
EE3230 Lecture 9: Datapath Subsystems

Ping-Hsuan Hsieh (謝秉璇)

Delta Building R908

EXT 42590

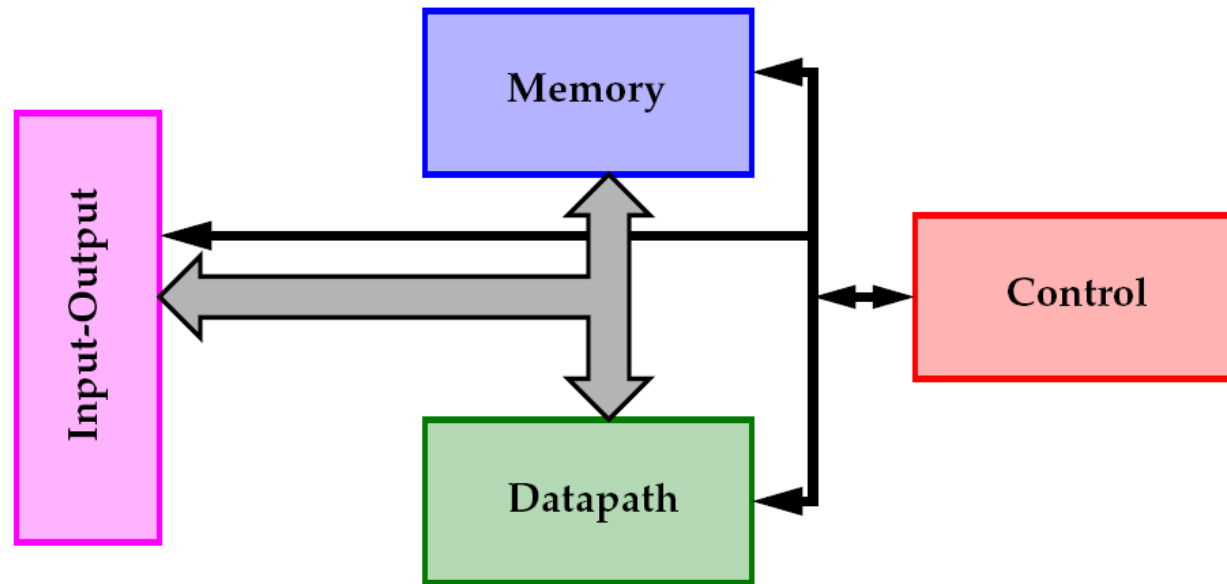
phsieh@ee.nthu.edu.tw

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. random access
 - Single-ported vs. multi-ported access
 - Dynamic vs. static
- the way data is accessed
- data retention characteristics

- **Control:**

- FSM (sequential circuit) implemented using random logics, PLAs, and memories

- **Interconnect and I/O:**

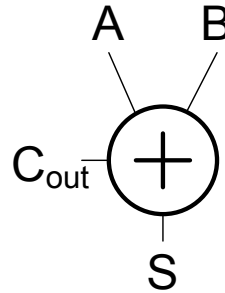
- Parasitic R, C, L affect performance for wires both on and off the chip
- Growing die size increases the length of the on-chip interconnect, increasing the impact of parasitics

Single-Bit Addition

- Half adder

$$S = A \oplus B$$

$$C_{\text{out}} = A \cdot B$$



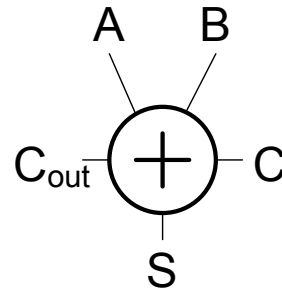
A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Full adder

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

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

$$= AB + BC + AC$$



A	B	C	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

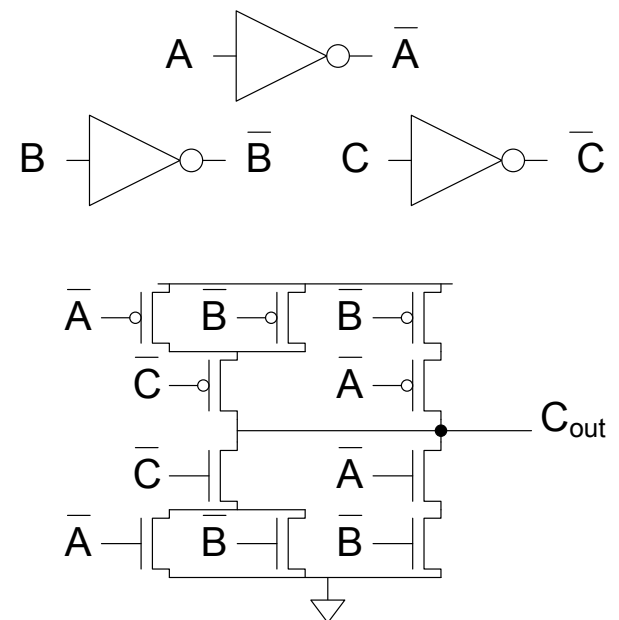
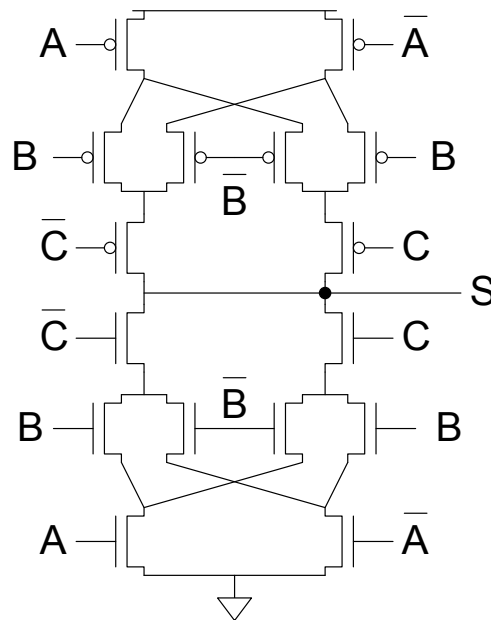
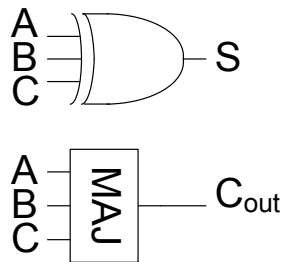
- Direct implementation from equations

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

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

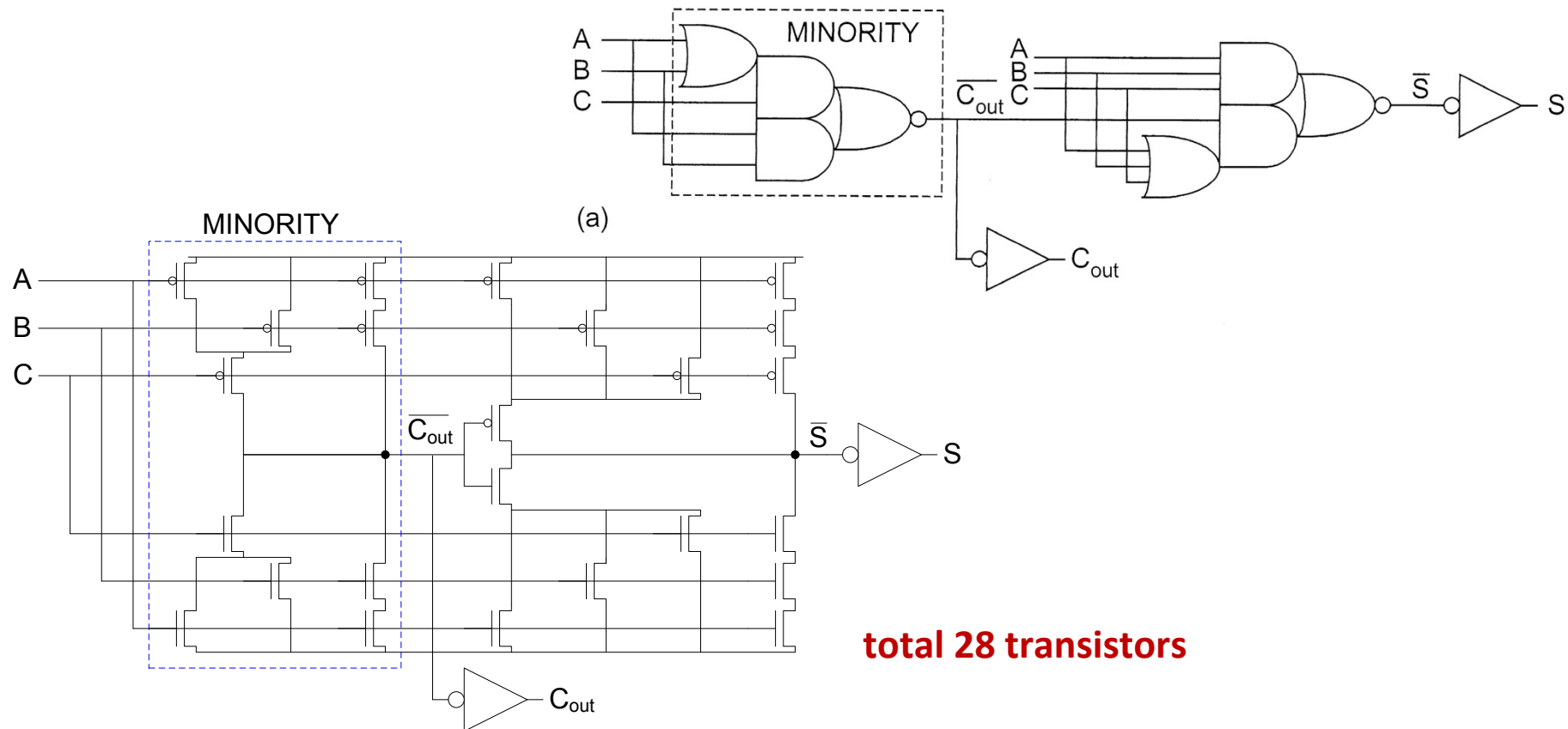
$$\text{INV} \quad (6 \text{ transistors})$$

total 32 transistors



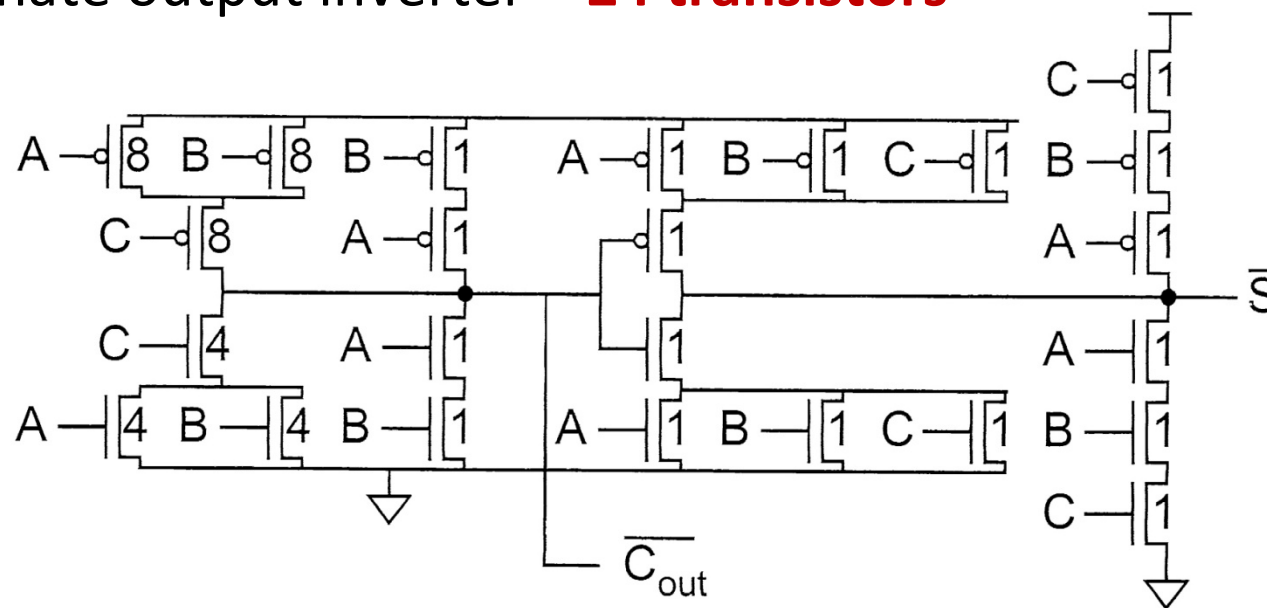
Full-Adder Design II

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



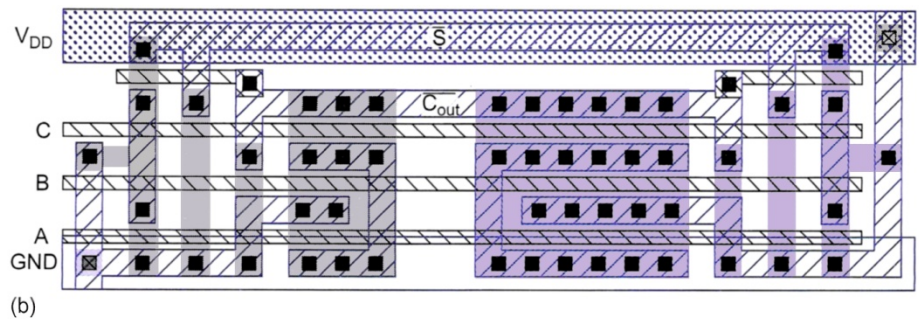
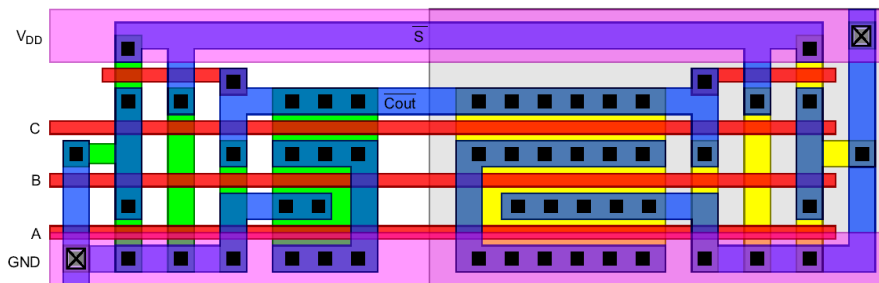
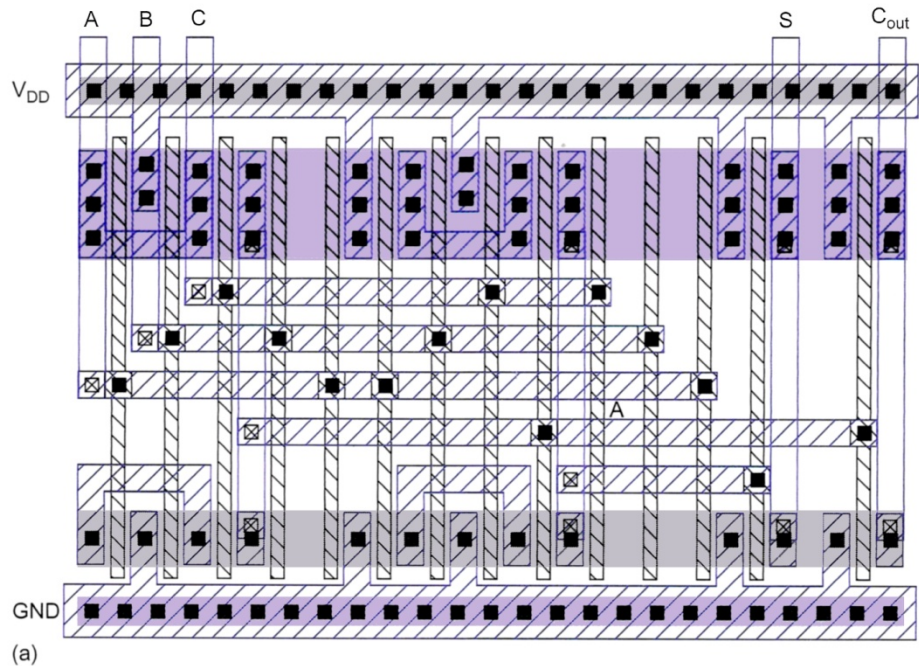
Critical Path Optimization

- **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**



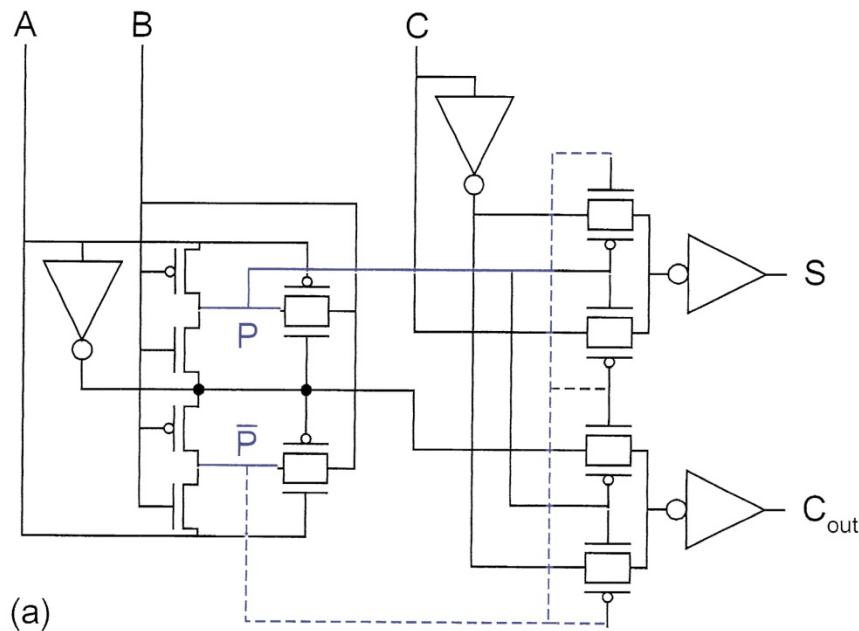
Layout

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

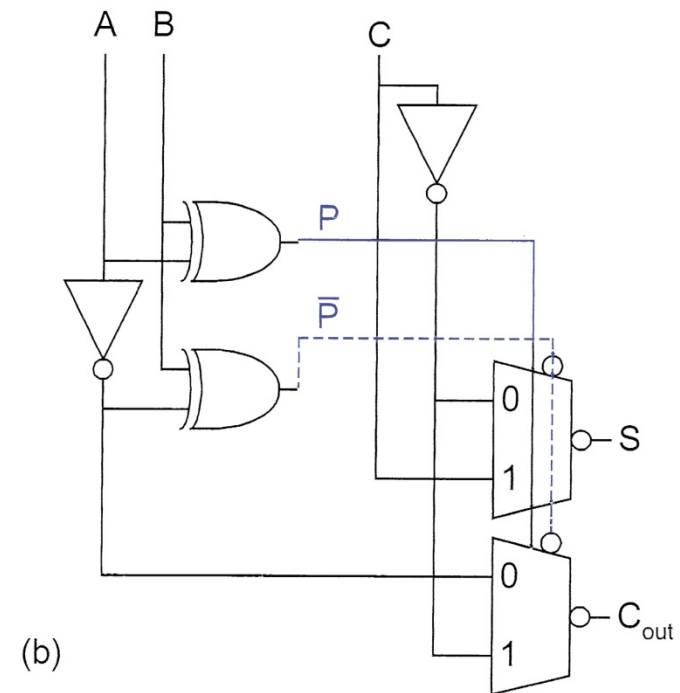


Full-Adder III

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

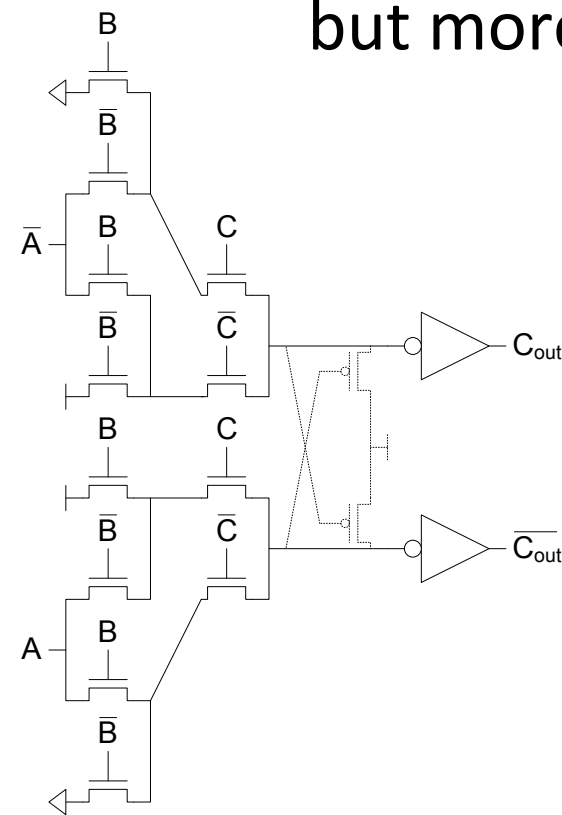
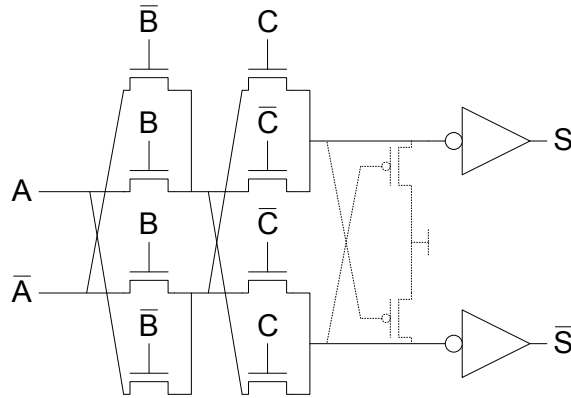


24 transistors



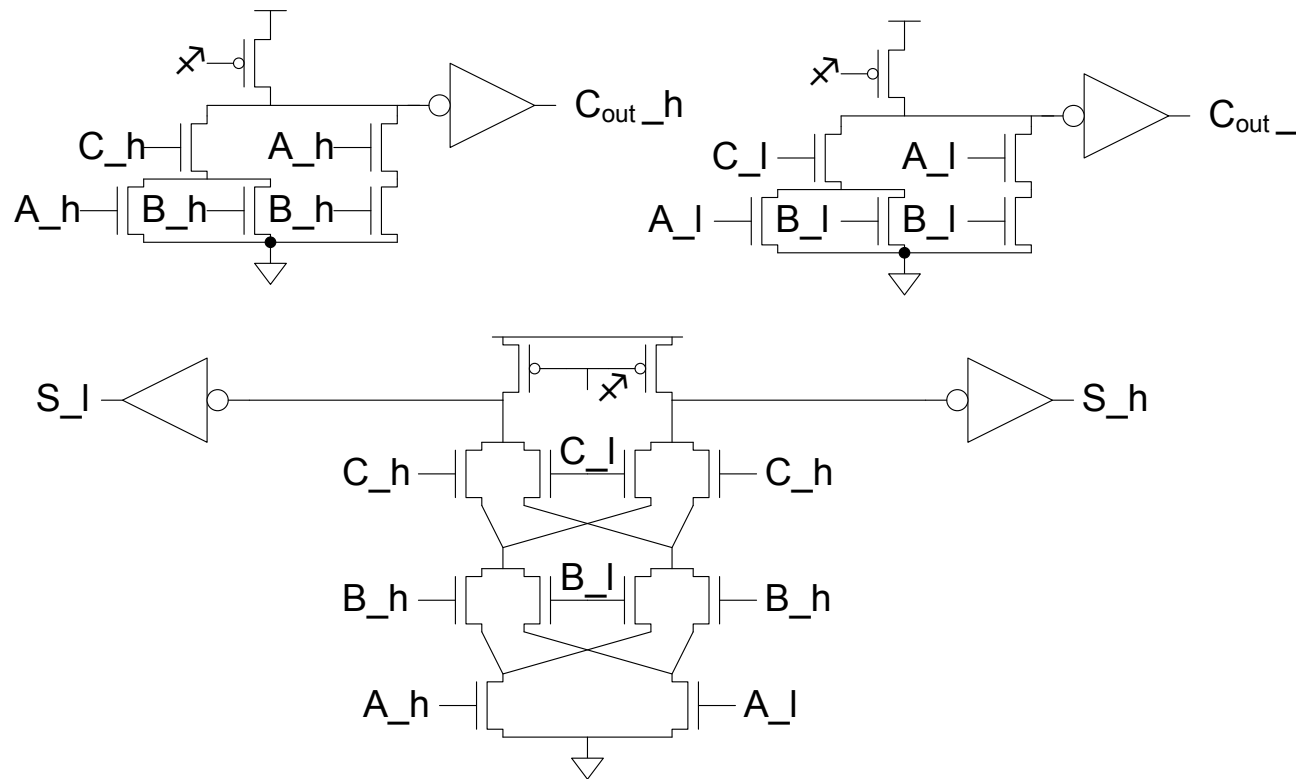
Full-Adder IV

- 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



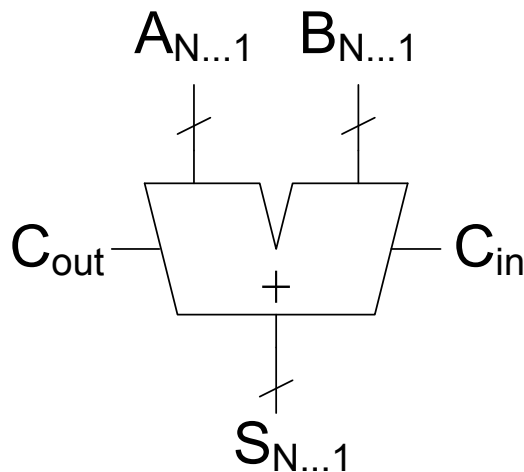
Full-Adder V

- Footless dual-rail domino
 - Very fast, but large area and power hungry
 - Similar delay for S and C_{out} $\rightarrow$ 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?



$$\begin{array}{r} \textcircled{0}000\textcircled{0} \\ 1111 \\ +0000 \\ \hline 1111 \end{array}$$

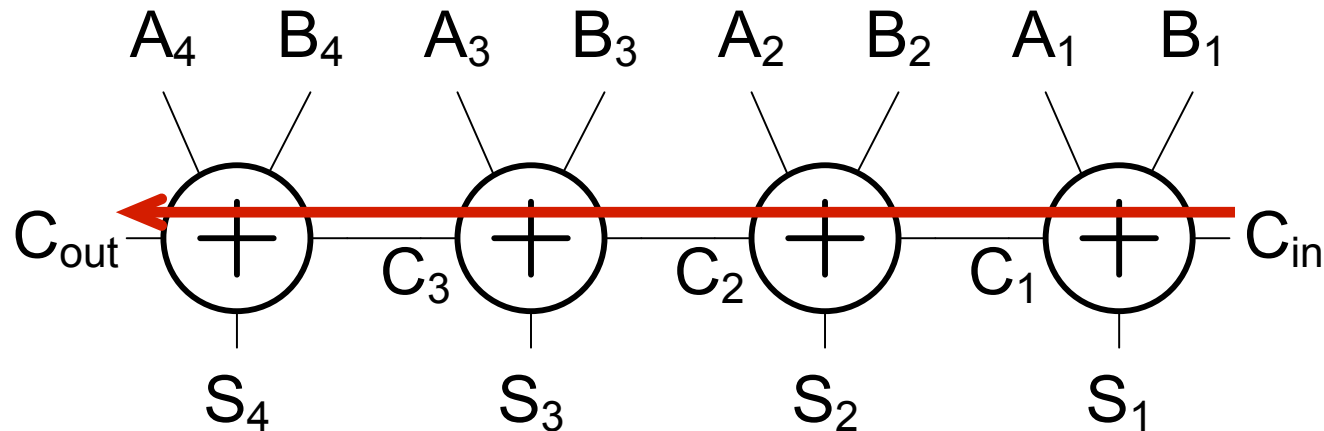
Diagram illustrating a 5-bit addition where the carry-in (C_{in}) is 0 and the carry-out (C_{out}) is 0. The inputs are 00000 and 1111, and the sum is 1111.

$$\begin{array}{r} \textcircled{1}111\textcircled{1} \\ 1111 \\ +0000 \\ \hline 0000 \end{array}$$

Diagram illustrating a 5-bit addition where the carry-in (C_{in}) is 1 and the carry-out (C_{out}) is 1. The inputs are 11111 and 1111, and the sum is 0000. The carry bits are labeled "carries".

Carry Ripple Adder

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



- 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 C_{in} **$(G = A \bullet B)$**
 - Propagate: $C_{out} = 1$ if and only if $C_{in} = 1$ **$(P = A \oplus B)$**
 - Kill: $C_{out} = 0$ independent of C_{in}

<i>A</i>	<i>B</i>	<i>C</i>	<i>G</i>	<i>P</i>	<i>K</i>	<i>C_{out}</i>	<i>S</i>
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

- G & P for multiple bits spanning i:j

$$\left. \begin{aligned} G_{i:j} &= G_{i:k} + P_{i:k} \cdot G_{k-1:j} \\ P_{i:j} &= P_{i:k} \cdot P_{k-1:j} \end{aligned} \right\} \text{Valency-2 group PG Logic}$$

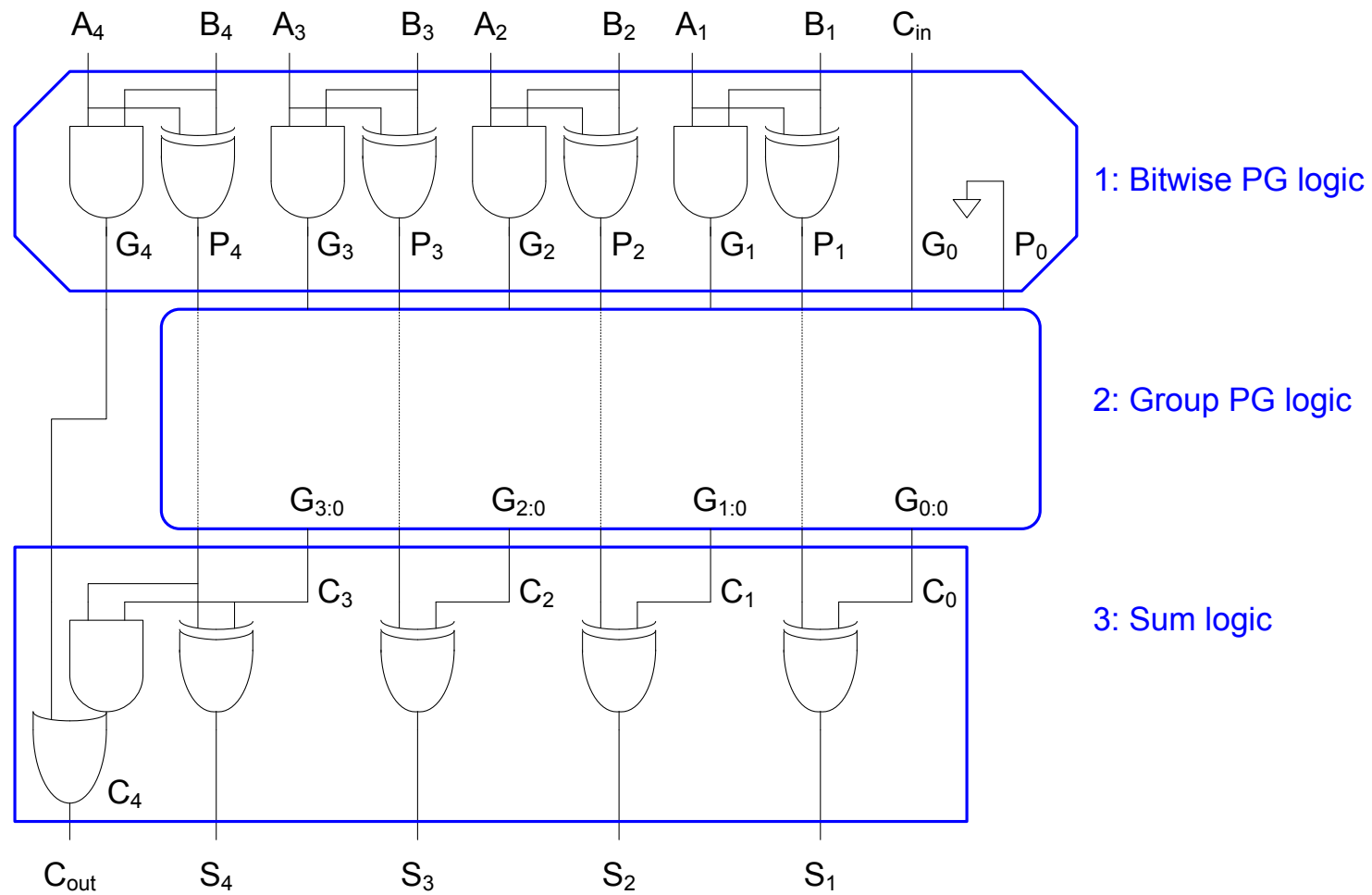
- Base case (single-bit)

$$\begin{aligned} G_{i:i} &\equiv G_i = A_i \cdot B_i & G_{0:0} &\equiv G_0 = C_{in} \\ P_{i:i} &\equiv P_i = A_i \oplus B_i & P_{0:0} &\equiv P_0 = 0 \end{aligned}$$

- 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



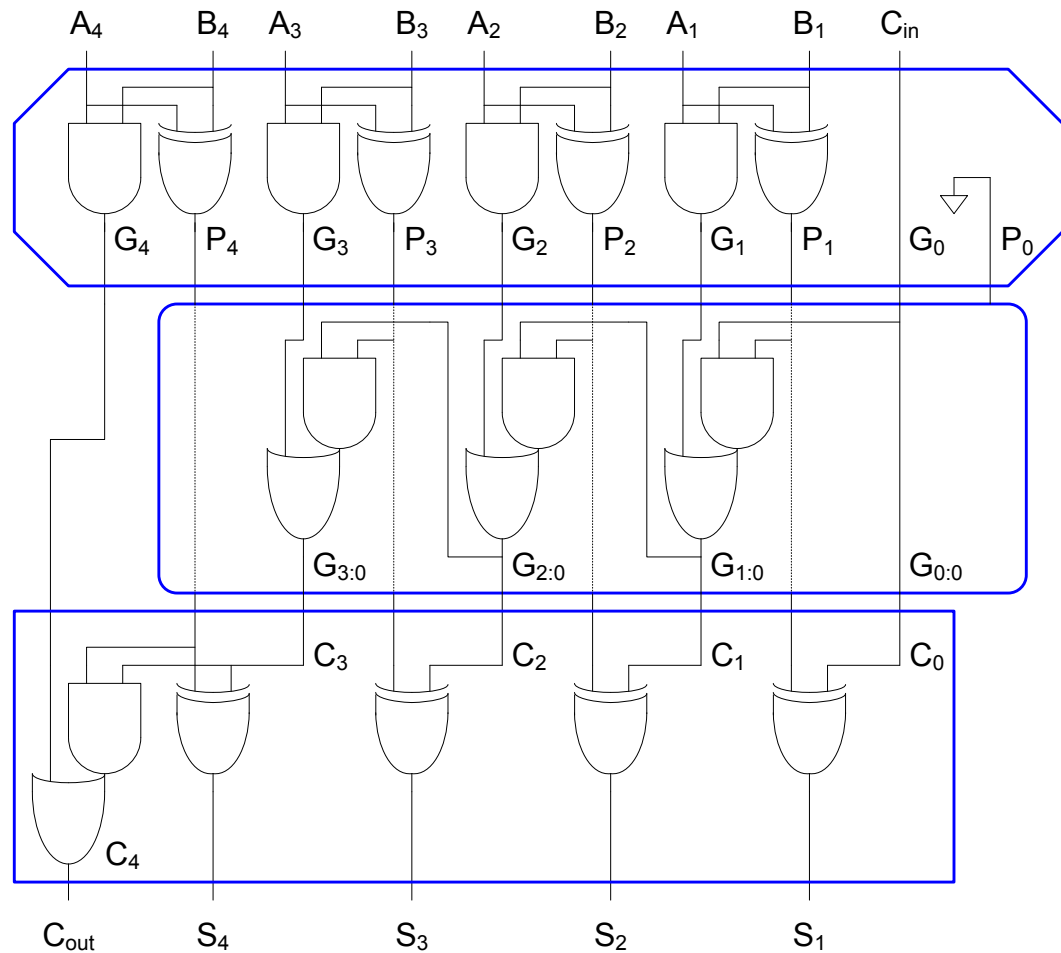
Carry-Ripple Revisited

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

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

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

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



Carry-Ripple PG Diagram

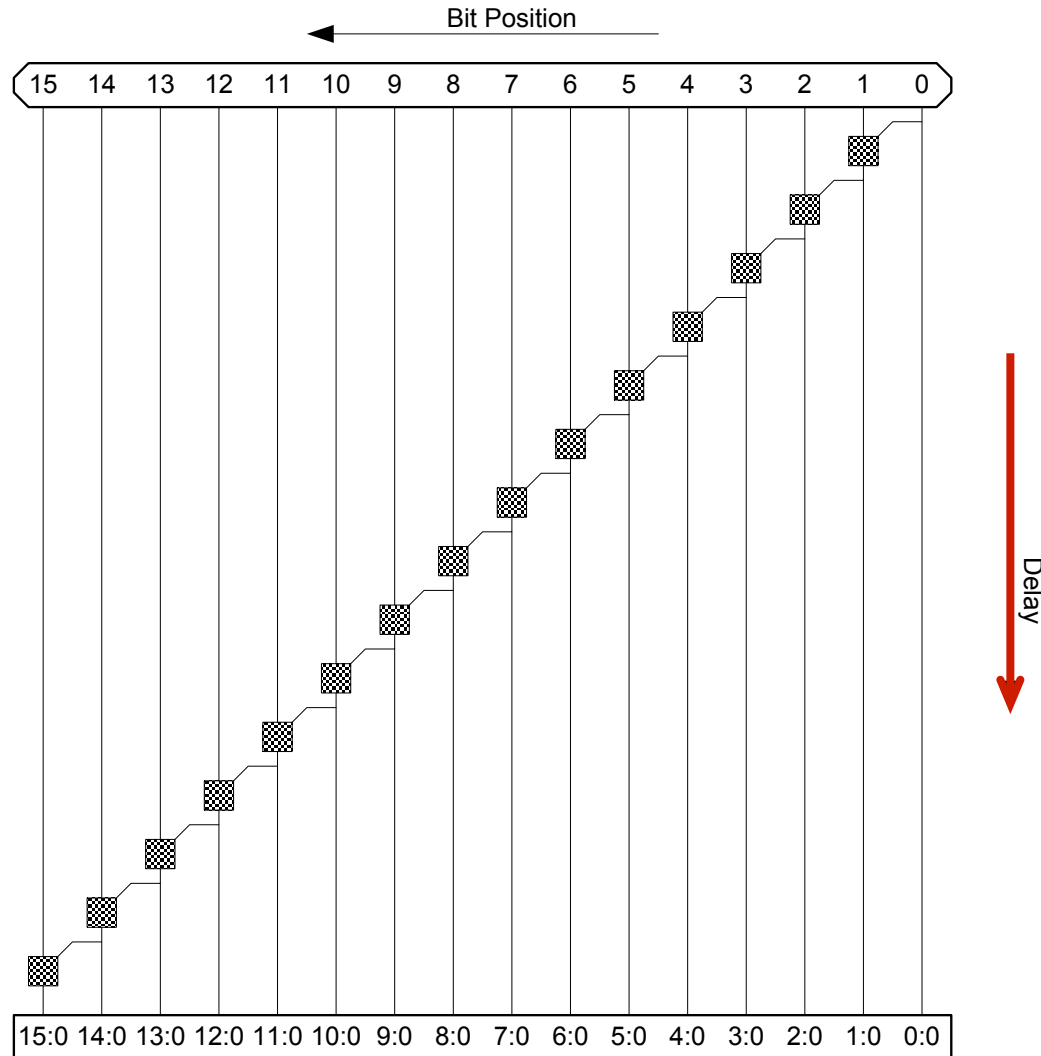
- For N-bit adder

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

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

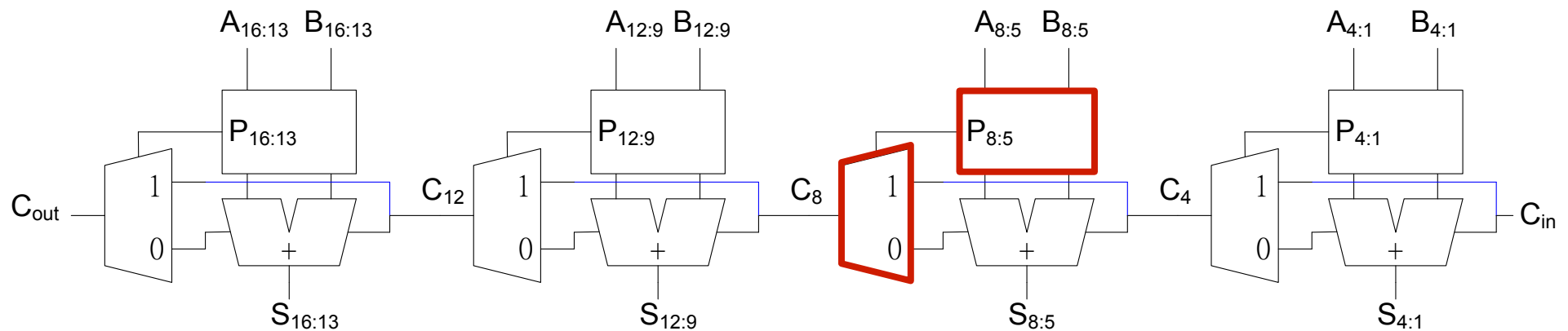


Carry-Skip Adder

- 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

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

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

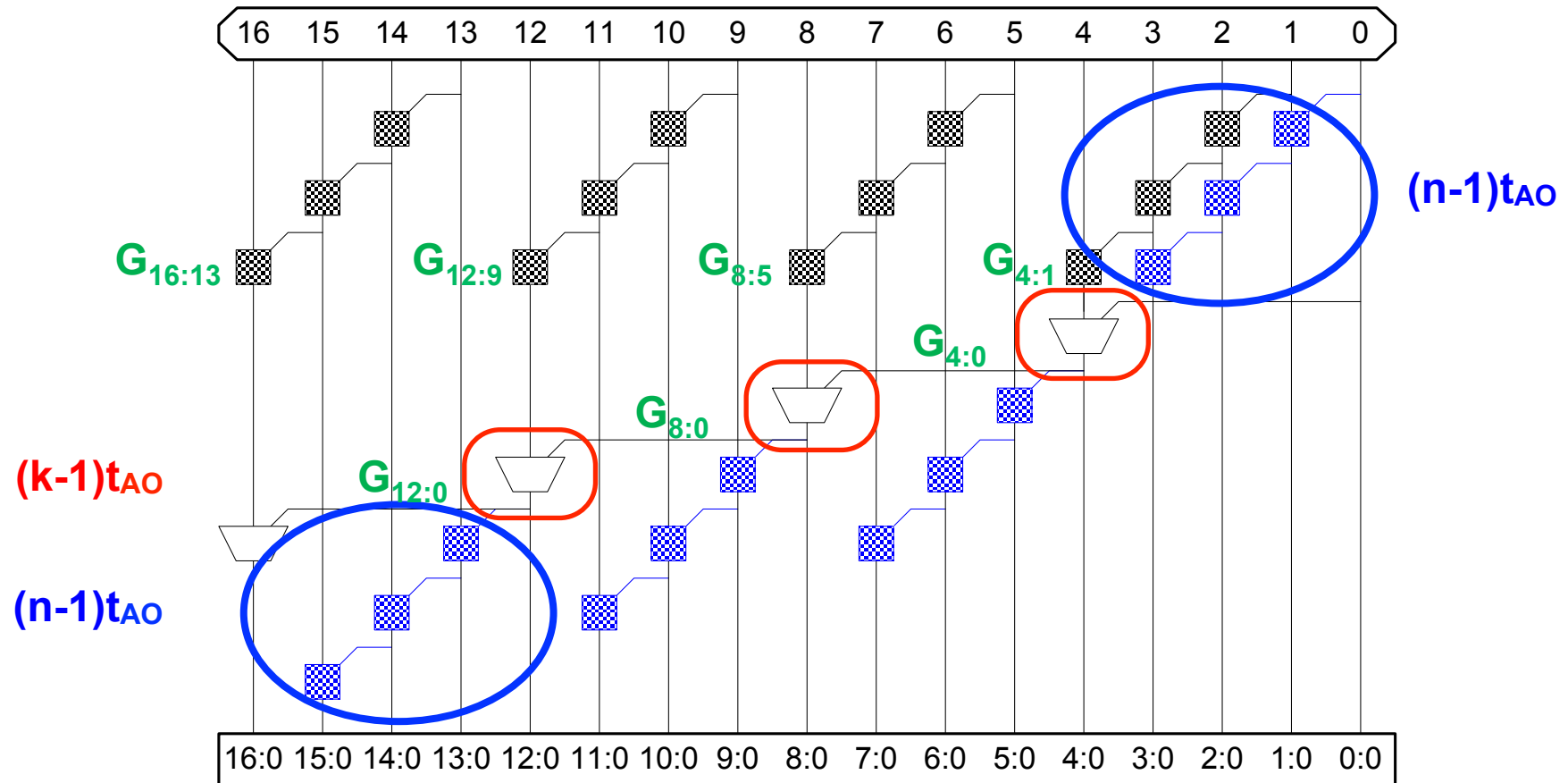


$$G_{12:0} = G_{12:9} + P_{12:9} \cdot G_{8:0}$$

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

Carry-Skip PG Diagram

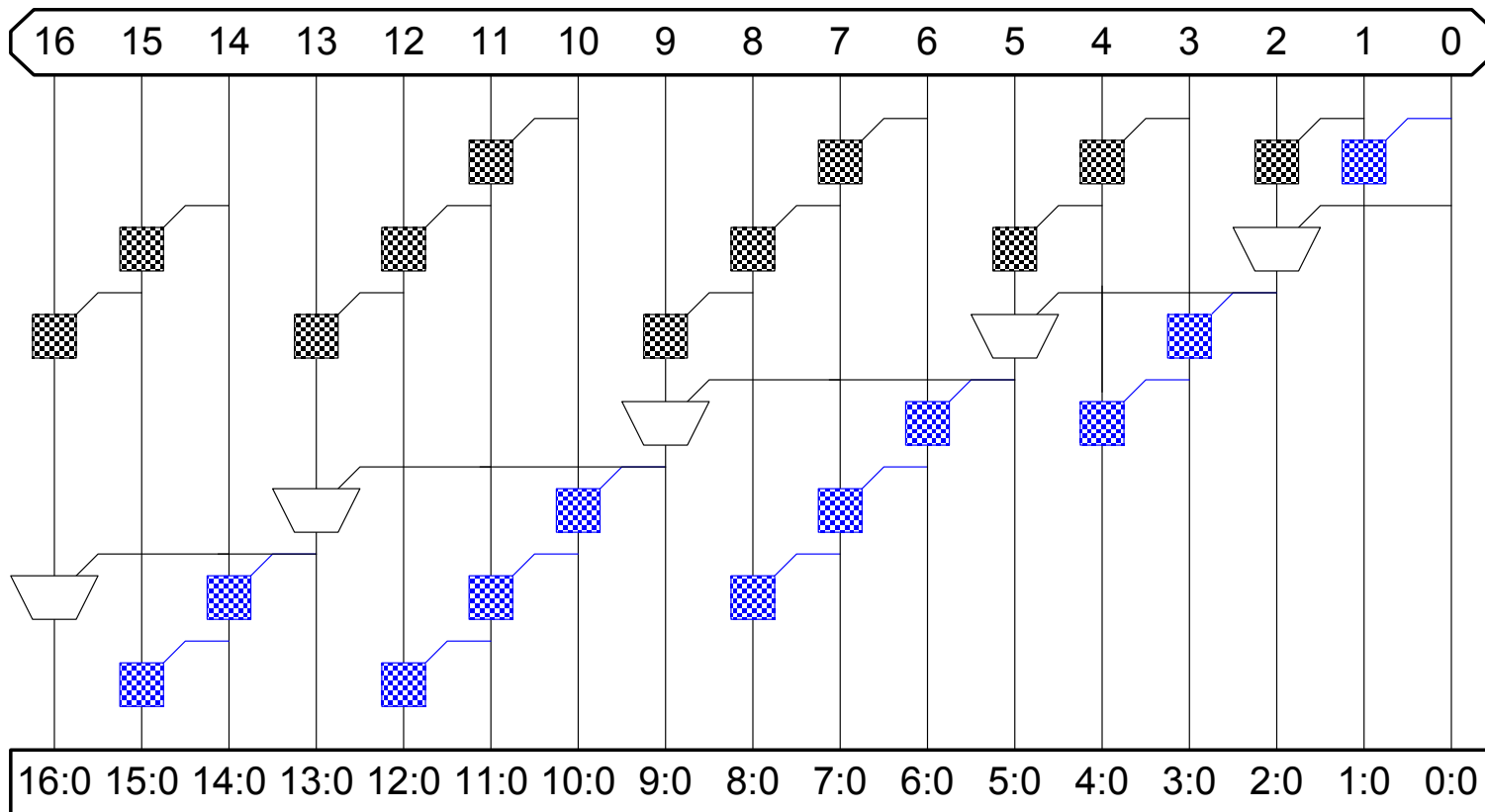
- For k n-bit groups (N = kn)



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

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]

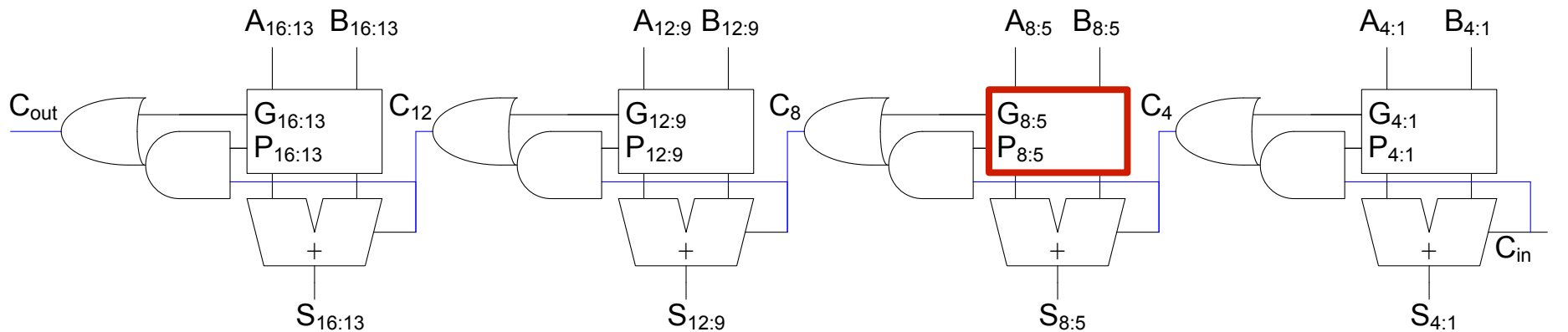


Carry-Lookahead Adder

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

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

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

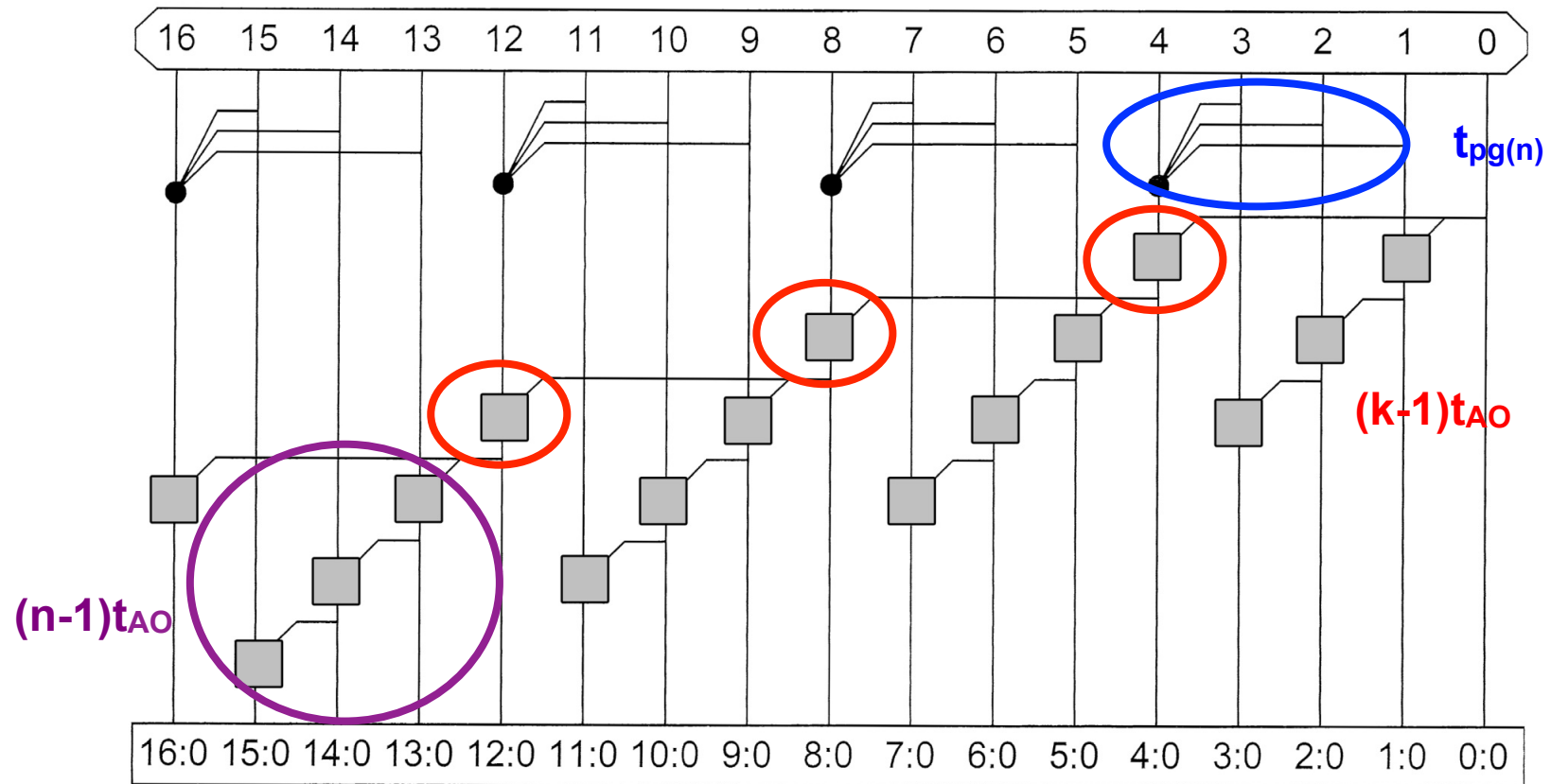


$$G_{12:0} = G_{12:9} + P_{12:9} \cdot G_{8:0}$$

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

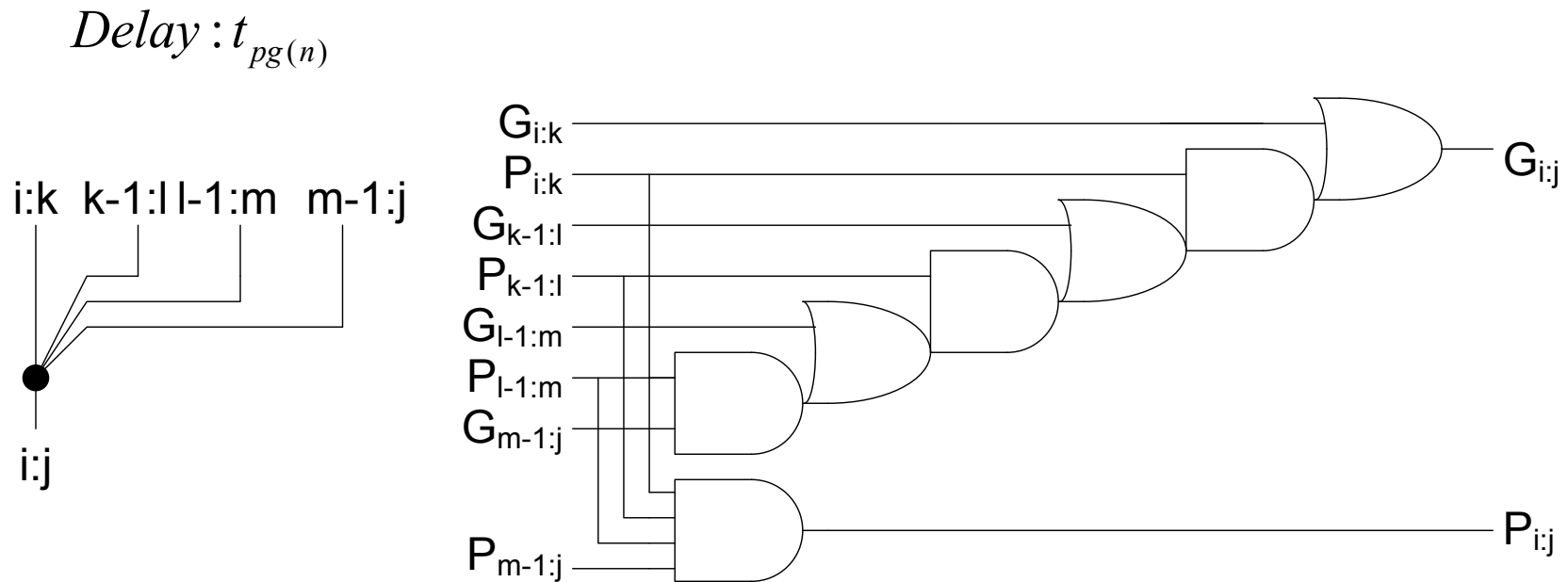
Carry-Lookahead PG Diagram

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



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

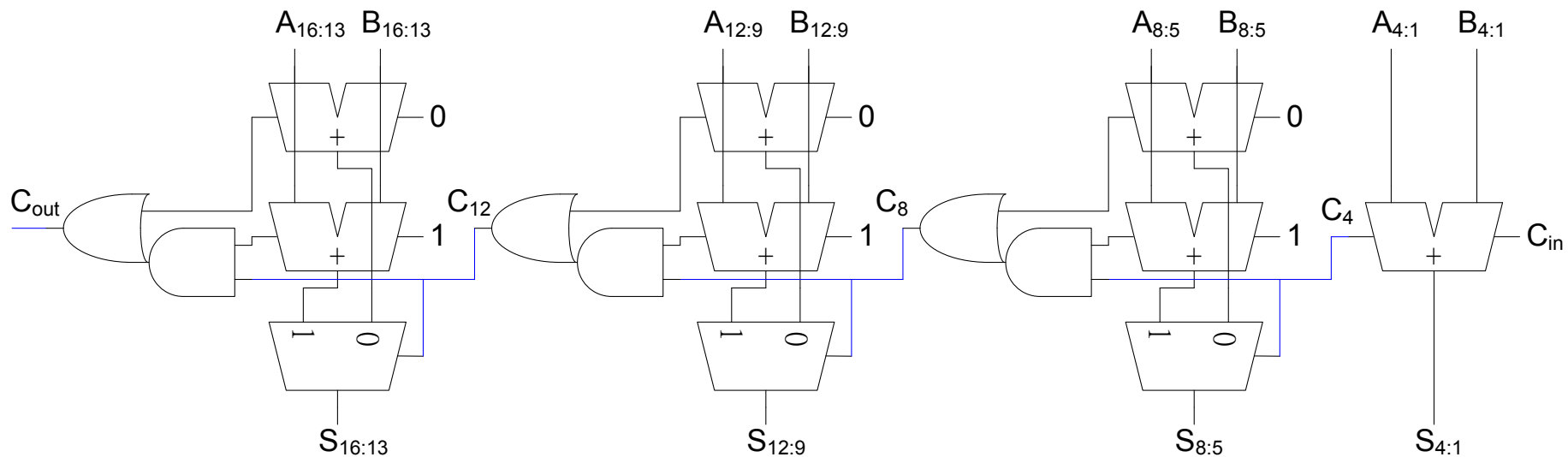
Higher-Valency Cells



Carry-Select Adder

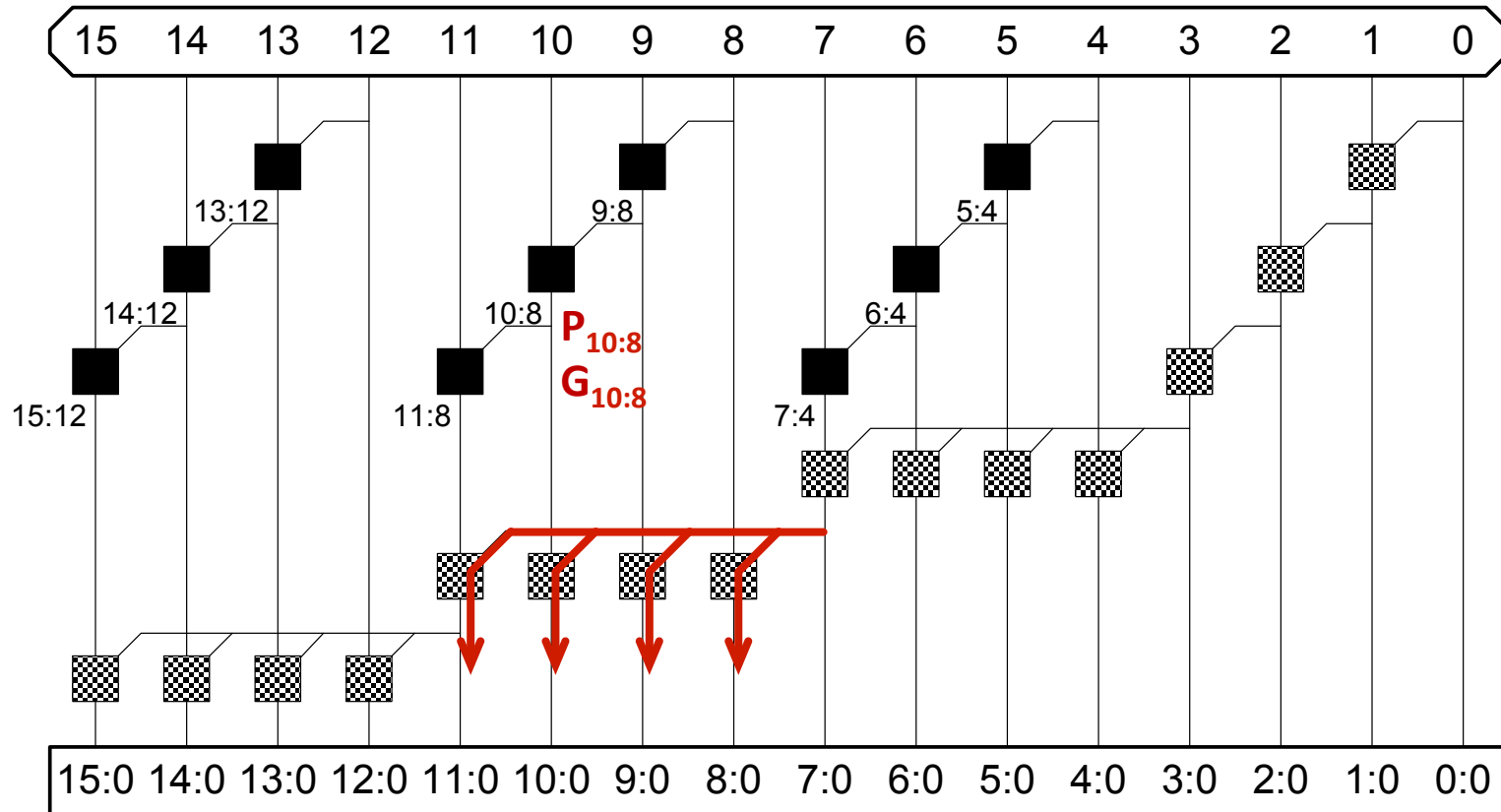
- Trick for critical paths dependent on late input C_{in}
 - Precompute two possible outputs for $C_{in} = 0, 1$
 - Select proper output when C_{in} arrives
- Carry-select adder precomputes n-bit sum

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



Carry-Increment Adder

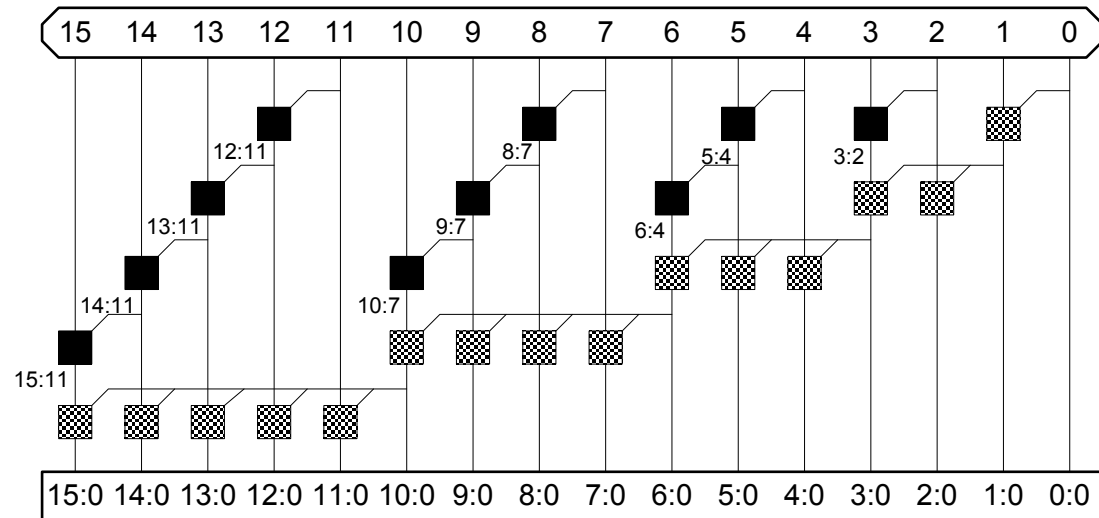
- Factor initial PG and final XOR out of carry-select



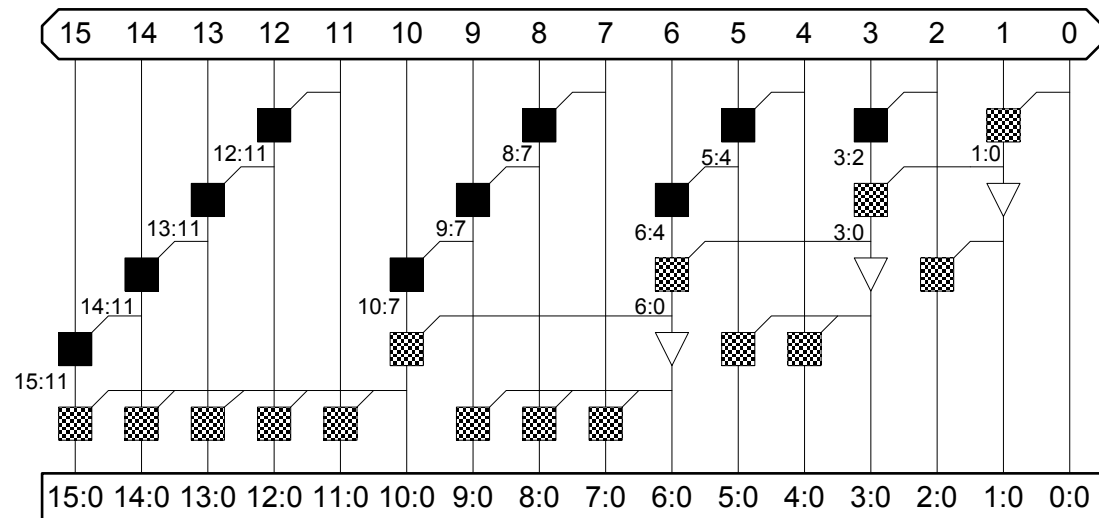
$$t_{\text{increment}} = t_{pg} + \left[(n - 1) + (k - 1) \right] t_{AO} + t_{\text{xor}}$$

Variable Group Size

- Fanout increases with group size



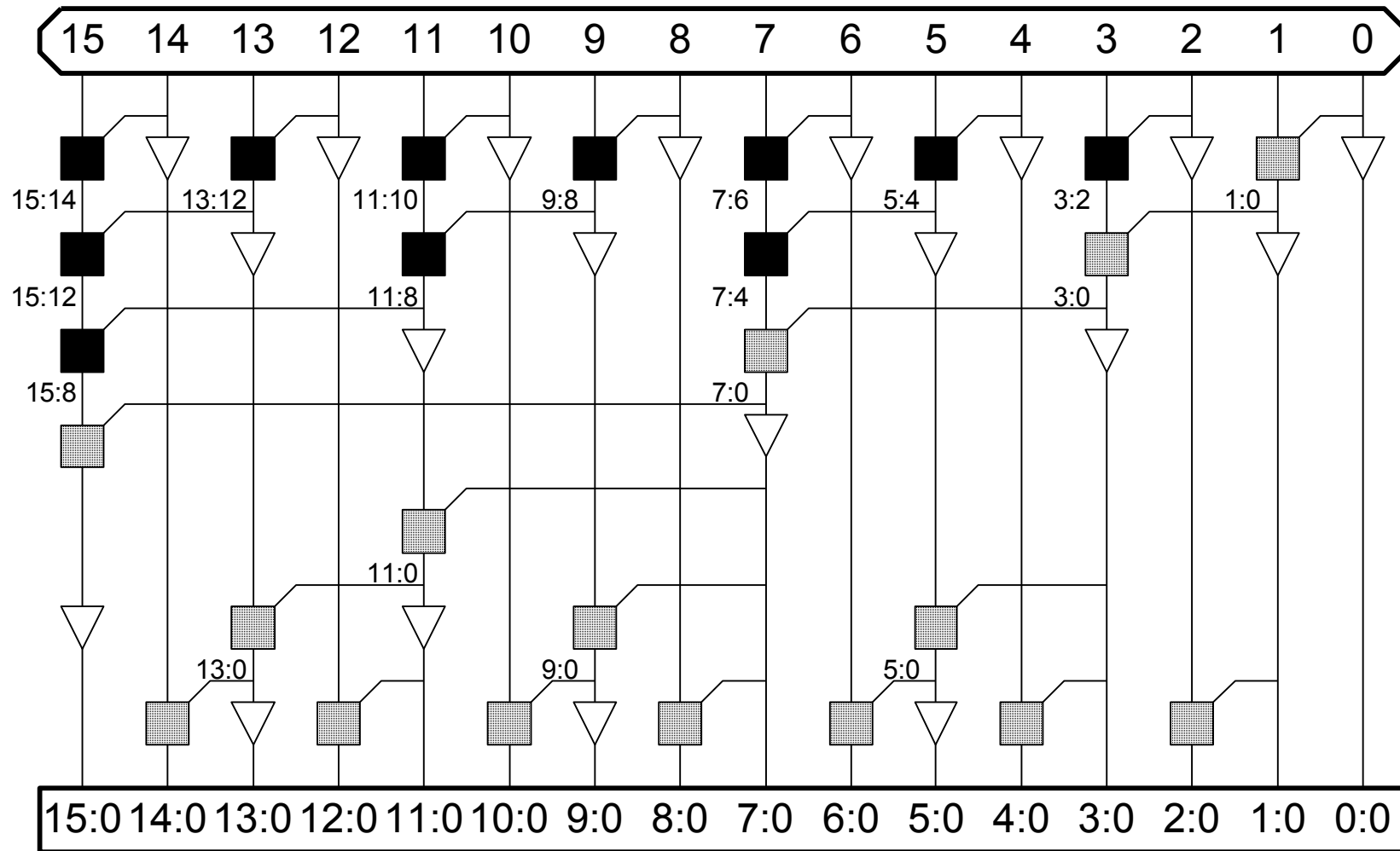
- Buffer noncritical signals
- Reduce branch effort



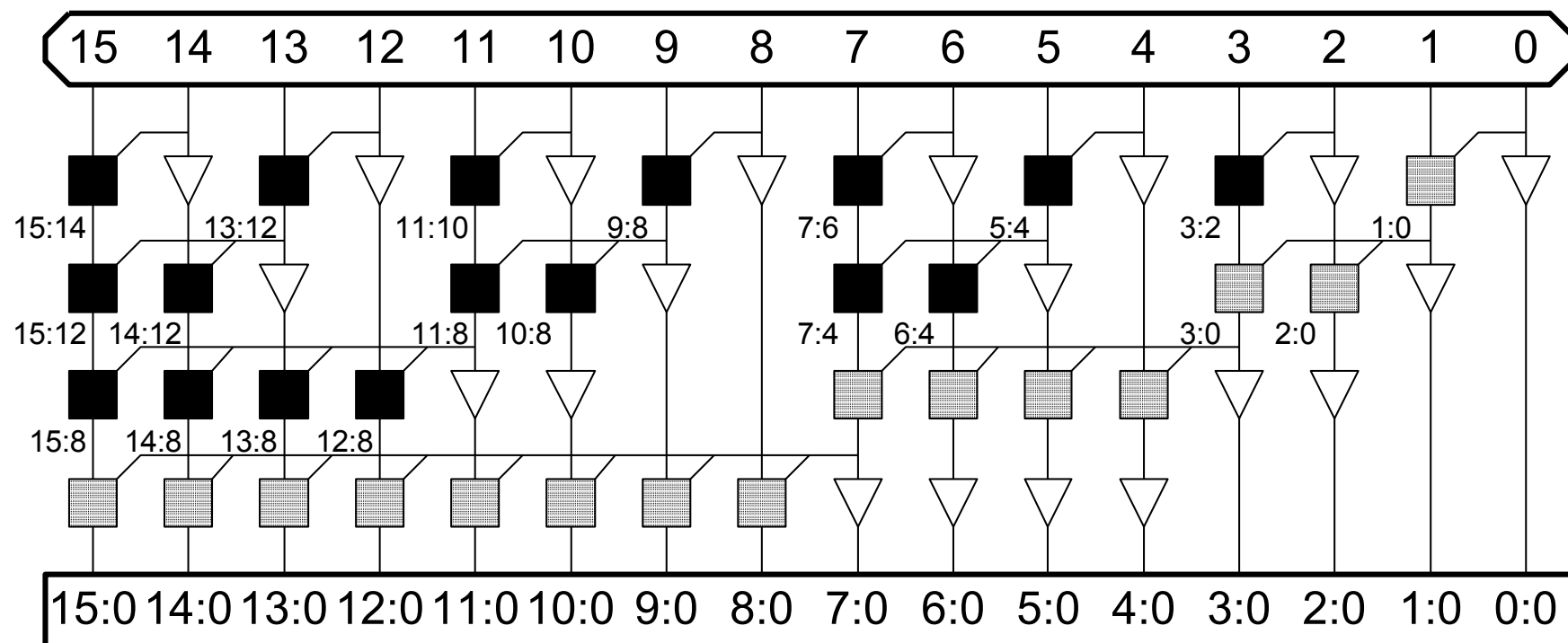
Tree Adder

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

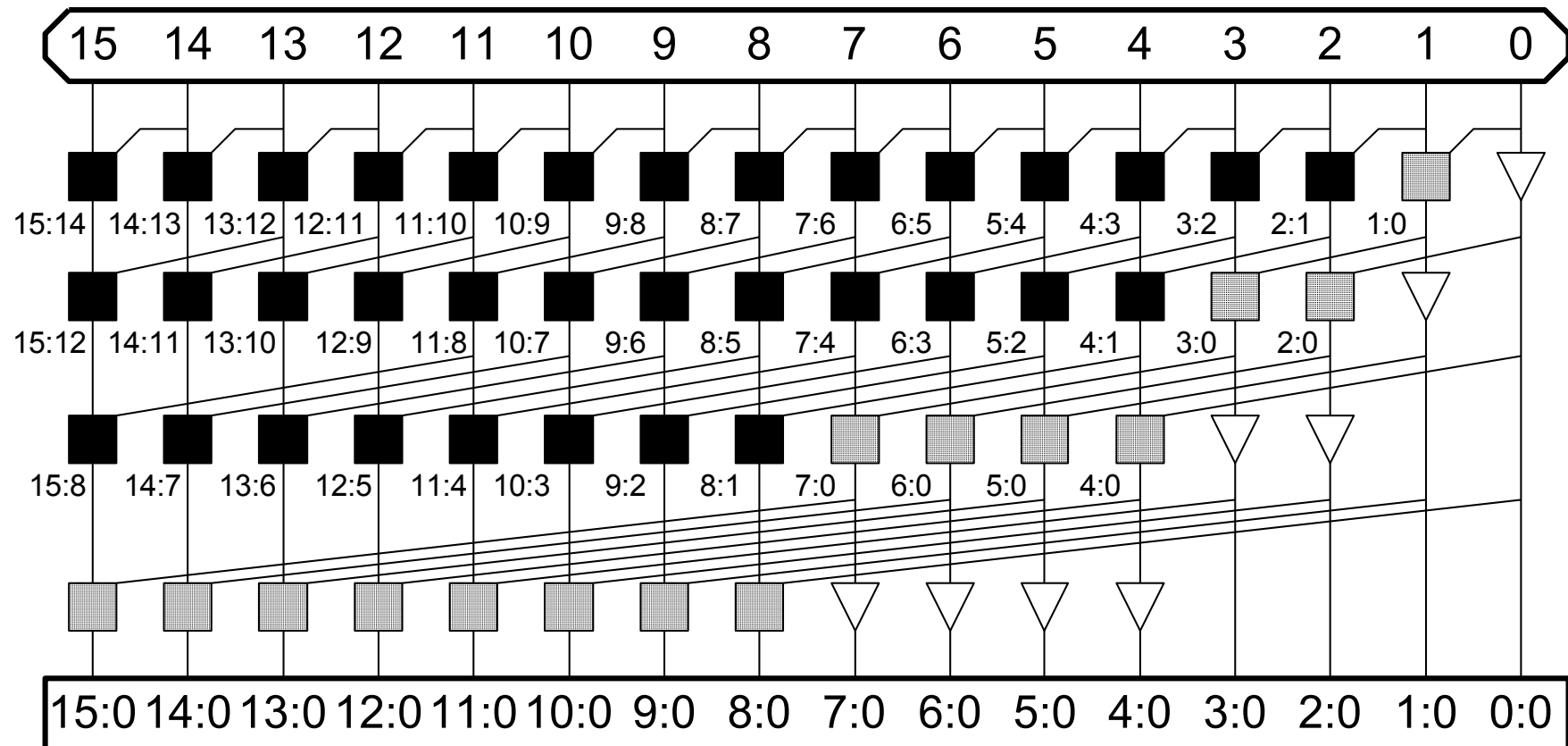
Brent-Kung



Sklansky



Kogge-Stone

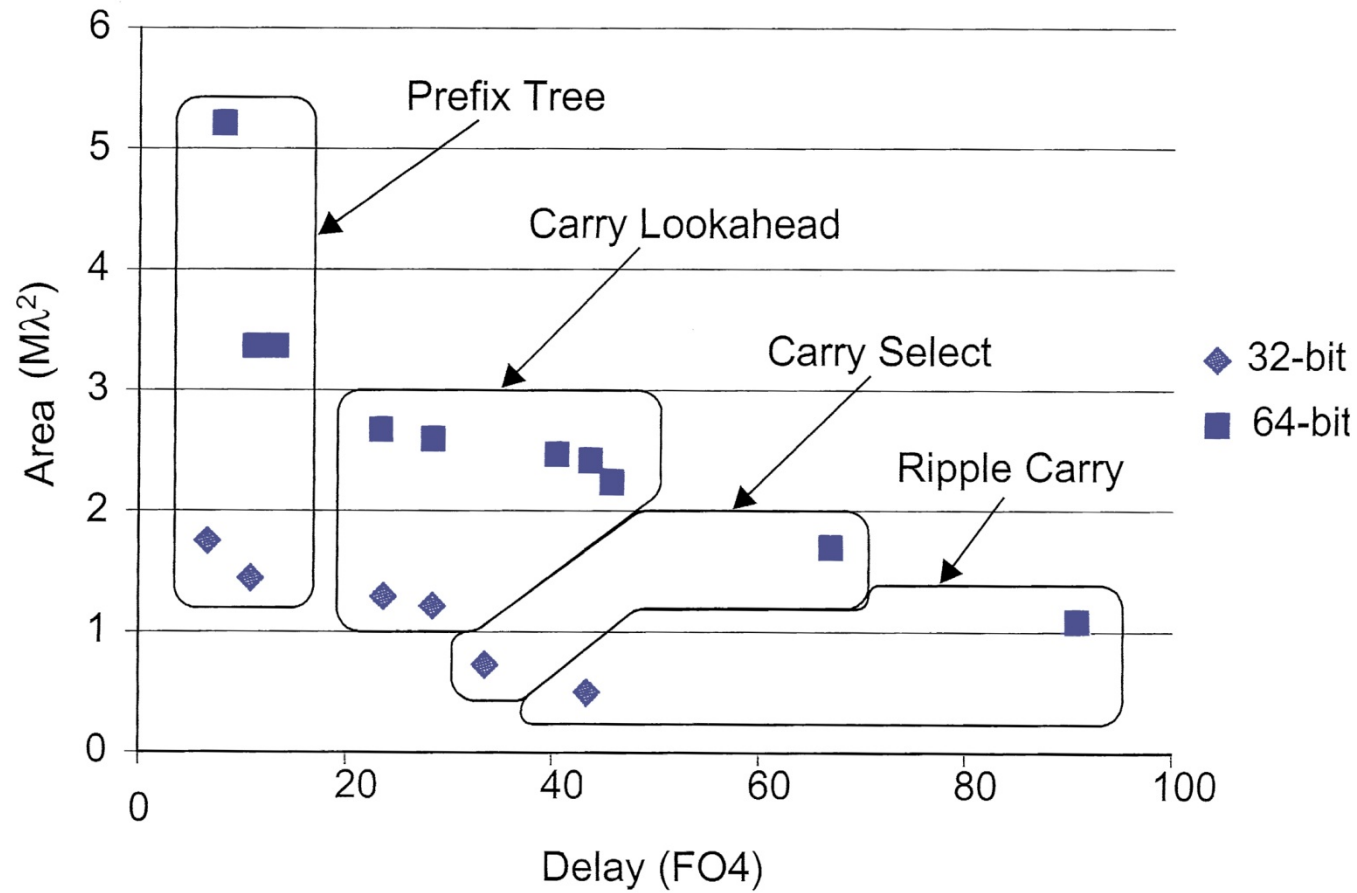


Summary

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

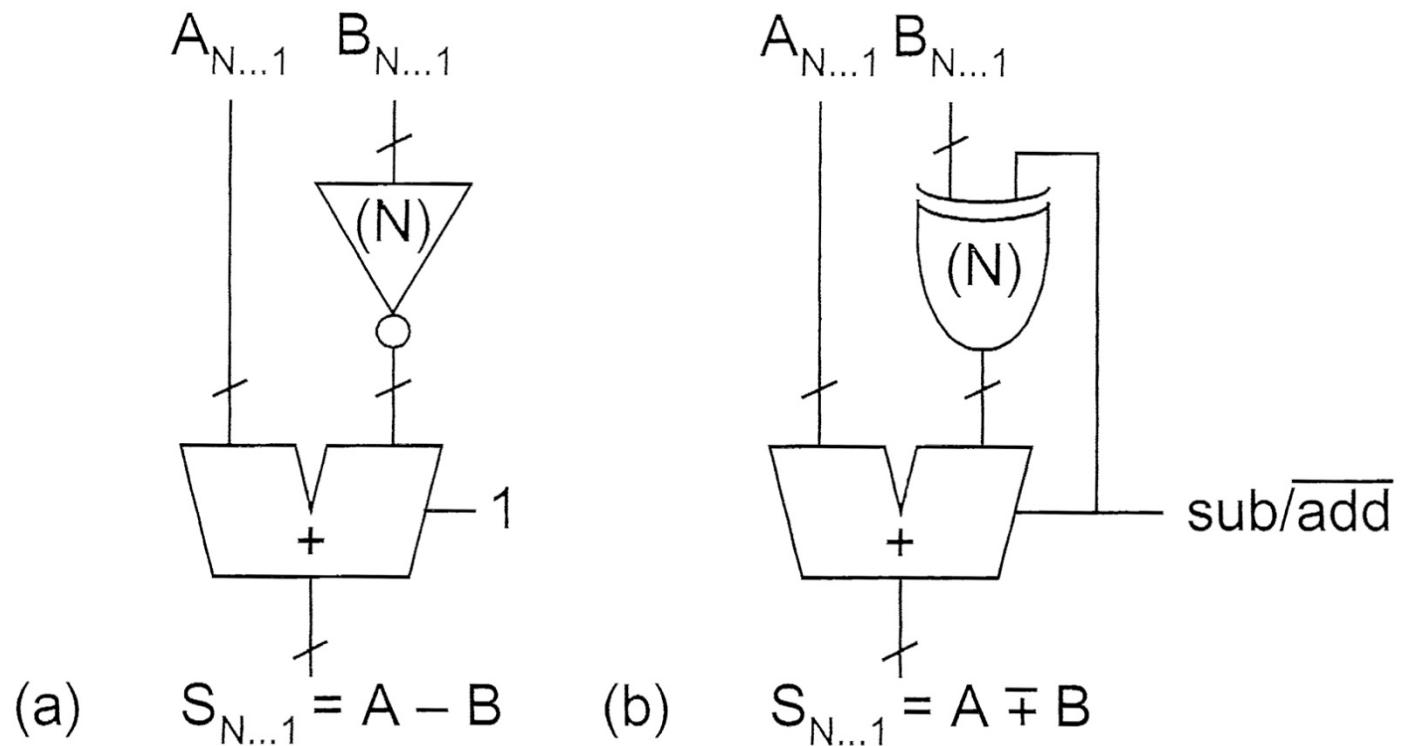
Architecture	Classification	Logic Levels	Max Fanout	Tracks	Cells
Carry-Ripple		$N-1$	1	1	N
Carry-Skip $n=4$		$N/4 + 5$	2	1	$1.25N$
Carry-Inc. $n=4$		$N/4 + 2$	4	1	$2N$
Brent-Kung	$(L-1, 0, 0)$	$2\log_2 N - 1$	2	1	$2N$
Sklansky	$(0, L-1, 0)$	$\log_2 N$	$N/2 + 1$	1	$0.5 N \log_2 N$
Kogge-Stone	$(0, 0,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

Summary



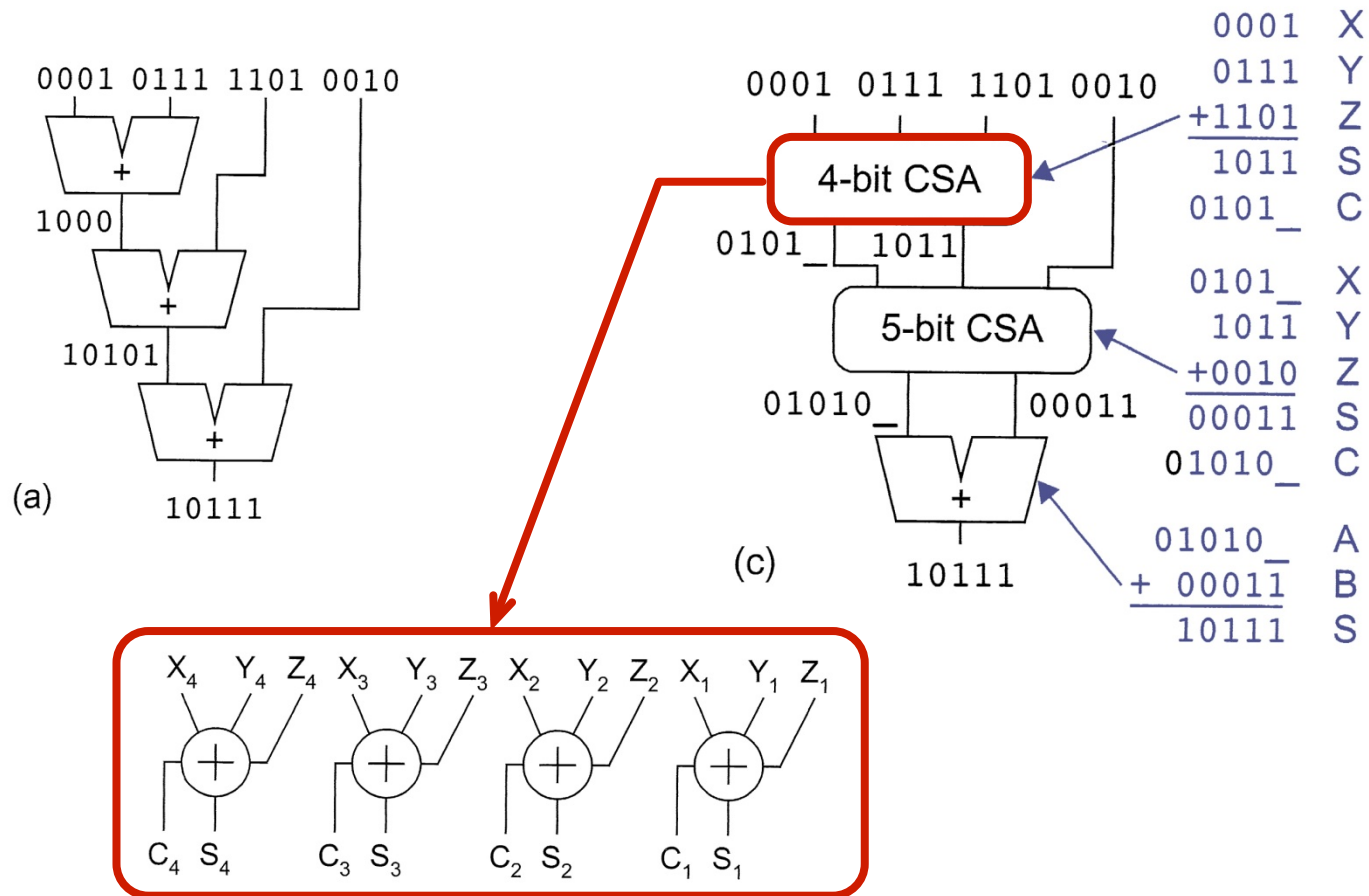
Subtraction

- 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

$$\begin{array}{rcl} 1100 & : & 12_{10} \\ \hline 0101 & : & 5_{10} \end{array}$$

Example

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

Example

1100	:	12_{10}	multiplicand
0101	:	5_{10}	multiplier
1100			] partial products
0000			
1100			
0000			
00111100	:	60_{10}	product

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

General Form

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

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

						y_5	y_4	y_3	y_2	y_1	y_0	
						x_5	x_4	x_3	x_2	x_1	x_0	
						$x_0 y_5$	$x_0 y_4$	$x_0 y_3$	$x_0 y_2$	$x_0 y_1$	$x_0 y_0$	
					$x_1 y_5$	$x_1 y_4$	$x_1 y_3$	$x_1 y_2$	$x_1 y_1$	$x_1 y_0$		
			$x_2 y_5$	$x_2 y_4$	$x_2 y_3$	$x_2 y_2$	$x_2 y_1$	$x_2 y_0$				
		$x_3 y_5$	$x_3 y_4$	$x_3 y_3$	$x_3 y_2$	$x_3 y_1$	$x_3 y_0$					
	$x_4 y_5$	$x_4 y_4$	$x_4 y_3$	$x_4 y_2$	$x_4 y_1$	$x_4 y_0$						
$x_5 y_5$	$x_5 y_4$	$x_5 y_3$	$x_5 y_2$	$x_5 y_1$	$x_5 y_0$							
p_{11}	p_{10}	p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	

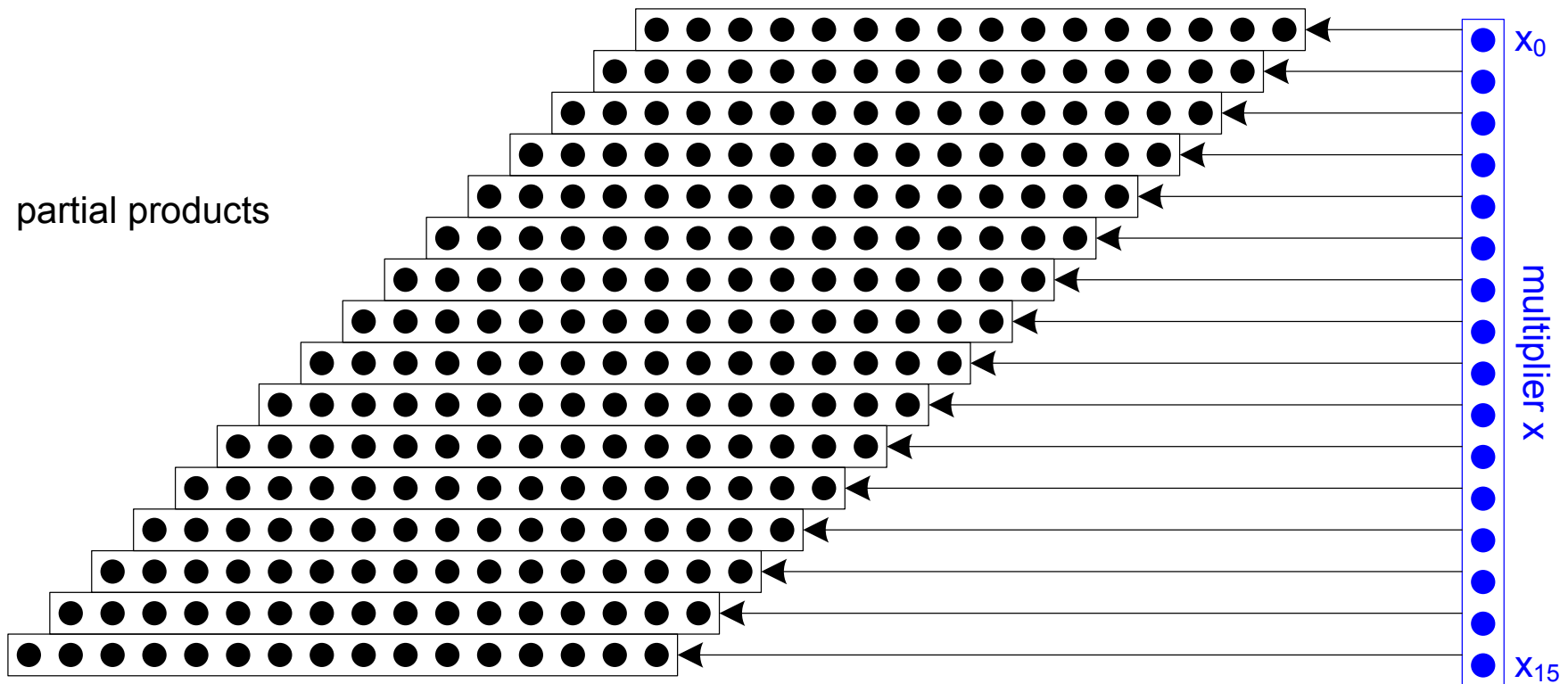
multiplicand
multiplier

partial
products

product

Dot Diagram

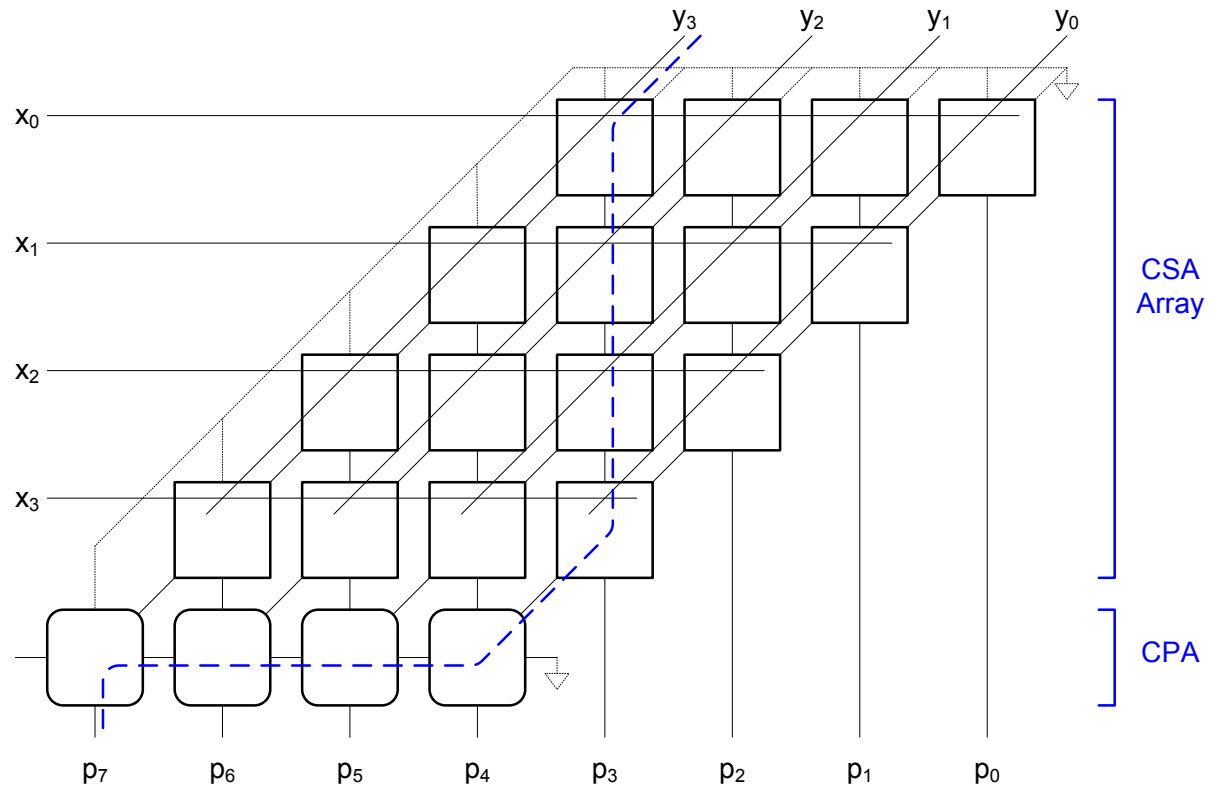
- Each dot represents a bit



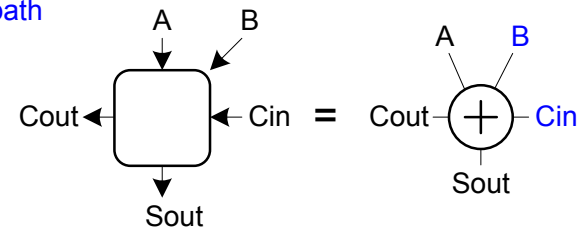
Unsigned Array Multiplication

- 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

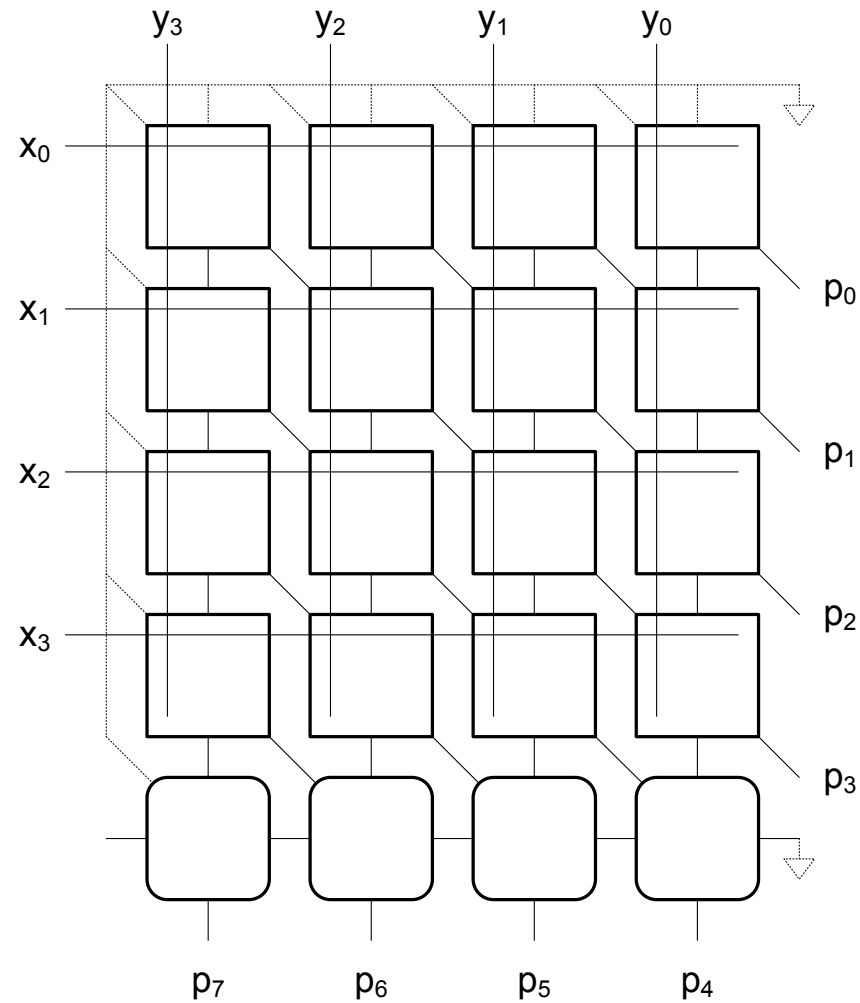


critical path



Rectangle Array

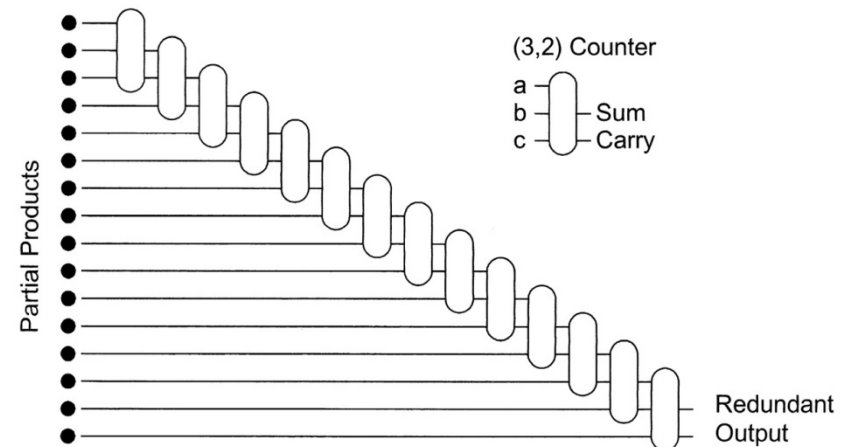
- To fit rectangular floorplan



Summation of Partial Product with CSA

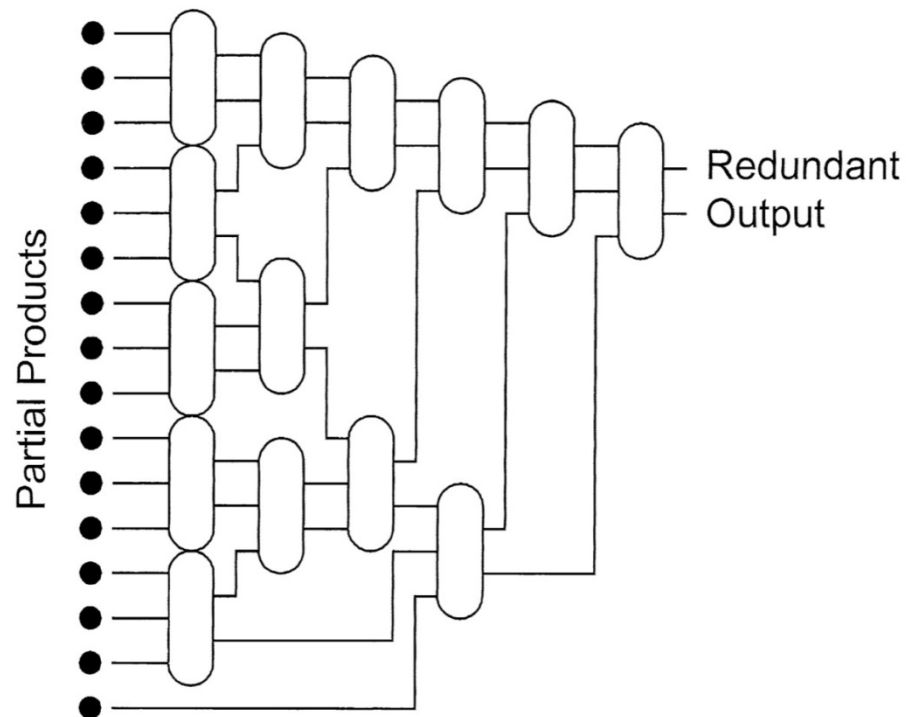
- CSA is effective a “1’s counter”
- Known as (3,2) counter

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



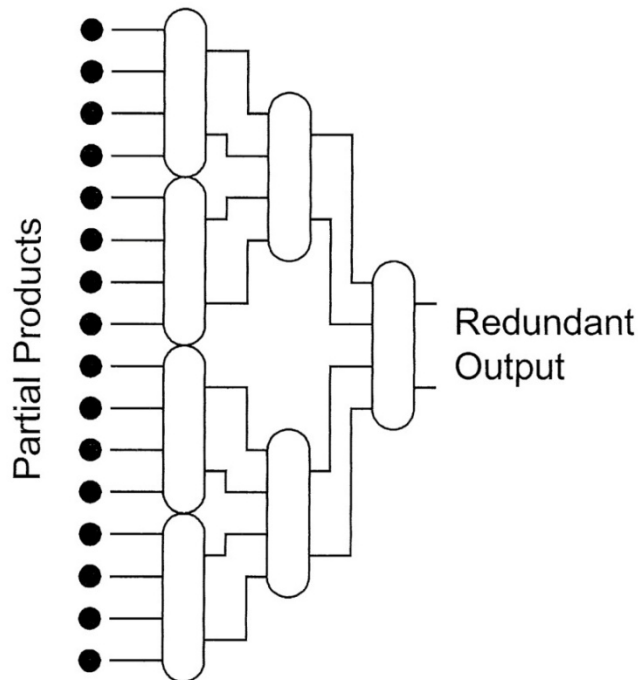
Wallace Tree Multiplication

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

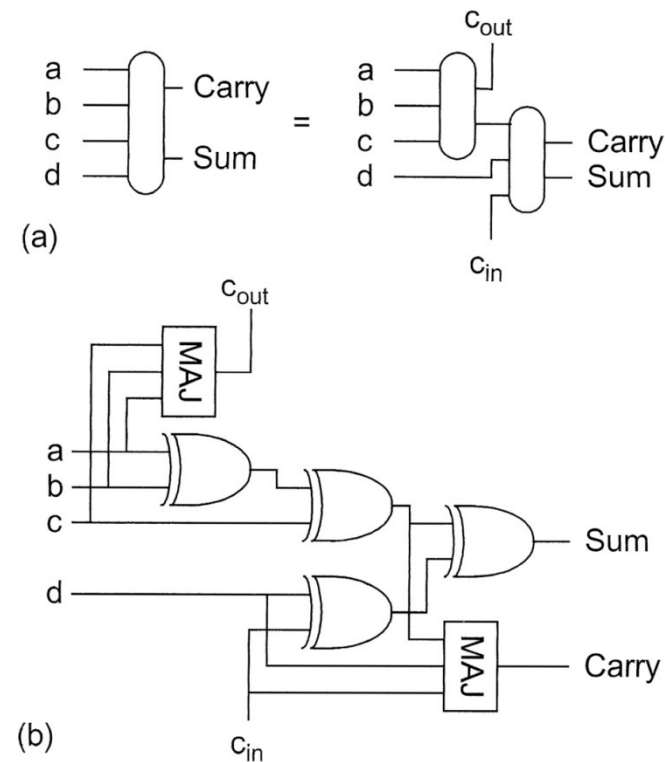


Wallace Tree Multiplication

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



(4:2) compressor



Summary

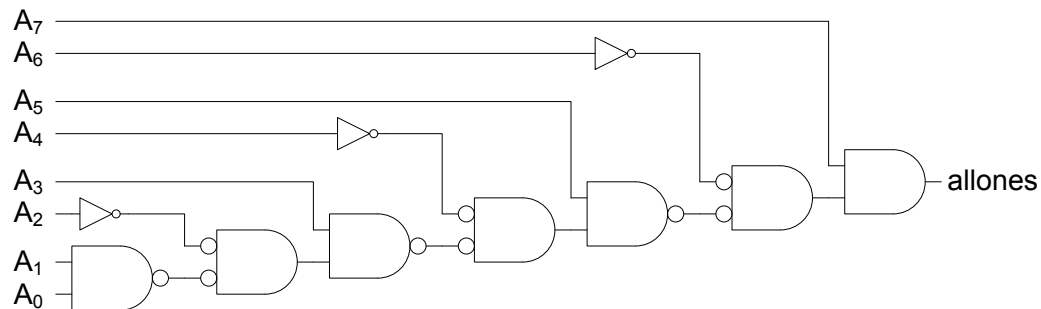
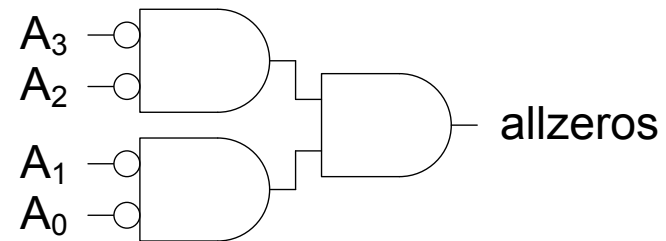
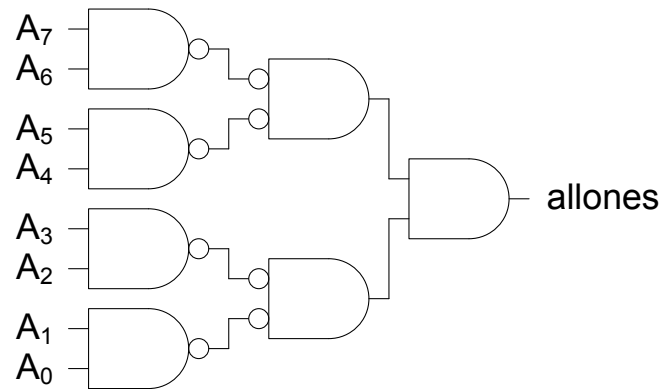
- For synthesis/APR design flow:
choose proper adder/multiplier architectures with various area/speed tradeoff based on applications
- Custom design at schematic level
provides the most performance optimization
- Wiring capacitance needs to be taken care of in multiplier design, compact cell and short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

Outline

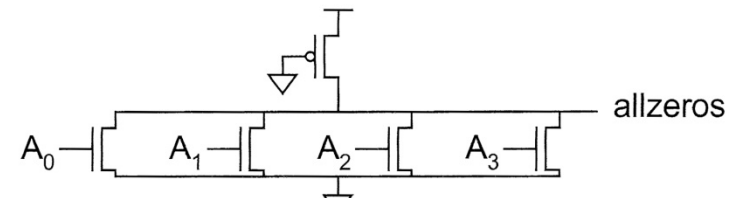
- Addition/Subtraction
- Multiplication
- **One/Zero Detectors**
- Comparators
- Counters
- Shifters

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

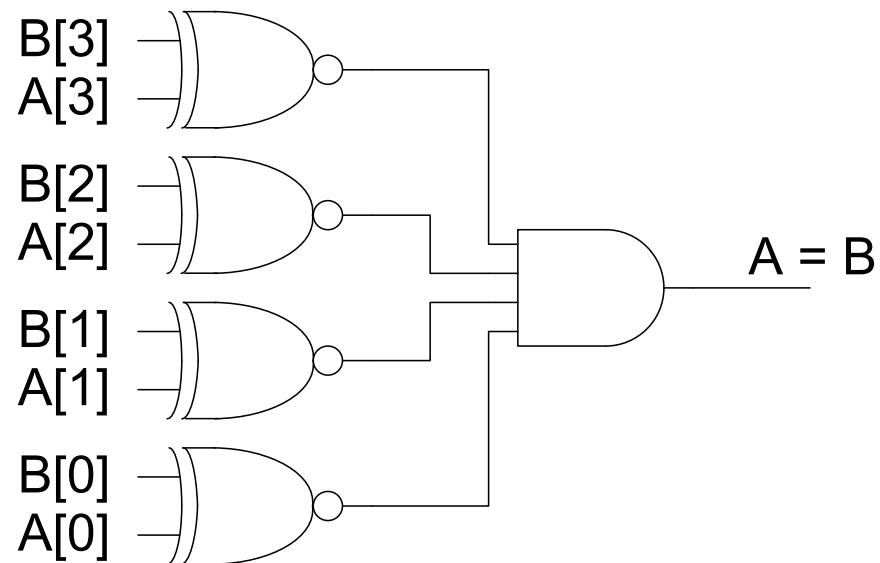
- Addition/Subtraction
- Multiplication
- One/Zero Detectors
- **Comparators**
- Counters
- Shifters

Comparators

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

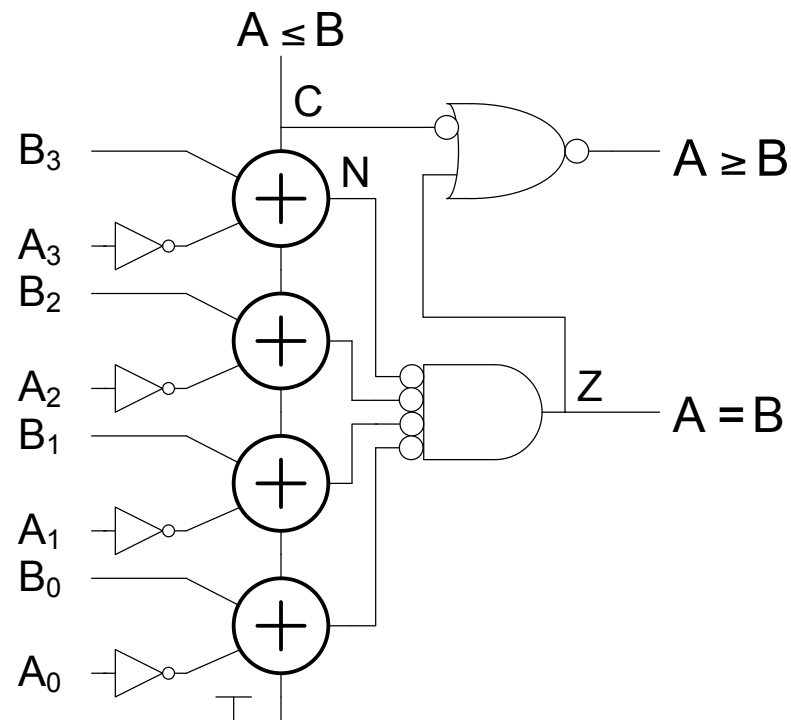
Equality Comparator

- 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



Signed vs. Unsigned

- For signed numbers, comparison is harder
 - C: carry out
 - Z: zero (all bits of A–B are 0)
 - N: negative (MSB of result)
 - V: overflow (inputs had different signs, output sign $\neq$ B)

Table 10.4 Magnitude comparison

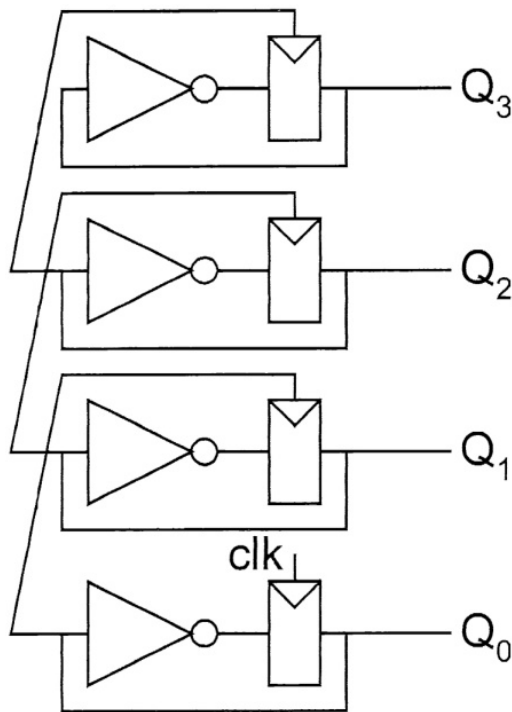
Relation	Unsigned Comparison	Signed Comparison
$A = B$	Z	Z
$A \neq B$	$\overline{Z}$	$\overline{Z}$
$A < B$	$\overline{C + Z}$	$\overline{(N \oplus V) + Z}$
$A > B$	$\overline{C}$	$(N \oplus V)$
$A \leq B$	C	$\overline{(N \oplus V)}$
$A \geq B$	$\overline{C} + Z$	$(N \oplus V) + Z$

Outline

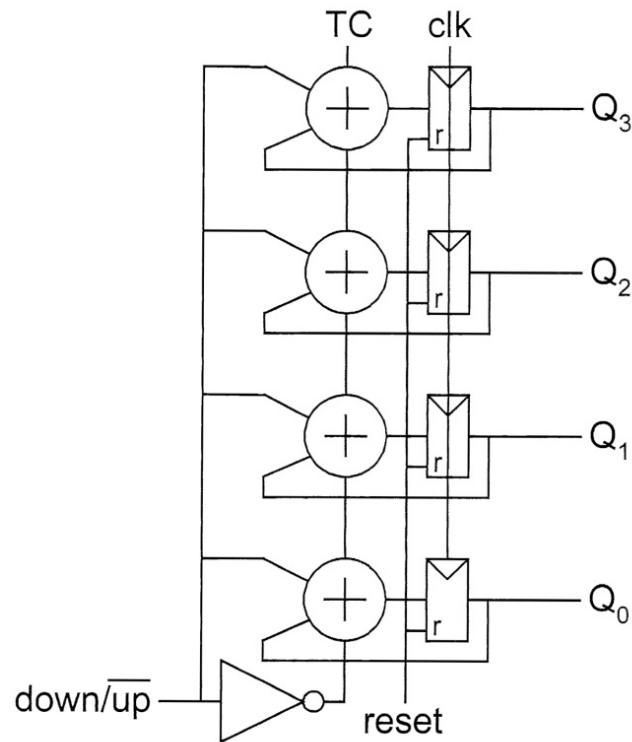
- Addition/Subtraction
- Multiplication
- One/Zero Detectors
- Comparators
- **Counters**
- Shifters

Binary Counters

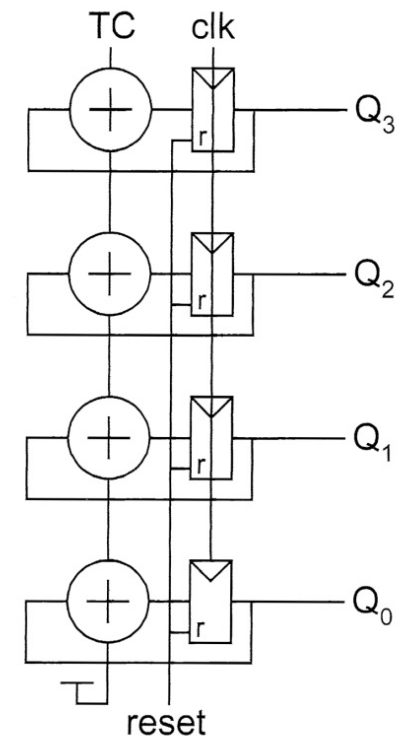
Asynchronous Counter



Synchronous Un/Down Counter



Synchronous Incrementer



(half adder only)

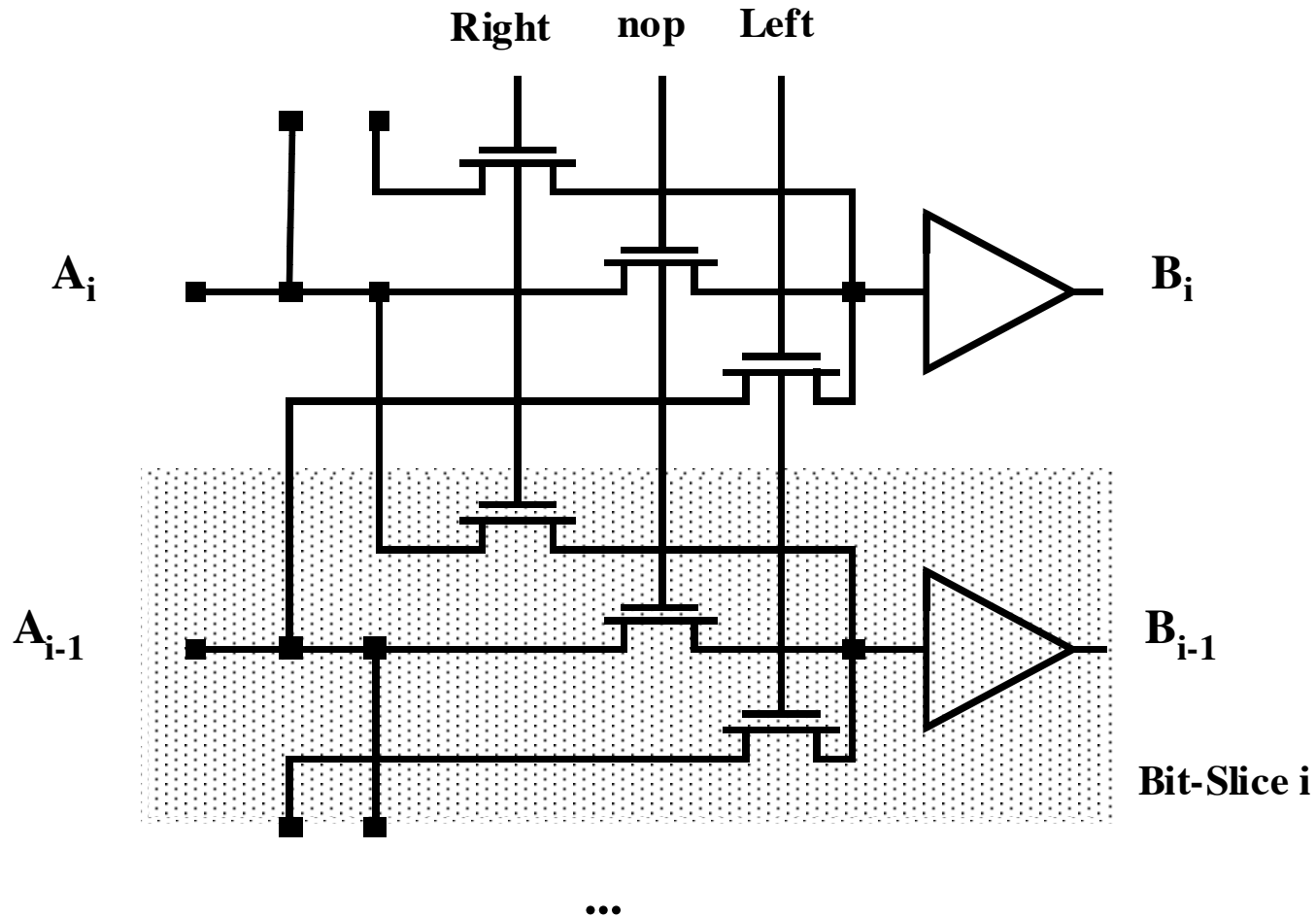
Outline

- Addition/Subtraction
- Multiplication
- One/Zero Detectors
- Comparators
- Counters
- **Shifters**

Shifters

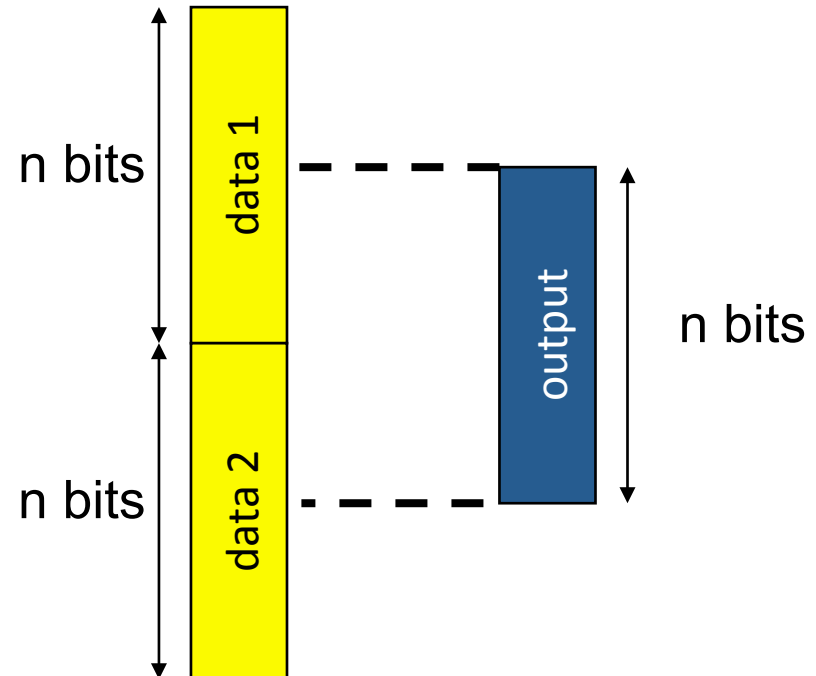
- **Logical Shifter:**
 - Shifts number left or right and fills with 0's
 - $1011 \text{ LSR } 1 = 0101$ $1011 \text{ LSL } 1 = 0110$
- **Arithmetic Shifter:**
 - Shifts number left or right.
Rt shift sign extends, Lf shift fills with 0's
 - $1011 \text{ ASR } 1 = 1101$ $1011 \text{ ASL } 1 = 0110$
- **Barrel Shifter (Rotator):**
 - Shifts number left or right and fills with lost bits
 - $1011 \text{ ROR } 1 = 1101$ $1011 \text{ ROL } 1 = 0111$
- Latch-based shifters only shift one bit per cycle
- Multiple-bit shifters require additional connectivity

Binary Shifter



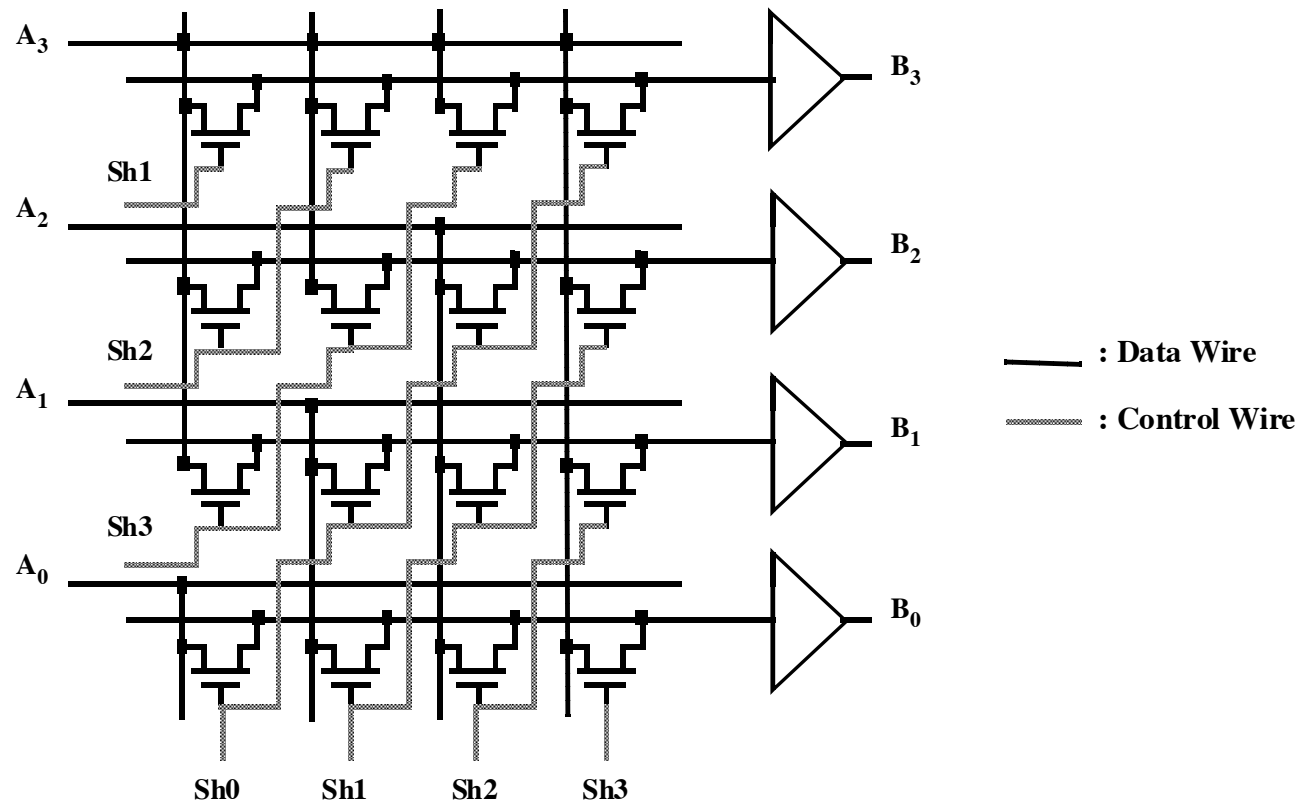
Barrel Shifter

- Can perform n -bit shifts in a single cycle
- Efficient layout
- Require transmission gates and long wires
- Accept $2n$ data inputs and n control signals, produce n data outputs



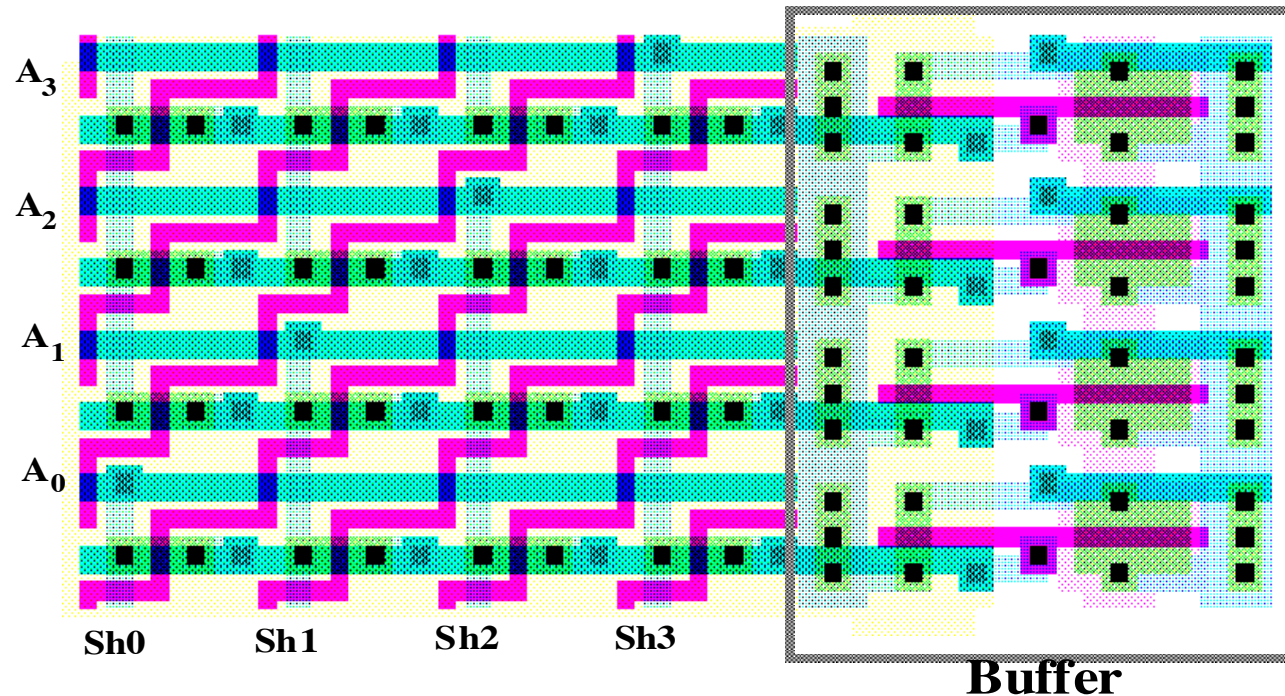
Barrel Shifter Structure

- Two-dimensional array



- Only one control signal is set to 1

4x4 Barrel Shifter



- Large number of cell, but each unit is small
- Area dominated by wires
 - Long delay through wire and transmission gate