Computer System Architecture(1,2장)

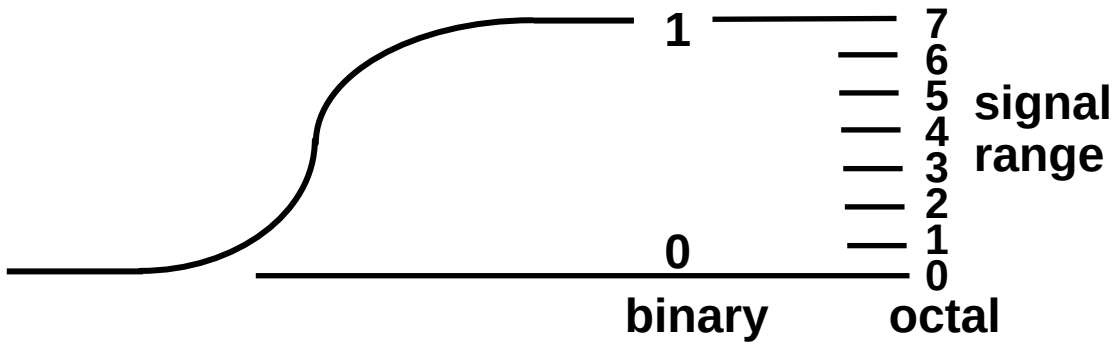
Hanbat National University Prof. Lee Jaeheung

DIGITAL LOGIC CIRCUITS

- 논리 게이트(Logic Gates)
- 부울 함수(Boolean Algebra)
- 맵의 간소화(Map Simplification)
- 조합 회로(Combinational Circuits)
- 플립 플롭(Flip-Flops)
- 순차회로(Sequential Circuits)
- 메모리 구성(Memory Components)
- 집적 회로(Integrated Circuits)

LOGIC GATES

- 디지털 컴퓨터
  - 컴퓨터가 2진 바이너리 신호를 이용하여 디지털 정보를 표현
  - 왜 2진수인가 ? 10진수 외 다른 수 대신 이용이 불가능할까 ?
    - Electronic signal

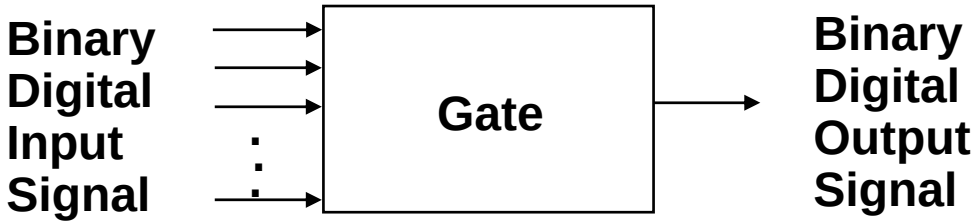


- Consider the calculation cost - Add

	0	1
0	0	1
1	1	10

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

BASIC LOGIC BLOCK - GATE -



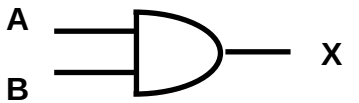
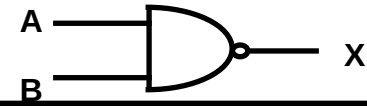
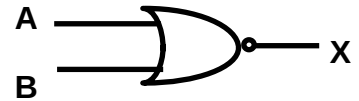
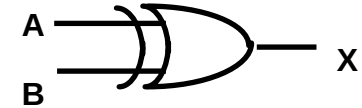
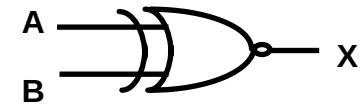
❖ 두 타입의 기초 블럭

- Combinational (조합) 논리 블록
  - Logic Blocks whose output logic value
  - 오직 현재 입력 값에 의해 출력값이 결정
- Sequential (순차) 논리 블록
  - Logic Blocks whose output logic value
  - 현재 입력값과 블록의 상태값(저장된 정보)에 의해 출력값이 결정

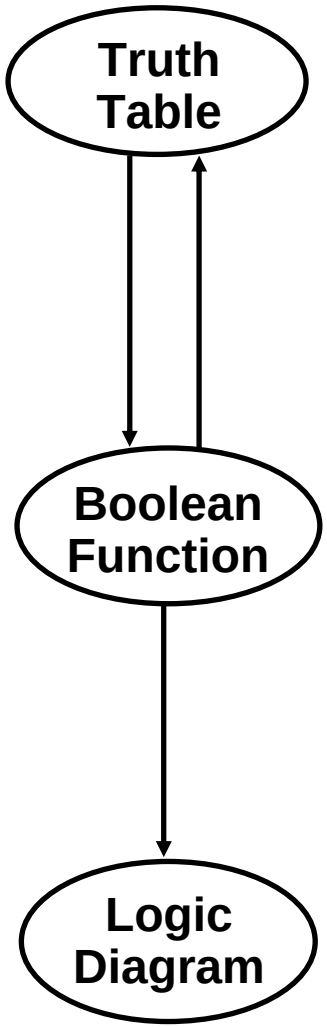
❖ 회로 함수의 표현 방법

- Truth Table
- Boolean Function
- Karnaugh Map

COMBINATIONAL GATES

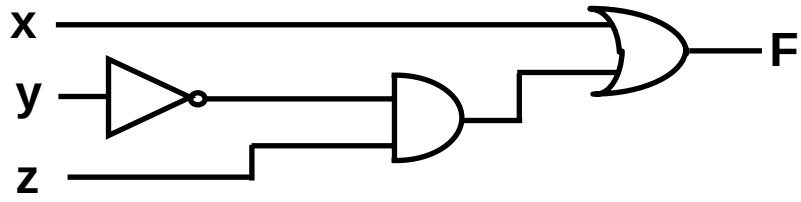
Name	Symbol	Function	Truth Table
AND		X = A • B or X = AB	A B | X 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		X = A + B	A B | X 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
I		X = A’	A | X 0 | 1 1 | 0
Buffer		X = A	A | X 0 | 0 1 | 1
NAND		X = (AB)’	A B | X 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		X = (A + B)’	A B | X 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR Exclusive OR		X = A ⊕ B or X = A’B + AB’	A B | X 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
XNOR Exclusive NOR or Equivalence		X = (A ⊕ B)’ or X = A’B’+ AB	A B | X 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

LOGIC CIRCUIT DESIGN



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

$F = x + y'z$



BOOLEAN ALGEBRA

부울 대수

- 2진 변수와 논리 연산자로 구성된 대수
- 부울 대수는 디지털 논리합성과 분석에 유용
  - 입출력 신호는 부울 변수에 의해 표현
  - 디지털 논리 함수는 논리연산자에 의해서 표현(Boolean Function)
  - 부울 함수로 부터 AND, OR, I 로 논리 다이어그램을 구성

진리표

- 디지털 논리 회로의 가장 기초적인 표현은 진리표
  - 입력값의 모든 조합(MINTERMS)으로 출력을 표현한 테이블
  - n 입력 변수 → 2n minterms

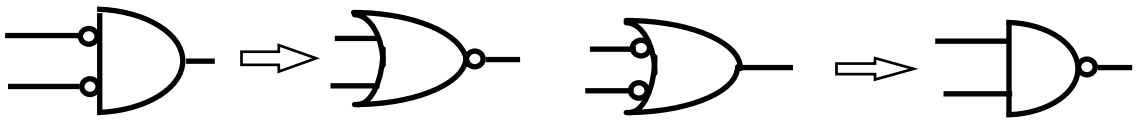
부울 대수에 의한 기본 정의

[1] x + 0 = x	[2] x • 0 = 0
[3] x + 1 = 1	[4] x • 1 = x
[5] x + x = x	[6] x • x = x
[7] x + x' = 1	[8] x • X' = 0
[9] x + y = y + x	[10] xy = yx
[11] x + (y + z) = (x + y) + z	[12] x(yz) = (xy)z
[13] x(y + z) = xy +xz	[14] x + yz = (x + y)(x + z)
[15] (x + y)' = x'y'	[16] (xy)' = x' + y'
[17] (x')' = x	

❖ 테이블의 용도

[15] and [16] : De Morgan's Theorem

- 부울 대수에 의한 간소화
- 다른 logic gates를 사용한 논리 블럭도를 얻기 위하여 동일한 부울함 수 유도
  - -- 보통 AND, OR, I
  - -- 이와 다른 형식으로 NANDs 나 NORs 로 표현
    - → De Morgans 정의에 의해 적용
      - x'y' = (x + y)' — I, AND → NOR
      - x'+ y'= (xy)' — I, OR → NAND

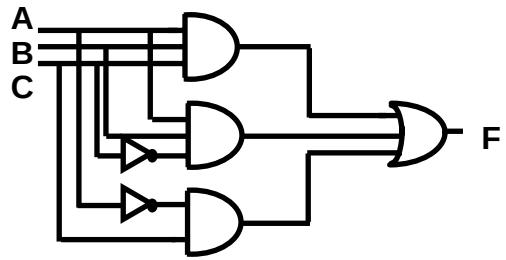


EQUIVALENT CIRCUITS

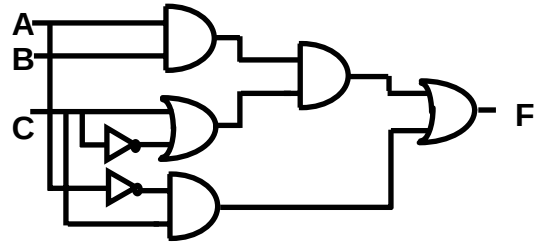
Many different logic diagrams are possible for a given Function

$$F = ABC + ABC' + A'C \quad \ldots \quad (1)$$
$$= AB(C + C') + A'C \quad [13] \quad (2)$$
$$= AB \cdot 1 + A'C \quad [7]$$
$$= AB + A'C \quad [4] \quad (3)$$

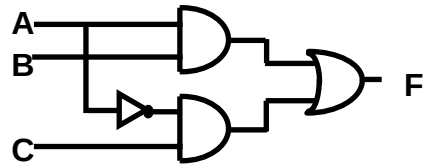
(1)



(2)



(3)



COMPLEMENT OF FUNCTIONS

디지털 논리 회로는 부울 함수를 이용하여 AND, OR, 그리고 Invert operator을 사용되어 표현 되어야 한다. → 부울 함수를 보충하는것.

- 모든값과 수식을 각각 정렬하여 괄호 안에 표현 한다.

$A,B,...,Z,a,b,...,z \Rightarrow A',B',...,Z',a',b',...,z'$

$(p + q) \Rightarrow (p + q)'$

- 모든 연산을 조합하여 재배치 한다.

AND ⇒ OR

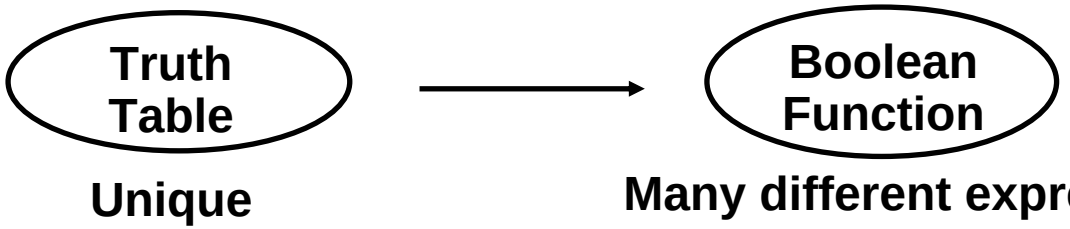
OR ⇒ AND

- 기본적으로, 넓은 범위는 드모르간 법칙을 이용한다.

$(x_1 + x_2 + ... + x_n)' \Rightarrow x_1'x_2'... x_n'$

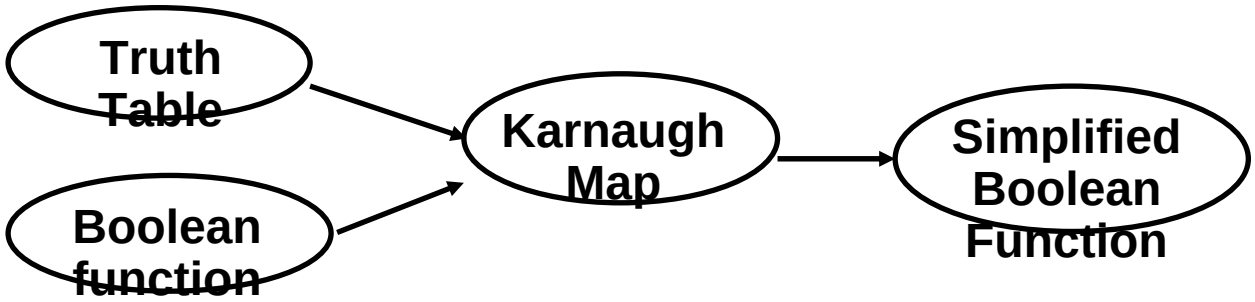
$(x_1x_2 ... x_n)' \Rightarrow x_1' + x_2' +...+ x_n'$

SIMPLIFICATION



❖ 부울 함수에 의한 간소화
- 최적화된 같은 표현을 찾아낸다.
- 간단한 함수로 나타내기 위해서는, 간단하게 완성된 수식을 정리하여야된다.
- 하지만, 복잡한 함수에서는 매우 어려운 일이다.

❖ 카르노 맵은 간단한 부울 함수로 나타낼수 있도록 해준다.



KARNAUGH MAP

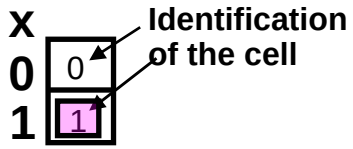
❖ **n**개의 입력을 갖는 논리 회로

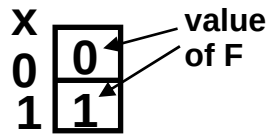
(n-variable sum-of-products form of Boolean Function, or Truth Table)

- **2^n** 개의 셀로 구성
- 각셀은 최소항**(minterm)**
- 함수 값**f(x)**는 최소항의 조합으로 표현
  함수 **f(x)**의 값은 각 입력값의 최소항의 조합
  → **1-**셀**, 0-**셀
- 이진 표현으로 나타내어지는 입력값은 십진수의 표현과 동일

Karnaugh Map

x	F
0	1
1	0

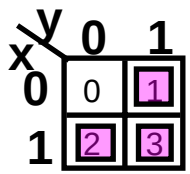


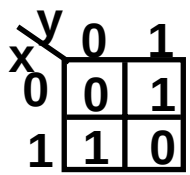


F(x) =Σ (1)

1-cell

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

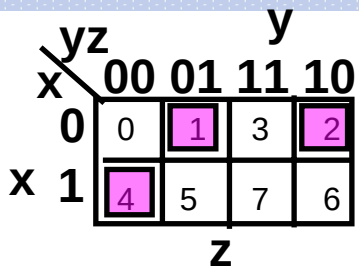




F(x,y) = Σ (1,2)

KARNAUGH MAP

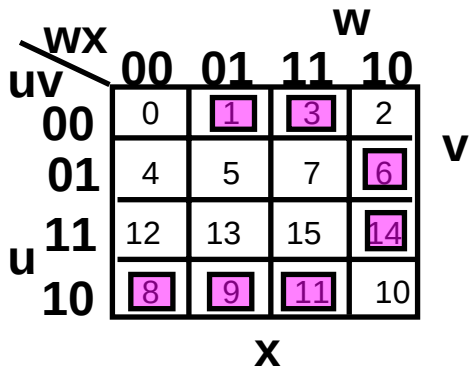
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0



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

$F(x,y,z) = \sum (1,2,4)$

u	v	w	x	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0



uv \ wx	00	01	11	10
00	0	1	1	0
01	0	0	0	1
11	0	0	0	1
10	1	1	1	0

$F(u,v,w,x) = \sum (1,3,6,8,9,11,14)$

MAP SIMPLIFICATION - 2
ADJACENT CELLS -

Rule: xy' +xy = x(y+y') = x

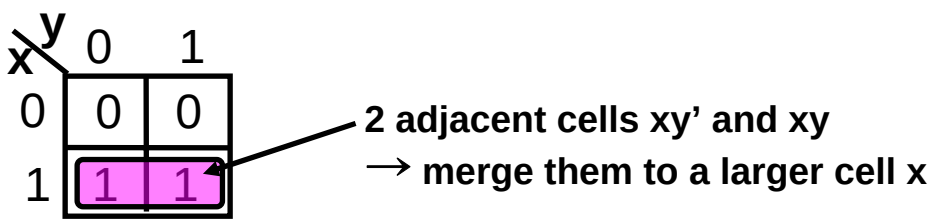
Adjacent cells

- 이진 표현은 비트 안에 있는 것 과 다르다.
  → 인접한 셀 최소항은 각각 다른 변수를 가지고 있다.

- 인접한 셀 **(1,0), (1,1)** 의 최소항은,
  x • y' --> x=1, y=0
  x • y --> x=1, y=1

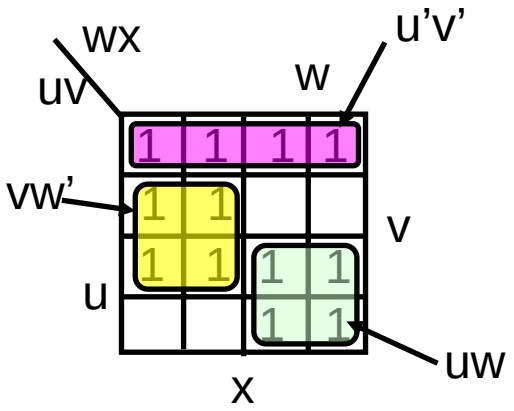
F = xy'+ xy can be reduced to F = x
From the map

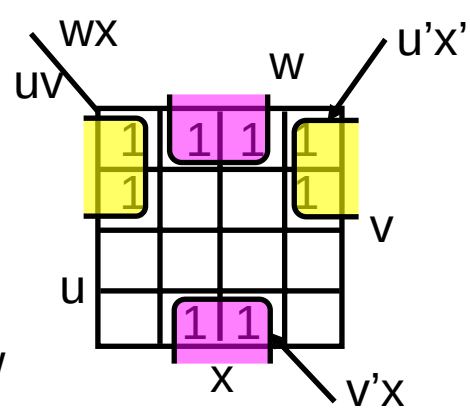
$F(x,y) = \sum (2,3)$
= xy'+ xy
= x



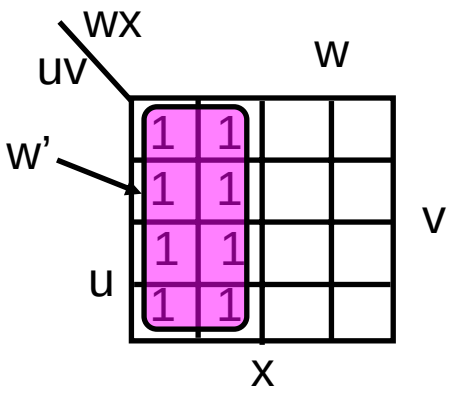
MAP SIMPLIFICATION MORE THAN 2 CELLS

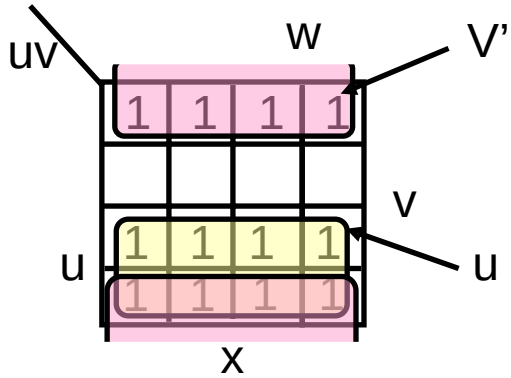
u'v'w'x' + u'v'w'x + u'v'wx + u'v'wx'
= u'v'w'(x'+x) + u'v'w(x+x')
= u'v'w' + u'v'w
= u'v'(w'+w)
= u'v'





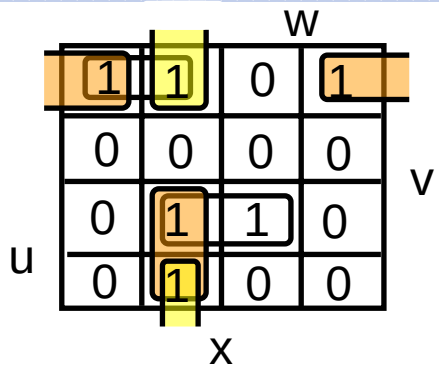
u'v'w'x'+u'v'w'x+u'vw'x'+u'vw'x+uvw'x'+uvw'x+uv'w'x'+uv'w'x
= u'v'w'(x'+x) + u'vw'(x'+x) + uvw'(x'+x) + uv'w'(x'+x)
= u'(v'+v)w' + u(v'+v)w'
= (u'+u)w' = w'





MAP SIMPLIFICATION

uv \ wx	00	01	11	10
00	1	1	0	1
01	0	0	0	0
11	0	1	1	0
10	0	1	0	0



$F(u,v,w,x) = \sum (0,1,2,9,13,15)$

(0,1), (0,2), (0,4), (0,8)
Adjacent Cells of 1
Adjacent Cells of 0
(1,0), (1,3), (1,5), (1,9)
...
...
Adjacent Cells of 15
(15,7), (15,11), (15,13), (15,14)

→

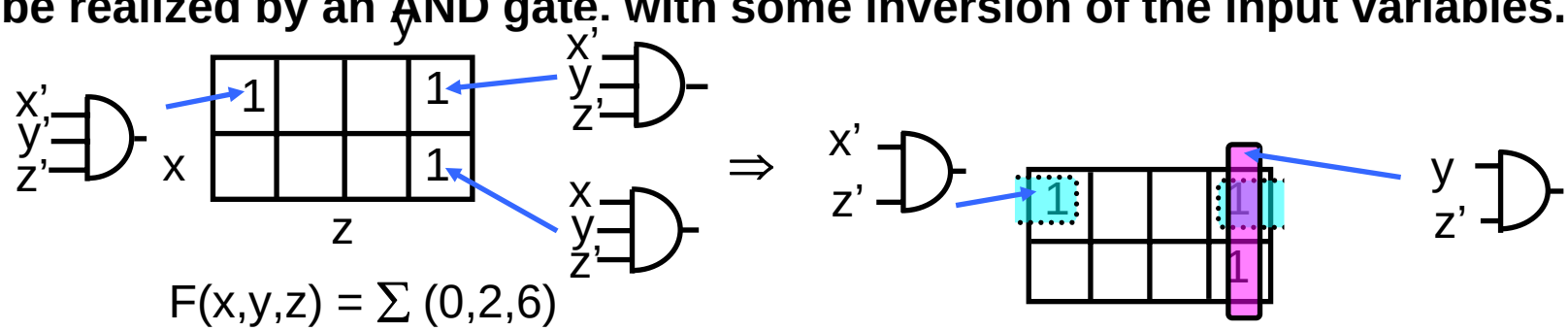
Merge (0,1) and (0,2)
--> u'v'w' + u'v'x'
Merge (1,9)
--> v'w'x
Merge (9,13)
--> uw'x
Merge (13,15)
--> uvx

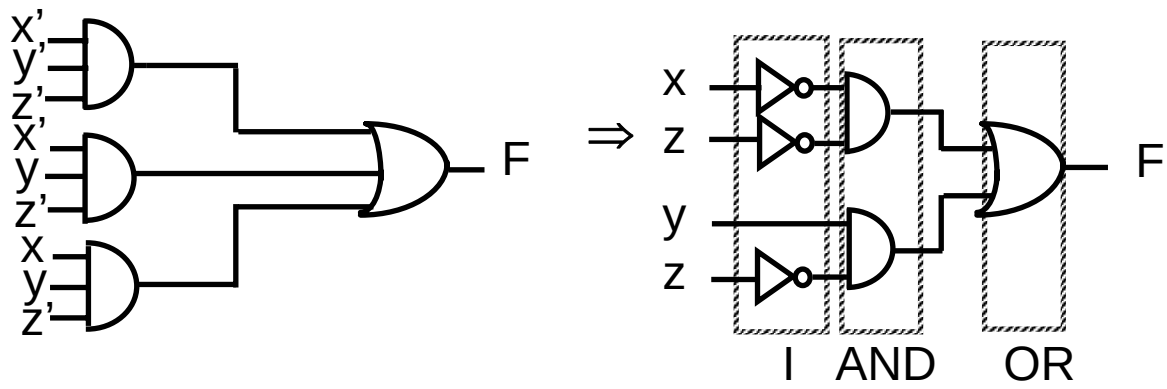
F = u'v'w' + u'v'x' + v'w'x + uw'x + uvx
But (9,13) is covered by (1,9) and (13,15)
F = u'v'w' + u'v'x' + v'w'x + uvx

IMPLEMENTATION OF K-MAPS - Sum-of-Products Form -

Logic function represented by a Karnaugh map can be implemented in the form of I-AND-OR

A cell or a collection of the adjacent 1-cells can be realized by an AND gate. with some inversion of the input variables.



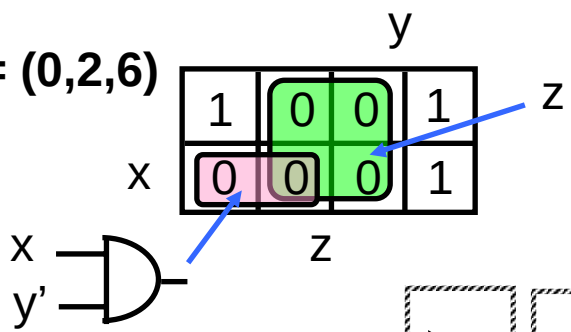


IMPLEMENTATION OF K-MAPS - Product-of-Sums Form -

Logic function represented by a Karnaugh map can be implemented in the form of I-OR-AND

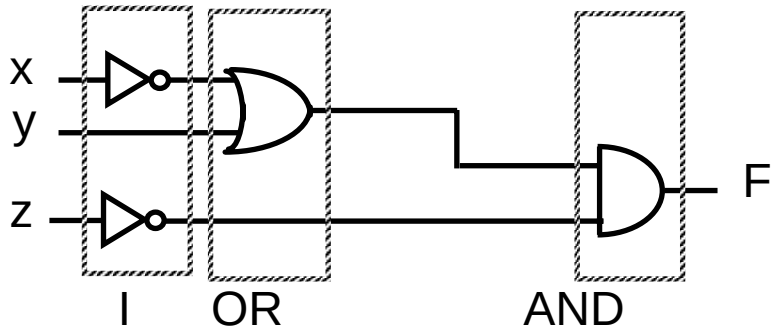
If we implement a Karnaugh map using 0-cells, the complement of F, i.e., F', can be obtained. Thus, by complementing F' using DeMorgan's theorem F can be obtained

F(x,y,z) = (0,2,6)



F' = xy' + z

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



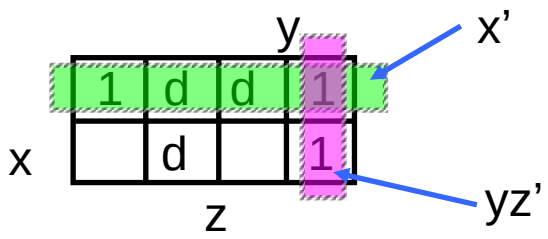
IMPLEMENTATION OF K-MAPS
- Don't-Care Conditions -

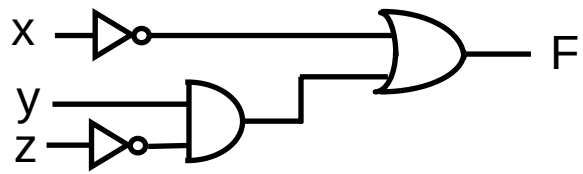
In some logic circuits, the output responses for some input conditions are don't care whether they are 1 or 0.

In K-maps, don't-care conditions are represented by d's in the corresponding cells.

Don't-care conditions are useful in minimizing the logic functions using K-map.

- **Can be considered either 1 or 0**
- **Thus increases the chances of merging cells into the larger cells**
  --> Reduce the number of variables in the product terms

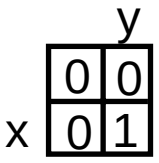




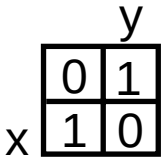
COMBINATIONAL LOGIC CIRCUITS

Half Adder

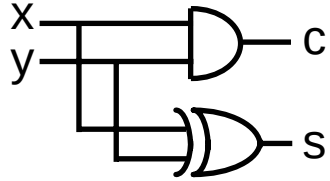
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



$c = xy$

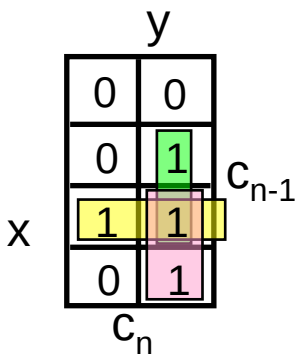


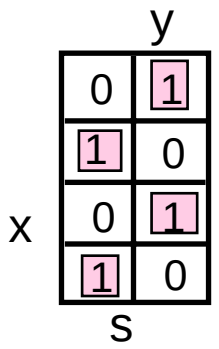
$s = xy' + x'y$
$= x \oplus y$



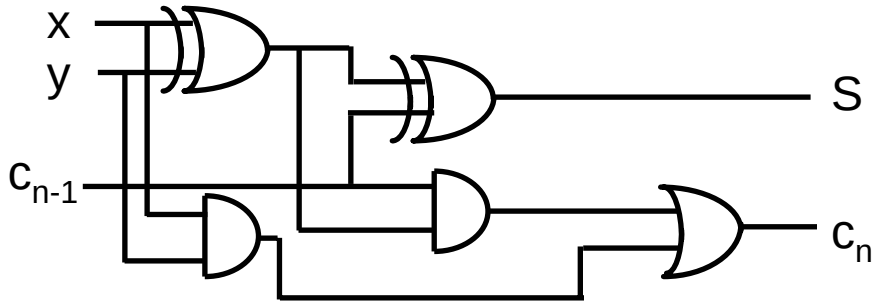
Full Adder

x	y	c_{n-1}	c_n	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





$c_n = xy + xc_{n-1} + yc_{n-1}$
$= xy + (x \oplus y)c_{n-1}$
$s = x'y'c_{n-1} + x'yc'_{n-1} + xyc'_{n-1} + xyc_{n-1}$
$= x \oplus y \oplus c_{n-1} = (x \oplus y) \oplus c_{n-1}$



COMBINATIONAL LOGIC CIRCUITS

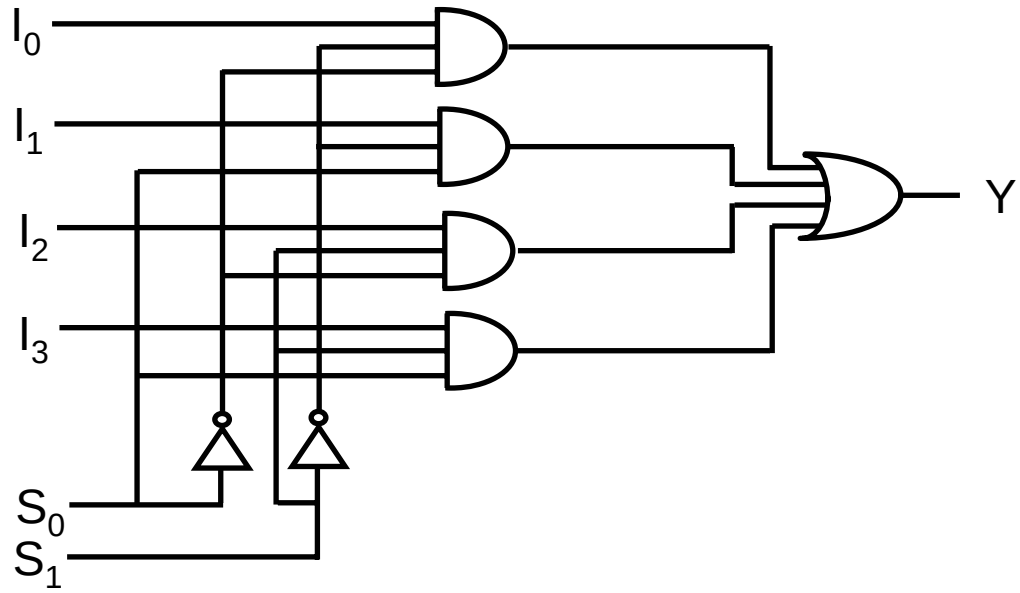
Other Combinational Circuits

Multiplexer
Encoder
Decoder
Parity Checker
Parity Generator
etc

MULTIPLEXER

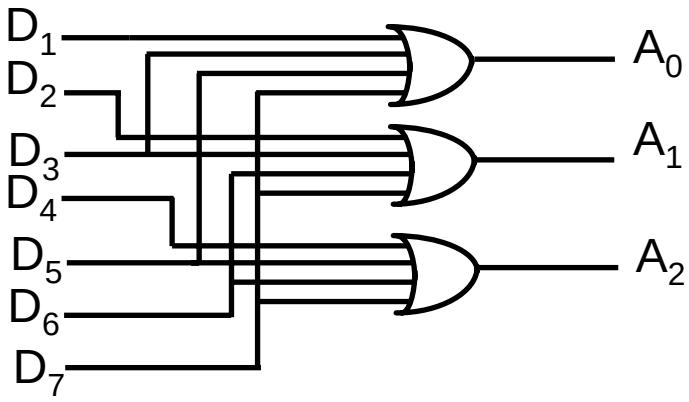
4-to-1 Multiplexer

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



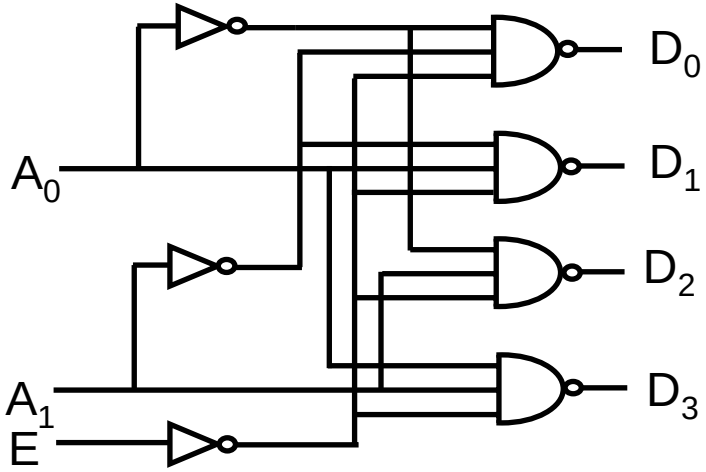
ENCODER/DECODER

Octal-to-Binary Encoder



2-to-4 Decoder

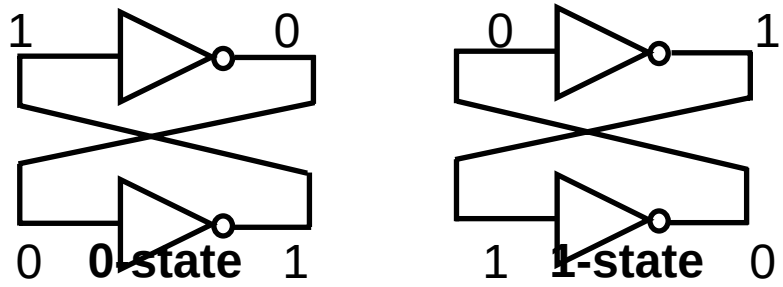
E	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	d	d	1	1	1	1



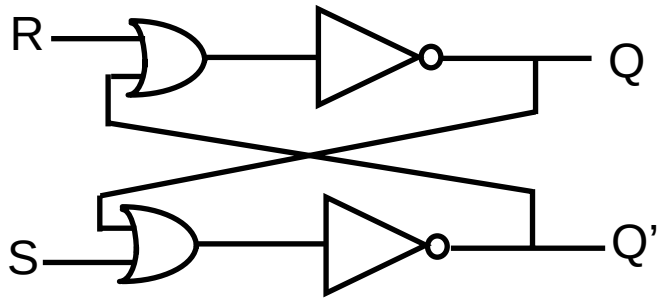
FLIP FLOPS

Characteristics

- **2 stable states**
- **Memory capability**
- **Operation is specified by a Characteristic Table**



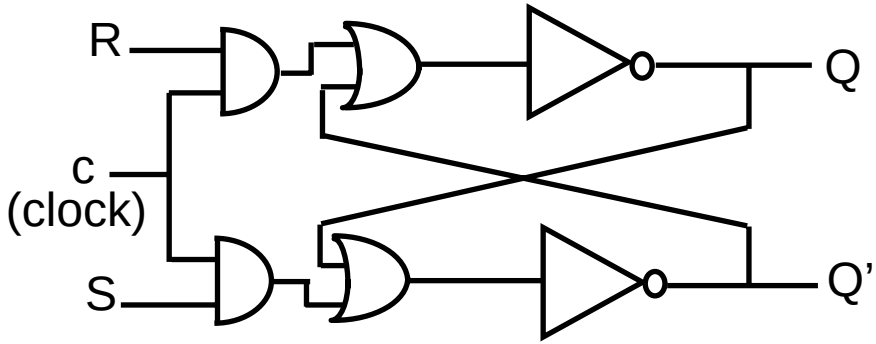
In order to be used in the computer circuits, state of the flip flop should have input terminals and output terminals so that it can be set to a certain state, and its state can be read externally.



S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	indeterminate (forbidden)

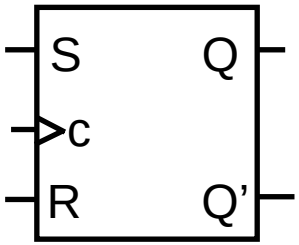
CLOCKED FLIP FLOPS

In a large digital system with many flip flops, operations of individual flip flops are required to be synchronized to a clock pulse. Otherwise, the operations of the system may be unpredictable.

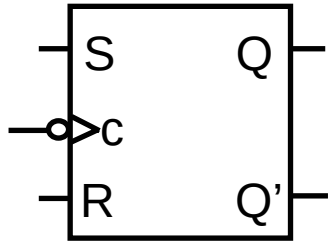


Clock pulse allows the flip flop to change state only when there is a clock pulse appearing at the c terminal.

We call above flip flop a Clocked RS Latch, and symbolically as

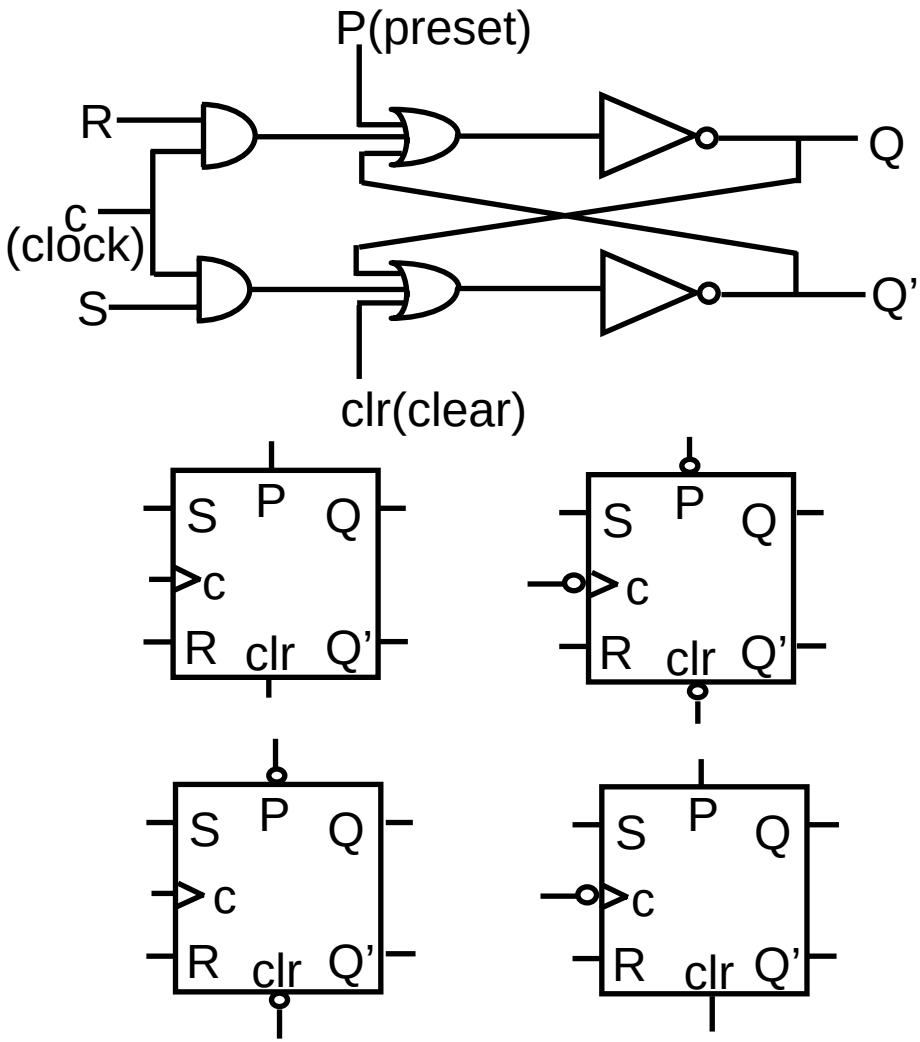


operates when clock is high



operates when clock is low

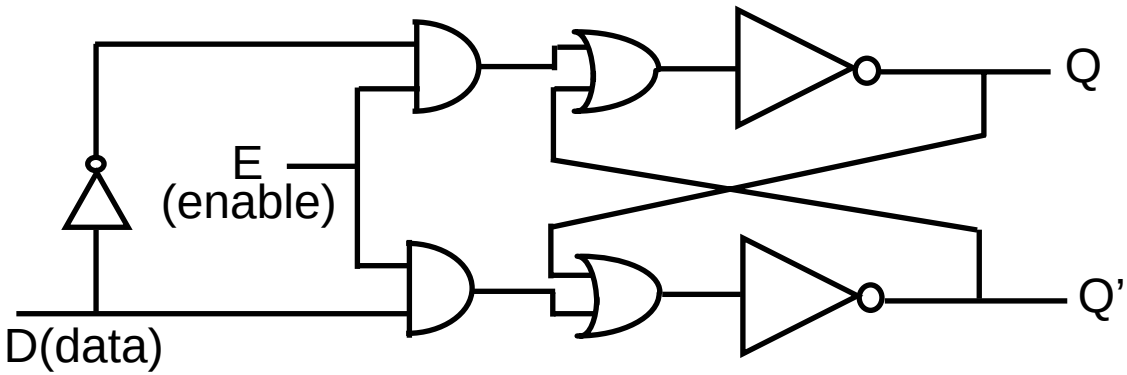
RS-LATCH WITH PRESET AND CLEAR INPUTS

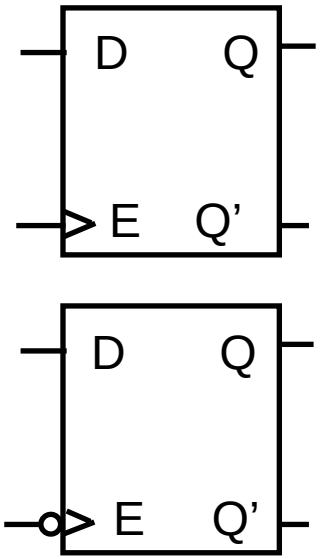


D-LATCH

D-Latch

Forbidden input values are forced not to occur
by using an inverter between the inputs





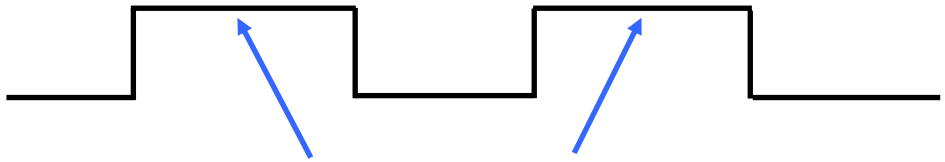
D	Q(t+1)
0	0
1	1

EDGE-TRIGGERED FLIP FLOPS

Characteristics

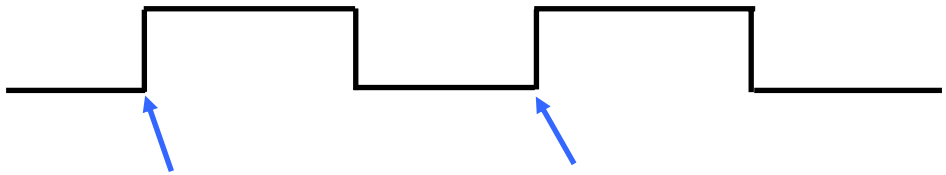
- State transition occurs at the rising edge or falling edge of the clock pulse

Latches



respond to the input only during these periods

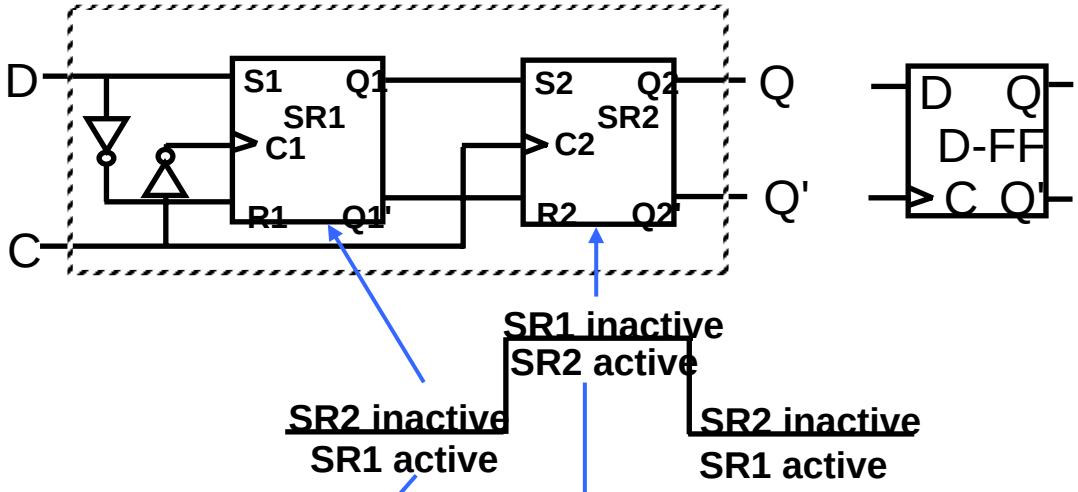
Edge-triggered Flip Flops (positive)



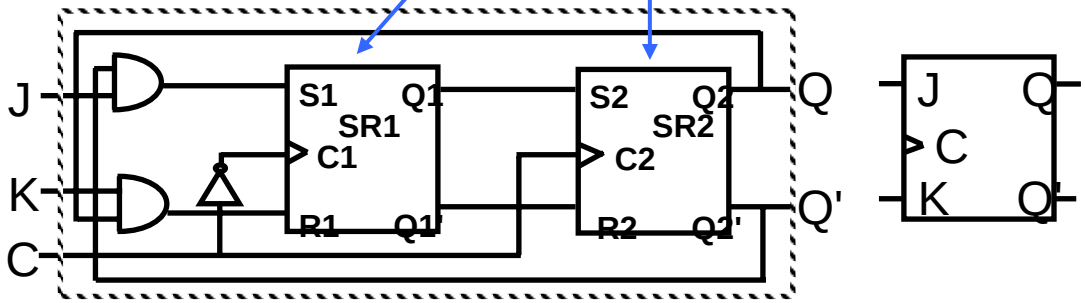
respond to the input only at this time

POSITIVE EDGE-TRIGGERED

D-Flip Flop



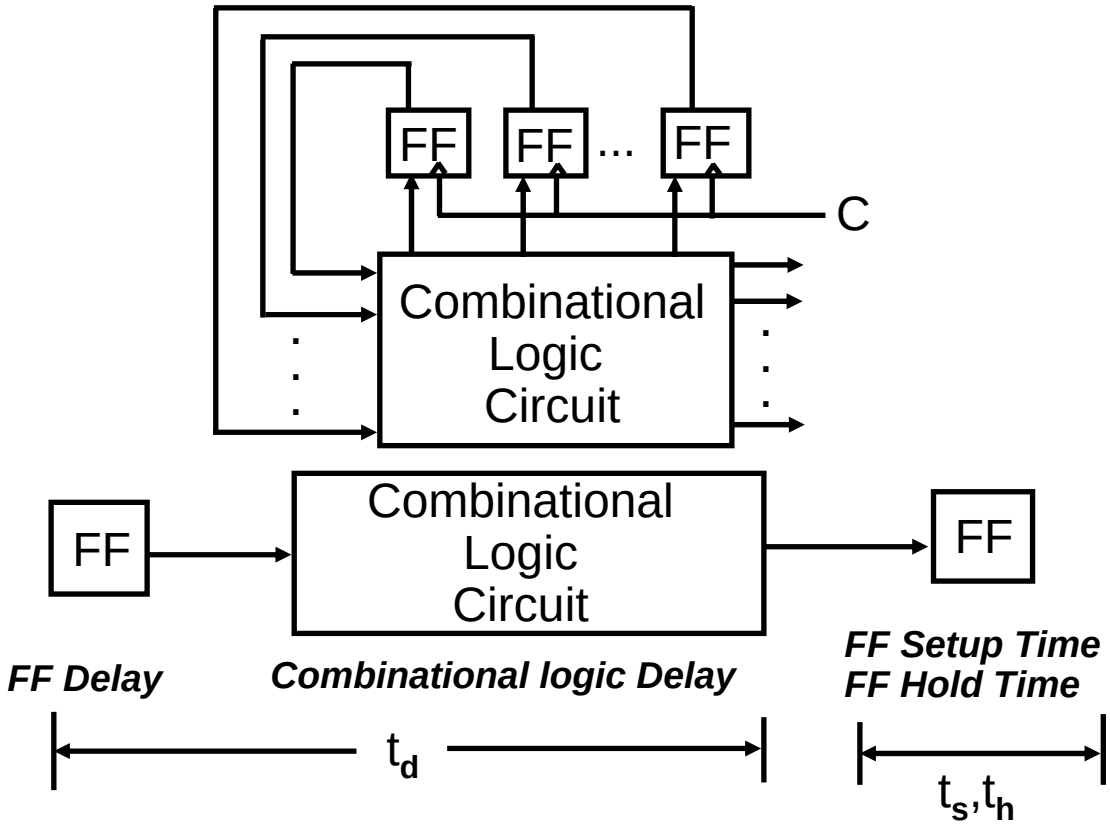
JK-Flip Flop



T-Flip Flop: JK-Flip Flop whose J and K inputs are tied together to make T input. Toggles whenever there is a pulse on T input.

CLOCK PERIOD

Clock period determines how fast the digital circuit operates.
How can we determine the clock period ?
Usually, digital circuits are sequential circuits which has some flip flops



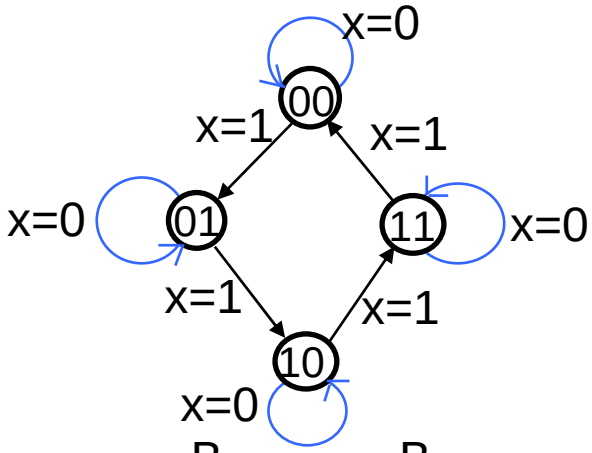
clock period $T = t_d + t_s + t_h$

DESIGN EXAMPLE

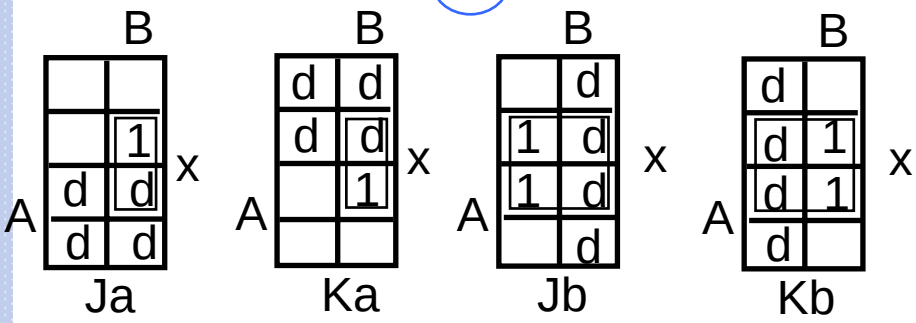
Design Procedure:

Specification ⇒ State Diagram ⇒ State Table ⇒ Excitation Table ⇒ Karnaugh Map ⇒ Circuit Diagram

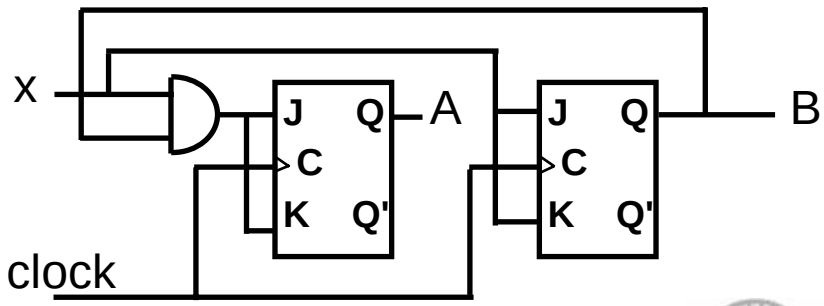
Example: 2-bit Counter -> 2 FF's



current state		input	next state		FF inputs			
A	B	x	A	B	Ja	Ka	Jb	Kb
0	0	0	0	0	0	d	0	d
0	0	1	0	1	0	d	1	d
0	1	0	0	1	0	d	d	0
0	1	1	1	0	1	d	d	1
1	0	0	1	0	d	0	0	d
1	0	1	1	1	d	0	1	d
1	1	0	1	1	d	0	d	0
1	1	1	0	0	d	1	d	1

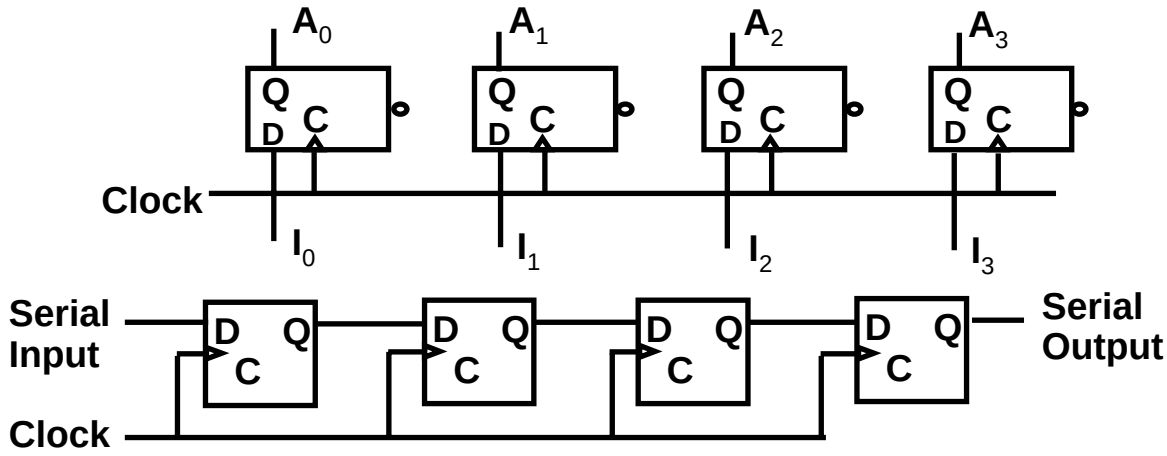


Ja = Bx Ka = Bx Jb = x Kb = x

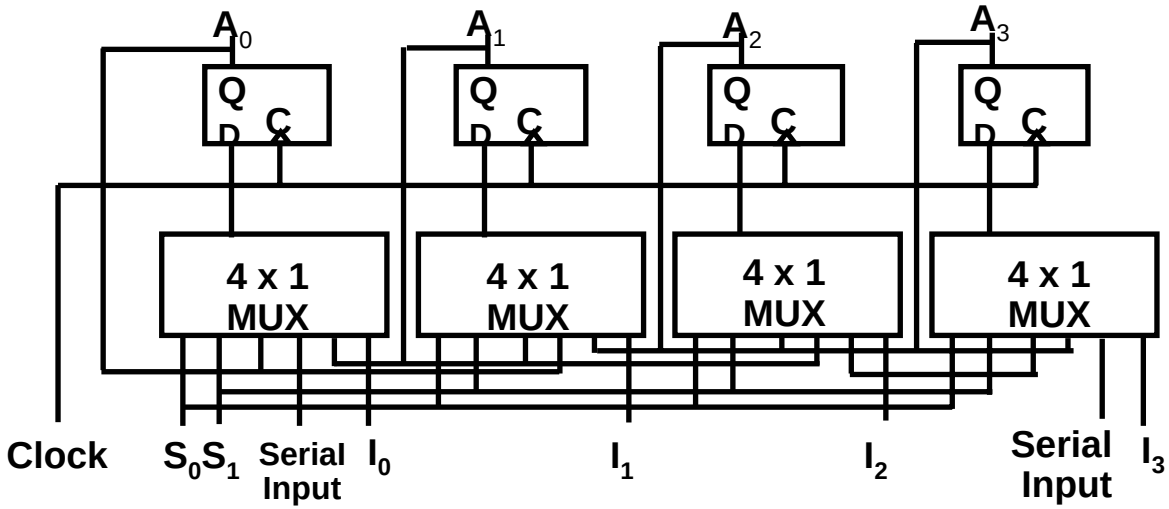


SEQUENTIAL CIRCUITS - Registers

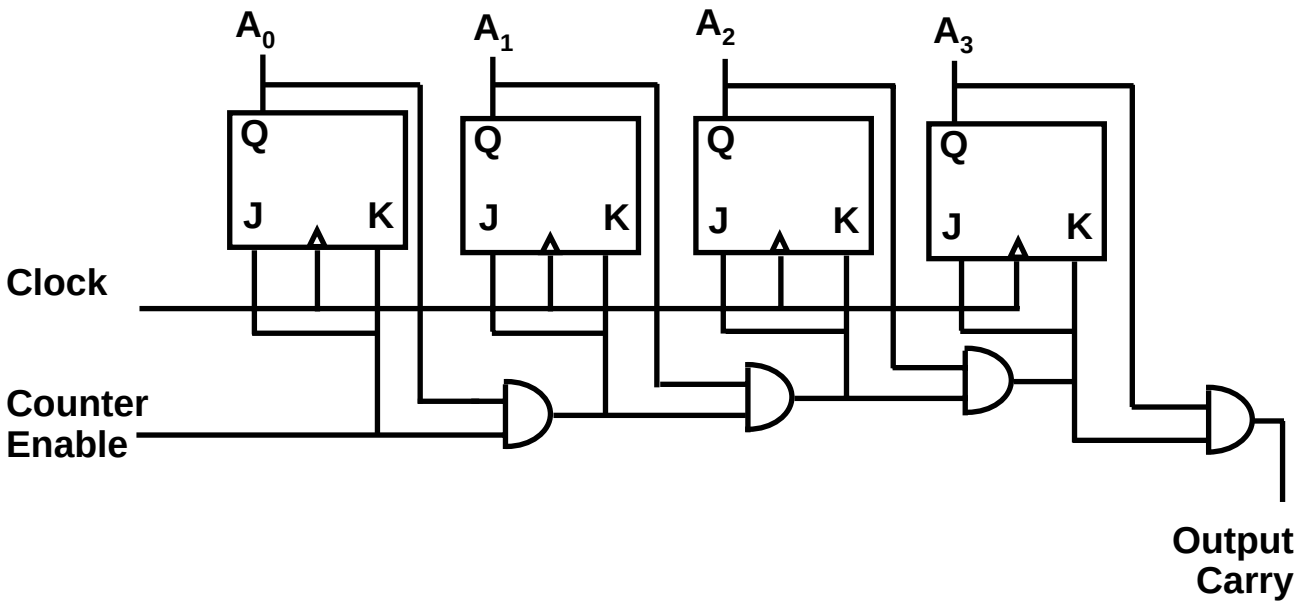
Shift Registers



Bidirectional Shift Register with Parallel Load

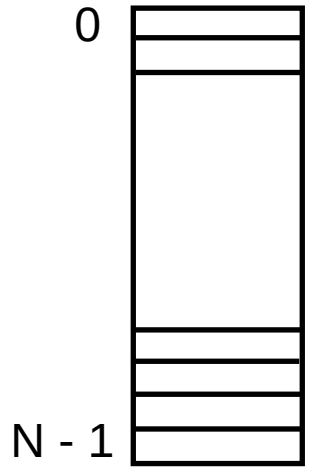


SEQUENTIUAL CIRCUITS - Counters



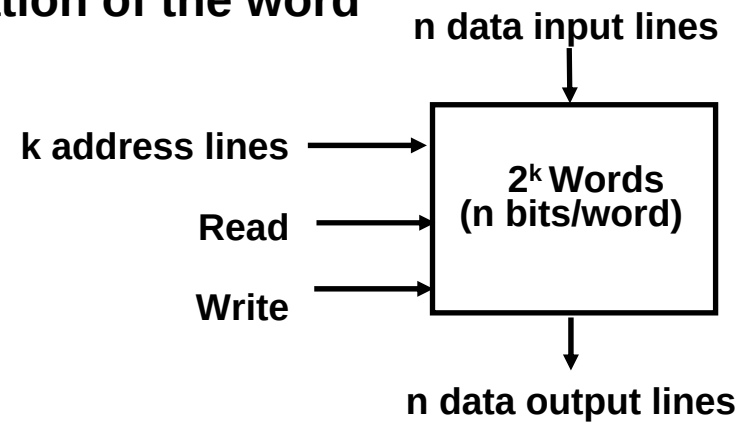
MEMORY COMPONENTS

- **Logical Organization**



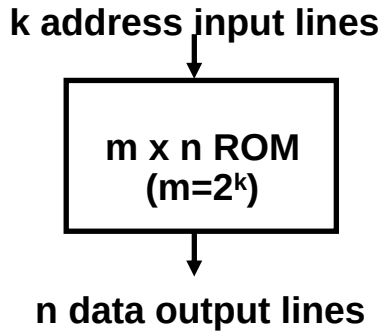
- **Random Access Memory**

- Each word has a unique address
- Access to a word requires the same time independent of the location of the word
- Organization



READ ONLY MEMORY(ROM)

❖ 특징
- 오직 읽기 연산에만
- 롬에 있는 정보는 생산되는 동안에 기록
- 구성도==========>>



출력 정보는 오직 입력 라인의 정보에 달려 있다.

-> 조합 논리 회로

$X_0=A'B' + B'C$
$X_1=A'B'C + A'BC'$
$X_2=BC + AB'C'$
$X_3=A'BC' + AB'$
$X_4=AB$

Canonical minterms

$X_0=A'B'C' + A'B'C + AB'C$
$X_1=A'B'C + A'BC'$
$X_2=A'BC + AB'C' + ABC$
$X_3=A'BC' + AB'C' + AB'C$
$X_4=ABC' + ABC$

address ABC	*Output* X_0	X_1	X_2	X_3	X_4
000	1	0	0	0	0
001	1	1	0	0	0
010	0	1	0	1	0
011	0	0	1	0	0
100	0	0	1	1	0
101	1	0	0	1	0
110	0	0	0	0	1
111	0	0	1	0	1

TYPES OF ROM

❖ **ROM**

- 생산되는 동안에 정보 입력
- 생산 공정에서 MASK Pattern 이용
- 변경 불가
- 적은 비용으로 대량생산에 적합. --> 마지막 공정에 이루어진다.

❖ **PROM (Programmable ROM)**

- 전기를 이용하여 PROM 에 프로그래밍하여 저장
- 변경불가
- 많은 비용이 든다. -> 개발 단계 시스템에서 사용 .
  -> 적은 용량의 시스템에 사용할수 있다.

❖ **EPROM (Erasable PROM)**

- 사용자가 전기를 사용하여 프로그램을 저장
- 저장내용을 지울수 있다. 자외선을 이용하여 내용을 지우고 바꿀수 있다.
- 재사용 할수 있으나, 많은 비용이 든다. → 개발 단계시스템에서 사용된다.
  → 생산단계에서는 사용되지 않는다.

INTEGRATED CIRCUITS

❖ 집적도에 따른 분류

· SSI - 소규모 집적(small-scale integration)
- 10개 이하의 독립적인 게이트가 하나의 칩에 포함
- 게이트의 입출력이 바로 외부 핀과 연결

· MSI - 중규모 집적(medium-scale integration)
- 10에서 200개까지의 게이트를 집적
- 기본적 디지탈 장치를 구현: 디코더, 가산기, 레지스터

· LSI - 대규모 집적(large-scale integration)
- 200에서 1000개까지의 게이트를 집적
- 디지탈 시스템을 형성: 프로세서, 메모리

· VLSI - 초대규모 집적(very-large-scale integration)
- 수천 개의 게이트를 하나의 칩에 집적
- 대형 메모리, 마이크로 컴퓨터 칩 형성

INTEGRATED CIRCUITS

❖ 구현하는 데 적용된 기술에 따른 분류

· TTL(transistor-transistor logic)
- 가장 많이 사용되고 있는 논리군
- 표준 TTL 외에 고속 TTL, 저전력 TTL, 고성능 쇼트키(Schottky) TTL
- 전원은 5v, Level은 0v와 3.5v

· ECL(emitter-coupled logic)
- 게이트의 트랜지스터는 불포화 상태에서 동작
- 고속도가 요구되는 시스템에 사용
- 전달 지연 시간(Propagation Delay Time)은 1~2 nano초

· MOS(metal-oxide semiconductor)
- 단상 트랜지스터 사용
- 대부분 NMOS

· CMOS(complementary MOS)
- NMOS와 CMOS를 연결하여 구성
- 회로의 밀도가 높고, 제조공정이 단순, 전력 소비가 적다