CSE140L: Components and Design Techniques for Digital Systems Lab

FSMs

Tajana Simunic Rosing
Hardware Description Languages and Sequential Logic

- **Flip-flops**
  - representation of clocks - timing of state changes
  - asynchronous vs. synchronous
- **FSMs**
  - structural view (FFs separate from combinational logic)
  - behavioral view (synthesis of sequencers – not in this course)
- **Datapath = data computation (e.g., ALUs, comparators) + registers**
  - use of arithmetic/logical operators
  - control of storage elements
Controller Design: Laser Timer Example

- **Problem:** Pressing button (b) once turns on laser for 3 clock cycles
- **Step 1:** Capture the FSM
- **Step 2:** Create architecture
- **Step 3:** Encode the states
- **Step 4:** Minimize logic
- **Step 5:** Implement & test

- How about synthesis into FPGA?
module LaserTimer(b, x, clk, rst);
input b, clk, rst;
output x;
reg x;

parameter S_Off = 2'b00,
      S_On1 = 2'b01,
      S_On2 = 2'b10,
      S_On3 = 2'b11;

reg [1:0] currentstate;
reg [1:0] nextstate;

// state register procedure
always @(posedge rst or posedge clk)
begin
  if (rst==1) // initial state
  begin
    currentstate <= S_Off;
  end
  else
  begin
    currentstate <= nextstate;
  end
end

// combinational logic procedure
always @(currentstate or b)
begin
  case (currentstate)
  S_Off: begin
    x <= 0; // laser off
    if (b==0) nextstate <= S_Off;
    else nextstate <= S_On1;
  end
  S_On1: begin
    x <= 1; // laser on
    nextstate <= S_On2;
  end
  S_On2: begin
    x <= 1; // laser still on
    nextstate <= S_On3;
  end
  S_On3: begin
    x <= 1; // laser still on
    nextstate <= S_Off;
  end
  endcase
end
endmodule // LaserTimer
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 twols = 2'b10;

    reg out;
    reg [2:1] state;    // state variables
    reg [2:1] next_state;

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

state assignment (easy to change, if in one place)
Moore Verilog FSM (cont’d)

always @(in or state)

    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 = twols;
                else    next_state = zero;
            end
        twols:
            // we've seen at least 2 ones
            begin
                if (in) next_state = twols;
                else    next_state = zero;
            end
    endcase

endmodule
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  reg out;
  reg state; // state variables
  reg next_state;

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

  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
Synchronous Mealy Machine

module reduce (clk, reset, in, out);
    input clk, reset, in;
    output out;
    reg out;
    reg state; // state variables

always @(posedge clk)
    if (reset) state = zero;
    else
        case (state)
            zero: // last input was a zero
                begin
                    out = 0;
                    if (in) state = one;
                    else state = zero;
                end
            one: // we've seen one 1
                if (in) begin
                    state = one; out = 1;
                end else begin
                    state = zero; out = 0;
                end
        endcase
endmodule
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 description
  reset place FSM in initial state
  C detect vehicle on the farm road
  TS short time interval expired
  TL long time interval expired

• outputs description
  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 description
  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></td>
<td></td>
<td>ST H F</td>
</tr>
<tr>
<td>0 0 0</td>
<td>HG</td>
<td>HG</td>
<td>0 Green Red</td>
</tr>
<tr>
<td>1 0 1</td>
<td>HG</td>
<td>HG</td>
<td>0 Green Red</td>
</tr>
<tr>
<td>0 1 0</td>
<td>HY</td>
<td>HY</td>
<td>1 Green Red</td>
</tr>
<tr>
<td>0 0 1</td>
<td>HY</td>
<td>FG</td>
<td>1 Yellow Red</td>
</tr>
<tr>
<td>1 0 0</td>
<td>FG</td>
<td>FG</td>
<td>0 Red Green</td>
</tr>
<tr>
<td>0 1 1</td>
<td>FG</td>
<td>FY</td>
<td>1 Red Green</td>
</tr>
<tr>
<td>0 0 0</td>
<td>FY</td>
<td>FY</td>
<td>0 Red Yellow</td>
</tr>
<tr>
<td>0 0 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)
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
endmodule
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
```
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