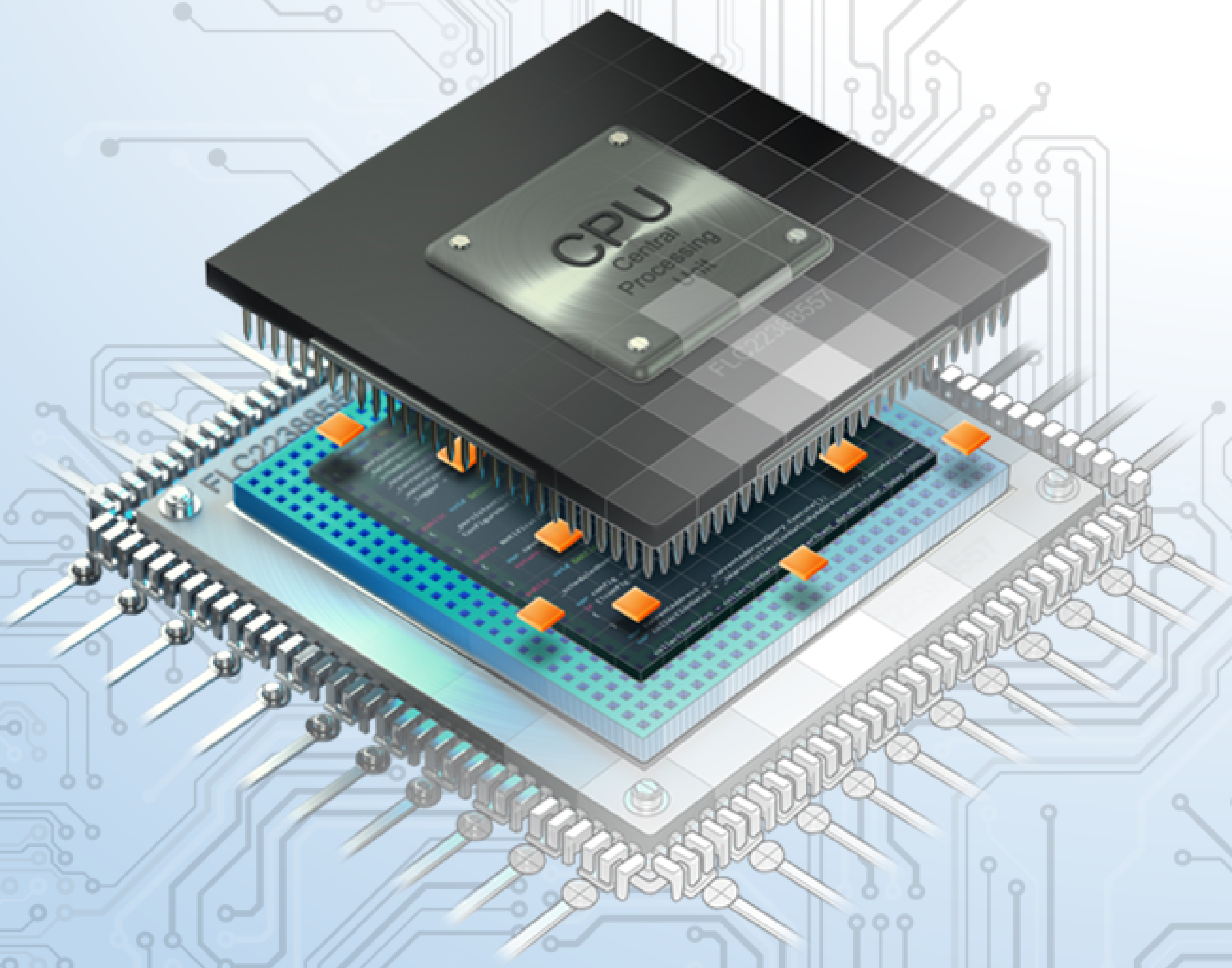
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller



Dr. Le Trong Nhan

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

REPORT
MICROCONTROLLER (CO3010)

Lab 5

Lecturer: Le Trong Nhan
Huynh Phuc Nghi
Class: L05
Github: https://github.com/NCT2311/VXL.git

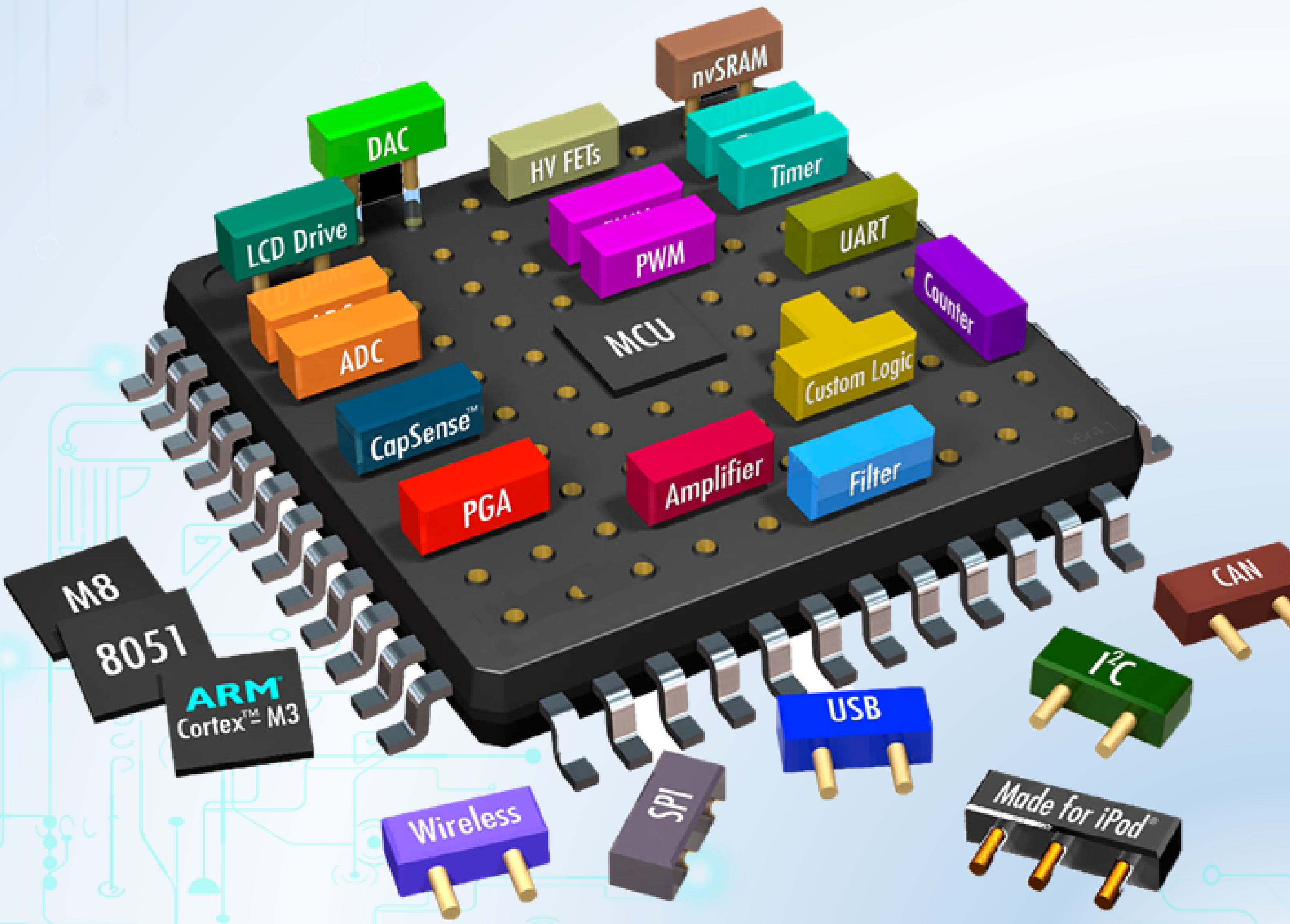
Name	**StudentID**
Nguyen Cong Thanh	1915144

Ho Chi Minh City, December 2021

Mục lục

CHƯƠNG 1

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.

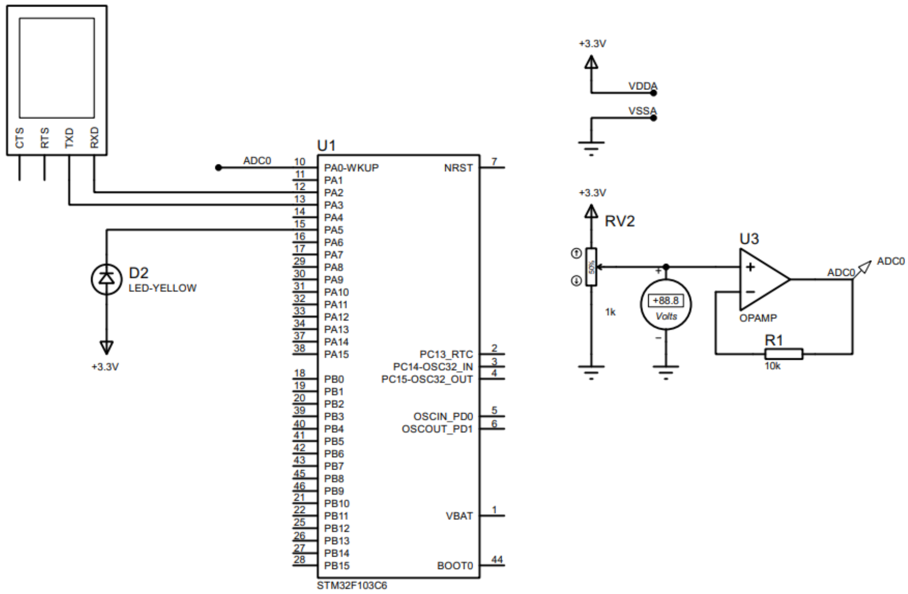
A critical issue, but not really frequently occurred, in the flow control is that the processing rate is slower than the transmission rate. Due to this reason each receiving device has a block of memory that is commonly known as buffer, that is used to store the incoming data until this data will be 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



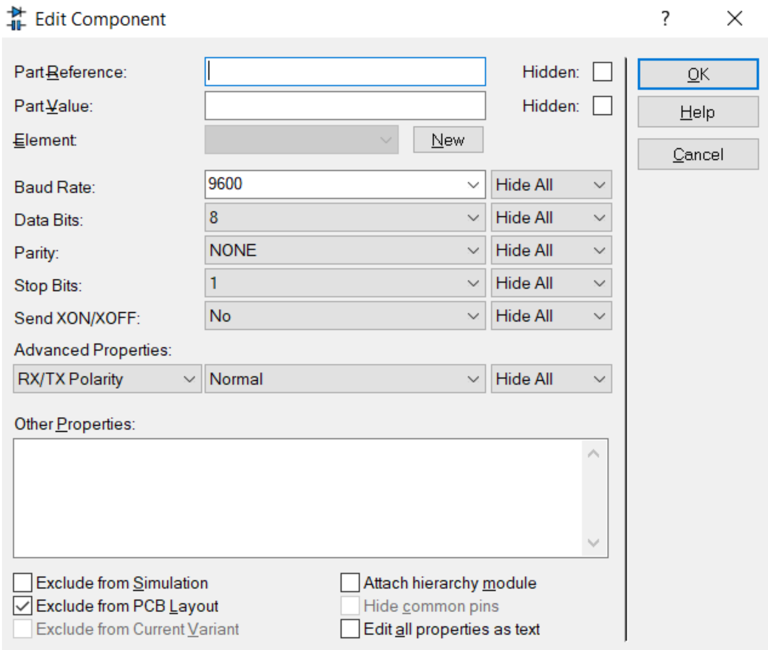
Hình 1.1: *Simulation circuit on Proteus*

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:



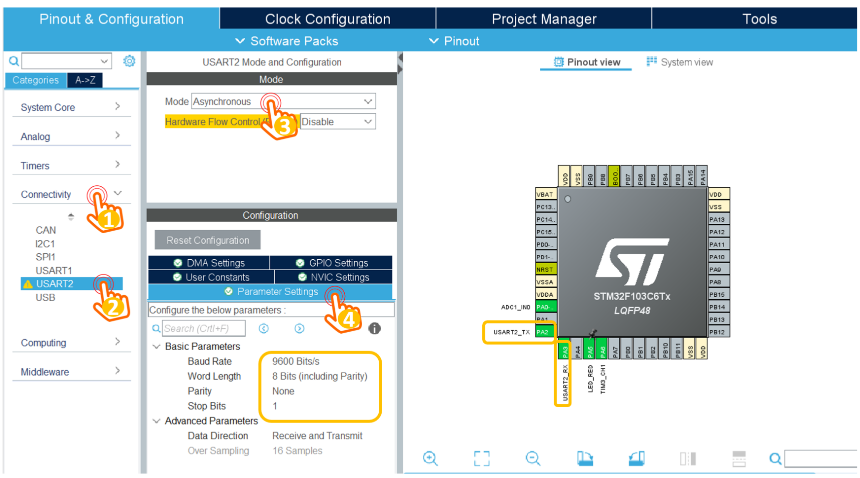
Hình 1.2: Terminal configuration

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

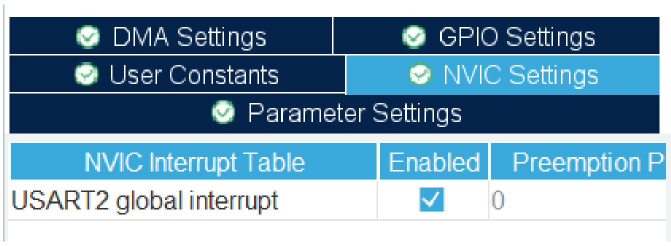
From the ioc file, select **Connectivity**, and then select the **USART2**. The parameter settings for UART channel 2 (USART2) module are depicted as follows:



Hình 1.3: 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.

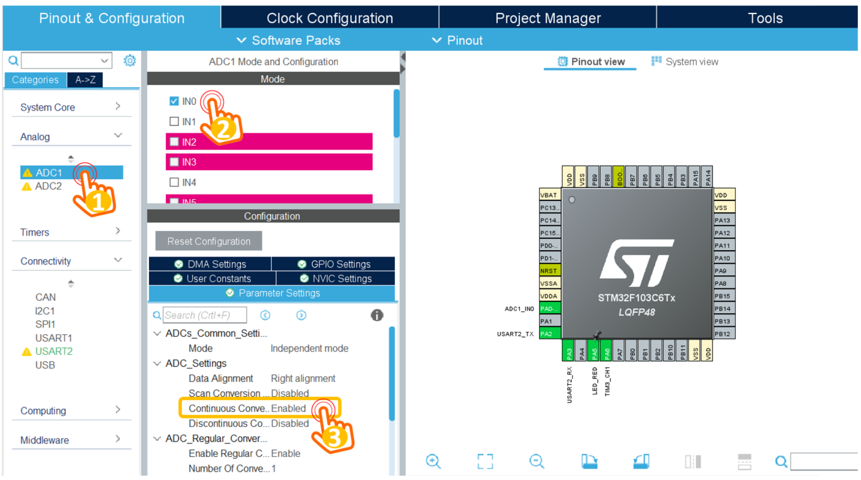
Finally, the NVIC settings are checked to enable the UART interrupt, as follows:



Hình 1.4: 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 1.5: 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.

```
/* USER CODE BEGIN 0 */
uint8_t temp = 0;

```

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
  if(huart->Instance == USART2){
    HAL_UART_Transmit(&huart2, &temp, 1, 50);
    HAL_UART_Receive_IT(&huart2, &temp, 1);
  }
}
/* USER CODE END 0 */
```

Program 1.1: 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:

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();

  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_ADC1_Init();

  HAL_UART_Receive_IT(&huart2, &temp, 1);

  while (1)
  {
    HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
    HAL_Delay(500);
  }

}
```

Program 1.2: Implement the main function

5 Sensor reading

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

```
uint32_t ADC_value = 0;
while (1)
{
  HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
  ADC_value = HAL_ADC_GetValue(&hadc1);
HAL_UART_Transmit(&huart2, (void *)str, sprintf(str, "%d\n"
    , ADC_value), 1000);
  HAL_Delay(500);
}
```

Program 1.3: 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.

```
#define MAX_BUFFER_SIZE  30
uint8_t temp = 0;
uint8_t buffer[MAX_BUFFER_SIZE];
uint8_t index_buffer = 0;
uint8_t buffer_flag = 0;
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
  if(huart->Instance == USART2){

    //HAL_UART_Transmit(&huart2, &temp, 1, 50);
    buffer[index_buffer++] = temp;
    if(index_buffer == 30) index_buffer = 0;

    buffer_flag = 1;
    HAL_UART_Receive_IT(&huart2, &temp, 1);
  }
}
```

Program 1.4: 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:

```c
while (1){
    if(buffer_flag == 1){
        command_parser_fsm();
        buffer_flag = 0;
    }
}
```

Program 1.5: State machine to extract the command

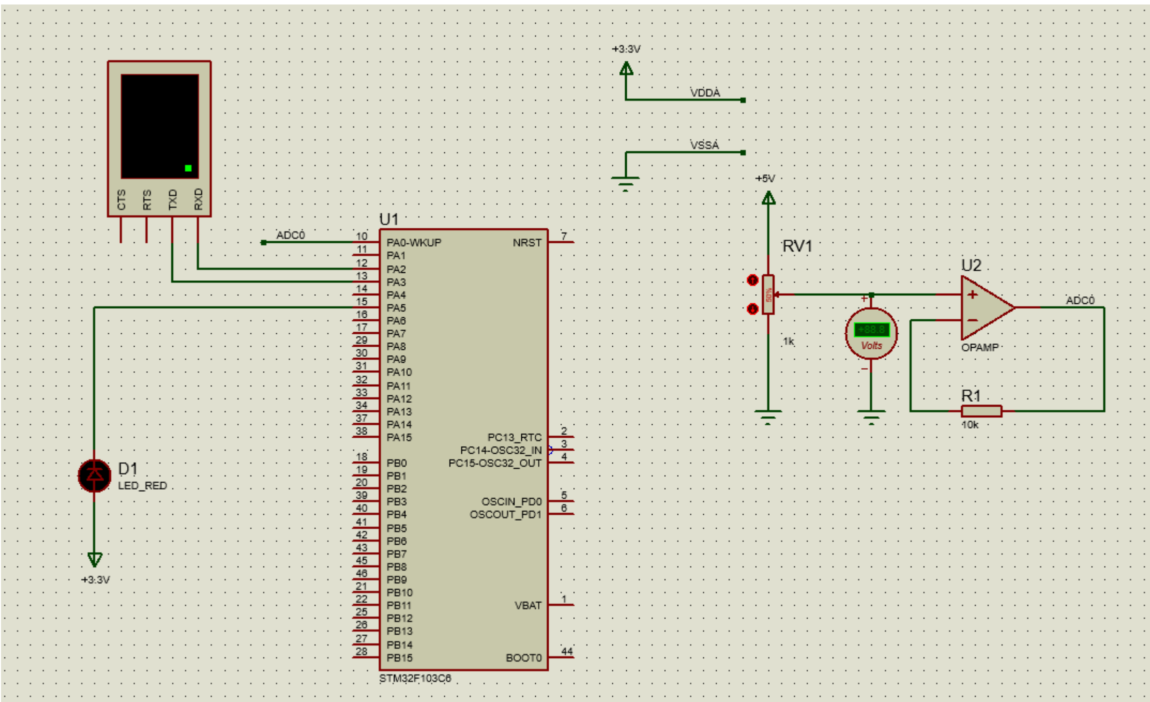
The output of the command parser is to set **command_flag** and **command_data**. In this project, there are two commands, **RTS** and **OK**. The program skeleton is proposed as follows:

```c
while (1){
    if(buffer_flag == 1){
        command_parser_fsm();
        buffer_flag = 0;
    }
    uart_communiation_fsm();
}
```

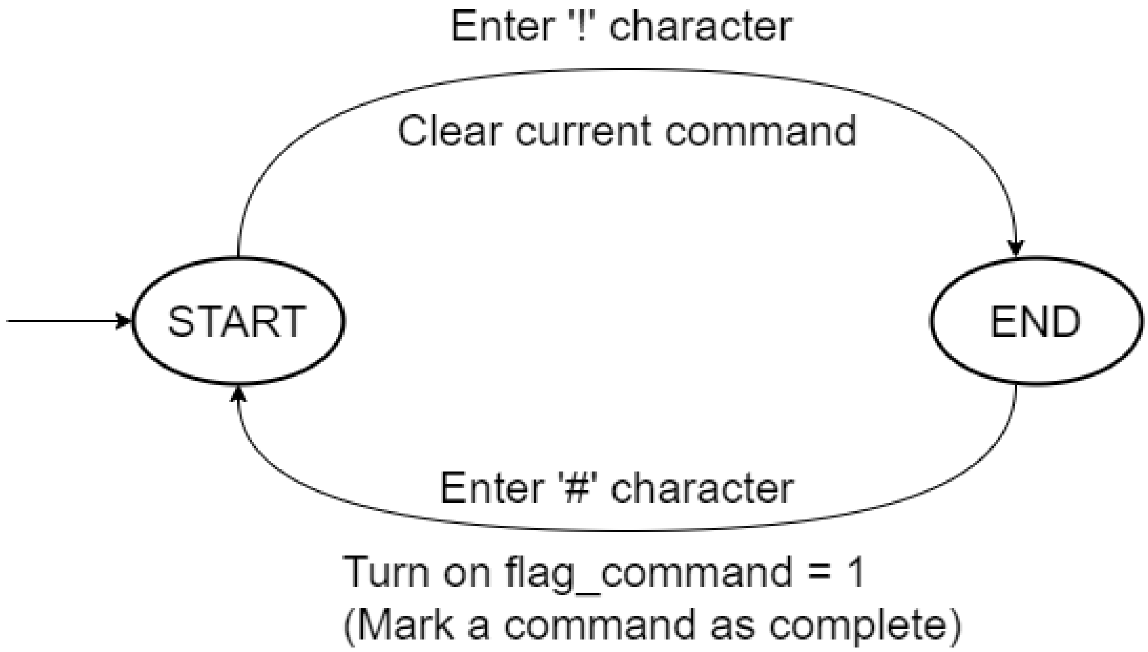
Program 1.6: Program structure

6.2 Project implementation

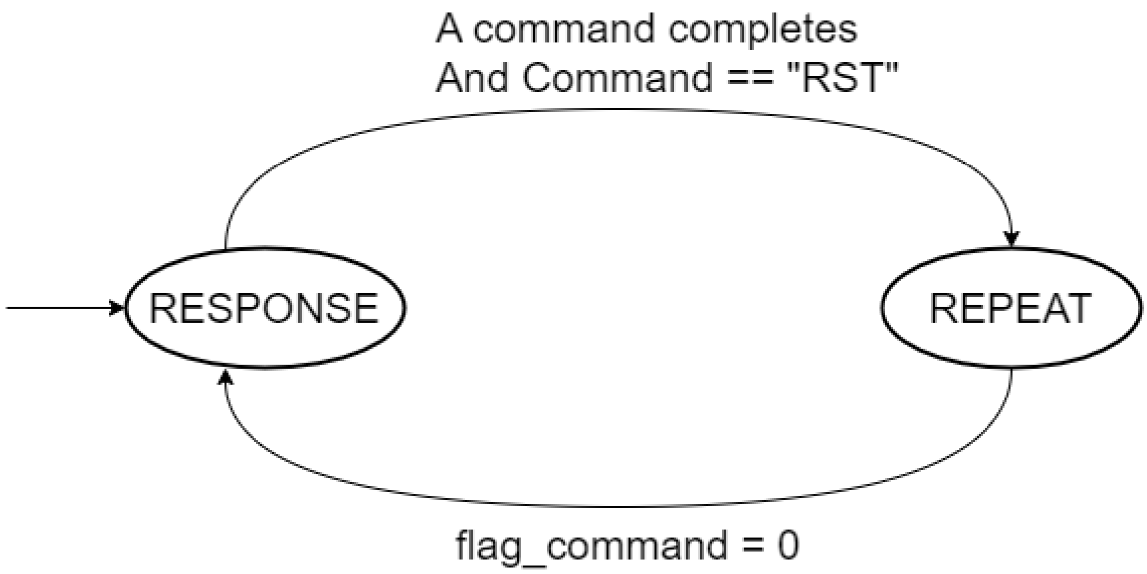
Students are proposed to implement 2 FSM in seperated modules. Students are asked to design the FSM before their implementations in STM32Cube.



Hình 1.6: Schema of Lab 5



Hình 1.7: Command Parser FSM



Hình 1.8: Uart Communication FSM

We set timer interrupt is 10ms.

```
#ifndef INC_TIMER_H_
#define INC_TIMER_H_
void setTimer0(int duration);
void setTimer1(int duration);
void timer_run();
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
    ;
int timer0_counter;
int timer0_flag;
int TIMER_CYCLE;
int timer1_counter;
int timer1_flag;
#endif /* INC_TIMER_H_ */
```

Program 1.7: Source code of timer.h

```
#include "main.h"
#include "timer.h"
TIMER_CYCLE = 10;
void setTimer0(int duration){
  timer0_counter = duration / TIMER_CYCLE;
  timer0_flag = 0;
}

void setTimer1(int duration){
  timer1_counter = duration / TIMER_CYCLE;
  timer1_flag = 0;
}

void timer_run(){
  if(timer0_counter > 0){
    timer0_counter --;
    if(timer0_counter == 0) timer0_flag = 1;
  }
  if(timer1_counter > 0){
    timer1_counter --;
    if(timer1_counter == 0) timer1_flag = 1;
  }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  timer_run();
}
```

Program 1.8: Source code of timer.c

```
#include <stdint.h>
#ifndef INC_COMMAND_PARSER_H_
#define INC_COMMAND_PARSER_H_
```

```c

void command_parser_fsm();
unsigned char get_flag();
char* get_command();
void clear_command();

extern uint8_t temp;

#endif /* INC_COMMAND_PARSER_H_ */
```

Program 1.9: Source code of command_parser.h

```c
#include "command_parser.h"
#include "main.h"
#include <string.h>
UART_HandleTypeDef huart2;

#define MAX_COMMAND_SIZE 30
uint8_t index_command = 0;
uint8_t command[MAX_COMMAND_SIZE]; //Declare a variable to
   store the input string with max size = 30
unsigned char flag_command = 0;
enum state_command {START, END};
enum state_command currentState = START;

void command_parser_fsm(){
  switch (currentState){
  case START:
    if(temp == '!')
    {
      currentState = END;
      index_command = 0;
      command[0] = "\0"; //Reset command
    }
    break;
  case END:
    if(temp == '#'){
      currentState = START;
      command[index_command] = '\0';
      flag_command = 1; //Mark a command as complete
    } else{
      command[index_command++] = temp; //Store input string
    in command[]
      if(index_command >= MAX_COMMAND_SIZE) index_command =
    0;
      flag_command = 0;
    }
    break;
  default:
    break;
```

```c
  }
}

unsigned char get_flag(){
  return flag_command;
}

char* get_command(){
  return command;
}

void clear_command(){
  command[0] = "\0";
}
```

Program 1.10: Source code of command_parser.c

```c
#ifndef INC_UART_COMMUNICATION_H_
#define INC_UART_COMMUNICATION_H_

void uart_communication_fsm();

#endif /* INC_UART_COMMUNICATION_H_ */
```

Program 1.11: Source code of uart_communication.h

```c
#include "uart_communication.h"
#include "main.h"
#include "command_parser.h"
#include "timer.h"
ADC_HandleTypeDef hadc1;
UART_HandleTypeDef huart2;
uint32_t ADC_value = 0;
char str[20];
enum state_uart {RESPONSE, REPEAT};
enum state_uart uartState = RESPONSE;
void uart_communication_fsm(){
  switch (uartState){
  case RESPONSE:
    {
      if((get_flag() == 1) && (strcmp(get_command(), "RST")
 == 0)){
        ADC_value = HAL_ADC_GetValue(&hadc1); //Get value
 of ADC_value variable
        HAL_UART_Transmit(&huart2, (void*)str, sprintf (str
, "\r\n!ADC=%d#\r\n", ADC_value), 1000); //Print
 response
        uartState = REPEAT;
        setTimer1(3000);
      }
```

```
      if((get_flag() == 1) && (strcmp(get_command(), "OK") == 0)){
        HAL_UART_Transmit(&huart2, str, sprintf(str, "\r\nEnd the communication\r\n"), 1000);
        clear_command();
      }
      break;
    }
  case REPEAT:
    {
      if((timer1_flag == 1) && (get_flag() == 1) && (strcmp(get_command(), "RST") == 0)){
        ADC_value = HAL_ADC_GetValue(&hadc1);
        HAL_UART_Transmit(&huart2, ( void*)str, sprintf (str, "!ADC=%d#\r\n", ADC_value), 1000);
        setTimer1(3000); //Repeat transmit !ADC=ADC_value# every 3 seconds
      } else if(get_flag() == 0){ //When we enter a new character => Stop loop and wait to a new response
        uartState = RESPONSE;
      }
    }
  default:
    break;
  }
}
```

Program 1.12: Source code of uart_communication.c

Link video demo: https://youtu.be/Z0Ne$_{o}lV2_{Y}$