HT32 MCU SPI 應用範例

文件編號：AN0616TC

簡介

SPI(Serial Peripheral Interface)由 4 個腳位組成的同步串列通訊介面，它的特色是高速、全雙工進行資料的傳輸，這種架構是由一個 Master(主設備)和一個或多個 Slave(從設備)組成。SPI 的典型應用包括：快閃記憶、EEPROM、SRAM、SD 卡與 TFT-LCD 等設備。

本文首先介紹 SPI 的通訊協定，接著針對 SPI 範例所需的資源下載與準備，然後介紹 SPI 範例的使用方法與範例展示，最後對 SPI 範例的設定變更與常用問題提供使用者參考。

縮寫說明

- API : Application Programming Interface (應用程式介面)
- SPI : Serial Peripheral Interface (串列週邊介面)
- CPOL : Clock Polarity (時脈極性)
- CPHA : Clock Phase (時脈相位)
- MOSI : Master Output / Slave Input (Master 發送資料 / Slave 接收資料)
- MISO : Master Input / Slave Output (Master 接收資料 / Slave 發送資料)
- SEL : Slave Select (從設備選擇)
- SCK : Serial Clock (串列時脈)
- MSB : Most Significant Bit (最高有效位)
- LSB : Least Significant Bit (最低有效位)

通訊協定

SPI 硬體介面

SPI 由 4 個訊號線組成分別，詳細介紹如下：

- SCK：時脈訊號，由 Master 提供同步時脈訊號給 Slave，Master 與 Slave 需根據 SCK 的變化對資料進行採樣或閂鎖(Latch)，SCK 即為 SPI 傳輸速率。
- MOSI：資料訊號，全稱 Master Output Slave Input，此訊號線由 Master 控制，用於傳輸資料給 Slave 接收。
- MISO：資料訊號，全稱 Master Input Slave Output，此訊號線由 Slave 控制，用於傳輸資料給 Master 接收。
- SEL：Slave 選擇訊號，此訊號線由 Master 控制，根據 SEL 電位告知 Slave 本次的傳輸是否有效。在一個 Master 連接多個 Slave 情況下，透過控制 I/O 模擬 SEL 作為仲裁，實現 Master 可連接多個 Slave 功能。

SPI 接綫方法與訊號方向，如下圖所示。

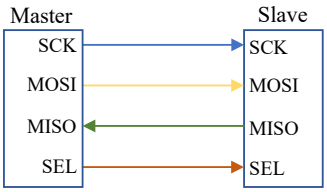

圖 1. SPI 接線圖

Holtek 提供了 SPI 的模組化 API，名稱為 Module_SPI，使用者可以直接透過 Module_SPI 完成繁瑣的硬體設定與 SPI 傳輸時的資料處理。Module_SPI 的功能整理如下：

1. 支援四種(CPOL、CPHA)資料傳輸格式，詳細請參考“SPI 通訊時序”章節。
2. 支援 SEL 有效準位設定。
3. 支援 MSB(Most Significant Bit)先傳輸或 LSB(Least Significant Bit)先傳輸。
4. 支援 Master 模式與 Slave 模式。
5. 支援 SPI 中斷或 PDMA 兩種資料搬運模式。

SPI 通訊時序

SPI 的通訊協定有兩個重要的參數 CPOL/CPHA，說明如下。

1、時脈極性 (Clock Polarity，CPOL)

- CPOL=0，時脈極性為 0 代表 SCK 空閒狀態為“低電位”
- CPOL=1，時脈極性為 1 代表 SCK 空閒狀態為“高電位”

2、時脈相位 (Clock Phase，CPHA)

- CPHA=0，時脈相位為 0 代表數據會在“第一個 SCK 改變準位”時被採樣
- CPHA=1，時脈相位為 1 代表數據會在“第二個 SCK 改變準位”時被採樣

透過上述的 CPOL、CPHA 可產生四種不同的資料採樣格式。

- CPOL=0、CPHA=0：數據會在 SCK 上升邊緣被採樣，在 SCK 下降邊緣時改變數據準位並在半週期內完成資料閂鎖。

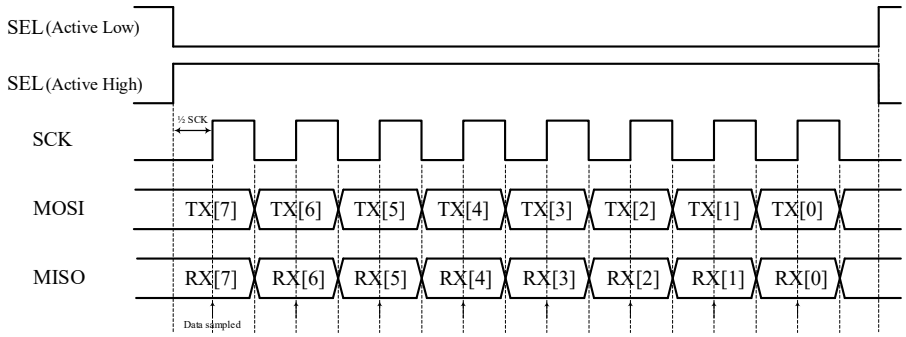

圖 2. CPOL=0、CPHA=0

- CPOL=0、CPHA=1：數據會在 SCK 下降邊緣被採樣，在 SCK 上升邊緣時改變數據準位並在半周期內完成資料閂鎖。

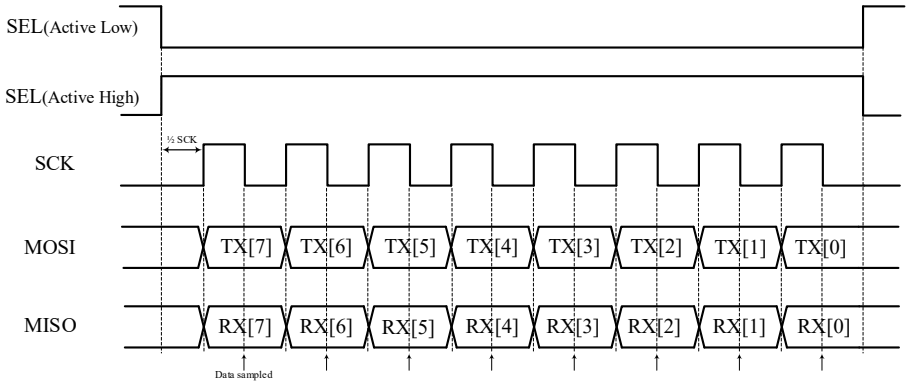

圖 3. CPOL=0、CPHA=1

- CPOL=1、CPHA=0：數據會在 SCK 下降邊緣被採樣，在 SCK 上升邊緣時改變數據準位並在半周期內完成資料閂鎖。

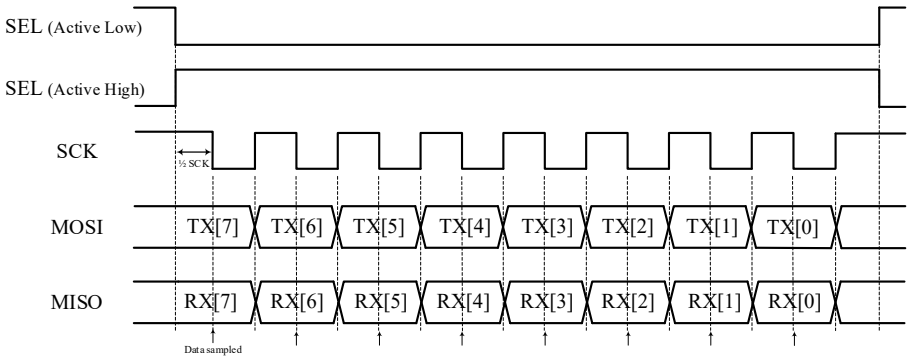

圖 4. CPOL=1、CPHA=0

- CPOL=1、CPHA=1：數據會在 SCK 上升邊緣被採樣，在 SCK 下降邊緣時改變數據準位並在半週期內完成資料閂鎖。

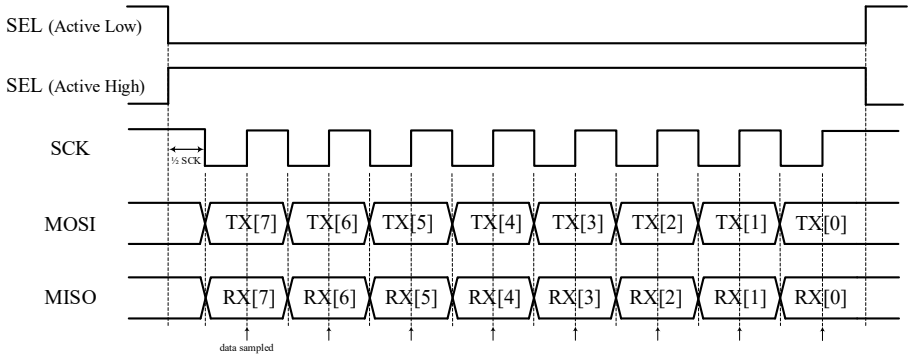

圖 5. CPOL=1、CPHA=1

資源下載與準備

本章節介紹使用到的範例程式與軟體工具，並說明如何對目錄及檔案路徑進行配置。

Firmware Library

使用範例前，請先確認已下載最新的 Holtek HT32 Firmware Library。下載位置如下方連結，可以選擇 HT32 不同系列的 Firmware Library，例如：HT32F5xxxx 系列(HT32_M0p_Vyyyymmdd.zip)或 HT32F1xxxx 系列(HT32_M3_Vyyyymmdd.zip)等，下載後請將其解壓縮。

壓縮檔內包含多個資料夾，可分類為 Document、Firmware Library、Tools 等項目，放置路徑以圖 6 作為舉例。在 Firmware_Library 資料夾內會看到 HT32 Firmware Library 壓縮檔，檔名為 HT32_STD_xxxxx_FWLib_Vm.n.r_s.zip。

下載路徑：https://mcu.holtek.com.tw/ht32/resource/

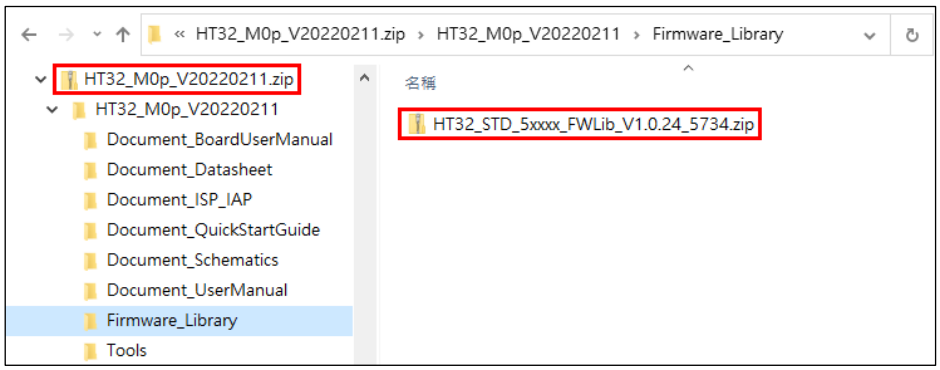

圖 6. HT32_M0p_Vyyyymmdd.zip 內容

應用範例程式

範例程式請透過下方連結下載，此範例程式被打包成一個 ZIP 壓縮檔，檔名為 HT32_APPFW_xxxxx_APPCODENAME_Vm.n.r_s.zip。檔案名稱格式說明，請看下圖 7。

下載路徑：https://mcu.holtek.com.tw/ht32/app.fw/Module_SPI

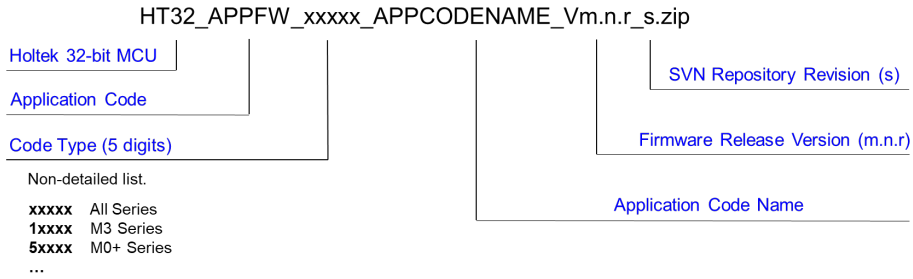

圖 7. 範例程式檔名介紹

檔案及目錄配置

由於 Application Code 範例程式不包含 HT32 Firmware Library 相關檔案，因此在開始編譯之前，需要將 Application Code 及 Firmware Library 解壓縮的檔案放置到正確的路徑。

Application Code 壓縮檔內通常包含 1 個至數個資料夾(例如：application、library)，放置路徑以圖 8 作爲舉例，只需將 application 等資料夾放到 HT32 Firmware Library 根目錄下，就可以完成檔案路徑的配置，以圖 9 作為舉例。

您也可以將 Application Code 及 HT32 Firmware Library 兩個壓縮檔，同時解壓縮到相同路徑，此操作具有同樣的效果。

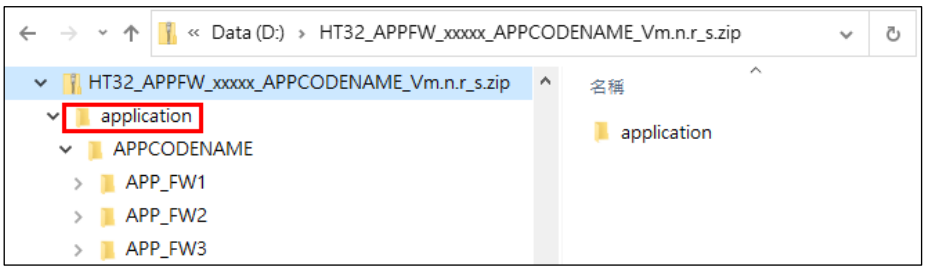

圖 8. HT32_APPFW_xxxxx_APPCODENAME_Vm.n.r_s.zip 內容

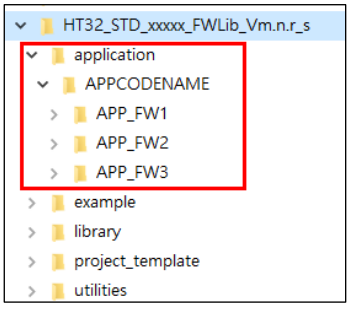

圖 9. 解壓縮路徑

終端機軟體

Application Code 範例程式可以透過 COM Port 傳遞訊息，作為功能選擇或狀態顯示。而主機端需要事先安裝終端機軟體，用戶可以選擇適合的連綫軟體，或者使用免費授權軟體，例如 Tera Term。

範例程式中 UART 介面配置，字長 8-bit，無奇偶校驗，1 個停止位元，串列傳輸速率設為 115200bps。

功能說明

本章節介紹 Application Code 的功能說明，包含目錄結構、API 架構、資源佔用等資訊。

目錄結構

Application Code 內包含資料夾 Application，相關檔案及說明如下表。

資料夾/檔案名稱	說明
\\application\Module_SPI\Master	
_CreateProject.bat	創建範例專案
_ProjectSource.ini	加入 Module_SPI 相關檔案
ht32_board_config.h	IC 週邊 I/O 相關設定
ht32fxxxxx_01_it.c	中斷控制相關程式
main.c	範例主程式
spi_module_config.h	Module_SPI 相關設定
\\application\Module_SPI\Slave	
_CreateProject.bat	創建範例專案
_ProjectSource.ini	加入 Module_SPI 相關檔案
ht32_board_config.h	IC 週邊 I/O 相關設定
ht32fxxxxx_01_it.c	中斷控制相關程式
main.c	範例主程式
spi_module_config.h	Module_SPI 相關設定
\\utilities\middleware	
spi_module.c* spi_module.h*	Module_SPI 原始碼

表 1. 範例程式目錄結構

註：Application Code 需用到 spi_module.c/h 有最低 Firmware Library 版本限制，版本限制會依照更新而不定時變動，查詢當前 Firmware Library 的版本限制請搜尋關鍵字“Dependency check”。若 Firmware Library 版本不符要求，請從“Firmware Library”章節中下載最新版本。

API 架構

各 API 必有一個重要參數 CH，CH 是一個“索引”決定對哪一組 SPI 進行控制，我們將其簡稱爲 SPI CH (SPI Channel)，而各組 SPI CH 與實體 HT_SPIx 的對應關係設定在 ht32_board_config.h 內。目前最多支援 4 組 SPI CH，因此定義了 4 個常數符號整理如下，作為參數 CH 給 API 作爲控制的依據。

- SPI_API_CH0：輸入參數，代表對第 1 組 SPI 進行控制或設定
- SPI_API_CH1：輸入參數，代表對第 2 組 SPI 進行控制或設定
- SPI_API_CH2：輸入參數，代表對第 3 組 SPI 進行控制或設定
- SPI_API_CH3：輸入參數，代表對第 4 組 SPI 進行控制或設定

若只用到 1 組 SPI CH 並不會造成記憶體的浪費，原因是支援 SPI CH 的數量可設定，透過預處理器(Preprocessor)移除未使用的 SPI CH 程式，減少記憶體的浪費。

API 架構如下圖所示。

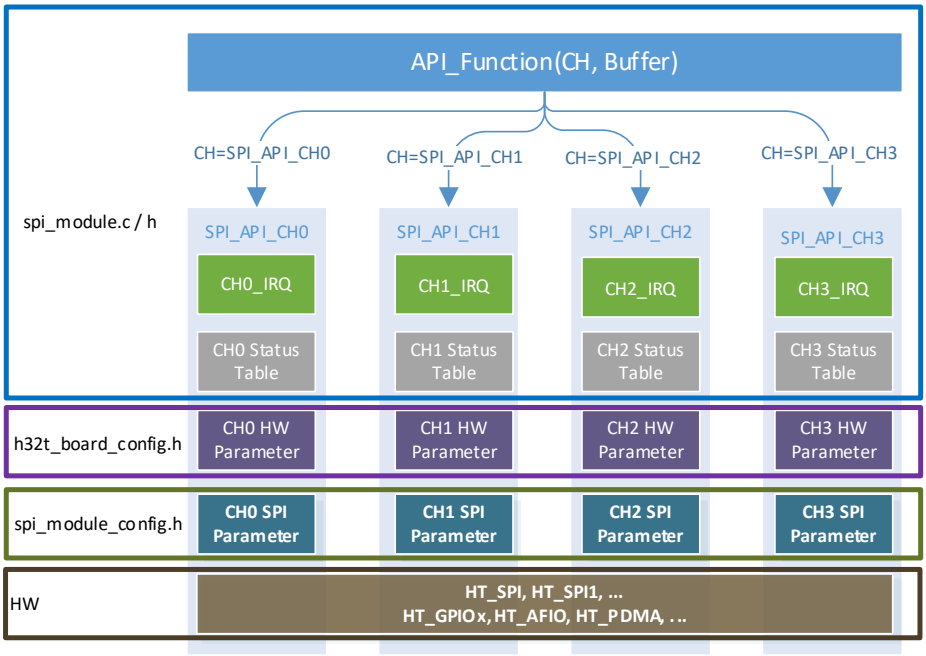

圖 10. API 架構圖

各 API 都是由 4 組 SPI CH 的相關設定或控制所組成，使用者透過參數“CH”指定 SPI CH 進行控制，若是與資料傳輸相關的 API，使用者只需多提供 Buffer 結構式參數，API 會根據 Buffer 結構式內容進行資料存取。Buffer 結構式請參考“API 說明”章節內的 SPIAPI_BufferTypeDef 說明。

spi_module.c/h 內僅有各組 SPI 的中斷(CHx_IRQ)與狀態表格(CHx Status)，而 SPI 通訊所需的一切設定都是參考 ht32_board_config.h 與 spi_module_config.h。

- ht32_board_config.h：硬體相關參數，包括：I/O 設定、實體 SPI Port 設定。SPI 範例只有使用到一組 SPI CH，因此下表以 SPI CH0 的設定進行說明，符號整理如下表所示。

符號	說明
HTCFG_CH0_SCK_GPIO_PORT	定義 CH0 的 SCK 的 Port 名稱，Example: A, B, C, …
HTCFG_CH0_SCK_GPIO_PIN	定義 CH0 的 SCK 的 Pin 編號，Example: 0~15
HTCFG_CH0_SEL_GPIO_PORT	定義 CH0 的 SEL 的 Port 名稱，Example: A, B, C, …
HTCFG_CH0_SEL_GPIO_PIN	定義 CH0 的 SEL 的 Pin 編號，Example: 0~15
HTCFG_CH0_MOSI_GPIO_PORT	定義 CH0 的 MOSI 的 Port 名稱，Example: A, B, C, …
HTCFG_CH0_MOSI_GPIO_PIN	定義 CH0 的 MOSI 的 Pin 編號，Example: 0~15
HTCFG_CH0_MISO_GPIO_PORT	定義 CH0 的 MISO 的 Port 名稱，Example: A, B, C, …
HTCFG_CH0_MISO_GPIO_PIN	定義 CH0 的 MISO 的 Pin 編號，Example: 0~15
HTCFG_SPI_CH0	定義 CH0 的硬體 SPI Port 名稱，Example：SPI0, SPI1, SPI2, …

表 2. ht32_board_config.h 內定義符號表

要修改 SPI CH 的 I/O 設定請參考對應型號的 Datasheet。ht32_board_config.h 目前設定僅支援 1 組 SPI CH，因此只有 SPI CH0 的 I/O 定義，如果要增加 SPI CH1~3 需新增各自的 I/O 定義，具體做法可參考“設定變更及常見問題”章節。

- spi_module_config.h : SPI CH 參數的相關設定，包含 CLK 傳輸速率、SPI 資料格式(CPOL、CPHA)、MSB First / LSB First、SEL 有效準位、資料搬運方式(中斷/PDMA)。定義的符號整理如下。

符號	說明
SPICFG_DEBUG_MODE	1： 致能 Debug 模式 0： 除能 Debug 模式
SPICFG_SUPPORT_CH	設定支援的 SPI 數量。Example：1~4
SPICFG_MISOD_IDLE_VALUE	1： 當閒置時回應 0xFF，Slave 模式下才有用 0： 當閒置時回應 0x00，Slave 模式下才有用
SPICFG_PDMA_MODE	1： 致能 PDMA 模式 0： 除能 PDMA 模式
SPICFG_CHx_PDMA_ENABLE	1： 致能 CHx 的 PDMA 模式，x=0~3 0： 除能 CHx 的 PDMA 模式，x=0~3
SPICFG_CHx_PDMA	1： 致能 PDMA 連接至 CHx 之 SPI1，x=0~3 0： 除能 PDMA 連接至 CHx 之 SPI0，x=0~3
SPICFG_CHx_MODE	1 : Master 模式，x=0~3 0 : Slave 模式，x=0~3
SPICFG_CHx_SEL_POLARITY	1 : SEL 高準位有效，x=0~3 0 : SEL 低準位有效，x=0~3
SPICFG_CHx_FIRST_BIT	1 : LSB First，x=0~3 0 : MSB First，x=0~3
SPICFG_CHx_CPOL	1 : CPOL=1，x=0~3 0 : CPOL=0，x=0~3
SPICFG_CHx_CPHA	1 : CPHA=1，x=0~3 0 : CPHA=0，x=0~3
_SPICFG_CHx_CLOCK_DIV	SCK 的頻率(SPI 的傳輸速度)，僅 Master 模式下有效

表 3. spi_module_config.h 內設定說明

若使用 Keil IDE 可透過“Configuration Wizard”進行設定。點選視窗左下方“Configuration Wizard”後看到下圖設定視窗。

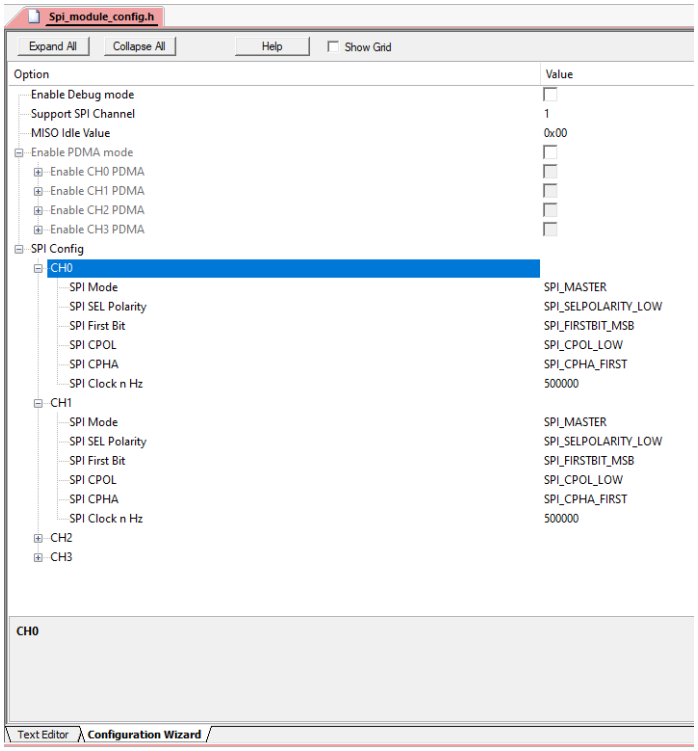

圖 11. Configuration Wizard

SPI 的參數中與通訊格式相關的設定，包括：SPICFG_CHx_SEL_POLARITY、SPICFG_CHx_FIRST_BIT、SPICFG_CHx_CPOL、SPICFG_CHx_CPHA，Master 與 Slave 兩者設定必須完全相同才可正常傳輸，因此使用前必須先相互確定通訊格式才能做出正確的設定。

API 架構特色可整理如下面 5 項：

1. 最多可支援4個SPI CH，代入參數為SPI_API_CH0、SPI_API_CH1、SPI_API_CH2、SPI_API_CH3。
2. SPI CH 數量可設定，並且未使用到的 SPI CH 不會造成記憶體浪費。
3. SPI 設定與 API 完全分離，可增加設定值的管理方便性，減少設定錯誤或遺漏設定的風險。
4. API 內未宣告任何全域的 TX / RX Data Buffer，Data Buffer 由應用層提供(僅宣告 Buffer 格式)。此做法讓記憶體可隨應用動態調整大小，避免記憶體的浪費。
5. 簡化 API 的輸入參數(最多 2 個參數)，降低使用的複雜度。

API 說明

API 總表如下。

API 名稱	說明
SPIAPI_Init()	SPI 初始化
SPIAPI_GetStatus()	取得 SPI 的傳輸狀態
SPIAPI_WriteRead()	開始進行資料傳輸
SPIAPI_GetReadBufferIndex()	回傳目前接收的資料量
SPIAPI_GetWriteBufferIndex()	回傳目前傳輸的資料量
SPIAPI_ClearStatus()	清除錯誤狀態

1. Application Code 範例程式的資料類型說明。

   - SPIAPI_BufferTypeDef

     資料形態 SPIAPI_BufferTypeDef 在資料傳輸時為使用者提供 API 的 Buffer 表格，API 會根據此參數取得資料的存取位置，以及資料長度等資訊。各參數的說明整理如下：

變量名稱	類型	說明
uTxRxLength	u32	代表要輸出/接收的資料量，以 Byte 為單位
puTxBuffer	u8*	傳輸的 Data Buffer 的指標
puRxBuffer	u8*	接收的 Data Buffer 的指標
uIsTxBufferFix	bool	TRUE：Dummy Data 模式，固定只傳輸第一個 Byte 的資料 FALSE：一般模式，傳輸 puTxBuffer 的內容
uIsRxBufferFix	bool	TRUE：Dummy Data 模式，固定只接收第一個 Byte 的資料 FALSE：一般模式，接收的資料存放到 puRxBuffer

   - SPIAPI_Status_Enum

     資料形態 SPIAPI_Status_Enum 為 SPI 的傳輸狀態，各常數整理如下：

常數名稱	說明
SPI_API_FINISH	本次傳輸已完成
SPI_API_BUSBUSY	本次傳輸未完成
SPI_API_SLAVE_ABORT	僅在 Slave 模式下會發生，代表 Slave 在一個 Byte 傳輸未完成 Master 將 SEL 改爲無效
SPI_API_SLAVE_RX_UNEXPECTED_DATA	僅在 Slave 模式下會發生，代表 Slave 在未準備讀寫時，Master 發出資料

2. Application Code 範例程式的函數說明。

函數	void SPIAPI_Init(u32 CH)	
功能	SPI 初始化	
輸入	CH	SPI_API_CH0，對第 0 組 SPI 進行初始化 SPI_API_CH1，對第 1 組 SPI 進行初始化 SPI_API_CH2，對第 2 組 SPI 進行初始化 SPI_API_CH3，對第 3 組 SPI 進行初始化
用法	SPIAPI_Init (SPI_API_CH0); /*進行第 0 組的 SPI 初始化*/	

函數	SPIAPI_Status_Enum SPIAPI_GetStatus(u32 CH)	
功能	取得 SPI 的傳輸狀態	
輸入	CH	SPI_API_CH0，對第 0 組 SPI 進行初始化 SPI_API_CH1，對第 1 組 SPI 進行初始化 SPI_API_CH2，對第 2 組 SPI 進行初始化 SPI_API_CH3，對第 3 組 SPI 進行初始化
輸出	SPI_API_FINISH	本次傳輸已完成
	SPI_API_BUSBUSY	本次傳輸未完成
	SPI_API_SLAVE_ABORT	僅在 Slave 模式下會發生，代表 Slave 在一個 Byte 傳輸未完成 Master 將 SEL 改爲無效
	SPI_API_SLAVE_RX_UNEXPECTED_DATA	僅在 Slave 模式下會發生，代表 Slave 在未準備讀寫時，Master 發出資料
用法	傳輸資料後可呼叫此 API 確定資料傳輸已完成。舉例如下： while(SPIAPI_GetStatus(SPI_API_CH0) == SPI_API_BUSBUSY);	

異常狀態處理辦法請參考“設定變更及常見問題”章節。

函數	bool SPIAPI_WriteRead(u32 CH, SPIAPI_BufferTypeDef *pSPIAPIBuffer)	
功能	開始進行資料傳輸	
輸入	CH	SPI_API_CH0，對第 0 組 SPI 進行初始化 SPI_API_CH1，對第 1 組 SPI 進行初始化 SPI_API_CH2，對第 2 組 SPI 進行初始化 SPI_API_CH3，對第 3 組 SPI 進行初始化
	pSPIAPIBuffer	TX/RX Buffer相關資訊，API會將資料讀/寫到此參數
輸出	TRUE	代表完成Buffer初始化，開始傳輸資料
	FALSE	Slave模式下發現錯誤狀態未清除 (Master：不會回應FALSE)
用法	bool status = SPIAPI_WriteRead(SPI_API_CH0,&SPIAPIBuffer);	

異常輸出處理辦法：

- 發生 FALSE 代表，錯誤狀態未清除，可能錯誤爲 SPI_API_SLAVE_ABORT 或 SPI_API_SLAVE_RX_UNEXPECTED_DATA。可呼叫 SPIAPI_GetStatus(CH)進行確認。

函數	u32 SPIAPI_GetReadBufferIndex(u32 CH)	
功能	回傳目前接收的資料量	
輸入	CH	SPI_API_CH0，對第 0 組 SPI 進行初始化 SPI_API_CH1，對第 1 組 SPI 進行初始化 SPI_API_CH2，對第 2 組 SPI 進行初始化 SPI_API_CH3，對第 3 組 SPI 進行初始化
輸出	Index	已接收的資料數量
用法	u32 RxIndex = SPIAPI_GetReadBufferIndex(SPI_API_CH0);	

函數	u32 SPIAPI_GetWriteBufferIndex(u32 CH)	
功能	回傳目前傳輸的資料量	
輸入	CH	SPI_API_CH0，對第 0 組 SPI 進行初始化 SPI_API_CH1，對第 1 組 SPI 進行初始化 SPI_API_CH2，對第 2 組 SPI 進行初始化 SPI_API_CH3，對第 3 組 SPI 進行初始化
輸出	Index	已傳輸的資料數量
用法	u32 TxIndex = SPIAPI_ GetWriteBufferIndex (SPI_API_CH0);	

函數	void SPIAPI_ClearStatus(u32 CH);	
功能	清除錯誤狀態	
輸入	CH	SPI_API_CH0，對第 0 組 SPI 進行初始化 SPI_API_CH1，對第 1 組 SPI 進行初始化 SPI_API_CH2，對第 2 組 SPI 進行初始化 SPI_API_CH3，對第 3 組 SPI 進行初始化
用法	使用 SPIAPI_ClearStatus(CH)可移除錯誤狀態 SPI_API_SLAVE_ABORT 與 SPI_API_SLAVE_RX_UNEXPECTED_DATA。舉例如下： SPIAPI_ClearStatus (SPI_API_CH0);	

API 使用實例

以下透過流程圖說明 Master/Slave 的程式。

- 加入 Header 檔

  Master/Slave 使用同一組 API，使用前請加入 Header 檔，程式碼舉例如下。

```
#include "middleware/spi_module.h"
```

- Master 讀寫流程

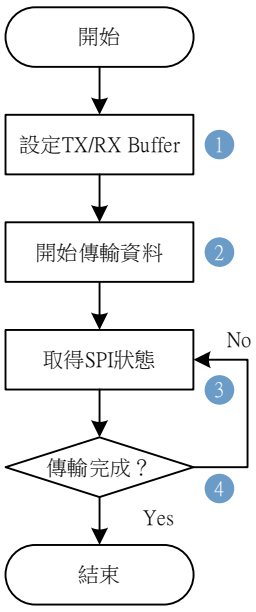

圖 12. Master 讀寫流程圖

(1) 設定 TX/RX Buffer：請透過 SPIAPI_BufferTypeDef 宣告 Buffer 表格。舉例如下，假設目標傳輸 19 筆序號資料。

```
SPIAPI_BufferTypeDef SPIAPIBuffer;
 u8 guReceiveBuffer[19];
 u8 guTransmitBuffer[19] = {0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,
                            0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,
                            0x10,0x11,0x12};
  SPIAPIBuffer.puRxBuffer = guReceiveBuffer;
  SPIAPIBuffer.puTxBuffer = guTransmitBuffer;
   SPIAPIBuffer.uTxRxLength = 19;
   SPIAPIBuffer.uIsTxBufferFix = FALSE;
   SPIAPIBuffer.uIsRxBufferFix = FALSE;
```

(2) 開始傳輸資料：舉例如下，使用 SPI CH0 進行資料傳輸。

```
SPIAPI_WriteRead(SPI_API_CH0,&SPIAPIBuffer);
```

(3) 取得 SPI 狀態：透過 SPIAPI_GetStatus()取得傳輸狀態。

(4) 判斷是否傳輸完成：結合第 3 點舉例如下，若是傳輸未完成則繼續取得狀態，直到傳輸完成。

```
while(SPIAPI_GetStatus(SPI_API_CH0) == SPI_API_BUSBUSY);
```

- Slave 讀寫流程

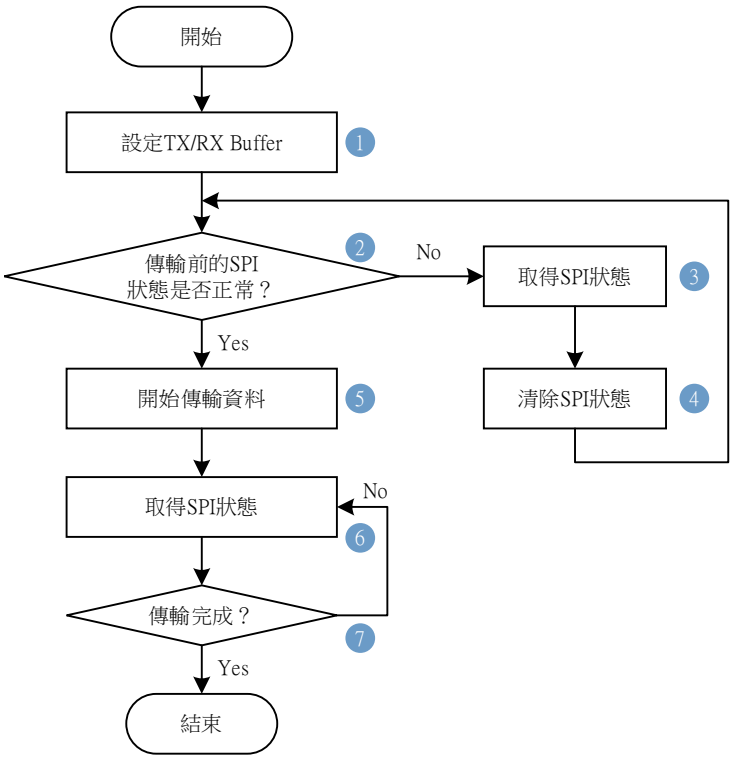

圖 13. Slave 讀寫流程圖

(1) 設定 TX/RX Buffer：請透過 SPIAPI_BufferTypeDef 宣告 TX/RX Buffer。舉例如下，假設目標傳輸 19 筆序號資料。

```
SPIAPI_BufferTypeDef SPIAPIBuffer;
 u8 guReceiveBuffer[19];
 u8 guTransmitBuffer[19] = {0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,
                            0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,
                            0x10,0x11,0x12};
  SPIAPIBuffer.puRxBuffer = guReceiveBuffer;
  SPIAPIBuffer.puTxBuffer = guTransmitBuffer;
  SPIAPIBuffer.uTxRxLength = 19;
  SPIAPIBuffer.uIsTxBufferFix = FALSE;
  SPIAPIBuffer.uIsRxBufferFix = FALSE;
```

(2) 傳輸前判斷 SPI 的狀態：此判斷程式會在 SPIAPI_WriteRead()內完成，並透過回傳值告知應用層。

(3) 取得 SPI 狀態：透過 SPIAPI_GetStatus()取得無法傳輸資料的原因並進行錯誤排除，例如透過 printf 將錯誤顯示。

(4) 清除 SPI 狀態：透過 SPIAPI_ClearStatus()清除錯誤的狀態。

(5) 開始傳輸資料：透過 SPIAPI_WriteRead()開始傳輸資料。結合第 2 點、第 3 點、第 4 點的程式舉例如下。

```
    while(1)
    {
        if( SPIAPI_WriteRead(SPI_API_CH0,&SPIAPIBuffer) == TRUE)
        {
            break;
        }
        printf("ERROR: %X\r\n",SPIAPI_GetStatus(SPI_API_CH0));
        SPIAPI_ClearStatus(SPI_API_CH0);
    }
```

(6) 取得 SPI 狀態：透過 SPIAPI_GetStatus()取得傳輸狀態。

(7) 判斷是否傳輸完成：結合第 6 點舉例如下，若是傳輸未完成則繼續取得狀態，直到傳輸完成。

```
while(SPIAPI_GetStatus(SPI_API_CH0) == SPI_API_BUSBUSY);
```

資源佔用

Module SPI 的資源占用如下表，以 HT32F52352 開 1 組 SPI CH 做為舉例。

Master	HT32F52352
ROM Size	752 Bytes
RAM Size	64 Bytes

表 4. Module SPI 中斷模式資源佔用

Master	HT32F52352
ROM Size	1,508 Bytes
RAM Size	120 Bytes

表 5. Module SPI PDMA 模式資源佔用

- 編譯環境 MDK-Arm V5.36，ARMCC V5.06 update 7 (build 960)
- 優化選項 Level 2 (-O2)

使用指南

本章節將介紹如何測試 Module_SPI 範例。

環境準備

執行此範例需準備兩塊 SK (Starter Kit)，SK 由 e-Link32 Lite 與 Target Board 組成。

本文以 HT32F52352 SK 為例，請完成以下操作：

(1) 將 USB 數據線連接 e-Link32 Lite 上的 USB 介面，位置標注在下圖(a)。

(2) 範例需用到 e-Link32 Lite 的 VCP(Virtual COM Port)功能，因此確保 UART Jumper-J2[*1] 的 PA5[*2] 與 DAP_TX 透過 Jumper 短路，位置標注在下圖(b)。

註：1. J2 位於 SK 上有兩個跳線設定，(1) PAx 與 DAP_TX 短路，或(2) PAx 與 RS232_TX 短路，其設定意義請參考 SK 的使用手冊。

2. 不同的 SK 各有不同，這裡以 PAx 來做統稱。位置標注在下圖(b)。

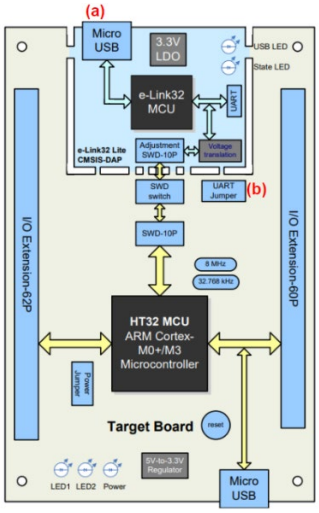

圖 14. HT32 Starter Kit 方塊圖

一塊 SK 為 Master，另外一塊為 Slave。兩塊板子的 SPI 排線對接，並且都透過 USB 接上 PC，接線關係如下圖所示。

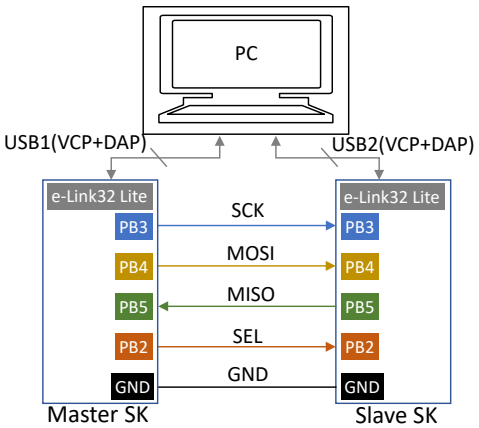

圖 15. SK 與 PC 接線圖

1. USB1 (VCP+DAP) : Master SK 的 e-Link32 Lite 請透過 USB 數據線連接 PC。
2. USB2 (VCP+DAP) : Slave SK 的 e-Link32 Lite 請透過 USB 數據線連接 PC。
3. SPI： 請將 Master SK 與 Slave SK 對接，SPI 的脚位定義請參考“ht32_board_config.h”，以 HT32F52352 SK 舉例如下。
   - 請使用杜邦線將 Master 與 Slave 的 PB2(SEL)、PB3(SCK)、PB4(MOSI)、PB5(MISO)各自連接。
   - 請使用杜邦線將 GND 相連。
4. e-Link32 Lite 與 SK 是透過 UART 連接，Master/Slave 使用 C function 的 RETARGET_Configuration 來完成相關設定，而它的 TX 與 RX 的脚位定義請參考檔案“\utilities\HT32_Board\ht32fxxxxx_sk.h”。

編譯與測試

硬體環境：SK 選用 ESK32-30501(HT32F52352)作舉例。

開發 IDE：以 Keil 作舉例。

Master 範例：路徑“\application\Module_SPI\Master”

此範例透過 COM Port 產生測試選單，使用者可透過終端機軟體與範例程式進行互動，請依照下方操作步驟完成測試環境。

Step 1. 上電測試

請根據“環境準備”章節將硬體環境設定完成。上電後 ESK32-30501 的 D9 Power LED 被點亮(板子左下方)，等 USB 完成列舉後 e-Link Lite 的 D1 USB LED 被點亮(板子右上)。若長時間 D1 未被點亮，請確認 USB 線是否可以通訊或重新插拔 USB。

Step 2. 產生 Master 與 Slave 範例專案打開\application\Module_SPI\Master 與\application\Module_SPI\Slave，各自執行_CreateProject.bat 來產生專案，如下圖所示。

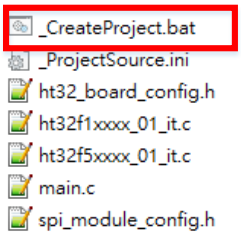

圖 16. 執行_CreateProject.bat 產生專案

Step 3. 開啟 Master 與 Slave 範例

開啟 Keil 專案“\Module_SPI\Slave\MDK_ARMv5\Project_52352.uvprojx”。

開啟 Keil 專案“\Module_SPI\Master\MDK_ARMv5\Project_52352.uvprojx”。

Step 4. 指定 Target Board：請指定 Master 與 Slave 的 Target Board，若 Keil 專案沒指定 Target Board，兩個專案都會燒錄同一塊 SK。指定 Target Board 可透過 Options for Target → Debug → Use: CMSIS-DAP Debugger → Settings → CMSIS_DAP+JTAG/SW Adapter 的下拉式選單指定 Target Board 並點選“OK”，若 Target Board 被選中會閃爍 D2 LED 一次。

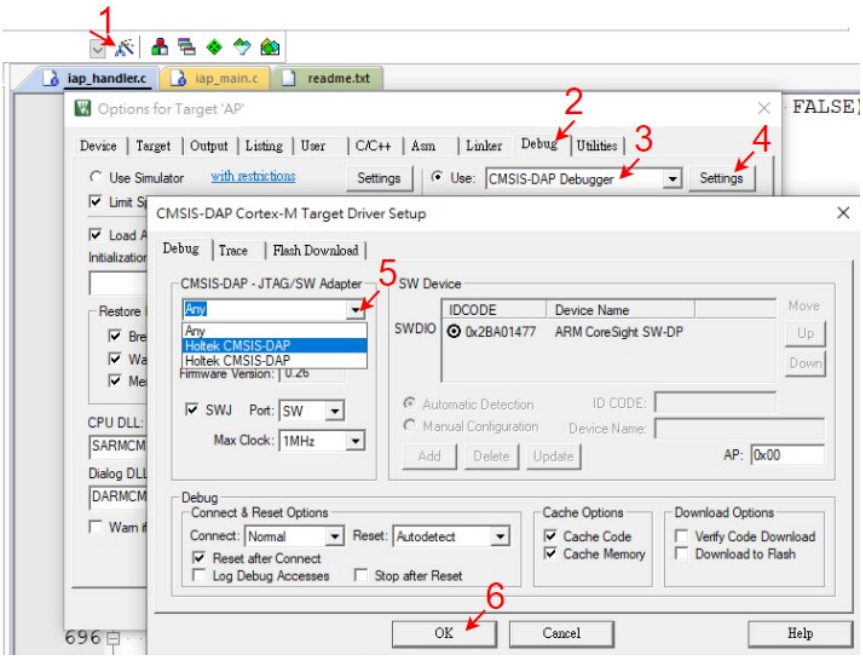

圖 17. Set the target board

Step 5. 編譯和燒錄：Master 專案與 Slave 專案各別點選“Batch Build”，完成後點選“Download”或快捷鍵“F8”。完成後 Build Output 會顯示“Batch Build”與“Download”的結果。

Step 6. 開啟工具：打開 2 個 Tera Term 並各別開啟 Master SK 與 Slave SK 對應的 COM Port (115200bps，8N1)，並特別留意 COM 編號是否正確，需確認此 COM Port 各自有 Master SK 與 Slave SK 產生。

Step 7. 重置系統：按壓 Master SK 與 Slave SK 各自的重置按鈕 B1 Reset。在 Master 的 COM Port 透過 Tera Term 顯示測試選單，如下圖；Slave 沒有選單不會出現任何提示。

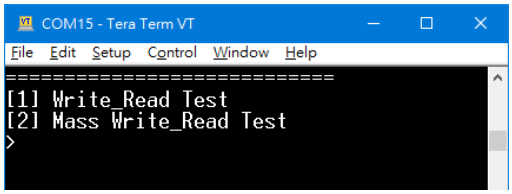

圖 18. Master Test Menu

Step 8. 開始測試：選“[1] Write_Read Test”，Master 與 Slave 會各自傳送一組 19 Bytes 的序號資料(0x00~0x12)，並顯示在各自的 Tera Term 上。

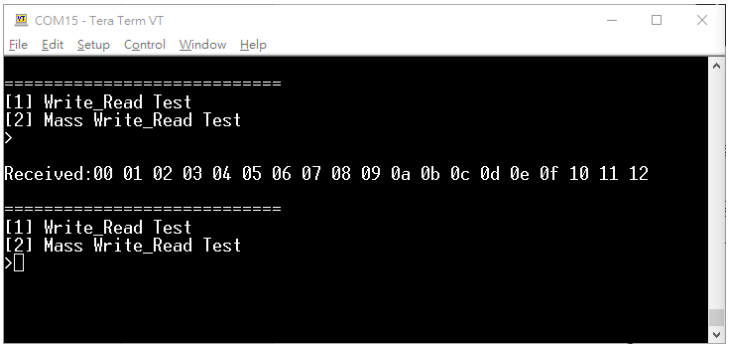

圖 19. Master 傳輸結果

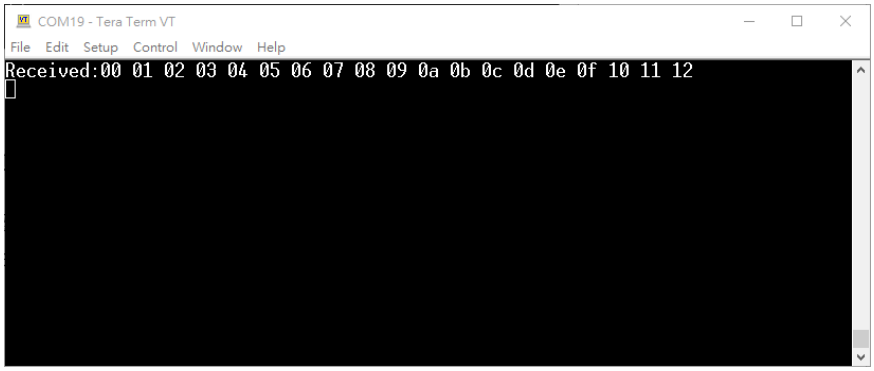

圖 20. Slave 傳輸結果

選“[2] Mass Write_Read Test”，Master 與 Slave 會各自傳送 10000 組 19 Bytes 的序號資料(0x00~0x12)，並顯示在各自的 Tera Term 上。

導入 SPI API

1. 加入\utilities\middleware\spi_module.c 到目標專案中。
2. 新增並撰寫 ht32_board_config.h，撰寫規定請直接參考範例並搭配“API 架構”中 ht32_board_config.h 的介紹。
3. 複製\utilities\middleware\spi_module_config_templet.h 到目標專案資料夾並改檔名為 spi_module_config.h。修改 spi_module_config.h 的內容，修改方法請直接參考範例程式，並搭配“API 架構”中 spi_module_config.h 的介紹。
4. 新增#include “spi_module.h”到目標.c/.cpp 檔中，檔案路徑為\utilities\middleware\ spi_module.h。
5. 在目標.c/.cpp 檔內撰寫 SPI Master/Slave 收送資料程式，請直接參考範例或參考“API 使用實例”內有完整的 API 的使用說明。

設定變更及常見問題

解決編譯錯誤

編譯時出現以下方錯誤訊息。代表 HT32 Firmware Library 版本太舊與 Module_SPI 不相容。

> error: #35: #error directive: !!! The version of HT32 Firmware Library is older than the module required. Please update HT32 Firmware Library.

請參考"資源下載與準備"的章節下載最新的 HT32 Firmware Library。

解決 SPI 收資料錯誤

如果接收資料時發現部分資料錯誤，請根據下方步驟排除問題。

1. 可先降低 CLK 的速率(例如 500kHz)，如傳輸正常請改善線材或縮短線材。
2. 如果資料還是錯誤請確認 Master 與 Slave 的通訊格式是否一致。

解決錯誤狀態

API SPIAPI_GetStatus()回應錯誤狀態排除方法如下：

1. API_BUSBUSY，資料傳輸未完成，可定時輪詢傳輸狀態直到傳輸完成。
2. SPI_API_SLAVE_ABORT，僅發生在 Slave 模式，此狀態代表 Master 發生意外或是通訊格式不一致。如果需清除此狀態請呼叫 SPIAPI_ClearStatus()。
3. SPI_API_SLAVE_RX_UNEXPECTED_DATA，僅發生在 Slave，此狀態代表 Master 與 Slave 的通訊協定不一致(Master 多送資料)。需檢查 Master 為何有不預期的資料或是兩者間有 Timing issues(Master 需等 Slave ready)，如果需清除此狀態請呼叫 SPIAPI_ClearStatus(CH)。

如何新增 SPI CH

範例目前只有 SPI CH0，如果要增加一組 SPI CH，設定方法請參考下方使用 HT32F52352 舉例。

- 修改 ht32_board_config.h

  HT32 MCU 的 I/O 本非所有都支援 SPI，因此那些 I/O 復用 SPI 功能請參考 Datasheet 的 Pin Assignment，我們使用可支援 SPI1 的 PA0~PA3。

Package			Alternate Function Mapping										
			AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10
64 LQFP	48 LQFP	33 QFN	System Default	GPIO	ADC	CMP	MCTM /GPTM	SPI	USART /UART	I²C	SCI	EBI	I2S
1	1	1	PA0		ADC_ IN0		GT1_ CH0	SPI1_ SCK	USR0_ RTS	I2C1_ SCL	SCI0_ CLK		I2S_WS
2	2	2	PA1		ADC_ IN1		GT1_ CH1	SPI1_ MOSI	USR0_ CTS	I2C1_ SDA	SCI0_ DIO		I2S_ BCLK
3	3	3	PA2		ADC_ IN2		GT1_ CH2	SPI1_ MISO	USR0_ TX				I2S_ SDO
4	4	4	PA3		ADC_ IN3		GT1_ CH3	SPI1_ SEL	USR0_ RX				I2S_SDI

圖 21. HT32F52352 Alternate Function Mapping

我們增加 PA0~PA3 在 ht32_board_config.h，如下圖所示。

```
#if defined(USE_HT32F52352_SK)
  #define HTCFG_CH0_SCK_GPIO_PORT              B
  #define HTCFG_CH0_SCK_GPIO_PIN               3
  #define HTCFG_CH0_SEL_GPIO_PORT              B
  #define HTCFG_CH0_SEL_GPIO_PIN               2
  #define HTCFG_CH0_MOSI_GPIO_PORT             B
  #define HTCFG_CH0_MOSI_GPIO_PIN              4
  #define HTCFG_CH0_MISO_GPIO_PORT             B
  #define HTCFG_CH0_MISO_GPIO_PIN              5
  #define HTCFG_SPI_CH0                        SPI0

  #define HTCFG_CH1_SCK_GPIO_PORT              A
  #define HTCFG_CH1_SCK_GPIO_PIN               0
  #define HTCFG_CH1_SEL_GPIO_PORT              A
  #define HTCFG_CH1_SEL_GPIO_PIN               3
  #define HTCFG_CH1_MOSI_GPIO_PORT             A
  #define HTCFG_CH1_MOSI_GPIO_PIN              1
  #define HTCFG_CH1_MISO_GPIO_PORT             A
  #define HTCFG_CH1_MISO_GPIO_PIN              2
  #define HTCFG_SPI_CH1                        SPI1
#endif
```

圖 22. CH1 硬體設定

spi_module_config.h：此設定檔有實現 Keil Configuration Wizard 的功能。設定方法請參考下圖。

a. 切換至 Configuration Wizard 介面。

b. 設定支援 2 組 SPI CH。

c. 設定 SPI CH1 的 SPI 通訊格式。

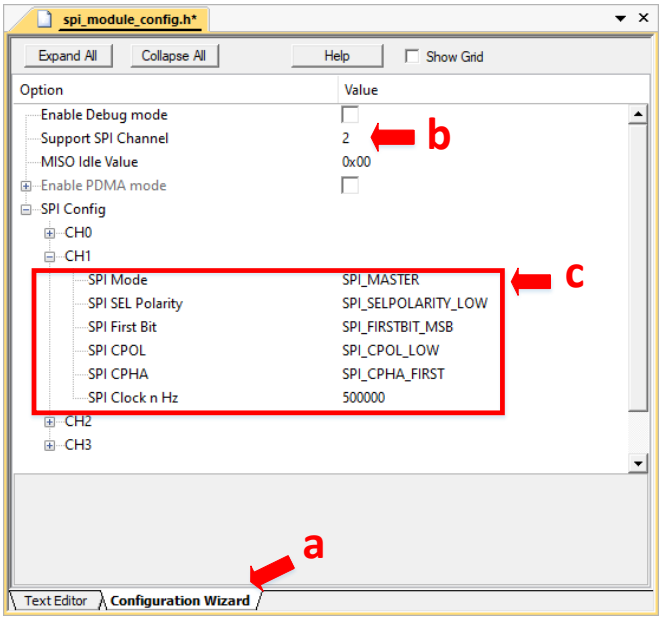

圖 23. CH1 SPI 設定

程式流程請參考“API 使用實例”，並將 SPI_API_CH0 改為 SPI_API_CH1 即可。

結論

本文針對 SPI Module 進行了詳細的介紹，使用者透過本文與範例程式可快速熟悉 API 架構與使用方法。

參考資料

如需進一步瞭解，敬請瀏覽 Holtek 官方網站 www.holtek.com。

版本及修訂說明

日期	作者	發行	修訂說明
2022.06.01	王宏綸	V1.00	第一版

免責聲明

本網頁所載的所有資料、商標、圖片、連結及其他資料等 (以下簡稱「資料」)，只供參考之用，盛群半導體股份有限公司及其關聯企業 (以下簡稱「本公司」) 將會隨時更改資料，並由本公司決定而不作另行通知。雖然本公司已盡力確保本網頁的資料準確性，但本公司並不保證該等資料均為準確無誤。本公司不會對任何錯誤或遺漏承擔責任。

本公司不會對任何人士使用本網頁而引致任何損害 (包括但不限於電腦病毒、系統故障、資料損失) 承擔任何賠償。本網頁可能會連結至其他機構所提供的網頁，但這些網頁並不是由本公司所控制。本公司不對這些網頁所顯示的內容作出任何保證或承擔任何責任。

責任限制

在任何情況下，本公司並不須就任何人由於直接或間接進入或使用本網站，並就此內容上或任何產品、資訊或服務，而招致的任何損失或損害負任何責任。

管轄法律

本免責聲明受中華民國法律約束，並接受中華民國法院的管轄。

免責聲明更新

本公司保留隨時更新本免責聲明的權利，任何更改於本網站發布時，立即生效。