VHDL - INTRO

2015.07.15

강의자 : **23**기 백두현

BARAM
KW univ
New generation of Robotics

목차

- INTRO
- GATE
- DECODER/ENCODER
- MUX/DEMUX
- FlipFlop/Latch

FPGA

- **FPGA(Field-Programmable Gate Array)**
  - 재프로그래밍 가능한 실리콘 칩.
  - 미리 구축된 로직 블록 및 프로그래밍 가능한 라우팅 리소스가 있어 납땜 작업 없이 맞춤 하드웨어 기능 실행을 위한 칩을 구성할 수 있음.

- **FPGA를 채택하는 이유**
  - FPGA가 ASIC 및 프로세서 기반 시스템에서 최상의 부분을 통합 하기 때문.
  - FPGA는 HW 타이밍된 속도와 안전성을 제공.
  - 맞춤 ASIC 디자인의 막대한 선행 비용에 견줄만한 비용을 요구 하지 않음.

FPGA 제작 대표회사

- **Altera**
  - Logic Element(LUT,FF) 와 OnChipMemory로 구성
    - LUT : GATE에 대한 Truth Table을 구현.

    => 모든 GATE구현 가능.

    => 하나의 LUT로 다수의 GATE구현가능
  - Chip가격이 비싸지만 교육용 Contents들이 많음.

- **Xilinx**
  - GATE와 FF 로 구성.
    - Chip가격이 싸고 IDE Tool, EDA Tool 이 Altera에 반해 강력함.

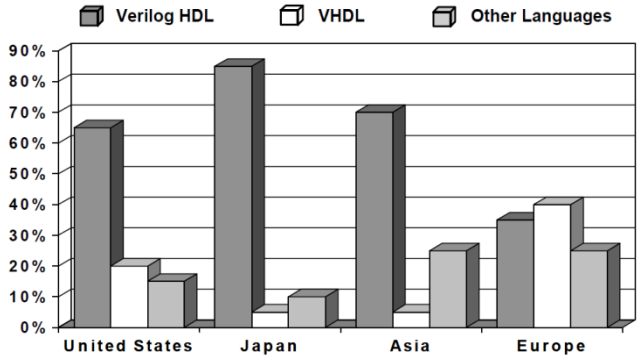
직접회로설계의 변천 과정

- **설계방법**
  - 70년대 : Transistor Level의 Layout 설계
  - 80년대 : Gate 또는 RTL Level의 회로 설계
  - 90년대 : Algorithm이나 기능 레벨의 설계

- **설계도구**
  - 70년대 : Layout Editor
  - 80년대 : Schematic Editor
  - 90년대 : HDL과 Synthesis Toll(Silicon Compiler)

HDL

- **VHDL(VHSIC Hardware Description Language)**
  - 학교나 연구실에서 학술적 용도로 사용.
  - VHDL 반복적인 기술을 생성할 수 있음.
  - Mypackage
    - Memory 관리 및 설계 편의 제공.



- **VerilogHDL**
  - 대기업에서 많이 사용.
  - 2차원 register 사용할 때 요소 접근이 불편함.
  - 복잡한 타이밍 신호를 기술 할 수 있음.

- **SoC 개발자라면 두 언어 모두 다룰 수 있어야 함.**

SoC 개발 Process

- **Specification 규정.**
  - System의 input과 output을 규정.
  - System의 동작 및 Limit를 규정.
- **Behavior Level 설계.**
  - Truth Table(K-MAP)
  - State Machine Diagram
  - Functional Simulation
  - Algorithm 검증
- **RTL Level 설계**
- **GATE Level 설계**
- **Layout 설계**

FPGA 구현 가능 단계

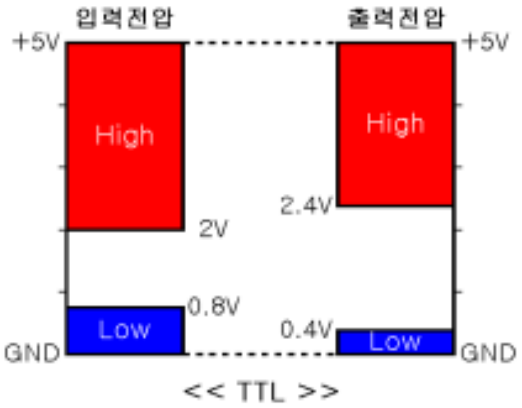
디지털이란?

- 디지털 회로란? 디지털 논리 회로의 조합

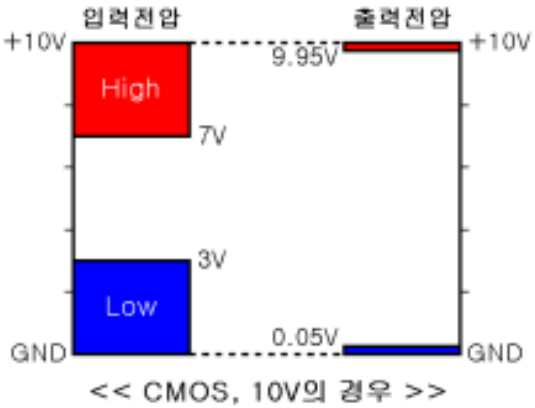
- 디지털은 일반적으로 2개 상태를 취하여, 각각 'Low'와 'High', 또는 'L' 와 'H'라고 표현된다. 일반적으로는 Low를 0으로, High를 1로 대응시킨다. (1과 0의 상태는 각각 전류가 흐른다/ 흐르지 않는다 의 상태를 의미)

디지털이란?

- 초기의 논리회로는 5V 기준이었다. 5V를 정해 놓고 이것에 맞추어 논리게이트나 마이크로 프로세서가 만들어졌다. 그래서 5V를 Vcc라고 표시한다.



<< TTL >>



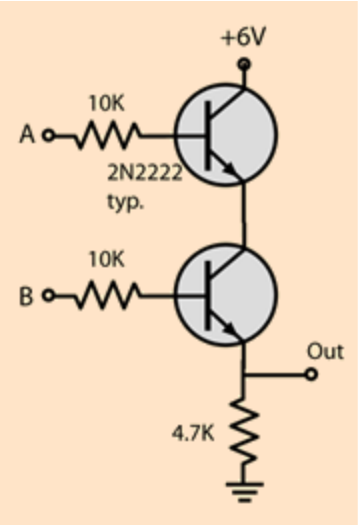
<< CMOS, 10V의 경우 >>

출처 : http://cpu.kongju.ac.kr

디지털 회로의 기본단위

- 디지털 회로의 기본단위 = gate

EX – AND gate



AND gate		
입력		출력
0	0	0
0	1	0
1	0	0
1	1	1

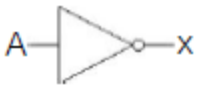
GATE

Buffer

x=A

A	x
0	0
1	1

Inverter

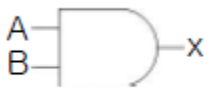


x=A'

A	x
0	1
1	0

GATE

AND



$x=A \cdot B$

$x=AB$

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

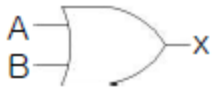
NAND

$x=(AB)'$

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

GATE

OR



x=A+B

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

NOR

x=(A+B)'

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

GATE

XOR(Exclusive – OR , Odd)

$x = A \oplus B$

$x = A \oplus B \oplus C$

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

A	B	C	X
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	1

XNOR(Exclusive – NOR, EVEN)

$x = (A \oplus B)'$

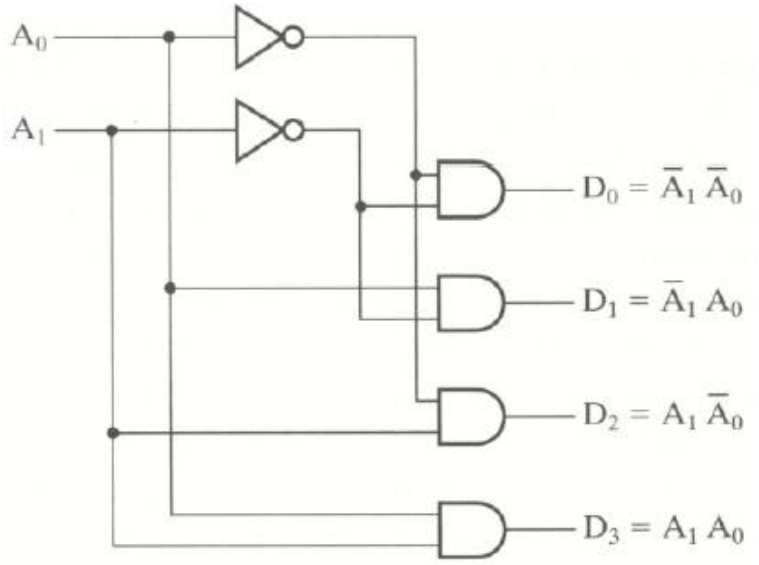
$x = (A \oplus B \oplus C)'$

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

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

DECODER/ENCODER

DECODER



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

- n-bit 입력 코드를 m-bit 출력 코드로 변환하는 조합논리회로.(단! n<= m <=2^n)
- 유효한 입력 코드에 대해서 유일한 코드가 출력 되어야 한다.

DECODER/ENCODER

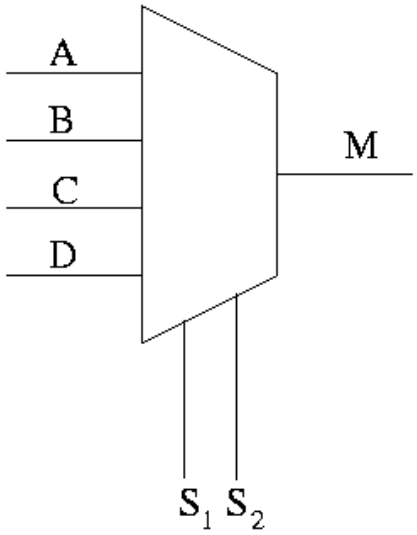
ENCODER

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

- DECODER의 역기능.
- 2^n혹은 이하의 입력 라인을 갖고 n개의 출력 라인을 갖는다.

MUX/DEMUX

MUX(Multiplexer)

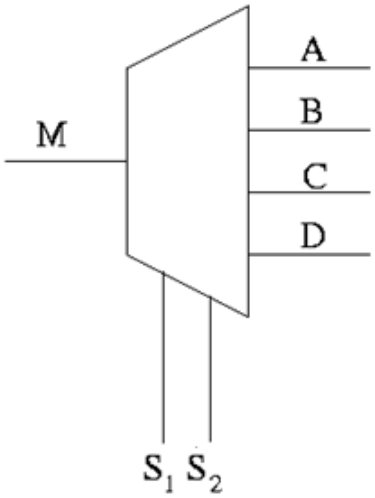


S1	S2	M
0	0	A
0	1	B
1	0	C
1	1	D

- MUX는 많은 입력 라인 중 하나를 선택하고 이를 단독 출력 라인으로 내보내는 조합회로.

MUX/DEMUX

DEMUX(Demultiplexer)



S1	S2	A	B	C	D
0	0	M	X	X	X
0	1	X	M	X	X
1	0	X	X	M	X
1	1	X	X	X	M

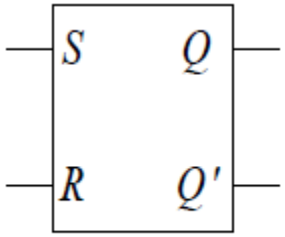
- mux의 역을 수행하는 회로, 데이터 분배기.
- 하나의 입력을 여러 개의 출력 라인 중 선택하여 출력.

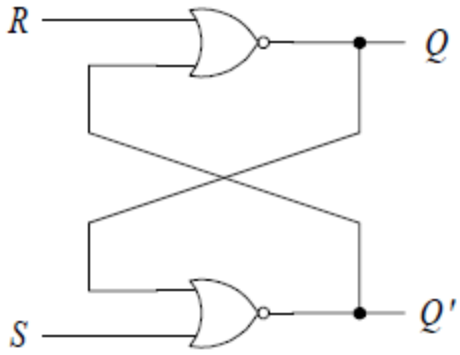
FlipFlop/Latch

- **Latch와 FlipFlop**
  - 두개의 안정 상태를 갖는 일종의 기억 회로.
  - Latch와 FlipFlop은 CLK의 유무에 따라 구별된다.
    - Latch는 비동기식으로 CLK를 사용하지 않음.
    - FlipFlop은 동기식으로 CLK를 사용함.

FlipFlop/Latch

- **SR Latch(using NOR)**

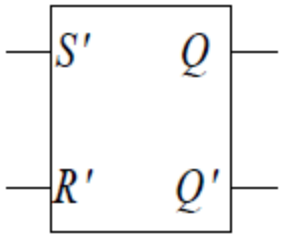




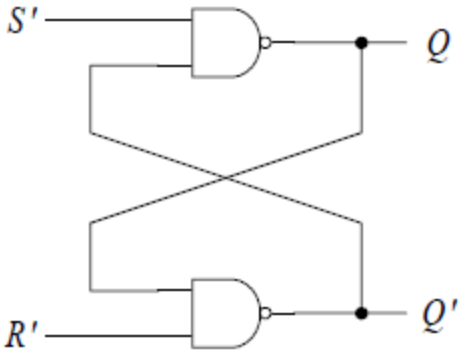
S	R	Q	Q_{next}	Q_{next}'
0	0	0	0	1
0	0	1	1	0
0	1	×	0	1
1	0	×	1	0
1	1	×	0	0

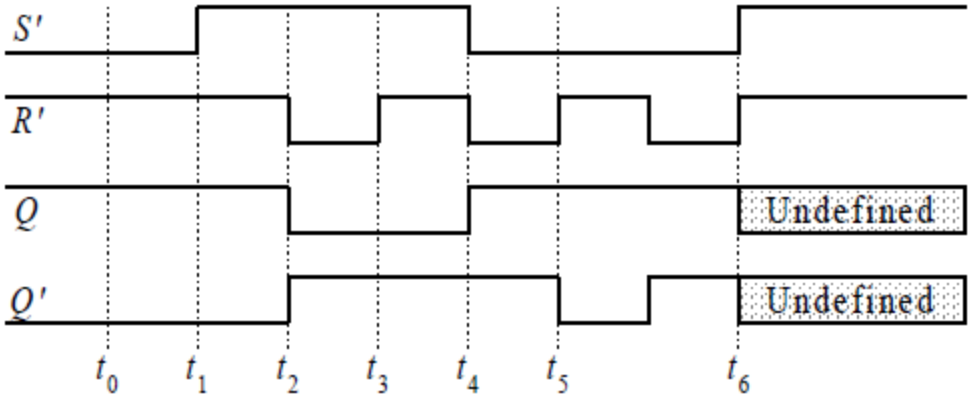
FlipFlop/Latch

- **SR Latch(using NAND)**



S	R	Q	Q_{next}	Q_{next}'
0	0	×	1	1
0	1	×	1	0
1	0	×	0	1
1	1	0	0	1
1	1	1	1	0

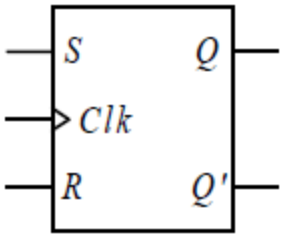




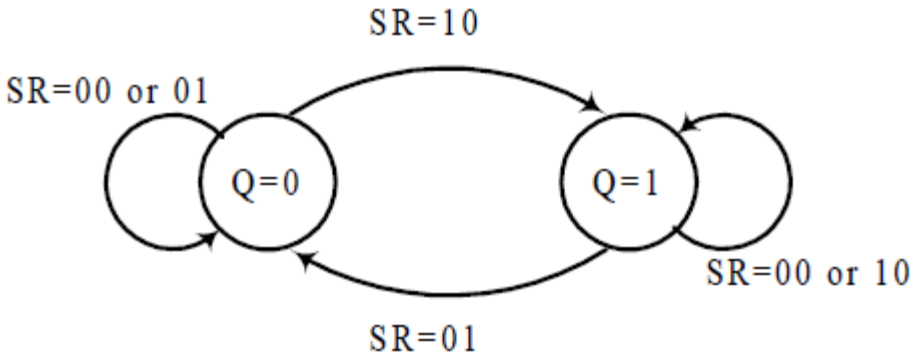
FlipFlop/Latch

- **SR FlipFlop**

SR



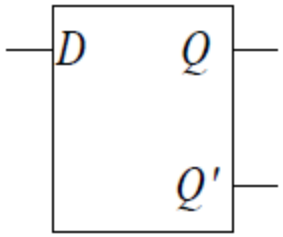
S	R	Q	Qnext
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	×
1	1	1	×



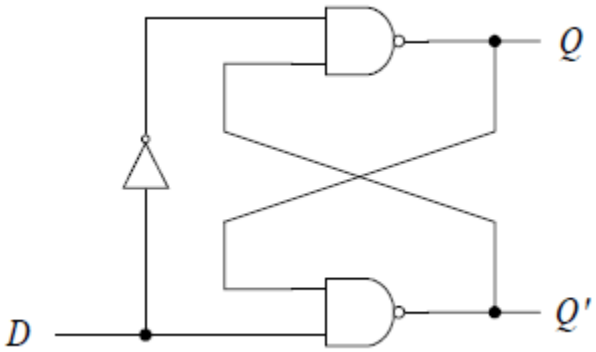
$$Q_{next} = S + R'Q$$
$$SR = 0$$

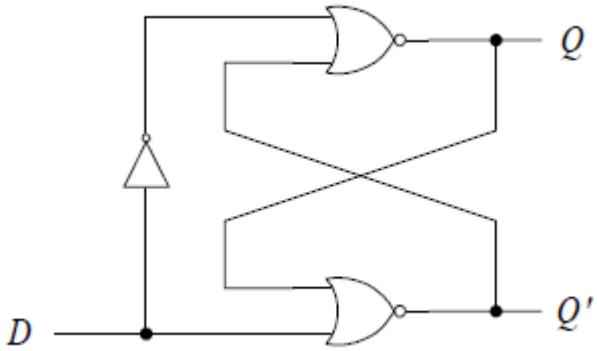
FlipFlop/Latch

- **D Latch**



D	Q	Q_{next}	Q_{next}'
0	×	0	1
1	×	1	0

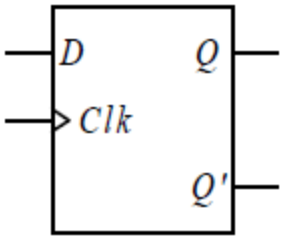




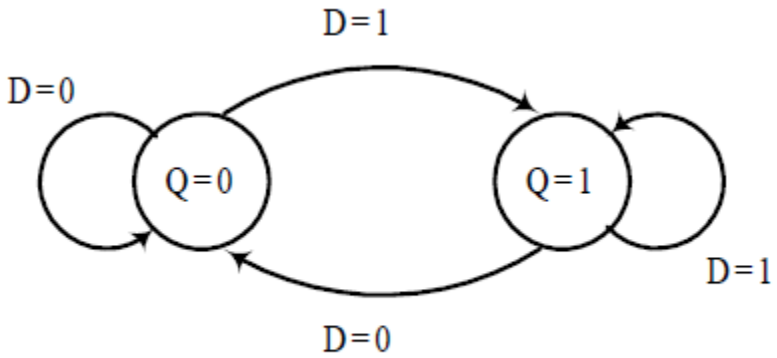
FlipFlop/Latch

- D FlipFlop

D



D	Q	Qnext
0	×	0
1	×	1

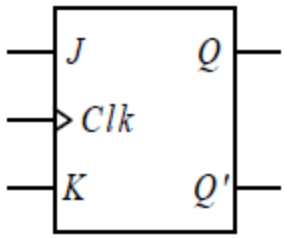


$$Q_{next} = D$$

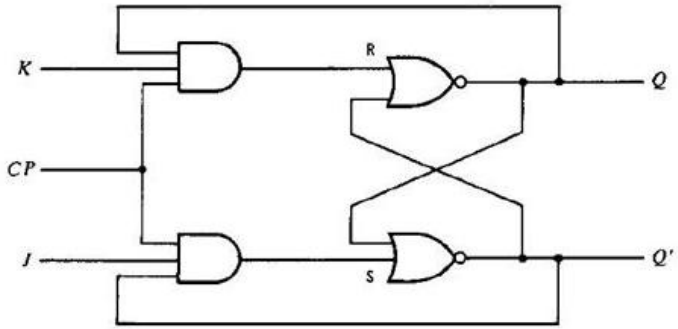
FlipFlop/Latch

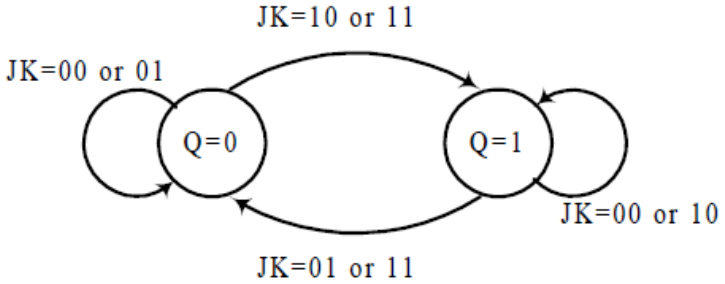
JK FlipFlop

JK



J	K	Q	Qnext
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	0

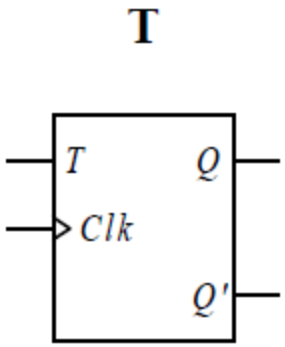


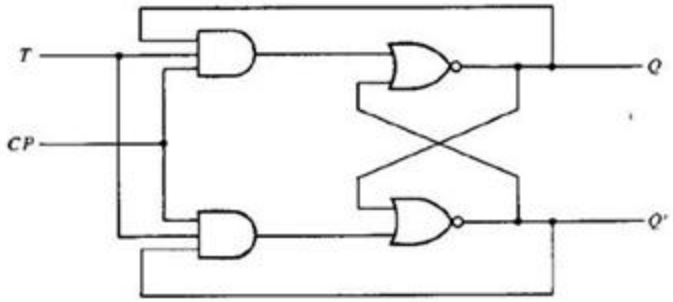


$$
\begin{aligned}
Q_{next} &= J'K'Q + JK' + JKQ' \\
&= J'K'Q + JK'Q + JK'Q' + JKQ' \\
&= K'Q(J'+J) + JQ'(K'+K) \\
&= K'Q + JQ'
\end{aligned}
$$

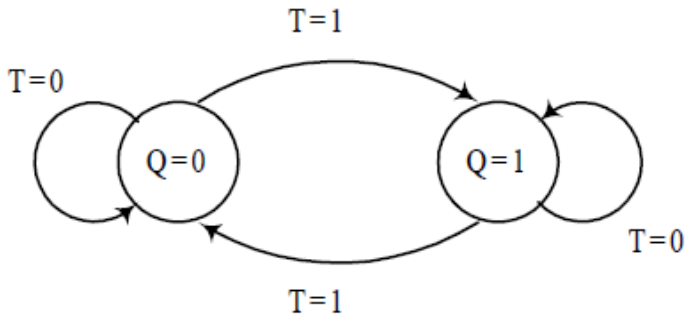
FlipFlop/Latch

T FlipFlop





T	Q	Q_{next}
0	0	0
0	1	1
1	0	1
1	1	0



$$Q_{next} = TQ' + T'Q = T \oplus Q$$