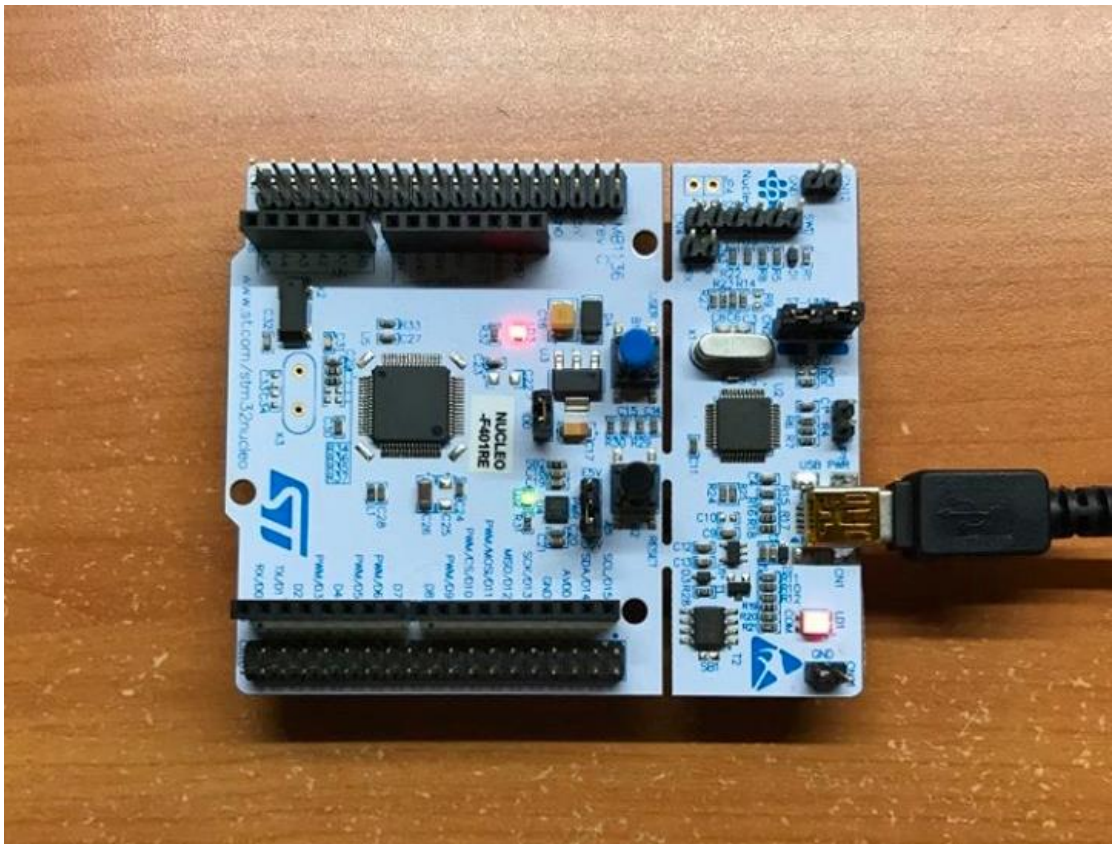
Embedded System Labs

Lab01

組名：In bed did system

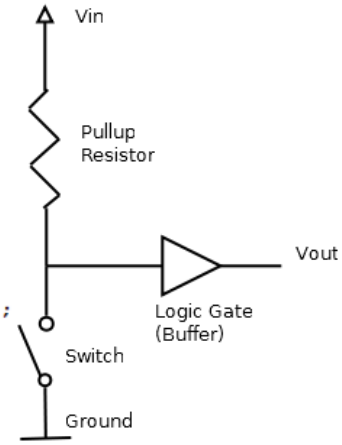
電機四 B05501032 林士鈞
電機四 B05901043 莊鎧爾
電機四 B05901097 林仕倫



Digital Input / Output and GPIO

這個實驗的目的在於熟悉 I/O 的初始設定和使用方式，需要完成的函數有 init_button、init_led、led_on、led_off 四個，並在 main 函數使用這四個函數。

init_button 中，由於為了省電，port 預設通常為 disable，因此需要先設定 port C 是 enable；接著是將要使用的 pins 設定成 input mode，需要設定 MODER 等於 00，有兩顆按鈕需要設定兩個值；最後要設定用 pull-up mode，pull-up 的電路大致如下圖，按鈕未下壓時輸出電壓會上拉，輸出電壓預設值為 1。



遇到的困難：這邊 ”A &= ~B | ~C” 等價於 “A = (A & ~B) | (A & ~C) 讓我們困惑了許久。

```
void init_button(void) {
  //Enable IO port C
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN;

  //Set pins to input mode
  GPIOC->MODER &= ~GPIO_MODER_MODER13_0 | ~GPIO_MODER_MODER13_1 ;

  //Set pins to pull-up mode
  GPIOC->PUPDR |= GPIO_PUPDR_PUPDR13_0;
  GPIOC->PUPDR &= ~GPIO_PUPDR_PUPDR13_1;

}
```

init_led 一樣需要設定 enable、output mode (值為 01)和 pull-up mode，與 init_button 不同的是，init_led 需要設定 push-pull output state 和 fast speed，push-pull mode 是與 open-drain mode 相對的存在，會讓 output 的值能快速的從 0 變成 1；speed 的選擇會影響耗電量與運行效率。

```
void init_led(void) {

  //Enable IO port A
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

  //Set the pin to output mode
  GPIOA->MODER |= GPIO_MODER_MODER5_0;
  GPIOA->MODER &= ~GPIO_MODER_MODER5_1;

  //Set the pin to push-pull output state
  GPIOA->OTYPER &= ~GPIO_OTYPER_OT_5 ;

  //Set pins to pull-up mode
  GPIOA->PUPDR |= GPIO_PUPDR_PUPDR5_0;
  GPIOA->PUPDR &= ~GPIO_PUPDR_PUPDR5_1;

  //Set pin to fast speed
  GPIOA->OSPEEDR &= ~GPIO_OSPEEDER_OSPEEDR5_0;
  GPIOA->OSPEEDR |= GPIO_OSPEEDER_OSPEEDR5_1;

}
```

led-on 和 led-off 就很直觀，BSRR 是一個 32bit 的 register，0~15 是 set，16~31 是 reset，led_on 需要設定 set，因此將 BSRRL 裡對應的 pin 設成 1，同理，reset 是設定 BSRRH。

```
// Set output pin PA_5 to high
void led_on(void) {
    GPIOA->BSRRL |= GPIO_BSRR_BS_5;

}

// Set output pin PA_5 to low
void led_off(void) {
    GPIOA->BSRRH |= GPIO_BSRR_BS_5;

}
```

main function 會先初始化 led 和 button，接著進入無限迴圈偵測按鈕有無下壓，若有則啟動 led，放開按鈕則關閉 led。

```
int main() {
  // Initialise LEDs and buttons
  init_led();
  init_button();

  while(1){
    // If the button is pressed turn on the LED
    if(!(GPIOC->IDR & GPIO_IDR_IDR_13)) {
      led_on();
    }
    else{ // Otherwise turn off the LED
      led_off();
    }
  }
}
```

Interrupt and Low Power Features

這個實驗的目的在於利用 interrupt 減少條件判斷，藉此達到省電的效果。

需要完成的檔案有 main.c、interrupts.c、leds.c、buttons.c，其中 leds.c 和 buttons.c 和 Lab02 非常相似，都是啟動 leds 和 buttons 的 IO 功能並且做一些初始化的設定，其中與 Lab02 不同之處是 leds.c 裡加了 toggle function，如下：

```
//Toggle the LED state
void toggle(void){
  GPIOA->ODR ^= GPIO_ODR_ODR_5;
}
```

Toggle 讀出 led 當下的 output data register 值，與 1 做 exclusive or 變成相反的值，因此 toggle 就像是 LED 的開關，觸發就會做一次亮暗的變化。

而 interrupts.c 是 lab03 的核心觀念，lab03 希望達到使用者點下按鈕之後會觸發 interrupt，之後執行 interrupt 相對應的程序，而我們把 interrupt 的對應程序設成 LED toggle，因此按下按鈕，LED 就會做一次亮暗變化，其中 interrupts.c 裡寫了一些 interrupt 的初始設定並且與 button 對應的 register 連結，以達到按下按鈕就啟動 interrupt 的功能。

```
void init_interrupts(void){
  //Start clock for the SYSCFG
  RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
  //Enable debug in low-power modes
  DBGMCU->CR |= DBGMCU_CR_DBG_SLEEP | DBGMCU_CR_DBG_STOP | DBGMCU_CR_DBG_STANDBY;
  //Setup interrupt configuration register for interrupts
  SYSCFG->EXTICR[3] |= SYSCFG_EXTICR4_EXTI13_PC;
  //Set the interrupt mask
  EXTI->IMR |= (EXTI_IMR_MR13);
  //trigger on falling edge
  EXTI->FTSR |= (EXTI_FTSR_TR13);
  //Enable interrupts
  __enable_irq();
  //Set priority
  NVIC_SetPriority(EXTI15_10_IRQn, 0);
  //Clear pending interrupts
  NVIC_ClearPendingIRQ(EXTI15_10_IRQn);
  //Enable the specific interrupt
  NVIC_EnableIRQ(EXTI15_10_IRQn);
}
```

在 main.c 裡，增加一個 interrupt 觸發時會執行的 function，如下

```
void EXTI15_10_IRQHandler(void){
  //Clear pending interrupts
  NVIC_ClearPendingIRQ(EXTI15_10_IRQn);
  //Check that the user button is pressed
  //Toggle the LED
  if (EXTI->PR & EXTI_PR_PR13) {
    toggle();
  }
  //Clear the EXTI pending register
  EXTI->PR |= EXTI_PR_PR13;
}
```

他會檢查 interrupt 對應的 pending register 是不是 1，如果是，就執行 toggle，之後把 pending register 清掉，其中比較容易搞錯的是清掉 pending register 要把它設成 1，這在文件中有提到。

最後，在 main function 中會先把 interrupt、button 和 led 先做好初始設定，之後就等待 interrupt 的到來，這種方式可以大幅減少耗能。

```
int main() {
  // Initialise LEDs and buttons
  init_led();
  init_button();
  init_interrupts();

  while(1){
  __wfi();  //Wait for interrupts
  }
}
```

Programming Using mbed API

此實驗的目的在於熟悉使用 mbed API 編寫程式，編寫上更有可讀性，且邏輯上更容易理解。

第一部分是介紹 I/O 的初始化設定與使用方式。原本 led 與 button 的初始化需要設定許多參數，而利用 API 物件化後只需一行就能完成參數設定。

```
// Create a DigitalOut objects for the LED
DigitalOut LED(LED1);

// Create a DigitalIn objects for the button.
DigitalIn  BUTTON (USER_BUTTON);
```

修改 output 或是讀入 input 也只要簡單的使用宣告過的變數，不需要知道 led 和 button 是哪個 port 的幾號 pin。

```
int main() {

  while(1) {
    // The buttons is active low
    // If the button is pressed the LED blinks twice per second
    if(!BUTTON){
      LED = !LED;
    }
    // Otherwise the LED is switch off.
    else {
      LED = 0;

    }
    wait(0.25);
  }
}
```

第二部分則是介紹 mbed API 的 interrupt 是如何使用。在初始化時除了宣告 API 物件外，還需要在 main 函數中設定 pull up mode 與下壓時要呼叫的函數，

```
// Create an InterruptIn object for the button.
InterruptIn  BUTTON (USER_BUTTON);
```

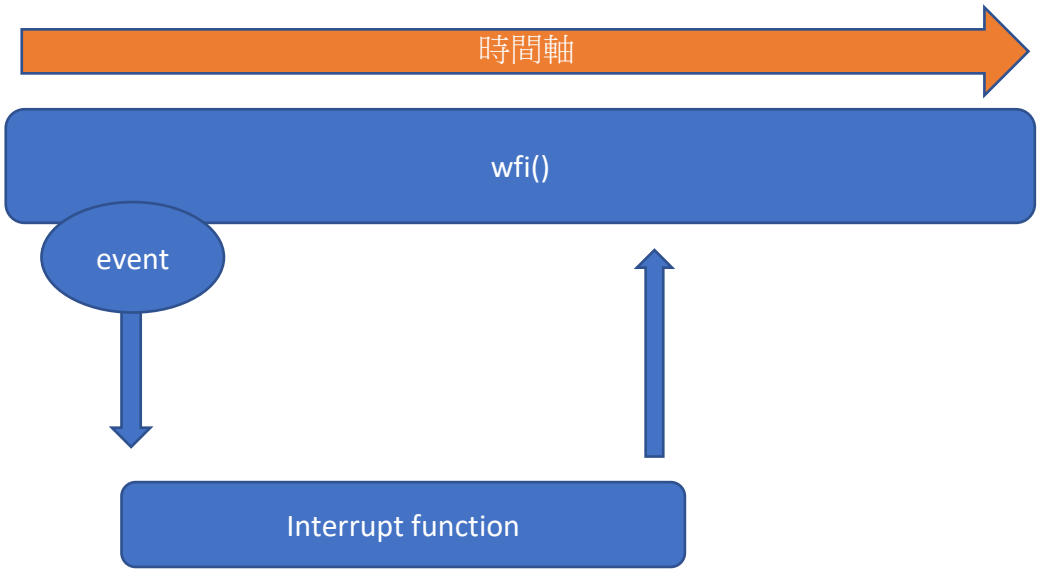
此外，還可以設定 Ticker 物件，進行 recurring interrupting，需要在 main 函數中用 attach 設定呼叫的函數與定時多久呼叫。

```
Ticker blinky;

int main() {
// Set up

    BUTTON.mode(PullUp);
    BUTTON.fall(&BUTTON_ISR);

    blinky.attach(&BLINKY_ISR, 4);
```

*多重 **Interrupt** 的運作方式*

Interrupt function 會綁定一個 event，當事件發生時，程序會執行 interrupt function.

單一的 interrupt 發生時，系統會立刻執行 interrupt function



當系統正在執行 interrupt function 時，恰巧又有一個 event 進來，系統會排定時程，並在第一個 interrupt function 執行完直接執行另一個