# ECGR 4101/5101, Fall 2015: Lab 6

## UART and Baud Rate Detection

Version 1.2 – 2015-11-12

**Objective:**

In this lab, you will create your own library for interfacing with the UART (Universal Asynchronous Receive/Transmit) on the RX63N. You will create functions for initializing the UART, transmitting bytes, and receiving bytes. Once your UART initialization code is complete, you will then attempt to detect the baud rate of a transmitting device.

**Background Information:**

While most microcontrollers have a hardware module dedicated to UART alone, the RX63N has a multipurpose serial communications hardware module call SCI (Serial Communications Interface). The SCI can be configured to operate as a UART. Be sure to read the section about SCI in the RX63N Hardware Manual.

In this lab we will be creating functions for initializing the UART with specific parameters, transmitting bytes through the UART, and polling for received bytes. For incoming data, most UARTs have a receive data flag in one of the registers which will be set when there is new data to be read. To poll for new data, the program must constantly check the receive data flag regularly, then read the data if it is available.

NOTE: The RX63N does not specify a receive data flag, but Bit 6 in the Serial Status Register is set to 1 when new data has arrived. In the datasheet, Renesas has labeled this as a "reserved" bit, however, it does function as a RX flag as long as the receive data interrupt is enabled.

**Assignment:**

1. Begin by creating an empty project workspace under the YRDKRX63N Simple Demos option.
2. Create functions for interfacing with the UART module. You may create your own, use functions from the Renesas peripheral driver library, or follow steps 3 – 10 to create a simple UART initialization, receive, and transmit function. You will not be graded on following the steps provided in 3-10. You may also want to look at some of the Renesas example code for help.
3. Once the project is set up, go to file->new to open a blank file. We will save this file as "customUART.h" under the src directory in your workspace. We will then open another blank document and save it in the same directory as "customUART.c". We do this because libraries will usually have a .h file to accompany a .c file.
4. Now we will populate the .h files with the function prototypes. Header files contain #define macros, function prototypes, and sometimes, but not often, entire functions.

Under normal circumstances, functions should only be written within the .c file which the header file corresponds to.

5. In your header file, include the board's header file
    a. #include "platform.h"
6. In your header file, create 4 function prototypes:
    a. void initUART(int baud, int parityEnable, int parityMode, int stopBits); // This function initializes the UART for us
    b. int newDataAvailable(void); // Checks the UART for new data
    c. char rx_UART(void); // This function reads the UART data register
    d. void tx_UART(char); // This function transmits data through the UART
7. In the customUART.c file, you will need to implement each of the functions described in the header file above.
    a. initUART() will configure the SCI (Serial Communications Interface) module for UART transmission at the specified baud rate passed to the function. It will also configure the UART to use 8 data bits. The function is also passed variables for the number of stop bits, parity mode, and parity type (even or odd). The UART also should not be configured to use any interrupts as well (Except for Receive Data for setting the new data flag), as we will be software polling for new data.
    b. newDataAvailable() will check the Receive Data Full Flag bit from the Serial Status Register (Bit 6). This flag bit is set when new data is ready to be read from the Received Data Register. This function will return a 1 when new data is present, and a 0 otherwise. Also note, there is a bit called ORER in the SSR register, which is used to detect overruns in data reception. If an overrun is detected, then it will be set to 1 and stay that way until manually cleared. When ORER is 1, the UART is unable to receive more data. In the newDataAvailable() function, I suggest clearing this bit at the beginning of the function.
    c. rx_UART() will read and return the data from the Received Data Register.
    d. tx_UART() will transmit a byte passed to it by writing it to the Transmit Data Register. Before writing to the register, the function will check the Transmit End Flag before writing to the Transmit Data Register. If the flag is high, indicating that data is still in the process of being transmitted, the function will wait for it to clear before writing to the register.
    e. Don't forget to include customUART.h in your customUART.c file.
8. To test your functions at this point you will need to connect your board to a computer using a RS-232 serial cable. Note that RS-232 is not the same as UART. While the communication protocol is the same, RS-232 uses voltage levels of -3 to -15 volts to express a 1, and +3 to +15 volts for a 0. UART implies TTL (Transistor-Transistor Logic) levels, which is 0 to 3.3 or 5 volts for 0's and 1's, respectively.
9. Once you have your board connected to a PC via RS-232 cable (you may have to use a RS-232 to USB adaptor if your computer does not have a serial port), you will need to have a serial terminal program ready to receive data from your board. If you aren't already using one, I would recommend "CoolTerm", which is a freeware program available online, for its simple interface.

10. Now that you are ready to connect, we need to write some test code to transmit and receive bytes. In your main() function, write code to initialize the UART at 9600 baud and transmit a character of your choosing when switch 1 is pressed. For example, inside of the `void sw1_callback(void)` function you can call `tx_UART('A').` To test the reception, call newDataAvailable() inside the while loop, and if new data is available, print the received data to the LCD.
11. Now that you have a UART interface working, write code to detect the baud rate of incoming data bits. At check out, the TA will connect an RX63N board to yours via RS-232 cable. The TA's board will be transmitting an unknown character bit at an unknown baud rate, unknown parity, and 1 stop bit. Transmissions will occur every 0.1 seconds, and the baud rate will be no faster than 119,600 baud and no slower than 150 baud. Only "standard" baud rates will be used (150, 300, 600, 1200, 2400, 4800, 9600, 14,400, 19200, 28,800, 31,250, 38,400, 56,000, 57,600, 115,200). Once the code has detected the baud rate and the character, the LCD should display on the first line the detected baud; on the second line the detected parity, and on the third line, the detected character.

**To Submit:**

- A zip file containing the workspace from this lab.
- Your lab check-off sheet at the demonstration

Grading will be changed such that 70% is your demonstration and 30% will be your code structure. Your code will be graded based on comments, proper alignment (indentations), function design, proper capitalization of definitions and variables, and design of custom functions.

# Embedded Systems Lab Demonstration Validation Sheet

This sheet should be modified by the student to reflect the current lab assignment being demonstrated

| | |
|---|---|
| Lab Number: | Lab 6 – UART Baud Rate Detection |
| Team Members | Team Member 1:<br>Team Member 2: |
| Date: | |

## Lab Demonstration Requirements

| REQ Number | Objective | Self-Review | TA Review |
|---|---|---|---|
| 1 | UART is able to initialize properly, transmit and receive. | | |
| 2 | The correct baud rate and parity is detected from a transmitting board. | | |
| 3 | The correct character is recognized from a transmitting board at an unknown baud rate. | | |

## Code Requirements (will not be graded during lab demo)

| REQ Number | Objective | Self-Review | TA Review |
|---|---|---|---|
| 4 | All code must be commented and indented properly. | | |