微處理機系統與介面技術 LAB 6

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 :**
* Basic

Make a counter(計數器), and print on putty for every second

* Bouns

1. **雙計數器實現**：第一個計數器每秒計數 2 次，第二個計數器每秒計數 3 次。
2. **按鍵控制計數器**：Key1 暫停第一個計數器，Key2 暫停第二個計數器。
3. **計數器獨立性**：Key1 僅影響第一個計數器，Key2 僅影響第二個計數器。

* **Timer是什麼 有什麼用**

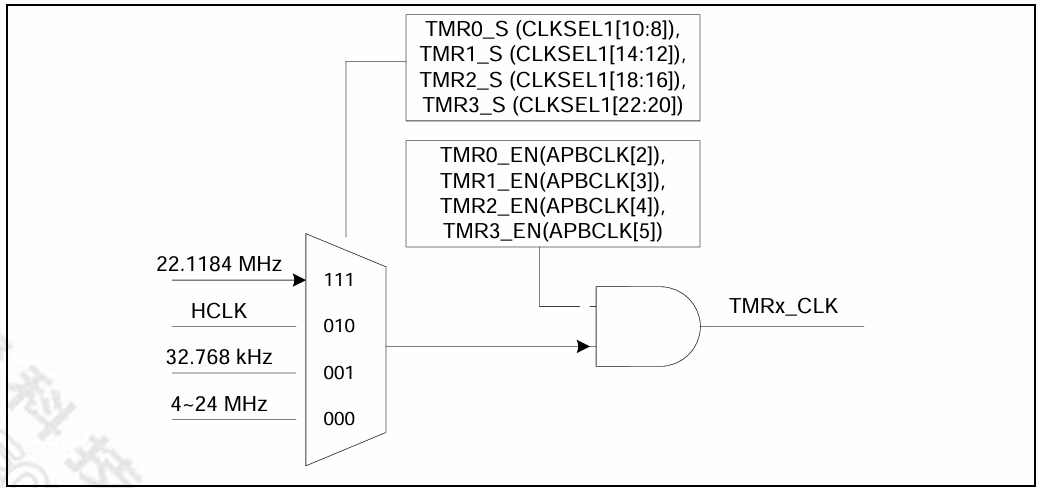
Timer 簡單的說就是一個會持續不斷的把 Clock 數量計算進入 Timer 中的模組，用於根據特定時間間隔觸發事件或執行任務。它是嵌入式系統中一個非常重要的模組，廣泛用於計時、週期性事件生成、訊號測量等用途。例如:

1. **計時**：根據硬體時鐘頻率，計算經過的時間
2. **週期性觸發**：可以設定為固定時間間隔觸發中斷，用於執行週期性任務。
3. **訊號生成**：用於產生 PWM（脈衝寬度調變）訊號，應用於馬達控制或亮度調節。

* **NUC140 timer**
  + block diagram

1. 時鐘輸入（TMRx\_CLK）

可以通過選擇不同的時鐘來源，提供timer的基礎頻率



1. 8 位預分頻計數器（8-bit Prescale）
   * + - 用於將時鐘頻率進一步分頻，降低時鐘頻率以適應計數需求。
       - 預分頻值可由軟體設定，從而改變輸入時鐘的頻率。
2. 24 位向上計數器（24-bit Up Timer）

* 用於進行實際的計數操作。
* 當計數器的值到達預設的比較值（TCMPR），可以觸發事件（如中斷或輸出訊號）。

1. 24 位比較寄存器（24-bit TCMPR）

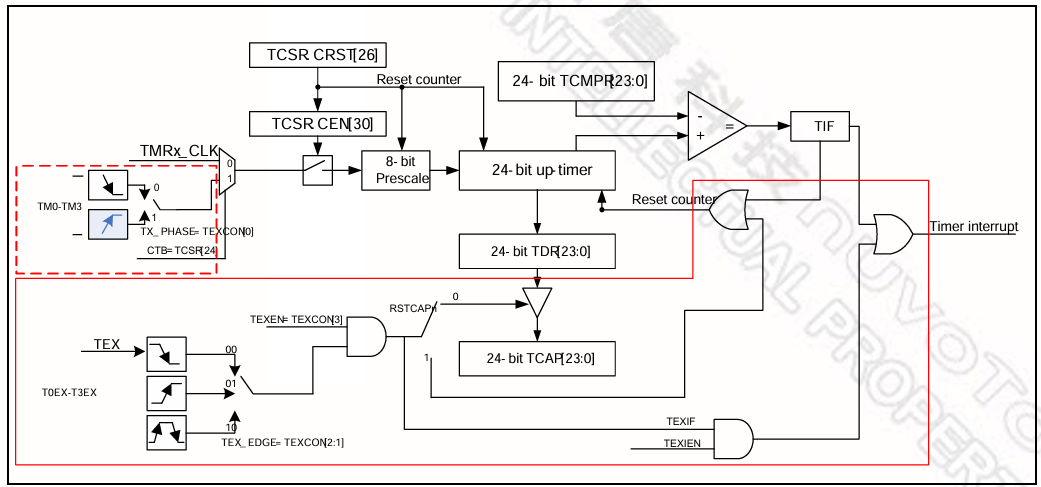
* 設定定時器的計數終點值。
* 當計數器值等於比較值時，產生中斷信號（TIF）。

1. 中斷旗標（TIF）

* 當計數器到達比較值時，設置此旗標，提示程式可以處理中斷。

1. 24 位資料寄存器（TDR 和 TCAP）

* TDR：用於儲存計數器當前值，軟體可以讀取此值。
* TCAP：用於捕捉外部事件信號的時間戳記



* + Timer mode

1. One-Shot mode

* 計數器啟動後，當計數到達設定的比較值時，產生一次中斷後停止計數。
* 適用於需要單次定時的場合，例如一次性延遲或事件觸發。

1. Periodic mode

* 計數器啟動後，當計數到達設定的比較值時，會產生中斷並自動重置計數，繼續下一個週期的計數。
* 適用於週期性任務，例如 LED 閃爍或週期性訊號生成。

1. Toggle mode

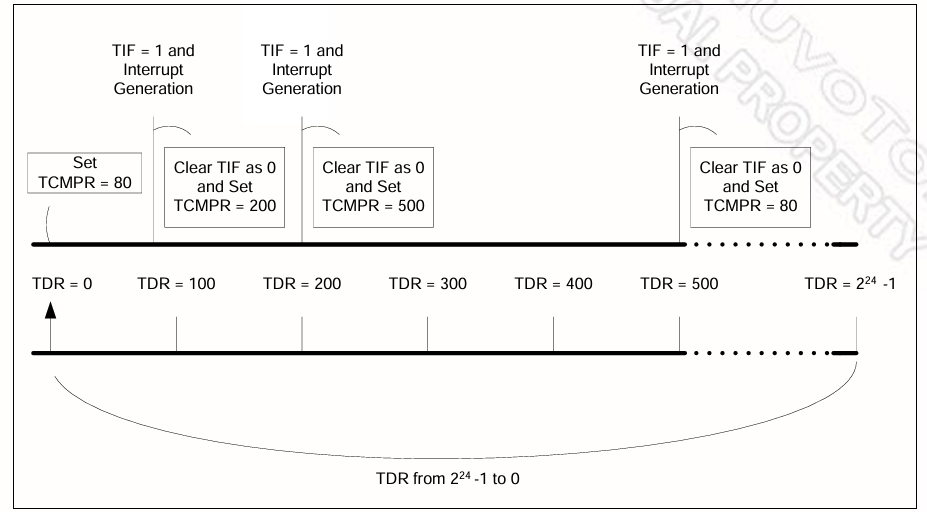
* 當計數到達比較值時，計數器會觸發中斷，並切換輸出訊號
* 適用於需要產生方波或脈衝訊號的場合

1. Continuous mode

* 計數器不斷計數，即使計數到達比較值並產生中斷，也不會停止計數，而是繼續運行，直到用戶手動改變設定。
* 適用於需要連續運行並可動態調整的應用，例如pwm。

由下方這張圖來解釋Continuous mode的運作流程

1. 初始設定 TCMPR = 80，計數器從 TDR = 0 開始計數，當計數值達到比較值 TCMPR = 80 時，會設置中斷旗標 TIF = 1，並觸發中斷信號通知 CPU，而計數器繼續運行。
2. CPU 在處理中斷時，清除中斷旗標 TIF = 0，並將比較值更新為 TCMPR = 200，計數器則從當前值繼續計數，直到 TDR = 200。
3. 當計數器達到新的比較值 TCMPR = 200 時，再次設置 TIF = 1 並觸發中斷，CPU 在清除旗標後更新比較值為 TCMPR = 500，計數器繼續運行。
4. 計數器在 TDR = 500 時再次觸發中斷，並在中斷中更新比較值，形成持續計數模式，直到計數值達到最大值(2^24 -1)，自動從零重新開始計數。



所以說這樣我們就可以用這個mode去實現pwm控制function，計數器從 0 開始計數，當計數值達到設定的比較值（TCMPR）時觸發中斷，改變輸出訊號的狀態（如從高變低），並在中斷例程中清除中斷旗標，然後設置新的比較值來決定下一次狀態切換的時間。這樣，通過對 TCMPR 的動態更新，可以靈活改變高電平和低電平的比例，從而控制 PWM 的占空比和頻率。

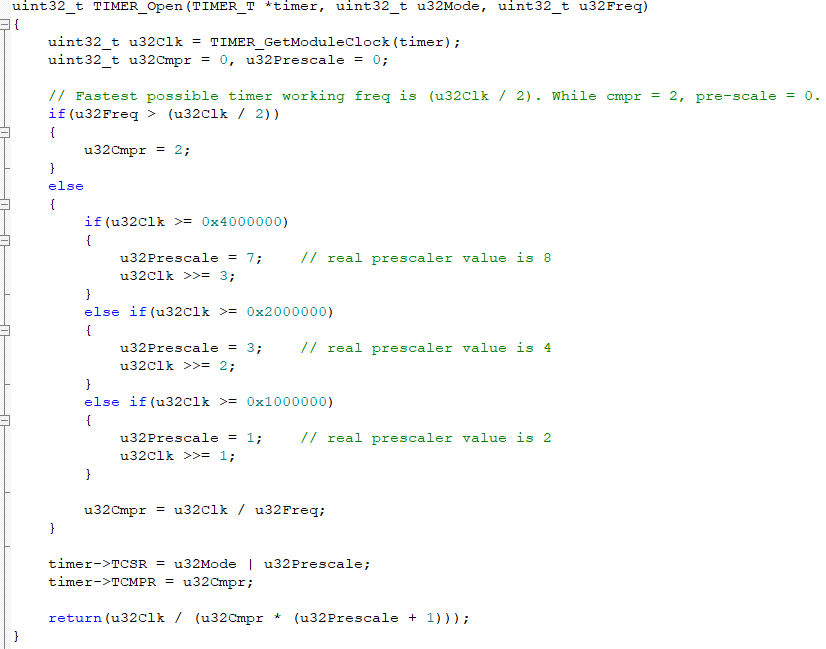
* + Timer open設定

這邊會先解釋後續會用到的Timer open這個function

如下面這個例子，TIMER0為所使用的timer定時器為timer0，並使用Periodic Mode，目標頻率以2 Hz運行，代表定時器每秒觸發中斷 2 次，即每 0.5 秒觸發一次



接下來會詳細介紹這段function是如何運作的



* <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。