Quiz Review

1. In this problem we want to design a three bit Grey code counter. This counter will cycle through the following states: 000, 001, 010, 011, 100, 101, 110, 111, in the order listed above. Further the counter should provide a reset input (call it R) which if it is set will return the counter to the state 000.

   a) Draw a state transition diagram for this counter. Each state should be labeled by the three bits listed above. You should label the transitions between these states with arrows labeled by the value R takes.

   b) Create and fill out a state transition table for this counter. It should have four inputs and three outputs.

   c) Draw a circuit which implements the counter and changes state at the positive edge of the clock. Your circuit should have as input R and as output the bits of the counter, S1, S2, and S3. You are allowed to use any-fan in AND and OR gates, inverters as well as

   

   ```
   D Q
   A
   Flip flop(s)
   Clock
   ```
Comparison of Mealy and Moore machines (cont’d)

- **Moore**

- **Mealy**

- **Synchronous Mealy**

FSM design

- **FSM-design procedure**
  1. State diagram and state-transition table
  2. State minimization
  3. State assignment (or state encoding)
  4. Minimize next-state logic
  5. Implement the design

Example: Sequence detector

- Design a circuit to detect 3 or more 1’s in a bit string
  - Assume Moore machine
  - Assume D flip-flops
  - Assume flip-flops have a reset

1. State diagram and state-transition table

```
<table>
<thead>
<tr>
<th>reset</th>
<th>current state</th>
<th>input</th>
<th>next state</th>
<th>current output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A/0</td>
<td>0</td>
<td>B/0</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>A/0</td>
<td>1</td>
<td>C/0</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>B/0</td>
<td>0</td>
<td>C/0</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>B/0</td>
<td>1</td>
<td>D/1</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>C/0</td>
<td>0</td>
<td>D/1</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>D/1</td>
<td>1</td>
<td></td>
<td>D</td>
</tr>
</tbody>
</table>
```
2. State minimization & 3. State encoding

- State diagram is already minimized
- Try a binary encoding

4. Minimize next-state logic

<table>
<thead>
<tr>
<th>MSB+</th>
<th>M</th>
<th>LSB+</th>
<th>M</th>
<th>OUT+</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0 0 1 0</td>
<td></td>
</tr>
</tbody>
</table>

MSB+ = L'In + MIn
LSB+ = L'In + MIn
Out+ = ML

Notation
M := MSB
L := LSB
In := Input

5. Implement the design

Example: vending machine

- Release item after 15 cents are deposited
- Single coin slot for dimes, nickels
- No change
Example: vending machine (cont’d)

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)

Minimize number of states - reuse states whenever possible

Example: Moore implementation

Mapping to logic

D1 = Q1 + D + Q0 N
D0 = Q0' + Q0 N' + Q1 N + Q1 D
OPEN = Q1 Q0
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 0 1</td>
<td>0 0</td>
<td>0 0 0 1 0</td>
</tr>
<tr>
<td>0 1 0 0 1</td>
<td>0 0</td>
<td>0 0 1 0 0</td>
</tr>
<tr>
<td>1 0 0 0 0</td>
<td>0 0</td>
<td>0 0 0 0 0</td>
</tr>
</tbody>
</table>

D0 = Q0 D' N'
D1 = Q0 N + Q1 D' N'
D2 = Q0 D + Q1 N + Q2 D' N'
D3 = Q1 D + Q2 D + Q2 N + Q3
OPEN = Q3

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

Equivalent Mealy and Moore state diagrams

Moore machine
- Outputs associated with state

Mealy machine
- Outputs associated with transitions
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
- Implementation now looks like a synchronous Mealy machine
  - it is common for programmable devices to have FF at end of logic

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?

Mealy and Moore examples (cont’d)
- Recognize A,B = 1,0 then 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

Example: reduce-1-string-by-1

- Remove one 1 from every string of 1s on the input

```
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
  reg [2:1] next_state;

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

  state assignment
  (easy to change, if in one place)

  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

  always @(state)
  case (state)
    zero: out = 0;
    one1: out = 0;
    twols: out = 1;
  endcase
endmodule
```

Verilog FSM - Reduce 1s example

- Moore machine

```
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
  reg [2:1] next_state;

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

  state assignment
  (easy to change, if in one place)

  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

  always @(state)
  case (state)
    zero: out = 0;
    one1: out = 0;
    twols: out = 1;
  endcase
endmodule
```
Mealy Verilog FSM

module reduce (clk, reset, in, out);
input clk, reset, in;
output out;
reg out;
reg state; // state variables
reg next_state;

always @(posedge clk)
  if (reset) state = zero;
  else state = next_state;
always @(in or state)
  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
      if (in) begin
        next_state = one; out = 1;
      end else begin
        next_state = zero; out = 0;
      end
  endcase
endmodule

Synchronous Mealy Machine

module reduce (clk, reset, in, out);
input clk, reset, in;
output out;
reg out;
reg state; // state variables
reg next_state;

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
        if (in) begin
          state = one; out = 1;
        end else begin
          state = zero; out = 0;
        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