**Basic and Finite State Machine LED and Switch I/O Programming**

**6th Laboratory Report for ECE 383**

**Microcomputers**

**Submitted by**

**Yichen Huang**

**11906882**

**Shomari Thomas**

**11672867**

**The University of Alabama**

**Tuscaloosa, Alabama 35487**

**March 4, 2020**

Table of Contents

[Abstract 3](#_Toc34253742)

[Introduction 4](#_Toc34253743)

[Prelab 5](#_Toc34253744)

[Task 1. Expanding the PIC24 Reference System Schematic 5](#_Toc34253745)

[Task 2. Expand the PIC24 System Printed Circuit Board Layout 6](#_Toc34253746)

[Procedure/Results 7](#_Toc34253747)

[Task 3. A Basic LED Problem 7](#_Toc34253748)

[Task 4. Software-Based Finite State Machine for LED/Switch I/O 7](#_Toc34253749)

[Task 5. Variable Rotating LED 8](#_Toc34253750)

[Conclusion 9](#_Toc34253751)

[Appendixes 10](#_Toc34253752)

[Appendix A – Reports of Task 2 10](#_Toc34253753)

[Appendix B – Source code of Task 3 11](#_Toc34253754)

[Appendix C – Source code and ACM chart of Task 3 13](#_Toc34253755)

[Appendix D – Source code and Answer of Task 4 19](#_Toc34253756)

# Abstract

Lab 6 was an introduction to using the basic I/O ports associated with the PIC24 to create a switch-based input and LED-based output. A program was created using the C language to create a software-based finite state machine to solve a basic I/O problem. During the lab, we expanded the system schematic and printed circuit board for the PIC24, created a software-based finite state machine to solve a simple LED/switch I/O problem, and created C code to solve a multicolor LED problem involving binary to gray code conversions.

For task 1, we expanded the PIC24 system schematic based on a given design, to include pushbuttons, resistors, and LEDs using PCB Artist. In task 2, we converted the new schematic created in task 1 to a printed circuit board layout and controlled and noted the locations of the components on the board. In task 3, a simple LED problem was solved using C code and the PIC24 hardware. For task 4, a software-based finite state machine was created to solve a given LED problem using two LEDs and a given circuit involving the PIC24 on a breadboard. In task 5, a multicolored RGB LED was used in a circuit with the PIC24, and C code was written to involve pushbuttons and conversion from binary codes to gray codes.

In Lab 6, we became familiar with involving the PIC24 in a breadboard layout, and how to write code to involve the I/O ports of the PIC24, and we verified the success of our programs with a demo using our implementations on the PIC24 hardware.

# Introduction

The objective of this lab is to become more familiar with PIC24 I/O and how to configure and use pins for input and output. Task one has us creating a system schematic for the PIC24 that includes resistors, switches and LEDs. Task two has us taking the system schematic created in task one and converting it into a printed circuit board schematic. Task three has us taking a given piece of code, running it on the PIC24, and then modifying the code and running it on the PIC24. Task four is the most complicated and involves creating a finite state machine that solves a basic I/O LED and switch button problem. Task five involves writing C code to run a program that turns on a RGB LED based on what switches are being held down. This code also converts binary codes to gray codes.

# Prelab

## Task 1. Expanding the PIC24 Reference System Schematic

In the first task, we create the new schematic circuit based on the previous lab. The figure of schematic circuit is Fig.1

A screenshot of a computer

Description automatically generated

Figure 1. Schematic Circuit of Task 1

## Task 2. Expand the PIC24 System Printed Circuit Board Layout

Task two had us take the system schematic created in Task one and convert it to a printed circuit board (Fig.2) and note where the components are on the board from the list of and y coordinates. We also created mounting holes to the printed circuit board design. To verify that our circuit board passed basic electrical design rules, we clicked Tools->Design Rule Check, checking both the Spacing, Nets, and Manufacturing boxes. This rule check reported that no errors were found in our designs. Finally, we generated two different reports on our printed circuit board. The first of these reports was a bill of materials CSV file. The second of these reports was a component positions report CSV file. They are both in the [Appendix A](#_Appendix_A_–) as Bill of Materials CSV File and Components Position CSV File.

A screen shot of a computer

Description automatically generated

Figure 2 PCB Layout

# Procedure/Results

## Task 3. A Basic LED Problem

Task three was broken into two parts, 3a and 3b. 3a included creating a new project, and copying code to a file in the project, and then downloading the code to the PIC24 and running the code to verify it is working properly. The program from 3a blinked an onboard LED at a constant rate. 3b included creating a new project and taking the code from 3a and modifying it to blink the LED at two different rates, one fast and one slow. The code for 3b was then downloaded to the PIC24 and verified to work properly. The source code will be shown in [Appendix B](#_Appendix_B_–).

## Task 4. Software-Based Finite State Machine for LED/Switch I/O

Task four was the most complex of the tasks for this lab and involved creating a software-based finite state machine to implement a simple LED problem using the PIC24 I/O ports on a breadboard configuration. This task also implemented the use of pushbuttons which were used a source of input to the program. The finite state machine solved a given problem and was implemented using C code. The code was then downloaded to the PIC24 and tested in the breadboard circuit. Fig.3 is the layout of bread board and the source code and ASM chart will be shown in [Appendix C](#_Appendix_C_–).

**A circuit board

Description automatically generated**

Figure 3. Layout of bread board for Task4

## Task 5. Variable Rotating LED

Task 5 implemented the use of RGB LED in a circuit with the PIC24 on the Microstick II. A C program was written to solve a given problem, and the code would display different binary or gray codes and convert between them based on what pushbuttons were being pushed or released. The C code was then downloaded to the PIC24 and tested in the breadboard circuit. The source code and answers of questions will be shown in Appendix D. Fig.4 is the breadboard layout of Task5.

**A circuit board

Description automatically generated**

Figure 4. Breadboard layout of Task5

# Conclusion

This sixth lab was a learning experience for the use of the PIC24 I/O ports. We were taught how to implement and run programs that used and configured some of the PIC24 I/O ports on a breadboard. We now know how to properly interlay the PIC24 on a breadboard to take full advantage of its I/O ports. We also know how to develop and write C code that can control the PIC24 ports, and solve problems using both onboard and external devices. Overall, we now have a much better understanding of the PIC24 I/O systems.

# Appendixes

## Appendix A – Reports of Task 2

### Bill of Material Report

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Component | Package | Value | Manuf | Manuf Part No | Distrib | Distrib Part No | Ref Name | Qty |
| C | DSC | 0.1uF |  |  |  |  | C2 C4 C5 | 3 |
| C | DSC | 10uF |  |  |  |  | C1 C3 | 2 |
| LED | DSC | LED 5MM RED |  |  |  |  | LED1 LED4 LED5 LED2 LED3 | 5 |
| LM2937-3.3 | T03B |  |  | LM2937ET-50 |  |  | U2 | 1 |
| PIC24HJ128GP502 | USER |  |  |  |  |  | U1 | 1 |
| PUSHBUTTON | USER |  |  |  |  |  | SW1 SW2 SW3 | 3 |
| R | DSC | 10K |  |  |  |  | R1 R2 | 2 |
| R | DSC | 910 |  |  |  |  | R3 R4 R5 R6 | 4 |
|  |  |  |  |  |  |  |  | 21 |
| PCB Artist Bill of Materials is provided for reference only and must be verified by the user. | | | | | | | |  |

### Position Report

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Component | Side | Centre X | Centre Y | Rotation |
| C1 | C | Top | 64 | 63.2 | 270 |
| C2 | C | Top | 6 | 36.8 | 90 |
| C3 | C | Top | 22 | 36.8 | 90 |
| C4 | C | Top | 71 | 52.2 | 270 |
| C5 | C | Top | 22 | 57.8 | 90 |
| LED1 | LED | Top | 70 | 42.6 | 0 |
| LED2 | LED | Top | 20 | 8.6 | 0 |
| LED3 | LED | Top | 30 | 8.6 | 0 |
| LED4 | LED | Top | 40 | 8.6 | 0 |
| LED5 | LED | Top | 50 | 8.6 | 0 |
| U2 | LM2937-3.3 | Top | 70.5 | 62.5 | 270 |
| U1 | PIC24HJ128GP502 | Top | 13.8 | 43.5 | 0 |
| SW1 | PUSHBUTTON | Top | 57 | 72 | 0 |
| SW2 | PUSHBUTTON | Top | 47 | 72 | 0 |
| SW3 | PUSHBUTTON | Top | 37 | 72 | 0 |
| R1 | R | Top | 24 | 24.9 | 270 |
| R2 | R | Top | 14.1 | 68 | 0 |
| R3 | R | Top | 30 | 24.9 | 270 |
| R4 | R | Top | 36 | 24.9 | 270 |
| R5 | R | Top | 42 | 24.9 | 270 |
| R6 | R | Top | 48 | 24.9 | 270 |

## Appendix B – Source code of Task 3

#### Task 3a

##### Source code

#include "pic24\_all.h"

#if \_\_PIC24HJ128GP502\_\_

#define LED1 \_LATA0 // MicroStick II definitions

#define CONFIG\_LED1() CONFIG\_RA0\_AS\_DIG\_OUTPUT()

#endif

int main(void) { CONFIG\_LED1(); LED1=0;

while (1) { // Infinite while loop

LED1 = !LED1; // Toggle LED1

DELAY\_MS(100); // Delay 100ms

}

return 0;

}

#### Task 3b

##### Source code

#include "pic24\_all.h"

#if \_\_PIC24HJ128GP502\_\_

#define LED1 \_LATA0 // MicroStick II definitions

#define CONFIG\_LED1() CONFIG\_RA0\_AS\_DIG\_OUTPUT()

#endif

int main(void) { CONFIG\_LED1(); LED1=0;

int i;

while (1) { // Infinite while loop

i = 0;

while(i <= 50) {

LED1 = !LED1; // Toggle LED1

DELAY\_MS(100); // Delay 100ms

i++;

}

i = 0;

while(i < 25) {

LED1 = !LED1; // Toggle LED1

DELAY\_MS(200); // Delay 200ms

i++;

}

}

return 0;

}

## Appendix C – Source code and ACM chart of Task 3

#### ACM Chart

A picture containing text, indoor, whiteboard, wall

Description automatically generated

#### Source Code

#include "pic24\_all.h"

#define CONFIG\_LED1() CONFIG\_RB15\_AS\_DIG\_OUTPUT()

#define LED1 \_LATB15

#define CONFIG\_LED2() CONFIG\_RB2\_AS\_DIG\_OUTPUT()

#define LED2 \_LATB2

inline void CONFIG\_SW1() {

CONFIG\_RB14\_AS\_DIG\_INPUT();

ENABLE\_RB14\_PULLUP();

}

inline void CONFIG\_SW2() {

CONFIG\_RB12\_AS\_DIG\_INPUT();

ENABLE\_RB12\_PULLUP();

}

#define SW1 \_RB14

#define SW1\_PRESSED() SW1 == 1

#define SW1\_RELEASED() SW1 == 0

#define SW2 \_RB12

#define SW2\_PRESSED() SW2 == 1

#define SW2\_RELEASED() SW2 == 0

typedef enum {

STATE\_RESET = 0,

STATE\_WAIT\_FOR\_PRESS1, STATE\_WAIT\_FOR\_RELEASE1,

STATE\_WAIT\_FOR\_PRESS2, STATE\_WAIT\_FOR\_RELEASE2,

STATE\_WAIT\_FOR\_PRESS3, STATE\_WAIT\_FOR\_RELEASE3,

STATE\_WAIT\_FOR\_PRESS4, STATE\_WAIT\_FOR\_RELEASE4,

STATE\_WAIT\_FOR\_PRESS5, STATE\_WAIT\_FOR\_RELEASE5,

STATE\_WAIT\_FOR\_PRESS6, STATE\_WAIT\_FOR\_RELEASE6,

STATE\_BLINK1

} STATE;

int main(void) {

configClock();

STATE e\_mystate;

configBasic(HELLO\_MSG);

CONFIG\_SW1(); //configure switch1

CONFIG\_SW2(); //configure switch2

CONFIG\_LED1(); //configure LED1

CONFIG\_LED2(); //configure LED2

DELAY\_US(1); //give pull-ups time to work

e\_mystate = STATE\_WAIT\_FOR\_PRESS1;

while(1) {

switch (e\_mystate) {

case STATE\_WAIT\_FOR\_PRESS1:

LED1 = 1; //turn on the LED1

if(SW1\_PRESSED()) e\_mystate = STATE\_WAIT\_FOR\_RELEASE1;

break;

case STATE\_WAIT\_FOR\_RELEASE1:

if(SW1\_RELEASED()) e\_mystate = STATE\_WAIT\_FOR\_PRESS2;

break;

case STATE\_WAIT\_FOR\_PRESS2:

DELAY\_MS(100);

LED1 = 0; //turn off LED1

if(SW1\_PRESSED()) e\_mystate = STATE\_WAIT\_FOR\_RELEASE2;

break;

case STATE\_WAIT\_FOR\_RELEASE2:

if(SW1\_RELEASED()) e\_mystate = STATE\_BLINK1;

break;

case STATE\_BLINK1:

LED1 = 1; //turn on LED1

DELAY\_MS(10);

LED1 = 0; //first blink

DELAY\_MS(10);

LED1 = 1; //turn on LED1

DELAY\_MS(10);

LED1 = 0; //second blink

DELAY\_MS(10);

LED1 = 1; //freeze LED1 on

e\_mystate = STATE\_WAIT\_FOR\_PRESS3;

break;

case STATE\_WAIT\_FOR\_PRESS3:

DELAY\_MS(100);

if(SW1\_PRESSED()) e\_mystate = STATE\_WAIT\_FOR\_RELEASE3;

break;

case STATE\_WAIT\_FOR\_RELEASE3:

if(SW1\_RELEASED()) {

if(SW2) {

e\_mystate = STATE\_WAIT\_FOR\_PRESS4;

break;

}

else {

e\_mystate = STATE\_WAIT\_FOR\_PRESS1; // back to start

break;

}

}

break;

case STATE\_WAIT\_FOR\_PRESS4:

DELAY\_MS(100);

if(SW1\_PRESSED()) {

LED2 = 0;

e\_mystate = STATE\_WAIT\_FOR\_RELEASE4;

}

else {

DELAY\_MS(10); // Blink LED2 #1

LED2 = !LED2;

}

break;

case STATE\_WAIT\_FOR\_RELEASE4:

if(SW1\_RELEASED()) e\_mystate = STATE\_WAIT\_FOR\_PRESS5;

break;

case STATE\_WAIT\_FOR\_PRESS5:

DELAY\_MS(100);

if(SW1\_PRESSED()) {

LED2 = 0;

e\_mystate = STATE\_WAIT\_FOR\_RELEASE5;//Second Release

}

else {

DELAY\_MS(10);

LED2 = !LED2;

}

break;

case STATE\_WAIT\_FOR\_RELEASE5:

if(SW1\_RELEASED()) e\_mystate = STATE\_WAIT\_FOR\_PRESS6;

break;

case STATE\_WAIT\_FOR\_PRESS6:

DELAY\_MS(100);

if(SW2\_PRESSED()) {

LED2 = 0;

e\_mystate = STATE\_WAIT\_FOR\_RELEASE6;

}

else {

DELAY\_MS(5); // Blink LED2 #2

LED2 = !LED2;

}

break;

case STATE\_WAIT\_FOR\_RELEASE6:

if(SW2\_RELEASED()) e\_mystate = STATE\_WAIT\_FOR\_PRESS1; // Back to top

break;

default:

e\_mystate = STATE\_WAIT\_FOR\_PRESS1;

}

}

return 0;

}

## Appendix D – Source code and Answer of Task 4

#### Source code

#include "pic24\_all.h"

#if \_\_PIC24HJ128GP502\_\_

#define R\_LED \_LATB15// MicroStick II definitions

#define CONFIG\_R\_LED() CONFIG\_RB15\_AS\_DIG\_OUTPUT()

#define G\_LED \_LATB14

#define CONFIG\_G\_LED() CONFIG\_RB14\_AS\_DIG\_OUTPUT()

#define B\_LED \_LATB13

#define CONFIG\_B\_LED() CONFIG\_RB13\_AS\_DIG\_OUTPUT()

#define SW1 \_RB12

#define SW2 \_RB11

#endif

//Figure (4.1)

//Figure (4.2)

//6

inline void CONFIG\_SW1() {

CONFIG\_RB12\_AS\_DIG\_INPUT(); // use RB13 for switch input

ENABLE\_RB12\_PULLUP(); // enable the pullup

}

inline void CONFIG\_SW2() {

CONFIG\_RB11\_AS\_DIG\_INPUT(); // use RB13 for switch input

ENABLE\_RB11\_PULLUP(); // enable the pullup

}

inline char bin2gray(char binNum)

{

char grayNum = binNum;

grayNum = grayNum >> 1;

return (grayNum ^ binNum);

}

int main(void) {

char binNum, grayNum;

CONFIG\_R\_LED();

CONFIG\_G\_LED();

CONFIG\_B\_LED();

CONFIG\_SW1();

CONFIG\_SW2();

R\_LED=0;

G\_LED=0;

B\_LED=0;

while (1) {

//7

binNum = 0x0;

grayNum = 0x0;

if ((SW1 == 0) && (SW2 == 0)) {R\_LED = G\_LED = B\_LED = 0;}

while ((SW1 == 0) && (SW2 == 0)) //SW1 & SW2 pressed

{

R\_LED = !R\_LED; // Toggle LED1

G\_LED = !G\_LED;

B\_LED = !B\_LED;

DELAY\_MS(10); // Delay 100ms

}

while ((SW1 == 1) && (SW2 == 1))

{

R\_LED = 1;

G\_LED = 1;

B\_LED = 1;

DELAY\_MS(15);

}

while ((SW1 == 1) && (SW2 == 0))

{

if (binNum == 8) {binNum = 0x0;}

if ((binNum & 1) == 1) {R\_LED = 1;}

else R\_LED = 0;

if ((binNum & 2) == 2) {G\_LED = 1;}

else G\_LED = 0;

if ((binNum & 4) == 4) {B\_LED = 1;}

else B\_LED = 0;

binNum++;

DELAY\_MS(10);

}

//8

while ((SW1 == 0) && (SW2 == 1))

{

if (grayNum == 8) {grayNum = 0x0;}

if ((bin2gray(grayNum)&1)==1) {R\_LED = 1;}

else R\_LED = 0;

if ((bin2gray(grayNum)&2)==2) {G\_LED = 1;}

else G\_LED = 0;

if ((bin2gray(grayNum)&4)==4) {B\_LED = 1;}

else B\_LED = 0;

grayNum++;

DELAY\_MS(10);

}

}

return 0;

}

#### Answers

Question 1. What are the advantages of binary codes and gray code respectively?

Answer:

Advantage of grey code: The difference between two adjacent value is always 1 bit.

Advantage of binary code: Binary code can use in truth tables.

Question 2. Can we convert a gray code value back to binary code? If the answer is yes, please describe one possible method. Otherwise please explain the reason.

Answer: The last digit of gray code is same as the last digit of original binary code. Therefore, we can convert the gray code back to binary code.

Let’s suppose the three-digit gray code as g3 g2 g1 and three-digit binary code as b3 b2 b1.

By the conversion from binary code to gray code, g3 = 0 XOR b3, g2 = b2 XOR b3, g1 = b2 XOR b1.

If b3 = 0, then g3 = 0. If b3 = 1, then g3 = 1. So, b3 = g3.

Then we can figure out, b2 = b3 XOR g2 and b1 = b2 XOR g1.

In general, to convert gray code to binary code, we can have

b3 = g3,

b2 = b3 XOR g2,

b1 = b2 XOR g1.