

微處理機系統與介面技術 LAB 3

系所：電機 學號 :612415013 姓名：蕭宥羽

<實驗器材>

NUC 140 V2.0 開發板



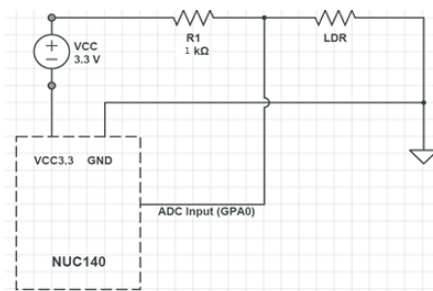
<實驗過程與方法>

實驗要求：將 LDR 收到的 ADC 數值用 putty 印出來

Putty 印出的頻率大概 0.5~1ms

用收到光敏電阻的數值閃爍 LED

電路接法：ADC 接收 3.3v 的分壓，將電壓轉為數位表示。



數位表示方法為 12 個 bits，所以表示的範圍是 0-4095

A/D Conversion Result		
[15:0]	RSLT	This field contains conversion result of ADC.
		When DMOF bit (ADCR[31]) set to 0, 12-bit ADC conversion result with unsigned format will be filled in RSLT[11:0] and zero will be filled in RSLT[15:12].
		When DMOF bit (ADCR[31]) set to 1, 12-bit ADC conversion result with 2's complement format will be filled in RSLT[11:0] and signed bits to will be filled in RSLT[15:12].

所使用到的 function 共有這些，SYS_Init、UART0_Init、GPIO_Init、ADC_Init 負責做初始化，ADC_IRQHandler 發生 ADC 中斷時會執行裡面的程式碼

AdcSingleModeTest 印出 ADC 值以及 LED 閃爍是由這支程式碼控制

```
void SYS_Init(void);
void UART0_Init(void);
void ADC_Init(void);
void GPIO_Init(void);
void AdcSingleModeTest(void);
void ADC_IRQHandler(void)
```

<Main function code>

1. SYS_Init

```
/* Configure the GPA0 - GPA3 ADC analog input pins */
SYS->GPA_MFP &= ~(SYS_GPA_MFP_PA0_Msk | SYS_GPA_MFP_PA1_Msk | SYS_GPA_MFP_PA2_Msk | SYS_GPA_MFP_PA3_Msk) ;
SYS->GPA_MFP |= SYS_GPA_MFP_PA0_ADC0 | SYS_GPA_MFP_PA1_ADC1 | SYS_GPA_MFP_PA2_ADC2 | SYS_GPA_MFP_PA3_ADC3 ;
SYS->ALT_MFP1 = 0;
```

對 MCU 的 PA0 到 PA3 引腳的功能配置，使得它們能夠作為 ADC 的模擬輸入通道。

- ✓ 清除引腳的多功能配置（使得引腳狀態為默認，準備配置）
- ✓ 設置引腳為 ADC 通道（GPA0 到 GPA3 分別對應 ADC0 到 ADC3）
- ✓ 將替代功能寄存器設為 0（確保這些引腳不啟用其他替代功能，只作為 ADC 輸入）

本次實驗只會用到 ADC2 也就是 PA2 這隻腳位做為 ADC 的輸入

2. ADC_Init

```
/* Set the ADC operation mode as single*/
/* input mode as single-end and enable the analog input channel 2 */
ADC_Open(ADC, ADC_ADCR_DIFFEN_SINGLE_END, ADC_ADCR_ADMD_SINGLE, 0x1 << 2);

/* Power on ADC module */
ADC_POWER_ON(ADC);

/* Clear the A/D interrupt flag for safe */
ADC_CLR_INT_FLAG(ADC, ADC_ADF_INT);
```

- ✓ ADC_Open(ADC, ADC_ADCR_DIFFEN_SINGLE_END, ADC_ADCR_ADMD_SINGLE, 0x1 << 2)

■ ADC_ADCR_DIFFEN_SINGLE_END

設置 ADC 為**單端模式**（Single-Ended），代表每個輸入信號都是相對於接地進行測量的

■ ADC_ADCR_ADMD_SINGLE

設置 ADC 的工作模式為**單次轉換模式**（Single Conversion）

每次只對指定的輸入通道進行一次模數轉換，完成後停止，等待下一次的啟動指令。

ADST 轉換完成後會 cpu 設為 low，也就是說轉換完後就不會再轉換，如果要在轉換的話要手動將 ADST 打開才行

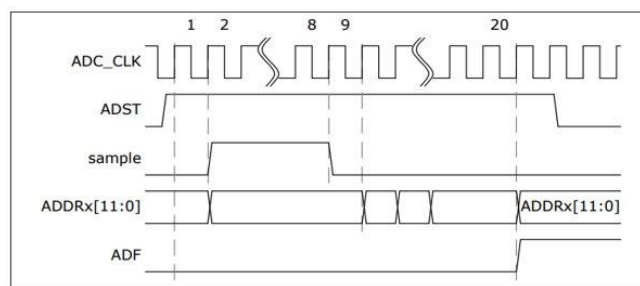


Figure 5-101 Single Mode Conversion Timing Diagram

■ 0x1 << 2

啟用**通道 2**。接下來的轉換將針對模擬輸入通道 2 進行。

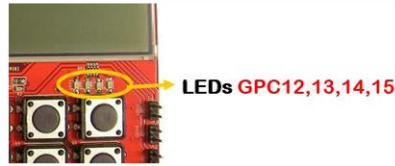
- ✓ ADC_POWER_ON(ADC): 為 ADC 模組供電，使得它從低功耗狀態進入工作狀態

- ✓ ADC_CLR_INT_FLAG(ADC, ADC_ADF_INT)

ADC_ADF_INT 是 ADC 的中斷標誌，表示轉換完成標誌位，當 ADC 完成模數轉換時，ADF 會被設置為 1，表示有新數據可以讀取，在啟動 ADC 之前，先清除中斷標誌，確保接下來的轉換是從乾淨的狀態開始

3. GPIO_Init

```
GPIO_SetMode(PC, (BIT12|BIT13|BIT14|BIT15), GPIO_PMD_OUTPUT);  
PC->DOUT = (PC->DOUT & 0x0FFF) | 0xF000;
```



啟用 LED，GPC12-15 設為 output mode，並將初始值設為 1(關)

先讓 PC->DOUT & 0x0FFF 把 PC12-15 設為 0 其他 bits 會被保留，再去 or 0xF000 將 PC12-15 設為 1，其他 bits 依然保留原本狀態。

4. ADC_IRQHandler

當 ADC 轉換完成後會將 ADC_ADF 舉起來，接這就會觸發中斷告訴 cpu ADC 已經轉換完成了，可以去讀取數據了，那如果 ADF 沒有被放下的話中段就會一直被觸發，所以轉換完成後我們帶回一個旗號 g_u32AdcIntFlag，並將 ADF 清除。

```
void ADC_IRQHandler(void)  
{  
    g_u32AdcIntFlag = 1;  
    /* 對ADSR.ADF寫1，這樣可以將此位元變成0 */  
    ADC_CLR_INT_FLAG(ADC, ADC_ADF_INT);  
}
```

5. AdcSingleModeTest

1. 函數中的主要變數

- ✓ i32ConversionData：用來存儲 ADC 轉換後的數字值
- ✓ scaledValue：將 ADC 的轉換結果縮放到 0 - 100 的範圍，用於更直觀地表示轉換結果
- ✓ referenceValue：根據 scaledValue 計算得出的值，用於控制 LED 的顯示效果
- ✓ GPIO_counter：用來控制哪些 GPIO 的 LED 被點亮
- ✓ conter：一個計數器，用來實現基於 referenceValue 的 LED 點亮行為

```
int32_t i32ConversionData;  
uint8_t scaledValue;  
uint16_t referenceValue = 0;  
uint8_t GPIO_counter = 12;  
uint8_t counter=0;
```

2. 初始化並啟動 ADC

- ADC_EnableInt(ADC, ADC_ADF_INT)
啟用 ADC 的轉換完成中斷 (ADF)，這樣當轉換完成時會觸發一個中斷信號。
- NVIC_EnableIRQ(ADC_IRQn)
啟用中斷控制器的 ADC 中斷，使得 CPU 能夠響應這個中斷
- ADC_CLR_INT_FLAG(ADC, ADC_ADF_INT)
清除 ADC 的中斷標誌位，以便於開始新的轉換
- ADC_START_CONV(ADC)
開始 ADC 的模數轉換。

```

/* Enable the ADC interrupt */
ADC_EnableInt(ADC, ADC_ADF_INT);
NVIC_EnableIRQ(ADC_IRQn);
ADC_CLR_INT_FLAG(ADC, ADC_ADF_INT);
ADC_START_CONV(ADC);

```

3. 進行連續的轉換循環

if (g_u32AdcIntFlag == 1)

當 g_u32AdcIntFlag 等於 1 時，表示 ADC 已完成一次模數轉換，並且中斷已觸發。這邊就會進入後面處理內容的環節

3.1 讀取轉換結果

- i32ConversionData = ADC_GET_CONVERSION_DATA(ADC, 2);

從 ADC 的數據暫存器中讀取通道 2 的轉換結果

因為是 12bits ADC，所以範圍是 0 到 4095

- g_u32AdcIntFlag = 0;

清除中斷旗號，準備下一次的轉換

- ADC_START_CONV(ADC);

因為是用 signal mode 做轉換，所以要手動重啟 ADC，進行下一次的模數轉換，確保 ADC 持續進行測量

3.2 顯示轉換結果

- scaledValue = (i32ConversionData * 100) / 4095;

縮放到 0 到 100，以便於更加直觀地顯示轉換結果。

- printf("Result of channel: %d\n", scaledValue);

3.3 控制 LED 顯示

- referenceValue = (100 - scaledValue) * 10;


據 scaledValue 計算 referenceValue，用於控制 LED 的亮滅頻率。因為作業希望越亮閃越慢，當越亮，電阻越低，分壓越低；所以這邊要做一個反轉讓 counter 數比較久才會超過 referenceValue，這樣就會閃比較慢。

- counter++;


- if (counter >= referenceValue)

當 counter 超過 referenceValue 時，會觸發 LED 點亮的變化


- LED 控制邏輯

 if (GPIO_counter > 15) GPIO_counter = 12;

控制 GPIO 循環在 GPIO 12 到 15 之間

 PC->DOUT = (PC->DOUT & 0x0FFF) | 0xF000;

將 GPIO 的低 12 位保留，將 12-15 設為 1

 PC->DOUT &= ~(1 << GPIO_counter);

1 << GPIO_counter 是將數字 1 左移 GPIO_counter 位，這樣可以在

GPIO_counter 所對應的位元位置上設置為 1，而其他所有位元都為 0，接著在做反轉變成只有 GPIO_counter 是 0 其他都是 1，在與 PC->DOUT 做 and，便可只將 GPIO_counter 這個位元設為 0，其他位元狀態不會被改變。

結合上念的程式碼便可以將 12-15 中 GPIO_counter 數到的腳位設成 0，也就是

將那顆 LED 點亮，其另外三顆關掉。

```
GPIO_counter++;  
counter = 0;  
counter 重置為 0，轉換完 LED 之後就會在重新數一次 counter
```

3.4 CLK_SysTickDelay(500); 這邊讓每跑一圈大概控在 0.5ms，所以上面 LED 閃爍的速度就是 $\text{referenceValue} \times 0.5\text{ms}$ ，這樣做記可以將 putty 印出的頻率控制在 0.5ms 左右，同時也實現 LED 的閃爍速率控制。

6. 其他

✓ critical section

原本有利用 disableint 關閉中斷，避免前後讀到的值是兩次不同轉換的值

```
/* Disable the ADC interrupt */  
ADC_DisableInt(ADC, ADC_ADF_INT);  
  
/* Enable the ADC interrupt */  
ADC_EnableInt(ADC, ADC_ADF_INT);
```

但我在這邊設置 我讀完值之後才會進行下一次的轉換，所以就不用用這個方式就可以達成效果

```
i32ConversionData = ADC_GET_CONVERSION_DATA(ADC, 2);  
g_u32AdcIntFlag = 0;  
ADC_START_CONV(ADC);
```

✓ 原本的程式碼是在這個 while loop 中等待轉換完成，但這樣的缺點是在轉換的過程中會一直卡在這邊，就沒有辦法做其他事情了。

```
/* Wait ADC interrupt (g_u32AdcIntFlag will be set at IRQ_Handler function) */  
while (g_u32AdcIntFlag == 0);
```

所以這邊用 if 去判斷是否轉換完成，如果轉換完成成後才會做下面的程式碼，還沒轉換完成的時候就可以執行其他的程式碼。

```
if(g_u32AdcIntFlag==1){  
    /* Get the conversion result of the ADC channel 2 */  
    i32ConversionData = ADC_GET_CONVERSION_DATA(ADC, 2);  
    g_u32AdcIntFlag = 0;  
    ADC_START_CONV(ADC);  
}
```

<心得與收穫>

這次的實驗雖然看似簡單，但在實作過程中發現，要成功完成 ADC 的功能，必須對 ADC 工作原理及其硬體配置有深入的了解。每一個步驟都需要謹慎設定，例如模數轉換模式的選擇（單次模式或連續模式）、模擬輸入通道的選擇，以及中斷處理的正確配置等，都需要對 ADC 模組以及系統中斷機制有充分的認識。此外，如何正確啟動 ADC，管理轉換結果，以及將數據

用於後續的應用（例如 LED 控制），這些細節都非常重要。

在過程中，我學習到了如何有效設置 ADC 的參數，包括單端模式的選擇與通道啟用，還理解了中斷的重要性以及如何在轉換完成後進行結果讀取和清除中斷標誌。這些操作看似簡單，但對於精確控制模數轉換的時序和確保系統的穩定性至關重要。

同時，我也體會到系統初始化和數據轉換的時序管理是 ADC 實作中非常關鍵的部分。從啟動 ADC 電源到處理每次轉換結果的中斷，每一個步驟都要求有系統性和精確度，這樣才能確保數據的正確性並避免在轉換過程中的異常情況。

這次實驗給了我寶貴的經驗，使我對嵌入式系統中的 ADC 模組有了更深刻的理解，也明白了如何在硬體和軟體之間協調以實現準確的模數轉換。