CSE 20221: Logic Design

Finite State Machines in Verilog

Jay Brockman
Department of Computer Science and Engineering
Department of Electrical Engineering
University of Notre Dame

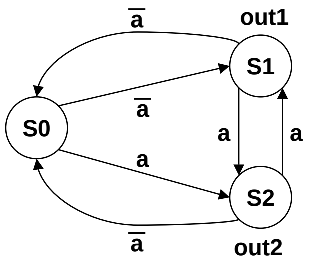
Key Points

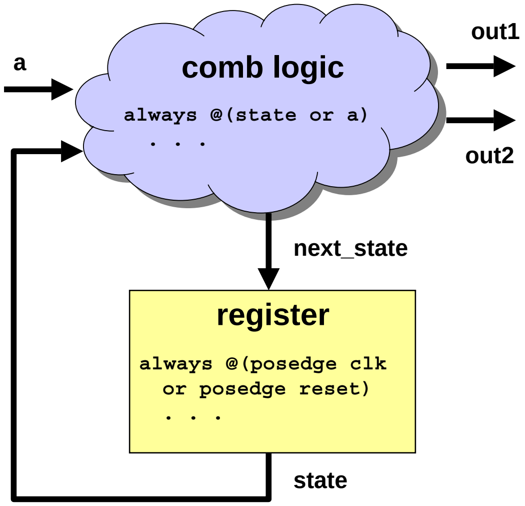
- Not difficult to describe FSM *behavior* in an HDL
- But goal is to *synthesize* good *hardware*
  - Need to be very precise so synthesis tools can do this
  - Subtle mistakes can have major effect
- My goal: equip you with a simple yet flexible and reliable approach to writing FSMs in Verilog
  - enough detail to avoid getting bitten
  - but also avoid confusion

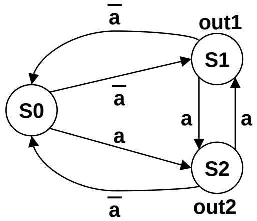
What we know so far

- Finite state machines
  - Binary encoded state names
  - Next state depends on
    - current state
    - input
  - Outputs depend on
    - Moore machine: state only
    - Mealy machine: state and inputs
- Verilog
  - Combinational logic
    - sensitive to changes in signal levels
  - Sequential logic (registers)
    - sensitive to changes in clock edges (and reset)

Describing FSM in Verilog





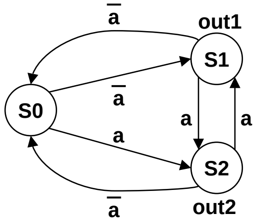


```
module fsm(clk, reset, a, out1, out2);
    input clk;
    input reset;
    input a;
    output reg out1;
    output reg out2;

   reg [1:0] state;
   reg [1:0] next_state;

   parameter s0 = 2'b00;
   parameter s1 = 2'b01;
   parameter s2 = 2'b10;
```

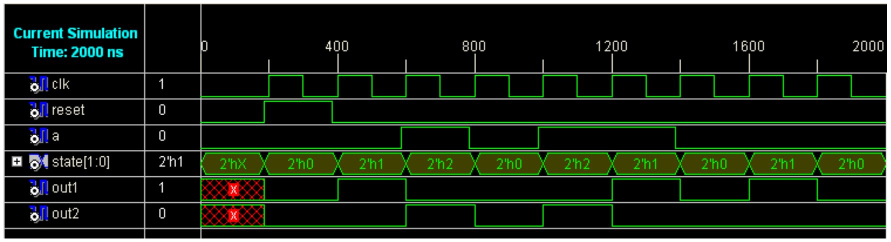
```
always @(posedge clk or posedge reset)
 if (reset)
   state <= s0;
 else
   state <= next_state;
```

```
   always @(state or a)
     case (state)
       s0: begin
         if (a == 0)
           next_state <= s1;
         else
           next_state <= s2;
         out1 <= 0;
         out2 <= 0;
       end
       s1: begin
         if (a == 0)
           next_state <= s0;
         else
           next_state <= s2;
         out1 <= 1;
         out2 <= 0;
       end
       s2: begin
         if (a == 0)
           next_state <= s0;
         else
           next_state <= s1;
         out1 <= 0;
         out2 <= 1;
       end
     endcase
endmodule
```

Simulation Results



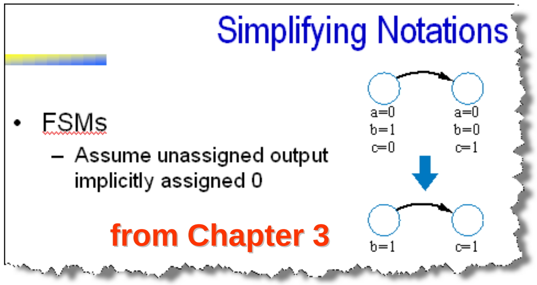
Looks good



Current Simulation Time: 2000 ns

```
                 0     400     800     1200     1600     2000
clk         1
reset       0
a           0
state[1:0]  2'h1   2'hX 2'h0 2'h1 2'h2 2'h0 2'h2 2'h1 2'h0 2'h1 2'h0
out1        1
out2        0
```

Unassigned Output (Beware!)



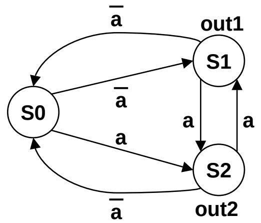
Simplifying Notations

- FSMs
  - Assume unassigned output implicitly assigned 0

from Chapter 3

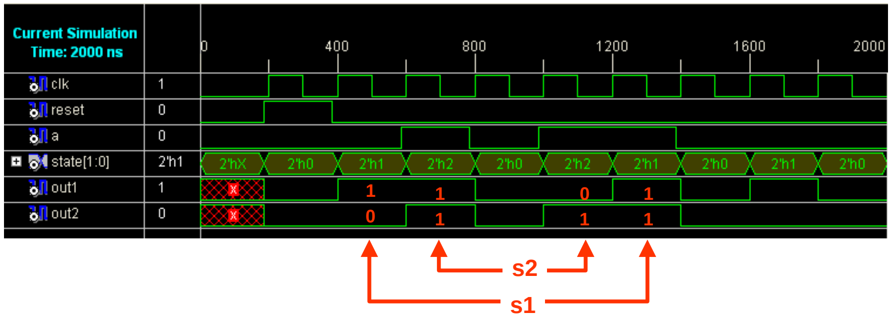
```
always @(state or a)
  case (state)
    s0: begin
      if (a == 0)
        next_state <= s1;
      else
        next_state <= s2;
      out1 <= 0;
      out2 <= 0;
    end
    s1: begin
      if (a == 0)
        next_state <= s0;
      else
        next_state <= s2;
      out1 <= 1;
    end
    s2: begin
      if (a == 0)
        next_state <= s0;
      else
        next_state <= s1;
      out2 <= 1;
    end
  endcase
endmodule
```

Unassigned Output: Simulation Results



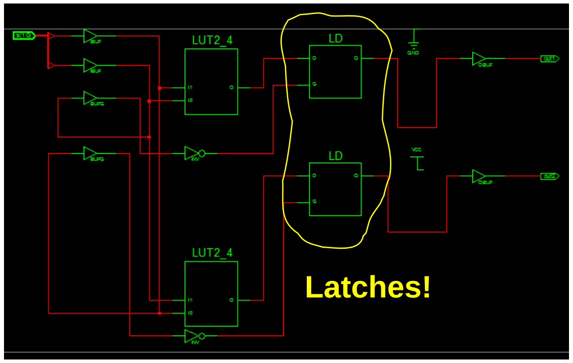
Not what we intended!

- *Doesn't* default to 0
- Keeps previous value

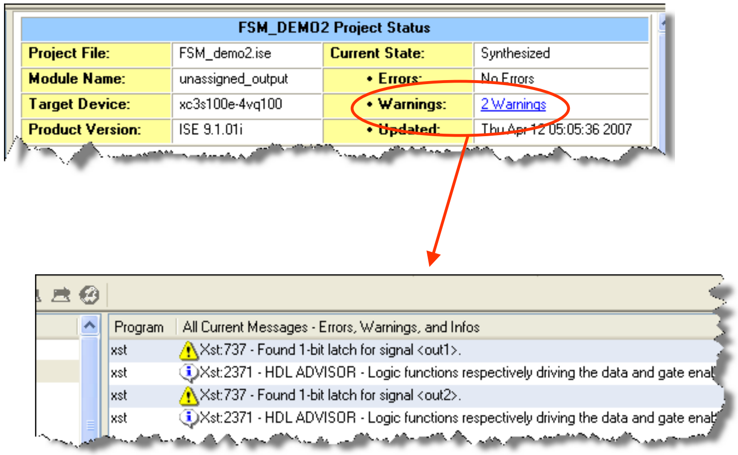


Synthesis of Unassigned Output

```
module unassigned_output(s, out1, out2);
    input [1:0] s;
    output reg out1;
    output reg out2;

   parameter s0 = 2'b00;
   parameter s1 = 2'b01;
   parameter s2 = 2'b10;

   always @(s)
    case (s)
      s0: begin
        out1 <= 0;
        out2 <= 0;
      end
      s1: out1 <= 1;
      s2: out2 <= 1;
    endcase
endmodule
```



Detecting Unassigned Output/Latch Problem



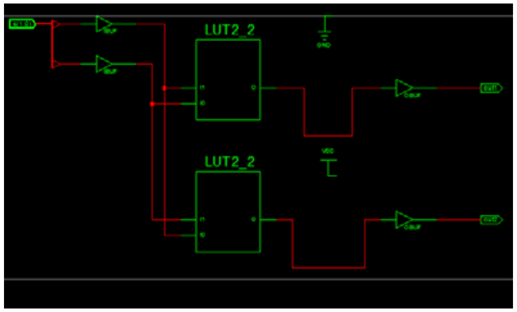
FSM_DEMO2 Project Status

Project File:	FSM_demo2.ise	Current State:	Synthesized
Module Name:	unassigned_output	• Errors:	No Errors
Target Device:	xc3s100e-4vq100	• Warnings:	2 Warnings
Product Version:	ISE 9.1.01i	• Updated:	Thu Apr 12 05:05:36 2007

Program	All Current Messages - Errors, Warnings, and Infos
xst	⚠ Xst:737 - Found 1-bit latch for signal <out1>.
xst	ℹ Xst:2371 - HDL ADVISOR - Logic functions respectively driving the data and gate enal
xst	⚠ Xst:737 - Found 1-bit latch for signal <out2>.
xst	ℹ Xst:2371 - HDL ADVISOR - Logic functions respectively driving the data and gate enal

Always assign all outputs! Don't leave undefined input cases!

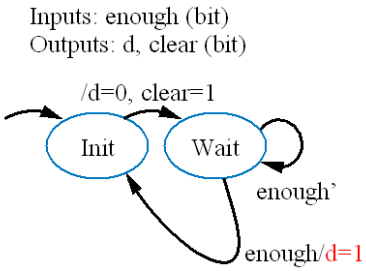
```
always @(s)
  case (s)
    s0: begin
      out1 <= 0;
      out2 <= 0;
    end
    s1: begin
      out1 <= 1;
      out2 <= 0;
    end
    s2: begin
      out1 <= 0;
      out2 <= 1;
    end
    default: begin
      out1 <= 0;
      out2 <= 0;
    end
  endcase
```



Define default input case so FSM can't get "stuck" (Also a latch problem)

Mealy Machine

outputs on transitions



```
always @(state or enough)
  case (state)
    s_init: begin
      next_state <= s_wait;
      d <= 0;  clear <= 1;
    end
    s_wait: begin
      if (!enough)
        begin
          next_state <= s_wait;
          d <= 0;  clear <= 0;
        end
      else
        begin
          next_state <= s_init;
          d <= 1;  clear <= 0;
        end
    end
  endcase
```

Binary vs. One-Hot Encoding

binary

```
parameter s0 = 2'b00;
parameter s1 = 2'b01;
parameter s2 = 2'b10;
```

one-hot

```
parameter s0 = 3'b001;
parameter s1 = 3'b010;
parameter s2 = 3'b100;
```

- Synthesis tools will generally optimize, regardless of how you encode states
- FPGAs generally prefer one-hot
  - less fan in → simpler logic
  - less fan in → faster logic
  - plenty of flip-flops available (approximately 1 per LUT)