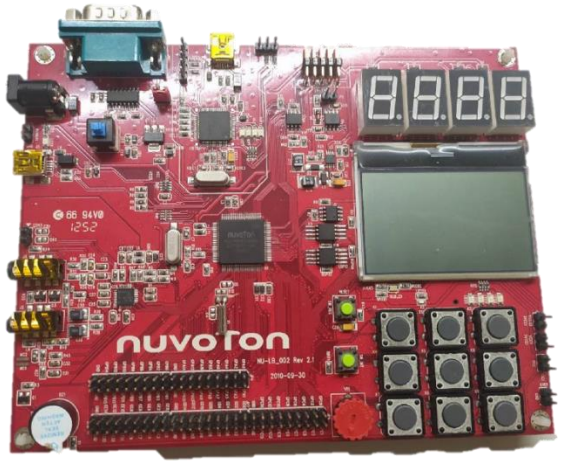
# 嵌入式作業系統 LAB 5

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>

下圖 1 首先要記得 include 第 23 行的 event_group.h，這次 LAB 將會使用到這個函式庫，而其餘則和之前的 LAB 實驗相同。

第 30~33 行，MODE1_BIT、MODE2_BIT、MODE3_BIT、MODE4_BIT: 定義了事件群組（xEventGroup）中不同模式或標誌的位元表示。例如，MODE1_BIT 代表位元 0，MODE2_BIT 代表位元 1，MODE3_BIT 代表位元 3，MODE4_BIT 代表位元 4。

```
17  #include "FreeRTOS.h"
18  #include "task.h"
19  #include "queue.h"
20  #include "croutine.h"
21  #include "timers.h"
22  #include "semphr.h"
23  #include "event_groups.h"
24  #include "string.h"
25
26  #define PLL_CLOCK           50000000
27
28  #define COMMAND_QUEUE_LENGTH 20
29
30  #define MODE1_BIT (1 << 0)
31  #define MODE2_BIT (1 << 1)
32  #define MODE3_BIT (1 << 3)
33  #define MODE4_BIT (1 << 4)
34
35  static void vTaskA(void* pvParameters);
36  static void vTaskB(void* pvParameters);
37  static void vTaskC(void* pvParameters);
38
39  QueueHandle_t xCommandQueue;
40  EventGroupHandle_t xEventGroup;
41
42  void vStartThreadTasks( void );
43
44  /* Function prototype declaration */
45  void SYS_Init(void);
46  void UART0_Init(void);
47
```

▲圖 1

下圖 2，147~213 行的函式在前幾此 LAB 都有提到，因此這裡不多加贅述，值得注意的是 220 行的函式，當呼叫 All_Bits_Reset() 函數時，它會使用 xEventGroupClearBits() 函數來清除事件群組中相應位元的值，將其設置為零。這可以在需要重置或清除所有模式或標誌的位元時使用，以確保它們處於初始狀態，不再處於任何已設置的狀態。

```
147  void blink_left_to_right(int freq){
173
174  void blink_right_to_left(int freq){
200
201
202  void blink_all(int freq){
212
213  void turn_RGBLED_off(){
219
220  void All_Bits_Reset(){
221      xEventGroupClearBits(xEventGroup, MODE1_BIT);
222      xEventGroupClearBits(xEventGroup, MODE2_BIT);
223      xEventGroupClearBits(xEventGroup, MODE3_BIT);
224      xEventGroupClearBits(xEventGroup, MODE4_BIT);
225
226  }
```

▲圖 2

下圖 3 為 main 函數，xEventGroup = xEventGroupCreate(); 創建一個事件群組 xEventGroup。vStartThreadTasks(); 呼叫函數 vStartThreadTasks()，用於創建並啟動 FreeRTOS 任務。vTaskStartScheduler(); 開始 FreeRTOS 任務調度器。

```
229  int main(void)
230  {
231      /* Unlock protected registers */
232      SYS_UnlockReg();
233      /* Init system, IP clock and multi-function I/O. */
234      SYS_Init();
235      /* Lock protected registers */
236      SYS_LockReg();
237          GPIO_Init();  // LED initial
238          UART0_Init();
239
240          printf("\n\n----------------LAB5-Basic----------------\r\n");
241          xCommandQueue = xQueueCreate(COMMAND_QUEUE_LENGTH,sizeof(char[10]));
242          xEventGroup = xEventGroupCreate();
243
244          vStartThreadTasks();
245          vTaskStartScheduler();
246
247          while(1);
248
249  }
250
251
252  void vStartThreadTasks( void )
253  {
254          xTaskCreate(vTaskA, "vTaskA",128,NULL,1,( xTaskHandle * ) NULL );  //px
255          xTaskCreate(vTaskB, "vTaskB",128,NULL,1,( xTaskHandle * ) NULL );
256          xTaskCreate(vTaskC, "vTaskC",128,NULL,1,( xTaskHandle * ) NULL );
257  }
```

▲圖 3

下圖 4 這個 vTaskA 是一個 FreeRTOS 任務，負責處理命令的輸入、解析和執行相應的操作。

char command[10];：建立一個長度為 10 的字元陣列，用來存放用戶輸入的命令。

主循環：

如果用戶輸入的命令是 "mode1"，則呼叫 xEventGroupSetBits(xEventGroup, MODE1_BIT); 函數，設置事件群組 xEventGroup 中的 MODE1_BIT 位元。

如果命令是 "mode2"、"mode3" 或 "mode4"，分別設置對應的事件群組位元。

如果命令不符合任何已知命令，則顯示 "Invalid command!" 訊息，並呼叫 turn_RGBLED_off(); 函數。

此函數將持續等待用戶輸入命令，解析並根據命令執行相應的操作。如果命令是已知的 "mode1"、"mode2"、"mode3" 或 "mode4"，則會設置相應的事件群組位元，否則會印出無效命令並執行特定操作。

```
259   static void vTaskA(void* pvParameters)
260   {
261       char command[10];
262
263       while (1) {
264           printf("Enter a command: ");
265           scanf("%s", command);
266                   printf("\nYour command is : %s\n",command);
267                   All_Bits_Reset();
268
269           if (strcmp(command, "mode1") == 0) {
270               xEventGroupSetBits(xEventGroup, MODE1_BIT);
271           } else if (strcmp(command, "mode2") == 0) {
272               xEventGroupSetBits(xEventGroup, MODE2_BIT);
273           } else if (strcmp(command, "mode3") ==0){
274                           xEventGroupSetBits(xEventGroup, MODE3_BIT);
275                   } else if (strcmp(command, "mode4") ==0){
276                           xEventGroupSetBits(xEventGroup, MODE4_BIT);
277                   } else {
278               printf("Invalid command!\n");
279                           turn_RGBLED_off();
280           }
281       }
282   }
```

▲圖 4

下圖 5 這個函數 vTaskB 是另一個 FreeRTOS 任務，負責根據事件群組中的位元狀態執行相應的操作。

EventBits_t bits = xEventGroupWaitBits(xEventGroup, MODE1_BIT | MODE2_BIT | MODE3_BIT | MODE4_BIT, pdFALSE, pdFALSE, portMAX_DELAY);：它會等待事件群組 xEventGroup 中的指定位元，其中 portMAX_DELAY 表示等待時間無限期，直到有指定的位元被設置。

如果 MODE1_BIT 位元被設置，則調用 blink_left_to_right(90); 函數來執行 LED 向右閃爍的動作。

如果 MODE2_BIT 位元被設置，則調用 blink_right_to_left(90); 函數來執行 LED 向左閃爍的動作。

如果 MODE3_BIT 位元被設置，則將所有 LED 持續亮起。

如果 MODE4_BIT 位元被設置，則調用 blink_all(90); 函數來執行所有 LED 一起閃爍的動作。

此任務持續監視事件群組中的位元狀態，並根據不同的狀態執行相應的操作。

```
static void vTaskB(void* pvParameters)
{
        while(1)
        {
                EventBits_t bits =
                xEventGroupWaitBits(xEventGroup, MODE1_BIT | MODE2_BIT | MODE3_BIT | MODE4_BIT , pdFALSE , pdFALSE , portMAX_DELAY);
              //( const EventGroupHandle_t xEventGroup,const EventBits_t uxBitsToWaitFor,  const BaseType_t xClearOnExit,  const BaseTy

                if ((bits & MODE1_BIT) != 0) {
                        blink_left_to_right(90);
                        //xEventGroupClearBits(xEventGroup, MODE1_BIT);
        }
                else if((bits & MODE2_BIT) != 0){
                        blink_right_to_left(90);
                }
                else if((bits & MODE3_BIT) != 0){
                        PC12=0;
                        PC13=0;
                        PC14=0;
                        PC15=0;
                }
                else if((bits & MODE4_BIT) != 0){
                        blink_all(90);
                }
        }
}
```

▲圖 5

下圖 6 這個函數 vTaskC 是另一個 FreeRTOS 任務，與前面的 TaskB 相似，根據事件群組中的位元狀態執行相應的操作。

EventBits_t bits = xEventGroupWaitBits(xEventGroup, MODE1_BIT | MODE2_BIT | MODE3_BIT | MODE4_BIT, pdFALSE, pdFALSE, portMAX_DELAY);：等待事件群組 xEventGroup 中的指定位元，portMAX_DELAY 表示等待時間無限期，直到有指定的位元被設置。

如果 MODE1_BIT 位元被設置，則將 PA14 設置為 0。

如果 MODE2_BIT 位元被設置，則將 PA13 設置為 0

如果 MODE3_BIT 位元被設置，則將 PA12 設置為 0。

如果 MODE4_BIT 位元被設置，則同時將 PA12、PA13 和 PA14 設置為 0。

此任務持續監視事件群組中的位元狀態，並根據這些位元的狀態執行不同的操作。這些操作是控制特定的 GPIO 腳位，從而影響硬體的行為或狀態。

```
static void vTaskC(void* pvParameters)
{
        while(1)
        {
                EventBits_t bits =
                xEventGroupWaitBits(xEventGroup, MODE1_BIT | MODE2_BIT | MODE3_BIT | MODE4_BIT , pdFALSE , pdFALSE , portMAX_DELAY);
                if ((bits & MODE1_BIT) != 0) {
                        PA14=0;
                        //xEventGroupClearBits(xEventGroup, MODE1_BIT);
        }
                else if((bits & MODE2_BIT) != 0){
                        PA13=0;
                }
                else if((bits & MODE3_BIT) != 0){
                        PA12=0;
                }
                else if((bits & MODE4_BIT) != 0){
                        PA12=0;
                        PA13=0;
                        PA14=0;
                }
        }
}
```

▲圖 6

## <心得與收穫>

這次 LAB5 是最後一次嵌入式系統這堂課的實作作業，我從中學習到 Event Group 的用法以及他的好處，通過 Event Group，你能夠將不同的事件綁定在一起，以便任務在某些特定事件發生時進行同步操作。這種機制為嵌入式系統提供了彈性和準確性，同時也提高了系統的效率和可靠性。

最後謝謝助教以及老師這學期的認真教學和付出，讓我收穫良多。