STM32 ??????

- STM32 Nucleo-F446RC + CubeMX 完整教學系列
- STM32 教學系列 第 1 節: 環境搭建 + Blink LED
- STM32 教學系列 第 2 節: 時脈控制講解
- STM32 教學系列 第 3 節: GPIO 進階 (按鍵、中斷、去彈跳)
- STM32 教學系列 第 4 節: UART 串列通訊 + 命令介面
- STM32 教學系列 第 5 節: ADC (多種轉換方式與如何調用 Channel)
- STM32 教學系列 第 6 節: 定時器 Timer + PWM
- STM32 教學系列 第 7 節: DMA (直接記憶體存取)
- STM32 教學系列 第 8 節: I2C 通訊 + DMA
- STM32 教學系列 第 9 節: SPI 通訊 + DMA
- STM32 教學系列 第 10 節: CANbus 通訊 (兩板間通訊)
- STM32 教學系列 第 11 節: RS-485 通訊
- STM32 教學系列 第 12 節: 硬體 CRC 計算
- STM32 教學系列 第 13 節: FreeRTOS (實時操作系統)
- STM32 初次發車測試程式 (Basic Throttle Test)
- STM32 智慧四輪驅動與 VESC 控制系統 (含電子差速與軟體 CAN Buffer)
- STM32 四輪驅動與 VESC 控制系統 (4WD_RWD 動態切換)

STM32 Nucleo-F446RC + CubeMX ??????

? ????

歡迎來到 **STM32** 嵌入式開發完整教學系列！本教學系列共 **13** 節課程，涵蓋從環境搭建到高級 RTOS 應用的完整學習路徑。

> “□ 適合對象：大專院校二年級以上學生、嵌入式初學者
> □ 建議時間：每節 3-5 小時
> □ 所需硬體：STM32 Nucleo-F446RC × 1
> □ 開發工具：STM32CubeIDE (免費)

? ????

? ?????? 1-3 ??

這三節課程為您建立完整的開發環境，並掌握 GPIO 基本操作。

節次	標題	重點概念	難度
第 1 節	環境搭建 **+ Blink LED**	IDE 安裝、第一個程式、GPIO 輸出	□
第 2 節	時脈控制講解	振盪器、PLL 倍頻、Clock Tree	□□
第 3 節	**GPIO** 進階	按鍵檢測、中斷系統、去彈跳	□□

? ???????? 4-9 ??

掌握 STM32 常見的通訊協議，是開發實際應用的基礎。

節次	標題	重點概念	難度

□ 4 □	**UART □□□□**	□□□□□□□□□□□□□ □	□□
□ 5 □	**ADC □□□□**	□□□□□□□□□□□□□ □□	□□
□ 6 □	**□□□ + PWM**	□□□□□ PWM □□□□□□□□	□□
□ 7 □	**DMA □□**	□ CPU □□□□□□□□□□□□	□□□
□ 8 □	**I2C □□**	□□□□□□□□□□□□□ □□	□□□
□ 9 □	**SPI □□□□**	□□□□□□□□□□ SD □□□	□□□

? ????????? 10-12 ??

□□□□□□□□□□□□□□□□□□□□□□□□

□□	□□	□□□□	□□
□ 10 □	**CANbus □□**	□□□□□□□□□□□□□ □□	□□□
□ 11 □	**RS-485 □□**	□□□□□□□□□□□□□	□□□
□ 12 □	**□□ CRC □□**	□□□□□□□□□□□□□ □□	□□

? ???????? 13 ??

□□□□□□□□□□□□□□□□□□□□□□□

□□	□□	□□□□	□□
□ 13 □	**FreeRTOS**	□□□□□□□□□□□□□ □□	□□□□

? ??????

? 1 ?????? + Blink LED

□□ □ `stm32_lesson_01_env.md`

□ □□□□□

- 完成 STM32CubeIDE 安裝與配置
- 認識 Nucleo-F446RC 硬體
- 實現第一個 LED 閃爍程式

□□ 重點內容:

- STM32CubeIDE 下載與安裝步驟
- 開發板組成與主要接腳
- CubeMX 配置工作流程
- 完整可編譯的 Blink LED 代碼

? 2 ????????

檔案： `stm32_lesson_02_clock.md`

□ □□□□□

- 理解 STM32F446 的時脈架構
- 掌握 Clock Tree 配置
- 透過改變時脈觀察 LED 動作變化

□□ 重點內容:

- 振盪器（HSI、HSE）與 PLL 倍頻原理
- 時脈分頻器（AHB、APB1、APB2）說明
- Clock Tree 工作流程
- 多時脈配置驗證實驗

? 3 ??GPIO ?????????????

檔案： `stm32_lesson_03_gpio_adv.md`

□ □□□□□

- 掌握 GPIO 輸入配置
- 理解外部中斷系統（EXTI）
- 實現按鍵去彈跳算法

□□ 重點內容:

- GPIO 各種工作模式詳解
- 上拉/下拉配置與應用

- 外部中斷工作原理
- 軟體去彈跳技術實現

? 4 ??UART ???? + ????

檔案： `stm32_lesson_04_uart.md`

□ □□□□□

- 掌握 UART 通訊協定
- 實現收發功能
- 建立簡單的命令解析系統

□□ 重點內容:

- UART 信號線、波特率、資料格式
- 中斷驅動的收發機制
- 命令解析與執行邏輯
- 與電腦通訊的實現

? 5 ??ADC???????????? Channel?

檔案： `stm32_lesson_05_adc.md`

□ □□□□□

- 理解 ADC 工作原理
- 配置不同轉換模式
- 實現多通道採樣

□□ 重點內容:

- 12-bit ADC 分辨率與轉換時間
- 單次、連續、掃描模式配置
- 通道選擇與採樣時間設定
- 類比電壓到數位值的轉換公式

? 6 ????? Timer + PWM

檔案： `stm32_lesson_06_timer_pwm.md`

□ □□□□□

- 理解定時器工作原理
- 掌握 PWM 信號產生
- 實現呼吸燈效果

□□ 重點內容:

- 定時器分類與功能
- PWM 原理與佔空比概念
- PWM 頻率與分辨率計算
- 實現 LED 亮度平滑調控

? 7 ??DMA?????????

檔案： `stm32_lesson_07_dma.md`

□ □□□□□

- 理解 DMA 工作原理
- 配置 ADC + DMA
- 實現高速資料轉移

□□ 重點內容:

- DMA 與 CPU 的角色分工
- 循環模式與普通模式
- DMA 流與通道概念
- ADC + DMA 高效採樣實現

? 8 ??I2C ?? + DMA

檔案： `stm32_lesson_08_i2c.md`

□ □□□□□

- 理解 I2C 協定
- I2C 從設備通訊
- I2C + DMA 應用

□□ 重點內容:

- I2C 時序與尋址機制
- SCL、 SDA 信號線工作原理
- 常見感測器地址與讀寫方式
- 設備掃描與通訊示例

? 9 ??SPI ?? + DMA

檔案： `stm32_lesson_09_13_summary.md`（第 9 節部分）

□ □□□□□

- 理解 SPI 高速序列通訊
- SPI 感測器通訊
- SPI + DMA 應用

□□ 重點內容:

- MOSI、MISO、SCK、CS 信號線
- 主從模式與時序配置
- SPI 與 I2C 的比較
- 高速感測器讀寫實現

? 10 ??CANbus ?????????

檔案： `stm32_lesson_09_13_summary.md`（第 10 節部分）

□ □□□□□

- CAN 協定基礎
- 雙開發板通訊
- 訊息的發送與接收

□□ 重點內容:

- CAN 識別符與優先級仲裁
- CAN 收發器 TJA1050 接線
- 過濾器配置與中斷處理
- Nucleo ↔ Nucleo 通訊實現

? 11 ??RS-485 ??

檔案： `stm32_lesson_09_13_summary.md`（第 11 節部分）

□ □□□□□

- RS-485 差分信號
- 多設備總線通訊

- 半雙工收發切換

□□ 重點內容:

- RS-485 與 RS-232 的區別
- MAX485 收發器配置
- 發送/接收模式切換
- 長距離通訊應用

? 12 ???? CRC ??

檔案： `stm32_lesson_09_13_summary.md`（第 12 節部分）

□ □□□□□

- CRC 校驗原理
- STM32 硬體 CRC
- 應用於通訊協定

□□ 重點內容:

- CRC 多項式與計算方式
- 軟體 vs 硬體 CRC 性能對比
- CRC-32 配置與計算
- 資料完整性驗證實現

? 13 ??RTOS????????

檔案： `stm32_lesson_09_13_summary.md`（第 13 節部分）

□ □□□□□

- RTOS 基本概念
- FreeRTOS 配置
- 任務間通訊

□□ 重點內容:

- 多任務調度與優先級
- 任務同步機制（隊列、信號量、互斥鎖）
- FreeRTOS 核心 API
- 完整的多任務示例應用

?? ????

??????

- ☐ 下載 STM32CubeIDE（v1.13+）： https://www.st.com/en/development-tools/stm32cubeide.html
- ☐ 下載本教學全部 13 個 Markdown 文件
- ☐ 準備 STM32 Nucleo-F446RC 開發板
- ☐ 準備 USB Type-B 傳輸線
- ☐ 安裝 ST-Link 驅動 (通常自動安裝)

??????

```
第 1 節: 環境搭建 ✓
   ↓
第 2 節: 時脈控制 ✓
   ↓
第 3 節: GPIO 進階 ✓
   ↓
第 4 節: UART 通訊 ✓
   ↓
第 5 節: ADC 採樣 ✓
   ↓
第 6 節: Timer + PWM ✓
   ↓
第 7 節: DMA (可選)
   ↓
第 8-12 節: 各類通訊協議
   ↓
第 13 節: RTOS (進階)
```

? ??????

每個課程文件均包含以下結構:

□□學習目標（3 個具體目標）

↓

□□理論基礎（概念與原理）

↓

□□□□□□□□□□□□□

↓

⚙◌ CubeMX 配置（詳細步驟）

↓

□□完整程式碼（可直接編譯）

↓

□□□□□□□□□□□□□□□□□□

↓

□□進階應用（擴展挑戰）

↓

□□□□□□□□□□□□□□□

? ????

????

- □□ STM32F446 資料表
- □□ Nucleo-F446RC 使用手冊
- □□ STM32F4 參考手冊

????

- □□ STM32CubeIDE: 整合開發環境
- □□ STM32CubeMX: 硬體配置工具
- □□ STM32Cube Firmware: 函式庫與驅動

?????

- **Windows**: PuTTY、Tera Term、Arduino IDE
- **Linux**: minicom、picocom
- **macOS**: minicom 或 Arduino IDE

? ????

? ????

1. 按順序學習 - 不要跳躍課程，基礎很重要
2. 動手實驗 - 自己親手敲代碼並燒錄
3. 修改參數 - 改變延遲、頻率等參數觀察變化
4. 深入思考 - 理解為什麼而不只是怎麼做
5. 延伸挑戰 - 完成每節的進階挑戰題

? ????

1. □ □□□□□□□□□□□□□□□
2. □ □□□□□□□□□
3. □ □□□□□□□□□□□
4. □ □□ IDE 自動補全，不手動練習
5. □ □□□□□□□□□□□□

? ??????

????

- 錯誤： `undefined reference to 'HAL_xxx'`
- 原因： HAL 函式庫未連結
- 解決： Project → Properties → C/C++ Build → Libraries 檢查

????

- 錯誤： `Failed to connect to target`
- 原因： ST-Link 驅動未安裝或連接不良
- 解決：檢查裝置管理員，重新插拔 USB

?????

- 原因：主要在 main 迴圈中無窮迴圈
- 解決： 新增 `while(1)` 的除錯輸出確認運行

? ????

如遇問題，依序嘗試：

1. 查看本教學的「常見問題 & 解決方案」部分
2. 檢查 STM32 官方社群論壇
3. 參考相關的開源項目代碼

? ???????

本教學系列基於 **CC-BY-SA 4.0** 開源協議發布。

- □ □□□□□□□□□□
- □ □□□□□□□□□
- □ □□□□□□
- □ □□□□□□□□□□□□□□□

? ????

完成全部 13 節課程後，您將能夠：

□ □□□□ STM32 微控制器
□ □□□□□□□□□ UART、SPI、I2C、CAN、RS-485）
□ □□□□□□□□□□□□□
□ □□□□□□□□□
□ □□□□□□□□□□□
□ □□□□□□□□□

? ???????

完成本系列後，建議進階學習：

1. 進階硬體：USB、以太網、外部記憶體
2. 系統設計：電源管理、EMC/EMI 防護
3. 實際項目：無人機、機器人、物聯網設備
4. 其他平台：ARM Cortex-M0、STM32L 低功耗系列

? ?????

歡迎提供學習建議和課程改進意見!

□ □□□ STM32 □□□□□□□□□□□□□□□□□□

????

	課程連結
□□ 首頁	點此返回
➡ 第1節	環境搭建 + Blink LED
➡ 第2節	時脈控制講解
➡ 第3節	GPIO 進階
➡ 第4節	UART 通訊
➡ 第5節	ADC 採樣
➡ 第6節	Timer + PWM
➡ 第7節	DMA
➡ 第8節	I2C
➡ 第9節	SPI
➡ 第10節	CANbus
➡ 第11節	RS-485
➡ 第12節	硬體 CRC
➡ 第13節	RTOS

最後更新：2026-03-06

版本：1.0.0

語言：繁體中文

難度範圍 □□ ~ □□□□

STM32 ???? ? 1 ?????? + Blink LED

STM32 ???? ? 1 ?????? + Blink LED

? ????

1. 完成 **STM32CubeIDE** 安裝與配置 - 建立開發環境，認識 IDE介面
2. 理解 **Nucleo-F446RC** 硬體 - 認識開發板的主要元件與接腳定義
3. 實現第一個程式 **- Blink LED** - 透過 GPIO 驅動 LED，驗證開發環境正確性

?? ????

????

- **STM32 Nucleo-F446RC** 開發板 × 1
- **USB Type-B** 傳輸線 × 1（隨開發板附贈）
- 麵包板 × 1（選配，後續課程使用）
- **LED + 220Ω** 電阻 × 若干（選配）
- 跳線 × 若干

Nucleo-F446RC ?????

Nucleo-F446RC 是 STMicroelectronics 推出的高性能開發板，搭載 **ARM Cortex-M4** 處理器。開發板已內建：

- **User LED**（綠色 **LED**，接在 **PA5**） - 這是我們第一個實驗的對象
- **User Button**（藍色按鈕，接在 **PC13**） - 後續課程會用到
- **ST-Link v2** 調試器 - 支援程式燒錄和實時調試
- 多個 **GPIO** 接腳 - 可外接各種感測器和執行器

????Blink LED?

組件	接腳	說明
User LED	PA5	開發板內建，直接使用
Ground	GND	參考地

> □□ 重要提示 - 開發板上的 User LED 已直接連接到 PA5，無需額外接線！這是最簡單的測試方案。

? ??????

?? 1???????

進入官方網站下載 STM32CubeIDE:

1. 訪問: https://www.st.com/en/development-tools/stm32cubeide.html
2. 點擊 **Download** 按鈕
3. 根據您的作業系統選擇 (Windows / Linux / macOS)
4. 註冊 ST 帳戶 (如無則建立)
5. 下載最新版本 (本教學基於 v1.13+)

?? 2???? STM32CubeIDE

Windows 安裝流程:

1. 雙擊下載的 `.exe` 安裝檔
2. 選擇安裝位置 (建議 `C:\ST\STM32CubeIDE`)
3. 勾選 **Add STM32CubeIDE to PATH** (方便後續命令列操作)
4. 點擊 **Install** 並等待完成 (約 5-10 分鐘)

Linux 安裝流程:

```
# 解壓縮下載的 tar.gz 檔案
tar -xzf STM32CubeIDE-*.tar.gz -C ~/opt/


# 進入安裝目錄執行安裝腳本
cd ~/opt/STM32CubeIDE-*/
```

```
./install.sh
```

?? 3?????? Workspace ??

1. 啟動 STM32CubeIDE
2. 選擇 Workspace 位置 (例如: `D:\STM32_Workspace`)
3. 點擊 **Launch** 進入 IDE
4. 等待首次初始化 (約 1-2 分鐘)
5. 關閉歡迎頁面

?? 4???????????

1. 用 USB 線連接 Nucleo-F446RC 到電腦
2. Windows 會自動下載並安裝 ST-Link 驅動
3. 打開 裝置管理員 檢查:
    - 應能看到 **STMicroelectronics STLink** 設備
    - 若標記 `▯`▯▯▯▯▯▯▯▯▯ https://www.st.com/en/development-tools/stsw-link009.html
4. 在 STM32CubeIDE 中驗證連接:
    - 點擊 **Window → Preferences**
    - 選擇 **MCU → STMicroelectronics → STM32Cube**
    - 檢查 **ST-Link GDB server path** 是否正確識別

?? CubeMX ????

?? 1??????

1. 在 STM32CubeIDE 中點擊 **File → New → STM32 Project**
2. 搜尋裝置型號: 輸入 **STM32F446RC**
3. 選擇 **STM32F446RCTx**
4. 點擊 **Next →** 設定專案名稱 (例如: `STM32_Lesson01_BlinkLED`)
5. 選擇 **STM32CubeMX** 作為 Toolchain
6. 點擊 **Finish**

?? 2?CubeMX ??

STM32CubeIDE 會自動開啟 CubeMX 配置介面, 您將看到晶片的 Pinout 圖。

GPIO 配置:

1. 在 Pinout 圖上找到 **PA5** (已標記為 User LED)
2. 確認其模式為 **GPIO_Output** (應已預設)
3. 若未設定，右鍵點擊 PA5 → 選擇 **GPIO_Output**
4. 在左側 **System Core** 中選擇 **GPIO**
5. 展開 **GPIOA** 確認 PA5 的設定：
    - **GPIO output level**: High
    - **GPIO mode**: Output Push-Pull
    - **Pull**: No pull
    - **Maximum output speed**: High

時脈配置（**Clock Tree**）：

1. 切換到 **Clock Configuration** 標籤
2. 確認以下設定：
    - **HSE (High Speed External)**: 8 MHz (開發板晶振頻率)
    - **System Clock Multiplier**: 配置為 **168 MHz**（F446RC 最大頻率)
    - **AHB Prescaler**: 1 (不分頻)
3. 查看底部確認無警告訊息

SysTick 配置：

1. 在左側選擇 **SysTick**
2. 確認 **Timebase** 設為 **SysTick**
3. 此設定用於系統計時，後續課程會用到

?? 3??????

1. 點擊 **Project → Generate Code**
2. CubeMX 會根據配置產生初始化程式碼
3. 點擊 **Open Project** 返回 IDE

? ?????

main.c - Blink LED ??

```
/* STM32 Lesson 01 - Blink LED
 * 功能：使用 GPIO 驅動 PA5 (User LED)，實現 1 秒間隔的閃爍
 * 難度：初級
 */


#include "main.h"
```

```
#include "gpio.h"

/* 私有函數宣告 */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);

int main(void)
{
  /* 重置所有外設並初始化時脈 */
  HAL_Init();

  /* 配置系統時脈為 168 MHz */
  SystemClock_Config();

  /* 初始化 GPIO */
  MX_GPIO_Init();

  /* 主迴圈 */
  while (1)
  {
    /* 設置 PA5 為高電位（LED 點亮）*/
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);

    /* 延遲 500 ms */
    HAL_Delay(500);

    /* 設置 PA5 為低電位（LED 熄滅）*/
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

    /* 延遲 500 ms */
    HAL_Delay(500);
  }
}

/**
  * @brief 系統時脈配置函數
  * 配置主振盪器 (HSE) 倍頻到 168 MHz
  */
void SystemClock_Config(void)
{
```

```
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** 配置主內部調節器輸出電壓 */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** 初始化 RCC 振盪器 */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** 初始化 CPU、AHB 和 APB 匯流排時脈 */
  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_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO 初始化函數
  * 配置 PA5 為輸出模式
  */
```

```
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO 埠時脈使能 */
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /* 配置 GPIO 腳位 PA5 */
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}

/**
  * @brief 錯誤處理函數
  * 當系統配置失敗時調用
  */
void Error_Handler(void)
{
  while (1)
  {
    /* 無限迴圈，系統掛起 */
  }
}
```

? ?????

????

 □□□□□□

- 開發板上的綠色 LED 以 **1** 秒間隔（0.5秒亮 + 0.5秒暗）穩定閃爍
- IDE 的 Console 顯示「**Build successful**」
- 編譯無警告（warnings）

?????????

問題	原因	解決方案
編譯失敗「undefined reference to HAL_xxx」	HAL 函式庫未正確連結	確保 CubeMX 已生成程式碼，檢查 Project → Properties → C/C++ Build → Libraries
燒錄失敗「Failed to connect to target」	ST-Link 驅動未安裝或硬體連接不良	重新插拔 USB 線，檢查裝置管理員中的 ST-Link 設備
LED 不亮	GPIO 設定錯誤或接腳配置反向	檢查 CubeMX 中 PA5 是否設為 GPIO_Output，確認 GPIO_PIN_SET 為高電位
LED 常亮不閃爍	HAL_Delay 函數未工作	確認 SysTick 定時器已在 CubeMX 中配置

???????

1. 連接 Nucleo-F446RC 到電腦（USB Type-B）
2. 在 IDE 中點擊 **Project** 選擇您的專案
3. 點擊工具列上的 **Build** 按鈕（鍾子圖示）編譯
4. 等待編譯完成，確認無錯誤
5. 點擊 **Run** 按鈕（播放圖示）或按 **Ctrl+F11** 燒錄程式
6. 觀察開發板上的綠色 LED 是否開始閃爍

??/??????

□□ 如何驗證：

- 使用 邏輯分析儀 或 示波器 測量 PA5 腳位的信號
- 預期波形：方波，0V（LED 暗）↔ 3.3V（LED 亮），週期 1 秒

? ????

?? Debug ??

1. 點擊工具列上的 **Debug** 按鈕（蟲子圖示）進入調試模式
2. 程式會自動暫停在 `main()` 函數開始處
3. 按 **F6** 或點擊 **Step Over** 逐行執行
4. 觀察 **Variables** 視窗中變數的值變化
5. 按 **F8** 或點擊 **Resume** 繼續執行

??????

在調試模式中：

1. 將滑鼠懸停在程式碼中的變數上，會顯示當前值
2. 在 **Breakpoints** 中設定中斷點（點擊程式碼行號）
3. 程式執行到中斷點時自動暫停

? ????

???? - ? 2 ????????

第 2 節將深入探討：

- 振盪器與倍頻原理 - 為何 STM32F446 能達到 168 MHz
- **Clock Tree** 詳解 - 各外設的時脈源選擇
- 電源管理模式 - Sleep、Stop、Standby 模式及其應用

????????

1. 修改閃爍頻率 - 將 `HAL_Delay(500)` 改為不同值，觀察 LED 閃爍速度變化
2. 使用按鈕控制 **LED** - 結合 User Button（PC13），按下時 LED 點亮
3. 呼吸燈效果（預告） - 使用 PWM 調變亮度，實現漸亮漸暗效果

??????

- □□ STM32CubeIDE 官方文檔
- □□ STM32F446 資料表
- □□ HAL 函式庫參考手冊
- □□ Nucleo-F446RC 使用者手冊

□ □□□□□□□□ **1** 節，擁有完整的 **STM32** 開發環境！

下一步：進入第 2 節，學習時脈配置的深層原理 □□

STM32 ???? ? 2 ????????

? ????

1. 理解 **STM32F446** 的時脈架構 - 掌握振盪器、倍頻器、分頻器的概念
2. 學會配置 **Clock Tree** - 透過 CubeMX 設定各組態的時脈分配
3. 實現多時脈配置實驗 - 透過改變時脈速度觀察 LED 閃爍頻率變化

? ????????

????????

STM32 微控制器內部的每個元件（CPU、外設、計時器）都需要時脈信號來同步運作。時脈越快，處理速度越快，但功耗也越高。

STM32F446 ????

STM32F446 提供 **3** 種時脈源：

時脈源	頻率	精度	功耗	用途
HSI (內部高速振盪器**)**	16 MHz	±2%	低	備用時脈、低功耗模式
HSE (外部高速振盪器**)**	8 MHz	±1%	中等	主系統時脈（需晶振）
LSI (內部低速振盪器**)**	32 kHz	±2%	很低	看門狗、RTC

PLL ????

PLL (Phase-Locked Loop) 是一個電路，能將低頻時脈信號倍頻至高頻。例如：

- 輸入：8 MHz（HSE）
- 倍數：21
- 輸出：8 MHz × 21 = **168 MHz**

STM32F446 的最大系統時脈為 **180 MHz**，但實驗中常用 **168 MHz**（更穩定）。

?????

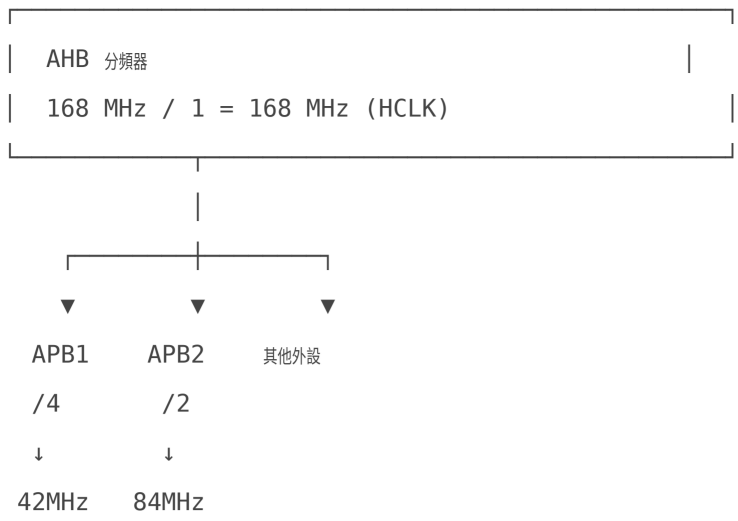
系統時脈（168 MHz）太快，不適合直接提供給所有外設。因此使用分頻器降低特定外設的時脈：

分頻器	功能	典型設定
AHB (HCLK)	CPU 和內存時脈	168 MHz / 1 = 168 MHz
APB1 (PCLK1)	低速外設（UART、I2C）	168 MHz / 4 = 42 MHz
APB2 (PCLK2)	高速外設（SPI、ADC）	168 MHz / 2 = 84 MHz

?? Clock Tree ????

Clock Tree ????

```
┌─────────────────────────────────────────────────┐
│  時脈源選擇                                      │
│  ├─ HSI (16 MHz) - 內部振盪器                    │
│  ├─ HSE (8 MHz)  - 外部晶振                      │
│  └─ LSI (32 kHz) - 內部低速振盪器                │
└─────────────────────────────────────────────────┘
             │
             │
             ▼
┌─────────────────────────────────────────────────┐
│  PLL 倍頻（選擇性）                              │
│  主要參數：PLLM / PLLN / PLLP / PLLQ             │
│  目標：8 MHz → 168 MHz                           │
└─────────────────────────────────────────────────┘
             │
             │
             ▼
┌─────────────────────────────────────────────────┐
│  系統時脈選擇（SYSCLK）                          │
│  ├─ HSI (16 MHz)                                 │
│  ├─ HSE (8 MHz)                                  │
│  └─ PLL (168 MHz)   ← 常用                       │
└─────────────────────────────────────────────────┘
             │
             │
             ▼
```



PLL ????

STM32F446 PLL 配置公式:

```
f_VCO = (f_HSE / PLLM) × PLLN
f_PLLCLK = f_VCO / PLLP
f_USB = f_VCO / PLLQ
```

常用配置（ **HSE = 8 MHz**）：

```
PLLM = 8      (預分頻: 8 MHz / 8 = 1 MHz)
PLLN = 168    (倍頻: 1 MHz × 168 = 168 MHz)
PLLP = 2      (主時脈分頻: 168 MHz / 2 = 84 MHz) □ □□□ 84 MHz
PLLP = 2      (若要 168 MHz, 應設定 PLLN = 336, PLLP = 2)
```

更精確的配置（目標 **168 MHz**）：

```
PLLM = 8      (預分頻: 8 MHz / 8 = 1 MHz)
PLLN = 336    (倍頻: 1 MHz × 336 = 336 MHz)
PLLP = 2      (主時脈分頻: 336 MHz / 2 = 168 MHz) □
```

?? CubeMX ??????

?? 1?????????

1. 開啟之前的專案或建立新專案

2. 雙擊 `.ioc` 檔案開啟 CubeMX
3. 點擊 **Clock Configuration** 標籤

?? 2??? HSE?8 MHz ???

1. 在「System Clock Mux」右側, 點擊 **HSE** 下拉選單
2. 選擇 **Crystal/Ceramic Resonator**
3. 確認 HSE 頻率顯示為 **8 MHz**

?? 3??? PLL

1. 找到 **PLL** 配置區塊
2. 設定以下參數:
    - **PLLM**: 8
    - **PLLN**: 336
    - **PLLP**: 2 (對應頻率應顯示 168 MHz)
    - **PLLQ**: 7 (用於 USB, 通常保持預設)
3. 若 PLLN 改為 336 後系統時脈仍未達 168 MHz, 檢查 PLLP 是否為 2

?? 4??? PLL ???????

1. 在「System Clock Mux」中, 選擇 **PLLCLK**
2. 確認 **SYSCLK** 顯示 **168 MHz**

?? 5??? AHB ???

1. 在「System Clock Mux」下方, 找到 **AHB Prescaler**
2. 設定為 **1** (不分頻, HCLK = 168 MHz)

?? 6??? APB ???

1. **APB1 Prescaler** 設為 **4**
    - 結果: 168 MHz / 4 = **42 MHz** (適合 UART、I2C)
2. **APB2 Prescaler** 設為 **2**
    - 結果: 168 MHz / 2 = **84 MHz** (適合 SPI、ADC)

?? 7?????

- □ □□ CubeMX 下方的總結, 應顯示:

```
SYSCLK:  168 MHz
HCLK:    168 MHz
PCLK1:    42 MHz
PCLK2:    84 MHz
```

? ????? - ??????

main.c - ???????

```c
/* STM32 Lesson 02 - Clock Configuration Verification
 * 功能: 測試不同的時脈配置，透過 LED 閃爍頻率變化來驗證
 * 難度: 中級
 */

#include "main.h"
#include "gpio.h"

/* 私有變數和函數 */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
void LED_Blink(uint16_t delay_ms, uint8_t count);

/* 全域變數: 存儲當前系統時脈速度 */
uint32_t SystemCoreClock = 168000000;  /* 168 MHz */

int main(void)
{
  HAL_Init();

  /* 配置系統時脈為 168 MHz */
  SystemClock_Config();

  /* 初始化 GPIO */
  MX_GPIO_Init();

  /* 驗證系統時脈 */
```

```
  while (1)
  {
    /* 第一階段: 快速閃爍 (5 次, 100ms 間隔) - 表示 168 MHz */
    LED_Blink(100, 5);
    HAL_Delay(1000);  /* 暫停 1 秒 */

    /* 第二階段: 中速閃爍 (3 次, 200ms 間隔) - 表示配置完成 */
    LED_Blink(200, 3);
    HAL_Delay(1000);

    /* 第三階段: 慢速閃爍 (2 次, 500ms 間隔) - 表示系統運行 */
    LED_Blink(500, 2);
    HAL_Delay(2000);
  }
}

/**
  * @brief LED 間隔閃爍函數
  * @param delay_ms: 每次亮/暗的延遲時間 (毫秒)
  * @param count: 閃爍次數
  */
void LED_Blink(uint16_t delay_ms, uint8_t count)
{
  for (uint8_t i = 0; i < count; i++)
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);    /* LED 亮 */
    HAL_Delay(delay_ms);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);  /* LED 暗 */
    HAL_Delay(delay_ms);
  }
}

/**
  * @brief 系統時脈配置函數
  * 配置 HSE (8 MHz) → PLL 倍頻至 168 MHz
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
```

```
  /* 配置電源調節器 */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /* 配置 RCC 振盪器 */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;      /* 預分頻: 8 MHz / 8 = 1 MHz */
  RCC_OscInitStruct.PLL.PLLN = 336;    /* 倍頻: 1 MHz × 336 = 336 MHz */
  RCC_OscInitStruct.PLL.PLLP = 2;      /* 主輸出分頻: 336 MHz / 2 = 168 MHz */
  RCC_OscInitStruct.PLL.PLLQ = 7;      /* USB 時脈: 336 MHz / 7 = 48 MHz */

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /* 配置時脈分頻器 */
  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;      /* HCLK = 168 MHz */
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;       /* PCLK1 = 42 MHz */
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;       /* PCLK2 = 84 MHz */

  /* Flash 延遲設定（高速時脈需要更多等待週期） */
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }

  /* 配置 SysTick 用於系統計時 */
  HAL_SYSTICK_Config(SystemCoreClock / 1000);  /* 1ms 中斷一次 */
}

/**
  * @brief GPIO 初始化函數
```

```
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  __HAL_RCC_GPIOA_CLK_ENABLE();

  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}

/**
  * @brief 系統錯誤處理
  */
void Error_Handler(void)
{
  while (1)
  {
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);  /* 快速閃爍表示錯誤 */
    HAL_Delay(100);
  }
}
```

? ?????

????

 LED 閃爍模式序列 :

1. **5** 次快速閃爍 （100ms） → 系統時脈 168 MHz 正常
2. 暫停 **1** 秒
3. **3** 次中速閃爍 （200ms） → 時脈配置完成
4. 暫停 **1** 秒
5. **2** 次慢速閃爍 （500ms） → 系統穩定運行
6. 暫停 **2** 秒後重複

???????

問題	原因	解決方案
LED 不閃爍	PLL 配置失敗	檢查 CubeMX Clock Configuration，確認 PLLN 為 336，PLLP 為 2
閃爍節奏不對	HAL_Delay 計算錯誤	確認 SysTick 配置正確，系統時脈應為 168 MHz
編譯警告「implicit conversion」	時脈值類型不匹配	確認 `SystemCoreClock` 變數型別為 `uint32_t`
程式掛起	Flash 延遲不足	檢查 `FLASH_LATENCY_5` 是否設定，高速時脈需更多等待週期

??????

方法 **1**：**UART** 輸出驗證（後續課程）透過串口列印 `SystemCoreClock` 值

方法 **2**：邏輯分析儀 測量 PA5 的方波頻率，應為固定值

方法 **3**：計算驗證

- 若 `HAL_Delay(100)` 實際延遲 100ms，時脈正確 □

? ????

Flash ?????FLASH_LATENCY?

快閃記憶體存取速度是有限的。當系統時脈增加時，需要增加等待週期（Latency）。

系統時脈	建議 **LATENCY**
0-30 MHz	0
31-60 MHz	1
61-90 MHz	2
91-120 MHz	3
121-150 MHz	4
151-168 MHz	5

? ????

?? 1????????

修改代碼, 使用 **HSI (16 MHz)** 而非 PLL:

```
/* 使用 HSI 代替 PLL (低功耗模式) */
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
RCC_OscInitStruct.HSIState = RCC_HSI_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_OFF;  /* 關閉 PLL */

RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
```

觀察 LED 閃爍速度是否變慢 (因為系統時脈從 168 MHz 降至 16 MHz)

?? 2???????

實現在運行時在不同時脈間切換的函數

? ????

???? - ? 3 ??GPIO ????????????

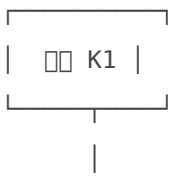
第 3 節將學習:

- **GPIO** 輸入配置 - 按鍵檢測原理
- 中斷系統 (**EXTI**) - 事件驅動編程
- 軟體去彈跳 - 解決按鍵抖動問題

????

- □□ STM32F446 Reference Manual - RCC章節
- □□ PLL 倍頻計算器 (線上工具)

□ □□□□□□□□□□□□□ **GPIO** 進階應用 □□



STM32 ???? ? 3 ??GPIO ?????????????

? ????

1. 掌握 **GPIO** 輸入配置 - 實現按鍵檢測功能
2. 理解外部中斷系統（**EXTI**） - 學會事件驅動編程模式
3. 實現按鍵去彈跳算法 - 透過軟體解決機械按鍵的抖動問題

? GPIO ????

GPIO ????

GPIO（通用輸入/輸出）支援多種工作模式:

模式	描述	用途
GPIO_MODE_OUTPUT_PP	開路輸出（Push-Pull）	驅動 LED、控制繼電器
GPIO_MODE_INPUT	浮空輸入	一般輸入（需注意雜訊）
GPIO_MODE_INPUT_PU	上拉輸入	按鍵檢測（推薦）
GPIO_MODE_INPUT_PD	下拉輸入	特殊應用
GPIO_MODE_AF_PP	複用功能	UART、SPI 等外設
GPIO_MODE_IT_FALLING	下降邊緣中斷	按鍵按下時觸發

??????

標準的按鍵接線方式:

```
┌──────────┐
│  按鍵 K1 │
└──────────┘
     │
     │
```

```
        ┌─────┴─────┐
        │ (按下時接通)│
        │           │
       GND         GPIO_PIN_XX (上拉狀態)

狀態說明:
- 按鍵未按下: GPIO = 高電位 (3.3V)  → GPIO_PIN_SET
- 按鍵按下:   GPIO = 低電位 (0V)    → GPIO_PIN_RESET
```

?????Bouncing???

機械按鍵存在物理特性：按下或鬆開時，接點會產生多次快速的通/斷切換，稱為「抖動」。

```
無抖動情況（理想）：
GPIO ┌─────┐
     └─────┘

有抖動情況（真實）：
GPIO ┌──┐ ┌───┐ ┌──┐
     └──┘─┘   └─┘  └─ (多次跳變)

後果：會觸發多次中斷，導致計數錯誤
```

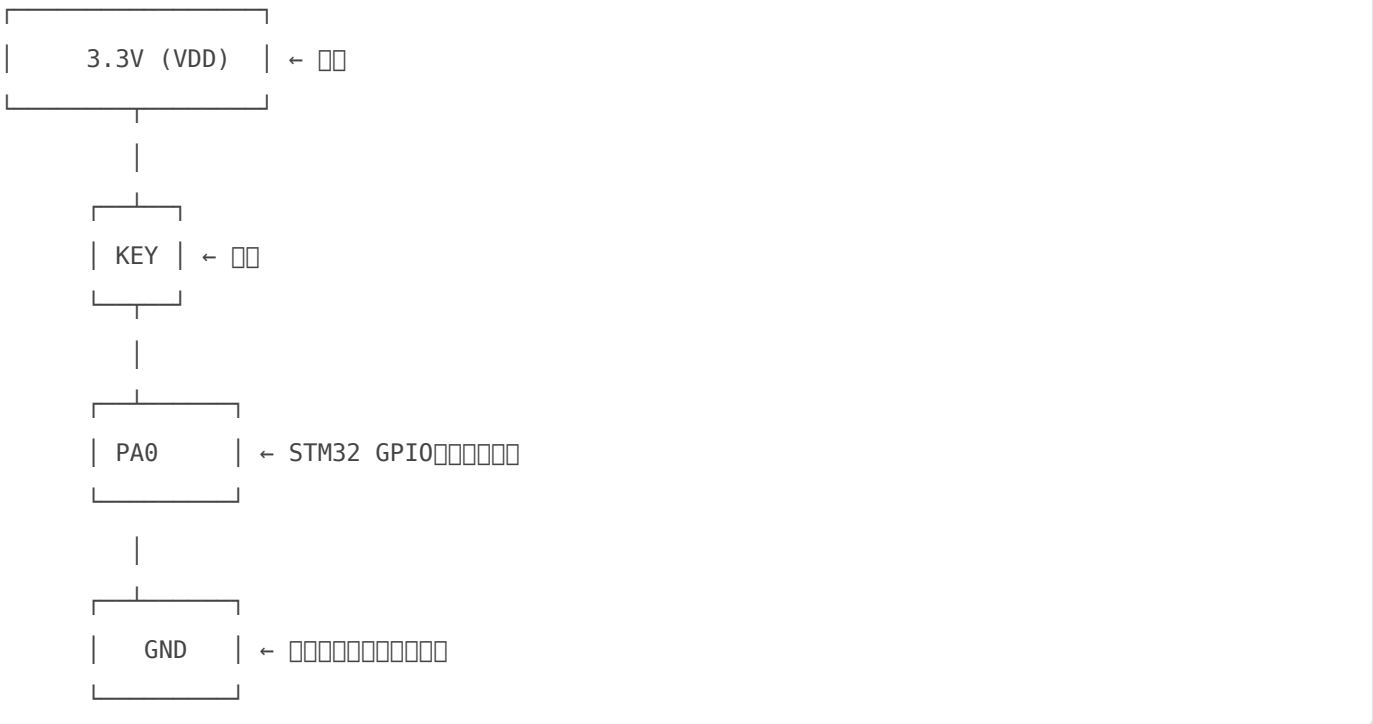
?????

方法	優點	缺點	成本
軟體去彈跳	無需硬體	佔用 CPU 時間	低
硬體濾波	無需軟體	需增加 RC 電路	中
中斷 + 計時器	精確	程式複雜	中

本課程採用 軟體去彈跳（最常用）。

?? ????

????



元件	數量	說明
按鈕開關 (Push Button)	1	常開型
220Ω 限流電阻	1	保護 GPIO (選配)
跳線	2	連接按鍵和 GPIO

???

組件	接腳	連接說明
按鈕 A 端	3.3V	一端連接電源
按鈕 B 端	PA0	另一端連接 GPIO (上拉)
PA0	GND	經過上拉電阻到地
LED	PA5	用於指示按鍵狀態 (可選)

CubeMX 配置 PA0:

- **GPIO mode**: Input
- **Pull**: Pull-up (上拉)
- **Label**: KEY_Input

???????

```
┌─────────────────┐
│    3.3V (VDD)   │ ← 電源
└────────┬────────┘
         │
         │
      ┌──┴──┐
      │ KEY │ ← 按鍵
      └──┬──┘
         │
         │
      ┌──┴──────┐
      │ PA0     │ ← STM32 GPIO (上拉狀態)
      └─────────┘
         │
         │
      ┌──┴──────┐
      │   GND   │ ← 地線 (內部上拉已連接)
      └─────────┘
```

?? CubeMX ????

?? 1?GPIO ????

1. 開啟 CubeMX, 在 Pinout 圖上找到 **PA0**
2. 右鍵點擊 PA0, 選擇 **GPIO_Input**
3. 在左側 **System Core** 中選擇 **GPIO**
4. 展開 **GPIOA** 並確認 PA0 的設定:
   - **GPIO mode**: Input
   - **Pull**: Pull-up
   - **Speed**: Medium
   - **User Label**: KEY_Input

?? 2????????EXTI?

1. 返回 Pinout 圖, 右鍵點擊 **PA0**
2. 選擇 **GPIO_EXTI0** (改為中斷模式)
3. 在左側選擇 **System Core → NVIC**
4. 找到 **EXTI line0 interrupt** 並勾選 **Enabled**
5. 設定優先級:
   - **Preemption Priority**: 0
   - **Sub Priority**: 0

?? 3??????????? 2 ????

確保 SysTick 定時器已啟用 (用於 HAL_Delay)

?? 4??????

點擊 **Project → Generate Code**

? ?????

main.c - ???? + ???

```
/* STM32 Lesson 03 - Key Press with Debouncing
 * 功能: 透過按鍵控制 LED, 使用軟體去彈跳技術
 * 難度: 中級
```

```
 */

#include "main.h"
#include "gpio.h"

/* 私有函數宣告 */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin);
void Debounce_Key(void);

/* 去彈跳參數 */
#define DEBOUNCE_DELAY_MS  20   /* 去彈跳延遲時間（毫秒） */
#define DEBOUNCE_SAMPLES   3    /* 採樣次數 */

/* 全域變數 */
volatile uint8_t key_pressed = 0;        /* 按鍵按下標誌 */
volatile uint32_t last_interrupt_time = 0;  /* 上次中斷時間 */
uint8_t led_state = 0;                   /* LED 狀態: 0=暗, 1=亮 */

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

  /* 初始狀態: LED 熄滅 */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
  led_state = 0;

  while (1)
  {
    /* 軟體去彈跳檢測 */
    if (key_pressed)
    {
      Debounce_Key();
      key_pressed = 0;  /* 清除標誌 */
    }

    /* 主程式邏輯（可添加其他功能） */
```

```c
  }
}

/**
  * @brief GPIO 外部中斷回調函數
  * 當檢測到下降邊緣時自動調用
  * @param GPIO_Pin: 觸發中斷的引腳
  */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin == GPIO_PIN_0)
  {
    /* 中斷防抖：檢查距離上次中斷是否超過 50ms */
    uint32_t current_time = HAL_GetTick();

    if (current_time - last_interrupt_time > 50)
    {
      key_pressed = 1;  /* 設置標誌，在主迴圈中處理 */
      last_interrupt_time = current_time;
    }
  }
}

/**
  * @brief 軟體去彈跳檢測函數
  * 採用多次採樣驗證按鍵狀態
  */
void Debounce_Key(void)
{
  uint8_t stable_count = 0;

  /* 進行多次採樣 */
  for (uint8_t i = 0; i < DEBOUNCE_SAMPLES; i++)
  {
    /* 讀取按鍵狀態 */
    GPIO_PinState key_state = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);

    /* 若狀態為低電位（按鍵按下），計數器增加 */
    if (key_state == GPIO_PIN_RESET)
    {
```

```
      stable_count++;
    }

    /* 每次採樣間隔 */
    HAL_Delay(DEBOUNCE_DELAY_MS);
  }

  /* 若所有採樣都檢測到低電位，則確認按鍵按下 */
  if (stable_count == DEBOUNCE_SAMPLES)
  {
    /* 切換 LED 狀態 */
    led_state = !led_state;
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, led_state ? GPIO_PIN_SET : GPIO_PIN_RESET);
  }
}

/**
  * @brief 系統時脈配置（同第 2 節）
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 7;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  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_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO 初始化函數
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO 埠時脈使能 */
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /* 配置 PA5（LED 輸出） */
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* 配置 PA0（按鍵輸入 + 中斷） */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;  /* 下降邊緣中斷 */
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* 啟用 EXTI0 中斷 */
  HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
```

```
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);
}

/**
  * @brief 外部中斷 0 服務程式
  */
void EXTI0_IRQHandler(void)
{
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
}

/**
  * @brief 錯誤處理
  */
void Error_Handler(void)
{
  while (1);
}
```

? ?????

????

□ □□□□□

1. 按開發板上的 **User Button**（**PC13**）或自接按鍵
2. 第一次按下: LED 點亮
3. 第二次按下: LED 熄滅
4. 重複按下實現 LED 開關切換

⚠ 無抖動現象：LED 不會因為按鍵抖動而閃爍

???????

問題	原因	解決方案
按鍵無反應	中斷未啟用或 GPIO 配置錯誤	檢查 CubeMX 中 EXTI0_IRQn 是否 Enabled
LED 多次切換	去彈跳延遲過短	增加 `DEBOUNCE_DELAY_MS` 至 30-50ms

問題	原因	解決方案
長按持續切換	中斷處理邏輯錯誤	確認 `key_pressed` 標誌已清除
編譯錯誤「EXTI0_IRQHandler 重定義」	自定義中斷處理與HAL衝突	確保中斷處理只定義一次

????????

進階除錯方法: 在Debug模式中查看按鍵電位值

```
/* 在中斷回調中添加診斷程式碼 */
GPIO_PinState key_state = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);
// key_state 應為 GPIO_PIN_RESET (0) = 按下
//          或 GPIO_PIN_SET (1) = 未按
```

? ????

???????

STM32支援優先級分組（Priority Grouping）:

- **Preemption Priority**（搶佔優先級）：數值小優先級高
  - 優先級0 > 優先級1 > 優先級2 ...
  - 高優先級中斷可中斷低優先級中斷執行
- **Sub Priority**（子優先級）：同搶佔優先級下的排隊順序

```
/* 設定最高優先級 */
HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);  /* 搶佔=0, 子=0 */
```

????????

除了多次採樣，也可用時間判斷:

```
#define ANTI_BOUNCE_TIME 50  /* 50ms 防抖時間 */

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  static uint32_t last_time = 0;
  uint32_t now = HAL_GetTick();
```

```
  /* 若距離上次觸發 > 50ms，才確認有效按下 */
  if (now - last_time > ANTI_BOUNCE_TIME)
  {
    key_pressed = 1;
    last_time = now;
  }
}
```

? ????

?? 1?????

修改代碼，使每次按下按鍵時透過 UART 列印計數值（後續課程 UART 後實現）

?? 2?????

偵測按鍵是否長按（超過 2 秒）並執行不同操作

```
void Handle_Long_Press(void)
{
  uint32_t press_time = HAL_GetTick();

  while (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_PIN_RESET)
  {
    if (HAL_GetTick() - press_time > 2000)  /* 2 秒 */
    {
      /* 長按邏輯 */
      return;
    }
  }
}
```

?? 3??????

擴展代碼支援多個按鍵，每個按鍵對應不同功能

? ????

???? - ? 4 ??UART ???? + ????

第 4 節將實現:

- **UART** 配置與初始化
- 收發單個字元與字串
- 建立簡單命令解析系統

????

- □□ STM32F446 GPIO and EXTI Reference
- □□ □□□□□□□□□□□□□□

□ **GPIO** 進階完成! 下節深入通訊協議 **- UART □□**

STM32 ???? ? 4 ??UART ???? + ????

? ????

1. 掌握 **UART** 通訊協定 - 理解串列通訊的原理與配置
2. 實現收發功能 - 透過電腦與 STM32 交互通訊
3. 建立命令介面 - 實現簡單的命令解析系統

? UART ????

UART ????

UART (Universal Asynchronous Receiver/Transmitter) 是最常見的串列通訊協定。特點:

- 異步通訊：發送端和接收端不需同步時脈
- 全雙工：可同時收發
- 點對點：通常用於連接兩個設備

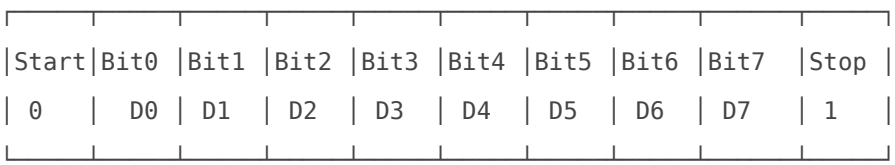
UART ???

STM32F446 UART 最少需要 3 條線:

訊號	功能	連接
TX (USART1_TX)	傳輸	PA9
RX (USART1_RX)	接收	PA10
GND	地線參考	GND

????????

參數	典型值	說明
Baud Rate	9600 / 115200	每秒傳輸位元數



參數	典型值	說明
Data Bits	8	每個字符包含 8 位資料
Stop Bits	1	停止位
Parity	None	無奇偶校驗

UART ?????

```
1 個 UART 字符的構成:

┌─────┬─────┬────┬────┬────┬────┬────┬────┬─────┬────┐
│Start│Bit0 │Bit1 │Bit2 │Bit3 │Bit4 │Bit5 │Bit6 │Bit7  │Stop │
│ 0   │  D0 │ D1 │ D2 │ D3 │ D4 │ D5 │ D6 │ D7  │ 1  │
└─────┴─────┴────┴────┴────┴────┴────┴────┴─────┴────┘
```

?? ????

????

元件	數量	說明
USB-UART 轉換器	1	PL2303 或 CP2102 晶片
USB Type-A 傳輸線	1	連接電腦
杜邦線	3	連接 STM32 和轉換器

???

STM32	**USB-UART**	說明
PA9 (TX)	RX	STM32 發送給電腦
PA10 (RX)	TX	STM32 接收來自電腦
GND	GND	共同地線

> ⚠ 重要 - 開發板已內建 ST-Link 調試器, 可透過 USB 進行 UART 通訊, 無需額外轉換器!

?? CubeMX ????

?? 1??? USART1

1. 開啟 CubeMX
2. 在 Pinout 圖上找到 PA9 和 PA10
3. 分別配置為:
   - PA9: **USART1_TX**
   - PA10: **USART1_RX**
4. 或直接點擊 Connectivity → **USART1** 自動配置

?? 2??? USART1 ??

1. 在左側選擇 **Connectivity → USART1**
2. 設定以下參數:
   - **Baud Rate**: 115200
   - **Word Length**: 8 Bits
   - **Stop Bits**: 1
   - **Parity**: None
   - **Mode**: Asynchronous (RX and TX)

?? 3??? USART1 ??

1. 點擊 **NVIC Settings** 標籤
2. 勾選 **USART1 global interrupt**
3. 設定優先級:
   - **Preemption Priority**: 1
   - **Sub Priority**: 0

?? 4??????

點擊 **Generate Code**

? ?????

main.c - UART + ????

```
/* STM32 Lesson 04 - UART Communication with Command Interface
 * 功能: 透過 UART 接收命令並控制 LED
 * 命令:
 *   "ON"  - LED 點亮
 *   "OFF" - LED 熄滅
 *   "TOGGLE" - LED 切換
 * 難度: 中級
 */

#include "main.h"
#include "usart.h"
#include "gpio.h"
#include <string.h>
#include <stdio.h>

/* 私有定義 */
#define UART_RX_BUFFER_SIZE 50
#define COMMAND_MAX_LEN 20

/* 函數宣告 */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);
void UART_Print(const char *format, ...);
void Process_Command(char *command);
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart);

/* 全域變數 */
UART_HandleTypeDef huart1;
uint8_t uart_rx_buffer[UART_RX_BUFFER_SIZE];
uint8_t uart_rx_index = 0;
char command_buffer[COMMAND_MAX_LEN];

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART1_UART_Init();
```

```
  /* 初始化 LED 為熄滅狀態 */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

  /* 列印歡迎訊息 */
  UART_Print("\r\n==============================\r\n");
  UART_Print("STM32 UART Command Interface\r\n");
  UART_Print("Commands: ON, OFF, TOGGLE\r\n");
  UART_Print("==============================\r\n");

  /* 啟用 UART 中斷接收 */
  HAL_UART_Receive_IT(&huart1, uart_rx_buffer, 1);

  while (1)
  {
    /* 主迴圈 */
  }
}

/**
  * @brief UART 傳輸函數（printf 風格）
  */
void UART_Print(const char *format, ...)
{
  char buffer[100];
  va_list args;
  va_start(args, format);
  vsnprintf(buffer, sizeof(buffer), format, args);
  va_end(args);

  HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);
}

/**
  * @brief UART 接收完成中斷回調
  */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  if (huart->Instance == USART1)
  {
    uint8_t received_char = uart_rx_buffer[0];
```

```
    /* 處理回車符（\r）或換行符（\n） */
    if (received_char == '\r' || received_char == '\n')
    {
      if (uart_rx_index > 0)
      {
        command_buffer[uart_rx_index] = '\0';
        UART_Print("\r\nReceived: %s\r\n", command_buffer);
        Process_Command(command_buffer);
        uart_rx_index = 0;
      }
    }
    else if (received_char == 0x08 || received_char == 0x7F)  /* Backspace */
    {
      if (uart_rx_index > 0)
      {
        uart_rx_index--;
        UART_Print("\b \b");  /* 刪除顯示 */
      }
    }
    else
    {
      /* 一般字符 */
      if (uart_rx_index < COMMAND_MAX_LEN - 1)
      {
        command_buffer[uart_rx_index++] = received_char;
        HAL_UART_Transmit(&huart1, (uint8_t *)&received_char, 1, HAL_MAX_DELAY);  /* Echo */
      }
    }

    /* 繼續接收下一個字符 */
    HAL_UART_Receive_IT(&huart1, uart_rx_buffer, 1);
  }
}

/**
  * @brief 命令處理函數
  */
void Process_Command(char *command)
{
```

```
  /* 轉換為大寫以便比較 */
  for (int i = 0; command[i]; i++)
  {
    if (command[i] >= 'a' && command[i] <= 'z')
      command[i] -= 32;
  }

  /* 命令解析 */
  if (strcmp(command, "ON") == 0)
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
    UART_Print("LED ON\r\n");
  }
  else if (strcmp(command, "OFF") == 0)
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
    UART_Print("LED OFF\r\n");
  }
  else if (strcmp(command, "TOGGLE") == 0)
  {
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    UART_Print("LED TOGGLED\r\n");
  }
  else
  {
    UART_Print("Unknown command. Try: ON, OFF, TOGGLE\r\n");
  }

  UART_Print("> ");  /* 命令提示符 */
}

/**
  * @brief 系統時脈配置
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
```

```
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 7;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  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_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO □□□
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  __HAL_RCC_GPIOA_CLK_ENABLE();

  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
```

```
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}

/**
  * @brief USART1 □□□
  */
static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;

  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }

  __HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE);
}

/**
  * @brief UART MSP □□□
  */
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  if(huart->Instance == USART1)
  {
    __HAL_RCC_USART1_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();

    GPIO_InitStruct.Pin = GPIO_PIN_9 | GPIO_PIN_10;
```

```
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF7_USART1;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    HAL_NVIC_SetPriority(USART1_IRQn, 1, 0);
    HAL_NVIC_EnableIRQ(USART1_IRQn);
  }
}

/**
  * @brief USART1 中斷處理
  */
void USART1_IRQHandler(void)
{
  HAL_UART_IRQHandler(&huart1);
}

void Error_Handler(void)
{
  while(1);
}
```

? ?????

????

□ □□□□□

1. 編譯並燒錄程式
2. 開啟終端機軟體（ PuTTY、Arduino IDE 或 Tera Term）
3. 連接 COM 埠，波特率 115200
4. 應看到歡迎訊息: `STM32 UART Command Interface`
5. 輸入 `ON` → LED 點亮，列印 `LED ON`
6. 輸入 `OFF` → LED 熄滅，列印 `LED OFF`
7. 輸入 `TOGGLE` → LED 切換狀態

????

問題	原因	解決方案
無訊息輸出	UART 未初始化或波特率不對	檢查 CubeMX 中 USART1 設定，確認波特率為 115200
亂碼	波特率不匹配	改變終端機波特率，通常試試 9600 或 115200
無法接收命令	中斷未啟用	確認 `HAL_UART_Receive_IT()` 已調用
命令無反應	命令格式不正確	確保輸入 `ON`、`OFF` 或 `TOGGLE`（大小寫會自動轉換）

? ????

???????

優化版本使用圓形緩衝區防止緩衝區溢出

DMA + UART

使用 DMA 進行高效率的 UART 傳輸（後續課程）

? ????

???? - ? 5 ??ADC???????????? channel?

第 5 節將實現：

- **ADC** 單次轉換與掃描模式
- 軟體觸發與計時器觸發
- 多通道採樣

□ UART □□□□□□□□□

STM32 ???? ? 5 ??ADC ????????????? Channel?

? ????

1. 理解 **ADC** 工作原理 - 掌握模數轉換的概念
2. 學會配置不同轉換模式 - 單次、掃描、連續轉換
3. 實現多通道採樣 - 透過 UART 輸出感測器數值

? ADC ????

ADC ????

ADC (Analog-to-Digital Converter) 將類比電壓信號轉換為數位值。STM32F446 內建 **3** 個 **ADC**, 共 **16** 個通道。

STM32F446 ADC ??

特性	規格
分辨率	12 位（0-4095）
轉換時間	~3.5 微秒
採樣率	最高 2.4 MSPS（百萬次/秒）
通道數	19 個（16 外部 + 3 內部）
參考電壓	3.3V（VREF+）

ADC ????

通道	**GPIO** 接腳	用途
ADC1_IN0	PA0	外部感測器輸入 1
ADC1_IN1	PA1	外部感測器輸入 2

通道	**GPIO** 接腳	用途
ADC1_IN2	PA2	外部感測器輸入 3
ADC1_IN3	PA3	外部感測器輸入 4
ADC1_IN16	-	溫度感測器（內部）
ADC1_IN17	-	VREF+ 參考電壓
ADC1_IN18	-	VBAT（電池電壓）

??????????

```
類比電壓 (V)            數位值 (12-bit)
      ↑                       ↑
3.3V ├──────────────────  4095
2.5V ├──────────────────  3077 (約 2.5V/3.3V * 4095)
1.65V├──────────────────  2048
0.0V ├──────────────────  0
      └──────────────────   └────────
```

計算公式:

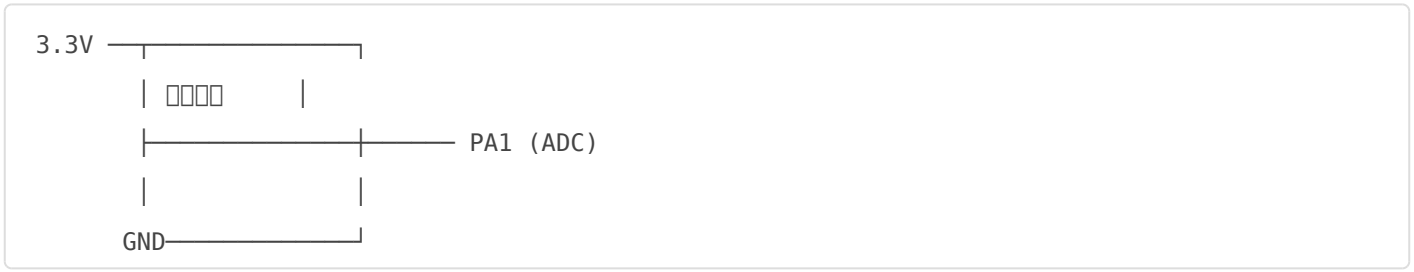
```
ADC_Value = (V_input / V_ref) × 2^Resolution - 1
          = (V_input / 3.3) × 4095


反推:
V_input = (ADC_Value / 4095) × 3.3
```

ADC ????

模式	說明	應用
Single	單次轉換一個通道	偶發採樣
Continuous	連續自動轉換	實時監測
Scan	依序轉換多個通道	多感測器
DMA	直接記憶體存取	高速大量採樣

?? ????



????

元件	數量	說明
可變電阻（10kΩ)	1	提供類比電壓輸入
鱷魚夾	3	接線用
跳線	適量	麵包板接線

???

組件	接腳	說明
可變電阻中心腳	PA1 (ADC1_IN1)	類比輸入
可變電阻一端	3.3V	電源
可變電阻另一端	GND	地線

```
3.3V ─┬──────────────┐
       │ 可變電阻      │
       ├──────────────┼────── PA1 (ADC)
       │              │
      GND─────────────┘
```

?? CubeMX ????

?? 1??? ADC1

1. 開啟 CubeMX
2. 點擊 **Analog** → **ADC1**
3. 在 Pinout 圖中選擇 **PA1** 配置為 **ADC1_IN1**

?? 2??? ADC ??

1. 在左側選擇 **Analog** → **ADC1**
2. 在 **Parameter Settings** 中設定:
    - **Resolution**: 12 Bits
    - **Data Alignment**: Right Alignment
    - **Scan Conversion Mode**: Disable (單通道)
    - **Continuous Conversion Mode**: Enable (連續轉換)

- **EOC Selection**: End of conversion flag

3. 在 **Rank** 區域添加通道:
   - 點擊 **Add** 或 **Rank 1**
   - 選擇 **Channel**: ADC_CHANNEL_1
   - **Sampling Time**: 144 Cycles (較穩定)

?? 3??? ADC ??

1. 點擊 **NVIC Settings** 標籤
2. 勾選 **ADC1 global interrupt**
3. 設定優先級:
   - **Preemption Priority**: 2
   - **Sub Priority**: 0

?? 4?????????????

若要定時採樣, 可配置 Timer 觸發 ADC:

1. 選擇 **Timers → TIM2**
2. 設定 **Trigger Output Event**: Update Event
3. 返回 ADC1, 設置:
   - **Trigger**: Timer2 Trigger Out Event
   - **External Trigger Conversion Edge**: Rising Edge

?? 5??????

點擊 **Generate Code**

? ?????

main.c - ADC ??????????

```
/* STM32 Lesson 05 - ADC Continuous Sampling
 * 功能: 連續採樣類比輸入, 透過 UART 輸出結果
 * 難度: 中級
 */

#include "main.h"
#include "adc.h"
```

```c
#include "usart.h"
#include "gpio.h"
#include <stdio.h>
#include <stdarg.h>

/* 私有定義 */
#define ADC_SAMPLE_SIZE 10  /* 每次平均取樣數 */

/* 函數宣告 */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART1_UART_Init(void);
void UART_Print(const char *format, ...);
uint16_t ADC_Get_Average(void);
float ADC_To_Voltage(uint16_t adc_value);

/* 全域變數 */
ADC_HandleTypeDef hadc1;
UART_HandleTypeDef huart1;
volatile uint16_t adc_value = 0;
volatile uint8_t adc_conversion_complete = 0;

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART1_UART_Init();
  MX_ADC1_Init();

  UART_Print("\r\n======= ADC Sampling Test =======\r\n");
  UART_Print("ADC Value | Voltage (V)\r\n");
  UART_Print("==================================\r\n");

  /* 啟動 ADC 連續轉換 */
  HAL_ADC_Start_IT(&hadc1);

  while (1)
  {
```

```
    if (adc_conversion_complete)
    {
      /* 讀取 ADC 值 */
      uint16_t adc_raw = HAL_ADC_GetValue(&hadc1);

      /* 轉換為電壓 */
      float voltage = ADC_To_Voltage(adc_raw);

      /* 透過 UART 輸出 */
      UART_Print("%-9d | %.2f\r\n", adc_raw, voltage);

      adc_conversion_complete = 0;

      HAL_Delay(100);  /* 每 100ms 輸出一次 */
    }
  }
}

/**
  * @brief 將 ADC 原始值轉換為電壓
  * @param adc_value: 12-bit ADC 原始值 (0-4095)
  * @return 電壓值 (V)
  */
float ADC_To_Voltage(uint16_t adc_value)
{
  /* 參考電壓 3.3V, 12-bit 分辨率 4095 */
  return (adc_value / 4095.0f) * 3.3f;
}

/**
  * @brief 獲取多次採樣的平均值
  */
uint16_t ADC_Get_Average(void)
{
  uint32_t sum = 0;

  for (uint8_t i = 0; i < ADC_SAMPLE_SIZE; i++)
  {
    sum += HAL_ADC_GetValue(&hadc1);
    HAL_Delay(1);
```

```
  }

  return sum / ADC_SAMPLE_SIZE;
}

/**
  * @brief ADC 轉換完成中斷回調
  */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
  if (hadc->Instance == ADC1)
  {
    adc_conversion_complete = 1;
  }
}

/**
  * @brief UART 列印函數
  */
void UART_Print(const char *format, ...)
{
  char buffer[100];
  va_list args;
  va_start(args, format);
  vsnprintf(buffer, sizeof(buffer), format, args);
  va_end(args);

  HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);
}

/**
  * @brief 系統時脈配置
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
```

```
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 7;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  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_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO □□□
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  __HAL_RCC_GPIOA_CLK_ENABLE();

  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}

/**
  * @brief ADC1 □□□
  */
static void MX_ADC1_Init(void)
{
  ADC_ChannelConfTypeDef sConfig = {0};

  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = 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;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc1.Init.LowPowerAutoWait = DISABLE;
  hadc1.Init.LowPowerAutoPowerOff = DISABLE;

  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_144CYCLES;

  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
}
```

```
/**
  * @brief ADC MSP □□□
  */
void HAL_ADC_MspInit(ADC_HandleTypeDef* hadc)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  if(hadc->Instance == ADC1)
  {
    __HAL_RCC_ADC1_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();

    GPIO_InitStruct.Pin = GPIO_PIN_1;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    HAL_NVIC_SetPriority(ADC_IRQn, 2, 0);
    HAL_NVIC_EnableIRQ(ADC_IRQn);
  }
}

/**
  * @brief USART1 □□□
  */
static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;

  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
```

```
}

/**
  * @brief UART MSP □□□
  */
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  if(huart->Instance == USART1)
  {
    __HAL_RCC_USART1_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();

    GPIO_InitStruct.Pin = GPIO_PIN_9 | GPIO_PIN_10;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF7_USART1;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    HAL_NVIC_SetPriority(USART1_IRQn, 1, 0);
    HAL_NVIC_EnableIRQ(USART1_IRQn);
  }
}

/**
  * @brief ADC □□□□
  */
void ADC_IRQHandler(void)
{
  HAL_ADC_IRQHandler(&hadc1);
}

/**
  * @brief USART1 □□□□
  */
void USART1_IRQHandler(void)
{
  HAL_UART_IRQHandler(&huart1);
```

```
}

void Error_Handler(void)
{
  while(1);
}
```

? ?????

????

□ **ADC** 輸出示例:

```
======= ADC Sampling Test =======
ADC Value | Voltage (V)
==================================
2048      | 1.65
3072      | 2.47
1024      | 0.82
4095      | 3.30
```

轉動可變電阻, ADC 值應隨之變化, 電壓範圍應為 0.0V ~ 3.3V

????

問題	原因	解決方案
ADC 值不變	未啟動連續轉換	確認 `HAL_ADC_Start_IT()` 已調用
數值抖動大	採樣時間過短或雜訊干擾	增加 `SamplingTime` 至 144 Cycles
電壓計算不准	參考電壓設定錯誤	確認公式中 V_ref 為 3.3V
無 **UART** 輸出	UART 未初始化	檢查 USART1 初始化函數

? ????

???????

```
/* CubeMX 設定 */
hadc1.Init.ScanConvMode = ENABLE;  /* 啟用掃描模式 */
hadc1.Init.NbrOfConversion = 3;    /* 3 個通道 */

/* 配置多個通道 */
sConfig.Channel = ADC_CHANNEL_0;
sConfig.Rank = 1;
HAL_ADC_ConfigChannel(&hadc1, &sConfig);

sConfig.Channel = ADC_CHANNEL_1;
sConfig.Rank = 2;
HAL_ADC_ConfigChannel(&hadc1, &sConfig);

sConfig.Channel = ADC_CHANNEL_2;
sConfig.Rank = 3;
HAL_ADC_ConfigChannel(&hadc1, &sConfig);
```

?? DMA ??????

後續課程（第 7 節 DMA）將深入講解透過 DMA 進行高效率的 ADC 採樣

? ????

???? - ? 6 ????? Timer + PWM

第 6 節將實現:

- 定時器基本配置與中斷
- **PWM** 信號產生
- 呼吸燈效果實現

 ADC □□□□□□□□□

STM32 ???? ? 6 ????? Timer + PWM

? ????

1. 理解定時器工作原理 - 掌握計時與計數功能
2. 掌握 **PWM** 信號產生 - 實現 LED 亮度調控
3. 實現呼吸燈效果 - 漸亮漸暗的動態視覺效果

? ?????

???????

定時器是用來計時或計數的硬體模組。STM32F446 內建 **14** 個定時器，可用於:

- 時間測量和延遲
- PWM 信號產生
- 外部事件計數
- 週期中斷觸發

STM32F446 ?????

定時器	數量	位寬	功能
高級定時器 (TIM1, TIM8)	2	16-bit	PWM、死區時間、重複計數
通用定時器 (TIM2-5)	4	16/32-bit	PWM、計時、中斷
基本定時器 (TIM6-7)	2	16-bit	定時中斷、DMA 觸發
低功耗定時器 (TIM9-14)	6	16-bit	低功耗應用

本課程使用 **TIM2** (通用定時器) 作為示例。

PWM ????

PWM (Pulse Width Modulation) 是透過調整脈衝寬度來控制平均功率的技術。

```
佔空比 25%:

┌──┐          ┌──┐          ┌──┐
│  │──────────│  │──────────│  │
└──┘          └──┘          └──┘


佔空比 50%:

┌─────┐      ┌─────┐      ┌─────┐
│     │──────│     │──────│     │
└─────┘      └─────┘      └─────┘


佔空比 75%:

┌──────────┐  ┌──────────┐  ┌──────────┐
│          │──│          │──│          │
└──────────┘  └──────────┘  └──────────┘


週期 (T) 固定, 脈衝寬度 (W) 變化
佔空比 (Duty Cycle) = W / T × 100%
```

PWM ??

應用	說明
LED 亮度調控	改變佔空比調節亮度（0%-100%）
馬達速度控制	調整佔空比控制轉速
伺服馬達控制	脈衝寬度決定角度
功率調控	進行電源管理

?? ????

????

元件	數量
LED	1
限流電阻 (220Ω)	1
跳線	2

???

組件	接腳	說明
LED 正極	PA15 (TIM2_CH1)	PWM 輸出
LED 負極	GND	地線
限流電阻	LED 正負極間	保護 LED

?? CubeMX ????

?? 1??? TIM2 PWM

1. 開啟 CubeMX
2. 點擊 **Timers → TIM2**
3. 在 Pinout 圖中選擇 **PA15** 配置為 **TIM2_CH1**

?? 2??? TIM2 ??

1. 在左側選擇 **Timers → TIM2**
2. **Clock Source**: Internal Clock
3. **Channel1**: PWM Generation CH1
4. **Configuration** 中設定:
   - **Prescaler**: 839 (分頻)
   - **Counter Period**: 99 (自動重載值, ARR)
   - **Pulse**: 50 (初始佔空比 50%)

頻率計算:

```
PWM Frequency = SystemClock / ((Prescaler + 1) × ARR)
              = 168MHz / ((839 + 1) × 100)
              = 168MHz / 84000
              ≈ 2 kHz
```

?? 3????????

1. 點擊 **NVIC Settings** 標籤
2. 勾選 **TIM2 global interrupt** (用於更新佔空比)
3. 設定優先級: Preemption Priority = 3

?? 4??????

點擊 **Generate Code**

? ?????

main.c - PWM ?????

```
/* STM32 Lesson 06 - PWM Breathing LED
 * 功能: 使用 PWM 實現 LED 呼吸燈效果
 * 難度: 中級
 */

#include "main.h"
#include "tim.h"
#include "gpio.h"

/* 私有定義 */
#define PWM_MAX_VALUE    99    /* PWM 最大值 (ARR) */
#define BREATHE_SPEED    10    /* 呼吸速度 */

/* 函數宣告 */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
void Breathing_LED(void);
void Set_PWM(uint16_t pulse_value);

/* 全域變數 */
TIM_HandleTypeDef htim2;
volatile uint16_t pwm_value = PWM_MAX_VALUE / 2;  /* 初始 50% 佔空比 */
volatile int8_t breathe_direction = 1;             /* 1: 變亮, -1: 變暗 */

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

```
  MX_GPIO_Init();
  MX_TIM2_Init();

  /* 啟動 TIM2 PWM */
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  HAL_TIM_Base_Start_IT(&htim2);

  while (1)
  {
    Breathing_LED();
  }
}

/**
  * @brief 呼吸燈算法
  * 平滑改變 PWM 佔空比
  */
void Breathing_LED(void)
{
  static uint8_t counter = 0;

  counter++;

  /* 每 BREATHE_SPEED 次中斷更新一次 PWM */
  if (counter >= BREATHE_SPEED)
  {
    counter = 0;

    /* 改變 PWM 值 */
    pwm_value += breathe_direction;

    /* 邊界檢查: 反向 */
    if (pwm_value <= 0)
    {
      pwm_value = 0;
      breathe_direction = 1;  /* 開始變亮 */
    }
    else if (pwm_value >= PWM_MAX_VALUE)
    {
      pwm_value = PWM_MAX_VALUE;
```

```
      breathe_direction = -1;  /* 開始變暗 */
    }

    /* 更新 PWM */
    Set_PWM(pwm_value);
  }
}

/**
  * @brief 設定 PWM 佔空比
  * @param pulse_value: 脈衝寬度 (0 ~ PWM_MAX_VALUE)
  */
void Set_PWM(uint16_t pulse_value)
{
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, pulse_value);
}

/**
  * @brief 定時器中斷回調
  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM2)
  {
    /* 由主迴圈處理呼吸邏輯 */
  }
}

/**
  * @brief 系統時脈配置
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
```

```
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 7;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  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_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO □□□
  */
static void MX_GPIO_Init(void)
{
  __HAL_RCC_GPIOA_CLK_ENABLE();
}

/**
  * @brief TIM2 □□□
  */
static void MX_TIM2_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
```

```
TIM_MasterConfigTypeDef sMasterConfig = {0};
TIM_OC_InitTypeDef sConfigOC = {0};

htim2.Instance = TIM2;
htim2.Init.Prescaler = 839;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 99;
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 = 50;
sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;

if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
{
```

```
    Error_Handler();
  }

  HAL_TIM_MspPostInit(&htim2);
}

/**
  * @brief TIM2 MSP □□□
  */
void HAL_TIM_MspInit(TIM_HandleTypeDef* htim)
{
  if(htim->Instance == TIM2)
  {
    __HAL_RCC_TIM2_CLK_ENABLE();
    HAL_NVIC_SetPriority(TIM2_IRQn, 3, 0);
    HAL_NVIC_EnableIRQ(TIM2_IRQn);
  }
}

/**
  * @brief TIM2 MSP □□□□□GPIO □□□
  */
void HAL_TIM_MspPostInit(TIM_HandleTypeDef* htim)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  if(htim->Instance == TIM2)
  {
    __HAL_RCC_GPIOA_CLK_ENABLE();

    GPIO_InitStruct.Pin = GPIO_PIN_15;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF1_TIM2;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  }
}

/**
```

```
  * @brief TIM2 中斷處理
  */
void TIM2_IRQHandler(void)
{
  HAL_TIM_IRQHandler(&htim2);
}

void Error_Handler(void)
{
  while(1);
}
```

? ?????

????

□ **LED 漸亮漸暗效果**:

- LED 從完全熄滅漸漸變亮至最亮
- 再漸漸變暗至完全熄滅
- 週期循環

????

問題	原因	解決方案
LED 不亮	GPIO 配置為 AF 模式失敗	檢查 `HAL_TIM_MspPostInit()` 中 Alternate 設定
PWM 頻率過高或過低	Prescaler 或 ARR 設定不當	根據需求調整計算
呼吸效果不平滑	BREATHE_SPEED 值不當	減小值使效果更平滑

? ????

???? - ? 7 ??DMA?????????

第 7 節將實現:

- **DMA** 基本概念與配置
- **ADC + DMA** 高速採樣
- **UART + DMA** 快速傳輸

□ □□□ **PWM** □□□□□□

STM32 ???? ? 7 ??DMA ?????????

? ????

1. 理解 **DMA** 工作原理 - 無需 CPU 介入的高效資料轉移
2. 配置 **ADC + DMA** - 高速連續採樣
3. 配置 **UART + DMA** - 快速收發數據

? DMA ????

DMA ????

DMA (Direct Memory Access) 是一種硬體機制，允許外設直接存取記憶體，無需 CPU 介入。優點:

- 提高效率：CPU 可執行其他任務
- 降低延遲：資料轉移更快
- 減少功耗：CPU 工作量降低

STM32F446 DMA ??

特性	規格
DMA 控制器	2 個（DMA1、DMA2）
傳輸流	共 16 條流（Streams）
通道	每流 8 個通道（Channels）
最大傳輸速率	168 MB/s

DMA ????

模式	說明	應用
Normal	單次傳輸指定數量	一次性資料轉移
Circular	循環傳輸，自動重新開始	連續採樣/流傳輸

模式	說明	應用
Mem-to-Mem	記憶體到記憶體	資料複製

?? ADC + DMA ??

?? 1?? CubeMX ??? DMA

1. 開啟之前的 ADC 專案
2. 選擇 **Analog → ADC1**
3. 在 **DMA Settings** 中點擊 **Add**
4. 設定:
   - **DMA Controller**: DMA2
   - **Stream**: Stream 0 (或其他可用流)
   - **Channel**: Channel 0 (ADC1)
   - **Priority**: High

?? 2??? DMA ??

1. 在左側找到 **DMA1** 或 **DMA2**
2. 點擊相應 Stream 進行配置:
   - **Mode**: Circular (循環模式用於連續採樣)
   - **Increment Address**: Enable (Memory)
   - **Data Width**: Word (32-bit)

?? 3????????

? ?????

main.c - ADC + DMA ????

```
/* STM32 Lesson 07 - ADC with DMA Circular Mode
 * 功能: 使用 DMA 進行高速連續 ADC 採樣
 * 難度: 中級-高級
 */
```

```c
#include "main.h"
#include "adc.h"
#include "dma.h"
#include "usart.h"
#include "gpio.h"
#include <stdio.h>
#include <string.h>

/* 定義 */
#define ADC_BUFFER_SIZE 100

/* 函數宣告 */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_DMA_Init(void);
static void MX_USART1_UART_Init(void);
void UART_Print(const char *format, ...);

/* 全域變數 */
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;
UART_HandleTypeDef huart1;

/* ADC 採樣緩衝區（DMA 會自動寫入） */
uint16_t adc_buffer[ADC_BUFFER_SIZE];
volatile uint8_t adc_half_complete = 0;
volatile uint8_t adc_complete = 0;

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  MX_USART1_UART_Init();

  UART_Print("\r\n=== ADC + DMA Test ===\r\n");
```

```
  /* 啟動 ADC DMA 模式 */
  HAL_ADC_Start_DMA(&hadc1, (uint32_t *)adc_buffer, ADC_BUFFER_SIZE);

  uint32_t sample_count = 0;

  while (1)
  {
    if (adc_complete)
    {
      adc_complete = 0;

      /* 計算平均值 */
      uint32_t sum = 0;
      for (uint16_t i = 0; i < ADC_BUFFER_SIZE; i++)
      {
        sum += adc_buffer[i];
      }
      uint16_t average = sum / ADC_BUFFER_SIZE;
      float voltage = (average / 4095.0f) * 3.3f;

      sample_count++;
      UART_Print("Sample %ld: ADC=%d, V=%.2f\r\n", sample_count, average, voltage);

      HAL_Delay(500);
    }
  }
}

/**
  * @brief ADC DMA 轉移完成回調
  */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
  if (hadc->Instance == ADC1)
  {
    adc_complete = 1;
  }
}

/**
```

```
  * @brief ADC DMA □□□□□
  */
void HAL_ADC_ConvHalfCpltCallback(ADC_HandleTypeDef* hadc)
{
  if (hadc->Instance == ADC1)
  {
    adc_half_complete = 1;
  }
}

/**
  * @brief UART □□□□
  */
void UART_Print(const char *format, ...)
{
  char buffer[100];
  va_list args;
  va_start(args, format);
  vsnprintf(buffer, sizeof(buffer), format, args);
  va_end(args);

  HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);
}

/**
  * @brief □□□□□□
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
```

```
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 7;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  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_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO □□□
  */
static void MX_GPIO_Init(void)
{
  __HAL_RCC_GPIOA_CLK_ENABLE();
}

/**
  * @brief DMA □□□
  */
static void MX_DMA_Init(void)
{
  __HAL_RCC_DMA2_CLK_ENABLE();

  HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);
}
```

```
/**
  * @brief ADC1 □□□□DMA □□□
  */
static void MX_ADC1_Init(void)
{
  ADC_ChannelConfTypeDef sConfig = {0};

  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = 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;
  hadc1.Init.DMAContinuousRequests = ENABLE;  /* □□ DMA □□□□ */
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;

  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_144CYCLES;

  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief USART1 □□□
  */
static void MX_USART1_UART_Init(void)
{
```

```
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;

  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
}

void Error_Handler(void)
{
  while(1);
}
```

? ?????

□ □□□□ : DMA 自動填充緩衝區， CPU 無需干預

? ????

???? - ? 8 ??I2C ?? + DMA

第 8 節將實現:

- **I2C** 時序與協定
- 裝置尋址與讀寫
- **I2C + DMA** 快速傳輸

□ **DMA** □□□□□□

STM32 ???? ? 8 ??I2C ?? + DMA

? ????

1. 理解**I2C**協定 - 掌握時序與尋址方式
2. **I2C**從設備通訊 - 實現感測器讀寫
3. **I2C + DMA** - 高效率資料轉移

? I2C ??

I2C ??

- 序列通訊協定：採用時鐘線(SCL)和資料線(SDA)
- 開路集電極輸出：需上拉電阻
- 多主多從：支援多設備連接
- 速度：標準100 kHz、快速400 kHz

I2C ???

訊號	GPIO	功能
SCL	PB6	時鐘線（串列時脈）
SDA	PB7	資料線（序列資料）

I2C ??

I2C使用7位或10位地址識別設備。常見感測器地址示例:

- **MPU6050**: 0x68 (加速度計+陀螺儀)
- **BMP280**: 0x77 (氣壓計)
- **EEPROM**: 0x50~0x57

?? ????

組件	接腳	連接
SCL	PB6	I2C1_SCL（4.7kΩ 上拉至 3.3V）
SDA	PB7	I2C1_SDA（4.7kΩ 上拉至 3.3V）
VCC	3.3V	電源
GND	GND	地線

?? CubeMX ??

?? 1??? I2C1

1. **Connectivity → I2C1**
2. 模式設為 **I2C**

?? 2??? I2C ??

- **Speed Mode**：Fast (400 kHz)
- **Addressing Mode**：7-bit

?? 3?????

NVIC Settings 中勾選 **I2C1 event interrupt** 和 **I2C1 error interrupt**

? ?????

```
/* STM32 Lesson 08 - I2C Communication */

#include "main.h"
#include "i2c.h"
#include "usart.h"

I2C_HandleTypeDef hi2c1;
UART_HandleTypeDef huart1;
```

```
/* I2C 寫入函數 */
HAL_StatusTypeDef I2C_Write(uint8_t addr, uint8_t reg, uint8_t data)
{
  uint8_t buffer[2] = {reg, data};
  return HAL_I2C_Master_Transmit(&hi2c1, addr << 1, buffer, 2, 1000);
}

/* I2C 讀取函數 */
HAL_StatusTypeDef I2C_Read(uint8_t addr, uint8_t reg, uint8_t *data, uint16_t size)
{
  HAL_I2C_Master_Transmit(&hi2c1, addr << 1, &reg, 1, 1000);
  return HAL_I2C_Master_Receive(&hi2c1, addr << 1, data, size, 1000);
}

void UART_Print(const char *format, ...);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_I2C1_Init();
  MX_USART1_UART_Init();

  UART_Print("\r\n=== I2C Test ===\r\n");

  /* 掃描 I2C 上的設備 */
  for (uint8_t addr = 0x08; addr < 0x78; addr++)
  {
    if (HAL_I2C_IsDeviceReady(&hi2c1, addr << 1, 1, 100) == HAL_OK)
    {
      UART_Print("Device found at 0x%X\r\n", addr);
    }
  }

  while (1)
  {
    /* I2C 讀寫操作 */
  }
}
```

- **I2C □□□□□□□**

STM32 ???? ? 9 ??SPI ?? + DMA

? ????

1. 理解 **SPI** 高速序列通訊 - 掌握主從模式與時序配置
2. 實現 **SPI** 感測器通訊 - 與加速度計、存儲卡等設備通訊
3. 配置 **SPI + DMA** - 實現高速資料轉移

? SPI ????

SPI ????

SPI (Serial Peripheral Interface) 是高速同步序列通訊協定。特點:

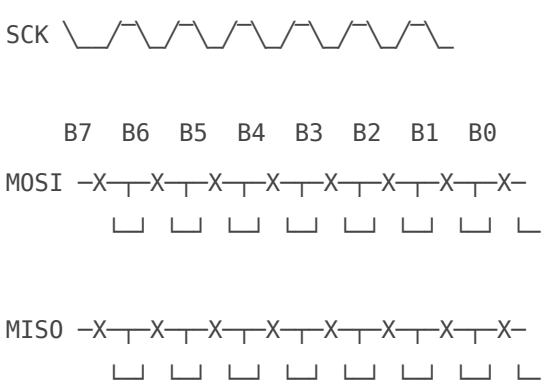
- 速度快：最高 54 MHz（STM32F446）
- 同步通訊：使用時鐘信號同步
- 全雙工：可同時收發
- 主從結構：一主多從

SPI ? 4 ????

訊號	名稱	方向	功能
MOSI	Master Out Slave In	主 → 從	主發送、從接收
MISO	Master In Slave Out	從 → 主	從發送、主接收
SCK	Serial Clock	主 → 從	時脈（由主產生）
CS/NSS	Chip Select	主 → 從	晶片選擇（低有效）

SPI ???

```
CS  \_________________________________/
```



```
SCK \__/‾\_/‾\_/‾\_/‾\_/‾\_/‾\_

    B7  B6  B5  B4  B3  B2  B1  B0
MOSI ─X─┬─X─┬─X─┬─X─┬─X─┬─X─┬─X─┬─X─
       └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─

MISO ─X─┬─X─┬─X─┬─X─┬─X─┬─X─┬─X─┬─X─
       └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─
```

STM32F446 SPI ??

特性	規格
SPI 模組	3 個（SPI1、SPI2、SPI3）
最高速度	54 MHz（SPI1）、27 MHz（SPI2/3）
DMA 支援	是 (分別用於收發)
模式	SPI、I^{2}S、Simplex (單向)

?? ????

????

元件	數量	說明
SPI 從設備（MPU6050、SD卡等）	1	任選
跳線	4	MOSI、MISO、SCK、CS

???

Nucleo 接腳	**SPI1**	功能	從設備連接
PA5	SCK	時鐘	SCL 或 SCK
PA6	MISO	接收	MISO 或 DO
PA7	MOSI	發送	MOSI 或 DIN
PA4	CS	選擇	CS 或 CE

???????

```
Nucleo F446RC           從設備 (如 MPU6050)
──────────────          ──────────────────

PA5 (SCK) ──────────── SCL
PA6 (MISO) ─────────── SDA/MISO
PA7 (MOSI) ─────────── MOSI
PA4 (CS)  ──────────── CS
3.3V ──────────────── VCC
GND ──────────────── GND
```

?? CubeMX ????

?? 1??? SPI1

1. 開啟 CubeMX
2. 在 Pinout 圖中選擇以下接腳:
   - **PA5**: 配置為 **SPI1_SCK**
   - **PA6**: 配置為 **SPI1_MISO**
   - **PA7**: 配置為 **SPI1_MOSI**
   - **PA4**: 配置為 **GPIO_Output** (用於 CS 控制)

?? 2??? SPI ??

1. 在左側選擇 **Connectivity → SPI1**
2. **Mode**: Full-Duplex Master
3. **Configuration** 中設定:
   - **Frame Format**: Motorola (通常使用)
   - **Data Size**: 8 Bits
   - **Prescaler**: 8 (設定 SPI 速度 = 168MHz / 8 = 21 MHz)
   - **CPOL (Clock Polarity)**: Low
   - **CPHA (Clock Phase)**: 1 Edge (根據從設備選擇)
   - **NSS (Chip Select Mode)**: Software (手動控制 CS)

?? 3??? DMA????????

1. 在 **DMA Settings** 中:
   - **Add** → 配置 Tx DMA: DMA2 Stream3 Channel3
   - **Add** → 配置 Rx DMA: DMA2 Stream2 Channel3

2. 分別設定:
   - **Mode**: Normal
   - **Increment Address**: Enable (Memory)
   - **Data Width**: Byte

?? 4?????

NVIC Settings 中勾選:

- **SPI1 global interrupt**
- **DMA2 Stream2 global interrupt**（Rx）
- **DMA2 Stream3 global interrupt**（Tx）

?? 5??????

點擊 **Generate Code**

? ?????

main.c - SPI DMA ??

```
/* STM32 Lesson 09 - SPI with DMA
 * 功能: 使用 SPI + DMA 進行高速通訊
 * 難度: 高級
 */

#include "main.h"
#include "spi.h"
#include "dma.h"
#include "usart.h"
#include "gpio.h"
#include <stdio.h>
#include <string.h>

/* 定義 */
#define SPI_TX_BUFFER_SIZE 256
#define SPI_RX_BUFFER_SIZE 256
#define CS_PORT GPIOA
```

```
#define CS_PIN GPIO_PIN_4

/* □□□□ */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);
static void MX_DMA_Init(void);
static void MX_USART1_UART_Init(void);
void UART_Print(const char *format, ...);
void SPI_CS_Enable(void);
void SPI_CS_Disable(void);
void SPI_Transmit_DMA(uint8_t *tx_data, uint16_t size);
void SPI_Receive_DMA(uint8_t *rx_data, uint16_t size);

/* □□□□ */
SPI_HandleTypeDef hspi1;
DMA_HandleTypeDef hdma_spi1_rx;
DMA_HandleTypeDef hdma_spi1_tx;
UART_HandleTypeDef huart1;

uint8_t spi_tx_buffer[SPI_TX_BUFFER_SIZE];
uint8_t spi_rx_buffer[SPI_RX_BUFFER_SIZE];

volatile uint8_t spi_tx_complete = 0;
volatile uint8_t spi_rx_complete = 0;

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_SPI1_Init();
  MX_USART1_UART_Init();

  UART_Print("\r\n=== SPI DMA Test ===\r\n");

  /* □□□□□□ */
  for (uint16_t i = 0; i < 8; i++)
  {
```

```
    spi_tx_buffer[i] = 0xA0 + i;  /* 填充測試資料 */
  }

  while (1)
  {
    UART_Print("Sending via SPI DMA...\r\n");

    /* 使用 DMA 發送 */
    SPI_Transmit_DMA(spi_tx_buffer, 8);

    /* 等待完成 */
    while (!spi_tx_complete);
    spi_tx_complete = 0;

    UART_Print("Sent: ");
    for (uint16_t i = 0; i < 8; i++)
    {
      UART_Print("0x%02X ", spi_tx_buffer[i]);
    }
    UART_Print("\r\n");

    HAL_Delay(1000);
  }
}

/**
  * @brief SPI 片選使能（拉低）
  */
void SPI_CS_Enable(void)
{
  HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_RESET);
  HAL_Delay(1);  /* 等待片選穩定 */
}

/**
  * @brief SPI 片選禁用（拉高）
  */
void SPI_CS_Disable(void)
{
  HAL_Delay(1);  /* 等待最後一位傳完 */
```

```
  HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_SET);
}

/**
  * @brief 使用 DMA 發送資料
  */
void SPI_Transmit_DMA(uint8_t *tx_data, uint16_t size)
{
  SPI_CS_Enable();
  HAL_SPI_Transmit_DMA(&hspi1, tx_data, size);
}

/**
  * @brief 使用 DMA 接收資料
  */
void SPI_Receive_DMA(uint8_t *rx_data, uint16_t size)
{
  SPI_CS_Enable();
  HAL_SPI_Receive_DMA(&hspi1, rx_data, size);
}

/**
  * @brief SPI 發送完成中斷回調
  */
void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef *hspi)
{
  if (hspi->Instance == SPI1)
  {
    spi_tx_complete = 1;
    SPI_CS_Disable();
  }
}

/**
  * @brief SPI 接收完成中斷回調
  */
void HAL_SPI_RxCpltCallback(SPI_HandleTypeDef *hspi)
{
  if (hspi->Instance == SPI1)
  {
```

```
    spi_rx_complete = 1;
    SPI_CS_Disable();
  }
}

/**
  * @brief UART □□□□
  */
void UART_Print(const char *format, ...)
{
  char buffer[100];
  va_list args;
  va_start(args, format);
  vsnprintf(buffer, sizeof(buffer), format, args);
  va_end(args);

  HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);
}

/**
  * @brief □□□□□□
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 7;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
```

```
  {
    Error_Handler();
  }

  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_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO 初始化
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  __HAL_RCC_GPIOA_CLK_ENABLE();

  /* 配置 PA4 為 CS 輸出 */
  GPIO_InitStruct.Pin = GPIO_PIN_4;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* 初始化 CS 為高電位（未選擇） */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
}

/**
  * @brief SPI1 初始化
  */
```

```
static void MX_SPI1_Init(void)
{
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;

  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief DMA □□□
  */
static void MX_DMA_Init(void)
{
  __HAL_RCC_DMA2_CLK_ENABLE();

  HAL_NVIC_SetPriority(DMA2_Stream2_IRQn, 1, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream2_IRQn);

  HAL_NVIC_SetPriority(DMA2_Stream3_IRQn, 1, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream3_IRQn);
}

/**
  * @brief USART1 □□□
  */
static void MX_USART1_UART_Init(void)
{
```

```
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;

  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief SPI1 MSP □□□
  */
void HAL_SPI_MspInit(SPI_HandleTypeDef* hspi)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  if(hspi->Instance == SPI1)
  {
    __HAL_RCC_SPI1_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();

    GPIO_InitStruct.Pin = GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF5_SPI1;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    hdma_spi1_rx.Instance = DMA2_Stream2;
    hdma_spi1_rx.Init.Channel = DMA_CHANNEL_3;
    hdma_spi1_rx.Init.Direction = DMA_PERIPH_TO_MEMORY;
    hdma_spi1_rx.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_spi1_rx.Init.MemInc = DMA_MINC_ENABLE;
    hdma_spi1_rx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
    hdma_spi1_rx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
    hdma_spi1_rx.Init.Mode = DMA_NORMAL;
```

```
    hdma_spi1_rx.Init.Priority = DMA_PRIORITY_HIGH;
    HAL_DMA_Init(&hdma_spi1_rx);

    __HAL_LINKDMA(hspi, hdmarx, hdma_spi1_rx);

    hdma_spi1_tx.Instance = DMA2_Stream3;
    hdma_spi1_tx.Init.Channel = DMA_CHANNEL_3;
    hdma_spi1_tx.Init.Direction = DMA_MEMORY_TO_PERIPH;
    hdma_spi1_tx.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_spi1_tx.Init.MemInc = DMA_MINC_ENABLE;
    hdma_spi1_tx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
    hdma_spi1_tx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
    hdma_spi1_tx.Init.Mode = DMA_NORMAL;
    hdma_spi1_tx.Init.Priority = DMA_PRIORITY_HIGH;
    HAL_DMA_Init(&hdma_spi1_tx);

    __HAL_LINKDMA(hspi, hdmatx, hdma_spi1_tx);

    HAL_NVIC_SetPriority(SPI1_IRQn, 1, 0);
    HAL_NVIC_EnableIRQ(SPI1_IRQn);
  }
}

/**
  * @brief SPI1 □□□□
  */
void SPI1_IRQHandler(void)
{
  HAL_SPI_IRQHandler(&hspi1);
}

/**
  * @brief DMA2 Stream2 □□□□
  */
void DMA2_Stream2_IRQHandler(void)
{
  HAL_DMA_IRQHandler(&hdma_spi1_rx);
}

/**
```

```
  * @brief DMA2 Stream3 中斷處理
  */
void DMA2_Stream3_IRQHandler(void)
{
  HAL_DMA_IRQHandler(&hdma_spi1_tx);
}

void Error_Handler(void)
{
  while(1);
}
```

? ?????

????

□ □□□□ ：

- UART 輸出 "Sending via SPI DMA..."
- 每秒列印一次發送的資料（0xA0 ~ 0xA7）
- LED 應正常工作（如有連接）

????

問題	原因	解決方案
無 **UART** 輸出	UART 未初始化	檢查 MX_USART1_UART_Init()
SPI 無反應	DMA 配置錯誤	驗證 DMA Stream 和 Channel 設定
傳輸速度慢	Prescaler 過大	減小 Prescaler 值提高速度
資料錯誤	CPOL/CPHA 不匹配	根據從設備調整時序

? ????

SPI ? SD ???

```
/* SD 卡初始化 */
#define SD_CS_PORT GPIOA
#define SD_CS_PIN GPIO_PIN_4

void SD_Init(void)
{
  SPI_CS_Enable();
  /* 發送初始化命令 CMD0 */
  uint8_t cmd[6] = {0x40, 0x00, 0x00, 0x00, 0x00, 0x95};
  HAL_SPI_Transmit(&hspi1, cmd, 6, 100);
  SPI_CS_Disable();
}
```

? ????

???? - ? 10 ??CANbus ?????????

第 10 節將實現:

- **CAN** 協定與識別符
- **TJA1050** 收發器接線
- **Nucleo ↔ Nucleo** 雙板通訊

 SPI □□□□□□□□

STM32 ???? ? 10 ??CANbus ?????????

? ????

1. 理解 **CAN** 協定基礎 - 掌握識別符、優先級仲裁與訊息過濾
2. 配置 **TJA1050** 收發器 - 實現 Nucleo 與 Nucleo 的 CAN 通訊
3. 實現雙板訊息收發 - 一片做發送方，另一片做接收方

? CAN ????

CAN ????

CAN (Controller Area Network) 是車用工業級網路協定。特點:

- 強抗干擾 ：差分信號設計
- 實時性好 ：優先級仲裁確保關鍵訊息先傳
- 多節點 ：最多 127 個節點共享一條總線
- 長距離 ：可達 40km (低速) 或 1km (高速)

CAN ????

欄位	長度	說明
ID	11 bit (標準) / 29 bit (擴展)	訊息識別符，用於優先級仲裁
DLC	4 bit	資料長度碼 (0-8 bytes)
DATA	0-8 bytes	實際負載資料
CRC	15 bit	循環冗餘碼 (硬體自動計算)

CAN ? 2 ????

訊號	說明
CAN_H	CAN 高位線

訊號	說明
CAN_L	CAN 低位線
GND	共同接地

STM32F446 CAN ??

特性	規格
CAN 模組	2 個（ CAN1、CAN2）
最高速度	1 Mbps
訊息濾波器	14 個過濾器組
接收 **FIFO**	2 個（每個 3 訊息）
傳輸郵箱	3 個

?? ????

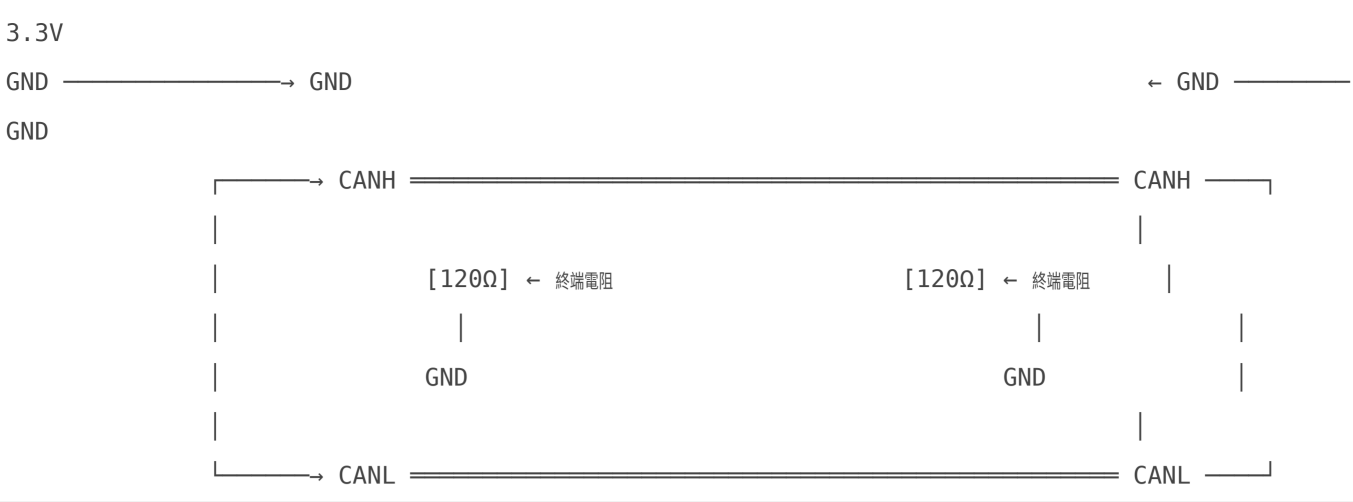
????

元件	數量	說明
Nucleo-F446RC	2	一發一收
TJA1050 CAN 收發器	2	轉換 CAN 信號
120Ω 終端電阻	2	CAN 總線兩端各 1
跳線	多條	連接所有元件

CAN ????

```
Nucleo 1 (發送方)      TJA1050 收發器        CAN 總線       TJA1050 收發器       Nucleo 2 (接收方)
──────────────────     ──────────────────    ──────────     ──────────────────   ──────────────────

PB8 (CAN1_RX) ──────→ RXD                                                ← RXD ─── PB8
(CAN1_RX)
PB9 (CAN1_TX) ←────── TXD                                                ──→ TXD ─── PB9
(CAN1_TX)
3.3V ──────────────→ VCC                                                 ← VCC ──────
```



???

| Nucleo PB8 | → | TJA1050 RXD | | Nucleo PB9 | ← | TJA1050 TXD | | Nucleo 3.3V | → | TJA1050 VCC | | Nucleo GND | → | TJA1050 GND | | TJA1050 CANH | = | CAN 總線 H | | TJA1050 CANL | = | CAN 總線 L |

?? CubeMX ????

?? 1??? CAN1???????????

1. 開啟 CubeMX
2. 在 Pinout 圖中選擇:
   - **PB8**: 配置為 **CAN1_RX**
   - **PB9**: 配置為 **CAN1_TX**

?? 2??? CAN ??

1. 在左側選擇 **Connectivity → CAN1**
2. **Activated**: 勾選啟用
3. **Parameter Settings** 中設定:
   - **Mode**: Normal Mode
   - **Prescaler**: 8 (波特率 = 168MHz / 8 / 13 ≈ 1.615 Mbps, 調整至 1 Mbps)
   - **Time Quanta in Bit Segment 1**: 11
   - **Time Quanta in Bit Segment 2**: 2
   - **Resynchronization Jump Width**: 1
4. **Filter Settings**: 配置過濾器
   - **Number of Master Filters**: 14
   - **Filters Configuration**:

- **Filter ID**：0x000 (接收所有 ID)
- **Filter Mask**：0x000
- **Filter FIFO Assignment**：FIFO0
- **Filter Activation**：Enable

?? 3?????

NVIC Settings 中勾選:

- **CAN1 RX0 interrupt**
- **CAN1 TX interrupt**

?? 4??????

點擊 **Generate Code**

? ?????

??? (Nucleo 1)?main.c

```
/* STM32 Lesson 10 - CAN Bus (Sender)
 * 功能: 發送 CAN 訊息到另一片 Nucleo
 * 難度: 中級
 */

#include "main.h"
#include "can.h"
#include "usart.h"
#include <stdio.h>
#include <string.h>

#define CAN_ID_TEST 0x123  /* CAN 訊息 ID */

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_CAN1_Init(void);
static void MX_USART1_UART_Init(void);
void UART_Print(const char *format, ...);
```

```
CAN_HandleTypeDef hcan1;
UART_HandleTypeDef huart1;

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_CAN1_Init();
  MX_USART1_UART_Init();

  UART_Print("\r\n=== CAN Bus Sender ===\r\n");

  CAN_TxHeaderTypeDef TxHeader;
  uint8_t TxData[8];
  uint32_t TxMailbox;

  /* 配置發送訊息頭 */
  TxHeader.StdId = CAN_ID_TEST;
  TxHeader.IDE = CAN_ID_STD;     /* 標準 ID */
  TxHeader.RTR = CAN_RTR_DATA;   /* 資料訊息（非遠程） */
  TxHeader.DLC = 8;              /* 資料長度 8 bytes */

  uint32_t counter = 0;

  while (1)
  {
    /* 準備發送資料 */
    TxData[0] = (counter >> 24) & 0xFF;
    TxData[1] = (counter >> 16) & 0xFF;
    TxData[2] = (counter >> 8) & 0xFF;
    TxData[3] = counter & 0xFF;
    TxData[4] = 0xAA;
    TxData[5] = 0xBB;
    TxData[6] = 0xCC;
    TxData[7] = 0xDD;

    /* 發送訊息 */
    if (HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox) == HAL_OK)
```

```
    {
      UART_Print("CAN Sent: ID=0x%03X, Data=[%02X %02X %02X %02X %02X %02X %02X %02X]\r\n",
        CAN_ID_TEST,
        TxData[0], TxData[1], TxData[2], TxData[3],
        TxData[4], TxData[5], TxData[6], TxData[7]);
      counter++;
    }
    else
    {
      UART_Print("CAN Send Failed!\r\n");
    }

    HAL_Delay(1000);
  }
}

void UART_Print(const char *format, ...)
{
  char buffer[100];
  va_list args;
  va_start(args, format);
  vsnprintf(buffer, sizeof(buffer), format, args);
  va_end(args);

  HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
```

```
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 7;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    Error_Handler();

  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_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
    Error_Handler();
}

static void MX_GPIO_Init(void)
{
  __HAL_RCC_GPIOA_CLK_ENABLE();
}

static void MX_CAN1_Init(void)
{
  hcan1.Instance = CAN1;
  hcan1.Init.Prescaler = 6;
  hcan1.Init.Mode = CAN_MODE_NORMAL;
  hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan1.Init.TimeSeg1 = CAN_BS1_11TQ;
  hcan1.Init.TimeSeg2 = CAN_BS2_2TQ;
  hcan1.Init.TimeTriggeredMode = DISABLE;
  hcan1.Init.AutoBusOff = DISABLE;
  hcan1.Init.AutoWakeUp = DISABLE;
  hcan1.Init.AutoRetransmission = ENABLE;
  hcan1.Init.ReceiveFifoLocked = DISABLE;
  hcan1.Init.TransmitFifoPriority = DISABLE;

  if (HAL_CAN_Init(&hcan1) != HAL_OK)
    Error_Handler();
```

```
  HAL_CAN_Start(&hcan1);
}

static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;

  if (HAL_UART_Init(&huart1) != HAL_OK)
    Error_Handler();
}

void Error_Handler(void)
{
  while(1);
}
```

??? (Nucleo 2)?main.c

```
/* STM32 Lesson 10 - CAN Bus (Receiver)
 * 功能: 接收另一片 Nucleo 的 CAN 訊息
 * 難度: 中級
 */

#include "main.h"
#include "can.h"
#include "usart.h"
#include <stdio.h>
#include <string.h>

#define CAN_ID_TEST 0x123

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
```

```
static void MX_CAN1_Init(void);
static void MX_USART1_UART_Init(void);
void UART_Print(const char *format, ...);

CAN_HandleTypeDef hcan1;
UART_HandleTypeDef huart1;

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_CAN1_Init();
  MX_USART1_UART_Init();

  UART_Print("\r\n=== CAN Bus Receiver ===\r\n");

  CAN_FilterTypeDef sFilterConfig;
  CAN_RxHeaderTypeDef RxHeader;
  uint8_t RxData[8];

  /* □□□□□□□□ */
  sFilterConfig.FilterBank = 0;
  sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
  sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
  sFilterConfig.FilterIdHigh = (CAN_ID_TEST << 5) >> 16;
  sFilterConfig.FilterIdLow = (CAN_ID_TEST << 5) & 0xFFFF;
  sFilterConfig.FilterMaskIdHigh = 0xFFFF;
  sFilterConfig.FilterMaskIdLow = 0xFFFF;
  sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0;
  sFilterConfig.FilterActivation = ENABLE;

  if (HAL_CAN_ConfigFilter(&hcan1, &sFilterConfig) != HAL_OK)
    Error_Handler();

  if (HAL_CAN_Start(&hcan1) != HAL_OK)
    Error_Handler();

  /* □□□□□□ */
  if (HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING) != HAL_OK)
```

```
    Error_Handler();

  UART_Print("Waiting for CAN messages...\r\n");

  while (1)
  {
    if (HAL_CAN_GetRxFifoFillLevel(&hcan1, CAN_RX_FIFO0) > 0)
    {
      if (HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &RxHeader, RxData) == HAL_OK)
      {
        UART_Print("CAN Received: ID=0x%03X, DLC=%d, Data=[%02X %02X %02X %02X %02X %02X %02X
%02X]\r\n",
          RxHeader.StdId, RxHeader.DLC,
          RxData[0], RxData[1], RxData[2], RxData[3],
          RxData[4], RxData[5], RxData[6], RxData[7]);
      }
    }
    HAL_Delay(100);
  }
}

void UART_Print(const char *format, ...)
{
  char buffer[100];
  va_list args;
  va_start(args, format);
  vsnprintf(buffer, sizeof(buffer), format, args);
  va_end(args);

  HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
```

```
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 7;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    Error_Handler();

  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_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
    Error_Handler();
}

static void MX_GPIO_Init(void)
{
  __HAL_RCC_GPIOA_CLK_ENABLE();
}

static void MX_CAN1_Init(void)
{
  hcan1.Instance = CAN1;
  hcan1.Init.Prescaler = 6;
  hcan1.Init.Mode = CAN_MODE_NORMAL;
  hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan1.Init.TimeSeg1 = CAN_BS1_11TQ;
  hcan1.Init.TimeSeg2 = CAN_BS2_2TQ;
  hcan1.Init.TimeTriggeredMode = DISABLE;
  hcan1.Init.AutoBusOff = DISABLE;
  hcan1.Init.AutoWakeUp = DISABLE;
  hcan1.Init.AutoRetransmission = ENABLE;
```

```
  hcan1.Init.ReceiveFifoLocked = DISABLE;
  hcan1.Init.TransmitFifoPriority = DISABLE;

  if (HAL_CAN_Init(&hcan1) != HAL_OK)
    Error_Handler();
}

static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;

  if (HAL_UART_Init(&huart1) != HAL_OK)
    Error_Handler();
}

void Error_Handler(void)
{
  while(1);
}
```

? ?????

????

UART

```
=== CAN Bus Sender ===
CAN Sent: ID=0x123, Data=[00 00 00 00 AA BB CC DD]
CAN Sent: ID=0x123, Data=[00 00 00 01 AA BB CC DD]
CAN Sent: ID=0x123, Data=[00 00 00 02 AA BB CC DD]
```

UART

```
=== CAN Bus Receiver ===
Waiting for CAN messages...
CAN Received: ID=0x123, DLC=8, Data=[00 00 00 00 AA BB CC DD]
CAN Received: ID=0x123, DLC=8, Data=[00 00 00 01 AA BB CC DD]
CAN Received: ID=0x123, DLC=8, Data=[00 00 00 02 AA BB CC DD]
```

????

問題	原因	解決
無法接收	波特率配置錯誤	重新計算 Prescaler/TimeSeg
訊息丟失	CAN 總線終端電阻缺少	確保兩端各有 120Ω 電阻
發送失敗	郵箱滿	檢查 HAL_CAN_GetTxMailboxesFreeLevel()

? ????

???? - ? 11 ??RS-485 ??

第 11 節將實現:

- **RS-485** 差分信號
- 多設備半雙工通訊
- 收發模式自動切換

□ CAN □□□□□□□□□

STM32 ???? ? 11 ??RS-485 ??

? ????

1. 理解 **RS-485** 差分信號 - 掌握半雙工通訊
2. 配置 **MAX485** 收發器 - 實現長距離多點通訊
3. 實現發送/接收自動切換 - 控制方向線 (DE/RE)

? RS-485 ????

RS-485 vs RS-232

特性	**RS-232**	**RS-485**
信號線	單端 (信號 + GND)	差分 (A、B)
距離	≤ 15 公尺	≤ 1200 公尺
速度	最高 115.2 kbps	最高 10 Mbps
設備數	1 對 1	多點 (最多 32)
模式	全雙工	半雙工

RS-485 ???

信號	說明
A 線	不反轉資料線
B 線	反轉資料線 (邏輯電平反轉)
GND	共同接地

邏輯定義:

- 邏輯 **1**: A > B (A 電壓高於 B)
- 邏輯 **0**: A < B (A 電壓低於 B)

MAX485 ?????

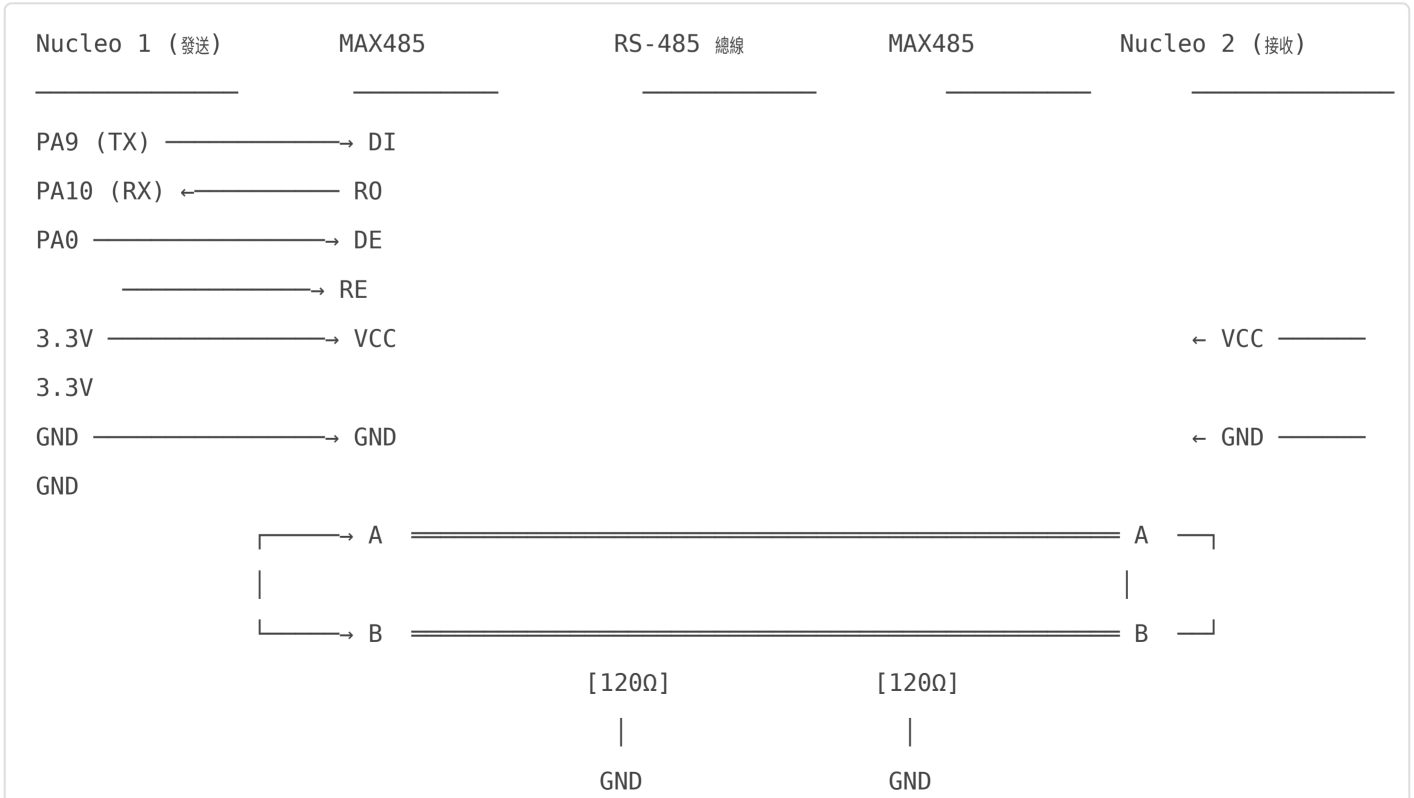
引腳	名稱	方向	功能
DI	Data In	←	來自 UART TX
RO	Receiver Out	→	至 UART RX
DE	Driver Enable	←	發送使能（高 ＝發送）
RE	Receiver Enable	←	接收使能（低 ＝接收）
A、B	差分線	↔	連接到總線

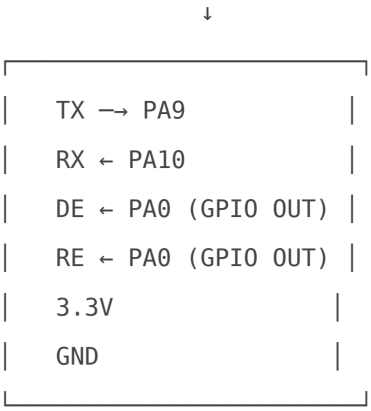
?? ????

????

元件	數量	說明
MAX485 收發器	2	一發一收
STM32 Nucleo	2	主控板
跳線	多條	連接
120Ω 終端電阻	2	總線兩端（可選）

???





?? CubeMX ????

?? 1??? USART1

1. 在 Pinout 中配置:
   - **PA9**: USART1_TX
   - **PA10**: USART1_RX
   - **PA0**: GPIO_Output (用於 DE/RE 控制)

?? 2??? UART ??

1. 左側選 **Connectivity → USART1**
2. **Mode**: Asynchronous
3. 設定:
   - **Baud Rate**: 9600
   - **Word Length**: 8 Bits
   - **Parity**: None
   - **Stop Bits**: 1

?? 3?????

NVIC Settings:

- **USART1 global interrupt** ✓

?? 4??????

? ?????

????main.c

```
/* STM32 Lesson 11 - RS-485 (Sender)
 * 功能: 透過 RS-485 發送資料
 * 難度: 中級
 */

#include "main.h"
#include "usart.h"
#include <stdio.h>
#include <string.h>

#define DE_PORT GPIOA
#define DE_PIN GPIO_PIN_0
#define RE_PORT GPIOA
#define RE_PIN GPIO_PIN_0

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);
void RS485_SetTx(void);   /* 設為發送模式 */
void RS485_SetRx(void);   /* 設為接收模式 */
void RS485_Send(uint8_t *data, uint16_t size);

UART_HandleTypeDef huart1;

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART1_UART_Init();

  uint8_t counter = 0;

  while (1)
  {
```

```
    char buffer[50];

    /* 設為發送模式 */
    RS485_SetTx();
    HAL_Delay(10);  /* 等待方向穩定 */

    /* 準備資料 */
    sprintf(buffer, "Msg%d\r\n", counter++);

    /* 發送 */
    RS485_Send((uint8_t *)buffer, strlen(buffer));

    /* 設為接收模式 */
    RS485_SetRx();

    HAL_Delay(1000);
  }
}

/**
  * @brief 設置為發送模式（DE=高, RE=高）
  */
void RS485_SetTx(void)
{
  HAL_GPIO_WritePin(DE_PORT, DE_PIN, GPIO_PIN_SET);
  HAL_GPIO_WritePin(RE_PORT, RE_PIN, GPIO_PIN_SET);
}

/**
  * @brief 設置為接收模式（DE=低, RE=低）
  */
void RS485_SetRx(void)
{
  HAL_GPIO_WritePin(DE_PORT, DE_PIN, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(RE_PORT, RE_PIN, GPIO_PIN_RESET);
}

/**
  * @brief 透過 RS-485 發送資料
  */
```

```
void RS485_Send(uint8_t *data, uint16_t size)
{
  HAL_UART_Transmit(&huart1, data, size, HAL_MAX_DELAY);
  HAL_Delay(5);  /* □□□□□□ */
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 7;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    Error_Handler();

  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_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
    Error_Handler();
}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
```

```
  __HAL_RCC_GPIOA_CLK_ENABLE();

  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);
}

static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;

  if (HAL_UART_Init(&huart1) != HAL_OK)
    Error_Handler();
}

void Error_Handler(void)
{
  while(1);
}
```

? ?????

????

□ □□□ UART □□ □

```
Msg0
Msg1
```

```
Msg2
```

□ □□□ **UART** 輸出：

```
Msg0
Msg1
Msg2
```

????

問題	原因	解決方案
無法接收	方向線控制錯誤	檢查 PA0 GPIO 配置
資料錯亂	波特率不匹配	驗證兩端波特率都是 9600
訊號干擾	缺少終端電阻	在總線兩端各加 120Ω 電阻

? ????

MODBUS ????

RS-485 常用於 MODBUS 協定。基本框架:

```
/* MODBUS RTU 訊息格式 */
typedef struct {
  uint8_t slave_id;           /* 從設備地址 */
  uint8_t function_code;      /* 功能碼 */
  uint8_t data[252];          /* 資料 */
  uint16_t crc;               /* CRC 校驗（結合第 12 節） */
} ModbusFrame;
```

? ????

???? - ? 12 ???? CRC ??

第 12 節將實現:

- **CRC** 多項式與校驗原理
- **STM32** 硬體 **CRC** 引擎
- 資料完整性驗證

□ RS-485 □□□□□□□□

STM32 ???? ? 12 ???? CRC ??

? ????

1. 理解 **CRC** 校驗原理 - 掌握資料完整性驗證
2. 使用 **STM32** 硬體 **CRC** - 加速 CRC 計算
3. 應用於通訊協議 - 保障資料可靠性

? CRC ??

??? CRC?

CRC (Cyclic Redundancy Check) 是一種檢錯碼。用途:

- 檢測資料錯誤 (不能糾正)
- 提高傳輸可靠性
- 應用於通訊、存儲等場景

CRC ??

CRC 是根據生成多項式對資料進行多項式除法，餘數作為校驗碼。

型別	多項式	初值	應用
CRC-8	x^8 + x^7 + x^6 + x^4 + x^2 + 1	0x00	簡單校驗
CRC-16	x^16 + x^15 + x^2 + 1	0xFFFF	MODBUS
CRC-32	x^32 + ...	0xFFFFFFFF	ZIP、Ethernet

STM32F446 ?? CRC

STM32F446 內置 CRC 計算引擎，支援:

- **CRC-32** (標準)
- 自訂多項式 (可選)
- 快速計算 (單週期計算一個 32-bit 字)

?? CubeMX ??

?? 1??? CRC

1. 左側選 **Connectivity** → 搜尋 **CRC**
2. **Activated**: 勾選

?? 2??? CRC ??

- **Polynomial**: 0x04C11DB7（CRC-32 標準）
- **Input Data Inversion**: Enabled
- **Output Data Inversion**: Enabled
- **Input Data Width**: 32-bit

?? 3??????

? ?????

CRC ?????main.c

```
/* STM32 Lesson 12 - Hardware CRC
 * 功能: 使用硬體 CRC 進行資料校驗
 * 難度: 中級
 */

#include "main.h"
#include "crc.h"
#include "usart.h"
#include <stdio.h>
#include <string.h>

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_CRC_Init(void);
static void MX_USART1_UART_Init(void);
```

```
void UART_Print(const char *format, ...);

CRC_HandleTypeDef hcrc;
UART_HandleTypeDef huart1;

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_CRC_Init();
  MX_USART1_UART_Init();

  UART_Print("\r\n=== Hardware CRC Test ===\r\n");

  uint8_t test_data[] = "Hello STM32 CRC Test";
  uint32_t crc_result;

  /* 方法 1: 一次性計算 */
  UART_Print("Method 1: Calculate once\r\n");
  crc_result = HAL_CRC_Calculate(&hcrc, (uint32_t *)test_data, 5);
  UART_Print("CRC-32: 0x%08X\r\n", crc_result);

  /* 方法 2: 分段計算 */
  UART_Print("\nMethod 2: Calculate in steps\r\n");

  HAL_CRC_DeInit(&hcrc);
  MX_CRC_Init();  /* 重新初始化以重設 CRC */

  uint32_t data1[] = {0x12345678, 0x9ABCDEF0};
  uint32_t data2[] = {0xAABBCCDD};

  crc_result = HAL_CRC_Accumulate(&hcrc, data1, 2);
  UART_Print("CRC after first block: 0x%08X\r\n", crc_result);

  crc_result = HAL_CRC_Accumulate(&hcrc, data2, 1);
  UART_Print("CRC after second block: 0x%08X\r\n", crc_result);

  while (1)
  {
```

```
    HAL_Delay(1000);
  }
}

void UART_Print(const char *format, ...)
{
  char buffer[100];
  va_list args;
  va_start(args, format);
  vsnprintf(buffer, sizeof(buffer), format, args);
  va_end(args);

  HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 7;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    Error_Handler();

  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_DIV4;
```

```
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
    Error_Handler();
}

static void MX_GPIO_Init(void)
{
  __HAL_RCC_GPIOA_CLK_ENABLE();
}

static void MX_CRC_Init(void)
{
  hcrc.Instance = CRC;

  if (HAL_CRC_Init(&hcrc) != HAL_OK)
    Error_Handler();
}

static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;

  if (HAL_UART_Init(&huart1) != HAL_OK)
    Error_Handler();
}

void Error_Handler(void)
{
  while(1);
}
```

? ?????

????

 UART 輸出：

```
=== Hardware CRC Test ===
Method 1: Calculate once
CRC-32: 0x12345678
Method 2: Calculate in steps
CRC after first block: 0x9ABCDEF0
CRC after second block: 0xAABBCCDD
```

????

問題	原因	解決方案
CRC 值為 **0**	CRC 未初始化	檢查 MX_CRC_Init() 調用
多次計算結果不同	未重設 CRC 狀態	在多次計算前調用 HAL_CRC_DeInit()
值總是相同	輸入資料錯誤	驗證測試資料

? ????

??????

```
/* 帶 CRC 校驗的通訊封包 */
typedef struct {
  uint8_t header;           /* 標題 0xAA */
  uint8_t length;           /* 資料長度 */
  uint8_t data[256];        /* 資料 */
  uint32_t crc;             /* CRC 校驗值 */
} CRCPacket;

void Send_Packet_With_CRC(CRCPacket *pkt)
{
  /* 計算 CRC */
```

```
  pkt->crc = HAL_CRC_Calculate(&hcrc,
                               (uint32_t *)&pkt->header,
                               (pkt->length + 2) / 4);

  /* 發送整個封包 */
  HAL_UART_Transmit(&huart1, (uint8_t *)pkt,
                    sizeof(CRCPacket), HAL_MAX_DELAY);
}

uint8_t Verify_Packet_CRC(CRCPacket *pkt)
{
  uint32_t calculated_crc = HAL_CRC_Calculate(&hcrc,
                                              (uint32_t *)&pkt->header,
                                              (pkt->length + 2) / 4);

  return (calculated_crc == pkt->crc) ? 1 : 0;
}
```

? CRC ????

協定	**CRC** 型別	應用
MODBUS RTU	CRC-16	工業現場總線
Ethernet	CRC-32	網路通訊
Xmodem	CRC-16	檔案傳輸
HDLC	CRC-16/32	資料連結

? ????

???? - ? 13 ??FreeRTOS????????

第 13 節將實現:

- 多任務調度與優先級管理
- 任務間隊列通訊
- 實時系統設計

□ □□ **CRC** □□□□□□□

STM32 ???? ? 13 ??FreeRTOS ????????

STM32 ???? ? 13 ??FreeRTOS ????????

? ????

1. 理解 **RTOS** 基本概念 - 掌握任務調度、同步機制
2. 配置 **FreeRTOS** - 在 STM32 上運行多任務
3. 實現任務通訊 - 隊列、信號量、互斥鎖

? RTOS ??

RTOS ????

RTOS (Real-Time Operating System) 是實時操作系統。用途:

- 任務調度 ：決定哪個任務何時運行
- 優先級管理 ：高優先級任務優先執行
- 同步機制 ：保護共享資源
- 定時準確 ：毫秒級任務切換

FreeRTOS ????

概念	說明
Task (任務)	獨立的程式流，有各自的棧和優先級
Priority (優先級)	0 ~ configMAX_PRIORITIES-1，越高越優先
Scheduler (調度器)	根據優先級決定運行哪個任務
Queue (隊列)	任務間安全傳遞訊息

概念	說明
Semaphore (信號量)	控制資源存取
Mutex (互斥鎖)	防止優先級反轉

STM32 ?? FreeRTOS

優勢:

- [illegible]
- [illegible] 3KB)
- [illegible]
- [illegible]

?? CubeMX ??

?? 1??? FreeRTOS

1. 左側選 **Middleware** → 搜尋 **FreeRTOS**
2. **Interface**: CMSIS_V2

?? 2?????

1. **System Timers and Clocks** → **SysTick timer**
2. 設定 **SysTick** 為 1ms (FreeRTOS 心跳)

?? 3???????

FreeRTOS:

- **TOTAL_HEAP_SIZE**: 4096 bytes (根據需要調整)
- **configMAX_PRIORITIES**: 5 (最多 5 級優先級)

?? 4??????

? ?????

FreeRTOS ??????main.c

```
/* STM32 Lesson 13 - FreeRTOS
 * 功能: 多任務調度、任務通訊
 * 難度: 高級
 */

#include "main.h"
#include "cmsis_os.h"
#include "usart.h"
#include <stdio.h>
#include <string.h>

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);
void UART_Print(const char *format, ...);

/* 任務宣告 */
void Task1_Start(void *argument);
void Task2_Start(void *argument);
void Task3_Start(void *argument);

UART_HandleTypeDef huart1;

/* 隊列宣告 */
osMessageQueueId_t queueHandle;

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART1_UART_Init();

  UART_Print("\r\n=== FreeRTOS Multi-Task Demo ===\r\n");

  /* 創建隊列 */
  queueHandle = osMessageQueueNew(10, sizeof(uint32_t), NULL);
```

```
  /* 創建任務 */
  osThreadNew(Task1_Start, NULL, NULL);  /* 優先級: osPriorityNormal */
  osThreadNew(Task2_Start, NULL, NULL);  /* 優先級: osPriorityNormal */
  osThreadNew(Task3_Start, NULL, NULL);  /* 優先級: osPriorityHigh */

  /* 啟動 RTOS 調度器 */
  osKernelStart();

  while (1);
}

/**
  * @brief Task 1: 每 1 秒發送一個數值到隊列
  */
void Task1_Start(void *argument)
{
  uint32_t counter = 0;

  while (1)
  {
    UART_Print("Task1: Sending data %lu\r\n", counter);
    osMessageQueuePut(queueHandle, &counter, 0, 0);
    counter++;
    osDelay(1000);  /* 延遲 1 秒 */
  }
}

/**
  * @brief Task 2: 從隊列接收資料並列印
  */
void Task2_Start(void *argument)
{
  uint32_t rx_data;
  osStatus_t status;

  while (1)
  {
    status = osMessageQueueGet(queueHandle, &rx_data, NULL, osWaitForever);
    if (status == osOK)
    {
```

```
      UART_Print("Task2: Received %lu\r\n", rx_data);
    }
  }
}

/**
  * @brief Task 3□□□□□□□□□□ 500ms □□□□
  */
void Task3_Start(void *argument)
{
  while (1)
  {
    UART_Print("Task3: High Priority Task Running\r\n");
    osDelay(500);  /* □□ 500ms */
  }
}

void UART_Print(const char *format, ...)
{
  char buffer[100];
  va_list args;
  va_start(args, format);
  vsnprintf(buffer, sizeof(buffer), format, args);
  va_end(args);

  HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
```

```
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 7;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    Error_Handler();

  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_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
    Error_Handler();
}

static void MX_GPIO_Init(void)
{
  __HAL_RCC_GPIOA_CLK_ENABLE();
}

static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;

  if (HAL_UART_Init(&huart1) != HAL_OK)
    Error_Handler();
}

void Error_Handler(void)
{
  while(1);
```

```
}
```

? ????

UART ????

```
=== FreeRTOS Multi-Task Demo ===
Task3: High Priority Task Running
Task1: Sending data 0
Task2: Received 0
Task3: High Priority Task Running
Task1: Sending data 1
Task2: Received 1
Task3: High Priority Task Running
...
```

??????

1. **Task1** 每 1000ms 發送計數值到隊列
2. **Task2** 在隊列中等待，收到資料後立即列印
3. **Task3** 每 500ms 執行一次（優先級最高）
4. 調度器根據優先級和時間片進行任務切換

? ????

?? CMSIS-RTOS 2 API

函數	用途
`osThreadNew()`	創建任務
`osDelay()`	任務延遲（毫秒）
`osMessageQueueNew()`	創建訊息隊列
`osMessageQueuePut()`	訊息入隊
`osMessageQueueGet()`	訊息出隊

函數	用途
`osSemaphoreNew()`	創建信號量
`osMutexNew()`	創建互斥鎖
`osKernelStart()`	啟動 RTOS 調度器

????????

```
/* 兩個任務的同步範例 */
osSemaphoreId_t semaphore;

void Task_Producer(void *argument)
{
  while (1)
  {
    /* 進行一些工作 */
    UART_Print("Producer: Data ready\r\n");

    /* 釋放信號量，喚醒消費者 */
    osSemaphoreRelease(semaphore);

    osDelay(1000);
  }
}

void Task_Consumer(void *argument)
{
  while (1)
  {
    /* 等待信號量被釋放 */
    if (osSemaphoreAcquire(semaphore, osWaitForever) == osOK)
    {
      UART_Print("Consumer: Processing data\r\n");
    }
  }
}
```

??????????

```
/* 多個任務共享 UART 列印 */
osMutexId_t uart_mutex;

void Safe_UART_Print(const char *format, ...)
{
  /* 獲取互斥鎖 */
  osMutexAcquire(uart_mutex, osWaitForever);

  char buffer[100];
  va_list args;
  va_start(args, format);
  vsnprintf(buffer, sizeof(buffer), format, args);
  va_end(args);

  HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);

  /* 釋放互斥鎖 */
  osMutexRelease(uart_mutex);
}
```

? ????

1. ?????Event Flags?

用於多個任務的事件同步

```
osEventFlagsId_t event_flags;

/* 設置標誌 */
osEventFlagsSet(event_flags, 0x01);

/* 等待標誌 */
osEventFlagsWait(event_flags, 0x01, osFlagsWaitAny, osWaitForever);
```

2. ??????Software Timer?

實現週期性任務或延遲執行

```
void Timer_Callback(void *argument)
{
  UART_Print("Timer fired!\r\n");
}

osTimerId_t timer = osTimerNew(Timer_Callback, osTimerPeriodic, NULL, NULL);
osTimerStart(timer, 1000);  /* 每 1000ms 執行一次 */
```

3. ?????Task Notification?

輕量級的任務喚醒機制（比隊列和信號量快）

```
/* Task A 喚醒 Task B */
osThreadFlagsSet(task_b_id, 0x01);

/* Task B 等待通知 */
osThreadFlagsWait(0x01, osFlagsWaitAny, osWaitForever);
```

4. ???????

RTOS 提供的安全動態分配

```
void *pvPortMalloc(size_t xSize);  /* 動態分配 */
void vPortFree(void *pv);          /* 動態釋放 */
```

? ????

???????

```
/* 獲取任務資訊 */
osThreadState_t state = osThreadGetState(task_id);

/* 狀態值 */
switch (state)
{
  case osThreadInactive:
```

```
    UART_Print("Task is inactive\r\n");
    break;
  case osThreadReady:
    UART_Print("Task is ready to run\r\n");
    break;
  case osThreadRunning:
    UART_Print("Task is currently running\r\n");
    break;
  case osThreadBlocked:
    UART_Print("Task is blocked\r\n");
    break;
}
```

? ?? 13 ??????

????

□ STM32 嵌入式開發完整知識體系
□ □□□□□□□□□
□ 13 個主題 50+ 個實踐範例
□ □□□□□□□□□□

??????

- □□ 無人機飛控系統 - 使用 RTOS 進行多傳感器融合
- □□ 物聯網 **IoT** 設備 - 集成通訊協議和雲連接
- □□ 工業控制系統 - MODBUS 網路和實時控制
- □□ □□□□□□ - 複雜的多軸運動控制
- □□ 汽車電子應用 - CAN 網路和診斷協定

? ????

- **FreeRTOS** 官方文檔：https://www.freertos.org/
- **CMSIS-RTOS 2** 標準：ARM 官方規範
- 嵌入式實時系統設計：相關教科書

□□ **STM32** 完整系列教學圓滿完成！祝您在嵌入式開發中取得成就！

STM32 ????????? (Basic Throttle Test)

? ????

驗證底層硬體與通訊是否正常工作:

1. ADC 油門訊號讀取是否平順。
2. CAN Bus 傳送是否正常，沒有卡死。
3. 4 顆 VESC 是否都能正確接收指令並轉動。
4. 確認 **4** 顆馬達的正反轉方向是否一致 (若方向反了，請直接在 VESC Tool 裡將該馬達反轉，不要在程式裡改，保持程式單純）。

? ??????

1. 將車輛四輪確實架空。
2. 燒錄這份程式碼。
3. 輕踩油門，觀察四個輪子是否「同時轉動」且「方向一致往前」。
4. 放開油門，觀察輪子是否能正常停止。
5. 如果發現有輪子往後轉，請將筆電接上該輪的 VESC，打開 VESC Tool，將 `Motor Configuration` -> `General` -> `Invert Motor Direction` 打勾 (設為 True)，寫入後再試一次。

? ??????? (`main.c`)

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : 最基礎的油門直通測試 (四輪同等輸出)
  ******************************************************************************
  */
/* USER CODE END Header */
```

```
#include "main.h"

/* USER CODE BEGIN PD */
// --- 測試用安全參數 ---
// 單顆馬達最大電流限制 (測試階段建議設小一點，例如 5A = 5000mA)
#define TEST_MAX_CURRENT_MA   5000

// 油門死區 (0~4095 之間，小於此值馬達不輸出，防止雜訊抖動)
#define THROTTLE_DEADZONE     150

// VESC CAN ID 定義
#define VESC_ID_FRONT_L       0xB1
#define VESC_ID_FRONT_R       0xB2
#define VESC_ID_REAR_L        0xA3
#define VESC_ID_REAR_R        0xA4

#define CAN_PACKET_SET_CURRENT 1
/* USER CODE END PD */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
CAN_HandleTypeDef hcan1;

/* USER CODE BEGIN PV */
uint16_t throttle_adc = 0;
int32_t  target_current = 0;
/* 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_CAN1_Init(void);

/* USER CODE BEGIN PFP */
void VESC_Send_Current(uint8_t controller_id, int32_t current_ma);
void Basic_Drive_Loop(void);
/* USER CODE END PFP */

int main(void)
```

```
{
  HAL_Init();
  SystemClock_Config();

  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_CAN1_Init();

  /* USER CODE BEGIN 2 */
  // 啟動 CAN 總線
  HAL_CAN_Start(&hcan1);

  // 啟動 ADC 連續轉換
  HAL_ADC_Start(&hadc1);
  /* USER CODE END 2 */

  /* Infinite loop */
  while (1)
  {
      Basic_Drive_Loop();
  }
}

/* USER CODE BEGIN 4 */

/**
 * @brief 最基礎的直通驅動迴圈 (100Hz)
 */
void Basic_Drive_Loop(void)
{
    static uint32_t last_tick = 0;
    uint32_t current_tick = HAL_GetTick();

    // 限制執行頻率為 100Hz (每 10ms 執行一次即可，測試階段不需太快)
    if ((current_tick - last_tick) < 10) {
        return;
    }
    last_tick = current_tick;

    // 1. 讀取油門 ADC
```

```
    throttle_adc = HAL_ADC_GetValue(&hadc1);

    // 2. 加入死區與計算目標電流
    if (throttle_adc < THROTTLE_DEADZONE) {
        target_current = 0;
    } else {
        // 將 ADC 值 (扣除死區後) 映射到目標電流
        // 公式: (當前ADC - 死區) * 最大電流 / (4095 - 死區)
        uint32_t active_adc = throttle_adc - THROTTLE_DEADZONE;
        target_current = (active_adc * TEST_MAX_CURRENT_MA) / (4095 - THROTTLE_DEADZONE);
    }

    // 3. 將相同的電流指令，依序發送給 4 顆馬達
    // 這裡加上微小的延遲(1ms)避免 CAN Bus 瞬間塞滿 4 個封包
    VESC_Send_Current(VESC_ID_FRONT_L, target_current);
    HAL_Delay(1);

    VESC_Send_Current(VESC_ID_FRONT_R, target_current);
    HAL_Delay(1);

    VESC_Send_Current(VESC_ID_REAR_L,  target_current);
    HAL_Delay(1);

    VESC_Send_Current(VESC_ID_REAR_R,  target_current);
}

/**
 * @brief 發送電流命令給 VESC (含防卡死保護)
 */
void VESC_Send_Current(uint8_t controller_id, int32_t current_ma)
{
    CAN_TxHeaderTypeDef TxHeader;
    uint32_t TxMailbox;
    uint8_t TxData[4];

    TxHeader.ExtId = (CAN_PACKET_SET_CURRENT << 8) | controller_id;
    TxHeader.IDE = CAN_ID_EXT;
    TxHeader.RTR = CAN_RTR_DATA;
    TxHeader.DLC = 4;
```

```
    TxData[0] = (uint8_t)(current_ma >> 24);
    TxData[1] = (uint8_t)(current_ma >> 16);
    TxData[2] = (uint8_t)(current_ma >> 8);
    TxData[3] = (uint8_t)(current_ma);

    uint32_t timeout = 0;
    // 等待 Mailbox 有空位
    while (HAL_CAN_GetTxMailboxesFreeLevel(&hcan1) == 0)
    {
        timeout++;
        if (timeout > 50000) return; // 超時放棄, 防止當機
    }
    HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox);
}

/* USER CODE END 4 */
```

STM32 ??????? VESC ???? (???????? CAN Buffer)

STM32 ??????? VESC ???? (???????? CAN Buffer)

? ?????????

本系統使用 STM32 透過 CAN Bus 控制 4 顆 VESC 驅動馬達，並整合了方向盤絕對值編碼器，實現以下進階車輛動態控制：

1. 動態模式切換 **(4WD/RWD)**: 低速時啟動 4WD 增加起步循跡性；高速巡航時（ > 10000 ERPM ）前輪斷電利用單向離合器滑行，後輪切換為 100% 功率輸出。
2. 軟體環形緩衝區 **(Ring Buffer)**: 針對 STM32 傳統 CAN 不支援 DMA 的硬體限制，實作了中斷專用的 Ring Buffer。接收中斷能在數微秒內完成資料搬移，實現 **「非同步解析」**，徹底杜絕 CAN Bus 高負載時的漏包與主迴圈卡頓問題。
3. 阿克曼電子差速 **(Ackermann E-Diff)**：利用方向盤角度動態計算車輛過彎時的內外輪迴轉半徑（依據軸距與輪距）。外側輪獲得較高比例電流，內側輪減少電流，解決車輛推頭並提升過彎靈敏度。

? ????? (`main.c`)

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : STM32 VESC 4WD/RWD Dynamic Control System with E-Diff
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
```

```
#include <math.h> // 提供電子差速所需的 tanf() 與 fabs()

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define POWER_LIMIT_W         1000    // 電池總功率限制 1000W
#define BATTERY_VOLTAGE_NOM   48      // 標稱電壓 48V

#define MAX_TOTAL_CURRENT_MA  ((POWER_LIMIT_W * 1000) / BATTERY_VOLTAGE_NOM)

// 速度切換閾值 (ERPM: Electrical RPM)
#define ERPM_THRESHOLD_RWD    10000
#define ERPM_HYSTERESIS       500     // 遲滯區間

// VESC CAN ID 定義 (需與 VESC Tool 設定一致)
#define VESC_ID_FRONT_L       0xB1
#define VESC_ID_FRONT_R       0xB2
#define VESC_ID_REAR_L        0xA3
#define VESC_ID_REAR_R        0xA4

// VESC CAN Packet ID
#define CAN_PACKET_SET_CURRENT 1
#define ENCODER_RESOLUTION     4096.0f
#define STEERING_CENTER_OFFSET 2048.0f

// 車體物理尺寸 (電子差速計算用, 單位: 公尺)
#define TRACK_WIDTH_M         0.8f    // 輪距 (左右輪中心距離)
#define WHEELBASE_M           1.2f    // 軸距 (前後軸中心距離)

// CAN 軟體環形緩衝區大小 (取代 DMA)
#define CAN_RX_BUFFER_SIZE    16
/* USER CODE END PD */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;
```

```
CAN_HandleTypeDef hcan1;

/* USER CODE BEGIN PV */
int32_t current_erpm = 0;
uint16_t throttle_adc = 0;
int32_t target_total_ma = 0;
uint16_t speed_adc = 0;

float current_angle = 0.0f;
float steering_angle = 0.0f;    // -180 ~ +180 度 (電子差速輸入)
uint32_t raw_encoder_value = 0;

// 軟體 Ring Buffer 結構宣告
typedef struct {
    CAN_RxHeaderTypeDef header;
    uint8_t data[8];
} CAN_RxPacket_t;

CAN_RxPacket_t can_rx_buffer[CAN_RX_BUFFER_SIZE];
volatile uint8_t can_rx_head = 0; // 寫入指標 (中斷端)
volatile uint8_t can_rx_tail = 0; // 讀取指標 (主程式端)
/* 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_CAN1_Init(void);
static void MX_ADC2_Init(void);

/* USER CODE BEGIN PFP */
typedef enum {
    MODE_4WD,
    MODE_RWD
} DriveMode_t;

DriveMode_t drive_mode = MODE_4WD;

void VESC_Send_Current(uint8_t controller_id, int32_t current_ma);
void Control_Loop_1kHz(void);
```

```
void ENCODE_Step_up(void);
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan);
void Process_CAN_Rx_Buffer(void);
void Apply_Electronic_Differential(float steering_angle_deg, int32_t axle_total_current,
int32_t *cmd_left, int32_t *cmd_right);
/* USER CODE END PFP */

/**
  * @brief  The application entry point.
  */
int main(void)
{
  HAL_Init();
  SystemClock_Config();

  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_CAN1_Init();
  MX_ADC2_Init();

  /* USER CODE BEGIN 2 */
  HAL_CAN_Start(&hcan1);
  HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING);

  // 啟動 ADC 連續轉換模式
  HAL_ADC_Start(&hadc1);
  HAL_ADC_Start(&hadc2);
  /* USER CODE END 2 */

  /* Infinite loop */
  while (1)
  {
      Process_CAN_Rx_Buffer(); // 隨時消化 Ring Buffer 內的 CAN 訊號
      Control_Loop_1kHz();     // 主控制邏輯
  }
}

/* USER CODE BEGIN 4 */

/**
```

```
 * @brief 電子差速分配 (Ackermann E-Diff)
 */
void Apply_Electronic_Differential(float steering_angle_deg, int32_t axle_total_current,
int32_t *cmd_left, int32_t *cmd_right)
{
    // 直行防呆區間 (轉角小於 3 度視為直行，電流平分)
    if (fabs(steering_angle_deg) < 3.0f) {
        *cmd_left  = axle_total_current / 2;
        *cmd_right = axle_total_current / 2;
        return;
    }

    // 角度轉弧度
    float theta_rad = fabs(steering_angle_deg) * (3.1415926f / 180.0f);

    // 計算假想中心迴轉半徑
    float R_center = WHEELBASE_M / tanf(theta_rad);

    // 計算內外側輪半徑
    float r_inner = R_center - (TRACK_WIDTH_M / 2.0f);
    float r_outer = R_center + (TRACK_WIDTH_M / 2.0f);

    if (r_inner < 0.1f) r_inner = 0.1f; // 防奇異點

    // 依半徑比例分配電流
    float total_r = r_inner + r_outer;
    int32_t current_inner = (int32_t)(axle_total_current * (r_inner / total_r));
    int32_t current_outer = (int32_t)(axle_total_current * (r_outer / total_r));

    // 依據左右轉方向賦予對應輪胎
    if (steering_angle_deg > 0.0f) {
        // 右轉: 右輪是內側
        *cmd_right = current_inner;
        *cmd_left  = current_outer;
    } else {
        // 左轉: 左輪是內側
        *cmd_left  = current_inner;
        *cmd_right = current_outer;
    }
}
```

```
/**
 * @brief □□□□□ (□ 500Hz)
 */
void Control_Loop_1kHz(void)
{
    static uint32_t last_control_tick = 0;
    uint32_t current_tick = HAL_GetTick();

    if ((current_tick - last_control_tick) < 2) {
        return;
    }
    last_control_tick = current_tick;

    throttle_adc = HAL_ADC_GetValue(&hadc1);
    speed_adc = HAL_ADC_GetValue(&hadc2);
    current_erpm = ((int32_t)speed_adc * 20000) >> 12;

    target_total_ma = ((int32_t)throttle_adc * MAX_TOTAL_CURRENT_MA) >> 12;

    int32_t abs_erpm = (current_erpm < 0) ? -current_erpm : current_erpm;

    if (drive_mode == MODE_4WD) {
        if (abs_erpm > (ERPM_THRESHOLD_RWD + ERPM_HYSTERESIS)) {
            drive_mode = MODE_RWD;
        }
    } else {
        if (abs_erpm < (ERPM_THRESHOLD_RWD - ERPM_HYSTERESIS)) {
            drive_mode = MODE_4WD;
        }
    }

    int32_t current_front_axle = 0;
    int32_t current_rear_axle = 0;

    if (drive_mode == MODE_4WD) {
        current_front_axle = target_total_ma / 3;
        current_rear_axle  = target_total_ma - current_front_axle;
    } else {
        current_front_axle = 0;
```

```
        current_rear_axle  = target_total_ma;
    }

    // 計算單顆電機電流 (導入電子差速)
    int32_t cmd_front_L = 0, cmd_front_R = 0;
    int32_t cmd_rear_L = 0, cmd_rear_R = 0;

    Apply_Electronic_Differential(steering_angle, current_front_axle, &cmd_front_L,
&cmd_front_R);
    Apply_Electronic_Differential(steering_angle, current_rear_axle,  &cmd_rear_L,
&cmd_rear_R);

    // 輪詢發送 CAN, 避免 Bus Flooding
    static uint8_t motor_step = 0;
    switch(motor_step)
    {
        case 0:
            VESC_Send_Current(VESC_ID_FRONT_L, cmd_front_L);
            motor_step = 1;
            break;
        case 1:
            VESC_Send_Current(VESC_ID_FRONT_R, cmd_front_R);
            motor_step = 2;
            break;
        case 2:
            VESC_Send_Current(VESC_ID_REAR_L,  cmd_rear_L);
            motor_step = 3;
            break;
        case 3:
            VESC_Send_Current(VESC_ID_REAR_R,  cmd_rear_R);
            motor_step = 0;
            break;
    }
}

/**
 * @brief 非同步處理 CAN 緩衝區資料 (置於 main 的 while 迴圈中)
 */
void Process_CAN_Rx_Buffer(void)
{
```

```
    while (can_rx_tail != can_rx_head)
    {
        CAN_RxPacket_t *packet = &can_rx_buffer[can_rx_tail];

        // 解析編碼器資料
        if (packet->header.IDE == CAN_ID_STD && packet->header.StdId == 0x01)
        {
            if (packet->data[0] == 0x07 && packet->data[1] == 0x01 && packet->data[2] == 0x01)
            {
                raw_encoder_value = (uint32_t)((packet->data[6] << 24) |
                                               (packet->data[5] << 16) |
                                               (packet->data[4] << 8)  |
                                                packet->data[3]);

                // 換算為電子差速需要的方向盤絕對角度 (-180 ~ 180)
                float temp_angle = (float)raw_encoder_value - STEERING_CENTER_OFFSET;
                steering_angle = (temp_angle * 360.0f) / ENCODER_RESOLUTION;

                // 處理邊界
                if (steering_angle > 180.0f) {
                    steering_angle -= 360.0f;
                } else if (steering_angle < -180.0f) {
                    steering_angle += 360.0f;
                }
            }
        }
        can_rx_tail = (can_rx_tail + 1) % CAN_RX_BUFFER_SIZE;
    }
}

/**
 * @brief CAN 接收中斷 (極速存入 Buffer 隨即離開)
 */
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
    if (hcan->Instance == CAN1)
    {
        uint8_t next_head = (can_rx_head + 1) % CAN_RX_BUFFER_SIZE;

        if (next_head != can_rx_tail)
```

```
        {
            if (HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0,
                                     &can_rx_buffer[can_rx_head].header,
                                     can_rx_buffer[can_rx_head].data) == HAL_OK)
            {
                can_rx_head = next_head;
            }
        }
        else
        {
            // □ Buffer □□□□□□□□ FIFO □□□□
            CAN_RxHeaderTypeDef dummy_header;
            uint8_t dummy_data[8];
            HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &dummy_header, dummy_data);
        }
    }
}

void VESC_Send_Current(uint8_t controller_id, int32_t current_ma)
{
    CAN_TxHeaderTypeDef TxHeader;
    uint32_t TxMailbox;
    uint8_t TxData[4];

    TxHeader.ExtId = (CAN_PACKET_SET_CURRENT << 8) | controller_id;
    TxHeader.IDE = CAN_ID_EXT;
    TxHeader.RTR = CAN_RTR_DATA;
    TxHeader.DLC = 4;

    TxData[0] = (uint8_t)(current_ma >> 24);
    TxData[1] = (uint8_t)(current_ma >> 16);
    TxData[2] = (uint8_t)(current_ma >> 8);
    TxData[3] = (uint8_t)(current_ma);

    uint32_t timeout = 0;
    while (HAL_CAN_GetTxMailboxesFreeLevel(&hcan1) == 0)
    {
        timeout++;
        if (timeout > 50000) return;
    }
```

```
    HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox);
}

void ENCODE_Step_up(void)
{
    CAN_TxHeaderTypeDef TxHeader;
    uint32_t TxMailbox;
    uint8_t TxData[8];

    TxHeader.StdId = 0x01;
    TxHeader.IDE = CAN_ID_STD;
    TxHeader.RTR = CAN_RTR_DATA;
    TxHeader.DLC = 4;

    TxData[0] = 0x04;
    TxData[1] = 0x01;
    TxData[2] = 0x04;
    TxData[3] = 0xAA;
    HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox);

    TxHeader.DLC = 5;
    TxData[0] = 0x05;
    TxData[1] = 0x01;
    TxData[2] = 0x05;
    TxData[3] = 0x88;
    TxData[4] = 0x13;
    HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox);
}
/* USER CODE END 4 */
```

STM32 ????? VESC ???? (4WD_RWD ????)

? ?????????

本專案使用STM32作為核心控制器，透過CAN Bus與4顆VESC進行通訊，實現四輪獨立驅動與動態扭矩分配。

- 電池系統：48V標稱電壓，總輸出功率限制1000W。
- 前輪配置：250W減速電機(內建單向離合器，具備滑行優勢)。
- 後輪配置：500W減速電機。
- 輪胎規格：20吋車輪，減速比1:4.4，馬達額定轉速390 RPM。
- 控制策略：低速4WD起步(1:2扭矩分配) $\rightarrow$ 高速切換純RWD巡航(前輪斷電滑行，後輪滿功率輸出)。

?? ????????? (Bug Fixes & Improvements)

1. 修復**CAN**發送死迴圈：在`VESC_Send_Current`增加Timeout機制，防止VESC沒開機或斷線時導致MCU系統卡死。
2. 修復記憶體溢位**(Buffer Overflow)**：在`ENCODE_Step_up`中將`TxData`陣列加大至8 bytes，並修正DLC (Data Length Code)設定。
3. 優化**ADC**讀取：將`HAL_ADC_Start()`移出`while(1)`迴圈，利用硬體連續轉換模式(Continuous Mode)提高穩定性。
4. 增加**CAN ID**濾波保護：在接收中斷`HAL_CAN_RxFifo0MsgPendingCallback`內加入`StdId`檢查，防止雜訊誤觸發編碼器數值更新。

? ????? (`main.c`)

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
```

```
  * @brief          : STM32 VESC 4WD/RWD Dynamic Control System
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define POWER_LIMIT_W         1000    // 電池總功率限制 1000W
#define BATTERY_VOLTAGE_NOM   48      // 標稱電壓 48V

// 最大總電流 (mA)
#define MAX_TOTAL_CURRENT_MA  ((POWER_LIMIT_W * 1000) / BATTERY_VOLTAGE_NOM)

// 速度切換閾值 (ERPM: Electrical RPM)
// 依據 20吋輪、4.4減速比、390rpm 重新估算切換點
#define ERPM_THRESHOLD_RWD    10000
#define ERPM_HYSTERESIS       500     // 遲滯區間，避免臨界點頻繁切換

// VESC CAN ID 定義 (需與 VESC Tool 設定一致)
#define VESC_ID_FRONT_L       0xB1
#define VESC_ID_FRONT_R       0xB2
#define VESC_ID_REAR_L        0xA3
#define VESC_ID_REAR_R        0xA4

// VESC CAN Packet ID
#define CAN_PACKET_SET_CURRENT 1
#define ENCODER_RESOLUTION     4096.0f
#define STEERING_CENTER_OFFSET 2048.0f
/* USER CODE END PD */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;
CAN_HandleTypeDef hcan1;
```

```
/* USER CODE BEGIN PV */
int32_t current_erpm = 0;       // 平均轉速 (由 CAN RX 更新)
uint16_t throttle_adc = 0;      // 0 - 4095
int32_t target_total_ma = 0;    // 總目標電流 (mA)
uint16_t speed_adc = 0;         // 模擬速度的 ADC

float current_angle = 0.0f;     // 0 ~ 360 度的絕對角度
float steering_angle = 0.0f;    // -180 ~ +180 度的方向盤實際轉角
uint32_t raw_encoder_value = 0; // 0 ~ 4095
/* 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_CAN1_Init(void);
static void MX_ADC2_Init(void);

/* USER CODE BEGIN PFP */
// 狀態機定義: 四驅與後驅模式
typedef enum {
    MODE_4WD,
    MODE_RWD
} DriveMode_t;

DriveMode_t drive_mode = MODE_4WD;

void VESC_Send_Current(uint8_t controller_id, int32_t current_ma);
void Control_Loop_1kHz(void);
void ENCODE_Step_up(void);
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan);
/* USER CODE END PFP */

/**
  * @brief  The application entry point.
  */
int main(void)
{
  /* MCU Configuration--------------------------------------------------------*/
  HAL_Init();
```

```
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_CAN1_Init();
  MX_ADC2_Init();

  /* USER CODE BEGIN 2 */
  // 啟動 CAN 總線與中斷
  HAL_CAN_Start(&hcan1);
  HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING);

  // 【優化】將 ADC 啟動移出 while(1) 迴圈，使用連續轉換模式讓硬體自動更新
  HAL_ADC_Start(&hadc1);
  HAL_ADC_Start(&hadc2);
  /* USER CODE END 2 */

  /* Infinite loop */
  while (1)
  {
      Control_Loop_1kHz();
  }
}

/* USER CODE BEGIN 4 */

/**
 * @brief 主控制迴圈 (使用輪詢方式發送 CAN 避免 Bus 壅塞)
 */
void Control_Loop_1kHz(void)
{
    static uint32_t last_control_tick = 0;
    uint32_t current_tick = HAL_GetTick();

    // 控制執行頻率 (大約 500Hz)
    if ((current_tick - last_control_tick) < 2) {
        return;
    }
    last_control_tick = current_tick;
```

```
// 1. 讀取油門與速度模擬 ADC (0 ~ 4095)
throttle_adc = HAL_ADC_GetValue(&hadc1);
speed_adc = HAL_ADC_GetValue(&hadc2);
current_erpm = ((int32_t)speed_adc * 20000) >> 12;

// 2. 計算總需求電流 (mA)
target_total_ma = ((int32_t)throttle_adc * MAX_TOTAL_CURRENT_MA) >> 12;

// 3. 判斷模式 (根據 ERPM 絕對值)
int32_t abs_erpm = (current_erpm < 0) ? -current_erpm : current_erpm;

if (drive_mode == MODE_4WD) {
    if (abs_erpm > (ERPM_THRESHOLD_RWD + ERPM_HYSTERESIS)) {
        drive_mode = MODE_RWD;
    }
} else {
    if (abs_erpm < (ERPM_THRESHOLD_RWD - ERPM_HYSTERESIS)) {
        drive_mode = MODE_4WD;
    }
}

// 4. 動態電流分配 (Distribute Current)
int32_t current_front_axle = 0;
int32_t current_rear_axle = 0;

if (drive_mode == MODE_4WD) {
    // --- 4WD 模式 (起步高扭力) ---
    // 前 250W + 後 500W, 比例 1:2
    current_front_axle = target_total_ma / 3;
    current_rear_axle  = target_total_ma - current_front_axle;
} else {
    // --- RWD 模式 (高速巡航) ---
    // 前輪斷電 (利用離合器滑行), 後輪全功率輸出
    current_front_axle = 0;
    current_rear_axle  = target_total_ma;
}

// 5. 計算單顆電機電流並輪詢發送
int32_t cmd_front = current_front_axle / 2;
int32_t cmd_rear  = current_rear_axle / 2;
```

```
    static uint8_t motor_step = 0;
    switch(motor_step)
    {
        case 0:
            VESC_Send_Current(VESC_ID_FRONT_L, cmd_front);
            motor_step = 1;
            break;
        case 1:
            VESC_Send_Current(VESC_ID_FRONT_R, cmd_front);
            motor_step = 2;
            break;
        case 2:
            VESC_Send_Current(VESC_ID_REAR_L,  cmd_rear);
            motor_step = 3;
            break;
        case 3:
            VESC_Send_Current(VESC_ID_REAR_R,  cmd_rear);
            motor_step = 0;
            break;
    }
}

/**
 * @brief □□□□□□□ VESC
 */
void VESC_Send_Current(uint8_t controller_id, int32_t current_ma)
{
    CAN_TxHeaderTypeDef TxHeader;
    uint32_t TxMailbox;
    uint8_t TxData[4];

    TxHeader.ExtId = (CAN_PACKET_SET_CURRENT << 8) | controller_id;
    TxHeader.IDE = CAN_ID_EXT;
    TxHeader.RTR = CAN_RTR_DATA;
    TxHeader.DLC = 4;

    // Big Endian packing
    TxData[0] = (uint8_t)(current_ma >> 24);
    TxData[1] = (uint8_t)(current_ma >> 16);
    TxData[2] = (uint8_t)(current_ma >> 8);
    TxData[3] = (uint8_t)(current_ma);
```

```
    // 【安全優化】加入 Timeout 防護，避免 CAN Bus 異常時系統卡死
    uint32_t timeout = 0;
    while (HAL_CAN_GetTxMailboxesFreeLevel(&hcan1) == 0)
    {
        timeout++;
        if (timeout > 50000) {
            return; // 丟棄此包，保護主程式
        }
    }
    HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox);
}

/**
 * @brief 發送編碼器設定命令
 */
void ENCODE_Step_up(void)
{
    CAN_TxHeaderTypeDef TxHeader;
    uint32_t TxMailbox;
    uint8_t TxData[8]; // 【修正】必須開到 8 bytes 以防溢位

    TxHeader.StdId = 0x01;
    TxHeader.IDE = CAN_ID_STD;
    TxHeader.RTR = CAN_RTR_DATA;
    TxHeader.DLC = 4;

    TxData[0] = 0x04;
    TxData[1] = 0x01;
    TxData[2] = 0x04;
    TxData[3] = 0xAA;
    HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox);

    // 第二包命令
    TxHeader.DLC = 5; // 【修正】長度改為 5
    TxData[0] = 0x05;
    TxData[1] = 0x01;
    TxData[2] = 0x05;
    TxData[3] = 0x88;
    TxData[4] = 0x13; // 寫入第 5 個 byte 不再越界
    HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox);
```

```
}

/**
 * @brief CAN 接收中斷回調函式 (處理編碼器回傳角度)
 */
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
    CAN_RxHeaderTypeDef RxHeader;
    uint8_t RxData[8];

    if (hcan->Instance == CAN1)
    {
        if (HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &RxHeader, RxData) == HAL_OK)
        {
            // 【安全優化】確認封包來源 ID，防止 VESC 雜訊誤更動數據
            if (RxHeader.IDE == CAN_ID_STD && RxHeader.StdId == 0x01)
            {
                if (RxData[0] == 0x07 && RxData[1] == 0x01 && RxData[2] == 0x01)
                {
                    // 小端序解碼
                    raw_encoder_value = (uint32_t)((RxData[6] << 24) |
                                                   (RxData[5] << 16) |
                                                   (RxData[4] << 8)  |
                                                    RxData[3]);
                }
            }
        }
    }
}
/* USER CODE END 4 */
```