# NUC1262 使用 LLSI 搭配 PDMA 控制 ARGB1

NuMicro® 32 位元系列微控制器範例代碼介紹

## 文件資訊

| | |
|---|---|
| 應用簡述 | 本範例代碼使用 NUC1262 系列的 LLSI 搭配 PDMA 控制 ARGB Gen 1 的 LED |
| BSP 版本 | NUC1262_Series_BSP_CMSIS_V3.00.003 |
| 開發平台 | NuMaker-NUC1262SE-V1.2 |

*The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.*

*Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.*

*All data and specifications are subject to change without notice.*

*For additional information or questions, please contact: Nuvoton Technology Corporation.*

www.nuvoton.com

# 1. 概述

本範例代碼使用 NUC1262 系列的 LLSI(LED Lighting Strip Interface)搭配 PDMA(Peripheral Direct Memory Access)來控制使用 ARGB Gen 1 協議的燈條，LLSI 為新唐 NUC1262 系列專為 ARGB 協議設計的 IP，用其他協議或 GPIO 控制 ARGB 的訊號會占用 CPU 的運算資源，使用 LLSI 搭配 PDMA 可以節省很多 CPU 的占用。

本範例代碼需要搭配使用 ARGB Gen 1 協議的燈條以觀察實際運作的燈效，也可以使用新唐開發的 NuMaker-Lighting-ARGB(介紹網址: NuMaker-Lighting-ARGB - Nuvoton)上板運行此範例代碼(如圖 1-1 )。

圖 1-1 NuMaker-Lighting-ARGB

## 1.1 原理

ARGB 協議僅需要三條硬體接線即可運作，分別是兩條電源線加上一條訊號線，標題中的 LLSI 即為 NUC1262 系列對此類訊號做輸出所設計的 IP。標準的 ARGB 為每顆燈傳送 24 bits 資料來決定此次傳輸該顆 RGB 燈應呈現的顏色，24 bits 資料包含 R、 G、 B 各 8 bits 的資料，8 bits 可以換成 0~255 的數字用以表示該顏色此次輸出應呈現的亮度。

以標準 ARGB 為例，訊號格式如圖 1-2。每個 bit 有 0 和 1 兩種表示方式， ARGB 的 0 Code 為先傳送 300 ns 的 High 接著傳送 900 ns 的 Low，ARGB 的 1 Code 則為先傳送 900 ns 的 High 接著傳送 300 ns 的 Low，連續傳送 24 bits 即為一個燈的 RGB 值。在不同的燈條上傳送 RGB 的順序可能會有所不同，本範例設定為 GRB 的順序來進行傳輸。在確定完想要的燈數量時並傳

送完每個燈的 RGB 值後，需要傳送一個 Reset Code 訊號，該訊號格式為大於 50 us 的 Low 訊號，此時整個 LED 燈條即顯示先前傳輸的 ARGB 資料的顏色。

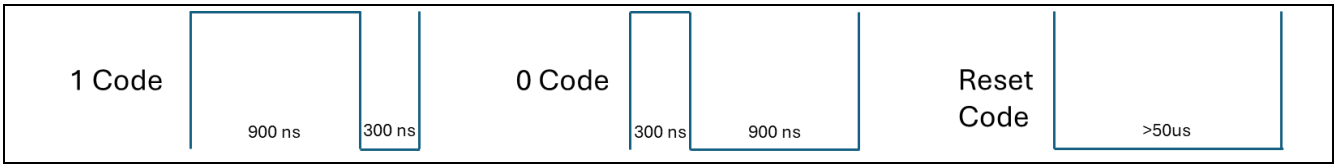


圖 1-2 ARGB 的訊號格式

不同廠商燈條的協議在 ARGB 上訊號格式可能有些許不同，LLSI 介面可以自由調整 0 Code、1 Code 和 Reset Code 的訊號格式，用以控制不同廠商的燈條。

LLSI 的傳送搭配 PDMA 以加快執行效率。PDMA 可以讓資料在不經過 CPU 即可將資料由記憶體位置 A 移到記憶體位置 B，在此應用中為將要輸出的 ARGB 資料移到 LLSI 輸出的暫存器(LLSI_DATA)。使用前需要對 PDMA 的傳輸設定值、數據來源位址、數據目標位址進行設定。

此範例代碼展示了四種燈效，分別為 Breath、Color Cycle、Rainbow Wave 和 Raindrop，下面會對四種燈效進行講解，每種燈效皆設定彩虹的 7 個顏色做循環，燈效如 1.2 章節圖片。

Breath 為呼吸燈的效果，整個燈條的亮度並非為一次線性函式的上升與下降，為了讓人眼看起來有呼吸的效果，在起始和快到最大亮度時放緩亮度的變化值。

Color Cycle 為整個燈條進行顏色的變換與漸變。計算時，先定好是以哪兩個顏色作為漸變色的兩端，接著根據當前時間使用線性方式來計算顯示顏色。

Rainbow Wave 為燈條往一個方向遞進顏色，同 Color Cycle 計算顯示顏色，Rainbow Wave 也需要根據當前時間來進行線性漸變顏色的計算。不同於 Color Cycle 的地方在於，漸變發生在同一時間的燈條上，並非整個燈條為同一個顏色，所以需要計算燈條上的每個燈需要顯示哪個顏色。

Raindrop 為多個燈往一個方向前進，此燈效需要計算每個燈需要在哪個時間點亮起，燈的顏色及亮度。

## 1.2 執行結果

本範例代碼執行編譯和燒錄的結果如圖 1-3 所示。

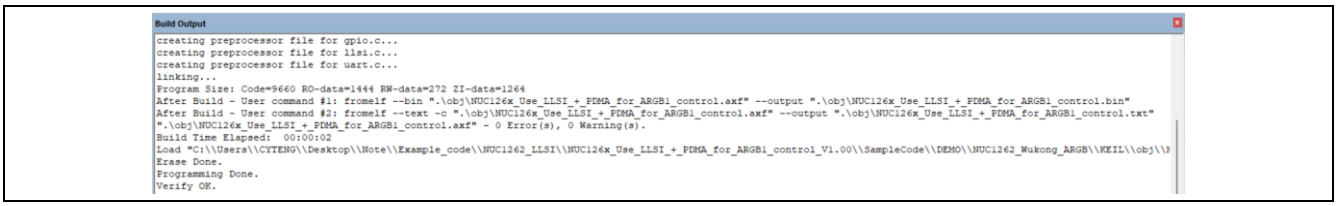


圖 1-3 執行編譯和燒錄結果

在終端機程式(ex: Tera Term)上你可以在每次切換時看到顯示的訊息，可以使用終端機程式下指令切換全部燈的燈效如圖 1-4 所示。

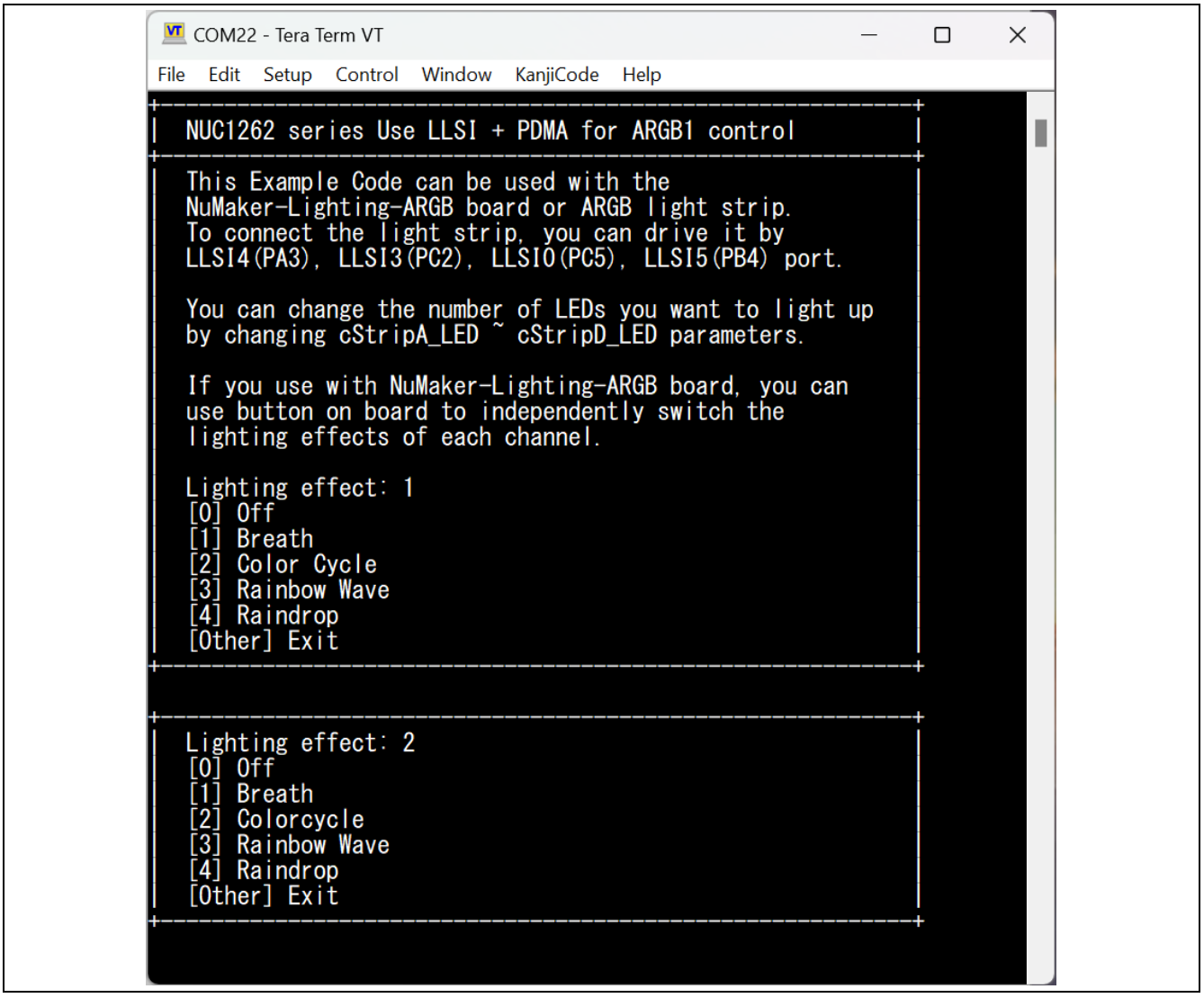


圖 1-4 終端機上顯示內容-終端機下達指令

若使用了 NuMaker-Lighting-ARGB 板，也可以利用 NuMaker-Lighting-ARGB 板上的按鈕獨立控制每個 LLSI 的燈效。每按下一次按鈕切換一次對應的 LLSI 燈效，順序為 Breath→Color Cycle→Rainbow Wave→Raindrop→Off→Breath 然後以此循環，如圖 1-5 所示。

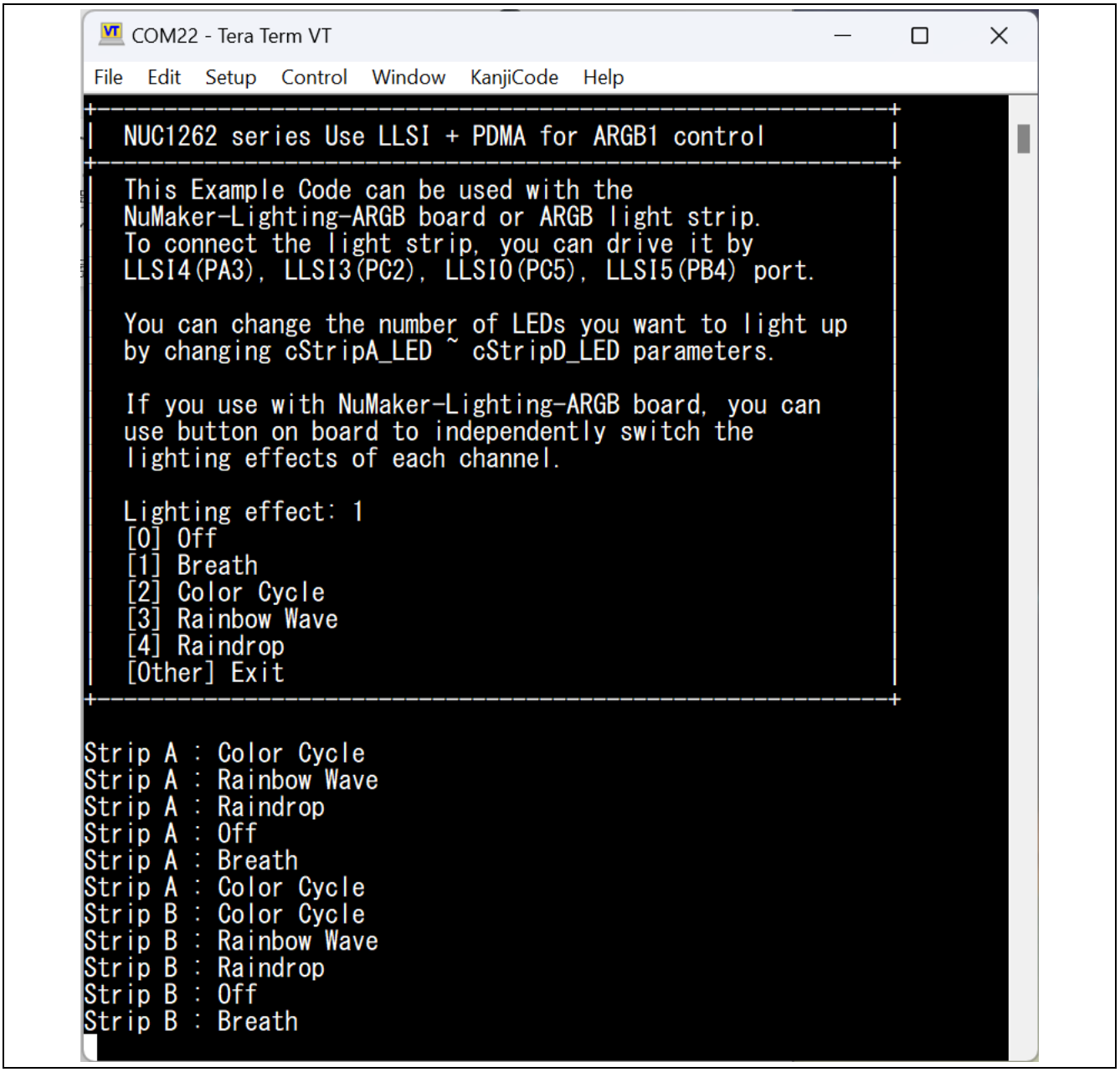


圖 1-5 終端機上顯示內容-NuMaker-Lighting-ARGB 板按鈕控制

四種燈效在 NuMaker-Lighting-ARGB 板和 ARGB 燈條上執行結果如圖 1-6~圖 1-21 所示。圖片僅截取相鄰的顏色變化作為示意，實際呈現以程式代碼運行為準。

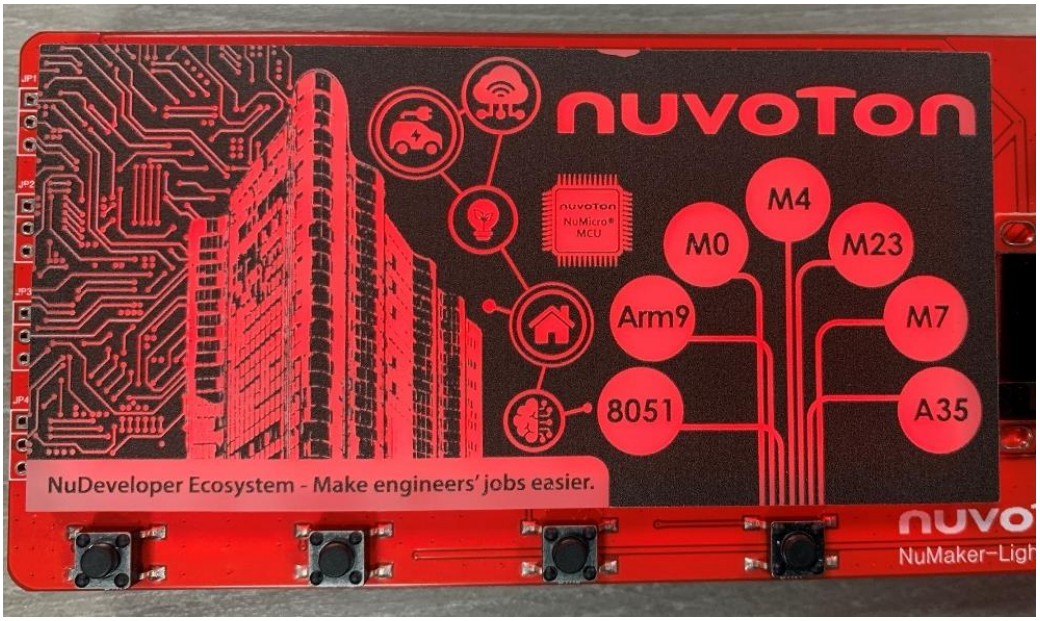


圖 1-6 ARGB 上板的 Breath 燈效(1)

圖 1-7 ARGB 上板的 Breath 燈效(2)

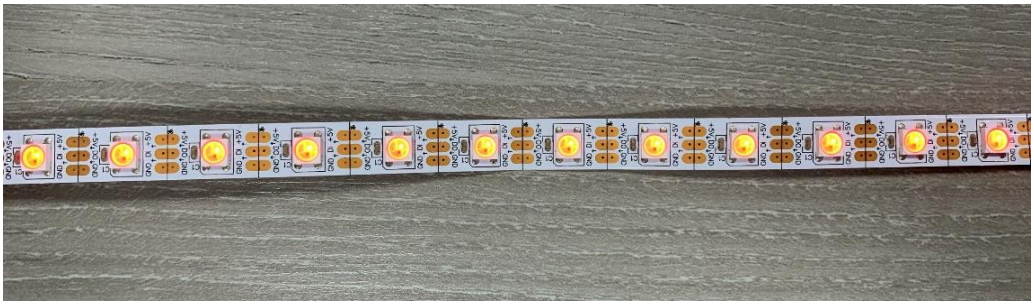

圖 1-8 燈條的 Breath 燈效(1)

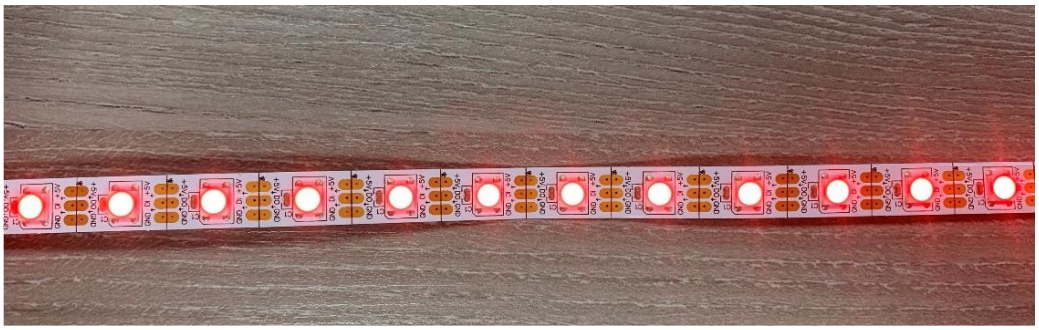

圖 1-9 燈條的 Breath 燈效(2)

圖 1-10 ARGB 上板的 Color Cycle 燈效(1)

圖 1-11 ARGB 上板的 Color Cycle 燈效(2)

圖 1-12 燈條的 Color Cycle 燈效(1)

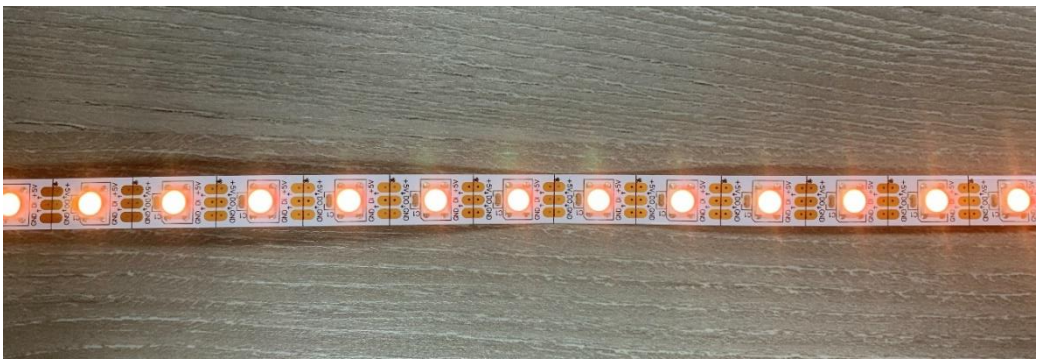

圖 1-13 燈條的 Color Cycle 燈效(2)

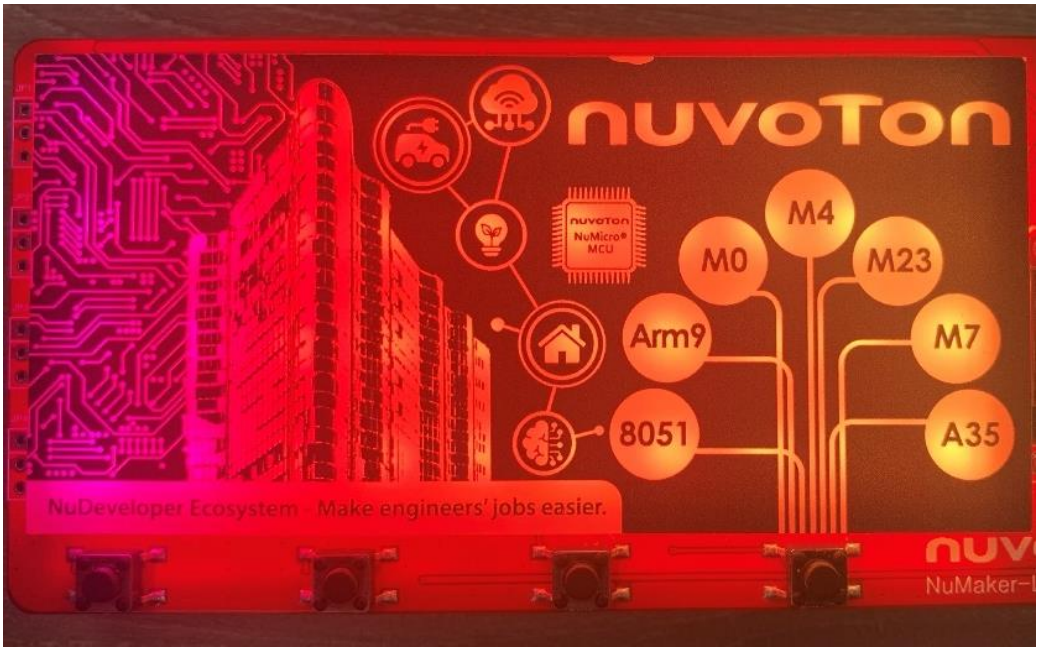


圖 1-14 ARGB 上板的 Rainbow Wave 燈效(1)

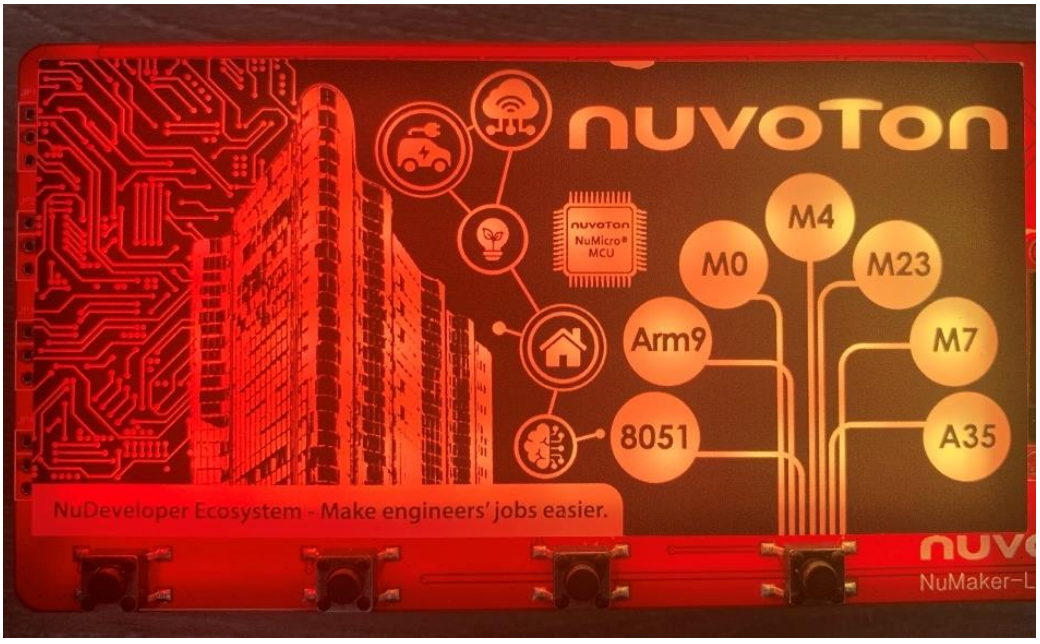


圖 1-15 ARGB 上板的 Rainbow Wave 燈效(2)

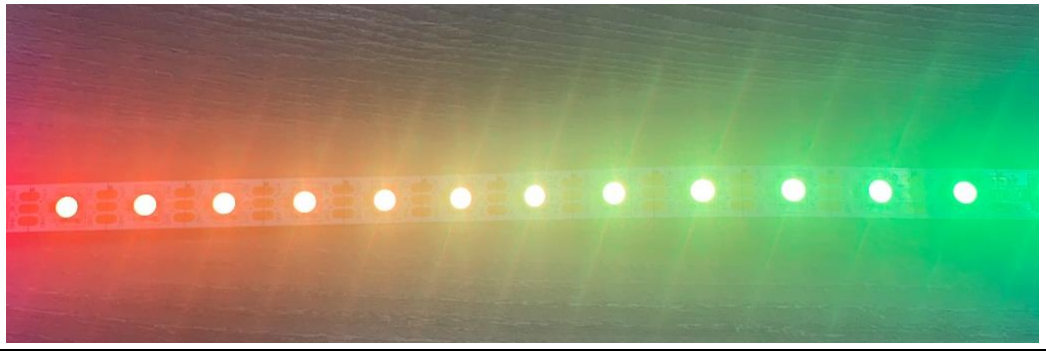

圖 1-16 燈條的 Rainbow Wave 燈效(1)

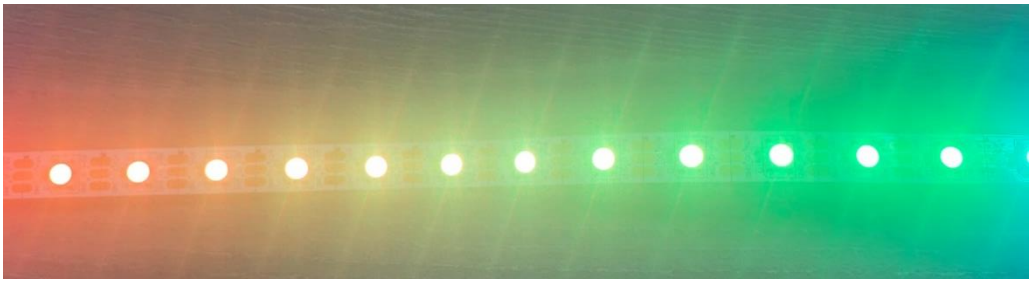

圖 1-17 燈條的 Rainbow Wave 燈效(2)

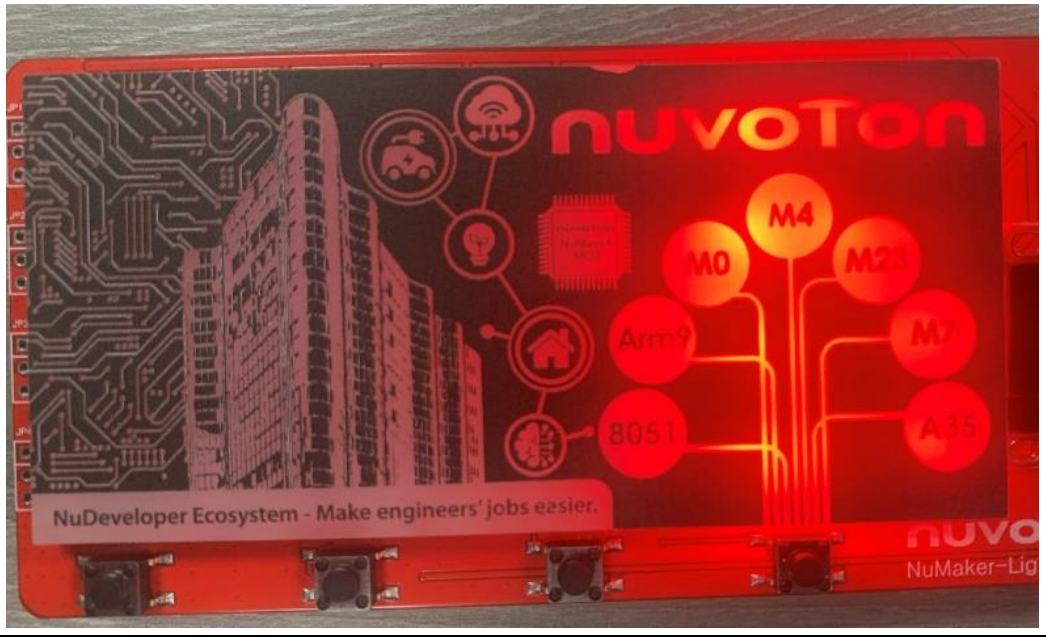


圖 1-18 ARGB 上板的 Raindrop 燈效(1)

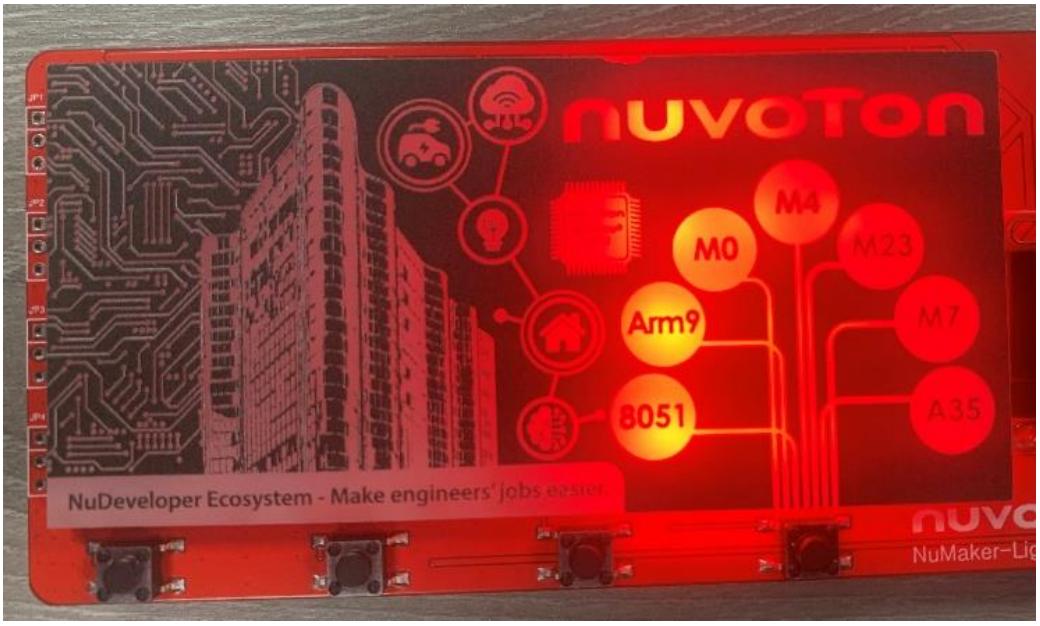


圖 1-19 ARGB 上板的 Raindrop 燈效(2)

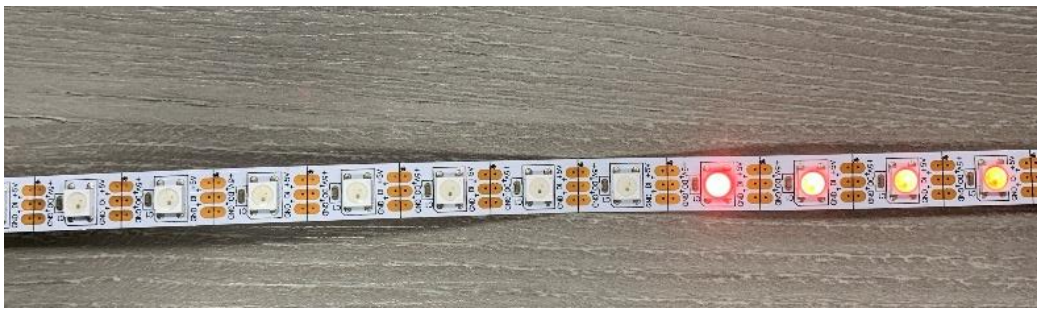

圖 1-20 燈條的 Raindrop 燈效(1)

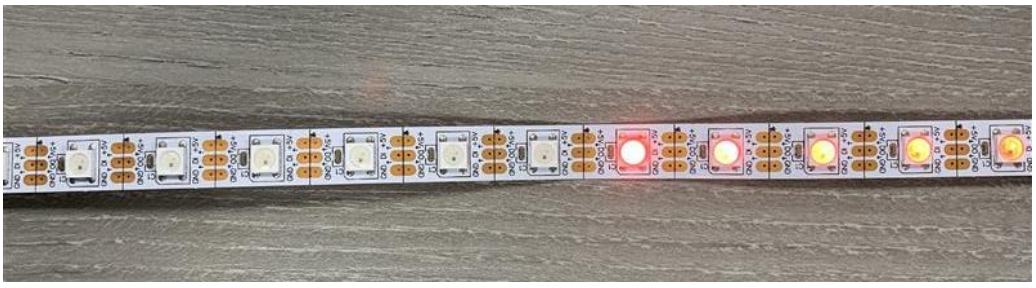

圖 1-21 燈條的 Raindrop 燈效(2)

## 2. 代碼介紹

初始化系統。代碼位於 *main.c*。

```
int32_t main(void)
{
    /* Unlock write-protected registers */
    SYS_UnlockReg();

    /* Init system and multi-function I/O */
    SYS_Init();

    /* Init UART IPs & enable interrupt */
    UART_Init();
```

對 LLSI 和 NuMaker-Lighting-ARGB 上板的按鈕進行初始化。

```
    /* Set Serial LED configuration(LLSI & PDMA) */
    LLSI_Initial();

    /* Init NuMaker-Lighting-ARGB button setting */
    Button_Initial();
```

函式 LLSI_Initial 內容，代碼位於 *LED_Control.c*。此處做 LLSI Clock、腳位、訊號格式、中斷和配合 PDMA 的設定。LLSI_Open 為 NUC1262 系列的 LLSI Library 的函式，用來對 LLSI 訊號格式的 Channel、模式、0 code、1 code 等等做設定。

```
void LLSI_Initial(void)
{
    uint8_t i;

    /* Unlock write-protected registers */
    SYS_UnlockReg();

    /* Enable module clock */
    CLK_EnableModuleClock(PDMA_MODULE);
    CLK_EnableModuleClock(LLSI4_MODULE);
    CLK_EnableModuleClock(LLSI3_MODULE);
    CLK_EnableModuleClock(LLSI0_MODULE);
    CLK_EnableModuleClock(LLSI5_MODULE);

    /*---------------------------------------------------------------------*/
    /* Init I/O Multi-function */
    /*---------------------------------------------------------------------*/
    /* Set LLSI multi-function pin */
    LLSI4_MFP_Setting();    // Strip1
    LLSI3_MFP_Setting();    // Strip2
    LLSI0_MFP_Setting();    // Strip3
    LLSI5_MFP_Setting();    // Strip4

    /* Set LLSI configuration */
    LLSI_Open(LLSI4, LLSI_MODE_SW, LLSI_FORMAT_GRB, HCLK_CLK, 1200, 300, 900, 50000,
cStripA_LED, LLSI_IDLE_LOW);
    LLSI_Open(LLSI3, LLSI_MODE_SW, LLSI_FORMAT_GRB, HCLK_CLK, 1200, 300, 900, 50000,
cStripB_LED, LLSI_IDLE_LOW);
    LLSI_Open(LLSI0, LLSI_MODE_SW, LLSI_FORMAT_GRB, HCLK_CLK, 1200, 300, 900, 50000,
cStripC_LED, LLSI_IDLE_LOW);
    LLSI_Open(LLSI5, LLSI_MODE_SW, LLSI_FORMAT_GRB, HCLK_CLK, 1200, 300, 900, 50000,
cStripD_LED, LLSI_IDLE_LOW);
```

```
    /* Set LLSI RGB format */
    for(i = 0 ; i < TOTAL_LED_AREA; i++)
    {
        if(PDMA_Mapping[i]->LED_Type == Type_RGB)
         {
            LLSI_SET_RGB_FORMAT((LLSI_T *)(APB1_BASE + 0x100000 * (PDMA_Mapping[i]-
>LLSI_Num % 2) + 0x54000 + 0x200 * (PDMA_Mapping[i]->LLSI_Num / 2)));
         }
     }

    /* Enable reset command function */
    LLSI_ENABLE_RESET_COMMAND(LLSI4);
    LLSI_ENABLE_RESET_COMMAND(LLSI3);
    LLSI_ENABLE_RESET_COMMAND(LLSI0);
    LLSI_ENABLE_RESET_COMMAND(LLSI5);

    /* Enable Reset command interrupt */
    LLSI_EnableInt(LLSI4, LLSI_RST_INT_MASK);
    LLSI_EnableInt(LLSI3, LLSI_RST_INT_MASK);
    LLSI_EnableInt(LLSI0, LLSI_RST_INT_MASK);
    LLSI_EnableInt(LLSI5, LLSI_RST_INT_MASK);

    /* Enable NVIC for LLSI */
    NVIC_EnableIRQ(LLSI4_IRQn);
    NVIC_EnableIRQ(LLSI3_IRQn);
    NVIC_EnableIRQ(LLSI0_IRQn);
    NVIC_EnableIRQ(LLSI5_IRQn);

    /* Reset PDMA module */
    SYS_ResetModule(PDMA_RST);

    /* Open Channel 0 ~ 3 */
    PDMA_Open(0x0F);

    /* PDMA Setting for LLSI */
    /* Transfer type is single transfer */
    PDMA_SetBurstType(0, PDMA_REQ_SINGLE, 0);
    PDMA_SetBurstType(1, PDMA_REQ_SINGLE, 0);
    PDMA_SetBurstType(2, PDMA_REQ_SINGLE, 0);
    PDMA_SetBurstType(3, PDMA_REQ_SINGLE, 0);

    /* Set source address, destination address */
    PDMA_SetTransferAddr(0, (uint32_t)StripALEDData, PDMA_SAR_INC, (uint32_t)&LLSI4->DATA,
PDMA_DAR_FIX);
    PDMA_SetTransferAddr(1, (uint32_t)StripBLEDData, PDMA_SAR_INC, (uint32_t)&LLSI3->DATA,
PDMA_DAR_FIX);
    PDMA_SetTransferAddr(2, (uint32_t)StripCLEDData, PDMA_SAR_INC, (uint32_t)&LLSI0->DATA,
PDMA_DAR_FIX);
    PDMA_SetTransferAddr(3, (uint32_t)StripDLEDData, PDMA_SAR_INC, (uint32_t)&LLSI5->DATA,
PDMA_DAR_FIX);

    /* Lock protected registers */
    SYS_LockReg();
}
```

函式 Button_Initial 內容，代碼位於 *Button.c*。按鈕使用 GPIO 的 Falling trigger interrupt 做觸發，此處為設定 GPIO Clock、設定腳位模式、中斷與 Debounce 功能。

```
void Button_Initial(void)
{
    /* Enable GPIOA module clock */
    CLK_EnableModuleClock(GPIOA_MODULE);

    /* Enable GPIOB module clock */
    CLK_EnableModuleClock(GPIOB_MODULE);
```

```
    /* Configure as Quasi-bidirectional mode */
    GPIO_SetMode(PB, BIT1, GPIO_MODE_QUASI);
    GPIO_SetMode(PB, BIT0, GPIO_MODE_QUASI);
    GPIO_SetMode(PB, BIT10, GPIO_MODE_QUASI);
    GPIO_SetMode(PA, BIT5, GPIO_MODE_QUASI);

    /* Enable interrupt by falling edge */
    GPIO_EnableInt(PB, 1, GPIO_INT_FALLING);
    GPIO_EnableInt(PB, 0, GPIO_INT_FALLING);
    GPIO_EnableInt(PB, 10, GPIO_INT_FALLING);
    GPIO_EnableInt(PA, 5, GPIO_INT_FALLING);
    NVIC_EnableIRQ(GPAB_IRQn);

    /* Enable interrupt de-bounce function and select de-bounce sampling cycle time is 256
clocks of LIRC clock */
    GPIO_SET_DEBOUNCE_TIME(GPIO_DBCTL_DBCLKSRC_LIRC, GPIO_DBCTL_DBCLKSEL_256);
    GPIO_ENABLE_DEBOUNCE(PB, BIT1);
    GPIO_ENABLE_DEBOUNCE(PB, BIT0);
    GPIO_ENABLE_DEBOUNCE(PB, BIT10);
    GPIO_ENABLE_DEBOUNCE(PA, BIT5);
}
```

主程式內容，輪詢 Systick 計時是否達到更新時間(此處更新時間為 25 ms)，如果達到更新時間檢查該燈效 LLSI 是否完成傳輸，完成傳輸則重新進入函式並設定 LLSI 的下一筆傳輸資料。

```
       if (LED_Frame_Update_flag == 1)
        {
            if (StripA_LEDSetting.fPDMA_Done)
            {
                /* Strip A LED data */
                StripA_LEDSetting.Mode_FUNC(&StripA_LEDSetting);

                /* Set LED data */
                Set_LED_Data(&StripA_LEDSetting);
            }
            if (StripB_LEDSetting.fPDMA_Done)
            {
                /* Strip B LED data */
                StripB_LEDSetting.Mode_FUNC(&StripB_LEDSetting);

                /* Set LED data */
                Set_LED_Data(&StripB_LEDSetting);
            }
            if (StripC_LEDSetting.fPDMA_Done)
            {
                /* Strip C LED data */
                StripC_LEDSetting.Mode_FUNC(&StripC_LEDSetting);

                /* Set LED data */
                Set_LED_Data(&StripC_LEDSetting);
            }

            if (StripD_LEDSetting.fPDMA_Done)
            {
                /* Strip D LED data */
                StripD_LEDSetting.Mode_FUNC(&StripD_LEDSetting);

                /* Set LED data */
                Set_LED_Data(&StripD_LEDSetting);
            }

            /* Clear flag */
            LED_Frame_Update_flag = 0;
        }
```

中斷函式 SysTick_Handler 內容，此處對燈效的運作有兩個功能，一個是更新主程式輪詢的更新 Flag，另一個是增加燈效更新的計數。

```
 void SysTick_Handler(void)
{
    /* Clear interrupt flag */
    SysTick->VAL = 0;

    /* Add LED mode counter */
    TimeCounterFrameUpdate++;
    StripA_LEDSetting.TimeCounter++;
    StripB_LEDSetting.TimeCounter++;
    StripC_LEDSetting.TimeCounter++;
    StripD_LEDSetting.TimeCounter++;

    /* Update LED frame data */
    if (TimeCounterFrameUpdate >= TIMER_FRAME_UPDATE)
    {
        LED_Frame_Update_flag = 1;
        TimeCounterFrameUpdate -= TIMER_FRAME_UPDATE;
    }
}
```

函式 Set_LED_Data 內容，代碼位於 *LED_Control.c*。用於將 ARGB 資料透過 PDMA 給 LLSI 腳位傳送出去。讓 PDMA 傳送資料到 LLSI 前需要設定傳輸的內容，包括資料大小、資料數量、PDMA 連接的 IP 和 LLSI 的模式。

```
void Set_LED_Data(volatile struct LED_Setting_Tag* LED_Setting)
{
    uint32_t u32Count;

    /* Calculate transfer count */
    u32Count = LED_Setting->Array_Size / 4;
    if(LED_Setting->Array_Size % 4)
        u32Count++;

    /* Set transfer count */
    PDMA_SetTransferCnt(LED_Setting->PDMA_Num, PDMA_WIDTH_32, u32Count);

    /* Set request source */
    PDMA_SetTransferMode(LED_Setting->PDMA_Num, 36 + LED_Setting->LLSI_Num, FALSE, 0);

    /* Change LLSI mode */
    LLSI_SET_PDMA_MODE((LLSI_T *)(APB1_BASE + 0x100000 * (LED_Setting->LLSI_Num % 2) +
0x54000 + 0x200 * (LED_Setting->LLSI_Num / 2)));

    /* Clear done flag */
    LED_Setting->fPDMA_Done = 0;
}
```

燈效函式 Breath 內容，此處利用 Systick 紀錄時間，透過 HDIV_Mod(取餘數函式)、HDIV_Div(除法函式)來計算當前燈光的亮度，BreathingBright 為呼吸燈經過一次完整週期的亮度變化陣列，BreathingArraySize 則為其陣列大小，由計算得出當前應該要在 BreathingBright 陣列中第幾個元素的亮度。TimeCounter 大於一個燈效的週期時，歸零 TimeCounter 且變為下個顏色。計算完成後，透過 Set_Single 函式將計算結果儲存並由 LLSI 發送。

```
void FUNC_Breath(volatile struct LED_Setting_Tag* LED_Setting)
{
    uint32_t Temp, TempR, TempG, TempB;

    /* Calculate Color */
    Temp = HDIV_Mod(HDIV_Div(LED_Setting->TimeCounter, Breath_C_E_B_D +
HDIV_Div(LED_Setting->Speed, Breath_C_E_S_I_P)), BreathingArraySize);
    TempR = HDIV_Div((LED_Setting->Color_R[LED_Setting->Choose_Flag] * LED_Setting-
>Brightness * *(BreathingBright + Temp)), Max_Speed * 100);
    TempG = HDIV_Div((LED_Setting->Color_G[LED_Setting->Choose_Flag] * LED_Setting-
>Brightness * *(BreathingBright + Temp)), Max_Speed * 100);
    TempB = HDIV_Div((LED_Setting->Color_B[LED_Setting->Choose_Flag] * LED_Setting-
>Brightness * *(BreathingBright + Temp)), Max_Speed * 100);

    /* Mapping Color to LED Format */
    Set_Single(LED_Setting->LED_Data, LED_Setting->LEDNum, TempR, TempG, TempB);

    /* Reset CountingTime */
    if(LED_Setting->TimeCounter >= ((Breath_C_E_B_D + HDIV_Div(LED_Setting->Speed,
Breath_C_E_S_I_P)) * BreathingArraySize))
    {
        LED_Setting->TimeCounter -= (Breath_C_E_B_D + HDIV_Div(LED_Setting->Speed,
Breath_C_E_S_I_P)) * BreathingArraySize;
        LED_Setting->Choose_Flag++;
        LED_Setting->Choose_Flag = (LED_Setting->Choose_Flag) % (LED_Setting->ColorNum);
    }

}
```

函式 Set_Single 內容，將資料存到設定好 LLSI 搭配 PDMA 的陣列中，會用 Set_LED_Data 函式將資料由 LLSI 發送出去。

```
void Set_Single(uint8_t *LED_DATA, uint32_t TotalLED, uint8_t Data_R, uint8_t Data_G,
uint8_t Data_B)
{
    uint32_t i;

    for(i = 0; i < TotalLED; i++)
    {
        *(LED_DATA + i * 3 + 0) = Data_R;
        *(LED_DATA + i * 3 + 1) = Data_G;
        *(LED_DATA + i * 3 + 2) = Data_B;
    }
}
```

燈效函式 Color Cycle 內容，正向和反向僅差在漸變順序不同。為達成此燈效需先確定漸變色兩端的顏色，Color1 和 Color2 陣列根據 TimeCounter 決定漸變色兩端顏色的 RGB 值。由 DisplayColor 陣列儲存最終由 LLSI 傳送出去的資料，計算則是由 CurrentTime(當前處於漸變色兩端的相對時間)和 DeltaT(到達漸變色兩端需要經過的時間)兩個變數決定 Color1 和 Color2 各自所占的比例。在 TimeCounter 經過一次循環的時間後，將此變數歸零。

```
void FUNC_Color_Cycle(volatile struct LED_Setting_Tag* LED_Setting)
{
    uint32_t DeltaT = Color_Cycle_C_D_B_N  + LED_Setting->Speed * Color_Cycle_C_D_S_I_P;
    uint32_t CurrentTime;
    uint8_t Color1[3];
    uint8_t Color2[3];
    uint8_t DisplayColor[3];
    uint32_t Temp;

        if(LED_Setting->Direction == Dir_Forward)
```

```
    {
        /* Take Color1 and Color2 from Rainbow array by TimeCounter */
        Temp = HDIV_Mod(HDIV_Div(LED_Setting->TimeCounter, DeltaT), LED_Setting-
>ColorNum);

        /* Set Color1 */
        Color1[0] = LED_Setting->Color_R[Temp];
        Color1[1] = LED_Setting->Color_G[Temp];
        Color1[2] = LED_Setting->Color_B[Temp];

        /* Set Color2 */
        Color2[0] = LED_Setting->Color_R[(Temp + 1) % LED_Setting->ColorNum];
        Color2[1] = LED_Setting->Color_G[(Temp + 1) % LED_Setting->ColorNum];
        Color2[2] = LED_Setting->Color_B[(Temp + 1) % LED_Setting->ColorNum];
    }
    else if(LED_Setting->Direction == Dir_Backward)
    {
        /* Take Color1 and Color2 from Rainbow array by TimeCounter */
        Temp = HDIV_Mod(HDIV_Div(LED_Setting->TimeCounter, DeltaT), LED_Setting-
>ColorNum);

        /* Set Color1 */
        Color1[0] =  LED_Setting->Color_R[LED_Setting->ColorNum - 1 - Temp];
        Color1[1] =  LED_Setting->Color_G[LED_Setting->ColorNum - 1 - Temp];
        Color1[2] =  LED_Setting->Color_B[LED_Setting->ColorNum - 1 - Temp];

        /* Set Color2 */
        Color2[0] = LED_Setting->Color_R[HDIV_Mod(((LED_Setting->ColorNum) + (LED_Setting-
>ColorNum) - 2 - Temp), LED_Setting->ColorNum)];
        Color2[1] = LED_Setting->Color_G[HDIV_Mod(((LED_Setting->ColorNum) + (LED_Setting-
>ColorNum) - 2 - Temp), LED_Setting->ColorNum)];
        Color2[2] = LED_Setting->Color_B[HDIV_Mod(((LED_Setting->ColorNum) + (LED_Setting-
>ColorNum) - 2 - Temp), LED_Setting->ColorNum)];

    }

    /* Reset CountingTime */
    if(LED_Setting->TimeCounter >= (DeltaT * LED_Setting->ColorNum))
    {
        LED_Setting->TimeCounter -= (DeltaT * LED_Setting->ColorNum);
    }

    /* Judge current time and set mixed Displaycolor by color1 and color2 */
    CurrentTime = HDIV_Mod(LED_Setting->TimeCounter, DeltaT);
    for(uint8_t i = 0; i < 3; i++)
        DisplayColor[i] = HDIV_Div(Color1[i] * (DeltaT - CurrentTime), DeltaT) +
HDIV_Div(Color2[i] * CurrentTime, DeltaT);

    /* Mapping Color to LED Format */
    Set_Single(LED_Setting->LED_Data, LED_Setting->LEDNum, DisplayColor[0],
DisplayColor[1], DisplayColor[2]);
}
```

圖 2-1、圖 2-2 為該段程式的可視化說明。TimeCounter 持續計時，DisplayColor 為當前顯示的顏色，該顏色由 CurrentTime 位置的兩個顏色混合所組成(圖 2-2)，判斷在哪兩個顏色之間可以用 TimeCounter 除以 DeltaT 的商數得出，Temp 記錄為哪兩個顏色。而兩個顏色各自所佔的比例可以用 CurrentTime 和 Delta 的計算得出。在 TimeCounter 大於 DeltaT 乘以 ColorNum 後歸零。

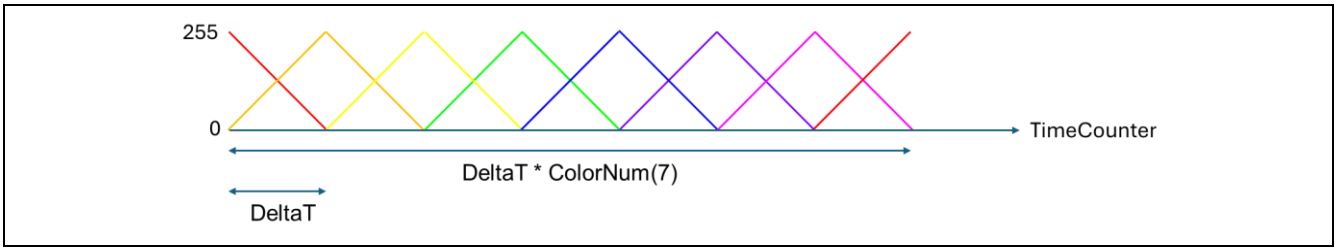


圖 2-1 Color Cycle 說明(1)

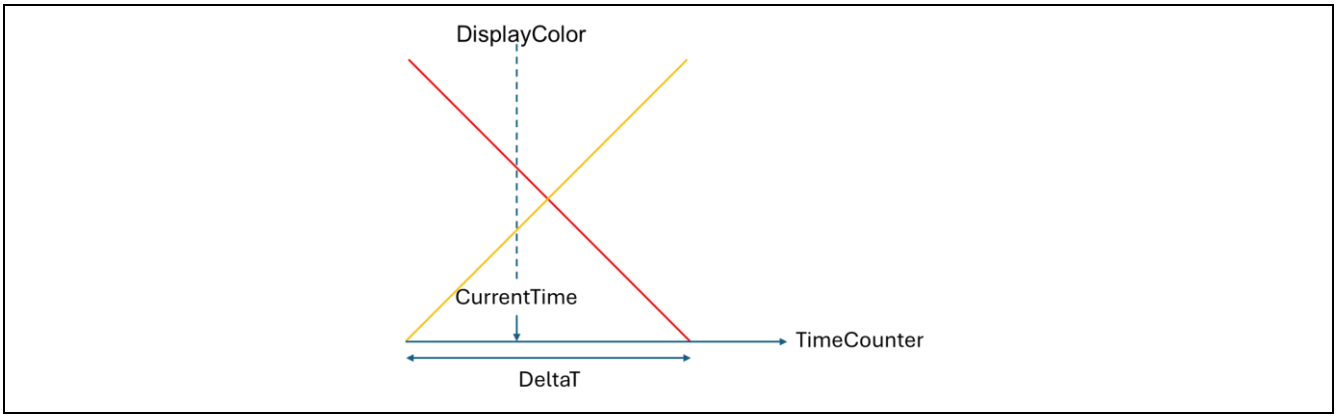


圖 2-2 Color Cycle 說明(2)

燈效函式 Rainbow Wave 內容，TotalDuration 變數為全部顏色循環所需的時間，NextTime 變數為用來計算每個燈應呈現的顏色，CurrentTime 即為此刻的時間點，deltaT 變數為漸變色兩端所經過的時間。以上的變數在 for 迴圈中皆會用做計算每個燈的 RGB 值，漸變一樣採用線性，根據 CurrentTime 數值在 deltaT 的位置對漸變色兩端的比例進行計算，最後得出應顯示的值。不同於 Color Cycle，此處須把每個燈所應顯示不同的顏色做計算，因此需要根據 LED_Num 變數(燈條上進行燈效的燈數)做數次計算。計算好後由 DisplayColor 陣列中儲存最終由 LLSI 傳送出去的資料。在 TimeCounter 經過一次循環的時間後，將此變數歸零。

```
void FUNC_Rainbow_Wave(volatile struct LED_Setting_Tag* LED_Setting)
{
    uint8_t DisplayColor[LED_Setting->LEDNum][3];
    static uint32_t i, j;
    uint32_t deltaT, TotalDuration, CurrentTime, NextTime;
    uint8_t ColorCursor1, ColorCursor2;

    /* Init Array */
    for(j = 0; j < LED_Setting->LEDNum; j++)
    {
        for(i = 0; i < 3; i++)
            DisplayColor[j][i] = 0;
    }

    /* Duration for one time effect */
    TotalDuration = HDIV_Div(LED_Setting->ColorNum * Rainbow_Wave_C_T_S_C_F_L_M  *
Rainbow_Wave_C_T_S_C_F_L_B / Rainbow_Wave_C_T_C_T_P, HDIV_Div(Max_Speed - LED_Setting-
>Speed, Rainbow_Wave_C_T_S_I_P) + Rainbow_Wave_C_T_B_D);
```

```
    /* Next LED time */
    NextTime = HDIV_Div(TotalDuration, LED_Setting->ColorNum *
Rainbow_Wave_C_T_S_C_F_L_M/Rainbow_Wave_C_T_C_T_P);

    /* Current Time */
    CurrentTime = HDIV_Mod(LED_Setting->TimeCounter, TotalDuration);

    /* Unit time for each color */
    deltaT = HDIV_Div(TotalDuration, LED_Setting->ColorNum);

    /* Reset CountingTime */
    if(LED_Setting->TimeCounter >= TotalDuration)
    {
        LED_Setting->TimeCounter -= TotalDuration;
    }

    /* Mix Color */
    for(j = 0; j < LED_Setting->LEDNum; j++, CurrentTime += NextTime)
    {
        if(CurrentTime >= TotalDuration)
            CurrentTime -= TotalDuration;

        if(CurrentTime >= (deltaT * LED_Setting->ColorNum))
        {
            /* Color Cursor */
            ColorCursor1 = 0;
            ColorCursor2 = 1;

            DisplayColor[LED_Setting->LEDNum - j - 1][0] = LED_Setting-
>Color_R[ColorCursor1];
            DisplayColor[LED_Setting->LEDNum - j - 1][1] = LED_Setting-
>Color_G[ColorCursor1];
            DisplayColor[LED_Setting->LEDNum - j - 1][2] = LED_Setting-
>Color_B[ColorCursor1];
        }
        else
        {
            /* Color Cursor */
            ColorCursor1 = HDIV_Mod(HDIV_Div(CurrentTime, deltaT), LED_Setting->ColorNum);
            ColorCursor2 = HDIV_Mod((ColorCursor1 + 1), LED_Setting->ColorNum);

            DisplayColor[LED_Setting->LEDNum - j - 1][0] = HDIV_Div(((LED_Setting-
>Color_R[ColorCursor1] * ((ColorCursor1 + 1) * deltaT - CurrentTime)) + \
                (LED_Setting->Color_R[ColorCursor2] * (CurrentTime - ColorCursor1 *
deltaT))), deltaT);
            DisplayColor[LED_Setting->LEDNum - j - 1][1] = HDIV_Div(((LED_Setting-
>Color_G[ColorCursor1] * ((ColorCursor1 + 1) * deltaT - CurrentTime)) + \
                (LED_Setting->Color_G[ColorCursor2] * (CurrentTime - ColorCursor1 *
deltaT))), deltaT);
            DisplayColor[LED_Setting->LEDNum - j - 1][2] = HDIV_Div(((LED_Setting-
>Color_B[ColorCursor1] * ((ColorCursor1 + 1) * deltaT - CurrentTime)) + \
                (LED_Setting->Color_B[ColorCursor2] * (CurrentTime - ColorCursor1 *
deltaT))), deltaT);
        }

    }

    /* Mapping Color to LED Format */
    /* Direction forward or backward */
    if(LED_Setting->Direction == Dir_Forward)
        Set_Array(LED_Setting->LED_Data, LED_Setting->LEDNum, (uint8_t *)DisplayColor,
LED_Setting->Brightness);
    else if(LED_Setting->Direction == Dir_Backward)
        Set_InverseArray(LED_Setting->LED_Data, LED_Setting->LEDNum, (uint8_t
*)DisplayColor, LED_Setting->Brightness);
}
```

圖 2-3、圖 2-4 和圖 2-5 為該段程式的可視化說明。TimeCounter 持續計時，DisplayColor 為顯示當前的顏色。不同於 Color Cycle，此處為一個根據燈條上的燈數所創建的二維陣列。先計算 CurrentTime 的值，並計算下一個燈更新的計數值(NextTime)，由此來得出每個燈條上的燈珠所顯示的顏色。因此最後得出的 DisplayColor 值會是該燈條上每個燈珠呈現的 RGB 值，並非跟 Color Cycle 一樣只有單一 RGB 值。每一個燈所呈現顏色計算方式同 Color Cycle，為線性方式計算。

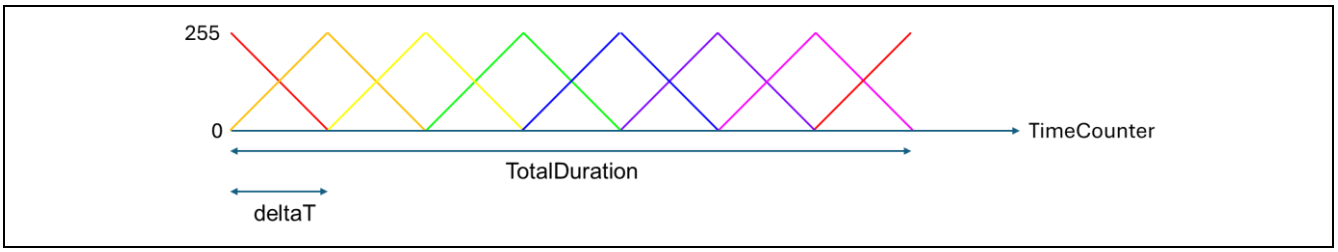


圖 2-3 Rainbow Wave 說明(1)

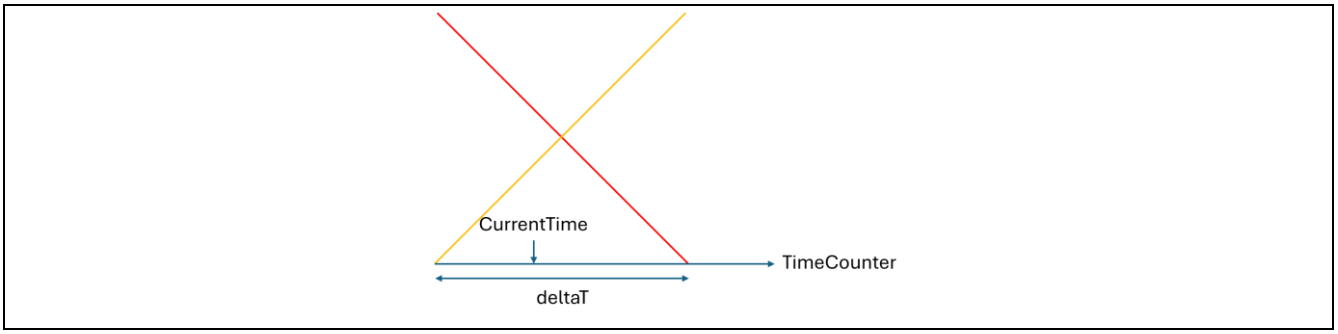


圖 2-4 Rainbow Wave 說明(2)

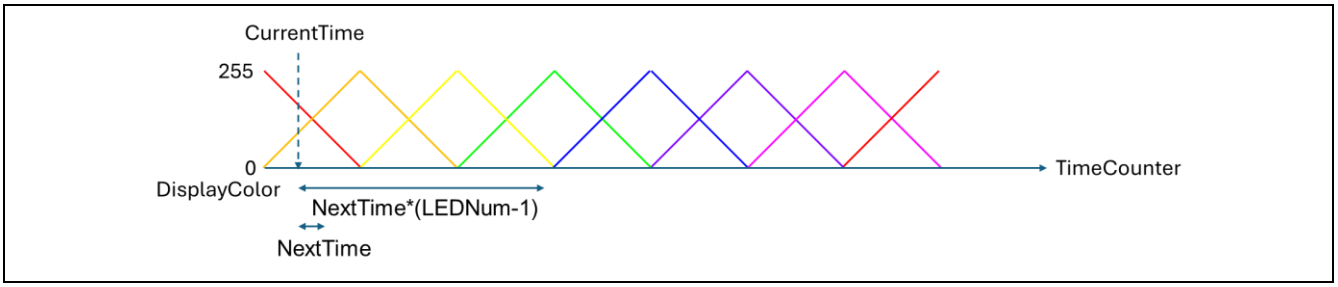


圖 2-5 Rainbow Wave 說明(3)

函式 Set_Array 內容，此函式功能與 Set_Single 函式相似，將資料存到設定好 LLSI 搭配 PDMA 的陣列中，會用前面的 Set_LED_Data 函式將資料由 LLSI 發送出去。與 Set_Single 函式不同的是，此處每個數值皆有可能不一樣，因此需要一一對應將資料填入。

```
void Set_Array(uint8_t *LED_DATA, uint32_t TotalLED, uint8_t *DisplayData, uint8_t
MaxBright)
{
```

```
    uint32_t i;
    uint8_t TempR, TempG, TempB;

    for(i = 0; i < TotalLED; i++)
    {
        TempR = HDIV_Div(*(DisplayData + i * 3 + 0) * MaxBright, 0xFF);
        TempG = HDIV_Div(*(DisplayData + i * 3 + 1) * MaxBright, 0xFF);
        TempB = HDIV_Div(*(DisplayData + i * 3 + 2) * MaxBright, 0xFF);
        *(LED_DATA + i * 3 + 0) = TempR;
        *(LED_DATA + i * 3 + 1) = TempG;
        *(LED_DATA + i * 3 + 2) = TempB;
    }
}
```

燈效函式 Raindrop 內容，DeltaT 變數為燈移動一次的時間，Duration 變數為燈完整的走完一次燈條的時間，DelayT 變數為在燈條全暗時加延時以讓燈不要不間斷地出現，Raindrop_Trailing 為製造拖尾效果的陣列。用 if 判斷現在是燈還在移動的時間或是燈條全暗的時間，並進入 for 迴圈設定此刻哪個燈為全亮、哪個燈為拖尾亮度、哪個燈為暗。連續更新燈條上的燈，就可以做出燈往一個方向移動的效果。在 TimeCounter 經過一次循環的時間後，將此變數歸零。

```
void FUNC_Raindrop(volatile struct LED_Setting_Tag* LED_Setting)
{
    uint8_t DisplayColor[LED_Setting->LEDNum + 2 * LED_Setting->LED_Length + 6][3]; //+4
    uint8_t Color[5][3];
    uint8_t Raindrop_i, Raindrop_j;
    uint32_t Duration;
    uint32_t Loop, DeltaT, DelayT;
    uint32_t TempColorR, TempColorG, TempColorB;

    /* Init Array */
    for (Raindrop_j = 0; Raindrop_j < LED_Setting->LEDNum + 2 * LED_Setting->LED_Length +
5; Raindrop_j++)
    {
        for (Raindrop_i = 0; Raindrop_i < 3; Raindrop_i++)
            DisplayColor[Raindrop_j][Raindrop_i] = 0;
    }

    for (Raindrop_j = 0; Raindrop_j < 5; Raindrop_j++)
    {
        for (Raindrop_i = 0; Raindrop_i < 3; Raindrop_i++)
            Color[Raindrop_j][Raindrop_i] = 0;
    }

    /* DeltaT is unit time */
    DeltaT = Raindrop_C_D_B_N + HDIV_Div(LED_Setting->Speed, Raindrop_C_D_S_I_P);

    /* Duration is one time pass through */
    Duration = DeltaT * (LED_Setting->LEDNum  + LED_Setting->LED_Length * 2);

    /* DelayT is delay on two ends */
    DelayT = HDIV_Div(LED_Setting->LEDNum * DeltaT, 2);

    /* Reset CountingTime */
    if (LED_Setting->TimeCounter >= ((Duration + DelayT)))
    {
        LED_Setting->TimeCounter -= (Duration + DelayT);
        LED_Setting->Choose_Flag++;
        LED_Setting->Choose_Flag = LED_Setting->Choose_Flag % LED_Setting->ColorNum;
    }

    if (LED_Setting->TimeCounter < Duration)
    {
```

```
        /* Loop to indicate which LED for first Color */
        Loop = HDIV_Div(LED_Setting->TimeCounter, DeltaT);

        TempColorR = LED_Setting->Color_R[LED_Setting->Choose_Flag];
        TempColorG = LED_Setting->Color_G[LED_Setting->Choose_Flag];
        TempColorB = LED_Setting->Color_B[LED_Setting->Choose_Flag];

        for (Raindrop_i = 0; Raindrop_i < LED_Setting->LED_Length; Raindrop_i++)
        {
            Color[Raindrop_i][0] = TempColorR * Raindrop_Trailing[Raindrop_i] / 100;
            Color[Raindrop_i][1] = TempColorG * Raindrop_Trailing[Raindrop_i] / 100;
            Color[Raindrop_i][2] = TempColorB * Raindrop_Trailing[Raindrop_i] / 100;
        }

        for (Raindrop_i = LED_Setting->LED_Length; Raindrop_i < LED_Setting->LED_Length +
1 ; Raindrop_i++)
        {
            Color[Raindrop_i][0] = TempColorR ;
            Color[Raindrop_i][1] = TempColorG ;
            Color[Raindrop_i][2] = TempColorB ;
        }

        for (Raindrop_j = Loop; Raindrop_j < (Loop + 5); Raindrop_j++)
        {
            for (Raindrop_i = 0; Raindrop_i < 3; Raindrop_i++)
            {
                DisplayColor[Raindrop_j][Raindrop_i] = Color[Raindrop_j  -
Loop ][Raindrop_i];
            }
        }
    }
    /* All Extinguish */
    else if (LED_Setting->TimeCounter < (Duration + DelayT))
    {
        for (Raindrop_j = 0; Raindrop_j < LED_Setting->LEDNum + 2 * LED_Setting-
>LED_Length; Raindrop_j++)
        {
            for (Raindrop_i = 0; Raindrop_i < 3; Raindrop_i++)
            {
                DisplayColor[Raindrop_j][Raindrop_i] = 0;
            }
        }
    }

    /* Mapping Color to LED Format */
    /* Direction forward or backward */
    if (LED_Setting->Direction == Dir_Forward)
        Set_Array(LED_Setting->LED_Data, LED_Setting->LEDNum, (uint8_t *)(DisplayColor +
5), LED_Setting->Brightness);
    else if (LED_Setting->Direction == Dir_Backward)
        Set_InverseArray(LED_Setting->LED_Data, LED_Setting->LEDNum, (uint8_t
*)(DisplayColor + 5), LED_Setting->Brightness);
}
```

# 3. 軟體與硬體需求

## 3.1 軟體需求

- BSP 版本
  - NUC1262_Series_BSP_CMSIS_V3.00.003
- IDE 版本
  - Keil uVision 5.39

## 3.2 硬體需求

- 電路元件
  - NuMaker-NUC1262SE V1.2
- 線路示意圖
  - 將 UART0 TX (PB.13) pin 腳連接到 PC UART RX，以顯示範例代碼的執行結果。再來可視是否有 NuMaker-Lighting-ARGB 上板執行操作。
  - 若有 NuMaker-Lighting-ARGB 上板，將上板接上 NuMaker-NUC1262SE 的 UNO 介面即可（圖 3-1）。用上板自帶的按鈕或是 PC 端執行燈效變換。
  - 若沒有 NuMaker-Lighting-ARGB 上板，使用協議為 ARGB 的燈條，接上 UNO 介面上的 5VCC 和 VSS 電源腳位，以及訊號腳位（可選擇 PA.3、PC.2、PC.5、PB.4）連接（圖 3-2）。用 PC 端執行燈效變換。

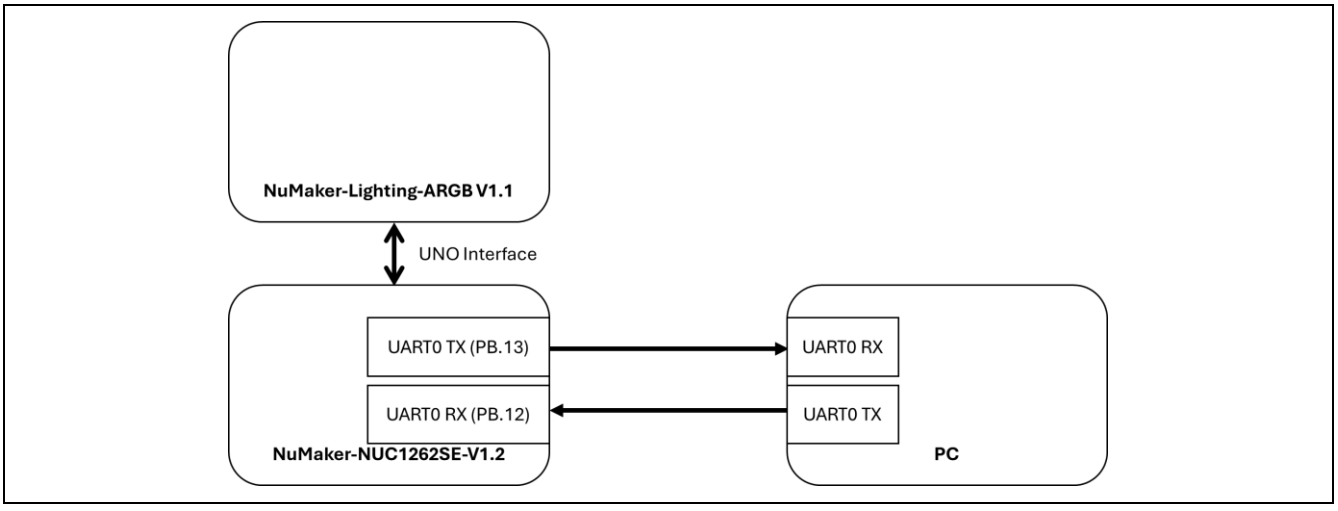


圖 3-1 線路示意圖(1)

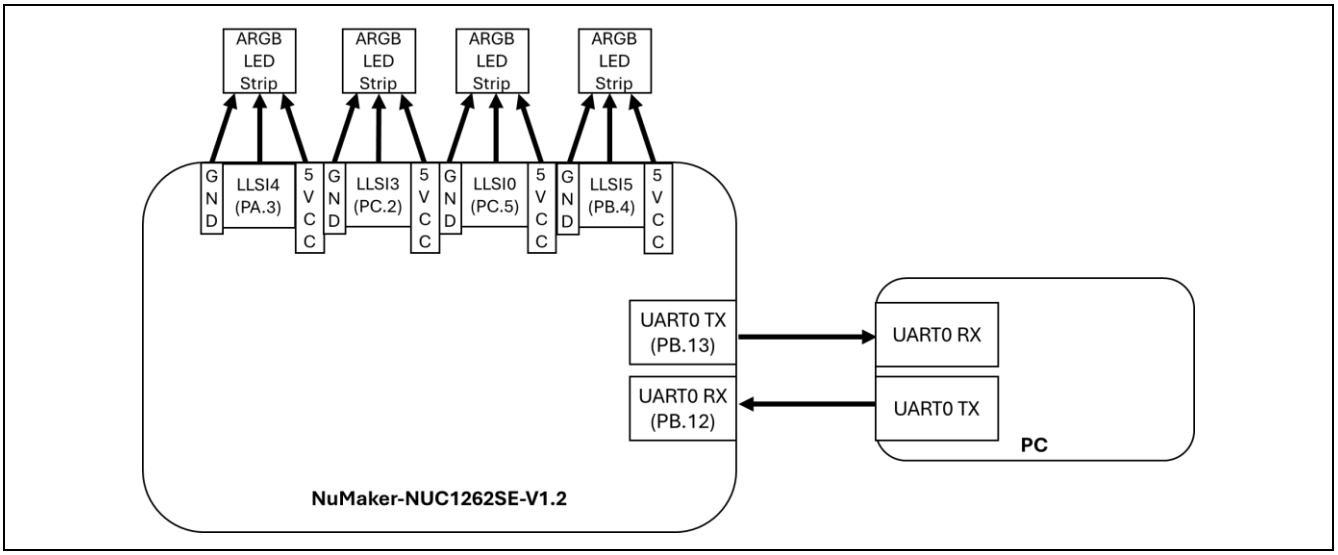
ARGB LED Strip
ARGB LED Strip
ARGB LED Strip
ARGB LED Strip
GND
LLSI4 (PA.3)
5VCC
GND
LLSI3 (PC.2)
5VCC
GND
LLSI0 (PC.5)
5VCC
GND
LLSI5 (PB.4)
5VCC
UART0 TX (PB.13)
UART0 RX (PB.12)
UART0 RX
UART0 TX
PC
NuMaker-NUC1262SE-V1.2

圖 3-2 線路示意圖(2)

# 4. 目錄資訊

目錄結構如下圖所示。

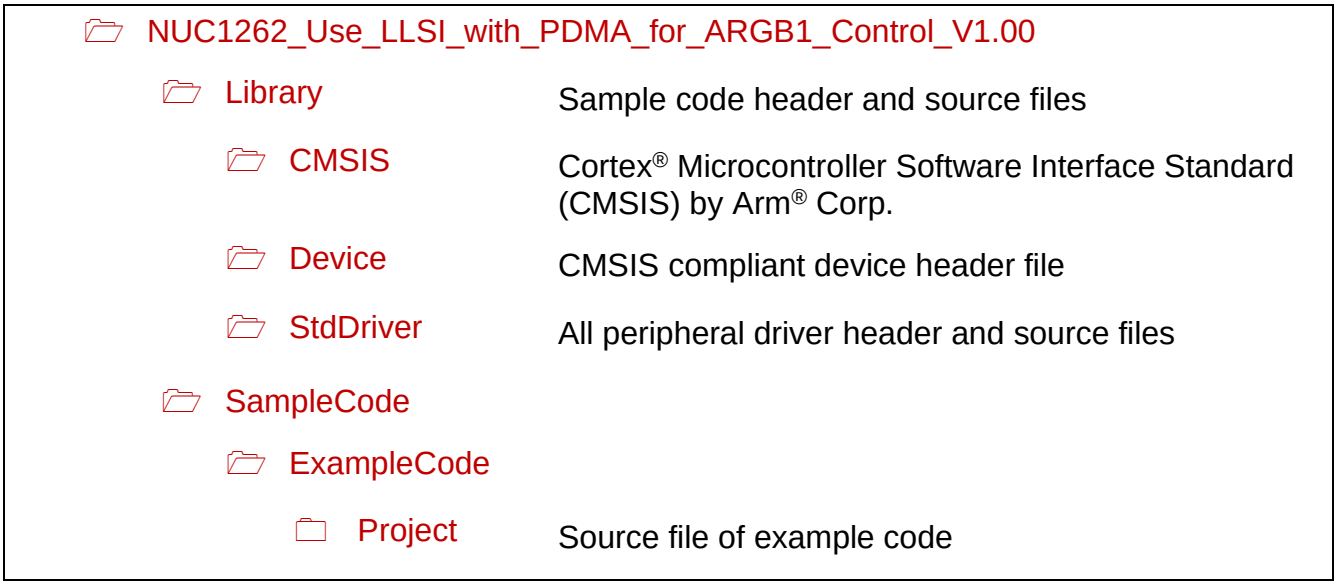


圖 4-1 目錄資訊

# 5. 範例程式執行

1. 根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，雙擊 *NUC1262_Use_LLSI_with_PDMA_for_ARGB1_Control.uvprojx*。
2. 進入編譯模式介面
   - 編譯
   - 下載代碼至記憶體
   - 進入 / 離開除錯模式
3. 進入除錯模式介面
   - 執行代碼
4. 使用終端機程式(ex: Tera Term)，根據 UART 回傳的訊息發送指令來切換燈效。

# 6. 修訂紀錄

| Date | Revision | Description |
|---|---|---|
| 2025.04.03 | 1.00 | 初始發布。 |

## Important Notice

**Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".**

**Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.**

**All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.**

*Please note that all data and specifications are subject to change without notice.*
*All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.*