ECEN 260 - Laboratory #10

Asynchronous Serial Communication

Brodric Young

Instructor: Brother Allred
March 12, 2025

Contents

List of Tables

List of Figures

1 Lab Overview

In this lab, we explored asynchronous serial communication using the Universal Asynchronous Receiver Transmitter (UART) protocol. This lab provided experience with UART communication, which is a fundamental aspect of embedded systems and microcontroller interfacing.

1.1 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.

2 Specifications

This project uses the UART communication mode of an STM32 microcontroller on a Nucleo development board, specifically the Nucleo32-L476RG [1]. An overview of the UART protocol may be found in Chapter 9 of [2].

The system we developed in this lab facilitated asynchronous serial communication between a laptop and the Nucleo board, and between two Nucleo boards. The system used USART2 for communication between the laptop and the Nucleo board via USB, and USART1 for communication between two Nucleo boards. The system could transmit and receive messages, control LEDs based on received commands, and handle regular messages.

For specifics on the wiring setup, see the schematics in Section 3.

2.1 Parts List

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

3 Schematic

This section is for the schematic of the complete wiring for this lab, seen in Figure 1. It shows the connection between my laptop and my microcontroller, the connection between my microcontroller and my lab partner's microcontroller, and the connection between my lab partner's microcontroller and my lab partner's laptop. It also includes the external LEDs. Every connection is labeled with the proper pins on the microcontrollers. Each laptop has wires from PA2 and PA3 of the microcontroller connected to it, representing the USB connection.

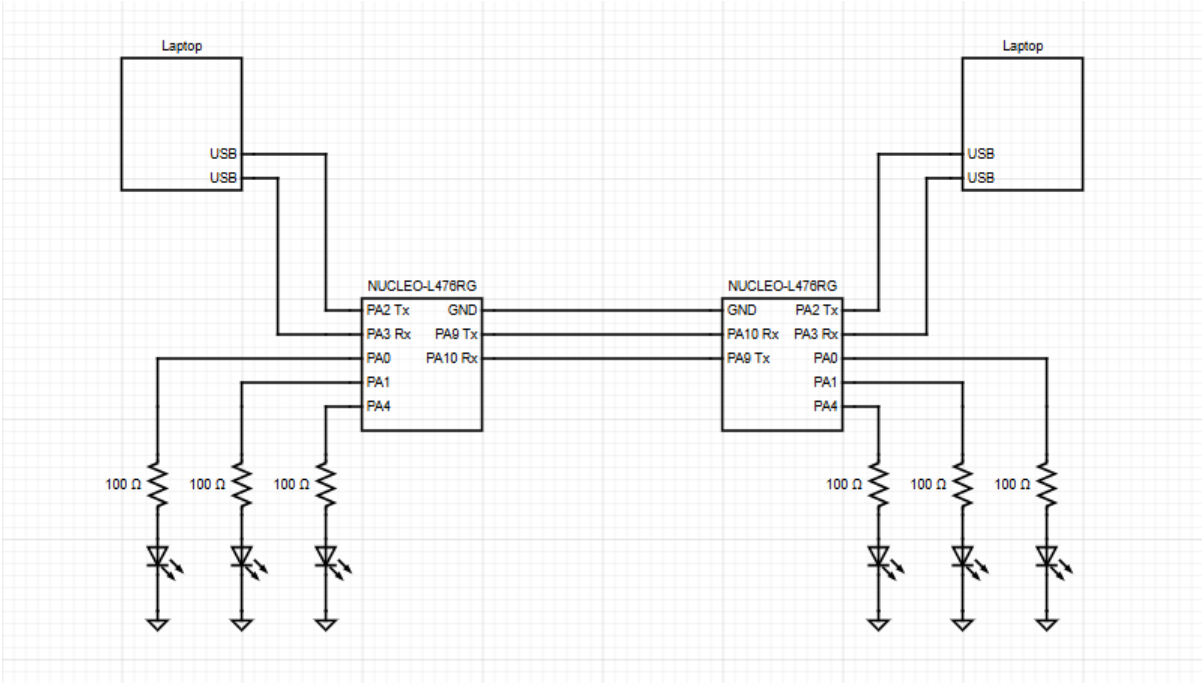

Figure 1: Schematic diagram for the lab.

4 Test Plan and Test Results

This test plan was designed to cover a range of scenarios, including common cases, edge cases, and error cases. The goal was to ensure that the system could handle typical usage as well as potential issues that might arise.

We tested valid commands like "GO," "GO_FASTER," "STOP," and "PARTY_TIME" to ensure that the system responded correctly by controlling the LEDs as specified. These tests confirmed that the system could handle standard inputs and perform the expected actions.

We considered the limits of supported inputs, such as a very long message. We also tested how the system responded to invalid commands, which were just messages sent to each others laptops, to ensure it didn't crash or behave strange.

We tried scenarios where messages were cut off or incomplete to see if the system could handle them gracefully without causing errors or misinterpretations.

4.1 Test Plan Procedure

Table 1: Test plan with expected and observed results

Case	Test Scenario	Steps	Expected Results	Actual Results
1	Test "GO" command, me to partner	1. Send "GO" from my laptop	Lab partner's green LED should turn on	Lab partner's green LED turned on
2	Test "GO" command, partner to me	1. Send "GO" from partner's laptop	My green LED should turn on	My green LED turned on
3	Test "GO_FASTER" command, me to partner	1. Send "GO_FASTER" from my laptop	Lab partner's yellow LED should turn on	Lab partner's yellow LED turned on
4	Test "GO_FASTER" command, partner to me	1. Send "GO_FASTER" from partner's laptop	My yellow LED should turn on	My yellow LED turned on
5	Test "STOP" command, me to partner	1. Send "STOP" from my laptop	Lab partner's red LED should turn on	Lab partner's red LED turned on

Case	Test Scenario	Steps	Expected Results	Actual Results
6	Test "STOP" command, partner to me	1. Send "STOP" from partner's laptop	My red LED should turn on	My red LED turned on
7	Test "PARTY_TIME" command, me to partner	1. Send "PARTY_TIME" from my laptop	Lab partner's LEDs should each blink in order	Lab partner's LEDs each blinked in order
8	Test "PARTY_TIME" command, partner to me	1. Send "PARTY_TIME" from partner's laptop	My LEDs should each blink in order	My LEDs each blinked in order
9	Test long message, me to partner	1. Send a long message from my laptop	Message should arrive on lab partner's laptop; no LED change	Message arrived on lab partner's laptop; no LED change
10	Test long message, partner to me	1. Send a long message from partner's laptop	Message should arrive on my laptop; no LED change	Message arrived on my laptop; no LED change
11	Test incomplete command "STO", me to partner	1. Send "STO" from my laptop	"STO" should arrive on partner's laptop; no LED change	"STO" arrived on partner's laptop; no LED change
12	Test incomplete command "STO", partner to me	1. Send "STO" from partner's laptop	"STO" should arrive on my laptop; no LED change	"STO" arrived on my laptop; no LED change

5 Code

The following code is the code executed in the infinite while loop from main.c, found in Section 5.1, and the callback function found in Section 5.2, also in main.c which is called when information is received over the UART connection.

5.1 Code for infinite while loop

```
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
      if (party_time) {
          HAL_GPIO_TogglePin(GREEN_LED_GPIO_Port, GREEN_LED_Pin); // Toggle
   green led
          HAL_Delay(100); // wait 100 ms
          HAL_GPIO_TogglePin(YELLOW_LED_GPIO_Port, YELLOW_LED_Pin); // Toggle
   yellow led
          HAL_Delay(100); // wait 100 ms
          HAL_GPIO_TogglePin(RED_LED_GPIO_Port, RED_LED_Pin); // Toggle red
   led
          HAL_Delay(100); // wait 100 ms
      }
    }
/* USER CODE END WHILE */
```

5.2 Code for UART info received callback

```
/* USER CODE BEGIN 4 */
//
   ////////////////////////////////////////////////////////////////////////////

// Function called when I receive a new message
// UART2 is between my latop and my board
// UART1 is between my lab partners board and my board
//
   ////////////////////////////////////////////////////////////////////////////

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
 }
```

```
17
18
19  // Check if byte received was on UART1 (from lab partner)
20  if (huart == &huart1){
21
22      // Take byte received from my laptop over UART2 and send to lab partner
    over UART1
23      HAL_UART_Transmit(&huart2, &uart1_byte, 1, UART_DELAY);
24      // Restart UART2's receive interrupt to wait for next byte from laptop
25      HAL_UART_Receive_IT(&huart1, &uart1_byte, 1); //start next byte receive
    interrupt
26
27      // If we get here, we received a byte from UART1 and it was placed in "
    uart1_byte"
28      party_time = false;
29
30
31      // If uart2_byte isn't \r, \n, or \0, it means the message isn't over yet
32      if ((uart1_byte != '\r') && (uart1_byte != '\n') && (uart1_byte != '\0'))
    {
33          // Add uart2_byte to the message
34          message[buffer_position] = uart1_byte;
35          buffer_position++;
36
37      } else {
38          // If we get here, it means we received \r, \n, or \0 and the message
     is over
39
40          // Check message
41          if (strcmp((char*)message, "PARTY_TIME") == 0) {
42              party_time = true; // party time turns on
43
44              // Prepare response message
45              strncpy((char*)response, "Task complete - PARTY_TIME turned on.\n
    \r", MAX_MESSAGE_SIZE);
46
47
48          } else if (strcmp((char*)message, "GO") == 0) {
49
50              // If "GO\n" was the message received -> Turn off yellow and red
    leds, turn on green
51              HAL_GPIO_WritePin(GREEN_LED_GPIO_Port, GREEN_LED_Pin,
    GPIO_PIN_SET);
52              HAL_GPIO_WritePin(YELLOW_LED_GPIO_Port, YELLOW_LED_Pin,
    GPIO_PIN_RESET);
53              HAL_GPIO_WritePin(RED_LED_GPIO_Port, RED_LED_Pin, GPIO_PIN_RESET)
    ;
54
55              // Prepare response message
56              strncpy((char*)response, "Task complete - 'GO'.\n\r",
    MAX_MESSAGE_SIZE);
57
58
59          } else if (strcmp((char*)message, "GO_FASTER") == 0) {
```

```
60
61              // If "GO_FASTER\n" was the message received -> Turn off green
    and red leds, turn on yellow
62              HAL_GPIO_WritePin(GREEN_LED_GPIO_Port, GREEN_LED_Pin,
    GPIO_PIN_RESET);
63              HAL_GPIO_WritePin(YELLOW_LED_GPIO_Port, YELLOW_LED_Pin,
    GPIO_PIN_SET);
64              HAL_GPIO_WritePin(RED_LED_GPIO_Port, RED_LED_Pin, GPIO_PIN_RESET)
    ;
65
66              // Prepare response message
67              strncpy((char*)response, "Task complete - 'GO_FASTER'\n\r",
    MAX_MESSAGE_SIZE);
68
69
70          } else if (strcmp((char*)message, "STOP") == 0) {
71
72              // If "STOP\n" was the message received -> Turn off green and
    yellow leds, turn on red
73              HAL_GPIO_WritePin(GREEN_LED_GPIO_Port, GREEN_LED_Pin,
    GPIO_PIN_RESET);
74              HAL_GPIO_WritePin(YELLOW_LED_GPIO_Port, YELLOW_LED_Pin,
    GPIO_PIN_RESET);
75              HAL_GPIO_WritePin(RED_LED_GPIO_Port, RED_LED_Pin, GPIO_PIN_SET);
76
77              // Prepare response message
78              strncpy((char*)response, "Task complete - 'STOP'\n\r",
    MAX_MESSAGE_SIZE);
79          }
80
81
82          // Send the response message to laptop
83          HAL_UART_Transmit(&huart2, response, strlen(response), UART_DELAY);
84
85          // Zero out message array and response array to get ready for the
    next message
86          memset(message,  0, sizeof(message));
87          memset(response, 0, sizeof(response));
88          buffer_position = 0;
89
90      } // end of full message received else
91
92      // Restart UART2's receive interrupt to wait for next byte
93      HAL_UART_Receive_IT(&huart1, &uart1_byte, 1); //start next byte receive
    interrupt
94
95  } // end of if byte received was on UART1 (from lab partner)
96 }
97 /* USER CODE END 4 */
```

6 Conclusion

In this lab, we explored asynchronous serial communication using the UART protocol, which is crucial for microcontroller interfacing. By writing and testing code to handle UART communication, we learned to transmit and receive messages, control LEDs based on commands, and manage data using interrupts. This experience reinforced our understanding of digital I/O and interrupts and highlighted how to use UART in developing complex embedded systems. This knowledge is essential for future projects.

The most useful part of the lab was implementing and testing the various commands to control the LEDs. This experience helped solidify my understanding of how to use UART for communication. It was particularly rewarding to see the LEDs respond to the commands as expected, which demonstrated the practical application of the concepts we learned in class. The lab built on my existing knowledge learned so far and provided a deeper understanding of asynchronous serial communication. Moving forward, I hope to apply these skills to more complex projects. This lab has given me a strong foundation in UART communication, which will be essential for future coursework and professional work in embedded systems design.

References

[1] "Stm32 nucleo-64 development board with stm32l476rg mcu." https://www.st.com/en/evaluation-tools/nucleo-l476rg.html. Accessed: 2023-06-21.

[2] J. Catsoulis, *Designing embedded hardware*. O'Reilly, second ed., 2005.