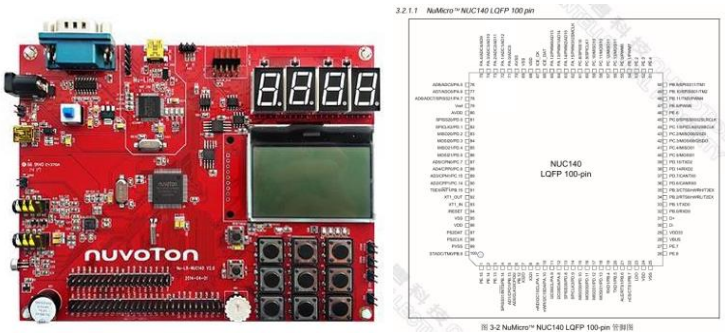
微處理機系統與介面技術 LAB 5

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 I2C Read 3 axis accelerometer and print on putty

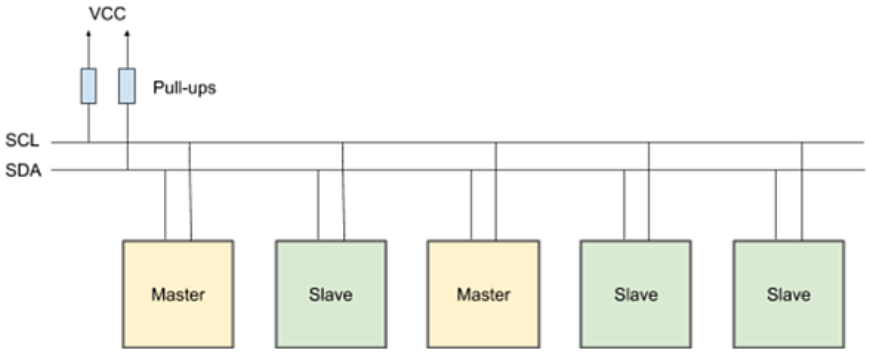
  Need to do calibration　Result = (Raw data ± offset)/(256 ± offset)

- I2C 原理 ：I2C 是一種同步的串列數據通訊方式

  使用 2 條線進行通訊 SCL（serial clock）、SDA（serial data），I2C 是一個 bus，在這個 bus 上所有的裝置都得透過這兩個訊號線相連

  ✓ SCL：產生時鐘信號來同步資料的傳輸

  ✓ SDA：用來傳輸兩邊的資料



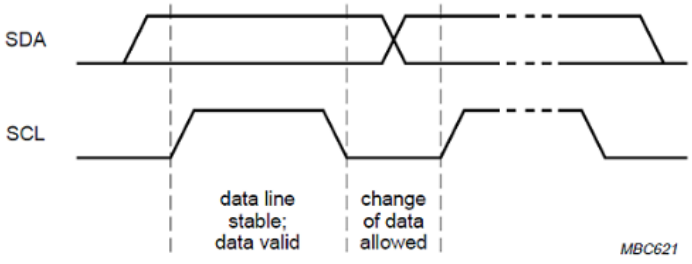
  ✓ I2C 的訊號邏輯規則

  1. 資料的有效性

     ✧ 資料的有效性取決於 SCL 時鐘訊號的狀態

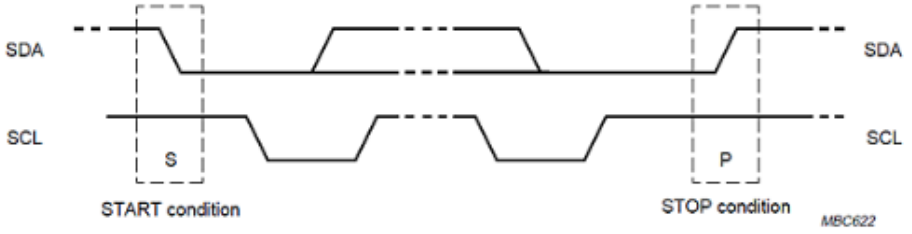
     ✧ 當 SCL 是 HIGH 時，SDA 狀態必須保持穩定，這表示此時的資料是有效的

     ✧ 當 SCL 是 LOW 時，SDA 的狀態可以改變



2. START STOP 傳輸開始、停止

當 SCL 為 high 時，如果 SDA 變動，有兩種特殊狀況：SCL high、SDA 下降代表 START；SCL high、SDA 上升代表 STOP 時序狀態。



3. Ack & Nack

每一個 8-bit 的資料傳輸結束後，會跟著一個 acknowledge bit(bit 9)。這個 acknowledge bit 固定由接收方產生，有兩種用法：

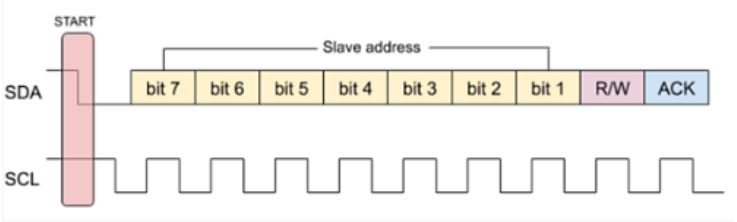
- ✧ 當 master 是 Transmitter、slave 是 Receiver，也就是說這個傳輸是 master 寫入資料到 slave 時，這個 acknowledge bit 是用來讓 slave 告訴 master「收到！」
- ✧ 反過來說， master 從 slave 讀取資料時，這個 acknowledge bit 是用來讓 master 告訴 slave「我還要接著讀，請繼續準備下一筆資料」或者是「夠了，我讀完了」。

✓ I2C 的傳輸流程

1. Slave Address

每次的傳輸都會由 master 發起(start bit)，接著會在 bus 傳送 slave address，去指定對哪一個 slave 操作，由下圖說明

1. master 發起 start bit，時脈開始傳送
2. master 傳送 slave address 以及操作方式(R/W)
3. slave 回傳 acknowledge bit

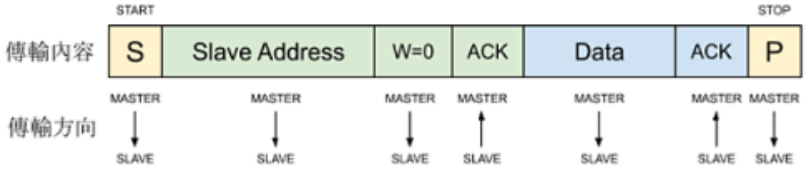


2. Write

Slave Address 傳輸完，Master 收到 acknowledge 後，就會繼續切換 SCL，並在每一個 SCL 爲 high 的脈波週期中，依序送出資料的每一個 bit，總共 8 個 bit。

第 9 個 bit 時，傳輸方向會換過來。Master 會在 SCL 變成 high 的第 9 個時脈週期中，放掉 SDA 不去驅動它(SDA 為 high)，並監聽來自 slave 的狀態；如果 slave 在此時有把 SDA 驅動到 low，就代表 slave 有正確地收到了這個 byte 的資料傳輸。

在 master 沒有送出 STOP 之前，就會持續這個循環，直到 master 送出 stop 結束傳輸。

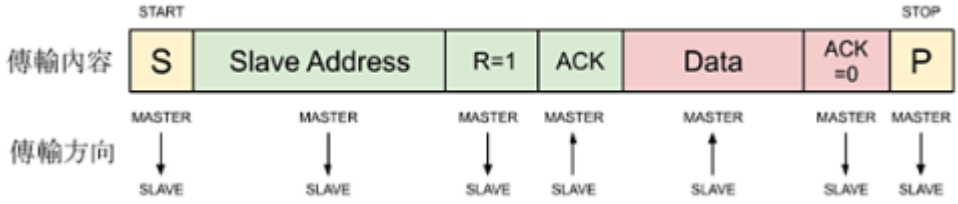


3. Read

Slave Address 傳輸完，Master 收到 acknowledge 後，slave 繼續傳送 data(8 個 bits)，第 9 個 bit 由 master 傳送 acknowledge bit 跟 slave 說還要不要繼續讀取

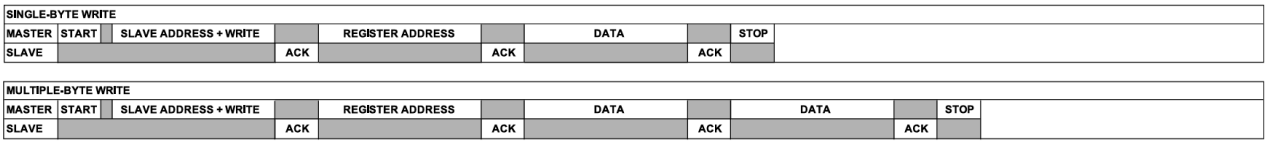
ACK=0： 讀取結束，接著 master 就會發送 stop bit

ACK=1： 還要繼續讀



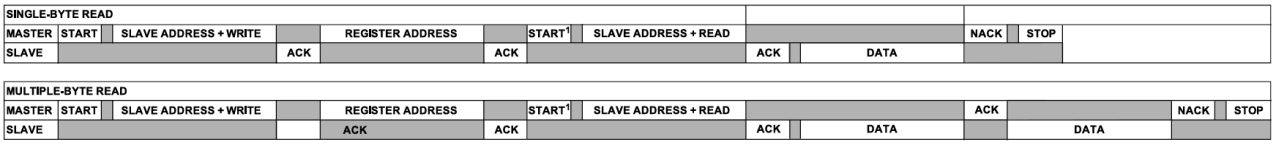
- ADXL345 I2C
  - ✓ Write

    master 寫入 adxl 的流程如下圖



  - ✓ Read

    讀取的流程如下圖，比寫入複雜一點的是，master 還是要先傳送要操作的 register 為只給 adxl345，所以會先用 write mode 傳輸，接著 restart(再發送一次 start bit)，指定 slave address and read mode，之後 adxl345 才會開始傳輸數據給 master



  - ✓ ADXL slave address
  - ✓ 本實驗我們將 ALT ADDRESS(SDO)接地，所以會使用 0x53 這個做為 ADXL 的 address

    藉由不同接法 adxl 會有不同 address，這樣可以接兩顆 adxl345 在 bus 上並且避免衝突

shown in Figure 41. With the ALT ADDRESS pin high, the 7-bit I^2C address for the device is 0x1D, followed by the R/$\overline{W}$ bit. This translates to 0x3A for a write and 0x3B for a read. An alternate I^2C address of 0x53 (followed by the R/$\overline{W}$ bit) can be chosen by grounding the SDO/ALT ADDRESS pin (Pin 12). This translates to 0xA6 for a write and 0xA7 for a read.

- NUC140 I2C
  - ✓ I2CON: I2C Control Register
    - EI (使能中斷)
      - ✧ 1：啟用 I2C 中斷。
      - ✧ 0：禁止 I2C 中斷。
      - ✧ 控制 I2C 模組是否可以觸發中斷。
    - ENS1 (I2C 控制器使能位)
      - ✧ 1：開啟 I2C 模組。
      - ✧ 0：禁用 I2C 模組。
      - ✧ 用來啟用或禁用整個 I2C 模組。
    - STA (I2C 起始控制位)
      - ✧ 1：進入起始模式，發送 START 信號。
      - ✧ 用來啟動 I2C 通訊，設定為 1 時，I2C 硬體會開始送出起始信號。
    - 位元 [4] - STO (I2C 停止控制位)
      - ✧ 1：發送 STOP 信號。
      - ✧ 在 I2C 傳輸結束時設置此位元來停止傳輸。
    - 位元 [3] - SI (I2C 中斷標誌)
      - ✧ 當 I2C 狀態改變時（例如完成一個操作）這個位元會被設置。
      - ✧ 當此位元為 1 時，表示有中斷需要處理，MCU 需進行相應處理。
    - 位元 [2] - AA (應答控制位)
      - ✧ 1：在主機模式下表示需要應答確認，在從機模式下表示準備好接收數據。
      - ✧ 0：不應答確認。
      - ✧ 這個位元用來控制是否發送應答信號。

Bits	描述	
[31:8]	**Reserved**	保留
[7]	**EI**	**使能中断** 1 = 使能 I²C 中断 0 = 禁用 I²C 中断
[6]	**ENS1**	**I²C 控制器使能位** 1 = 使能 0 = 禁用 设置使能 I²C 串行控制器功能。当 ENS1=1，I²C 串行功能使能。SDA 和 SCL 对应的多功能管脚功能必须设置为 I²C 功能。
[5]	**STA**	**I²C 起始控制位** 设置 STA 为 1，进入主机模式，如果总线处于空闲状态，I²C 硬件会送出 START 或 重复 START 条件。
[4]	**STO**	**I²C 停止控制位** 在主机模式，设置 STO 来传送一个 STOP 条件到总线，然后 I²C 硬件将会检查总线状况，如果检测到一个 STOP 状况，这个标志会被硬件自动清除。在 I²C 从机模式，设置 STO 复位 I²C 硬件来定义“无编址”从机模式，这表示在从机接收模式不再接收从主机传送装置发送的数据。
[3]	**SI**	**I²C 停止标志** 当一个新的 I²C 状态出现在寄存器 I2CSTATUS 时，SI 标志由硬件置位，并且如果 EI (I2CON [7]) 位被置位，则产生 I²C 中断请求。SI 必须由软件通过向该位写 '1' 清零。
[2]	**AA**	**应答控制位** 当 AA=1 先于地址或数据接收，在以下两种情况：1.) 从机正在应答主机发送的地址，2.) 接收设备正在应答发送设备发送的数据，在 SCL 线上的应答时钟脉冲期间将返回一

		个应答（SDA 上的低电平）。 当 AA = 0 先于地址或数据接收，则在 SCL 线上的应答时钟脉冲期间将返回一个非应答（SDA 上的高电平）。
[1:0]	**Reserved**	保留

✓ I2CDAT: I2C Data Register

Bits	**描述**	
[31:8]	**Reserved**	保留
[7:0]	**I2CDAT**	**I^2C 数据寄存器** Bit [7:0] 为 8 位 I^2C 串行端口传输数据

✓ I2CSTATUS: I2C Status Register

Bits	**描述**	
[31:8]	**Reserved**	保留
[7:0]	**I2CSTATUS**	**I^2C 状态寄存器** 低三位一直为 0。高五位是状态码，共有 26 个可能的状态码。当 I2CSTATUS [7:0] 的值为 F8H时，没有串行中断请求。所有其他的 I2CSTATUS [7:3] 的值对应 I^2C 的状态。当进入其中任一状态时，就会产生状态中断请求 (SI = 1)。在 SI 被硬件置位或 SI 被软件复位后一个机器周期，有效状态码出现在 I2CSTATUS[7:3] 中。 此外，00H 状态表示总线错误。总线错误发生在 START 或 STOP 条件出现在帧结构不正确的位置。不正确的位置比如是在串行传输地址字节、数据字节或应答位期间。

<Mian function code>

✓ 會使用以下這些 function

I2C 的程式原理是，收到 start、ack/nack、stop 會進入中斷並得到該狀態的狀態碼，中斷期間會做對應的操作，就如同上面所介紹了，這樣就可以完成 i2c 的操作

1. I2C0_IRQHandler(void)

這是 I2C0 中斷處理函數。當 I2C 發生中斷時，會執行這個函數：

- I2C_GET_STATUS(I2C0) 用來取得 I2C 的當前狀態。
- 如果偵測到超時 (I2C_GET_TIMEOUT_FLAG)，則清除超時旗標。
- 如果有註冊的 I2C 回調函數 (s_I2C0HandlerFn)，則執行該函數。

2. I2C_MasterRx(uint32_t u32Status)

這是 I2C 讀取操作的回調函數，用於處理 I2C 傳輸中的不同狀態：

- 0x08: 已經發送 START，準備傳送 SLA+W（Slave Address + Write）。
- 0x18: SLA+W 被發送，並且已收到 ACK（應答）。
- 0x20: SLA+W 被發送，並且收到 NACK（未應答），停止並重新開始。
- 0x28: 資料已發送且已收到 ACK，進行下一步。
- 0x10: 發送重複 START，準備 SLA+R（Slave Address + Read）。

- 0x40: SLA+R 被發送且已收到 ACK。
- 0x58: 已接收資料且返回 NACK，停止 I2C 讀取。

3. I2C_MasterTx(uint32_t u32Status)

這是 I2C 寫入操作的回調函數，用於處理 I2C 傳輸中的不同狀態：

- 0x08: 發送 START。
- 0x18: SLA+W 被發送並收到 ACK，將要寫入的註冊位址發送到從設備。
- 0x20: SLA+W 被發送並收到 NACK，停止並重新開始。
- 0x28: 資料已發送且收到 ACK，若有資料要發送，繼續發送，否則停止並完成。

4. SYS_Init(void)

用來初始化系統，包括時鐘及 I/O 設定：

- 啟用內部 RC 振盪器、外部 12MHz 晶體、PLL 等來配置系統時鐘。
- 啟用 UART0 和 I2C0 的模組時鐘。
- 設定 I2C0 的 SDA (PA.8) 和 SCL (PA.9)，以及 UART0 的 RXD 和 TXD。

5. I2C0_Init(void)

初始化 I2C0 模組：

- 設置 I2C 的 bus 時鐘為 100kHz。
- 啟用 I2C 中斷並在 NVIC 中使能 I2C0 的 IRQ。

6. I2C0_Close(void)

關閉 I2C0：

- 禁用 I2C0 中斷並清除 NVIC 中的中斷設置。
- 關閉 I2C0 及其時鐘，節省功耗。

7. ADXL_WriteBytes(uint8_t slvaddr, uint8_t reg_addr, uint8_t data)

用於向 ADXL345 加速度計寫入資料：

- 設定從設備位址和註冊位址，並將要寫入的資料存入 g_u8MstTxData。
- 設定 I2C 傳輸回調函數為 I2C_MasterTx，然後發送 START 信號來開始資料傳輸。
- 使用 while 等待傳輸完成。

8. ADXL_ReadBytes(uint8_t slvaddr, uint8_t reg_addr)

用於從 ADXL345 加速度計讀取資料：

- 設定從設備位址和註冊位址。
- 設定 I2C 傳輸回調函數為 I2C_MasterRx，然後發送 START 信號來開始讀取。
- 使用 while 等待讀取完成，並返回讀取的資料。

9. ADXL_init(void)

初始化 ADXL345 加速度計：

- 設置電源控制 (POWER_CTL)、資料格式 (DATA_FORMAT) 及 FIFO 控制 (FIFO_CTL) 等寄存器。
- 使用 ADXL_WriteBytes 向從設備地址 0x53 寫入配置。

10. my_round(float number)

進行浮點數的四捨五入運算，返回整數。正數大於等於 0 時，+0.5f 後轉換為整數；負數小於 0 時，-0.5f 後轉換為整數。

11. ADXL_calibrate(void)

對 ADXL345 進行校準：

- 重複讀取 X、Y、Z 軸 10 次並取平均，計算出平均偏移量。
- 計算補償偏移值（將 X、Y、Z 平均偏移除以 4），並將其寫入偏移寄存器 (0x1E, 0x1F, 0x20)。

12. main(void)

- 解鎖受保護的寄存器，進行系統和 UART 初始化。
- 初始化 I2C0 和 ADXL345。
- 讀取設備的 Device ID，並進行加速度計的校準。
- 進入無窮迴圈，不斷讀取 X、Y、Z 軸的資料並轉換為 g 值，最後將結果打印出來。

<心得與收穫>

這次的 I2C 實驗看似簡單，但在實作過程中發現，成功實現 I2C 通訊功能，需要對 I2C 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 I2C 的主從機模式選擇、啟動與停止條件的生成，以及應答訊號的控制，都需要對 I2C 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 I2C、管理資料的傳輸、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 I2C 的參數，包括主從機模式的選擇、傳輸速率的設置，以及如何在資料傳輸中管理 START 和 STOP 信號。我理解了應答訊號（ACK/NACK）的重要性，並明白了如何控制中斷來處理通訊過程中的不同狀態，以確保資料的正確傳輸。這些操作看似簡單，但對於精確控制 I2C 通訊過程以及確保從機正確接收資料都至關重要。

同時，我也體會到在 I2C 實作中，系統初始化和資料傳輸時序管理是非常關鍵的部分。從 I2C 控制暫存器的配置，到生成 START 信號並處理應答，再到釋放 STOP 信號，每一個步驟都要求精確和系統性，以確保資料的正確性，並避免在通訊過程中發生干擾或錯誤。此外，如何有效管理中斷及超時情況也是確保 I2C 通訊穩定的重要挑戰。

這次實驗給了我寶貴的經驗，使我對嵌入式系統中的 I2C 模組有了更深刻的理解，也明白了如何在硬體與軟體之間協調，以實現穩定、準確的 I2C 通訊。這些經驗不僅加強了我對 I2C 的認識，也增強了我在嵌入式系統開發中的實作能力。