Lab Assignment 8—*"The Name is Baud…, James Baud"*
RS-232 Communications Server

Overview:
In this lab you will use the ReadyAVR to create a simple **RS232 communications server**, and you'll communicate with it using the PuTTY terminal program.

Lab Procedure:
Lab07 introduced you to using the USART, so I'm not going to walk you through every step of this lab, but rather give you the requirements and let you work out the details. Here they are:

1. You must create a **communications server** capable of duplex communication using any of the following parameter settings (selectable via `#define` statements at compile time):

   baud rates: 9600, 19200, 38400, or 57600
   data bits: 7 or 8
   stop bits: 1 or 2
   parity type: 0 (none), 1 (odd), or 2 (even)

2. Communications should be strictly *asynchronous* (i.e. no handshaking), and PuTTY must be set up as a "generic" RS232 terminal. Ensure this by configuring PuTTY to **<u>not</u>** send implicit Carriage Returns (CRs) with Line Feeds (LFs), to **<u>not</u>** send implicit LFs with CRs, and by setting PuTTY's Local Echo to "**Force off**."

3. Upon power up or reset, your communications server must display the following status message on PuTTY's terminal screen:

   ```
   RS232 Communications Server
   Status: Active
   Firstname Lastname
   EECE3624
   Fall 2022
   ```

   Use your name for `Firstname Lastname` and transmit **two CR-LF <u>pairs</u>** ***before*** the first line and **two *after* the last line** (this creates a blank line above and below the status message). Except for the name, your output must look **<u>identical</u>** to that shown above.

4. After the status message, whenever a key is typed in PuTTY your communications server must acknowledge receiving the character by turning LED0 on for **75mS**, then turning it back off again. Also, the received character must be echoed (transmitted) back to the terminal server exactly as received, **with three exceptions**:

   a. When a CR is received and sent back, you must also send a LF character.

   b. If the "**Esc**" key is pressed, the communications server must display the **status message** on the terminal screen as shown in 3. above. This is called a "soft" reset (as opposed to a "hard" reset like pressing the on-board Reset button).

   c. If the user types the three characters `end` in succession <u>with no intervening characters</u>, after echoing the "`d`" character the communication server must send **two CR-LF pairs** followed by the word "**`Bye`**" and then <u>stop responding to any more input until the ReadyAVR is restarted via a "hard" reset</u>.

As part of this lab, you must use the same `UARTLibrary.h` file you were given in Lab07, **but you must develop your own UART functions**. In other words, **you are not allowed to use or include the precompiled `libUARTLibrary.a` file from Lab07 in your final Lab08 program, but must instead write your own functions**.

Accept the GitHub classroom link, and make a local repo in your existing solution directories. Then add this to your top level solution. There is a prototype file, `UARTLibrary.c`, which has empty versions of the functions you need to add to match the `UARTLibrary.h`. Be sure to read this header file as it has some of the details of what the functions are and shows exactly how they must be called. The `Lab08.c` file is ready to accept your main code. **Do not modify the `UARTLibrary.h` file in any way** since it's essentially the interface requirements document for your new set of UART communication functions.

While your final submission cannot rely on the `libUARTLibrary.a` file, **you *are* allowed to use it for testing and debugging**. (Remember, to use the precompiled library you must add it to your project as described in Lab07.) But you should only use it for development purposes since I will only use your `UARTLibrary.c` file to test your code! So remove `libUARTLibrary.a` from your project before you begin your final testing. **If this isn't clear, or you're unsure about something, ask**.

Once your communications server is up and running, configure PuTTY to match the comm port settings of your program. Use `#define` statements in your Lab08 C program to set the given baud rate, parity, etc. (the `UARTLibrary.h` header file shows an example of how to do this), and connect PuTTY to your communications server. Also, don't forget to turn off "Local echo" in PuTTY and make sure PuTTY doesn't implicitly add any CR or LF characters to what's been typed by the user. Once PuTTY is connected to your server, any characters you type should be echoed back to PuTTY, with the exception of the three cases mentioned above. As mentioned in Lab Procedure #1, you will need to test three different settings for baud rate (you are welcome to test more!). As we discussed in class, you cannot change baud rate on the fly, so you will need to change your `#define` parameters and change the PuTTY settings to test different configurations. We sure to discuss which speeds you tested and any interesting differences you saw.

This is a **one week lab** and must be demonstrated at the beginning of lab on **Friday, November 4th**. For this lab, no lab report is required. **You must upload your properly formatted and commented `main.c`, `UARTLibrary.c`, and `UARTLibrary.h` files to the GitHub assignment repo**, in addition to showing this lab working to get full credit. Extra credit points are available for a full lab report.