

***SCHOOL OF ELECTRICAL***

***ENGINEERING and TRADES***

***Traffic Lights Project***

***505-704 Embedded Systems 1***

***2011***

***Final REPORT***

***BY***

***Student Name: Pavel Boryseiko***

***Date: 6 June 2011***

# Abstract:

Traffic Lights. An often overlooked system that maintains order on the roads of the world. Since traffic lights are a integral part of everyday life, the composition of how they would work is an integral part of this report. This document will describe the process of modelling a traffic light system to the best ability of the PIC 18F4520 microcontroller and readily available chips. This report will dwell deep into the process of designing the circuit necessary to simulate the system, but instead of using an inductive loop metal detector to detect traffic, this project will use push buttons to simulate traffic. Not only that, but this project will go into detail as to how to create a state machine to control the cycle of the lights with pedestrian crossings and right turns. By the end of this report, there will be no doubt as to how a traffic light system could possibly be simulated using a PIC 18F4520 microcontroller, a Real Time Clock chip, an LCD, some LEDs, some push buttons and a few lines of C code.

Table of Contents

[Abstract: i](#_Toc295320061)

[1. Introduction 1](#_Toc295320062)

[1.1. Specifications 1](#_Toc295320063)

[1.2. Purpose 1](#_Toc295320064)

[2. Design Process 2](#_Toc295320065)

[2.1. Hardware Design 2](#_Toc295320066)

[2.2. Software Design 2](#_Toc295320067)

[2.2.1. State Machine 3](#_Toc295320068)

[2.2.2. LCD Display with RTC 4](#_Toc295320069)

[2.2.3. USART with RTC 5](#_Toc295320070)

[3. Conclusion 6](#_Toc295320071)

[5. Appendix A](#_Toc295320072)

# Introduction

## Specifications

The specifications dictated that a traffic light control system was to be designed using the PIC 18F4520 microcontroller. The minimum requirement was to control a cross-roads intersection with a red/orange/green light set for each through road., and to achieve extra marks, right-turn arrows and pedestrian crossing lights were to be added.

To allow for expansion, driving the lights through a binary decoder was recommened as that allowed 4 output pins to become 16.

The design should be based on a state machine and should should “rest” on the main road and only change state when a vehicle arrives on the side road. Vehicle arrivals should be simulated with a push-button input. Using a binary encoder to monitor 16 inputs on 4 port pins is recommended to minimize the number of ports used on the PIC 18F4520.

The length of time the lights will stay green for the side road varies between 2 and 30 seconds depending on the traffic volume, with the timing information is to be sourced from an external Real Time Clock chip.

After each change of state the system is to generate a “trace” message which it transmits out an RS232 port. This message could either be printed to a line printer or captured by a terminal program.

An LCD is to be provided which will display the current time and state.

## Purpose

This document is the report which specifies the steps taken in the design of the software and hardware for the Traffic Light Control project. This report will dwell deep into the design process of the hardware using PROTEUS, explaining every significant hardware part and its purpose in the overall hardware design of the project. Also included will be an in depth documentation of the programming code using MPLAB, explaining all states, functions and how each hardware component was incorporated.

# Design Process

The first step in the design process was to create a simple drawing of what the cross intersection would look like and how it would operate. The drawing is shown in Figure 1. Since the specifications did not explicitly identify which lanes should operate in what order, the order was established by drafting an initial flowchart to be used during the programming stages. This flowchart was based on a State Machine as per the specification and can be seen in Figure 2.

## Hardware Design

During the design process of the circuit with PROTEUS, most of the hardware components were specified in the specs. The few that were not, like what type of RTC and LCD had to be chosen. Since the specification asked for the use of a Binary Encoder/Decoder to minimize the number of ports used on the PIC 18F4520, a compatible model had to be chosen as well. The DS1307 chip was chosen as the real time clock and the 74HC45154 as the Encoder and the74HC147 as the decoder. The LCD was the LM016L. These were chosen as they had simulation models in the PROTEUS software which made them ideal for this project.

As the hardware design was in progress, it was found that to make the simulation of the whole project easier to interpret, the circuit needed to be split up into 2 seperate sheets. These sheets can be seen in Figure 3 and Figure 4.

Figure 4 is the TOP LEVEL design. This shows a simulated x-intersection, with the LEDs serving as the red/orange/green lights. These will be solid green to demonstrate the green light, flash to demonstrate the yellow and turn off to demonstrate the red. This was done to reduce the number of I/O used. The buttons will act as traffic going across/stopping on the lane. Also shown is the LCD screen which will display the current time as well as the present state of the system. The terminal is also provided which will show the “trace” message. This will display the state that the system has changed to as well as the time as which is changed.

## Software Design

Once the hardware was all wired up in PROTEUS, the programming of the PIC 18F4520 was started using the MPLAB IDE. To make the programming division of the project easier, the whole project was broken down into 3 parts:

* State Machine
* LCD display with RTC
* USART with RTC

These were initially programmed seperately and tested before being merged into the whole Traffic Light Project.

Since it is not practical to copy and paste all 800 lines of programming code into this document, snippets of important code will be taken and described in the following sections.

### State Machine

That main part of the program is the State Machine. This controls the process by which the lights operate and in what order. A single state can be seen in Figure 5. There are a total of 6 states as apposed to the initial 12 discussed in the flow chart. This difference will be explained further in the conclusions, but is mainly due to the limitations of the PROTEUS VSD and wiring. The states are as follows:

1. Main road both lanes going straight
2. Main road both lanes turning
3. Side road both lanes going straight
4. Side road both lanes turning
5. Main road both lanes going straight with pedestrains crossing side road
6. Side road both lanes going straight with pedestrians crossing main road

A switch statement simulates the state machine as each case of the switch statement will demonstrate each of the 6 main states. The state shown in Figure 5 is the most complex. This is the default/initial state the traffic light system is in. This handles every other possible state. The way the state machine moves onto the next state is by monitoring inputs from the buttons wired to the encoder.

if(checkPortB==MAIN\_STRAIGHT\_B1 || checkPortB==MAIN\_STRAIGHT\_B2)

{

if(minTime < maxTime-2)

{

minTime = minTime+2;

}

}

if(checkPortB==MAIN\_TURN\_B1 || checkPortB==MAIN\_TURN\_B2)

{

newState = 1;

}

An example of the IF statement that handles a button action is shown above. The top IF statement checks whether the current states’ button was pressed simulating traffic. The bottom IF statement parses whether any of the other button were triggered. This simulates a car stopping in a particular lane.

In each state, these IF statements all occur while the a timer has not reached its minimum time. This is shown below:

do

{

…

} while (currentTime < minTime);

Since the default state doesn’t automatically change to a different state after the timer has reached the minimum time, the second DO WHILE loop was introduced to wait for an input from the buttons after the minimum time has elapsed. This is seen below:

do

{

…

}while(newState==0);

newState corresponds to any other state that is not the current state.

The rest of the states emulate this same pattern but instead of the second DO WHILE, the system reverts to the default state. Ie. If no buttons are pressed (no traffic in any other lanes), during the timing cycles, the system reverts to the Main Straight state.

### LCD Display with RTC

As per the specifications, the LCD has to continulay display the current time and the state that the system is in. The programming code that operates the LCD screen is shown below. Beacuse the RTC is needed to obtain the current time, it will also be explained.

void setup\_clock(void);

Using I2C communication bus, this methods sets up all needed bytes to contact and manipulate the RTC. The method accesses the RTC by first sending the RTC’s address and the write mode (0xD0) down the I2C bus then waits for the acknowledge bit from the RTC. Once thats recieved, the method writes to the RTC to set the current time from to the computer to the current time in the RTC.

void get\_time(void);

Using the I2C again, this method goes through the same process to identify the RTC as setup\_clock. After doing so, it writes which address will be accessed (0x00) then, restarts the I2C, before sending another command to read the data in that address (0xD1) which will be the time and stores the values in hours, minutes, seconds.

void display\_time(unsigned char state[]);

This method is used to display the time on the first line of the LCD screen by manipulating the values obtained from the I2C by shifting them left by 4. Then they are converted to ASCII since thats what the LCD requires to display the correct character. The array that the method recieves is used to display the current state on the second line of the LCD.

### USART with RTC

By importing the usart.h it is possible to eliminate bit banging which would complicate the methods involved in sending data across the serial line through to the terminal. To initialize the USART, the following code is used:

OpenUSART( USART\_TX\_INT\_OFF &

USART\_RX\_INT\_OFF &

USART\_ASYNCH\_MODE &

USART\_EIGHT\_BIT &

USART\_CONT\_RX &

USART\_BRGH\_HIGH,

25 );

This runs the method that opens the USART communication between the traffic light program and the terminal program. This method is found in usart.c. This sets the following things:

* Sets the transmit and recieve interrupts off
* sets it to asynchronous mode so that it would be possible to set the baud rate
* sets it to eight bit transmit/recieve mode
* sets it to high speed transmission
* setting the SPBRG to 25 sets the baud rate to 9600 due to the below formula:

For High Speed:

SPBRG = (Fosc / (16 x Baud rate))

For Low Speed:

SPBRG = (Fosc / (64 x Baud rate))

Once the initialization was done, the following methods were called to display the trace message onto the terminal.

void send\_USART(const unsigned char \*text)

This accepts the current state message array as a parameter, which it then displays one letter at a time on the terminal screen.

void display\_USART(void)

This is used in conjunction with the I2C’s get\_time method to display the time when the system switches state.

# Conclusion

Combining the above parts after testing was simple due to the use of the h files. By copying and pasting chunks of code as needed, it was possible to create a fully functional Traffic Control System which met the initial specifications.

Even though the (Straight and Turning) state for each road was removed due to the inability to use the same LED twice, which could easily be remedied by altering the wiring of the hardware and even though the green/yellow/red cycle is simulated with an LED, the proof of concept is there and this Traffic Control Project is entirely valid in a real life situation.

# Appendix

Figure – Draft of the X-Intersection

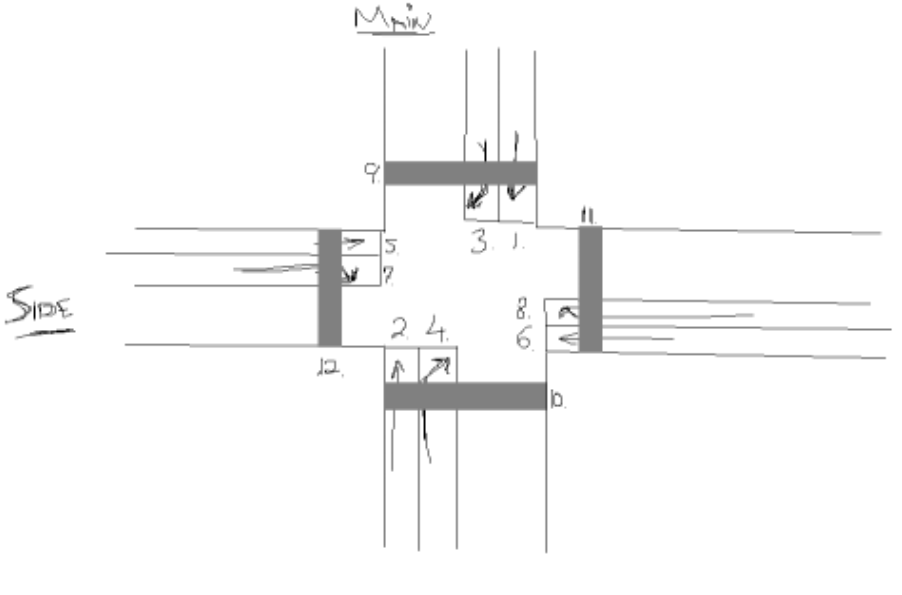


Figure - State Diagram



Figure - Hardware Sheet 1

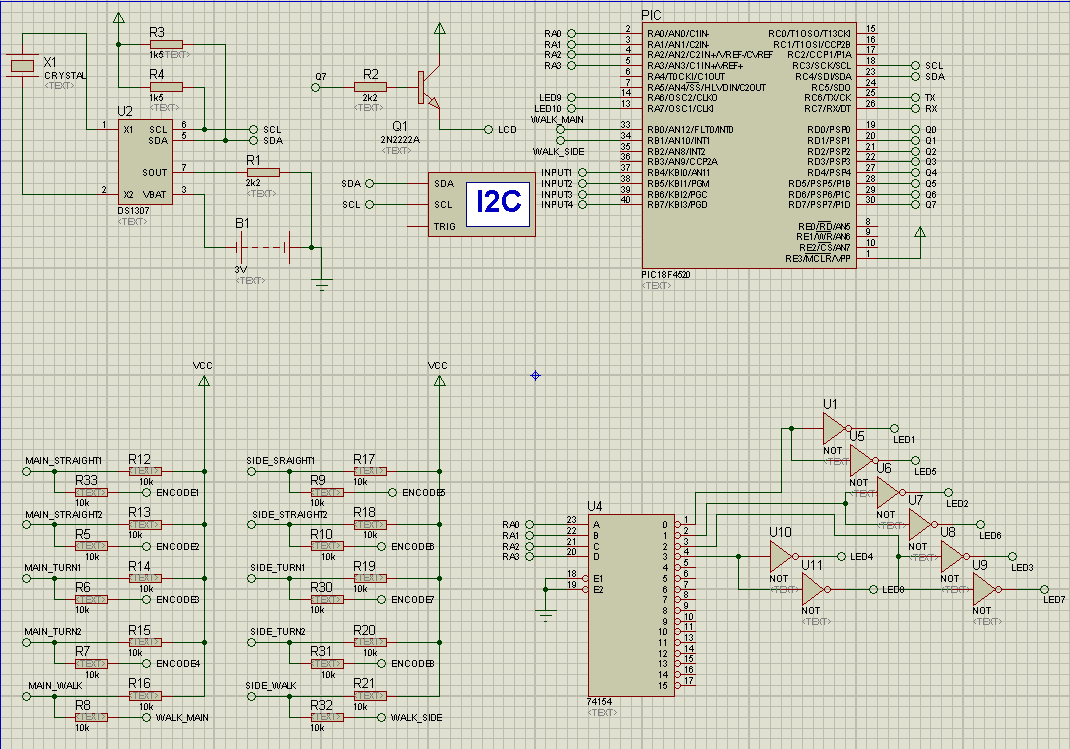


Figure - Hardware Sheet 2

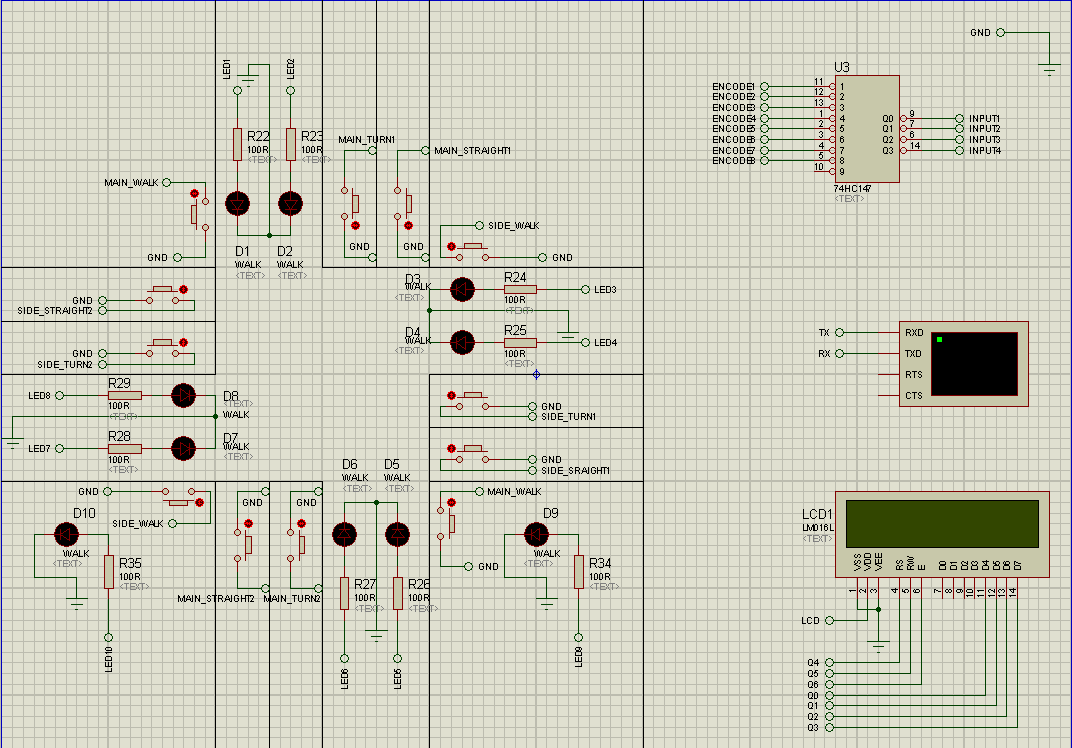


Figure - State Machine

switch(state)

{

//======================DEFAULT/MAIN ROAD STRAIGHTS====================

case(MAIN\_STRAIGHT):

currentTime = 0;

minTime = 8;

newState = 0;

get\_time();

display\_USART();

send\_USART(main\_straight\_msg\_USART);

set\_LED(led[0]);

do

{

get\_time();

display\_time(main\_straight\_msg);

delay\_ms(1000);

currentTime = currentTime++;

checkPortB = PORTB&0xF0;

checkPortB = checkPortB>>0x4;

if(!MAIN\_WALK\_B)

{

main\_peds\_on = 1;

}

if(!SIDE\_WALK\_B)

{

side\_peds\_on = 1;

}

if(checkPortB==MAIN\_STRAIGHT\_B1 || checkPortB==MAIN\_STRAIGHT\_B2)

{

if(minTime < maxTime-2)

{

minTime = minTime+2;

}

}

if(checkPortB==MAIN\_TURN\_B1 || checkPortB==MAIN\_TURN\_B2)

{

newState = 1;

}

else if((checkPortB==SIDE\_STRAIGHT\_B1 || CheckPortB==SIDE\_STRAIGHT\_B2)&&(newState!=1))

{

newState = 2;

}

else if((checkPortB==SIDE\_TURN\_B1 || checkPortB==SIDE\_TURN\_B2)&&(newState!=1)&&(newState!=2))

{

newState = 3;

}

} while (currentTime < minTime);

do

{

get\_time();

display\_time(main\_straight\_msg);

checkPortB = PORTB&0xF0;

checkPortB = checkPortB>>0x4;

if(!MAIN\_WALK\_B)

{

main\_peds\_on = 1;

}

if(!SIDE\_WALK\_B)

{

side\_peds\_on = 1;

}

if(checkPortB==MAIN\_TURN\_B1 || checkPortB==MAIN\_TURN\_B2)

{

newState = 1;

}

else if((checkPortB==SIDE\_STRAIGHT\_B1 || checkPortB==SIDE\_STRAIGHT\_B2)&&(newState!=1))

{

newState = 2;

}

else if((checkPortB==SIDE\_TURN\_B1 || checkPortB==SIDE\_TURN\_B2)&&(newState!=1)&&(newState!=2))

{

newState = 3;

}

}while(newState==0);

if(newState==1)

{

blink\_LED(led[0]);

state = MAIN\_TURN;

}

else if(newState==2)

{

blink\_LED(led[0]);

state = SIDE\_STRAIGHT;

}

else if(newState==3)

{

blink\_LED(led[0]);

state = SIDE\_TURN;

}

break;

//==========================MAIN ROAD TURNS============================

case(MAIN\_TURN):