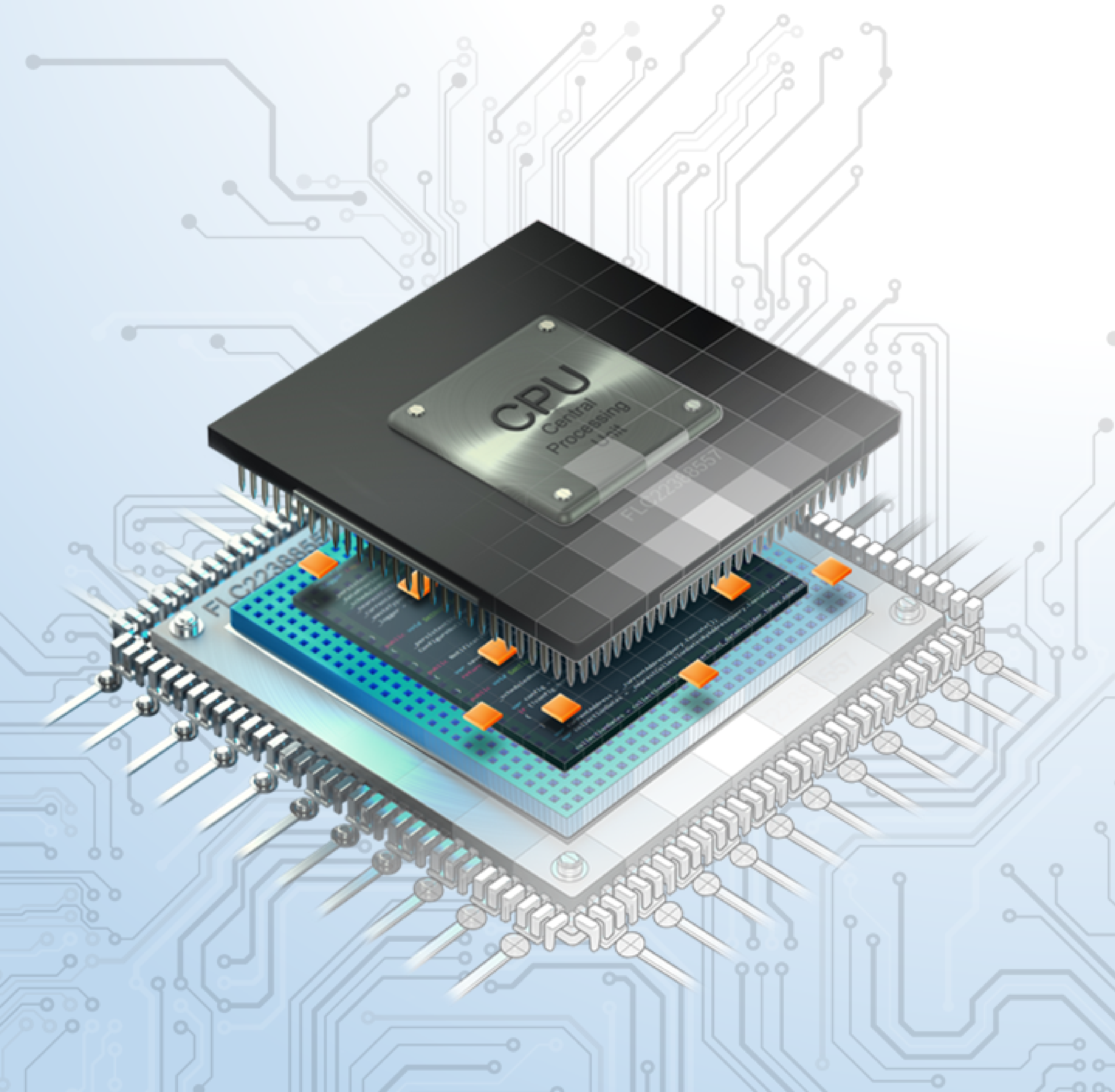
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller



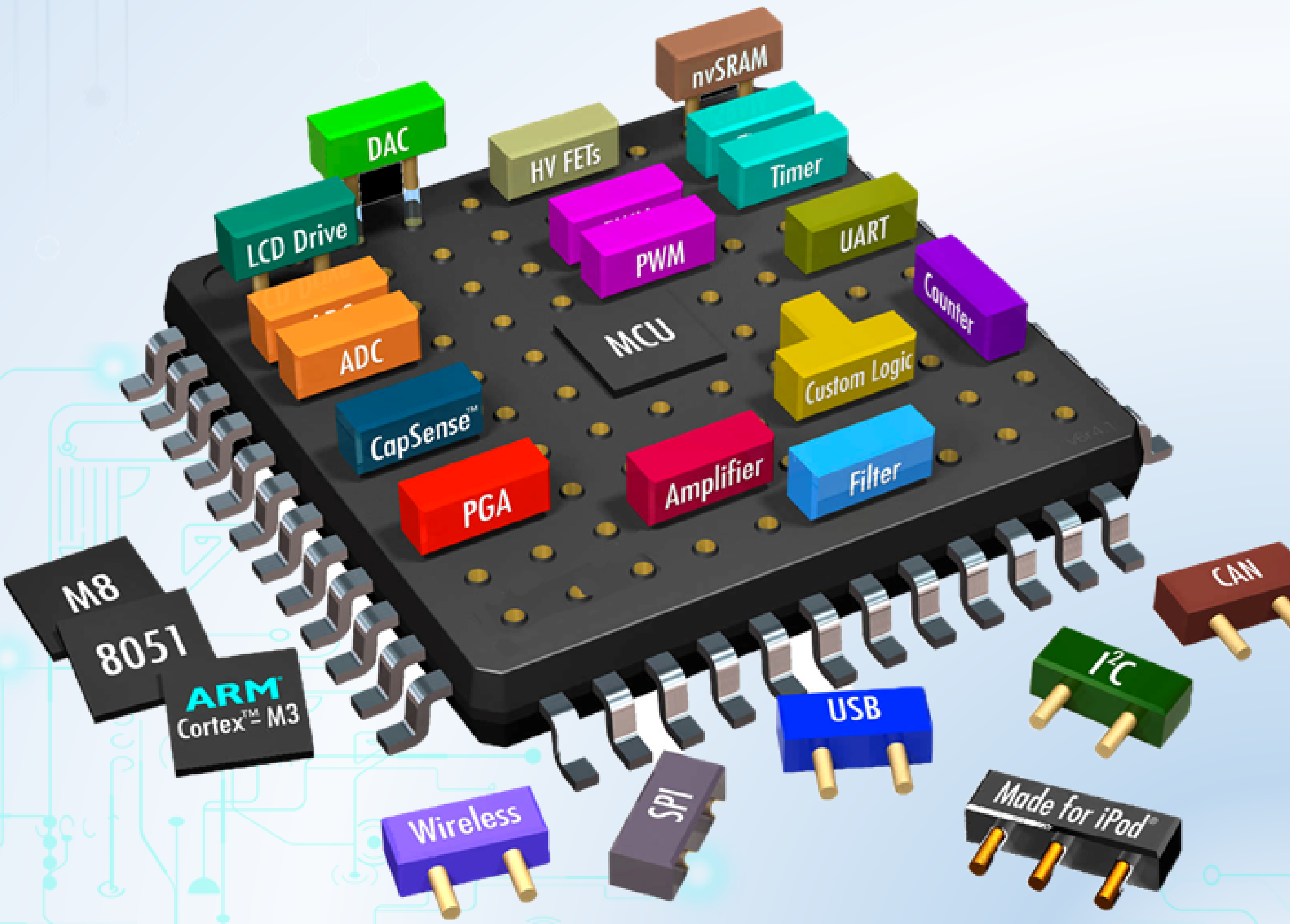
Dr. Le Trong Nhan

- **Name**: HOANG VAN PHI
- **MSSV**: 2252608
- **Class**: CO3010_CC01
- **Teacher**: NGUYEN THIEN AN
- **Lab**: 5
- **Link GitHub**: https://github.com/PhiuTheWind/VXL

Mục lục

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.

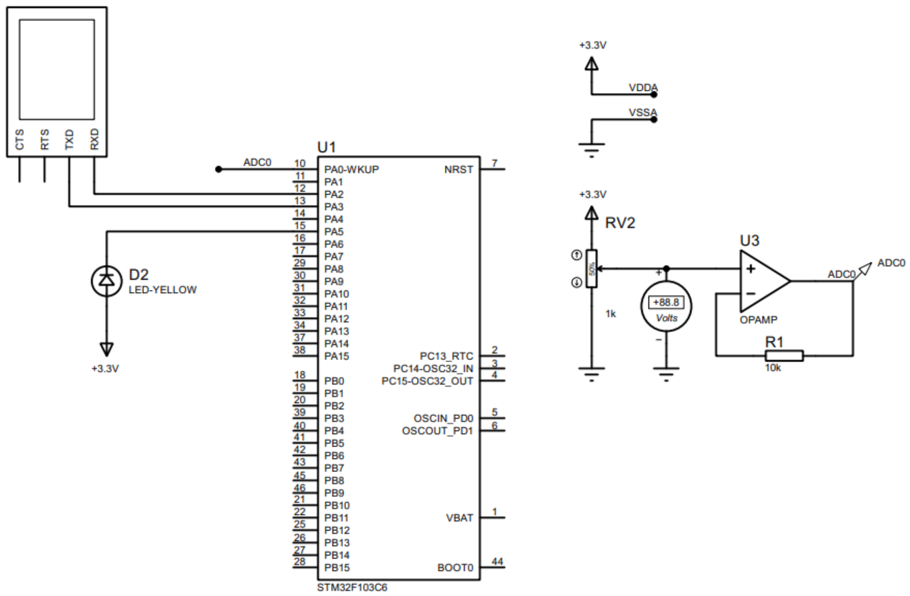
A critical issue, but not really frequently occurred, in the flow control is that the processing rate is slower than the transmission rate. Due to this reason each receiving device has a block of memory that is commonly known as buffer, that is used to store the incoming data until this data will be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



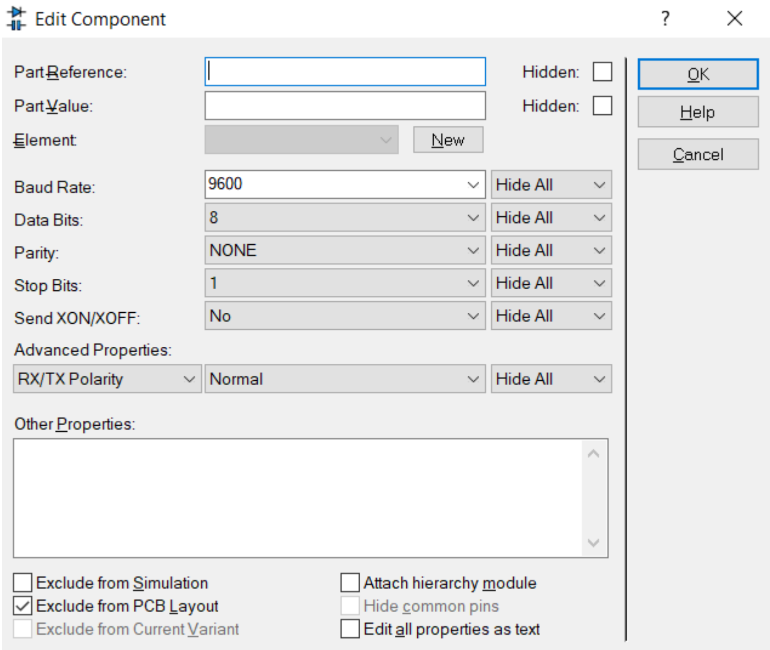
Hình 6.1: Simulation circuit on Proteus

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:



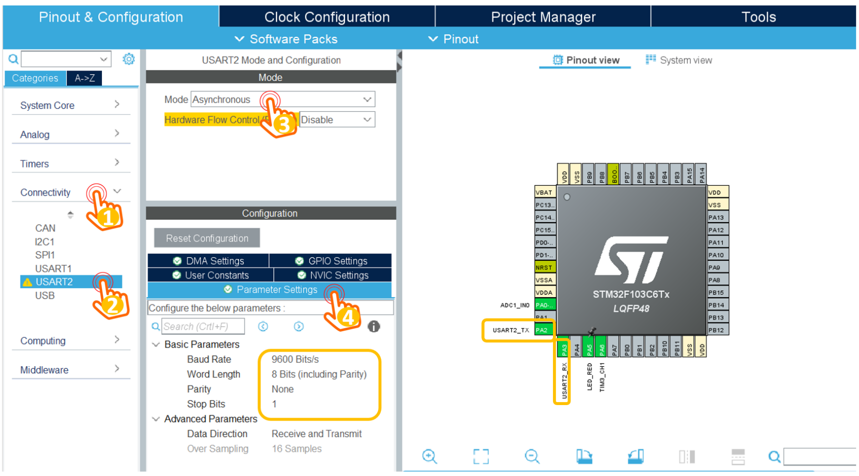
Hình 6.2: Terminal configuration

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

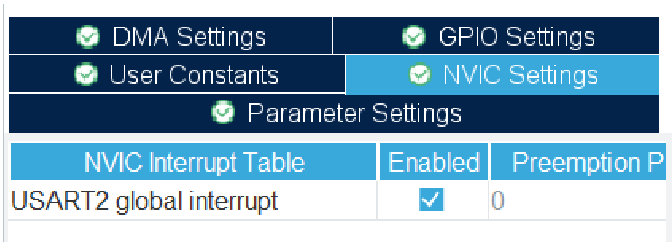
From the ioc file, select **Connectivity**, and then select the **USART2**. The parameter settings for UART channel 2 (USART2) module are depicted as follows:



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.

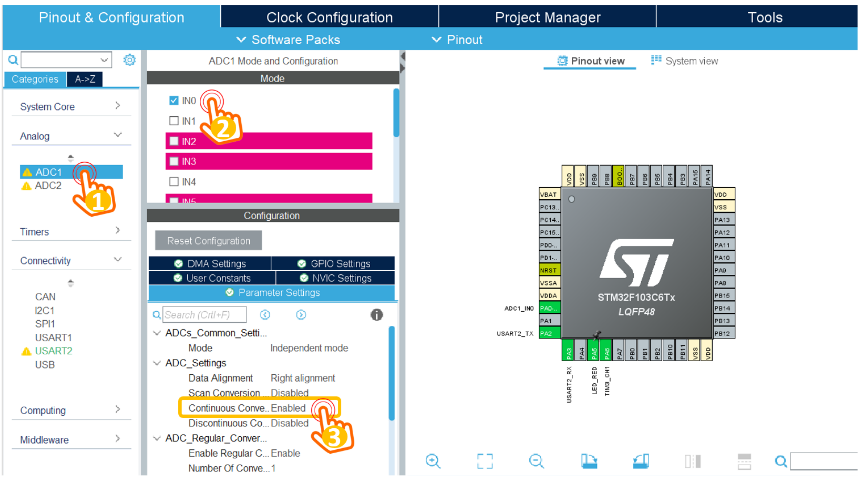
Finally, the NVIC settings are checked to enable the UART interrupt, as follows:



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.

```c
/* 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 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:

```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 6.2: 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 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.

```
#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 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:

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

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:

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

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.

```
/*
 * uart.h
 *
 *  Created on: Dec 9, 2024
 *      Author: phihv
 */

#ifndef INC_UART_H_
#define INC_UART_H_

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <main.h>
#include <string.h>
#include <timer.h>

#define INIT 1
#define PARSER 2
#define WAIT 3
#define SENDING 4
#define END 5
#define MAX_BUFFER_SIZE 30

extern UART_HandleTypeDef huart2;
extern ADC_HandleTypeDef hadc1;

extern uint16_t ADC_value;
extern uint8_t temp;
extern uint8_t text[MAX_BUFFER_SIZE];
extern uint8_t index_text;

extern uint8_t buffer[MAX_BUFFER_SIZE];
extern uint8_t index_buffer;
extern uint8_t buffer_flag;
extern uint16_t old_value;

extern int parser_status;
extern int uart_status;
extern int RST_flag;
extern int has_ok;
extern int check_first;

extern char str[MAX_BUFFER_SIZE];

void check_content();
void clean_contet();
void command_parser_fsm();
void uart_communication_fsm();


#endif /* INC_UART_H_ */
```

Program 6.7: uart.h

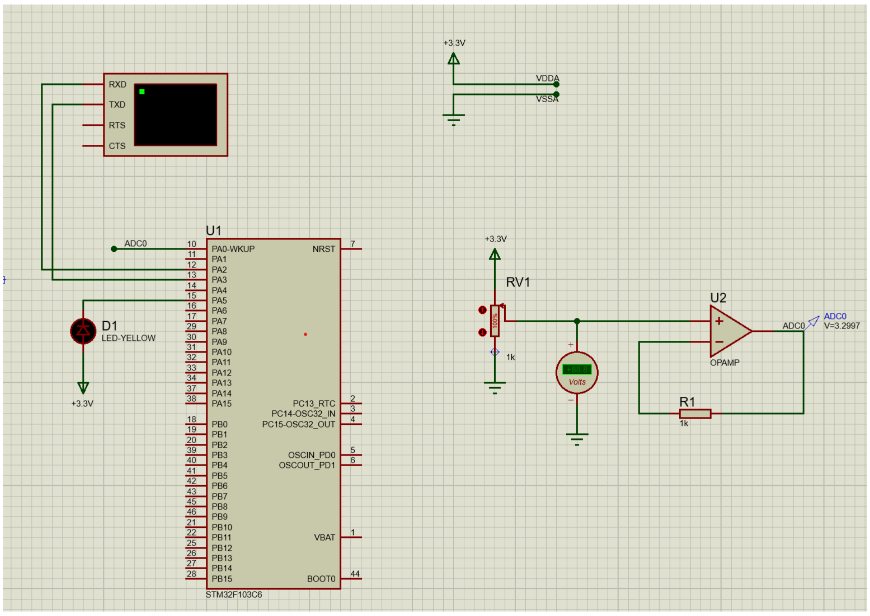
```c
/*
 * uart.c
 *
 *  Created on: Dec 9, 2024
 *      Author: phihv
 */


```

```c
#include "uart.h"


uint16_t ADC_value = 0;
uint16_t old_value = 0;

uint8_t text[MAX_BUFFER_SIZE]= {0};
uint8_t index_text = 0;

uint8_t temp = 0;
uint8_t buffer[MAX_BUFFER_SIZE]={0};
uint8_t index_buffer = 0;
uint8_t buffer_flag = 0;

int has_ok = 0;
int check_first = 1;
int parser_status = 0;
int uart_status = 0;
int RST_flag = 0;


char str[MAX_BUFFER_SIZE]={0};

void clean_content(){
  memset(text, 0, sizeof(text));
}

void check_content(){
  if(text[0]=='R' && text[1]=='S' && text[2]=='T' && text[3] == '#'){ // If !RST# => flag on
    RST_flag=1;
    clean_content();
  }
  else if(text[0]=='O' && text[1]=='K' && text[2]=='#' && RST_flag==1){   // If !OK# => end
    uart_status=END;
    RST_flag=0;
  }
  else if((text[0]!='R' || text[1]!='S' || text[2]!='T' || text[3] != '#') && RST_flag==0){ // Wrong syntax content
    uart_status=END;
    HAL_UART_Transmit(&huart2, (uint8_t*)str, sprintf(str, "Wrong syntax \n"), 1000);
  }
  else{  // The last case user have not enter "OK"
    uart_status=SENDING;
  }
}
void command_parser_fsm(){
```

```
  switch(parser_status){
  case INIT:
    if(temp == '!'){ // If the first is !, it will move to the parser to add the char into buffer, else, do not thing
      parser_status=PARSER;
      index_text = 0;
    }
    break;
  case PARSER: // Check if Enter was push or not, if Push, move to check content between ! and #, else continue save data
    if(temp == '\r'){
      check_content();
      parser_status=INIT;
    }
    else if(temp != '\r'){
      text[index_text] = temp;
      index_text++;
      if(index_text == MAX_BUFFER_SIZE){
        index_text = 0;
      }
    }
    break;
  default:
    break;
  }
}

void uart_communication_fsm(){
  switch(uart_status){
  case WAIT: // Wait the signal of RST_flag
    if(RST_flag==1){
      uart_status=SENDING;
      setTimer2(200);
    }
    break;
  case SENDING: //Sending data when  RST_flag = 1 and timer 2 count down to 0
    if (timer2_flag == 1) {
      if(check_first == 1)
      {
        ADC_value= HAL_ADC_GetValue(&hadc1);
        HAL_UART_Transmit(&huart2, (uint8_t*)str, sprintf(str, "!ADC=%d#\r\n", ADC_value), 1000);
        old_value = ADC_value;
        check_first = 0;
        setTimer2(300);
      }
```

```
      else if(check_first == 0)
      {
        HAL_UART_Transmit(&huart2, (uint8_t*)str, sprintf(str, "!ADC=%d#\r\n", old_value), 1000);
        setTimer2(300);
      }
    }
    break;
  case END: // clear memory and back to Wait for new communication
    clean_content();
    uart_status = WAIT;
    HAL_UART_Transmit(&huart2, (uint8_t*)str, sprintf(str, "End. \n"), 1000);
    check_first = 1;
    break;
  default:
    break;
  }
}
```

Program 6.8: uart.c



Hình 6.6: Proteus Simulation

```
/* USER CODE BEGIN Header */
/**

  * @file           : main.c
  * @brief          : Main program body

  * @attention
  *
```

```c
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "uart.h"
#include "timer.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables
```

```
    ------------------------------------------------------------
    */
ADC_HandleTypeDef hadc1;

TIM_HandleTypeDef htim2;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code
    ------------------------------------------------------------
    */
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */

/* USER CODE BEGIN PV */
/* USER CODE END PV */

/* Private function prototypes
    -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);
/* USER CODE BEGIN 0 */
/* USER CODE END 0 */

int main(void) {
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

  /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_ADC1_Init();
```

```c
  MX_USART2_UART_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim2);
  HAL_UART_Receive_IT (&huart2 , &temp , 1) ;
  HAL_ADC_Start(&hadc1);

  setTimer1(50);
  parser_status = INIT;
  uart_status = WAIT;
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if(timer1_flag == 1)
    {
      HAL_GPIO_TogglePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin);
      setTimer1(50);
    }
    if(buffer_flag == 1)
    {
      command_parser_fsm();
      buffer_flag=0;
    }
    uart_communication_fsm();

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief Callback UART khi  n h n  d  l i u
  */
void HAL_UART_RxCpltCallback ( UART_HandleTypeDef * huart ) {
 if(huart -> Instance == USART2 ){
   buffer[index_buffer] = temp;
   index_buffer++;
   if( index_buffer == 30) index_buffer = 0;

   buffer_flag = 1;
   HAL_UART_Transmit(&huart2, &temp, 1, 1000);
   HAL_UART_Receive_IT (&huart2 , &temp , 1);
```

```
  }
}



/**
  * @brief Callback Timer  n g t     nh     k
  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM2) {
        timerRun(); //  G i   h  m   x    l í timer
    }
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
```

```
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
   FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
```

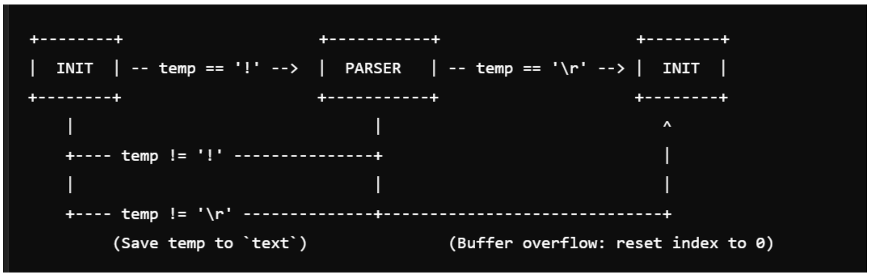
```
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 7999;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 9;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload =
   TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL
   ;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig
   ) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode =
   TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &
   sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 9600;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
```

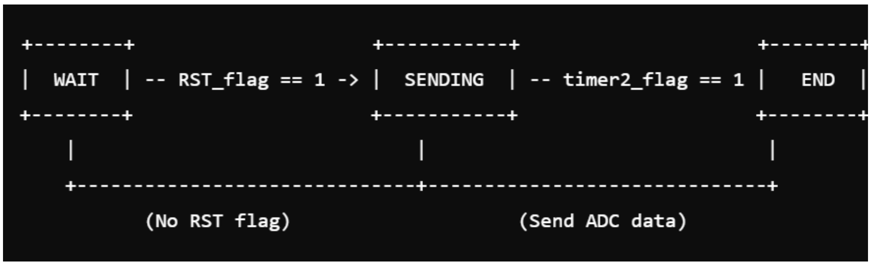
```c
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin,
   GPIO_PIN_RESET);

  /*Configure GPIO pin : LED_YELLOW_Pin */
  GPIO_InitStruct.Pin = LED_YELLOW_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LED_YELLOW_GPIO_Port, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error
   occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL
   error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
```

```c
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#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