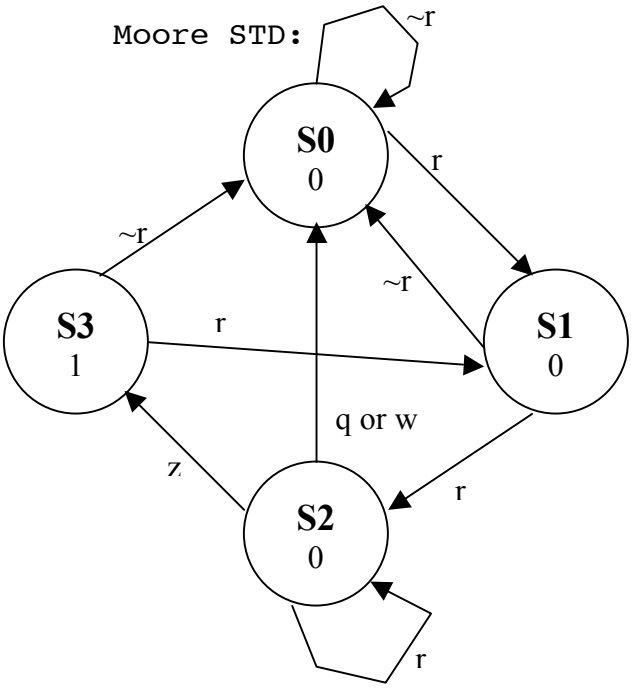
University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Science

EECS 150 D. E. Culler
Fall 2007

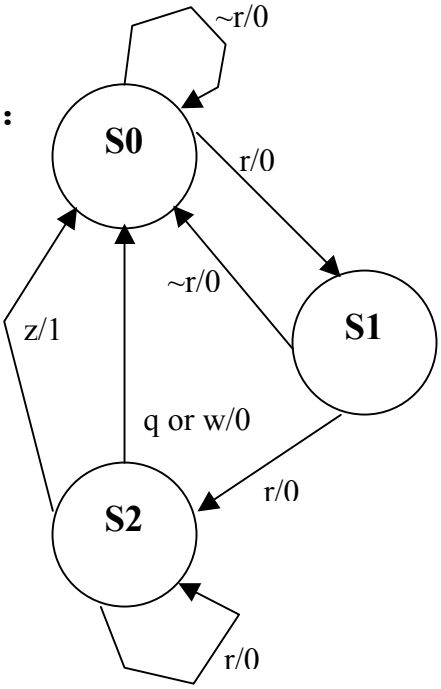
Homework #5: Memory and Protocols
Assigned 10/4/2007, Due 10/12/2007 at 2 PM

Problem 1: Draw a state transition diagram (STD) for a Moore machine that has
input i ∈ { `q, r, w, z` },
output o ∈ { `0, 1` }
and recognizes instances of the pattern `rrz`.
For example, input
`qrwzrrzrzrrrzw`
produces the output
`00000010000010.`

`Moore STD:`



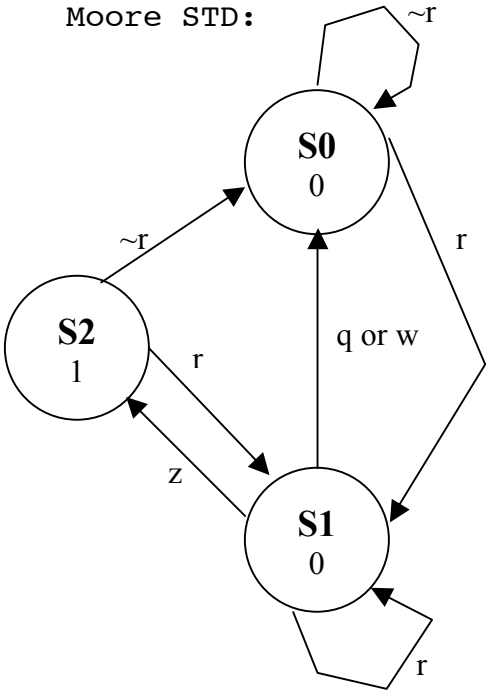
`Mealy STD:`



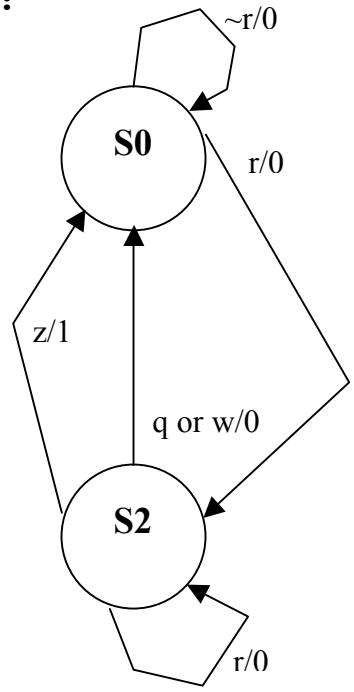
Draw a STD for an equivalent Mealy machine.

Problem 2: Modify your state diagram to recognize patterns of the form `r+z`, where `r+` means a sequence of one or more `r's.` What is the output for the input in problem 1?

`Moore STD:`

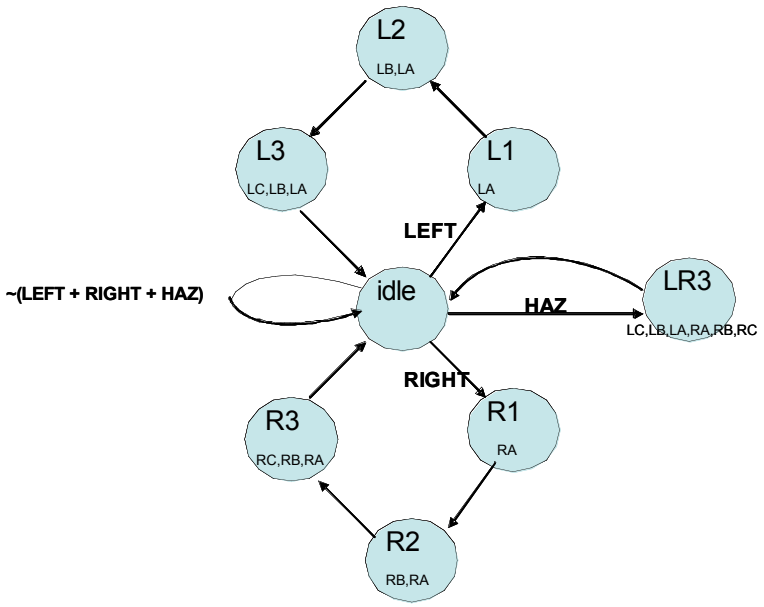


`Mealy STD:`



Input`:` `qrwzrrzrzrrrzw`
Output: `00000010100010`

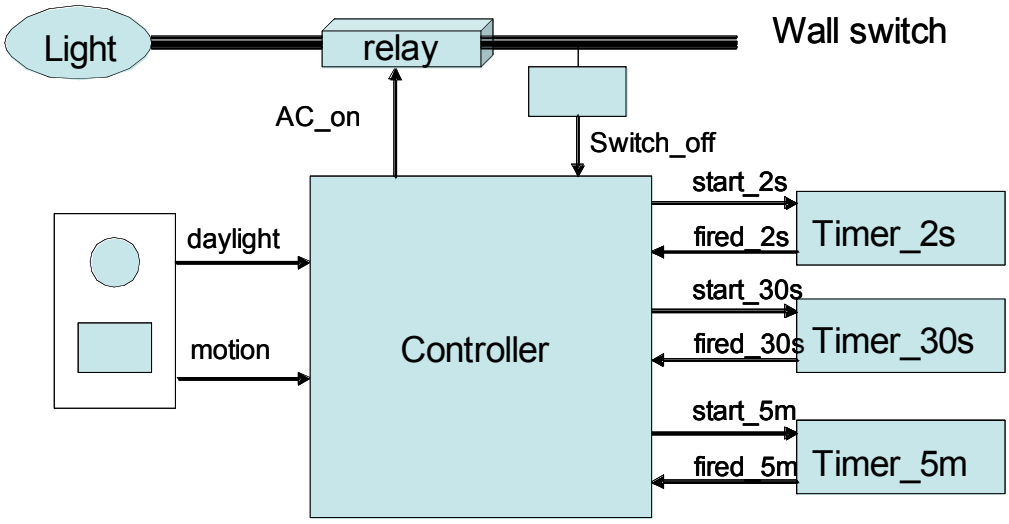
Problem 3: The old T-bird had a series of three taillights on either side (LC|LB|LA and RA|RB|RC) that lit up from the center our when you turned on the blinker. Here's a state transition diagram for the controller. Give the corresponding symbolic state transition table.



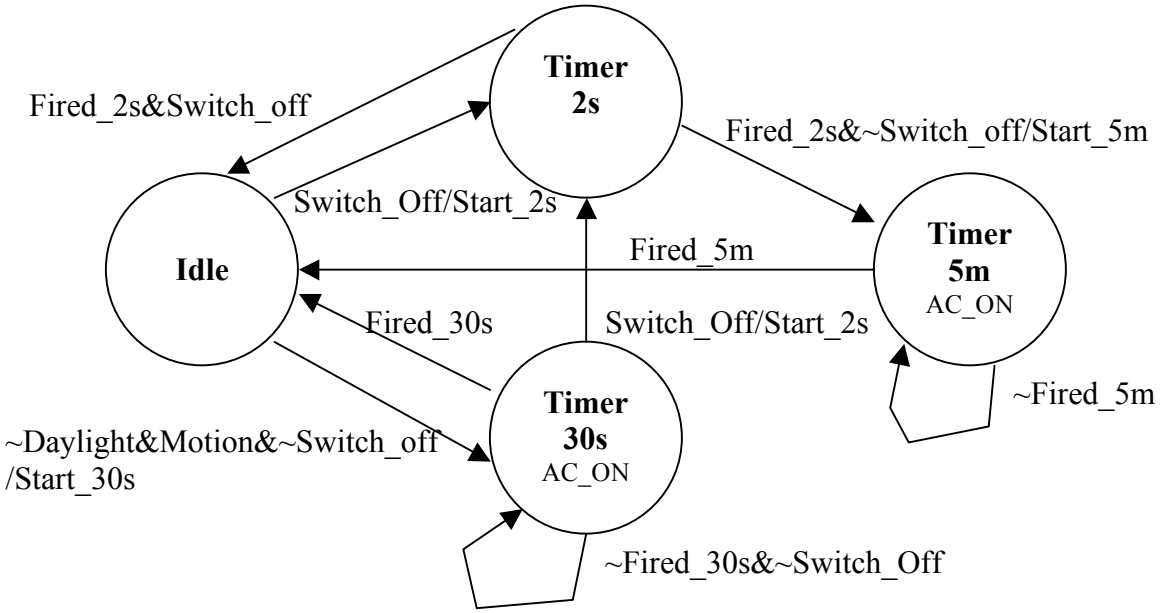
Current State	Current Outputs	Inputs	Next State
Idle	-	LEFT	L1
		RIGHT	R1
		HAZ	LR3
L1	LA	-	L2
L2	LB,LA	-	L3
L3	LC,LB,LA	-	Idle
R1	RA	-	R2
R2	RB,RA	-	R3
R3	RC,RB,RA	-	Idle
LR3	LC,LB,LA,RA,RB,RC	-	Idle

Problem 4. In this problem you will design a (now familiar) motion-based light controller shown in the diagram below. The wall switch is normally left in the ON position, supplying power to the controller and the light. The normal behavior of this controller is to turn the light ON for 30 seconds after motion is detected, but only if it is dark out. In addition, there is an override mode where if the switch is turned OFF and back ON within 2 seconds it turns the light on for 5 minutes, regardless of daylight or motion. Override can occur while the light is ON, either due to motion or test.

Your controller has two digital inputs based on the photocell and passive infrared sensors: **daylight** and **motion.** In addition, it has a circuit that detects when the switch is turned off and provides the input **Switch_off**. (The controller has a backup capacitor so it can run for a while with the switch off.) It has an output **AC_on** that controls a relay to actually turn the light on and off. It has three preset timers, each with a **start** control point and a **fired** signal. The start signal for a timer should be asserted for one cycle to start the timer. Each time it is pulsed it will reset the timer.



a. Give the symbolic state transition diagram for a Moore FSM controller. (This should be a clear high level specification of the behavior.)



b. Give a symbolic state transition table. (This should be 1-1 with your STD. It should not contain the specific concrete encodings of the states. It must cover all the possible inputs in each state using appropriate don't-cares to make it concise.)

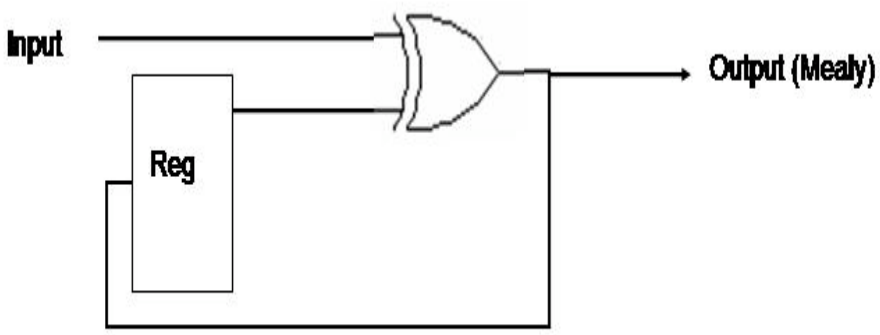
Current State	Inputs	Next State	Outputs
Idle	~Daylight&Motion&~Switch_Off	Timer_30s	Start_30s
	Switch_Off	Timer_2s	Start_2s
Timer_30s	~Fired_30s&~Switch_Off	Timer_30s	AC_on
	Switch_Off	Timer_2s	Start_2s
	Fired_30s&~Switch_Off	Idle	-
Timer_2s	Fired_2s&Switch_Off	Idle	-
	Fired_2s&~Switch_Off	Timer_5m	Start_5m
Timer_5m	~Fired_5m	Timer_5m	AC_on
	Fired_5m	Idle	-

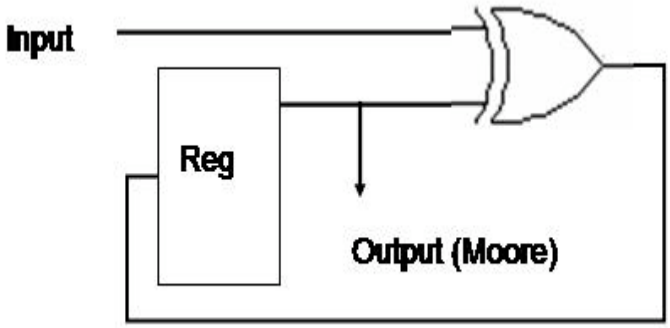
c. Implement your controller as a behavioral verilog module. The structure of the code should closely reflect your STD and STT. Use the same symbolic state names with parameter statement to give the concrete encodings. It should have two ALWAYS blocks. The combinational block should have a case statement with an arm for each state.

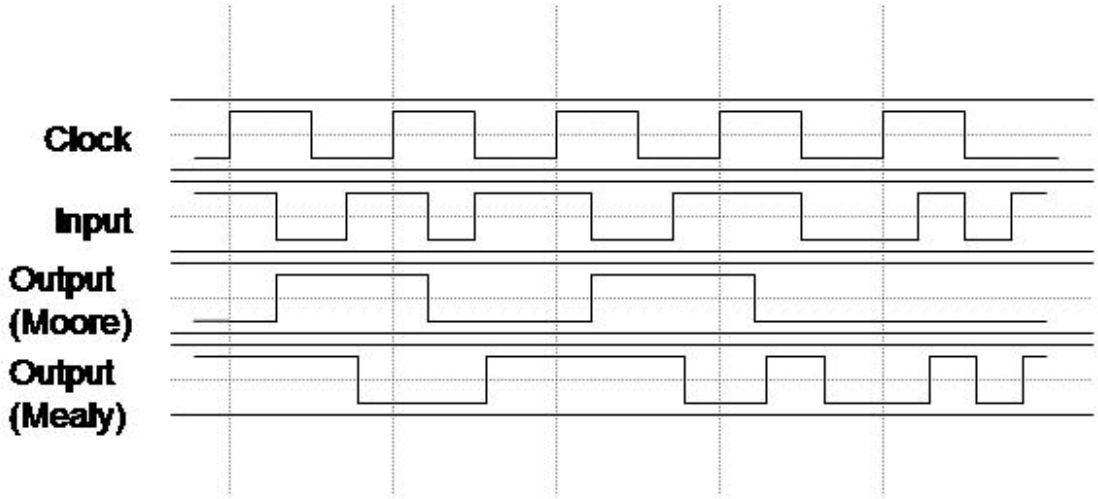
```
module LightCtrl( clk, switch_off, daylight, motion, AC_on, start30, fire30, start5, fire5,
start2, fire2)
 input clk, switch_off, daylight, motion, fire30, fire5, fire2;
 output AC_on, start30, start5, start2;

 reg state, nextstate;
 wire detect;
 assign detect = motion & ~daylight;

 always @ (cs, motion, daylight, switch_off, fire30, fire5, fire2)
  case (state)
    idle:
       AC_on = 0; start30=0; start5=0; start2=0;
       if (switch_off) begin nextstate = timer_2s; start2=1;
       else if (detect) nextstate = timer_30s: start30=1;
    timer30s:
       AC_on=1; start30=0; start5=0; start2=0;
       If(switch_off) begin begin nextstate = timer_2s; start2=1;
       else if(fire30) nextstate=idle; AC_on=0;
    timer2s:
        AC_on=0; start30=0; start5=0; start2=0;
        if (fire2&switch_off) nextstate = idle;
        else if (fire2&~Switch_off) nextstate = timer5m; start5m=1;
    timer5m:
       AC_on=1; start30=0; start5=0; start2=0;
       If(fire5) nextstate=idle; AC_on=0;
 end

 always @(posedge clk)
    begin
        start <= nextstate;
    end
```

Problem 5

a) Given the following Mealy and Moore circuit representations, complete the timing diagram given the input. The clock has a period of 8 ns and the register has a clk-to-Q delay of 2 ns and the XOR has a delay of 1 ns.



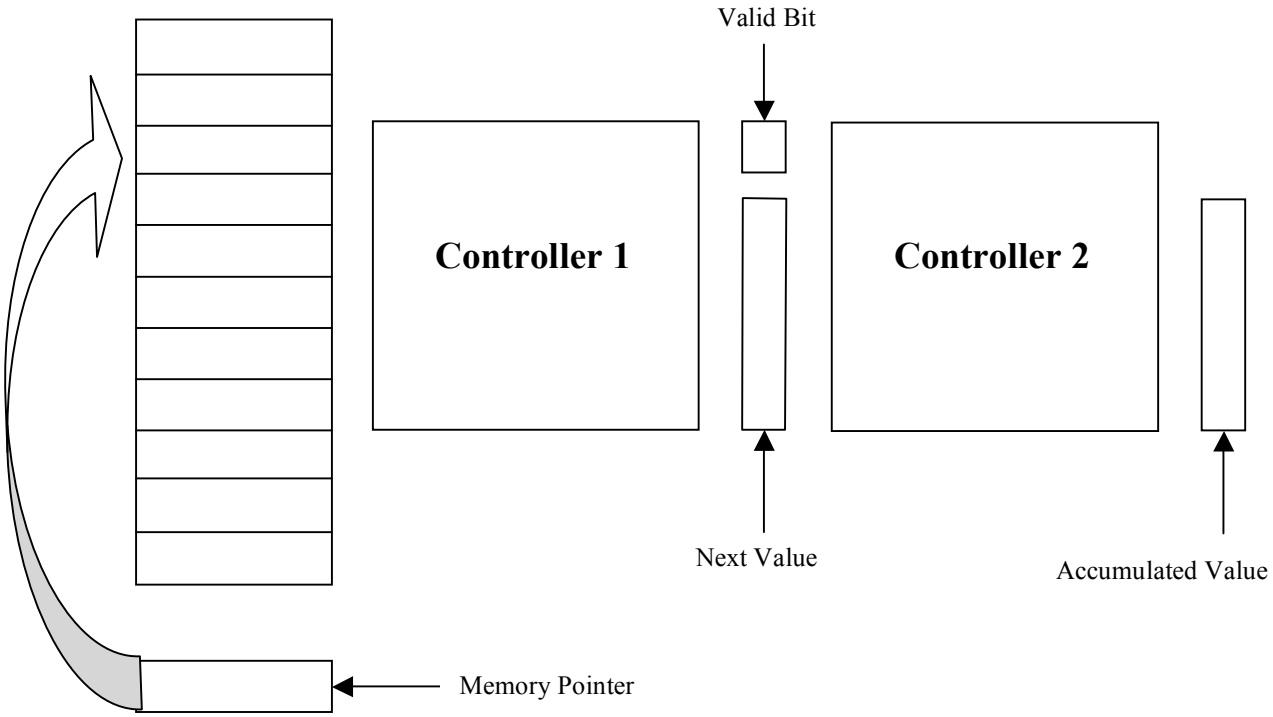




b) Explain the implications of using a Moore versus Mealy representation of the outputs for this circuit.
In a mealy machine the output values will change whenever the input values change. In the Moore machine the output values will only change on the clock edge.

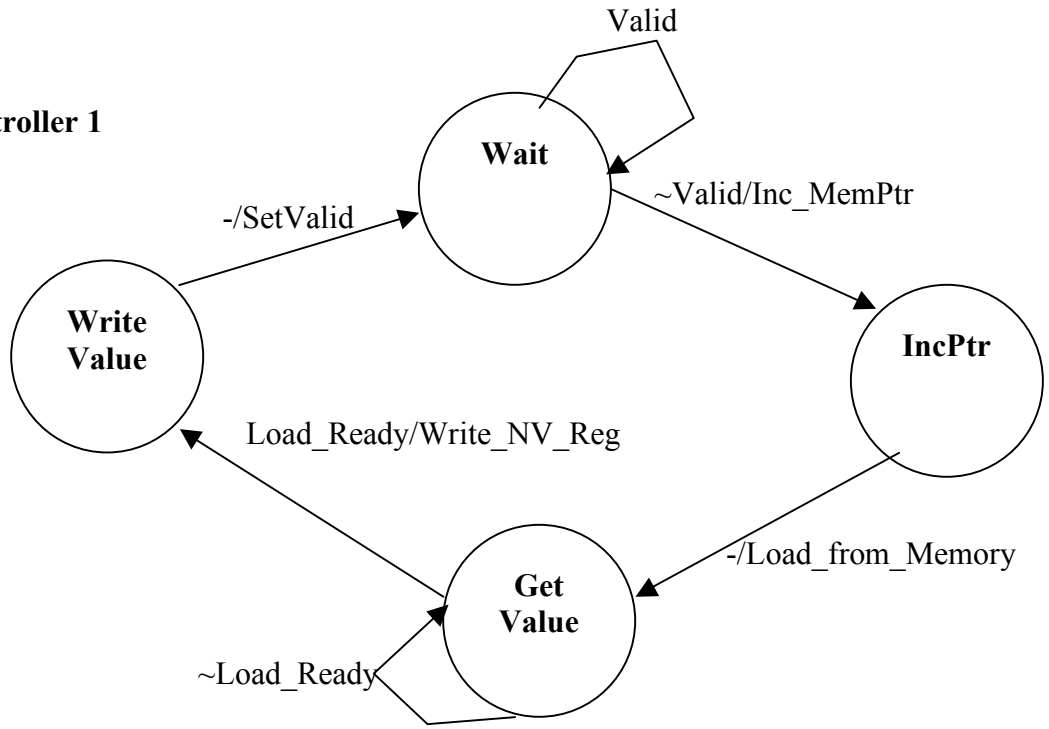
Problem 6. Draw the State Transition Diagrams for the controllers below. Controller 1 is the data producer; it streams a sequence of values to the shared, single word buffer. Concretely, it reads the value from the memory location pointed to by the memory pointer, places it in the next value register, and sets the valid bit to inform the consumer that the shared buffer is full. It increments the memory pointer before reading the next word. Controller 2 is the data consumer; it takes the value from the next value register when valid, clears the valid bit, accumulates the value and places it in the accumulated value register. The true controllers operate independently and run and potentially different rates. The producer cannot over-write the buffer until it has been emptied, and the consumer cannot read it until it is full. Be careful to design the FSM's such that memory locations are not added in multiple times in a row and that values are not lost.

Note that this problem is a little bit open ended. You will need to specify the inputs and outputs of your two cooperating state machines, as well as determining the symbolic states and the functional behavior.



Note there are many possible solutions. An example solution is given on the next page.

Controller 1



Controller 2