ECE 2300
Digital Logic & Computer Organization
Spring 2022

More Verilog
Finite State Machines
Announcements

• HW 3 due tomorrow

• No lab sessions next week
  – Prelim 1 review on Monday (2/21) at 7:30pm in TA zoom room
Exercise: Assignments in Verilog

• Which one(s) of the following Verilog code snippets infer sequential logic

(a) input clk, d;
    output q;
    assign q = clk & d;

(b) input clk, d;
    output reg q;
    always @ (posedge clk)
    begin
        q <= d;
    end

(c) input clk, d;
    output reg q;
    always @ (clk)
    begin
        if ( clk ) // D latch ?
            q <= d;
    end
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**

**Simulator Interpretation**

\[
Y_{\text{next}} = A \& B \\
Z_{\text{next}} = \sim(A \& B) \quad \text{/* negating the new Y */} \\
Y_{\text{next}} = A \& B \\
Z_{\text{next}} = \sim Y \quad \text{/* negating the old Y */}
\]

- Right-hand side (RHS) evaluated **sequentially**
- Assignment to LHS is **immediate**
- RHS evaluated in parallel (order doesn’t matter)
- Assignment to LHS is **delayed** until end of always block
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

Actual Circuit
(Synthesizer Interpretation)

When a reg appears on RHS of a blocking assignment ("Y" here), use its input for connection ("A&B" here)

When a reg appears on RHS of a nonblocking assignment (Y here), use its output 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**
Lecture 8: 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

Lecture 8: 12
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 FSM

Mealy FSM
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

Moore FSM
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

Mealy FSM

Lecture 8: 15
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)
Before Next Class

• H&H 4.9

Next Time

More Finite State Machines