Digital Integrated Circuits

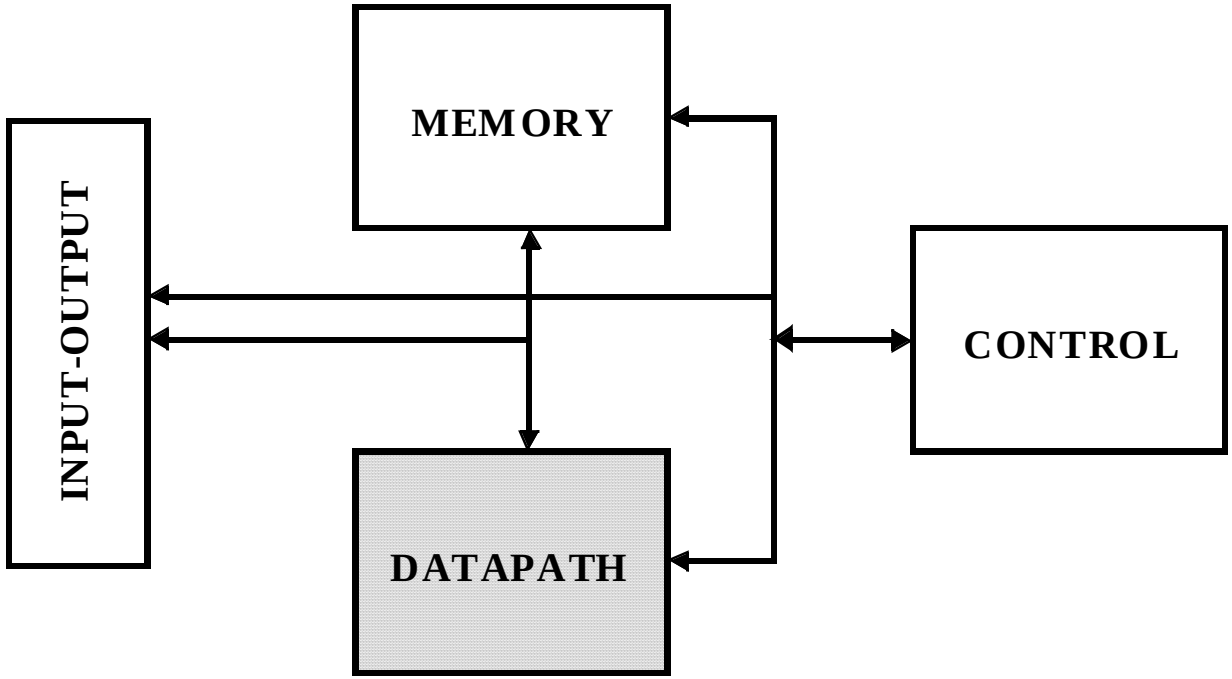
A Design Perspective

Jan M. Rabaey
Anantha Chandrakasan
Borivoje Nikolic

Arithmetic Circuits

January, 2003

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

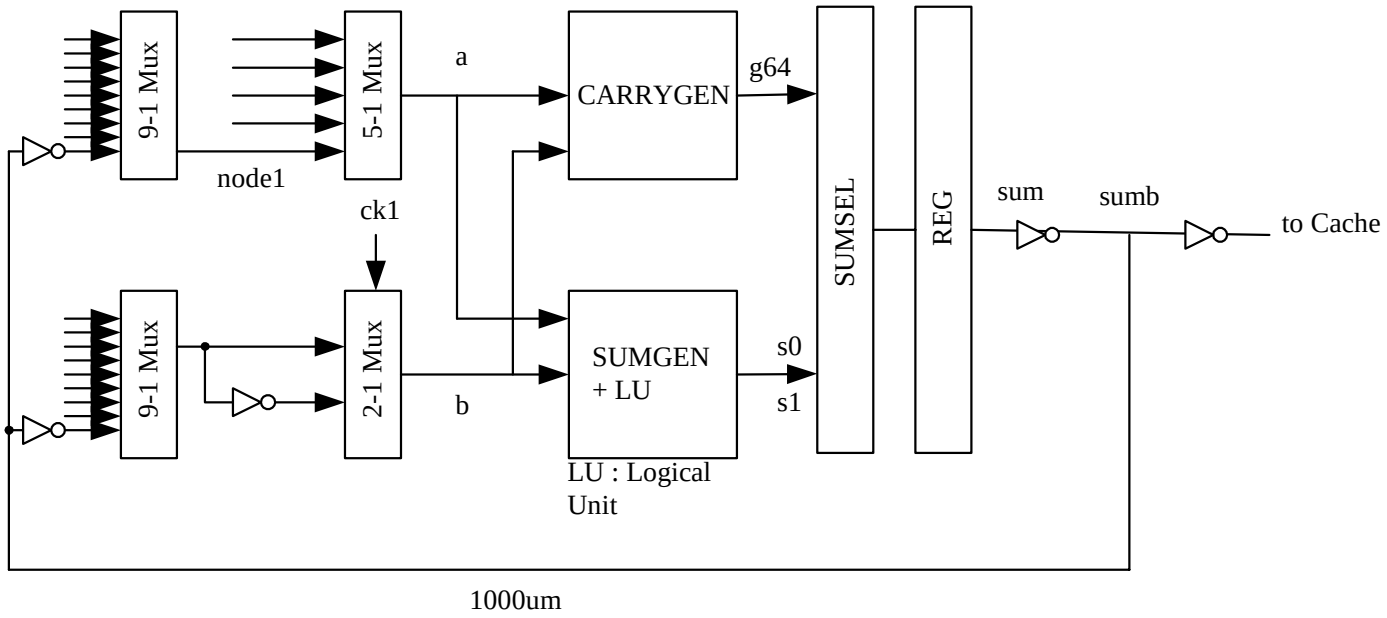
Control

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

Interconnect

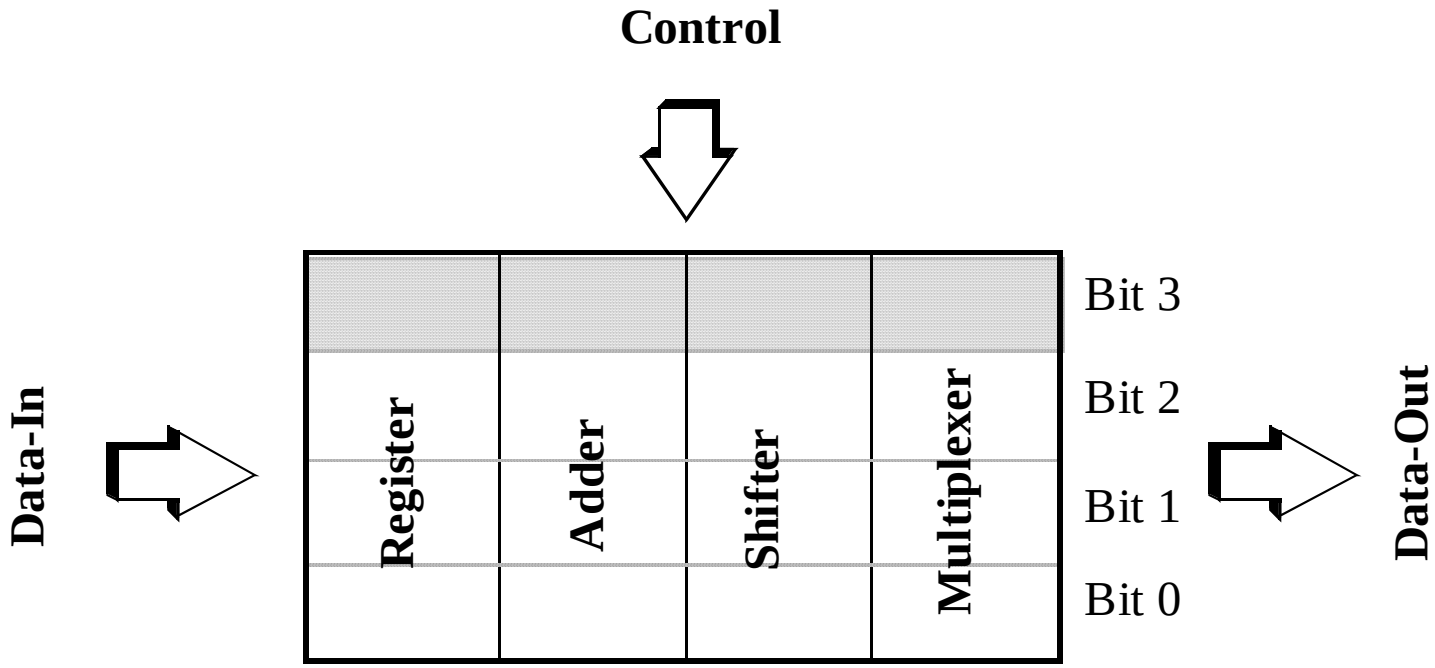
- Switches
- Arbiters
- Bus

An Intel Microprocessor



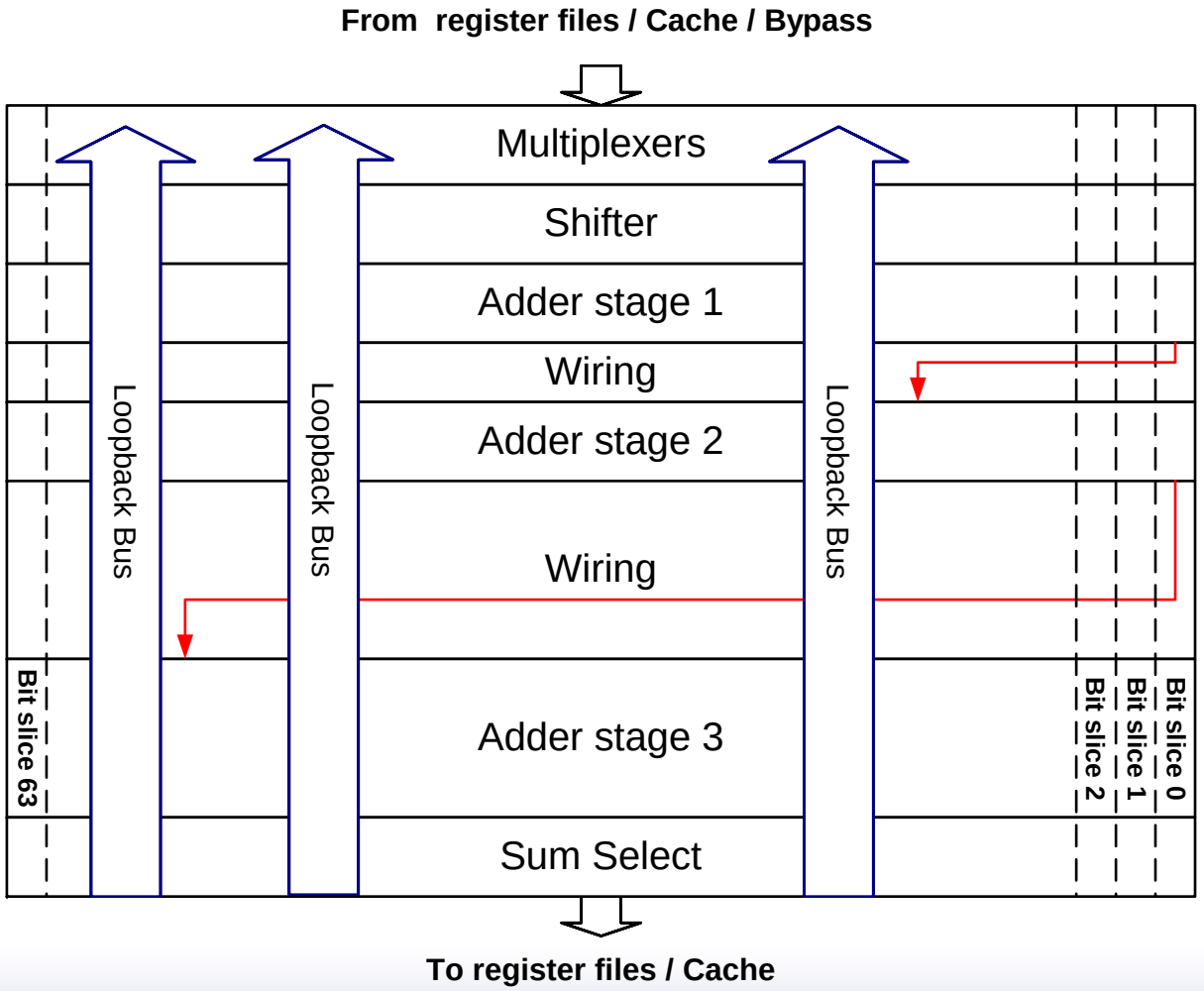
Itanium has 6 integer execution units like this

Bit-Sliced Design

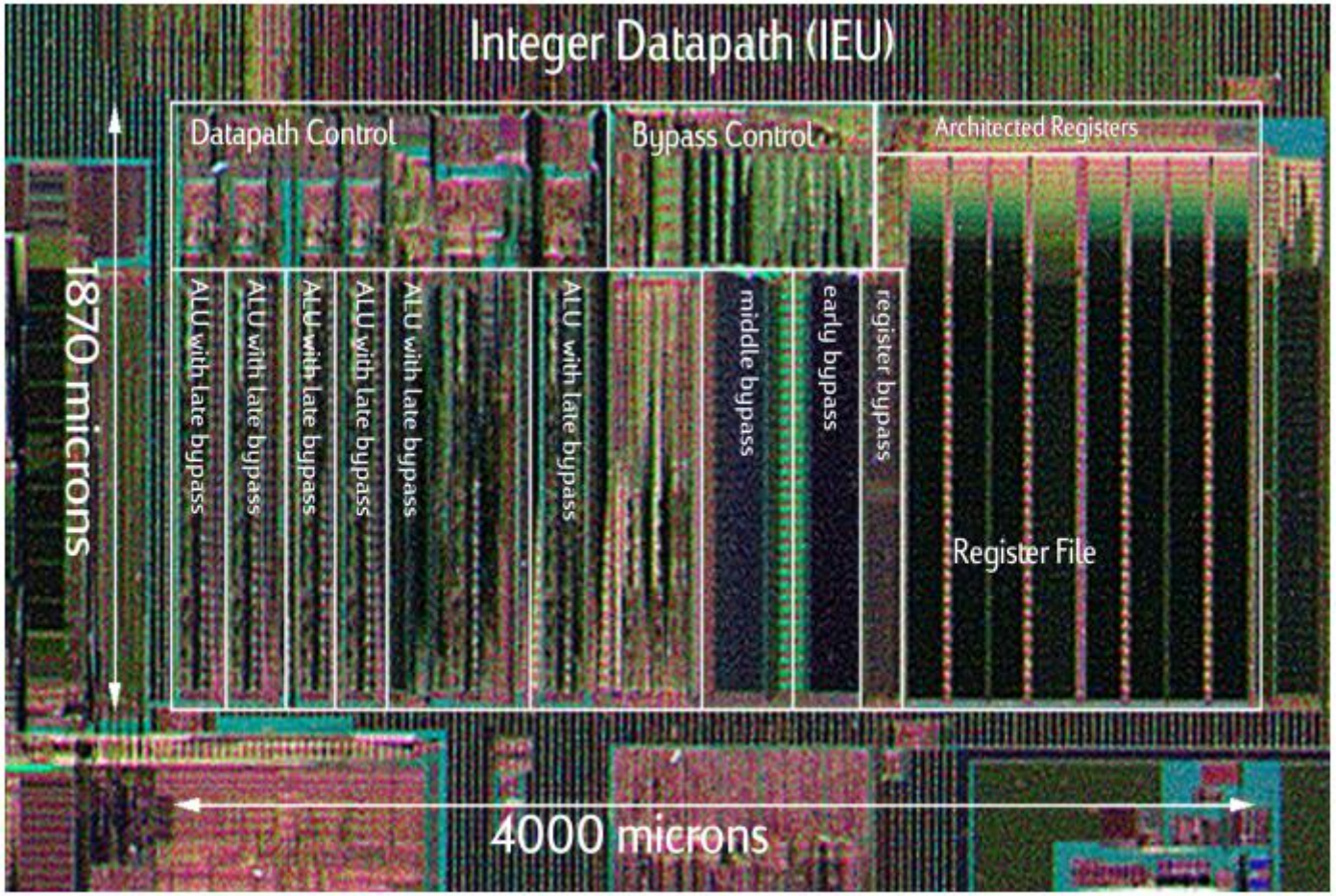


Tile identical processing elements

Bit-Sliced Datapath

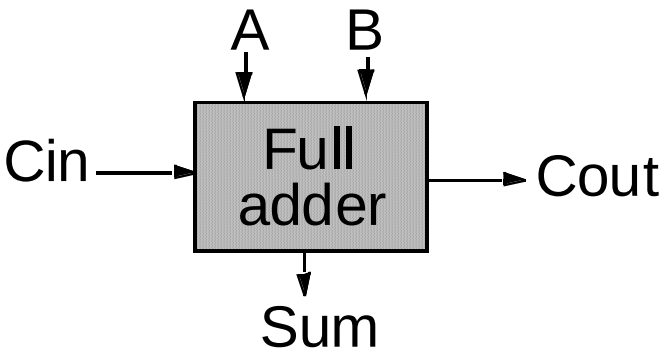


Itanium Integer Datapath



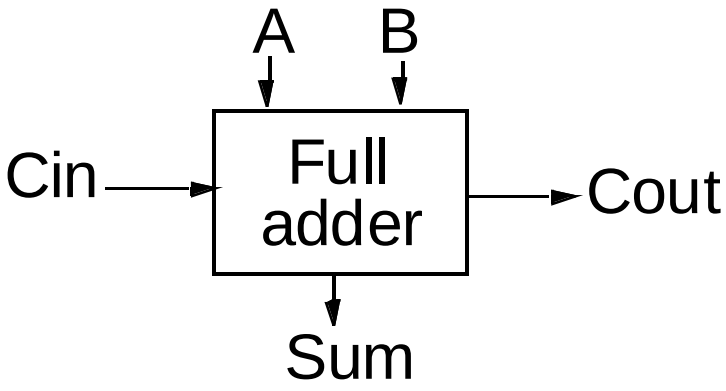
Adders

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

The Binary Adder



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

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

$$C_o = AB + BC_i + AC_i$$

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

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

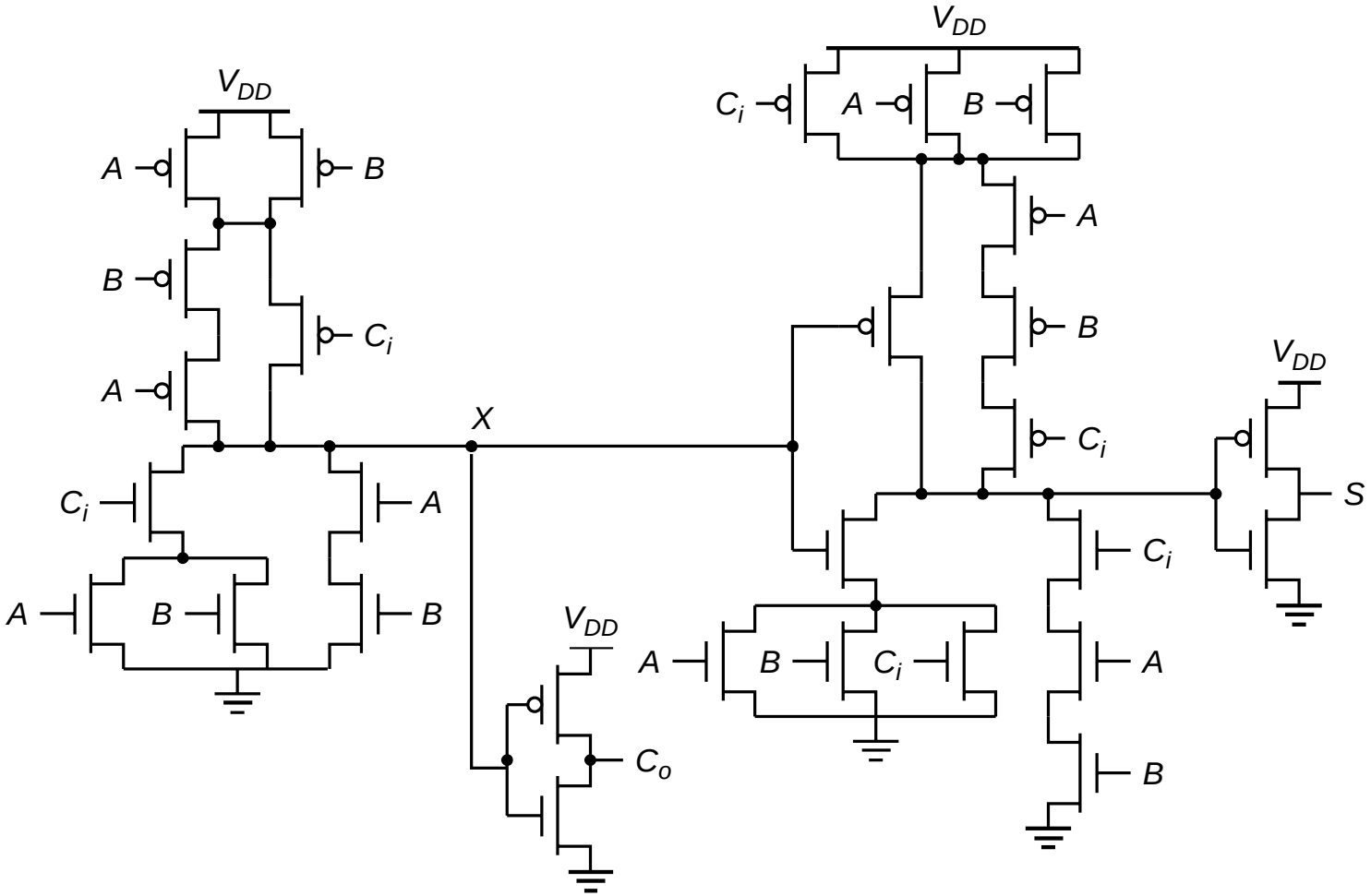
$$S(G,P) = P \oplus C_i$$

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

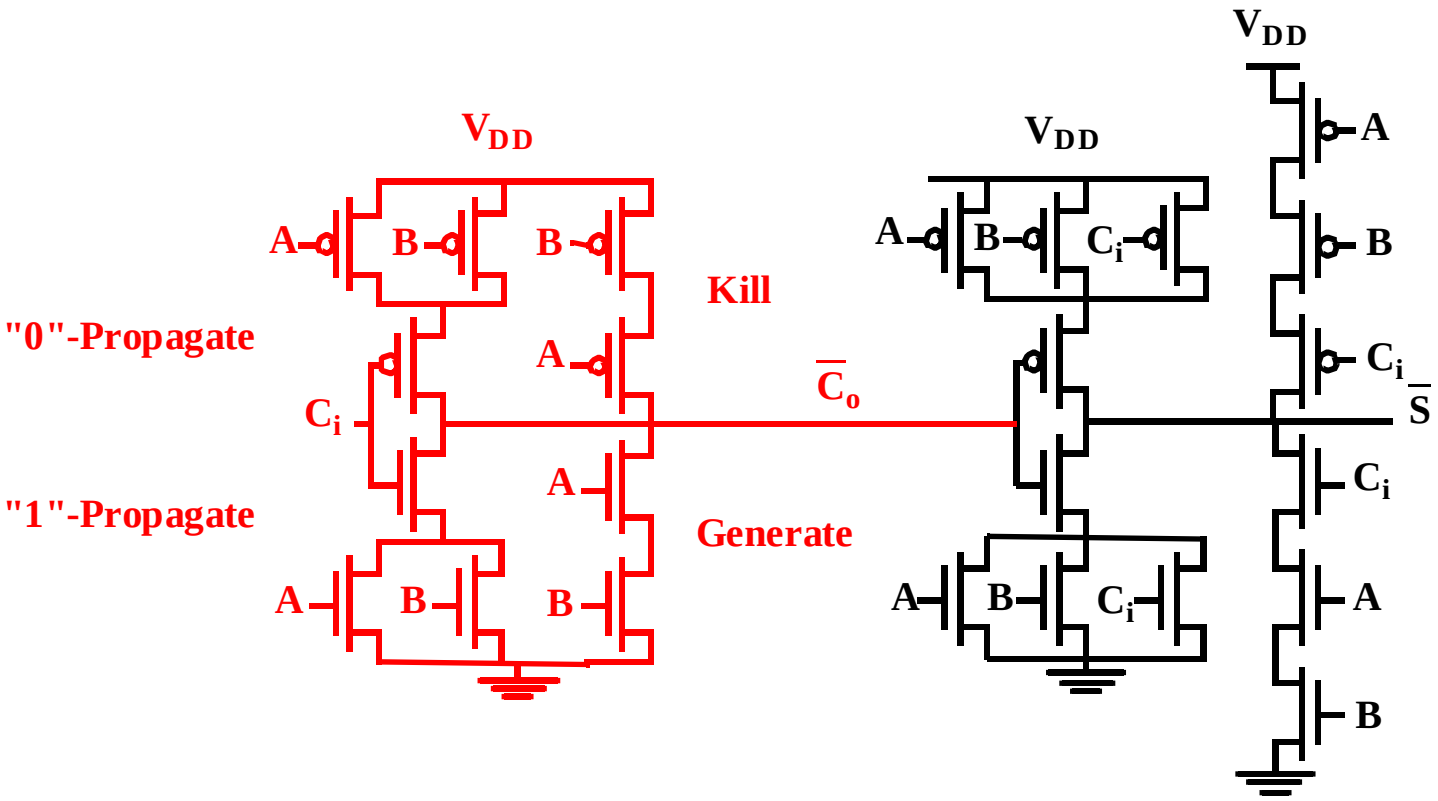
Propagate (P) = A + B

Complimentary Static CMOS Full Adder



28 Transistors

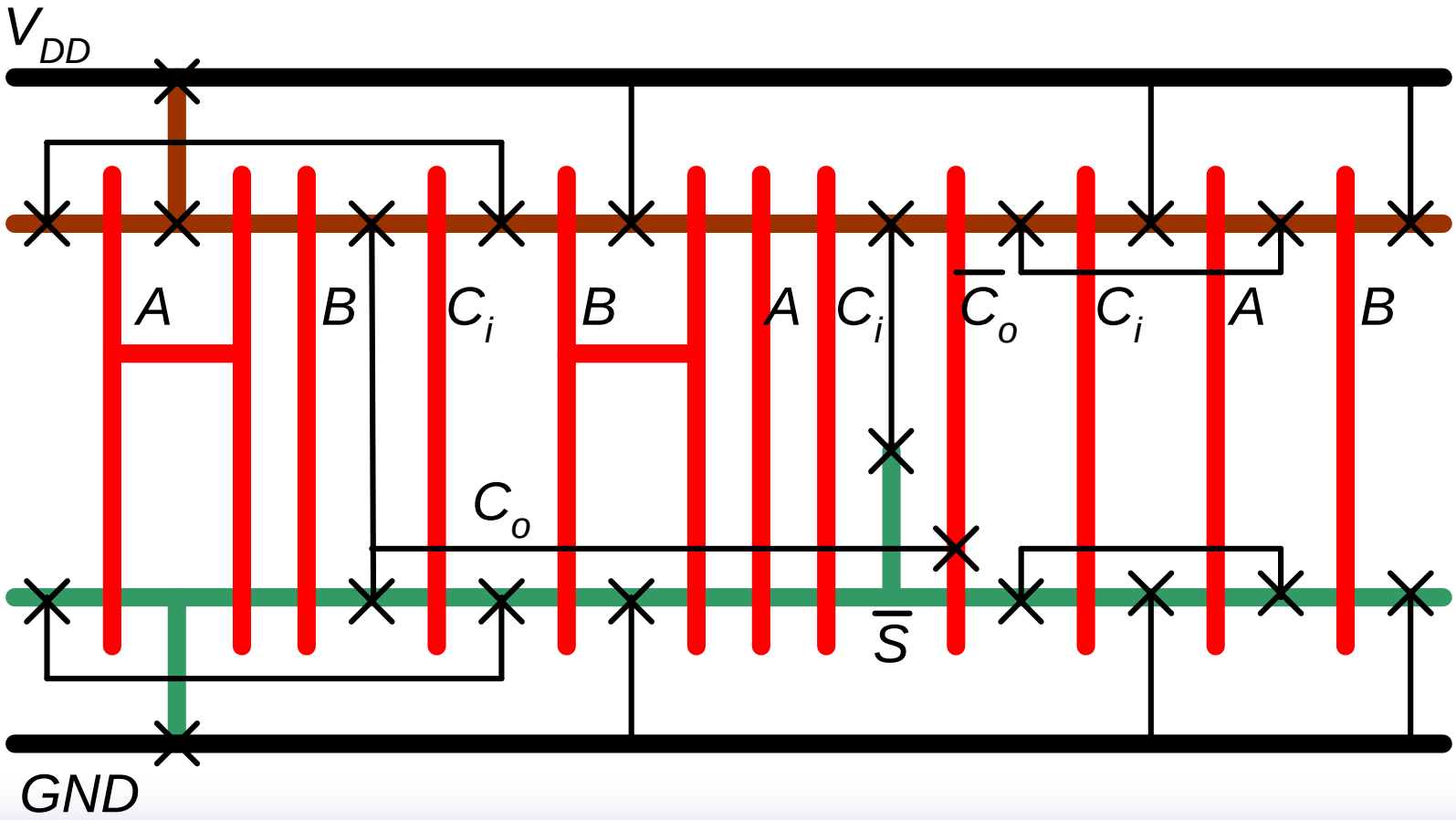
A Better Structure: The Mirror Adder



24 transistors

Mirror Adder

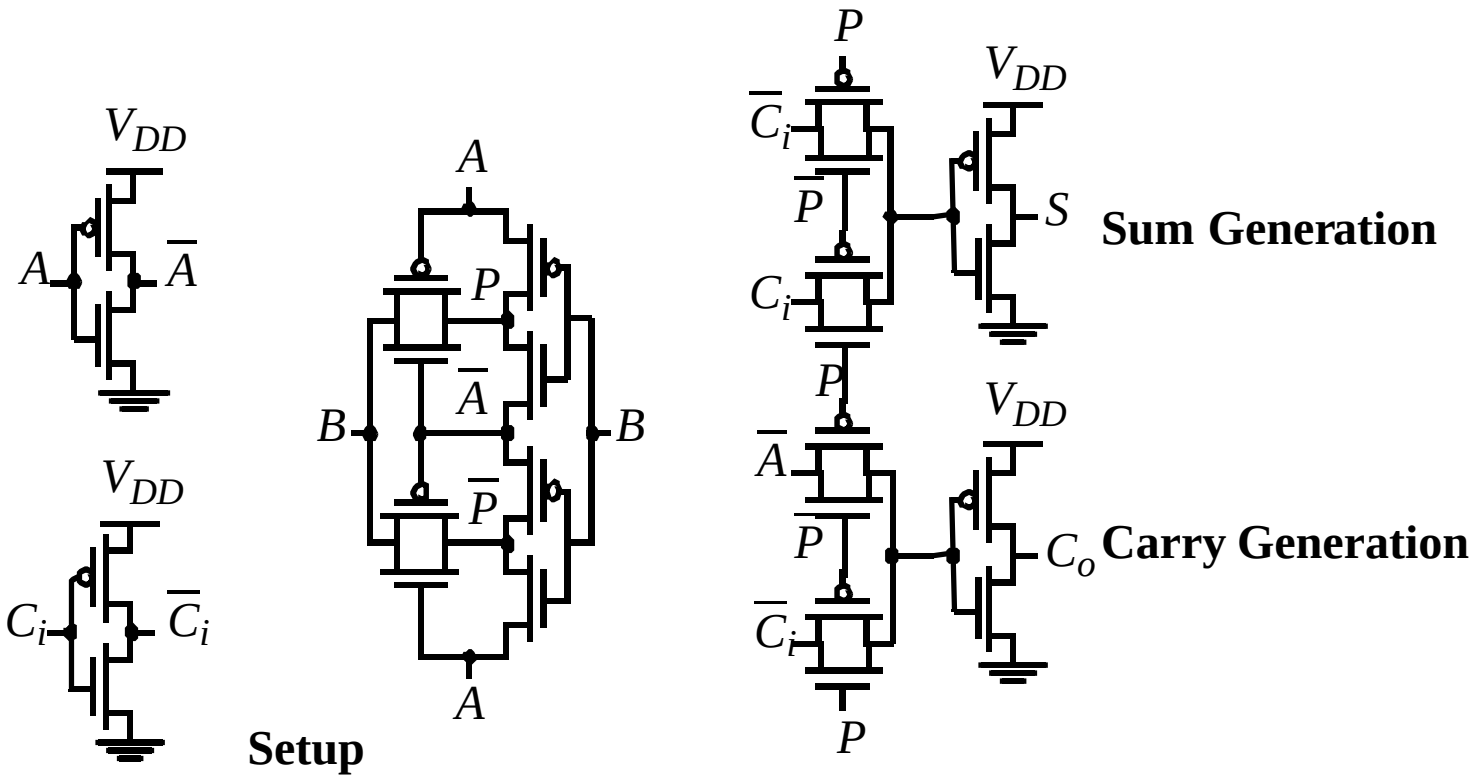
Stick Diagram



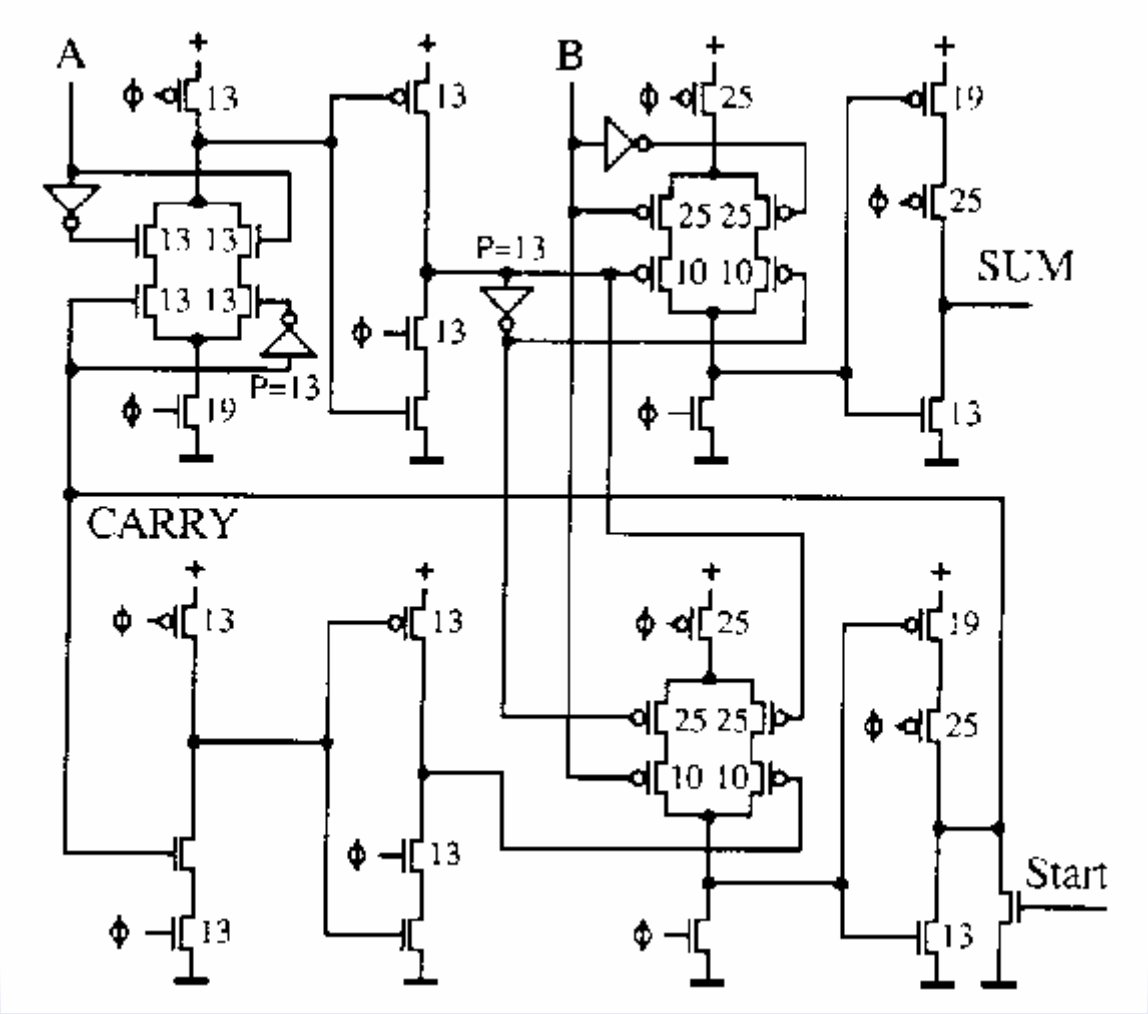
The Mirror Adder

- **The NMOS and PMOS chains are completely symmetrical. 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 transistors connected to C_i are placed closest to the output.**
- **Only the transistors in the carry stage have to be optimized for optimal speed. All transistors in the sum stage can be minimal size.**

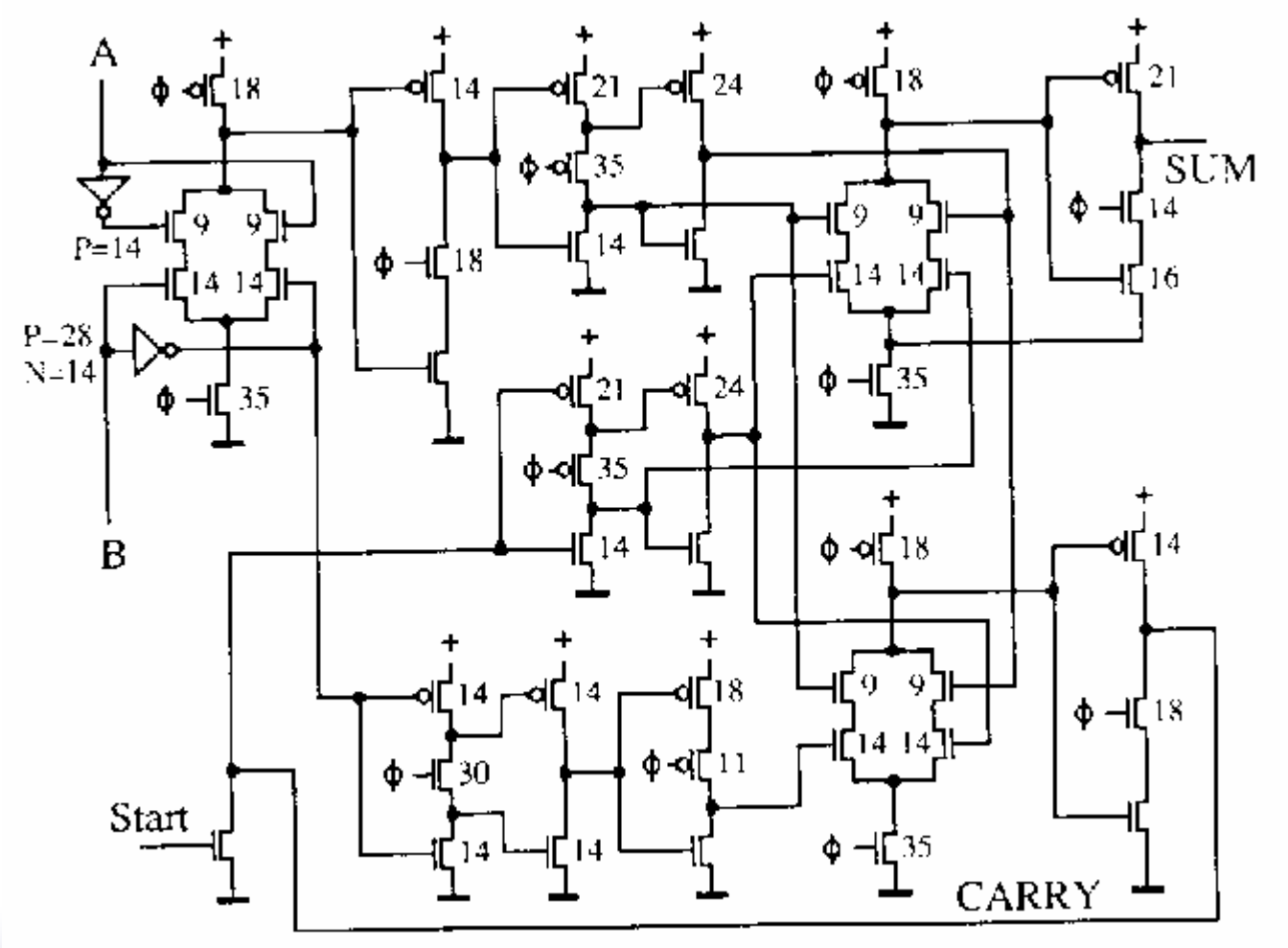
Transmission Gate Full Adder



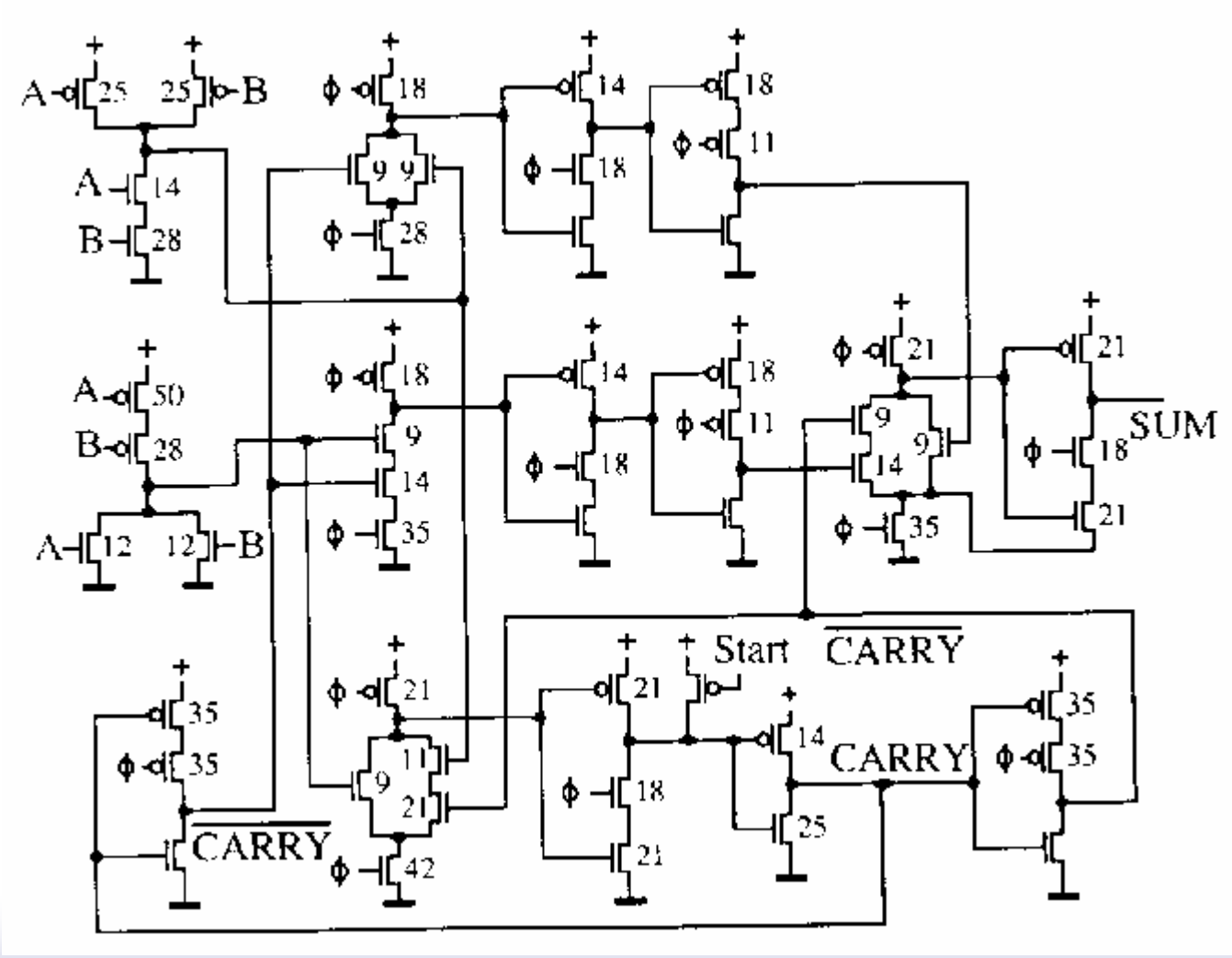
One-phase dynamic CMOS adder



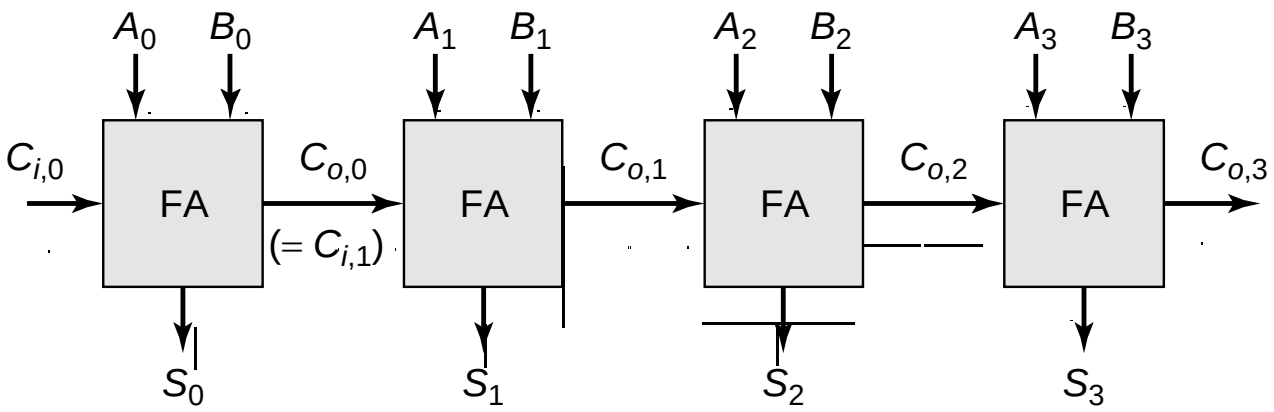
One-phase dynamic CMOS adder



One-phase dynamic CMOS adder



The Ripple-Carry Adder



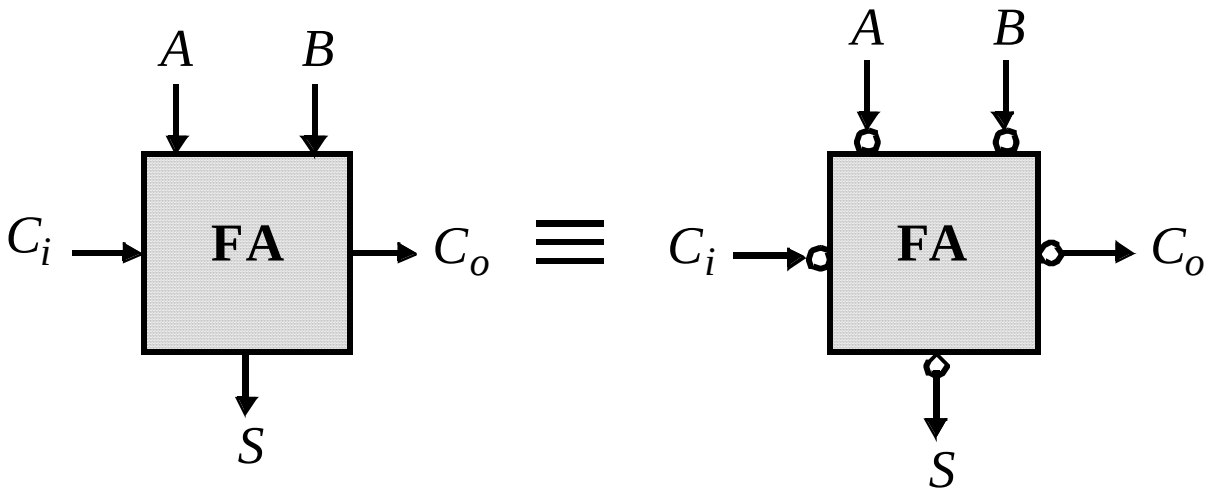
Worst case delay linear with the number of bits

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

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

Goal: Make the fastest possible carry path circuit

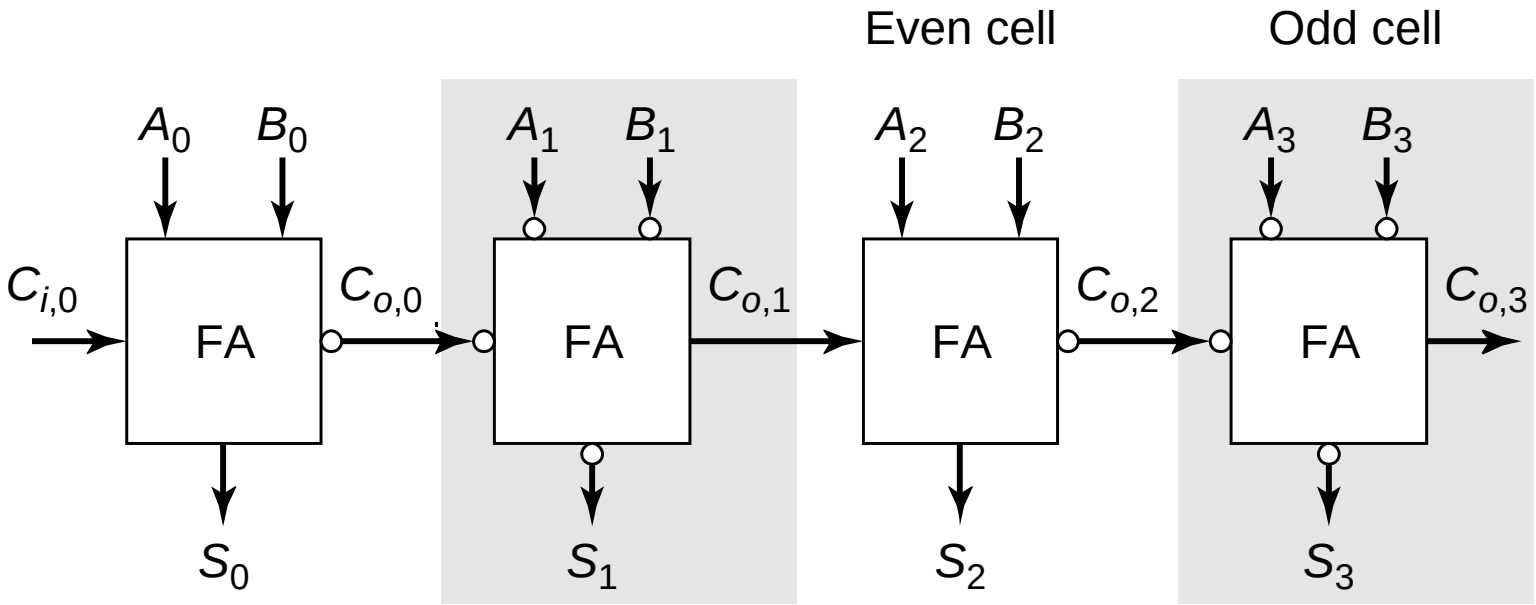
Inversion Property



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

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

Minimize Critical Path by Reducing Inverting Stages



Exploit Inversion Property

Carry Look-Ahead Adders

Carry boolean equations

$$C_i = G_i + P_i C_{i-1} = \overline{K_i + P_i \overline{C}_{i-1}}$$

$$\overline{C}_i = K_i + P_i \overline{C}_{i-1} = \overline{G_i + P_i C_{i-1}}$$

Having defined:

$$G_i = A_i B_i$$

$$P_i = A_i + B_i$$

$$K_i = \overline{A_i + B_i} = \overline{A}_i \overline{B}_i$$

Carry-Lookahead Adders

Therefore:

$$C_1 = G_1 + P_1 C_0$$

$$C_2 = G_2 + P_2 (G_1 + P_1 C_0)$$

$$C_3 = G_3 + P_3 (G_2 + P_2 (G_1 + P_1 C_0))$$

$$C_4 = G_4 + P_4 (G_3 + P_3 (G_2 + P_2 (G_1 + P_1 C_0)))$$

Carry-Lookahead Adders

Two-level carry equations:

$$C_1 = G_1 + P_1 C_0$$

$$C_2 = G_2 + P_2 G_1 + P_2 P_1 C_0$$

$$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0$$

$$C_4 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_0$$

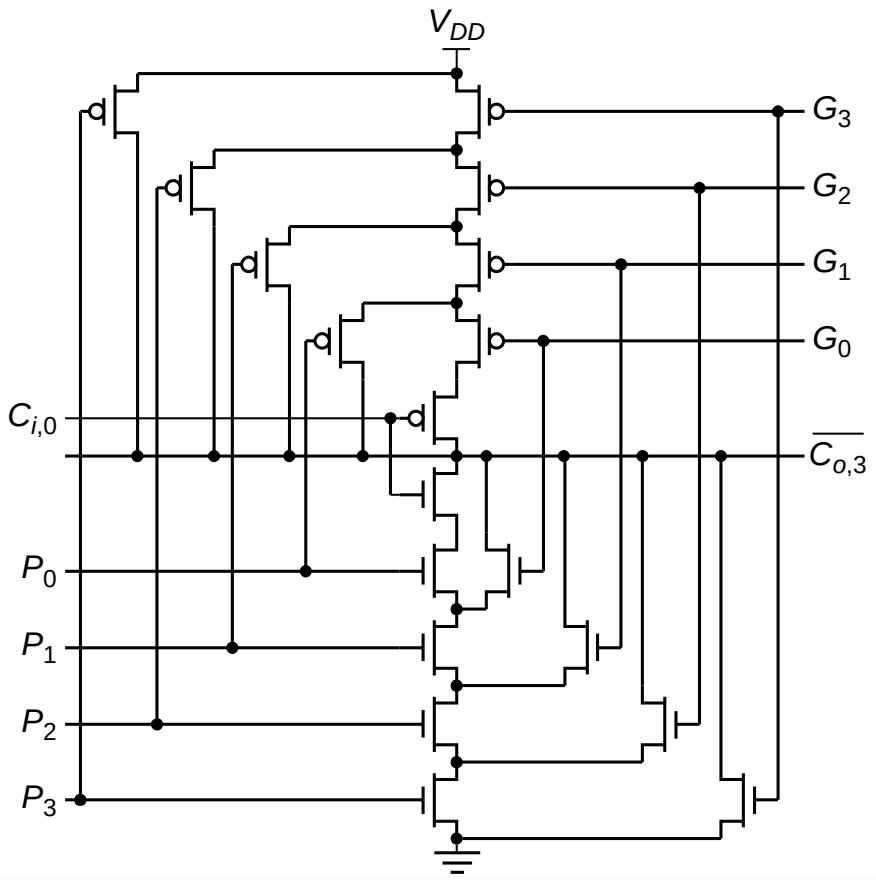
Look-Ahead: Topology

Expanding Lookahead equations:

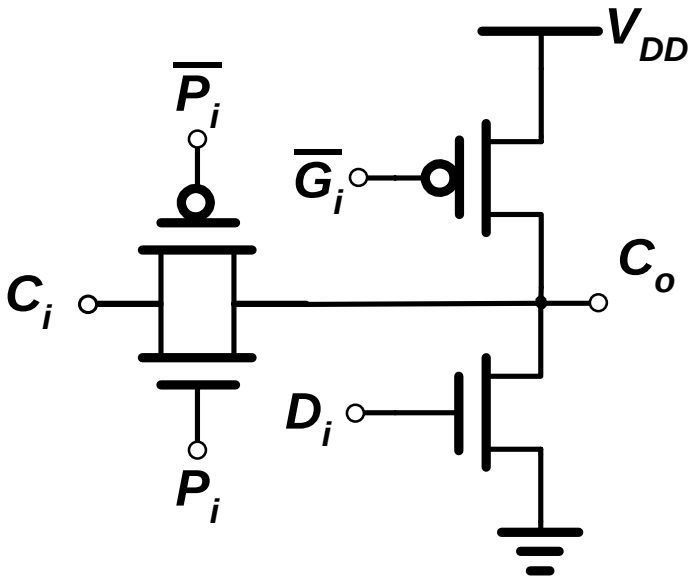
$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}C_{o,k-2})$$

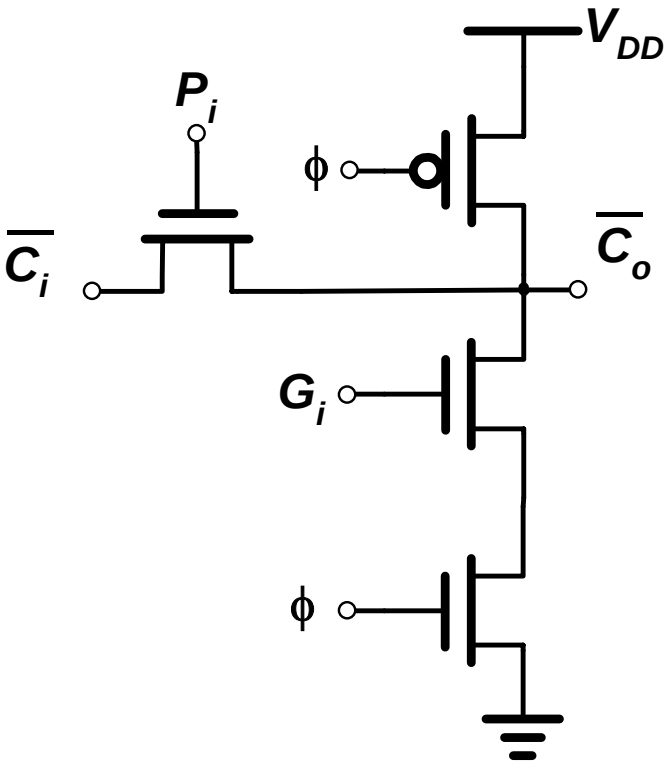
All the way:

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

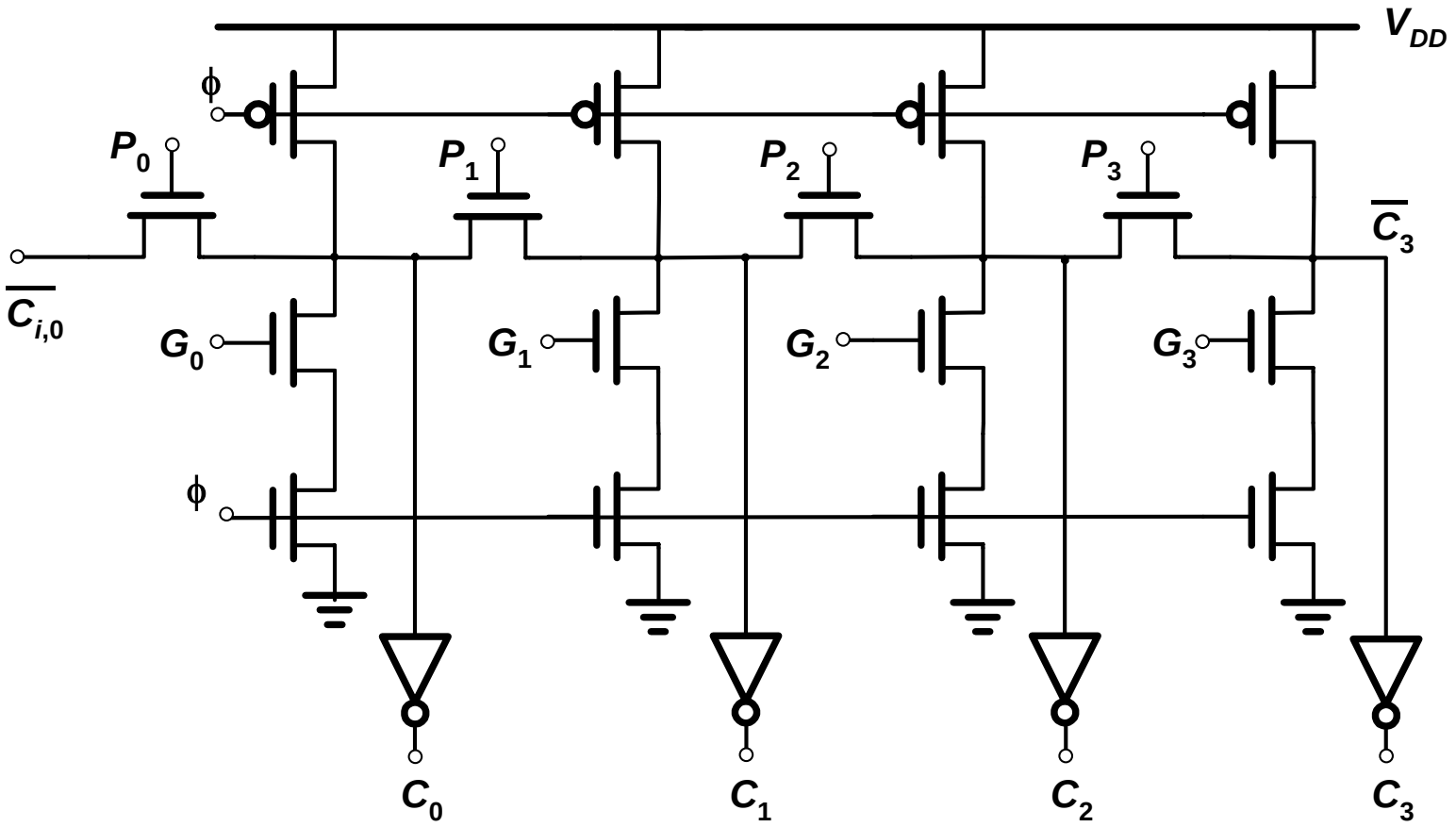


Manchester Carry Chain



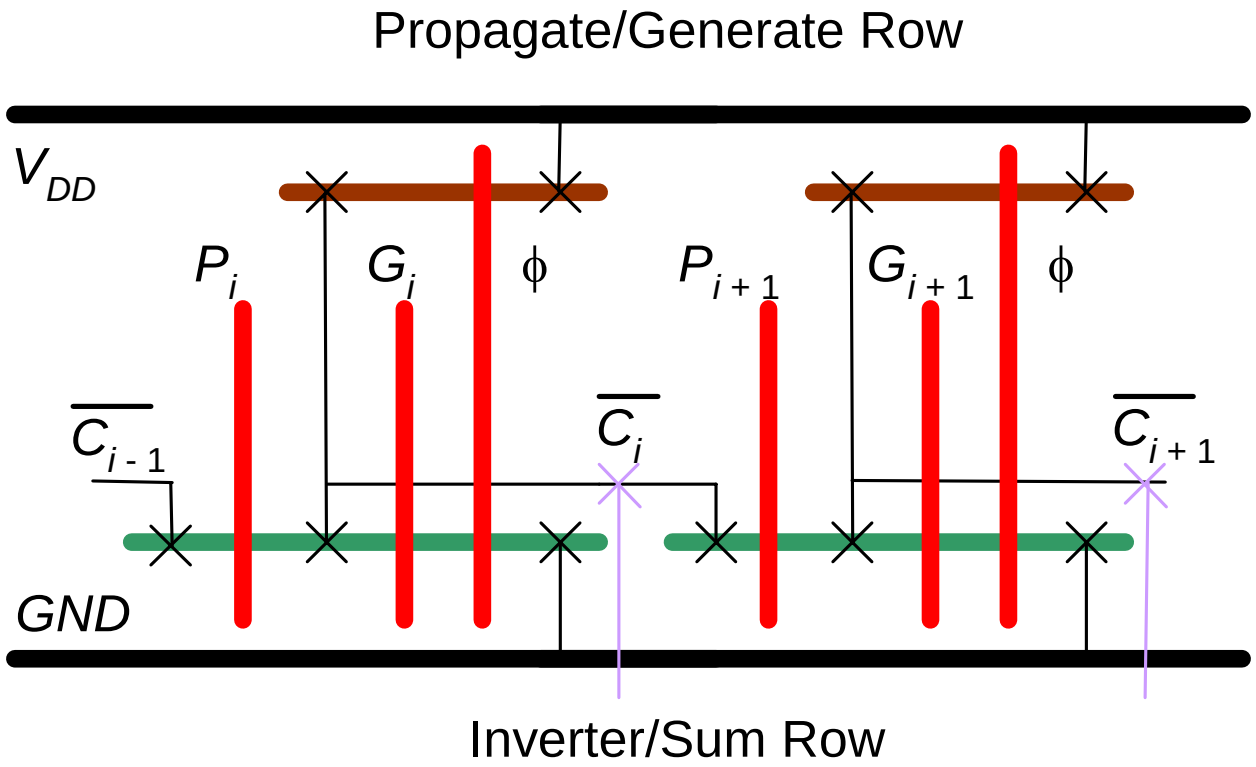


Manchester Carry Chain

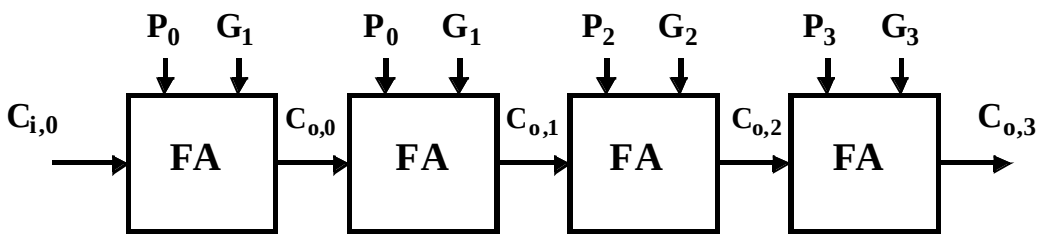


Manchester Carry Chain

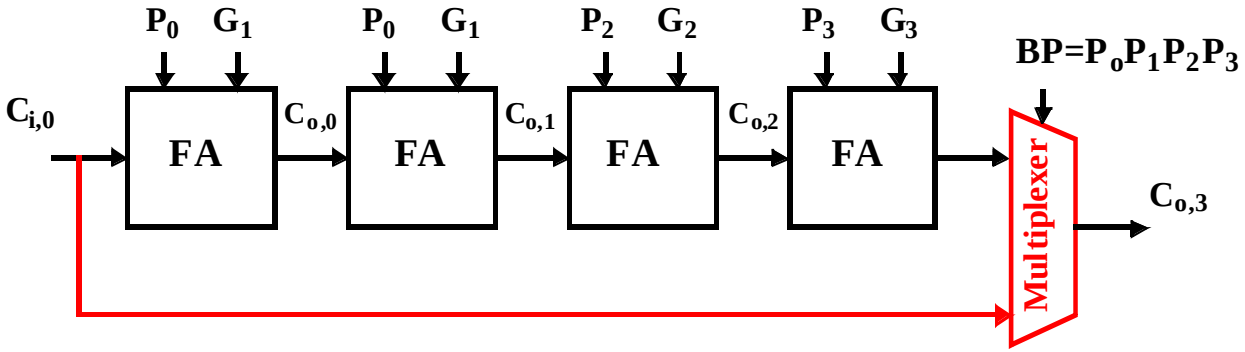
Stick Diagram



Carry-Bypass Adder

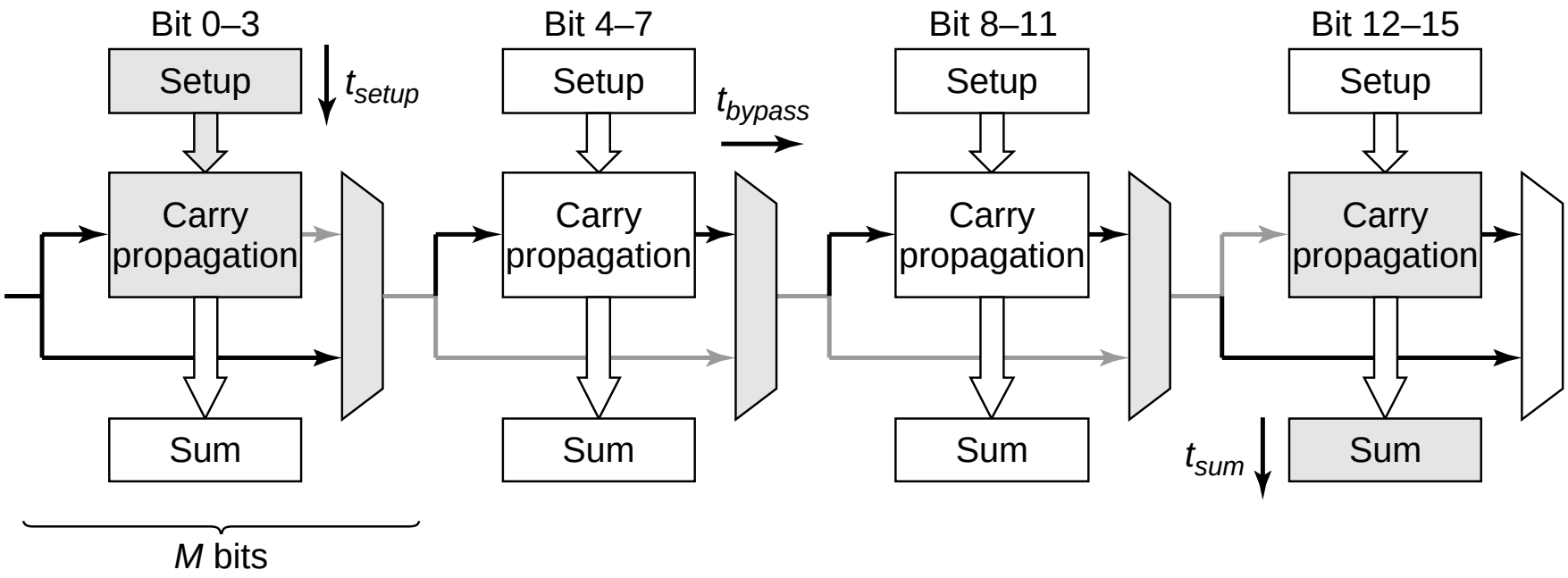


Also called Carry-Skip



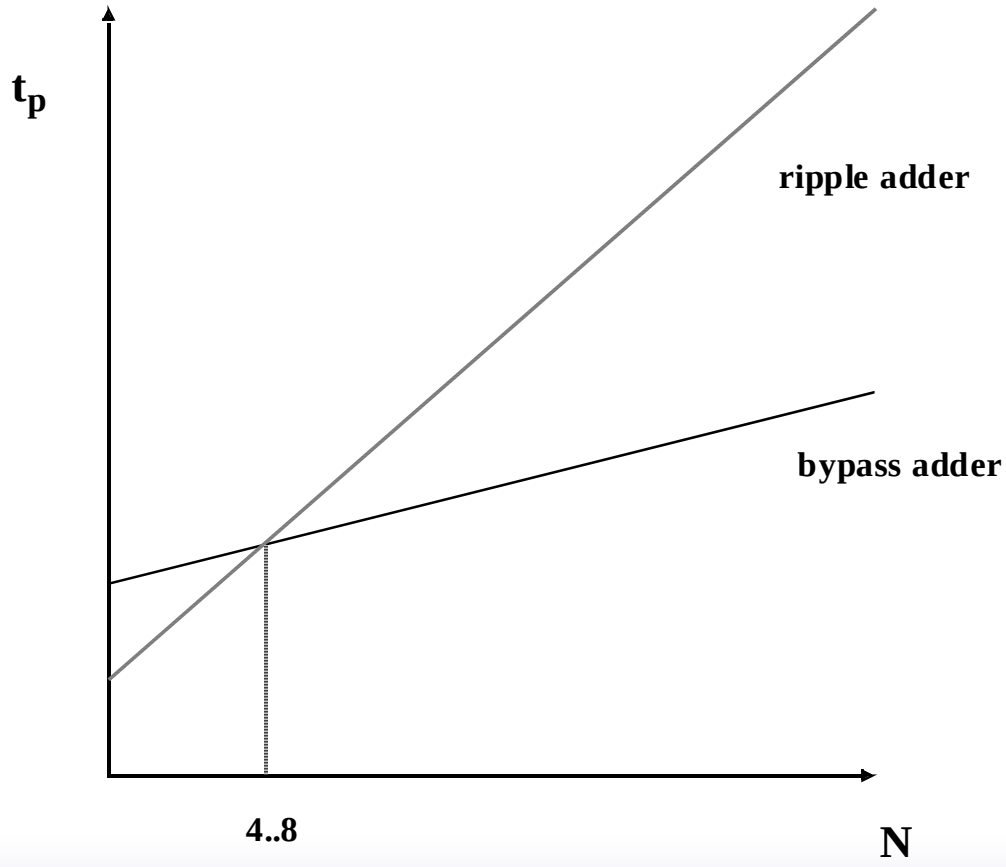
Idea: If (P0 and P1 and P2 and P3 = 1) then $C_{o3} = C_0$, else “kill” or “generate”.

Carry-Bypass Adder (cont.)

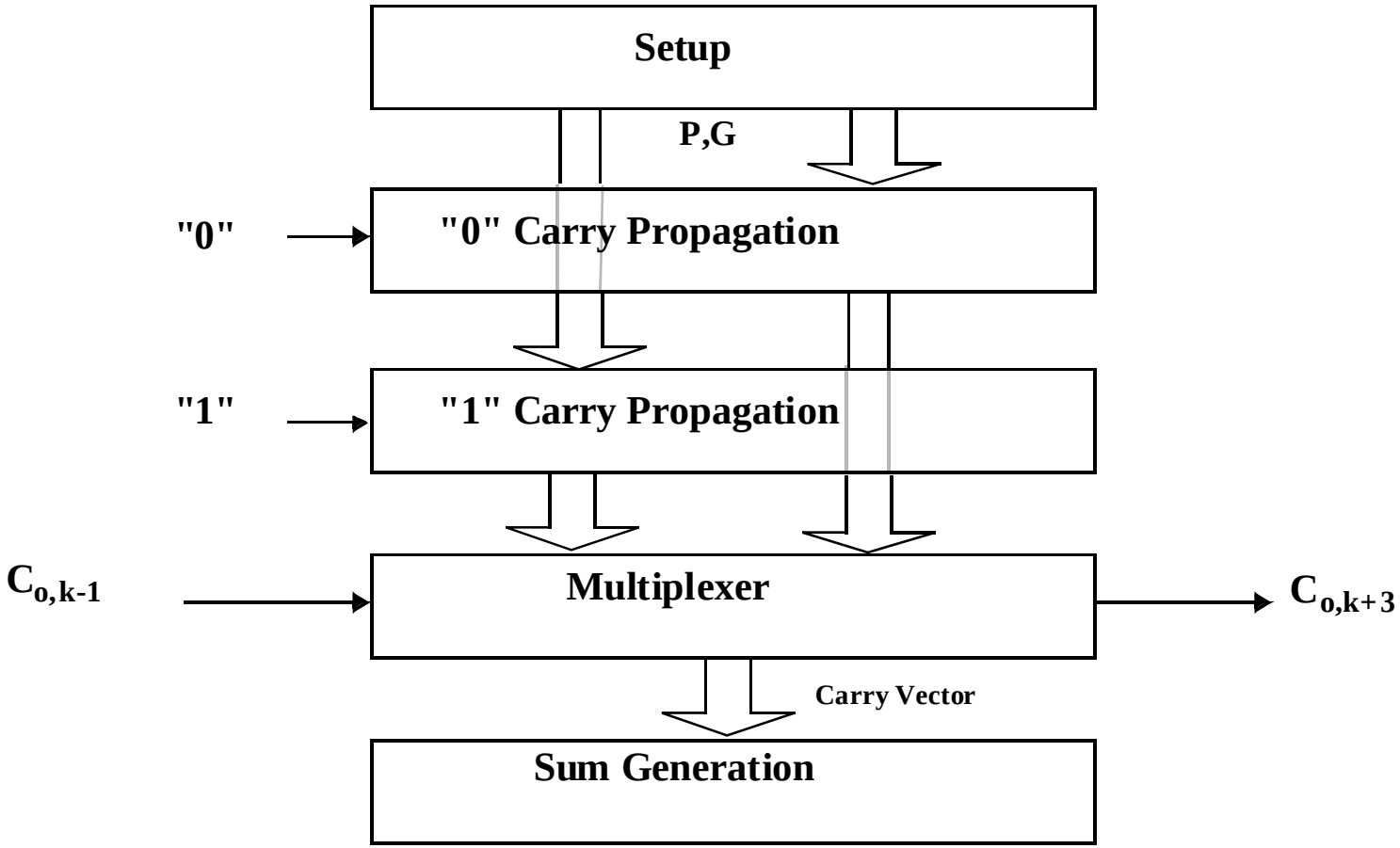


$$t_{adder} = t_{setup} + M_{tcarry} + (N/M - 1)t_{bypass} + (M - 1)t_{carry} + t_{sum}$$

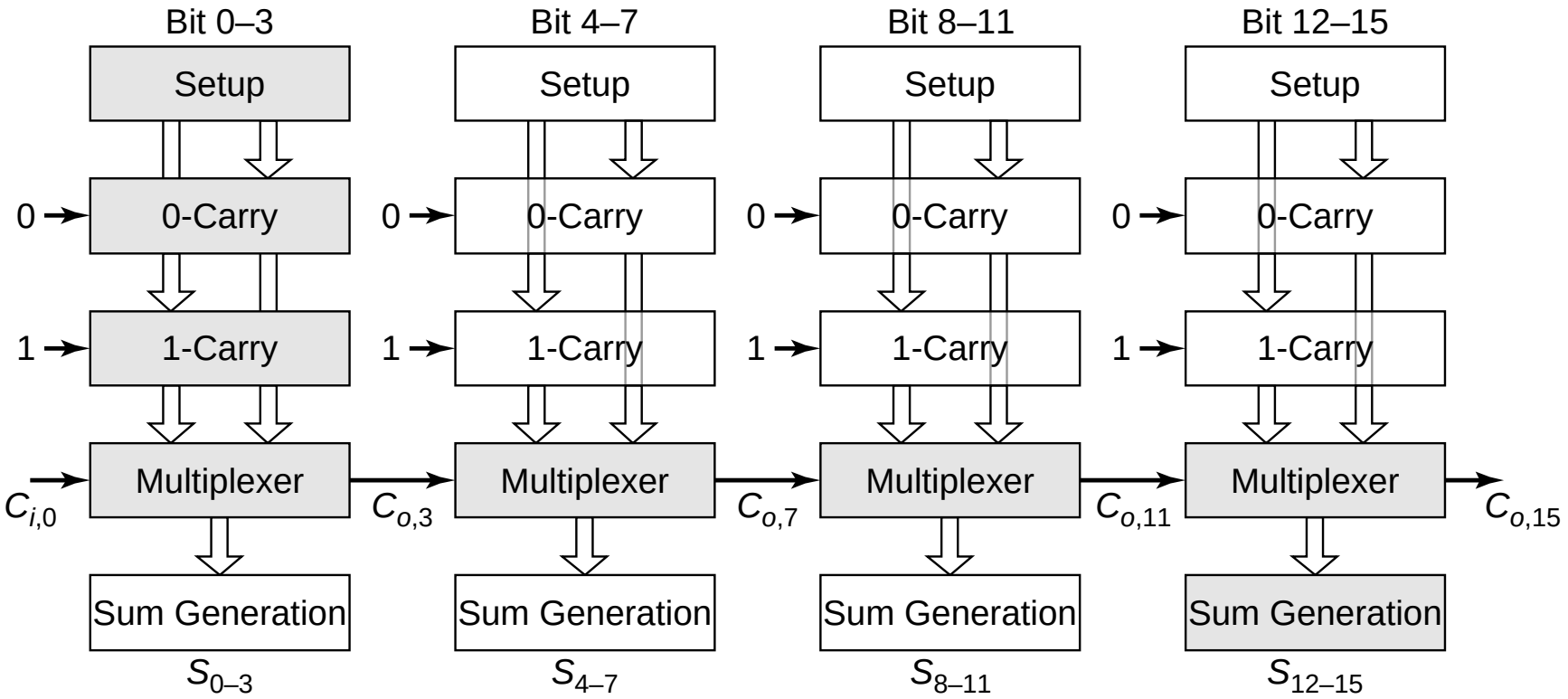
Carry Ripple versus Carry Bypass



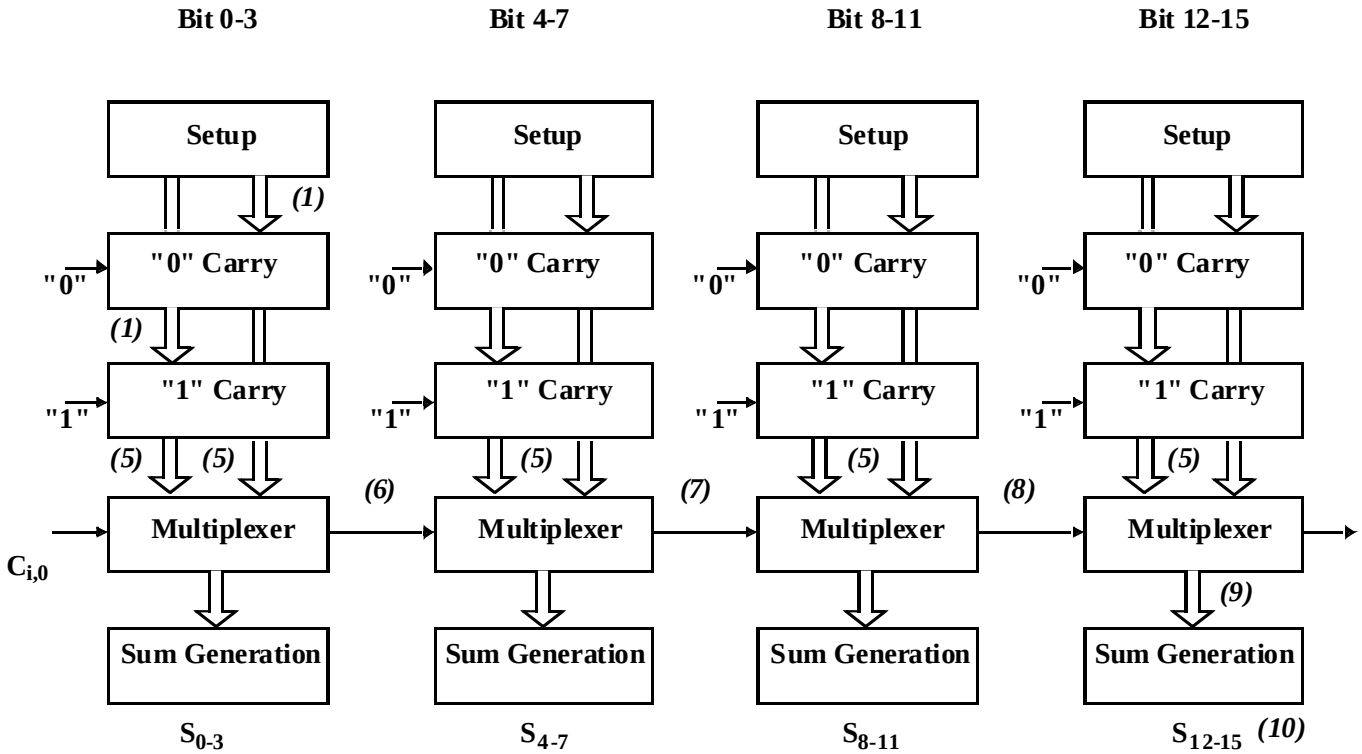
Carry-Select Adder



Carry Select Adder: Critical Path

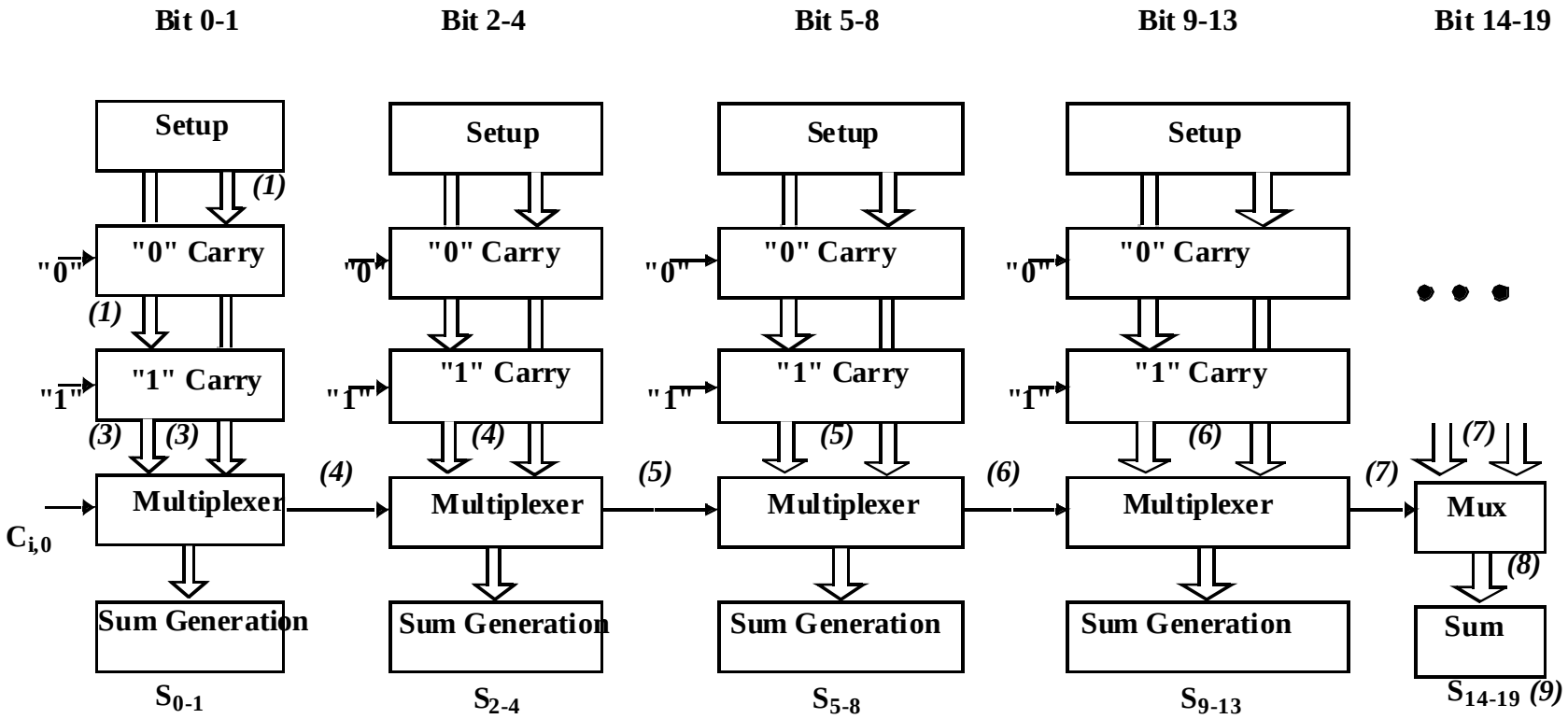


Linear Carry Select



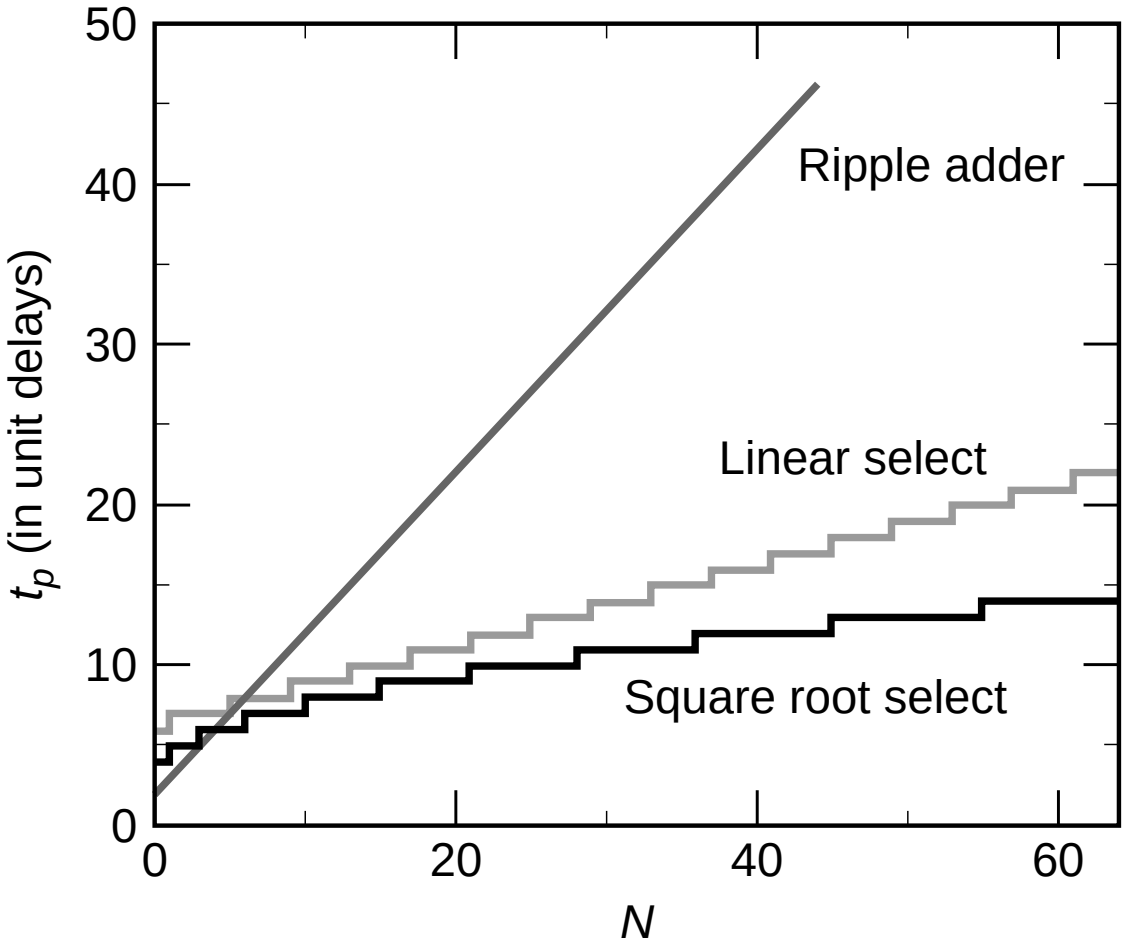
$$t_{add} = t_{setup} + \left(\frac{N}{M}\right) t_{carry} + M t_{mux} + t_{sum}$$

Square Root Carry Select



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

Adder Delays - Comparison



"O" Operator

- Definizione

$$(G_1^2, P_1^2) = (g_2, p_2) \circ (g_1, p_1) = (g_2 + p_2\, g_1,\ p_2\, p_1)$$

$$G_{2:1} = G_1^2 = g_2 + p_2\, g_1$$
$$P_{2:1} = P_1^2 = p_2\, p_1$$

Properties of the "O" operator

Proprietà associativa

$$((g_3, p_3) \circ (g_2, p_2)) \circ (g_1, p_1) = (g_3, p_3) \circ ((g_2, p_2) \circ (g_1, p_1))$$

Dimostrazione

$$((g_3, p_3) \circ (g_2, p_2)) \circ (g_1, p_1) = (g_3 + p_3\, g_2, p_3\, p_2) \circ (g_1, p_1) =$$
$$(g_3 + p_3\, g_2 + p_3\, p_2\, g_1, p_3\, p_2\, p_1)$$
$$(g_3, p_3) \circ ((g_2, p_2) \circ (g_1, p_1)) = (g_3, p_3) \circ (g_2 + p_2\, g_1, p_2\, p_1) =$$
$$(g_3 + p_3(g_2 + p_2\, g_1), p_3\, p_2\, p_1) = (g_3 + p_3\, g_2 + p_3\, p_2\, g_1, p_3\, p_2\, p_1)$$

Properties of the "O" operator

Idempotenza

$$(g,p) \circ (g,p) = (g,p)$$

Elemento neutro: (0,1)

$$(g,p) \circ (0,1) = (g,p)$$

N.B.: Non gode della proprietà commutativa

$$(g_2,p_2) \circ (g_1,p_1) \neq (g_1,p_1) \circ (g_2,p_2)$$

Group Generate and Propagate

Definizione:

$$(G_0^k, P_0^k) = (g_0, p_0) \text{ per } k = 0$$

$$(G_0^k, P_0^k) = (g_k, p_k) \circ (g_{k-1}, p_{k-1}) \circ \cdots \circ (g_0, p_0) \text{ per } k > 0$$

Inoltre, se $i < k$:

$$(G_i^k, P_i^k) = (g_k, p_k) \circ (g_{k-1}, p_{k-1}) \circ \cdots \circ (g_i, p_i)$$

Group Generate and Propagate

Teorema:

Se $C_{in} = 0$, allora $G_0^k = C_{out,k}$

Dimostrazione per induzione:

Posto $k = 0$, $C_{in} = 0$,

$$G_0^0 = g_0$$

$$C_{out,0} = g_0 + p_0 C_{in} = g_0$$

$$G_0^0 = C_{out,0}$$

Group Generate and Propagate

Posto $k = 1$, $C_{in,1} = C_{out,0} = g_0$,

$$G_0^1 = g_1 + p_1\, g_0$$

$$C_{out,1} = g_1 + p_1\, C_{in,1} = g_1 + p_1\, g_0$$

$$G_0^1 = C_{out,1}$$

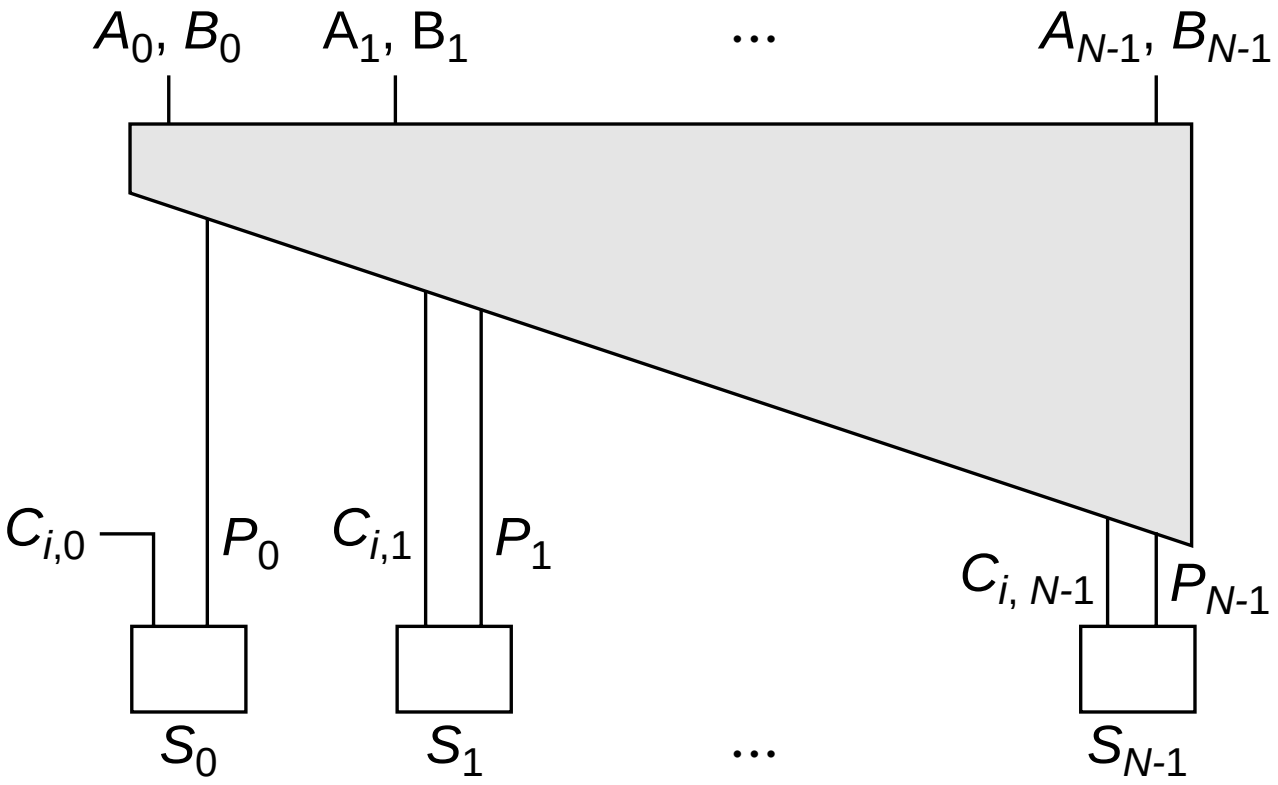
Posto $k > 1$, $C_{in,k} = C_{out,k-1} = G_0^{k-1}$

$$(G_0^k, P_0^k) = (g_k, p_k) \circ (G_0^{k-1}, P_0^{k-1}) = (g_k + p_k\, G_0^{k-1}, p_k\, P_0^{k-1})$$

$$C_{out,k} = g_k + p_k\, C_{in,k} = g_k + p_k\, G_0^{k-1}$$

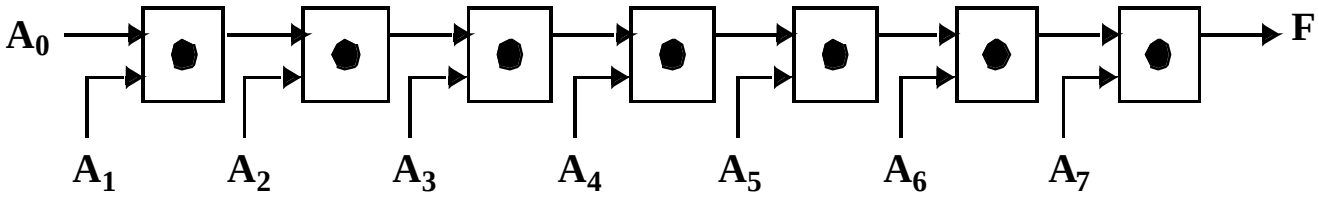
$$G_0^k = C_{out,k}$$

Look-Ahead - Basic Idea

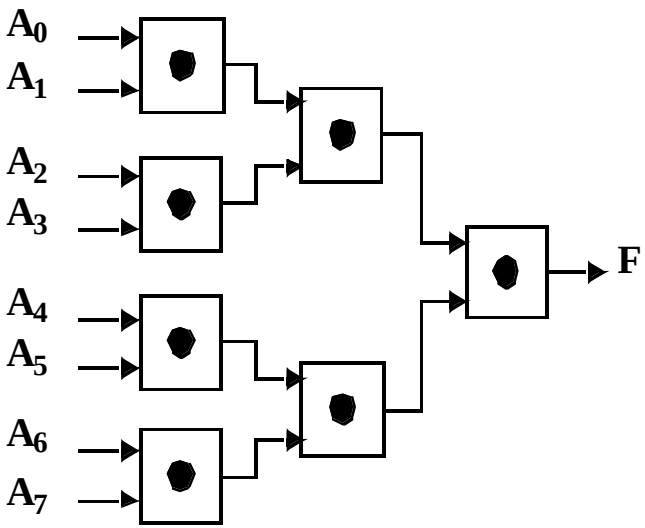


$$C_{o,k} = f(A_k, B_k, C_{o,k-1}) = G_k + P_k C_{o,k-1}$$

Logarithmic Look-Ahead Adder

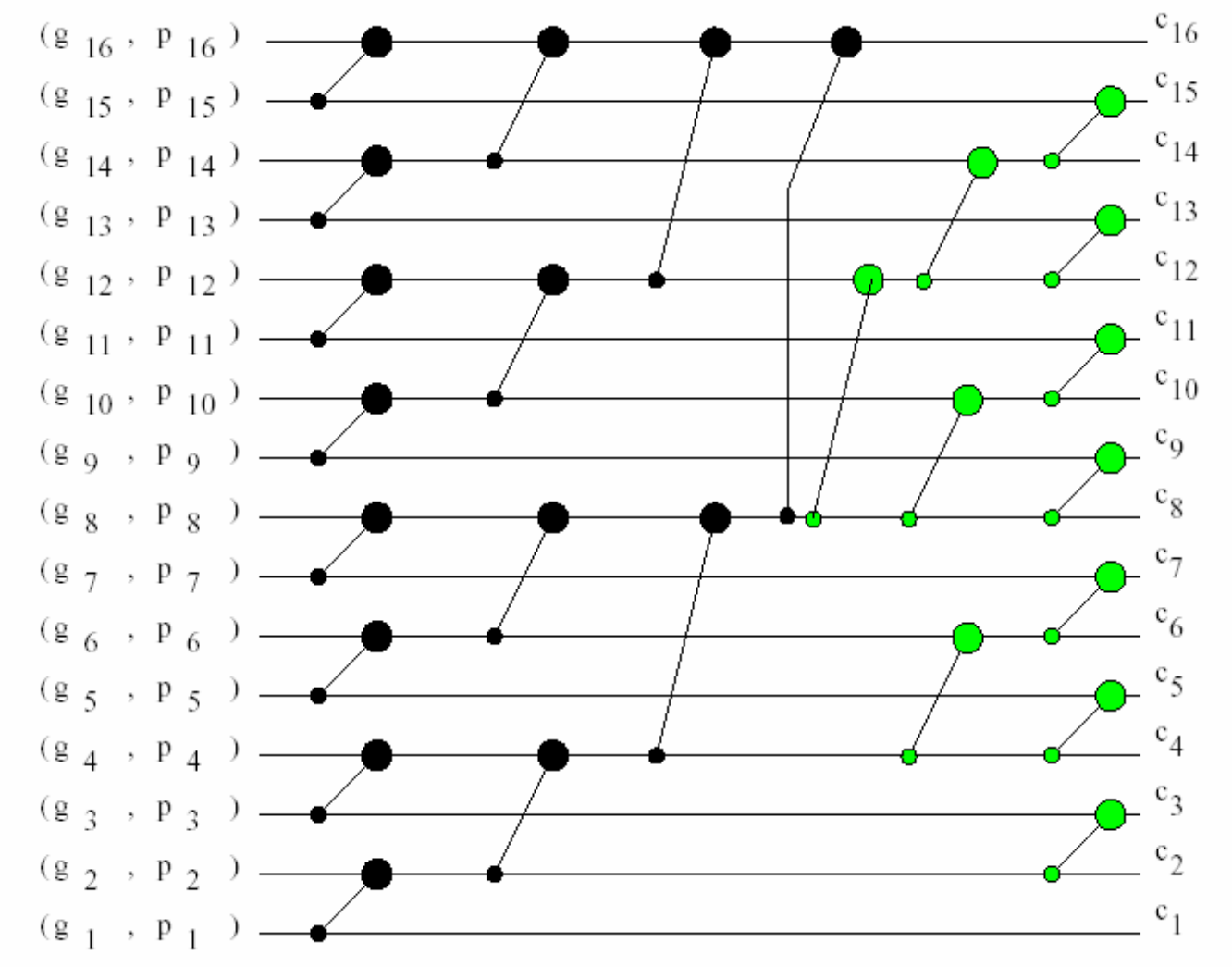


$t_p \sim N$

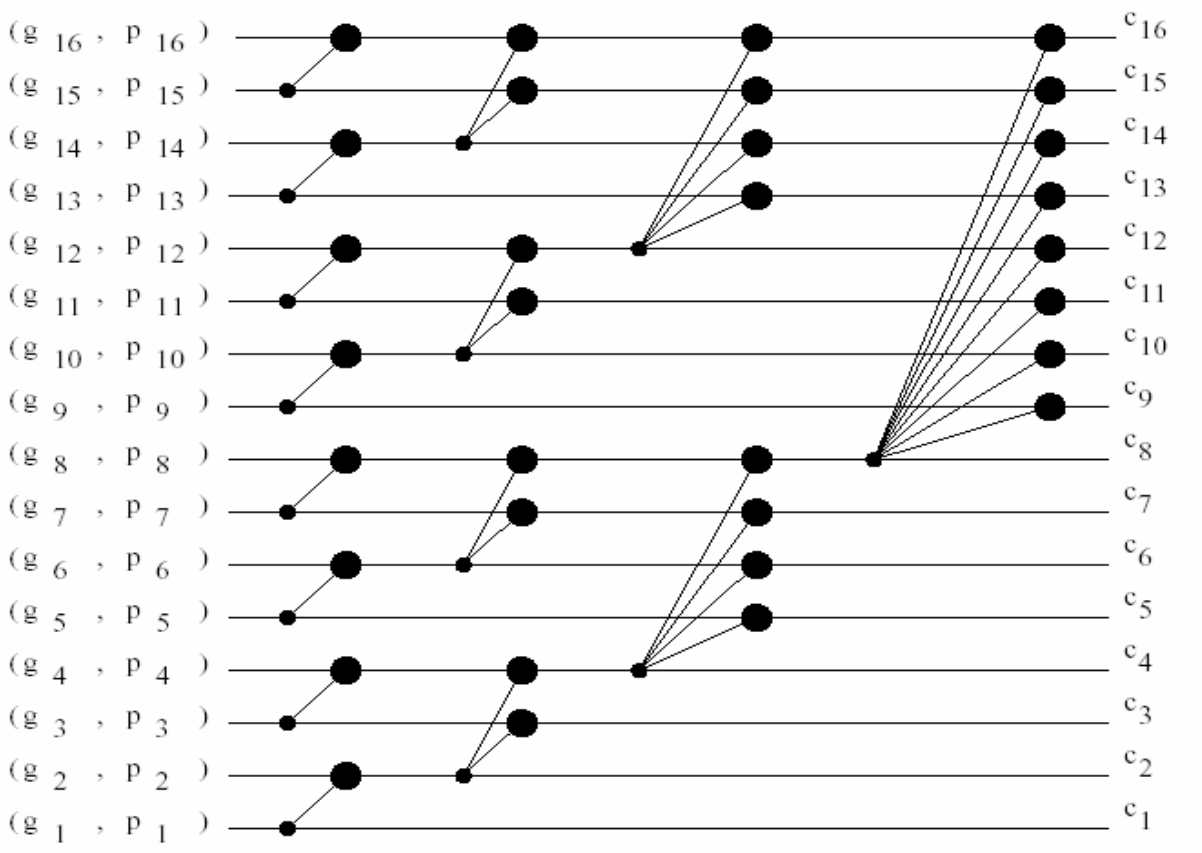


$t_p \sim \log_2(N)$

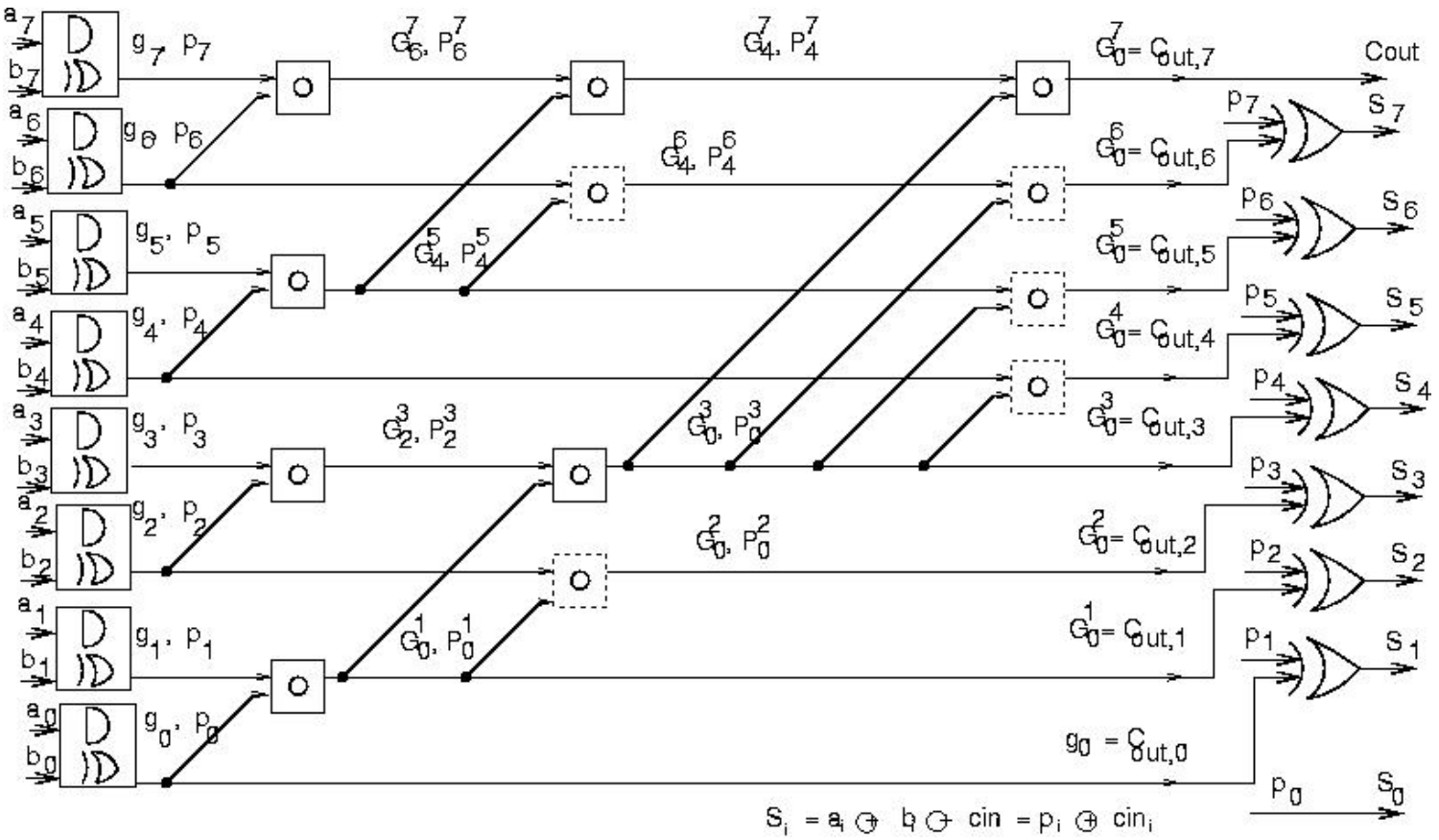
Brent-Kung BLC adder



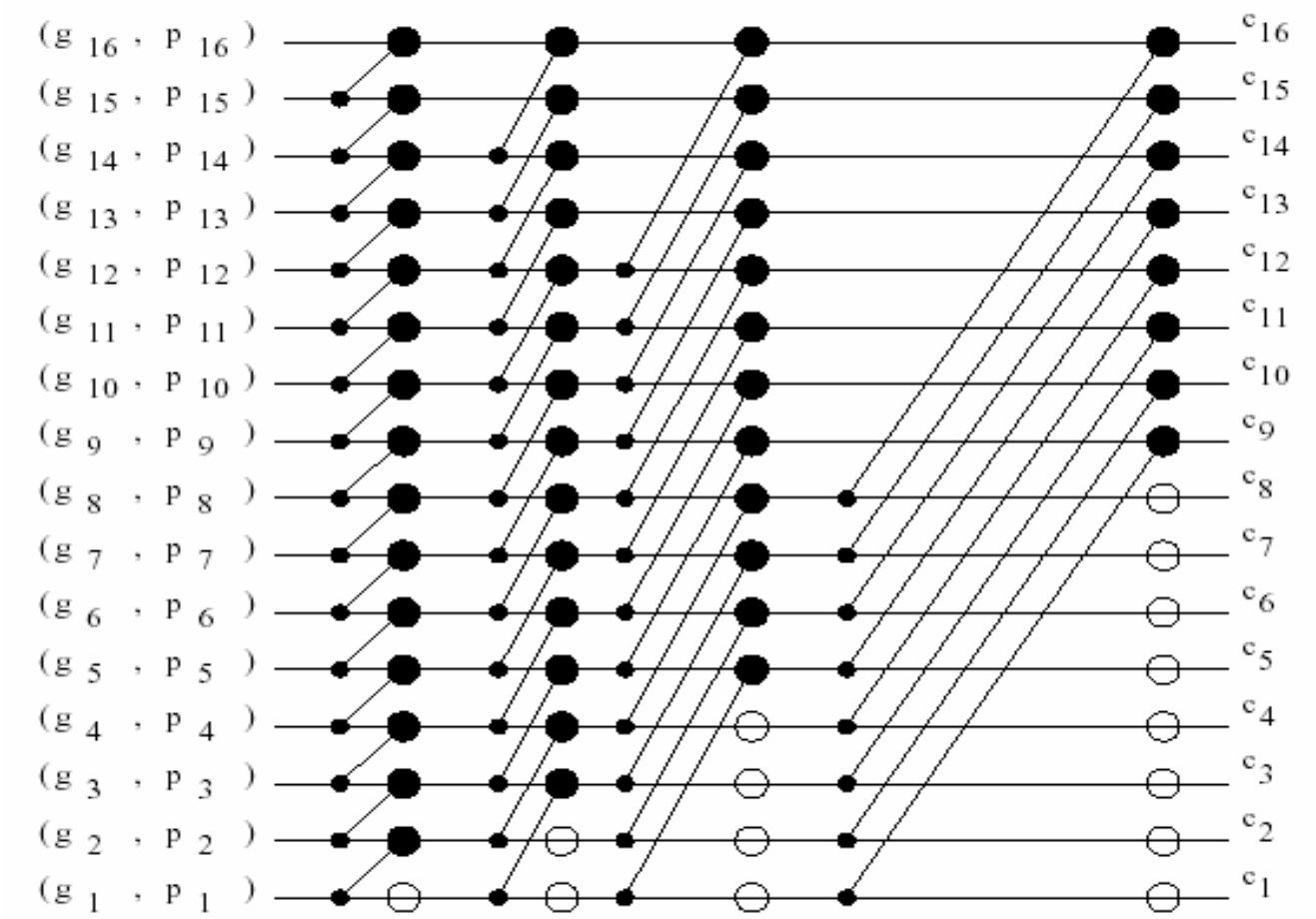
Folding of the inverse tree



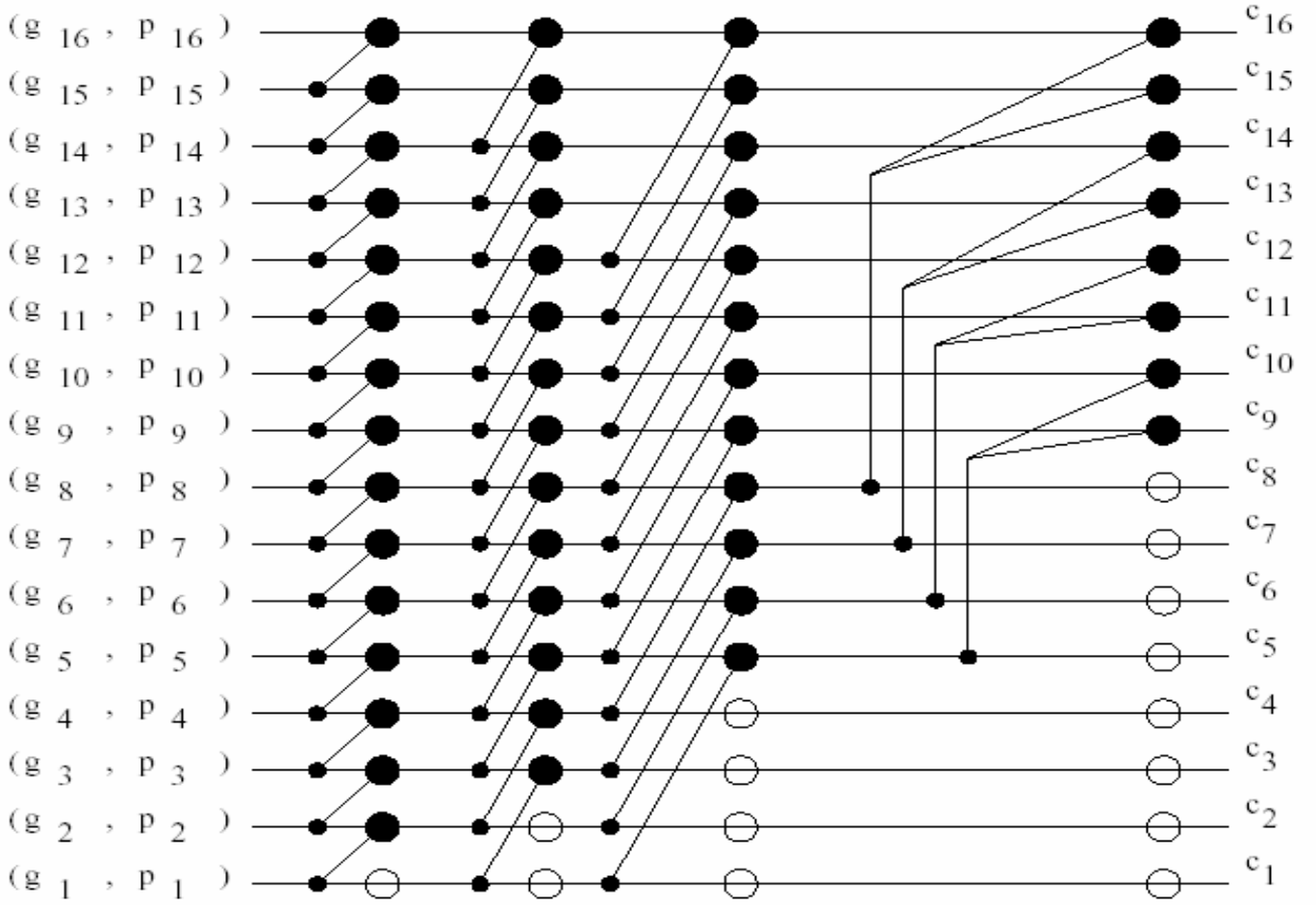
Folding the inverse tree



Dense tree with minimum fan-out



Dense tree with simple connections



Carry Lookahead Trees

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

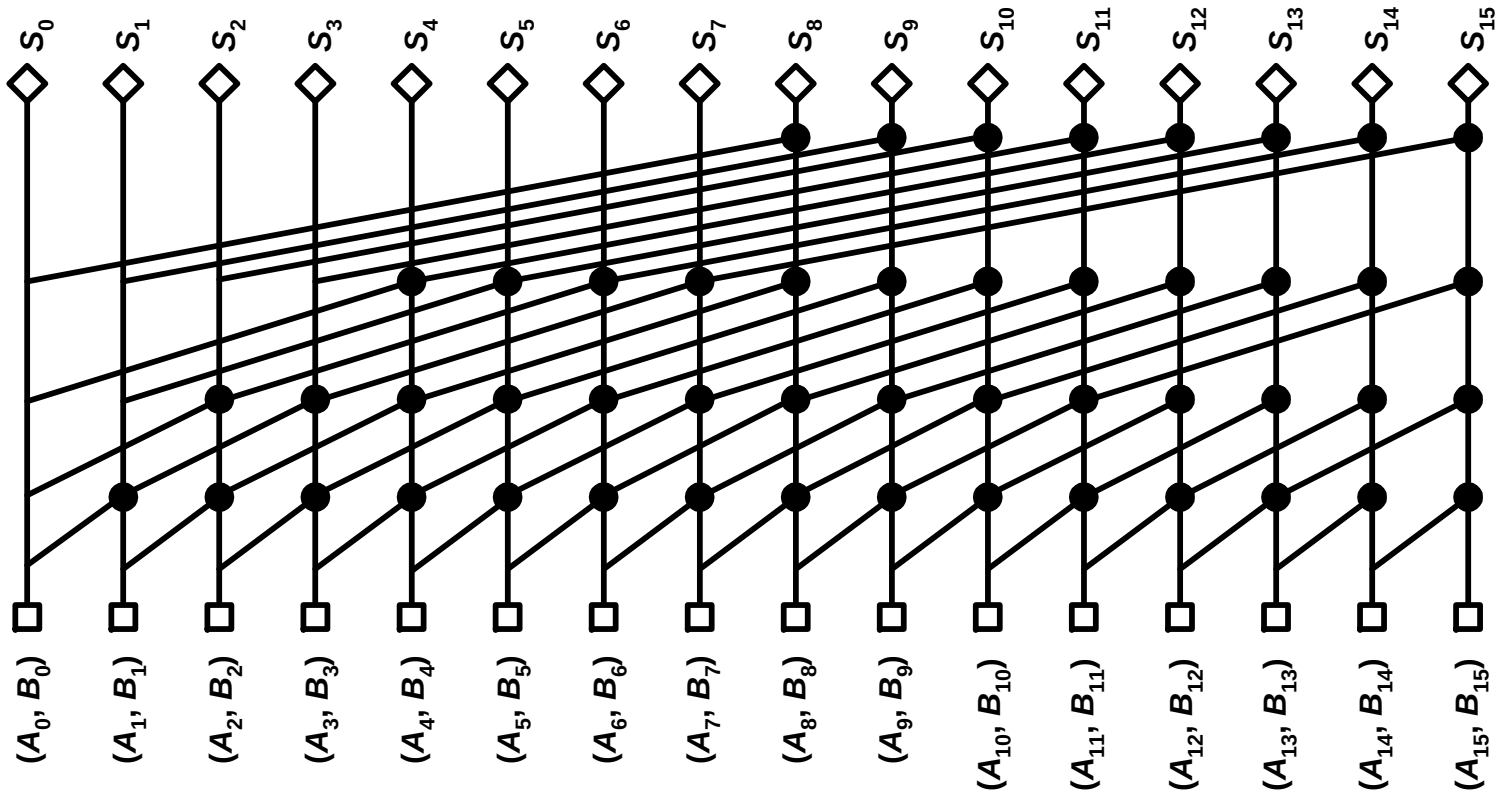
$$C_{o,1} = G_1 + P_1 G_0 + P_1 P_0 C_{i,0}$$

$$C_{o,2} = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{i,0}$$

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

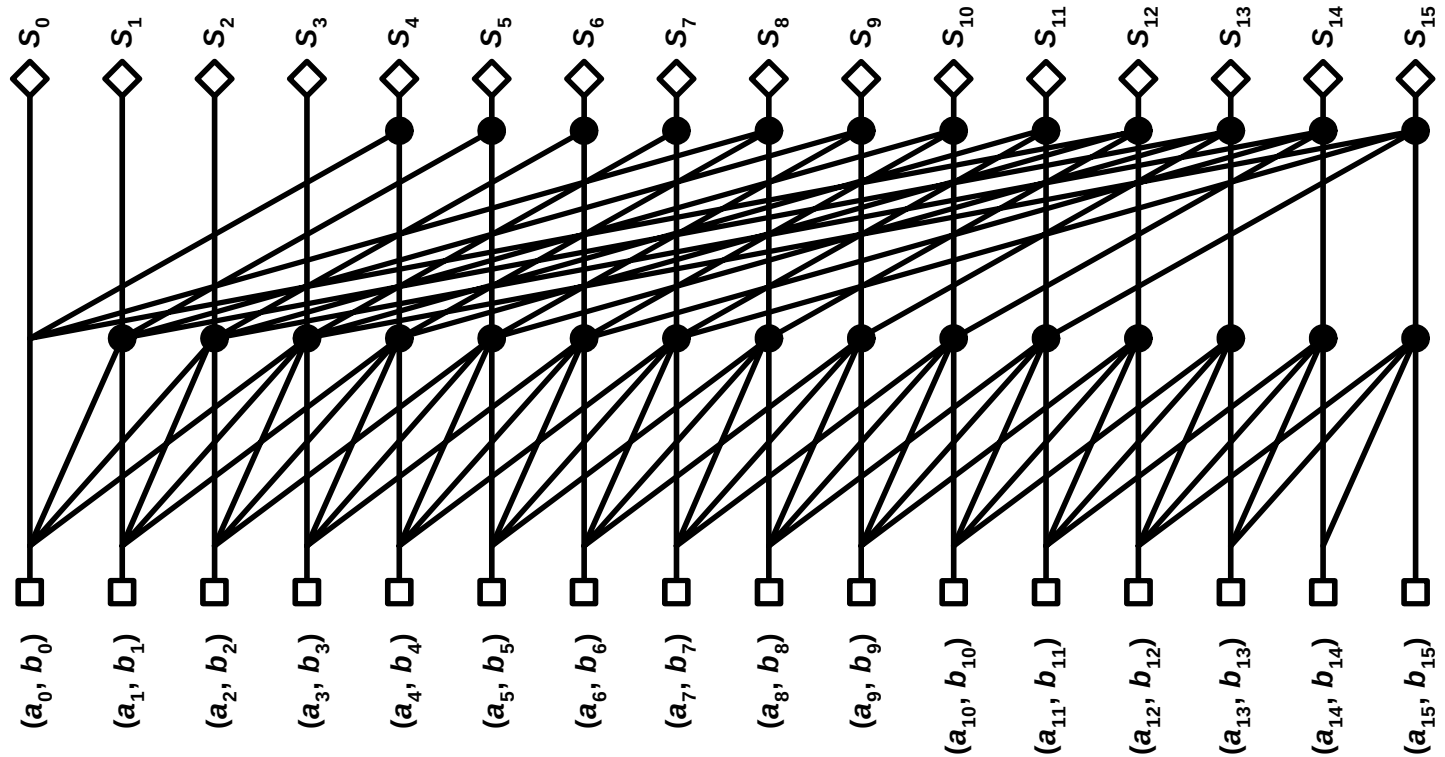
Can continue building the tree hierarchically.

Tree Adders



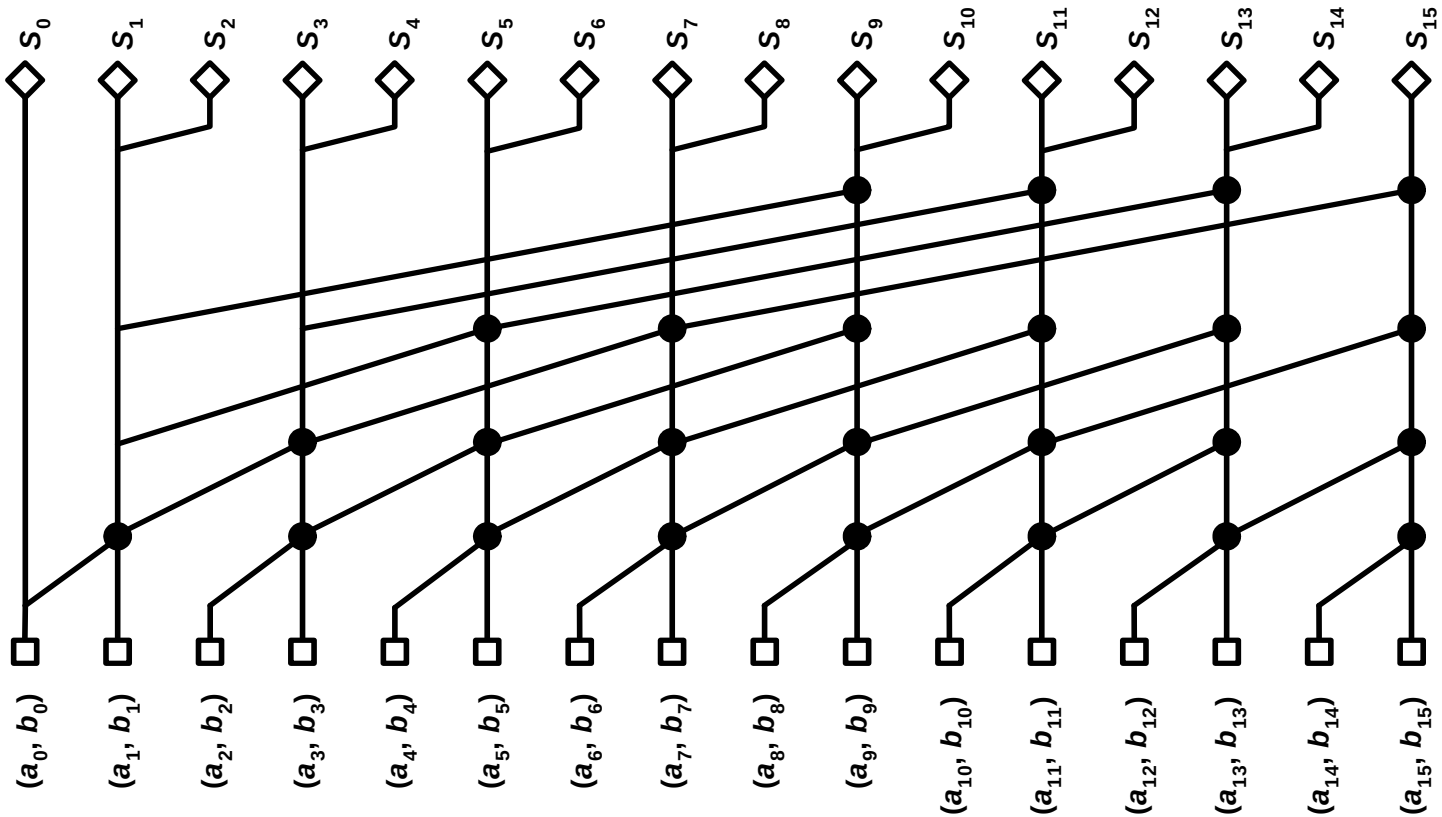
16-bit radix-2 Kogge-Stone tree

Tree Adders



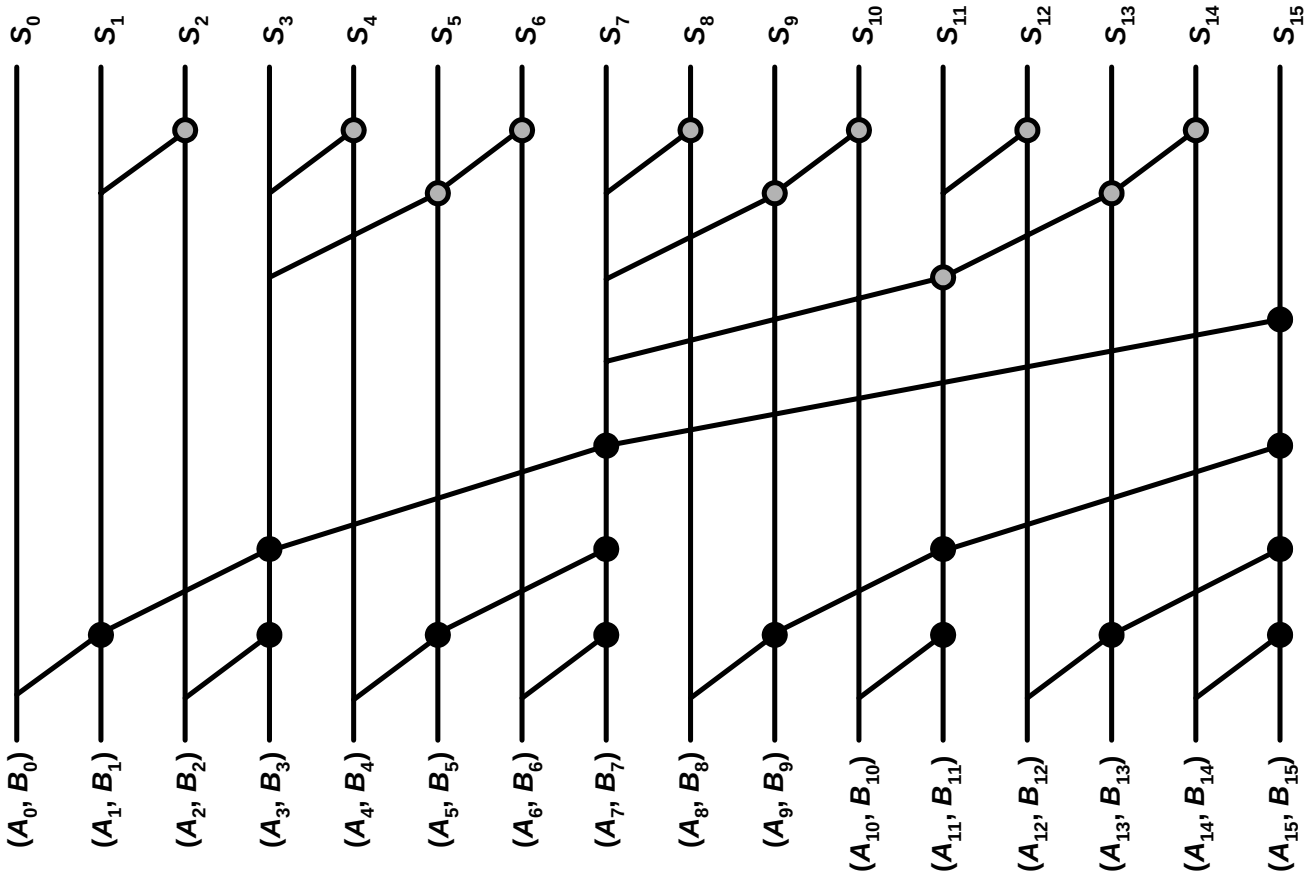
16-bit radix-4 Kogge-Stone Tree

Sparse Trees



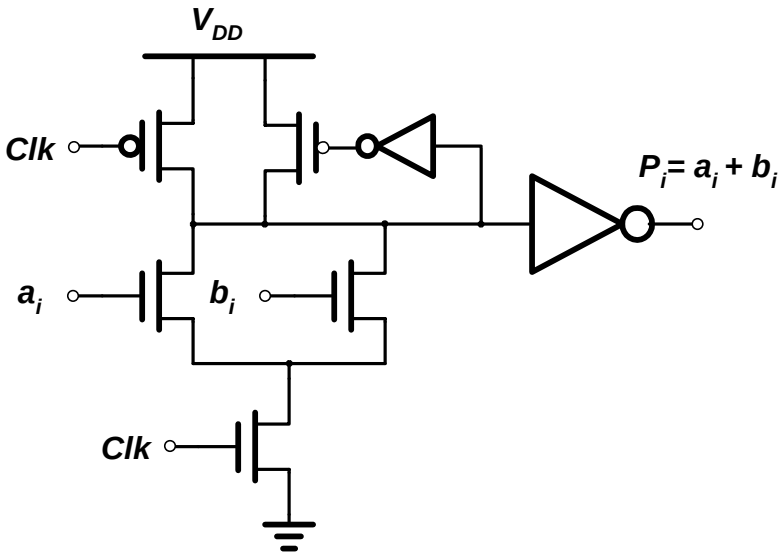
16-bit radix-2 sparse tree with sparseness of 2

Tree Adders

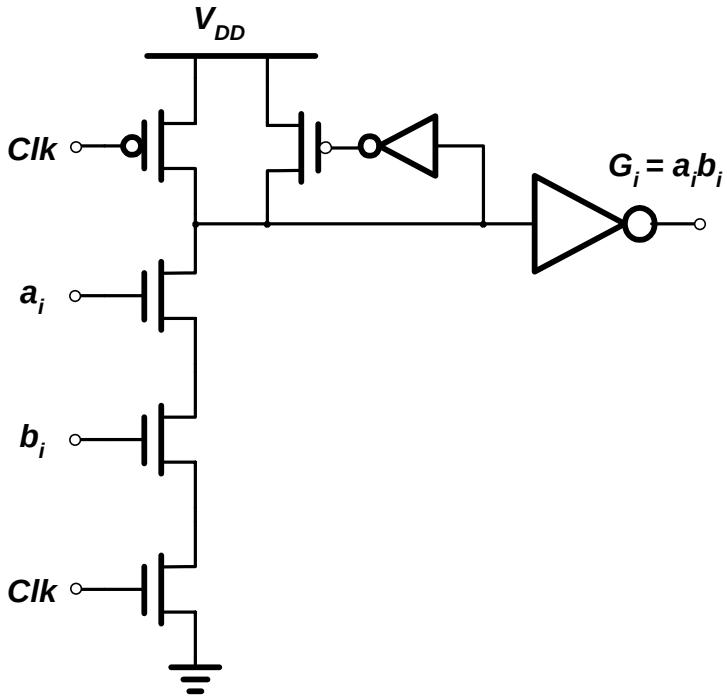


Brent-Kung Tree

Example: Domino Adder

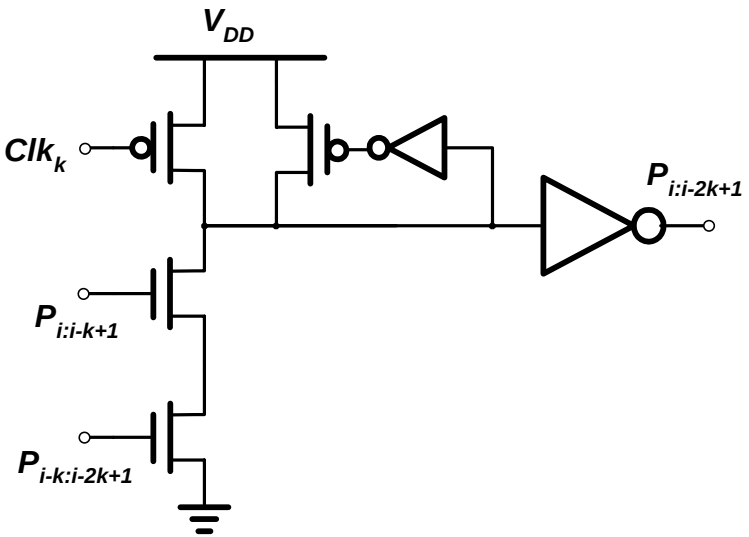


Propagate

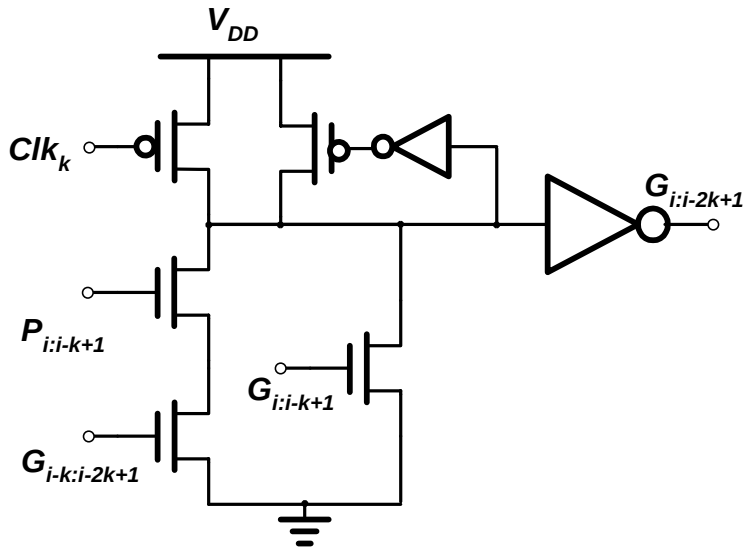


Generate

Example: Domino Adder

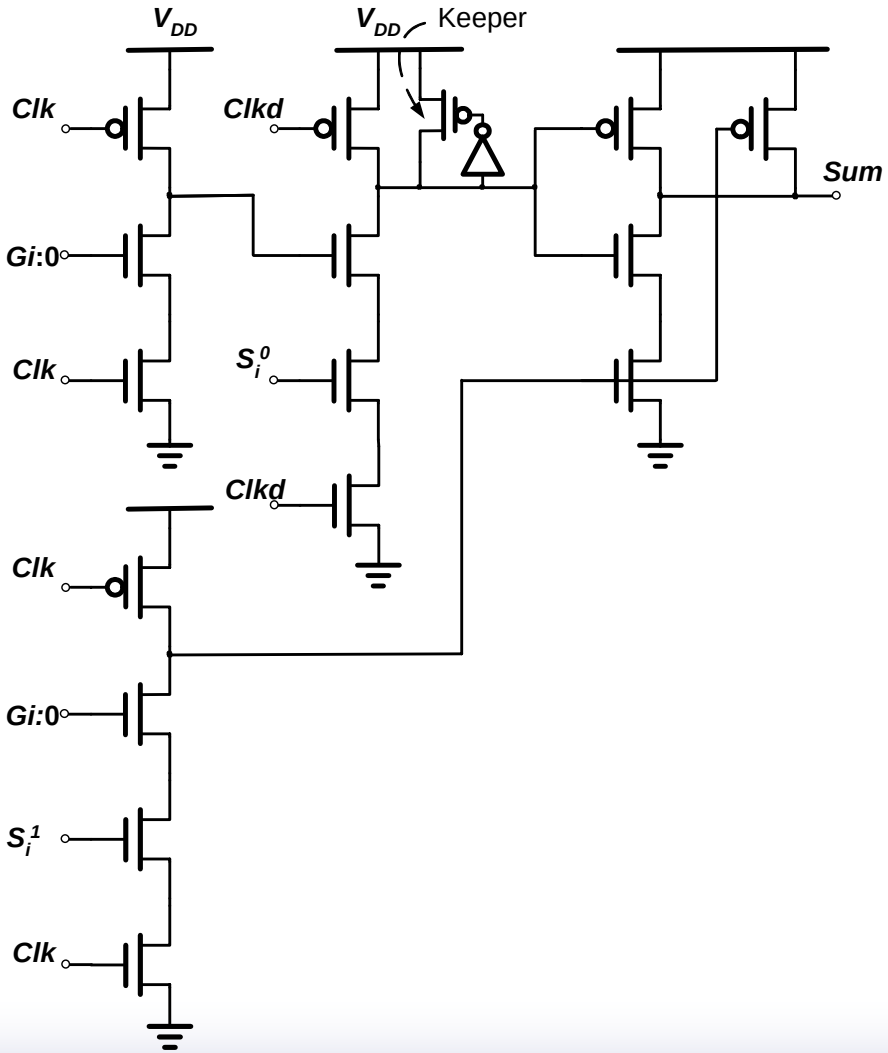


Propagate



Generate

Example: Domino Sum



Adders – Summary

Adder	Complexity	Latency
Ripple carry	N	N
Carry select [1]	$2N$	$(2\sqrt{N}-1)$
Carry select [2]	$2N$	$\sqrt{(2N-7/4)}+1/2$
Carry skip	N	$2(\sqrt{2N}-1)$
Bin. look. (tree)	$2(N-1)-\log N$	$2\log N$
Bin. look. (array)	$N\log N$	$2+\log N$

[1] Uniform blocks sized $K=\sqrt{N}$

[2] Scaled blocks with $K_{min}=1$ or $K_{min}=2$

Multipliers

The Binary Multiplication

$$X = \sum_{n=0}^{N-1} x_n\, 2^n\,, \qquad Y = \sum_{n=0}^{N-1} y_n\, 2^n$$

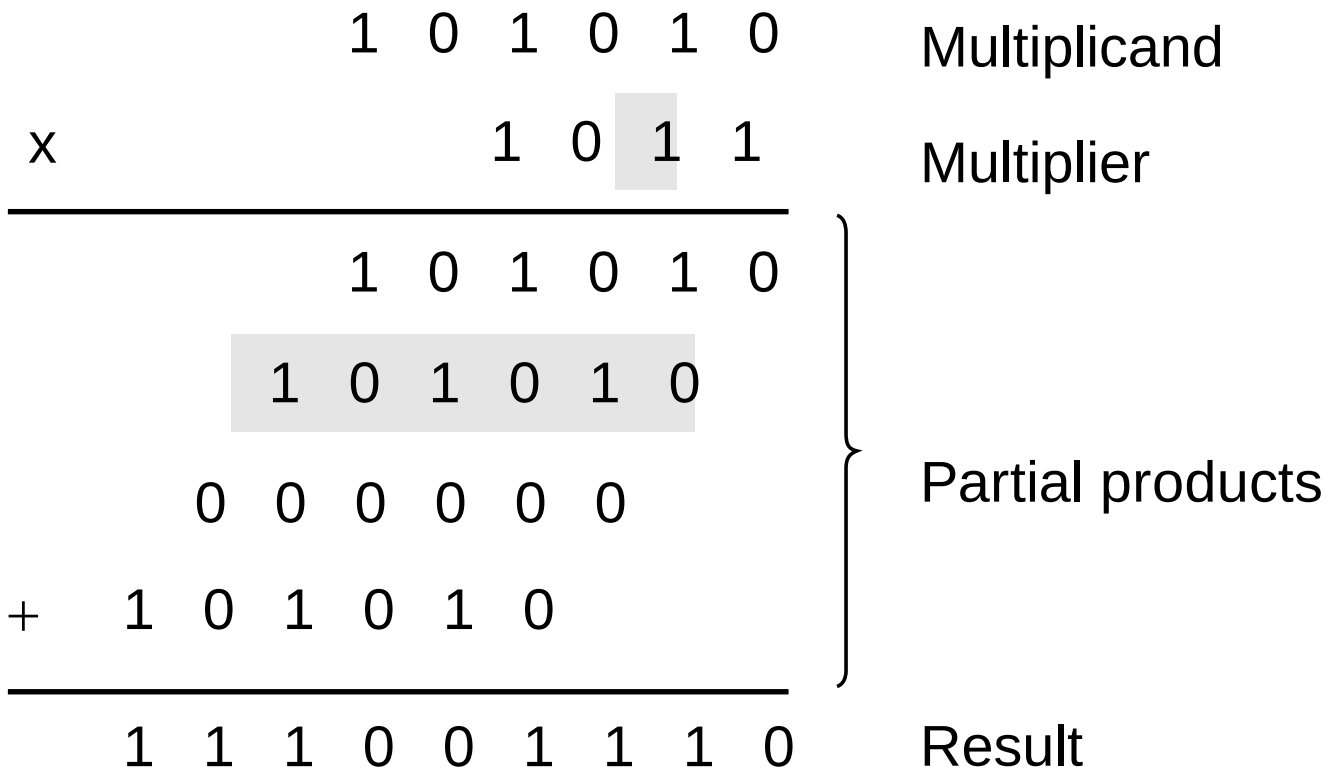
$$Z = XY = \left(\sum_{n=0}^{N-1} x_n\, 2^n\right)\left(\sum_{m=0}^{N-1} x_m\, 2^m\right)$$

$$= \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x_n\, y_m\, 2^{n+m} = \sum_{n=0}^{2N-1} z_n\, 2^n$$

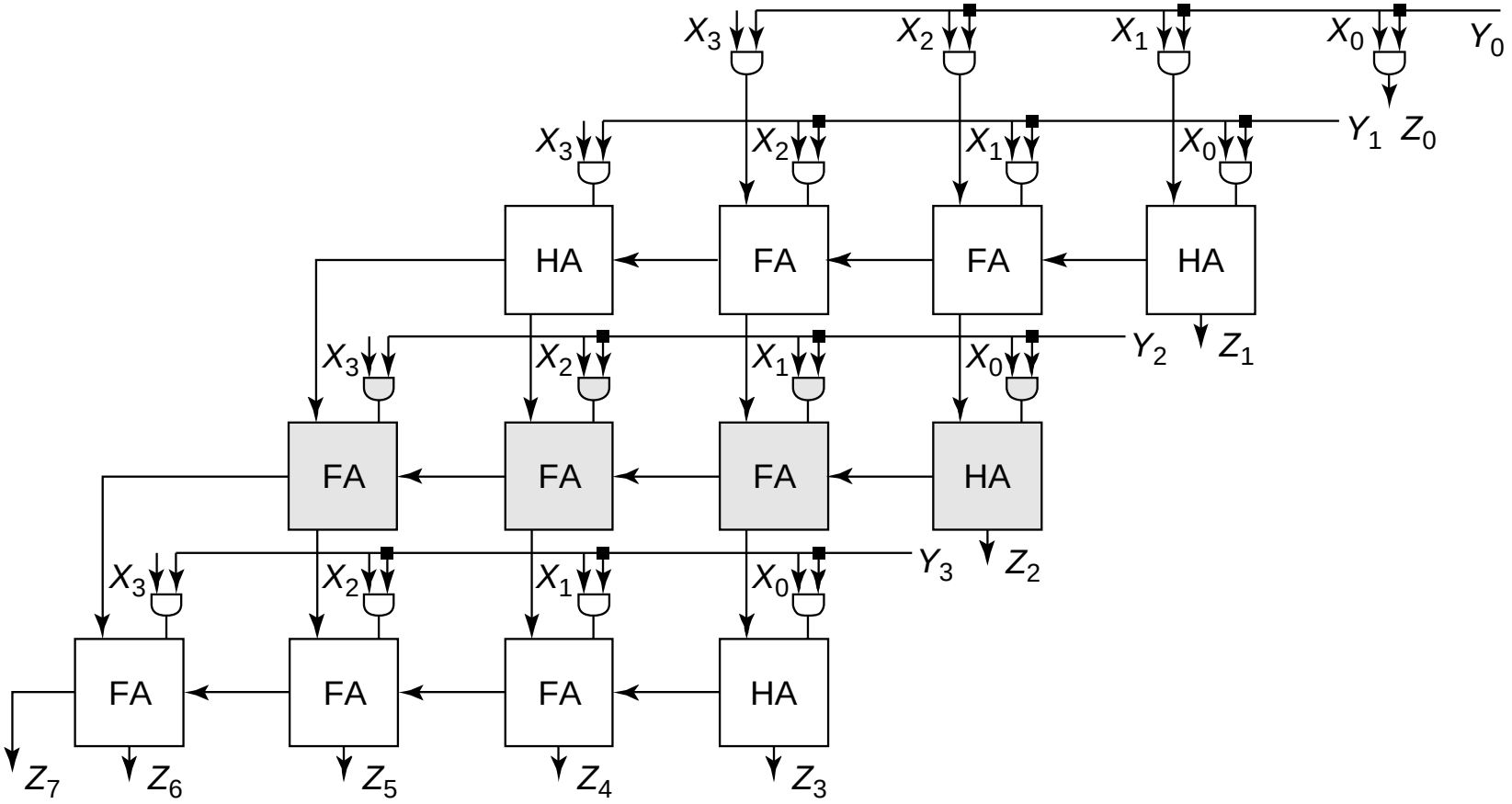
The Binary Multiplication

Multiplicand				y_3	y_2	y_1	y_0	$\times$
Multiplier				x_3	x_2	x_1	x_0	$=$
				x_0y_3	x_0y_2	x_0y_1	x_0y_0	
Partial			x_1y_3	x_1y_2	x_1y_1	x_1y_0		
Products		x_2y_3	x_2y_2	x_2y_1	x_2y_0			
	x_3y_3	x_3y_2	x_3y_1	x_3y_0				
Result	z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0

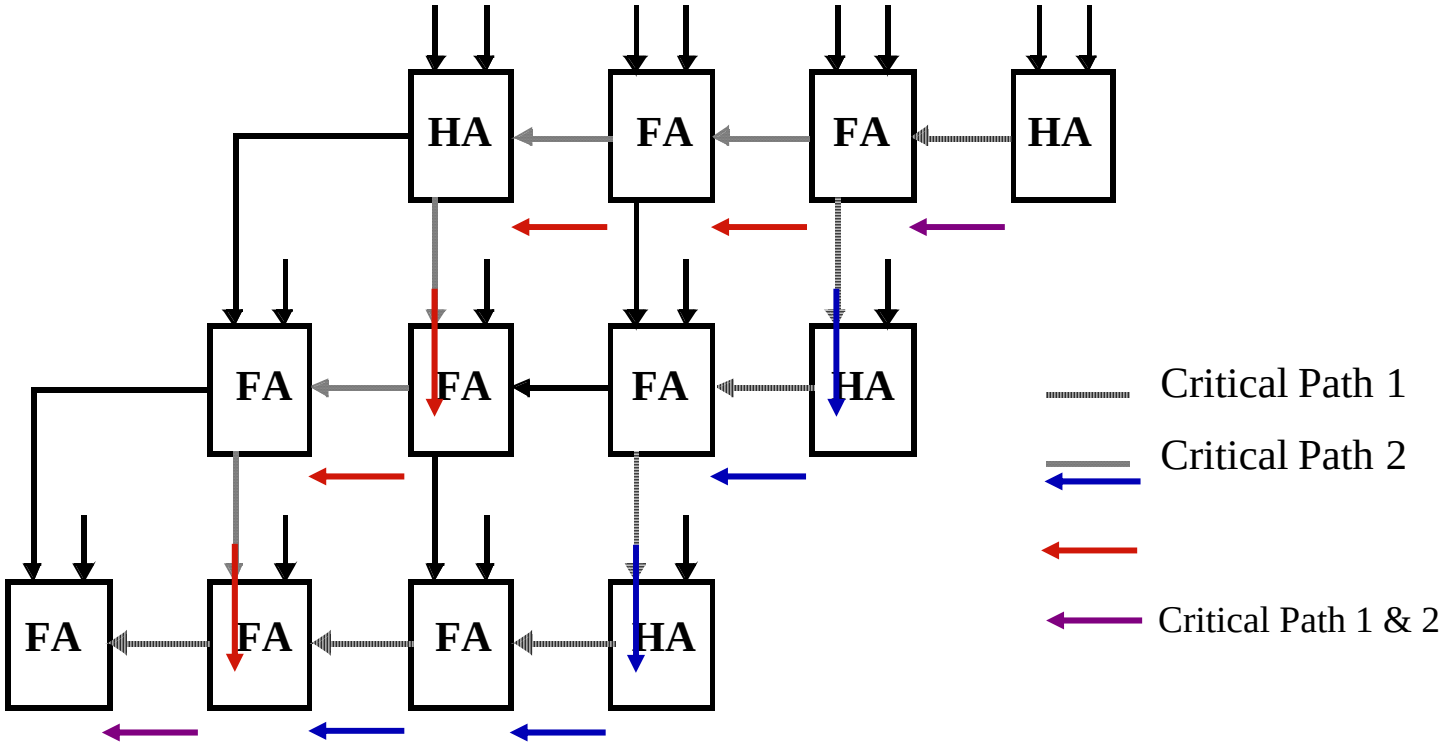
The Binary Multiplication



The Array Multiplier

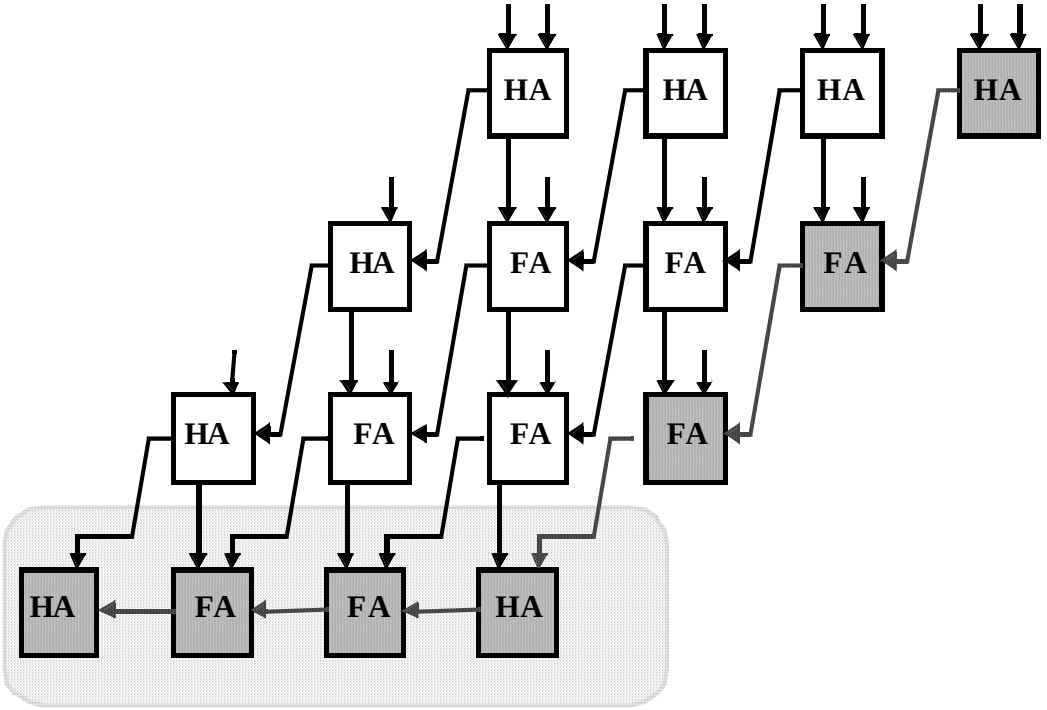


The MxN Array Multiplier — Critical Path



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

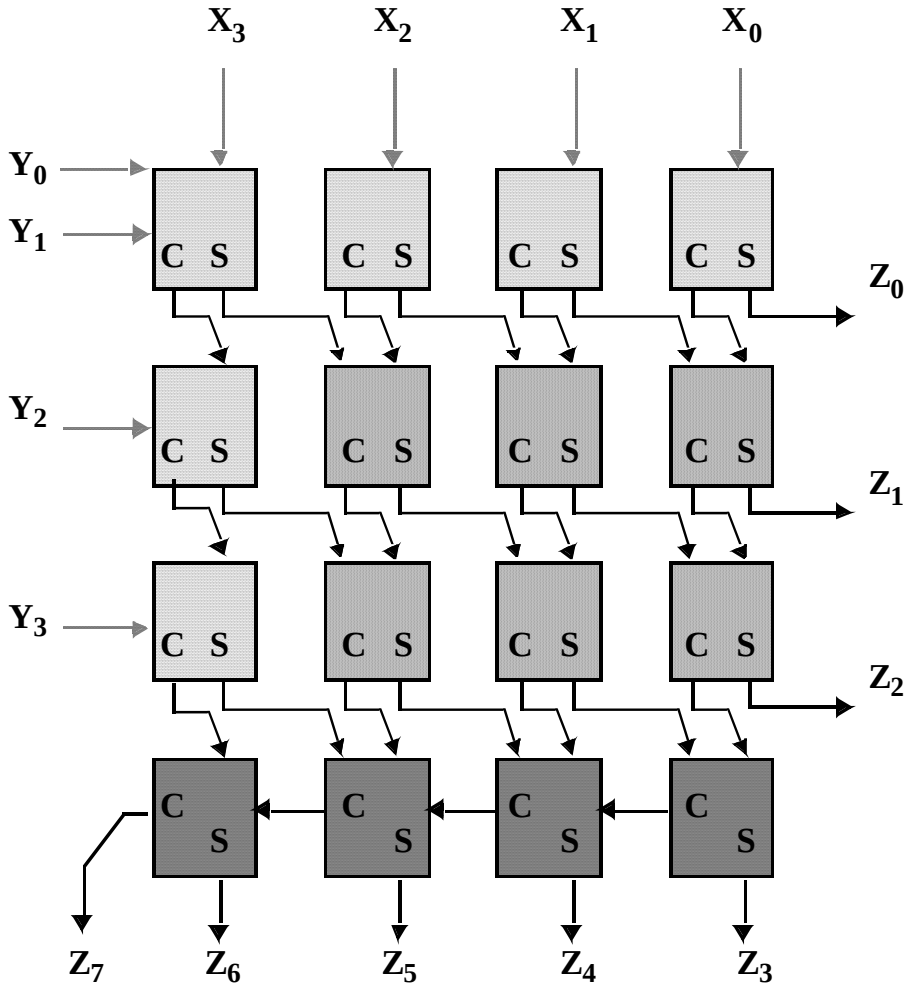
Carry-Save Multiplier



Vector Merging Adder

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

Multiplier Floorplan



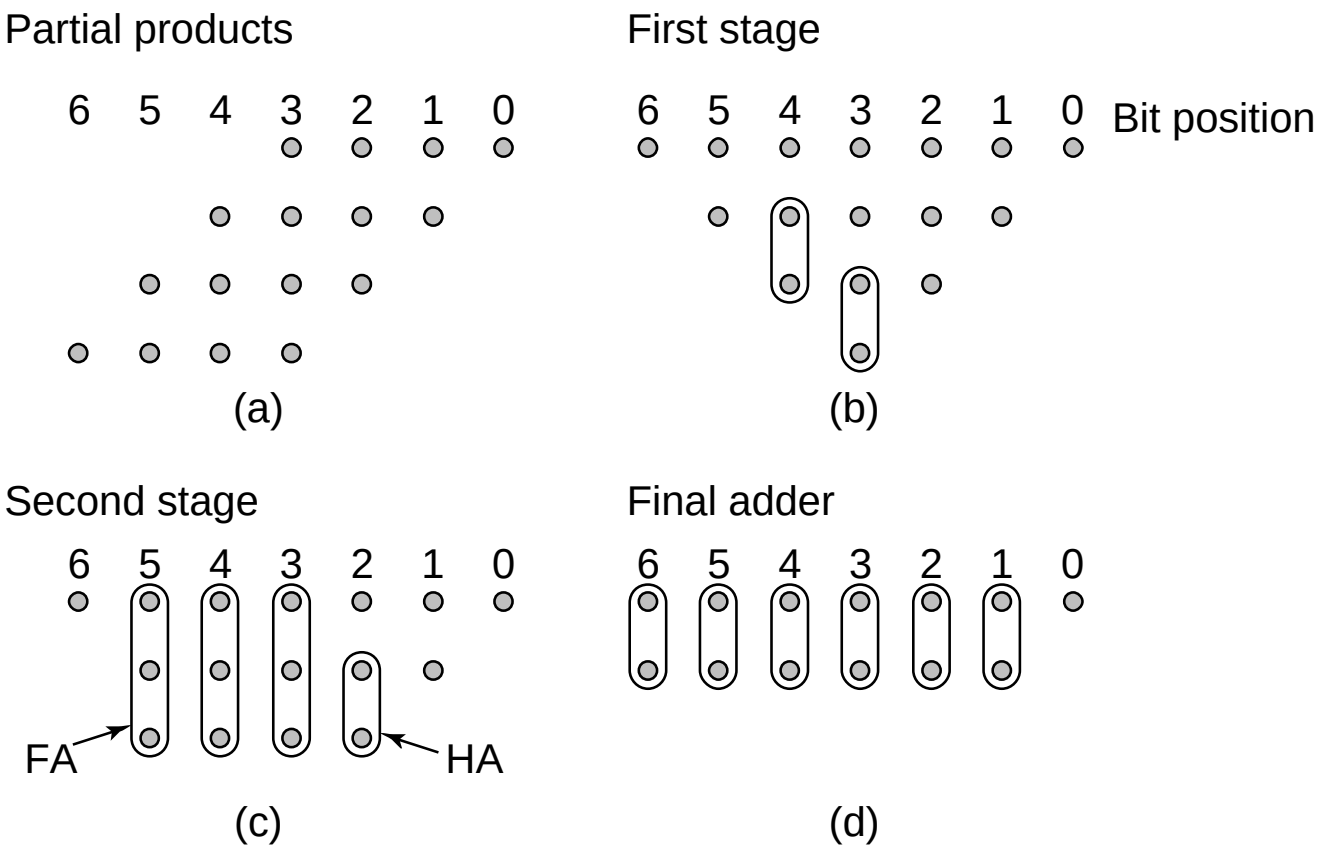
HA Multiplier Cell

FA Multiplier Cell

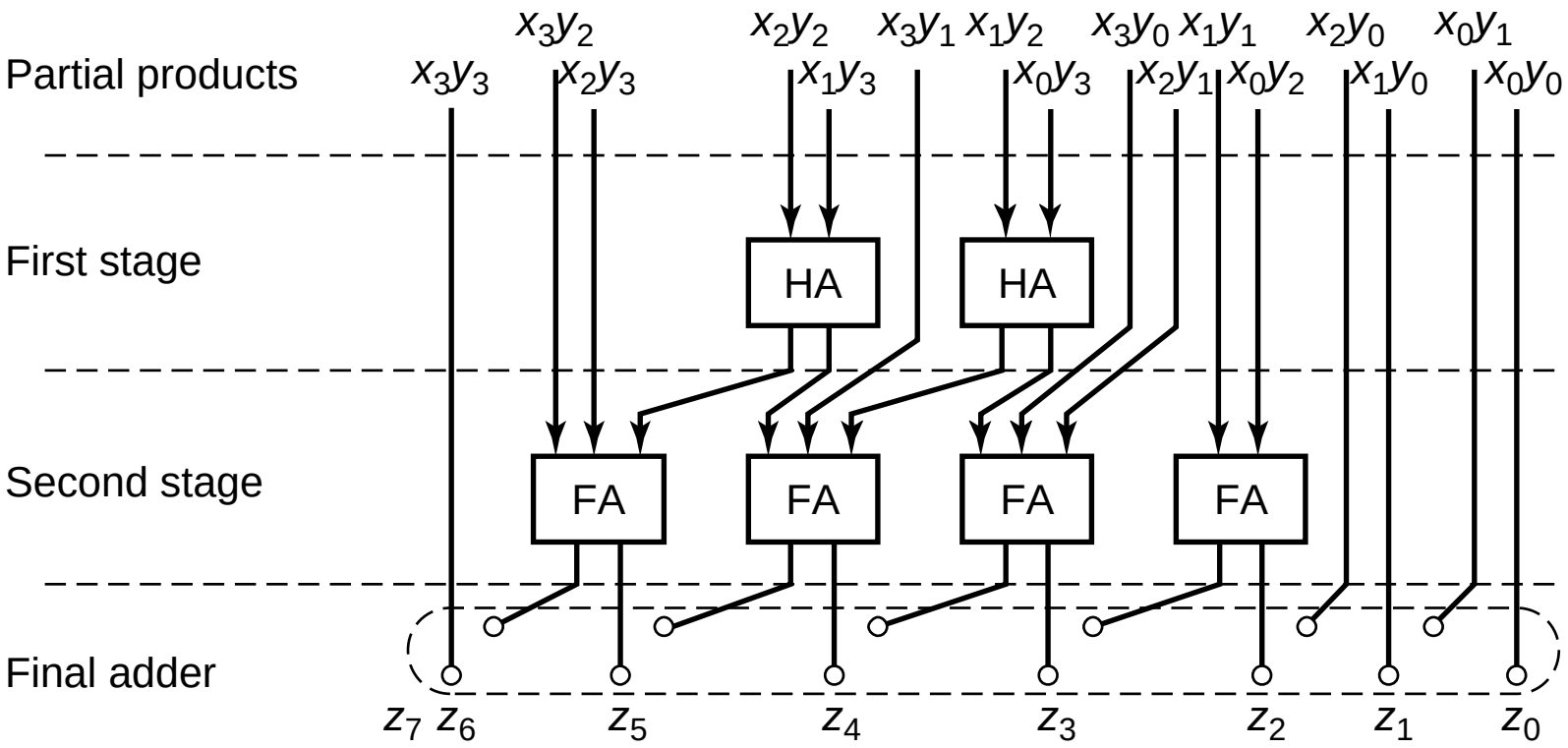
Vector Merging Cell

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

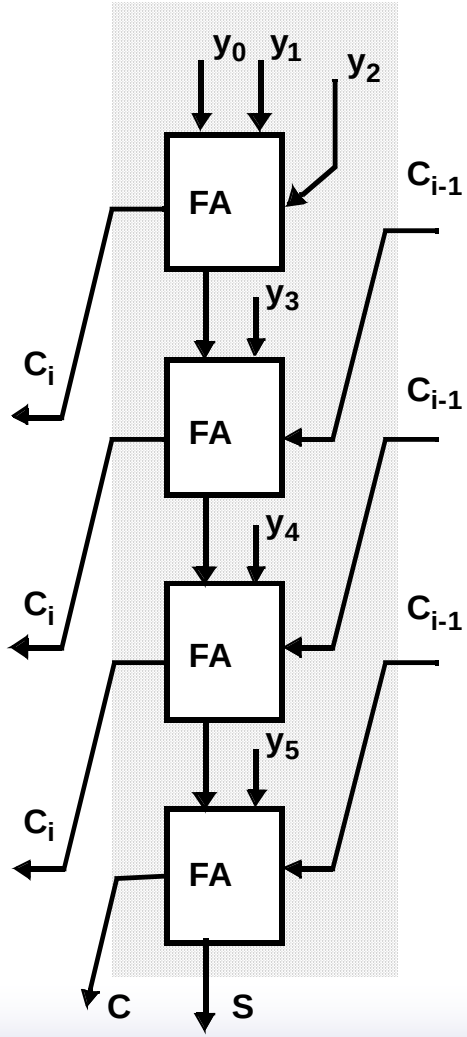
Wallace-Tree Multiplier

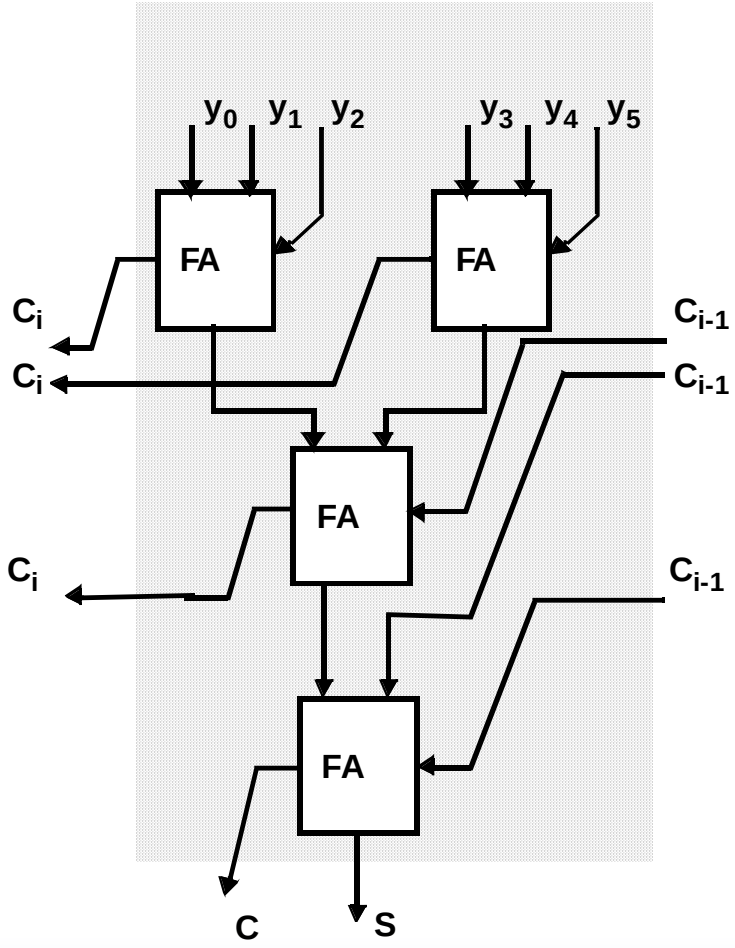


Wallace-Tree Multiplier



Wallace-Tree Multiplier





Wallace Tree Mult. Performance

$$\left(\frac{3}{2}\right)^n = \frac{N}{2}$$

$$n \log_2\left(\frac{3}{2}\right) = \log_2\left(\frac{N}{2}\right)$$

$$n = \frac{\log_2(N/2)}{\log_2(3/2)} = 1.71\left(\log_2 N - 1\right)$$

$$t_{mult} = 1.71\left(\log_2 N - 1\right) t_{sum} + t_{adder}$$

Wallace Tree Multiplier Complexity

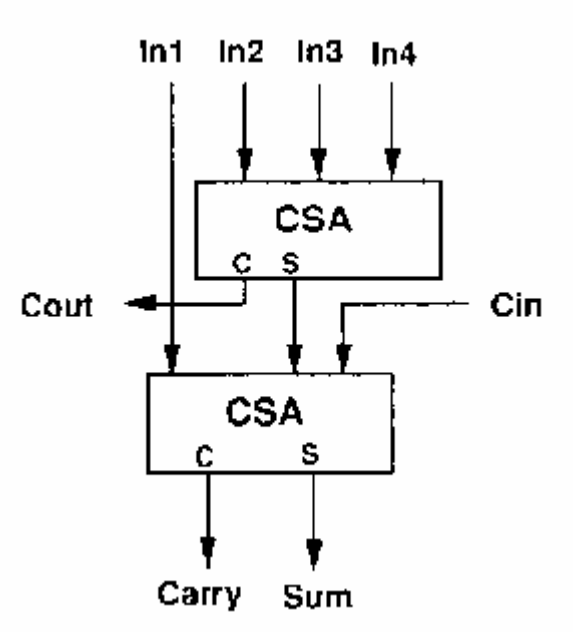
$$N_{CSA} = \frac{N}{3} + \left(\frac{2}{3}\right)\frac{N}{3} + \left(\frac{2}{3}\right)^2\frac{N}{3} + \ldots\ldots + \left(\frac{2}{3}\right)^{n-1}\frac{N}{3}$$

$$= \frac{N}{3}\sum_{n=0}^{n}\left(\frac{2}{3}\right)^{n-1}$$

$$= \frac{N}{3}\frac{1-(2/3)^n}{1-2/3} = N-2$$

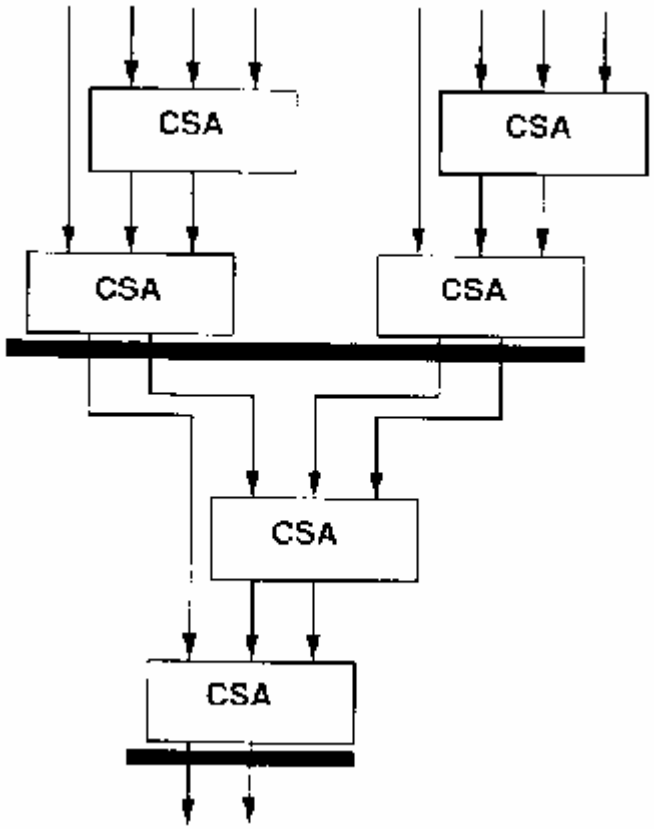
$$A_{WTM} \simeq N\,(N-2)\,A_{adder}$$

4:2 Adder

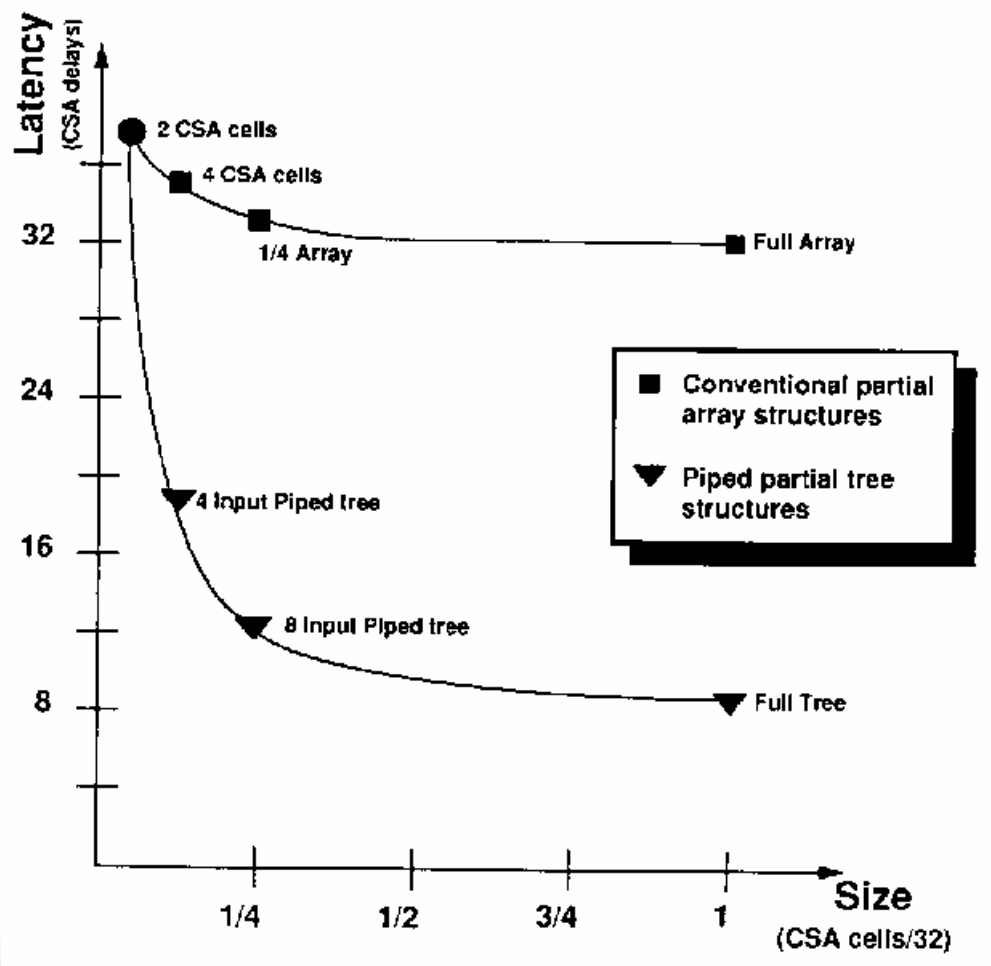


n	Cin	Cout	Carry	Sum
0	0	0	0	0
1	0	0	0	1
2	0	*	*	0
3	0	1	0	1
4	0	1	1	0
0	1	0	0	1
1	1	0	1	0
2	1	*	*	1
3	1	1	1	0
4	1	1	1	1

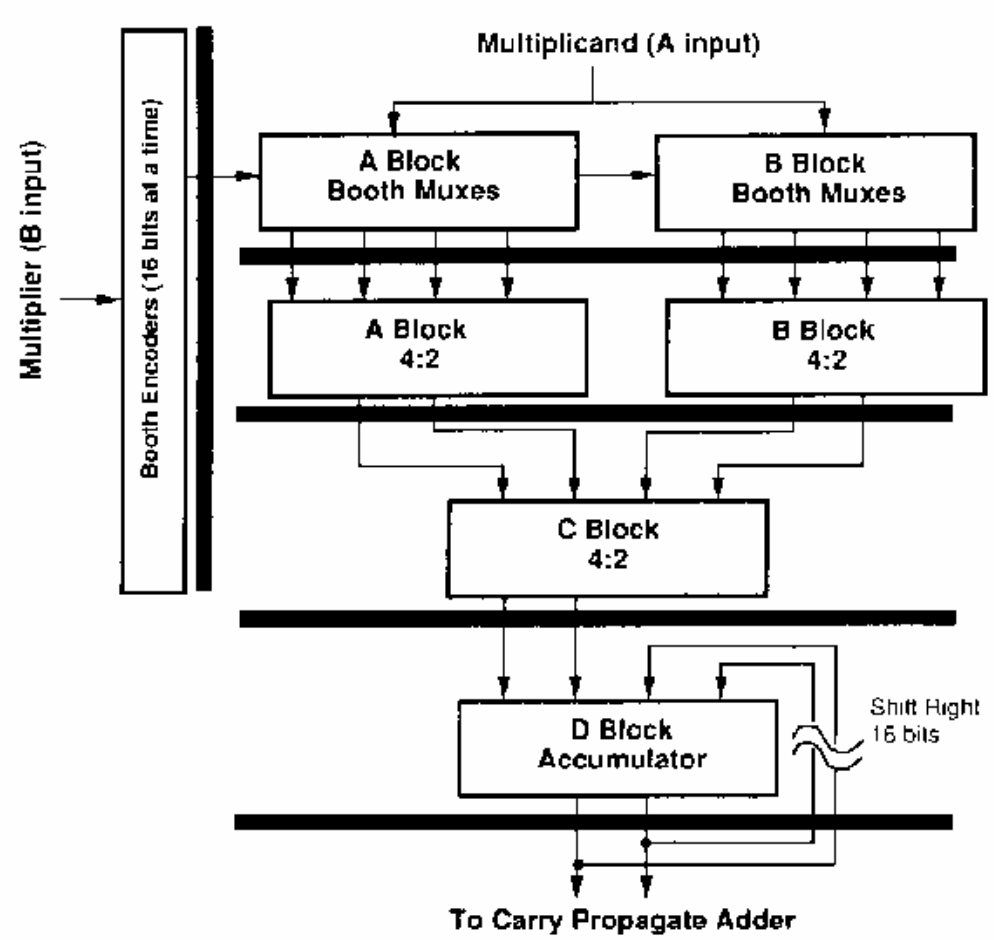
Eight-input Tree



Architectural comparison of multiplier solutions



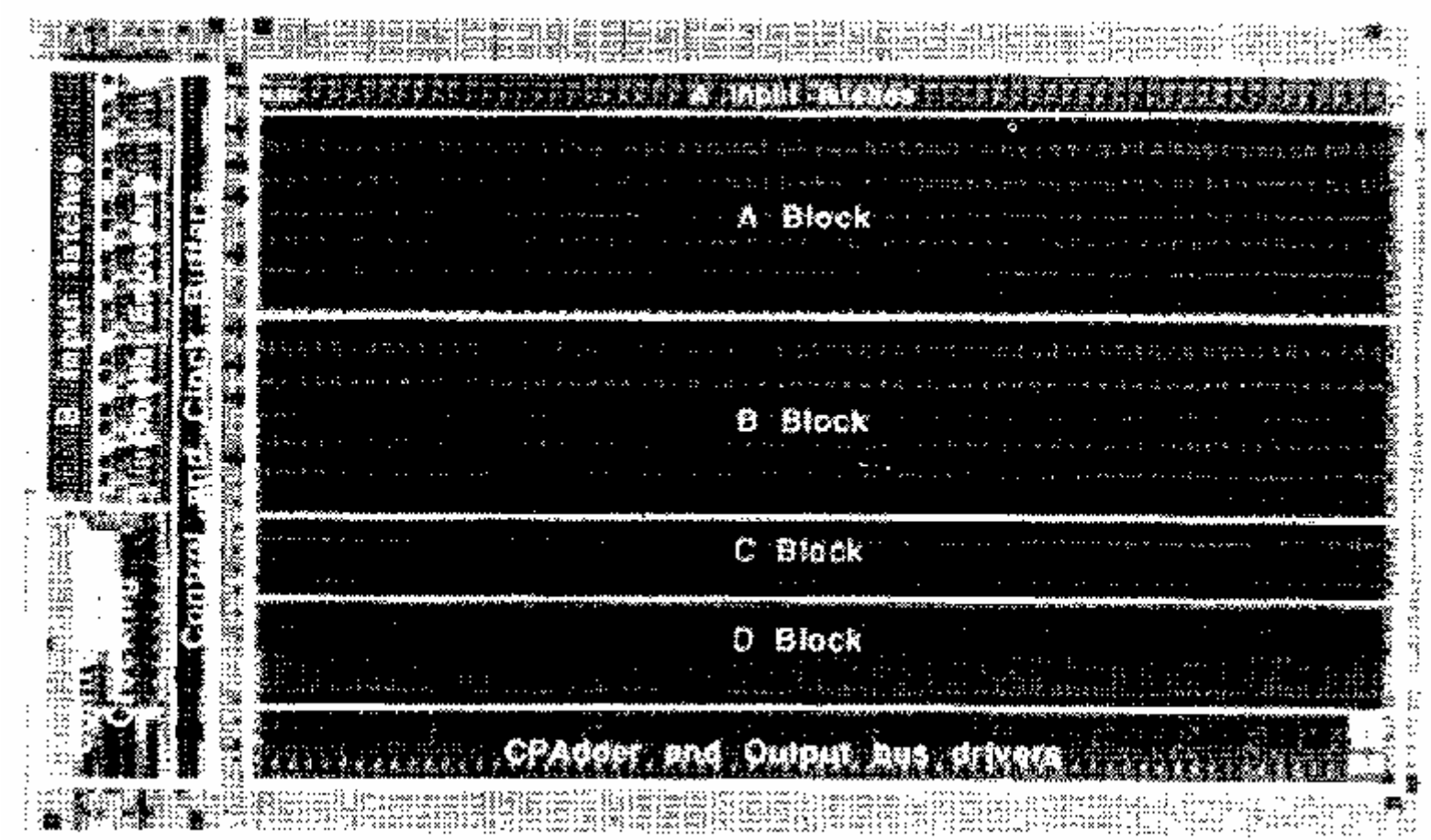
SPIM Architecture



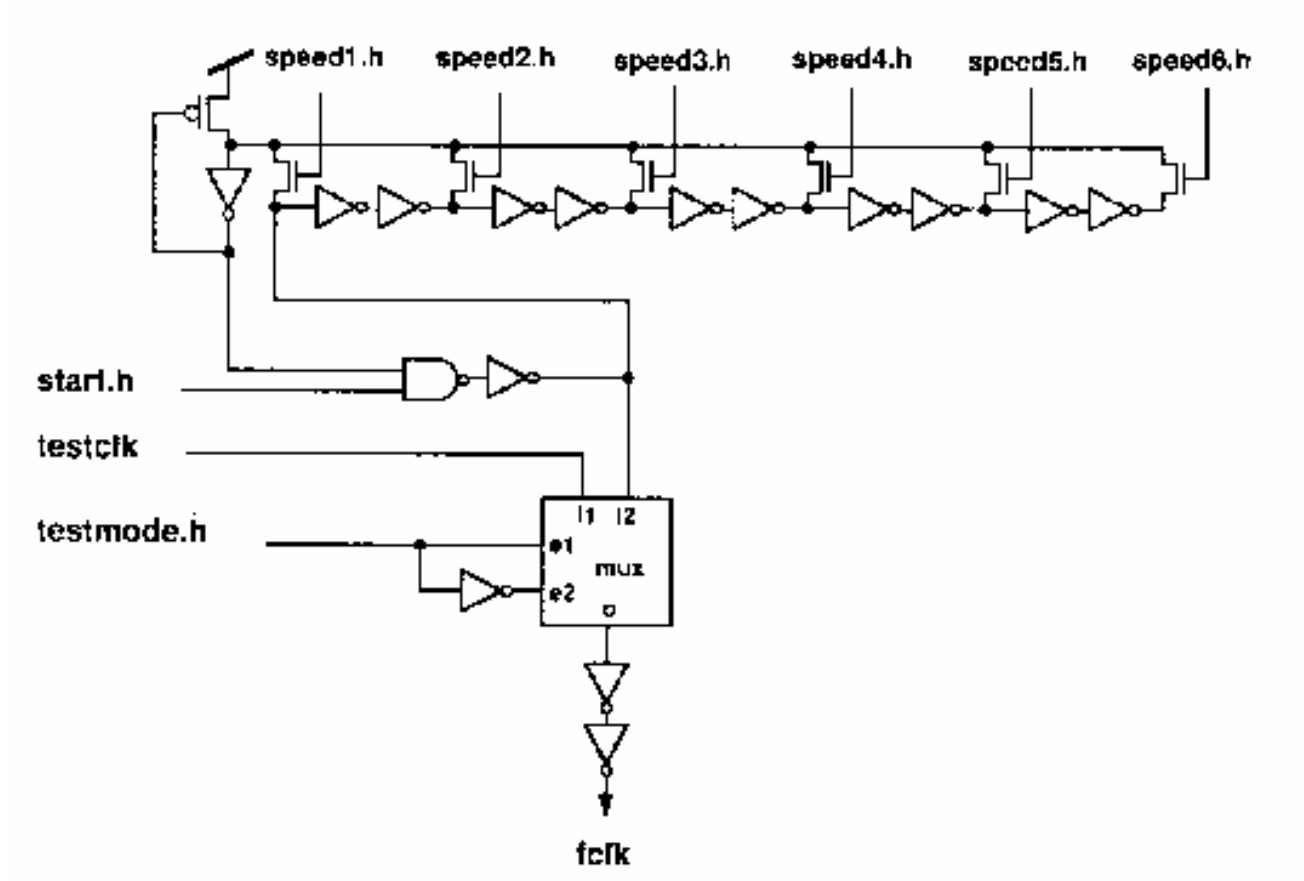
SPIM Pipe Timing

Action \ Cycle	0	1	2	3	4	5	6	7
Booth Encode	startup 0-15	16-31	32-47	48-63				
A and B block Booth Muxs		0-15	16-31	32-47	48-63			
A Block CSA's			0-7	16-23	32-39	48-55		
B Block CSA's			8-15	24-31	40-47	56-63		
C Block				0-15	16-31	32-47	48-63	
D Block					clear 0-15	16-31	32-47	48-63

SPIM Microphotograph



SPIM clock generator circuit



Binary Tree Multiplier Performance

$$2^n = \frac{N}{2}$$

$$n = \log_2\left(\frac{N}{2}\right)$$

$$t_{mult} = 2\left(\log_2 N - 1\right) t_{sum} + t_{adder}$$

Binary Tree Multiplier Complexity

$$N_{CSA} = \frac{N}{4} + \left(\frac{1}{2}\right)\frac{N}{4} + \left(\frac{1}{2}\right)^2\frac{N}{4} + \ldots\ldots + \left(\frac{1}{2}\right)^{n-1}\frac{N}{4}$$

$$= \frac{N}{4}\sum_{k=0}^{n-1}\left(\frac{1}{2}\right)^k$$

$$= \frac{N}{4}\frac{1-(1/2)^n}{1-1/2} = \frac{N}{2} - 1$$

$$A_{BTM} \simeq N\,(N-2)\,A_{adder}$$

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

Multipliers – Summary

Multiplier	Complexity	Latency
Serial multiplier	N	N^2
Serial carry save	N	$2N$
Array multiplier	$N(N-2)$	$N-2$
Wallace tree	$N(N-2)$	$1.71\,(\log N - 1)$
Binary tree	$N(N-2)$	$2\,(\log N - 1)$

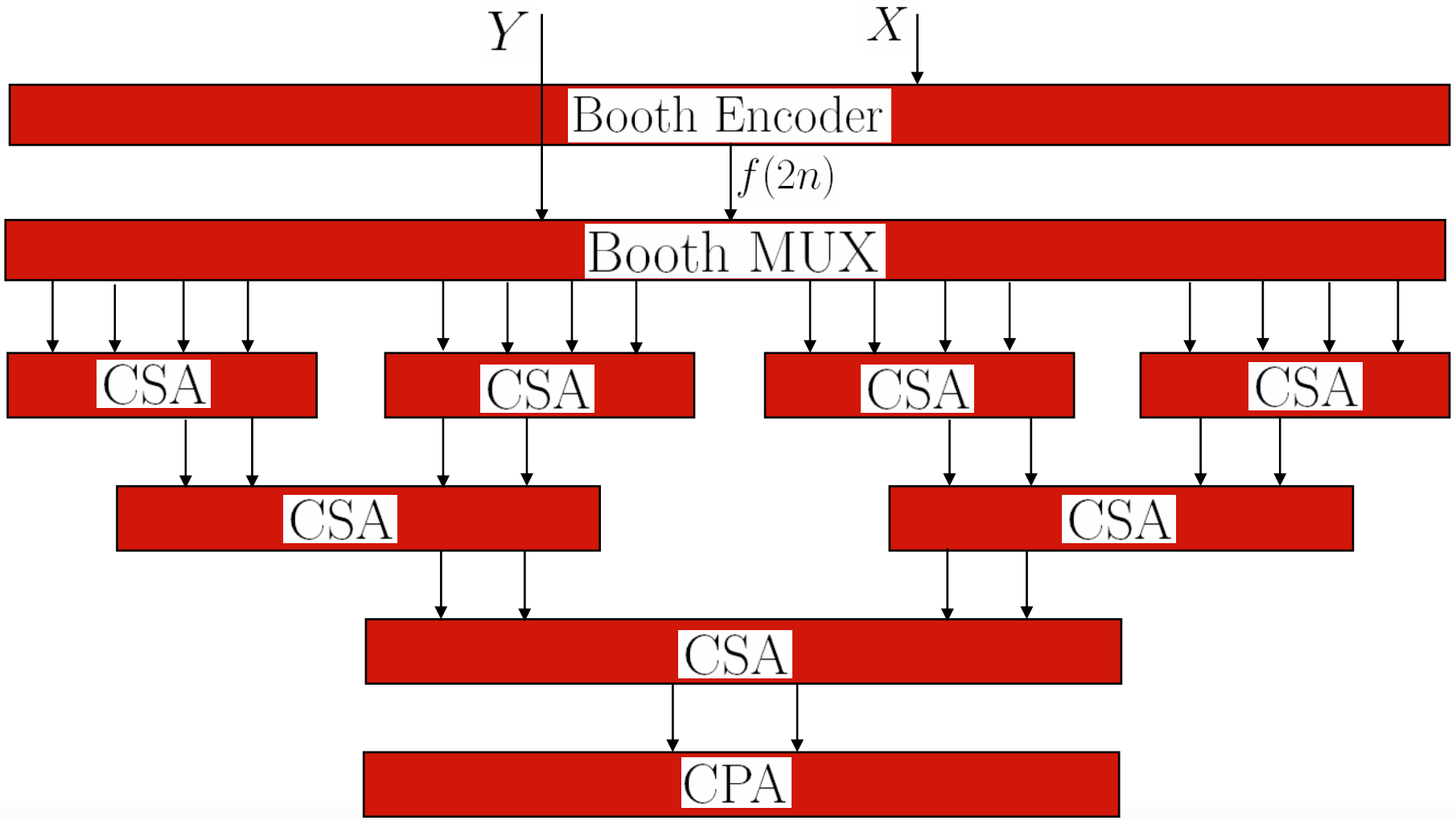
Booth encoding

$$
\begin{aligned}
X &= \sum_{n=0}^{N-2} x_n \, 2^n - x_{N-1} \, 2^{N-1} \\
&= \sum_{n=0}^{N/2-1} x_{2n} \, 2^{2n} + \sum_{n=1}^{N/2-1} x_{2n-1} \, 2^{2n-1} - x_{N-1} \, 2^{N-1} \\
&= \sum_{n=0}^{N/2-1} \left(x_{2n-1} + x_{2n} - 2x_{2n+1} \right) 2^{2n}
\end{aligned}
$$

Booth encoding

x_{2n+1}	x_{2n}	x_{2n-1}	$f(2n)$	$f(2n)\,Y$
0	0	0	0	0
0	0	1	1	Y
0	1	0	1	Y
0	1	1	2	$2Y$
1	0	0	-2	$-2Y$
1	0	1	-1	$-Y$
1	1	0	-1	$-Y$
1	1	1	0	0

Tree Multiplier with Booth Encoding



The f.p. addition algorithm

- Exponent comparison and swap (if needed)
- Mantissas' aligment
- Addition
- Normalization of result
- Rounding of result

The f.p. multiplication algorithm

- Mantissas' multiplication
- Exponent addition
- Mantissa normalization and exponent adjusting (if needed)
- Rounding of result

Dividers

Iterative Division (Newton-Raphson)

$$\frac{a}{b} = \frac{1}{b} \times a$$

$$f(x) = \frac{1}{x} - b = 0 \quad \rightarrow \quad x^* = \frac{1}{b}$$

$$y(x) \simeq f(x_0) + f'(x_0)\,(x - x_0)$$

Iterative Division (Newton-Raphson)

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = x_0 - \frac{1/x_0 - b}{-1/x_0^2} = x_0(2 - b\,x_0)$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = x_1 - \frac{1/x_1 - b}{-1/x_1^2} = x_1(2 - b\,x_1)$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{1/x_n - b}{-1/x_n^2} = x_n(2 - b\,x_n)$$

Quadratic Convergence of the Newton Method

$$\varepsilon_n = \left| \frac{x_n - 1/b}{1/b} \right| = |1 - b\,x_n|$$

$$\begin{aligned} \varepsilon_{n+1} &= |1 - b\,x_{n+1}| = |1 - b\,x_n(2 - b\,x_n)| \\ &= |(1 - b\,x_n)^2| = \varepsilon_n^2 \end{aligned}$$

Properties of the Newton Method

- Asymptotically quadratic convergence
- Correction of round-off errors
- Final multiplication by *a* generates a round-off problem incompatible with the Standard IEEE-754

Iterative Division (Goldschmidt)

Define two sequences x_n and y_n such that

$$x_0 = a \quad \text{and} \quad y_0 = b$$

$$\frac{a}{b} = \frac{x_0}{y_0} = \frac{x_1}{y_1} = \frac{x_2}{y_2} = \ldots\ldots = \frac{x_n}{y_n} = \ldots\ldots$$

$$\delta = 1 - y_0 \quad \rightarrow \quad |\delta| < 1$$

$$y_1 = y_0(1+\delta) = 1 - \delta^2$$

Iterative division (Goldschmidt)

$$x_1 = x_0(1+\delta)$$

$$y_1 = y_0(1+\delta) = 1-\delta^2$$

$$x_2 = x_0(1+\delta)(1+\delta^2)$$

$$y_2 = y_0(1+\delta)(1+\delta^2) = 1-\delta^4$$

$$x_n = x_0(1+\delta)(1+\delta^2).....(1+\delta^{2^{n-1}})$$

$$y_n = y_0(1+\delta)(1+\delta^2).....(1+\delta^{2^{n-1}}) = 1-\delta^{2^n}$$

Iterative Division (Goldschmidt)

- The sequence of x_n tends to a/b
- The convergence is quadratic
- In its present form, this method is affected by roud-off errors

Modified Goldschmidt Algorithm (correction of round-off errors)

$$
\begin{aligned}
x_1 &= x_0\,(1+\delta_0) \\
y_1 &= y_0\,(1+\delta_0) = 1-\delta_0^2 \\
x_2 &= x_0\,(1+\delta_0)(1+\delta_1) \\
y_2 &= y_0\,(1+\delta_0)(1+\delta_1) \\
x_n &= x_0\,(1+\delta_0)(1+\delta_1).....(1+\delta_{n-1}) \\
y_n &= y_0\,(1+\delta_0)(1+\delta_1).....(1+\delta_{n-1})
\end{aligned}
$$

The Link between the Newton-Raphson and Goldschmidt Methods

By setting $\delta_n = 1 - b\,x_n$, the Newton method provides

$$\begin{aligned} x_1 &= x_0(1+\delta_0) \\ x_2 &= x_0(1+\delta_0)(1+\delta_1) \\ .... &= ..... \\ x_n &= x_0(1+\delta_0)(1+\delta_1)\,.....\,(1+\delta_{n-1}) \end{aligned}$$

The Link between the Newton-Raphson and Goldschmidt Methods

By expressing δ_n vs δ_0, the Newton method provides

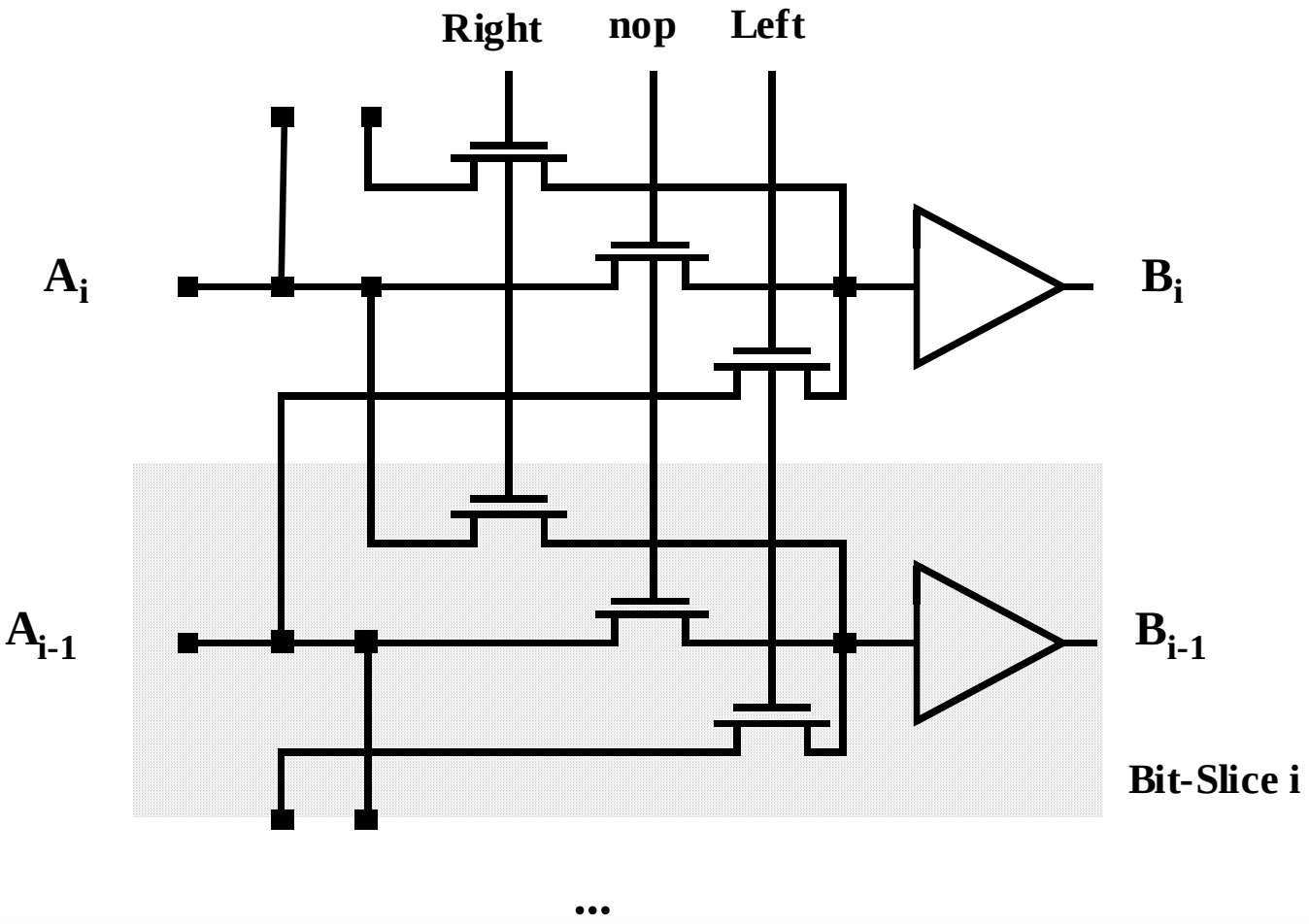
$$
\begin{aligned}
x_1 &= x_0(1+\delta_0) \\
x_2 &= x_0(1+\delta_0)(1+\delta_0^2) \\
.... &= ..... \\
x_n &= x_0(1+\delta_0)(1+\delta_0^2) ..... (1+\delta_0^{2^{n-1}})
\end{aligned}
$$

Comparison between Newton and Goldschmidt methods

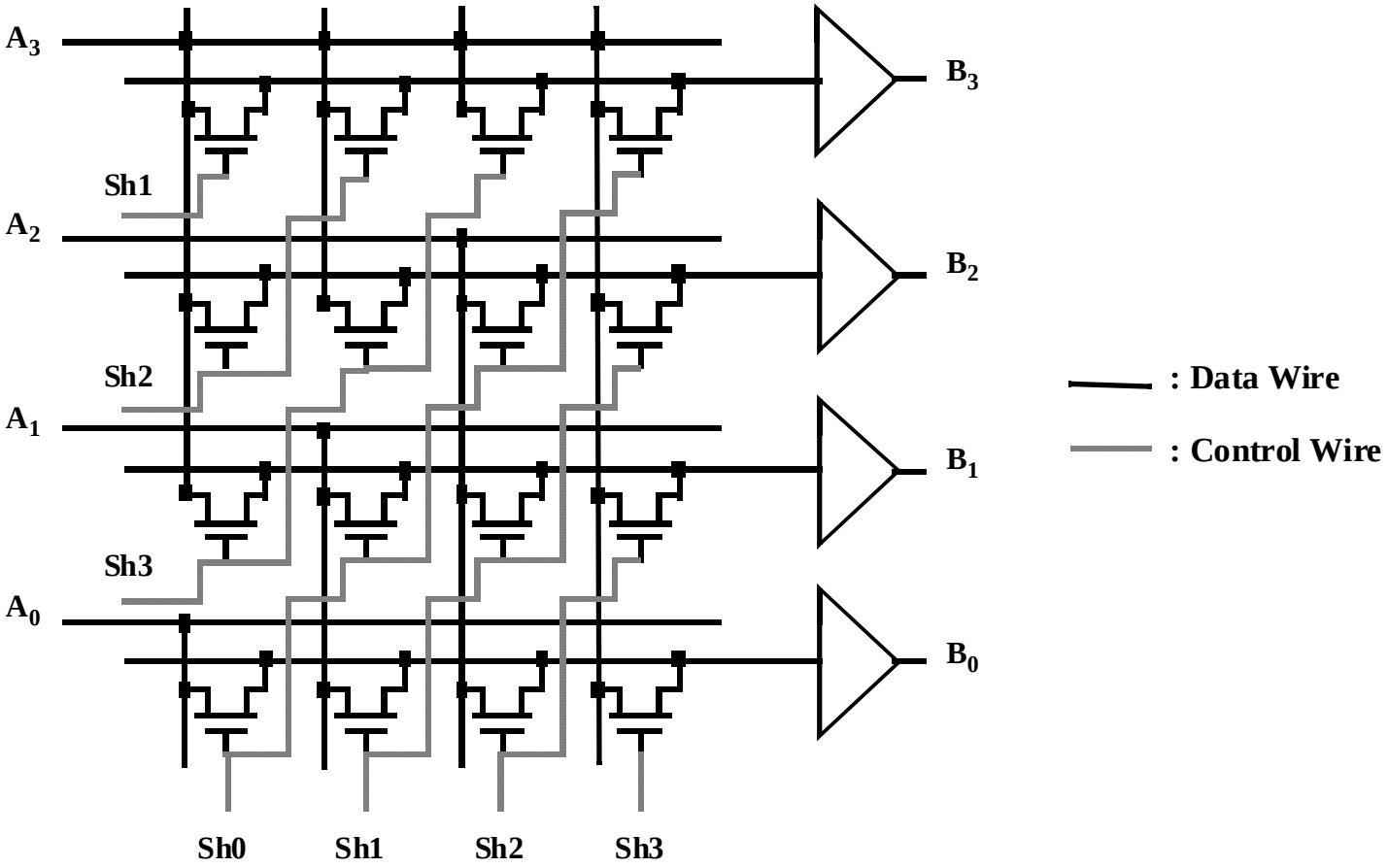
- The Newton and Goldschmidt methods are essentially equivalent;
- Both methods exhibit an asymptotically quadratic convergence;
- Both methods are able to correct round-off errors;
- The Goldschmidt methods directly computes the *a*/*b* ratio.

Shifters

The Binary Shifter

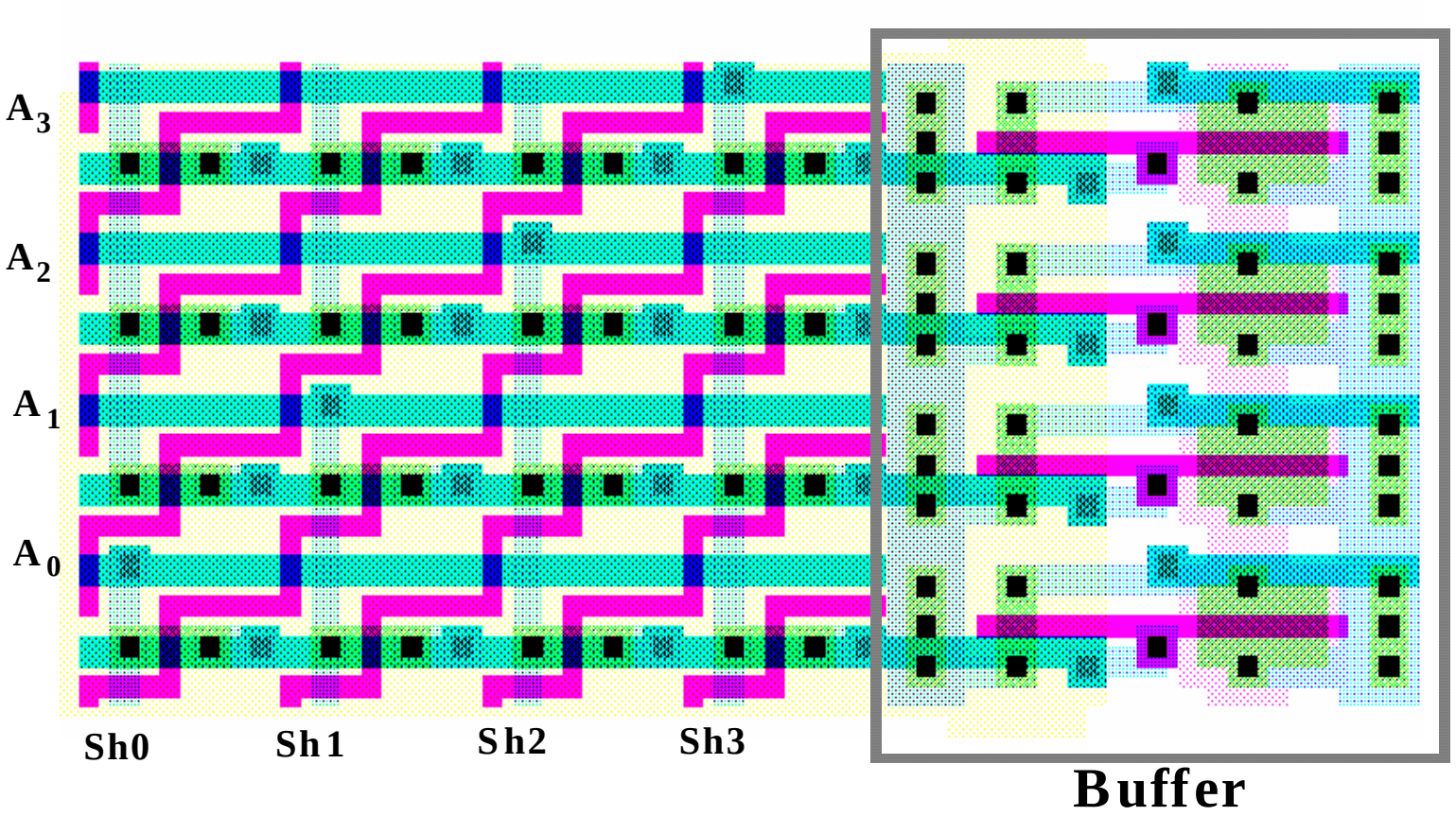


The Barrel Shifter



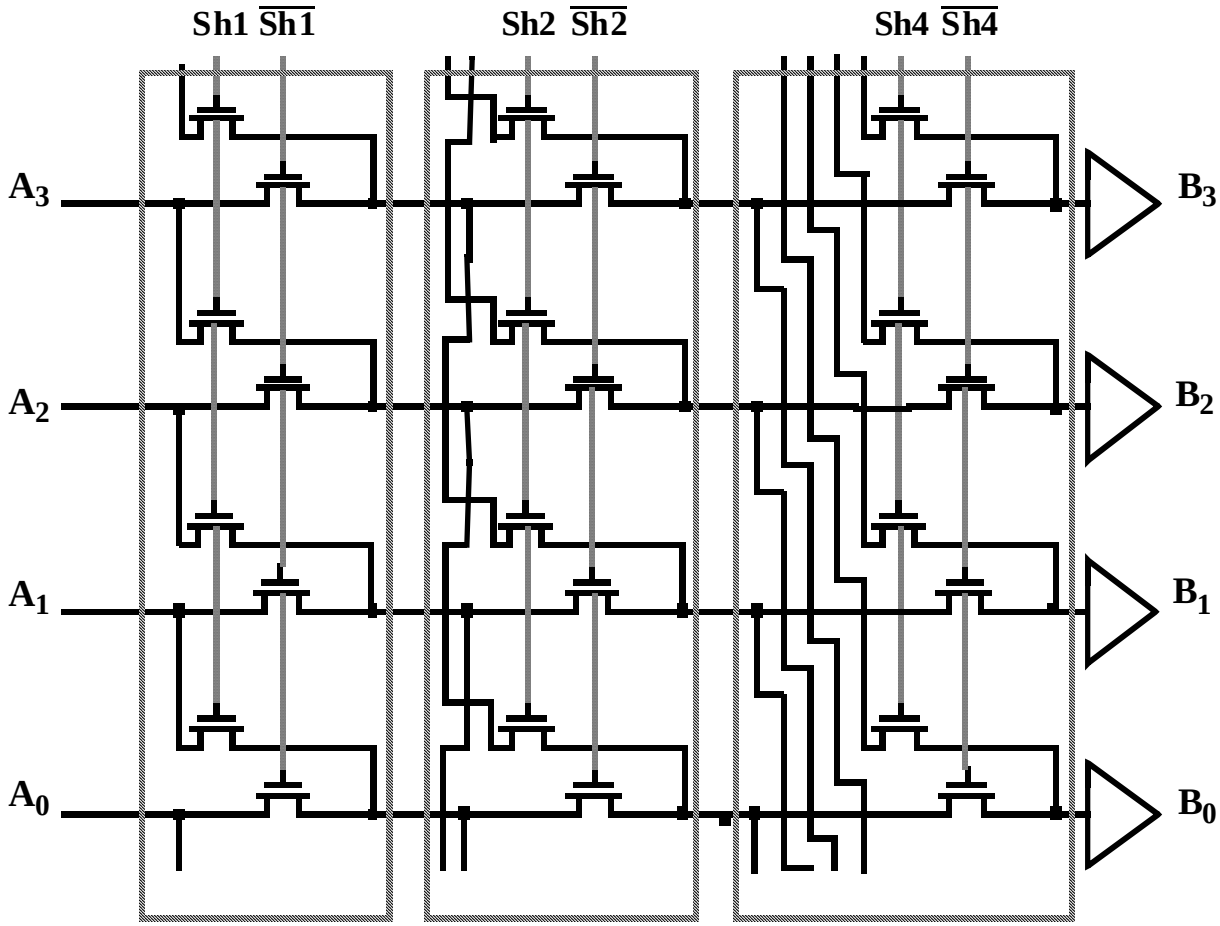
Area Dominated by Wiring

4x4 barrel shifter

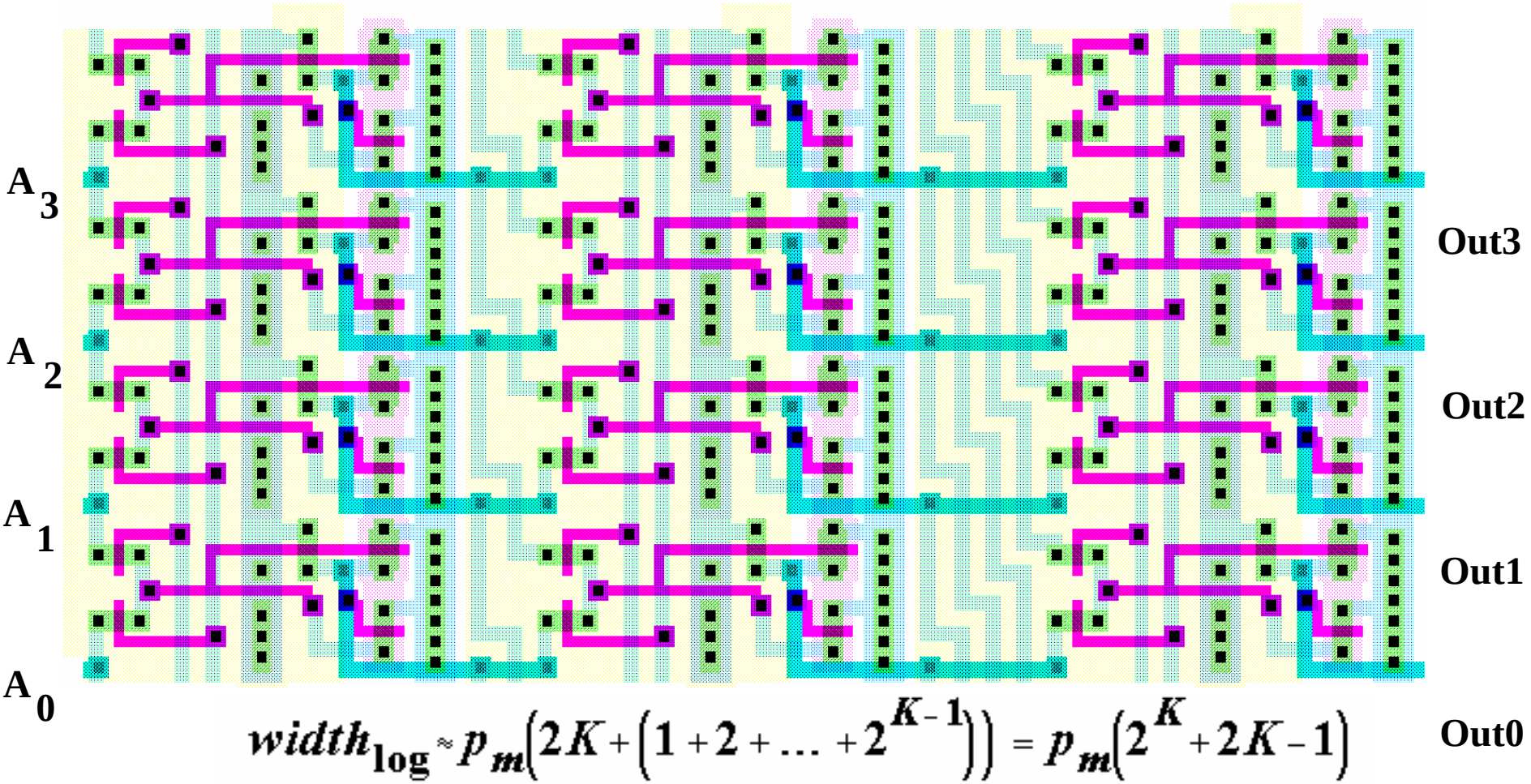


$$Width_{barrel} \sim 2\, p_m\, M$$

Logarithmic Shifter



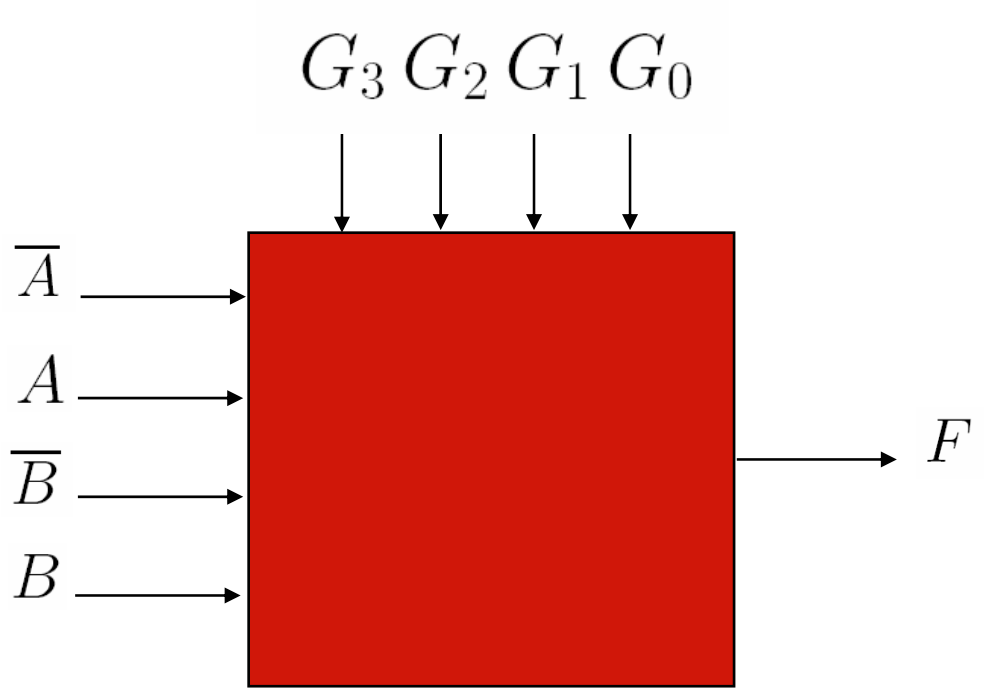
0-7 bit Logarithmic Shifter



$$width_{\log} \approx p_m\left(2K + \left(1 + 2 + \ldots + 2^{K-1}\right)\right) = p_m\left(2^K + 2K - 1\right)$$

ALUs

Two-bit MUX



$$F(A,B) = G_0 \cdot \overline{A}\,\overline{B} + G_1 \cdot \overline{A}\,B + G_2 \cdot A\,\overline{B} + G_3 \cdot A\,B$$

Two-bit MUX Truth Table

A	B	F
0	0	G_0
0	1	G_1
1	0	G_2
1	1	G_3

$$F(A,B) = G_0 \cdot \overline{A}\,\overline{B} + G_1 \cdot \overline{A}\,B + G_2 \cdot A\,\overline{B} + G_3 \cdot A\,B$$

Two-bit Selector Truth Table

G_3	G_2	G_1	G_0	$F(A,B)$	G_3	G_2	G_1	G_0	$F(A,B)$
0	0	0	0	0	1	0	0	0	$A \cdot B = \overline{\overline{A} + \overline{B}}$
0	0	0	1	$\overline{A} \cdot \overline{B} = \overline{A + B}$	1	0	0	1	$A \cdot B + \overline{A} \cdot \overline{B}$
0	0	1	0	$\overline{A} \cdot B = \overline{A + \overline{B}}$	1	0	1	0	B
0	0	1	1	$\overline{A}$	1	0	1	1	$\overline{A \cdot \overline{B}} = \overline{A} + B$
0	1	0	0	$A \cdot \overline{B} = \overline{\overline{A} + B}$	1	1	0	0	A
0	1	0	1	$\overline{B}$	1	1	0	1	$\overline{\overline{A} \cdot B} = A + \overline{B}$
0	1	1	0	$A \cdot \overline{B} + \overline{A} \cdot B$	1	1	1	0	$\overline{\overline{A} \cdot \overline{B}} = A + B$
0	1	1	1	$\overline{A \cdot B} = \overline{A} + \overline{B}$	1	1	1	1	1

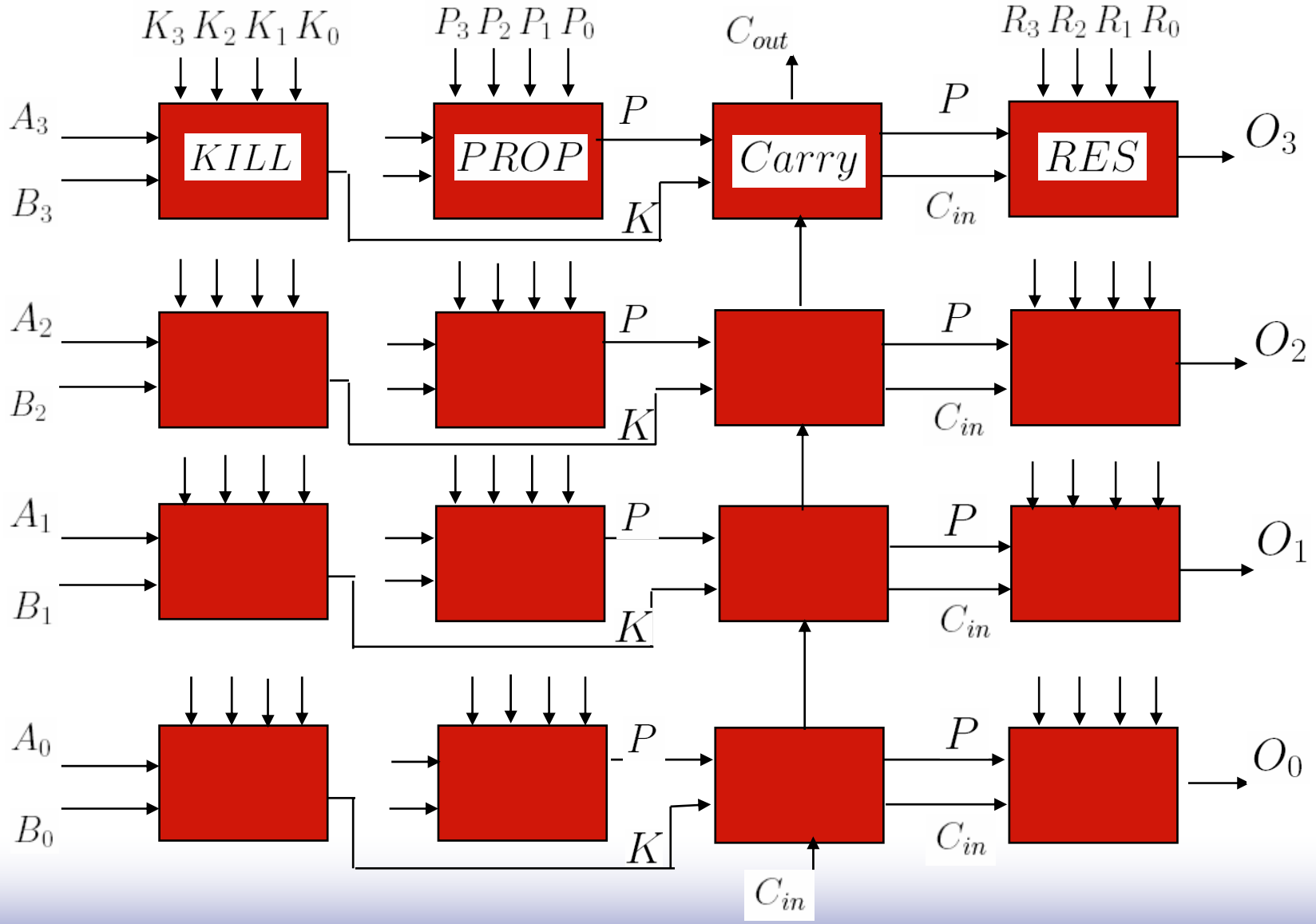
$$F(A,B) = G_0 \cdot \overline{A}\,\overline{B} + G_1 \cdot \overline{A}\,B + G_2 \cdot A\,\overline{B} + G_3 \cdot A\,B$$

Carry-chain Truth Table

C_{in}	$KILL$	$PROP$	C_{out}
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	N.A.
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	N.A.

$$C_{out} = \overline{K + P\overline{C}_{in}} = \overline{K}\left(\overline{P} + C_{in}\right)$$

ALU block diagram (Mead-Conway)



ALU Operations

Function	$KILL$	$PROP$	RES	C_{in}	CF
$A + B$	1	6	6	0	0
$A + B + C_{in}$	1	6	6	1	0
$A - B$	2	9	6	2	0
$B - A$	4	9	6	2	0
$A - B - C_{in}$	2	9	6	1	0
$B - A - C_{in}$	4	9	6	1	0

ALU Operations

Operation	$KILL$	$PROP$	RES	C_{in}	CS
$A+1$	3	12	6	2	0
$B+1$	5	10	6	2	0
$A-1$	12	3	9	2	0
$B-1$	10	5	9	2	0
$-A$	12	3	6	2	0
$-B$	10	5	6	2	0

ALU Operations

Operation	$KILL$	$PROP$	RES	C_{in}	CS
$A.and.B$	0	8	12	0	0
$A.or.B$	0	14	12	0	0
$A.xor.B$	0	6	12	0	0
$\overline{A}$	0	3	12	0	0
$\overline{B}$	0	5	12	0	0
A	0	12	12	0	0
B	0	10	12	0	0

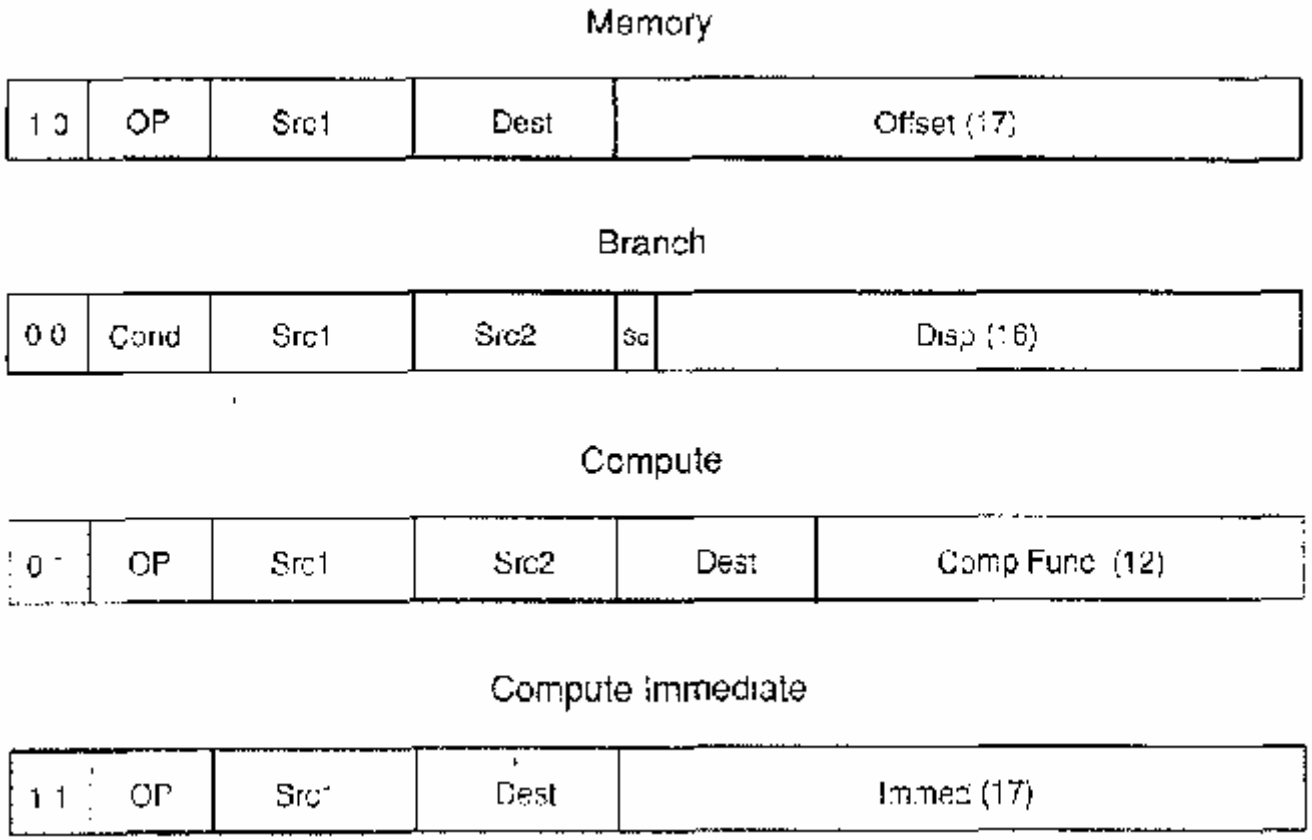
ALU Operations

Function	$KILL$	$PROP$	RES	C_{in}	CF
Multiply	1	14	14	0	1
Divide	3	15	15	0	2
A/O	0	14	12	0	3
Mask	10	5	8	2	0
SHL A	3	0	10	0	0
Zero	0	0	0	0	0

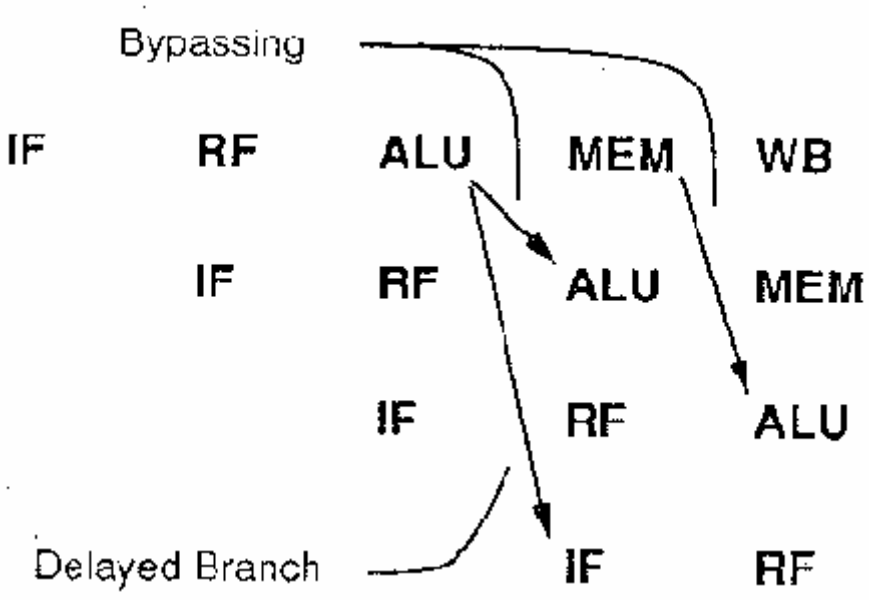
ALU Operations

Operation	Select	Flag bit	$KILL$	$PROP$	RES
Uncond.	0	×	- - - -	- - - -	- - - -
Multiply	1	0	- - - 0	- - 0 -	- - 0 -
Multiply	1	1	- - - -	0 - - -	0 - - -
Divide	2	0	- - - 0	- 0 0 -	0 - - 0
Divide	2	1	- - 0 -	0 - - 0	0 - - 0
AND/OR	3	0	- - - -	- - - -	- - - -
AND/OR	3	1	- - - -	- 0 0 -	- - - -

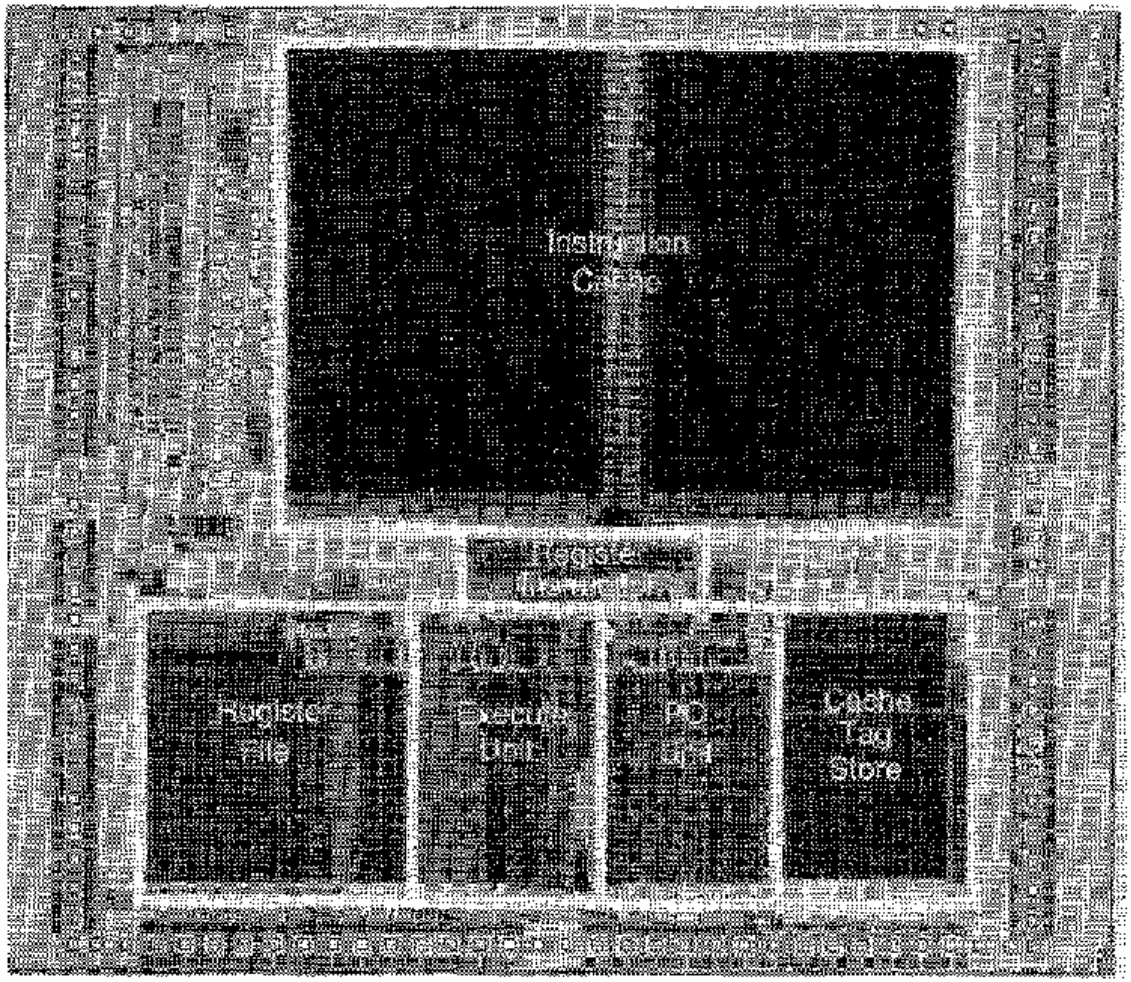
MIPS-X Instruction Format



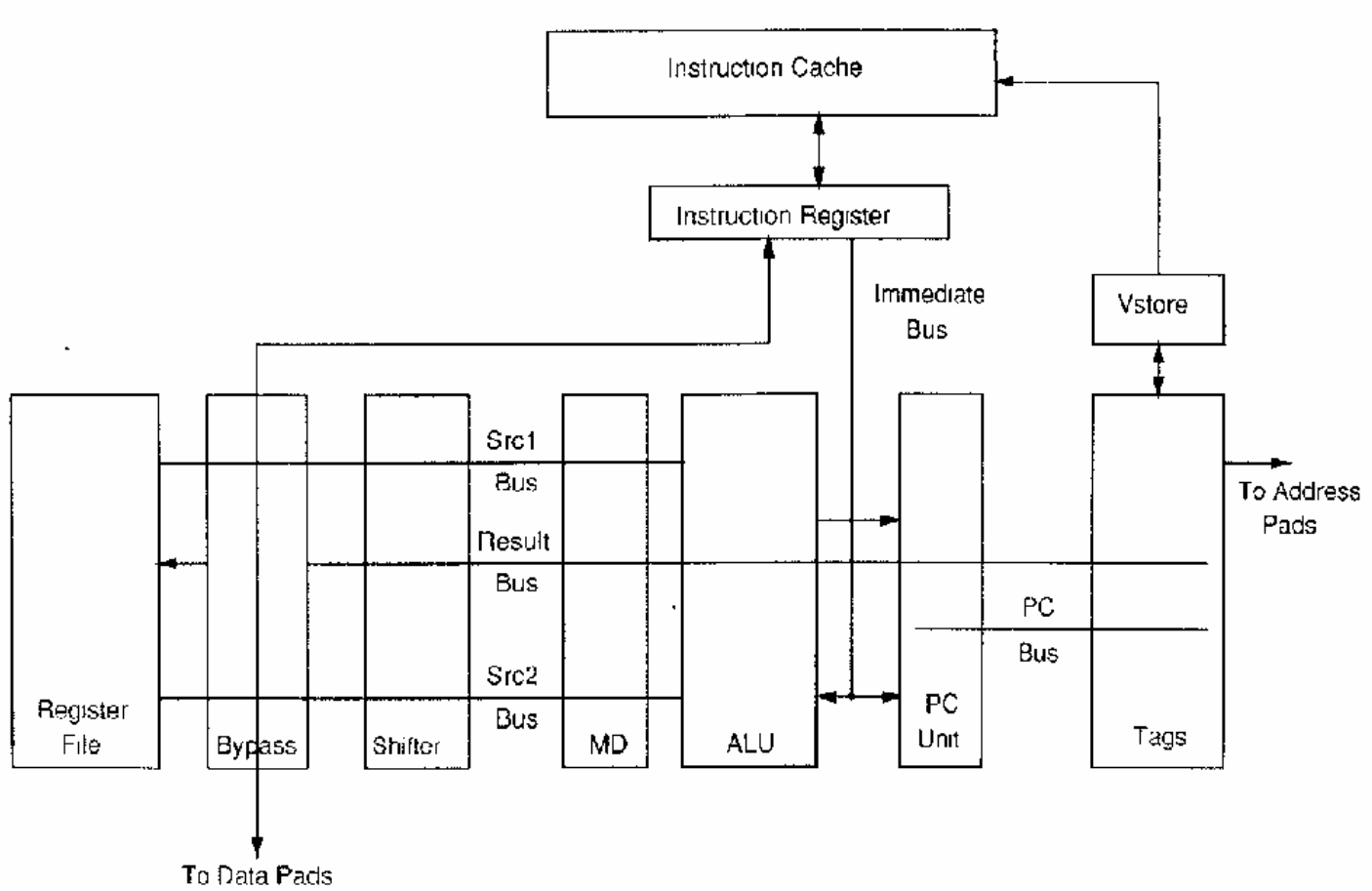
Pipeline dependencies in MIPS-X



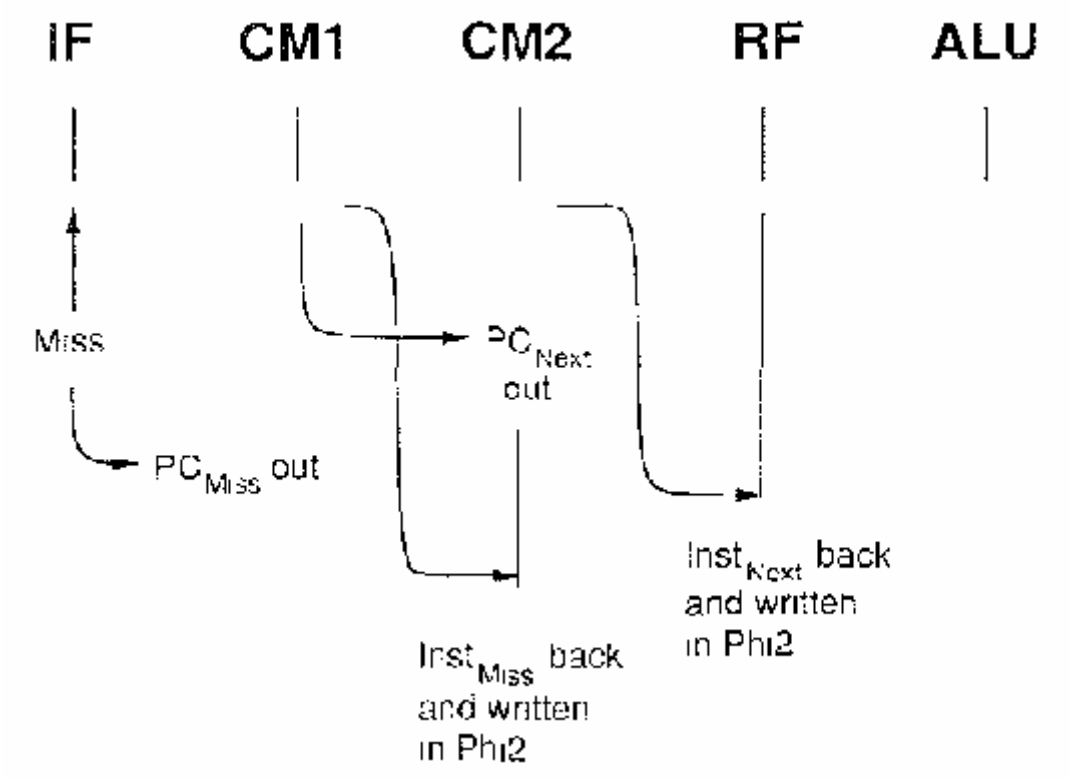
Die Photo of MIPS-X



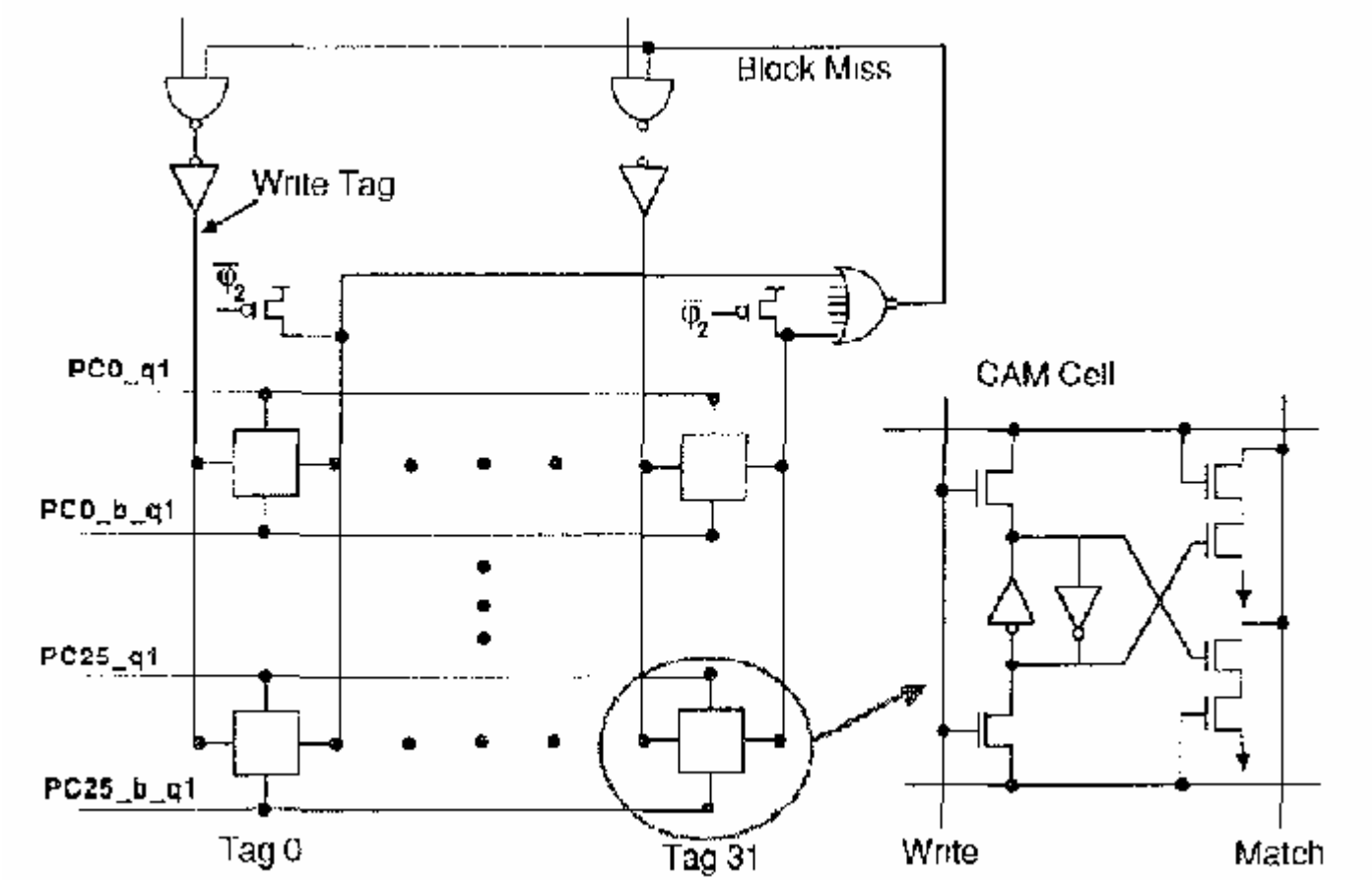
MIPS-X Architecture



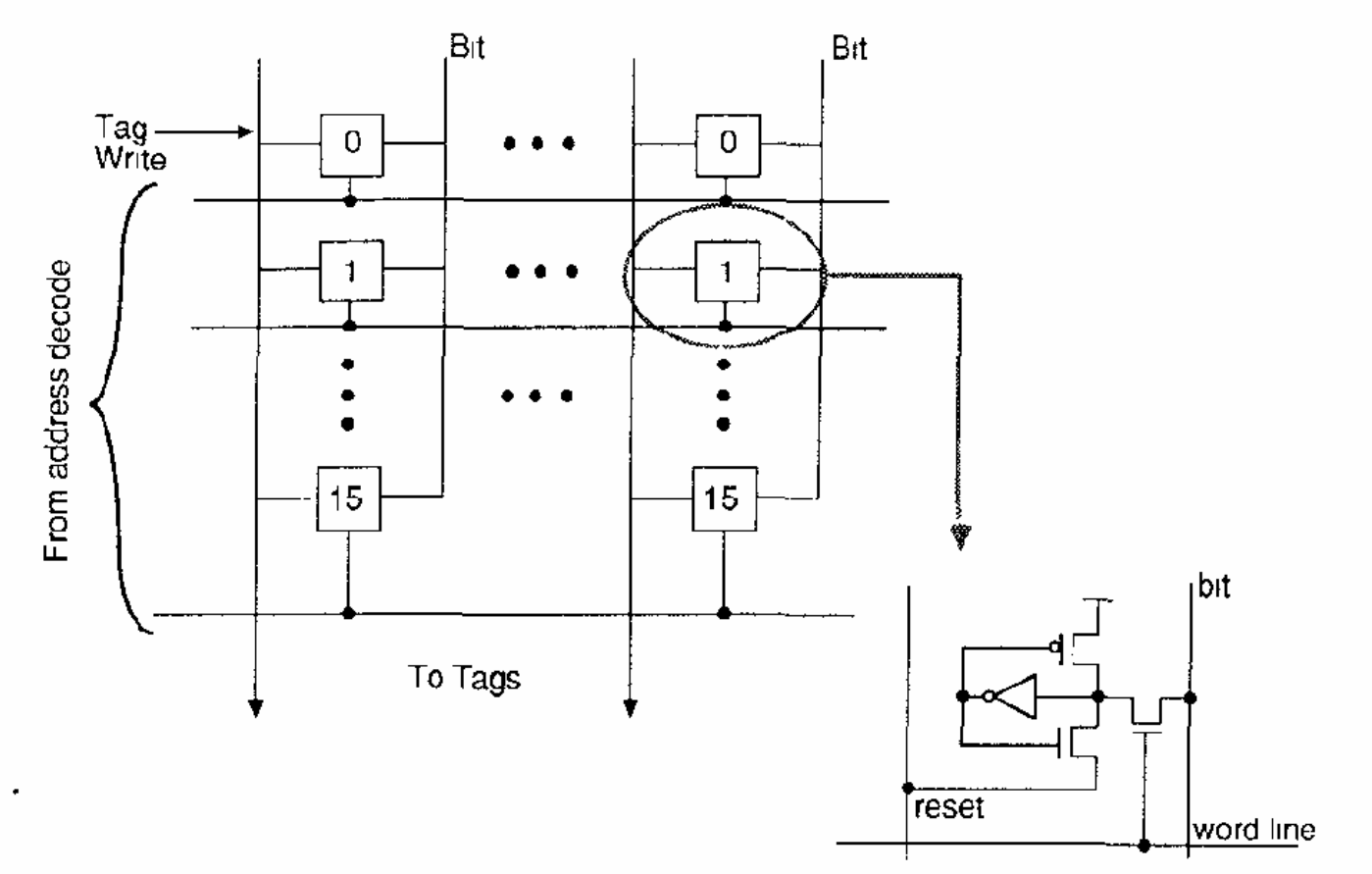
MIPS-X Instruction Cache-miss timing



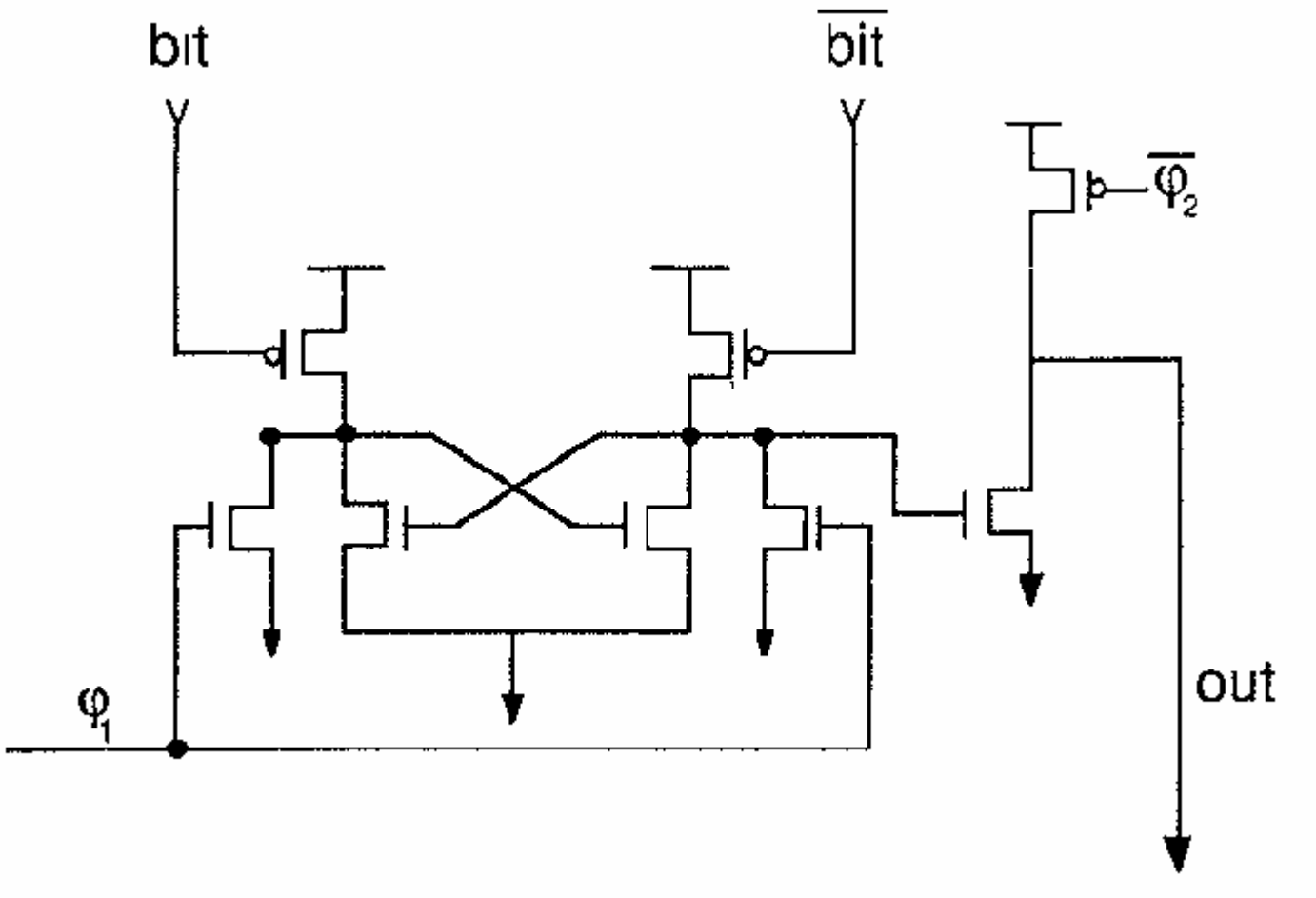
MIPS-X Tag Memory



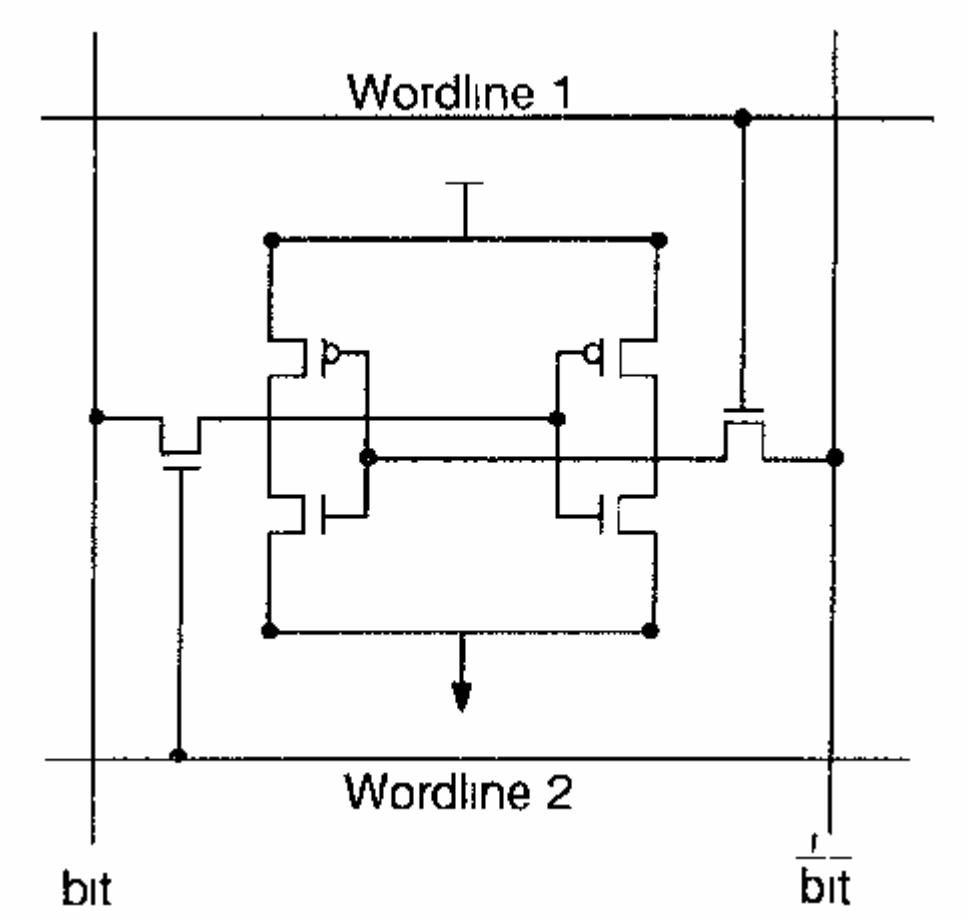
MIPS-X Valid Store Array



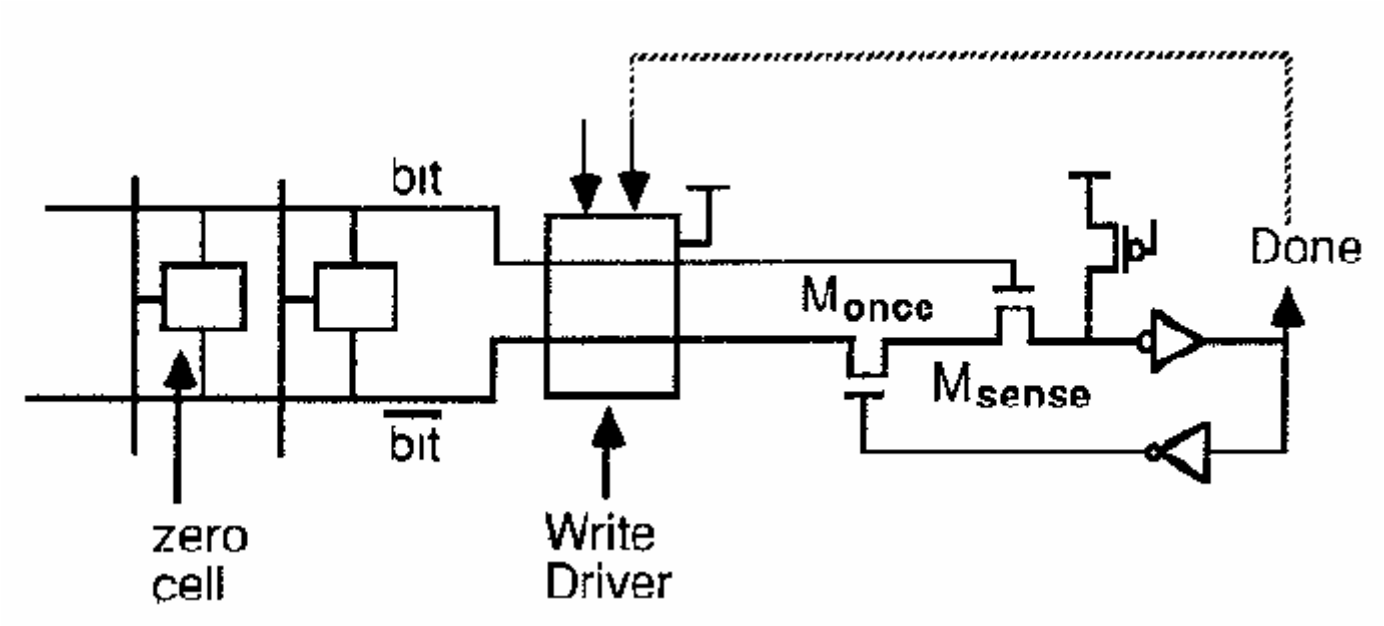
RAM Sense Amplifier



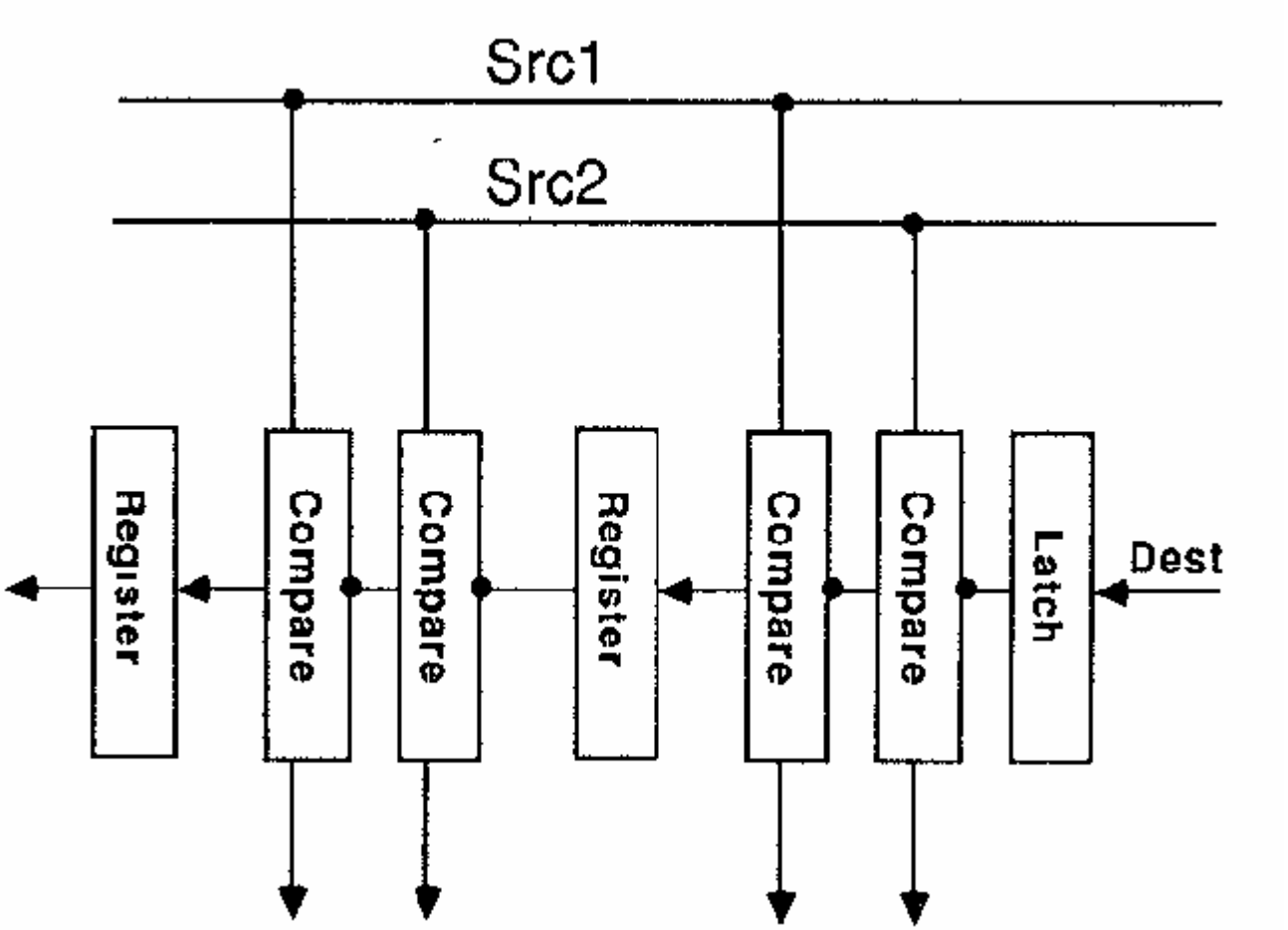
CMOS Dual-port Register Cell



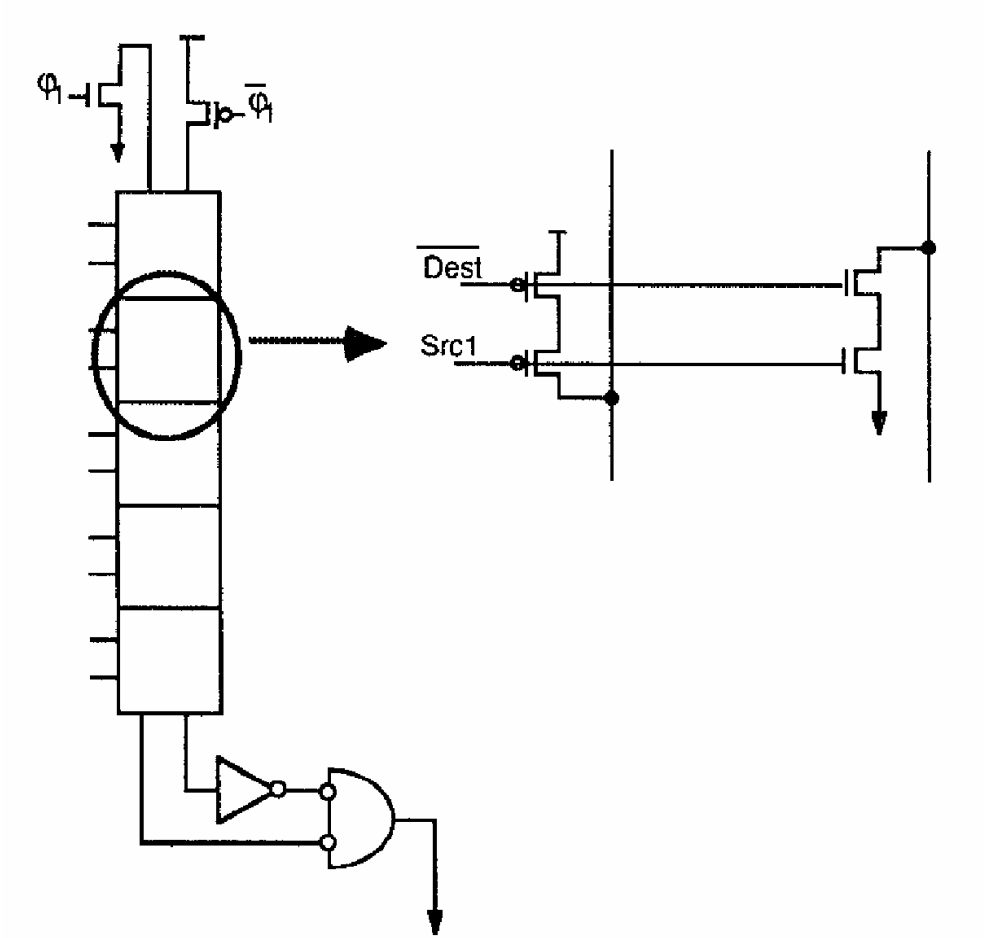
Self-timed bit-line write circuit



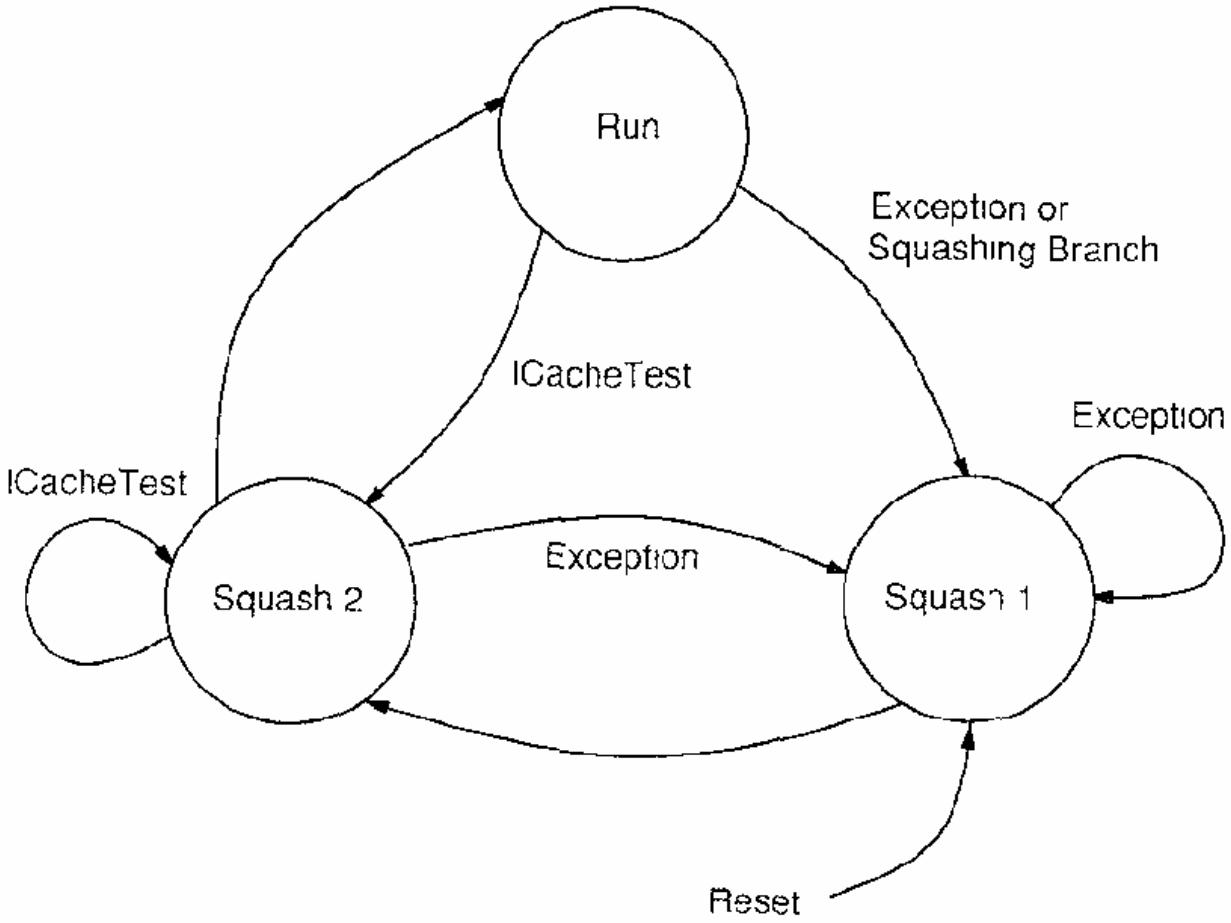
Register bypass logic



Schematic of comparator circuit



Squash FSM


Run
Exception or
Squashing Branch
ICacheTest
ICacheTest
Exception
Squash 2
Exception
Squash 1
Reset

Cache-miss FSM