ECE 2300
Digital Logic & Computer Organization
Spring 2021

More Verilog
Finite State Machines
Announcements

• HW 2 due tomorrow

• Lab 2A due on Monday 3/8
  – No lecture, lab, and OH next Tuesday & Wednesday

• A bit more about Prelim 1
  – Open book, open notes
  – You are expected to (1) sign into zoom 5 mins earlier, and (2) have your camera on the whole time
  – More instructions will be posted next week
Recap: Assignments in Verilog

• Continuous assignments apply to combinational logic only

• *Always blocks* contain a set of procedural assignments (blocking or nonblocking)
  – Can be used to model *either combinational or sequential logic*
  – Always blocks execute concurrently with other always blocks, instance statements, and continuous assignment statements in a module
### Always Block Examples

- **Which one(s) of the above Verilog code will result in sequential logic**

<table>
<thead>
<tr>
<th>Code</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>reg Z;</code>&lt;br&gt;always @ (clk) begin&lt;br&gt;  Z = A &amp; B;&lt;br&gt; end</td>
<td>??</td>
</tr>
<tr>
<td><code>reg Z;</code>&lt;br&gt;always @ (posedge clk) begin&lt;br&gt;  Z &lt;= A &amp; B;&lt;br&gt; end</td>
<td>YES</td>
</tr>
<tr>
<td><code>reg Z;</code>&lt;br&gt;always @ (A, B) begin&lt;br&gt;  Z &lt;= A &amp; B;&lt;br&gt; end</td>
<td>NO</td>
</tr>
</tbody>
</table>
Recap: Procedural Assignments

• **Blocking assignments** ( = )
  – Simulation behavior: Right-hand side (RHS) evaluated sequentially; assignment to LHS is immediate

• **Nonblocking assignment** ( <= )
  – Simulation behavior: RHS evaluated in parallel (order doesn’t matter); assignment to LHS is delayed until end of always block

```verilog
reg Y, Z;
always @ (posedge clk)
begin
  Y = A & B;
  Z = ~Y;
end
```

**Blocking assignments**

```verilog
Y_{next} = A & B
Z_{next} = ~(A & B)
```

```verilog
reg Y, Z;
always @ (posedge clk)
begin
  Y <= A & B;
  Z <= ~Y;
end
```

**Nonblocking assignments**

```verilog
Z_{next} = ~Y // the old Y
Y_{next} = A & B
```
Recap: Procedural Assignments

```
reg Y, Z;
always @ (posedge clk) begin
  Y = A & B;
  Z = ~Y;
end
```

**Blocking assignments**

```
reg Y, Z;
always @ (posedge clk) begin
  Y <= A & B;
  Z <= ~Y;
end
```

**Nonblocking assignments**

Y appears on RHS of a blocking assignment <=> use its input signal for connection

Y appears on RHS of a nonblocking assignment <=> use its output signal for connection
Net and Variable Types

- **We will mainly use two data type classes**
  - *wire*: represents a physical connection (or net) between hardware elements
    - A stateless way of connected two elements
    - Can only be used to model combinational logic
    - Cannot be used in the left-hand side in an always block
  - *reg*: similar to wires, but can be used to store information (or state) like registers
    - This is used in the behavioral style only
    - Can be used to model both combinational & sequential logic
    - Cannot be used in the left-hand side of a continuous assignment statement
(Improperly Created) *Inferred* Latches

- To infer combinational logic, you’re recommended to ensure that each variable within an *always* block gets assigned a value (under all possible conditions)
  - Otherwise, the Verilog compiler assumes that the last value should be used, and will create a latch

```verilog
reg out;
always @(d, sel)
begin
    /* out not always assigned a value; **latch inferred** */
    if (sel == 1'b1)
        out = d;
    else
        out = ~d;
end
```

```verilog
reg out;
always @(d, sel)
begin
    /* out assigned a value in both conditions; **latch not inferred** */
    if (sel == 1'b1)
        out = d;
    else
        out = ~d;
end
```
• A Finite State Machine (FSM) is an abstract representation of a sequential circuit
  – The state embodies the condition of the system at this particular time
  – The combinational logic determines the output and next state values
  – The output values may depend only on the current state value, or on the current state and input values
Elements of an FSM

1. A finite number of **inputs**
2. A finite number of **outputs**
3. A finite number of **states**
4. A specification of all **state transitions**

Can be described by a **state diagram**
FSM: General Form

- Inputs and current state determine state transitions
- Output changes determined by changes in
  - Current state, or
  - Current state + inputs
Moore Machine

Outputs only depend on current state value
Mealy Machine

Outputs depend on input and current state values
State Diagram

- **Visual specification of an FSM**
  - Bubble for every state
  - Arcs showing state transitions
  - Input values shown on the arcs
  - Output values shown within the bubbles (Moore) or on the arcs (Mealy)
  - Clock input implicit (always present, triggering state transitions)
Moore State Diagram

- 1 input, 1 output, 3 states
- Bubble for each state
- State transitions (arcs) for each input value
- Input values on the arcs
- Output values within the bubbles
- Starts at S0 when Reset asserted
Mealy State Diagram

- 1 input, 1 output, 2 states
- Bubble for each state
- State transitions (arcs) for each input value
- Input values on the arcs (first number)
- Output values on the arcs (second number)
- Starts at S0 when Reset asserted
FSM Design Procedure

(1) Understand the problem statement and determine inputs and outputs

(2) Identify states and create a state diagram

(3) Determine the number of required FFs

(4) Implement combinational logic for outputs and next state

(5) Simulate the circuit to test its operation
Example FSM: Pattern Detector

- Monitors the input, and outputs a 1 whenever a specified input pattern is detected.

- Example: Output a 1 whenever 111 is detected on the input over 3 consecutive clock cycles.
  - Overlapping patterns also detected (1111...)

- Input $In$
- Output $Out$
- $Reset$ causes FSM to start in initial state
- $Clock$ input not shown (always present)
111 Pattern Detector: Moore State Diagram
111 Pattern Detector: Mealy State Diagram
Example FSM: Pushbutton Lock

- Two pushbutton inputs, X1 and X2
- One output, UL (“Unlock”)

- UL = 1 when X1 is pushed, followed by X2 being pushed twice (X1, X2, X2)

- Represent X1 and X2 as two bit input
  - 00: neither button pushed
  - 10: X1 pushed
  - 01: X2 pushed
  - 11: both pushed, reset the lock
Pushbutton Lock: Moore State Diagram

- **Output**: UL=1 with \((X1, X2, X2)\)
- **Input**: 00 (neither), 10 (X1), 01 (X2), 11 (reset)
Pushbutton Lock: Mealy State Diagram

- Output: UL=1 with (X1, X2, X2)
- Input: 00 (neither), 10 (X1), 01 (X2), 11 (reset)
Readings Before Next Class

Tuesday 3/9: No Lecture

Thursday 3/11: More Finite State Machines
H&H 4.9