Last Lecture
- Finite State Machines

Today
- Moore and Mealy Machines

Counter/shift-register model
- Values stored in registers represent the state of the circuit
- Combinational logic computes:
  - next state
    - function of current state and inputs
  - outputs
    - values of flip-flops

General state machine model
- Values stored in registers represent the state of the circuit
- Combinational logic computes:
  - next state
    - function of current state and inputs
  - outputs
    - function of current state and inputs (Mealy machine)
    - function of current state only (Moore machine)

State machine model (cont’d)
- States: S₁, S₂, ..., Sₖ
- Inputs: I₁, I₂, ..., Iₘ
- Outputs: O₁, O₂, ..., Oₙ
- Transition function: Fₛ(Sᵢ, Iⱼ)
- Output function: Fₒ(Sᵢ) or Fₒ(Sᵢ, Iⱼ)
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 may occur if one isn’t careful
- 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 clock edge

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
Registered Mealy machine (really Moore)

- Synchronous (or registered) Mealy machine
- registered state AND outputs
- 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

Example: vending machine

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

Example: vending machine

- Suitable abstract representation
- tabulate typical input sequences:
  - 3 nickels
  - nickel, dime
  - dime, nickel
  - two dimes
- draw state diagram:
  - inputs: 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

- Minimize number of states - reuse states whenever possible

Symbolic state table:

<table>
<thead>
<tr>
<th>Present state</th>
<th>Inputs</th>
<th>Next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0¢</td>
<td>0</td>
<td>0</td>
<td>0¢</td>
</tr>
<tr>
<td>0¢</td>
<td>1</td>
<td>10¢</td>
<td>0</td>
</tr>
<tr>
<td>5¢</td>
<td>0</td>
<td>5¢</td>
<td>0</td>
</tr>
<tr>
<td>5¢</td>
<td>1</td>
<td>10¢</td>
<td>0</td>
</tr>
<tr>
<td>10¢</td>
<td>0</td>
<td>10¢</td>
<td>0</td>
</tr>
<tr>
<td>10¢</td>
<td>1</td>
<td>15¢</td>
<td>0</td>
</tr>
<tr>
<td>15¢</td>
<td>0</td>
<td>15¢</td>
<td>1</td>
</tr>
<tr>
<td>15¢</td>
<td>1</td>
<td>15¢</td>
<td>0</td>
</tr>
</tbody>
</table>
Example: vending machine (cont’d)

- Uniquely encode states

<table>
<thead>
<tr>
<th>Present state</th>
<th>Inputs</th>
<th>Next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 Q0 D N</td>
<td>D1 D0</td>
<td>open</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>0 1</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>1 1</td>
<td>1 1</td>
<td></td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 0</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1 0</td>
<td>1 1</td>
<td></td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>1 1</td>
<td>1 1</td>
<td></td>
</tr>
<tr>
<td>1 1 - -</td>
<td>1 1</td>
<td>1 1</td>
<td></td>
</tr>
</tbody>
</table>

Example: Moore implementation

- Mapping to logic

Example: vending machine (cont’d)

- One-hot encoding

<table>
<thead>
<tr>
<th>Present state</th>
<th>Inputs</th>
<th>Next state output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q3 Q2 Q1 Q0</td>
<td>D N</td>
<td>D3 D2 D1 D0 open</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 0</td>
<td>0 0 0 1 0</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1</td>
<td>0 0 1 0 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 1</td>
<td>1 1 1 1 1</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1</td>
<td>0 1 0 0 0</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>1 1</td>
<td>1 1 1 1 1</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 1</td>
<td>1 1 1 1 1</td>
</tr>
</tbody>
</table>

Equivalent Mealy and Moore state diagrams

- Moore machine
  - outputs associated with state

- Mealy machine
  - outputs associated with transitions
Example: Mealy implementation

D0 = Q0'N + Q0N' + Q1N + Q1D
D1 = Q1 + D + Q0N
OPEN = Q1Q0 + Q1N + Q1D + Q0D

make sure OPEN is 0 when reset
- by adding AND gate

Vending machine: Moore to synch. Mealy

- OPEN = Q1Q0 creates a combinational delay after Q1 and Q0 change in Moore implementation
- This can be corrected by retiming, i.e., move flip-flops and logic through each other to improve delay
- OPEN.d = (Q1 + D + Q0N)(Q0'N + Q0N' + Q1N + Q1D) = Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D

Vending machine: Mealy to synch. Mealy

- OPEN.d = Q1Q0 + Q1N + Q1D + Q0D
- OPEN.d = (Q1 + D + Q0N)(Q0'N + Q0N' + Q1N + Q1D)
  = Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D
Mealy and Moore examples

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

HDLs and Sequential Logic

- Flip-flops
  - representation of clocks - timing of state changes
  - asynchronous vs. synchronous
- FSMs
  - structural view (FFs separate from combinational logic)
  - behavioral view (synthesis of sequencers – not in this course)
- Data-paths = data computation (e.g., ALUs, comparators) + registers
  - use of arithmetic/logical operators
  - control of storage elements

Mealy and Moore examples (cont’d)

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

Example: reduce-1-string-by-1

- Remove one 1 from every string of 1s on the input
Verilog FSM - Reduce 1s example

Moore machine

```verilog
module reduce (clk, reset, in, out);
    input clk, reset, in;
    output out;
    parameter zero  = 2'b00;
    parameter one1  = 2'b01;
    parameter two1s = 2'b10;
    reg out;
    reg [2:1] state; // state variables
    always @(posedge clk)
        if (reset) state = zero;
        else state = next_state;
    always @(in or state)
        case (state)
            zero: // last input was a zero
                begin
                    if (in) next_state = one1;
                    else next_state = zero;
                end
            one1: // we've seen one 1
                begin
                    if (in) next_state = two1s;
                    else next_state = zero;
                end
            two1s: // we've seen at least 2 ones
                begin
                    if (in) next_state = two1s;
                    else next_state = zero;
                end
        endcase
endmodule
```

Mealy Verilog FSM

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

Synchronous Mealy Machine

```verilog
module reduce (clk, reset, in, out);
    input clk, reset, in;
    output out;
    reg out;
    reg state; // state variables
    always @(posedge clk)
        if (reset) state = zero;
        else
            case (state)
                zero: // last input was a zero
                    begin
                        out = 0;
                        if (in) state = one;
                        else state = zero;
                    end
                one: // we've seen one 1
                    begin
                        if (in) begin
                            state = one; out = 1;
                        end else begin
                            state = zero; out = 1;
                        end
                    end
            endcase
endmodule
```
Finite state machines 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
- Hardware description languages