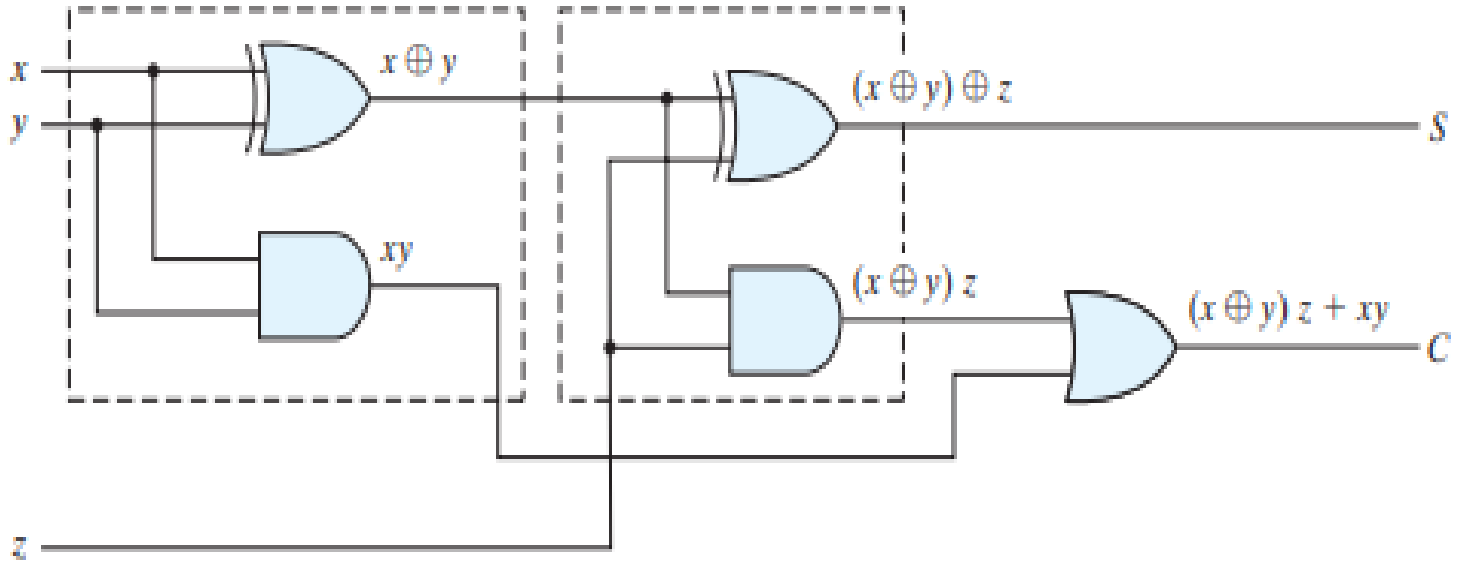
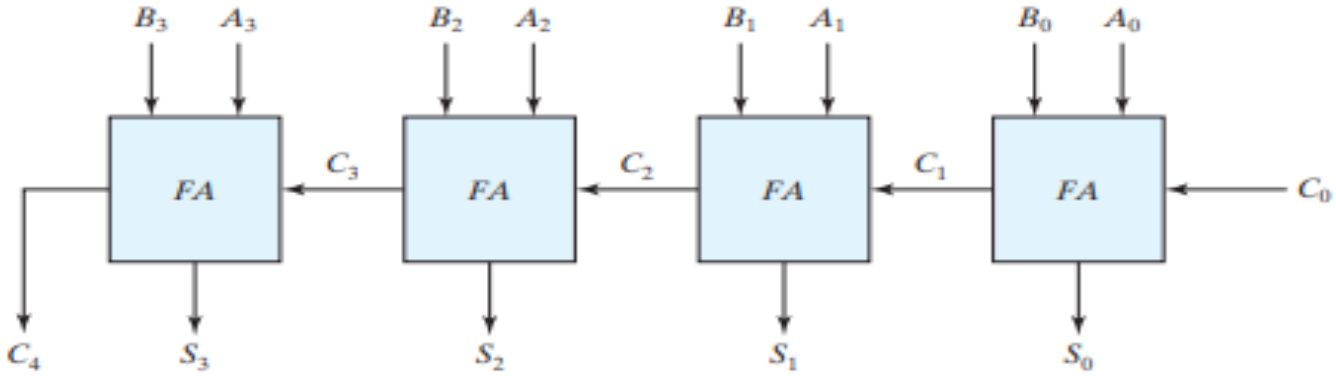
CHAPTER 4

Binary Adder

- A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It 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.



- The interconnection of four full-adder (FA) circuits to provide a four-bit binary ripple carry adder.

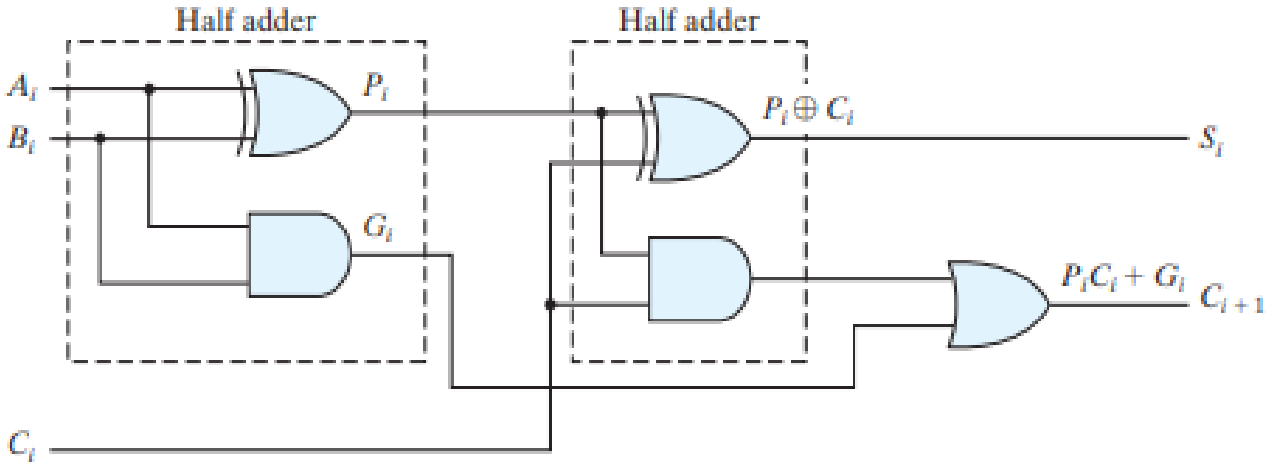


Subscript *i*:	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

- The carries are connected in a chain through the full adders.
- The input carry to the adder is C0 , and it ripples through the full adders to the output carry C4.
- The S outputs generate the required sum bits.
- An n -bit adder requires n full adders, with each output carry connected to the input carry of the next higher order full adder.

Carry Propagation

- Full adder with P and G



- An obvious solution for reducing the carry propagation delay time is to employ faster gates with reduced delays.
- Another solution is to increase the complexity of the equipment in such a way that the carry delay time is reduced.
- There are several techniques for reducing the carry propagation time in a parallel adder.
- The most widely used technique employs the principle of carry lookahead logic.

- We define two new binary variables

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

- The output sum and carry can respectively be expressed as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

- G_i is called a carry generate , and it produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i .
- P_i is called a carry propagate , because it determines whether a carry into stage i will propagate into stage i + 1.

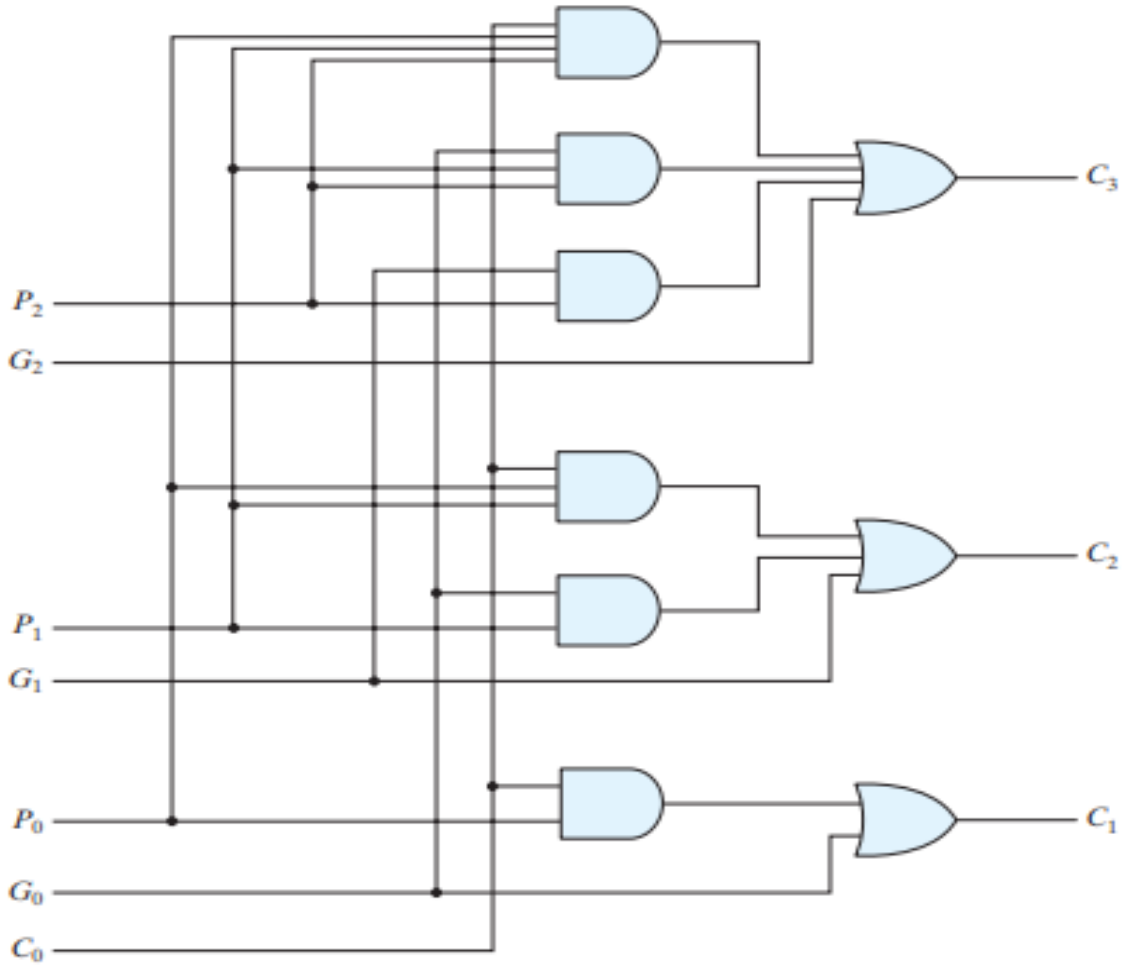
- The Boolean functions for the carry outputs of each stage and substitute the value of each C_i from the previous equations:

$$C_0 = \text{input carry}$$
$$C_1 = G_0 + P_0C_0$$

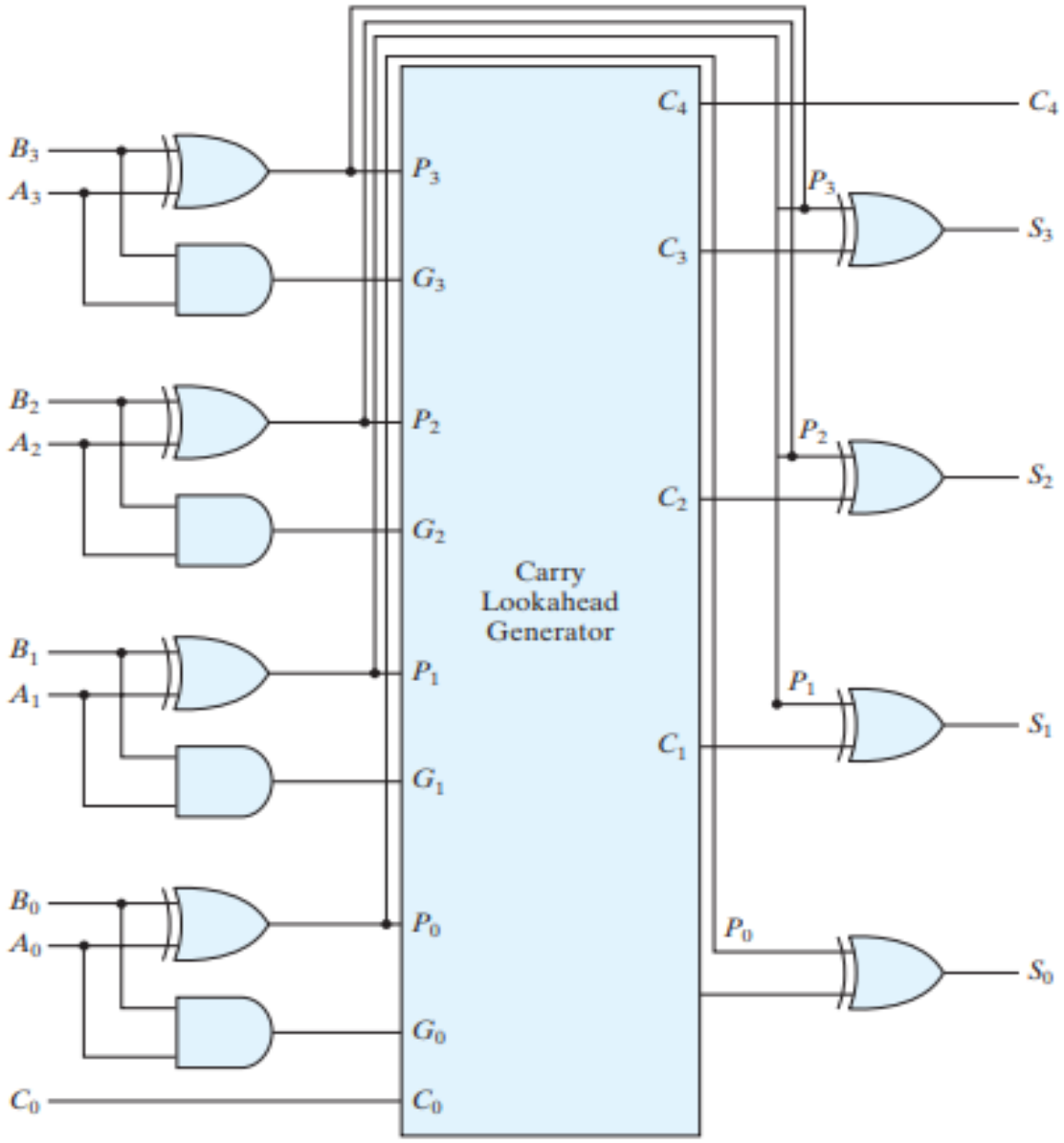
$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$

$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 = P_2P_1P_0C_0$$

Logic diagram of carry lookahead generator



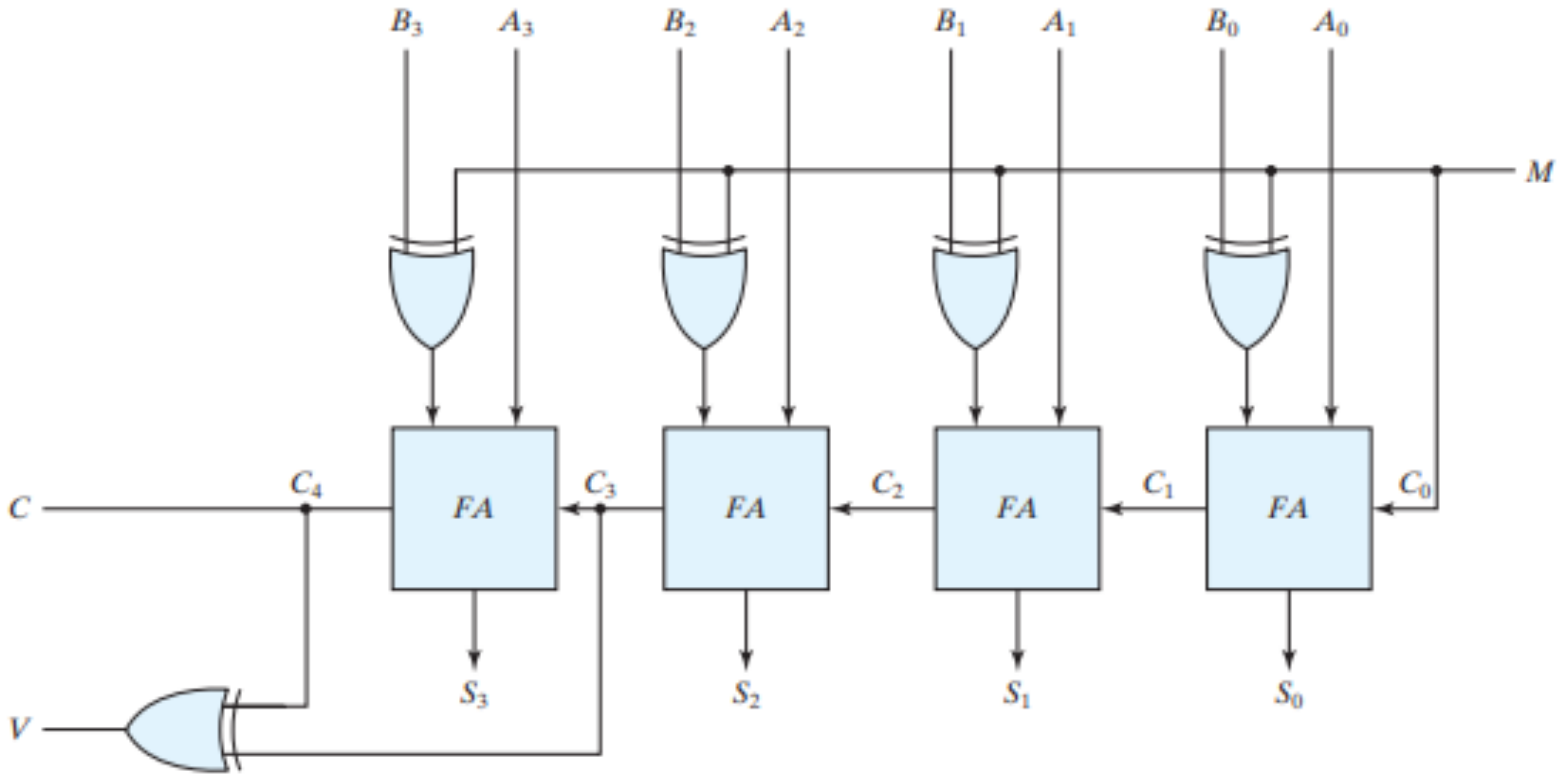
Four-bit adder with carry lookahead



Binary Subtractor

- The subtraction A - B can be done by taking the 2's complement of B and adding it to A.

Four-bit adder–subtractor (with overflow detection)



DECIMAL ADDER

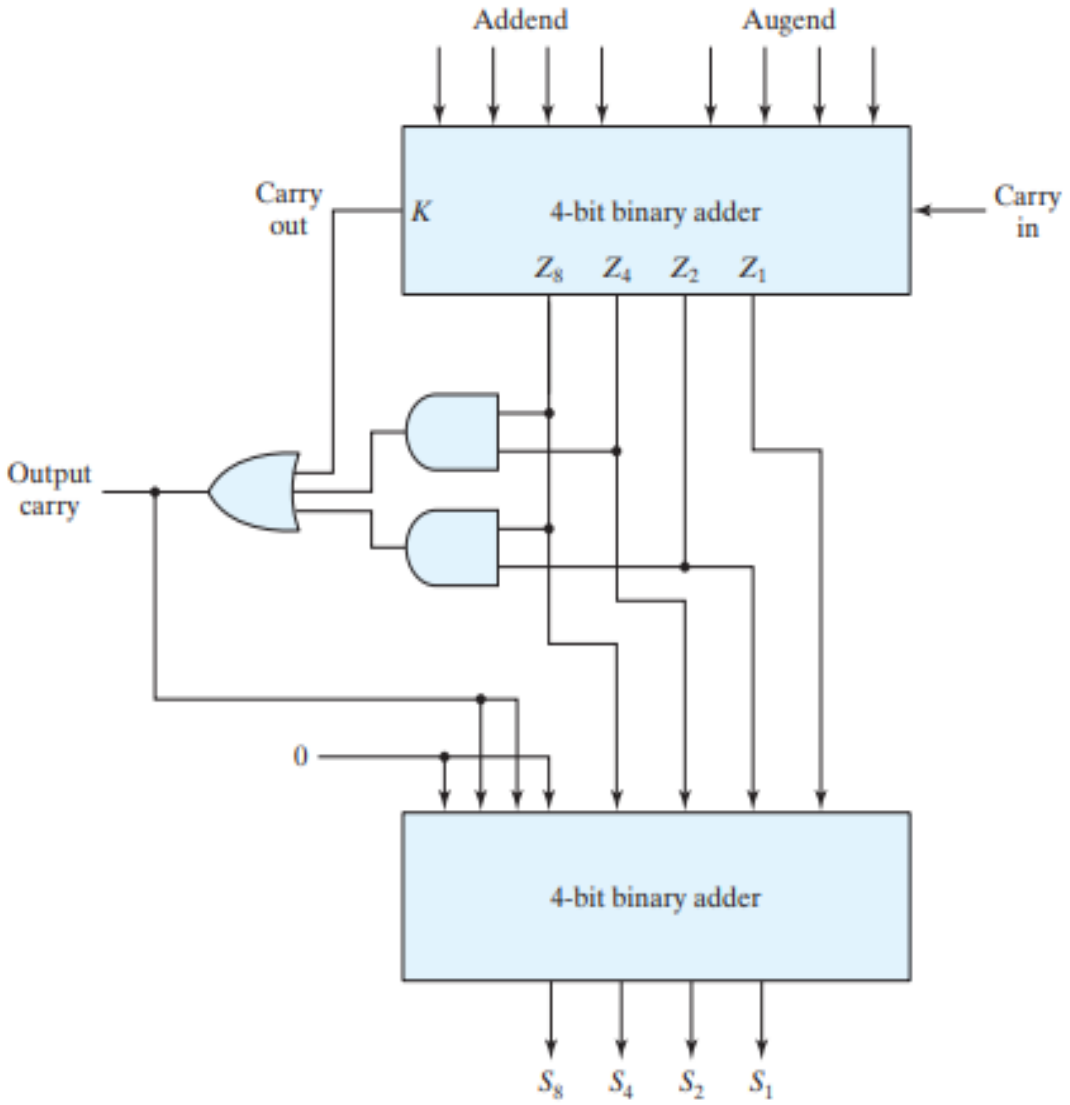
- Computers or calculators that perform arithmetic operations directly in the decimal number system represent decimal numbers in binary coded form.
- An adder for such a computer must employ arithmetic circuits that accept coded decimal numbers and present results in the same code.

BCD Adder

- Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 9 + 9 + 1 = 19, the 1 in the sum being an input carry.
- When the binary sum is equal to or less than 1001, the corresponding BCD number is identical, and therefore no conversion is needed.
- When the binary sum is greater than 1001, we obtain an invalid BCD representation. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.

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



Block diagram of a BCD adder

- The condition for a correction and an output carry can be expressed by the Boolean function $C = K + Z_8Z_4 + Z_8Z_2$.
- When C = 1, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage.

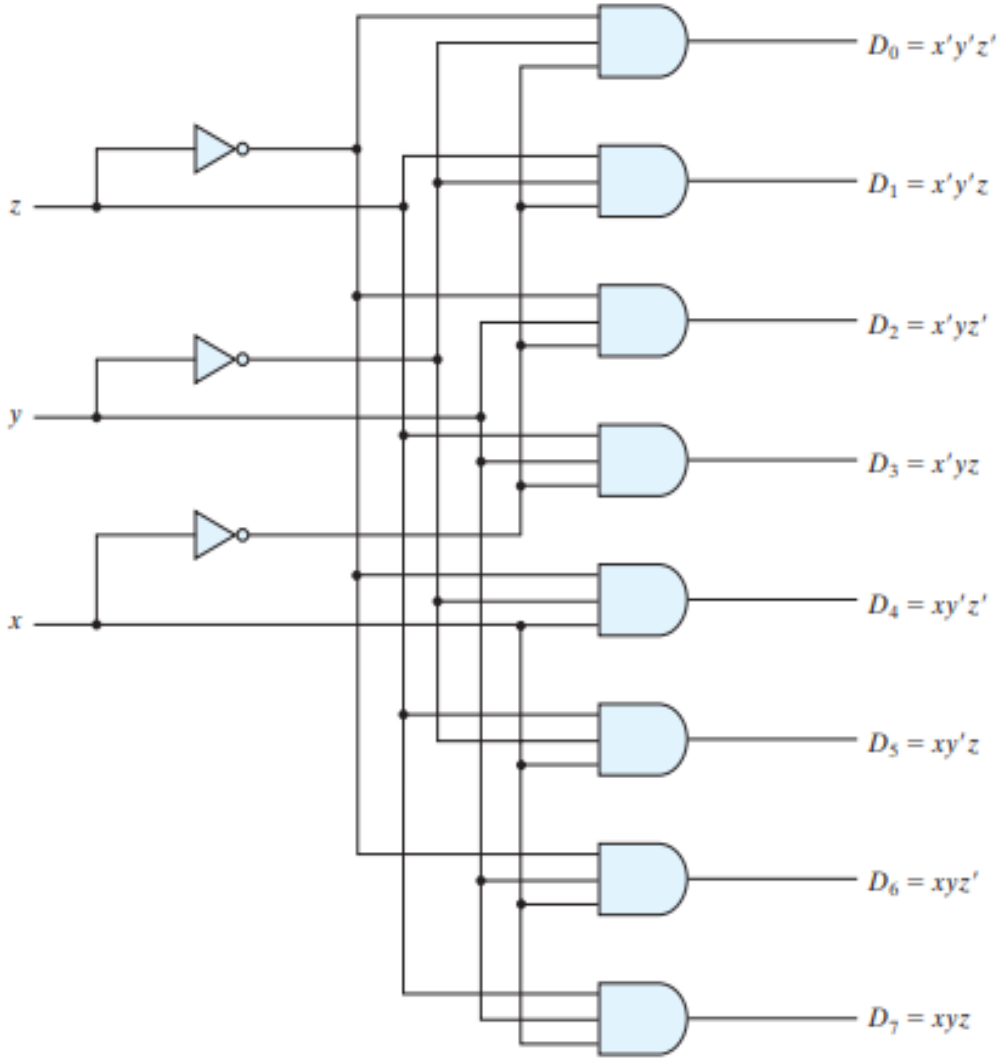
DECODERS

- A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.
- The decoders presented here are called n -to- m -line decoders, where $m \leq 2^n$.
- Their purpose is to generate the 2^n minterms of n input variables.
- Each combination of inputs will assert a unique output.

Three-to-eight-line decoder

- The three inputs are decoded into eight outputs, each representing one of the minterms of the three input variables.

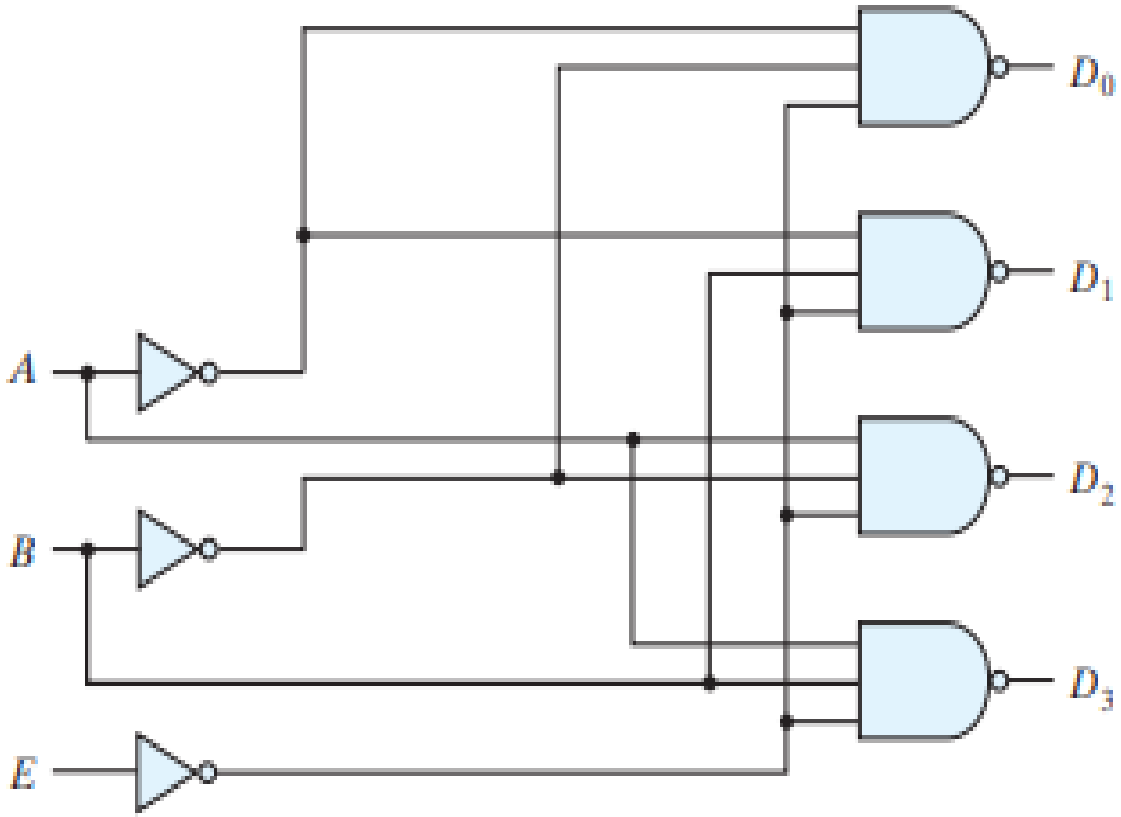
Three-to-eight-line decoder



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

A two-to-four-line decoder with an enable input constructed with NAND gates

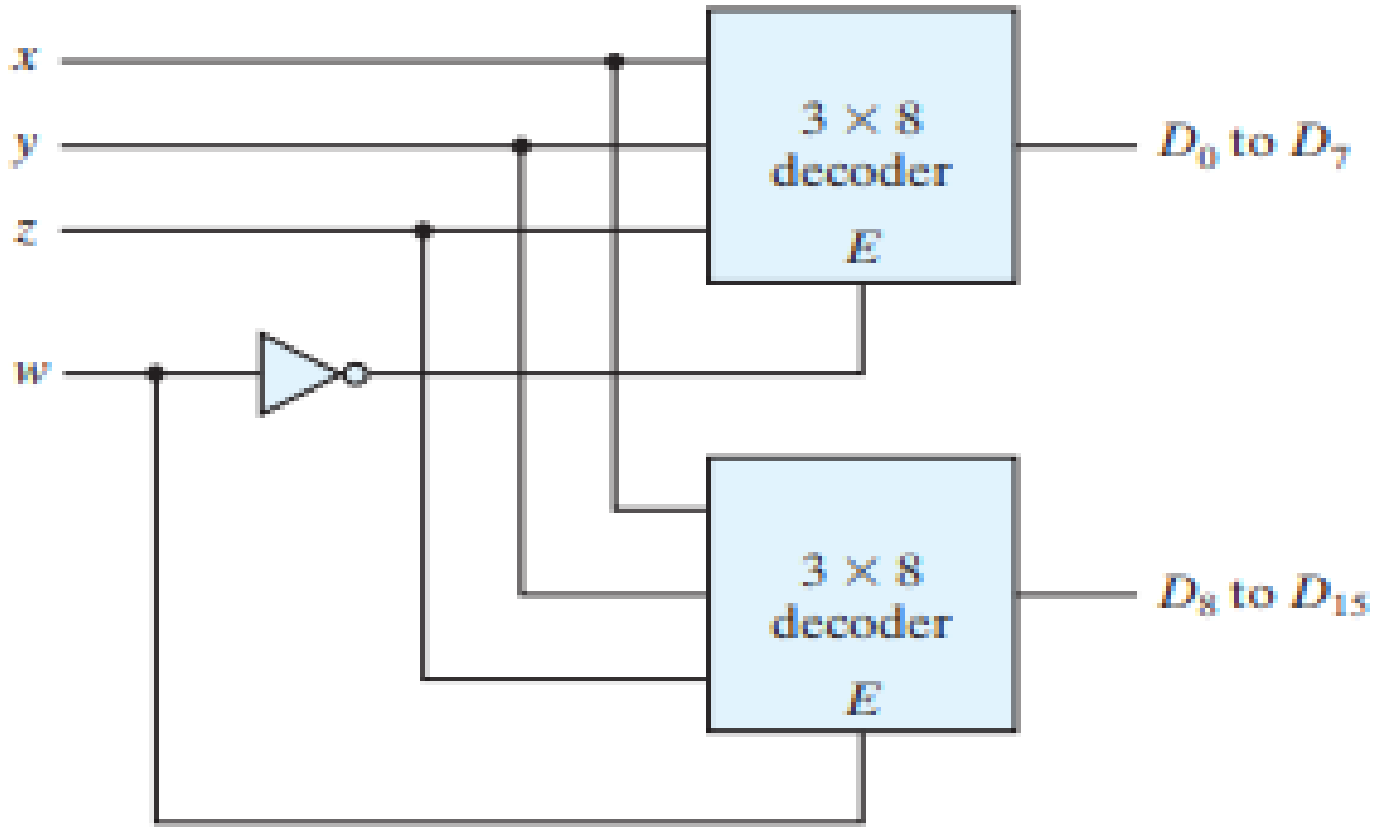


(a) Logic diagram

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

4 ×16 decoder constructed with two 3 × 8 decoders

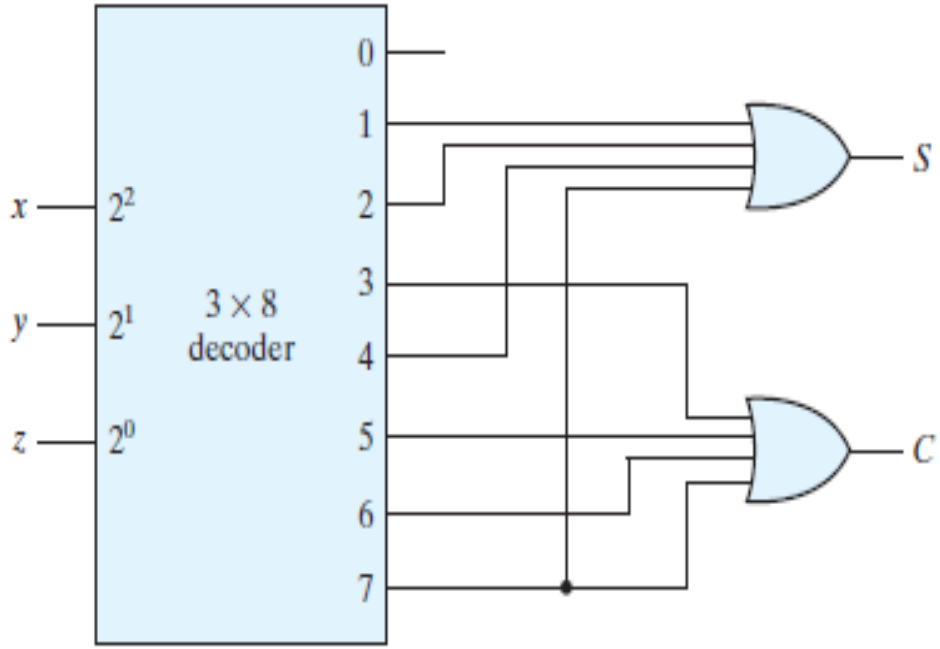


Combinational Logic Implementation

- Any combinational circuit with n inputs and *m* outputs can be implemented with an *n* -to-2^n -line decoder and *m* OR gates.
- The procedure for implementing a combinational circuit by means of a decoder and OR gates requires that
  - The Boolean function for the circuit be expressed as a sum of minterms.
  - A decoder is then chosen that generates all the minterms of the input variables.
  - The inputs to each OR gate are selected from the decoder outputs according to the list of minterms of each function.

Ex: full adder

- $S(x, y, z) = \Sigma(1, 2, 4, 7)$
- $C(x, y, z) = \Sigma(3, 5, 6, 7)$



ENCODERS

- An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has 2^n (or fewer) input lines and n output lines.

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

- The encoder can be implemented with OR gates whose inputs are determined directly from the truth table.

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

- The encoder can be implemented with three OR gates.
- If two inputs are active simultaneously, the output produces an undefined combination.
- For example, if *D*3 and *D*6 are 1 simultaneously, the output of the encoder will be 111.
- To resolve this ambiguity, encoder circuits must establish an input priority to ensure that only one input is encoded.

Priority Encoder

- A priority encoder is an encoder circuit that includes the priority function.
- The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

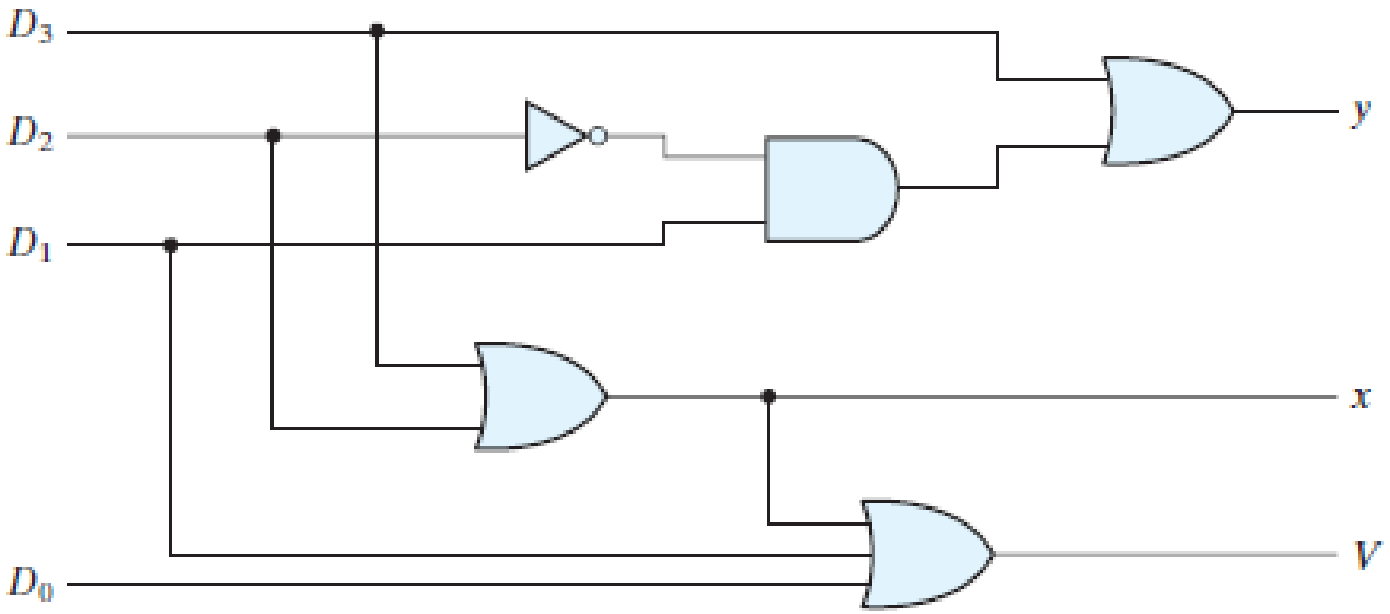
Truth Table of a Priority Encoder

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

- In addition to the two outputs x and y , the circuit has a third output designated by V ; this is a *valid* bit indicator that is set to 1 when one or more inputs are equal to 1.
- If all inputs are 0, there is no valid input and V is equal to 0.
- The other two outputs are not inspected when V equals 0 and are specified as don't-care conditions.

- Boolean functions

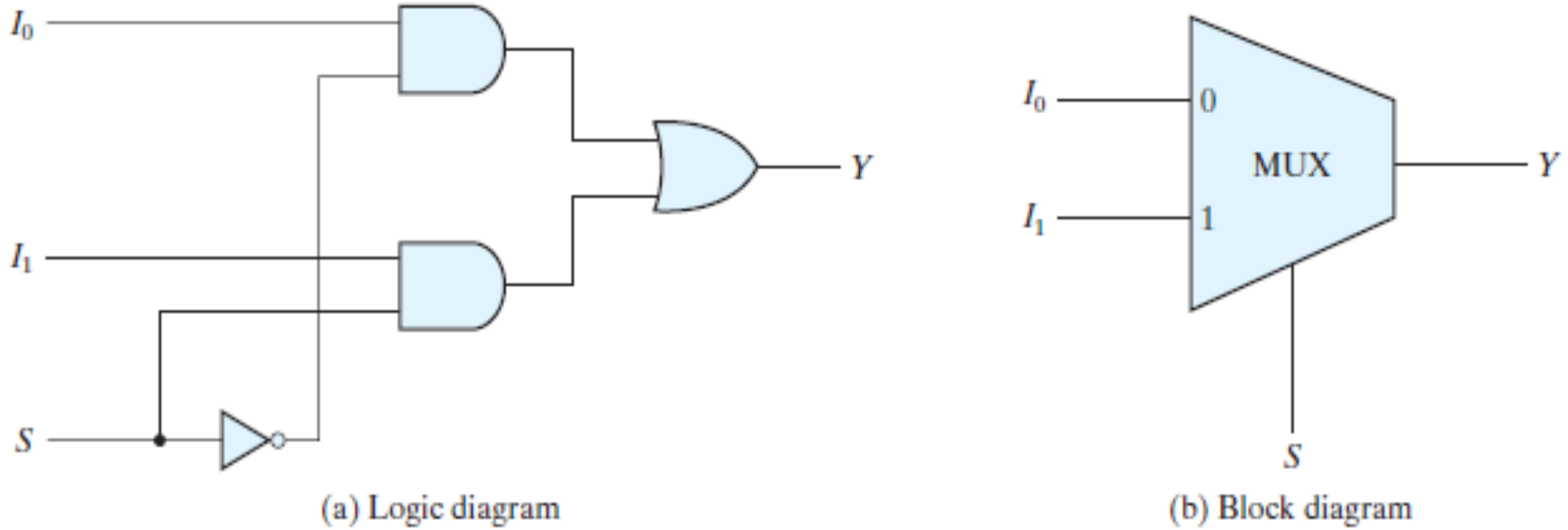
$$x = D_2 + D_3$$
$$y = D_3 + D_1 D_2'$$
$$V = D_0 + D_1 + D_2 + D_3$$



MULTIPLEXERS

- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- The selection of a particular input line is controlled by a set of selection lines.
- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

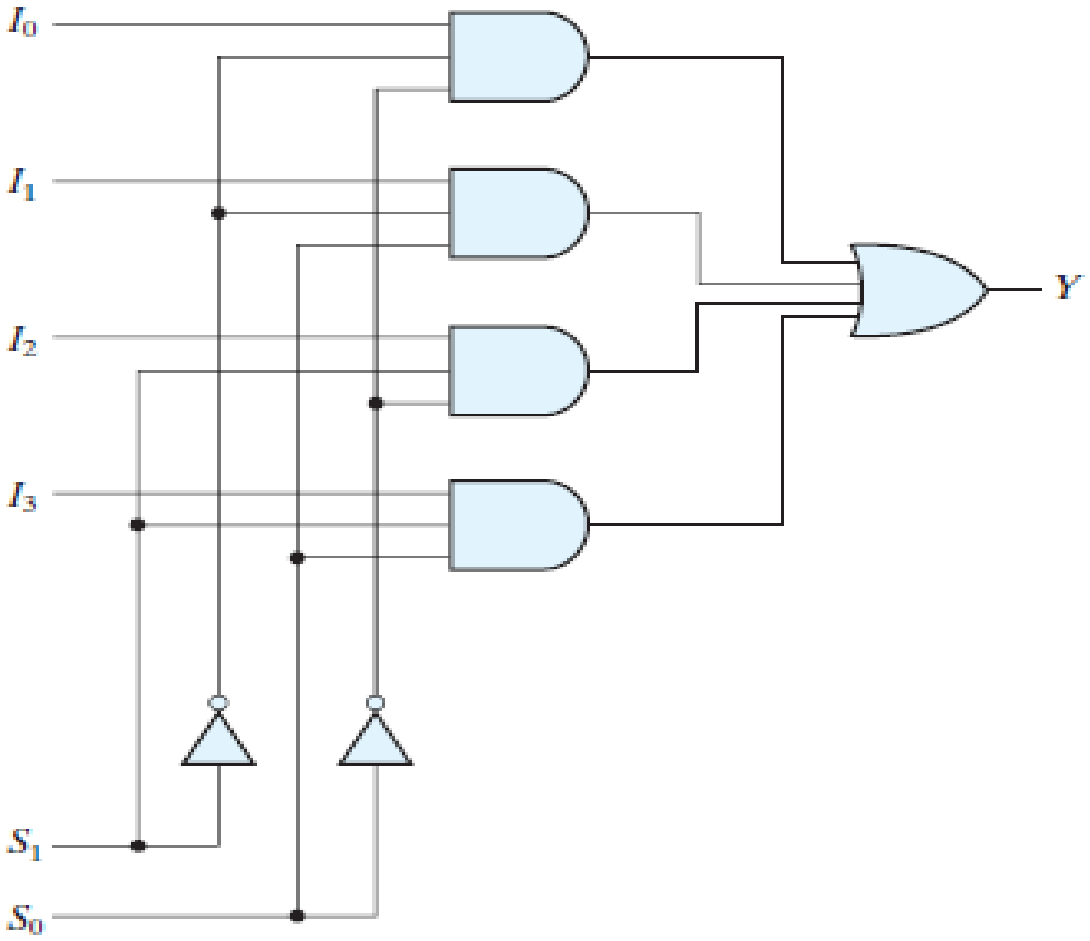
Two-to-one-line multiplexer



(a) Logic diagram

(b) Block diagram

Four-to-one-line multiplexer

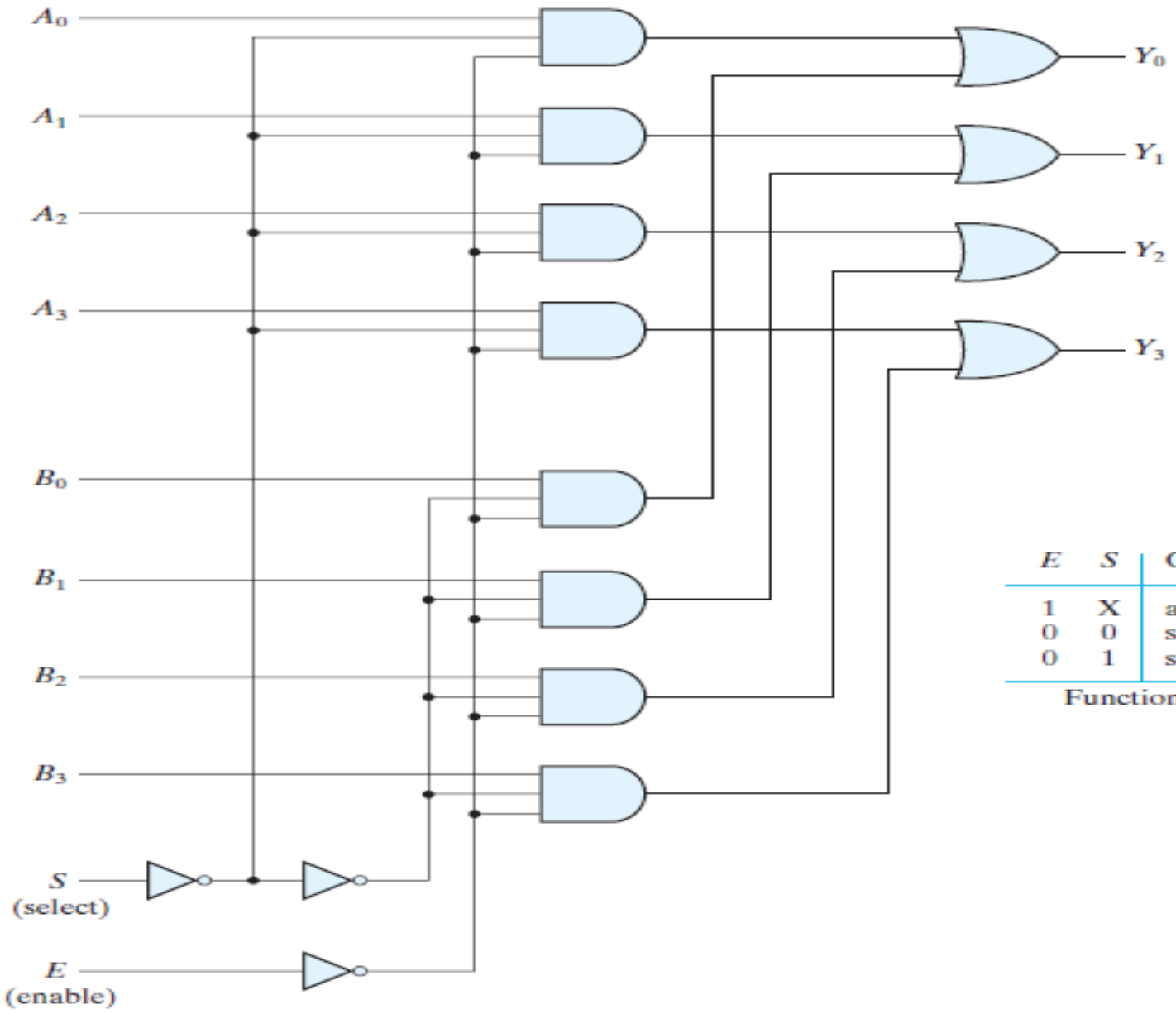


(a) Logic diagram

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

(b) Function table

Quadruple two-to-one-line multiplexer



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

Function table

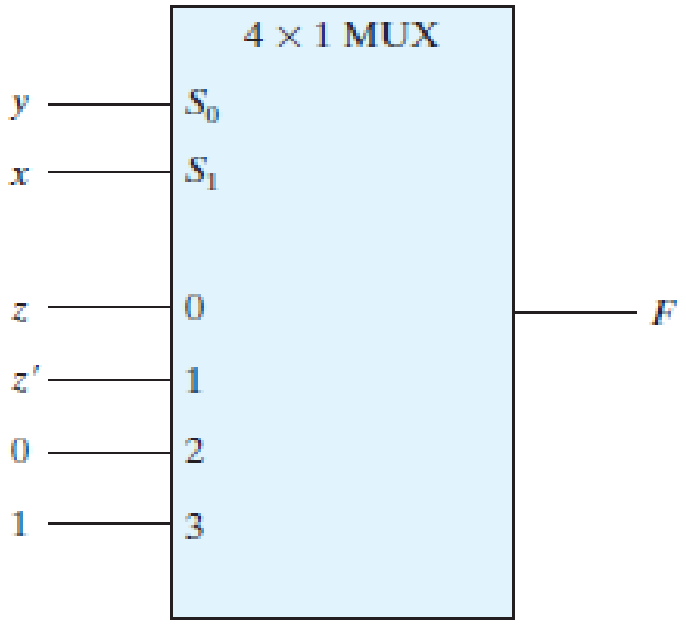
Boolean Function Implementation

- Method for implementing a Boolean function of *n* variables with a multiplexer that has *n* - 1 selection inputs.
- The first *n* - 1 variables of the function are connected to the selection inputs of the multiplexer.

Ex: consider the Boolean function: $F(x, y, z) = \Sigma(1, 2, 6, 7)$

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

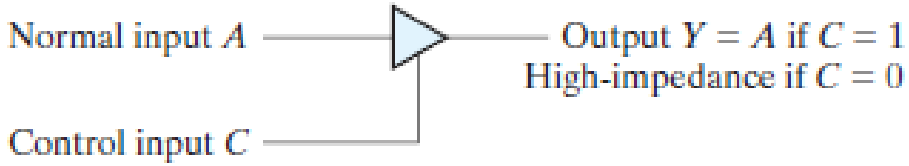


(b) Multiplexer implementation

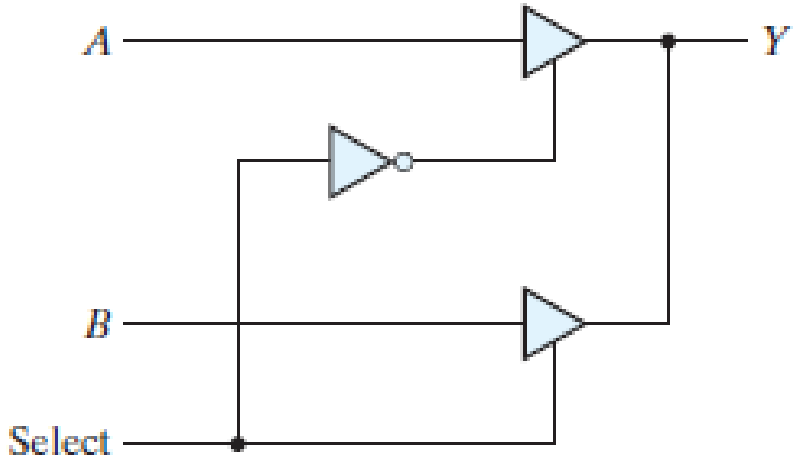
Three-State Gates

- A multiplexer can be constructed with three-state gates—digital circuits that exhibit three states.
- Two of the states are signals equivalent to logic 1 and logic 0 as in a conventional 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.

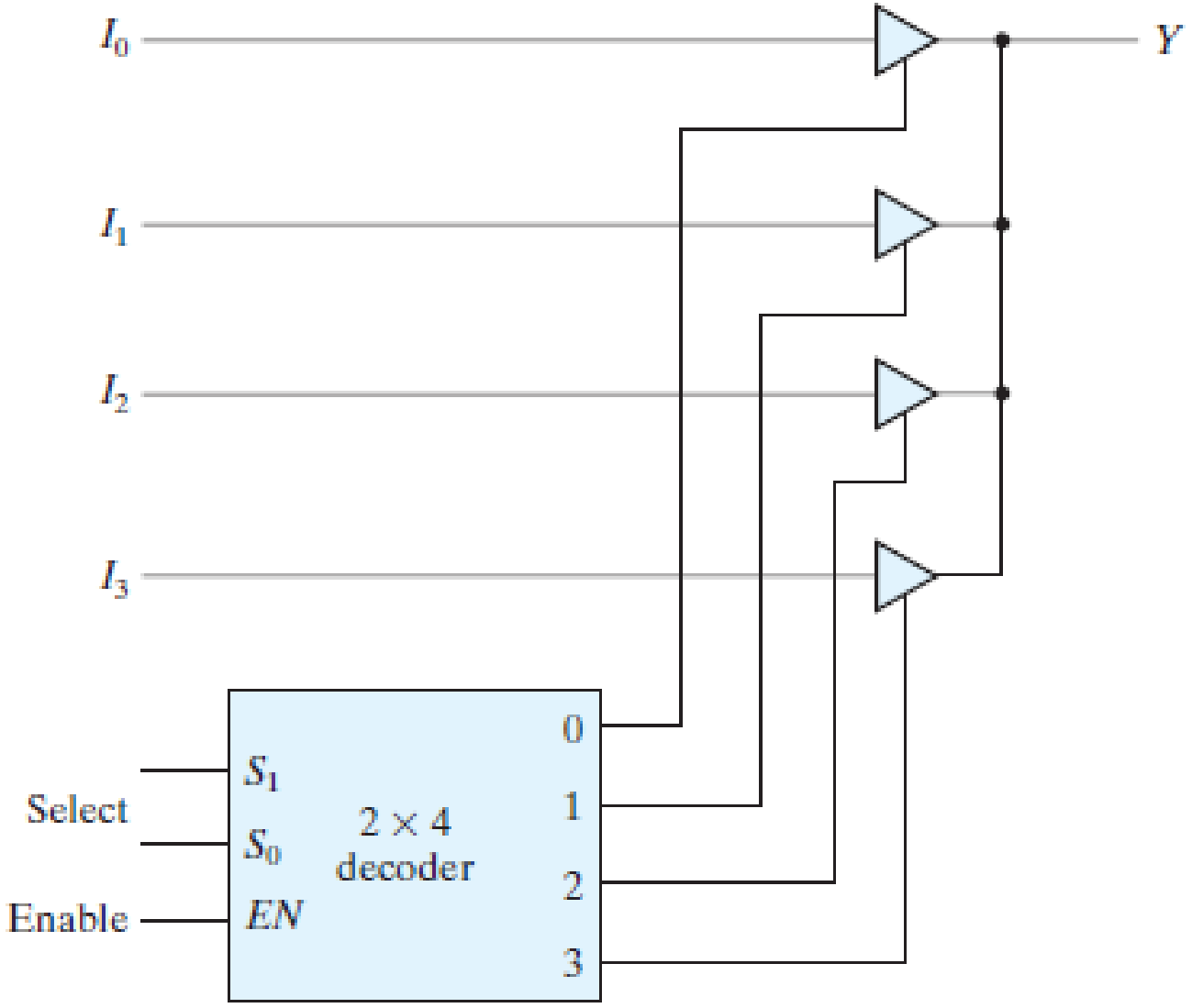
- The graphic symbol for a three-state buffer gate



The construction of multiplexers with three-state buffers



(a) 2-to-1-line mux



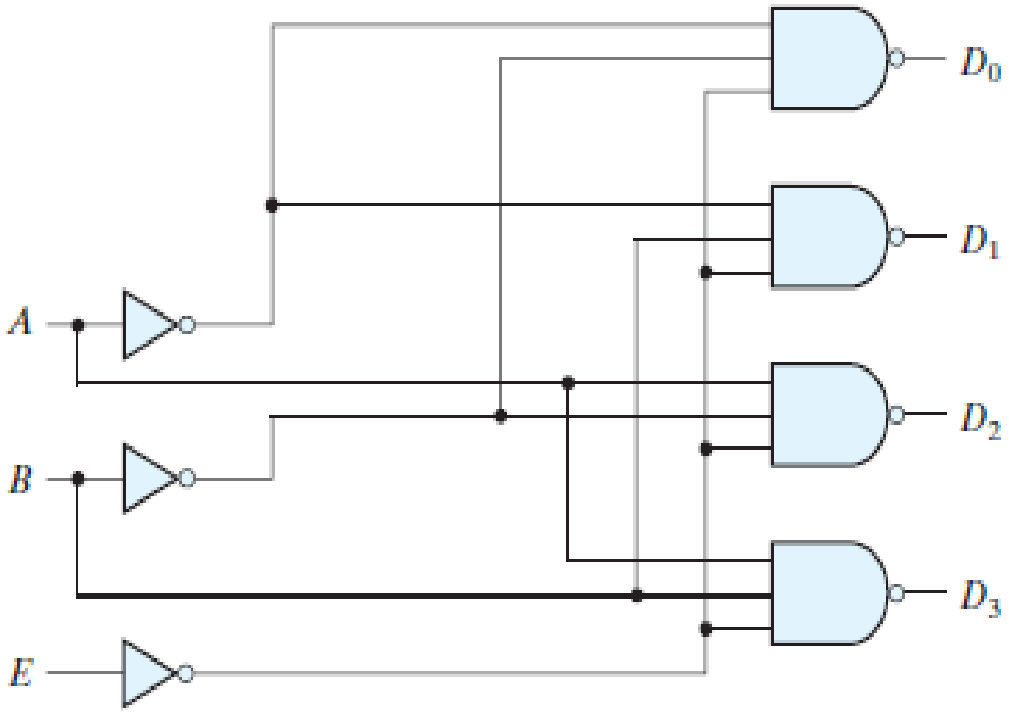
(b) 4-to-1-line mux

HDL MODELS OF COMBINATIONAL CIRCUITS

- The logic of a module can be described in any one (or a combination) of the following modeling styles:
  - Gate-level modeling using instantiations of predefined and user-defined primitive gates.
  - Dataflow modeling using continuous assignment statements with the keyword **assign** .
  - Behavioral modeling using procedural assignment statements with the keyword **always** .

Gate-Level Modeling

- Verilog statements to specify two vectors:
  - **output** [0: 3] D;
  - **wire** [7: 0] SUM;



HDL Example 4.1 (Two-to-Four-Line Decoder)

```
// Gate-level description of two-to-four-line decoder
// Refer to Fig. 4.19 with symbol E replaced by enable, for clarity.

module decoder_2x4_gates (D, A, B, enable);
 output      [0: 3]          D;
 input                       A, B;
 input                       enable;
 wire                        A_not,B_not, enable_not;

 not
   G1  (A_not, A),
   G2  (B_not, B),
   G3  (enable_not, enable);
 nand
   G4  (D[0], A_not, B_not, enable_not),
   G5  (D[1], A_not, B, enable_not),
   G6  (D[2], A, B_not, enable_not),
   G7  (D[3], A, B, enable_not);
endmodule
```

Hierarchical design

- Two or more modules can be combined to build a hierarchical description of a design.
- There are two basic types of design methodologies:
  - top down and bottom up: the top-level block is defined and then the subblocks necessary to

    build the top-level block are identified.
  - *bottom-up* design: the building blocks are first identified and then combined to build the top-level block.
- Example, the binary adder. It can be considered as a top-block component built with four full-adder blocks, while each full adder is built with two half-adder blocks.
- In a top-down design, the four-bit adder is defined first, and then the two adders are described.
- In a bottom-up design, the half adder is defined, then each full adder is constructed, and then the four-bit adder is built from the full adders.

A bottom-up hierarchical description of a four-bit adder

```
// Gate-level description of four-bit ripple carry adder: Description of half adder ( Fig. 4.5b )
module half_adder (S, C, x, y);
output S, C;
input x, y;
// Instantiate primitive gates
xor (S, x, y);
and (C, x, y);
endmodule
// Description of full adder ( Fig. 4.8 )
 module full_adder (S, C, x, y, z);
output S, C;
input x, y, z;
wire S1, C1, C2;
// Instantiate half adders
half_adder  HA1 (S1, C1, x, y);
half_adder  HA2 (S, C2, S1, z);
or G1 (C, C2, C1);
endmodule
```

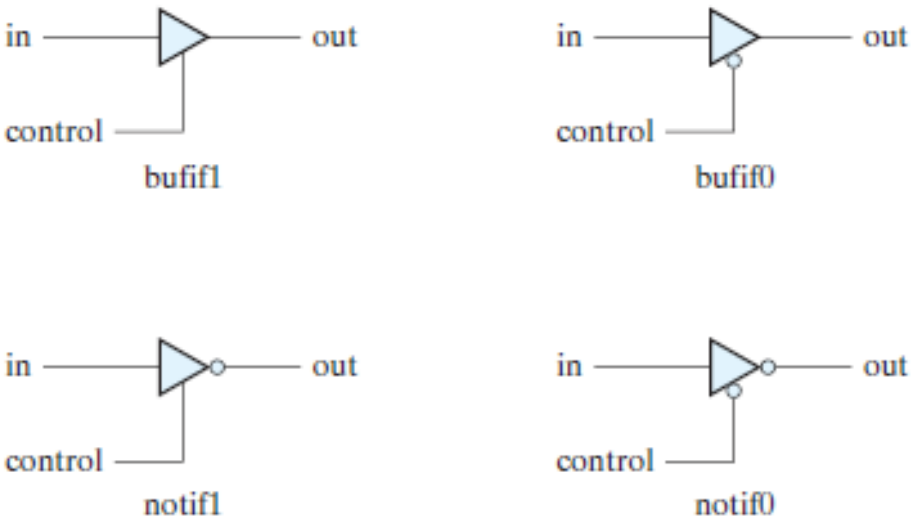
```
// Description of four-bit adder ( Fig. 4.9 )

module ripple_carry_4_bit_adder (Sum, C4, A, B, C0);
output [3: 0] Sum;
output C4;
input [3: 0] A, B;
input C0;
wire C1, C2, C3; // Intermediate carries
// Instantiate chain of full adders
full_adder    FA0 (Sum[0], C1, A[0], B[0], C0),
              FA1 (Sum[1], C2, A[1], B[1], C1),
              FA2 (Sum[2], C3, A[2], B[2], C2),
              FA3 (Sum[3], C4, A[3], B[3], C3);
endmodule
```

- Note that modules can be instantiated (nested) within other modules, but module declarations cannot be nested; that is, a module definition (declaration) cannot be placed within another module declaration.
- In other words, a module definition cannot be inserted into the text between the **module** and **endmodule** keywords of another module.
- The only way one module definition can be incorporated into another module is by instantiating it.

Three-State Gates

- The high-impedance state is symbolized by **z** in Verilog.
- There are four types of three-state gates

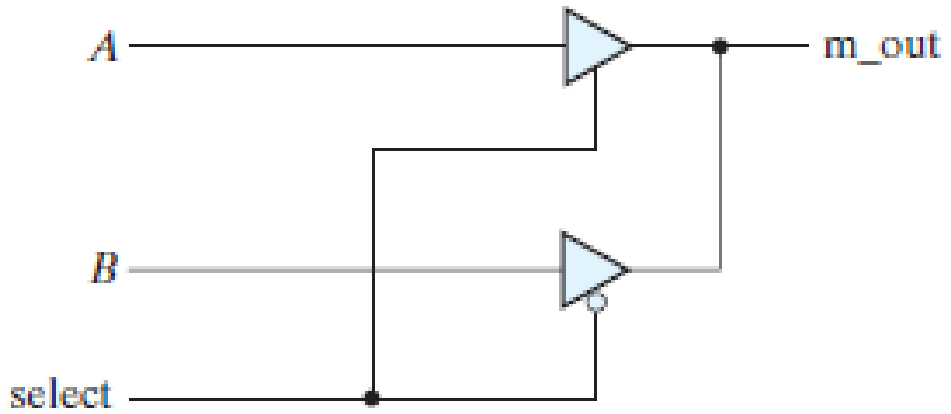
gate name (*output*, *input*, *control*);



- The HDL description must use a **tri** data type for the output
- In order to show that they have a common connection, it is necessary to declare *m_out* with the keyword **tri.**

```
module mux_tri (m_out, A, B, select);
  output  m_out;
  input   A, B, select;
  tri     m_out;

  bufif1 (m_out, A, select);
  bufif0 (m_out, B, select);
endmodule
```

Two-to-one-line multiplexer with three-state buffers

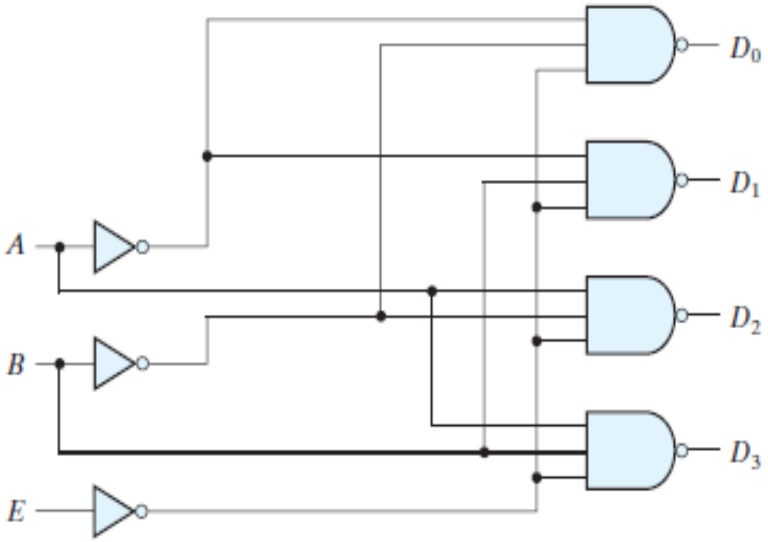


Dataflow Modeling

- Dataflow modeling of combinational logic uses a number of operators that act on binary operands to produce a binary result.
- Dataflow modeling uses continuous assignments and the keyword **assign.**

Some Verilog HDL Operators

Symbol	Operation	Symbol	Operation
+	binary addition		
−	binary subtraction		
&	bitwise AND	&&	logical AND
|	bitwise OR	||	logical OR
^	bitwise XOR		
~	bitwise NOT	!	logical NOT
= =	equality		
>	greater than		
<	less than		
{}	concatenation		
?:	conditional		




```
// Dataflow description of two-to-four-line decoder

// See Fig. 4.19. Note: The figure uses symbol E, but the
// Verilog model uses enable to clearly indicate functionality.

module decoder_2x4_df (                          // Verilog 2001, 2005 syntax
   output     [0: 3]          D,
   input                      A, B,
                              enable
);
   assign     D[0] = !((!A) && (!B) && (!enable)),
              D[1] = !(*!A) && B && (!enable)),
              D[2] = !(A && B && (!enable)
              D[3] = !(A && B && (!enable))
endmodule
```

Dataflow description of four-bit adder

- The additionlogic is described by a single statement using the operators of addition and concatenation.
- The plus symbol (+) specifies the binary addition of the four bits of *A* with the four bits of *B* and the one bit of *C in* .
- The target output is the *concatenation* of the output carry *C_out* and the four bits of *Sum*.
- Concatenation of operands is expressed within braces and a comma separating the operands.
- Thus, *{C_out, Sum}* represents the five-bit result of the addition operation.

```
module binary_adder (
   output [3: 0]          Sum,
   output                 C_out,
   input [3: 0]           A, B,
   input                  C_in
);

   assign {C_out, Sum} = A + B + C_in;
endmodule
```

The conditional operator

- This operator takes three operands:

  condition ? true-expression : false-expression;

- The condition is evaluated. If the result is logic 1, the true expression is evaluated and used to assign a value to the left-hand side of an assignment statement.
- If the result is logic 0, the false expression is evaluated.
- The two conditions together are equivalent to an if–else condition.
- The continuous assignment **assign** OUT select ? A : B; specifies the condition that *OUT = A* if *select* = 1, else *OUT = B* if *select* =0.

```
// Dataflow description of two-to-one-line multiplexer

module mux_2x1_df(m_out, A, B, select);
  output        m_out;
  input         A, B;
  input         select;

  assign m_out = (select)? A : B;
endmodule
```

Behavioral Modeling

- Behavioral modeling represents digital circuits at a functional and algorithmic level.
- Behavioral descriptions use the keyword **always** , followed by an optional event control expression and a list of procedural assignment statements.
- The target output of a procedural assignment statement must be of the **reg** data type.
- The target output of an assignment may be continuously updated, a **reg** data type retains its value until a new value is assigned.

```
// Behavioral description of two-to-one-line multiplexer

module mux_2x1_beh (m_out, A, B, select);
  output      m_out;
  input       A, B, select;
  reg         m_out;

  always      @(A or B or select)
    if (select == 1) m_out = A;
    else m_out 5 B;
endmodule
```

Behavioral description of four-to-one line multiplexer

```
module mux_4x1_beh
( output reg m_out,
  input      in_0, in_1, in_2, in_3,
  input [1: 0] select
);
always @ (in_0, in_1, in_2, in_3, select)      // Verilog 2001, 2005 syntax
  case (select)
    2'b00:          m_out = in_0;
    2'b01:          m_out = in_1;
    2'b10:          m_out = in_2;
    2'b11:          m_out = in_3;
  endcase
endmodule
```