

Lab #11: Vending Machine

자판기 설계 + FSMs

연세대학교 컴퓨터 과학과 디지털 논리회로 실습

Computer Science Department of
Yonsei University

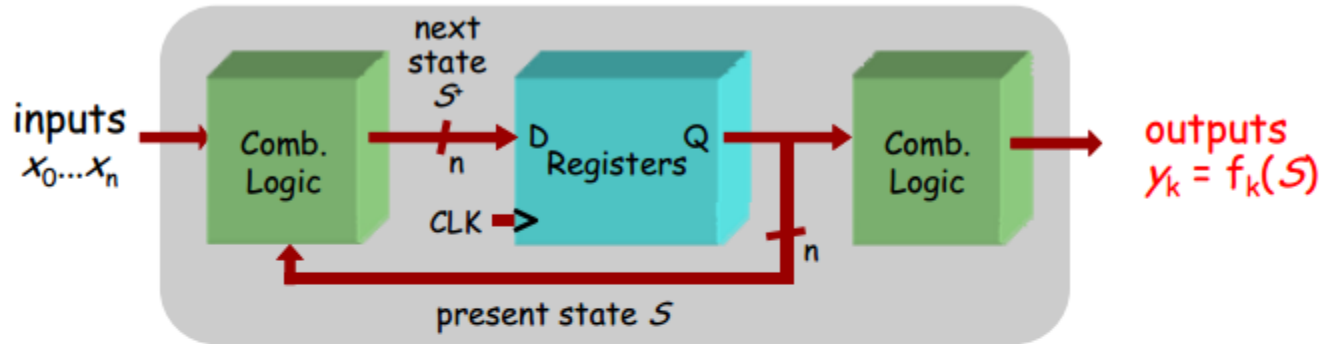


Some of these slides contain material developed and copyrighted by
Gim P. Hom (Lecturer, MIT), Prof. Steve Ward (MIT), Prof. Shin Dug Kim (Yonsei), Prof. Anantha Chandrakasan (MIT),
Prof. Arvind and Prof. Krste Asanovic (MIT)

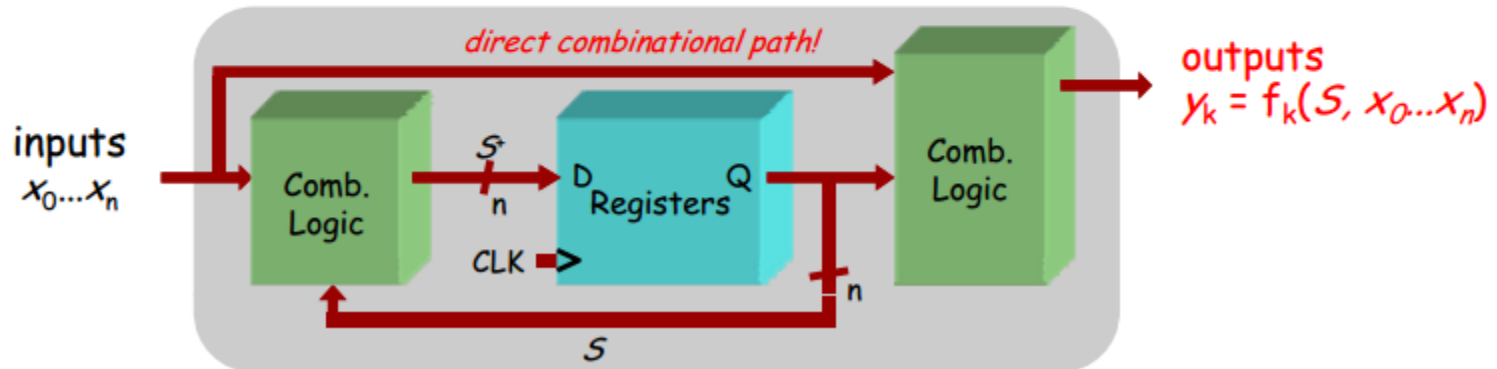
[Review] FSMs

- 두 종류의 FSM
- Moore 와 Mealy FSMs: 출력 값을 어떻게 만드는가의 차이

• Moore FSM:

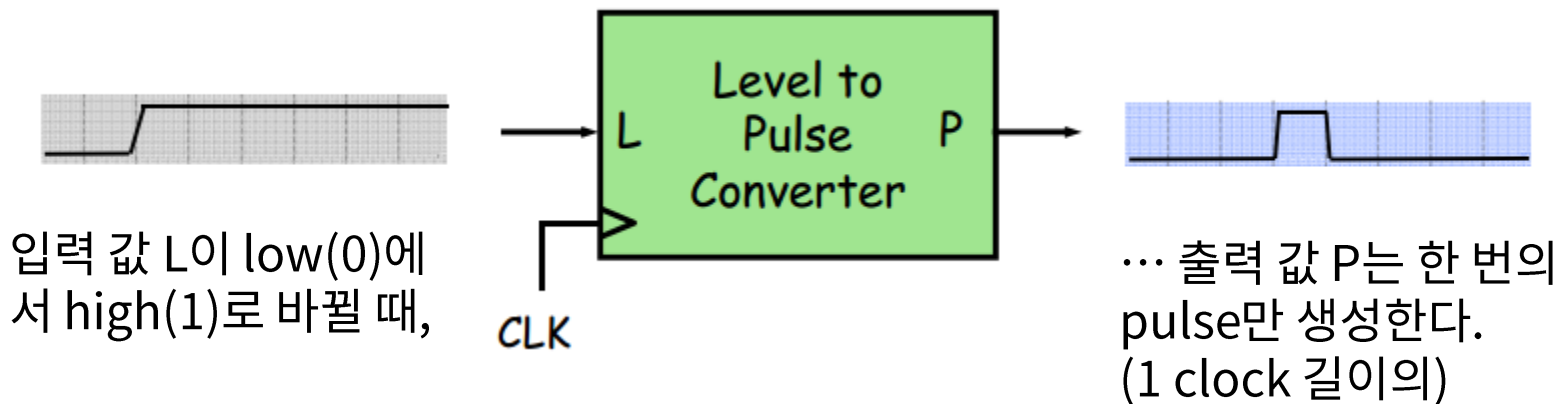


• Mealy FSM:



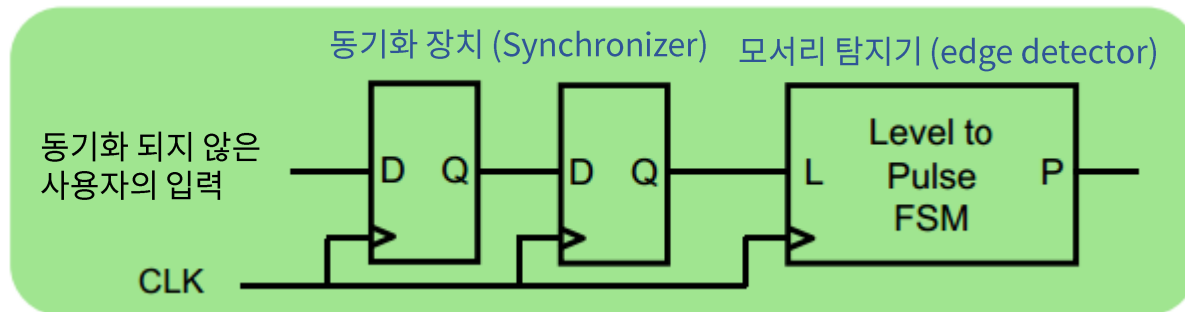
[Review] 설계 예시: Level-to-Pulse

- 이 level-to-pulse converter는 입력 값이 0에서 1로 바뀔 때 (positive edge가 감지되었을 때), single-cycle pulse를 생성한다.
- 동기적인 edge detector라고 볼 수 있다.
- 다음과 같은 특징이 있다.
 - 사람에 의해서 임의적인 순간에 버튼이나 스위치가 눌러진다.
 - 생성된 pulse는 counter에 대한 입력으로도 이루어질 수 있다.

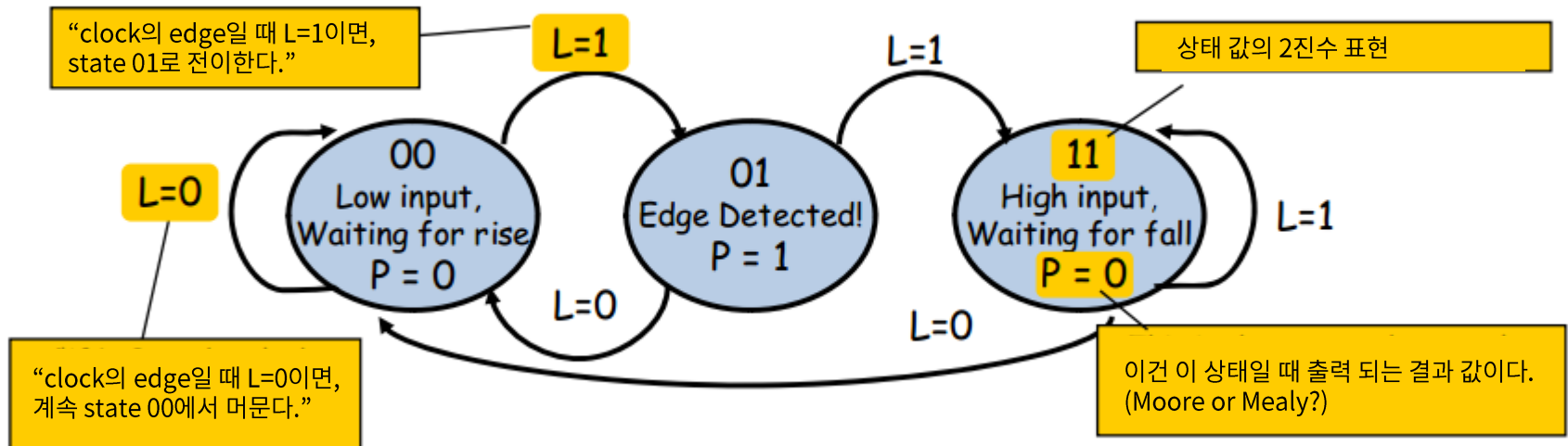


[Review] 1단계: 상태 전이도

- 우리가 목표로 하는 시스템의 블록 다이어그램

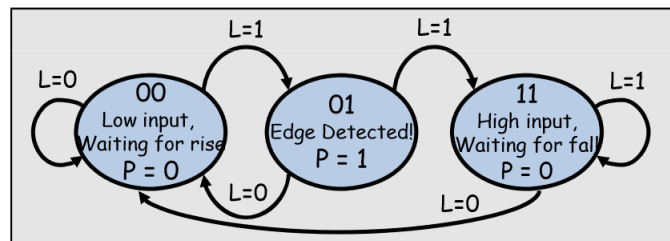


- 상태 전이도(State transition diagram)는 FSM과 설계 목표를 묘사하는 것에 있어서 유용하다.



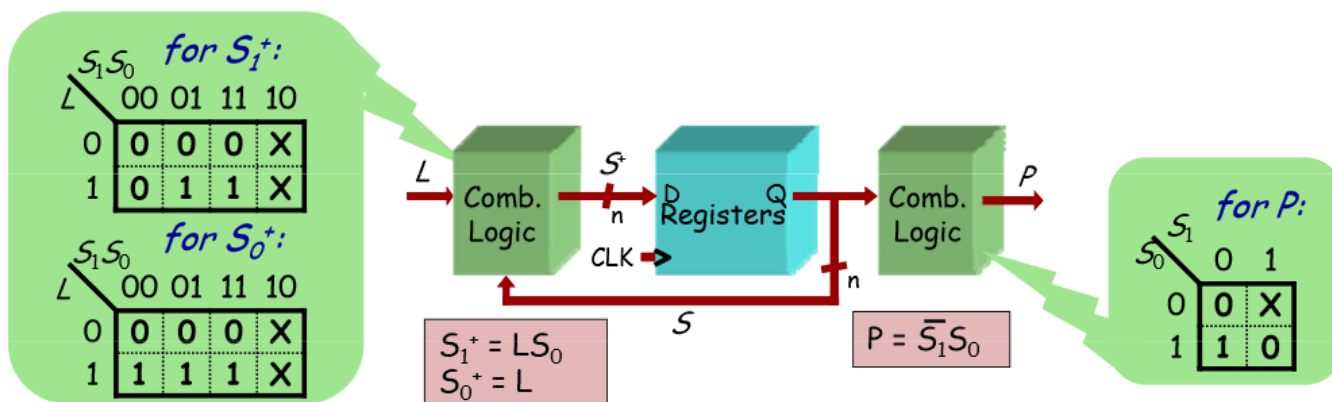
[Review] 2단계: 논리 전개

- 상태 전이도는 손쉽게 상태 전이표로 변환이 가능하다.

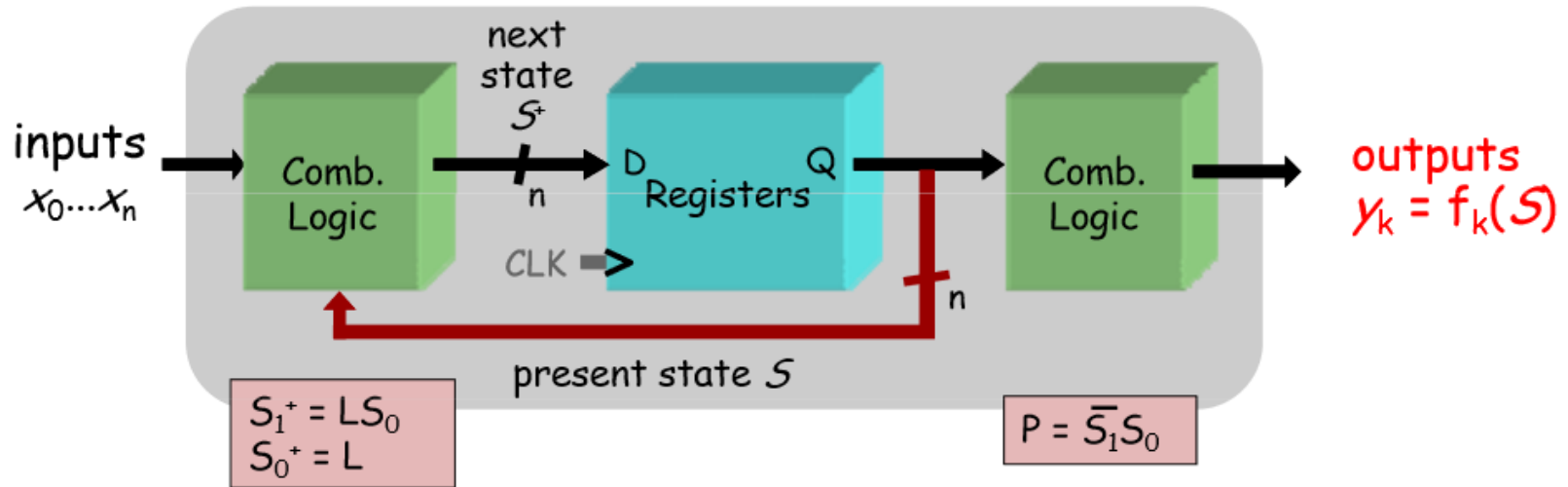


Current State		In	Next State		Out
S_1	S_0	L	S_1^+	S_0^+	P
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	1
1	1	0	0	0	0
1	1	1	1	1	0

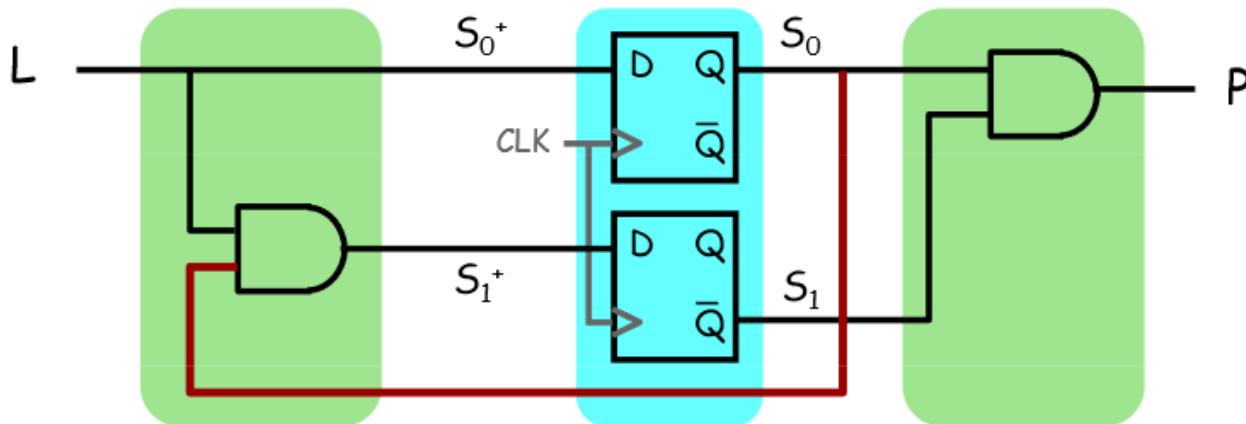
- 그리고 Karnaugh maps을 이용해서, 조합 논리 회로를 도출할 수 있다.



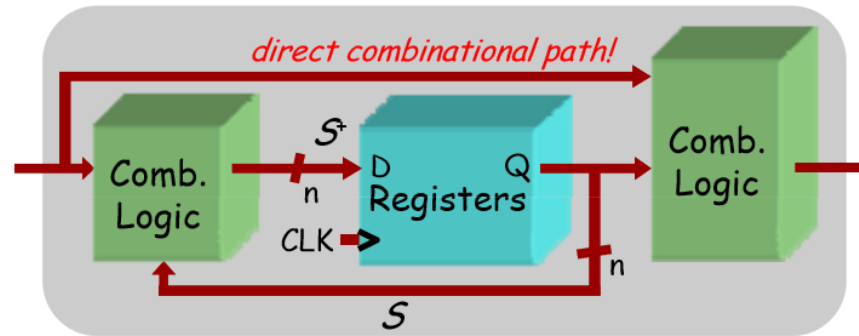
[Review] Moore machine 형태의 Level-to-Pulse 전환기



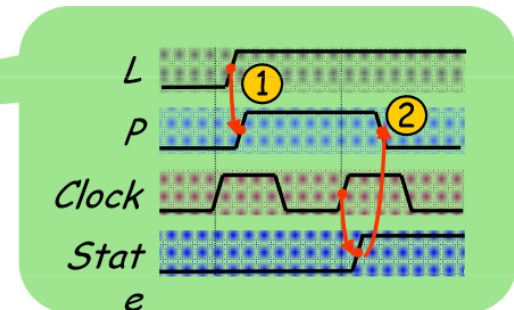
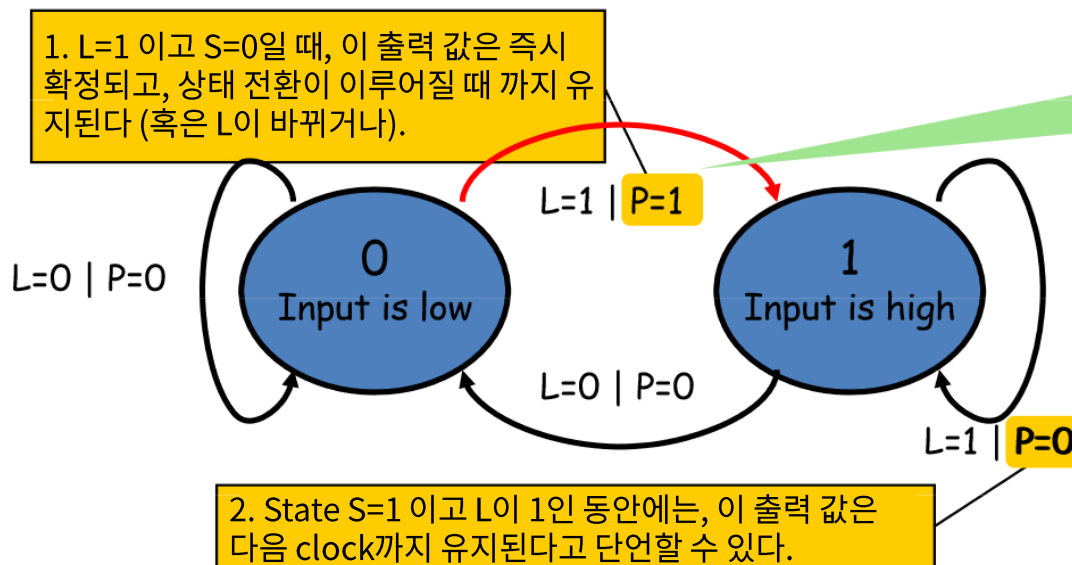
- Level-to-pulse 전환기의 Moore FSM 회로 구현:



[Review] Mealy Level-to-pulse의 설계

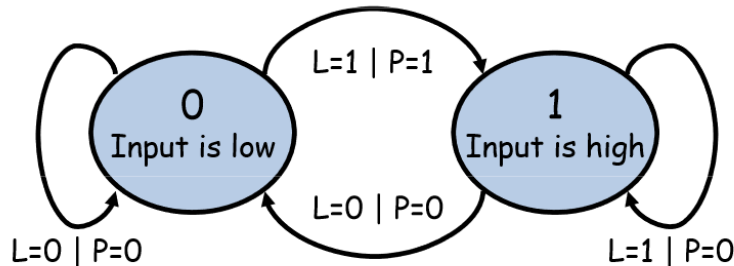


- 출력 값이 입력 값과, 현재의 상태에 의해서 결정된다.
- 따라서 Mealy FSM들은 Moore FSM 구현 물과 비교할 때, 좀 더 적은 상태의 개수를 가진다.



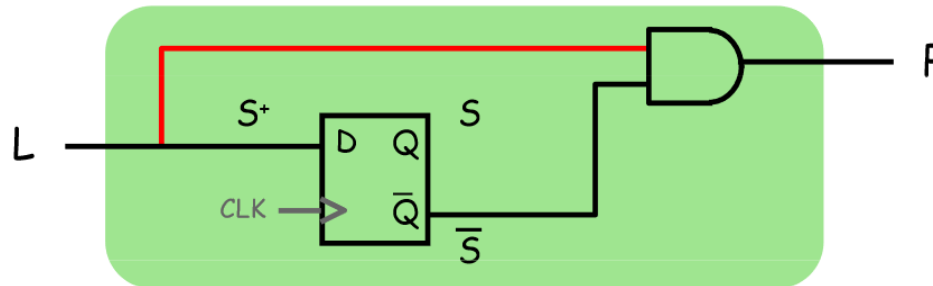
출력 값이 바뀌는 건 순간적이다.
상태가 전환되는 시점은 clock의 edge 부분.

[Review] Mealy Level-to-pulse 전환기



Pres. State	In	Next State	Out
S	L	S^+	P
0	0	0	0
0	1	1	1
1	0	0	0
1	1	1	0

- Level-to-pulse 전환기의 Mealy FSM 회로의 구현 물:

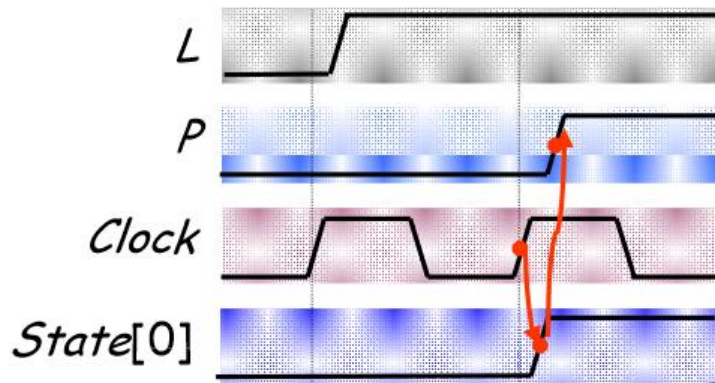


- FSM의 state는 아주 간단하게 직전 L 값을 저장한다.

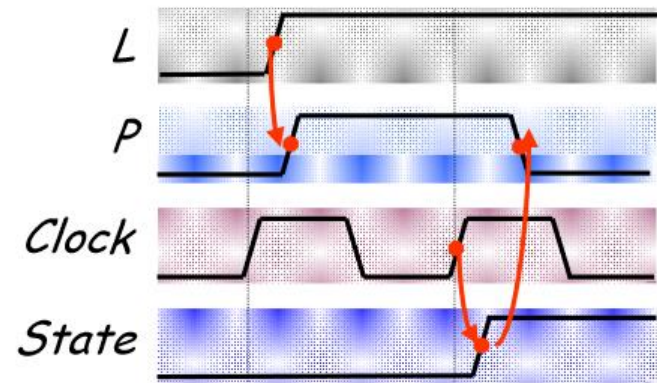
[Review] Moore/Mealy Trade-Offs

- 도대체 Moore machine과 Mealy machine은 무엇이 다른 거야?
 - Moore: $\text{outputs} = f(\text{state})$ // 오직 현재 state를 기반으로 해서 결과 값이 산출 됨.
 - Mealy: $\text{outputs} = f(\text{state and input})$ // 현재 state와 입력 값에 의존적임.
 - Mealy의 결과 값은 보통 Moore의 결과 값보다 1 cycle 빠르게 산출된다.

Moore: delayed assertion of P



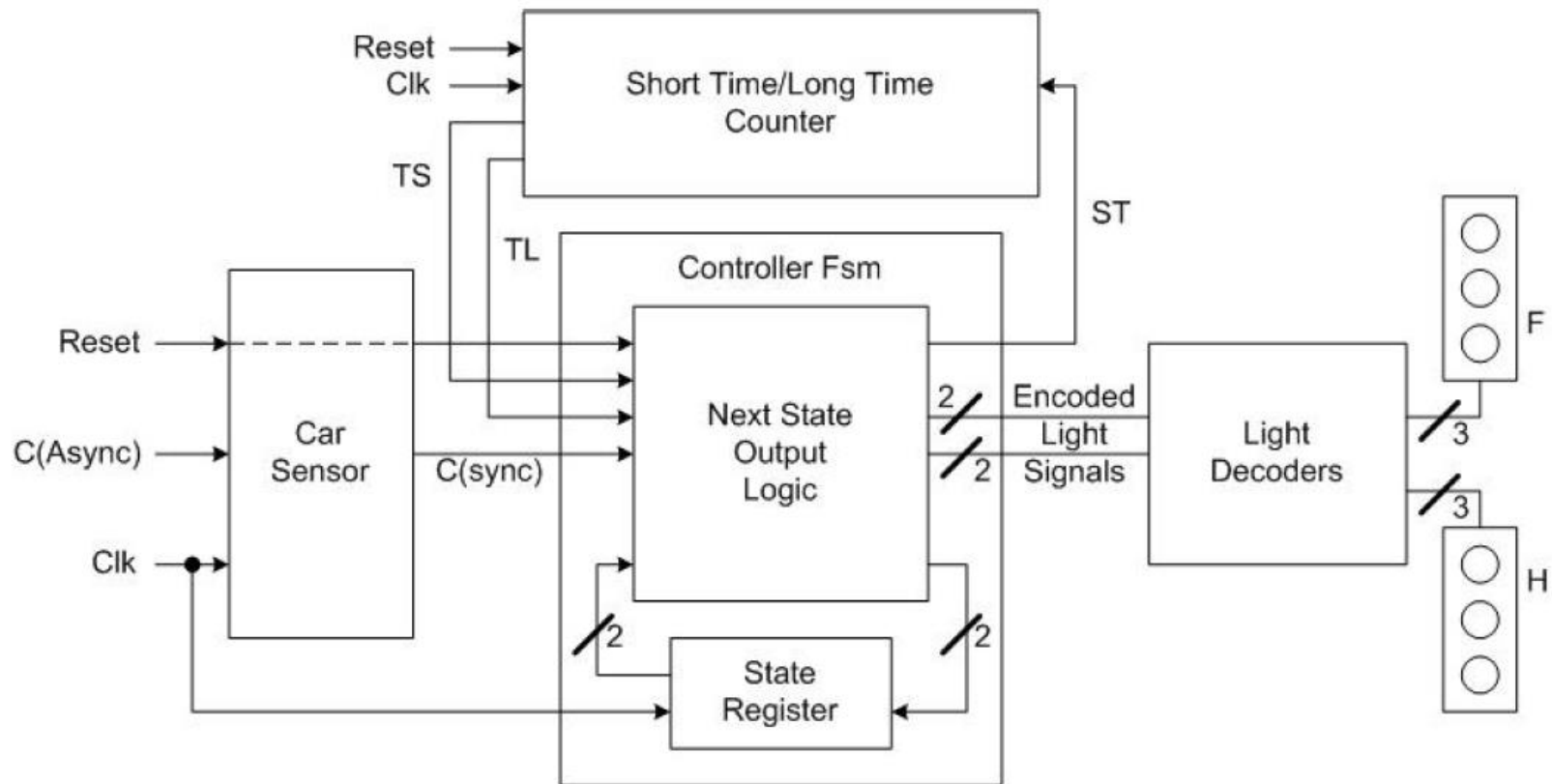
Mealy: immediate assertion of P



- Moore FSM과 비교할 때, Mealy FSM은 보통...
 - 개념화 하거나 설계하는 과정에서 좀 더 복잡함을 느낀다.
 - 하지만 더 적은 state를 가진다.

[Review] Traffic Light Controller

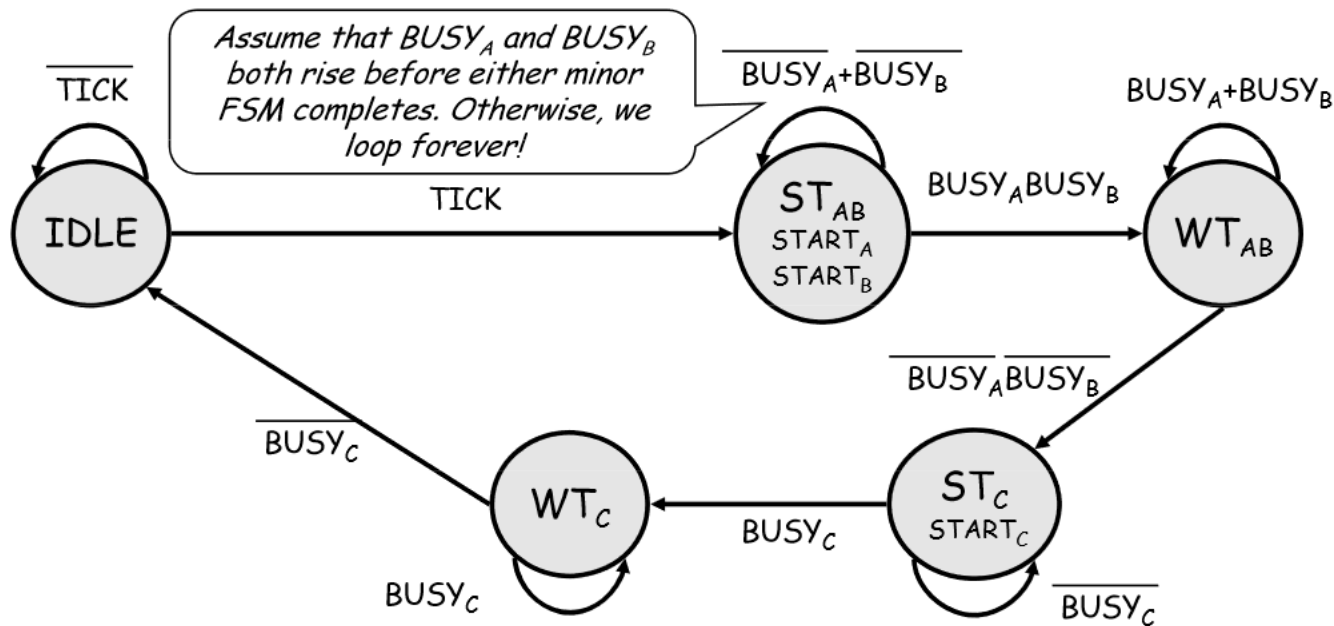
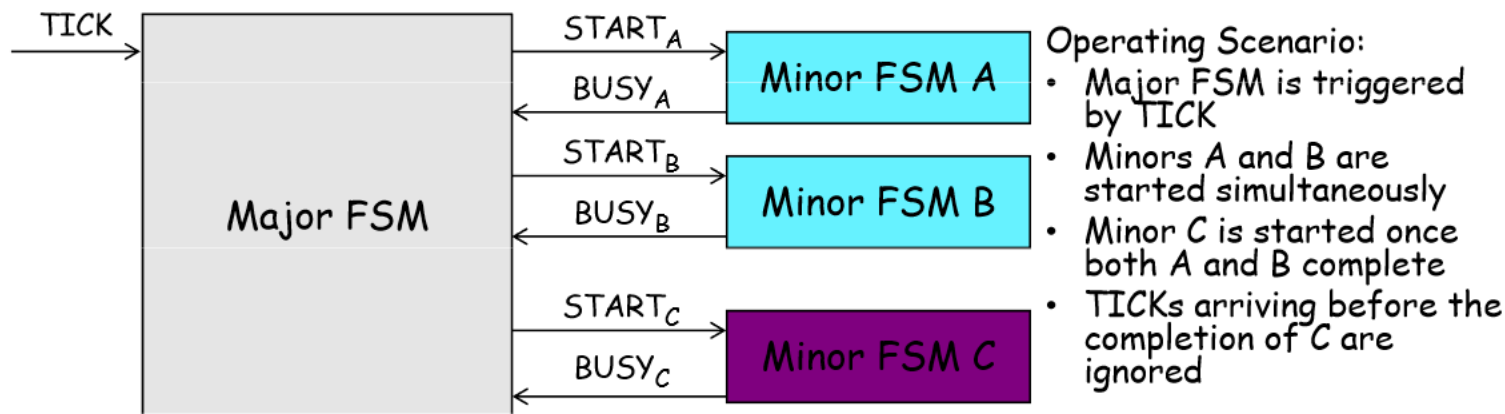
- 신호등 제어기의 Block Diagram



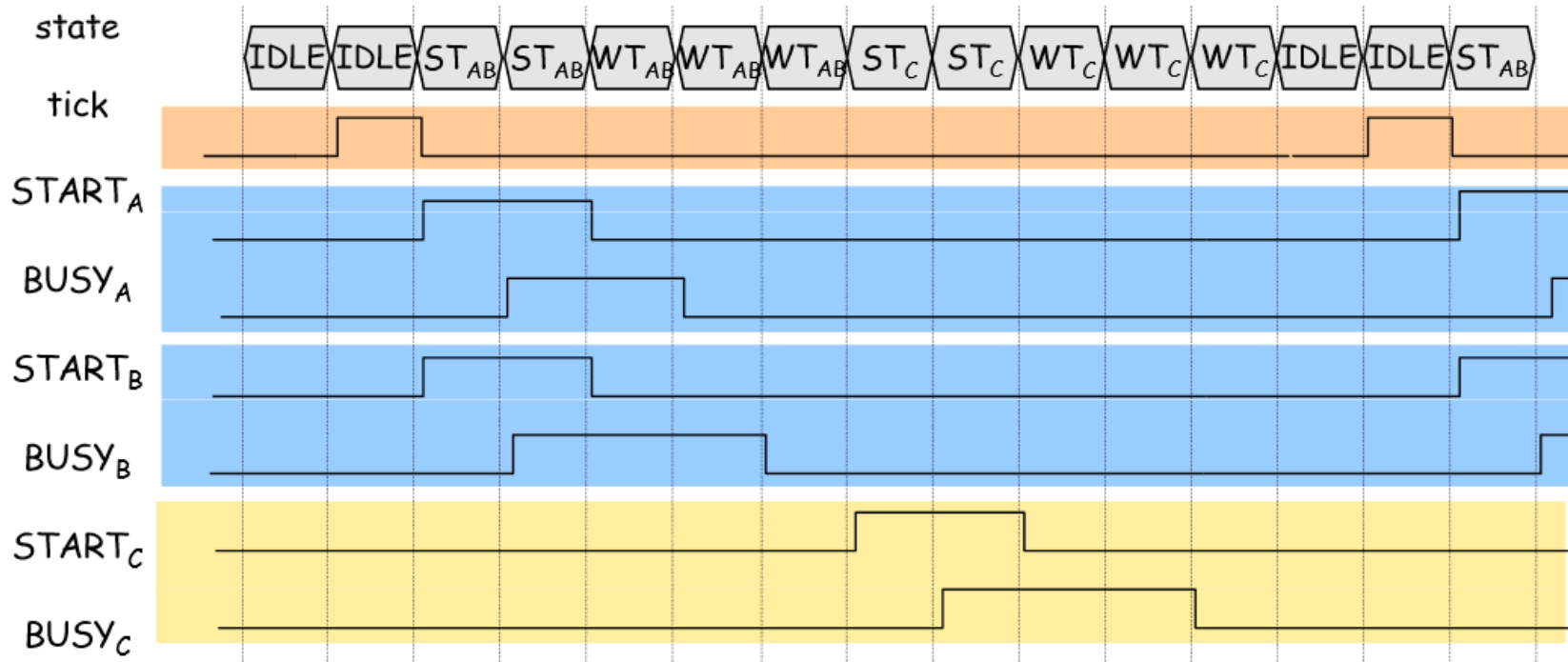
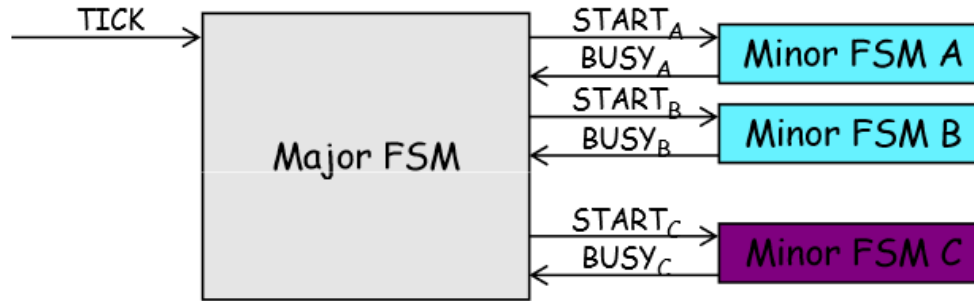
[Review] Traffic Light Controller

- Communicating FSMs: 2개 이상의 FSM이 적용된 설계에서의 이슈
 - Next State Output Logic – main FSM
 - Short Time/Long Time Counter – minor FSM
 - 이 2개의 FSM은 서로 영향을 준다 → 서로의 상태에 대해서 알려주기 위한 통신이 필요
 - Short Time/Long Time Counter → Next State Output Logic
 - TS와 TL이 사용됨: TS는 Short Timer가 만료되었을 때, TL은 Long Timer가 만료되었을 때 신호(1)를 보냄
 - Next State Output Logic → Short Time/Long Time Counter
 - ST가 사용됨: ST는 Timer의 작동을 다시 초기 상태로 돌림 (즉, 해당 시점부터 다시 타이머를 사용하려 할 때)
- 왜 굳이 한 시스템 내부에서 다중의 FSM을 사용할까?
 - Minor FSM이 적용되지 않는다면, Traffic Light Controller의 Main FSM의 state 수는 급격히 늘어난다.
 - 가령, 1초마다 clock 신호가 들어온다 하면, TL가 7초 일 때 최소 7번의 상태 변화가 일어난다.
 - 만약 여기에 C (차량 감지) 센서 입력까지 조건으로 붙으면 더 복잡해진다.
 - 짧은/긴 시간 간격이 지났음을 알려주는 Timer 모듈 하나를 따로 만드는 것만으로도, 많은 수의 state를 절약할 수 있다.

다중 FSM 예시: 4개의 FSM



다중 FSM 예시: 4개의 FSM

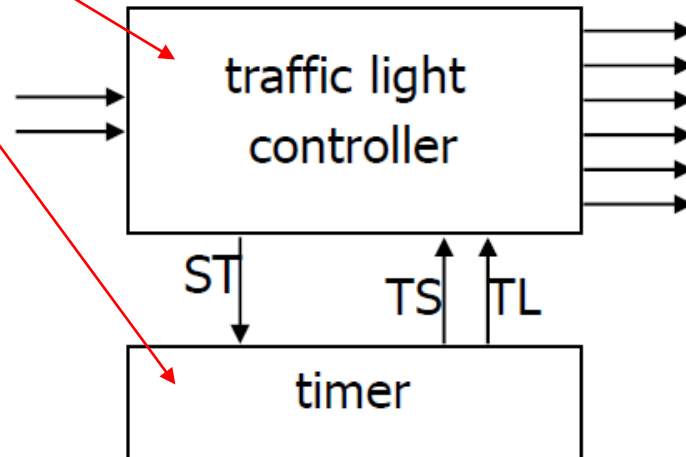


[Review] Traffic Light Controller

- Verilog HDL로 구현하기 / FPGA로 Programming 하기
- 스타일 #01] Structural Verilog (이론 수업 slide 참조)

```
module main(HR, HY, HG, FR, FY, FG, reset, C, Clk);  
  output HR, HY, HG, FR, FY, FG;  
  input  reset, C, Clk;
```

```
  Timer part1(TS, TL, ST, Clk);  
  FSM part2(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, Clk);  
endmodule
```

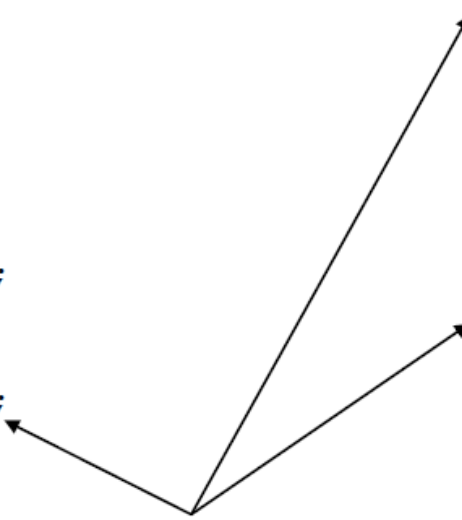


[Review] Traffic Light Controller

- Traffic Light Controller의 main module

- 여기서 output logic도 함께 포함되어 있다.
- Main FSM

```
module FSM(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, Clk);  
    output    HR;  
    output    HY;  
    output    HG;  
    output    FR;  
    output    FY;  
    output    FG;  
    output    ST;  
    input     TS;  
    input     TL;  
    input     C;  
    input     reset;  
    input     Clk;  
  
    reg [6:1] state;  
    reg      ST;  
  
    parameter highwaygreen  = 6'b001100;  
    parameter highwayyellow = 6'b010100;  
    parameter farmroadgreen  = 6'b100001;  
    parameter farmroadyellow = 6'b100010;  
  
    assign HR = state[6];  
    assign HY = state[5];  
    assign HG = state[4];  
    assign FR = state[3];  
    assign FY = state[2];  
    assign FG = state[1];  
endmodule
```



FSM에서 상태를 나타내는 bit들과
출력을 위한 코드부분

[Review] Traffic Light Controller

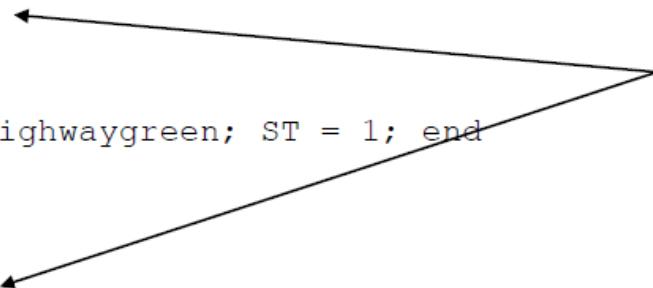
- Traffic Light Controller의 main module

- 여기서 output logic도 함께 포함되어 있다.
- Main FSM

```
initial begin state = highwaygreen; ST = 0; end

always @(posedge Clk)
begin
    if (reset)
        begin state = highwaygreen; ST = 1; end
    else
        begin
            ST = 0;
            case (state)
                highwaygreen:
                    if (TL & C) begin state = highwayyellow; ST = 1; end
                highwayyellow:
                    if (TS) begin state = farmroadgreen; ST = 1; end
                farmroadgreen:
                    if (TL | !C) begin state = farmroadyellow; ST = 1; end
                farmroadyellow:
                    if (TS) begin state = highwaygreen; ST = 1; end
            endcase
        end
    end
endmodule
```

clock edge에 의해서
작동되는 case 문



[Review] Traffic Light Controller

- Timer FSM
 - Main FSM을 지원하기 위한, minor FSM

```
module Timer(TS, TL, ST, Clk);  
    output TS;  
    output TL;  
    input    ST;  
    input    Clk;  
    integer  value;  
  
    assign TS = (value >= 4); // 5 cycles after reset  
    assign TL = (value >= 14); // 15 cycles after reset  
  
    always @(posedge ST) value = 0; // async reset  
  
    always @(posedge Clk) value = value + 1;  
  
endmodule
```

[Review] Traffic Light Controller

스타일 #02]

```

1 module traffic_light_controller(clk, reset_in, C, farmroad_light, highway_light);
2     input    clk, reset_in, C;
3     output   [2:0] farmroad_light, highway_light;
4
5     reg      [1:0] next_state, state;
6     wire     TS, TL;
7     reg      ST;
8     reg      car_detect;
9
10    wire      reset;
11
12    parameter HG = 0, HY = 1, FG = 2, FY = 3;
13
14    button          b_reset(clk, reset_in, reset);
15    timer_for_controller time_counter(clk, ST, TL, TS);
16
17
18
19    always @ (*) begin
20        if (reset) begin
21            next_state = HG;
22            ST = 1;
23        end
24        else begin
25            case (state)
26                HG: next_state = (C & TL ? HY : state);
27                HY: next_state = (TS ? FG : state);
28                FG: next_state = (TL|~C ? FY : state);
29                FY: next_state = (TS ? HG : state);
30                default: next_state = HG;
31            endcase
32            if (state != next_state) ST = 1;
33            else ST = 0;
34        end
35    end
36
37    always @ (posedge clk) state <= next_state;
38
39    assign farmroad_light[2] = ~state[1];
40    assign farmroad_light[1] = state[1]&state[0];
41    assign farmroad_light[0] = state[1]&~state[0];
42
43    assign highway_light[2] = state[1];
44    assign highway_light[1] = ~state[1]&state[0];
45    assign highway_light[0] = ~state[1]&~state[0];
46
47 endmodule
48

```

```

1 module button(clk, b_in, b_out);
2     input    clk, b_in;
3     output   b_out;
4
5     reg      r1, r2, r3;
6
7     always @(posedge clk)
8     begin
9         r1 <= b_in;
10        r2 <= r1;
11        r3 <= r2;
12    end
13
14    assign b_out = ~r3 & r2;
15
16 endmodule
17

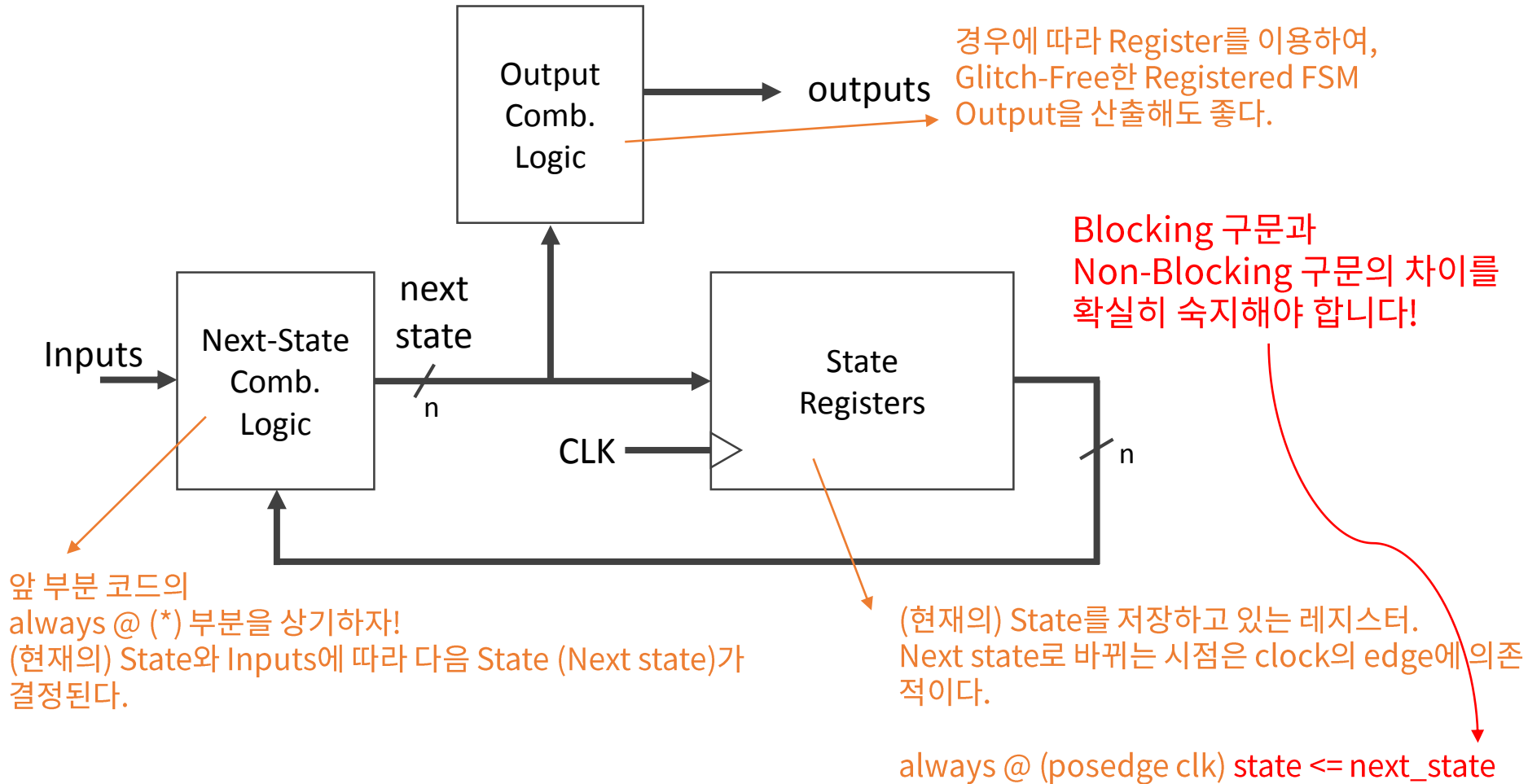
```

```

1 module timer_for_controller(clk, st, tl, ts);
2     input    clk, st;
3     output   tl, ts;
4
5     reg      [15:0] count;
6     reg      [15:0] msec;
7
8     //32.768 * 10^3 * 10^3
9     //32768 * 10^3
10    always @ (posedge clk)
11    begin
12        if (st == 1) msec = 0;
13        if (count == 32768) msec = msec + 1;
14        count = count + 1;
15    end
16
17
18    assign ts = (msec >= 1000);
19    assign tl = (msec >= 2500);
20
21 endmodule
22

```

[Review] Traffic Light Controller



Vending Machine: 자판기

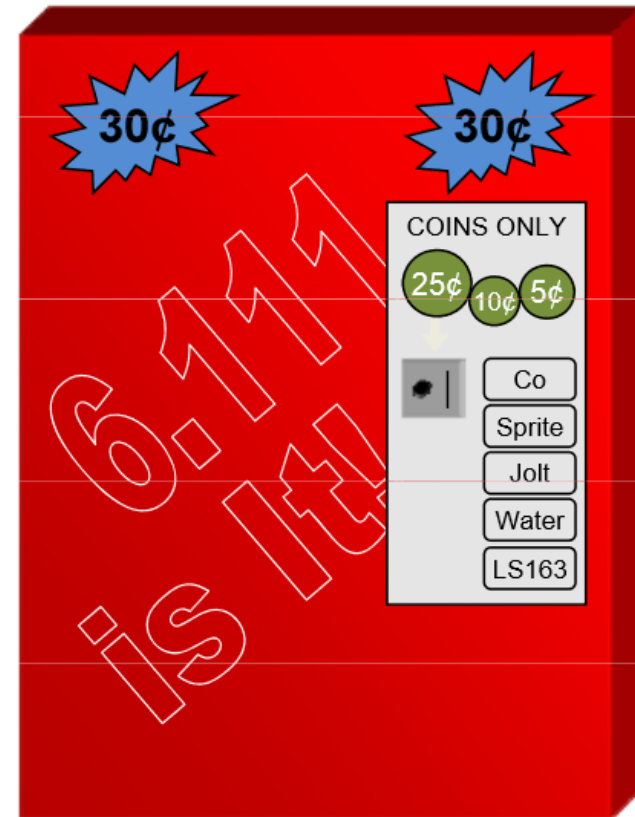
- 간단한 자판기 설계
 - 커피 한 종류만 제공하는 자판기
 - 커피 한 잔: 150원
 - 투입 가능한 동전 종류
 - 50원 / 100원
 - 동작 과정
 - 사용자가 Reset 버튼을 누른 후, 동전을 넣어서 150원이 되면 자동으로 커피가 나온다.
 - 우선적으로 잔돈을 거슬러 주지 않는 자판기를 설계한 후,
 - 추가적으로 200원이 들어오면 잔돈을 거슬러 주는 기능을 설계해본다.
- 거스름돈 기능 구현 시:
 - Inputs: 1 per clock으로 제한
 - coin_50: 동전 인식기에서 50원을 인식함 (50원이 들어옴)
 - coin_100: 동전 인식기에서 100원을 인식함 (100원이 들어옴)
 - Outputs: 마찬가지로, 1 per clock으로 제한
 - coffee_out: coffee가 나옴
 - change_out: 거스름돈이 나옴 (여기서는 50원)
- FPGA에서가 아닌, Modelsim을 통한 시뮬레이션 진행

Vending Machine: 자판기

- 간단한 자판기 설계
 - 유의 사항
 - 간단한 커피 자판기를 설계해 보는 것이 목적이므로 다음과 같은 한계점이 존재함
 - 최대 200원 까지 밖에 받을 수 없음
 - 동전 삽입 조합은 다음과 같음
 - (50원, 50원, 50원), (50원, 100원), (100원, 50원), (100원, 100원 '거스름돈이 생기는 경우')
 - Simulation은 testbench로 타이밍을 쉽게 통제할 수 있는 환경임
 - 실제 FPGA에서 올바르게 동작하기 위해서는 입출력의 시간 통제에 신경을 써야 함.

Vending Machine: 자판기

- 다른 사례: MIT 6.111의 Vending Machine
- 우리가 오늘 실습에서 다룰 자판기 보다 조금 더 복잡함
 - 음료가 여러 가지 있음. (하지만 가격은 모두 \$0.30으로 동일)
 - 동전의 개수가 3개임. (25, 10, 5센트)
- 입력
 - Q – quarter (25 센트)
 - D – dime (10 센트)
 - N – nickel (5 센트)
- 출력
 - DC – 음료수 캔이 나옴
 - DD – dime을 거슬러 줌
 - DN - nickel을 거슬러 줌



이 시스템에서는 어떤 State들이 존재하는가?

- 초기 상태 (내지는 대기(idle) 상태)



- 돈이 들어와서 적립되는 모든 경우의 수에 대한 상태



- 그럼 자판기에 최대로 넣을 수 있는 돈은 얼마일까?

- 25 센트 (아직 음료를 구매할 수 없는 조합 중 최대인 값) + 1 quarter (25센트, 가장 큰 동전)

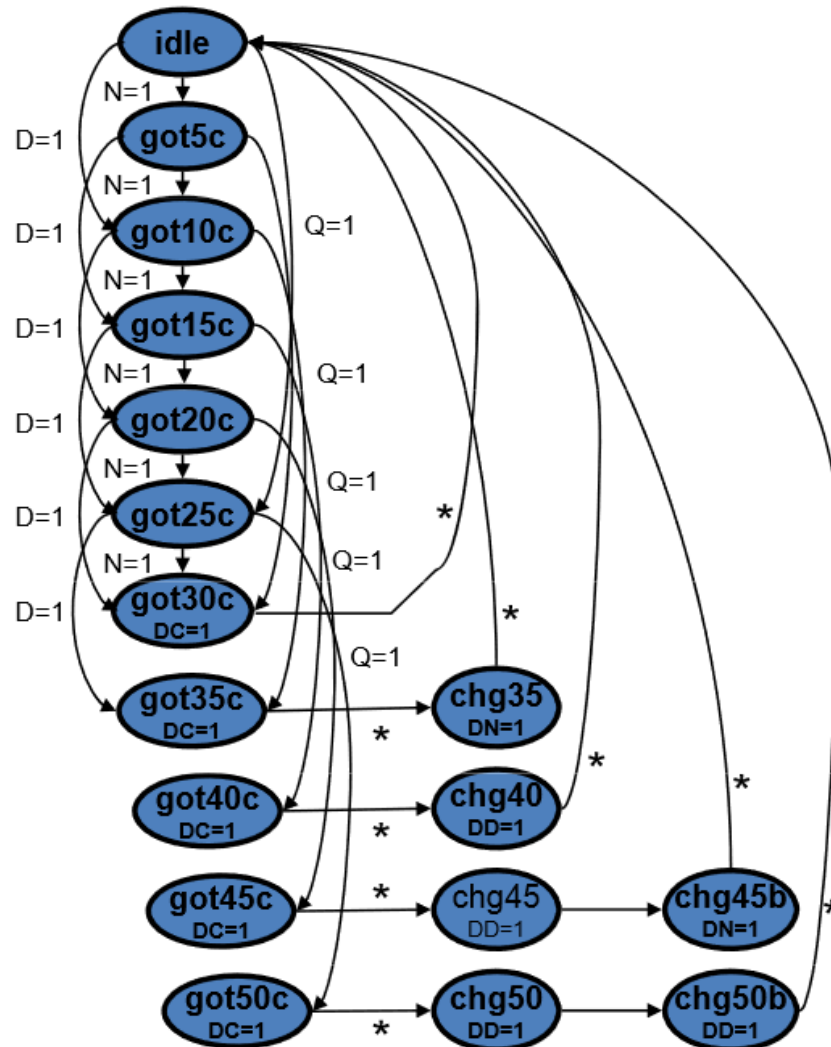


- 캔이 나오고, 거스름 돈을 거슬러주는 과정에서의 상태들



Moore 자판기

- 상태 전이도

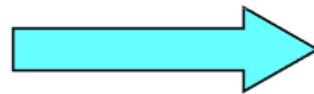
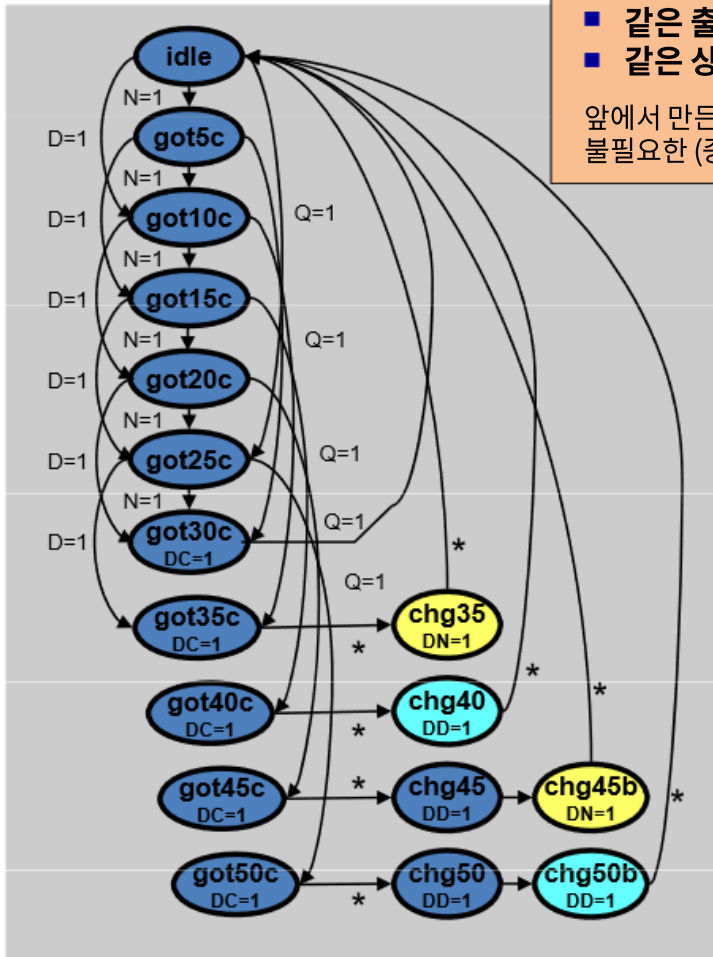


상태를 줄여보자 - 최적화

중복된 상태들은 다음을 가진다:

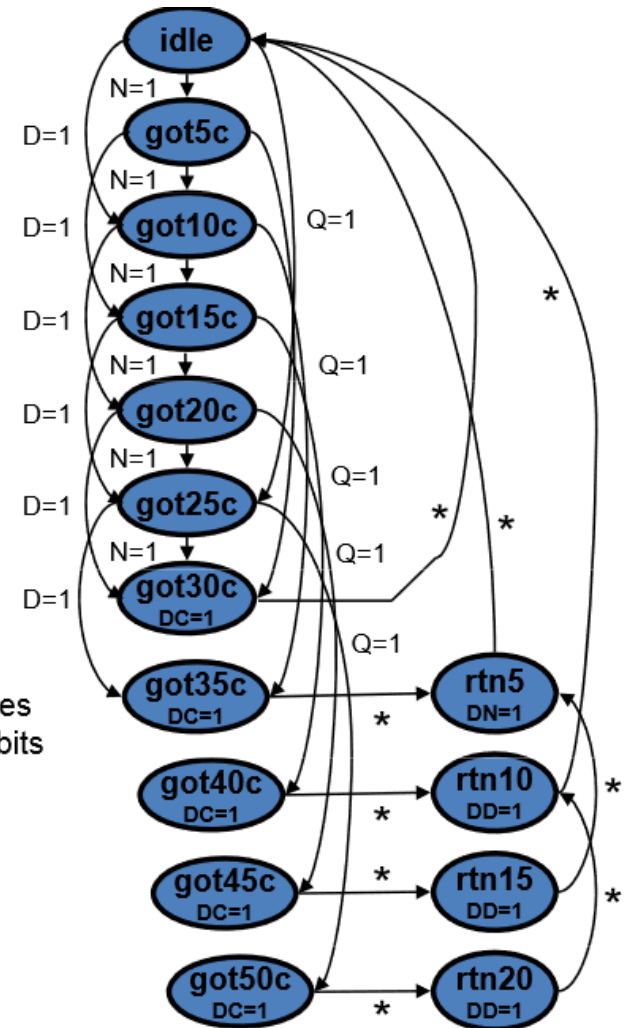
- 같은 출력 값
- 같은 상태 전환

앞에서 만든 다이어그램에서는 2개의 불필요한 (중복된) 상태들이 존재한다.

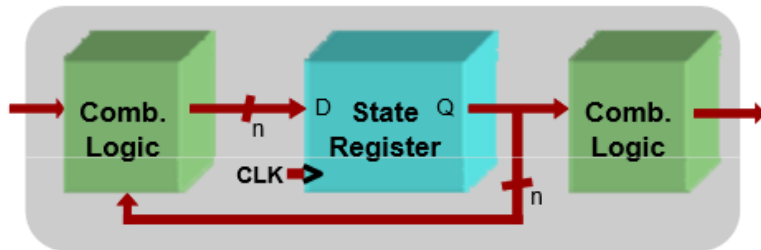


17 states
5 state bits

15 states
4 state bits



Moore 자판기 구현: Verilog



***FSMs are easy in Verilog.
Simply write one of each:***

- **State register**
(sequential always block)
- **Next-state combinational logic**
(comb. always block with case)
- **Output combinational logic block**
(comb. always block *or* assign statements)

```
module mooreVender (
    input N, D, Q, clk, reset,
    output DC, DN, DD,
    output reg [3:0] state);

    reg next;
```

States defined with **parameter keyword**

```
parameter IDLE = 0;
parameter GOT_5c = 1;
parameter GOT_10c = 2;
parameter GOT_15c = 3;
parameter GOT_20c = 4;
parameter GOT_25c = 5;
parameter GOT_30c = 6;
parameter GOT_35c = 7;
parameter GOT_40c = 8;
parameter GOT_45c = 9;
parameter GOT_50c = 10;
parameter RETURN_20c = 11;
parameter RETURN_15c = 12;
parameter RETURN_10c = 13;
parameter RETURN_5c = 14;
```

State register defined with sequential always block

```
always @ (posedge clk or negedge reset)
    if (!reset) state <= IDLE;
    else state <= next;
```

Moore 자판기 구현: Verilog

Next-state logic within a combinational **always** block

```
always @ (state or N or D or Q) begin
    case (state)
        IDLE:      if (Q) next = GOT_25c;
                   else if (D) next = GOT_10c;
                   else if (N) next = GOT_5c;
                   else next = IDLE;

        GOT_5c:    if (Q) next = GOT_30c;
                   else if (D) next = GOT_15c;
                   else if (N) next = GOT_10c;
                   else next = GOT_5c;

        GOT_10c:   if (Q) next = GOT_35c;
                   else if (D) next = GOT_20c;
                   else if (N) next = GOT_15c;
                   else next = GOT_10c;

        GOT_15c:   if (Q) next = GOT_40c;
                   else if (D) next = GOT_25c;
                   else if (N) next = GOT_20c;
                   else next = GOT_15c;

        GOT_20c:   if (Q) next = GOT_45c;
                   else if (D) next = GOT_30c;
                   else if (N) next = GOT_25c;
                   else next = GOT_20c;
```

```
GOT_25c:  if (Q) next = GOT_50c;
           else if (D) next = GOT_35c;
           else if (N) next = GOT_30c;
           else next = GOT_25c;
```

```
GOT_30c:  next = IDLE;
GOT_35c:  next = RETURN_5c;
GOT_40c:  next = RETURN_10c;
GOT_45c:  next = RETURN_15c;
GOT_50c:  next = RETURN_20c;
```

```
RETURN_20c: next = RETURN_10c;
RETURN_15c: next = RETURN_5c;
RETURN_10c: next = IDLE;
RETURN_5c:  next = IDLE;
```

```
default: next = IDLE;
endcase
end
```

Combinational output assignment

```
assign DC = (state == GOT_30c || state == GOT_35c ||
             state == GOT_40c || state == GOT_45c ||
             state == GOT_50c);
assign DN = (state == RETURN_5c);
assign DD = (state == RETURN_20c || state == RETURN_15c ||
             state == RETURN_10c);
endmodule
```

