實驗名稱：實驗七 STM32 Clock and Timer

實驗目的：瞭解STM32的各種clock source使用與修改、timer使用原理、PWM使用原理與應用

實驗步驟、結果與程式流程說明：

7.1 Modify system initial clock

- 利用先前 lab 所實作的 assemlby 版本 GPIO_init 與 delay_1s 初始化 GPIO 與 delay。
- 修改 SYSCLK 的 clock source 以及相關的 prescaler 使得 CPU frequency(HCLK)為 1MHz。
- 觀察修改前後 LED 燈閃爍的頻率。
- 當使用者按下 user button 便依以下順序改變 CPU system clock(HCLK)， 1MHz -> 6MHz -> 10MHz ->16MHz -> 40MHz ->1MHz ->…

下面這是程式開始時初始化 system clock 為 1MHz 的部分。

```
8  void SystemClock_Config(void)
9  {
10     RCC->CR |= RCC_CR_HSION;// turn on HSI16 oscillator
11     while((RCC->CR & RCC_CR_HSIRDY) == 0);//check HSI16 ready
12     RCC->CFGR |= 0x01; // use HSI16 as SYSCLK
13     RCC->CFGR |= 0xB<<4;//SYSCLK divide by 16. SYSCLK = 16MHz/16 = 1Mhz
14     return;
15 }
```

這邊是判斷 button 有沒有被按下的 debounce 部分。button 要持續被按著直到 count 從 5000 扣到 0 才算有被按到，並在鬆開時回傳訊號。

```
17 int user_press_button()
18 {
19     int count=5000;
20     uint32_t debounce;
21     while(count>0)
22     {
23         count--;
24         debounce = GPIOC->IDR & 1<<13;
25         if(debounce!=0)
26             return 0;
27     }
28     if(debounce!=0)
29         return 0;
30     else
31     {
32         while(!(GPIOC->IDR & 1<<13));
33         return 1;
34     }
35 }
```

這是主程式的部分，在初始化 system clock 和 GPIO 之後就是持續偵測 button 有沒有被按下來更改 system clock 的頻率，還有讓 LED 變化。

```
37 int main()
38 {
39     int state = 0;
40     SystemClock_Config();
41     GPIO_init();
42     while(1)
43     {
44         if (user_press_button()) ← 如果 button 被按下
45         {
46             //TODO: Update system clock rate
47             state++;
48             if(state==5)
49                 state = 0;
50             switch(state)
51             {
52             case 0: // 1MHz
53                 RCC->CR &= ~RCC_CR_PLLON; // set PLLON 0
54                 while((RCC->CR & RCC_CR_PLLRDY) == 1); // PLL stop ← 確定 PLL 是關掉的
55                 RCC->CFGR &= 0xFFFFFF0C;
56                 RCC->CFGR |= 0xB<<4; // SYSCLK divide by 16 ← 把 HSI16 調成 1MHz
57                 RCC->CFGR |= 0x1; // use HSI16 as SYSCLK 並設成 system clock
58                 break;
59             case 1: // 6MHz
60                 RCC->CR &= ~RCC_CR_PLLON; // set PLLON 0
61                 while((RCC->CR & RCC_CR_PLLRDY) == 1); // PLL stop ← 確定 PLL 是關掉的
62                 RCC->PLLCFGR &= 0xF8FF8083;
63                 RCC->PLLCFGR |= 0x06000922; // div: 8 mul:9 div:3 ← 把 PLL 調成 6MHz
64                 RCC->CR |= RCC_CR_PLLON;
65                 while((RCC->CR & RCC_CR_PLLRDY) == 0);//check PLL ready ← 確定 PLL 啟動
66                 RCC->PLLCFGR |= RCC_PLLCFGR_PLLREN; 並 enable
67                 RCC->CFGR &= 0xFFFFFF0C;
68                 RCC->CFGR |= 0x3; // use PLL as SYSCLK ← 把 PLL 設成 system clock
69                 break;
70             case 2: // 10MHz
71                 RCC->CFGR &= 0xFFFFFF0C; ← 先把 system clock 設成別的 clock
72                 RCC->CR &= ~RCC_CR_PLLON; // set PLLON 0
73                 while((RCC->CR & RCC_CR_PLLRDY) == 1); // PLL stop ← 確定 PLL 是關掉的
74                 RCC->PLLCFGR &= 0xF8FF8083;
75                 RCC->PLLCFGR |= 0x08000A72; // div:2 mul:10 div:8 ← 把 PLL 調成 10MHz
76                 RCC->CR |= RCC_CR_PLLON;
77                 while((RCC->CR & RCC_CR_PLLRDY) == 0);//check PLL ready ← 確定 PLL 啟動
78                 RCC->PLLCFGR |= RCC_PLLCFGR_PLLREN; 並 enable
79                 RCC->CFGR &= 0xFFFFFF0C;
80                 RCC->CFGR |= 0x3; // use PLL as SYSCLK ← 把 PLL 設成 system clock
81                 break;
82             case 3: // 16MHz
83                 RCC->CR &= ~RCC_CR_PLLON; // set PLLON 0
84                 while((RCC->CR & RCC_CR_PLLRDY) == 1); // PLL stop ← 確定 PLL 是關掉的
85                 RCC->CR |= RCC_CR_HSION;// turn on HSI16 oscillator
86                 while((RCC->CR & RCC_CR_HSIRDY) == 0);//check HSI16 ready ← 確認 HSI16 開啟
87                 RCC->CFGR &= 0xFFFFFF0C; // clear div, src
88                 RCC->CFGR |= 0x1; // use HSI16 as SYSCLK ← HSI16 維持 16MHz 並設成 system clock
89                 break;
```

```
90              case 4: // 40MHz
91                  RCC->CR &= ~RCC_CR_PLLON; // set PLLON 0
92                  while((RCC->CR & RCC_CR_PLLRDY) == 1); // PLL stop
93                  RCC->PLLCFGR &= 0xF9FF8083;
94                  RCC->PLLCFGR |= 0x01000A12; // div:2 mul:10 div:2
95                  RCC->CR |= RCC_CR_PLLON;
96                  while((RCC->CR & RCC_CR_PLLRDY) == 0);//check PLL ready
97                  RCC->CFGR &= 0xFFFFFF0C;
98                  RCC->CFGR |= 0x3; // use PLL as SYSCLK
99                  break;
100             }
101         }
102         else
103         GPIOA->BSRR = (1<<5); // LED on
104         delay_1s();
105         GPIOA->BRR = (1<<5); // LED off
106         delay_1s();
107     }
108 }
```

← 確定 PLL 是關掉的

← 把 PLL 調成 40MHz

← 確定 PLL 啟動

← 把 PLL 設成 system clock

7.2 計時器

完成作業說明裡main.c中的Timer_init()與Timer_start(); 並使用STM32 timer實做一個計時器會從0上數(Upcounting) TIME_SEC秒的時間。顯示到小數點以下第二位，結束時7-SEG LED停留在TIME_SEC的數字。使用polling的方式取得 timer CNT register值並換算成時間顯示到7-SEG LED上。

0.01 ≤ TIME_SEC ≤ 10000.00 (超過範圍直接顯示 0.00)

這是一開始先初始化 timer 的部分。

```
120 void Timer_init(TIM_TypeDef *timer)
121 {
122     //TODO: Initialize timer
123     RCC->APB1ENR1 |= RCC_APB1ENR1_TIM3EN; ← enable timer 3
124     TIM3->CR1 |= 0x1; ← enable timer 3 output
125     TIM3->PSC |= (uint32_t) 39999; // prescaler, 4000000/40000=100 ← 每 1/100 秒 count 一次
126     TIM2->ARR &= (uint32_t) 0;
127     TIM3->ARR |= (uint32_t) 10000; // reload value ← 設定 counter 上限
128     TIM3->EGR = TIM_EGR_UG;//Reinitialize the counter ← 用前面的設定 initialize timer 3
129     return;
130 }
```

這是啟動計時器的部分。

```
132 void Timer_start(TIM_TypeDef *timer)
133 {
134     //TODO: start timer and show the time on the 7-SEG LED.
135     Display(0); ← 一開始 7-segment 先顯示 0
136     TIM3->CR1 |= TIM_CR1_CEN;//start timer ← 啟動 timer 3 的 counter
137     return;
138 }
```

這是主程式的部分，完成各項初始化之後會先判斷要數的時間有沒有超過範圍，再來取樣 counter 裡的數字，判斷到了沒，到了就關掉 counter 並顯示設定的數字，還沒到就顯示取樣到的數字並持續取樣。

```
140 int main()
141 {
142     int counter_time;
143     GPIO_init();
144     max7219_init();
145     Timer_init(TIM3);
146     Timer_start(TIM3);
147     if(TIME_SEC>10000 || TIME_SEC<0.01) ← 判斷有沒有超過範圍
148         return 0;
149     while(1)
150     {
151         //TODO: Polling the timer count and do lab requirements.ir
152         counter_time = TIM3->CNT; ← 取樣
153         if(counter_time>=100*TIME_SEC) // check if times up ← 如果時間到了
154         {
155             TIM3->CR1 &= ~TIM_CR1_CEN; ← 關掉 counter
156             Display(100*TIME_SEC); ← 顯示設定的時間
157             return 0;
158         }
159         Display(counter_time);//display the time on the 7-SEG LED ← 顯示取樣到的時間
160     }
161 }
```

7.3 Music keypad

利用timer產生並輸出Duty cycle為50%的PWM訊號，並以Lab6中的keypad為鍵盤，當使用者在按下不同keypad按鍵時產生特定頻率(參考下表)的PWM方波給蜂鳴器，沒按鍵或按到沒功能的鍵時不發出聲音。

Keypad 對應音名

X0	X1	X2	X3
Y0	Do	Re	Mi
Y1	Fa	So	La
Y2	Si	HDo	
Y3			

音名頻率對應表

音名	Do	Re	Mi	Fa	So	La	Si	HDo
頻率(Hz)	261.6	293.7	329.6	349.2	392.0	440.0	493.9	523.3

7.3.1 Music 音色實驗

在前一實驗中的 keypad 增加 2 個功能按鈕用以調整 PWM 輸出的 Duty cycle(範圍 10%~90%，每按一次鍵調整 5%)，觀察是否會影響蜂鳴器所發出的聲音大小或音色。

將 timer 2 channel 1 的輸出 pin 腳 PA 5 設成 alternate function 1 的 PWM output。

```
void GPIO_init_AF()
{
    //TODO: Initial GPIO pin as alternate function
    GPIOA->MODER &= 0xFFFFF3FF;
    GPIOA->MODER |= 0x00000800; // set PA5 AF mode
    GPIOA->AFR[0] |= 0x00100000; // set PA5 as AF1
}
```

初始化 timer 的部分。

```
void Timer_init()
{
    //TODO: Initialize timer
    RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN; ← enable timer 2
    TIM2->CR1 |= TIM_CR1_ARPE; ← enable timer 2 auto reload
    TIM2->PSC |= (uint32_t) 100; // prescaler ← 之後會再修改所以先隨意填一個不會太大的值
    TIM2->ARR &= (uint32_t) 0;
    TIM2->ARR |= (uint32_t) 99; // reload value ← 設定 counter 上限為 99
    TIM2->EGR = TIM_EGR_UG;//Reinitialize the counter ← 用前面的設定 initialize timer 2
    TIM2->EGR = TIM_EGR_CC1G; ← enable capture/compare event
}
```

設定 PWM output 相關的 register。

```
void PWM_channel_init()
{
    //TODO: Initialize timer PWM channel
    TIM2->CCR1 |= 49; // compare value ← 設定 counter 等於 49 的時候 PWM 輸出要變化
    TIM2->CCMR1 |= TIM_CCMR1_OC1M_2;
    TIM2->CCMR1 |= TIM_CCMR1_OC1M_1; // PWM mode 1   ← 將 timer 2 channel 1 設成 PWM mode 1
    TIM2->CCMR1 |= TIM_CCMR1_OC1PE; ← enable output compare 1 preload
    TIM2->CCER |= TIM_CCER_CC1E; // enable OC1 output
}                                                  ↑ output compare 1
```

initialize keypad 的 GPIO 參數和 PA 5 的 PUPD。

```
void keypad_init()
{
    // SET keypad GPIO OUTPUT
    // Set PA6,7,8,9 as output mode
    GPIOA->MODER &= 0xFFF55FFF;
    // set PA6,7,8,9 as Pull-up output
    GPIOA->PUPDR |= 0x00055000;
    // Set PA6,7,8,9 as high speed
    GPIOA->OSPEEDR |= 0x000AA000;
    // Set PA6,7,8,9 as output high
    GPIOA->ODR |= 1111<<6;

    // SET keypad GPIO INPUT
    // Set PB5,6,7,9 as input mode
    GPIOB->MODER &= 0xFFF303FF;
    // set PB5,6,7,9 as Pull-down input
    GPIOB->PUPDR |= 0x8A800;
    // Set PB5,6,7,9 as high speed
    GPIOB->OSPEEDR |= 0x0002A800;

    // set PA5 Pull-down
    GPIOA->PUPDR |= 0x00000800;
}
```

主程式的部分是持續偵測 keypad 有沒有被按下，有的話如果是該發出聲音的 key 就用調整 prescaler 的方式調整頻率並輸出，如果是調整 duty cycle 的 key 就改變 PWM 的 compare value。

```
225 int main()
226 {
227     //TODO: Scan the keypad and use PWM to send the corre
228     int i, j, k;
229     int flag_keypad, flag_debounce, flag_keypad_c;
230     int position_c, position_r;
231     int Table[4][4] = {261.6, 293.7, 329.6, 0,
232                        349.2, 392.0, 440.0, 0,
233                        493.9, 523.3,     0, 0,
234                            -1,    0,    -2, 0};
235     GPIO_init();
236     GPIO_init_AF();
237     keypad_init();
238     Timer_init();
239     PWM_channel_init();
240     //max7219_init();
241
242     while(1)
243     {
244         GPIOA->ODR |= 1111<<6; //set PA6,7,8,9 (row) high
245         flag_keypad = GPIOB->IDR & 1111<<5;
246         if(flag_keypad==0)
247         {
248             TIM2->CR1 &= ~TIM_CR1_CEN; ← 如果 keypad 沒有被按下就把 counter 停掉
249             //Display(0);
250         }
251         else if(flag_keypad!=0) ← 如果 keypad 被按下了
252         {
253             k=45000;
254             while(k!=0)
255             {
256                 flag_debounce = GPIOB->IDR & 1111<<5; ← 做 debounce
257                 k--;
258             }
259             if(flag_debounce!=0) ← 如果 debounce 通過
260             {
261                 for(i=0;i<4;i++)
262                 {   //scan keypad from first row
263                     position_r = i+6;
264                     //set PA6,7,8,9(row) low and set pin high from PA5
265                     GPIOA->ODR &= 0xFC3F;
266                     GPIOA->ODR |= 1<<position_r;
267                     for(j=0;j<4;j++)                    ← 一排一排掃 key
268                     {   //read input from first column
269                         position_c = j+5;
270                         if(j==3)
271                             position_c++;
272                         flag_keypad_c = GPIOB->IDR&1<<position_c;
273                         if(flag_keypad_c!=0)
274                         {
275                             if(Table[i][j]>0) ← 如果這個 key 該發出聲音
276                             {
277                                 TIM2->PSC = (uint32_t) (40000/Table[i][j]); ← 改變 prescaler
278                                 TIM2->CR1 |= TIM_CR1_CEN; ← 啟動 counter 來發出聲音
279                                 //Display(Table[i][j]);
280                             }
```

ARR+1

freq*100 = 4M/PSC

PSC = 40k/freq

```
281                             else if(Table[i][j]==-1) ← 如果這個 key 該減少 duty cycle
282                             {
283                                 if(TIM2->CCR1>9) ← 如果 duty cycle>10%
284                                     TIM2->CCR1 -= 5; ← duty cycle - 5%
285                                 //Display(TIM2->CCR1+1);
286                             }
287                             else if(Table[i][j]==-2) ← 如果這個 key 該增加 duty cycle
288                             {
289                                 if(TIM2->CCR1<89) ← 如果 duty cycle<90%
290                                     TIM2->CCR1 += 5; ← duty cycle + 5%
291                                 //Display(TIM2->CCR1+1);
292                             }
293                             while(flag_keypad!=0)
294                             {                                                    ← 等到 key 被放開
295                                 flag_keypad = GPIOB->IDR & 1111<<5;
296                             }
297                             break;
298                         }
299                     }
300                 }
301             }
302         }
303     }
304 }
```

心得討論與應用聯想：

這次的作業寫了非常久，久到只要想到後面積著還沒寫的其他作業就覺得接下來幾天大概要灌咖啡才撐得下去了。exp 1 只是因為單純的一開始還對 timer 不熟，所以多花了一些時間才弄懂 timer 參數的調整順序，還有到底有哪些 bits 要改。exp 2 說實在並不複雜，但是我卡了很久很久、試著改了其他很多參數才發現原來是我的 RCC_APB1ENR1_TIM3EN 打成 RCC_APB1ENR1_TIM2EN 了，所以 counter 才沒有動靜。exp 3 本來就花不少時間研究 PWM 的設定，但加上去之後 buzzer 都只動一下就沒反應了，所以我又看了很久的 PWM 參數，可是找不出我還有什麼該設的沒設到，後來無計可施之下我把 counter 的值直接丟給 7-segment 來看看到底發生什麼事，發現 counter 數到 ARR 之後居然沒有停下來，於是開始去看關於 counter 到底是怎麼運作跟 interrupt 相關的東西，但還是找不出原因。最後在各種亂嘗試之下才終於發現雖然 reference manual 裡說 ARR 的 reset value 是 0x00000000，但是顯然實際上不是，裡面有不知道多少、總之很大的數字，加上我用 or 的方式 assign ARR 的值，造成 counter 遲遲跑不到可以停下來的 ARR 值、PWM 過了 compare value 掉到 0 之後就沒有 counter reset 回到 1 的機會了。

雖然做的過程一波三折，但是作業的內容感覺還是很實用的。有了 timer 跟 PWM 之後不管是要定時相關的應用、發出聲音、甚至是 ADC 之類的功能好像可以都做得出來了，期末專題應該也會用得上。

是說我覺得這次作業最大的教訓是參數要看清楚，還有絕對不要太相信 reference manual 裡說的 reset value。