Microprocessor Systems
Lab 1: IDE & ANSI Display

Nick Choi Samuel Deslandes

1 Introduction

The overall goal of this lab is to become familiar with utilizing the registers on the 8051 as well as performing fundamental I/O operations with it. This lab also served as an introduction to the VT100 Terminal and the utilization of the ANSI escape sequences it features.

This lab was divided into 3 parts. The first was an exercise in basic terminal I/O in which a C program was written to await a user keystroke and, if printable, output it onto the terminal display. The second part was an enhancement of the first part with heavy usage of ANSI escape codes to format text on the display. The ANSI escape codes are special sequences beginning with <ESC>(\033 in octal or $1B in hex) which modify terminal behavior. This includes features such as underlined text, background/foreground color changes, changing the cursor position, selectable scrolling, etc... The third part of the lab involved configuring ports on the 8051 to be either inputs or outputs. In this part of the lab hardware connected to the input port could be used to change the state of hardware connected to the output ports of the 8051.

2 Methods

2.1 Software

The code for parts 1, 2 and 3 can be found in Appendix A, B and C respectively. All code was uploaded and run on the 8051 through the programming/debugging USB port.

In the first section of the lab, a C program was written so that user input from a keyboard could be displayed in the ANSI terminal. The 8051 determined which keyboard character was pressed using the "getchar()" and "putchar()" functions defined in the "putget.h" header file. It should be noted that the "getchar()" function was modified from the original header file to not echo captured keystrokes. The modified version can be found in section 5.1 below. Whenever a printable character was input, it was printed to the terminal display with the following message: "The keyboard character is *." Unprintable characters were not displayed and the program would await the user's next input. Since the program runs in an infinite loop, pressing the <ESC>key both restarted the C program and cleared the terminal. In order to determine whether the input was printable or not, the captured character's ASCII was compared with the printable characters on the ASCII table, which range from 0x20 (space) through 0x7E (~).

The second part builds upon the C program written for the first adding various text effects and formatting within the VT100 terminal, all of which could be accomplished by utilizing ANSI escape sequences. Table 1 in section 5.3.1 contains all of the ANSI escape sequences used for this lab. The first modification was to change the background color to blue and the foreground color to yellow. This was done using the codes "\033[1;43m" and "\033[1;33m" respectively. The first line of text contains program termination information similar to that of part 1; It is printed on line 2 of the terminal and is horizontally centered. This can be accomplished using the escape code for changing the cursor position "\033[{row};{col}H". In this case 2 and 30 were used for the row and column respectively. As was done in part 1, the keyboard response must also be displayed, this time on line 6 with the keyed in character printed in white. The same code above was used to jump the cursor to line 6, and "\033[1;37m" is used to change the text's color to white. If the keyed in character is not printable, the message "The keyboard character $XX is 'not printable'." should appear starting halfway down the terminal (line 12 in our case) and work its way down the screen. This text should be blinking and make an audible 'BEL' noise when a non-printable character is keyed in. The terminal should be set such that only the bottom half of the screen scrolls. Although the escape code to change the position of the cursor could be used accomplish this, the codes to save and restore the cursors position were used instead so that we would not have to directly keep track of what line to jump to. In order to do this, we had it so that even before a non-printable was keyed in, the position of the next error message was already stored and ready to be restored and written to when the time came. The error messages were ended with the ASCII newline escape sequence ('\n') to move the cursor down, and this new position was saved for printing the next error. When the error messages reach the bottom of the terminal, only the lines included in the selective scrolling area would scroll. The ANSI codes for saving and restoring the cursor's position are "\033[s" and "\033[u" respectively. In order to set a selective scrolling area the code "\033[{start},{stop}r" must be used. Only the lines between {start} and {stop} will scroll. To get the audible feedback portion the ASCII escape sequence '\a' was used. When writing the code block pertaining to printing the error message, it is important to remember to turn off the underscore and blinking once the desired text with these effects has been printed.

Part 3 took the code in a very different direction and focused more on the hardware than the software. In terms of software, all of port 1 had to be configured to be in open-drain mode (input), and all of port 2 to push-pull mode (output). This was done by setting all of the bits of the P1MDOUT special function register (SFR) low, and all of those of P2MDOUT high. This should be done in the 'PORT_INIT()' function. In the main function, all that had to be done was read the value of port 1 into port 2. This could be done using the P1 and P2 port SFRs.

2.2 Hardware

Parts 1 and 2 of this lab did not require any hardware other than a serial-to-USB adapter in order to interface with the terminal.

In the third section of the lab, the pins on port 1 were connected to the input device, the potentiometer module, and pins on port 2 were connected to the output device, the LED module. Since there were only 4 potentiometers on the potentiometer module only the first

4 pins on each port were used (P1.0 - P1.3 and P2.0 - P2.3). These corresponded to pins 12, 13, 10, and 11 on the EVB for port 1, and 29, 30, 27, and 28 for port 2. A circuit diagram can be found in the appendices, section 5.4.1.

The input pins of the module have inverters incorporated into them so that whenever a logic high is applied to an input pin, the signal is changed to be a logic low. This inversion allows the LED to illuminate because there will be a voltage difference across the anode and cathode of the LED. If a logic low is applied to the input pin, the signal is changed to be a logic high. This removes the voltage difference between the anode and cathode of the LED and thus prevents the LED from lighting up.

3 Results

By completing section one of the lab, a functioning C program was produced which read keyboard input and displayed it in an ANSI terminal. After completing section two, an improved version of the program from section one was produced. This version reacted to user input and manipulated the output of the ANSI terminal in several ways. The final deliverable was an LED module which was controlled by a potentiometer module.

4 Conclusion

The end results of this lab ultimately matched with the initial goals however there were numerous instances where our systems expected behavior did not correspond to its actual behavior. In order to properly produce the results that we needed for this lab, various hardware and software debugging techniques were utilized in order to verify the performance of each section of the lab. With some of these testing techniques, it was determined that the inputs to the 8051 utilize Schmitt triggers in order to prevent oscillating logic highs and logic lows. The use of Schmitt triggers makes the inputs of the 8051 more resilient to noise because they create separate thresholds for high and low values rather than having one shared boundary value.

If more time was given to complete this lab assignment, additional conditional statements could be added to further enhance the responses of the ANSI terminal to user input. These statements could incorporate more escape sequences to display new error messages, to integrate hardware components to the error messages or to further manipulate the text formatting in the ANSI terminal.

5 Appendices

5.1 Modified putget.h

```
//------------------------------------------------------------------------------
// putget.h
//------------------------------------------------------------------------------
```

```
// Title:                 Microcontroller Development: putchar() & getchar()
    functions.
// Author:                Dan Burke
// Date Created:          03.25.2006
// Date Last Modified:    03.25.2006
//
// Description:           http://chaokhun.kmitl.ac.th/~kswichit/easy1/easy1_3.
    html
//
//
// Target:                C8051F120
// Tool Chain:            KEIL C51

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

// putchar()
//------------------------------------------------------------------------

void putchar(char c)
{
    while(!TI0);
    TI0=0;
    SBUF0 = c;
}

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

// getchar()
//------------------------------------------------------------------------

char getchar(void)
{
    char c;
    while(!RI0);
    RI0 =0;
    c = SBUF0;
// Echoing the get character back to the terminal is not normally part of
    getchar()
//    putchar(c);     // echo to terminal
    return SBUF0;
}
```

5.2 Part 1

5.2.1 Code

```
//------------------------------------------------------------------------

// Lab1-1
//------------------------------------------------------------------------

// Nick Choi, Samuel Deslandes
// C program which awaits a user keystoke and if printable, displas it on the
    terminal
```

```
//------------------------------------------------------------------------------------------

// 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;
  //unsigned char row = 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("Exit command is: <ESC>  \n\n\r");
  while(1)
    {
    choice = getchar(); // Await and capture user input
```

```
    if (choice == 0x1b){ // If <ESC> is pressed end the program
      return;
    }
    else if (choice >= 0x20 && choice <= 0x7E){    //Check if input is
        printable
      printf("The keyboard character is ");
      putchar();
      printf(".\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
}
```

5.3 Part 2

5.3.1 ANSI Escape Sequence Table

\033[2J	Clear screen and return cursor to home
\033[1;33m	Set text color to yellow
\033[1;43m	Set background color to blue
\033[1;37m	Set text color to white
\033[{row};{col}H	Move cursor position to ({row},{col})
\033[{start};{stop}r	Set scroll area from {start} to {stop}
\033[s	Save cursor position
\033[u	Restore cursor position
\033[5m	Turn blinking text on
\033[25m	Turn blinking text off
\033[4m	Underline text
\033[24m	Turn underline text off

Table 1: Quick reference table of ANSI escape sequences

5.3.2 Code

```
//------------------------------------------------------------------------------
// Lab1-2
//------------------------------------------------------------------------------
// Nick Choi, Samuel Deslandes
// An enhancment of the program from Lab1-1, which uses ANSI escape sequences
    to format
// text on the VT100 terminal.
//------------------------------------------------------------------------------
// 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[1;33m");          // Set the text to be yellow (background is
     already blue)


  printf("\033[2;30H");        // Center text
  printf("Exit command is: <ESC>  \n\n\r");

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

  printf("\033[12;1H");       // Jump to line 12 and save position
  printf("\033[s");

    while(1)
    {

    // Get the keyboard character and output it to the terminal
    printf("\033[6;1H");       // Move cursor to row 6
    printf("The keyboard character is ");

    printf("\033[6;27H");        // Move cursor
    choice = getchar();
    if (choice == 0x1b){
      return;
```

```
    }
    else if (choice >= 0x20 && choice <= 0x7E){    //Check if input is
        printable
      printf("\033[1;37m");           // Set character to white
      putchar(choice);
      printf("\033[1;33m.");            // Print a yellow period after the
          entered key
    }
    else{
      printf("\033[u\a");                // Restore cursor position and beep
      printf("\033[5m");                 // Turn blink on
      printf("The keyboard character $%02x is ", choice);
      printf("\033[4m'not printable'\033[24m.\n\r");     // Underscore
      printf("\033[s");                  // Save cursor position
      printf("\033[25m");                // Blink off

    }
    }
}

//------------------------------------------------------------------------------
// 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
}
```

5.4 Part 3

5.4.1 Circuit Schematic

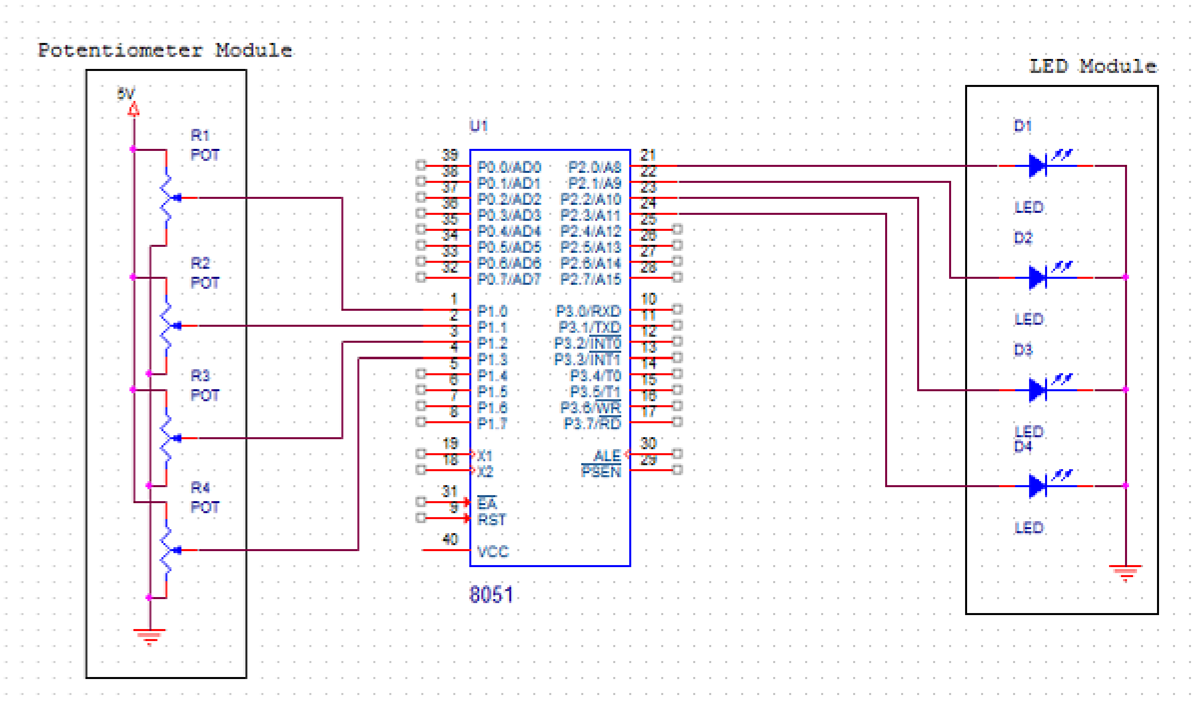

Figure 1: Circuit schematic for part 3

5.4.2 Code

```
//-----------------------------------------------------------------------------
// Lab1-3
//-----------------------------------------------------------------------------
// C program which reads input from a potentiometer module and writes that
   value to port 2
// which is connected to an LED module
//-----------------------------------------------------------------------------
// 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;
  unsigned char port1;

    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");                // Clear screen and reset curosr
  P2 = 0xF0;
  printf("Starting value P2 = 0x%02x\n\r",P2);
  while(1){
    port1 = P1;    // Read port 1
    P2 = port1;    // Write to port 2
  }

}

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

// 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 (XBARE) and weak
        pull-up
    P0MDOUT |= 0x01;                     // Set TX0 on P0.0 pin to push-pull

  P2MDOUT = 0xFF;             // Set port 2 to push-pull
```

```
  P1MDOUT = 0x00;                           // Set port 1 to open-drain

    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
}
```

6 References