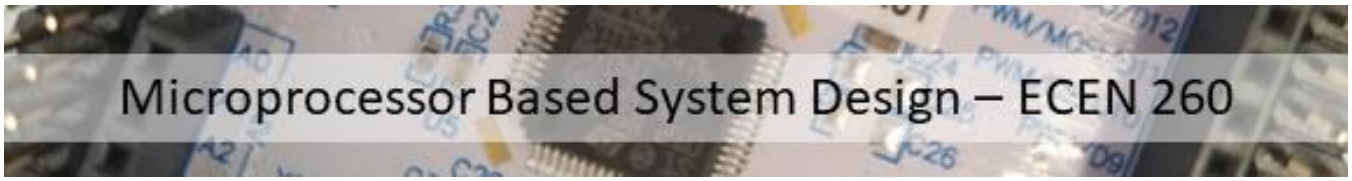


ECEN 260 Lab 10
Asynchronous Serial Communication

Lab Objectives

- Learn how to send a UART message between your laptop and Nucleo board.
- Learn how to send a UART message between two Nucleo boards.
- Begin learning to write a lab report in LaTeX.

Required Parts

- 1 Nucleo-L476RG development board
- 1 USB cable
- 1 breadboard (protoboard)
- 3 single-color LEDs
- 3 resistors
- several jumper wires

Note: This lab is to be completed with a lab partner. You will each need a Nucleo board so your two boards can communicate with each other. If you miss lab, you will need to coordinate with your instructor about borrowing a second Nucleo board to finish the lab on your own.

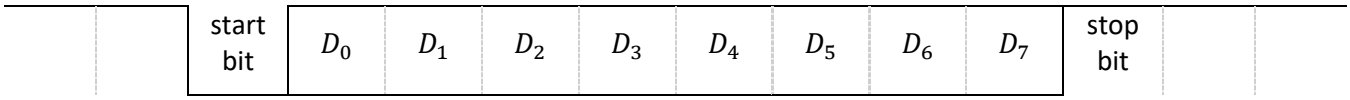
Background: UART Serial Communication

Serial communication sends data over a single wire, one bit at a time. Serial communication can be synchronous (sharing a clock) or asynchronous (no shared clock). For this lab, you will be doing asynchronous serial communication.

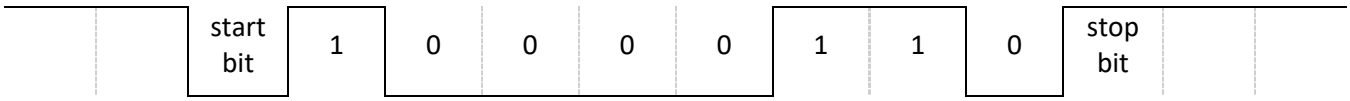
Asynchronous serial communication requires that both the transmitter and the receiver have their own internal communication clocks running at the same frequency. A single “start bit” is used at the beginning of a message to allow the receiver to align its internal communication clock with the data. The clock used for data transmission is usually a few orders of magnitude slower than the CPU clock, so that the CPU can execute several instructions during each cycle of the communication clock.

One of the most common protocols for asynchronous serial communication is the Universal Asynchronous Receiver Transmitter (UART) protocol. UART supports sending 7 or 8 bits at a time (usually for an ASCII character, but the data can be anything – such as measurements from a sensor – not just text). A UART message (frame) may include a parity bit for error detection. UART also includes one or two “stop bits” at the end of the frame to provide a buffer between frames. Between frames, the signal is held at a high voltage. (Keeping the voltage high when no signal is being sent makes it easier to detect if the communication line has been broken.)

The default for UART is 8 bits, no parity bit, 1 stop bit, and LSB first. Here is how such a frame appears:



As an example, if the data being sent were the ASCII code of the character 'a' (0x61 or 0b01100001), the data frame sent on the wire would have the following voltage signal:



UART supports multiple *baud* values as well (for bit transmission, baud = bits per second). The baud determines the data transmission rate. The inverse of the baud determines the duration length of a single bit. A common baud for low-speed data transmission is 9600 bits/second.

Standard Baud Rates	Bit Duration (inverse)
1200 bits/sec	0.833 ms/bit
2400 bits/sec	0.417 ms/bit
4800 bits/sec	0.208 ms/bit
9600 bits/sec	0.104 ms/bit
19200 bits/sec	0.0521 ms/bit
38400 bits/sec	0.0260 ms/bit
57600 bits/sec	0.0174 ms/bit
115200 bits/sec	0.00868 ms/bit

There is also a modified version of UART that *does* support sharing a clock to avoid needing start and stop bits. This version is called U**S**ART (Universal **S**ynchronous/Asynchronous Receiver Transmitter). A USART can operate in either synchronous *or* asynchronous mode. We will use it in asynchronous mode.

UART on the STM32-L476RG

The STM32-L476RG chip has six universal serial modules available for serial communication:

- USART1, USART2, USART3 (which support both synchronous *or* asynchronous communication)
- UART4, UART5 (which only support asynchronous communication)
- LPUART1 (which is a low-power UART that can be put to sleep when not in use)

Module	TxPin/RxPin pairs
USART1	PA9/PA10 or PB6/PB7 or PG9/PG10
USART2	PA2/PA3 or PD5/PD6
USART3	PB10/PB11 or PC4/PC5 or PC10/PC11 or PD8/PD9
UART4	PA0/PA1 or PC10/PC11
UART5	PC12/PD2
LPUART1	PB11/PB10 or PC1/PC0

On the Nucleo board, USART2 is defaulted to PA2/PA3 (labeled with D1/D0 Arduino names) for the serial connection over the USB. Therefore, those pins are reserved for USB traffic, and should not be used for other communication.

In this lab, you will use USART2 (PA2/PA3) to send messages between your laptop and your Nucleo board over the USB connection (Part 1). You will also use USART1 (PA9/PA10) for another UART channel to send messages between your Nucleo board and your lab partner's Nucleo board (Part 2). You will expand on this in Part 3.

Understanding the Code:

Read through these code descriptions to better understand the code you will add:

- `HAL_UART_Receive_IT()` - This function is used to tell the communication module to begin waiting to receive a message from the UART and then trigger an interrupt (IT) when the message is received. The first parameter is the handle (address) to the UART module you are using. The second parameter is the char array/pointer of where to put the message. The third parameter is the length of message to receive. We will receive 1 byte at a time.
- `HAL_UART_Transmit()` - This function is used to send a message with the UART. The first parameter is the handle (address) to the UART module. The second parameter is the char array/pointer that holds the message, the third parameter is the length of the message, and the fourth parameter is the max time it should wait for the transmitter to be ready to send..
- `HAL_UART_RxCpltCallback()` - This is the callback function initiated by the ISR for the UART receive module. This is where we will put the code we want to execute when a message is received from the UART. Its input parameter is the handle (address) of the UART module that received the message. We will check this parameter to see which UART received the message.
- C strings: The message we receive will be stored in the char array `message[]` that you already initialized. A C string is a char array that ends in a null character '\0' – which has the ASCII code of 0x00. So, before you fill the array with the message, you should zero out the array so that it will always have a zero in the spot right after the message.
- `memset()` - Whenever we need to reset the array to all zeros, we can be done with this function. The first parameter is the array, the second parameter is the value you want to set each item in the array to, and the third parameter is the maximum size of the array.
- `strcmp()` - When reading the message, you can compare the string you received with another string using this function. The first parameter is the first string, and the second parameter is the second string. The return value is 0 if the two strings are equal.
- `strncpy()` - When changing a C string to a different message, you can copy a string into it using this function. The first parameter is the destination string, the second parameter is the source string, and the third parameter is the maximum array size.
- *Enter* values: There are two ASCII characters associated with hitting the "Enter" key on your keyboard to start a new line: '\n' is the "new line" or "line feed" (LF) character which has the ASCII code of 0x0A, and '\r' is the "carriage return" (CR) character which has the ASCII code of 0x0D. Some systems use '\n', some use '\r', and some use both. Traditionally, a line feed meant to move the cursor down a row and a carriage return meant to move the cursor back to the start of the line.

Part 0: Setting Up

We need a way for your laptop to transmit and receive over the USB using a serial connection.

1. Windows: If you don't have PuTTY installed on your laptop, do that now. Putty can be used for serial communication. (If you are on a Mac or Linux, you may instead use the "screen" command-line tool – see Appendix A at the end – or you may install the Arduino IDE to use its serial monitor to interact with the serial port from your laptop.

We will use the IOC Device Configuration Tool to configure both USART2 and USART1.

2. Create a new STM32 Project.
    a. Under "Board Selector," choose the Nucleo-L476RG.
    b. Provide a Project Name, such as "Lab10."
    c. Leave the "Targeted Project Type" as "STM32Cube" – do not select "Empty."
    d. On "Initialize all peripherals with their default…" choose Yes.
3. In the "Device Configuration Tool" GUI:
    a. Change PC13 to RESET_STATE (we aren't using the button).
    b. Configure USART2:
        i. Notice that PA2 and PA3 are already configured as the Tx/Rx pins for USART2. These pins are also connected to the USB.
        ii. Under "Categories," expand "Connectivity," and select USART2. This should open the "USART2 Mode and Configuration" pane.
            1. Note that "Mode" should already default as "Asynchronous."
            2. Down in the "Configuration" section, select the "Parameter Settings" tab.
                a. Change "Baud Rate" to 9600.
                b. Note that "Word length" should already default to 8.
                c. Note that "Parity" should already default to "None."
                d. Note that "Stop Bits" should already default to 1.
            3. Switch to the "NVIC Settings" tab.
                a. Tick the "Enabled" box next to "USART2 global interrupt."
    c. Configure USART1:
        i. Click on pin PA9 and select USART1_TX.
        ii. Click on pin PA10 and select USART1_RX.
        iii. Under "Categories," expand "Connectivity," and select USART1. This should open the "USART1 Mode and Configuration" page.
            1. Change "Mode" to "Asynchronous."
            2. Down in the "Configuration" section, select the "Parameter Settings" tab.
                a. Change "Baud Rate" to 9600.
                b. Note that "Word length" should already default to 8.
                c. Note that "Parity" should already default to "None."
                d. Note that "Stop Bits" should already default to 1.
            3. Switch to the "NVIC Settings" tab.
                a. Tick the "Enabled" box next to "USART1 global interrupt."
    d. Save to generate the code. You can also do this manually by selecting Project -> Generate Code.

4. In the “USER CODE BEGIN Includes” section, include the following libraries:

```
#include "string.h"  // library for C strings
#include <stdbool.h> // library for bool data type
```

5. In the “USER CODE BEGIN PM” section, add the following private macros (PM):

```
#define UART_DELAY 100       // wait max of 100 ms between frames in message
#define MAX_MESSAGE_SIZE 100 // 100 characters maximum message size
```

6. In the “USER CODE BEGIN PV” section, add the following private variables (PV):

```
uint8_t message[MAX_MESSAGE_SIZE]  = {0}; // char array to store message received
uint8_t response[MAX_MESSAGE_SIZE] = {0}; // char array to store response message
uint8_t uart1_byte;                       // byte received from UART1
uint8_t uart2_byte;                       // byte received from UART2
uint8_t buffer_position = 0;              // how many bytes received so far
bool blink = false;                       // LED blink status
```

7. Add the following code in the “USER CODE BEGIN 2” section in main, before the while loop to start the UART modules:

```
// Start UART1 and UART2 to receive interrupts
HAL_UART_Receive_IT(&huart1, &uart1_byte, 1); // put byte from UART1 in "uart1_byte"
HAL_UART_Receive_IT(&huart2, &uart2_byte, 1); // put byte from UART2 in "uart2_byte"
```

8. Add the following code in the “USER CODE BEGIN 3” section. The code should still be inside the while loop, so be careful it goes before the closing curly bracket that ends the while loop. This code will check the global variable “blink” to see if it should toggle LED2.

```
if (blink){
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin); // Toggle LED2
    HAL_Delay(200); // wait 200 ms
}
```

You are now ready for Part 1 in which you will add the code that will transmit (Tx) and receive (Rx) between your laptop and microcontroller using the USART2 module over the USB connection.

Part 1: Laptop-Microcontroller Communication over USART2

You will now add the callback function (`HAL_UART_RxCpltCallback`) which the ISR calls whenever an interrupt is triggered indicating that a UART module has received new data.

1. Download the code from here.
2. Add the code after main in the "USER CODE BEGIN 4" section.
3. Read through the code once to get a feel for what it does.
4. Plug in your microcontroller.
5. Save and load the code onto the board.

Now you will use PuTTY on your laptop to transmit and receive messages over the USB connection with your microcontroller. (If you have a Mac or Linux, see Appendix A and/or ask your instructor for help.)

6. Find out with COM port you are connected to.
    a. Open the Windows Device Manager. (Search for it in the Start menu.)
    b. Expand the "Ports (COM & LPT)" section.
    c. Find the COM# your STM board's STLink connection is on.
7. Open PuTTY.
    a. Change the "Connection type" from SSH to Serial.
    b. Verify that the "Speed" is set to 9600.
    c. Change the "Serial Line" from COM1 to the COM# your board is connected to.
    d. In the Category settings to the left, select "Terminal."
        i. Change "Local Echo" to "Force on." This will allow you to see what type.
        ii. Change "Local Line Editing" to "Force on." This will allow you to edit what you type and only send it once you hit Enter.
        iii. **Note:** If you find later that you have issues with either Line Feeds or Carriage Returns not showing, you can also check the boxes "Implicit CR in every LF" and "Implicit LR in every CR."
    e. Click "Open" to start the serial connection.
8. Experiment with typing messages:
    a. "LED2_ON" should turn on your LED, with a message received back saying "Task complete - LED2 turned on."
    b. "LED2_OFF" should turn off your LED, with a message received back saying "Task complete - LED2 turned off."
    c. "LED2_BLINK" should set the blink variable so the main while loop starts blinking your LED, which a message received back saying "Task complete - LED2 blinking."
    d. Any other message should cause a message to be received back saying "Task not recognized."

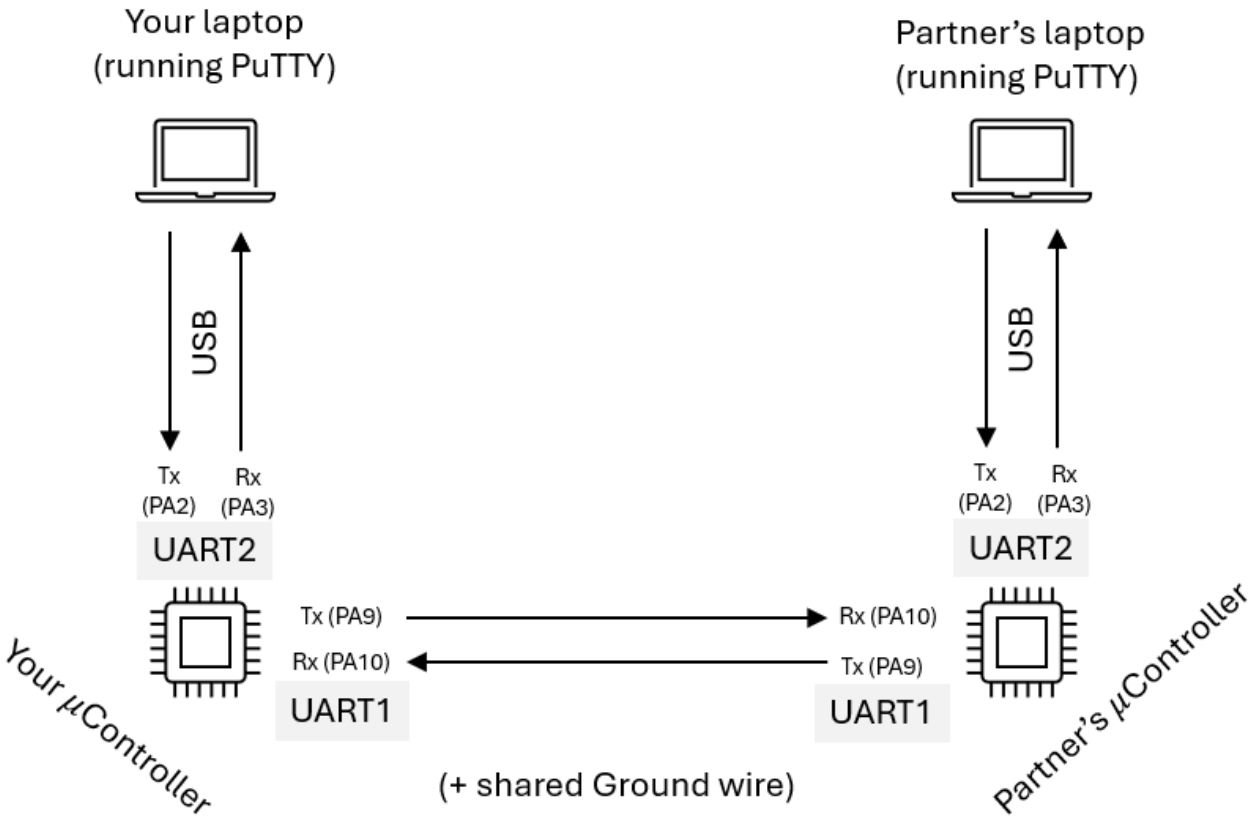
Verify that you understand the code:

9. Now that you have seen the code in action, read through the Callback code a second time to get a better understanding of what each line of code does.

You are now ready to move on to Part 2. If your lab partner isn't finished with Part 1, help him/her.

Part 2: Microcontroller-to-Microcontroller Communication over USART1

For this part of the lab, you will take any message received from your laptop via USART**2** and immediately send it from your microcontroller to your lab partner's microcontroller over USART**1**, and you will take any message you receive from your lab partner's microcontroller over USART**1** and immediately send it from your microcontroller to your laptop over USART**2**. Essentially, you will have a chat session between your laptops *through* your microcontrollers.



1. Connect the USART1 channel between your microcontroller and your lab partner's microcontroller:
    a. Connect GND to GND so that both microcontrollers share the same voltage level as zero volts.
    b. Connect the PA9 TX pin (labeled D8) on your board to the PA10 Rx pin (labeled D2) on your lab partner's board.
    c. Connect the PA9 TX pin (labeled D8) on your lab partner's board to the PA10 Rx pin (labeled D2) on your board.

What to do if you are in a 3-person group:

If you are in a 3-person group, you will create a "ring" network. Person A's board will transmit to Person B's board. Person B's board will transmit to Person C's board, and Person C's board will transmit to Person A's board. Your instructor can help.

2. Change the HAL_UART_RxCpltCallback function to the following:

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){

    // Check if byte received was on UART2 (from laptop)
    if (huart == &huart2){

        // If we get here, we received a byte from UART2 and it was placed in "uart2_byte"

        // Take byte received from my laptop over UART2 and send to lab partner over UART1
        HAL_UART_Transmit(&huart1, &uart2_byte, 1, UART_DELAY);

        // Restart UART2's receive interrupt to wait for next byte from laptop
        HAL_UART_Receive_IT(&huart2, &uart2_byte, 1); //start next byte receive interrupt
    }

    // Check if byte received was on UART1 (from lab partner)
    if (huart == &huart1){

        // If we get here, we received a byte from UART1 and it was placed in "uart1_byte"

        // Take byte received from lab partner over UART1 and send to my laptop over UART2
        HAL_UART_Transmit(&huart2, &uart1_byte, 1, UART_DELAY);

        // Restart UART1's receive interrupt to wait for next byte
        HAL_UART_Receive_IT(&huart1, &uart1_byte, 1); //start next byte receive interrupt
    }
}
```

3. Save and load the code onto your board.
4. Open PuTTY with the same settings as before.
5. Type messages and send them back and forth to each other.

Part 3: Designing your own Communication System

With your lab partner, change the code in the Callback function to check the messages that are being received from the other microcontroller (the messages from UART1) and compare them with certain "tasks" to be performed, such as turning on or off different LEDs that you add to your boards. Use the code from Part 1 as a guide. The goal is for each of you to be able to control several LEDs connected to the other person's board with commands. Do at least three LEDs. Remember that you can configure a pin as an output pin using the .ioc file's "Device Configuration Tool."

'A'-level versus 'B'-level project: A 'B'-level project would have the ability for you each to turn on or off multiple LEDs individually on your lab partner's board. An 'A'-level project would have the ability for you to also send a "blink" command to each of the LEDs individually or some other advanced command like changing the PWM intensity of each color of a tri-color RGB LED.

Test Plan

With your lab partner, come up with at least 5 scenarios that you can test. Each scenario should explicitly list the precise steps to perform that test. Consider including edge cases and error scenarios.

Lab Report

You will each write your own lab report. Don't forget that this week you will being using LaTeX to create your lab reports (see instructions on the next page).

Schematic for your report

Your schematic should show the connection between your laptop and your microcontroller, the connection between your microcontroller and your partner's microcontroller, and the connection between your partner's microcontroller and your partner's laptop. You should also include any external LEDs. Make sure to label specific pins and to include the shared ground connection. For each laptop, just make a box called "laptop" that has the wires from PA2 and PA3 connected to it, representing the USB connection.

Demonstration of Correct Operation

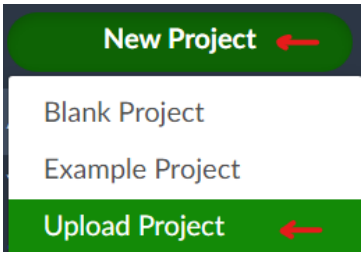
(While the lab has three parts, you will only submit your code for Part 3. Similarly, you will only create a test plan and record results and a video for Part 3.)

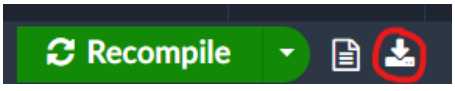
As soon as you complete the lab, record a video of your Part 3 system working correctly. Then, upload (1) the video and (2) a copy of your main.c file to the "Lab 10 Demo & Code" assignment in I-Learn. Please also add a comment indicating if your project is an 'A'-level or 'B'-level project.

Lab Report Instructions

From now on, you will use LaTeX to generate your lab reports. LaTeX is a typesetting language that can generate technical documents using code.

1. Create a free account in Overleaf (see the reading).
2. Download the zip file (ECEN_260-Lab_Report_Template_LaTeX.zip) from the Lab 10 Report assignment in I-Learn. It contains the LaTeX code for your lab report template.
3. Create a new LaTeX project in Overleaf using the template zip file by uploading the zip file (see image to the right).
4. Start with the `main.tex` file and edit the template according to the instructions in the code as you insert the material for your lab report (see descriptions of what should go in each section below). You may ask your instructor if you get stuck.
5. Note that underscores '_' are special characters in LaTeX. If you need an underscore, you must escape it using a backslash '_'
6. After you are finished with your report, download the generated PDF (see image to the right).
7. Submit the generated PDF to the "Lab 10 Report" assignment in I-Learn.





Don't forget that I love you! :)

Appendix A: Using `screen` to interface with the serial port

For Mac or Linux, the simplest approach may be to use the Arduino IDE's Serial Monitor to send/receive messages. You may also use the screen tool on a command line to interface with the serial port. The messages you type with the screen tool will be send over the USB serial connection, and messages received over the serial connection will display in the terminal.

The 'screen' program is based on the VT-100 emulator and comes default with local-echo on (meaning that you can see what you type).

You run the screen command by typing `screen`, followed by the "file" name of the I/O device you are using to transmit/receive messages, followed by the baud rate.

So, first find the "file" name for your USB connection my doing:

```
ls /dev/tty*usb*
```

That should show the file name. Then type:

```
screen filename baudrate
```

where you replace filename with the file name you found and replace baudrate with 9600.

Here's an example:

```
rex-imac:~ lynn$ ls /dev/tty*usb*
/dev/tty.usbmodem22102
rex-imac:~ lynn$ screen /dev/tty.usbmodem22102 9600
```

Note: Screen doesn't accept CTRL-C as break, and disconnecting will usually leave the terminal in a weird state. However, unlike the PuTTY, restarting or resetting the STM32's USB link will NOT disconnect screen. Generally, you just kill the terminal and open a new one and restart screen, if necessary.