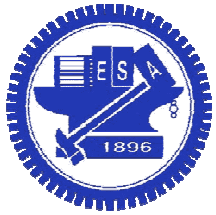
Combinational Circuits



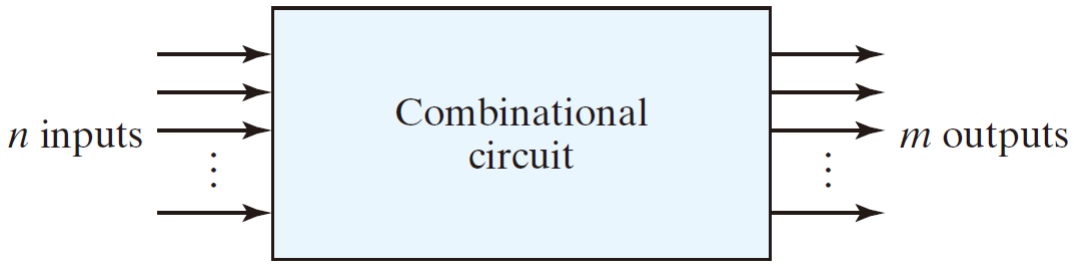
Chun-Jen Tsai
National Chiao Tung University
10/22/2012

Logic Circuits for Digital Systems

- Logic circuits for digital systems may be combinational or sequential.
- A combinational circuit consists of logic gates whose outputs at any time are determined from only the current inputs.
- A sequential circuit contains small memory elements
  - the outputs are a function of the current inputs and the state of the memory elements
  - The memory elements imply that the outputs also depend on past inputs, not just current inputs

Combinational Circuit

- A combinational circuits
  - 2^n possible combinations of input values



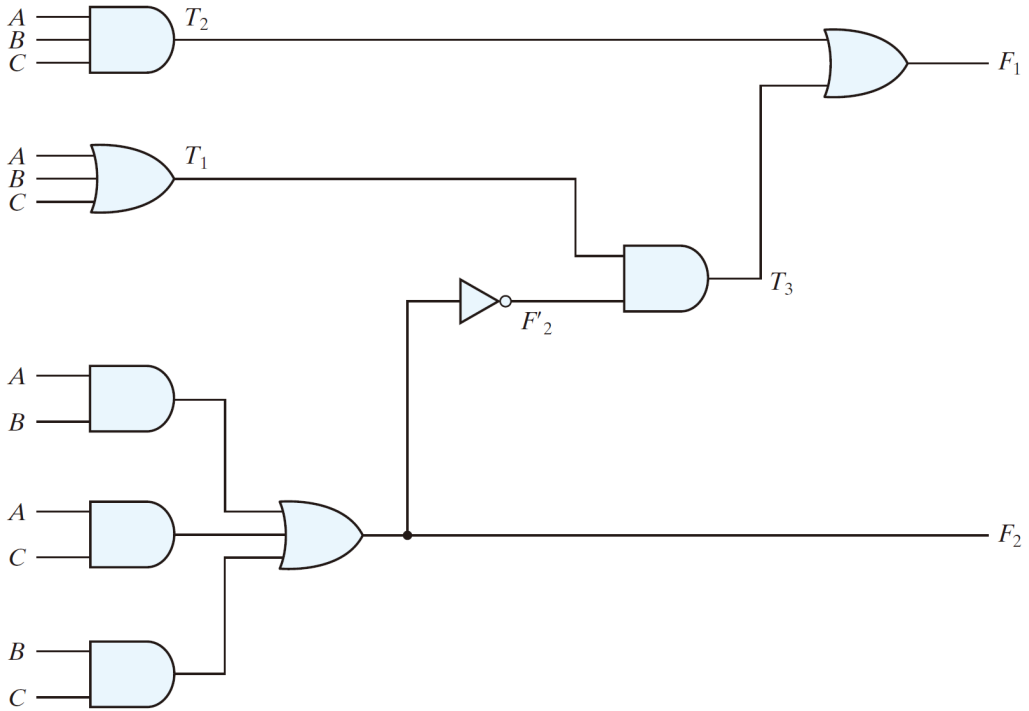
  - Specific functions
    - Adders, subtractors, comparators, decoders, encoders, and multiplexers
    - MSI circuits or standard cells

Analysis of Combinational Circuits

- In general, give a logic diagram, we must perform some analysis to understand the function it implements:
- Steps of analysis:
  - Make sure that the logic is combinational, not sequential
    - No feedback path or memory elements
  - Label inputs, intermediate gate outputs, and the final outputs
  - Derive its Boolean functions (truth table)
  - A verbal explanation of its function

Example: Analysis of a Circuit (1/2)

- The following logic diagram has three inputs, A, B, C, and two outputs:
  - $F_2 = AB+AC+BC$
  - $T_1 = A+B+C$
  - $T_2 = ABC$
  - $T_3 = F_2'T_1$
  - $F_1 = T_3+T_2$



Example: Analysis of a Circuit (2/2)

- Put the function in canonical form:
  - $F_1 = T_3 + T_2 = F_2' T_1 + ABC$
    $= (AB + AC + BC)'(A + B + C) + ABC$
    $= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC$
    $= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC$
    $= A'BC' + A'B'C + AB'C' + ABC$

- The function is a full adder
  - F_1 is the sum
  - F_2 is the carry

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 of Combinational Circuits

- The design procedure of combinational circuits
  - State the function behavior (system specification)
  - Determine the inputs and outputs
    - The input and output variables are assigned symbols
  - Derive the truth table
  - Derive the simplified Boolean functions
  - Draw the logic diagram and verify the correctness

Design Implementation

- Today, Electronic Design Automation (EDA) tools are used for design implementation.
  - EDA tools take functional description (expressed in HDL codes or schematic diagrams) and design constraints as input, and generate a “netlist” as output.
- Considerations for Logic minimization:
  - Number of gates
  - Number of inputs to a gate
  - Propagation delay
  - Number of interconnection
  - Limitations of the driving capabilities

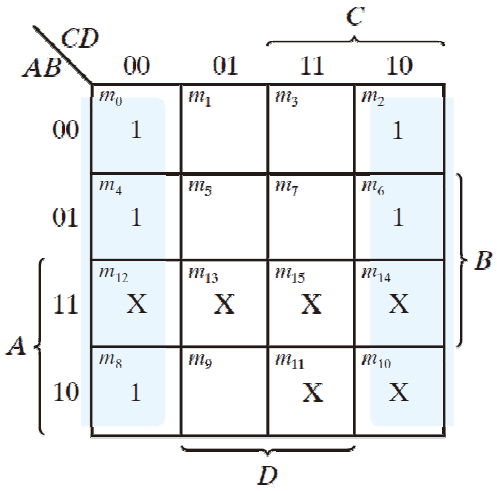
Code Conversion (1/4)

- Design a circuit to convert from BCD to excess-3 codes
- System specification:

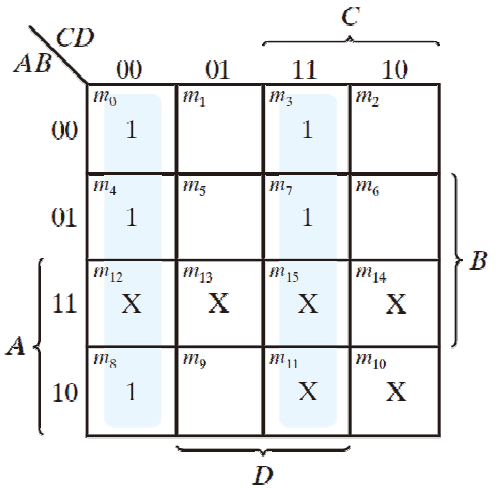
Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Code Conversion (2/4)

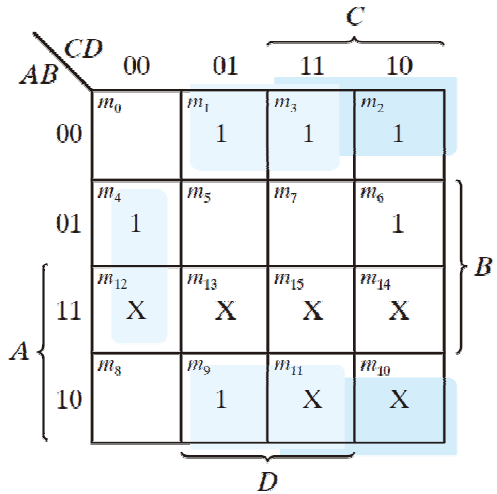
- K-maps:



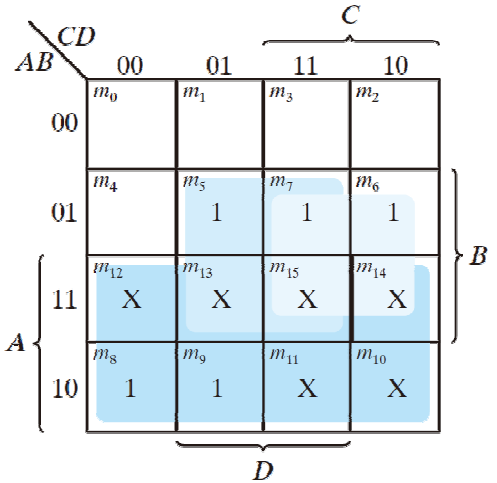
$z = D'$



$y = CD + C'D'$



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



$w = A + BC + BD$

Code Conversion (3/4)

- The simplified functions
  - $z = D'$

    $y = CD + C'D'$

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

    $w = A + BC + BD$
- Another implementation
  - $z = D'$

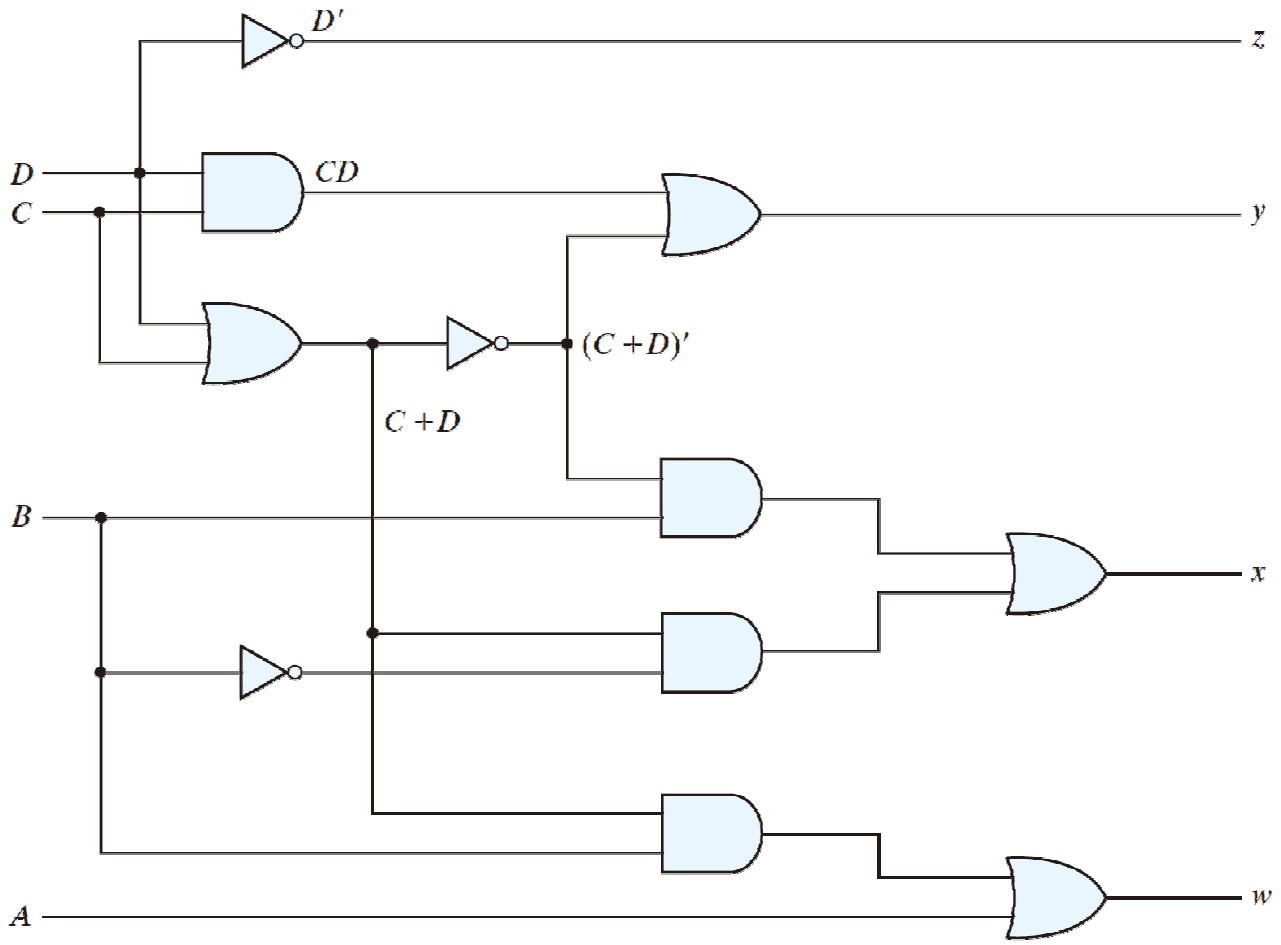
    $y = CD + C'D' = CD + (C + D)'$

    $x = B'C + B'D + BC'D' = B'(C + D) + B(C + D)'$

    $w = A + BC + BD$

Code Conversion (4/4)

- The logic diagram:



One-bit Half-Adder (1/2)

- The behavior of a 1-bit half-adder is as follows:
  - Input variables: x and y
  - Output variables: Sum (S) and carry (C)
  - Truth table:

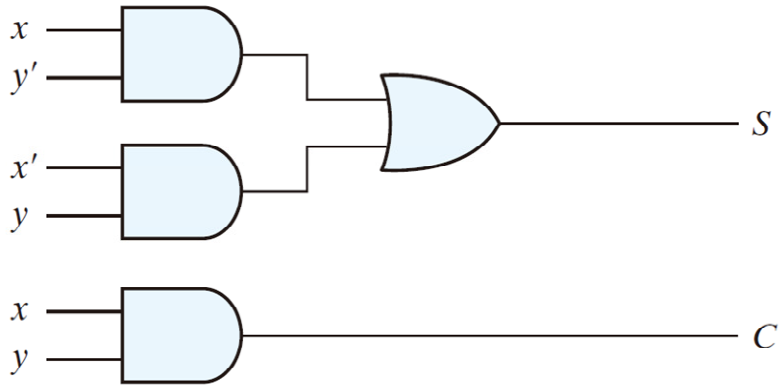
Half Adder

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

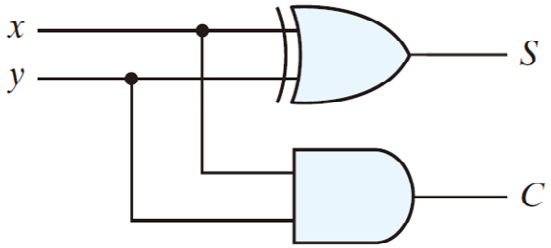
  - Boolean functions:
    - $S = x'y + xy' = x \oplus y$
    - $C = xy$

One-bit Half-Adder (2/2)

- Gate-level implementation:



(a) $S = xy' + x'y$
$C = xy$



(b) $S = x \oplus y$
$C = xy$

- Alternative implementations:
  - $S = (x + y)(x' + y')$
  - $S' = xy + x'y' \rightarrow S = (C + x'y')'$
  - $C = xy = (x' + y')'$

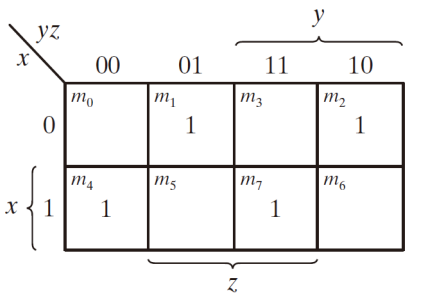
One-bit Full-Adder (1/3)

- The behavior of a 1-bit full-adder is as follows:
  - Input variables: x , y, and z (carry from previous bit)
  - Output variables: Sum (S) and carry (C)
  - Truth table:

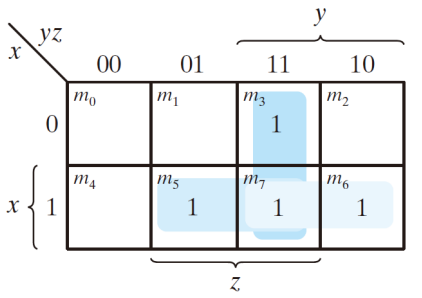
Full Adder

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

  - K-maps:



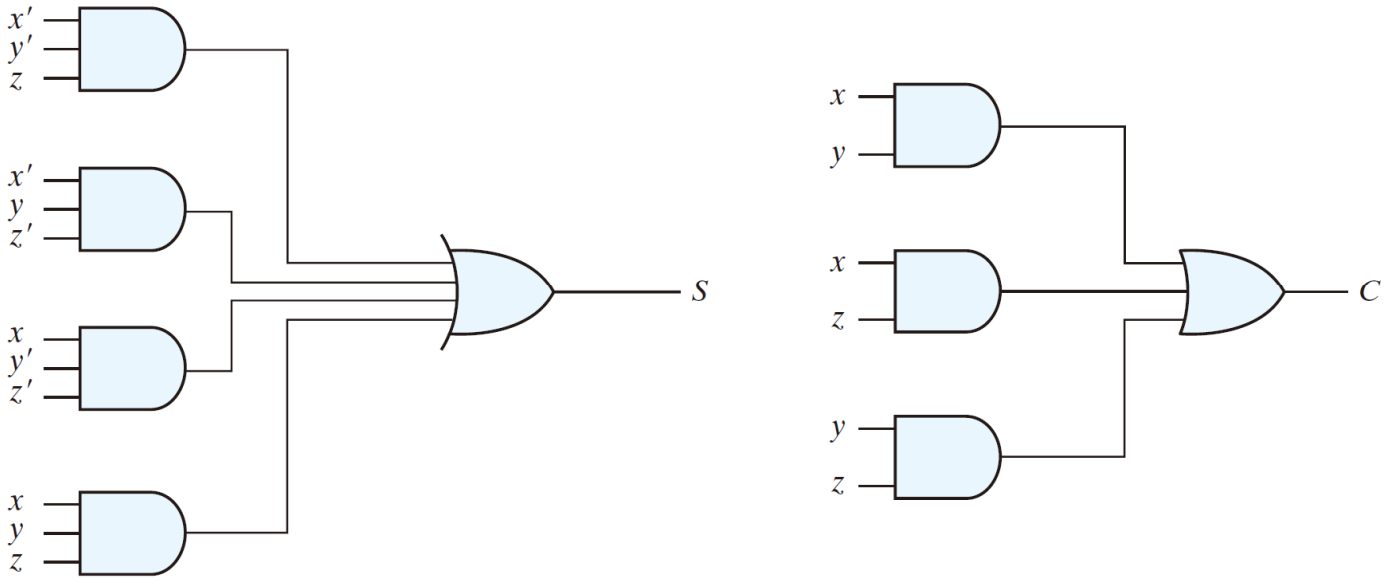
(a) $S = x'y'z + x'yz' + xy'z' + xyz$



(b) $C = xy + xz + yz$

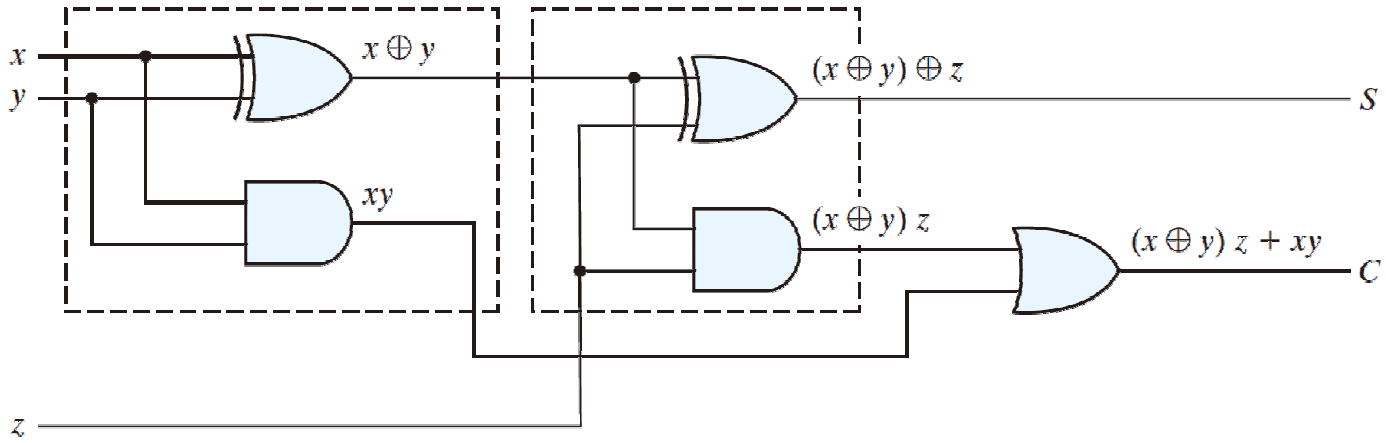
One-bit Full-Adder (2/3)

- Gate-level implementation:
  - $S = x'y'z + x'yz' + xy'z' + xyz$
  - $C = xy + xz + yz$



One-bit Full-Adder (3/3)

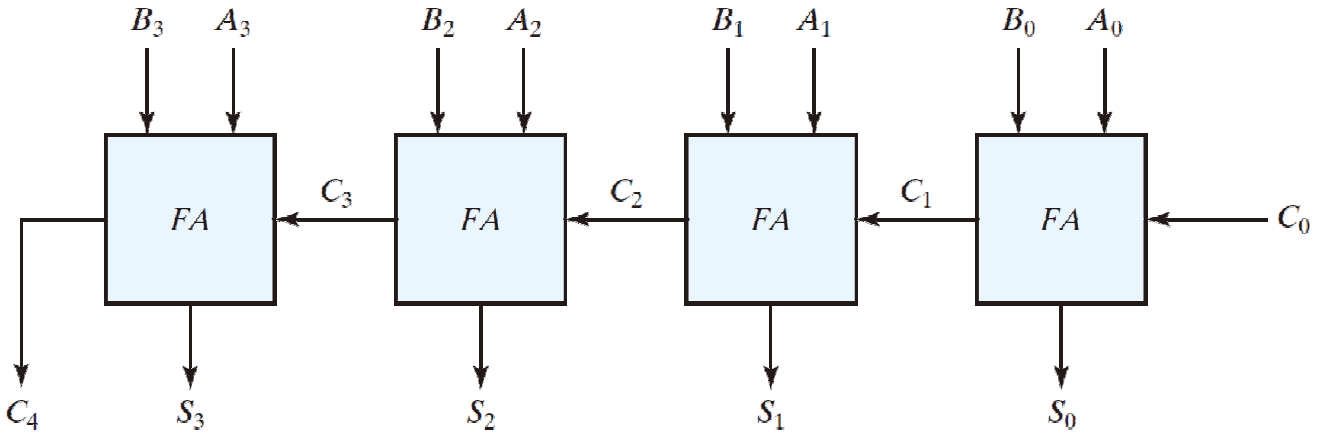
- We can use two 1-bit half-adders to construct a 1-bit full adder:
  - $S = (x'yz' + xy'z') + (x'y'z + xyz) = (x \oplus y)z' + (x \oplus y)'z = (x \oplus y) \oplus z$
  - $C = xy + xz + yz = xy + (x + y)z = xy + (x \oplus y)z$



Binary Adder

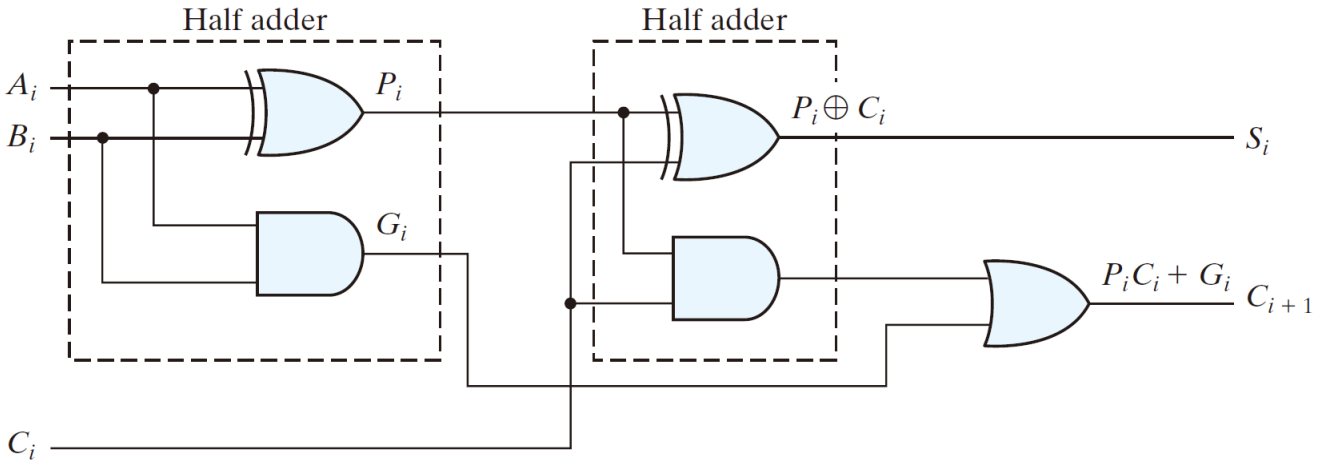
- The behavior of a 4-bit binary adder

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



Carry Propagation of Adders

- The carry propagation problem:
  - when the correct outputs are available
  - the critical path counts (the worst case)
  - $(A_1, B_1, C_1) > C_2 > C_3 > C_4 > (C_5, S_4)$
  - The propagation delay is 8 gate levels for a 4-bit adder

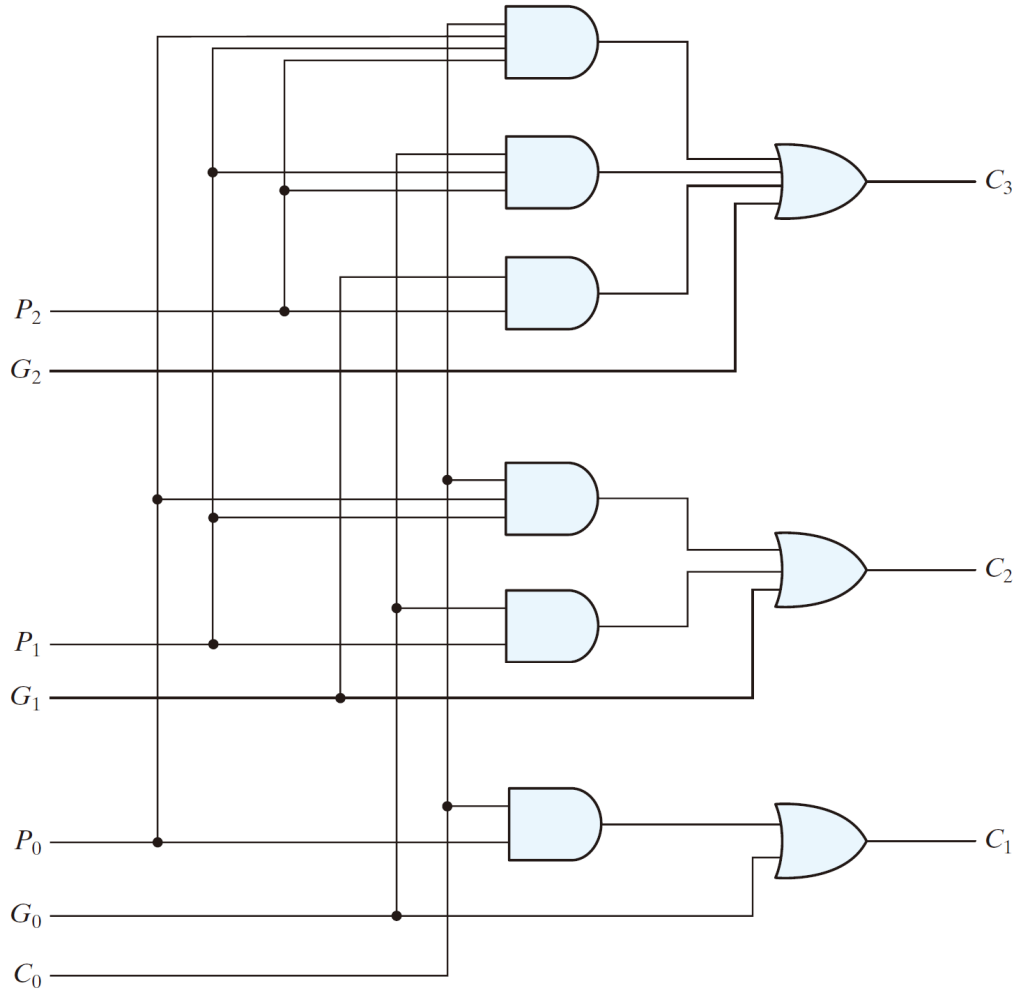


Reduction of Propagation Delay

- To reduce the carry propagation delay, we can
  - Employ faster gates
  - Use look-ahead carry logic (more complex)
- Look-ahead carry generation:
  - Define two new variables:
    - Carry propagate $P_i = A_i \oplus B_i$
    - Carry generate $G_i = A_i B_i$
  - Now, sum $S_i = P_i \oplus C_i$
  - Carry: $C_{i+1} = G_i + P_i C_i$
    - $C_1 = G_0 + P_0 C_0$
    - $C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_0$
    - $C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

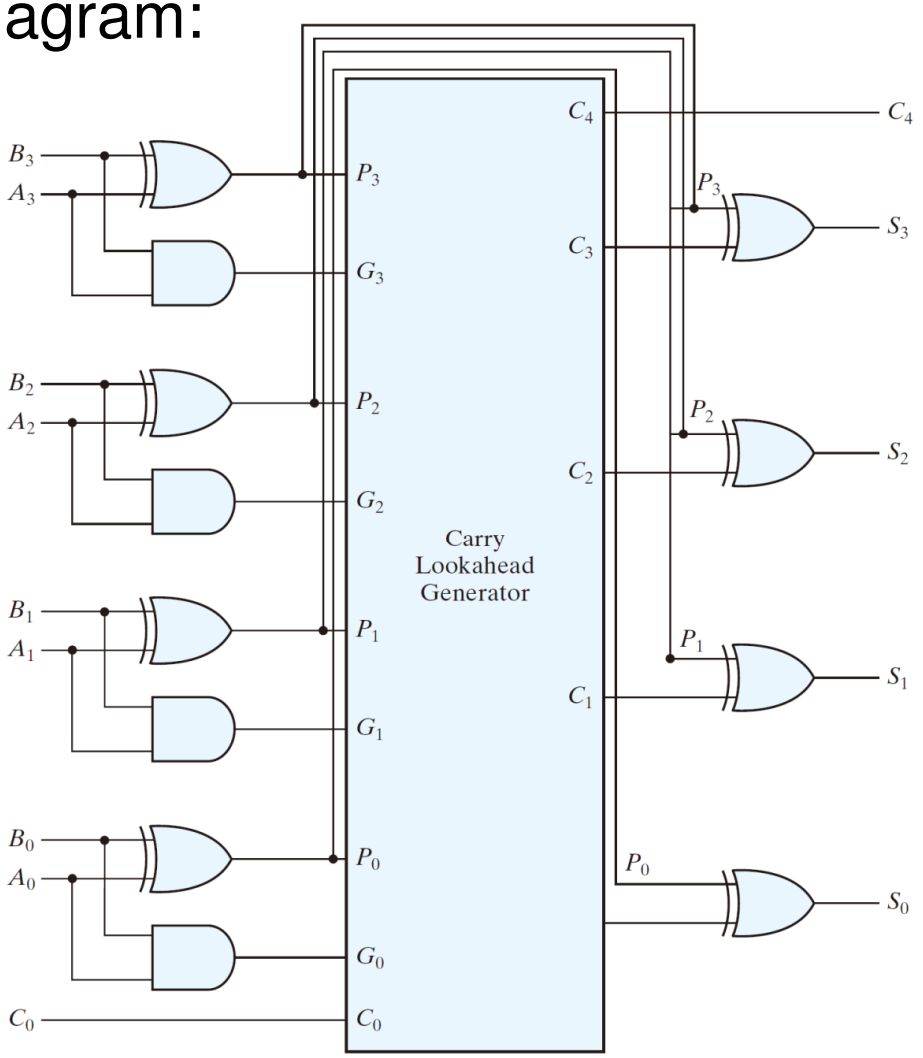
Look-Ahead Carry Generator

- Logic diagram:



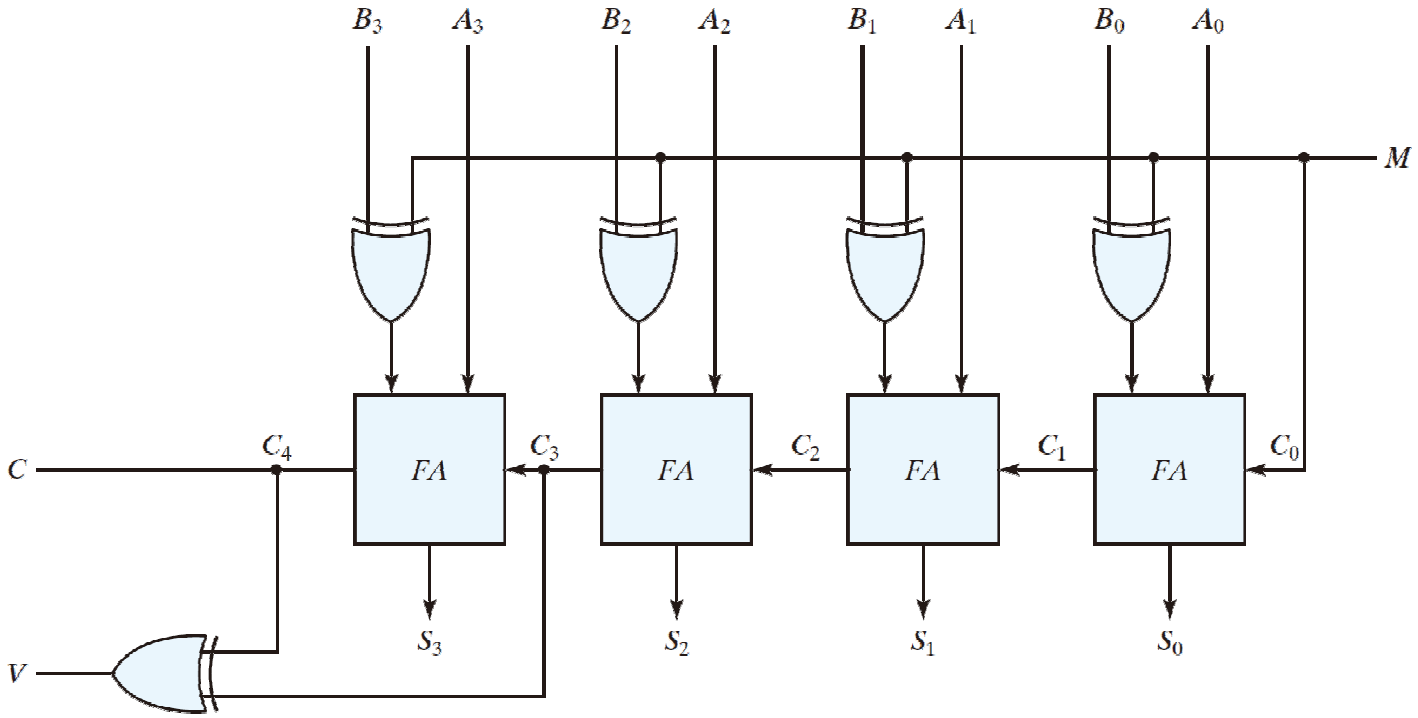
Carry-Look Ahead Adder

- Three-level logic diagram:
  - $P_i = A_i \oplus B_i$
  - Carry generation
  - $S_i = P_i \oplus C_i$



Binary Subtractor (1/2)

- Logic behavior:
  - $A - B = A +$ (2's complement of B)
- Boolean expression:
  - If $M = 0$, $A + B$; if $M = 1$, $A + B' + 1$



Binary Subtractor (2/2)

- Since our adder has finite bits, we must perform overflow detection:
  - Add two positive numbers and obtain a negative number
  - Add two negative numbers and obtain a positive number
- Overflow detection rule:
  - If $V = 0$, no overflow; $V = 1$, overflow
- Examples:

carries:	0 1	carries:	1 0
+70	0 1000110	−70	1 0111010
+80	0 1010000	−80	1 0110000
+150	1 0010110	−150	0 1101010

Decimal Adder (1/3)

- A BCD adder adds two BCD numbers:
  - 9 inputs: two BCD codes and one carry-in
  - 5 outputs: one BCD code and one carry-out
- The truth table has 2^9 entries, too large for efficient implementation
- Question: can we use binary full adders to implement BCD adders?
  - The sum of any two digits $\leq 9 + 9 + 1 = 19$
  - After binary addition, we perform binary to BCD conversion

Decimal Adder (2/3)

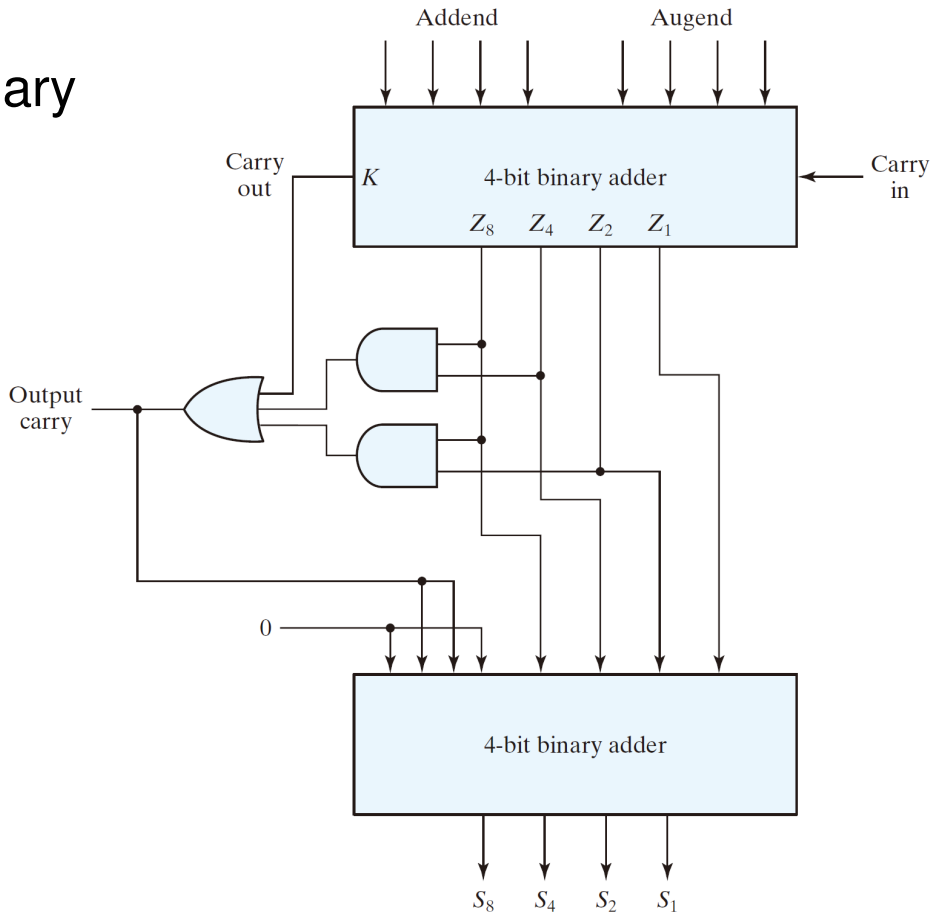
- Binary-to-BCD conversion of numbers ≤ 19

Derivation of BCD Adder

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)

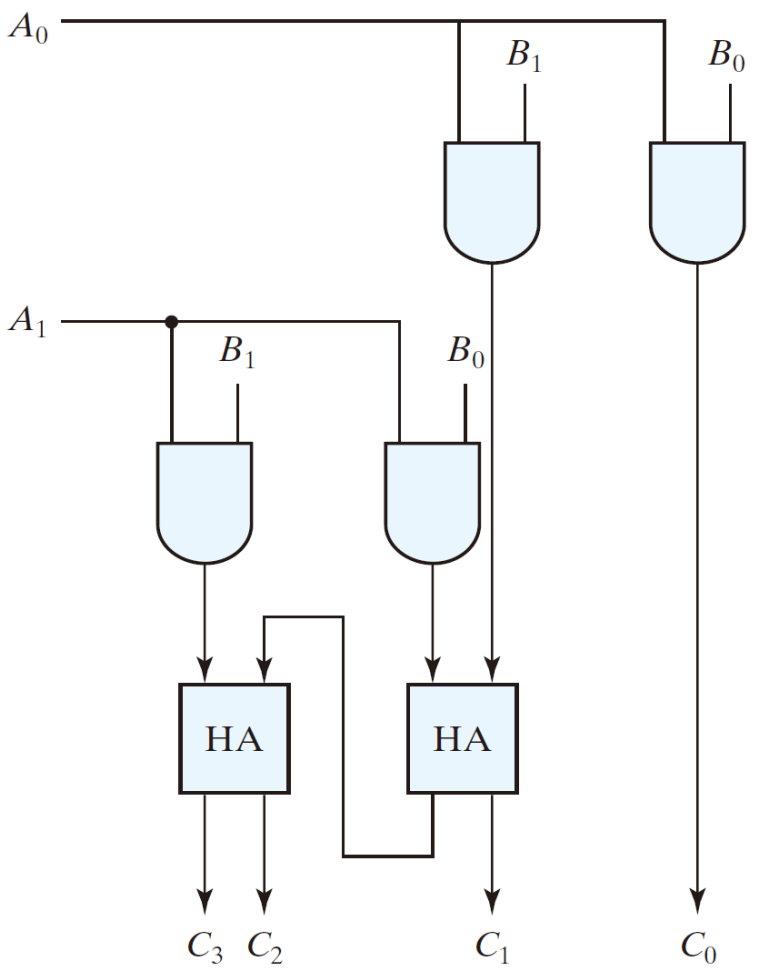
- Conversion is needed if the sum > 9
  - The carry of BCD sum $C = 1$ if $K = 1$ or $Z_8Z_4 = 1$ or $Z_8Z_2 = 1$
    $\rightarrow C = K + Z_8Z_4 + Z_8Z_2$
  - If $C = 1$, modify the binary sum by $-(10)_d$ or +6
- Block diagram:



Binary Multiplier (1/2)

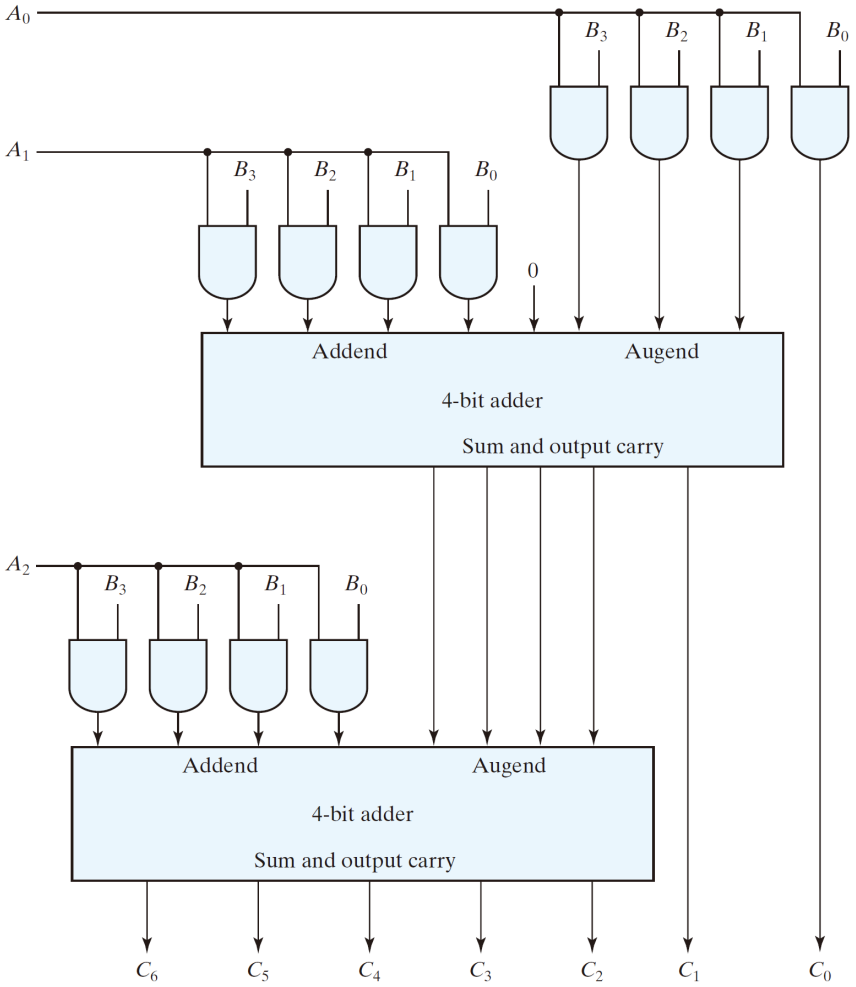
- Two-bit multiplier for $(B_1B_0)\times(A_1A_0)$:
  - $C_0 = A_0B_0$
  - $C_1 = A_0B_1 \oplus A_1B_0$
  - $C_2 = A_1B_1 \oplus (A_0B_1)(A_1B_0)$
  - $C_3 = (A_1B_1)(A_0B_1)(A_1B_0)$

		B_1	B_0
		A_1	A_0
		A_0B_1	A_0B_0
	A_1B_1	A_1B_0	
C_3	C_2	C_1	C_0



Binary Multiplier (2/2)

- 4-bit by 3-bit multiplier for $(B_3B_2B_1B_0)\times(A_2A_1A_0)$:



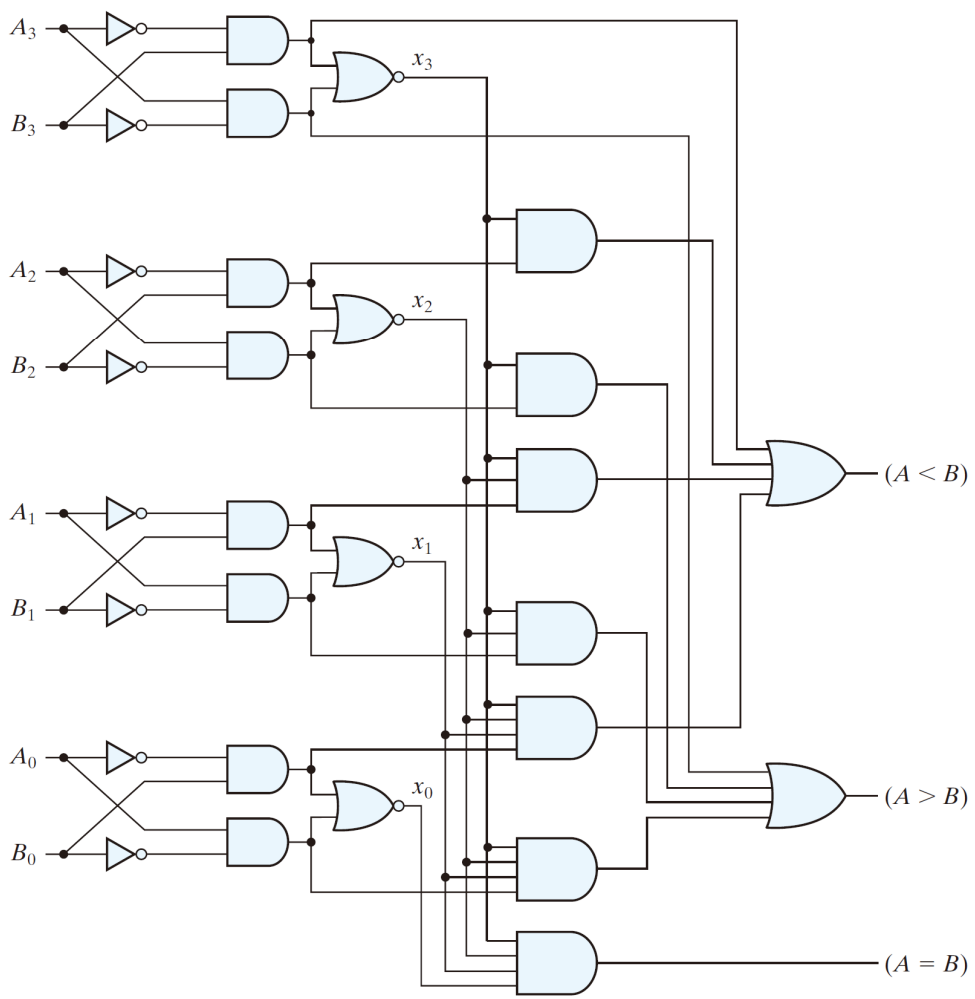
Magnitude Comparator (1/2)

- Using the truth table to implement an n-bit number comparator requires 2^{2n} entries – too large.
- Functions of comparing

  $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$
  - $F_{A=B} = x_3x_2x_1x_0$, where $x_i = (A_i'B_i + A_iB_i')'$, for $i = 0, 1, 2, 3$.
  - $F_{A>B} = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$
  - $F_{A<B} = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$

Magnitude Comparator (2/2)

- Logic diagram:



Decoder (1/2)

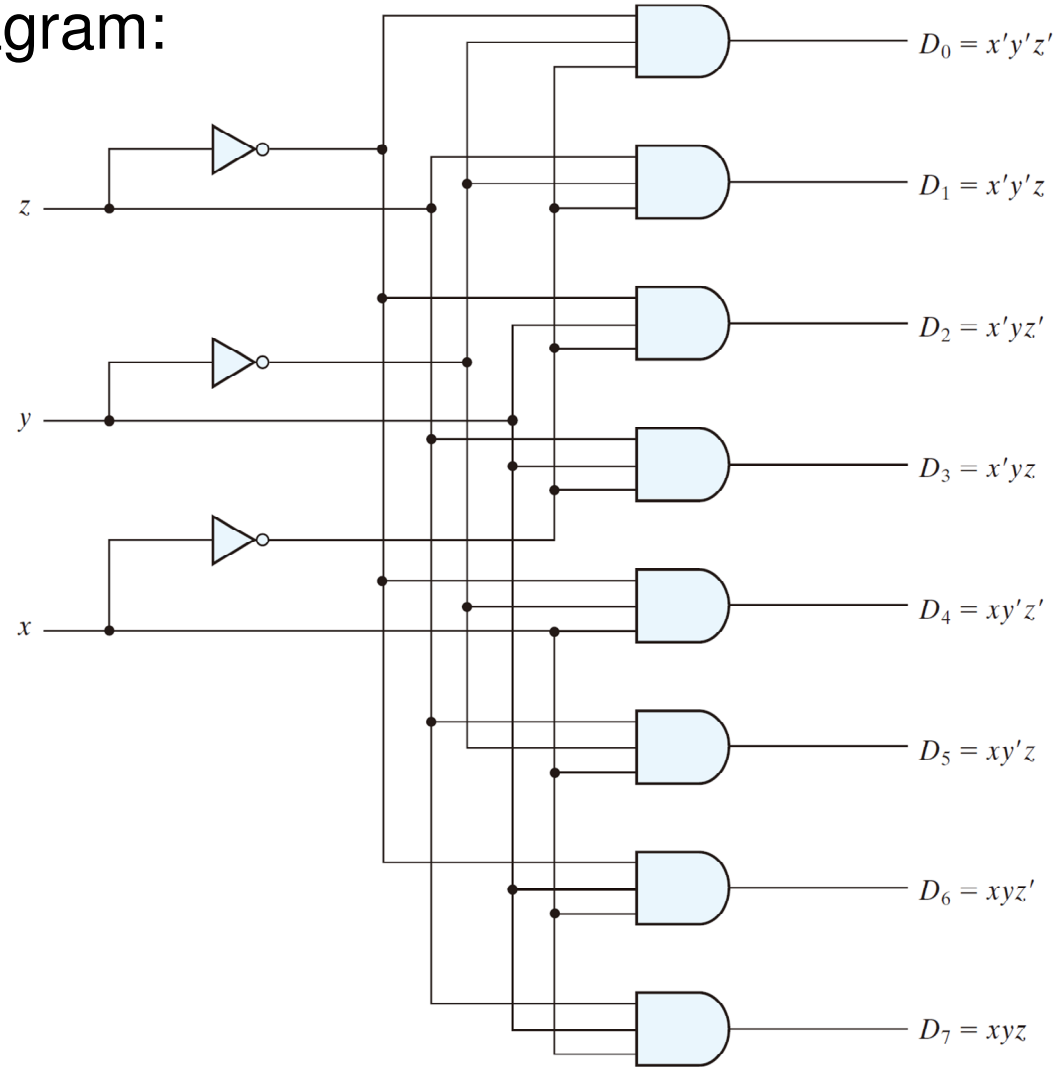
- An n-to-m decoder selects a particular output line give an input binary codeword:
  - an input binary code of n bits has 2^n possible output lines
  - only one output can be active (high) at any time
- Example: a 3-to-8 decoder

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

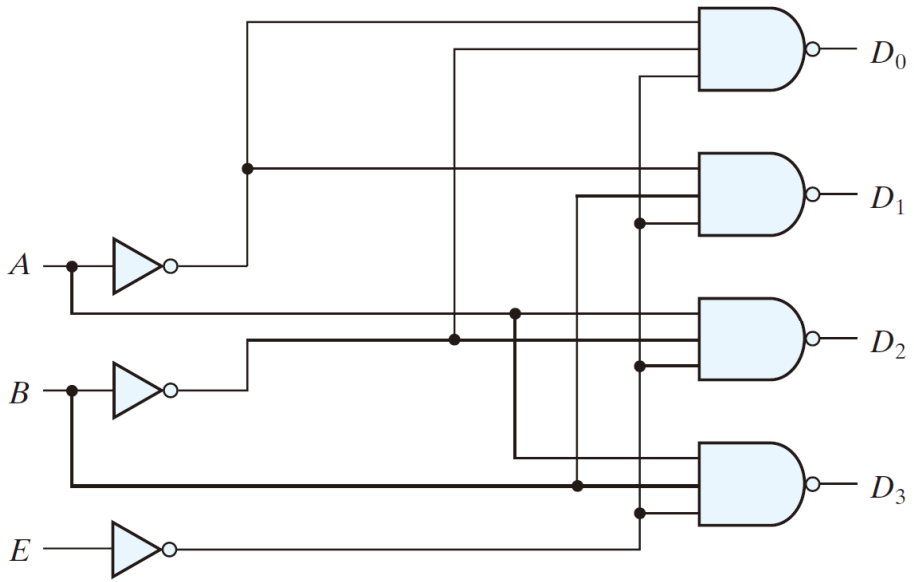
Decoder (2/2)

- Logic diagram:



Decoder Implementation with NAND

- If NAND gates are used to implement a decoder, it is more efficient to use “active-low” logic
- Example: a 2-to-4 decoder with an enable input



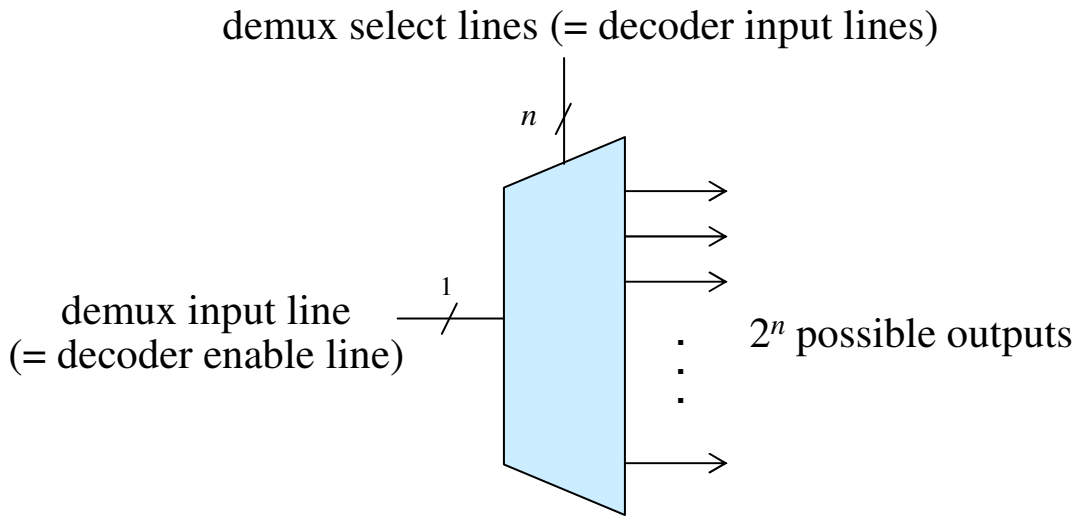
(a) Logic diagram

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	1

(b) Truth table

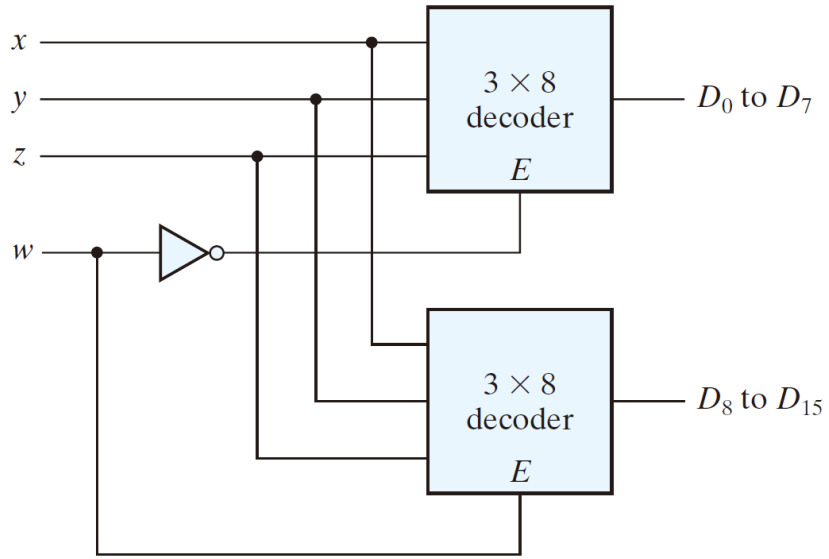
Demultiplexers

- A demultiplexor (demux) is a decoder with an enable input line



Decoder Extension

- Large decoders can be constructed using smaller decoders with enable inputs.
- A 4×16 decoder constructed with two 3×8 decoders:

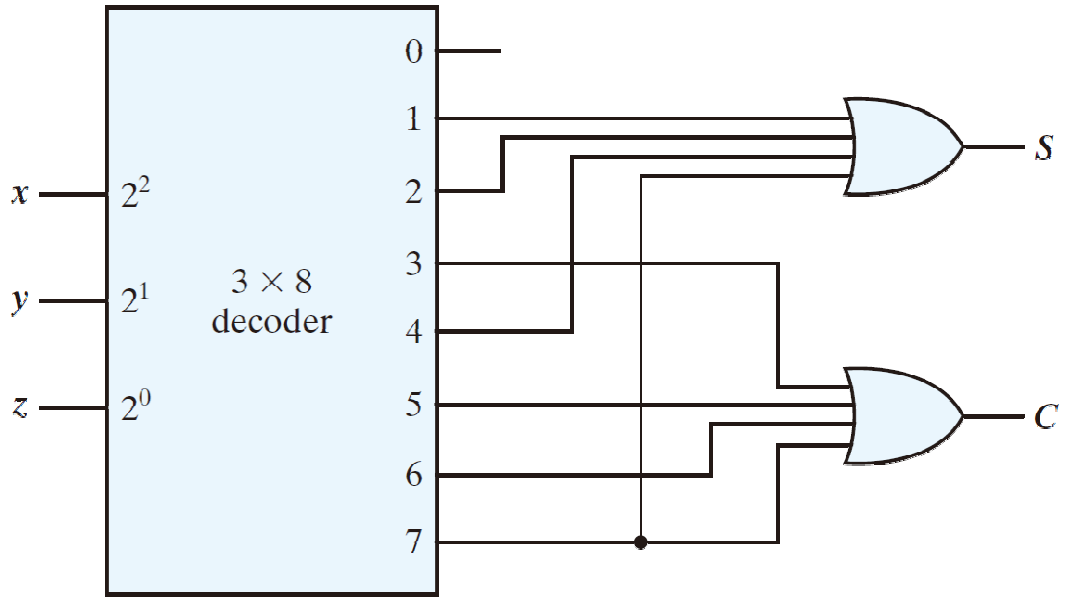


Combinational Logic Implementation

- A decoder can implement any combinational logics
  - Each output of the decoder represents a selected minterm
  - A function is implemented by summing all its minterms using an OR gate at the output of the decoder
  - For an n–input, m–output function, we need an $n \times 2^n$ deocder and m OR gates (each one has multiple inputs that corresponds to the number of minterms for each outputs)
- To reduce the number of inputs to OR gates, we can use NOR to combine the minterms of F'
  - If there are k minterms in F, there are $2^n - k$ minterms in F'.
- In general, it is not a practical implementation

Example: A Full Adder

- Implementing a 1-bit full adder using a decoder:
  - $S(x, y, z)=\Sigma(1, 2, 4, 7)$
  - $C(x, y, z)= \Sigma(3, 5, 6, 7)$



Encoder

- Encoder is the inverse function of a decoder
- Example: 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

- Boolean functions:

$$X = D_4+D_5+D_6+D_7,\ Y = D_2+D_3+D_6+D_7,\ Z = D_1+D_3+D_5+D_7$$

Priority Encoder (1/3)

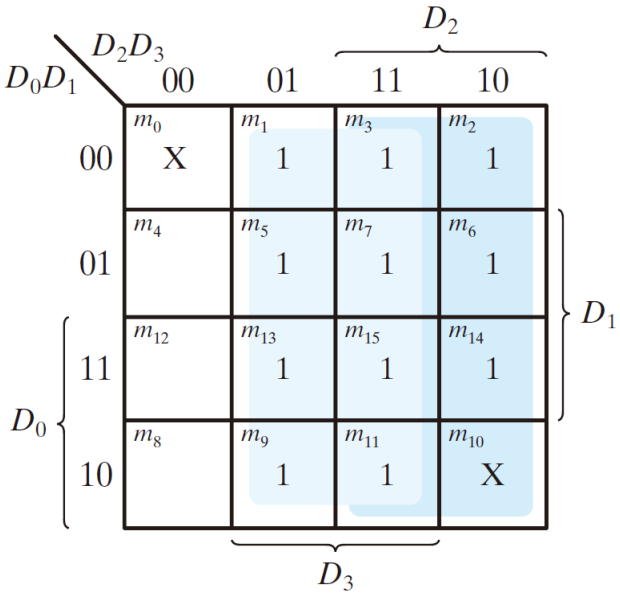
- A priority encoder is an encoder that applies some priority policy to generate the output when multiple inputs are active simultaneously
- Example:

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

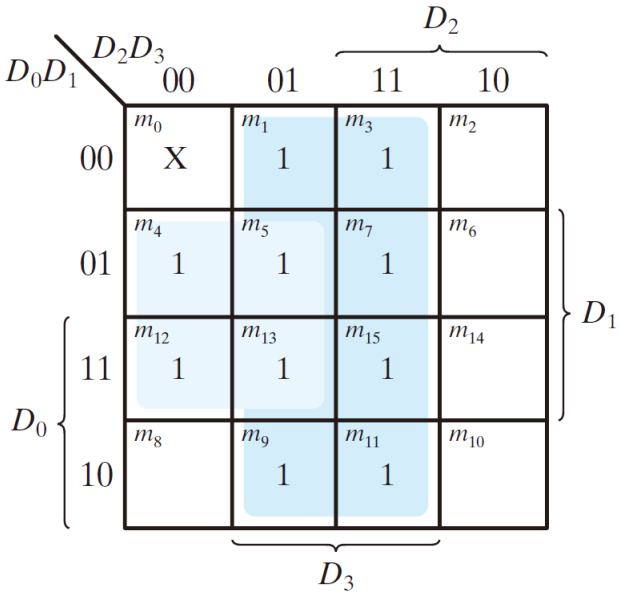
  - D_3 has the highest priority
  - D_0 has the lowest priority
  - X: don't-care conditions
  - V: valid output indicator

Priority Encoder (2/3)

- The K-maps of the priority encoder:



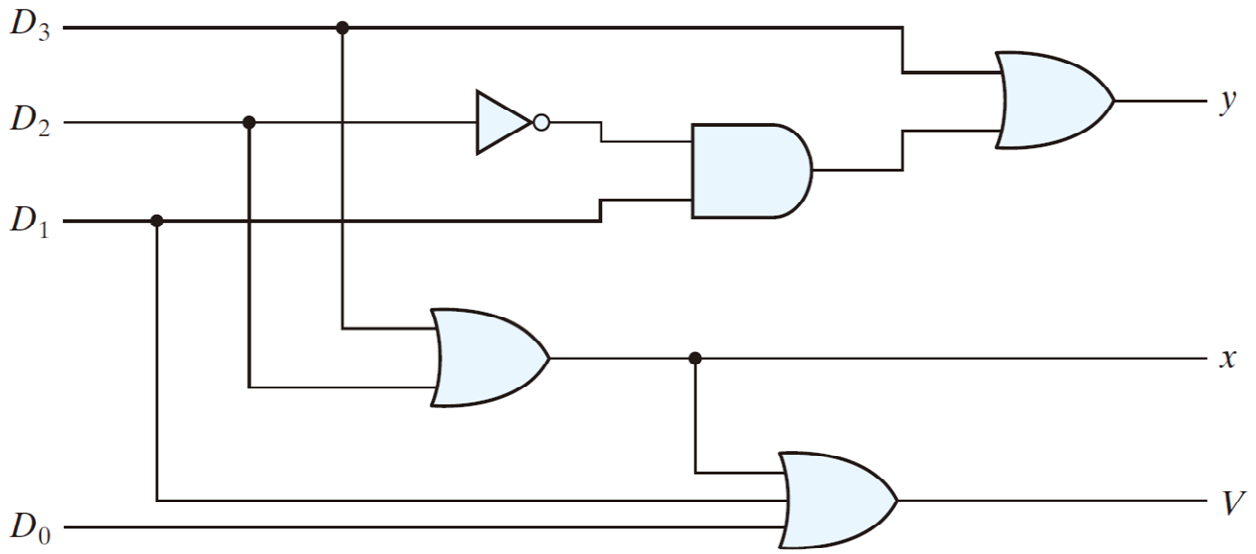
$x = D_2 + D_3$



$y = D_3 + D_1D'_2$

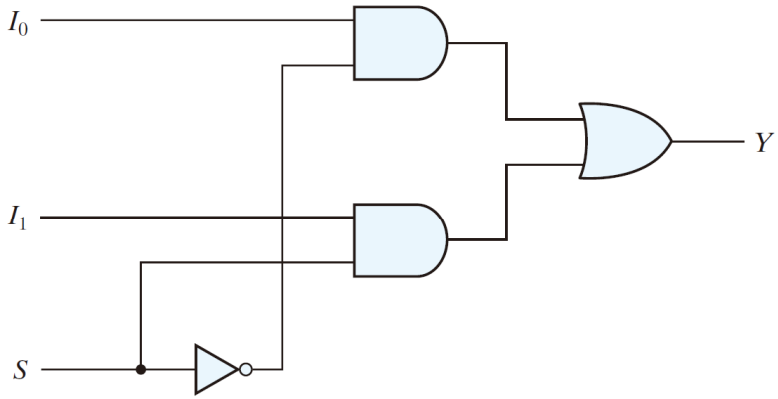
Priority Encoder (3/3)

- Logic diagram:
  - $x = D_2 + D_3$
  - $y = D_3 + D_1 D_2'$
  - $V = D_0 + D_1 + D_2 + D_3$

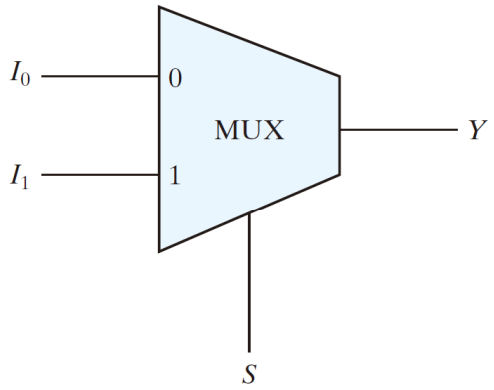


Multiplexers

- A multiplexor (mux) selects binary information from one of input lines and direct it to a single output line
- There are 2^n input lines, n selection lines and one output line
- Example: 2-to-1-line multiplexer



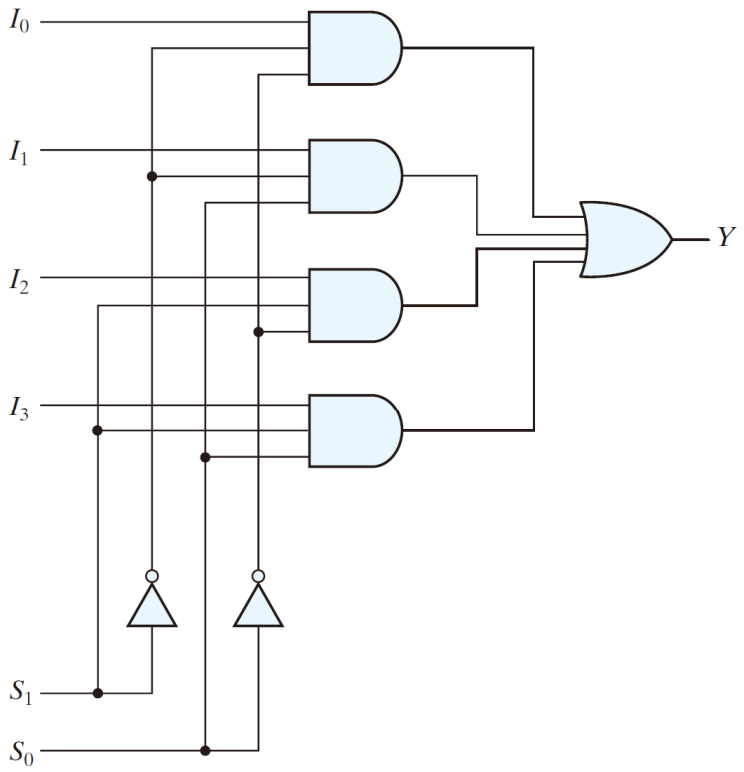
(a) Logic diagram



(b) Block diagram

4-to-1-line Multiplexer

- Logic diagram of a 4×1 multiplexor:



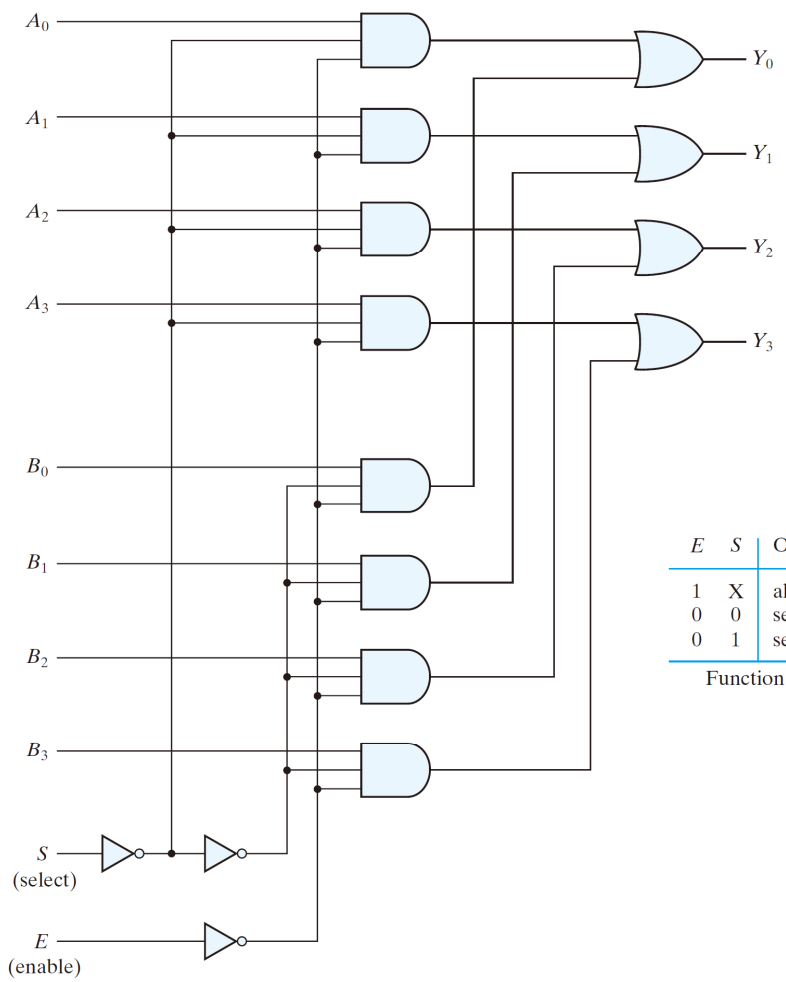
(a) Logic diagram

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

Multiplexor Extension

- n-bit signal selection logic can be constructed using n 1-bit multiplexors:



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

Function table

Multiplexors and Decoders

- Note that a 2^n-to-1 multiplexor is equivalent to an n-to-2^n decoder plus an OR gate
  - The decoder decodes the n select lines and determines which input line of the multiplexor will be active
  - An additional OR gate that combines all the output lines of the decoder generate the 1-bit output of the multiplexor
- Similar to decoders, multiplexors can be used to implement combinational circuits as well

Boolean Function Implementation

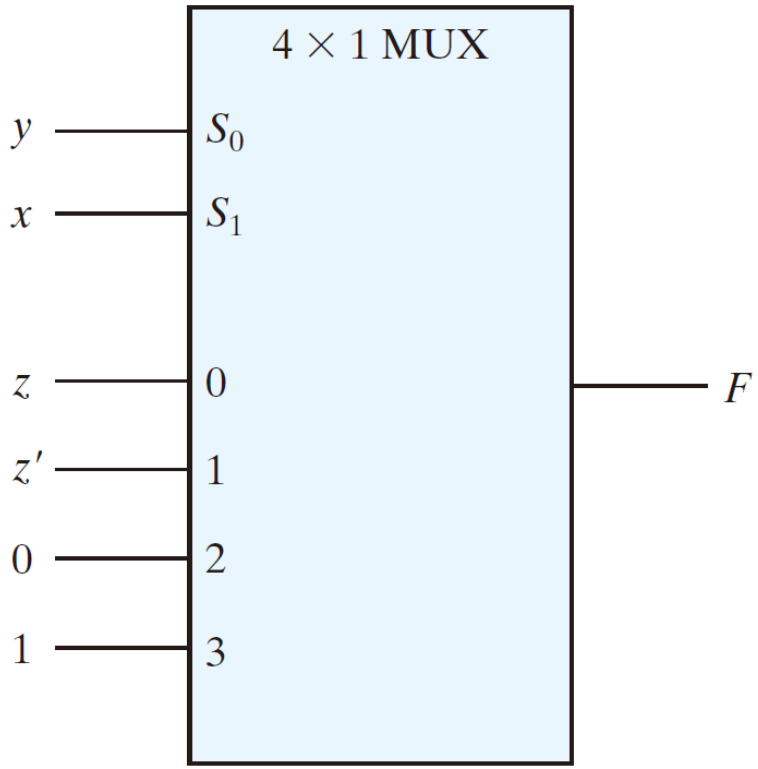
- 2^n-to-1 mux can implement any Boolean function of n input variable
  - Function inputs connects to the select lines of the mux and the minterms of the function maps to 1's in the input lines
  - Inefficient
- A better solution is to implement the Boolean function using 2^{n-1}-to-1 mux
  - $n-1$ of these variables connects to the selection lines
  - the remaining variable (and its complement) connect to the inputs

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

- Implement the function with a 4-to-1 multiplexor:

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table

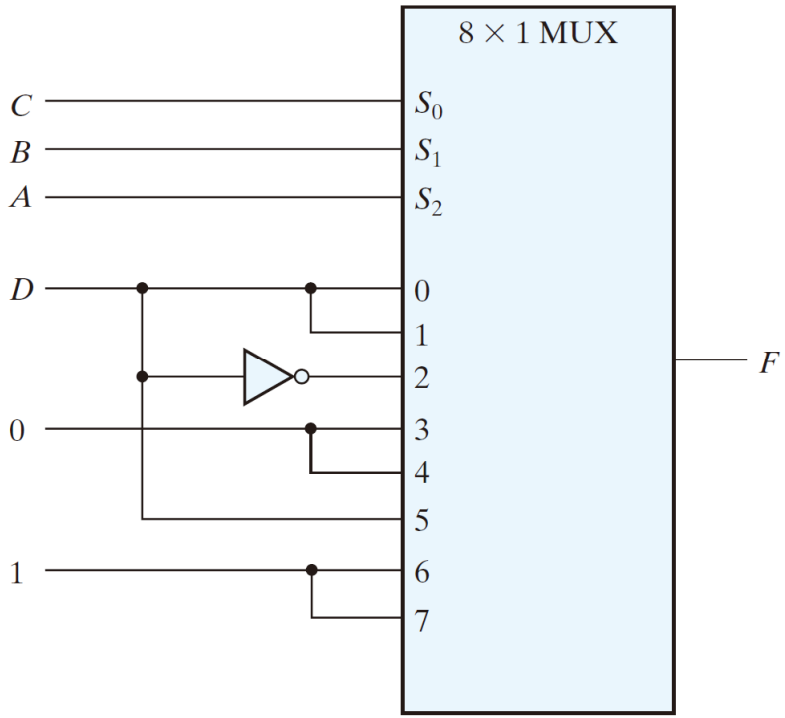


(b) Multiplexer implementation

Example: 4-input Function

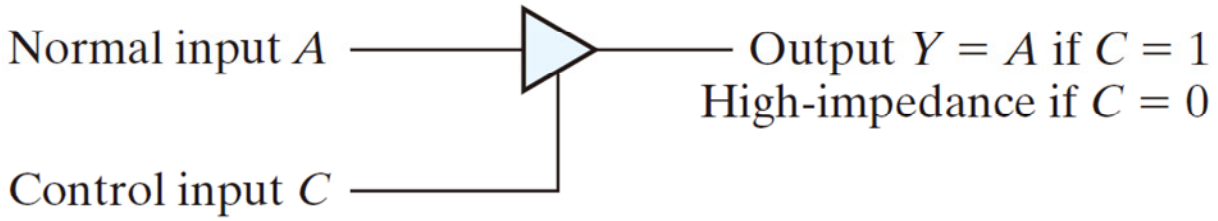
- $F(A, B, C, D) = \Sigma(1, 3, 4, 11, 12, 13, 14, 15)$

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	



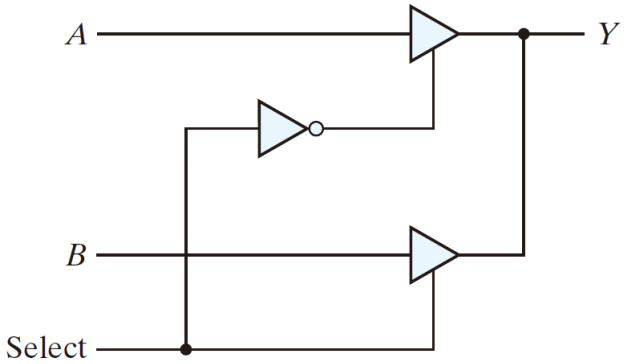
Three-State Gates

- A multiplexor can be implemented using a tri-state gate (namely, a tri-state buffer)
- The output states of a tri-state gate are: 0, 1, and high-impedance
  - A high-impedance wire is an “open circuit” wire that will not affect the signal level of the wire it connects to.

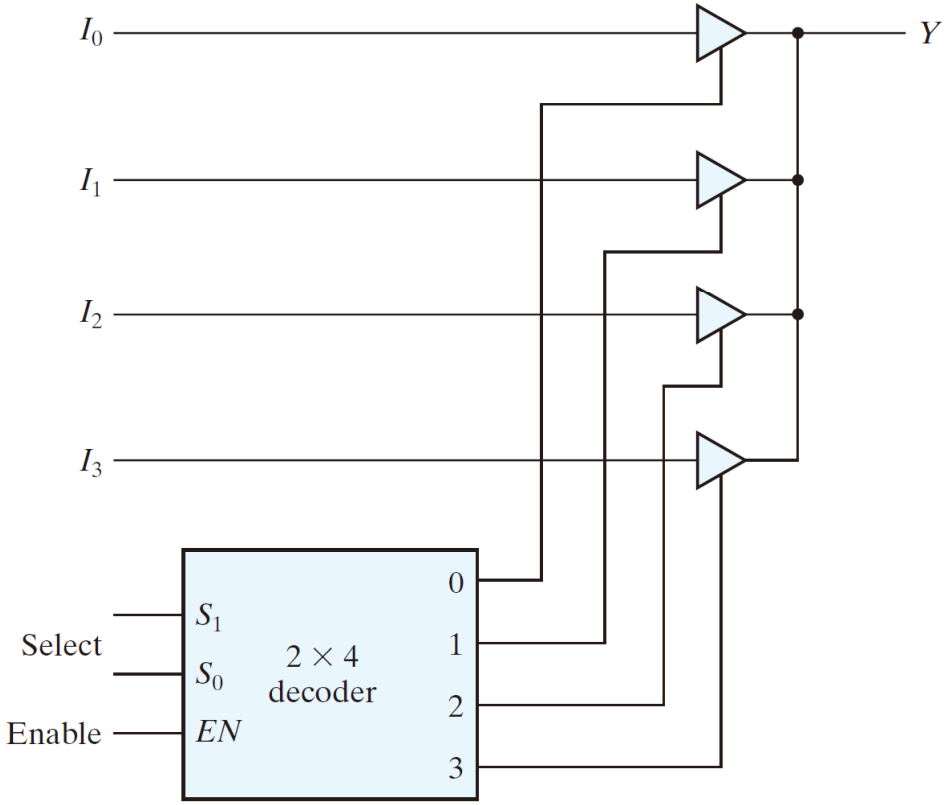


Example: 4-to-1 Multiplexor

- A 4-to-1 Mux implemented with a tri-state buffer:



(a) 2-to-1-line mux



(b) 4-to-1-line mux