ECGR 4101/5101

Lab 5: UART and Queues

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. You will also create a queue structure for storing received characters and displaying them onto the development board's LCD.

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.

Assignment:

1. Begin by creating an empty project workspace under the YRDKRX63N Simple Demos option. Consult the supplemental instruction from the previous lab if you need help doing this. If you do, at the step where you select "Tutorial Project", select "Empty Application" instead and click "Finish."
2. 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.
3. 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.
4. In your header file, include the board's header file
    a. #include "platform.h"
5. In your header file, create 4 function prototypes:
    a. void initUART(int baud);// 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
6. 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, 1 stop bit, and no parity. The UART also should not be configured to use any interrupts as well, as we will be software polling for new data.
   b. newDataAvailable() will check the Receive Data Full Flag bit from the Serial Status Register. 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.
   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 Data Empty 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.
7. 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.
8. 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.
9. 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.
10. After you have tested your functions, it is time to include a queue structure. From the website, download the queue.c and queue.h files and include them in your project. Read the files and consult the notes from class on how to use the queue structure.
11. For the final demonstration, construct your main() such that:
    a. When characters are transmitted from the PC through the terminal program to the RX63N board, the characters are then stored in a queue.
    b. Set the maximum queue length to 10. When switch 1 is pressed on the board, have the contents of the queue printed to the LCD and dequeued.

c. If switch 2 is pressed on the board, have the tail of the queue dequeued, and transmitted back to the PC.

To Submit:

- A pdf containing your project report. Paragraphs describing the objective, what you did to complete that objective, and code snippets throughout the description as they are relevant. When describing the library function you created, be sure to explain any design decisions you made.
- Your main.c, customUART.c, and customUART.h files
- A zip file containing the 4 files mentioned above
- Your lab check-off sheet at the demonstration

Grading will be changed such that 40% is your demonstration, 20% will be your code structure, and the remaining 40% will be your report. 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 5 – UART and Queues
Team Members	Team Member 1 Name Here Team Member 2 Name Here
Date:	MM/DD/YYYY

Lab Requirements

Obtain a list of the Lab requirements from the end of the lab handout and type them here, perform a self-review and indicate with an X if you met each requirement or not.

REQ Number	Objective	Self-Review	TA Review
1	Characters are received and stored in the queue		
2	Characters are properly displayed on the screen and dequeued when switch 1 is pressed		
3	Characters are properly transmitted to the PC and dequeued when switch 2 is pressed		

Comments

If you were unable to meet a particular requirement or if a requirement required a custom implementation describe briefly why you were unable to meet the requirement or what your custom implementation was in the box below, also if you have any additional considerations that you would like to be taken into account while grading you may type them here as well.
