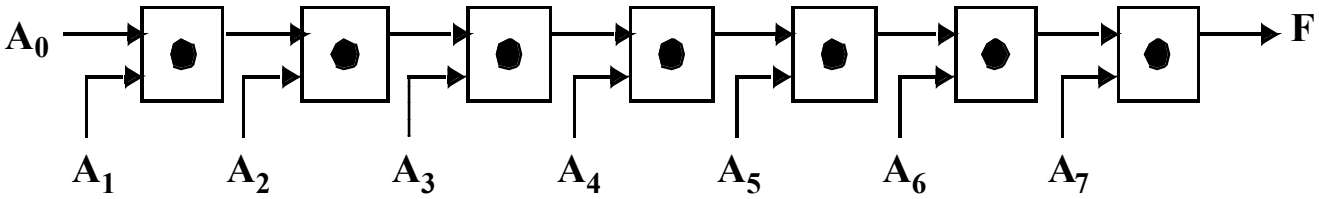
$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}C_{o,k-2})$$

All the way:

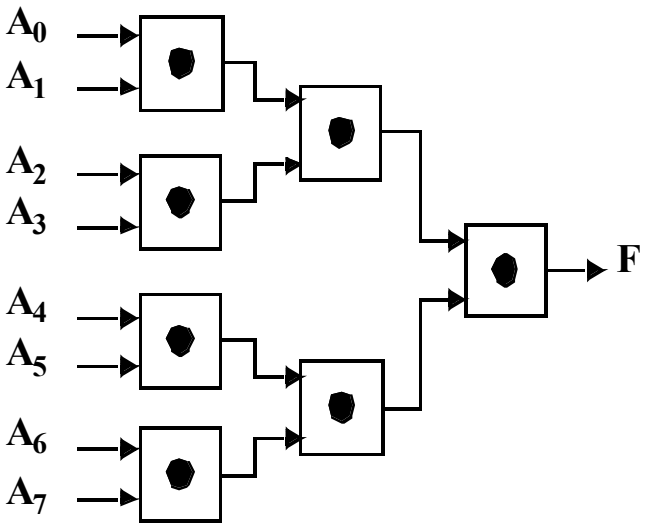
$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}(\ldots + P_1(G_0 + P_0 C_{i,0})))$$



Look-Ahead Adder: Logarithmic adder



$t_p \sim N$



$t_p \sim \log_2(N)$

Carry Look-Ahead Trees

$C_0 = G_0 + P_0 C_{in}$

$C_1 = G_1 + P_1 C_0$

$C_2 = G_2 + P_2 C_1$

$C_3 = G_3 + P_3 C_2$

$C_0 = G_0 + P_0 C_{in}$

$C_1 = G_1 + P_1 C_0 = G_1 + G_0 P_1 + P_1 P_0 C_{in} = G_{1:0} + P_{1:0} C_0$

$(G_{1:0} = G_1 + P_1 G_0 \qquad P_{1:0} = P_1 P_0)$

$C_2 = G_2 + P_2 C_1 = G_2 + G_1 P_2 + G_0 P_2 P_1 + P_2 P_1 P_0 C_{in} = G_{2:1} + P_{2:1} C_0$

$(G_{2:1} = G_2 + P_2 G_1 \qquad P_{2:1} = P_2 P_1)$

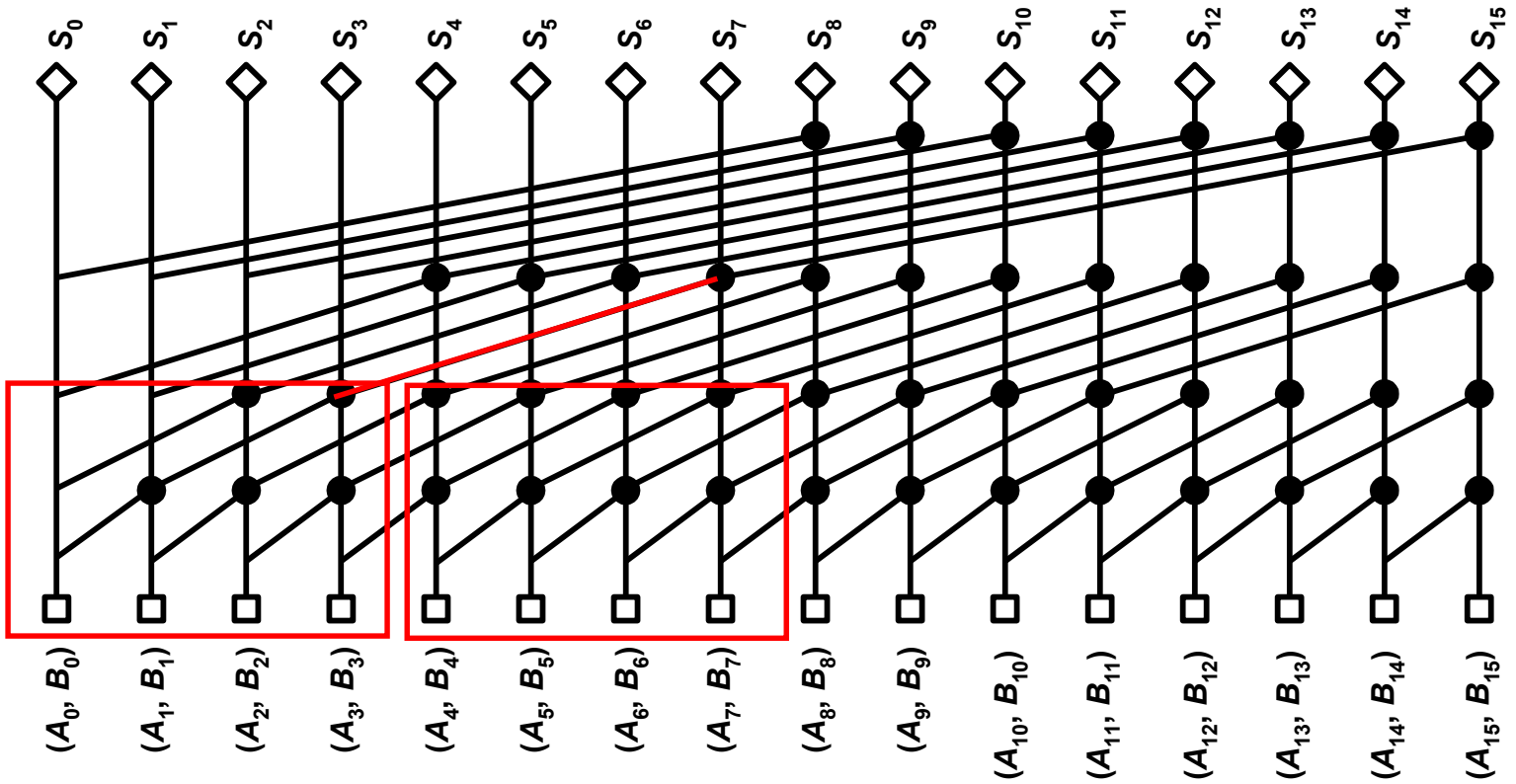
$C_3 = G_3 + P_3 C_2 = G_3 + G_2 P_3 + G_1 P_3 P_2 + G_0 P_3 P_2 P_1 + P_3 P_2 P_1 P_0 C_{in}$

$= G_{3:2} + P_{3:2} C_1 = G_{3:2} + P_{3:2}(G_{1:0} + P_{1:0} C_0) = (G_{3:2} + P_{3:2} G_{1:0}) + P_{3:2} P_{1:0} C_0$

Can continue building the tree hierarchically.

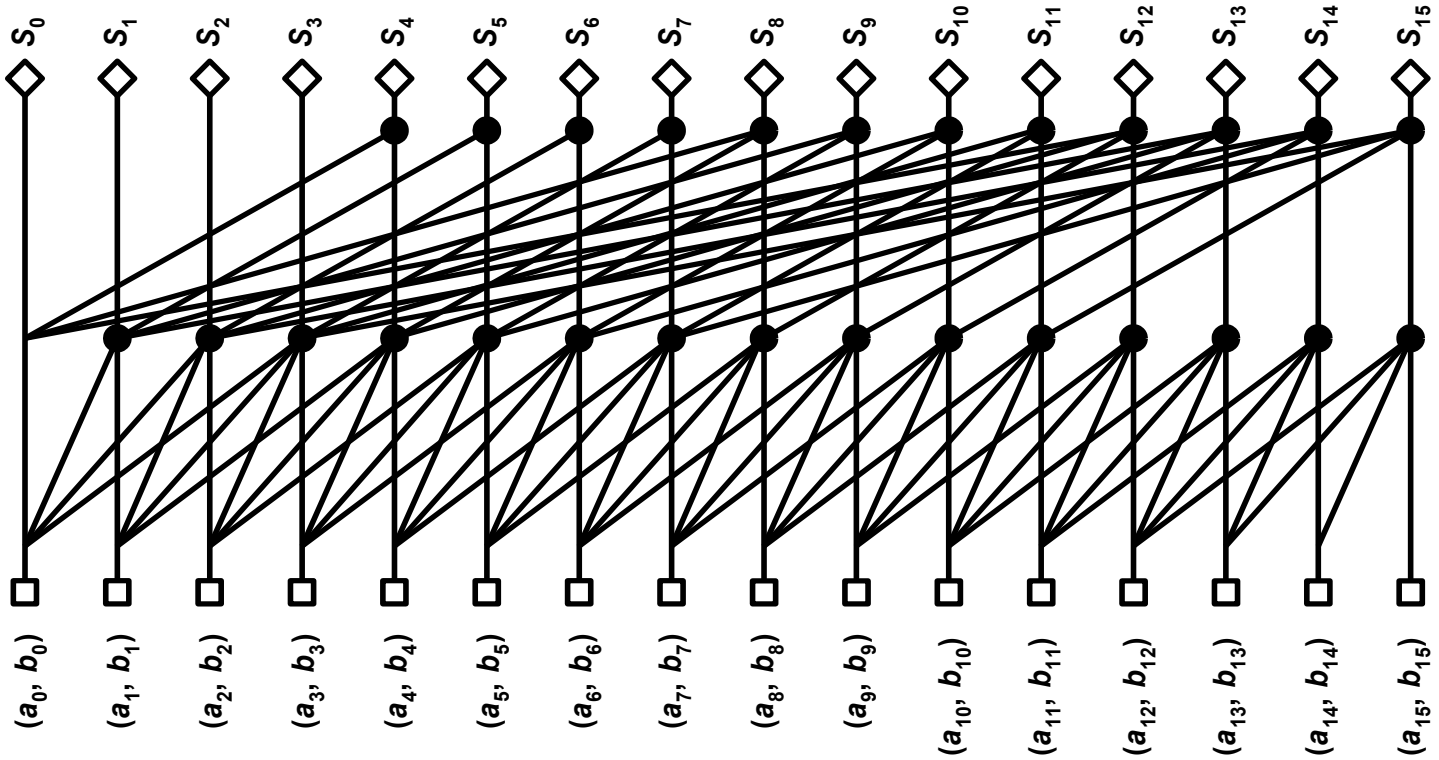
$G_{3:2} = (G_3 + P_3 G_2)$ and $P_{3:2} = P_3 P_2$ are called dot products.

Tree Adders



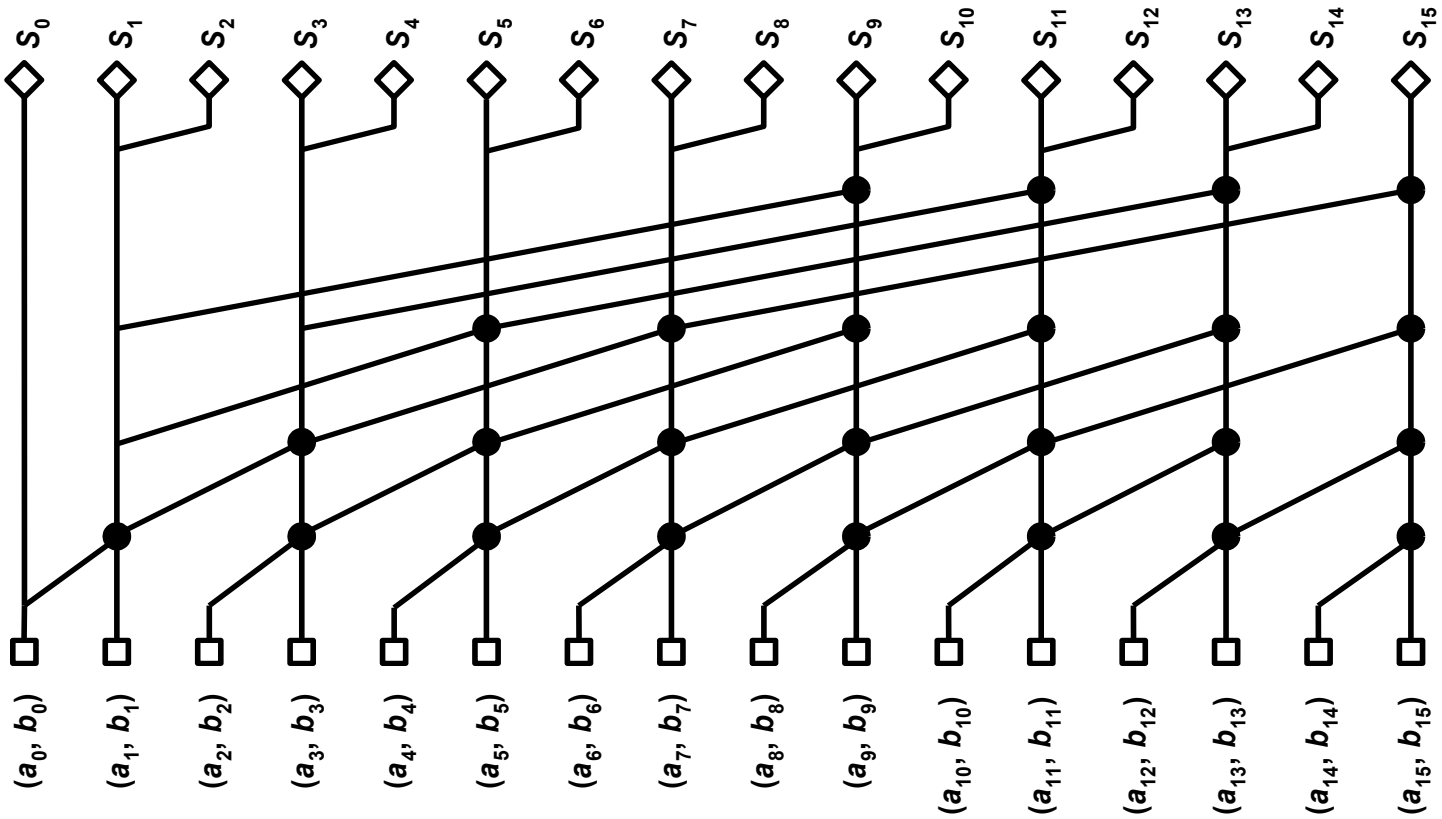
16-bit radix-2 Kogge-Stone tree (radix 2 means that the tree is Binary: it combines two dot product or carry words at a time at Each level of hierarchy)

Tree Adders



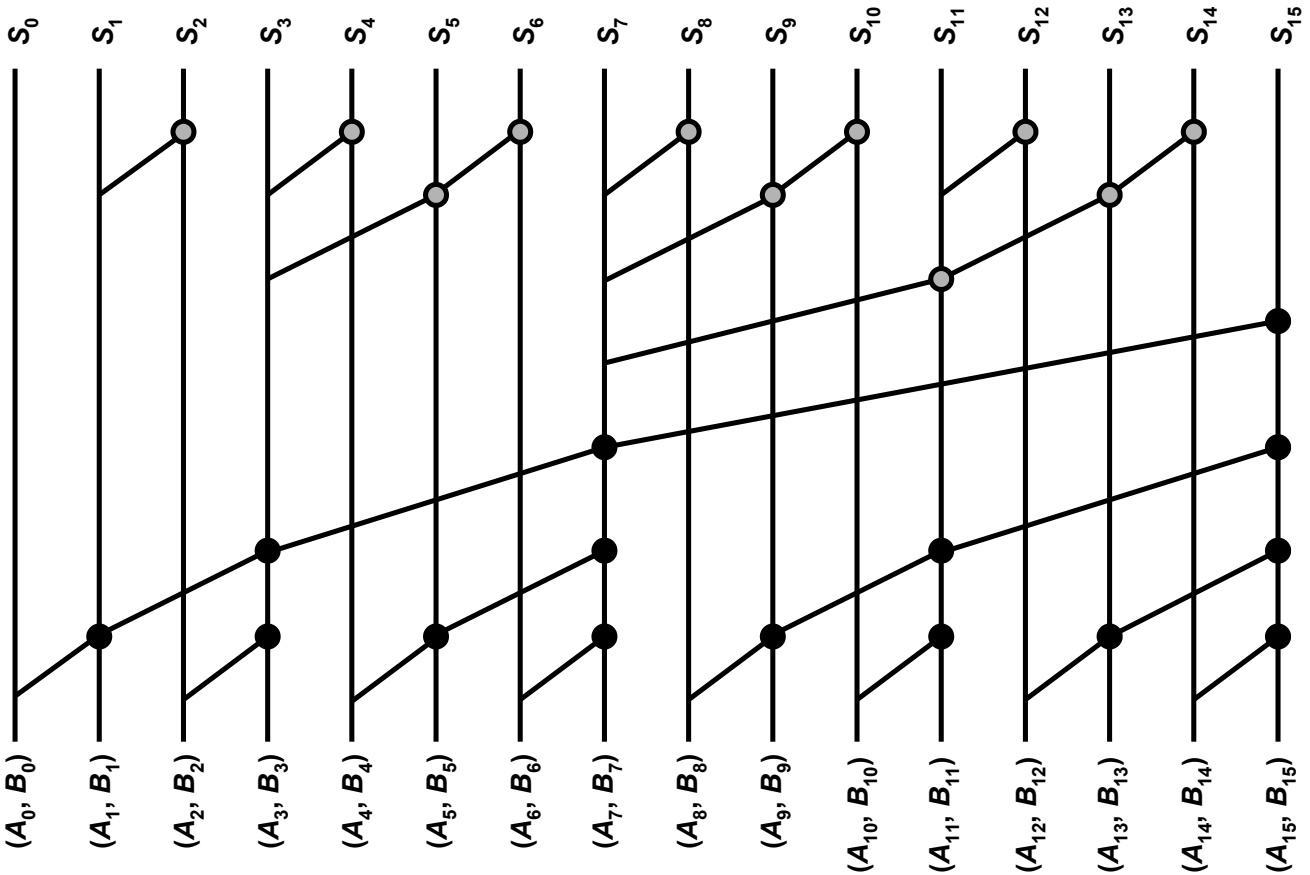
16-bit radix-4 Kogge-Stone Tree

Sparse Trees



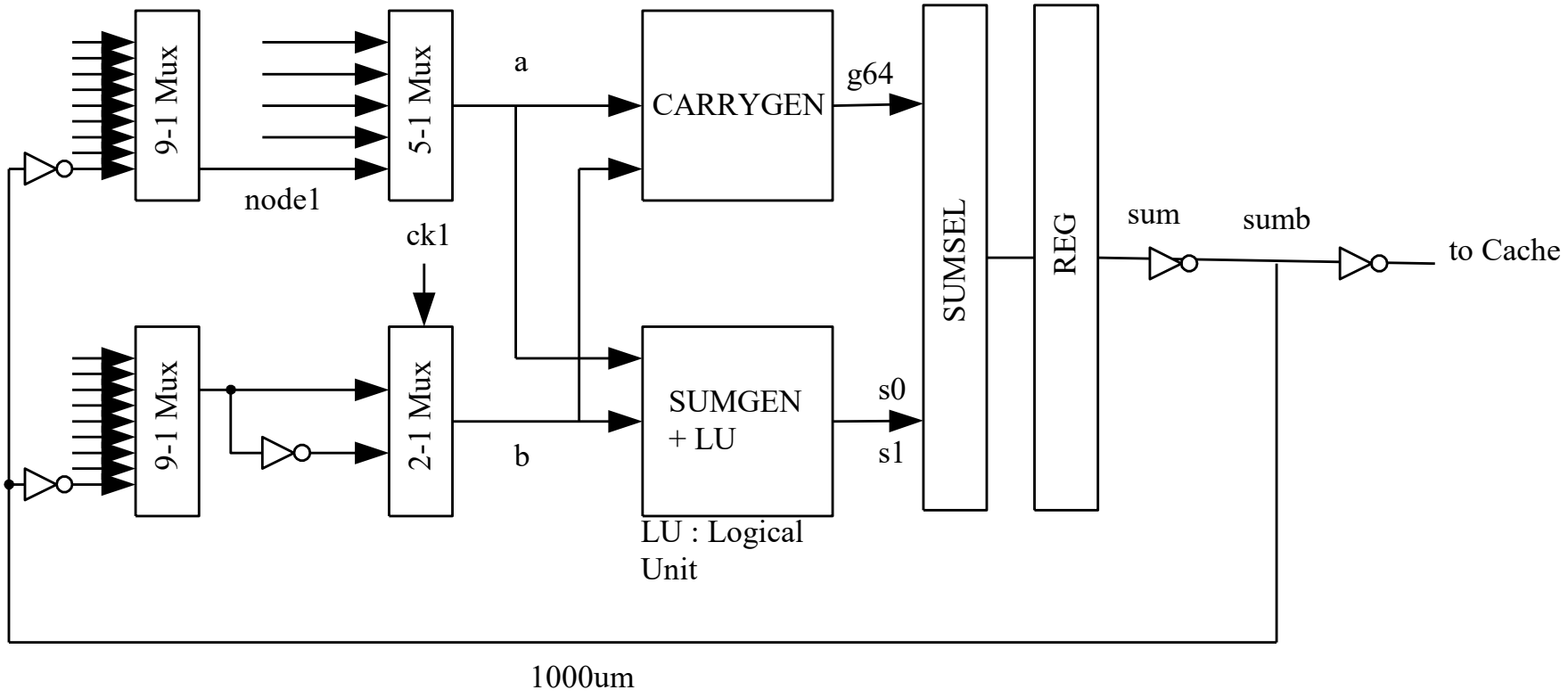
16-bit radix-2 sparse tree with sparseness of 2

Tree Adders



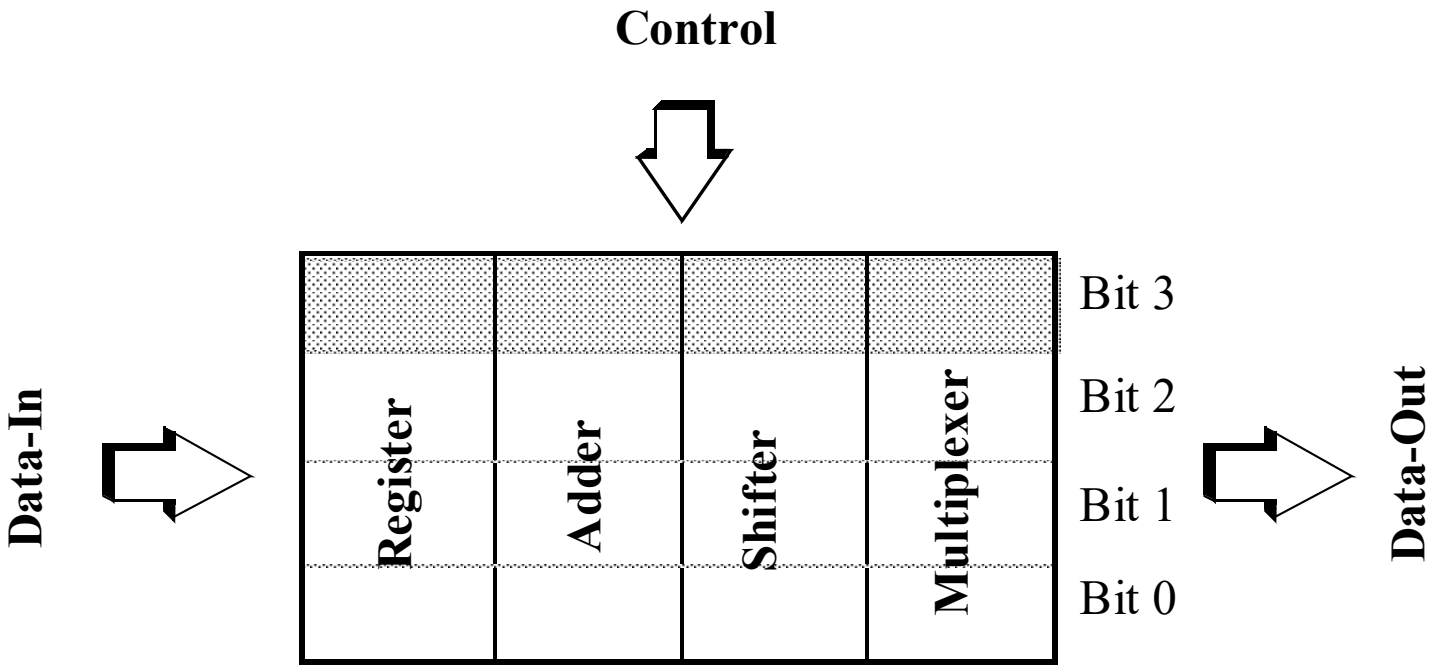
Brent-Kung Tree

Intel Itanium Microprocessor



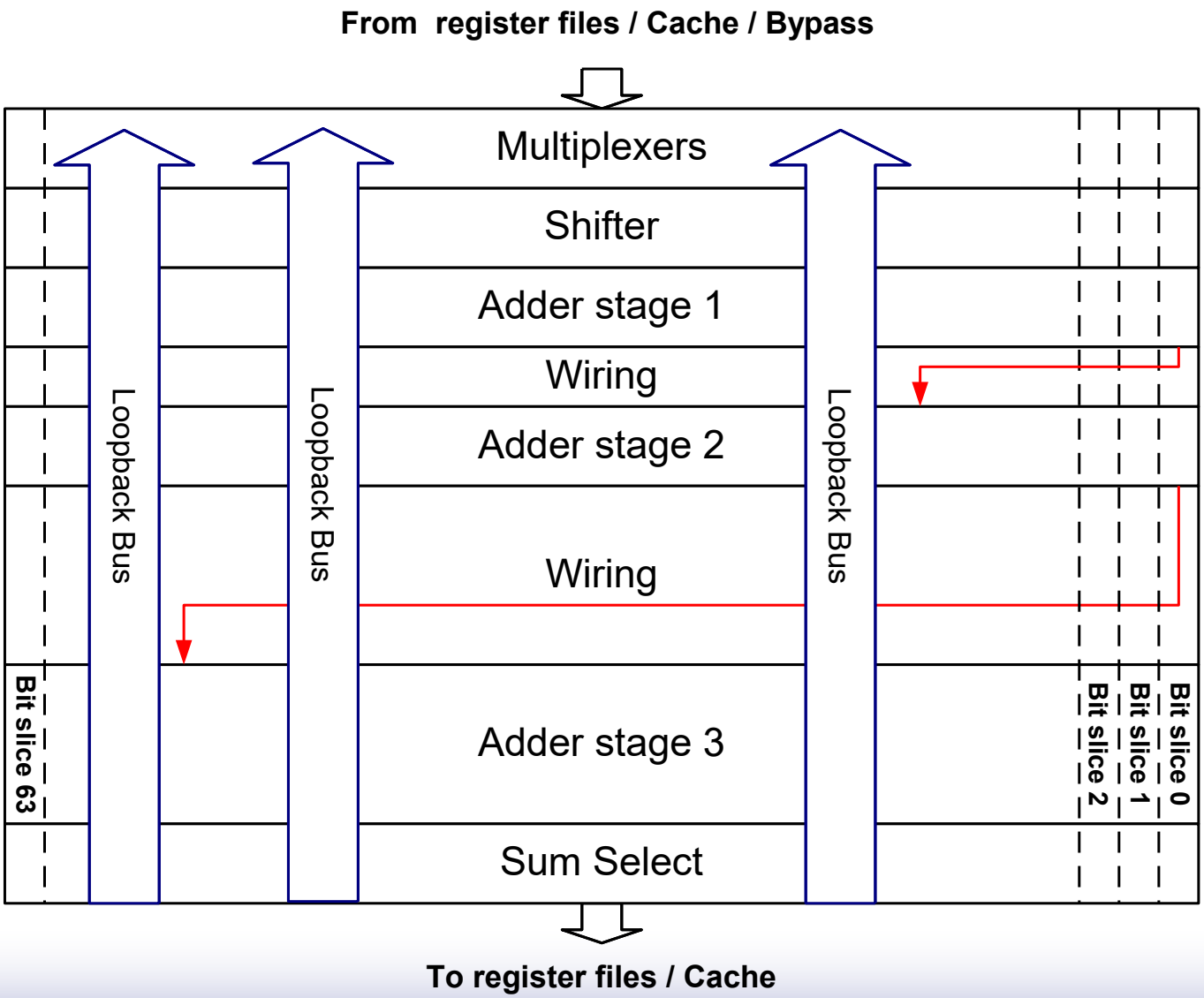
Itanium has 6 integer execution units like this

Bit-Sliced Design



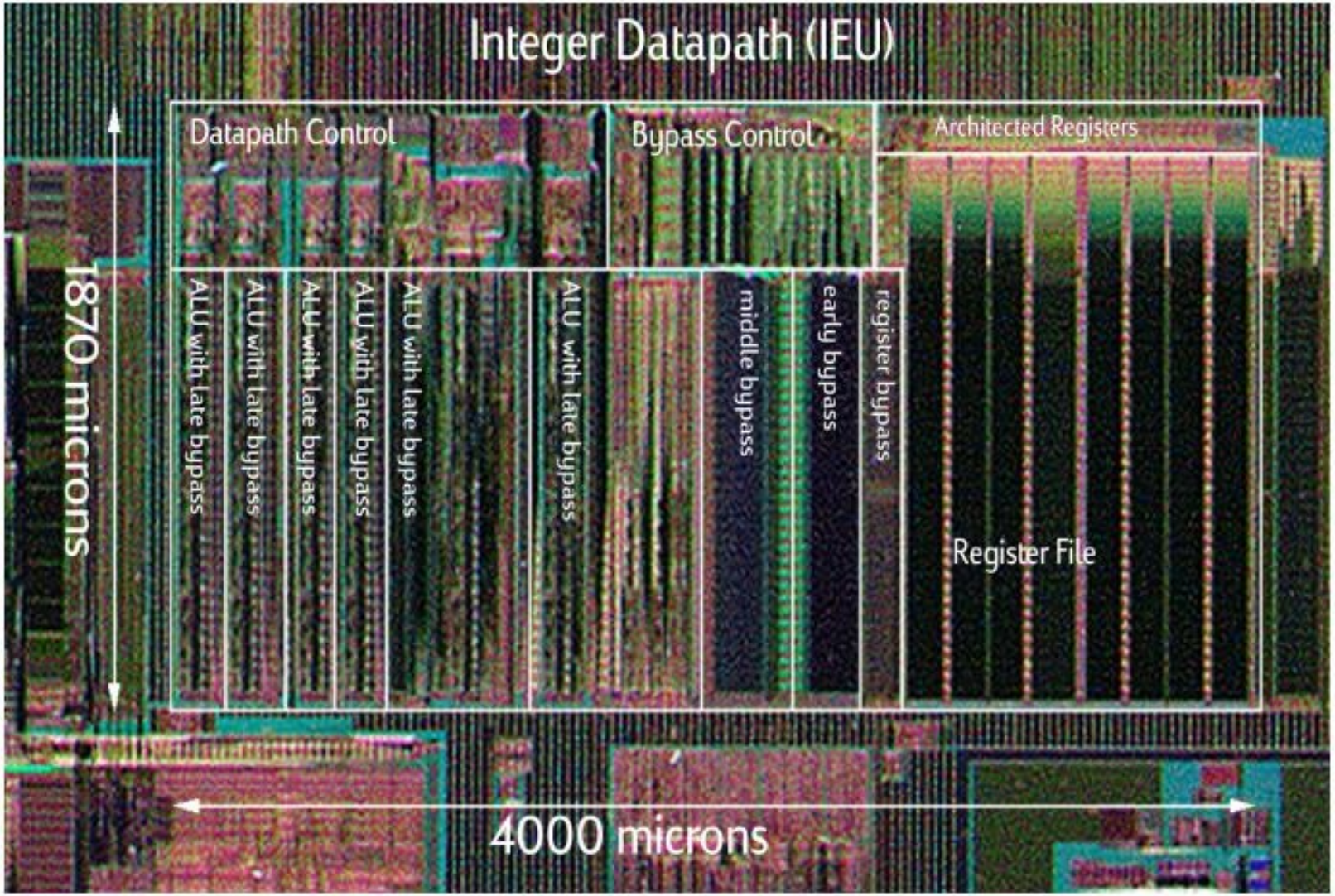
Tile identical processing elements

Bit-Sliced Datapath



The adder is implemented as a radix-4 Carry Look-Ahead adder, the red lines are forwarding the results of different stages

Itanium Integer Datapath



Multipliers

The Binary Multiplication

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

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

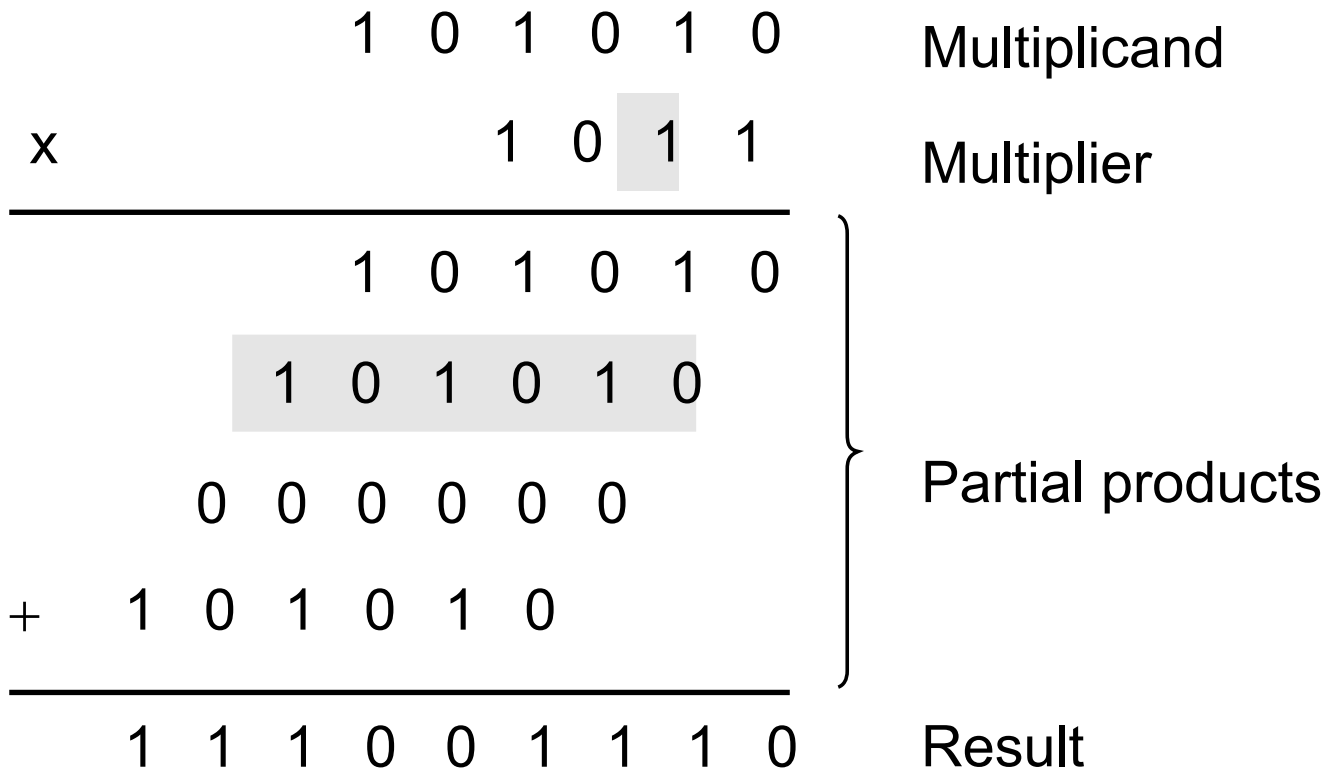
$$= \sum_{i=0}^{M-1} \left(\sum_{j=0}^{N-1} X_i Y_j 2^{i+j} \right)$$

with

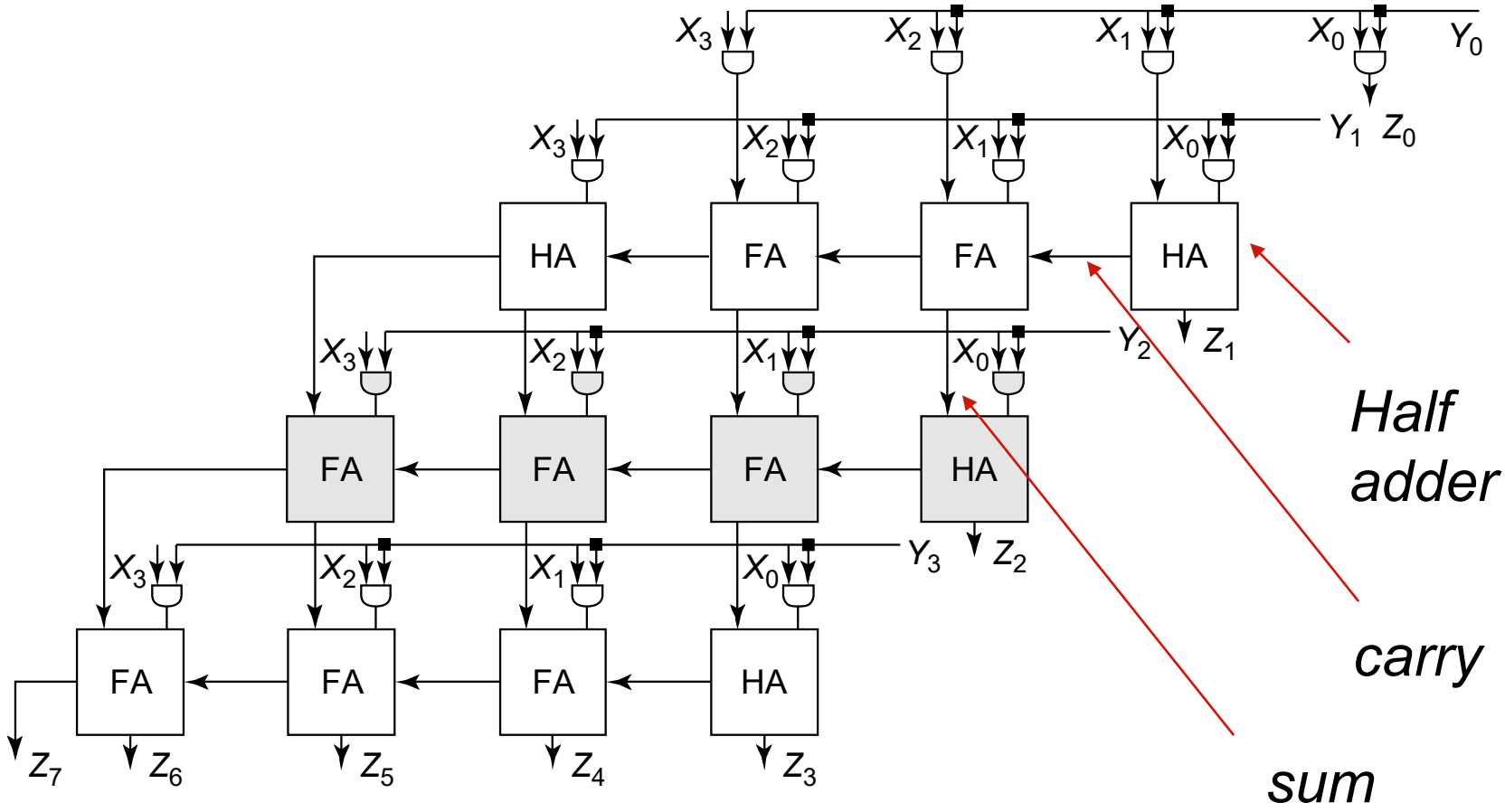
$$X = \sum_{i=0}^{M-1} X_i 2^i$$

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

The Binary Multiplication

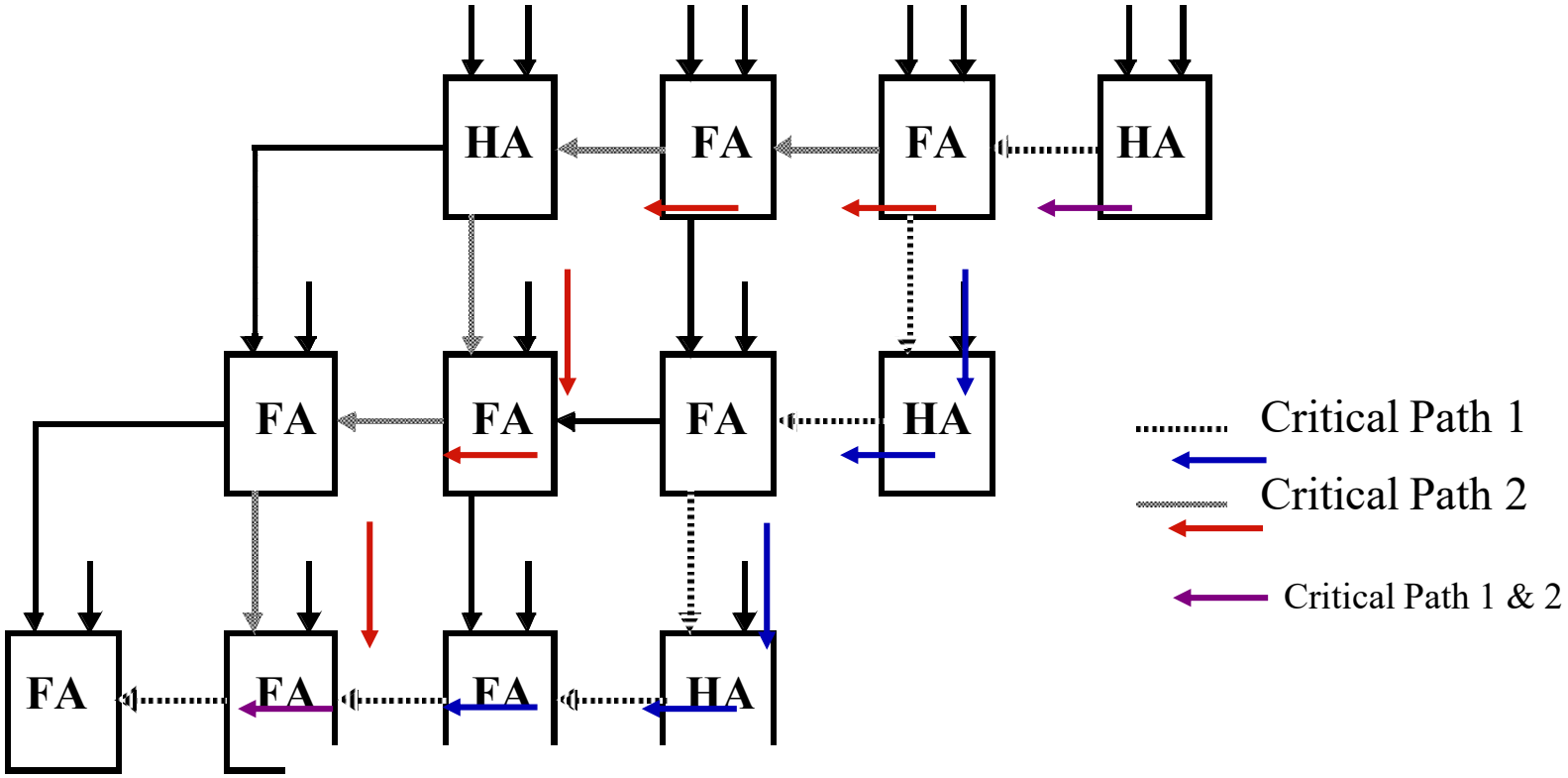


The Array Multiplier (4 by 4)



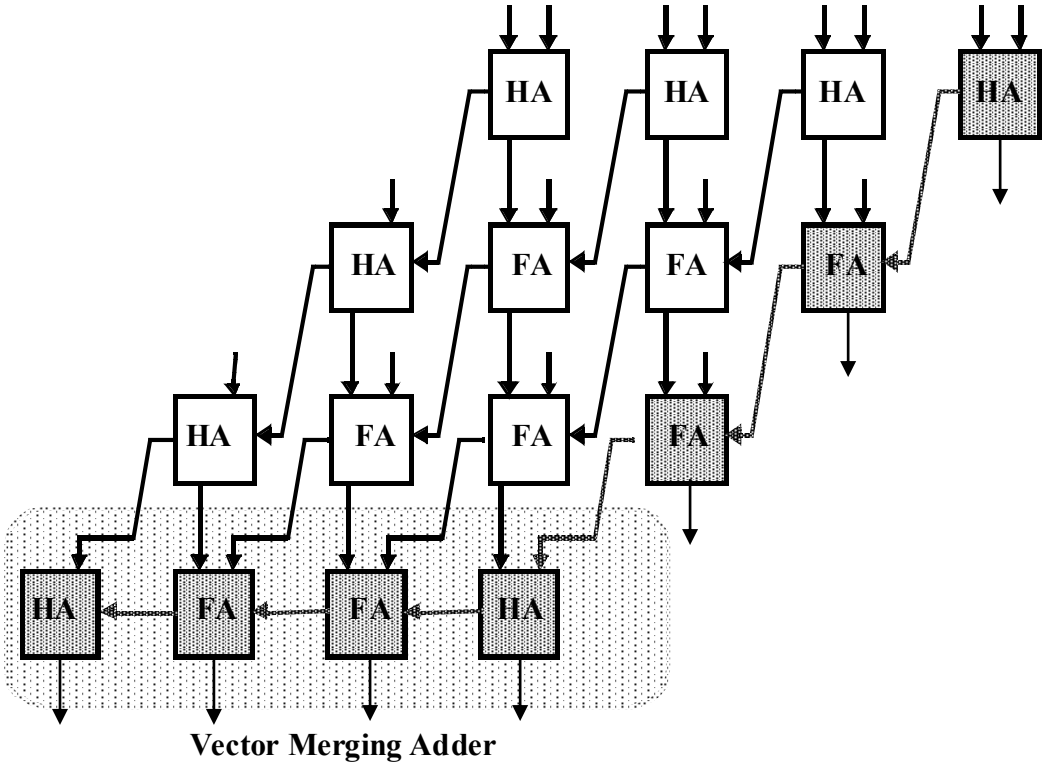
The carryout of the last adder for Y_i is forwarded to Y_{i+1}

The MxN Array Multiplier — Critical Path



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

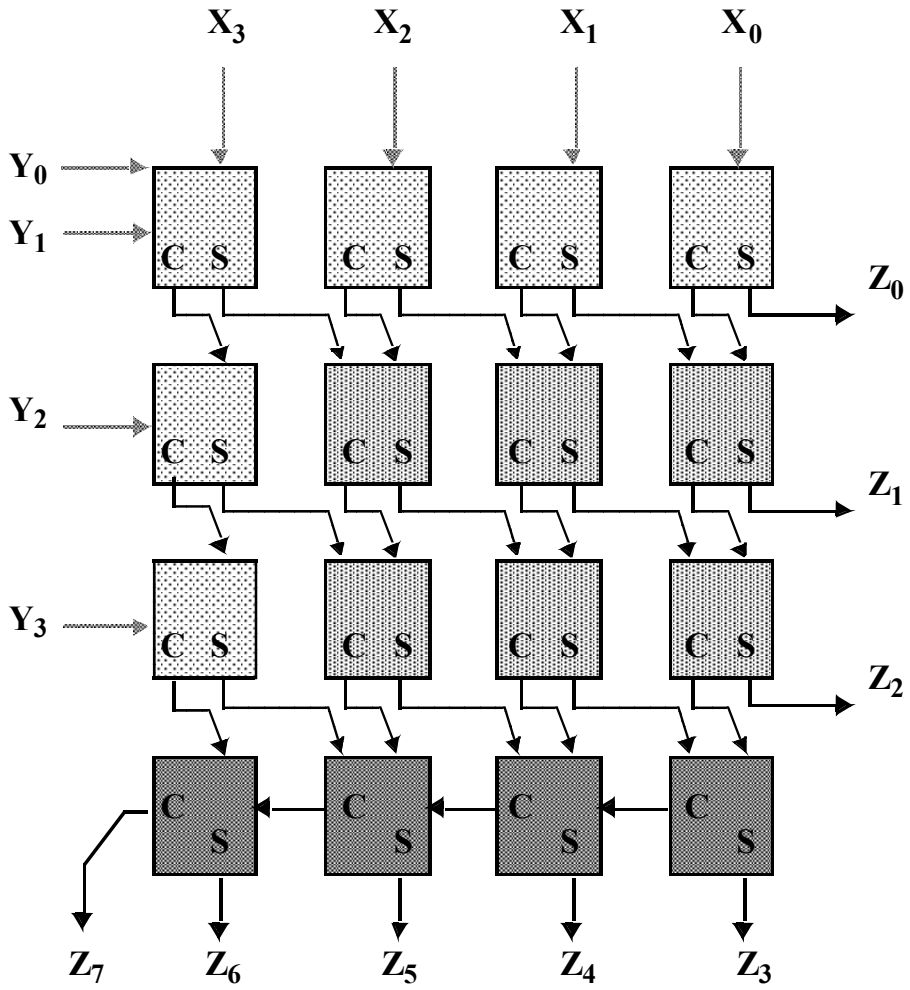
Carry-Save Multiplier



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

- A more efficient realization can be obtained by noticing that the multiplication results does not change when the output carry bits are passed diagonally downwards instead of to the right.
- But need extra adders (vector merging adders) that can use fast carry look ahead adders (since results come at the same time)
- Critical path is uniquely defined

Multiplier Floorplan



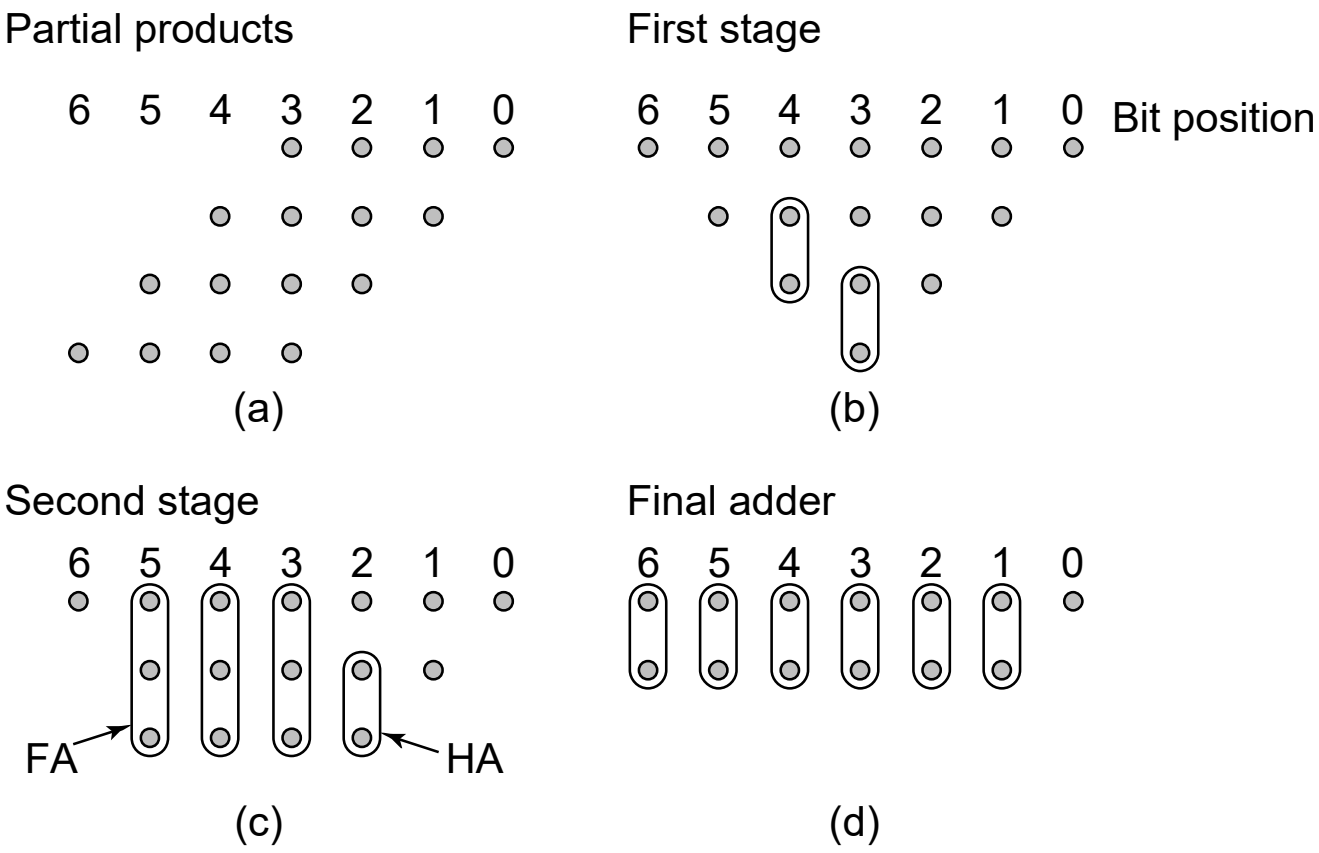
HA Multiplier Cell

FA Multiplier Cell

Vector Merging Cell

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

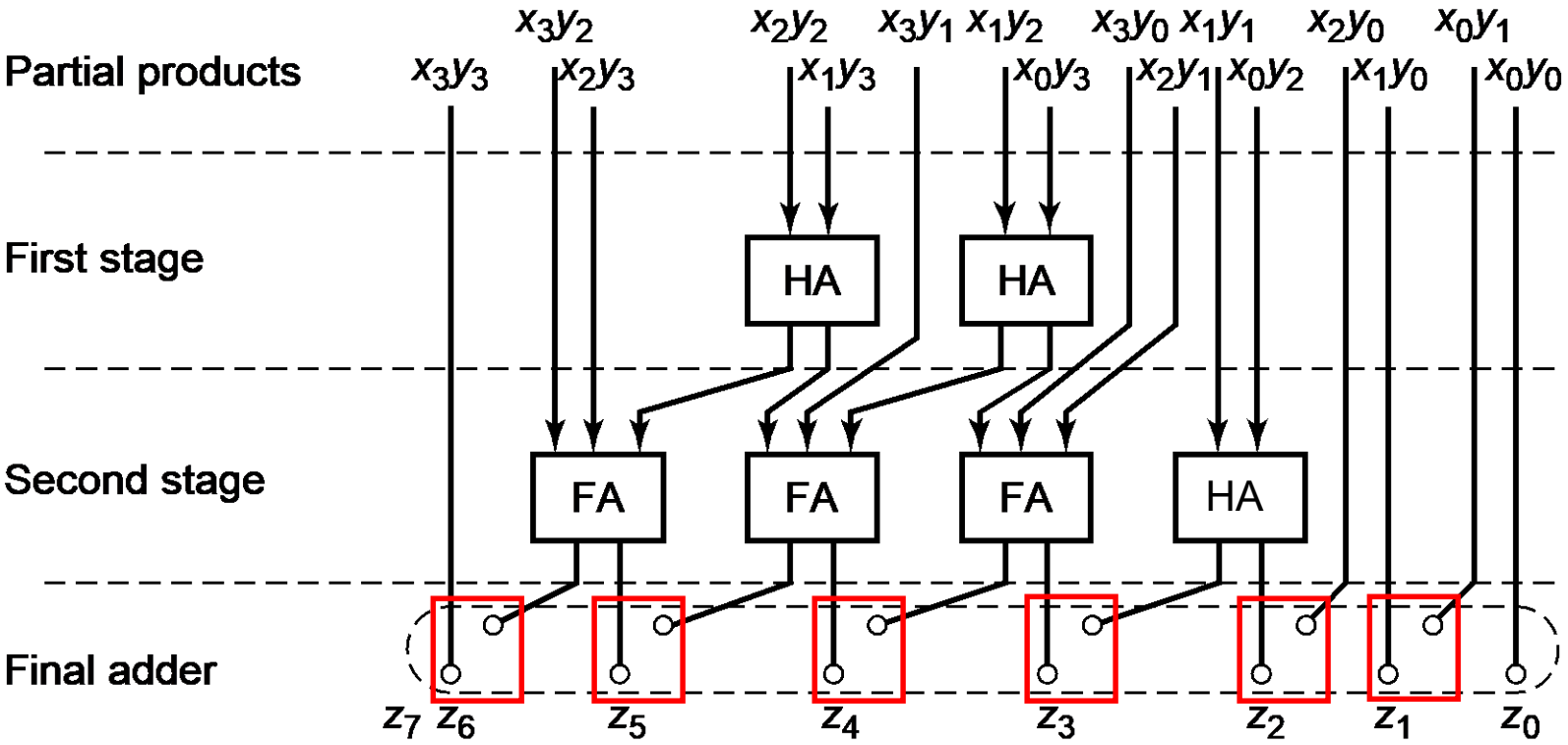
Wallace-Tree Multiplier



Save the number of full adders

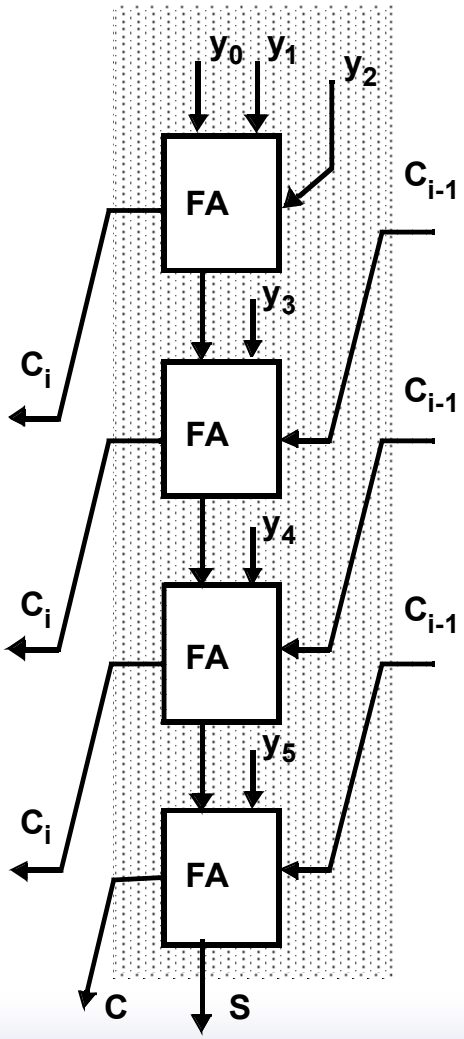
Increase the complexity of routing

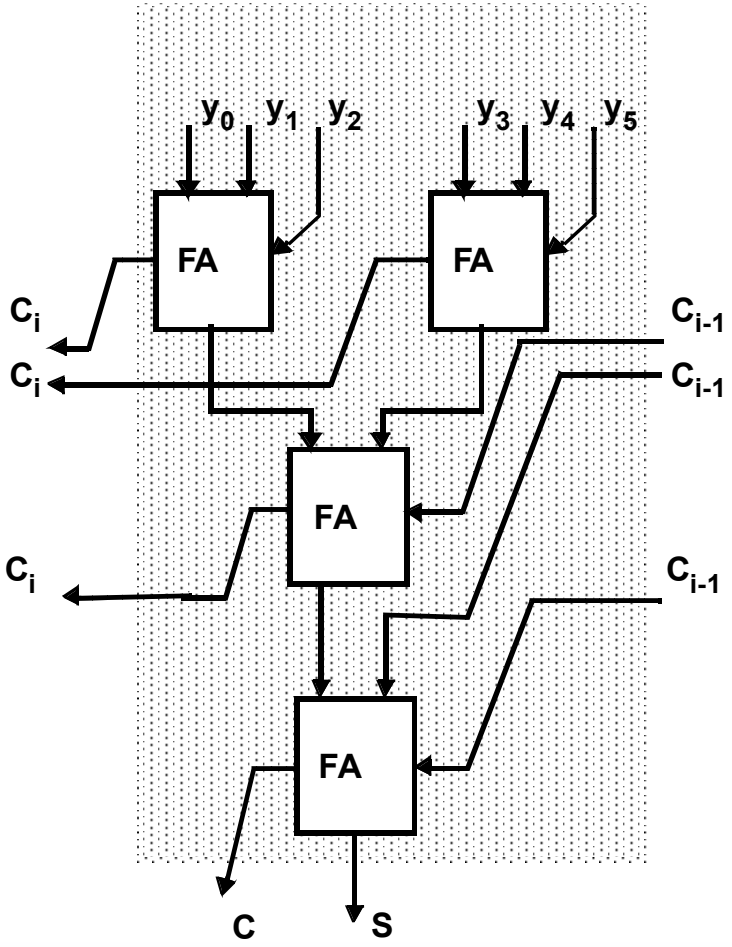
Wallace-Tree Multiplier



Can use carry Look-Ahead adder for the last stage

Wallace-Tree Multiplier





Booth encoding

- Multiply by 01111110 gives 8 partial products, but two are all zero. Add these zero is waste of time.
- Instead, multiply by $100000\bar{1}0$, where $\bar{1}$ stands for -1. Then you need to only add (actually subtract) partial products, which improves speed
- This kind of transformation is called *booth encoding*. It reduces the number of partial product to at most half of the original multiplier width.
- The encoding logic is easily incorporated in the overall multiplier design.

Multipliers —Summary

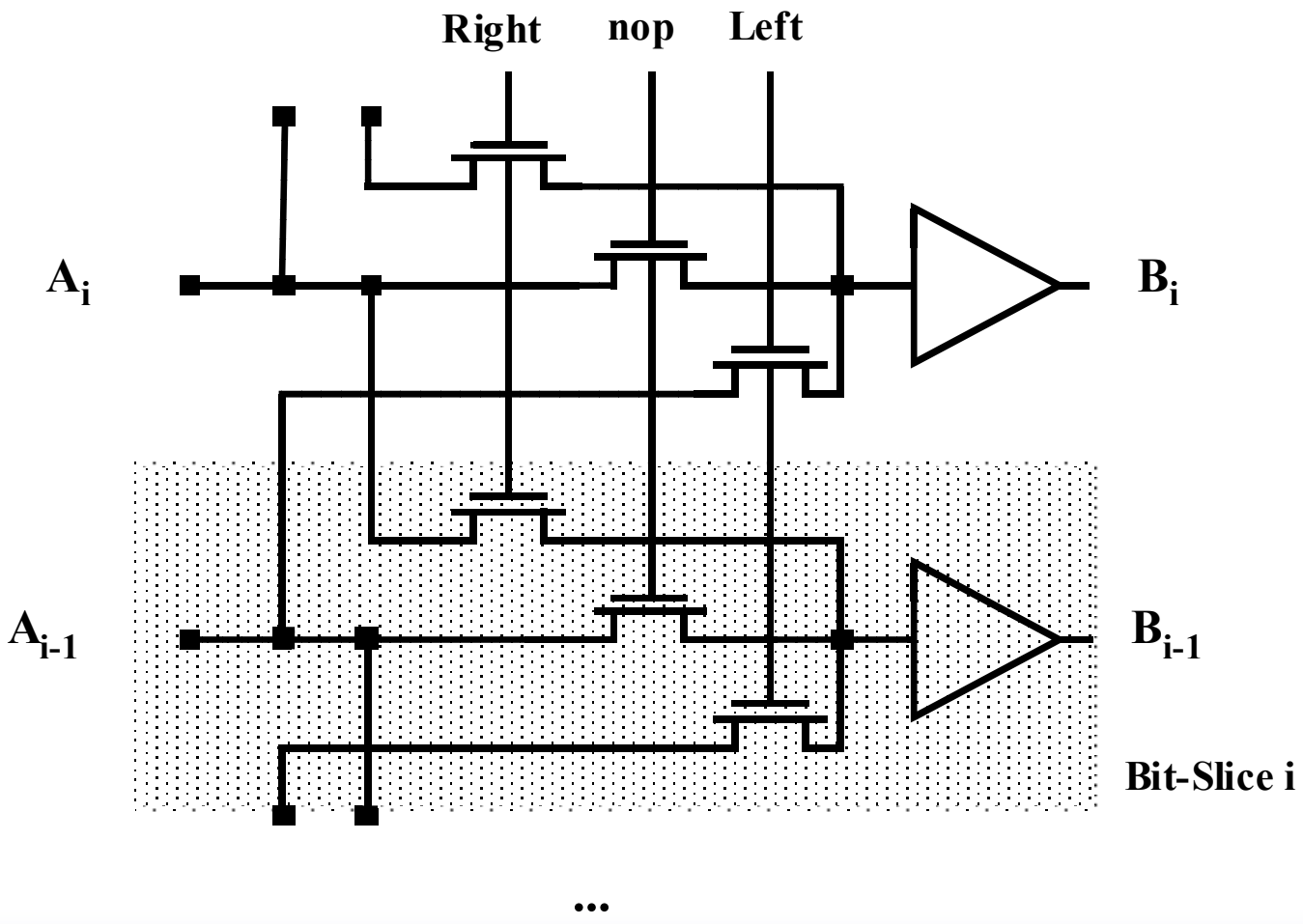
- **Optimization Goals Different Vs Binary Adder**
- **Once Again: Identify Critical Path**
- **Other possible techniques**
  - **Logarithmic versus Linear (Wallace Tree Mult)**
  - **Data encoding (Booth)**
  - **Pipelining**

FIRST GLIMPSE AT SYSTEM LEVEL OPTIMIZATION

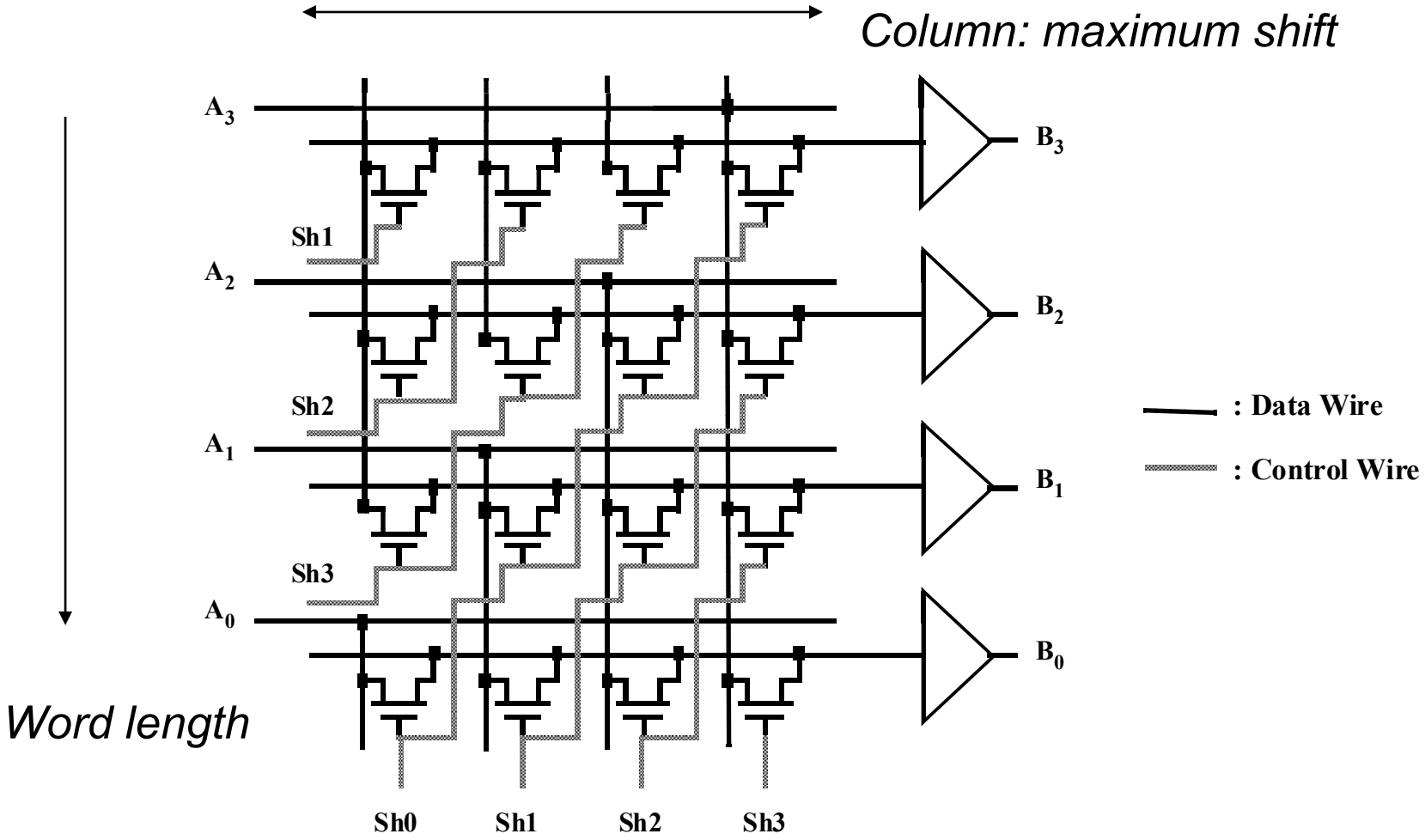
This is also why algorithmic invention has significant meaning to VLSI design.

Shifters

The Binary Shifter

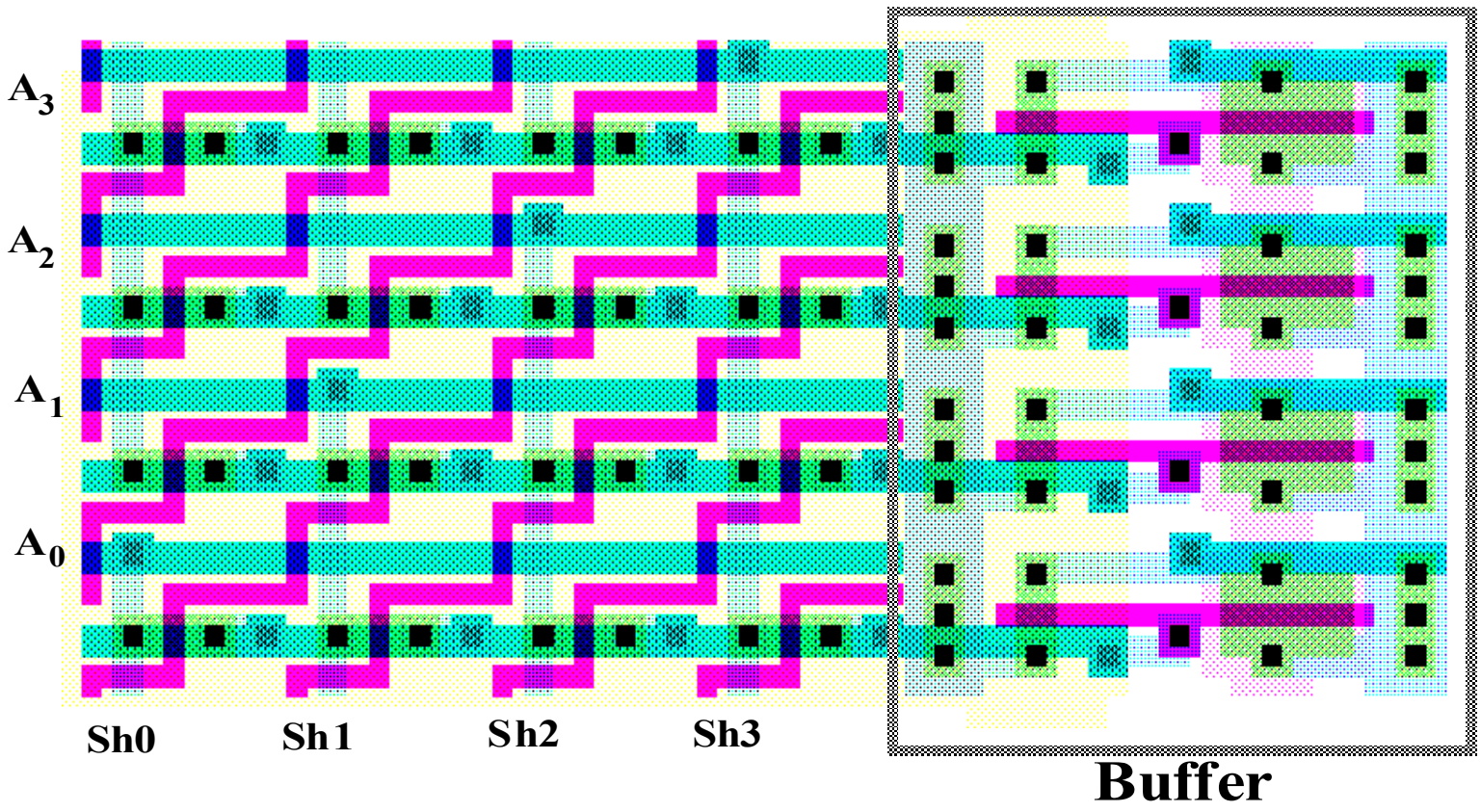


The Barrel Shifter



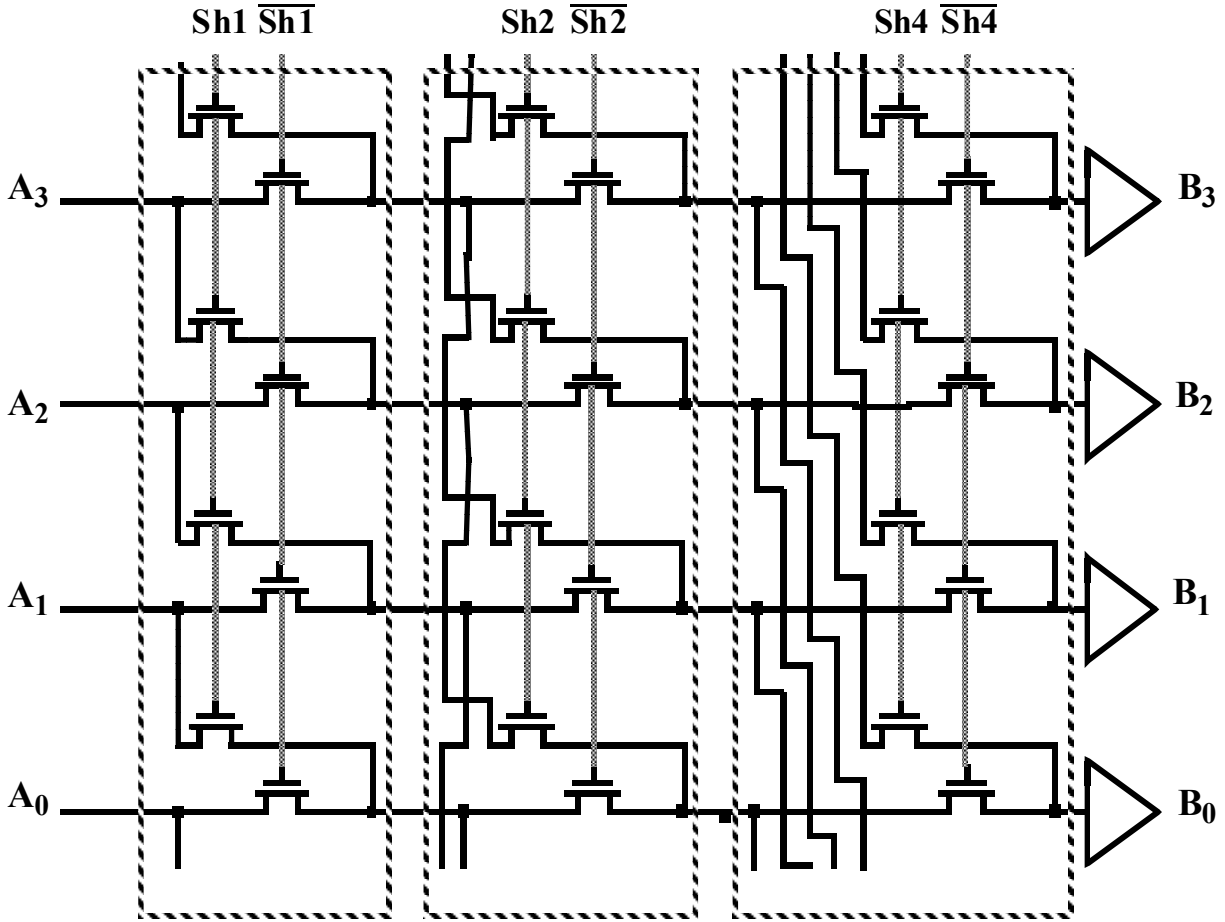
Area Dominated by Wiring

4x4 barrel shifter



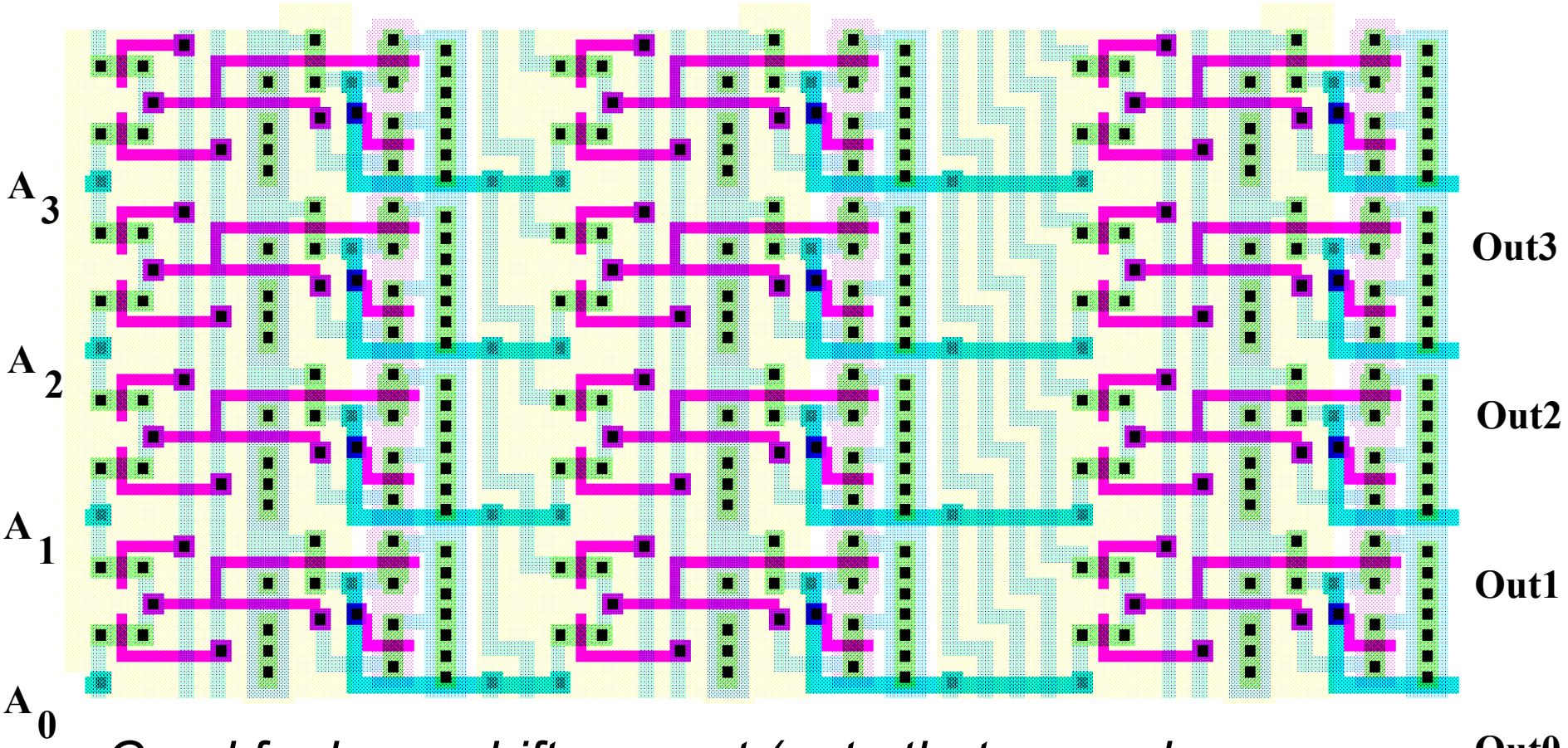
- *Coder/decoder required to set shift bits*
- *Signal pass through one gate independent of shift amount (parasitic capacitance may change the picture)*

Logarithmic Shifter



No separate coder/decoder is required

0-7 bit Logarithmic Shifter



Good for large shift amount (note that cascade pass transistor slow down the gate and generate weak signals, buffers may be needed)

Building Blocks for Digital Architectures

Arithmetic unit

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

(comparator, divider, sin, cos etc)