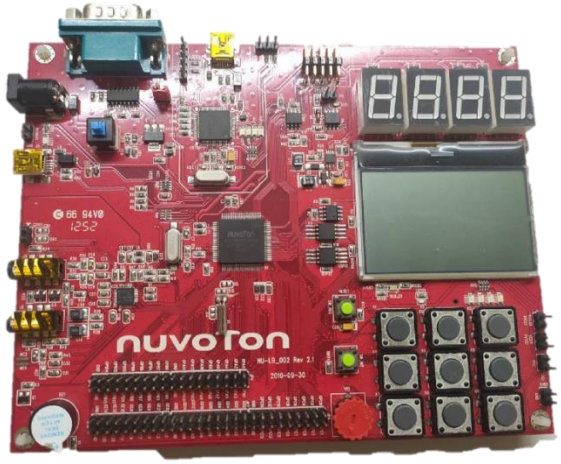
嵌入式作業系統 LAB 2

系所：通訊四　　學號 :409430030　　姓名:翁佳煌

<實驗器材及環境>

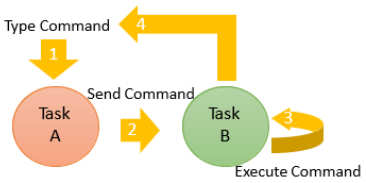
NUC 140 開發板



FreeRTOSv10.4.1

<實驗過程與方法>

Basic:



首先根據下圖 1，

35~36 行:定義了兩個任務：TaskA 和 TaskB。

43~44 行:宣告了 SYS_Init 和 UART0_Init 的函數，負責初始化系統和 UART。

48~51 行:定義了一個名為 Command_t 的結構，其中包含一個大小為 50 的字元陣列 (command[50])。這個結構用在存儲命令字串。

54~55 行:宣告了兩個 queue，分別為 xCommandQueueA 和 xCommandQueueB。這些隊列的類型是 QueueHandle_t，這是一個指向 FreeRTOS 隊列的句柄。但在 basic 中，只會用到 xCommandQueueA，其中 TaskA 負責發送命令到這個 queue 裡面，而 TaskB 負責讀取這個 queue 並做出相應的動作。

64~79 行:負責初始化對應的 GPIO，有 LEDs、RGB LED、Buzzer。

```
34
35  static void TaskA(void* pvParameters);
36  static void TaskB(void* pvParameters);
37
38
39  void vStartThreadTasks( void );
40
41
42  /* Function prototype declaration */
43  void SYS_Init(void);
44  void UART0_Init(void);
45
46  /*------------------LAB2------------------*/
47  // Define command structure
48  typedef struct {
49      char command[50]; // Store the command string
50
51  } Command_t;
52
53
54  QueueHandle_t xCommandQueueA; // Task A puts commands here
55  QueueHandle_t xCommandQueueB; // Task B reads and executes commands
56  /*------------------LAB2------------------*/
```

```
64  void GPIO_Init()
65  {
66          //INIT LEDS
67          GPIO_SetMode(PC, BIT12, GPIO_PMD_OUTPUT);
68          GPIO_SetMode(PC, BIT13, GPIO_PMD_OUTPUT);
69          GPIO_SetMode(PC, BIT14, GPIO_PMD_OUTPUT);
70          GPIO_SetMode(PC, BIT15, GPIO_PMD_OUTPUT);
71
72          //INIT RGB LED
73          GPIO_SetMode(PA,BIT12,GPIO_PMD_OUTPUT);
74          GPIO_SetMode(PA,BIT13,GPIO_PMD_OUTPUT);
75          GPIO_SetMode(PA,BIT14,GPIO_PMD_OUTPUT);
76
77          //Buzzer
78          GPIO_SetMode(PB,BIT11,GPIO_PMD_OUTPUT);
79  }
80
```

▲圖 1

再來看到下圖 2 中的 main 函數。

xCommandQueueA 和 xCommandQueueB，使用 xQueueCreate 函數建立兩個隊列。這些隊列用於任務之間的溝通，可以用於在不同的任務中傳遞命令。

vStartThreadTasks()如下圖 3，啟動任務的函數。包括創建和啟動在前文提到的 TaskA 和 TaskB。

vTaskStartScheduler()啟動 FreeRTOS 任務調度器，開始執行定義的任務。

```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();  // LEDs and RGB LEDs initial
        UART0_Init();

        xCommandQueueA = xQueueCreate(COMMAND_QUEUE_LENGTH, sizeof(Command_t));
        xCommandQueueB = xQueueCreate(COMMAND_QUEUE_LENGTH, sizeof(Command_t));

        printf("\n\n---LAB2 Basic---\r\n");

        //UART_FunctionTest();

        vStartThreadTasks();
        vTaskStartScheduler();

        while(1);

}
```

▲圖 2

```c
void vStartThreadTasks( void )
{
        xTaskCreate(TaskA, "TaskA",128,NULL,1,( xTaskHandle * ) NULL );
        xTaskCreate(TaskB, "TaskB",128,NULL,1,( xTaskHandle * ) NULL );
}
```

▲圖 3

接下來看到下圖 4，TaskA 的部分，在 TaskA 函數中，程式進入一個無窮迴圈，等待使用者輸入指令。使用 scanf 函數讀取輸入的指令，將其複製到 cmd 結構的 command 成員中，然後透過 xQueueSend 函數將命令發送到 xCommandQueueA 佇列中。portMAX_DELAY 參數表示如果佇列已滿，則任務將等待直到有空間可用。

```c
static void TaskA(void* pvParameters)
{
        Command_t cmd;

        //size_t size = sizeof(Command_t);
        //printf("Size of Command_t structure: %zu bytes\n", size);

        while(1)
        {
            char input[50];
            printf("\n\nInput Command:");
            scanf("%s",input);
            printf("%s\n",input);

            strcpy(cmd.command, input);

          // Send the command to Task B
        xQueueSend(xCommandQueueA, &cmd, portMAX_DELAY);    //queue handle //a pointer to the item //waiting time

            vTaskDelay(5);
        }
}
```

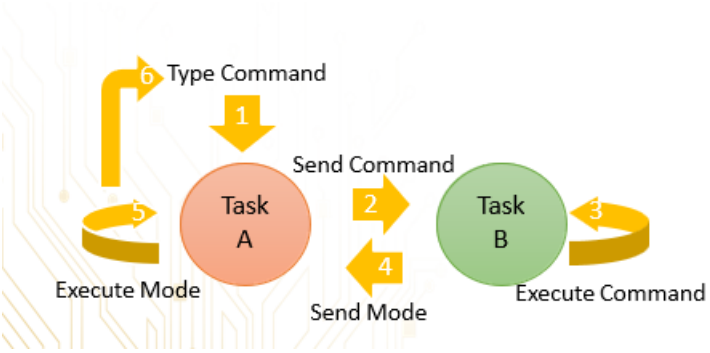
▲圖 4

再來是下圖 5，TaskB 的部分，在 TaskB 函數中，程式進入一個無窮迴圈，等待從 xCommandQueueA 佇列接收命令。使用 xQueueReceive 函數，如果成功接收到命令（pdPASS），則進行命令處理。在這裡，根據不同的命令，控制不同的外設，例如打開或關閉 LED、啟動或停止蜂鳴器、改變 RGB LED 的顏色等。如果接收到未知的命令，則輸出錯誤訊息。

```
294  static void TaskB(void* pvParameters)
295  {
296          Command_t cmd;
297          while(1)
298          {
299                  if (xQueueReceive(xCommandQueueA, &cmd, portMAX_DELAY) == pdPASS)
300          {
301                          // Process the command
302              if (strcmp(cmd.command, "LED0") == 0)
303              {
304                                  // Turn on LED0
305                  printf("LEDs turned ON\n");
306                                    PC12=0;
307                                    PC13=0;
308                                    PC14=0;
309                                    PC15=0;
310                          }
311                          else if(strcmp(cmd.command, "LED1") == 0)
312                          {
313                                 printf("LED turned OFF\n");
314                                    PC12=1;
315                                    PC13=1;
316                                    PC14=1;
317                                    PC15=1;
318                          }
319                          else if(strcmp(cmd.command,"BUZ0")==0)
320                          {
321                                    printf("buzzer turned ON\n");
322                                    PB11=0;
323                          }
324                          else if(strcmp(cmd.command,"BUZ1")==0)
325                          {
326                                    printf("buzzer turned ON\n");
327                                    PB11=1;
328                          }
329
330                          else if(strcmp(cmd.command,"BLU0")==0)
331                          {
332                                    printf("Change the color of RGB LED to Blue\n");
333                                    PA12=0;
334                          }
335                          else if(strcmp(cmd.command,"BLU1")==0)
336                          {
337                                    printf("Turn off RGB LED of Blue\n");
338                                    PA12=1;
339                          }
340
341                          else if(strcmp(cmd.command,"GRE0")==0)
342                          {
343                                    printf("Change the color of RGB LED to Green\n");
344                                    PA13=0;
345                          }
346                          else if(strcmp(cmd.command,"GRE1")==0)
347                          {
348                                    printf("Turn off RGB LED of Green\n");
349                                    PA13=1;
350                          }
351
352                          else if(strcmp(cmd.command,"RED0")==0)
353                          {
354                                    printf("Change the color of RGB LED to Red\n");
355                                    PA14=0;
356                          }
357                          else if(strcmp(cmd.command,"RED1")==0)
358                          {
359                                    printf("Turn off RGB LED of Red\n");
360                                    PA14=1;
361                          }
362                          else{
363                                    printf("Wrong command!");
364                          }
365                  }
366          }
367  }
```

▲圖 5

Bonus:



在來是 LAB2 Bonus 的部分，35~55 行絕大部分都與 Basic 大同小異，唯一差別是為了達到讓 LED 不斷閃爍的功能，只靠兩個 Task 是無法達成的，因為 TaskA 接收到 TaskB 後，如果要不斷閃爍 LED 的話，會導致整個 Input Command 卡死無法順利運行，因此只能另外建一個 TaskC 以及 queueC 來負責執行 LED 不斷閃爍的功能。

此外，後續初始化的部分都與 Basic 相同，就不再贅述。

```
34
35  static void TaskA(void* pvParameters);
36  static void TaskB(void* pvParameters);
37  static void TaskC(void* pvParameters);
38
39  void vStartThreadTasks( void );
40
41  /* Function prototype declaration */
42  void SYS_Init(void);
43  void UART0_Init(void);
44
45  /*------------------LAB2-----------------*/
46  // Define command structure
47  typedef struct {
48      char command[10]; // Store the command string
49  } Command_t;
50
51  int keep_blink=1;
52
53  QueueHandle_t xCommandQueueA; // Task A puts commands here
54  QueueHandle_t xCommandQueueB; // Task B reads and executes commands
55  QueueHandle_t xCommandQueueC;
56  /*------------------LAB2-----------------*/
```

▲圖 6

下圖 7 的部分是控制 LED 從左到右或是從右到左閃爍的函式，應用簡單的開關和延遲來達成，等等後面會使用到。

```
157 void blink_left_to_right(int freq){
158                                 PC12=1;
159                                 PC13=1;
160                                 PC14=1;
161                                 PC15=1;
162
163                                 PC12=0;
164                                 vTaskDelay(freq);
165                                 PC12=1;
166                                 vTaskDelay(freq);
167
168                                 PC13=0;
169                                 vTaskDelay(freq);
170                                 PC13=1;
171                                 vTaskDelay(freq);
172
173                                 PC14=0;
174                                 vTaskDelay(freq);
175                                 PC14=1;
176                                 vTaskDelay(freq);
177
178                                 PC15=0;
179                                 vTaskDelay(freq);
180                                 PC15=1;
181                                 vTaskDelay(freq);
182 }
183
```

```
184 void blink_right_to_left(int freq){
185                                 PC12=1;
186                                 PC13=1;
187                                 PC14=1;
188                                 PC15=1;
189
190                                 PC15=0;
191                                 vTaskDelay(freq);
192                                 PC15=1;
193                                 vTaskDelay(freq);
194
195                                 PC14=0;
196                                 vTaskDelay(freq);
197                                 PC14=1;
198                                 vTaskDelay(freq);
199
200                                 PC13=0;
201                                 vTaskDelay(freq);
202                                 PC13=1;
203                                 vTaskDelay(freq);
204
205                                 PC12=0;
206                                 vTaskDelay(freq);
207                                 PC12=1;
208                                 vTaskDelay(freq);
209 }
210
```

▲圖 7

下圖 8，212~216 行負責把 LED 顏色改成白色，217~221 行負責關掉所有 RGB_LED。

```
212 void change_color_to_white(int freq){
213     PA12 = 0; // Blue component
214     PA13 = 0; // Green component
215     PA14 = 0; // Red component
216 }
217 void turn_RGBLED_off(){
218     PA12 = 1; // Blue component
219     PA13 = 1; // Green component
220     PA14 = 1; // Red component
221 }
```

▲圖 8

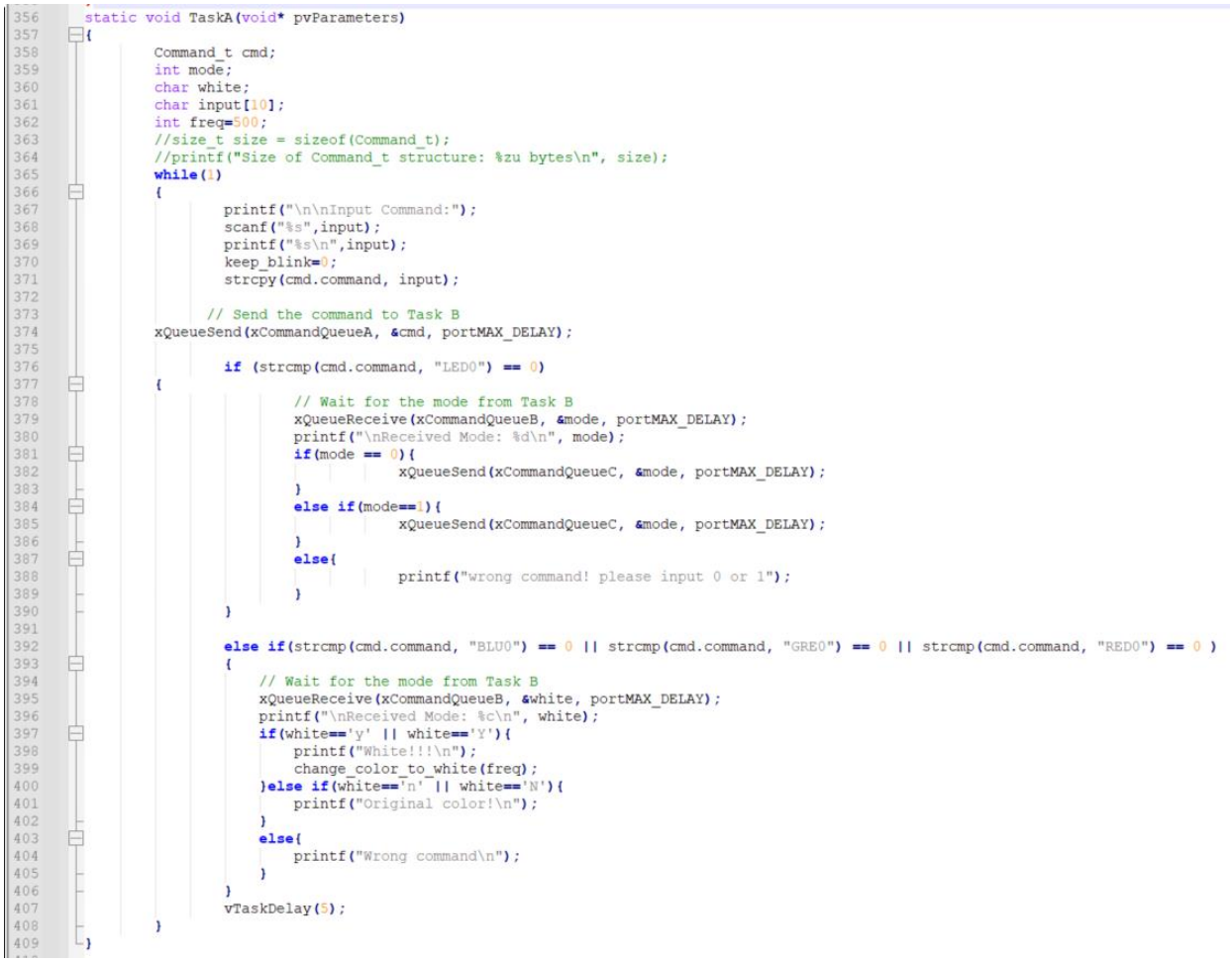
接下來看到下圖 9 的 main 函數，236~239 行創建三個不同的佇列 xCommandQueueA、xCommandQueueB 和 xCommandQueueC，用於在不同的任務間傳遞 Command_t 類型的命令。vStartThreadTasks()啟動任務。這可能是用來初始化並啟動 FreeRTOS 中的各種任務的函數。

vTaskStartScheduler()啟動 FreeRTOS 調度器，開始任務的執行。

```
224  int main(void)
225  {
226      /* Unlock protected registers */
227      SYS_UnlockReg();
228      /* Init system, IP clock and multi-function I/O. */
229      SYS_Init();
230      /* Lock protected registers */
231      SYS_LockReg();
232
233          GPIO_Init();  // LEDs and RGB LEDs initial
234          UART0_Init();
235
236          xCommandQueueA = xQueueCreate(COMMAND_QUEUE_LENGTH, sizeof(Command_t));
237          xCommandQueueB = xQueueCreate(COMMAND_QUEUE_LENGTH, sizeof(Command_t));
238          xCommandQueueC = xQueueCreate(COMMAND_QUEUE_LENGTH, sizeof(Command_t));
239          printf("\n\n---LAB2 Bonus---\r\n");
240
241          //UART_FunctionTest();
242
243          vStartThreadTasks();
244          vTaskStartScheduler();
245
246          while(1);
247
248  }
```

▲圖 9

看到下圖 10，負責接收使用者輸入的指令，然後通過佇列將該指令發送到 xCommandQueueA。根據不同的指令，它還會等待 Task B 發送的相關模式或參數到 xCommandQueueB，TaskA 在藉由 xQueueReceiveB 來做相對應的動作。另外，若是 LED0 的命令，則會進入到 376 行的判斷式，此時，會接收 queueB 裡面的 mode，在把 mode 傳到 QueueC 裡面，讓 TaskC 去讀取 queueC 來完成不斷閃爍 LED，這樣才不會導致 TaskA 無法進入下一輪 Input Command 的步驟。



```c
static void TaskA(void* pvParameters)
{
        Command_t cmd;
        int mode;
        char white;
        char input[10];
        int freq=500;
        //size_t size = sizeof(Command_t);
        //printf("Size of Command_t structure: %zu bytes\n", size);
        while(1)
        {
                printf("\n\nInput Command:");
                scanf("%s",input);
                printf("%s\n",input);
                keep_blink=0;
                strcpy(cmd.command, input);

              // Send the command to Task B
        xQueueSend(xCommandQueueA, &cmd, portMAX_DELAY);

                if (strcmp(cmd.command, "LED0") == 0)
        {
                        // Wait for the mode from Task B
                        xQueueReceive(xCommandQueueB, &mode, portMAX_DELAY);
                        printf("\nReceived Mode: %d\n", mode);
                        if(mode == 0){
                                xQueueSend(xCommandQueueC, &mode, portMAX_DELAY);
                        }
                        else if(mode==1){
                                xQueueSend(xCommandQueueC, &mode, portMAX_DELAY);
                        }
                        else{
                                printf("wrong command! please input 0 or 1");
                        }
                }

                else if(strcmp(cmd.command, "BLU0") == 0 || strcmp(cmd.command, "GRE0") == 0 || strcmp(cmd.command, "RED0") == 0 )
                {
                    // Wait for the mode from Task B
                    xQueueReceive(xCommandQueueB, &white, portMAX_DELAY);
                    printf("\nReceived Mode: %c\n", white);
                    if(white=='y' || white=='Y'){
                        printf("White!!!\n");
                        change_color_to_white(freq);
                    }else if(white=='n' || white=='N'){
                        printf("Original color!\n");
                    }
                    else{
                        printf("Wrong command\n");
                    }
                }
                vTaskDelay(5);
        }
}
```

▲圖 10

看到下圖 11，TaskB 的部分，負責處理從 xCommandQueueA 佇列接收到的命令。它根據不同的命令執行相應的操作，並且對於某些命令，它會等待從 TaskA 來的額外信息，並將回應發送到 xCommandQueueB 佇列。

變數宣告：

Command_t cmd：保存接收到的命令的結構。

int mode;：用於保存 LED 模式的整數變數。

char white：用於保存是否將 RGB LED 顏色更改為白色的字符變數。

接收命令：

if (xQueueReceive(xCommandQueueA, &cmd, portMAX_DELAY) == pdPASS)：從 xCommandQueueA 佇列接收命令。

處理不同的指令：

對於 "LED0"，它打開 LED，然後等待來自 TaskA 的模式信息（0 或 1），並將模式發送到 xCommandQueueB 佇列。

對於 "LED1"，它關閉 LED。

對於 "BUZ0" 和 "BUZ1"，它開啟或關閉蜂鳴器。

對於 "BLU0"、"GRE0"、"RED0"，它改變 RGB LED 的顏色並等待是否轉換為白色的信息，然後將信息發送到 xCommandQueueB 佇列。

對於 "BLU1"、"GRE1"、"RED1"，它關閉相應顏色的 RGB LED。

對於 "RGB1"，它將 RGB LED 關閉。

對於未知的指令，它輸出錯誤信息。

```
static void TaskB(void* pvParameters)
{
        Command_t cmd;
        int mode;
        char white;
        while(1)
        {
                if (xQueueReceive(xCommandQueueA, &cmd, portMAX_DELAY) == pdPASS)
        {
                    // Process the command
            if (strcmp(cmd.command, "LED0") == 0)
            {
                        // Turn on LED0
                printf("LEDs turned ON\n");
                            PC12=0;
                            PC13=0;
                            PC14=0;
                            PC15=0;
                            printf("LED Mode(0/1):");
                            scanf("%d",&mode);
                            xQueueSend(xCommandQueueB, &mode, portMAX_DELAY);
                        }
                        else if(strcmp(cmd.command, "LED1") == 0)
                        {
                              printf("LED turned OFF\n");
                                PC12=1;
                                PC13=1;
                                PC14=1;
                                PC15=1;
                        }
                        /*-----------------------------------------------------*/
                        else if(strcmp(cmd.command,"BUZ0")==0)
                        {
                                printf("buzzer turned ON\n");
                                PB11=0;
                        }
                        else if(strcmp(cmd.command,"BUZ1")==0)
                        {
                                printf("buzzer turned ON\n");
                                PB11=1;
                        }
                        /*-----------------------------------------------------*/
                        else if(strcmp(cmd.command,"BLU0")==0)
                        {
                                turn_RGBLED_off();
                                printf("Change the color of RGB LED to Blue\n");
                                PA12=0;

                                printf("Change to white or not?(y/n):");
                                scanf("%c",&white);
                                xQueueSend(xCommandQueueB, &white, portMAX_DELAY);
                        }
                        else if(strcmp(cmd.command,"BLU1")==0)
                        {
                                printf("Turn off RGB LED of Blue\n");
                                PA12=1;
                        }
                        /*-----------------------------------------------------*/
                        else if(strcmp(cmd.command,"GRE0")==0)
                        {
                                turn_RGBLED_off();
                                printf("Change the color of RGB LED to Green\n");
                                PA13=0;

                                printf("Change to white or not?(y/n):");
                                scanf("%c",&white);
                                xQueueSend(xCommandQueueB, &white, portMAX_DELAY);
                        }
                        else if(strcmp(cmd.command,"GRE1")==0)
                        {
                                printf("Turn off RGB LED of Green\n");
                                PA13=1;
                        }
                        /*-----------------------------------------------------*/
                        else if(strcmp(cmd.command,"RED0")==0)
                        {
                                turn_RGBLED_off();
                                printf("Change the color of RGB LED to Red\n");
                                PA14=0;

                                printf("Change to white or not?(y/n):");
                                scanf("%c",&white);
                                xQueueSend(xCommandQueueB, &white, portMAX_DELAY);
                        }
                        else if(strcmp(cmd.command,"RED1")==0)
                        {
                                printf("Turn off RGB LED of Red\n");
                                PA14=1;
                        }
                        /*-----------------------------------------------------*/
                        else if(strcmp(cmd.command,"RGB1")==0){
                                printf("RGB LED off\n");
                                turn_RGBLED_off();
                        }
                        else{
                                printf("Wrong command!");
                        }
                }
        }
}
```

▲圖 11

最後是下圖 12，TaskC 的部分，它負責處理從 xCommandQueueC 佇列接收到的模式信息。根據接收到的模式，該任務會啟動不同的 LED 閃爍模式。

變數宣告：

int mode：保存接收到的模式信息的整數變數。

int freq = 30：保存閃爍頻率的整數變數，初始值我設為 30。

接收模式信息：

if (xQueueReceive(xCommandQueueC, &mode, portMAX_DELAY))：從 xCommandQueueC 佇列接收模式信息。

處理不同的模式：

如果模式是 0，表示左到右的閃爍模式，它將 keep_blink 設置為 1，然後在 keep_blink 為 1 的情況下執行 blink_left_to_right 函數，該函數負責左到右的 LED 閃爍。這個循環會一直執行，直到 keep_blink 變為 0。如果模式是 1，表示右到左的閃爍模式，同樣地，它將 keep_blink 設置為 1，然後在 keep_blink 為 1 的情況下執行 blink_right_to_left 函數，該函數負責右到左的 LED 閃爍。

閃爍將持續到 TaskA 中下一個命令輸入後，會將 keep_blink 設為 0，從而停止迴圈，讓 LED 燈熄滅。

```
334  static void TaskC(void* pvParameters)
335  {
336          int mode;
337          int freq=30;
338          while(1){
339                  if (xQueueReceive(xCommandQueueC, &mode, portMAX_DELAY))
340          {
341                          keep_blink=1;
342                          printf("Blink mode in c: %d\n",mode);
343                          if(mode == 0){
344                                  while(keep_blink){
345                                      blink_left_to_right(freq);
346                                  }
347                              }
348                          else if(mode==1){
349                                  while(keep_blink){
350                                      blink_right_to_left(freq);
351                                  }
352                              }
353                  }
354          }
355  }
```

▲圖 12

<心得與收穫>

這次 LAB2 在 Bonus 的部分花了比較多時間在處理連續閃爍 LED 的部分，由於不確定到底要不要連續，因此兩個版本都有做，這個過程讓我更深入地了解了嵌入式系統的開發和 FreeRTOS 的使用，這些經驗將對我未來非常有幫助。