Sequential Logic Implementation

- Models for representing sequential circuits
- Abstraction of sequential elements
- Finite state machines and their state diagrams
- Inputs/outputs
- Mealy, Moore, and synchronous Mealy machines
- Finite state machine design procedure
- Verilog specification
- Deriving state diagram
- Deriving state transition table
- Determining next state and output functions
- Implementing combinational logic

Mealy vs. Moore Machines

- Moore: outputs depend on current state only
- Mealy: outputs depend on current state and inputs
- Ant brain is a Moore Machine
- Output does not react immediately to input change
- We could have specified a Mealy FSM
  - Outputs have immediate reaction to inputs
  - As inputs change, so does next state, doesn’t commit until clocking event

Specifying Outputs for a Moore Machine

- Output is only function of state
- Specify in state bubble in state diagram
- Example: sequence detector for 01 or 10

Specifying Outputs for a Mealy Machine

- Output is function of state and inputs
- Specify output on transition arc between states
- Example: sequence detector for 01 or 10

Comparison of Mealy and Moore Machines

- Mealy Machines tend to have less states
  - Different outputs on arcs (n^2) rather than states (n)
  - Moore Machines are safer to use
  - Outputs change at clock edge (always one cycle later)
  - In Mealy machines, input change can cause output change as soon as logic is done - a big problem when two machines are interconnected - synchronous feedback
  - Mealy Machines react faster to inputs
  - React in same cycle - don’t need to wait for clock
  - In Moore machines, more logic may be necessary to decode state into outputs - more gate delays after

Mealy and Moore Examples

- Recognize A,B = 0,1
- Mealy or Moore?
Mealy and Moore Examples (cont'd)

- Recognize $A, B = 1, 0$ then 0, 1
- Mealy or Moore?

Registered Mealy Machine (Really Moore)

- Synchronous (or registered) Mealy Machine
- Registered state AND outputs
- Easy to implement in programmable logic
- Moore Machine with no output decoding
- Outputs computed on transition to next state rather than after entering
- View outputs as expanded state vector

Registered Mealy Machine

<table>
<thead>
<tr>
<th>Inputs</th>
<th>States</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>zero</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>one</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>two</td>
<td>2</td>
</tr>
</tbody>
</table>

Verilog FSM - Reduce 1's Example

- Change the first 1 to 0 in each string of 1's
- Example Moore machine implementation

```
// State assignment
parameter zero = 0, one = 1, two = 2;
module reduce (out, clk, reset, in);
  output out;
  input clk, reset, in;
  reg out;
  reg [1:0] state;
  // state register
  reg [1:0] next_state;
  // next state register
  always @(in or state)
  case (state)
    zero: begin // last input was a zero
      out = 0;
      if (in) next_state = one;
      else next_state = zero;
    end
    one: begin // we've seen one 1
      out = 0;
      if (in) next_state = two;
      else next_state = zero;
    end
    two: begin // we've seen at least 2 ones
      out = 1;
      if (in) next_state = two;
      else next_state = zero;
    end
    default: begin // in case we reach a bad state
      out = 0;
      next_state = zero;
    end
  case
endmodule
```

Moore Verilog FSM (cont'd)

```
always @(posedge clk)
  if (reset) state <= zero;
  else state <= next_state;
endmodule
```

Moore Verilog FSM for Reduce-1s Example

```
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  reg [1:0] state;
  // state register
  reg [1:0] next_state;
  // next state register
  parameter zero = 0, one = 1;
  always @(in or state)
    case (state)
      zero: begin // last input was a zero
        if (in) state = one;
        else state = zero;
        next_state = zero;
      end
      one: begin // we've seen one 1
        next_state = one;
        out = 1;
        if (in) state = zero;
        else next_state = zero;
      end
      two: begin // we've seen at least 2 ones
        next_state = two;
        out = 0;
        if (in) state = two;
        else next_state = zero;
      end
      default: begin // in case we reach a bad state
        out = 0;
        next_state = zero;
      end
    case
  endcase
endmodule
```
Synchronous Mealy Verilog FSM for Reduce-1s Example

```verilog
code
module reduce (clk, reset, in, out);
input clk, reset, in;
output out;
reg state;
reg next_state;
reg next_out;

always @(in or state)
begin
    case (state)
        0: begin
            next_state = one;
            if (in) next_state = one;
        end
        1: begin
            next_state = zero;
        end
        default: begin
            next_state = zero;
        end
    endcase

    // state register
    reg state;
    state <= next_state;

    // output
    out <= next_out;
endmodule
```

Example: Vending Machine

```
Example: Vending Machine (cont'd)

Suitable Abstract Representation

- Tabulate typical input sequences:
  - 1 nickel
  - 1 dime, nickel
  - 2 dimes

- Draw state diagram:
  - Denote N, D, reset
  - Output: open chute

- Assumptions:
  - Assume N and D asserted for one cycle
  - Each state has a self loop for N + D = 0 (no coin)
```

Example: Vending Machine (cont'd)

```
Example: Vending Machine (cont'd)

Minimize number of states - reuse states whenever possible

<table>
<thead>
<tr>
<th>Reset</th>
<th>present state</th>
<th>inputs</th>
<th>next state</th>
<th>output open chute</th>
</tr>
</thead>
<tbody>
<tr>
<td>----</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>----</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>----</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

symbolic state table
```
Example: Vending Machine (cont’d)

Uniquely Encode States

<table>
<thead>
<tr>
<th>Present State</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>D</th>
<th>N</th>
<th>open</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Q0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Q1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Q2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Q3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>zero</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>five</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ten</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>fifteen</td>
</tr>
</tbody>
</table>

Example: Vending Machine (cont’d)

One-hot Encoding

<table>
<thead>
<tr>
<th>Present State</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>D</th>
<th>N</th>
<th>open</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Q0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Q1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Q2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Q3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>zero</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>five</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ten</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>fifteen</td>
</tr>
</tbody>
</table>

Example: Vending Machine (cont’d)

Mapping to Logic

Equivalent Mealy and Moore State Diagrams

Mealy machine
- outputs associated with transitions
- Mealy machine
- outputs associated with state transitions

Moore Verilog FSM for Vending Machine

module vending (open, Clk, Reset, N, D);
input Clk, Reset, N, D; output open;
reg open; reg state; // state register
reg next_state;
parameter zero = 0, five = 1, ten = 2, fifteen = 3;
always @(posedge clk)
if (reset) state <= zero;
else state <= next_state;
endmodule

Mealy Verilog FSM for Vending Machine

module vending (open, Clk, Reset, N, D);
input Clk, Reset, N, D; output open;
reg open; reg state; // state register
reg next_state; reg next_open;
parameter zero = 0, five = 1, ten = 2, fifteen = 3;
always @(posedge clk)
if (reset) begin
    state <= zero;
    open <= 0;
end
else begin
    next_state = zero; next_open = 0;
    if (!Reset) next_state = fifteen;
    else if (N) next_state = ten;
    // More cases...
end
Example: Traffic Light Controller

- A busy highway is intersected by a little used farmroad.
- Detectors C sense the presence of cars waiting on the farmroad.
- If vehicle on farmroad, highway lights go from Green to Yellow to Red, allowing the farmroad lights to become green.
- These stay green only as long as a farmroad car is detected but never longer than a set interval.
- When these are met, farm lights transition from Green to Yellow to Red, allowing highway to return to green.
- Even if farmroad vehicles are waiting, highway gets at least a set interval at green.

Assume you have an interval timer that generates:
- A short time pulse (TS) and
- A long time pulse (TL),
- in response to a set (ST) signal.
- TS is to be used for timing yellow lights and TL for green lights.

Traffic Light Controller (cont’d)

Tabulation of Inputs and Outputs

<table>
<thead>
<tr>
<th>Input Description</th>
<th>Output Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>F</td>
</tr>
<tr>
<td>Present state</td>
<td>C</td>
</tr>
<tr>
<td>TL</td>
<td>ST</td>
</tr>
<tr>
<td>TS</td>
<td></td>
</tr>
<tr>
<td>TL+ST</td>
<td></td>
</tr>
</tbody>
</table>

Tabulation of unique states - some light configurations imply others

State Description

- S0: highway green (farm road red)
- S1: highway yellow (farm road red)
- S2: farm road green (highway red)
- S3: farm road yellow (highway red)

Example: Traffic Light Controller (cont’d)

Tabulation of Inputs and Outputs

<table>
<thead>
<tr>
<th>Input Description</th>
<th>Output Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Present state</td>
<td>Next state</td>
</tr>
<tr>
<td>Present state</td>
<td>Outputs</td>
</tr>
<tr>
<td>TL</td>
<td>ST</td>
</tr>
<tr>
<td>TS</td>
<td></td>
</tr>
<tr>
<td>TL+ST</td>
<td></td>
</tr>
</tbody>
</table>

Consider state assignments

- Generate state table with symbolic states
- Consider state assignments

Traffic Light Controller Verilog

```verilog
module traffic (ST, Clk, Reset, C, TL, TS);
    reg [3:0] state, next_state, ST;
    reg C, TL, TS;
    always @(C or TL or TS or state)
        case (state)
            S0 : if (Reset) begin state <= S0; ST <= 0; end
                else if (TL || C) begin
                    next_state = S1; next_ST = 1;
                end
            S1 : if (Reset) begin state <= S1; ST <= 1; end
                else if (TS') begin
                    next_state = S2; next_ST = 1;
                end
            S2 : if (Reset) begin state <= S2; ST <= 0; end
                else if (TL+C') begin
                    next_state = S3; next_ST = 1;
                end
            S3 : if (Reset) begin state <= S3; ST <= 0; end
                else if (Reset) begin state <= S3; ST <= 0; end
        endcase
    next_state = S0; ST = 0;
endmodule
```

Example: Traffic Light Controller (cont’d)

Highway/farm road intersection

Farm road

Highway

Detector

Car sensors

Controller

Traffic Lights

Traffic Controller Implementation

Traffic Controller Implementation

Traffic Controller Implementation

Traffic Controller Implementation
Logic for Different State Assignments

- SA1
  \[ NS1 = C • TL' • PS1 • PS0 + TS • PS1 • PS0 + TL' • PS1 • PS0' + PS1 • PS0 + C' • TL • PS1 • PS0 \]
  \[ ST = C • TL • PS1' • PS0' + TS' • PS1 + C' • PS1' • PS0 \]
  \[ H1 = PS0 \]
  \[ F1 = PS0' \]

- SA2
  \[ NS2 = C • TL • PS1 + TS • PS1 + TL • PS1 + PS0 + C' • TL' • PS1 + TL' • PS1 + PS0' \]
  \[ NS3 = C' • PS2 + TL • PS2 + TL • PS2 + PS3 + TS • PS3 + TL' • PS0 + PS3 + TL • PS2 + PS3 \]
  \[ ST = C • TL • PS1 + C' • TL' • PS2 + TL • PS2 + PS3 + TS • PS3 + TL' • PS0 + PS3 + TL • PS2 + PS3 \]
  \[ H1 = PS0 \]
  \[ F1 = PS0' \]

Vending Machine Example Revisited

(PLD mapping)

- D0 = \( \text{reset'}(Q0'N + Q0N' + Q1N + Q1D) \)
- D1 = \( \text{reset'}(Q1 + D + Q0N) \)
- OPEN = \( Q0'N + Q0N' + Q1N + Q1D \)

Vending Machine (Retimed PLD Mapping)

- OPEN = \( \text{reset'}(Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D) \)

Vending Machine (cont’d)

- OPEN = Q1Q0 creates a combinational delay after Q1 and Q0 change
- This can be corrected by retiming, i.e., move flip-flops and logic through each other to improve delay
- OPEN = \( \text{reset'}(Q1 + D + Q0N + Q0Q1'N + Q0N' + Q0Q1D) \)
- Implementation now looks like a synchronous Mealy machine
- Common for programmable devices to have FF at end of logic

Sequential Logic Implementation

Summary

- Models for representing sequential circuits
  - Abstraction of sequential elements
  - Finite state machines and their state diagrams
  - Inputs/outputs
  - Mealy, Moore, and synchronous Mealy machines

- Finite state machine design procedure
  - Verilog specification
  - Deriving state diagram
  - Deriving state transition table
  - Determining next state and output functions
  - Implementing combinational logic