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
  - 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 - asynchronous 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 $A = 1$, $B = 0$ then $Q$.

- Mealy or Moore?

- Registered state and output.

- Avoids 'glitchy' outputs.

- Easy to implement in PLDs.

- 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 (Really Moore)**

- Synchronous (or registered) Mealy machine.

- Registered state and output.

- Avoids 'glitchy' outputs.

- Easy to implement in PLDs.

- Moore machine with no output decoding.

- Outputs computed on transition to next state rather than after entering.

- View outputs as expanded state vector.

---

**Verilog FSM - Reduce 1s Example**

- Change the first 1 to 0 in each string of 1s.

- Example Moore machine implementation.

---

**Moore Verilog FSM (cont’d)**

- Always with output.

- Example Moore machine implementation.

---

**Moore Verilog FSM (cont’d)**

- Always with output.

- Example Moore machine implementation.

---

**Mealy Verilog FSM for Reduce-1s Example**

- Always with output.

- Example Mealy machine implementation.

---
Synchronous Mealy Verilog FSM for Reduce-1s Example

module reduce (clk, reset, in, out);
input clk, reset, in; output out;
reg out; reg state; // state register
reg next_state;
reg next_out;
parameter zero = 0, one = 1;
always @(in or state)
case (state)
zero: begin // last input was a zero
if (in) next_state = one;
else next_state = zero;
next_out = 0;
end
one: // we've seen one 1
if (in) begin
next_state = one;
next_out = 1;
end else begin
next_state = zero;
next_out = 0;
end
endcase
always @(posedge clk)
if (reset) begin
state <= zero;
out <= 0;
end else begin
state <= next_state;
out <= next_out;
end
endmodule

Example: Vending Machine

- Release item after 15 cents are deposited
- Single coin slot for dimes, nickels
- No change

Example: Vending Machine (cont’d)

Example: Vending Machine (cont’d)

Example: Vending Machine (cont’d)

Example: Vending Machine (cont’d)
Example: Vending Machine (cont’d)

- One-hot Encoding

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>0011</td>
<td>0011</td>
</tr>
<tr>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>0101</td>
<td>0101</td>
</tr>
<tr>
<td>0110</td>
<td>0110</td>
</tr>
<tr>
<td>0111</td>
<td>0111</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>1001</td>
<td>1001</td>
</tr>
<tr>
<td>1010</td>
<td>1010</td>
</tr>
<tr>
<td>1011</td>
<td>1011</td>
</tr>
<tr>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>1101</td>
<td>1101</td>
</tr>
<tr>
<td>1110</td>
<td>1110</td>
</tr>
<tr>
<td>1111</td>
<td>1111</td>
</tr>
</tbody>
</table>

- Parameter definitions

```verilog
define parameter:
zero = 0
five = 1
ten = 2
fifteen = 3
```

- Initial state

```verilog
next_state = value
```

- State transition matrix

```verilog
Moore Verilog FSM for Vending Machine
```
Example: Traffic Light Controller (cont')

II Tabulation of Inputs and Outputs

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Outputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Detect vehicle on farm road</td>
<td>FG, FY, FR</td>
<td>Assert green/yellow/red highway lights</td>
</tr>
<tr>
<td>TL</td>
<td>Long time interval expired</td>
<td>HG, HY, HR</td>
<td>Assert green/yellow/red highway lights</td>
</tr>
<tr>
<td>TS</td>
<td>Short time interval expired</td>
<td>ST</td>
<td>Start timing a short or long interval</td>
</tr>
</tbody>
</table>

II Tabulation of unique states - some light configurations imply others

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>Highway green (farm road red)</td>
</tr>
<tr>
<td>S1</td>
<td>Highway yellow (farm road red)</td>
</tr>
<tr>
<td>S2</td>
<td>Farm road green (highway red)</td>
</tr>
<tr>
<td>S3</td>
<td>Farm road yellow (highway red)</td>
</tr>
</tbody>
</table>

Traffic Light Controller Verilog

```
module traffic (ST, Clk, Reset, C, TL, TS);
input Clk, Reset, C, TL, TS; output ST;
reg ST;
reg state;
reg next_state; reg next_ST;
parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3;
always @(C or TL or TS or state)
case (state)
  S0: if (!(TL && C)) begin
    next_state = S0; next_ST = 0;
  end
  else if (TL || C) begin
    next_state = S1; next_ST = 1;
  end
  ...
endcase
always @(posedge Clk)
if (Reset) begin state <= S0; ST <= 0; end
else begin state <= next_state; ST <= next_ST; end
endmodule
```

Logic for Different State Assignments

**SA1**

\[
\begin{align*}
N0 &= C \cdot TL + P0 + C' \cdot TL + P0 + TL \cdot P0 + P0 \\
N1 &= C \cdot TL + P0 + C' \cdot TL + P0 + TL \cdot P0 + P0 \\
F1 &= P0 \\
Q1 &= P0 + P0 \\
\end{align*}
\]

**SA2**

\[
\begin{align*}
N0 &= C \cdot TL + P0 + C' \cdot TL + P0 + TL \cdot P0 + P0 \\
N1 &= C \cdot TL + P0 + C' \cdot TL + P0 + TL \cdot P0 + P0 \\
F1 &= P0 \\
Q1 &= P0 + P0 \\
\end{align*}
\]

**SA3**

\[
\begin{align*}
N0 &= C \cdot TL + P0 + C' \cdot TL + P0 + TL \cdot P0 + P0 \\
N1 &= C \cdot TL + P0 + C' \cdot TL + P0 + TL \cdot P0 + P0 \\
F1 &= P0 \\
Q1 &= P0 + P0 \\
\end{align*}
\]

Vending Machine Example Revisited (PLD mapping)

\[
\begin{align*}
D0 &= \text{reset}'(Q0'N + Q0N' + Q1N + Q1D) \\
D1 &= \text{reset}'(Q1 + D + Q0N) \\
OPEN &= Q1Q0 \\
\end{align*}
\]
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 = reset’(Q1D + Q0NQ0N’ + Q1N + Q1D) = reset’(Q1Q0N’ + Q1N + Q0’ND + Q0N’D)
- 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
  - Deriving state diagram
  - Deriving state transition table
  - Determining next state and output functions
  - Implementing combinational logic