Finite State Recognizers and Sequence Detectors

ECE 152A – Winter 2012
Reading Assignment

- **Brown and Vranesic**
  - 8 Synchronous Sequential Circuits
    - 8.4 Design of Finite State Machines Using CAD Tools
      - 8.4.1 Verilog Code for Moore-Type FSMs
      - 8.4.2 Synthesis of Verilog Code
      - 8.4.3 Simulating and Testing the Circuit
      - 8.4.4 Alternative Styles of Verilog Code
      - 8.4.5 Summary of Design Steps When Using CAD Tools
      - 8.4.6 Specifying the State Assignment in Verilog Code
      - 8.4.7 Specification of Mealy FSMs Using Verilog
Reading Assignment

- **Roth**
  - 14 Derivation of State Graphs and Tables
    - 14.1 Design of a Sequence Detector
    - 14.2 More Complex Design Problems
    - 14.2 Guidelines for Construction of State Graphs
Mealy and Moore Machines

- **Mealy Machine**
  - Output is a function of present state and present input
    - Outputs valid on clock edge (transition)
  - Simpler (possibly)
  - Faster (possibly)
  - Outputs “glitch”
  - Used for synchronous (clocked) designs
Mealy and Moore Machines

- Moore Machine
  - Output is a function of present state only
    - Outputs valid after state transition
  - More “stable” than Mealy machine
    - Outputs do not glitch
  - Asynchronous (no clock) or synchronous designs
Deterministic Recognizers

- **State Diagram**
  - Also referred to as Deterministic Transition Graph
  - Next state transition is determined uniquely by present state and present input

- **Deterministic Recognizer**
  - Classifies input strings into two classes:
    - Those it accepts
    - Those it rejects
Deterministic Recognizers

- Sequential Lock Analogy
  - Accepted string corresponds to of the combination of the lock
    - Accepted string opens the lock
    - Rejected string leaves the lock closed

- Provides a basis for general purpose, finite state machine (FSM) design
  - Controllers, peripheral interfaces, etc.
Deterministic Recognizers

- Definition of states
  - Starting (or initial) state must be defined
  - The states whose assigned output is 1 are referred to as *accepting* (or *terminal*) states
  - The states whose assigned output is 0 are called *rejecting* (or *nonterminal*) states

- Above definition of states and control implies a Moore finite-state machine
  - With the requirement of a defined initial state
Deterministic Recognizers

Definition of acceptance and recognition

- A string is accepted by a machine if and only if the state that the machine enters after having read the rightmost symbol is an accepting state
- Otherwise, the string is rejected

- The set of strings recognized by a machine thus consists of all the input strings that take the machine from its starting state to an accepting state
Regular Expressions

- Concerned here with the characterization of sets of strings recognized by finite automata.
- A compact language for describing such sets of strings is known as the language of regular expressions.
  - Example 01(01)* describes the set consisting of those strings that can be formed by concatenating one or more 01 strings:
    - 01 + 0101 + 010101 + 01010101 + ...
Design Example

- Design a Moore machine that recognizes the input string ending with 101
  - Any string ending in 101 will be accepted
    - Regular expression is \((1+0)^(101)\)
    - 111101 recognizes (accepts) string on sixth input
  - The machine’s output goes to one each time the sequence 101 is detected
    - 10101 recognizes (accepts) string on the fifth input
    - Circuit’s output goes high on third input and fifth input
Design Example

- State Diagram

Starting State

Accepting State
### Design Example

- **State table with secondary state assignment**

<table>
<thead>
<tr>
<th>PS</th>
<th>AB</th>
<th>NS x=0</th>
<th>NS x=1</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>10</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>00</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>10</td>
<td>01</td>
<td>1</td>
</tr>
</tbody>
</table>
Design Example

- Next State Maps

A⁺ = x'B + xAB'

Z = AB (from state table)

B⁺ = x
Design Example

- Design can now be implemented
  - In discrete hardware, directly from next state maps with D flip-flops or using excitation tables for T or JK flip-flops
  - In Verilog directly from state table
    - Verilog implementation follows
Moore Machine - Verilog Implementation

- **Verilog Code**
  - state[1] = state variable A
  - state[0] = state variable B
  - Symbolic states
    - zero, one, two, three

```verilog
module moore (clk, x, z);
  input clk, x;
  output z;
  reg [1:0] state;

  parameter [1:0] zero = 2'b00, one = 2'b01, two = 2'b10, three = 2'b11;

  assign z = state[1] & state[0];

  always @(posedge clk)
    case (state)
      zero : if (x == 0)
        state <= zero;
        else
        state <= one;
      one  : if (x == 0)
        state <= one;
        else
        state <= two;
      two  : if (x == 0)
        state <= zero;
        else
        state <= three;
      three : if (x == 0)
        state <= two;
        else
        state <= one;
      default: state <= zero;
    endcase
  endmodule
```
Moore Machine – Verilog Implementation

- Timing Simulation

Input sequence

<table>
<thead>
<tr>
<th>Name:</th>
<th>50.0ns</th>
<th>100.0ns</th>
<th>150.0ns</th>
<th>200.0ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>I clk</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>I x</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>B state</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>O z</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Terminal state
String accepted
Moore output (stable for following period)
Conversion to Mealy Machine

- Recall difference between Mealy and Moore machine is in generation of output
- Note state table for design example

<table>
<thead>
<tr>
<th>PS</th>
<th>AB</th>
<th>NS</th>
<th>x=0</th>
<th>x=1</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>10</td>
<td>01</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>00</td>
<td>11</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>10</td>
<td>01</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Next states are the same, but output is different.
## Conversion to Mealy Machine

- Assign Moore output (state) to Mealy transition

<table>
<thead>
<tr>
<th>PS</th>
<th>AB</th>
<th>x=0</th>
<th>x=1</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>10</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>00</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>10</td>
<td>01</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PS</th>
<th>x=0</th>
<th>x=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>00,0</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>10,0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>00,0</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>10,0</td>
</tr>
</tbody>
</table>
## Conversion to Mealy Machine

- Note that rows 1 and 3 of the state table are identical
- Identical rows can be combined into a single state

<table>
<thead>
<tr>
<th>PS</th>
<th>AB</th>
<th>NS $x=0$</th>
<th>NS $x=1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>00,0</td>
<td>01,0</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>10,0</td>
<td>01,0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>00,0</td>
<td>11,1</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>10,0</td>
<td>01,0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PS</th>
<th>AB</th>
<th>NS $x=0$</th>
<th>NS $x=1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>00,0</td>
<td>01,0</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>10,0</td>
<td>01,0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>00,0</td>
<td>01,1</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>10,0</td>
<td>01,1</td>
</tr>
</tbody>
</table>
Conversion to Mealy Machine

- Because outputs in a Mealy machine are associated with the transition and not the next state, states 1 and 3 can be combined
  - Call combined state “state 1” and eliminate state 3
    - New state 1 entered with output of 0 from old state 1
    - New state 1 entered with output of 1 from unchanged state 2
  - Technically, no longer a finite state recognizer because of Mealy implementation
    - No longer an acceptance “state”
Conversion to Mealy Machine

- State diagram
Conversion to Mealy Machine

- Next state and output maps

<table>
<thead>
<tr>
<th>A'B</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ A^+ = x'B \]
\[ B^+ = x \]
\[ z = xA \]
Mealy Machine – Verilog Implementation

- **Verilog Code**
  - **Output assigned to declaratively (wire)**
  - **Implementation with case statement**

```verilog
module mealy (clk, x, z);
  input clk, x;
  output z;
  reg [1:0] state;
  parameter [1:0] zero = 2'b00, one = 2'b01, two = 2'b10;
  assign z = x & state[1];
  always @(posedge clk)
    case (state)
      zero: if (x == 0)
        state <= zero;
      else
        state <= one;
      one: if (x == 0)
        state <= two;
      else
        state <= one;
      two: if (x == 0)
        state <= zero;
      else
        state <= one;
      default:
        state <= zero;
    endcase
endmodule
```
Mealy Machine – Verilog Implementation

- Timing Simulation

Input sequence: 1 1 0 1 0 1 1 1 0 0

<table>
<thead>
<tr>
<th>Name</th>
<th>clk</th>
<th>50.0ns</th>
<th>100.0ns</th>
<th>150.0ns</th>
<th>200.0ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>[I] clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[I] x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[B] state</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>[O]z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

String accepted

Mealy output (valid on active clock edge)
Mealy Machine – Verilog Implementation

- Alternative Verilog Code
  - Implemented directly from next state equations for state variables A and B

```verilog
module mealy2 (clk, x, z);
  input clk, x;
  output z;
  reg A, B;
  assign z = x & A;
  always @ (posedge clk)
    begin
      A <= ~x & B;
      B <= x;
    end
endmodule
```
Mealy Machine - Verilog Implementation

- Timing Simulation (alternative code)

Input sequence: 1 1 0 1 0 1 1 1 0 0

<table>
<thead>
<tr>
<th>Name</th>
<th>50.0ns</th>
<th>100.0ns</th>
<th>150.0ns</th>
<th>200.0ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B.Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

string accepted
Mealy output (valid on active clock edge)
Conversion from Mealy to Moore

States that can be entered with different outputs (0 and 1 in this case) must be split.
Conversion from Mealy to Moore

States that can be entered with different outputs (0 and 1 in this case) must be split.
Conversion from Mealy to Moore

Input = 1

0/0

0

1/0

1

1/1

2

0/0

1/0

0/0

1/0

1A

0

1/0

1B

1

1/0
Conversion from Mealy to Moore
Conversion from Mealy to Moore

Output = 0

Output = 1
Conversion from Mealy to Moore

... the original Moore machine

Output = 0

Output = 1