VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Microprocessor - Microcontroller

Lab Report - CO3010

Lab 5

Advisor(s):	Phan Văn Sỹ	
Student(s):	Chu Le Hoang	2352346

HO CHI MINH CITY, NOVEMBER 2025

Contents

1 Overall

Lab schematics and the source codes are submitted via GitHub link: https://github.com/KennyLe298/MPU-MCU

2 Proteus Schematic

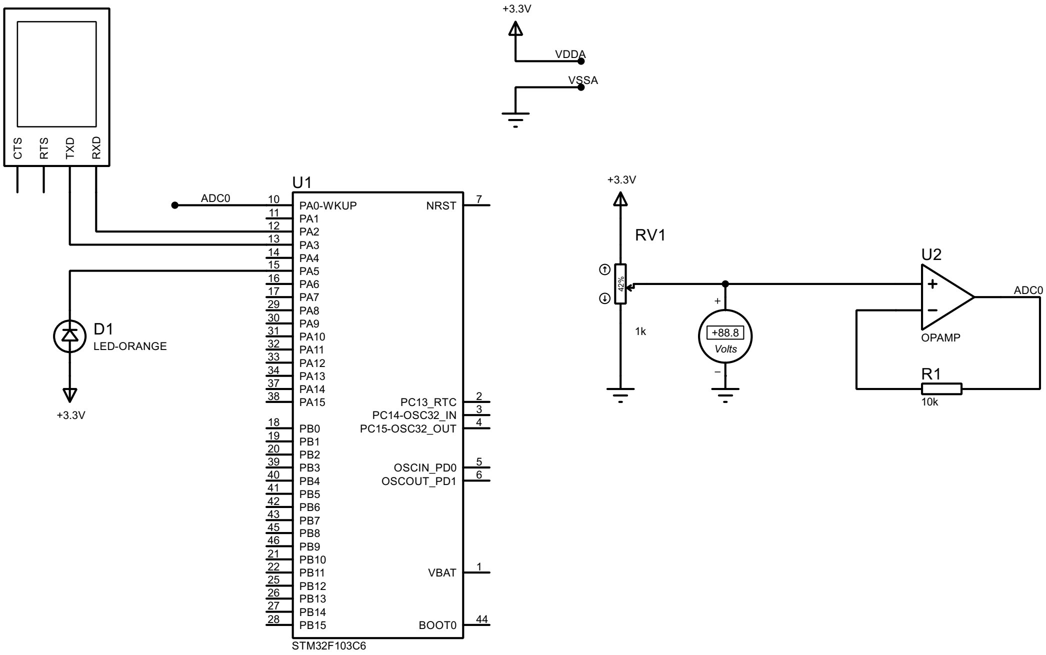

Figure 2.1: Lab 5 schematic

3 Problem

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.

4 Implementation

The project is designed using two independent Finite State Machines (FSMs) as suggested by the lab manual: one for parsing commands and one for managing the communication state.

The reception of data is handled by the HAL_UART_RxCpltCallback interrupt. This ISR places the incoming byte (temp) into a global buffer at buf_idx and sets a buf_flag to 1. The main loop polls this flag.

4.1 Command Parser FSM

The command_parser_fsm() is called in the main loop whenever buf_flag is set. It is responsible for reading the last byte from the buffer (buffer[buf_idx-1]) and processing it.

The FSM waits in an IDLE state. When it receives a !, it moves to the WAIT_CMD state. It then appends all subsequent characters to a temporary command_buffer until it receives a #.

Once the # is received, it compares the command_buffer string against "RST" or "OK". If a match is found, it sets the global command_flag and command_data variables for the main communication FSM to consume.

4.2 UART Communication FSM

The uart_fsm() runs continuously in the main loop and manages the protocol's state. The FSM is defined by the following states:

COMM_IDLE: This is the initial state, waiting for the command_flag to be set to CMD_RST.

COMM_SEND_ADC: When triggered, this state reads the current ADC value, formats it into the string [!ADC=xxxx#], and transmits it via UART. It then saves the current time (last_tick) and transitions to COMM_WAIT_OK.

COMM_WAIT_OK: The FSM waits in this state for the command_flag to be set to CMD_OK.

If CMD_OK is received, it clears the flag and returns to COMM_IDLE

If 3000ms (3 seconds) pass and no CMD_OK is received, the timeout occurs. The FSM re-sends the previous ADC packet (without re-reading the sensor) and restarts the 3-second timer. This satisfies the error control requirement

5 Source code

5.1 command_parser.h

```c
#define MAX_BUFFER_SIZE 30
typedef enum{
  CMD_NONE = 0,
  CMD_RST,
  CMD_OK
} CommandType;
//Variables
extern uint8_t temp;
extern uint8_t buffer[MAX_BUFFER_SIZE];
extern uint8_t buf_idx;
extern uint8_t buf_flag;
extern uint8_t command_flag;
extern CommandType command_data;
void command_parser_init(void);
void command_parser_fsm(void);
```

5.2 command_parser.c

```c
//Variables
```

```
uint8_t temp = 0;
uint8_t buffer[MAX_BUFFER_SIZE];
uint8_t buf_idx = 0;
uint8_t buf_flag = 0;
uint8_t command_flag = 0;
CommandType command_data = CMD_NONE;
typedef enum{
  IDLE = 0,
  WAIT_CMD,
  WAIT_END
} ParserState;
static ParserState parser_state = IDLE;
static uint8_t command_buffer[MAX_BUFFER_SIZE];
static uint8_t cmd_idx = 0;
void command_parser_init(void)
{
  parser_state = IDLE;
  cmd_idx = 0;
  command_flag = 0;
  command_data = CMD_NONE;
  buf_idx = 0;
  buf_flag = 0;
  memset(buffer, 0, MAX_BUFFER_SIZE);
  memset(command_buffer, 0, MAX_BUFFER_SIZE);
}
void command_parser_fsm(void){
  uint8_t cur_idx;
  if(buf_idx == 0) {
    cur_idx = MAX_BUFFER_SIZE - 1;
  } else {
    cur_idx = buf_idx - 1;
  }
  uint8_t cur_char = buffer[cur_idx];
  switch(parser_state){
    case IDLE:
```

```
    if(cur_char == '!'){
      parser_state = WAIT_CMD;
      cmd_idx = 0;
      memset(command_buffer, 0, MAX_BUFFER_SIZE);
    }
    break;
    case WAIT_CMD:
    if(cur_char == '#'){
      command_buffer[cmd_idx] = '\0';
      if(strcmp((char*)command_buffer, "RST") == 0){
        command_data = CMD_RST;
        command_flag = 1;
      }
      else if(strcmp((char*)command_buffer, "OK") == 0){
        command_data = CMD_OK;
        command_flag = 1;
      }
      else{
        command_data = CMD_NONE;
        command_flag = 0;
      }

      parser_state = IDLE;
      cmd_idx = 0;
    }
    else if(cur_char != '!'){
      if(cmd_idx < MAX_BUFFER_SIZE - 1){
        command_buffer[cmd_idx++] = cur_char;
      }
      else {
        parser_state = IDLE;
        cmd_idx = 0;
      }
    }
    break;
    default:
    parser_state = IDLE;
    break;
  }
}
```

5.3 uart.h

```
void uart_init(ADC_HandleTypeDef *hadc, UART_HandleTypeDef *
   huart);
void uart_fsm(void);
```

5.4 uart.c

```
typedef enum{
  COMM_IDLE = 0,
  COMM_WAIT_RST,
  COMM_SEND_ADC,
  COMM_WAIT_OK
} CommState;
static CommState comm_state = COMM_IDLE;
static ADC_HandleTypeDef* adc_handle;
static UART_HandleTypeDef* uart_handle;
static uint32_t adc_value = 0;
static uint32_t timeout_counter = 0;
static uint32_t last_tick = 0;
#define TIMEOUT_MS 3000
#define RESPONSE_FORMAT "!ADC=%lu#\r\n"
void uart_init(ADC_HandleTypeDef *hadc, UART_HandleTypeDef *
   huart) {
  comm_state = COMM_IDLE;
  adc_handle = hadc;
  uart_handle = huart;
  adc_value = 0;
  timeout_counter = 0;
```

```c
  last_tick = HAL_GetTick();
  HAL_ADC_Start(adc_handle);
}
static void send_adc_response(void){
  char response[50];
  int len = sprintf(response, RESPONSE_FORMAT, adc_value);
  HAL_UART_Transmit(uart_handle, (uint8_t*)response, len, 1000);
}
void uart_fsm(void){
  uint32_t cur_tick = HAL_GetTick();
  switch(comm_state){
    case COMM_IDLE:
    if(command_flag == 1 && command_data == CMD_RST){
      command_flag = 0;
      comm_state = COMM_SEND_ADC;
    }
    break;
    case COMM_SEND_ADC:
    adc_value = HAL_ADC_GetValue(adc_handle);
    send_adc_response();
    last_tick = HAL_GetTick();
    comm_state = COMM_WAIT_OK;
    break;
    case COMM_WAIT_OK:
    if(command_flag == 1 && command_data == CMD_OK){
      command_flag = 0;
      comm_state = COMM_IDLE;
    }
    else if((cur_tick - last_tick) >= TIMEOUT_MS){
      send_adc_response();
      last_tick = HAL_GetTick();
    }
    else if(command_flag == 1 && command_data == CMD_RST){
      command_flag = 0;
      comm_state = COMM_SEND_ADC;
    }
    break;
    default:
    comm_state = COMM_IDLE;
    break;
  }
}
```

5.5 main.c

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  if(huart->Instance == USART2){
    buffer[buf_idx++] = temp;
    if(buf_idx >= MAX_BUFFER_SIZE){
      buf_idx = 0;
    }
    buf_flag = 1;
    HAL_UART_Receive_IT(&huart2, &temp, 1);
  }
}

command_parser_init();
uart_init(&hadc1, &huart2);
HAL_UART_Receive_IT(&huart2, &temp, 1);
while (1)
{
  if(buf_flag == 1){
    command_parser_fsm();
    buf_flag = 0;
  }
  uart_fsm();
}
```

6 Summary

The demo of this project will be presented onsite and graded by the lecturer.