Prelab 7: Asynchronous Serial Communications

Table of Contents

[Goal and Background Information 2](#_Toc86093059)

[Data Flow Diagram 2](#_Toc86093060)

[Control Flow Diagram 3](#_Toc86093061)

[Implementation 3](#_Toc86093062)

[getstrU1 CFD 4](#_Toc86093063)

[Testing and Verification 4](#_Toc86093064)

[Conclusion 5](#_Toc86093065)

# Goal and Background Information

The goal of lab 7 is to implement an asynchronous serial communications using the UART or Universal Asynchronous Receiver-Transmitter that is mounted on the PIC32. This will be combined with the code created in lab 5 and 6 to make a mock factory. The stepper motor will be operated by both a person on the floor and a person sitting at a terminal communicating with the PIC32. The stepper ‘operator’ can change the stepper behavior by pressing buttons while the person using the terminal needs to input commands. The PIC32 and the terminal used will be communicating at a rate of 19.2 kbaud using RS232.

The UART is a type of microprocessor that translates data between serial and parallel forms, usually between two devices that need to communicate. The UART that will be used in this lab is capable of full duplex communications. The Cerebot MX7CK board in the lab can host up to two UARTs at a time. The UART will use a communication standard called RS232 that has standard pins, data rates, and voltage levels. More recently, RS232 has been made so that there are lower voltage thresholds than when RS232 was originally made. This allows for 5-volt powered logic to be able to control RS232 communications directly. RS232 is useful for one-to-many communications and for where there are only a few devices connected (like this lab). RS232 is not useful when there are many devices talking to many other devices on a single line. It should be noted that the Cerebot board will not be using the standard DB9 connector for RS232 but will be using USB connectors instead.

## Data Flow Diagram

**Diagram

Description automatically generated**

## Control Flow Diagram

**Diagram

Description automatically generated**

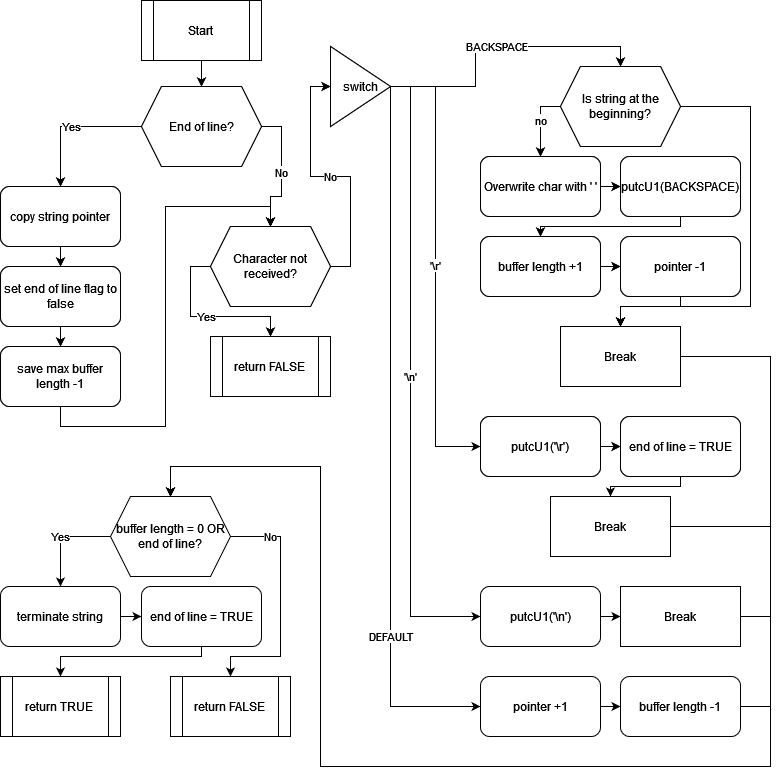
# Implementation

The implementation of the code for the lab went as planned. The most difficult part of the lab was moving the functions and ISRs into their respective libraries. There were several user introduced issues while doing this. A large portion of these issues were because of incorrectly defining extern variables within the library files and incorrectly included header files. The use of extern variables is first introduced in this lab and allows for different files to share variables with each other. Extern variables require that one file declare an extern variable with an initial value and the other functions that use the variable declare the extern variable with no initial value. This will tell the linker that the extern variable is declared somewhere else and that the variables use the same memory location. It should also be noted that a control flow diagram for the getstrU1 function is located at the end of the implementation section.

The UART receive functions are in the background because they are in a while loop collecting information from the Putty terminal and waiting for a null character. If the UART receive functions were in the foreground then things that need to happen while the program is waiting for a null character, like the change notice interrupts and the stepper motor timer interrupts would not happen if they were a lower priority. Another consequence of having the UART receive functions in the foreground is that if a null character was sent terminating an input string and you began immediately typing again the receive functions would be given priority over interpreting the previous string. This would force the CPU to go back to waiting for a null character instead of interpreting the data entered.

The \_mon\_putc function is used to set up the serial port to function as the console when using the printf function. This allows for an ease of uses when communicating using RS232 because the person writing the code can use the printf function instead of using the putsU1 function from the comm.c library.

## getstrU1 CFD



# Testing and Verification

This lab was validated using observation. This included observation of the stepper motor, LCD screen and the putty terminal on the PC. The stepper motor was timed using a wristwatch to estimate the RPM to ensure that the program was calculating the correct delay times for a specified RPM. This was done by counting how many revolutions the stepper motor made in 15 seconds and multiplying by four. The LCD screen was checked that it was displaying the correct information about the stepper motor, direction, step mode, and speed. It was also checked that it changed information depending upon user input through the buttons and the putty terminal. Lastly, the putty terminal was used to send strings to the program and change the stepper motor’s direction, step mode, and speed. This was done by using the putty and ensuring that the program echoed back the input string with a message counter, the LCD screen reflected the inputted string, and the stepper motor changed its motion to match the motion described by the input from the string.

The code for this lab was broken down into four subsystems. The LCD subsystem to control the LCD, the UART subsystem to handle communication between the PIC23 and the putty terminal on the PC, the buttons subsystem for the button input, and the stepper motor subsystem for the stepper motor. The LCD subsystem included all the necessary functions to setup and write information to the LCD screen. This includes things like a function to setup the PMP and then the LCD screen, write to the LCD, read from the LCD, and see if the LCD controller is busy. The UART subsystem was implemented in a similar fashion as the LCD subsystem in that it contains the necessary functions to setup and use the UART. The UART subsystem contains functions to setup the UART for RS232 communication over USB, read a information using RS232, and output information using RS232. The buttons subsystem includes functions to setup the change notice interrupt for the buttons used. It also included the functions used to read and decode the button input into useable information. The stepper motor subsystem includes the functions needed to setup the timer 1 interrupt, the necessary FSM to control the stepper motor behavior, and the function to send output signals to the stepper motor.

The process of breaking down the code into the four subsystems started with all the functions being in the main c file. Then a header and c file were created for the subsystem that I intended to modularize. After that, the functions that needed to be in the subsystem were copied and pasted into the subsystem c file and the function prototype moved into the subsystem header file. The necessary actions of declaring extern variables that the main c file and the subsystems could share were implemented as needed along the way. After each function was moved from the main c file to its corresponding subsystem the system was reuploaded to the PIC32 and the PIC32 was checked to ensure that it displayed the proper behavior. If after moving a function into its corresponding subsystem and the PIC32 was not behaving as expected function that was relocated was further debugged. This limited debugging time by having only one issue come up at a time when each function was moved. If the PIC32 behaved as expected the next function would then be migrated. The process of moving one function to its subsystem one at a time and debugging any issues that arose from moving the function repeated until all the functions that were needed in each subsystem were where they needed to be.

# Conclusion

There are advantages and disadvantages to parallel communication and serial communication. Parallel communication is faster because more than one bit of data can be transmitted at a time, but it requires a dedicated line for each bit of data. Serial communication can only send one bite of data at a time, but it requires only 2 wires. A good example of parallel communication is the communication between the PMP and the LCD controller. The PMP sends a byte of data to the LCD controller using 8 lines in parallel fashion. An example of serial communication is the RS232 communication between the UART and the putty terminal on the PC. The PMP can send 8 bits of data as fast as the LCD controller allows. This means that every enable cycle the LCD controller sends/receives 8 bits of data. Compared to the RS232 serial communication, that can only send 8 bits of data using 8 ‘communication cycles’ on one line.