

C H A P T E R

12

신호등 제어기의 설계

교통신호등 제어기는 FSM을 사용한 상태 천이도로 표현할 수 있다. 본 강의교재에서는 직각 교차로에서 차량은 직진만 할 수 있으며 차량이 가로 방향으로 이동하면 세로 방향 차량은 대기해야 하고, 반대로 세로 방향의 차량이 이동하면 가로 방향의 차량이 대기해야 한다. 그림 12.1은 교통신호등 제어기의 상태도이다. 교통신호등의 초기 상태는 YY이다. YY는 대기 상태이며, 대기 상태가 해제되면 신호등은 정상적으로 작동을 한다. 대기 상태의 다음 상태로는 RY 상태이다. 예를 들어, 가로 방향이 Red 상태이고 세로 방향이 Yellow 상태를 말한다. 그 다음 상태는 RG 상태이다. 이 상태는 가로 방향이 Red이고 세로 방향이 Green 상태이며, 세로 방향으로만 차량이 이동 가능한 상태이다. 이 교통신호등의 상태 천이도는 $YY \rightarrow RY \rightarrow GR \rightarrow YR \rightarrow RG \rightarrow RY$ 상태로 이루어진다.

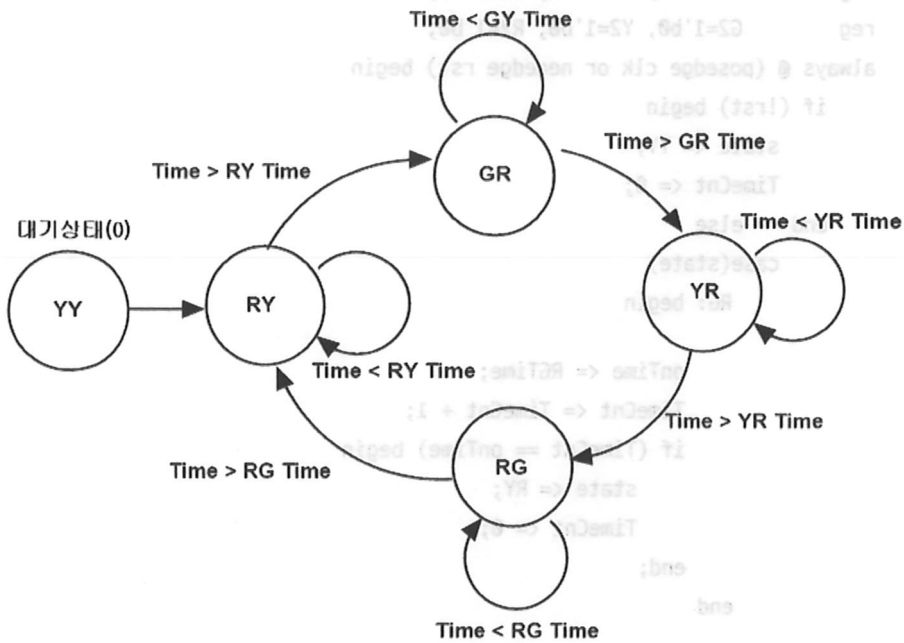



그림 12.1 교통신호등 제어기의 상태도

그림 12.2는 교통신호등 제어기의 Verilog 코드이다. 교통신호등 5개의 상태를 parameter로 지정하고 첫 번째 always 문구는 if문을 사용하여 교통신호등의 상태 천이를 표현하였고 두 번째 always 문구는 case 문을 사용하여 각 상태의 출력을 표현하였다.

 코드 12-1. 교통신호등 제어기의 Verilog Code

```

`timescale 1ns / 1ps

module traffic_light(clk, rst, G1,Y1, R1,G2, Y2,R2 );
    input      clk, rst/*, test*/;
    output     G1,Y1, R1,G2, Y2,R2;

    parameter  YY=3'b000, RY=3'b001, GR=3'b010, YR=3'b011, RG=3'b100;
    reg [2:0]   state=YY;

    parameter  RGTime=10, RYTime=3, GRTime=15, YRTime=3;

    integer     TimeCnt, onTime;

    reg         G1=1'b0, Y1=1'b0, R1=1'b0;
    reg         G2=1'b0, Y2=1'b0, R2=1'b0;
    always @ (posedge clk or negedge rst) begin
        if (!rst) begin
            state <= YY;
            TimeCnt <= 0;
        end else
            case(state)
                RG: begin
                    onTime <= RGTime;
                    TimeCnt <= TimeCnt + 1;
                    if (TimeCnt == onTime) begin
                        state <= RY;
                        TimeCnt <= 0;
                    end;
                end
                RY: begin
                    onTime <= RYTime;
                    TimeCnt <= TimeCnt + 1;
                    if (TimeCnt == onTime) begin
                        state <= GR;
                        TimeCnt <= 0;
                    end;
                end
            endcase
    end
end

```

```

GR: begin
    onTime <= GRTime;
    TimeCnt <= TimeCnt + 1;
    if (TimeCnt == onTime) begin
        state <= YR;
        TimeCnt <= 0;
    end;
end

YR: begin
    onTime <= YRTime;
    TimeCnt <= TimeCnt + 1;
    if (TimeCnt == onTime) begin
        state <= RG;
        TimeCnt <= 0;
    end;
end

YY:
    state <= RY;
endcase

end

always @ (state) begin
    case(state)
        RG:begin
            R1 <= 1'b1; Y1 <= 1'b0; G1 <= 1'b0;
            R2 <= 1'b0; Y2 <= 1'b0; G2 <= 1'b1;
        end
        RY:begin
            R1 <= 1'b1; Y1 <= 1'b0; G1 <= 1'b0;
            R2 <= 1'b0; Y2 <= 1'b1; G2 <= 1'b0;
        end
        GR:begin
            R1 <= 1'b0; Y1 <= 1'b0; G1 <= 1'b1;
            R2 <= 1'b1; Y2 <= 1'b0; G2 <= 1'b0;
        end
        YR:begin

```

```

        R1 <= 1'b0; Y1 <= 1'b1; G1 <= 1'b00;
        R2 <= 1'b1; Y2 <= 1'b0; G2 <= 1'b0;
    end
    YY:begin
        R1 <= 1'b0; Y1 <= 1'b1; G1 <= 1'b0;
        R2 <= 1'b0; Y2 <= 1'b1; G2 <= 1'b0;
    end
    default;;
endcase
end
endmodule

```

그림 12.2 교통신호등 제어기의 Verilog 코드

그림 12.3은 교통신호등 제어기의 테스트 벤치 코드이고 그림 12.4는 시뮬레이션 파형이다. 그림과 같이 교통신호등의 상태는 RY → GR → YR → RG → RY 상태로 천이된다.

코드 12-2. 교통 신호등 제어기의 테스트 벤치 코드

```

`timescale 1ns / 1ps

module tb_traffic_light;
    reg clk, rst;
    wire G1, Y1, R1, G2, Y2, R2;

    traffic_light U0 (clk, rst, G1,Y1, R1,G2, Y2,R2 );
    initial begin
        clk = 1'b0;
        rst = 1'b0;
        #100;
        rst = 1'b1;
        #700;
        $finish;
    end
    always #5 clk = ~clk;
endmodule

```

그림 12.3 교통신호등 제어기의 테스트 벤치 코드

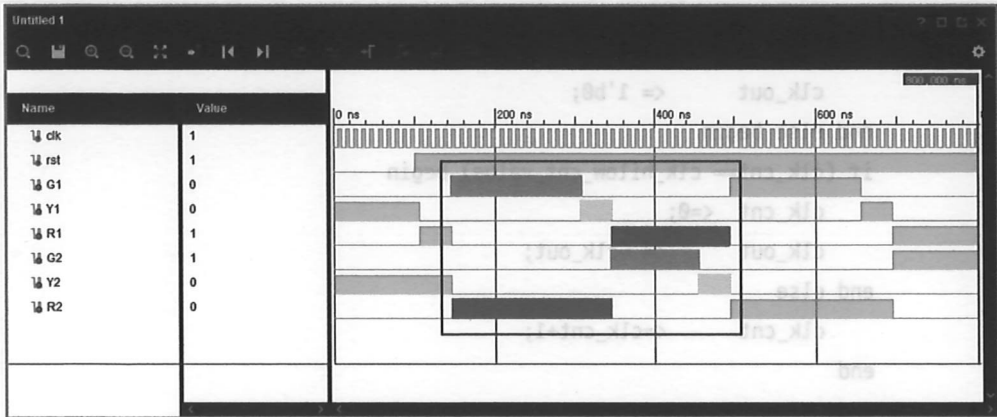


그림 12.4 교통신호등 제어기의 시뮬레이션 파형

교통신호등 제어기를 실제 보드에 다운로드 하려면 분주 회로를 사용해야 한다. 보드의 높은 주파수 때문에 계수 상태를 확인하기 어렵다. 분주 회로를 사용하여 교통신호등 제어기에 들어가는 입력 clk은 1Hz의 주파수를 가진다. 1Hz의 주파수를 입력으로 받는 교통신호등 제어기를 FSK III 보드에 다운로드 하면 신호등 상태를 쉽게 확인할 수 있다. 그림 12.5는 분주 회로의 Verilog 코드이고 그림 12.6은 교통신호등 제어기의 최상위 모듈이다.

코드 12-3. 분주 회로의 Verilog 코드

```
`timescale 1ns / 1ps
```

```
module frq_clk_1hz (clk, reset, clk_out);
```

```
    parameter main_clk=100000000, output_freq=1;
```

```
    parameter clk_hilow_cnt_value=((main_clk/output_freq)/2)-1;
```

```
    input      clk;
```

```
    input      reset;
```

```
    output reg clk_out;
```

```
    integer    clk_cnt;
```

```
    always@(negedge reset, posedge clk) begin
```

```
        if (reset == 1'b0) begin
```

```

        clk_cnt      <= 0;
        clk_out      <= 1'b0;
    end else begin
        if (clk_cnt== clk_hilow_cnt_value) begin
            clk_cnt <=0;
            clk_out      <= ~clk_out;
        end else
            clk_cnt      <=clk_cnt+1;
        end
    end
endmodule

```

그림 12.5 분주 회로의 Verilog 코드

코드 12-4. 교통신호등 제어기의 최상위 모듈

```

module Top_module(clk, rst, G1,Y1, R1,G2, Y2, R2);
    input      clk, rst;
    output     G1,Y1, R1,G2, Y2,R2;
    wire clk_1hz;
    frq_clk_1hz u0 (clk, rst, clk_1hz);
    traffic_light u1 (clk_1hz, rst, G1,Y1, R1,G2, Y2,R2 );
endmodule

```

그림 12.6 교통신호등 제어기의 최상위 모듈

시뮬레이션이 끝나면 실습 보드를 활용하여 동작 검증을 시작한다. 실습 보드의 PMOD_JA[0]-[2], PMOD_JA[4]-[6] 번 핀에 출력 LED를 할당한다. 표 12.1은 입력 신호와 출력 신호의 핀 할당 정보이다. 아래 정보를 가지고 Verilog 코드를 합성 후에 xdc 파일을 작성하여 핀 정보를 설정해준다. 핀 설정이 끝나면 Implementation과 Bitstream 파일을 생성하고, 실습 보드와 컴퓨터를 연결하여 (name).bit 파일을 실습 보드에 로딩한다.

비트 파일을 로딩하고 출력 LED를 사용하여 동작 검증을 한다. 그림 12.7은 외부 LED와 실습 보드의 JA 핀에 연결하여 교통신호등을 구현한 화면이다.

표 12.1 신호 이름과 할당된 입/출력 핀

신호 이름	입/출력	장치 종류	키트 이름	핀 번호
clk	입력	clock input	clock	R4
rst	입력	PUSH 스위치	reset_sw	U7
G1	출력	LED	PMOD_JA	C13
Y1	출력	LED	PMOD_JA	B13
R1	출력	LED	PMOD_JA	A15
G2	출력	LED	PMOD_JA	A13
Y2	출력	LED	PMOD_JA	A14
R2	출력	LED	PMOD_JA	B17

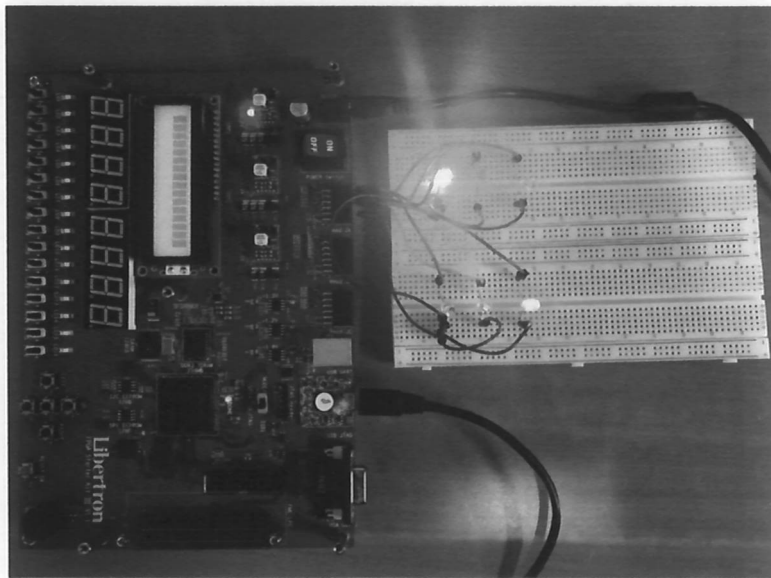
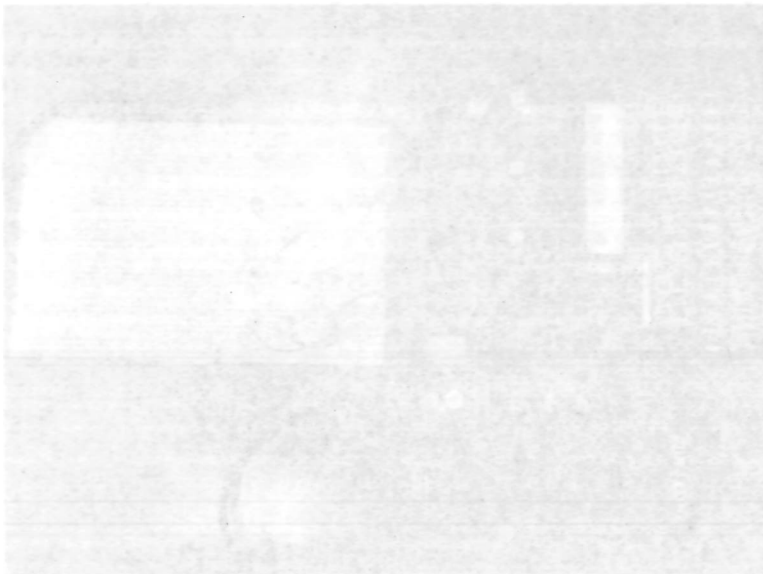


그림 12.7 교통신호등 제어기의 구현

■ 교통신호등 제어기의 실습 과정

- ① Vivado 소프트웨어를 실행시키고 project 파일을 생성한다. - (2장 2.1절 참조)
- ② Verilog 코드를 작성하고 시뮬레이션을 진행한다. - (2장 2.2절, 2.3절 참조)
- ③ 시뮬레이션 파형의 작동을 확인 후에 합성을 진행한다. - (2장 2.4절 참조)
- ④ 합성된 디자인을 열고 핀 정보를 할당하고 xdc 파일을 생성한다. - (2장 2.6절 참조)
- ⑤ Implementation과 Bitstream 과정을 진행하여 bit 파일을 생성한다. - (2장 2.7절 참조)
- ⑥ 실습 보드에 bit 파일을 로딩한다. - (2장 2.7절 참조)
- ⑦ 보드의 입력과 출력장치를 활용하여 동작 검증을 한다. 보드의 JA 핀에 세 가지 색상의 LED를 연결하여 상태를 관찰하고 동작 검증을 한다.



본도: 교통신호등 제어기 실습 보드