Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Лабораторна робота №2

Мікропроцесорні технології інтернету речей

Тема: *Аналогово-цифрове перетворення. UART*

Виконав
студент групи ІП-11:
Панченко С. В.

Перевірив:
Стельмах А.А.

Київ 2025

ЗМІСТ

1 Мета лабораторної роботи.....6

2 Завдання.....7

3 Виконання.....8

4 Висновок.....13

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ.....14

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Симуляція роботи мікропроцесора в програмі Proteus. Генерація коду за допомогою програми Cube MX. Реалізація АЦП. Робота з UART. Створення і компіляція робочої програми на мові програмування C++.

2 ЗАВДАННЯ

Щоб отримати 60

1. Налаштувати АЦП
2. Налаштувати UART
3. Виводити в UART значення АЦП

Щоб отримати 85

1. Виконати всі завдання з «Щоб отримати 60»
2. Налаштувати ще один UART та реалізувати базові команди такі як: увімкнути / вимкнути світлодіод

Щоб отримати 100

1. Виконати всі завдання з «Щоб отримати 100»
2. Реалізувати команди по налаштуванні таймерів на певний час в секундах, а також ШІМ.

3 ВИКОНАННЯ

Налаштував максимальну частоту тактування, виконав налаштування пінів та периферії, необхідної для виконання лабораторної роботи.

Використовується ADC1, увімкнений режим без сканування (ScanConvMode = DISABLE), без розриву (DiscontinuousConvMode = DISABLE), запуск — програмний (ExternalTrigConv = SOFTWARE_START). Режим безперервної конверсії (ContinuousConvMode = ENABLE). Кількість конверсій: NbrOfConversion = 1.

Налаштування USART2 для відладки/моніторингу. Ініціалізований як UART_MODE_TX_RX. Використовується DMA для передачі (huart2 → DMA_Channel7).

Вирішив одразу використовувати DMA для захоплення великої кількості відліків АЦП та відсутності обмежень у швидкості виконання інших завдань. У мене UART2 з DMA на передачу (DMA1_Channel7) — для швидкої та частої відправки повідомлень, особливо при роботі з АЦП та ШІМ. Але якщо викликати HAL_UART_Transmit_DMA() двічі поспіль, не дочекавшись завершення попередньої передачі — нічого не відправиться або буде сміття. HAL не ставить їх у чергу — він просто відмовляється запускати другу передачу, якщо перша ще триває. У функції uart_send_dma() використовується прапорець uart_busy, який захищає від цього. HAL_UART_Transmit_DMA() не копіює дані, а просто починає передачу з вказаного буфера. Якщо передати рядок з локального масиву (char msg[64]), і він вийде з області видимості — будуть артефакти. У uart_send_dma() зберігається рядок у глобальному tx_buf[], який існує весь час передачі.

DMA в режимі Circular — це апаратний буфер АЦП. Черга рядків у uart_queue — це програмний буфер UART передачі.

```
#define UART_QUEUE_LEN 8
#define UART_LINE_LEN 128
```

```
char uart_queue[UART_QUEUE_LEN][UART_LINE_LEN];

void uart_send_dma(const char* str) {
  if (!uart_busy && uart_q_count == 0) {
    // можна одразу передавати
    HAL_UART_Transmit_DMA(...);
  } else if (uart_q_count < UART_QUEUE_LEN) {
    // додаємо в чергу
    strncpy(uart_queue[uart_q_tail], str, UART_LINE_LEN - 1);
    uart_q_tail = (uart_q_tail + 1) % UART_QUEUE_LEN;
    uart_q_count++;
  }
}
```

Чому: DMA Circular Mode працює тільки на прийом даних від АЦП і взагалі не стосується UART. А UART DMA працює в режимі Normal, навіть якщо встановити в CubeMX Circular — це не робить передачу по UART циклічною. Якщо викликати HAL_UART_Transmit_DMA() до завершення попередньої передачі — вся передача ігнорується. Саме тому потрібен власний кільцевий буфер uart_queue[], щоб не втратити рядки, коли UART зайнятий.

Отже, DMA працює в кільцевому режимі (Circular). ADC працює в ContinuousConvMode = ENABLE. HAL один раз викликає HAL_ADC_Start_DMA(...), і після цього adc_buf[0..4095] постійно перезаписуються свіжими значеннями. Можна читати adc_buf[0] або будь-який інший біт масиву. У проекті використовується усереднення, щоб згладити шум:

```
uint32_t sum = 0;
for (int i = 0; i < 64; i++) {
    sum += adc_buf[(ADC_BUF_LEN - 1) - i]; // останні 64 значення
}
uint16_t avg = sum / 64;
snprintf(msg, sizeof(msg), "ADC_avg = %u\r\n", avg);
```

Наступне завдання. Реалізовано додатковий UART — USART3 (RX). USART3 працює в режимі прийому та передачі (RX) по перериваннях. Піни: PB10 (TX), PB11 (RX). Конфігурація: 115200 бод, 8N1. HAL_UART_Receive_IT(&huart3, &rx_byte, 1). Головним було реалізувати

RxCpltCallback().

Буферизація команд з USART3. Команди приходять по одному символу. Формуються в буфері rx_buffer[]. Коли надходить \n або \r — виконується розбір.

Команди: LED_ON / LED_OFF. Світлодіод на PC13. Включення / виключення: HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET); // ON, HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET); // OFF.

Наступне завдання. Реалізація ШІМ на TIM2, PA0 (CH1). Ініціалізація через MX_TIM2_Init(), з режимом PWM1. Prescaler: 72-1 (1 мкс), Period: 100-1 → частота 10 кГц. Канал: TIM_CHANNEL_1. Початковий Pulse = 0.

```
htim2.Init.Prescaler = 72-1;
htim2.Init.Period = 100-1;
...
sConfigOC.OCMode = TIM_OCMODE_PWM1;
```

Команда LED_PWM <яскравість %> <тривалість мс>. Наприклад: LED_PWM 35 2000.

Обробка у HAL_UART_RxCpltCallback():

```
if (strncmp(rx_buffer, "LED_PWM", 7) == 0) {
  int brightness;
  uint32_t duration;
  if (sscanf(rx_buffer, "LED_PWM %d %lu", &brightness, &duration) == 2) {
    StartLedPWM((uint8_t)brightness, duration);
  }
}
```

Функція StartLedPWM() конфігурує PA0 в режим AF_PP (альтернативна функція для PWM), стартує HAL_TIM_PWM_Start, встановлює __HAL_TIM_SET_COMPARE(...), зберігає pwm_end_time = HAL_GetTick() + duration.

```
void StartLedPWM(uint8_t brightness_percent, uint32_t duration_ms) {
  ...
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, brightness_percent);
  pwm_end_time = HAL_GetTick() + duration_ms;
}
```

Функція UpdateLedPWM() в main-loop. В головному циклі while(1) викликається UpdateLedPWM(). Перевіряє: якщо HAL_GetTick() >= pwm_end_time → виконує: HAL_TIM_PWM_Stop(...), повертає PA0 в GPIO (режим OUTPUT_PP), вимикає світлодіод (LOW).

4 ВИСНОВОК

5 ВИСНОВОК

У ході виконання лабораторної роботи я успішно провів симуляцію роботи мікропроцесора в програмі Proteus та згенерував базовий код за допомогою Cube MX. Реалізував роботу з АЦП для зчитування аналогових сигналів та впровадив DMA для оптимізації передачі даних. Налаштував обмін інформацією через UART з використанням буферизації та механізму переривань. Розробив систему керування світлодіодом через UART-команди, включаючи ШІМ-регулювання яскравості. У процесі роботи навчився ефективно налаштовувати периферійні пристрої мікроконтролера, працювати з DMA для розвантаження процесора, реалізовувати асинхронну комунікацію та керувати зовнішніми пристроями за допомогою таймерів. Впевнився у перевагах використання DMA для підвищення ефективності роботи з периферією, важливості правильної буферизації даних та доцільності застосування Proteus для тестування перед фізичним програмуванням. Також переконався, що мова C++ забезпечує необхідну гнучкість і ефективність для програмування вбудованих систем, а симуляція в Proteus дозволяє швидко перевірити роботу програми перед завантаженням у мікроконтролер.

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду

(Найменування програми (документа))

Жорсткий диск

(Вид носія даних)

(Обсяг програми (документа), арк.)

Студента групи ІП-11 4 курсу

Панченка С. В

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

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

******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 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"

/* Private includes
----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
```

```
/* USER CODE END Includes */

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

/* USER CODE END PTD */

/* Private define
------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define ADC_BUF_LEN 4096
#define UART_QUEUE_LEN 8
#define UART_LINE_LEN  128
/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables
-----------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

TIM_HandleTypeDef htim2;

UART_HandleTypeDef huart2;
UART_HandleTypeDef huart3;
DMA_HandleTypeDef hdma_usart2_tx;

/* USER CODE BEGIN PV */
uint16_t adc_buf[ADC_BUF_LEN];
char tx_buf[128];
```

```
char rx_buffer[64];
uint16_t tx_len = 0;
volatile uint8_t uart_busy = 0;
uint8_t uart3_rx_byte;
char uart_queue[UART_QUEUE_LEN][UART_LINE_LEN];
uint8_t uart_q_head = 0;
uint8_t uart_q_tail = 0;
uint8_t uart_q_count = 0;
char rx_buffer[64];
uint8_t rx_byte;
uint8_t rx_index = 0;


/* USER CODE END PV */

/* Private function prototypes
-----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_TIM2_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code
---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void uart_send_dma(const char* str) {
  if (!uart_busy && uart_q_count == 0) {
    tx_len = strlen(str);
    if (tx_len == 0 || tx_len >= sizeof(tx_buf)) return;
    memcpy(tx_buf, str, tx_len);
    tx_buf[tx_len] = '\0';
```

```
    uart_busy = 1;
    HAL_UART_Transmit_DMA(&huart2, (uint8_t*)tx_buf, tx_len);
  }
  else if (uart_q_count < UART_QUEUE_LEN) {
    strncpy(uart_queue[uart_q_tail], str, UART_LINE_LEN - 1);
    uart_queue[uart_q_tail][UART_LINE_LEN - 1] = '\0';
    uart_q_tail = (uart_q_tail + 1) % UART_QUEUE_LEN;
    uart_q_count++;
  }
}



volatile uint8_t pwm_active = 0;
uint32_t pwm_end_time = 0;

void StartLedPWM(uint8_t brightness_percent, uint32_t duration_ms)
{
    if (brightness_percent > 100) brightness_percent = 100;

    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,
brightness_percent);

    pwm_active = 1;
    pwm_end_time = HAL_GetTick() + duration_ms;
}

void UpdateLedPWM()
{
    if (pwm_active && HAL_GetTick() >= pwm_end_time)
```

```
    {
        HAL_TIM_PWM_Stop(&htim2, TIM_CHANNEL_1);

        GPIO_InitTypeDef GPIO_InitStruct = {0};
        GPIO_InitStruct.Pin = GPIO_PIN_0;
        GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
        HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);

        pwm_active = 0;
    }
}




/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU
Configuration--------------------------------------------------------
---*/
```

```
  /* Reset of all peripherals, Initializes the Flash interface and
the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  MX_USART2_UART_Init();
  MX_USART3_UART_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */
  HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_buf, ADC_BUF_LEN);
  HAL_UART_Receive_IT(&huart3, &uart3_rx_byte, 1);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
      UpdateLedPWM();
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
```

```
  /* USER CODE END 3 */
}

/**
  * @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_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  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_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
```

```
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) !=
HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV6;
  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 */

  /* 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();
  }
  /* 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};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 72-1;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 100-1;
```

```
  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();
  }
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2,
&sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1)
!= HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */
  HAL_TIM_MspPostInit(&htim2);
```

```
}

/**
  * @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 = 115200;
  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 USART3 Initialization Function
```

```
  * @param None
  * @retval None
  */
static void MX_USART3_UART_Init(void)
{

  /* USER CODE BEGIN USART3_Init 0 */


  /* USER CODE END USART3_Init 0 */

  /* USER CODE BEGIN USART3_Init 1 */

  /* USER CODE END USART3_Init 1 */
  huart3.Instance = USART3;
  huart3.Init.BaudRate = 115200;
  huart3.Init.WordLength = UART_WORDLENGTH_8B;
  huart3.Init.StopBits = UART_STOPBITS_1;
  huart3.Init.Parity = UART_PARITY_NONE;
  huart3.Init.Mode = UART_MODE_TX_RX;
  huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart3.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART3_Init 2 */


  /* USER CODE END USART3_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
```

```
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel1_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);
  /* DMA1_Channel7_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel7_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel7_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
  // USART3 TX (PB10)
  GPIO_InitStruct.Pin = GPIO_PIN_10;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  // USART3 RX (PB11)
  GPIO_InitStruct.Pin = GPIO_PIN_11;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
/* USER CODE END MX_GPIO_Init_1 */
```

```
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);

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

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

/* USER CODE BEGIN 4 */
// Called when first half of buffer is filled
void HAL_ADC_ConvHalfCpltCallback(ADC_HandleTypeDef* hadc) {

}

// Called when buffer is completely filled
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) {
    uint32_t sum = 0;
      if (hadc->Instance == ADC1) {
          char msg[64];
          for (int i = 0; i < 64; i++) {
              sum += adc_buf[(ADC_BUF_LEN - 1) - i];
          }
          uint16_t avg = sum / 64;
          snprintf(msg, sizeof(msg), "ADC_avg = %u\r\n", avg);
          uart_send_dma(msg);
```

```
      }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    if (huart->Instance == USART3)
        {
            if (rx_byte == '\n' || rx_byte == '\r')
            {
                rx_buffer[rx_index] = '\0';

                if (strcmp(rx_buffer, "LED_ON") == 0)
                {
                    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13,
GPIO_PIN_RESET);
                    char msg[] = "LED is ON\r\n";
                    HAL_UART_Transmit(&huart3, (uint8_t*)msg,
strlen(msg), 100);
                }
                else if (strcmp(rx_buffer, "LED_OFF") == 0)
                {
                    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13,
GPIO_PIN_SET);
                    char msg[] = "LED is OFF\r\n";
                    HAL_UART_Transmit(&huart3, (uint8_t*)msg,
strlen(msg), 100);
                }
                else if (strncmp(rx_buffer, "LED_PWM", 7) == 0)
                {
                    int b_tmp;
                    uint32_t duration = 0;
                    uint8_t brightness = 0;

                    if (sscanf(rx_buffer, "LED_PWM %d %lu",
&b_tmp, &duration) == 2)
                    {
                        brightness = (uint8_t)b_tmp;
```

```
                        char msg[64];
                        char percent_str[5];
                        char duration_str[12];

                        itoa(brightness, percent_str, 10);
                        itoa(duration, duration_str, 10);

                        strcpy(msg, "PWM ");
                        strcat(msg, percent_str);
                        strcat(msg, "% for ");
                        strcat(msg, duration_str);
                        strcat(msg, " ms\r\n");

                        HAL_UART_Transmit(&huart3, (uint8_t*)msg,
strlen(msg), 100);
                        StartLedPWM(brightness, duration);
                    }
                    else
                    {
                        char msg[] = "BAD FORMAT\r\n";
                        HAL_UART_Transmit(&huart3, (uint8_t*)msg,
strlen(msg), 100);
                    }
                }
                else
                {
                    char msg[] = "UNKNOWN\r\n";
                    HAL_UART_Transmit(&huart3, (uint8_t*)msg,
strlen(msg), 100);
                }

                rx_index = 0;
                memset(rx_buffer, 0, sizeof(rx_buffer));
            }
            else
            {
                rx_buffer[rx_index++] = rx_byte;
```

```
                if (rx_index >= sizeof(rx_buffer))
                    rx_index = 0;
            }

            HAL_UART_Receive_IT(&huart3, &rx_byte, 1);
        }
}


void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
  if (huart == &huart2) {
    if (uart_q_count > 0) {
      strncpy(tx_buf, uart_queue[uart_q_head], UART_LINE_LEN);
      tx_len = strlen(tx_buf);
      uart_q_head = (uart_q_head + 1) % UART_QUEUE_LEN;
      uart_q_count--;

      HAL_UART_Transmit_DMA(&huart3, (uint8_t*)tx_buf, tx_len);
    } else {
      uart_busy = 0;
    }
  }
}

/* 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
/**
  * @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 */
```