NUC1262 使用 LLSI 搭配 PDMA 控制 ARGB1

NuMicro® 32 位系列微控制器范例代码介绍

文件信息

应用简述	本范例代码使用 NUC1262 系列的 LLSI 搭配 PDMA 控制 ARGB Gen 1 的 LED 灯
BSP 版本	NUC1262_Series_BSP_CMSIS_V3.00.003
开发平台	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

本范例代码使用 NUC1262 系列的 LLSI(LED Lighting Strip Interface)搭配 PDMA(Peripheral Direct Memory Access)来控制使用 ARGB Gen 1 协议的灯条，LLSI 为新唐 NUC1262 系列专为 ARGB 协议设计的 IP，用其他协议或 GPIO 控制 ARGB 的信号会占用 CPU 的运算资源，使用 LLSI 搭配 PDMA 可以节省很多 CPU 的占用。

本范例代码需要搭配使用 ARGB Gen 1 协议的灯条以观察实际运作的灯效，也可以使用新唐开发的 NuMaker-Lighting-ARGB(介绍网址: NuMaker-Lighting-ARGB - Nuvoton)上板运行此范例代码(如图 1-1)。

图 1-1 NuMaker-Lighting-ARGB

1.1 原理

ARGB 协议仅需要三条硬件接线即可运作，分别是两条电源线加上一条信号线，标题中的 LLSI 即为 NUC1262 系列对此类信号做输出所设计的 IP。标准的 ARGB 为每颗灯传送 24 bits 数据来决定此次传输该颗 RGB 灯应呈现的颜色，24 bits 数据报含 R、G、B 各 8 bits 的数据，8 bits 可以换成 0~255 的数字用以表示该颜色此次输出应呈现的亮度。

以标准 ARGB 为例，信号格式如图 1-2。每个 bit 有 0 和 1 两种表示方式， ARGB 的 0 Code 为先传送 300 ns 的 High 接着传送 900 ns 的 Low，ARGB 的 1 Code 则为先传送 900 ns 的 High 接着传送 300 ns 的 Low，连续传送 24 bits 即为一个灯的 RGB 值。在不同的灯条上传送 RGB 的顺序可能会有所不同，本范例设定为 GRB 的顺序来进行传输。在确定完想要的灯数量时并传送完每个灯的 RGB 值后，需要传送一个 Reset Code 信号，该信号格式为大于 50 us 的 Low 信号，此时整个 LED 灯条即显示先前传输的 ARGB 资料的颜色。

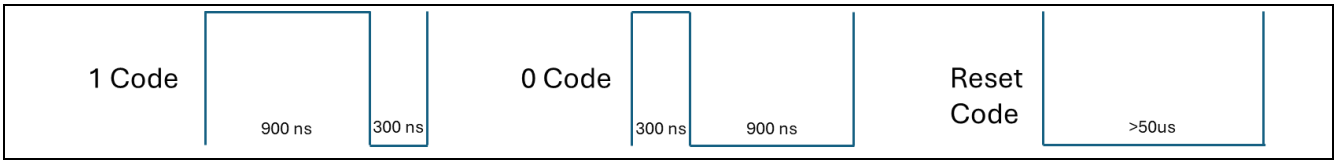

图 1-2 ARGB 的信号格式

不同厂商灯条的协议在 ARGB 上信号格式可能有些许不同，LLSI 接口可以自由调整 0 Code、1 Code 和 Reset Code 的信号格式，用以控制不同厂商的灯条。

LLSI 的传送搭配 PDMA 以加快执行效率。PDMA 可以让数据在不经过 CPU 即可将数据由内存位置 A 移到内存位置 B，在此应用中为将要输出的 ARGB 数据移到 LLSI 输出的寄存器(LLSI_DATA)。使用前需要对 PDMA 的传输设定值、数据来源地址、数据目标地址进行设定。

此范例代码展示了四种灯效，分别为 Breath、Color Cycle、Rainbow Wave 和 Raindrop，下面会对四种灯效进行讲解，每种灯效皆设定彩虹的 7 个颜色做循环，灯效如 1.2 章节图片。

Breath 为呼吸灯的效果，整个灯条的亮度并非为一次线性函数的上升与下降，为了让人眼看起来有呼吸的效果，在起始和快到最大亮度时放缓亮度的变化值。

Color Cycle 为整个灯条进行颜色的变换与渐变。计算时，先定好是以哪两个颜色作为渐变色的两端，接着根据当前时间使用线性方式来计算显示颜色。

Rainbow Wave 为灯条往一个方向递进颜色，同 Color Cycle 计算显示颜色，Rainbow Wave 也需要根据当前时间来进行线性渐变颜色的计算。不同于 Color Cycle 的地方在于，渐变发生在同一时间的灯条上，并非整个灯条为同一个颜色，所以需要计算灯条上的每个灯需要显示哪个颜色。

Raindrop 为多个灯往一个方向前进，此灯效需要计算每个灯需要在哪个时间点亮起，灯的颜色及亮度。

1.2 执行结果

本范例代码执行编译和烧录的结果如图 1-3 所示。

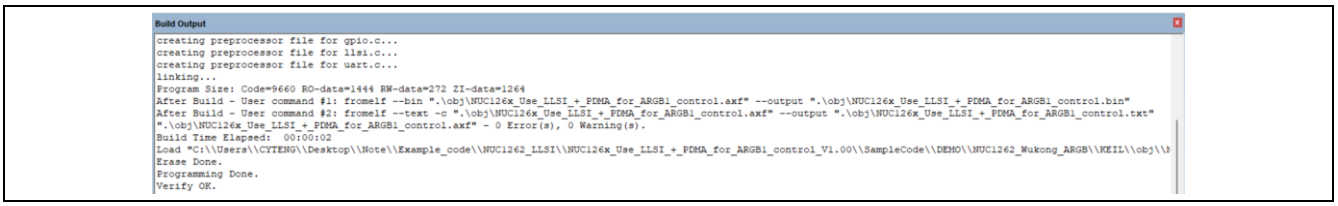

图 1-3 执行编译和烧录结果

在终端机程序(ex: Tera Term)上你可以在每次切换时看到显示的讯息，可以使用终端机程序下指令切换全部灯的灯效如图 1-4 所示。

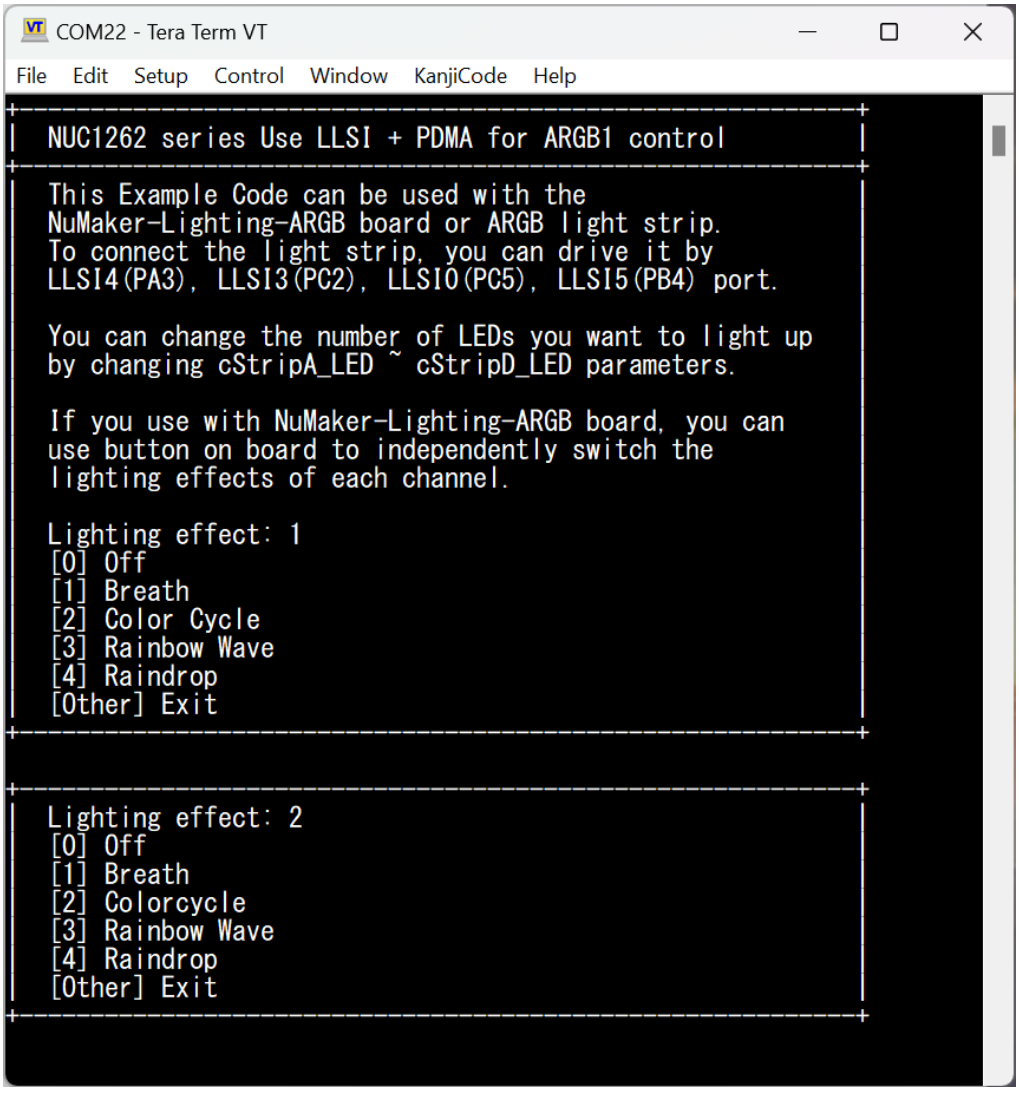

图 1-4 终端机上显示内容-终端机下达指令

若使用了 NuMaker-Lighting-ARGB 板，也可以利用 NuMaker-Lighting-ARGB 板上的按钮独立控制每个 LLSI 的灯效。每按下一次按钮切换一次对应的 LLSI 灯效，顺序为 Breath→Color Cycle→Rainbow Wave→Raindrop→Off→Breath 然后以此循环，如图 1-5 所示。

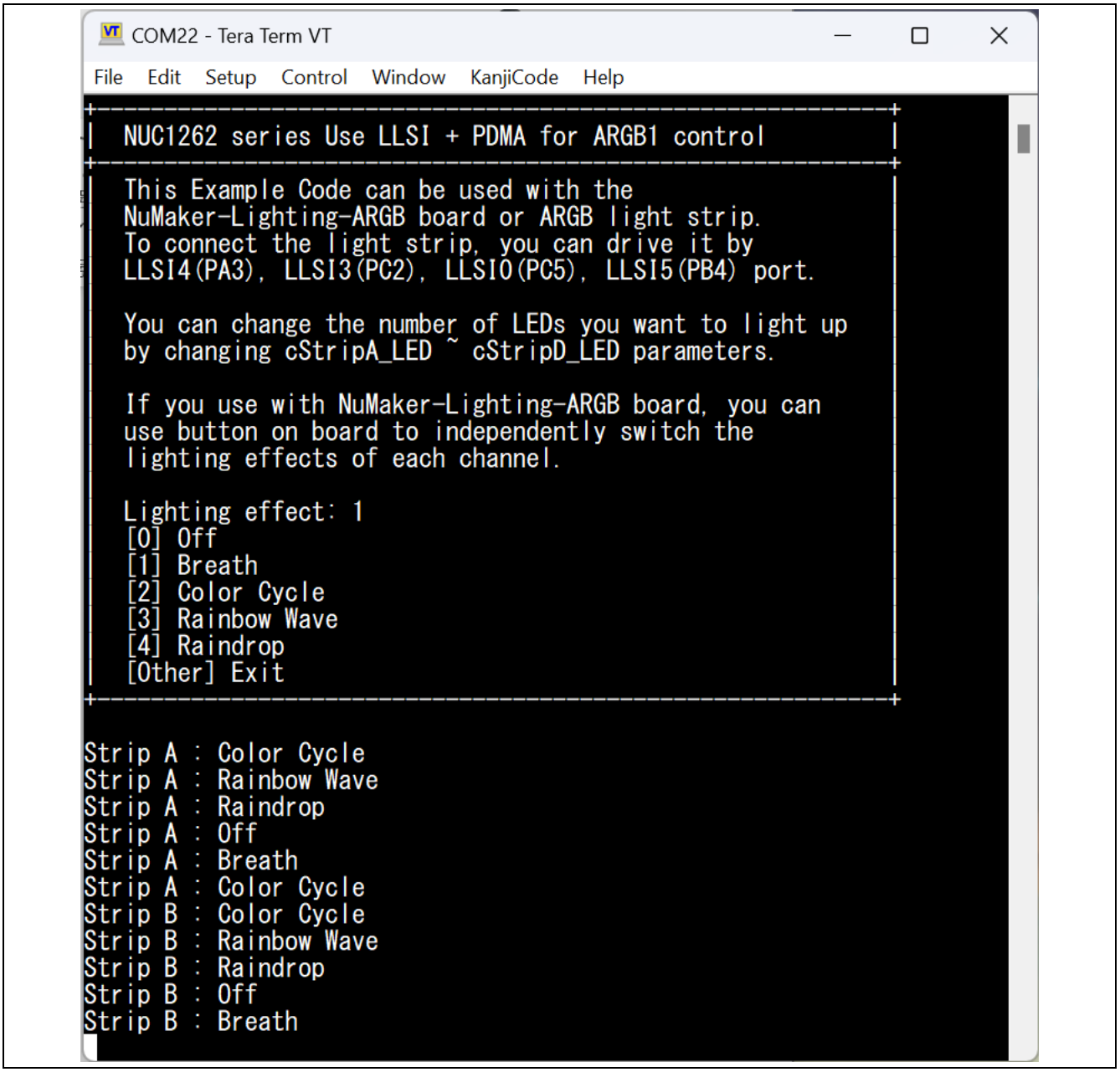

图 1-5 终端机上显示内容-NuMaker-Lighting-ARGB 板按钮控制

四种灯效在 NuMaker-Lighting-ARGB 板和 ARGB 灯条上执行结果如图 1-6~图 1-21 所示。图片仅截取相邻的颜色变化作为示意，实际呈现以程序代码运行为准。

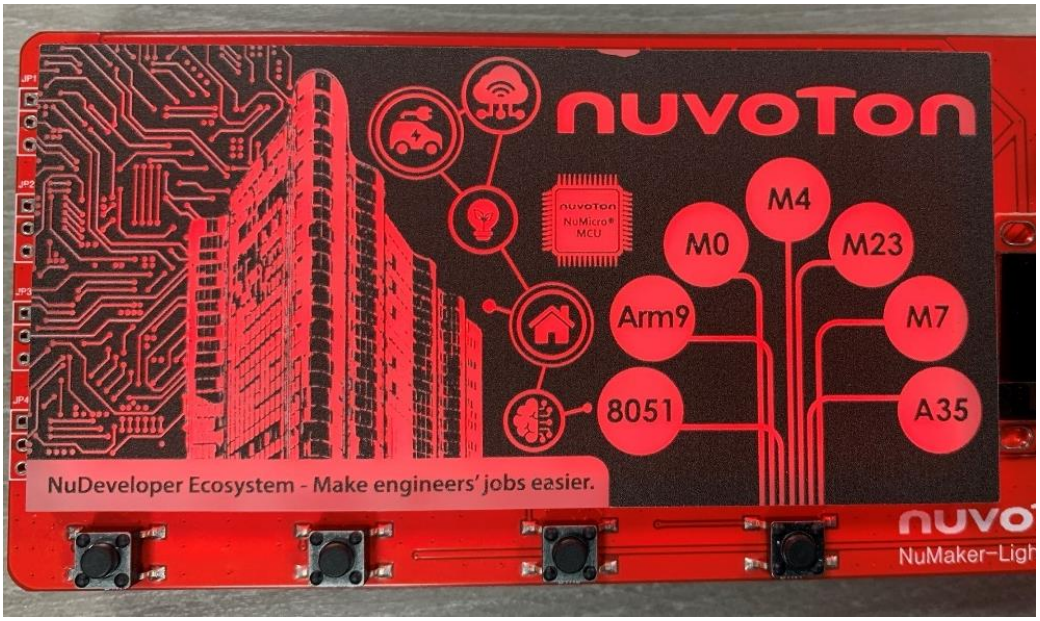

图 1-6 ARGB 上板的 Breath 灯效(1)

图 1-7 ARGB 上板的 Breath 灯效(2)

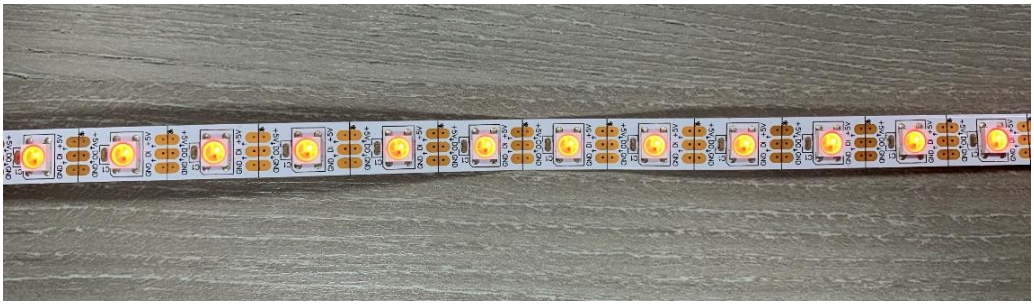

图 1-8 灯条的 Breath 灯效(1)

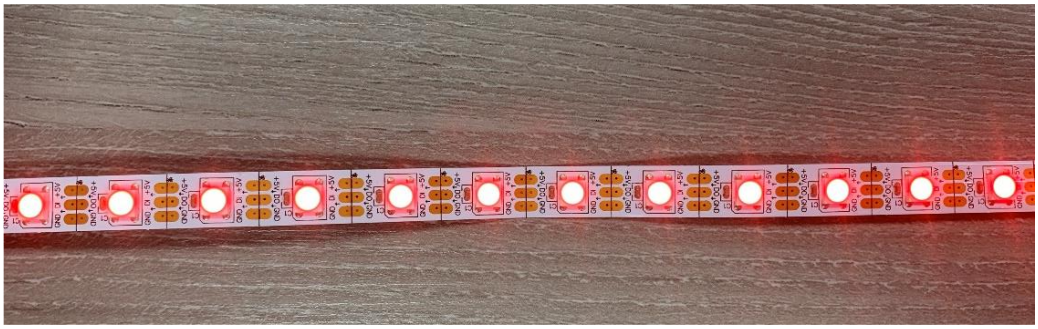

图 1-9 灯条的 Breath 灯效(2)

图 1-10 ARGB 上板的 Color Cycle 灯效(1)

图 1-11 ARGB 上板的 Color Cycle 灯效(2)

图 1-12 灯条的 Color Cycle 灯效(1)

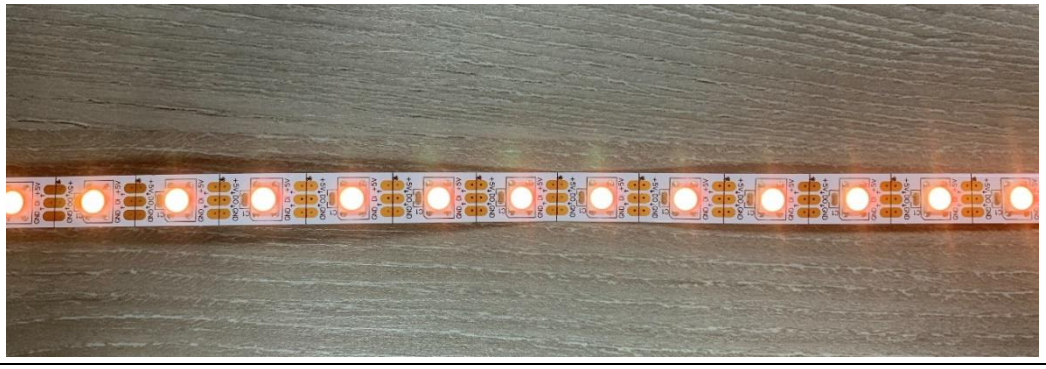

图 1-13 灯条的 Color Cycle 灯效(2)

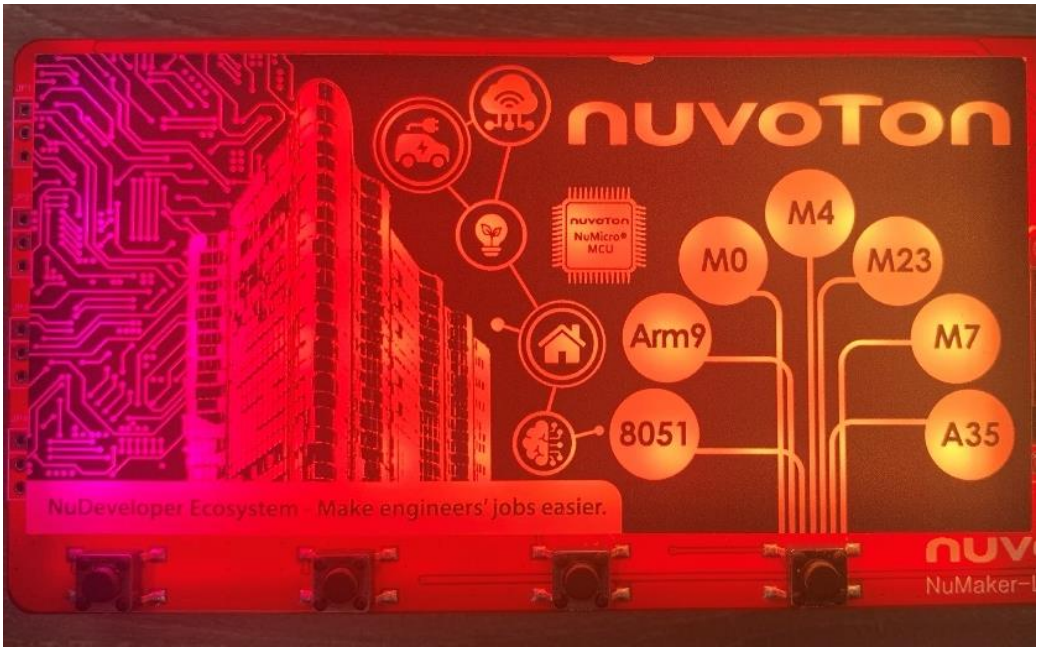

图 1-14 ARGB 上板的 Rainbow Wave 灯效(1)

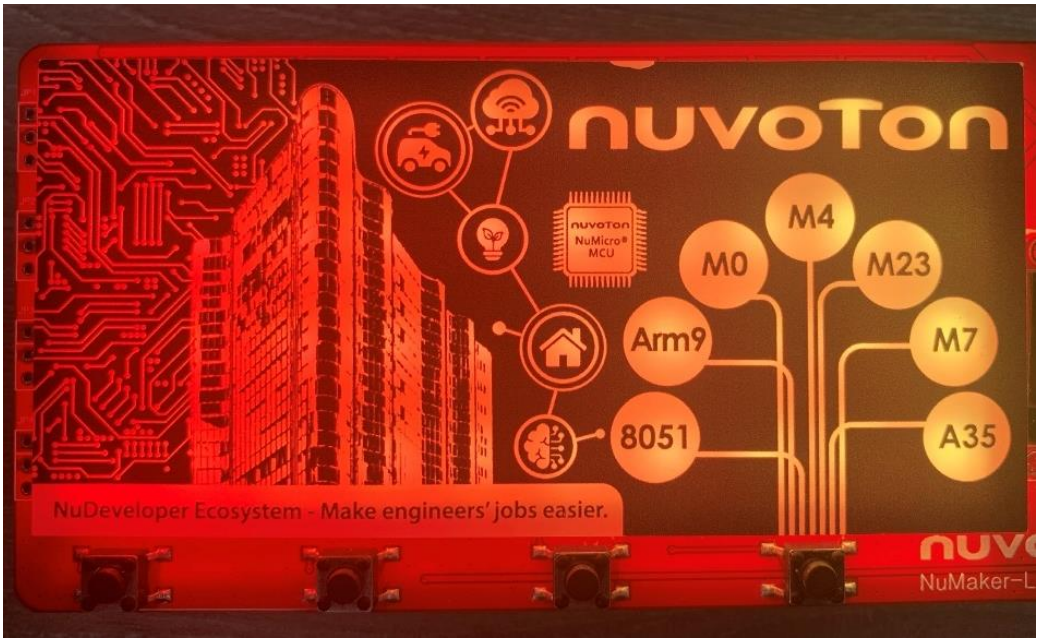

图 1-15 ARGB 上板的 Rainbow Wave 灯效(2)

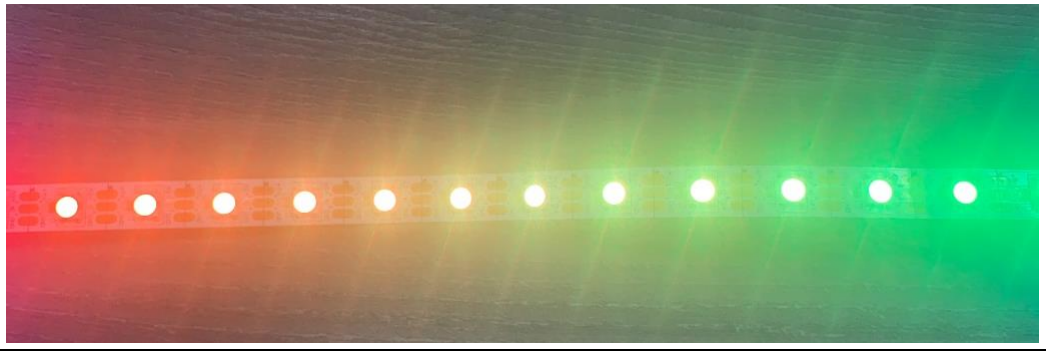

图 1-16 灯条的 Rainbow Wave 灯效(1)

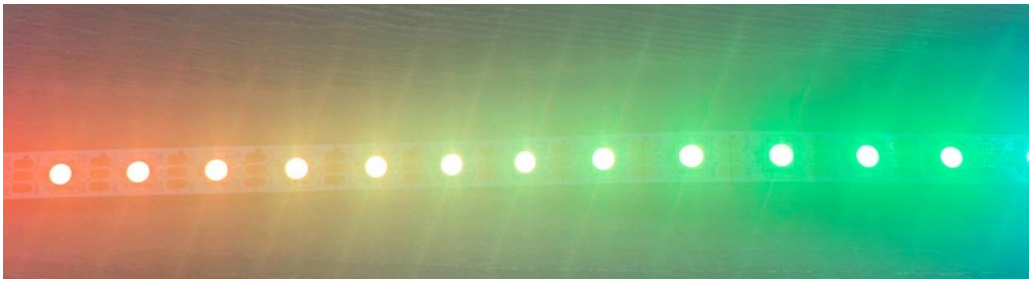

图 1-17 灯条的 Rainbow Wave 灯效(2)

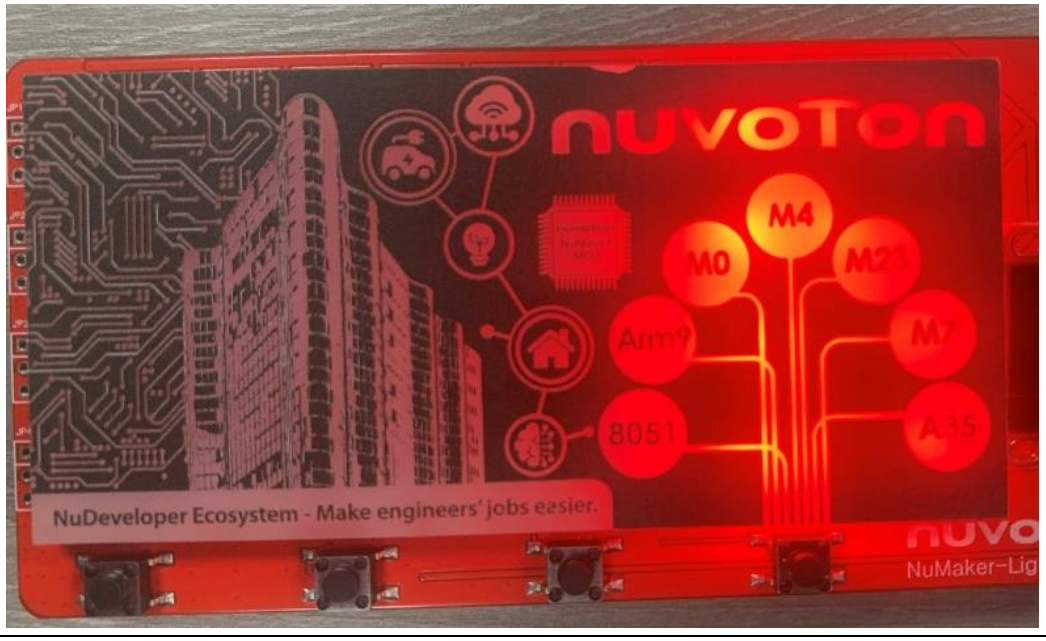

图 1-18 ARGB 上板的 Raindrop 灯效(1)

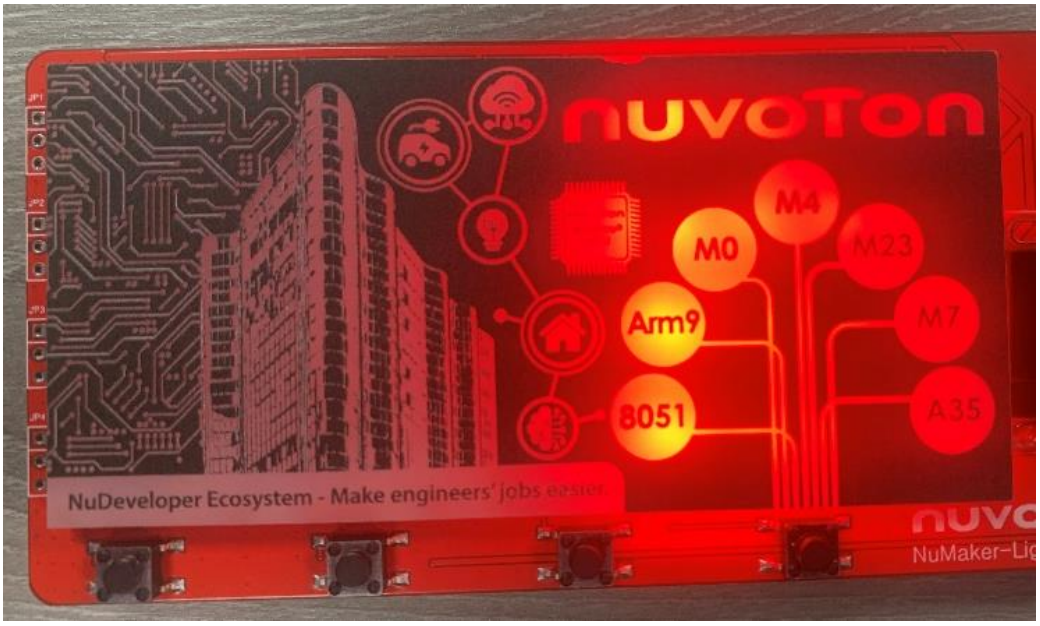

图 1-19 ARGB 上板的 Raindrop 灯效(2)

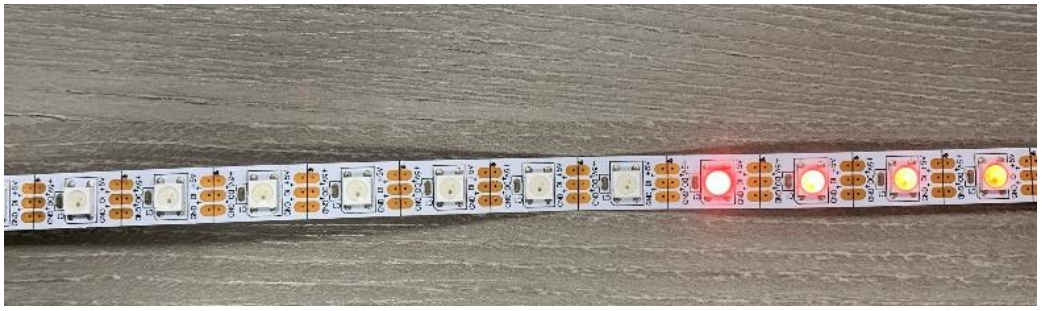

图 1-20 灯条的 Raindrop 灯效(1)

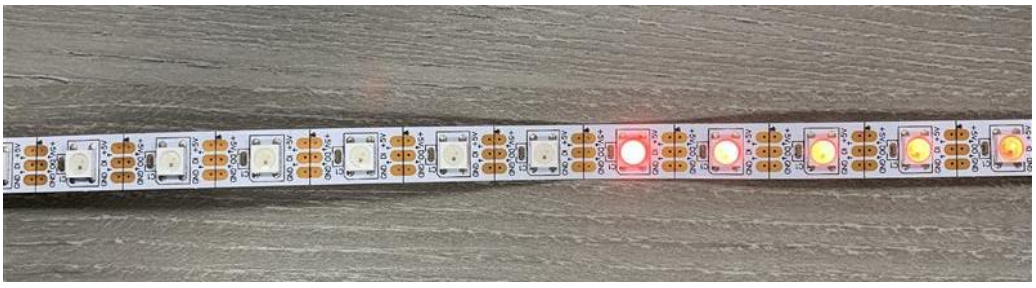

图 1-21 灯条的 Raindrop 灯效(2)

2. 代码介绍

初始化系统。代码位于 *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();
```

对 LLSI 和 NuMaker-Lighting-ARGB 上板的按钮进行初始化。

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

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

函式 LLSI_Initial 内容，代码位于 *LED_Control.c*。此处做 LLSI Clock、脚位、信号格式、中断和配合 PDMA 的设定。LLSI_Open 为 NUC1262 的 LLSI Library 的函式，用来对 LLSI 信号格式的 Channel、模式、0 code、1 code 等等做设定。

```
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，代码位于 *Button.c*。按钮使用 GPIO 的 Falling trigger interrupt 做触发，此处为设定 GPIO Clock、设定脚位模式、中断与 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);
}
```

主程序内容，轮询 Systick 计时是否达到更新时间(此处更新时间为 25 ms)，如果达到更新时间检查该灯效 LLSI 是否完成传输，完成传输则重新进入函数并设定 LLSI 的下一笔传输数据。

```
       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;
        }
```

中断函式 SysTick_Handler 内容，此处对灯效的运作有两个功能，一个是增加主程序轮询的更新 Flag，另一个是增加灯效更新的计数。

```
 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 内容，代码位于 *LED_Control.c*。用于将 ARGB 数据透过 PDMA 给 LLSI 脚位传送出去。让 PDMA 传送数据到 LLSI 前需要设定传输的内容，包括数据大小、数据数量、PDMA 连接的 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;
}
```

灯效函式 Breath 内容，此处利用 Systick 纪录时间，透过 HDIV_Mod(取余数函数)、HDIV_Div(除法函数)来计算当前灯光的亮度，BreathingBright 为呼吸灯经过一次完整周期的亮度变化数组，BreathingArraySize 则为其数组大小，由计算得出当前应该要在 BreathingBright 数组中第几个元素的亮度。TimeCounter 大于一个灯效的周期时，归零 TimeCounter 且变为下个颜色。计算完成后，透过 Set_Single 函数将计算结果储存并由 LLSI 发送。

```
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 内容，将数据存到设定好 LLSI 搭配 PDMA 的数组中，会用 Set_LED_Data 函式将数据由 LLSI 发送出去。

```
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;
    }
}
```

灯效函式 Color Cycle 内容，正向和反向仅差在渐变顺序不同。为达成此灯效需先确定渐变色两端的颜色，Color1 和 Color2 数组根据 TimeCounter 决定渐变色两端颜色的 RGB 值。由 DisplayColor 数组储存最终由 LLSI 传送出去的数据，计算则是由 CurrentTime(当前处于渐变色两端的相对时间)和 DeltaT(到达渐变色两端中间需要经过的时间)两个变量决定 Color1 和 Color2 各自所占的比例。在 TimeCounter 经过一次循环的时间后，将此变量归零。

```
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]);
}
```

图 2-1、图 2-2 为该段程序的可视化说明。TimeCounter 持续计时，DisplayColor 为当前显示的颜色，该颜色由 CurrentTime 位置的两个颜色混合所组成(图 2-2)，判断在哪两个颜色之间可以用 TimeCounter 除以 DeltaT 的商数得出，Temp 记录为哪两个颜色。而两个颜色各自所占的比例可以用 CurrentTime 和 Delta 的计算得出。在 TimeCounter 大于 DeltaT 乘以 ColorNum 后归零。

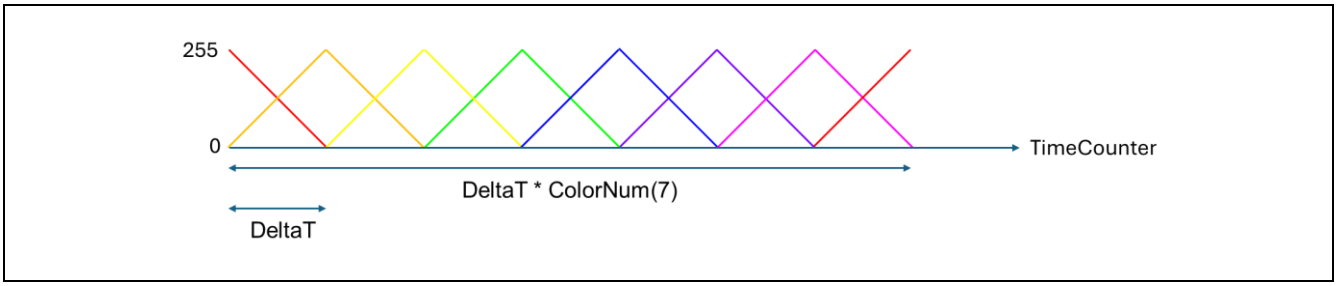

图 2-1 Color Cycle 说明(1)

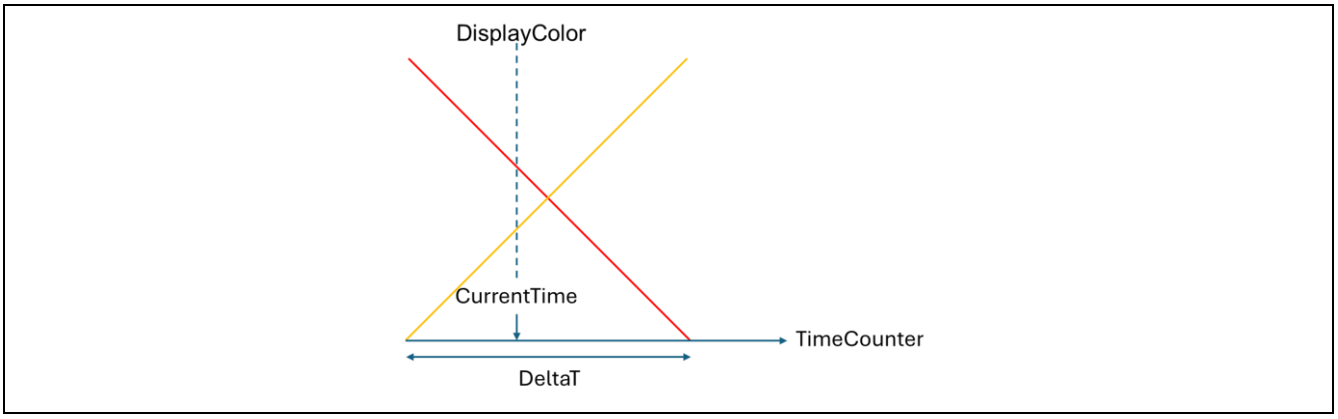

图 2-2 Color Cycle 说明(2)

灯效函式 Rainbow Wave 内容，TotalDuration 变量为全部颜色循环所需的时间，NextTime 变量为用来计算每个灯应呈现的颜色，CurrentTime 即为此刻的时间点，deltaT 变量为渐变色两端所经过的时间。以上的变量在 for 循环中皆会用做计算每个灯的 RGB 值，渐变一样采用线性，根据 CurrentTime 数值在 deltaT 的位置对渐变色两端的比例进行计算，最后得出应显示的值。不同于 Color Cycle，此处须把每个灯所应显示不同的颜色做计算，因此需要根据 LED_Num 变量(灯条上进行灯效的灯数)做数次计算。计算好后由 DisplayColor 数组中储存最终由 LLSI 传送出去的数据。在 TimeCounter 经过一次循环的时间后，将此变量归零。

```
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);
}
```

图 2-3、图 2-4 和图 2-5 为该段程序的可视化说明。TimeCounter 持续计时，DisplayColor 为显示当前的颜色。不同于 Color Cycle，此处为一个根据灯条上的灯数所创建的二维数组。先计算 CurrentTime 的值，并计算下一个灯更新的计数值(NextTime)，由此来得出每个灯条上的灯珠所显示的颜色。因此最后得出的 DisplayColor 值会是该灯条上每个灯珠呈现的 RGB 值，并非跟 Color Cycle 一样只有单一 RGB 值。每一个灯所呈现颜色计算方式同 Color Cycle，为线性方式计算。

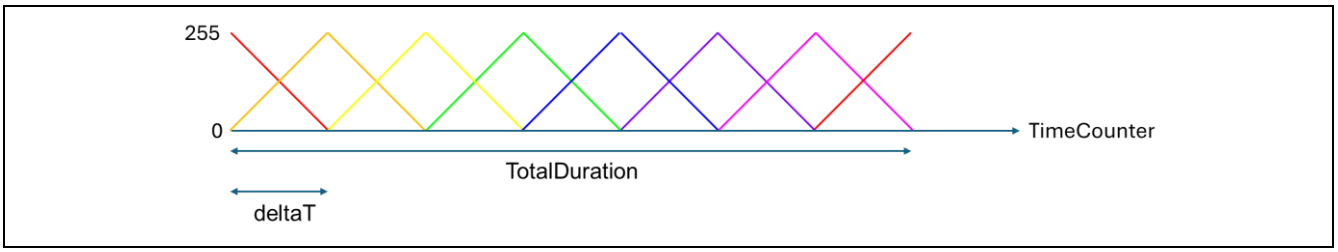

图 2-3 Rainbow Wave 说明(1)

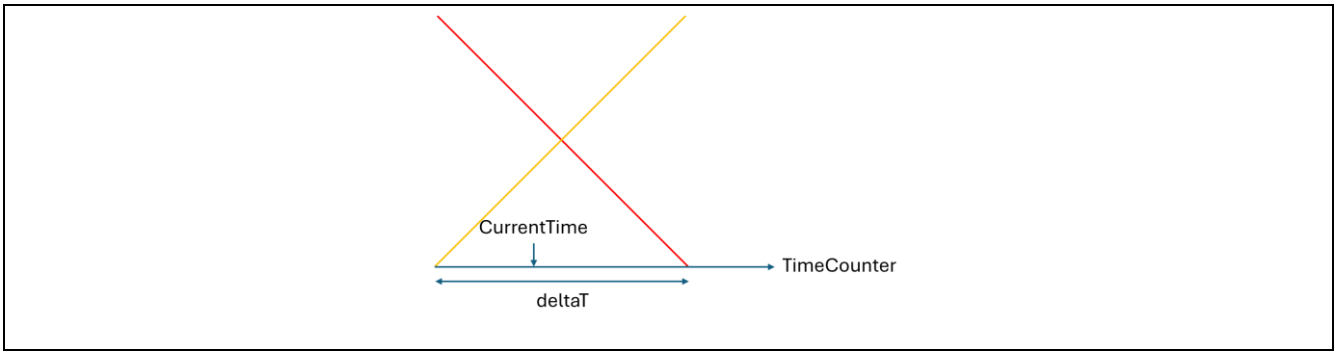

图 2-4 Rainbow Wave 说明(2)

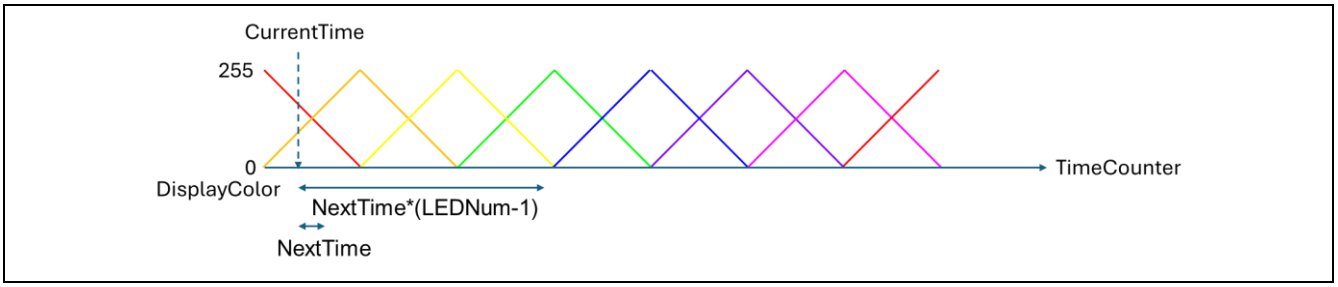

图 2-5 Rainbow Wave 说明(3)

函式 Set_Array 内容，此函式功能与 Set_Single 函式相似，将数据存到设定好 LLSI 搭配 PDMA 的数组中，会用前面的 Set_LED_Data 函式将数据由 LLSI 发送出去。与 Set_Single 函式不同的是，此处每个数值皆有可能不一样，因此需要一一对应将数据填入。

```
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;
    }
}
```

灯效函式 Raindrop 内容，DeltaT 变量为灯移动一次的时间，Duration 变量为灯完整的走完一次灯条的时间，DelayT 变量为在灯条全暗时加延时以让灯不要不间断地出现，Raindrop_Trailing 为制造拖尾效果的数组。用 if 判断现在是灯还在移动的时间或是灯条全暗的时间，并进入 for 循环设定此刻哪个灯为全亮、哪个灯为拖尾亮度、哪个灯为暗。连续更新灯条上的灯，就可以做出灯往一个方向移动的效果。在 TimeCounter 经过一次循环的时间后，将此变量归零。

```
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. 软件与硬件需求

3.1 软件需求

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

3.2 硬件需求

- 电路组件
  - NuMaker-NUC1262SE V1.2
- 线路示意图
  - 将 UART0 TX (PB.13) pin 脚连接到 PC UART RX，以显示范例代码的执行结果。再来可视是否有 NuMaker-Lighting-ARGB 上板执行操作。
  - 若有 NuMaker-Lighting-ARGB 上板，将上板接上 NuMaker-NUC1262SE 的 UNO 接口即可(图 3-1)。用上板自带的按钮或是 PC 端执行灯效变换。
  - 若没有 NuMaker-Lighting-ARGB 上板，使用协议为 ARGB 的灯条，接上 UNO 接口上的 5VCC 和 VSS 电源脚位，以及信号脚位(可选择 PA.3、PC.2、PC.5、PB.4)连接(图 3-2)。用 PC 端执行灯效变换。

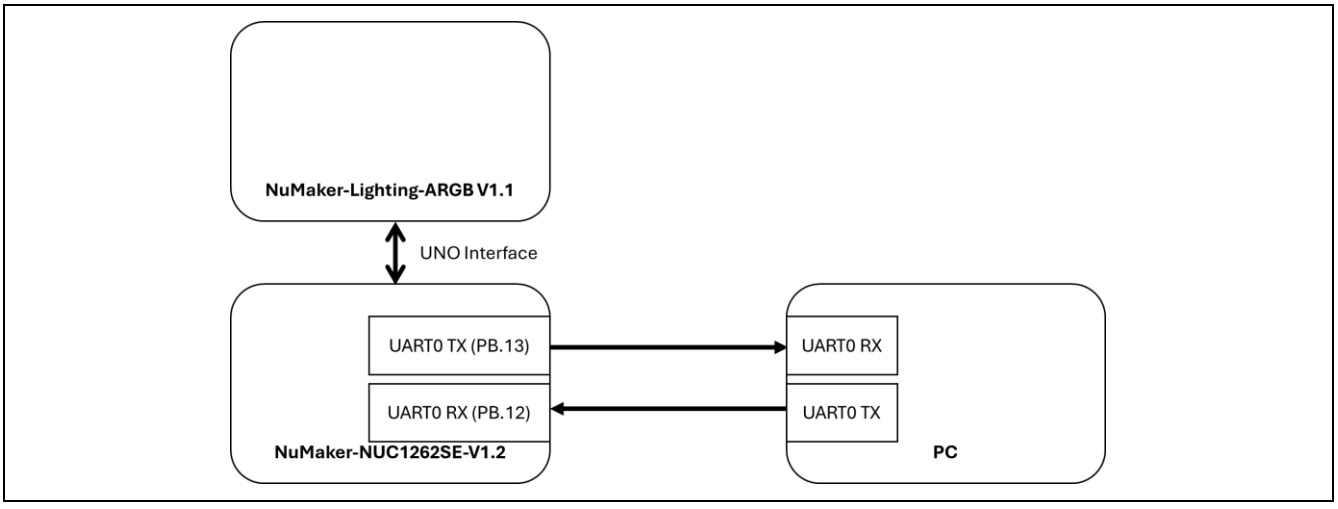

图 3-1 线路示意图(1)

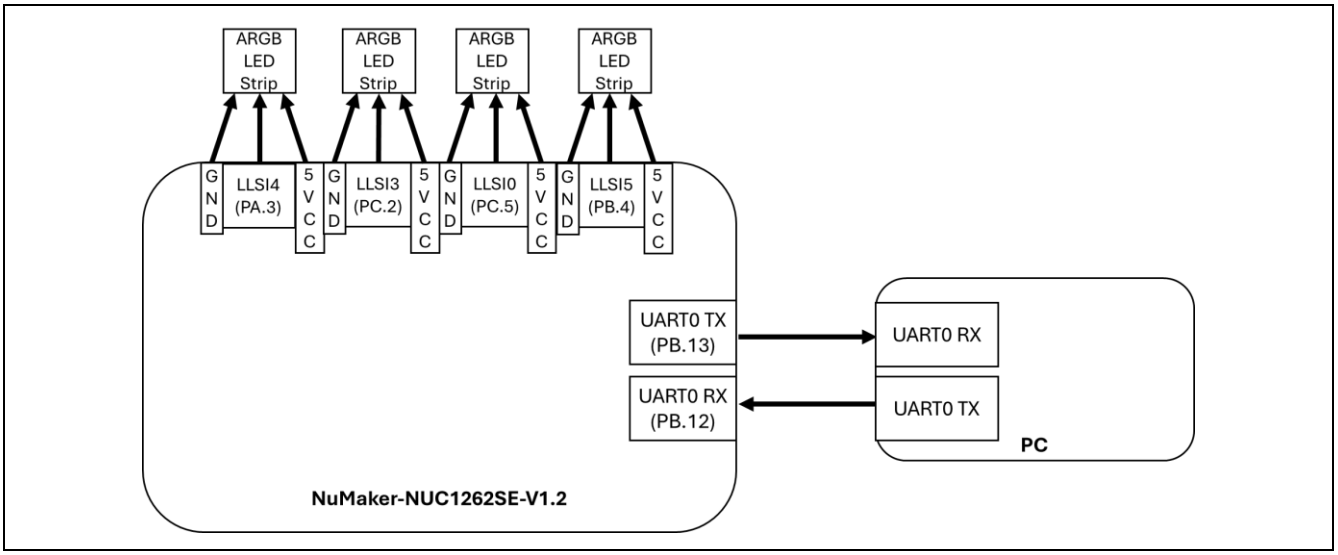
ARGB LED Strip
ARGB LED Strip
ARGB LED Strip
ARGB LED Strip
GND
LLSI4 (PA.3)
5VCC
GND
LLSI3 (PC.2)
5VCC
GND
LLSI0 (PC.5)
5VCC
GND
LLSI5 (PB.4)
5VCC
UART0 TX (PB.13)
UART0 RX (PB.12)
UART0 RX
UART0 TX
PC
NuMaker-NUC1262SE-V1.2

图 3-2 线路示意图(2)

4. 目录信息

目录结构如下图所示。

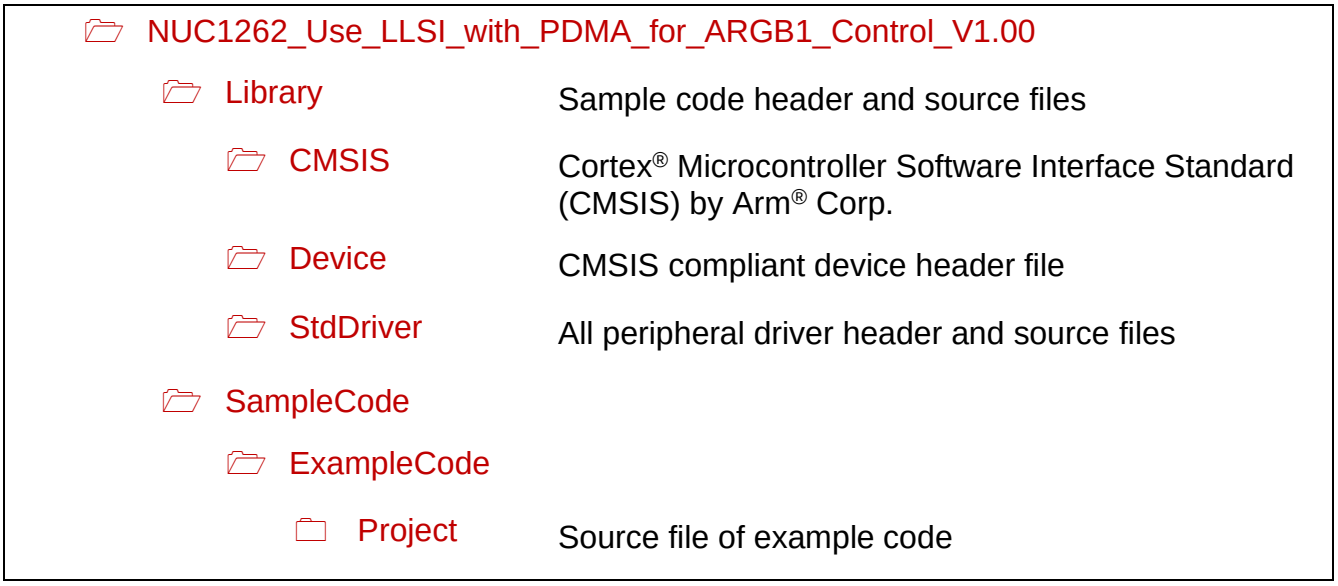

图 4-1 目录信息

5. 范例程序执行

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 *NUC1262_Use_LLSI_with_PDMA_for_ARGB1_Control.uvprojx*。
2. 进入编译模式接口
   - 编译
   - 下载代码至内存
   - 进入 / 离开除错模式
3. 进入除错模式接口
   - 执行代码
4. 使用终端机程序(ex: Tera Term)，根据 UART 回传的讯息发送指令来切换灯效。

6. 修订纪录

Date	Revision	Description
2025.04.03	1.00	初始发布。

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.