More Finite State Machines
Announcements

• Prelim 1, Thursday 2/24, 1:00-2:15pm (in class)
  – Arrive early by 12:55pm
  – Instructor OH this week moved to Wed 4:30pm
  – Extra TA OHs (virtual) on Wed from 7:30pm
Review: Moore Machine

Outputs only depend on current state value
Review: Mealy Machine

Outputs depend on input and current state values

- Outputs
- Inputs
- Current State
- Next State
- FF
- Next State
- Current State
- FF

Output Combinational Logic

Next State Combinational Logic
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 \textit{In}
- Output \textit{Out}
- \textit{Reset} causes FSM to start in initial state
- \textit{Clock} input not shown (always present)
State Diagrams

Moore

- Init: Out = 0
- Got1: Out = 0
- Got11: Out = 0
- Got111: Out = 1

Mealy

- Init: Out = 0
- Got1: Out = 0
- Got11: Out = 1
Transition/Output Table

• Shows the next state (S*) and output values for each combination of current state (S) and inputs

• Used to derive the minimized state transition (S*) and output Boolean equations
Moore Transition/Output Table 1

<table>
<thead>
<tr>
<th>Current State (S)</th>
<th>Next State (S*)</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>In = 0</td>
<td>In = 1</td>
</tr>
<tr>
<td>Init</td>
<td>Init</td>
<td>Got1</td>
</tr>
<tr>
<td>Got1</td>
<td>Init</td>
<td>Got11</td>
</tr>
<tr>
<td>Got11</td>
<td>Init</td>
<td>Got111</td>
</tr>
<tr>
<td>Got111</td>
<td>Init</td>
<td>Got111</td>
</tr>
</tbody>
</table>

- Version 1: uses descriptive state names
Moore Transition/Output Table 2

- Version 2: uses state binary encodings
Minimized Equations for $S^*$ and Out

<table>
<thead>
<tr>
<th>$S_1$ $S_0$</th>
<th>$S_1^<em>$ $S_0^</em>$</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\text{In} = 0$</td>
<td>$\text{In} = 1$</td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 1</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>1 0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>1 1</td>
</tr>
<tr>
<td>1 1</td>
<td>0 0</td>
<td>1 1</td>
</tr>
</tbody>
</table>
Minimized Equations for $S^*$ and $Out$

$S_1^* = S_0 \cdot In + S_1 \cdot In$

$S_0^* = S_0^' \cdot In + S_1 \cdot In$

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$S_1^*$ ($In = 0$)</th>
<th>$S_1^*$ ($In = 1$)</th>
<th>$Out$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>00</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>00</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>

$Out = S_1 \cdot S_0$
Mealy Transition/Output Table 1

<table>
<thead>
<tr>
<th>Current State (S)</th>
<th>Next State (S*), Out</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>In = 0</td>
</tr>
<tr>
<td>Init</td>
<td>Init, 0</td>
</tr>
<tr>
<td>Got1</td>
<td>Init, 0</td>
</tr>
<tr>
<td>Got11</td>
<td>Init, 0</td>
</tr>
</tbody>
</table>

- Version 1: uses descriptive state names

Lecture 9: 13
Mealy Transition/Output Table 2

- Version 2: uses state binary encodings
Minimized Equations for $S^*$ and $Out$

| $S1 \ S0$ | $S1^* \ S0^*$, $Out$ |
|-----------|----------------|----------------|
|           | $In = 0$   | $In = 1$   |
| 0 0       | 0 0, 0    | 0 1, 0    |
| 0 1       | 0 0, 0    | 1 0, 0    |
| 1 0       | 0 0, 0    | 1 0, 1    |
Minimized Equations for $S^*$ and $Out$

$S_1^* = S_0 \cdot In + S_1 \cdot In$

$S_0^* = S_1' \cdot S_0' \cdot In$

$Out = S_1 \cdot In$

<table>
<thead>
<tr>
<th>$S1$ $S0$</th>
<th>$S1^<em>$ $S0^</em>$, Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>In = 0</td>
</tr>
<tr>
<td>0 0</td>
<td>0 0, 0</td>
</tr>
<tr>
<td>0 0</td>
<td>0 1, 0</td>
</tr>
<tr>
<td>0 1</td>
<td>1 0, 0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0, 1</td>
</tr>
</tbody>
</table>
FSMs in Verilog

<module statement>
<input and output declarations>

<reg declarations>

<parameter or typedef statement>

<always block for next state>

<always block for output>

<always block for state FFs>

endmodule

Suggested coding style for FSM
module PatDetectMoore (Clk, In, Reset, Out);
input Clk, In, Reset;
output Out;

reg Out;
reg [1:0] Scurr, Snext;

parameter [1:0] Init = 2'b00,
                Got1 = 2'b01,
                Got11 = 2'b10,
                Got111 = 2'b11;
Moore FSM in Verilog

always @ (In, Scurr)
begin
    case (Scurr)
        Init: if (In == 1) Snext = Got1; else Snext = Init;
        Got1: if (In == 1) Snext = Got11; else Snext = Init;
        Got11: if (In == 1) Snext = Got111; else Snext = Init;
        Got111: if (In == 1) Snext = Got111; else Snext = Init;
        default: Snext = Init;
    endcase
end
```verilog
always @ (Scurr)
    if (Scurr == Got111) Out = 1;
    else Out = 0;

always @ (posedge Clk)
    if (Reset == 1) Scurr <= Init;
    else Scurr <= Snext;
endmodule
```
module PatDetectMealy (Clk, In, Reset, Out);
input Clk, In, Reset;
output Out;

reg Out;
reg [1:0] Scurr, Snext;

parameter [1:0] Init = 2'b00,
                 Got1 = 2'b01,
                 Got11 = 2'b10;
always @ (In, Scurr)
begin
    case (Scurr)
        Init: if (In == 1) Snext = Got1; else Snext = Init;
        Got1: if (In == 1) Snext = Got11; else Snext = Init;
        Got11: if (In == 1) Snext = Got11; else Snext = Init;
        default: Snext = Init;
    endcase
    Snext = Init;
end
Mealy FSM in Verilog

always @ (Scurr, In)
    if ((Scurr == Got11) && (In == 1)) Out = 1;
    else Out = 0;

always @ (posedge Clk)
    if (Reset == 1) Scurr <= Init;
    else Scurr <= Snext;

endmodule

output comb logic

update state FFs
(current state)
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 sequence X1, X2, X2
- **Input:** 00 (neither), 10 (X1), 01 (X2), 11 (reset)
### Moore Transition/Output Table 1

<table>
<thead>
<tr>
<th>Current State (S)</th>
<th>Next State (S*)</th>
<th>UL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Input 0 0 (neither)</td>
<td>Input 0 1 (X2)</td>
</tr>
<tr>
<td>Init</td>
<td>Init</td>
<td>Init</td>
</tr>
<tr>
<td>X1</td>
<td>X1</td>
<td>X1-X2</td>
</tr>
<tr>
<td>X1-X2</td>
<td>X1,X2</td>
<td>X1-X2-X2</td>
</tr>
<tr>
<td>X1-X2-X2</td>
<td>X1-X2-X2</td>
<td>X1-X2-X2</td>
</tr>
</tbody>
</table>

- **Version 1: uses descriptive state names**
Moore Transition/Output Table 2

<table>
<thead>
<tr>
<th>$S_1$ $S_0$</th>
<th>$S_1^<em>$ $S_0^</em>$</th>
<th>UL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Input 0 0</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>1 0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>1 1</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
</tbody>
</table>

- Version 2: uses state binary encodings
Pushbutton Lock: Mealy State Diagram

- **Output:** UL=1 with sequence X1, X2, X2
- **Input:** 00 (neither), 10 (X1), 01 (X2), 11 (reset)
Mealy Transition/Output Table

<table>
<thead>
<tr>
<th>$S_1$ $S_0$</th>
<th>$S_1^<em>$ $S_0^</em>$, UL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Input 0 0</td>
</tr>
<tr>
<td>0 0</td>
<td>0 0, 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1, 0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0, 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1, 1</td>
</tr>
</tbody>
</table>

- uses state binary encodings
Analyzing the Sequential Logic

What does this circuit do?

Write down transition and output equations

\[ S_0^* = \]
\[ S_1^* = \]
\[ \text{Out} = \]
Exercise: Reconstruct State Diagram

- Complete the transition/output table and state diagram
- Identify the functionality of the FSM (Hint: Pattern detector?)

\[ S_0^* = \text{In}' \]
\[ S_1^* = \text{In}’ \cdot S_1 \cdot S_0’ + \text{In} \cdot S_0 \]
\[ \text{Out} = S_1 \cdot S_0 \]

<table>
<thead>
<tr>
<th>( S_1 ) ( S_0 )</th>
<th>( S_1^* ) ( S_0^* )</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 1</td>
<td>0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>1 0</td>
</tr>
<tr>
<td>1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Before Next Class

Thursday 2/24: In-class Prelim 1

Next Lecture:
More FSMs, Timing, Clocking
H&H 2.9, 4.6, 4.9