EECS 151/251A Homework 4

Due Monday, Feb 24th, 2020

For this HW Assignment

You will be asked to write several Verilog modules as part of this HW assignment. You will need to test your modules by running them through a simulator. As shown in discussion 2, a highly suggested simulator is [https://www.edaplayground.com](https://www.edaplayground.com) which is a free, online, Verilog simulator.

Warning: As per our EECS151 “no register inference policy”, you will need to use the register library in EECS151.v when using registers in your Verilog. EECS151.v is located at [https://inst.eecs.berkeley.edu/~eecs151/sp20/files/lib/EECS151.v](https://inst.eecs.berkeley.edu/~eecs151/sp20/files/lib/EECS151.v). We will only accept solutions using this library.

You can import this library by adding ‘include "EECS151.v"' to your Verilog code.

Examples of creating a testbench can be found in discussion 2 slides [https://inst.eecs.berkeley.edu/~eecs151/sp20/files/discussion2.pdf](https://inst.eecs.berkeley.edu/~eecs151/sp20/files/discussion2.pdf)

Problem 1. Vending Machine FSM

You need to design a vending machine controller that should exhibit the following behavior:

- The vending machine delivers a package of gum after it has received 15 cents in coins.
- The machine has a single coin slot that accepts only nickels and dimes, one coin at a time.
- A sensor indicates to the controller whether a nickel or dime has been inserted in the coin slot.
- The controller’s output causes a single package of gum to be released when it has received at least 15 cents, and the controller is reset.
- The machine does not give change. For example, if two dimes are inserted, it will release the gum and reset.

For this problem:

(a) Design a Moore finite state machine for the vending machine.
(b) Draw a state transition diagram for your FSM. Define your states, inputs, and outputs.

Solution:

(c) Draw a circuit diagram consisting of logic gates and registers for your system. Don’t worry about optimal state encoding. Assume your Flip-Flop registers have RESET pins and can reset to any value. Don’t worry about optimal state encoding.

Solution:
(d) Design it using behavioral Verilog (using case statements), and provide a testbench and test results.

Solution:

```verilog
'include "EECS151.v"
module VendingMachineFSM(N, D, open, reset, clk);
  output reg open;
  input N, D;
  input reset;
  input clk;
  wire [1:0] present_state,
  reg [1:0] next_state;

  // Defined state encoding:
  localparam S0 = 2'b00; // 0 cent
  localparam S1 = 2'b01; // 5 cent
  localparam S2 = 2'b10; // 10 cent
  localparam S3 = 2'b11; // 15 cent

  REGISTER_R #(.N(2), .INIT(S0)) vending_state_reg
    .q(present_state), .d(next_state), .rst(reset), .clk(clk));

  always @(present_state or N or D)
    open = 1b'0;
  case (present_state)
    S0 : begin
      if (N == 1)
        next_state = S1;
      else if (D == 1)
        next_state = S2;
```
Problem 2. Traffic Light Controller FSM

There is a busy intersection in Berkeley at Martin Luther King Jr Way (MLK Way) and University Avenue (Uni Ave), as shown below. There are detectors on MLK Way that give a signal \( C \) if there is a vehicle waiting to cross Uni Ave. The Traffic Light Controller should exhibit the following behavior:

- The lights in the Uni Ave direction should be green as long as there are no cars on MLK Way.
- When a car is detected on MLK Way, the Uni Ave lights should go from yellow to red. MLK Way will be green when Uni Ave is red.
- MLK Way will stay green as long as there are cars detected or until a set interval has passed, since Uni Ave is busier and needs to return to green. If these conditions are met, MLK Way will go from green to yellow to red. Uni Ave will be green when MLK Way is red.
- Even if there are cars detected on MLK Way, Uni Ave will stay green for a set interval.
- There is an external timer outside the controller that takes in a control signal \( ST \) (set timer) that asserts a signal \( TS \) after a short interval (for yellow to red timing) and a signal \( TL \) after a longer time interval (for green lights). The timer is reset when \( ST \) is asserted.

```verilog
else
    next_state = S0;
end
S1 : begin
    if (N == 1)
        next_state = S2;
    else if (D == 1)
        next_state = S3;
    else
        next_state = S1;
end
S2 : begin
    if (N == 1 || D == 1)
        next_state = S3;
    else
        next_state = S2;
end
S3 : begin
    open = 1b'1;
    next_state = S0;
end
default: begin
    next_state = present_state;
end
endcase
endmodule
```
For this problem:

(a) Design a Mealy finite state machine for the Traffic Light Controller.

<table>
<thead>
<tr>
<th>Solution:</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Transition Table</td>
</tr>
</tbody>
</table>

(b) Draw a state transition diagram for your FSM. Define your states, inputs, and outputs.

<table>
<thead>
<tr>
<th>Solution:</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="State Transition Diagram" /></td>
</tr>
</tbody>
</table>

(c) Draw a circuit diagram consisting of logic gates and registers for your system. Assume your Flip-Flop registers have RESET pins and can reset to any value. Don’t worry about optimal state encoding.

<table>
<thead>
<tr>
<th>Solution:</th>
</tr>
</thead>
</table>
(d) Design it in behavioral Verilog (using case statements), and provide a testbench and test results.

Solution:

```
`include "EECS151.v"
module TrafficControllerFSM(UNI_G, UNI_Y, UNI_R, MLK_G, MLK_Y, MLK_R, ST, TS, TL, C, reset, clk);
output reg UNI_G;
output reg UNI_Y;
output reg UNI_R;
output reg MLK_G;
output reg MLK_Y;
output reg MLK_R;
output reg ST;
input TS;
input TL;
input C;
input reset;
input clk;
wire [1:0] present_state,
reg [1:0] next_state;

// Defined state encoding:
localparam S0 = 2'b00; // uni_green, mlk_red
localparam S1 = 2'b01; // uni_yellow, mlk_red
localparam S2 = 2'b10; // mlk_green, uni_red
localparam S3 = 2'b11; // mlk_yellow, uni_red
```
REGISTER_R #(N(2), INIT(S0)) light_state_reg (.q(present_state),
\rightarrow .d(next_state), .rst(reset), .clk(clk));

always @(present_state or TS or TL or C)
case (present_state)
  S0 : begin
    if (TL == 1 && C == 1)
      next_state = S1;
      ST = 1;
      {UNI_G, UNI_Y, UNI_R, MLK_G, MLK_Y, MLK_R} = 6b'010001;
    else
      next_state = S0;
      ST = 0;
      {UNI_G, UNI_Y, UNI_R, MLK_G, MLK_Y, MLK_R} = 6b'100001;
  end
  S1 : begin
    if (TS == 1)
      next_state = S2;
      ST = 1;
      {UNI_G, UNI_Y, UNI_R, MLK_G, MLK_Y, MLK_R} = 6b'001100;
    else
      next_state = S1;
      ST = 0;
      {UNI_G, UNI_Y, UNI_R, MLK_G, MLK_Y, MLK_R} = 6b'010001;
  end
  S2 : begin
    if (TL == 1 || C == 0)
      next_state = S3;
      ST = 1;
      {UNI_G, UNI_Y, UNI_R, MLK_G, MLK_Y, MLK_R} = 6b'001010;
    else
      next_state = S2;
      ST = 0;
      {UNI_G, UNI_Y, UNI_R, MLK_G, MLK_Y, MLK_R} = 6b'001100;
  end
  S3 : begin
    if (TS == 0)
      next_state = S0;
      ST = 1;
      {UNI_G, UNI_Y, UNI_R, MLK_G, MLK_Y, MLK_R} = 6b'100001;
    else
      next_state = S3;
      ST = 0;
      {UNI_G, UNI_Y, UNI_R, MLK_G, MLK_Y, MLK_R} = 6b'001010;
  end
  default: begin
    next_state = present_state;
  end
(c) Re-do part (c), **For your state encoding, use one-hot encoding.** Re-draw a circuit diagram consisting of logic gates and registers for your system accordingly.

**Solution:**

![Circuit Diagram](image)

**Problem 3. Moore vs. Mealy Machine**

Design a finite state machine that asserts a single output whenever its input string has at least two 1’s in a row. The input takes in 1 bit at a time. For example, an input sequence of 0110111 would yield an output sequence of 0010011. Since the 2nd and 3rd bits are both 1’s, the 3rd output would be 1. Since the 4th, 5th, and 6th bits are 1’s, the 5th and 6th output would be 1’s. For this problem:

(a) **Draw diagrams for both a Moore and Mealy machine.**

**Solution:**

![Circuit Diagram](image)
(b) Define your states, inputs, and outputs.

**Solution:**

**Moore Machine:**

<table>
<thead>
<tr>
<th>PS</th>
<th>In</th>
<th>NS</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>00</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Mealy Machine:**

<table>
<thead>
<tr>
<th>PS</th>
<th>In</th>
<th>NS</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(c) Draw an expected waveform diagram.

**Solution:**

**Moore Machine:**
Mealy Machine:

(d) Design both in Verilog, and provide a testbench and test results.

Solution:

```verilog
`include "EECS151.v"

module StringCounterMoore(in, out, reset, clk);
    output out;
    input in;
    input reset;
    input clk;
    wire [1:0] present_state,
    reg [1:0] next_state;

    // Defined state encoding:
    localparam S0 = 2'b00; // zero 1's
    localparam S1 = 2'b01; // one 1
    localparam S2 = 2'b10; // two ones

    REGISTER_R #(.N(2), .INIT(S0)) vending_state_reg
    ↪ ( .q(present_state), .d(next_state), .rst(reset), .clk(clk));

    assign out = (present_state == S2); // at least two 1's

    always @(present_state or in)
    out = 1'b0;
    case (present_state)
        S0 : begin
            if (in == 1)
                next_state = S1;
            else
```
```verilog
next_state = S0;
end
S1 : begin
    if (in == 1)
        next_state = S2;
    else
        next_state = S0;
end
S2 : begin
    out = 1b'1;
    if (in == 1)
        next_state = S2;
    else
        next_state = S0;
end
default: begin
    next_state = S0;
end
derncase
endmodule

module StringCounterMealy(in, out, reset, clk);
output reg out;
input in;
input reset;
input clk;
wire [1:0] present_state,
reg [1:0] next_state;

// Defined state encoding:
localparam S0 = 2'b00; // zero 1's
localparam S1 = 2'b01; // one 1
REGISTER_R #(N(2), INIT(S0)) vending_state_reg
    (.q(present_state), .d(next_state), .rst(reset), .clk(clk));

always @(present_state or in)
    out = 1b'0;
case (present_state)
    S0 : begin
        if (in == 1)
            next_state = S1;
        else
            next_state = S0;
    end
    S1 : begin
        if (in == 1)
```
next_state = S1;
out = 1b'1;
else
  next_state = S0;
end
default: begin
  next_state = S0;
end
endcase
endmodule
Problem 4: State Reduction

Given the state transition diagram below, determine which states should be combined to determine the reduced state diagram. Use the implication chart method.

\[
\begin{array}{c}
S_0 = S_3 = S_4, \quad S_1 = S_5 \\
\end{array}
\]

Solution:
Problem 5: State Assignment

Given the state transition diagram below, select a good state assignment, justifying your answer in terms of the state assignment heuristics presented in class.

Solution:

High Priority: (B, C), (E, A)
Medium Priority: (A, D), (D, C)
Lowest Priority: 0/0: (E, D, C, B)
1/0: (B, C, A)

Problem 6: FSM Complexity: 251A only — Optional Challenge Question for 151

(a) For a 12 state FSM, how many unique state encoding assignments exist assuming that the state register holds no fewer than 4 bits, and no greater than 12? You may write out the answer in an analytical form, a numerical answer is not necessary.
Solution:

For an n-bit state register, the number of possible encodings is $2^n$ from which you can choose 12 from to encode the states. Therefore the total number of unique state encoding assignments can be written as

$$
\sum_{n=4}^{12} \binom{2^n}{12}
$$

(b) How many unique FSMs of 1 input, 1 output, and 5 states exist? You may write out the answer in an analytical form, a numerical answer is not necessary.

Solution:

From each of the 5 states, for each input of 1 and 0 there are 5 possible transitions. There are $2^5$ input/output combinations. Therefore the total number of unique FSMs can be written as

$$(5 \cdot 5)^5 \cdot 2^5$$