<u>A crash course in RS232 communication with the PIC16F877A</u>

Based on the experiences of Ali Azeem and Peter Hu
ECE 445 Group 20, Class of 2011
Last updated 4/23/2011

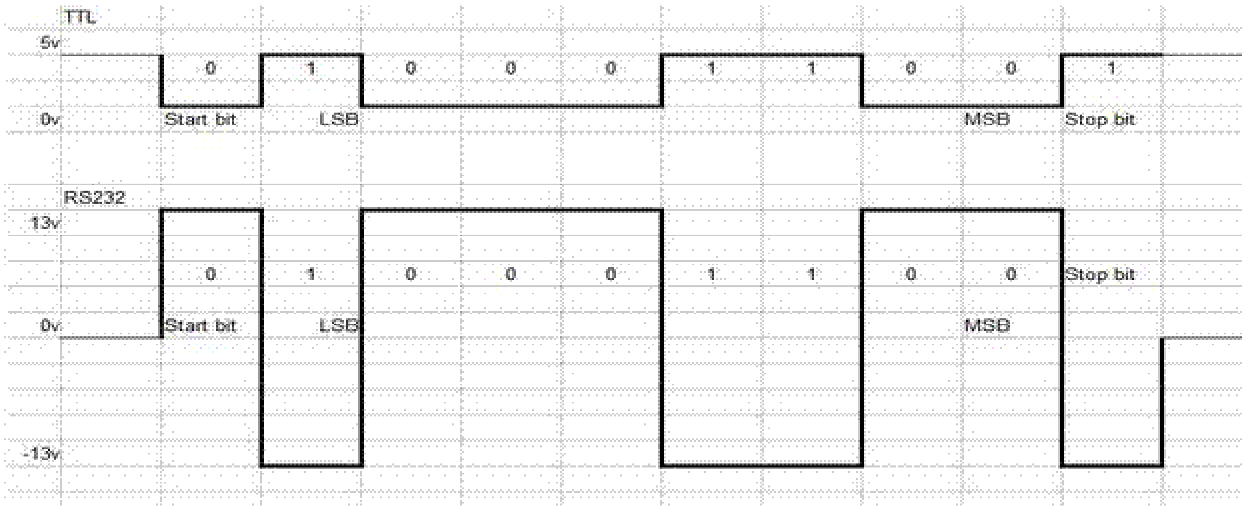
Many senior design projects involve some sort of serial communication between devices. UART/TTL is a serial standard that is incorporated in many devices that you will come across; the PIC16F877A also has a UART/TTL serial interface port. This port will also allow you to communicate with devices that follow the RS232 standard such as a desktop computer (there is a difference, which will be described below). The information in this guide is based on our experience with the PIC16F877A, but should theoretically work with any PIC16 series microcontroller provided everything is set up right.

In our case, we had a RFID reader that received a data request code from a PIC, and sent the corresponding data back to the PIC. This data was then sent out to a computer via RS232.

TTL Standard:
TTL, also called UART or USART, is a serial data transfer standard. Data is sent in the form of ASCII characters along with start and stop bits. The voltage levels are the standard 0 volts for “off”/“low”/“0”, and 5 volts for “on”/“high”/“1”. If I were to send the letter

RS232 Standard:
This is similar to TTL and is the standard that computer serial ports follow. The main difference between RS232 and TTL is the logic level scheme. For “off”/“low”/“0”, the voltage level is ~13 volts. For “on”/“high”/“1”, it is about -13 volts.

TTL/RS232 Comparison. Note the positions of the LSB and MSB.



Level matching:

Due to this inversion and voltage difference, you will not be able to send data via TTL and have an RS232 device decipher it; strange things will happen. To compensate, you will need an IC called MAX232. The MAX232 will function as a voltage level regulator between the TTL and RS232 devices; if you send data into the TTL side, it will step the voltage up and invert it according to the RS232 standard. Similarly, if you send data from an RS232 device, this IC will step down the voltage and invert it accordingly. The parts shop had these ICs when we took the course, but they are pretty cheap anyways. If you are so inclined, you could do this manually (via OP amps or even just BJTs), but we did not see any real benefit in doing so. We wired up the MAX232 according to this diagram and had no problems whatsoever. http://sodoityourself.com/wp-content/uploads/2007/02/circuit_232.jpg, "uC serial" refers to the microcontroller's TX/RX pins. Make sure you are soldering to the correct pins on the DB9 connector.

Interconnection:
When interconnecting devices, bear in mind the that "*whatever is transmitted by one device is received by the other*". So if I were to send data between two devices, I would interconnect the TX and RX pins. If device 1 sends something, it is sent out from the TX pin. For device 2 to read this data, it must be read via the RX pin. In the case of the MAX 232, treat it as an 'adapter' or passive device. We connected it according to the diagram in the previous section and had no problems.

Baud rate:
This is the rate at which data is sent. It is measured in bits/second. For example, 38400 baud = 38400 bits/second. Hence each bit has a time span of 1/baud rate. Make sure baud rates of devices match. The most common baud rate for devices is 9600, so we used that for some time, only to realize that our device ran at 38400.

Other parameters:
RS232 also has the capability of adding parity bits (in which case, we assume there will be 7 data bits instead of 8), but we did not incorporate any parity bits as the majority of devices do not have parity bits and it adds unnecessary complexity. We assume (having not done this ourselves) that if you add parity bits, just make sure the sending/receiving devices both have this taken into account.

Sending data:
The following preprocessor directive is needed to define an output stream for the PIC
```
    #use rs232(baud=38400, xmit=PIN_C6, stream=PC)
```
This treats the output stream "PC" as if it were a variable. For most cases, you actually will not need to use the "PC" identifier. To output data via the TX pin of the PIC, a.k.a. PIN_C6, use a fprintf statement, the syntax of which is similar to a standard printf statement. :
```
    fprintf(PC, "Hello World!\n");
```
Open PuTTY and select serial, and specify the appropriate baud rate. If everything is connected properly, you should see the text on the screen.

Reading data:
The following preprocessor directive is needed to define an input stream for the PIC.
```
    #use rs232(baud=38400, rcv=PIN_C7, stream=READ)
```
To read a letter use the `fgetc(stream)` function:

Example:
```
    char group20isawesome;
    char a[10];
    group20isawesome = fgetc(READ); //gets the first char
    printf("%c", group20isawesome);

    for(i=0; i<10; i++){
         a[i] = fgetc(READ); //gets a string of 10 characters
         }
```

Manipulation of data:
Since we now have a character array with our data, we can print the string, certain characters, etc. The data type is char, so if you want to do mathematical manipulations you will have to cast the array values (perhaps with a loop) as int, float, etc.

Example: Char to int casting
```
    char c = 'A';
    int x = (int)c;  //Casting from char to int. The value of x
    becomes 65: the ASCII code of 'A'
```

Additional information/tips:
For projects that require reading/writing from multiple serial ports, take a look at the PIC16LF1527, PIC16F1946, and PIC16F1947 (same as PIC16F1946, but with more memory). These all have two UART ports. Microchip gives out free microcontroller samples (Google "microchip samples") from their website provided you use a .edu email address, and they ship pretty quickly. Bear in mind that these come in square packages (make sure you get the TQFP64 packaged devices), and you will either have to purchase or make your own TQFP64-to-DIP adapter. If you do decide to make your own adapter, check with Mark Smart in the parts shop first, some students have done so in previous years, so the parts shop might still have their design files.

The RX and TX connections do not have to be in a closed loop. During our tests, we used one PIC16F877A to output data which was sent to the RX pin of a device (data request code). In response, the device would send data out of its TX pin, which was read into a <u>different</u> PIC16F877A, which then sent it out to the computer.

If you are looking at your data stream on an oscilloscope, use a lower baud rate. From our experience, if you send data at 9600 baud, each data bit is ~104µS. There is a trick to doing this however. Have your oscilloscope time sweep speed set at a speed lower than your baud rate. Assuming your screen is set such that you can see where "high" and "low" should be, you will see what appears to be a blur of squares moving. Pause the oscilloscope ("run" button in our case) and then change the time sweep to zoom in. If you try to pause it while it is running, you will only see a few data bits. Remember, RS232 will be inverted, so we suggest doing this on the TTL side to avoid confusion.

If you use a RS232 - USB adapter, make sure you know what the port name is (ex. "COM5"). This can be found in the Device Manager in the Control Panel.

If you are using Eagle to design your PCB, you will have to "invoke" the MAX232 IC in your schematic to allow you to connect things to pins 15 and 16.

Helpful link on PIC C data types, operators, etc.
http://www.swarthmore.edu/NatSci/echeeve1/Ref/C%20for%20PIC/C_Intro.html

Good luck!