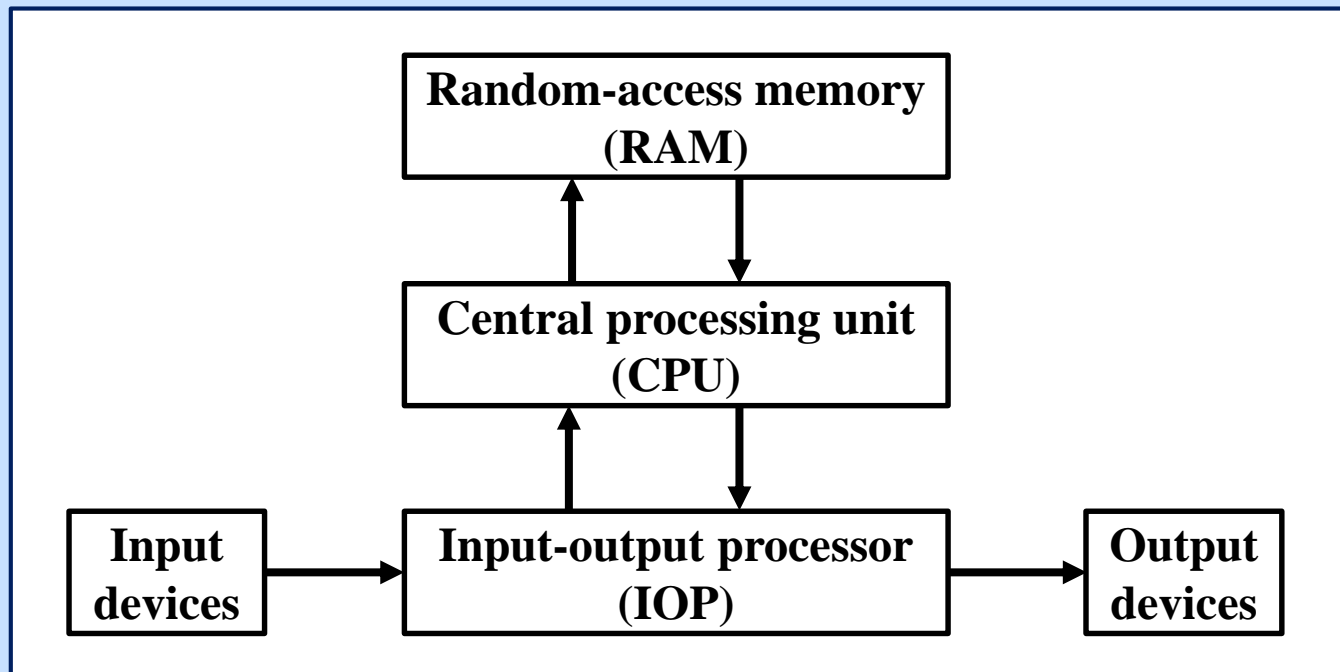


디지털 논리회로

1.1 디지털 컴퓨터의 구성

◆ 컴퓨터 구성

- 중앙처리장치 : 산술 및 논리 연산 부분, 레지스터, 제어 회로 등으로 구성
- 기억장치 : 명령어와 데이터를 저장 (RAM)
- 입출력 프로세서 : 컴퓨터와 외부 세계와의 통신과 데이터 전송을 제어









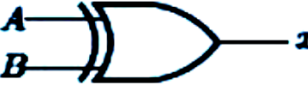

◆ 컴퓨터 구조적 설계

- 컴퓨터 조직 (organization) : 하드웨어 구성품들의 동작 방식과 이들의 연결 방식에 대한 것으로 사용자가 이용할 수 없는 구조의 구현 (파이프라인 구조 등)
- 컴퓨터 설계 (design) - 제시된 컴퓨터의 사양에 따라 적절한 하드웨어를 선택하고 그들간의 연결 방식을 결정하여 컴퓨터 하드웨어를 설계
- 컴퓨터 구조 (architecture) - 사용자의 입장에서 컴퓨터의 구조나 동작에 관심을 두고서 정보의 형식이나 명령어 집합, 메모리 주소 기법 등을 연구, 사용자가 직접 이용할 수 있는 부분 (명령어 세트, 레지스터, 메모리 관리 테이블 구조, 예외 처리 모델 등)
- 컴퓨터의 구조적 설계는 프로세서나 메모리 같은 다양한 기능적 모듈의 사양을 가지고 컴퓨터 시스템을 설계하는 것

1.2 논리 게이트

◆ 입력 논리의 필요 조건을 만족할 때 1 또는 0을 만드는 하드웨어 블록

AND	 $x=A \cdot B$ or $x=AB$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	x	0	0	0	0	1	0	1	0	0	1	1	1
A	B	x															
0	0	0															
0	1	0															
1	0	0															
1	1	1															
OR	 $x=A+B$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	x	0	0	0	0	1	1	1	0	1	1	1	1
A	B	x															
0	0	0															
0	1	1															
1	0	1															
1	1	1															
Inverter	 $x=A'$	<table><tr><th>A</th><th>x</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	x	0	1	1	0									
A	x																
0	1																
1	0																
Buffer	 $x=A$	<table><tr><th>A</th><th>x</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	A	x	0	0	1	1									
A	x																
0	0																
1	1																

NAND	 $x = (AB)'$	<table> <tr> <th>A</th><th>B</th><th>x</th></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table>	A	B	x	0	0	1	0	1	1	1	0	1	1	1	0
A	B	x															
0	0	1															
0	1	1															
1	0	1															
1	1	0															
NOR	 $x = (A+B)'$	<table> <tr> <th>A</th><th>B</th><th>x</th></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table>	A	B	x	0	0	1	0	1	0	1	0	0	1	1	0
A	B	x															
0	0	1															
0	1	0															
1	0	0															
1	1	0															
Exclusive-OR (XOR)	 $x = A \oplus B$ or $x = A'B + AB'$	<table> <tr> <th>A</th><th>B</th><th>x</th></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table>	A	B	x	0	0	0	0	1	1	1	0	1	1	1	0
A	B	x															
0	0	0															
0	1	1															
1	0	1															
1	1	0															
Exclusive-NOR or equivalence	 $x = (A \oplus B)'$ or $x = A'B' + AB$	<table> <tr> <th>A</th><th>B</th><th>x</th></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	A	B	x	0	0	1	0	1	0	1	0	0	1	1	1
A	B	x															
0	0	1															
0	1	0															
1	0	0															
1	1	1															

1.3 부울 대수

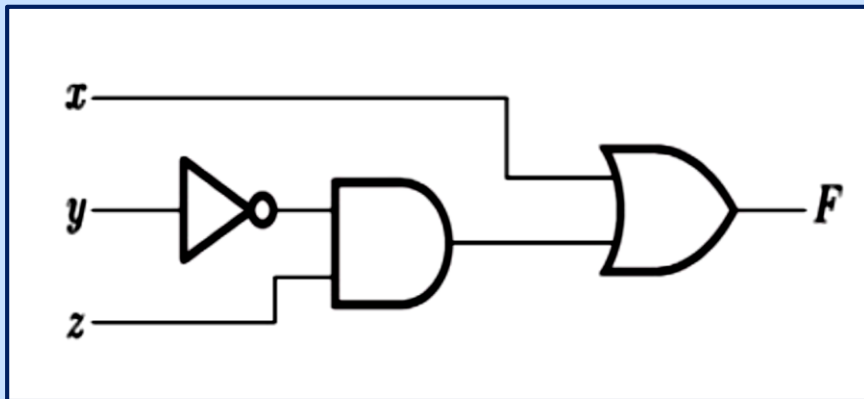
◆ 기본적인 논리 동작 : AND, OR, 보수

◆ 부울 대수는 디지털 회로의 해석과 설계를 쉽게 하는데 목적이 있음

- 변수 사이의 진리표 관계를 대수 형식으로 표시할 때 편리
- 논리도의 입출력 관계를 대수 형식으로 표시할 때 편리
- 같은 기능을 가진 더 간단한 회로로 표현할 때 편리

◆ 진리표와 논리도

- $F = x + y'z$



x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

◆ 부울 대수의 기본 관계

$$(1) \ x + 0 = x$$

$$(3) \ x + 1 = 1$$

$$(5) \ x + x = x$$

$$(7) \ x + x' = 1$$

$$(9) \ x + y = y + x$$

$$(11) \ x + (y + z) = (x + y) + z$$

$$(13) \ x(y + z) = xy + xz$$

$$(15) \ (x + y)' = x'y'$$

$$(17) \ (x')' = x$$

$$(2) \ x \cdot 0 = 0$$

$$(4) \ x \cdot 1 = x$$

$$(6) \ x \cdot x = x$$

$$(8) \ x \cdot x' = 0$$

$$(10) \ xy = yx$$

$$(12) \ x(yz) = (xy)z$$

$$(14) \ x + yz = (x + y)(x + z)$$

$$(16) \ (xy)' = x' + y'$$

◆ De Morgan 정리 (NOR와 NAND를 취급하는데 매우 중요)

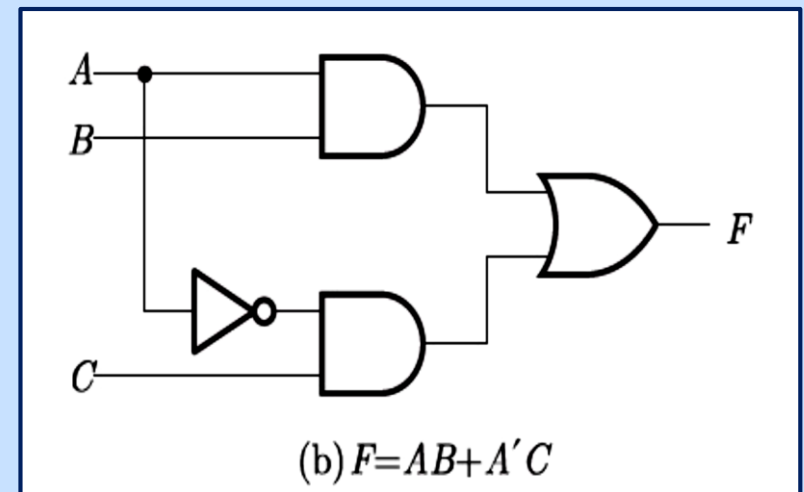
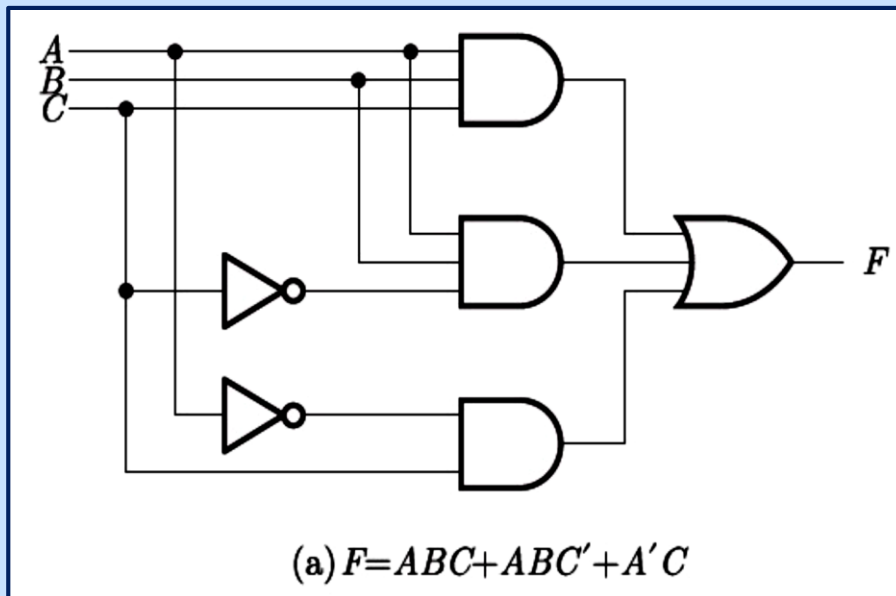
$$(x + y + z)' = x'y'z', \quad (xyz)' = x' + y' + z'$$

- 모든 OR 연산은 AND로, 모든 AND 연산은 OR로 바꾸어줌

$$F = AB + C'D' + B'D, \quad F' = (AB)'(C'D')'(B'D)' = (A' + B')(C + D)(B + D')$$

◆ 간소화

$$F = ABC + ABC' + A'C = AB(C + C') + A'C = AB + A'C$$



1.4 맵의 간소화

◆ 맵을 사용한 부울 함수의 간소화 : Karnaugh 맵, Veitch 다이어그램

$$- F(x, y, z) = \sum(1, 4, 5, 6, 7)$$

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

- 두 개, 세 개, 네 개 변수를 갖는 함수에 대한 맵

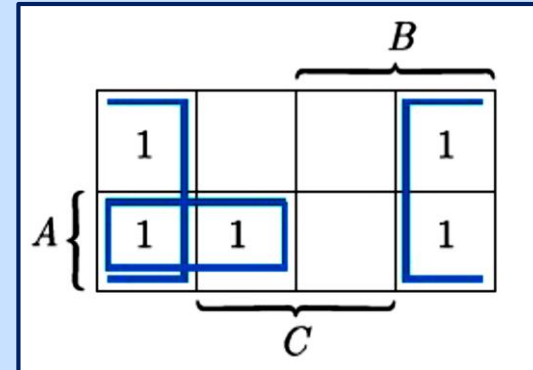
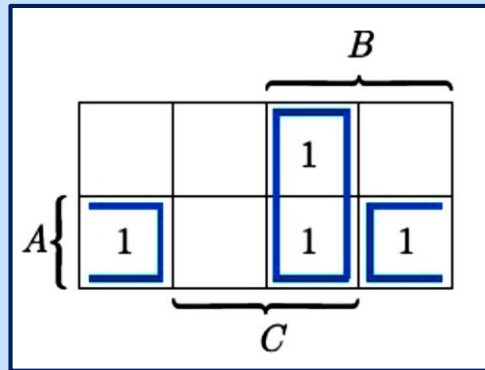
		B	
		0	1
A	0	0	1
	1	2	3

		B			
		00	01	11	10
A	0	0	1	3	2
	1	4	5	7	6

		C			
		00	01	11	10
A	00	0	1	3	2
	01	4	5	7	6
	11	12	13	15	14
	10	8	9	11	10

- 예1) $F(A, B, C) = \sum(3, 4, 6, 7) = BC + AC'$

- 예2) $F(A, B, C) = \sum(0, 2, 4, 5, 6) = C' + AB'$



◆ 논리곱의 논리합 (sum of products), 논리합의 논리곱 (product of sums)

- 예4) $F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$

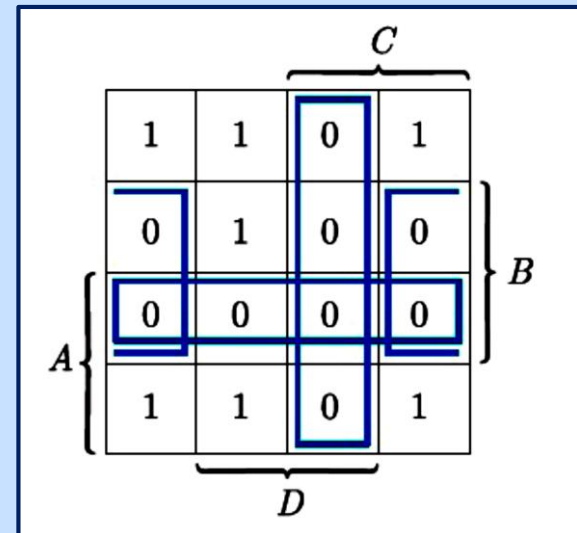
① sum of products

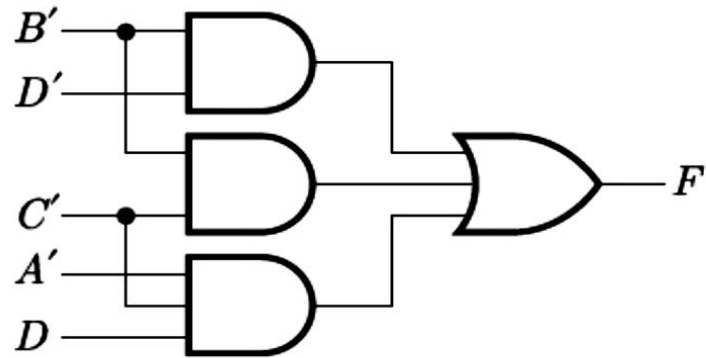
$$F = B'D' + B'C' + A'C'D$$

② product of sums

$$F' = AB + CD + BD'$$

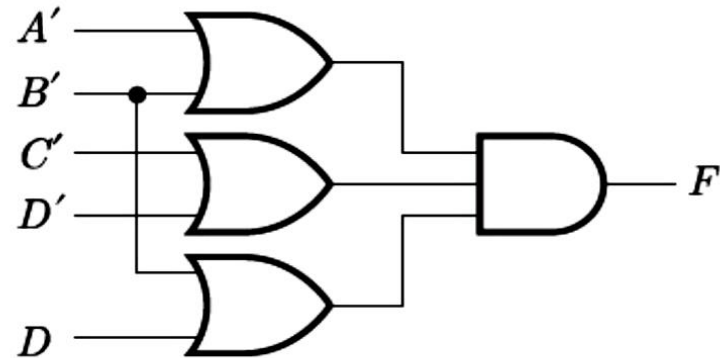
$$F = (A' + B')(C' + D')(B' + D)$$





(a) Sum of products:

$$F = B'D' + B'C' + A'C'D$$



(b) Product of sums:

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

◆ Don't Care 조건

– 예5) $F(A, B, C) = \sum(0, 2, 6)$

$D(A, B, C) = \sum(1, 3, 5)$

X를 0로 간주하면 $F = A'C' + BC'$

X를 1로 간주하면 $F = A' + BC'$

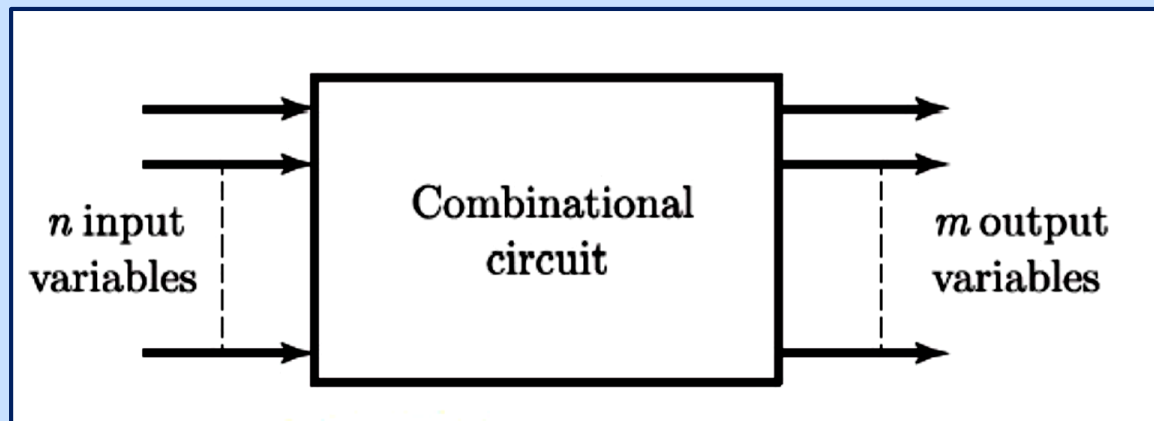
	B		
	1	x	x
A	0	x	0
	1	1	1
	C		

1.5 조합 회로

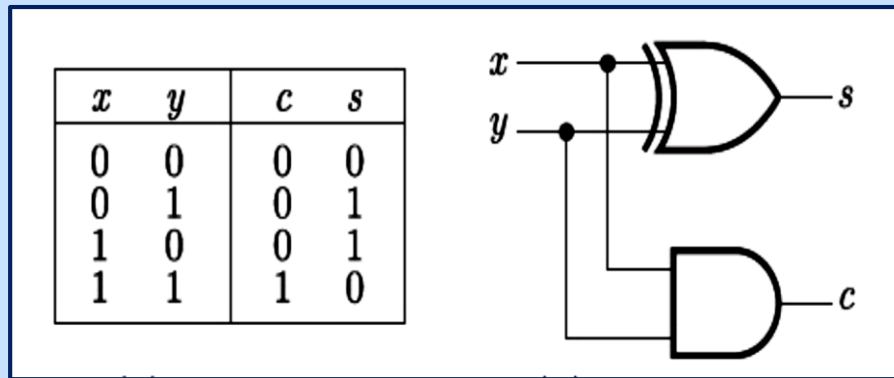
◆ 입력과 출력을 가진 논리 게이트의 집합으로 출력은 입력들의 조합의 함수

◆ 조합 회로 설계 절차

- 문제가 주어지면 입력과 출력 변수에 문자 기호를 붙임
- 입력과 출력 사이의 관계를 정의하는 진리표를 유도
- 각 출력에 대한 간소화된 부울 함수를 얻음
- 논리도를 그림



◆ 반가산기 : 비트 두 개를 서로 산술적으로 가산하는 조합 회로

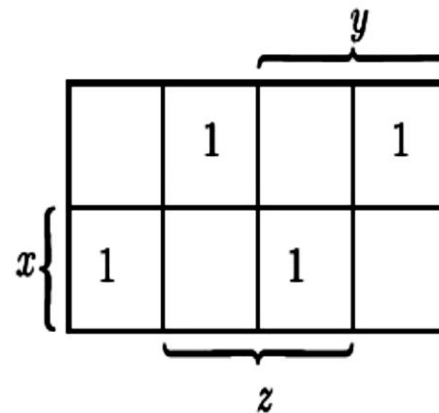


$$S = x'y + xy' = x \oplus y$$

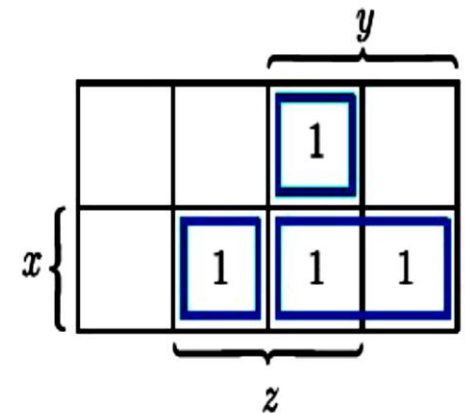
$$C = xy$$

◆ 전가산기 : 비트 두 개와 이전 단의 캐리를 가산하는 조합 회로 (1비트 덧셈)

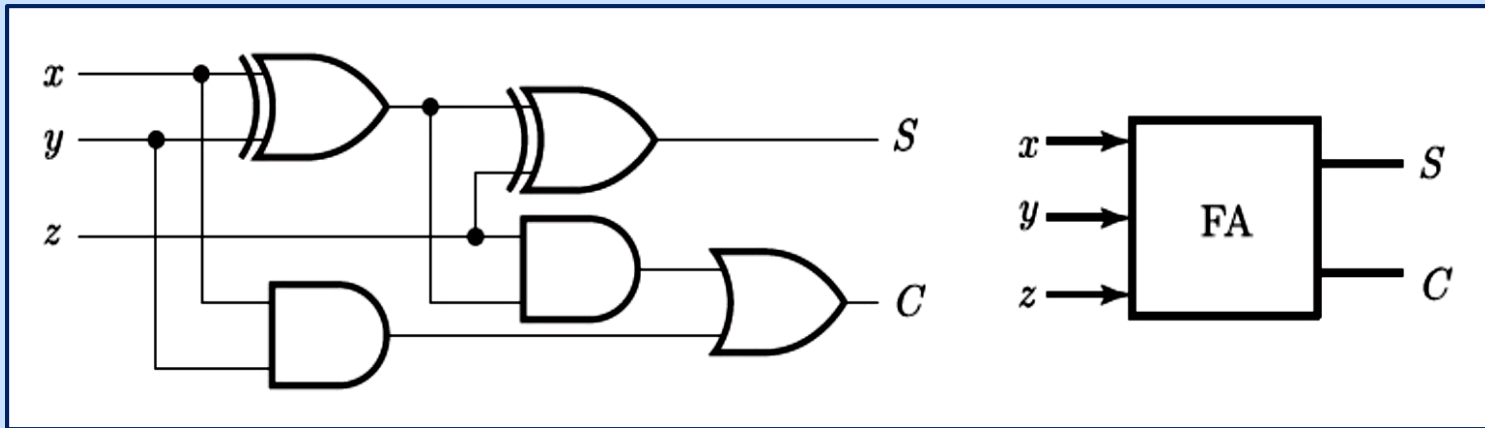
입력			출력	
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



$$\begin{aligned}
 S &= x'y'z + x'yz' + xy'z' + xyz \\
 &= x \oplus y \oplus z
 \end{aligned}$$



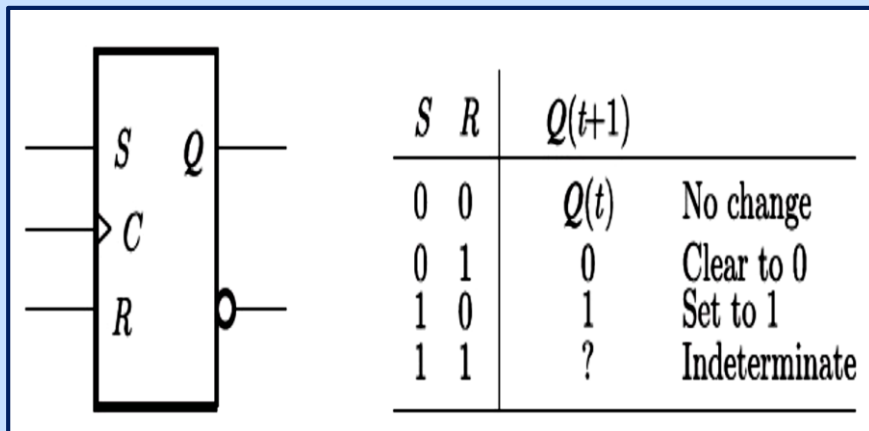
$$\begin{aligned}
 C &= xy + xz + yz \\
 &= xy + (x'y + xy')z \\
 &= xy + (x \oplus y)z
 \end{aligned}$$



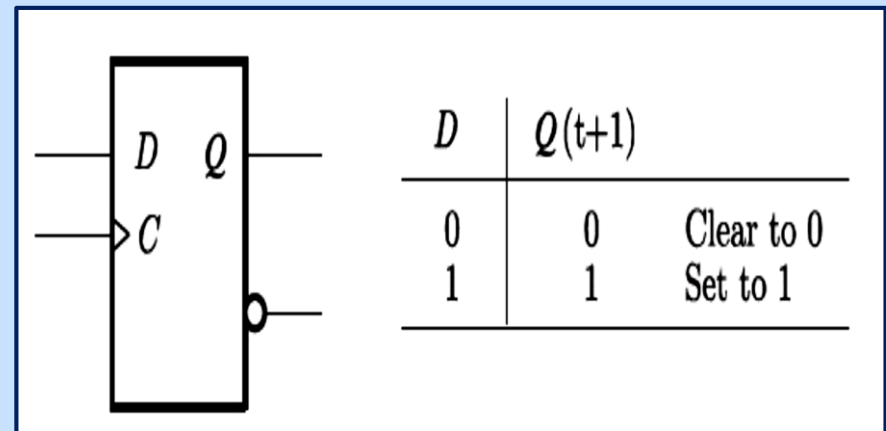
1.6 플립플롭

◆ 플립플롭은 한 비트의 정보를 저장하는 이진 셀로 입력 펄스가 상태 변환을 일으키기 전까지는 이진 상태를 그대로 유지

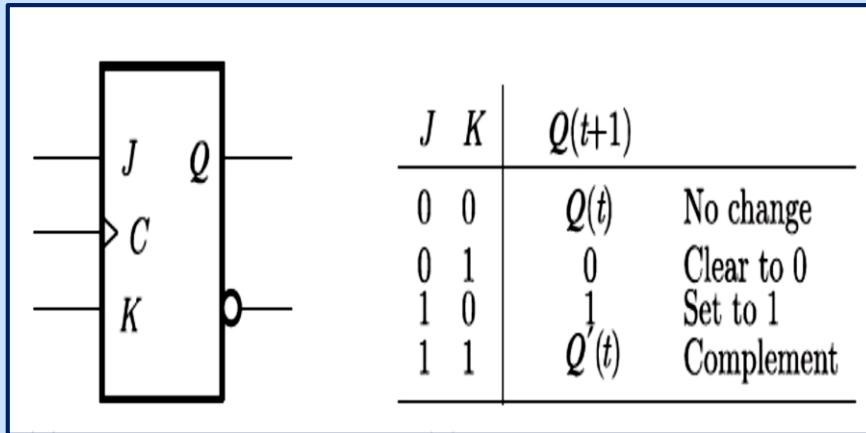
– SR 플립플롭



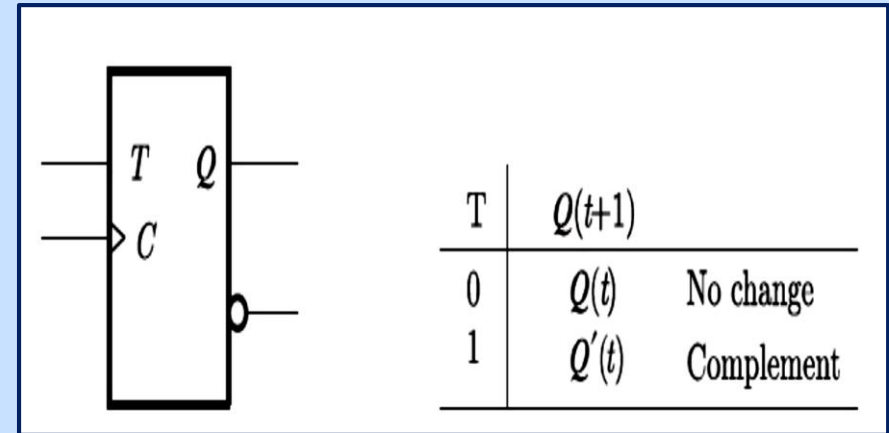
– D 플립플롭



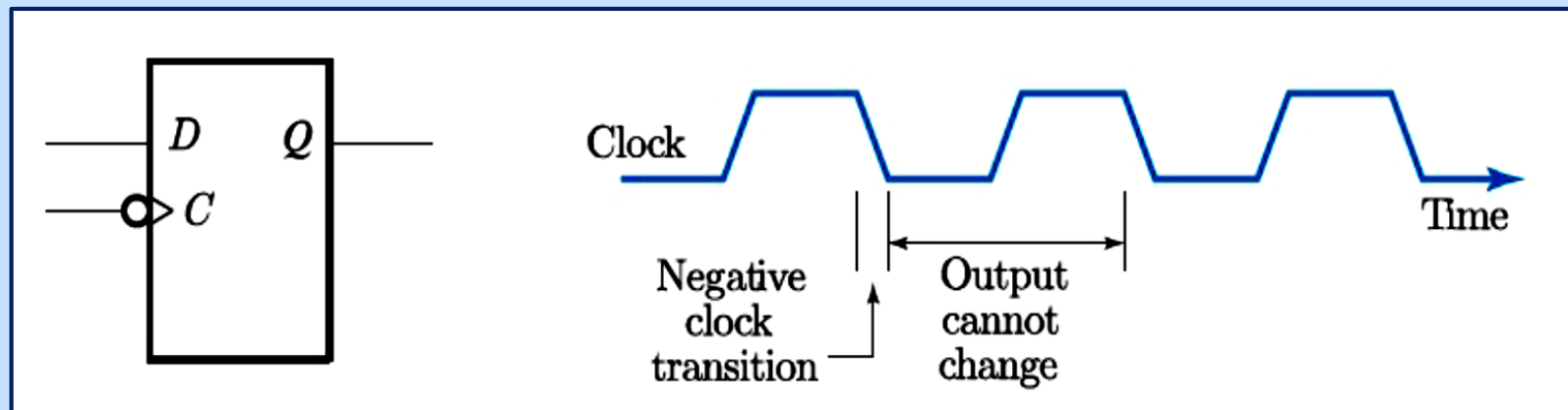
- JK 플립플롭



- T 플립플롭



◆ 모서리-변이형 플립플롭 : 상태 변경을 클럭 펄스의 변이 동안 동기화함



- ◆ 플립플롭 여기표 : 현재 상태와 다음 상태를 알 때 플립플롭에 어떤 입력을 주어야 하는 지를 나타낸 표

SR 플립플롭			
Q(t)	Q(t+1)	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

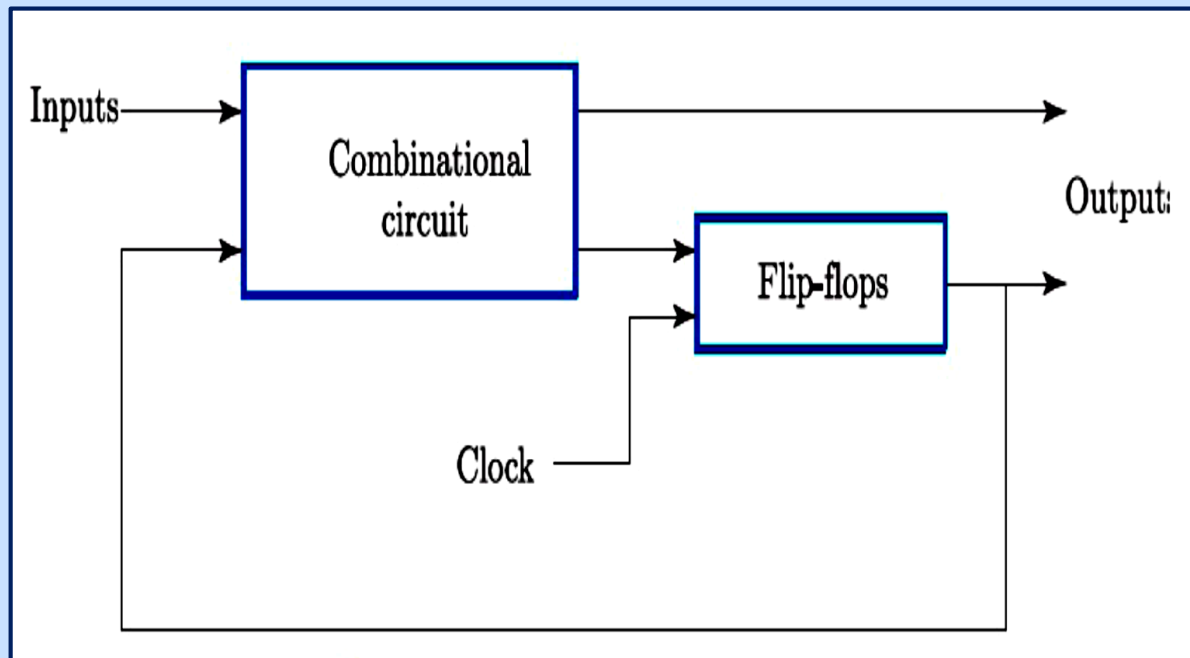
D 플립플롭		
Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

JK 플립플롭			
Q(t)	Q(t+1)	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

T 플립플롭		
Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

1.7 순차 회로

◆ 플립플롭과 게이트를 서로 연결한 회로

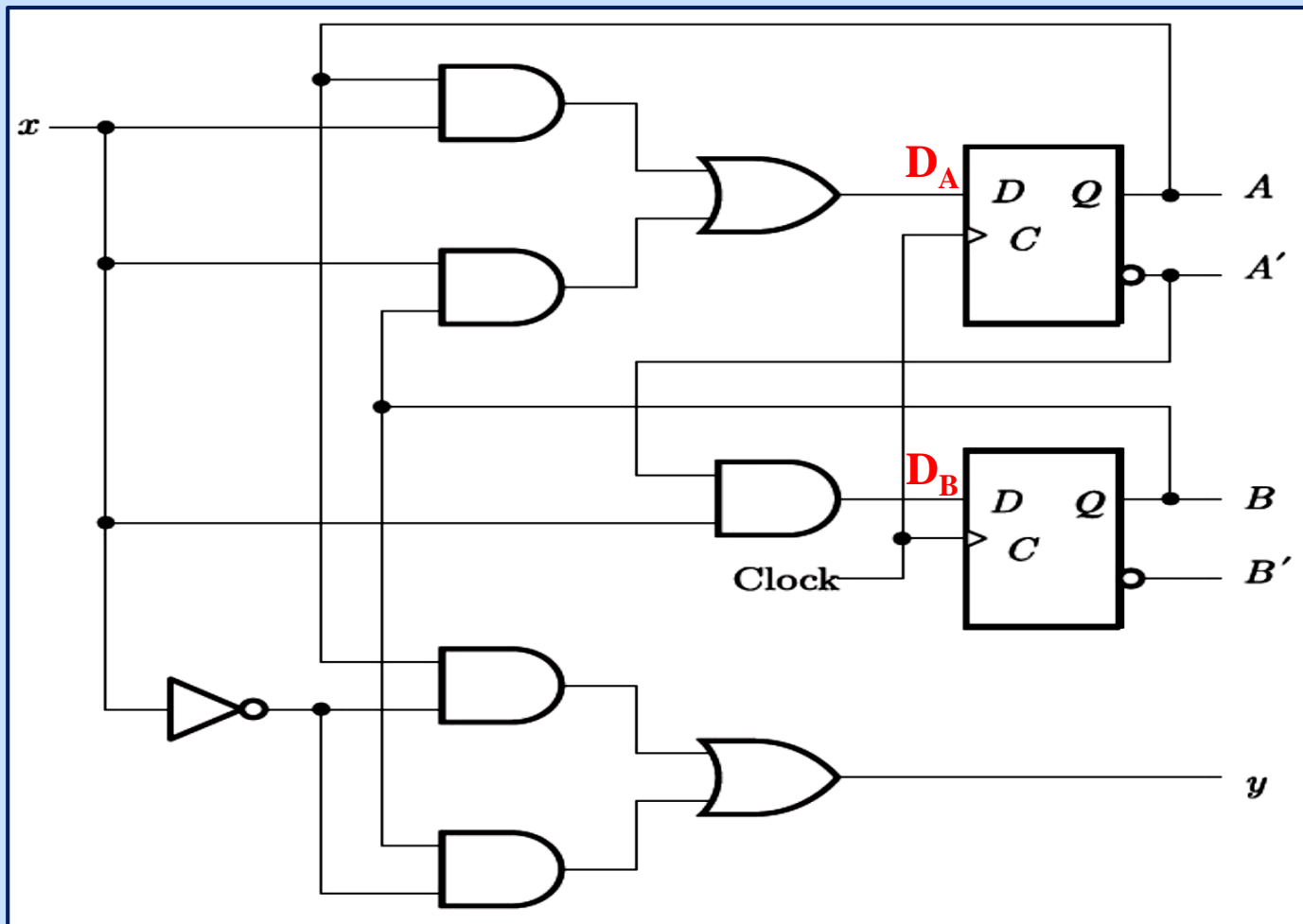


- 순차 회로의 예

$$D_A = Ax + Bx,$$

$$D_B = A'x,$$

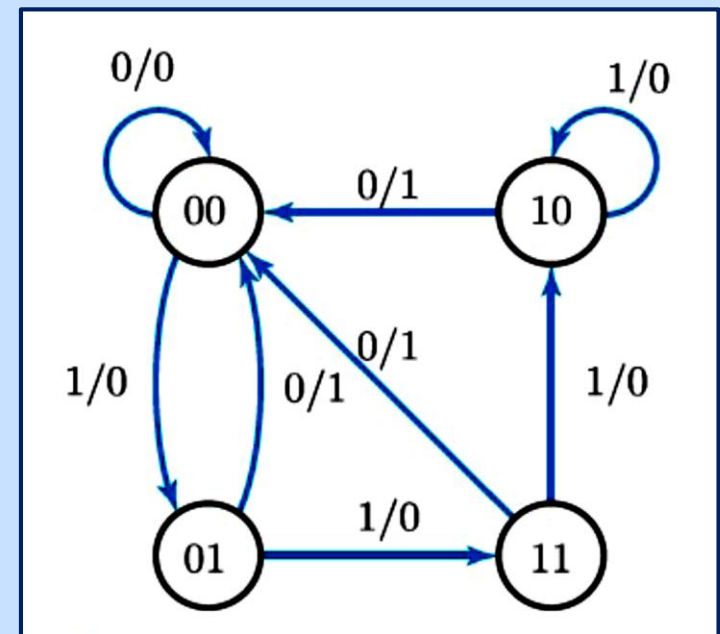
$$y = Ax' + Bx'$$



◆ 상태표와 상태도

- 출력과 다음 상태는 모두 입력과 현재 상태에 의해 결정
- 상태도에서 상태는 원으로 표시하고 상태 사이의 천이는 원 사이를 연결하는 직선으로 표시

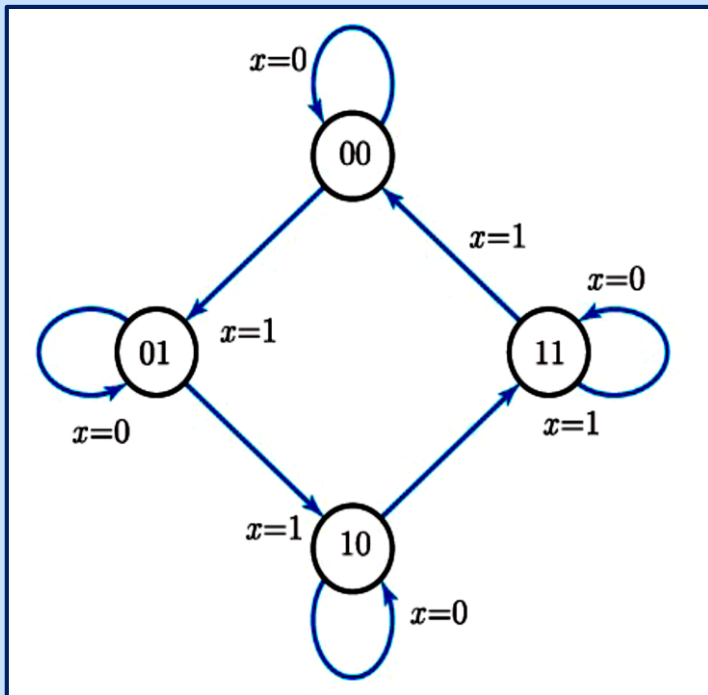
현재 상태		입력	다음 상태		출력
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0



◆ 순차 회로의 설계 절차

- 회로의 특성으로부터 상태도를 그림
- 상태의 비트수로부터 필요한 플립플롭의 개수와 회로의 입력수를 결정
- 상태표를 작성하고 확장하여 플립플롭의 여기표를 만들
- karnaugh 맵을 이용하여 간소화된 입력조건을 구한 후에 순차 회로를 구성

◆ 설계 예 : 2비트 이진 카운터



현재 상태		입력	다음 상태		플립플롭 입력			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	0	0	×	0	×
0	0	1	0	1	0	×	1	×
0	1	0	0	1	0	×	×	0
0	1	1	1	0	1	×	×	1
1	0	0	1	0	×	0	0	×
1	0	1	1	1	×	0	1	×
1	1	0	1	1	×	0	×	0
1	1	1	0	0	×	1	×	1

