

VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY  
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY  
Faculty of Computer Science and Engineering



---

CC02 — Lab Report

# Microprocessor - Microcontroller Lab 1

---

**Supervisors:** Nguyen Thien An  
**Students:** Nguyen Minh Dang 2252154

Ho Chi Minh City, December 9, 2024



## Contents

<b>1</b>	<b>Exercise</b>	<b>2</b>
1.1	Read ADC value . . . . .	4
1.2	FSM . . . . .	4
1.2.1	FSM for command parser . . . . .	4
1.2.2	FSM for command status . . . . .	5
1.3	UART Communication . . . . .	7
1.3.1	Receive . . . . .	7
1.3.2	Transmit . . . . .	7
1.4	Command processing . . . . .	8
1.4.1	Command parser . . . . .	8
1.4.2	Command status . . . . .	8
1.5	Variables and main() . . . . .	9
1.5.1	Variables . . . . .	9
1.5.2	main() . . . . .	10
<b>2</b>	<b>Problem about this Lab</b>	<b>11</b>
	<b>References</b>	<b>12</b>

## 1 Exercise

The GitHub link for the lab schematics, source code is at [here](https://github.com/dangalpha78/Workspace-for-Microprocessor---Microcontroller/tree/main/Lab5) or in this link: <https://github.com/dangalpha78/Workspace-for-Microprocessor---Microcontroller/tree/main/Lab5>.

The schematic for this lab is located here:

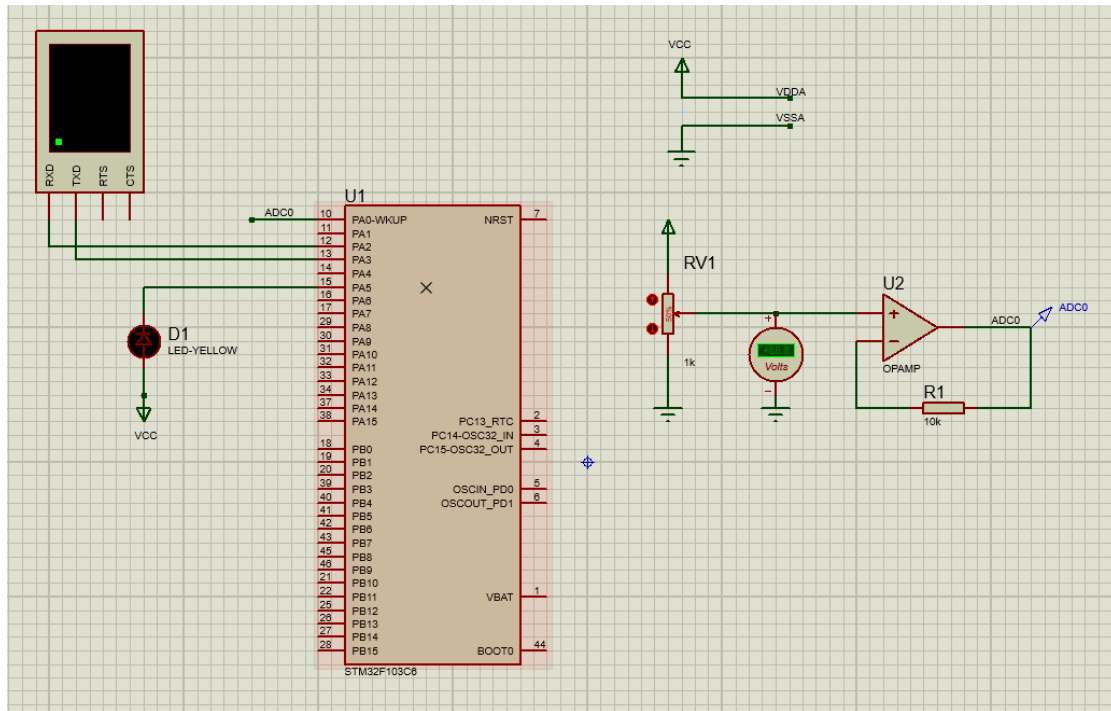


Figure 1: The schematic.

- Virtual Terminal:
  - Simulates serial communication via UART, allowing data exchange between the microcontroller and software such as Proteus.
  - Connected to the TXD (transmit) and RXD (receive) pins of the microcontroller.
  - Used for debugging or sending/receiving text commands during simulation.
- ADC and Operational Amplifier:
  - The analog-to-digital converter (ADC) converts the analog voltage from the op-amp into a digital value for processing.
  - The operational amplifier (U2) is configured as a buffer to prevent signal distortion by providing high input impedance and low output impedance.
  - Ensures that accurate voltage levels are fed to the ADC without affecting the source.
- Potentiometer:
  - A variable resistor used to adjust the voltage at its wiper, which serves as the input to the operational amplifier.



- Allows for manual control of the analog signal level fed into the op-amp and ADC.
- Typically used to test or calibrate the input signal range in the circuit.

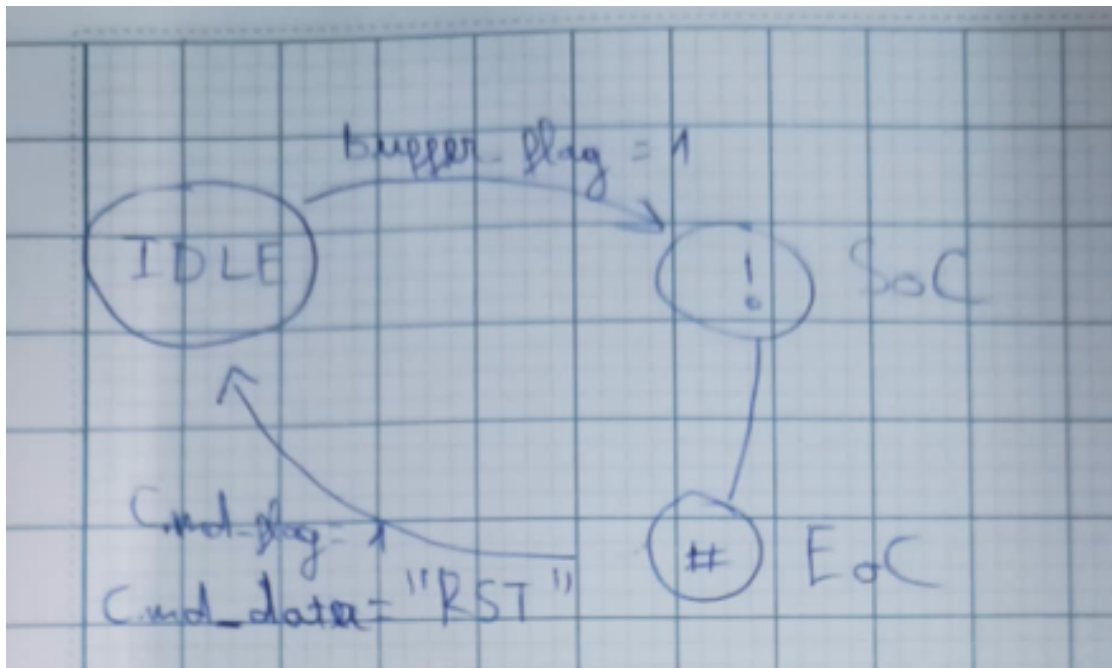
## 1.1 Read ADC value

The `read_ADC` function starts the ADC conversion, waits for it to complete, retrieves the value, and stores it in `ADC_value`, then stops the ADC conversion.

```
1 void read_ADC(void) {
2     HAL_ADC_Start(&hadc1);
3     HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
4     ADC_value = HAL_ADC_GetValue(&hadc1);
5     HAL_ADC_Stop(&hadc1);
6 }
```

## 1.2 FSM

### 1.2.1 FSM for command parser



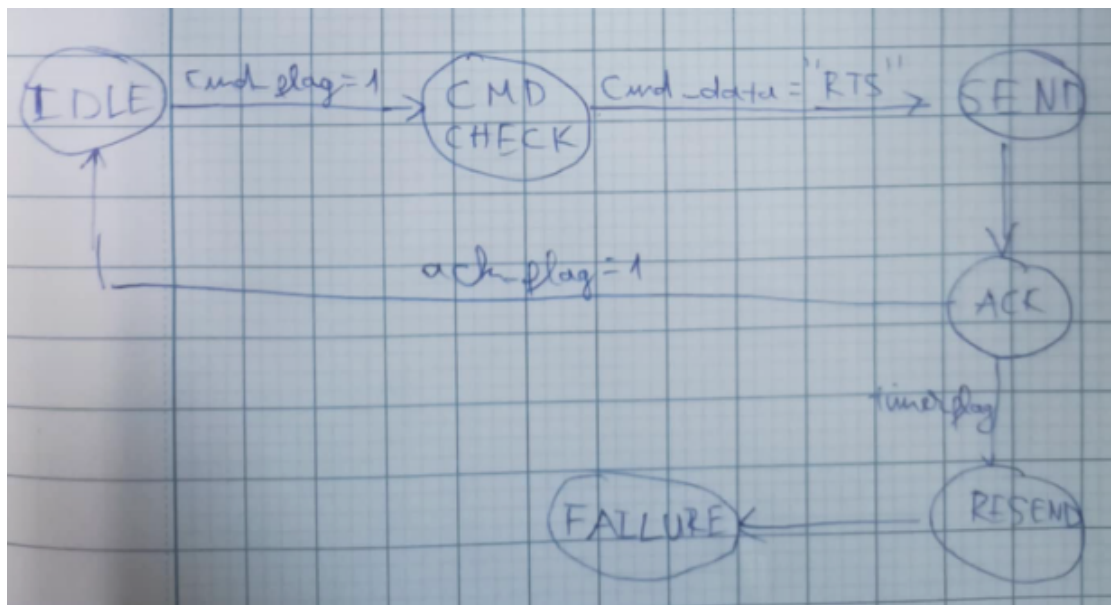
**Figure 2:** *FSM for command parser.*

The diagram represents a finite-state machine (FSM) for parsing commands entered via a virtual terminal through UART. The input characters are stored in a buffer.

- IDLE State:
  - The system waits for input from the virtual terminal.
  - Transitions to the SoC (Start of Command) state when `buffer_flag = 1`, indicating that the data is available in the buffer.
- SoC State (!):

- Detects the start of a command sequence.
- Stores the input character into cmd\_data, if it is not '#' continue store characters.
- Transitions to the EoC (End of Command) state after receiving the '#' character, signaling the end of the command.
- EoC state (#):
  - Processes the command in the buffer. Also sets cmd\_flag on after we get the character '#'.
  - Returns to the IDLE state after processing is complete.

### 1.2.2 FSM for command status



**Figure 3:** FSM for command parser.

The diagram represents a state machine for handling commands received via UART from a virtual terminal. The flow is as follows:

- IDLE State:
  - The system waits for the cmd\_flag to be set, indicating a command is available in the buffer.
  - Transitions to the CMD CHECK state if cmd\_flag = 1.
- CMD CHECK State:
  - Verifies the content of cmd\_data.
  - If the command is valid (e.g., cmd\_data = "RTS"), transitions to the SEND state to execute the request.



- SEND State:
  - Executes the requested action and waits for acknowledgment (`ack_flag = 1`).
  - If acknowledgment is received, transitions to the ACK state.
- ACK State:
  - Indicates successful execution and transitions back to the IDLE state.
- FAILURE State:
  - If no acknowledgment is received within a timeout period, transitions to the RESEND state to retry.
  - May eventually return to IDLE after several retries or report a failure.

## 1.3 UART Communication

### 1.3.1 Receive

The buffer stores each entered character sequentially. When a character is entered, it is placed in the next empty position in the `buffer`, and the program continues receiving characters. When the buffer reaches its maximum size (`MAX_BUFFER_SIZE`) or a complete command (`cmd_flag == 1`) is received, the `index_buffer` is reset to 0 to start receiving characters for a new command. Simultaneously, `buffer_flag` is set to 1 to indicate that a character has been entered.

```
1 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
2     if (huart->Instance == USART2) {
3         buffer[index_buffer++] = temp;
4
5         if (index_buffer == MAX_BUFFER_SIZE || cmd_flag == 1) {
6             //buffer[index_buffer] = '\0';
7             index_buffer = 0;
8         }
9
10        buffer_flag = 1;
11
12        HAL_UART_Receive_IT(&huart2, &temp, 1);
13    }
14 }
```

### 1.3.2 Transmit

When the `cmd_action` function is called, it transmits the value of `ADC_value` (the ADC result) to the virtual terminal using the UART communication protocol. The message is formatted as `!ADC=<value>#`. After sending the data, the function waits for 3 seconds (`HAL_Delay(3000)`) before proceeding to the next action.

```
1 void cmd_action(void){
2     HAL_UART_Transmit(&huart2, (char*)str, sprintf(str, "!ADC=%d#", ADC_value),
3     1000);
4     HAL_Delay(3000);
5 }
```



## 1.4 Command processing

### 1.4.1 Command parser

The `command_parser_fsm` function is a finite state machine (FSM) that processes incoming characters from the buffer. It transitions between two states: SOC (Start of Command) and EOC (End of Command). When the SOC state is active and the received character is `!`, it indicates the start of a command, and the function prepares for command processing. When the EOC state is active and the received character is `#`, it marks the end of the command. The command data is stored in the `cmd_data` array, and a flag `cmd_flag` is set to signal that a command is complete.

```
1 void command_parser_fsm(void) {
2     uint8_t received_char = buffer[index_buffer - 1];
3     switch (current_read_state){
4         case SOC:
5             if (received_char == '!') {
6                 auto_action_flag = 0;
7                 current_read_state = EOC;
8                 cmd_index = 0;
9                 HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
10            }
11            break;
12
13        case EOC:
14            if (received_char == '#'){
15                cmd_data[cmd_index] = '\0';
16                current_read_state = SOC;
17                cmd_flag = 1;
18            }
19            else if (cmd_index < MAX_CMD_LENGTH - 1){
20                cmd_data[cmd_index++] = received_char;
21            }
22            break;
23
24        default:
25            current_read_state = SOC;
26            break;
27    }
28 }
```

### 1.4.2 Command status

The `uart_communication_fsm` function is a finite state machine (FSM) that handles different communication states. It starts in the AUTO state, where it checks if the `cmd_flag` is set to 1, which transitions the FSM to the CMD\_CHECK state. If the `auto_action_flag` is set to 1, the FSM transitions to the SEND state. In the CMD\_CHECK state, the function checks if the `cmd_data`

matches predefined commands (such as "stopSen" or "askSen"). If the "stopSen" command is received, it toggles a red LED and returns to the AUTO state. If "askSen" is received, it triggers an ADC read and transitions to the SEND state. After processing the command, the FSM returns to the AUTO state. The SEND state performs the command action via the `cmd_action` function and then transitions back to AUTO.

```
1 void uart_communication_fsm(void){
2     switch (current_state) {
3     case AUTO:
4         if (cmd_flag == 1) {
5             current_state = CMD_CHECK;
6             cmd_flag = 0;
7         }
8         else if (auto_action_flag == 1){
9             current_state = SEND;
10        }
11        break;
12
13    case CMD_CHECK:
14        if (strcmp((char *)cmd_data, stopSen, 2) == 0) {
15            current_state = AUTO;
16            auto_action_flag = 0;
17            HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
18        }
19        else if (strcmp((char *)cmd_data, askSen, 3) == 0) {
20            read_ADC();
21            current_state = SEND;
22            auto_action_flag = 1;
23        }
24        current_state = AUTO;
25        break;
26
27    case SEND:
28        cmd_action();
29        current_state = AUTO;
30        break;
31
32    default:
33        current_state = AUTO;
34        break;
35    }
36 }
```

## 1.5 Variables and main()

### 1.5.1 Variables

```
1 typedef enum {
```

```
2     IDLE,
3     SOC,
4     EOC
5 } CommandRead;
6 CommandRead current_read_state = IDLE;
7
8 static enum {
9     AUTO,
10    CMD_CHECK,
11    SEND,
12    ACK,
13    FAILURE,
14    RESEND
15 } current_state = AUTO;
16 #define MAX_BUFFER_SIZE 30
17 const char askSen[3] = "RST";
18 const char stopSen[2] = "OK";
19 uint8_t temp = 0;
20 uint8_t buffer[MAX_BUFFER_SIZE];
21 uint8_t index_buffer = 0;
22 uint8_t buffer_flag = 1;
23
24 #define MAX_CMD_LENGTH 10
25
26 uint8_t cmd_data[MAX_CMD_LENGTH];
27 uint8_t cmd_flag = 0;
28 static uint8_t cmd_index = 0;
29
30 #define MAX_RETRIES 3
31 uint8_t auto_action_flag = -1;
```

### 1.5.2 main()

```
1 HAL_UART_RxCpltCallback(&huart2);
2 HAL_TIM_Base_Start_IT(&htim2);
3 uint32_t ADC_value = 0;
4 uint8_t str[20];
5 while (1)
6 {
7     /* USER CODE END WHILE */
8     if (buffer_flag == 1) {
9         auto_action_flag = 0;
10        command_parser_fsm();
11        buffer_flag = 0;
12    }
13    uart_communication_fsm();
14    /* USER CODE BEGIN 3 */
15 }
```



## 2 Problem about this Lab

In this lab, I encountered a problem where the `HAL_Delay(3000)` in the `cmd.action()` function causes a 3-second delay, during which no actions are recognized. As a result, any input entered during this period is not recorded, or it is only recognized after 3 seconds. For example, when entering `!OK#` too fast in 3 seconds, the system will only recognize the `#` character, which prevents the ADC value from stopping as expected.

I thought of a solution by combining it with a timer interrupt, as in previous labs, but due to time constraints, I was unable to complete it.



## References