ECE 2300
Digital Logic & Computer Organization
Spring 2018

More Verilog
Finite State Machines
Announcements

- **Prelim 1, Thursday 3/1, 1:25pm, 75 mins**
  - Arrive early by 1:20pm
  - Review sessions
    - Monday 2/26, 7:00pm, PHL 407
    - Tuesday 2/27, 7:00pm, PHL 407
  - Coverage: Lecture 1~7, HW 1~3
  - Sample prelim released on CMS
  - Solutions to HW 1 & 2 released on CMS
    - HW3 solution to be released on Sunday

- **TA-led Verilog tutorial Next Tuesday (2/27)**
  - Bring your laptop!
Review: Verilog Design Styles

• **Structural**
  – A module is built from other (sub-)modules via *instance statements*

• **Behavioral**
  – Use *continuous assignments* and/or procedural code in *always blocks* to indicate what actions to take
Review: 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

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

**Blocking assignments**

\[
Y_{\text{next}} = A \& B \\
Z_{\text{next}} = \neg(A \& B)
\]

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

**Nonblocking assignments**

\[
Z_{\text{next}} = \neg Y \quad // \text{the old } Y \\
Y_{\text{next}} = A \& B
\]
Review: Procedural Assignments

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

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

• Corresponding circuit diagrams

• Y and Z are flip-flops in actual hardware
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

- 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 dout;
always @(din, c)
begin
  /* dout not always assigned a value; latch inferred */
  if (c == 1'b1)
    dout = din;
else
  dout = ~din;
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

Inputs \rightarrow \text{Combinational Logic} \rightarrow \text{Outputs}

\text{Current State} \rightarrow \text{FF} \rightarrow \text{Next State}

\text{FF} \rightarrow \text{CLK} \rightarrow \text{FF}
Moore Machine

Outputs only depend on current state value
Lecture 8: Mealy Machine

Outputs depend on input and current state values

Inputs → Outputs

- Output Combinational Logic
- Next State Combinational Logic
- FF
- FF
- Current State → Next State

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
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 D 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 \( \text{In} \)
- Output \( \text{Out} \)
- \( \text{Reset} \) causes FSM to start in initial state
- \( \text{Clock} \) 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)
Readings Before Next Class

Tuesday 2/27: Verilog Tutorial
Tutorial C: Introduction to Verilog and Quartus II

Tuesday 3/6: More Finite State Machines
H&H 4.9