**Lab 1 IDE and ANSI Display Report**

Shane Conaboy

Justin Yost

ECSE-4790 Microprocessor Systems

September 21, 2015

**Introduction**

This lab acted as an introductory assignment for programming and interfacing with the 8051 microcontroller. The first part used the VT100 Terminal Interface on the 8051 to create a simple program that took keyboard input from the user and had different responses based what key was pressed. The latter half of the lab was to perform basic input/output (I/O) with the microcontroller using potentiometers as input to control the lighting of six LEDs. The potentiometers were read in on the analog to digital converter (ADC), and the threshold positions of the potentiometer for when the LEDs would turn on or off were observed. An extension to the lab was added by wiring a 4-bit, 7-segment display to the output of the 8051 and using it to display the number lit LEDs.

**Materials and Methods**

For developing the code to interface with the VT100 Terminal and performing the desired outputs, the given Hello.c file was first examined and then modified. Using the printf command in conjunction with the VT100/ANSI escape sequences, the desired output was achieved. The process for generating the program was looking up the correct escape sequences for each part and incrementally implementing and testing each one. After figuring out how the escape sequences worked, this part of the lab just became following the instructions and was very straightforward.

To start the second half of the lab, getting signals sent to the LEDs was the first step. Again, the given Hello.c file was examined and modified. To send and receive signals, the appropriate ports needed to be set – port 2 for output and port 1 for input. To set a port for output, the PxMDOUT needed to be set to 0xFF, in this case P2MDOUT = 0xFF. For input, the PxMDOUT needed to be set to 0x00 and the PxMDIN to 0xFF, in this case P1MDOUT = 0x00 and P1MDIN = 0xFF. After the ports are set, the inputs and outputs were wired to the matching pins on the EVB. By setting logic high and low values to port 2, the corresponding LEDs could be turned on and off.

To test the input into the 8051, the single pole, double throw switches were wired to power, ground, and port 1 pins P1.0 through P1.5. By detecting either power or ground on the input, the corresponding LED could be toggled on or off by setting port 2 output equal to port 1 input and using the switches to toggle. After this was working properly, the potentiometer was a natural extension; it was wired to P1.0 on the 8051 and used to control LED1 in a similar way as the switches.

The extension to the lab had a 7-segment display wired to output pins P3.0 through P3.3. The program used the switches as inputs and counted the number of logic highs on pins P1.0 through P1.5. To count the logic highs, each bit was ANDed with a 1, and if the result was greater than 0, the count was incremented, as shown in the following block of code.

for(i=0; i<6; i++) { // Iterate through the 6 I/O we are monitoring

count = ((P1 & (1<<i)) > 0) ? count+1 : count;

}

The final circuit schematic and physical circuit are shown in Appendix Aand Appendix B respectively.

**Results/Analysis**

The VT100 Terminal program worked as intended. The user was able to type in a keyboard character and the terminal would display what that character was or that the character was not printable with the corresponding hex number associated with that character. The “not printable” was underlined and would blink. An example of what the screen looked like is shown in Figure 1 below.

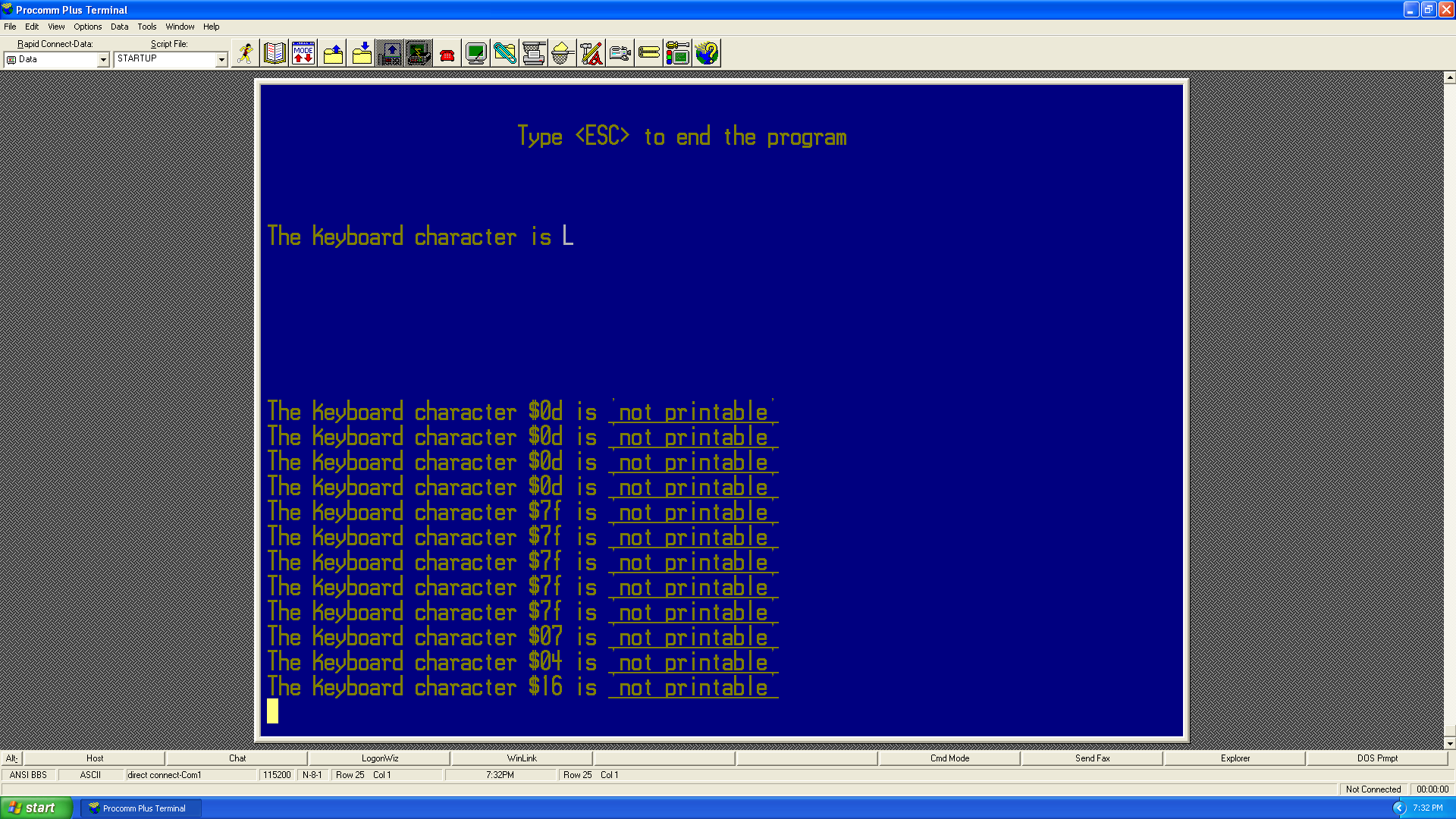
****

Figure : VT100 Terminal

The only problem that was encountered was when the “not printable” section of the terminal would scroll, part of the screen where the scrolling occurred would have a black background. The error was determined to be because after causing the words to blink, the screen attributes were reset, causing the background color to reset to black. Before the color was set to blue again, a newline character was printed, causing the black background to scroll with the lower half of the terminal. Placing the newline character after setting the background blue again easily solved the problem.

Controlling the LEDs with the switches and potentiometer both worked as expected. Having the switch in the *A* position turned on the LED and the *B* position turned off the LED. Using the potentiometer, it was determined that the 8051 microcontroller uses a Schmitt trigger on its analog inputs. Integrating the 7-segment display was also successful – it would display the number of LEDs lit regardless of which ones were on. A picture of the 7-segment display in action is shown in Figure 2 below.

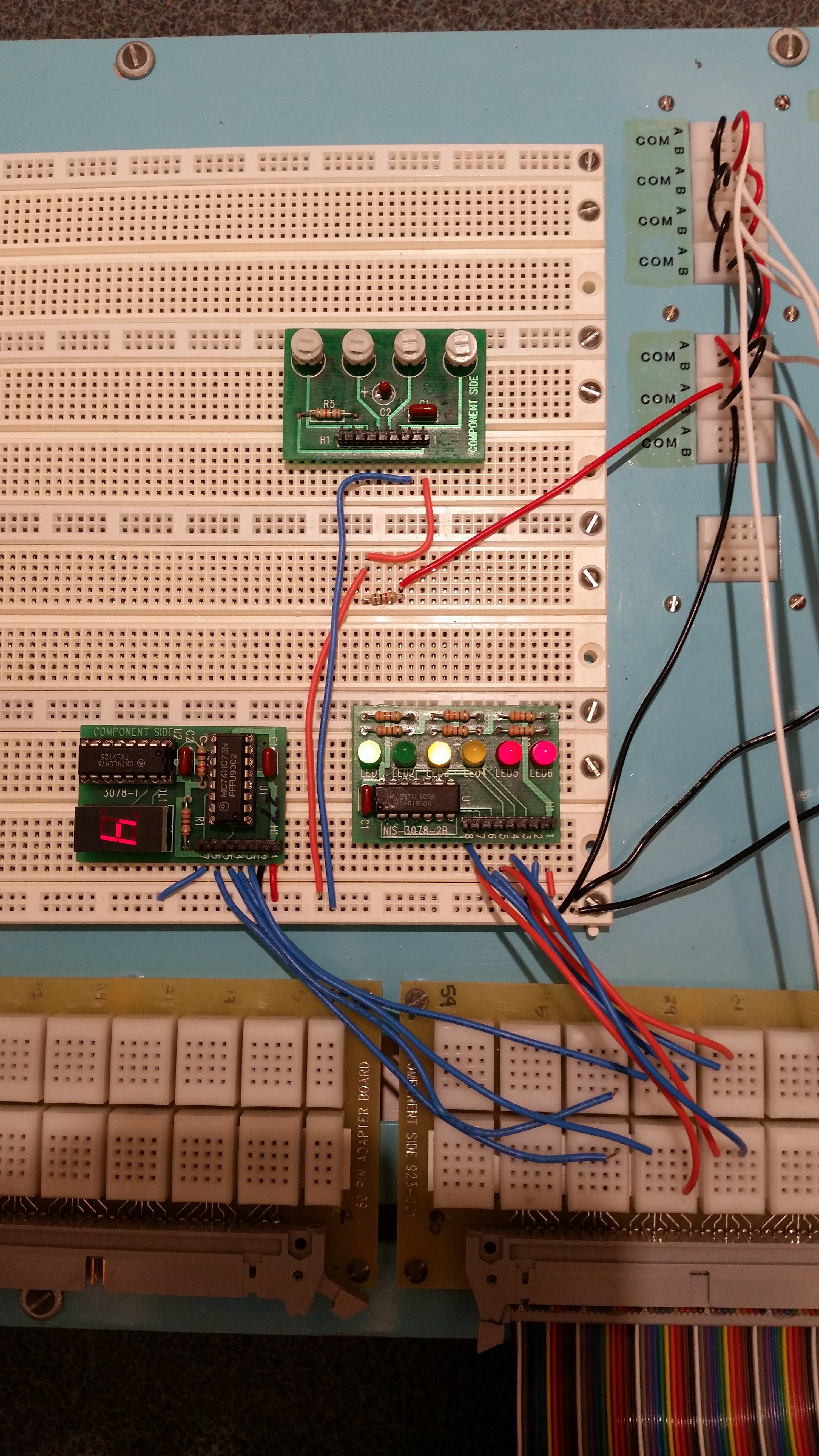
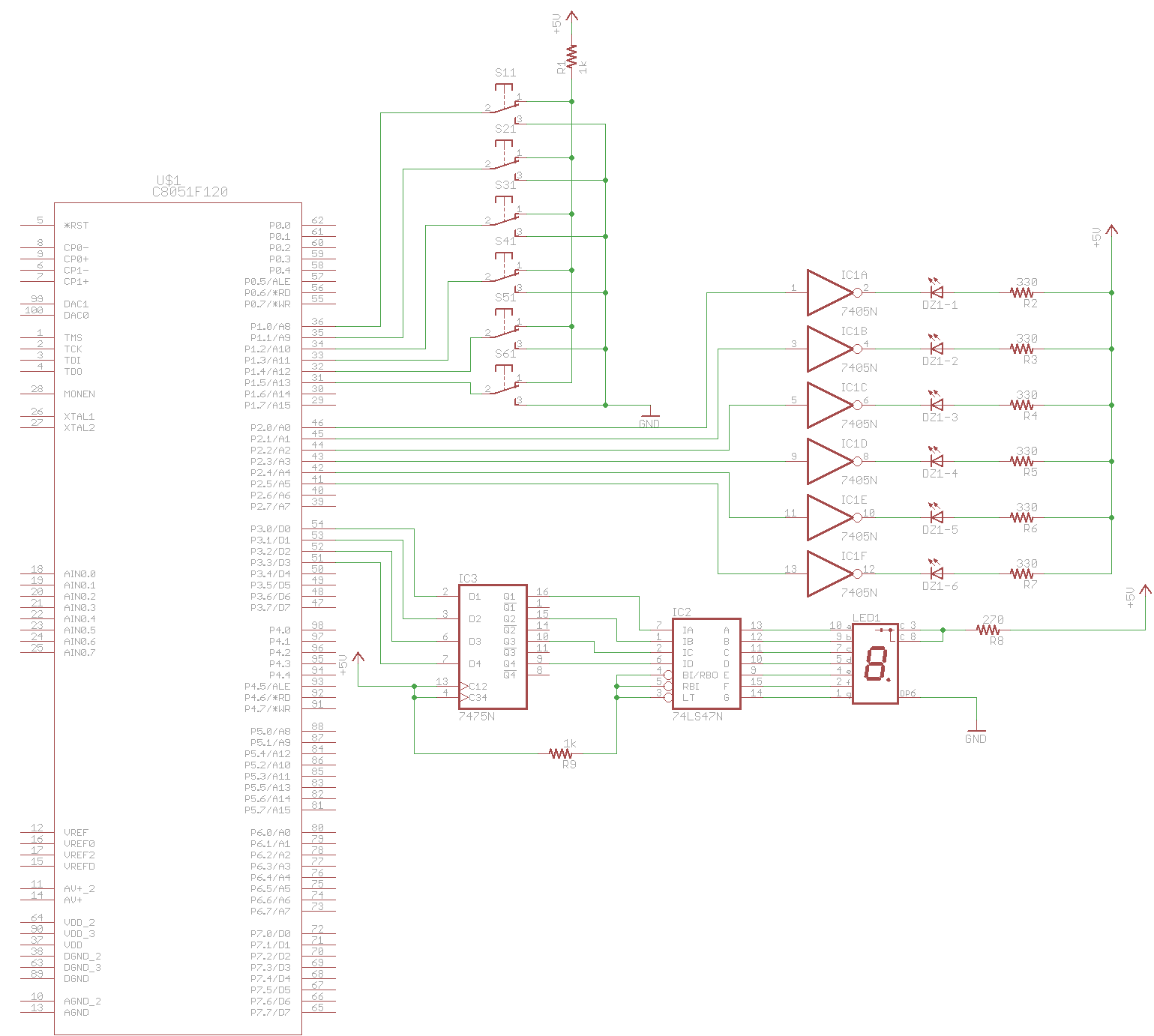


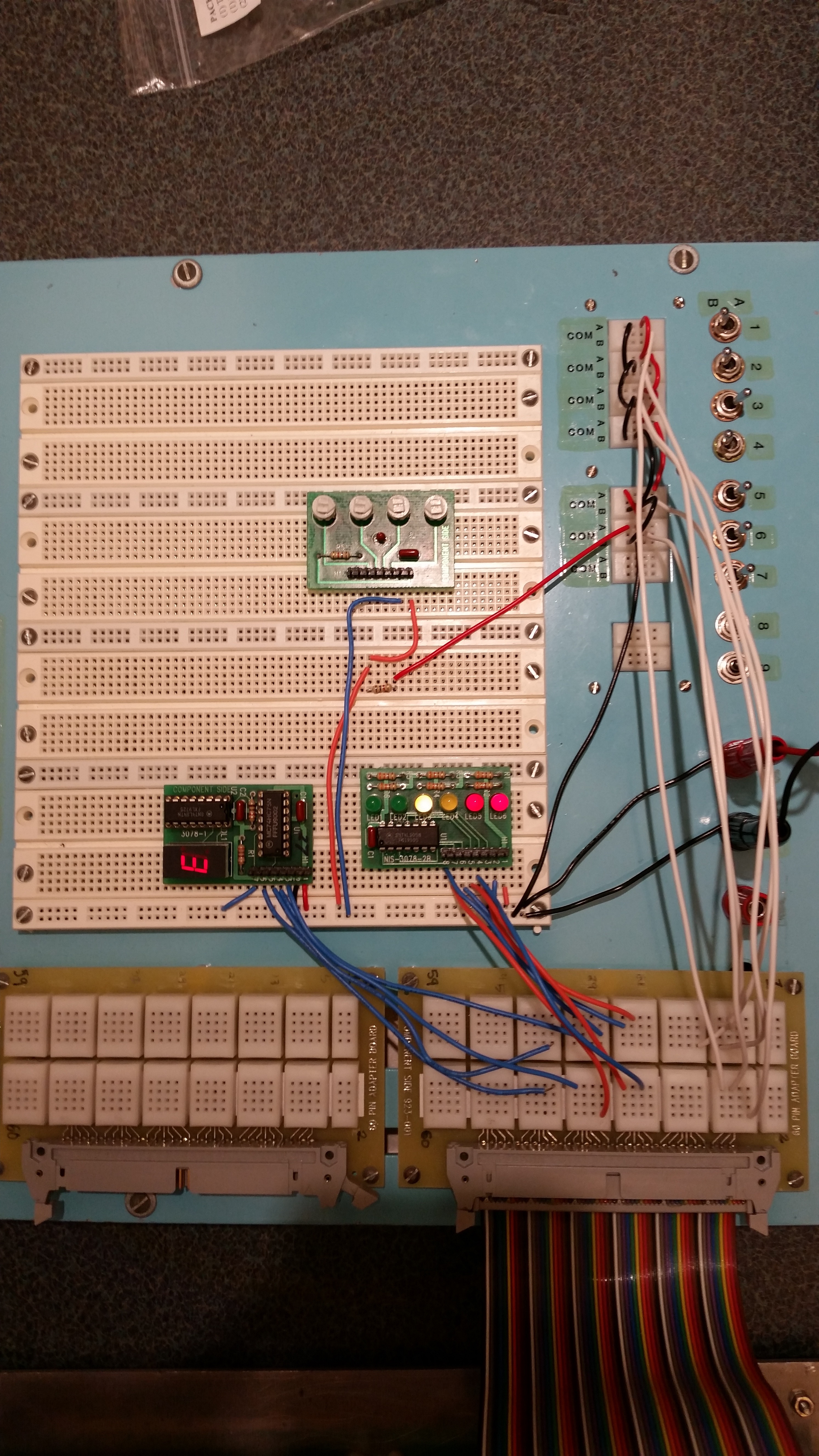
Figure : 7-Segment Display Counting LEDs Lit

**Conclusions**

By interfacing with a few different components of the 8051 microcontroller, this lab provided a good review of the basic functionality of the SiLabs IDE and what is needed to set up peripherals on the 8051. The VT100 Terminal touched on print commands, user input, and baud rate selection, while the LED section involved port and crossbar initialization, some bit logic, and becoming familiar with the EVB pins to wire everything correctly. Most importantly, however, the lab provided exposure to the 8051 technical manual, which will need to be referenced extensively in future labs.

**Appendix A: Circuit Schematic**



**Appendix B: Physical Circuit**

Appendix C: Code, Part 2

//------------------------------------------------------------------------------------

// lab1\_part2.c

//------------------------------------------------------------------------------------

//8051 Test program to demonstrate serial port I/O. This program writes a message on

//the console using the printf() function, and reads characters using the getchar()

//function. An ANSI escape sequence is used to clear the screen if a '2' is typed.

//A '1' repeats the message and the program responds to other input characters with

//an appropriate message.

//------------------------------------------------------------------------------------

// Includes

//------------------------------------------------------------------------------------

#include <c8051f120.h>

#include <stdio.h>

#include "putget.h"

//------------------------------------------------------------------------------------

// Global Constants

//------------------------------------------------------------------------------------

#define EXTCLK 22118400 // External oscillator frequency in Hz

#define SYSCLK 49766400 // Output of PLL derived from (EXTCLK \* 9/4)

#define BAUDRATE 115200 // UART baud rate in bps

//------------------------------------------------------------------------------------

// Function Prototypes

//------------------------------------------------------------------------------------

void main(void);

void SYSCLK\_INIT(void);

void PORT\_INIT(void);

void UART0\_INIT(void);

//------------------------------------------------------------------------------------

// MAIN Routine

//------------------------------------------------------------------------------------

void main(void)

{

char choice;

WDTCN = 0xDE; // Disable the watchdog timer

WDTCN = 0xAD;

PORT\_INIT(); // Initialize the Crossbar and GPIO

SYSCLK\_INIT(); // Initialize the oscillator

UART0\_INIT(); // Initialize UART0

SFRPAGE = UART0\_PAGE; // Direct output to UART0

printf("\033[2J"); // Erase screen & move cursor to home position

printf("\033[33m"); // Use Yellow Text

printf("\033[44m"); // Use a Blue background for the terminal

printf("\033[13;0H"); // Set the cursor position to the lower half of the terminal

printf("\033[s"); // Save the cursor position to return to later

printf("\033[13;25r"); // Apply scrolling to lower half of terminal

printf("\033[2;23H"); // Center help text

printf("Type <ESC> to end the program\n");

while(1)

{

choice=getchar(); // Read input character to process

if(choice == 0x1b) // Escape character

return;

if(choice > 126 || choice < 32) { // Characters in this range are not printable

printf("\033[u"); // Restore the cursor position

printf("\033[5mThe keyboard character $%02x is \033[4m'not printable'\033[0m", choice); // User message with blink and underline

printf("\033[33m"); // Yellow text

printf("\033[44m\n\r"); // Blue background

printf("\033[s"); // Save the cursor position

putchar(0x07); // Beep

}

else {

printf("\033[6;0H"); // Set cursor to line 6

printf("The keyboard character is \033[37m%c\033[33m\r", choice); // Printable character is output overwriting previous value

}

}

}

//------------------------------------------------------------------------------------

// SYSCLK\_Init

//------------------------------------------------------------------------------------

//

// Initialize the system clock to use a 22.1184MHz crystal as its clock source

//

void SYSCLK\_INIT(void)

{

int i;

char SFRPAGE\_SAVE = SFRPAGE; // Save Current SFR page

SFRPAGE = CONFIG\_PAGE;

OSCXCN = 0x67; // Start ext osc with 22.1184MHz crystal

for(i=0; i < 256; i++); // Wait for the oscillator to start up

while(!(OSCXCN & 0x80));

CLKSEL = 0x01;

OSCICN = 0x00;

SFRPAGE = CONFIG\_PAGE;

PLL0CN = 0x04;

SFRPAGE = LEGACY\_PAGE;

FLSCL = 0x10;

SFRPAGE = CONFIG\_PAGE;

PLL0CN |= 0x01;

PLL0DIV = 0x04;

PLL0FLT = 0x01;

PLL0MUL = 0x09;

for(i=0; i < 256; i++);

PLL0CN |= 0x02;

while(!(PLL0CN & 0x10));

CLKSEL = 0x02;

SFRPAGE = SFRPAGE\_SAVE; // Restore SFR page

}

//------------------------------------------------------------------------------------

// PORT\_Init

//------------------------------------------------------------------------------------

//

// Configure the Crossbar and GPIO ports

//

void PORT\_INIT(void)

{

char SFRPAGE\_SAVE = SFRPAGE; // Save Current SFR page

SFRPAGE = CONFIG\_PAGE;

XBR0 = 0x04; // Enable UART0

XBR1 = 0x00;

XBR2 = 0x40; // Enable Crossbar and weak pull-up

P0MDOUT |= 0x01; // Set TX0 pin to push-pull

P1MDOUT = 0x00; // Port 1 open drain

P1MDIN = 0xFF; // Port 1 digital input

P2MDOUT = 0xFF;

P3MDOUT = 0xFF;

SFRPAGE = SFRPAGE\_SAVE; // Restore SFR page

}

//------------------------------------------------------------------------------------

// UART0\_Init

//------------------------------------------------------------------------------------

//

// Configure the UART0 using Timer1, for <baudrate> and 8-N-1

//

void UART0\_INIT(void)

{

char SFRPAGE\_SAVE = SFRPAGE; // Save Current SFR page

SFRPAGE = UART0\_PAGE;

SCON0 = 0x50; // Mode 1, 8-bit UART, enable RX

SSTA0 = 0x10; // SMOD0 = 1

SFRPAGE = TIMER01\_PAGE;

TMOD &= ~0xF0;

TMOD |= 0x20; // Timer1, Mode 2, 8-bit reload

TH1 = -(SYSCLK/BAUDRATE/16); // Set Timer1 reload baudrate value T1 Hi Byte

CKCON |= 0x10; // Timer1 uses SYSCLK as time base

TL1 = TH1;

TR1 = 1; // Start Timer1

SFRPAGE = UART0\_PAGE;

TI0 = 1; // Indicate TX0 ready

SFRPAGE = SFRPAGE\_SAVE; // Restore SFR page

}

Appendix D: Coe, Part 3

//------------------------------------------------------------------------------------

// lab1\_part3.c

//------------------------------------------------------------------------------------

//8051 Test program to demonstrate port I/O. This program reads the inputs on port 1

//and sets them as outputs on port 2 to match port 1. Additionally, the number of

//input pins that are high is counted and this number is output on the first four pins

//of port 3 to be displayed by a 7 segment display.

//------------------------------------------------------------------------------------

// Includes

//------------------------------------------------------------------------------------

#include <c8051f120.h>

#include <stdio.h>

#include "putget.h"

//------------------------------------------------------------------------------------

// Global Constants

//------------------------------------------------------------------------------------

#define EXTCLK 22118400 // External oscillator frequency in Hz

#define SYSCLK 49766400 // Output of PLL derived from (EXTCLK \* 9/4)

#define BAUDRATE 115200 // UART baud rate in bps

//------------------------------------------------------------------------------------

// Function Prototypes

//------------------------------------------------------------------------------------

void main(void);

void SYSCLK\_INIT(void);

void PORT\_INIT(void);

void UART0\_INIT(void);

//------------------------------------------------------------------------------------

// MAIN Routine

//------------------------------------------------------------------------------------

void main(void)

{

char i;

char count;

WDTCN = 0xDE; // Disable the watchdog timer

WDTCN = 0xAD;

PORT\_INIT(); // Initialize the Crossbar and GPIO

SYSCLK\_INIT(); // Initialize the oscillator

UART0\_INIT(); // Initialize UART0

SFRPAGE = UART0\_PAGE; // Direct output to UART0

while(1)

{

P2 = P1; // Set the outputs on port 2 to match the inputs on port 1

count = 0;

for(i=0; i<6; i++) // Iterate through the 6 I/O we are monitoring

{

count = ((P1 & (1<<i)) > 0) ? count+1:count; // Increment count if bit i on P1 is HIGH

}

P3 = count; // Port 3 is set to the value of count to be displayed by the 7 segment display

}

}

//------------------------------------------------------------------------------------

// SYSCLK\_Init

//------------------------------------------------------------------------------------

//

// Initialize the system clock to use a 22.1184MHz crystal as its clock source

//

void SYSCLK\_INIT(void)

{

int i;

char SFRPAGE\_SAVE = SFRPAGE; // Save Current SFR page

SFRPAGE = CONFIG\_PAGE;

OSCXCN = 0x67; // Start ext osc with 22.1184MHz crystal

for(i=0; i < 256; i++); // Wait for the oscillator to start up

while(!(OSCXCN & 0x80));

CLKSEL = 0x01;

OSCICN = 0x00;

SFRPAGE = CONFIG\_PAGE;

PLL0CN = 0x04;

SFRPAGE = LEGACY\_PAGE;

FLSCL = 0x10;

SFRPAGE = CONFIG\_PAGE;

PLL0CN |= 0x01;

PLL0DIV = 0x04;

PLL0FLT = 0x01;

PLL0MUL = 0x09;

for(i=0; i < 256; i++);

PLL0CN |= 0x02;

while(!(PLL0CN & 0x10));

CLKSEL = 0x02;

SFRPAGE = SFRPAGE\_SAVE; // Restore SFR page

}

//------------------------------------------------------------------------------------

// PORT\_Init

//------------------------------------------------------------------------------------

//

// Configure the Crossbar and GPIO ports

//

void PORT\_INIT(void)

{

char SFRPAGE\_SAVE = SFRPAGE; // Save Current SFR page

SFRPAGE = CONFIG\_PAGE;

XBR0 = 0x04; // Enable UART0

XBR1 = 0x00;

XBR2 = 0x40; // Enable Crossbar and weak pull-up

P0MDOUT |= 0x01; // Set TX0 pin to push-pull

P1MDOUT = 0x00; // Port 1 open drain

P1MDIN = 0xFF; // Port 1 digital input

P2MDOUT = 0xFF;

P3MDOUT = 0xFF;

SFRPAGE = SFRPAGE\_SAVE; // Restore SFR page

}

//------------------------------------------------------------------------------------

// UART0\_Init

//------------------------------------------------------------------------------------

//

// Configure the UART0 using Timer1, for <baudrate> and 8-N-1

//

void UART0\_INIT(void)

{

char SFRPAGE\_SAVE = SFRPAGE; // Save Current SFR page

SFRPAGE = UART0\_PAGE;

SCON0 = 0x50; // Mode 1, 8-bit UART, enable RX

SSTA0 = 0x10; // SMOD0 = 1

SFRPAGE = TIMER01\_PAGE;

TMOD &= ~0xF0;

TMOD |= 0x20; // Timer1, Mode 2, 8-bit reload

TH1 = -(SYSCLK/BAUDRATE/16); // Set Timer1 reload baudrate value T1 Hi Byte

CKCON |= 0x10; // Timer1 uses SYSCLK as time base

TL1 = TH1;

TR1 = 1; // Start Timer1

SFRPAGE = UART0\_PAGE;

TI0 = 1; // Indicate TX0 ready

SFRPAGE = SFRPAGE\_SAVE; // Restore SFR page

}