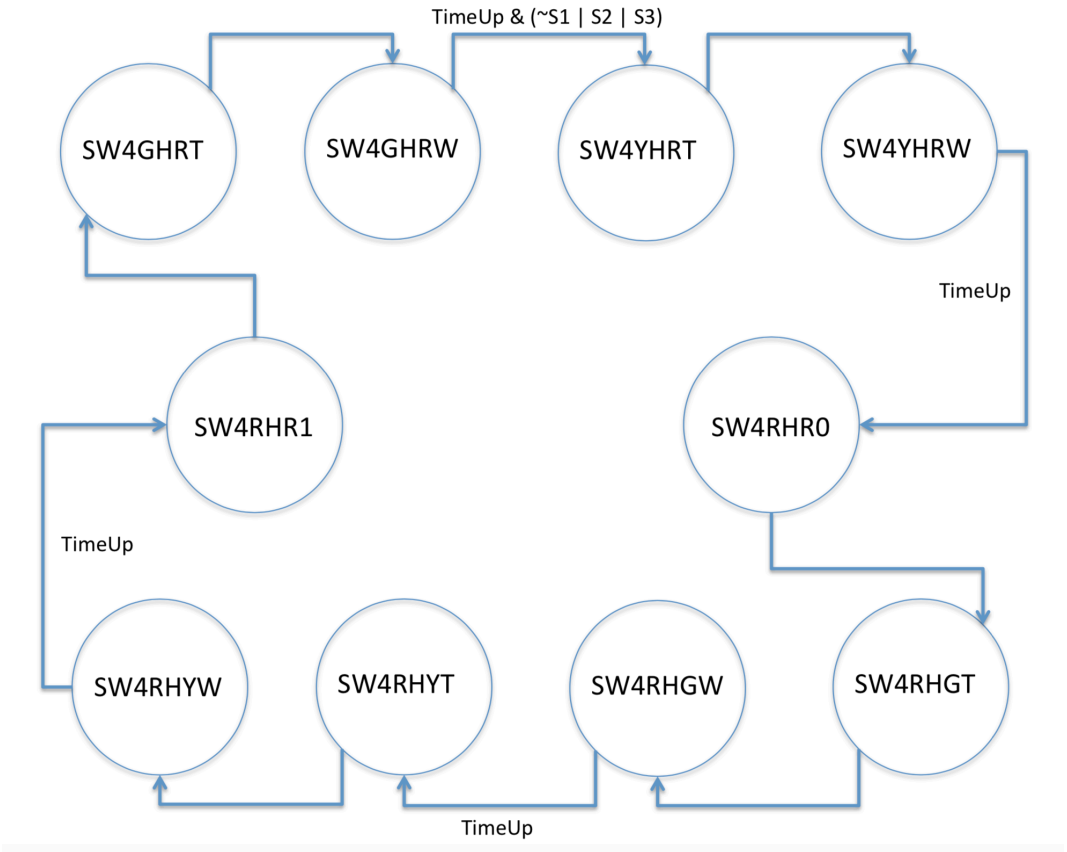
ECE 485/585
Fall 2019
Homework 1 Example Solution

This is an example solution for Homework 1. There are several ways to design the FSM depending upon whether you make use of one or more timers.

Using multiple timers is acceptable. However the solution here uses only one. This requires some additional states because we cannot load the timer with the value required for the next duration while simultaneously decrementing the counter for the current duration. My reasoning is that additional timers incur more logic than a couple of additional states. Moreover, repeatedly (re-)loading a timer will require more power.

I provided block diagrams of the counter and the top-level module and FSM previously so am not repeating those here. Although the state names may appear cryptic at first, they describe the state of the lights and timers. Thus SW4YHRT means SW 4th light is Yellow, Harrison lights are Red and we are *loading* the timer value, while SW4YHRW means SW 4th light is Yellow, Harrison lights are Red and we are *waiting* on the timer value.

To save space, the state transition diagram omits arrows for remaining in the same state if none of the conditions indicated for other transitions are satisfied.



In the code, we've followed the convention described in class an used in the TBird taillight design of using three procedural blocks: one sequential block (always posedge Clock) for the state register and reset, one combinational depending solely on State for the output logic (for Moore FSM), one combinational for the NextState generation logic.

To ensure correct synthesis (e.g. avoiding unwanted latches) there are assignments to Load, Decr, and Value before the case statement so that there is an assignment to these regardless of which path is taken. There is a default in the case statement to ensure that L1, L2, and L3 are assigned regardless of the path taken. Similarly, there is a default for the case statement determining NextState. Although the case statement covers all legal states and each case item ensures there is an assignment to NextState, this provides a little robustness. If the state register is corrupted this will ensure we transition to a valid state instead of remaining "stuck" in an illegal state.

I'm showing only a short testbench in order to demonstrate (again) cycle-based stimulus. My actual testbench incorporates assertions and generates thousands of cycles using random stimulus to ensure that your designs behave correctly.

I included the timer code here only for convenience – it allows me to compile and run the entire project using a single file.

```
`timescale 100ms/100ms
// time units will be 1/10 second, so clock period of 10 yields 1Hz clock


//
// Synthesizable counter module
//

module counter(clk, reset, load, value, decr, timeup);
input clk, reset, load, decr;
input [7:0] value;
output timeup;

reg [7:0] count;

assign timeup = (count == 0) ? 1 : 0;

always @(posedge clk)
  begin
  if (reset)
    count <= 0;
  else if (load)
    begin
    count <= value;
    end
  else if (decr && (count != 0))
    begin
    count <= count - 8'b1;
    end
  else
    count <= count;
  end
endmodule


module realfsm(Clock, Reset, S1, S2, S3, L1, L2, L3, Load, Count, Value, TimeUp);
input Clock;
input Reset;
input               // sensors for approaching vehicles
    S1,             // Northbound on SW 4th Avenue
    S2,             // Eastbound on SW Harrison Street
    S3;             // Westbound on SW Harrison Street

output reg [1:0]    // outputs for controlling traffic lights (01, 10, 11 for green, yellow, red)
    L1,             // light for NB SW 4th Avenue
    L2,             // light for EB SW Harrison Street
    L3;             // light for WB SW Harrison Street

output reg Load, Count; // Countdown timer's Load and Count controls
output reg [7:0] Value; // Value to be loaded into countdown timer
input TimeUp;           // Signal from countdown timer indicating reached 0

localparam
    SW4GHRT = 10'b0000000001,
    SW4GHRW = 10'b0000000010,
    SW4YHRT = 10'b0000000100,
    SW4YHRW = 10'b0000001000,
    SW4RHR0 = 10'b0000010000,
    SW4RHGT = 10'b0000100000,
    SW4RHGW = 10'b0001000000,
    SW4RHYT = 10'b0010000000,
    SW4RHYW = 10'b0100000000,
    SW4RHR1 = 10'b1000000000;

localparam
    GREEN  = 2'b01,
    YELLOW = 2'b10,
    RED    = 2'b11;

// Note that these are total durations.  We'll load the counter
// with 2 less than these numbers.   One less because the state
// during which we load the value is one cycle and we aren't able
// to sample the TimeUp signal until the cycle after the count
// becomes 0, so there's a second cycle.

parameter
  YELLOWDURATION =   5,
  NSGREENDURATION = 45,
  EWGREENDURATION = 15;
```

```
reg [9:0] State, NextState;

always @(posedge Clock)
begin
if (Reset)
    State <= SW4RHR1;       // safe "initial" state upon reset
else
    State <= NextState;
end


// output logic
always @(State)
begin
{Load, Count, Value} = 0;
case (State)
    SW4GHRT:
        begin
        L1 = GREEN; L2 = RED; L3 = RED;
        Load = 1;
        Value = NSGREENDURATION - 2;
        Count = 0;
        end

    SW4GHRW:
        begin
        L1 = GREEN; L2 = RED; L3 = RED;
        Count = 1;
        end

    SW4YHRT:
        begin
        L1 = YELLOW; L2 = RED; L3 = RED;
        Load = 1;
        Value = YELLOWDURATION - 2;
        Count = 0;
        end

    SW4YHRW:
        begin
        L1 = YELLOW; L2 = RED; L3 = RED;
        Count = 1;
        end

    SW4RHR0:
        begin
        L1 = RED; L2 = RED; L3 = RED;
        end

    SW4RHGT:
        begin
        L1 = RED; L2 = GREEN; L3 = GREEN;
        Load = 1;
        Value = EWGREENDURATION -2;
        Count = 0;
        end

    SW4RHGW:
        begin
        L1 = RED; L2 = GREEN; L3 = GREEN;
        Count = 1;
        end

    SW4RHYT:
        begin
        L1 = RED; L2 = YELLOW; L3 = YELLOW;
        Load = 1;
        Value = YELLOWDURATION - 2;
        Count = 0;
        end

    SW4RHYW:
        begin
        L1 = RED; L2 = YELLOW; L3 = YELLOW;
        Count = 1;
        end

SW4RHR1:
```

```
        begin
        L1 = RED; L2 = RED; L3 = RED;
        end

default:
        begin
        L1 = RED; L2 = RED; L3 = RED;
        end
endcase
end


// next state logic
always @(State, S1, S2, S3, TimeUp)
begin
case (State)
    SW4GHRT:
        NextState = SW4GHRW;

    SW4GHRW:
        if (TimeUp && (~S1 || S2 || S3))
            NextState = SW4YHRT;
        else
            NextState = SW4GHRW;

    SW4YHRT:
        NextState = SW4YHRW;

    SW4YHRW:
        if (TimeUp)
            NextState = SW4RHR0;
        else
            NextState = SW4YHRW;

    SW4RHR0:
        NextState = SW4RHGT;

    SW4RHGT:
        NextState = SW4RHGW;

    SW4RHGW:
        if (TimeUp)
            NextState = SW4RHYT;
        else
            NextState = SW4RHGW;

    SW4RHYT:
        NextState = SW4RHYW;

    SW4RHYW:
        if (TimeUp)
            NextState = SW4RHR1;
        else
            NextState = SW4RHYW;

    SW4RHR1:
        NextState = SW4GHRT;

    default:
        NextState = SW4RHR1;
endcase
end
endmodule




module fsm(Clock, Reset, S1, S2, S3, L1, L2, L3);
input Clock;
input Reset;
input               // sensors for approaching vehicles
    S1,             // Northbound on SW 4th Avenue
    S2,             // Eastbound on SW Harrison Street
    S3;             // Westbound on SW Harrison Street

output reg [1:0]    // outputs for controlling traffic lights (01, 10, 11 for green, yellow, red)
    L1,             // light for NB SW 4th Avenue
    L2,             // light for EB SW Harrison Street
    L3;             // light for WB SW Harrison Street
```

```
wire Load, Count;
wire [7:0] Value;
wire TimeUp;

counter c (Clock, 1'b0, Load, Value, Count, TimeUp);
realfsm f (Clock, Reset, S1, S2, S3, L1, L2, L3, Load, Count, Value, TimeUp);

endmodule



// For demonstration purposes only.

module Top;

reg Clock;
reg Reset;
reg S1, S2, S3;
wire [1:0] L1, L2, L3;

fsm  f0(Clock, Reset, S1, S2, S3, L1, L2, L3);

initial
begin
Clock = 1;
forever #(5) Clock = ~Clock;
end

initial
begin
Reset = 1;
repeat (2) @(negedge Clock);
Reset = 0;
S1 = 0; S2 = 0; S3 = 0;
repeat (200) @(negedge Clock);
S1 = 1; S2 = 0; S3 = 0;
repeat (100) @(negedge Clock);
S1 = 1; S2 = 1; S3 = 0;
repeat (100) @(negedge Clock);
S1 = 0; S2 = 0; S3 = 0;
repeat (100) @(negedge Clock);
$finish();
end

endmodule
```