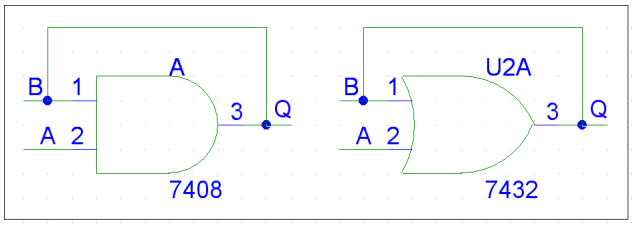
3. 순차 논리 회로 설계

- 순차논리회로는 여러 개의 F/F과 궤환경로를 가지는 조합논리회로로 구성

feedback

3.1 latch 회로 설계 : 일종의 기억 회로

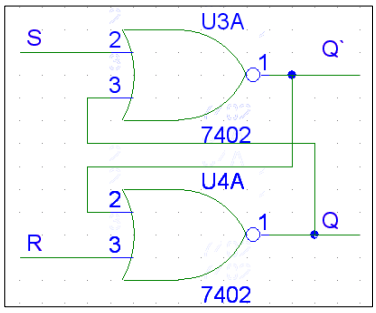
1) 기본 기억 회로

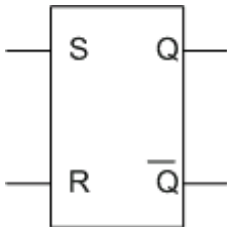


- 왼쪽회로에서 초기상태를 A=1, Q=1이라 가정하고, A를 '0'으로 바꾸면 Q는 '0'인 상태가 되고 궤환회로를 제거하지 않으면 이 상태는 계속 유지.
- 오른쪽회로에서 초기상태를 A=0, Q=0이라 가정하고, A=1로 바꾸면, Q=1인 상태로 되어 궤환회로를 제거하지 않으면 이 상태를 계속 유지.

2) RS-latch 회로 : R - reset, S - set.

(1) NOR gate로 구성





입력		출력		출력상태
R	S	Q_{t+1}	$\overline{Q_{t+1}}$	
0	0	Q_t	$\overline{Q_t}$	무변화
0	1	1	0	set
1	0	0	1	reset
1	1	0	0	사용금지

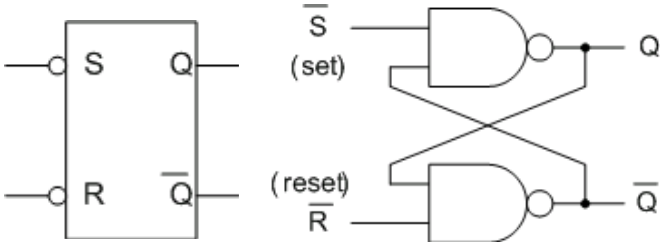
```
module NOR_LATCH(R, S, Q, Q_bar);
    input R, S;
    output Q, Q_bar;

    nor U1 (Q, R, Q_bar);
    nor U2 (Q_bar, S, Q);

endmodule
```

```
module NOR_LATCH_1(R, S, Q, Q_bar);
    input R, S;
    output Q, Q_bar;
    reg Q;
    assign Q_bar = ~Q;
    always @(R or S)
        case({R,S})
          2'b01 : Q<=1;
          2'b10 : Q<=0;
          default : Q<=Q;
        endcase
endmodule
```

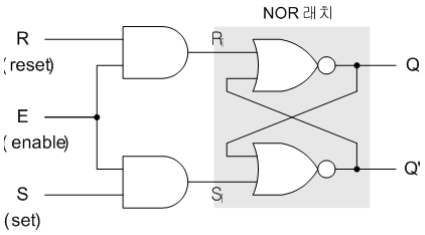
(2) NAND gate로 구성



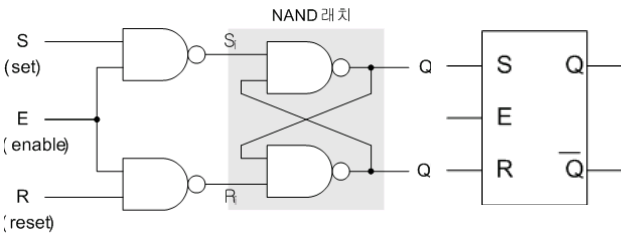
입력		출력		출력상태
$\overline{R}$	$\overline{S}$	Q_{t+1}	$\overline{Q_{t+1}}$	
0	0	1	1	사용금지
0	1	0	1	reset
1	0	1	0	set
1	1	Q_t	$\overline{Q_t}$	무변화

```
module NAND_LATCH(bR, bS, Q, Q_bar);
    input bR, bS;
    output Q, Q_bar;
    nand U1 (Q, bS, Q_bar);
    nand U2 (Q_bar, bR, Q);
endmodule
```

3) Enable 단자가 있는 RS-latch 회로



(a) NOR 래치를 이용한 회로도



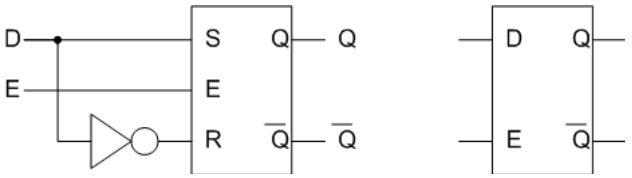
(b) NAND 래치를 이용한 회로도

입력			출력		출력상태
E	R	S	Q_{t+1}	$\overline{Q_{t+1}}$	
0	X	X	Q_t	$\overline{Q_t}$	무변화
1	0	0	Q_t	$\overline{Q_t}$	무변화
1	0	1	1	0	set
1	1	0	0	1	reset
1	1	1	0	0	사용금지

```
module NOR_LATCH_E(E,R,S,Q, Q_bar);
    input E, R, S;
    output Q, Q_bar;
     and U0 (X0, S, E);
     and U1 (X1, R,E);
     nor U2 (Q_bar, X0, Q);
     nor U3 (Q, X1, Q_bar);
endmodule
```

```
module NAND_LATCH_E(E,R,S,Q, Q_bar);
    input E, R, S;
    output Q, Q_bar;
     nand U0 (X0, R, E);
     nand U1 (X1, S, E);
     nor U2 (Q_bar, X0, Q);
     nor U3 (Q, X1, Q_bar);
endmodule
```

4) D-latch 회로 : 저장장치로서 1bit 논리의 처리 및 저장이 가능.



D	Q_t	Q_{t+1}	$\overline{Q_{t+1}}$
0	0	0	1
0	1	0	1
1	0	1	0
1	1	1	0

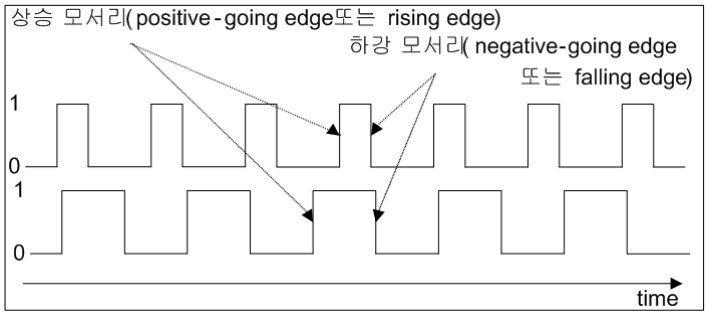
$Q_{t+1} = D$

```
module D_LATCH_E(E, D, Q, Q_bar);
    input E, D;
    output Q, Q_bar;
    reg Q, Q_bar;

    always @(D or E)
  begin
    if(E)
      begin
        Q<=D;
        Q_bar<=~D;
      end
  end
endmodule
```

3.2 플립플롭(flip-flop) 설계

- clock : 일반적으로 그림과 같이 시간에 따라 0과 1 값을 주기적으로 반복하여 갖는 신호.



always 문의 이벤트 신호
-상승 에지
always @(posedge 클럭 이름)
-하강 에지
always @(negedge 클럭 이름)

- F/F : clock 신호에 의해 입력 신호에 의한 출력을 얻을 수 있는 회로.
clock이 인가되기 전에는 이전 상태를 그대로 유지하는 기억 회로.
- F/F 사용 시 주의할 점

① 클럭 검출의 에지에 벡터의 인덱스는 사용불가.

```
always @ (posedge CLK[1])  // error
```

② set/reset 조건에는 벡터의 비트는 사용불가.

```
always @ (posedge CLK)
if(RESET[1])  // error
...
```

③ set/reset 조건에 복잡한 수식 사용불가.

```
always @ (posedge CLK)
```

```
    if(RESET == (2-1))   // error
    ...
```

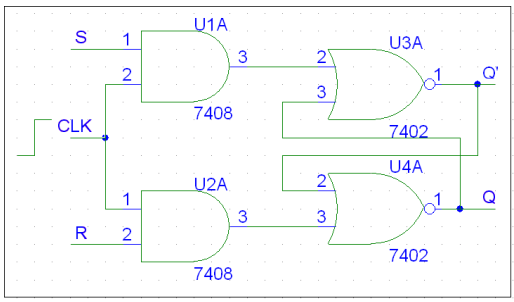
④ ‘if’문은 ‘always’문 블록의 처음에 있어야만 함.

```
    always @ (posedge CLK)
    ...                          // error
    if(RESET)
    ...
```

1) RS-F/F

- RS latch : 입력의 변화에 의해 출력이 결정 => 비동기식 회로.
- RS-F/F : 입력이 변하더라도 CLK이 인가되지 않으면 출력의 변화가 없는 동기식 회로. (반드시 CLK이 인가되어야만 출력이 변화.)

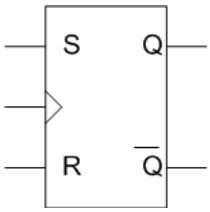
① RS-F/F



CLK	R	S	Q_{t+1}	S	R	Q_t	Q_{t+1}
↑	0	0	Q_t	0	0	0	0
↑	0	1	1	0	0	1	1
↑	1	0	0	0	1	0	0
↑	1	1	불허	0	1	1	0
↓	x	x	Q_t	1	0	0	1
				1	0	1	1
				1	1	0	x
				1	1	1	x

S \ $R\,Q_t$	00	01	11	10
0	0	1	0	0
1	1	1	x	x

$Q_{t+1} = S + \overline{R}Q_t$




```
module RS_FF_1(CLK, R, S, Q, Q_bar);
  input CLK, R, S;
  output Q, Q_bar;
  reg Q, Q_bar;
   always @(posedge CLK)
   case({R,S})
     2'b01 : begin Q<=1; Q_bar<=0; end
     2'b10 : begin Q<=0; Q_bar<=1; end
   default : begin Q<=Q; Q_bar<=Q_bar; end
   endcase
endmodule
```

```
module RS_FF_2(CLK, R, S, Q, Q_bar);
   input CLK, R, S;
   output Q, Q_bar;
    reg Q, Q_bar;
      always @(posedge CLK)
      begin
       Q     <=   S|(~R & Q);
       Q_bar <= ~(S|(~R & Q));
      end
endmodule
```

```
module RS_FF(CLK, R, S, Q, Q_bar);
    input R, S, CLK;
    output Q, Q_bar;
    reg Q, Q_bar;
    always @(posedge CLK)
     casex({R,S})
      2'b00 : begin Q<=Q; Q_bar = ~Q; end
      2'b01 : begin Q<=1'b1; Q_bar = 1'b0; end
      2'b10 : begin Q<=1'b0; Q_bar = 1'b1; end
      2'b11 : begin Q<=1'bx; Q_bar = 1'bx; end
     endcase
 endmodule
```

② Preset 단자와 clear 단자를 가진 RS-F/F

- preset : 입력에 관계없이 출력이 '1'이 되도록 함.
- clear : 입력에 관계없이 출력이 초기화('0')가 되도록 함.

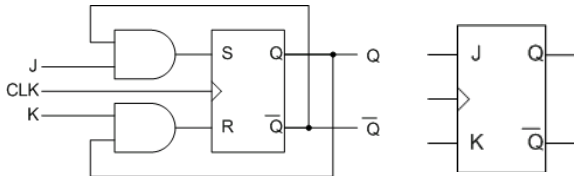
```
module RS_FF_RE_CL(CLK, PRESET, CLR, R, S, Q, Q_bar);
    input CLK, R, S, PRESET, CLR;
    output Q, Q_bar;
    reg Q, Q_bar;
     always @(posedge CLK or posedge PRESET or posedge CLR)
       if(PRESET) begin Q<=1; Q_bar<=0; end
       else if(CLR) begin Q<=0; Q_bar <=1; end
       else  begin
        Q <= S|(~R & Q);
        Q_bar <= ~(S|(~R & Q));
       end
endmodule
```

```
module RS_FF_RE_CL_1(CLK, PRESET, CLR, R, S, Q, Q_bar);
    input CLK, R, S, PRESET, CLR;
    output Q, Q_bar;
    reg Q, Q_bar;
     always @(posedge CLK)
       if(PRESET) begin Q<=1; Q_bar<=0; end
       else if(CLR) begin Q<=0; Q_bar <=1; end
       else  begin
        Q <= S|(~R & Q);
        Q_bar <= ~(S|(~R & Q));
       end
endmodule
```

3) JK-F/F

- RS-F/F의 사용금지부분(R=S=1)이 토글로 동작하도록 개선한 F/F.

① JK-F/F



J	K	Q_{t+1}	상태	J	K	Q_t	Q_{t+1}
0	0	Q_t	무변화	0	0	0	0
0	1	0	reset	0	0	1	1
1	0	1	set	0	1	0	0
1	1	$\overline{Q_t}$	토글	0	1	1	0
				1	0	0	1
				1	0	1	1
				1	1	0	1
				1	1	1	0

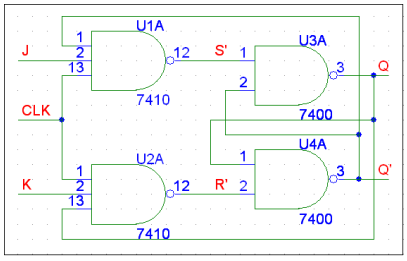
J \ $K\ Q_t$	00	01	11	10
0	0	1	0	0
1	1	1	0	1

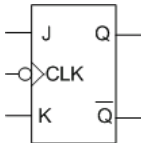
$$Q_{t+1} = J\overline{Q_t} + \overline{K}Q_t$$

```
module JK_FF(CLK, J, K, Q, Q_bar);
    input CLK, J, K;
    output Q, Q_bar;
    reg Q, Q_bar;
     always @(posedge CLK)
        case({J,K})
         2'b00 : begin Q<=Q; Q_bar<=Q_bar; end
         2'b01 : begin Q<=0; Q_bar<=1; end
         2'b10 : begin Q<=1; Q_bar<=0; end
         2'b11 : begin Q<=~Q; Q_bar<=~Q_bar; end
        endcase
endmodule
```

```
module JK_FF_1(CLK, J, K, Q, Q_bar);
    input CLK, J, K;
    output Q, Q_bar;
    reg Q, Q_bar;
     always @(posedge CLK)
        begin
        Q       <=  (J & ~Q)|(~K & Q);
        Q_bar <=~((J & ~Q)|(~K & Q));
        end
endmodule
```

② NAND gate를 이용한 JK-F/F






```
module JK_FF_2 (CLK, J, K, Q, Q_bar);
    input CLK, J, K;
    output Q, Q_bar;
    reg Q, Q_bar;
     always @(negedge CLK)
        case({J,K})
         2'b00 : begin Q<=Q; Q_bar<=Q_bar; end
         2'b01 : begin Q<=0;  Q_bar<=1;      end
         2'b10 : begin Q<=1;  Q_bar<=0;      end
         2'b11 : begin Q<=~Q; Q_bar<=~Q_bar; end
        endcase
endmodule
```

③ Preset 단자와 clear 단자를 가진 JK-F/F

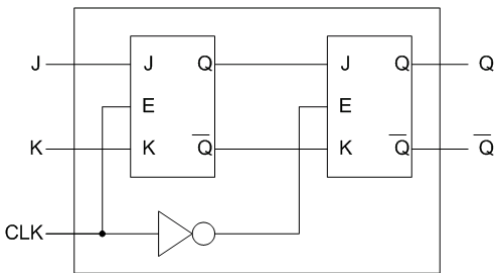
```
module JK_FF_PR_CL(PR, CLR, CLK, J, K, Q, Q_bar);    // 비동기식
    input PR, CLR, CLK, J, K;
    output Q, Q_bar;
```

```
    reg Q, Q_bar;
     always @(posedge CLK or negedge PR or negedge CLR)
       if(!PR)         begin Q<=1; Q_bar<=0; end
       else if(!CLR) begin Q<=0; Q_bar<=1; end
       else
        case({J,K})
        2'b00 : begin Q<=Q; Q_bar<=Q_bar; end
        2'b01 : begin Q<=0; Q_bar<=1; end
        2'b10 : begin Q<=1; Q_bar<=0; end
        2'b11 : begin Q<=~Q; Q_bar<=~Q_bar; end
       endcase
endmodule
```

```
module JK_FF_PR_CL_1(PR, CLR, CLK, J, K, Q, Q_bar);    // 동기식
    input PR, CLR, CLK, J, K;
    output Q, Q_bar;
    reg Q, Q_bar;
     always @(posedge CLK)
       if(!PR)         begin Q<=1; Q_bar<=0; end
       else if(!CLR) begin Q<=0; Q_bar<=1; end
       else
        case({J,K})
        2'b00 : begin Q<=Q; Q_bar<=Q_bar; end
        2'b01 : begin Q<=0; Q_bar<=1; end
        2'b10 : begin Q<=1; Q_bar<=0; end
        2'b11 : begin Q<=~Q; Q_bar<=~Q_bar; end
       endcase
endmodule
```

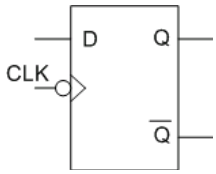
4) master-slave JK-F/F

- JK-F/F은 피드백으로 인해 입력을 변화시킬 수 있고 변화된 입력에 의해 또다시 출력이 변화하는 문제가 있음. 해결책 : master-slave JK-F/F
- 회로는 입력단의 F/F은 클럭의 상승에지에서 동작하고 슬레이브는 하강에지에서 동작하도록 구성.




```
module JK_MS_FF(CLK, J, K, Q0, Q_bar0, Q, Q_bar);
    input CLK, J, K;
    output Q0, Q_bar0, Q, Q_bar;
    reg Q0, Q_bar0, Q, Q_bar;
     always @(posedge CLK)
       case({J,K})
        2'b01 : begin Q0<=0;    Q_bar0<=1; end
        2'b10 : begin Q0<=1;    Q_bar0<=0; end
        2'b11 : begin Q0<=~Q; Q_bar0<=~Q_bar; end
       endcase
     always @(negedge CLK)
       case({Q0,Q_bar0})
        2'b01 : begin Q<=0;    Q_bar<=1; end
        2'b10 : begin Q<=1;    Q_bar<=0; end
        2'b11 : begin Q<=~Q; Q_bar<=~Q_bar; end
       endcase
endmodule
```

5) D-F/F : RS-F/F나 JK-F/F의 입력단자를 하나로 만든 F/F.



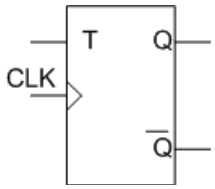
D	Q_t	Q_{t+1}	$\overline{Q_{t+1}}$
0	0	0	1
0	1	0	1
1	0	1	0
1	1	1	0

$Q_{t+1} = D$

```
module D_FF(CLK, D, Q, Q_bar);
    input CLK, D;
    output Q, Q_bar;
    reg Q, Q_bar;
     always @(negedge CLK )
       if(D)   begin Q<=1; Q_bar<=0; end
       else   begin Q<=0; Q_bar<=1; end
 endmodule
```

```
module D_FF_PR_CL(CLK, PR, CLR, D, Q, Q_bar); // preset, clear 추가
    input CLK, D, PR, CLR;
    output Q, Q_bar;
    reg Q, Q_bar;
     always @(negedge CLK or negedge PR or negedge CLR)
       if(!PR)            begin Q<=1; Q_bar<=0; end
       else if (!CLR) begin Q<=0; Q_bar<=1; end
       else               begin Q<=D; Q_bar<=~D; end
 endmodule
```

6) T(toggle)-F/F



입력신호		출력신호	
CLK	T	Q_{t+1}	$\overline{Q_{t+1}}$
↑	0	Q_t	$\overline{Q_t}$
↑	1	$\overline{Q_t}$	Q_t

```
module T_FF(CLK, T, Q, Q_bar);
    input CLK, T;
    output Q, Q_bar;
    reg Q, Q_bar;
     always @(posedge CLK)
       if(!T)   begin Q<=Q; Q_bar<=~Q; end
       else begin Q<=~Q; Q_bar<= Q; end
endmodule
```

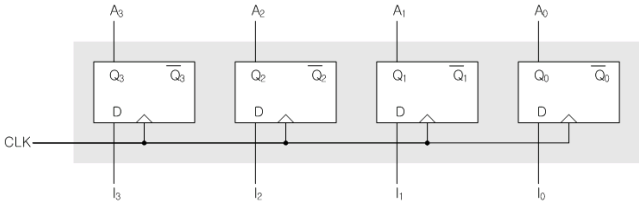
다음의 두 소스는 각각 reset 신호와 set 신호가 있는 경우에 대한 것이다.

```
module T_FF_RESET(CLK, RESET, Q, Q_bar);
    input CLK, RESET;
    output Q, Q_bar;
     reg Q, Q_bar;
       always @(negedge CLK or negedge RESET)
       if(!RESET)   begin Q<=0; Q_bar<=1; end
       else begin Q<=~Q; Q_bar<= Q; end
 endmodule
```

```
module T_FF_SET(CLK, SET, Q, Q_bar);
    input CLK, SET;
    output Q, Q_bar;
     reg Q, Q_bar;
       always @(negedge CLK or negedge SET)
       if(!SET)   begin Q<=1; Q_bar<=0; end
       else begin Q<=~Q; Q_bar<= Q; end
 endmodule
```

3.3 Register 회로 설계

1) 병렬 레지스터



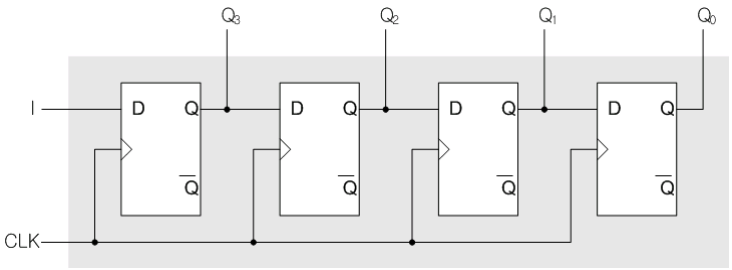
D-F/F을 이용한 4-bit 레지스터

- 레지스터 소거(clear)는 F/F의 reset을 통해 이루어짐.

입력신호				출력신호
$\overline{RESET}$	E	CLK	Data	Y
0	x	x	x	0
1	0	x	x	Y
1	1	↑	I	I
1	1	↓	I	Y

```
module RESISTER_P(CLK, RESET, E, I, Y);
    input CLK, RESET, E;
    input [3:0] I;
    output [3:0] Y;
    reg [3:0] Y;
        always @(posedge CLK or negedge RESET)
        if(!RESET) Y<= 4'b0000;
        else if (E) Y<= I;
        else Y<=4'b0000;
endmodule
```

2) 시프트 레지스터



4비트 시프트 레지스터

```
module RESISTER_S(CLK, RESET, SI, Y);
    input CLK, RESET, SI;
    output [3:0] Y;
    reg [3:0] Y;
    always @(posedge CLK or negedge RESET)
    if(!RESET) Y<=0;
    else       Y<={Y,SI};
endmodule
```

```
module RESISTER_S_1(CLK, RESET, SI, Y);
    input CLK, RESET, SI;
    output [3:0] Y;
    reg [3:0] Y;
    always @(posedge CLK or negedge RESET)
    if(!RESET) Y<=0;
    else begin Y<= Y<<1; Y[0]<= SI; end
endmodule
```

3.4 카운터 설계

- 클럭 인가 방식에 따라

 = 비동기식 카운터 : 입력 클럭 펄스가 앞단의 출력값에 영향을 받아 동작.

 = 동기식 카운터 : 각 단의 클럭을 동시에 동기시키는 방식.

 = 결합형 카운터 : 동기식과 비동기식을 결합한 형태.

- 계수 방식에 따라

= 2^n진 카운터 : 2비트 2진 카운터(4)/ 3비트 2진 카운터(8)/...

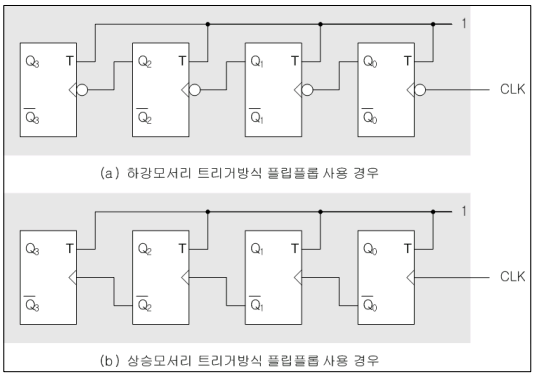
= 모듈러스 카운터 : 귀환펄스/리셋/직접 리셋을 이용.

= 시프트 카운터 : 링 카운터, 존슨 카운터.

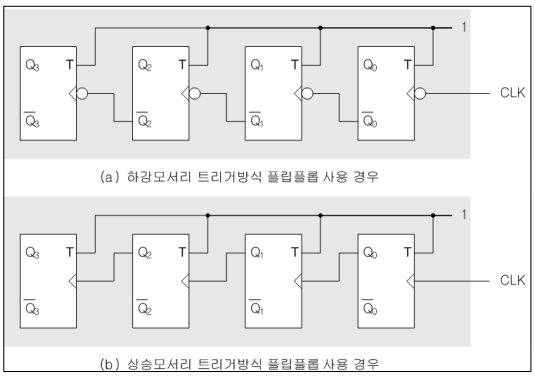
1) 비동기식 2진 카운터 (2^n진 카운터)

- 업 카운터 : 전단의 출력 Q를 다음 단의 하강 클럭이나, 전단의 출력 $\overline{Q}$를 다음 단의 상승 클럭에 인가.

- 다운 카운터 : 업 카운터의 반대로 연결.



업 카운터



다운 카운터

업 카운터			
Q_3	Q_2	Q_1	Q_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

다운 카운터			
Q_3	Q_2	Q_1	Q_0
1	1	1	1
1	1	1	0
1	1	0	1
1	1	0	0
1	0	1	1
1	0	1	0
1	0	0	1
1	0	0	0
0	1	1	1
0	1	1	0
0	1	0	1
0	1	0	0
0	0	1	1
0	0	1	0
0	0	0	1
0	0	0	0

```
module COUNT4_AS_R(CLK,RESET,Y3,Y2,Y1,Y0);
   input CLK, RESET;     // 4비트 2진 업 카운터(T-F/F)
   output Y3,Y2,Y1,Y0;
   reg Y3,Y2,Y1,Y0;

   always @(negedge CLK or negedge RESET)
     if(!RESET) Y0<=0;
     else       Y0<=~Y0;
   always @(negedge Y0 or negedge RESET)
     if(!RESET) Y1<=0;
     else       Y1<=~Y1;
   always @(negedge Y1 or negedge RESET)
     if(!RESET) Y2<=0;
     else       Y2<=~Y2;
   always @(negedge Y2 or negedge RESET)
     if(!RESET) Y3<=0;
     else       Y3<=~Y3;
endmodule
```

```
module COUNT4_AS_R_1(CLK,RESET,Y3,Y2,Y1,Y0);
   input CLK, RESET; // 4bit 2진 업 카운터(D-F/F)
   output Y3,Y2,Y1,Y0;
   reg Y3,Y2,Y1,Y0;

   always @(posedge CLK or negedge RESET)
     if(!RESET) Y0<=0;
     else       Y0<=~Y0;
   always @(negedge Y0 or negedge RESET)
     if(!RESET) Y1<=0;
     else       Y1<=~Y1;
   always @(negedge Y1 or negedge RESET)
     if(!RESET) Y2<=0;
     else       Y2<=~Y2;
   always @(negedge Y2 or negedge RESET)
     if(!RESET) Y3<=0;
     else       Y3<=~Y3;
endmodule
```

3) 비동기식 모듈러스 카운터

- 모듈러스 카운터 : 원하는 값까지만 카운터한 다음 다시 처음부터 카운터.

① 6진 카운터

입력		출력		
CLK		Y2	Y1	Y0
0	↓	0	0	0
1	↓	0	0	1
2	↓	0	1	0
3	↓	0	1	1
4	↓	1	0	0
5	↓	1	0	1
0(6)	↓	0	0	0

```
module COUNT6_MOD(CLK, RESET, Y);
    input CLK, RESET;
    output [2:0] Y;
    reg [2:0] Y;
    always @(negedge CLK or posedge RESET)
      if(RESET)     Y<=3'b000;
      else if(Y==0) Y<=3'h1;
      else if(Y==1) Y<=3'h2;
      else if(Y==2) Y<=3'h3;
      else if(Y==3) Y<=3'h4;
      else if(Y==4) Y<=3'h5;
      else if(Y==5) Y<=3'h0;
endmodule
```

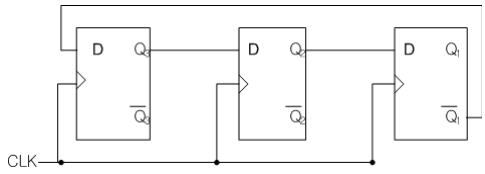
4) 동기식 카운터

입력		출력			
CLK		Y3	Y2	Y1	Y0
0	↓	0	0	0	0
1	↓	0	0	0	1
2	↓	0	0	1	0
3	↓	0	0	1	1
4	↓	0	1	0	0
5	↓	0	1	0	1
6	↓	0	1	1	0
7	↓	0	1	1	1
8	↓	1	0	0	0
9	↓	1	0	0	1
0(10)	↓	0	0	0	0

```
module COUNT10_SYN(CLK, RESET, Y); // 10진 카운터
    input CLK, RESET;
    output [3:0] Y;
    reg [3:0] Y;
     always @(negedge CLK or posedge RESET)
       if(RESET) Y<=4'b0000;
       else if (Y==9) Y<=0;
       else Y<=Y+1;
endmodule
```

```
module COUNT8_SYN(CLK, RESET, Y); // 8진 카운터
    input CLK, RESET;
    output [2:0] Y;
    reg [2:0] Y;
     always @(negedge CLK or posedge RESET)
       if(RESET) Y<=3'b000;
       else Y<=Y+1;
endmodule
```

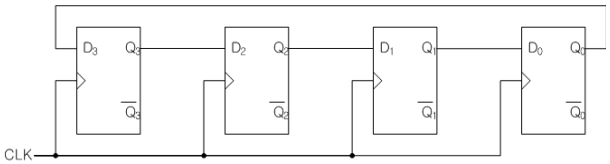
5) 3비트 존슨 카운터

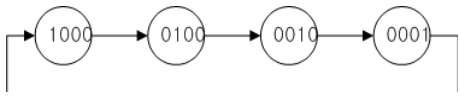


입력		출력		
CLK		Y2	Y1	Y0
0	↓	0	0	0
1	↓	0	0	1
2	↓	0	1	1
3	↓	1	1	1
4	↓	1	1	0
5	↓	1	0	0
1(6)	↓	0	0	1

```
module JOHNSON_C3(CLK, RESET, Y);
    input CLK, RESET;
    output [2:0] Y;
    reg [2:0] Y;
      always @(negedge CLK or posedge RESET)
        if(RESET) Y<=3'b000;
        else begin Y <= Y << 1; Y[0]<=~Y[2]; end
endmodule
```

6) 4비트 링 카운터



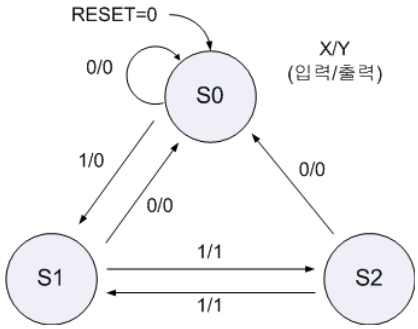


입력		출력			
CLK		Y3	Y2	Y1	Y0
0	↓	0	0	0	1
1	↓	0	0	1	0
2	↓	0	1	0	0
3	↓	1	0	0	0
0(4)	↓	0	0	0	1

```
module RING_C4(CLK, RESET, Y);
    input CLK, RESET;
    output [3:0] Y;
    reg [3:0] Y;
      always @(negedge CLK or posedge RESET)
        if(RESET) Y<=4'b0001;
        else   Y<= {Y[2],Y[1],Y[0],Y[3]};
endmodule
```

3.5 FSM(finite state machine) 회로 설계

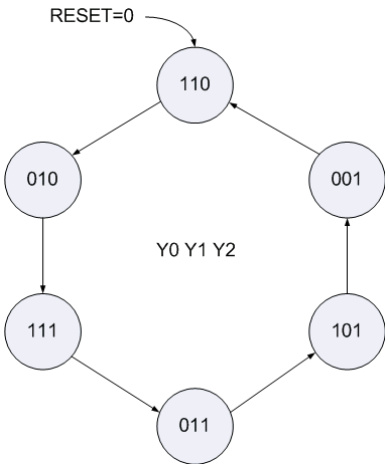
1) 밀리(Mealy) 머신




```
module MEALY_M_1(CLK,RESET,X,Y);
    input CLK, RESET,X;
    output  Y;
    reg  Y;
    reg [1:0] ST_C, ST_N;
    parameter S0=0, S1=1, S2=2;
      always @(negedge CLK or negedge RESET)
        begin
        if(!RESET) ST_C<=S0;
        else   ST_C <= ST_N;
        end
      always @(ST_C or X)
        case (ST_C)
        S0 : if(!X) begin ST_N<=S0; Y<=0; end    else begin ST_N<=S1; Y<=0;   end
        S1 : if(!X) begin ST_N<=S0; Y<=0; end    else begin ST_N<=S2; Y<=1;   end
        S2 : if(!X) begin ST_N<=S0; Y<=0; end    else begin ST_N<=S1; Y<=1;   end
        endcase
  endmodule
```

2) 무어(Moore) 머신 : skip

3.6 파형 발생기




```
module WAVE_GEN_1(CLK,RESET,Y);
    input CLK, RESET;
    output [2:0] Y;
    reg [2:0] Y;
    reg [2:0] ST_C, ST_N, TMP;
    reg TM;
    parameter S0=6, S1=2, S2=7, S3=3, S4=5, S5=1;
      always @(negedge CLK or negedge RESET)
        if(!RESET)  ST_C<=S0;
        else  ST_C <= ST_N;
      always @(ST_C)
        begin
        case (ST_C)
        S0 : begin TMP <= S0; ST_N<= S1; end
        S1 : begin TMP <= S1; ST_N<= S2; end
        S2 : begin TMP <= S2; ST_N<= S3; end
        S3 : begin TMP <= S3; ST_N<= S4; end
        S4 : begin TMP <= S4; ST_N<= S5; end
        S5 : begin TMP <= S5; ST_N<= S0; end
        endcase
        Y[0]<=TMP[2]; Y[1]<=TMP[1]; TM<=TMP[0];
        end
      always @(negedge CLK)
        Y[2] <= TM;
    endmodule
```

3.7 주파수 분주 회로

- 분주 : 입력된 클럭을 1/2, 1/5, 1/10 등의 클럭이 발생하도록 하는 것.

1) 10분주 회로

```
module freq_div_10(CLK, RESET, Y_10);
    input CLK, RESET;
    output  Y_10;
    reg  Y_10;
    reg [2:0] COUNT;

      always @(posedge CLK or posedge RESET)
        if(RESET)
            begin
              COUNT<=0;
              Y_10<=0;
            end
        else
        begin
          if(COUNT >=4)
              begin
                COUNT<=0;
                Y_10 <= !Y_10;
              end
          else COUNT<=COUNT+1;
        end
endmodule
```

2) 100분주 회로

```
module freq_div_100(CLK,RESET,Y_100);
    input CLK, RESET;
    output  Y_100;
    reg  Y_100;
    reg [6:0] COUNT;

    always @(posedge CLK or posedge RESET)
      if(RESET)
        begin
          COUNT<=0;
          Y_100<=0;
        end
     else
     begin
       if(COUNT >=49)
         begin
           COUNT<=0;
           Y_100 <= !Y_100;
         end
       else COUNT<=COUNT+1;
     end
endmodule
```

3.8 응용회로설계1 - 1000카운터 설계

```
module COUNT1000_C_SEG(CLK, RESET, SEG_C, SEG_SEL);
   input CLK, RESET;
   output [6:0] SEG_C;
   output [2:0] SEG_SEL;
   reg [6:0] SEG_C;
   reg [3:0] COUNT_1, COUNT_10, COUNT_100, COUNT_TMP, CLK_COUNT;
   reg [1:0] SEL_SEG;
   reg [2:0] SEG_SEL;
   reg CLK1;

   always @(negedge CLK or posedge RESET) // CLK/10
   if(RESET) CLK1<=0;
   else if (CLK_COUNT==9)  begin CLK_COUNT<=0; CLK1<=1; end
   else begin CLK_COUNT <= CLK_COUNT+1;  CLK1<=0; end

   always @(negedge CLK1 or posedge RESET) // COUNT_1
   if(RESET) COUNT_1<=0;
   else if (COUNT_1==9)  COUNT_1<=0;
   else COUNT_1 <= COUNT_1+1;

   always @(negedge CLK1 or posedge RESET) // COUNT_10
   if(RESET) COUNT_10<=0;
   else if (COUNT_1 ==9)            begin
    if (COUNT_10==9) COUNT_10<=0;
    else COUNT_10 <= COUNT_10+1;    end

   always @(negedge CLK1 or posedge RESET) // COUNT_100
   begin
   if(RESET) COUNT_100<=0;
   else if((COUNT_1 ==9) && (COUNT_10==9)) begin
   if (COUNT_100==9) COUNT_100<=0;
   else COUNT_100 <= COUNT_100+1;           end
   end

   always@(negedge CLK1)             // SEC_SEG
    if(SEL_SEG==2) SEL_SEG <=0;
    else SEL_SEG <= SEL_SEG+1;

   always@(COUNT_1 or COUNT_10 or COUNT_100 or SEL_SEG)
     case (SEL_SEG)
       0 : COUNT_TMP <= COUNT_1;
       1 : COUNT_TMP <= COUNT_10;
       2 : COUNT_TMP <= COUNT_100;
    endcase

   always@(SEL_SEG)
    case(SEL_SEG)
       0 : SEG_SEL <= 3'b110;
       1 : SEG_SEL <= 3'b101;
```

```
        2 : SEG_SEL <= 3'b011;
    endcase

  SEG_DEC U0 (COUNT_TMP,SEG_C);
 endmodule

module SEG_DEC(DIGIT, SEG_DEC);
  input [3:0] DIGIT;
  output [6:0] SEG_DEC;
  reg [6:0] SEG_DEC;
  always @(DIGIT)
  case (DIGIT)              // gfe_dcba
    0 : SEG_DEC <= 7'h3f;   // 011_1111
    1 : SEG_DEC <= 7'h06;   // 000_0110
    2 : SEG_DEC <= 7'h5b;   // 101_1011
    3 : SEG_DEC <= 7'h4f;   // 100_1111
    4 : SEG_DEC <= 7'h66;   // 010_0110
    5 : SEG_DEC <= 7'h6d;   // 110_1101
    6 : SEG_DEC <= 7'h7c;   // 111_1100
    7 : SEG_DEC <= 7'h07;   // 000_0111
    8 : SEG_DEC <= 7'h7f;   // 111_1111
    9 : SEG_DEC <= 7'h67;   // 110_0111
  endcase
endmodule
```

3.9 응용회로설계2 - 초시계

- 다음 소스는 입력 주파수를 1㎑로 하여 1000분주한 다음 CLK로 이용함.

1) 십의 자리와 1의 자리를 분리(separator)하는 회로 설계

```
 module SEPARATOR(CLR, FREQ, SEC_10, SEC_01);
   input CLR, FREQ;
   output [3:0] SEC_01;
   output [2:0] SEC_10;
   reg [8:0] COUNTS;
   reg [5:0] SEC;
   reg [3:0]  SEC_01;
   reg [2:0]  SEC_10;
   reg SEC_CLK,MIN_CLK;

    always@(posedge FREQ or  posedge CLR)        begin     // CLK
      if(CLR) begin COUNTS <= 0;  SEC_CLK <= 1; end
      else if (COUNTS >= 499) begin    COUNTS <= 0;  SEC_CLK <= ~SEC_CLK; end
      else                          COUNTS <= COUNTS + 1;       end

   always @(posedge SEC_CLK or  posedge CLR)  begin             // SEC_COUNT
           if(CLR)     SEC <= 0;
           else if(SEC>=59)  SEC <= 0;
           else        SEC <= SEC + 1; end
    always@(SEC)     begin                                      // SEPA_SEC
      if (SEC <= 9)      begin  SEC_10 <= 0; SEC_01 <= SEC; end
      else if (SEC <= 19) begin SEC_10 <= 1; SEC_01 <= SEC - 10; end
```

```
      else if (SEC <= 29) begin SEC_10 <= 2; SEC_01 <= SEC - 20; end
      else if (SEC <= 39) begin SEC_10 <= 3; SEC_01 <= SEC - 30; end
      else if (SEC <= 49) begin SEC_10 <= 4; SEC_01 <= SEC - 40; end
      else if (SEC <= 59) begin SEC_10 <= 5; SEC_01 <= SEC-50; end
      else                begin SEC_10 <= 0; SEC_01 <= 0; end       end
  endmodule
```

2) 동적(dynamic)인 7-세그먼트(SEG_DEC)에 출력하는 프로그램

```
module DYN_SEG_DEC(CLR,CLK, SEC_10, SEC_01, SEG_DEC, COM);
  input [3:0] SEC_10, SEC_01;
  input CLR,CLK;
  output [6:0] SEG_DEC;
  output [1:0] COM;
  reg [6:0] SEG_DEC;
  reg [3:0] DIGIT;
  reg [1:0]COM;
  always @(posedge CLK or posedge CLR)
   begin
     if(CLR) begin COM<=0; SEG_DEC<=2; end
     else  begin
      if(COM==1) begin DIGIT <=SEC_01; COM<=2'b10; end
                 else begin DIGIT<=SEC_10; COM<=2'b01; end
     case (DIGIT)              // gfe_dcba
      0 : SEG_DEC <= 7'h3f;  // 011_1111
      1 : SEG_DEC <= 7'h06;  // 000_0110
      2 : SEG_DEC <= 7'h5b;  // 101_1011
      3 : SEG_DEC <= 7'h4f;  // 100_1111
      4 : SEG_DEC <= 7'h66;  // 010_0110
      5 : SEG_DEC <= 7'h6d;  // 110_1101
      6 : SEG_DEC <= 7'h7c;  // 111_1100
      7 : SEG_DEC <= 7'h07;  // 000_0111
      8 : SEG_DEC <= 7'h7f;  // 111_1111
      9 : SEG_DEC <= 7'h67;  // 110_0111
    endcase
   end
  end
endmodule
```

3) 동적방식의 초 시계 설계

```
module DYN_SEC_CLOCK(FREQ, CLR, SEG_DEC, COM);
    input FREQ, CLR;
    output [6:0] SEG_DEC;
    output [1:0] COM;
    reg [6:0] SEG_DEC;
    reg [3:0] SEC_01;
    reg [2:0] SEC_10;
    reg [1:0] COM;
    reg SEC_CLK;

      SEPERATOR   U0(CLR, FREQ, SEC_10, SEC_01);
      DYN_SEG_DEC U1(CLR, FREQ, SEC_10, SEC_01, SEG_DEC, COM);

endmodule
```