Programming Assignment 3: Traffic Light Controller

CS141 Spring 2019

Part 1 due March 1, and Part 2 due March 8

Introduction

The programming assignments thus far have only required combinational logic. The wires and gates you have been using so far are stateless elements that implement logical circuits. In this lab, we will focus on sequential logic, where elements can have state, like the flip-flops discussed in lecture. Stateful elements are crucial to most computations, corresponding roughly to variables in programming languages.

You may now use behavioral Verilog, including always and initial blocks, regs, control logic (if/else, case statements), and any arithmetic operators you would like, so there are no restrictions on the Verilog you can use. You should take a look at the Sequential Circuits with Verilog PDF before starting this assignment. There are lots of new constructs you will make use of and you should be sure you have a solid understanding before jumping into this assignment. **Be sure to attend section and post any questions on Piazza. Start early on this assignment - it is not easy and will likely take more time than PA2!**

Assignment

For this assignment, you'll be designing a controller for the traffic light of a 4-way intersection and running it on your FPGA. The intersection has a North-South road, an East-West road, and pedestrian cross walks.

Modules

In main.v you will see 5 modules instantiated: `debouncer`, `clock_generator`, `clk_divider`, `timer`, and `traffic_light_controller`, and logic interfacing the modules, the switches, and the lights.

The LEDs and switches will be used to implement the lights and the presence of cars/pedestrians. LED[7:5] will represent the red, yellow, and green lights for the North-South road, respectively. LED[2:0] will represent the red, yellow, and green lights for the East-West road, respectively. LED[4:3] will represent the North-South and East-West pedestrian walk signals, respectively. Additionally, any of switch[7:5] will represent a car waiting on the NS road, any of switch[2:0] will represent a car waiting on the EW road, and either of switch[4:3] will represent a pedestrian waiting to cross the street. We will use 2 different reset buttons - the red reset button to reset the clock divider and the down button for the traffic light controller and the timer - because of the different frequencies that the modules operate at.

To create this traffic light controller, you will implement 3 modules - a clock divider, a timer, and the controller itself.

Debouncer

When you press or release a button on your FPGA, it may actually trigger several brief back-and-forth transitions before settling at the changed value. This is referred to as "bounce" in the button. The `debounce` module ensures that any changes that last less than 100ms are ingored, so that each button press and release only triggers one change in the value. Feel free to take a look at this module, but you do not need to understand how it works. We insantiate 3 different debouncers and pass in the raw values of the buttons (`rstb_button`, `button_center`, `button_down`) and output a debounced version (`rstb`, `button_center_db`, and `button_down_db`). Note: the reset button on your FPGA is active-low, which means it has a value of 0 when pressed and 1 when released. We invert `rstb` to produce the active-high `rst` signal, which is what should be passed into your modules.

Clock Generator

Your FPGA has a 100MHz unbufferd clock source, which is not usable for use in many designs. The `clock_generator` module takes in the 100MHz unbuffered clock, and produces a usable 100MHz signal. This module was autogenerated by Xilinx ISE tools, and you shouldn't worry about how it works, nor should you be concerned with the difference between buffered and unbuffered clocks.

TODO: Clock Divider

We want our Traffic Light Controller and Timer to operate at a frequency of 1Hz, so we can work in units of seconds. Complete the implementation of `clk_divider.v` to take in a 100MHz clock signal and output a 1Hz clock signal. This means that for every 100 million positive edges of the input clock, there should be one positive edge of the output clock. There should also be a reset signal that makes it such divider resets to a state in which it has seen 0 positive edges from the input thus far.

TODO: Timer

We'd like states of our traffic light controller to last for certain durations, so we need a timer that is flexible. In `timer.v`, complete the implementation of a timer that meets the following requirements. The timer produces a 4-bit (unsigned) output (`out`) that represents the current time remaining. The timer takes a reset (`rst`) input, which when asserted, should reset the timer to the maximum value. The timer also takes an initial value (`init`) and a (`load`) signal. When load is high, the current time remaining should get set to the `init` value. There is also an enable (`en`) input, and when `en` is high, the timer should decrement by 1, but should stop at 0. If `en` is not asserted, the timer should stay at its current value (unless a load or reset occurs). `rst` should take highest precedence, followed by `load`, and then `en`. The output should only change on positive edges of the input clock, which should be the 1Hz signal from the clock divider.

TODO: Traffic Light Controller

The traffic light controller takes a 1Hz `clk` and a `rst`. It also has the necessary inputs and outputs to interface with the timer module - `timer`, `timer_rst`, `timer_en`, `timer_init`, and `timer_load`.

Finally, the controller takes `car_ns`, `car_ew`, and `ped` to indicate the presence of a car on either road or a pedestrian, and the controller should output `led_ns`, `led_ew`, and `led_ped` to turn on the appropriate LEDs in `main.v`.

Complete the implementation of `traffic_light_controller.v`, such that your traffic light controller obeys follow the following rules.

1. The traffic light should reset to an idle state in which both lights are red and both pedestrian walk signals are off.
2. The idle state should go immediately to the pedestrian state, in which both lights are red, and both pedestrian signals are on.
3. The pedestrian state should last for 15 seconds before transitioning to the next state.
4. If there is a car on the NS road and not on the EW road, the next state from the pedestrian state should have the NS light Green and the EW light red.
5. If there is a car on the EW road and not on the NS road, the next state should have the EW light green and the NS light red.
6. If there is a car on both roads or neither road, the pedestrian state should go to the road that was red most recently (i.e. if the NS was green more recently, EW should be green next). After reset, the pedestrian state should lead to green on the NS road.
7. Green lights should last for 10 seconds and should immediately transition to yellow lights on the same road.
8. Yellow lights should last for 5 seconds.
9. Following a yellow light, if there is a pedestrian, the controller should proceed to the pedestrian state. Otherwise, it should go to green on the other road.
10. When one traffic light is green or yellow, the other should be red.
11. When one traffic light is green or yellow, the same pedestrian light should be on, and the opposite pedestrian light should be off.

HINT: The staff solution uses multiple FSM states for each combination of lights listed above. Depending on your implementation, it could be a good idea to have an initialization state to set the timer to the proper count and en execution state in which the timer is enabled. Additionally, it is acceptable if your controller takes an additional cycle after the counter reaches 0 before either the light changes or the timer starts.

Deliverable

Part 1 (due March 1st, 5pm):

For part 1, you should implement the clock divider and the timer. In this part, you should light up the FPGA LEDs with the output of the timer and use the switches and center button to provide the initialization value and load signal to the timer. The timer should reset properly with a press of the down button. You should submit a testbench for each module and a short writeup of your design and testing strategies. For this part, you should also submit an initial FSM diagram for the traffic light controller module.

Part 2 (due March 8th, 5pm):

For part 2, you will implement the traffic light controller module to complete this assignment. You should also write a testbench, a short writeup of your deisng and testing strategies, and an updated FSM diagram to reflect any changes that you may have made. We encourage you to start on part 2 as soon as possible, since the breakdown of the parts is likely not 50-50. Getting the FSM to work on the FPGA is **HARD**!

You will demo each part on the due date and will be expected to demonstrate that your implementation works in simulation and on the FPGA. You should have a testbench that produces the appropriate stimuli to demonstrate a working simulation and should be able to walk the TFs through the waveforms to demonstrate your implementation.

Submit the following on Canvas for each part:

1. Your Xilinx project (including testbenches)
2. An FSM diagram
3. A description (one or two paragraphs) of your simulation/testing methodology

Unzipping the compressed file should yield the following general directory structure (the individual file/folder names don't have to match what is written below, but be descriptive when appropriate):

`lab3_xilinx_project/` `fsm_diagram` `description.doc` / `description.txt` / `description.pdf`

As you work on this assignment, remember that you should only have one module for each source file. This helps keep your code much more organized and modular. Also, please be sure to thoroughly comment your Verilog!

Here is the rubric we will be using for grading:

Part 1 (60 Points)

Component	Points
Clock Divider Implementation	15
Timer Implemetnation	15
Testbench Demonstrates Functionality in Simulation	10
Works on FPGA	10
FSM Diagram	5
Writeup	5

Part 2 (75 Points)

Component	Points
Traffic Light Controller Implementation	30
Testbench Demonstrates Functionality in Simulation	15
Works on FPGA	15
FSM Diagram	10
Write up	5

Submission

You should submit the assignment to the page on Canvas. Please see the submission guidelines for how to prepare your project for submission.