TTR9_WSM ?????????-????

撰寫人 **：** 范紹捷 **/**動力組 ***/5~9***代 [TOC]

1. ????

本專案使用 **STM32G431** 系列微控制器，主要功能為:

1. 輪速監測 ：利用霍爾感測器擷取脈衝，透過 TIM3 (ETR模式) 計算轉速 (RPM)。
2. 碟盤 **/**胎溫監測 ：透過 SPI 介面讀取 MAX31856 晶片，擷取 K-Type 熱電偶溫度。
3. **CAN** 通訊收發 ：將上述數據打包透過 FDCAN (Classic Mode) 每 100ms 發送至車載網路 (ID: `0x512`)，並具備接收其他 CAN 節點數據的能力。

---

2. ??????? (Pinout)

功能	腳位 **(Pin)**	**STM32** 設定模式	備註說明
SPI1 SCK	PA5	SPI1_SCK	MAX31856 時鐘線
SPI1 MISO	PA6	SPI1_MISO	MAX31856 資料回傳 (極重要，斷線會讀到 0度)
SPI1 MOSI	PA7	SPI1_MOSI	MAX31856 指令發送
SPI1 CS	PA4	GPIO_Output	片選訊號 (Active Low)
MAX Fault	PA2	GPIO_Input	故障警示，必須開啟 **Pull-Up (**上拉 **)**
輪速訊號	PB4 (或其他)	TIM3_ETR2	接收輪速感測器方波
CAN TX/RX	PA12 / PA11	FDCAN1_TX / RX	連接 CAN Transceiver (需並聯 120Ω 終端電阻)

image

---

3. CubeMX ???? (???? ??)

為了避免程式碼產生後行為異常，CubeMX 的設定請務必再三確認:

1. **FDCAN1** 設定：

- `Frame Format`: **Classic mode** (傳統 CAN 格式，勿選 FD，除非接收端支援)。
- `NVIC Settings`: 必須勾選 **FDCAN1 interrupt 0** (接收中斷才會觸發)。

2. **TIM3 (輪速計數器)**:
   - 選擇 TIM3 ➔ 設定為 `External Clock Mode 1` 使用外部輸入時脈
   - `Clock Source`: `ETR2` 使用外部輸入時脈
   - `Counter Mode`: `Up` 上數
   - `Prescaler`: 設為 **15** (極度重要! 用來過濾車輛震動帶來的高頻雜訊，否則轉速會爆衝到 65535)。
   - `Counter Period`: `65535` 上數上限 (開最大記憶體空間)
   - `Clock Polarity`: `inverted` (若使用 `low` 計數) 正負緣觸發
   - `Clock Filter`: 可設定為 0 或加上 Debounce (依訊號品質)
     upload_0eb357da1ed2f46f202123acd37e848a
     upload_8ee292b24440a0598d2be015df7d23c2
   - □□ 功能程式碼

```
tick_count = TIM2->CNT;     //讀取TIM2的計數器暫存器
TIM2->CNT = 0;              //清空TIM2的暫存器
```

3. **TIM1 (100ms 系統時基鬧鐘)**:
   - `Clock Source`: Internal Clock。
   - 設定 Prescaler 與 Period 使觸發週期為 100ms。
   - `NVIC Settings`: 必須勾選 **TIM1 update interrupt**。

---

4. ?????????

燒入程式前要先進入 `CUBE progemer` 更改 `boot0` 狀態。 image

請嚴格依照註解標示的 `USER CODE BEGIN` 區塊將程式碼填入 `main.c` 中。

A. ???????? (`USER CODE BEGIN 0`)

```
/* USER CODE BEGIN 0 */

// --- 系統控制旗標與變數 ---
volatile uint8_t can_send_flag = 0;     // 100ms 觸發發送旗標
volatile uint32_t raw_wheel_pulse = 0; // 存放 TIM3 擷取到的原始脈衝數

// --- CAN 通訊用變數 ---
FDCAN_RxHeaderTypeDef RxHeader; // 存放 CAN 接收標頭
uint8_t RxData[8];              // 存放 CAN 接收資料
volatile uint16_t received_rpm = 0;    // 測試接收用的變數
```

```
volatile float received_temp = 0.0f;  // 測試接收用的變數

/* USER CODE END 0 */
```

B. ?????? (`USER CODE BEGIN PFP`)

```
/* USER CODE BEGIN PFP */

// 宣告手刻的 SPI 讀寫函式，避免編譯器報錯 (implicit declaration)
uint8_t MAX31856_ReadReg(uint8_t regAddr);
void MAX31856_WriteReg(uint8_t regAddr, uint8_t data);
float Read_MAX31856_Temp_Directly(void);

// 新增：解決斷電重啟失效的強效初始化函式 (改名避免與 Library 衝突)
void MAX31856_PowerOn_Fix(void);

/* USER CODE END PFP */
```

C. ???????? (`USER CODE BEGIN 2`)

這段放在 `main()` 函式內， `while(1)` 迴圈之前。

```
  /* USER CODE BEGIN 2 */

  // === 0. 喚醒 MAX31856 並初始化 (解決斷電重啟讀不到溫度的 Bug) ===
  MAX31856_PowerOn_Fix();

  // === 1. CAN Bus 接收過濾器設定 (沒設會收不到東西) ===
  FDCAN_FilterTypeDef sFilterConfig;
  sFilterConfig.IdType = FDCAN_STANDARD_ID;
  sFilterConfig.FilterIndex = 0;
  sFilterConfig.FilterType = FDCAN_FILTER_MASK;
  sFilterConfig.FilterConfig = FDCAN_FILTER_TO_RXFIFO0;
  sFilterConfig.FilterID1 = 0x000; // 0x000 代表接收所有 ID
  sFilterConfig.FilterID2 = 0x000;
  HAL_FDCAN_ConfigFilter(&hfdcan1, &sFilterConfig);
  HAL_FDCAN_ConfigGlobalFilter(&hfdcan1, FDCAN_REJECT, FDCAN_REJECT, FDCAN_FILTER_REMOTE,
FDCAN_FILTER_REMOTE);
```

```
// === 2. CAN Bus 啟動與中斷開啟 ===
HAL_FDCAN_Start(&hfdcan1);
HAL_FDCAN_ActivateNotification(&hfdcan1, FDCAN_IT_RX_FIFO0_NEW_MESSAGE, 0);

// === 3. SPI MAX31856 狀態檢查 (Debug用) ===
// 透過讀取暫存器確認 SPI 接線是否正常
uint8_t fault_status = MAX31856_ReadReg(0x0F); // 若為 0 代表無硬體錯誤
uint8_t cold_junc = MAX31856_ReadReg(0x0A);    // 讀取冷端(室溫)確認通訊正常

// === 4. 啟動 Timer (計數器與定時器) ===
HAL_TIM_Base_Start(&htim3);      // 啟動輪速計數
HAL_TIM_Base_Start_IT(&htim1);   // 啟動 100ms 中斷定時器

// === 5. CAN 發送標頭設定 ===
TxHeader.Identifier = 0x512; // 發送的 CAN ID
TxHeader.IdType = FDCAN_STANDARD_ID;
TxHeader.TxFrameType = FDCAN_DATA_FRAME;
TxHeader.DataLength = FDCAN_DLC_BYTES_8;
TxHeader.ErrorStateIndicator = FDCAN_ESI_ACTIVE;
TxHeader.BitRateSwitch = FDCAN_BRS_OFF;
TxHeader.FDFormat = FDCAN_CLASSIC_CAN; // 強制設定為傳統 CAN
TxHeader.TxEventFifoControl = FDCAN_NO_TX_EVENTS;
TxHeader.MessageMarker = 0;

/* USER CODE END 2 */
```

D. ????? (`USER CODE BEGIN 3`)

這段放在 `main()` 函式內的 `while(1)` 之中。

```
    /* USER CODE BEGIN 3 */

    // 當 TIM1 (100ms) 中斷舉旗時，執行數據轉換與發送
    if (can_send_flag == 1) {

        // --- 1. 計算 RPM ---
        // 依據你的感測器齒數與時間差換算 (此處假設係數為 15.0)
        float wheel_rpm = (float)raw_wheel_pulse * 15.0f;
```

```
        // --- 2. 讀取溫度 ---
        // 使用底層直讀函式，避免原本 Library 未開啟自動轉換的 Bug
        float current_temp = Read_MAX31856_Temp_Directly();

        // --- 3. 打包資料 (浮點數轉整數) ---
        int16_t send_rpm = (int16_t)wheel_rpm;
        int16_t send_temp = (int16_t)(current_temp * 100); // 放大100倍保留小數

        TxData[0] = send_rpm & 0xFF;        // RPM Low Byte
        TxData[1] = (send_rpm >> 8) & 0xFF; // RPM High Byte
        TxData[2] = send_temp & 0xFF;       // Temp Low Byte
        TxData[3] = (send_temp >> 8) & 0xFF;// Temp High Byte
        TxData[4] = 0x00; // 保留
        TxData[5] = 0x00;
        TxData[6] = 0x00;
        TxData[7] = 0x00;

        // --- 4. 發送 CAN 封包 ---
        HAL_FDCAN_AddMessageToTxFifoQ(&hfdcan1, &TxHeader, TxData);

        // --- 5. 放下旗標 ---
        can_send_flag = 0;
    }

  } // end of while(1)
  /* USER CODE END 3 */
```

E. ?????????? (`USER CODE BEGIN 4`)

這段請放在 `main.c` 檔案的最底端。包含了 **Timer** 中斷、 **CAN** 接收中斷 以及 **SPI** 底層讀寫函式 。

```
/* USER CODE BEGIN 4 */

// ========================================
// 1. Timer 週期中斷 (每 100ms 觸發)
// ========================================
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM1) {
```

```
        // 讀取 TIM3 擷取到的輪速脈衝，然後立刻歸零
        raw_wheel_pulse = __HAL_TIM_GET_COUNTER(&htim3);
        __HAL_TIM_SET_COUNTER(&htim3, 0);

        // 通知主迴圈進行 CAN 發送
        can_send_flag = 1;
    }
}

// ========================================
// 2. CAN 接收中斷 Callback
// ========================================
void HAL_FDCAN_RxFifo0Callback(FDCAN_HandleTypeDef *hfdcan, uint32_t RxFifo0ITs) {
    if ((RxFifo0ITs & FDCAN_IT_RX_FIFO0_NEW_MESSAGE) != RESET) {
        if (HAL_FDCAN_GetRxMessage(hfdcan, FDCAN_RX_FIFO0, &RxHeader, RxData) == HAL_OK) {

            // 處理接收到的資料 (例如解析其他節點送來的 0x512)
            if (RxHeader.Identifier == 0x512) {
                received_rpm = (RxData[1] << 8) | RxData[0];
                int16_t raw_temp = (RxData[3] << 8) | RxData[2];
                received_temp = (float)raw_temp / 100.0f;
            }
        }
        // 處理完畢後，務必重新啟動接收中斷
        HAL_FDCAN_ActivateNotification(hfdcan, FDCAN_IT_RX_FIFO0_NEW_MESSAGE, 0);
    }
}

// ========================================
// 3. MAX31856 暫存器讀取 (Debug/底層測試用)
// ========================================
uint8_t MAX31856_ReadReg(uint8_t regAddr) {
    uint8_t txData[2];
    uint8_t rxData[2] = {0, 0};

    txData[0] = regAddr & 0x7F; // 位址 MSB 為 0 代表 Read
    txData[1] = 0xFF;           // Dummy byte

    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_RESET);
    HAL_SPI_TransmitReceive(&hspi1, txData, rxData, 2, HAL_MAX_DELAY);
```

```
    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_SET);

    return rxData[1];
}

// ========================================
// 4. MAX31856 溫度暴力直讀 (繞過函式庫 Bug)
// ========================================
float Read_MAX31856_Temp_Directly(void) {
    uint8_t txData[4] = {0x0C & 0x7F, 0xFF, 0xFF, 0xFF};
    uint8_t rxData[4] = {0, 0, 0, 0};

    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_RESET);
    HAL_SPI_TransmitReceive(&hspi1, txData, rxData, 4, HAL_MAX_DELAY);
    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_SET);

    uint32_t temp24 = (rxData[1] << 16) | (rxData[2] << 8) | rxData[3];
    temp24 = temp24 >> 5; // 溫度數據佔 19 bit

    if (temp24 & 0x40000) {
        temp24 |= 0xFFF80000; // 處理負數二補數
    }
    return (int32_t)temp24 * 0.0078125f; // 解析度 1/128
}

// ========================================
// 5. MAX31856 暫存器寫入 (喚醒與設定用)
// ========================================
void MAX31856_WriteReg(uint8_t regAddr, uint8_t data) {
    uint8_t txData[2];
    txData[0] = regAddr | 0x80; // MSB = 1 代表寫入模式 (Write)
    txData[1] = data;

    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_RESET);
    HAL_SPI_Transmit(&hspi1, txData, 2, HAL_MAX_DELAY);
    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_SET);
}

// ========================================
// 6. MAX31856 強效初始化 (斷電重啟修復)
```

```
// ========================================
void MAX31856_PowerOn_Fix(void) {
    HAL_Delay(100); // 1. 等待晶片電源穩定
    MAX31856_WriteReg(0x00, 0x80); // 2. 設定 CR0: 開啟自動連續轉換模式
    MAX31856_WriteReg(0x01, 0x03); // 3. 設定 CR1: 指定使用 K-Type 熱電偶
    HAL_Delay(200); // 4. 等待第一次溫度轉換完成
}

/* USER CODE END 4 */
```

---

5. Vector DBC ?? (CAN ????)

此為本專案專屬的 `.dbc` 檔案內容，可直接載入 PCAN, CANoe 或其他 CAN 軟體以解析數據:

```
VERSION ""

NS_ :
□NS_DESC_
□CM_
□BA_DEF_
□BA_
□VAL_
□CAT_DEF_
□CAT_
□FILTER
□BA_DEF_DEF_
□EV_DATA_
□ENVVAR_DATA_
□SGTYPE_
□SGTYPE_VAL_
□BA_DEF_SGTYPE_
□BA_SGTYPE_
□SIG_TYPE_REF_
□VAL_TABLE_
□SIG_GROUP_
□SIG_VALTYPE_
□SIGTYPE_VALTYPE_
□BO_TX_BU_
```

```
□BA_DEF_REL_
□BA_REL_
□BA_DEF_DEF_REL_
□BU_SG_REL_
□BU_EV_REL_
□BU_BO_REL_
□SG_MUL_VAL_

BS_:

BU_: STM32_Node DASHBOARD

BO_ 1298 Wheel_Data: 8 STM32_Node
 SG_ Wheel_RPM : 0|16@1- (1,0) [-32768|32767] "rpm" Vector__XXX
 SG_ Brake_Temp : 16|16@1- (0.01,0) [-270|1300] "degC" Vector__XXX

CM_ SG_ 1298 Wheel_RPM "Raw Wheel Speed from Hall Sensor";
CM_ SG_ 1298 Brake_Temp "K-Type Thermocouple Temperature via MAX31856";
```

---

6. ?? ??????? Debug ??

1. 編譯錯誤: `initializer element is not constant`
   - 原因：在 C 語言中，把函式呼叫（如 `MAX31856_ReadReg()`）放在 `main()` 之外的全域變數區。
   - 解法：全域變數只能用常數初始化。執行函式賦值必須寫在 `main()` 或其他函式內部。
2. 連結錯誤: `undefined reference to xxx`
   - 原因：只有宣告（Prototype），卻沒有把函式的本體（Body）貼進 `main.c` 裡，導致最後連結 `.elf` 檔時找不到程式碼。
3. **SPI** 讀取溫度永遠是 **0** 度
   - 排查步驟：利用手寫底層函式讀取冷端補償暫存器 (`0x0A`)。如果讀得到室溫（例如 `28`），代表硬體線路完美無缺。
   - 真兇：使用的 GitHub 第三方函式庫未正確設定晶片進入「連續轉換模式 (Automatic Conversion Mode)」。改用底層硬讀函式即可解決。
4. **FDCAN** 接收不到任何資料
   - 原因： FDCAN 預設會「拒絕」所有未經 Filter 允許的封包。
   - 解法：在啟動 CAN 前，必須宣告並設定 `FDCAN_FilterTypeDef`，且中斷處理完畢後，必須重新呼叫 `HAL_FDCAN_ActivateNotification` 重啟接收。
5. MAX31856 斷電重啟後讀不到溫度 / 需要按 Reset 才能動
   - 原因：微控制器開機速度比 SPI 晶片快，或是直接重啟時硬體暫存器狀態沒有正確回到預設值，導致無法進入工作狀態。

- 解法：在程式初啟動時給予 `HAL_Delay(100)` 讓電源穩定，再自己實作強制寫入暫存器設定的函式（如本範例中的 `MAX31856_PowerOn_Fix`），並再等待 `200ms` 讓第一筆轉換完成。

6. 編譯錯誤: `conflicting types for 'MAX31856_Init'`
   - 原因：自己撰寫的初始化函式名稱，剛好跟 `#include` 進來的第三方 `.h` 檔案內定義好的函式撞名了，且兩者參數不同導致 `C` 語言編譯器崩潰。
   - 解法：將自己定義的函式改名（例如改成 `MAX31856_PowerOn_Fix`）即可完美避開衝突。

Revision #3
Created 2026-04-01 02:06:10 UTC by TaipeiTechRacing
Updated 2026-04-11 14:36:20 UTC by AI Agent