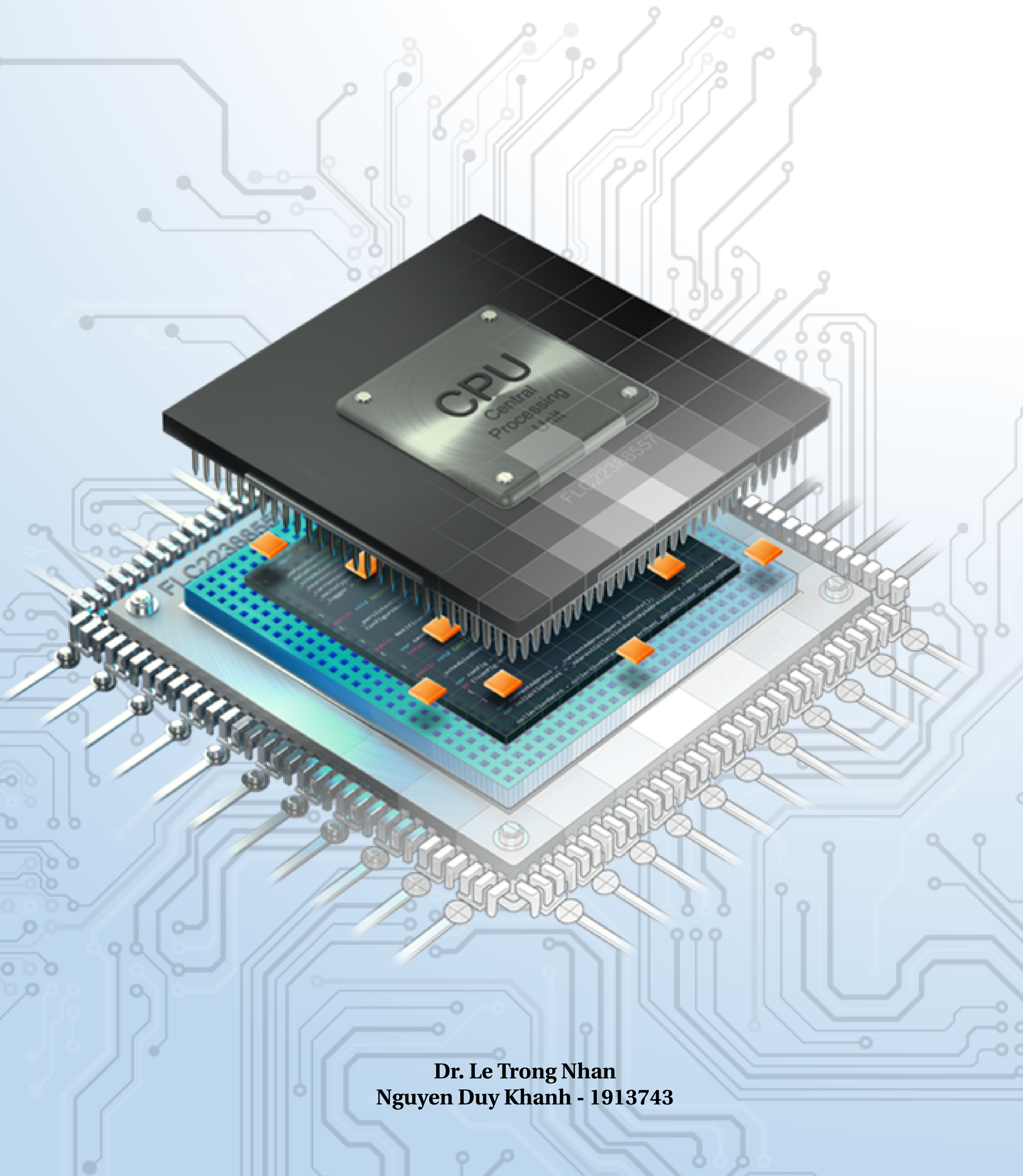
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller

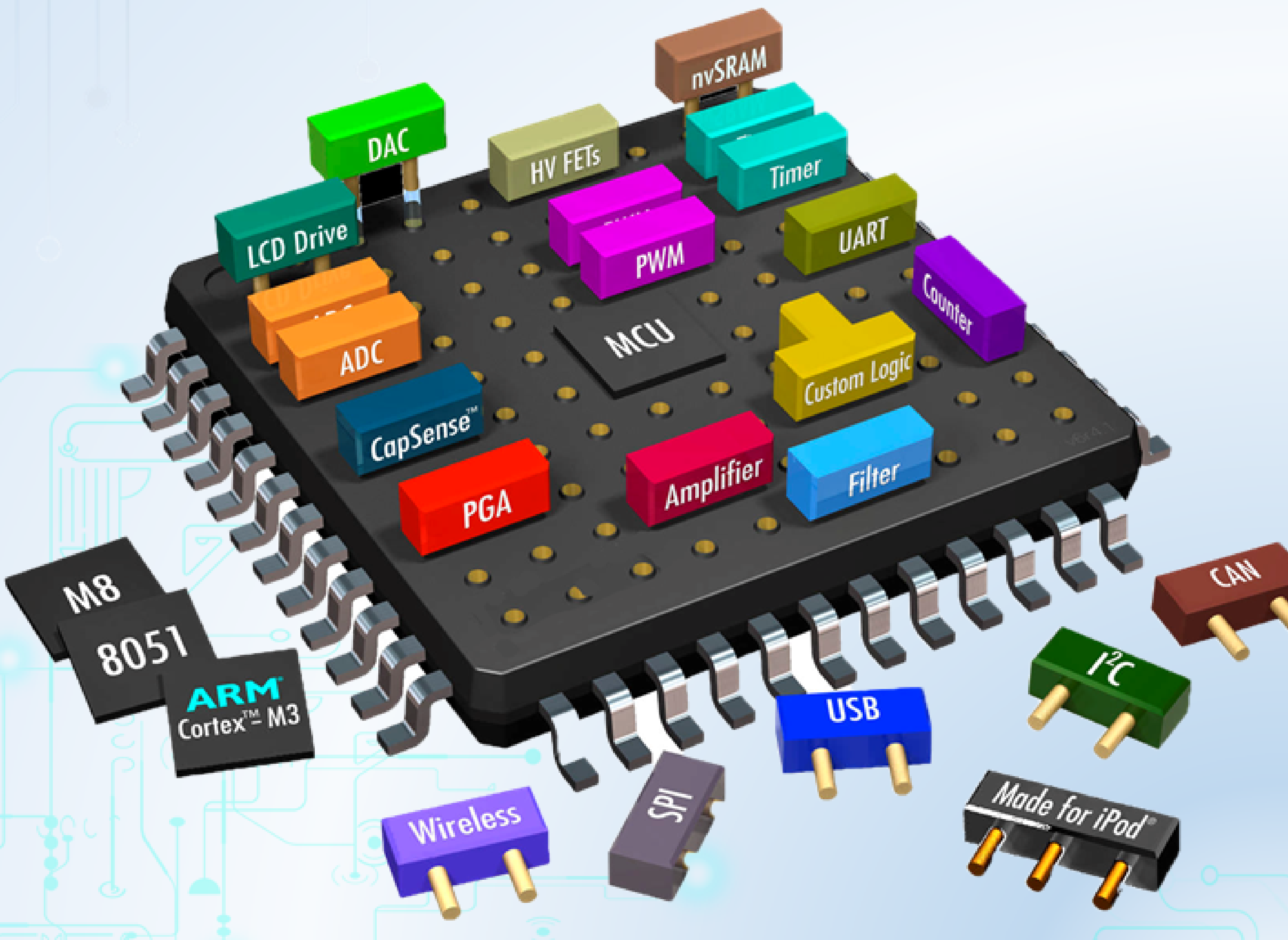


Dr. Le Trong Nhan
Nguyen Duy Khanh - 1913743

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.

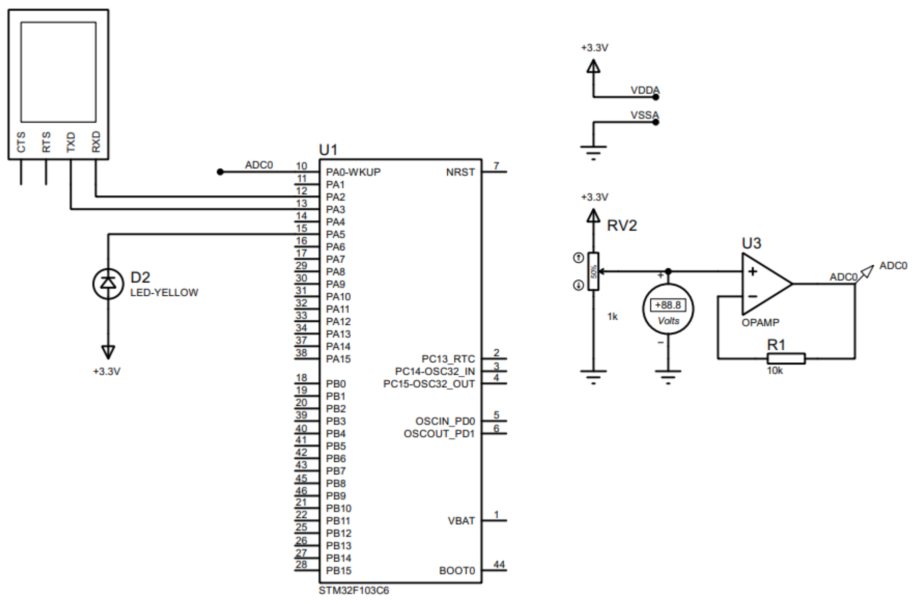
A critical issue, but not really frequently occurred, in the flow control is that the processing rate is slower than the transmission rate. Due to this reason each receiving device has a block of memory that is commonly known as buffer, that is used to store the incoming data until this data will be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



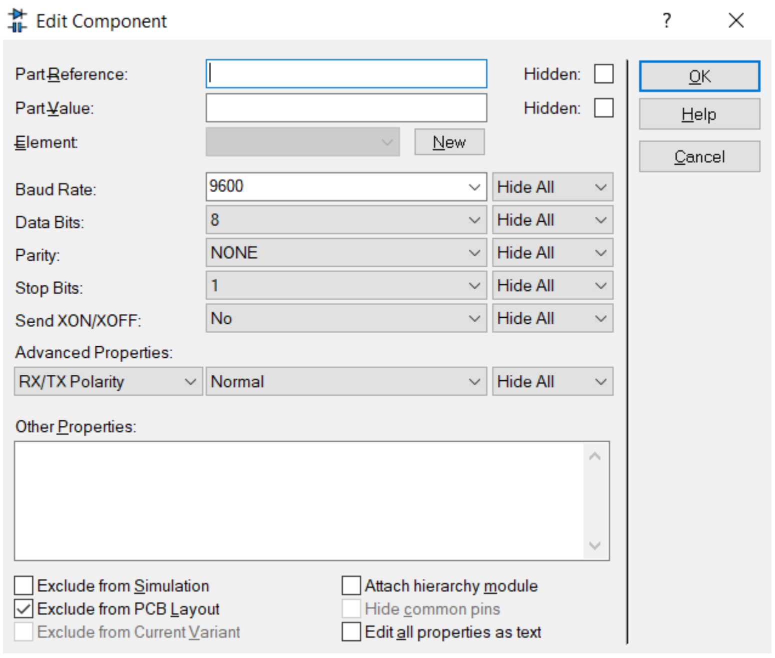
Hình 1.1: *Simulation circuit on Proteus*

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:



Edit Component

Part Reference:		Hidden: ☐	
Part Value:		Hidden: ☐	
Element:		New	
Baud Rate:	9600	Hide All	
Data Bits:	8	Hide All	
Parity:	NONE	Hide All	
Stop Bits:	1	Hide All	
Send XON/XOFF:	No	Hide All	
Advanced Properties:			
RX/TX Polarity	Normal	Hide All	

Other Properties:

☐ Exclude from Simulation ☐ Attach hierarchy module
☑ Exclude from PCB Layout ☐ Hide common pins
☐ Exclude from Current Variant ☐ Edit all properties as text

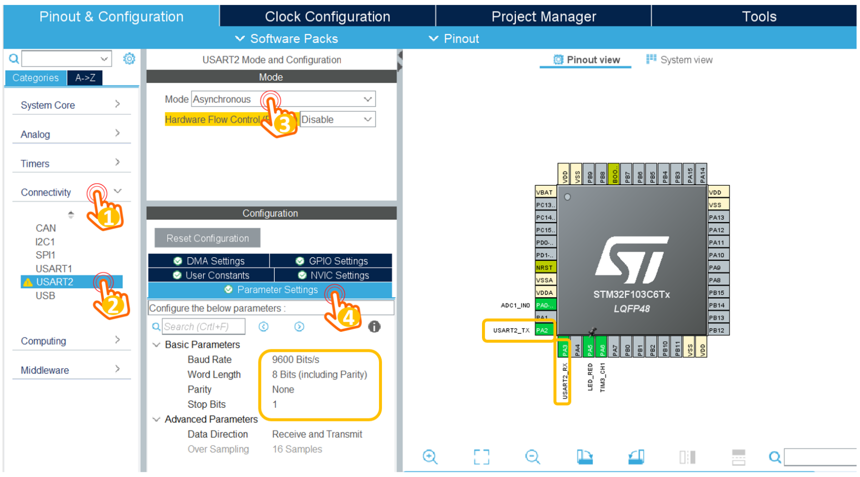
Hình 1.2: Terminal configuration

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:



USART2 Mode and Configuration

Mode: Asynchronous
Hardware Flow Control (RS232): Disable

Configure the below parameters :

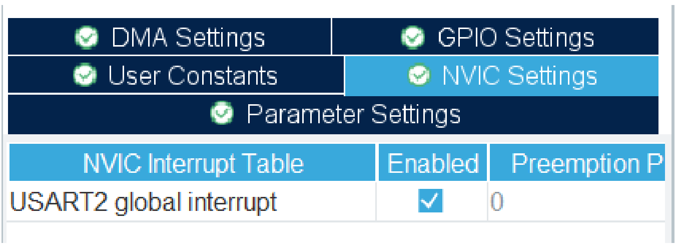
Basic Parameters
- Baud Rate: 9600 Bits/s
- Word Length: 8 Bits (including Parity)
- Parity: None
- Stop Bits: 1

Advanced Parameters
- Data Direction: Receive and Transmit
- Over Sampling: 16 Samples

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:

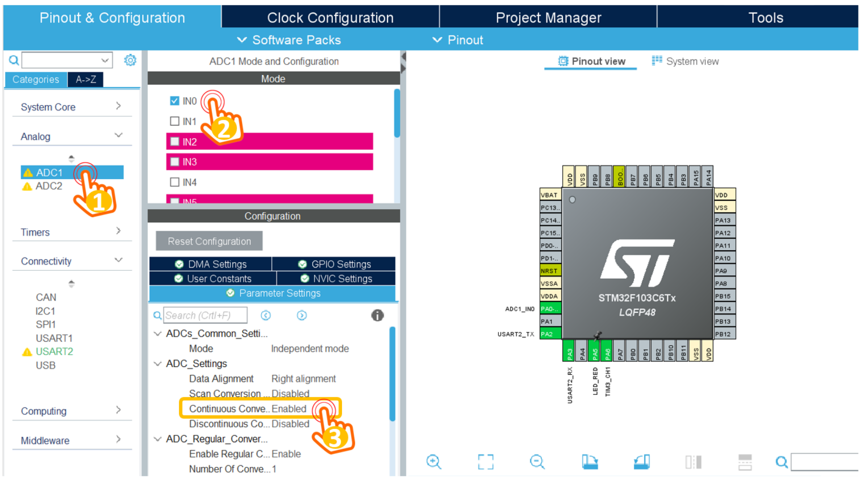

DMA Settings | GPIO Settings | User Constants | NVIC Settings | Parameter Settings

NVIC Interrupt Table	Enabled	Preemption P
USART2 global interrupt	✓	0

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:



ADCs_Common_Setti...
Mode Independent mode
ADC_Settings
Data Alignment Right alignment
Scan Conversion ... Disabled
Continuous Conve... Enabled
Discontinuous Co... Disabled
ADC_Regular_Conver...
Enable Regular C... Enable
Number Of Conve... 1

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){
```

```c
  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:

```c
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:

```c
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.

```c
#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();
}
```

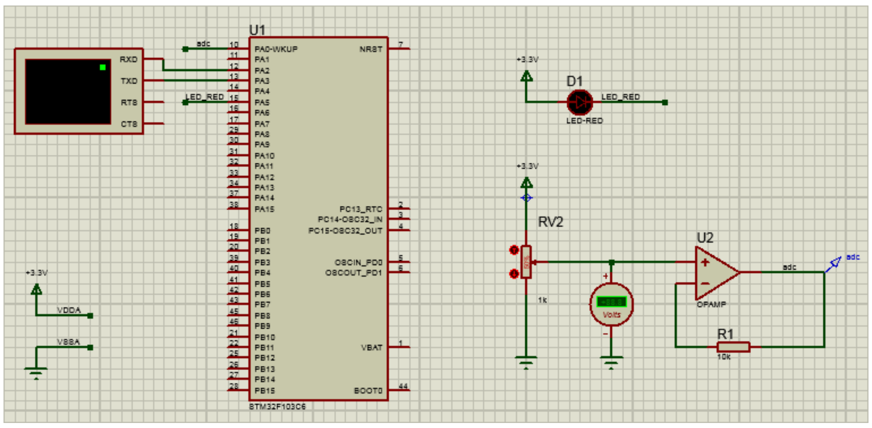
Program 1.6: Program structure

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.

7 Report

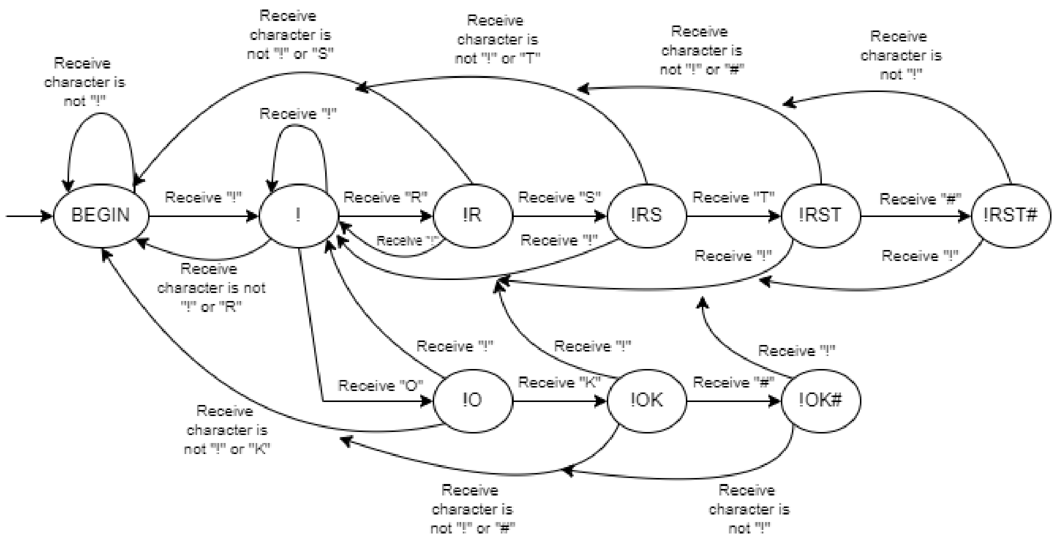
7.1 Proteus schematic



Hình 1.6: Schemation in Proteus

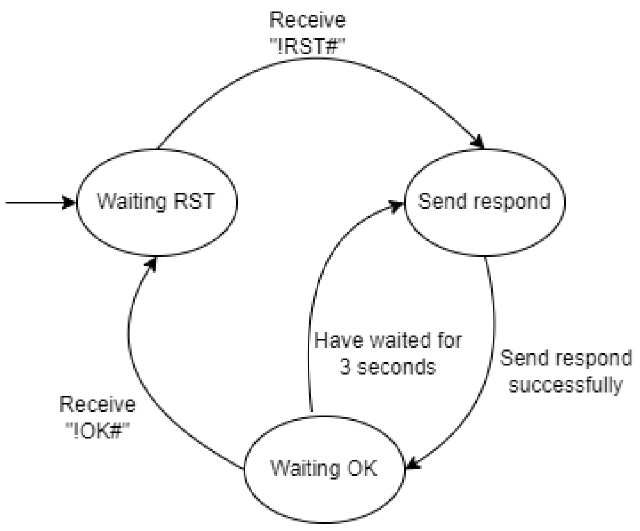
7.2 Finite state machine

1. Command parse


BEGIN
!
!R
!RS
!RST
!RST#
!O
!OK
!OK#
Receive character is not "!"
Receive character is not "!" or "S"
Receive character is not "!" or "T"
Receive character is not "!" or "#"
Receive character is not "!"
Receive "!"
Receive "!"
Receive "R"
Receive "S"
Receive "T"
Receive "#"
Receive "!"
Receive "!"
Receive "!"
Receive "!"
Receive character is not "!" or "R"
Receive "!"
Receive "!"
Receive "!"
Receive "O"
Receive "K"
Receive "#"
Receive character is not "!" or "K"
Receive character is not "!" or "#"
Receive character is not "!"

Hình 1.7: *FSM command parse*

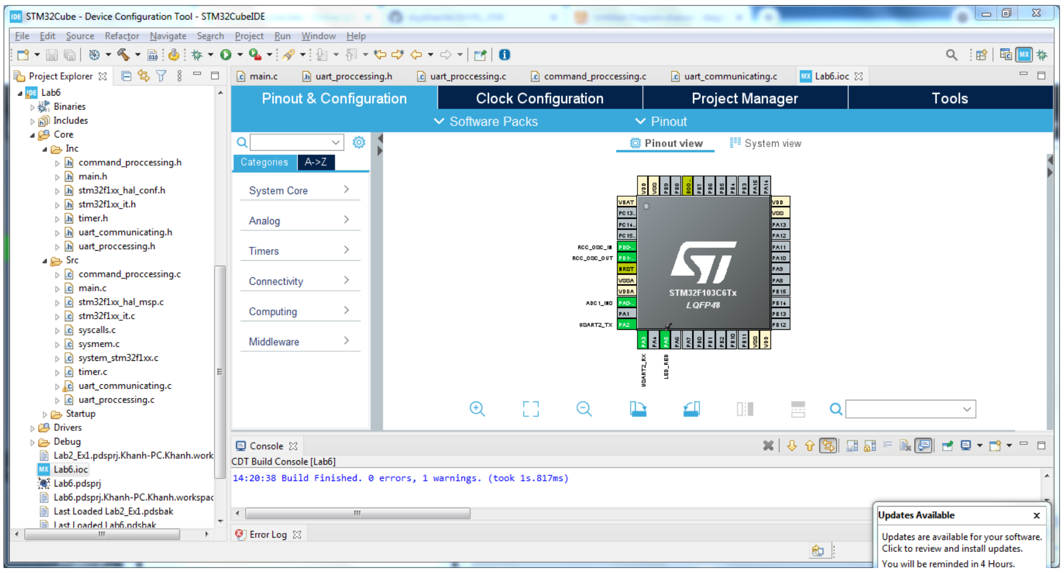
2. UART communicate



Hình 1.8: *FSM uart communicate*

7.3 STM32 project

7.3.1 Project organization



CDT Build Console [Lab6]

```
14:20:38 Build Finished. 0 errors, 1 warnings. (took 1s.817ms)
```

Updates Available

Updates are available for your software. Click to review and install updates.

You will be reminded in 4 Hours.

Hình 1.9: Project organization

7.3.2 *main* function

```c
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_ADC1_Init();
  MX_USART2_UART_Init();
  HAL_ADC_Start(&hadc1);
  HAL_TIM_Base_Start_IT(&htim2);
  // Init timer
  setTimer0(10);
  setTimer1(10);
  HAL_UART_Receive_IT(&huart2, &temp, 1);
  while (1)
  {
    if(getBufferFlag() == 1){
      command_parse_fsm();
      //Set buffer_flag = 0
      resetBufferFlag();
    }
    uart_communication_fsm();
    // For debugging
    if(getTimer1Flag() == 1){
      HAL_GPIO_TogglePin(GPIOA, LED_RED_Pin);
```

```
        setTimer1(1000);
    }

  }
}
```

Program 1.7: Source code in main function

7.3.3 Module timer

Chứa các hàm cho software timer và hàm timer interrupt.

1. *timer.h*

```
#include "main.h"

#ifndef INC_TIMER_H_
#define INC_TIMER_H_

void setTimer0(int);
void setTimer1(int);
void timer_run(void);
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *)
    ;
int getTimer0Flag(void);
int getTimer1Flag(void);
void stopTimer0();

#endif
```

Program 1.8: Source code in file timer.h

2. *timer.c*

```
#include "timer.h"
#include "main.h"

#define TIMER_CYCLE 10

static int timer0_counter = 0;
static int timer1_counter = 0;
static int timer0_flat = 0;
static int timer1_flat = 0;

void setTimer0(int duration){
  timer0_counter = duration / TIMER_CYCLE;
  timer0_flat = 0;
}
void setTimer1(int duration){
  timer1_counter = duration / TIMER_CYCLE;
  timer1_flat = 0;
}
```

```c
void timer_run(){
  if(timer0_counter > 0){
    timer0_counter--;
    if(timer0_counter == 0) timer0_flat = 1;

  }
  if(timer1_counter > 0){
    timer1_counter--;
    if(timer1_counter == 0) timer1_flat = 1;
  }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *
   htim){
  if(htim->Instance == TIM2){
    timer_run();
  }
}

int getTimer0Flag(){
  return timer0_flat;
}

int getTimer1Flag(){
  return timer1_flat;
}

void stopTimer0(){
  timer0_counter = 0;
  timer0_flat = 0;
}
```

Program 1.9: Source code in file timer.c

7.3.4 Module command parse

Phân tích lệnh nhận được từ terminal.

1. *command_processing.h*

```c
#include "main.h"

#ifndef INC_COMMAND_PROCCESSING_H_
#define INC_COMMAND_PROCCESSING_H_

// Finite state machine for command processing
void command_parse_fsm();
// Get state which received "!RST#"
uint8_t isCommandRST();
// Get state which received "!OK#"
uint8_t isCommandOK();
```

```

#endif
```

Program 1.10: Source code in file command_processing.h

2. *command_processing.c*

```
#include "command_proccessing.h"
#include "uart_proccessing.h"
#include "main.h"

enum CMD{
  BEGIN,
  EXCLAMATION_POINT,
  R_CHARACTER,
  S_CHARACTER,
  T_CHARACTER,
  O_CHARACTER,
  K_CHARACTER,
  RECEIVED_RST,
  RECEIVED_OK
};

enum CMD command_state = BEGIN;

void command_parse_fsm(){
  switch (command_state) {
    case BEGIN:
      if(getChar() == 33){
        command_state = EXCLAMATION_POINT;
      }
      break;
    case EXCLAMATION_POINT:
      if(getChar() == 33){
        command_state = EXCLAMATION_POINT;
      }
      else if(getChar() == 82){
        command_state = R_CHARACTER;
      }
      else if(getChar() == 79){
        command_state = O_CHARACTER;
      }
      else{
        command_state = BEGIN;
      }
      break;
    case R_CHARACTER:
      if(getChar() == 33){
        command_state = EXCLAMATION_POINT;
      }
      else if(getChar() == 83){
        command_state = S_CHARACTER;
      }
      else{
        command_state = BEGIN;
      }
      break;
    case S_CHARACTER:
      if(getChar() == 33){
        command_state = EXCLAMATION_POINT;
      }
      else if(getChar() == 84){
        command_state = T_CHARACTER;
      }
      else{
        command_state = BEGIN;
      }
      break;
    case T_CHARACTER:
      if(getChar() == 33){
        command_state = EXCLAMATION_POINT;
      }
      else if(getChar() == 35){
        command_state = RECEIVED_RST;
      }
      else{
        command_state = BEGIN;
      }
      break;
    case O_CHARACTER:
      if(getChar() == 33){
        command_state = EXCLAMATION_POINT;
      }
      else if(getChar() == 75){
        command_state = K_CHARACTER;
      }
      else{
        command_state = BEGIN;
      }
      break;
    case K_CHARACTER:
      if(getChar() == 33){
        command_state = EXCLAMATION_POINT;
      }
      else if(getChar() == 35){
        command_state = RECEIVED_OK;
      }
      else{
        command_state = BEGIN;
      }
```

```c
      break;
    case RECEIVED_RST:
      if(getChar() == 33){
        command_state = EXCLAMATION_POINT;
      }
      else command_state = BEGIN;
      break;
    case RECEIVED_OK:
      if(getChar() == 33){
        command_state = EXCLAMATION_POINT;
      }
      else command_state = BEGIN;
      break;
    default:
      break;
  }
}

uint8_t isCommandRST(){
  if(command_state == RECEIVED_RST) return 1;
  return 0;
}

uint8_t isCommandOK(){
  if(command_state == RECEIVED_OK) return 1;
  return 0;
}
```
Program 1.11: Source code in file command_processing.c

7.3.5 Module uart communication

Dựa vào lệnh nhận được để gửi các phản hồi.

1. *uart_communicating.h*

```c
#ifndef INC_UART_COMMUNICATING_H_
#define INC_UART_COMMUNICATING_H_

//Finite state machine for uart communication
void uart_communication_fsm();

#endif
```
Program 1.12: Source code in file uart_communicating.h

2. *uart_communicating.c*

```c
#include "uart_communicating.h"
#include "command_proccessing.h"
#include "timer.h"
#include "main.h"
```

```c
#include <stdio.h>

#define PERIOD_FOR_RESEND 3000

enum UART{
  WAITTING_RST,
  WAITTING_OK,
  SEND_REP
};

// Array use for printing response
char buffer_uart[20];
enum UART uart_state = WAITTING_RST;

uint32_t adc_value = 0;

extern UART_HandleTypeDef huart2;
extern ADC_HandleTypeDef hadc1;

void uart_communication_fsm(){
  switch (uart_state) {
    case WAITTING_RST:
      if(isCommandRST()){
        // I put this command here because the value of packet is not change in the same request
        adc_value = HAL_ADC_GetValue(&hadc1);
        // Change state
        uart_state = SEND_REP;
      }
      break;
    case SEND_REP:
      // Send packet
      HAL_UART_Transmit(&huart2, (void*)buffer_uart, sprintf(buffer_uart, "\r\n!ADC=%d#", adc_value),200);
      // Change state
      uart_state = WAITTING_OK;
      setTimer0(PERIOD_FOR_RESEND);
      break;
    case WAITTING_OK:
      // Received command "!OK#"
      if(isCommandOK()){
        stopTimer0();
        uart_state = WAITTING_RST;
        HAL_UART_Transmit(&huart2, (void*)buffer_uart, sprintf(buffer_uart, "\r\n"),200);
      }
      // Have waited command "!OK#" for 3s
      if(getTimer0Flag() == 1){
            uart_state = SEND_REP;
        }
        break;
      default:
        break;
    }
}
```

Program 1.13: Source code in file uart_communicating.c

7.3.6 Module uart processing

Chứa hàm uart interrupt để nhận kí tự từ terminal.

1. *uart_processing.h*

```
#include "main.h"

#ifndef INC_UART_PROCCESSING_H_
#define INC_UART_PROCCESSING_H_

// Uart interrupt function
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
   ;
// Get buffer_flag when received character from
   terminal
uint8_t getBufferFlag();
// Set buffer_flag = 0
void resetBufferFlag();
// Get character received from terminal
uint8_t getChar();

#endif
```

Program 1.14: Source code in file uart_processing.h

2. *uart_processing.c*

```
#include "uart_proccessing.h"
#include "main.h"

#define MAX_SIZE  30
extern uint8_t temp = 0;
uint8_t buffer[MAX_SIZE];
uint8_t index_buffer = 0;
uint8_t buffer_flag = 0;

extern UART_HandleTypeDef huart2;

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
   {
  if(huart->Instance == USART2){
    buffer[index_buffer++] = temp;
    if(index_buffer >= MAX_SIZE) index_buffer = 0;
    buffer_flag = 1;
    HAL_UART_Receive_IT(&huart2, &temp, 1);
    // Display character have just entered
    HAL_UART_Transmit(&huart2, &temp, 1, 50);
  }
}

uint8_t getBufferFlag(){
  return buffer_flag;
}

void resetBufferFlag(){
  buffer_flag = 0;
}

uint8_t getChar(){
  return temp;
}
```
Program 1.15: Source code in file uart_processing.c

7.4 Video demo

Video demo: Click here!