**Traffic Light System Project Report**

by

**Joshua Susana**

**A Thesis**

*Submitted to the Faculty of Indiana University Purdue University Indianapolis*

*In Partial Fulfillment of the Requirements for the degree of*

**Bachelor of Science in Computer Engineering**

**Logo

Description automatically generated**

Department of Electrical and Computer Engineering at IUPUI

Indianapolis, Indiana

May 2024

**TABLE OF** **CONTENTS**

LIST OF FIGURES 3

LIST OF TABLES 4

INTRODUCTION 5

PROJECT REQUIREMENTS 6

1. DESIGN DETAILS 7

1.1 Hardware Design 7

1.2 Software Design 9

IMPLEMENTATION DETAILS 11

EXPERIMENT RESULTS 14

DISCUSSION 15

CONCLUSION 16

REFERENCES 17

APPENDIX A. SPECIFICATION OF ALL PARTS 18

APPENDIX B. COMPLETE SOFTWARE CODE 19

APPENDIX C. VIDEO OF PRODUCT IN OPERATOIN 31

LIST OF FIGURES

Figure 1. Schematic Diagram of Hardware Design Top Half 7

Figure 2. Schematic Diagram of Hardware Design Bottom Half 8

Figure 3. Block Diagram of Software Design 9

Figure 4. Flowchart of Operation Sequence 10

Figure 5. LED Wiring Diagram 11

Figure 6. Push Button Wiring Diagram 12

Figure 7. Layout Diagram 13

LIST OF TABLES

Table 1. List of Parts and Tools 11

Introduction

As the world population grows rapidly, so has the number of vehicles on the road. This has been leading to heavier traffic, leading to longer time traveling on the road, which leads to the burning of more fossil fuels to make up for that fuel. Faulty traffic systems worsen this issue by increasing the travel time. This project tries to ease this issue by implementing a traffic light system to efficiently manage traffic flow.

This project focuses on creating an embedded system that uses a microcontroller, push buttons, and LED’s to simulate a traffic light system managing traffic flow.

Overall, this project tries to demonstrate the applicability of using embedded systems concepts to solve real-life problems. The development of an efficient and reliable traffic light system can significantly improve traffic flow and reduce travel time, improving the quality of life for drivers and decreasing the contribution of climate change from vehicles.

**PROJECT REQUIREMENTS**

The requirements of this project were outlined by the professor Stanley Yung-Ping Chien. Only the requirements that will need to be mentioned here will be mentioned. These include the must of using a PIC microcontroller, and in addition to using general purpose I/O, for each student in a group, two peripheral components must be used.

This project used the PIC18F4331 microcontroller without the aid of a premade PIC circuit board, as will be shown in detail in the schematic, wiring, and layout diagrams.

This project also used general purpose I/O for the LED’s representing the traffic lights. For the peripheral components, since one student was in the group, only two peripheral components were needed. Technically three were used for the project. These included the TIMER0 module to measure the timing of switching the traffic lights, the INT0 external interrupt for the north-south walk button, and the INT1 external interrupt for the east-west walk button.

**1. DESIGN DETAILS**

**1.1 Hardware Design**

Diagram, schematic

Description automatically generated

Figure 1. Schematic Diagram of Hardware Design Top Half

Diagram, schematic

Description automatically generated

Figure 2. Schematic Diagram of Hardware Design Bottom Half

The top half of the schematic diagram shows how the PIC18F4331 microcontroller is connected to the PICKit 4 and the push buttons.

On the PICKit 4, pins five through eight are unused. For the rest of the pins, pin one is the master clear, pin two is the positive supply voltage, pin three is the ground voltage, pin four is the program data pin, and pin five is the program clock pin. From microcontroller to PICKit, the connections are as follows: MCLR to pin 1, AVDD and VDD to pin 2, AVSS and VSS to pin 3, RB7 to pin 4, and RB6 to pin 5. Note that for AVDD and VDD, both of these pins do not actually directly connect to pin 2. The connection is made through an equivalent circuit on the breadboard. The same concept applies to AVSS and VSS.

From microcontroller to push buttons, the connections are as follows: RC3 to NS Walk Button, and RC4 to EW Walk Button.

The bottom half of the schematic diagram shows how the PIC18F4331 microcontroller is connected to the LED’s. From microcontroller to LED’s, the connections are as follows: RD1 to NS Red, RD2 to NS Yellow, RD3 to NS Green, RD0 to EW Red, RE2 to EW Yellow, RC5 to EW Green, RC6 to NS Walk Go, RC2 to NS Walk Stop, RD4 to EW Walk Go, and RC0 to EW Walk stop.

**1.2 Software Design**

No data structures were used to implement this project, so data structure definitions will not be defined in this project report.

Diagram

Description automatically generated

Figure 3. Block Diagram of Software Design

Diagram

Description automatically generated

Figure 4. Flowchart of Operation Sequence

Notice in Figure 4 that the traffic lights always change the same way regardless of when the walk button is pressed. This is similar to many traffic lights in the real world, where the walk button does not make the pedestrian’s axis turn green faster, but simply tells them when they can walk.

**IMPLEMENTATION DETAILS**

Table 1. List of Parts and Tools

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Model** | **Source of Purchasing** | **Price** |
| PIC18F | 4331 – I/P | Mouser Electronics | $8.23 |
| PICKit In-Circuit Debugger | PICKit 4 | Mouser Electronics | $90.32 |
| Jumper Wires | M/M 6" 40 PCS | Digi-Key Electronics | $6.00 |
| 2 Red LED’s | DIFFUSED T-1 3/4 T/H | Digi-Key Electronics | $1.32 |
| Green LED | DIFFUSED T-1 3/4 T/H | Digi-Key Electronics | $0.66 |
| 2 Tactile Switches | SPST-NO 0.05A 12V | Digi-Key Electronics | $0.52 |

Diagram

Description automatically generated

Figure 5. LED Wiring Diagram

For this wiring diagram, the five volts would be coming from the microcontroller pin, the second component from top to bottom is the LED, the third component is the resistor, and the bottom component is the ground voltage.

A picture containing text, shoji, clock, tiled

Description automatically generated

Figure 6. Push Button Wiring Diagram

For this wiring diagram, the five volts would be coming from the microcontroller pin, the middle component is the push button/tactile switch, and the bottom component is the ground voltage.

Diagram, schematic

Description automatically generated

Figure 7. Layout Diagram

A layout diagram is simply the physical placements of the components within the embedded system. There is not much detail here since the schematic and wiring diagrams fulfill that task. For this project, the PICKit 4/power supply is always near the top left, the PIC18F4331 sits at the top of the breadboard, the LED’s are placed right below the microcontroller, and the push buttons/tactile switches sit right below the LED’s.

For software tools, the IDE used was MPLAB X IDE Version 6.05. The compiler used was the XC8 Compiler Version 2.40. When it comes to configuration settings, for compiler and debugger settings, nothing was changed. The “default” was used. For libraries, the only one used was the “xc.h” library.

**EXPERIMENT RESULTS**

The project requirements that will be compared with the results in the section will be does the project mainly use a PIC microcontroller, specifically the PIC18F4331, is general purpose I/O used, are at least two peripheral components were used, and does the entire project function at one hundred percent.

First, as stated and shown in diagrams many times in this project report, the project mainly uses a PIC microcontroller. No other microcontroller is used.

Second, general purpose I/O is used for the ten LED’s that make up the traffic lights and the walk lights.

Third, three peripheral components were used. Again, they were the TIMER0 module for the traffic light changes, the INT0 external interrupt for the NS walk button, and the INT1 external interrupt for the EW Walk button.

Lastly, the project has a functionality of one hundred percent, with no bugs/glitches found after testing. This will be shown clearer in the video of the product in operation.

**DISCUSSION**

A limitation in this project is if a car or pedestrian is waiting on an axis that is currently red, and there are no other cars/pedestrians, they still have to wait the full time for their axis to turn green. A solution/improvement to this problem would be to first, add sensors to all four cardinal directions so that cars that are waiting can be detected. When these sensors detect a car, the application to the source code would be similar to the walk buttons by setting a flag, since the detection of a car/pedestrian does not currently affect the traffic light switches. Then, polling or an interrupt could be used to constantly check the flags of the walk buttons and the sensors. The condition in the polling/interrupt would be if only one axis has a flag set, and the axis is not currently green, then change the traffic light phase immediately to the intersecting axis being yellow. This way, the traffic light switching will now “naturally” be at the fastest time to switch to green for the desired car/pedestrian.

As stated in the introduction of this project report, the significance of this project is that it models an efficient traffic light system, which in turn can provide shorter travel times, and less burning of fossil fuels.

**CONCLUSION**

In conclusion, the traffic light system fulfills more than outlined in the project requirements section. The hardware and software design implementation were carefully planned out so that it could be shown that the traffic light system demonstrates one hundred percent of its functionality. The experiment results emphasize the fact that the project requirements have been met. Lastly, a discussion was made as to how the traffic light system would benefit from sensors and some additions to the “algorithm” with those sensors.

Overall, this embedded systems final project was a great way to learn more about the field by providing an outlet to design and implement an embedded system that addresses a real-world problem.

**REFERENCES**

Empty, since I did not reference any online/outside material such as online websites/articles.

**APPENDIX A. SPECIFICATION OF ALL PARTS**

* PIC18F4331 microcontroller
* MPLAB PICKit 4 In-Circuit Debugger
* Male/Male Jumper Wires, 20 cm
* 51K Ω Resistor
* 820 Ω Resistors
* Red LED’s Diffused T-1 3/4 T/H
* Yellow LED’s Diffused T-1 3/4 T/H
* Green LED’s Diffused T-1 3/4 T/H
* Tactile Switches SPST-NO 0.05A 12V
* Breadboard 830 Point Solderless

**APPENDIX B. COMPLETE SOFTWARE CODE**

**Configuration.h**

#ifndef XC\_HEADER\_TEMPLATE\_H

#define XC\_HEADER\_TEMPLATE\_H

#include <xc.h> // include processor files - each processor file is guarded.

#pragma config OSC = IRCIO // Oscillator Selection bits (Internal oscillator block, port function on RA6 and port function on RA7)

#pragma config FCMEN = ON // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor enabled)

#pragma config IESO = ON // Internal External Oscillator Switchover bit (Internal External Switchover mode enabled)

// CONFIG2L

#pragma config PWRTEN = ON // Power-up Timer Enable bit (PWRT enabled)

#pragma config BOREN = ON // Brown-out Reset Enable bits (Brown-out Reset enabled)

// BORV = No Setting

// CONFIG2H

#pragma config WDTEN = OFF // Watchdog Timer Enable bit (WDT disabled (control is placed on the SWDTEN bit))

#pragma config WDPS = 32768 // Watchdog Timer Postscale Select bits (1:32768)

#pragma config WINEN = OFF // Watchdog Timer Window Enable bit (WDT window disabled)

// CONFIG3L

#pragma config PWMPIN = OFF // PWM output pins Reset state control (PWM outputs disabled upon Reset (default))

#pragma config LPOL = HIGH // Low-Side Transistors Polarity (PWM0, 2, 4 and 6 are active-high)

#pragma config HPOL = HIGH // High-Side Transistors Polarity (PWM1, 3, 5 and 7 are active-high)

#pragma config T1OSCMX = ON // Timer1 Oscillator MUX (Low-power Timer1 operation when microcontroller is in Sleep mode)

// CONFIG3H

#pragma config FLTAMX = RC1 // FLTA MUX bit (FLTA input is multiplexed with RC1)

#pragma config SSPMX = RC7 // SSP I/O MUX bit (SCK/SCL clocks and SDA/SDI data are multiplexed with RC5 and RC4, respectively. SDO output is multiplexed with RC7.)

#pragma config PWM4MX = RB5 // PWM4 MUX bit (PWM4 output is multiplexed with RB5)

#pragma config EXCLKMX = RC3 // TMR0/T5CKI External clock MUX bit (TMR0/T5CKI external clock input is multiplexed with RC3)

#pragma config MCLRE = ON // MCLR Pin Enable bit (Enabled)

// CONFIG4L

#pragma config STVREN = ON // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)

#pragma config LVP = OFF // Low-Voltage ICSP Enable bit (Low-voltage ICSP disabled)

// CONFIG5L

#pragma config CP0 = OFF // Code Protection bit (Block 0 (000200-000FFFh) not code-protected)

#pragma config CP1 = OFF // Code Protection bit (Block 1 (001000-001FFF) not code-protected)

// CONFIG5H

#pragma config CPB = OFF // Boot Block Code Protection bit (Boot Block (000000-0001FFh) not code-protected)

#pragma config CPD = OFF // Data EEPROM Code Protection bit (Data EEPROM not code-protected)

// CONFIG6L

#pragma config WRT0 = OFF // Write Protection bit (Block 0 (000200-000FFFh) not write-protected)

#pragma config WRT1 = OFF // Write Protection bit (Block 1 (001000-001FFF) not write-protected)

// CONFIG6H

#pragma config WRTC = OFF // Configuration Register Write Protection bit (Configuration registers (300000-3000FFh) not write-protected)

#pragma config WRTB = OFF // Boot Block Write Protection bit (Boot Block (000000-0001FFh) not write-protected)

#pragma config WRTD = OFF // Data EEPROM Write Protection bit (Data EEPROM not write-protected)

// CONFIG7L

#pragma config EBTR0 = OFF // Table Read Protection bit (Block 0 (000200-000FFFh) not protected from table reads executed in other blocks)

#pragma config EBTR1 = OFF // Table Read Protection bit (Block 1 (001000-001FFF) not protected from table reads executed in other blocks)

// CONFIG7H

#pragma config EBTRB = OFF // Boot Block Table Read Protection bit (Boot Block (000000-0001FFh) not protected from table reads executed in other blocks)

#endif

**Final\_Project\_Main.c**

#include <xc.h>

#include "Configuration.h"

#define \_XTAL\_FREQ 31000

#define NS\_RedLight RD1

#define NS\_YellowLight RD2

#define NS\_GreenLight RD3

#define EW\_RedLight RD0

#define EW\_YellowLight RE2

#define EW\_GreenLight RC5

#define NS\_WalkRed RC2

#define NS\_WalkGo RC6

#define EW\_WalkRed RC0

#define EW\_WalkGo RD4

//Global variables so that both ISR's and main can read/use them

int global\_phase = 1;

int NS\_WalkFlag = 0;

int EW\_WalkFlag = 0;

int Startup = 1;

void \_\_interrupt(high\_priority) high\_ISR() {

if(INT0IF == 1) {

/\*For the NS walk button, if the NS axis is either green or yellow, then

set the NS\_WalkFlag and clear the interrupt flag. Otherwise, set the

NS\_WalkFlag, but turn WalkRed LED on and WalkGo LED off\*/

if(global\_phase == 1) {

NS\_WalkFlag = 1;

INT0IF = 0;

return;

}

/\*Different for this phase for some reason, since without the first 2

lines in the if-statement, WalkRed LED turns on and WalkGo LED turns

off.\*/

if(global\_phase == 2) {

NS\_WalkRed = 0;

NS\_WalkGo = 1;

NS\_WalkFlag = 1;

INT0IF = 0;

return;

}

if(global\_phase == 3) {

NS\_WalkFlag = 1;

INT0IF = 0;

return;

}

if(global\_phase == 4) {

NS\_WalkRed = 1;

NS\_WalkGo = 0;

NS\_WalkFlag = 1;

INT0IF = 0;

return;

}

if(global\_phase == 5 || global\_phase == 6) {

NS\_WalkRed = 1;

NS\_WalkGo = 0;

NS\_WalkFlag = 1;

INT0IF = 0;

return;

}

if(global\_phase == 7) {

NS\_WalkRed = 1;

NS\_WalkGo = 0;

NS\_WalkFlag = 1;

INT0IF = 0;

return;

}

if(global\_phase == 8) {

NS\_WalkRed = 1;

NS\_WalkGo = 0;

NS\_WalkFlag = 1;

INT0IF = 0;

return;

}

}

if(INT1IF == 1) {

/\*For the EW walk button, if the EW axis is either green or yellow, then

set the EW\_WalkFlag and clear the interrupt flag. Otherwise, set the

EW\_WalkFlag, but turn WalkRed LED on and WalkGo LED off\*/

if(global\_phase == 1) {

EW\_WalkRed = 1;

EW\_WalkGo = 0;

EW\_WalkFlag = 1;

INT1IF = 0;

return;

}

if(global\_phase == 2) {

EW\_WalkRed = 1;

EW\_WalkGo = 0;

EW\_WalkFlag = 1;

INT1IF = 0;

return;

}

if(global\_phase == 3) {

EW\_WalkRed = 1;

EW\_WalkGo = 0;

EW\_WalkFlag = 1;

INT1IF = 0;

return;

}

if(global\_phase == 4) {

EW\_WalkRed = 1;

EW\_WalkGo = 0;

EW\_WalkFlag = 1;

INT1IF = 0;

return;

}

if(global\_phase == 5) {

EW\_WalkFlag = 1;

INT1IF = 0;

return;

}

/\*Different for this phase for some reason, since without the first 2

lines in the if-statement, WalkRed LED turns on and WalkGo LED turns

off.\*/

if(global\_phase == 6) {

EW\_WalkRed = 0;

EW\_WalkGo = 1;

EW\_WalkFlag = 1;

INT1IF = 0;

return;

}

if(global\_phase == 7) {

EW\_WalkFlag = 1;

INT1IF = 0;

return;

}

if(global\_phase == 8) {

EW\_WalkRed = 1;

EW\_WalkGo = 0;

EW\_WalkFlag = 1;

INT1IF = 0;

return;

}

}

}

void \_\_interrupt(low\_priority) low\_ISR() {

if(TMR0IF == 1) {

/\*Note that global\_phases in reverse order, since otherwise it will

increment 1 through 8 and execute all of the if-statements at once\*/

if(global\_phase == 8) {

NS\_RedLight = 1;

NS\_YellowLight = 0;

NS\_GreenLight = 0;

EW\_RedLight = 1;

EW\_YellowLight = 0;

EW\_GreenLight = 0;

/\*Note that this is the only if-statement we use the return

keyword. Otherwise, it will execute this if-statement, then the

code in the if(global\_phase == 1) statement\*/

global\_phase = 1;

TMR0IF = 0;

return;

}

if(global\_phase == 7) {

NS\_RedLight = 1;

NS\_YellowLight = 0;

NS\_GreenLight = 0;

EW\_RedLight = 0;

EW\_YellowLight = 1;

EW\_GreenLight = 0;

/\*If EW WalkGo LED is on, detected through EW\_WalkFlag, then make

sure EW\_WalkRed LED turns on and EW\_WalkGo LED off immediately\*/

if(EW\_WalkFlag == 1) {

EW\_WalkRed = 1;

EW\_WalkGo = 0;

EW\_WalkFlag = 0;

}

global\_phase++;

}

if(global\_phase == 5 || global\_phase == 6) {

NS\_RedLight = 1;

NS\_YellowLight = 0;

NS\_GreenLight = 0;

EW\_RedLight = 0;

EW\_YellowLight = 0;

EW\_GreenLight = 1;

/\*If EW push button was pressed before this axis turns green,

detected through EW\_WalkFlag, then immediately turn WalkGo LED

on.\*/

if(EW\_WalkFlag == 1) {

EW\_WalkRed = 0;

EW\_WalkGo = 1;

}

global\_phase++;

}

if(global\_phase == 4) {

NS\_RedLight = 1;

NS\_YellowLight = 0;

NS\_GreenLight = 0;

EW\_RedLight = 1;

EW\_YellowLight = 0;

EW\_GreenLight = 0;

global\_phase++;

}

if(global\_phase == 3) {

NS\_RedLight = 0;

NS\_YellowLight = 1;

NS\_GreenLight = 0;

EW\_RedLight = 1;

EW\_YellowLight = 0;

EW\_GreenLight = 0;

/\*If NS WalkGo LED is on, detected through NS\_WalkFlag, then make

sure NS\_WalkRed LED turns on and NS\_WalkGo LED off immediately\*/

if(NS\_WalkFlag == 1) {

NS\_WalkRed = 1;

NS\_WalkGo = 0;

NS\_WalkFlag = 0;

}

global\_phase++;

}

if(global\_phase == 1 || global\_phase == 2) {

NS\_RedLight = 0;

NS\_YellowLight = 0;

NS\_GreenLight = 1;

EW\_RedLight = 1;

EW\_YellowLight = 0;

EW\_GreenLight = 0;

/\*If NS push button was pressed before this axis turns green,

detected through NS\_WalkFlag, then immediately turn WalkGo LED

on.\*/

if(NS\_WalkFlag == 1) {

NS\_WalkRed = 0;

NS\_WalkGo = 1;

}

if(Startup == 1) {

NS\_WalkRed = 1;

NS\_WalkGo = 0;

EW\_WalkRed = 1;

EW\_WalkGo = 0;

Startup = 0;

}

global\_phase++;

}

TMR0IF = 0;

}

}

void main(void) {

//For crystal frequency

IRCF2 = 0;

IRCF1 = 0;

IRCF0 = 0;

//Setting LED's as output

TRISD1 = 0;

TRISD2 = 0;

TRISD3 = 0;

TRISD0 = 0;

TRISE2 = 0;

TRISC5 = 0;

TRISC2 = 0;

TRISC6 = 0;

TRISC0 = 0;

TRISD4 = 0;

//Initializing all LED's to off

NS\_RedLight = 0;

NS\_YellowLight = 0;

NS\_GreenLight = 0;

EW\_RedLight = 0;

EW\_YellowLight = 0;

EW\_GreenLight = 0;

NS\_WalkRed = 0;

NS\_WalkGo = 0;

EW\_WalkRed = 0;

EW\_WalkGo = 0;

RCON = 0b10000000; //Setting IPEN bit in RCON register

T0CON = 0b11000101; //Enabling timer as 8-bit with prescale value of

//1:64

INTCON = 0b11110000; //Setting GIE, PEIE, TMR0IE, and INT0 bits

INTCON2 = 0b01100000; //Setting INT0 and INT1 on rising edge, and setting

//TMR0 as a low priority interrupt

INTCON3 = 0b01001000; //Enabling INT1 external interrupt as high priority

//Note that the INT0 external interrupt is always

//high priority.

while(1);

return;

}

**APPENDIX C. VIDEO OF PRODUCT IN OPERATION**

A hyperlink to a YouTube video will be provided below.

<https://www.youtube.com/watch?v=3bQae0YVTDs>