3. Verilog Modeling

3.1. Gate-Level Modeling of FSMs

```
module MooreDetect01_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)
  );

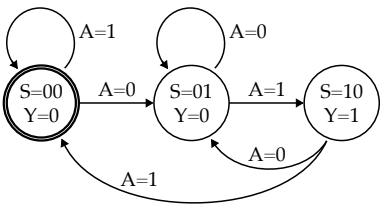
  // Combinational: next state logic

  not( s0_next, s0 );
  and( s1_next, s0, a );

  // Combinational: output logic

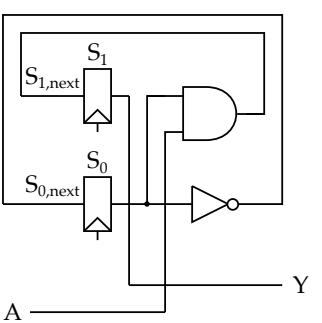
  assign y = s1;

endmodule
```



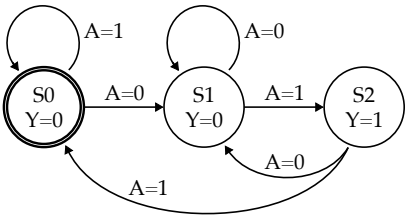
S_1	S_0	A	$S_{1,next}$	$S_{0,next}$	Y
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	0	0	1

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



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, 1'b0 } : state_next = S1;
24       { S0, 1'b1 } : state_next = S0;
25       { S1, 1'b0 } : state_next = S1;
26       { S1, 1'b1 } : state_next = S2;
27       { S2, 1'b0 } : state_next = S1;
28       { S2, 1'b1 } : 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

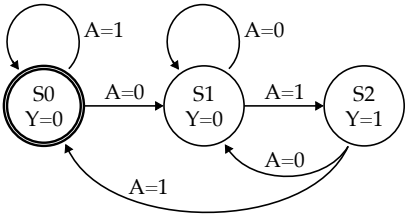
```
module MooreDetect01_RTL
(
  input  logic clk,
  input  logic rst,
  input  logic a,
  output logic y
);
  localparam S0 = 2'b00;
  localparam S1 = 2'b01;
  localparam S2 = 2'b10;

  // 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 )
      S0 : state_next = ( a ) ? S0 : S1;
      S1 : state_next = ( a ) ? S2 : S1;
      S2 : state_next = ( a ) ? S0 : S1;
      default : state_next = 'x;
    endcase
  end

  // Combinational: output logic
  always_comb begin
    case ( state )
      S0      : y = 0;
      S1      : y = 0;
      S2      : y = 1;
      default : y = 'x;
    endcase
  end

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

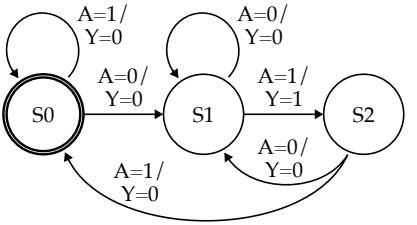
```
module MealyDetect01_RTL
(
  input  logic clk,
  input  logic rst,
  input  logic a,
  output logic y
);
  localparam S0 = 2'b00;
  localparam S1 = 2'b01;
  localparam S2 = 2'b10;

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

  always_comb begin
    case ( { state, a } )
      { S0, 1'b0 } : state_next = S1;
      { S0, 1'b1 } : state_next = S0;
      { S1, 1'b0 } : state_next = S1;
      { S1, 1'b1 } : state_next = S2;
      { S2, 1'b0 } : state_next = S1;
      { S2, 1'b1 } : state_next = S0;
      default      : state_next = 'x;
    endcase
  end

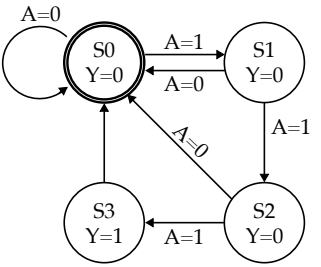
  always_comb begin
    casez ( { state, a } )
      { S0, 1'b? } : y = 0;
      { S1, 1'b0 } : y = 0;
      { S1, 1'b1 } : y = 1;
      { S2, 1'b? } : y = 0;
      default      : y = 'x;
    endcase
  end

endmodule
```



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

```
1  module CounterFSM_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   localparam S3 = 2'b11;
12
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   always_comb begin
23     casez ( { state, a } )
24       { S0, 1'b0 } : state_next = S0;
25       { S0, 1'b1 } : state_next = S1;
26       { S1, 1'b0 } : state_next = S0;
27       { S1, 1'b1 } : state_next = S2;
28       { S2, 1'b0 } : state_next = S0;
29       { S2, 1'b1 } : state_next = S3;
30       { S3, 1'b? } : state_next = S0;
31       default      : state_next = 'x;
32     endcase
33   end
34
35   always_comb begin
36     case ( state )
37       S0      : y = 0;
38       S1      : y = 0;
39       S2      : y = 0;
40       S3      : y = 1;
41       default : y = 'x;
42     endcase
43   end
44 endmodule
```



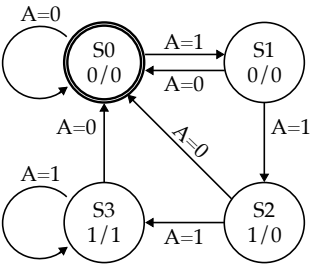
S	A	S_{next}	Y
S0	0	S0	0
S0	1	S1	0
S1	0	S0	0
S1	1	S2	1
S2	0	S0	0
S2	1	S3	0
S3	X	S0	1

```
module MooreDetect11or111_RTL
(
  input  logic clk,
  input  logic rst,
  input  logic a,
  output logic y,
  output logic z
);
  localparam S0 = 2'b00;
  localparam S1 = 2'b01;
  localparam S2 = 2'b10;
  localparam S3 = 2'b11;

  // ... state register ...

  always_comb begin
    case ( { state, a } )
      { S0, 1'b0 } : state_next = S0;
      { S0, 1'b1 } : state_next = S1;
      { S1, 1'b0 } : state_next = S0;
      { S1, 1'b1 } : state_next = S2;
      { S2, 1'b0 } : state_next = S0;
      { S2, 1'b1 } : state_next = S3;
      { S3, 1'b0 } : state_next = S0;
      { S3, 1'b1 } : state_next = S3;
      default      : state_next = 'x;
    endcase
  end

  task automatic set_outputs
  (
    input logic y_,
    input logic z_
  );
    y = y_;
    z = z_;
  endtask
```



S	A	S_{next}	Y	Z
S0	0	S0	0	0
S0	1	S1	0	0
S1	0	S0	0	0
S1	1	S2	0	0
S2	0	S0	1	0
S2	1	S3	1	0
S3	0	S0	1	1
S3	1	S3	1	1

```
  always_comb begin
    case ( state )
      //                        y   z
      S0      : set_outputs(  0,  0 );
      S1      : set_outputs(  0,  0 );
      S2      : set_outputs(  1,  0 );
      S3      : set_outputs(  1,  1 );
      default : set_outputs( 'x, 'x );
    endcase
  end

endmodule
```