EECS 151 Disc 4

Rahul Kumar (session 1)
Yukio Miyasaka (session 2)
Contents

- State machines (Moore and Mealy)
- STD (State Transition Diagram)
- State assignment and encoding
- STD to Verilog
- STD to gate-level circuits
State Machines

- A way to systematically design sequential circuits
- Moore: outputs depend only on current state
- Mealy: outputs depend on current state and input
Example

Design a state machine to determine whether an unsigned binary integer is divisible by 3.

Inputs:
- \( x \): the input integer. Given to FSM one bit at a time.
- \( \text{rst} \): synchronous reset. At the next clock edge, clears any state associated with the FSM.

Output: one bit indicating if the value seen so far is divisible by 3.
Example

Input is supplied one bit at a time, MSB first.

Example: 1010111
Example

Draw a state transition diagram for this state machine.
Example

- Key idea: keep track of the number modulo 3.
- If 0, the number is divisible by 3.
- Can we track $N \% 3$ without storing $N$?
Example

What happens when we get a new bit? In pseudocode:

\[
\text{next} = 2 \times \text{prev} + \text{bit}
\]

Same equation applies modulo 3!

\[
\text{next}\_\text{residue} = (2 \times \text{prev}\_\text{residue} + \text{bit}) \mod 3
\]
Example

So keep 3 states, S0/S1/S2, corresponding to residues mod 3.

S0: if input is 1, go to S1. Else, stay at S0.

S1: if input is 1, go to S0. Else, go to S2.

S2: if input is 1, stay at S2. Else, go to S1.
Example (Moore Machine)
Example

Implement this FSM as a Moore machine in Verilog, using a binary state encoding.
module div3_moore(
    input clk,
    input x,
    input rst,
    output out
);

localparam S0 = 2'b00;
localparam S1 = 2'b01;
localparam S2 = 2'b10;
reg [1:0] next_state;
wire [1:0] state;
REGISTER_R #(.N(2), .INIT(S0)) state_reg (.q(state), .d(next_state), .rst(rst), .clk(clk));

assign out = state == S0;

always @(*) begin
    case (state)
        S0: begin
            next_state = x ? S1 : S0;
        end
        S1: begin
            next_state = x ? S0 : S2;
        end
        S2: begin
            next_state = x ? S2 : S1;
        end
    endcase
end
endmodule
Now implement the same logic as a Mealy machine. The output should update as soon as the input updates.
Example (Mealy Machine)
Example

```verilog
module div3_mealy(
    input clk,
    input x,
    input rst,
    output out
);
  localparam S0 = 2'b00;
  localparam S1 = 2'b01;
  localparam S2 = 2'b10;
  reg [1:0] next_state;
  wire [1:0] state;
  REGISTER_R #( .N(2), .INIT(S0)) state_reg (.q(state), .d(next_state), .rst(rst), .clk(clk));

  assign out = next_state == S0;

  always @(*) begin
    case (state)
      S0: begin
        next_state = x ? S1 : S0;
      end
      S1: begin
        next_state = x ? S0 : S2;
      end
      S2: begin
        next_state = x ? S2 : S1;
      end
    endcase
  end
endmodule
```
Example: Washing machine

- Takes two steps: wash and spin
- Wash twice if a double wash switch D is on
  - wash -> wash -> spin
- Receives a signal T from a timer:
  - When it should start washing
  - At the end of each step
State Transition Diagram

Ignore output signal for now
State Assignment

Initial = state 0
Wash = state 1
Wash2 = state 3
Spin = state 2

<table>
<thead>
<tr>
<th>s1\s0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial</td>
<td>Wash</td>
</tr>
<tr>
<td>1</td>
<td>Spin</td>
<td>Wash2</td>
</tr>
</tbody>
</table>

Transitions in binary encoding
module washing_machine_binary(input clk, rst, T, D);
localparam Initial = 2'b00;
localparam Wash    = 2'b01;
localparam Wash2   = 2'b11;
localparam Spin    = 2'b10;

reg [1:0] next_state;
wire[1:0] state;
REGISTER_R #(.N(2)) state_reg (.q(state), .d(next_state), .rst(rst), .clk(clk));

always @(*) begin
  next_state = state;
  case (state)
    Initial: if(T) next_state = Wash;
    Wash:   if(T) next_state = D? Wash2: Spin;
    Wash2:  if(T) next_state = Spin;
    Spin:   if(T) next_state = Initial;
  endcase
end
endmodule
Gate-Level Circuit with Binary Encoding

<table>
<thead>
<tr>
<th>s1</th>
<th>s0</th>
<th>T</th>
<th>D</th>
<th>n1</th>
<th>n0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>0</td>
<td>-</td>
<td>x</td>
<td>y</td>
</tr>
</tbody>
</table>
Gate-Level Circuit with Binary Encoding

<table>
<thead>
<tr>
<th>s1</th>
<th>s0</th>
<th>T</th>
<th>D</th>
<th>n1</th>
<th>n0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>0</td>
<td>-</td>
<td>x</td>
<td>y</td>
</tr>
</tbody>
</table>

K-map for n1

<table>
<thead>
<tr>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
</tr>
<tr>
<td>00</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

K-map for n0

<table>
<thead>
<tr>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
</tr>
<tr>
<td>00</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>
Gate-Level Circuit with Binary Encoding

<table>
<thead>
<tr>
<th></th>
<th>s1</th>
<th>s0</th>
<th>T</th>
<th>D</th>
<th>n1</th>
<th>n0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>0</td>
<td>-</td>
<td>x</td>
<td>y</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>s1s0</th>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>10</td>
</tr>
</tbody>
</table>

\[ n_1 = s_0 T + s_1 T' \]

<table>
<thead>
<tr>
<th></th>
<th>s1s0</th>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

\[ n_0 = s_0 T' + s_1' T D + s_0' s_1' T \]
Gate-Level Circuit with Binary Encoding

\[ n_1 = s_0 T + s_1 T' \]
\[ n_0 = s_0 T' + s_1' TD + s_0' s_1' T \]
\[ = s_0 T' + s_1' T (D + s_0') \]
One-hot Encoding

- state 0 = 0001  (Initial)
- state 1 = 0010  (Wash)
- state 2 = 0100  (Spin)
- state 3 = 1000  (Wash2)

State assignment doesn’t really matter in one-hot encoding
Verilog with One-hot Encoding

module washing_machine_onehot(input clk, rst, T, D);
wire [3:0] next_state, state;
REGISTER_R #(N(4)) state_reg (.q(state),
.d(next_state), .rst(rst), .clk(clk));

assign next_state[0] = (state[0] & ~T) | (state[2] & T);
assign next_state[1] = (state[1] & ~T) | (state[0] & T);
endmodule

Focus on incoming edges for each node
Gate-Level Circuit with One-hot Encoding

```
assign next_state[0] = (state[0] & ~T) | (state[2] & T);
assign next_state[1] = (state[1] & ~T) | (state[0] & T);
```