Theoretical Part

Introduction

In this lab we explored the use of 8 steps needed for the FSM method to develop an FSM controller which we then needed to use to implement a traffic light system for two intersecting streets. Components such as counters, comparator, and multiplexers need to be used to complete the Traffic Light Top Level Entity. As the problem specifications state, there is a Main Street and a Side Street, each with their own traffic light (MSTL,SSTL respectively). Since the Main Street contains more traffic, it has priority, only if it's timer (MSC) runs out and there's a car waiting at the Side Street light, will the lights change. The sensor is implemented by using a push button on the Altera Board. The user-set counters, MSC and SSC, are defined by using 4 switches each. The traffic lights, MSTL and SSTL, are represented by 3 LEDs each.

Discussion of problem

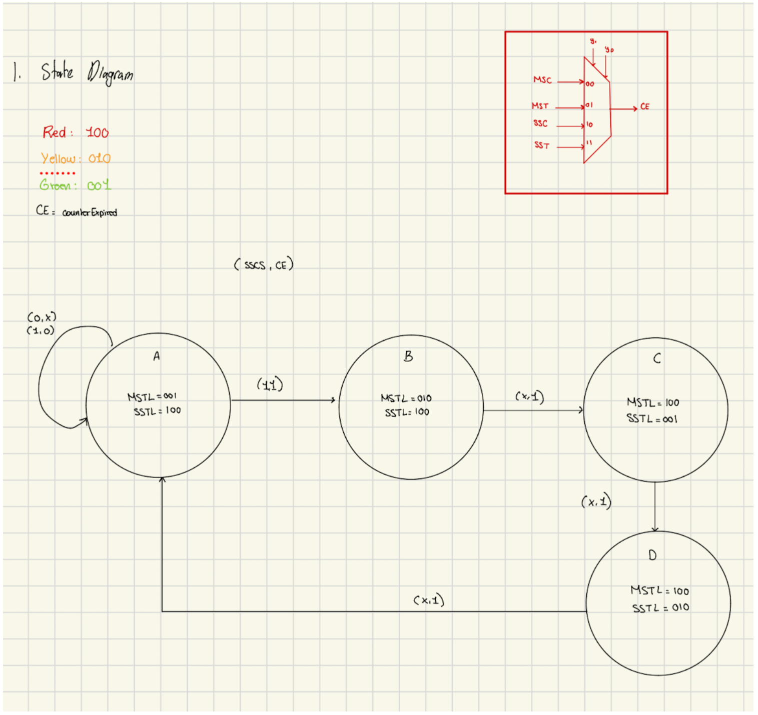
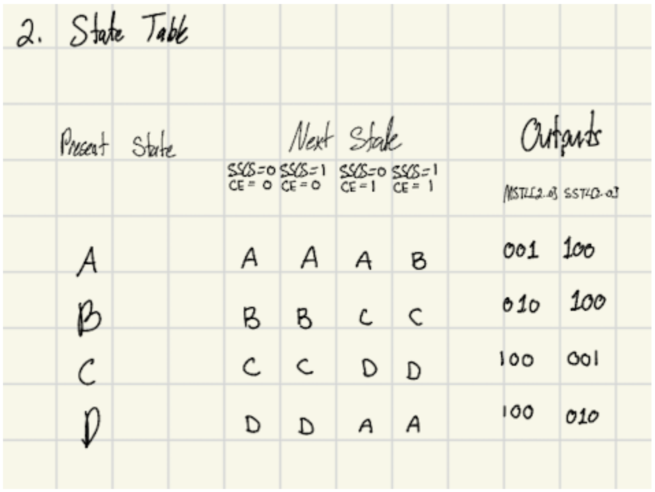


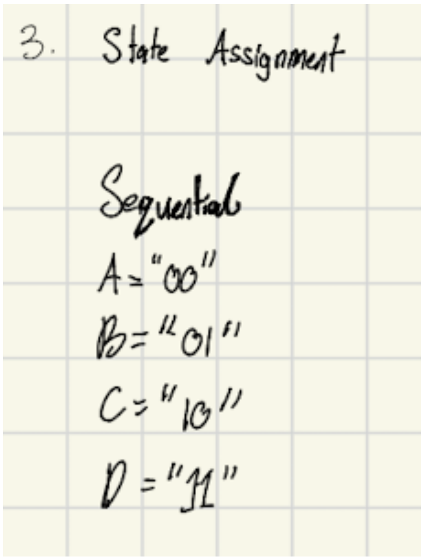
Figure 1: State diagram



2. State Table

Present State	Next State				Outputs	
	SSCS=0 CE = 0	SSCS=1 CE = 0	SSCS=0 CE = 1	SSCS=1 CE = 1	MSTL[2..0]	SSTL[2..0]
A	A	A	A	B	001	100
B	B	B	C	C	010	100
C	C	C	D	D	100	001
D	D	D	A	A	100	010

Figure 2: State table


3. State Assignment

Sequential

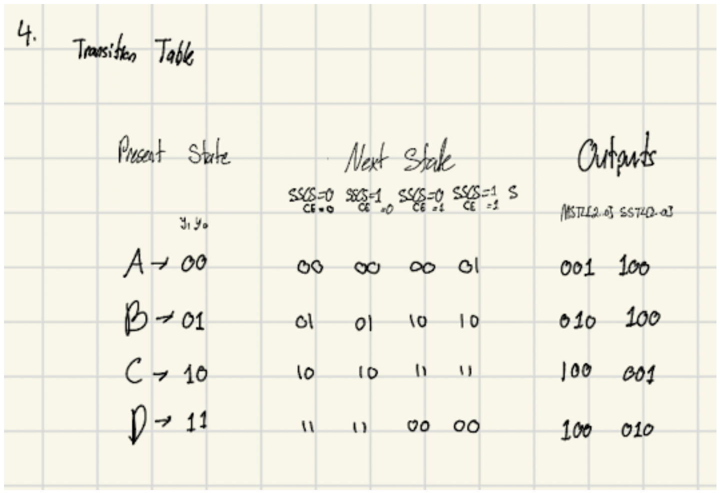
A = "00"

B = "01"

C = "10"

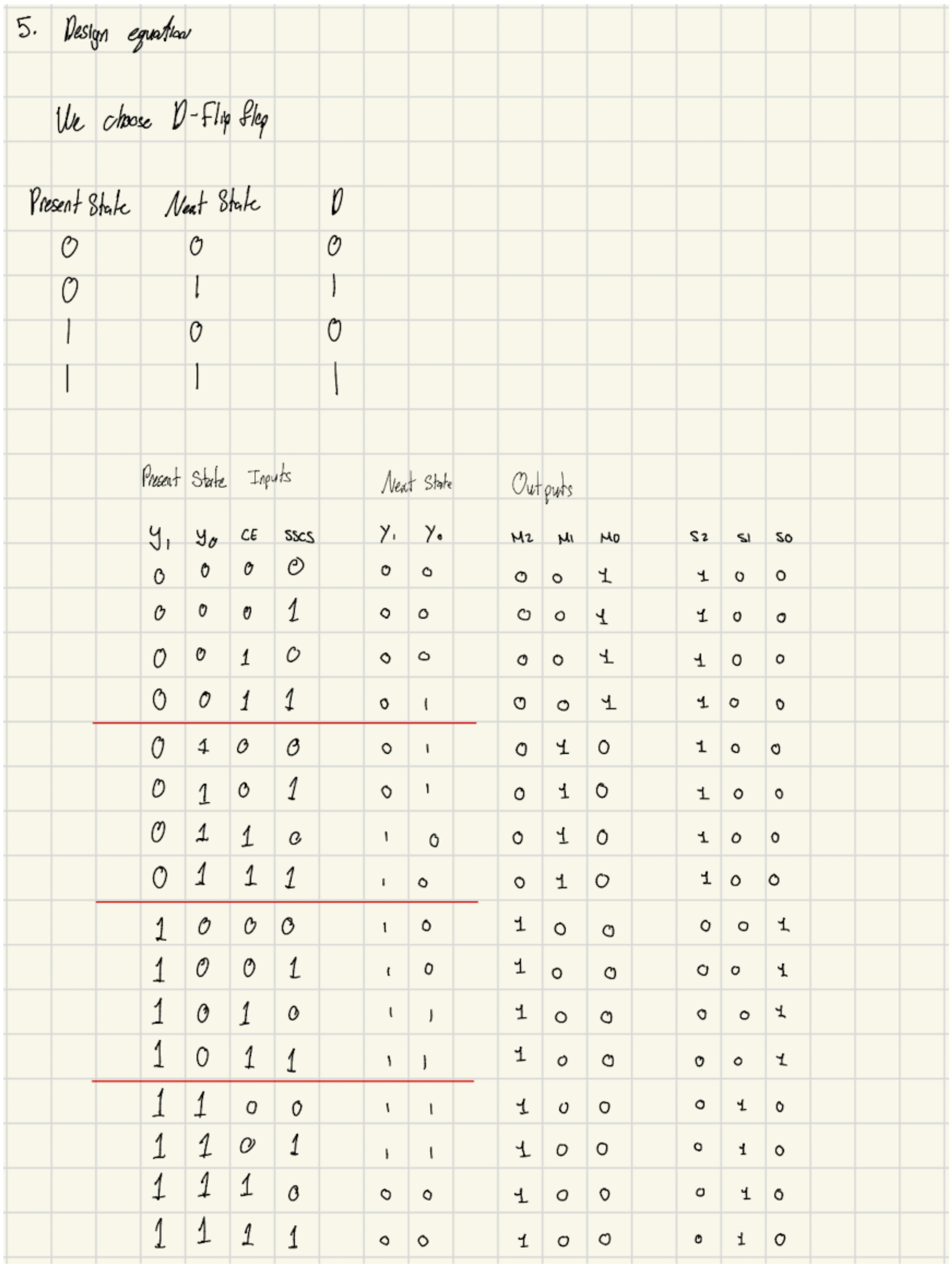
D = "11"

Figure 3: State assignment


4. Transition Table

Present State	Next State				Outputs	
$y_1 y_0$	SSCS=0 CE=0	SSCS=1 CE=0	SSCS=0 CE=1	SSCS=1 CE=1 S	MSTL[2..0]	SSTL[2..0]
A → 00	00	00	00	01	001	100
B → 01	01	01	10	10	010	100
C → 10	10	10	11	11	100	001
D → 11	11	11	00	00	100	010

Figure 4: Transition table



5. Design equation

We choose D-Flip flop

Present State	Next State	D
0	0	0
0	1	1
1	0	0
1	1	1

Present State		Inputs		Next State		Outputs					
y_1	y_0	CE	SSCS	Y_1	Y_0	M2	M1	M0	S2	S1	S0
0	0	0	0	0	0	0	0	1	1	0	0
0	0	0	1	0	0	0	0	1	1	0	0
0	0	1	0	0	0	0	0	1	1	0	0
0	0	1	1	0	1	0	0	1	1	0	0
0	1	0	0	0	1	0	1	0	1	0	0
0	1	0	1	0	1	0	1	0	1	0	0
0	1	1	0	1	0	0	1	0	1	0	0
0	1	1	1	1	0	0	1	0	1	0	0
1	0	0	0	1	0	1	0	0	0	0	1
1	0	0	1	1	0	1	0	0	0	0	1
1	0	1	0	1	1	1	0	0	0	0	1
1	0	1	1	1	1	1	0	0	0	0	1
1	1	0	0	1	1	1	0	0	0	1	0
1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	0	0	0	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	0

Figure 5: Design equations part I

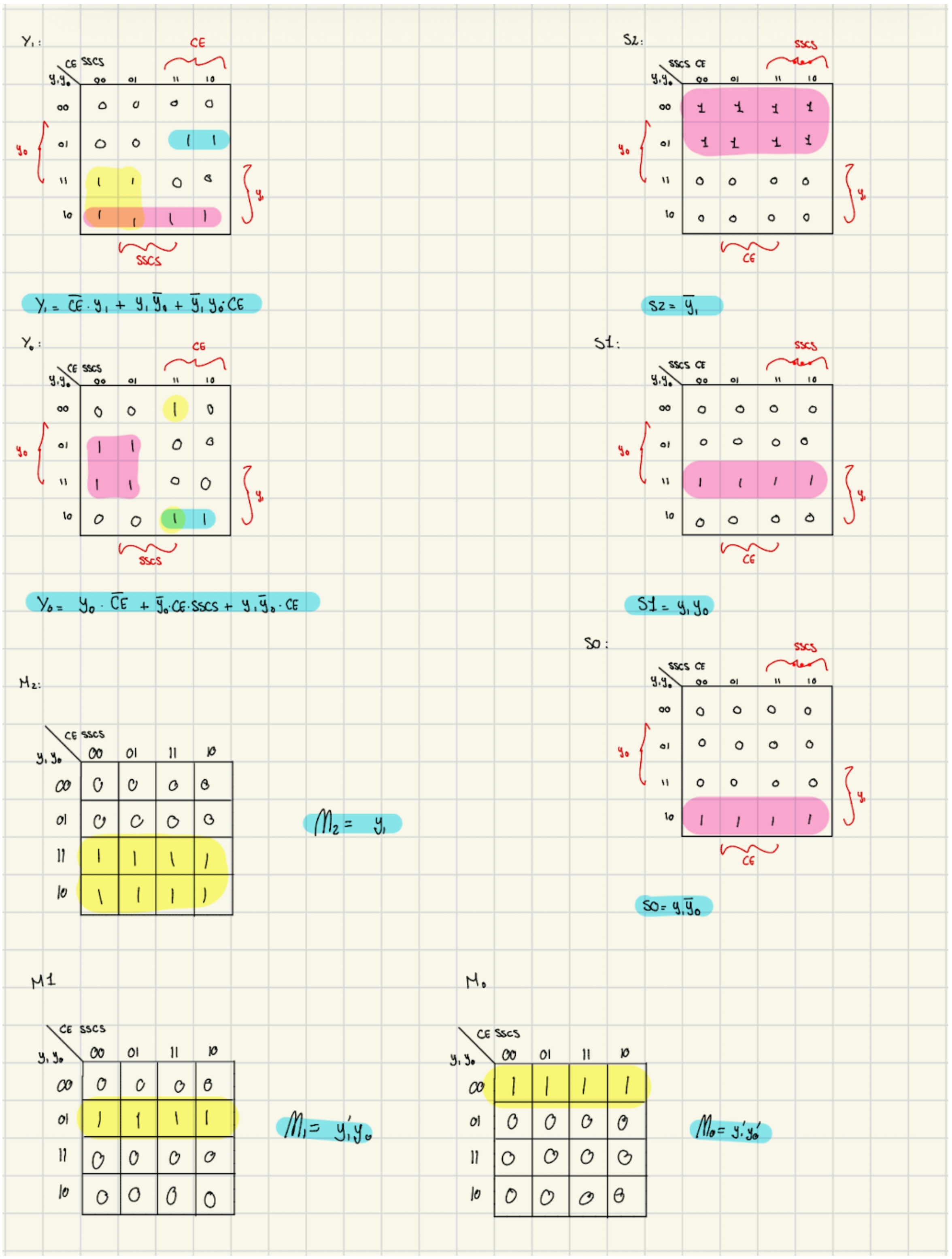


Figure 6: Design equations part II

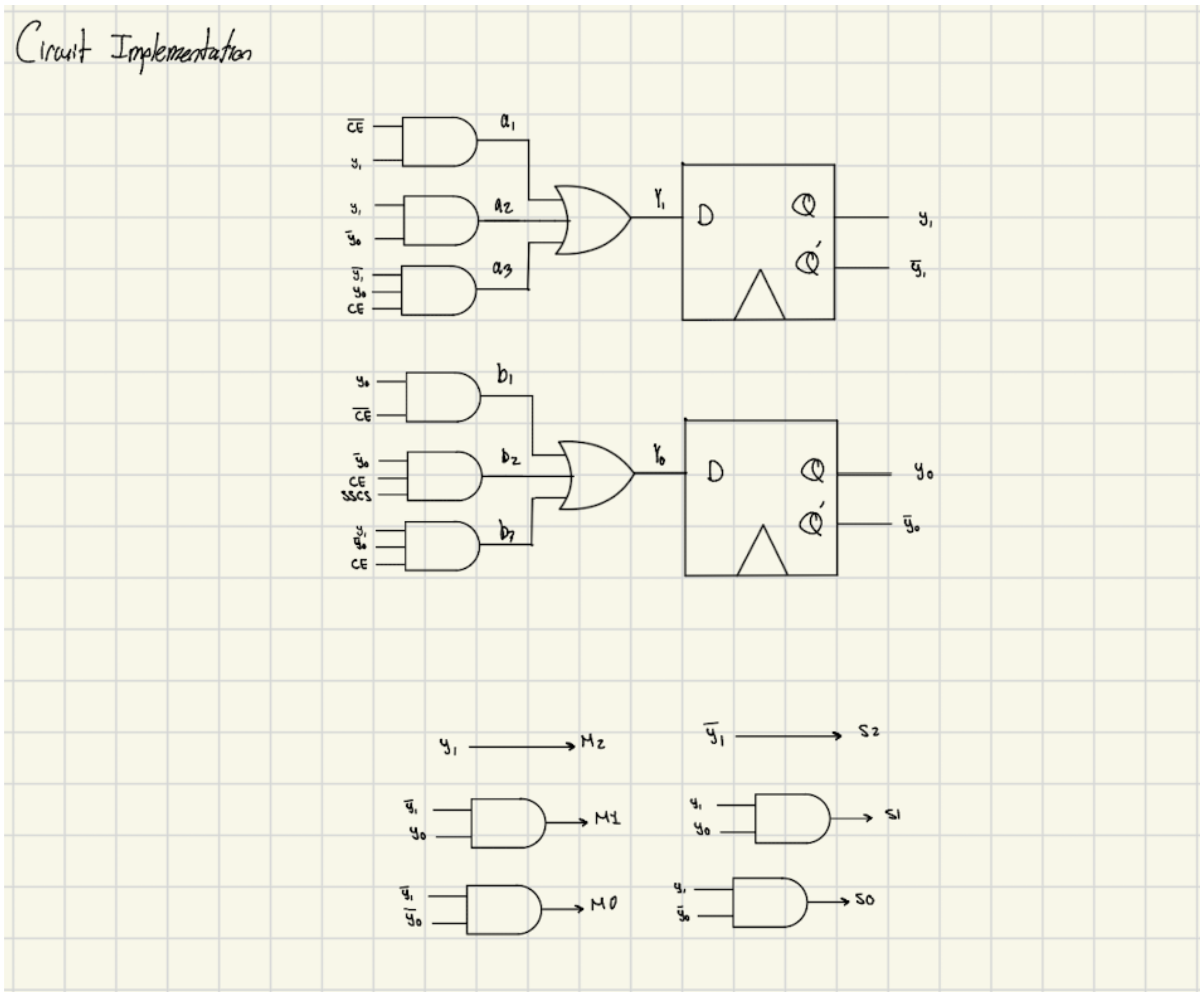


Figure 7: Circuit implementation

We designed a sequential circuit using the FSM approach. Our design follows a Moore-type FSM which means the outputs depend only on the current state rather than the inputs. We started by defining the inputs and outputs, then drew the state diagram to show how the system transitions between states based on signals like SSCS and CE. Next, we created the state table to list the present states, next states, and outputs. Each state was defined using a sequential assignment (A = 00, B = 01, C = 10, D = 11) for hardware implementation. Afterward, we built the transition table and simplified the logic using K-maps to find the minimal Boolean equations for the next-state and output logic. Finally, we used these equations to draw the circuit implementation with AND, OR, and NOT gates feeding D flip-flops. This circuit reproduces the FSM behaviour and transitions correctly through all our required states.

Discussion of algorithmic solution

Our team has been hired to design a traffic light controller. A traffic light controls the traffic at an intersection between a main street MS and a side street SS. The main street has higher priority than the side one and thus stays green until at least one car on the side street arrives at the intersection. A car sensor SSCS is present on the side street for such a purpose. The sensor outputs a '1' at the presence of a car on the side street and a '0' otherwise. Normal operation of the traffic light consists of the main street's traffic light (MSTL) being green and the side street's (SSTL) being red. The MSTL stays green for an in-system-programmable (ISP) period MSC, after which if there is a car on the SS, the MSTL turns yellow for a set period MT, then turns to red. The SSTL then turns green for a ISP period SSC, after which it automatically turns yellow (even if there are cars at the intersection) for a set period SST, then red. The MSTL goes back to green at that point. If after MST, there are no cars on the SS, then the MSTL stays green until a car arrives, at which time, the MSTL immediately cycles through its yellow then red changes. The input/output specifications of the traffic light controller are shown in the table below.

Port Type	*Name*	*Description*
Input	GClock	Global clock needed to synchronize the circuitry
Input	GReset	Global reset needed to bring the internals to known states
Input	MSC[3..0]	Main street counter 4-bit input
Input	SSC[3..0]	Side street counter 4-bit input
Input	SSCS	Side street's car sensor input
Output	MSTL[2..0]	Main street's traffic light output using one-hot encoding
Output	SSTL[2..0]	Side street's traffic light output using one-hot encoding

Figure 8: Input/Output table

Design Part

Discussion of used components

Traffic Light Controller: This is the top-level entity of the design. It connects all the main components, including the FSM, counters, and multiplexers, to form the complete traffic light controller. The system contains four counters, whose outputs are fed into the first 4-to-1 multiplexer. The FSM state bits (y1, y0) serve as the select lines for this MUX which allows the circuit to choose which counter is active depending on the current state. A second 4-to-1 multiplexer is used to select between the input signals SSC and MSC or the predefined constants MST and SST (which in our case are both set to 7 seconds). This selection also depends on the FSM state (y1, y0). The outputs of both multiplexers are sent to the comparator module, which produces a CE (Counter Expired) signal once the timer reaches its maximum value. As the counter starts from 0 and counts until the counter is equal to the max value, it goes through n+1 clock cycles. Since we only want n clock cycles, we subtracted 1 from the output of the second MUX before feeding it into the comparator (please see drawing below for better visualization). The CE signal is then fed back to the FSM which triggers a transition to the next state. The Traffic Light Controller therefore synchronizes all internal modules, which ensures correct timing and smooth transitions between light phases. Inside the traffic light controller, we also have a clock divider to slow down the internal clock on the Altera board, since if it's going too fast all red, yellow, and green lights will be turn on simultaneously to the naked eye. Finally, we also have a debouncer for the SSCS input as it will be a button. The output of the debouncer is then fed into the FSM controller to determine if we leave the initial state.

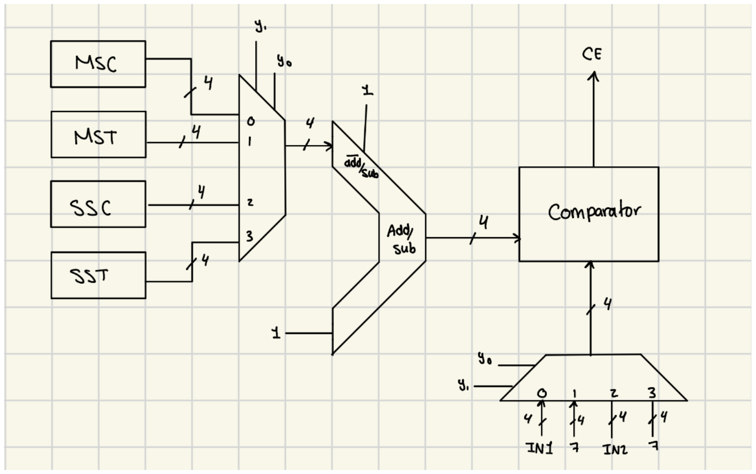
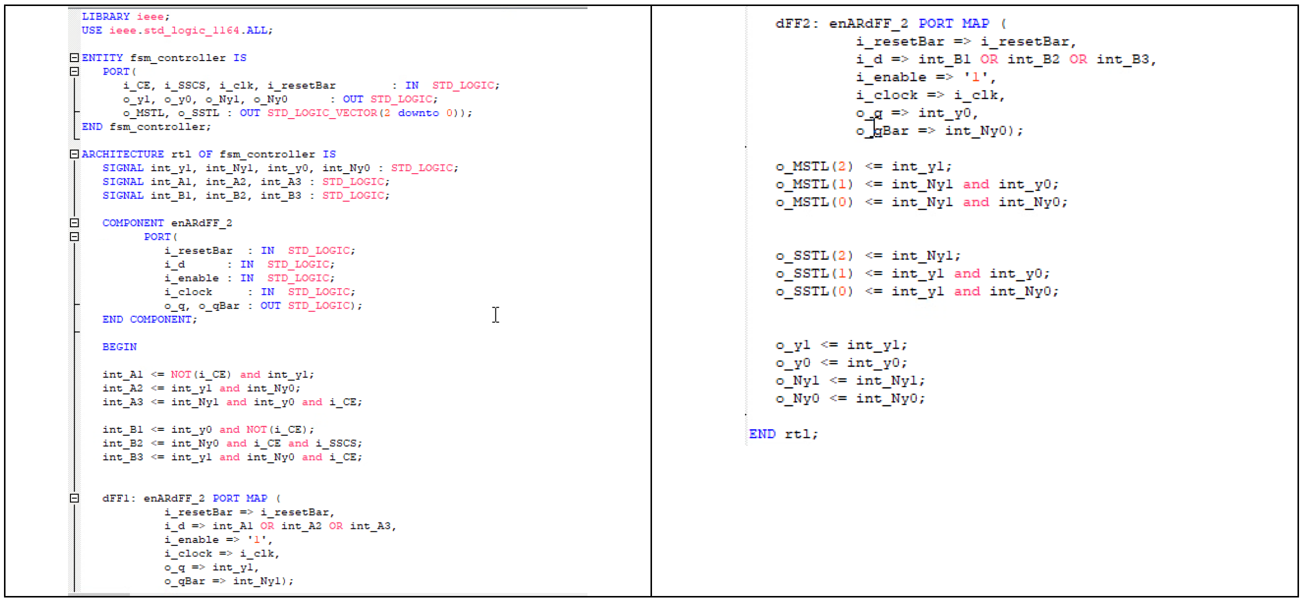


Figure 9: Counter Expired logic diagram

FSM: Our Finite State Machine (FSM) component implements the control logic of the traffic light system. It defines the different light states (such as red, yellow, and green) and manages the transitions between them based on the clock and enable signals. The FSM follows a Moore-type architecture which means the outputs depend only on the current state. It generates the control outputs (M2, M1, M0, S2, S1, S0) according to the active state which ensures that the lights change in a stable and synchronized sequence.

Discussion of actual solution (VHDL code)

FSM controller



```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY fsm_controller IS
   PORT(
      i_CE, i_SSCS, i_clk, i_resetBar        : IN  STD_LOGIC;
      o_y1, o_y0, o_Ny1, o_Ny0      : OUT STD_LOGIC;
      o_MSTL, o_SSTL : OUT STD_LOGIC_VECTOR(2 downto 0));
END fsm_controller;

ARCHITECTURE rtl OF fsm_controller IS
   SIGNAL int_y1, int_Ny1, int_y0, int_Ny0 : STD_LOGIC;
   SIGNAL int_A1, int_A2, int_A3 : STD_LOGIC;
   SIGNAL int_B1, int_B2, int_B3 : STD_LOGIC;

   COMPONENT enARdFF_2
         PORT(
            i_resetBar  : IN  STD_LOGIC;
            i_d       : IN  STD_LOGIC;
            i_enable : IN  STD_LOGIC;
            i_clock     : IN  STD_LOGIC;
            o_q, o_qBar : OUT STD_LOGIC);
   END COMPONENT;

   BEGIN

   int_A1 <= NOT(i_CE) and int_y1;
   int_A2 <= int_y1 and int_Ny0;
   int_A3 <= int_Ny1 and int_y0 and i_CE;

   int_B1 <= int_y0 and NOT(i_CE);
   int_B2 <= int_Ny0 and i_CE and i_SSCS;
   int_B3 <= int_y1 and int_Ny0 and i_CE;


   dFF1: enARdFF_2 PORT MAP (
            i_resetBar => i_resetBar,
            i_d => int_A1 OR int_A2 OR int_A3,
            i_enable => '1',
            i_clock => i_clk,
            o_q => int_y1,
            o_qBar => int_Ny1);

   dFF2: enARdFF_2 PORT MAP (
            i_resetBar => i_resetBar,
            i_d => int_B1 OR int_B2 OR int_B3,
            i_enable => '1',
            i_clock => i_clk,
            o_q => int_y0,
            o_qBar => int_Ny0);

   o_MSTL(2)  <= int_y1;
   o_MSTL(1)  <= int_Ny1 and int_y0;
   o_MSTL(0)  <= int_Ny1 and int_Ny0;


   o_SSTL(2)  <= int_Ny1;
   o_SSTL(1)  <= int_y1 and int_y0;
   o_SSTL(0)  <= int_y1 and int_Ny0;


   o_y1  <= int_y1;
   o_y0  <= int_y0;
   o_Ny1  <= int_Ny1;
   o_Ny0  <= int_Ny0;

END rtl;
```

Figure 10: FSM Controller VHDL

<u>Traffic Light Controller</u>

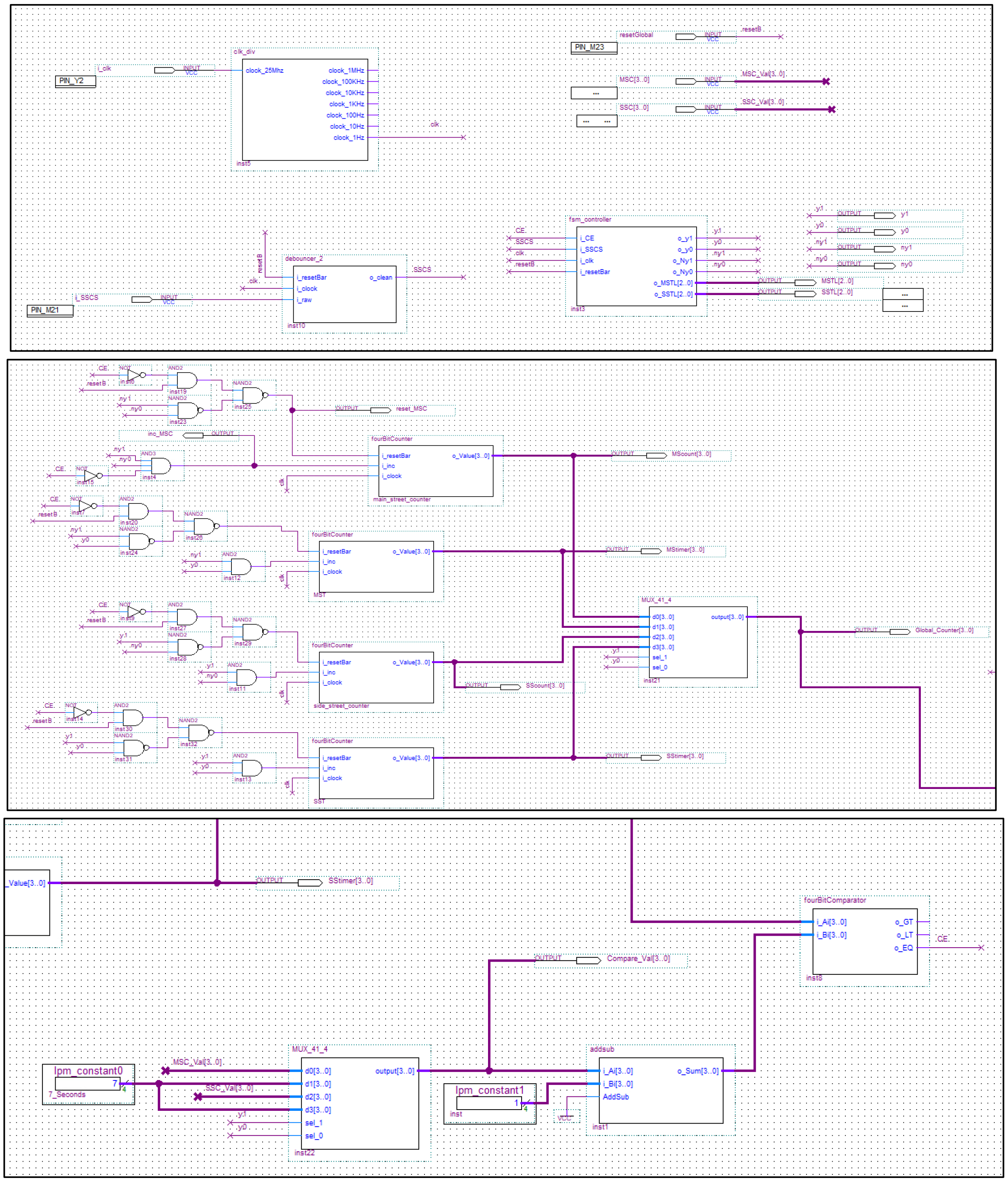


Figure 11: Traffic light controller BDF

Discussion of tools (Bonus)

The lab required the use of 2 tools in specific. The Altera FPGA board provided during the lab to test/simulate the digital circuits designed, and Intel Quartus II to write the VHDL code and design the digital circuit.

Discussion of challenging problems (Bonus)

1. Resetting the counter once it reached the time: The traffic light would behave correctly at the first run (it would go to state B once the CE signal turned on) but since the counter was not resetting to zero after the first iteration, it would take much longer to enter state B as it would go past the time it was set to. Deriving the equation for the counter reset input was challenging as it was difficult to think of the logic in active-low
   Solution: we derived the equation that would properly reset the counter through trial and error.
2. All three lights turned on for both traffic lights when testing it on the board: not the expected result
   Solution: slowed down the clock using the clock divider (1Hz) as we realised it just meant the states were all happening very quickly and simply appeared to be the wrong output.
3. The comparator was outputting the CE after n+1 clock cycles. In our implementation we output CE to be 1 when the counter is **equal** to the value set by the user (n) it's being fed from the multiplexer. Therefore, it would have to go through 0-n until it's CE=1, meaning n+1 clock cycles.
   Solution: We simply added an Adder/Subtractor component that is always in subtraction mode (AddBar/Sub = 1). We fed the user-set value (n) into the first operand and '1' into the second giving us n-1 which we then inputted into the comparator. This way the comparator will signal CE=1 when it's gone through exactly n clock cycles.

Real Implementation

Shown simulation/synthesis results

FSM controller:

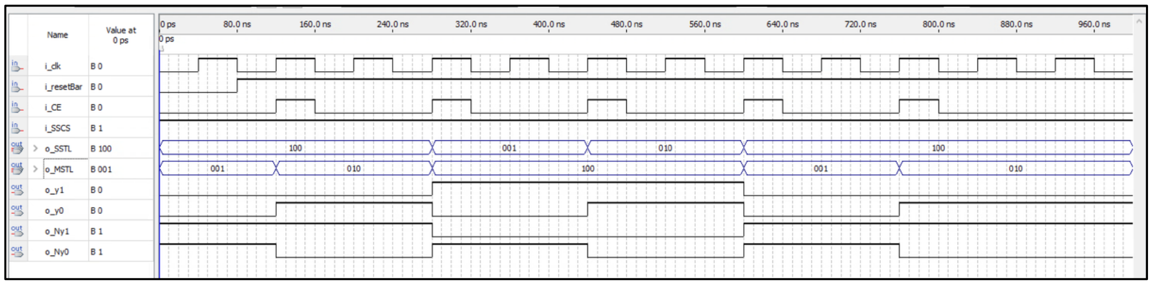
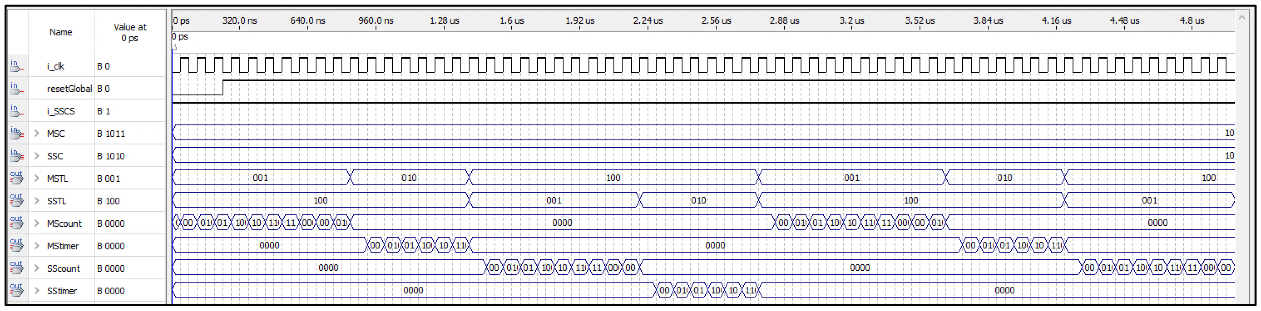


Figure 12: Waveform for FSM controller

Traffic light top level entity:

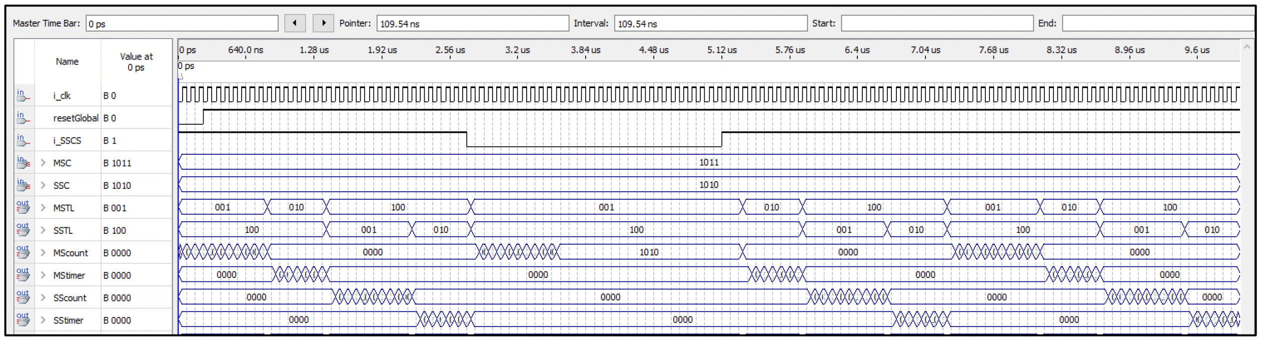
Sensor always ON



Name	Value at 0 ps
i_clk	B 0
resetGlobal	B 0
i_SSCS	B 1
MSC	B 1011
SSC	B 1010
MSTL	B 001
SSTL	B 100
MScount	B 0000
MStimer	B 0000
SScount	B 0000
SStimer	B 0000

Figure 13: Waveform for Traffic Light Controller when SCSS =1

Sensor turns OFF for a bit



Master Time Bar: 0 ps Pointer: 109.54 ns Interval: 109.54 ns Start: End:

Name	Value at 0 ps
i_clk	B 0
resetGlobal	B 0
i_SSCS	B 1
MSC	B 1011
SSC	B 1010
MSTL	B 001
SSTL	B 100
MScount	B 0000
MStimer	B 0000
SScount	B 0000
SStimer	B 0000

Figure 14: Waveform for Traffic Light Controller when SCSS = 0 for a period

Verification (actual simulation + live demonstration)

The demo for this lab went perfectly. The traffic light system behaved exactly as it should. Using the dip switches we set the main street counter (MSC) to 15 and the side street counter (SSC) to 14. Both the sensor input and the global reset were set to HIGH (1) as the buttons are active low. We entered the first state (MSTL: 001 SSTL: 100) for 15 seconds, then since SSCS =1, we moved on to the next state (MSTL: 010 SSTL: 100) for 7 seconds (our pre-defined time inside the code), then up next we had (MSTL: 100 SSTL: 001) for 14 seconds, and finally (MSTL:100 SSTL: 010) for also 7 seconds. After that, we re-enter the loop until we held down the SSCS button (SSCS = 0) which causes the system to stay at the initial state since there's no one waiting at the side street.

Discussion/Conclusion

In conclusion, we designed and implemented a structural Moore FSM to control traffic lights at a two-way intersection. Our system prioritized the main street and used a sensor to detect cars on the side street which adjusts the light sequence accordingly. We integrated programmable timers and handled multi-bit outputs for both traffic signals. The final design functioned as expected, and the lab helped solidify our understanding of FSMs and structural VHDL in this course.