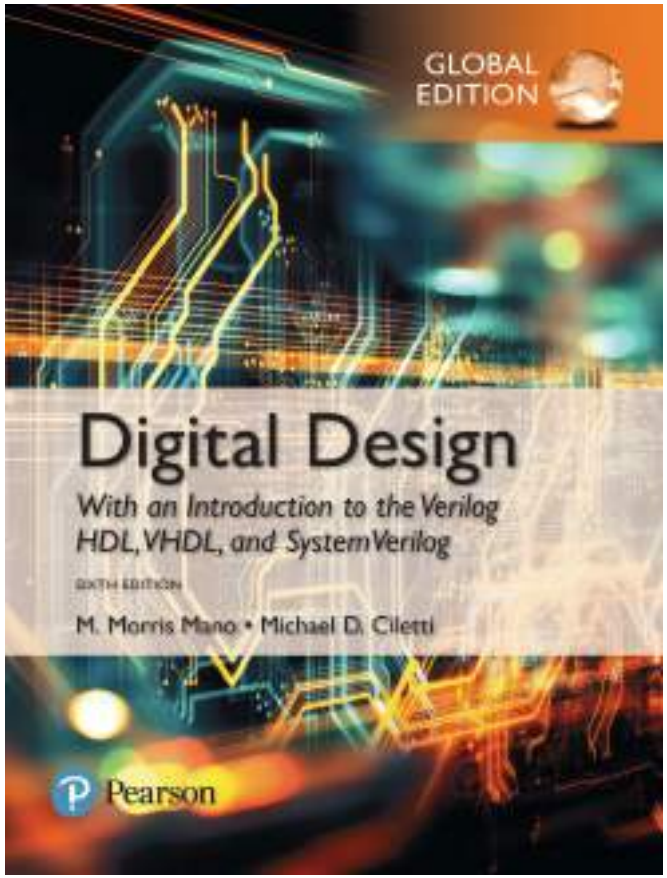
Digital Design

With an Introduction to the Verilog HDL, VHDL, and SystemVerilog

6th Edition, Global Edition



Chapter 04

Combinational Logic

조합 논리

전자공학과 김동한 교수

Pearson

4.1 개요 & 4.2 조합회로

- 논리회로
  - 조합형: 현재 입력값의 조합에 의해 출력 결정 (4장)
  - 순차형: 저장 가능한 요소를 추가. 저장된 값과 입력값에 따라 변함 따라서 시간 순서에 따른 입력값과 저장된 내부 상태값에 따라 출력이 달라짐 (5장)

- 조합회로: 논리게이트의 연결로 이루어짐. 조합 논리 게이트는 입력단의 신호에 의해 출력을 만듬. 만약 레지스터(저장 장치)가 조합 게이트와 같이 사용하면, 전체회로를 순차회로라 생각할 수 있음

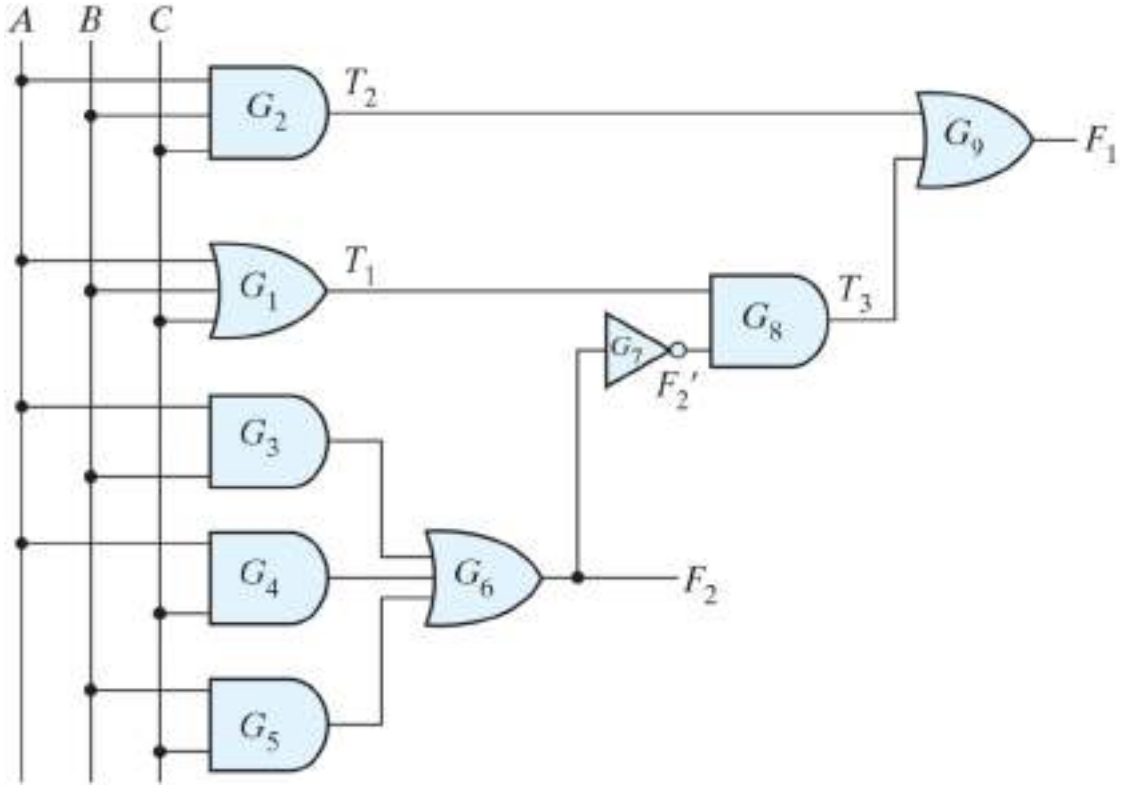
Figure 4.1
Block diagram of combinational circuit.

4.3 분석 과정

- 조합회로의 분석 과정
  - 논리도로 시작해서 부울함수, 진리표, 회로동작 설명 등을 통해 회로가 수행하는 기능을 알아내는 것. 첫단계는 조합회로인지 순차회로인지를 확인하는 것. 조합회로의 다이어그램은 피드백이나 저장 요소가 없는 논리 게이트들로 구성됨
  - 논리도가 조합회로인 것이 확인되면, 부울 함수의 출력값 또는 진리표를 얻을 수 있음. 방법은
    1. 입력 변수값에 따라 달라지는 게이트 출력에 임의의 기호를 붙임
    2. 입력 변수와 먼저 기호를 붙인 게이트의 함수로 되어 있는 이어지는 다음 게이트에 대해서도 다른 임의의 기호를 붙임
    3. 회로의 출력을 얻을 때까지 위의 과정 반복
    4. 입력 변수에 대한 부울 함수를 구할 때까지 반복

• 예)

Figure 4.2
Logic diagram for analysis example.



$F_2 = AB + AC + BC$

$T_1 = A + B + C$

$T_2 = ABC$

- $T_3 = F'_2 T_1$

  $F_1 = T_3 + T_2$

  $= F'_2 T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC$

  $= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC$

  $= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC$

  $= A'BC' + A'B'C + AB'C' + ABC$
- 위의 예처럼 부울 함수를 얻은 후에는 쉽게 진리표를 얻을 수 있음
- 만약 부울 함수를 유도하지 않고 논리도로부터 진리표를 직접 얻으려면,
  1. 입력 변수의 수를 결정함 (n개의 입력에서 2^n개의 입력조합가능)
  2. 선택된 게이트의 출력에 임의의 기호를 붙임
  3. 먼저 입력 변수 만의 함수인 게이트들의 출력에 대한 진리표를 구함
  4. 모든 게이트의 출력에 대한 진리표를 만듬

Table 4.1
Truth Table for the Logic Diagram of Fig. 4.2.

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 설계 과정

- 조합회로의 설계 과정
  - 설계 목적을 규정한 설계 사양(specification)에서부터 출발하여 논리도를 얻을 수 있는 논리회로도 또는 부울 함수의 집합을 구하는 것으로 마무리
    1. 필요한 입출력의 갯수를 구하고, 기호를 지정
    2. 진리표를 구함
    3. 입력 변수의 함수로서 각 출력에 대한 간략화된 부울 함수를 얻음
    4. 논리도를 그리고 확인함(시뮬레이션)
  - 실제 설계에 있어서는 게이트의 수, 게이트에 대한 입력의 수, 게이트를 통과하는 신호의 전파시간, 상호 연결 수, 각 게이트의 구동 능력의 한계(출력단에 연결할 수 있는 게이트의 수) 등의 설계 제약 조건을 고려해야 함

• 예) 코드변환(입력: BCD, 출력: 3초과 코드)

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

- 10진수를 나타내기 위해 4개의 입력 비트(ABCD)와 4개의 출력 비트(wxyz) 사용
- 2^4=16개 중 10개를 제외한 6개는 don't care 조합
- 다음 그림처럼 맵을 그리고 간략화된 부울 함수를 구함. 얻어진 표현식은 두 개 이상의 출력에 대해 공통 게이트를 사용할 수 있도록 조작가능

$$z = D'$$

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

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

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

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

- (C + D)가 세 개의 출력 각각에 사용됨. 원래는 AND 게이트 7개, OR 게이트 3개가 필요했는데, (C + D)를 공통으로 사용해서 AND 게이트 4개, OR 게이트 4개로 구현. 이러한 3-레벨 논리 회로는 더 적은 게이트를 필요로 하고, 모든 게이트는 2개 이하의 입력을 사용함

Figure 4.3
Maps for BCD-to-excess-3 code converter.

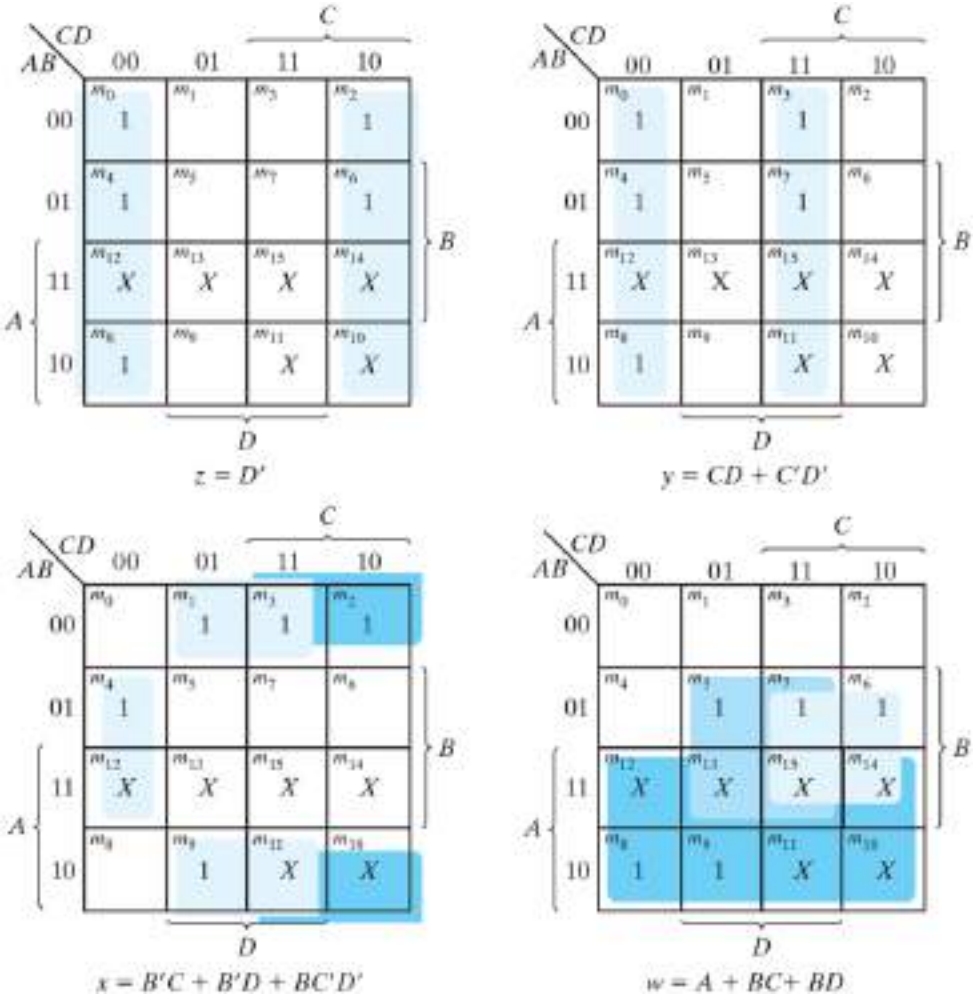
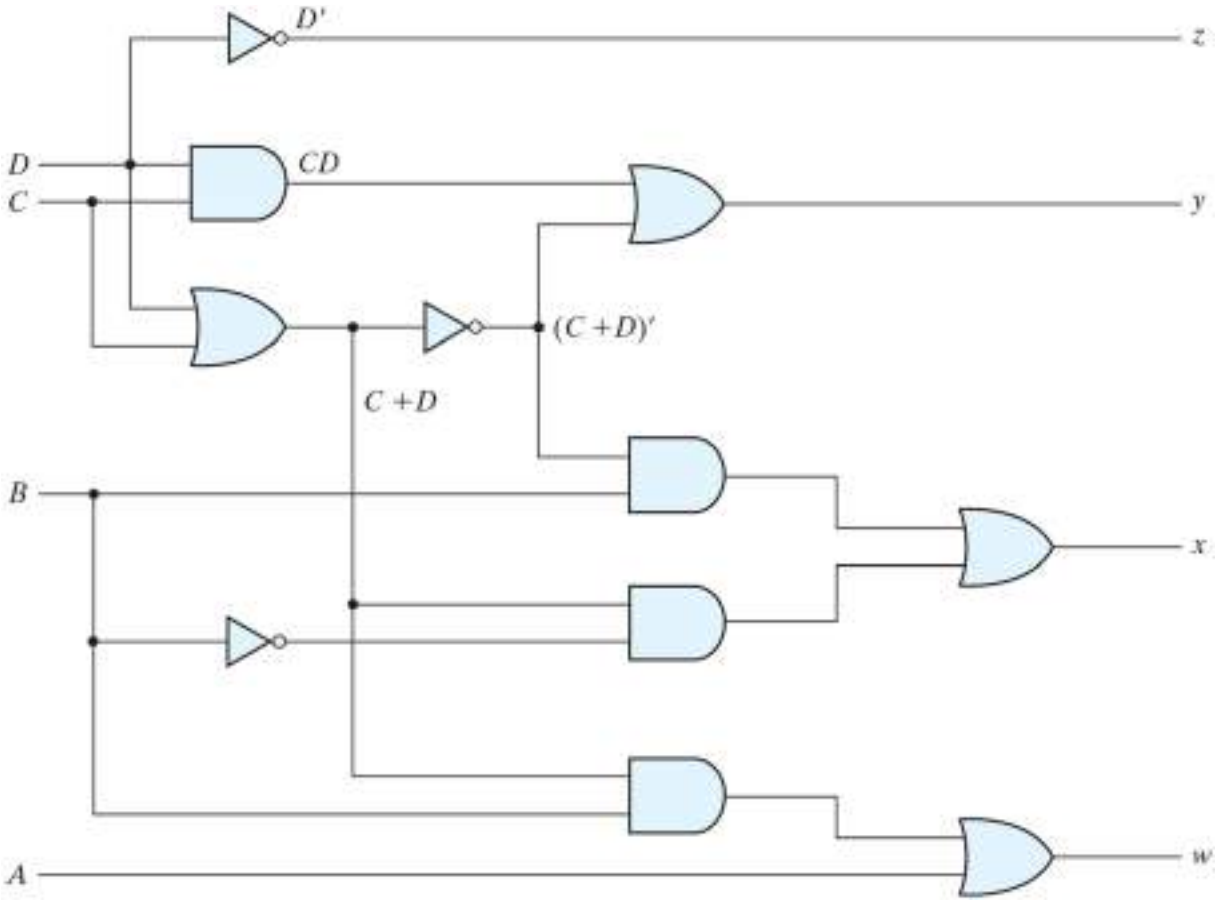


Figure 4.4
Logic diagram for BCD-to-excess-3 code converter.



4.5 2진 가산기-감산기

- 2진 가산기: 2개의 2진 숫자를 더함. 0 + 0 = 0, 0 + 1 = 1 + 0 = 1, 1 + 1 = 10(2자리 숫자가 되고 상위 비트를 캐리(carry)라고 함)
- 반가산기(half adder): 두 비트의 합산을 수행하는 조합회로
- 전가산기(full adder): 세 비트의 합산을 수행하는 조합회로. 2개의 반가산기로 구현가능
- 반가산기를 설계하고 나서 전가산기를 만듦. n개의 전가산기를 모아서 n비트의 두 숫자를 더하는 가산기를 만듦. 감산회로는 보수 회로를 이용해서 만듦

4.5 2진 가산기-감산기

- 반가산기: 입력 변수 x, y, 출력변수 S(합), C(캐리)

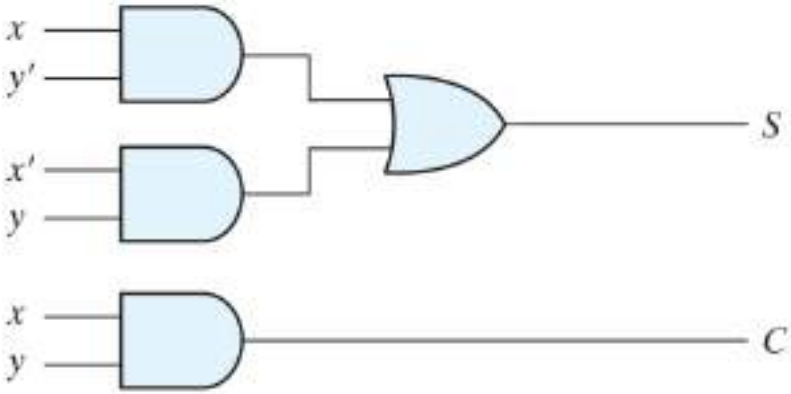
  $S = x'y + xy'$

  $C = xy$

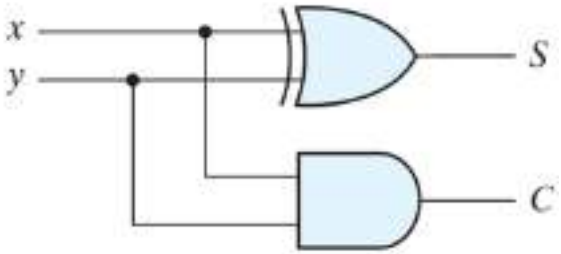
Table 4.3
Half Adder.

x	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Figure 4.5
Implementation of half adder.



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



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

4.5 2진 가산기-감산기

- 전가산기: n비트 2진수의 덧셈은 최하위 비트부터 순차적으로 전가산기를 이용함. 해당 자리 2비트와 이전 자리의 캐리를 고려함
- 입력 변수 x, y, z(이전 비트의 캐리), 출력변수 S(합), C(캐리)

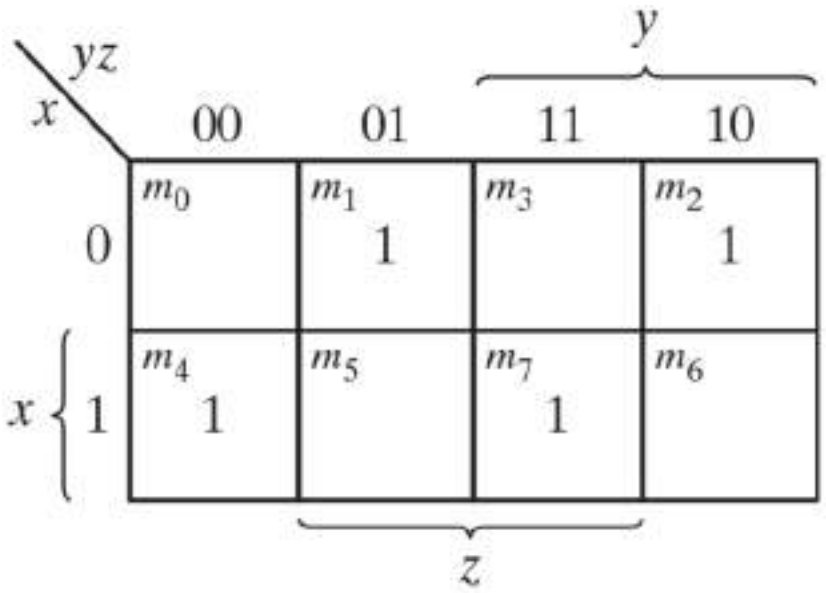
$$S = x'y'z + x'yz' + xy'z' + xyz$$

$$C = xy + xz + yz$$

Table 4.4
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

Figure 4.6
K-Maps for full adder.



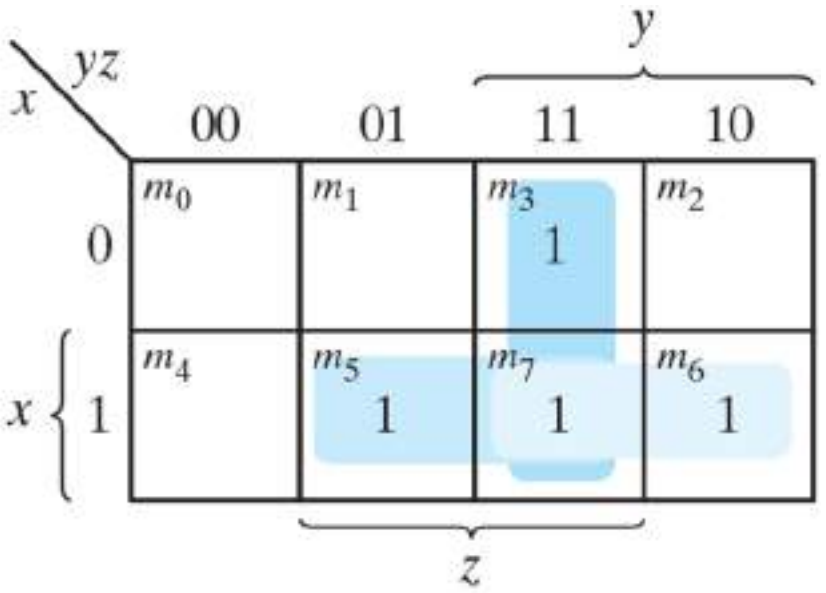
(a) $S = x'y'z + x'yz' + xy'z' + xyz$

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

$= z'(xy' + x'y) + z(xy + x'y')$

$= z'(xy' + x'y) + z(xy' + x'y)'$

$= z \oplus (x \oplus y)$



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

$C = xy'z + x'yz + xy$

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

Figure 4.7
Implementation of full adder in sum-of-products form.

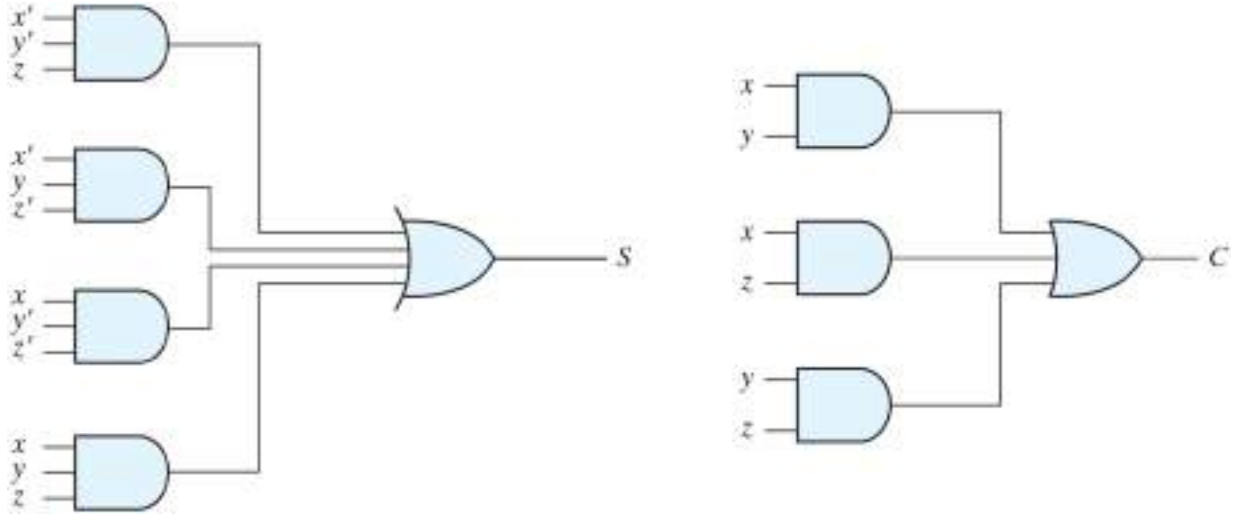
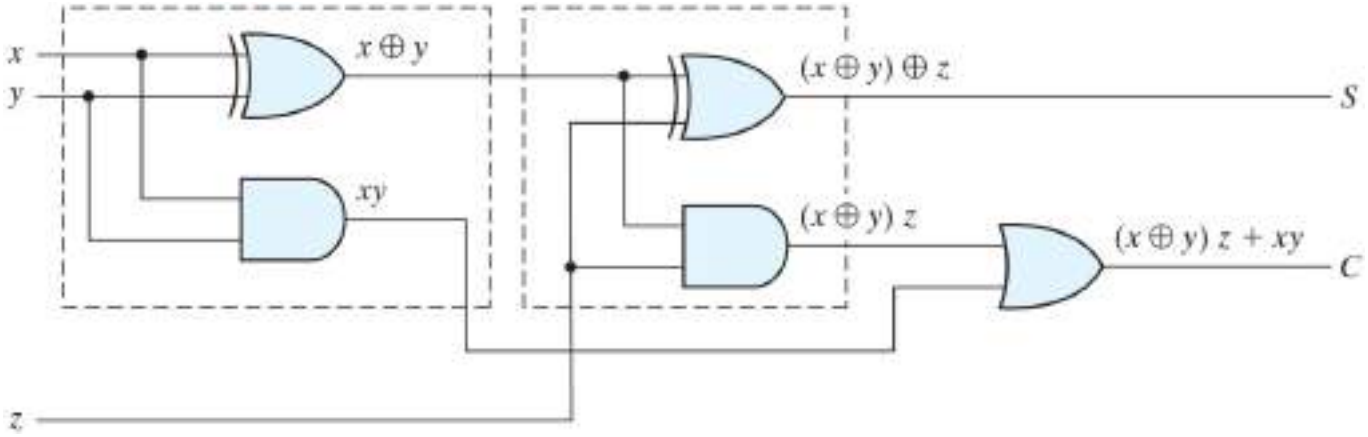
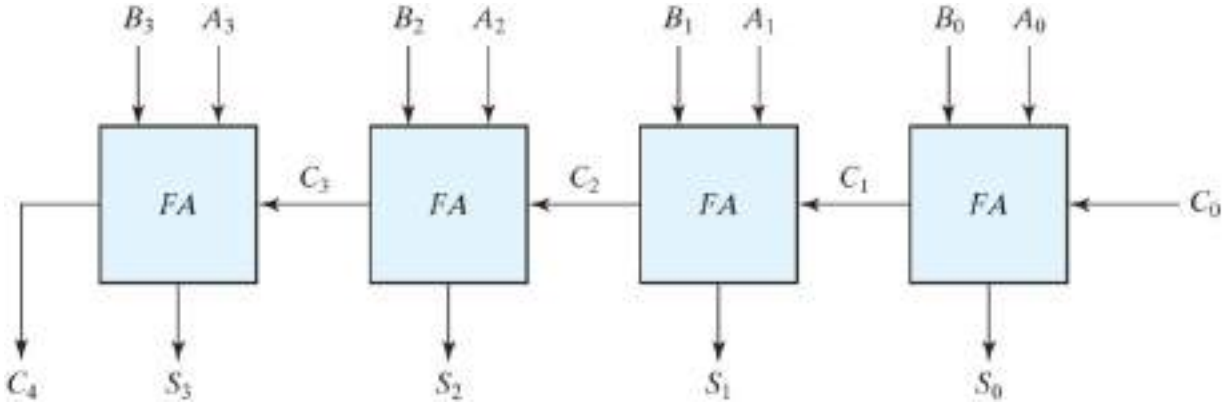


Figure 4.8
Implementation of full adder with two half adders and an OR gate.



- 2진 가산기: 첫번째 단의 전가산기의 캐리출력을 다음 단의 전가산기의 입력으로 연결시킨 것으로 전가산기의 종속연결로 구성. n비트 수의 덧셈은 n개의 전가산기가 필요하거나 (n-1)개의 전가산기와 하나의 반가산기가 필요함

Figure 4.9
Four-bit adder.



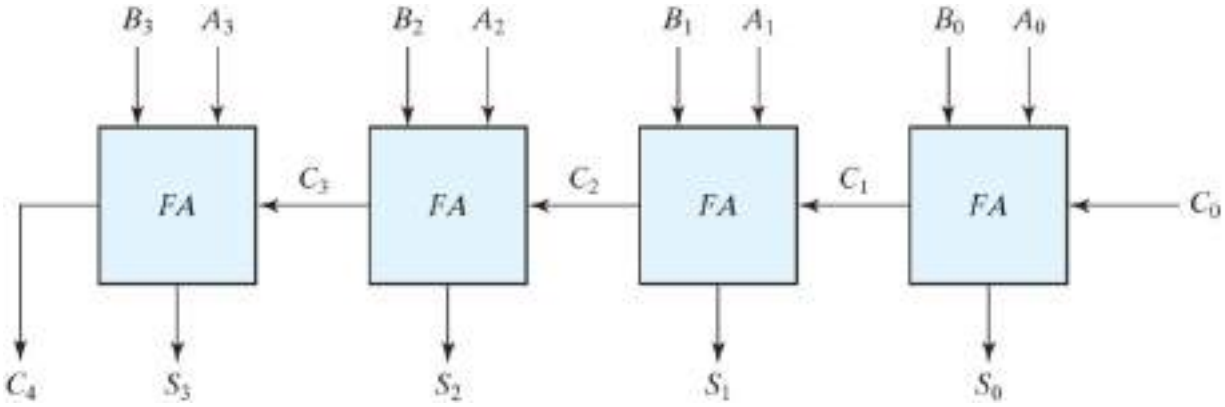
- 4비트 2진 리플 캐리 가산기를 구성하는 4개의 전가산기(FA)회로의 상호접속을 나타냄. 전가산기의 출력 캐리가 다음 상위단 전가산기의 입력 캐리에 연결됨

Binary Adder

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

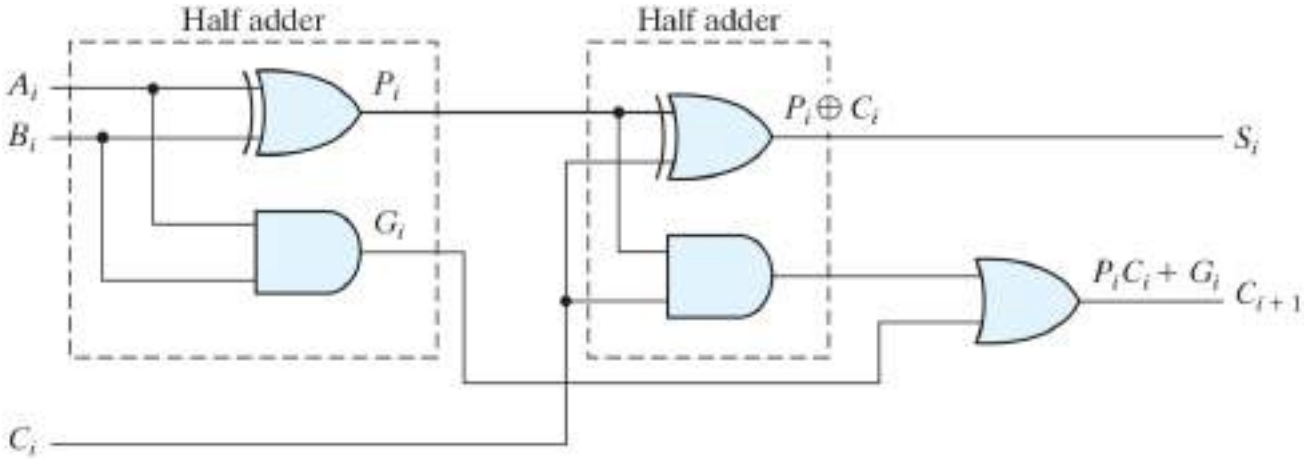
- 캐리의 전파: 아래처럼 병렬 2진수 덧셈은 모든 비트가 동시에 계산에 사용될 수 있음. 신호는 덧셈 출력을 만들기 위해서 여러 게이트들을 통과해야 함. 전체 신호전파시간은 게이트의 전파지연과 통과해야 하는 게이트 단계의 수를 곱한 것임. 따라서 가산기의 최장 전파 지연은 전가산기를 통해 캐리가 전파하는데 필요한 시간임. 임의의 가산기에서 주어진 단계의 S_i값은 그 단계로 입력 캐리가 전달된 후에나 정상적인 최종값에 도달함. (예: 아래 S_3)

Figure 4.9
Four-bit adder.



- 입력 캐리 C_i로부터 출력 캐리 C_{i+1}에 이르는 신호는 AND와 OR게이트를 통하여 전파하는데 2개의 게이트 레벨을 가짐. 결국 n비트의 가산에서는 캐리가 입력에서 출력까지 전파하기 위해서 2n개의 게이트 레벨을 가짐. 덧셈과정에서 소비되는 시간이 심각하므로, 캐리룩어헤드(carry lookahead logic)의 원리를 이용함

Figure 4.10
Full adder with *P* and *G* shown.



$P_i = A_i \oplus B_i$ → 캐리 전파(carry propagation)

$G_i = A_i B_i$ → 캐리 생성(carry generation)

$S_i = P_i \oplus C_i$

$C_{i+1} = G_i + P_i C_i$

- 각 단계의 캐리 출력에 대한 부울 함수를 쓰면,

C_0 = 입력 캐리

$C_1 = G_0 + P_0 C_0$

$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0)$

$= G_1 + P_1 G_0 + P_1 P_0 C_0$

$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

- 각 캐리에 대한 부울 함수는 OR게이트와 AND게이트로 구현 가능. 그림 4.11처럼 C_3는 C_2와 C_1이 전파하는 것을 기다릴 필요가 없음. 동작 속도는 빨라지지만 추가적인 회로의 복잡도 증가함.
- 그림 4.12는 캐리 룩어헤드 구조를 갖는 4비트 가산기임

Figure 4.11
Logic diagram of carry lookahead generator.

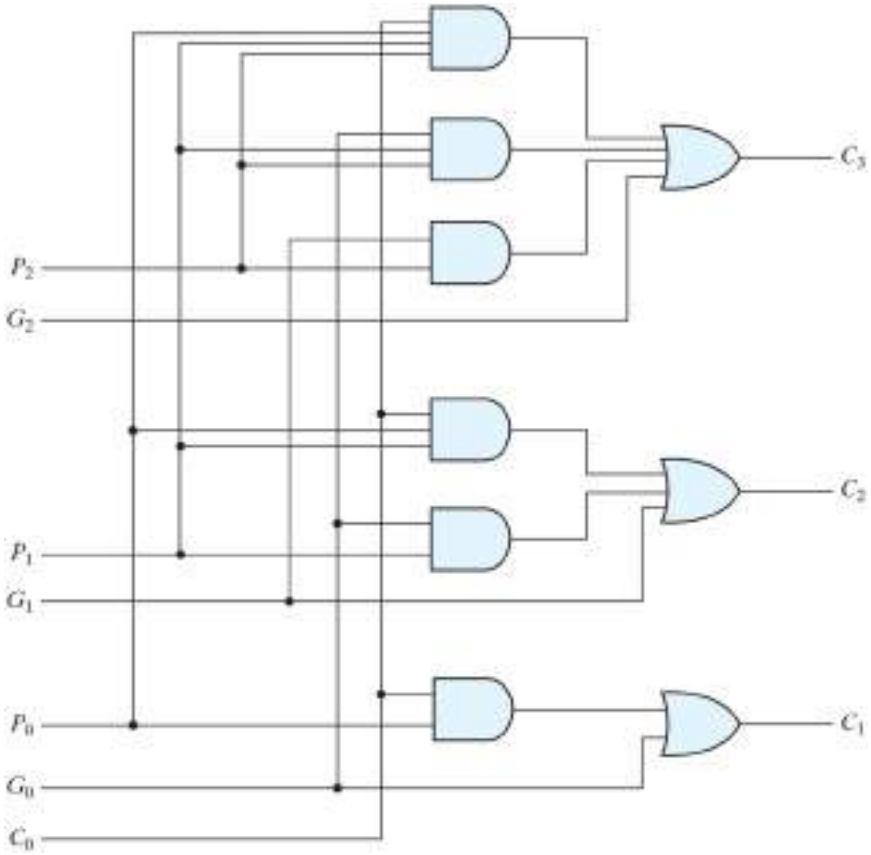
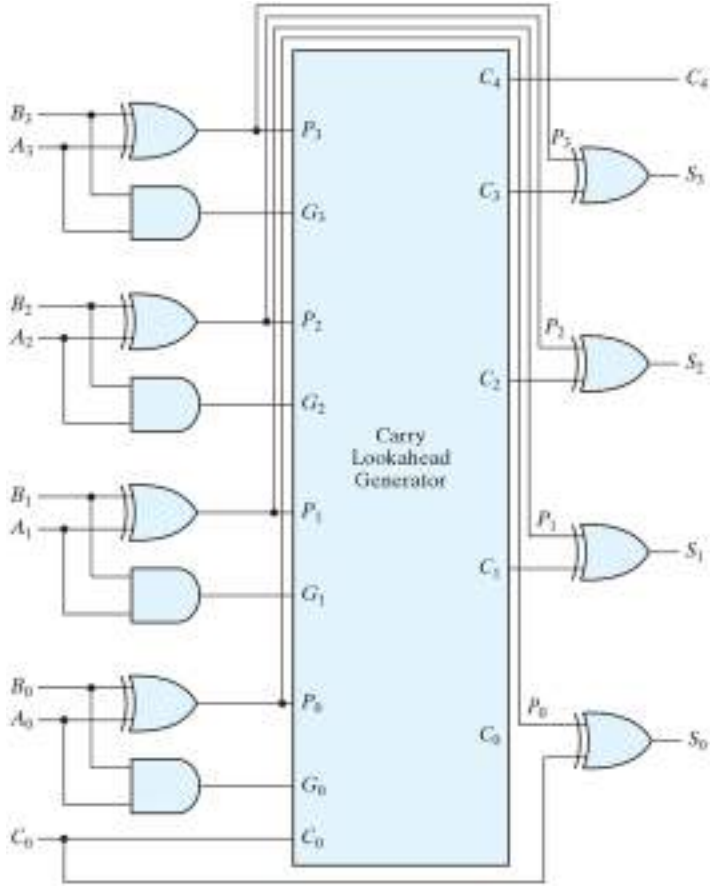
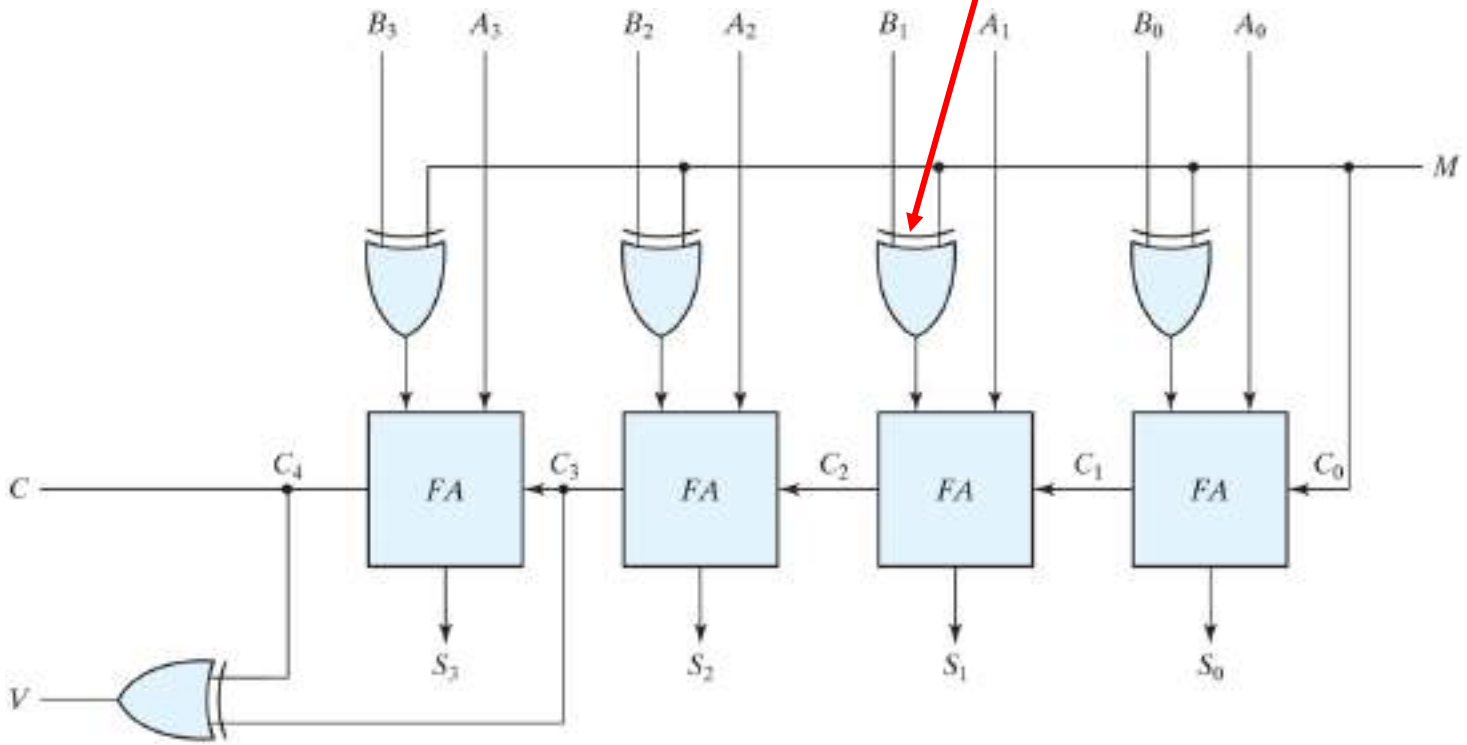


Figure 4.12
Four-bit adder with carry lookahead.



- 2진 감산기: 뺄셈 A – B는 B에 대한 2의 보수를 A에 더함. 2의 보수는 1의 보수에 1을 더함. 1의 보수는 인버터로 구현하고 입력 캐리를 통해 합에 1을 더함.
- 입력 B와 전가산기(FA)의 입력 사이에 인버터(여기선 XOR)가 있는 가산기로 구성. 입력 캐리 C_0는 감산할 때 1이 되어야 함

Figure 4.13
Four-bit adder–subtractor (with overflow detection).



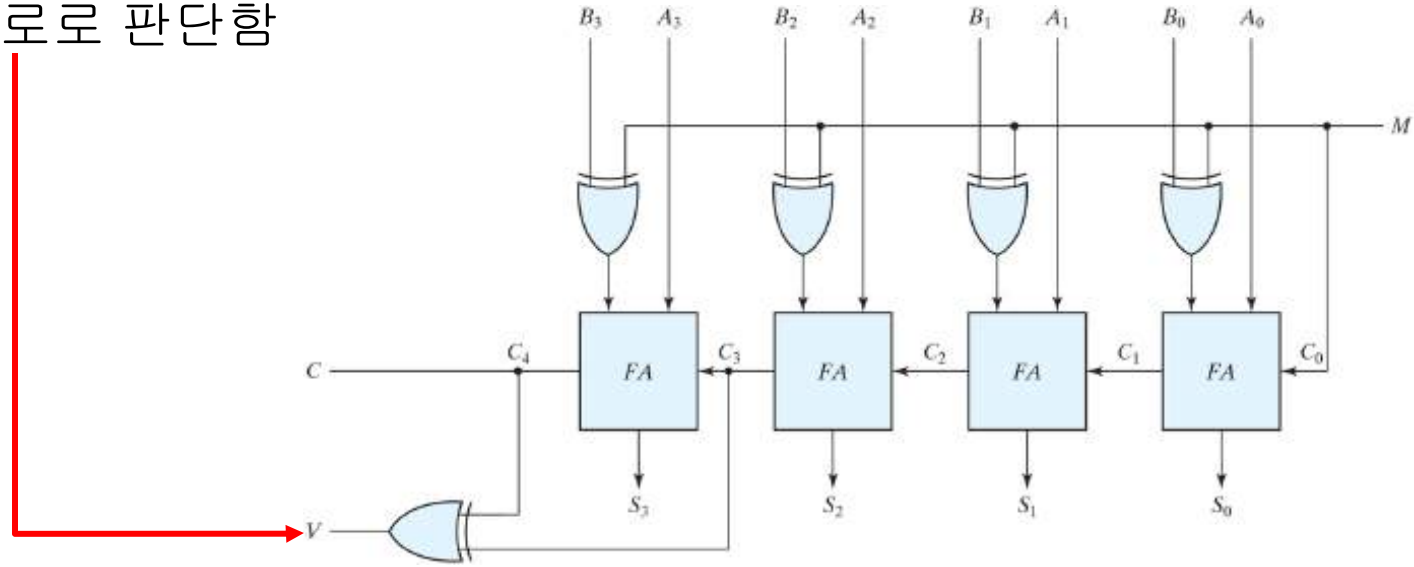
- 부호가 없는 수(unsigned number)의 경우, A >= B일 때 A – B이고, A < B일 때 B에 대한 2의 보수에서 A를 뺌. 그러나 부호가 있는 수(signed number)인 경우에는 오버플로(overflow)가 없다고 가정할 때 A – B임
- 모드 입력 M = 0이면 가산기((B⊕ 0 = B, 입력 캐리 = 0)로, M = 1이면 감산기(B⊕ 1 = B', C_0 = 1)로 동작
- 사용자는 부호가 있는 수의 연산인지 부호가 없는 수의 연산인지에 따라 덧셈이나 뺄셈의 결과를 다르게 해석해야 함

- 오버플로(overflow): n 비트의 두 수를 더해서 n+1의 자리를 차지할 때
  - 부호가 없는 수가 더해질 때, 오버플로는 최상위 비트의 캐리 출력으로부터 감지됨.
  - 부호가 있는 수가 더해질 때,
    1. 최상위 비트는 부호임. 음수는 2의 보수형태임
    2. 부호 비트는 수의 일부분으로 다뤄지고, 최종캐리는 오버플로 아님.

- 오버플로는 두수 모두가 양수이거나 음수일 때 발생함
- 캐리: 0 1 1 0

+70	0 1000110	-70	1 0111010
+80	0 1010000	-80	1 0110000
+150	1 0010110	-150	0 1101010
	음의 결과(X)		양의 결과(X)

- 오버플로 상태는 부호 비트 자리에 들어가는 캐리와 부호 비트 자리에서 발생하는 캐리 출력을 보면 알 수 있음. 이 두개의 캐리가 다르면 오버플로임. 따라서 이 두개의 캐리를 XOR 게이트에 통과시켜 1이면 오버플로로 판단함



- 그림 4.14에서 두 수가 부호가 없는 경우, C비트는 가산 후의 캐리 혹은 감산 후의 빌림수(borrow)임. 부호가 있는 경우, V비트는 오버플로를 감지함. 가산 혹은 감산 후에 V = 0이면, n비트의 결과값은 정확한 것임. 만약 V = 1이며, 결과값이 n+1비트이므로 오버플로임. n+1비트는 부호를 나타내지만 시프트되어 버리는 값임

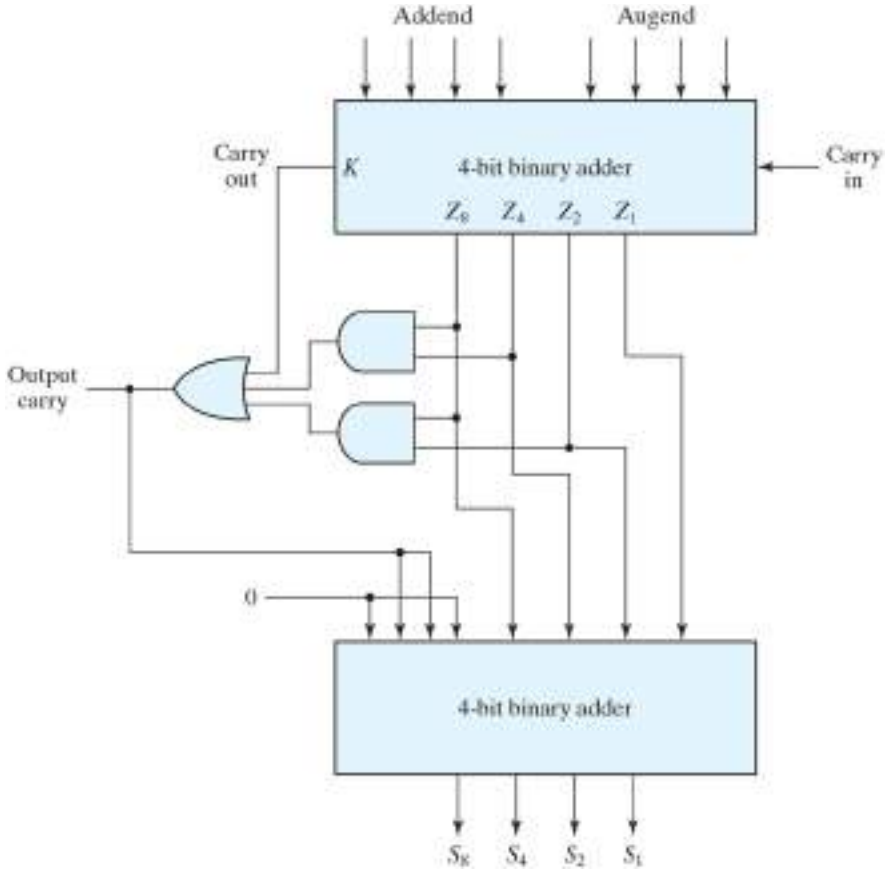
4.6 10진 가산기

- BCD로 된 2개의 10진수 숫자를 산술적으로 더하면, 9 + 9 + 1(캐리) = 19보다 클 수는 없음.
- 2개의 BCD 숫자를 4비트 2진 가산기에 넣으면 표 4.5의 왼쪽 컬럼과 같음. 하지만 결과값을 다시 BCD로 표현하려면 두번째 컬럼과 같아야 함. 이때의 규칙은
  1. 0~9까지는 동일 (1001)
  2. C = K + Z_8Z_4 + Z_8Z_2 (K는 캐리)
  3. C = 1일 때, 2진수 결과값에 0110을 더함

Table 4.5
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

Figure 4.14
Block diagram of a BCD adder.



Pearson

4.7 2진 곱셈기

- 피승수는 최하위 비트로부터 시작해서 승수의 각 비트를 곱함. 각각의 비트에서 계산된 곱셈은 부분곱의 형태로 생성됨. 비트별로 얻어지는 부분곱들은 왼쪽으로 한 자리씩 이동되며, 최종 곱의 결과는 부분곱의 합으로 이루어짐

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
P_3 & P_2 & P_1 & P_0
\end{array}
$$

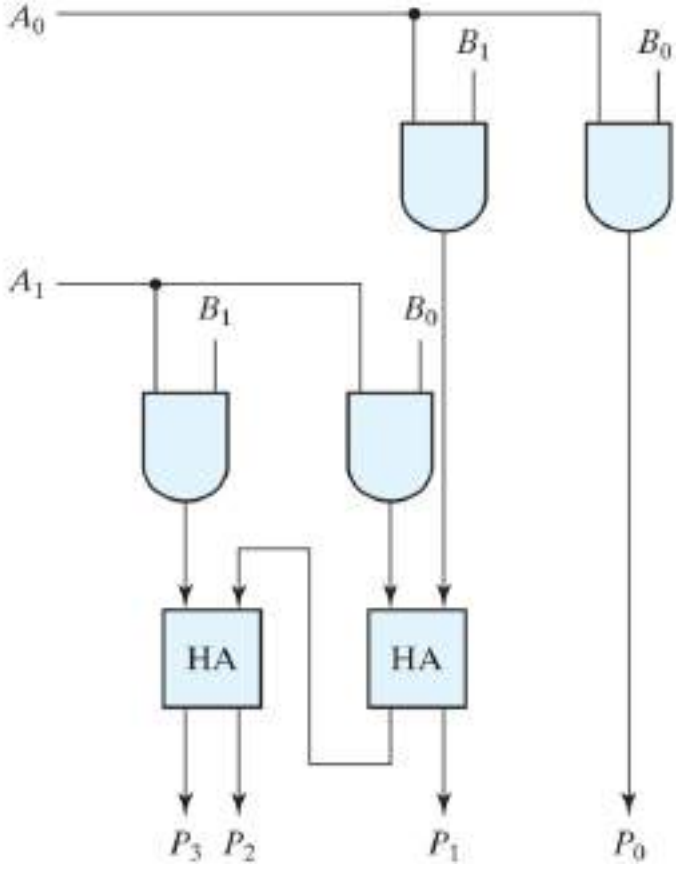
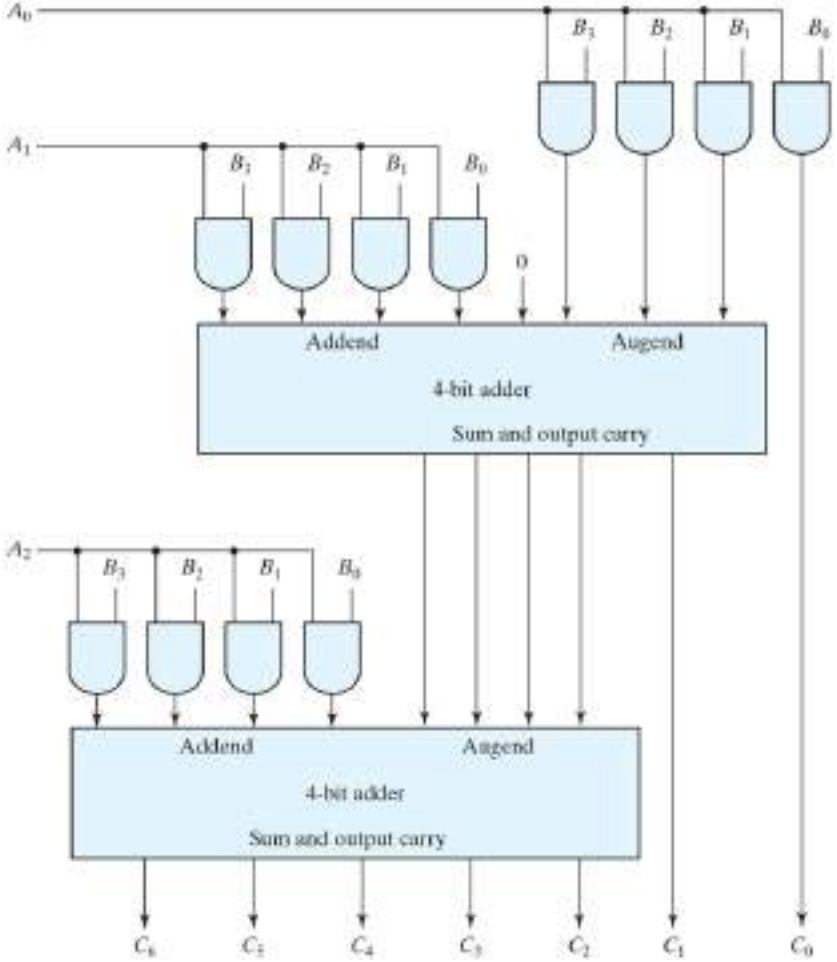


Figure 4.15
Two-bit by two-bit binary multiplier.

$B_0B_1 \times A_1A_0 = C_3C_2C_1C_0$

- A_0와 B_0B_1곱은 AND, A_1과 B_0B_1 곱은 AND로 자리를 맞춰서 Half adder로 구성

Figure 4.16
Four-bit by three-bit binary multiplier.



- 그림 4.16처럼 4비트 수(K=4)와 3비트 수(J=3)의 곱셈기의 구성요소는 (J X K) AND게이트 (4X3=12)와 (J-1)K-bit 가산기 (2개의 4비트 가산기) 필요

4.8 크기 비교기

- 3개의 출력값: A > B, A = B, A < B
- 2개의 n비트 수를 비교하기 위해서는 2^{2n}개의 항목을 갖는 진리표를 만들어야 하지만, 규칙성을 찾으면 진리표 없이 쉽게 가능
- 여기서는 4비트 비교기 설계를 예로 설명함 ($A = A_3A_2A_1A_0$, $B = B_3B_2B_1B_0$)

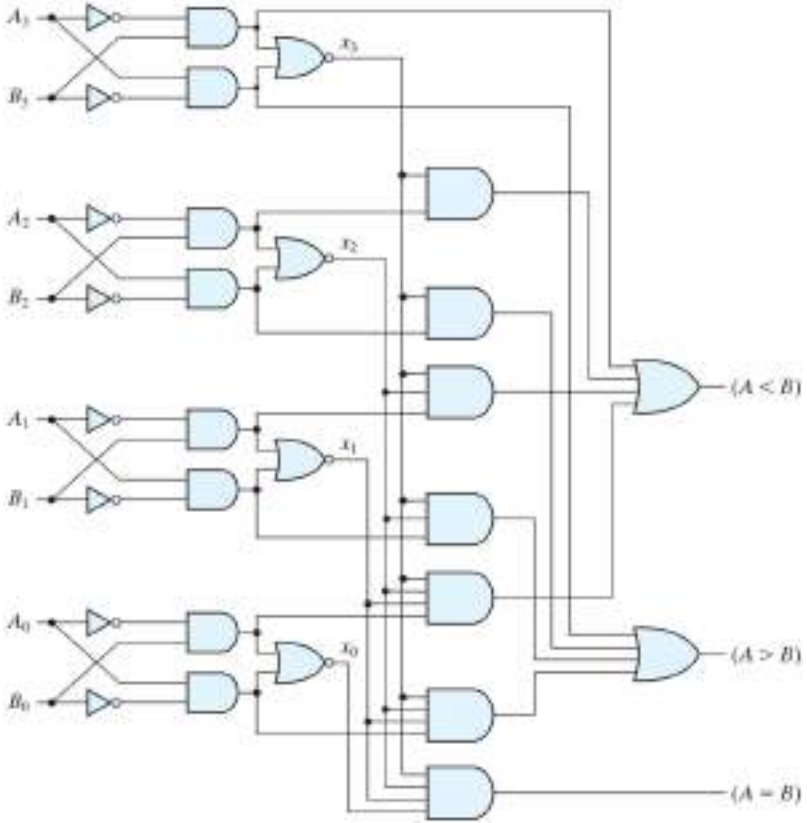


Figure 4.17
Four-bit magnitude comparator.

Pearson

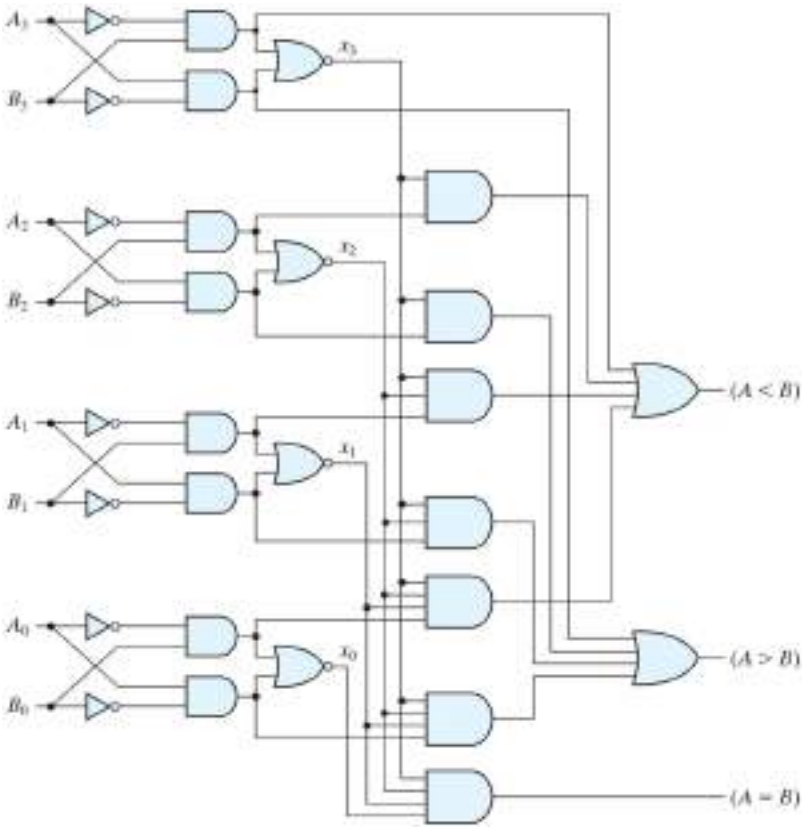
1. A = B일 때, A3 = B3, A2 = B2, A1 = B1, A0 = B0

   → $x_i = A_iB_i + A_i'B_i'$, $(A=B) = x_3x_2x_1x_0$

2. A > B or A < B일 때, 큰자리수 비트부터 검사해서 같지 않은 숫자의 쌍에 도달할 때 A가 1, B가 0이면 A > B, A가 0, B가 1이면 A < B 됨

   → $(A>B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$

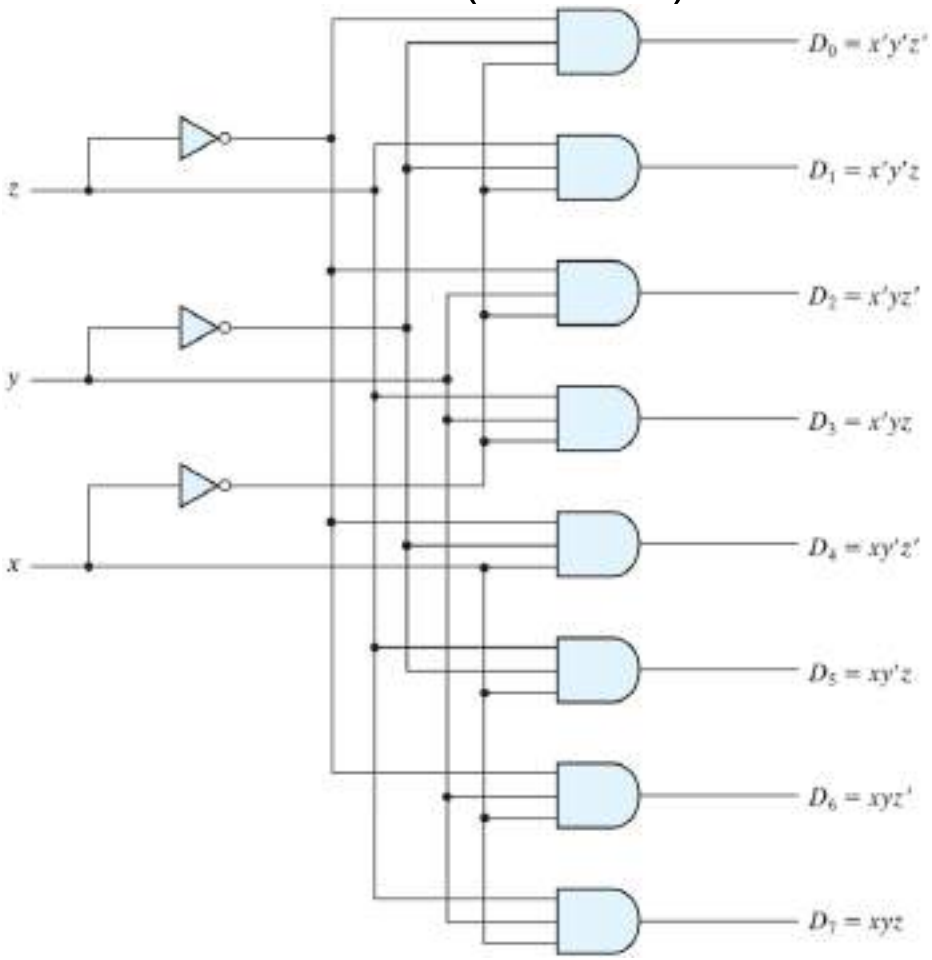
   $(A<B) = A_3'B_3 + x_3A_2'B_2' + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$



4.9 디코더

- 2진 정보를 담은 n개의 입력을 최대. 2^n개의 출력으로 변환함. n-to-m 라인 디코더는 n개의 입력 변수로부터 2^n개의 최소항(minterm)을 만듬

Figure 4.18
Three-to-eight-line decoder.



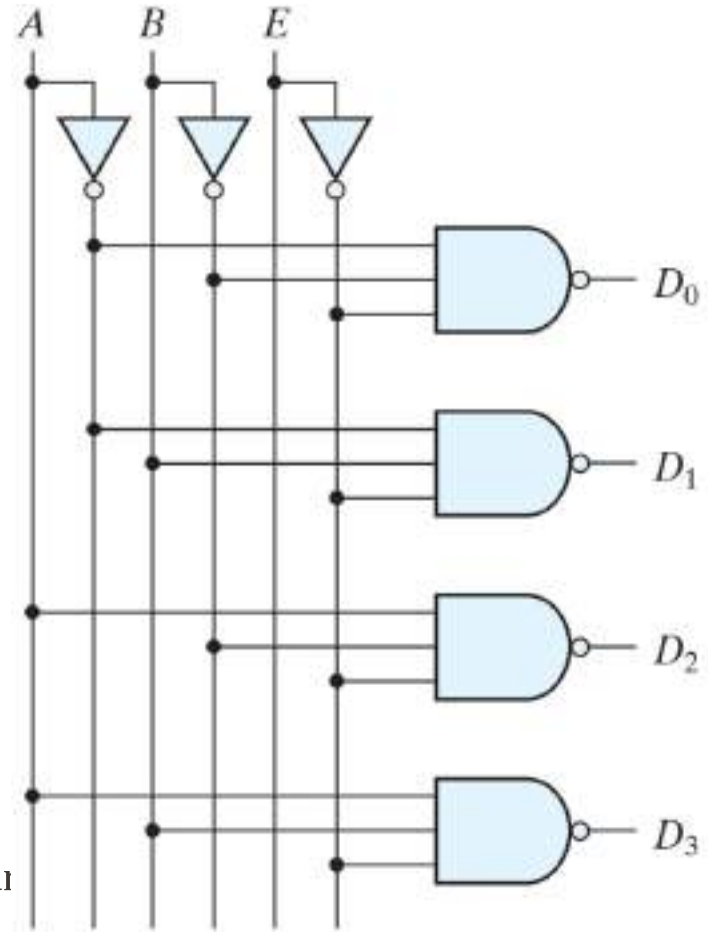
- 옆 그림은 3비트 2진수를 1개의 8진수로 디코딩함 (표 4.6)
- AND게이트 대신 NAND 게이트 사용 가능하고, enable 입력을 추가할 수 있음 (그림 4.19)

Table 4.6
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

- AND게이트 대신 NAND 게이트 사용 가능하고, enable 입력을 추가할 수 있음 (그림 4.19)
- 디멀티플렉서(demultiplexer): 단일 입력으로 정보를 받아서 2^n개의 출력 중에 하나를 선택하여 내보내는 회로. (E가 데이터입력)

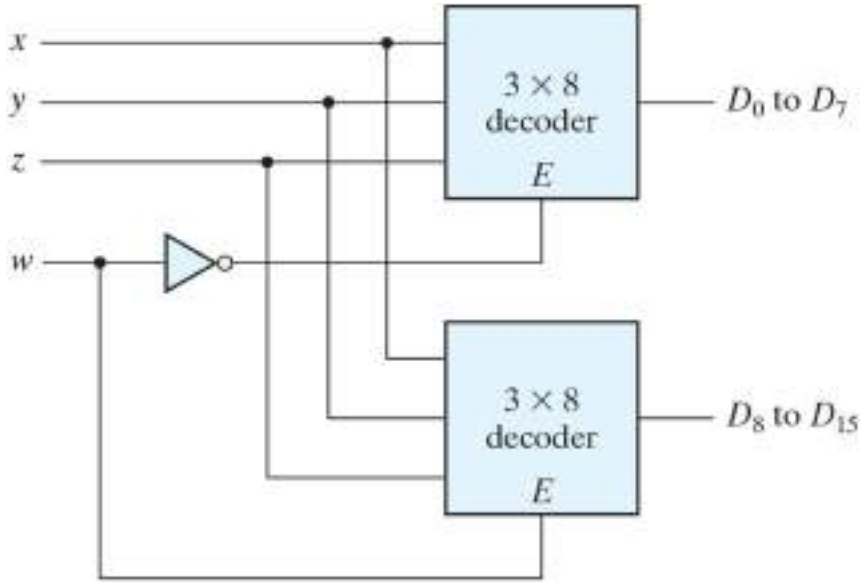
Figure 4.19
Two-to-four-line decoder with enable input.



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

- 인에이블 입력을 갖는 디코더는 더 큰 디코더 회로로 서로 연결하여 구성 가능

Figure 4.20
4 x 16 decoder constructed with two 3 x 8 decoders.

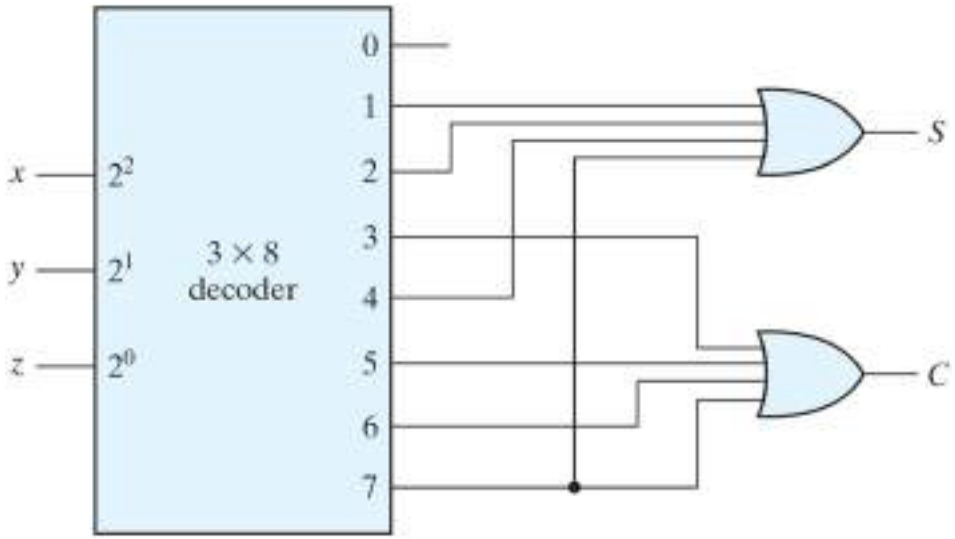


- w = 0일 때, 위쪽 인코더는 인에이블 상태이며 다른 쪽은 동작하지 않은 상태임. 아래쪽 디코더 출력은 전부 0이며, 위쪽 8개 출력은 0000에서 0111까지의 최소항을 만듦. 반대로 w = 1일 때, 아래쪽 디코더 출력은 1000에서 1111에 이르는 최소항을 만듦

- 조합 논리의 구현
  - 디코더는 n개의 입력 변수에 대한 2^n개의 최소항을 제공하고 임의의 부울 함수는 최소항의 합으로 표시할 수 있기 때문에 디코더의 외부에 OR게이트를 사용하여 구현가능
  - n개의 입력과 m개의 출력을 갖는 임의의 조합회로를 하나의 n-to-2^n 라인 디코더와 m개의 OR 게이트로 구현가능
  - 예를 들어, 전가산기의 진리표로 부터 합과 캐리에 대한 최소항들의 합의 함수를 얻을 수 있음

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

Figure 4.21
Implementation of a full adder with a decoder.



4.10 인코더

• 디코더 기능과 반대되는 디지털 회로. 2^n의 입력과 n개의 출력을 가짐

Table 4.7
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

$z = D_1 + D_3 + D_5 + D_7$

$y = D_2 + D_3 + D_6 + D_7$

$x = D_4 + D_5 + D_6 + D_7$ → 3개의 OR 게이트로 구현 가능

- 하지만 두 입력이 동시에 1일 때는 출력은 정의되지 않은 조합을 만들어 냄. 또한 전부 0인 출력이 D_0가 1일 때 발생함. 따라서 동시에 2개 이상의 입력이 1일 때 가장 높은 우선순위를 가진 입력을 취함. 아래 표에서 V는 모든 입력이 0이면 유효한 입력은 없다는 뜻으로 0이 됨. X로 인해 16개의 최소항 진리표 대신 간단한 표 4.8이 됨. D_3가 우선순위가 제일 높음

$$x = D_2 + D_3$$

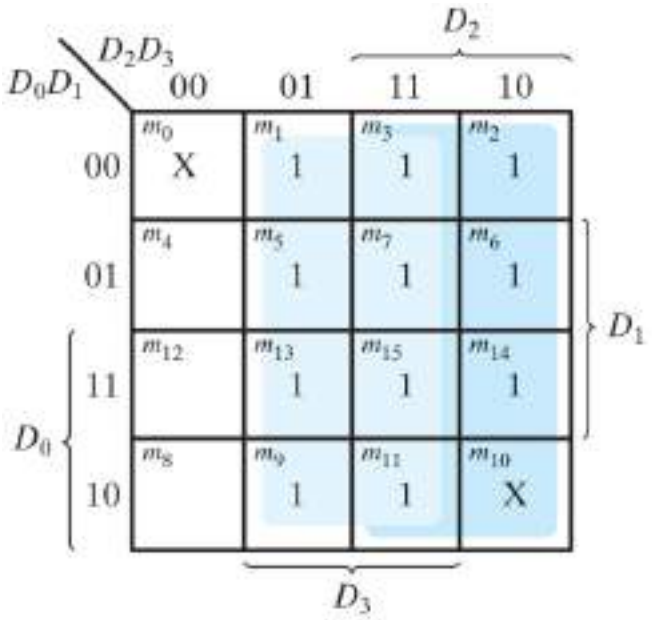
$$y = D_3 + D_1D_2'$$

$$V = D_0 + D_1 + D_2 + D_3$$

Table 4.8
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

Figure 4.22
Maps for a priority encoder.



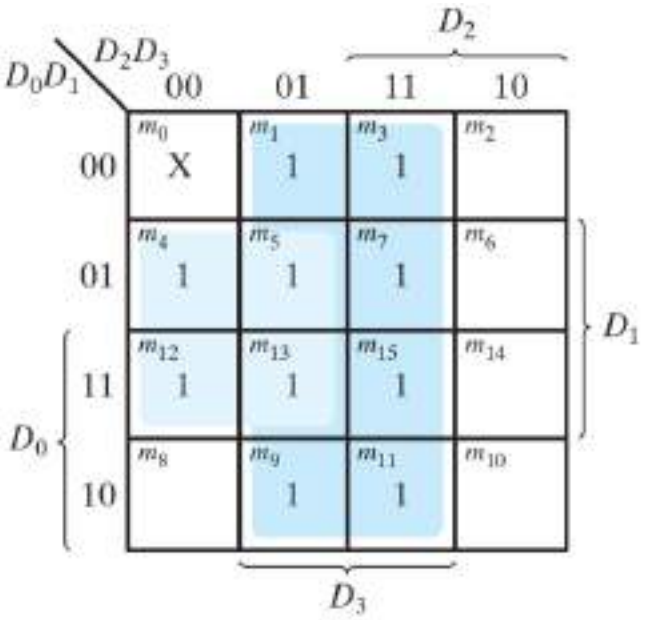
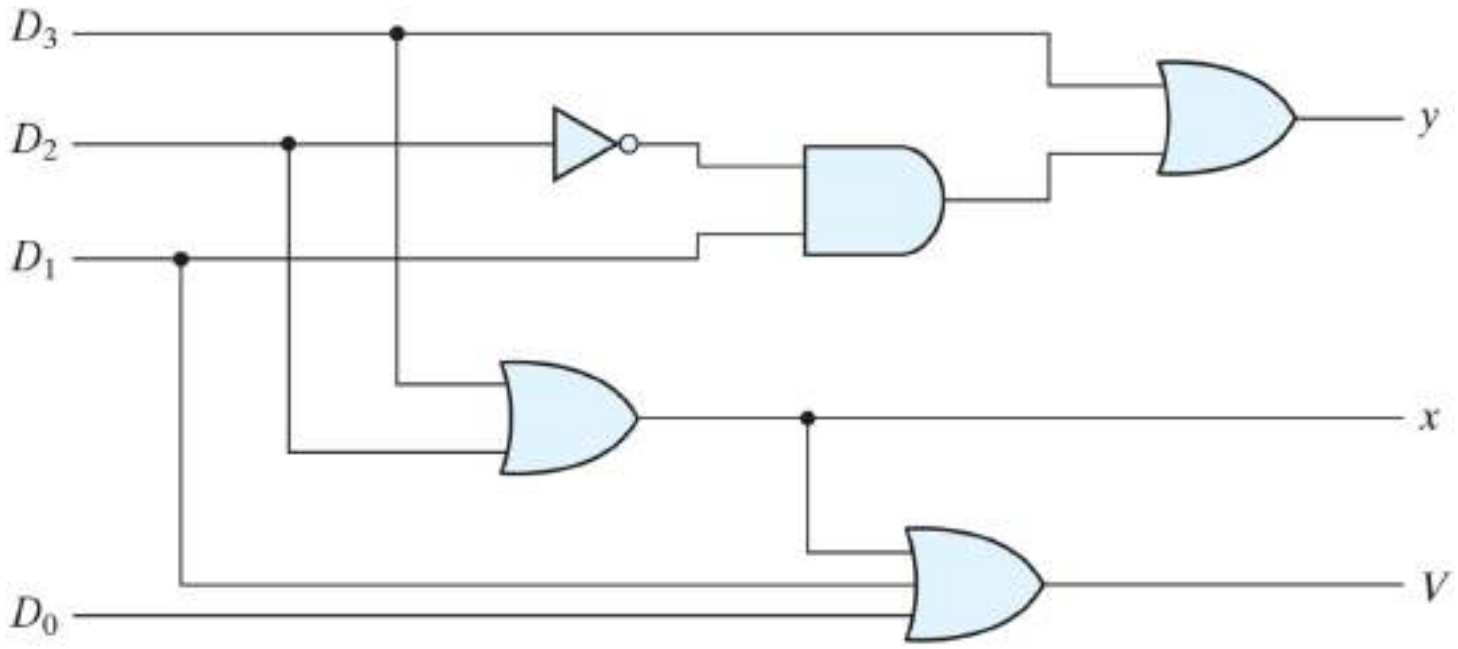


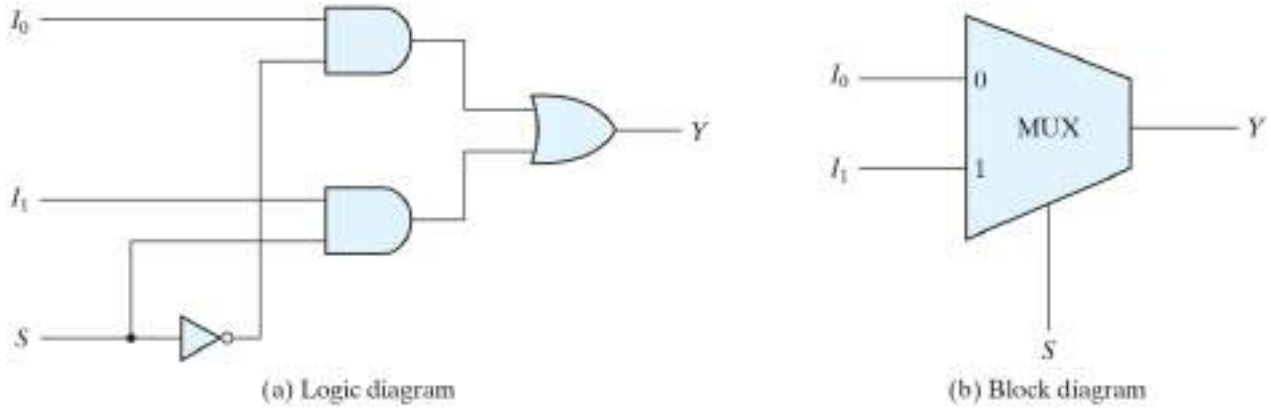
Figure 4.23
Four-input priority encoder.



4.11 멀티플렉서(Multiplexer, MUX)

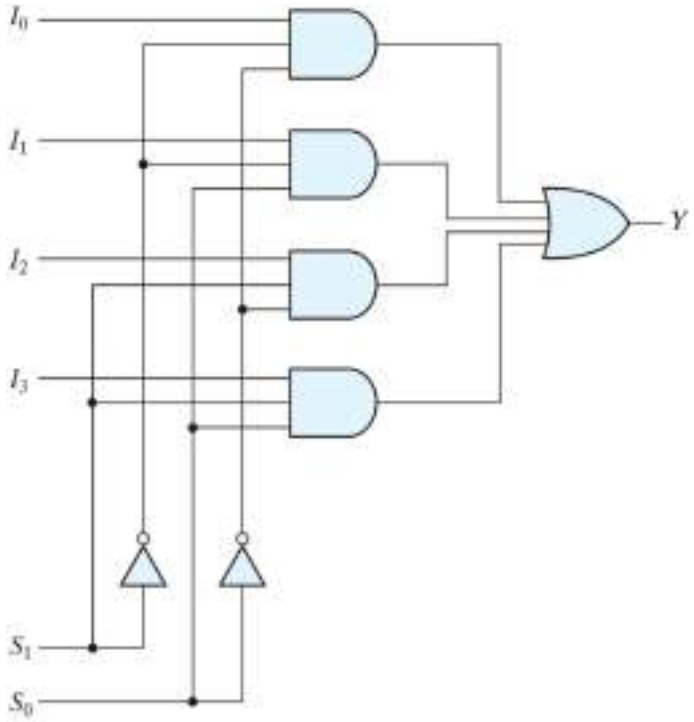
- 많은 입력 중에서 하나의 2진 정보를 선택하여 출력으로 연결. 2^n개의 입력과 n개의 선택제어 신호
  - 2-to-1 라인 멀티플렉서

Figure 4.24
Two-to-one-line multiplexer.



– 4-to-1 라인 멀티플레서는 I_0에서 I_3까지의 입력, S_1과 S_0는 선택, AND 게이트의 출력은 OR 게이트에 입력

Figure 4.25
Four-to-one-line multiplexer.



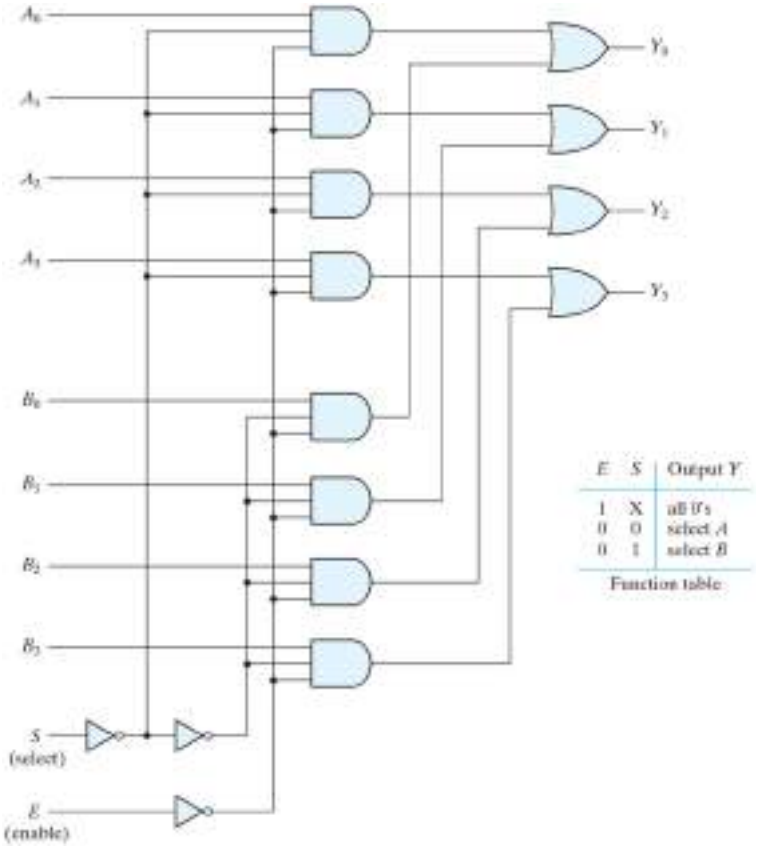
(a) Logic diagram

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

(b) Function table

  - 4-to-1 라인 멀티플레서는 I_0에서 I_3까지의 입력, S_1과 S_0는 선택, AND 게이트의 출력은 OR 게이트에 입력
- 데이터 선택기(data selector)라고도 함
- 멀티플렉서의 AND 게이트와 인버터는 디코더 회로와 유사함. 선택 입력을 디코딩 함
- 인에이블(enable) 입력으로 동작 제어 가능
- 다중 비트 선택 기능과 enable 입력의 예는 다음과 같음

Figure 4.26
Quadruple two-to-one-line multiplexer.

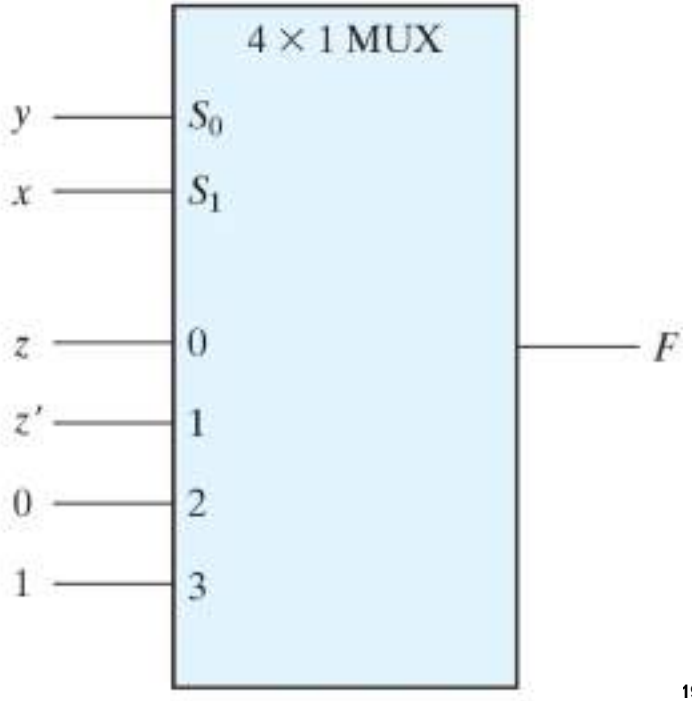


- 부울 함수의 구현: 그림 4.21처럼 디코더는 OR게이트를 추가해서 임의의 부울 함수를 구현할 수 있었음.
  - n-1개의 선택 입력을 갖는 멀티플렉서를 이용하여 n개의 변수를 갖는 부울 함수를 구현 가능
  - 함수의 처음 n-1개의 변수는 멀티플렉서의 선택 입력선으로 연결하고 남은 하나의 변수는 데이터 입력으로 사용
  - 예를 들어, F(x, y, z) = Σ(1, 2, 6, 7)

Figure 4.27
Implementing a Boolean function with a multiplexer.

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

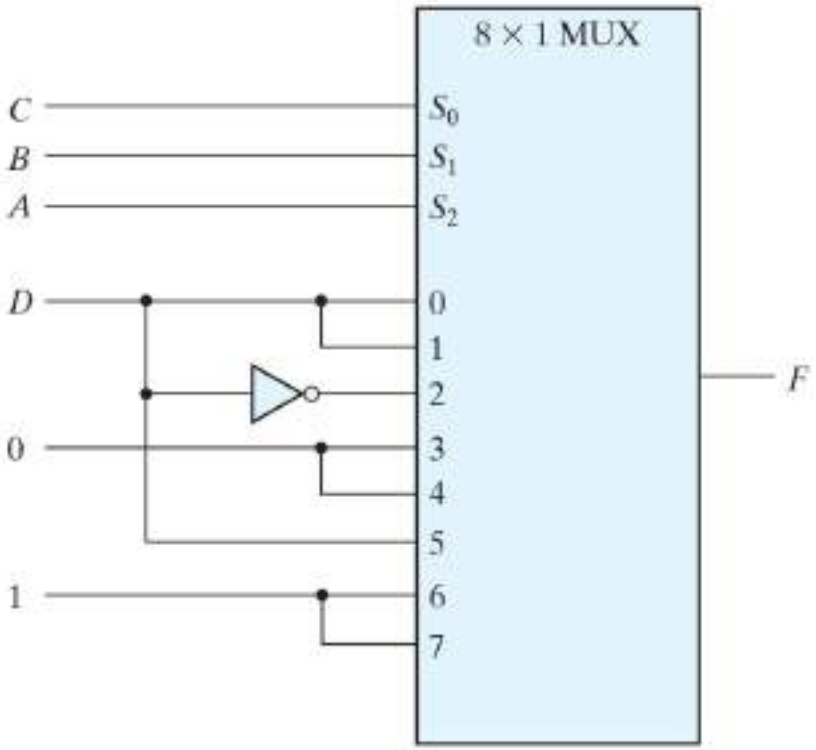


(b) Multiplexer implementation

– 예를 들어, F(A, B, C, D) = Σ(1, 3, 4, 11, 12, 13, 14, 15)

Figure 4.28
Implementing a four-input function with a multiplexer.

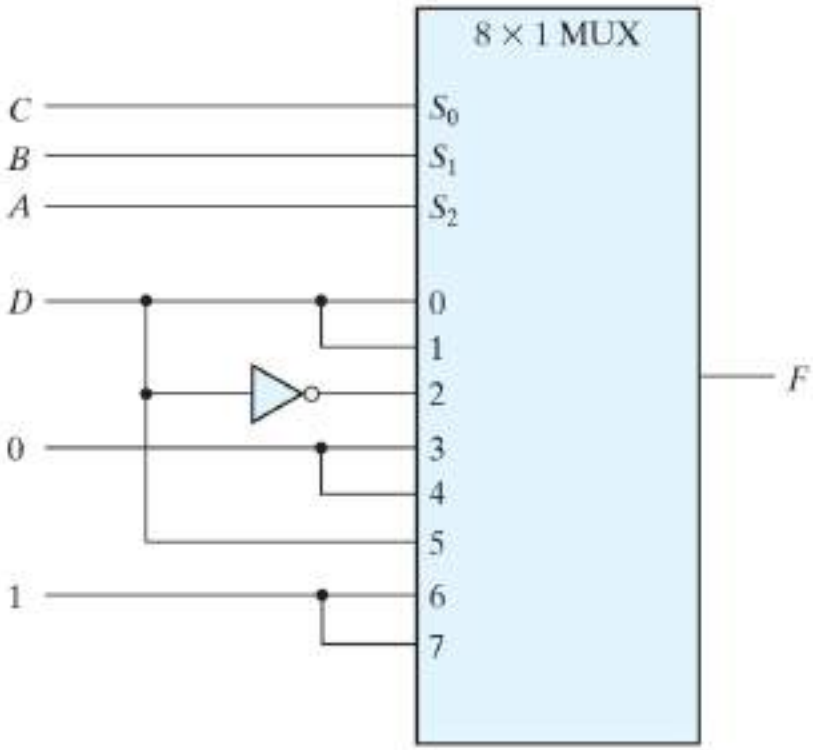
A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	



– 예를 들어, F(A, B, C, D) = Σ(1, 3, 4, 11, 12, 13, 14, 15)

Figure 4.28
Implementing a four-input function with a multiplexer.

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	



- 3-상태 게이트
  - 멀티플렉서는 3-상태 게이트로 구성될 수 있음.
  - 3- 상태란 논리 1과 0에 대응하는 신호와 고임피던스(hign-impedance) 상태
  - 일반적으로 버퍼 게이트로 사용됨
  - 고임피던스 상태란
    1. 개방회로와 같은 동작. 출력은 끊어진 것처럼 보임
    2. 논리적으로는 의미가 없음
    3. 3-상태 게이트 출력에 연결된 회로는 게이트의 입력에 의한 영향을 받지 않음

Figure 4.29
Graphic symbol for a three-state buffer.

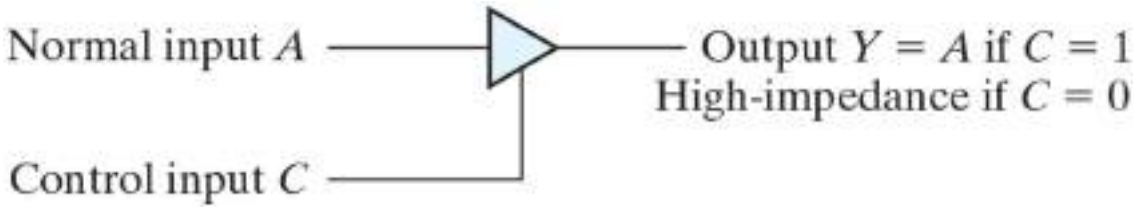


Figure 4.30
Multiplexers with three-state gates.