**IDE and ANSI Display**

***C8051 Microcontroller***

Jeffrey Pistacchio

Christopher Pybus

Microprocessor Systems

15 September 2016

# Introduction

The purpose of this exercise was to familiarize the team with the C8051 microcontroller unit – but more specifically, to refresh the team’s knowledge with registers, built in hardware, display functions, and input/output. The goal was to create a user interface using American National Standard Institute Codes (ANSI Codes) that would read into a terminal on a computer in order to display information to the user. ANSI codes were also used to change text and terminal appearance. Additionally, input/output functionality was to be used to enable and disable LEDs on a protoboard given specific input from a potentiometer on said protoboard.

# Methods and Procedure

The team split the exercise into two separate goals, each with its own program in order to separate and compartmentalize tasks. Goal 1 was to create a program capable of interfacing with the terminal with ANSI, read input from the user, and display output depending on input. Goal 2 was to build a protoboard with LEDs and potentiometers, wire those to the C8051, and use input/output methods to control those physical devices. Each goal is explained below in more detail.

## Goal 1

As stated above, Goal 1 dealt mainly with input and output from the computer terminal using ANSI. The objective was the take input from the user using the ANSI terminal, test to see if it was a printable character, then print it to the terminal if it was and display an error message if it was not. The team split up this goal into smaller tasks that each contributed to the overall goal and in the end, to the overall project. Each task is listed below with a short description of how the team managed to accomplish it:

1. Change the color of the background:  
   In order to do this, the team had to use what is called an ANSI escape sequence. These sequences can be sent to the computer terminal using normal printf() statements, but they contain information that the terminal will use to change something about the way it formats text, while not actually displaying the phrase that was sent. It is almost like a command sequence being sent to the terminal, but again, it uses the regular printf() statement. In this case, the object was to change the background of the screen, so the ANSI statement used was <ESC>[44;m or as it appears in the code printf(“\033[44m”); A full list of escape sequences can be found in Appendix A.
2. Change the color of the text:  
   This task was also handled using an escape sequence. In this case, the sequence was <ESC>[33;m or as it appears in the code printf(“\033[33m”); Again, a full list of escape sequences, their codes, and their meanings can be found in Appendix A.
3. Set a scrollable area:  
   This task involved setting a portion of the screen (specifically lines 12-25) to scrollable, so that whenever a new line command \n is issued, it would scroll the lines in the specific section if there was no more room for the new line. This code was slightly more complicated as there are two situations that arise:
   1. A new line is needed and there is plenty of space in the scrollable field
   2. A new line is needed but there is no more space in the scrollable field

In order to solve this problem and distinguish between the two situations, a variable of the current line number was kept so the program would know what line was to be used next. Every time a new line was to be added, the program would check the variable to find out what line it is on currently. If the line number was under 25, that meant that there was room for another line as 25 is the max and the number has not reached 25 yet. The program would simply add the line of text and increment the variable. If the line number was 25 (which is the last line in the terminal), it would scroll the entire area to make room for another line of text and remove the top line of text from the scrollable area.

1. Move the cursor around the screen:

This was accomplished by using another escape sequence. This sequence takes in two numbers, the first is a row and the second is a column, which together locate a specific point on the terminal. The sequence is <ESC>[xx;yyH, where xx and yy represent the row and column numbers. It appears in the code as printf(“\033[xx;yyH”);

1. Test input characters for printability:  
   The team realized that this task must be something that many C-programmers must do and as such, there must already be a solution in place for it. After researching the problem, the team discovered that there was, in fact, a built in method in C that tested char variables to see if they were printable characters. The method used was isprint(char); which simply took in one char and returned a Boolean value. The team used to the validate all user input. Another approach would have been to read in the actual ASCII values of the inputted character and use if-statements to test printability.
2. Underline and blinking text:

This was another escape sequence that told the ANSI terminal to underline whatever text followed the sequence and make it blink every second. The sequence is <ESC>[4;5m, which appears in the code as printf(“\033[4;5m”);

The final output of Goal 1 resulted in a program that appeared similar to Figure 1 below. Full code for Goal 1 can be seen in Appendix B.

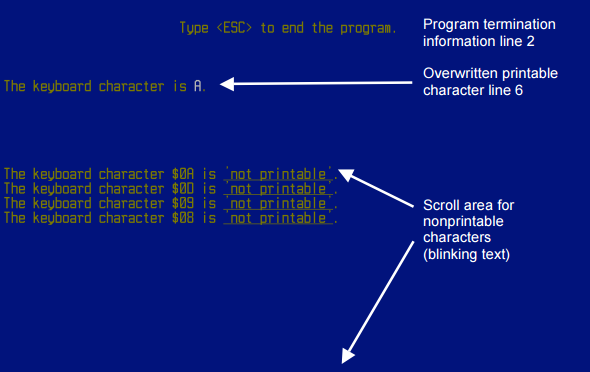


Figure 1: display from Goal 1 of the project.

## Goal 2

The second portion of the overall project was to use the C8051’s input/output pins to control LEDs and read the resistance of a potentiometer. The objective was to wire up four LEDs to the output pins of the 8051, and four potentiometers to the input pins, then read the resistance of the potentiometer in the code and output an on or off signal to the LEDs from the code. The team split up the tasks into smaller, more manageable tasks in order to accomplish the goal efficiently. These smaller tasks are listed below, each with a short explanation:

1. Determine SFR paging:  
   SFR paging is a technique that SiLabs has implemented in order to increase the number of registers that the user can access and control, but at the same time, keep the same number of bytes in the 8051 architecture. The concept is that there are multiple “pages” of registers, and the code must first set the “page” using a command, and then all of the registers on that page are accessible. The catch is that all the other pages are inaccessible. For Goal 2, the team had to ensure that they were always referencing the correct page, because if they weren’t, the program would not function correctly.
2. Setup output using Port 2 on the crossbar:  
   The C8051 has two settings when dealing with input/output: pushpull and open-drain. Which one a port is set to determines whether the port will be an input or an output. For Port 2, pins 0-3, the team wanted to use as outputs to the LEDs. This way, the team could send a high signal along one of the Port 2 pins and turn one of the LEDs on. In order to setup pins 0-3 on port 2 for pushpull, the following line of code is used P2MDOUT |= 0x0F; The 0F hex represents 0000 1111, which sets the first 4 bits on the P2MDOUT register to high, making them pushpull, and readying them for output to the LEDs.
3. Setup input using Port 1 on the crossbar:  
   The team decided to use Port 1, pins 0 through 3, as the input from the potentiometers on the protoboard. The setup is done in two steps. First, the pins are set to open drain. Next, those pins that were set to open drain are pulsed with a high signal to activate them. The code for this is below:  
   P1MDOUT &= 0xF0;  
   P1 |= 0x0F;
4. Assign sbit variables to LEDs pins and potentiometer pins:  
   In order to simplify life, the team assigned each input pin and each output pin with an sbit so they could reference it by the variable instead of reading individual bits from the 8-bit Port. These were set with the following code:  
   \_\_sbit \_\_at (0x90) Pot0; //Input on Port 1, Pin 0 (12 on board)

\_\_sbit \_\_at (0x91) Pot1; //Input on Port 1, Pin 1 (13 on board)

\_\_sbit \_\_at (0x92) Pot2; //Input on Port 1, Pin 2 (10 on board)

\_\_sbit \_\_at (0x93) Pot3; //Input on Port 1, Pin 3 (11 on board)

\_\_sbit \_\_at (0xA0) LED0; //Output on port 2, Pin 0, (29 on board)

\_\_sbit \_\_at (0xA1) LED1; //Output on port 2, Pin 0, (30 on board)

\_\_sbit \_\_at (0xA2) LED2; //Output on port 2, Pin 0, (27 on board)

\_\_sbit \_\_at (0xA3) LED3; //Output on port 2, Pin 0, (28 on board)

1. Setup a control loop that checks input from the potentiometers:

The team needed a main control loop that would constantly check the values of the input from the potentiometers, and then change the status of the LEDs as necessary. This was done with a simple infinite while loop in the main function. Before entering the loop, the program will register all the values of the potentiometer. After entering the loop, it will constantly check the current value against the saved value. If the value has changed, it will update text on the screen, LED, and saved value accordingly.

A picture of the team’s protoboard with LEDs and potentiometers can be seen in figure 2 below. The code for Goal 2 can be found in Appendix C.

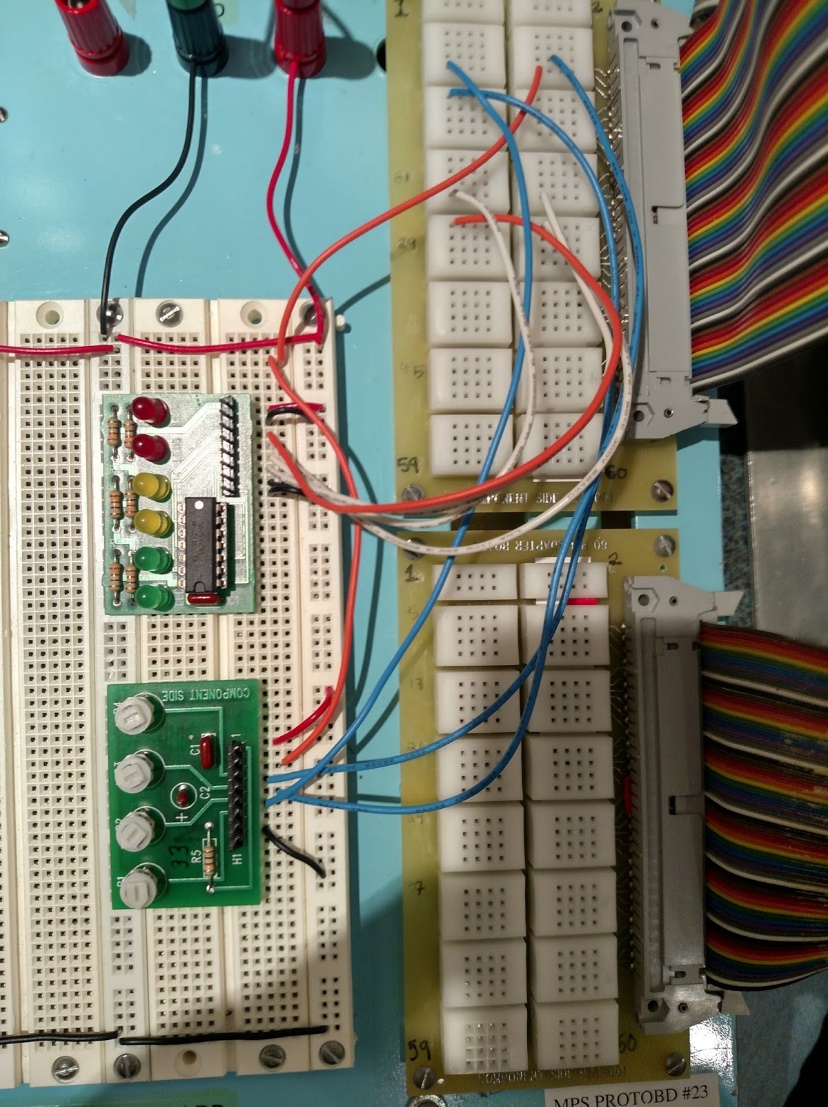


Figure 2: protoboard with LED displays and potentiometers

# Results and Analysis

The results for Goal 1 were better than expected. By using isprint() rather than checking ASCII values, the team was able to successfully check arrow keys for printability. This is normally a complicated task as arrow keys are formatted differently than any other key on the keyboard – they begin with an escape sequence which would normally quit the program. Table 1 below shows all keys on the keyboard and whether or not it was printable with the team’s program.

Table 1: Printability of Keys on Keyboard

*Note: “yes” signifies the key was printed after the user typed it, “no” signifies the key was not printed after the user typed it and the blinking error message was generated. No keys in the table produced unexpected output.*

|  |  |
| --- | --- |
| **Key** | **Print** |
| (space) | yes |
| ! | yes |
| " | yes |
| # | yes |
| $ | yes |
| % | yes |
| & | yes |
| ' | yes |
| ( | yes |
| ) | yes |
| \* | yes |
| + | yes |
| , | yes |
| - | yes |
| . | yes |
| / | yes |
| 0 | yes |
| 1 | yes |
| 2 | yes |
| 3 | yes |
| **Key** | **Print** |
| 4 | yes |
| 5 | yes |
| 6 | yes |
| 7 | yes |
| 8 | yes |
| 9 | yes |
| : | yes |
| ; | yes |
| < | yes |
| = | yes |
| > | yes |
| ? | yes |
| @ | yes |
| a/A | yes |
| b/B | yes |
| c/C | yes |
| d/D | yes |
| e/E | yes |
| f/F | yes |
| g/G | yes |
| **Key** | **Print** |
| h/H | yes |
| i/I | yes |
| j/J | yes |
| k/K | yes |
| l/L | yes |
| m/M | yes |
| n/N | yes |
| o/O | yes |
| p/P | yes |
| q/Q | yes |
| r/R | yes |
| s/S | yes |
| t/T | yes |
| u/U | yes |
| v/V | yes |
| w/W | yes |
| x/X | yes |
| y/Y | yes |
| z/Z | yes |
| [ | yes |
| **Key** | **Print** |
| \ | yes |
| ] | yes |
| ^ | yes |
| \_ | yes |
| ` | yes |
| (tab) | no |
| (control) | no |
| (alt) | no |
| (enter) | no |
| (back) | no |
| (insert) | no |
| (delete) | no |
| (home) | no |
| (end) | no |
| (p down) | no |
| (p up) | no |
| (l arrow) | no |
| (r arrow) | no |
| (u arrow) | no |
| (d arrow) | no |

The results for Goal 2 were on par with expectations. The code responded to changes in the input potentiometers as planned, and outputted signals to the LEDs as appropriate. Text display was output to the terminal to keep the user apprised of the status of all four LEDs. As a test, the team turned each potentiometer from low to high, and then back to low ten times and recorded the resulting LED change (on as potentiometer goes high, and then back off as potentiometer goes low). These results can be seen in Tables 2 through 5.

Table 2: Testing of potentiometer 1 wired to LED 1

|  |  |
| --- | --- |
| Attempt | LED Change |
| 1 | Yes |
| 2 | Yes |
| 3 | Yes |
| 4 | Yes |
| 5 | Yes |
| 6 | Yes |
| 7 | Yes |
| 8 | Yes |
| 9 | Yes |
| 10 | Yes |

Table 3: Testing of potentiometer 2 wired to LED 2

|  |  |
| --- | --- |
| Attempt | LED Change |
| 1 | Yes |
| 2 | Yes |
| 3 | Yes |
| 4 | Yes |
| 5 | Yes |
| 6 | Yes |
| 7 | Yes |
| 8 | Yes |
| 9 | Yes |
| 10 | Yes |

Table 4: Testing of potentiometer 3 wired to LED 3

|  |  |
| --- | --- |
| Attempt | LED Change |
| 1 | Yes |
| 2 | Yes |
| 3 | Yes |
| 4 | Yes |
| 5 | Yes |
| 6 | Yes |
| 7 | Yes |
| 8 | Yes |
| 9 | Yes |
| 10 | Yes |

Table 5: Testing of potentiometer 4 wired to LED 4

|  |  |
| --- | --- |
| Attempt | LED Change |
| 1 | Yes |
| 2 | Yes |
| 3 | Yes |
| 4 | Yes |
| 5 | Yes |
| 6 | Yes |
| 7 | Yes |
| 8 | Yes |
| 9 | Yes |
| 10 | Yes |

# Conclusion

The team successfully completed the exercise by making two separate programs that accomplished all the required goals. The team valued their idea of splitting the entire project into two separate goals, and then further splitting those goals into smaller tasks. Each task was simple enough that it could be tested individually, and then added to the code. This approach allowed the team to finish quickly, as it is easy to debug a simple task as opposed to an entire program. Had the team attempted to write the entire program at once, the amount of time needed to debug and stabilize the code would have exceeded the time allotted.

Given more time and resources, the team would have liked to test other forms of input and output. For example, a pushbutton input to toggle the LED, or a buzzer as the output. Maybe even a motor or some sort of physical movement as the output. Also, the team would have liked to program a better error message for keys that were not printable, or even find a way to make all keys printable. For example, if an arrow key was typed (which is normally not a printable character), maybe the Unicode arrow symbol (→) could have been displayed.

# Appendixes

This section contains referenced material.

## Appendix A - ANSI Escape Sequences

## 

## Appendix B – Code for Goal 1

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

// Includes

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

#include <c8051f120.h>

#include <stdio.h>

#include <ctype.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**;**

int lastLineUsed **=** 12**;**

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[0;44;33m"**);** //reset attributes, set background to blue, set forground to yellow

printf**(**"\033[2;30HType <ESC> to end the program\n\n\n\n\r"**);**

printf**(**"The keyboard character is: "**);**

printf**(**"\033[12;25r"**);** //set scroll area

**while(**1**)**

**{**

printf**(**"\033[H"**);** //go home, top left

choice **=** getchar**();** //get the character input

printf**(**"\033[2k"**);** //erase the top left line because there was just a typed character

**if(**isprint**(**choice**))**

**{**

printf**(**"\033[%i;40H \n"**,** lastLineUsed**);** //move cursor to the scrollable area

**if(**lastLineUsed **<** 25**)**

**{**

lastLineUsed **=** lastLineUsed **+** 1**;**

**}**

printf**(**"The keyboard character %x is\033[4;5m not printable\033[0;44;33m"**,**

**(**choice **&** 0xff**));**

printf**(**"\033[H"**);** //go back home

**}**

**else**

**{**

printf**(**"\033[6;28H"**);** //move cursor to the correct position

printf**(**"\033[37m%c"**,** choice **);** //print typed character, in white

printf**(**"\033[H"**);** //go back home

**}**

// select which option to run

//P1 |= 0x40; // Turn green LED on

//if (choice == '0')

// return;

//else if(choice == '1')

// printf("\n\nHere we go again.\n\n\r");

//else if(choice == '2') // clear the screen with <ESC>[2J

// printf("\033[2J");

//else

//{

// inform the user how bright he is

// P1 &= 0xBF; // Turn green LED off

// printf("\n\rA \"");

// putchar(choice);

// printf("\" is not a valid choice.\n\n\r");

//}

**}**

**}**

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

// 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 **=** 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 **=** 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 on P0.0 pin to push-pull

P1MDOUT **|=** 0x40**;** // Set green LED output P1.6 to push-pull

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 **=** SFRPAGE**;** // Save Current SFR page

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**;**

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

SSTA0 **=** 0x10**;** // SMOD0 = 1

TI0 **=** 1**;** // Indicate TX0 ready

SFRPAGE **=** SFRPAGE\_SAVE**;** // Restore SFR page

**}**

## Appendix C – Code for Goal 2

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

// Includes

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

#include <c8051f120.h>

#include <stdio.h>

#include <ctype.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**);**

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

// Global Variables

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

\_\_sbit \_\_at **(**0x90**)** Pot0**;** //Input on Port 1, Pin 0 (12 on board)

\_\_sbit \_\_at **(**0x91**)** Pot1**;** //Input on Port 1, Pin 1 (13 on board)

\_\_sbit \_\_at **(**0x92**)** Pot2**;** //Input on Port 1, Pin 2 (10 on board)

\_\_sbit \_\_at **(**0x93**)** Pot3**;** //Input on Port 1, Pin 3 (11 on board)

\_\_sbit \_\_at **(**0xA0**)** LED0**;** //Output on port 2, Pin 0, (29 on board)

\_\_sbit \_\_at **(**0xA1**)** LED1**;** //Output on port 2, Pin 0, (30 on board)

\_\_sbit \_\_at **(**0xA2**)** LED2**;** //Output on port 2, Pin 0, (27 on board)

\_\_sbit \_\_at **(**0xA3**)** LED3**;** //Output on port 2, Pin 0, (28 on board)

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

// MAIN Routine

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

void main**(**void**)**

**{**

char Mem0**;**

char Mem1**;**

char Mem2**;**

char Mem3**;**

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

Mem0 **=** Pot0**;**

Mem1 **=** Pot1**;**

Mem2 **=** Pot2**;**

Mem3 **=** Pot3**;**

**while(**1**)**

**{**

printf**(**"\033[H"**);** //go home, top left

**if(**Pot0 **!=** Mem0 **||** Pot1 **!=** Mem1 **||** Pot2 **!=** Mem2 **||** Pot3 **!=** Mem3**)**

**{**

printf**(**"\033[2J"**);** //erase screen

**if(**Pot0 **==** 1**)**

**{**

LED0 **=** 1**;**

printf**(**"LED0 is HIGH"**);**

**}**

**else**

**{**

LED0 **=** 0**;**

printf**(**"LED0 is LOW"**);**

**}**

Mem0 **=** Pot0**;**

printf**(**"\n\r"**);**

**if(**Pot1 **==** 1**)**

**{**

LED1 **=** 1**;**

printf**(**"LED1 is HIGH"**);**

**}**

**else**

**{**

LED1 **=** 0**;**

printf**(**"LED1 is LOW"**);**

**}**

Mem1 **=** Pot1**;**

printf**(**"\n\r"**);**

**if(**Pot2 **==** 1**)**

**{**

LED2 **=** 1**;**

printf**(**"LED2 is HIGH"**);**

**}**

**else**

**{**

LED2 **=** 0**;**

printf**(**"LED2 is LOW"**);**

**}**

Mem2 **=** Pot2**;**

printf**(**"\n\r"**);**

**if(**Pot3 **==** 1**)**

**{**

LED3 **=** 1**;**

printf**(**"LED3 is HIGH"**);**

**}**

**else**

**{**

LED3 **=** 0**;**

printf**(**"LED3 is LOW"**);**

**}**

Mem3 **=** Pot3**;**

**}**

printf**(**"\033[H"**);** //go home, top left

**}**

**}**

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

// 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 **=** 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 **=** 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 on P0.0 pin to push-pull

//Port 1 (input)

P1MDOUT **&=** 0xF0**;** //Sets input pins 0-3 using F0: 1111 0000

P1 **|=** 0x0F**;** //Pules input pins 0-3 with 0000 1111

//Port 2 (output)

P2MDOUT **|=** 0x0F**;** //set output pins (port 1) to pushpull using 0000 1111

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 **=** SFRPAGE**;** // Save Current SFR page

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**;**

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

SSTA0 **=** 0x10**;** // SMOD0 = 1

TI0 **=** 1**;** // Indicate TX0 ready

SFRPAGE **=** SFRPAGE\_SAVE**;** // Restore SFR page

**}**