LAB 5

COMMUNICATION USING THE UART AND WIFI INTERFACES AND INTERRUPTS

INTRODUCTION

This week in lab you will fully configure the UART for serial communication between the CyBot and PC. UART stands for Universal Asynchronous Receiver/Transmitter and is a standard component of microcontrollers. A UART provides asynchronous serial data communications for compatible devices. In Labs 3 and 4, you became familiar with the use of the UART on the TM4C123 microcontroller, as you were given precompiled code in a library and used functions for initializing the UART and sending and receiving bytes of data.

You may want to quickly refamiliarize yourself with what you did with the UART in Labs 3 and 4.

- Refer to Part 4 of Lab 3, in which you sent messages between PuTTY on the PC and the CyBot using the RS232 serial cable and functions in the cyBot_uart.h header file. These functions were implemented for you in the libcybotUART.lib library.
    - While optional, you might have tried to drive the CyBot from the PuTTY interface, and you might have tried to disconnect the serial cable and use the WiFi interface instead.
    - Refer to the "UART and WiFi Board Quick Reference Sheet" (UART and WiFi Board Quick Reference Sheet.pdf) for instructions.
- In Part 1 of Lab 4, you confirmed that what you did in Lab 3 with the UART still worked (always nice to get something working that worked before). Then you implemented some of the UART initialization code to set up the GPIO alternate function for the UART interface. Thus from Lab 4, you have some of the initialization code already written.
    - You also used precompiled functions in the cyBot_Scan.h header file, implemented for you in the libcybotScan.lib library, to collect sensor data from the CyBot and send it to PuTTY.
    - You might have tried to use the WiFi interface.

You can use your work from Labs 3 and 4 to get up and running in this lab. However, you will be replacing the cyBot_uart.h and libcybotUART.lib files with your own code in uart.h and uart.c files. In other words, you will be implementing the UART functions that you were using from a library in previous labs.

In this lab, you will fully configure UART1 and write polling (busy-wait) functions for sending and receiving bytes of data. That should round-out your understanding of UART device programming. Then you will get familiar with a key concept in embedded systems – interrupts. You will remove the polling function for receiving data, and instead enable an interrupt when a byte is received in the UART. Doing so, your code will not have to busy-wait in a loop waiting for a key to be pressed in PuTTY, possibly ignoring other work to do, and instead your code will be notified when the character from the keystroke has been received. This functionality will be very useful for the lab project.

BACKGROUND

What does it mean for data transfer to happen asynchronously? Asynchronous means that the transmitter and receiver do not share a common clock. In other words, the sender transmits data using its own timing source,

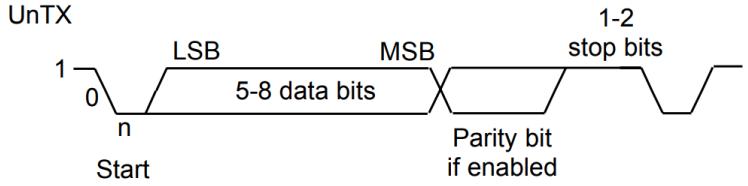
rather than sharing a clock with the receiver. Conversely, for synchronous data transfer, both the transmitter and receiver access the data according to the same clock. Therefore, a special line for the clock signal is required. For asynchronous communication, i.e., UART, the transmitter and the receiver instead need to agree on a data transfer speed. Both transmitter and receiver set up their own internal circuits to make sure that the data transfer follows that agreement. In addition, synchronization bits are added to the data by the transmitter and used by the receiver to correctly read the information.

Data communications are serial, which means that information is sent serially as a sequence of bits. The bits are grouped together to form serial frames. A serial frame can contain 5 – 8 bits of data, and there are synchronization bits that distinguish the start and end of a frame. There is also the option of a parity bit to provide simple error detection over a communication medium that may be noisy.

Serial communication sends one bit at a time on a single wire. The number of bits that can be transferred per second is indicated by baud rate (bit rate). This tells us the speed of communication between the transmitter and receiver.

Below is a diagram that depicts how data are transferred. Information about baud rate generation can be found in section **14.3.2** of the **Tiva Datasheet**. Section **14.3.3** describes in more detail how data transmission works.

Figure 14-2. UART Character Frame



Communication requires at least one transmitter and receiver, or two endpoints (one transmitting and one receiving). The CyBot is one endpoint, and your lab computer will be the other endpoint. The lab PC uses a PuTTY terminal to facilitate communications from the PC end. The PuTTY software talks directly to the serial port (i.e. COM1) of your PC. As you type characters in the PuTTY terminal, ASCII characters will be sent to the serial port and over the wire in the cable.

Both endpoints must use the same frame format and baud rate. Frame format indicates the number of bits in the frame and consists of a start bit, data bits, an optional parity bit, and 1 – 2 stop bits. The standard number of data bits in a frame can be 5, 6, 7, or 8. The parity bit can be odd, even, or none.

To generate the transmit clock in the UART (the rate at which it transmits bits on the TX wire), the UART uses a 16 MHz system clock and a clock divisor of 16 (a configurable parameter in the interface). There are two registers in a UART for configuring the baud rate: UARTn_IBRD_R and UARTn_FBRD_R. These registers combine into a 22-bit register used to calculate baud rate, i.e., a 16-bit integer part (IBRD) and a 6-bit fractional part (FBRD). Note that these represent a 22-bit binary fixed-point value. The equations for determining the integer and fractional values for these registers are given in the datasheet, textbooks, and other resources.

Code examples for sending/receiving data are in **Figure 8.73** on **page 662** in the Bai textbook and in class materials.

Reference Files

The following reference files will be used in this lab:

- lab5_template.c, contains a main function template that you will implement for this lab
- lcd.c, program file containing various LCD functions
- lcd.h, header file for lcd.c
- timer.c, program file containing various wait commands
- timer.h, header file for timer.c
- cyBot_Scan.h, header file for pre-compiled library for CyBot sensor scanning
- libcybotScan.lib: pre-compiled library for CyBot sensor scanning (note: must change extension of file from .txt to .lib after copying)
- TI Tiva TM4C123G Microcontroller Datasheet
- TI TM4C123G Register Definitions C header file: REF_tm4c123gh6pm.h
- Cybot baseboard and LCD schematics: Cybot-Baseboard-LCD-Schematic.pdf
- UART and WiFi Board Quick Reference Sheet: UART and WiFi Board Quick Reference Sheet.pdf
- GPIO and UART register lists and tables: GPIO-UART-registers-tables.pdf
- Reading guides for GPIO, UART, and interrupts, as needed

The code files are available to download from a Google Drive folder:
https://drive.google.com/drive/folders/1LPneqbbOkbS_1rPpEeJsEhUi4gdsLAk5?usp=sharing

In addition to the files that have already been provided for you, you will need to write your own **uart.c** file and **uart.h** file and the associated functions for setting up and using UART. Separate functionalities should be in separate functions for good code quality and reusability. This means that in your uart.c file you should write separate functions for initializing/configuring UART, sending characters, and receiving characters. Remember to use good naming conventions for function names and variables. For example, you may want to name your UART initialization function **uart_init** as there may be other initialization functions you will write in later labs that will eventually have to be used together. Minimally, we recommend defining the following functions:

```
void uart_init(void);
void uart_sendChar(char data);
char uart_receive(void);
```

Prelab

See the prelab assignment in Canvas and submit it prior to the start of lab.

Structured Pairing

You are expected to continue to use structured pairing in this lab and in future labs. It was introduced in Lab 2.

Part 1: Receive and Display Text

Implement the functions for the prototypes given in uart.h for serial communication using UART1 via Port B.

In order to set up UART communication, you should review section **14.4 Initialization and Configuration for UART** in the **Tiva Datasheet**. This will walk you through step by step setting up the necessary registers.

Additionally, **Figure 8.73** on **page 662** of the Bai textbook provides a good example for initializing, transmitting, and receiving using the UART. **Other class resources also have examples.** You will use a baud rate of 9600.

Use the instructions in the UART and WiFi Board Quick Reference Sheet for testing your program running at 9600 baud using the serial cable.

Write a function that will receive characters from your computer's PuTTY terminal window. The program should display the entire series of characters on the LCD when 20 characters have been received, or when you press ENTER ('/r'). When you press ENTER, you should not print out the character ('\r') on the LCD.

For debugging purposes, you should display each character on LCD line 1 as you receive it. You will also need to buffer these characters (store them in an array) as you receive them and prior to displaying them all together. Remember you will display the entire series of characters after 20 characters have been received, or after ENTER is pressed. As each character is received and placed into the buffer, a number should appear on the LCD that indicates how many characters are currently stored in the buffer, and you should print the character received. Once 20 characters have been received, clear the display prior to printing the entire series of characters on LCD line 1.

Don't worry about editing your series of characters. You don't have to manage your buffer to accommodate backspace or delete. However, if you are looking for an additional challenge, you may want to explore this feature.

CHECKPOINT:

Characters should be displayed on the LCD when 20 characters have been received (sent from PuTTY to CyBot) or ENTER is pressed.

PART 2: CHARACTER ECHO

In part 1, your focus was on receiving characters on the microcontroller. This part will focus on transmitting characters back to PuTTY. You will write a program to transmit each received character back to PuTTY.

Before beginning, turn off local echo on PuTTY by selecting **Terminal → Line Discipline Options → Force off local echo and local line editing**. When 'enter' is pressed only a **Carriage Return ('\r')** is sent by PuTTY. When transmitting back to Putty, you should send a **Line Feed ('\n')** following the Carriage Return. This will tell PuTTY to advance to the next line of the display.

CHECKPOINT:

The bot should transmit (echo) each character it receives back to PuTTY to be displayed.

PART 3: USING WIFI

The WiFi board is configured to operate only at a baud rate of 115,200. Recalculate the baud rate parameters for 115,200, and verify that your code still works at this new baud rate using the RS232 serial cable.

Use the instructions in the UART and WiFi Board Quick Reference Sheet for testing your program running at 115,200 baud using WiFi.

In addition to receiving and echoing characters from PuTTY, send messages to PuTTY from your program. Use the uart_sendStr() function to send messages to PuTTY. You can choose what messages to send, when to send them, etc. – you may want to play around with this.

CHECKPOINT:

Verify round-trip communication between the CyBot and PuTTY using WiFi. The bot should be transmitting each character it receives back to PuTTY to be displayed. Messages from the bot should be displayed in PuTTY.

Sketch a flowchart that illustrates the communication behavior of your program, showing the flow of control for receiving a character from PuTTY, echoing the character back to PuTTY, and sending a string (message) to PuTTY. For example, in what order do these tasks (receiving, echoing, sending) execute in your program? Be prepared to talk to your TA about the communication behavior.

PART 4: UART USING INTERRUPTS

If the uart_receive() function is implemented as a blocking receive, that means the function is waiting in a loop until the UART receives a byte. A slow input device (or user typing in PuTTY) could result in a program spending all of its time busy-waiting for data and being unresponsive to other tasks. One way to make programs more efficient is to use interrupts. Think about Part 3 of the lab. Suppose that instead of having your program wait for a byte to be received, the bytes are automatically received and echoed. If needed by your program, the bytes could also be saved in a buffer, and your program would simply read bytes from the buffer when needed while doing other tasks.

This approach is shown below in Figure 12.2 from the VYES book (here "Fifo" means a character buffer).

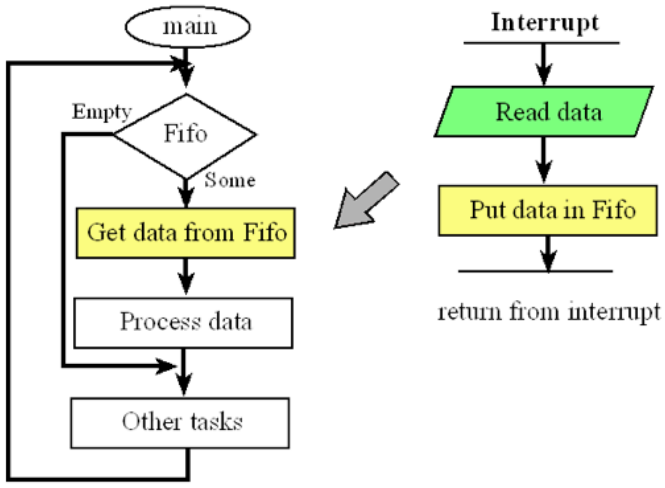


Figure 12.2. For an input device we can use a FIFO to pass data from the ISR to the main program.

In this part of the lab, you will re-write your code from Part 3 to use interrupts. Using interrupts requires the following steps:

1. Enable interrupts from UART1 RX (receiving a byte triggers a UART interrupt)
2. Enable the interrupt controller (NVIC) to process UART1 interrupts and tell the CPU which interrupt handler function (or interrupt service routine, ISR) to execute
3. Enable interrupts in the system (letting the CPU suspend executing your main program, and go execute the ISR)
4. Declare volatile global variables as needed for the ISR to share data with your main program (an ISR does not return a value, and does not have arguments, so global variables are used)
5. Write an interrupt handler function (ISR) to "service the interrupt"
6. Write the main program so that it uses global variables as needed with the ISR

Most of these steps have been implemented for you in this part of the lab. For now, our goal is to introduce you to interrupts so you become aware of what they are, why they are important, and generally how they work. You should gain some awareness of interrupt processing features of the TM4C123 microcontroller, even if you don't understand all of the coding details. Your lab project will involve the use of several interrupts.

****Note**: In order to use interrupts, you must include driverlib/interrupt.h and stdbool.h. You should use the following header files.

```
#include <stdbool.h>
#include "driverlib/interrupt.h"
```

Tip: In order to view functions that may be useful for configuring interrupts, once the interrupt library is included as a header file, you can use Ctrl + Click on the name of the include file to open it. This is also useful for viewing the file containing the register macros.

Refer to the following "interrupt" code files for Lab 5 and browse through all of them:

- lab5-interrupt_template.c
- uart-interrupt.h
- uart-interrupt.c

Your first task is to update the code in uart-interrupt.c.

1. Copy code to this file as appropriate based on the initialization code you wrote in Parts 1-3 (i.e., from uart.c, where you replaced the "???" placeholders). Also copy code to implement the functions (i.e., the "TODO" placeholders). Also, make sure you are using a baud rate of 115,200.
2. Notice the new "?????" placeholders in this file. You need to write code for these. The comments in the code are intended to provide guidance.
3. Start with the code in the uart_interrupt_init() function to initialize UART1 to use receive interrupts. Note that you will only be using receive (RX) interrupts. You will not use interrupts for transmit (TX).
    a. You will need to look up specific registers in the Tiva datasheet to complete the code.
    b. The NVIC enable registers (NVIC_ENx_R) appear complicated in the datasheet. Instead, all you need to know is that there is one enable bit in an NVIC enable register for each device that can interrupt. These devices are numbered starting with 0 (called the Interrupt Number, or IRQ Number, where IRQ stands for Interrupt ReQuest). The Interrupt/IRQ Number for UART1 is 6. This is shown in Table 2-9 in the datasheet. That means that bit 6 in the NVIC enable register is used to enable or disable UART1 interrupts.
4. Next see the UART1_Handler() function, which is the interrupt handler or interrupt service routine (ISR). You, as an embedded programmer, write the code for this function based on how you want to

respond to the UART1 interrupt. However, your program does not call this function. The interrupt system of the microcontroller tells the CPU if and when to execute this function. Some ISR code has been written for you.

a. Start by writing code for the "?????" placeholders in the ISR. Replacing these placeholders should make the ISR fully functional to receive and echo characters from PuTTY.
b. Test that receiving and echoing work using interrupts before adding any new functionality that uses the global shared variables.
c. In other words, at this point, once you have replaced all placeholders (???, ?????, TODO), you can build your program with the main function in lab5-interrupt_template.c, and it should support round-trip communication between PuTTY and the CyBot.

Remember, the files from Parts 1-3 implemented code to receive bytes from PuTTY using a polling method for getting input. In this part of the lab, you are using an interrupt method for getting input.

OPTIONAL: Using the comments in the code for guidance, add new functionality to your main program based on a command key pressed by the user in PuTTY. The response to the command key could be as simple as sending a message back to PuTTY. However, the command key could also affect other tasks in your program, if those tasks are checking the command flag, such as before scanning again or before moving again. There are many ways to program this, and the code and comments suggest one approach.

In your main program, send a message to PuTTY in response to a command key press.

CHECKPOINT:

Verify round-trip communication between the CyBot and PuTTY using interrupts. The bot should be transmitting each character it receives back to PuTTY to be displayed.

DEMONSTRATIONS:

1. **Debug demo using debugging tools to explain something about the internal workings of your system –** The TA will announce any specific debugging requirements at the start of lab; otherwise you will create your own debug demo based on your needs and interests in the lab.
2. **Q&A demo showing the ability to formulate and respond to questions –** This can be done in concert with the other demos.
3. **Functional demo of a lab milestone –** The TA will announce the specific milestone at the start of lab. The milestone is based on the checkpoints.