第 39 組　　報告撰寫者:0316306 周煥然

實驗名稱: 實驗八 Interrupt and Exception

實驗目的:

1. 了解 STM32 SysTick timer設定
2. 了解 STM32 NVIC和External interrupt設定和原理

實驗原理:

參考上課講義

實驗步驟:

Lab8.1: SysTick timer interrupt

- 實作一個SysTick interrupt handler，當中斷發生時toggle LED燈明暗。
- 當使用者按下user button開啟或關閉SysTick timer。

Notes: 設定 SysTick clock source 為 10MHz，SysTick timer 每 0.5 秒 interrupt 一次。

Lab8.2 : Keypad external interrupt

這部分的實驗主要請同學將Lab6中所實作的鍵盤掃描程式改成利用SysTick與外部中斷EXTI完成(無須掃描迴圈)。主要原理由以下3個部分完成。

- 將Column output掃描由SysTick interrupt handler完成，中斷時間間隔0.1s，當SysTick中斷發生時更改scan column。
- 在SysTick interrupt handler中設定並啟動keypad row的4個input腳為負邊緣觸發(Negative trigger)的外部中斷
- 當EXIT中斷發生時讀取4個input的值，並根據目前column掃描狀態判斷是哪個鍵按下。

在主程式中依使用者所按下的按鍵值利用lab6的display()顯示至7段顯示器上。

Lab8.3: 製作簡單鬧鐘

利用SysTick timer、User button和蜂鳴器設計一個簡單的鬧鐘，

- 利用keypad輸入計時鬧鐘倒數時間並即時顯示至7-Seg LED，每一個數字代表設定幾秒(2為2秒)
- 輸入為0時則沒反應，繼續等待下次輸入，
- 按下User button則代表時間輸入完畢

- 啟動一秒觸發一次interrupt的Systick timer開始倒數，
- 利用7-seg LED顯示目前倒數的時間秒數
- 當時間到後，蜂鳴器便會響起(在SysTick interrupt handler中利用while loop讓蜂鳴器持續發出聲音，頻率自訂)
- 直到使用者按下User button後才會停止發出聲音並回到等待使用者輸入狀態，注意SysTick開始計時到使用者關閉蜂鳴器的期間，keypad不會有任何作用。(程式會由user button觸發一個nested interrupt)

實驗結果:

Lab8.1: SysTick timer interrupt

先將 System clock 設為10Mhz，之後用以下方式設定SysTick timer。

```
// setup systick
int *STK_CTRL = 0xE000E010;
int *STK_LOAD = 0xE000E014;
int *STK_VAL  = 0xE000E018;
*STK_LOAD = 5000000;
*STK_VAL = 0;
*STK_CTRL = 7;
```

而在SysTick_Handler中將LED打開和關閉。
當user按下按鍵時改變SCB->SHCSR register控制是否active SysTick timer event。

Lab8.2: Keypad external interrupt

在 EXTI_Setup()中設定 EXTI 相關的 Register，將四個 input pin 設為 negative trigger 的 interrupt。

```
RCC->APB2ENR |= 1;
SYSCFG->EXTICR[1] |= (0x2<<12); //pc7
SYSCFG->EXTICR[1] |= (0x1<<8);    //pb6
SYSCFG->EXTICR[2] = 0x0;          //pa8 pa9

EXTI->IMR1 |= (0b1111<<6);
EXTI->RTSR1 = 0;
EXTI->FTSR1 |= (0b1111<<6);
NVIC EnableIRQ(EXTI9 5 IRQn);
```

而在SysTick_handler中每次掃描一個column，若有按鍵按下時就會觸發interrupt，進入EXTI9_5_IRQHandler()，讀取EXTI->CR的值得知是哪個pin interrupt，並根據此刻的output state 得知是哪個鍵被按下。

```
switch(scan_state)
{
case 0:
    scan_state = 1;
    GPIOB->BSRR = X0;
    GPIOB->BRR = X0;
    break;
case 1:
    scan_state = 2;
    GPIOB->BSRR = X1;
    GPIOB->BRR = X1;
    break;
case 2:
    scan_state = 3;
    GPIOB->BSRR = X2;
    GPIOB->BRR = X2;
    break;
case 3:
    scan_state = 0;
    GPIOB->BSRR = X3;
    GPIOB->BRR = X3;
    break;
}
```

Lab8.3: 製作簡單鬧鐘

在EXTI_Setup()中設定EXTI相關的Register，將user bottom(pc13)設定為postive trigger的interrupt。並設定每個interrupt的priority，user bottom的priority的最高，其次為SysTick，最低的為keypad input。讀鍵盤的方式與Lab8.2相同。而程式主要會有兩個state，一個是user輸入的state，另一個則是開始倒數計時及觸發alarm的state。而systick在user input state 時為每0.1秒interrupt一次來掃描keypad，而在倒數計時時則是每1秒interrupt一次。當user按下bottom時則切換state並設定SysTick interrupt的頻率。

```
void EXTI_Setup(){
//TODO: Setup EXTI interrupt
    RCC->APB2ENR |= 1;
    SYSCFG->EXTICR[1] |= (0x2<<12); //pc7
    SYSCFG->EXTICR[1] |= (0x1<<8);    //pb6
    SYSCFG->EXTICR[2] = 0x0;          //pa8 pa9
    SYSCFG->EXTICR[3] |= (0x2<<4);  //pc13

    EXTI->IMR1 |= (0b1111<<6);
    EXTI->IMR1 |= (1<<13);
    EXTI->RTSR1 = 0;
    EXTI->RTSR1 |=(1<<13);
    EXTI->FTSR1 |= (0b1111<<6);
    NVIC_EnableIRQ(EXTI9_5_IRQn);
    NVIC_EnableIRQ(EXTI15_10_IRQn);
    NVIC_SetPriority(EXTI15_10_IRQn,1);
    NVIC_SetPriority(SysTick_IRQn,2);
    NVIC_SetPriority(EXTI9_5_IRQn,3);
    //NVIC_DisableIRQ(EXTI9_5_IRQn);
}
```

心得討論與應用聯想：

這次實驗與以往相當不同，主要是要利用各種不同的 event 與 interrupt 去做對應的事情，讓程式不再是用 polling 的方式去讀取各種狀態，如此可以同時間控制更多的 io 並做出更複雜的應用。