Digital IC Lecture *Adders*

Deyu Tu, PhD

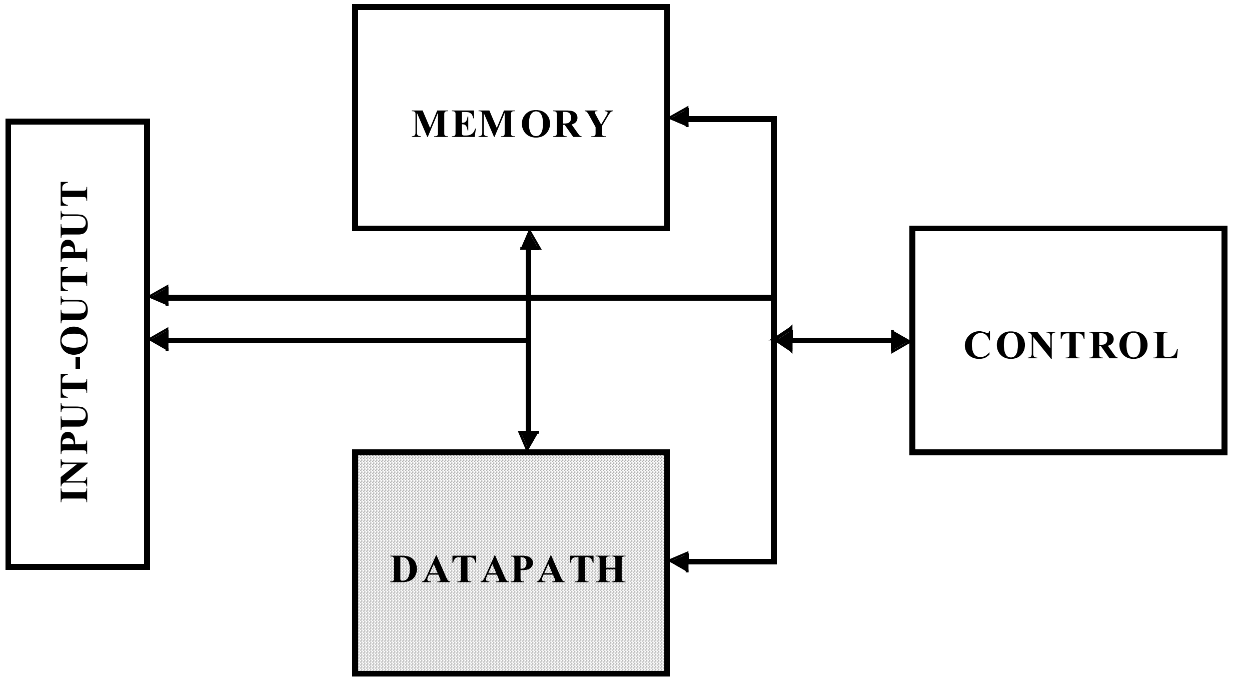
deyu.tu@liu.se
013-285851
Laboratory of Organic Electronics, Campus Norrköping

LINKÖPING UNIVERSITY

Outline

- Introduction
- The Binary Adder
- Single-bit Adder Design
- N-bit Adder Design

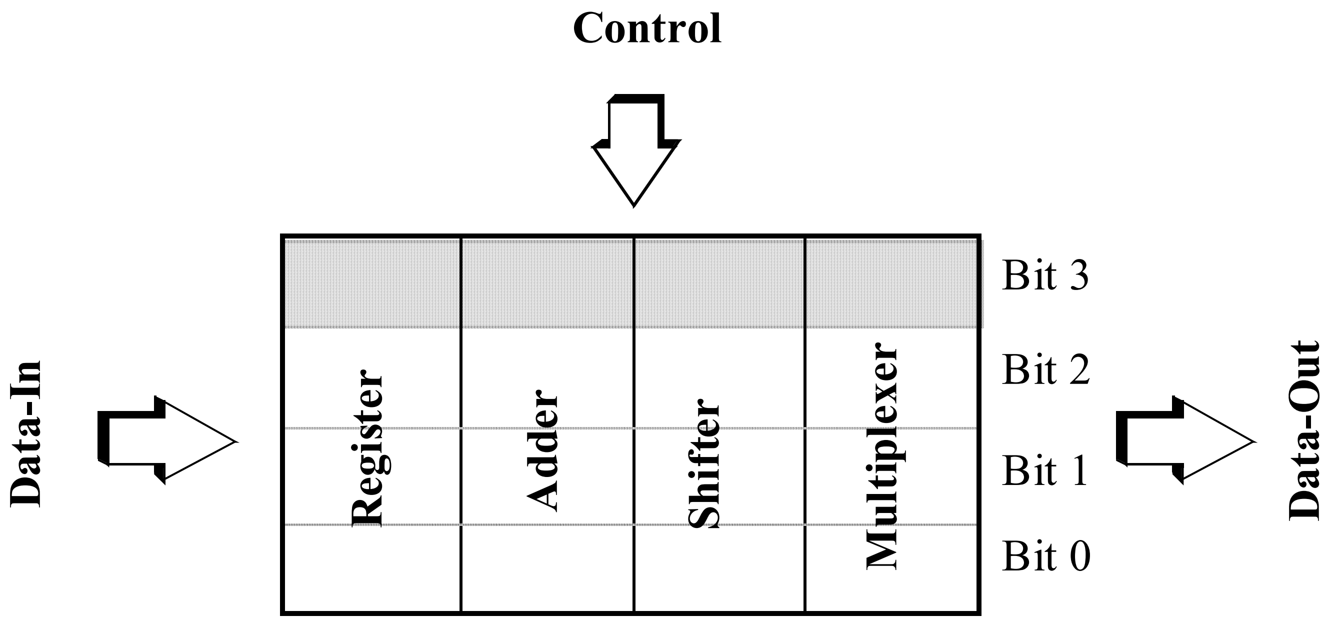
A Generic Digital Processor



Building Blocks

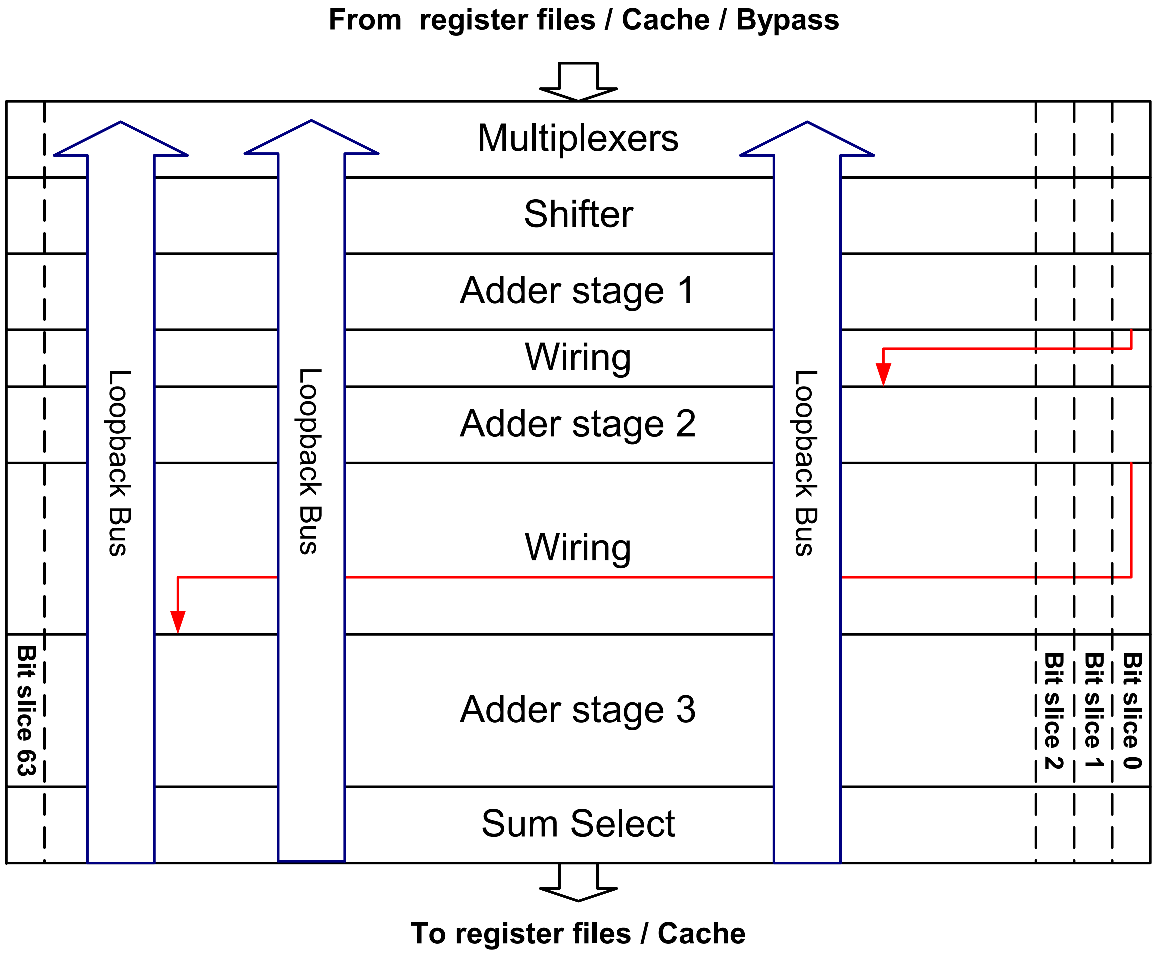
- Arithmetic unit
  - Bit-siliced datapath (adder, multiplier, shifter, comparator, etc.)
- Memory
  - RAM, ROM, Buffer, Shift registers
- Control
  - Finite state machine (PLA, random logic), Counters
- Interconnect
  - Switches, Arbiters, Bus

Bit-sliced Design



Tile identical processing elements

Bit-sliced Datapath



Half Adder

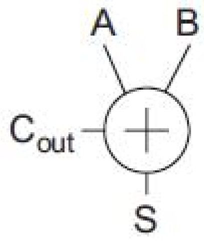
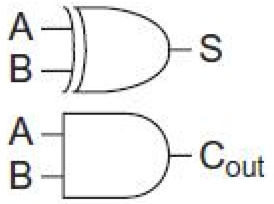


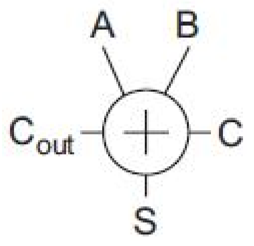
TABLE 11.1 Truth table for half adder

A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$
$$C_{out} = A \cdot B$$

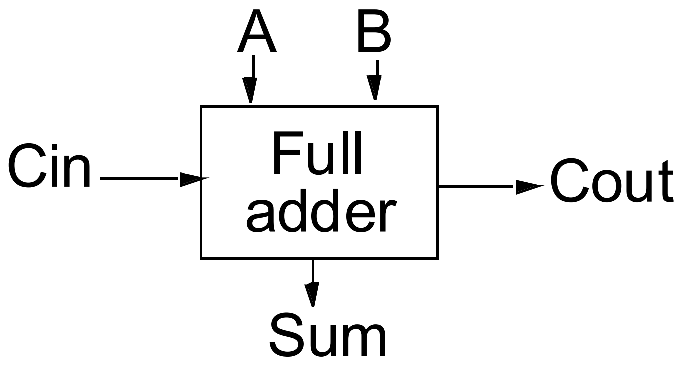


Full Adder



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



$$\mathbf{S} = \mathbf{A} \oplus \mathbf{B} \oplus \mathbf{C_i}$$

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

$$\mathbf{C_o} = \mathbf{AB + BC_i + AC_i}$$

Express *S* and C_0 as a function of *G, D, P*

Three new variables which ONLY depend on A, B

Generate (G) = AB

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

Propagate (P) = A ⊕ B

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

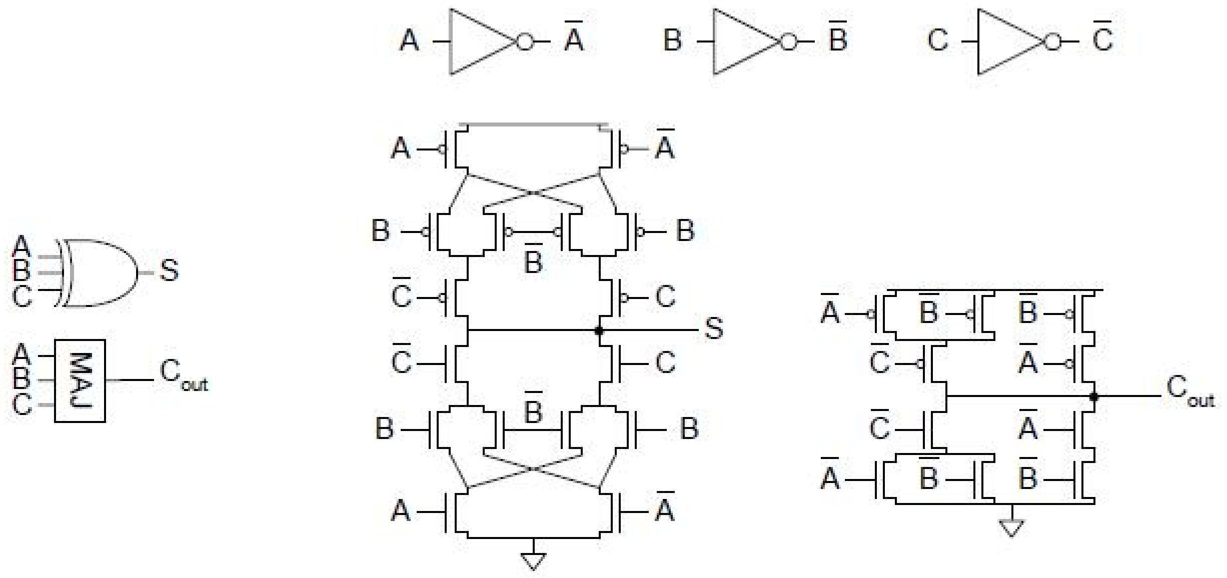
$$S(G, P) = P \oplus C_i$$

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

Full Adder Design I

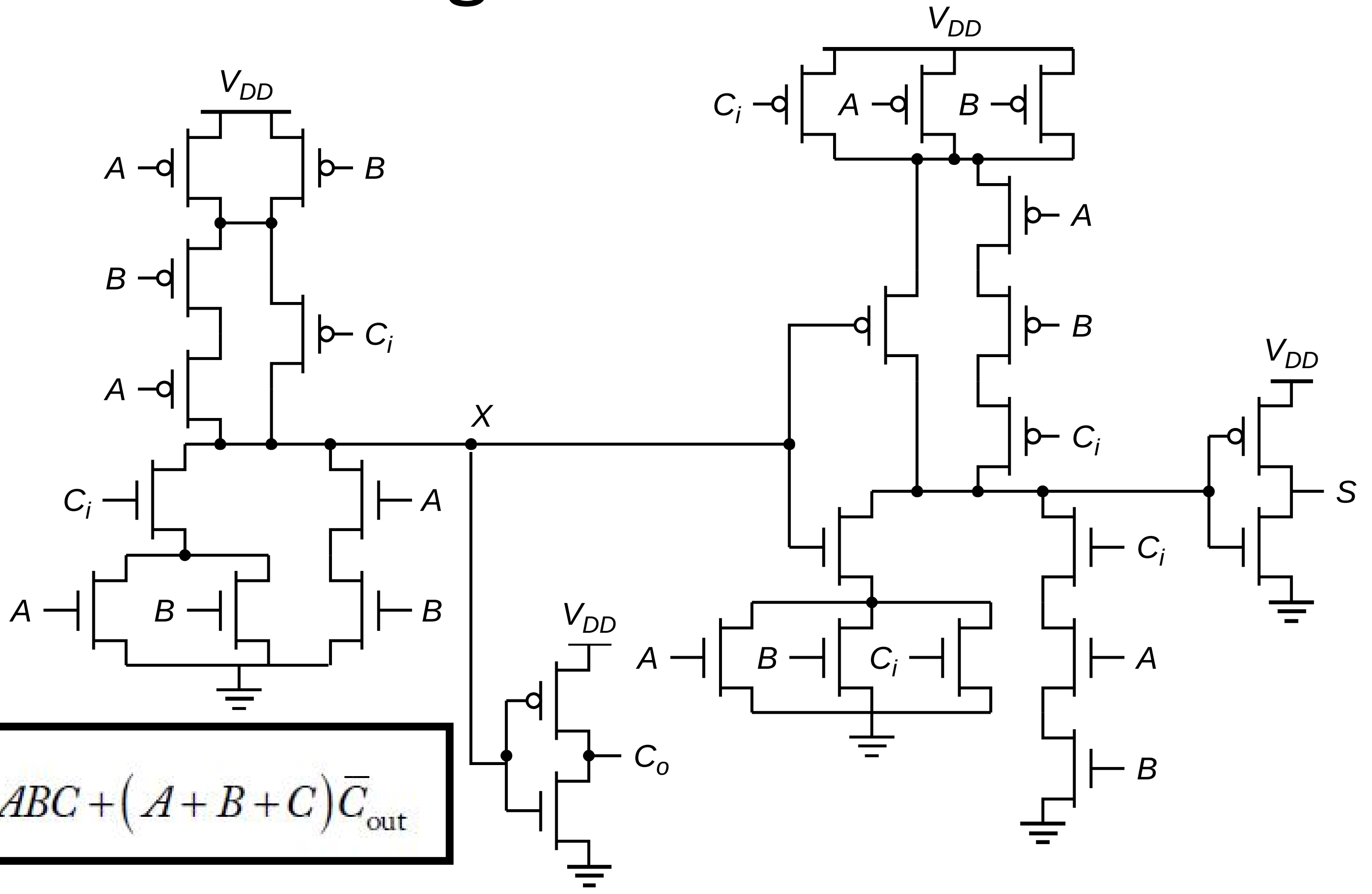
$$S = A \oplus B \oplus C$$

$$C_{\text{out}} = MAJ(A, B, C)$$



Complementary static CMOS, 32 Transistors

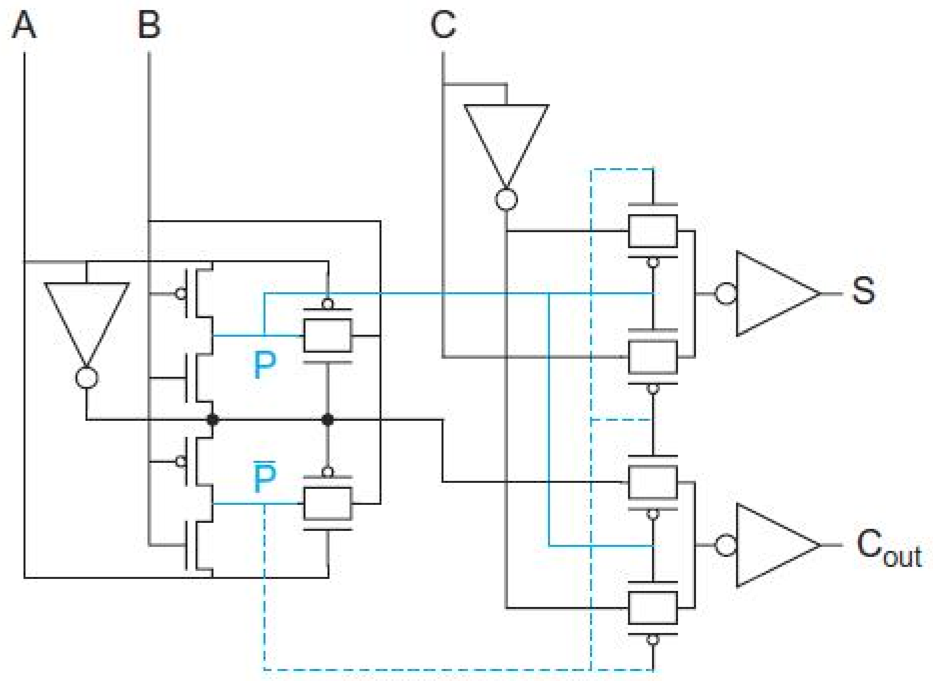
Full Adder Design II



$$S = ABC + (A + B + C)\bar{C}_{\text{out}}$$

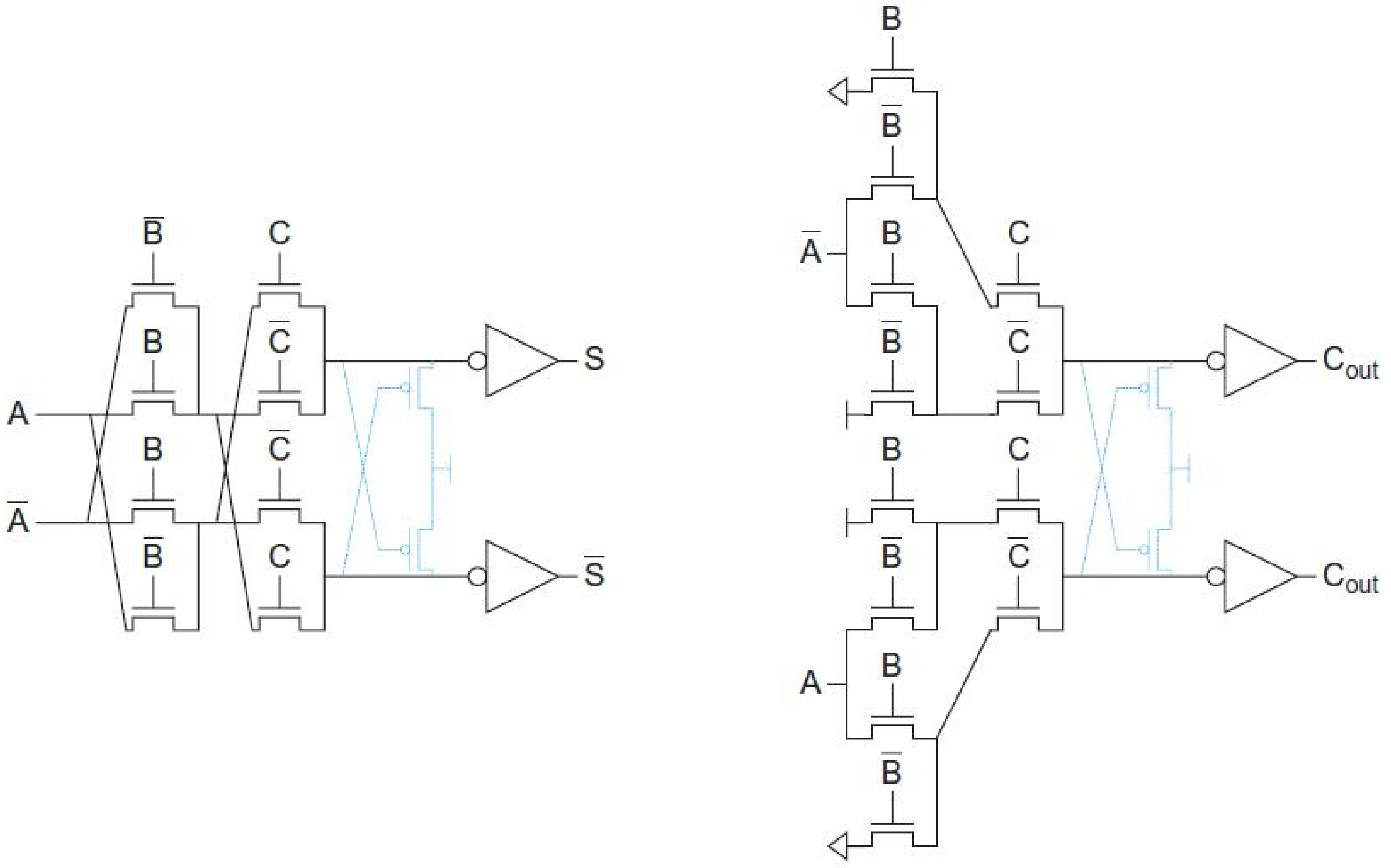
Complementary static CMOS, 28 Transistors

Full Adder Design III



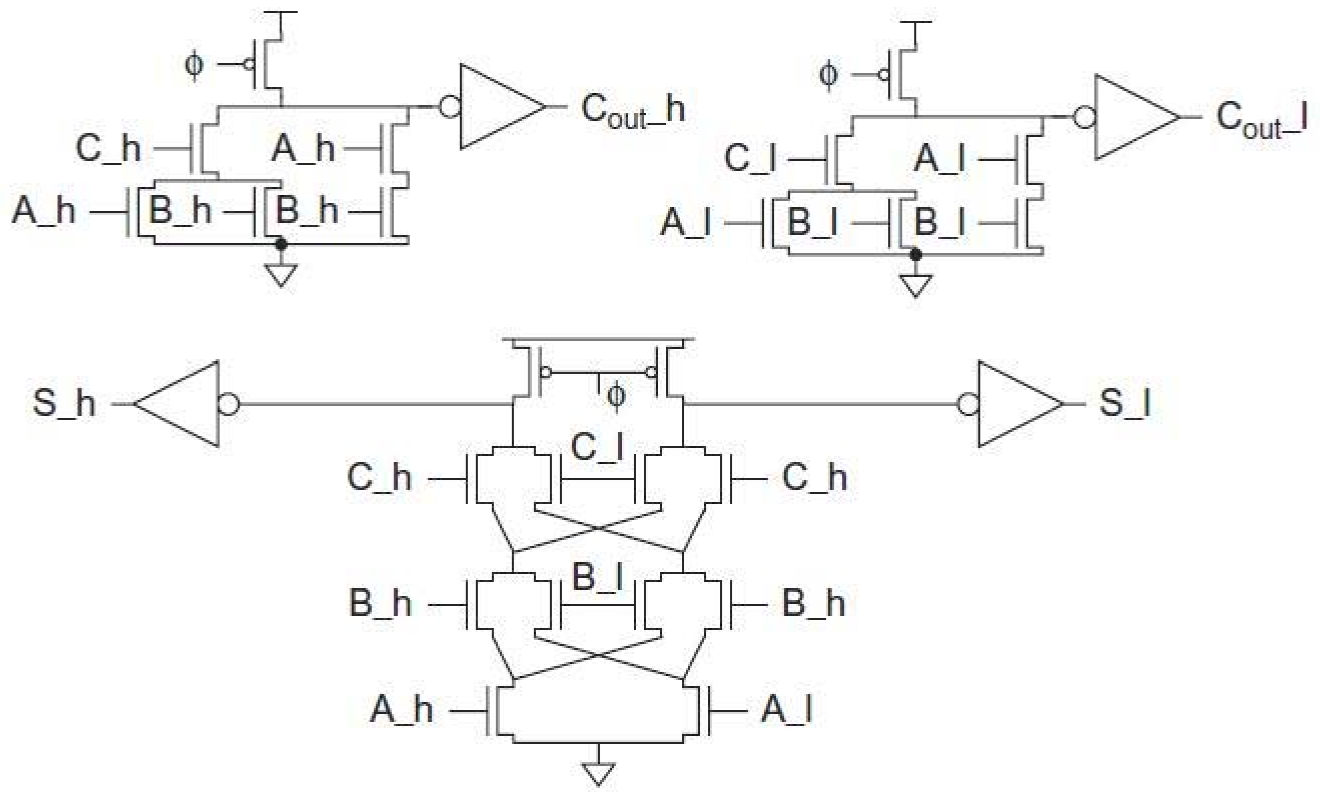
Transmission gate, 24 Transistors

Full Adder Design IV



Complementary Pass-Transistor Logic, 32 T

Full Adder Design V



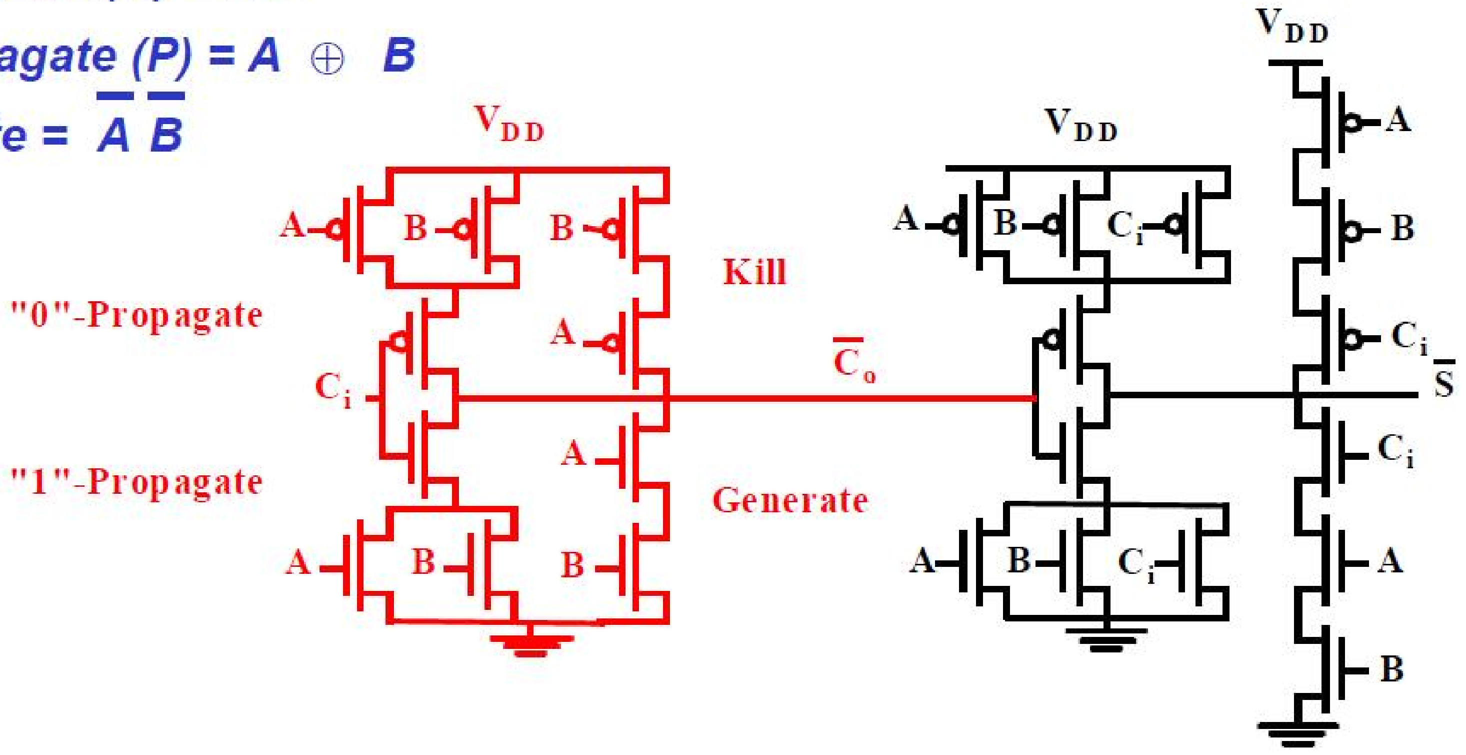
Dual-rail Domino

Full Adder Design VI

Generate (G) = AB

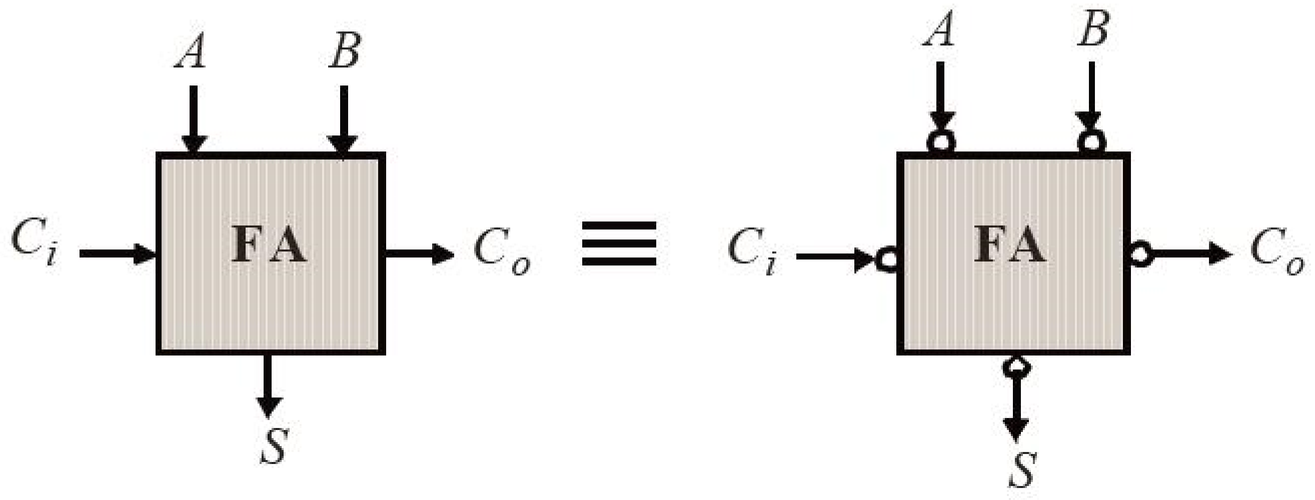
Propagate (P) = A $\oplus$ B

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



Mirror Adder, 24 Transistors

Inversion Property

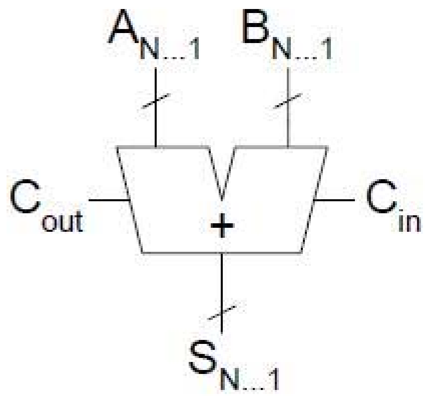


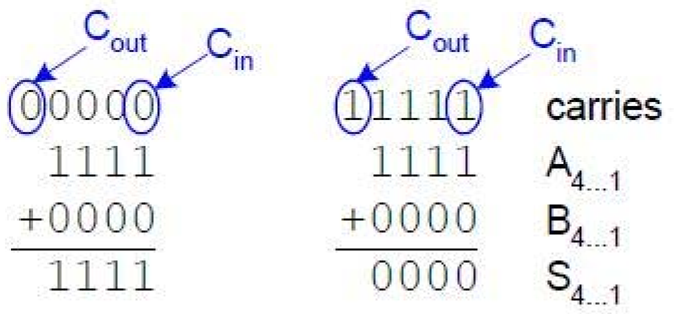
$$\bar{S}(A,B,C_i) = S(\bar{A},\bar{B},\overline{C_i})$$

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

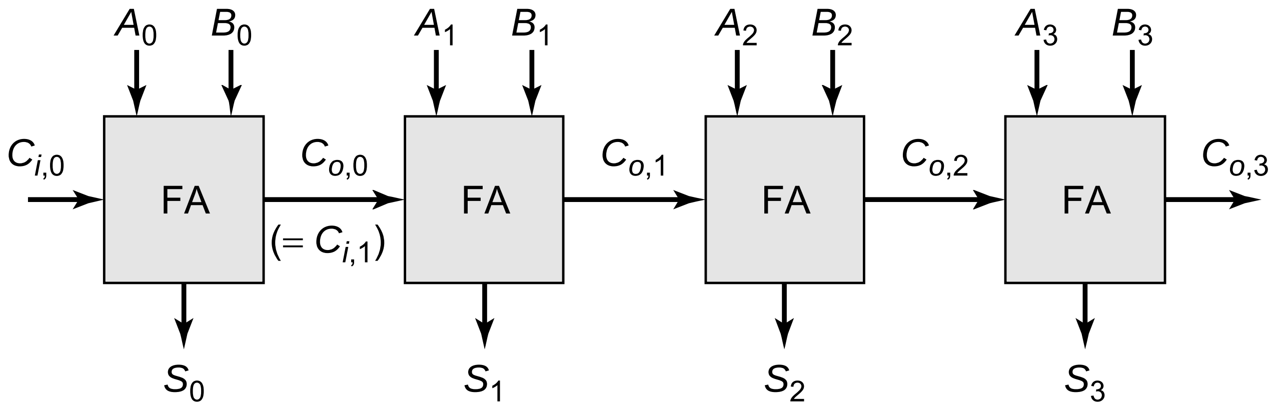
Carry Propagate Adders

- N-bit Adder





The Ripple-Carry Adder



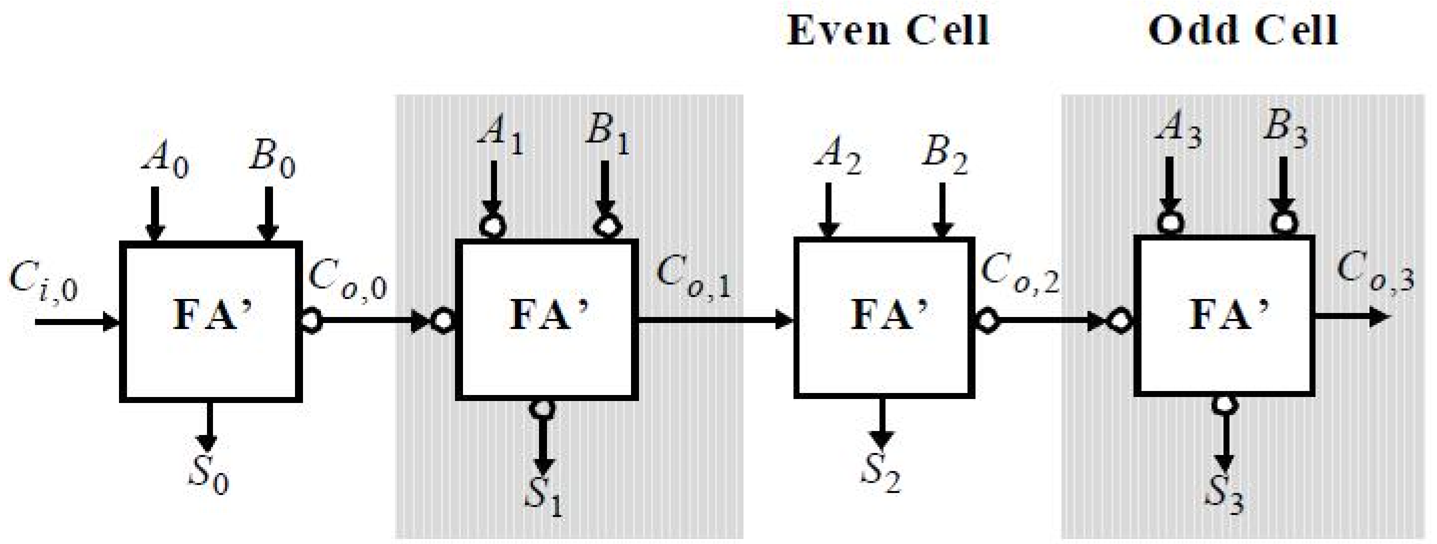
Worst case delay linear with the number of bits

$$t_d = O(N)$$

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

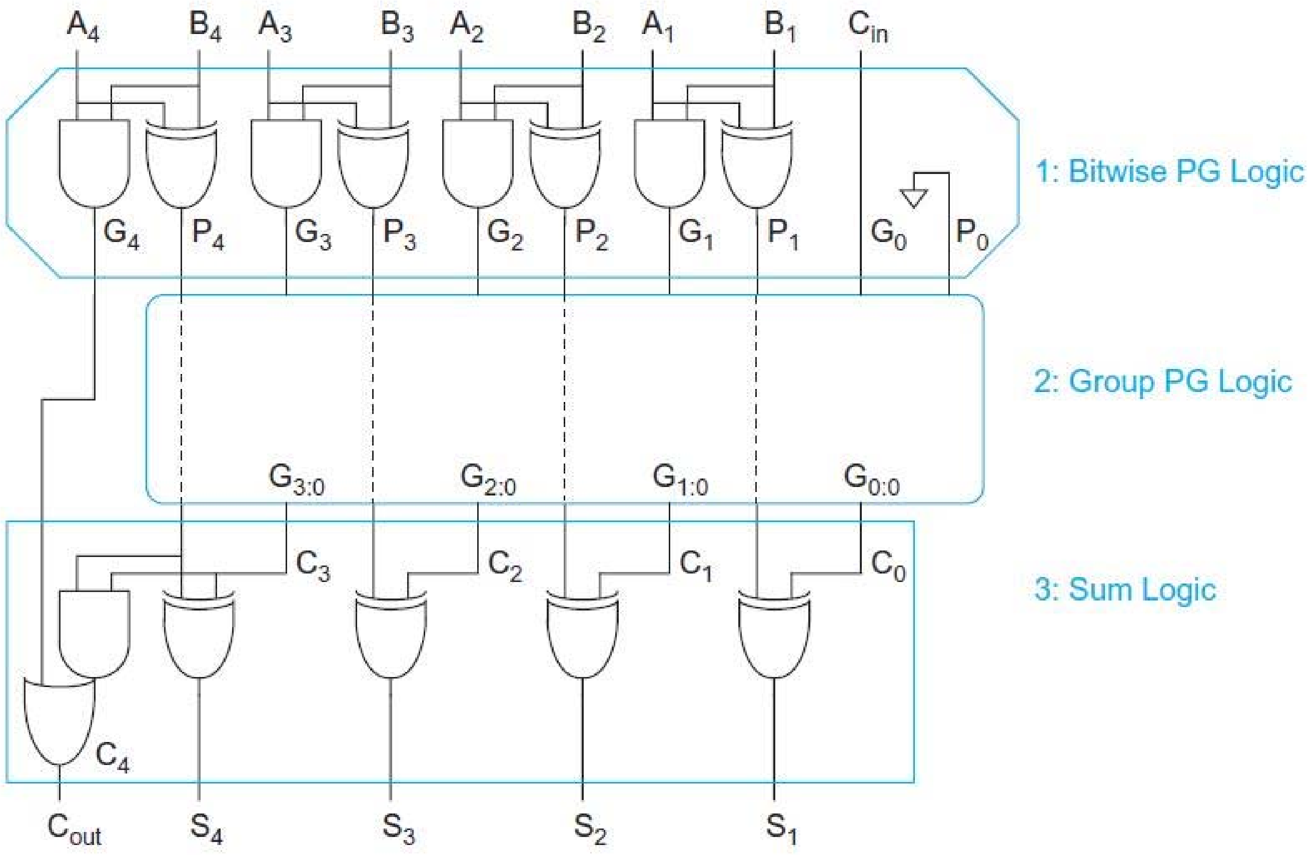
Goal: Make the fastest possible carry path circuit

Minimize Critical Path

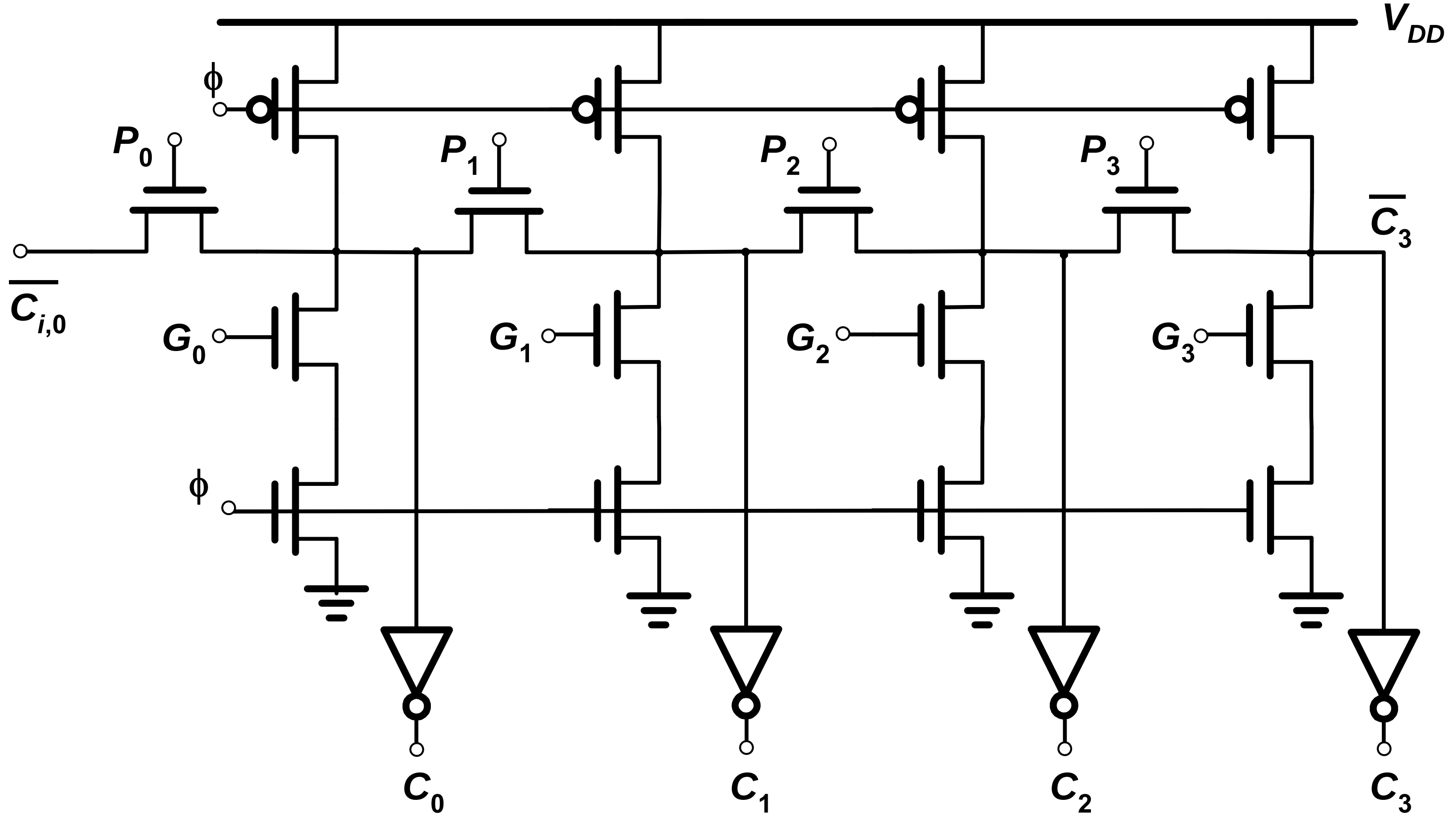


Exploit Inversion Property
(2 different cells needed)

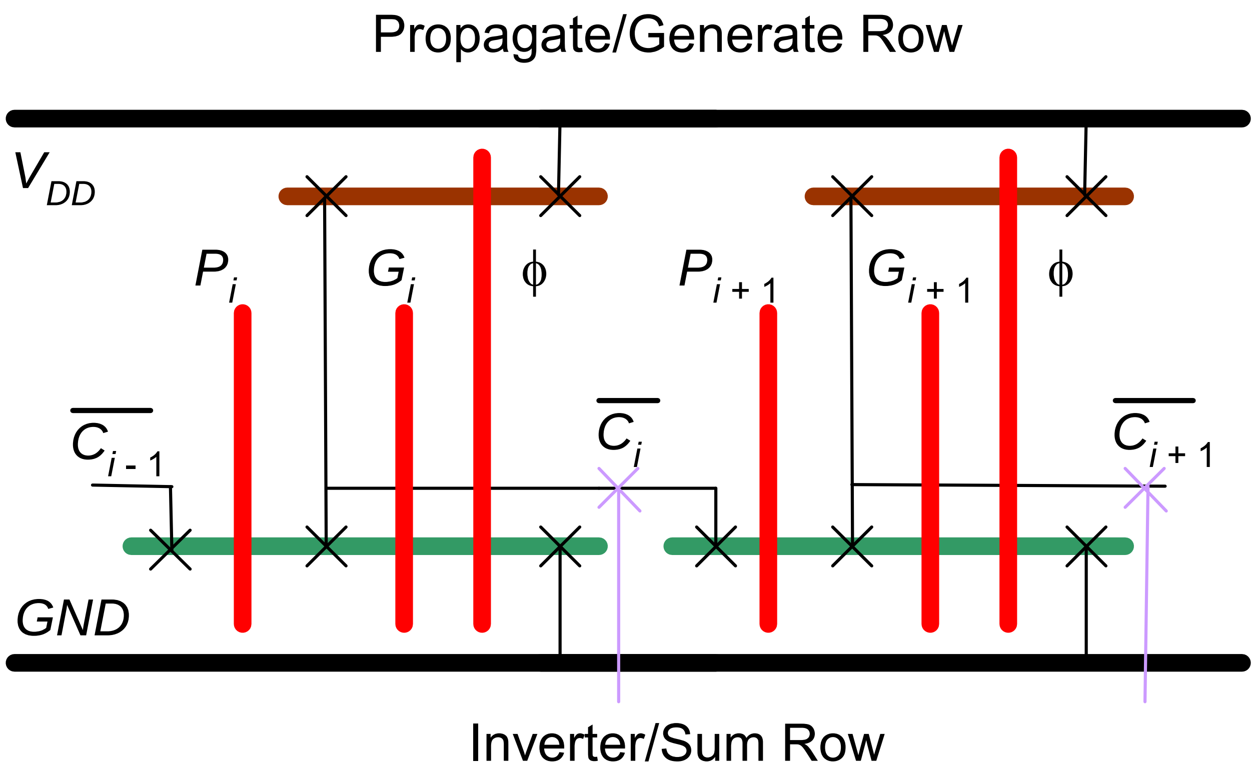
Carry Generation and Propagation



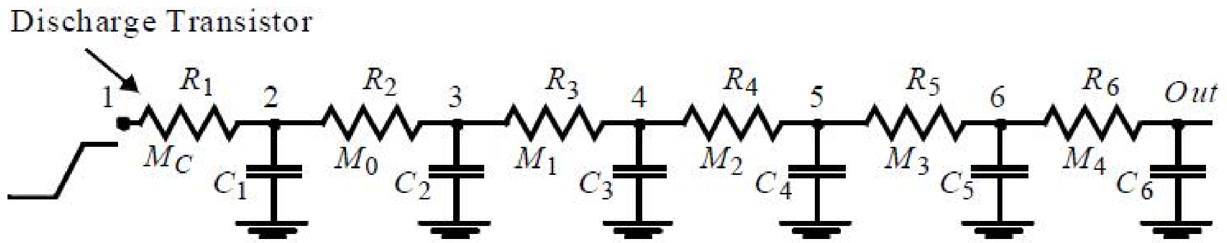
Manchester Carry Chain



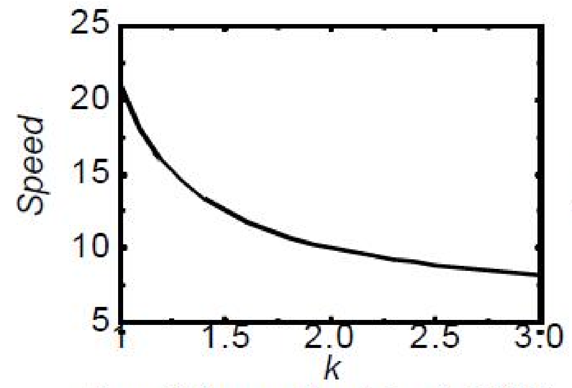
Stick Diagram of Manchester Carry Chain



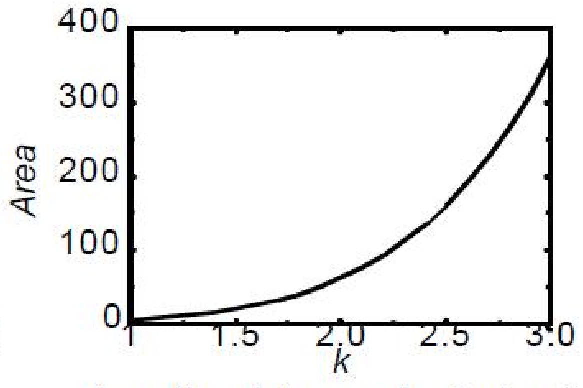
Sizing Manchester Carry Chain



$$t_p = 0.69 \sum_{i=1}^{N} C_i \left(\sum_{j=1}^{i} R_j \right)$$



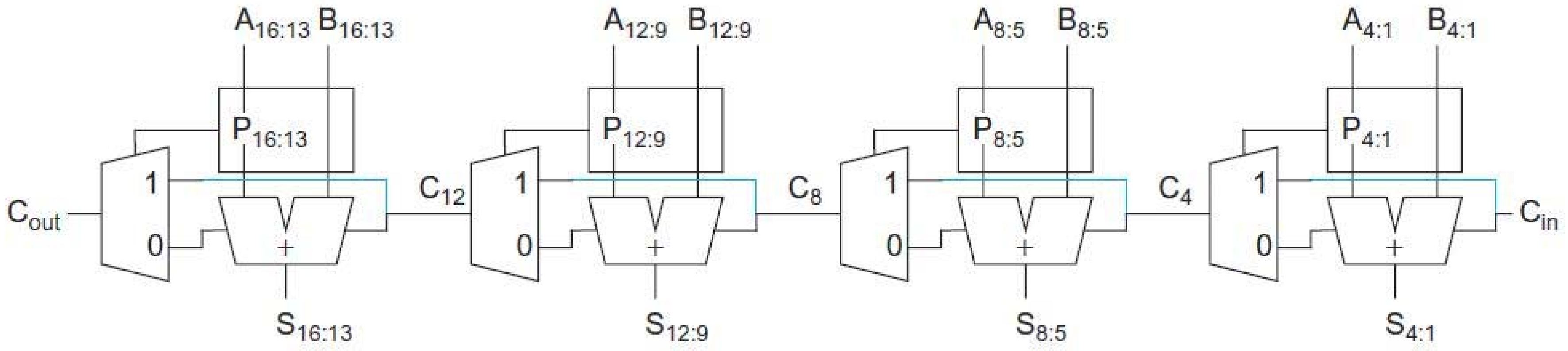
Speed (normalized by $0.69RC$)



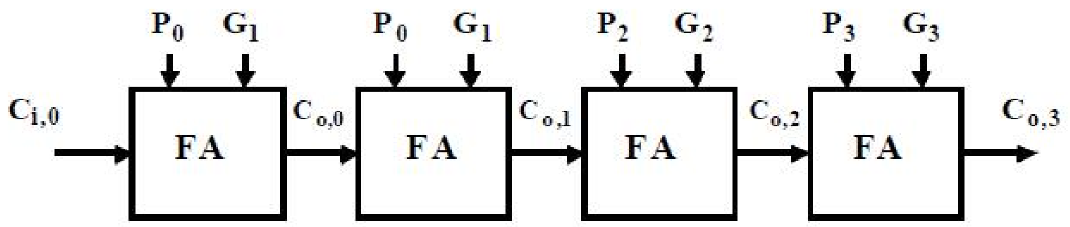
Area (in minimum size devices)

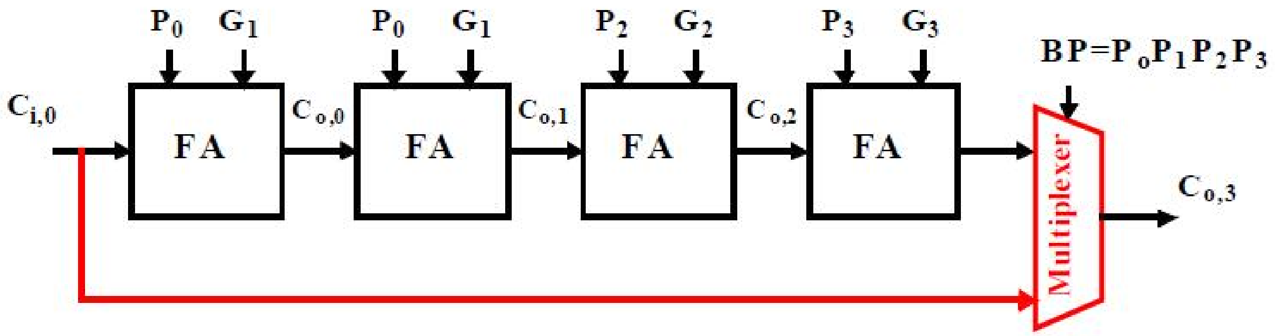
Carry-Bypass/Skip Adder

- Carry-ripple is slow through all N stages
- Carry-bypass allow carry to skip over groups of n-bits



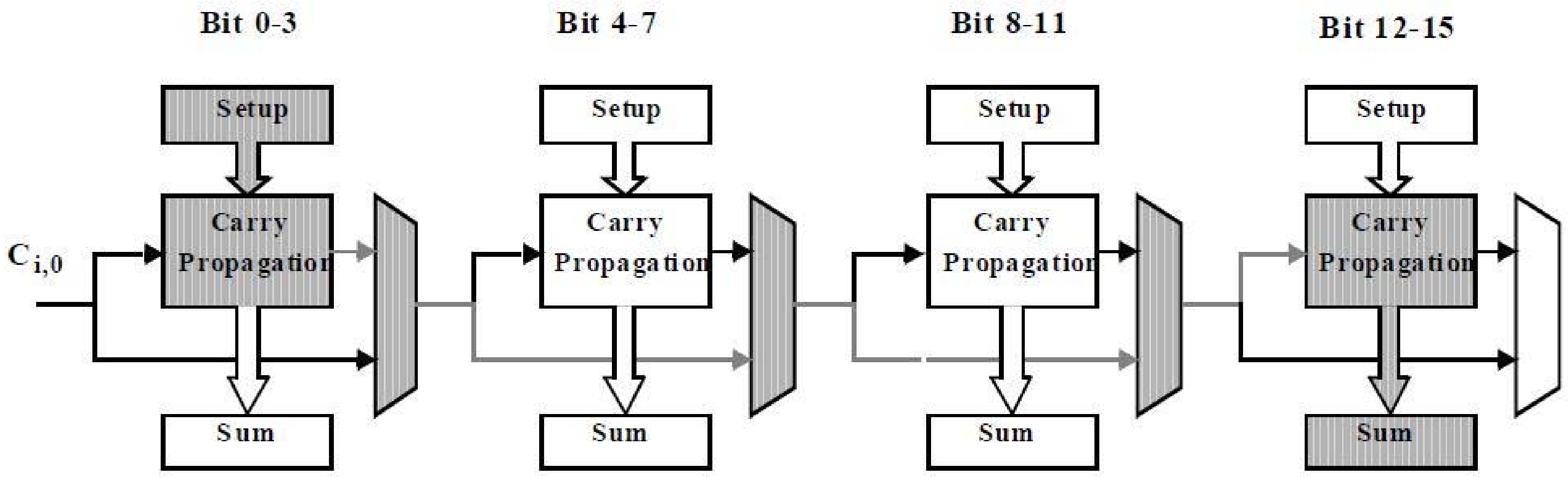
Carry-Bypass Adder





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

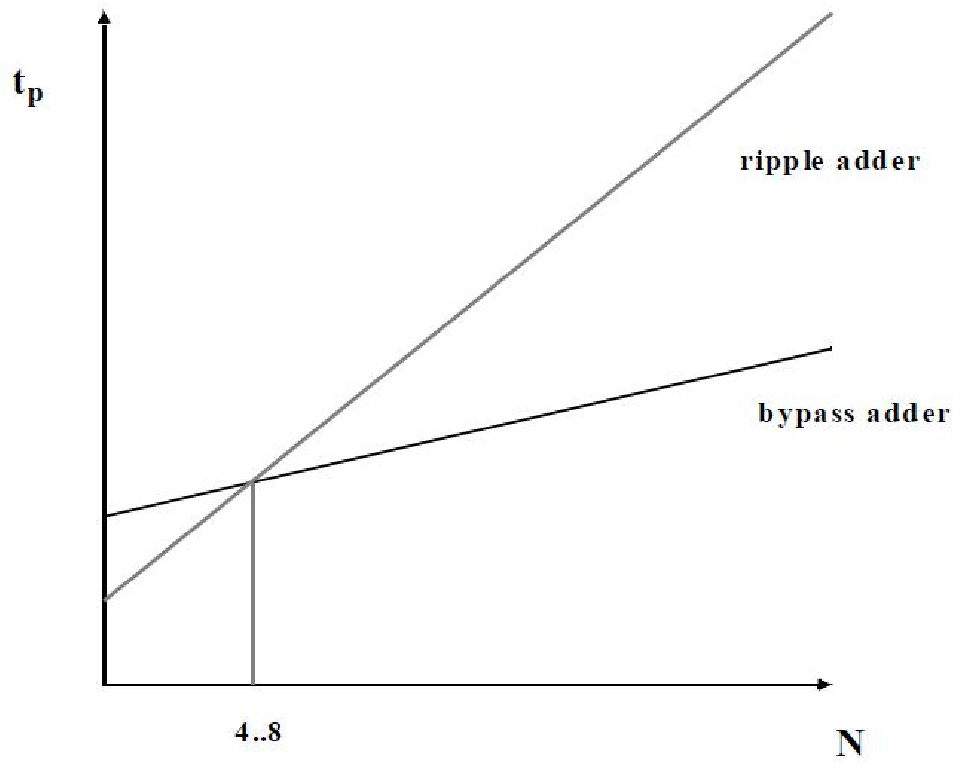
Carry-Bypass Adder



***N* bits, *M* bits per block**

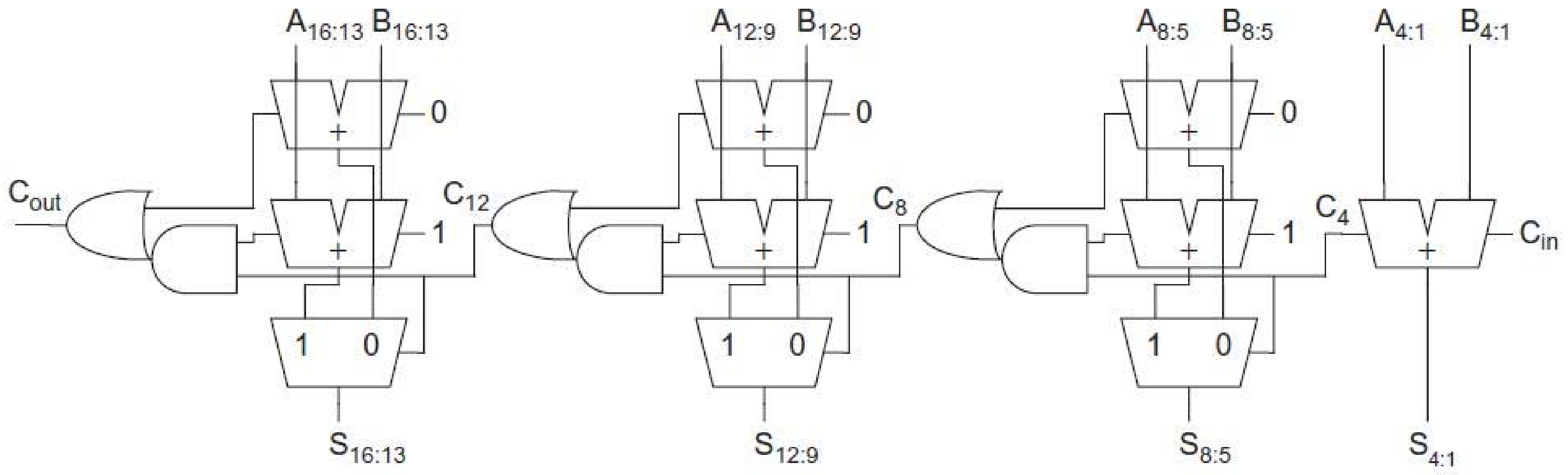
$$t_{adder} = t_{setup} + Mt_{carry} + (N/M\text{-}1)t_{bypass} + (M\text{-}1)t_{carry} + t_{sum}$$

Carry Ripple *vs.* Carry Bypass

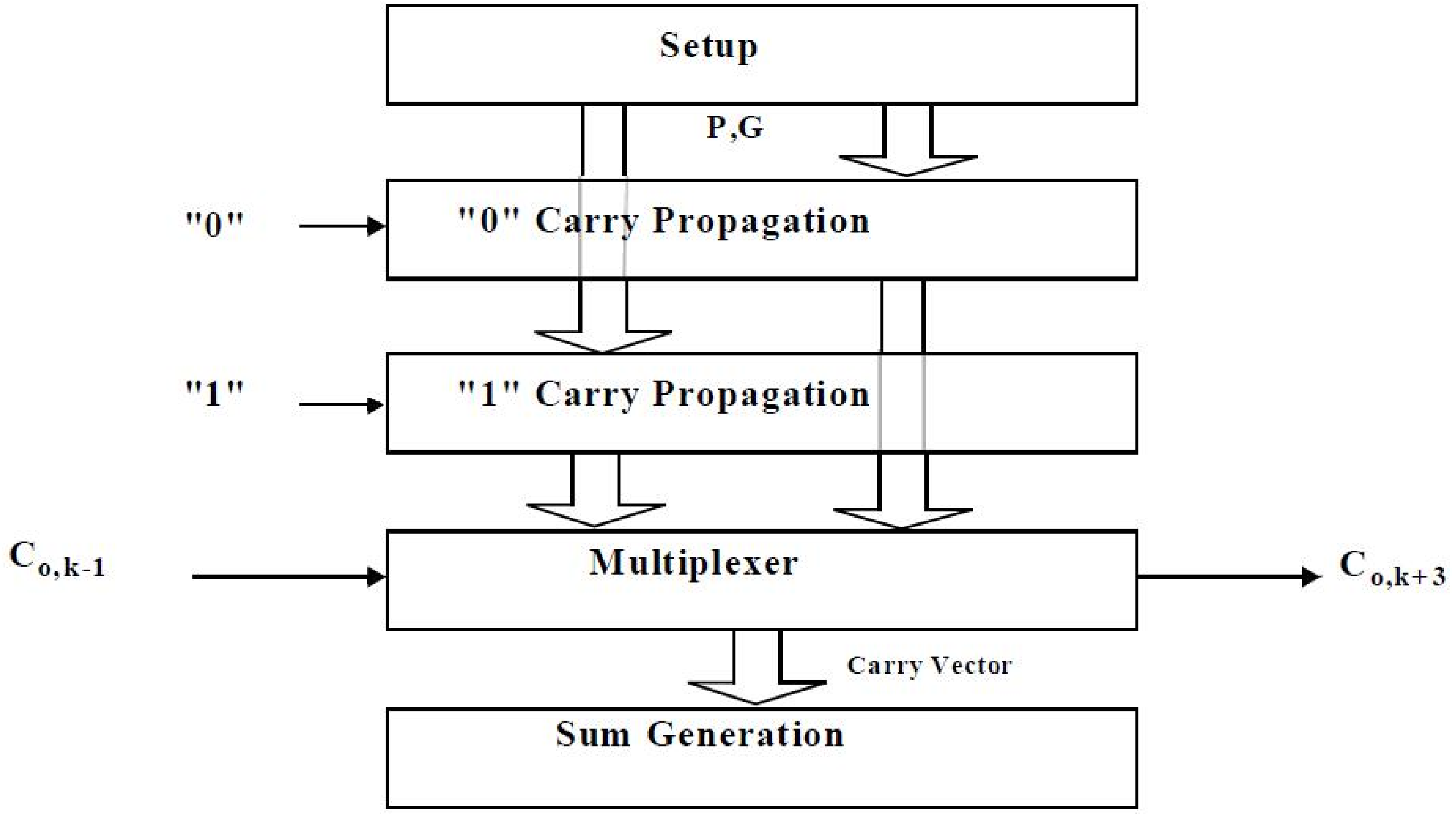


Carry-Select Adder

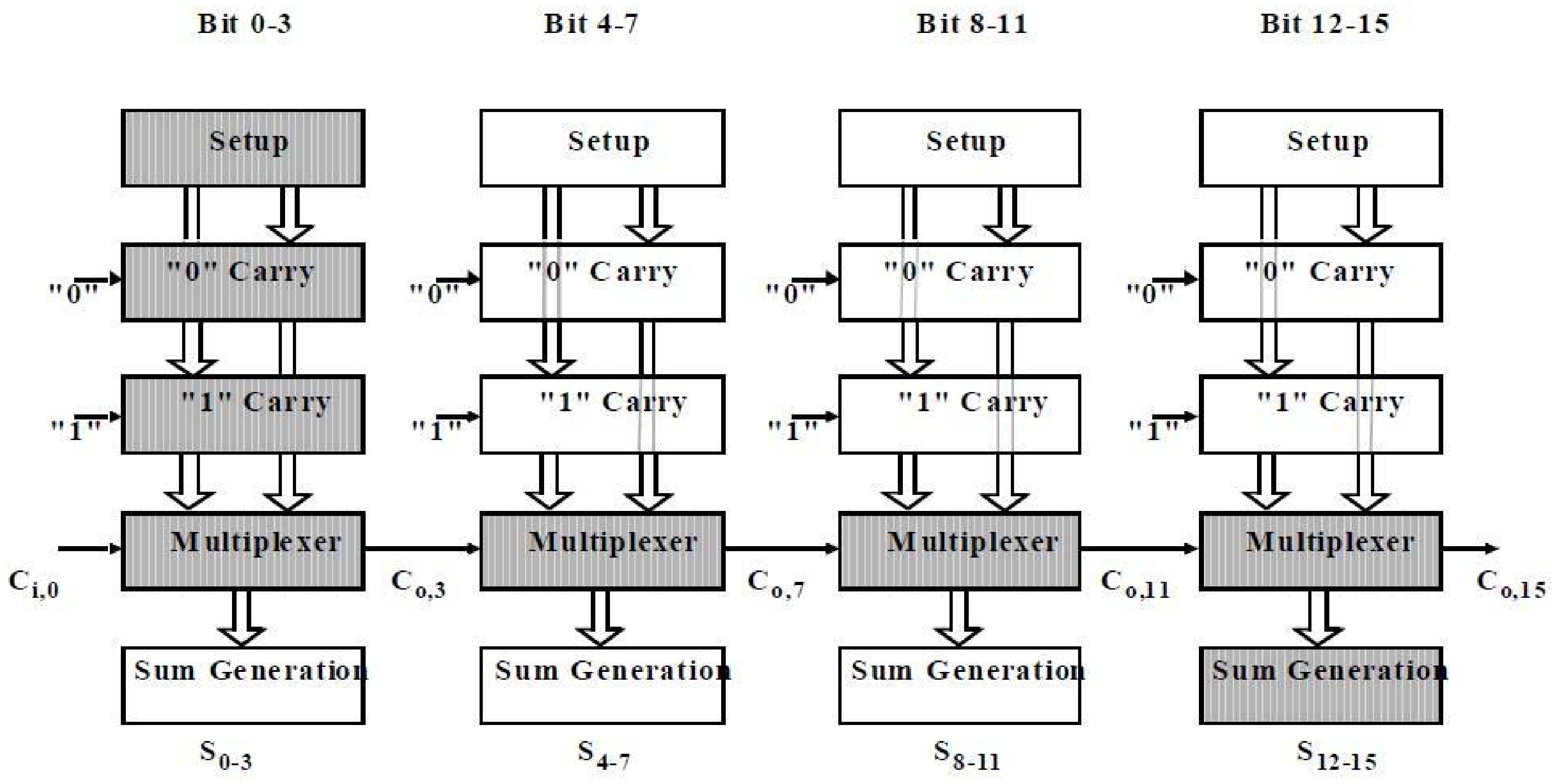
- Anticipate both possible values of the carry input
- Select the correct values when the carry input arrives



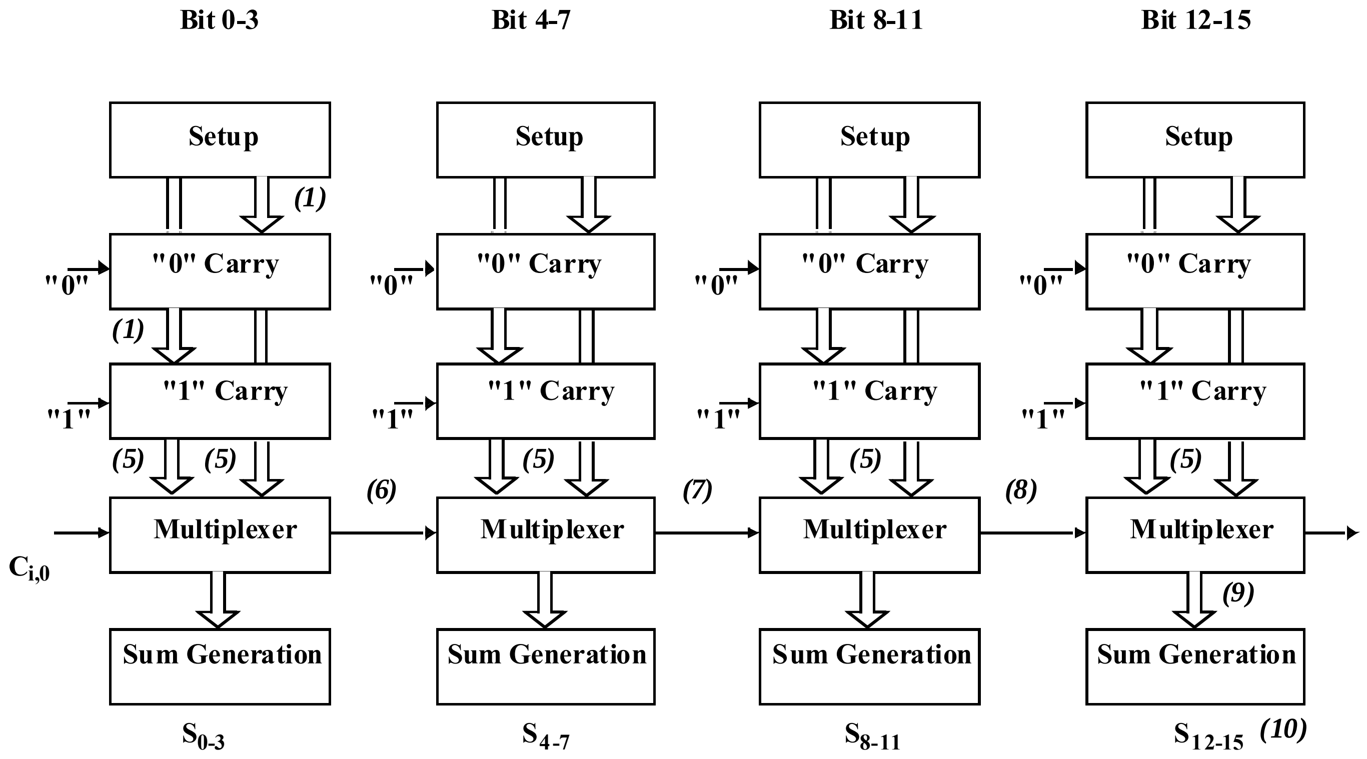
Carry-Select Adder



Carry-Select Adder: Critical Path

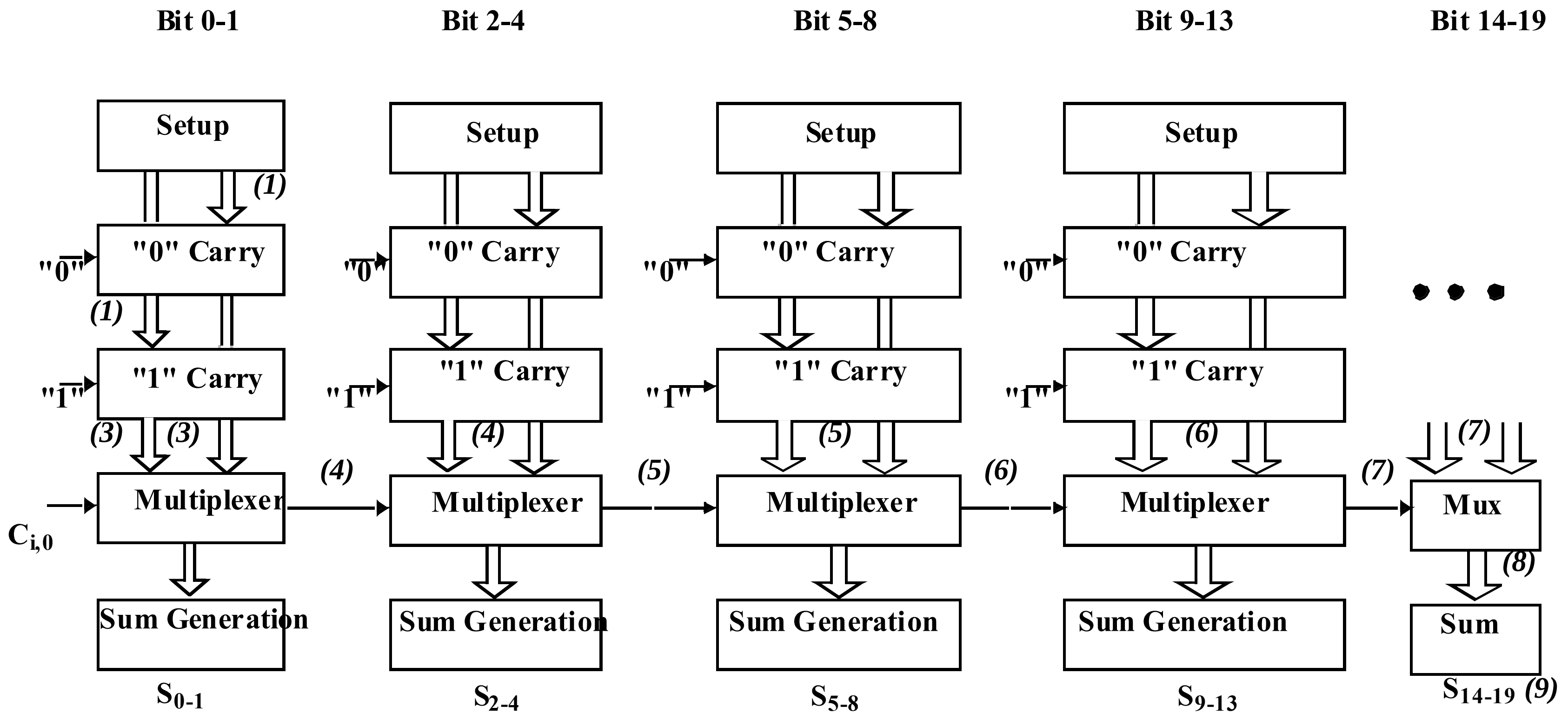


Linear Carry Select



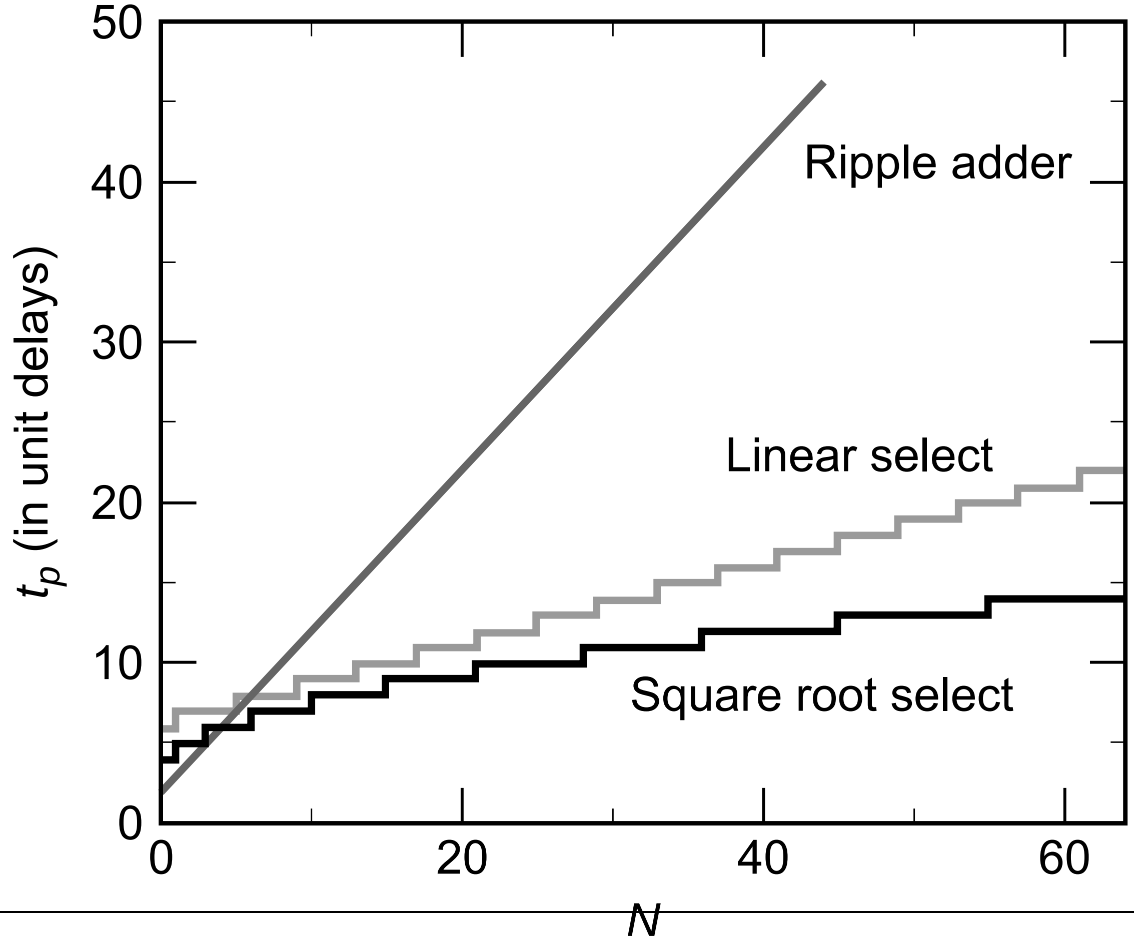
$$t_{add} = t_{setup} + \left(\frac{N}{M}\right) t_{carry} + M t_{mux} + t_{sum}$$

Square Root Carry Select



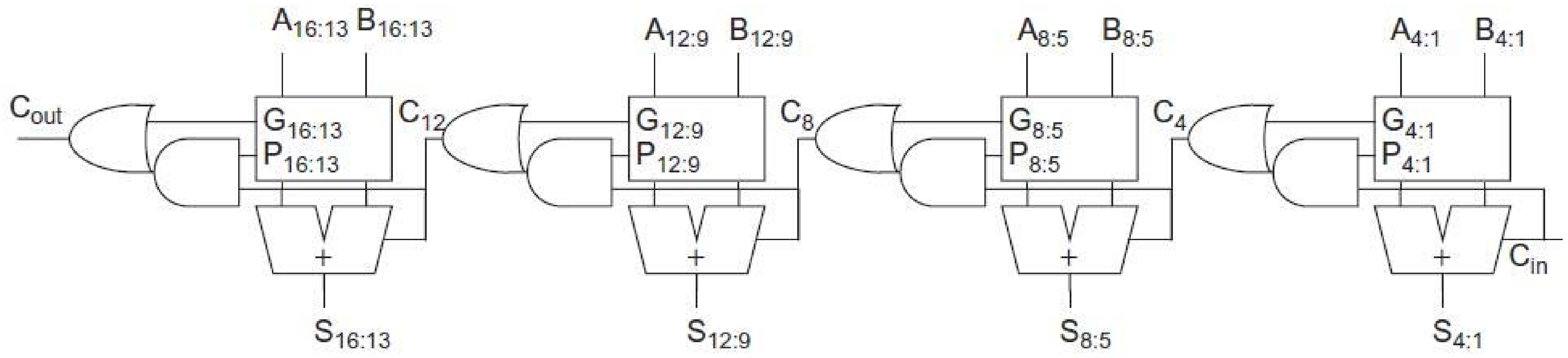
$$t_{add} = t_{setup} + P \cdot t_{carry} + (\sqrt{2N})t_{mux} + t_{sum}$$

Comparison of Adder Delay

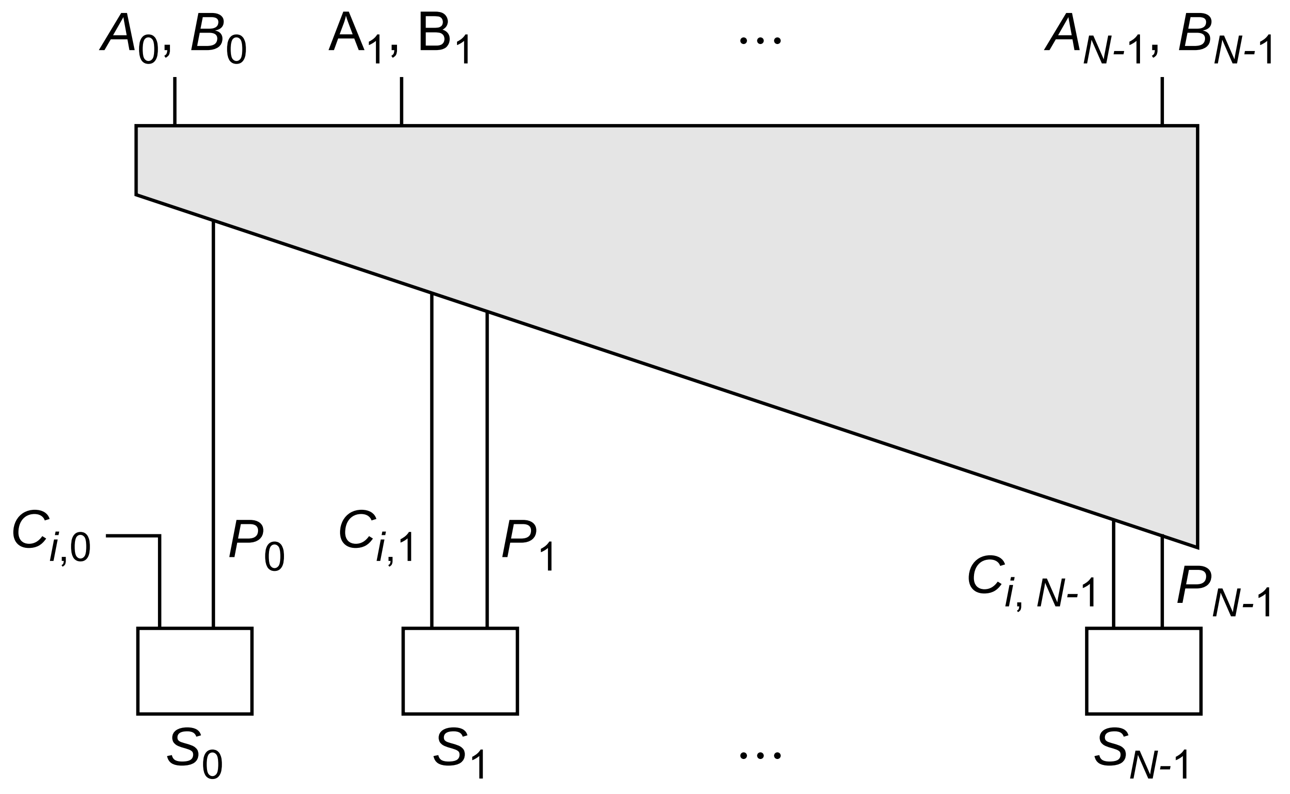


Carry-Lookahead Adder

- Compute G for many bit in parallel



Concept of Lookahead



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

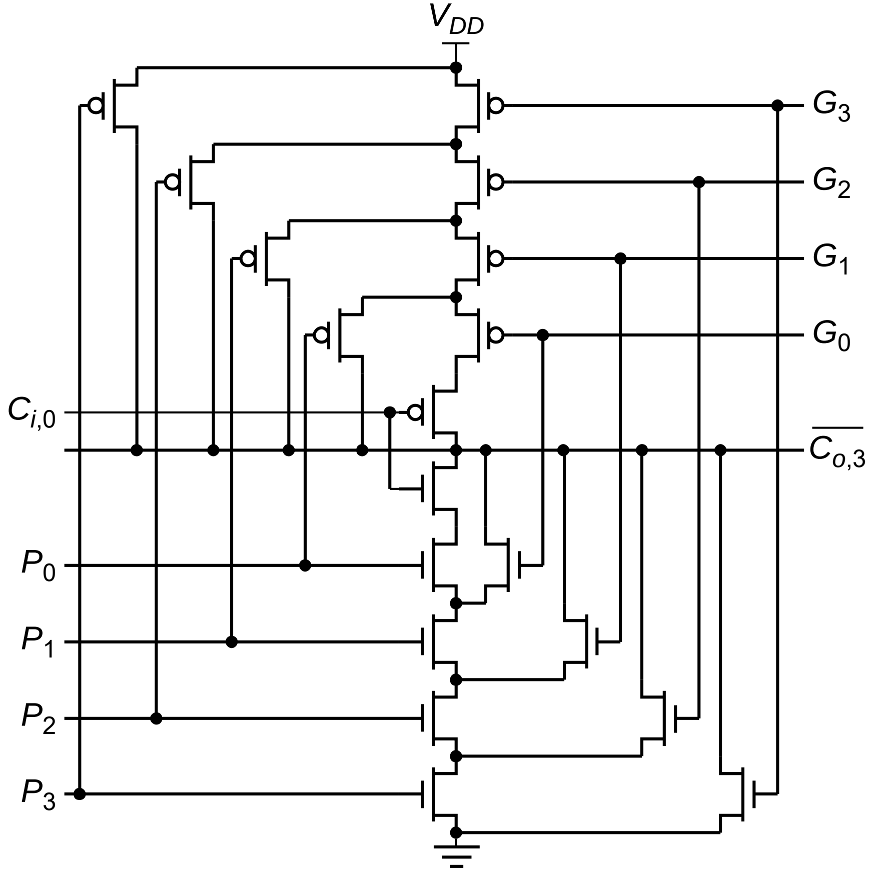
Topology of Lookahead

Expanding Lookahead equations:

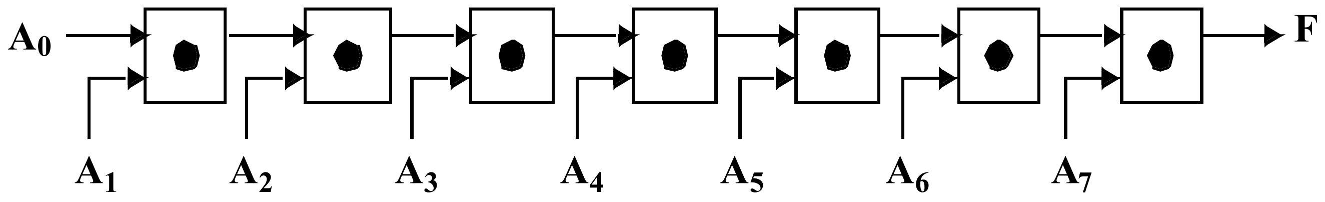
$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}C_{o,k-2})$$

All the way:

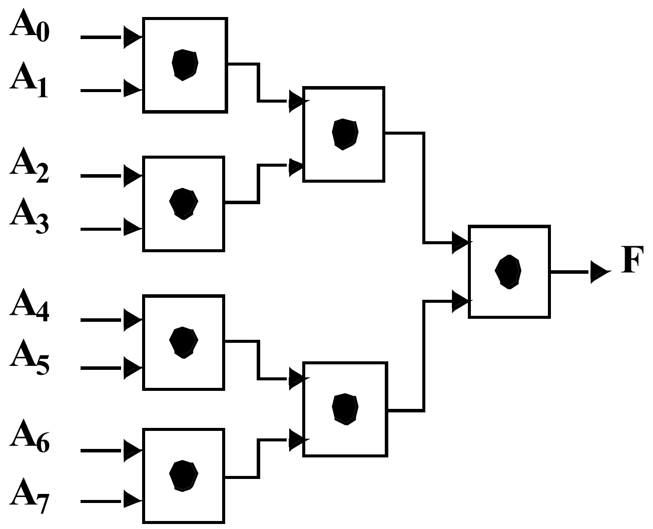
$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}(\ldots + P_1(G_0 + P_0 C_{i,0})))$$



Logarithmic Lookahead Adder



$t_p \sim N$



$t_p \sim \log_2(N)$

Carry Lookahead Trees

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

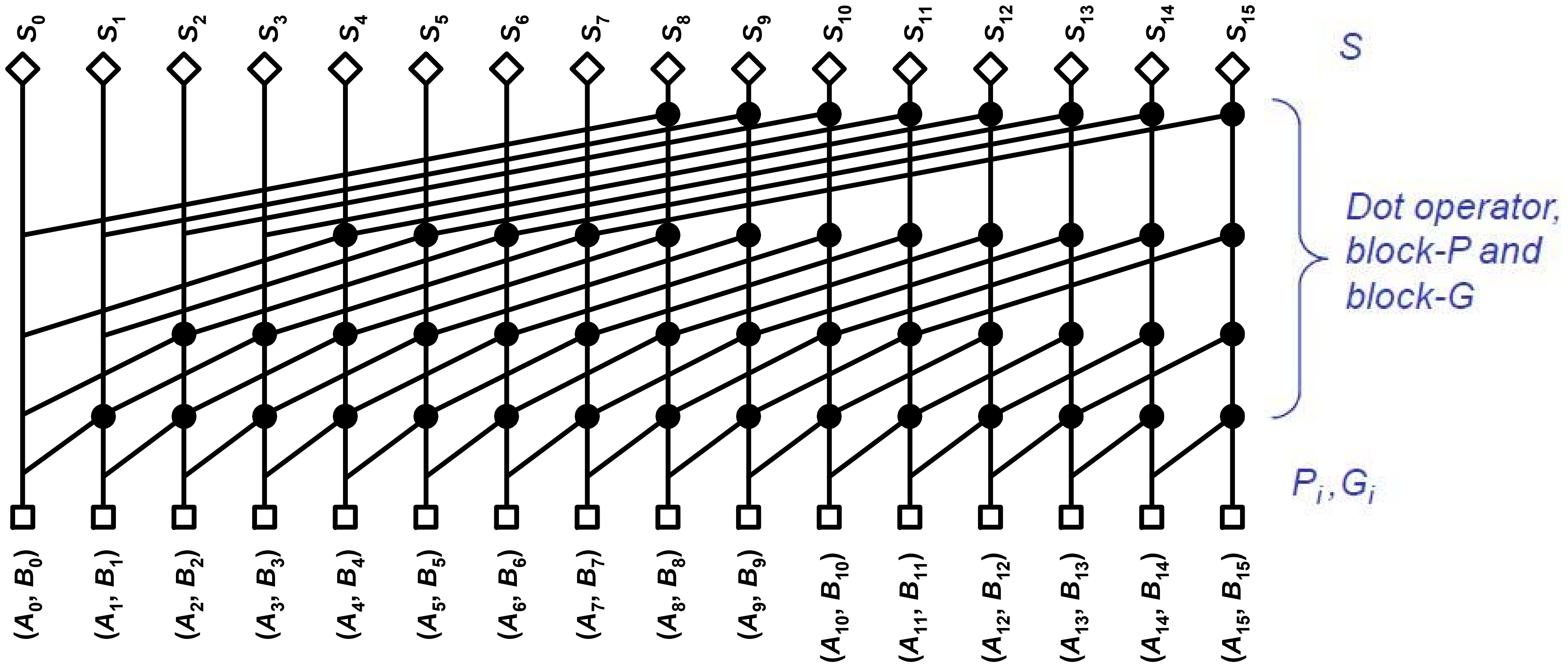
$$C_{o,1} = G_1 + P_1 G_0 + P_1 P_0 C_{i,0}$$

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

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

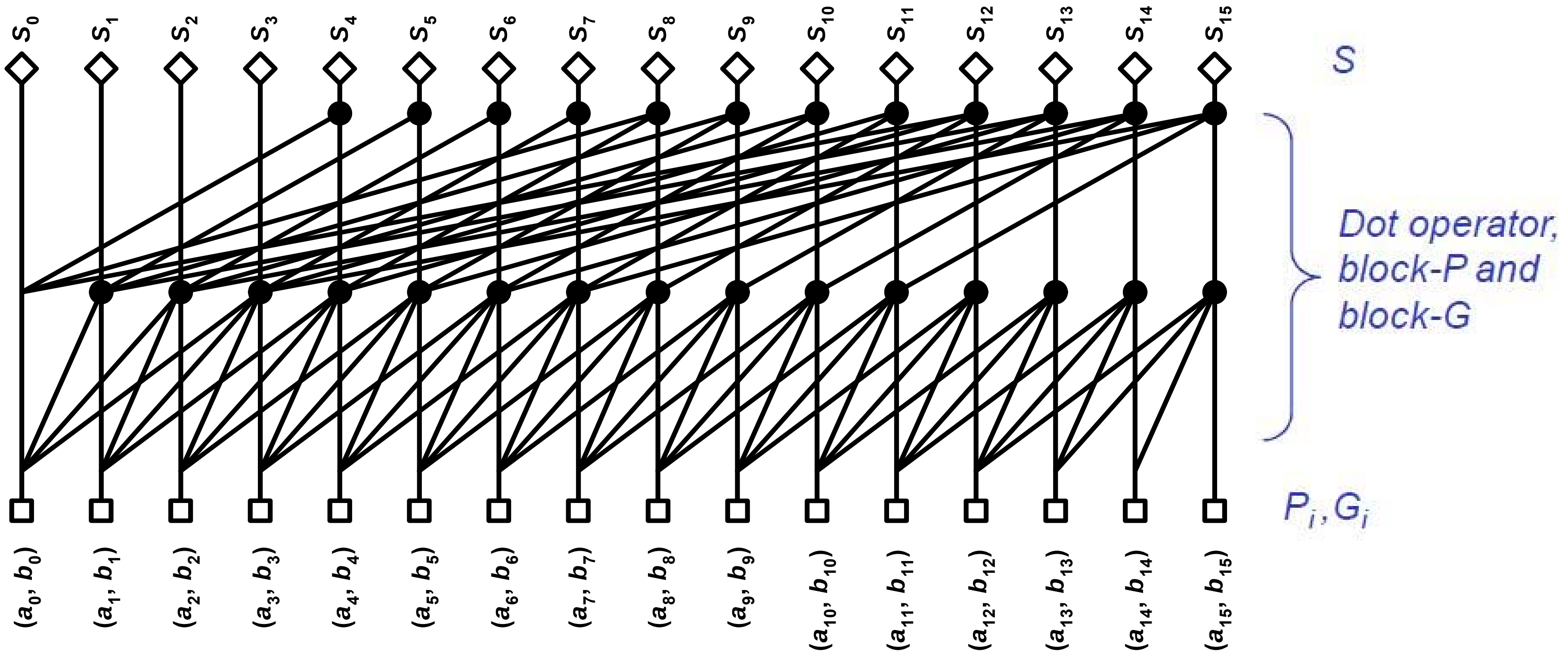
Can continue building the tree hierarchically.

Tree Adder



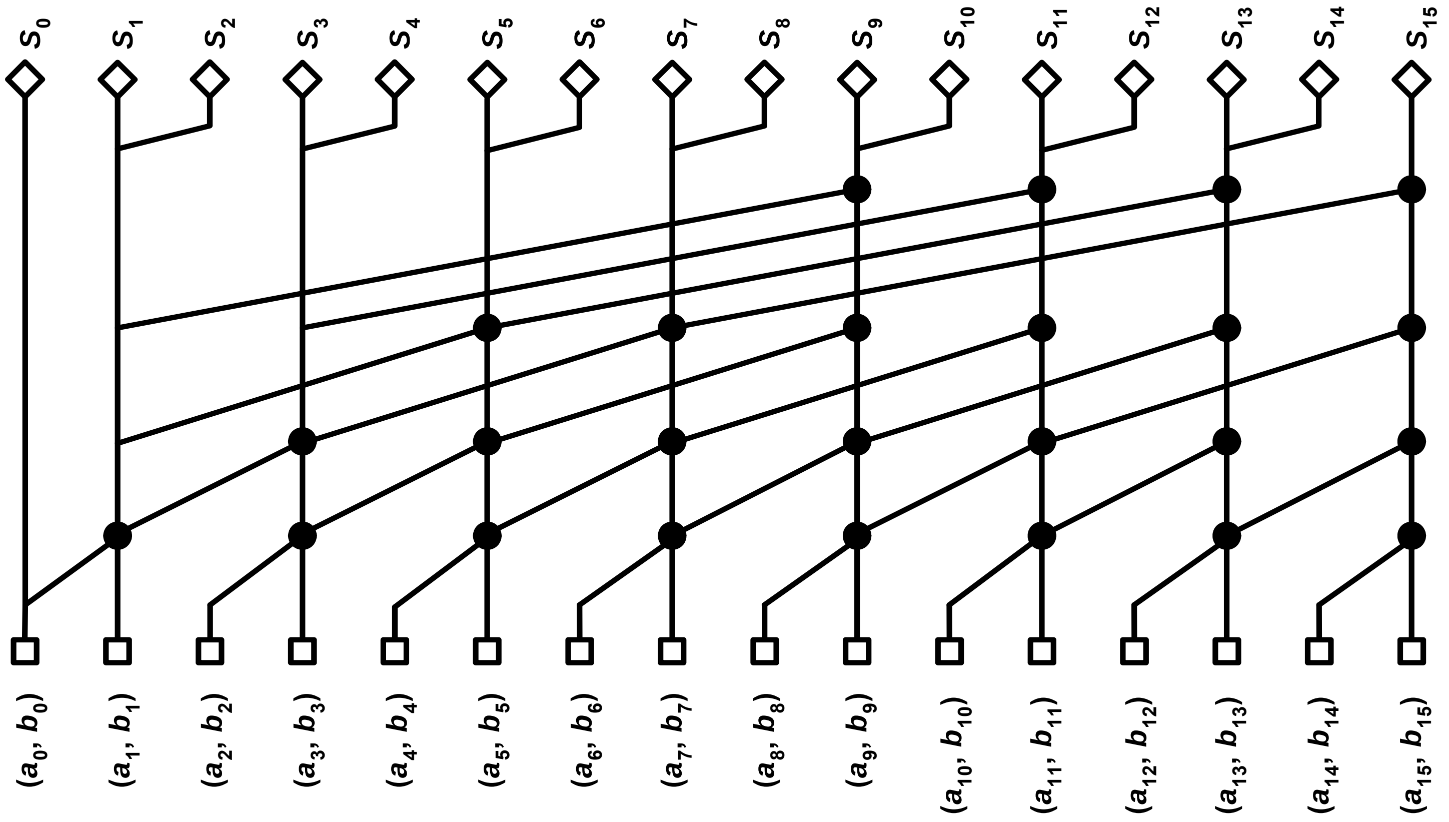
16-bit radix-2 Kogge-Stone tree

Tree Adder



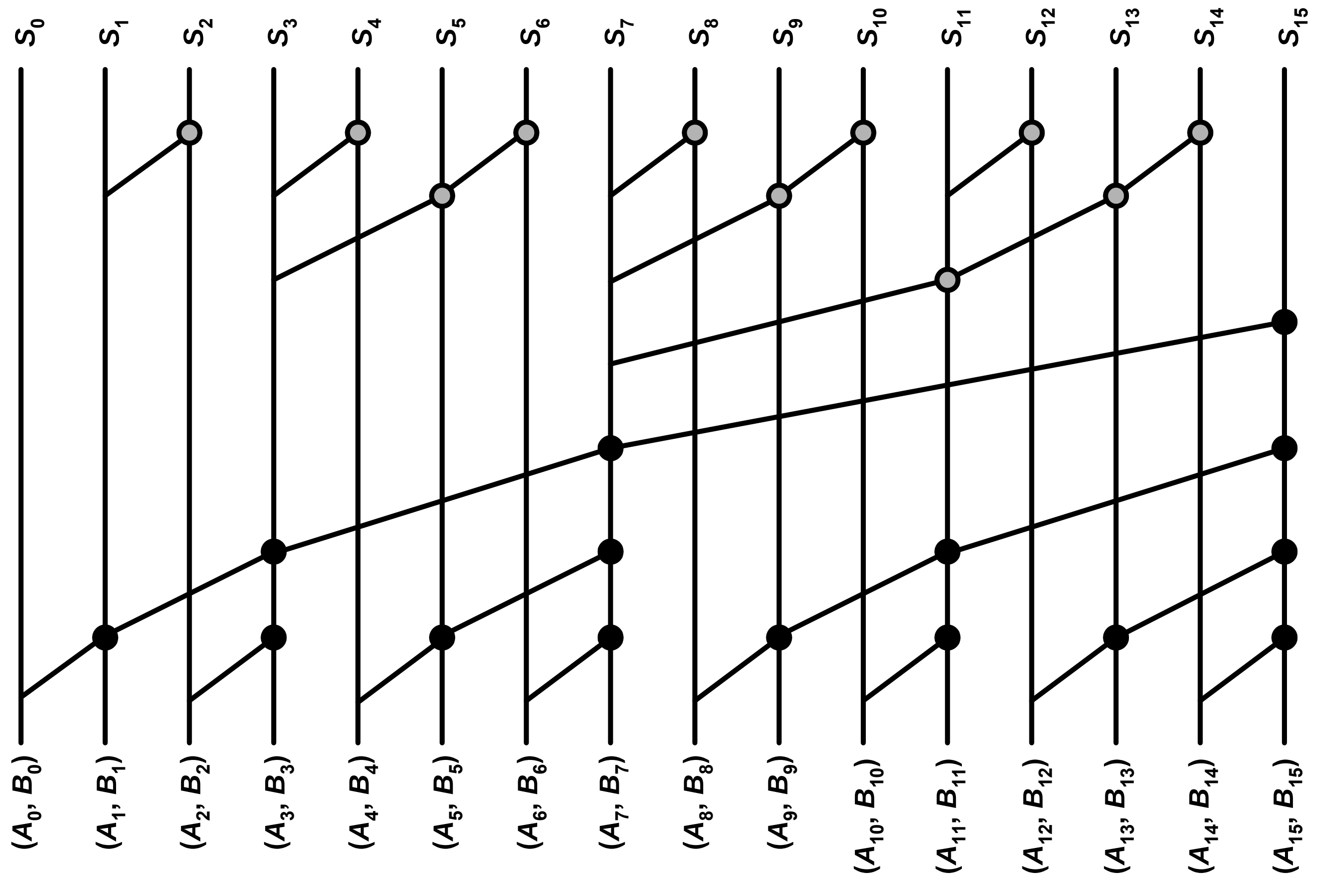
16-bit radix-4 Kogge-Stone Tree

Sparse Trees

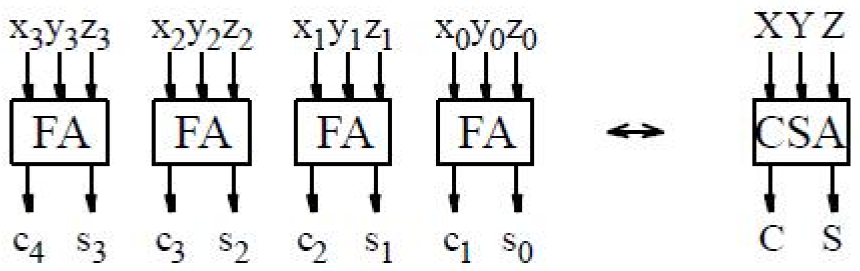


16-bit radix-2 sparse tree with sparseness of 2

Brent-Kung Tree



Carry-Save Adder

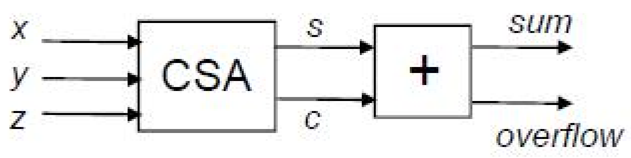


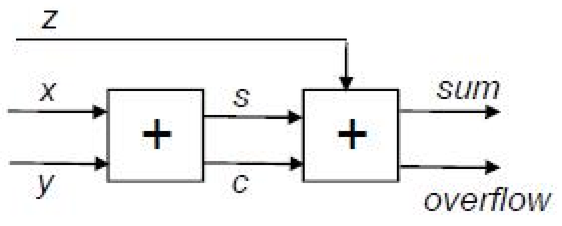
x:			1	0	0	1	1
y:			1	1	0	0	1
z:	+		0	1	0	1	1
s:			0	0	0	0	1
c:	+	1	1	0	1	1	
sum:		1	1	0	1	1	1

No carry used here

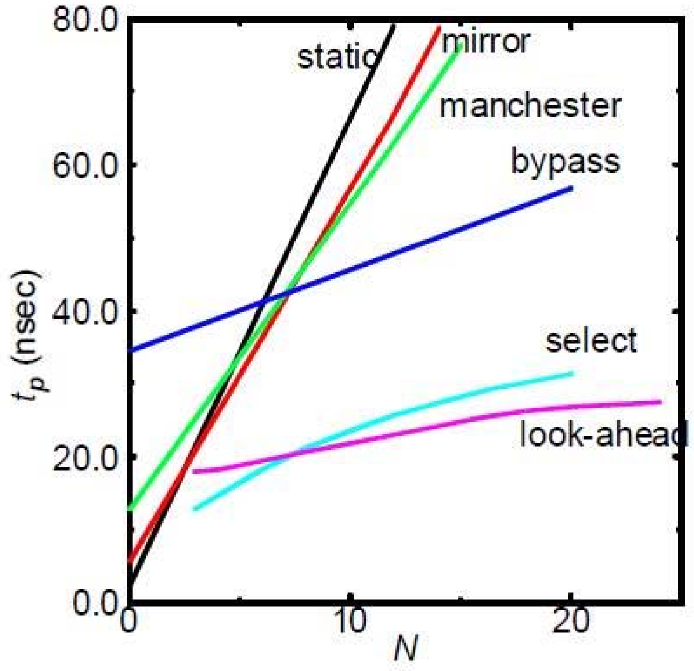
Extra adder

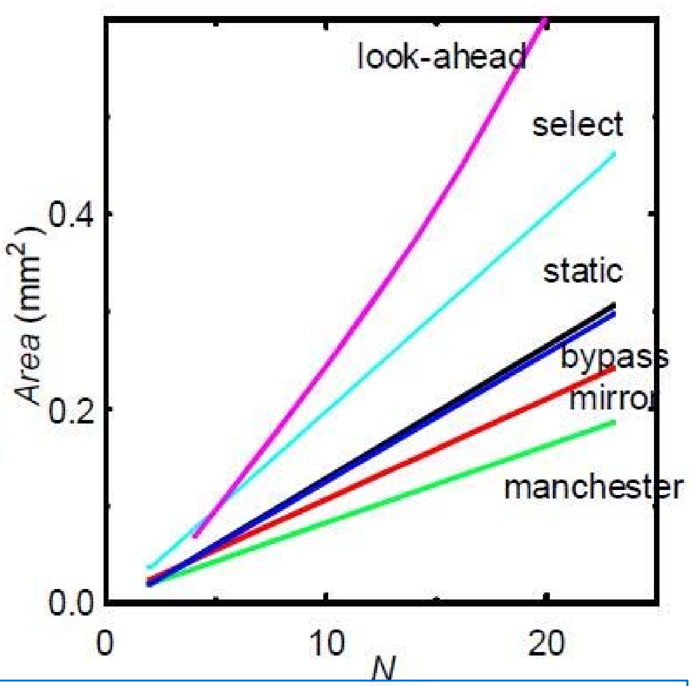
To be compared with 2 std cascaded adders





Summary





Design as a Trade-Off: Area – speed – power requirements should be verified in early design phases to choose the right structure

Reference

- *Digital Integrated Circuits*, by Jan M. Rabaey, et al.
- *CMOS VLSI Design*, by David Harris, et al.

www.liu.se

LINKÖPING
UNIVERSITY