Billy Koech and Nicolas Weinger

CS 141: Computing Hardware

Lab 3 part1 : Traffic Light Controller

Design, Testing and Simulation methodology

**Timer**

The timer is implemented with an asynchronous reset. A current state variable set to 4’b1111 through a nonblocking assignment when the reset is asserted; otherwise it is set to the next state variable. The next state variable is calculated through a blocking assignment. This is implemented through a sequence of if else statements in the following priority order: load (if load is asserted), setting the next state to zero (if the current state is zero), decreasing the counter (if enable is asserted) and finally setting the next state to the current state (in case none of the above conditions are met).

**Timer Testbench**

The testbench tests for the following functionality:

1. **Load:** Load is asserted then the init variable is set to a 4 bit value. The testbench waits for one clock cycle to ensure that the init value is sampled then the output is checked to see if it is equal to init.
2. **Reset:** 4’b0100 is loaded then reset is asserted. The output is then checked to see if it is equal to 4’b1111
3. **Test down counter:** Reset is asserted to ensure that the timer starts at 15. Enable is then asserted and the testbench waits for 10 clock cycles to make the timer count down by 10. It then checks that the output is equals 5.
4. **Test if timer stops at zero:** Reset is asserted to ensure that the timer starts at 15. Enable is asserted and the testbench runs for 20 clock cycles. The output is then checked to see whether or not it is equal to zero.
5. **Test if timer holds value when enable is not asserted:** Reset is first asserted then the clock runs for 5 clock cycles to make the timer count down by 5. Enable is then disabled and the clock runs for 5 more clock cycles. The output is then checked to see if it’s still equal to 15 minus 5 and not 15 minus 10.
6. **Test if reset takes precedence over enable:** Reset is asserted while enable is asserted. The output is then checked to see if it is equal to 15.
7. **Test if reset takes precedence over load:** Reset is asserted while load is asserted. The output is then checked to see if it is equal to 15.
8. **Test if load takes precedence over enable:** Load is asserted while enable is asserted. The output is then checked to see if it is equal to the init variable.

**Clock Divider Testbench**

The clock divider testbench simply pulses the clock input of a parameterized clock divider. We then observed the waveforms to check that the divider was indeed behaving as expected.

**Traffic controller**

The traffic controller is programmed based on the Present-State-Next-State table. Please see the attached Excel file (NSPS Diagram.xlsx) for a full table showing outputs for each state and the transitions to next states for given inputs. The next state logic is implemented using blocking assignments which update a **next\_state** register. Transition to the next state then occur at every clock cycle by the use of nonblocking assignments.

**Testing strategy**

The traffic controller is tested by assigning values to the traffic controller inputs **car\_ns, car\_ew, ped** and viewing the waveforms to see confirm that it follows the expected sequence. For example, the inputs are assigned the following values:

**car\_ns = 0; // no car ns**

**car\_ew = 0; // no car ew**

**ped = 0; // no ped**

then the simulation is executed and the waveforms observed to see if they follow the following order of transition: Idle -> Pedestrian for 15clk cycles -> NS green for 10clk cycles -> NS Yellow for 5clk cycles -> Pedestrian for 15clk cycles > EW green for 10clk cycles -> EW Yellow for 5clk cycles -> Pedestrian for 15clk cycles -> …

The waveform is also checked to ensure that when one side is green the other is red and when one side is green the the pedestrian light is on and vice versa.

This procedure is repeated for different inputs of **car\_ns, car\_ew, ped.**