Brennan Reamer
Gabriel Sytangco
Professor Junsangsri
ELEC-2275
11 December 2020

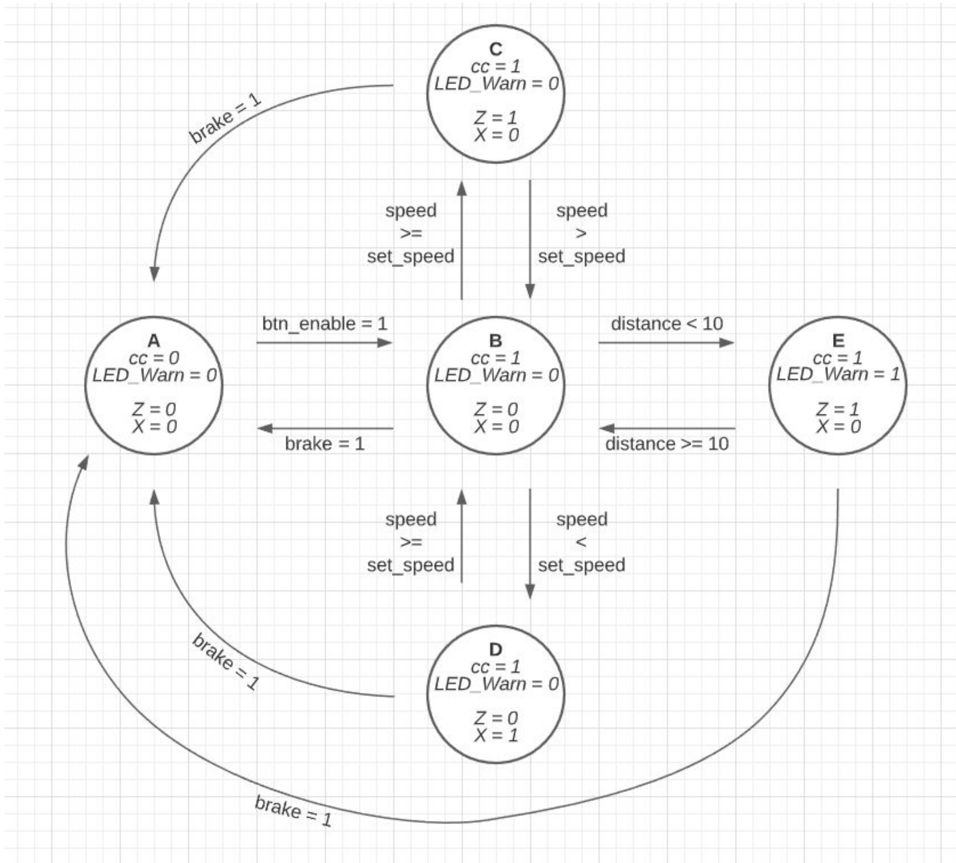
Digital Logic Final Project

Abstract

We were tasked with developing an adaptive cruise control system for a vehicle. The constraints were that

- The driver must be able to set a speed anywhere between 0 and 64 mph
- The vehicle must maintain that speed by accelerating or decelerating by 1 mph at each clock cycle until the current speed matches the set speed
- The cruise control is deactivated and needs to be re-enabled manually when the brake is pressed
- The cruise control allows the driver to press the accelerator, but when it is released, the cruise control is re-enabled automatically
- The vehicle must keep at least 10 meters behind any vehicles in front of it, as well as warning the driver and activating the brakes if the vehicle is approaching another vehicle too fast

Solution


C
cc = 1
LED_Warn = 0
Z = 1
X = 0
A
cc = 0
LED_Warn = 0
Z = 0
X = 0
B
cc = 1
LED_Warn = 0
Z = 0
X = 0
E
cc = 1
LED_Warn = 1
Z = 1
X = 0
D
cc = 1
LED_Warn = 0
Z = 0
X = 1
brake = 1
speed >= set_speed
speed > set_speed
btn_enable = 1
brake = 1
distance < 10
distance >= 10
speed >= set_speed
speed < set_speed
brake = 1
brake = 1

Code:

```verilog
/*The following module is designed to be an adaptive cruise control unit. The program allows the user to toggle cruise control on any input speed
equal to or lower than 64 mph. The body of the code is almost entirely state based, with automated accelerate and deccelerate functions for matching
speeds. The user may also manually accelerate or deccelerate, deactivating the cruise control if brakes are applied. Sensors are also simulated in
the program for objects in front of the user's car. Objects/other cars cannot be closer than 10 meters from the user's car. Otherwise, brakes are
applied and a warning LED will flash.


Inputs
------------------------------------------------------------------------------------------------------------------------------------------------------

initial_speed = 6-bit binary input to represent the initial speed of the user's car. Maximum speed is 64.
speed_otherCar = 6-bit binary input to represent the speed of the other car. Maximum speed is 64.
btn_enable = Button that enables/toggles cruise control
clk = Input clock onboard
brake = Button that toggles the brakes when held down
accel = Button that toggles the accelerator when held down
btn_reset = Button that enables/toggles a reset, setting all 7-segments to 00
distance = 3-bit binary input to represent the distance between the user's car and the car in front of it. Maximum distance is 10.

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


Outputs
------------------------------------------------------------------------------------------------------------------------------------------------------

cc_led = LED that indicates cruise control is on/off
LED_Warning = LED that toggles when the user car is too close to the other car
seg1_set_speed = 7-segment display showing the set speed of the user's car (0-9)
seg2_set_speed = 7-segment display showing the set speed of the user's car (0-6)
seg1_speed = 7-segment display showing the current speed of the user's car (0-9)
seg2_speed = 7-segment display showing the current speed of the user's car (0-6)
seg1_otherCar_speed = 7-segment display showing the speed of the other car (0-9)
seg2_otherCar_speed = 7-segment display showing the speed of the other car (0-6)

------------------------------------------------------------------------------------------------------------------------------------------------------
*/

module FP_CruiseControl (input [5:0] initial_speed, speed_otherCar, input btn_enable, clk, brake, accel, btn_reset, input [3:0] distance, output reg cc_LED, LED_Warning,

    //Regulate a variable to be used with the clock
    reg seg_clk;
    //Regulate variables X and Z for the states
    reg X, Z;
    //Create a parameter for the maximum count of the clock
    parameter MAX_COUNT = 10000000;
    //Regulate a 24-bit binary variable to represent the count of the clock
    reg [23:0] count;
    //Regulate a 6-bit binary variable for the set speed
    reg [5:0] set_speed;
    //Regulate a variable for the current speed
    reg speed;
```

```
----------------

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

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

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

.ED, LED_Warning, output reg [6:0] seg1_set_speed, seg2_set_speed, seg1_speed, seg2_speed, seg1_otherCar_speed, seg2_otherCar_speed);
```

```verilog
//Create local parameters for the next state
localparam A = 0, B = 1, C = 2, D = 3, E = 4;
//Regulate 2 2-bit binary variables representative of the car's state
reg [1:0] present_state, next_state;


//Next State

always @ (*) begin
   case (present_state)
      A: begin
         if (btn_enable == 0) begin
            set_speed = speed;
            next_state = B;
         end else begin
            next_state = A;
         end
      end
      B: begin
         if (brake == 0) begin
            next_state = A;
         end else if ((distance < 10) || ((speed_otherCar < speed) && (distance < 20))) begin
            next_state = E;
         end else if (speed > set_speed) begin
            next_state = C;
         end else if (speed < set_speed) begin
            next_state = D;
         end else begin
            next_state = B;
         end
      end
      C: begin
         if (brake == 0) begin
            next_state = A;
         end else if (speed <= set_speed || accel == 0) begin
            next_state = B;
         end else begin
            next_state = C;
         end
      end
      D: begin
         if (brake == 0) begin
            next_state = A;
         end else if (speed >= set_speed) begin
            next_state = B;
         end else begin
            next_state = D;
         end
      end
```

```
            E: begin
               if (brake == 0) begin
                  next_state = A;
               end else if (distance >= 10 && speed_otherCar > speed) begin
                  next_state = B;
               end else begin
                  next_state = E;
               end
            end
         endcase

         //Output

         case (present_state)
            A: begin
                  cc_LED = 0; LED_Warning <= 0; Z <= 0; X <= 0;
               end
            B: begin
                  cc_LED = 1; LED_Warning <= 0; Z <= 0; X <= 0;
               end
            C: begin
                  cc_LED = 1; LED_Warning <= 0; Z <= 1; X <= 0;
               end
            D: begin
                  cc_LED = 1; LED_Warning <= 0; Z <= 0; X <= 1;
               end
            E: begin
                  cc_LED = 1; LED_Warning <= 1; Z <= 1; X <= 0;
               end
            default: begin
                        cc_LED = 0; LED_Warning <= 0; Z <= 0; X <= 0;
                     end
         endcase
      end



      //Maintain Speed

      always @ (posedge clk) begin
         speed = initial_speed;
         if (Z == 1) begin
            speed = speed - 1;
         end else if (X == 1) begin
            speed = speed + 1;
         end
         //Clk
         if (count < MAX_COUNT) begin
            count = count + 1;
         end else begin
            count <= 0;
            seg_clk <= ~seg_clk;
```

```
        end
    end

    //Reset
    always @ (posedge seg_clk) begin
        if (btn_reset == 0) begin
            present_state <= A; //Initial State
        end else begin
            present_state <= next_state;
        end
    end

//*Set/Input Speed
//7-seg 1

    always @ (*) begin
        case (set_speed)
            0:seg1_set_speed = 7'b1000000;
            1:seg1_set_speed = 7'b1111001;
            2:seg1_set_speed = 7'b0100100;
            3:seg1_set_speed = 7'b0110000;
            4:seg1_set_speed = 7'b0011001;
            5:seg1_set_speed = 7'b0010010;
            6:seg1_set_speed = 7'b0000010;
            7:seg1_set_speed = 7'b1111000;
            8:seg1_set_speed = 7'b0000000;
            9:seg1_set_speed = 7'b0010000;
        default: seg1_set_speed = 7'b1000000;
        endcase
        case (set_speed)
            0:seg2_set_speed = 7'b1000000;
            1:seg2_set_speed = 7'b1111001;
            2:seg2_set_speed = 7'b0100100;
            3:seg2_set_speed = 7'b0110000;
            4:seg2_set_speed = 7'b0011001;
            5:seg2_set_speed = 7'b0010010;
            6:seg2_set_speed = 7'b0000010;
        default: seg2_set_speed = 7'b1000000;
        endcase
        case (speed)
            0:seg1_speed = 7'b1000000;
            1:seg1_speed = 7'b1111001;
            2:seg1_speed = 7'b0100100;
            3:seg1_speed = 7'b0110000;
            4:seg1_speed = 7'b0011001;
            5:seg1_speed = 7'b0010010;
            6:seg1_speed = 7'b0000010;
            7:seg1_speed = 7'b1111000;
            8:seg1_speed = 7'b0000000;
            9:seg1_speed = 7'b0010000;
        default: seg1_speed = 7'b0000000;
        endcase
```

```
        case (speed)
            0:seg2_speed = 7'b1000000;
            1:seg2_speed = 7'b1111001;
            2:seg2_speed = 7'b0100100;
            3:seg2_speed = 7'b0110000;
            4:seg2_speed = 7'b0011001;
            5:seg2_speed = 7'b0010010;
            6:seg2_speed = 7'b0000010;
        default: seg2_speed = 7'b0000000;
        endcase
        case (speed_otherCar)
            0:seg1_otherCar_speed = 7'b1000000;
            1:seg1_otherCar_speed = 7'b1111001;
            2:seg1_otherCar_speed = 7'b0100100;
            3:seg1_otherCar_speed = 7'b0110000;
            4:seg1_otherCar_speed = 7'b0011001;
            5:seg1_otherCar_speed = 7'b0010010;
            6:seg1_otherCar_speed = 7'b0000010;
            7:seg1_otherCar_speed = 7'b1111000;
            8:seg1_otherCar_speed = 7'b0000000;
            9:seg1_otherCar_speed = 7'b0010000;
        default: seg1_otherCar_speed = 7'b1000000;
        endcase
        case (speed_otherCar)
            0:seg2_otherCar_speed = 7'b1000000;
            1:seg2_otherCar_speed = 7'b1111001;
            2:seg2_otherCar_speed = 7'b0100100;
            3:seg2_otherCar_speed = 7'b0110000;
            4:seg2_otherCar_speed = 7'b0011001;
            5:seg2_otherCar_speed = 7'b0010010;
            6:seg2_otherCar_speed = 7'b0000010;
        default: seg2_otherCar_speed = 7'b1000000;
        endcase
    end

endmodule
```

Discussion

The verilog code we wrote was able to run successfully on the digital logic board and simulate an adaptive cruise control system. We faced problems with displaying two-digit numbers on the seven-segment displays. They kept printing only one digit on each of the seven-segment displays instead of the whole two-digit number. This was corrected by assigning multiple variables to the seven-segment display outputs, which we then could change individually in the cases to output two-digit numbers properly. We also had challenges with our always blocks. Originally, we had too many always blocks, so the code would not run. We had an always block for each section of our code.This problem was solved by condensing our code to fit into less always blocks. Once we condensed our code, the error no longer showed up. This project could be worked on further by implementing algorithms to accelerate and decelerate the car based on the speed limits and the distance to the car in front. These algorithms could turn the adaptive cruise control system into a self-driving system for a vehicle.