Chapter 4
Combinational Logic

Combinational Circuit Analysis

The University Of Alabama in Huntsville Computer Science

Procedure

cs309

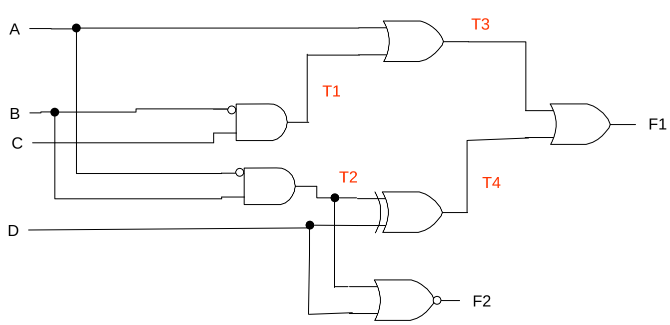
1. Assign a label to all internal signals.
2. For each labeled signal that is the output of a gate, write the function of the gate's inputs
3. Substitute functions of labels until you have the circuit output(s) as a direct function of the circuit inputs.

The University Of Alabama in Huntsville Computer Science

Circuit Analysis Example (1)

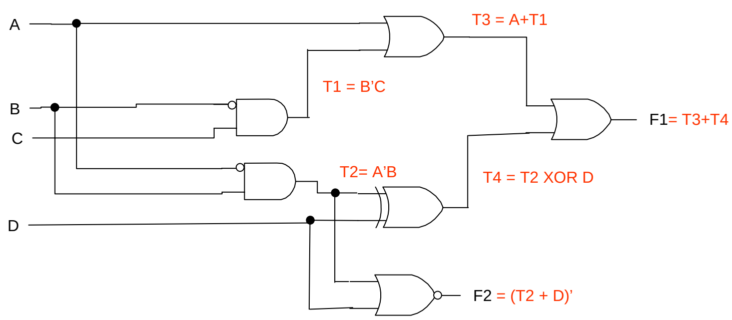
cs309

Assign internal labels



Circuit Analysis Example (2)

Write functions



Circuit Analysis Example (3)

Substitute

T1 = B'C

T2 = A'B

T3 = A + T1
= A + B'C

T4 = T2 XOR D
= T2D' + T2'D
= (A'B)D' + (A'B)'D
= A'BD' + AD + B'D

F1 = T3 + T4
= A + B'C + A'BD' + AD + B'D
= A + B'C + BD' + B'D

F2 = (T2 + D)'
= (A'B + D)'
= (A'B)'D'
= AD' + B'D'

Combinational Circuit Synthesis

The University Of Alabama in Huntsville Computer Science

Procedure

cs309

1. Determine inputs and outputs
2. Truth Table(s)
3. Simplify to desired form
4. Draw logic diagram
5. Verify

G. W. Cox – Spring 2010

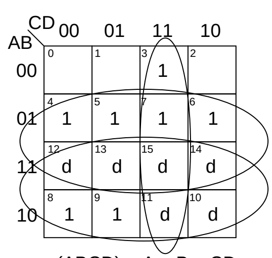
Example: BCD-XS5 Code Converter (1)

- Design a circuit that converts a BCD-coded input to the equivalent Excess-5 code

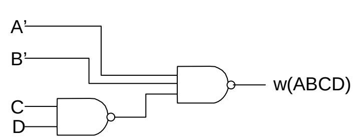
- From the description,
  - Inputs = ABCD
  - Outputs = wxyz

A B C D	w x y z
0 0 0 0	0 1 0 1
0 0 0 1	0 1 1 0
0 0 1 0	0 1 1 1
0 0 1 1	1 0 0 0
0 1 0 0	1 0 0 1
0 1 0 1	1 0 1 0
0 1 1 0	1 0 1 1
0 1 1 1	1 1 0 0
1 0 0 0	1 1 0 1
1 0 0 1	1 1 1 0
1 0 1 0	d d d d
1 0 1 1	" "
1 1 0 0	" "
1 1 0 1	" "
1 1 1 0	" "
1 1 1 1	" "

Example: BCD-XS5 Code Converter (2)



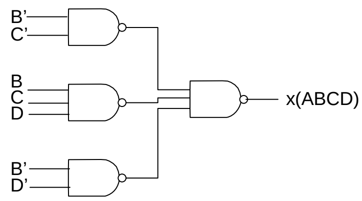
w(ABCD) = A + B + CD



Example: BCD-XS5 Code Converter (3)

AB \ CD	00	01	11	10
00	1	1		1
01			1	
11	d	d	d	d
10	1	1	d	d

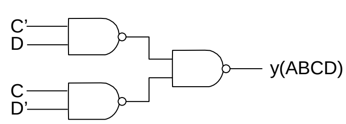
x(ABCD) = B'C' + BCD + B'D'



Example: BCD-XS5 Code Converter (4)

AB \ CD	00	01	11	10
00		1		1
01		1		1
11	d	d	d	d
10		1	d	d

y(ABCD) = C'D + CD'



Example: BCD-XS5 Code Converter (5)

AB \ CD	00	01	11	10
00	1			1
01	1			1
11	d	d	d	d
10	1		d	d

z(ABCD) = D'

D' ———— z(ABCD)

The University Of Alabama in Huntsville Computer Science

Simplifying multiple-output circuits

cs309

- Note that if you are designing a circuit with multiple outputs, it may be possible to form terms in common between the output functions so that the overall circuit is simpler

x \ yz	00	01	11	10
0			1	1
1				

f1(xyz) = x'y

x \ yz	00	01	11	10
0	1		1	1
1		1	1	

f2(xyz) = x'y' + yz + xz

x \ yz	00	01	11	10
0			1	1
1				

f1(xyz) = x'y

x \ yz	00	01	11	10
0	1		1	1
1		1	1	

f2(xyz) = x'y' + x'y + xz
= x'y + xz + f1(xyz)

Adder circuits:
Ripple-Carry Adders

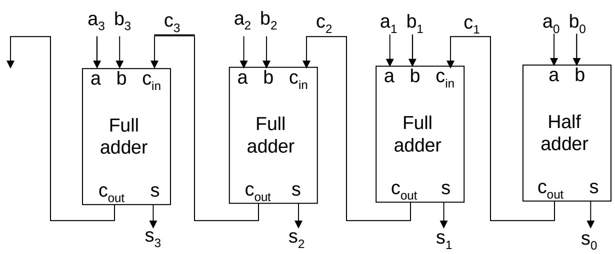
The University Of Alabama in Huntsville Computer Science

Ripple-Carry Adder (1)

cs309

"Full Addition" (3 bits) → c_1 ← "Half Addition" (2 bits)

c_1	
a_1	a_0
b_1	b_0
---------	
s_1	s_0



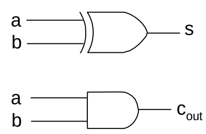
G. W. Cox – Spring 2010

Ripple-Carry Adder (2)

Half adder

a	b	c_{out}	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

s = a XOR b
c = ab



The University Of Alabama in Huntsville Computer Science

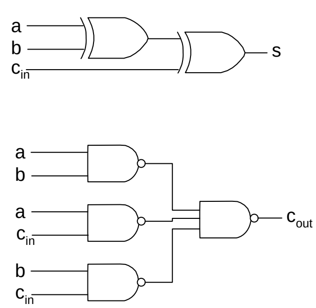
Ripple-Carry Adder (3)

cs309

Full adder

a	b	c_{in}	c_{out}	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 = a XOR b XOR c_{in}
c = ab + ac_{in} + bc_{in}



G. W. Cox – Spring 2010

A problem with ripple-carry


```
 01111…1111111
+00000…0000001
--------------
 10000…0000000
```

In this addition,
the carry generated in bit 0 propagates to the bit 1 position, where
a carry is then generated and propagates to the bit 2 position, where
a carry is then generated and propagates to the bit 3 position, where
…

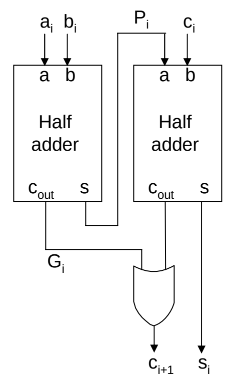
For long words, ripple-carry is slow.

Carry-Lookahead Adders

Faster adders – Carry Lookahead (1)

cs309

Note that we could build a Full Adder like this:



Define:

$P_i = A_i$ XOR B_i

$G_i = A_iB_i$

Then

$S_i = P_i$ XOR c_i
and
$c_{i+1} = G_i + P_ic_i$

The University Of Alabama in Huntsville Computer Science

Gate-level implementation

cs309

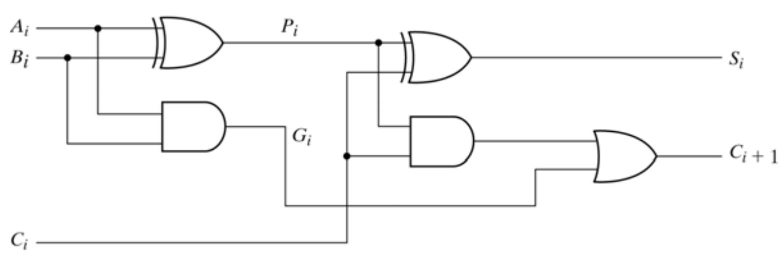


Fig. 4-10 Full Adder with P and G Shown

The University Of Alabama in Huntsville Computer Science

Faster adders – Carry Lookahead (2)

cs309

We have $c_{i+1} = G_i + P_i c_i$, so:

c_0 = input carry

$c_1 = G_0 + P_0 c_0$

$c_2 = G_1 + P_1 c_1$
$= G_1 + P_1 G_0 + P_1 P_0 c_0$

$c_3 = G_2 + P_2 c_2$
$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

…

Since P_i and G_i are both calculated directly from A_i and B_i, there is no “ripple”

Carry-lookahead generator

cs309

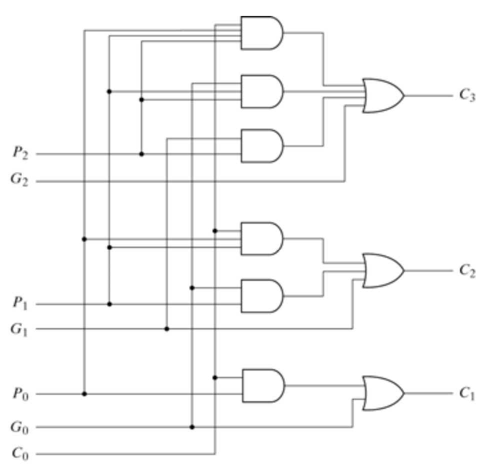


Fig. 4-11 Logic Diagram of Carry Lookahead Generator

4-bit Carry Lookahead Adder

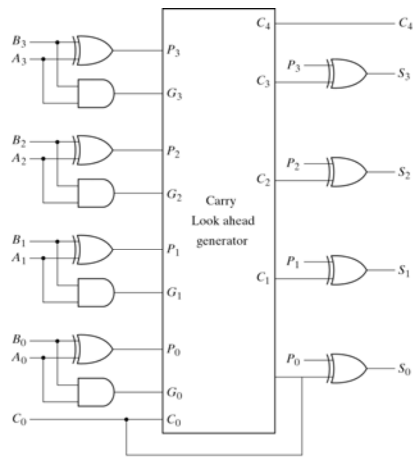
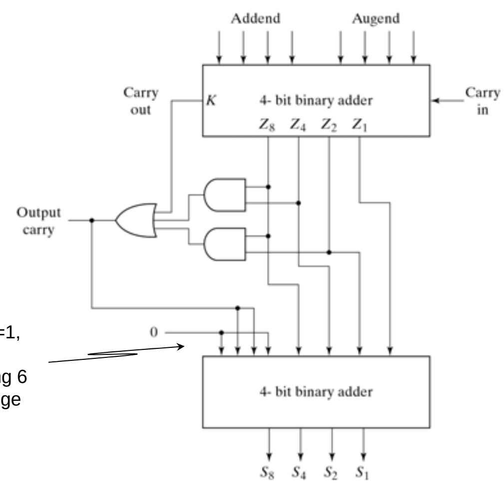


Fig. 4-12 4-Bit Adder with Carry Lookahead

BCD Adder



If the output carry =1, this = "0110", so it takes care of adding 6 to adjust out-of-range digits

Fig. 4-14 Block Diagram of a BCD Adder

Multiplying

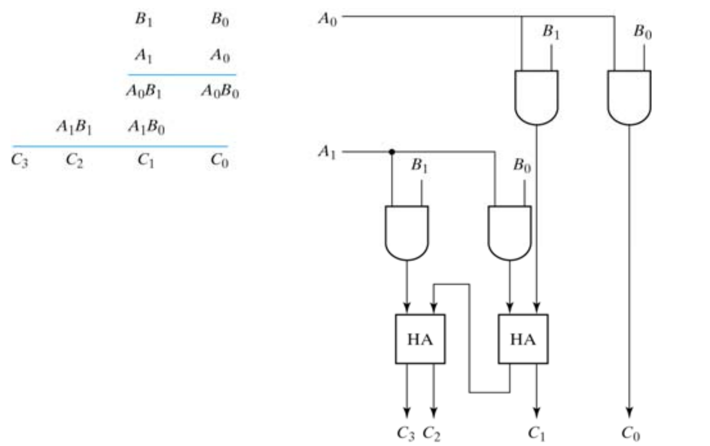


Fig. 4-15 2-Bit by 2-Bit Binary Multiplier

Larger multiplier using 4-bit adders as components

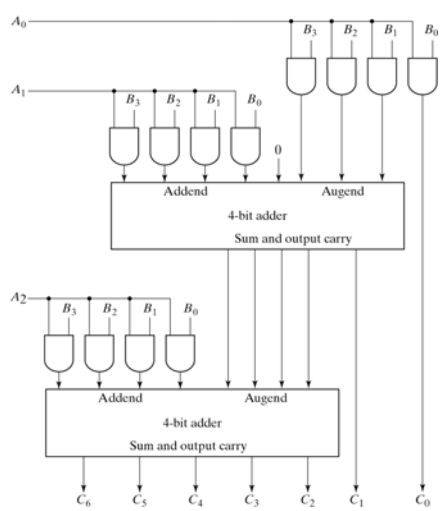


Fig. 4-16 4-Bit by 3-Bit Binary Multiplier

Magnitude Comparators

Function: For 2 unsigned n-bit inputs A and B, produce 1-bit outputs, “A=B”, “A<B”, and A>B”

Approach:

- Equal
  - If all bits are equal, the numbers are equal
- Greater than
  - A>B if:
    - A=1dddd… AND B=0dddd… OR
    - A=x1ddd… AND B=x0ddd… OR
    - A=xy1dd… AND B=xy0dd… OR
    - …

    That is,
    $A_nB_n' + (A_nB_n+A_n'B_n')A_{n-1}B_{n-1}' + \ldots$

A 4-bit Magnitude comparator

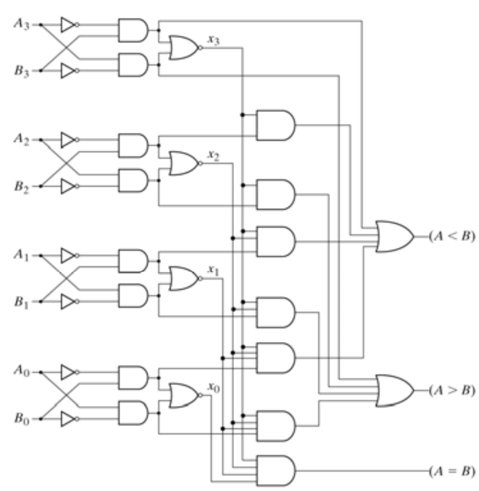


Fig. 4-17 4-Bit Magnitude Comparator

Decoders

An "n-to-2^n" decoder has n inputs and 2^n outputs. When the binary number "x" is input, output "x"=1; the rest of the outputs = 0.

x	y	z	D0	D1	D2	D3	D4	D5	D6	D7
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

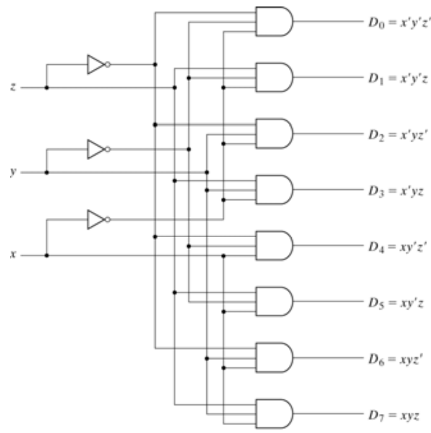
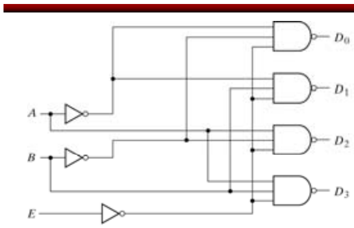


Fig. 4-18 3-to-8-Line Decoder

Combining Decoders



(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	0

(b) Truth table

Fig. 4-19 2-to-4-Line Decoder with Enable Input

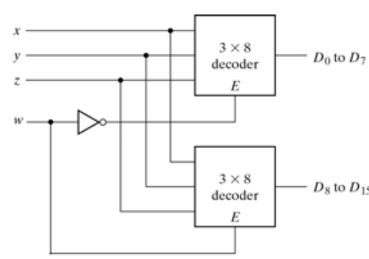


Fig. 4-20 4 × 16 Decoder Constructed with Two 3 × 8 Decoders

Implementing Boolean functions with decoders

If f(abcd…) = Σm(x,y,z,w…), we can implement f by simply ORing the decoder outputs x,y,z,w,…

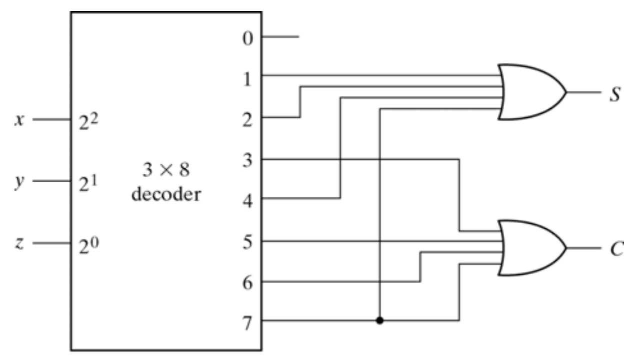


Fig. 4-21 Implementation of a Full Adder with a Decoder

Encoders

The inverse of a decoder. Set input line number “n” = 1, and the output is the binary-encoded value “n”.

D0	D1	D2	D3	D4	D5	D6	D7	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

x = D4 + D5 + D6 + D7

y = D2 + D3 + D6 + D7

z = D1 + D3 + D5 + D7

Priority Encoders

Handles the problem that arises when more than one input =1 by establishing an input priority.

D0	D1	D2	D3	x	y	VALID
0	0	0	0	d	d	0
1	0	0	0	0	0	1
d	1	0	0	0	1	1
d	d	1	0	1	0	1
d	d	d	1	1	1	1

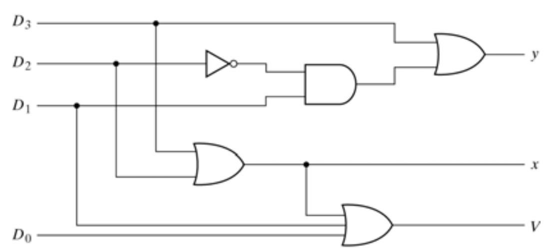
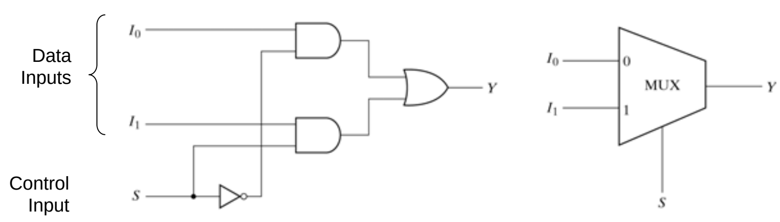


Fig. 4-23 4-Input Priority Encoder

Multiplexers

A multiplexer (or “Mux”) has multiple data inputs, a control input (which may consist of several signals) and one output. The value on the control inputs selects one data input, which is passed to the output.

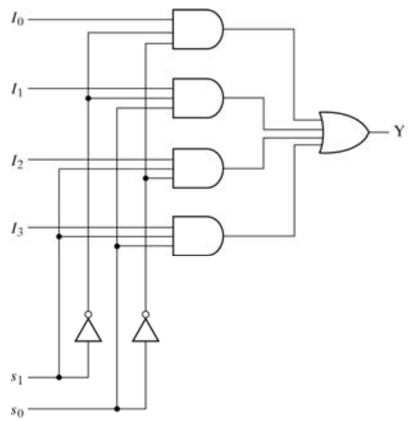


(a) Logic diagram

(b) Block diagram

Fig. 4-24 2-to-1-Line Multiplexer

Multiplexers



(a) Logic diagram

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

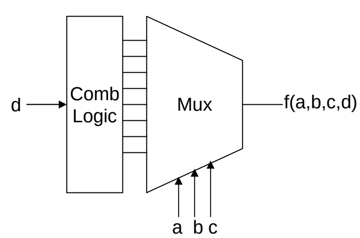
(b) Function table

Fig. 4-25 4-to-1-Line Multiplexer

Implementing Boolean functions with Muxes

A Mux with 2^n data inputs and n-1control lines can be used to implement a Boolean function of n variables.

The idea:

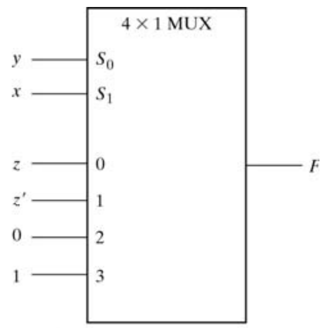


An example

F(xyz) = Σm(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

Fig. 4-27 Implementing a Boolean Function with a Multiplexer

The University Of Alabama in Huntsville Computer Science

Another example

cs309

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	

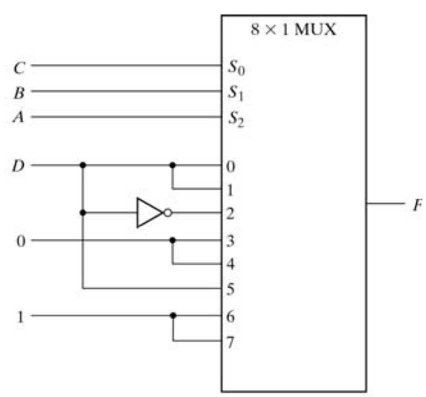
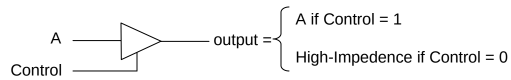
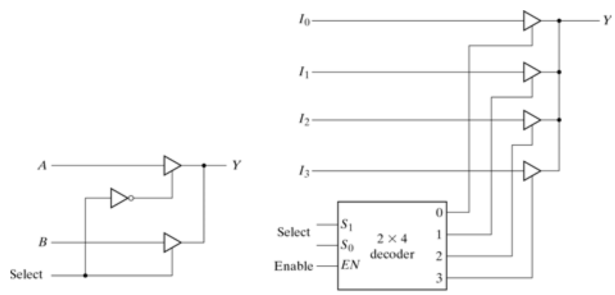


Fig. 4-28 Implementing a 4-Input Function with a Multiplexer

Tri-State logic



In the high-impedence state, the output is effectively disconnected from downstream elements



(a) 2-to-1- line mux

(b) 4 - to - 1 line mux

Fig. 4-30 Multiplexers with Three-State Gates