微處理機系統與介面技術 LAB 4

系所：電機 學號 :612415013 姓名：蕭宥羽

<實驗器材>

NUC 140 V2.0 開發板



<實驗過程與方法>

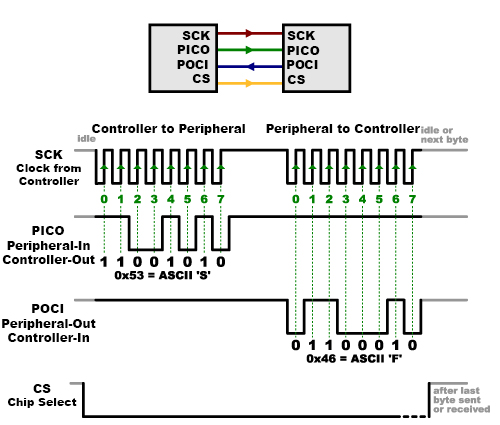
* 實驗要求 :

使用SPI Read 3 axis accelerometer and print on putty

Need to do calibration Result = (Raw data ± offset)/(256 ± offset)

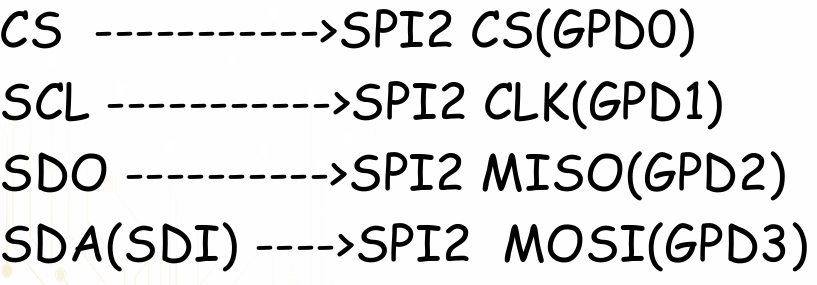
* SPI原理 : SPI是一種同步的串列數據通訊方式，通常可在全雙工模式下運作。

使用 4 條線進行通訊：SCK（時鐘）、MOSI（主輸出從輸入）、MISO（主輸入從輸出）和 SS（從機選擇）。主機通過 SCK 產生時鐘信號來同步資料的傳輸，MOSI 用於主機向從機發送數據，MISO 則用於從機向主機回傳數據。SS 用於選擇要通訊的特定從機，並控制資料的開始和結束。



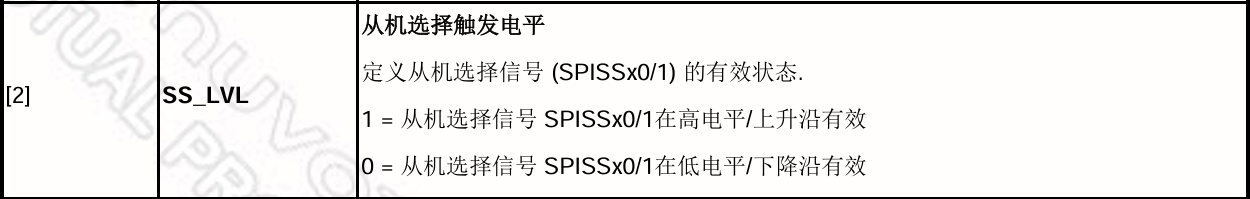
* SPI優缺點
* 優點
* 比非同步串列更快
* 可以支援多個設備
* 接收硬體可以是簡單的移位暫存器
* 缺點
* 需要多條的信號線
* 通信必須事先明確定義（不能隨時發送隨機數量的資料）
* 控制器必須控制所有通訊（外圍裝置不能直接相互通信）
* 通常需要為每個外設提供單獨的 SS 線，如果需要大量slaver，這可能會有問題。
* NUC140 SPI

本次lab使用SPI2，對應的GPIO為GPD，對應的腳位如下圖



* SPI register
* SPI->SSR (從機選擇暫存器)
* SS\_LVL

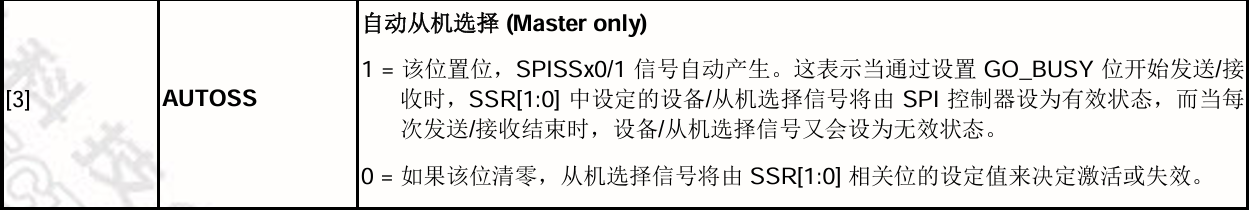
從機選擇訊號的觸發電平，選擇訊號（SS, Slave Select）是用高電平還是低電平來激活 SPI 從機



* AUTOSS

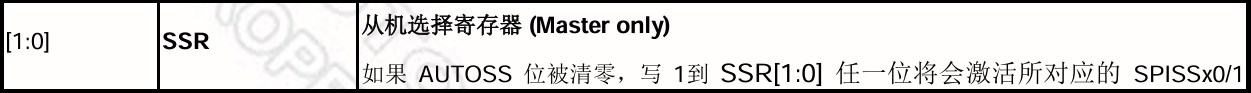
如果設為1的話，允許主機自動控制從機選擇訊號，當 SPI 控制器設置 GO\_BUSY 位元開始傳輸時，從機選擇信號（SPISSx0/1）將會自動變為有效狀態，表示開始與指定的從機通信。傳輸完成後，從機選擇信號自動恢復為無效。

本次lab要將這邊設為0，手動控制從機選擇訊號。



* SSR

從機選擇暫存器，當 AUTOSS 未啟用時，用於手動選擇與主機通信的從機



* SPI->CNTRL (控制與狀態暫存器)
* SLAVE

NUC140可以作為 master or slave，這邊設置SPI 的操作模式為主機或從機模式



* CLKP

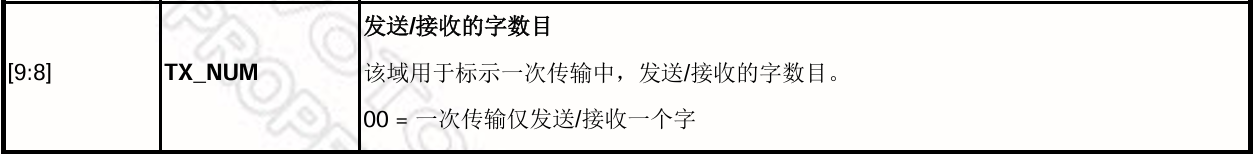
決定 SPI 時鐘信號的空閒狀態，如果 CLKP = 1，也就是說 SPI 時鐘在空閒狀態下保持高電平，那麼數據觸發會發生在變成low的過程中(下降沿or上升沿)

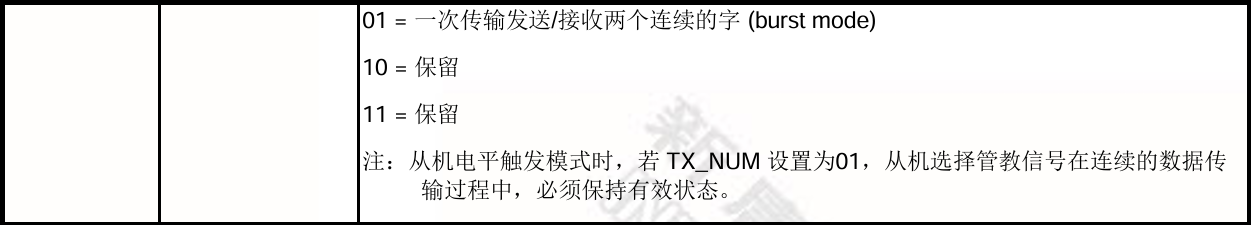


* TX\_NUM, TX\_BIT\_LEN

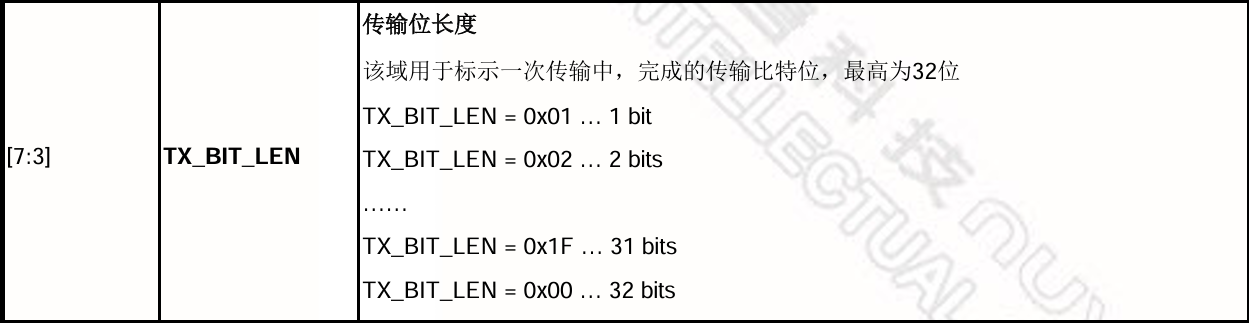
發送/接收的字數與位元長度

* TX\_NUM : 可以選擇一次傳輸一個或兩個字



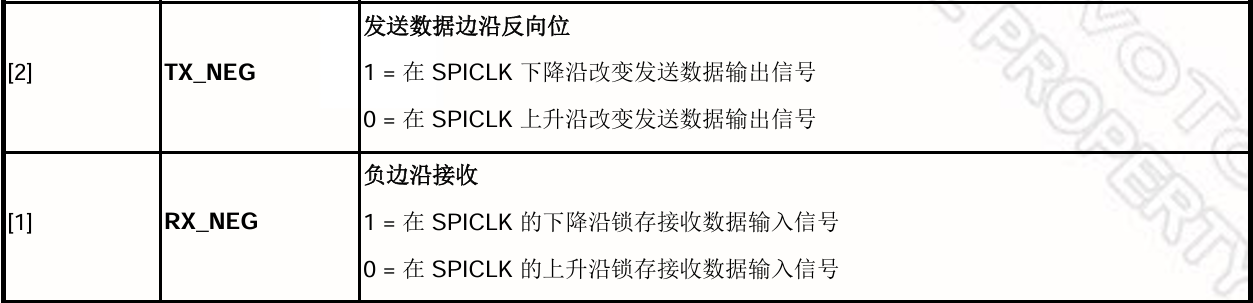


* TX\_BIT\_LEN : 指定 SPI 傳輸的位元數，可以傳輸不同位元長度的數據

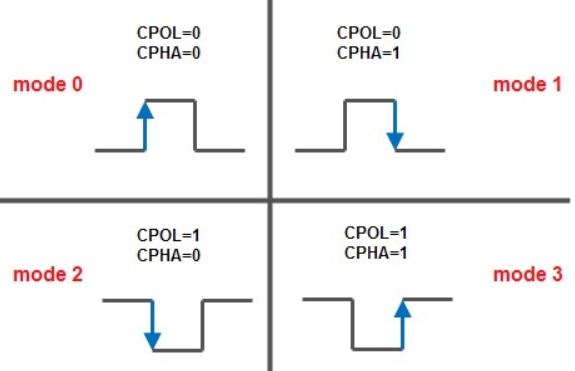


* TX\_NEG, RX\_NEG

SPI 在傳輸和接收數據時是使用時鐘的上升沿還是下降沿。

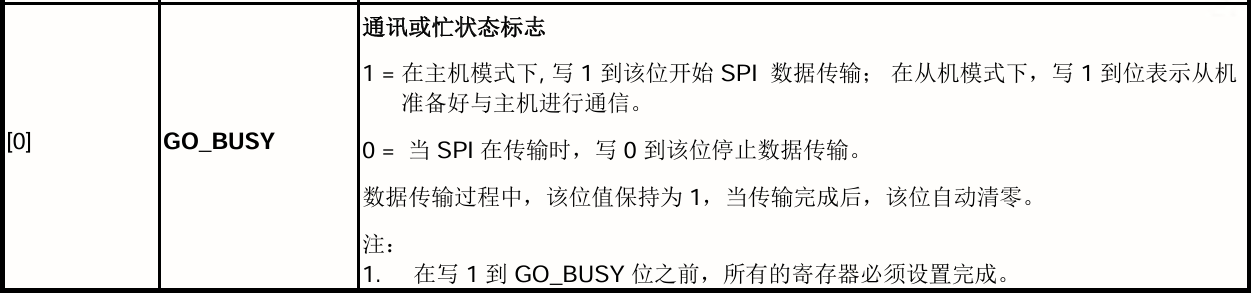


像是下圖這樣，選擇觸發方式



* GO\_BUSY

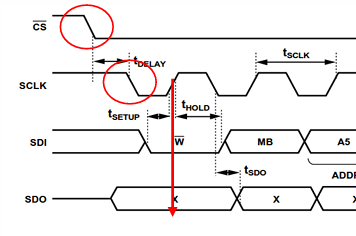
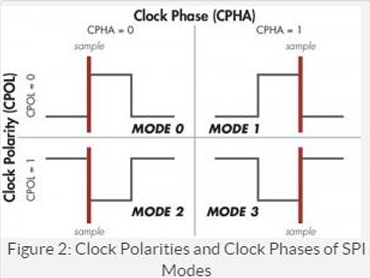
控制 SPI 的啟動和傳輸過程，設定為 1 會開始傳輸，當傳輸完成後自動清零。



* ADXL345 SPI
* NUC140 SPI設置

MCU SPI 的設定要根據ADXL的接收方式去做設定，由下面的圖(ADXL data sheet)可以看出

* ss/cs 拉低會啟動SPI通信 : SS\_LVL要設成0 (低電平有效)
* NUC140 為 master : SLAVE = 0 (master mode)
* 時鐘空閒時為高電平 : CPOL = 1，設置CLKP = 1
* 時鐘的上升沿觸發 : CPHA = 1，設置TX\_NEG, RX\_NEG = 0 (mode3)
* TX\_BIT\_LEN(8 bit data length for each word transmit) : TX\_BIT\_LEN=0x08
* TX\_NUM(One word in one transfer): TX\_NUM=00 一次傳輸一個字

* SPI 4-Wire Write

1. CS 拉低，SPI 開始通信
2. 主機通過 SDI 發送控制位和地址位，表示要寫入的暫存器

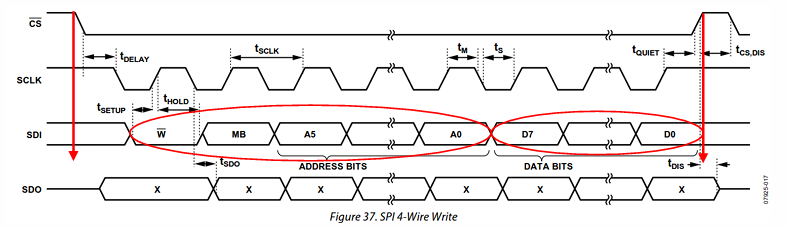
* W/R (bit7) : write(0) read(1)
* MB (bit6) :

MB = 1，表示將連續讀取或寫入多個位元組(連續的暫存器地址)

MB = 0，表示僅進行單位元組的讀取或寫入

* Address (bit0-bit5)

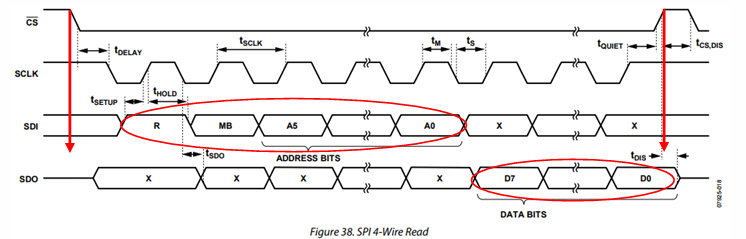
1. 接著主機發送數據位（D7-D0），要寫入到指定暫存器的數據
2. CS 拉高， SPI 傳輸結束



* SPI 4-Wire Read

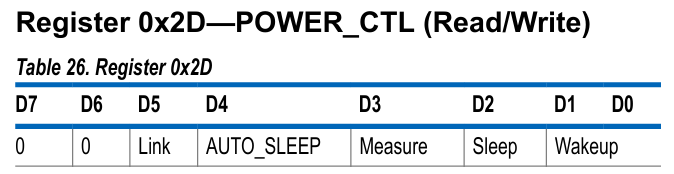
1,2,4同SPI 4-Wire Write

3. 從機根據接收到的地址位讀取相應的寄存器，並通過 SDO 發送數據位（D7-D0）給主機

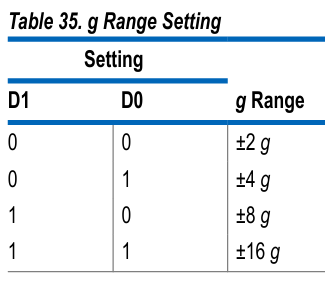
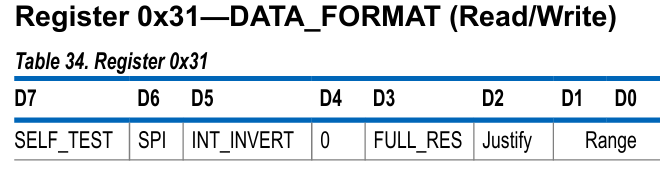


* Initial ADXL345
* POWER\_CTL(0x2D): 0x08

D3 = 1，其餘位元為 0，啟用測量模式



* DATA\_FORMAT(0x31): 0x0B(0000\_1011)

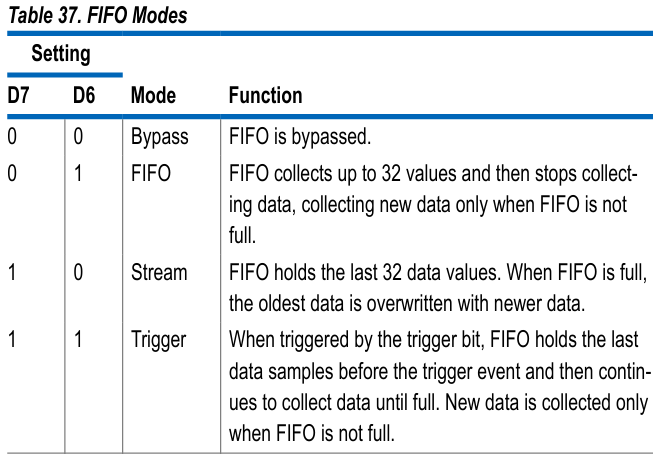
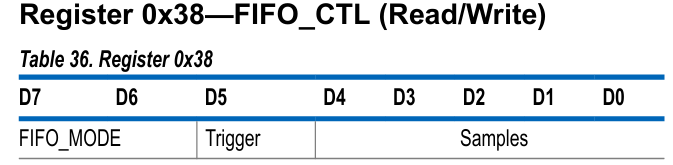


* FIFO\_CTL(0x38): 0x80

D7-D6（FIFO\_MODE）：設定 FIFO 的工作模式()

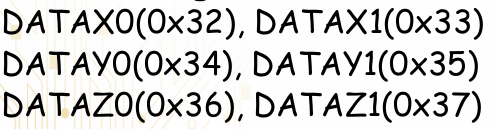
* STREAM mode(D7-D6= 10)

當 FIFO 緩衝器滿了時，最舊的數據會被丟棄，以便保存新的數據。因此，FIFO 始終保存最新的數據樣本。

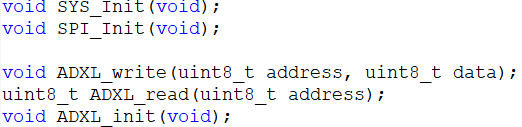


* ADXL data register

ADXL345 的加速度數據是由 16 位元組成的，每個軸的數據分為兩個 8 位暫存器存儲



* <Mian function code>
* 會使用以下這些function



1. SYS\_Init

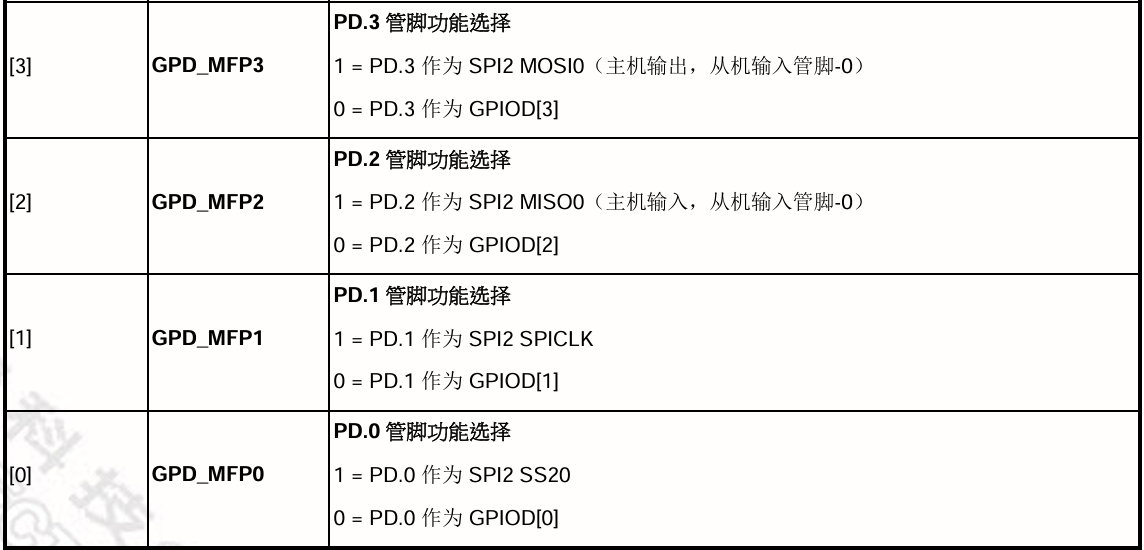
PD.0 設置為 SPI2 SS0，用於選擇從機。

PD.1 設置為 SPI2 SPICLK，用於 SPI2 的時鐘信號。

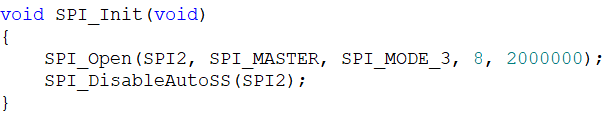
PD.2 設置為 SPI2 MISO0，用於數據的輸入（從機到主機）。

PD.3 設置為 SPI2 MOSI0，用於數據的輸出（主機到從機）。

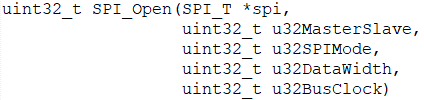




1. SPI\_Init
   * SPI\_Open(SPI2, SPI\_MASTER, SPI\_MODE\_3, 8, 2000000);
     + SPI2：要初始化的 SPI 模組(SPI2)
     + SPI\_MASTER：設置 SPI2 為主機模式（Master）
     + SPI\_MODE\_3 : CPOL = 1, CPHA = 1
     + 8 : 數據長度為 8 位元
     + 2000000：SPI 的時鐘頻率



* SPI\_Open實際操作方式是如下圖，他會對spi->CNTRL去做寫入，完成SPI的初始設定





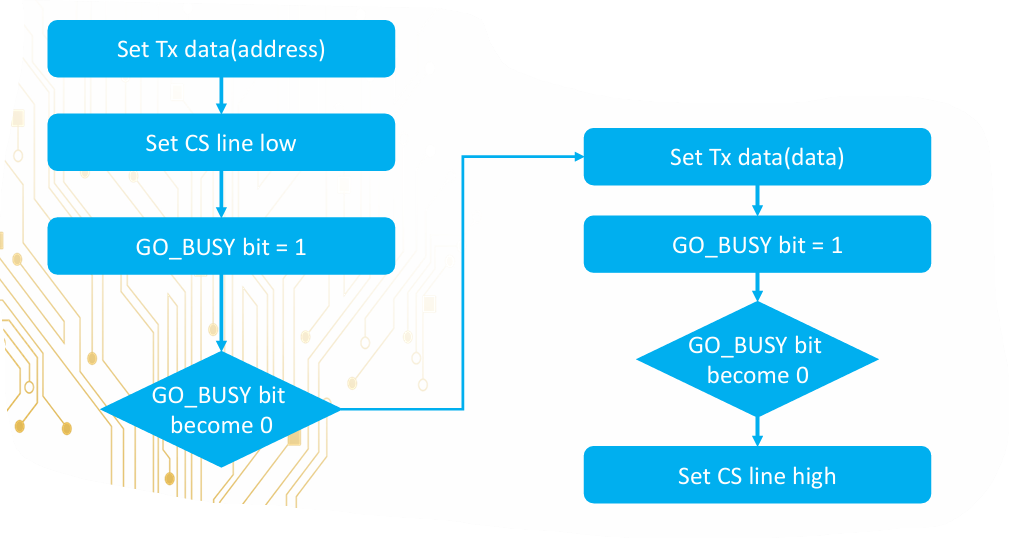
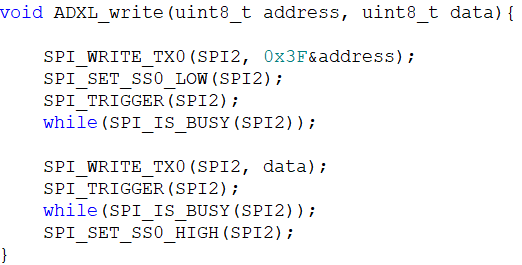
1. ADXL\_write

這邊就是ADXL寫入的流程，MCU藉由SPI傳遞暫存器位置給ADXL，接這再傳送要寫入的內容，對ADXL暫存器做寫入

* SPI\_WRITE\_TX0(SPI2, 0x3F & address);

0x3F & address：這裡將地址與 0x3F 進行按位與操作，讓最高位為 0，做寫操作

* SPI\_SET\_SS0\_LOW(SPI2); 拉低從機選擇信號，開始通信
* SPI\_TRIGGER(SPI2); while (SPI\_IS\_BUSY(SPI2)); 觸發 SPI 傳輸並等待完成
* 接這做同樣的流程寫入數據
* SPI\_SET\_SS0\_HIGH(SPI2); 將 SS0 拉高，表示此次通信結束

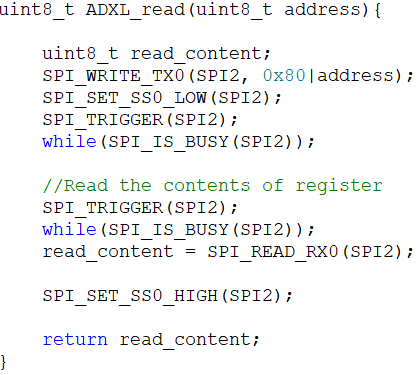
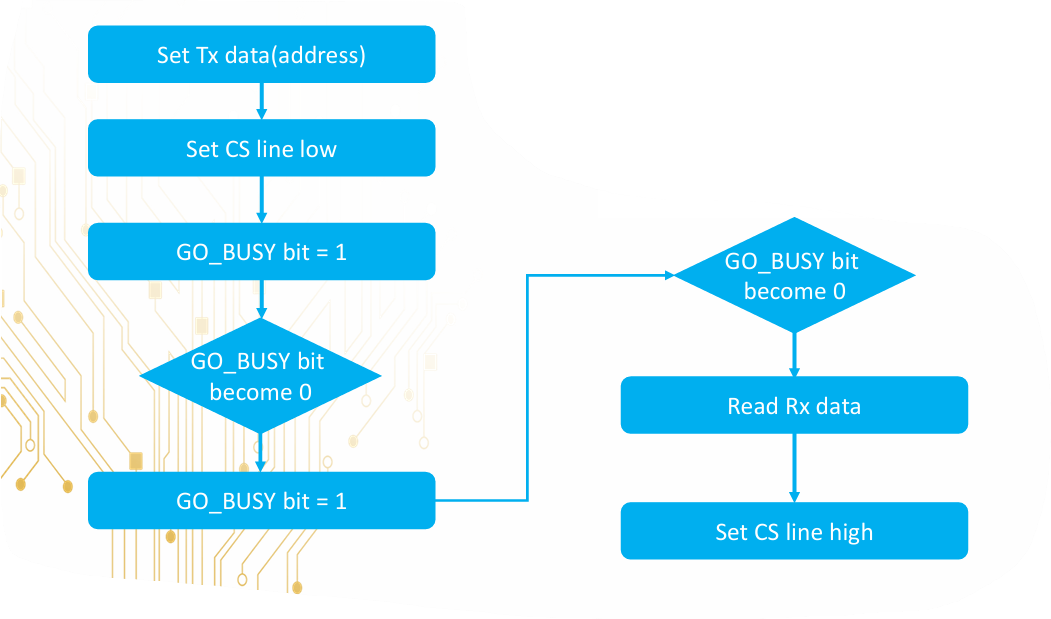


1. ADXL\_read

這邊跟write很類似，MCU藉由SPI傳遞暫存器位置給ADXL，決定我要讀取哪個ADXL的暫存器，接著ADXL用SPI將暫存器內容傳給MCU

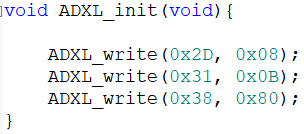
* SPI\_WRITE\_TX0(SPI2, 0x80|address);

0x80|address：這裡將地址與 0x80 進行按位與操作，讓最高位為 1，做讀操作

1. ADXL\_init

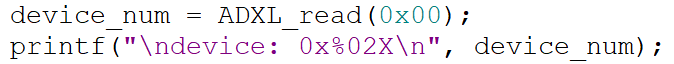
對ADXL的這些暫存器寫入這些值 (上面有說這些暫存器的作用)



1. Main

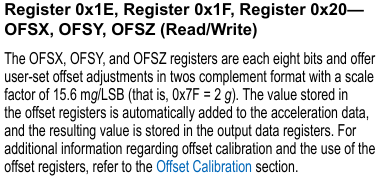
* 先讀取 ADXL 0x00 的內容，讀到的值一定要是 0xE5，我用這邊檢視SPI的傳輸程式碼是否是正常的





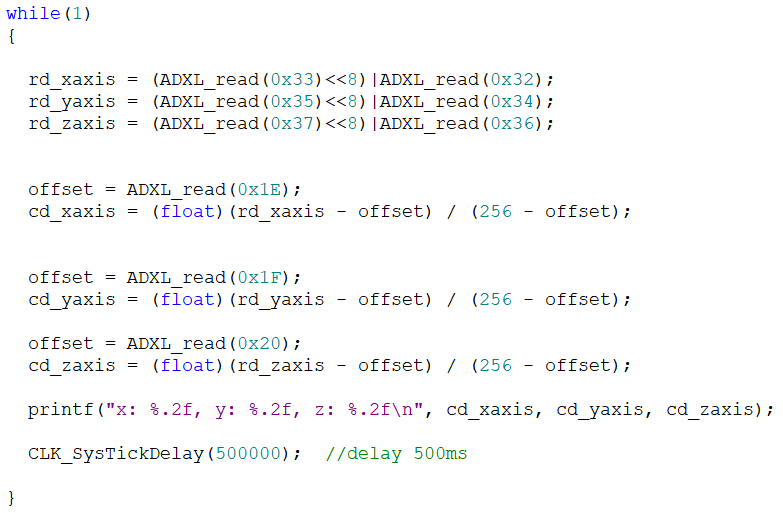
* while(1)
* rd\_xaxis = (ADXL\_read(0x33) << 8) | ADXL\_read(0x32);
  + **ADXL345** 的加速度數據是 **16 位元**，分為高位和低位來存儲在暫存器中
  + 先讀取高位（0x33），並左移 8 位，再與低位（0x32）進行按位 OR，以獲得完整的 **16 位 X 軸加速度數據**。
  + Y Z 是同樣的概念
* 偏移補償

ADXL\_read(0x1E)、ADXL\_read(0x1F)、ADXL\_read(0x20) 分別讀取 X、Y、Z 三個軸的偏移暫存器（偏移補償）的值，用於補償加速度感測器的零點漂移。



* 最後用offset進行校正，得到最終x y z的值





<心得與收穫>

這次的 SPI 實驗看似簡單，但在實作過程中發現，成功實現 SPI 功能，需要對 SPI 通訊協議及其硬體配置有深入的理解。每一個步驟都需要仔細設定，例如 SPI 的主從模式選擇、時鐘極性和相位的配置（CPOL、CPHA），以及從機選擇信號（SS）的控制，都需要對 SPI 模組的工作機制及硬體設計有充分的認識。此外，如何正確初始化 SPI，管理數據的傳輸，以及確保通訊的正確性，這些細節都非常重要。

在實驗過程中，我學習到如何有效設置 SPI 的參數，包括主從模式的選擇、數據位寬的設置，以及如何確保通訊時序與數據取樣的正確性。我理解了 SPI 模式（Mode 0-3）對數據傳輸的影響，並且明白了如何配置從機選擇信號以實現對不同設備的控制。這些操作看似簡單，但對於精確控制 SPI 通訊過程以及確保從機正確接收數據都非常重要。

同時，我也體會到在 SPI 實作中，系統初始化和數據傳輸的時序管理是非常關鍵的部分。從 SPI 控制寄存器的配置，到拉低並釋放從機選擇信號，每一個步驟都要求系統性和精確度，以確保數據的正確性並避免在通訊過程中的干擾與錯誤。

這次實驗給了我寶貴的經驗，使我對嵌入式系統中的 SPI 模組有了更深刻的理解，也明白了如何在硬體和軟體之間協調，以實現穩定、準確的 SPI 通訊。