Traffic Signal System

Created By Volodymyr Semenov

Template Created by: Dr. Winikus
CSE241 Spring 2022

Introduction

The created system is to be used as a traffic signal controller for an intersection of two one-way roads. It relies on a counting sensor that changes bits after every car that passes as well as a weather sensor and an enable sensor that can turn off the counting sensor. The controller will control 2 traffic lights which can either be red or green. This paper will discuss the problem, the development of the system, as well as the construction and testing of the system.

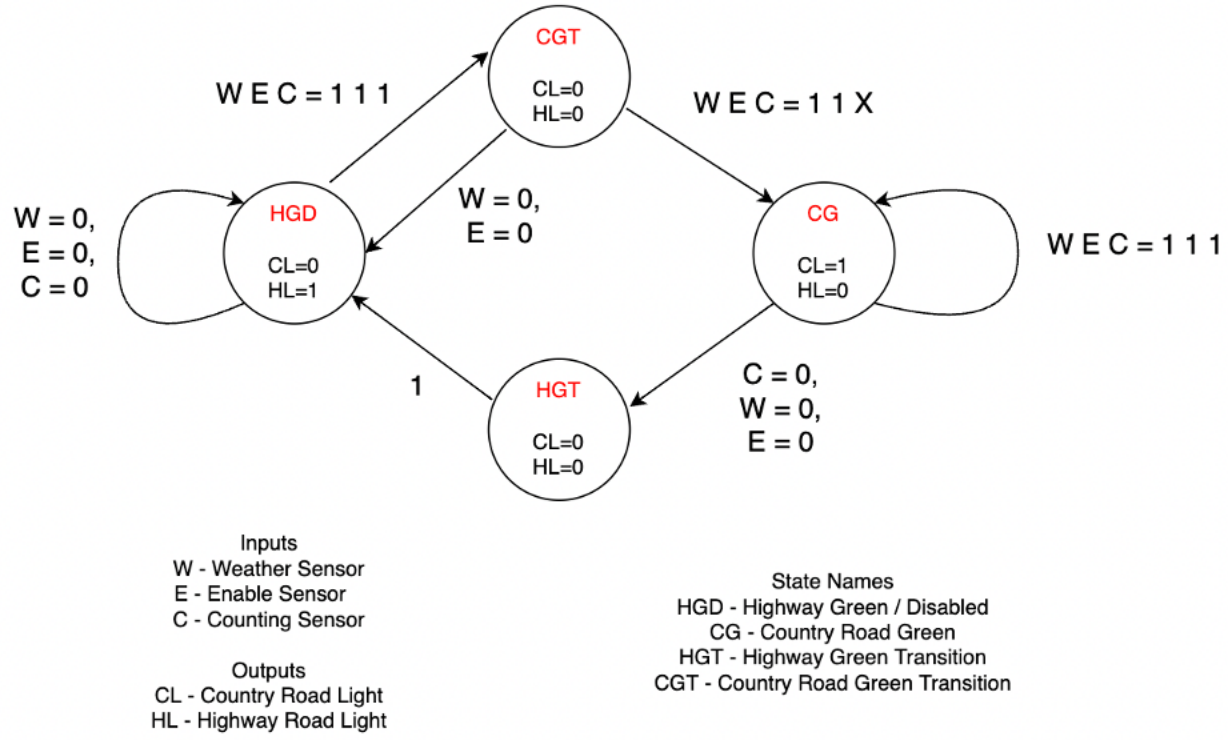
Design Problem

The system was created to safely allow cars to cross an intersection between two one-way roads. The design has to incorporate a counting sensor that counts the cars that pass the intersection. When it snows, the counting sensor needs to be disabled with a weather sensor. Additionally, there needs to be an enable sensor that can disable the counting sensor on its own. When the counting sensor is disabled, the country road has to have a red light. Each signal can be red or green, and both signals should never be green at the same time so crashes would be less likely to occur. There has to be a period where both signals are red in between signal changes to decrease collisions.

Design Development

The creation of the design began with simplifying the design problem into only necessary information and creating a state diagram that fits the requirements of the problem. Then a state table was created based on the state diagram. Next, the states were assigned Flip Flop values which allowed for the creation of a transition table. The values on the transition table were K-mapped and then groupings of Prime Implicants were made on the K-maps which led to the formation of transition and output equations. Next, an SV file was created to test the system and make sure that it transitioned between the states correctly and had the correct outputs. Finally, a schematic was created following the 4 equations which were used to connect the inputs of the Flip Flops and the outputs.

State Diagram



State Table

Present State	Next State								Output	
State	W = 0				W = 1				CL	HL
	E = 0		E = 1		E = 0		E = 1			
	C = 0	C = 1	C = 0	C = 1	C = 0	C = 1	C = 0	C = 1		
CGT	HGD	HGD	HGD	HGD	HGD	HGD	CG	CG	0	0
HGT	HGD	HGD	HGD	HGD	HGD	HGD	HGD	HGD	0	0
CG	HGT	HGT	HGT	HGT	HGT	HGT	HGT	CG	1	0
HGD	HGD	HGD	HGD	HGD	HGD	HGD	HGD	CGT	0	1

State Assignments

Country Green Transition (CGT) - 00
Highway Green Transition (HGT) - 01
Country Green (CG) - 10
Highway Green/Disabled (HGD) - 11

Transition Table

Present State			Next State														Output	
State Name	Q1	Q0	W = 0								W = 1						CL	HL
			E = 0				E = 1				E = 0				E = 1			
			C = 0		C = 1		C = 0		C = 1		C = 0		C = 1		C = 0		C = 1			
			Q1*	Q0*	Q1*	Q0*	Q1*	Q0*	Q1*	Q0*	Q1*	Q0*	Q1*	Q0*	Q1*	Q0*	Q1*	Q0*		
CGT	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0
HGT	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
CG	1	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	0
HGD	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1

Transition Equations

D1 = Q1' + Q0(E' + C' + W') + Q0'ECW

D0 = E' + W' + Q1'Q0 + Q1C'

Output Equations

Country Light (CL) = Q1 * Q0'

Highway Light (HL) = Q1 * Q0

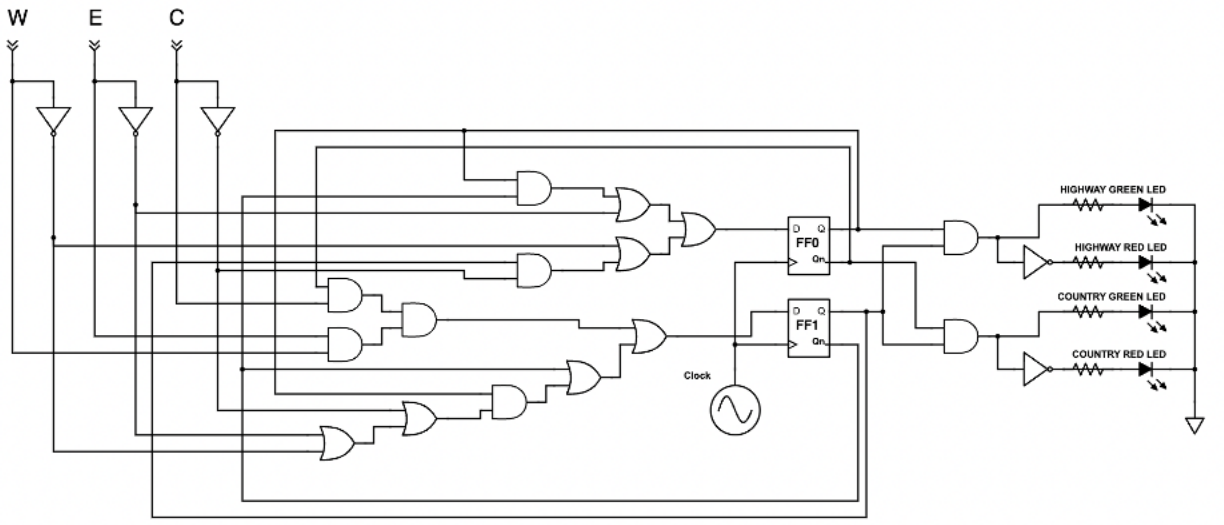
D1 and D0 are flip flop D inputs. W is the weather sensor, E is the enable sensor, and C is the counting sensor.

Bill of Materials

- 2 Red LEDs
- 2 Green LEDs
- Wires
- 1 Full Sized Breadboard
- 4 100 Ohm Resistors
- 1 SN74HC04 chip
- 2 SN74HC08 chip
- 2 SN74HC32 chip
- 1 CD74HC175 chip
- 1 Counting Sensor
- 1 Input Sensor
- 1 Enable Sensor / Switch
- 1 Clock

Physical Construction

To physically build the system, start by adding the two flip flops and choosing one to be the MSB and one to be the LSB. Next, connect the 2 green LEDs to GRD on the negative end and to the equations Q1 * Q0 and Q1 * Q0’ with a resistor in series on the positive end. Connect the Red LEDs to the inverted inputs of the green LEDs. Next, connect the clock to the clock input on each flip flop. Lastly, connect the D input on the flip flops to with the MSB Flip flop having D1 = Q1' + Q0(E' + C' + W') + Q0'ECW and the LSB having D0 = E' + W' + Q1'Q0 + Q1C'.



Testing

To test the system, monitor it and ensure it passes the following 8 test cases. In these cases, we will check that it transitions correctly based on the current state and the input and that the outputs are correct for the state.

First, test that the system correctly stays in State HGD (Highway Green / Disabled). It needs to stay in this state when the weather sensor is 0, the enable sensor is 0, or the counting sensor is 0. Three test cases are setting WEC to 110, 101, and 011 with W being the first bit, E being the second bit, and C being the bit, and the system should stay in the HGD state for all 3 of these input combinations. Additionally, make sure the Highway has a green light and the Country Road has a red light while it is in this state.

Next, check that it can transition from this state to the CGT State (Country Green Transition) and back correctly. While in the HGD state, set WEC to 111 and it should transition to the CGT state. In this state, the lights on both roads should be red. While the system is in the

CGT state, quickly switch W or E or both to 0 and it should transition back to the HGD state with the country signal being red and the highway signal being green.
Next check that it can transition from the CGT state to the CG state (Country Green). You can do this by setting W and E to 1 while in the HGD state and it should transition to the CGT state and then the CG state. In the CG state, there should be a red light on the highway and a green light on the country road.

The system should stay in this CG state while all 3 inputs are 1, so wait a couple of clock cycles and make sure the system doesn't transition to another state. Finally, when any input switches to 0, the system should cycle to the HGT state and then back to the HGD state. Test this by switching any of the inputs to 0. In the HGT state, make sure both the country road light and the highway light are red.

SystemVerilog Implementation

When creating the system verilog file, first an additional DFF module was made which models a D-Flip Flop. It was instantiated twice in the main SignalSystem module to hold the state values. Behavioral Logic was used to cause the flip flop to update on positive clock edges. Continuous Assignment was used in the SignalSystem module to improve readability over a gate-level implementation. Behavioral Logic was used in the testbench file to set the inputs needed to go through the test cases needed to verify that the system generally works.

```
//Name: Volodymyr Semenov
//Date Created: May 15
//Purpose: Signal State Machine
//Modules present: DFF, SignalSystem
//Date of last update: May 17


//D-Flip Flop Module to be used in the Signal System
module DFF(D, Clk, Q, QN);
  //Inputs
  input D, Clk;
  //Outputs
  output Q, QN;
  //Wires to hold Values
  reg Q, QN;

  //Sets initial flip flop value to 1
  initial
        begin
        Q <= 1;
        end

  //On Clock trigger updates Output Q
  always@(posedge Clk)
        Q <= D;

  //Sets output QN to the opposite of Q on every update of Q
  always@(Q)
        QN <= !Q;
```

```
endmodule


module SignalSystem(W, E, C, CLK, HL, CL);
  //Inputs
  input W, E, C, CLK;
  //Outputs
  output HL, CL;
  //Wires to Hold Inputs and Outputs of called Flip Flop Modules
  wire Q_0, QN_0, D_0;
  wire Q_1, QN_1, D_1;

  //Instantiates DFF twice for state storage
  DFF S0(.D(D_0), .Clk(CLK), .Q(Q_0), .QN(QN_0));
  DFF S1(.D(D_1), .Clk(CLK), .Q(Q_1), .QN(QN_1));

  //Continuous Assignment Statements for D-Flip Flop Inputs
  assign D_0 = ~E | ~W | (QN_1 & Q_0) | (Q_1 & ~C);
  assign D_1 = QN_1 | (QN_0 & E & C & W) | (Q_0 & (~E | ~C | ~W ));

  //Continuous Assignment Statements for both Outputs
  assign HL = Q_0 & Q_1;
  assign CL = QN_0 & Q_1;
endmodule
```

SystemVerilog Test Bench

```
//Name: Volodymyr Semenov
//Date Created: May 15
//Purpose: Signal State Machine Testing
//Modules present: SignalSystem_tb
//Date of last update: May 17

//SignalSystem Testbench Module
module SignalSystem_tb;
  //Reg to hold inputs
  reg W_tb, E_tb, C_tb, CLK_tb;
  //Wires for outputs
  wire HL_tb, CL_tb;
  //Call on SignalSystem Module
  SignalSystem SystemTest(.W(W_tb),.E(E_tb),.C(C_tb),.CLK(CLK_tb),.HL(HL_tb),.CL(CL_tb));

  //Sets initial Values
   initial
        begin
        W_tb = 1;
        E_tb = 1;
        C_tb = 0;
        CLK_tb = 0;
        end

  //Cycles through different States with 3 inputs
  //All Inputs present to improve readability
  always
        begin
        //Should Stay in State HGD
        #2 W_tb = 1; E_tb = 1; C_tb = 0;
        #2 W_tb = 0; E_tb = 1; C_tb = 1;
```

```
        #2 W_tb = 1; E_tb = 0; C_tb = 1;

        //Should transition to State CGT
        #2 W_tb = 1; E_tb = 1; C_tb = 1;

        //Should transition to State HGD
        #2 W_tb = 1; E_tb = 0; C_tb = 1;

        //Should transition to State CGT
        #2 W_tb = 1; E_tb = 1; C_tb = 1;

        //Should transition to and stay in State CG
        #2 W_tb = 1; E_tb = 1; C_tb = 1;
        #2 W_tb = 1; E_tb = 1; C_tb = 1;

        //Should transition to State HGT
        #2 W_tb = 1; E_tb = 1; C_tb = 0;

        //Should transition to State HGD
        #2 W_tb = 0; E_tb = 0; C_tb = 0;
        end

    //Cycles Clock
    always
        begin
        #1 CLK_tb = 1;
        #1 CLK_tb = 0;
        end

    //Setup Waveform
    initial
        begin
        //Creates File
      $dumpfile("DFFTest.vcd");
        //Dumps Variables
        $dumpvars;
        end

    initial
        begin
        //Seperates from warnings
        $display("----------------- Results -----------------");
        //Shows Inputs
        $display("CLK\tW\tE\tC\tHL\tCL");
        //Watches inputs and prints a new line every time inputs changes
        $monitor("%d\t%d\t%d\t%d\t%d\t%d",CLK_tb,W_tb, E_tb,C_tb,HL_tb, CL_tb);
        end

    //Ends the testbench after 2 input cycles
    initial
        #44 $finish;

    endmodule
```

Analysis of the System

To analyze the system with a log file of inputs and outputs, make sure it passes the test cases mentioned in the testing section. With a log file, it's not possible to manually change the inputs, but it is possible to view how the inputs changed and make sure that the outputs changed correctly. Not all test cases might be present, but analyze the system with what is available. The transition table could also be used as a reference of the intended system behavior. If using the transition table, check that the outputs are correct based on the previous state and inputs.

Analysis with the waveform should be almost identical to analysis with the log file of the inputs and outputs. The only difference would be that you need to look at the wave and determine if it is at a 0 or a 1 at every period. Other than that, follow the directions for the analysis with the log file.

Analysis of the system with a video recording should also be similar to analysis with the log file, but you need to determine what the inputs and outputs are instead of reading if they are 1s or 0s. You can assume the counting sensor switches between 1 and 0 after every car passes and you can assume that the weather sensor is a 1 if it is not snowing and a 0 if it is. If the enable sensor is visible in the video, figure out its value from that, but if it is not, assume its value based on the behavior of the system. If the signals never switch, assume it is a 0. Otherwise, it is a 1. Other than that, follow the directions for the analysis with the log file.

Conclusion

A signal control system was designed, implemented, and tested for an intersection based on the requirements of the specified design problem. It was designed using the principle of a Moore State machine and sequential and combinational logic techniques. The system worked as expected and passed all of the created test cases. A system verilog design file and testbench were created which modeled the system in a HDL. A physical implementation was also created on a breadboard that worked as expected.