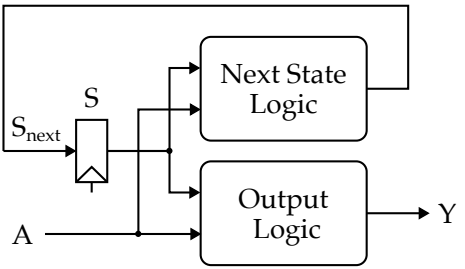
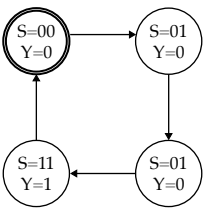
2. Finite-State Machines



- *Finite state machines* (FSM) are abstractions for sequential gate-level networks
  - Finite number of inputs (A)
  - Finite number of outputs (Y)
  - Finite number of states (S)
  - Designated reset state
  - FSM next-state equation ($S_{next}(S, A)$)
  - FSM output equation ($Y(S)$ or $Y(S, A)$)
- Two kinds of FSMs
  - Moore FSMs: Outputs only depend on current state ($Y(S)$)
  - Mealy FSMs: Outputs depend on current state and inputs ($Y(S, A)$)
- FSMs can be represented using:
  - FSM diagrams
  - FSM next state and output tables (i.e., truth tables)
  - FSM next state and output equations

2.1. FSM Abstraction Triangle

Diagrams



Next State & Output Tables

S_1	S_0	$S_{1,next}$	$S_{0,next}$	Y
0	0	0	1	0
0	1	1	0	0
1	0	1	1	0
1	1	0	0	1

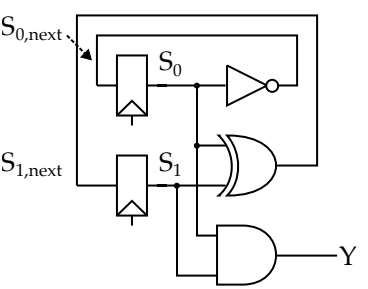
Next State & Output Equations

$$S_{0,next} = \overline{S_1}\,\overline{S_0} + S_1\overline{S_0}$$

$$S_{1,next} = \overline{S_1}S_0 + S_1\overline{S_0}$$

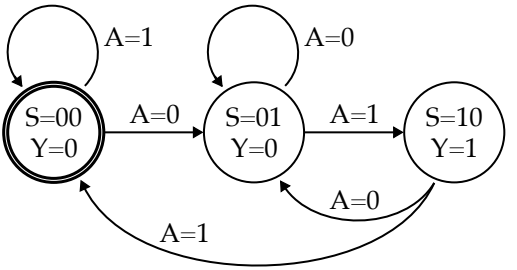
$$Y = S_1 S_0$$

Gate-Level Networks



2.2. Moore FSMs

FSM Diagram



FSM Next State and Output Table

S_1	S_0	A	$S_{1,next}$	$S_{0,next}$	Y
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			

FSM Implicit-Clock Simulation Table

cycle	S	A	S_{next}	Y
0	00	0	01	0
1		0		
2		1		
3		0		
4		0		
5		0		
6		1		
7		1		
8		0		
9		1		
10		0		

FSM Next State and Output Equations

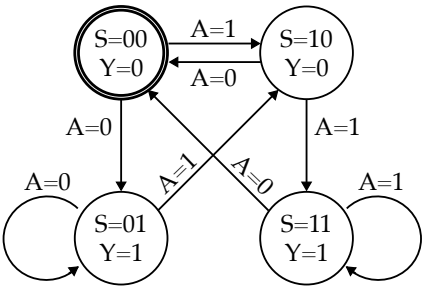
A \ S_1S_0	00	01	11	10
0				
1				

A \ S_1S_0	00	01	11	10
0				
1				

A \ S_1S_0	00	01	11	10
0				
1				

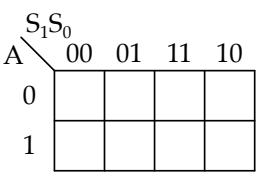
FSM Sequential Gate-Level Network

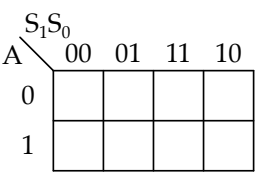
Implement the following FSM at the gate level.

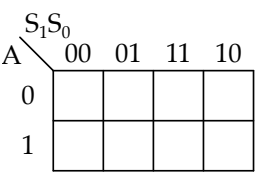


What does this FSM do?

S_1	S_0	A	$S_{1,next}$	$S_{0,next}$	Y
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

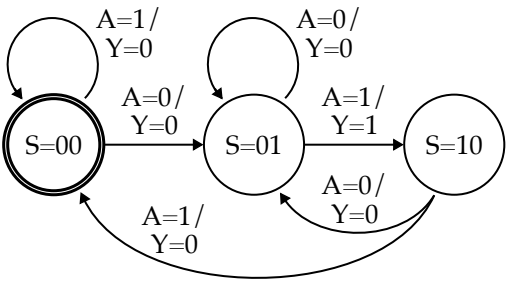






2.3. Mealy FSMs

FSM Diagram



FSM Next State and Output Table

S_1	S_0	A	$S_{1,next}$	$S_{0,next}$	Y
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			

FSM Implicit-Clock Simulation Table

cycle	S	A	S_{next}	Y
0	00	0	01	0
1		0		
2		1		
3		0		
4		0		
5		0		
6		1		
7		1		
8		0		
9		1		
10		0		

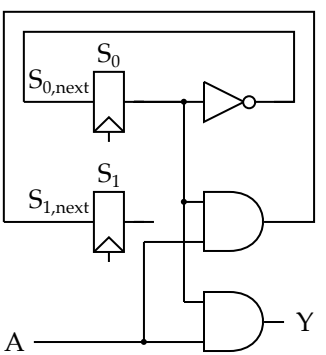
FSM Next State and Output Equations

A \ S_1S_0	00	01	11	10
0				
1				

A \ S_1S_0	00	01	11	10
0				
1				

A \ S_1S_0	00	01	11	10
0				
1				

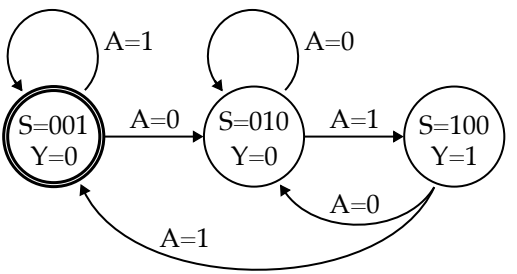
FSM Sequential Gate-Level Network



Optimized FSM Diagram

2.4. FSM State Encoding

- Often there is freedom in choosing how to encode the states
- Different encodings result in different FSM next state and output equations (and thus produce different gate-level implementations)
- Two common FSM encodings
  - *Binary encoding* uses a binary to encode each state
  - *One-hot encoding* uses a separate bit to encode each state



S_2	S_1	S_0	A	$S_{1,next}$	$S_{1,next}$	$S_{0,next}$	Y
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
1	0	0	0				
1	0	0	1				

2.5. Designing FSMs

- Step 1. Understand problem statement, determine inputs/outputs
- Step 2. Choose Moore vs. Mealy, identify states, create state diagram
- Step 3. Determine state encoding
- Step 4. Create FSM next state and output tables
- Step 5. Determine FSM next state and output equations

Design an FSM with one input (A) and two outputs (Y,Z) which monitors the input and sets Y to one when it has seen two or more consecutive ones and sets Z to one when it has seen three or more consecutive ones.

Step 1. Understand problem statement, determine inputs/outputs

Step 2. Choose Moore vs. Mealy, identify states, create state diagram

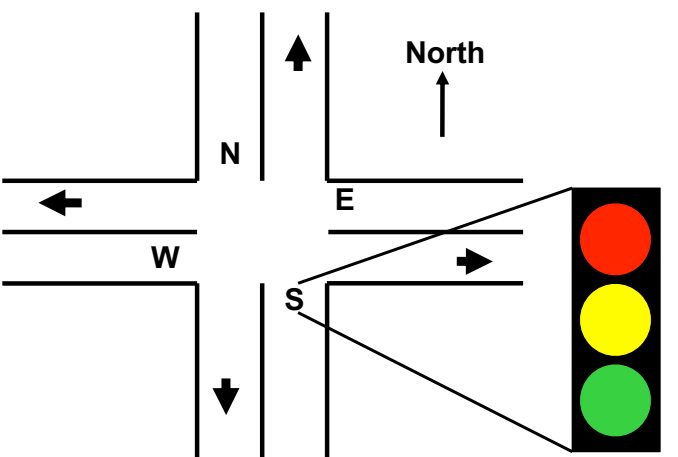
Step 3. Determine state encoding

Step 4. Create FSM next state and output tables

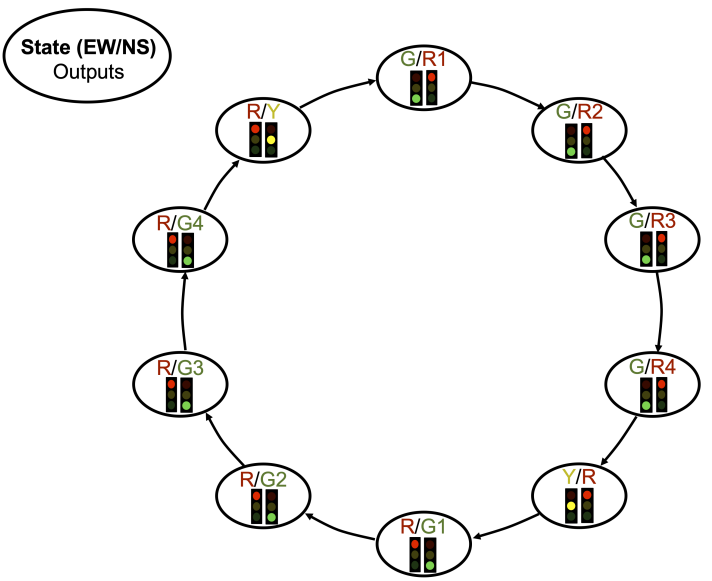
Step 3. Determine FSM next state and output equations

2.6. Factoring FSMs

- Design a 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

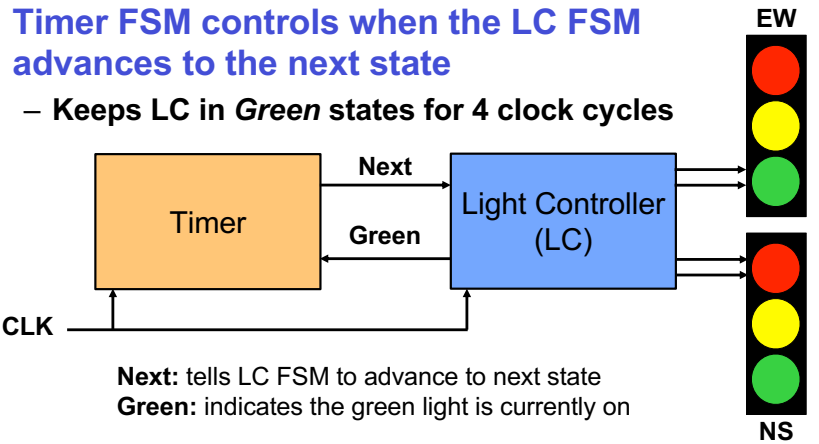


- Requires an FSM with 10 states
  - E/W green and N/S red1 for 5 seconds
  - E/W green and N/S red2 for 5 seconds
  - E/W green and N/S red3 for 5 seconds
  - E/W green and N/S red4 for 5 seconds
  - E/W yellow and N/S red for 5 seconds
  - E/W red and N/S green1 for 5 seconds
  - E/W red and N/S green2 for 5 seconds
  - E/W red and N/S green3 for 5 seconds
  - E/W red and N/S green4 for 5 seconds
  - E/W red and N/S yellow for 5 seconds



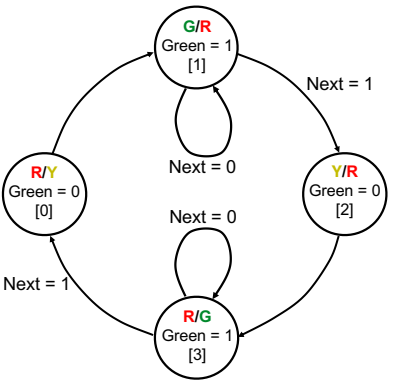
- Factoring FSMs involves breaking an FSM into multiple communicating FSMs
  - Simplifies large FSMs
  - May result in fewer states

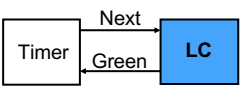
- **Light Controller (LC) FSM has 4 states**
  - **G/R, Y/R, R/G, R/Y**
- **Timer FSM controls when the LC FSM advances to the next state**
  - **Keeps LC in *Green* states for 4 clock cycles**



Next: tells LC FSM to advance to next state
Green: indicates the green light is currently on

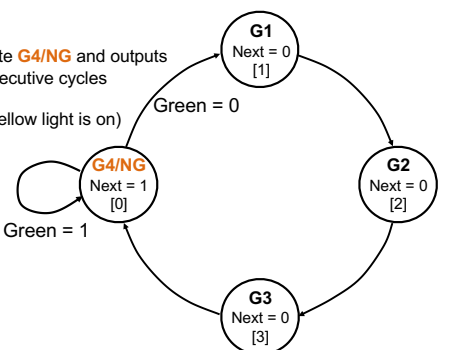
Light Controller (LC) FSM

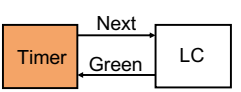




Timer FSM

Timer stays in state **G4/NG** and outputs Next=1 for 2 consecutive cycles
G4: Last Green
NG: Not Green (yellow light is on)

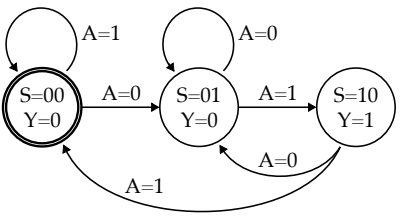




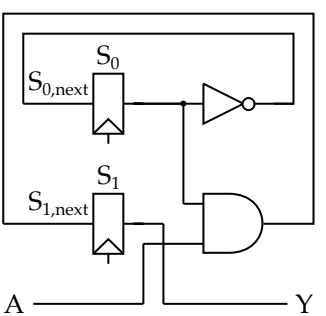
3. Verilog Modeling

3.1. Gate-Level Modeling of FSMs

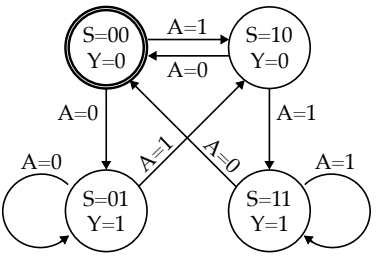
```
1  module MooreDetect01_GL
2  (
3    input  wire clk,
4    input  wire rst,
5    input  wire a,
6    output wire y
7  );
8    // Sequential: state flip-flops
9
10   wire s0, s0_next;
11
12   DFFR_GL s0_dffr
13   (
14     .clk (clk),
15     .rst (rst),
16     .d   (s0_next),
17     .q   (s0)
18   );
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
```



$$S_{1,next} = S_0 A$$
$$S_{0,next} = \overline{A}$$
$$Y = S_1$$



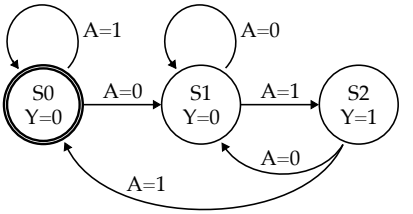
Implement 00/11 detector in Verilog using gate-level modeling.

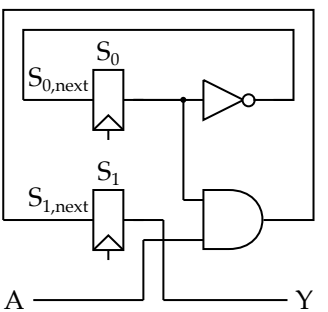



```
module MooreDetect00or11_GL
(
  input  wire clk,
  input  wire rst,
  input  wire a,
  output wire y
);
  // Sequential: state flip-flops

  wire s0, s0_next;
  DFFR_GL s0_dffr( .clk(clk), .rst(rst), .d(s0_next), .q(s0) );

  wire s1, s1_next;
  DFFR_GL s1_dffr( .clk(clk), .rst(rst), .d(s1_next), .q(s1) );
```

```
endmodule
```

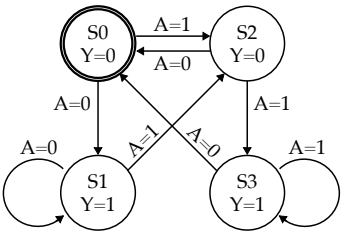
3.2. RTL Modeling of FSMs






```
module MooreDetect01_RTL
(
  input  logic clk,
  input  logic rst,
  input  logic a,
  output logic y
);
  localparam STATE_S0 = 2'b00;
  localparam STATE_S1 = 2'b01;
  localparam STATE_S2 = 2'b10;

  // Sequential: state register
  logic [1:0] state, state_next;
  Register_2b_RTL state_reg
  (
    .clk (clk),
    .rst (rst),
    .d   (state_next),
    .q   (state)
  );

  // Combinational: next state logic
  always_comb begin
    case ( state )
      STATE_S0 : state_next = ( a ) ? STATE_S0 : STATE_S1;
      STATE_S1 : state_next = ( a ) ? STATE_S2 : STATE_S1;
      STATE_S2 : state_next = ( a ) ? STATE_S1 : STATE_S0;
      default  : state_next = 'x;
    endcase
  end

  // Combinational: output logic
  always_comb begin
    case ( state )
      STATE_S0 : begin y = 0; end
      STATE_S1 : begin y = 0; end
      STATE_S2 : begin y = 1; end
      default  : state_next = 'x;
    endcase
  end

endmodule
```

Implement 00/11 detector in Verilog using RTL modeling.




```
module MooreDetect00or11_RTL
(
  input logic clk, input logic rst,
  input logic a, output logic y
);
  // Sequential: state register
  localparam STATE_S0 = 2'b00;
  localparam STATE_S1 = 2'b01;
  localparam STATE_S2 = 2'b10;
  localparam STATE_S3 = 2'b11;
  logic [1:0] state, state_next;
  Register_2b_RTL state_reg( .clk(clk), .rst(rst),
                             .d(state_next), .q(state) );
```

```
endmodule
```