EECS 1010 01
Combinational Logic

黃稚存

國立清華大學
資訊工程學系

Lecture 04-1

Outline

- Introduction
  - Combinational Circuits
  - Analysis Procedure
  - Design Procedure
- Binary Adder-Subtractor
- Decimal Adder
- Binary Multiplier
- Magnitude Comparator
- Decoder and Encoder
- Multiplexer
- Three-State Gates

Objectives

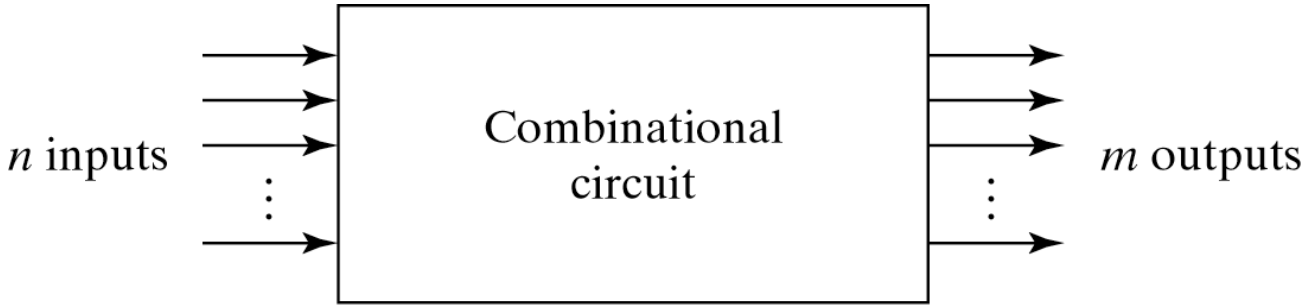
- Know how to analyze a combinational logic circuit, given its logic diagram
- Understand the half-adder and full-adder
- Understand the overflow and underflow
- Understand the implementation of a binary adder, BCD adder, and binary multiplier
- Understand fundamental combinational logic circuits: decoder, encoder, priority encoder, multiplexer, and three-state gate

Introduction

- Combinational Circuits
- Analysis Procedure
- Design Procedure

Logic Circuits for the Digital System

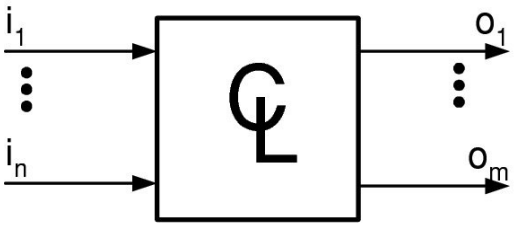
- Combinational circuits
  - Logic circuits whose outputs at any time are determined *directly* and *only* from the present input combination
    - Memoryless

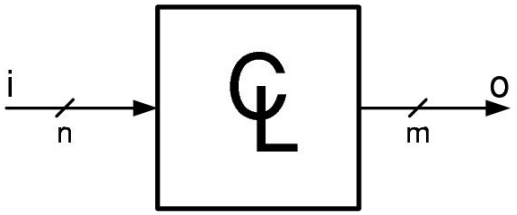


- Sequential circuits (Chapter 5)
  - Circuits that employ memory elements + (combinational) logic gates
  - Outputs are determined from the present input combination as well as the state of the memory cells

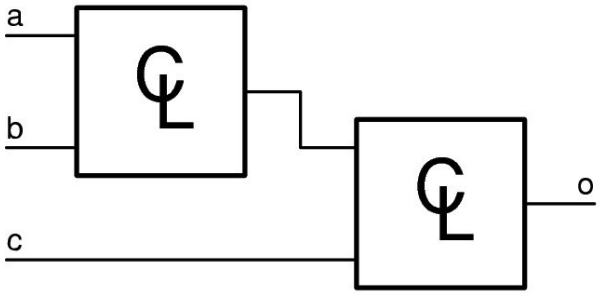
Combinational Logic Circuits

- Memoryless: o=f(i)
  - Used for control, arithmetic, and data steering

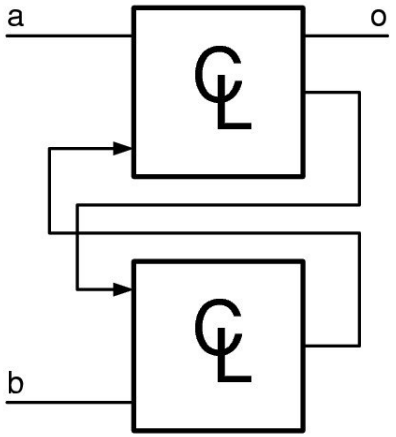




- Combinational logic circuits are closed under acyclic composition
- Cyclic composition of two combinational logic circuits
  - The feedback variable can remember the history of the circuits
  - Sequential logic circuit



Acyclic



Cyclic

Analysis Procedure

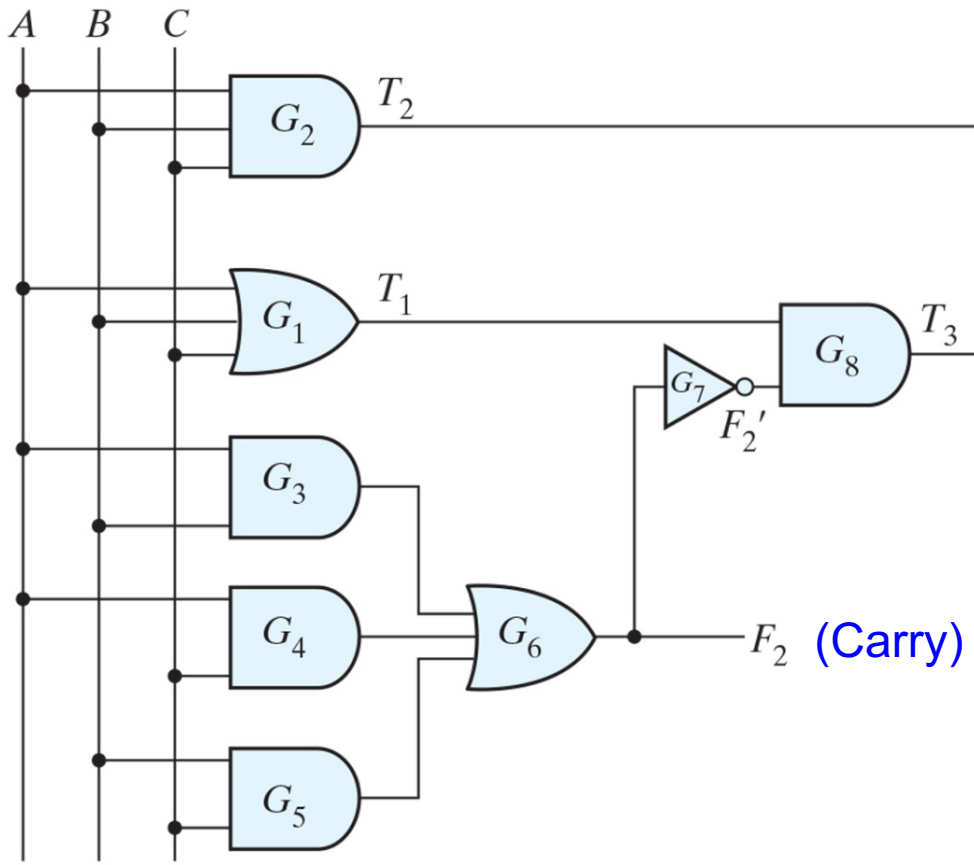
- Analysis for an available logic diagram
  - Make sure the given circuit is combinational
    - No *feedback path* or *memory element*
  - Derive the corresponding *Boolean functions*
  - Derive the corresponding *truth table*
  - Verify and analyze the design
    - Logic simulation (waveforms)
  - Explain the function

Derivation of Boolean Functions (1/2)

- Label all gate outputs that are functions of the input variables only. Determine the functions.
- Label all gate outputs that are functions of the input variables and previously labeled gate outputs. And find the functions.
- Repeat previous step until all the primary outputs are obtained.

Derivation of Boolean Functions (2/2)

- Full adder example:



- T_2=ABC
- T_1=A+B+C
- T_3=F_2'T_1
- F_2=AB+AC+BC (Carry)
- F_1=T_3+T_2 (Sum)
  =F_2'T_1+ABC
  =(AB+AC+BC)'(A+B+C)+ABC
  =A'BC'+A'B'C+AB'C'+ABC

Derivation of Truth Table (1/2)

- For *n* input variables
  - List all the 2^n input combinations from 0 to $2^n - 1$
  - Partition the circuit into small single-output blocks and label the output of each block
  - Obtain the truth table of the blocks depending on the input variables only
  - Proceed to obtain the truth tables for other blocks that depend on previously defined truth tables

Derivation of Truth Table (2/2)

- Full Adder Example

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

Design Procedure

① Specification
- From the specifications, determine the inputs, outputs, and their symbols

② Formulation
- Derive the truth tables (functions) from the relationship between the inputs and outputs

③ Optimization
- Derive the simplified Boolean functions for each output function

④ Technology mapping
- Derive the logic diagram based on the implementation technology

⑤ Verification
- Verify the design

Code Conversion Example (1/3)

- BCD-to-excess-3 code converter

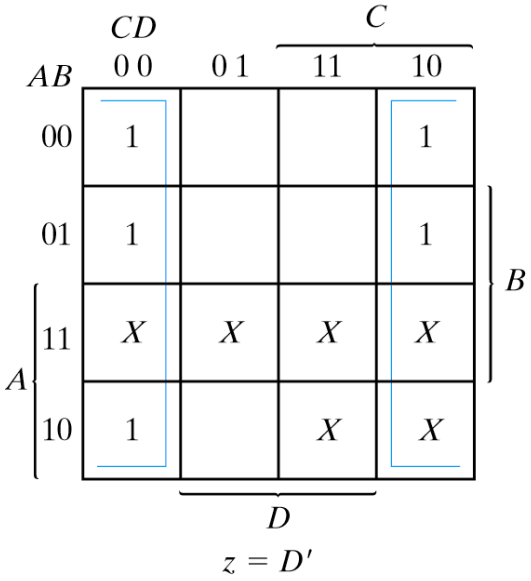
(1) Define the spec and IOs
(2) Derive truth table

① Spec
- input (ABCD), output (wxyz) (MSB to LSB)
- ABCD: 0000~1001 (0~9)

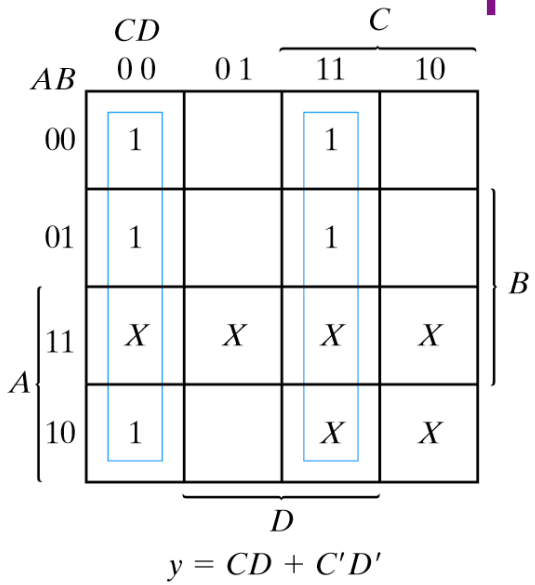
② Formulation
- wxyz = ABCD+0011

Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Code Conversion Example (2/3)



$z = D'$

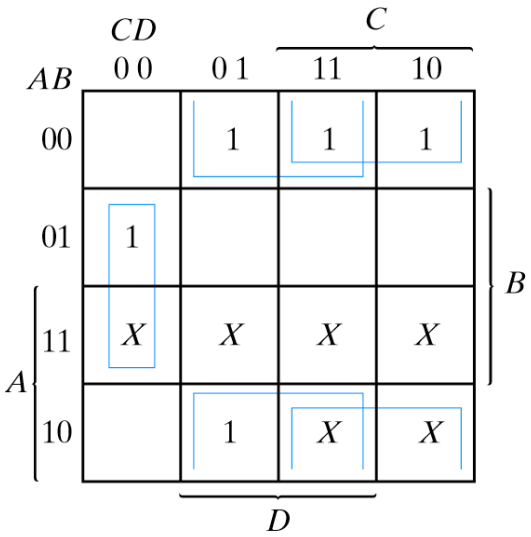


$y = CD + C'D'$

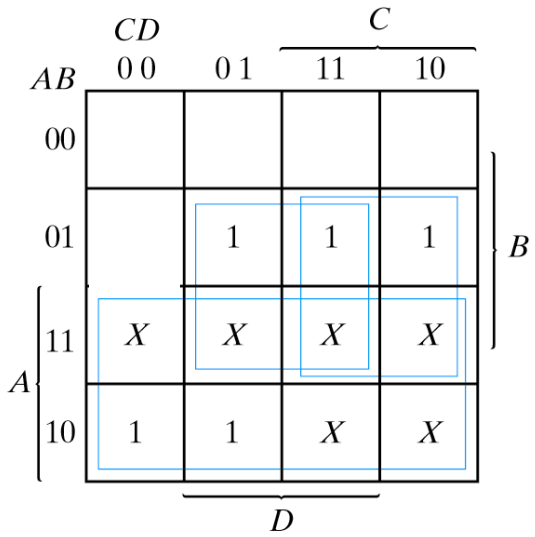
③ Optimization:

- Determine simplified Boolean function

- $z = D'$ $(C+D)'$
- $y = CD + C'D'$
- $x = B'C + B'D + BC'D'$
- $w = A + BC + BD$



$X = B'C + B'D + BC'D'$



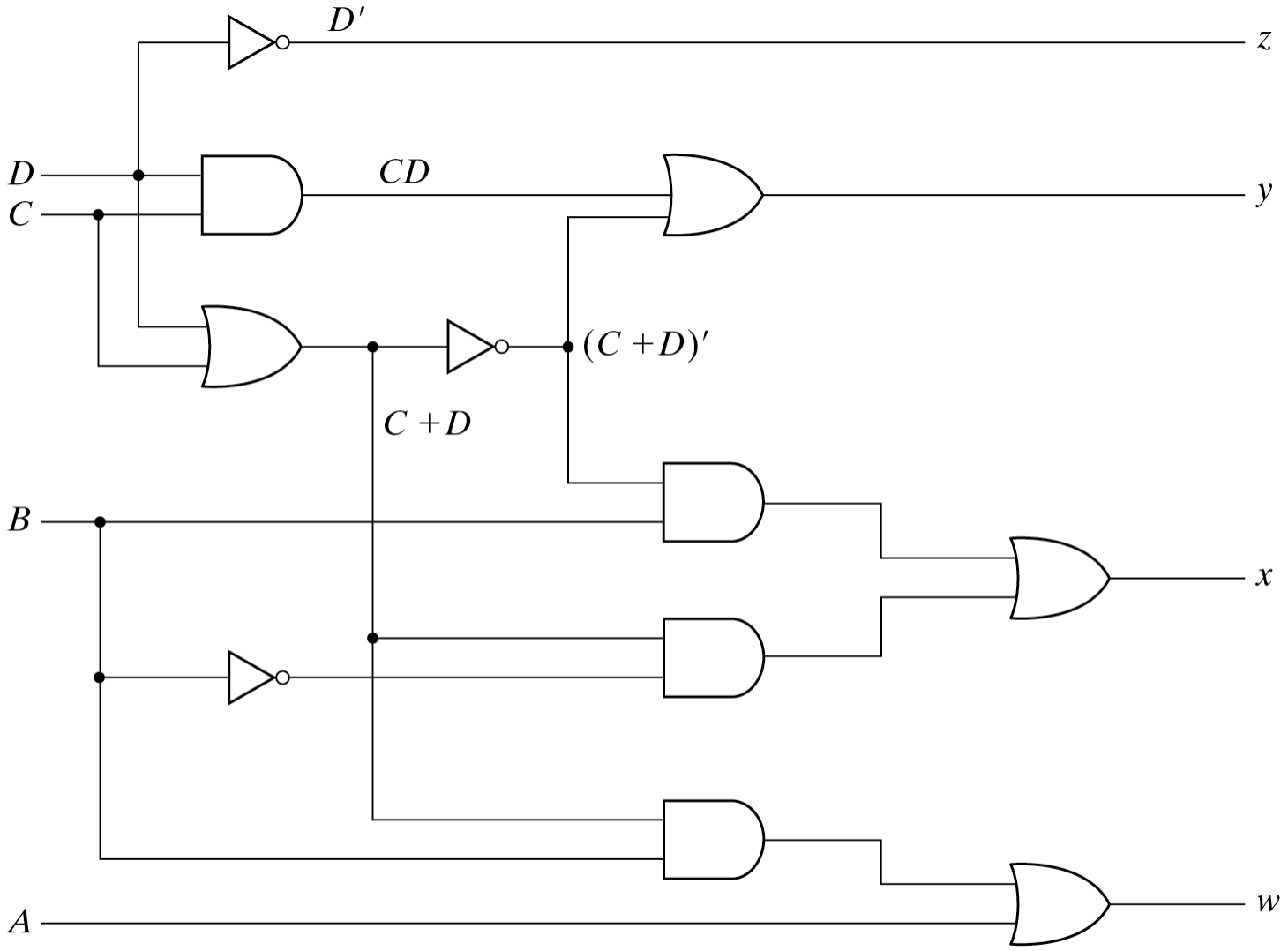
$w = A + BC + BD$

➔ multi-level, non-standard

- $z = D'$
- $y = CD + (C + D)'$
- $x = B'(C + D) + B(C + D)'$
- $w = A + B(C + D)$

Code Conversion Example (3/3)

④ Logic diagram

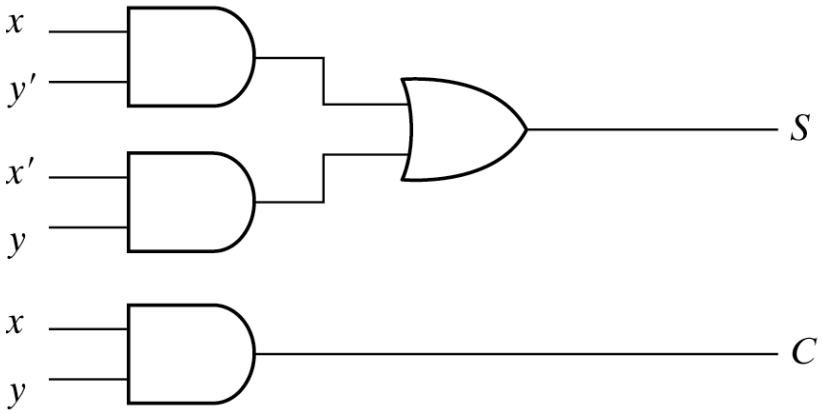


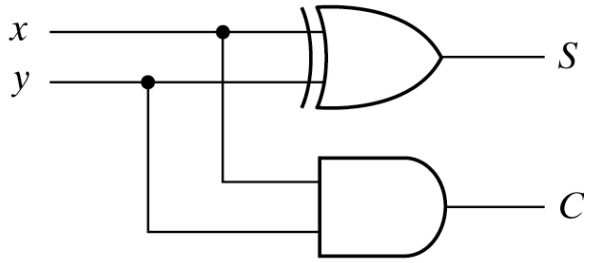
Binary Adder-Subtractor

Binary Half Adder (HA)

- IOs
  - Input: x, y
  - Outputs: C (carry), S (sum)
- Truth table and functions
  - $S = x'y + xy' = x \oplus y$
  - $C = xy$
- Logic diagram

x y	C S
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0

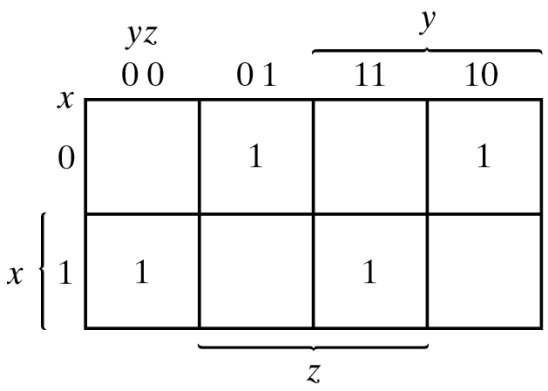




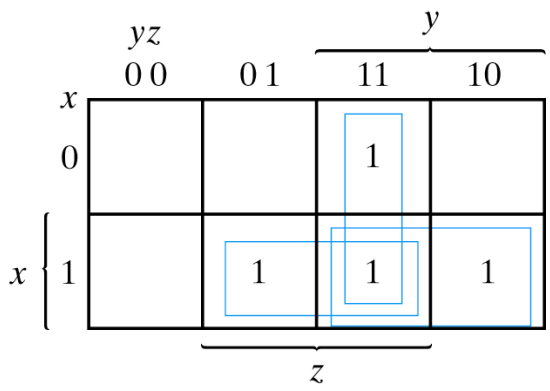
Binary Full Adder (FA) (1/3)

- IOs
  - Input: x, y, z (carry from previous lower significant bit)
  - Outputs: C (carry), S (sum)
- Truth table

x	y	z	C	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



$$C = \sum(3,5,6,7)$$
$$= xy + yz + xz$$



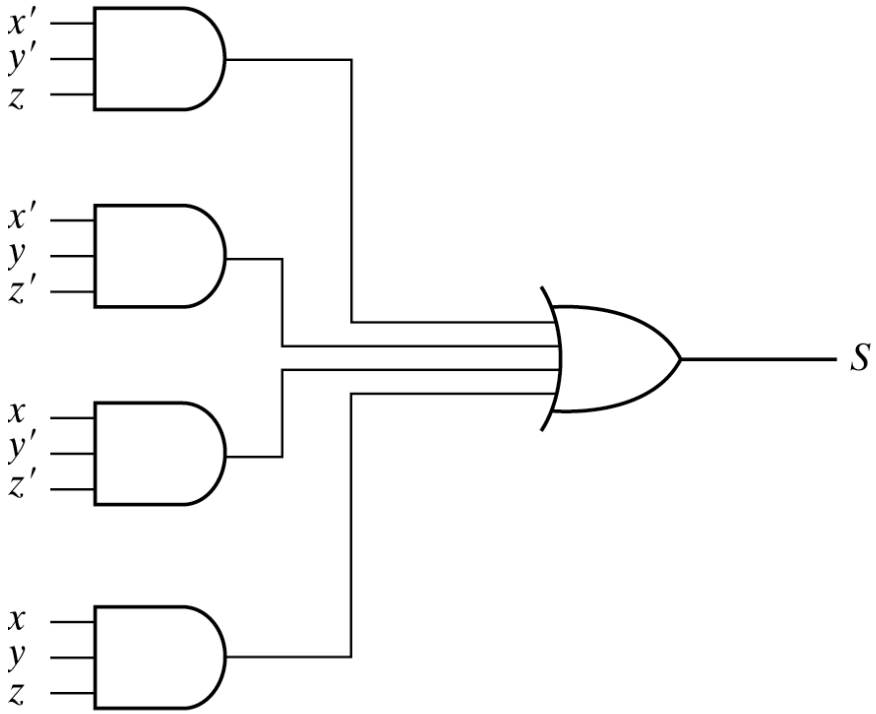
$$S = \sum(1,2,4,7)$$
$$= x'y'z + x'yz' + xy'z' + xyz = x \oplus y \oplus z$$

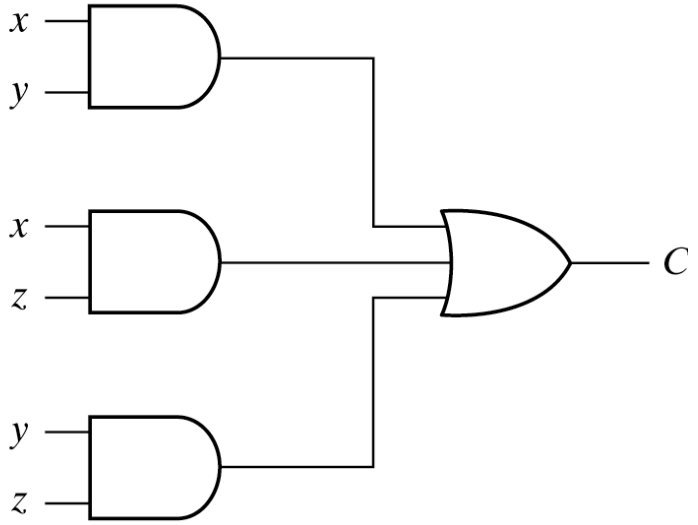
Binary Full Adder (FA) (2/3)

- 2-level logic diagram

$$S = x'y'z + x'yz' + xy'z' + xyz = x \oplus y \oplus z$$

$$C = xy + yz + xz$$

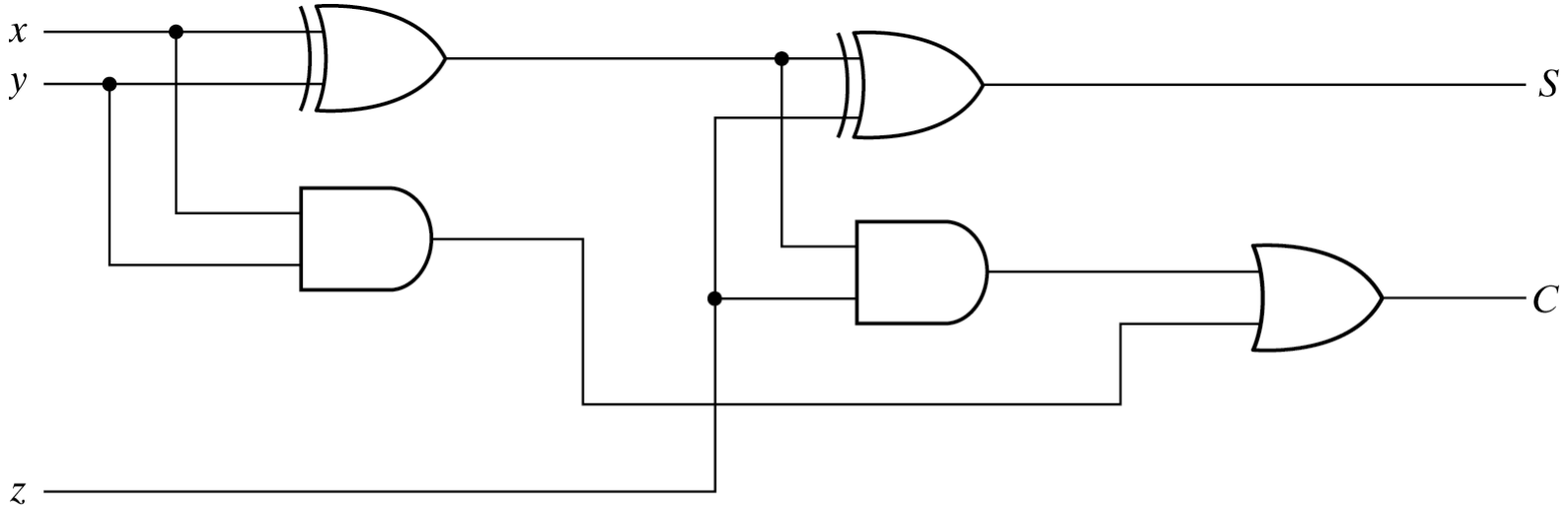




Binary Full Adder (FA) (3/3)

- Full adder implemented with half adders
  - Two half adders and one OR gate

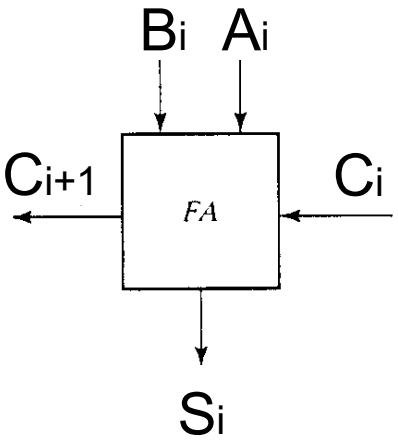
$$S = (x \oplus y) \oplus z$$
$$C = x'yz + xy'z + (xyz' + xyz)$$
$$= (x \oplus y)z + xy$$

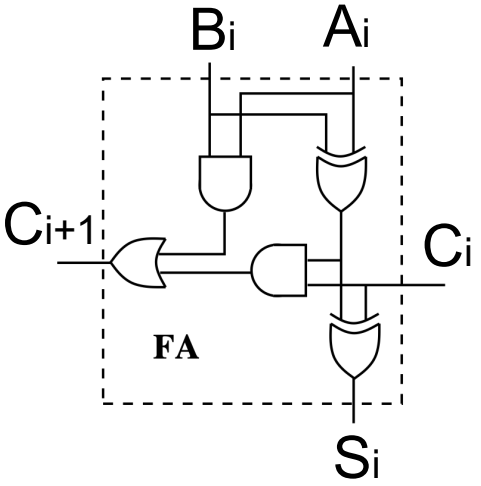


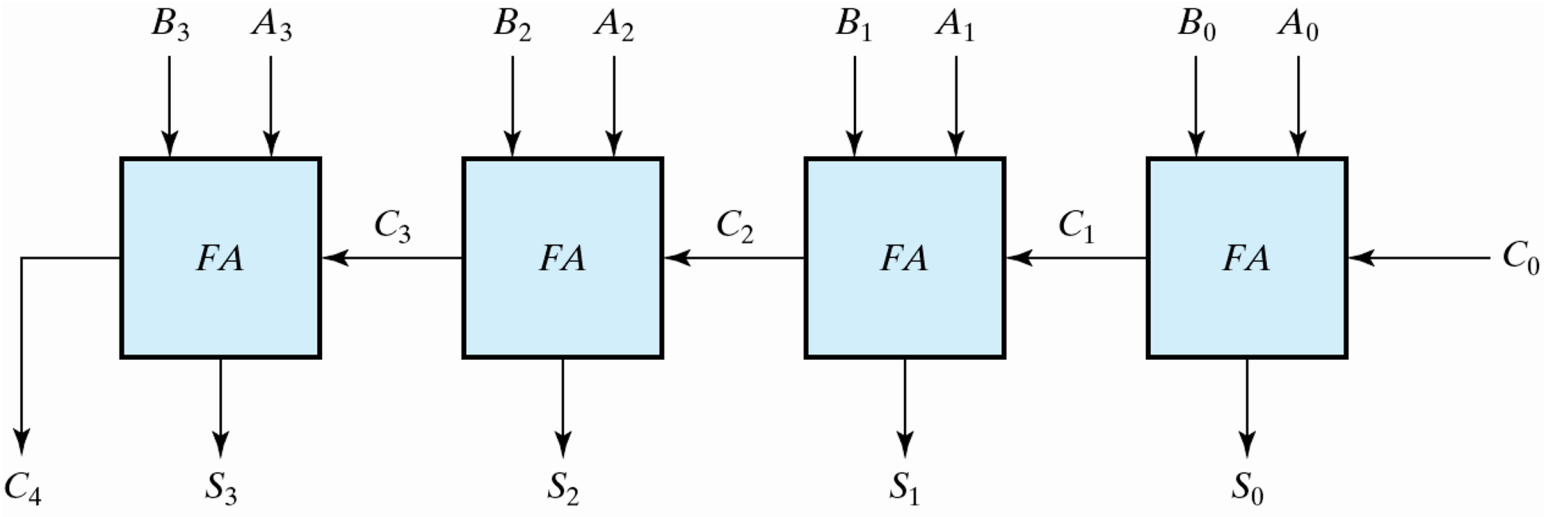
Binary (Ripple-Carry) Adder (1/2)

- Produces arithmetic sum of two binary numbers

Subscript i:	3	2	1	0	
Input carry					C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum					S_i
Output carry					C_{i+1}

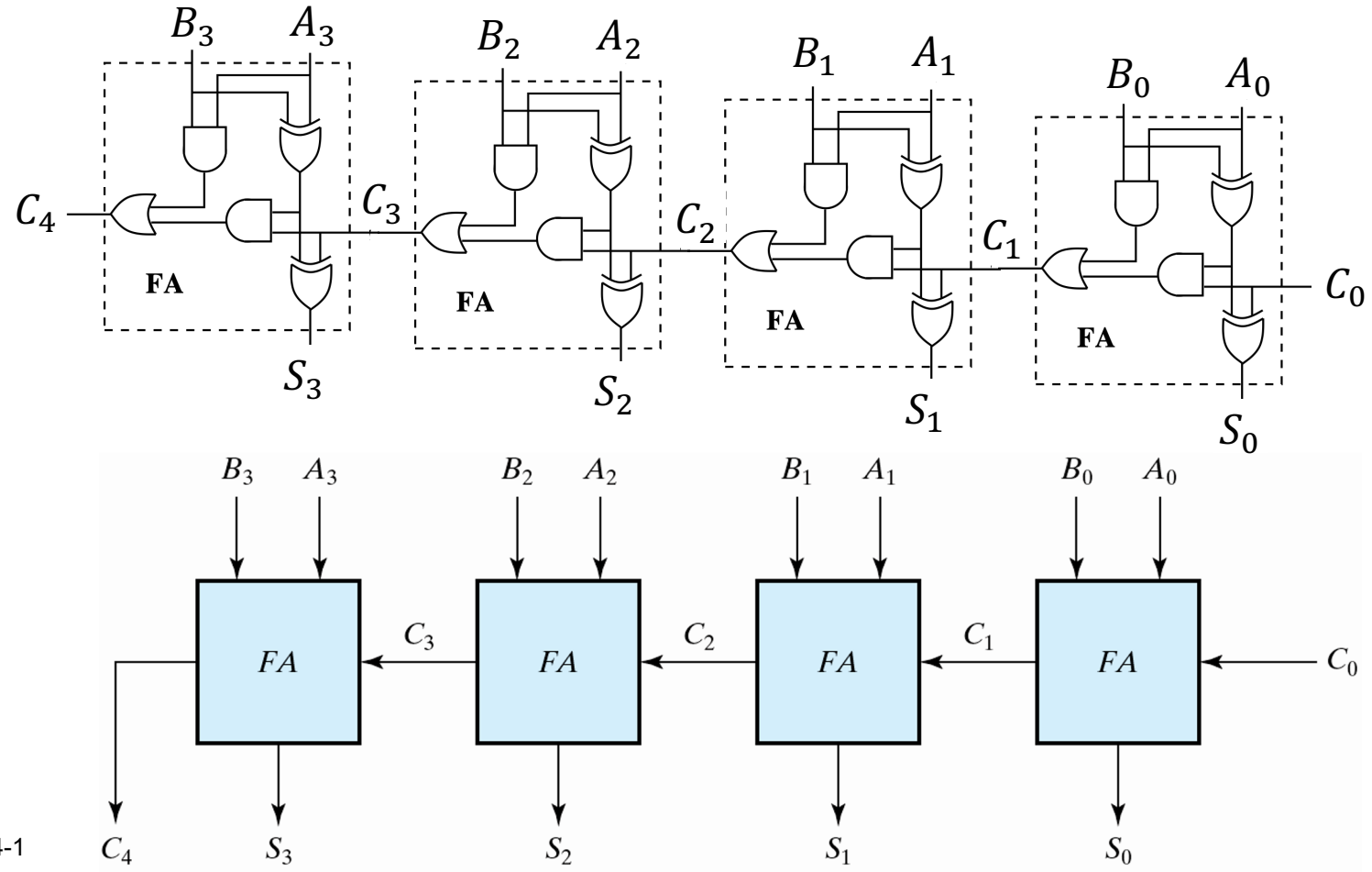






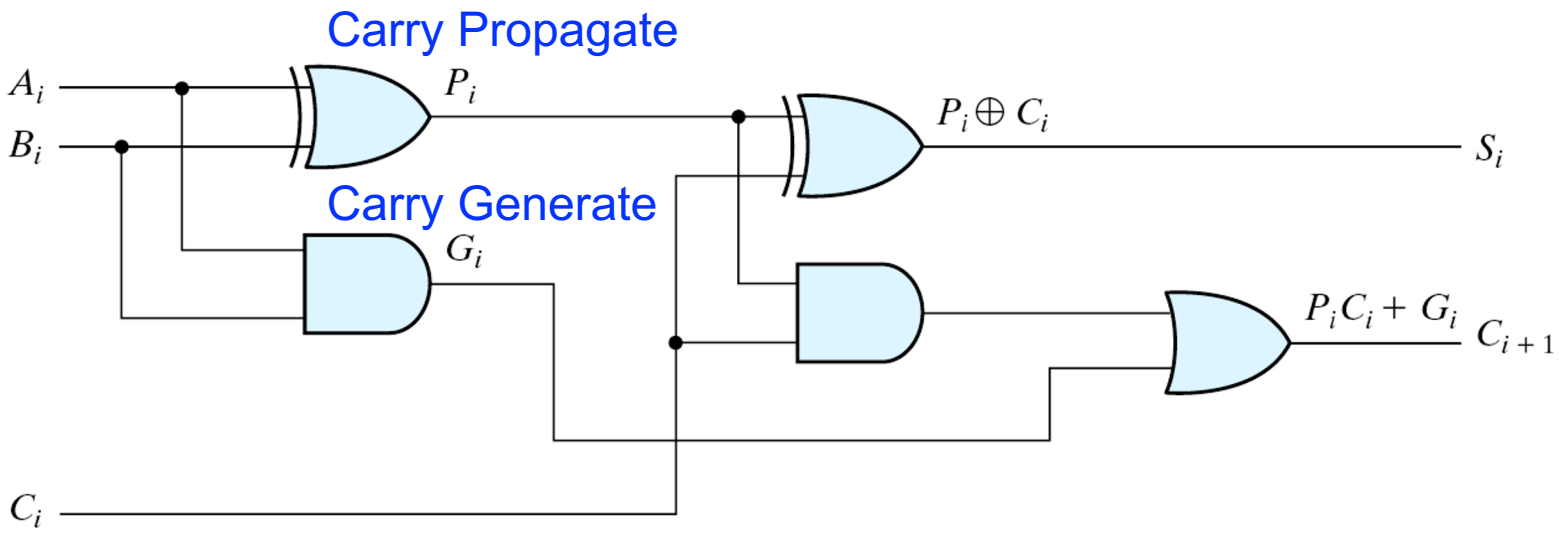
Binary (Ripple-Carry) Adder (2/2)

- The computation time of a ripple-carry adder grows linearly with word length *n*
  - T=O(n) due to carry chain



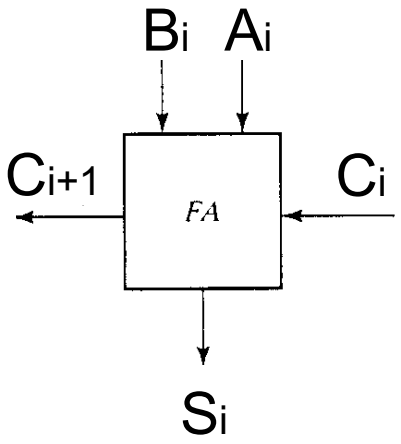
Carry Propagation

- For ripple carry adder
  - Longest propagation delay: $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow C_4$
  - 8 gate levels (for 4-bit adder)
  - $2n$ gate levels for the carry to propagate from input to output for an n-bit adder



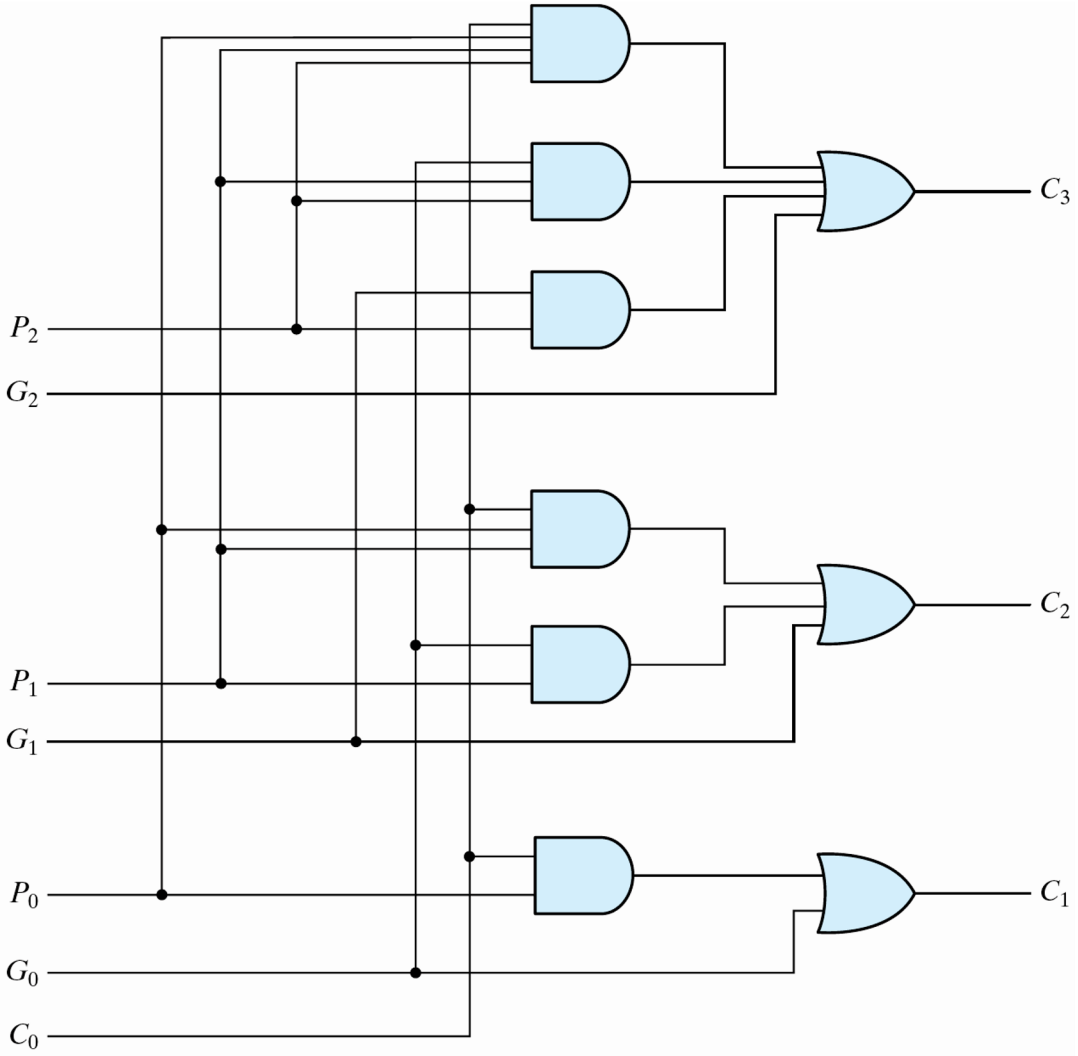
Carry Lookahead Adder (CLA) (1/5)

- Reduce the carry propagation delay
  - Using faster gates
  - Parallel adders, e.g., the *carry lookahead adder (CLA)*
- Carry Lookahead Adder (CLA)
  - Carry propagate: $P_i = A_i \oplus B_i$
  - Carry generate: $G_i = A_i B_i$
  - Sum: $S_i = P_i \oplus C_i$
  - Carry: $C_{i+1} = G_i + P_i C_i$
  - $C_1 = G_0 + P_0 C_0$ $= A_0 B_0 + (A_0 \oplus B_0) C_0$
  - $C_2 = G_1 + P_1 C_1$
  - $C_3 = G_2 + P_2 C_2$



Carry Lookahead Adder (CLA) (2/5)

- Logic diagram of carry lookahead generator

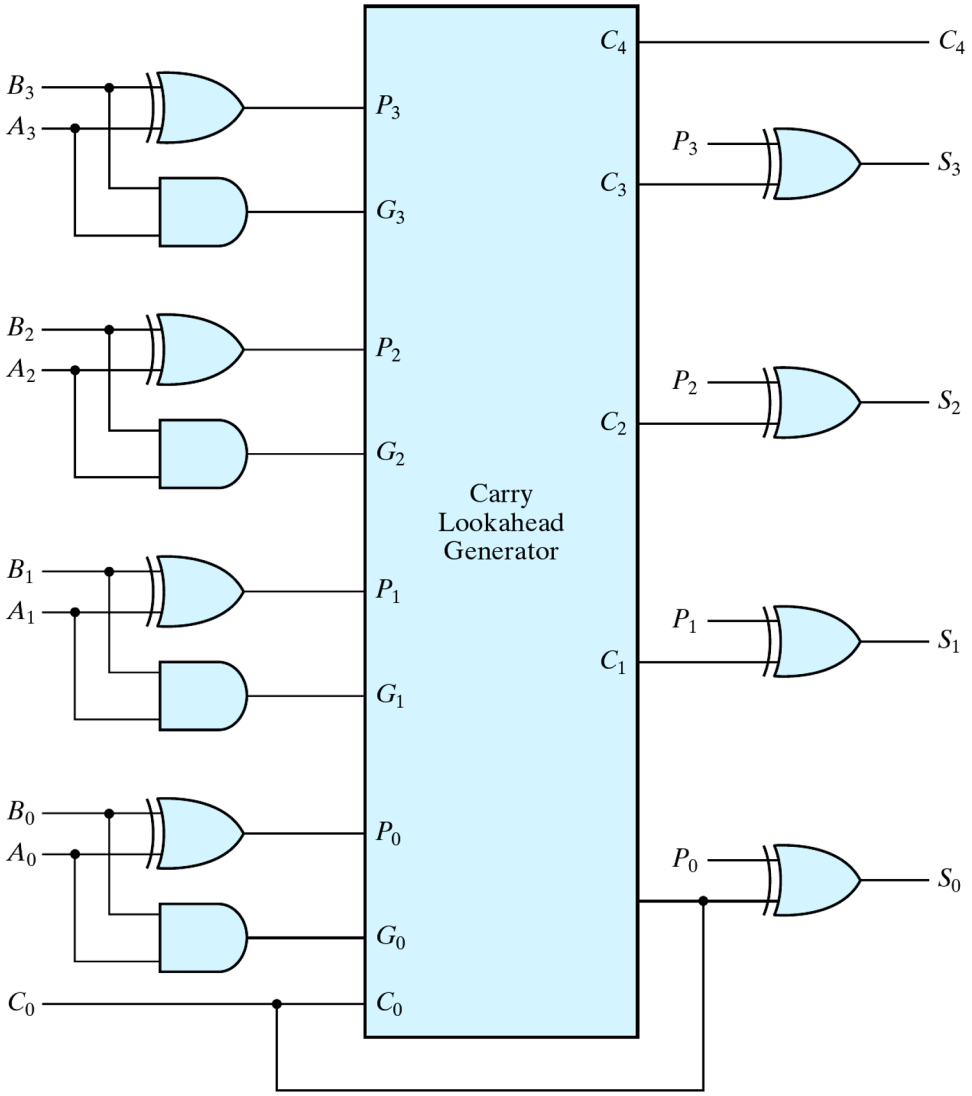


Carry Lookahead Adder (CLA) (3/5)

- No carry (propagation) chain
- 2-level logic achieved
- However, fan-in number of AND/OR gates still grows gradually
  - The overall delay is not constant
  - Fan-in number is usually limited to 4 bits

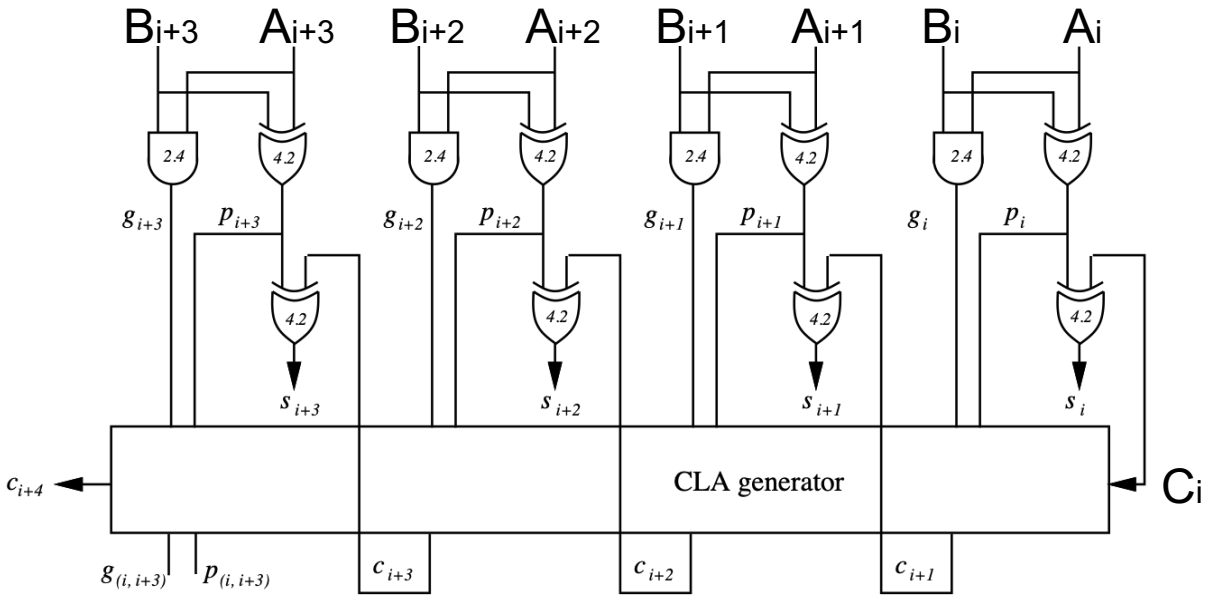
Carry Lookahead Adder (CLA) (4/5)

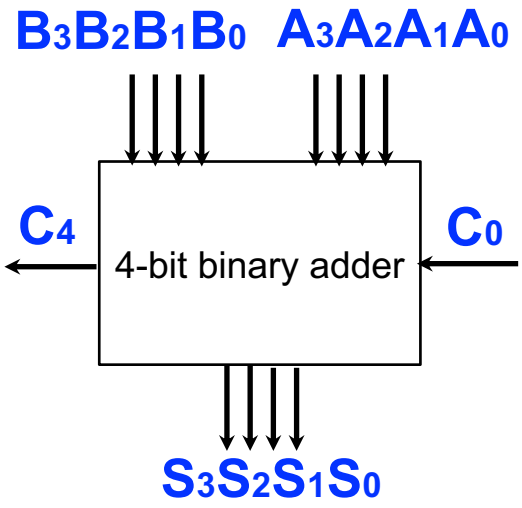
- 4-bit adder with carry lookahead



Carry Lookahead Adder (CLA) (5/5)

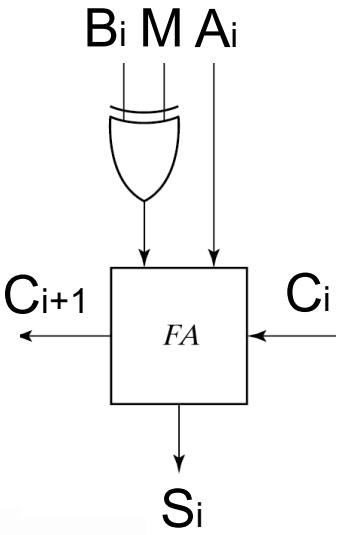
- Detailed logic diagram of 4-bit CLA

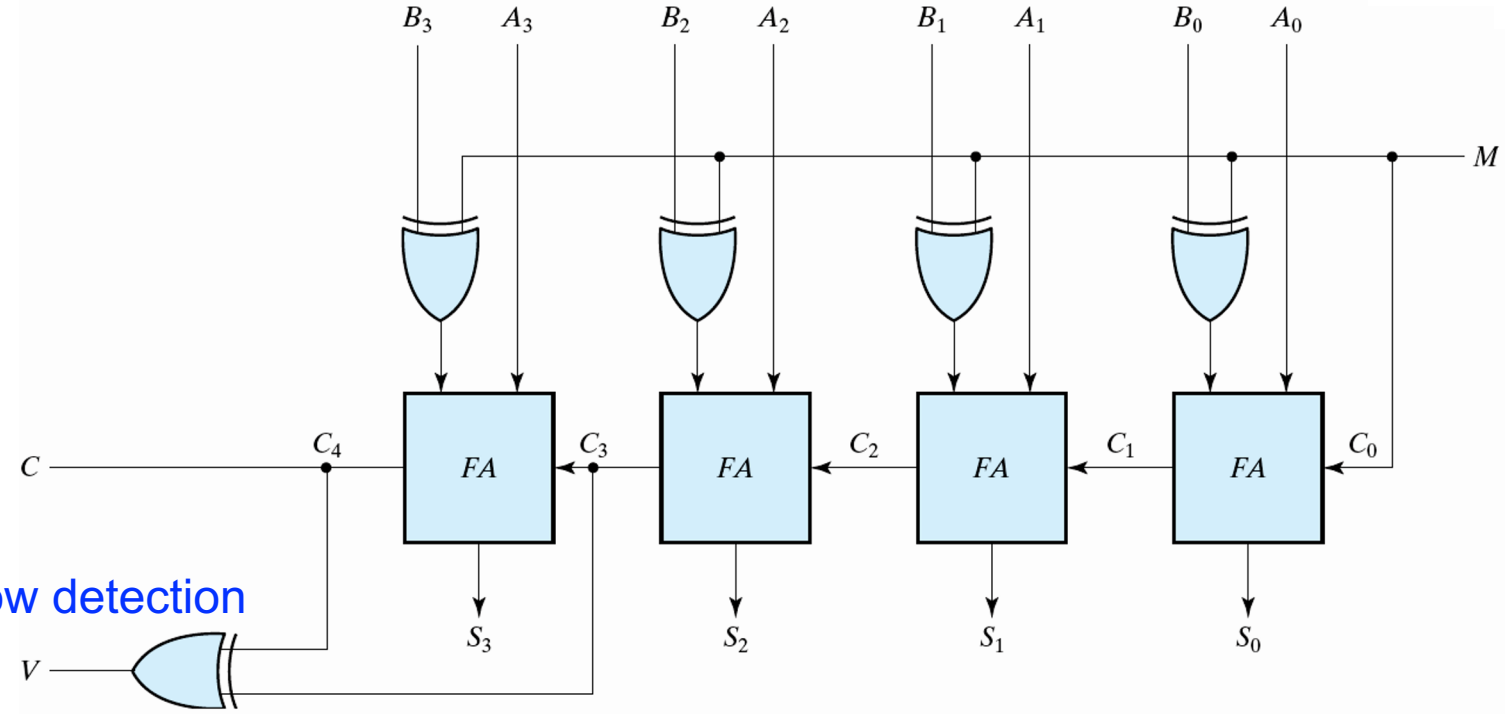




Binary Adder/Subtractor

- Binary subtraction normally is performed by adding the minuend to the 2's complement of the subtrahend
- 4-bit adder-subtractor
  - M = 0: A + B
  - M = 1: A – B = A + (2's complement of B) = A + B' + 1





oVerflow detection

Overflow (1/2)

- Adding two positive numbers and obtaining a negative number; or
- Adding two negative numbers and obtaining a positive number
- Overflow can be detected
  - If the carry into the sign bit position and the carry out of the sign bit position are not equal
  - V = 1 when the overflow occurs

Overflow (2/2)

- For *n*-bit adder/subtractor, overflow occurs if

$$V = c_{n-1} \oplus c_n$$

① If $c_{n-1} = 1$ && $c_n = 0$

$\Longrightarrow A_{n-1} = B_{n-1} = 0$

$\Longrightarrow S_{n-1} = 1$

$\Longrightarrow$ Adding two positive numbers, result is negative

② If $c_{n-1} = 0$ && $c_n = 1$

$\Longrightarrow A_{n-1} = B_{n-1} = 1$

$\Longrightarrow S_{n-1} = 0$

$\Longrightarrow$ Adding two negative numbers, result is positive

Decimal Adder

Decimal Adder (1/3)

- Addition of 2 decimal digits in BCD
  - 9 inputs: 2 BCD's and one carry-in
  - 5 outputs: 1 BCD and one carry-out
- Design approaches
  - Truth table with 2^9 entries (seriously?!)
  - Using binary adders
    - A digit in BCD cannot exceed 9
      - The max sum 9 + 9 + 1 = 19
    - Binary to BCD (adding 6 for correction)

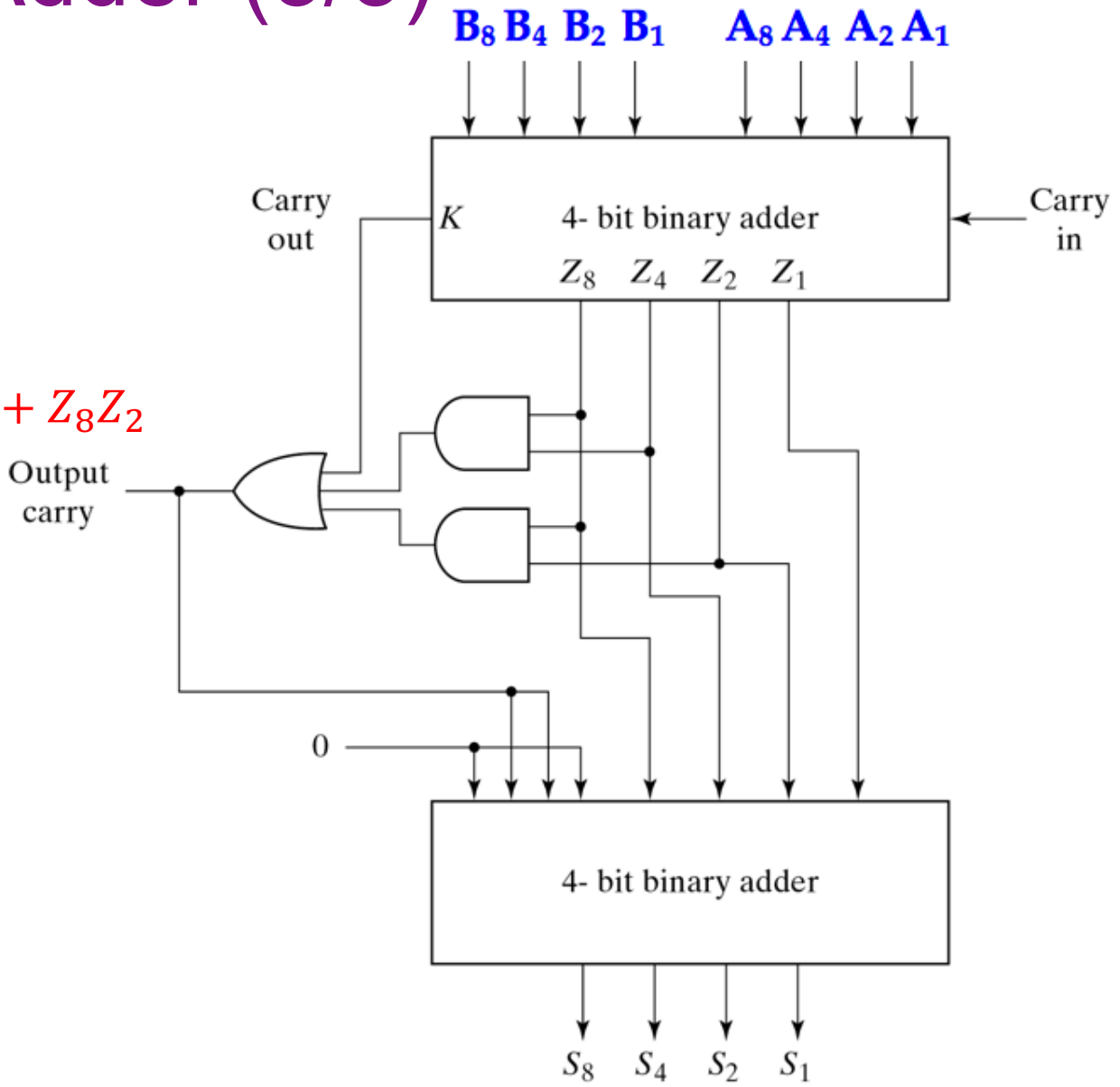
Decimal symbol	BCD digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Decimal Adder (2/3)

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

Decimal Adder (3/3)

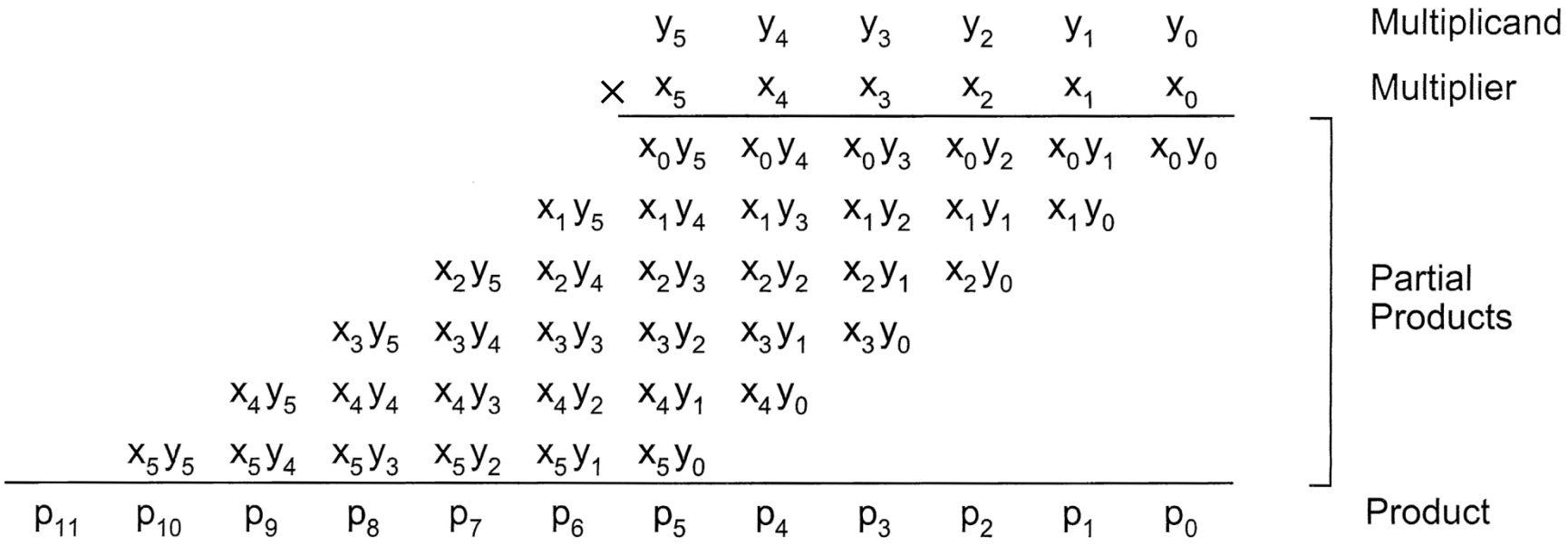
$$C = K + Z_8Z_4 + Z_8Z_2$$



Binary Multiplier

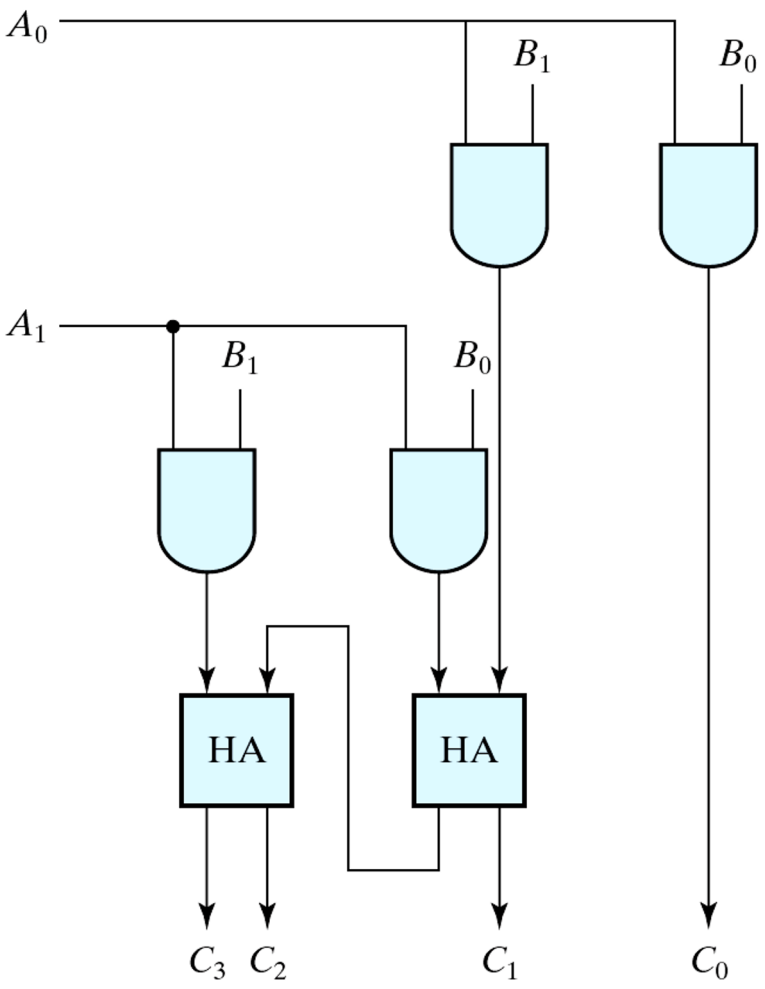
Multiplication

- Multiplication consists of
  - Generation of partial products
  - Accumulation of shifted partial products

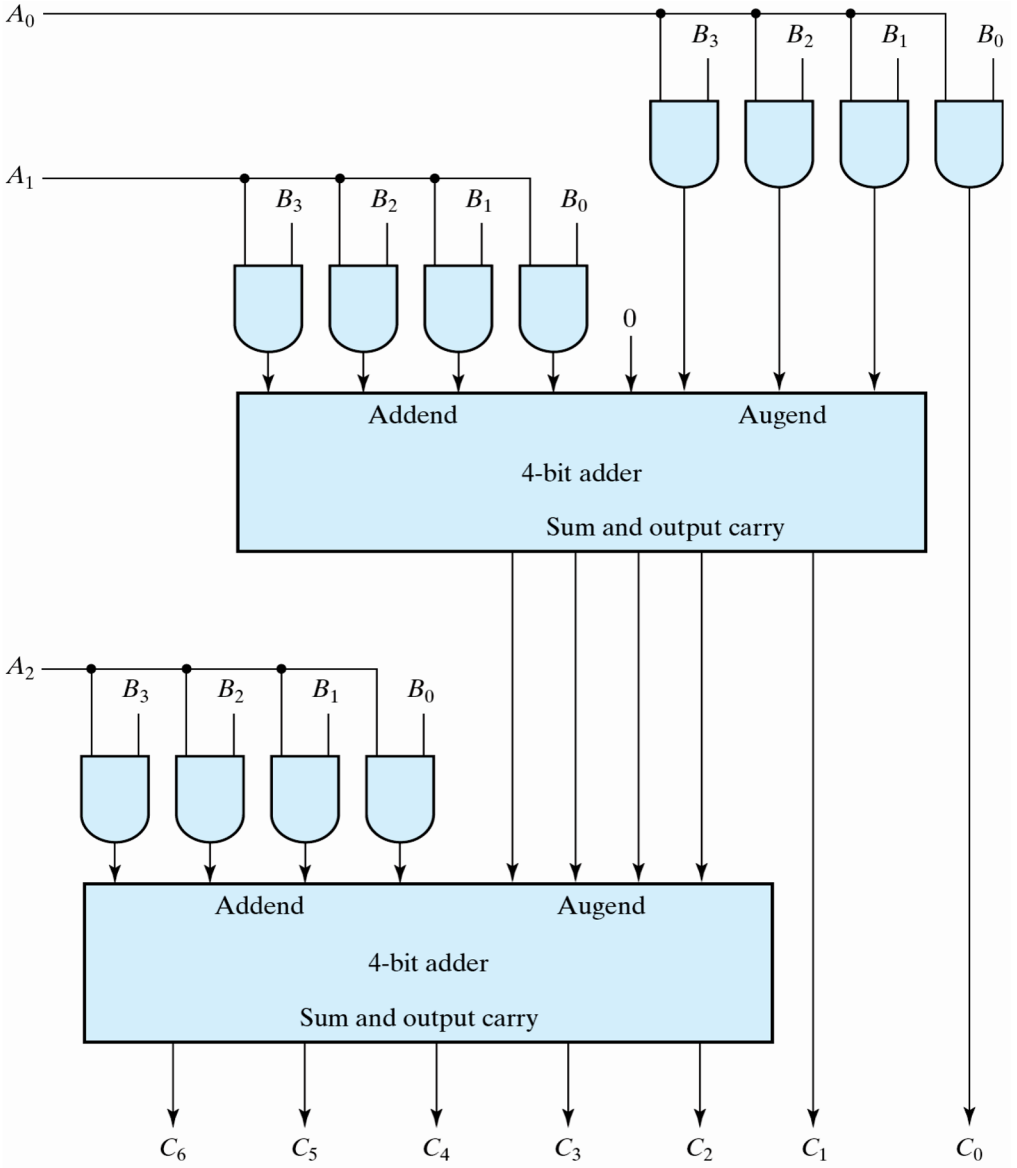


2-bit by 2-bit Binary Multiplier

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$

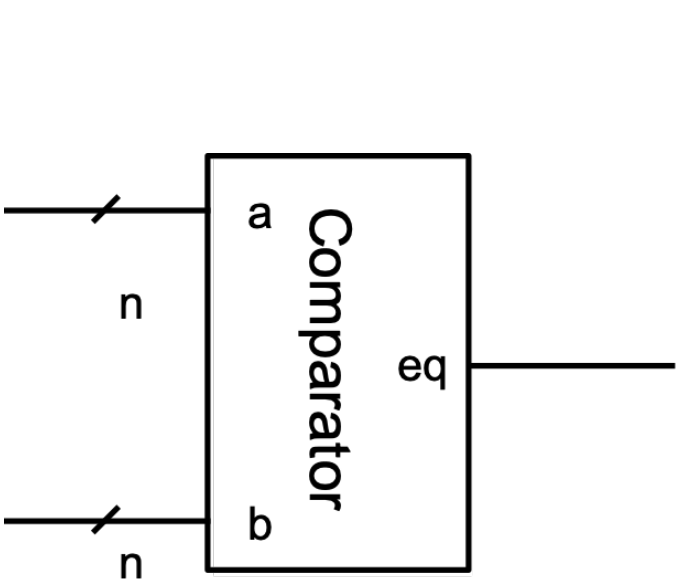


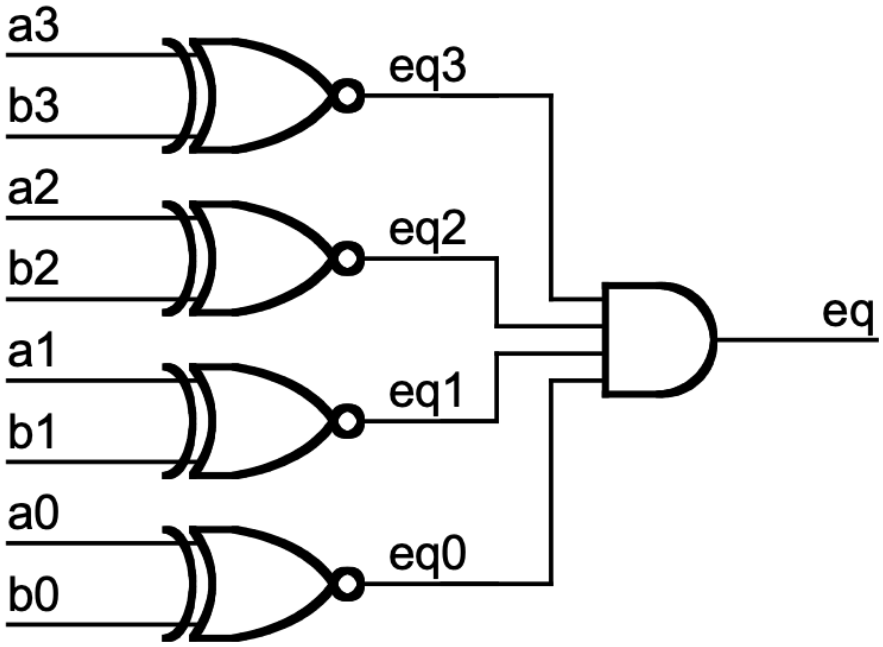
4-bit by 3-bit Binary Multiplier



Magnitude Comparator

Equality Comparator



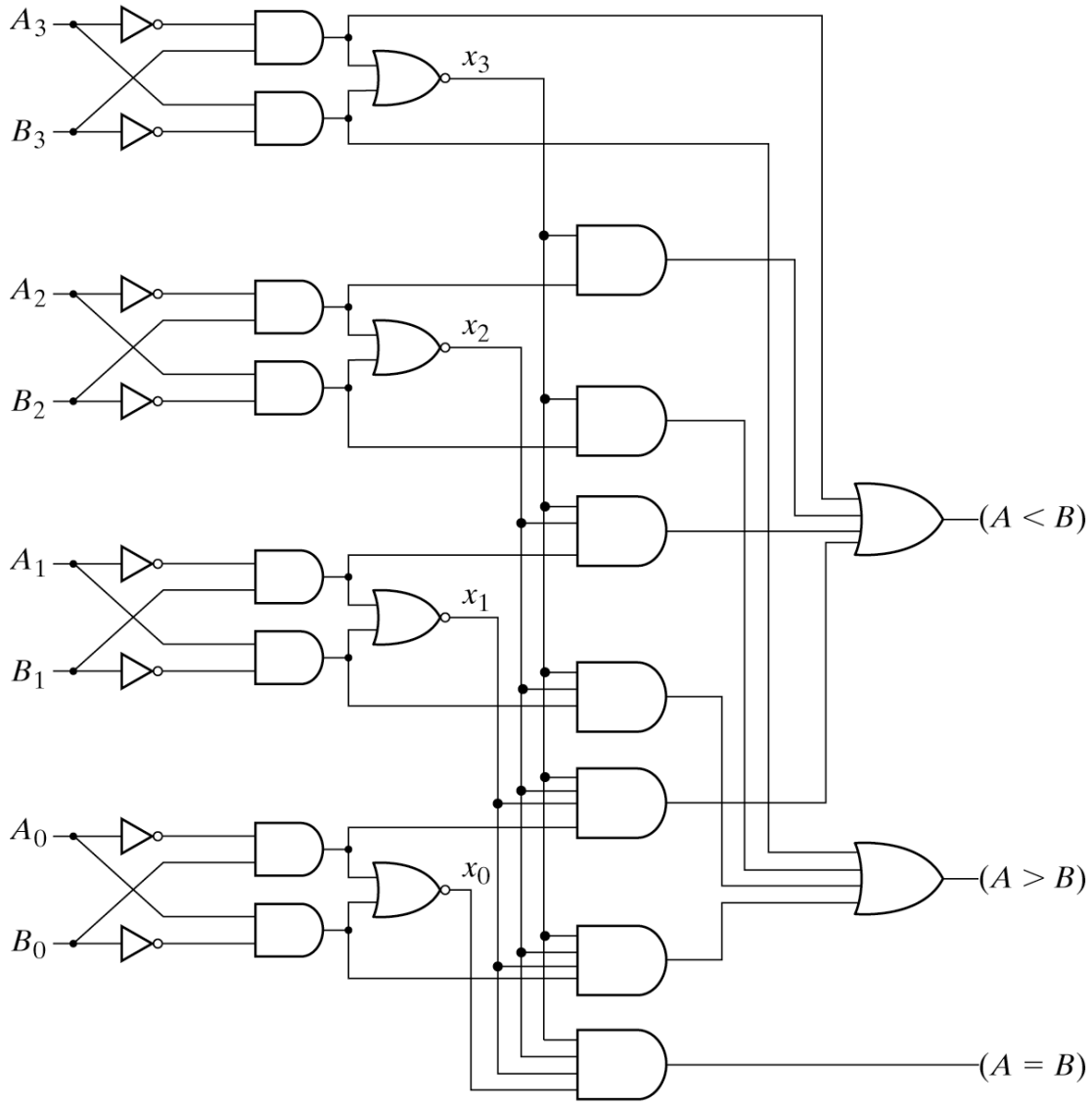


Magnitude Comparator (1/2)

- Comparison of two positive numbers, three possible results
  - $A = B$, $A > B$, $A < B$
- Design approaches (for *n*-bit numbers)
  - By the truth table: 2^{2n} rows => not practicable
  - By algorithm to build a regular circuit

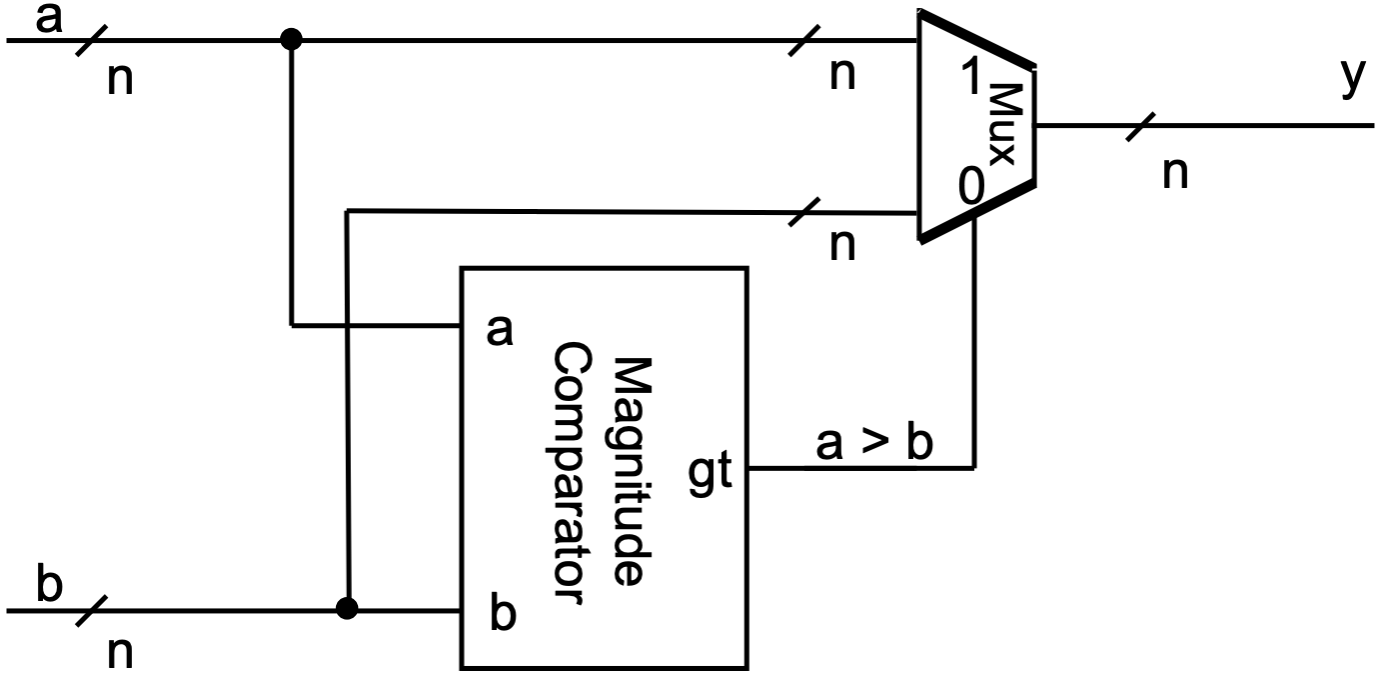
    $A = A_3A_2A_1A_0$, $B = B_3B_2B_1B_0$
    - $A = B$, if $A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$ and $A_0 = B_0$
      - Equality: $x_i = A_iB_i + A_i'B_i' = (A \oplus B)'$ (XNOR)
      - $A = B \rightarrow x_3x_2x_1x_0$
    - $A > B$
      - $A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$
    - $A < B$
      - $A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$

Magnitude Comparator (2/2)



Maximum Unit

$$y = \max\{a, b\}$$



Decoders

Decoders

- A decoder is a combinational circuit that converts binary information from *n* input lines to an *m* (maximum of 2^n) unique output lines
  - *n*-to-*m*-line decoder (or $n \times m$ decoder)
  - Output variables are mutually exclusive because only one output can be equal to 1 at any time (the vary 1-minterm)
  - Can be implemented with AND gates

3-to-8-line Decoder (1/2)

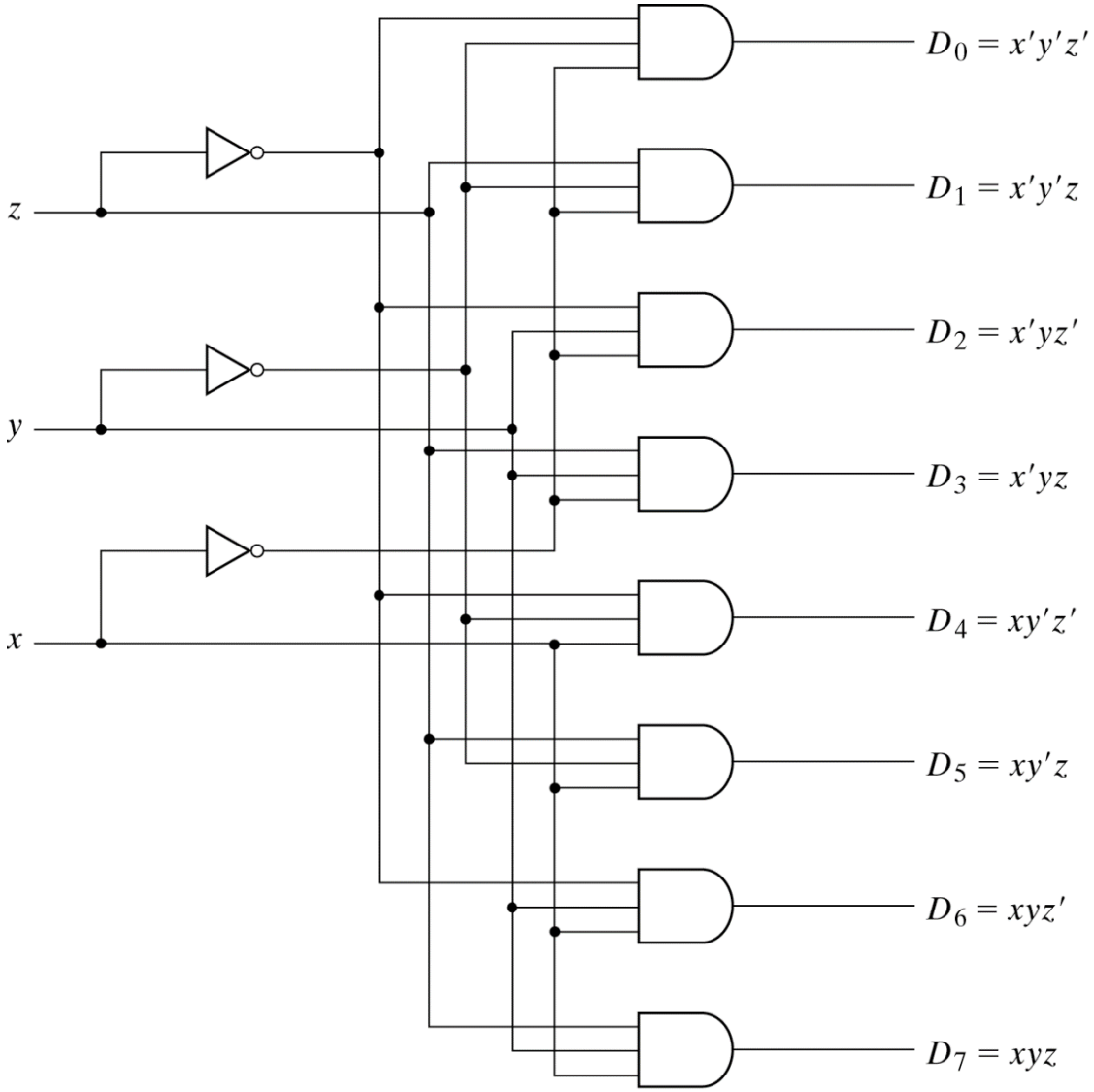
Truth Table of a Three-to-Eight-Line Decoder

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Binary Coding

One-hot Coding

3-to-8-line Decoder (2/2)



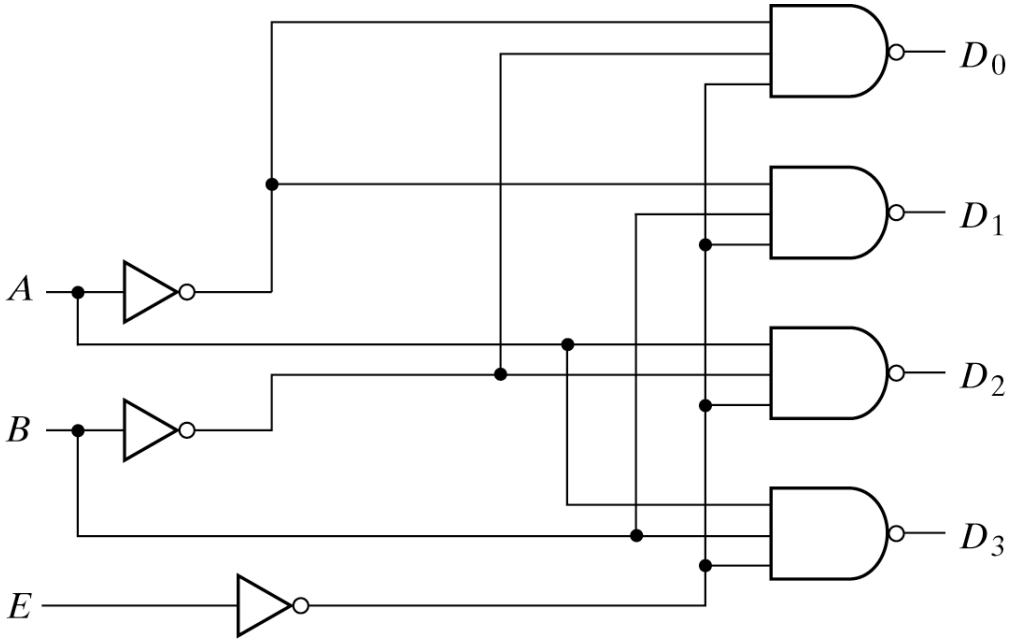
Decoders with Enable Input (1/3)

- Line decoder with enable control (E)
- Also called demultiplexer (DMUX, DEMUX)

Decoders with Enable Input (2/3)

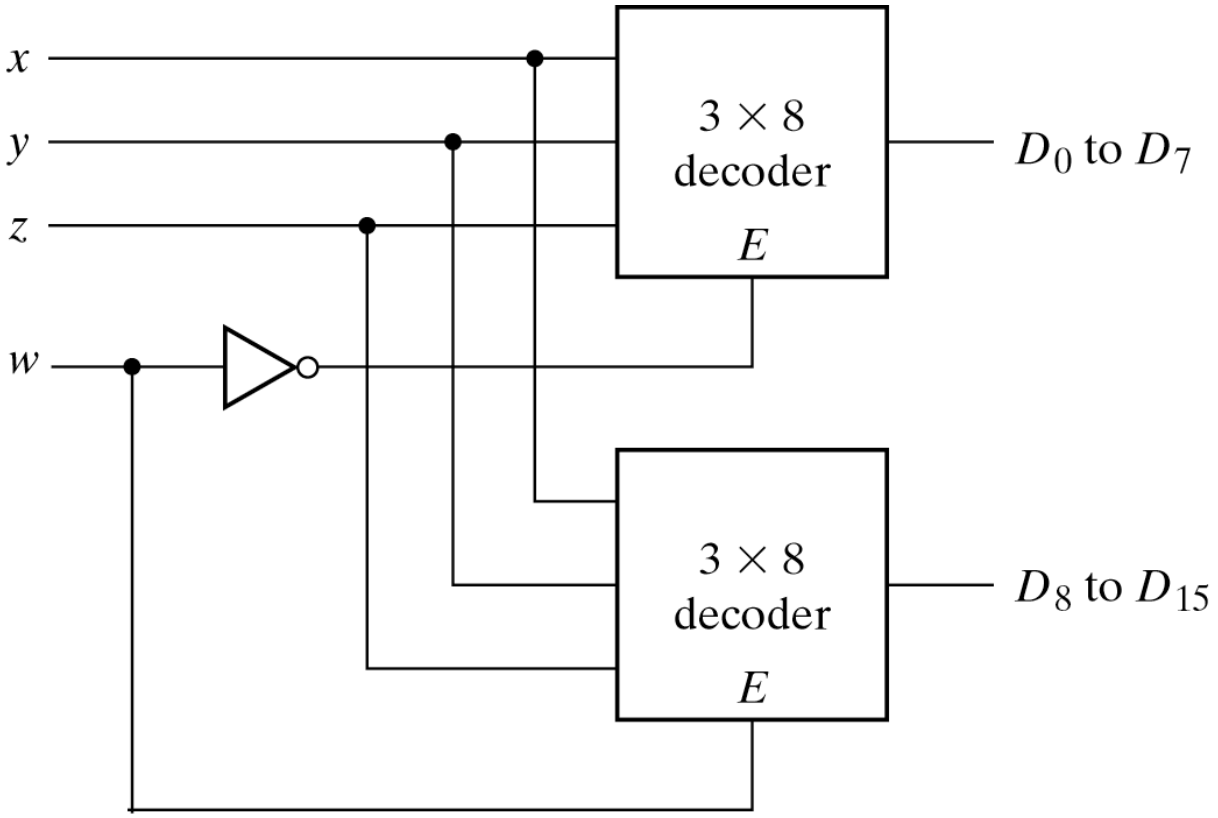
- Constructed with NAND gates
  - Decode minterms in their complemented form

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0



Decoder Expansion

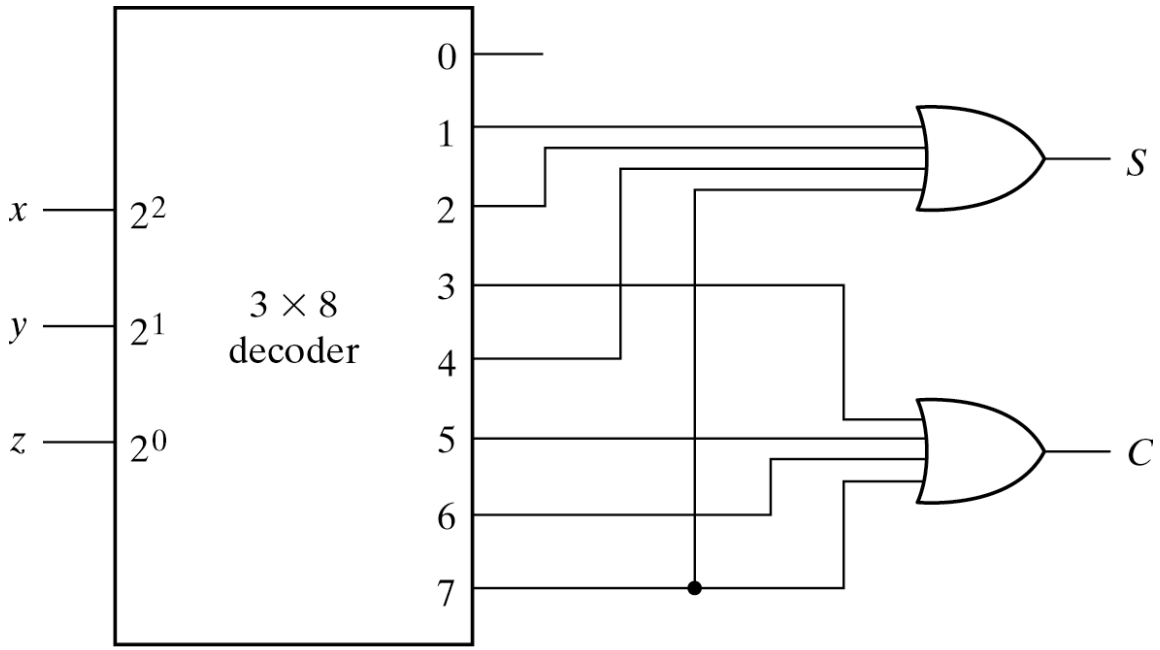
- Larger decoders can be implemented with smaller decoders
  - 4x16 decoder with two 3x8 decoders



Combinational Logic Implementation (1/2)

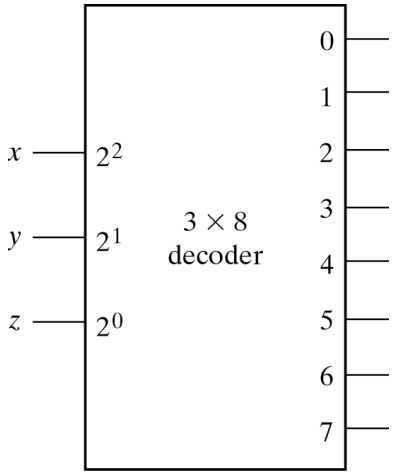
- Any combinational circuit with *n* inputs and *m* outputs can be implemented with an *n*-to-2^n decoder in conjunction with *m* external OR gates
- A full-adder
  - $S(x, y, z) = \sum(1, 2, 4, 7)$
  - $C(x, y, z) = \sum(3, 5, 6, 7)$

x	*y*	*z*	*C*	*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



Combinational Logic Implementation (2/2)

- A function with $k > 2^n/2$ minterms can be expressed in its complement form with $2^n - k$ minterms
  - NOR gates are used instead of OR gates
- If NAND gates are used for the decoder, the output OR gates are replaced by NAND gates
  - AND-OR => NAND-NAND

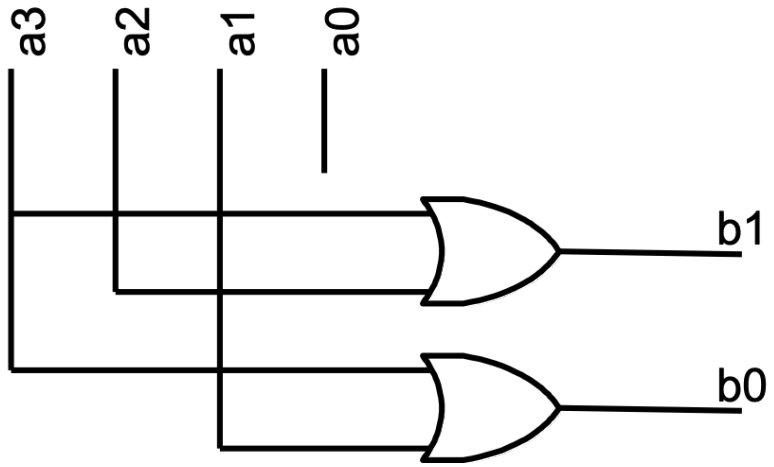


Encoders

Encoder (1/3)

- An encoder is an inverse of a decoder
  - Converts a one-hot input signal to a binary-encoded output signal
  - Other input patterns are forbidden in the truth table
- Example: a 4-to-2 encoder

a3	a2	a1	a0	b1	b0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



$$b_0 = a_3 + a_1$$

$$b_1 = a_3 + a_2$$

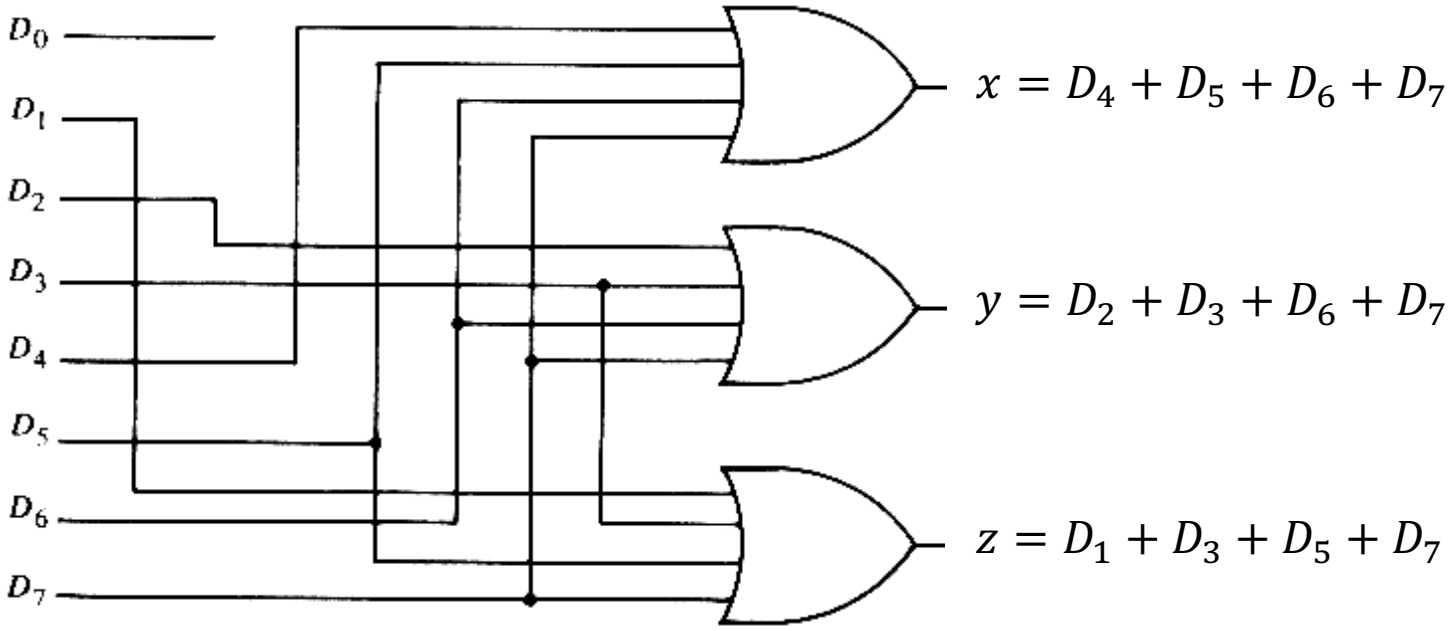
Encoders (2/3)

- A combinational logic that performs the inverse operation of a decoder
  - Only one input has value 1 at any given time
  - Can be implemented with OR gates

Truth Table of an Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Encoders (3/3)



- Limitations

Illegal Inputs

  - When both D_3 an D_6 go high, output will be 111 → ambiguity
  - Use priority encoder!

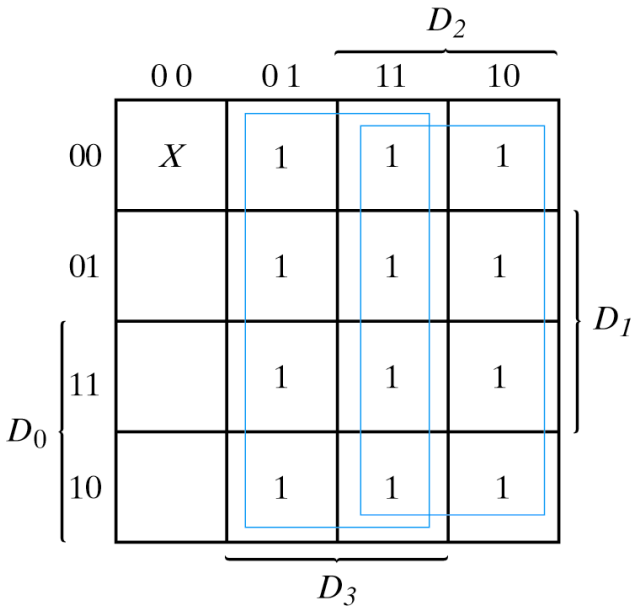
Priority Encoders (1/2)

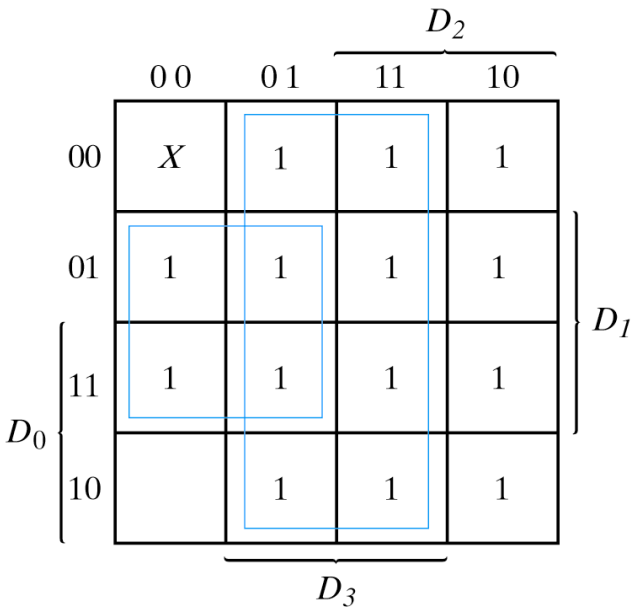
- Ensure only one of the input is encoded
- D_3 has the highest priority, while D_0 has the lowest priority
- X is the don't care conditions
- V is the valid output indicator

Truth Table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Priority Encoders (2/2)

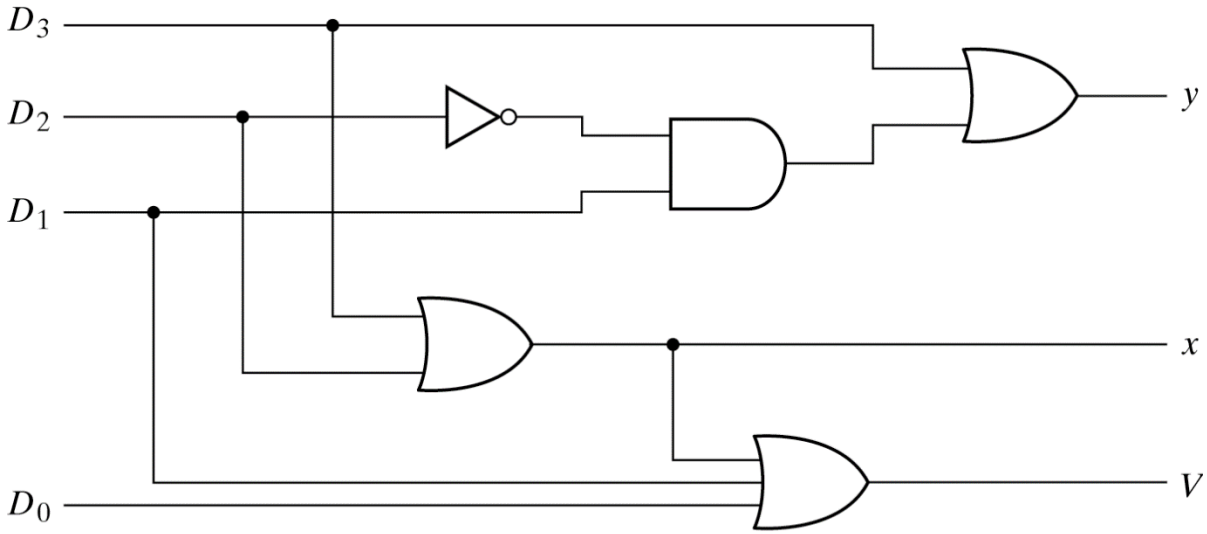




$x = D_2 + D_3$

$y = D_3 + D_1 D_2'$

$V =$



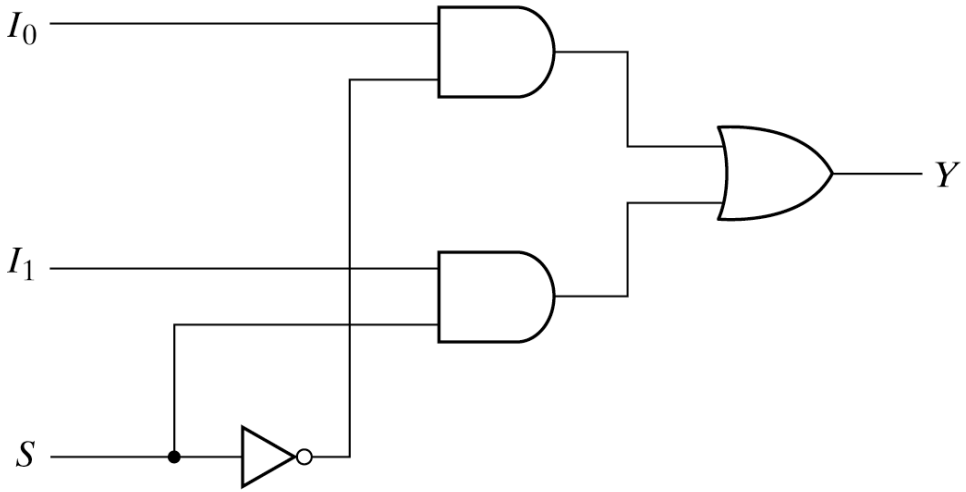
Multiplexer

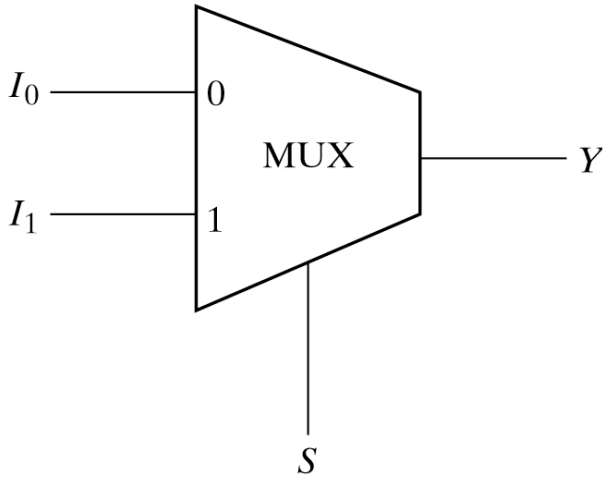
Multiplexers

- A multiplexer (or MUX) selects (usually by *n* select lines) binary information from one of many (usually 2^n) input lines and directs it to a single output line
- 2-to-1 MUX (2:1 MUX)

S	Y
0	I_0
1	I_1

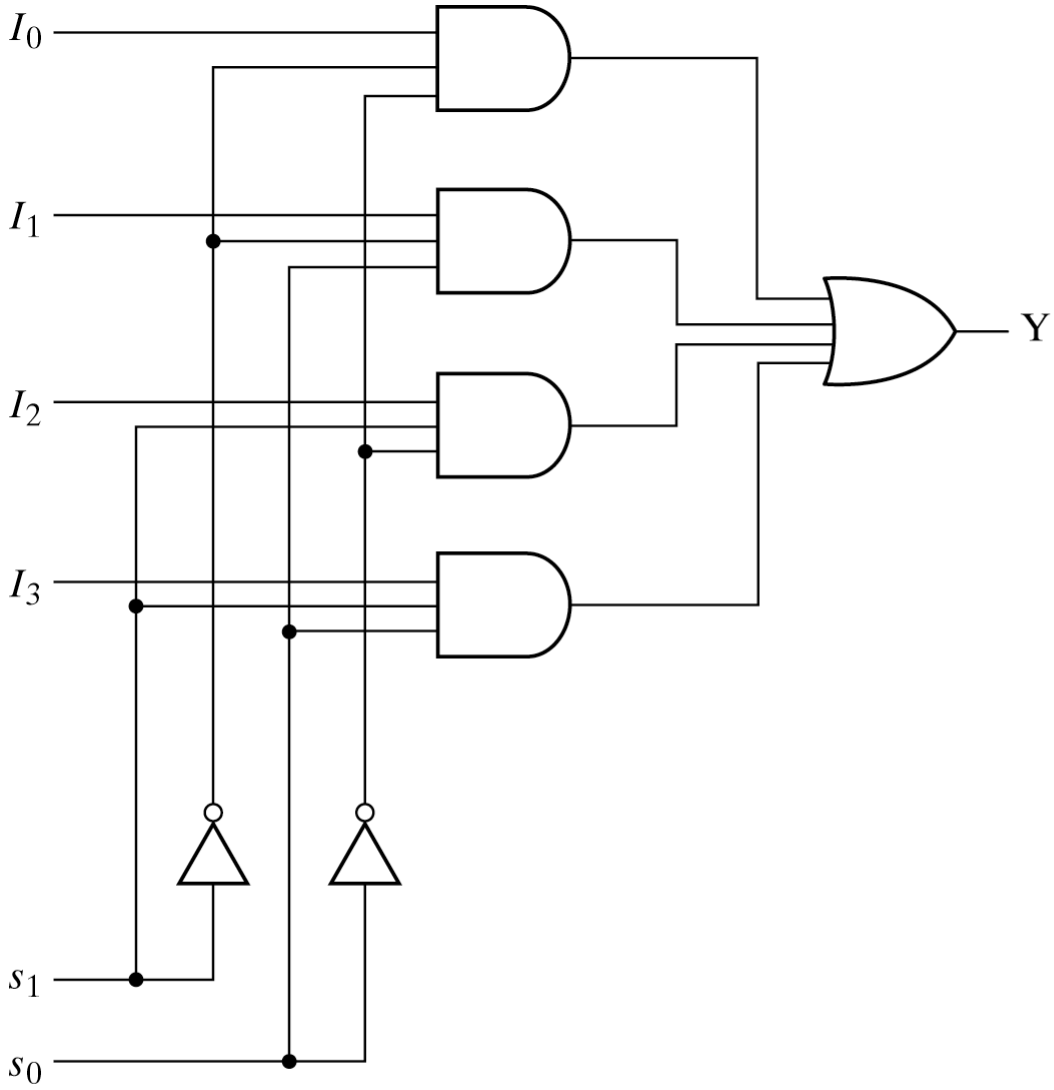
$$Y = S'I_0 + SI_1$$





4-to-1-line MUX

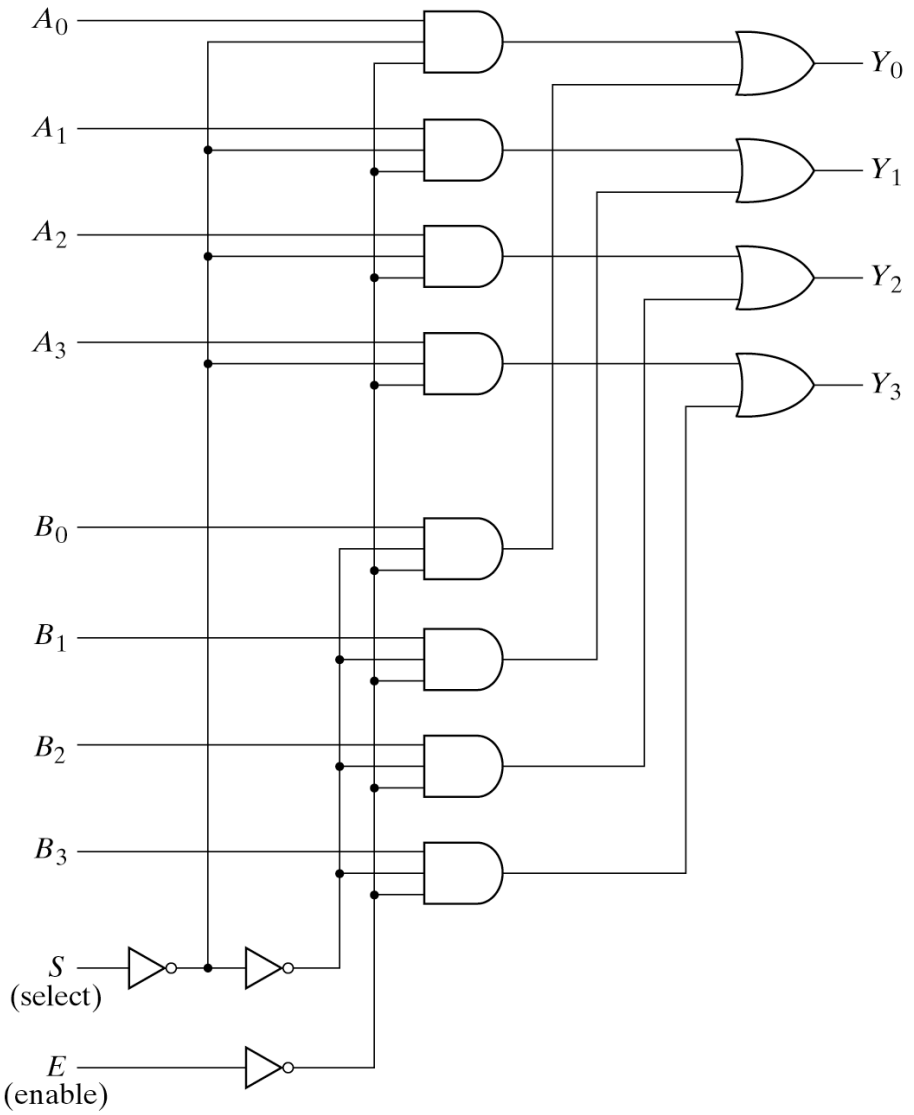
s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



Quadruple 2-to-1-line MUX

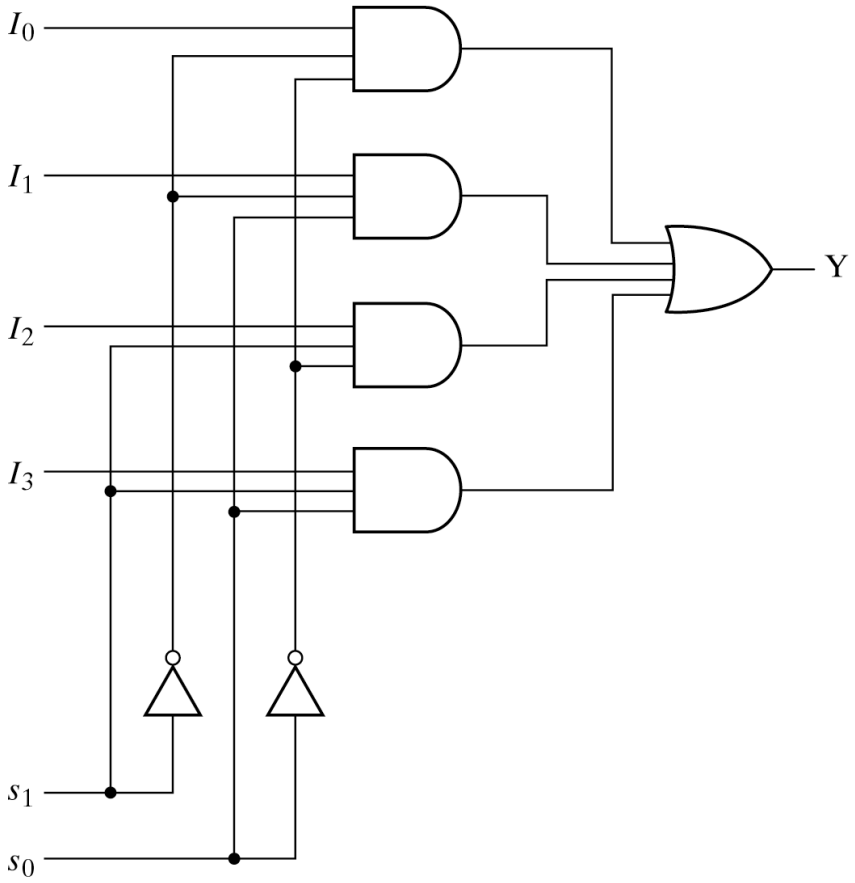
Function table

E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B



Boolean Function Implementation (1/4)

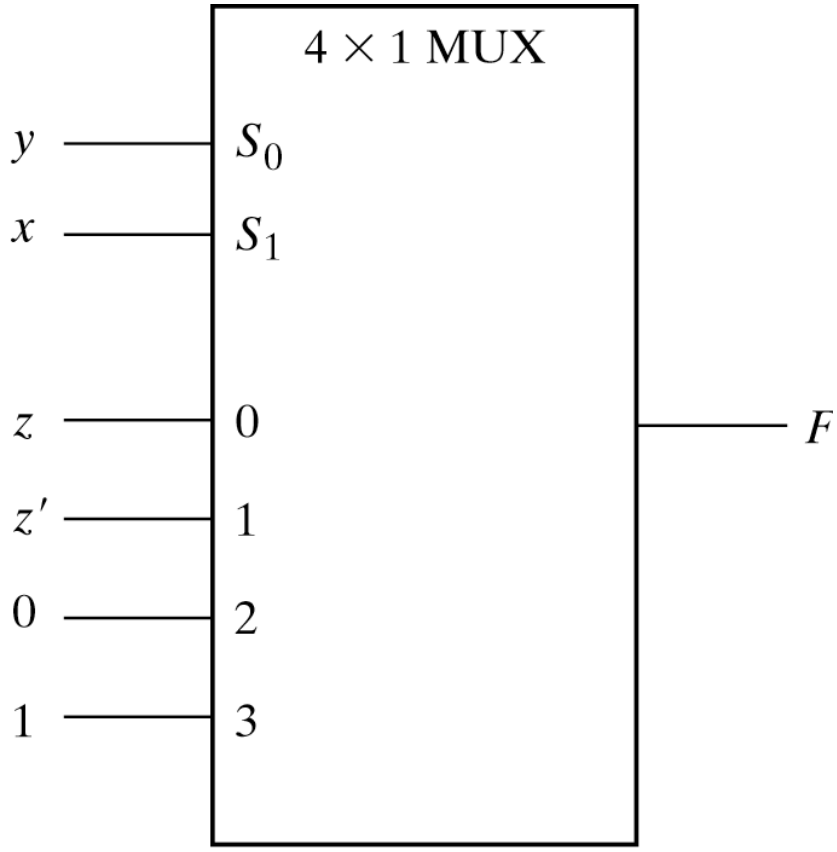
- MUX: decoder + OR gate
- 2^{n-1}-to-1 MUX can implement any Boolean function of n input variable



Boolean Function Implementation (2/4)

- Example: $F(x, y, z) = \sum(1, 2, 6, 7)$

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



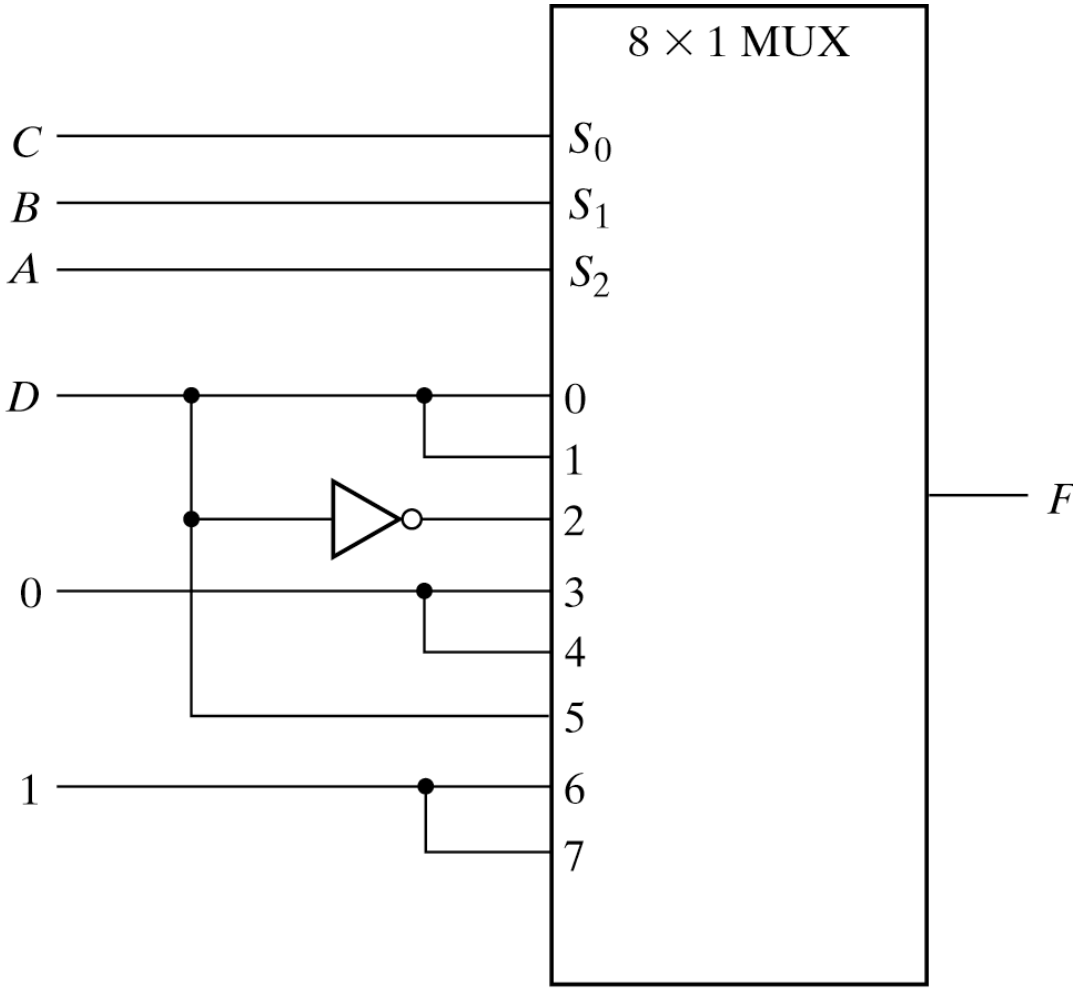
Boolean Function Implementation (3/4)

- For an *n*-variable function
- Assign an ordering sequence of the $n - 1$ input variables to the selection input of MUX
- The last (rightmost) variable will be used for the input lines
- Construct the truth table
- Consider a pair of consecutive minterms starting from m_0
- Determine the input lines according to the last variable and output signals in the truth table

Boolean Function Implementation (4/4)

- Example: $F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$

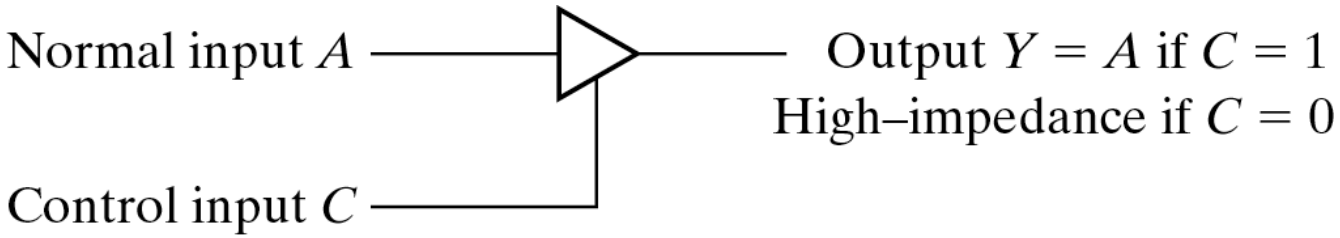
A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	



Three-State Gates

Three-State Gates (1/2)

- A multiplexer can be constructed with three-state gates
- Output states: 0, 1, and Z (high-impedance, or open circuits)



C	Y
0	Z
1	A

- Three-state gates can be used to build up a *bus*
  - A communication channel among different modules in a digital system

Three-State Gates (2/2)

- Examples: multiplexers with three-state gates