



HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY  
COMPUTER ENGINEERING

# Microcontroller



Dr. Le Trong Nhan

- **Name:** HOANG VAN PHI
- **MSSV:** 2252608
- **Class:** CO3010\_CC01
- **Teacher:** NGUYEN THIEN AN
- **Lab:** 5
- **Link GitHub:** <https://github.com/PhiuTheWind/VXL>

---

# Mục lục

---

<b>Chapter 1. LED Animations</b>	<b>7</b>
1 Exercise and Report . . . . .	8
1.1 Exercise 1 . . . . .	8
1.2 Exercise 2 . . . . .	9
1.3 Exercise 3 . . . . .	11
1.4 Exercise 4 . . . . .	13
1.5 Exercise 5 . . . . .	18
1.6 Exercise 6 . . . . .	21
1.7 Exercise 7 . . . . .	23
1.8 Exercise 8 . . . . .	23
1.9 Exercise 9 . . . . .	23
1.10 Exercise 10 . . . . .	24
<b>Chapter 2. Timer Interrupt and LED Scanning</b>	<b>27</b>
1 Exercise and Report . . . . .	28
1.1 Exercise 1 . . . . .	28
1.2 Exercise 2 . . . . .	29
1.3 Exercise 3 . . . . .	31
1.4 Exercise 4 . . . . .	33
1.5 Exercise 5 . . . . .	34
1.6 Exercise 6 . . . . .	35
1.7 Exercise 7 . . . . .	37
1.8 Exercise 8 . . . . .	38
1.9 Exercise 9 . . . . .	40
1.10 Exercise 10 . . . . .	44
<b>Chapter 3. Buttons/Switches</b>	<b>49</b>

1	Objectives . . . . .	50
2	Introduction . . . . .	50
3	Basic techniques for reading from port pins . . . . .	52
3.1	The need for pull-up resistors . . . . .	52
3.2	Dealing with switch bounces . . . . .	52
4	Reading switch input (basic code) using STM32 . . . . .	57
4.1	Input Output Processing Patterns . . . . .	57
4.2	Setting up . . . . .	58
4.2.1	Create a project . . . . .	58
4.2.2	Create a file C source file and header file for input reading . . . . .	58
4.3	Code For Read Port Pin and Debouncing . . . . .	60
4.3.1	The code in the input_reading.c file . . . . .	60
4.3.2	The code in the input_reading.h file . . . . .	61
4.3.3	The code in the timer.c file . . . . .	61
4.4	Button State Processing . . . . .	62
4.4.1	Finite State Machine . . . . .	62
4.4.2	The code for the FSM in the input_processing.c file . . . . .	63
4.4.3	The code in the input_processing.h . . . . .	63
4.4.4	The code in the main.c file . . . . .	64
5	Exercises and Report . . . . .	65
5.1	Specifications . . . . .	65
5.2	Exercise 1: Sketch an FSM . . . . .	66
5.3	Exercise 2: Proteus Schematic . . . . .	66
5.4	Exercise 3: Create STM32 Project . . . . .	67
5.5	Exercise 4: Modify Timer Parameters . . . . .	67
5.6	Exercise 5: Adding code for button debouncing . . . . .	68
5.7	Exercise 6: Adding code for displaying modes . . . . .	73
5.8	Exercise 7: Adding code for increasing time duration value for the red LEDs . . . . .	79
5.9	Exercise 8: Adding code for increasing time duration value for the amber LEDs . . . . .	79
5.10	Exercise 9: Adding code for increasing time duration value for the green LEDs . . . . .	79
5.11	Exercise 10: To finish the project . . . . .	82

<b>Chapter 4. Digital Clock Project</b>	<b>83</b>
<b>Chapter 5. A cooperative scheduler</b>	<b>85</b>
1 Introduction . . . . .	86
1.1 Super Loop Architecture . . . . .	86
1.2 Timer-based interrupts and interrupt service routines . . . . .	87
2 What is a scheduler? . . . . .	88
2.1 The co-operative scheduler . . . . .	88
2.2 Function pointers . . . . .	89
2.3 Solution . . . . .	90
2.3.1 Overview . . . . .	91
2.3.2 The scheduler data structure and task array . . . . .	92
2.3.3 The initialization function . . . . .	93
2.3.4 The 'Update' function . . . . .	93
2.3.5 The 'Add Task' function . . . . .	94
2.3.6 The 'Dispatcher' . . . . .	95
2.3.7 The 'Delete Task' function . . . . .	97
2.3.8 Reducing power consumption . . . . .	97
2.3.9 Reporting errors . . . . .	98
2.3.10 Adding a watchdog . . . . .	99
2.3.11 Reliability and safety implications . . . . .	100
2.3.12 Portability . . . . .	100
3 Objectives . . . . .	100
4 Problem . . . . .	101
5 Demonstration . . . . .	101
6 Submission . . . . .	102
7 References . . . . .	105
<b>Chapter 6. Flow and Error Control in Communication</b>	<b>107</b>
1 Introduction . . . . .	108
2 Proteus simulation platform . . . . .	109
3 Project configurations . . . . .	110
3.1 UART Configuration . . . . .	110
3.2 ADC Input . . . . .	111
4 UART loop-back communication . . . . .	111
5 Sensor reading . . . . .	112

6	Project description . . . . .	113
6.1	Command parser . . . . .	113
6.2	Project implementation . . . . .	114

# CHƯƠNG 6

## Flow and Error Control in Communication



# 1 Introduction

Flow control and Error control are the two main responsibilities of the data link layer, which is a communication channel for node-to-node delivery of the data. The functions of the flow and error control are explained as follows.

Flow control mainly coordinates with the amount of data that can be sent before receiving an acknowledgment from the receiver and it is one of the major duties of the data link layer. For most of the communications, flow control is a set of procedures that mainly tells the sender how much data the sender can send before it must wait for an acknowledgment from the receiver.

A critical issue, but not really frequently occurred, in the flow control is that the processing rate is slower than the transmission rate. Due to this reason each receiving device has a block of memory that is commonly known as buffer, that is used to store the incoming data until this data will be processed. In case the buffer begins to fill-up then the receiver must be able to tell the sender to halt the transmission until once again the receiver become able to receive.

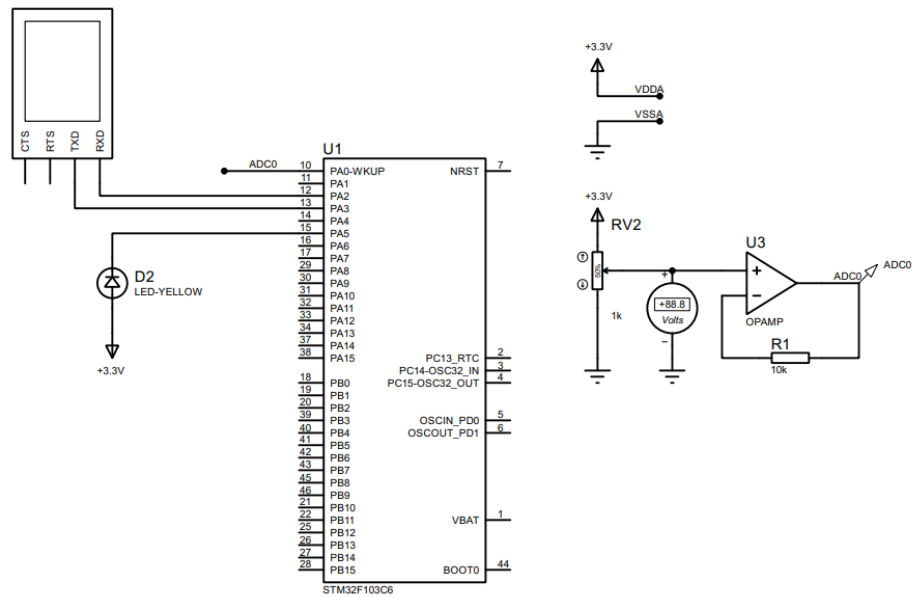
Meanwhile, error control contains both error detection and error correction. It mainly allows the receiver to inform the sender about any damaged or lost frames during the transmission and then it coordinates with the re-transmission of those frames by the sender.

The term Error control in the communications mainly refers to the methods of error detection and re-transmission. Error control is mainly implemented in a simple way and that is whenever there is an error detected during the exchange, then specified frames are re-transmitted and this process is also referred to as Automatic Repeat request (ARQ).

The target in this lab is to implement a UART communication between the STM32 and a simulated terminal. A data request is sent from the terminal to the STM32. Afterward, computations are performed at the STM32 before a data packet is sent to the terminal. The terminal is supposed to reply an ACK to confirm the communication successfully or not.



## 2 Proteus simulation platform



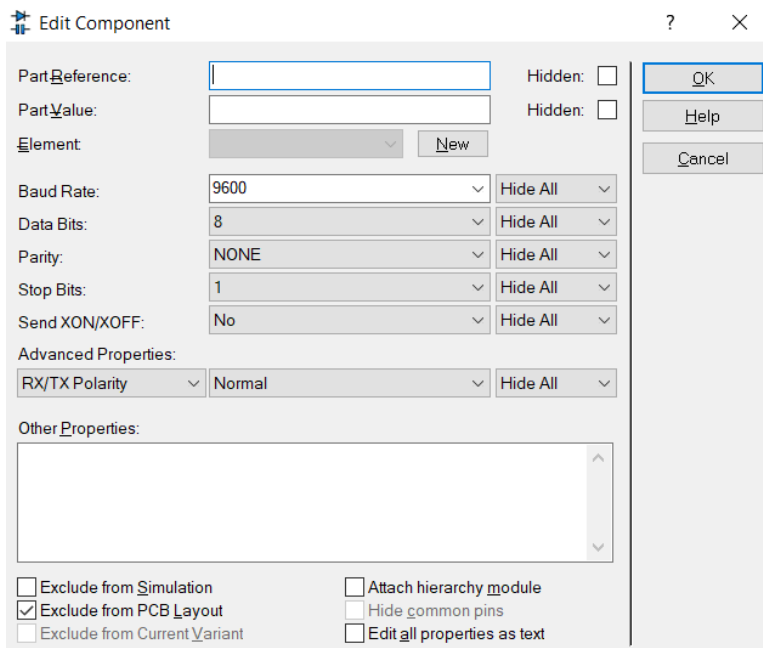
Hình 6.1: Simulation circuit on Proteus

Some new components are listed bellow:

- Terminal: Right click, choose Place, Virtual Instrument, then select VIRTUAL TERMINAL.
- Variable resistor (RV2): Right click, choose Place, From Library, and search for the POT-HG device. The value of this device is set to the default 1k.
- Volt meter (for debug): Right click, choose Place, Virtual Instrument, the select DC VOLTMETER.
- OPAMP (U3): Right click, choose Place, From Library, and search for the OPAMP device.

The opamp is used to design a voltage follower circuit, which is one of the most popular applications for opamp. In this case, it is used to design an adc input signal, which is connected to pin PA0 of the MCU.

Double click on the virtual terminal and set its baudrate to 9600, 8 data bits, no parity and 1 stop bit, as follows:



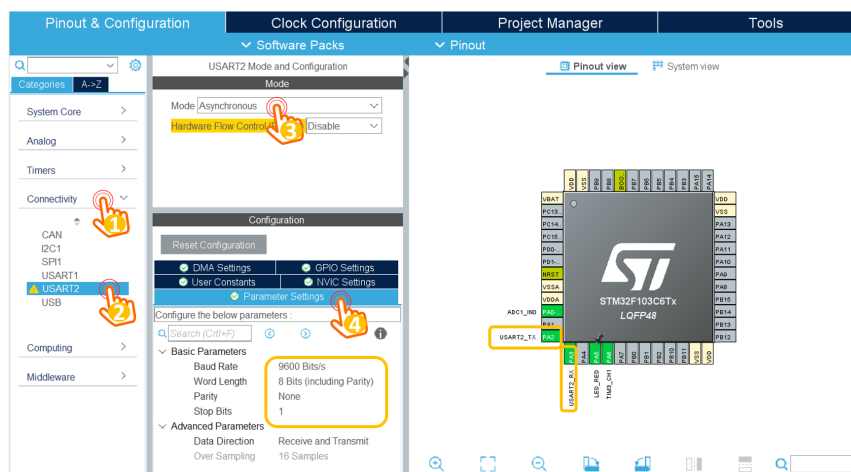
Hình 6.2: Terminal configuration

### 3 Project configurations

A new project is created with following configurations, concerning the UART for communications and ADC input for sensor reading. The pin PA5 should be an GPIO output, for LED blinky.

#### 3.1 UART Configuration

From the ioc file, select **Connectivity**, and then select the **USART2**. The parameter settings for UART channel 2 (USART2) module are depicted as follows:



Hình 6.3: UART configuration in STMCube

The UART channel in this lab is the Asynchronous mode, 9600 bits/s with no Parity and 1 stop bit. After the uart is configured, the pins PA2 (Tx) and PA3(Rx) are enabled.

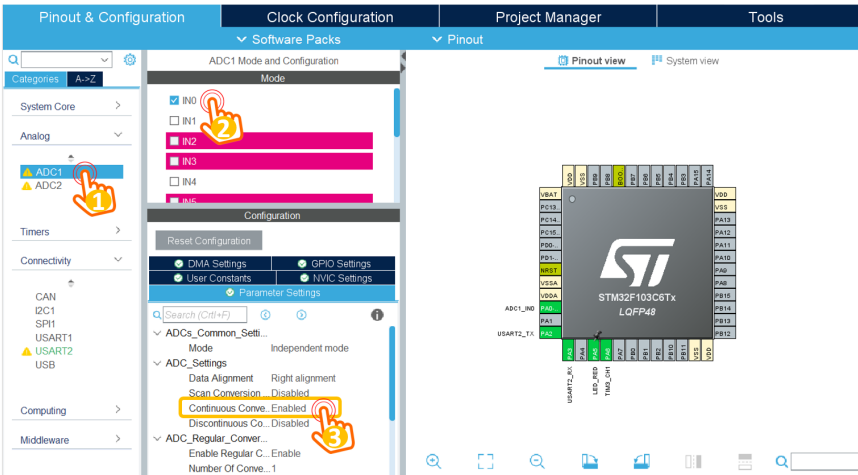
Finally, the NVIC settings are checked to enable the UART interrupt, as follows:

<input checked="" type="checkbox"/> DMA Settings	<input checked="" type="checkbox"/> GPIO Settings	
<input checked="" type="checkbox"/> User Constants	<input checked="" type="checkbox"/> NVIC Settings	
<input checked="" type="checkbox"/> Parameter Settings		
NVIC Interrupt Table	Enabled	Preemption P
USART2 global interrupt	<input checked="" type="checkbox"/>	0

Hình 6.4: Enable UART interrupt

### 3.2 ADC Input

In order to read a voltage signal from a simulated sensor, this module is required. By selecting on **Analog**, then **ADC1**, following configurations are required:



Hình 6.5: Enable UART interrupt

The ADC pin is configured to PA0 of the STM32, which is shown in the pinout view dialog.

Finally, the PA5 is configured as a GPIO output, connected to a blinky LED.

## 4 UART loop-back communication

This source is required to add in the main.c file, to verify the UART communication channel: sending back any character received from the terminal, which is well-known as the loop-back communication.

```

1  /* USER CODE BEGIN 0 */
2  uint8_t temp = 0;
3
4  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
5      if(huart->Instance == USART2){
6          HAL_UART_Transmit(&huart2, &temp, 1, 50);
7          HAL_UART_Receive_IT(&huart2, &temp, 1);
8      }
9  }
10 /* USER CODE END 0 */

```

Program 6.1: Implement the UART interrupt service routine

When a character (or a byte) is received, this interrupt service routine is invoked. After the character is sent to the terminal, the interrupt is activated again. This source code should be placed in a user-defined section.

Finally, in the main function, the proposed source code is presented as follows:

```

1  int main(void)
2  {
3      HAL_Init();
4      SystemClock_Config();
5
6      MX_GPIO_Init();
7      MX_USART2_UART_Init();
8      MX_ADC1_Init();
9
10     HAL_UART_Receive_IT(&huart2, &temp, 1);
11
12     while (1)
13     {
14         HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
15         HAL_Delay(500);
16     }
17
18 }

```

Program 6.2: Implement the main function

## 5 Sensor reading

A simple source code to read adc value from PA0 is presented as follows:

```

1  uint32_t ADC_value = 0;
2  while (1)
3  {
4      HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
5      ADC_value = HAL_ADC_GetValue(&hadc1);

```

```

6 HAL_UART_Transmit(&huart2, (void *)str, sprintf(str, "%d\n"
    , ADC_value), 1000);
7     HAL_Delay(500);
8 }

```

Program 6.3: ADC reading from AN0

Every half of second, the ADC value is read and its value is sent to the console. It is worth noticing that the number ADC\_value is convert to ascii character by using the sprintf function.

The default ADC in STM32 is 13 bits, meaning that 5V is converted to 4096 decimal value. If the input is 2.5V, ADC\_value is 2048.

## 6 Project description

In this lab, a simple communication protocol is implemented as follows:

- From the console, user types **!RST#** to ask for a sensory data.
- The STM32 response the ADC\_value, following a format **!ADC=1234#**, where 1234 presents for the value of ADC\_value variable.
- The user ends the communication by sending **!OK#**

The timeout for waiting the **!OK#** at STM32 is 3 seconds. After this period, its packet is sent again. **The value is kept as the previous packet.**

### 6.1 Command parser

This module is used to received a command from the console. As the reception process is implement by an interrupt, the complexity is considered seriously. The proposed implementation is given as follows.

Firstly, the received character is added into a buffer, and a flag is set to indicate that there is a new data.

```

1 #define MAX_BUFFER_SIZE 30
2 uint8_t temp = 0;
3 uint8_t buffer[MAX_BUFFER_SIZE];
4 uint8_t index_buffer = 0;
5 uint8_t buffer_flag = 0;
6 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
7     if(huart->Instance == USART2){
8
9         //HAL_UART_Transmit(&huart2, &temp, 1, 50);
10        buffer[index_buffer++] = temp;
11        if(index_buffer == 30) index_buffer = 0;

```

```

12
13     buffer_flag = 1;
14     HAL_UART_Receive_IT(&huart2, &temp, 1);
15 }
16 }

```

Program 6.4: Add the received character into a buffer

A state machine to extract a command is implemented in the while(1) of the main function, as follows:

```

1 while (1){
2     if(buffer_flag == 1){
3         command_parser_fsm();
4         buffer_flag = 0;
5     }
6 }

```

Program 6.5: State machine to extract the command

The output of the command parser is to set **command\_flag** and **command\_data**. In this project, there are two commands, **RTS** and **OK**. The program skeleton is proposed as follows:

```

1 while (1){
2     if(buffer_flag == 1){
3         command_parser_fsm();
4         buffer_flag = 0;
5     }
6     uart_communiation_fsm();
7 }

```

Program 6.6: Program structure

## 6.2 Project implementation

Students are proposed to implement 2 FSM in seperated modules. Students are asked to design the FSM before their implementations in STM32Cube.

```

1 /*
2  * uart.h
3  *
4  * Created on: Dec 9, 2024
5  * Author: phihv
6  */
7
8 #ifndef INC_UART_H_
9 #define INC_UART_H_
10
11 #include <stdint.h>
12 #include <stdio.h>
13 #include <stdlib.h>

```

```

14 #include <main.h>
15 #include <string.h>
16 #include <timer.h>
17
18 #define INIT 1
19 #define PARSER 2
20 #define WAIT 3
21 #define SENDING 4
22 #define END 5
23 #define MAX_BUFFER_SIZE 30
24
25 extern UART_HandleTypeDef huart2;
26 extern ADC_HandleTypeDef hadc1;
27
28 extern uint16_t ADC_value;
29 extern uint8_t temp;
30 extern uint8_t text[MAX_BUFFER_SIZE];
31 extern uint8_t index_text;
32
33 extern uint8_t buffer[MAX_BUFFER_SIZE];
34 extern uint8_t index_buffer;
35 extern uint8_t buffer_flag;
36 extern uint16_t old_value;
37
38 extern int parser_status;
39 extern int uart_status;
40 extern int RST_flag;
41 extern int has_ok;
42 extern int check_first;
43
44 extern char str[MAX_BUFFER_SIZE];
45
46 void check_content();
47 void clean_contet();
48 void command_parser_fsm();
49 void uart_communication_fsm();
50
51
52 #endif /* INC_UART_H_ */

```

Program 6.7: uart.h

```

1 /*
2  * uart.c
3  *
4  * Created on: Dec 9, 2024
5  * Author: phihv
6  */
7
8

```

```

9  #include "uart.h"
10
11
12 uint16_t ADC_value = 0;
13 uint16_t old_value = 0;
14
15 uint8_t text[MAX_BUFFER_SIZE]= {0};
16 uint8_t index_text = 0;
17
18 uint8_t temp = 0;
19 uint8_t buffer[MAX_BUFFER_SIZE]={0};
20 uint8_t index_buffer = 0;
21 uint8_t buffer_flag = 0;
22
23 int has_ok = 0;
24 int check_first = 1;
25 int parser_status = 0;
26 int uart_status = 0;
27 int RST_flag = 0;
28
29
30 char str[MAX_BUFFER_SIZE]={0};
31
32 void clean_content(){
33     memset(text, 0, sizeof(text));
34 }
35
36 void check_content(){
37     if(text[0]=='R' && text[1]=='S' && text[2]=='T' && text
38         [3] == '#'){ // If !RST# => flag on
39         RST_flag=1;
40         clean_content();
41     }
42     else if(text[0]=='O' && text[1]=='K' && text[2]=='#' &&
43         RST_flag==1){ // If !OK# => end
44         uart_status=END;
45         RST_flag=0;
46     }
47     else if((text[0]!='R' || text[1]!='S' || text[2]!='T' ||
48         text[3] != '#') && RST_flag==0){ // Wrong syntax content
49         uart_status=END;
50         HAL_UART_Transmit(&huart2, (uint8_t*)str, sprintf(str,
51             "Wrong syntax \n"), 1000);
52     }
53     else{ // The last case user have not enter "OK"
54         uart_status=SENDING;
55     }
56 }
57 void command_parser_fsm(){

```



```

54 switch(parser_status){
55 case INIT:
56     if(temp == '!'){ // If the first is !, it will move to
                        the parser to add the char into buffer, else, do not
                        thing
57         parser_status=PARSER;
58         index_text = 0;
59     }
60     break;
61 case PARSER: // Check if Enter was push or not, if Push,
                move to check content between ! and #, else continue
                save data
62     if(temp == '\r'){
63         check_content();
64         parser_status=INIT;
65     }
66     else if(temp != '\r'){
67         text[index_text] = temp;
68         index_text++;
69         if(index_text == MAX_BUFFER_SIZE){
70             index_text = 0;
71         }
72     }
73     break;
74 default:
75     break;
76 }
77 }
78
79 void uart_communication_fsm(){
80     switch(uart_status){
81     case WAIT: // Wait the signal of RST_flag
82         if(RST_flag==1){
83             uart_status=SENDING;
84             setTimer2(200);
85         }
86         break;
87     case SENDING: //Sending data when RST_flag = 1 and timer
                    2 count down to 0
88         if (timer2_flag == 1) {
89             if(check_first == 1)
90             {
91                 ADC_value= HAL_ADC_GetValue(&hadc1);
92                 HAL_UART_Transmit(&huart2, (uint8_t*)str, sprintf(
str, "!ADC=%d#\r\n", ADC_value), 1000);
93                 old_value = ADC_value;
94                 check_first = 0;
95                 setTimer2(300);
96             }

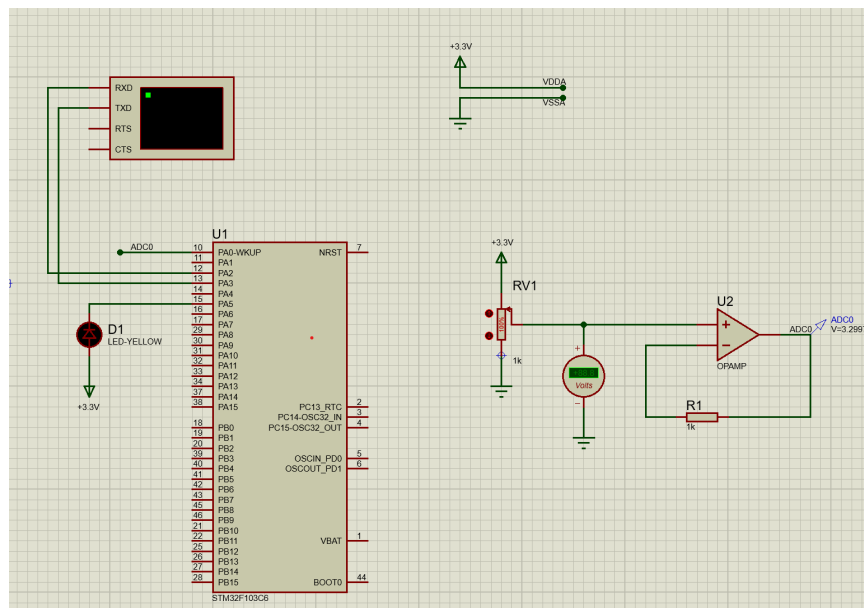
```

```

97     else if(check_first == 0)
98     {
99         HAL_UART_Transmit(&huart2, (uint8_t*)str, sprintf(
100 str, "!ADC=%d#\r\n", old_value), 1000);
101         setTimer2(300);
102     }
103 }
104 break;
105 case END: // clear memory and back to Wait for new
106 communication
107     clean_content();
108     uart_status = WAIT;
109     HAL_UART_Transmit(&huart2, (uint8_t*)str, sprintf(str,
110 "End. \n"), 1000);
111     check_first = 1;
112     break;
113 default:
114     break;
115 }
116 }

```

Program 6.8: uart.c



Hình 6.6: Proteus Simulation

```

1  /* USER CODE BEGIN Header */
2  /**
3
4   * @file           : main.c
5   * @brief          : Main program body
6
7   * @attention
8   *

```

```

9  * Copyright (c) 2024 STMicroelectronics.
10 * All rights reserved.
11 *
12 * This software is licensed under terms that can be found
    in the LICENSE file
13 * in the root directory of this software component.
14 * If no LICENSE file comes with this software, it is
    provided AS-IS.
15 *
16
    *****

17 */
18 /* USER CODE END Header */
19 /* Includes
    -----
    */
20 #include "main.h"
21 #include "uart.h"
22 #include "timer.h"
23 /* Private includes
    -----
    */
24 /* USER CODE BEGIN Includes */
25
26 /* USER CODE END Includes */
27
28 /* Private typedef
    -----
    */
29 /* USER CODE BEGIN PTD */
30
31 /* USER CODE END PTD */
32
33 /* Private define
    -----
    */
34 /* USER CODE BEGIN PD */
35
36 /* USER CODE END PD */
37
38 /* Private macro
    -----
    */
39 /* USER CODE BEGIN PM */
40
41 /* USER CODE END PM */
42
43 /* Private variables

```

```

-----
*/
44 ADC_HandleTypeDef hadc1;
45
46 TIM_HandleTypeDef htim2;
47
48 UART_HandleTypeDef huart2;
49
50 /* USER CODE BEGIN PV */
51
52 /* USER CODE END PV */
53
54 /* USER CODE BEGIN PFP */
55
56 /* USER CODE END PFP */
57
58 /* Private user code
-----
*/
59 /* USER CODE BEGIN 0 */
60
61 /* USER CODE END 0 */
62
63 /**
64  * @brief The application entry point.
65  * @retval int
66  */
67
68 /* USER CODE BEGIN PV */
69 /* USER CODE END PV */
70
71 /* Private function prototypes
-----*/
72 void SystemClock_Config(void);
73 static void MX_GPIO_Init(void);
74 static void MX_ADC1_Init(void);
75 static void MX_USART2_UART_Init(void);
76 static void MX_TIM2_Init(void);
77 /* USER CODE BEGIN 0 */
78 /* USER CODE END 0 */
79
80 int main(void) {
81     /* USER CODE BEGIN 1 */
82
83     /* USER CODE END 1 */
84
85     /* Initialize all configured peripherals */
86     MX_GPIO_Init();
87     MX_ADC1_Init();

```

```

88     MX_USART2_UART_Init();
89     MX_TIM2_Init();
90     /* USER CODE BEGIN 2 */
91     HAL_TIM_Base_Start_IT(&htim2);
92     HAL_UART_Receive_IT (&huart2 , &temp , 1) ;
93     HAL_ADC_Start(&hadc1);
94
95     setTimer1(50);
96     parser_status = INIT;
97     uart_status = WAIT;
98     /* USER CODE END 2 */
99
100    /* Infinite loop */
101    /* USER CODE BEGIN WHILE */
102    while (1)
103    {
104        if(timer1_flag == 1)
105        {
106            HAL_GPIO_TogglePin(LED_YELLOW_GPIO_Port ,
LED_YELLOW_Pin);
107            setTimer1(50);
108        }
109        if(buffer_flag == 1)
110        {
111            command_parser_fsm();
112            buffer_flag=0;
113        }
114        uart_communication_fsm();
115
116        /* USER CODE END WHILE */
117
118        /* USER CODE BEGIN 3 */
119    }
120    /* USER CODE END 3 */
121 }
122
123 /**
124  * @brief Callback UART khi nhận dữ liệu
125  */
126 void HAL_UART_RxCpltCallback ( UART_HandleTypeDef * huart )
127 {
128     if(huart -> Instance == USART2 ){
129         buffer[index_buffer] = temp;
130         index_buffer++;
131         if( index_buffer == 30) index_buffer = 0;
132
133         buffer_flag = 1;
134         HAL_UART_Transmit(&huart2, &temp, 1, 1000);
135         HAL_UART_Receive_IT (&huart2 , &temp , 1);

```

```

135 }
136 }
137
138
139
140 /**
141  * @brief Callback Timer n g t      nh      k
142  */
143 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
144 {
145     if (htim->Instance == TIM2) {
146         timerRun(); // G i h m x l í timer
147     }
148 }
149
150 /**
151  * @brief System Clock Configuration
152  * @retval None
153  */
154 void SystemClock_Config(void)
155 {
156     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
157     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
158     RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
159
160     /** Initializes the RCC Oscillators according to the
161     specified parameters
162     * in the RCC_OscInitTypeDef structure.
163     */
164     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI
165     ;
166     RCC_OscInitStruct.HSISState = RCC_HSI_ON;
167     RCC_OscInitStruct.HSICalibrationValue =
168     RCC_HSICALIBRATION_DEFAULT;
169     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
170     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
171     {
172         Error_Handler();
173     }
174
175     /** Initializes the CPU, AHB and APB buses clocks
176     */
177     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK |
178     RCC_CLOCKTYPE_SYSCLK
179     | RCC_CLOCKTYPE_PCLK1 |
180     RCC_CLOCKTYPE_PCLK2;
181     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
182     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
183     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

```

```

178 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
179
180 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
181     FLASH_LATENCY_0) != HAL_OK)
182 {
183     Error_Handler();
184 }
185 PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
186 PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;
187 if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
188 {
189     Error_Handler();
190 }
191
192 /**
193  * @brief ADC1 Initialization Function
194  * @param None
195  * @retval None
196  */
197 static void MX_ADC1_Init(void)
198 {
199
200     /* USER CODE BEGIN ADC1_Init 0 */
201
202     /* USER CODE END ADC1_Init 0 */
203
204     ADC_ChannelConfTypeDef sConfig = {0};
205
206     /* USER CODE BEGIN ADC1_Init 1 */
207
208     /* USER CODE END ADC1_Init 1 */
209
210     /** Common config
211     */
212     hadc1.Instance = ADC1;
213     hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
214     hadc1.Init.ContinuousConvMode = ENABLE;
215     hadc1.Init.DiscontinuousConvMode = DISABLE;
216     hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
217     hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
218     hadc1.Init.NbrOfConversion = 1;
219     if (HAL_ADC_Init(&hadc1) != HAL_OK)
220     {
221         Error_Handler();
222     }
223
224     /** Configure Regular Channel
225     */

```

```

226     sConfig.Channel = ADC_CHANNEL_0;
227     sConfig.Rank = ADC_REGULAR_RANK_1;
228     sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
229     if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
230     {
231         Error_Handler();
232     }
233     /* USER CODE BEGIN ADC1_Init 2 */
234
235     /* USER CODE END ADC1_Init 2 */
236
237 }
238
239 /**
240  * @brief TIM2 Initialization Function
241  * @param None
242  * @retval None
243  */
244 static void MX_TIM2_Init(void)
245 {
246
247     /* USER CODE BEGIN TIM2_Init 0 */
248
249     /* USER CODE END TIM2_Init 0 */
250
251     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
252     TIM_MasterConfigTypeDef sMasterConfig = {0};
253
254     /* USER CODE BEGIN TIM2_Init 1 */
255
256     /* USER CODE END TIM2_Init 1 */
257     htim2.Instance = TIM2;
258     htim2.Init.Prescaler = 7999;
259     htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
260     htim2.Init.Period = 9;
261     htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
262     htim2.Init.AutoReloadPreload =
        TIM_AUTORELOAD_PRELOAD_DISABLE;
263     if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
264     {
265         Error_Handler();
266     }
267     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL
        ;
268     if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig
        ) != HAL_OK)
269     {
270         Error_Handler();
271     }

```



```

272 sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
273 sMasterConfig.MasterSlaveMode =
    TIM_MASTERSLAVEMODE_DISABLE;
274 if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &
    sMasterConfig) != HAL_OK)
275 {
276     Error_Handler();
277 }
278 /* USER CODE BEGIN TIM2_Init 2 */
279
280 /* USER CODE END TIM2_Init 2 */
281
282 }
283
284 /**
285  * @brief USART2 Initialization Function
286  * @param None
287  * @retval None
288  */
289 static void MX_USART2_UART_Init(void)
290 {
291
292     /* USER CODE BEGIN USART2_Init 0 */
293
294     /* USER CODE END USART2_Init 0 */
295
296     /* USER CODE BEGIN USART2_Init 1 */
297
298     /* USER CODE END USART2_Init 1 */
299     huart2.Instance = USART2;
300     huart2.Init.BaudRate = 9600;
301     huart2.Init.WordLength = UART_WORDLENGTH_8B;
302     huart2.Init.StopBits = UART_STOPBITS_1;
303     huart2.Init.Parity = UART_PARITY_NONE;
304     huart2.Init.Mode = UART_MODE_TX_RX;
305     huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
306     huart2.Init.OverSampling = UART_OVERSAMPLING_16;
307     if (HAL_UART_Init(&huart2) != HAL_OK)
308     {
309         Error_Handler();
310     }
311     /* USER CODE BEGIN USART2_Init 2 */
312
313     /* USER CODE END USART2_Init 2 */
314
315 }
316
317 /**
318  * @brief GPIO Initialization Function

```

```

319  * @param None
320  * @retval None
321  */
322 static void MX_GPIO_Init(void)
323 {
324     GPIO_InitTypeDef GPIO_InitStruct = {0};
325     /* USER CODE BEGIN MX_GPIO_Init_1 */
326     /* USER CODE END MX_GPIO_Init_1 */
327
328     /* GPIO Ports Clock Enable */
329     __HAL_RCC_GPIOA_CLK_ENABLE();
330
331     /*Configure GPIO pin Output Level */
332     HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin,
333                       GPIO_PIN_RESET);
334
335     /*Configure GPIO pin : LED_YELLOW_Pin */
336     GPIO_InitStruct.Pin = LED_YELLOW_Pin;
337     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
338     GPIO_InitStruct.Pull = GPIO_NOPULL;
339     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
340     HAL_GPIO_Init(LED_YELLOW_GPIO_Port, &GPIO_InitStruct);
341
342     /* USER CODE BEGIN MX_GPIO_Init_2 */
343     /* USER CODE END MX_GPIO_Init_2 */
344 }
345
346 /* USER CODE BEGIN 4 */
347 /* USER CODE END 4 */
348
349 /**
350  * @brief This function is executed in case of error
351  * occurrence.
352  * @retval None
353  */
354 void Error_Handler(void)
355 {
356     /* USER CODE BEGIN Error_Handler_Debug */
357     /* User can add his own implementation to report the HAL
358     error return state */
359     __disable_irq();
360     while (1)
361     {
362     }
363     /* USER CODE END Error_Handler_Debug */
364 }
365
366 #ifndef USE_FULL_ASSERT

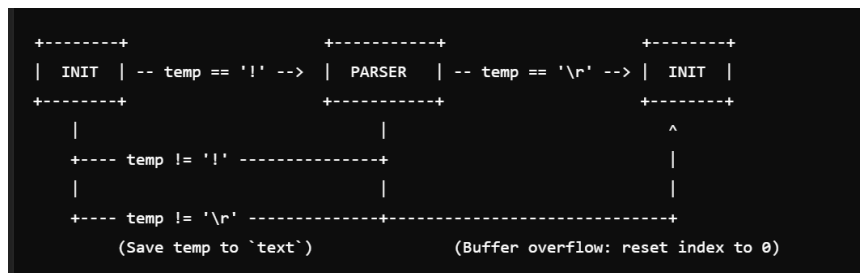
```

```

365 /**
366  * @brief Reports the name of the source file and the
        source line number
367  *         where the assert_param error has occurred.
368  * @param file: pointer to the source file name
369  * @param line: assert_param error line source number
370  * @retval None
371  */
372 void assert_failed(uint8_t *file, uint32_t line)
373 {
374     /* USER CODE BEGIN 6 */
375     /* User can add his own implementation to report the file
        name and line number,
376     ex: printf("Wrong parameters value: file %s on line %d
        \r\n", file, line) */
377     /* USER CODE END 6 */
378 }
379 #endif /* USE_FULL_ASSERT */

```

Program 6.9: main.c



Hình 6.7: *FSM\_command\_parser\_fsm* Simulation



Hình 6.8: *FSM\_command\_parser\_fsm* Simulation