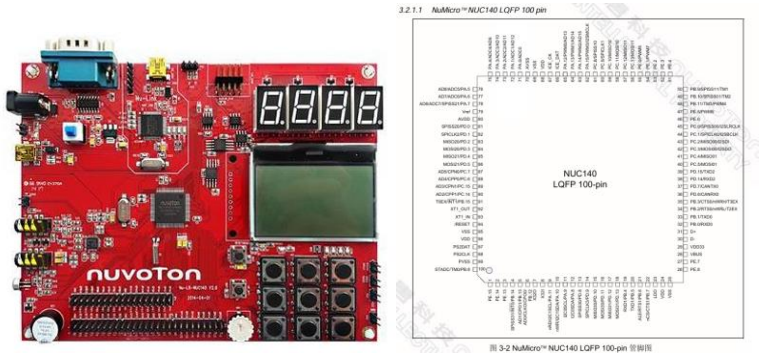
微處理機系統與介面技術 LAB 6

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

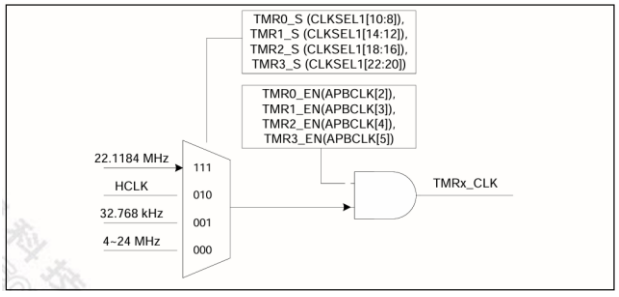
    Make a counter(計數器), and print on putty for every second
  - ✓ Bouns
    1. **雙計數器實現**：第一個計數器每秒計數 2 次，第二個計數器每秒計數 3 次。
    2. **按鍵控制計數器**：Key1 暫停第一個計數器，Key2 暫停第二個計數器。
    3. **計數器獨立性**：Key1 僅影響第一個計數器，Key2 僅影響第二個計數器。

- **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



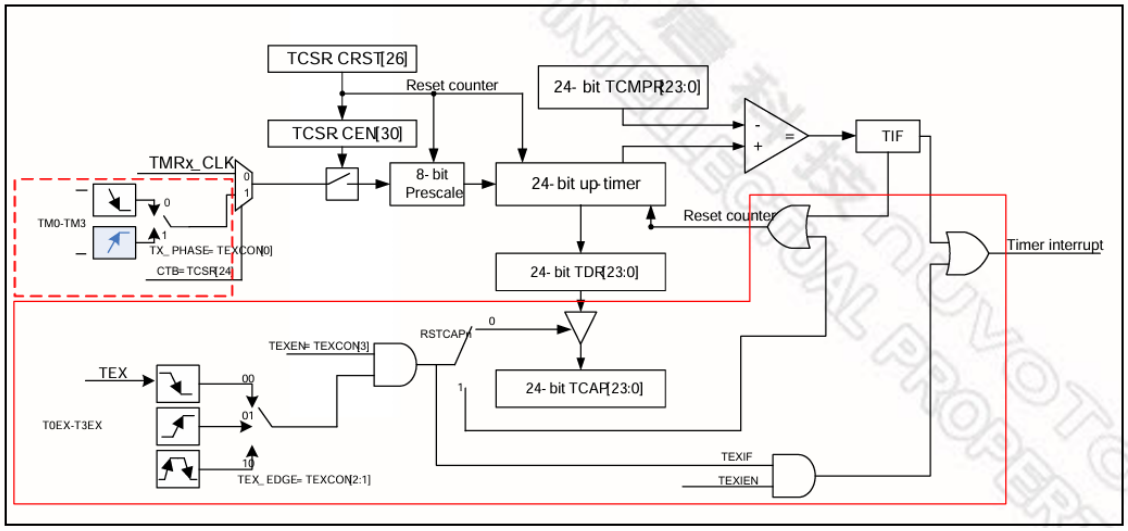
2. 8 位預分頻計數器（8-bit Prescale）
   - ✓ 用於將時鐘頻率進一步分頻，降低時鐘頻率以適應計數需求。
   - ✓ 預分頻值可由軟體設定，從而改變輸入時鐘的頻率。

3. 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）。

4. 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）。

5. 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。

6. 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



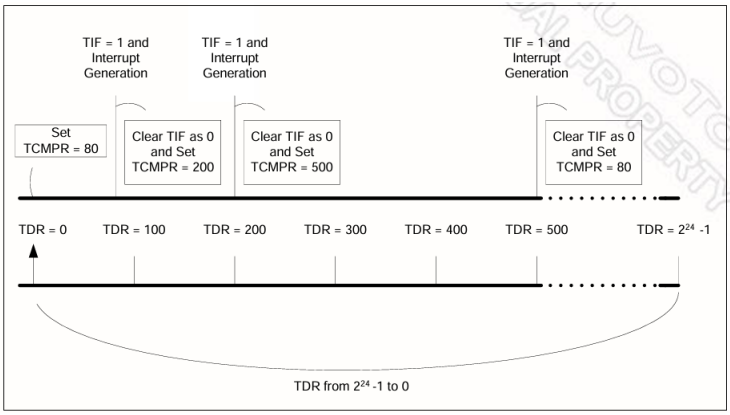
■ Timer mode

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。
2. 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。
3. 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

4. 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

```
/* Open Timer0 in periodic mode, enable int
TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, 2);
```

接下來會詳細介紹此 function 是如何運作的

◆

```
uint32_t u32Clk = TIMER_GetModuleClock(timer);
uint32_t u32Cmpr = 0, u32Prescale = 0;
```

使用 TIMER_GetModuleClock 函數獲取時鐘頻率

初始化比較值（u32Cmpr）和分頻器值（u32Prescale）

◆

```
if(u32Freq > (u32Clk / 2))
{
    u32Cmpr = 2;
}
```

如果用戶要求的目標頻率（u32Freq）高於定時器能達到的最高頻率（u32Clk / 2），直接將比較值（u32Cmpr）設為 2。

```
else
{
    if(u32Clk >= 0x4000000)
    {
        u32Prescale = 7;    // real prescaler value is 8
        u32Clk >>= 3;
    }
    else if(u32Clk >= 0x2000000)
    {
        u32Prescale = 3;    // real prescaler value is 4
        u32Clk >>= 2;
    }
    else if(u32Clk >= 0x1000000)
    {
        u32Prescale = 1;    // real prescaler value is 2
        u32Clk >>= 1;
    }

    u32Cmpr = u32Clk / u32Freq;
}
```

根據定時器的輸入時鐘頻率（u32Clk）的範圍選擇適合的分頻值（u32Prescale）

1. 如果時鐘頻率 u32Clk >= 0x4000000（64 MHz）
   - ✧ 設定分頻值為 7（實際分頻值為 7 + 1 = 8）。
   - ✧ 將時鐘頻率右移 3 位，相當於除以 8（u32Clk >>= 3）。
2. 如果時鐘頻率 u32Clk >= 0x2000000（32 MHz）
   - ✧ 設定分頻值為 3（實際分頻值為 3 + 1 = 4）。
   - ✧ 將時鐘頻率右移 2 位，相當於除以 4（u32Clk >>= 2）。
3. 如果時鐘頻率 u32Clk >= 0x1000000（16 MHz）
   - ✧ 設定分頻值為 1（實際分頻值為 1 + 1 = 2）。
   - ✧ 將時鐘頻率右移 1 位，相當於除以 2（u32Clk >>= 1）。
4. 計算比較值（u32Cmpr）

   使用分頻後的時鐘頻率（u32Clk），計算比較值，使得計數器達到比較值後的頻率等於用戶設定的目標頻率（u32Freq）。

$$u32Cmpr = \frac{u32Clk}{u32Freq}$$

◆ Example1 : u32Clk = 48 MHz、u32Freq = 2 Hz

1. 選擇分頻值

   u32Clk >= 0x2000000，選擇分頻值為 4 (u32Prescale = 3)

$$u32Clk = \frac{48MHz}{4} = 12MHz$$

2. 計算比較值

$$u32Cmpr = \frac{u32Clk}{u32Freq} = \frac{12MHz}{2} = 6{,}000{,}000$$

計數器需要計數 6,000,000 次，我們設定的頻率 2hz

Example2 : u32Clk = 12 MHz、u32Freq = 2 Hz

1. 選擇分頻值

   u32Clk = 12 MHz 不滿足上述條件，跳過所有分支，u32Prescale = 0（無分頻）

2. 計算比較值

$$u32Cmpr = \frac{u32Clk}{u32Freq} = \frac{12MHz}{2} = 6{,}000{,}000$$

◆

```
timer->TCSR = u32Mode | u32Prescale;
timer->TCMPR = u32Cmpr;

return(u32Clk / (u32Cmpr * (u32Prescale + 1)));
```

設定定時器控制寄存器 (TCSR) : 模式和分頻值

設定定時器比較值 (TCMPR)

回傳頻率的值 (Example2 : $\frac{12MHz}{6{,}000{,}000*(0+1)} = 2Hz$)

◆ 完整的 function 內容如下

```
uint32_t TIMER_Open(TIMER_T *timer, uint32_t u32Mode, uint32_t u32Freq)
{
    uint32_t u32Clk = TIMER_GetModuleClock(timer);
    uint32_t u32Cmpr = 0, u32Prescale = 0;

    // Fastest possible timer working freq is (u32Clk / 2). While cmpr = 2, pre-scale = 0.
    if(u32Freq > (u32Clk / 2))
    {
        u32Cmpr = 2;
    }
    else
    {
        if(u32Clk >= 0x4000000)
        {
            u32Prescale = 7;    // real prescaler value is 8
            u32Clk >>= 3;
        }
        else if(u32Clk >= 0x2000000)
        {
            u32Prescale = 3;    // real prescaler value is 4
            u32Clk >>= 2;
        }
        else if(u32Clk >= 0x1000000)
        {
            u32Prescale = 1;    // real prescaler value is 2
            u32Clk >>= 1;
        }

        u32Cmpr = u32Clk / u32Freq;
    }

    timer->TCSR = u32Mode | u32Prescale;
    timer->TCMPR = u32Cmpr;

    return(u32Clk / (u32Cmpr * (u32Prescale + 1)));
}
```

<Mian function code>

1.

```
/* Enable Timer 0~3 module clock */
CLK_EnableModuleClock(TMR0_MODULE);
CLK_EnableModuleClock(TMR1_MODULE);

/* Select Timer 0~3 module clock source */
CLK_SetModuleClock(TMR0_MODULE, CLK_CLKSEL1_TMR0_S_HXT, NULL);
CLK_SetModuleClock(TMR1_MODULE, CLK_CLKSEL1_TMR0_S_HXT, NULL);
```

啟用 Timer0 和 Timer1 的硬體時鐘模組
設定 Timer0 和 Timer1 的時鐘來源為 HXT

2.

```
/* Open Timer0 in periodic mode, enable interrupt and 1 interrupt tick per second */
TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, 2);
TIMER_EnableInt(TIMER0);

/* Open Timer1 in periodic mode, enable interrupt and 2 interrupt ticks per second */
TIMER_Open(TIMER1, TIMER_PERIODIC_MODE, 3);
TIMER_EnableInt(TIMER1);
```

將 Timer0 和 Timer1 開啟並設置為週期模式，目標頻率分別為 2Hz and 3Hz
並啟用 timer 中斷，這樣 Timer0 每秒觸發 2 次中斷；Timer1 則是每秒 3 次

3.

```
void TMR0_IRQHandler(void)
{
    if(TIMER_GetIntFlag(TIMER0) == 1)
    {
        /* Clear Timer0 time-out interrupt flag */
        TIMER_ClearIntFlag(TIMER0);

        g_au32TMRINTCount[0]++;
    }
}
```

```
void TMR1_IRQHandler(void)
{
    if(TIMER_GetIntFlag(TIMER1) == 1)
    {
        /* Clear Timer1 time-out interrupt flag */
        TIMER_ClearIntFlag(TIMER1);

        g_au32TMRINTCount[1]++;
    }
}
```

✓ TIMER_GetIntFlag(TIMER0)：檢查是否是 Timer0 的中斷。
✓ TIMER_ClearIntFlag(TIMER0)：清除 Timer0 的中斷旗標，允許下一次中斷發生。
✓ g_au32TMRINTCount[0]：用於累加 Timer0 中斷觸發的次數，作為計數器使用。

Timer1 同理，所以這樣 g_au32TMRINTCount[0]每秒會被加次， g_au32TMRINTCount[1]每秒會被加 3 次，這樣就可以實現題目的要求第一個計數器每秒計數 2 次，第二個計數器每秒計數 3 次

4. 按鍵掃描 (GPIO lab1)

```
uint8_t ScanKey(void)
{

  PA0=1; PA1=1; PA2=0; PA3=1; PA4=1; PA5=1;
  CLK_SysTickDelay(10);
  if (PA3==0) return 1;
  if (PA4==0) return 4;
  if (PA5==0) return 7;

  PA0=1; PA1=0; PA2=1; PA3=1; PA4=1; PA5=1;
  CLK_SysTickDelay(10);
  if (PA3==0) return 2;
  if (PA4==0) return 5;
  if (PA5==0) return 8;

  PA0=0; PA1=1; PA2=1; PA3=1; PA4=1; PA5=1;
  CLK_SysTickDelay(10);
  if (PA3==0) return 3;
  if (PA4==0) return 6;
  if (PA5==0) return 9;
  return 0;
}
```

```
void OpenKeyPad(void)
{
    PA -> PMD = (PA -> PMD & 0xFFFFF000)|0x0FD5;
}
```

5.

```
current_key_value = ScanKey();

if (prev_key_value_1 == 1 && current_key_value == 0) {
  timer0_flag = !timer0_flag;
  if (timer0_flag) {
      TIMER_Start(TIMER0);
  } else {
      TIMER_Stop(TIMER0);
  }
}
prev_key_value_1 = current_key_value;

if (prev_key_value_2 == 2 && current_key_value == 0) {
  timer1_flag = !timer1_flag;
  if (timer1_flag) {
      TIMER_Start(TIMER1);
  } else {
      TIMER_Stop(TIMER1);
  }
}
prev_key_value_2 = current_key_value;
```

- ✓ current_key_value = ScanKey();
  使用 **ScanKey()** 函數掃描按鍵
  - ✧ **1**：對應控制 **Timer0** 的按鍵。
  - ✧ **2**：對應控制 **Timer1** 的按鍵。
  - ✧ **0**：表示未檢測到按鍵。
- ✓ 控制 Timer0 的啟停
  - ✧ 檢查是否檢測到按鍵 **1** 按下又釋放
  - ✧ timer0_flag = !timer0_flag;
    每按一次(按下又釋放)key1，會反轉 timer0_flag 狀態，timer0_flag 去控制是否停止 timer，這樣就可以實現按鍵對 counter 的開始與暫停
- ✓ Timer1 同理

6. Print 結果出來

```
printf("\rcounter1: %d   counter2: %d"   ,g_au32TMRINTCount[0],g_au32TMRINTCount[1]);
```

<過程中遇到的困難>

在這次實驗過程中，我最初嘗試使用單一的 Timer 來實作所有功能，但發現這樣會使整體程式邏輯變得較為複雜，尤其在處理多項計時任務時，程式的可讀性和可維護性都受到影響。後來，在助教的建議下，我改為使用兩個 Timer 分別處理不同的功能。這種方式不僅讓整體程式的架構更為清晰，也大幅降低了程式的複雜度，使程式更容易理解與維護。

<心得與收穫>

在這次的 Timer 實驗中，看似簡單的計時功能，實際實作過程中卻讓我深刻體會到，成功實現 Timer 的功能需要對其運作原理及硬體配置有一定的掌握。例如，Timer 的模式選擇（如週期模式、單次模式）、比較值（TCMPR）的設定、以及中斷的觸發與處理等，每一個步驟都需要細緻地設計，才能確保 Timer 能夠準確地完成計時功能。

在實驗過程中，我學習到如何有效配置 Timer 的參數，包括選擇合適的時鐘源、設置分頻值以調整 Timer 的運行頻率，以及如何正確設定比較值以實現精確的計時。我理解了 Timer 中斷的核心作用，並掌握了如何在中斷服務例程中處理計數邏輯，確保 Timer 能夠穩定運行。此外，我還學會了如何使用多個 Timer 協作完成不同的計時任務，從而提升了程式的靈活性和可讀性。

同時，我也體會到在 Timer 的實作過程中，初始化和中斷管理是非常關鍵的部分。從 Timer 模組的時鐘啟動與配置，到設置比較值以觸發中斷，再到清除中斷旗標，每一個操作都需要精確執行，否則可能會導致計時誤差或中斷錯誤。

這次實驗給了我寶貴的經驗，使我對嵌入式系統中的 Timer 模組有了更深刻的認識，也學會了如何在硬體與軟體之間進行協調，實現穩定、準確的計時功能。這些經驗不僅加強了我對 Timer 的理解，也增強了我在嵌入式系統開發中的實作能力，尤其是在面對多任務計時需求時能夠設計出更高效的解決方案。