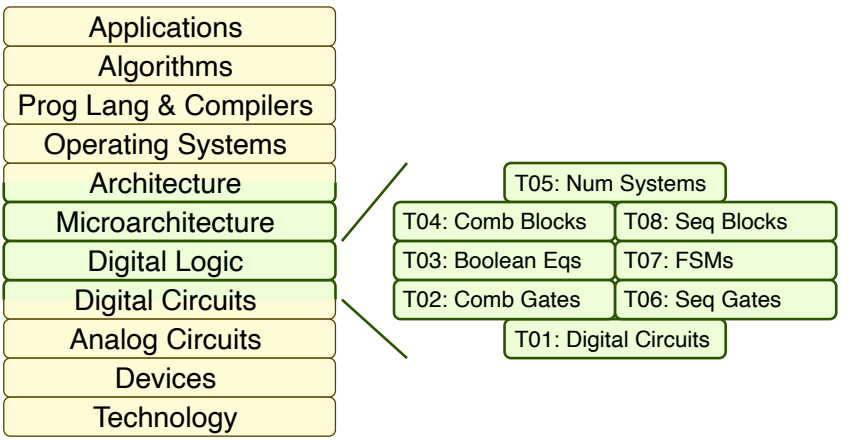
ECE 2300 Digital Logic and Computer Organization Fall 2024

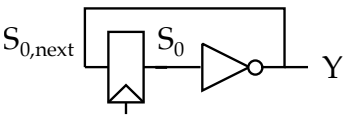
Topic 7: Finite State Machines

School of Electrical and Computer Engineering
Cornell University

revision: 2024-10-10-10-12

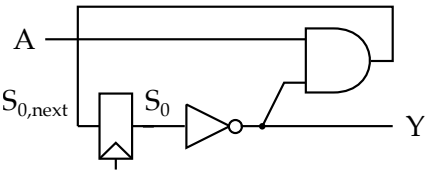


1. From Logic Gates to Finite-State Machines



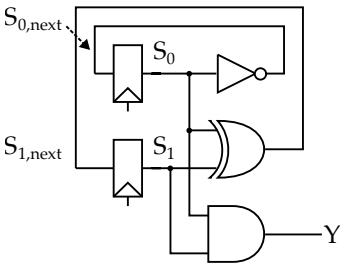
Assume all flip-flops are reset to zero.

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



Assume all flip-flops are reset to zero.

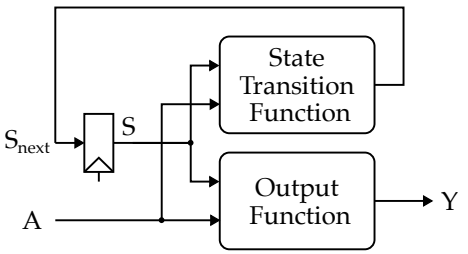
A	S_0	$S_{0,next}$	Y
0	0		
0	1		
1	0		
1	1		



Assume all flip-flops are reset to zero.

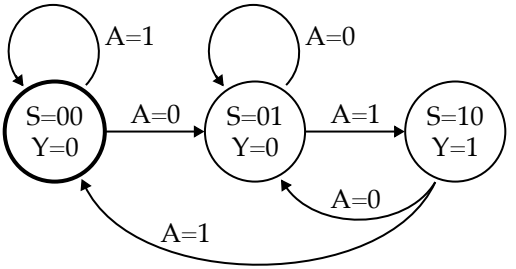
S_1	S_0	$S_{1,next}$	$S_{0,next}$	Y
0	0			
0	1			
1	0			
1	1			

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 transition function ($S_{next}(S, A)$)
  - FSM output function ($Y(S)$ 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 transition and output functions
  - FSM transition and output tables
  - FSM diagrams

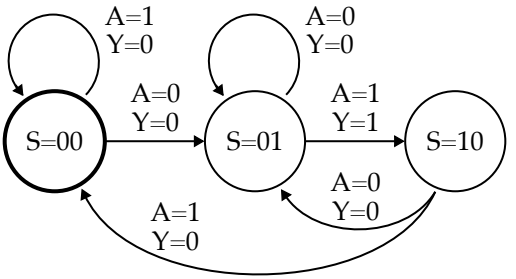
2.1. Moore FSMs



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

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			

2.2. Mealy FSMs

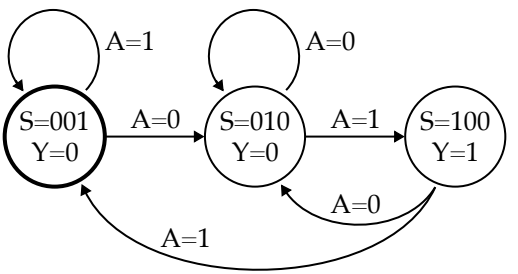


cycle	0	1	2	3	4	5	6	7	8	9	10
A	0	0	1	0	0	0	1	1	0	1	0
S	00										
S_{next}	01										
Y	0										

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			

2.3. FSM State Encoding

- Often there is freedom in choosing how to encode the states
- Different encodings result in different FSM transition and output functions (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				

2.4. 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 state transition and output tables
- Step 5. Determine FSM state transition and output functions

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

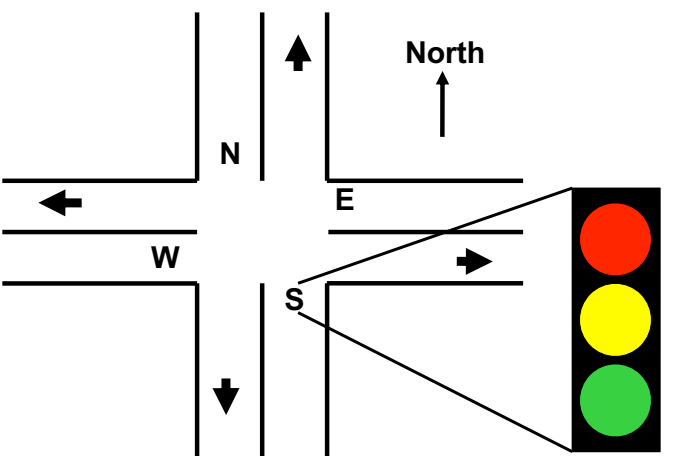
Step 3. Determine state encoding

Step 4. Create FSM state transition and output tables

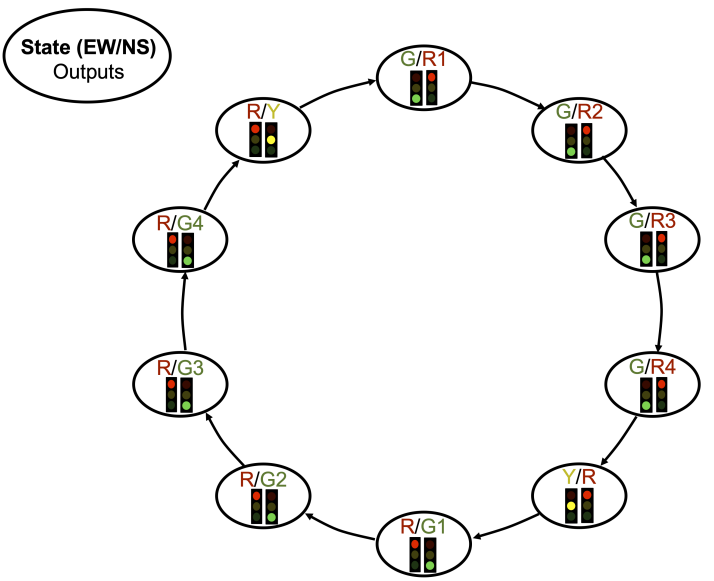
Step 3. Determine FSM state transition and output functions

2.5. 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 yellow 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 20 seconds

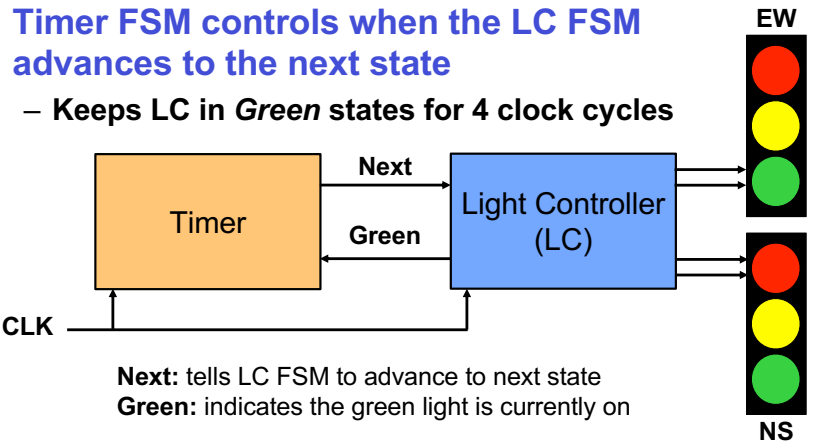


- Requires an FSM with 10 states
  - E/W green and N/S red1 for 5 seconds
  - E/W green and N/S red2 for 5 seconds
  - E/W green and N/S red3 for 5 seconds
  - E/W green and N/S red4 for 5 seconds
  - E/W yellow and N/S red for 5 seconds
  - E/W red and N/S green1 for 5 seconds
  - E/W red and N/S green2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



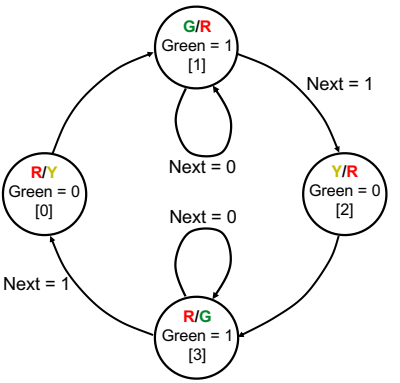
- Factoring FSMs involves breaking an FSM into multiple communicating FSMs
  - Simplifies large FSMs
  - May result in fewer states

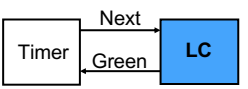
- **Light Controller (LC) FSM has 4 states**
  - **G/R, Y/R, R/G, R/Y**
- **Timer FSM controls when the LC FSM advances to the next state**
  - **Keeps LC in *Green* states for 4 clock cycles**



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

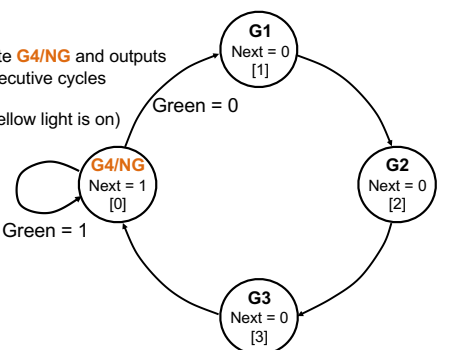
Light Controller (LC) FSM

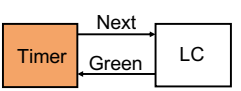




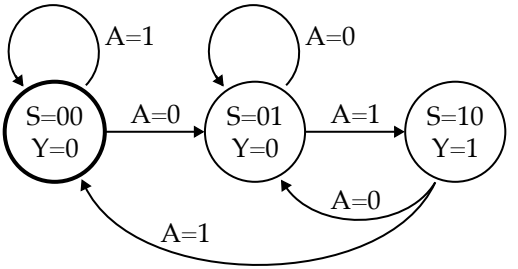
Timer FSM

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

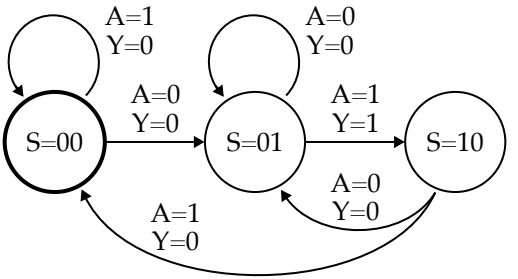




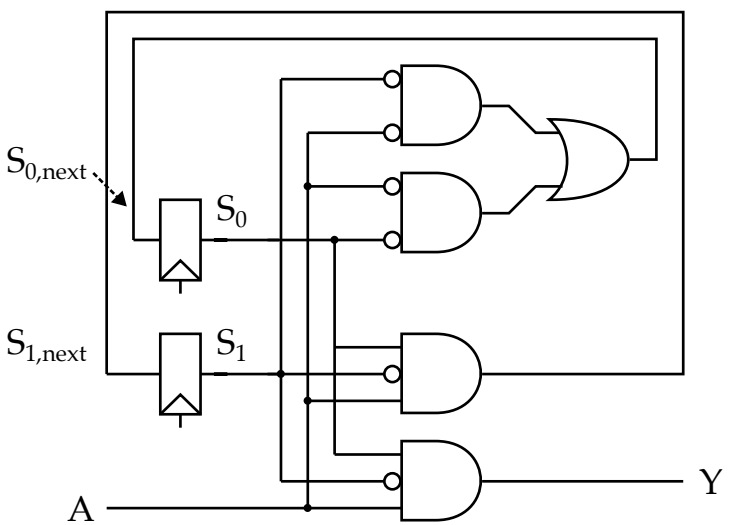
3. From Finite-State Machines to Logic Gates



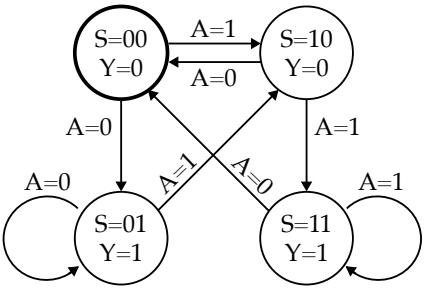
$$
\begin{aligned}
S_{1,next} &= \overline{S_1} S_0 A \\
S_{0,next} &= \overline{S_1}\,\overline{A} + \overline{S_0}\,\overline{A} \\
Y &= S_1 \overline{S_0}
\end{aligned}
$$



$$
\begin{aligned}
S_{1,next} &= \overline{S_1} S_0 A \\
S_{0,next} &= \overline{S_1}\,\overline{A} + \overline{S_0}\,\overline{A} \\
Y &= \overline{S_1} S_0 A
\end{aligned}
$$

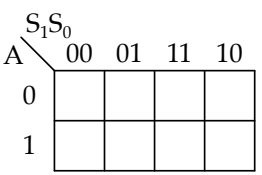


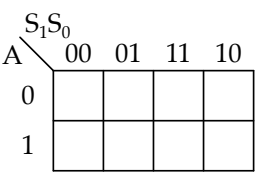
Implement the following FSM at the gate level.

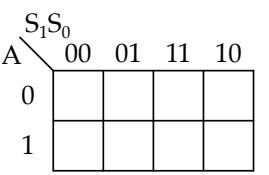


What does this FSM do?

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			



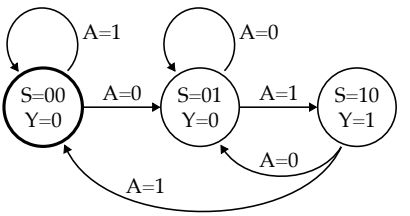




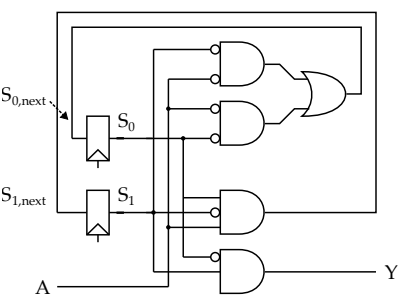
4. Modeling Finite-State Machines with Verilog

4.1. Gate-Level Modeling of FSMs

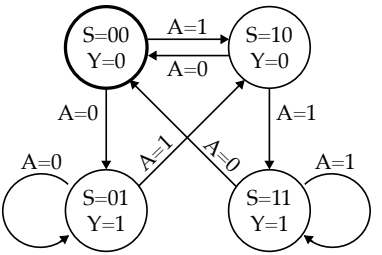
```
1  module Detect01_GL
2  (
3    wire clk,
4    wire rst,
5    wire a
6  );
7    // Sequential: state flip-flops
8
9    wire s0, s0_next;
10
11   DFFR_GL s0_dffr
12   (
13     .clk (clk),
14     .rst (rst),
15     .q   (s0_next),
16     .d   (s0)
17   );
18
19   wire s1, s1_next;
20
21   DFFR_GL s1_dffr
22   (
23     .clk (clk),
24     .rst (rst),
25     .q   (s1_next),
26     .d   (s1)
27   );
28
29   // Combinational: state transition
30
31   assign s0_next = ( ~s1 & ~a ) | ( ~s0 & ~a );
32   assign s1_next = ~s1 & s0 & a;
33
34   // Combinational: output logic
35
36   assign y = s1 & ~s0;
37
38 endmodule
```



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



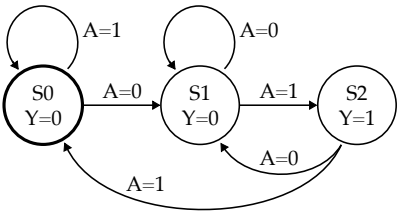
Implement 00/11 detector in Verilog using gate-level modeling.



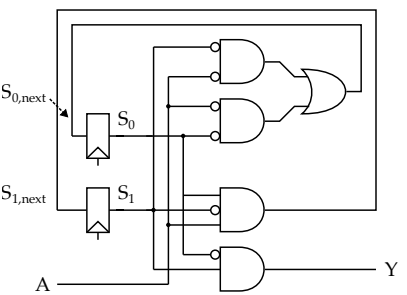

```
module Detect00or11_GL
(
  wire clk,
  wire rst,
  wire a
);
  // Sequential: state flip-flops

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

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

```
endmodule
```

4.2. RTL Modeling of FSMs



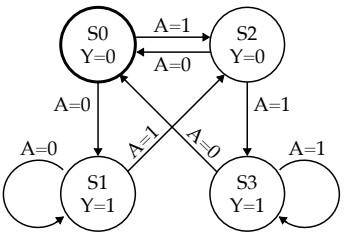
$$
\begin{aligned}
S_{1,next} &= \overline{S_1} S_0 A \\
S_{0,next} &= \overline{S_1}\,\overline{A} + \overline{S_0}\,\overline{A} \\
Y &= S_1 \overline{S_0}
\end{aligned}
$$




```
module Detect01_RTL
(
  logic clk,
  logic rst,
  logic a
);
  // Sequential: state register

  localparam STATE_S0 = 2'b00;
  localparam STATE_S1 = 2'b01;
  localparam STATE_S1 = 2'b10;

  logic state, state_next;

  Register_2b_RTL state_reg
  (
    .clk (clk),
    .rst (rst),
    .q   (state_next),
    .d   (state)
  );

  // Combinational: state transition
  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;
    endcase
  end

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

endmodule
```

Implement 00/11 detector in Verilog using RTL modeling.

```
module Detect00or11_RTL
(
  logic clk, logic rst, logic a
);
  // Sequential: state register
  localparam STATE_S0 = 2'b00;
  localparam STATE_S1 = 2'b01;
  localparam STATE_S1 = 2'b10;
  localparam STATE_S1 = 2'b11;
  logic state, state_next;
  Register_2b_RTL state_reg( .clk(clk), .rst(rst),
                             .q(state_next), .d(state) );
```




```
endmodule
```