

Microprocessor Systems
Lab 3: Asynchronous & Synchronous Serial
Communications

Nick Choi Samuel Deslandes

10/17/16

1 Introduction

The overall goal of this lab is to become familiar with configuring the UART and SPI serial communication protocols on the 8051 and utilizing their respective interrupts to perform terminal operations. The lab is divided into three sections. In the first section, a C program is created to configure both UART0 and UART1 on the 8051 to react to keyboard presses. The code constantly polls both of the ports to check whether a keyboard character has been entered into the terminal. Once a character is pressed, the program will echo the character back to the terminal on both UART0 and UART1. If the entered character is an `ESC` key, the program displays a brief message and stops the program on both UARTs. Unlike past labs, the `getchar()` function could not be used to check whether a keyboard button was pressed because it would only check UART0. In the second section, the programs logic changed so that interrupts would be used rather than polling. Additionally, the program needed to be altered so that two 8051s could communicate through their UART1 connections. In the third section, a new C program is written echo characters using the 8051s SPI protocol. Whenever a character is entered on the 8051s UART0, it will be echoed to the terminal through the 8051s SPI0. A loopback wire is used to connect the MISO and MOSI pins to verify that the SPI configuration was successful. Once the SPI0 is configured correctly, the code is modified again so that the 8051 can communicate with a 68HC11 via SPI0. The 8051 functions as the master device and the 68HC11 functions as the slave device.

2 Methods

2.1 Software

The code for parts 1, 2 and 3 can be found in the appendix below. All code was uploaded and run on the 8051 through the programming/debugging USB port.

2.1.1 Part 1

The purpose of C program for the first section of the lab was to write a procedure that would monitor the UART0 and UART1 serial ports continuously, and echo a message received on one port to the other. For this section of the lab, this was accomplished by polling the receive interrupt flags RI0 and RI1 for UART0 and UART1 respectively.

Since this lab uses both UART0 and UART1, both ports must be enabled on the crossbar. The enable bit for UART0 can be found on the XBR0 SFR, XBR2 for that of UART1. As always, the crossbar enable bit in XBR2 must be set. UART0 was configured to be in mode 1 (8-bit UART with variable baud rate), and use Timer2 to generate a baud rate of 9600bps. UART0 configuration can set in the SSTA0 and SCON0 SFRs. Timer2 was configured to be in auto-reload mode and use SYSCLK as a base. These settings can be found in the TMR2CN and TMR2CF SFRs. To count to 9600, the reload value must be set to `0xFEBC`. This was calculated as follows:

$$\text{Model.BaudRate} = \frac{\text{SYSCLK}}{16 * (65536 - \text{ReloadValue})} \quad (1)$$

For a desired baud rate of 9600 and SYSCLK of 49,766,400Hz, this comes out to:

$$9600 = \frac{49766400}{16 * (65536 - \text{ReloadValue})} \quad (2)$$

$$\text{ReloadValue} = 65212 \quad (3)$$

This is equivalent to 0xFEBC in hex.

UART1 was similarly configured, but used Timer1 to generate a baud rate of 115200bps. UART1 can be configured in the SCON1 SFR. Timer1 was set to be an 8-bit counter with auto-reload, and use SYSCLK as a base; these changes can be made in the TMOD SFR. The Timer1 reload value should be set as 0xE5. This was calculated as shown below:

$$\text{Model_BaudRate} = \frac{\text{SYSCLK}}{16 * (256 - \text{ReloadValue})} \quad (4)$$

$$\text{ReloadValue} = 229 \quad (5)$$

This is equivalent to 0xE5 in hex.

When configuring both UARTs be sure that the receive enable bits are set in the SCON SFRs and that the timers have been started. In addition to this, in the P0MDOUT SFR configure the TX pins (P0.0, P0.2) as push-pull for outputs, and the RX pins (P0.1, P0.3) as open-drain for inputs; Also, in the P0 SFR set the inputs (RX pins) for high impedance.

The main routine of this program consists of two sub-routines ‘checkSBUF()’ and ‘echo()’ which are run in an infinite loop. The ‘checkSBUF()’ function continuously polls RI0 and RI1, and returns the value stored in the UART data register when one of the flags is set. The UART data register (SBUF_n) is used for both a receive and a transmit buffer. When data is read from SBUF, as is the case for the ‘checkSBUF()’ function, it comes from the receive buffer. When data is written to SBUF it goes to the transmit buffer and is eventually transmitted when a full byte has been written. The returned value from ‘checkSBUF()’ is then passed into the ‘echo()’ function, where it is loaded into both SBUF0 and SBUF1 for transmission to both terminals. Recall that SBUF0 and SBUF1 are on different pages, and that SFRPAGE should be changed accordingly.

2.1.2 Part 2

This section of the lab was divided into two parts. The task for the first part was similar to that of section 1, however rather than polling, interrupts were used. The second part involved connecting UART1 to that of another 8051, so that when data is sent to UART0 of one of the 8051s, it echoed to the UART0 of the other microcontroller. The initial configurations were all the same as they were in section 1, with the addition of having to set the global interrupt enable bit (EA), as well as the individual UART0 and UART1 interrupt enable bits which can be found on the IE and EIE2 SFRs respectively. With these bits set, whenever the transmit interrupt flags (TIn) or receive interrupt flags (RIn) are set, the CPU vectors to the appropriate interrupt service routine (ISR); These flags must be cleared manually by the software. The UART0 interrupt has a priority level of 4, while the UART1

interrupt has a priority of 20. One thing to keep in mind in regard to these interrupts is that while UART0 interrupts are enabled, unless the UART0 and UART1 interrupt priorities are swapped, UART1 will not receive any interrupts. Since both receiving and transmitting will trigger an interrupt, the ISR must poll each flag to verify the source of the interrupt.

For part 1 of this section of the lab, the main would only check if the command to end the program had been sent; everything else was handled in the ISRs. The UART0 ISR checks if the source of the interrupt was a receive by checking if RI0 has been set. If so, it is cleared and the ‘echo()’ subroutine is called. After this, UART0 interrupts are temporarily disabled to allow UART1 interrupts to go through. The UART1 ISR has the same routine, however rather than disabling UART0 interrupts, it re-enables them.

The program for part 2 was handled slightly differently. The main routine now continuously polled two flags ‘UART0_flag’ and ‘UART1_flag’, which would be set in the ISRs of UART0 and UART1 respectively. The ‘echo()’ function is also used for this part, of the lab, but with the slight modification of disabling UART0 interrupts before loading data into the SBUF registers. The UART0 ISR checks whether the source of the interrupt is a receive, and if it is it clears RI0, reads SBUF0 into a global variable, and sets the UART0_flag. The UART1 ISR does the same, but enables UART0 interrupts at the end of the ISR.

In the main routine, once the UART0_flag has been set, the program calls the ‘echo()’ function, and clears the flag. If the UART1_flag has been set, the program disables UART0 interrupts, transmits the character to UART0 via the SBUF0 register, re-enables UART0 interrupts, then clears the flag. In this case, the character is sent only to UART0 rather than being echoed to both ports to avoid a feedback loop of infinite transmissions between the two microcontrollers.

In order to allow interrupts on UART1 to go through, in the main routine there is a brief period of time in which UART0 is disable. This small window was implemented by incrementing a variable to 255 in a for loop.

2.1.3 Part 3

The third section of this lab deals with the serial peripheral interface bus (SPI), a synchronous serial communication interface. The goal of this section was to communicate with the 68HC11 as a slave device, and use it to echo back any characters that get sent to it. Although the primary focus of this lab is SPI, UART0 is also used to send data to the terminal. As was the case with UART, SPI must also be enabled on the crossbar to function. It’s enable bit can be found on XBR0. The relevant configuration SFRs for SPI are SPI0CFG, SPI0CN, and SPI0CKR. In SPI0CFG, enable master mode, so that the 8051 acts as the master device. In the SPI0CN register, enable SPI0, and set it to operate in 4-wire single master mode. The SPI0CKR register stores data pertaining to SPI clock rate, which can be calculated as follows:

$$SCK = \frac{SYSCLK}{2 \times (SPI0CKR + 1)} \quad (6)$$

Setting SPI0CKR to 0x18 with a SYSCLK of 49,766,400 results in a SCK of 995,328Hz. For SPI communication, in the P0MDOUT SFR, the SCK, MOSI, and NSS pins (P0.2, P0.4,

P0.5 respectively) must be set to push-pull.

The program for this section used 3 sub-routines: ‘SPI0_READ()’, which handles receiving data from the slave device and outputting it to the terminal, ‘SPI0_WRITE()’, which handles receiving user input, and sending that data to a slave device, and ‘write_dummy()’ which sends a byte of garbage data (a dummy) to the slave device. The main routine simply waits for user input by polling RI0, and when set, clears the flag and calls the ‘SPI0_WRITE()’ and ‘SPI0_READ()’ functions.

The ‘write_dummy()’ function first selects the slave by clearing NSSMD0. It then checks to make sure that SPI is not busy, then clears SPIF and writes a byte of data (0xFE in this case) to the SPI data register SPI0DAT. Like the UART data register SBUF, SPI0DAT is used both to receive and transmit. Checking that the SPI is not busy with a transmission can be done by monitoring bit 7 of the SPI0CFG register. SPIF is the SPI interrupt flag, which is set at the end of a data transfer. Like RIn and TIn, this flag must be cleared manually.

The ‘SPI0_WRITE()’ function starts by loading the data in SBUF0 into a variable. It then selects the slave, ensures that SPI is not busy, clears SPIF, and loads the from the variable into SPI0DAT to be sent to the slave. It then waits for the transmission to end and clears SPIF. Because the slave device contains a second shift register, the echoed response will be delayed by one transmission. This is why dummy bytes must be read and written in between every actual read and write. The next step in the ‘SPI0_WRITE()’ routine is to read the dummy byte, which is done by selecting the slave and moving the dummy byte out of SPI0DAT. This program will output the dummy char to the terminal. After reading the dummy char, the input character (received on UART0) is displayed on the terminal and the ‘write_dummy()’ function is called. The ‘SPI0_READ()’ function follows the same procedure as reading the dummy byte.

The display is formatted using ANSI escape sequences to split the screen horizontally, with the top half used to display the input received on UART0, and the bottom half for the echoed character (received on the MISO pin of SPI). In order to make the two halves scroll independently, the scroll area must be redefined to the half being printed to before each print. A for loop was used to move the cursor down the appropriate amount of rows. This was all done using ANSI escape sequences.

2.2 Hardware

The hardware for section 1 involved connecting a DB9 adapter to be used at the UART1 port. A schematic for this can be viewed in the appendix below. The TD pin on the DB9 was connected to the input of a hex inverter, with the output connected to the RX1 pin of UART1 (P0.3). The TX1 pin (P0.2) was connected to the input of an inverter, with the output connected to the RD pin on the DB9. In order to establish a common ground between the adapter and the microcontroller, the ground pin on the DB9 was connected to pin 1 on the 60 pin bus (DGND), which was connected to the ground on the power supply used.

Section 2 called for connecting the UART1s of two microcontrollers. This was done by connecting the TX1 pin of one 8051 to the RX1 pin of the other, and vice versa for the RX1 pin. A diagram of this can also be found in the appendix below.

In section 3, the 68HC11 EVB was used as a SPI slave device. It was connected to the 8051 by connecting the MISO pins (pin 22 on 68HC11 to P0.3 of 8051), the MOSI pins (pin 23 on 68HC11 to P0.4 of 8051), the SCK pins (pin 24 on 68HC11 to P0.2 of 8051), the SS* pin on the 68HC11 to the NSS pin on the 8051 (pin 24 on 68HC11 to P0.5 of 8051), and establishing a common ground between the two devices (pin 1 on both 60 pin buses). The schematic for this can be viewed in the appendix section below.

3 Results

By completing section one of the lab, a functioning C program was produced to echo characters between the 8051s UART0 and UART1 ports via polling. After completing section two of the lab, a C program was developed to echo characters between two 8051s UART ports. The UART0 ports are used to take in the keyboard presses and the UART1 ports are used to communicate between the two processors. By completing section three of the lab, a C program was created to use the 8051s SPI0 port to echo characters back and forth with an 68HC11.

4 Conclusion

The end results of the lab generally matched with the initial goals however there were numerous instances where the system did not behave as expected. During section 2 and 3 of the lab, a lot of time was spent testing and debugging the C programs to control the serial communication ports. An enhancement was added to section two of the lab so that the characters echoed by both UARTs would scroll on the screen in two separate scrolling areas. If more time was given to complete this lab assignment, additional enhancements would be added to section 3 of the lab so that the slave would react to certain keypresses. This would increase the functionality of the SPI0 communication and would make the program more interactive for the user.

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 Circuit Schematic for section 1

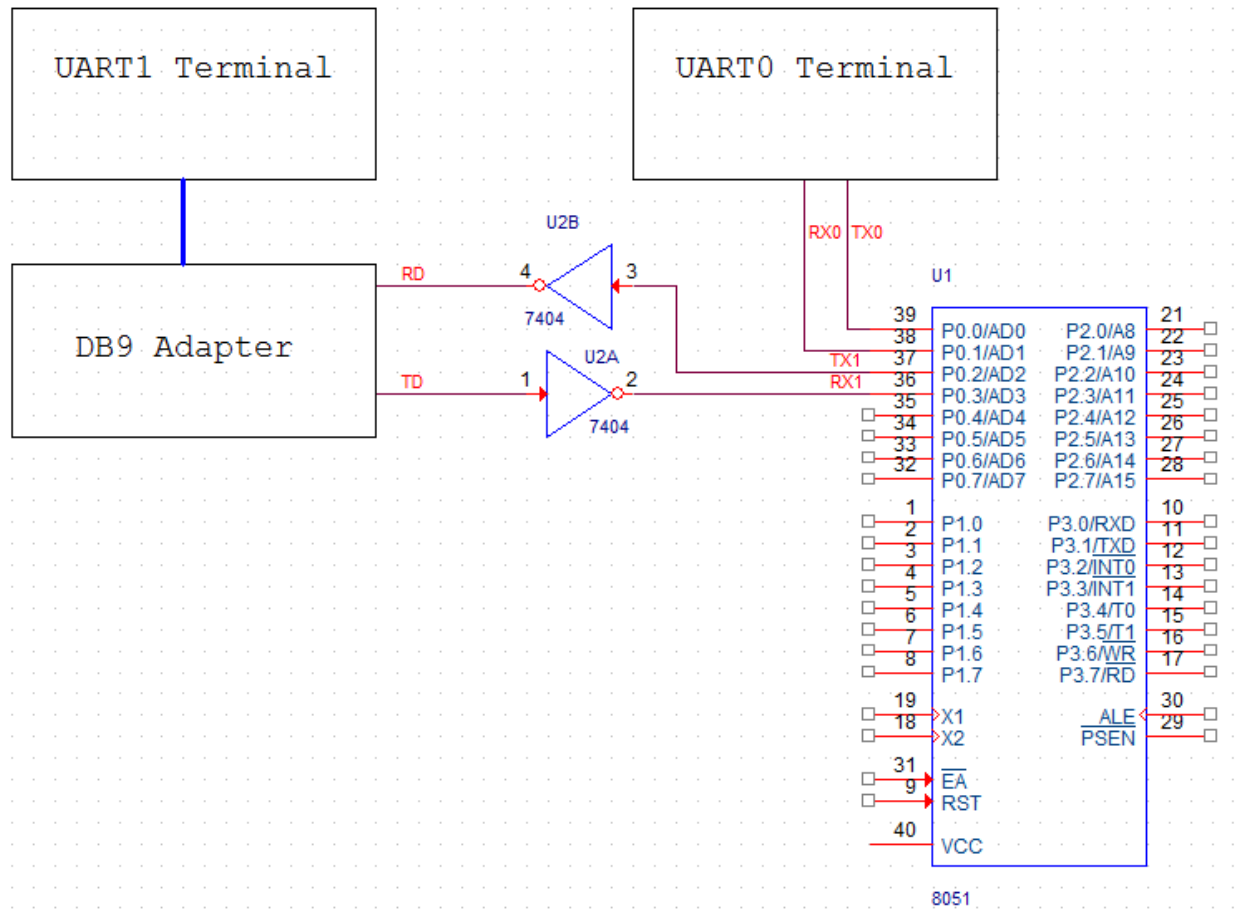


Figure 1: Circuit schematic for part 1

5.2.2 Code for part 1

```
//-----  
// Includes  
//-----  
#include <c8051f120.h>  
#include <stdio.h>  
#include <stdlib.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  
//#define BAUDRATE    19200     // UART baud rate in bps  
  
//-----  
// Function PROTOTYPES  
//-----  
void main(void);  
void PORT_INIT(void);  
void SYSCLK_INIT(void);  
void UART0_INIT(void);  
void UART1_INIT(void);  
void TIMER0_INIT(void);  
char checkSBUF(char CUR_PAGE);  
void echo(char character);  
  
//-----  
// Main Function  
//-----  
  
void main(void){  
    // Declare local variables  
    char message;  
    unsigned long i;  
  
    WDTCN = 0xDE;                // Disable the watchdog timer  
    WDTCN = 0xAD;  
  
    PORT_INIT();                // Initialize the Crossbar and GPIO  
    SYSCLK_INIT();              // Initialize the oscillator  
    UART0_INIT();               // Initialize UART0  
    UART1_INIT();               // Initialize UART1  
  
    SFRPAGE = UART1_PAGE;       // Direct output to UART1  
  
    printf("\033[2J");           // Erase screen & move cursor to home position  
    printf("Test of the printf() function.\n\n\r");  
  
    SFRPAGE = UART0_PAGE;       // Direct output to UART1  
  
    printf("\033[2J");           // Erase screen & move cursor to home position  
    printf("Test of the printf() function.\n\n\r");  
  
    while(1)  
    {  
        message = checkSBUF(SFRPAGE);  
        if(message == 27){  
            printf("\n\n\rStopping now...");  
            SFRPAGE = UART1_PAGE;  
            printf("\n\n\rStopping now...");  
            for(i = 0; i < 2000000; i++);  
            return;  
        }  
    }  
}
```

```

    }
    echo(message);
}
}

char checkSBUF(char CUR_PAGE){
    char serial_dat0;
    char serial_dat1;

    while(1){
        SFRPAGE = UART0_PAGE;
        if(RI0){
            serial_dat0 = SBUF0;
            RI0 = 0;
            SFRPAGE = CUR_PAGE;
            return serial_dat0;
        }
        SFRPAGE = UART1_PAGE;
        if(RI1){
            serial_dat1 = SBUF1;
            RI1 = 0;
            SFRPAGE = CUR_PAGE;
            return serial_dat1;
        }
    }
}

void echo(char character){
    char SFRPAGE_SAVE;
    SFRPAGE_SAVE = SFRPAGE;

    SFRPAGE = UART0_PAGE;
    SBUF0 = character;

    SFRPAGE = UART1_PAGE;
    SBUF1 = character;

    SFRPAGE = SFRPAGE_SAVE;
}

//-----
// Interrupts
//-----

//-----
// 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;
    EA      = 1;                // Enable interrupts as selected.
    XBR0     = 0x04;            // Enable UART0.
    XBR1     = 0x00;            //
    XBR2     = 0x44;            // Enable Crossbar and weak pull-ups and UART1.
    P0MDOUT = 0x05;            // P0.0 (TX0) and P0.2 (TX1) are configured as Push-Pull for
    output, P0.1 (RX0) and P0.3 (RX1) are Open-drain
    P0       = ~0x05;          // Additionally, set P0.0=0, P0.1=1, P0.2=0, P0.3=1

    SFRPAGE = SFRPAGE_SAVE;    // Restore SFR page.
}

```

```

//-----
// SYSCLK_Init
//-----

// Initialize the system clock 22.1184Mhz

void SYSCLK_INIT(void){
    int i;

    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR page.

    SFRPAGE = CONFIG_PAGE;
    OSCXCN = 0x67;             // Start external oscillator
    for(i=0; i < 256; i++);    // Wait for the oscillator to start up.
    while(!(OSCXCN & 0x80));    // Check to see if the Crystal Oscillator Valid Flag is set.
    CLKSEL = 0x01;             // SYSCLK derived from the External Oscillator circuit.
    OSCICN = 0x00;             // Disable the internal oscillator.

    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;             // SYSCLK derived from the PLL.

    SFRPAGE = SFRPAGE_SAVE;    // Restore SFR page.
}

//-----
// UART0_Init
//-----

// Configure the UART0 using Timer1, for 9600 and 8-N-1.
void UART0_INIT(void)
{
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR page.

    SFRPAGE = TMR2_PAGE;
    TMR2CN = 0x00;             // Auto-reload mode, use clock defined in TMR2CF,
    TMR2CF = 0x08;             // Timer 2 uses SYSCLK as time base

    RCAP2H = 0xFE;             // Set timer 2 auto-reload value for 9600bps
    RCAP2L = 0xBC;

    TR2 = 1;                   // Start timer 2

    SFRPAGE = UART0_PAGE;
    SC0N0 = 0x50;              // Set Mode 1: 8-Bit UART
    SSTA0 = 0x15;              // UART0 baud rate divide-by-two disabled (SMOD0 = 1) and use
    TMR2.                                //
    TI0 = 1;                   // Indicate TX0 ready.

    SFRPAGE = SFRPAGE_SAVE;    // Restore SFR page
}

```

```

// Configure the UART1 using Timer1, for 115200 and 8-N-1.
void UART1_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   = 40;                        // Set Timer1 reload baudrate value T1 Hi Byte
    CKCON |= 0x10;                     // Timer1 uses SYSCLK as time base
    TL1   = TH1;
    TR1   = 1;                         // Start Timer1

    SFRPAGE = UART1_PAGE;
    SCON1  = 0x30;                     // Mode 1, 8-bit UART, enable RX
    TI1    = 1;                       // Indicate TX1 ready

    SFRPAGE = SFRPAGE_SAVE;           // Restore SFR page
}

```

5.3.1 Circuit Schematic for section 2

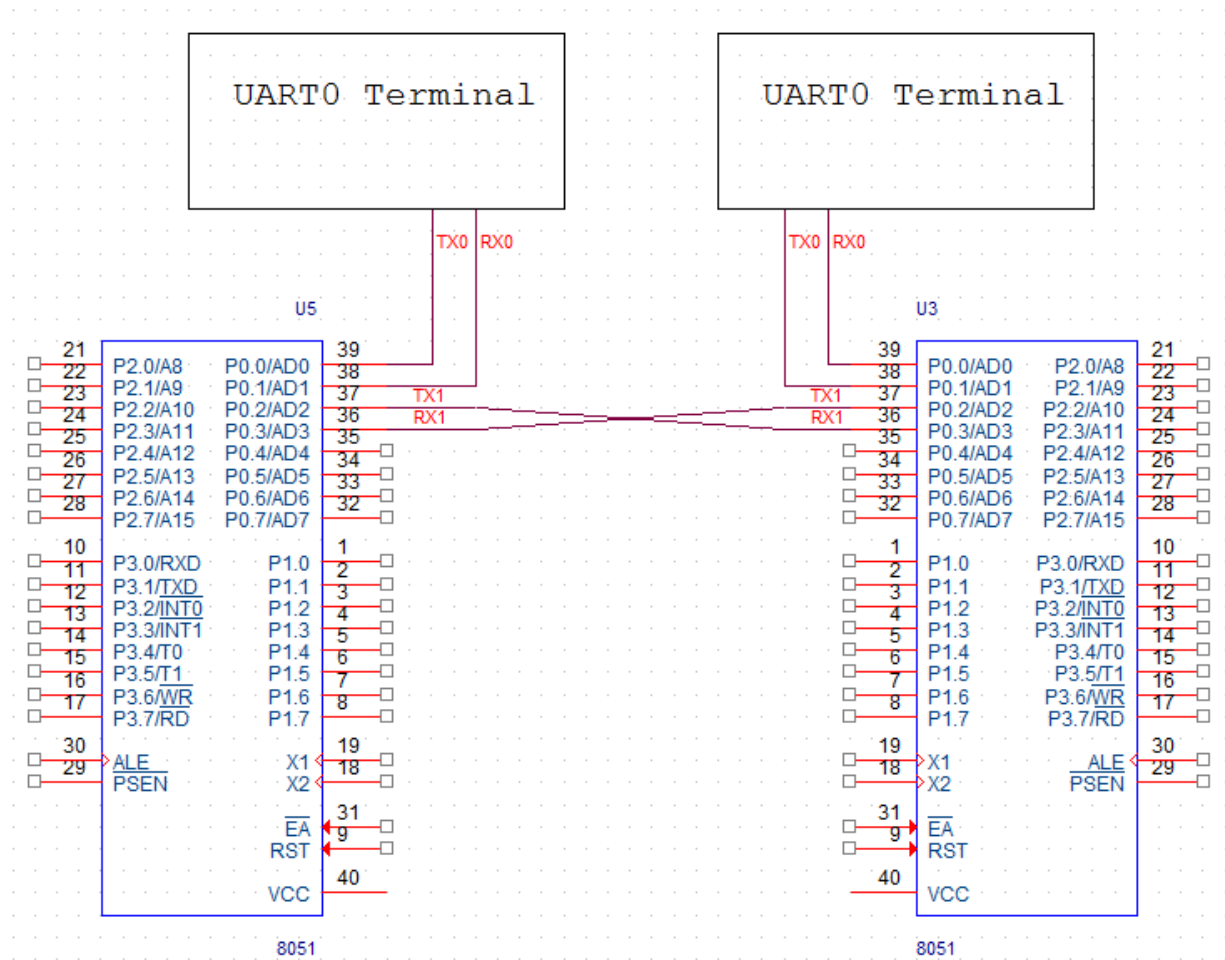


Figure 2: Circuit schematic for part 2

5.3.2 Code for part 2

```
//-----  
// Includes  
//-----  
#include <c8051f120.h>  
#include <stdio.h>  
#include <stdlib.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  
//#define BAUDRATE    19200     // UART baud rate in bps  
char CURR_PAGE;  
char exit_flag = 0;  
char UART0_flag = 0;  
char UART1_flag = 0;  
char choice = 0;  
  
//-----  
// Function PROTOTYPES  
//-----  
void main(void);  
void PORT_INIT(void);  
void SYSCLK_INIT(void);  
void UART0_INIT(void);  
void UART1_INIT(void);  
void TIMER0_INIT(void);  
char checkSBUF(char CURR_PAGE);  
void echo(char character);  
void INTERRUPT_INIT(void);  
void UART0_int(void) __interrupt 4;  
void UART1_int(void) __interrupt 20;  
  
//-----  
// Main Function  
//-----  
  
void main(void){  
    // Declare local variables  
    char message;  
    unsigned long i;  
    char j;  
  
    WDTCN = 0xDE;                // Disable the watchdog timer  
    WDTCN = 0xAD;  
  
    PORT_INIT();                 // Initialize the Crossbar and GPIO  
    SYSCLK_INIT();               // Initialize the oscillator  
    UART0_INIT();                // Initialize UART0  
    UART1_INIT();                // Initialize UART1  
    INTERRUPT_INIT();  
  
    SFRPAGE = UART0_PAGE;        // Direct output to UART1  
  
    printf("\033[2J");            // Erase screen & move cursor to home position  
    printf("Test of the printf() function.\n\n\r");  
  
    while(1)  
    {  
        SFRPAGE = UART0_PAGE;  
        if(exit_flag){  
            printf("\n\n\rStopping now...");  
        }  
    }  
}
```

```

        SFRPAGE = UART1PAGE;
        printf("\n\n\rStopping now...");
        for(i = 0; i < 20000; i++);
        break;
    }

    for(j=0; j < 255; j++){
        ES0 = 0;
    }
    ES0 = 1;

    if(UART0_flag){
        echo(choice);
        UART0_flag = 0;
    }
    if(UART1_flag){
        ES0 = 0;
        SBUF0 = choice;
        ES0 = 1;
        UART1_flag = 0;
    }
}
return;
}

void echo(char character){
    if(character == 27){
        exit_flag = 1;
        return;
    }
    ES0 = 0;
    SFRPAGE = UART0PAGE;
    SBUF0 = character;

    SFRPAGE = UART1PAGE;
    SBUF1 = character;
}

//-----
// Interrupts
//-----
void UART0_int(void) __interrupt 4{
    CURR_PAGE = SFRPAGE;
    SFRPAGE = UART0PAGE;
    if(RI0){
        RI0 = 0;
        choice = SBUF0;
        UART0_flag = 1;
    }
    SFRPAGE = CURR_PAGE;
}

void UART1_int(void) __interrupt 20{
    CURR_PAGE = SFRPAGE;
    SFRPAGE = UART1PAGE;
    if(RI1){
        RI1 = 0;
        choice = SBUF1;
        UART1_flag = 1;
    }
    ES0 = 1;
    SFRPAGE = CURR_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    = 0x44;            // Enable Crossbar and weak pull-ups and UART1.
    POMDOUT = 0x05;            // P0.0 (TX0) and P0.2 (TX1) are configured as Push-Pull for
        output, P0.1 (RX0) and P0.3 (RX1) are Open-drain
    P0      = 0x0A;            // Additionally, set P0.0=0, P0.1=1, P0.2=0, P0.3=1

    SFRPAGE = SFRPAGE_SAVE;    // Restore SFR page.
}

//-----
// SYSCLK_Init
//-----

// Initialize the system clock 22.1184Mhz

void SYSCLK_INIT(void){
    int i;

    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR page.

    SFRPAGE = CONFIG_PAGE;
    OSCXCN = 0x67;            // Start external oscillator
    for(i=0; i < 256; i++);    // Wait for the oscillator to start up.
    while(!(OSCXCN & 0x80));    // Check to see if the Crystal Oscillator Valid Flag is set.
    CLKSEL  = 0x01;            // SYSCLK derived from the External Oscillator circuit.
    OSCICN  = 0x00;            // Disable the internal oscillator.

    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;            // SYSCLK derived from the PLL.

    SFRPAGE = SFRPAGE_SAVE;    // Restore SFR page.
}

void INTERRUPT_INIT(void){
    EA = 1;                    // Enable interrupts as selected.
    ES0 = 1;
    EIE2 |= 0x40;
}

// Configure the UART0 using Timer1, for 9600 and 8-N-1.
void UART0_INIT(void)
{
    char SFRPAGE_SAVE;

```



```

SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page.

SFRPAGE = TMR2_PAGE;
    TMR2_CN = 0x00;           // Auto-reload mode, use clock defined in TMR2CF,
TMR2_CF = 0x08;              // Timer 2 uses SYSCLK as time base

RCAP2H = 0xFD;               // Set timer 2 auto-reload value for 4800bps
RCAP2L = 0x78;

TR2 = 1;                     // Start timer 2

SFRPAGE = UART0_PAGE;
    SCON0 = 0x50;             // Set Mode 1: 8-Bit UART
    SSTA0 = 0x15;            // UART0 baud rate divide-by-two disabled (SMOD0 = 1) and use
    TMR2.                                     TMR2.
    TI0 = 1;                  // Indicate TX0 ready.

SFRPAGE = SFRPAGE_SAVE;      // Restore SFR page
}

// Configure the UART1 using Timer1, for 115200 and 8-N-1.
void UART1_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 = 40;                  // Set Timer1 reload baudrate value T1 Hi Byte
    CKCON |= 0x10;            // Timer1 uses SYSCLK as time base
    TL1 = TH1;
    TR1 = 1;                  // Start Timer1

    SFRPAGE = UART1_PAGE;
    SCON1 = 0x30;             // Mode 1, 8-bit UART, enable RX
    TI1 = 1;                  // Indicate TX1 ready

    SFRPAGE = SFRPAGE_SAVE;      // Restore SFR page
}

```

5.4 Part 3

5.4.1 Circuit Schematic for section 2

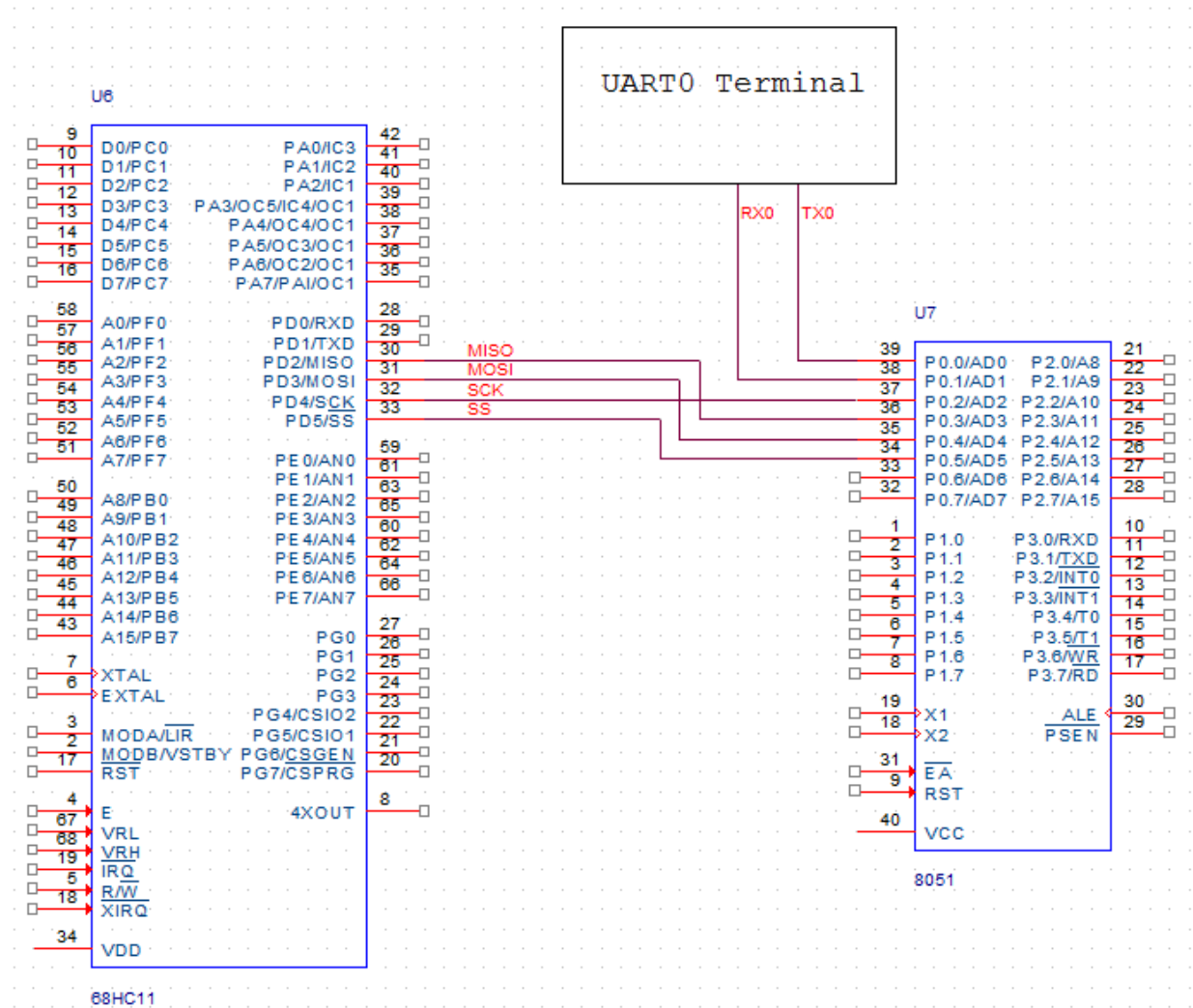


Figure 3: Circuit schematic for part 3

5.4.2 Code for part 3

```
//-----
// 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
char choice;
char scroll_dwn = 0;
char dummy;

//-----
// Function Prototypes
//-----
void main(void);
void SYSCLK_INIT(void);
void PORT_INIT(void);
void UART0_INIT(void);
void SPI0_READ(void);
void SPI0_WRITE(void);
void write_dummy(void);

//-----
// MAIN Routine
//-----
void main(void)
{
    WDTCN = 0xDE;                    // Disable the watchdog timer
    WDTCN = 0xAD;

    SYSCLK_INIT();                  // Initialize the oscillator
    PORT_INIT();
    UART0_INIT();

    SFRPAGE = UART0_PAGE;

    printf("\033[2J");                // Erase screen & move cursor to home position
    printf("Local char typed");
    printf("\033[13;0H");
    printf("Received char\n\r");      // jump down type
    printf("\033[s");                 // save position

    while(1)
    {
        SFRPAGE = UART0_PAGE;
        if(RI0){
            RI0 = 0;
            SPI0_WRITE();
            SPI0_READ();
        }
    }
}

void SPI0_WRITE(void){
    char i;
    char j;
```

```

choice = SBUF0;          // get char from terminal
SFRPAGE = SPI0_PAGE;
SPIF = 0;
NSSMD0 = 0;              // select slave
while(SPI0CFG & 0x80);    // Make sure SPI is not busy
SPIF = 0;                // clear SPIF
SPI0DAT = choice;         // Load char into SPI0DAT
for(i=0;i<100;i++);      // Small delay
while(!SPIF);            // Wait until transmission complete
SPIF = 0;

//Read Dummy
NSSMD0 = 1;              // Release slave
dummy = SPI0DAT;         // Get received char from SPI0DAT (dummy)
//for(i=0;i<100;i++);    // Small delay

if(choice == 0x7F){
    while(SPI0DAT != 0xFF){
        for(i=0;i<100;i++);
        write_dummy();
        SPI0_READ();
        //if(choice == 0xFF) break;
    }
}
else{
    //ANSI formatting
    printf("\033[2;12r");
    printf("\033[2;0H");
    for(j=0;j<scroll_dwn;j++){
        printf("\033[B");
    }

    //Output local char
    printf("Choice is: %c\r",choice);

    write_dummy();
}
}

void write_dummy(void){
    //Write Dummy
    NSSMD0 = 0;          // Select slave
    while(SPI0CFG & 0x80); // Make sure SPI is not busy
    SPIF = 0;            // Clear SPIF
    SPI0DAT = 0xFE;       // Load dummy into SPI0DAT
    //for(i=0;i<100;i++); // Small delay
    while(!SPIF);        // Wait until transmission complete
    SPIF = 0;
}

void SPI0_READ(void){
    char j;

    NSSMD0 = 1;          // Release slave
    //Output dummy
    printf("\033[2;30H");
    printf("DUMMY: 0x%x", dummy);
    SPIF = 0;

    //ANSI format
    printf("\033[14;24r");
    printf("\033[14;0H");
    for(j=0;j<scroll_dwn;j++){
        printf("\033[B");
    }

    //Output choice

```

```

//choice = SPI0DAT;
printf("Data read from SPI0DAT is: %c\r",choice);

scroll_dwn += 1;
if (scroll_dwn > 10) scroll_dwn = 10;

SPIF = 0;
}

//-----
// 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 = 0x06;                          // Enable UART0, SPI
    XBR1 = 0x00;
    XBR2 = 0x40;                          // Enable Crossbar and weak pull-up

    P0MDOUT &= ~0x02; //~0x0A
    P0MDOUT |= 0x35;                      // Set pins 0,2,4,5 to push-pull

```

```

    //P0      |= 0x02;                // RX0 pin and to high impedance

SFRPAGE = SPI0_PAGE;
    SPI0CFG = 0x40;                // Master mode
    SPI0CN  = 0x0D;                // Enable SPI
    SPI0CKR = 0x18;                // SPI clock rate for 995,328

EA = 1;

SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page
}

void UART0_INIT(void)
{
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;        // Save Current SFR page

    SFRPAGE = TIMER0_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
    T10      = 1;                  // Indicate TX0 ready

    SFRPAGE = SFRPAGE_SAVE;        // Restore SFR page
}

```

6 References

“MPS Lab 3,” in RPI ECSE Department, 2016. [Online]. Available: http://www.rpi.edu/dept/ecse/mps/MPS_Lab_Ex3-Serial.pdf. Accessed: Oct. 16, 2016.

“C8051 Manual,” in RPI ECSE Department, 1.4 ed., 2005. [Online]. Available: <https://www.ecse.rpi.edu/courses/CStudio/Silabs/C8051F12x-13x.pdf>. Accessed: Oct. 16, 2016.