Chapter 4: Combinational Logic

4.1 Introduction

4.2 Combinational Circuits

4.3 Analysis Procedure

4.4 Design Procedure

4.5 Binary Adder–Subtractor

4.6 Decimal Adder

4.7 Binary Multiplier

4.8 Magnitude Comparator

4.9 Decoders

4.10 Encoders

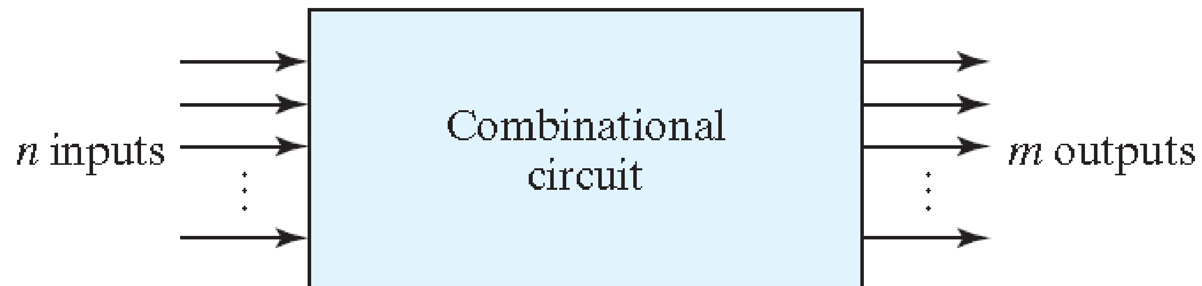
4.11 Multiplexers

4.1 Introduction

- Logic circuits may be ***combinational*** or ***sequential***.
- A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.
- The operation of combinational circuits can be specified logically by a set of Boolean functions.
- Sequential circuits contain *storage* elements in addition to logic gates.
- The outputs of sequential circuits are a function of the inputs and the state of the storage elements.
- Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the circuit behavior must be specified by a time sequence of inputs and internal states.
- Sequential circuits are discussed in Chapters 5 and 8 .

4.2 Combinational Circuits

- A combinational circuit consists of an interconnection of logic gates.
- Combinational circuits react to the values at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data.
- A block diagram of a combinational circuit is shown.



- The n inputs come from an external source; the m outputs are produced by the combinational circuit and go to an external destination.
- Each input and output variable is an analog electrical signal whose values are interpreted to be a binary signal that represents logic 1 and logic 0.

- For n input variables, there are 2^n possible combinations.
- For each possible input combination, there is one possible value for each output.
- Thus, a combinational circuit can be specified with a truth table that lists the output values for each combination of input variables.
- A combinational circuit also can be described by m Boolean functions, one for each output variable.
- Each output function is expressed in terms of the n input variables.

- Several extensively used combinational circuits, such as adders, subtractors, multipliers, comparators, decoders, encoders, and multiplexers, are available in standard integrated circuit components and used as *standard cells* in complex very large scale integrated (VLSI) circuits.

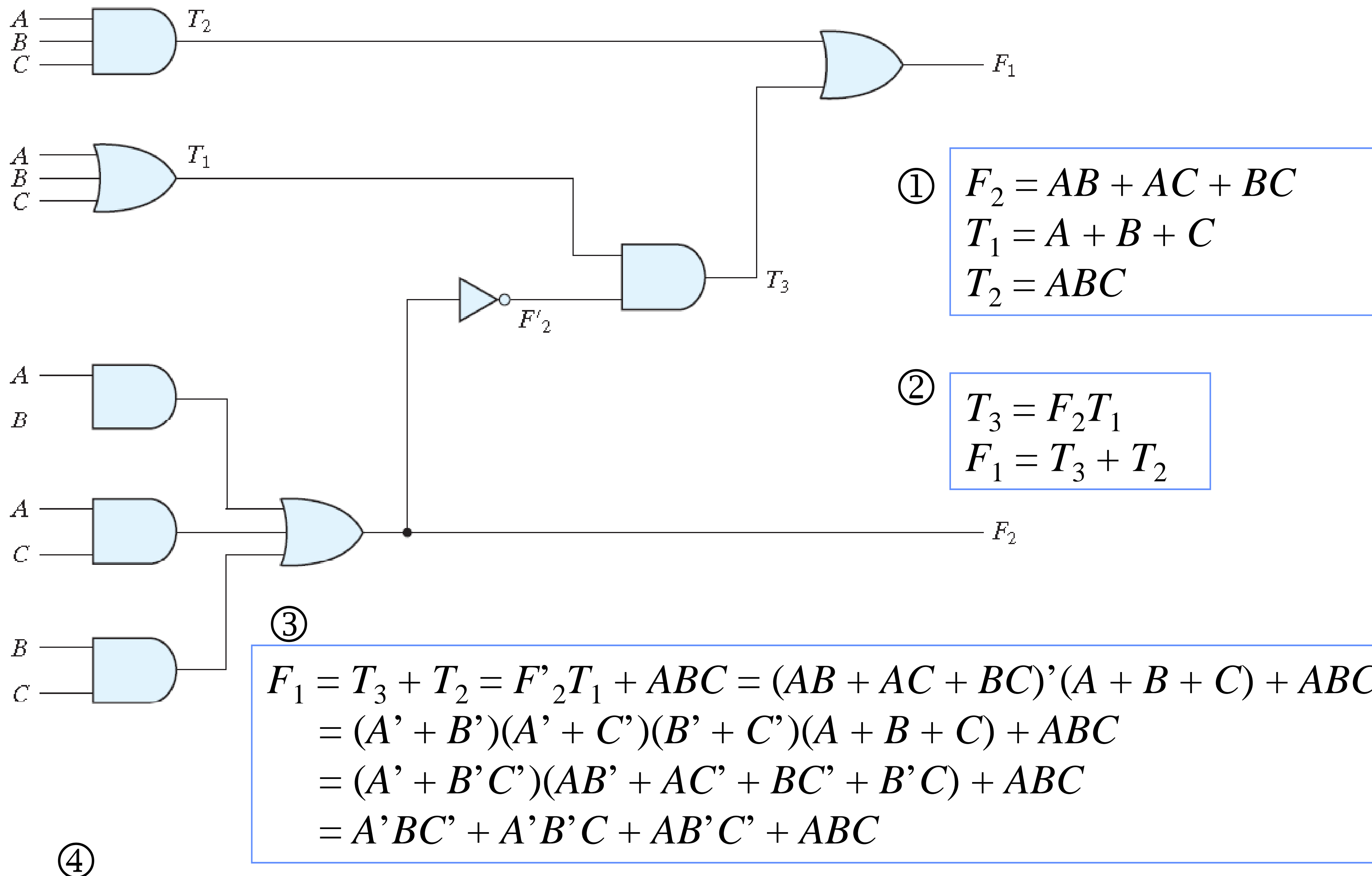
4.3 Analysis Procedure

- The analysis is to determine the function of an implemented circuit.
- This task starts with a given logic diagram and culminates with a set of Boolean functions, a truth table, or, possibly, an explanation of the circuit operation.
- The analysis can be performed manually by finding the Boolean functions or truth table or by using a computer simulation program.
- The first step in the analysis is to make sure that the given circuit is combinational and not sequential.
- **A combinational circuit has no feedback paths** or **memory elements**.
- A feedback path is a connection from the output of one gate to the input of a second gate whose output forms part of the input to the first gate.
- Feedback paths in a digital circuit define a sequential circuit.

Steps to Obtain Output Boolean Functions

- To obtain the output Boolean functions from a logic diagram, we proceed as follows:
  1. Label all gate outputs that are a function of input variables with arbitrary symbols—but with meaningful names. Determine the Boolean functions for each gate output.
  2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.
  3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.
  4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

Example



①
$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

②
$$T_3 = F'_2 T_1$$
$$F_1 = T_3 + T_2$$

③
$$\begin{aligned} F_1 = T_3 + T_2 = F'_2 T_1 + ABC &= (AB + AC + BC)'(A + B + C) + ABC \\ &= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\ &= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\ &= A'BC' + A'B'C + AB'C' + ABC \end{aligned}$$

④

F_1 ➔ sum of a full-adder, F_2 ➔ carry of a full-adder

Steps to Obtain Truth Table

- Obtain the truth table directly from the logic diagram as follows:
  1. Determine the number of input variables. For n inputs, form the 2^n possible input combinations and list the binary numbers from 0 to $(2^n - 1)$ in a table.
  2. Label the outputs of selected gates with arbitrary symbols.
  3. Obtain the truth table for the outputs of those gates which are a function of the input variables only.
  4. Proceed to obtain the truth table for the outputs of those gates which are a function of previously defined values until the columns for all outputs are determined.

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

4-4 Design Procedure

- The design is to derive a logic circuit or a set of Boolean functions from the specification of the design objective.
- The design procedure involves the following steps:

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean functions for each output as a function of the input variables.
4. Draw the logic diagram and verify the correctness of the design (manually or by simulation).

Design Exploration

- Truth table gives the exact definition of a combinational circuit.
- The output binary functions listed in the truth table are simplified by any available method, such as algebraic manipulation, the map method, or a computer-based simplification program.
- Practical design must consider such constraints as the number of gates, number of inputs to a gate, propagation time of the signal through the gates, number of interconnections, limitations of the driving capability of each gate (i.e., the number of gates to which the output of the circuit may be connected), and various other criteria that must be taken into consideration when designing integrated circuits.
- Since each constraint is dictated by the particular application, it is difficult to make a general statement about what constitutes an acceptable implementation.
- In most cases, the simplification begins by satisfying an elementary objective, such as producing the simplified Boolean functions in a standard form, then the simplification proceeds with further steps to meet other performance criteria.

Code Conversion Example

- BCD to excess-3 code
  - The truth table

Table 4.2
Truth Table for Code Conversion Example

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

AB \ CD	00	01	11	10
00	m_0 1	m_1	m_3	m_2 1
01	m_4 1	m_5	m_7	m_6 1
11	m_{12} X	m_{13} X	m_{15} X	m_{14} X
10	m_8 1	m_9	m_{11} X	m_{10} X

$z = D'$

AB \ CD	00	01	11	10
00	m_0 1	m_1	m_3 1	m_2
01	m_4 1	m_5	m_7 1	m_6
11	m_{12} X	m_{13} X	m_{15} X	m_{14} X
10	m_8 1	m_9	m_{11} X	m_{10} X

$y = CD + C'D'$

AB \ CD	00	01	11	10
00	m_0	m_1 1	m_3 1	m_2 1
01	m_4 1	m_5	m_7	m_6 1
11	m_{12} X	m_{13} X	m_{15} X	m_{14} X
10	m_8	m_9 1	m_{11} X	m_{10} X

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

AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5 1	m_7 1	m_6 1
11	m_{12} X	m_{13} X	m_{15} X	m_{14} X
10	m_8 1	m_9 1	m_{11} X	m_{10} X

$w = A + BC + BD$

Simplified Functions and Logic Diagram

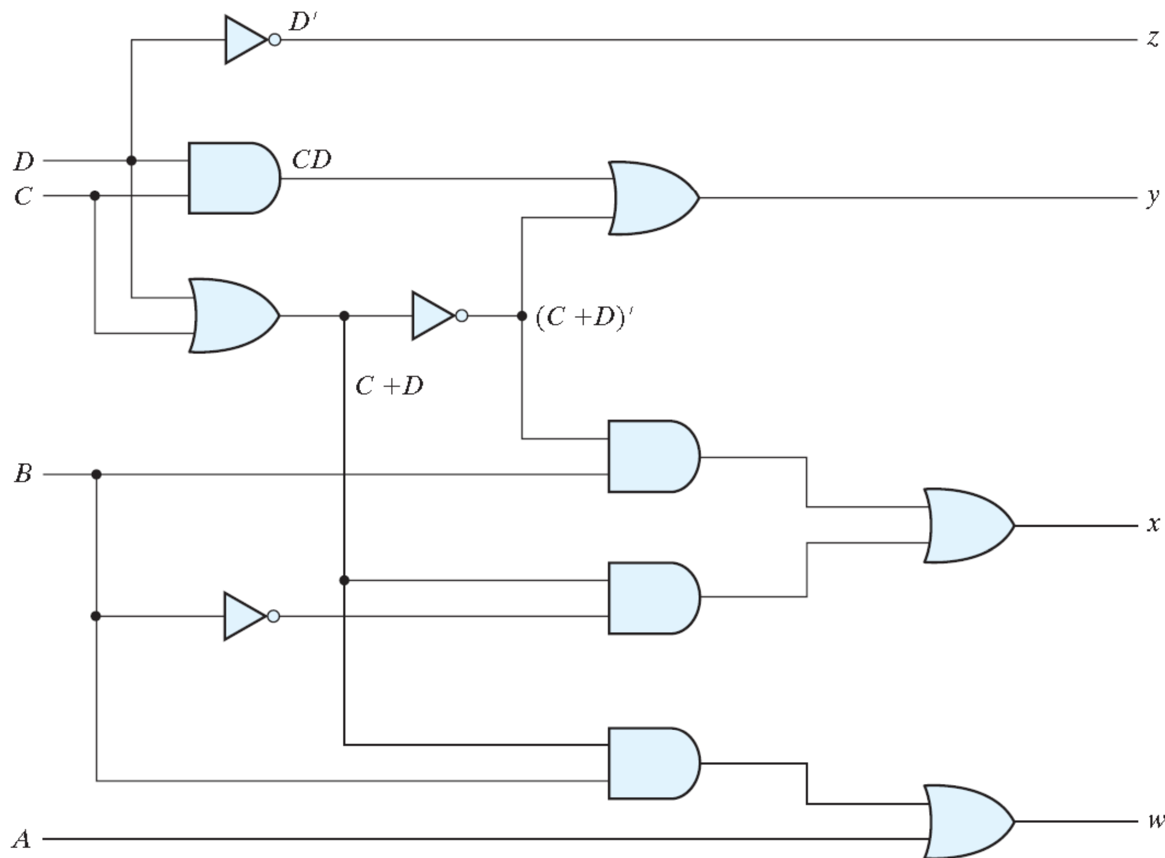
$$z = D'$$

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

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

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

$$w = A + BC + BD = A + B(C + D)$$



4-5 Binary Adder-Subtractor

- Half adder
  - 0 + 0 = 0; 0 + 1 = 1; 1 + 0 = 1; 1 + 1 = 10
  - two input variables: x, y
  - two output variables: C (carry), S (sum)
  - 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

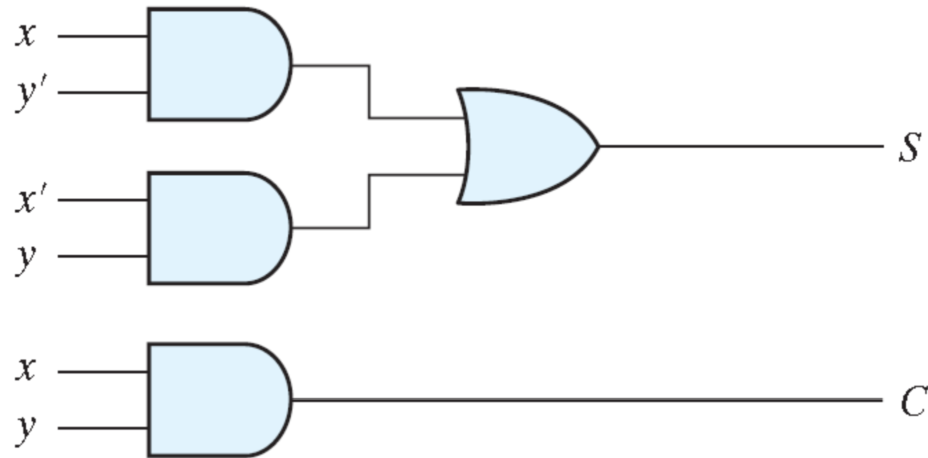
$$S = x'y + xy'$$

$$C = xy$$

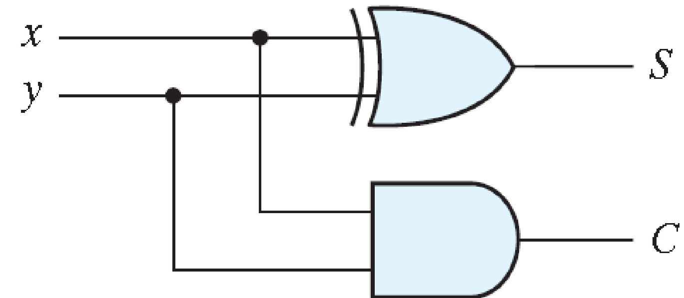
Implementation of Half Adder

- sum of products

exclusive-OR and AND



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



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

Full-Adder

- The arithmetic sum of three input bits
- Three input bits
  - *x*, *y*: two significant bits
  - *z*: the carry bit from the previous lower significant bit
- Two output bits: *C*, *S*
- 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

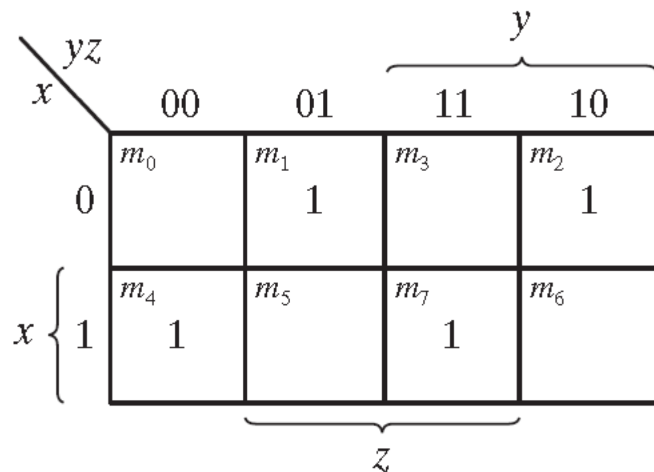
Boolean Functions of Full Adder

Full Adder

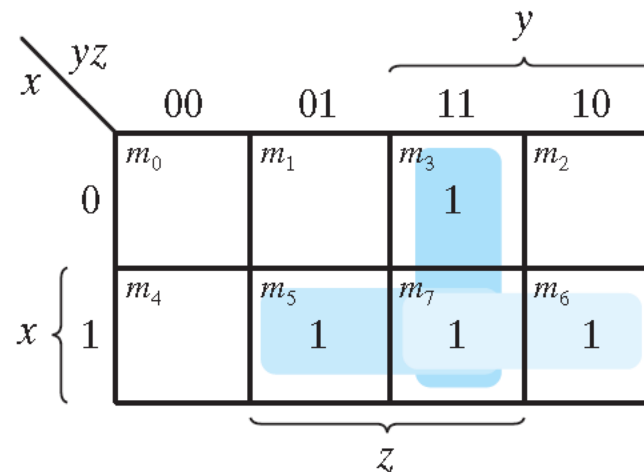
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

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

$$C = xy + xz + yz$$



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

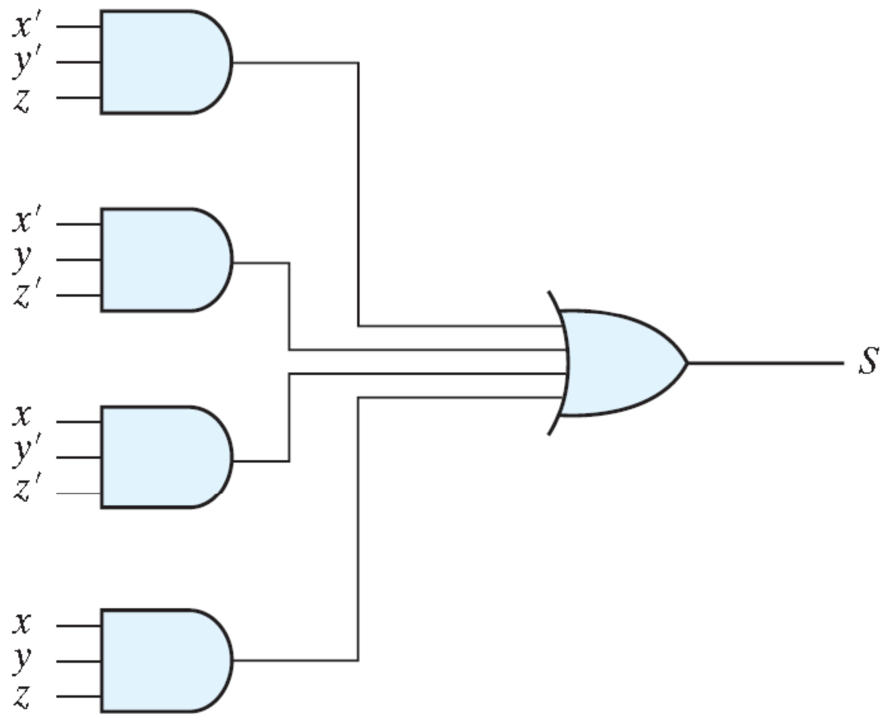


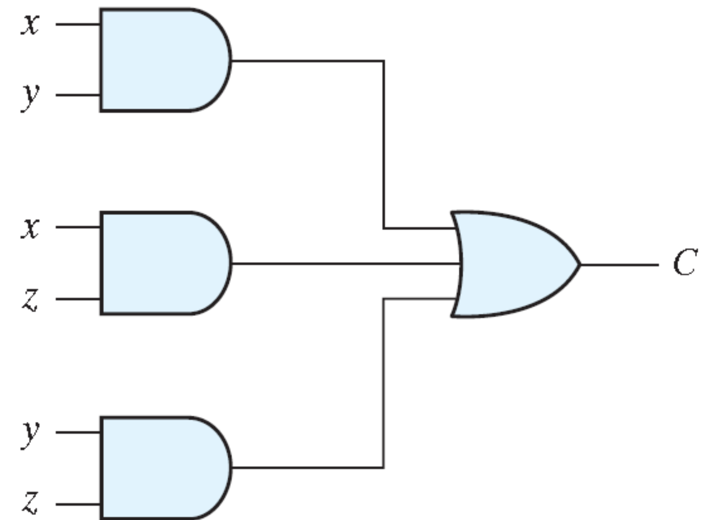
(b) $C = xy + xz + yz$

$$S = x \oplus y \oplus z$$

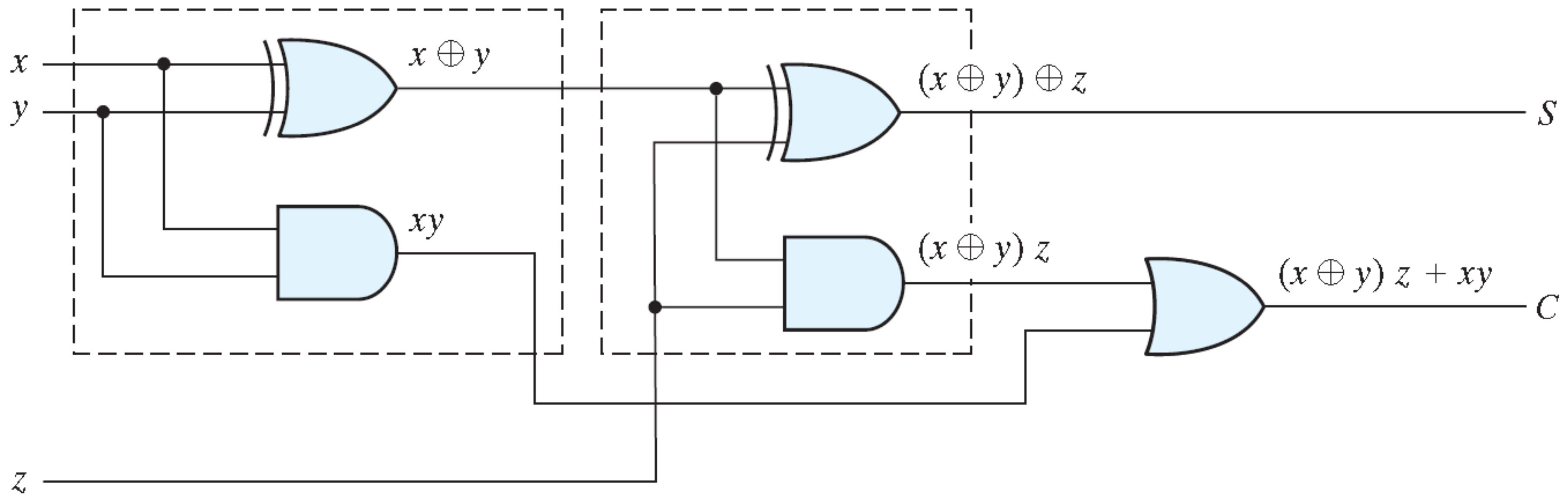
$$C = xy + (x + y)\, z$$

Sum-of-Product Implementation of Full Adder





Implementation of Full Adder with Half Adders



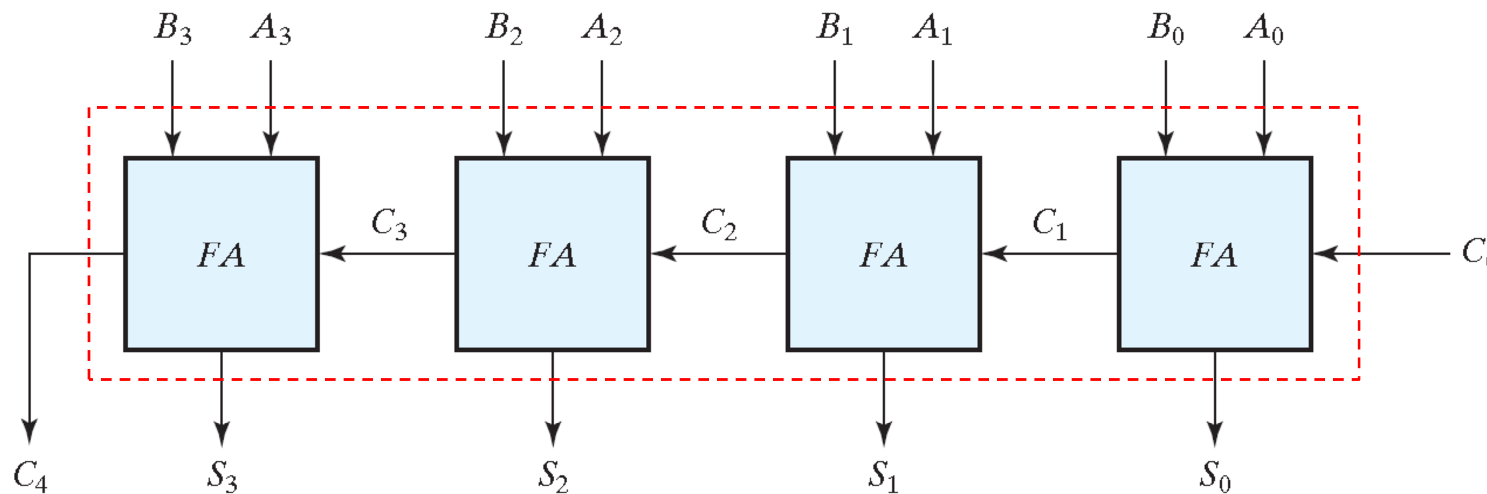
$$S = x \oplus y \oplus z$$

$$C = xy + (x + y)z$$
$$= xy + (x \oplus y)z$$

Binary Adder

- A binary adder can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain (called *ripple-carry* adder).
- Example: 4-bit binary adder

Is it possible to design it with truth table?



$A = 1011$

$B = 0011$

$S = 1110$

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}

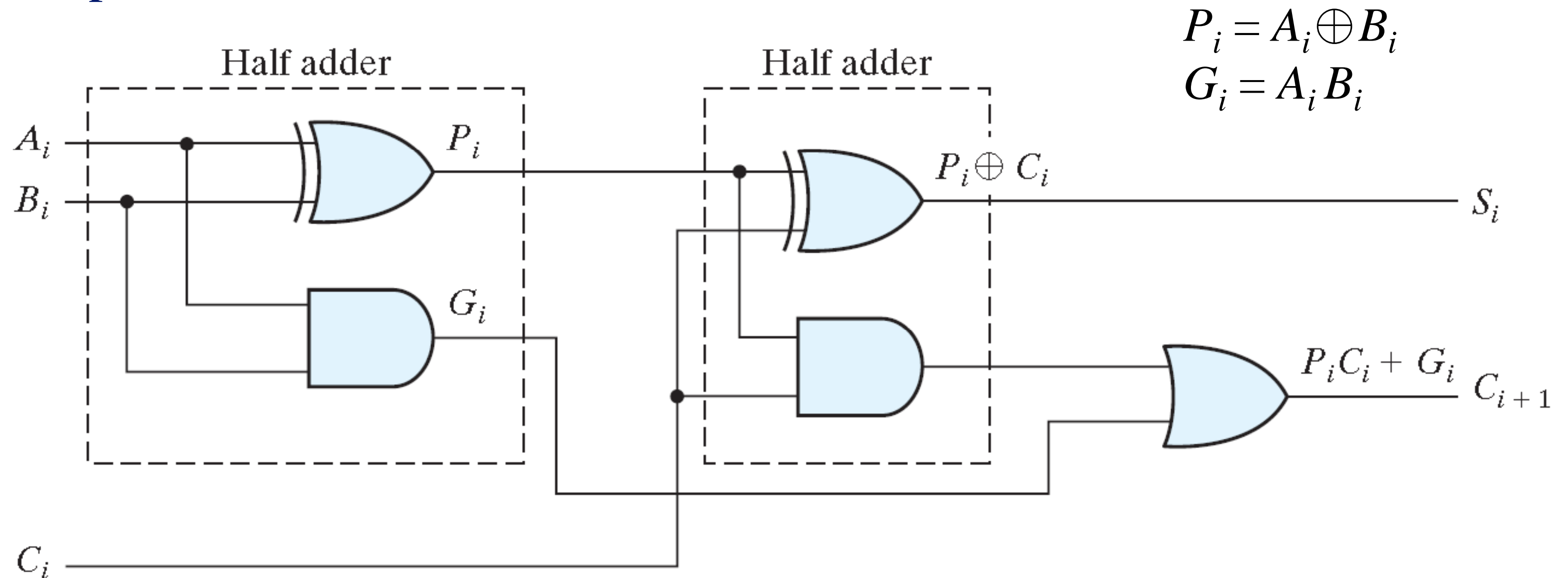
Carry Propagation

- The total propagation time is equal to the sum of the propagation delay of logic gates along a path from input to output.
- The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adders. (called *critical path*)
- Each bit of the sum output, S_i, in the adder will be in its steady-state final value only after the input carry to that stage has been propagated.
- Consider output S_3 in the 4-bit adder, inputs A_3 and B_3 are available as soon as input signals are applied to the adder.
- However, input carry C_3 does not settle to its final value until C_2 is available from the previous stage.
- Similarly, C_2 has to wait for C_1 and so on down to C_0.
- Thus, only after the carry propagates and ripples through all stages will the last output S_3 and carry C_4 settle to their final correct value.
- This will be a serious problem if the adder has a long bit length.

Carry Propagation in Binary Adder

- Re-label the half-adder implementation with $P_i = A_i \oplus B_i$ and $G_i = A_i B_i$
- P_i and G_i settle to steady-state values after A_i and B_i propagate through their respective gates.
- The input carry C_i to the output carry C_{i+1} propagates through an AND gate and an OR gate. If there are four full adders in the adder, the output carry C_4 would have $2 * 4 = 8$ gate levels from C_0 to C_4.
- For an n-bit adder, there are $2n$ gate levels for the carry to propagate from input to output.

$P_i = A_i \oplus B_i$
$G_i = A_i B_i$

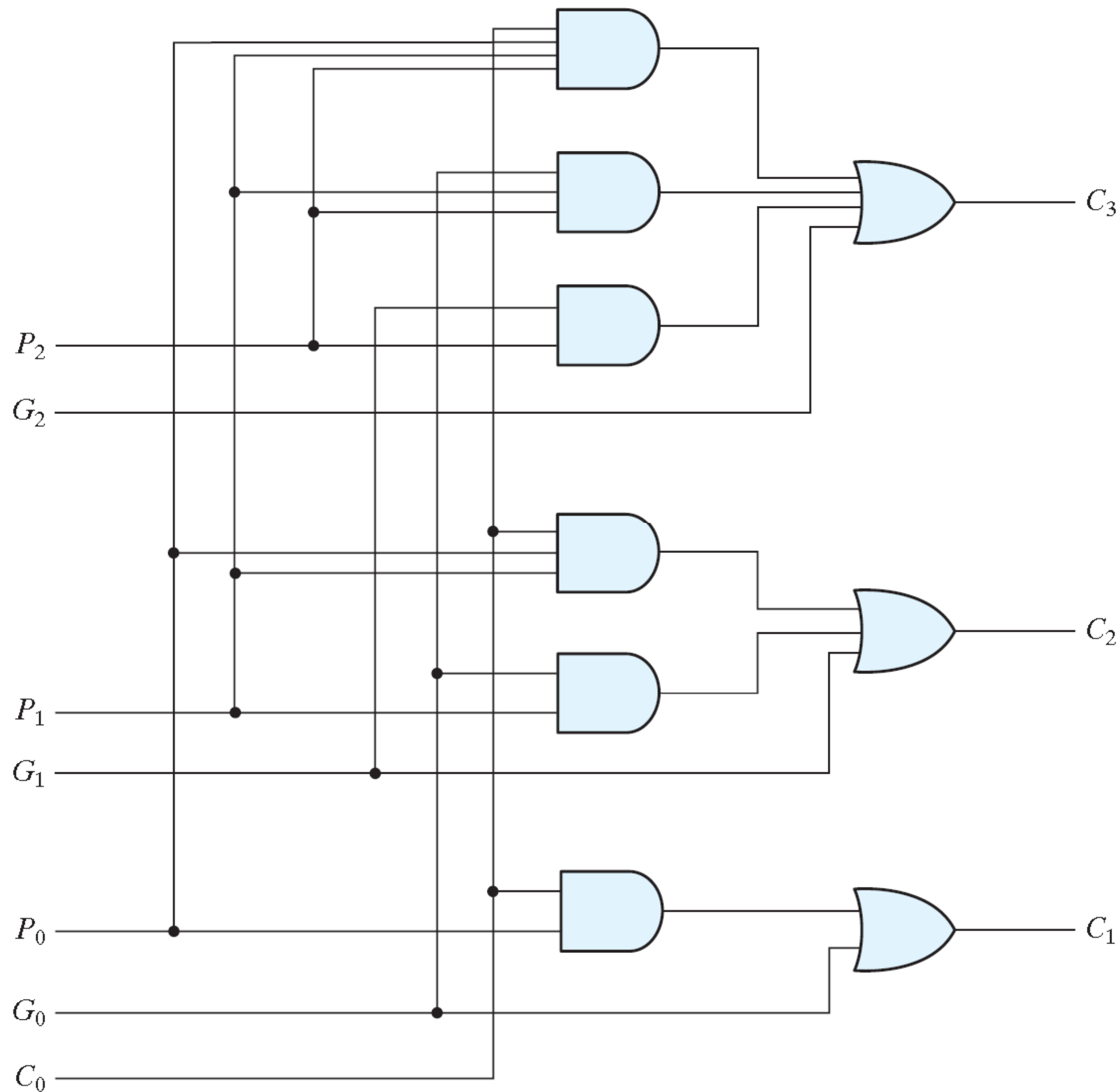


Carry Look-ahead Adder

- Reduce the carry propagation delay using look-ahead carry (more complex mechanism, yet faster)
- Two signals defined: *Carry Propagate*: $P_i = A_i \oplus B_i$ and *Carry Generate*: $G_i = A_i B_i$
- Sum and Carry are re-defined as: $S_i = P_i \oplus C_i$ and $C_{i+1} = G_i + P_i C_i$
- The carry signals of the adder become
  - C_0 = carry input
  - $C_1 = G_0 + P_0 C_0$
  - $C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$
  - $C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$
- C_3 does not have to wait for C_2 and C_1 to propagate; in fact, C_3 is propagated at the same time as C_1 and C_2.

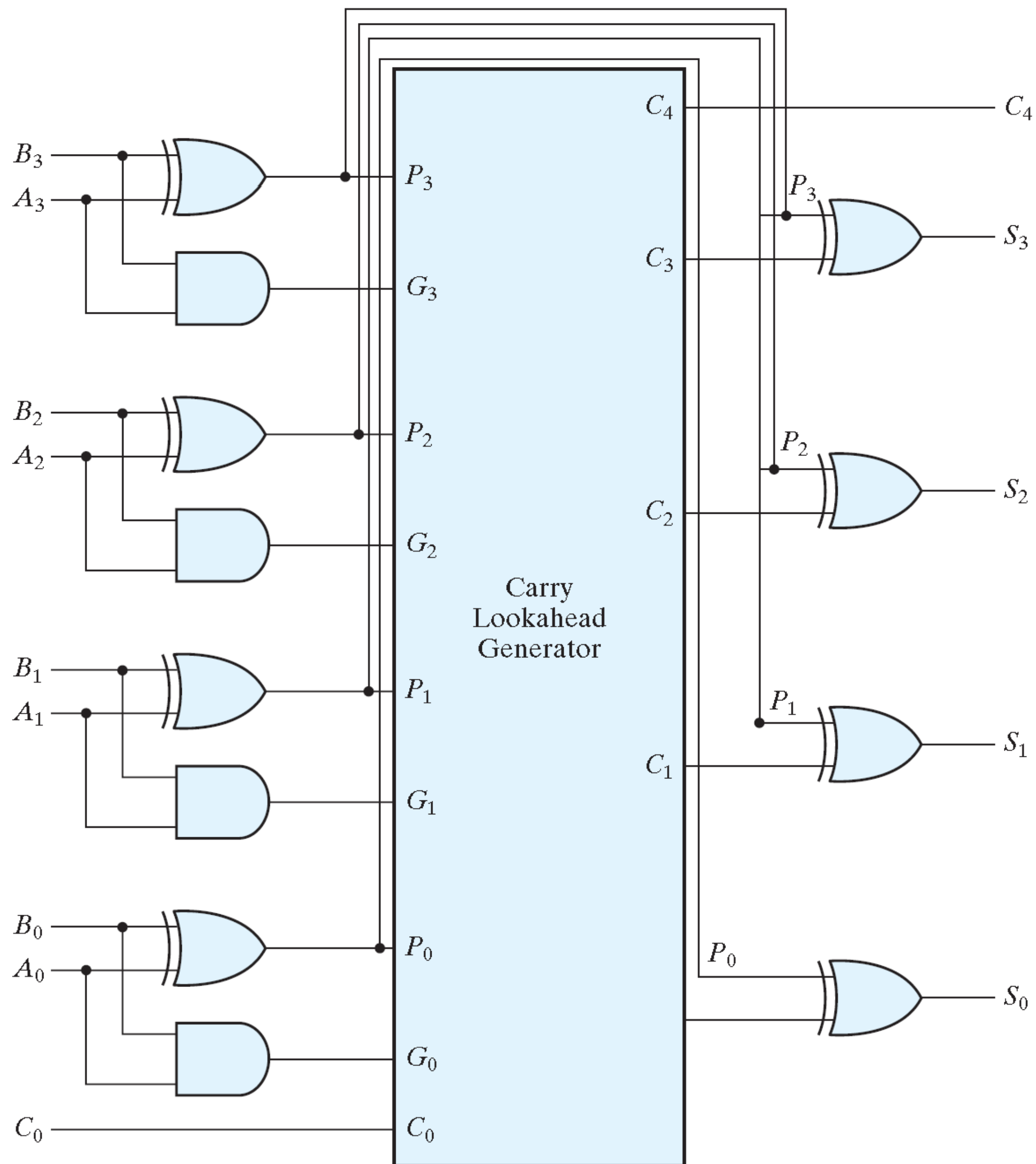
Carry Look-ahead Generator

$P_i - A_i \oplus B_i$
$G_i = A_i B_i$



Where is its longest propagation delay path?

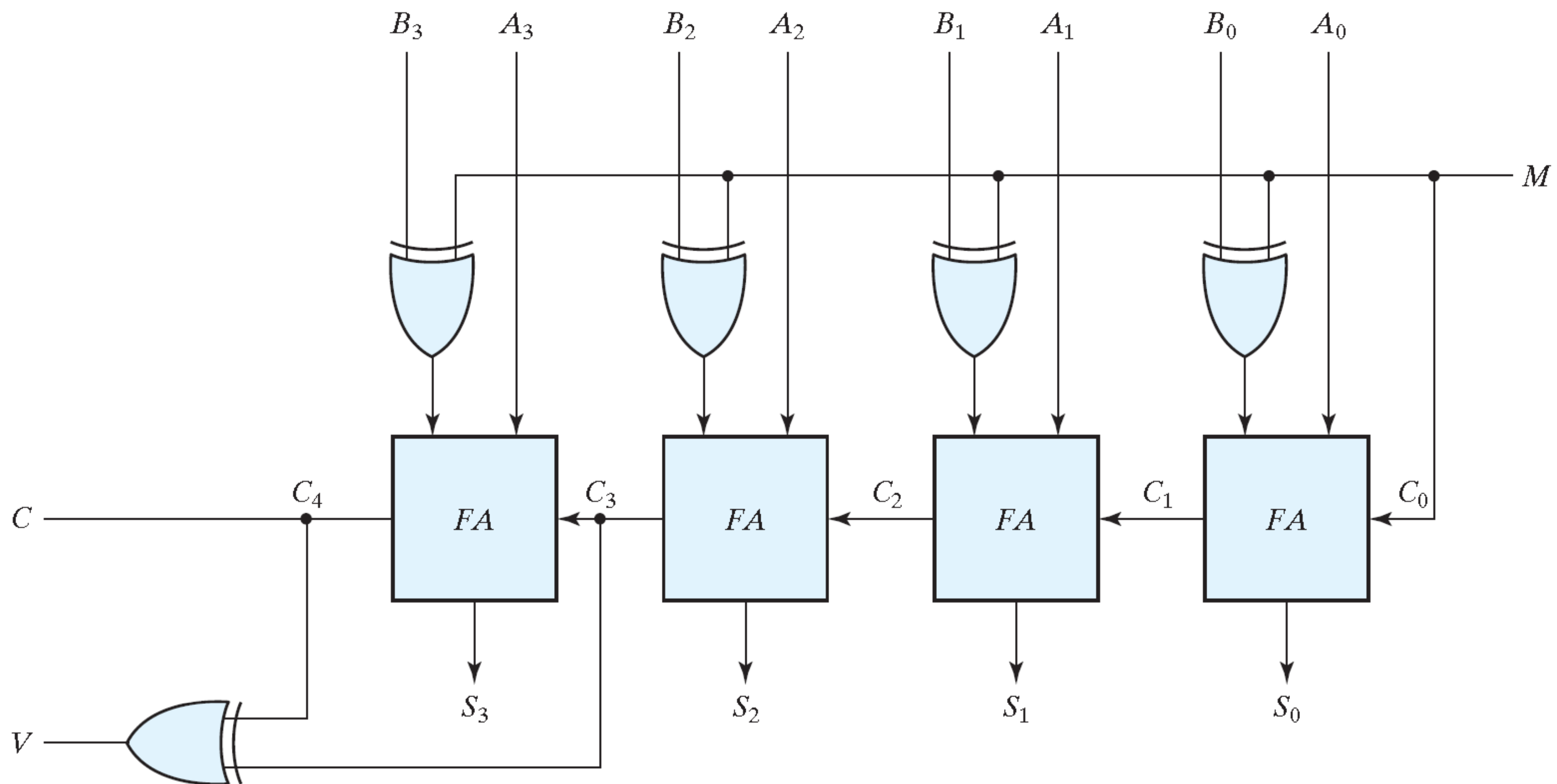
Four-bit Carry Look-ahead Adder



There exist many other faster adders and are not mentioned in this course.

Binary Adder-Subtractor

- A - B = A + (2's complement of B)
- 4-bit adder-subtractor
  - $M = 0 \rightarrow A + B$; $M = 1 \rightarrow A + B' + 1$
- Output V is for detecting an *overflow*.



Overflow

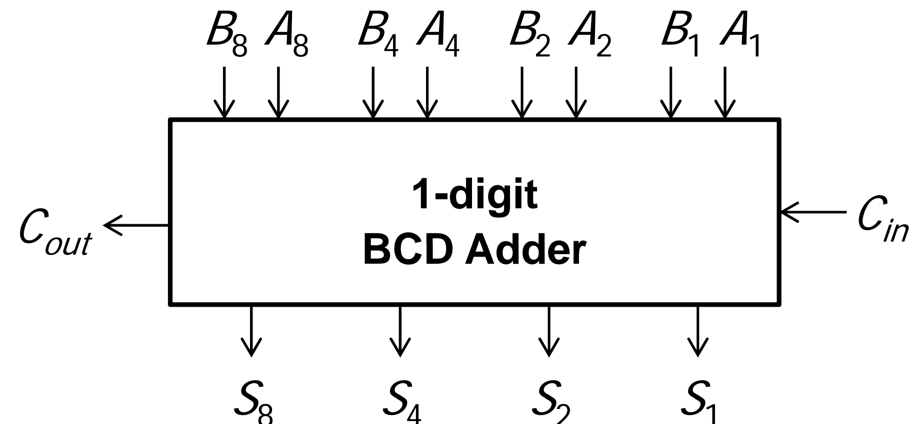
- Overflow: two *n*-digit numbers are added and the sum becomes (*n*+1)-digit.
- Overflow is a problem in computers because the number of bits that hold the number is finite and a (*n*+1)-bit result cannot be stored in an *n*-bit word.
- Many computers detect the occurrence of an overflow, a corresponding flip-flop (1-bit memory) is set that can then be checked by the user (or program).
- Add two positive numbers and obtain a negative number
- Add two negative numbers and obtain a positive number
- An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position.
- $V = 0$ → no overflow; $V = 1$ → overflow (see previous page)

Example: 8-bit signed addition, 2's complement, ranges -128 ~ +127

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

4-6 Decimal Adder

- Add two BCD's
  - 9 inputs: two BCD's and one carry-in
  - 5 outputs: one BCD and one carry-out



- Design approaches
  - A truth table with 2^9 entries
  - use binary full Adders
    - » the decimal sum must be not larger than 19 (= 9 + 9 + 1)
    - » the BCD sum is no larger than 9; $(S_8S_4S_2S_1) \leqq (1001)$

The Sum of a 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

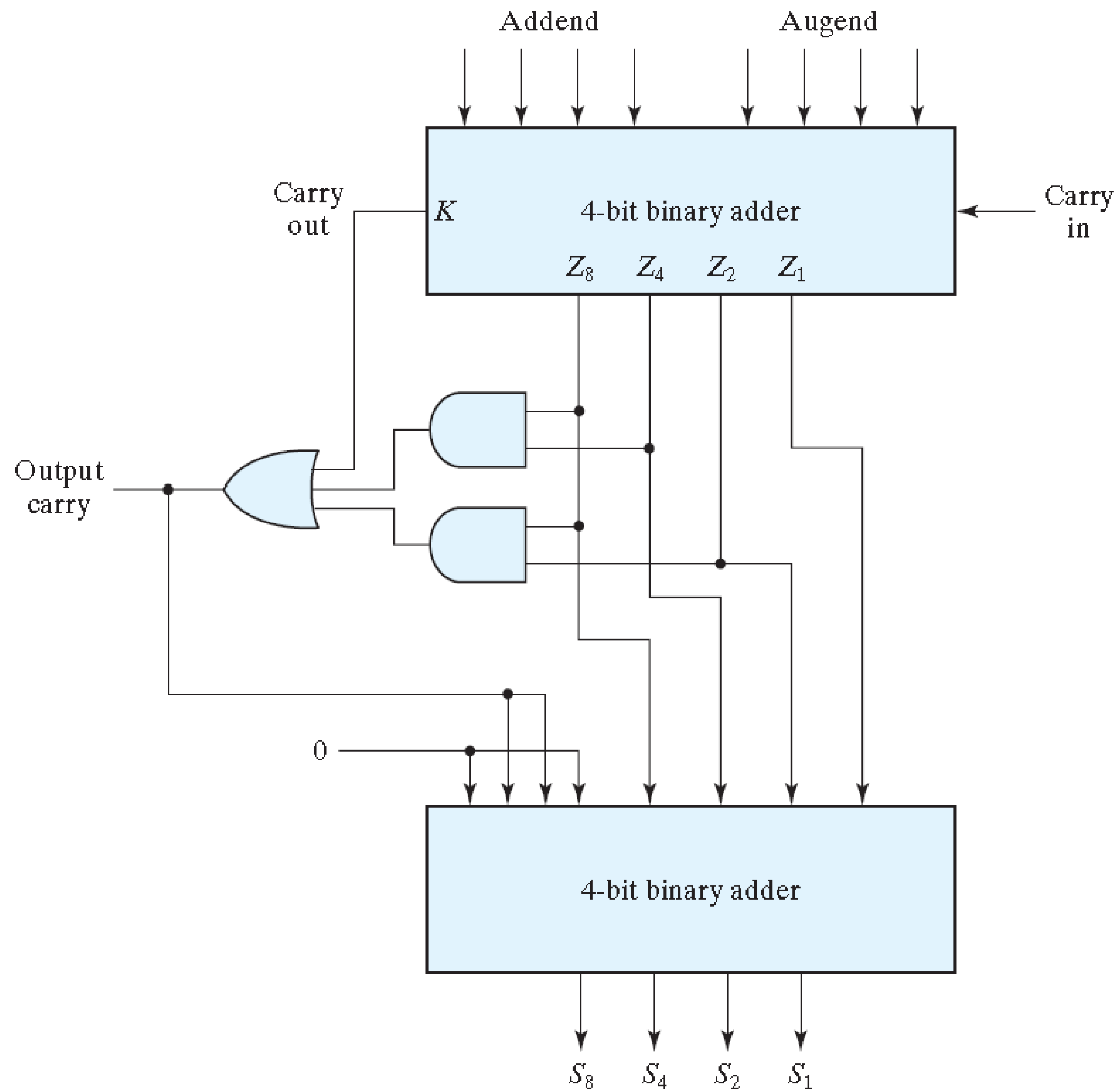
BCD Adjustment

- When the binary sum is equal to or less than 1001, the corresponding BCD number is identical, no conversion is needed.
- When the binary sum is greater than 1001, an addition of 6 (0110) converts it to the correct BCD representation and also produces an output carry as required.
- Modifications are needed if the sum > 9 (1001)
  - C must be set to 1, if
    - » $K = 1$, or
    - » $Z_8 Z_4 = 1$, or
    - » $Z_8 Z_2 = 1$

$$C = K + Z_8 Z_4 + Z_8 Z_2$$
$$= K + Z_8 (Z_4 + Z_2)$$

- When $C = 1$, add 0110 to the binary sum.

BCD Adder



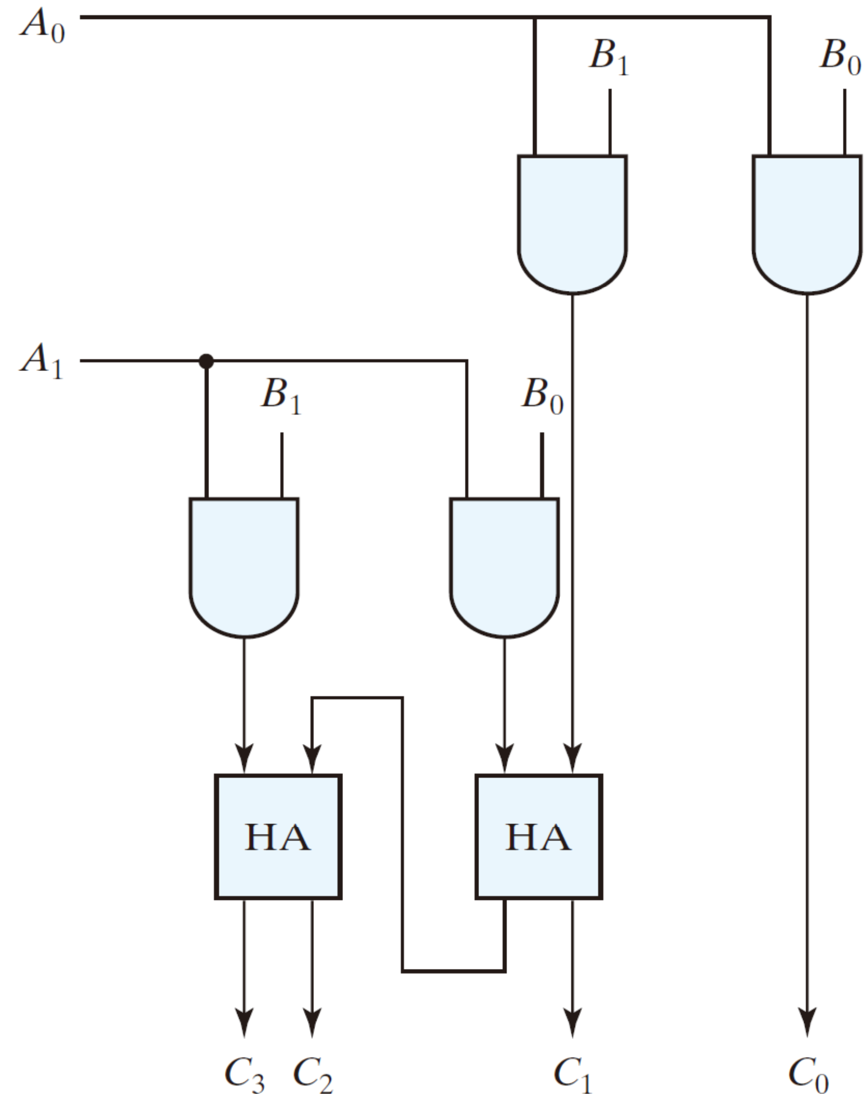
Binary Multiplier

- Performed in the same way as multiplication of decimal numbers.
- Partial products: AND operations.
- 2-bit x 2-bit ➔ 4-bit

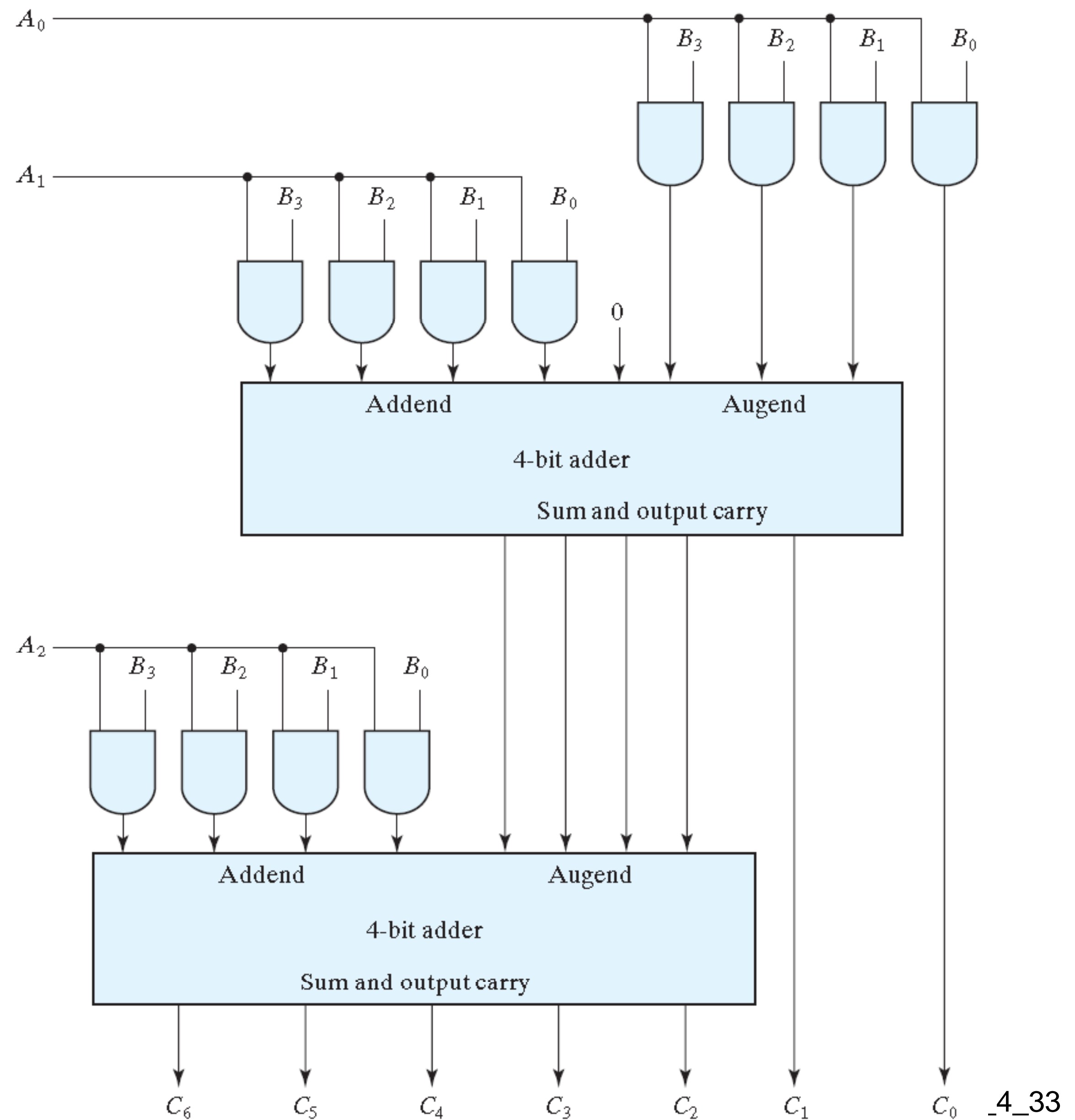
$$
\begin{array}{ccccc}
 & & & B_1 & B_0 \\
 & & \times & A_1 & A_0 \\
\hline
 & & & A_0B_1 & A_0B_0 \\
 & + & A_1B_1 & A_1B_0 & \\
\hline
C_3 & & C_2 & C_1 & C_0
\end{array}
$$

1-bit multiplication ➔ AND

0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1



- For J multiplier and K multiplicand bits, we need $(J * K)$ AND gates and $(J - 1)$ K-bit adders to produce a product of $(J + K)$ bits.
- $K = 4$ and $J = 3$:
  - 12 AND gates and
  - two 4-bit adders
  - produce a 7-bit product.



There are a lot of multipliers has been presented for high speed applications.

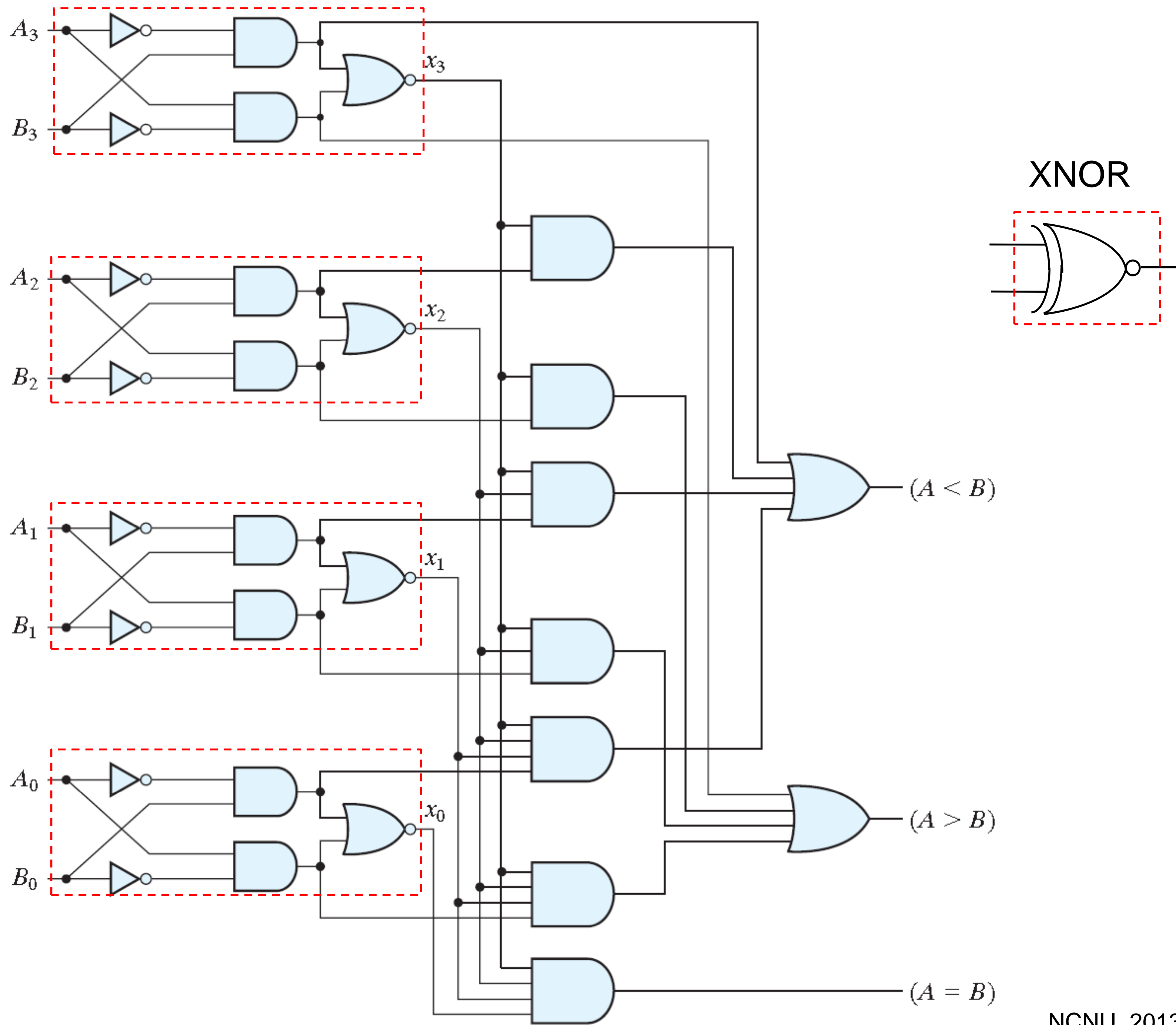
4-8 Magnitude Comparator

- A *magnitude comparator* compares two numbers *A* and *B* and determines their relative magnitudes.
- The results of comparison between two numbers are:
  - A > B, A = B, A < B
- Design Approaches
  - the truth table for two n-bit numbers comparison
    - » 2^{2n} entries - too cumbersome for large *n*
  - use inherent regularity of the problem (algorithm approach)
    - » algorithm — a procedure which specifies a finite set of steps
    - » reduce design efforts
    - » reduce human errors

Comparison Algorithm

- Consider two 4-bit numbers, $A = A_3A_2A_1A_0$, $B = B_3B_2B_1B_0$
- A and B are equal ($A = B$) if $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$, and $A_0 = B_0$.
- The equality of each pair of bits can be expressed with an exclusive-NOR function as: $x_i = A_iB_i + A_i' B_i'$ for $i = 0, 1, 2, 3$
- $x_i - 1$ only if the pair of bits in position i are equal (both are 1 or both are 0).
- For equality to exist ($A = B$), all x_i variables must be equal to 1: $(A = B) = x_3x_2x_1x_0$

- To determine whether ($A > B$) or ($A < B$), starting from the MSB, if the two bits are equal, then compare the next lower significant pair of bits until a pair of unequal bits is reached.
- If the corresponding bit of A is 1 and that of B is 0, we conclude that $A > B$.
- If the corresponding digit of A is 0 and that of B is 1, we have $A < B$.
- The sequential comparison can be expressed by the two Boolean functions
  – $(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$
  – $(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$



NCNU_2013_DD_4_36

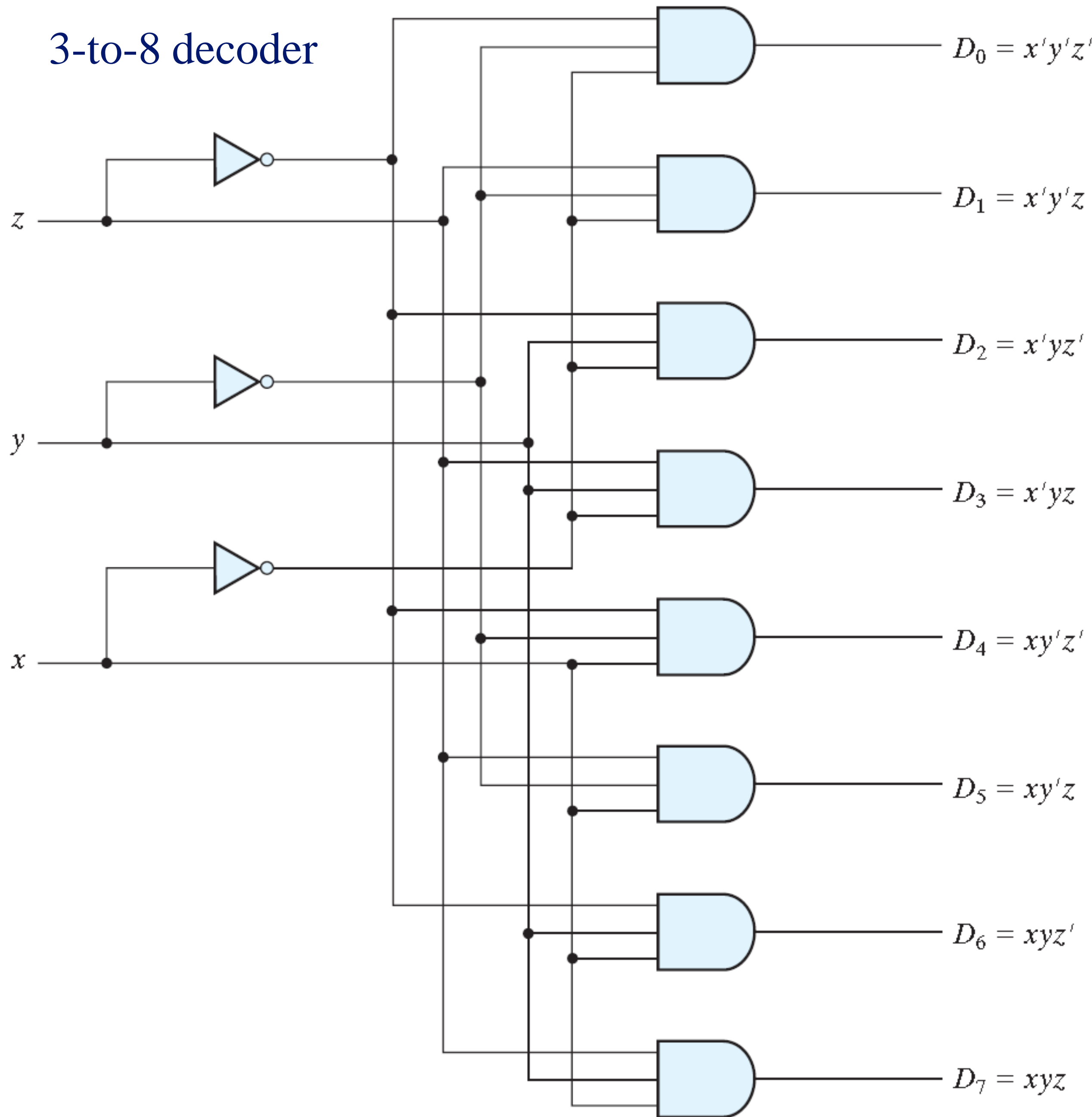
4-9 Decoders

- A *decoder* converts binary information from n input lines to a maximum of 2^n unique output lines.
- A n-to-m decoder ($m \leqq 2^n$)
  - a binary code of n bits has 2^n distinct information
  - n input variables; up to 2^n output lines
  - only one output can be active (high) at any time

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

3-to-8 decoder

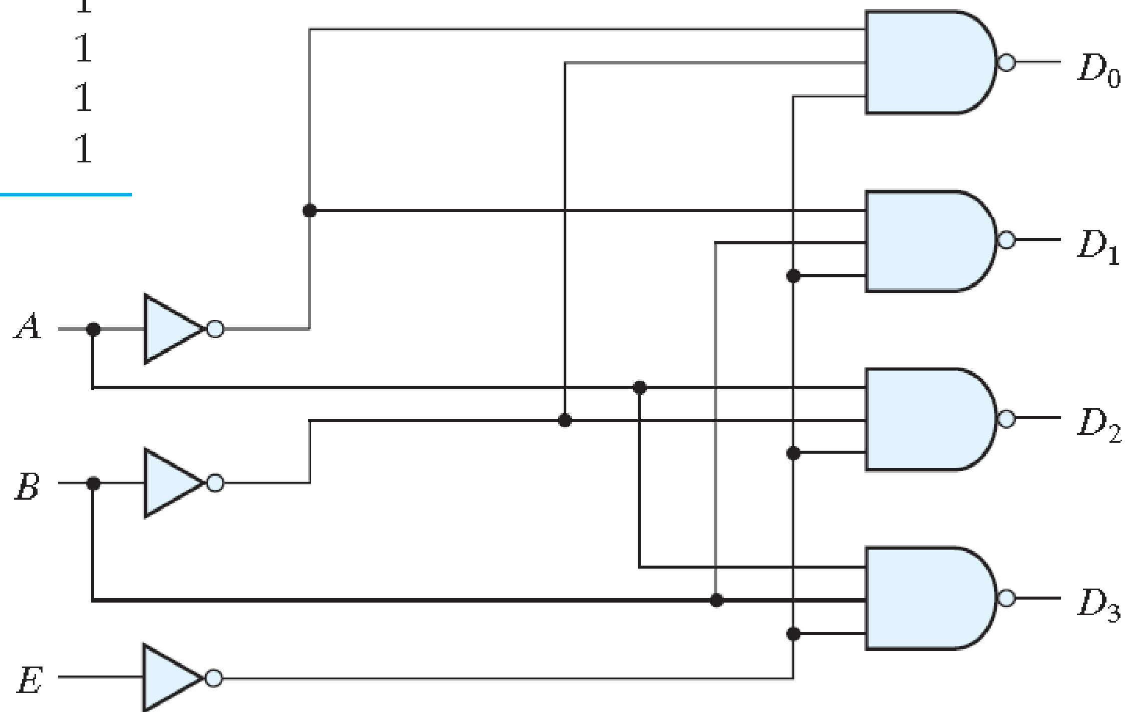


each output = a minterm

Two-to-four Decoder with Enable

- *Enable* input is added to control the circuit operation.

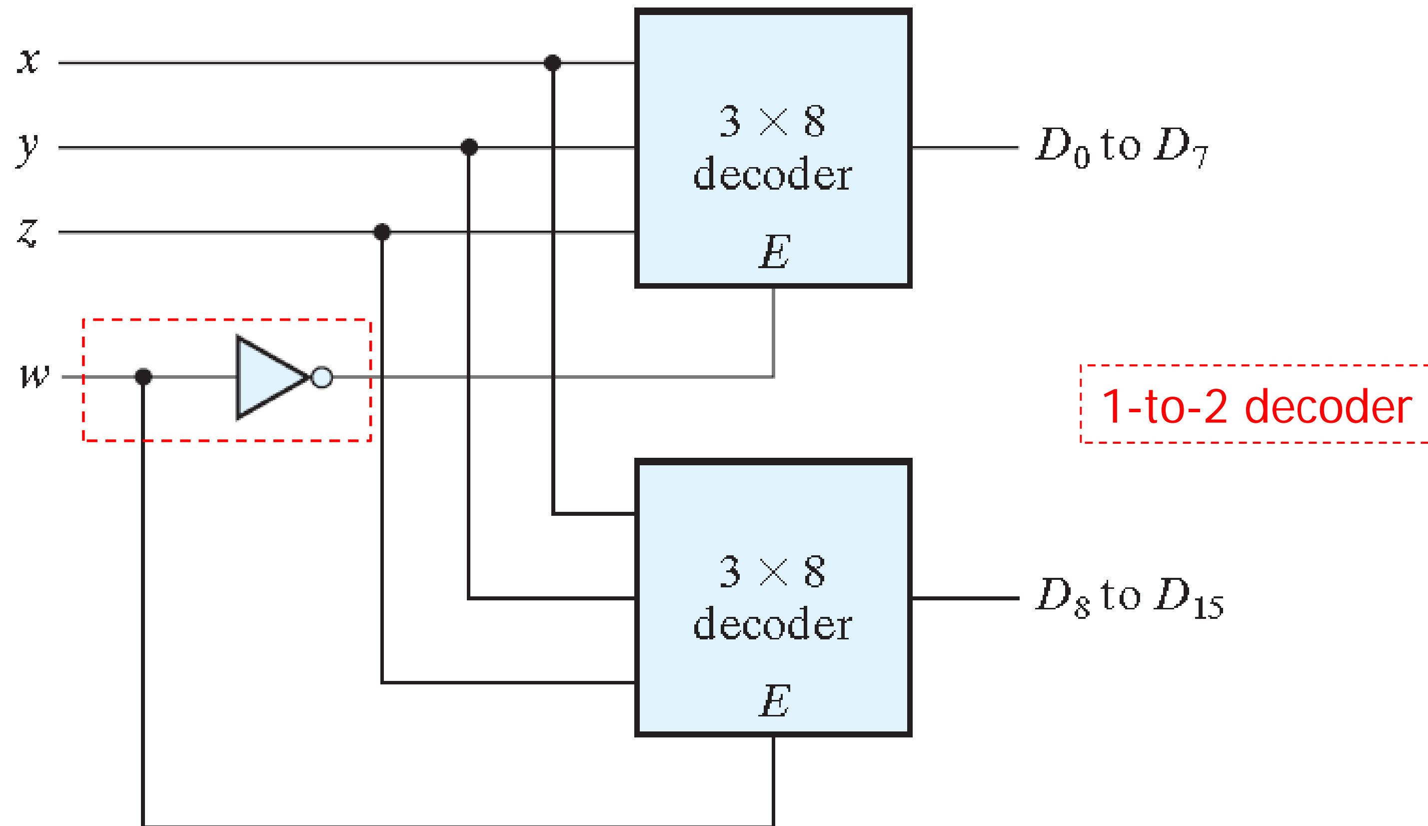
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	1



The demultiplexer described in textbook is questionable.
It will be talked later.

Decoder Expansion

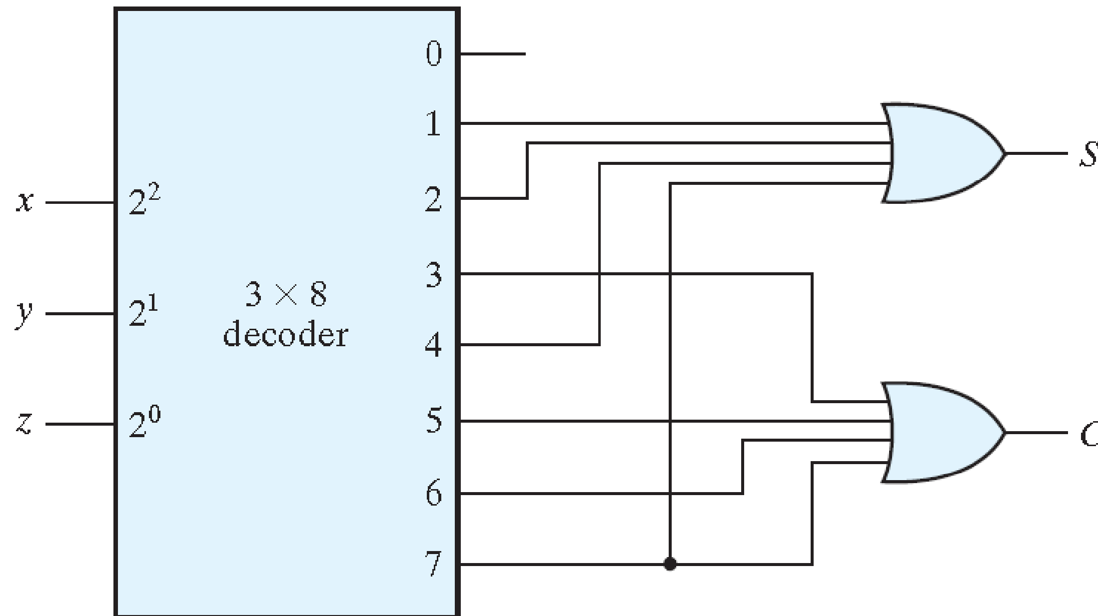
- Expand two 3-to-8 decoder to a 4-to-16 decoder



- How about a 5-to-32 decoder?

Universal Combinational Logic Implementation

- A decoder provides the 2^n minterms of n input variables.
- A decoder and an external OR gate can implement any Boolean function of n input variables in sum-of-minterm form.
- For example, see Table 4.4, a full-adder has its sum $S(x,y,z) = \Sigma(1,2,4,7)$ and carry $C(x,y,z) = \Sigma(3,5,6,7)$.



- Two possible approaches using decoder
  - OR(minterms of F): k inputs
  - NOR(minterms of F'): $2^n - k$ inputs

4-10 Encoders

- The inverse function of a decoder
- 2^n (or fewer) input lines and n output lines
- The output lines generate the binary code corresponding to the input value.
- Example:

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

$z = D_1 + D_3 + D_5 + D_7$ can be implemented with OR gates

$y = D_2 + D_3 + D_6 + D_7$

$x = D_4 + D_5 + D_6 + D_7$

Priority Encoder

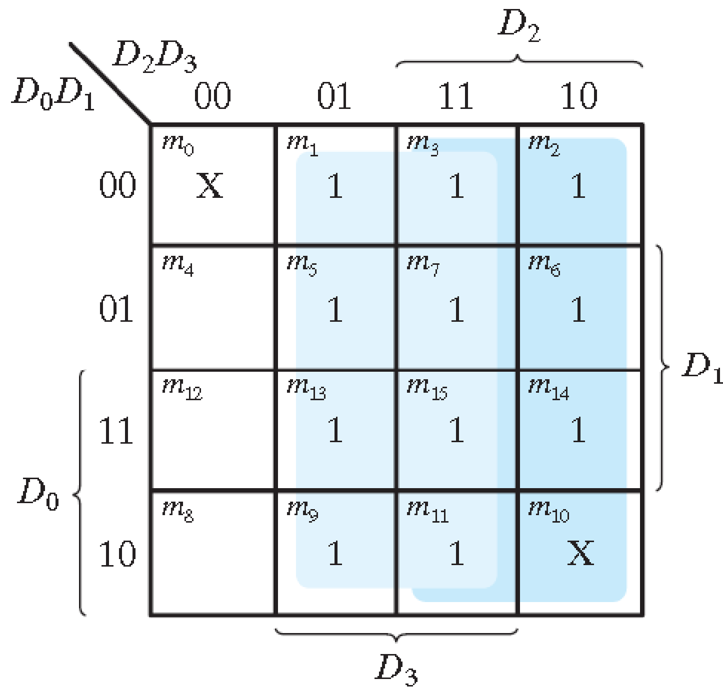
- Encoder that includes the priority function
- Resolve the ambiguity of illegal inputs, only one of the input is encoded, the input having the highest priority will take precedence.
- Example:

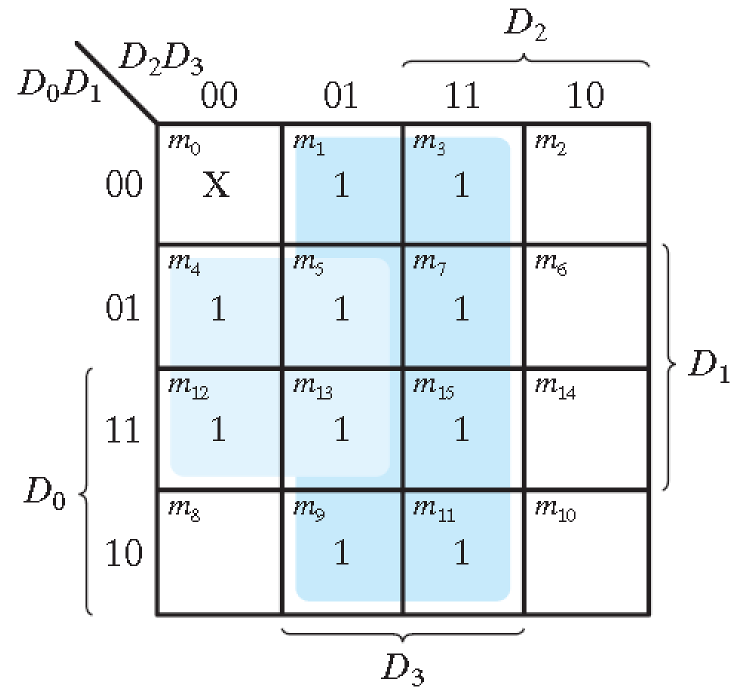
Truth Table of a Priority Encoder

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

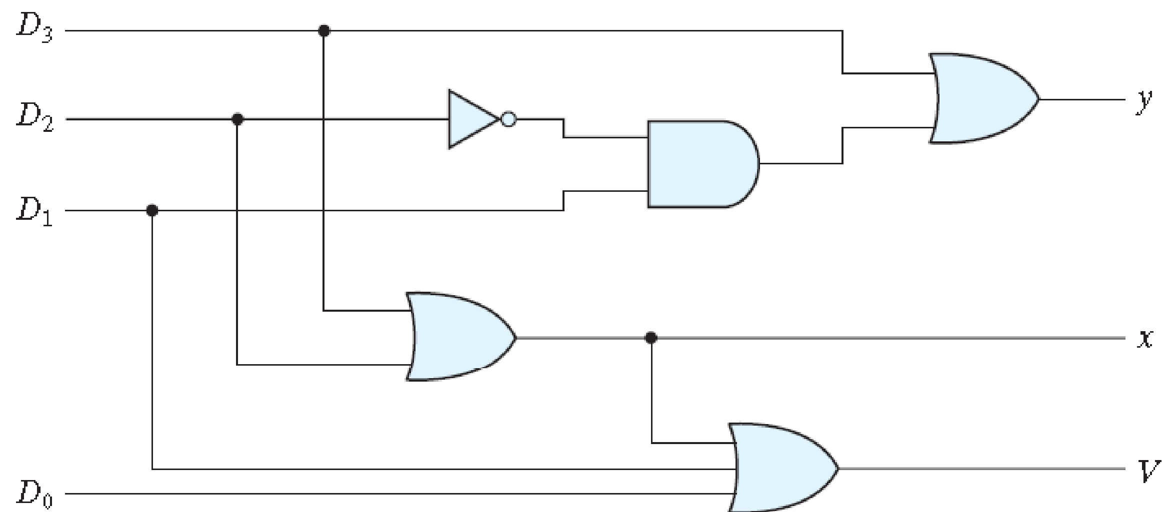
  - » D_3 has the highest priority
  - » D_0 has the lowest priority
  - » X: don't-care conditions
  - » V: valid output indicator

Maps for Simplifying *x* and *y*





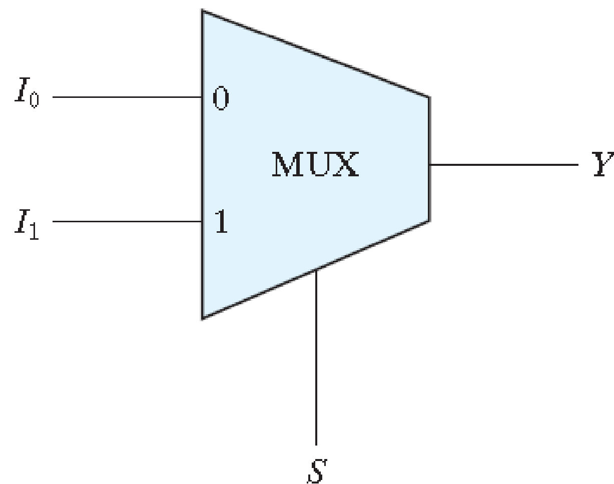
- $x = D_2 + D_3$
- $y = D_3 + D_1 D_2'$
- $V = D_0 + D_1 + D_2 + D_3$

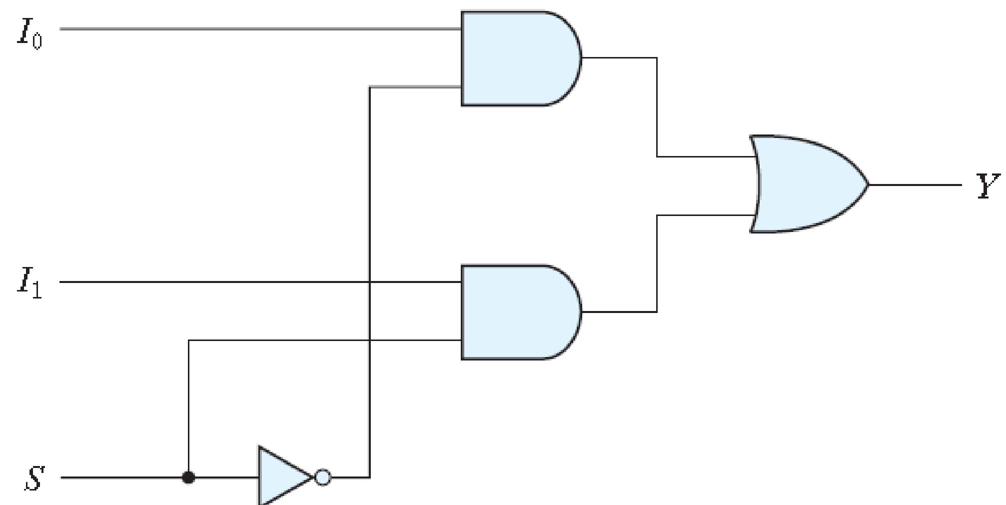


4-11 Multiplexers

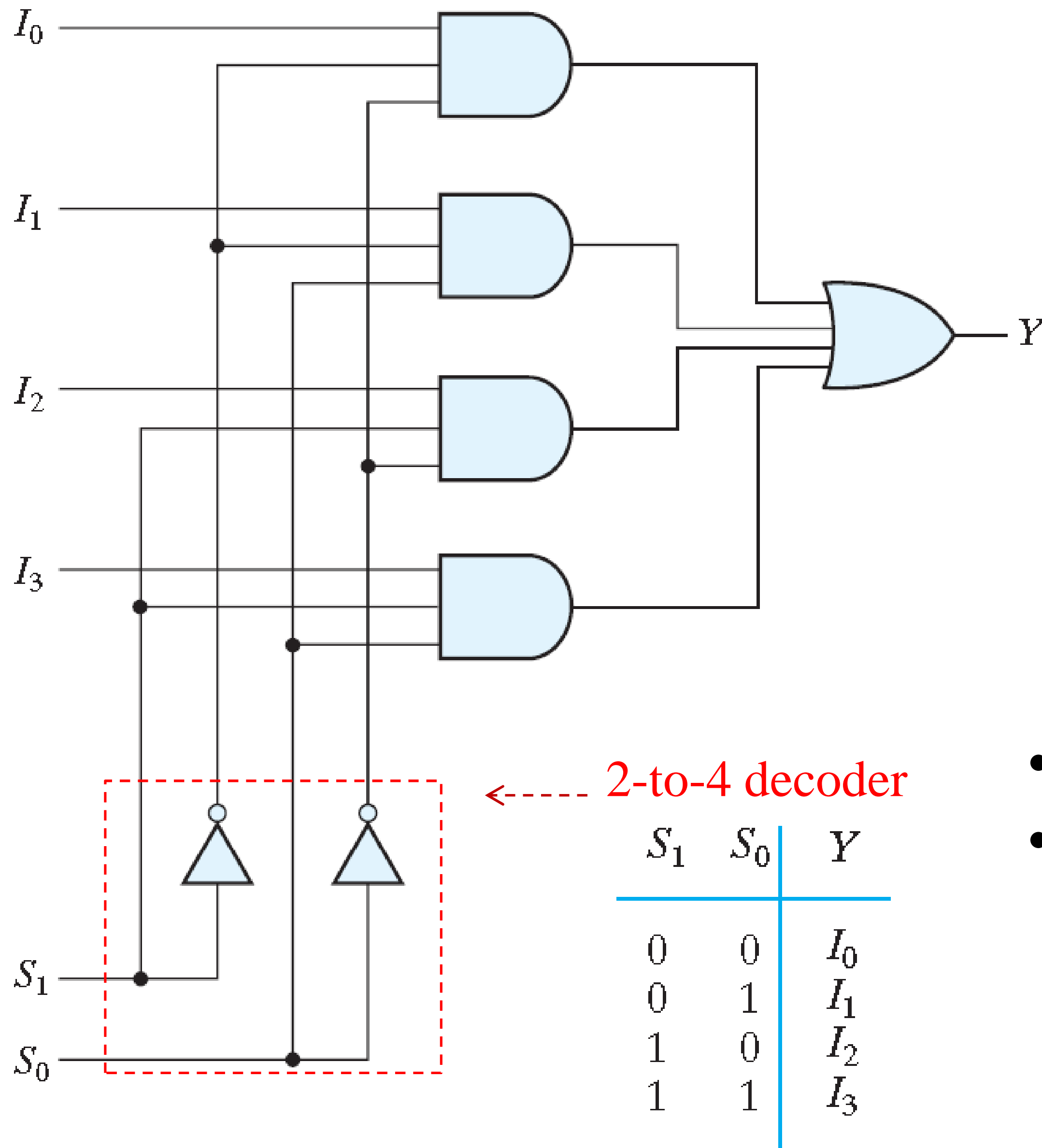
- Select from one of many inputs and directs it to a single output, controlled by a set of selection lines. A multiplexer is also called a *data selector*.
- Normally, there are 2^n inputs and n selection lines whose bit combinations determine which input is selected.
- Example: (two-to-one multiplexer)
  - $Y = I_0$ if $S = 0$, and $Y = I_1$ if $S = 1$.
  - $Y = S'I_0 + SY_1$

S	I_0	I_1	Y
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1





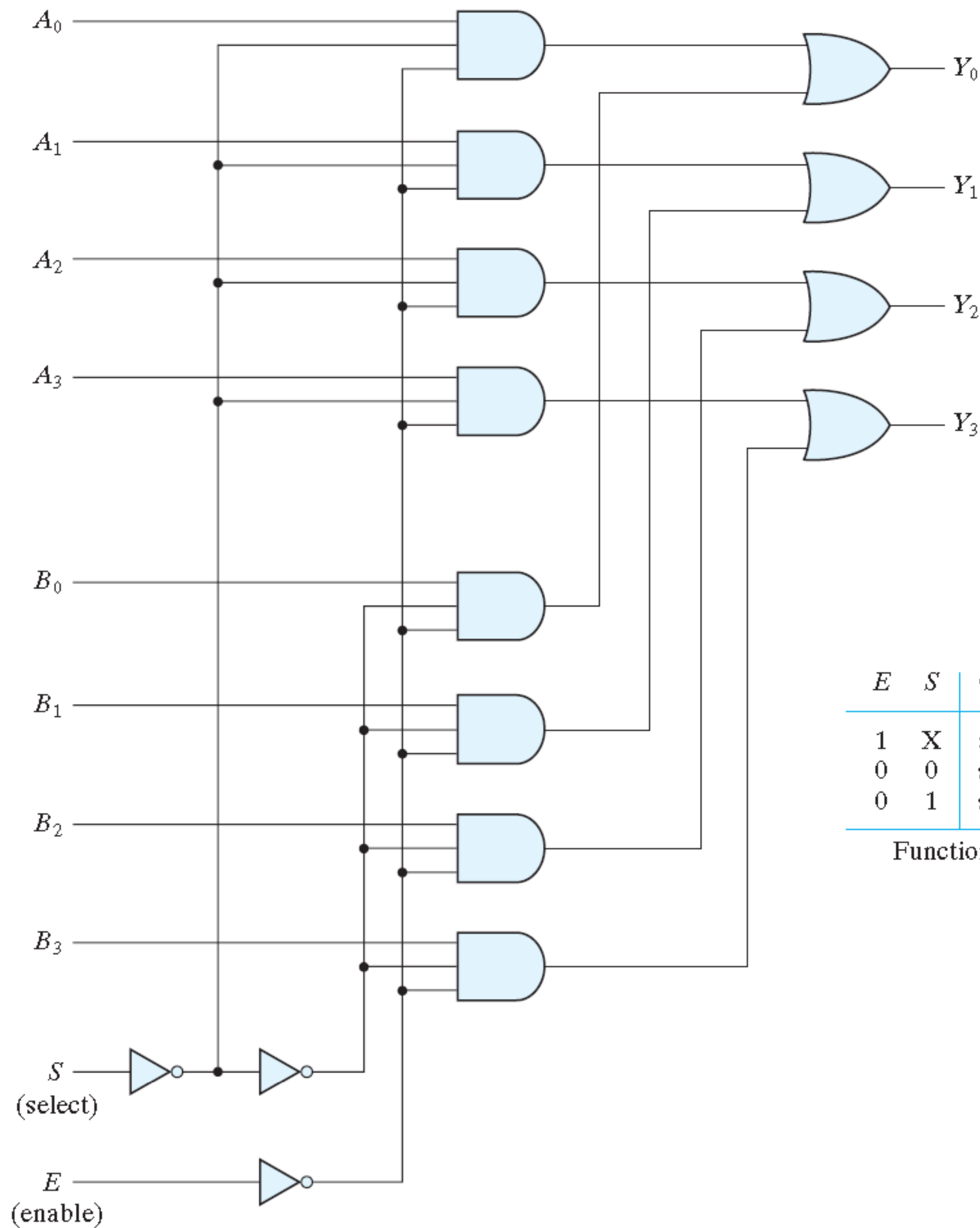
4-to-1 Multiplexer



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

- AND gates act as a switch
- OR gate will face a large fan-in if the input number grow

Quadruple 2-to-1 Multiplexer



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

Function table

$$A = A_3 A_2 A_1 A_0$$

$$B = B_3 B_2 B_1 B_0$$

$$Y = Y_3 Y_2 Y_1 Y_0$$

$Y = A$ means

$$Y_3 = A_3, \; Y_2 = A_2$$

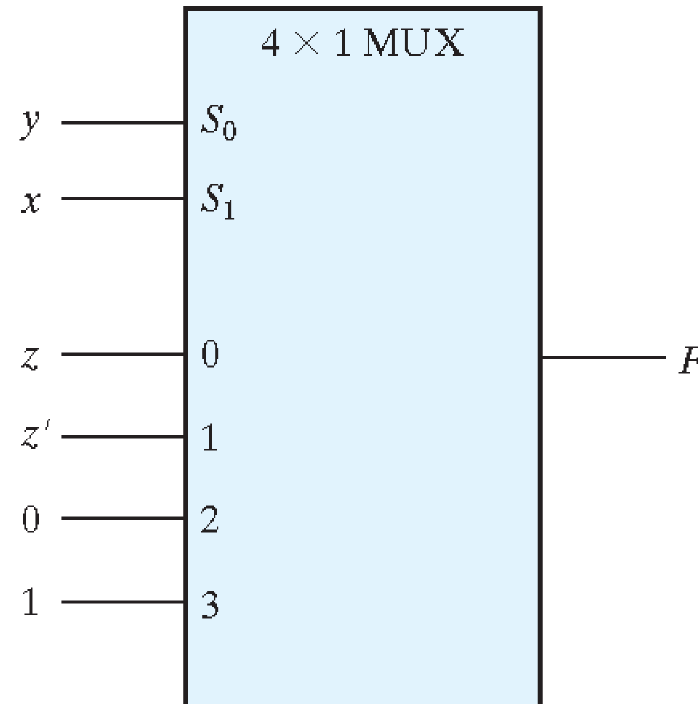
$$Y_1 = A_1, \; Y_0 = A_0$$

Boolean Function Implementation

- MUX has a structure composed of a decoder and an OR gate
- 2^n-to-1 MUX can implement any Boolean function of $n+1$ input variables
- n of these input variables are used as the selection lines
- The remaining single variable is used for the data inputs.
- If the single variable is denoted by z, each data input of the multiplexer will be z, z', 1, or 0.
- Example: $F(x, y, z) = \Sigma(1, 2, 6, 7)$

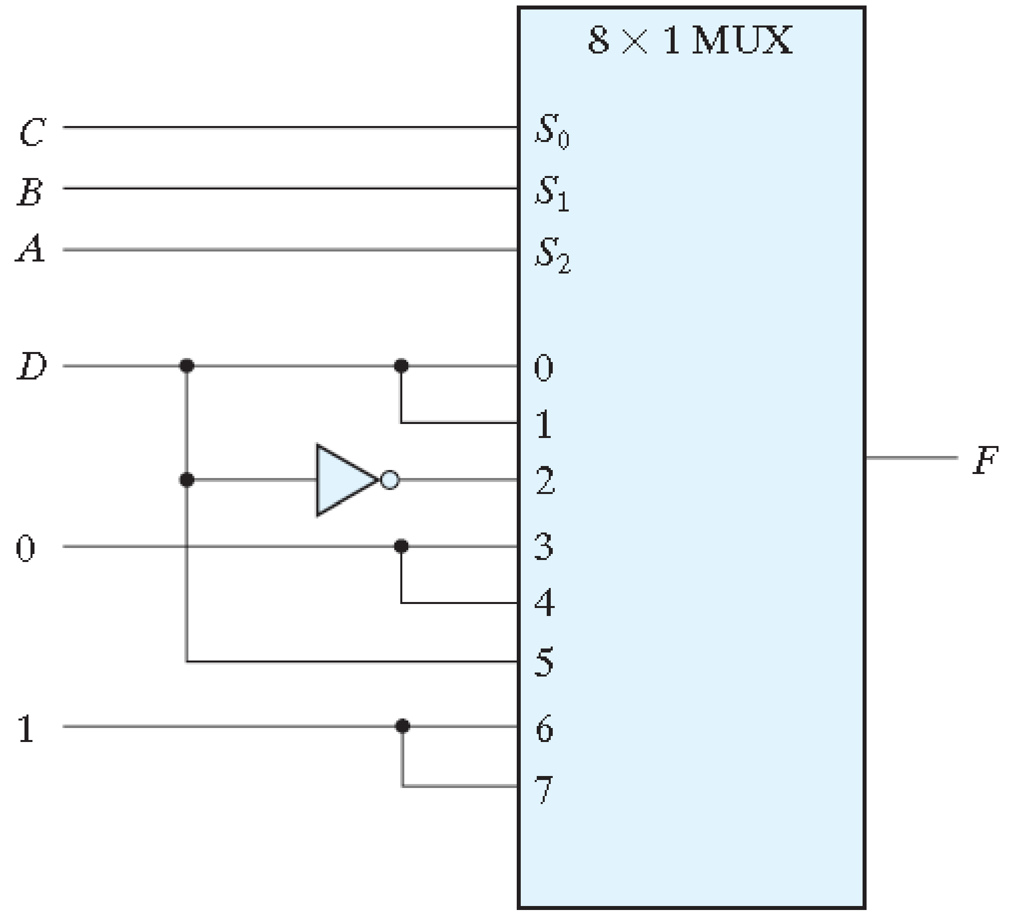
x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



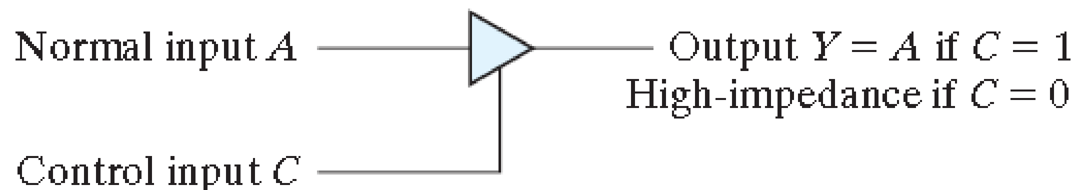
Another example: $F(A, B, C, D) = \Sigma\,(1, 3, 4, 11, 12, 13, 14, 15)$

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	



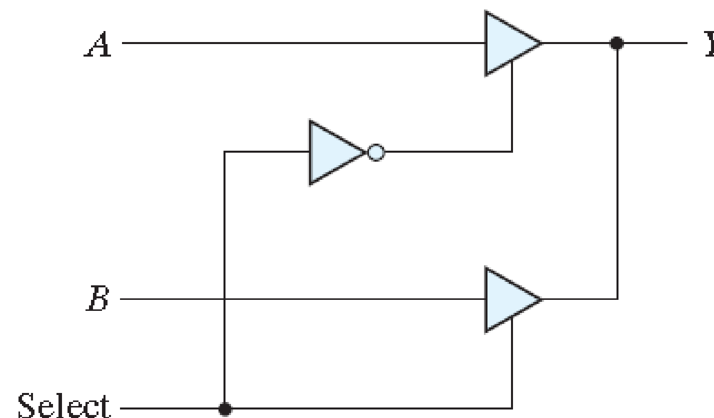
Three-state Gate

- The third state is a *high-impedance* state in which
  - (1) the logic behaves like an open circuit, which means that the output appears to be disconnected,
  - (2) the circuit has no logic significance, and
  - (3) the circuit connected to the output of the three-state gate is not affected by the inputs to the gate.
- The most commonly used is the three-state buffer gate. (*a.k.a.* tri-state buffer)

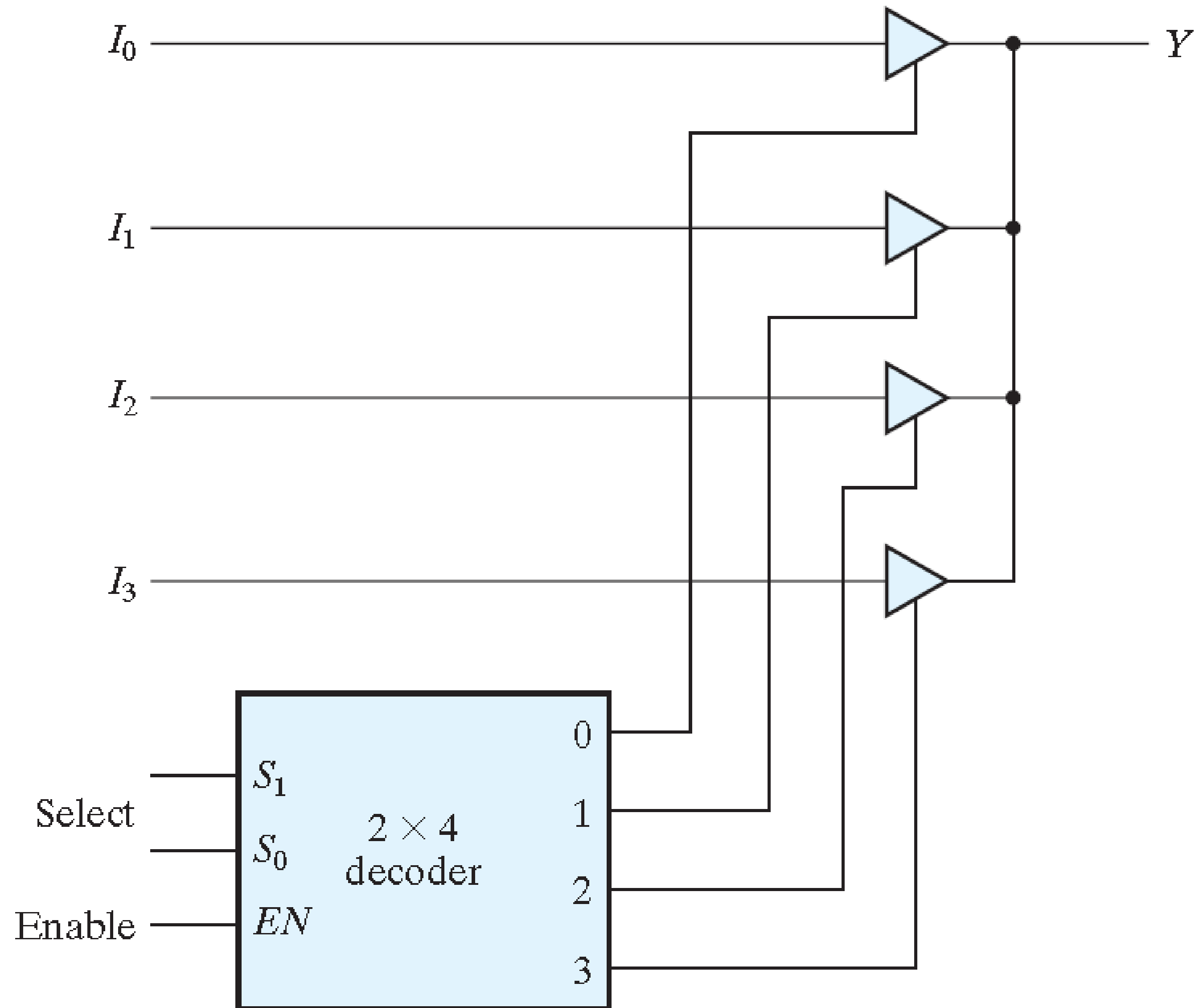


- A multiplexer can be constructed with three-state gates.
- Example: 2-to-1 multiplexer

  $Y = A$ if Select $= 0$, $Y = B$ if Select $= 1$



4-to-1 Multiplexer using Three-state Buffer



Homework #4

- 4.2
- 4.4 (a)
- 4.9
- 4.16
- 4.32 (a)