

MULTI MASTER SENSOR HUB

中文版技術文件

台灣科技大學

電子所 碩二 劉晏廷

內容

0. 編輯歷史.....	6
1. 關於這份文件.....	7
1.1.    文件目的.....	7
1.2.    如何使用.....	7
1.3.    文件摘要.....	7
2. 系統總覽.....	8
2.1.    系統架構.....	8
2.2.    訊息序列圖.....	10
3. 系統設計.....	13
3.1.    開發環境介紹.....	13
3.1.1.    開發軟體.....	13
3.1.1.1.    Keil µVision 5 .....	13
3.1.1.2.    STM32CubeMX .....	14
3.1.2.    測試軟體.....	15
3.1.2.1.    AccessPort137.....	15
3.1.3.    輔助工具.....	16
3.1.3.1.    USBDevview .....	16
3.1.3.2.    Device Monitoring Studio.....	18
3.2.    開發規格.....	19
3.2.1.    硬體規格.....	19
3.2.2.    融合後資料封包規範.....	19
3.3.    文件標準.....	20
3.3.1.    微控制器初始化設置.....	24
3.3.1.1.    System Clock Configure .....	25
3.3.1.2.    MX GPIO Initialization .....	27
3.3.1.3.    MX DMA Initialization.....	28
3.3.1.4.    MX USB Device Initialization.....	30
3.3.1.5.    MX TIM2 Initialization .....	30
3.3.1.6.    MX I2C Initialization.....	33
3.3.1.7.    MX FMPI2C1 Initialization.....	35
3.3.2.    感測器參數設定.....	37
3.3.2.1.    FIFO_CTRL5.....	37
3.3.2.2.    CTRL1_XL .....	37
3.3.2.3.    CTRL2_G .....	37
3.3.2.4.    CTRL6_C .....	38
3.3.2.5.    CTRL7_G .....	38

3.3.2.6.	使用者介面.....	38
3.3.3.	感測器初始化設置.....	41
3.3.4.	動作捕捉.....	41
3.3.4.1.	Read_Value().....	42
3.3.4.2.	Timestamp() .....	43
3.3.4.3.	DataFusion() .....	43
4.	效能表現.....	44
4.1.	收發正確性.....	44
4.2.	收發速度.....	47
4.2.1.	AccessPort.....	47
4.2.2.	Device Monitoring Studio.....	48
4.2.3.	USB 輸出 .....	50
4.3.	收發能量損耗.....	51
4.3.1.	HAL_Delay()對功耗的影響 .....	51
4.3.2.	FOR 迴圈對功耗的影響.....	52
5.	GitHub 相關.....	55
5.1.	GitHub 說明-前置作業 .....	55
5.1.1.	GitHub 帳號建立.....	55
5.1.2.	GitHub 圖型化介面.....	58
5.2.	GitHub 使用說明.....	61
6.	使用者介面.....	错误!未定义书签。
6.1.	介面介紹.....	错误!未定义书签。
6.2.	.....	65
7.	結語.....	66
7.1.	本專案目的.....	66
7.2.	展望.....	错误!未定义书签。
7.3.	定義、縮詞與簡寫 .....	错误!未定义书签。
7.4.	參考文獻.....	66

## 圖目錄

圖 2-1 系統架構圖 .....	8
圖 2-2 系統組成圖 .....	8
圖 2-3 六軸感測器 .....	9
圖 2-4 資料獲取模組 .....	9
圖 2-5 資料獲取程序的訊息序列圖 .....	10
圖 2-6 資料獲取模組的架構 .....	10
圖 2-7 一般資料獲取路徑 .....	11
圖 2-8 使用 DMA 與多條硬體式 I <sup>2</sup> C 匯流排.....	12
圖 2-9 資料融合與送出 .....	12
圖 3-1 Keil μVision 5 操作介面 .....	14
圖 3-2 STM32CubeMX 操作介面 .....	15
圖 3-3 AccessPort137 操作介面 .....	16
圖 3-4 USBDevview 操作介面 .....	17
圖 3-5 Device Monitoring Studio 操作介面 .....	18
圖 3-6 Data format .....	19
圖 3-7 開發版設置 .....	21
圖 3-8 STM32F412 功能設置.....	22
圖 3-9 清除 PA8 腳位 .....	23
圖 3-10 產生程式原始碼 .....	24
圖 3-11 Flow chart of Main.c.....	錯誤!未定义书签。
圖 3-12 Main.c .....	錯誤!未定义书签。
圖 3-13 STM32CubeMX 的 Clock Configuration .....	25
圖 3-14 範例程式碼下載 .....	26
圖 3-15 System Clock Configuration .....	26
圖 3-16 GPIO Initialization .....	27
圖 3-17 STM32CubeMX 的 Pin Configuration .....	28
圖 3-18 DMA Initialization.....	29
圖 3-19 STM32CubeMX 的 DMA 配置 .....	29
圖 3-20 MX USB Device Initialization .....	30
圖 3-21 CDC_Transmit_FS().....	30
圖 3-22 MX TIM2 Initialization .....	31
圖 3-23 TIM2 Configuration.....	32
圖 3-24 MX I2C Initialization .....	33
圖 3-25 I2C1 Configuration.....	34
圖 3-26 MX FMPI2C Initialization .....	35
圖 3-27 FMPI2C Configuration.....	36
圖 3-28 FIFO_CTRL5 暫存器 .....	37
圖 3-29 CTRL1_XL 暫存器 .....	37
圖 3-30 CTRL2_G 暫存器 .....	38

圖 3-31 CTRL6_C 暫存器 .....	38
圖 3-32 CTRL7_G 暫存器 .....	38
圖 3-33 AccessPort 介面 .....	39
圖 3-34 感測器參數設定流程圖 .....	40
圖 3-35 I2C 感測器設置 .....	41
圖 3-36 FMPI2C 感測器設置 .....	41
圖 3-37 動作捕捉程式碼 .....	42
圖 3-38 Read_Value() .....	42
圖 3-39 DataConvertA() .....	43
圖 3-40 Timestamp().....	43
圖 3-41 DataFusion() .....	43
圖 4-1 AiQ Software .....	44
圖 4-2 Stored file .....	45
圖 4-3 AccessPort result.....	48
圖 4-4 Choosing Device.....	48
圖 4-5 Session Configuration.....	49
圖 4-6 Select Packet View .....	49
圖 4-7 Device Monitoring Studio result .....	49
圖 4-8 USB Cable .....	50
圖 4-9 Oscilloscope result.....	50
圖 4-10 Hardware Testing Component .....	51
圖 4-11 HAL_Delay().....	52
圖 4-12 FOR 迴圈程式碼 .....	53
圖 5-1 github 首頁 .....	55
圖 5-2 註冊帳號資訊 .....	错误!未定义书签。
圖 5-3 創建帳號-第二步驟 .....	56
圖 5-4 創建帳號-第三步驟 .....	错误!未定义书签。
圖 5-5 登入頁面 .....	错误!未定义书签。
圖 5-6 Settings 頁面 .....	错误!未定义书签。
圖 5-7 認證成功後的頁面 .....	错误!未定义书签。
圖 5-8 圖型化界面下載頁面 .....	错误!未定义书签。
圖 5-9 安裝頁面 .....	59
圖 5-10 GitHub 桌面捷徑 .....	错误!未定义书签。
圖 5-11 Log in 頁面 .....	60
圖 5-12 AiQ 專案頁面.....	61
圖 5-13 圖型化界面的頁面 .....	错误!未定义书签。

## 表目錄

表 1 編輯歷史.....	6
表 2 Data format discription .....	19
表 3-3 Timer 對應表.....	30
表 4 Test results.....	45
表 5 timestamp analysis .....	47
表 6 HAL_Delay()與功耗之關係(使用 DMA).....	52
表 7 HAL_Delay()與功耗之關係(使用非 DMA).....	52
表 8 FOR 迴圈與功耗之關係(使用 DMA).....	53
表 9 FOR 迴圈與功耗之關係(使用非 DMA).....	53

# 0.編輯歷史

編輯歷史是用於紀錄編輯順序，每次對文件進行修改後都會在這一部分留下紀錄。

Revision	Date	Author	Changes
0.0	2016/12/07	Tim Liu	Preliminary release
0.1	2016/12/09	Tim Liu	新增 關於這份文件 的內容
0.2	2016/12/12	Tim Liu	新增 系統總覽 的內容
0.3	2016/12/16	Tim Liu	新增 系統內容 的內容
0.4	2016/12/23	Tim Liu	新增 4.1 設計方法與規範 的內容
0.5	2016/12/30	Tim Liu	新增 4.1 設計方法與規範 的內容
0.6	2017/01/06	Tim Liu	1.修改 1.2 如何使用 原內容與 1.3 文件摘要類似，故修改 2.修改 2.1 系統架構 將系統架構圖修改，並把各原件介紹拉到圖下做說明，並修改 2.2 系統特性的內容 3.修改 3.1 命令表 將 3.1 的內容做細分，新增 3.1.1.1、3.1.1.2 以及 3.1.2.1、 3.1.2.2 的內容
0.7	2017/01/13	Tim Liu	1.新增 5.GitHub 相關 的內容 (5.1 GitHub 說明-前置作業) 2.根據 AiQ IK16 SW spec v1.1_20170110 AiQ Smart Clothing app spec v1.0_20170110 兩份文件來修改內容與 MSC 圖。
0.8	2017/01/20	Tim Liu	1.修改 3.2 MSC 圖及 4.1.2 完整 MSC 圖 的內容，針對新的 spec 做修正。
0.9	2017/02/16	Tim Liu	新增 2.1.2 USB type-C 新增圖輔助說明
1.0	2017/07/14	Tim Liu	修改大部分內容，依照現在的架構 作糾正，並將實作方法新增於文中
1.1	2017/08/15	Tim Liu	修改內容，並新增效能表現。

表 1 編輯歷史

# 1. 關於這份文件

## 1.1. 文件目的

設計本文件的目的是為了讓所有開發者、使用者可以快速上手，在短時間內熟悉本產品。如果您是開發者，您將可以透過這份文件了解設計邏輯，並且有效率的開發您的應用。如果您是使用者，您將可以透過這份文件了解操作順序，讓您在使用上可以花最少的時間熟悉產品並開始您的訓練。

## 1.2. 如何使用

本文件的內容不僅適合開發者，也適合一般的使用者。對於開發者而言，本文件提供整個系統的設計邏輯，程式碼對應，完整訊息流向，讓開發者可以輕易了解整個系統的運作，並快速的找到您想修改的部分進行研究。對於一般使用者而言，本文件提供操作順序，可以讓一般大眾輕而易舉的使用本系統，在使用上有任何疑問也歡迎不吝回饋給我們知道。

## 1.3. 文件摘要

本文件分為 7 部分：

「關於這份文件」將會告訴您這份文件的設計目的，理念以及如何更有效率的使用這份文件

「系統總覽」將會告訴您這個系統的架構及此系統的重要特性，若您是想要粗略的了解整個系統的運作，這個章節將會幫助您許多。

「系統設計」將會告訴您這個系統設計的流程及邏輯，若您想要徹頭徹尾的了解這個系統，這個章節再適合不過了。

「效能表現」將會告訴您這個系統的性能評估以及測試內容，讓您可以針對此效能設計您的應用。

「GitHub 相關」將會告訴您本專案的 GitHub 相關資訊，如帳號及說明等等，若您想要親自體驗 coder，非常歡迎您閱讀這一章節。

「使用者操作」將會告訴您如何操作本系統，讓您可以快速上手。

「結語」將會告訴您這個系統的展望以及相關參考文獻。

## 2. 系統總覽

### 2.1. 系統架構

本系統架構如圖 2-1 所示，此架構包含四顆或多顆慣性感測器，一個資料獲取模組以及一個動作顯示模組，所有的慣性感測器都由導電纖維彼此連接，並匯集到資料獲取模組。資料獲取模組透過 I<sup>2</sup>C 匯流排讀取慣性感測器所產生的資料，進行資料融合，並將融合後資料透過 USB 介面傳送給動作顯示模組，動作顯示模組可以是電腦或是頭戴式顯示器，捕捉的資料經由演算法計算透過動畫將人體動作即時呈現。

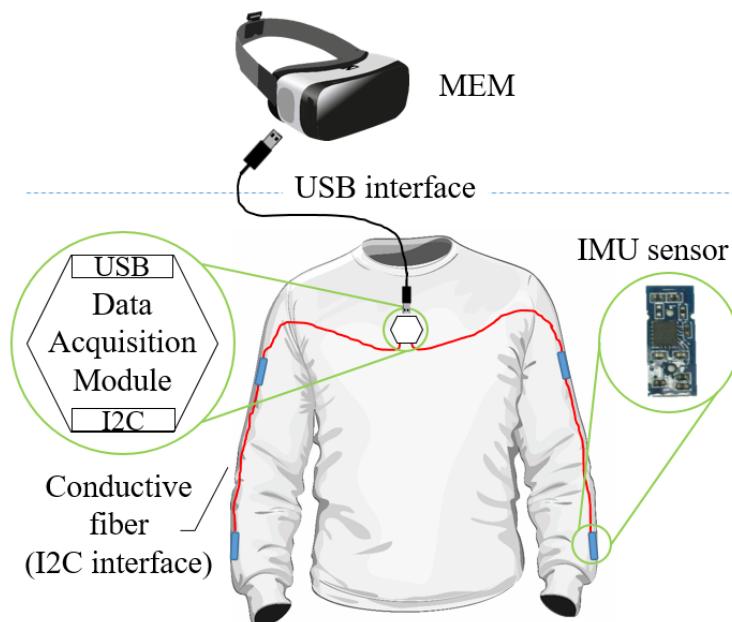


圖 2-1 System architecture

根據系統組成圖，以下將逐一說明其中的元件。

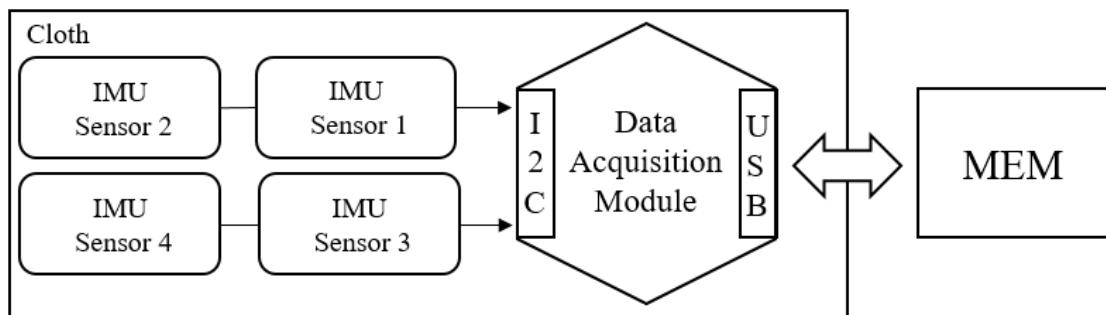


圖 2-2 System components

IMU Sensor：慣性感測器。型號為 LSM6DS33，此慣性感測器為六軸感測器，六軸分別是重力加速度計的 X 軸、Y 軸、Z 軸，及陀螺儀的 X 軸、Y 軸、Z 軸共六軸，透過修改感測器上的暫存器的值，可以對感測器做不同的設定，詳細說明請看[2]，感測器會即時偵測肢體的行為，將偵測到的資料儲存於對應的暫存器中，資料獲取模組可以讀取該暫存器來取得偵測的資料。

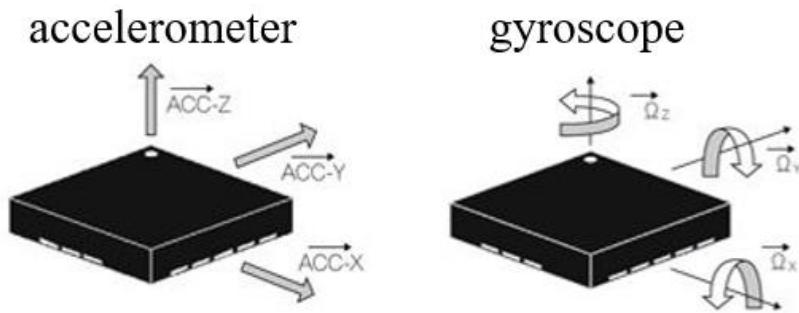


圖 2-3 Six-axis IMU sensor

Data Acquisition Module：資料獲取模組，此模組運行在微控制器上，此微控制器的型號是 STM32F412。STM32F412 可同時支援多達 4 組 I2C 匯流排，CPU 運算能力好。資料獲取模組的主要功能是定時從慣性感測器的暫存器中讀取資料，做資料融合，並將融合後資料傳送給動作顯示模組。一邊使用 I<sup>2</sup>C 與慣性感測器溝通，一邊使用 USB 與動作顯示模組溝通。

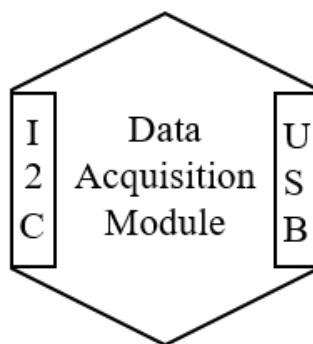


圖 2-4 Data acquisition module

MEM:動作顯示模組，通常是個人電腦或是頭戴式顯示頭盔，在此模組上會有一款軟體執行在其中，此軟體使用 android studio 以及 unity 做開發。主要功能是將資料獲取模組傳送過來的融合後資料做處理，經過演算法將資料轉換成動畫所需的參數，利用 Unity 所開發的動畫呈現出來，該軟體由 Jmex 公司提供。

## 2.2. 訊息序列圖

圖 2-5 是資料獲取程序的訊息序列圖，資料獲取模組會依序讀取慣性感測器的資料，花費時間將資料融合，並將融合後資料送給動作顯示模。在此我們定義整個資料獲取程序所需的时间為獲取間隔，本系統的主要工作就是將獲取間隔最小化，將會透過圖 2-6 來說明資料獲取程序中的時間損耗以及我們提出的實現方法。

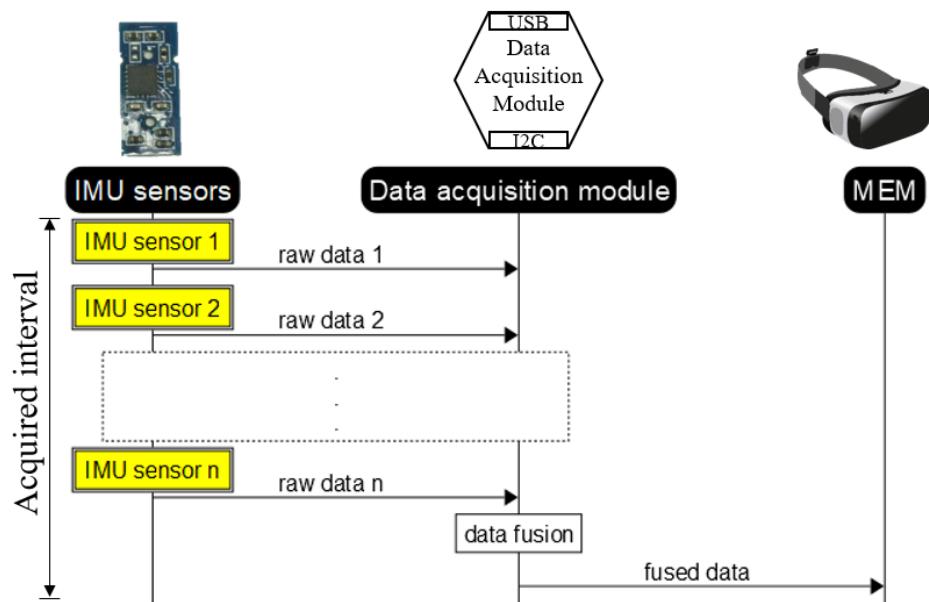
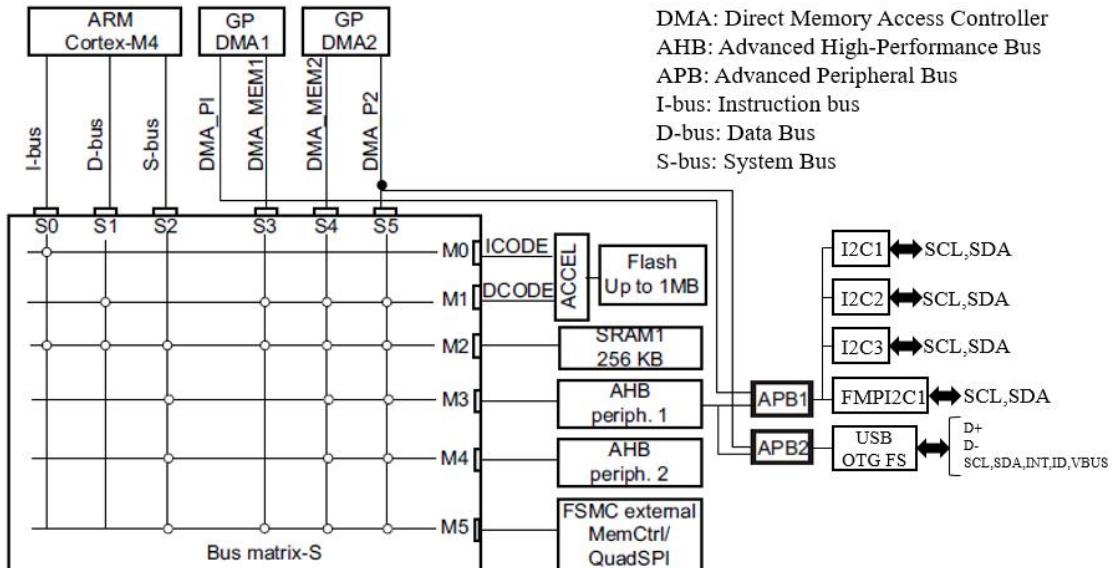


圖 2-5 The message sequence chart of data acquisition module

圖 2-6 Architecture of data acquisition module

圖 2-6 是資料獲取模組的架構，此架構的組成是由 32 位元多層高階高性能匯流排 (Advanced High-Performance Bus, AHB) 總線矩陣內接了所有的主模組：中央處理器(Central Processing Unit, CPU)、直接記憶體存取控制器(Direct Memory Access, DMA) 及從模組：快閃記憶體(Flash Memory)、隨機存取記憶體(Random-Access Memory, RAM)、高階高性能匯流排及高階周邊匯流排外設(Advanced Peripheral Bus, APB)。



當 CPU 想要讀取在 I<sup>2</sup>C1 匯流排上的裝置資料時，CPU 會透過系統匯流排 (System bus, S-bus) 以及 AHB 周邊匯流排 1 來讀取，透過資料匯流排 (Data bus, D-bus) 將資料送到快閃記憶體中儲存，如圖 2-7 所示。在同一時間只能有一筆資料在匯流排上傳輸，如果有多個裝置的資料需要讀取時，CPU 會花費很多時間與資源來重複此步驟。

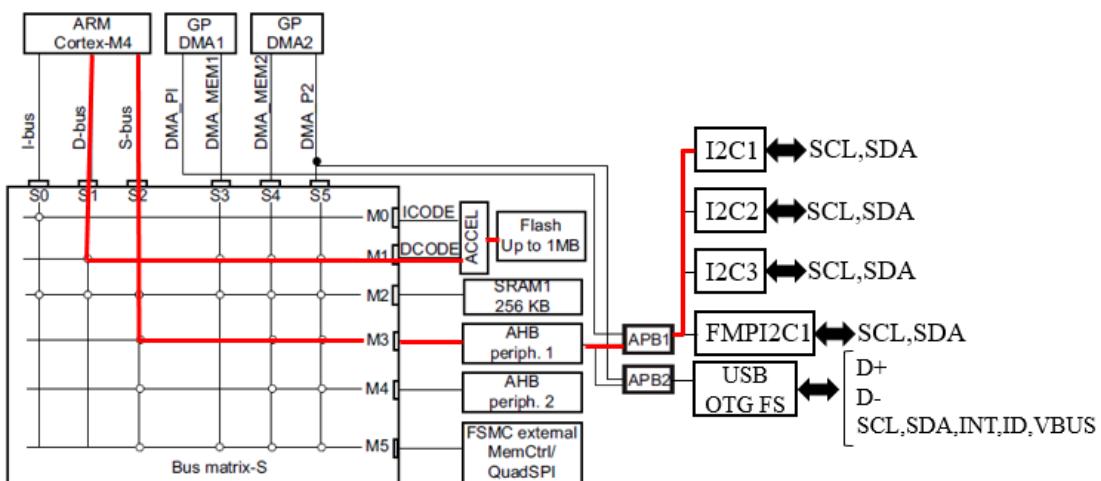


圖 2-7 General data acquisition path

為了加快獲取速度，我們使用 DMA 控制器，DMA 可以快速搬移資料無須透過 CPU，同時使用多條 I<sup>2</sup>C 匯流排來減少讀取時間，我們將不同的 I<sup>2</sup>C 匯流排配置到不同的 DMA 流，如此一來 DMA 便可以獨立地交換資料。

為了降低 CPU 的負荷，我們使用硬體式 I<sup>2</sup>C 匯流排取代軟體式 I<sup>2</sup>C 匯流排。軟體式 I<sup>2</sup>C 匯流排需要 CPU 重複執行多行程式碼來建立連線，但硬體式 I<sup>2</sup>C 匯流排只需要幾行程式碼即可設定好連線，此後匯流排就會自動運作，因此 CPU 可以獲得更多的資源來執行資料融合。

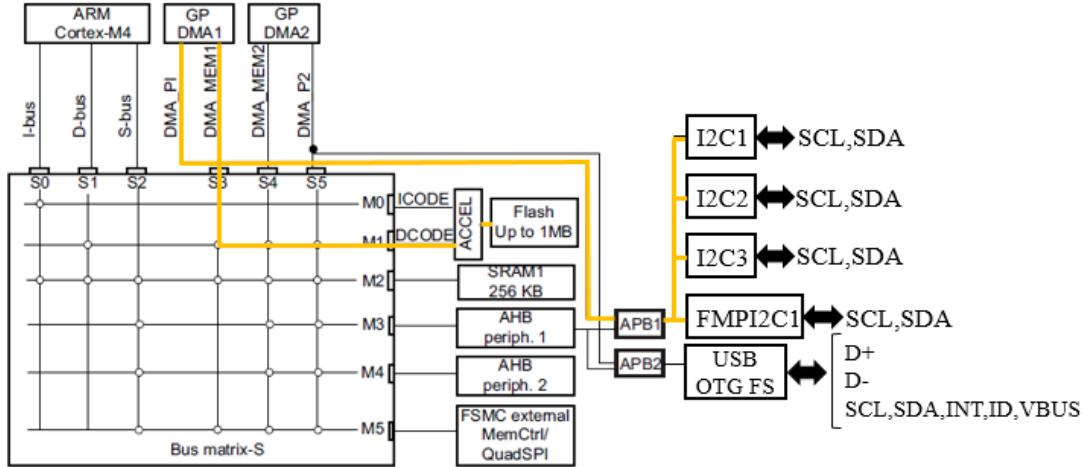


圖 2-8 Using DMA and multiple hardware I<sup>2</sup>C buses

存取完所有裝置的資料後，CPU 會透過 D-bus 從快取記憶體讀取資料，執行資料融合，並將融合後資料送到動作顯示模組，為了將傳送時間最小化，我們使用 USB 2.0 全速模式當作傳輸介面，USB 2.0 FS 的最大傳輸速率為 12 Mbits/sec，換言之，1.5Mbytes/sec，這個速度相當足夠。

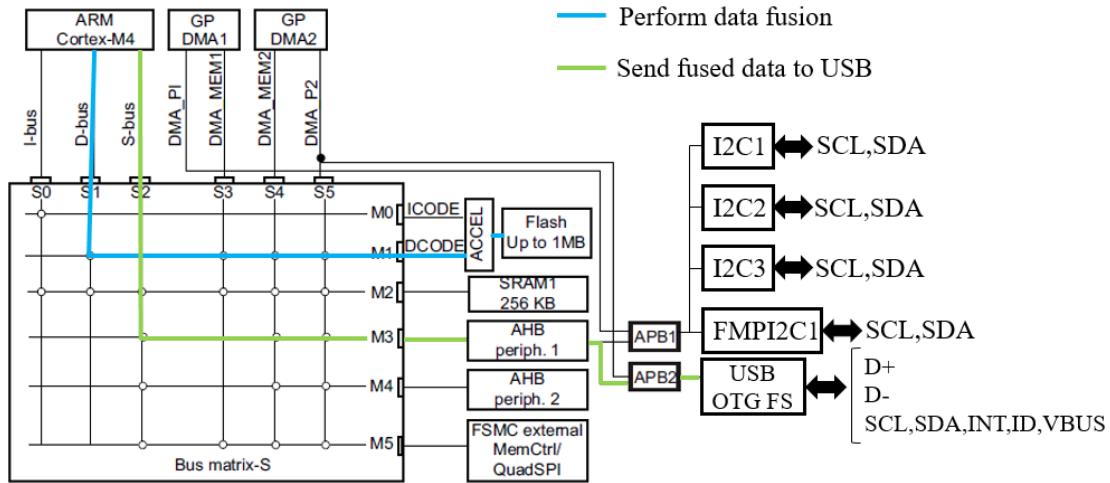


圖 2-9 Data fusion and transmitted

根據以上的討論，本系統將會實現一個有效率的資料獲取模組，可以達到取樣頻率 200Hz 以及支援 8 顆慣性感測器。

# 3. 系統設計

接下來的章節將會提供足夠的資訊讓開發者可以生產這個系統

## 3.1. 開發環境介紹

本節的內容將會介紹本系統在開發過程中使用的軟體，分別是開發軟體、測試軟體以及輔助工具，所有軟體都是可以在網路上自行下載的，接下來的內容會逐一介紹每款軟體的功能及特色。

### 3.1.1. 開發軟體

這部分將會介紹開發所使用的軟體，分別是 Keil μVision 5 以及 STM32CubeMX。

#### 3.1.1.1. Keil μVision 5

Keil 是德國知名軟體公司 Keil (現已併入 ARM 公司) 開發的微控制器軟體發展平臺，現在由國內三家代理商提供技術支援和相關服務。作為目前 ARM 內核單片機開發的主流工具，它提供了包括 C 編譯器、巨集組譯、連接器、庫管理、一個功能強大的模擬調試器在內的完整開發方案，並和一個基於 Cortex-M、Cortex-R4、ARM7、ARM9 等處理器設備所支援的開發環境(也就是 uVision)組合在一起，使得它在模擬方面功能變得非常強大，因此成為眾多開發 ARM 應用的工程師的首選。

下載網址: <https://www.keil.com/demo/eval/arm.htm>

下載教學: <https://www.youtube.com/watch?v=GOGDQQQgiGo>

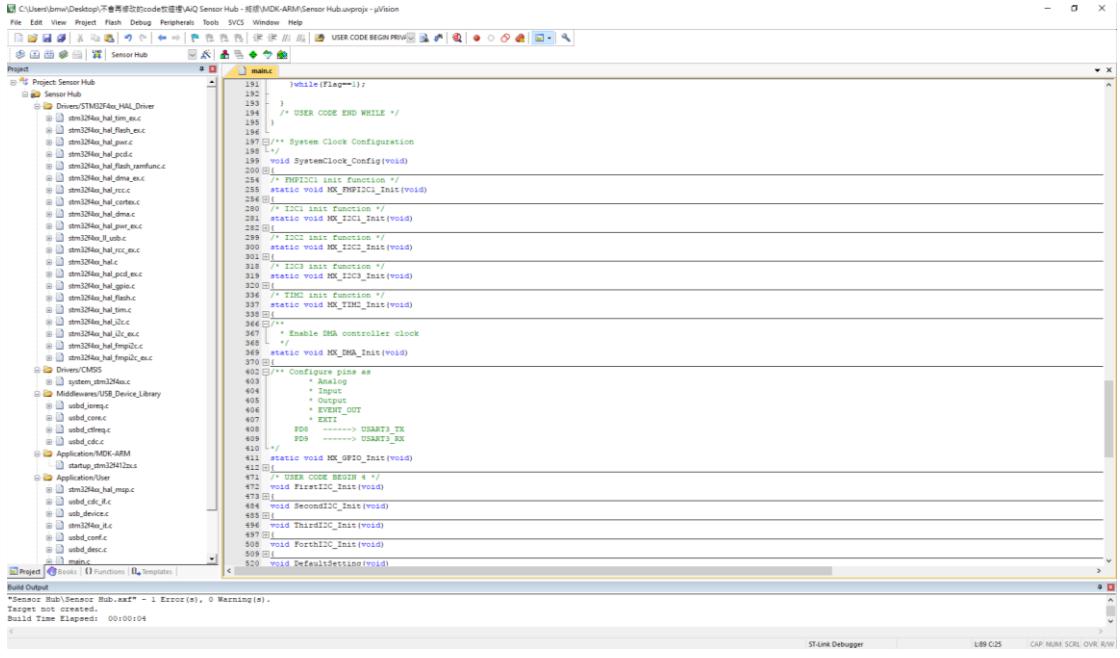


圖 3-1 Keil μVision 5 operating page

### 3.1.1.2. STM32CubeMX

本系統使用 STM32CubeMX 做大部分的設定，並透過 STM32CubeMX 來產生原始程式碼。STM32CubeMX 是由 STMicroelectronics 提供，可以大大的減少開發時間，STM32CubeMX 是一個配置 STM32 代碼的工具，它把很多東西封裝的比較好，硬體抽象層、中間層、示例代碼等，它包含了 STM32 所有系列的晶片，包含示例和樣本(Examples and demos)、中間組件(Middleware Components)、硬體抽象層(Hardware abstraction layer)。圖形化的介面使的操作更容易並且網路上有非常多的使用心得及問與答，ST 公司也有完整得教學例程，開發者可以依照步驟產生相對應的初始原始碼，再根據此原始碼做延伸與開發。

#### 【軟體特點】

1).直觀的選擇 STM32 微控制器。

2).微控制器圖形化配置：

-自動處理引腳衝突 -動態設置確定的時鐘樹

-可以動態確定參數設置的外圍和中間件模式和初始化

-功耗預測

3).C 代碼工程生成器覆蓋了 STM32 微控制器初始化編譯軟體，如 IAR,KEIL,GCC。

4).可獨立使用或作為 Eclipse 插件使用。

原文網址：<https://read01.com/2OgxxE.html>

下載網址：<http://www.st.com/en/embedded-software/stm32cubef4.html>

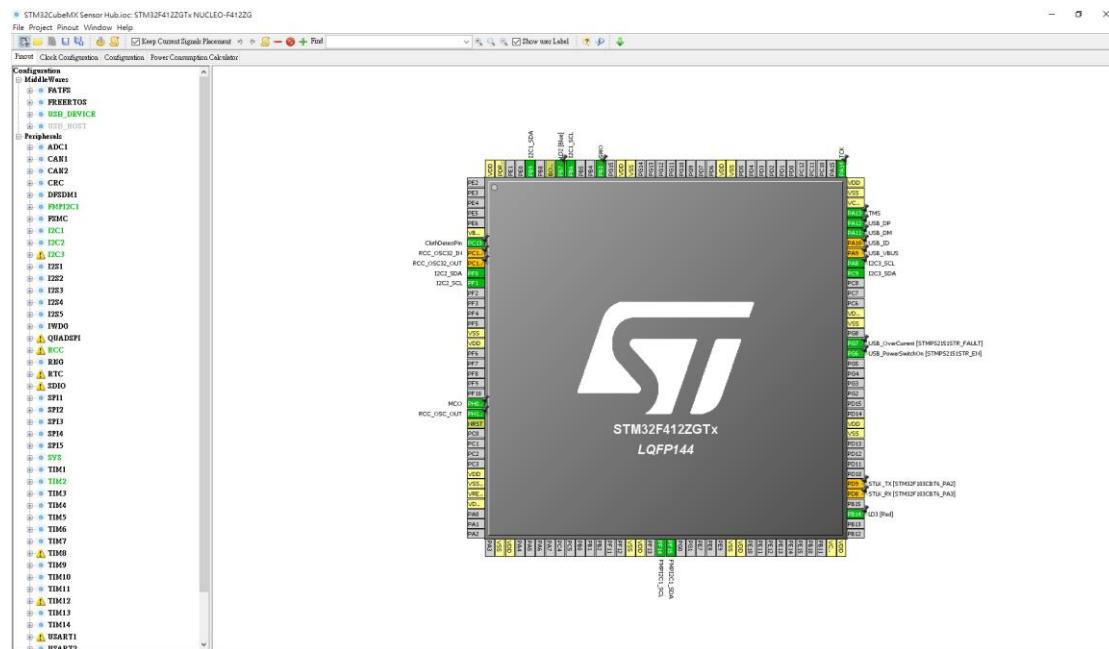


圖 3-2 STM32CubeMX operating page

### 3.1.2. 測試軟體

這部分將會介紹測試所使用的軟體，本系統使用 AccessPort137 做測試。

#### 3.1.2.1. AccessPort137

測試軟體方面，我們使用 Accessport137 與集線器做溝通。AccessPort 是一款先進的序列埠監聽、模擬與 RS232 資料分析的工具。

##### 【軟體特點】

- 1). 監控串口：具有埠監控功能，可以監控、攔截、保存所收發的資料（NT/2K/XP）。
- 2). 串口調試：支援常用的串口操作功能，支援大資料量的收發、保存，支援自動

發送。

3).動態變參：在不改變當前所打開埠的情況下，能動態改變埠參數（如：串列傳輸速率、校驗位、流控制等）。

4).雙模編輯：資料發送區內嵌十六進位編輯器（類似 UltraEdit），支援十六進位<=>文本雙模式切換編輯，支援 unicode。

5).國際版本：國際版，支援多國語言。

6).無需安裝：介面友好，方便易用。

下載網址：<http://www.sudt.com/en/ap/>



圖 3-3 AccessPort137 operating page

### 3.1.3. 輔助工具

這部分將會介紹所使用的輔助工具，分別是 USBDeview 以及 Device Monitor Studio。

#### 3.1.3.1. USBDeview

USBDeview 是一套小型免安裝的實用程式，其可列出目前已連接到使用者電腦的所有 USB 裝置以及所有使用者以前使用過的 USB 裝置，由於本系統需

要開發 USB 的功能，透過 USBDeview 可以監控到每一款 USB 裝置，方便使用者追蹤並實作。

對於每個 USB 裝置，將顯示其額外資訊：裝置名稱/描述、裝置類型、序號(大型存放裝置)、日期/時間(該裝置首次與最後使用)、廠商識別碼、產品識別碼等等...USBDeview 允許您解除安裝以前使用過的 USB 裝置，還可中斷目前已連接到使用者電腦的 USB 裝置。使用者也可以在遠端電腦上使用 USBDeview，只要您使用管理員權限登錄到該電腦即可，有不同的提示燈號告知使用者目前裝置的連接狀況。

灰燈：該裝置未連接。

綠燈：該裝置已連接。可以直接拔除該裝置而不需安全地移除。

紫燈：該裝置已連接。在您要將它直接拔除之前必須從 USBDeview 或 Windows 「安全地移除硬體」選項來中斷裝置連接。

紅燈：該裝置已停用。

下載網址：[http://www.nirsoft.net/utils/usb\\_devices\\_view.html](http://www.nirsoft.net/utils/usb_devices_view.html)

The screenshot shows the main interface of USBDeview. At the top, there's a menu bar with File, Edit, View, Options, Help. Below the menu is a toolbar with icons for New, Open, Save, Print, Find, and Help. The main area is a grid table with the following columns:

Device Name	Description	Device Type	Connected	Safe To Unplug	Disabled	USB Hub	Drive Letter	Serial Number	Created Date	Last Plug/Unplug	VendorID	ProductID	Firmware Rev.	USB Class
Port #0005.Hub_#0...	Apple iPhone	Mass Storage	No	Yes	No	0		1580049991200	2017/6/30 T#0...	2017/6/30 T#0...	0	1.00	06	0
Port #0005.Hub_#0...	Apple iPhone	Still Imaging	No	Yes	No	0			2017/6/30 T#0...	2017/6/30 T#0...	05sec	12a8	5.40	06
Port #0005.Hub_#0...	Apple Mobile Device Ethernet	Vendor Specific	No	Yes	No	0			2017/6/30 T#0...	2017/6/30 T#0...	05sec	12a8	5.40	ff
Port #0006.Hub_#0...	Apple Mobile Device USB Drive	Still Imaging	No	Yes	No	0		0469f1f25c3397a...	2017/6/30 T#0...	2017/6/30 T#0...	05sec	12a8	5.40	06
0000.0014.0000.006...	CDC - HID Composite	HID (Human Interface Device)	No	Yes	No	0			2017/6/30 T#0...	2017/6/30 T#0...	05sec	12a8	5.40	06
0000.0014.0000.006...	Intelligent Camera	Video	No	Yes	No	0	COM9		2017/6/30 T#0...	2017/6/30 T#0...	0483	5741	2.00	03
Port #0007.Hub_#0...	Bluetooth(R) Smart Bluetooth(R)	Bluetooth Device	Yes	Yes	No	0			2017/6/30 T#0...	2017/6/30 T#0...	05sec	0000	0.12	0e
Port #0008.Hub_#0...	Jeffelieh Transcend 8GB USB Device	Mass Storage	No	Yes	No	0		JQ40E7X6	2017/6/30 T#0...	2017/6/30 T#0...	05sec	0000	1.00	08
Port #0008.Hub_#0...	SRT USB Device	Mass Storage	No	Yes	No	E:		0000000005894C3...	2017/6/30 T#0...	2017/6/30 T#0...	05sec	0000	1.00	08
0000.0014.0000.006...	STMicroelectronics STLink Virtual COM Port	Vendor Specific	No	No	No	0			2017/7/27 T#1...	2017/7/27 T#1...	0483	374b	1.00	ff
0000.0014.0000.006...	STMicroelectronics STLink Virtual COM Port	Communication	No	No	No	0			2017/7/27 T#1...	2017/7/27 T#1...	0483	374b	1.00	ff
0000.0014.0000.006...	STMcomelectronics STLink Virtual COM Port	Communication	No	Yes	No	0	COM5		2017/7/27 T#1...	2017/7/27 T#1...	0483	374b	1.00	02
Port #0002.Hub_#0...	STMcomelectronics Virtual COM Port	Communication	No	Yes	No	0	COM7	00000000001A	2017/7/27 T#1...	2017/7/27 T#1...	0483	5740	2.00	02
Port #0018.Hub_#0...	Storulet Transcend USB Device	Mass Storage	No	Yes	No	0		5318 RDF903800...	2017/7/18 T#1...	2017/7/18 T#1...	174c	5106	80.00	08
Port #0004.Hub_#0...	Unknown Device (Device Descriptor Req.)	Unknown	No	No	No	0			2017/7/27 T#1...	2017/7/27 T#1...	0000	0000	0.00	00
Port #0004.Hub_#0...	Unknown Device (Device Descriptor Req.)	Unknown	No	No	No	0			2017/7/27 T#1...	2017/7/27 T#1...	0000	0000	0.00	00
Port #0001.Hub_#0...	US Composite Device	Unknown	No	Yes	No	D:		066FF3F32353547...	2017/7/27 T#1...	2017/7/27 T#1...	0483	374b	1.00	00
Port #0006.Hub_#0...	US Composite Device	Unknown	No	Yes	No	0		066FF3F32353547...	2017/7/27 T#1...	2017/7/27 T#1...	0483	374b	1.00	00
Port #0006.Hub_#0...	US Composite Device	Unknown	No	Yes	No	0		00000000001A	2017/6/3 T#0...	2017/6/3 T#0...	0483	572b	2.00	00
Port #0006.Hub_#0...	US Composite Device	Unknown	No	Yes	No	0		00000000001A	2017/6/3 T#0...	2017/6/3 T#0...	0483	5741	2.00	00
Port #0006.Hub_#0...	US Composite Device	Unknown	Yes	Yes	No	0			2017/6/3 T#0...	2017/6/3 T#0...	0483	374b	1.00	00
0000.0014.0000.006...	US Input Device	HID (Human Interface Device)	No	No	No	0			2017/6/3 T#0...	2017/6/3 T#0...	0483	572b	2.00	03
USB OPTICAL MOUSE	USB Input Device	HID (Human Interface Device)	Yes	Yes	No	0			2017/7/27 T#1...	2017/6/2 T#0...	093a	2510	1.00	03
0000.0014.0000.001...	USB Mass Storage Device	Mass Storage	No	Yes	No	D:			2017/7/27 T#1...	2017/6/16 T#0...	0483	374b	1.00	08
0000.0014.0000.006...	USB Mass Storage Device	Mass Storage	No	Yes	No	0			2017/7/27 T#1...	2017/6/16 T#0...	0483	374b	1.00	08
0000.0014.0000.006...	USB 存儲装置	Communication	No	Yes	No	0	COM6		2017/6/3 T#0...	2017/6/3 T#0...	0483	572b	2.00	02
0000.0014.0000.006...	USB 存儲装置	Communication	No	Yes	No	0	COM4		2017/6/3 T#0...	2017/6/3 T#0...	0483	5741	2.00	02
Port #0002.Hub_#0...	USB 旁路	Communication	No	Yes	No	0	COM8	00000000001A	2017/6/2 T#0...	2017/6/2 T#0...	0483	5730	2.00	02

圖 3-4 USBDeview operating page

### 3.1.3.2. Device Monitoring Studio

Device Monitoring Studio 是一款監視、分析和記錄通過電腦連接和埠傳來的資料的軟體工具。此軟體工具使您能夠監視和分析通過 USB 連接的設備，以及串列終端和網路連接。可以查看關於這些專案中的每一個的資訊，包括埠，速度，設備位址，連接狀態和從屬設備，它有一個很好的回應時間，它不負擔電腦的性能，有很多選項來修補。

#### 【軟體特點】

- 1). 支援 USB，網路和串口同時監控。
- 2). 允許調查資料流程的內容。
- 3). 允許跟蹤進出 PC 的所有資料。
- 4). 允許保存日誌，以便在不同的 PC 上重播。
- 5). 允許在幾秒鐘內從千百萬位元組的流量資料中查找模式。
- 6). 支援規則運算式模式搜索流量資料。

下載網址: <https://www.hhdsoftware.com/device-monitoring-studio>

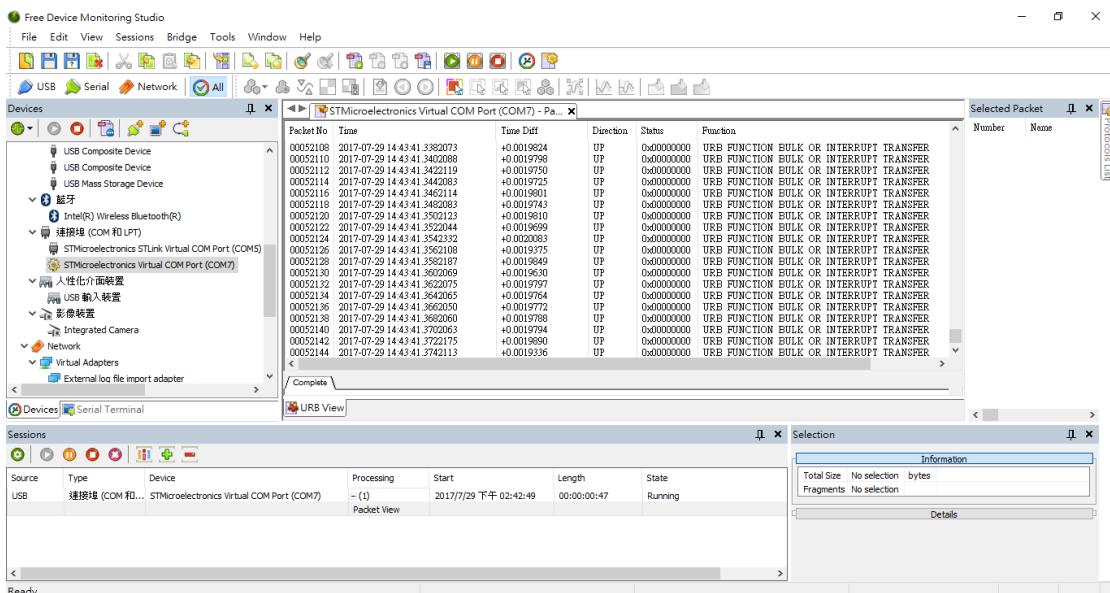


圖 3-5 Device Monitoring Studio operating page

## 3.2. 開發規格

本章節會描述程資料獲取模組的開發規格，本系統參考標準文件Smart Clothing Sensor Hub S/W Specification V0.9[3]做為開發標準。

### 3.2.1. 硬體規格

資料獲取模組需要連接 8 顆慣性感測器，每顆慣性感測器的數值輸出速率為 208Hz，慣性感測元件的加速度計靈敏度設為  $\pm 8g$ ，陀螺儀靈敏度設為 2000dps。並且使用慣性感測器上的 FIFO 來儲存捕捉到的資料，讓資料獲取模組每次都可以捕捉到最新的值，FIFO 的模式選擇連續模式(Continuous mode)，FIFO 的輸出速率設為 208Hz。

### 3.2.2. 融合後資料封包規範

封包的型式需參考標準文件第 7 頁[3]，如圖 3-6 所示。一筆完整的資料封包由標頭檔、時間戳記以及感測器資料組成，而感測器資料又由 I<sup>2</sup>C 匯流排編號、感測器型號、陀螺儀原始資料以及加速度計原始資料所組成，每個組成的大小標示在圖上，每筆封包為 113 bytes，其中每一個組成的介紹請見表 2。

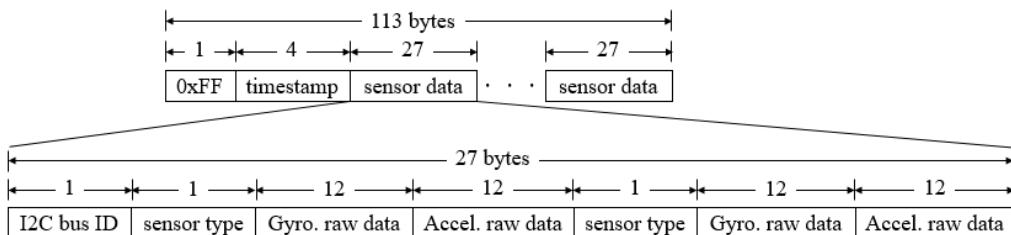


圖 3-6 Data format

表 2 Data format description

Item	Description	Length	Data
Timestamp	Timestamp of the read sensor data.	4 bytes	The timer count. Timer interval: 1ms
I <sup>2</sup> C bus ID	This field carries the I <sup>2</sup> C bus ID as well as the amount of IMU sensors connected.	1 byte	For example: 0x12: 1st I <sup>2</sup> C bus connects 2 IMU sensors
Sensor type	The type of sensor e.g. 6-axis or 9-axis IMU sensor	1 byte	0x01: 6-axis sensor 0x02: 9-axis sensor Others: reserved
Sensor raw data	Raw data of a IMU sensor	12 bytes / (18 bytes)	Accel X + Accel Y + Accel Z + Gyro X + Gyro Y + Gyro Z + (Magnet X + Magnet Y + Magnet Z)

### 3.3. 文件標準

本章節將會呈現程式碼的產生與介紹。圖 3-7 是本系統感測器集線器軟體部分的主程式，其中分為四個部分—集線器初始化設置、感測器參數設定、感測器初始化設置以及開始動作捕捉。

```
148 int main(void)
149 {
150     /* MCU Configuration-----*/
151
152     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
153     HAL_Init();
154
155     /* Configure the system clock */
156     SystemClock_Config();
157
158     /* Initialize all configured peripherals */
159     MX_GPIO_Init();
160     MX_DMA_Init();
161     MX_TIM2_Init();
162     MX_USB_DEVICE_Init();
163     MX_FMPI2C1_Init();
164     MX_I2C1_Init();
165     MX_I2C2_Init();
166     MX_I2C3_Init();
167
168     /* USER CODE BEGIN 2 */
169     HAL_TIM_Base_Start_IT(&htim2);
170     /* USER CODE END 2 */
171
172     /* Infinite loop */
173     /* USER CODE BEGIN WHILE */
174     while(1)
175     {
176         DefaultSetting();
177         GetClothInfo();
178         FirstI2C_Init();
179         SecondI2C_Init();
180         ThirdI2C_Init();
181         ForthI2C_Init();
182
183         do{
184             Read_Value();
185             Timestamp();
186             DataFusion();
187         }while(Flag==1);
188
189     }
190 }
```

圖 3-7 Main.c

其中：

程式碼 153 行到 169 行為集線器初始化設置。

程式碼 176 行為感測器參數設定。

程式碼 177 行到 181 行為感測器初始化設置。

程式碼 183 行到 187 行為動作捕捉。

3.2.1 到 3.2.4 將會介紹每一行程式碼的內容以及 STM32CubeMX 的設定。

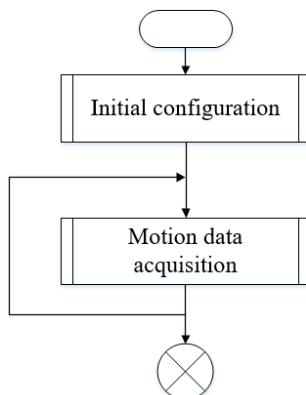


圖 3-8 Flow chart of Main.c

首先，要先產生程式原始碼，打開 STM32CubeMX，點下 New Project，選擇開發版種類、MCU 系列、正確的開發版型號，此計畫所使用的是 STM32F412ZGT6，因此選擇 STM32F412ZGTx 當作開發參考模組。如圖 3-9。

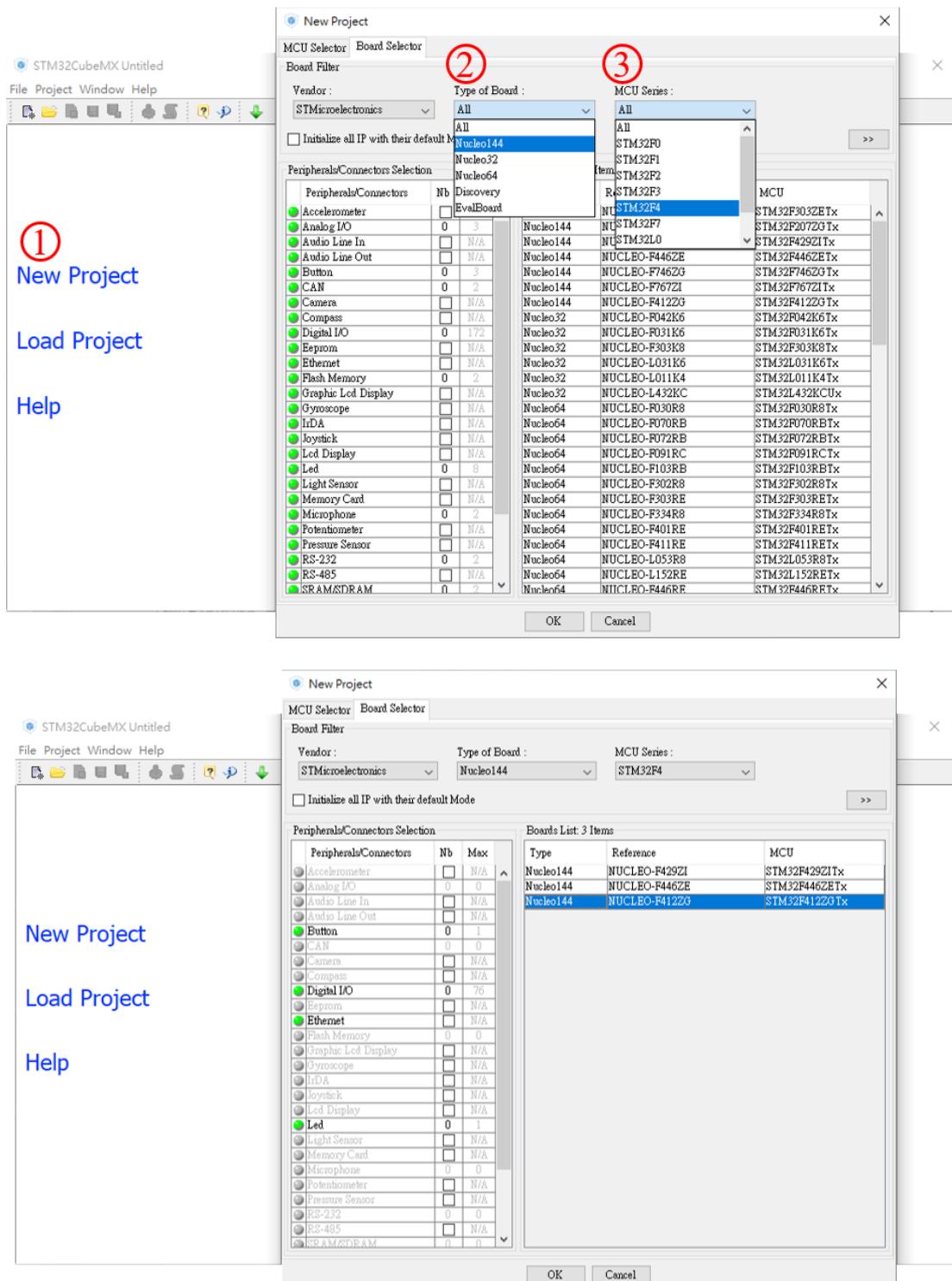


圖 3-9 Evaluation board configuration

接著會進入選擇設置的介面，圖是本系統中所使用的功能，分別為FMPI2C1、I2C1、I2C2、I2C3、RCC、TIM2、USB\_OTG\_FS。每一個功能的選項請參照圖 3-10 中的設定，詳細的功能介紹與設定會在後文中提及。

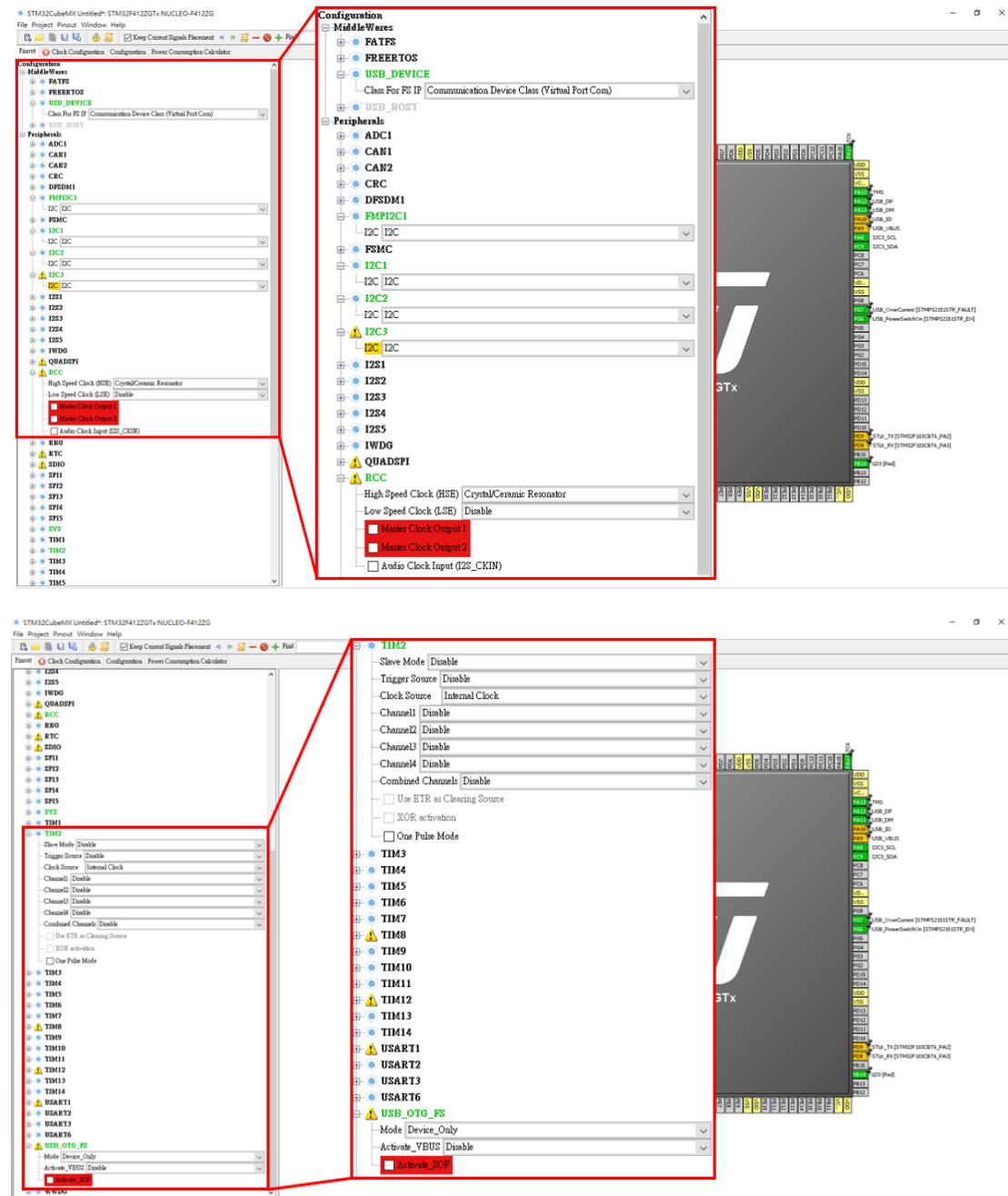


圖 3-10 STM32F412 function configuration

注意！其中 I2C3 的 PA8 腳位被系統預設給 USB\_OTG\_FS\_SOF，需要先將此腳位設定成 Reset\_State，才能開啟此腳位，如圖 3-11。

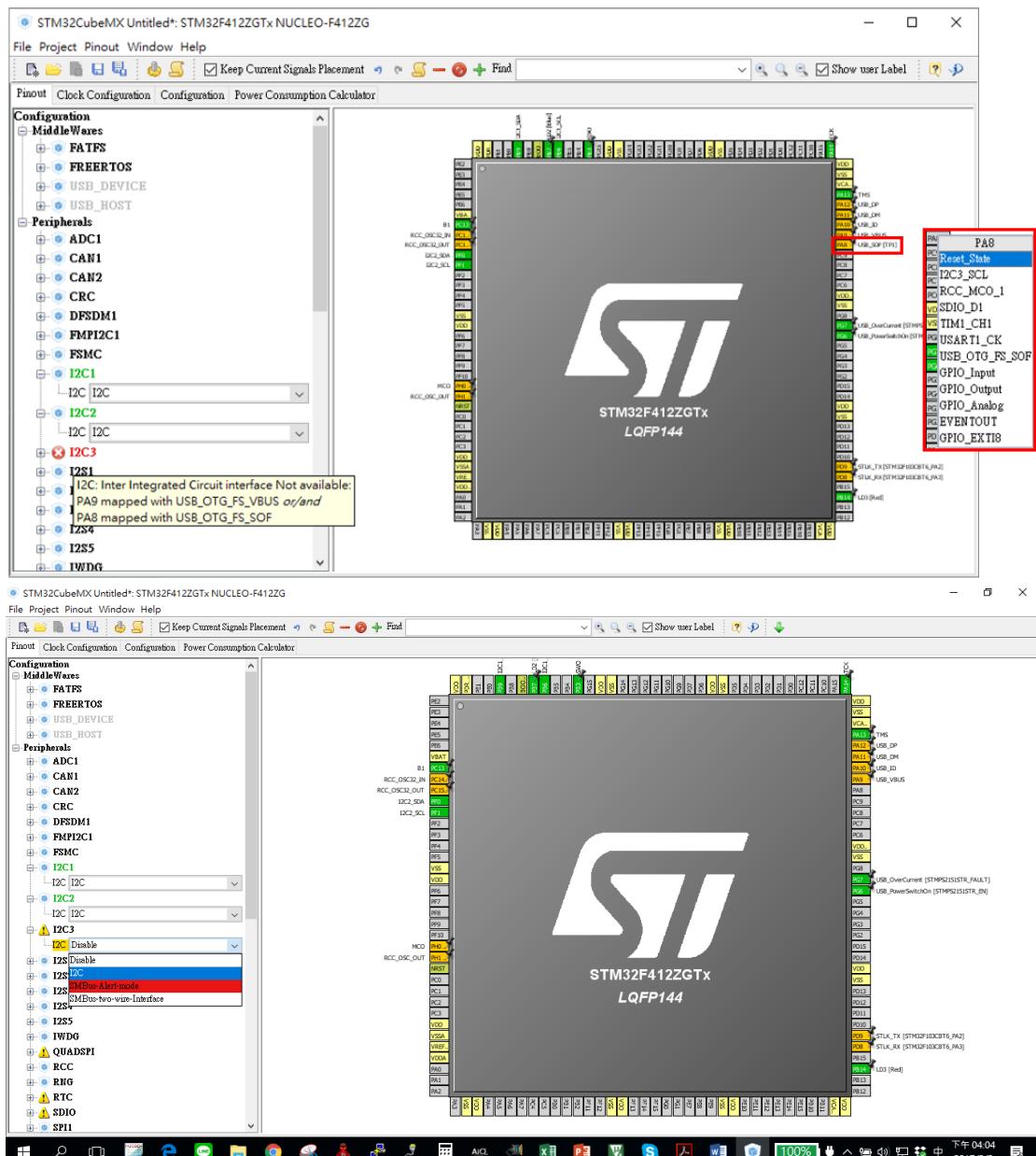


圖 3-11 Clear PA8

時脈與腳位的配置請參考 3.2.1 到 3.2.4 的內容，或是 Demo 影片 Create project 的操作示範，當所有配置都設定完成後，點選 Project->Generate Code，輸入專案名稱(Project name)，選擇開發環境，本系統使用 MDK-ARM V5(Keil μVision 5)，按下 OK 即可產生程式原始碼，使用者便可以基於該程式原始碼，開啟自己的應用開發。

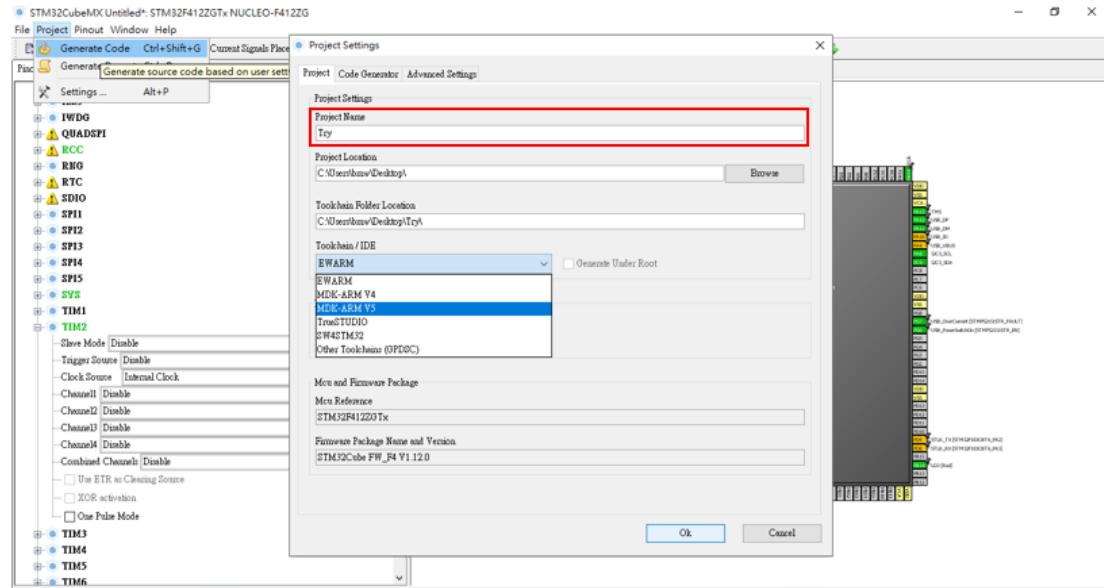


圖 3-12 Generate source code

### 3.3.1. 移植專案

當使用者創建好新專案後，可以選擇自行開發或是使用本系統所提供的程式碼做應用，若是要自行開發，請參考 3.3.2-3.3.6 的內容。

若是要使用本系統所提供的程式碼，請先參考 5.1-5.2，將本系統的專案內容下載到電腦中，接著打開專案所在的資料夾，如圖，本系統提供了兩種專案，一個是 General，另一個是 User Interface，General 是一般的專案，使用者無須自行設定慣性感測器參數，User Interface 是可以讓使用者自行設定慣性感測器參數的專案，使用者可以依照自己的需求來設定參數。

### 3.3.2. 微控制器初始化設置

本節會介紹微控制器初始化設置中，所使用到的函式，這些函式都是由 STM32CubeMX 產生，因此筆者在此只會介紹一些筆者有修改的參數，不做詳細的說明。

### 3.3.2.1. System Clock Configure

這一函式是用來配置系統的時脈，系統時脈決定了整個周邊裝置的頻率，頻率相同，才能做溝通，這部分將可以透過 STM32CubeMX 的 Clock Configuration 頁面做設定與調整。

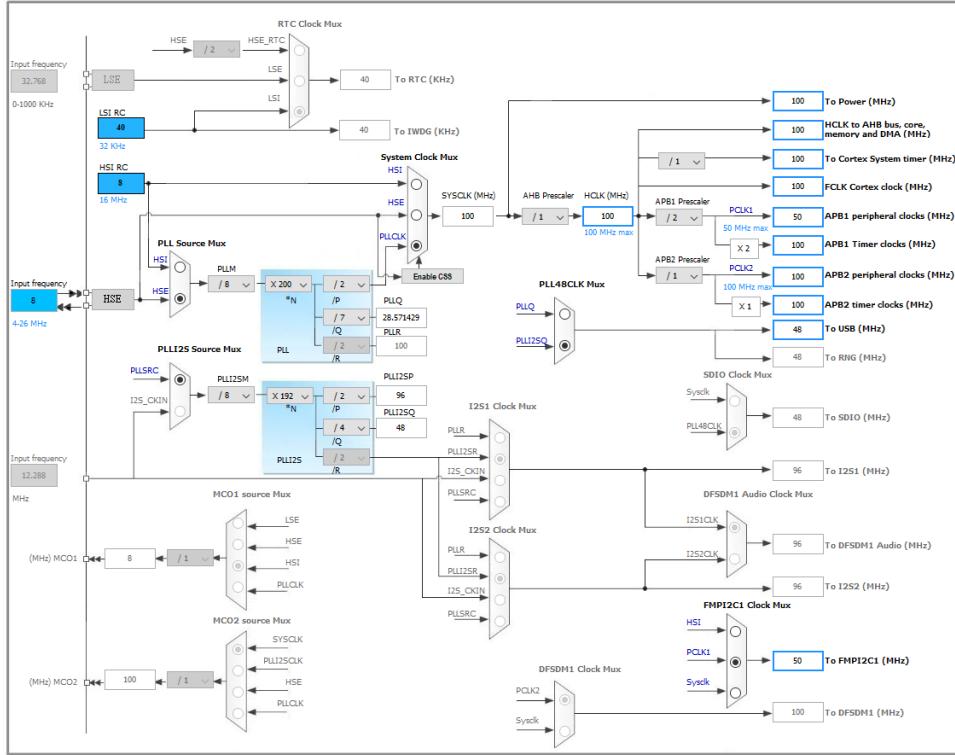


圖 3-13 STM32CubeMX-Clock Configuration

本系統所選擇的是 External High Speed Clock(HSE)，也就是外部震盪器所產生的時脈，來做為頻率來源，時脈參數的設定基準是參考 STMicroelectronics 所提供的範例程式碼來設定，經過參數的調教後，設定周邊裝置的時脈為 100MHz，且 USB 的時脈為 48MHz。

範例程式碼載點: <http://www.st.com/en/embedded-software/stm32cubef4.html>

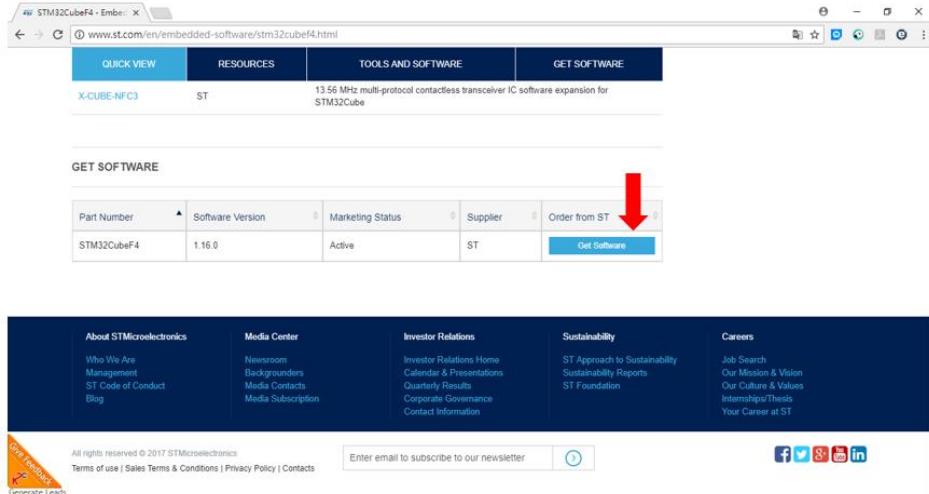


圖 3-14 Sample code download

```

213 void SystemClock_Config(void)
214 {
215
216     RCC_OscInitTypeDef RCC_OscInitStruct;
217     RCC_ClkInitTypeDef RCC_ClkInitStruct;
218     RCC_PeriphCLKInitTypeDef PeriphClkInitStruct;
219
220     __HAL_RCC_PWR_CLK_ENABLE();
221
222     __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
223
224     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
225     RCC_OscInitStruct.HSEState = RCC_HSE_ON;
226     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
227     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
228     RCC_OscInitStruct.PLL.PLLM = 8;
229     RCC_OscInitStruct.PLL.PLLN = 200;
230     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
231     RCC_OscInitStruct.PLL.PLLQ = 7;
232     RCC_OscInitStruct.PLL.PLLR = 2;
233     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
234     {
235         Error_Handler();
236     }
237
238     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_PCLK1;
239     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
240     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
241     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
242     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
243     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
244     {
245         Error_Handler();
246     }
247
248     PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_CLK48|RCC_PERIPHCLK_FMPI2C1;
249     PeriphClkInitStruct.PLLI2S.PLLI2SN = 192;
250     PeriphClkInitStruct.PLLI2S.PLLI2SM = 8;
251     PeriphClkInitStruct.PLLI2S.PLLI2SR = 2;
252     PeriphClkInitStruct.PLLI2S.PLLI2SQ = 4;
253     PeriphClkInitStruct.Clk48ClockSelection = RCC_CLK48CLKSOURCE_PLLI2SQ;
254     PeriphClkInitStruct.Fmpi2c1ClockSelection = RCC_FMPI2C1CLKSOURCE_APB;
255     if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
256     {
257         Error_Handler();
258     }
259
260     HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
261
262     HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
263
264     /* SysTick_IRQn interrupt configuration */
265     HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
266 }

```

圖 3-15 System Clock Configuration

注意到此程式中，程式碼第 260 行，這行程式碼是用來決定系統中 tick 的時脈，此 tick 與諸多程式有相關，其中我們有使用到的就是 HAL\_Delay()，詳情請見 3.3.4.1。

### 3.3.2.2. MX GPIO Initialization

這一函式是用來配置 I/O 腳位，在使用一開始 STM32CubeMX，我們需要選定所使用的開發板，當開發版決定時，STM32CubeMX 便會將預設的腳位設定，如 LED，讀者也可以自行新增 I/O 腳位，如本系統中的 Wake Up Pin，此函式還會開通腳位的時脈，使腳位可以正常運作。

```
501 static void MX_GPIO_Init(void)
502 {
503     GPIO_InitTypeDef GPIO_InitStruct;
504
505     /* GPIO Ports Clock Enable */
506     __HAL_RCC_GPIOC_CLK_ENABLE();
507     __HAL_RCC_GPIOF_CLK_ENABLE();
508     __HAL_RCC_GPIOH_CLK_ENABLE();
509     __HAL_RCC_GPIOB_CLK_ENABLE();
510     __HAL_RCC_GPIOD_CLK_ENABLE();
511     __HAL_RCC_GPIOG_CLK_ENABLE();
512     __HAL_RCC_GPIOA_CLK_ENABLE();
513
514     /*Configure GPIO pin : WakeUp_Pin */
515     GPIO_InitStruct.Pin = WakeUp_Pin;
516     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
517     GPIO_InitStruct.Pull = GPIO_NOPULL;
518     HAL_GPIO_Init(WakeUp_GPIO_Port, &GPIO_InitStruct);
519
520     /*Configure GPIO pins : LD3_Pin LD2_Pin */
521     GPIO_InitStruct.Pin = LD3_Pin|LD2_Pin;
522     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
523     GPIO_InitStruct.Pull = GPIO_NOPULL;
524     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
525     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
526
527     /*Configure GPIO pins : STLK_RX_Pin STLK_TX_Pin */
528     GPIO_InitStruct.Pin = STLK_RX_Pin|STLK_TX_Pin;
529     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
530     GPIO_InitStruct.Pull = GPIO_NOPULL;
531     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
532     GPIO_InitStruct.Alternate = GPIO_AF7_USART3;
533     HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
534
535     /*Configure GPIO pin : USB_PowerSwitchOn_Pin */
536     GPIO_InitStruct.Pin = USB_PowerSwitchOn_Pin;
537     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
538     GPIO_InitStruct.Pull = GPIO_NOPULL;
539     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
540     HAL_GPIO_Init(USB_PowerSwitchOn_GPIO_Port, &GPIO_InitStruct);
541 }
```

圖 3-16 GPIO Initialization

而針對每一個 I/O 腳位，如果想要更詳細的配置，透過 STM32CubeMX 的 Pin Configuration 頁面做設定與調整。

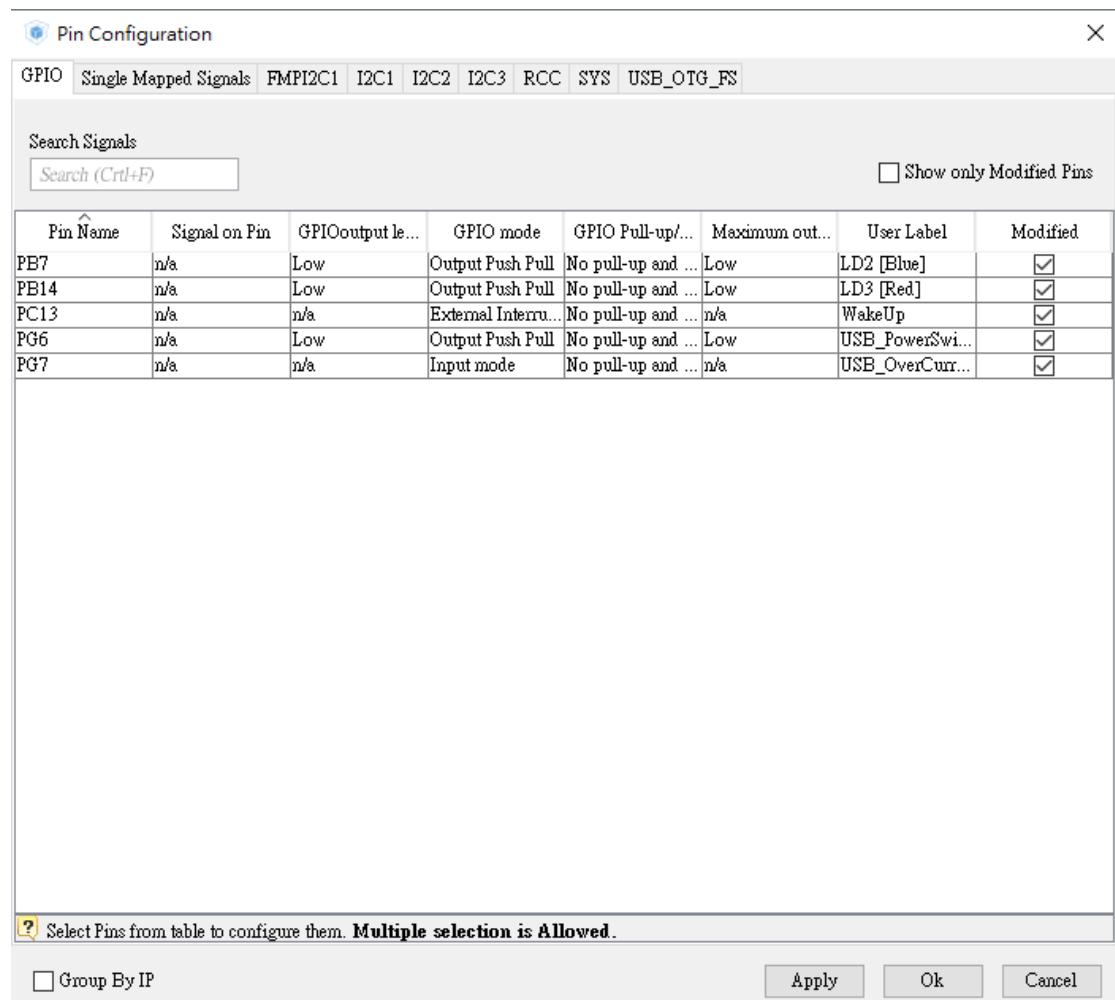


圖 3-17 STM32CubeMX-Pin Configuration

### 3.3.2.3. MX DMA Initialization

這一函式是用來設定 Direct Memory Access (DMA)，因為使用 STM32CubeMX 的緣故，分配 DMA 非常容易的事情，讀者只要依照標準文件的規範，便能輕鬆分配完 DMA，也可以依照需求來設定每一條 DMA 的優先權，詳情請見 STM32F412 Reference Manual Chapter 9: Direct memory access controller (DMA)[1]。

```

383 static void MX_DMA_Init(void)
384 {
385     /* DMA controller clock enable */
386     __HAL_RCC_DMA1_CLK_ENABLE();
387
388     /* DMA interrupt init */
389     /* DMA1_Stream0_IRQHandler configuration */
390     HAL_NVIC_SetPriority(DMA1_Stream0_IRQn, 0, 0);
391     HAL_NVIC_EnableIRQ(DMA1_Stream0_IRQn);
392     /* DMA1_Stream1_IRQHandler configuration */
393     HAL_NVIC_SetPriority(DMA1_Stream1_IRQn, 0, 0);
394     HAL_NVIC_EnableIRQ(DMA1_Stream1_IRQn);
395     /* DMA1_Stream2_IRQHandler configuration */
396     HAL_NVIC_SetPriority(DMA1_Stream2_IRQn, 0, 0);
397     HAL_NVIC_EnableIRQ(DMA1_Stream2_IRQn);
398     /* DMA1_Stream3_IRQHandler configuration */
399     HAL_NVIC_SetPriority(DMA1_Stream3_IRQn, 0, 0);
400     HAL_NVIC_EnableIRQ(DMA1_Stream3_IRQn);
401     /* DMA1_Stream4_IRQHandler configuration */
402     HAL_NVIC_SetPriority(DMA1_Stream4_IRQn, 0, 0);
403     HAL_NVIC_EnableIRQ(DMA1_Stream4_IRQn);
404     /* DMA1_Stream5_IRQHandler configuration */
405     HAL_NVIC_SetPriority(DMA1_Stream5_IRQn, 0, 0);
406     HAL_NVIC_EnableIRQ(DMA1_Stream5_IRQn);
407     /* DMA1_Stream6_IRQHandler configuration */
408     HAL_NVIC_SetPriority(DMA1_Stream6_IRQn, 0, 0);
409     HAL_NVIC_EnableIRQ(DMA1_Stream6_IRQn);
410     /* DMA1_Stream7_IRQHandler configuration */
411     HAL_NVIC_SetPriority(DMA1_Stream7_IRQn, 0, 0);
412     HAL_NVIC_EnableIRQ(DMA1_Stream7_IRQn);
413
414 }
415

```

圖 3-18 DMA Initialization

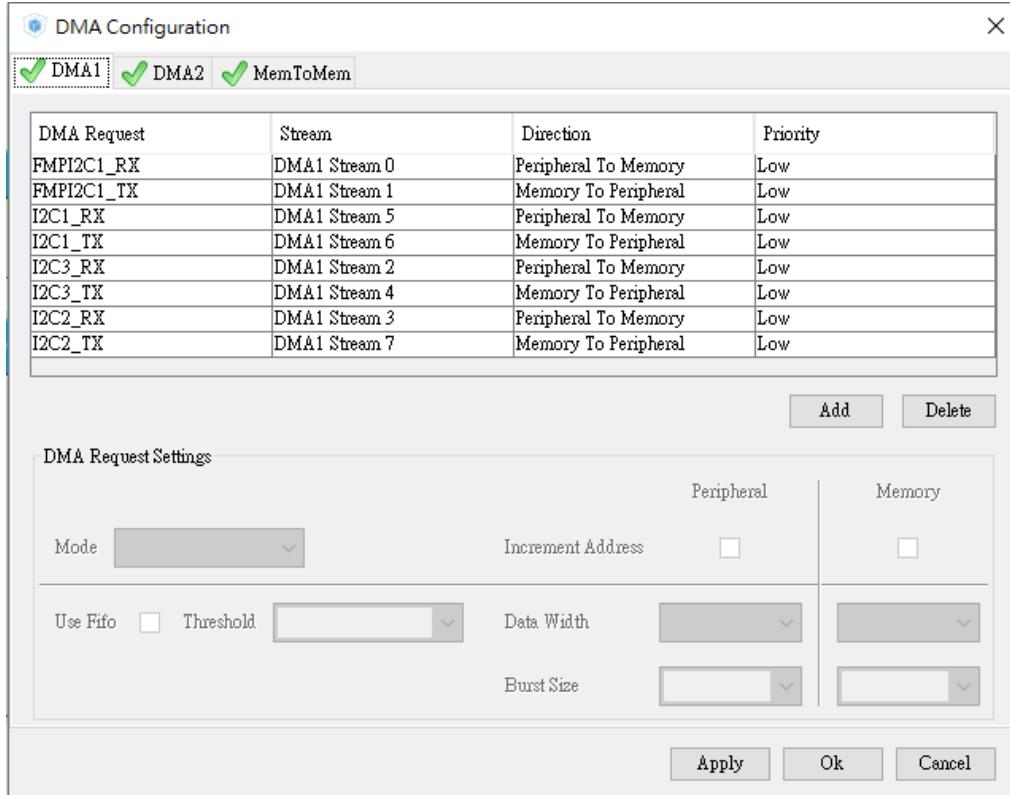


圖 3-19 STM32CubeMX-DMA configuration

### 3.3.2.4. MX USB Device Initialization

這一個函式是用來初始化 USB 功能，註冊 USB 類型，並啟用 USB 的功能。

```
47 void MX_USB_DEVICE_Init(void)
48 {
49     /* Init Device Library,Add Supported Class and Start the library*/
50     USBD_Init(&hUsbDeviceFS, &FS_Desc, DEVICE_FS);
51
52     USBD_RegisterClass(&hUsbDeviceFS, &USBD_CDC);
53
54     USBD_CDC_RegisterInterface(&hUsbDeviceFS, &USBD_Interface_fops_FS);
55
56     USBD_Start(&hUsbDeviceFS);
```

圖 3-20 MX USB Device Initialization

其中，我們會使用到 CDC\_Transmit\_FS()來發送訊息，此函式在 usbd\_cdc\_it.c 中，但這函式並非完整，此函式的輸入有 Len，但函式內容並未使用到，因此需要新增一行程式碼來解決此問題。

```
270 uint8_t CDC_Transmit_FS(uint8_t* Buf, uint16_t Len)
271 {
272     uint8_t result = USBD_OK;
273     /* USER CODE BEGIN 7 */
274     memcpy(UserTxBufferFS,Buf,sizeof(uint8_t)*Len);
275     USBD_CDC_HandleTypeDef *hcdc = (USBD_CDC_HandleTypeDef*)hUsbDeviceFS.pClassData;
276     if (hcdc->TxState != 0){
277         return USBD_BUSY;
278     }
279     USBD_CDC_SetTxBuffer(&hUsbDeviceFS, Buf, Len);
280     result = USBD_CDC_TransmitPacket(&hUsbDeviceFS);
281     /* USER CODE END 7 */
282     return result;
283 }
284 }
```

圖 3-21 CDC\_Transmit\_FS()

### 3.3.2.5. MX TIM2 Initialization

STM32F412 總共提供 14 個 Timer 供開發者使用，請見下表。

TIM #	Purpose
TIM1 & TIM8	Advance Control Timer
TIM2 – TIM5	General Purpose Timer
TIM6 & TIM7	Basic Timer
TIM9 – TIM14	General Purpose Timer

表 3-3 Timer

Timer 顧名思義就是每一段固定時間會做某一件事情，因此使用者可以自行設定這段固定的時間。本系統選擇開啟 TIM2 的功能，當作紀錄 timestamp 的標

準，以下是本系統舉例：

由於 TIM2 是由 APB1 Timer Clock 當作時脈來源，因此 TIM2 的時脈為 100MHz，在 TIM 的設定上，主要調整兩個參數，第一為程式碼 358 行的除頻器，除頻器的範圍落在  $0 - 2^{16}$ ，第二為程式碼 360 行的週期，週期的範圍落在  $0 - 2^{32}$ 。筆者想設定這段固定時間為 1 毫秒，固設定除頻器為 9999，週期為 9。

已知時脈為 100MHz，先經過除頻器後，為  $\frac{100 \times 10^6}{9999+1} = 10000$ ，這代表記數到 10000 時為一秒，再經過週期的計算，為  $\frac{10000}{9+1} = 1000$ ，代表這一秒內，系統會觸發 1000 次，因此可達到每 1 毫秒觸發一次的目的。

```
350  /* TIM2 init function */
351  static void MX_TIM2_Init(void)
352  {
353
354      TIM_ClockConfigTypeDef sClockSourceConfig;
355      TIM_MasterConfigTypeDef sMasterConfig;
356
357      htim2.Instance = TIM2;
358      htim2.Init.Prescaler = 9999;
359      htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
360      htim2.Init.Period = 9;
361      htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
362      if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
363      {
364          Error_Handler();
365      }
366
367      sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
368      if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
369      {
370          Error_Handler();
371      }
372
373      sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
374      sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
375      if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
376      {
377          Error_Handler();
378      }
379 }
```

圖 3-22 MX TIM2 Initialization

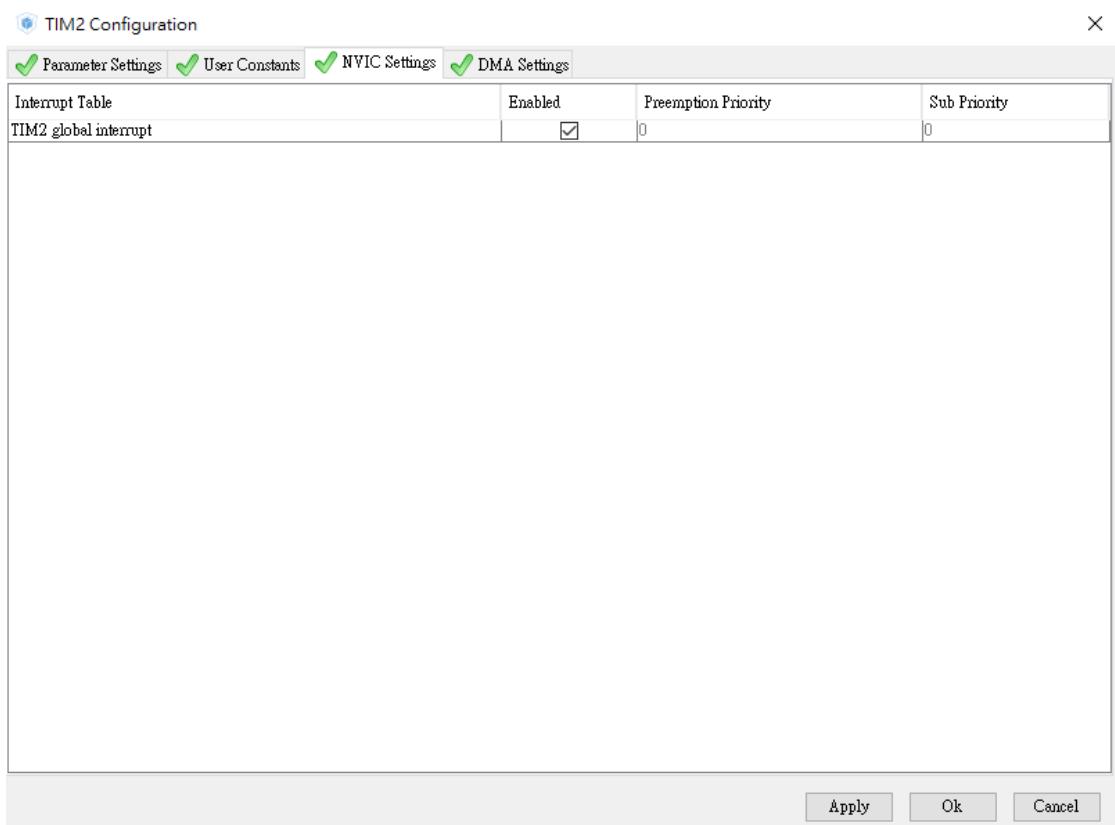
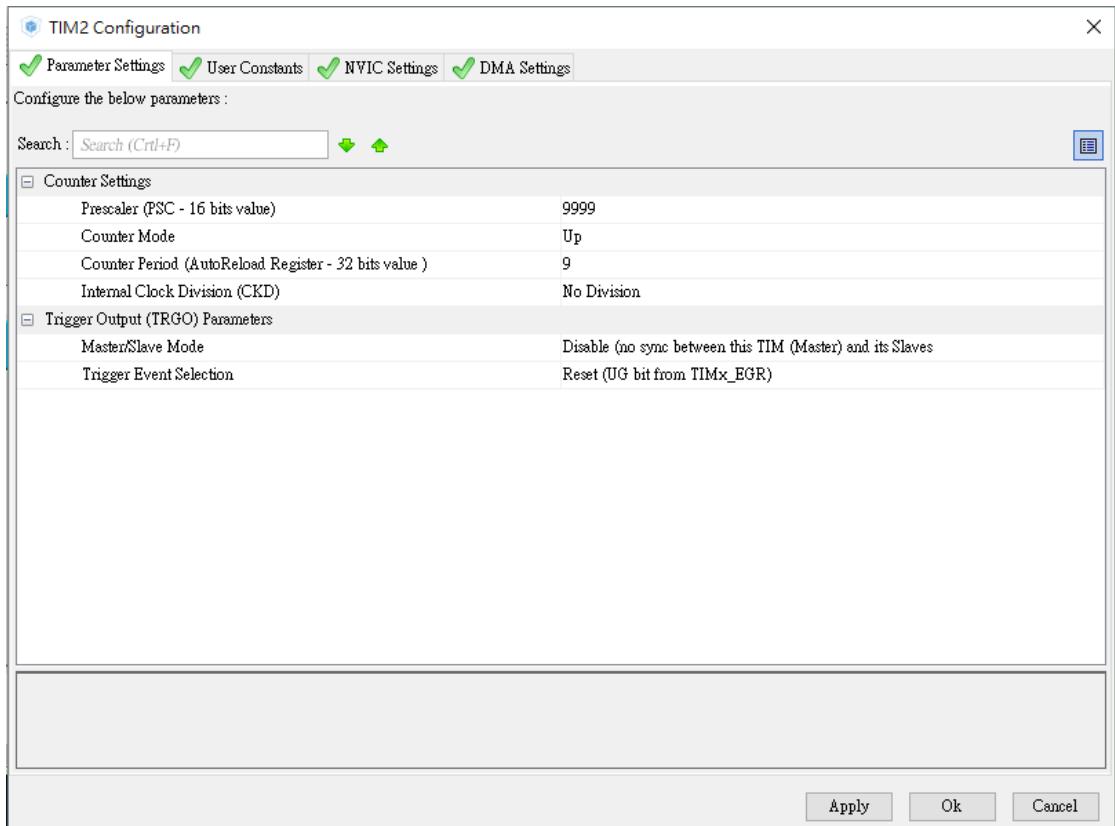


圖 3-23 TIM2 Configuration

### 3.3.2.6. MX I2C Initialization

本系統總共使用 4 組 I<sup>2</sup>C 來做通訊，其中 3 組為一般的 I<sup>2</sup>C，1 組為 Fast Mode Plus 的 I<sup>2</sup>C。

先介紹一般的 I<sup>2</sup>C，I<sup>2</sup>C 有兩個模式可以選擇，分別是 Standard Mode 與 Fast Mode，Standard 的通信速度最高可為 100kHz，而 Fast Mode 的通信速度可達 400kHz，I<sup>2</sup>C 的時脈速度將會決定整體資料交換的速度，因此可以依照個人需求來做設定。下面列的三個函式是用來初始化 I<sup>2</sup>C 通訊界面的，並且設定 I<sup>2</sup>C 的時脈速度。可以看到程式碼 291 的 Clock speed 為 150,000，單位為 Hz，也就是 150kHz，屬於 I<sup>2</sup>C 的 Standard mode，通信速度也可以透過 STM32CubeMX 來做設定，詳情請看 STM32F412 Reference Manual Chapter 24: Inter-integrated circuit (I<sup>2</sup>C) interface[1]。

```
287 static void MX_I2C1_Init(void)
288 {
289
290     hi2c1.Instance = I2C1;
291     hi2c1.Init.ClockSpeed = 150000;
292     hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
293     hi2c1.Init.OwnAddress1 = 0;
294     hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
295     hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
296     hi2c1.Init.OwnAddress2 = 0;
297     hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
298     hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
299     if (HAL_I2C_Init(&hi2c1) != HAL_OK)
300     {
301         Error_Handler();
302     }
303 }
304 }
```

圖 3-24 MX I2C Initialization

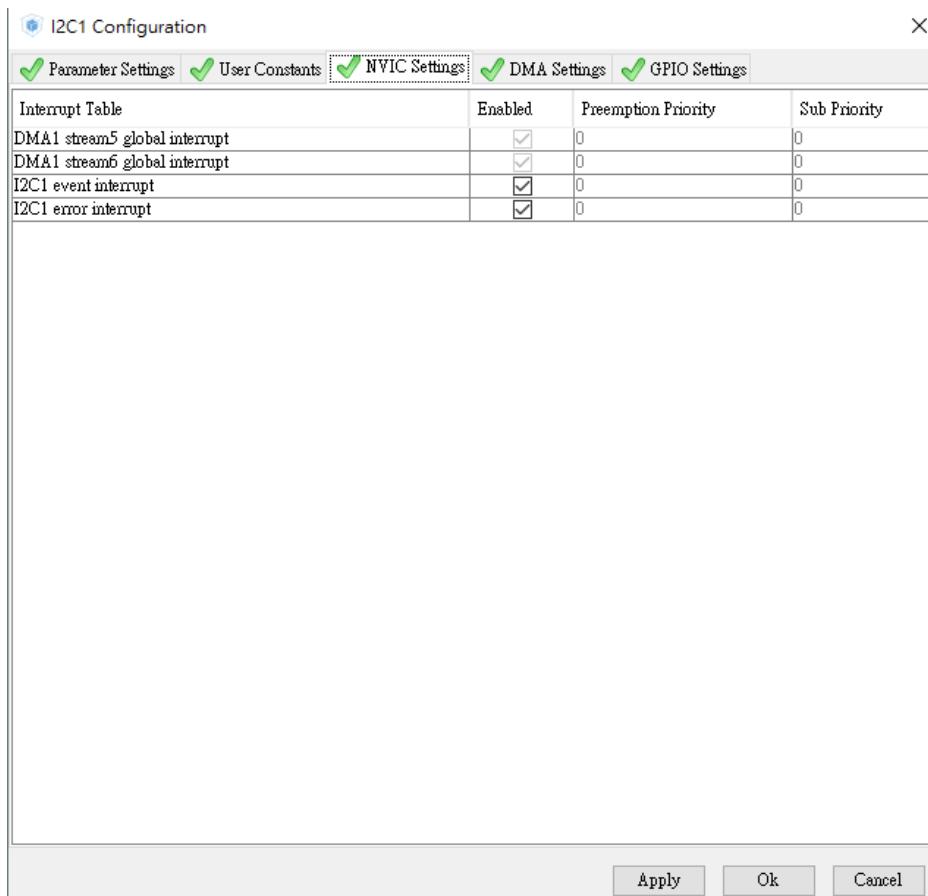
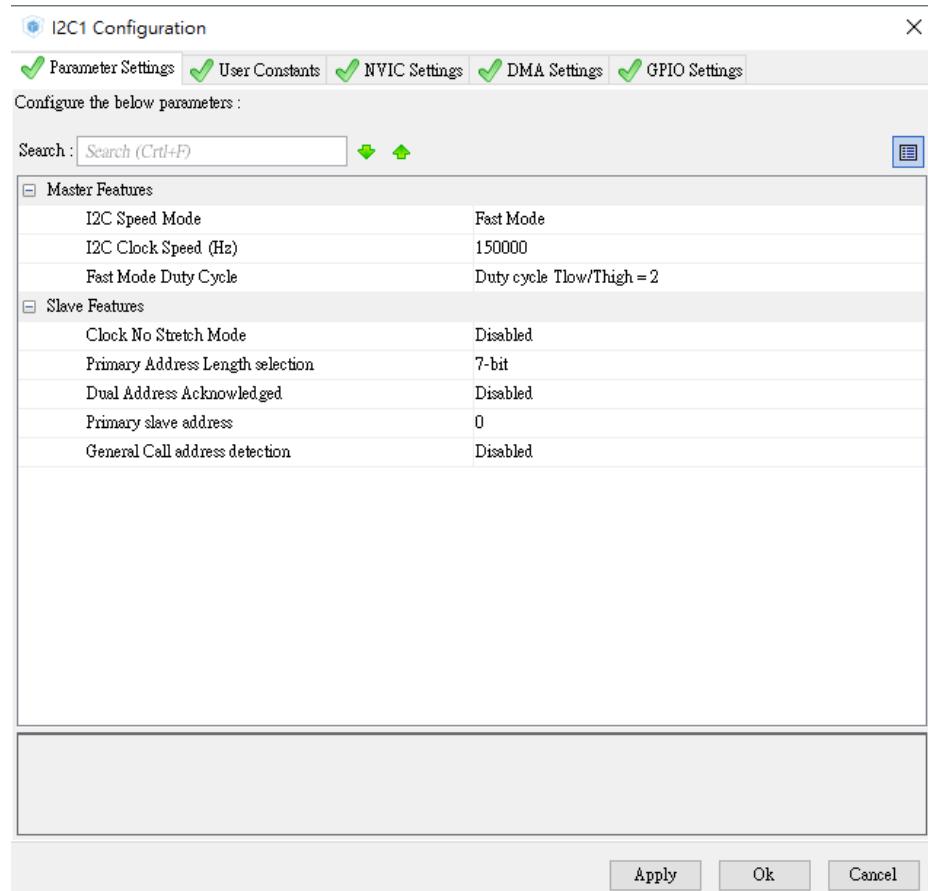


圖 3-25 I2C1 Configuration

### 3.3.2.7. MX FMPI2C1 Initialization

Fast Mode Plus I<sup>2</sup>C，也就是比 Fast Mode 的速度更快一階級，可高達 1MHz。這一函式是用來初始化 FMPI2C 的通訊界面，這邊的設定就不像一般 I<sup>2</sup>C 那麼直觀，不過一樣可以透過 STM32CubeMX 來設定速度，詳情請看 STM32F412 Reference Manual Chapter 23: Fast-mode Plus Inter-integrated circuit (FMPI2C) interface[1]。

```
250 static void MX_FMPI2C1_Init(void)
251 {
252     hfmpi2c1.Instance = FMPI2C1;
253     hfmpi2c1.Init.Timing = 0x20100D5D;
254     hfmpi2c1.Init.OwnAddress1 = 0;
255     hfmpi2c1.Init.AddressingMode = FMPI2C_ADDRESSINGMODE_7BIT;
256     hfmpi2c1.Init.DualAddressMode = FMPI2C_DUALADDRESS_DISABLE;
257     hfmpi2c1.Init.OwnAddress2 = 0;
258     hfmpi2c1.Init.OwnAddress2Masks = FMPI2C_OA2_NOMASK;
259     hfmpi2c1.Init.GeneralCallMode = FMPI2C_GENERALCALL_DISABLE;
260     hfmpi2c1.Init.NoStretchMode = FMPI2C_NOSTRETCH_DISABLE;
261     if (HAL_FMPI2C_Init(&hfmpi2c1) != HAL_OK)
262     {
263         Error_Handler();
264     }
265
266
267     /**Configure Analogue filter
268      */
269     if (HAL_FMPI2CEx_ConfigAnalogFilter(&hfmpi2c1, FMPI2C_ANALOGFILTER_ENABLE) != HAL_OK)
270     {
271         Error_Handler();
272     }
273
274 }
```

圖 3-26 MX FMPI2C Initialization

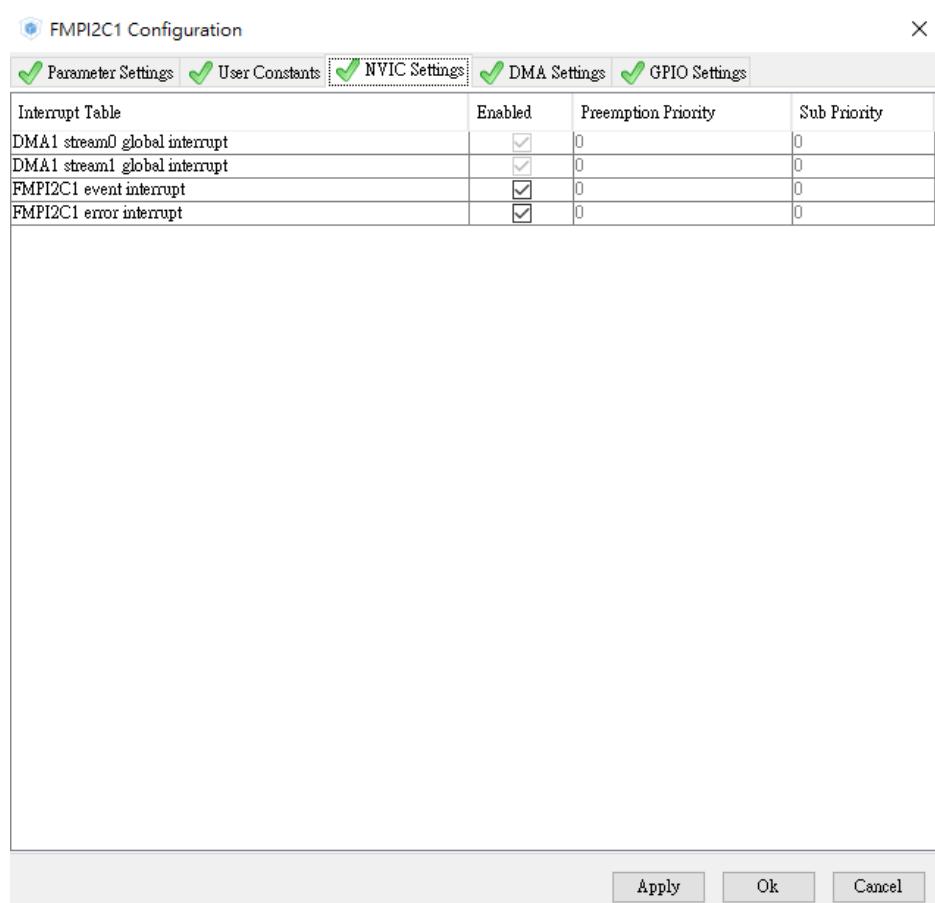
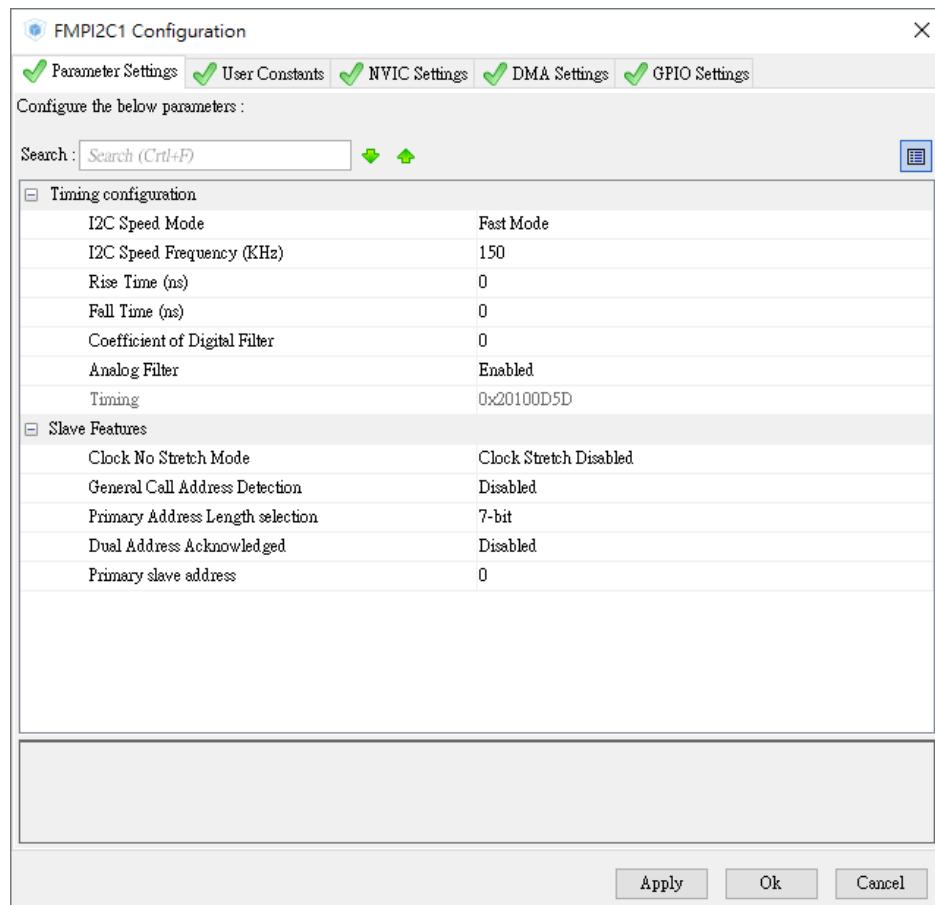


圖 3-27 FMPI2C Configuration

### 3.3.3. 感測器參數設定

微控制器初始化設置後，系統已經設定了時序、GPIO、USB 等等功能，接著要決定感測器的參數為何，本系統設定了 FIFO\_CTRL5、CTRL1\_XL、CTRL2\_G、CTRL6\_C、CTRL7\_G。

#### 3.3.3.1. FIFO\_CTRL5

此 FIFO\_CTRL5 暫存器包含 FIFO 的運作模式以及 FIFO 的輸出傳輸速度，關於 FIFO 的運作模式請見 LSM6DS33 Data Sheet 6.2 節[2]，在本系統預設中，FIFO 的模式使用 Continuous 模式，這個模式下緩衝器會依照順序儲存到來的資料，當緩衝器裝滿時，便會刪除最老舊的資料，以便裝下最新的資料。FIFO 的輸出傳輸速度則是設定為 208Hz，與加速度計及陀螺儀的輸出速度相同。

0 <sup>(1)</sup>	ODR_FIFO_3	ODR_FIFO_2	ODR_FIFO_1	ODR_FIFO_0	FIFO_MODE_2	FIFO_MODE_1	FIFO_MODE_0
------------------	------------	------------	------------	------------	-------------	-------------	-------------

圖 3-28 FIFO\_CTRL5 register

#### 3.3.3.2. CTRL1\_XL

此 CTRL1\_XL 暫存器包含輸出傳輸速度、靈敏度及濾波器頻寬。感測器中的加速度計初始狀態是 power-off 模式，需要透過此暫存器來啟動，輸出傳輸速度有多種選擇，前文提到不同的速度會有不同的功耗，而感測器的功耗與工作模式有關，工作模式由 CTRL6\_C 的 XL\_HM\_MODE 來設定，濾波器頻寬只有在高效能模式時才需要設定，此系統在普通效能模式，因此無須設定。此系統預設為輸出傳輸速度是 208Hz，靈敏度是±8g。

ODR_XL3	ODR_XL2	ODR_XL1	ODR_XL0	FS_XL1	FS_XL0	BW_XL1	BW_XL0
---------	---------	---------	---------	--------	--------	--------	--------

圖 3-29 CTRL1\_XL register

#### 3.3.3.3. CTRL2\_G

此 CTRL2\_G 暫存器包含輸出傳輸速度及靈敏度。感測器中陀螺儀的初始狀態是 power-off 模式，需要透過此暫存器來啟動，輸出傳輸速度有多種選擇，前

文提到不同的速度會有不同的功耗，而感測器的功耗與工作模式有關，工作模式由 CTRL7\_G 的 G\_HM\_MODE 來設定，此系統預設為輸出傳輸速度是 208Hz，靈敏度是±2000dps。

ODR_G3	ODR_G2	ODR_G1	ODR_G0	FS_G1	FS_G0	FS_125	0 <sup>(1)</sup>
--------	--------	--------	--------	-------	-------	--------	------------------

圖 3-30 CTRL2\_G register

### 3.3.3.4. CTRL6\_C

我們修改此 CTRL6\_C 的 XL\_HM\_MODE，因為整體系統要考量低功耗設計，因此將此值設定為 1:取消高效能模式，此暫存器的其他功能用不到，所以設定預設值即可。

TRIG_EN	LVLen	LVL2_EN	XL_HM_MODE	0 <sup>(1)</sup>	0 <sup>(1)</sup>	0 <sup>(1)</sup>	0 <sup>(1)</sup>
---------	-------	---------	------------	------------------	------------------	------------------	------------------

圖 3-31 CTRL6\_C register

### 3.3.3.5. CTRL7\_G

我們修改此 CTRL7\_G 的 G\_HM\_MODE，因為整體系統要考量低功耗設計，因此將此值設定為 1:取消高效能模式，此暫存器的其他功能用不到，所以設定預設值即可。

G_HM_MODE	HP_G_EN	HPCF_G1	HPCF_G0	HP_G_R_ST	ROUNDING_STATUS	0 <sup>(1)</sup>	0 <sup>(1)</sup>
-----------	---------	---------	---------	-----------	-----------------	------------------	------------------

圖 3-32 CTRL7\_G register

### 3.3.3.6. 使用者介面

本系統也撰寫了一個使用者介面的功能，使用者可以透過 AccessPort 來設定感測器參數，設定時請參考 LSM6DS33 Data Sheet[2]，圖 3-33 是 AccessPort 介面，使用者可以遵循以下的設定方法來設定參數。

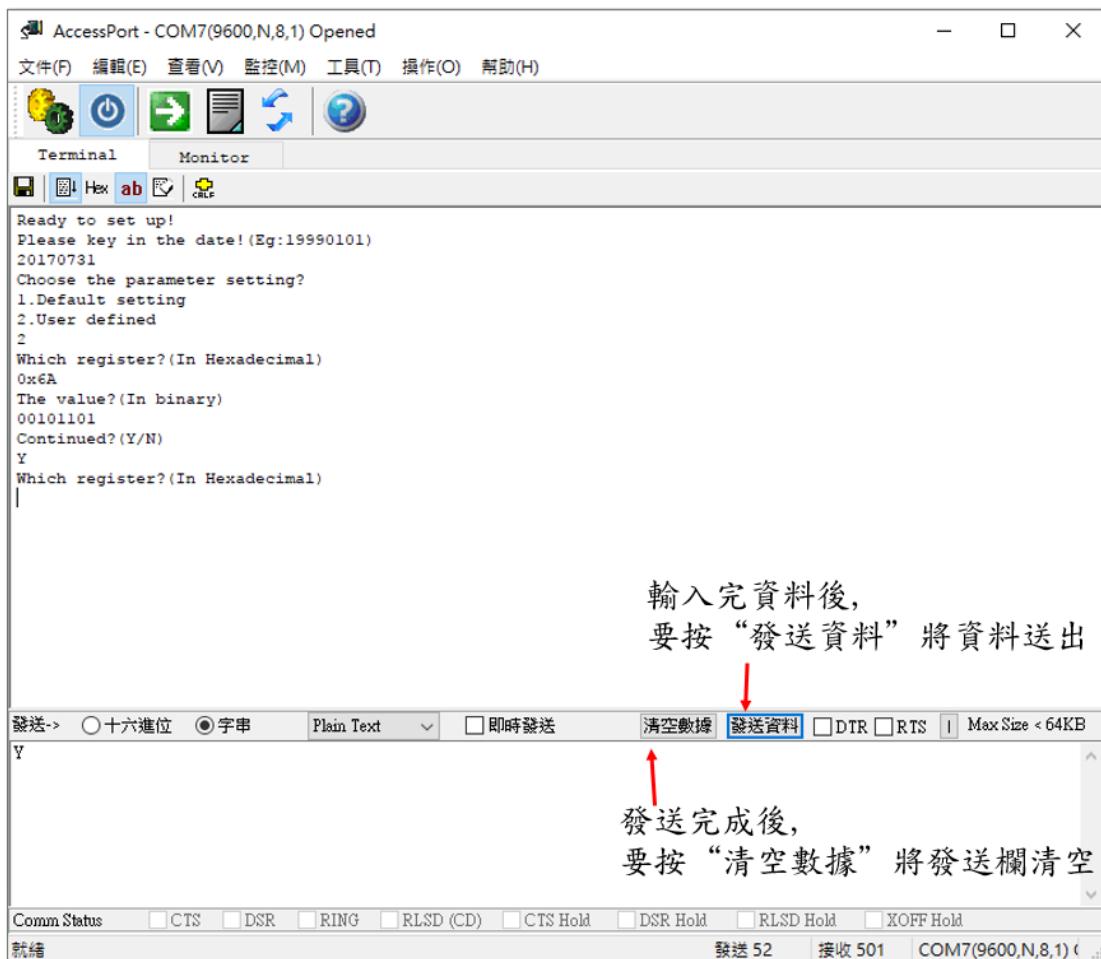


圖 3-33 AccessPort

圖 3-34 是設定感測器參數的步驟。當微控制器開機後(上電後)，會先作微控制器初始化設置，接著便會進入設定階段。

步驟一、打開 AccessPort，會看到”Ready to set up!”，告知使用者現在可以進行設定。

步驟二、接著會出現”Please key in the date!(Eg:19990101)”，這時使用者需要入當時的日期，西元+月份+日期，輸入完成後按 **發送資料** 發送，發送完成後，記得按 **清空數據** 將發送欄清空。

步驟三、日期輸入完程後，便會出現”Choose the parameter setting?”系統提供兩個選擇，1 是使用預設設定，2 是使用者自行設定。若選擇預設設定，則輸入 1 並按 **發送資料** 發送，這時系統便會採用預設值來做感測器初始化設置。若使用者想

要自行設定，即輸入 2 並按 **發送資料** 發送，發送完成記得按 **清空數據** 將發送欄清空。

步驟四、接著就會出現”Which Register?(In Hexadecimal)”告知使用者輸入想要設定的暫存器名稱，注意，必須輸入十六進制，輸入完成後要記得多按一下空白鍵，例如”0x6A “，輸入完按 **發送資料** 發送。發送完成後便會出現”The value?(In binary)”告知使用者輸入暫存器的值。這時要先按 **清空數據** 將發送欄清空，在輸入暫存器的數值，注意，必須輸入二進制，輸入完成後要記得多按一下空白鍵，例如”00111110 “，輸入完按 **發送資料** 發送。

步驟五、發送成功後便會出現 Continued? 告知使用者是否需要繼續設定，如果要請輸入 Y，系統便會重複步驟四，如果不需繼續設定，請輸入 N，系統便會依照使用者輸入的參數做感測器初始化設置。

步驟六、結束，關閉 AccessPort 並打開 MEM 的軟體。

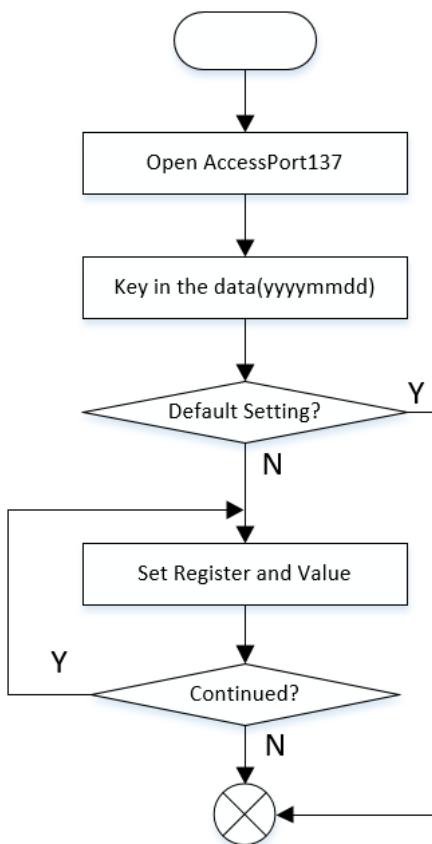


圖 3-34 The flow chart of sensor parameters setting

### 3.3.4. 感測器初始化設置

當感測器參數設定完成後，微控制器便會依照剛剛的設定來做感測器初始化設置，如果是預設設定，便會依照本系統原先的設定來做設置。如圖 3-35 所見，I<sup>2</sup>C1 - I<sup>2</sup>C3 使用相同的函式:HAL\_I2C\_Mem\_Write\_DMA()，來將資料寫入暫存器中，這邊要注意的是，如果讀者沒有開啟 DMA 的功能，就無法使用該函式，而要使用 HAL\_I2C\_Mem\_Write()。

```
572 void FirstI2C_Init(void)
573 {
574     for (Num=1; Num<Q; ++Num)
575     {
576         //-----Sensor A part-----
577         HAL_I2C_Mem_Write_DMA(&hi2c1, (uint16_t)SensorA << 1, Register[Num], 1, &Value[Num], 1);
578         HAL_Delay(2);
579         //-----Sensor B part-----
580         HAL_I2C_Mem_Write_DMA(&hi2c1, (uint16_t)SensorB << 1, Register[Num], 1, &Value[Num], 1);
581         HAL_Delay(2);
582     }
583 }
```

圖 3-35 I<sup>2</sup>C sensor parameters setting

因為本系統中有使用到第四組 I<sup>2</sup>C，根據前文所述，第四組 I<sup>2</sup>C 需要開通 STM32F412 的 FMPI2C 這個功能，所以第四組感測器就必須透過 HAL\_FMPI2C\_Mem\_Write\_DMA()這一函式將資料寫入暫存器內，與一般的 I<sup>2</sup>C 一樣，如果沒有開啟 DMA 的功能，就必須使用 HAL\_FMPI2C\_Mem\_Write()。

```
608 void ForthI2C_Init(void)
609 {
610     for (Num=1; Num<Q; ++Num)
611     {
612         //-----Sensor A part-----
613         HAL_FMPI2C_Mem_Write_DMA(&hfmpi2c1, (uint16_t)SensorA << 1, Register[Num], 1, &Value[Num], 1);
614         HAL_Delay(2);
615         //-----Sensor B part-----
616         HAL_FMPI2C_Mem_Write_DMA(&hfmpi2c1, (uint16_t)SensorB << 1, Register[Num], 1, &Value[Num], 1);
617         HAL_Delay(2);
618     }
619 }
```

圖 3-36 FMPI2C sensor parameters setting

### 3.3.5. 動作捕捉

動作捕捉是本模組的核心，將透過程式碼來做說明。圖 3-37 式動作捕捉的程式碼，是一個 do-while 的迴圈，跳出迴圈的條件是使用者按下 user bottom，使得 Flag 變成 1，代表使用者不想再繼續做動作捕捉，因此停止了動作捕捉的程序。而可以看到程是由三部分組成，分別是 Read\_Value()、Timestamp() 以及 DataFusion()。

```

199 do{
200     Read_Value();
201     Timestamp();
202     DataFusion();
203 }while(Flag==1);

```

圖 3-37 Motion capture code

### 3.3.5.1. Read\_Value()

這一副程式是用來讀取感測器所捕捉到的值，圖 3-38 是此副程式的程式碼，此副程式透過 HAL\_I2C\_Mem\_Read\_DMA() 這一函式來讀取感測器上的暫存器，並使用 DataConvert() 將資料丟到正確的資料結構位置中，見圖 3-39。

```

577     uint8_t buf1[12]={0},buf2[12]={0},buf3[12]={0},buf4[12]={0},buf5[12]={0},buf6[12]={0},buf7[12]={0},buf8[12]={0};
578     void Read_Value(void)
579     {
580         HAL_I2C_Mem_Read_DMA(&hi2c1,(uint16_t)SensorA << 1, (uint16_t) OUTX_L_G, 1, buf1,12);
581         HAL_I2C_Mem_Read_DMA(&hi2c2,(uint16_t)SensorA << 1, (uint16_t) OUTX_L_G, 1, buf2,12);
582         HAL_I2C_Mem_Read_DMA(&hi2c3,(uint16_t)SensorA << 1, (uint16_t) OUTX_L_G, 1, buf3,12);
583         HAL_FMPI2C_Mem_Read_DMA(&hfmpi2c1,(uint16_t)SensorA << 1, (uint16_t) OUTX_L_G, 1, buf4,12);
584         DataConvertA();
585         HAL_Delay(1);
586         HAL_I2C_Mem_Read_DMA(&hi2c1,(uint16_t)SensorB << 1, (uint16_t) OUTX_L_G, 1, buf5,12);
587         HAL_I2C_Mem_Read_DMA(&hi2c2,(uint16_t)SensorB << 1, (uint16_t) OUTX_L_G, 1, buf6,12);
588         HAL_I2C_Mem_Read_DMA(&hi2c3,(uint16_t)SensorB << 1, (uint16_t) OUTX_L_G, 1, buf7,12);
589         HAL_FMPI2C_Mem_Read_DMA(&hfmpi2c1,(uint16_t)SensorB << 1, (uint16_t) OUTX_L_G, 1, buf8,12);
590         DataConvertB();
591         HAL_Delay(1);
592     }

```

圖 3-38 Read\_Value()

注意到此函副程式中有兩個 HAL\_Delay()，這兩個 HAL\_Delay() 是用來等待 i2c->state 復位。在實作的過程中發現，如果欲使用相同的 I2C 匯流排來讀取慣性感測器的值，則該 I2C 匯流排程式中的 i2c->state 需要一段時間將此狀態從 HAL\_I2C\_STATE\_BUSY 復位成 HAL\_I2C\_STATE\_READY，否則會讀取過程會發生錯誤，導致讀不到慣性感測器的值，而程式中最小的標準延遲程式就是 HAL\_Delay()，HAL\_Delay() 的單位是 1ms，只能輸入整數，因此 HAL\_Delay(1) 代表延遲 1 毫秒。

目前的架構中，只能實現 8 顆慣性感測器，其原因是現有的 STM32 開發版中，最多只能支援 4 組 I2C，而目前 STMicroelectronics 所能提供的慣性感測器，其 I2C 的位址只有一個位元的差異，也就是說同一條 I2C 匯流排只能串接兩顆相同的慣性感測器。但未來的實際應用上，使用的慣性感測器數目絕對會大於現有架構的數目(8 顆)，假如未來新增慣性感測器的數目時，則必須修改此副程式的內容，來增加讀取的數目，修改方法如圖所示，可以看到我們假設新增了新的慣性感測器-SensorC，我們將讀取 SensorC 的程式新增在原本讀取 SensorB 的後面，一樣使用一個 HAL\_Delay(1) 來做延遲，這樣便可以讀取 12 顆慣性感測器的值。

```

595 void DataConvertA(void)
596 {
597     Packet.FirstI2CsensorA.Acc_X.LSB=buf1[6];
598     Packet.FirstI2CsensorA.Acc_X.MSB=buf1[7];
599     Packet.FirstI2CsensorA.Acc_Y.LSB=buf1[8];
600     Packet.FirstI2CsensorA.Acc_Y.MSB=buf1[9];
601     Packet.FirstI2CsensorA.Acc_Z.LSB=buf1[10];
602     Packet.FirstI2CsensorA.Acc_Z.MSB=buf1[11];
603
604     Packet.FirstI2CsensorA.Gyro_X.LSB=buf1[0];
605     Packet.FirstI2CsensorA.Gyro_X.MSB=buf1[1];
606     Packet.FirstI2CsensorA.Gyro_Y.LSB=buf1[2];
607     Packet.FirstI2CsensorA.Gyro_Y.MSB=buf1[3];
608     Packet.FirstI2CsensorA.Gyro_Z.LSB=buf1[4];
609     Packet.FirstI2CsensorA.Gyro_Z.MSB=buf1[5];
610
611     Packet.SecondI2CsensorA.Acc_X.LSB=buf2[6];
612     Packet.SecondI2CsensorA.Acc_X.MSB=buf2[7];
613     Packet.SecondI2CsensorA.Acc_Y.LSB=buf2[8];
614     Packet.SecondI2CsensorA.Acc_Y.MSB=buf2[9];
615     Packet.SecondI2CsensorA.Acc_Z.LSB=buf2[10];
616     Packet.SecondI2CsensorA.Acc_Z.MSB=buf2[11];
617

```

圖 3-39 DataConvertA() (partial)

### 3.3.5.2. Timestamp()

這一副程式是用來讀取當下的時間戳記，前文提到本系統使用 TIM2 的功能，每一毫秒記錄一次時間，而資料在傳送到動作顯示模組之前，必須加上時間戳記來紀錄時間及順序，此副程式是將變數 timestamp 餵入對應的資料結構位置中。

```

738 void Timestamp(void)
739 {
740     Packet.data.stamp1=timestamp ;
741     Packet.data.stamp2=timestamp >> 8;
742     Packet.data.stamp3=timestamp >> 16;
743     Packet.data.stamp4=timestamp >> 24;
744 }

```

圖 3-40 Timestamp()

### 3.3.5.3. DataFusion()

此副程式的功能是將整合後的資料傳送給動作顯示模組，因此只有一行傳送的程式碼。

```

745 void DataFusion(void)
746 {
747     CDC_Transmit_FS((uint8_t*)&Packet,sizeof(Packet));
748 }

```

圖 3-41 DataFusion()

### 3.3.6. 其他實現

#### 3.3.6.1. 資料封包結構

為了儲存捕捉到的資料，本系統建立了一個資料結構，如圖，此資料結構是依照 3.3.2 中提及的融合後資料封包規範來實現。

#### 3.3.6.2. CDC\_Transmit\_FSO

## 4. 效能表現

本系統所實現的是資料獲取模組，一個收發模組要考慮的點有三項，1.收發正確性、2.收發速度、3.收發能量耗損。因此在本章節中將會依照這三項項目來個別測試並介紹其效能表現。

### 4.1. 收發正確性

在收發正確性的測試項目中，使用 Jmex 所提供的動作顯示軟體來捕捉慣性感測器的原始資料，按下 Store Raw Data，軟體會儲存 1000 筆資料，可以用來分析正確性。



圖 4-1 AiQ Software

儲存的檔案是 txt 檔，內容如圖 4-2，是 10 進制的表示法。

圖 4-2 Stored file

並且使用治具夾住慣性感測器，確保感測器不會晃動，治具如圖 4-3。

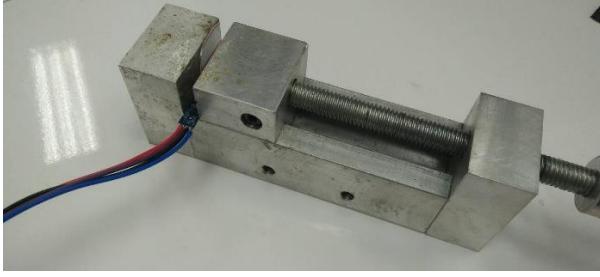


圖 4-3 Fixture

#### 4.1.1. 加速度計驗證

首先要將原始資料轉成可讀的資料，需要經過一些計算，每個軸向的資料都含有 LSB 及 MSB，要先將這兩者合併起來。

$$\text{Raw} = (\text{MSB} \ll 8 \mid \text{LSB})$$

這個值的範圍會介於-32768~32767之間，我們需要將這個質做標準化，使其落在-1~+1之間，因此要做一個判斷式，其中的 $\times 8$ 是因為本系統加速度計的靈敏度是 $\pm 8g$ 。

$$\text{if } \begin{cases} Raw < 32768, & \widehat{Raw} = \frac{Raw \times 8}{32768} \\ Raw \geq 32768, & \widehat{Raw} = \frac{(Raw - 65536) \times 8}{32768} \end{cases}$$

如此一來便可以得到正確的加速度值，再使用空間向量公式來求角度，可以得到與直角坐標系的角度差，測試結果如表 4。

$$\cos \theta = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|} = \frac{a_x b_x + a_y b_y + a_z b_z}{\sqrt{a_x^2 + a_y^2 + a_z^2} \sqrt{b_x^2 + b_y^2 + b_z^2}}$$

因為加速度計會受地心引力的影響，因此會有些許誤差，這誤差可以透過演算法做修正。

表 4 Accelerometer test results

+X	Standard	Test Avg.	Error	-X	Standard	Test Avg.	Error
X	1	0.989893	1.0107%	X	-1	-1.000413	0.0413%
Y	0	-0.002304	0.2304%	Y	0	0.006761	0.6761%

Z	0	0.009418	0.09418%	Z	0	0.012251	1.2251%
Degree	0°	0.56116°	0.56116°	Degree	180°	179.19862°	0.80138°
+Y	Standard	Test Avg.	Error	-Y	Standard	Test Avg.	Error
X	0	-0.006483	0.6483%	X	0	0.005395	0.5395%
Y	1	1.003032	0.3032%	Y	-1	-0.997247	0.2753%
Z	0	0.007309	0.7309%	Z	0	-0.001793	0.1793%
Degree	0°	0.55804°	0.55804°	Degree	180°	179.67338°	0.32662°
+Z	Standard	Test Avg.	Error	-Z	Standard	Test Avg.	Error
X	0	0.02184	2.184%	X	0	-0.00667	0.667%
Y	0	-0.005785	0.5785%	Y	0	0.007973	0.7973%
Z	1	1.019753	1.9753%	Z	-1	-0.987666	1.2334%
Degree	0°	1.26922°	1.26922°	Degree	180°	179.67338°	0.32662°

### 4.1.2. 陀螺儀驗證

接著要驗證陀螺儀的正確性，本模組一樣使用治具來驗證陀螺儀的正確性。

測試方法：我們使用治具夾住慣性感測器，並旋轉治具，如圖 4-5，此操作沿著 Y 軸旋轉 +90 度，因此我們可以捕捉到旋轉的度數。對於捕捉到的原始資料，一樣要做標準化，使其落在 -360~+360 之間，因此要做一個判斷式，其中的 × 2000 是因為本系統陀螺儀的靈敏度是 ±2000dps。

$$\text{if } \begin{cases} Raw < 32768, \quad \widehat{Raw} = \frac{Raw \times 2000}{32768} \\ Raw \geq 32768, \quad \widehat{Raw} = \frac{(Raw - 65536) \times 2000}{32768} \end{cases}$$

測試結果如表 5，會發現陀螺儀的誤差值較大，是因為此感測器是慣性感測器，在做旋轉時，當感測器停止，仍然有慣性作用力存在，仍會對慣性感測器施力，造成角度量測的誤差。

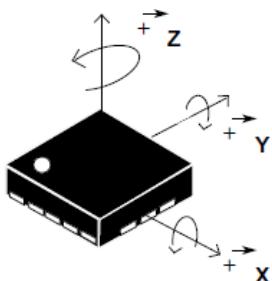


圖 4-4 Coordinate system

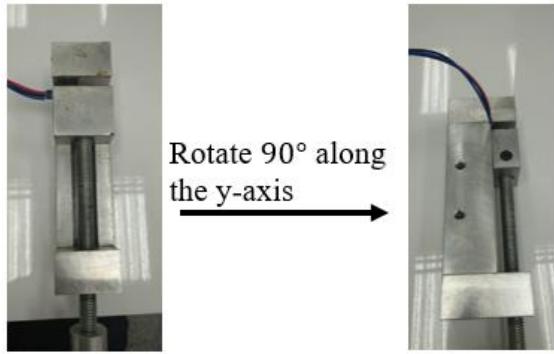


圖 4-5 Fixture operation

表 5 Gyroscope test result

Direction	Standard	Test	Error
Along x-axis, 90°	90°	104.065°	15.628%
Along x-axis, -90°	-90°	-110.046°	22.273%
Along y-axis, 90°	90°	105.201°	16.89%
Along y-axis, -90°	-90°	-98.633°	9.592%
Along z-axis, 90°	90°	113.831°	26.479%
Along z-axis, -90°	-90°	-101.929°	13.254%

## 4.2. 收發速度

在收發速度的測試項目中，使用 AccessPort、Device Monitoring Studio 以及示波器來做測試。測試順序為：使用 AccessPort 獲取模組所傳輸的封包，並觀察其 timestamp 的變化，計算出時間差，使用 Device Monitoring Studio 來監視封包傳輸時間差是否正確，最後使用示波器觀察 USB 的輸出，驗證時間是否相符。

### 4.2.1. AccessPort

首先，先使用 AccessPort 接收一段時間的封包，並分析封包內容中的 timestamp，發現結果如表 3，每一筆資料的 timestamp 的間隔都是 2，也就代表說本系統的獲取速率是 2 毫秒。

表 6 timestamp analysis

timestamp	timestamp gap (ms)
C6 25 00 00	2
C8 25 00 00	2
CA 25 00 00	2
CC 25 00 00	2
CE 25 00 00	2
D0 25 00 00	2

D2 25 00 00	2
D4 25 00 00	2
D6 25 00 00	2
D8 25 00 00	2

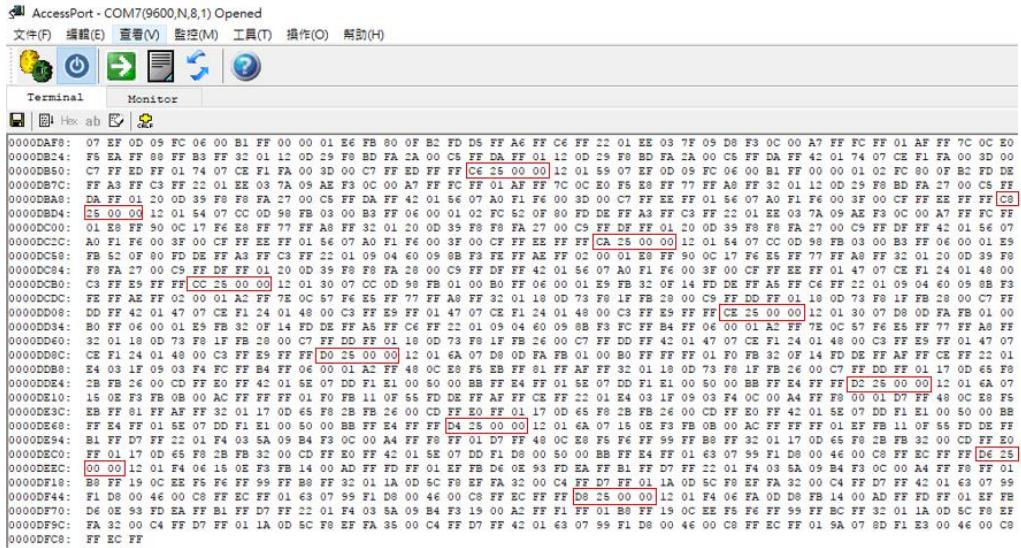


圖 4-6 AccessPort result

#### **4.2.2. Device Monitoring Studio**

再使用 Device Monitoring Studio 來驗證資料的接收時間，先打開 Device Monitoring Studio，選擇想要監控的裝置，本系統用來傳輸的裝置是 COM7，因此對 COM7 按右鍵，選擇 Start Monitoring。

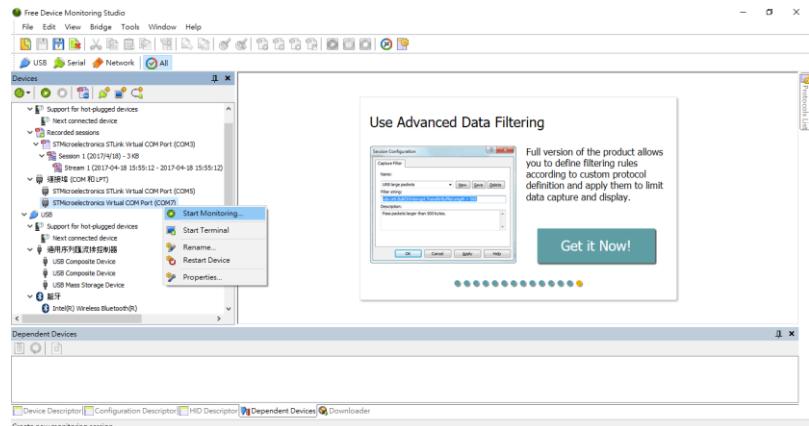


圖 4-7 Choosing Device

接著會出現 Session Configuration 的頁面，選擇 Generic 即可。

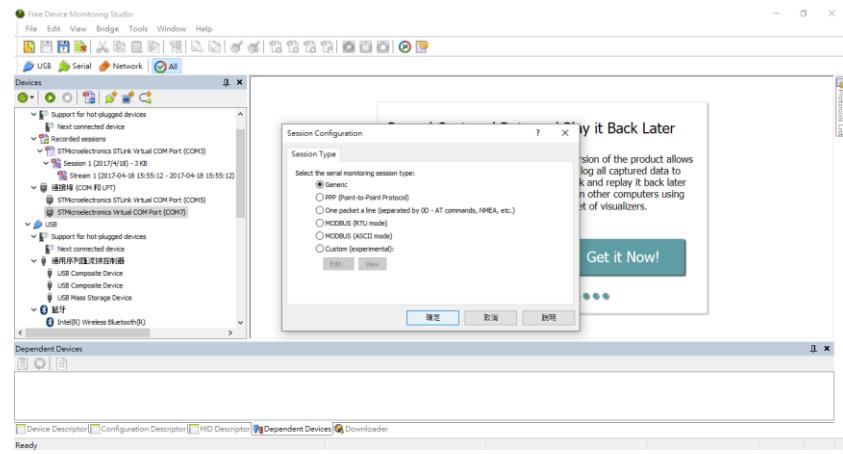


圖 4-8 Session Configuration

我們要監控的是每一筆封包的時間差，所以選擇 Packet View。

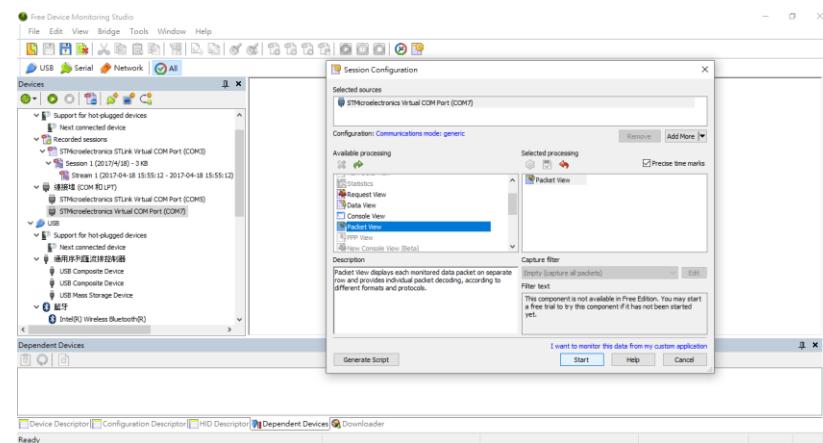


圖 4-9 Select Packet View

按下 Start 之後就會監控畫面，觀察之間的時間差可以發現，每筆資料間隔時間為 2 毫秒左右，與 AccessPort 的結果相符。

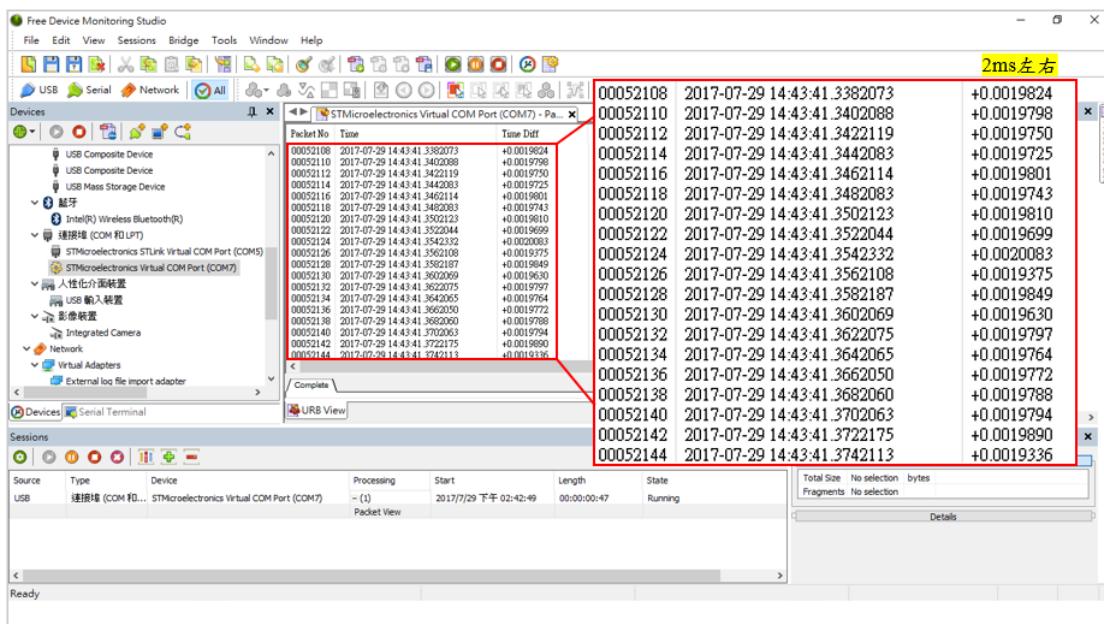


圖 4-10 Device Monitoring Studio result

### 4.2.3. USB 輸出

接著再使用示波器觀察 USB 的輸出，從硬體的方面作驗證。USB 的傳輸線由四條線組成，紅色為電源線，黑色為接地線，白色為訊號負線，綠色為訊號正線。我們用示波器來觀察，其中，黃色是訊號正線，綠色是訊號負線。經過觀察可以發現，封包之間的傳輸間隔為 1.965916 毫秒，與 AccessPort 及 Device Monitoring Studio 結果相符。

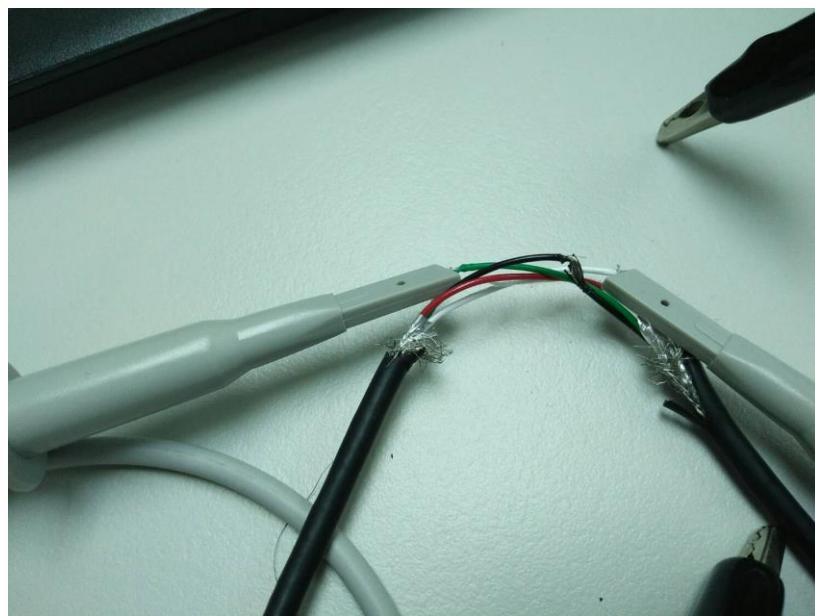


圖 4-11 USB Cable

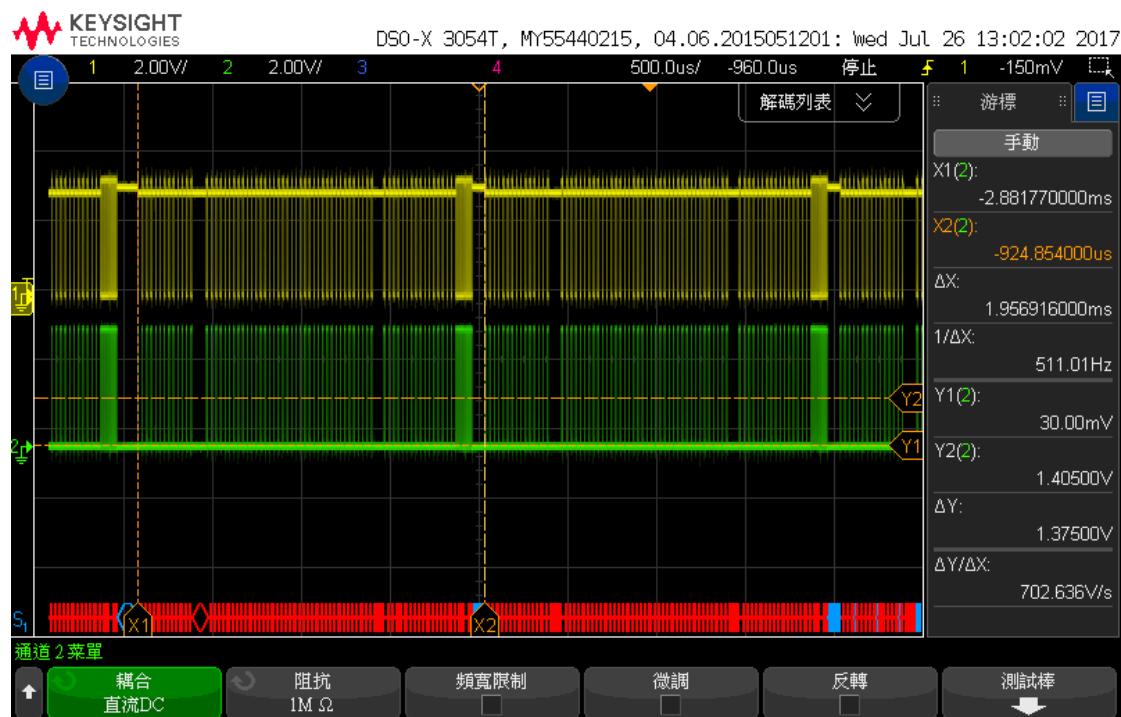


圖 4-12 Oscilloscope result

## 4.3. 收發能量損耗

在收發能量損耗的測試項目中，我們使用三用電表以及電源供應器來測試，測試配置如圖所示，本測試要量測的是電流損耗，因此要串聯。電源供應器的正極與三用電表的正極相連，三用電表的負極當作是開發板的正極，提供電壓，開發板的負極在與電源供應器的負極相連，形成迴路。開發板運作時會透過 USB 傳輸線傳送資料給筆電，藉由筆電上的動作顯示軟體檢測開發板是否正常運作。將電源供應器的輸出電壓調整為 3.3V，並將三用電表的測試模式調整為電流，單位為 mA，設定完成後便可以開始測試。

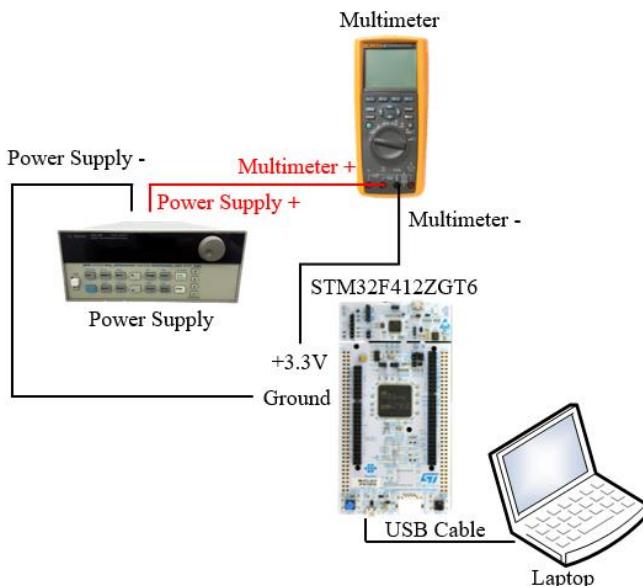


圖 4-13 Hardware Testing Component

本資料獲取模組使用 DMA 的功能來提高取樣速率，閱讀 Using STM32F4 MCU power modes with best dynamic efficiency[4] 中的 1.2 可知，若使用 DMA 的功能來獲取資料，在獲取資料的期間內，使 MCU 進入 sleep mode 可以達到降低功耗的效果，但本實作中並沒有讓 MCU 進入 sleep mode，而是使 MCU 持續工作來實現即時傳輸資料。因此本節將探討 DMA 與 MCU 功耗表現之關係。

### 4.3.1. HAL\_Delay()對功耗的影響

本節將討論 HAL\_Delay()與系統功耗的關係，本測試將修改 HAL\_Delay()的數值來調整 MCU 的工作時間，觀察不同工作時間整體功耗的表現如何。

由 3.2.4 可知，整個動作捕捉的程式碼由 HAL\_I2C\_Mem\_Read\_DMA()、HAL\_Delay()、DataConvertA()、DataConvertB()、Timestamp()及 CDC\_Transmit\_FS()組成，其中 HAL\_I2C\_Mem\_Read\_DMA()與 CPU 無關；而 DataConvertA()、DataConvertB()以及 Timestamp()都只是簡單的數值交換；CDC\_Transmit\_FS()是 STM32CubeMX 產生的副程式，本系統直接拿來使用；因此真正可以修改並增加 CPU 工作時間的是 HAL\_Delay()。

HAL\_Delay() 的程式碼如圖 4-14，原則上就是重複檢測當前時間:HAL\_GetTick()與舊時間:tickstart 的差異，符合輸入的 delay 數值即跳出。

```

338     __weak void HAL_Delay(__IO uint32_t Delay)
339     {
340         uint32_t tickstart = 0U;
341         tickstart = HAL_GetTick();
342         while((HAL_GetTick() - tickstart) < Delay)
343     {
344     }
345 }
```

圖 4-14 HAL\_Delay()

於是本節便修改 HAL\_Delay()的數值，並測試功耗，結果如表 6。

表 7 HAL\_Delay() power consumption (with DMA)

DMA				
HAL_Delay()	Max (mA)	Avg (mA)	Min (mA)	Delay (ms)
1	94.69	90.55	86.48	2
2	93.95	89.98	85.97	4
3	93.89	89.91	85.81	6
4	94.15	90.11	86.01	8
5	94.03	89.96	85.83	10
10	94.06	90.00	85.61	20
50	93.93	89.84	85.50	100

接著使用非 DMA 的功能重複上述實驗，測試結果如表 7。

表 8 HAL\_Delay() power consumption (without DMA)

非 DMA				
HAL_Delay()	Max (mA)	Avg (mA)	Min (mA)	Delay (ms)
1	85.43	81.93	77.37	4
2	87.94	83.89	79.75	6
3	88.88	84.96	80.96	8
4	89.83	85.85	81.79	10
5	90.10	86.14	81.80	12
10	91.26	87.28	83.31	22
50	92.38	88.38	84.41	102

經過觀察可以發現，使用 DMA 的情況下，調整 HAL\_Delay()的數值並不會影響到功耗，但在使用非 DMA 的情況下，若 HAL\_Delay()的數值越高，消耗的功率就會越大，代表增加 CPU 的工作時間是會影響功耗的，但為了測試是否使用 DMA 後，不論 CPU 的工作時間多長，都不會影響到功耗，因此將 HAL\_Delay()改成 FOR 迴圈，請見 4.3.2。

### 4.3.2. FOR 迴圈對功耗的影響

本模組之實作使用 HAL\_Delay()來使 i2c->state 復位，但觀察 HAL\_Delay()對功耗的影響並非直覺，所以使用另外一種延遲的方法，使用一個 FOR 迴圈重複執行加法與減法，來取代 HAL\_Delay()拖延的時間。並修改執行的次數，程式碼如圖 4-15。

```

592     for(int i=0;i<10000;i++)
593     {
594         a++;
595     }
607     for(int i=0;i<10000;i++)
608     {
609         a--;
610     }

```

圖 4-15 FOR loop code

表 9 FOR loop power consumption (with DMA)

DMA				
i	Max (mA)	Avg (mA)	Min (mA)	Delay (ms)
10000	90.76	86.79	82.30	2
20000	89.47	85.58	81.11	3
30000	89.19	85.27	80.75	4
40000	88.53	84.64	80.27	5

表 10 FOR loop power consumption (without DMA)

非 DMA				
i	Max (mA)	Avg (mA)	Min (mA)	Delay (ms)
10000	86.90	82.95	78.57	5
20000	88.16	84.17	79.86	6
30000	88.72	84.87	80.93	7
40000	89.19	85.34	81.38	8

經過觀察可以發現，在使用 DMA 的情況下，當 i 值越高，代表 CPU 需要花更多時間執行 FOR 迴圈，使得 CPU 工作的時間越長，系統的耗能越低，相反的，使用非 DMA 的情況下，CPU 工作時間越長，系統的耗能越高。

因此可以推論，本系統的實作中，使用 DMA 是沒有降低功耗的，推論原因是本系統使用 DMA 來獲取小資料(12 bytes)，因為過於頻繁(2ms)使得 CPU 沒有時間進入 sleep mode 來節省能源，但也因此得到了傳輸量的表現。

以 HAL\_Delay()的例子來說，在同樣是 HAL\_Delay(1)的條件下，DMA 每筆資料之間的延遲是 2ms，非 DMA 每筆資料之間的延遲是 4ms，其中每筆資料都有 8 顆慣性感測器的資料，因此可以說 DMA 的資料速度是 0.25ms/顆，非 DMA 的資料速度是 0.5ms/顆，如果系統可以容忍的延遲是 5ms，代表使用非 DMA 能收到  $\frac{5\text{ ms}}{0.5\text{ ms}}$ (顆/筆)慣性感測器的資料，也就是 10 顆慣性感測器的資料，而使用 DMA 將可以收到  $\frac{5\text{ ms}}{0.25\text{ ms}}$ (顆/筆)慣性感測器的資料，也就是 20 顆，這將會有助於

未來增加慣性感測器的應用。

# 5. GitHub 相關

## 5.1. GitHub 說明-前置作業

本系統的程式都會上傳到雲端平台 GitHub 上，供各位使用者下載開發，使用 GitHub 開發前，建議先創建一組帳號，如此一來，在開發上便可以更有系統，也可以與他人一同進行開發。

### 5.1.1. GitHub 帳號建立

首先，要建立帳號。各位使用者可以先連上 GitHub 的官網：<https://github.com/>



圖 5-1 GitHub webpage

如果已經有 GitHub 的帳號，可以直接點選”Sign In”登入。如果還沒有帳號，可以點選”Sign up for GitHub”創立帳號。

A close-up screenshot of the GitHub sign-up form. It shows the three input fields: 'Pick a username', 'Your email address', and 'Create a password'. Below the password field is a note: 'Use at least one letter, one numeral, and seven characters.' A large green 'Sign up for GitHub' button is at the bottom. At the very bottom of the form, there is a small note: 'By clicking "Sign up for GitHub", you agree to our [terms of service](#) and [privacy policy](#). We'll occasionally send you account related emails.'

圖 5-2 Create account information

依序填入帳號、email(已註冊的 email 不能使用)、密碼(至少要一個小寫的字

母，密碼最短不可少於 7 個字元)。接著按下”Sign up for GitHub”，會進入下面這個頁面，如圖，使用者可以依照自己的需求選擇是否需要付費，選擇完成後請按下”Continue”。

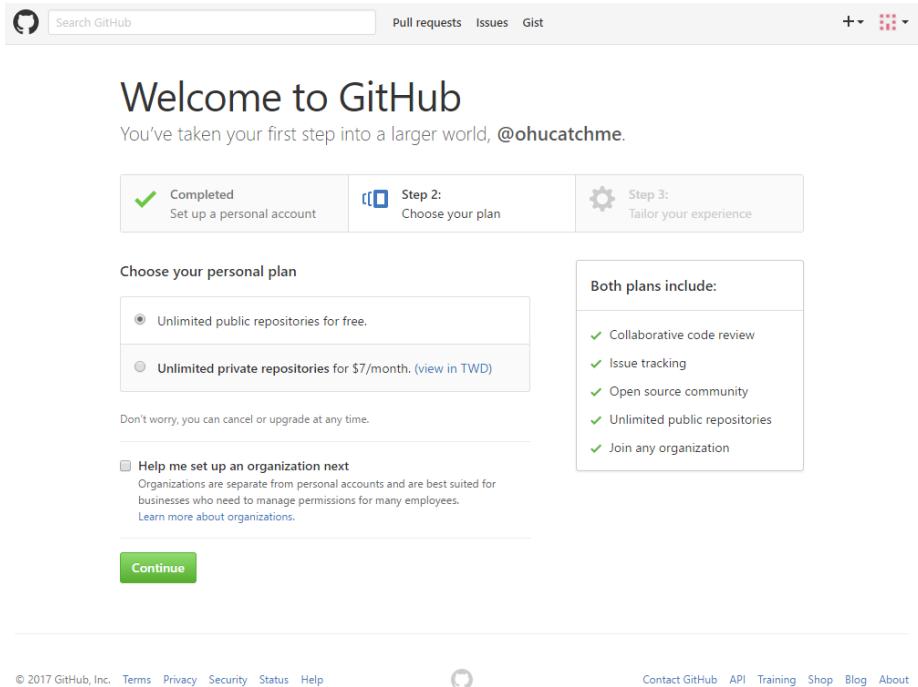


圖 5-3 Create account- step 2

接著會請使用者依照自己的狀況勾選，如下圖，填完後按”Submit”後即完成帳號創立，也可以按下”skip this step”跳過此步驟。

The screenshot shows the GitHub account creation process at Step 3: Tailor your experience. The layout is similar to Step 2, with a navigation bar at the top. The title 'Welcome to GitHub' and user handle '@ohucatchme.' are present. Three steps are shown: 'Completed Set up a personal account', 'Step 2: Choose your plan' (blue square icon), and 'Step 3: Tailor your experience' (gear icon). The 'How would you describe your level of programming experience?' section has three radio buttons: 'Very experienced' (selected), 'Somewhat experienced', and 'Totally new to programming'. The 'What do you plan to use GitHub for? (check all that apply)' section has six checkboxes: 'Design', 'Research', 'Project Management', 'Development', 'School projects', and 'Other (please specify)'. The 'Which is closest to how you would describe yourself?' section has three radio buttons: 'I'm a professional', 'I'm a student', and 'I'm a hobbyist' (selected). The 'What are you interested in?' section is a large input field with placeholder text 'e.g. tutorials, android, ruby, web-development, machine-learning, open-source'. At the bottom are 'Submit' and 'skip this step' buttons.

圖 5-4 Create account-step 3

創建完成後 GitHub 會直接登入，頁面如下。

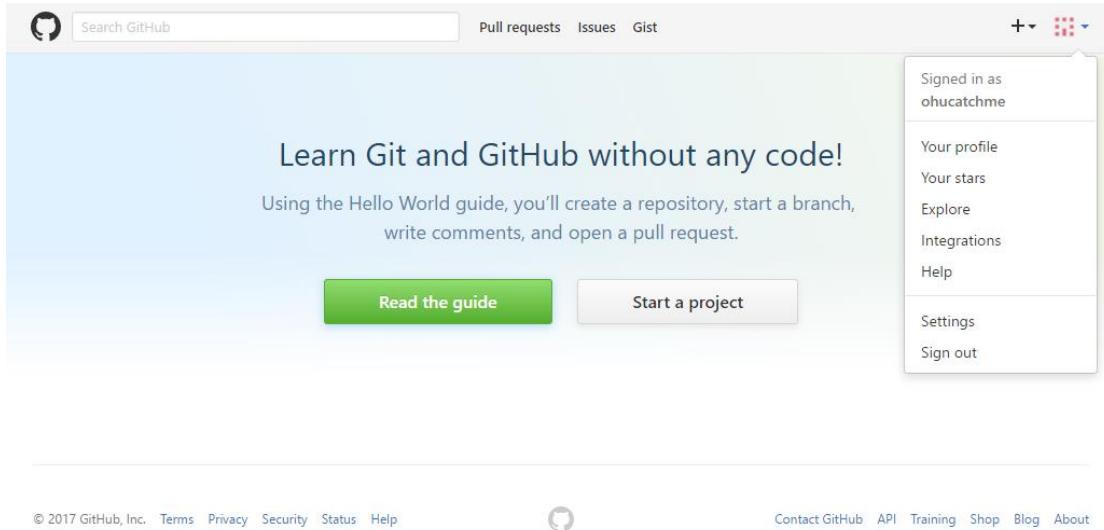


圖 5-5 Login webpage

接下來要進行 email 認證，點右上角的圖示，下拉選項會看到”Settings”，點選後會進入以下頁面。會看到 Emails 有一個警示符號，這時使用者需要進入當時輸入的信箱，點選連結進行認證。

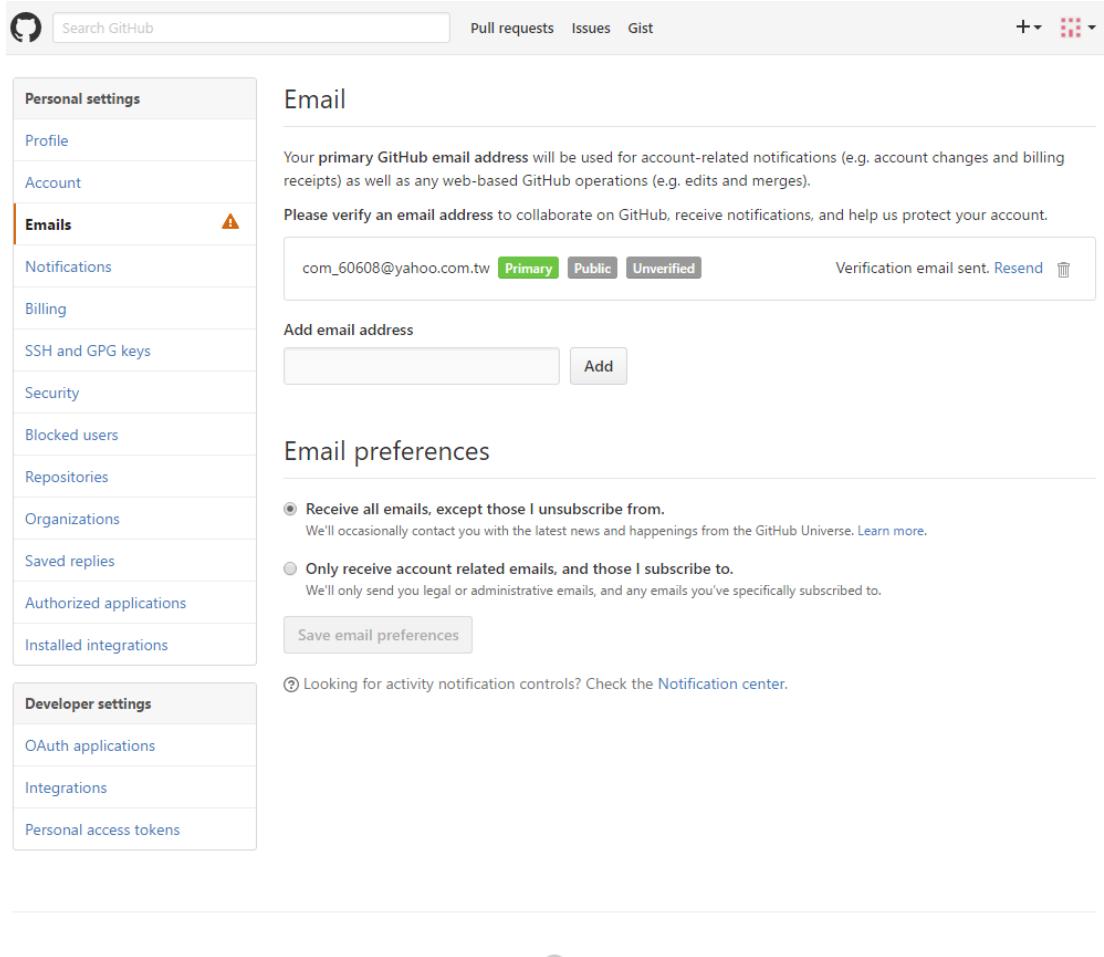


圖 5-6 Setting page

點選連結後會直接回到 GitHub 頁面，這時點開”Settings”可以發現 Emails 的

警示符號消失，代表認證成功。接著就可以開始開發自己的專案，或是上網下載別人的專案來使用。

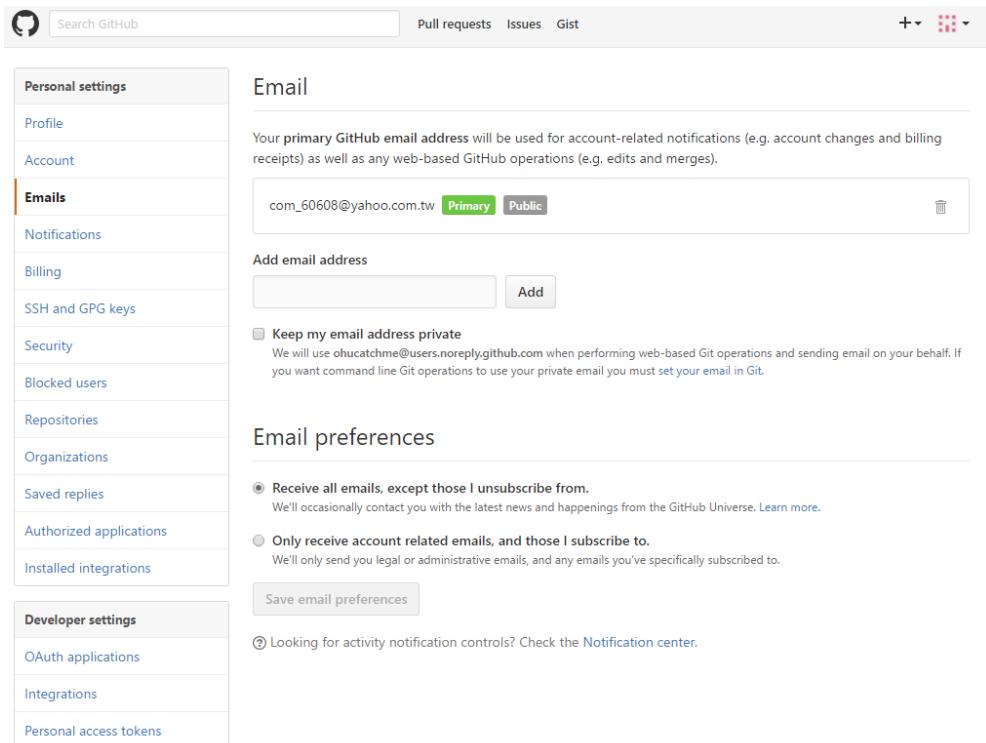


圖 5-7 Account certification success

## 5.1.2. GitHub 圖型化介面

新增帳號後，並不建議直接在網站上面開發專案，貼心的是 GitHub 有提供圖型化界面，可以透過這個網址下載: <https://desktop.github.com/>。點選”Download for Windows”開始下載。

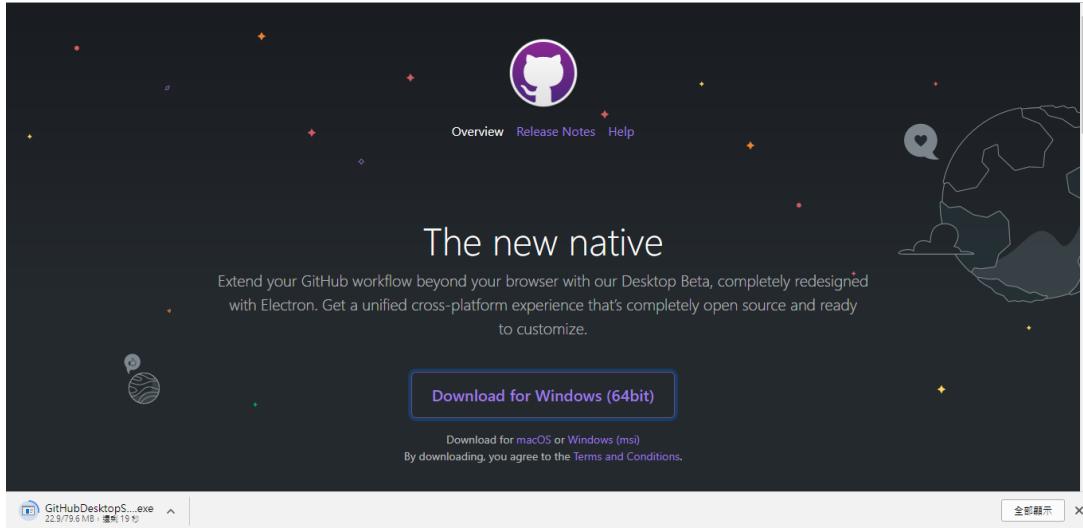


圖 5-8 GitHub Desktop download webpage

下載完成後，點開按下執行”開始安裝，會跑出下面的頁面。

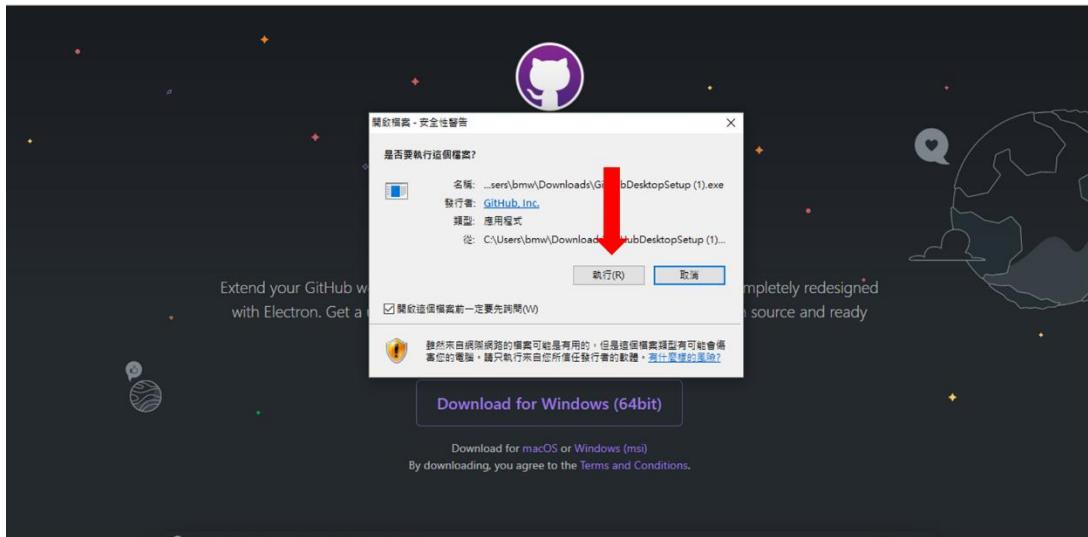


圖 5-9 Install page

安裝完成後，桌面會出現一個捷徑-GitHub Desktop。



圖 5-10 GitHub Desktop icon

圖型化界面的頁面如下，首先會需要登入，點選”Sign into GitHub.com”，並輸入 GitHub 帳號與密碼。

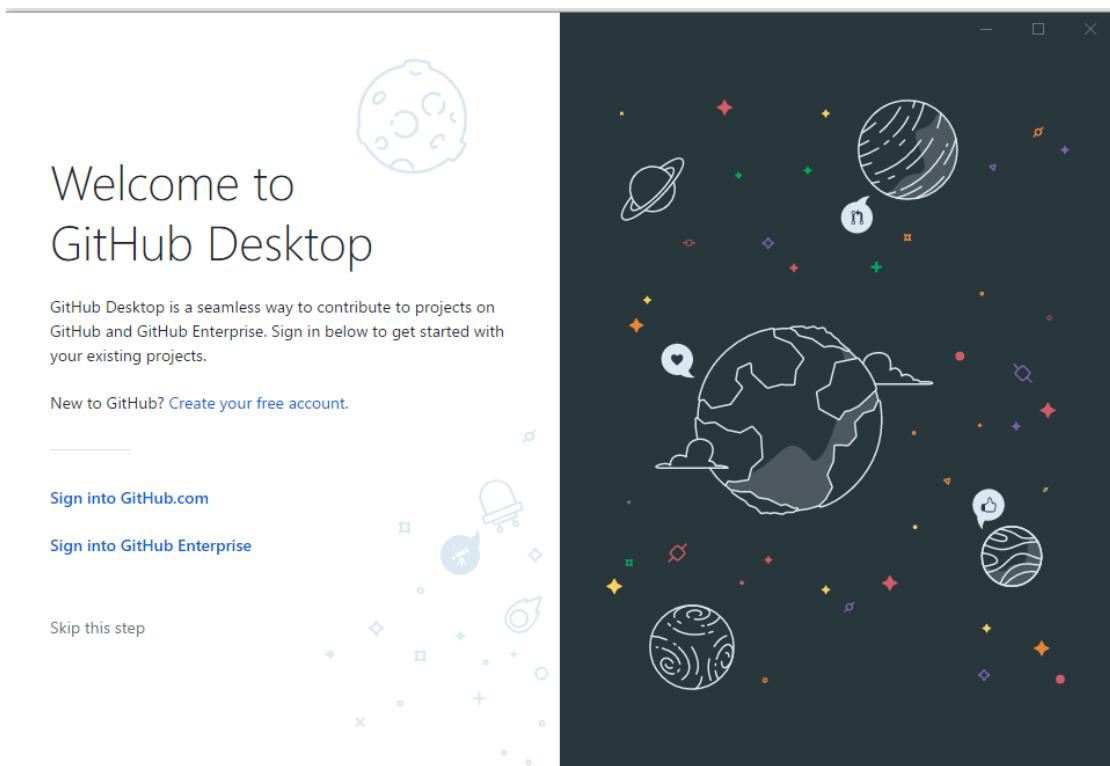


圖 5-11 Sign into GitHub.com

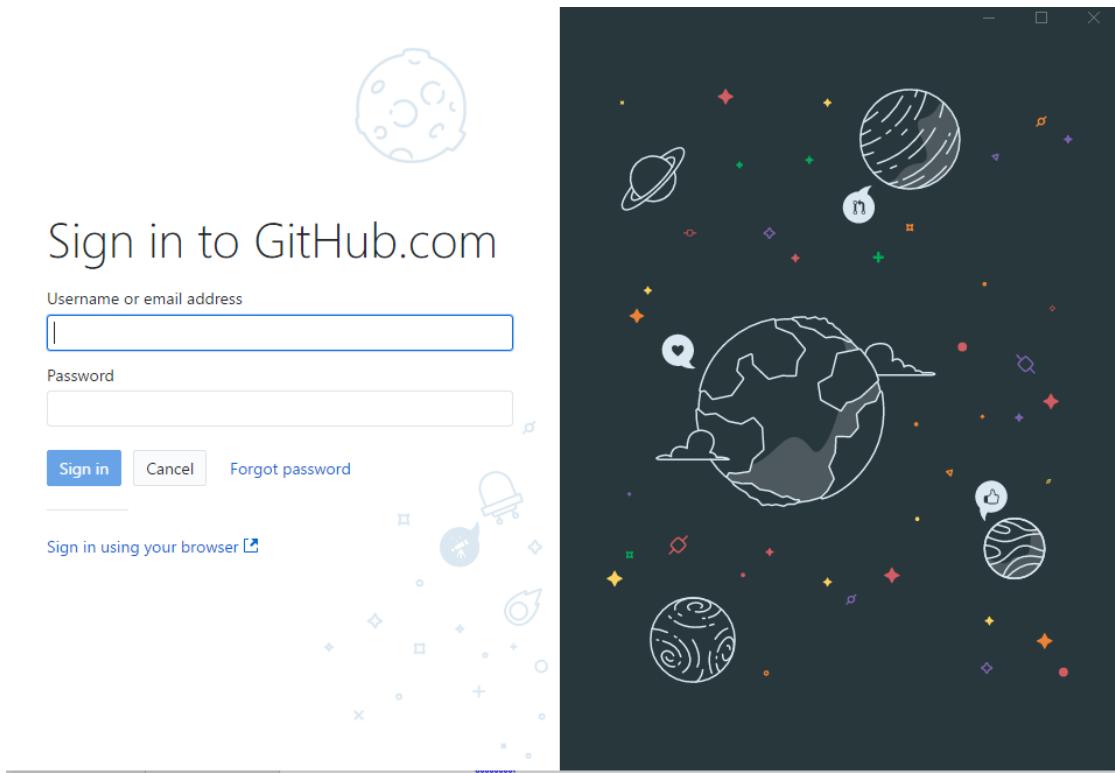


圖 5-12 Login page

登入後，會進入以下頁面，注意到左上角 repository，目前沒有任何專案，所以出現的文字是”Select a repository”，若要新增 repository 請參考 5.2。

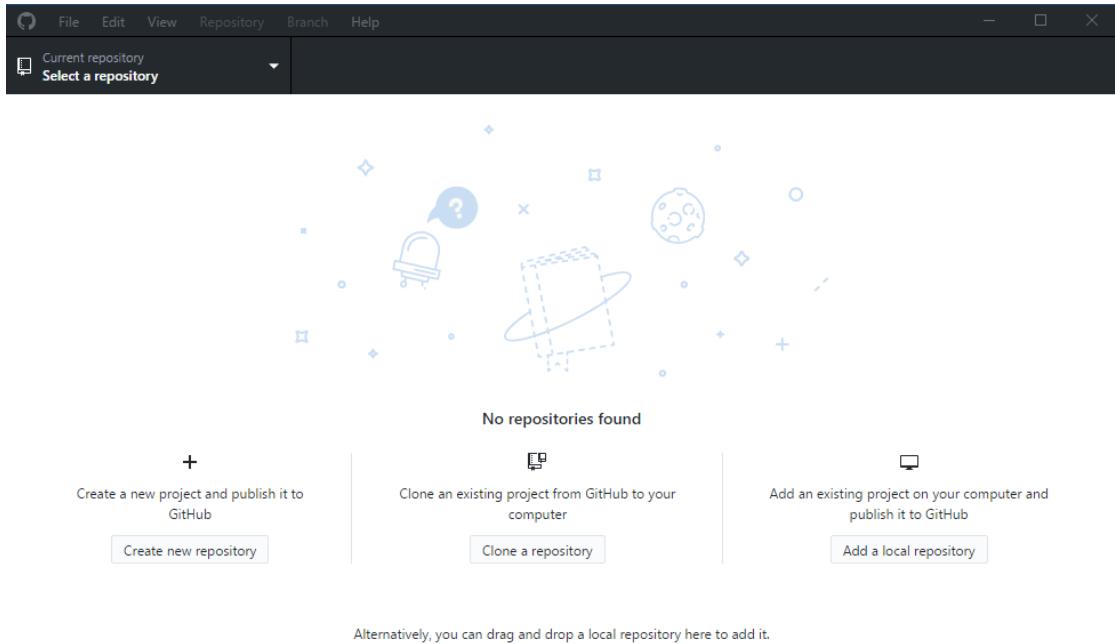


圖 5-13 GitHub Desktop page

gmail 帳號為:**aiqsmartclothing@gmail.com**

密碼為:54cjo4uzj6 (智慧衣服)

GitHub 的帳號為:aiqsmartclothing@gmail.com

密碼為:54cjo4uzj6 (智慧衣服)

## 5.2. GitHub 使用說明

### 5.2.1. 下載專案

回到 GitHub 的主畫面，找到 AiQ 的專案並下載，網址：  
<https://github.com/aiqsmartclothing/MultiMasterSensorHub>

下載專案並設定，進入上述網址，會進入以下的頁面，點開”Clone or download”的下拉選單，點選”Open in Desktop”，下載專案到主機。由於我們已經安裝了 GitHub Desktop 的程式，所以 GitHub 會自動呼叫它來執行後續動作，點選開啟「GitHubDesktop.exe」。

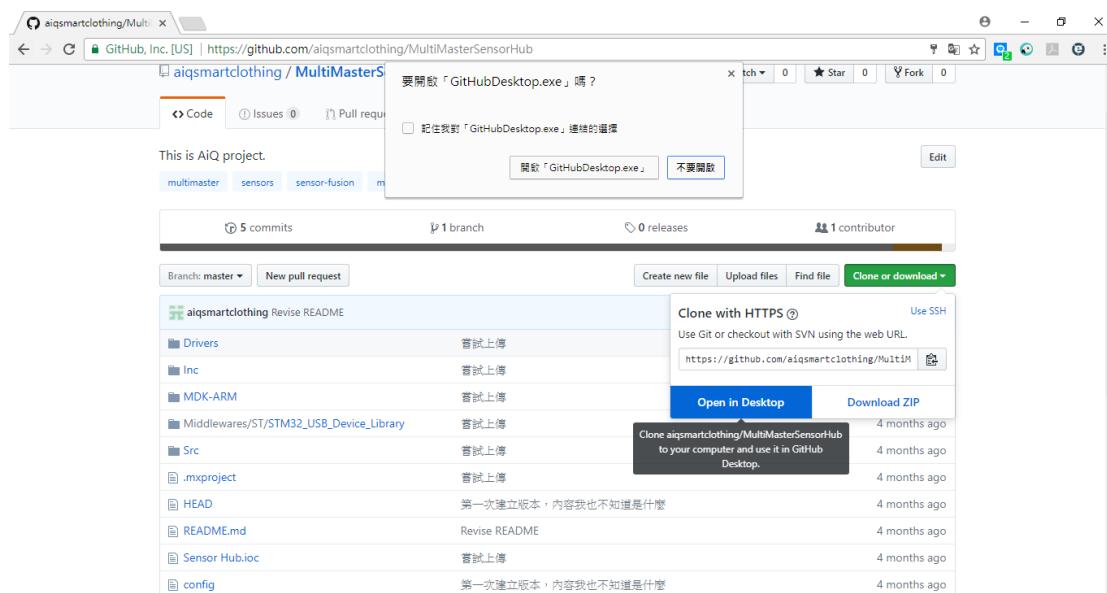


圖 5-14 AiQ Project page

接著會跑出以下畫面，按下”Clone”將 GitHub 上的專案複製到本地，儲存的位址在 C:\Users\bmw\Documents\GitHub

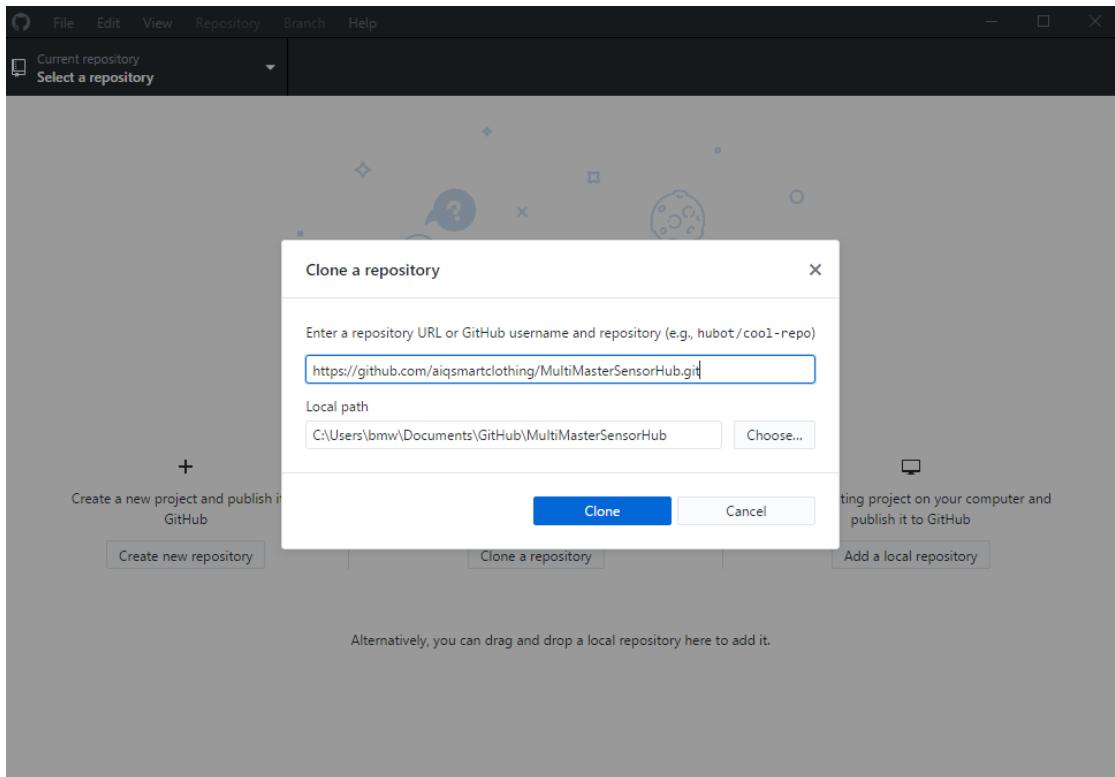


圖 5-15 Clone to local

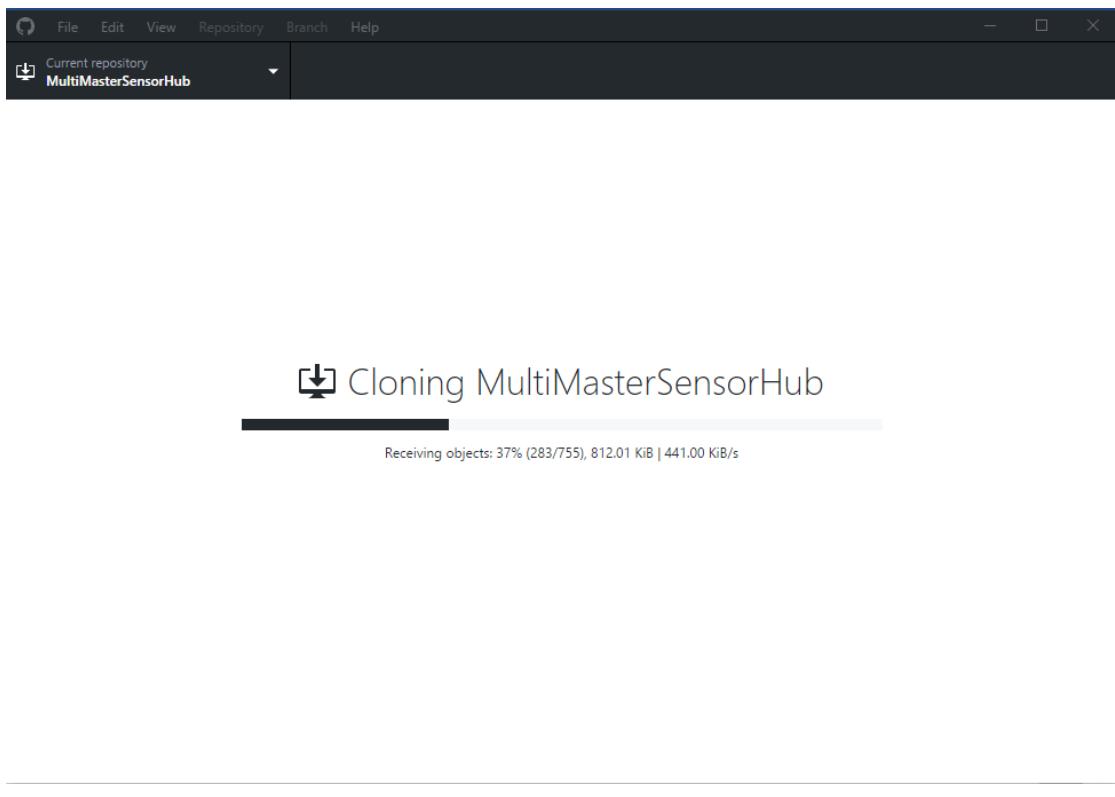


圖 5-16 Cloning

複製完成後會如圖 5-17，注意到左上角 repository 從變成 MultiMasterSensorHub，代表已經成功複製該專案到本地，這時便可以使用此專案。

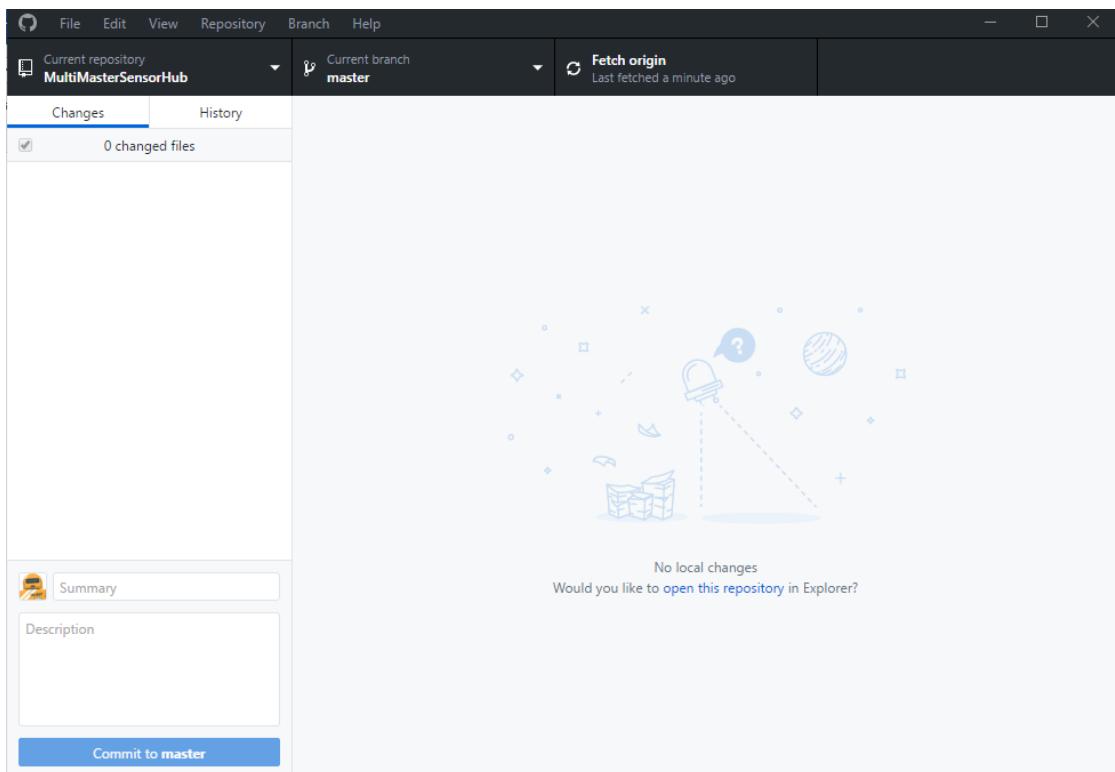


圖 5-17 Clone completely

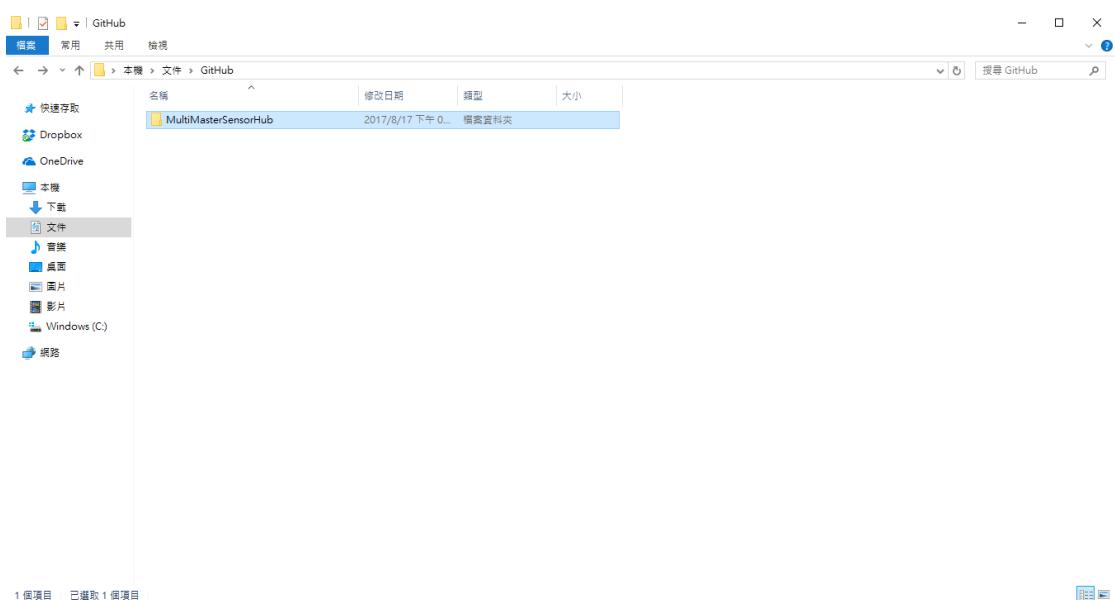


圖 5-18 Local file

## 5.2.2. 上傳新版本

當我們對專案做了修改後，可以上傳新專案到 GitHub，可以依照以下步驟來上傳。

首先，我們新增一個 txt 檔，檔名為 New。

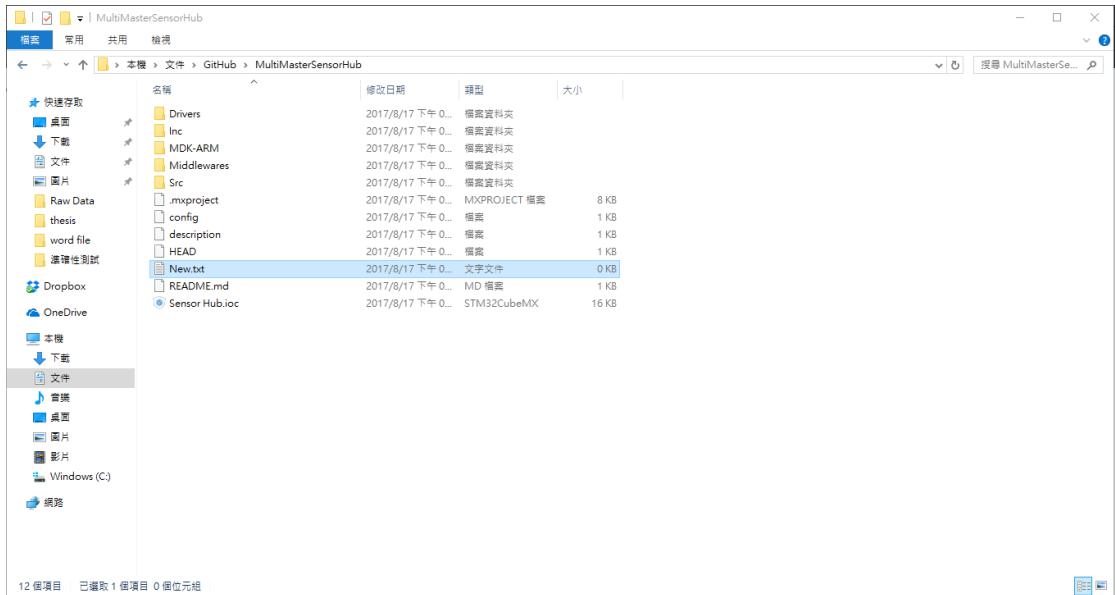


圖 5-19 Create a new text

打開 GitHub Desktop，會發現新增了一個檔案，我們可以在紅色方框內輸入這次新版本的標題與描述，並按”Commit to master”。

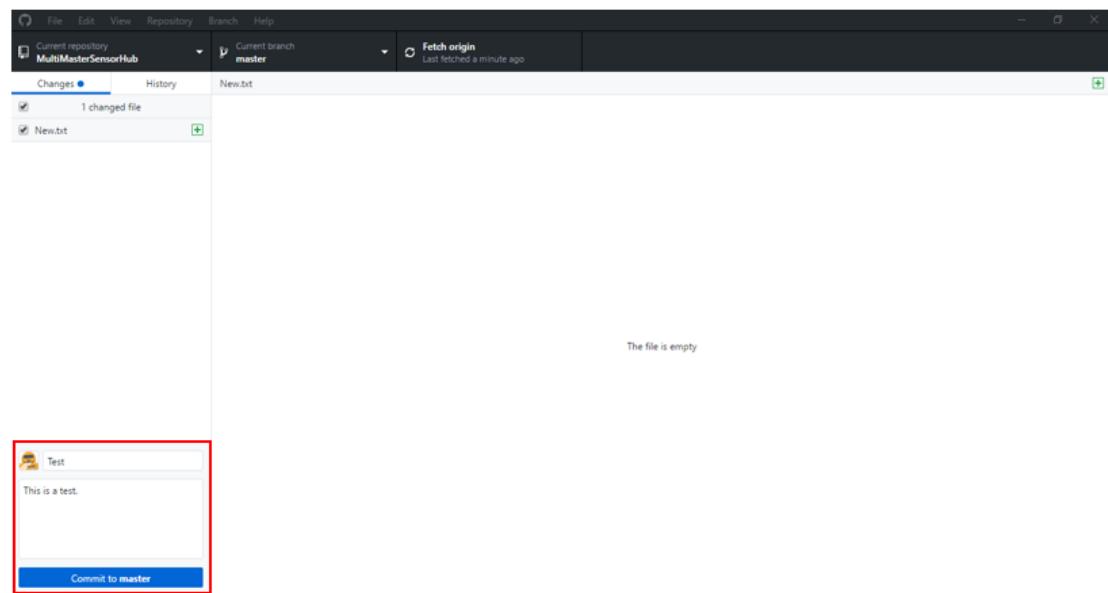


圖 5-20 New change

按下”Commit to master”後，可以看到藍色按鈕 Commit to master 下方出 Committed just now，代表這次要上傳的新版本內容，確認完畢後，按下 Push origin 上傳。接著回到 GitHub 確認是否有上傳成功，可以發現新增了 New.txt。

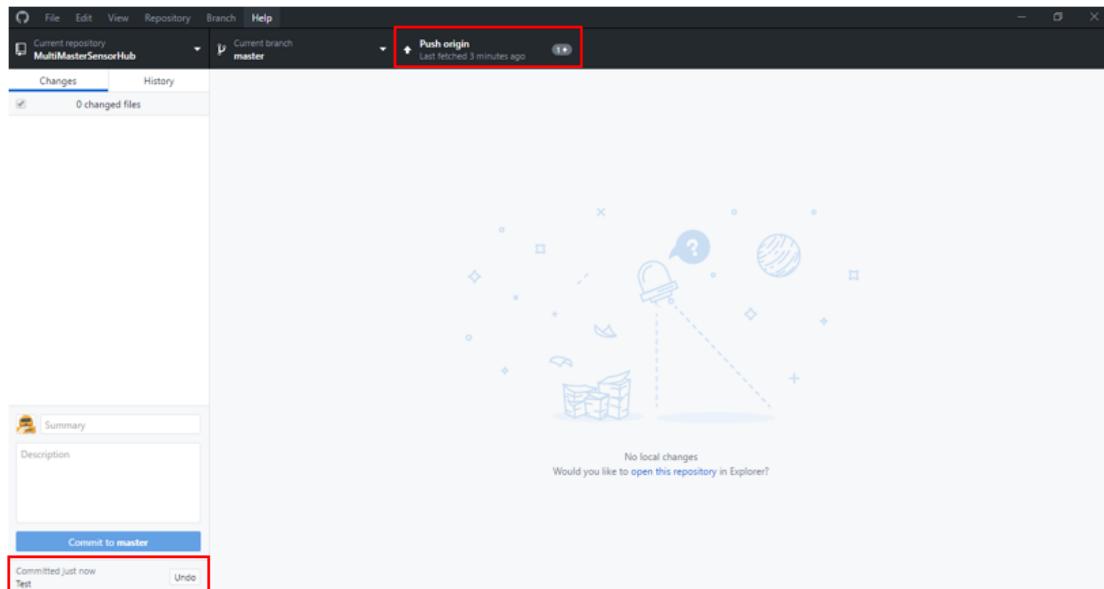


圖 5-21 Confirm and upload

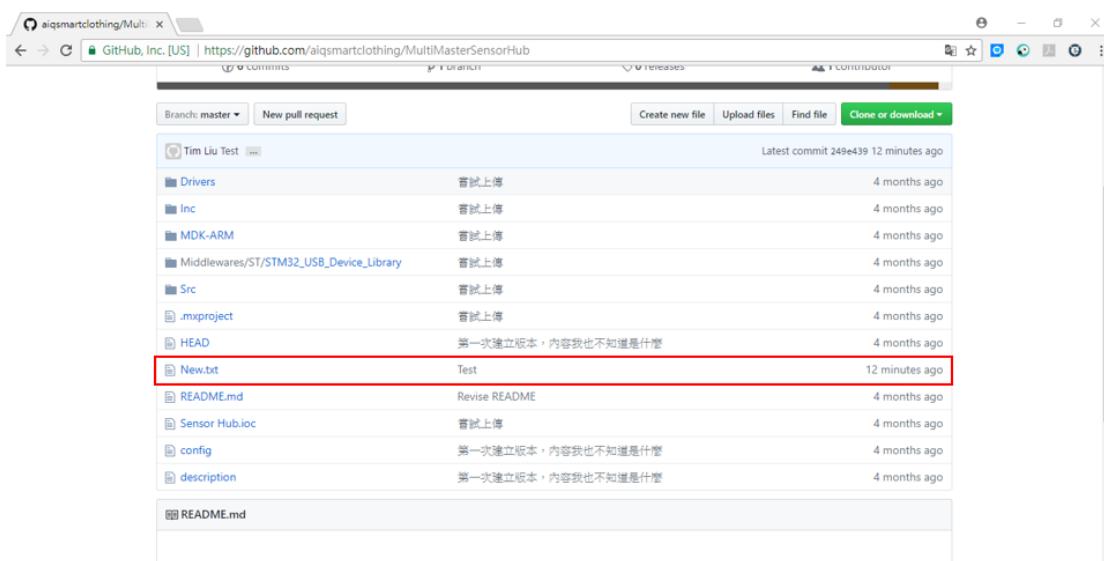


圖 5-22 New file upload success

## 6. 結語

### 6.1. 本模組目的

本模組的開發是希望提供一個簡單的資料獲取模組，讓想要獲取資料的開發者、使用者可以透過該模組來取得資料，做後續的發展與應用。

### 6.2. 展望

本資料獲取模組只是一個底層的開發，對於整個動作捕捉系統來說是基本的設計，期許未來有更多的開發人員可以基於此架構設計出更多的應用，也期望各開發人員可以不吝指教，讓這個資料獲取模組可以日益進步。

### 6.3. 參考文獻

- [1] STMicroelectronics, “STM32F412 advanced ARM®-based 32-bit MCUs,” RM0402, Jun. 2016 [Revised Jun. 2016].
- [2] STMicroelectronics, “iNEMO inertial module:always-on 3D accelerometer and 3D gyroscope, “LSM6DS33 datasheet, Oct. 2015 [Revised Oct. 2015].
- [3] Smart Clothing Sensor Hub S/W Specification V0.9
- [4] STMicroelectronics, “Using STM32F4 MCU power modes with best dynamic efficiency, “AN4365, May. 2014 [Revised May. 2014].