Lab 8: Serial Communication with the Simon Board

CpE 214: Digital Engineering Lab II

Last revised: January 1, 2013 (CAC)

In this lab, we will be extending the Simon board's functionality. In the first part of the lab, you will be asked to create code for transmitting and receiving signals mapped to the Simon Board's push buttons through to the RS232 Port. In the second part of the lab, you will verify your design by then reading the signals out using a terminal application like PuTTY. The skills learned will be very valuable in future work and projects.

0.1 Outline and Concepts

1. Introduction to the Serial interface
2. Interrupts Review
3. Writing the C program
4. Testing with Serial Port Monitor

1 Introduction to the Serial interface

The RS-232 protocol was the main way to transmit serially for many years, and is still a staple for many embedded applications in industry. It consists of either 9 pins or 25 pins. In this lab we will be using the 9 pin model. The pins are set up like in Figure 1.

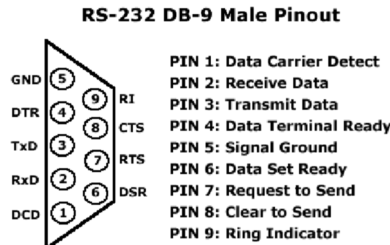

FIG. 1: Serial Pins

In this lab we'll be using the transmit Data pin and the Signal Ground to print ASCII characters over the RS232 port to be captured by the Serial Port Monitor package on Blackboard. Typically embedded systems, particularly from a software perspective, like to work in the ASCII character encoding format. So they convert a random character into whatever hex value ASCII specifies and send that as the byte. When the other end receives it, the high-level software decodes the ASCII and prints the original character out.

2 Interrupts Review

As you should have learned in the lecture, interrupts can be VERY powerful tools in an embedded system programmer's bag of tricks. However, not everything should necessarily be done in an interrupt fashion. Sometimes very tight timing requirements makes using interrupts less desirable, because you have less control of the overhead and extra instructions associated with making an abrupt conditional branch to this area in the code memory. An interrupt typically does these things upon being triggered:

1. Finish executing any last instruction (pipeline flushed)
2. Push Program Counter (PC) onto the Stack
3. Shift the pointers for R0-R7 into another area of RAM, in case the Interrupt Service Routine (ISR) needs to use them
4. LJMP to the Interrupt Vector Table address for the given interrupt
5. Execute the ISR (which usually clears the Interrupt Flag)
6. Shift pointers R0-R7 back to their original positions
7. Pop the old PC location off the Stack, and LJMP to that location
8. Resume program as normal

Take sending an receiving data over a serial port for example. The individual using the device has a lot of control of when data is sent, but very little control of what and when another device might send to you. Therefore, for the purposes of this project, we are going to MANUALLY send bytes of data to our PC terminal, and let Interrupts do the work for us when receiving data. This way we don't have to explicitly write in our main() function extra conditional statements to check for when data is received. This is a huge advantage because sometimes your program may be in an execution area where it is very difficult to be checking often, or at all. Interrupts frees us from this issue altogether.

3 Writing the C program

Write the C code to output any arbitrary letter, i.e., the first four letters of your name, for each of the 4 buttons when pressed to the RS232 port. You will also need to have your code continuously monitor for when a byte of data is received, and light an LED when it does. Start like usual, by creating a new project in Keil as well as remembering to set the create output Hex file option for Target 1. Some useful header information and includes are necessary and available on Blackboard:

- uart.c
- uart.h
- reg932.h

Copy these three files to your project's directory, then add them into the project by right-clicking on Source->Target 1-> Add Files. Then create a main C file called lab8.c, and simply include the following header information:

```
#include "uart.h"
#include "reg932.h"
sbit button1=P2^3; // green button
sbit button2=P2^2; // amber button
sbit button3=P2^1; // yellow button
sbit button4=P2^0; // red button
```

- additionally you'll want to set the buttons to be bidirectional inside your main() with: *P2M1=0x00*. You'll also want to call the uart_init() function as well which sets up the serial port.

Before we go any further, we need to modify uart_init(). In the lectures, you have probably seen that the original 8051 uses Timer 1 to generate a baud rate signal for the UART (serial port), and therefore requires you to setup, configure, and load it first. The Simon board has its own dedicated registers/timer specifically for this purpose called BRGR. The internal RC oscillator is running at 7.3728 MHz. The formula for calculating the BRGR register for a specific baud rate with our oscillator is given by:

$$BAUD = \frac{7.3728 * 10^6}{BRGR + 16} \tag{1}$$

You will need to locate the following segments of code within the uart_init() function of uart.c, and modify them according to these steps. You'll want to have the datasheet handy:

1. Set SCON (serial control register) to transmit and receive. See page 59 of the datasheet.
2. Initially disable BRGEN within the BRGCON register (page 58). Load BRGR with the proper hex value for a 9600 baud rate. BRGR will have to be split into upper (BRGR1) and lower (BRGR0) bytes. Then enable BRGEN and SBRGS in BRGCON.
3. Determine what pins the RxD and TxD are (see page 5 or 29). Then you need to configure P1M1 and P1M2 (page 26) to set RxD as input, and TxD as push-pull (higher current sourcing).
4. Give the serial port highest interrupt priority by setting IP0 and IP0H appropriately (page 23/24). It is numbered interrupt 4 (and thus corresponds to 4th bit in each).
5. Lastly set the global interrupt enable EA, then individually set the serial interrupt ES.

Now back to writing our program:

- *What to do for transmitting a letter?*

Create a simple function in your main that loops and checks for a given button press (active low). Then have it print a letter with the uart_transmit(char) function already defined for you in uart.c.

- *What about lighting an LED when I receive?*

This is pretty simple. In your uart.c file, there is a uart_isr() function that is the ISR itself. Inside it, you will see there are two conditions for the Receive Flag (RI), and Transmit Flag (TI). You want to place your LED code inside the conditional for when the RI flag is high. In our case, the value of the incoming byte is irrelevent; we only need to know when it happens.

4 Testing with Serial Port Monitor

You may use the Serial Port Monitor tool on Blackboard to send and receive information between the board and the PC. But you first need to flash your hex file onto the board.

1. Use FlashMagic to flash your hex file to the Simon Board. Then turn your board to the run position.
2. Open up Serial Port Monitor and choose serial port COM1. Choose Open Port as in Figure 2. Note that you can't be flashing and using Serial Port Monitor at the same time.
3. Verify that you are able to successfully transmit your characters from the Simon Board to the terminal.
4. After you are able to send, go ahead and try sending data to the Simon Board. Simply type a character into the blank and hit Send. Does your LED light?

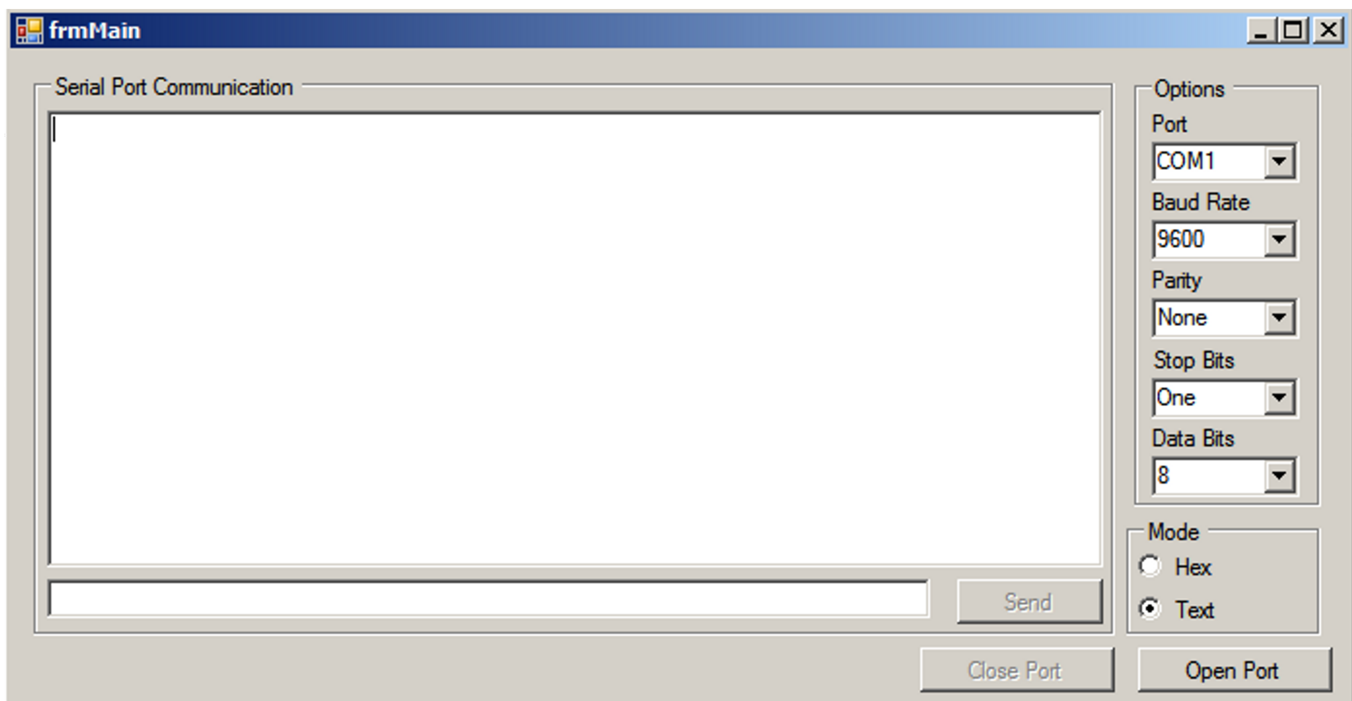

FIG. 2: Serial Port Monitor screen

- *What if it doesn't seem to be working?*

If the problem is you cannot transmit or receive to the board, then clearly your Serial port configuration is not correct (or incomplete). If it's isolated to receiving, then it's possible that the interrupt enable (ES) or global interrupt enable (EA) might have been omitted or cleared. Either way, have your instructor look over it to determine what's going on.

5 Questions (attach at end of your report)

1. Describe in a brief summary how one would go about setting up the FPGA to take the place of the PC terminal for serial communication.
2. Assuming the FPGA can receive the data, what sort of data structures would be convenient to help store and access all of the received characters?