組別：6，組員：王柏堯、張彧豪

微處理機 Lab4 Report

一、 實驗名稱：STM32 GPIO System

二、 實驗目的：

- 了解 STM32 基本輸出入 I/O port 使用原理。
- 設計簡易 LED 跑馬燈程式。
- 了解按鈕與指撥開關使用原理。

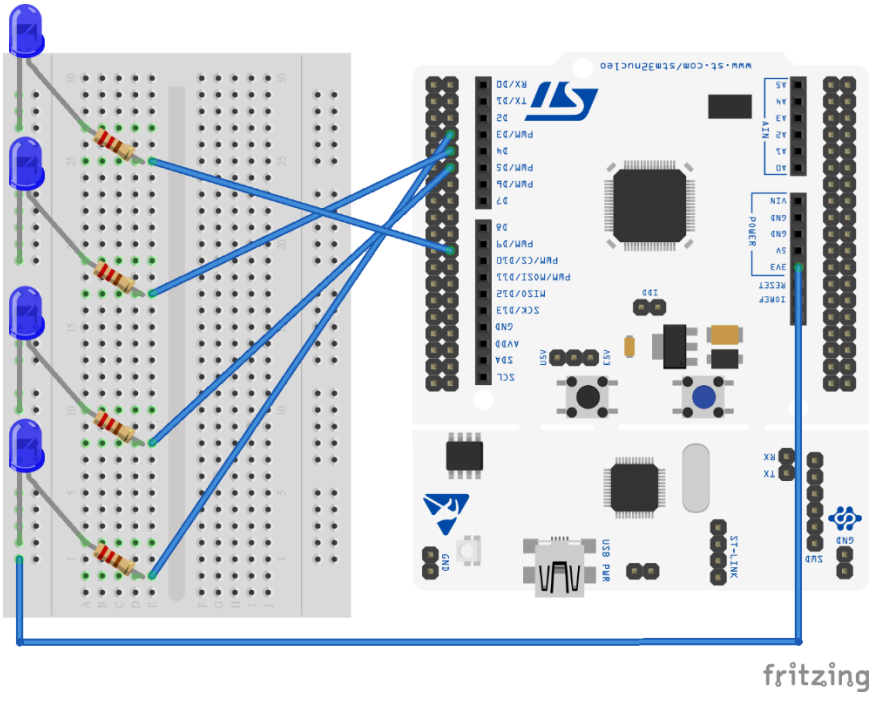
三、 實驗步驟：

1. LED pattern displayer。
2. Push button 的基本操作。
3. 密碼鎖與 DIP Switch 的使用。

四、 實驗結果與分析：

1. LED pattern displayer：

電路簡圖：



這題的接線並不難，就像老師上課講義內的接就可以了。我們覺得比較難的地方是該怎麼把跑馬燈的每個不同的動作都寫進去自動化的過程裡，因為只有在邊際時 LED 才會只亮一顆，其它時候都是兩顆，我們的做法是把 GPIOB 的 2、7port 也拿來用，這樣跑馬燈的每個不同動作其實就是在讓連續兩顆燈亮起來，只是位置不同而已，每隔一秒位移一次資料輸出就可以達成了，而且因為 2、7port 沒有接上 LED 燈，所以當跑馬燈跑到邊際時，還是只會亮一顆 LED 燈，以此解決在邊際時需要特別判斷的問題且也符合題目的要求。而另外一個問題點是要怎麼讓每次的動作持續一秒，延遲的方法我們是參考老師上課講義的，只不過那個數字需要湊一下，湊到大概延遲一秒即可。

2. Push button 的基本操作：

我們覺得這題的關鍵就在於要怎麼處理按鈕彈跳(bounce)的問題，按鈕是屬於 active-low 形式且題目要求是” 按鈕按一下”之後跑馬燈才有動作，我們處理的方式處理按鈕彈起來時的彈跳問題並搭配有限狀態機的使用。

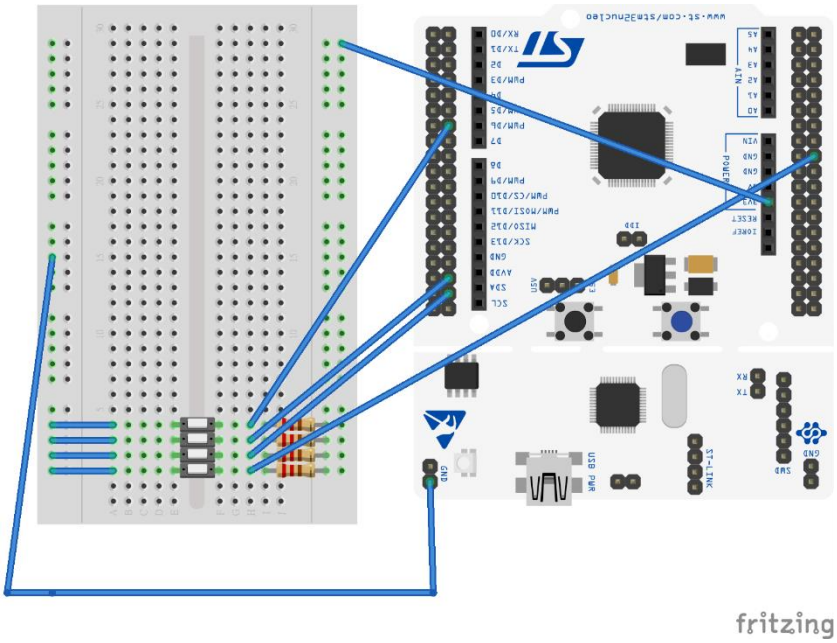
處理方式是用迴圈方式收集目前按鈕輸入並存在一個暫存器 r6，因為按鈕是 active-low 的，所以如果按鈕確實被壓下去時，這時候收集到的輸入應該都是 0，而彈上去的時候應該會收集到 0 跟 1 交錯的訊號，也就是機械彈跳所造成的，這也是我們用來確認按鈕有無被壓下去的方法，而有限狀態機就是在幫我們區分這些狀態，初始化時先把 r6 設成 0xFFFFFFF、狀態設 s1，當發現 r6 的值等於 0 時，表示按鈕確實被壓下，狀態跳至 s2，這時若發現 r6 不為 0 時，代表按鈕被放開了，狀態跳至 s2，把跑馬燈新狀態參數準備好並把狀態調回初始狀態 s1 後跳至 DisplayLED 去改變跑馬燈目前動作。

切換跑馬燈動作這邊就比較簡單，我們也是用有限狀態機才控制跑馬燈動作，這個狀態機只有兩個狀態，動跟不動，如果是動的狀態，就是執行第一題的 code，如果不動，直接跳去 Delay。

解決如何處理彈跳問題後還有一個盲點就是，處理彈跳問題的 code 該放在哪裡? 因為輸入就像中斷一樣，必須及時處理，拿第一題的思路來看的話就是要把佔用 CPU 時間最多的那一段 code 直接改成處理彈跳問題，因為 Delay 才是跑馬燈程式中佔用 CPU 時間最多的程序，所以把 Delay 這段改成處理彈跳問題是最好的選擇，讓延遲變成處理彈跳問題，讓按鈕輸入不管在何時都能被及時處理。

3. 密碼鎖與 DIP Switch 的使用：

電路簡圖：



這題最困難的部分就是電路了，因為題目要求是 active-low 的開關，所以必須要接四個上拉電阻，而上拉電阻的接法就是電阻一端接 VCC，另一端接到 pin 腳上，而 switch 則是一端接到 pin 腳上，另一端則接地，這樣當 switch 沒有連通時輸入的針腳讀到的訊號為 1，反之為 0，大致上的電路如上圖所示。四個 switch 使用的 GPIO

port 由下到上分別是 GPIOB 7~10port。

這題的做法是讓 CPU 一直處理彈跳問題，有點類似 busy wait，因為不知道按鈕何時會被按下，所以必須這樣實作。這裡處理彈跳問題的方式與第二題一樣；當確認按鈕被按一次後，讀取 Switch 的數值與 password 做比較，分別做相對應的 LED 閃爍動作。這裡實作閃爍的方法跟前兩題類似，只不過每次輸出 LED 值前要先把上次的輸出 NOT 一次後再輸出。

五、 心得討論與應用聯想：

這次作業較容易，大致上都是在處理 GPIO 如何開啟，以及沒有公公線時要如何實作電路(但後來借到公公線了)，以及計算 DELAY 要多久。而困擾我們比較久的是在第二題如果連續快按兩次按鈕時，有時候燈泡會亮錯，經過長時間的除錯發現是 branch 之後回去的地方錯誤導致燈泡顯示錯誤，改成對的邏輯之後連續快按兩次按鈕的問題就解決了。