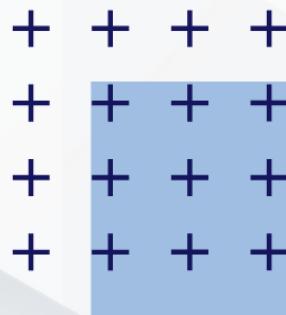
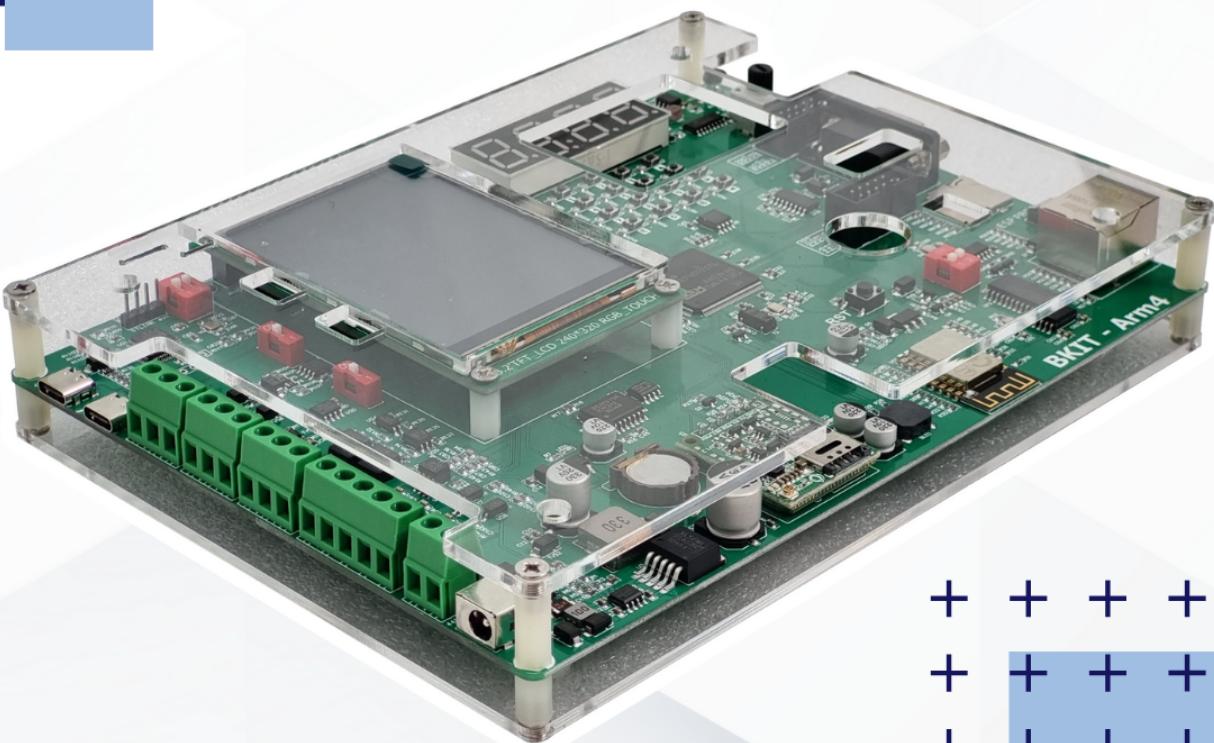


KIT THÍ NGHIỆM BKIT

ARM4

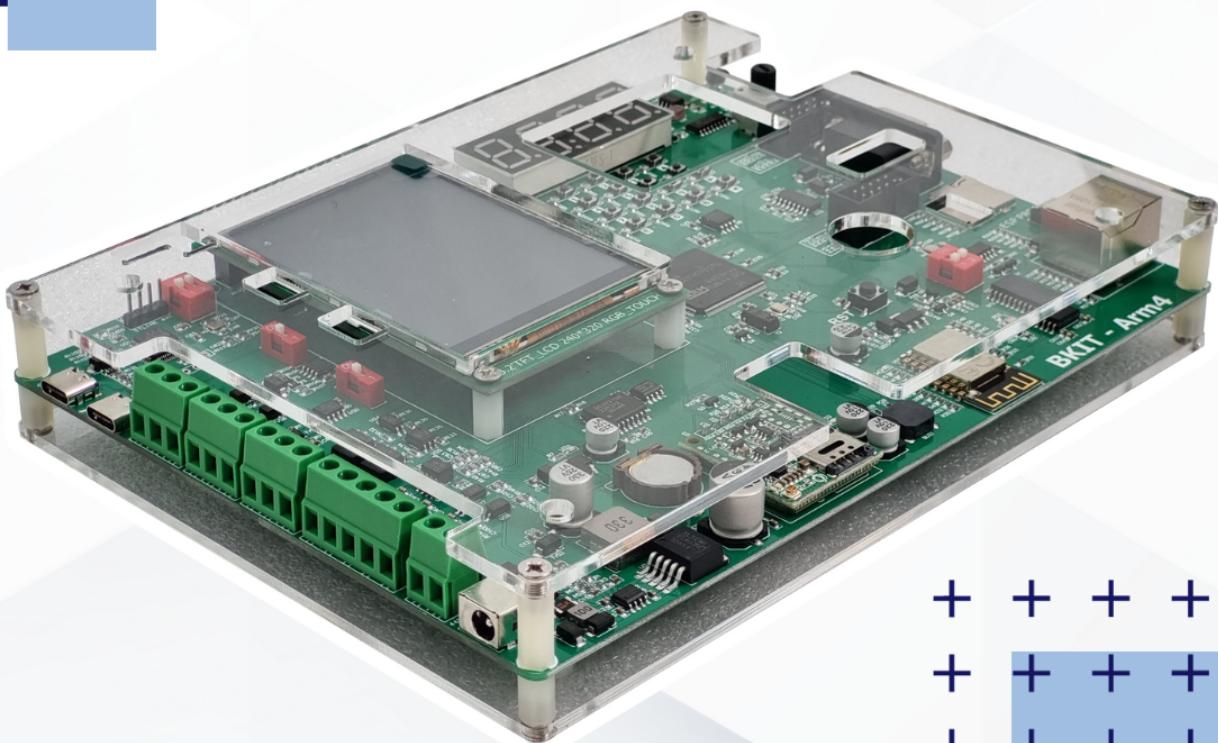


Mục lục

Chapter 5. Universal Asynchronous Receiver-Transmitter	3
1 Mục tiêu	4
2 Giới thiệu	4
3 Cơ sở lý thuyết	5
3.1 UART	5
3.2 RS232	6
4 Hướng dẫn cấu hình	7
4.1 Cấu hình UART-RS232	7
5 Hướng dẫn lập trình	8
5.1 Thư viện uart.h	8
6 Bài tập và báo cáo	15
6.1 Bài tập 1	15
6.2 Bài tập 2	15
6.3 Bài tập 3	15

CHƯƠNG 5

Universal Asynchronous Receiver-Transmitter



1 Mục tiêu

- Tìm hiểu giao tiếp UART.
- Biết cách sử dụng module RS232 trên Kit thí nghiệm.
- Biết cách sử dụng phần mềm để giao tiếp giữa Kit thí nghiệm và máy tính thông qua RS232.

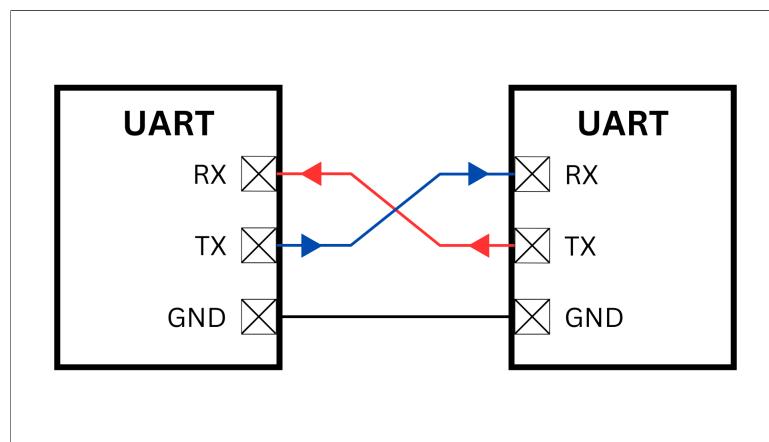
2 Giới thiệu

UART (Universal Asynchronous Receiver-Transmitter) là một giao thức truyền thông không đồng bộ đáng tin cậy và được sử dụng rộng rãi, thường được áp dụng cho việc liên lạc giữa các thiết bị và module như Wifi, Bluetooth, Xbee, module đọc thẻ RFID trong Raspberry Pi, Arduino và các vi điều khiển khác.

3 Cơ sở lý thuyết

3.1 UART

UART là giao thức truyền thông được sử dụng trong giao tiếp nối tiếp. Giao thức này thường được sử dụng trong truyền thông khoảng cách ngắn (trong cùng một thiết bị hoặc các thiết bị trên cùng bo mạch). Trong giao tiếp UART, hai thiết bị UART giao tiếp trực tiếp với nhau. UART chuyển đổi dữ liệu song song từ một thiết bị điều khiển thành dạng nối tiếp, truyền dữ liệu nối tiếp tới thiết bị nhận UART, sau đó chuyển đổi dữ liệu nối tiếp thành dữ liệu song song cho thiết bị nhận. Chỉ cần hai dây để truyền dữ liệu giữa hai thiết bị UART. Luồng dữ liệu từ chân Tx của truyền UART đến chân Rx của UART nhận:



Hình 5.1: Sơ đồ giao tiếp giữa 2 thiết bị qua UART

Khi thiết bị UART nhận diện được bit khởi đầu, nó bắt đầu đọc bit theo một tần số cụ thể, được gọi là baudrate. Baudrate là chỉ số đo tốc độ truyền dữ liệu, thể hiện bằng bit/giây (bps). Cả hai UART đều phải hoạt động ở cùng một tốc độ truyền. Sự khác biệt về tốc độ truyền giữa thiết bị truyền và thiết bị nhận chỉ có thể chênh lệch khoảng 10%. Cả hai thiết bị UART cũng cần được cấu hình để truyền và nhận cùng một cấu trúc gói dữ liệu.

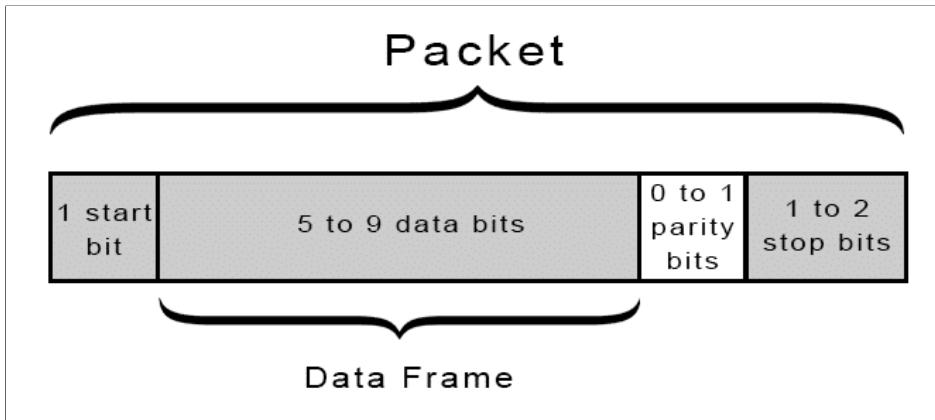
UART có thể được hiện thực theo một trong ba chế độ:

- Full duplex: Giao tiếp đồng thời đến và đi từ mỗi master và slave.
- Half duplex: Dữ liệu đi theo một hướng tại một thời điểm.
- Simplex: Chỉ giao tiếp một chiều.

Dữ liệu truyền UART được tổ chức thành các gói. Mỗi gói chứa 1 bit bắt đầu, 5 đến 9 bit dữ liệu, bit chẵn lẻ tùy chọn và 1 hoặc 2 bit dừng:

Wires Used	2
Maximum Speed	Any speed up to 115200 baud, usually 9600 baud
Synchronous or Asynchronous?	Asynchronous
Serial or Parallel?	Serial
Max # of Masters	1
Max # of Slaves	1

Hình 5.2: Các điểm cần lưu ý về UART



Hình 5.3: Dataframe của UART

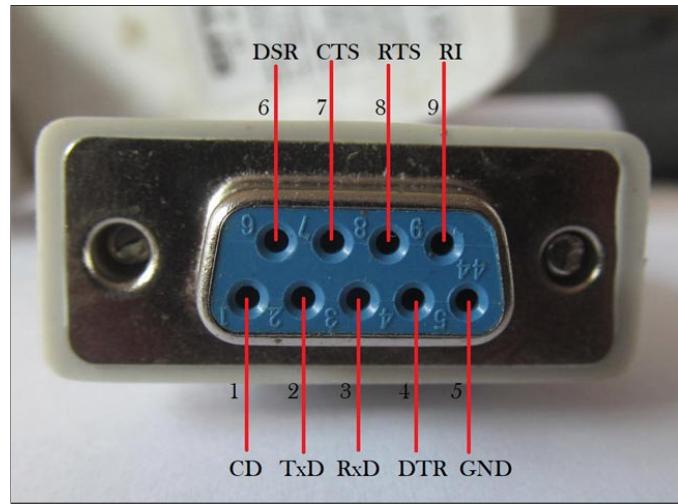
3.2 RS232

RS232 là một hình thức truyền dữ liệu nối tiếp. Nó được thiết kế để cho phép truyền và nhận dữ liệu theo cách nối tiếp, tức là dữ liệu được truyền từng bit một và theo thứ tự liên tiếp theo chuỗi thời gian.

RS232 cho phép giao tiếp full-duplex, tức là việc đường truyền và nhận dữ liệu có thể được sử dụng đồng thời. RS232 được dùng phổ biến để giao tiếp trong khoảng cách tương đối ngắn (tối đa khoảng 15m).

Cấu tạo của RS232:

- Thực tế là cổng RS232 có hai loại đầu nối: DB-25 và DB-9. DB-25 có 25 chân, phục vụ nhiều ứng dụng, nhưng không phải ứng dụng nào cũng cần dùng hết 25 chân. Do đó, đầu nối 9 chân được thiết kế để thuận lợi hơn cho việc kết nối các thiết bị.

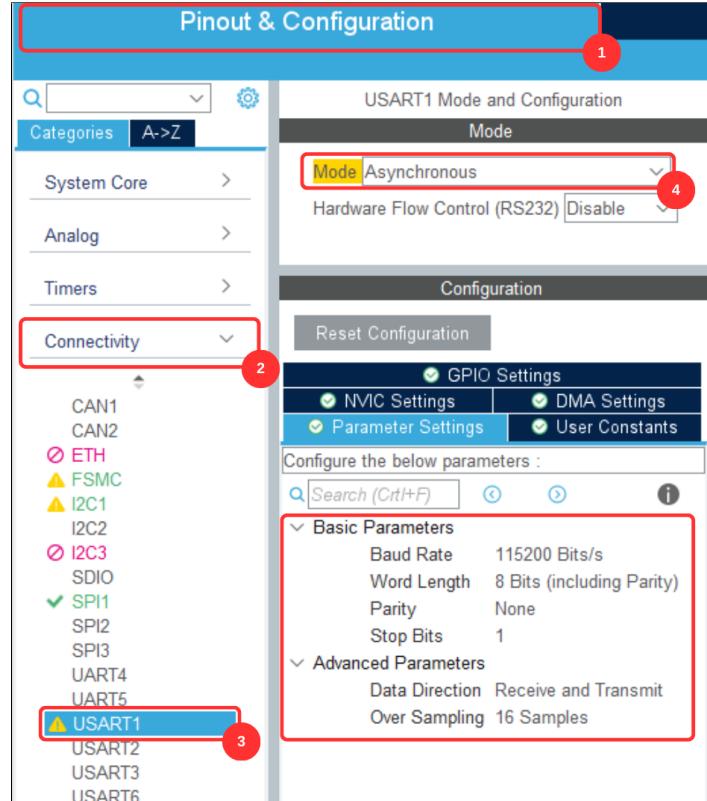


Hình 5.4: Cổng RS232 DB9

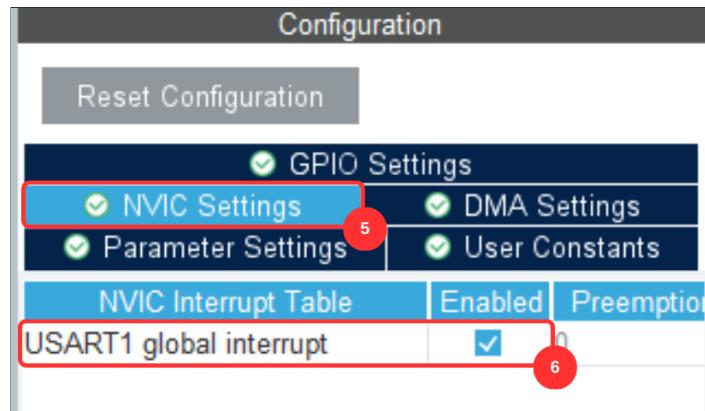
4 Hướng dẫn cấu hình

4.1 Cấu hình UART-RS232

Trên Kit thí nghiệm, module RS232 được điều khiển thông qua module UART1 trên MCU. Trong file .ioc chúng ta sẽ cấu hình USART1 như sau:

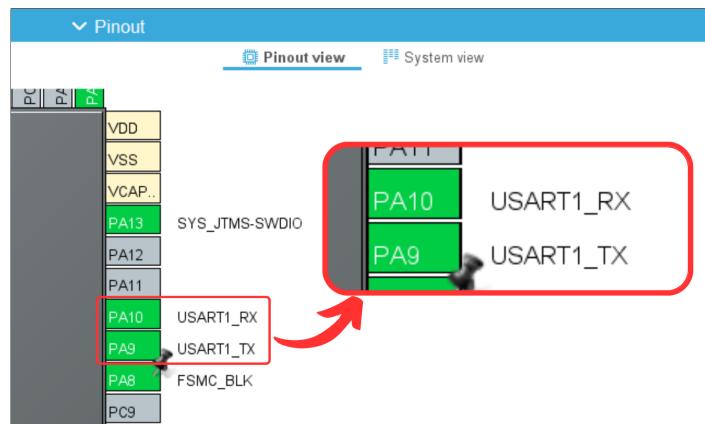


Hình 5.5: Config USART1



Hình 5.6: Config UART1 (tiếp theo)

Sau khi cấu hình, ta có Pinout view như sau:



Hình 5.7: Pinview out sau khi cấu hình UART1

5 Hướng dẫn lập trình

5.1 Thư viện uart.h

Các file **uart.h** và **uart.c** có thể được sao chép từ project mẫu **Bai5_UART**.

void uart_init_rs2320

- **Mô tả:** Khởi tạo RS232.
- **Tham số:** Không có
- **Giá trị trả về:** Không có.

void uart_Rs232SendString(uint8_t* str)

- **Mô tả:** Gửi chuỗi ký tự qua RS232.

- **Tham số:**

- **str:** Chuỗi ký tự cần gửi.

- **Giá trị trả về:** Không có.

void uart_Rs232SendBytes(uint8_t* bytes, uint16_t size)

- **Mô tả:** Gửi chuỗi byte qua RS232.

- **Tham số:**

- **bytes:** Mảng chứa dữ liệu.

- **size:** Độ dài dữ liệu.

- **Giá trị trả về:** Không có.

void uart_Rs232SendNum(uint32_t num)

- **Mô tả:** Gửi số dưới dạng kí tự qua RS232.

- **Tham số:**

- **num:** Số cần gửi.

- **Giá trị trả về:** Không có.

Ngoài ra ta có hàm **HAL_UART_RxCpltCallback** là hàm ngắn (interrupt service routine) sẽ được gọi mỗi khi nhận được dữ liệu. Sau này chúng ta có thể sửa đổi hàm này tùy theo mục đích xử lý dữ liệu nhận vào. Trong ví dụ hiện tại, hàm này sẽ gửi lại dữ liệu được nhận.

```

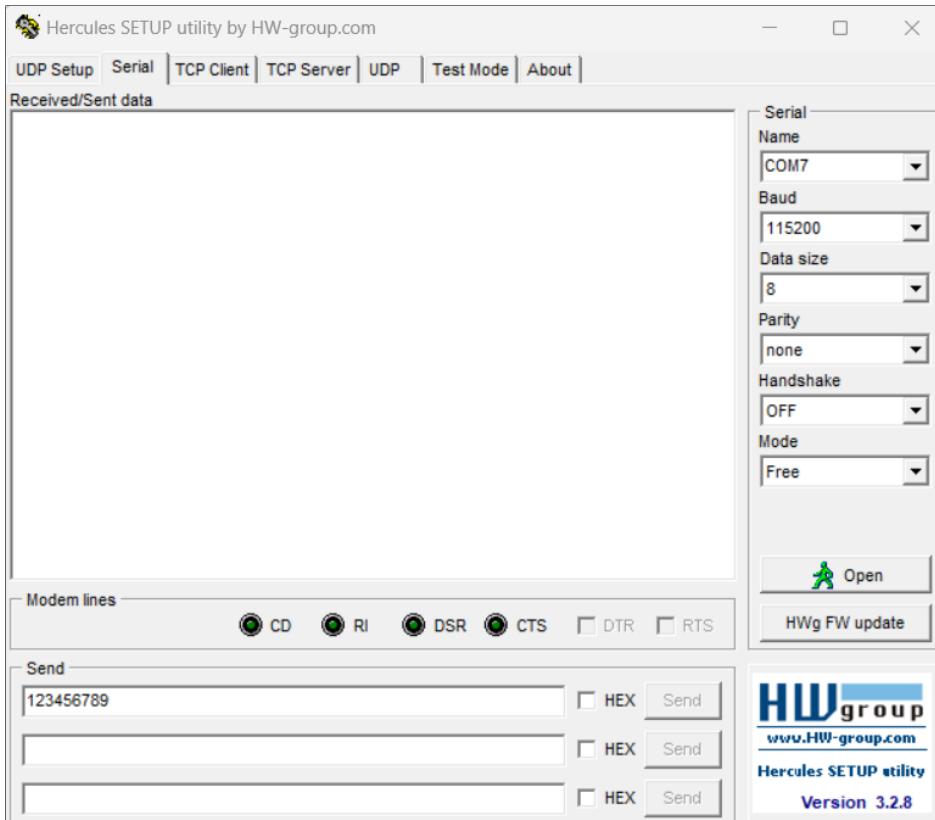
1 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
2     if(huart->Instance == USART1){
3         // rs232 isr
4         // can be modified
5         HAL_UART_Transmit(&huart1, &receive_buffer1, 1, 10);
6
7         // turn on the receive interrupt
8         HAL_UART_Receive_IT(&huart1, &receive_buffer1, 1);
9     }
10 }
```

Program 5.1: Hàm interrupt

Trong bài thí nghiệm này, cổng RS232 trên Kit thí nghiệm sẽ được kết nối với máy tính. Để thể hiện được dữ liệu nhận và truyền trên máy tính, ta cần sử dụng một trong các phần mềm hỗ trợ sau:

- Serial Debug Assistant (tải tại Microsoft Store).
- Hercules Terminal (tải tại đây).

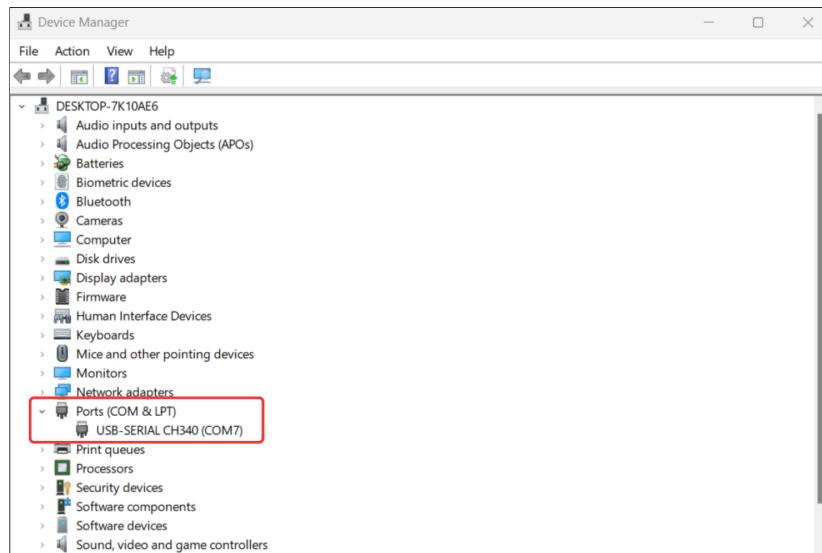
Phần hướng dẫn này sẽ sử dụng phần mềm **Hercules Terminal**.



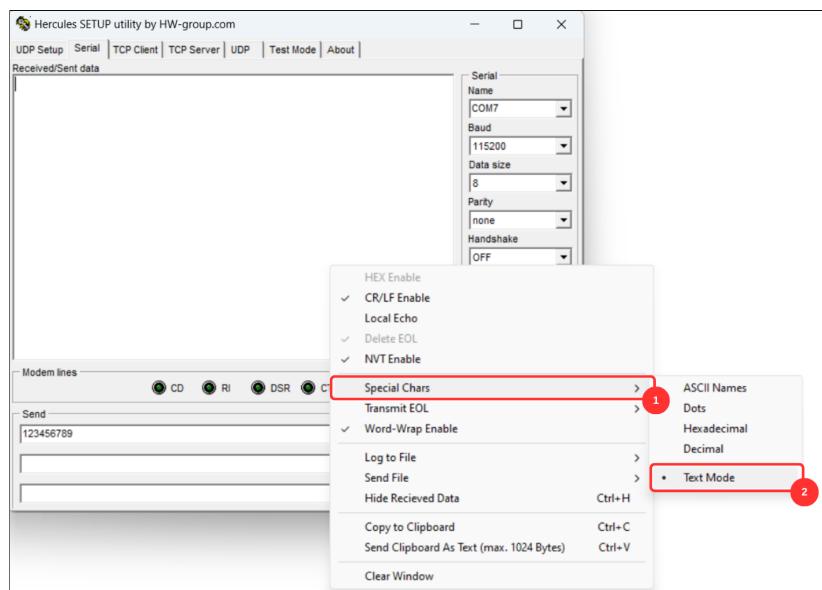
Hình 5.8: Giao diện phần mềm Hercules Terminal

Để phần mềm có thể gửi và nhận dữ liệu, ta cần chọn cổng COM đúng với cổng đang kết nối với dây cáp RS232. Ta cần phải kiểm tra cổng COM đang sử dụng trong **Device Manager** (Window + X -> Device Manager).

Như hình trên ta đang sử dụng cổng COM7. Ta tiến hành điều chỉnh các thông số **Name** thành COM7, thông số **Baud**, **Data size**, **Parity** giống với những gì ta đã config UART trên Kit thí nghiệm. Sau đó chọn **Open**. Ta cũng cần phải chỉnh chế độ hiển thị dữ liệu để có thể dễ dàng quan sát.



Hình 5.9: Kiểm tra cổng COM



Hình 5.10: Chính chế độ hiển thi

Bây giờ chúng ta sẽ hiện thực gửi UART với dữ liệu là thời gian lấy được từ module DS3231 mỗi khi nút "E" trên ma trận phím được nhấn. Đầu tiên, chúng ta khởi tạo module UART-RS232 trong **system_init()** bằng hàm **uart_rs232_init()**. Sau đó, một hàm mới **test_Uart()** được tạo và gọi mỗi 50ms. Trong hàm này ta sẽ bắt tín hiệu nút nhấn "E" trên Kit thí nghiệm như đã hiện thực ở lab 3. Mỗi khi nút "E" được nhấn, ta sẽ gọi các hàm để gửi dữ liệu giờ, phút, giây theo định dạng **<giờ>:<phút>:<giây>** sang máy tính thông qua module UART-RS232 (dữ liệu về thời gian sẽ được đọc từ đồng hồ thời gian thực như ở lab 4). Để thuận tiện cho việc quan sát dữ liệu trên màn hình máy tính, ta gửi thêm kí tự xuống dòng ('\n').

1 // ...

```

2
3 /* USER CODE BEGIN Includes */
4 #include "software_timer.h"
5 #include "led_7seg.h"
6 #include "button.h"
7 #include "lcd.h"
8 #include "picture.h"
9 #include "ds3231.h"
10 #include "uart.h"
11 /* USER CODE END Includes */
12
13 // ...
14
15 /* USER CODE BEGIN PFP */
16 void system_init();
17 void test_LedDebug();
18 void test_Uart();
19 /* USER CODE END PFP */
20
21 /* Private user code
22 -----
23 */
24 /* USER CODE BEGIN 0 */
25
26 /**
27 * @brief The application entry point.
28 * @retval int
29 */
30 int main(void)
31 {
32 // ...
33 /* USER CODE BEGIN 2 */
34 system_init();
35 /* USER CODE END 2 */
36
37 /* Infinite loop */
38 /* USER CODE BEGIN WHILE */

```

```

39 while (1)
40 {
41     /* USER CODE END WHILE */
42     while(!flag_timer2);
43     flag_timer2 = 0;
44     button_Scan();
45     test_LedDebug();
46     ds3231_ReadTime();
47     test_Uart();
48     /* USER CODE BEGIN 3 */
49 }
50 /* USER CODE END 3 */
51 }
52
53 // ...
54
55 /* USER CODE BEGIN 4 */
56 void system_init(){
57     timer_init();
58     led7_init();
59     button_init();
60     lcd_init();
61     uart_init_rs232();
62     setTimer2(50);
63 }
64
65 uint16_t count_led_debug = 0;
66
67 void test_LedDebug(){
68     count_led_debug = (count_led_debug + 1)%20;
69     if(count_led_debug == 0){
70         HAL_GPIO_TogglePin(DEBUG_LED_GPIO_Port, DEBUG_LED_Pin);
71     }
72 }
73
74 void test_Uart(){
75     if(button_count[12] == 1){
76         uart_Rs232SendNum(ds3231_hours);
77         uart_Rs232SendString(":");

```

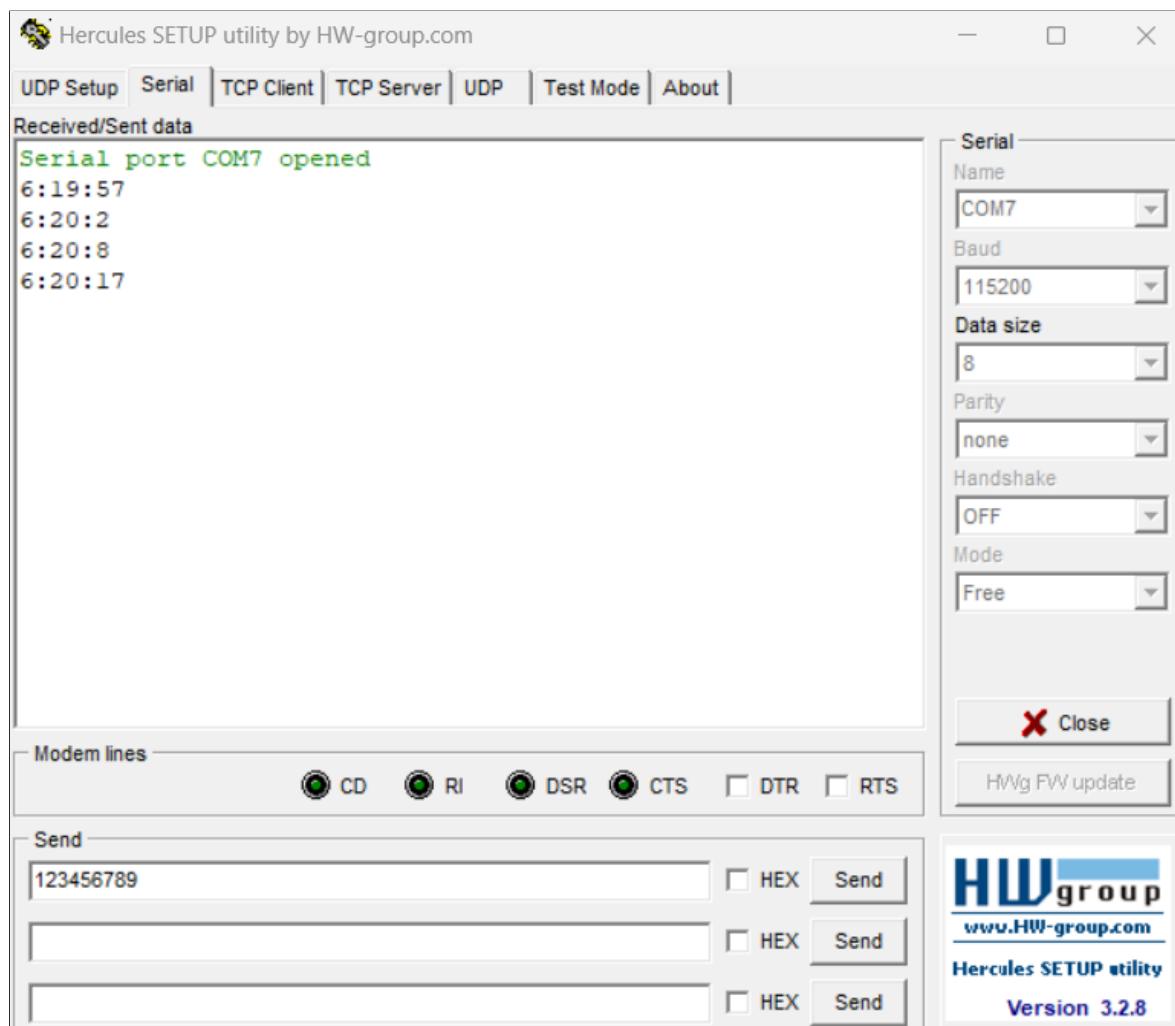
```

78     uart_Rs232SendNum(ds3231_min);
79     uart_Rs232SendString(":");
80     uart_Rs232SendNum(ds3231_sec);
81     uart_Rs232SendString("\n");
82 }
83 */
84 /* USER CODE END 4 */
85 // ...

```

Program 5.2: main.c

Kết quả:



Hình 5.11: Kết quả thu được từ ví dụ mẫu

6 Bài tập và báo cáo

6.1 Bài tập 1

Sử dụng ring buffer để lưu trữ các dữ liệu nhận được trong **Uart Receive Interrupt**. Sau đó bật cờ và xử lí dữ liệu nhận được trong màn main.

6.2 Bài tập 2

Nâng cấp bài tập về đồng hồ điện tử đã được hiện thực ở lab 4. Thêm một chế độ cho phép cập nhật thời gian thông qua giao tiếp RS232 với máy tính. Ở chế độ này, các giá trị thời gian sẽ được cập nhật lần lượt. Sau đây là một ví dụ khi hệ thống yêu cầu cập nhật giờ:

- Màn hình LCD sẽ hiển thị thêm dòng chữ "Updating hours ..."
- Kit thí nghiệm sẽ gửi dữ liệu **request** "Hours" đến máy tính.
- Máy tính sẽ gửi lại dữ liệu **response** là chuỗi kí tự chứa giá trị giờ muốn cập nhật. Gợi ý: sử dụng chế độ gửi dữ liệu của phần mềm **Hercules**.
- Hệ thống lưu lại giá trị vừa nhận được và chuyển sang yêu cầu cập nhật giá trị tiếp theo.
- Sau khi hệ thống đã nhận được tất cả giá trị thời gian, hệ thống sẽ lưu lại giá trị đó vào IC thời gian thực DS3231.

6.3 Bài tập 3

Tiếp tục phát triển ứng dụng ở bài tập 2, thêm các chức năng sau:

- Tại thời điểm 10s sau khi hệ thống gửi request đến máy tính, nếu không có phản hồi thì hệ thống sẽ gửi lại request. Nếu sau 3 lần gửi mà không được phản hồi, hệ thống sẽ báo lỗi thông qua LCD và quay về chế độ hoạt động bình thường.
- Nếu dữ liệu response từ máy tính không hợp lệ, hệ thống sẽ gửi lại request.