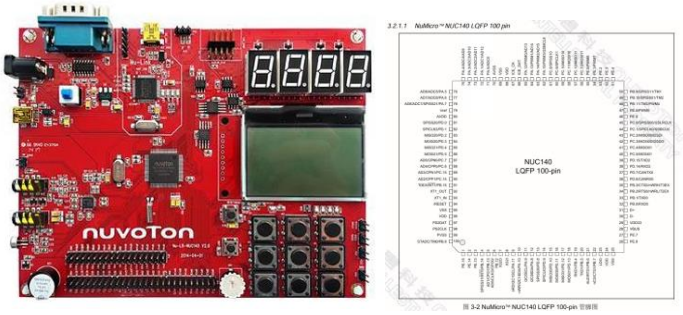
微處理機系統與介面技術 LAB 2

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 ：將鍵盤輸入的字印到 putty 上面，要在按下 Enter 時再把字印出來，並根據輸入的字串控制 RGB LED 燈。

MCU 透過 rx 端接收 pc 鍵盤按下的字元，將資料存到 MCU 的記憶體中，按下 enter 後再將這筆字串與 led 命令去比較決定 led 的開與關，最後再將這串字串藉由 tx 端一個字元一個字元的發送到 putty 上。

```
void SYS_Init(void)
{
void UART0_Init()
{
void GPIO_Init(void)
{
void ParseCommand(char *command)
{
int main(void)
{
void UART02_IRQHandler(void)
{
void UART_TEST_HANDLE()
{
```

SYS_Init、UART0_Init、GPIO_Init 負責做初始化

ParseCommand 負責比較輸入得字串是否符合 led 命令，並控制 led 開與關

UART02_IRQHandler 發生 UART 中斷時會執行裡面的程式碼

UART_TEST_HANDLE 負責處理 rx tx 傳輸

<Mian function code>

1. SYS_Init

31	30	29	28	27	26	25	24
GPB_TYPE[15:8]							
23	22	21	20	19	18	17	16
GPB_TYPE[7:0]							
15	14	13	12	11	10	9	8
GPB_MFP[15:8]							
7	6	5	4	3	2	1	0
GPB_MFP[7:0]							

[1]	GPB_MFP1	PB.1 管腳功能選擇 1 = PB.1 作為 UART0 TXD 0 = PB.1 作為 GPIOB[1]
[0]	GPB_MFP0	PB.0 管腳功能選擇 1 = PB.0 作為 UART0 RXD 0 = PB.0 作為 GPIOB[0]

```
/* Set GPB multi-function pins for UART0 RXD and TXD */
SYS->GPB_MFP &= ~(SYS_GPB_MFP_PB0_Msk | SYS_GPB_MFP_PB1_Msk);
SYS->GPB_MFP |= SYS_GPB_MFP_PB0_UART0_RXD | SYS_GPB_MFP_PB1_UART0_TXD;
```

將 GPB_MFP 暫存器的 bit0 bit1 設為 1， PB1 作為 UART0 TXD，PB.0 作為 UART0 RXD 先保留其他位元的狀態 再將 bit1 2 設為 1

2. UART0_Init

```
SYS_ResetModule(UART0_RST); /* Reset UART0 */
UART_Open(UART0, 9600);     /* Configure UART0 and set UART0 Baudrate */
UART_EnableInt(UART0, (UART_IER_RDA_IEN_Msk | UART_IER_THRE_IEN_Msk | UART_IER_TOUT_IEN_Msk));
```

這邊對 uart 做初始化，設定 baud rate、開啟中斷等等

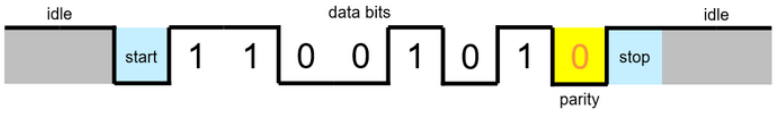
2.1 UART_Open(UART0, 9600);

將 uart0 的 baud rate 設為 9600

```
void UART_Open(UART_T* uart, uint32_t u32baudrate)
{
    uint8_t u8UartClkSrcSel, u8UartClkDivNum;
    uint32_t u32ClkTbl[4] = {__HXT, 0, 0, __HIRC};
    uint32_t u32Baud_Div = 0;

    /* Get UART clock source selection */
    u8UartClkSrcSel = (CLK->CLKSEL1 & CLK_CLKSEL1_UART_S_Msk) >> CLK_CLKSEL1_UART_S_Pos;

    /* Get UART clock divider number */
    u8UartClkDivNum = (CLK->CLKDIV & CLK_CLKDIV_UART_N_Msk) >> CLK_CLKDIV_UART_N_Pos;

    /* Select UART function */
    uart->FUN_SEL = UART_FUNC_SEL_UART;

    /* Set UART line configuration */
    uart->LCR = UART_WORD_LEN_8 | UART_PARITY_NONE | UART_STOP_BIT_1;

    /* Set UART Rx and RTS trigger level */
    uart->FCR &= ~(UART_FCR_RFITL_Msk | UART_FCR_RTS_TRI_LEV_Msk);

    /* Get PLL clock frequency if UART clock source selection is PLL */
    if(u8UartClkSrcSel == 1)
        u32ClkTbl[u8UartClkSrcSel] = CLK_GetPLLClockFreq();

    /* Set UART baud rate */
    if(u32baudrate != 0)
    {
        u32Baud_Div = UART_BAUD_MODE2_DIVIDER((u32ClkTbl[u8UartClkSrcSel]) / (u8UartClkDivNum + 1), u32baudrate);

        if(u32Baud_Div > 0xFFFF)
            uart->BAUD = (UART_BAUD_MODE0 | UART_BAUD_MODE0_DIVIDER((u32ClkTbl[u8UartClkSrcSel]) / (u8UartClkDivNum + 1), u32baudrate));
        else
            uart->BAUD = (UART_BAUD_MODE2 | u32Baud_Div);
    }
}
```

- Baud rate 的設置方式是將 UART_CLK 進行分頻之後得到我們想要的傳輸速度。
- uart->FUN_SEL = UART_FUNC_SEL_UART;
  將 UART 設置為標準 UART 模式，不啟用其他功能模式（如 IrDA 或 RS-485）。

- uart->LCR = UART_WORD_LEN_8 | UART_PARITY_NONE | UART_STOP_BIT_1;
  設置 UART 的線路參數，8 位元資料長度 (UART_WORD_LEN_8)、無奇偶校驗 (UART_PARITY_NONE)、1 個停止位 (UART_STOP_BIT_1)



這些位元共同組成了 UART 傳輸的一個完整資料包。傳輸開始時，UART 會先傳送一個 start bit，用來告知接收端即將開始傳輸資料。接著，會依序傳送 8 個資料位元，如果設定了校驗位元（parity bit），系統會根據奇偶校驗規則來檢查傳輸過程中的錯誤。最後，傳送一個或多個 stop bit，用來標誌資料傳輸的結束。

- `u32Baud_Div = UART_BAUD_MODE2_DIVIDER((u32ClkTbl[u8UartClkSrcSel]) / (u8UartClkDivNum + 1), u32baudrate);`
  計算波特率除數，根據時鐘頻率和目標 baud rate 來計算 UART 的除數值。

- 將計算出的除數寫入 UART 的暫存器，確保 UART 能夠正確地以指定的波特率運行。

```
if(u32Baud_Div > 0xFFFF)
    uart->BAUD = (UART_BAUD_MODE0 | UART_BAUD_MODE0_DIVIDER((u32ClkTbl[u8UartClkSrcSel]) / (u8UartClkDivNum + 1), u32baudrate));
else
    uart->BAUD = (UART_BAUD_MODE2 | u32Baud_Div);
```

Mode	DIV_X_EN	DIV_X_ONE	Divider X	BRD	波特率公式
0	0	0	Don't care	A	UART_CLK / [16 * (A+2)]
1	1	0	B	A	UART_CLK / [(B+1) * (A+2)] , B must >= 8
2	1	1	Don't care	A	UART_CLK / (A+2), A must >=7

2.2 UART_EnableInt (UART0, (UART_IER_RDA_IEN_Msk | UART_IER_THRE_IEN_Msk | UART_IER_TOUT_IEN_Msk));

```
void UART_EnableInt(UART_T*  uart, uint32_t u32InterruptFlag)
{

    /* Enable UART specified interrupt */
    UART_ENABLE_INT(uart, u32InterruptFlag);

    /* Enable NVIC UART IRQ */
    if(uart == UART0)
        NVIC_EnableIRQ(UART02_IRQn);
    else if(uart == UART1)
        NVIC_EnableIRQ(UART1_IRQn);
    else
        NVIC_EnableIRQ(UART02_IRQn);

}
```

```
#define UART_ENABLE_INT(uart, u32eIntSel)     ((uart)->IER |= (u32eIntSel))
```

中断使能寄存器 (UA_IER)

寄存器	偏移量	R/W	描述	复位后的值
UA_IER	UART0_BA+0x04	R/W	UART0 中断使能寄存器	0x0000_0000
	UART1_BA+0x04	R/W	UART1 中断使能寄存器	0x0000_0000
	UART2_BA+0x04	R/W	UART2 中断使能寄存器	0x0000_0000

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
DMA_RX_EN	DMA_TX_EN	AUTO_CTS_EN	AUTO_RTS_EN	TIME_OUT_EN	Reserved		LIN_RX_BRK_IEN
7	6	5	4	3	2	1	0
Reserved	WAKE_EN	BUF_ERR_IEN	RTO_IEN	MODEM_IEN	RLS_IEN	THRE_IEN	RDA_IEN

將 UA_IER 暫存器的 bit1 bit0 bit11 設為 1 開啟

[11]	TIME_OUT_EN	**Time Out 计数器使能** 1 = 使能 Time-out 计数器 0 = 禁用 Time-out 计数器

[1]	THRE_IEN	**发送保持寄存器空中断使能** 1 = 使能 INT_THRE 0 = 禁用 INT_THRE
[0]	RDA_IEN	**可接收数据中断使能** 1 = 使能 INT_RDA 0 = 禁用 INT_RDA

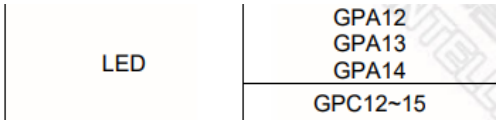
- TIME_OUT_EN

  啟用接收資料的超時檢測，防止資料傳輸中斷。

- THRE_IEN
  檢測 UART 的發送緩衝區是否空閒，通知系統可以發送新的資料。
- RDA_IEN
  檢測接收緩衝區是否有新資料可讀，通知系統讀取接收到的資料。

3. GPIO_Init

```
void GPIO_Init(void)
{
    GPIO_SetMode(PA, (BIT12|BIT13|BIT14), GPIO_PMD_OUTPUT);
    PA12 = 1;
    PA13 = 1;
    PA14 = 1;
}
```



啟用 LED，GPA12-14 打開，並將初始值設為 1(關)

4. ParseCommand

```
void ParseCommand(char *command)
{
    if (strcmp(command, "red on") == 0)
    {
        PA14 = 0;
    }
    else if (strcmp(command, "red off") == 0)
    {
        PA14 = 1;
    }
    else if (strcmp(command, "green on") == 0)
    {
        PA13 = 0;
    }
    else if (strcmp(command, "green off") == 0)
    {
        PA13 = 1;
    }
    else if (strcmp(command, "blue on") == 0)
    {
        PA12 = 0;
    }
    else if (strcmp(command, "blue off") == 0)
    {
        PA12 = 1;
    }
}
```

char *command：這是一個字串，代表要解析的命令，如 "red on"、"green off" 等。
strcmp(command, "red on") == 0：使用 strcmp 函數來比較傳入的命令字串與具體的命令，如 "red on"。如果相等，表示此命令被觸發。最後對應不同顏色的命令，函數會對控制 LED 的 GPIO 進行操作。

5. UART02_IRQHandler

```
DCD     TMR2_IRQHandler
DCD     TMR3_IRQHandler
DCD     UART02_IRQHandler
DCD     UART1_IRQHandler
DCD     SPI0_IRQHandler
```

DCD 表示「定義代碼指令」，用來將每個硬體中斷源與它的處理函數（IRQHandler）相關聯。每當發生中斷時，這個向量表會引導程序跳轉到對應的中斷處理函數。
UART02_IRQHandler，表示當 UART0 或 UART2 觸發中斷時，系統會跳轉到 此函數處理該中斷。

6. UART_TEST_HANDLE

```
uint8_t u8InChar = 0xFF;
uint32_t u32IntSts = UART0->ISR;
```

中断状态控制寄存器 (UA_ISR)

寄存器	偏移量	R/W	描述	复位后的值
UA_ISR	UART0_BA+0x1C	R/W	UART0 中断状态寄存器	0x0000_0002
	UART1_BA+0x1C	R/W	UART1 中断状态寄存器	0x0000_0002
	UART2_BA+0x1C	R/W	UART2 中断状态寄存器	0x0000_0002

31	30	29	28	27	26	25	24
HW_LIN_RX_BREAK_INT	Reserved	HW_BUF_ERR_INT	HW_TOUT_INT	HW_MODEM_INT	HW_RLS_INT	Reserved	
23	22	21	20	19	18	17	16
HW_LIN_RX_BREAK_IF	Reserved	HW_BUF_ERR_IF	HW_TOUT_IF	HW_MODEM_IF	HW_RLS_IF	Reserved	
15	14	13	12	11	10	9	8
LIN_RX_BREAK_INT	Reserved	BUF_ERR_INT	TOUT_INT	MODEM_INT	RLS_INT	THRE_INT	RDA_INT
7	6	5	4	3	2	1	0
LIN_RX_BREAK_IF	Reserved	BUF_ERR_IF	TOUT_IF	MODEM_IF	RLS_IF	THRE_IF	RDA_IF

- u8InChar：用來存儲接收到的字元。
- u32IntSts：儲存 UART 中斷狀態暫存器的值（UART0->ISR），用來判斷當前的中斷類型。
- 中斷部分:本次實驗共使用到兩個中斷

1. `if(u32IntSts & UART_ISR_RDA_INT_Msk)`

   檢測是否發生了 RDA 中斷。如果發生了接收中斷，表示 UART 接收緩衝區有資料準備好，可以讀取。

```
/* Get all the input characters */
while(UART_IS_RX_READY(UART0))
{
    /* Get the character + */
    u8InChar = UART_READ(UART0);
```

   UART_IS_RX_READY 確認 UART 是否有資料可讀，如果有資料，則進入迴圈並從 UART 讀取一個字元，並將其存入 u8InChar。

2. `if(u32IntSts & UART_ISR_THRE_INT_Msk)`

   用來檢測是否發生了 THRE(發送保持暫存器空中斷)的標誌。表示 UART 的發送緩衝區已經空了，可以接受新的資料進行傳輸。

- 在程式最前面有設置兩個 buffer，g_u8RecDate(size:1024)、cmdBuffer(size: 64)

```
uint8_t g_u8RecData[RXBUFSIZE]  = {0};

volatile uint32_t g_u32comRbytes = 0;
volatile uint32_t g_u32comRhead  = 0;
volatile uint32_t g_u32comRtail  = 0;

char cmdBuffer[CMD_BUFSIZE];
uint32 t cmdIndex = 0;
```

- ✓ cmdBuffer：存放接收到的資料，用於做字串比較

```
if(u8InChar == 0x0D)
{
    g_bEnter = TRUE;
    cmdBuffer[cmdIndex] = '\0';
    cmdIndex = 0;
    ParseCommand(cmdBuffer);

}
else
{
    if(cmdIndex < CMD_BUFSIZE - 1)
    {
        cmdBuffer[cmdIndex++] = u8InChar;
    }
    else
    {
        cmdIndex = 0;
        printf("\nCommand too long!\n");
        printf("\nInput:");
    }
}
```

當 enter 沒被按下的時候，將 rx 接收到的字元存到 cmdBuffer 中，如果超過 buffer 容量的時候，會將 index 指回 0(清空的作用)；當 enter 被按下的時候，先將 g_bEnter 設成 true，並設定 cmdBuffer[cmdIndex] = '\0'; 將字串的結尾設定為空字元，代表字串結束。接著將 cmdIndex 設為 0，讓下一次使用時可以用一個空的 buffer ParseCommand(cmdBuffer); 比較傳入的字串與 led 命令，進行 LED 的控制。

- ✓ g_u8RecDate：處理 rx tx 資料，用頭尾指標的方式去指向 buffer 的 index，當接收到資料時會先將資料放到 buffer[tail]的位置，並將 tail+1 以方便下一次放置資料；當要發送資料時，會將 buffer[head]的內容由 tx 送出，當發送完後會將 head+1，下一次發送就會發送下一個位置的資料，如果 tail==head 的話代表接收到的資料已經都被發送完了，另外當 tail head 要超過 BUFSIZE 時會將其設為 0，重新開始新的一圈。
  - ◆ Rx

```
/* Check if buffer full */
if(g_u32comRbytes < RXBUFSIZE)
{
    /* Enqueue the character */
    g_u8RecData[g_u32comRtail] = u8InChar;
    g_u32comRtail = (g_u32comRtail == (RXBUFSIZE - 1)) ? 0 : (g_u32comRtail + 1);
    g_u32comRbytes++;
}
```

  - ◆ Tx：這邊設置當 g_bEbter==true 時才會開始發送

```
if(u32IntSts & UART_ISR_THRE_INT_Msk)
{
  if(g_bEnter){
      uint16_t tmp;
      tmp = g_u32comRtail;
      if(g_u32comRhead != tmp)
      {
          u8InChar = g_u8RecData[g_u32comRhead];
          UART_WRITE(UART0, u8InChar);
          g_u32comRhead = (g_u32comRhead == (RXBUFSIZE - 1)) ? 0 : (g_u32comRhead + 1);
          g_u32comRbytes--;
      }else{
          g_bEnter = FALSE;
          printf("\nInput:");
      }
  }
}
```

- 經過這樣的流程之後，就可以達成鍵盤輸入的字印到 putty 上面，要在按下 Enter 時再把字印出來，並根據輸入的字串控制 RGB LED 燈

<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 UART 的功能，必須對 UART 通訊協定和其相關的硬體配置有深入的了解。每個步驟都需要仔細設定，例如波特率的選擇、中斷的處理方式，以及接收和傳輸的資料緩衝區管理等，都要求對 UART 模組和系統中斷控制的充分認識。此外，系統初始化和資料傳輸的時序也十分重要，這些都需要具備對 UART 硬體資源的清晰理解，才能順利進行資料通訊並確保功能正確運作。