**Basic and Finite State Machine LED and Switch**

**I/O Programming**

**6th Laboratory Report for ECE 383**

**Microcomputers**

**Submitted by**

**Shomari Thomas**

**11672867**

**Yichen Huang**

**11906882**

**The University of Alabama**

**Tuscaloosa, Alabama 35487**

**March 26, 2020**

**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**

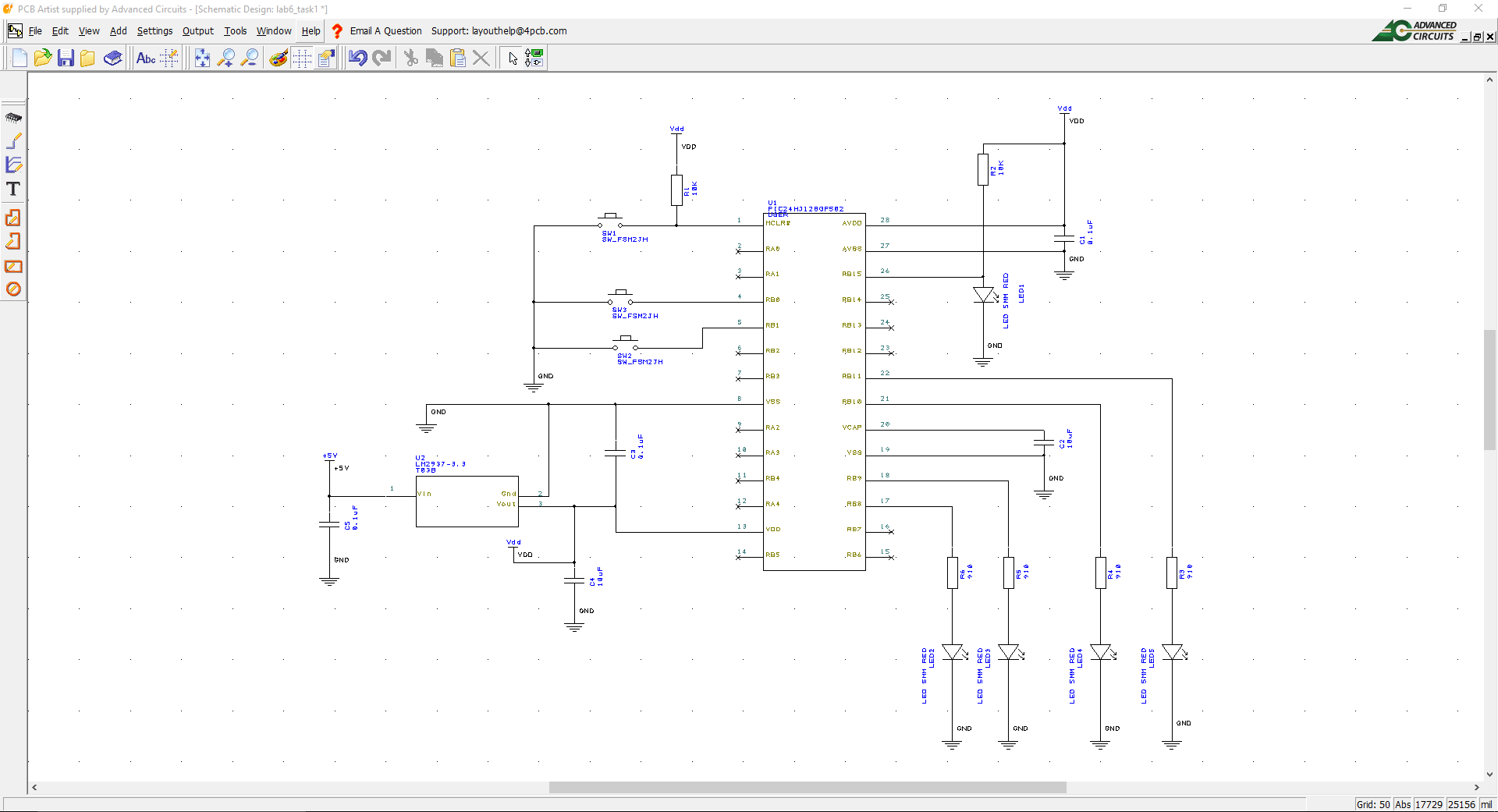
The objective of this lab is to become more familiar with PIC24 I/O and how to configure and use pins for input and output. Task one has us creating a system schematic for the PIC24 that includes resistors, switches and LEDs. Task two has us taking the system schematic created in task one and converting 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.

**Procedure/Results**

Task 1 - Expanding the PIC24 Reference System Schematic

Task one had us create a circuit in PCB Artist based on a given design.

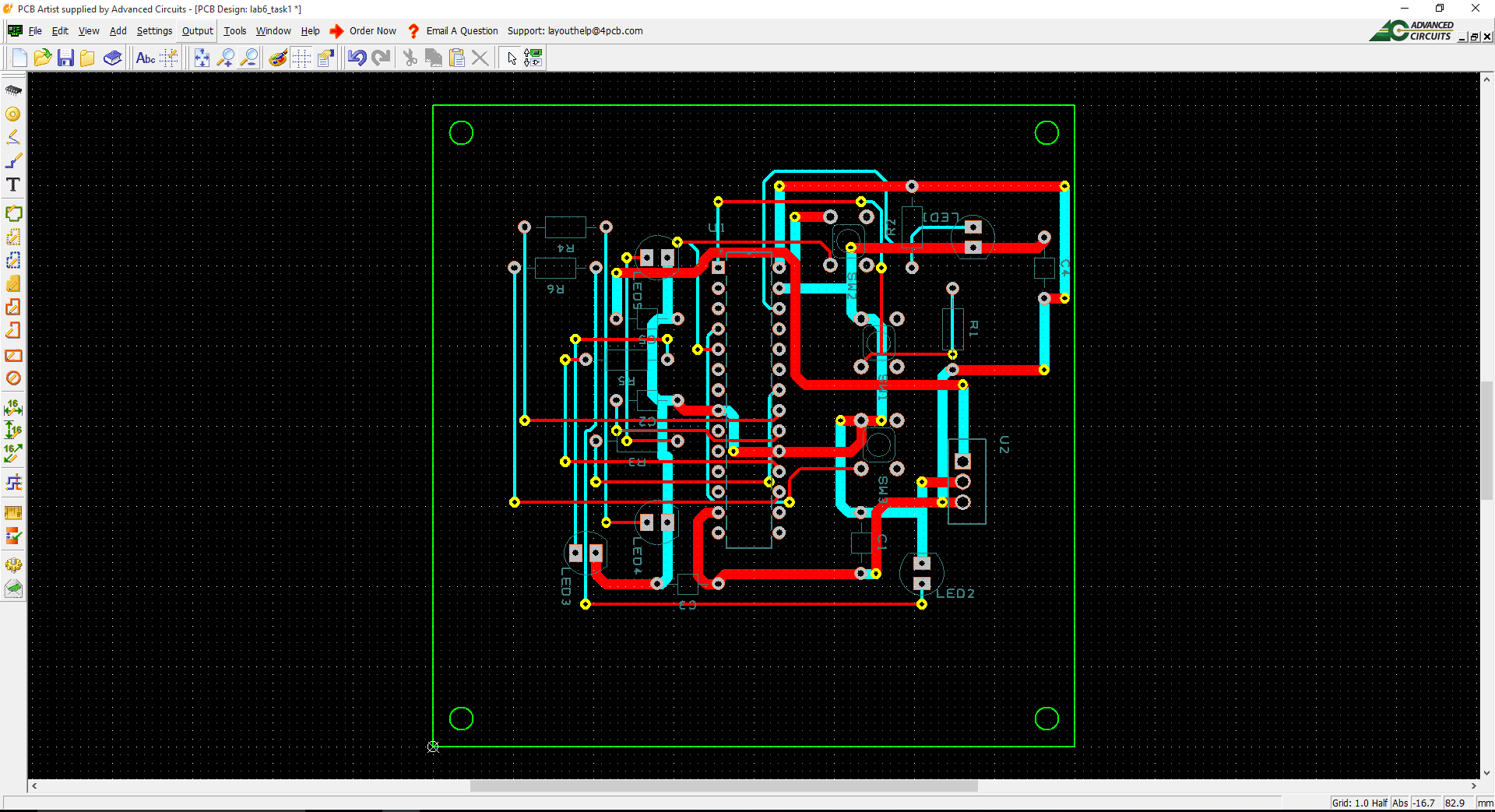
**Figure (3.1)**



Task 2 - Expanded 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, 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 as Bill of Materials CSV File and Components Position CSV File

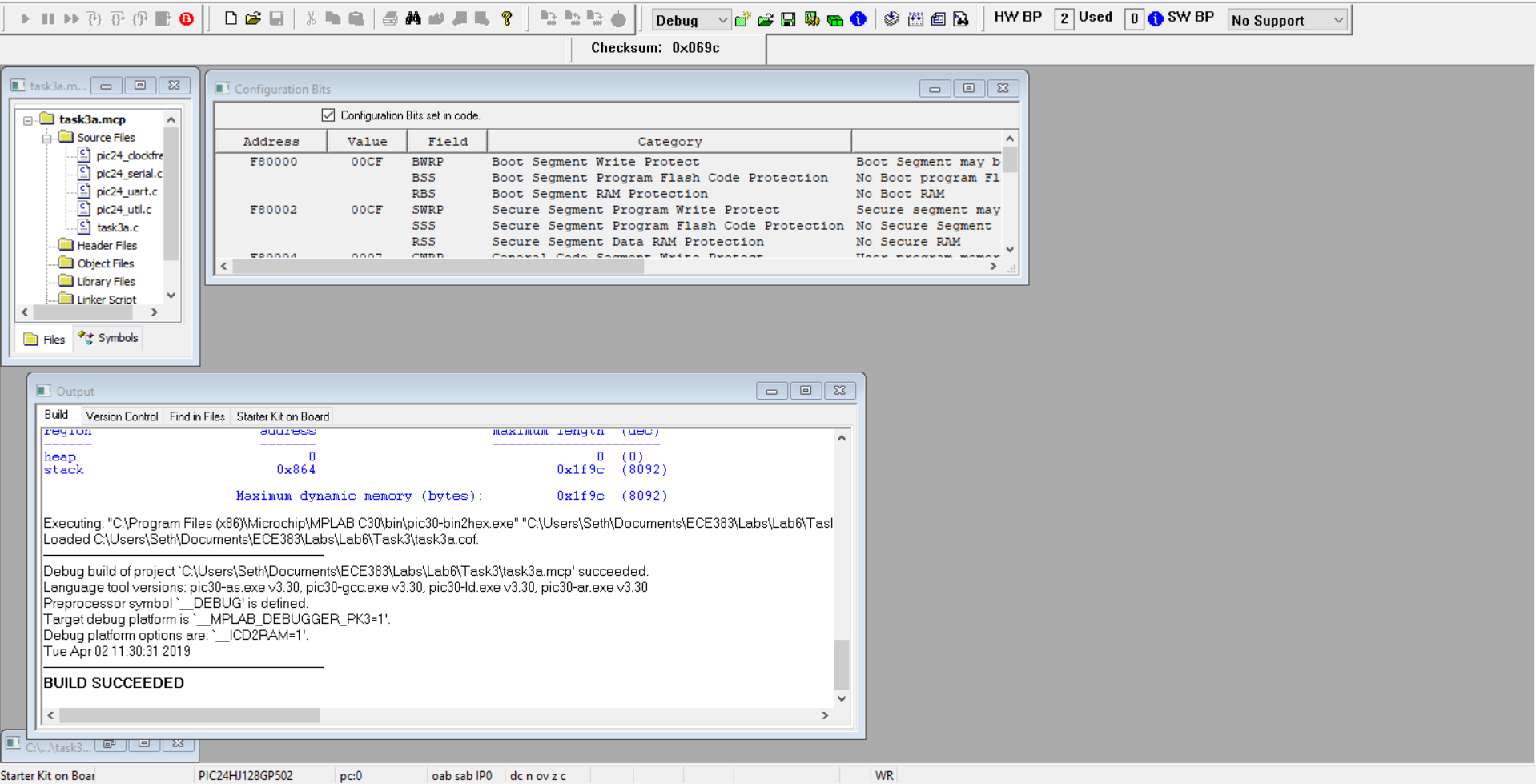
**Figure (2.1)**



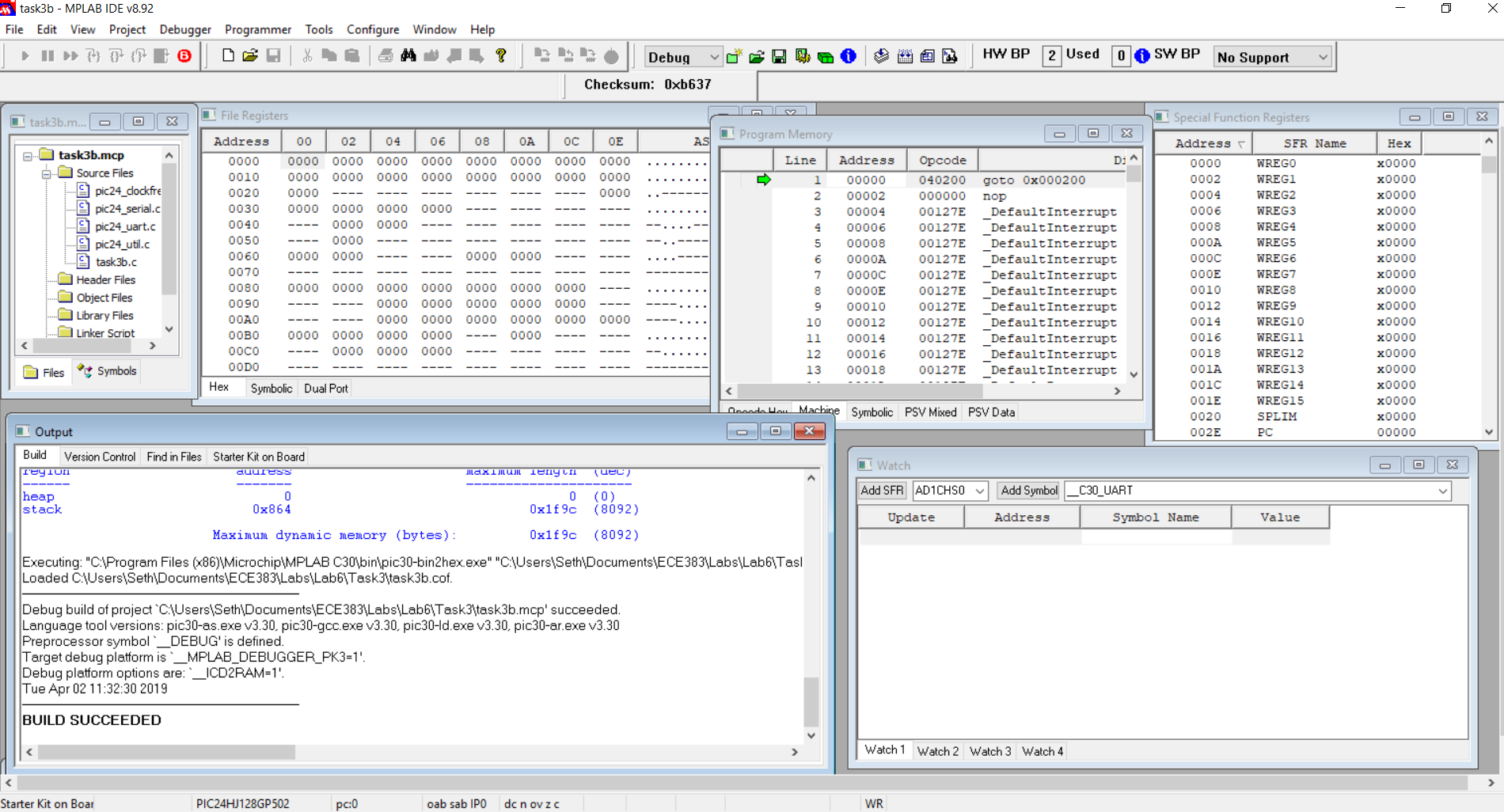
Task 3 - A Basic LED Problem

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.

**Figure (3.1)**

****

**Figure (3.2)**

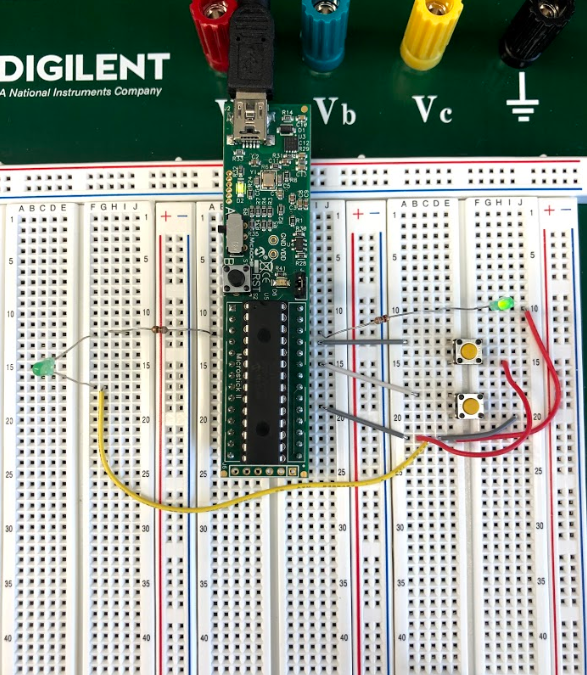
****

**Figure (3.3) - APPENDIX**

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.

**Figure (4.1)**

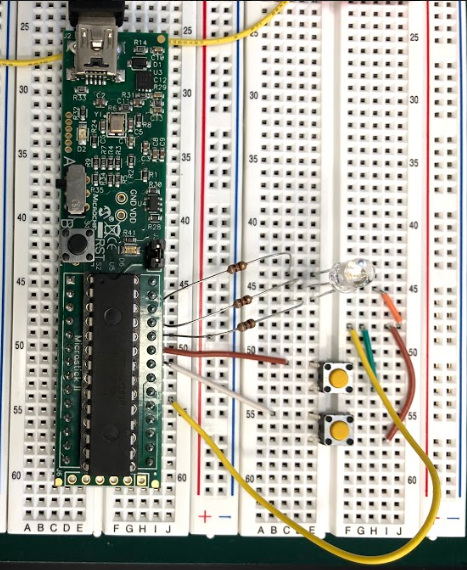
****

**Figure (4.2) - APPENDIX**

Task 5 - Variable Rotating LED

Task 5 implemented the use of a 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.

**Figure (5.1)**

****

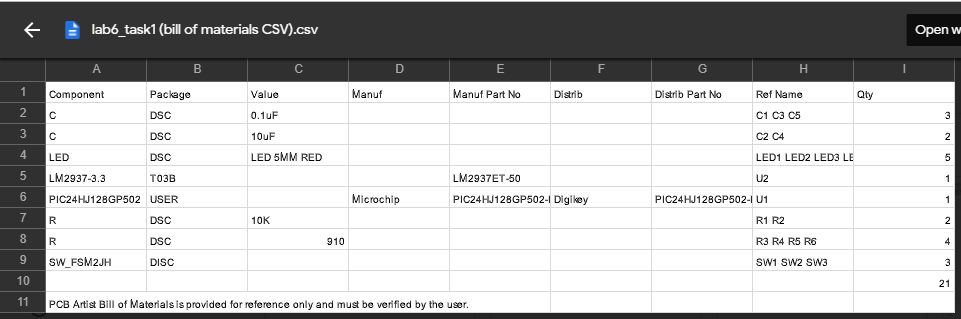
**Figure (5.2) - APPENDIX**

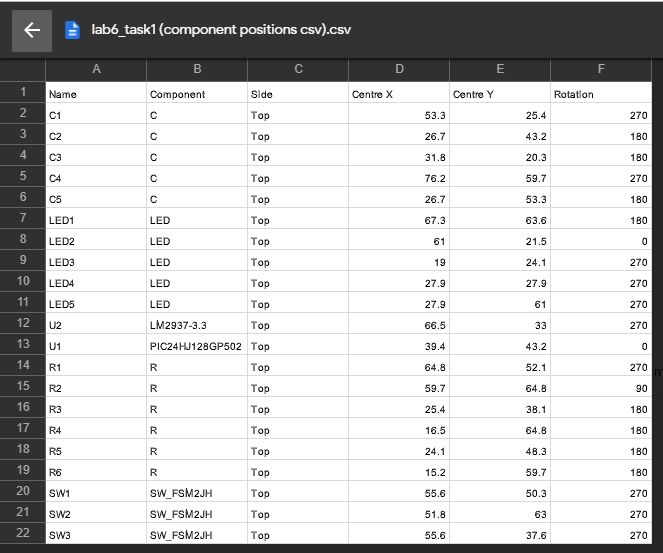
**Conclusion**

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.

**Appendix**

**Task 1: Bill of Materials CSV File**

****

**Task 1: Components Position CSV File**

**Task 3b: Figure (3.2)**

#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 a = 0; // variable used in for loop

int time = 0; // variable tracks time in ms

while (1) { // Infinite while loop

for(a=50; a>0; a--){ // for loop to last for 5 seconds, 50\*100ms

LED1 = !LED1; // Toggle LED1

time += 100; // Increase time increment by amount of time delay.

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

}

for(a=100; a>0; a--){ // for loop to last for 5 seconds, 100\*50ms

LED1 = !LED1; // Toggle LED1

DELAY\_MS(50); // Delay 100ms

}

}

return 0;

}

**Task 4: Figure (4.2)**

//Task4.c

#include "pic24\_all.h"

/\*

A program that uses a finite state machine approach for

toggling an LED whenever a pushbutton switch is pressed

and released. Demonstrates the use of debounce delays

when polling a switch input.

\*/

/// LED1

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

#define LED1 \_LATB15 // led1 state

// LED2

#define CONFIG\_LED2() CONFIG\_RA1\_AS\_DIG\_OUTPUT()

#define LED2 \_LATA1 //led2 state

/// Switch1 configuration

void CONFIG\_SW1() {

CONFIG\_RB12\_AS\_DIG\_INPUT(); // use RB12 for switch input

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

}

/// Switch2 configuration

void CONFIG\_SW2() {

CONFIG\_RB14\_AS\_DIG\_INPUT(); // use RB14 for switch input

ENABLE\_RB14\_PULLUP(); // enable the pullup

}

#define SW1 \_RB12 // switch state

#define SW1\_PRESSED() (SW1==0) // switch test

#define SW1\_RELEASED() (SW1==1) // switch test

#define SW2 \_RB14 // switch state

#define SW2\_PRESSED() (SW2==0) // switch test

#define SW2\_RELEASED() (SW2==1) // switch test

#define DEBOUNCE\_DLY 15 //Debounce delay

typedef enum {

STATE\_RESET = 0, //Reset state

STATE\_WAIT\_FOR\_PRESS1, //First wait for press state

STATE\_WAIT\_FOR\_PRESS2, //Second wait for press state

STATE\_WAIT\_FOR\_PRESS3, //Third wait for press state

STATE\_WAIT\_FOR\_PRESS4, //Fourth wait for press state

STATE\_WAIT\_FOR\_PRESS5, //Fifth wait for press state

STATE\_WAIT\_FOR\_PRESS6, //Sixth wait for press state

STATE\_WAIT\_FOR\_RELEASE1, //First wait for release state

STATE\_WAIT\_FOR\_RELEASE2, //Second wait for release state

STATE\_WAIT\_FOR\_RELEASE3, //Third wait for release state

STATE\_WAIT\_FOR\_RELEASE6, //Sixth wait for release state

STATE\_BLINKLED2\_1, //First state to Blink LED 2 and wait for release 4

STATE\_BLINKLED2\_2 //Second state to Blink LED 2 and wait for release 5

} STATE;

STATE e\_lastState = STATE\_RESET;

void printNewState (STATE e\_currentState) { //Function to print out state

if (e\_lastState != e\_currentState) {

switch (e\_currentState) {

case STATE\_WAIT\_FOR\_PRESS1:

outString("STATE\_WAIT\_FOR\_PRESS1\n");

break;

case STATE\_WAIT\_FOR\_PRESS2:

outString("STATE\_WAIT\_FOR\_PRESS2\n");

break;

case STATE\_WAIT\_FOR\_PRESS3:

outString("STATE\_WAIT\_FOR\_PRESS3\n");

break;

case STATE\_WAIT\_FOR\_PRESS4:

outString("STATE\_WAIT\_FOR\_PRESS4\n");

break;

case STATE\_WAIT\_FOR\_PRESS5:

outString("STATE\_WAIT\_FOR\_PRESS5\n");

break;

case STATE\_WAIT\_FOR\_PRESS6:

outString("STATE\_WAIT\_FOR\_PRESS6\n");

break;

case STATE\_WAIT\_FOR\_RELEASE1:

outString("STATE\_WAIT\_FOR\_RELEASE1\n");

break;

case STATE\_WAIT\_FOR\_RELEASE2:

outString("STATE\_WAIT\_FOR\_RELEASE2\n");

break;

case STATE\_WAIT\_FOR\_RELEASE3:

outString("STATE\_WAIT\_FOR\_RELEASE3\n");

break;

case STATE\_WAIT\_FOR\_RELEASE6:

outString("STATE\_WAIT\_FOR\_RELEASE6\n");

break;

case STATE\_BLINKLED2\_1:

outString("STATE\_BLINKLED2\_1\n");

break;

case STATE\_BLINKLED2\_2:

outString("STATE\_BLINKLED2\_2\n");

break;

default:

outString("Unexpected state\n");

break;

}

}

e\_lastState = e\_currentState;

}

int main (void) {

STATE e\_mystate;

/\*\* GPIO config \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CONFIG\_SW1(); //configure switch

CONFIG\_SW2(); //configure switch

CONFIG\_SW1(); //configure switch

CONFIG\_SW1(); //configure switch

DELAY\_US(1); //give pullups a little time

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

while (1) {

switch (e\_mystate) {

case STATE\_WAIT\_FOR\_PRESS1:

LED1 = 1;

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

break;

case STATE\_WAIT\_FOR\_RELEASE1:

if (SW1\_RELEASED()) {

LED1 = 0;

e\_mystate = STATE\_WAIT\_FOR\_PRESS2;

}

break;

case STATE\_WAIT\_FOR\_PRESS2:

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

break;

case STATE\_WAIT\_FOR\_RELEASE2:

if (SW1\_RELEASED()) {

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

//blink LED1 twice

int a = 0;

for (a = 0; a < 4; a++){

LED1 = !LED1;

DELAY\_MS(DEBOUNCE\_DLY);

}

LED1 = 1;

}

break;

case STATE\_WAIT\_FOR\_PRESS3:

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

break;

case STATE\_WAIT\_FOR\_RELEASE3:

if (SW1\_RELEASED()) {

if(SW2\_PRESSED())

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

else {

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

}

}

break;

case STATE\_WAIT\_FOR\_PRESS4:

if (SW1\_PRESSED()) e\_mystate = STATE\_BLINKLED2\_1;

break;

case STATE\_BLINKLED2\_1:

LED2 = !LED2; // Toggle LED1

DELAY\_MS(85); // Delay 100ms

if (SW1\_RELEASED()) {

LED2 = 0;

e\_mystate = STATE\_WAIT\_FOR\_PRESS5;

}

break;

case STATE\_WAIT\_FOR\_PRESS5:

LED2 = 0;

if (SW1\_PRESSED()) e\_mystate = STATE\_BLINKLED2\_2;

break;

case STATE\_BLINKLED2\_2:

LED2 = !LED2; // Toggle LED2

DELAY\_MS(85); // Delay 100ms

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

break;

case STATE\_WAIT\_FOR\_PRESS6:

LED2 = !LED2; // Toggle LED2

DELAY\_MS(35); // Delay 50ms

if (SW1\_PRESSED()) e\_mystate = STATE\_WAIT\_FOR\_RELEASE6;

break;

case STATE\_WAIT\_FOR\_RELEASE6:

LED2 = !LED2; // Toggle LED2

DELAY\_MS(35); // Delay 50ms

if (SW1\_RELEASED()) {

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

}

break;

default:

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

break;

}// end switch(e\_mystate)

DELAY\_MS(DEBOUNCE\_DLY); //

// doHeartbeat(); // ensure we are alive, not used in Microstick II

} // end while (1)

}

**Task 5: Figure (5.2)**

#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(100);

}

//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(100);

}

}

return 0;

}