ECE 571
Introduction to SystemVerilog
Spring 2021
Homework 2

My electric garage door opener has a single button and works as follows. If the door is closed and the button is pressed it raises (opens) the door. If the door is open and the button is pressed it lowers (closes) the door. If the button is pressed while the door is in motion, the door stops. If the button is pressed when the door is not in motion but is neither completely open nor completely closed, then it causes the door to move in the opposite direction it most recently moved. So, for example, if I pressed the button while the door was closing (but not yet fully closed) the door will stop. If I press the button again, the door will raise.

Draw a state transition diagram for the garage door controller. It has three inputs: Button, DoorOpen, and DoorClosed. Button is true if the button has been pressed. DoorOpen is connected to a sensor and is true if the door is fully open (raised). DoorClosed is connect to sensor and is true if the door is fully closed (lowered). The controller has two outputs for controlling the garage door motor: RaiseDoor (which turns the motor on to raise the door) and LowerDoor (which turns the motor on to lower the door).

Be sure to use descriptive names for the states and to label all state transitions. Assume a Moore machine implementation. Create a table that indicates the values of the outputs in each state.

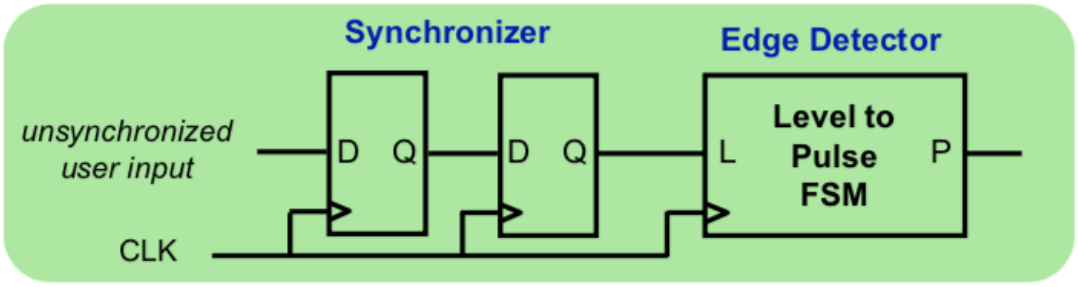
In reality, my garage door opener has a safety feature. There's a light sensor that detects the presence of an object in the path of the door and stops the door if the door is descending. If the door is stopped in this manner, it remains stopped until the button is pressed (at which time the door changes direction, and is raised). Modify your state transition diagram and output table to reflect this behavior. The additional input signal Obstacle is true if there's an obstacle in the path of the door.

It turns out there's yet another feature in my garage door opener. It takes between 10-14 seconds for the door to go from fully open to fully closed or from fully closed to fully open. In order to prevent the motor from burning out if there's a failure of some kind (e.g. the sensors that detect when the door is fully open or fully closed), there's a timeout feature. If, while raising or lowering the door, 16 seconds has elapsed, the motor is turned off until the button is pressed, when it reverses direction. To implement this feature there's a counter. It has an Increment input and an output Timeout which is asserted when the counter has counted 16 seconds. The Timeout output remains asserted until the counter is (asynchronously) reset with the ResetTimer input.

Modify the state transition diagram and output table to implement these features.

We didn't talk about the details of the button. We want a single cycle pulse when the button is pressed. Otherwise the Button signal will remain high for multiple clock signals, causing the door to repeatedly reverse direction. The button must be released and pressed again to generate another pulse.

Part of the pulse generator circuit discussed in class includes an edge detector. However, we (subtly) assumed the Start signal was synchronized. In this application, a human presses the button and thus the initial Button input is not synchronized and must first be synchronized with the clock (to avoid metastability issues).
Here's a circuit that does that.



Assume the system clock is 10Hz. Because a human cannot press a button twice in less than 100 ms, this will ensure we don't miss any button presses.

Note, however, that we need to drive the seconds counter with a 1Hz clock. So you'll need to divide the system clock by 10 before using it as the clock for the counter. There are several techniques for this. A common one is to use a switch-tail ring counter (often used in a Johnson counter) which you'll use for this assignment.

For this assignment, create five design modules:

A button synchronizer with edge detector
A 5-bit switch-tail ring counter
A timeout counter
The FSM
A top-level module that instantiates all of the above to produce the system

Use (synthesizable RTL-level) procedural behavioral modeling for all the design modules.

Create a testbench for the synchronizer, the switch-tail ring counter, and the timeout counter. Create a testbench for the top-level module that demonstrates that your system works. You do not need to submit a testbench for the FSM itself.
Your top level module *must* be declared as follows:

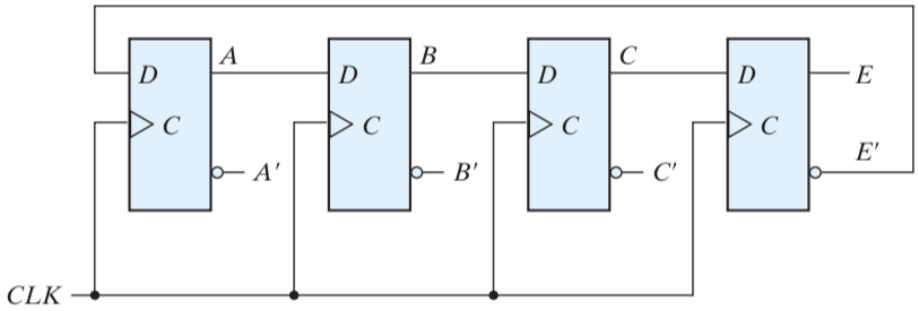
```
module GarageDoor(
          input Clock, Reset, ButtonAsynch, Obstacle, DoorUp, DoorDown,
          output RaiseDoor, LowerDoor);
```

All design files and testbenches must be in a single directory that is your username. This directory should be zipped and submitted to D2L as HW2.zip. Individual files should be named as follows:

design.pdf	Your design documentation (state transition diagram, table)
edge.v	A button synchronizer with edge detector
edgetb.v	Testbench for the button synchronizer
switchtail.v	A 5-bit switch-tail ring counter
switchtailtb.v	Testbench for the switch-tail ring counter
timeout.v	A timeout counter
timeouttb.v	Testbench for the timeout counter
fsm.v	A Moore style finite state machine to control the garage door
garagedoor.v	The garage door system (FSM and other modules above)
garagedoortb.v	Testbench for the garage door system

Switch-tail ring counter

A 4-stage switch tail ring counter is shown below. Note that it's just like a shift register except the input to the MSB is the complement of the LSB. The (synchronous) reset is not shown.



The outputs are:

Sequence number	Flip-flop outputs			
	A	*B*	*C*	*E*
1	0	0	0	0
2	1	0	0	0
3	1	1	0	0
4	1	1	1	0
5	1	1	1	1
6	0	1	1	1
7	0	0	1	1
8	0	0	0	1

Notice that the effect is to produce four signals which are essentially (phase shifted) 50% duty cycle clocks with 1/8th the frequency of the original clock. So, we can create a simple circuit that will divide an input clock by 10 (sometimes called a decade counter) by using a 5-bit switch-tail ring counter.