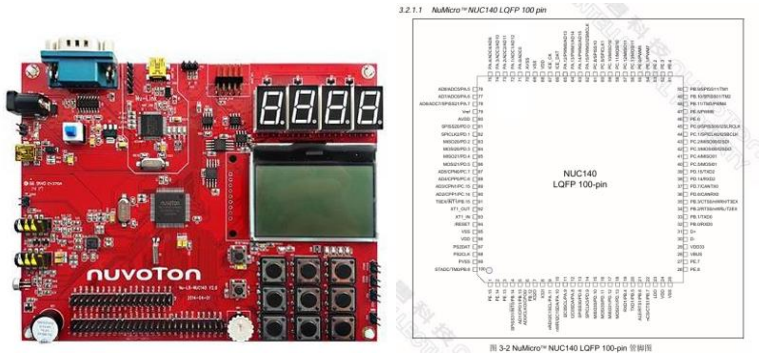
微處理機系統與介面技術 LAB 7

系所：電機 學號 :612415013 姓名：蕭宥羽

<實驗器材>

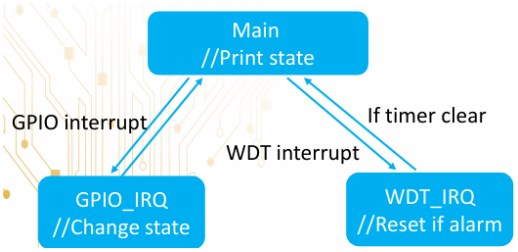
NUC 140 V2.0 開發板



<實驗過程與方法>

✓ 實驗要求：

- Let emergency alarm can rese
- Use WDT reset the emergency alarm when the state is not safe



✓ Watch Dog 是什麼 有什麼用

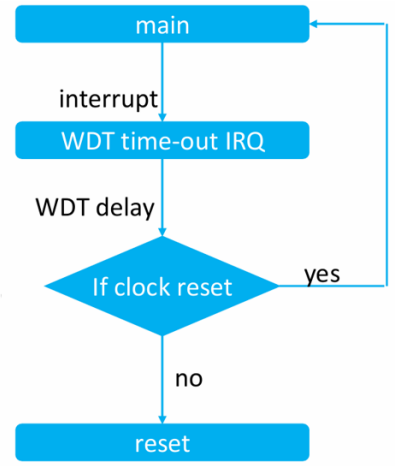
Watchdog 是一種計時器機制，用來監控系統是否正常運作。一旦系統因某些原因（如程式執行錯誤、陷入無窮迴圈或硬體故障）而無法回應，Watchdog 會自動重啟系統或執行特定的安全動作，以確保系統恢復正常。

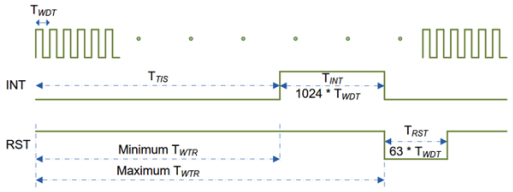
- Watchdog 的基本概念
  - ✧ 工作原理：Watchdog 是一個硬體計時器，系統需在正常運行時定期對它進行"餵狗"操作（重置計時器）。如果超過預定時間未重置，Watchdog 會認為系統出現問題，執行重啟或其他指定的動作。
  - ✧ 餵狗（Feeding the Watchdog）：定期重置 Watchdog 計時器的過程。如果程式無法按時執行餵狗，表示系統可能已卡住或進入異常狀態。
- Watchdog 有什麼用

  需要高可靠性或持續運行的設備中。例如，醫療設備、工業控制器以及自動駕駛車輛等應用。另外，它也能有效處理程式死鎖或其他異常情況，確保系統能夠及時恢復到穩定的狀態。
- WDT 的工作機制和觸發流程

  1. 計數階段
     - 計數器開始運行，並根據設定的超時時間運行
     - 當計數器達到超時間隔時，觸發中斷

2. 中斷延遲階段
   - 在中斷觸發後，計數器進入延遲期
   - 如果此期間內執行餵狗操作，計數器將重置，系統繼續正常運行
3. 復位階段
   - 如果延遲期結束後仍未餵狗，WDT 生成復位信號，觸發系統重啟

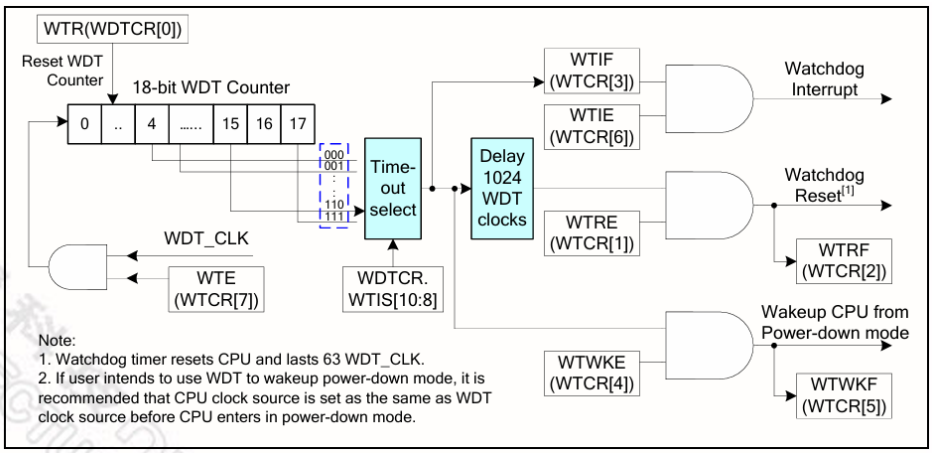




✓ NUC140 watch dog

■ Block diagram

1. **18-bit WDT 計數器**
   - 看門狗定時器是一個 **18** 位的計數器，從 **0** 開始計數，當計數達到超時條件時，會觸發相應的事件。
   - **Reset WDT Counter**：如果系統在計數完成前執行了看門狗計數器的重置，計數器會回到 **0**，避免超時。
2. **WDT_CLK**
   - 看門狗的時鐘信號（**WDT_CLK**）控制計數器的遞增速度
3. 超時條件選擇（**Timeout Select**）
   - **WTCR.WTIS[10:8]** 設置了看門狗的超時條件，可以選擇不同的超時時間（如 **2^n** 個時鐘週期）
4. **Delay 1024 WDT Clocks**
   - 在計數器達到超時條件後，看門狗會進一步延遲 **1024** 個 **WDT_CLK**，給系統留下一個緩衝期用於處理當前任務。



■ 看門狗定時器控制暫存器(WTCR)

31	30	29	28	27	26	25	24
DBGACK_WDT	Reserved						
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
Reserved					WTIS		
7	6	5	4	3	2	1	0
WTE	WTIE	WTWKF	WTWKE	WTIF	WTRF	WTRE	WTR

■ WTIS

✧ 功能：用於設置看門狗定時器的溢出時間間隔。

✧ 溢出時間 ：計數器達到設定的溢出值時觸發中斷的那段時間

[10:8]	WTIS	**看门狗定时器间隔选择**（写保护位） 这些位选择看门狗定时器的定时溢出间隔。

WTIS	定时溢出间隔	中断周期	WTR 定时溢出间隔 (WDT_CLK=10 kHz)
000	$2^4 * T_{WDT}$	$(2^4 + 1024) * T_{WDT}$	1.6 ms ~ 104 ms
001	$2^6 * T_{WDT}$	$(2^6 + 1024) * T_{WDT}$	6.4 ms ~ 108.8 ms
010	$2^8 * T_{WDT}$	$(2^8 + 1024) * T_{WDT}$	25.6 ms ~ 128 ms
011	$2^{10} * T_{WDT}$	$(2^{10} + 1024) * T_{WDT}$	102.4 ms ~ 204.8 ms
100	$2^{12} * T_{WDT}$	$(2^{12} + 1024) * T_{WDT}$	409.6 ms ~ 512 ms
101	$2^{14} * T_{WDT}$	$(2^{14} + 1024) * T_{WDT}$	1.6384 s ~ 1.7408 s
110	$2^{16} * T_{WDT}$	$(2^{16} + 1024) * T_{WDT}$	6.5536 s ~ 6.656 s
111	$2^{18} * T_{WDT}$	$(2^{18} + 1024) * T_{WDT}$	26.2144 s ~ 26.3168 s

■ WTE

✧ 控制看門狗定時器的啟用或禁用

[7]	WTE	**看门狗定时器使能**（写保护位） 0 = 禁用看门狗定时器（该动作将重置内部计数器） 1 = 使能看门狗定时器

■ WTIE, WTIF

✧ **WTIE**

✓ 控制是否允許看門狗中斷

✓ 用途：啟用時，WDT 在計數器達到溢出時間時觸發中斷，給程式提供修復的機會。

如果 WTIE 沒有啟用

1. 看門狗溢出時，不會產生中斷信號。
2. 系統無法利用中斷進行修復，直接進入延遲期並等待可能的復位。
3. 適合那些希望直接 reset 的場景，但可能降低對異常的容錯處理能力。

[6]	WTIE	**看门狗中断使能**（写保护位） 0 = 禁用看门狗中断 1 = 使能看门狗中断

- ✧ **WTIF**

  中斷是否已經發生的狀態標誌。像是這次 lab 中的 WDT_CLEAR_TIMEOUT_INT_FLAG()就是將 WTIF 清除，這邊後面會再提到。

[3]	**WTIF**	**看门狗定时器中断标志** 如果看门狗定时器中断使能，该位置位表示看门狗定时器中断发生。 0 = 看门狗定时器中断没有发生。 1 = 看门狗定时器中断发生。 注：该位写 1清除。

- ■ **WTRF, WTRE**
  - ✧ **WTRF -** 看門狗是否觸發了系統復位(是否有 reset)
  - ✧ **WTRE -** 控制是否允許 WDT 執行系統復位(如果只需要中斷而不希望觸發復位，可以禁用該位)

[2]	**WTRF**	**看门狗定时器复位标志位** 当看门狗定时器溢出引发复位，硬件将置位该位，通过读取该位可以确认复位是否由看门狗引起。该位通过写1手动清除，如果 **WTRE** 禁用，看门狗定时器溢出对该位无影响。 0 = 看门狗定时器服务没发生。 1 = 看门狗定时器复位发生。 注：该位写 1 清除。
[1]	**WTRE**	**看门狗定时器复位使能**（写保护位） 置该位将使能看门狗定时器复位功能。 0 = 禁用看门狗定时器复位功能。 1 = 使能看门狗定时器复位功能。

- ■ **WTR**

  用於重置看門狗計數器，定期執行此操作可防止看門狗超時並觸發中斷或復位

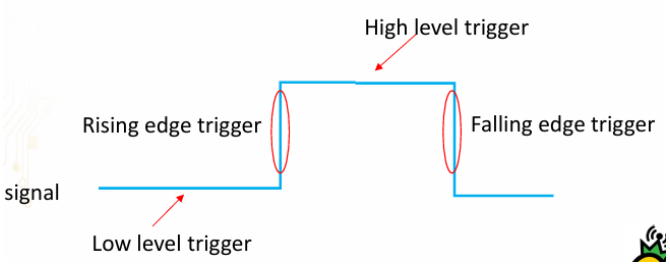
[0]	**WTR**	**清看门狗定时器**（写保护位） 置该位将清除看门狗定时器。 0 = 写 0到该位无效果。 1 = 重置看门狗定时器。 注：该位将被硬件自动清除。

✓ GPIO

- ■ Interrupt

  本次 lab 的的按鍵是用 GPIO 的中斷去觸發的，這邊會先介紹 GPIO 的中斷模式，包括「電平觸發」和「邊緣觸發」兩種方式。

  1. 電平觸發（Level Trigger）
     - ✧ 低電平觸發（Low Level Trigger）：當訊號保持在低電平（0）時，會觸發中斷
     - ✧ 高電平觸發（High Level Trigger）：當訊號保持在高電平（1）時，會觸發中斷。
  2. 邊緣觸發（Edge Trigger）
     - ✧ 下降邊觸發（Falling Edge Trigger）：當訊號從高電平變為低電平時，會觸發中斷。
     - ✧ 上升邊觸發（Rising Edge Trigger）：當訊號從低電平變為高電平時，會觸發中斷。



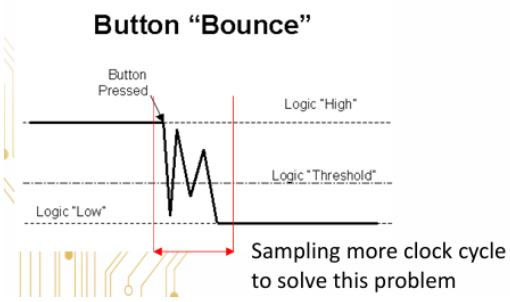
- de-bounce

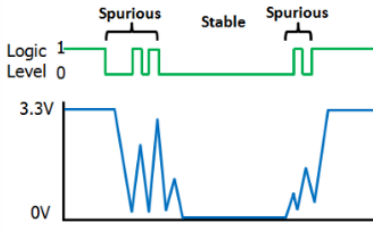
  為了避免按鍵彈跳導致按一次卻使 safe-alarm 一直切換，所以要使用 de-bounce

  ✧ 什麼是按鍵彈跳

  當按下或釋放按鈕時，機械式按鈕的觸點會因為物理震動而導致電壓在短時間內快速地上下波動，這被稱為「**彈跳現象**」。

  - 當按鈕被按下時，訊號並非直接穩定到 high，而是經歷了一段反覆震盪的過程。
  - 訊號會多次穿越 Logic Threshold，造成不穩定的輸入。





  ✧ 如果直接使用 GPIO 中斷來偵測按鈕的狀態，這些短暫的電壓波動會被誤判為多次有效中斷，這樣的行為會導致系統觸發中斷的次數遠超過預期。

  ✧ **硬體除抖（Hardware Debouncing）**

  本次 lab 是使用硬體除抖的方式來防止按鈕跳動

  - GPIO_SET_DEBOUNCE_TIME(GPIO_DBCLKSRC_LIRC, GPIO_DBCLKSEL_1024);

    1. GPIO_DBCLKSRC_LIRC

       選擇除抖的時鐘來源為低頻時鐘（LIRC，Low Internal RC Oscillator）。

    2. GPIO_DBCLKSEL_1024

       設置除抖的時間間隔為時鐘的 1024 個週期。當訊號穩定超過這段時間，才會被視為有效信號。

```
GPIO_SET_DEBOUNCE_TIME(GPIO_DBCLKSRC_LIRC, GPIO_DBCLKSEL_1024);
```

  - GPIO_ENABLE_DEBOUNCE(PA, BIT3 | BIT4 | BIT5);

    PA3、PA4 、PA5 啟用除抖功能

```
GPIO_ENABLE_DEBOUNCE(PA, BIT3 | BIT4 | BIT5);
```

<Mian function code>

這邊我會先介紹 watch dog timer 的一些重要 function 後續再說明程式流程

1. SYS_UnlockReg()

解鎖受保護的寄存器，允許對系統控制暫存器進行修改，NUC140 對重要 register（如 **WTCR**）實施寫保護機制，防止意外修改。所以我們要用 SYS_UnlockReg()解鎖保護暫存器，使其暫時允許修改。

2. WDT_Open(WDT_TIMEOUT_2POW14, WDT_RESET_DELAY_1026CLK, TRUE, FALSE);

   ✧ **u32TimeoutInterval** :設置看門狗定時器的溢出時間間隔，寫入 WTIS 暫存器

   WDT_TIMEOUT_2POW14: 設置看門狗的超時時間為 2^14 個時鐘週期

   ✧ **u32ResetDelay**: 配置看門狗重置前的延遲時間

   WDT_RESET_DELAY_1026CLK: 設置超時後延遲 1024 個時鐘週期再觸發重置

   ✧ **u32EnableReset** : 設置是否啟用看門狗超時後的系統重置功能

   ✧ **u32EnableWakeup** : 設置看門狗是否能在低功耗模式下喚醒 CPU

```
void WDT_Open(uint32_t u32TimeoutInterval,
              uint32_t u32ResetDelay,
              uint32_t u32EnableReset,
              uint32_t u32EnableWakeup)
{
    WDT->WTCRALT = u32ResetDelay;

    WDT->WTCR = u32TimeoutInterval | WDT_WTCR_WTE_Msk |
                (u32EnableReset << WDT_WTCR_WTRE_Pos) |
                (u32EnableWakeup << WDT_WTCR_WTWKE_Pos);
    return;
}
```

3. WDT_EnableInt();

   ✧ 啟用看門狗中斷功能

   ✧ 設置 WTCR 中的中斷使能位（如 WTIE），當看門狗超時時，觸發中斷請求（IRQ）

```
static __INLINE void WDT_EnableInt(void)
{
    WDT->WTCR |= WDT_WTCR_WTIE_Msk;
    return;
}
```

4. NVIC_EnableIRQ(WDT_IRQn); -啟用看門狗的中斷服務

5. WDT_IRQHandler

   ✧ **if(WDT_GET_TIMEOUT_INT_FLAG() == 1)**

   WDT_GET_TIMEOUT_INT_FLAG() 用於檢查看門狗的超時中斷旗標（WTIF）。如果返回值為 1，表示看門狗超時中斷已觸發，需要處理。

```
#define WDT_GET_TIMEOUT_INT_FLAG()      ((WDT->WTCR & WDT_WTCR_WTIF_Msk)? 1 : 0)
```

   ✧ **WDT_CLEAR_TIMEOUT_INT_FLAG();** -清除看門狗的超時中斷旗標

   ✧ **WDT_RESET_COUNTER(); -重置看門狗計數器(餵狗)**

   ■ **WDT->WTCR & ~(WDT_WTCR_WTIF_Msk | WDT_WTCR_WTWKF_Msk | WDT_WTCR_WTRF_Msk)**

   使用位元運算清除以下旗標（設為 0）

   ✧ WDT_WTCR_WTIF_Msk（Timeout Interrupt Flag）：表示超時中斷已觸發

   ✧ WDT_WTCR_WTWKF_Msk（Wakeup Flag）：表示看門狗喚醒事件已觸發

- ✧ WDT_WTCR_WTRF_Msk（Reset Flag）：表示看門狗觸發了系統重置

■ **| WDT_WTCR_WTR_Msk**

- ✧ 重置計數器位（Watchdog Timer Reset）
- ✧ 將該位設置為 1，指示硬體重置看門狗計數器

```
#define WDT_RESET_COUNTER()          (WDT->WTCR = (WDT->WTCR & ~(WDT_WTCR_WTIF_Msk | WDT_WTCR_WTWKF_Msk | WDT_WTCR_WTRF_Msk)) | WDT_WTCR_WTR_Msk)
```

✧ **GPIO_statue? 重置看門狗計數器 ： 不動作**

根據目前狀態判斷是否要讓 WDT 重啟系統

```
void WDT_IRQHandler(void)
{
    if(WDT_GET_TIMEOUT_INT_FLAG() == 1)
    {
        /* Clear WDT time-out interrupt flag */
        WDT_CLEAR_TIMEOUT_INT_FLAG();

        printf("WDT time-out interrupt occurred.\n");
        if(GPIO_statue){
          WDT_RESET_COUNTER();
          printf("No problem!!!\n");
        }
        else{
          printf("Alarm!!!!~~~reset\n");
        }

    }
}
```

這邊我接著介紹 GPIO 的一些重要 function

1. 啟用 GPIO 中斷　　-PA3　-下沿觸發 -防抖動

```
//GPIO_EnableInt(PA, 3, GPIO_INT_RISING);
GPIO_EnableInt(PA, 3, GPIO_INT_FALLING);
NVIC_EnableIRQ(GPAB_IRQn);

GPIO_SET_DEBOUNCE_TIME(GPIO_DBCLKSRC_LIRC, GPIO_DBCLKSEL_1024);
GPIO_ENABLE_DEBOUNCE(PA, BIT3 | BIT4 | BIT5);
```

2. GPIO PA3 設為雙通道模式這樣 PA3 平時就會是 high， PA2 設為接地，當第一顆按鍵被按下時 PA3 就會是 low，這樣看 PA3 狀態就可以知道第一顆案件是否被按下

```
void OpenKeyPad(void)
{
  GPIO_SetMode(PA, BIT0, GPIO_PMD_QUASI);
  GPIO_SetMode(PA, BIT1, GPIO_PMD_QUASI);
  GPIO_SetMode(PA, BIT2, GPIO_PMD_QUASI);
  GPIO_SetMode(PA, BIT3, GPIO_PMD_QUASI);
  GPIO_SetMode(PA, BIT4, GPIO_PMD_QUASI);
  GPIO_SetMode(PA, BIT5, GPIO_PMD_QUASI);
  PA0=1; PA1=1; PA2=0; PA3=1; PA4=1; PA5=1;
}
```

3. GPAB_IRQHandler

   ✧ if(GPIO_GET_INT_FLAG(PA, BIT3))

   檢查 PA3 引腳是否設置了中斷旗標

   ✧ GPIO_CLR_INT_FLAG(PA, BIT3);

   清除 PA3 引腳的中斷旗標，表示這次中斷已被處理，防止重複觸發。

   ✧ 如果按鈕被按下就反轉狀態 GPIO_statue = !GPIO_statue;

```
void GPAB_IRQHandler(void)
{
    /* To check if PB.3 interrupt occurred */

    if(GPIO_GET_INT_FLAG(PA, BIT3))
    {
        GPIO_CLR_INT_FLAG(PA, BIT3);
        GPIO_statue = !GPIO_statue;
        printf("change!!!\n");
    }
    else
    {
        /* Un-expected interrupt. Just clear all PA, PB interrupts */
        PA->ISRC = PA->ISRC;
        PB->ISRC = PB->ISRC;
        printf("Un-expected interrupts.\n");
    }
}
```

整體程式流程

1. **按鍵與狀態切換**：

   ✧ 按下按鍵觸發 GPIO 中斷，反轉 GPIO_statue。

   ✧ GPIO_statue 的值模擬系統的正常或異常狀態。

2. **看門狗監控**：

   ✧ 若 GPIO_statue = TRUE，系統正常，看門狗被重置。

   ✧ 若 GPIO_statue = FALSE，看門狗進入超時並觸發中斷。

3. **看門狗的自動恢復**：

   ✧ 如果異常狀態持續，看門狗會觸發系統重置，恢復正常運行。

這樣就可以達成 Let emergency alarm can rese、Use WDT reset the emergency alarm when the state is not safe 這樣的要求了

<過程中遇到的困難>

在這次 Lab 的實作中，困難點在於對看門狗的理解不夠深入。一開始雖然知道它的功能是防止系統卡死，但在實際配置暫存器時，像超時設定（WTIS）、重置延遲（WTR）等細節完全不熟悉，這邊花了許多時間去了解，花費不少精力。

<心得與收穫>

在這次的 Watchdog 實驗中，看似簡單的系統監控功能，實際實作過程中卻讓我深刻體會到，要成功實現 Watchdog 的功能，需要對其運作原理及硬體配置有一定的掌握。例如，超時條件的設定（WTIS）、重置延遲的配置（WTR）、以及中斷處理邏輯等，每一個步驟都需要細緻地設計，才能讓 Watchdog 在系統中穩定地發揮其監控作用。

在實驗過程中，我學習到如何有效配置 Watchdog 的參數，包括選擇合適的超時時間、啟用中斷功能，以及正確處理中斷旗標以確保系統能正常運行。我理解了 Watchdog 中斷的核心作用，並掌握了如何在中斷服務例程中結合系統狀態（例如 GPIO 的輸入）來決定是否進行「餵狗」，從而避免系統被誤重置。此外，我還學會了如何處理 Watchdog 與其他模組（如 GPIO 按鍵中斷）的協作，讓系統狀態能夠準確反映實際情況，進一步提升系統的穩定性和可靠性。

同時，我也體會到在 Watchdog 的實作過程中，暫存器配置和中斷管理是非常關鍵的部分。從超時條件的設定，到中斷觸發後清除旗標，直到決定是否重置計數器，每一個操作都需要精確執行，否則可能會導致系統無法正確進行自動監控或頻繁地重置。這些細節讓我理解到，Watchdog 不僅僅是一個簡單的防護機制，更是系統可靠性設計中的核心模組之一。

這次實驗給了我寶貴的經驗，使我對嵌入式系統中的 Watchdog 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、可靠的系統監控功能。這些經驗不僅加強了我對 Watchdog 的理解，也增強了我在嵌入式系統開發中的實作能力，尤其是在面對系統穩定性與錯誤恢復需求時，能夠設計出更高效的解決方案。