Abstract

This project designs and simulates a digital traffic light controller for a T-intersection to enhance traffic flow and safety. The controller manages three traffic directions and includes a pedestrian button for safe crossing. Operating in four modes, the system cycles through specified intervals, reverting to pedestrian mode when needed. The design process involved creating a state diagram, state table, Boolean functions, and sequential circuit using Verilog. A behavioral coding approach improved reliability. Rigorous testing ensured accurate mode transitions and timing, highlighting the project's contribution to cost-effective, reliable traffic management systems.

TABLE OF CONTENTS

1 Introduction

Traffic light controllers are essential components in modern traffic management systems, playing a crucial role in maintaining the smooth and safe flow of vehicles at intersections. Their importance cannot be overstated as they help prevent accidents, reduce traffic congestion, and improve overall road safety. A traffic light controller requires a precise and error-free implementation to ensure that signal transitions are timely and accurate, thus preventing potential hazards. While the seamless transitioning and reliability are paramount, the production of these controllers must also be cost-effective to facilitate widespread adoption and integration into existing infrastructure. Balancing these aspects – safety, precision, and cost-efficiency – is critical for developing a successful traffic light controller system.

1.1 Project Target

The objective of this project is to design and simulate a digital traffic light controller for a T-intersection with one input and four outputs. Traffic flows from three directions, with two lights for direction A (forward and left turn) and one each for directions B and C. A pedestrian button halts traffic for safe crossing. The system operates in four modes, cycling through modes 1, 2, and 3 with specified intervals, and switches to mode 0 for pedestrians when needed. Utilizing binary signals and a 1 Hz clock, the controller aims to ensure precise, reliable, and safe traffic management at the intersection.

1.2 Relevant Theory

In order to successfully design the controller, is it required that a series design steps and choices are carefully studied and made. The design steps are committed as follows:

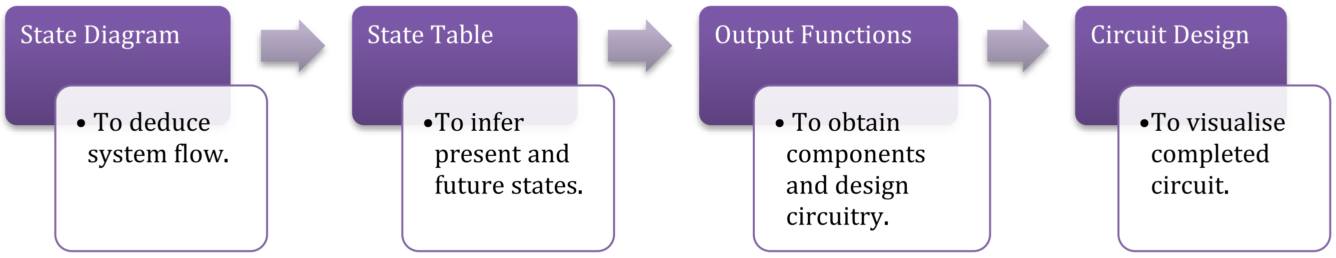

State Diagram
• To deduce system flow.
State Table
•To infer present and future states.
Output Functions
• To obtain components and design circuitry.
Circuit Design
•To visualise completed circuit.

Figure 1: Diagram representing design process steps.

The aforementioned steps are crucial and are meant to solely provide a 'backbone' to the design procedure. They were followed to generate the final product. Beginning with the *State Diagram*, the states are numbered, aligned and connected to illustrate the flow of the controller.

Figure 2 to the right depicts all four possible states. *Mode 0* shows the destination state before which the button signal is necessarily HIGH. Any state will revert back to *Mode 0* if the button signal is HIGH.

On the other hand, if the button signal is LOW, the states continuously cycle between *Mode 1*, *Mode 2* and *Mode 3*, each of which represents a designated value for each 'traffic light' and lasts for a distinct and set duration.
Since four states exist, only two flip-flops are required to represent them all.

The next step is to draw the *State Table*, available in **Figure 3**. Since our system can be described as a *Mealy Machine*, the input *I* is taken into account.

Q_1Q_0: represent the current state of the system.
$Q_1{}^+Q_0{}^+$: both represent the future state.
A_1, A_2, B and C: all represent 'traffic lights', and are output signals.
D_1D_0: are the values of the two D flip-flops which were chosen for the design of this system.

The convention of HIGH = GREEN ; LOW = RED is followed. There are three input signals available, hence $2^3 = 8$ input possibilities exist. We can also notice that the values of D_1D_0 will definitely be equal to $Q_1{}^+Q_0{}^+$.

Proceeding to the penultimate step of the theoretical stage, we obtain the Boolean functions for all output signals. **Figure 4** shows the exact process for each output signal and the deduced equations.

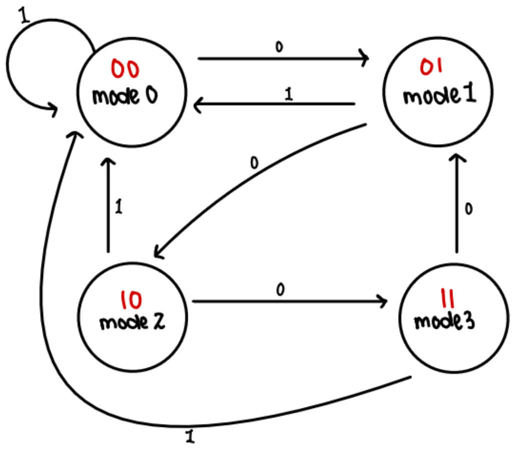
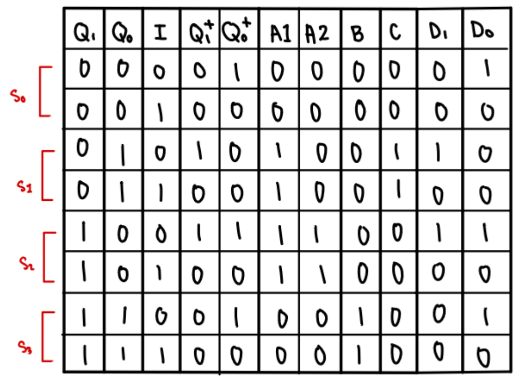


Figure 2: Shows the four possible states of the system and their respective connections.



	Q_1	Q_0	I	Q_1^+	Q_0^+	A1	A2	B	C	D_1	D_0
S_0	0	0	0	0	1	0	0	0	0	0	1
	0	0	1	0	0	0	0	0	0	0	0
S_1	0	1	0	1	0	1	0	0	1	1	0
	0	1	1	0	0	1	0	0	1	0	0
S_2	1	0	0	1	1	1	1	0	0	1	1
	1	0	1	0	0	1	1	0	0	0	0
S_3	1	1	0	0	1	0	0	1	0	0	1
	1	1	1	0	0	0	0	1	0	0	0

Figure 3: Represents the State Table, and the chosen D-FF values for each state.

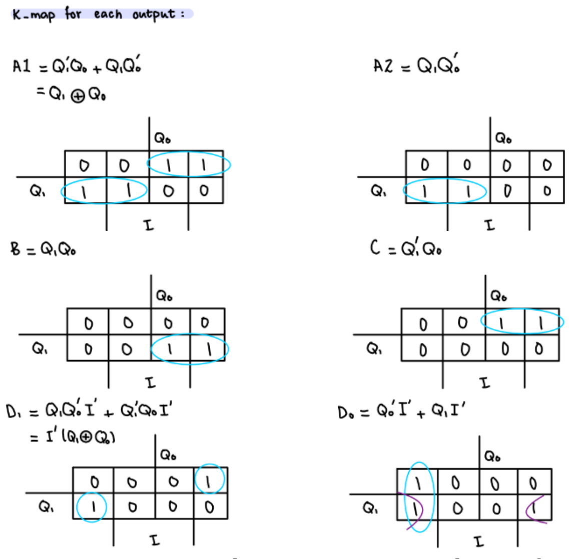


Figure 4: Represents the respective Karnaugh maps for each output signal in terms of the three input signals.

The ultimate stage of the *Relevant Theory* section is the circuit design. **Figure 5** to the right depicts the final image of the controller. Nodes on wires depict *branches* of said wire. With this step, the sequential circuit is complete. The final design required six AND gates, two XOR gates, one NOT gate and two D flip-flops.

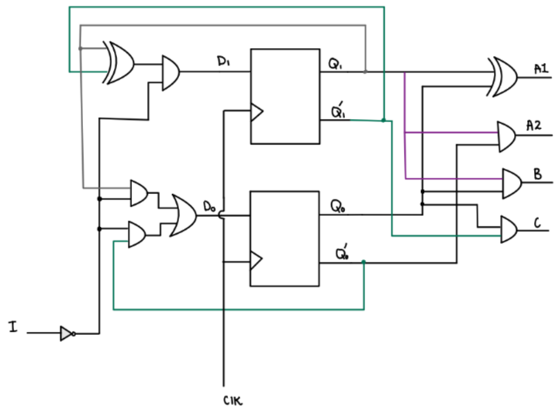


Figure 5: Depicts the final circuit representation of the traffic controller.

2 Code Approaches

Multiple approaches were considered for the Verilog code. There were attempts to design the circuit structurally using a counter for the clock of the traffic light controller and a regular clock for the counter.

```
module fivebitcount(I, OUT, CLK, reset);
  input I, CLK, reset;
  output [4:0] OUT;
  TFF X1 (I, CLK, reset, OUT[0]);
  TFF X2 (I, OUT[0], reset, OUT[1]);
  TFF X3 (I, OUT[1], reset, OUT[2]);
  TFF X4 (I, OUT[2], reset, OUT[3]);
  TFF X5 (I, OUT[3], reset, OUT[4]);
endmodule
module fivebitcount_tb;
  reg I_TB;
  reg CLK_TB = 0;
  reg reset_TB;
  wire [4:0] OUT_TB;
  fivebitcount uut (I_TB,OUT_TB,CLK_TB,reset_TB);
         always #1 CLK_TB = ~CLK_TB;
         initial begin
         $display("Time\tI\tOUT");
         I_TB = 1;
         reset_TB = 1;
         #10;
         reset_TB = 0;
         repeat(60) begin
         #2 $display("%0t\t%b\t%b", $time, I_TB, OUT_TB);
         end
    $finish;
  end
endmodule

module TFF(T, clk0, reset, Q);
  input T, clk0, reset;
  output reg Q;
  always @ (posedge clk0 or reset) begin
    if (reset) begin
      Q <= 0;
    end

    else if (T) begin
      Q <= ~Q;
    end

    else begin
      Q <= Q;
    end
  end
endmodule
module customDFF(D,clk,Q,notQ);
input clk, D;
output reg Q;
output reg notQ;

always @ (posedge clk)
begin
         Q<=D;
         notQ<=~D;
end
endmodule
```

```
module traffic(I,CLK,A1,A2,B,C);

         input I,CLK;

         output A1,A2,B,C;
         wire W1,W2,W3,W4,W5,W6,W7,W8,W9,W10;

         assign CLK = and()

         not(W1,I);
         and(W2,W6,W1);
         or(W10,W2,W3);
         customDFF x1(W10,CLK,W7,W6);
         and(W3,W1,W9);
         and(W4,W5,W1);
         xor(W5,W9,W8);
         customDFF x2(W4,CLK,W9,W8);
         xor(A1,W9,W7);
         and(A2,W9,W6);
         and(B,W9,W7);
         and(C,W7,W8);

endmodule

module traffic_tb;
  reg I_TB;
  reg CLK_TB = 0;
  wire A1_TB, A2_TB, B_TB, C_TB;

  traffic uut (I_TB,CLK_TB,A1_TB,A2_TB,B_TB,C_TB);


  always #5 CLK_TB = ~CLK_TB;


  initial begin

    I_TB = 1;
    #10;
    I_TB=0;

    repeat (20) begin
      #10; // Wait for a few clock cycles
      $display("Time: %0t | I: %b | A1: %b | A2: %b | B: %b | C: %b", $time, I_TB, A1_TB, A2_TB, B_TB, C_TB);
    end

    $finish;
  end
endmodule
```

This approach included a module for a 5-bit counter built with a T flip-flop, a D flip-flop and the traffic light controller as well as its test bench. This attempt failed due to reasons beyond understanding, possibly mainly due to being unable to rest the values of the flip flops before starting. Either way, this approach was way too complicated and tedious to improve upon, diagnose or trace.

After the failure of preliminary structural approaches, the approach was shifted to a mainly behavioural approach. The final product appears as such:

```
module traffic_lights_controller(clk,reset,button,A1, A2, B, C);
         input clk,reset,button;
         output reg A1, A2, B, C;
         reg [1:0] mode;
         reg [4:0] monitor;
parameter MODE0 = 2'b00, MODE1 = 2'b01, MODE2 = 2'b10, MODE3 =
2'b11, DURATION_MODE1 = 15, DURATION_MODE2 = 5, DURATION_MODE3 =
10;
    always @(posedge clk or reset) begin
        if (reset) begin
            mode <= MODE1;
            monitor <= DURATION_MODE1;
        end
                 else if (button && mode != MODE0) begin
            if (monitor == 0) begin
                mode <= MODE0;
                monitor <= DURATION_MODE1;
            end
                          else begin
                monitor <= monitor - 1;
            end

        end
                 else begin
            if (monitor == 0) begin
                case (mode)
                    MODE1: begin
                        mode <= MODE2;
                        monitor <= DURATION_MODE2;
                    end

                    MODE2: begin
                        mode <= MODE3;
                        monitor <= DURATION_MODE3;
                    end

                    MODE3: begin
                        mode <= MODE1;
                        monitor <= DURATION_MODE1;
                    end

                    MODE0: begin
                        mode <= MODE1;
                        monitor <= DURATION_MODE1;
                    end

                endcase
            end
                          else begin
                monitor <= monitor - 1;
            end
        end
    end

    always @(mode) begin
        case (mode)

            MODE0: begin
                A1 = 0;A2 = 0; B = 0; C = 0;
            end
            MODE1: begin
                A1 = 1; A2 = 0; B = 0; C = 1;
            end
            MODE2: begin
                A1 = 1;A2 = 1; B = 0; C = 0;
            end
            MODE3: begin
                A1 = 0; A2 = 0; B = 1; C = 0;
            end

        endcase
    end
endmodule
```

```
module traffic_lights_controller_tb;

    reg clk,reset,button;
    wire A1,A2,B,C;

    traffic_lights_controller TEST(clk,reset,button,A1,A2,B,C);

    always #1 clk = ~clk;

    initial begin

        clk = 0;
        reset = 1;
        button = 0;

        #2 reset = 0;
        #60 button = 1;
        #10 button = 0;
        #200;

        $finish;
    end

    initial begin
        $monitor("Time: %t | Mode: %d | A1: %b | A2: %b | B: %b |
C: %b",$time, TEST.mode, A1, A2, B, C);
    end
endmodule

/*
          The test bench tests for the use of the button and also
without,
          hence the only one test bench and not 2.
*/
```

This approach was simpler, easier to trace and less demanding to maintain. It also does not explicitly require flip-flops or a counter. It also required only one test bench to test for both using the button and without.

In addition, this approach disregarded the use of multiple clock sources, unlike the previous approach, and is successful in transitioning between multiple states smoothly and accurately.

3 RESULTS AND DISCUSSIONS

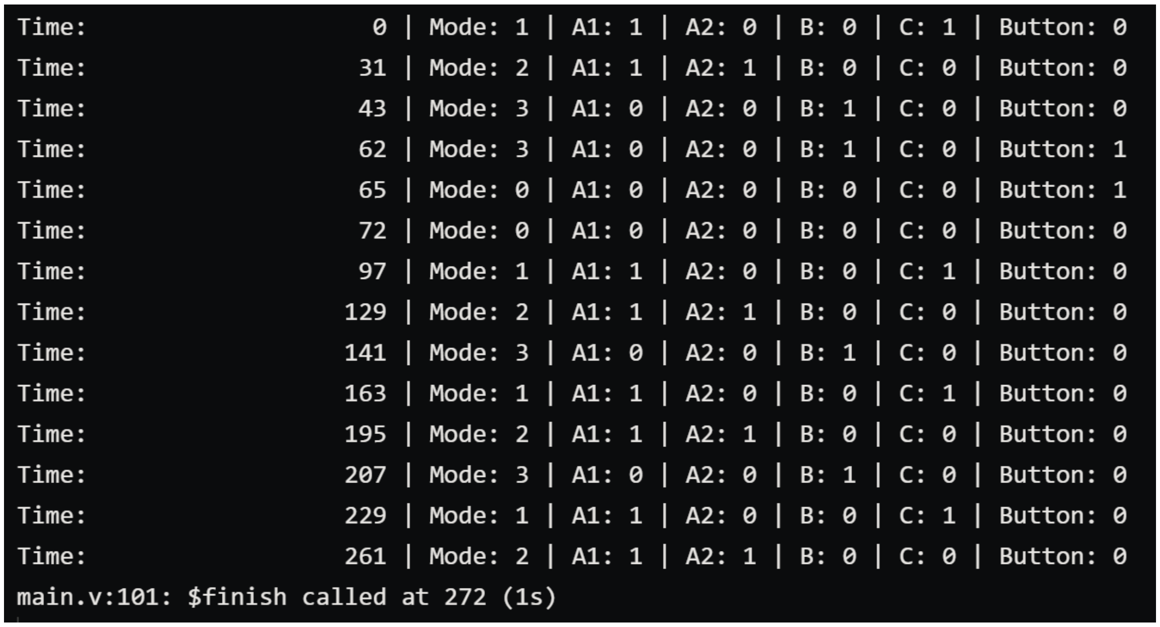

```
Time:                    0 | Mode: 1 | A1: 1 | A2: 0 | B: 0 | C: 1 | Button: 0
Time:                   31 | Mode: 2 | A1: 1 | A2: 1 | B: 0 | C: 0 | Button: 0
Time:                   43 | Mode: 3 | A1: 0 | A2: 0 | B: 1 | C: 0 | Button: 0
Time:                   62 | Mode: 3 | A1: 0 | A2: 0 | B: 1 | C: 0 | Button: 1
Time:                   65 | Mode: 0 | A1: 0 | A2: 0 | B: 0 | C: 0 | Button: 1
Time:                   72 | Mode: 0 | A1: 0 | A2: 0 | B: 0 | C: 0 | Button: 0
Time:                   97 | Mode: 1 | A1: 1 | A2: 0 | B: 0 | C: 1 | Button: 0
Time:                  129 | Mode: 2 | A1: 1 | A2: 1 | B: 0 | C: 0 | Button: 0
Time:                  141 | Mode: 3 | A1: 0 | A2: 0 | B: 1 | C: 0 | Button: 0
Time:                  163 | Mode: 1 | A1: 1 | A2: 0 | B: 0 | C: 1 | Button: 0
Time:                  195 | Mode: 2 | A1: 1 | A2: 1 | B: 0 | C: 0 | Button: 0
Time:                  207 | Mode: 3 | A1: 0 | A2: 0 | B: 1 | C: 0 | Button: 0
Time:                  229 | Mode: 1 | A1: 1 | A2: 0 | B: 0 | C: 1 | Button: 0
Time:                  261 | Mode: 2 | A1: 1 | A2: 1 | B: 0 | C: 0 | Button: 0
main.v:101: $finish called at 272 (1s)
```

Figure 6: Shows the results for the test bench of the final coded module.

Figure 6 above shows the resulting final output of the module and its accompanying test bench. It is clearly displayed that all modes are successfully transitioned to and from. At unit time sixty, the button is pressed for a certain amount of time and then released, simulating a button press. The current state is terminated at the next clock cycle of the button being pressed and *Mode 0* is geared into effect.

Since the clock frequency cannot be set to exactly 1 Hz, it is instead set to 2 Hz, demanding that the time duration for each mode is halved in the code to correct durations in real-time simulation.

The simulation runs for two hundred sixty one time units in total.

4 CONCLUSIONS

In conclusion, this project successfully designed and simulated a traffic light controller for a T-intersection, addressing the critical aspects of precision and cost-efficiency. The implementation was meticulously executed, starting from a state diagram and progressing through the creation of a state table, Boolean functions, and finally the circuit design. This thorough process ensured accurate and reliable signal transitions across all modes. The initial structural coding attempts were replaced by a more effective behavioral approach, simplifying maintenance and enhancing traceability. The Verilog code was tested rigorously, demonstrating smooth transitions and correct responses to the pedestrian button input.

5 References

1. Bhattacharyya, S., Chattopadhyay, S., Banerjee, S., & Chakrabarti, P. P. (2010). A novel synthesis technique for VHDL and Verilog designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 29*(6), 952-965. doi:10.1109/TCAD.2010.2046865

2. Liu, J., Li, S., & Dai, X. (2011). A comparative study of VHDL and Verilog HLS tools for DSP system design. *Proceedings of the 2011 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2753-2756. doi:10.1109/ISCAS.2011.5938140

3. Kumar, M., Patra, M., & Mahapatra, K. K. (2014). High-level synthesis from Verilog: An FPGA synthesis perspective. *Proceedings of the 2014 International Conference on Advances in Electronics Computers and Communications (ICAECC)*, 1-6. doi:10.1109/ICAECC.2014.7002442

4. Fummi, F., Martini, F., Poncino, M., & Sciuto, D. (1998). Native simulation of compiled Verilog HDL. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 17*(10), 1040-1052. doi:10.1109/43.725412

5. Keating, M., & Bricaud, P. (1999). Reuse methodology manual for system-on-a-chip designs: with examples in Verilog and VHDL. *Kluwer Academic Publishers*. ISBN: 978-0792384600