Dayananda Sagar College of Engineering
Department of Electronics and Communication

FUNDAMENTALS OF VLSI DESIGN

By
Dr.Usha. C

MODULE-3
CMOS Sub System Design

Introduction

Most Chip functions can be divided into the following categories:

1. Datapath operators
2. Memory elements
3. Control structures
4. Special-purpose cells
    - I/O
    - Power distribution
    - Clock Generation and distribution
    - Analog

- CMOS system design consists of partitioning the system into subsystems
- Many options exist that makes tradeoffs between
    - Speed
    - Density
    - Programmability
    - Ease if design
    - Other variables

Datapath Operators

- Benefit from the structured design principles of hierarchy, regularity, modularity and locality
- N identical circuits used to process N-bit data
- Related data operators are place physically adjacent to each other to reduce wire length and delay
- Data is arranged to flow in one direction, while control signals are introduced in a direction orthogonal to the dataflow
- Common datapath operators include
  - Adders
  - One/Zero detectors
  - Comparators
  - Counters
  - Boolean logic Units
  - Error correcting code blocks
  - Shifters
  - Multipliers
  - Dividers

Addition/Subtraction

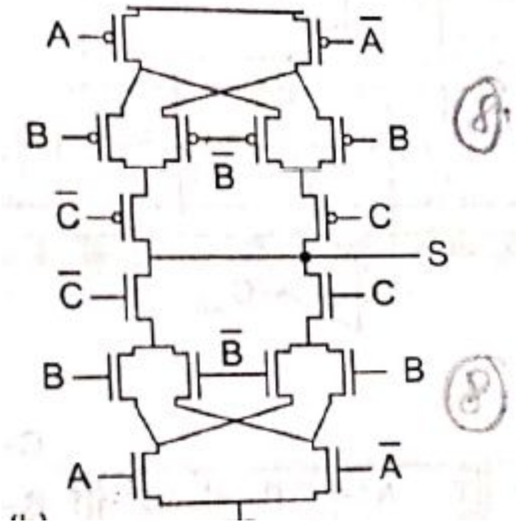
- Addition forms basis for many processing operations from counting to multiplication to filtering
- Adder architectures serve different speed/ area requirements

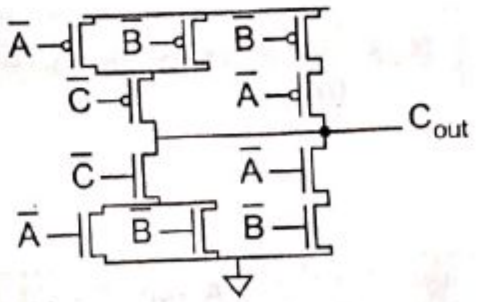
Single Bit Addition

Full Adder

Half Adder

Full Adder Design(CMOS Circuit)


A
$\bar{A}$
B
$\bar{B}$
B
$\bar{C}$
C
S
$\bar{C}$
C
B
$\bar{B}$
B
A
$\bar{A}$


$\bar{A}$
$\bar{B}$
$\bar{B}$
$\bar{C}$
$\bar{A}$
C_{out}
$\bar{C}$
$\bar{A}$
$\bar{A}$
$\bar{B}$
$\bar{B}$

A Compact design to reduce the number of transistor is that S can be factored to reverse Cout term

PTL/CPL logic for Full adder Design

Transmission Gate Full Adder Design

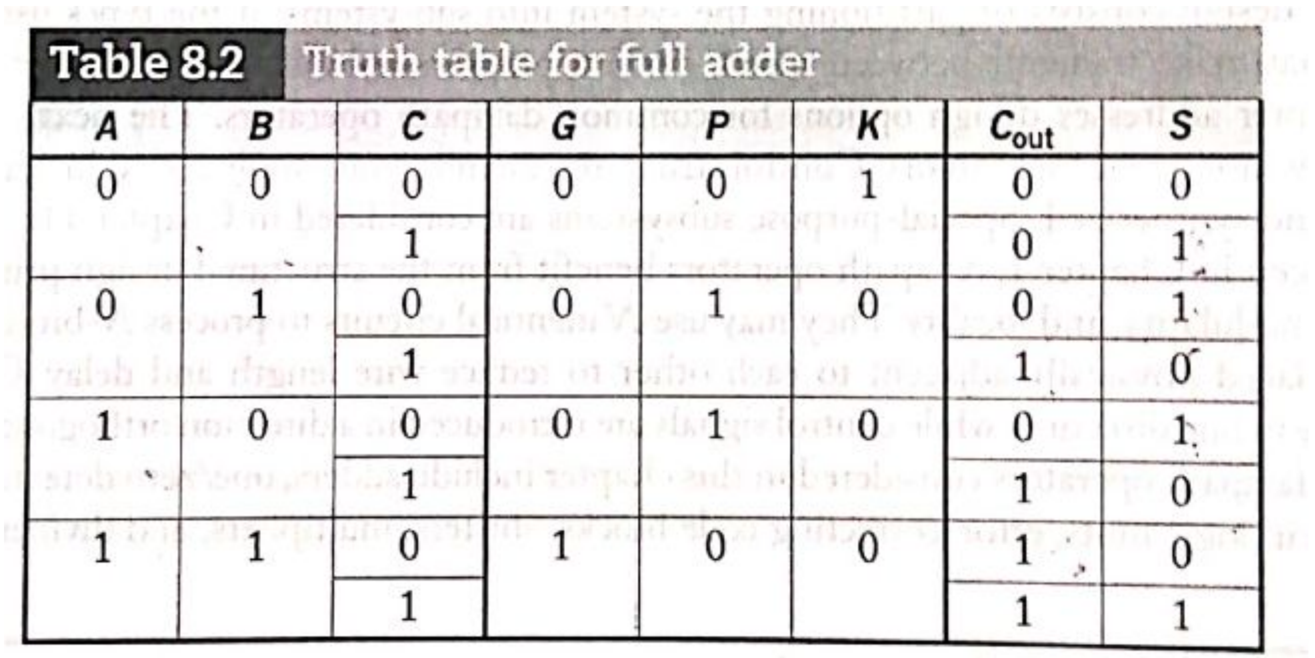
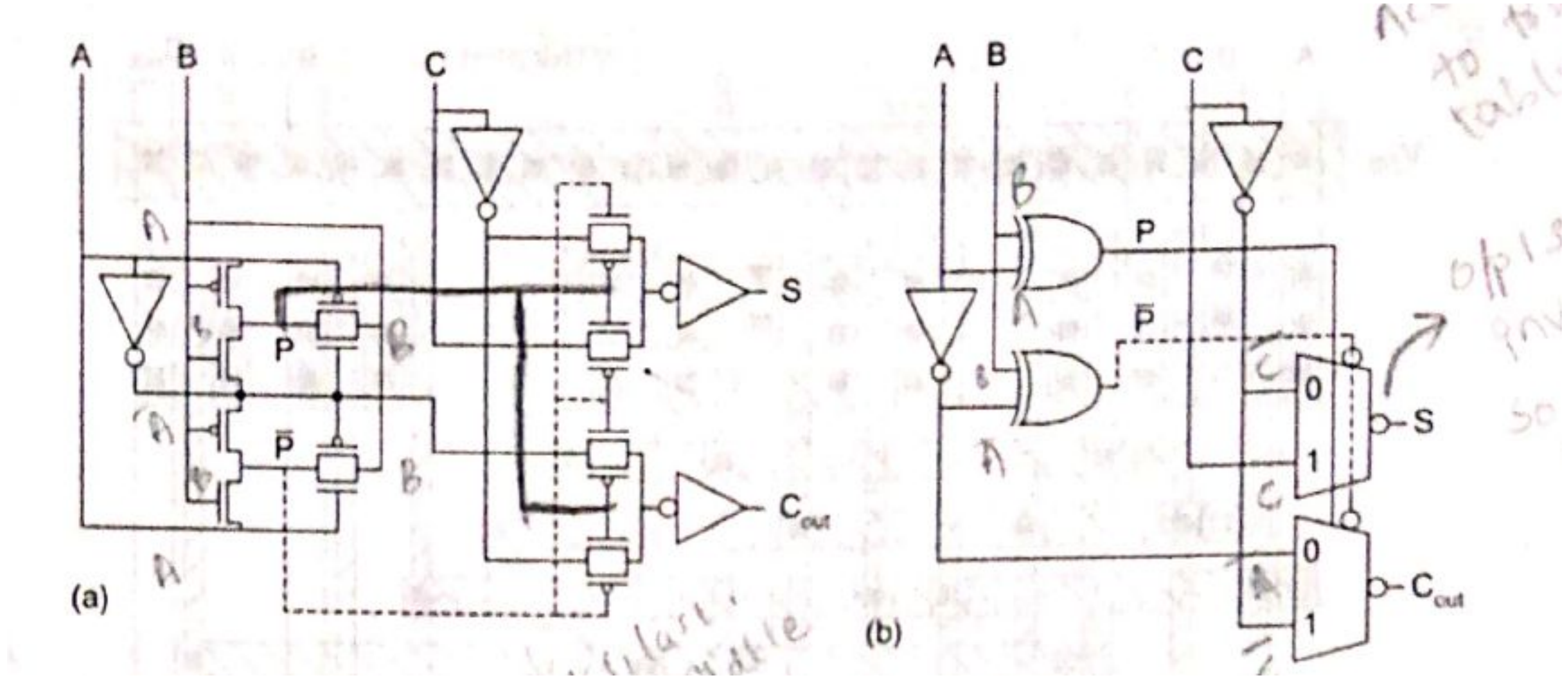


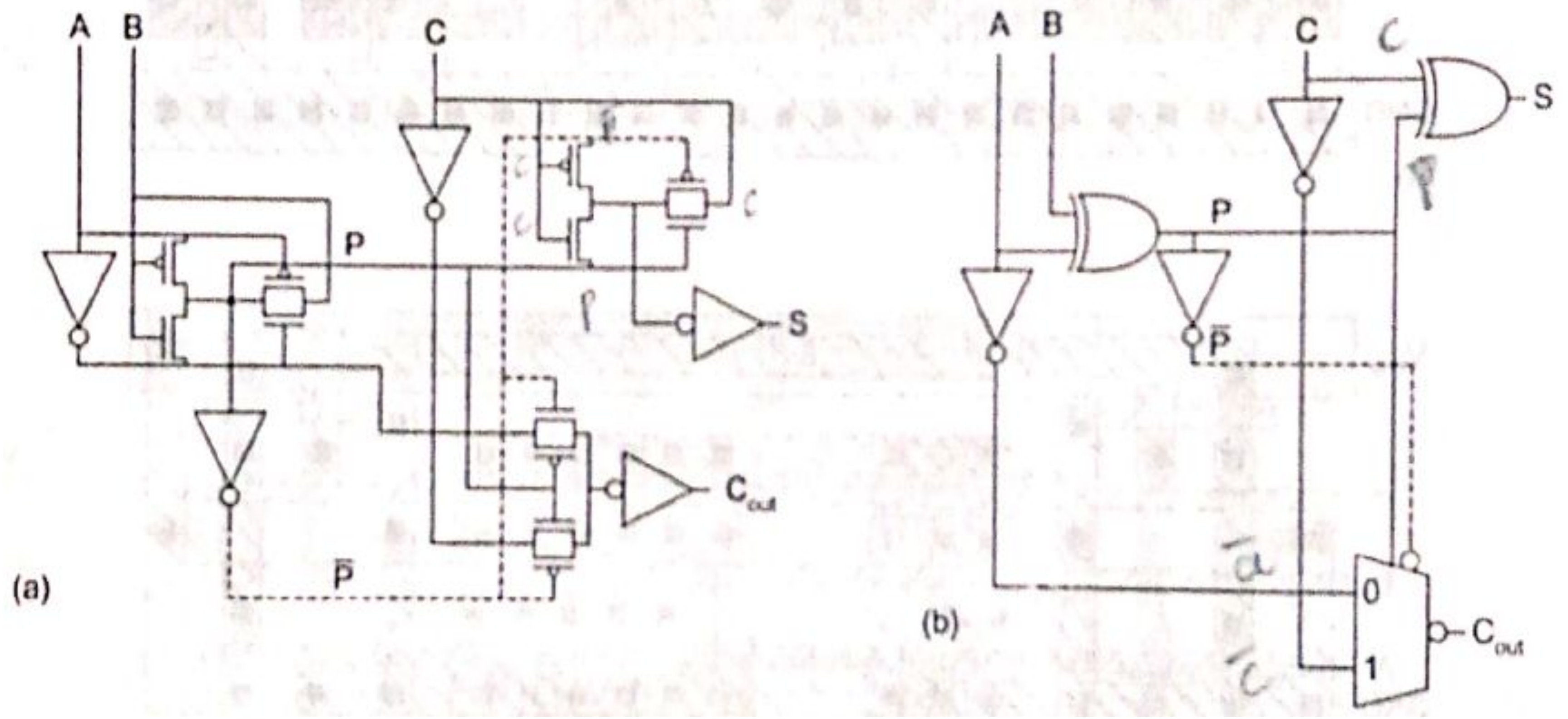
Table 8.2 Truth table for full adder

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

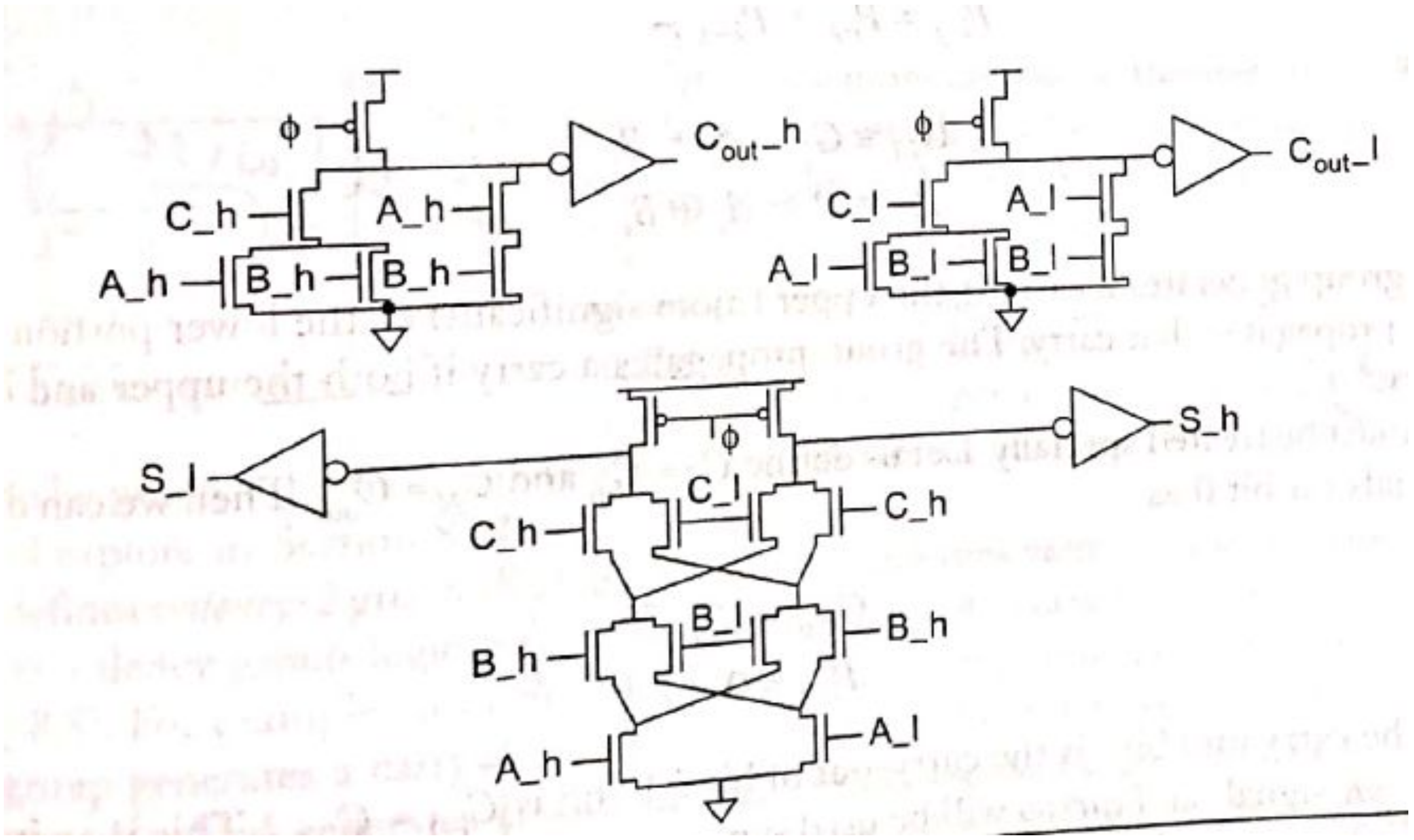
Full adder Design using Transmission gates to form Multiplexers and XOR's



Zuhang Full Adder



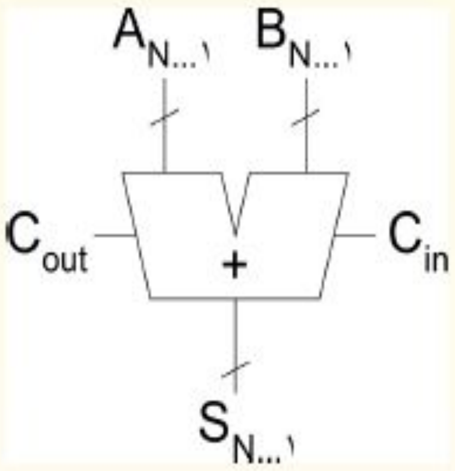
Dual Rail Domino Full Adder

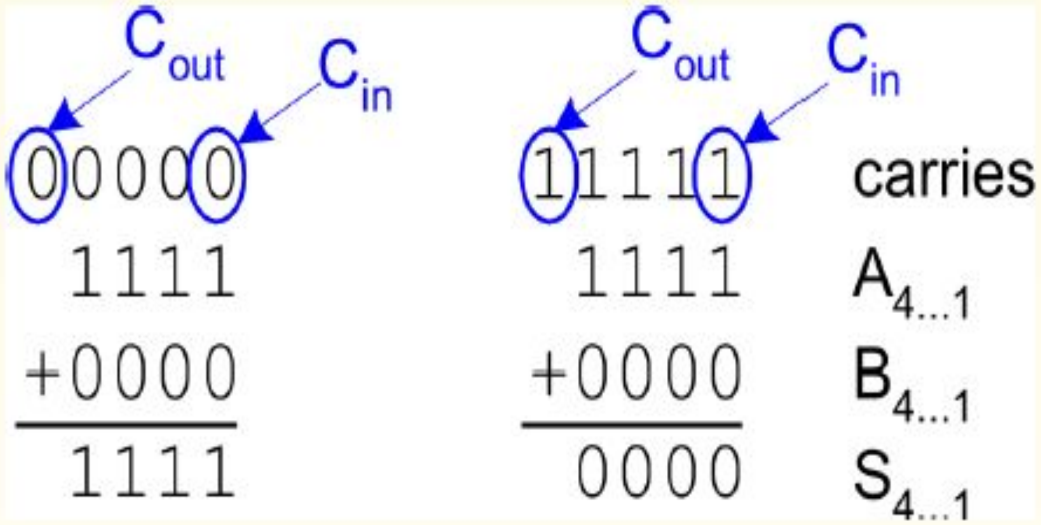


Carry Propagate Addition

BN…………….B
AN…….A1

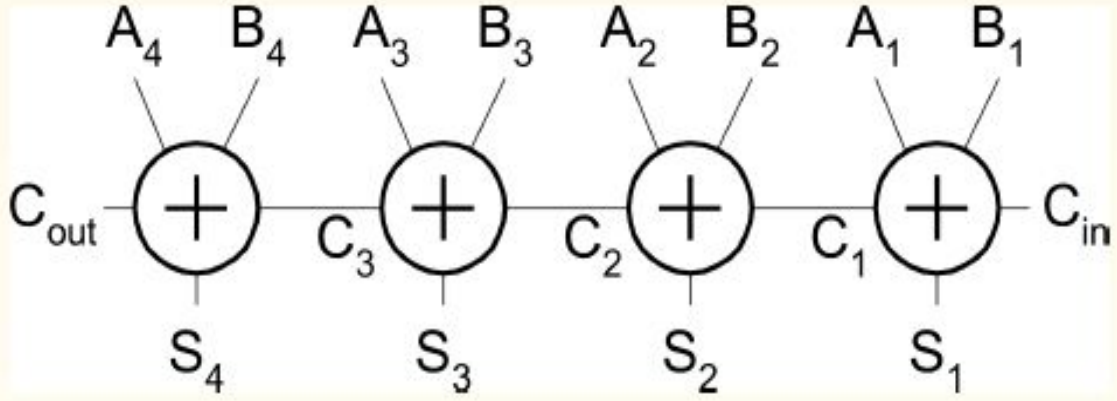
- N-bit adder called CPA
  - Each sum bit depends on all previous carries
  - How do we compute all these carries quickly?



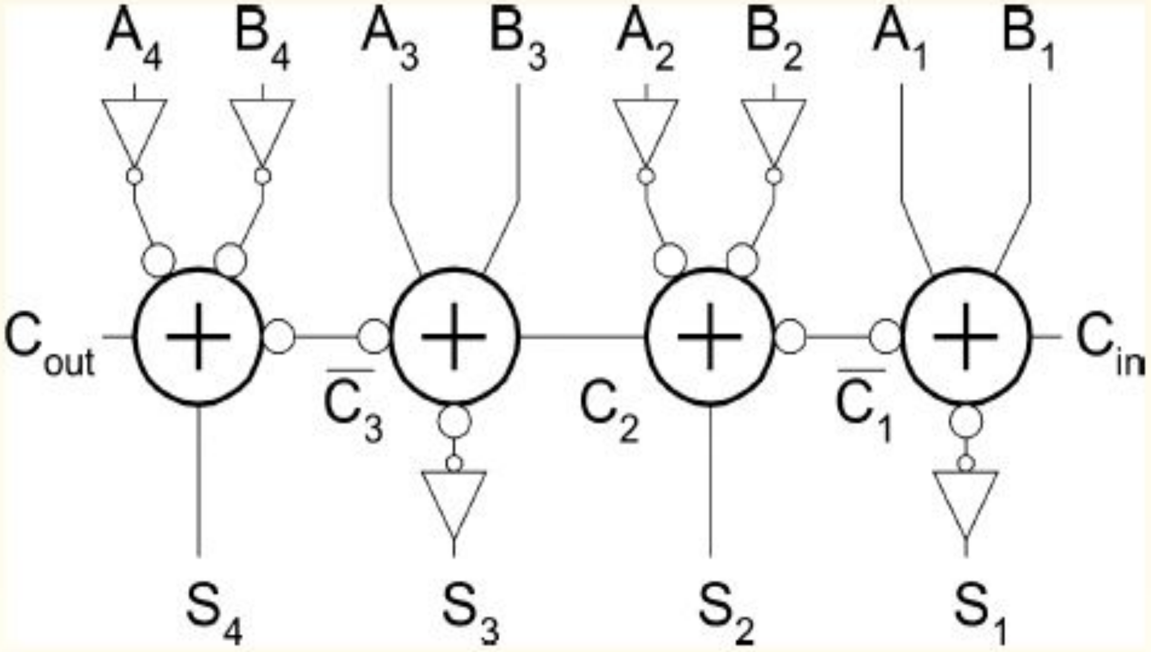


Carry Ripple Adder

- Simplest design: cascade full adders
  - Critical path goes from C_{in} to C_{out}
  - Design full adder to have fast carry (small delay for carry signal)



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



Ci-1=Gi-1:0

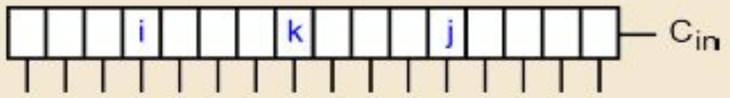
Carry Generation and Propagation

For a full adder, define what happens to carries

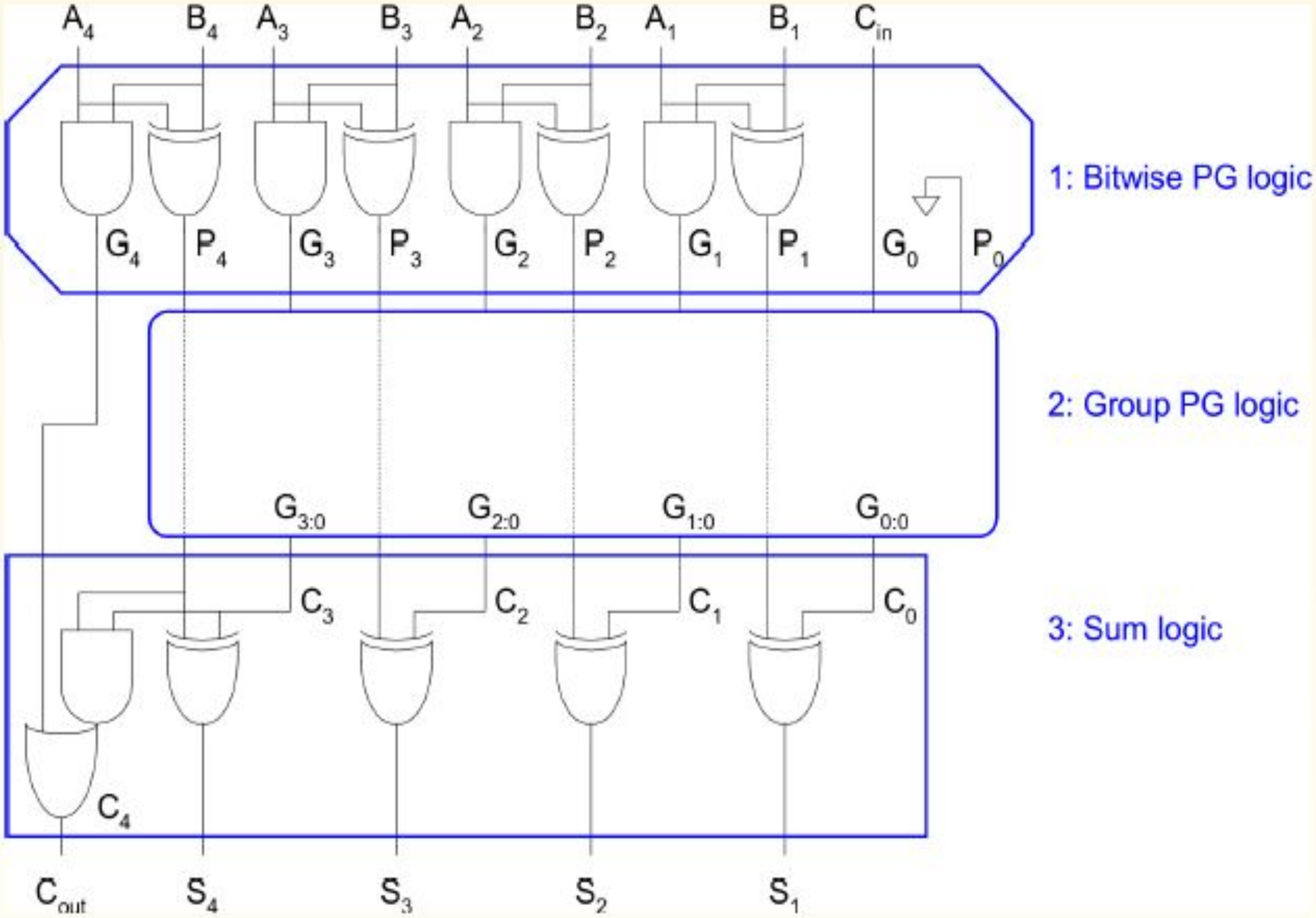
- **Generate**: $C_{out} = 1$, independent of C
  - $G = A \cdot B$
- **Propagate**: $C_{out} = C$
  - $P = A \oplus B$
- **Kill**: $C_{out} = 0$, independent of C
  - $K = \overline{A} \cdot \overline{B}$

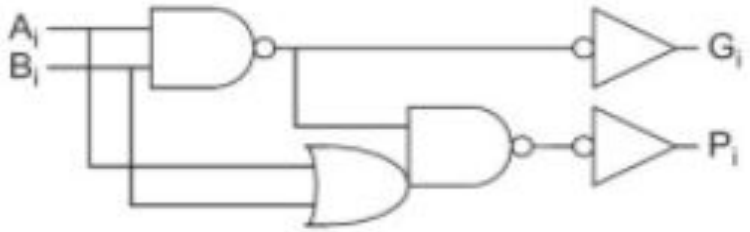
Generate and Propagate for groups spanning i:j

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



- Base Case
  - $G_{i:i} \equiv G_i = A_i \cdot B_i,\qquad G_{0:0} = G_0 = C_{in}$
  - $P_{i:i} \equiv P_i = A_i \oplus B_i,\qquad P_{0:0} = P_0 = 0$
- Sum: $S_i = P_i \oplus G_{i-1:0}$


A_4
B_4
A_3
B_3
A_2
B_2
A_1
B_1
C_{in}
G_4
P_4
G_3
P_3
G_2
P_2
G_1
P_1
G_0
P_0
1: Bitwise PG logic
2: Group PG logic
$G_{3:0}$
$G_{2:0}$
$G_{1:0}$
$G_{0:0}$
C_3
C_2
C_1
C_0
3: Sum logic
C_4
C_{out}
S_4
S_3
S_2
S_1



shared bitwise propagate-generate (PG) logic

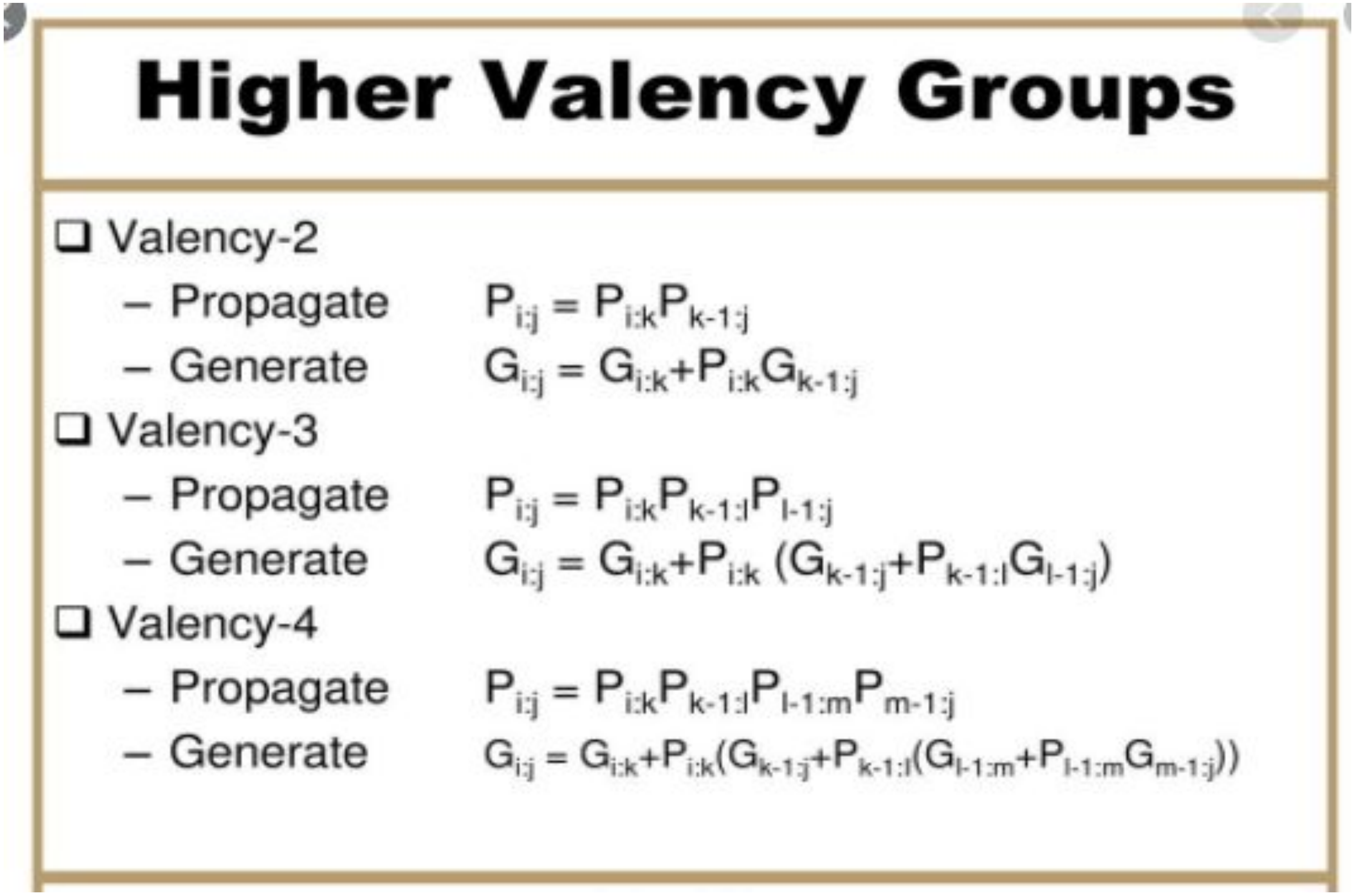
$$G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j} \qquad P_{i:j} = P_{i:k} \cdot P_{k-1:j}$$

Combines pair of smaller groups is called valency-2 group PG logic.

To use fewer stages for carry propagation, higher valency comprising more complex gates is possible, e.g. valency-4:

$$P_{i:j} = P_{i:k} \cdot P_{k-1:l} \cdot P_{l-1:m} \cdot P_{m-1:j} \qquad i \geq k > l > m > j$$

$$G_{i:j} = G_{i:k} + P_{i:k}\left(G_{k-1:l} + P_{k-1:l}\left(G_{l-1:m} + P_{l-1:m} \cdot G_{m-1:j}\right)\right)$$

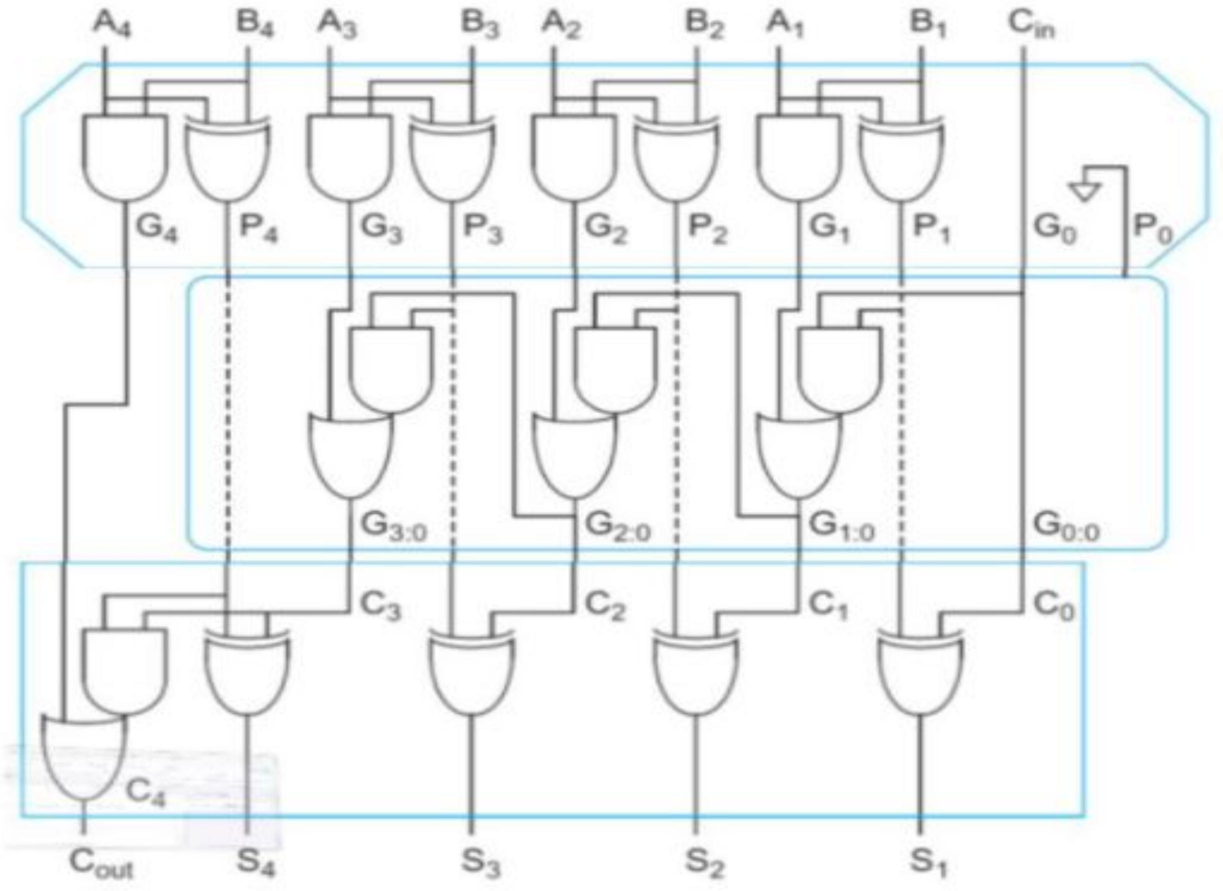


Higher Valency Groups

- Valency-2
  - Propagate $P_{i:j} = P_{i:k}P_{k-1:j}$
  - Generate $G_{i:j} = G_{i:k}+P_{i:k}G_{k-1:j}$
- Valency-3
  - Propagate $P_{i:j} = P_{i:k}P_{k-1:l}P_{l-1:j}$
  - Generate $G_{i:j} = G_{i:k}+P_{i:k}\,(G_{k-1:j}+P_{k-1:l}G_{l-1:j})$
- Valency-4
  - Propagate $P_{i:j} = P_{i:k}P_{k-1:l}P_{l-1:m}P_{m-1:j}$
  - Generate $G_{i:j} = G_{i:k}+P_{i:k}(G_{k-1:j}+P_{k-1:l}(G_{l-1:m}+P_{l-1:m}G_{m-1:j}))$

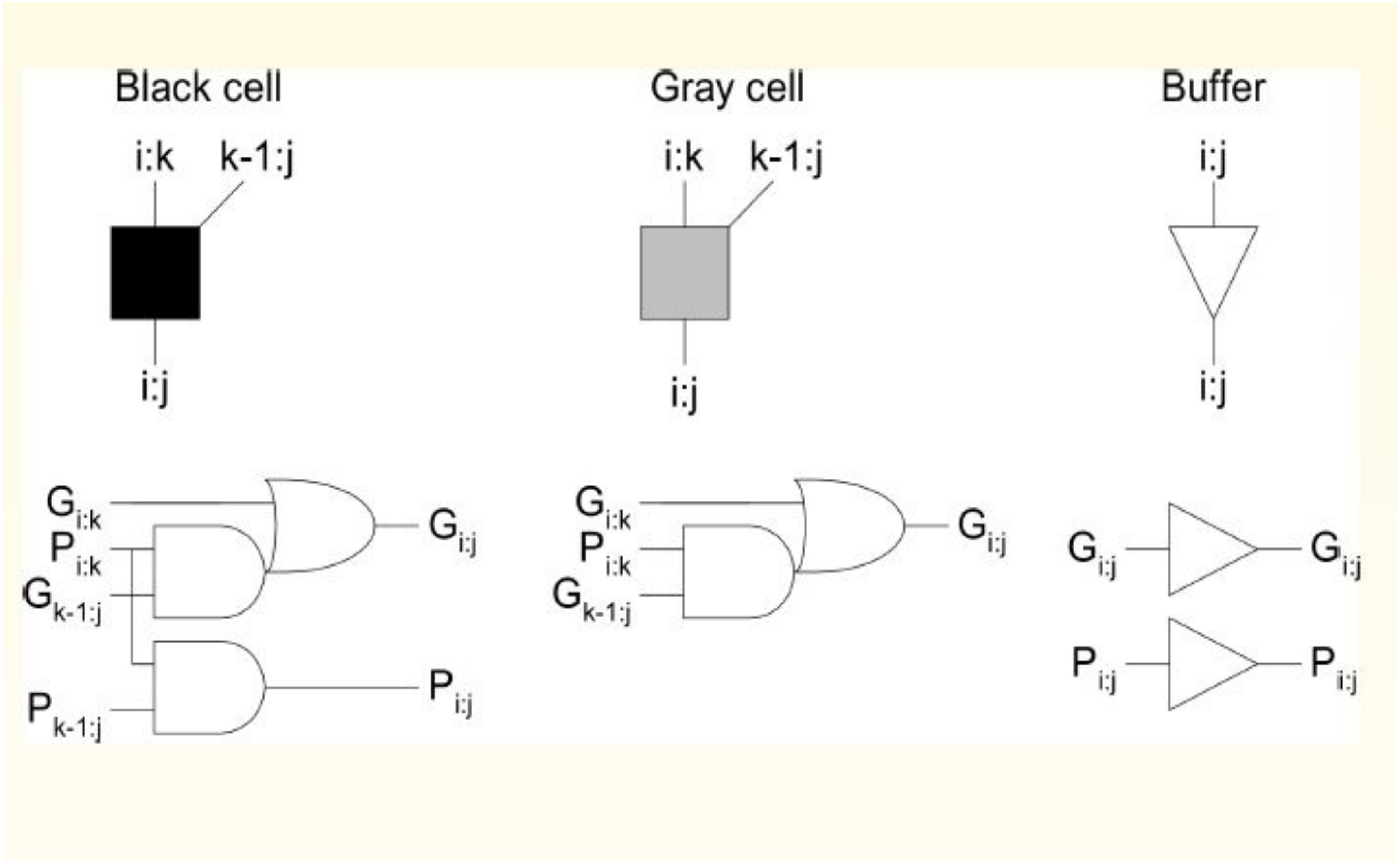
PG Carry-Ripple Addition

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

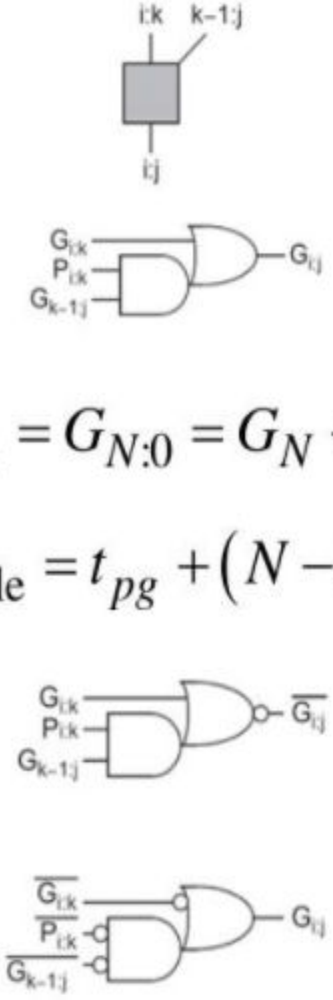


Ci=Gi:0

Black Cell: Group Generate and propagate logic
Gray Cell: Group generate logic

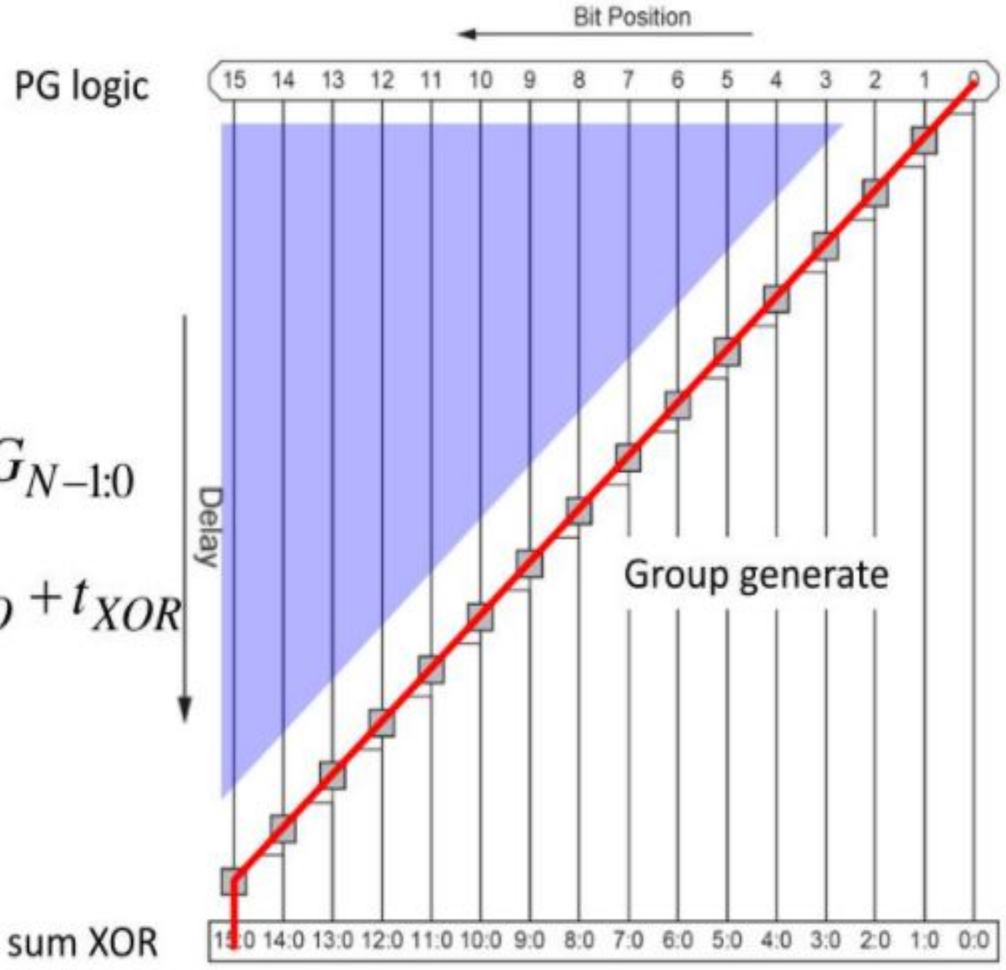


Adder architecture diagram

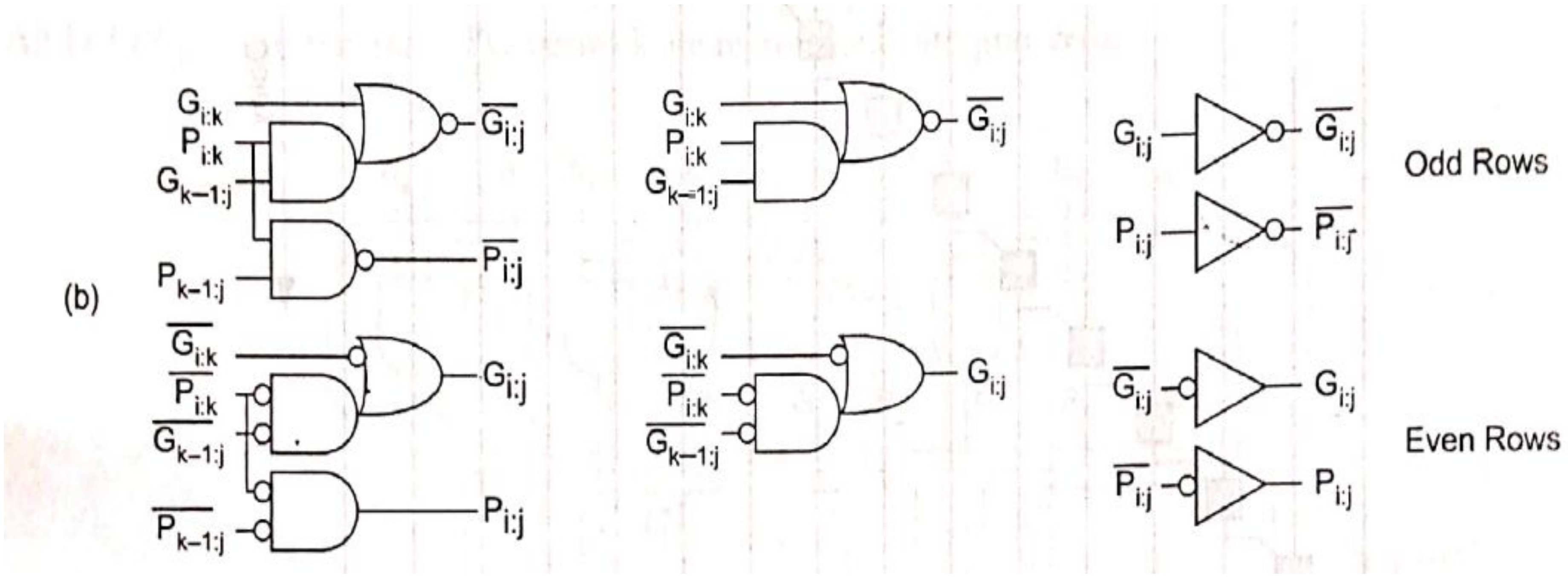


$$C_{\text{out}} = G_{N:0} = G_N + P_N G_{N-1:0}$$

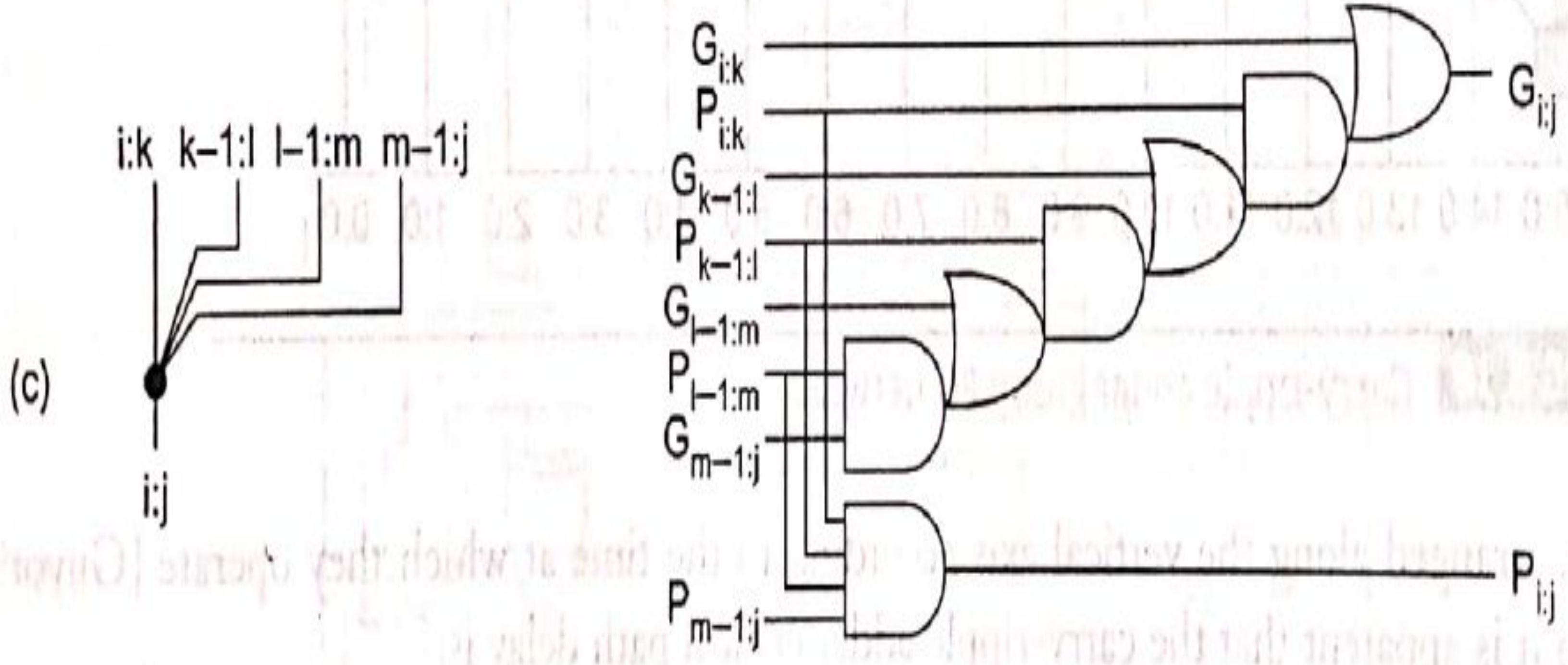
$$t_{\text{ripple}} = t_{pg} + (N-1)t_{AO} + t_{XOR}$$



PG network to remove extraneous inverters

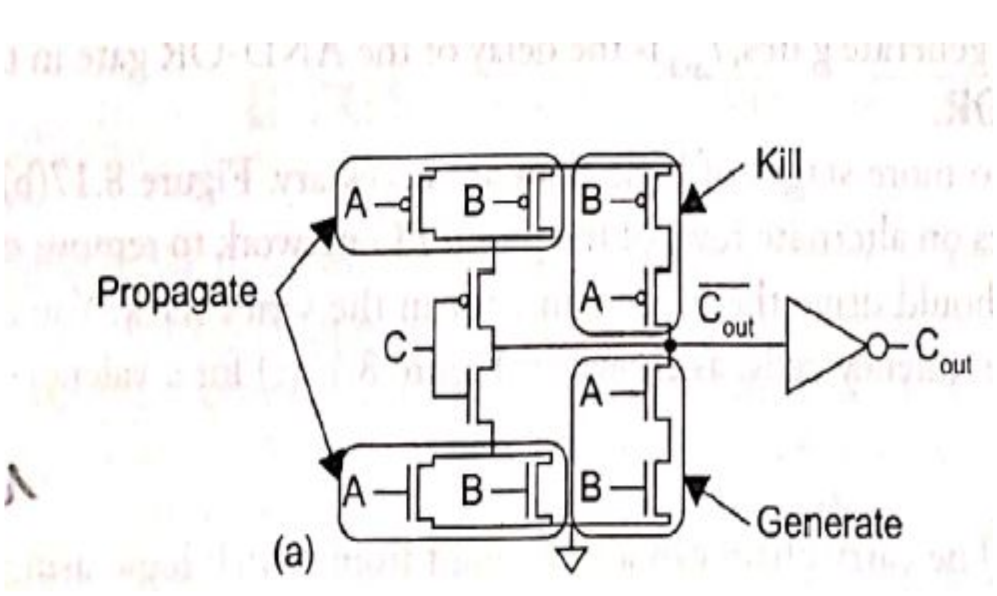


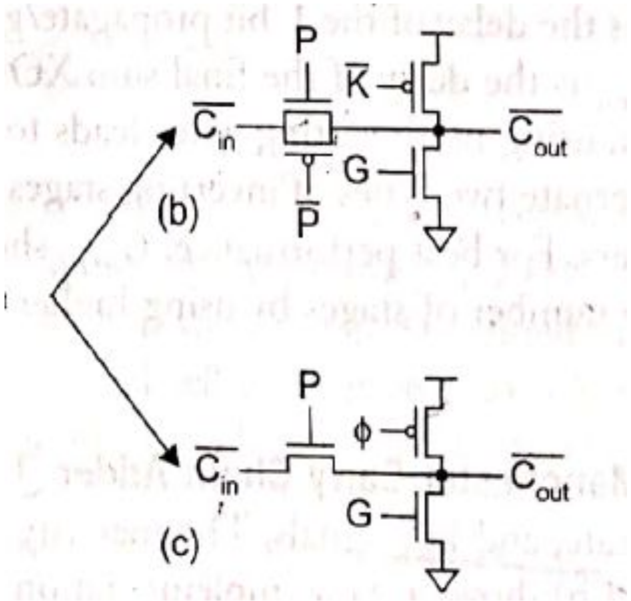
Reduce the number of stages by using higher valency cells



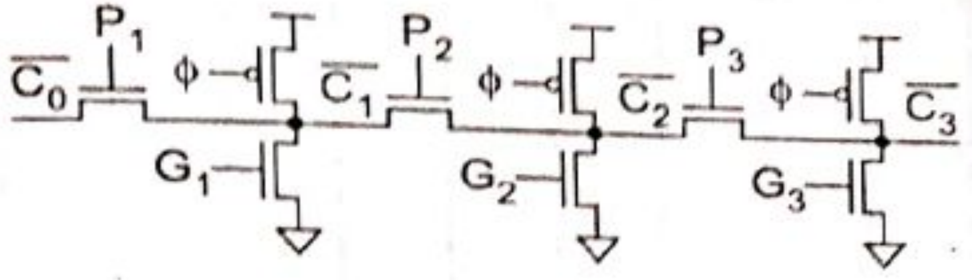
Manchester Carry Chain Adder

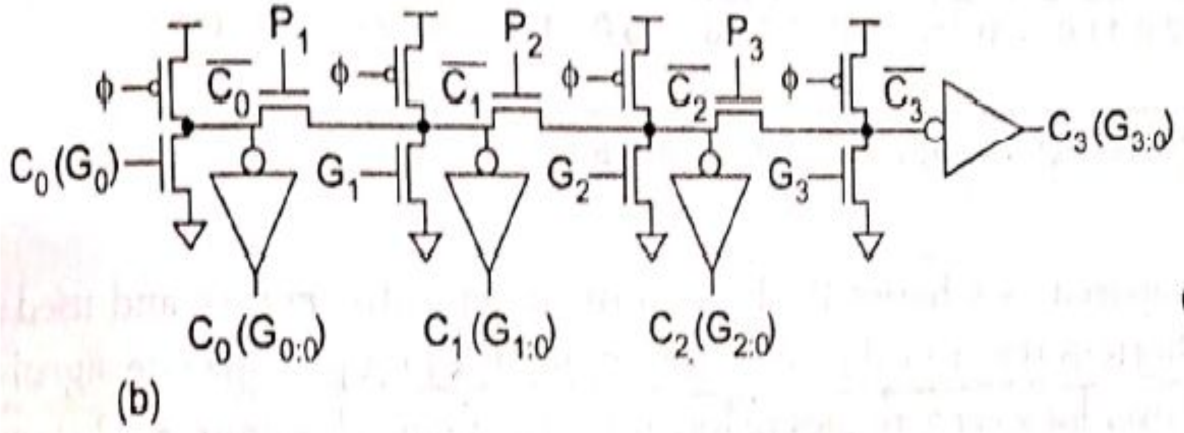
Carry Chain Designs



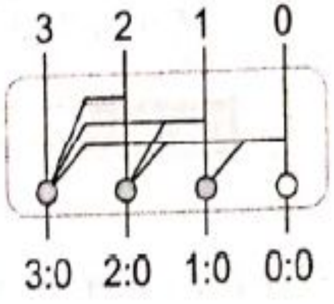


Multiple stages



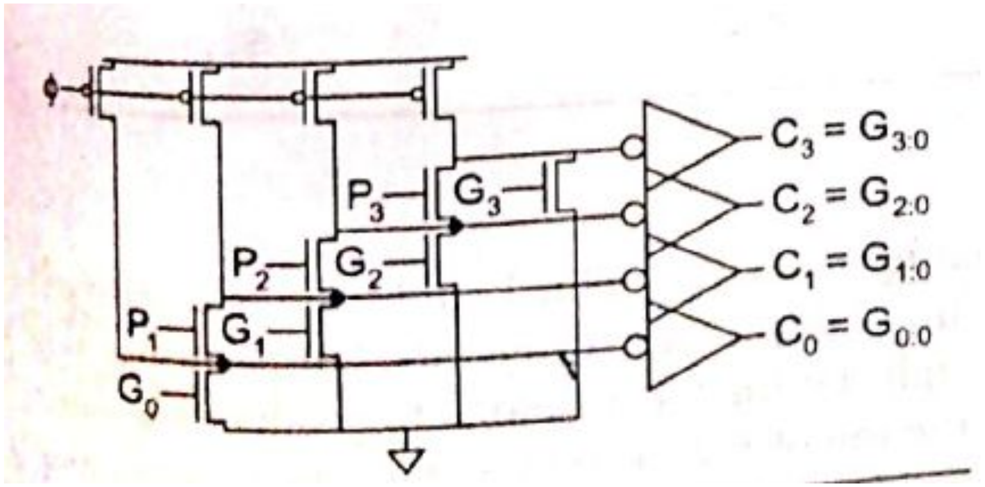


(b)

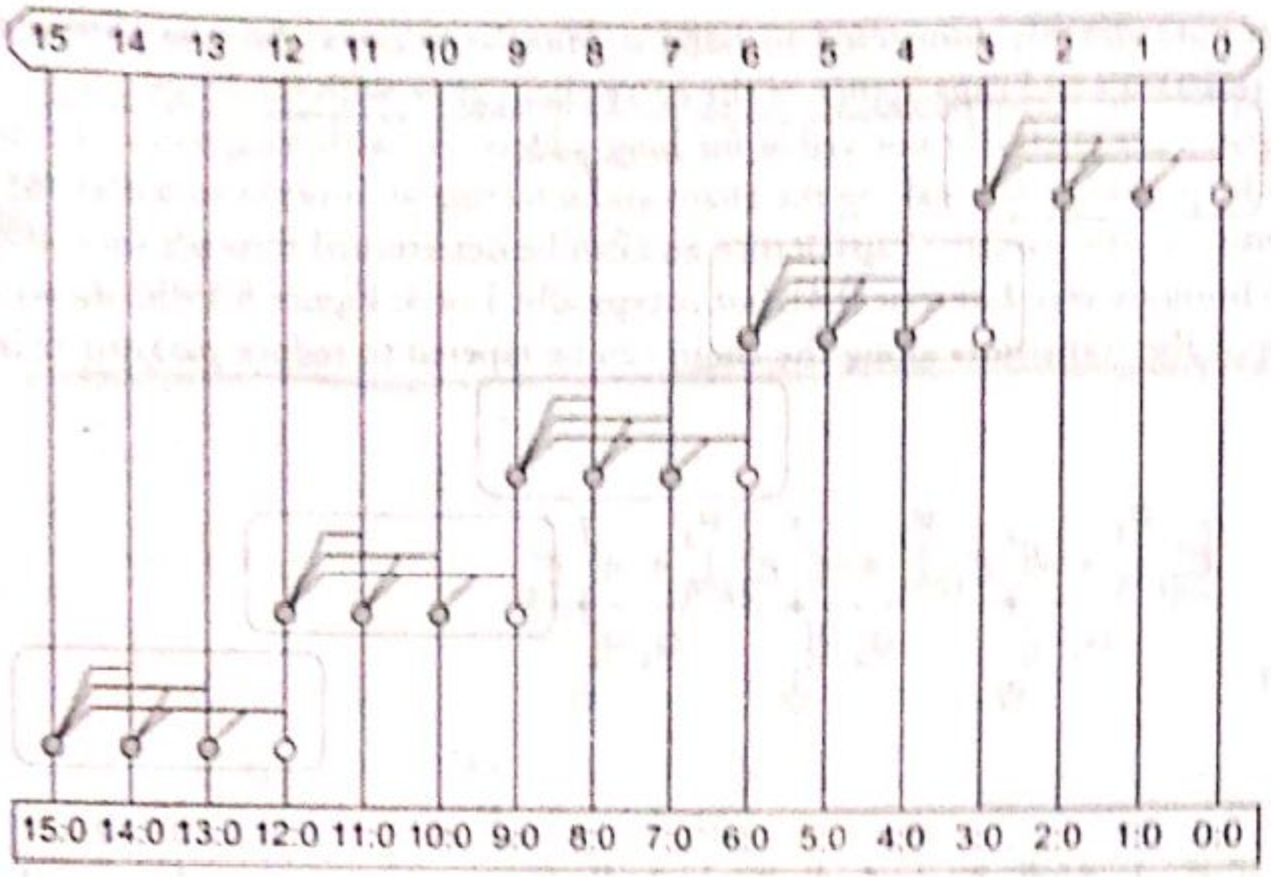


(c)

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

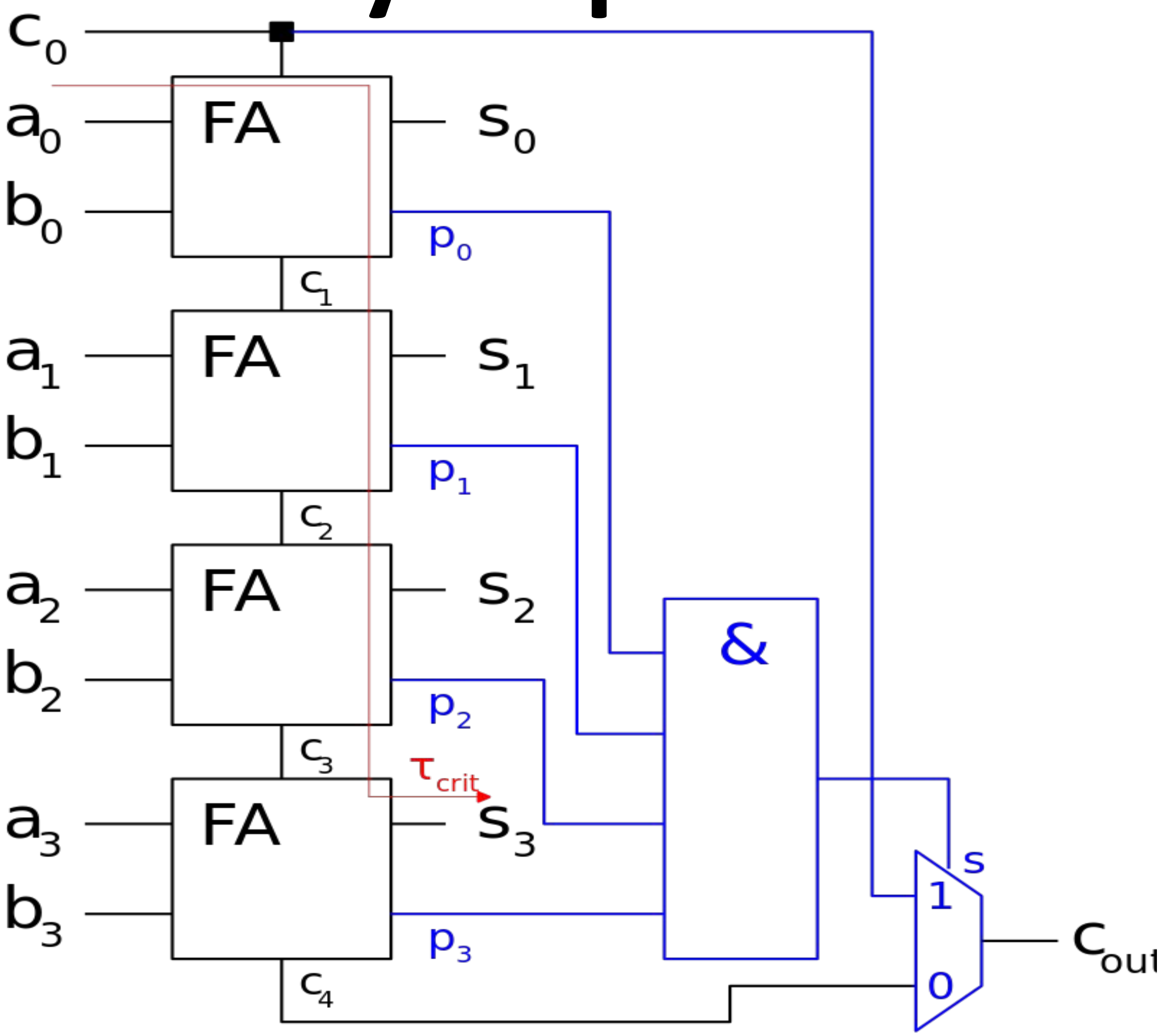


Equivalence of Manchester Carry Chain and Multiple output domino gate

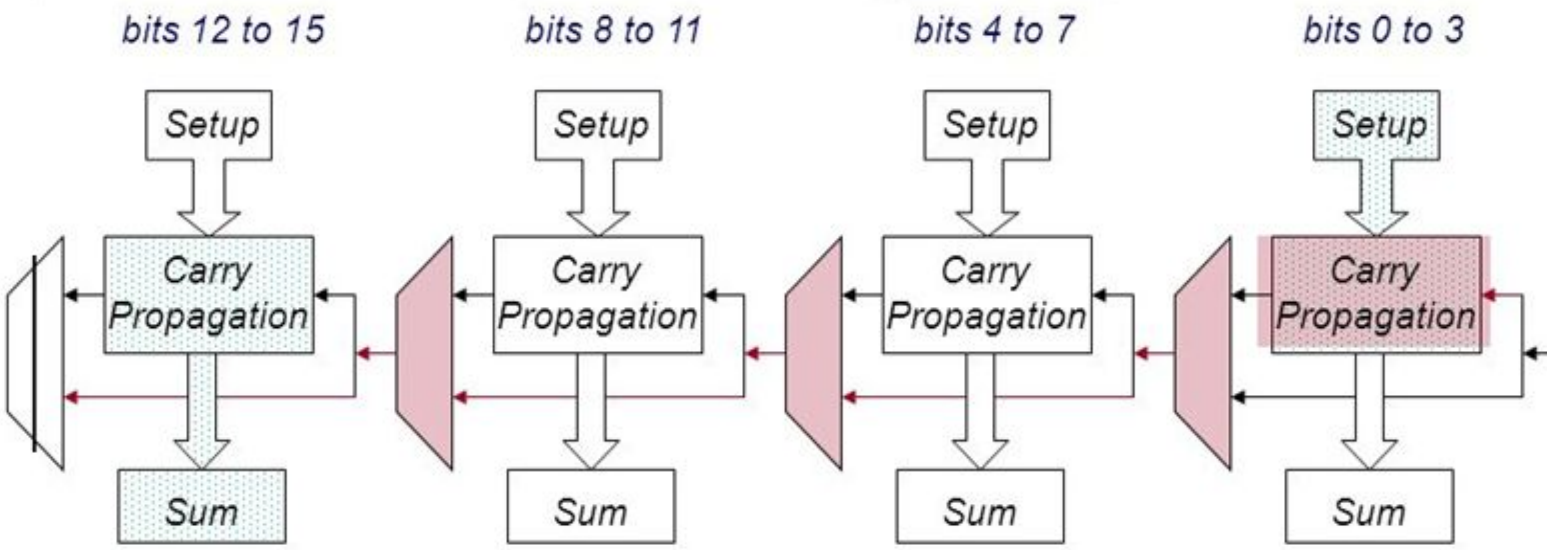


Manchester Carry Chain Adder group PG network

Carry Skip Adder



4-bit Block Carry-Skip Adder



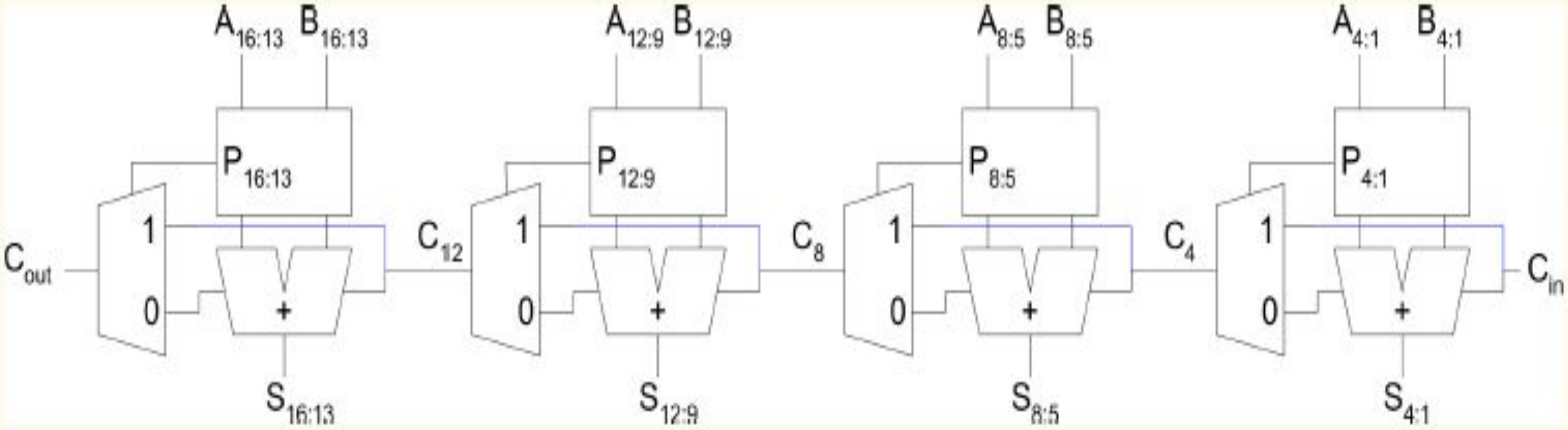
Worst-case delay $\rightarrow$ *carry from bit 0 to bit 15 = carry generated in bit 0, ripples through bits 1, 2, and 3, skips the middle two groups (B is the group size in bits), ripples in the last group from bit 12 to bit 15*

$$T_{add} = t_{setup} + B\, t_{carry} + ((N/B) - 1)\, t_{skip} + (B - 1) t_{carry} + t_{sum}$$

For k n-bit groups $(N = nk)$

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

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



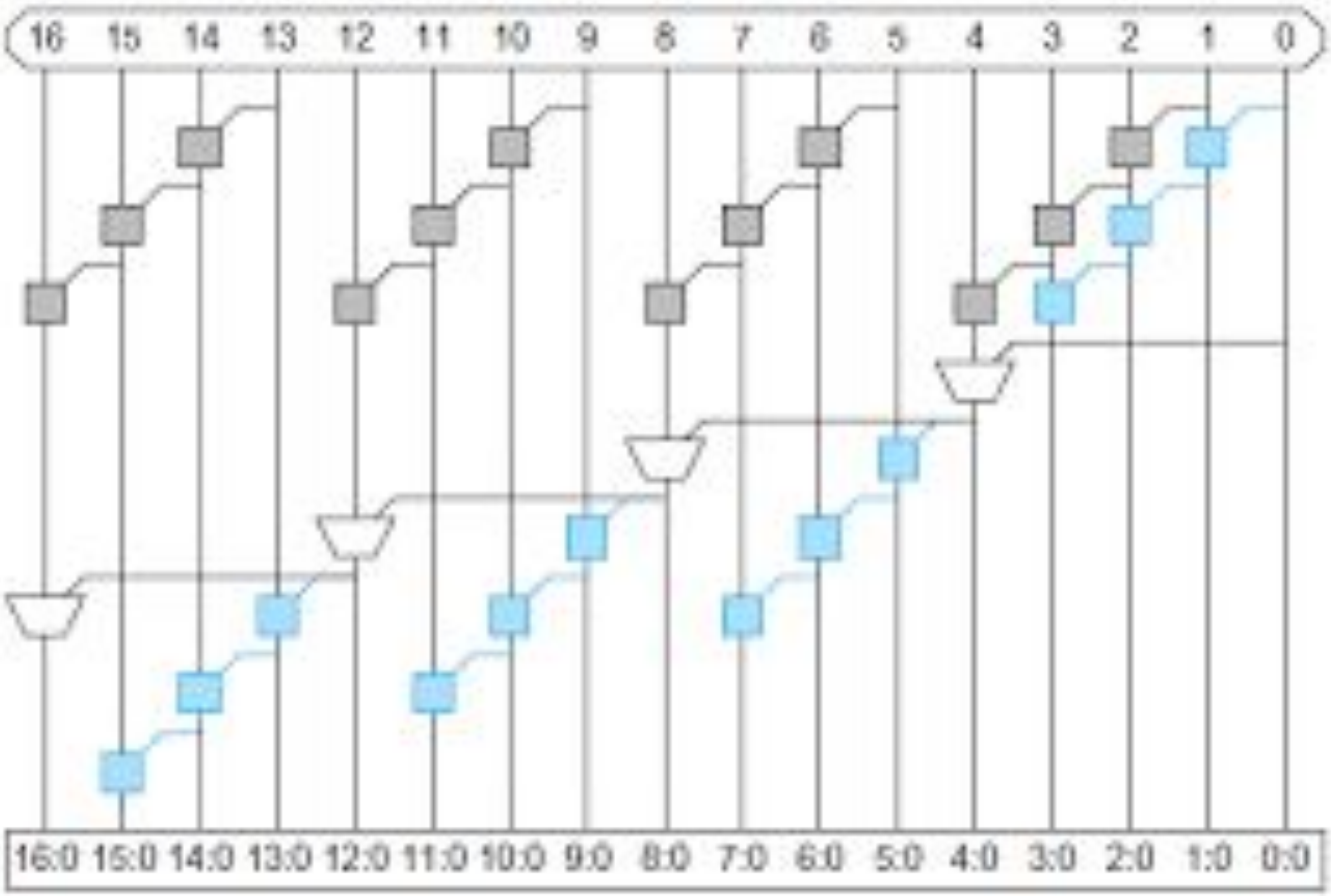
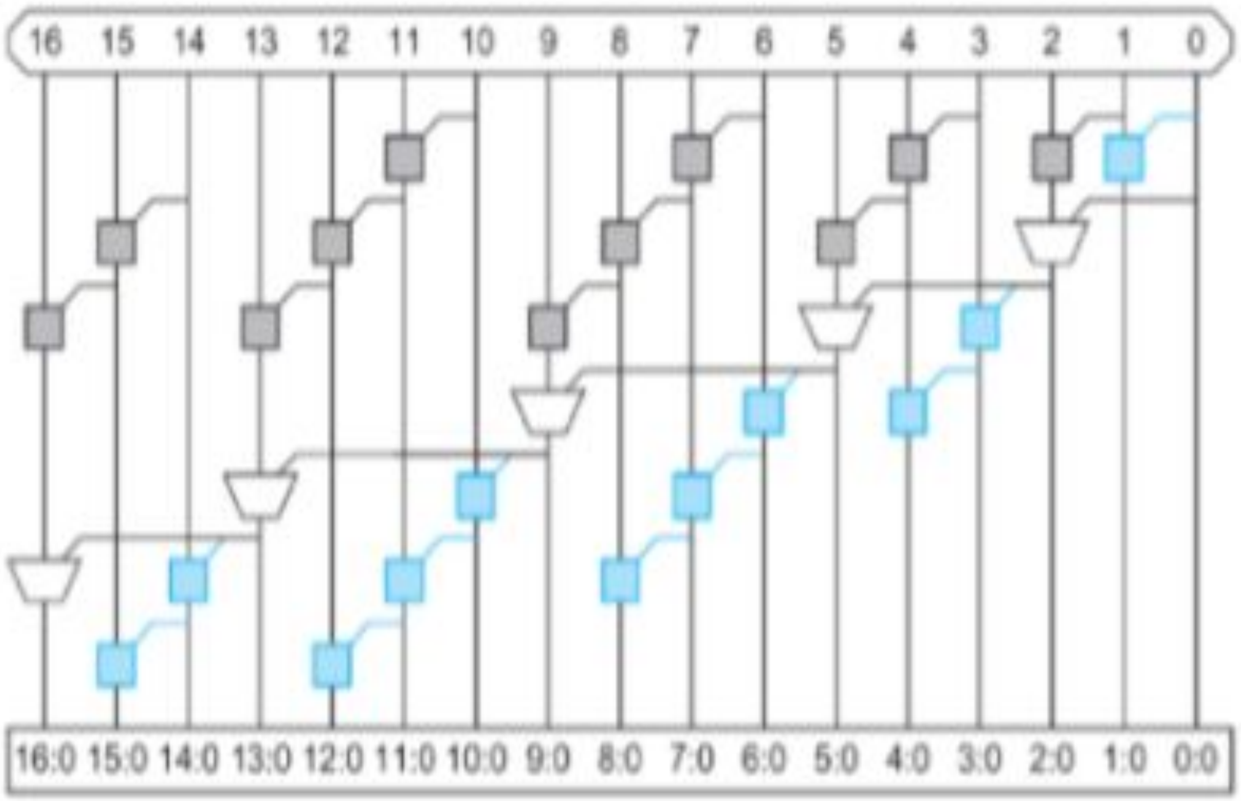
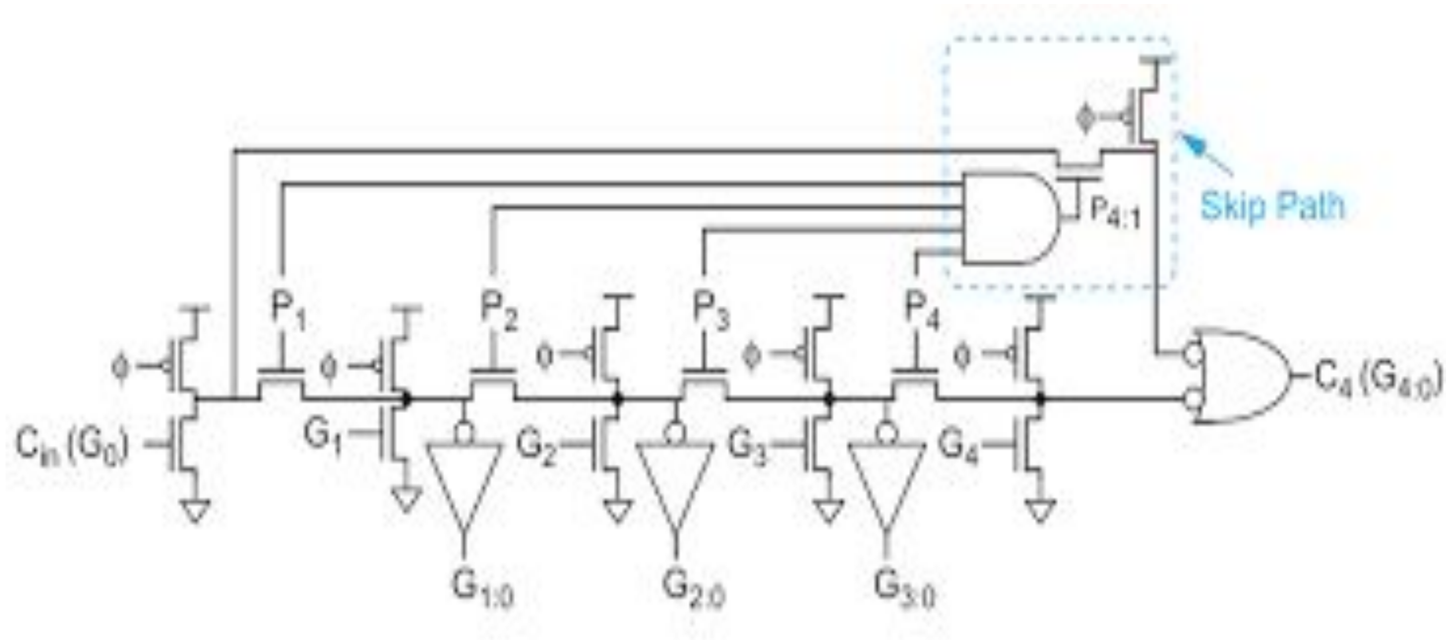


FIGURE 11.18 Carry-skip adder PG network

Variable Group Size Carry Skip Adder

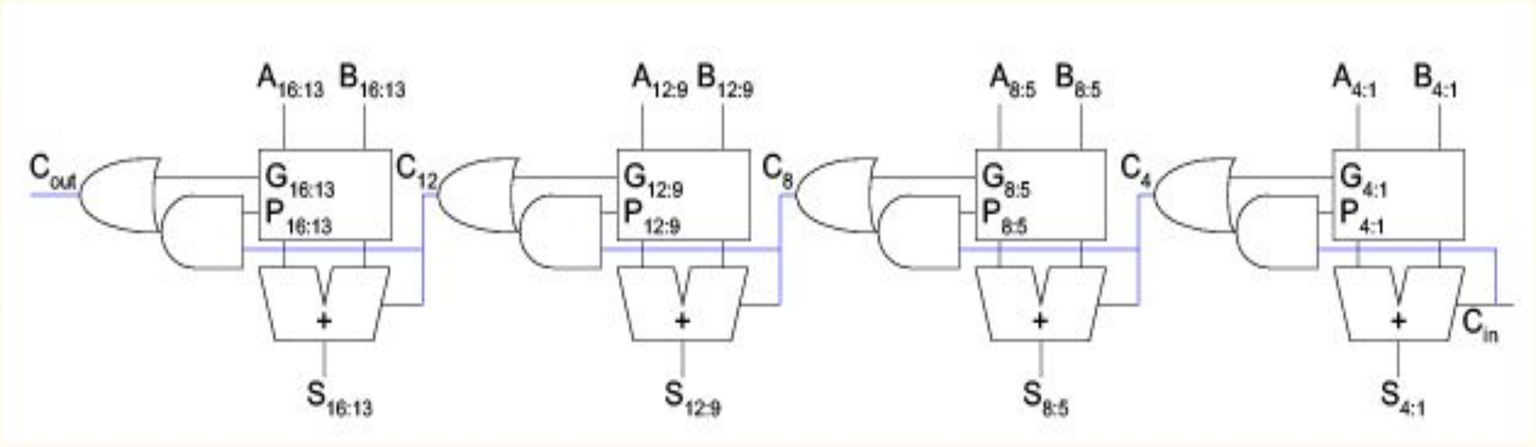


Carry Skip adder Manchester Stage



Carry Lookahead Adder

- Carry-lookahead adder computes $G_{i:0}$ for many bits in parallel
- Uses higher-valency cells with more than two inputs



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

CLA PG Network

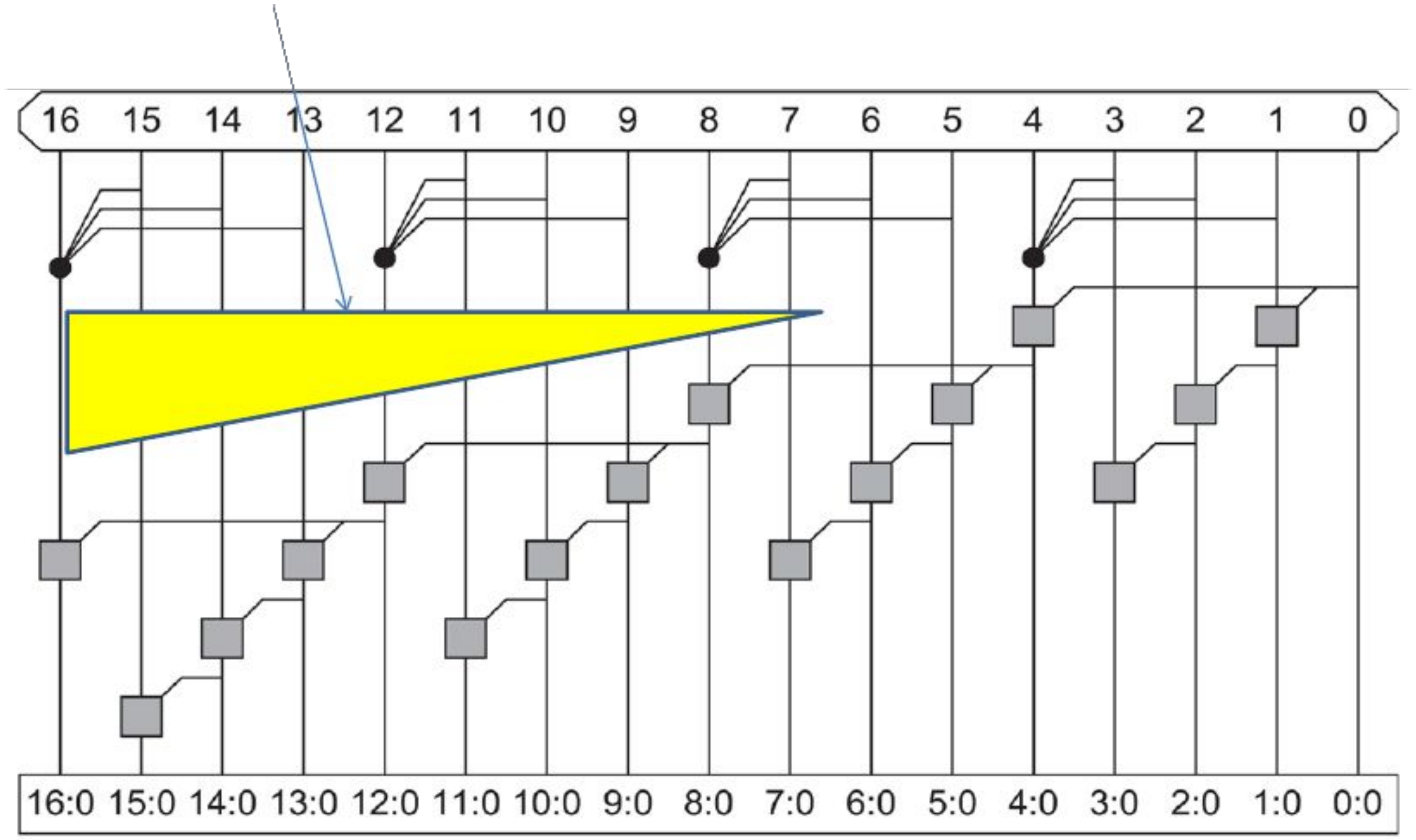
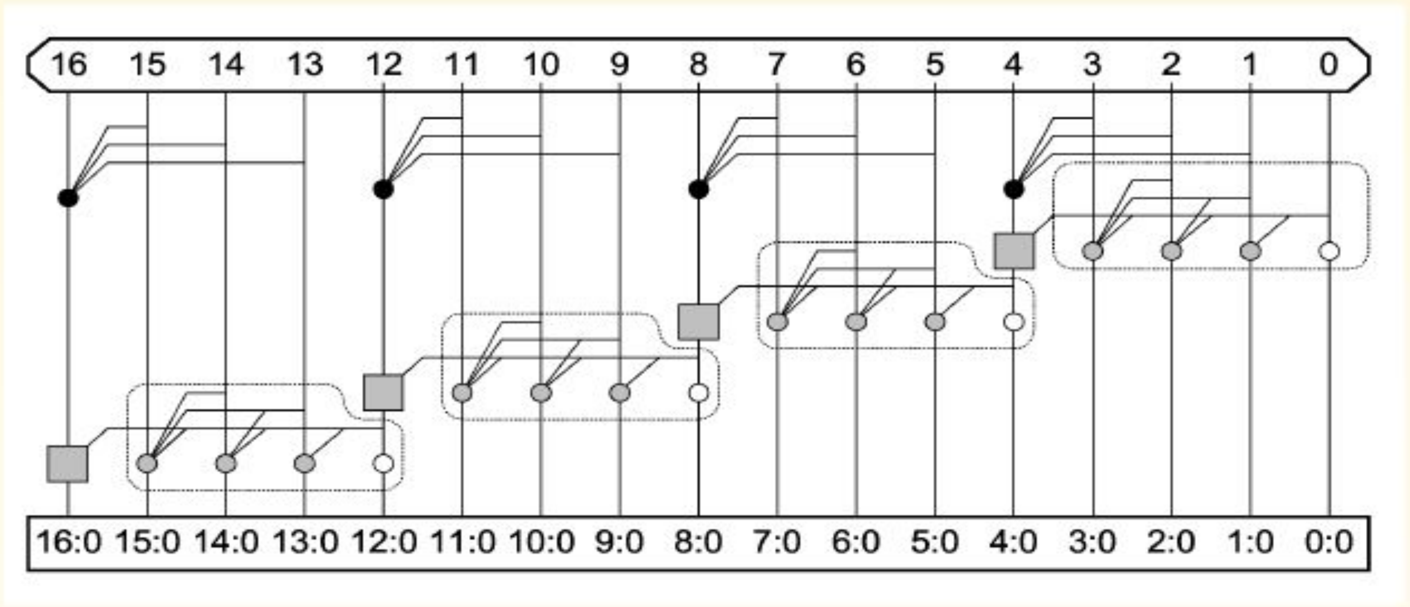
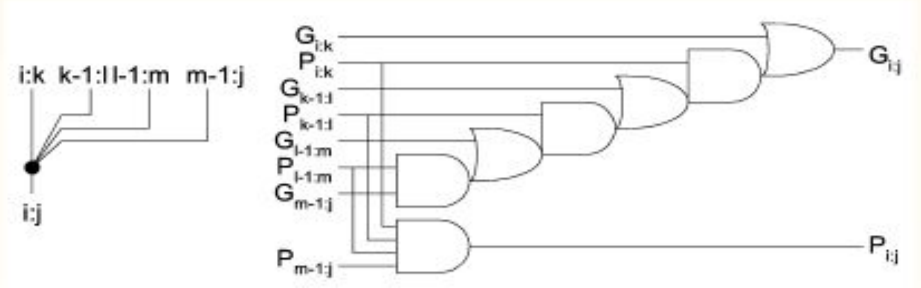


FIG 10.27 Carry-lookahead adder group PG network

Improved CLA PG Network

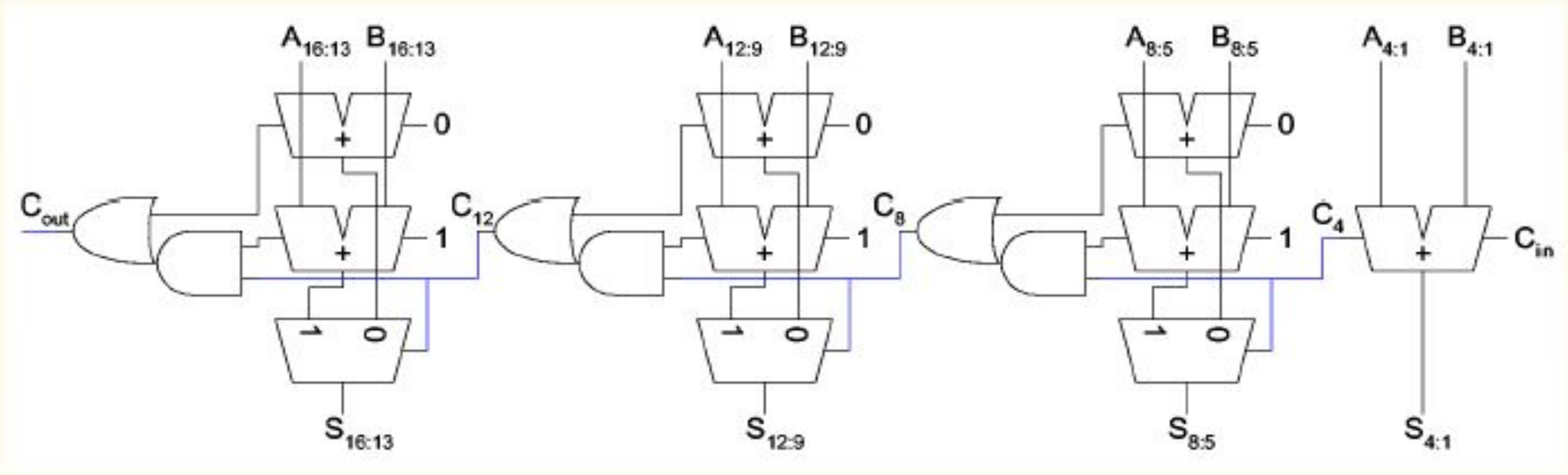


Higher Valency Cells



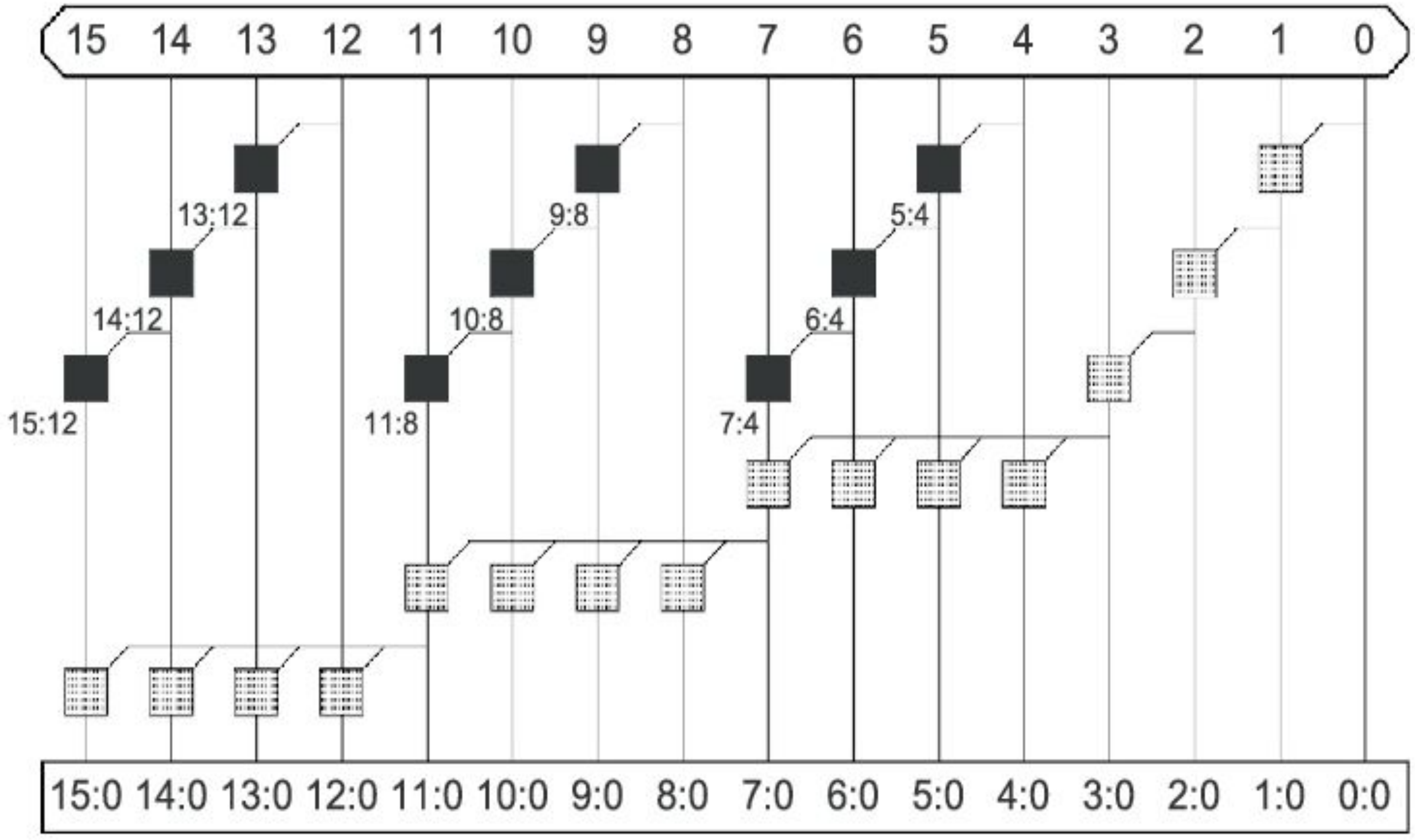
Carry Select Adder

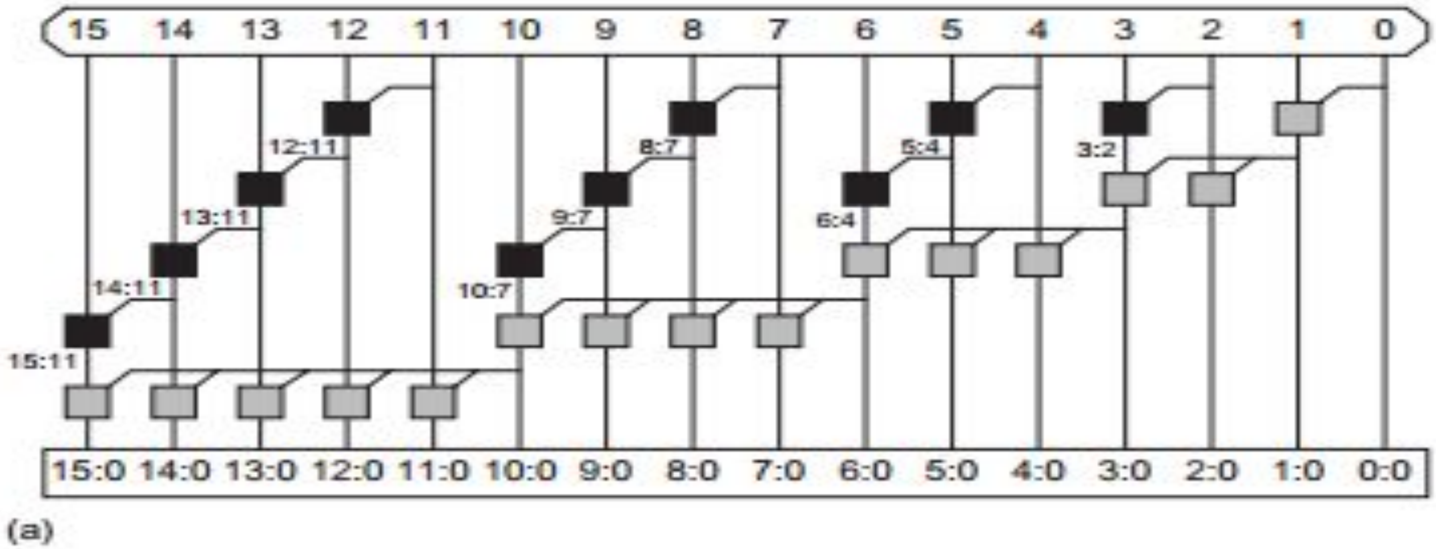
- Trick for critical paths dependent on late input X
  - Precompute two possible outputs for X = 0, 1
  - Select proper output when X arrives
- Carry-select adder precomputes n-bit sums for both possible carries into n-bit group



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

Carry Increment Adder





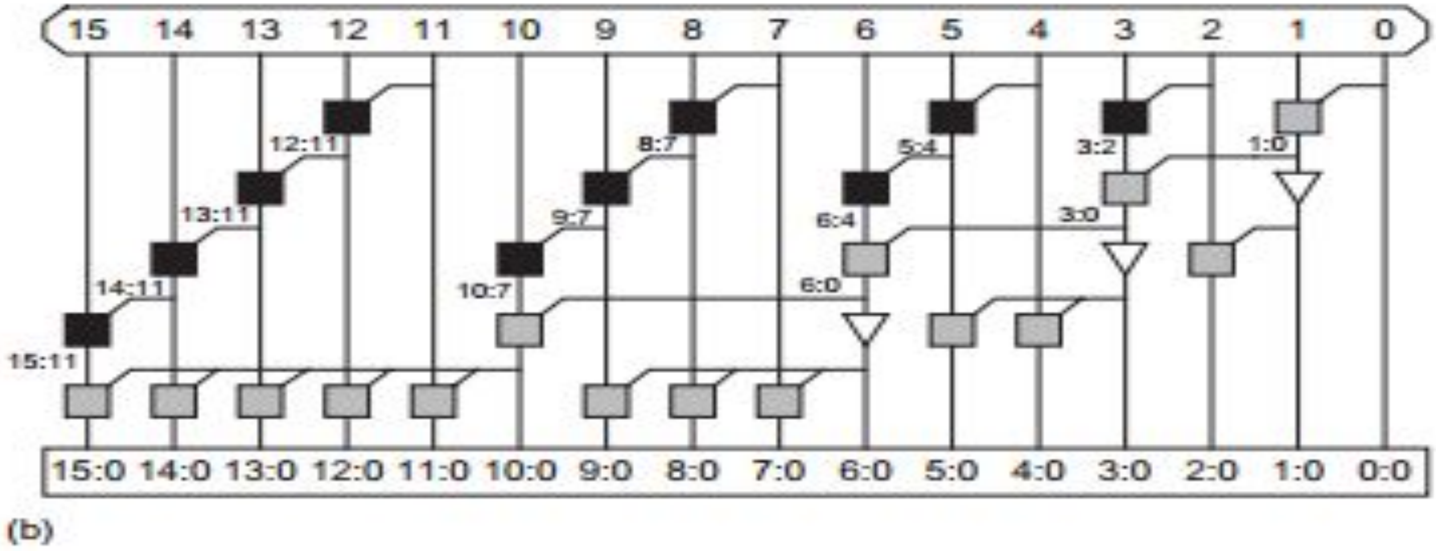
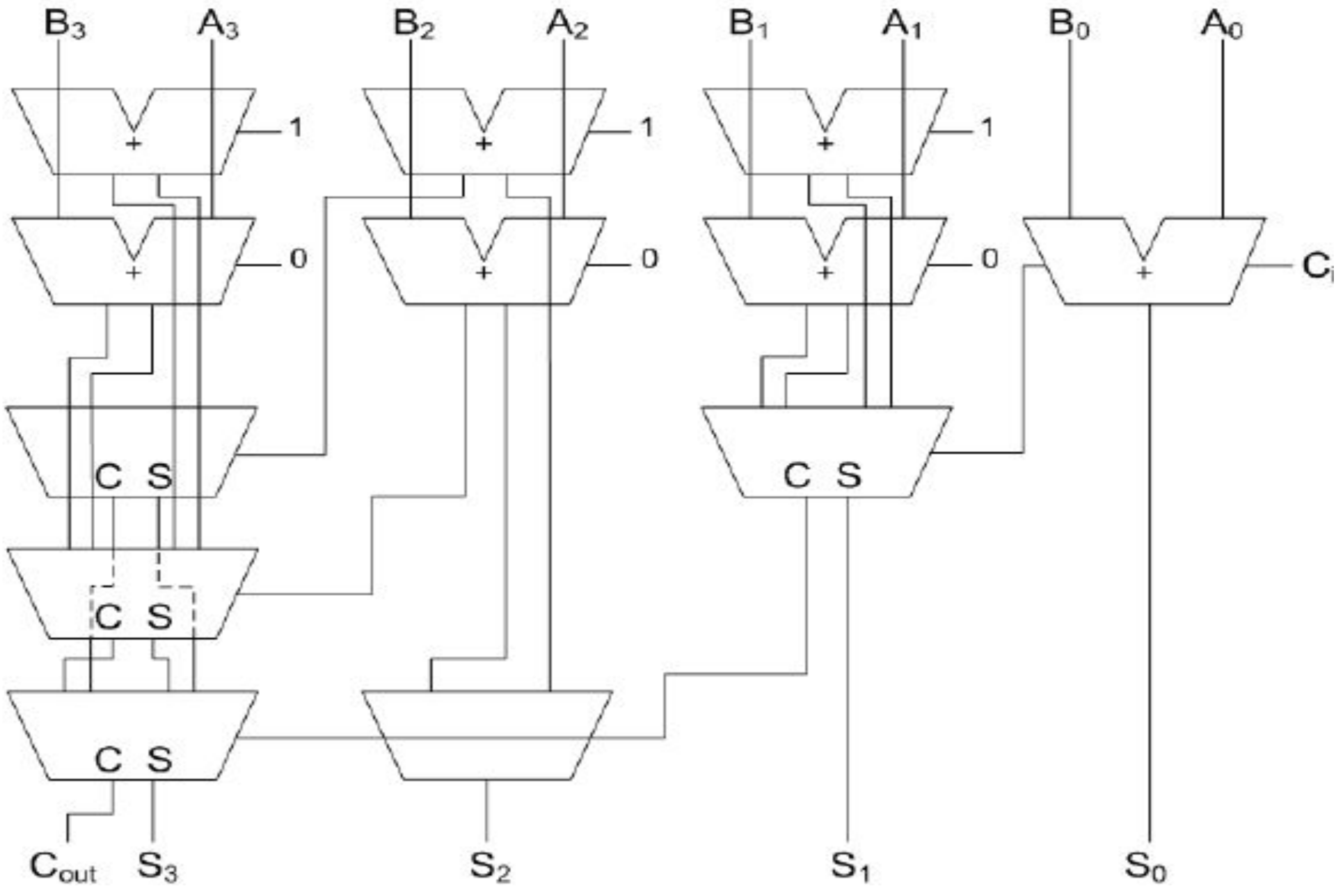
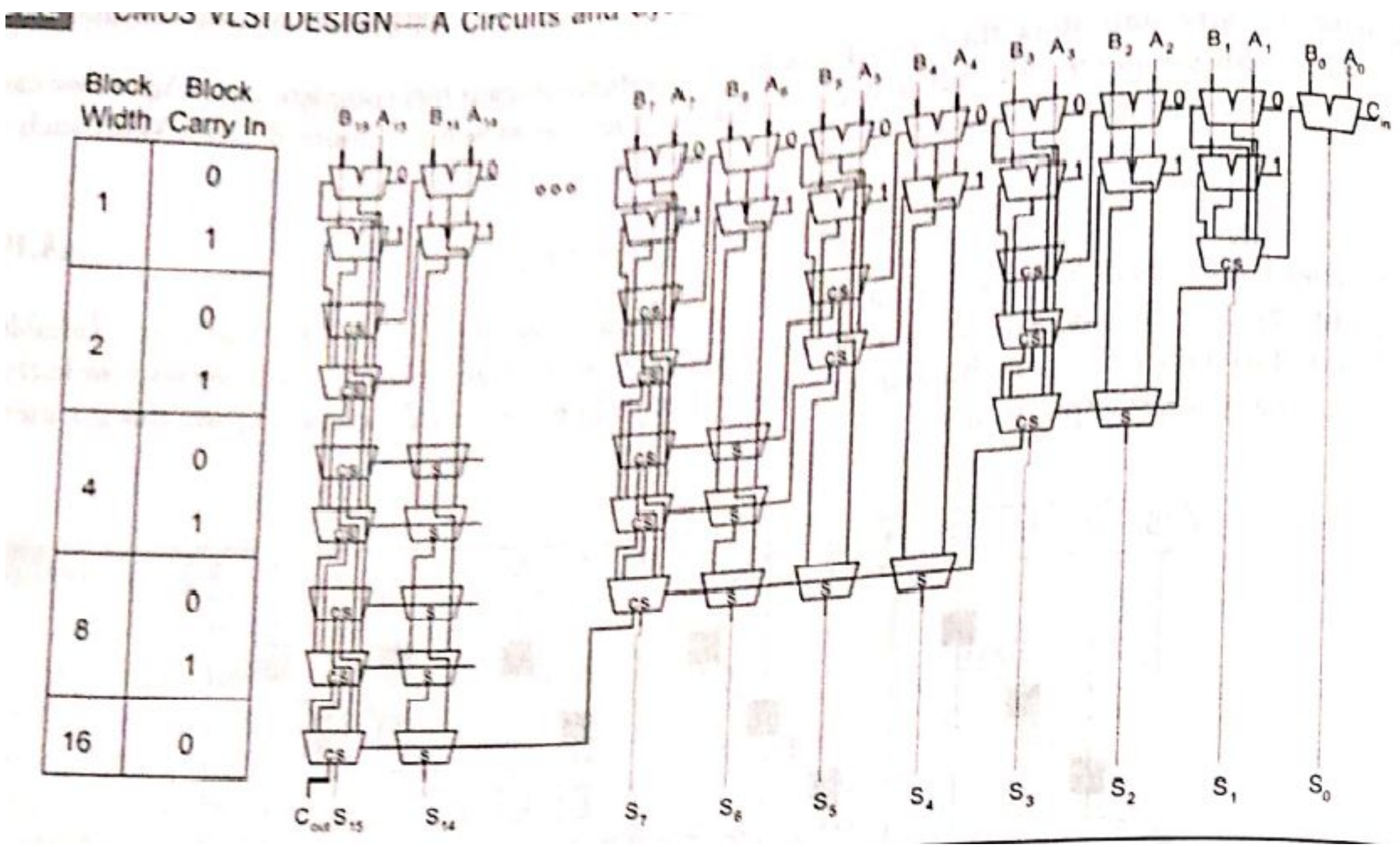


FIGURE 11.26 Variable-length carry-increment adder

4 bit Conditional Sum



16 bit Conditional Sum Adder


CMOS VLSI DESIGN — A Circuits and

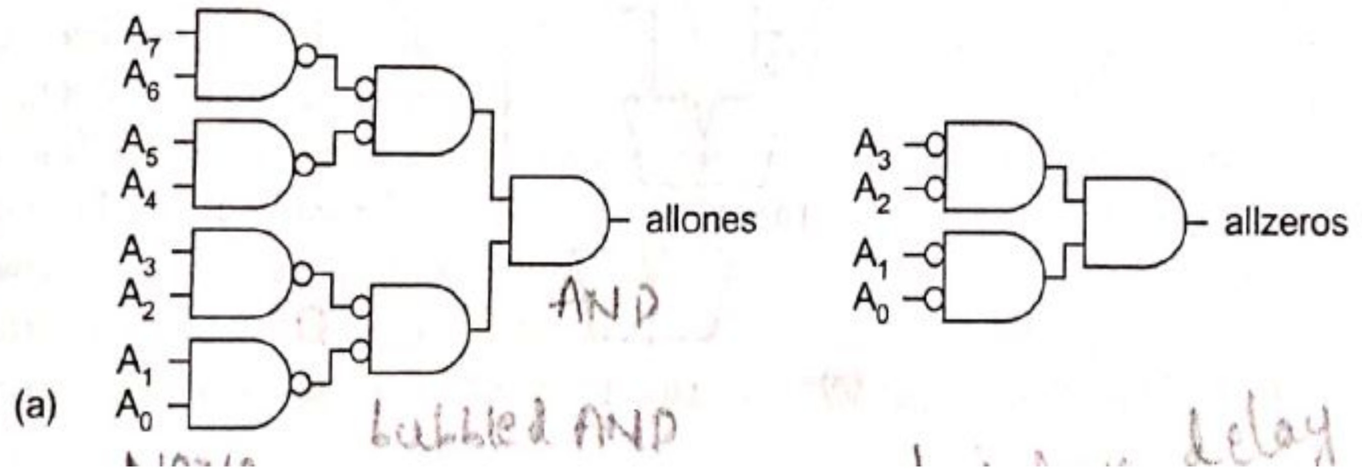
Block Width	Block Carry In
1	0
	1
2	0
	1
4	0
	1
8	0
	1
16	0

16-Bit Conditional Sum Adder Example

		x	0	0	1	0	0	1	1	0	1	1	1	0	1	0	1	0	
		y	0	1	0	0	1	0	1	1	0	1	0	1	1	1	0	1	
Block width	Block carry-in	Block sum and block carry-out	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	c_{in}
1	0	s	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	1	
		c	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0
	1	s	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0		
		c	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1		
2	0	s	0	1	1	0	1	1	0	1	0	0	1	1	0	1	1	1	
		c	0		0		0		1		1		0		1		0		
	1	s	1	0	1	1	0	0	1	0	0	1	0	0	1	0			
		c	0		0		1		1		1		1		1				
4	0	s	0	1	1	0	0	0	0	1	0	0	1	1	0	1	1	1	
		c	0				1				1				1				
	1	s	0	1	1	1	0	0	1	0	0	1	0	0					
		c	0				1				1								
8	0	s	0	1	1	1	0	0	0	1	0	1	0	0	0	1	1	1	
		c	0								1								
	1	s	0	1	1	1	0	0	1	0									
		c	0																
16	0	s	0	1	1	1	0	0	1	0	0	1	0	0	0	1	1	1	
		c	0																
	1	s																	
		c																	

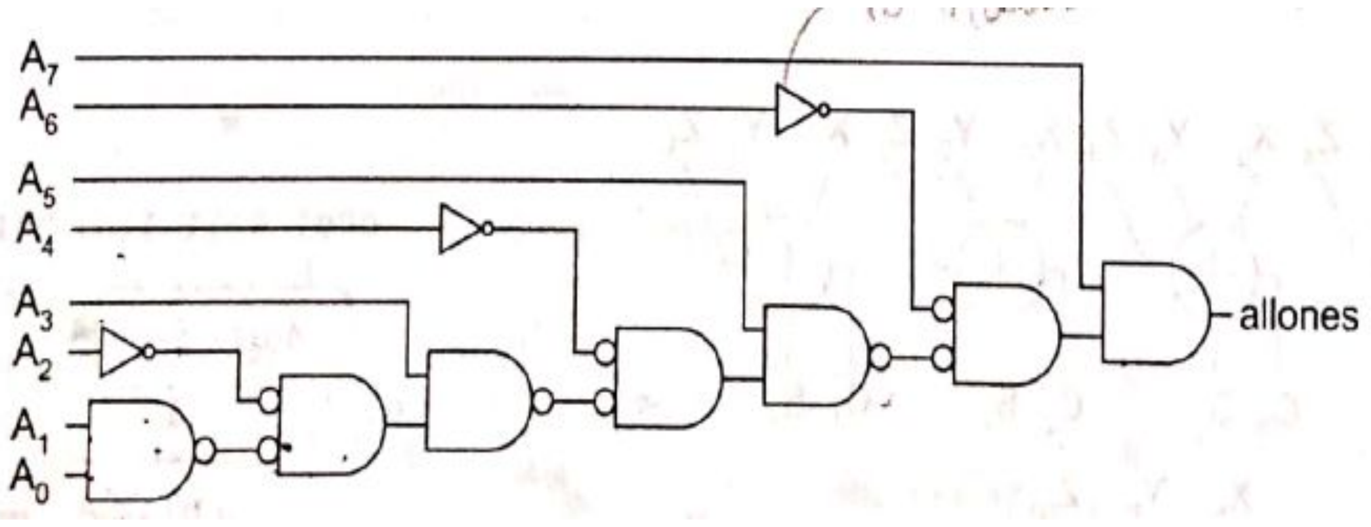
c_{out}

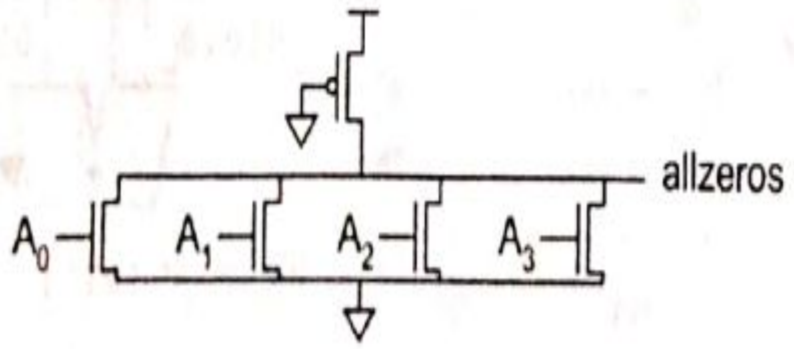
One/Zero Dectectors



$$G_{\text{and}}(N) = \left(\frac{4}{3}\right)^{\log_2 N} = N^{\log_2 \frac{4}{3}} = N^{0.415}$$

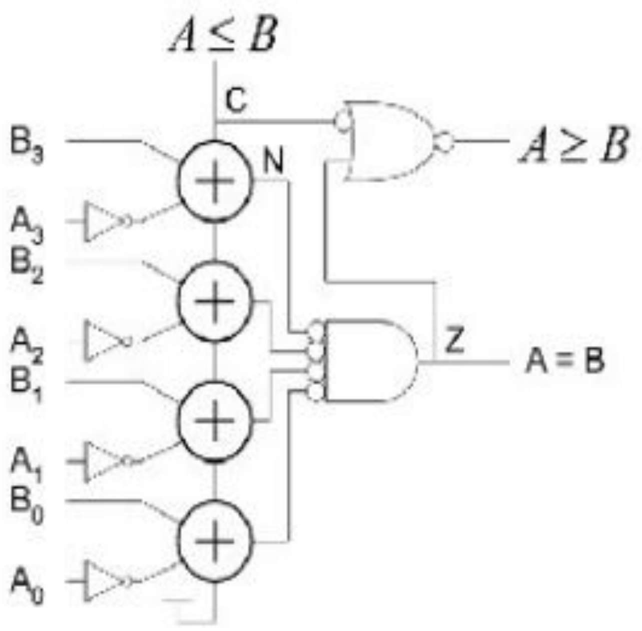
$$D \approx (\log_4 F)\, t_{FO4} = (\log_4 H + 0.415 \log_4 N)\, t_{FO4}$$


A_7
A_6
A_5
A_4
A_3
A_2
A_1
A_0
allones


allzeros
A_0
A_1
A_2
A_3

Magnitude Comparator

- Compute B-A and look at sign
- B-A = B + ~A + 1
- 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 ≠ B)

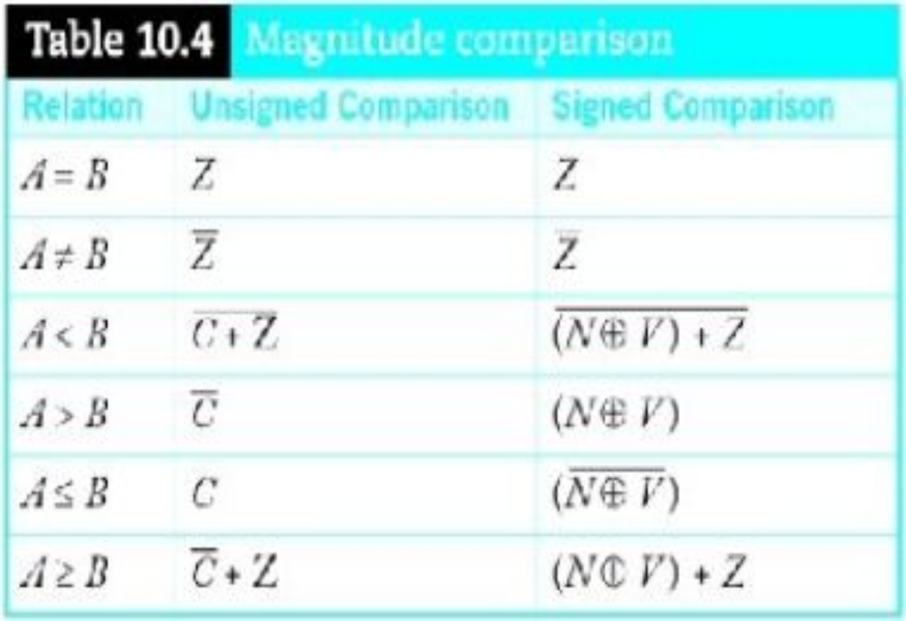
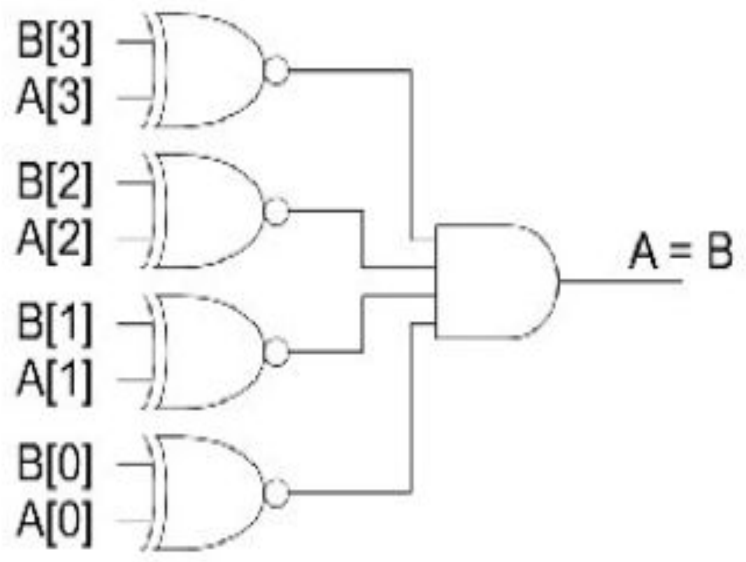


Table 10.4 Magnitude comparison

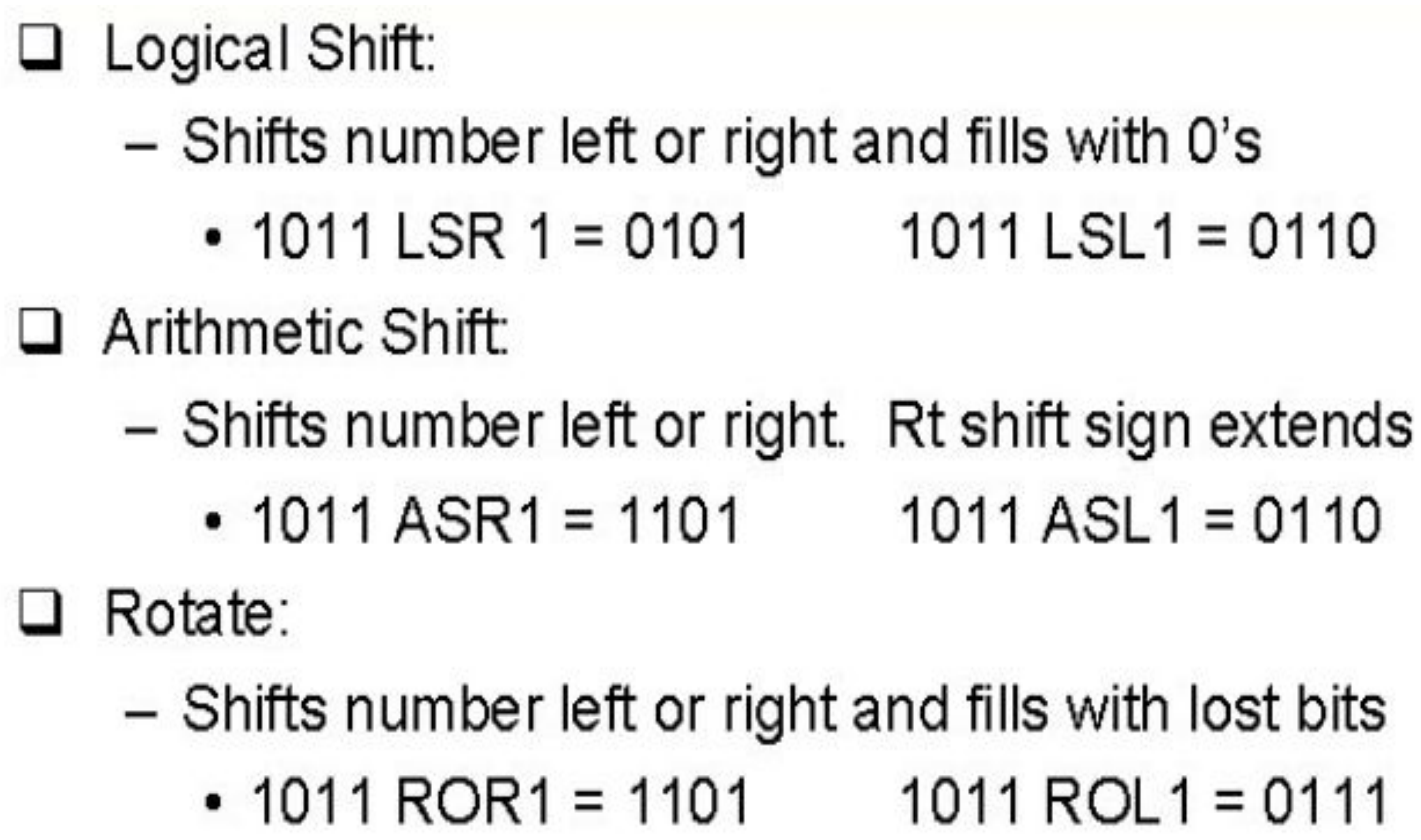
Relation	Unsigned Comparison	Signed Comparison
$A = B$	Z	Z
$A \neq B$	$\overline{Z}$	$\overline{Z}$
$A < B$	$\overline{C + Z}$	$\overline{(N \oplus V) + Z}$
$A > B$	$\overline{C}$	$(N \oplus V)$
$A \leq B$	C	$(\overline{N \oplus V})$
$A \geq B$	$\overline{C} + Z$	$(N \oplus V) + Z$

Equality Comparator

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



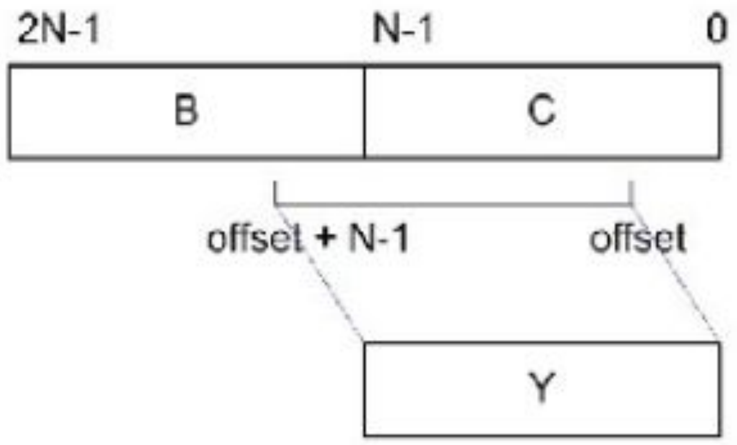
Shifters



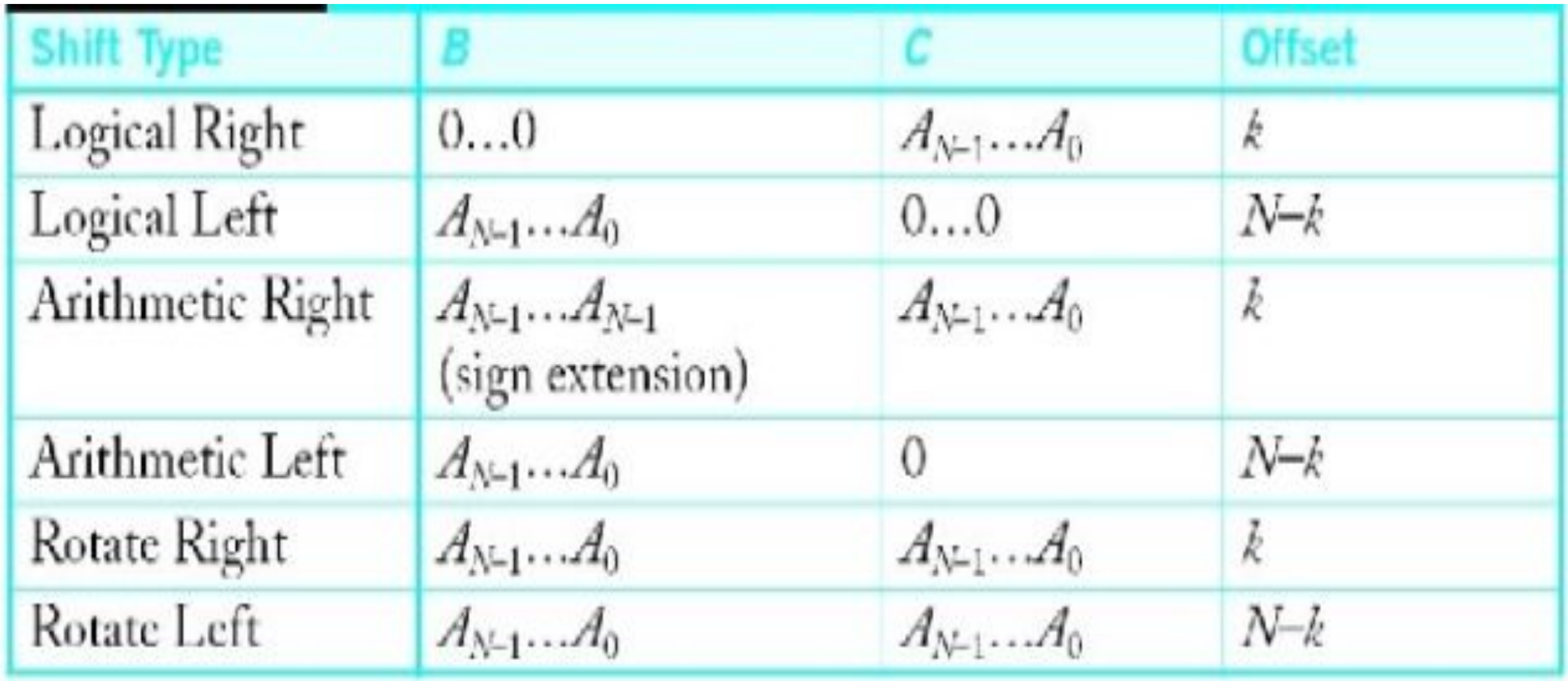
- Logical Shift:
  - Shifts number left or right and fills with 0's
    - 1011 LSR 1 = 0101   1011 LSL1 = 0110
- Arithmetic Shift:
  - Shifts number left or right. Rt shift sign extends
    - 1011 ASR1 = 1101   1011 ASL1 = 0110
- Rotate:
  - Shifts number left or right and fills with lost bits
    - 1011 ROR1 = 1101   1011 ROL1 = 0111

Funnel Shifters

- A funnel shifter can do all six types of shifts
- Selects N-bit field Y from 2N-bit input
  - Shift by k bits ($0 \le k < N$)



Funnel Shift Operation

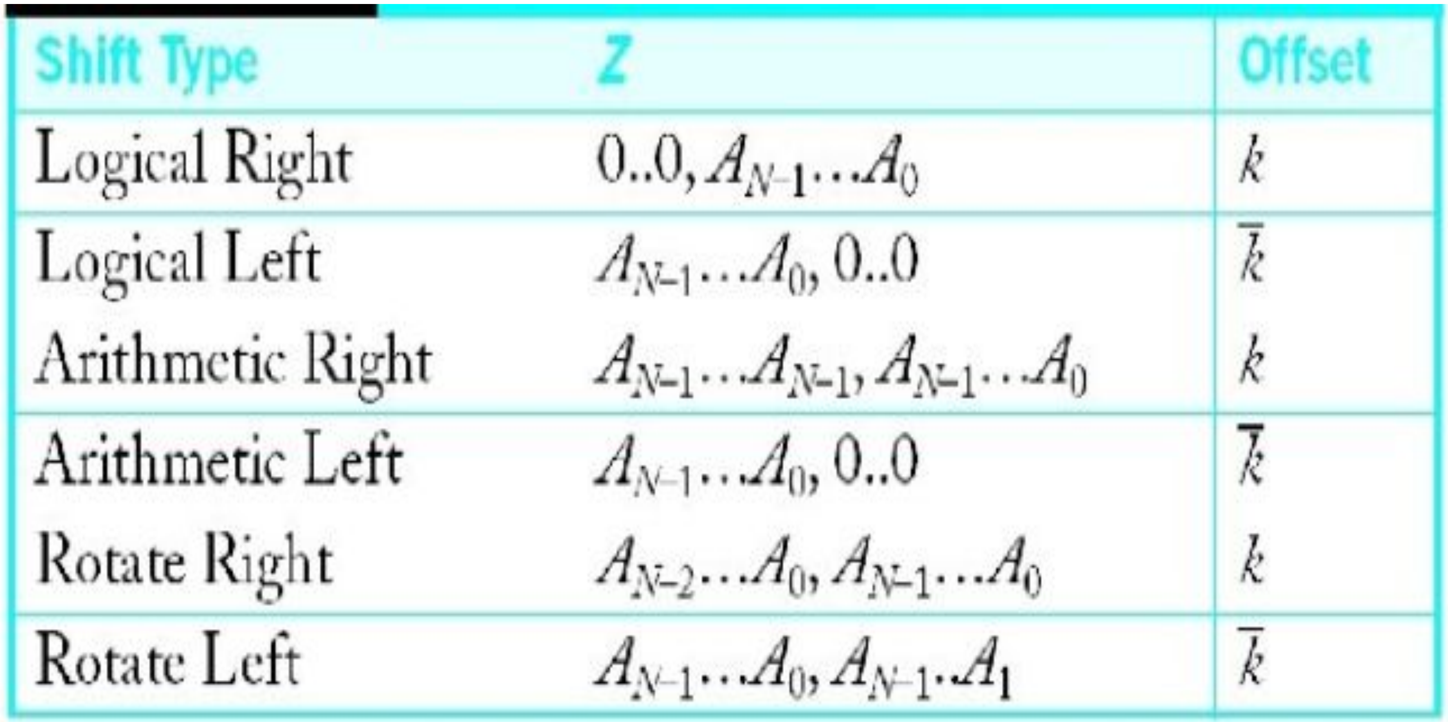


Shift Type	B	C	Offset
Logical Right	0…0	$A_{N-1}\ldots A_0$	k
Logical Left	$A_{N-1}\ldots A_0$	0…0	$N-k$
Arithmetic Right	$A_{N-1}\ldots A_{N-1}$ (sign extension)	$A_{N-1}\ldots A_0$	k
Arithmetic Left	$A_{N-1}\ldots A_0$	0	$N-k$
Rotate Right	$A_{N-1}\ldots A_0$	$A_{N-1}\ldots A_0$	k
Rotate Left	$A_{N-1}\ldots A_0$	$A_{N-1}\ldots A_0$	$N-k$

- Computing N-k requires an adder

Simplified Funnel Shifter

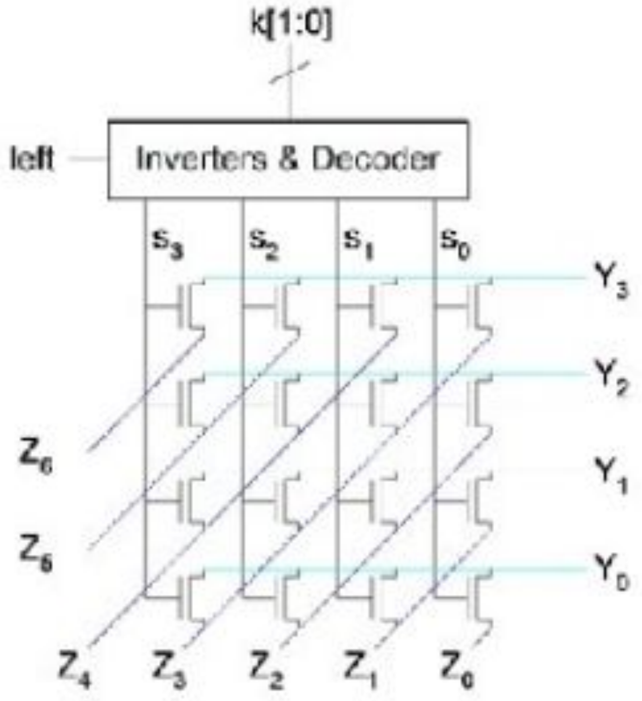
- Optimize down to 2N-1 bit input

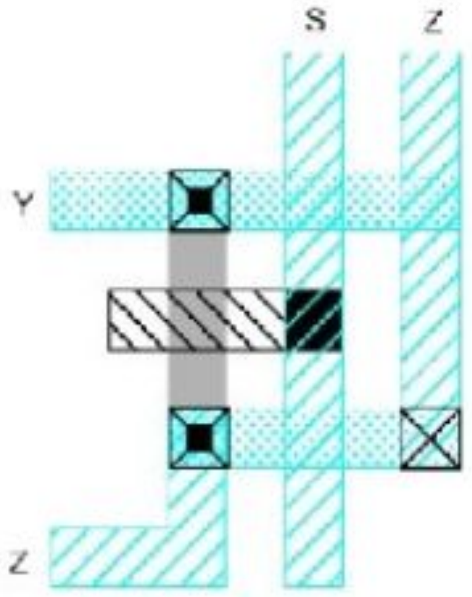


Shift Type	Z	Offset
Logical Right	$0..0, A_{N-1} \ldots A_0$	k
Logical Left	$A_{N-1} \ldots A_0, 0..0$	$\overline{k}$
Arithmetic Right	$A_{N-1} \ldots A_{N-1}, A_{N-1} \ldots A_0$	k
Arithmetic Left	$A_{N-1} \ldots A_0, 0..0$	$\overline{k}$
Rotate Right	$A_{N-2} \ldots A_0, A_{N-1} \ldots A_0$	k
Rotate Left	$A_{N-1} \ldots A_0, A_{N-1} .. A_1$	$\overline{k}$

Funnel Shifter Design-1

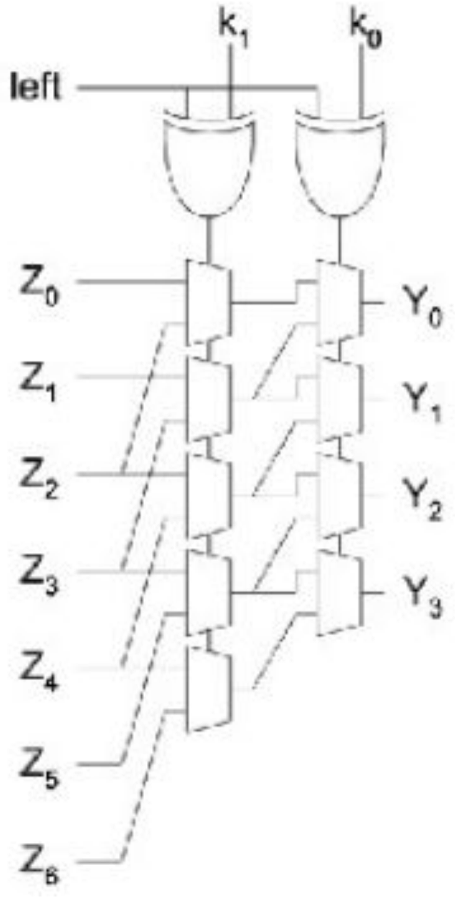
- N N-input multiplexers
  - Use 1-of-N hot select signals for shift amount
  - nMOS pass transistor design (V_t drops!)



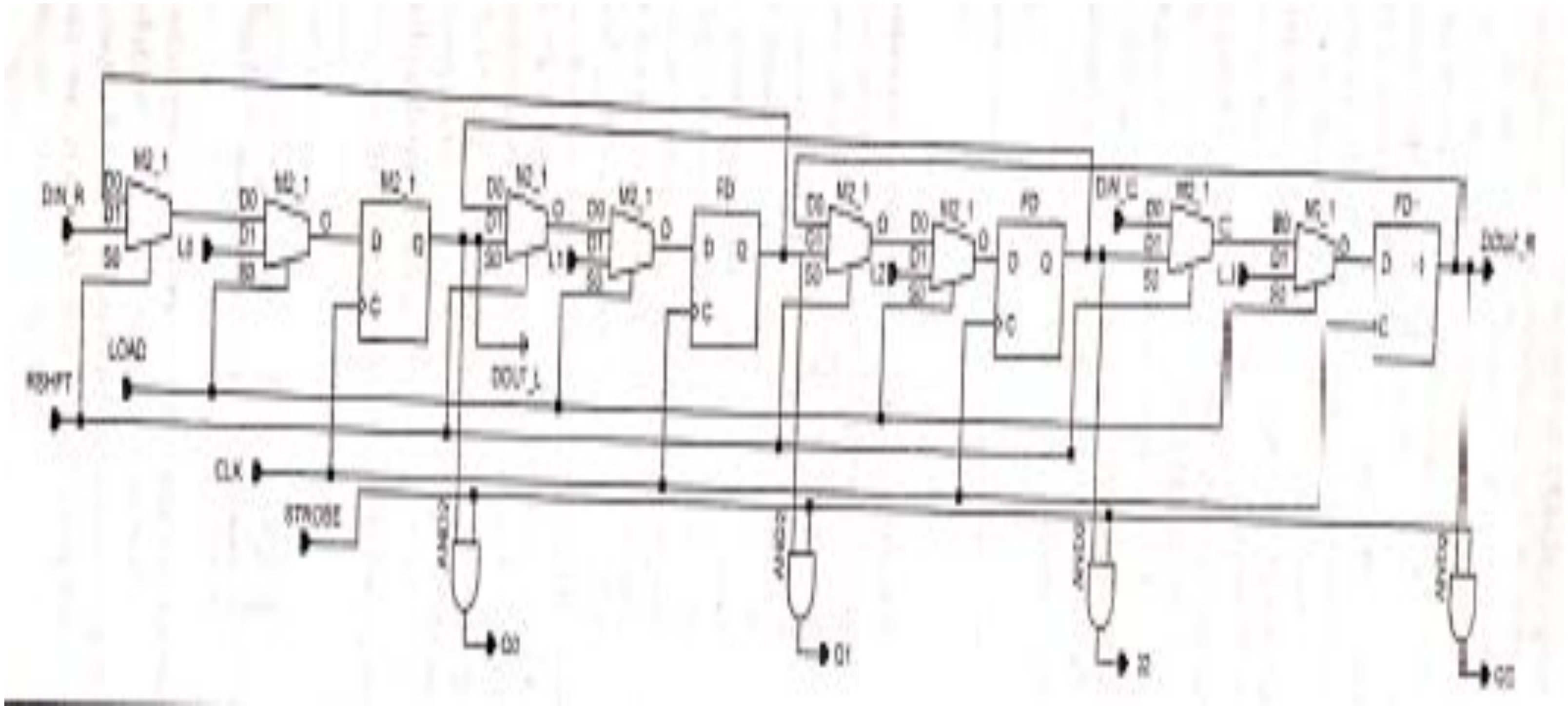


Funnel Shifter Design-2

- Log N stages of 2-input muxes
  - No select decoding needed



Universal Shift Register



Multiplication

- Example:

```
1100 : 12₁₀
0101 : 5₁₀
────
```

Multiplication

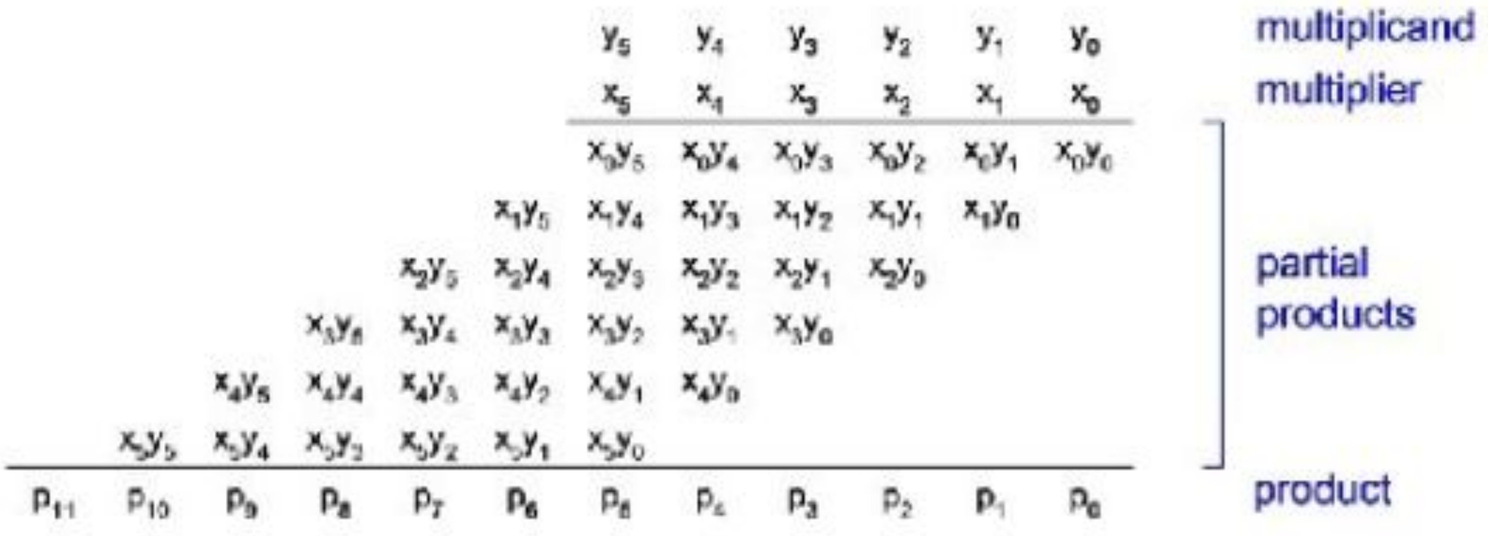
- Example:

```
     1100  :  12₁₀        multiplicand
     0101  :  5₁₀         multiplier
     ----
     1100              ┐
    0000               │  partial
   1100                │  products
  0000                 ┘
 --------
 00111100  :  60₁₀        product
```

- M x N-bit multiplication
  - Produce N M-bit partial products
  - Sum these to produce M+N-bit product

General Form

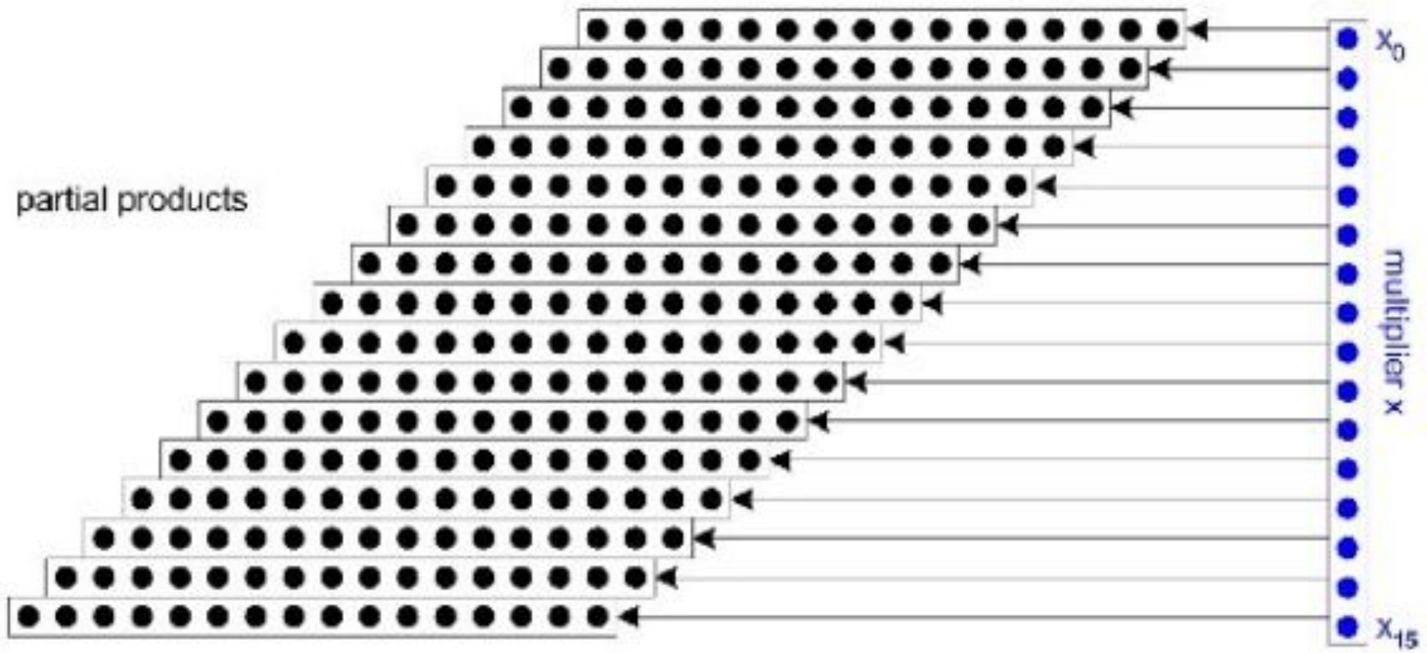
- Multiplicand: $Y = (y_{M-1}, y_{M-2}, \ldots, y_1, y_0)$
- Multiplier: $X = (x_{N-1}, x_{N-2}, \ldots, x_1, x_0)$
- Product: $$P = \left(\sum_{j=0}^{M-1} y_j 2^j\right)\left(\sum_{i=0}^{N-1} x_i 2^i\right) = \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} x_i y_j 2^{i+j}$$



						y_5	y_4	y_3	y_2	y_1	y_0	multiplicand
						x_5	x_4	x_3	x_2	x_1	x_0	multiplier
						x_0y_5	x_0y_4	x_0y_3	x_0y_2	x_0y_1	x_0y_0	
					x_1y_5	x_1y_4	x_1y_3	x_1y_2	x_1y_1	x_1y_0		
				x_2y_5	x_2y_4	x_2y_3	x_2y_2	x_2y_1	x_2y_0			partial products
			x_3y_5	x_3y_4	x_3y_3	x_3y_2	x_3y_1	x_3y_0				
		x_4y_5	x_4y_4	x_4y_3	x_4y_2	x_4y_1	x_4y_0					
	x_5y_5	x_5y_4	x_5y_3	x_5y_2	x_5y_1	x_5y_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	product

Dot Diagram

- Each dot represents a bit



Efficient Parallel Approaches

- Unsigned Multiplication
- 2’s Complement
- Wallace trees
- Hybrid array/tree structures
- Serial Multiplications