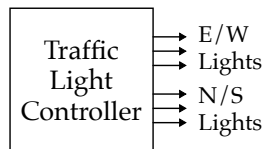


2.6. Factoring FSMs

- Design an FSM for a traffic light controller

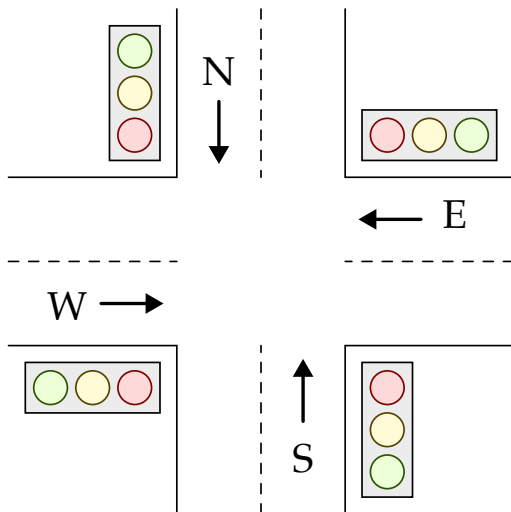
- Four-way intersection with traffic lights

- Opposing lanes sequence together
 - Turn on green light for 20 seconds
 - Turn on yellow light for 5 seconds
 - Turn on red light for 25 seconds



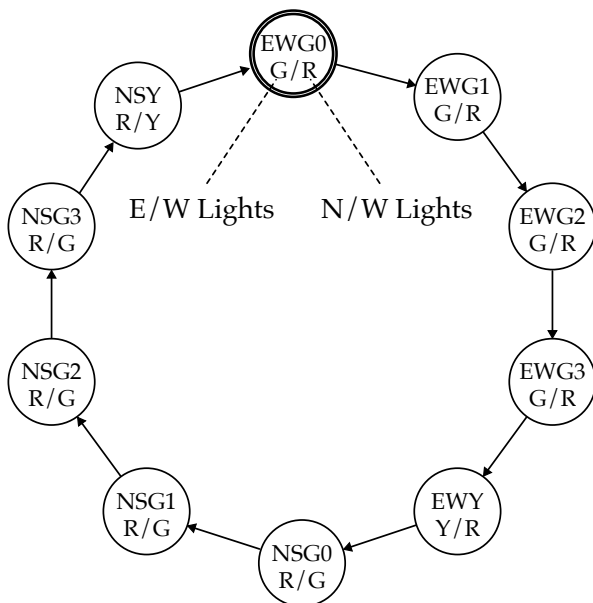
- Four scenarios

- E/W green and N/S red for 20 seconds
- E/W yellow and N/S red for 5 seconds
- E/W red and N/S green for 20 seconds
- E/W red and N/S yellow for 5 seconds

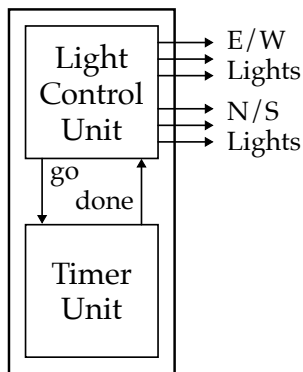


- Requires an FSM with 10 states

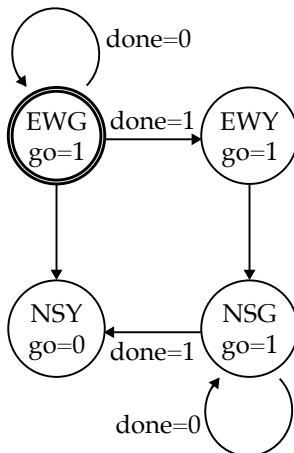
- EWG0: E/W green and N/S red1 for 5 seconds
- EWG1: E/W green and N/S red2 for 5 seconds
- EWG2: E/W green and N/S red3 for 5 seconds
- EWG2: E/W green and N/S red4 for 5 seconds
- EWY: E/W yellow and N/S red for 5 seconds
- NSG0: E/W red and N/S green1 for 5 seconds
- NSG1: E/W red and N/S green2 for 5 seconds
- NSG2: E/W red and N/S green3 for 5 seconds
- NSG2: E/W red and N/S green4 for 5 seconds
- NSY: E/W red and N/S yellow for 5 seconds



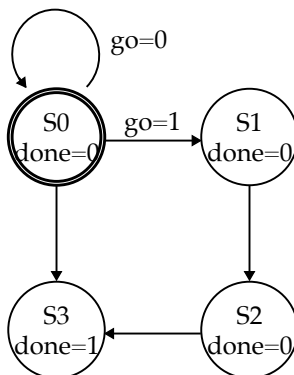
- Factoring FSMs involves using the *modularity* design principle to decompose an FSM into multiple communicating FSMs
 - Simplifies large FSMs
 - May result in fewer states



Light Control Unit FSM



Timer Unit FSM



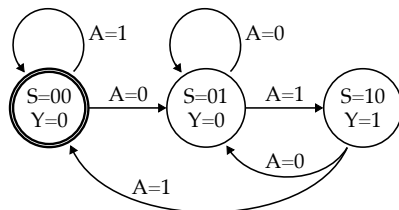
3. Verilog Modeling

3.1. Gate-Level Modeling of FSMs

```

1 module MooreDetect01_GL
2 (
3   input wire clk,
4   input wire rst,
5   input wire a,
6   output wire y
7 );
8   // Sequential: state flip-flops
9
10  wire s0, s0_next;
11
12  DFFR_GL s0_dffr
13  (
14    .clk (clk),
15    .rst (rst),
16    .d   (s0_next),
17    .q   (s0)
18  );
19
20  wire s1, s1_next;
21
22  DFFR_GL s1_dffr
23  (
24    .clk (clk),
25    .rst (rst),
26    .d   (s1_next),
27    .q   (s1)
28  );
29
30  // Combinational: next state logic
31
32  not( s0_next, s0 );
33  and( s1_next, s0, a );
34
35  // Combinational: output logic
36
37  assign y = s1;
38
39  endmodule

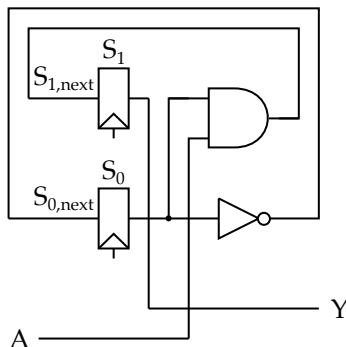
```



$$S_{1,next} = S_0 A$$

$$S_{0,next} = \overline{A}$$

$$Y = S_1$$

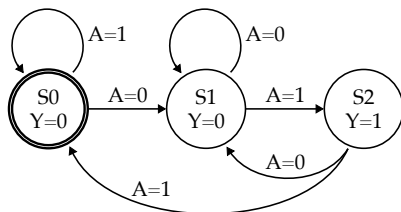


3.2. RTL Modeling of FSMs

```

1 module MooreDetect01_RTL
2 (
3   input logic clk,
4   input logic rst,
5   input logic a,
6   output logic y
7 );
8   localparam S0 = 2'b00;
9   localparam S1 = 2'b01;
10  localparam S2 = 2'b10;
11
12  logic [1:0] state, state_next;
13  Register_2b_RTL state_reg
14  (
15    .clk (clk),
16    .rst (rst),
17    .d   (state_next),
18    .q   (state)
19  );
20
21  always_comb begin
22    case ( { state, a } )
23      { S0, 0 } : state_next = S1;
24      { S0, 1 } : state_next = S0;
25      { S1, 0 } : state_next = S1;
26      { S1, 1 } : state_next = S2;
27      { S2, 0 } : state_next = S1;
28      { S2, 1 } : state_next = S0;
29      default   : state_next = 'x;
30    endcase
31  end
32
33  always_comb begin
34    case ( state )
35      S0      : y = 0;
36      S1      : y = 0;
37      S2      : y = 1;
38      default : y = 'x;
39    endcase
40  end
41
42 endmodule

```

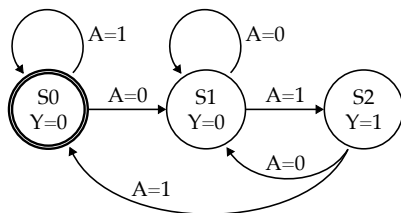


S	A	S _{next}	Y
S0	0	S1	0
S0	1	S0	0
S1	0	S1	0
S1	1	S2	0
S2	0	S1	1
S2	1	S0	1

```

1 module MooreDetect01_RTL
2 (
3   input logic clk,
4   input logic rst,
5   input logic a,
6   output logic y
7 );
8   localparam S0 = 2'b00;
9   localparam S1 = 2'b01;
10  localparam S2 = 2'b10;
11
12  // Sequential: state register
13  logic [1:0] state, state_next;
14  Register_2b_RTL state_reg
15  (
16    .clk (clk),
17    .rst (rst),
18    .d   (state_next),
19    .q   (state)
20  );
21
22  // Combinational: next state logic
23  always_comb begin
24    case ( state )
25      S0 : state_next = ( a ) ? S0 : S1;
26      S1 : state_next = ( a ) ? S2 : S1;
27      S2 : state_next = ( a ) ? S0 : S1;
28      default : state_next = 'x;
29    endcase
30  end
31
32  // Combinational: output logic
33  always_comb begin
34    case ( state )
35      S0 : y = 0;
36      S1 : y = 0;
37      S2 : y = 1;
38      default : y = 'x;
39    endcase
40  end
41
42 endmodule

```



S	A	S _{next}	Y
S0	0	S1	0
S0	1	S0	0
S1	0	S1	0
S1	1	S2	0
S2	0	S1	1
S2	1	S0	1