EE365 Midterm: Traffic Simulation

Timothy Kopp

November 10, 2010

Abstract

A traffic signal simulator was successfully designed, implemented, and tested for the DE1 board. The traffic simulator has two streets of varying priority, as well as a crosswalk having highest priority. The system displays the red and green lights for the streets and crosswalk, as well as timers indicating how much longer a street or crosswalk will have a green light. The system gives precedence to the pedestrians without allowing them to stop the flow of traffic. The system was implemented in VHDL and proven to function as designed on Altera's DE1 board. The functioning system was demonstrated in Clarkson University's Undergraduate ECE Laboratory on November 10, 2010.

1 Problem Specification and Interpretation

The goal was to design and implement a simulation of a traffic light control system for the DE1 board. The simulated intersection is an intersection of two roads with crosswalks. The simulation uses the DE1 board's green LEDs to represent "Go" when on. Likewise, the red LEDs are used to indicate "Stop."

Of the two streets, one street is a main street, or high priority street, and the other is a side street, or low priority street. From an outside perspective there are three states: where the high priority street has a green light and all else have red lights, where the low priority street has a green light and all else have red lights, and where the pedestrians have a green light and all else have red lights. At no point should more than one green light be on.

The high priority street has a green light until either a pedestrian presses the button at the crosswalk or the sensors detect a car at the intersection of the low-priority street. The sensors and button are simulated with *KEY1* and *KEY2* respectively. When one of these events happens, the system will count down five seconds, and then switch the lights appropriately for a time, four seconds for pedestrians and nine for the low priority street. After that time, the lights switch back to the high priority street. The timers for the lights are displayed using the DE1 board's seven-segment displays.

Pedestrians using the crosswalk have the highest priority, although the system cannot allow a pedestrian to prevent the flow of traffic on the roads. The specification was not specific as to how this issue was to be dealt with. The designed system allows the pedestrian to take priority over the next street in the queue twice per rotation. That is, the pedestrian can prevent other streets from taking their turns only two times before the high priority street gets a turn. This ensures that while the pedestrians will have the highest priority, they will not be able to abuse the priority and stop traffic flow in any direction.

The system also has a reset button for use by the technicians if an error occurs. This button is represented by the DE1 board's *KEY0*.

2 Problem Decomposition

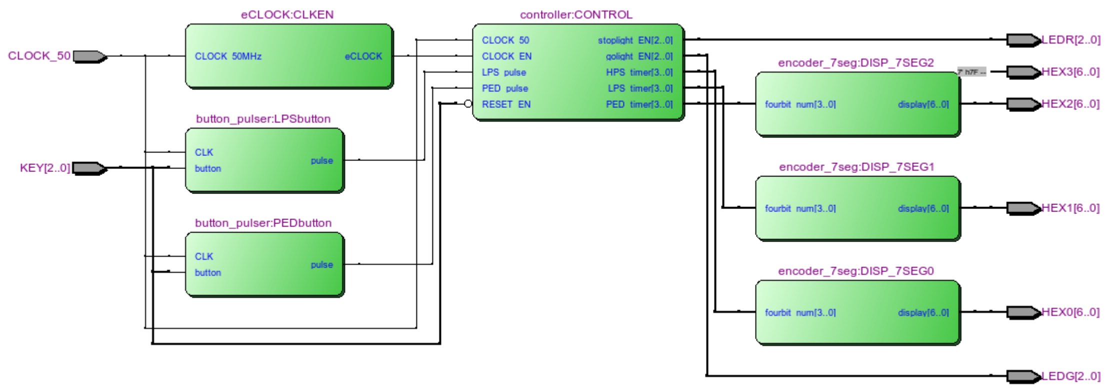

Figure 1: Block Diagram of *trafficsim*

The problem was decomposed into multiple components to increase modularity and component reuse. Generic components were used where possible to increase reusability and ease of modification. Additionally, modules created for previous projects were used.

Figure 1 is a block diagram of the top-level VHDL entity *trafficsim*. The $50MHz$ clock that comes on the DE1 board is fed into a *eCLOCK*, which produces a clock enable signal at a rate of $1Hz$. This signal is wired to the controller for use with the timers. The request KEYs are wired through *button_pulser*s which take the button press signals of arbitrary length, and produce an associated $20ns$ pulse. These pulse signals are wired to the controller where they are used to determine whom gets the green light.

The controller is the module that determines whom gets which lights and for how long. Using the input signals from the *eCLOCK* and *button_pulser*s, it determines the current state and outputs the appropriate signals to the LEDs and the numbers for use by the *encoder_7seg*s.

The countdown signals from the controller are wired to the *encoder_7seg*s, which decode the numbers into signals usable by the seven-segment displays to which they are wired. The output signals for the red and green lights from the controller are wired to the corresponding LEDs.

3 Detailed Design

The *eCLOCK* component created for EE365 Project 2 was reused without modification. The VHDL source code for this component can be found in Appendix A, Listing 1.

The second module takes a button press lasting many $50MHz$ clock cycles, and producing a single associated $20ns$ pulse. The *button_pulser* component created for the same purpose in EE365 Project 2 was reused without modification. The source code can be found in Appendix A, Listing 2.

The next component takes a four-bit number in binary form, and converts it to the appropriate signals for the seven-segment displays on the DE1 board. The *encoder_7seg* used in EE365 Project 2 was reused without modification for this purpose. The VHDL source code can be found in Appendix A, Listing 3.

The final module is the controller. The controller acts in five states. Figure 2 shows the state diagram for the controller. When a request comes from one of the KEYs, the controller registers that a request was made and acts on it when it is appropriate.

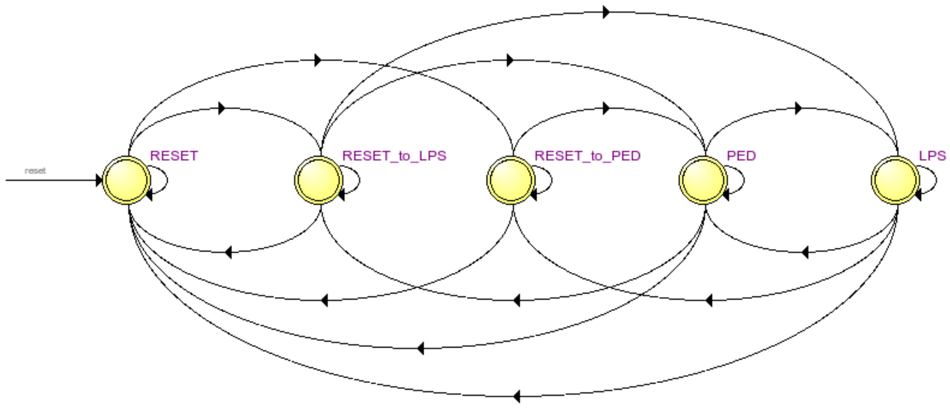

Figure 2: State machine for controller

The *RESET* state is the default state in which the high priority street(HPS) has a green light. If the controller has a request to change to the low priority street(LPS), then this state will transition to the *RESET_to_LPS* state. If the controller has a request to change to the pedestrian(PED) state, then the *RESET* state will transition to the *RESET_to_PED* state. If there exists requests to transition to both states, then the pedestrian state takes precedence.

The *RESET_to_LPS* state outputs a green light for the HPS, and red for all others. It counts down the five seconds for the HPS to continue receiving a green light, and the changes the state to the *LPS* state. This state contains logic such that if a pedestrian makes a change request while the timer is counting down, and has not exceeded the number of turns a pedestrian can have, it will transition to the *PED* state instead. Likewise, the *RESET_to_PED* state will count down and transition to the *PED* state. It does not require special logic, since nothing has precedence over the pedestrian.

When in the *LPS* state, the *controller* outputs a green light for the LPS and a red for all others. It counts down for the specified nine seconds, and then transitions back to the *RESET* state. If a pedestrian request is made during this state, then after the countdown is complete then a transition to the *PED* state will occur instead. This state contains logic to determine if the pedestrian has interrupted too many times, and if so, simply transitions to the *RESET* state.

When in the *PED* state, the *controller* outputs a green light for the PED and a red for the others. It counts down for the specified four seconds, and then transitions back to the *RESET* state. In the event that the pedestrian took the turn of the LPS, the state will instead change to the *LPS* state after countdown.

At any point, if *KEY0*, the clear button, is pressed, then the state and all outputs asynchronously change to the *CLEAR* state. The *controller* was implemented using VHDL generics such that one could easily vary the times that the different streets get during their green lights. The VHDL source code for the *controller* can be found in Appendix A, Listing 4.

The top-level component is the *trafficsim*. This component wires the modules together as depicted in Figure 1.

4 Alternative Designs

One alternate method that was explored was to not limit the number of times the pedestrian could take the turn of a street. Although this would give the pedestrian a higher priority, as specified, it allows the pedestrian to stop traffic in one direction by swapping turns with one of the streets. Although the design that was implemented will make the pedestrian wait longer in edge cases, it ensures that traffic is able to flow in all directions.

Another design would have been to break the *controller* module into smaller subcomponents. Although modular design such as this is generally preferred, it was not desirable in this application. Any logic elements, such as determining if the pedestrian has had too many turns, is highly specific to the traffic simulator. Separating it into a module would have yielded a component or components that were not easily reusable. Additionally, it would have increased the number of components, requiring the designer to either define a meta-component composed of these subcomponents, the meta-component serving the same purpose as the *controller*, or require the end-user to instantiate multiple entities for a single logical component. Additionally this modularization does not increase the ease of understanding or readability in any way. Another functionality to break into modules would have been the counters. Although this seems like an attractive idea at first, it would force the end user to specify generic parameters for three distinct components, even though the parameters are related. This would increase the coupling between the components, eliminating any increase in modularity that was apparently gained.

5 Specification Testing

The components that were reused from previous projects were not unit tested, as they have been unit tested and proved to function as designed previously. The only subcomponent created specifically for this project was the *controller*. Due to the simplicity of the top level component, the *trafficsim* required little testing assuming black-boxed modules. As a result, testing the system as a whole was the most reasonable test plan.

SignalTap Analyzer was used to ensure that the state changes were occurring precisely when they were designed to. This was determined by comparing the input pulses from the buttons to the outputs on the LEDs. Figure 3 shows the SignalTap output for the transition from the *RESET* state to the *LPS* state and back. The zeroeth LED corresponds to the HPS, the first LED corresponds to the LPS, and the second LED corresponds to the pedestrian. The three countdown signals are the numbers shown on the seven-segment displays on the DE1 board for the corresponding street/crosswalk. The two pulses are the requests coming in for the state to change. Figure 3 clearly shows that the correct transitions are made when an LPS request is made. Figure 4 can be interpreted identically for the pedestrian case.

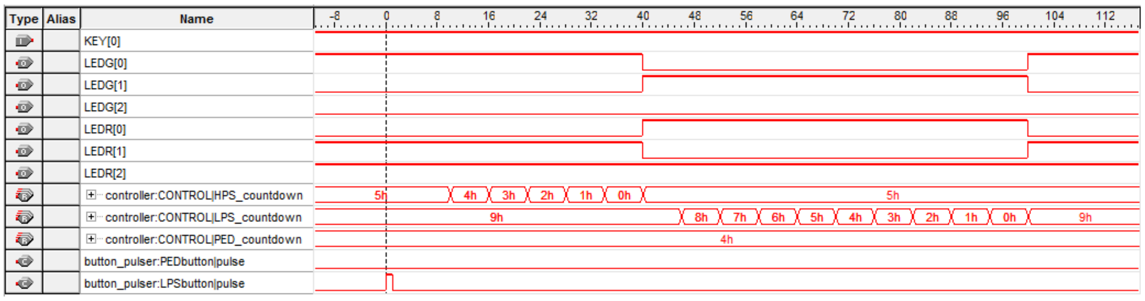

Figure 3: Transition from *RESET* to *LPS* and back.

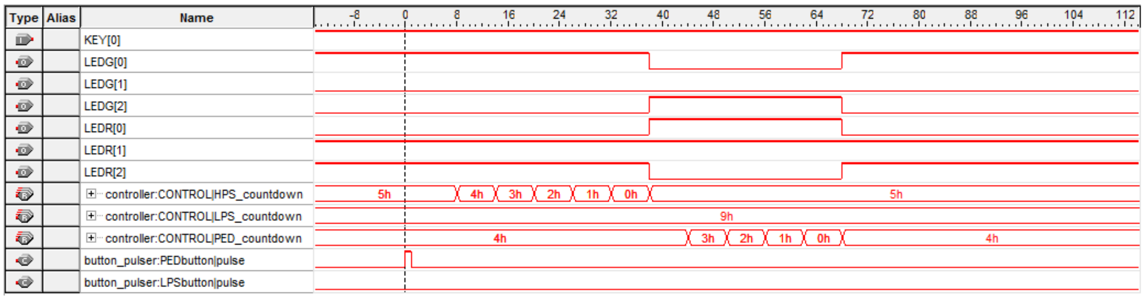

Figure 4: Transition from *RESET* to *PED* and back.

After testing that the individual state transitions worked properly, the system was tested to ensure that the logic in place to give pedestrians precedence without allowing them to stop traffic was tested. This involved manually testing the system by sending it requests before, during, and after countdowns, and making sure that the syste behaved as expected. Finally, the system was stress tested. Multiple requests were sent in short periods of time, in alternating sequence, and coinciding with state transitions. All of these tests passed; the system behaved as expected.

6 Results

The specified system was successfully designed, implemented, and tested. The traffic signal simulator functions as designed. Precedence is given to pedestrians, but not in such a way that they can stop traffic flow. The HPS always has the green light unless a request is made. The resources on the DE1 board used by the system are outlined in Table 1. The functioning design was demonstrated in Clarkson University's Undergraduate ECE Laboratory on November 10, 2010.

Table 1: DE1 Board Resource Usage

Total Logic Elements	158/18,752 (<1%)
Total Combinational Functions	153/18,752 (<1%)
Dedicated Logic Registers	59/18,752 (<1%)
Total Registers	59
Total Pins	38/315 (12%)

References

[1] J. F. Wakerly, *Digital Design Principles and Practices.* Pearson Prentice Hall, 4 ed., 2006.

[2] T. L. Floyd, *Digital Fundamentals.* Pearson Prentice Hall, 10 ed., 2009.

A VHDL Source Code Listing

Listing 1: eCLOCK.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

-- eCLOCK produces a clock enable signal
-- that counts to numHz at a rate of 50MHz.
entity eCLOCK is
    generic( numHz : integer := 1 );
    port(
            CLOCK_50MHz :in std_logic;
            eCLOCK      :out std_logic);
end eCLOCK;

architecture counterclock of eCLOCK is
    signal num: integer range 0 to 50000000 - numHz;
begin
    process(CLOCK_50MHz)
    begin
        if rising_edge(CLOCK_50MHz) then
            if num = 50000000 - numHz then
                num <= 0;
                eCLOCK <= '1';
            else
                num <= num + 1;
                eCLOCK <= '0';
            end if;
        end if;
    end process;
end architecture counterclock;
```

Listing 2: button_pulser.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

-- button_pulser takes a pulse of
-- arbitrary length, and generates
-- a pulse the length of one clock
-- pulse. This makes it ideal for
-- use with debounced buttons.

entity button_pulser is
    port(
            CLK     :in std_logic;
            button  :in std_logic;
            pulse   :out std_logic);
```

```
end button_pulser;

architecture pulser of button_pulser is
    signal shift_reg :std_logic_vector(2 downto 0);
begin
    process(CLK)
    begin
        if rising_edge(CLK) then
            shift_reg(0) <= button;
            shift_reg(1) <= shift_reg(0);
            shift_reg(2) <= shift_reg(1);
        end if;
    end process;
    pulse <=  shift_reg(1) and (not shift_reg(2));
end architecture pulser;
```

Listing 3: encoder_7seg.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity encoder_7seg is
    port(
            fourbit_num :in std_logic_vector(3 downto 0);
            display         :out std_logic_vector(6 downto 0));
end encoder_7seg;

architecture bcd of encoder_7seg is
begin
    process (fourbit_num)
    begin
        case fourbit_num is
            when "0000" =>display<= "1000000";
            when "0001" =>display<= "1111001";
            when "0010" =>display<= "0100100";
            when "0011" =>display<= "0110000";
            when "0100" =>display<= "0011001";
            when "0101" =>display<= "0010010";
            when "0110" =>display<= "0000010";
            when "0111" =>display<= "1111000";
            when "1000" =>display<= "0000000";
            when "1001" =>display<= "0010000";
            when others =>display<= "1111111";
        end case;
    end process;
end bcd;

-- If architecture is unspecified, default to hex
architecture hexadecimal of encoder_7seg is
begin
```

```
    process (fourbit_num)
    begin
        case fourbit_num is
            when "0000" =>display<= "1000000";
            when "0001" =>display<= "1111001";
            when "0010" =>display<= "0100100";
            when "0011" =>display<= "0110000";
            when "0100" =>display<= "0011001";
            when "0101" =>display<= "0010010";
            when "0110" =>display<= "0000010";
            when "0111" =>display<= "1111000";
            when "1000" =>display<= "0000000";
            when "1001" =>display<= "0010000";
            when "1010" =>display<= "0001000";
            when "1011" =>display<= "0000011";
            when "1100" =>display<= "1000110";
            when "1101" =>display<= "0100001";
            when "1110" =>display<= "0000110";
            when "1111" =>display<= "0001110";
            when others =>display<= "1111111";
        end case;
    end process;
end hexadecimal;
```

Listing 4: controller.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.NUMERIC_STD.all;

entity controller is
    generic(
                HPS_count : integer := 5;
                LPS_count : integer := 9;
                PED_count : integer := 4);

    port(
            CLOCK_50     : in std_logic;
            PED_pulse    : in std_logic;
            LPS_pulse    : in std_logic;
            RESET_EN     : in std_logic;
            CLOCK_EN     : in std_logic;
            stoplight_EN : out std_logic_vector(2 downto 0);
            golight_EN   : out std_logic_vector(2 downto 0);
            HPS_timer    : out std_logic_vector(3 downto 0);
            LPS_timer    : out std_logic_vector(3 downto 0);
            PED_timer    : out std_logic_vector(3 downto 0));
end controller;
```

```
architecture three_way_intersection of controller is
    -- State Machine Type
    type state_type is (RESET,RESET_to_LPS, RESET_to_PED, LPS, PED);
    signal current_state : state_type;

    -- Initialization Register
    signal startup : std_logic := '0';

    -- Registers for the light countdowns
    signal HPS_countdown : integer range 0 to HPS_count;
    signal LPS_countdown : integer range 0 to LPS_count;
    signal PED_countdown : integer range 0 to PED_count;

    -- Registers to queue requests
    signal LPS_queue : std_logic := '0';
    signal PED_queue : std_logic := '0';

    -- Registers to keep track of how many
    -- turns the pedestrian has taken
    signal LPS_precede : std_logic := '0';
    signal PED_already : std_logic := '0';
begin

    process(CLOCK_50)
    begin
        if rising_edge(CLOCK_50) then

            -- Take requests when applicable
            if PED_pulse = '1' then
                PED_queue <= '1';
            end if;
            if LPS_pulse = '1' then
                LPS_queue <= '1';
            end if;

            -- Initialization
            if startup = '0' then
                startup <= '1';
                LPS_queue <= '0';
                PED_queue <= '0';
                current_state <= RESET;
                HPS_countdown <= HPS_count;
                LPS_countdown <= LPS_count;
                PED_countdown <= PED_count;
                golight_EN <= "001";
                stoplight_EN <= "110";

            -- Clear state
            -- This happens asynchronously
            elsif RESET_EN = '1' then
                current_state <= RESET;
                PED_queue <= '0';
                LPS_queue <= '0';
                golight_EN <= "001";
                stoplight_EN <= "110";
                HPS_countdown <= HPS_count;
                LPS_countdown <= LPS_count;
                PED_countdown <= PED_count;

            -- All other state changes happen synchronously
            elsif CLOCK_EN = '1' then

                -- RESET STATE
                if current_state = RESET  then
                    --Transition to the correct state
                    --if there is a request.
                    if PED_queue = '1' then
                        current_state <= RESET_to_PED;
                    elsif LPS_queue = '1' then
                        current_state <= RESET_to_LPS;
                    end if;
                    golight_EN <= "001";
                    stoplight_EN <= "110";
                    HPS_countdown <= HPS_count;
                    LPS_countdown <= LPS_count;
                    PED_countdown <= PED_count;
                    LPS_precede <= '0';
                    PED_already <= '0';

                -- RESET_to_PED STATE
                elsif current_state = RESET_to_PED then
                    if HPS_countdown = 0 then
                        current_state <= PED;
                        golight_EN <= "100";
                        stoplight_EN <= "011";
                        HPS_countdown <= HPS_count;
                    else
                        HPS_countdown <= HPS_countdown -1;
                        golight_EN <= "001";
                        stoplight_EN <= "110";
                    end if;

                -- RESET_to_LPS STATE
                elsif current_state = RESET_to_LPS then
                    if HPS_countdown = 0 then
                        -- We will allow the pedestrian to
                        -- take the turn of the LPS, since
                        -- it has higher priority.
                        if PED_queue = '1' then
                            current_state <= PED;
                            golight_EN <= "100";
                            stoplight_EN <= "011";
                            -- We make a note that the pedestrian
                            -- has already stolen a turn once
                            LPS_precede <= '1';

                        else -- go to LPS as planned
                            current_state <= LPS;
                            golight_EN <= "010";
                            stoplight_EN <= "101";
                        end if;
                        HPS_countdown <= HPS_count;
                    else
                        HPS_countdown <= HPS_countdown -1;
                        golight_EN <= "001";
                        stoplight_EN <= "110";
                    end if;

                -- LPS STATE
                elsif current_state = LPS then
                    if LPS_countdown = 0 then

                        -- Don't allow the pedestrian to steal
                        -- a turn if they have already done so
                        -- twice.
                        if PED_queue = '1'
                        and LPS_precede = '1'
                        and PED_already = '1' then
                            current_state <= RESET_to_PED;
                            golight_EN <= "001";
                            stoplight_EN <= "110";
                            LPS_precede <= '0';
                        -- If the above condition was false then
                        -- it is okay for the pedestrian to steal
                        -- the turn.
                        elsif PED_queue = '1' then
                            current_state <= PED;
                            golight_EN <= "100";
                            stoplight_EN <= "011";
                            PED_already <= '1';

                        else --Go to RESET as planned.
                            current_state <= RESET;
                            golight_EN <= "001";
                            stoplight_EN <= "110";
                        end if;
                        LPS_queue <= '0';
                        LPS_countdown <= LPS_count;
                        HPS_countdown <= HPS_count;
                        PED_countdown <= PED_count;
                        else
                            LPS_countdown <= LPS_countdown-1;
                            golight_EN <= "010";
                            stoplight_EN <= "101";
                        end if;

                    -- PED STATE
                    elsif current_state = PED then
                        if PED_countdown = 0 then
                            -- Go to LPS if it's turn was stolen from PED
                            if LPS_precede = '1' and PED_already = '0' then
                                current_state <= LPS;
                                golight_EN <= "010";
                                stoplight_EN <= "101";

                            -- If the turn was not stolen, let HPS go first
                            elsif LPS_queue = '1' then
                                current_state <= RESET_to_LPS;
                                golight_EN <= "010";
                                stoplight_EN <= "101";
                                LPS_precede <= '1';

                            else -- RESET as planned
                                current_state <= RESET;
                                golight_EN <= "001";
                                stoplight_EN <= "110";
                            end if;
                            PED_queue <= '0';
                            PED_countdown <= PED_count;
                            HPS_countdown <= HPS_count;
                            LPS_countdown <= LPS_count;
                        else
                            PED_countdown <= PED_countdown-1;
                            golight_EN <= "100";
                            stoplight_EN <= "011";
                        end if;

                    -- ERROR STATE
                    -- Never used purposely, but will pick up error if
                    -- bits are flipped by EM radiation, etc.
                    else
                        golight_EN <= "000";
                        stoplight_EN <= "111";
                    end if;
                end if;
            end if;
end process;

HPS_timer <= std_logic_vector(to_unsigned(HPS_countdown, 4));
```

```
    LPS_timer <= std_logic_vector(to_unsigned(LPS_countdown, 4));
    PED_timer <= std_logic_vector(to_unsigned(PED_countdown, 4));

end architecture three_way_intersection;
```