

EECS 151/251A Homework 4

Due Monday, Feb 22th, 2021

For this HW Assignment

You will be asked to write several Verilog modules as part of this HW assignment. You are encouraged to test them to verify functionality by running them through a testbench. As in Homework 2, a highly suggested simulator is <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/sp21/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.

As with previous homework assignments, **please include a short (1-2 sentence) explanation of your work and the design/analysis process unless otherwise directed in the problem.**

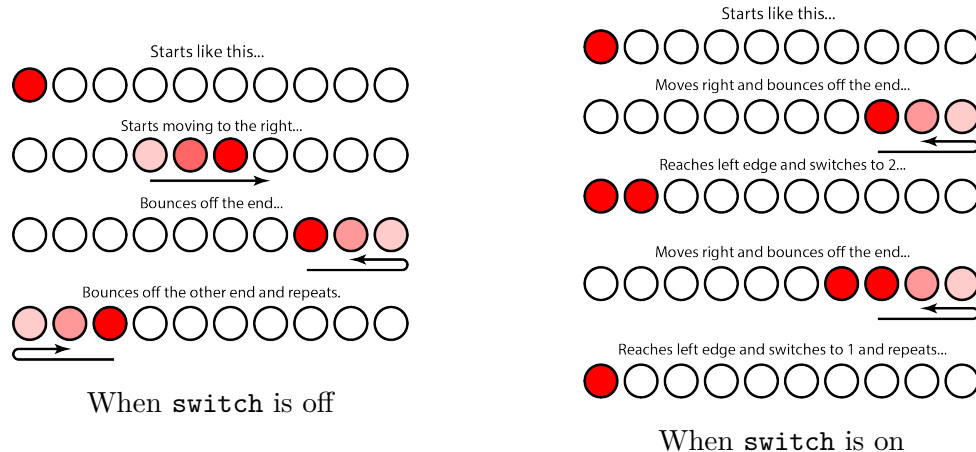
Problem 1. Light Display

As a side project, you have been consulted for the design of a simple light display for use as a decorative element in your friend's livestreaming setup. Your friend asks for the following features for their light fixture:

- The display is a bar of LEDs, which have a common cathode (negative connection), but independent anodes (positive connections).
- The bar of LEDs starts up with one LED on at the left edge.
- When running, the lights will switch on and off such that it gives the appearance of the light traveling back and forth along the bar. That is it will look like one light is traveling to the right, bouncing off the right end of the bar, then travelling left until it bounces off the left end, and so on.
- There will be an input `reset` to reset the bar to the initial state of having one light on at the left end.
- There will be another input `switch` that, when on, causes the number of lights to change between just 1 light on and 2 lights on every time the light reaches the left end of the bar.
- If `switch` is off, the display will keep whichever number of lights it currently has on and maintain the bouncing behavior.

- If **reset** is on, revert to the initial state of one light at the left. This takes precedence over all other transitions.

Your friend also provides the following illustration that demonstrates the points above.



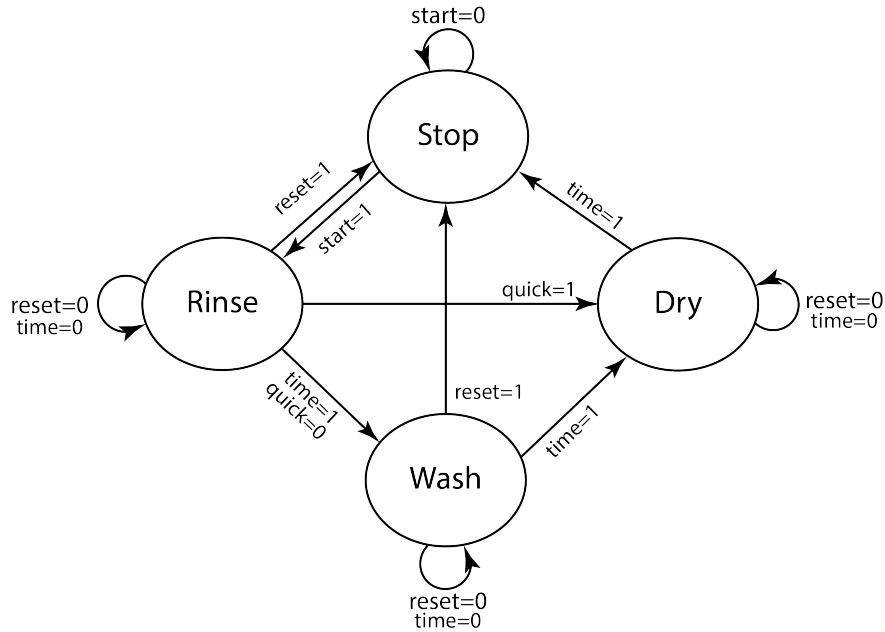
The lights will be controlled by a configurable shift register, which can change both the shifting direction as well as load in a set value, which will be used when switching between 1 and 2 lights. The shift register has a 2-bit configuration input and will shift right if the input is 01, left if it is 10, stop if the input is 00, and load a new value if the input is 11. The shift register also generates a `left_edge` and `right_edge` flag when the leftmost and rightmost bit are 1, respectively. (*You do not need to generate the loaded values, just the configuration bits of the shift register.*)

For this problem:

- Draw and label the State Transition Diagram for a Moore Machine that performs the above behavior by setting the configuration bits for the shift register.
- Write a behavioral Verilog module of the state machine. **Remember to use the EECS151.v library!**

Problem 2. FSM Design

Consider a simple dishwasher that has an option for a quick rinse cycle or a full wash cycle. The dishwasher will first rinse the dishes with hot water, then decide whether to wash with detergent or skip straight to drying depending on the setting. If you open the door of the dishwasher prematurely, a door switch will trigger a **reset** and the dishwasher will stop and go back to the initial state. A timer raises a **time** flag when the dishwasher is ready to move to the next state. The dishwasher has three outputs: **water**, **detergent**, **heat**. During the rinse cycle, it will request water. During the wash cycle, it will request water and detergent. During the dry cycle it will request only heat. Here is a conceptual state transition diagram for the dishwasher:



For each of the following scenarios, please provide:

- A state transition diagram with the state names and encoding (e.g. STOP (00)), as well as outputs labeled appropriately.
- A circuit diagram of the state machine. The state machine will receive the inputs `reset`, `start`, `quick`, and `time`. The state machine must have the outputs `water`, `detergent`, and `heat`. You may use logic gates of 4 inputs or fewer as well as multiplexers to implement your next state logic.
- A quick (1-2 sentence) summary of your design process and decisions.

Design the FSM for each of these scenarios:

- Binary-encoded Moore Machine
- Binary-encoded Mealy Machine
- One-hot Moore Machine
- One-hot Mealy Machine

Problem 3: Bit-Serial Network Interface

Design an FSM that will act as a receiver for a bit-serial network interface. The interface receives each packet serially, and the packets follow a special format:

```
<src address (4 Bytes)><dst address (4B)><payload (512B)><checksum(1B)>
```

Where `src` is the address of the sending node, `dst` is the address of the intended recipient, `payload` is the data content being send, and `checksum` is the check value to detect bit corruption in the received packet. This packet arrives `src` field first, with bits following afterwards (i.e. the packet is "read" left to right).

The sender node will compute the checksum of each packet by taking the sum of all the previous bits of the packet (i.e. the `src`, `dst`, and `payload` fields) modulo 256, so the result is 1 byte long. It then takes the *1's complement* of this sum and appends it to the packet as the checksum value. The receiving node then checks the contents of the package by summing the `src`, `dst`, and `payload` fields with the checksum field. Therefore, if the bits are all correct, the final value of this sum should total to 0. If this is not 0, then the check fails and the receiver knows that the packet it has received has somehow been corrupted.

In this design, the address of the receiving node is stored in a 4B wide register, which we will name `my_addr`. Your FSM should check for both the destination address as well as the packet fidelity, each with their respective outputs. The checks and input/output specification of the receiver is as follows:

- Check that we are the intended recipient (i.e. the `dst` matches `my_addr`), and if the match succeeds, set output `packet_match` true.
- Check that the packet is uncorrupted, and if the test succeeds, set output `check_match` true.
- **The two match signals each have a `packet_match_valid` and `check_match_valid` signal associated with them. This valid signal should be set to True synchronously with the completion of their respective checks.**
- The receiver should pause the input from the serial interface until it is done processing the current packet. To achieve this, the interface will not feed new bits to the receiver until the receiver sets a `input_ready` signal high. Similarly, the interface presents a signal to the `input_valid` input of the receiver. **The receiver should only accept new bits when these signals are both high.**

You are given access to several auxilliary blocks to use as part of this state machine. There is a parametrizable clock-enabled shift register and counter that you may use to assist the state machine by generating flags to be used as additional inputs to the FSM. You may use Verilog comparison operators as well as the addition operator to implement the checks.

You are free to choose between a binary-encoded SM or one-hot SM, and between Moore or Mealy machines. Please discuss this choice in the summary. For this problem, turn in:

-
- A state transition diagram with outputs labeled.
 - A block diagram of your FSM. You may abstract the counters, adders, comparators, and shift register as black box modules, but the state registers and next state logic for the state machine must be elaborated in full.
 - A behavioral Verilog implementation of the state machine.
 - A short (3-4 sentence) description of your design procedure and decisions.