EE3230 Lecture 9: Datapath Subsystems

Ping-Hsuan Hsieh (謝秉璇)

Delta Building R908

EXT 42590

phsieh@ee.nthu.edu.tw

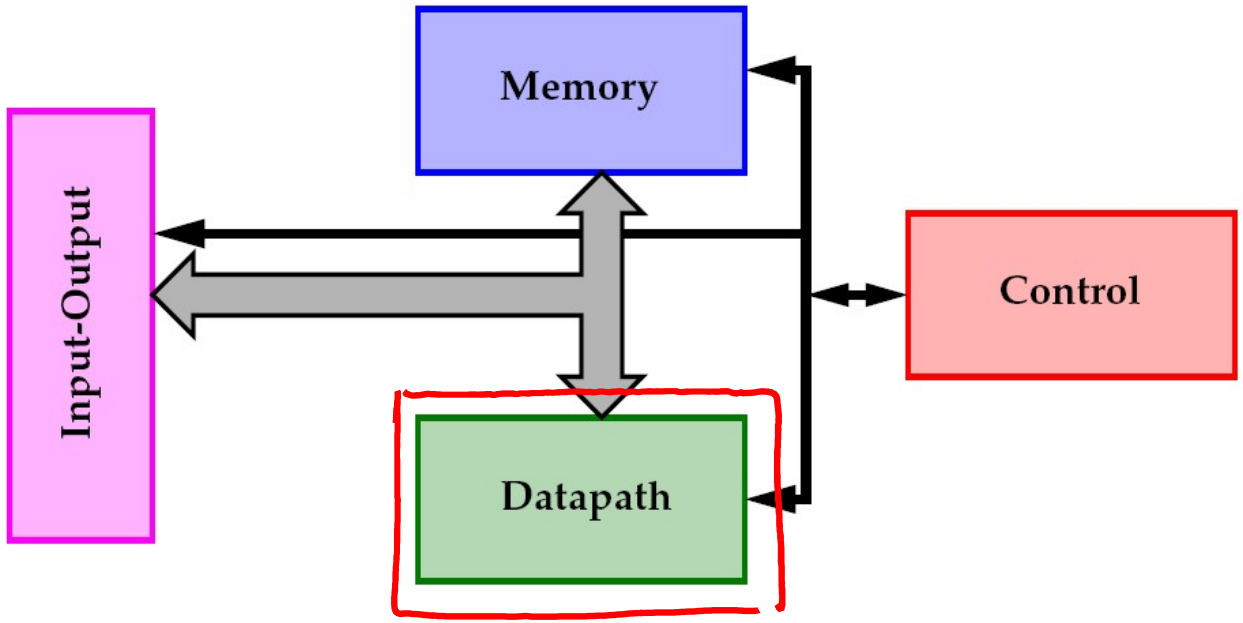
Outline

- **Addition/Subtraction**
- Multiplication

- One/Zero Detectors
- Comparators
- Counters
- Shifters

Digital Device Components

- Processors are composed of many basic components



- **Datapath** comprises the core
- Other components are for supporting
  - Power management and distribution, clock generation and distribution, analog, RF modules, etc.

Digital Device Components

- **Memory:**
  - Read-only vs. read-write
  - Sequential vs. ramdom access
  - Single-ported vs. multi-ported access

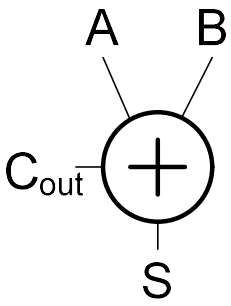
  } the way data is accessed

  - Dynamic vs. statc —— data retention characteristics
- **Control:**
  - FSM (sequential circuit) implemented using random logics, PLAs, and memories
- **Interconnect and I/O:**
  - Parasitic R, C, L affect performance for wires both on and off the chip
  - Growing die size increases the length of the on-chip interconnect, increasing the impact of parasitics

Single-Bit Addition

- Half adder

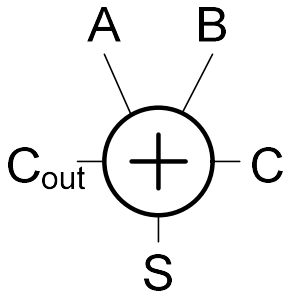
$$S = A \oplus B$$
$$C_{out} = A \bullet B$$

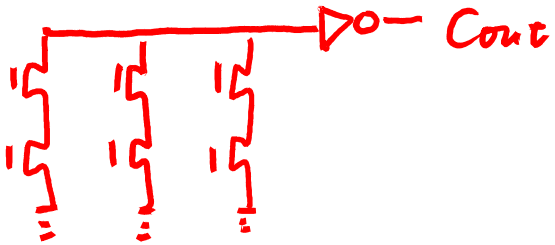


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Full adder

$$S = A \oplus B \oplus C$$
$$C_{out} = MAJ(A, B, C)$$
$$= AB + BC + AC$$





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

Full-Adder Design I

- Direct implementation from equations

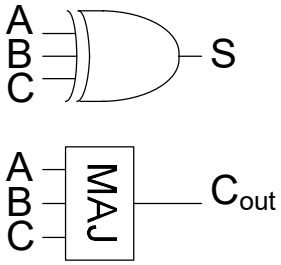
$$S = A \oplus B \oplus C \quad \text{(16 transisotrs)}$$

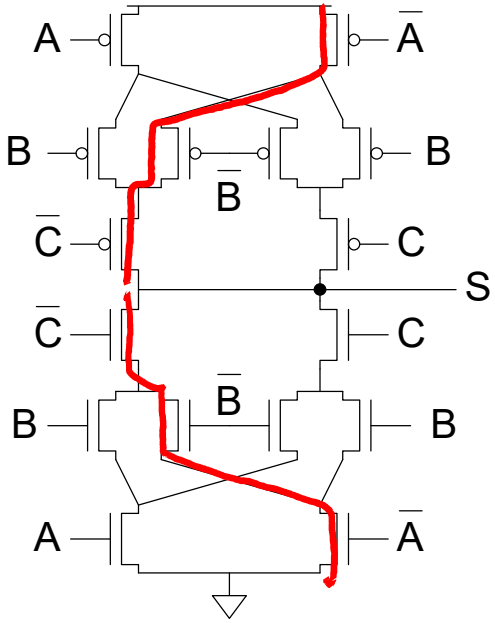
$$C_{out} = MAJ(A,B,C) \quad \text{(10 transisors)}$$

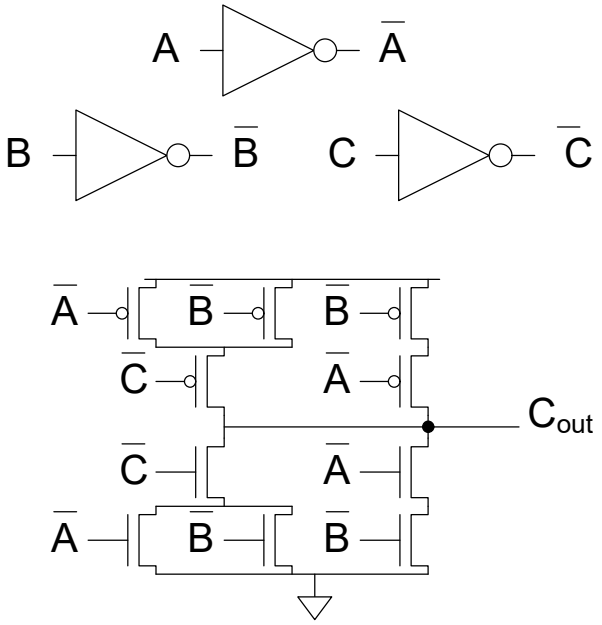
INV (6 transistors)

total 32 transistors

mirror structure

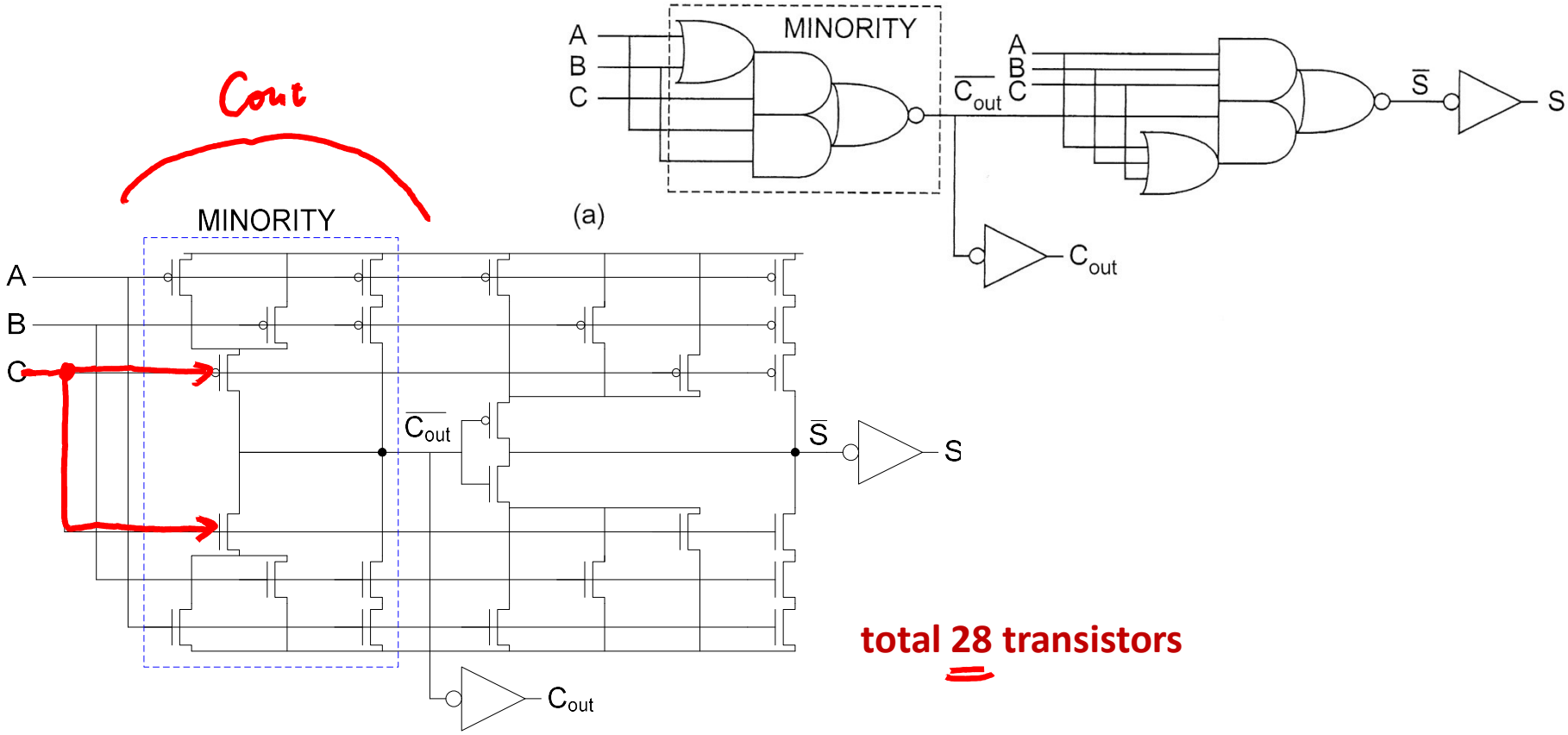






Full-Adder Design II

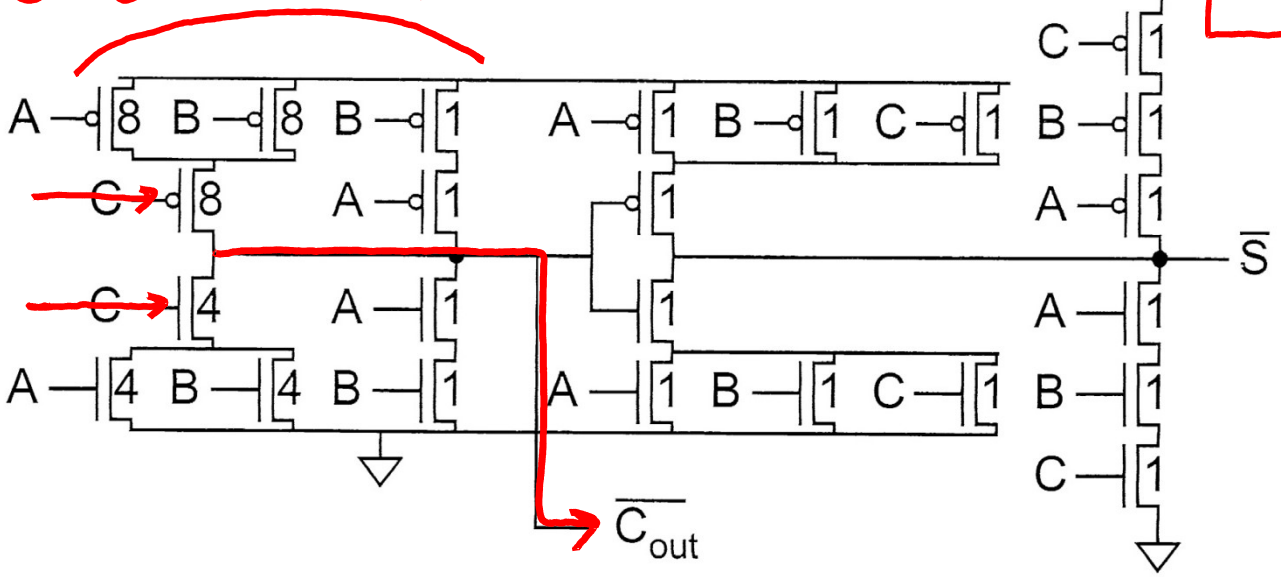
- Compute S from C_{out} : $S = ABC + (A + B + C)(\overline{C_{out}})$
- Mirror structure



total 28 transistors

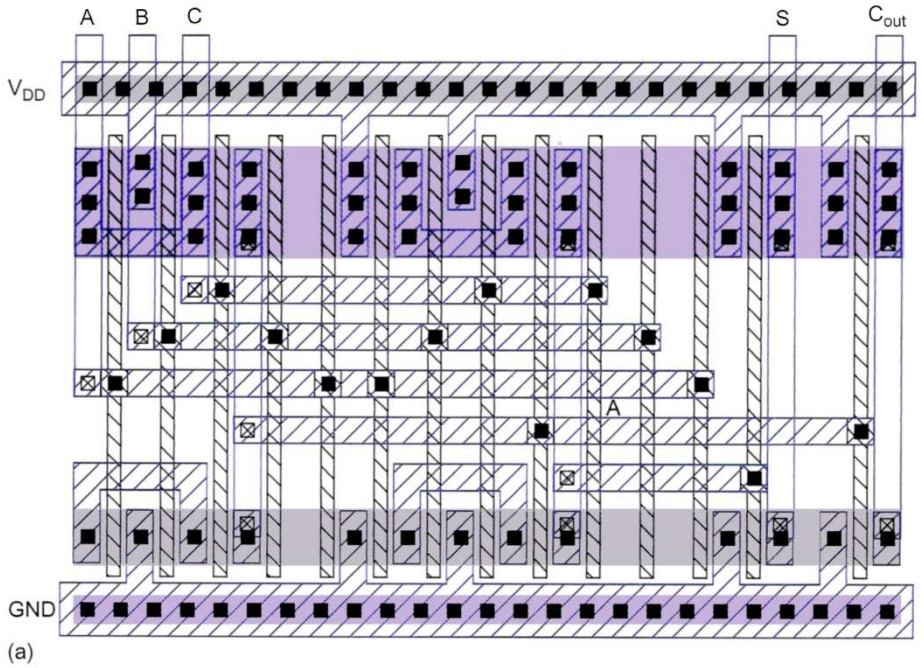
Critical Path Optimization

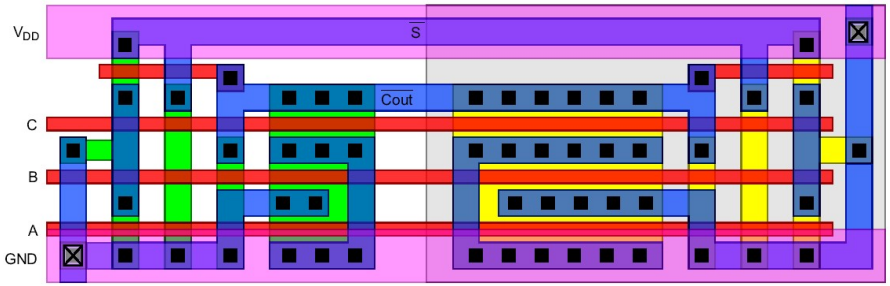
- **C to C_{out}** in ripple-propagate adders
- Feed carry-in (C) to inner inputs
- Minimum-size transistors at summing (S) logic for minimizing branching effort on the critical path
- Asymmetric gate to reduce logical effort on carry-in
- Eliminate output inverter – **24 transistors**

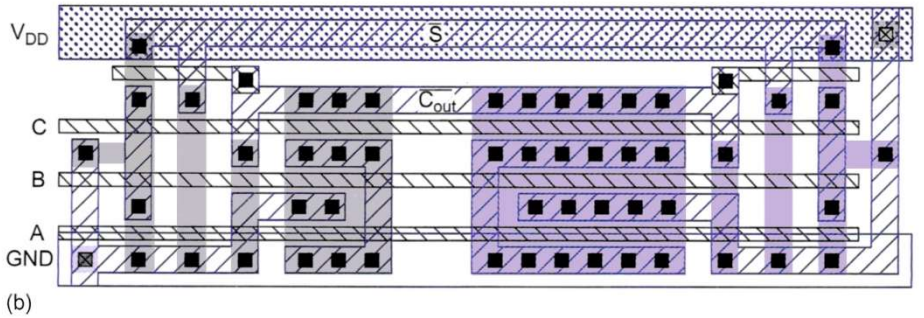


Layout

- Standard cell layout style
  - Fixed cell height
  - Fixed active area pitch
  - Fixed transistor size
- Datapath layout
  - Horizontal poly placement
  - Smaller height
  - Various transistor size

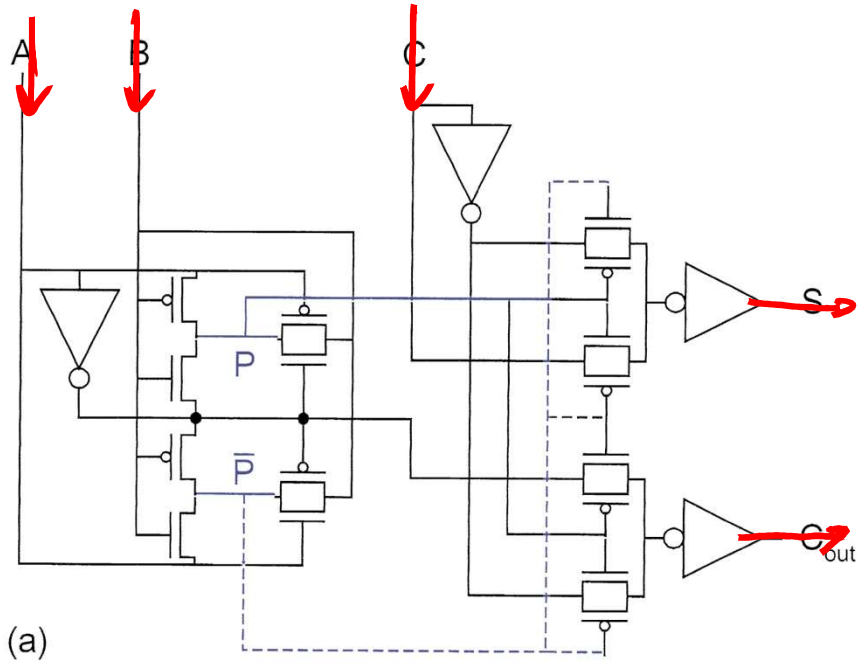




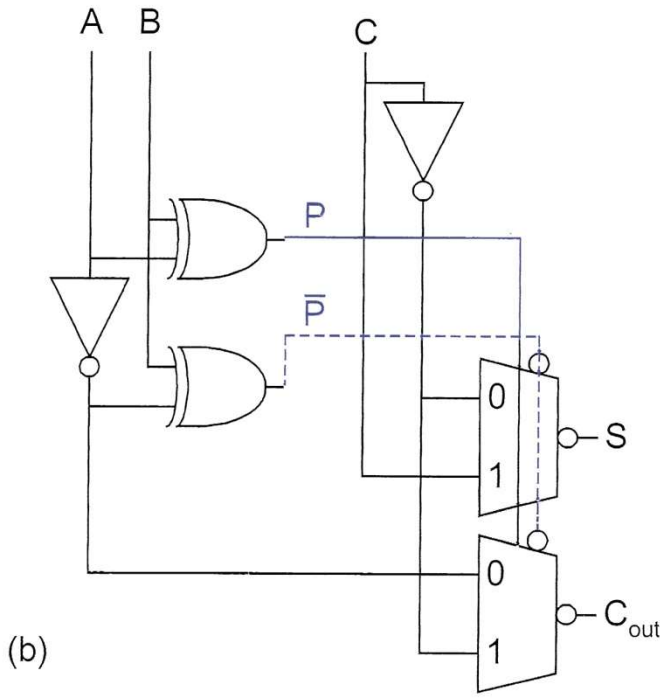


Full-Adder III

- Transmission gate full adder
  - Buffered outputs of proper polarity
  - Equal delay for S and C_{out} (important for multipliers)

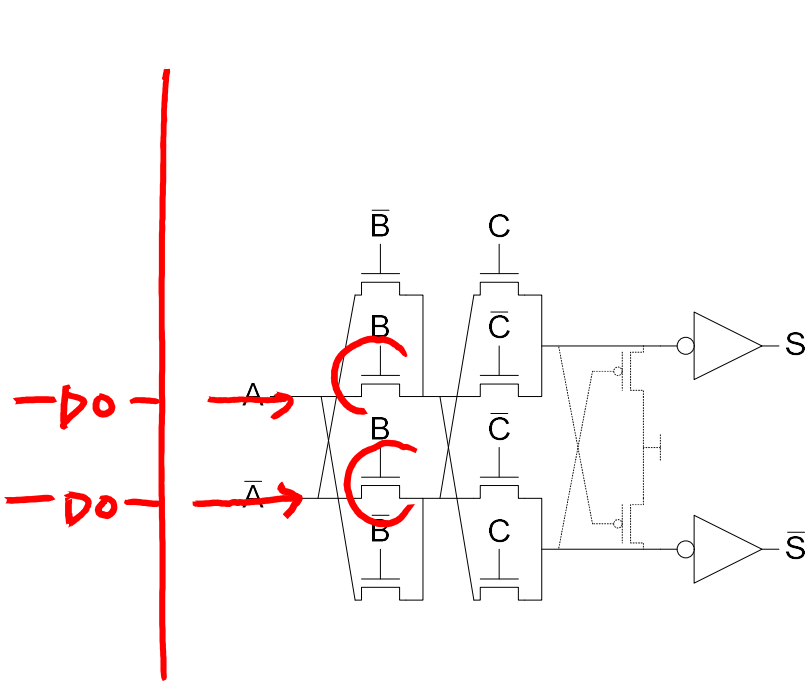


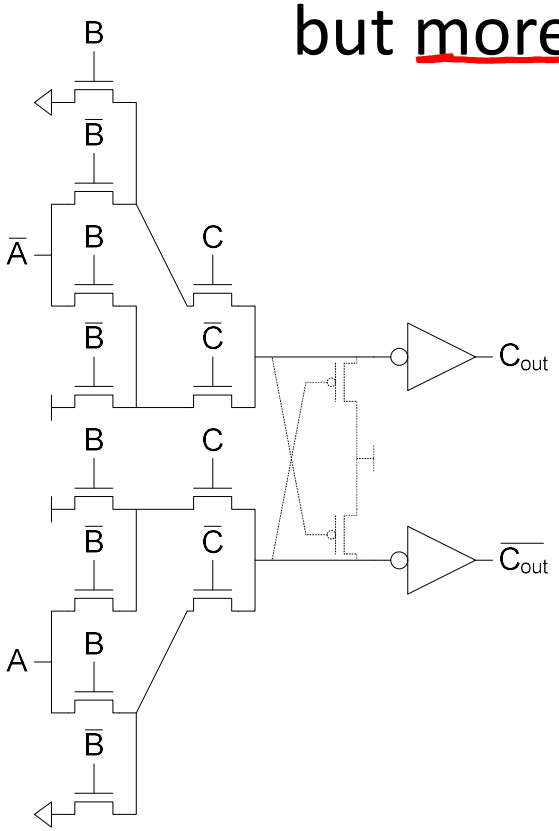
24 transistors



Full-Adder IV

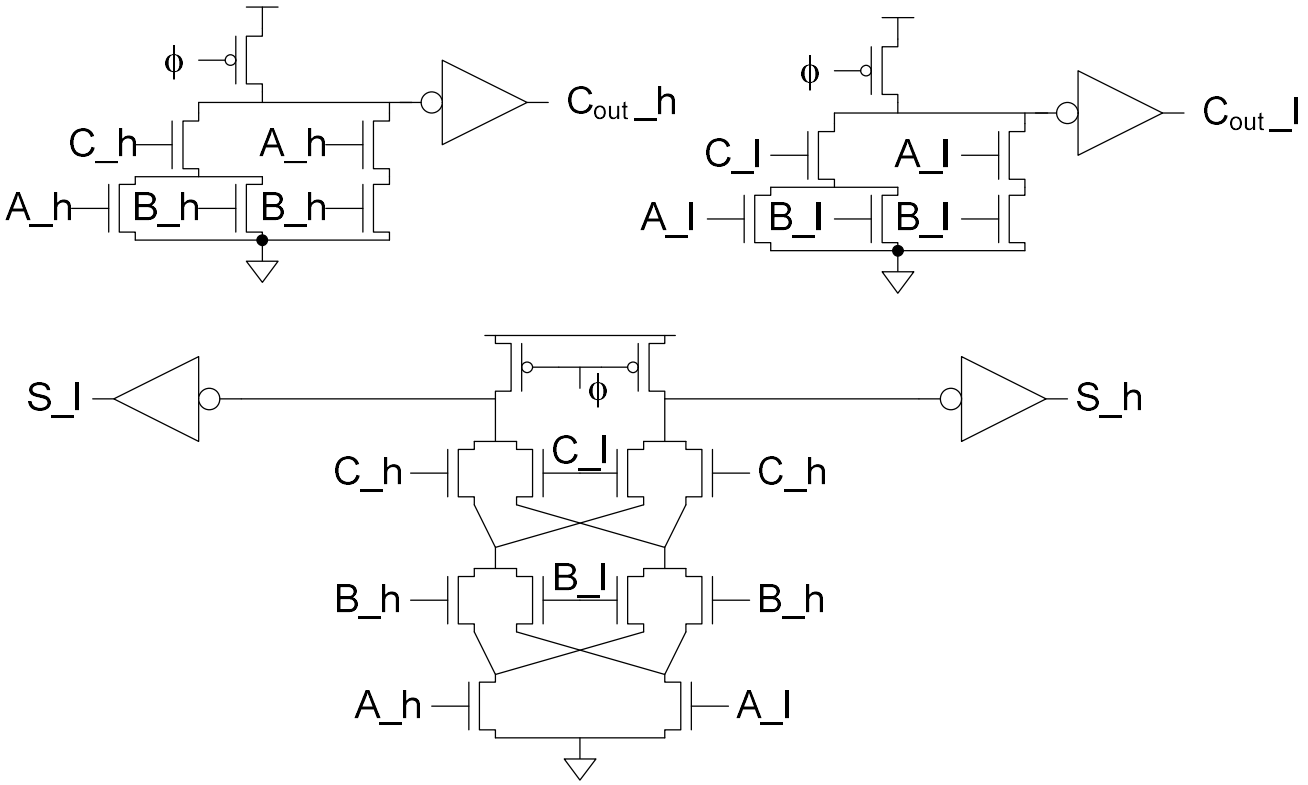
- Complementary Pass Transistor Logic (CPL)
- Compared to design-I: 2x faster, 30% lower power
- Compared to design-II: slightly faster, similar power, but more area





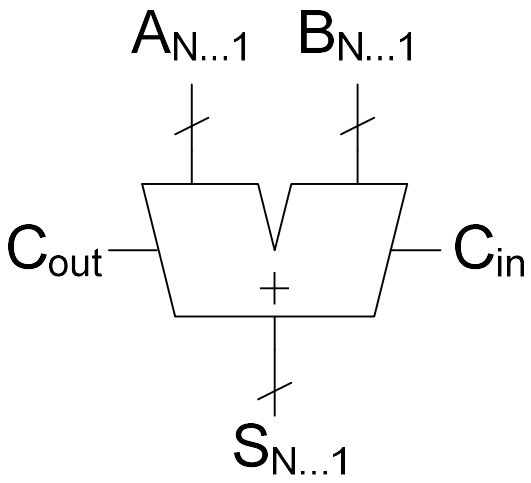
Full-Adder V

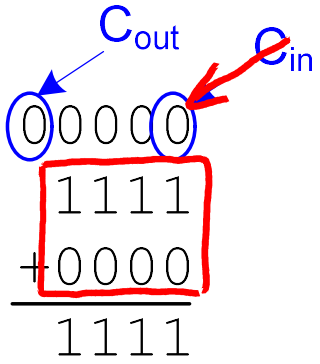
- Footless dual-rail domino
  - Very fast, but large area and power hungry
  - Similar delay for S and C_{out} → used in very fast multipliers

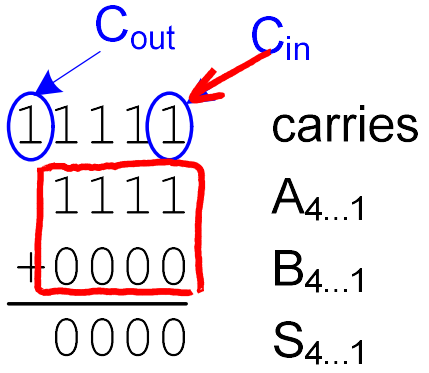


Carry Propagate Adders (CPA)

- Multiple-bit (N) adder
  - Each sum bit depends on all previous carries
  - How do we compute all these carries quickly?

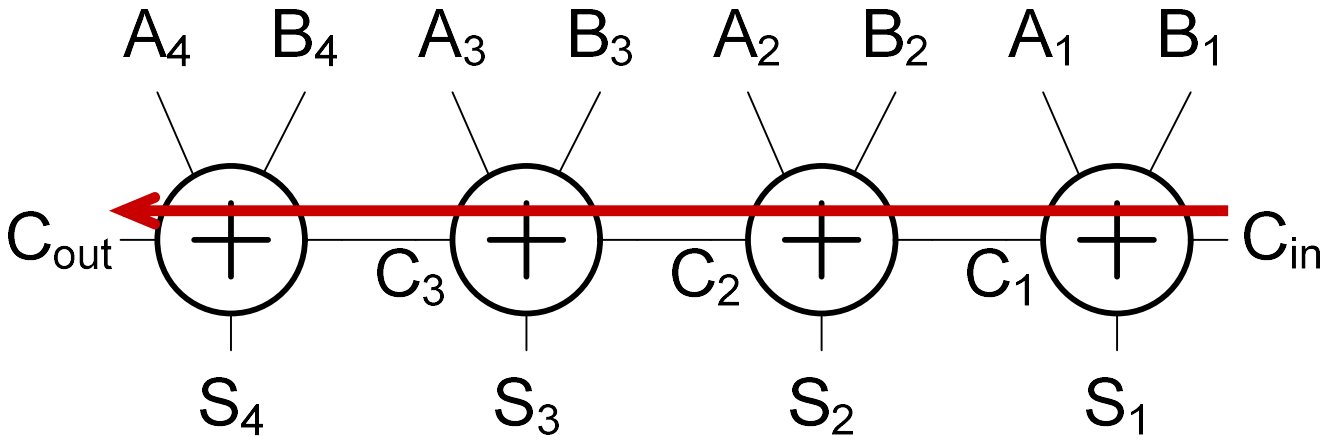






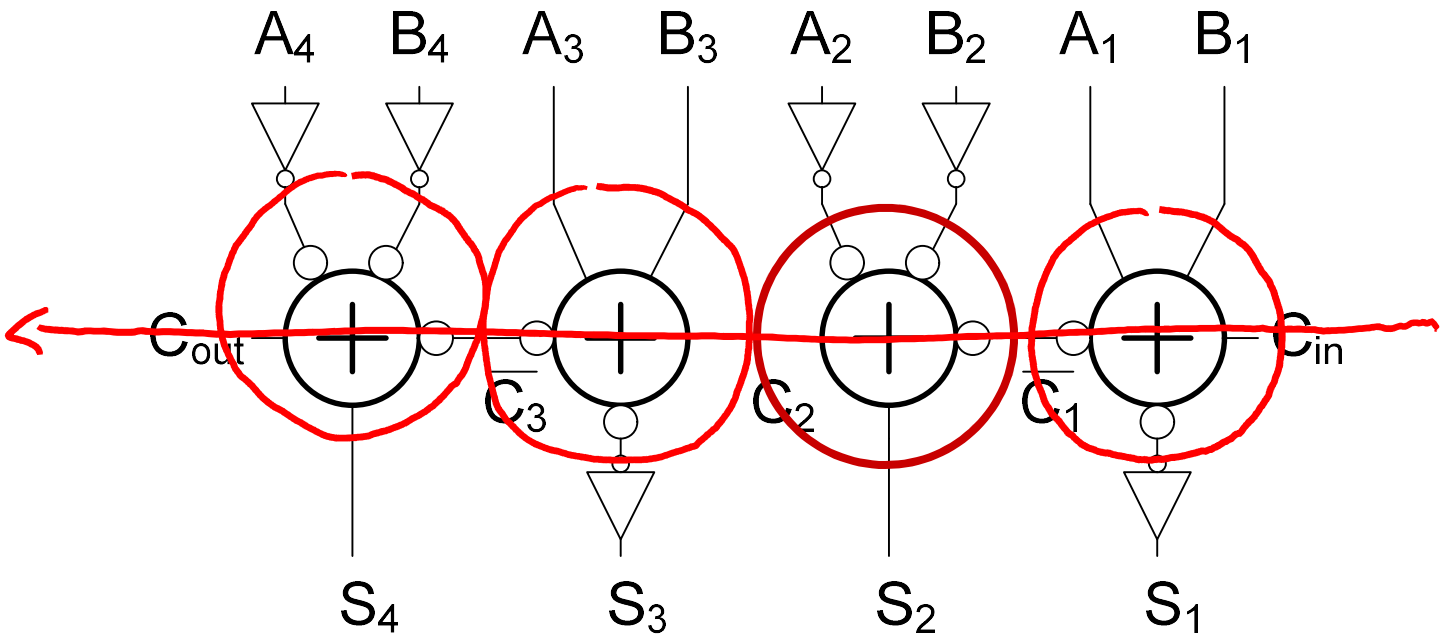
Carry Ripple Adder

- Simplest design: cascade full adders
  - Critical path goes from C_{in} to C_{out}
  - Design full adders with fast carry delay



Inversions

- Critical path passes through majority gate
  - Built from minority + inverter
  - Eliminate inverter and use inverting full adder



Generate/Propagate/Kill

- Equations often factored into GPK defining what happens to carries
  - Generate: C_{out} = 1 independent of Cin **(G = A • B)**
  - Propagate: C_{out} = 1 if and only if C_{in} = 1 **(P = A ⊕ B)**
  - Kill: C_{out} = 0 independent of C_{in}

A	B	C	G	P	K	C_{out}	S
0	0	0	0	0	1	0	0
		1				0	1
0	1	0	0	1	0	0	1
		1				1	0
1	0	0	0	1	0	0	1
		1				1	0
1	1	0	1	0	0	1	0
		1				1	1

Generate/Propagate

- Base case (single-bit)

$$G_{i:i} \equiv G_i = A_i \bullet B_i \qquad G_{0:0} \equiv G_0 = C_{in}$$

$$P_{i:i} \equiv P_i = A_i \oplus B_i \qquad P_{0:0} \equiv P_0 = 0$$

- G & P for multiple bits spanning i:j

$$G_{i:j} = G_{i:k} + P_{i:k} \bullet G_{k-1:j}$$

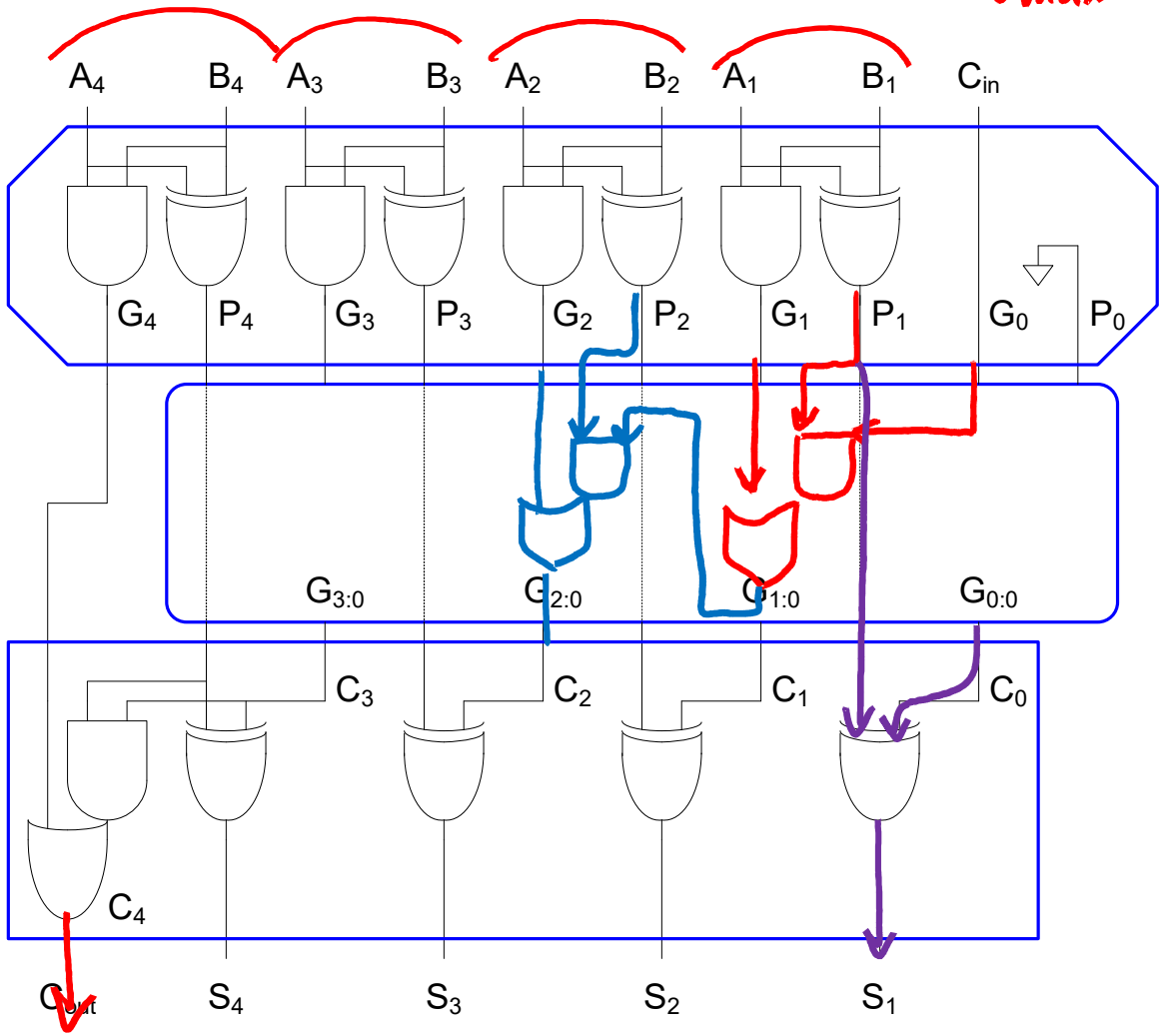
$$P_{i:j} = P_{i:k} \bullet P_{k-1:j}$$

Valency-2 group PG Logic

- Sum

$$C_{i-1} = G_{i-1:0}, \quad S = P \oplus C, \quad S_i = P_i \oplus G_{i-1:0}$$

PG Logic



$\Delta t_{max} = \Delta t_{PG} + 3 \cdot \Delta t_{AO} + \Delta t_{XOR}$

4-1

1: Bitwise PG logic

2: Group PG logic

3: Sum logic

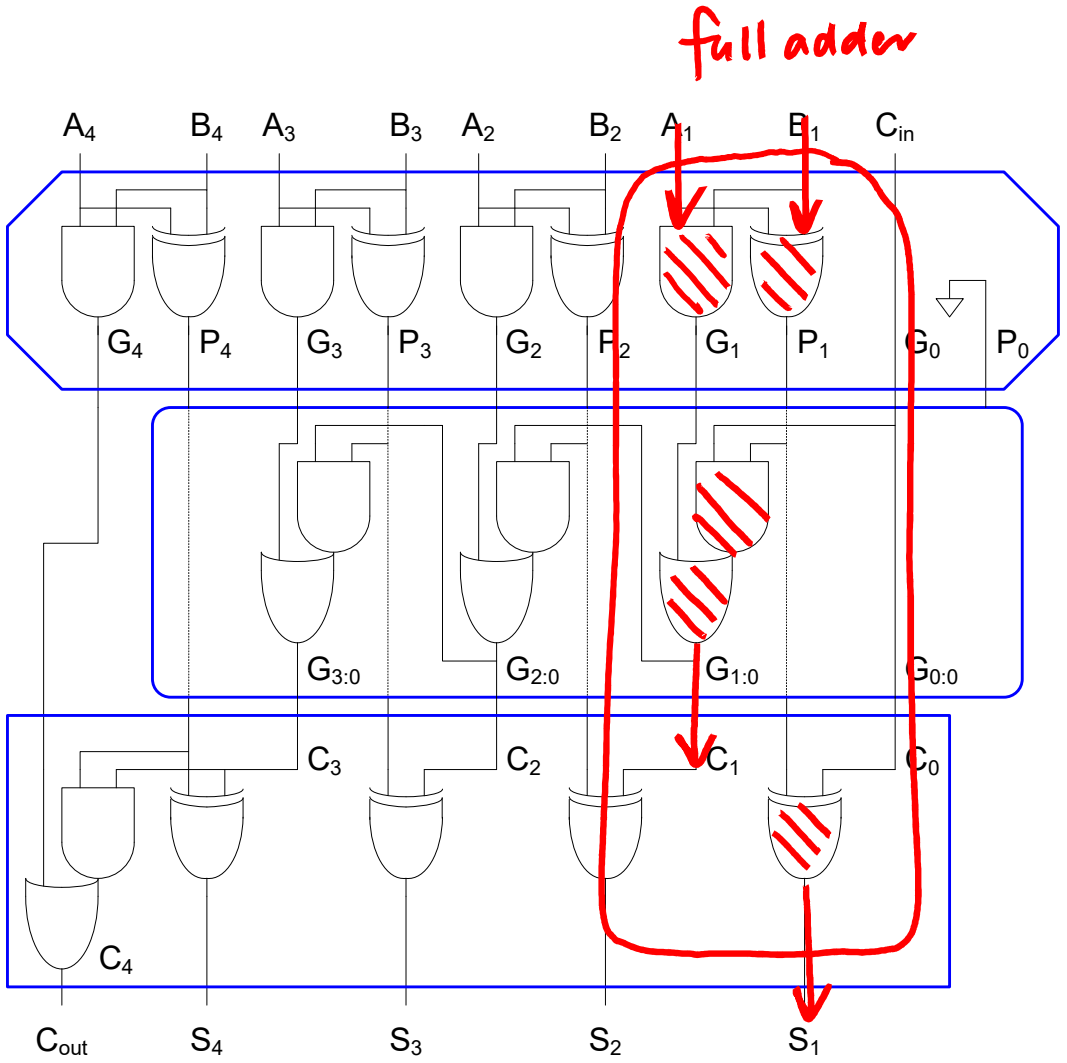
Carry-Ripple Revisited

$$C_i = A_i B_i + (A_i + B_i) C_{i-1}$$
$$= A_i B_i + (A_i \oplus B_i) C_{i-1}$$
$$= G_i + P_i C_{i-1}$$

$$\boxed{G_{i:0} = G_i + P_i \bullet G_{i-1:0}}$$

$$S_i = P_i \oplus G_{i-1:0}$$

$$C_{out} = G_{i:0}$$



Carry-Ripple PG Diagram

- For N-bit adder

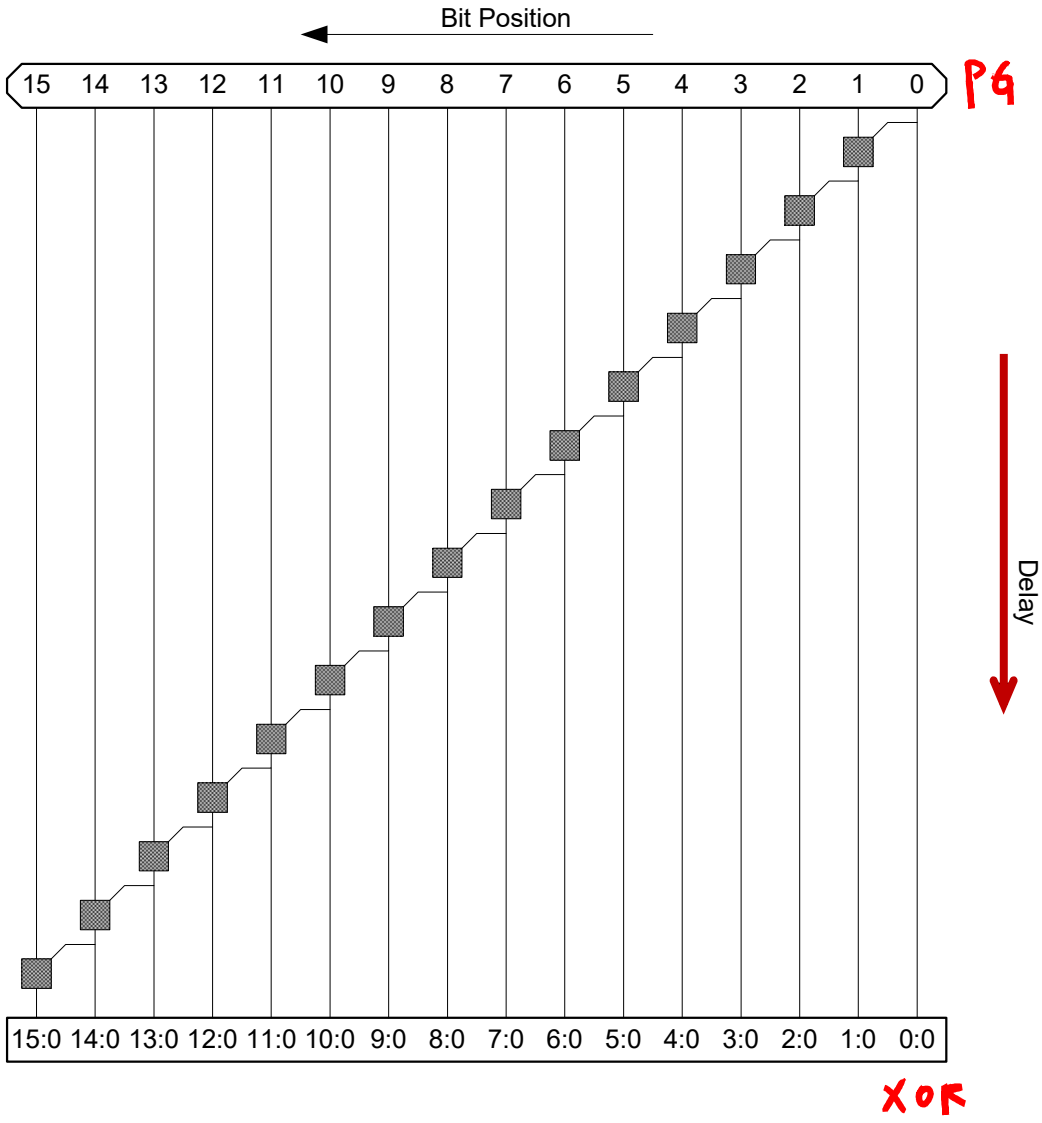
N=15 in this figure

$$t_{\text{ripple}} = t_{pg} + (N-1)t_{AO} + t_{\text{xor}}$$

t_{pg} : delay of 1-bit propagate /generate gate

t_{AO} : delay of AND-OR gate in grey cell

t_{xor} : delay of final sum XOR

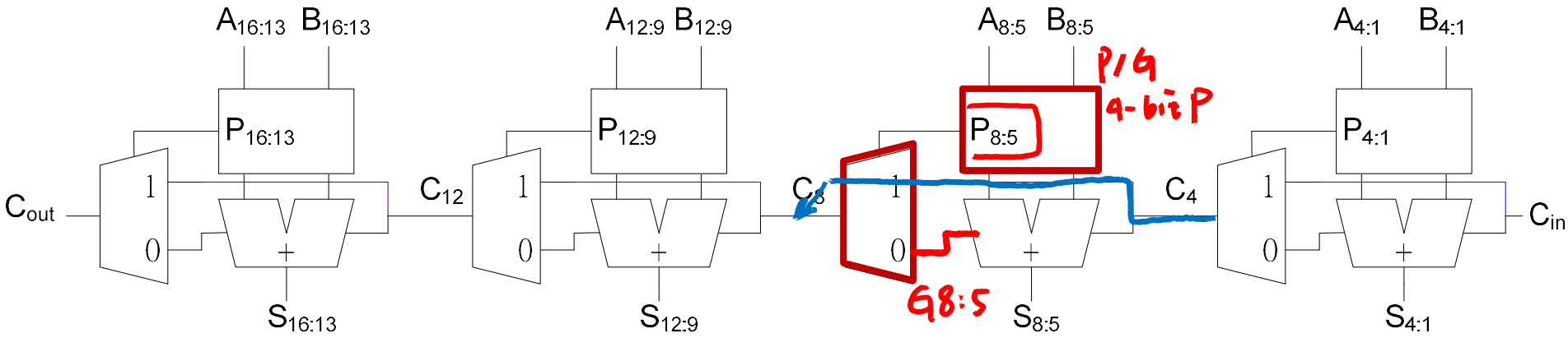


Carry-Skip Adder

- Carry-ripple is slow through all stages
- Carry-skip allows carry to skip over groups of n bits
  - Decision based on n-bit propagate signal

$$G_{16:0} = G_{16:13} + P_{16:13} \bullet G_{12:0}$$

$$G_{8:0} = G_{8:5} + P_{8:5} \bullet G_{4:0}$$

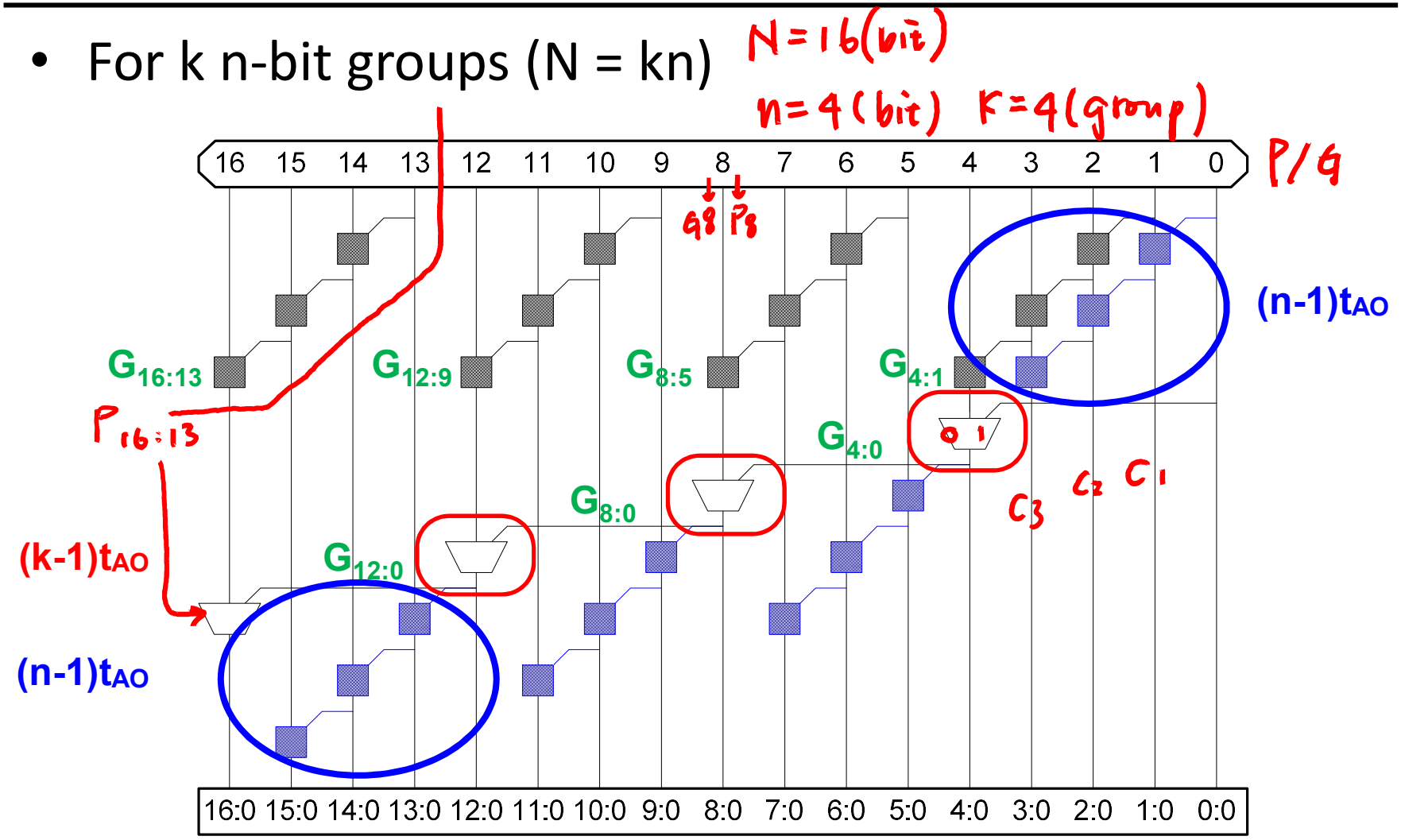


$$G_{12:0} = G_{12:9} + P_{12:9} \bullet G_{8:0}$$

$$G_{4:0} = G_{4:1} + P_{4:1} \bullet G_{0:0}$$

Carry-Skip PG Diagram

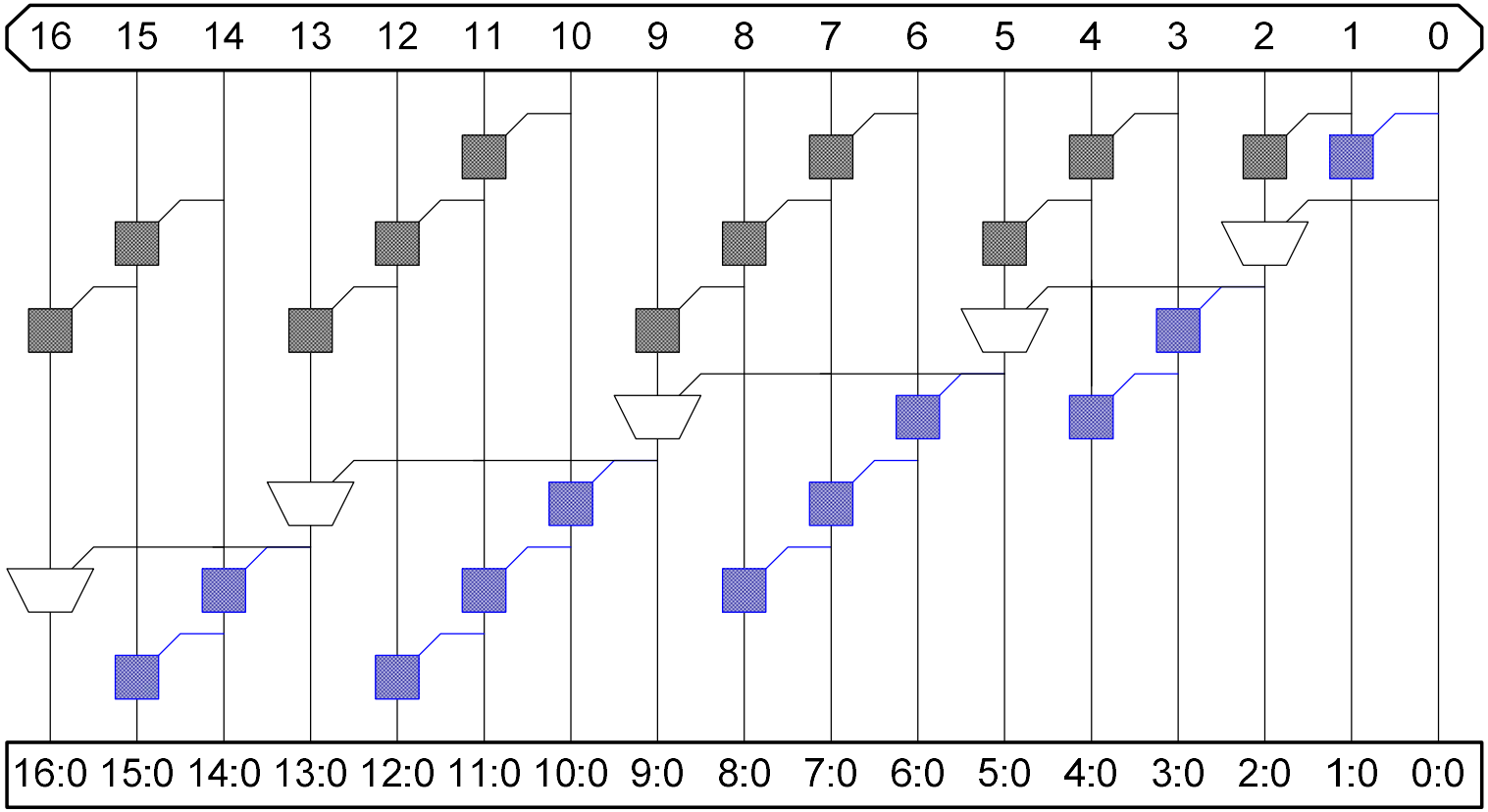
- For k n-bit groups (N = kn)



$$t_{\text{skip}} = t_{pg} + \left[2(n-1) + (k-1)\right] t_{AO} + t_{\text{xor}}$$

Variable Group Size

- Delay grows as O(sqrt(N))
  - [2,3,4,4,3] saves 2 levels of logic compared to [4,4,4,4]

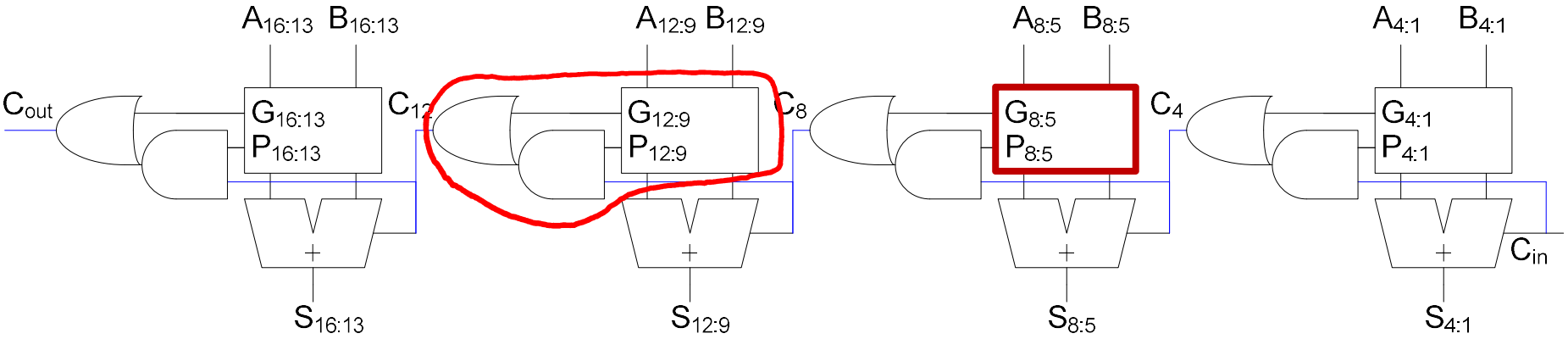


Carry-Lookahead Adder

- Computes both $P_{i:j}$ and $G_{i:j}$ for many bits in parallel.
- Uses higher-valency cells with more than two inputs

$$G_{16:0} = G_{16:13} + P_{16:13} \bullet G_{12:0}$$

$$G_{8:0} = G_{8:5} + P_{8:5} \bullet G_{4:0}$$

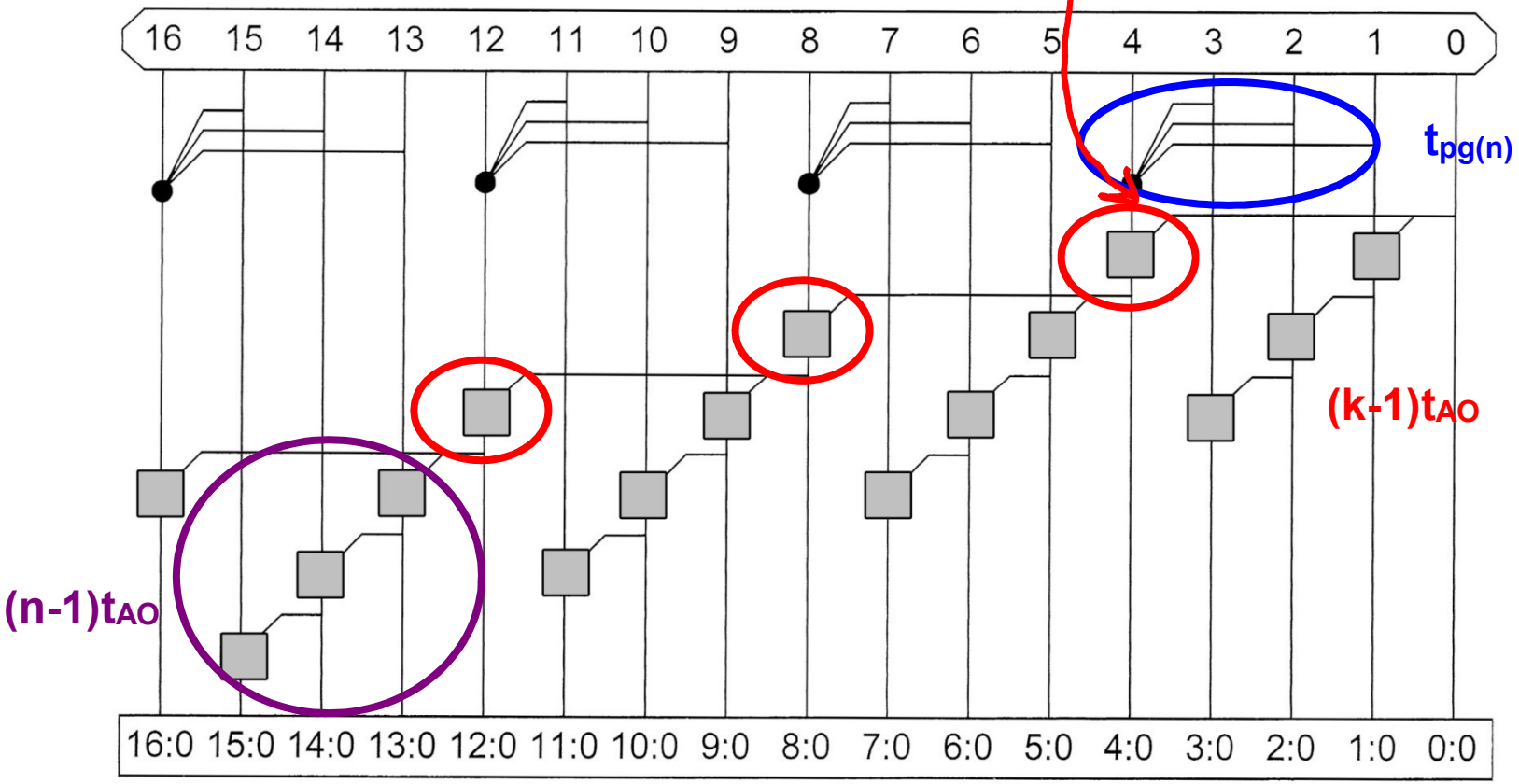


$$G_{12:0} = G_{12:9} + P_{12:9} \bullet G_{8:0}$$

$$G_{4:0} = G_{4:1} + P_{4:1} \bullet G_{0:0}$$

Carry-Lookahead PG Diagram

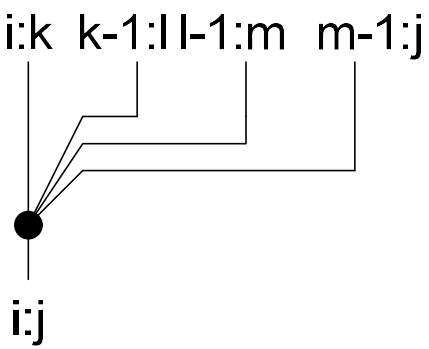
- For k n-bit groups (N = kn)

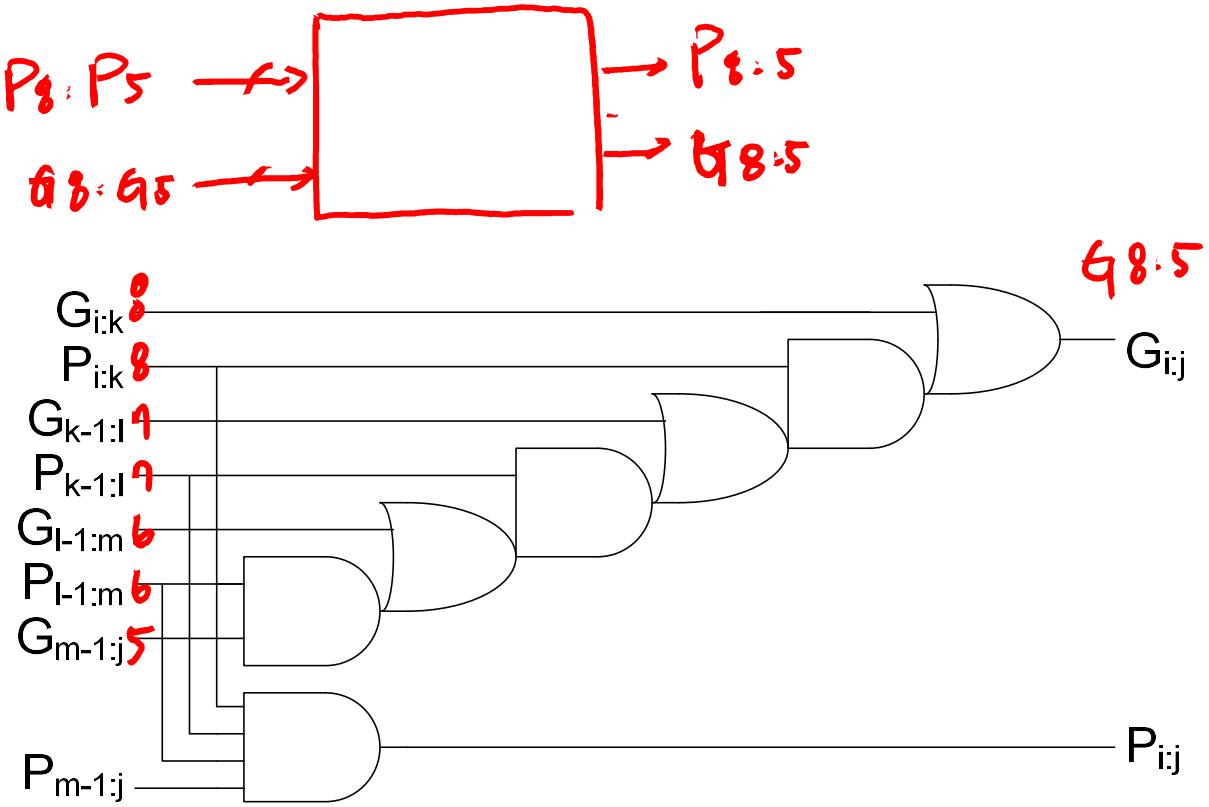


$$t_{cla} = t_{pg} + t_{pg(n)} + \left[(n-1)+(k-1)\right]t_{AO} + t_{\text{xor}}$$

Higher-Valency Cells

$$Delay: t_{pg(n)}$$

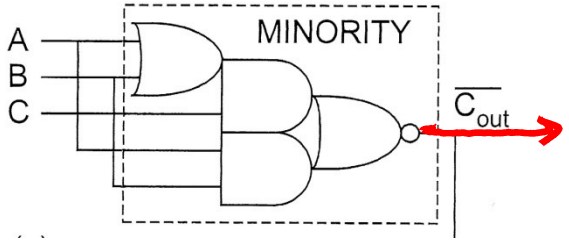


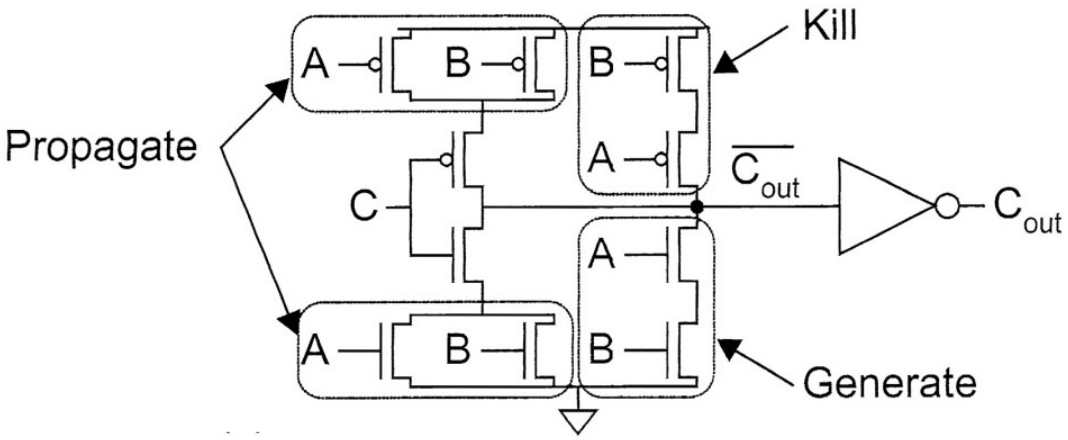


Manchester Carry Chain Adder

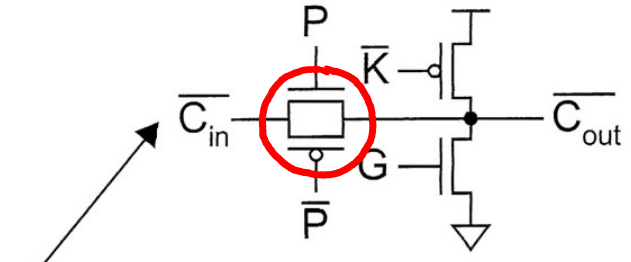
reduce transistor count (power reduction)

speed-up

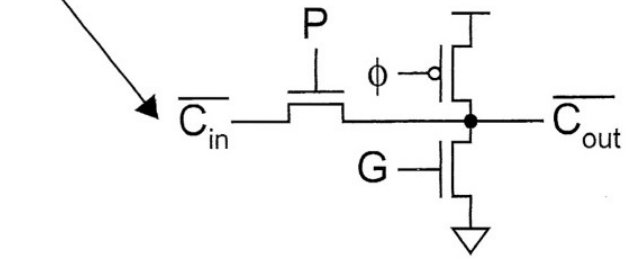




Pass Transistor Logic



Static

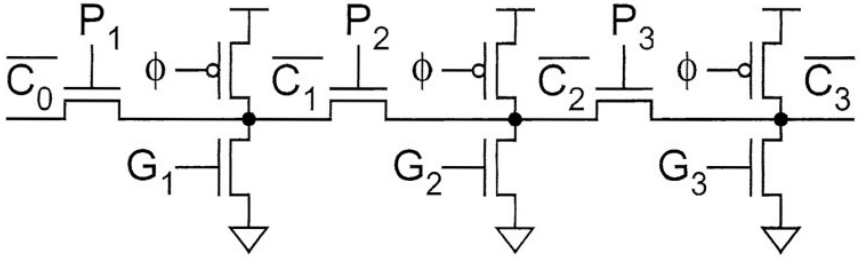


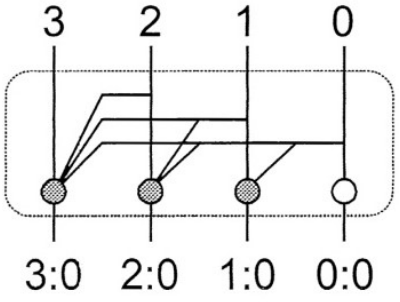
Dynamic

Manchester Carry Chain

- Valency-4 generate

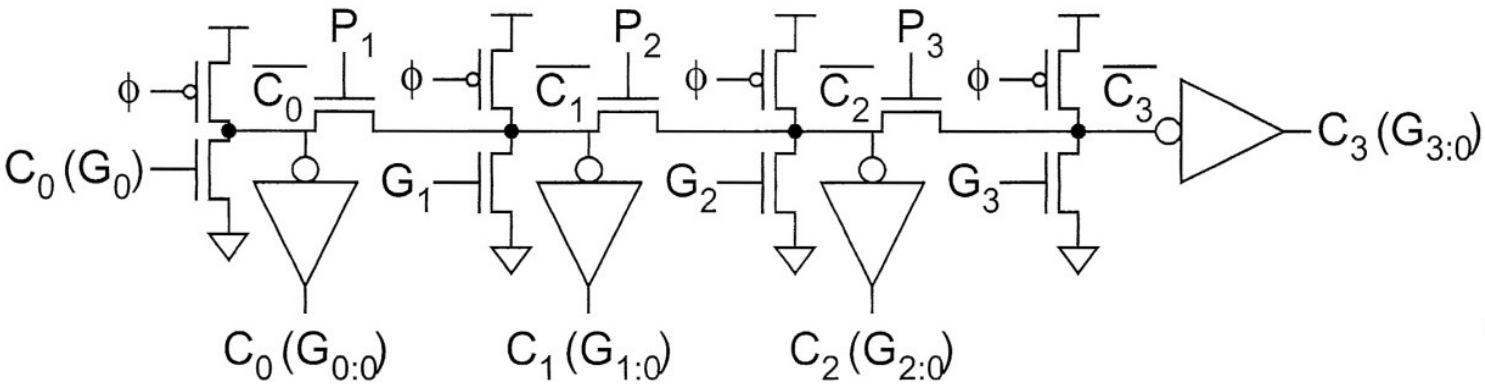
$$C_0 = G_{0:0} = C_0$$
$$C_1 = G_{1:0} = G_1 + P_1 C_0$$
$$C_2 = G_{2:0} = G_2 + P_2 \left(G_1 + P_1 C_0 \right)$$
$$C_3 = G_{3:0} = G_3 + P_3 \left(G_2 + P_2 \left(G_1 + P_1 C_0 \right) \right)$$

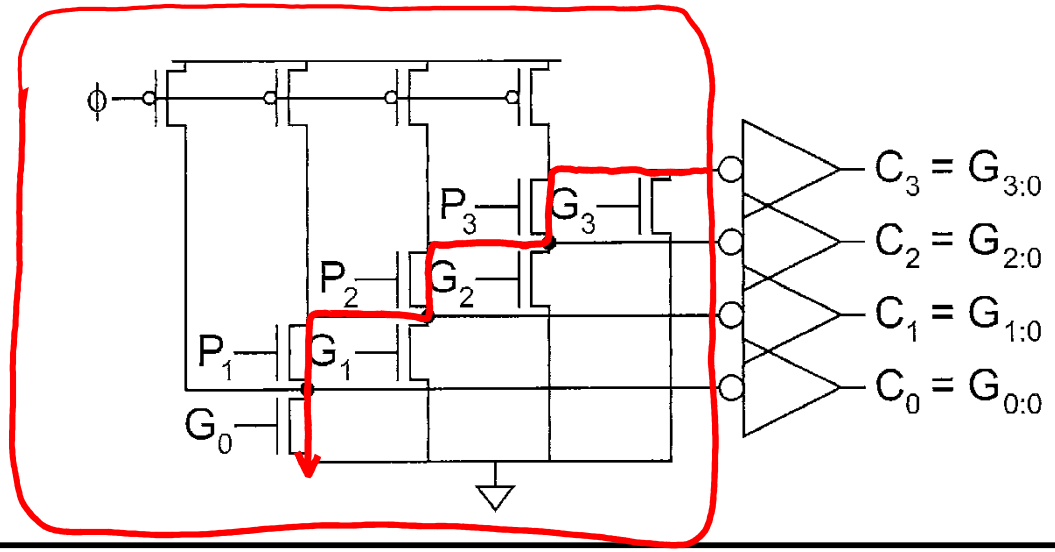




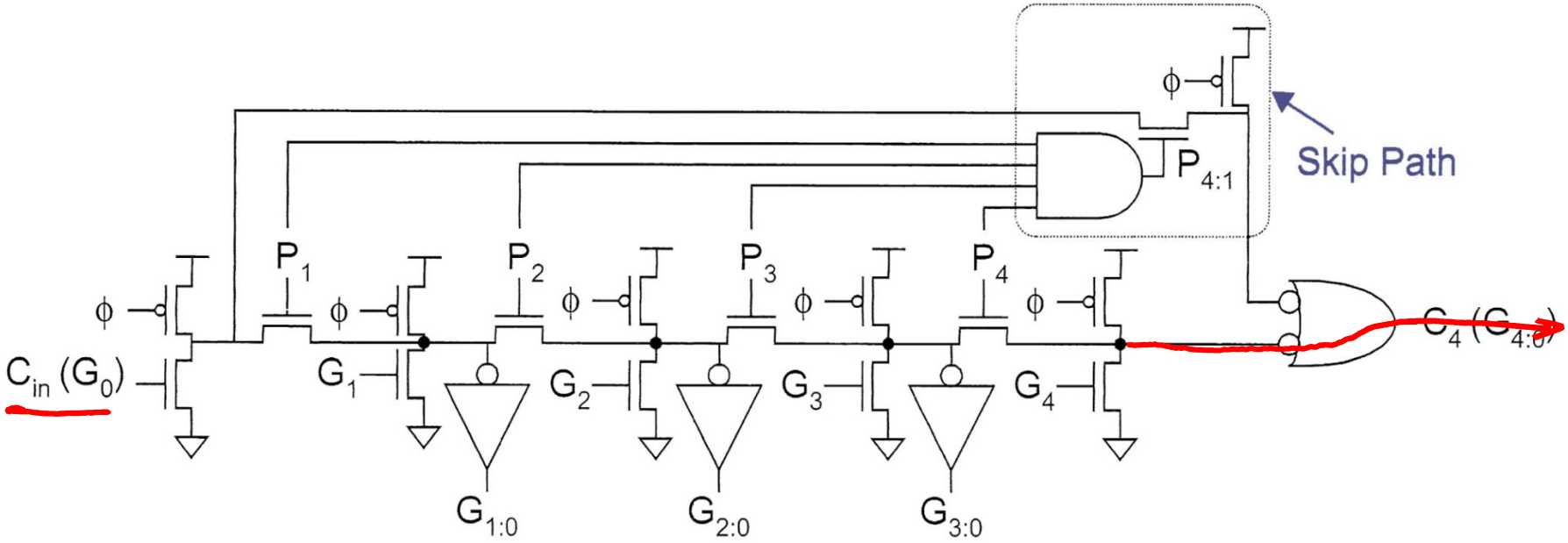
Manchester Carry Chain – Domino

- Multiple outputs





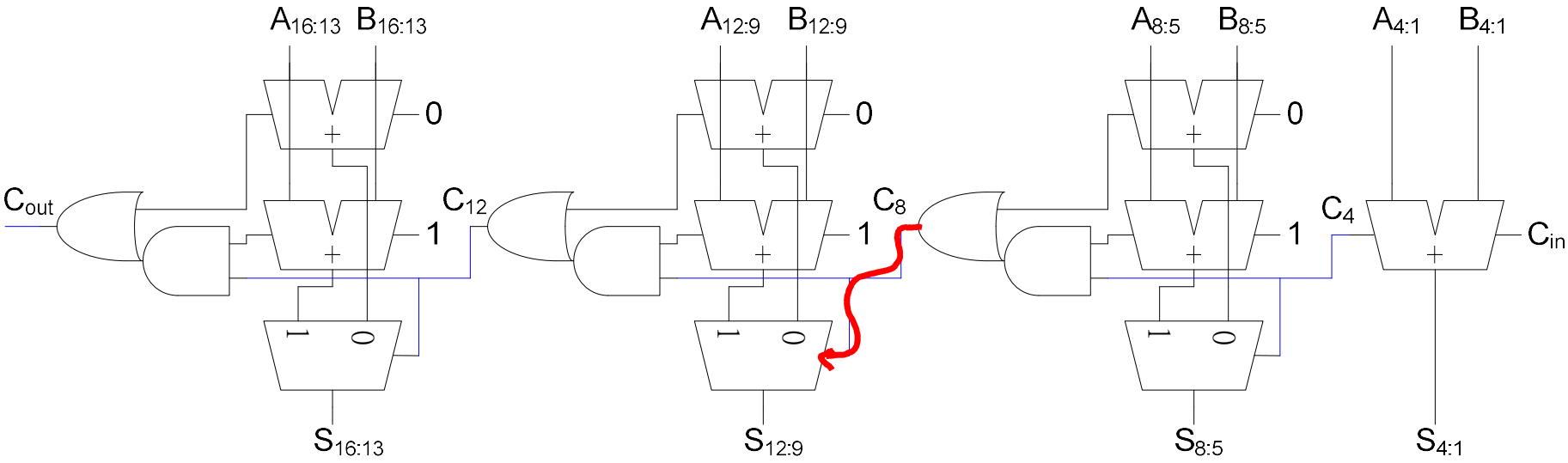
Carry-Skip Manchester Stage



Carry-Select Adder

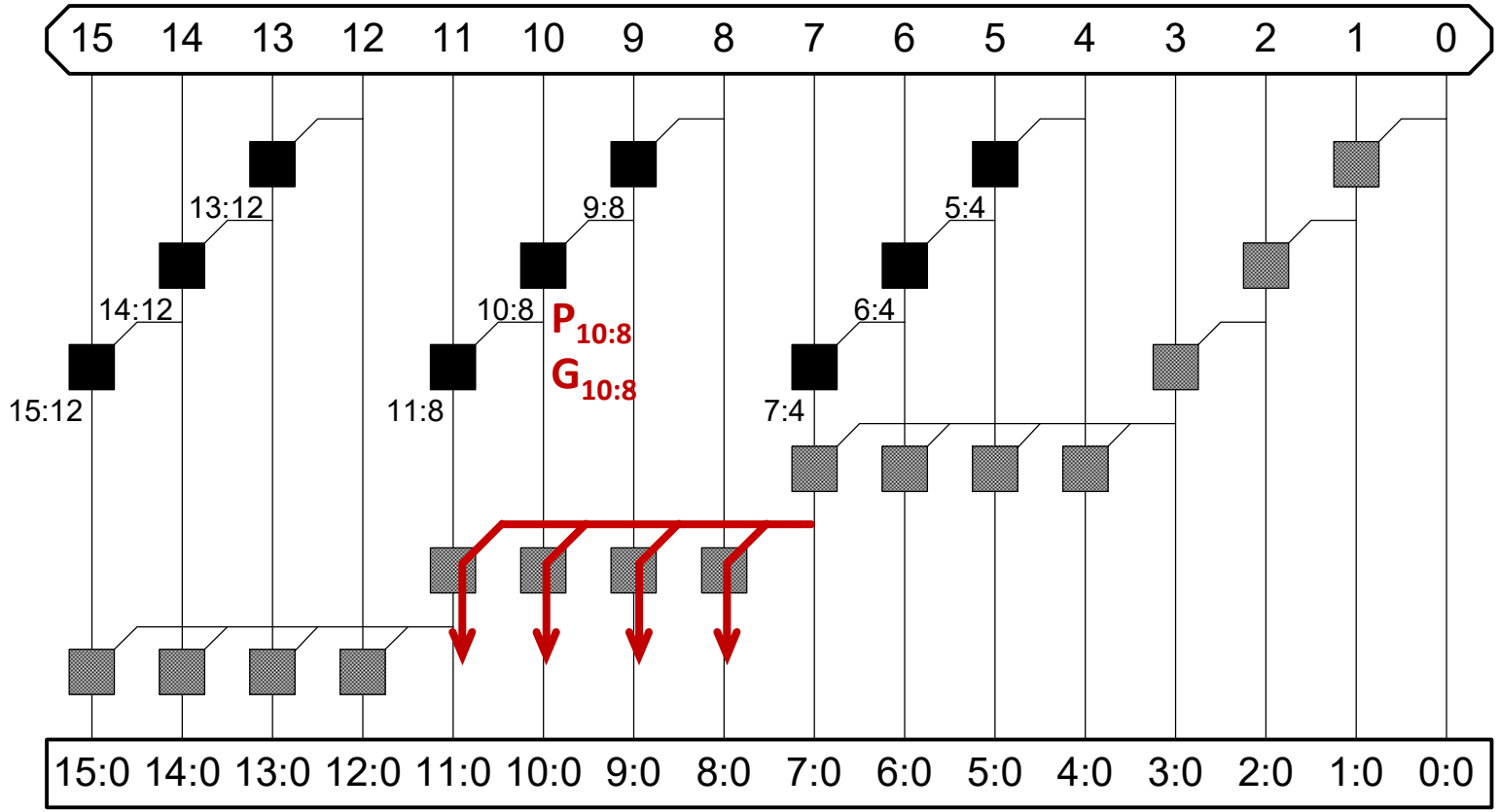
- Trick for critical paths dependent on late input C_{in}
  - Precompute two possible outputs for C_{in} = 0, 1
  - Select proper output when C_{in} arrives
- Carry-select adder precomputes n-bit sum

$$t_{\text{select}} = t_{pg} + \left[n + (k-2)\right] t_{AO} + t_{\text{mux}}$$



Carry-Increment Adder

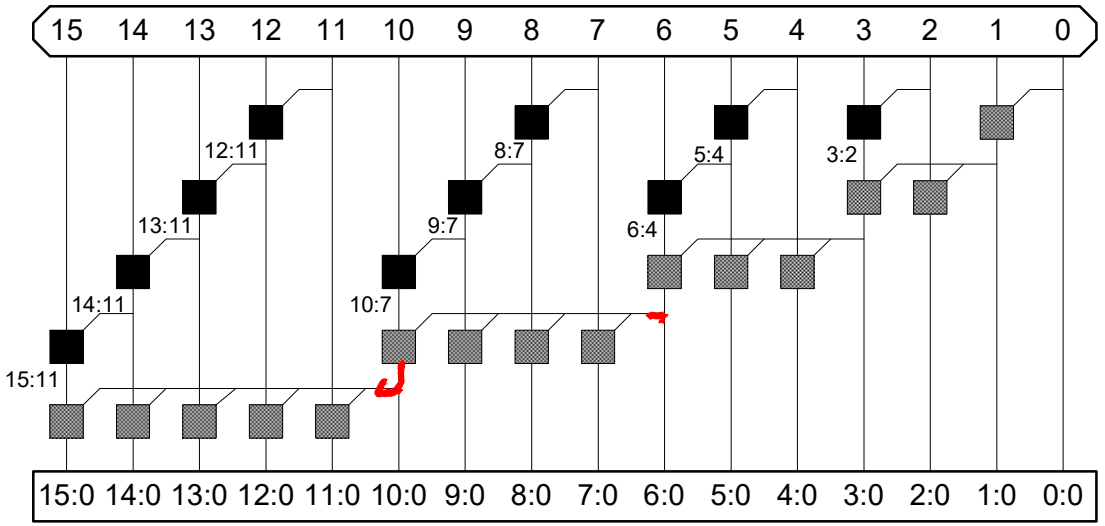
- Factor initial PG and final XOR out of carry-select



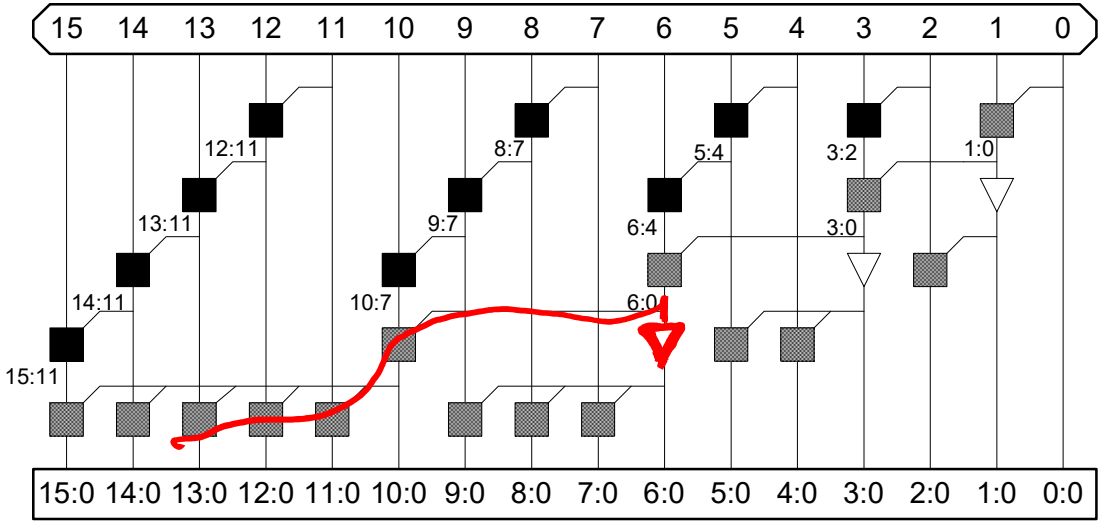
$$t_{\text{increment}} = t_{pg} + \left[(n-1) + (k-1)\right] t_{AO} + t_{\text{xor}}$$

Variable Group Size

- Fanout increases with group size



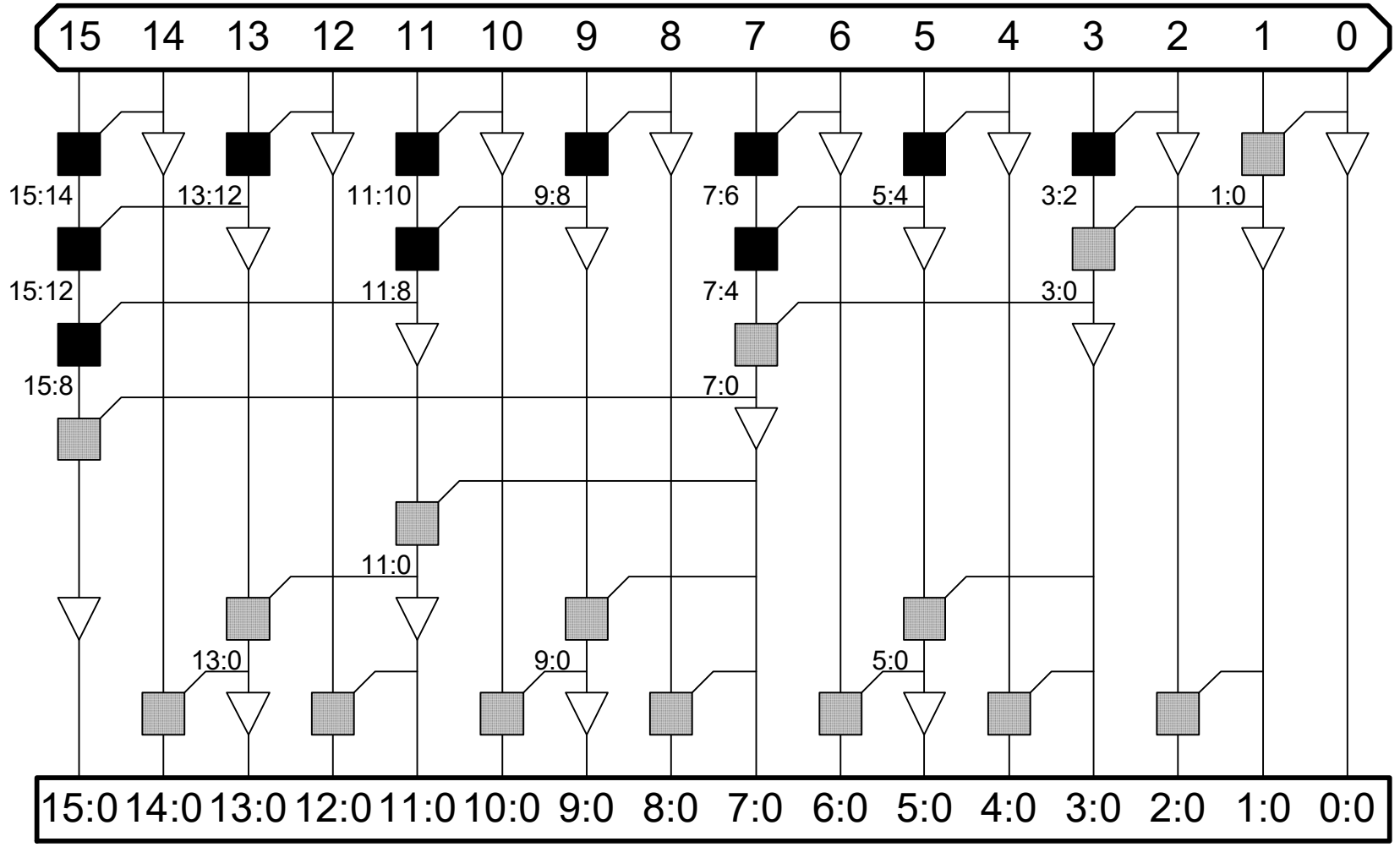
- Buffer noncritical signals
- Reduce branch effort



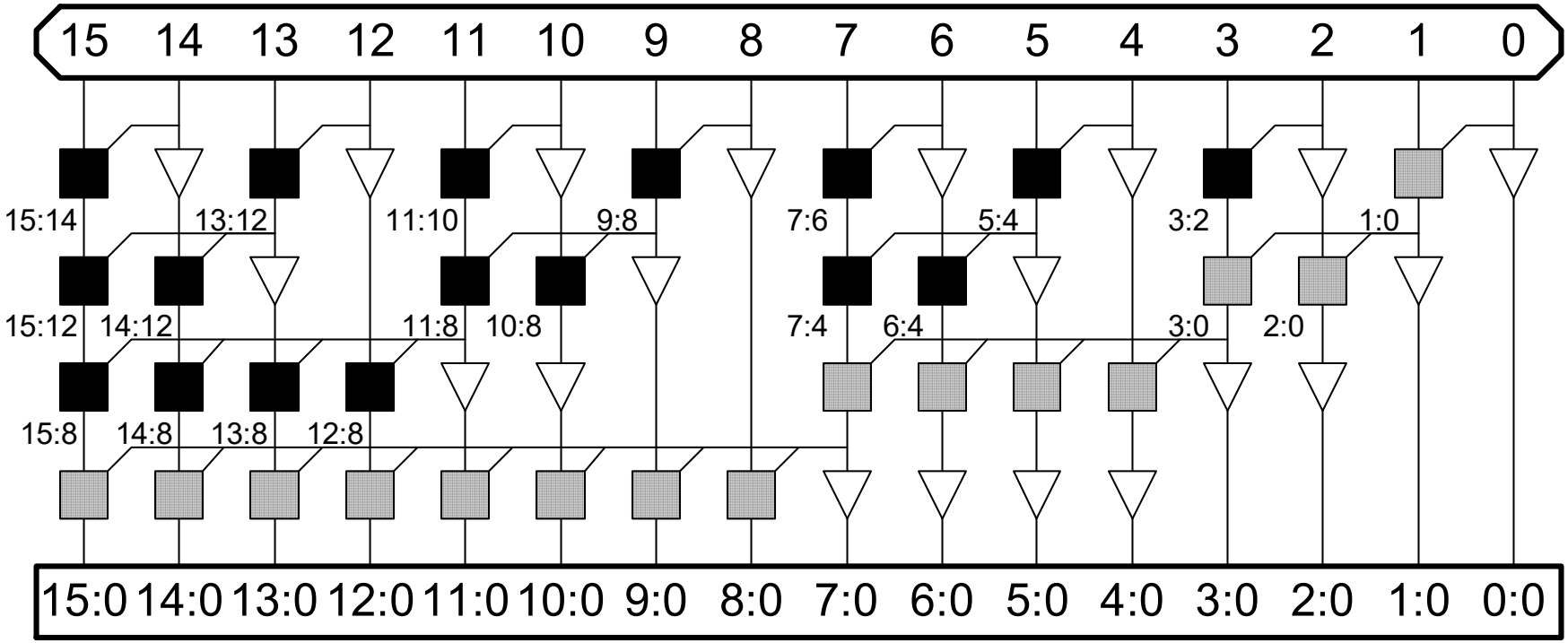
Tree Adder

- If lookahead is good, lookahead across lookahead!
  - Recursive lookahead gives O(log N) delay
- Many variations on tree adders

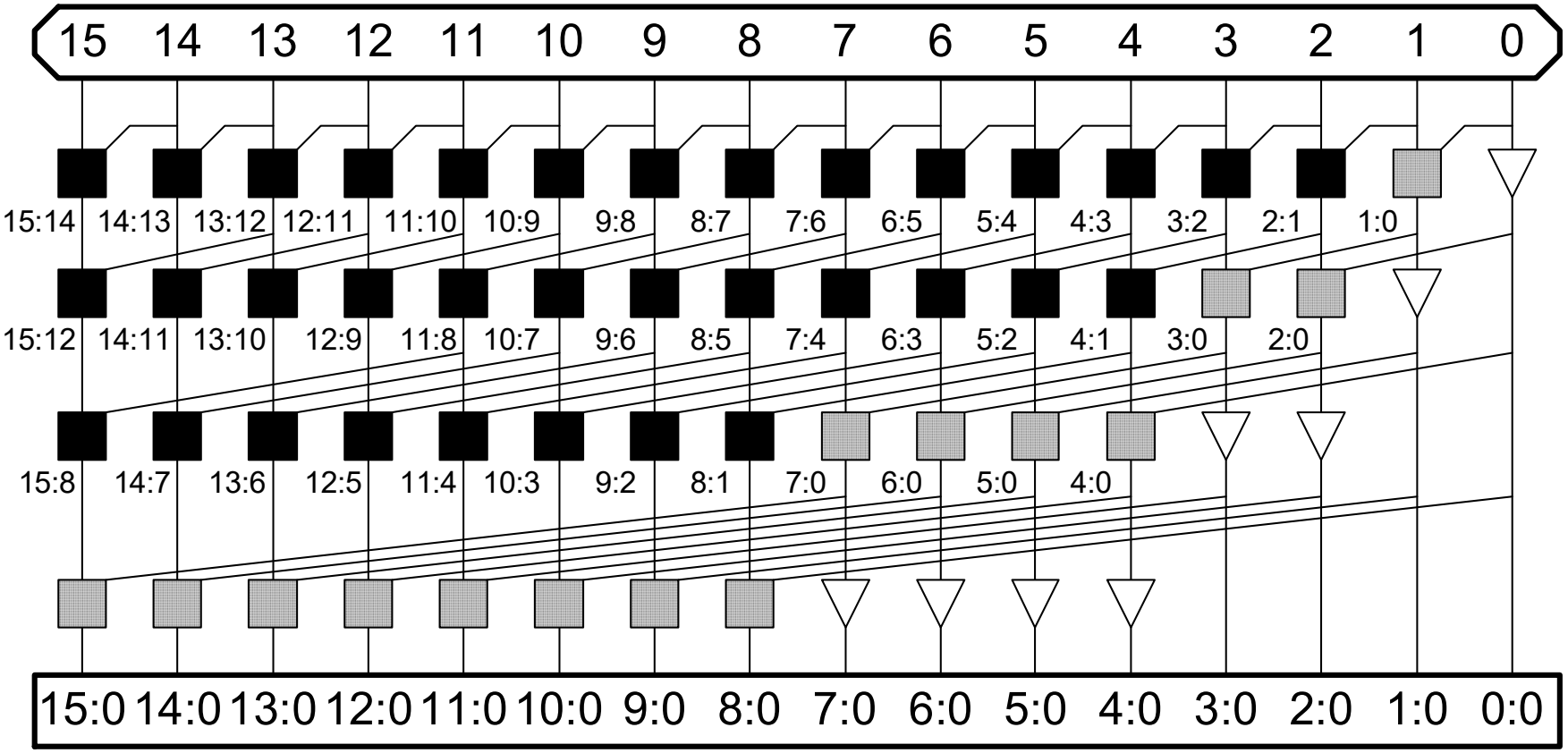
Brent-Kung


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15:14
13:12
11:10
9:8
7:6
5:4
3:2
1:0
15:12
11:8
7:4
3:0
15:8
7:0
11:0
13:0
9:0
5:0
15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0

Sklansky


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15:14 13:12 11:10 9:8 7:6 5:4 3:2 1:0
15:12 14:12 11:8 10:8 7:4 6:4 3:0 2:0
15:8 14:8 13:8 12:8
15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0

Kogge-Stone

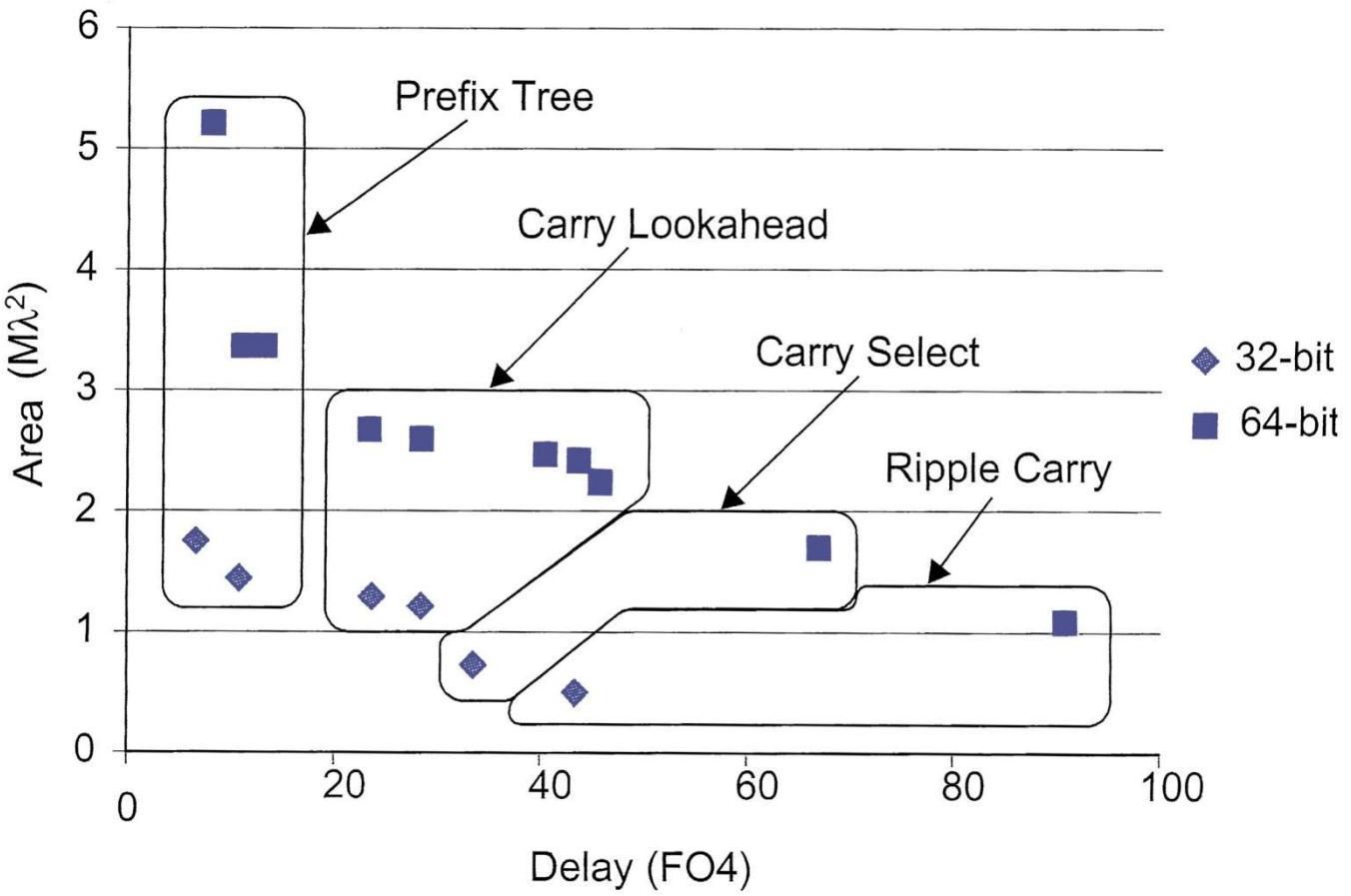

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15:14 14:13 13:12 12:11 11:10 10:9 9:8 8:7 7:6 6:5 5:4 4:3 3:2 2:1 1:0
15:12 14:11 13:10 12:9 11:8 10:7 9:6 8:5 7:4 6:3 5:2 4:1 3:0 2:0
15:8 14:7 13:6 12:5 11:4 10:3 9:2 8:1 7:0 6:0 5:0 4:0
15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0

Summary

- Various structures offer area/power/delay tradeoffs
- Choose the best one for your application

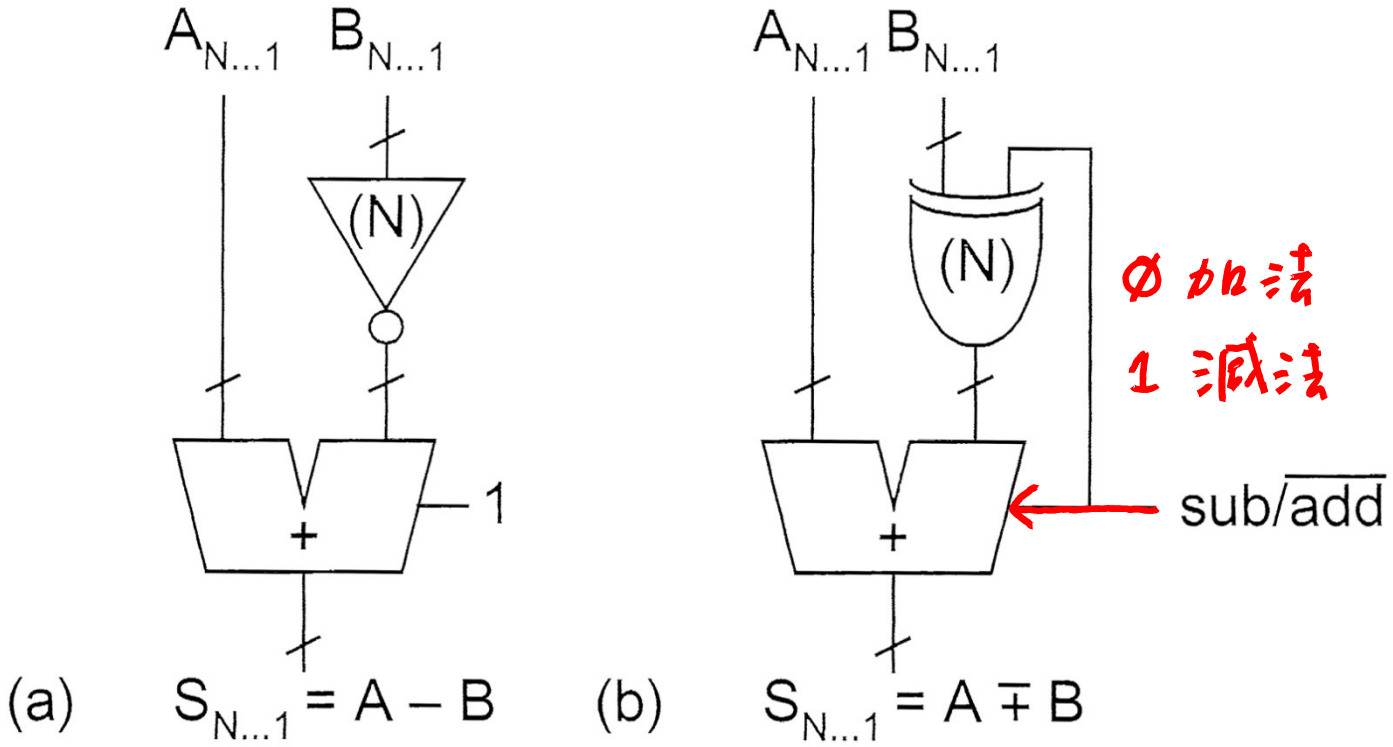
Architecture	Classification	Logic Levels	Max Fanout	Tracks	Cells
Carry-Ripple		N-1	1	1	N
Carry-Skip n=4		N/4 + 5	2	1	1.25N
Carry-Inc. n=4		N/4 + 2	4	1	2N
Brent-Kung	(L-1, 0, 0)	$2\log_2 N - 1$	2	1	2N
Sklansky	(0, L-1, 0)	$\log_2 N$	N/2 + 1	1	$0.5\ N\log_2 N$
Kogge-Stone	(0, 0, L-1)	$\log_2 N$	2	N/2	$N\log_2 N$

Summary


Prefix Tree
Carry Lookahead
Carry Select
Ripple Carry
32-bit
64-bit
Area (Mλ²)
Delay (FO4)
0
1
2
3
4
5
6
0
20
40
60
80
100

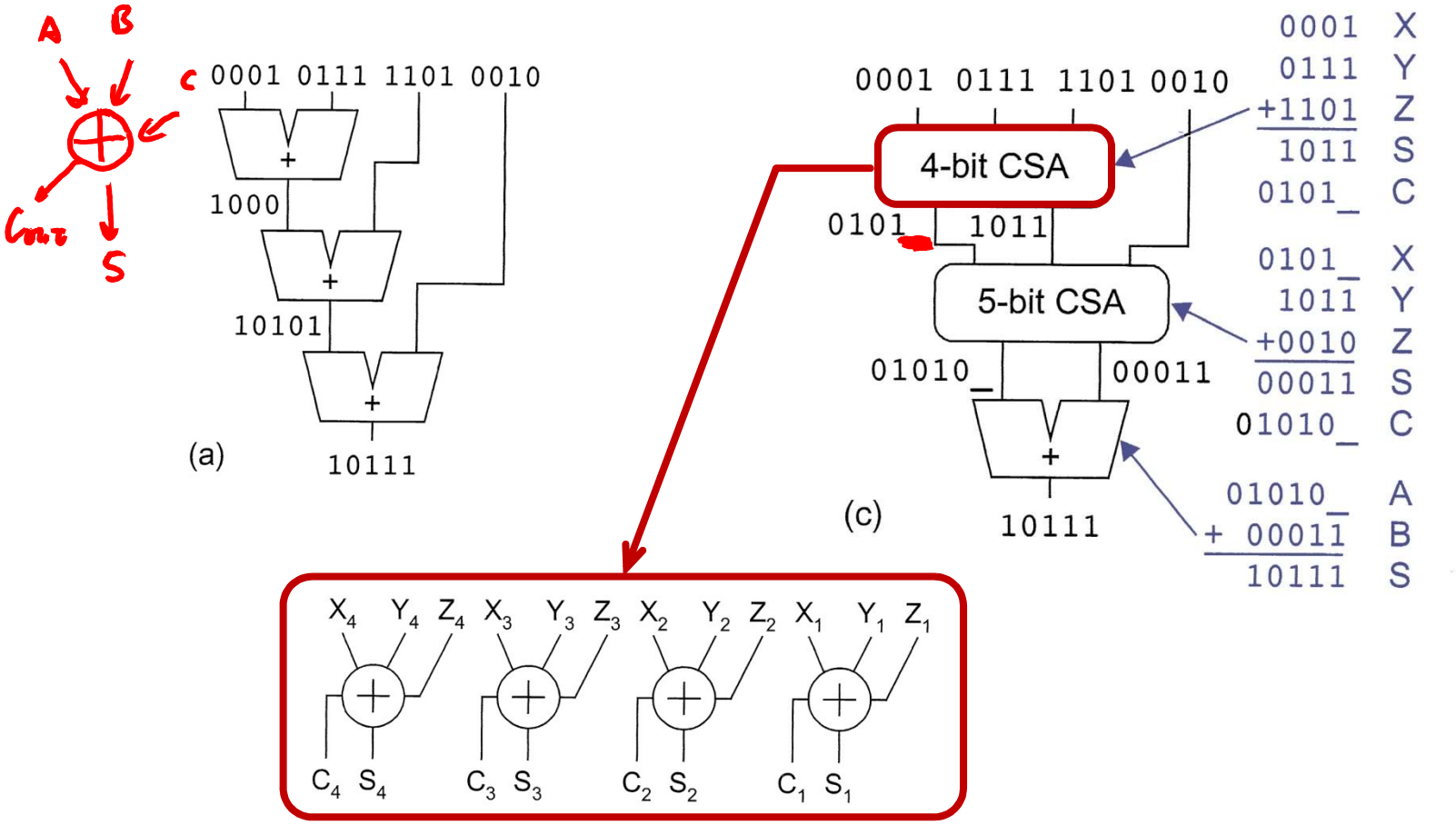
Subtraction

- 2's complement



Multiple-Input Addition

- Carry-Save Adder (CSA)
- K words can be summed with k-2 CSA and 1 CPA

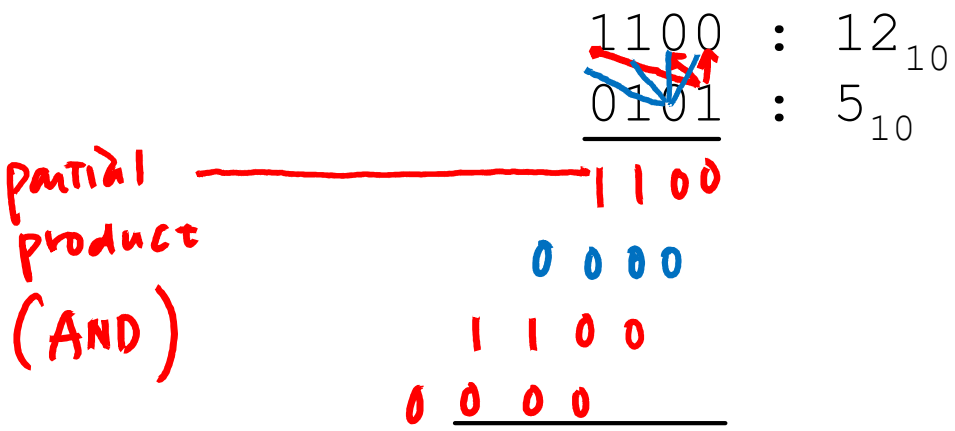


Outline

- Addition/Subtraction
- **Multiplication**

- One/Zero Detectors
- Comparators
- Counters
- Shifters

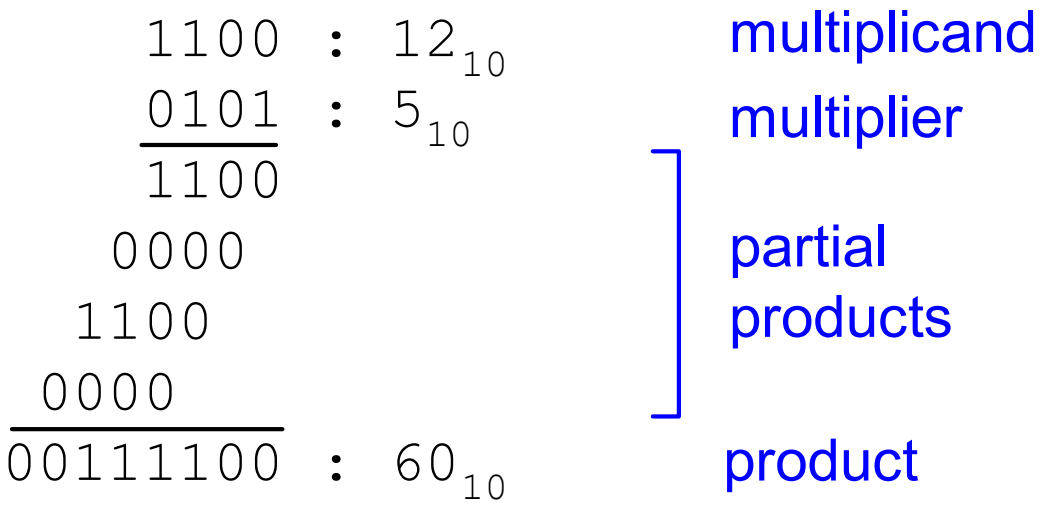
Example



Example

$$
\begin{array}{rl}
1100 & : 12_{10} \\
0101 & : 5_{10} \\
\hline
1100 & \\
0000 & \\
1100 & \\
0000 & \\
\hline
00111100 & : 60_{10}
\end{array}
$$

Example

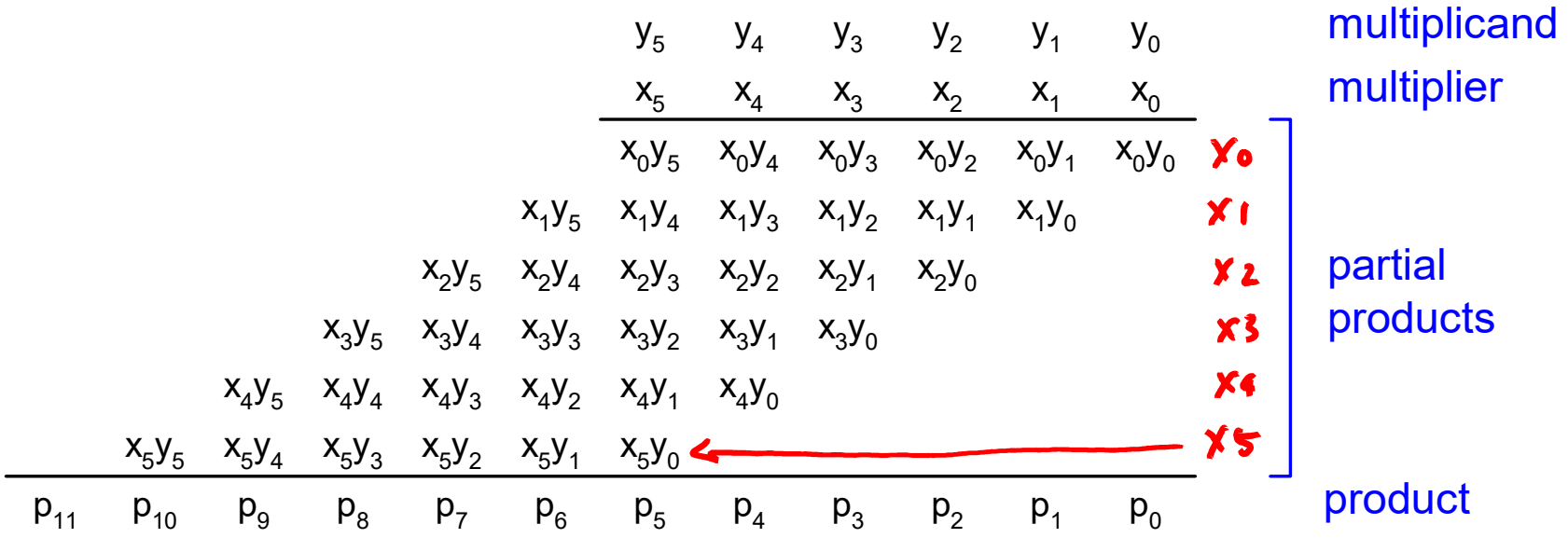


- M x N-bit multiplication
  - Produce N M-bit partial products
  - Sum these to produce (M+N–1)-bit product

General Form

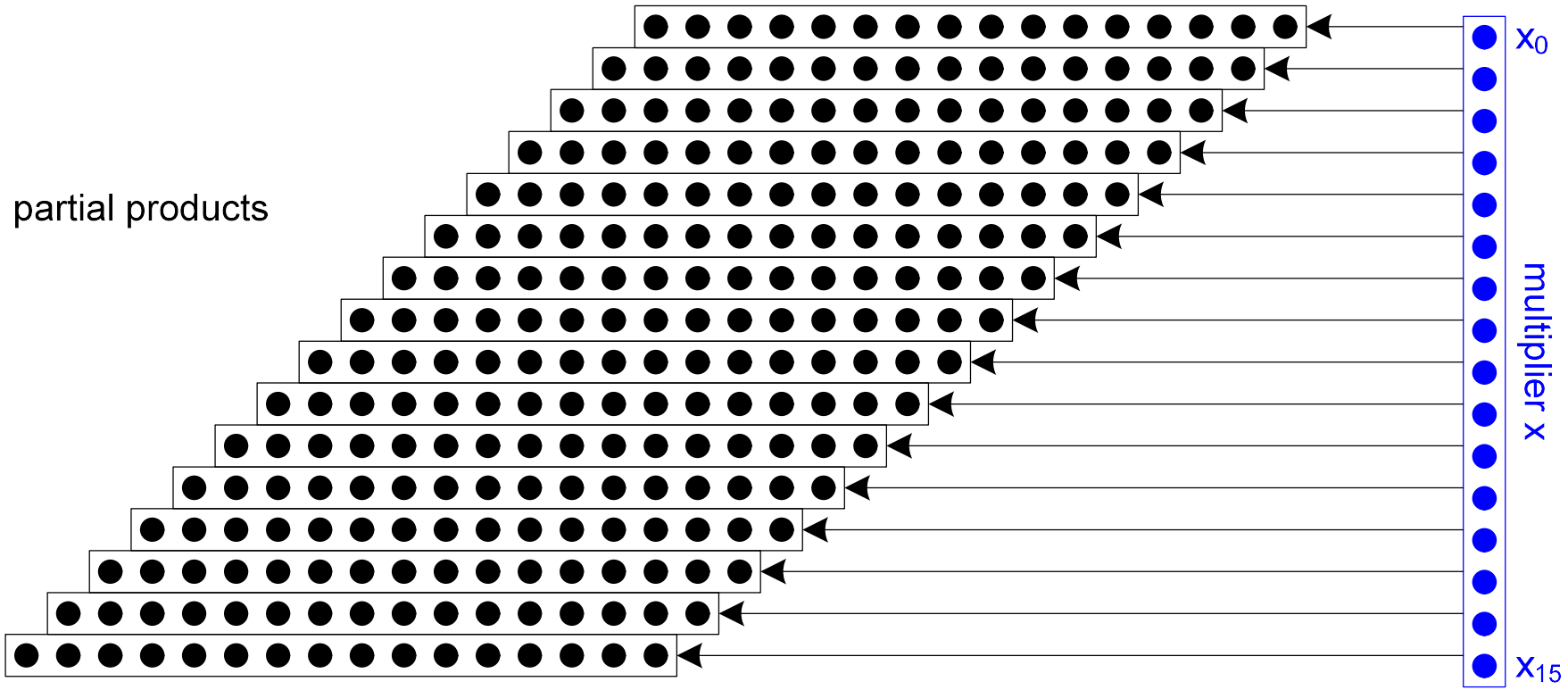
- Multiplicand: $Y = (y_{M-1}, y_{M-2}, \ldots, y_1, y_0)$
- Multiplier: $X = (x_{N-1}, x_{N-2}, \ldots, x_1, x_0)$
- Product:

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



Dot Diagram

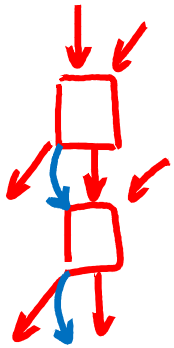
- Each dot represents a bit

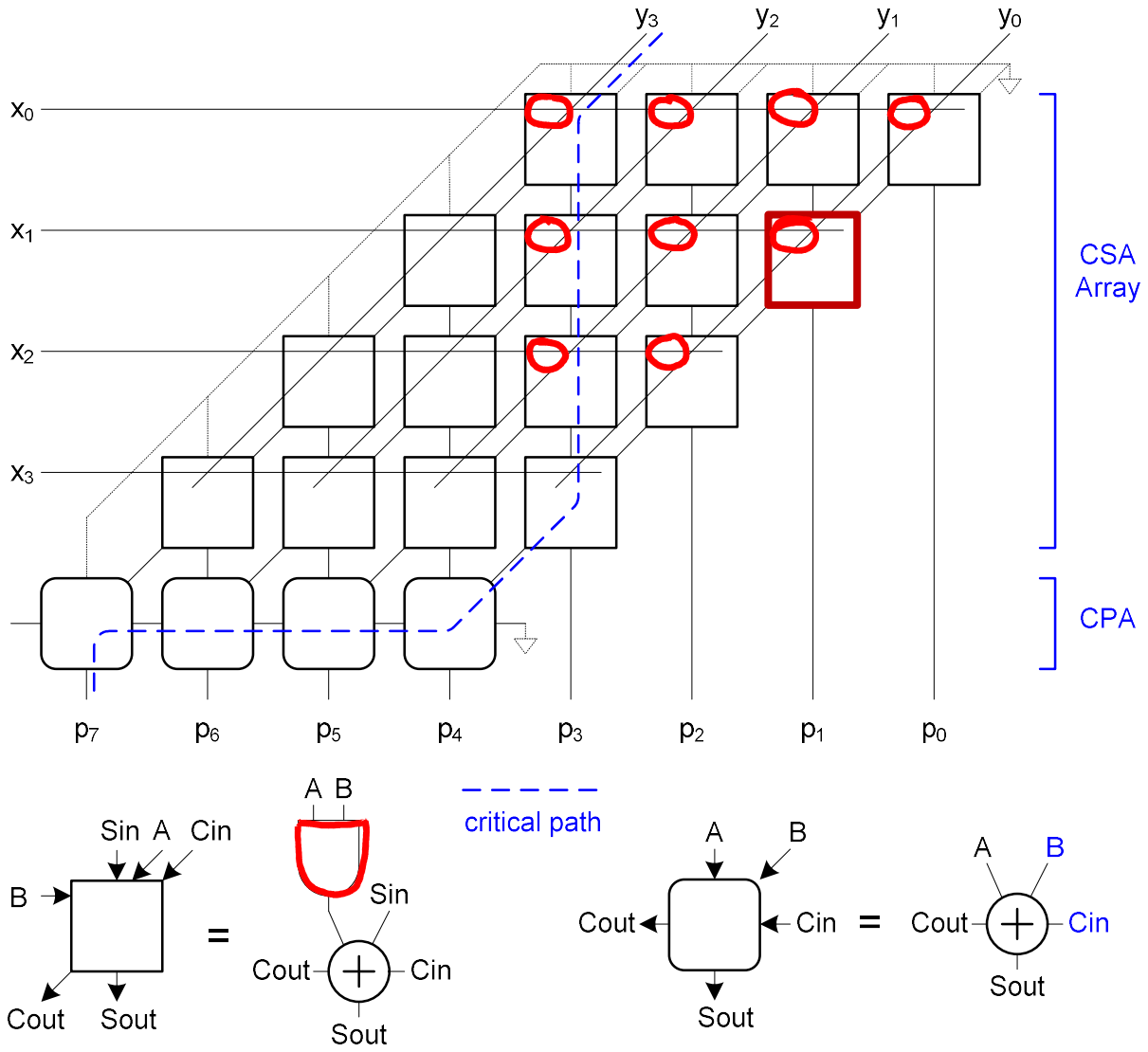


Unsigned Array Multiplication

- Fast multipliers use carry-save adders (CSA) to sum the partial products.
  - A CSA typically has 1.5~2 FO4 inverter delay independent of the width of the partial product
  - A carry-propagate adder (CPA) tends to have 4~15 FO4 inverter delay depending on the width, architecture, and circuit family

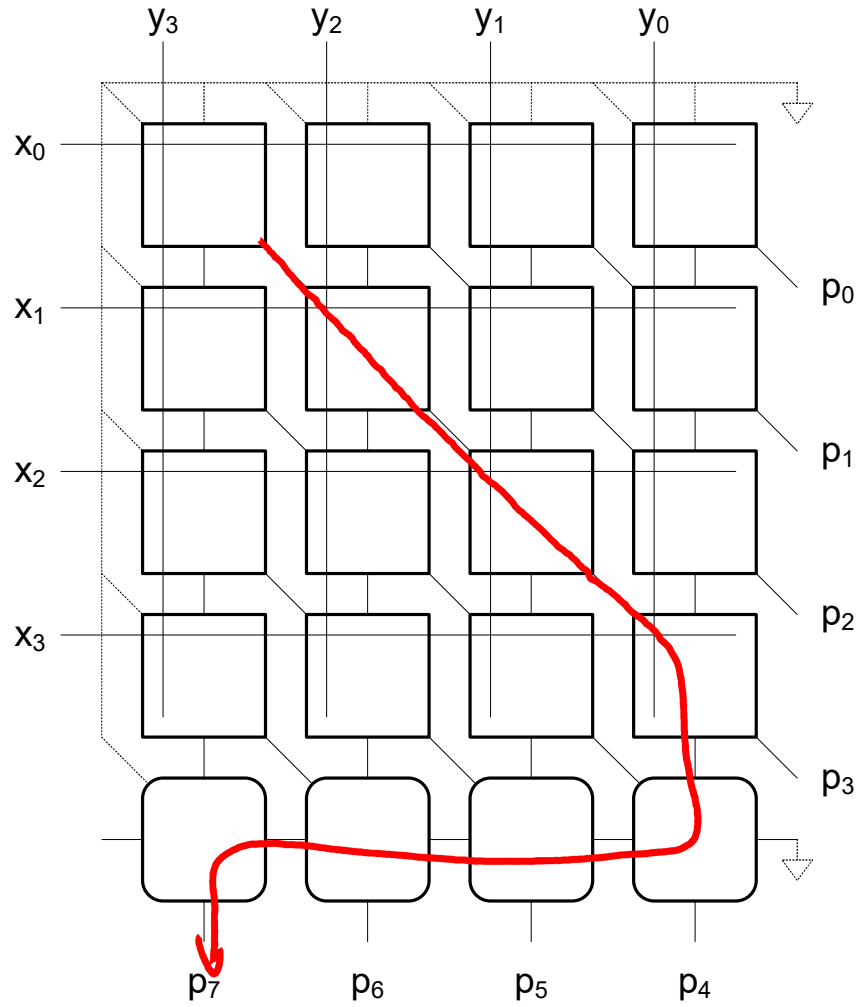
Array Multiplier





Rectangle Array

- To fit rectangular floorplan

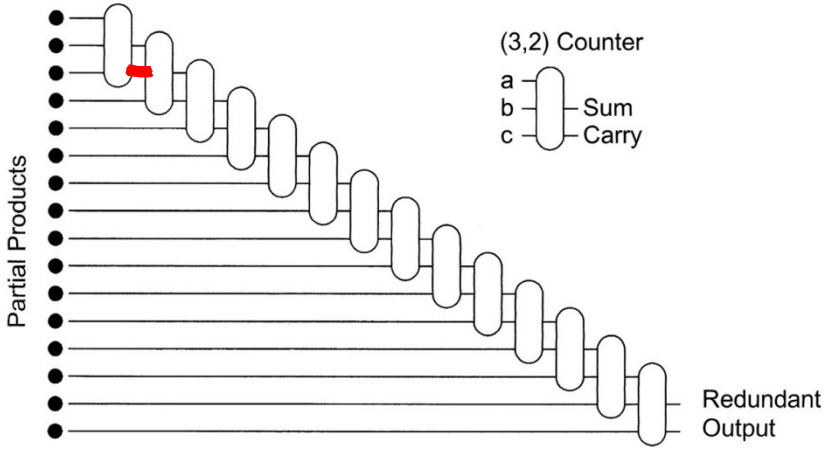


Summation of Partial Product with CSA

- CSA is effective a “1’s counter”
- Known as (3,2) counter

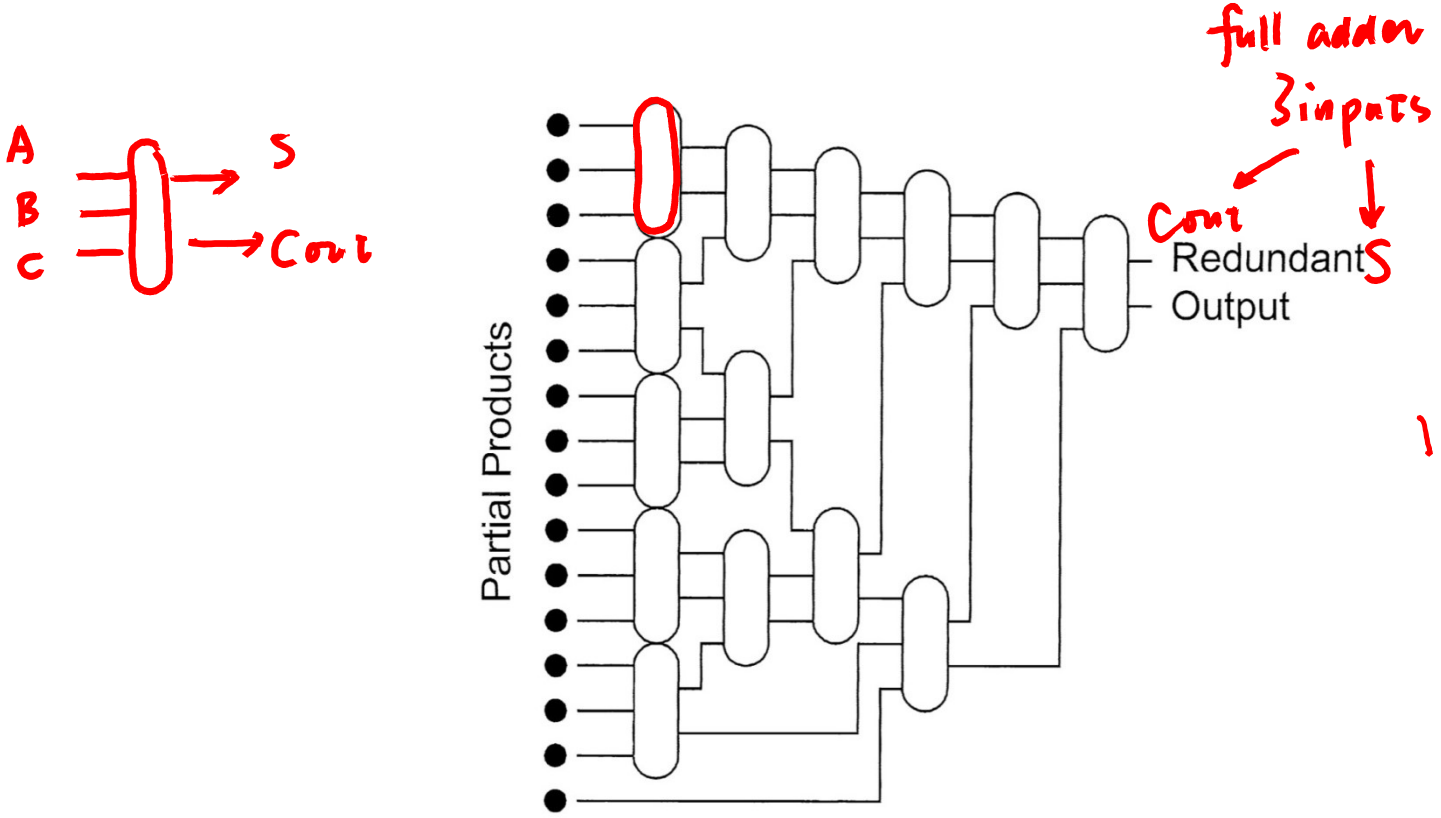
Table 10.14 An adder as a 1’s counter

A	B	C	Carry	Sum	Number of 1’s
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	2
1	0	0	0	1	1
1	0	1	1	0	2
1	1	0	1	0	2
1	1	1	1	1	3



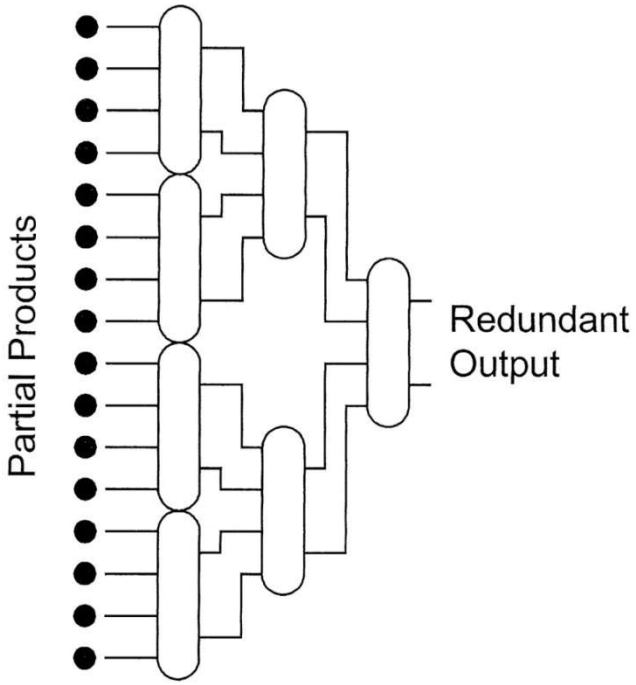
Wallace Tree Multiplication

- Wallace tree architecture speed up column addition, requiring $[\log_{3/2}(N/2)]$ levels of (3,2) counters.

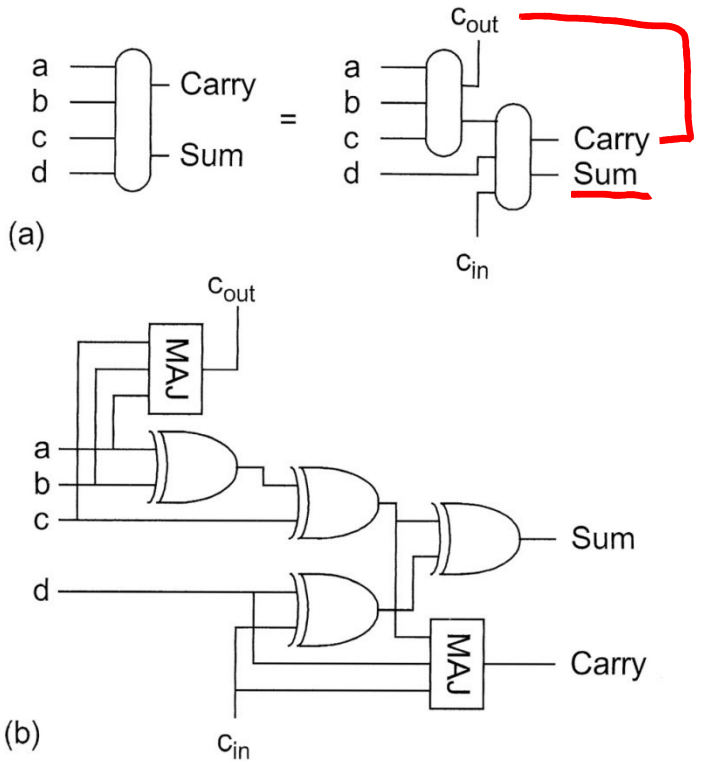


Wallace Tree Multiplication

- (4:2) compressors used for more regular layout, requiring $[\log_2(N/2)]$ levels of (5,3) counters.



(4:2) compressor



Summary

- For synthesis/APR design flow:
  choose proper adder/multiplier architectures with various area/speed tradeoff based on applications
- Custom design at schematic level
  provides the most performance optimization
- Wiring capacitance needs to be taken care of in multiplier design, compact cell and short wires results in fast, small, and low-power designs

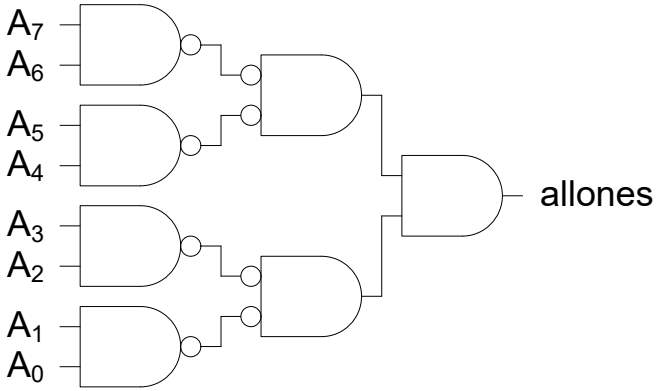
Outline

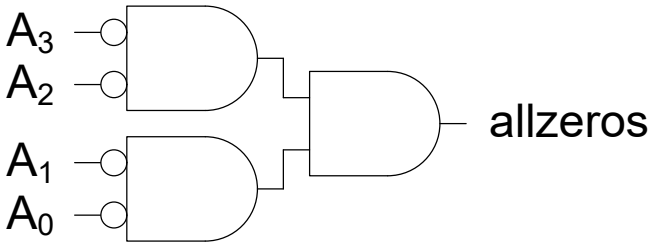
- Addition/Subtraction
- Multiplication

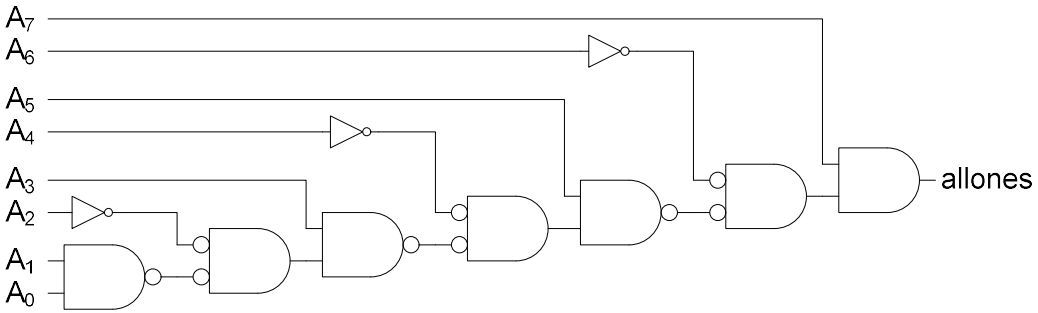
- **One/Zero Detectors**
- Comparators
- Counters
- Shifters

1's and 0's Detectors

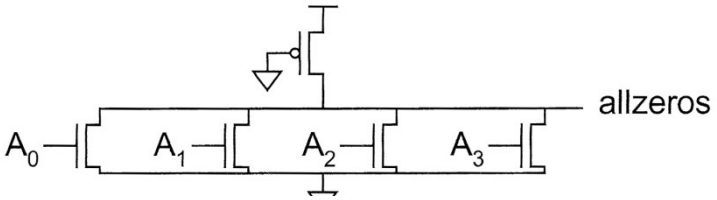
- 1's detector: N-input AND gate
- 0's detector: NOTs + 1's detector (N-input NOR)







Mimic the adder delay



"wired-OR"

Outline

- Addition/Subtraction
- Multiplication

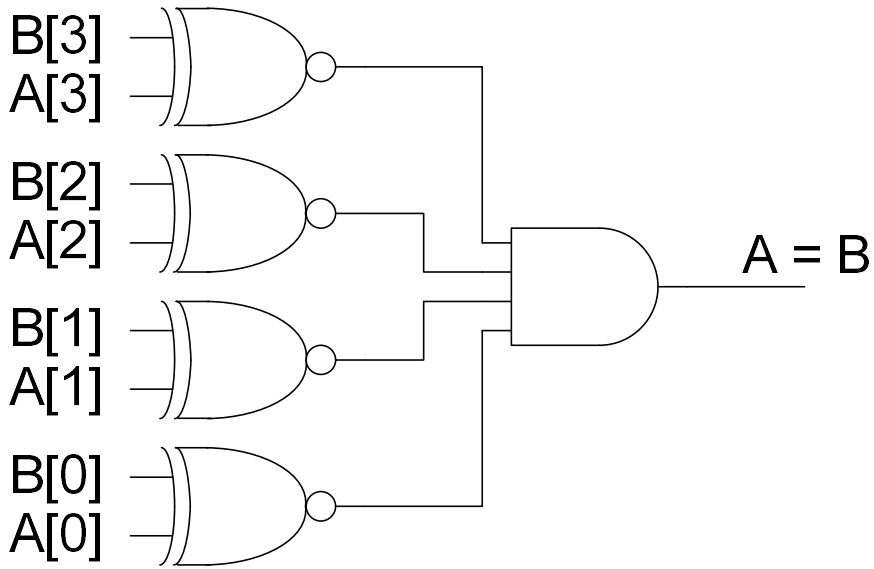
- One/Zero Detectors
- **Comparators**
- Counters
- Shifters

Comparators

- Equality comparator: $A = B$
- Magnitude comparator: $A < B$

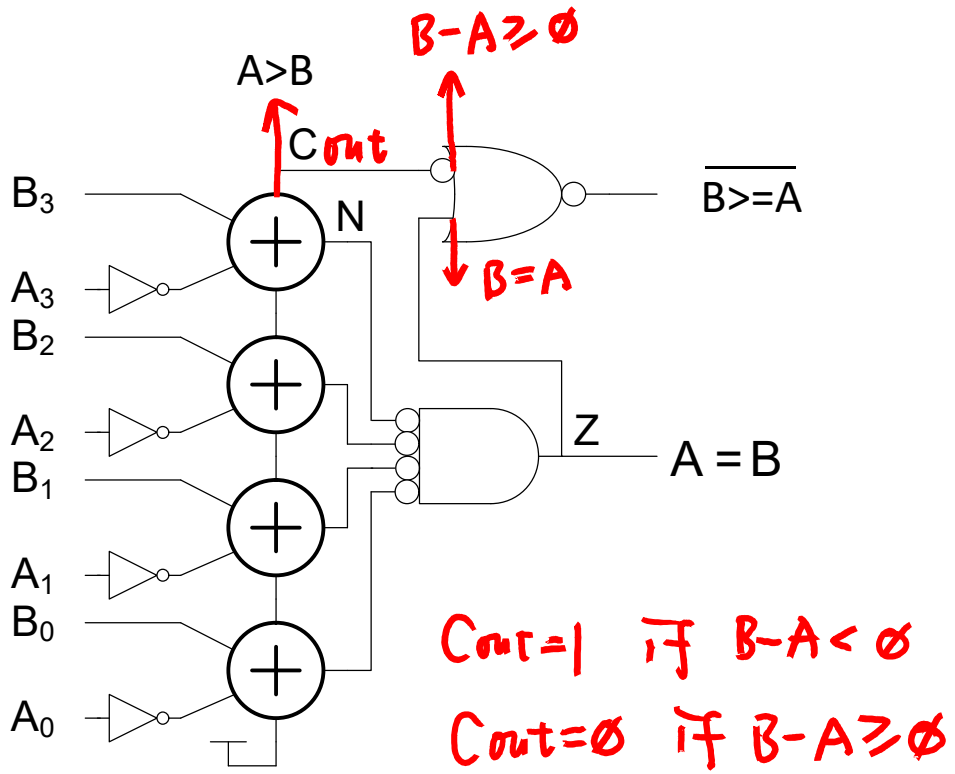
Equality Comparator

- Check if each bit is equal (XNOR, aka equality gate)
- 1's detection on bitwise equality



Magnitude Comparator

- Compute B–A and check the final sign bit
- For unsigned numbers, carry out is sign bit



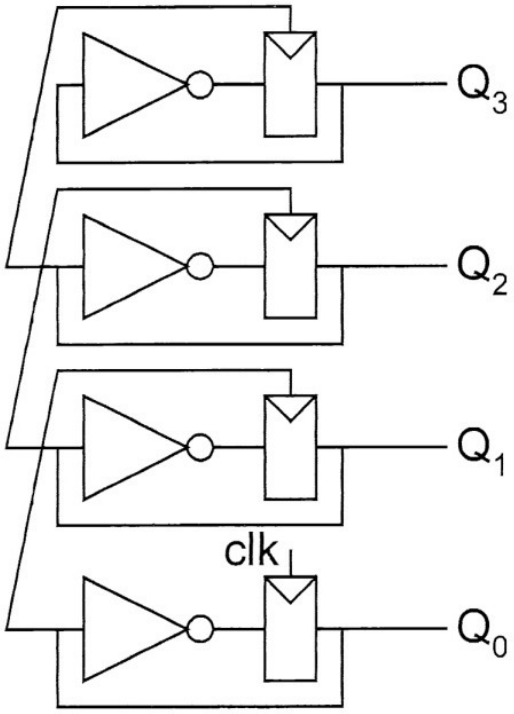
Outline

- Addition/Subtraction
- Multiplication

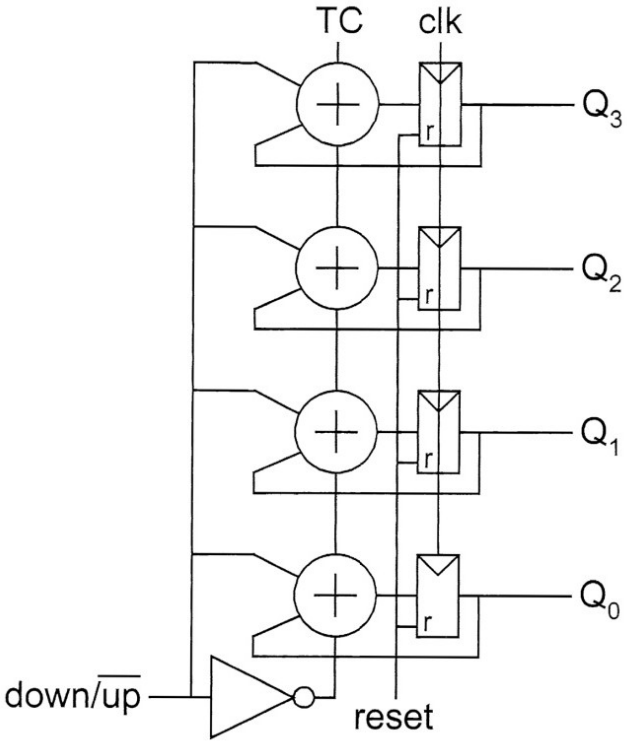
- One/Zero Detectors
- Comparators
- **Counters**
- Shifters

Binary Counters

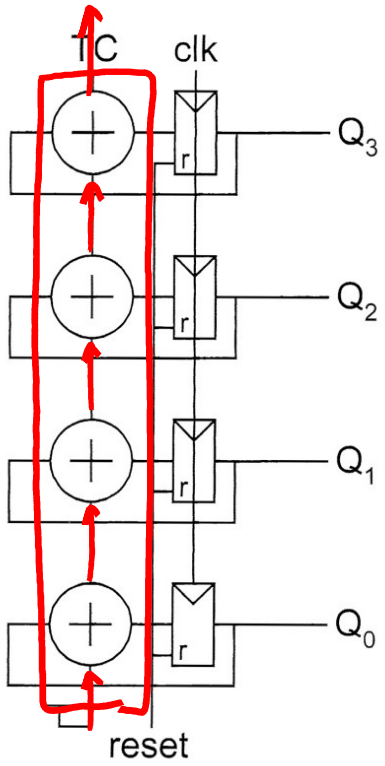
Asynchronous Counter



Synchronous Un/Down Counter



Synchronous Incrementer



(half adder only)