CSE140L: Components and Design Techniques for Digital Systems Lab

FSMs

Slides from Tajana Simunic Rosing
FSM design example – Moore vs. Mealy

- Remove one 1 from every string of 1s on the input
Verilog FSM - Reduce 1s example

- Moore machine

```verilog
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;

  parameter zero  = 2'b00;
  parameter one1  = 2'b01;
  parameter two1s = 2'b10;

  reg out;
  reg [1:0] state;    // state variables
  reg [1:0] next_state;

always @(posedge clk)begin
  if (reset) state = zero;
  else       state = next_state;
end
```

Always include a reset signal !!!
Moore Verilog FSM (cont’d)

```
always @(in or state)
begin
    case (state)
        zero:
            // last input was a zero
            begin
                if (in) next_state = one1;
                else next_state = zero;
            end
        one1:
            // we've seen one 1
            begin
                if (in) next_state = two1s;
                else next_state = zero;
            end
        two1s:
            // we've seen at least 2 ones
            begin
                if (in) next_state = two1s;
                else next_state = zero;
            end
    endcase
end
```

crucial to include all signals that are input to state determination

```
    always @(
    state) begin
        case (state)
            zero: out = 0;
            one1: out = 0;
            two1s: out = 1;
        endcase
    end
endmodule
```

note that output depends only on state
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  reg out;
  reg state; // state variables
  reg next_state;

always @(posedge clk) begin
  if (reset) state = zero;
  else       state = next_state;
end
always @(in or state)
case (state)
  zero:    // last input was a zero
    begin
      out = 0;
      if (in) next_state = one;
      else    next_state = zero;
    end
  one:     // we've seen one 1
    if (in) begin
      next_state = one; out = 1;
    end else begin
      next_state = zero; out = 0;
    end
endcase
endmodule
Mealy/Moore Summary

- Mealy machines tend to have fewer states
  - different outputs on arcs (i*n) rather than states (n)
- Mealy machines react faster to inputs
  - react in same cycle – don't need to wait for clock
  - delay to output depends on arrival of input
- Moore machines are generally safer to use
  - outputs change at clock edge (always one cycle later)
  - in Mealy machines, input change can cause output change as soon as logic is done – a big problem when two machines are interconnected – asynchronous feedback
EXTRA EXAMPLE (NOT MANDATORY)
Example: Traffic light controller

- Highway/farm road intersection
Traffic light controller (cont.)

• Detectors C sense the presence of cars waiting on the farm road
  – with no car on farm road, light remain green in highway direction
  – if vehicle on farm road, highway lights go from Green to Yellow to Red, allowing
    the farm road lights to become green
  – these stay green only as long as a farm road car is detected but never longer than
    a set interval; after the interval expires, farm lights transition from Green to Yellow
    to Red, allowing highway to return to green
  – even if farm road vehicles are waiting, highway gets at least a set interval of green

• Assume you have an interval timer that generates:
  – a short time pulse (TS) and
  – a long time pulse (TL),
  – in response to a set (ST) signal.
  – TS is to be used for timing yellow lights and TL for green lights
Traffic light controller (cont.)

- **inputs**
  - reset: place FSM in initial state
  - C: detect vehicle on the farm road
  - TS: short time interval expired
  - TL: long time interval expired

- **outputs**
  - HG, HY, HR: assert green/yellow/red highway lights
  - FG, FY, FR: assert green/yellow/red highway lights
  - ST: start timing a short or long interval

- **state**
  - HG: highway green (farm road red)
  - HY: highway yellow (farm road red)
  - FG: farm road green (highway red)
  - FY: farm road yellow (highway red)
Traffic light controller (cont.)

- Generate state table with symbolic states
- Consider state assignments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Present State</th>
<th>Next State</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>C TL TS</td>
<td>HG</td>
<td>HG</td>
<td>ST H F</td>
</tr>
<tr>
<td>0 – –</td>
<td>HG</td>
<td>HG</td>
<td>0 Green Red</td>
</tr>
<tr>
<td>– 0 –</td>
<td>HG</td>
<td>HG</td>
<td>0 Green Red</td>
</tr>
<tr>
<td>1 1 –</td>
<td>HG</td>
<td>HY</td>
<td>1 Green Red</td>
</tr>
<tr>
<td>– – 0</td>
<td>HY</td>
<td>HY</td>
<td>0 Yellow Red</td>
</tr>
<tr>
<td>– – 1</td>
<td>HY</td>
<td>FG</td>
<td>1 Yellow Red</td>
</tr>
<tr>
<td>1 0 –</td>
<td>FG</td>
<td>FG</td>
<td>0 Red Green</td>
</tr>
<tr>
<td>0 – –</td>
<td>FG</td>
<td>FY</td>
<td>1 Red Green</td>
</tr>
<tr>
<td>– 1 –</td>
<td>FG</td>
<td>FY</td>
<td>1 Red Green</td>
</tr>
<tr>
<td>– – 0</td>
<td>FY</td>
<td>FY</td>
<td>0 Red Yellow</td>
</tr>
<tr>
<td>– – 1</td>
<td>FY</td>
<td>HG</td>
<td>1 Red Yellow</td>
</tr>
</tbody>
</table>

SA1: HG = 00, HY = 01, FG = 11, FY = 10
SA2: HG = 00, HY = 10, FG = 01, FY = 11
SA3: HG = 0001, HY = 0010, FG = 0100, FY = 1000 (one-hot)

(output encoding – similar problem to state assignment (Green = 00, Yellow = 01, Red = 10))

Generate state table with symbolic states
Consider state assignments

- Present State
- Next State
- Outputs

C TL TS HG HG HG
– 0 – HG HG HG
1 1 – HG HY HY
– – 0 HY HY HY
– – 1 HY FG FG
1 0 – FG FG FG
0 – – FG FY FY
– 1 – FG FY FY
– – 0 FY FY FY
– – 1 FY HG HG

SA1: HG = 00, HY = 01, FG = 11, FY = 10
SA2: HG = 00, HY = 10, FG = 01, FY = 11
SA3: HG = 0001, HY = 0010, FG = 0100, FY = 1000 (one-hot)
Traffic light controller FSM

- Specification of inputs, outputs, and state elements

```verilog
module FSM(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, Clk);
  output HR;
  output HY;
  output HG;
  output FR;
  output FY;
  output FG;
  output ST;
  input TS;
  input TL;
  input C;
  input reset;
  input Clk;
  reg [6:1] state;
  reg ST;

  parameter highwaygreen = 6'b001100;
  parameter highwayyellow = 6'b010100;
  parameter farmroadgreen = 6'b100001;
  parameter farmroadyellow = 6'b100010;

  assign HR = state[6];
  assign HY = state[5];
  assign HG = state[4];
  assign FR = state[3];
  assign FY = state[2];
  assign FG = state[1];
```

specify state bits and codes for each state as well as connections to outputs
Traffic light controller FSM

initial begin state = highwaygreen; ST = 0; end

always @(posedge Clk)
begin
  if (reset)
    begin state = highwaygreen; ST = 1; end
  else
    begin
      ST = 0;
      case (state)
        highwaygreen:
          if (TL & C) begin state = highwayyellow; ST = 1; end
        highwayyellow:
          if (TS) begin state = farmroadgreen; ST = 1; end
        farmroadgreen:
          if (TL | !C) begin state = farmroadyellow; ST = 1; end
        farmroadyellow:
          if (TS) begin state = highwaygreen; ST = 1; end
      endcase
    end
end
endmodule
Timer FSM for traffic light controller

module Timer(TS, TL, ST, Clk);
  output TS;
  output TL;
  input ST;
  input Clk;
  integer value;

assign TS = (value >= 4); // 5 cycles after reset
assign TL = (value >= 14); // 15 cycles after reset

always @(posedge ST) value = 0; // async reset

always @(posedge Clk) value = value + 1;
endmodule
Complete traffic light controller

- Tying it all together (FSM + timer) with structural Verilog (same as a schematic drawing)

```verilog
module main(HR, HY, HG, FR, FY, FG, reset, C, Clk);
    output HR, HY, HG, FR, FY, FG;
    input  reset, C, Clk;

    Timer part1(TS, TL, ST, Clk);
    FSM    part2(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, Clk);
endmodule
```

![Traffic light controller diagram](#)
Finite state machines summary

- Models for representing sequential circuits
  - abstraction of sequential elements
  - finite state machines and their state diagrams
  - inputs/outputs
  - Mealy, Moore, and synchronous Mealy machines

- Finite state machine design procedure
  - deriving state diagram
  - deriving state transition table
  - determining next state and output functions
  - implementing combinational logic

- Hardware description languages
  - Use good coding style
  - Communicating FSMs