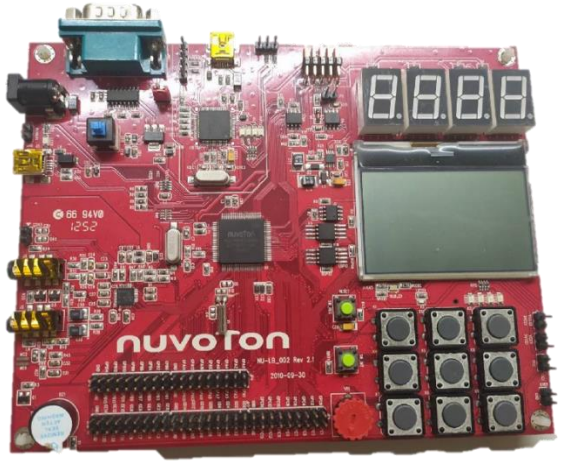
嵌入式作業系統 LAB 3

系所：通訊四　　學號 :409430030　　姓名:翁佳煌

<實驗器材及環境>

NUC 140 開發板　　FreeRTOSv10.4.1



<實驗過程與方法>

Basic:

首先下**圖 1** 先函式宣告和定義，vTaskMsgPro1 和 vTaskMsgPro2 是兩個用於創建 FreeRTOS 任務（tasks）的函式。第 30 行宣告一個全域變數 cnt，用來顯示經過的秒數。vStartThreadTasks 用來初始化和啟動這些任務。

```
27  static void vTaskMsgPro1(void* pvParameters);
28  static void vTaskMsgPro2(void* pvParameters);
29
30  int cnt=0;
31  void vStartThreadTasks( void );
32
33  /* Function prototype declaration */
34  void SYS_Init(void);
35  void UART0_Init(void);
36
```

▲圖 1

在來看到下**圖 2**，首先看到 129 行，Task2_vTimerCallback 是一個回呼（Callback）函式，用來在計時器觸發時印出訊息。

接來為 **main 函式:**

呼叫 vStartThreadTasks() 初始化並啟動 FreeRTOS 的任務。

使用 vTaskStartScheduler() 開始 FreeRTOS 調度器。程式進入無窮迴圈，等待 FreeRTOS 調度器運行。

155~159 行 **vStartThreadTasks 函式:**

使用 xTaskCreate 函式建立兩個 FreeRTOS 任務 vTaskMsgPro1 和 vTaskMsgPro2。

161~170 行 **vTaskMsgPro1 函式:**

vTaskMsgPro1 是一個無窮迴圈的任務，每秒輸出一次訊息，顯示 cnt 變數的值。

vTaskDelay(1000) 使該任務每次執行後延遲 1000 個 FreeRTOS 的 tick，達到每秒執行一次的效果。

172~188 行 **vTaskMsgPro2 函式:**

vTaskMsgPro2 是另一個無窮迴圈的任務。

此任務內部檢查一個靜態的計時器 xTimer2 是否為 NULL，如果是則建立一個計時器，並開始它。該計時器每 3000 個 ticks 周期性地觸發一次，觸發時調用第 129 行的 Task2_vTimerCallback 函式。

```
129  void Task2_vTimerCallback(){
130          printf("%d second. I'm Timer. OhYa!\n",cnt);
131  }
132
133  int main(void)
134  {
135      /* Unlock protected registers */
136      SYS_UnlockReg();
137      /* Init system, IP clock and multi-function I/O. */
138      SYS_Init();
139      /* Lock protected registers */
140      SYS_LockReg();
141
142          GPIO_Init();  // LED initial
143
144          UART0_Init();
145
146          printf("----------------LAB3----------------\r\n");
147          vStartThreadTasks();
148          vTaskStartScheduler();
149
150          while(1);
151
152  }
```

```
155  void vStartThreadTasks( void )
156  {
157          xTaskCreate(vTaskMsgPro1, "vTaskMsgPro1",128,NULL,2,( xTaskHandle * ) NULL );  //pxTaskCode , pcName,usStackD
158          xTaskCreate(vTaskMsgPro2, "vTaskMsgPro2",128,NULL,1,( xTaskHandle * ) NULL );
159  }
160
161  static void vTaskMsgPro1(void* pvParameters)
162  {
163          while(1)
164          {
165                  //Task1 count per second (use vTaskDelay)
166                  printf("%d second. I'm Delay\n",cnt);
167                  cnt++;
168                  vTaskDelay(1000);
169          }
170  }
171
172  static void vTaskMsgPro2(void* pvParameters)
173  {
174          static TimerHandle_t xTimer2 = NULL;
175          while(1)
176          {
177                  if(xTimer2==NULL){
178                       xTimer2 = xTimerCreate("Timer2", pdMS_TO_TICKS(3000), pdTRUE, (void *)0, Task2_vTimerCallback);
179                       if(xTimer2!=NULL){
180                              if(xTimerStart(xTimer2,0)==pdPASS){
181                                      printf("Timer2 start\n");
182                              }else{
183                                      printf("Failed to start timer2\n");
184                              }
185                       }
186                  }
187          }
188  }
189
```

▲圖 2

Bonus:

Bonus 的部分需要用到 keypad 來啟用兩個不同時間的 Timer。

首先看到 main 函式，配置和啟用中斷，將 PA（Port A）的引腳（BIT0 到 BIT5）設定為準雙向模式（Quasi-bidirection mode）。使用 GPIO_EnableInt 將 PA0、PA1 和 PA2 設定為下降沿觸發的中斷。設定 PA3 為低電位，PA4 和 PA5 為高電位。啟用中斷的 Debounce 功能。要這麼做是因為當設定中斷時，特別是在使用物理按鈕或開關時，可能會出現彈跳現象。按下或釋放按鈕時，接點之間可能會產生短暫的連續接觸和斷開，這會造成微小的電壓波動，導致系統誤判多次觸發中斷。為了解決這個問題，使用中斷時的「Debounce」機制可以有效降低或消除這種誤判。它會在中斷觸發前等待一小段時間，確保穩定的信號狀態已經建立，避免短暫的不穩定狀態被誤判為真正的觸發事件。

呼叫 vStartThreadTasks() 初始化並啟動 FreeRTOS 的任務。

使用 vTaskStartScheduler() 啟動 FreeRTOS 調度器。

```c
int main(void)
{
    /* Unlock protected registers */
    SYS_UnlockReg();
    /* Init system, IP clock and multi-function I/O. */
    SYS_Init();
    /* Lock protected registers */
    SYS_LockReg();

        GPIO_Init();  // LED initial

        UART0_Init();

            /*  Configure PE.5 as Quasi-bidirection mode and enable interrupt by falling edge trigger */
        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);

        GPIO_EnableInt(PA, 0, GPIO_INT_FALLING);
        GPIO_EnableInt(PA, 1, GPIO_INT_FALLING);
        GPIO_EnableInt(PA, 2, GPIO_INT_FALLING);
        //NVIC_EnableIRQ(GPAB_IRQn);

        PA3=0; PA4=1; PA5=1;
        /* Enable interrupt de-bounce function and select de-bounce sampling cycle time is 1024 clocks of LIRC clock */
    GPIO_SET_DEBOUNCE_TIME(GPIO_DBCLKSRC_LIRC, GPIO_DBCLKSEL_1024);
    GPIO_ENABLE_DEBOUNCE(PA, BIT0 | BIT1 | BIT2);

        printf("----------------LAB3----------------\r\n");
        vStartThreadTasks();
        vTaskStartScheduler();

        while(1);

}
```

▲圖 3

下**圖 4** 為 vStartThreadTasks 的函數，建立三個任務：
vTaskMsgPro1：處理按下 PA2（key1）的中斷事件，並使用定時器進行相應操作。
vTaskMsgPro2：處理按下 PA1（key2）的中斷事件，也使用定時器進行相應操作。
vTaskCountTime：簡單的計時任務，每秒顯示一次計數值 cnt。

```c
void vStartThreadTasks( void )
{
        xTaskCreate(vTaskMsgPro1, "vTaskMsgPro1",128,NULL,1,( xTaskHandle * ) NULL );  //
        xTaskCreate(vTaskMsgPro2, "vTaskMsgPro2",128,NULL,1,( xTaskHandle * ) NULL );
        xTaskCreate(vTaskCountTime, "vTaskCountTime",128,NULL,1,( xTaskHandle * ) NULL );
}
```

▲圖 4

再來看到**圖 5** 的 vTaskCountime，它只負責印出目前時間並印出。

```c
static void vTaskCountTime(void* pvParameters){
        while(1)
        {
                //Task1 count per second (use vTaskDelay)
                printf("%d second. I'm Delay\n",cnt);
                cnt++;
                vTaskDelay(1000);
        }
}
```

▲圖 5

接下來看到下**圖 6**，Task2_vTimerCallback 和 Task1_vTimerCallback，它們是 FreeRTOS 定時器的回條函式，用於處理定時器到期時的操作。

當 xTimer2 的定時器到期時，執行該回條函式。該函式會印出訊息 "Stop timer 2" 並停止 xTimer2 的定時器，然後將 pressed 變數設為 0。

當 xTimer1 的定時器到期時，執行該回條函式。該函式會印出訊息 "Stop timer 1"。停止 xTimer1 的定時器。將 pressedkey1 變數設為 0。

```
143  void Task2_vTimerCallback(){
144          printf("Stop timer 2\n");
145          //printf("%d second. I'm Timer. OhYa!\n",cnt);
146          xTimerStop(xTimer2,portMAX_DELAY);
147          pressed=0;
148  }
149
150  void Task1_vTimerCallback(){
151          printf("Stop timer 1\n");
152          xTimerStop(xTimer1,portMAX_DELAY);
153          pressedkey1=0;
154  }
```

▲圖 6

接著看到下**圖 7**，vTaskMsgPro1，其功能是處理按下 PA2（key1）的中斷事件。
主要監聽 PA2（key1）的中斷事件，使用 GPIO_GET_INT_FLAG 來檢查是否發生中斷事件。

如果中斷事件發生，立即清除中斷旗標（GPIO_CLR_INT_FLAG(PA, BIT2)），表示已處理該中斷。
如果定時器 xTimer1 尚未創建，則建立一個 5 秒長的定時器。
如果按鍵 key1 被按下（pressedkey1 == 0），則啟動定時器 xTimer1。如果按鍵 key1 已經被按下（pressedkey1 == 1），則停止定時器 xTimer1，並輸出相應的訊息。

簡單來說，這個任務的設計主要是在按下按鍵 key1 時啟動一個定時器，再次按下時則停止該定時器。這種模式可以用於在按鍵按下時執行某些操作，並在按鍵釋放時停止相關的計時器。

```
static void vTaskMsgPro1(void* pvParameters)
{
        while(1){
                if(GPIO_GET_INT_FLAG(PA, BIT2)) //key1
                {
                    //printf("i am key1~~~\n");
                    GPIO_CLR_INT_FLAG(PA, BIT2);

                    if(xTimer1==NULL){
                            xTimer1 = xTimerCreate("Timer1", pdMS_TO_TICKS(5000), pdTRUE, (void *)0, Task1_vTimerCallback);
                    }

                    if(pressedkey1==0){
                            pressedkey1=1;
                            //xTimerChangePeriod(xTimer1, pdMS_TO_TICKS(5000),0);

                        if(xTimer1!=NULL){
                                if(xTimerStart(xTimer1,0)==pdPASS){
                                        printf("Timer1 start\n");
                                }else{
                                        printf("Failed to start timer1\n");
                                }
                        }
                    }else{
                            if(pressedkey1){
                                //stop timer;
                                printf("You pressed to stop Timer1 !!!\n");
                                xTimerStop(xTimer1,portMAX_DELAY);
                                pressedkey1=0;
                            }
                        }
                }
            }
}
```

▲圖 7

下**圖 8** 這段程式碼是任務 vTaskMsgPro2，與之前的任務 vTaskMsgPro1 類似，用於處理按下 PA1（key2）的中斷事件。

負責監聽 PA1（key2）的中斷事件，使用 GPIO_GET_INT_FLAG 檢查是否發生了中斷。如果中斷事件發生，立即清除中斷旗標（GPIO_CLR_INT_FLAG(PA, BIT1)），表示已處理該中斷。

如果定時器 xTimer2 尚未創建，則建立一個週期為 10 秒的定時器。
如果按鍵 key2 被按下（pressed == 0），則啟動定時器 xTimer2。
如果按鍵 key2 已經被按下（pressed == 1），則停止定時器 xTimer2，並輸出相應的訊息。

這個任務的設計與 vTaskMsgPro1 類似，都是在按下按鍵時啟動一個定時器，再次按下時則停止該定時器。

```
static void vTaskMsgPro2(void* pvParameters)
{
        while(1)
        {
                if(GPIO_GET_INT_FLAG(PA, BIT1))     //key2
                {
                        //printf("i am key2~~~\n");
                        GPIO_CLR_INT_FLAG(PA, BIT1);

                        if(xTimer2==NULL){
                                xTimer2 = xTimerCreate("Timer2", pdMS_TO_TICKS(10000), pdTRUE, (void *)0, Task2_vTimerCallback);
                        }

                        if(pressed==0){
                            pressed=1;
                            //xTimerChangePeriod(xTimer2, pdMS_TO_TICKS(10000),0);
                             if(xTimer2!=NULL){
                                    if(xTimerStart(xTimer2,0)==pdPASS){
                                            printf("Timer2 start\n");
                                    }else{
                                            printf("Failed to start timer2\n");
                                    }
                             }
                        }else{
                                if(pressed){
                                //stop timer;
                                printf("You pressed to stop Timer2 !!!\n");
                                xTimerStop(xTimer2,portMAX_DELAY);
                                pressed=0;
                            }
                        }
                }
        }
}
```

▲圖 8

<心得與收穫>

這次 LAB 的實驗剛好是我 FreeRTOS 手冊報告的部分，整體實作下來比較沒有遇到太大的問題，此外，這次實驗讓我更加理解到軟體定時器在嵌入式系統中的強大，它能夠有效地管理時間相關等任務，並且是開發系統的重要元素之一。這些任務在確保系統準時執行特定操作方面提供了靈活性和控制權。