Zachary Byron and Adrian Marusic

28 April 2016

Traffic Light Implementation with Crosswalk

Final Project Write Up

**Abstract:** The primary purpose of this project is to demonstrate proficiency in designing, programming, and building a system that implements digital logic. Furthermore students will demonstrate and document their project to the professor for grading. The project described in this document implements the Cyclone II FPGA to control the timing and operation of a four-way traffic stop with four-way crosswalks.

**Specifications / Requirements :**

Implement a four-way traffic light using the Cyclone II FPGA. The traffic light displays three lights in one direction, 12 lights total. The logic controls these lights to indicate that traffic should stop in one direction and go or slow down (in preparation for stopping). Each direction receives equal share of light time (half green or yellow, and half red). Additionally, to accommodate foot traffic, crosswalk buttons have been implemented that interrupt the cycle of the lights. The button causes a disruption in the equal share of light time in the intersection because it gives priority to the walker so that the lights’ next cycle allows them to cross the street safely.

**Description of Operation:**

The logic has 6 states. As time passes, the output displays the next state depending on the number of clock ticks. As someone presses the crosswalk button, the lights will cycle to a safe state (ie both directions display red lights). The following state will be one where the pedestrian can cross safely. This means the traffic lights are green parallel to the pedestrian. Depending on the state of the intersection when the crosswalk is pressed, the lights will move into the expected next state, or move to an unexpected state, whichever benefits the pedestrian.

**Diagrams & Drawings**:

Block Diagram:



State Table:

|  |  |  |
| --- | --- | --- |
| Preset State | Input (button 1 )/(button2)  B1/B2=0 /0 0/1 1/0 1/1 | Output |
| A | B B B B | LedRed = “000001”  LedGreen= “001000” |
| B | C C F C | LedRed = “000001”  LedGreen= “110000” |
| C | D D F D | LedRed = “001001”  LedGreen= “000000” |
| D | E E E E | LedRed = “001000”  LedGreen= “000001” |
| E | F C F F | LedRed = “001000”  LedGreen= “000110” |
| F | A C A A | LedRed = “001001”  LedGreen= “000000” |

**State Diagram:**

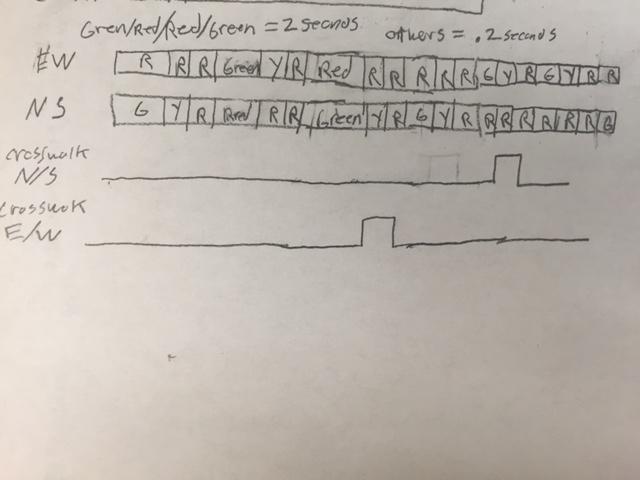


Because each direction has the same light color, the state diagram can be simplified to north-south and east-west. The diagram shows the normal cycle of the lights (sequential order) and the interruption due to pressing the crosswalk buttons. There are two crosswalk buttons, one for each direction of walking.

**Flow chart**



**Timing Diagram:**



**Programs for FPGA:**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.all;

USE IEEE.STD\_LOGIC\_unsigned.all;

ENTITY trafficlightwithcrosswalk IS

PORT (CLOCK\_50: IN STD\_LOGIC; --onboard 50MHz clock

KEY: IN STD\_LOGIC\_VECTOR (1 DOWNTO 0); --Used as crosswalk buttons

LEDR: OUT STD\_LOGIC\_VECTOR (5 downto 0); --red light representation

LEDG: OUT STD\_LOGIC\_VECTOR (5 DOWNTO 0));

--green and yellow light representation

END trafficlightwithcrosswalk;

ARCHITECTURE stateMachine OF trafficlightwithcrosswalk IS

TYPE state\_type IS (s0, s1, s2, s3, s4, s5); --represents states in state diagram

SIGNAL state: state\_type; --allows for use in case statement

CONSTANT Longdelay: INTEGER := 100000000;--longer delay for green states of about 2 seconds

CONSTANT Shortdelay: INTEGER := 10000000;--shorter delay for yellow and red states for about .2 seconds

BEGIN

PROCESS (Clock\_50, KEY(1), KEY(0))

variable Count: INTEGER RANGE 0 TO 100000000; --counter “slows” clock\_50

variable CWNS: std\_LOGIC := '0'; -- counter store a north/south crosswalk press

variable CWEW: std\_LOGIC := '0'; -- counter store a East/west crosswalk press

BEGIN

IF (Clock\_50'EVENT AND rising\_edge(Clock\_50)) THEN --trigger statement for crosswalk at rising edge of clock

IF (KEY(0) = '0' AND CWNS = '0') THEN

CWNS := '1'; -- activates the crosswalk north-south

END IF;

IF (KEY(1) = '0' AND CWEW = '0') THEN

CWEW := '1'; --activates crosswalk east-west

END IF;

CASE state IS -- identifies which state the machine is in

WHEN s0 =>

IF Count = Longdelay THEN

--Triggers after the counter delays

state <=s1; --Next state

Count := 0; --Resets count

ELSE

state <=s0; --remains in current state until time limit reached

Count := Count + 1; -- counts up timer

END IF;

WHEN s1 =>

IF Count = Shortdelay AND CWNS = '1' AND CWEW = '0' THEN

--if crosswalk button is activated, change next state

state <= s5;

Count := 0; --reset count

CWNS := '0'; --reset crosswalk buttons

CWEW := '0';

ELSIF Count =Shortdelay THEN

state <= s2; --normal counter progression

Count := 0; --reset count for next state

ELSE

state <= s1; --stay in state until time is met

Count := Count + 1;

END IF;

WHEN s2 =>

IF Count = Shortdelay AND CWNS = '1' AND CWEW = '0' THEN

state <= s5;

Count := 0; --reset count

CWNS := '0'; --reset crosswalk buttons

CWEW := '0';

ELSIF Count = Shortdelay THEN

state <= s3; --normal counter progression

Count := 0; --reset count for next state

ELSE

State <= s2; --stay in state until time is met

Count := Count + 1;

END IF;

WHEN s3 =>

IF Count = Longdelay THEN

--crosswalk buttons don’t change operation here

state <= s4;

Count := 0;

ELSE

state <= s3;

Count := Count + 1;

END IF;

WHEN s4 =>

IF Count = Shortdelay AND CWNS = '0' AND CWEW = '1' THEN --triggers the east-west crosswalk

state <= s2; -- shifts to alternate state after count

Count := 0;

CWNS := '0'; --clears crosswalk buttons

CWEW := '0';

ELSIF Count = SEC1 THEN -- this is a normal operation in the cycle

state <= s5;

Count := 0;

ELSE --stays in state until time is met

state <= s4;

Count := Count + 1;

END IF;

WHEN s5 =>

IF Count = Shortdelay AND CWNS = '0' AND CWEW = '1' THEN

state <= s2;

Count := 0;

ELSIF Count = SEC1 THEN --this is a normal operation in the counter

state <= s0;

Count := 0;

ELSE --stays in state until time is met

state <= s5;

Count := Count + 1;

END IF;

WHEN OTHERS => STATE <= s0; --catch all

END CASE;

END IF;

END PROCESS;

PROCESS(state)

BEGIN

CASE state is --second case statement gives light outputs to states

when s0 => LEDR <= "000001"; LEDG <= "001000"; -- Red/Green

when s1 => LEDR <= "000001"; LEDG <= "110000";

-- Red/Yellow(signified by 2 adjacent green lights)

when s2 => LEDR <= "001001"; LEDG <= "000000"; --(Red/Red)

when s3 => LEDR <= "001000"; LEDG <= "000001"; --(Green/Red)

when s4 => LEDR <= "001000"; LEDG <= "000110"; --(Yellow/Red)

when s5 => LEDR <= "001001"; LEDG <= "000000"; --(Red/Red)

when others => LEDR <= "000000"; --catch all

END CASE;

END PROCESS;

END stateMachine;

-----------------------------------------------------------------------------

**Summary**:

Students used the accumulation of their knowledge throughout the semester to design, synthesize and construct a digital logic circuit. This project used techniques involving state diagrams, boolean algebra, state reduction and VHDL to implement a four way traffic light that permits safe and speedy crosswalking.

**Results/Conclusions:**

Our device was able to meet almost all of the specifications in our proposal. We we not able to implement the Left-Turn only lane due to time constraints but that that was a reach goal in our project. We were able to create, however, a functional 4 way traffic intersection with realistic delay in the state changes of the lights as well as implement a pedestrian crosswalk. In all 4 roadways. Realistically we were not able to use a yellow LED due to the lack of one on the logic board. We compensated for this by using 2 green LED’s simultaneously to represent the yellow light. One unexpected result was that by assuming that the pedestrian would walk if able to, we were able to limit the complexity of the crosswalk and make if for the most part, a simple implementation into our base code.