NUC1262 Use LLSI with PDMA for ARGB1 Control

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses NUC1262 LLSI with PDMA to control ARGB Gen 1 LED lights.
BSP Version	NUC1262_Series_BSP_CMSIS_V3.00.003
Hardware	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. Overview

This example code uses NUC1262 series LLSI (LED Lighting Strip Interface) in conjunction with PDMA(Peripheral Direct Memory Access) to control LED light strips that use the ARGB Gen 1 protocol. LLSI is an IP specifically designed for the ARGB protocol in the NUC1262 series by Nuvoton. Using other protocols or GPIO to control the ARGB signal would consume CPU resources, whereas using LLSI with PDMA can significantly reduce CPU usage.

To observe the actual lighting effects, this example code needs to be used with LED light strips that support the ARGB Gen 1 protocol. Alternatively, it can be run on the NuMaker-Lighting-ARGB board (Website: NuMaker-Lighting-ARGB - Nuvoton) provided by Nuvoton (as shown in Figure 1-1).

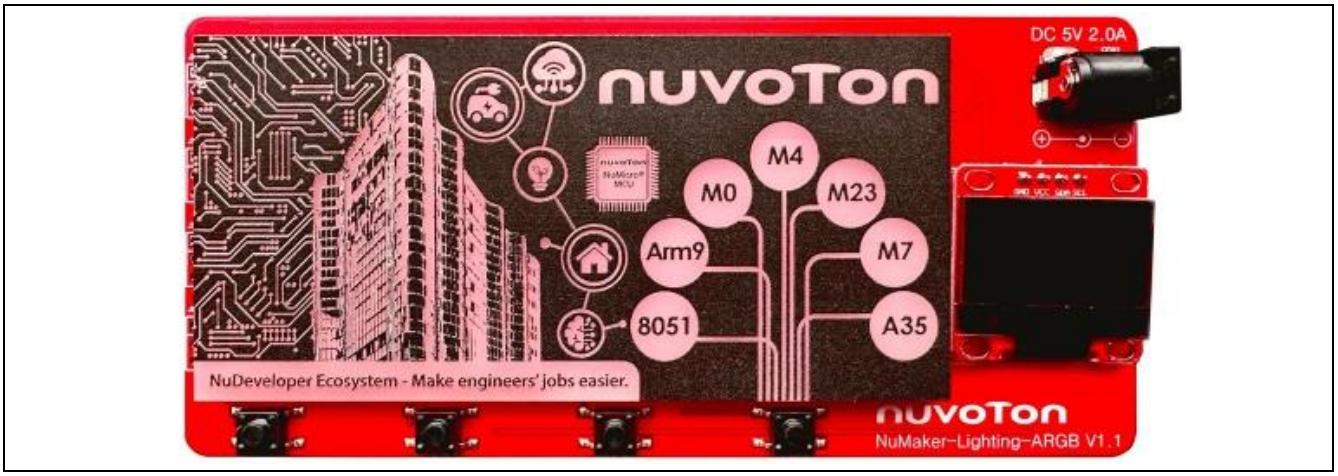

Figure 1-1 NuMaker-Lighting-ARGB

1.1 Principle

The ARGB protocol only requires three hardware connections to operate: two power lines and one signal line. The LLSI mentioned in the title refers to the IP designed by the NUC1262 series for outputting such signals. In the standard ARGB protocol, 24 bits of data are transmitted for each RGB light to determine the color to be displayed. The 24-bit data consists of 8 bits each for R, G, and B, where each 8-bit value represents the brightness of the corresponding color from 0 to 255.

Taking the standard ARGB as an example, the signal format is shown in Figure 1.2. Each bit can be represented by 0 or 1. The 0 code in ARGB is transmitted as 300 ns of high followed by 900 ns of low, while the 1 code is transmitted as 900 ns of high followed by 300 ns of low. Transmitting 24 bits in sequence represents the RGB value of one light. The order of RGB transmission may vary on different light strips. In this example, the transmission is set in the order of GRB. After determining the desired number of lights and transmitting the RGB values for each light, a Reset Code signal needs to be sent. The format of this signal is a low signal greater than 50 us. At this point, the entire LED light strip will display the color of the previously transmitted ARGB data.

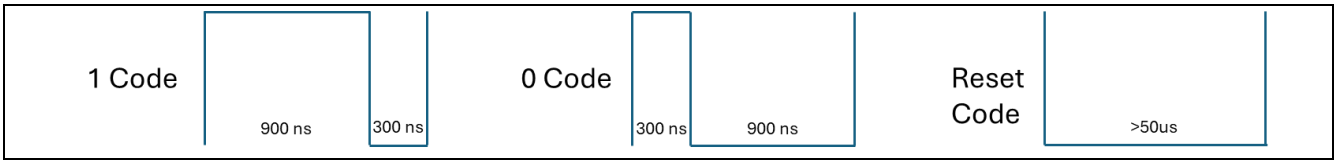

Figure 1-2 The signal format of ARGB

Different manufacturers may have slight variations in the signal format for the ARGB protocol on their light strips. The LLSI interface allows for flexible adjustment of the signal format for the 0 code, 1 code, and Reset code, enabling control of different manufacturers' light strips.

The LLSI transmission is combined with PDMA to enhance execution efficiency. PDMA allows data to be transferred from memory location A to memory location B without involving the CPU. In this application, it is used to transfer the ARGB data to the LLSI output register (LLSI_DATA). Prior to usage, PDMA transfer settings, source data address, and destination data address need to be configured.

This example code demonstrates four lighting effects: Breath, Color Cycle, Rainbow Wave, and Raindrop. Each effect is set to cycle through seven rainbow colors, as shown in chapter 1.2 image.

Breath effect simulates a breathing pattern, where the overall brightness of the light strip does not linearly increase or decrease. To create the breathing effect, the rate of brightness change is slowed down at the start and near the maximum brightness.

Color Cycle effect involves a gradual transition of colors across the entire light strip. The two colors for the transition are determined, and the current color is calculated using a linear method based on the current time.

Rainbow Wave effect moves the colors along the light strip in one direction. Similar to the Color Cycle effect, the calculation of the displayed color is based on the current time using a linear transition. The difference is that the transition occurs on individual lights at the same time, rather than the entire light strip being the same color. Therefore, it is necessary to calculate which color each light on the strip should display.

Raindrop effect involves multiple lights moving in one direction. This effect requires calculating the time at which each light should turn on, as well as the color and brightness of the light.

1.2 Demo Result

The compilation and programming results of this example code are shown in Figure 1-3.

```
Build Output
creating preprocessor file for qspi.c...
creating preprocessor file for pdma.c...
linking...
Program Size: Code=8500 RO-data=892 RW-data=20 ZI-data=1368
After Build - User command #1: fromelf --bin ".\obj\M251_QSPI_simulate_PWM.axf" --output ".\obj\M251_QSPI_simulate_PWM.bin"
After Build - User command #2: fromelf --text -c ".\obj\M251_QSPI_simulate_PWM.axf" --output ".\obj\M251_QSPI_simulate_PWM.txt"
".\obj\M251_QSPI_simulate_PWM.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:02
Load "C:\\Users\\CYTENG\\Desktop\\Note\\Example_code\\EC_M251_QSPI_Simulate_PWM_with_PDMA_V1.00\\SampleCode\\ExampleCode\\DEMO\\M251_QSPI_simula
Erase Done.
Programming Done.
Verify OK.
```

Figure 1-3 Compilation and Programming Results

In the terminal program (ex: Tera Term), you can see the displayed message each time you switch. You can use commands in the terminal program to switch the lighting effects of all lights as shown in Figure 1-4.

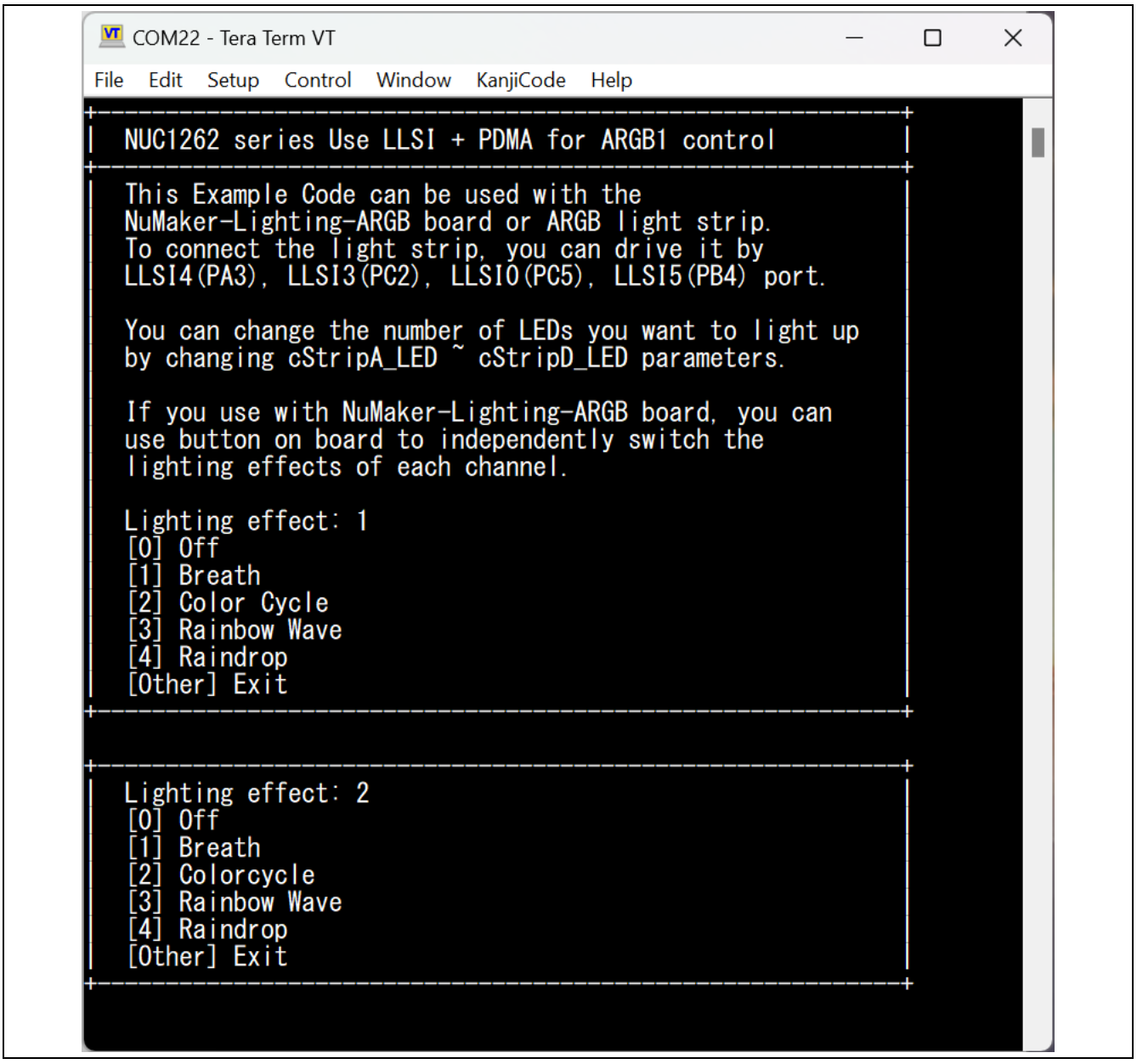

Figure 1-4 Terminal Display Content - Terminal Command Execution

If you use the NuMaker-Lighting-ARGB board, you can also independently control the lighting effects of each LLSI using the buttons on the board. Each press of the button switches the corresponding LLSI lighting effect in the sequence: Breath → Color Cycle → Rainbow Wave→ Raindrop → Off → Breath, and then it cycles through again, as shown in Figure 1-5.

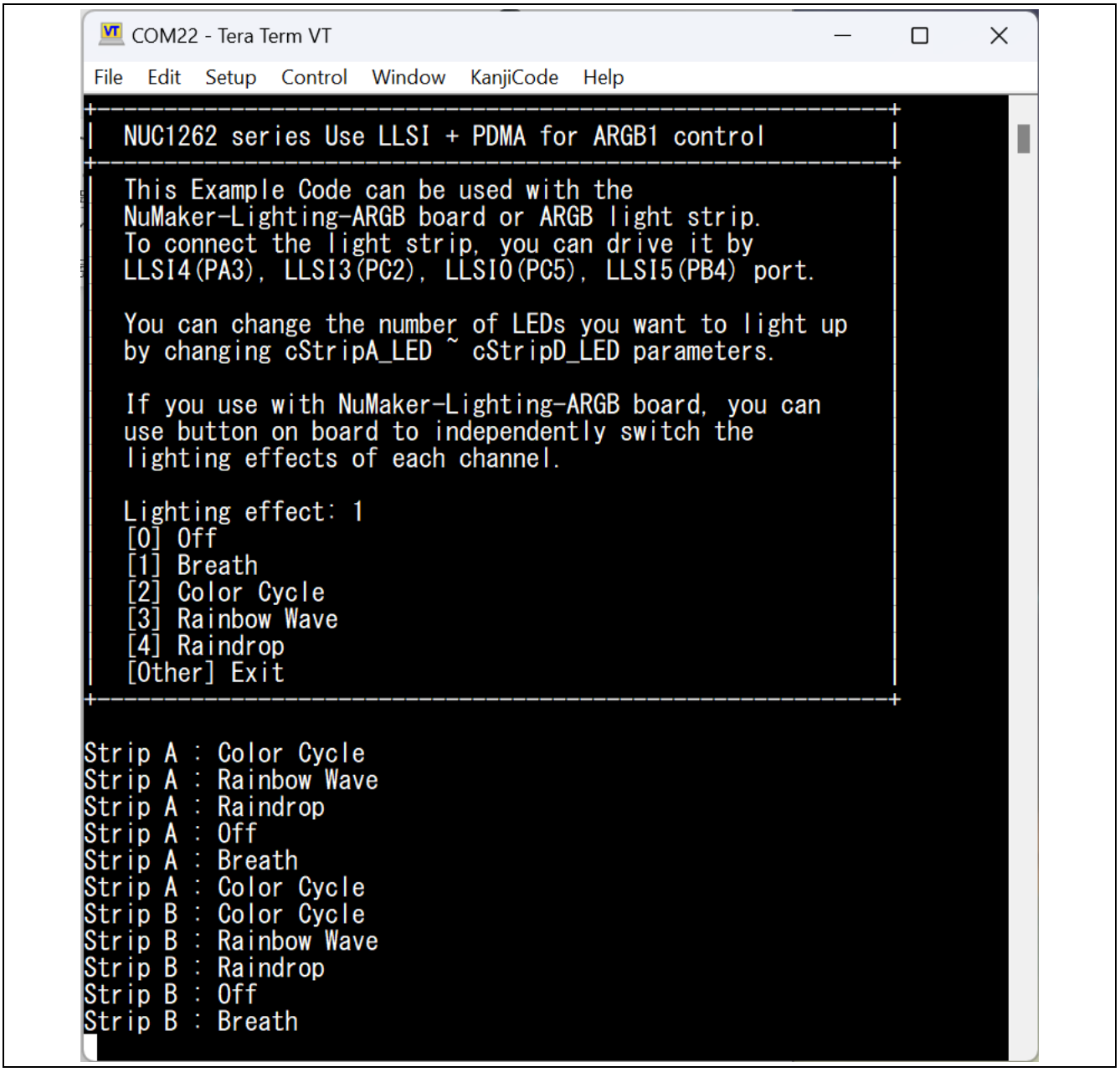

Figure 1-5 Terminal Display Content - NuMaker-Lighting-ARGB Board Button Control

The results of running the four lighting effects on the NuMaker-Lighting-ARGB board with an ARGB light strip are shown in Figure 1-6 to Figure 1-21. The images only capture adjacent color transitions as an illustration, and the actual presentation may vary based on the executed program code.

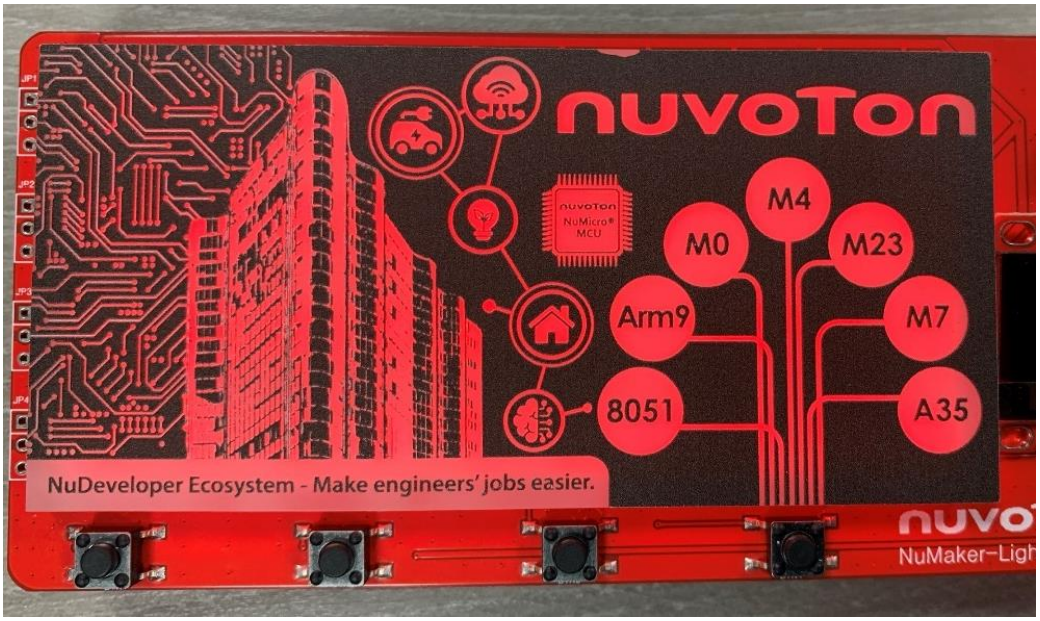

Figure 1-6 The Breath lighting effect on the ARGB board(1)

Figure 1-7 The Breath lighting effect on the ARGB board(2)

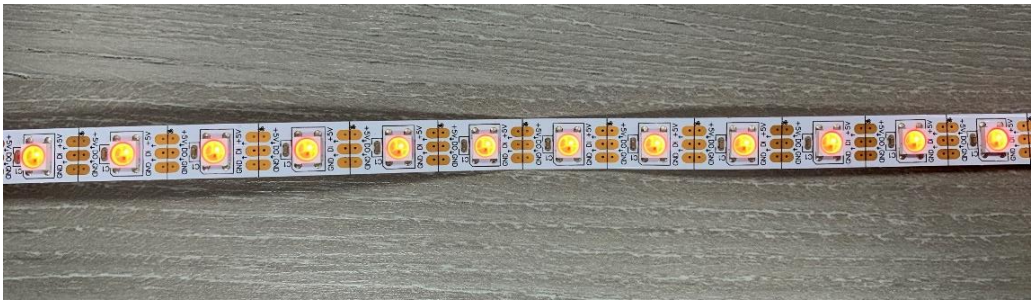

Figure 1-8 The Breath lighting effect on the light strip(1)

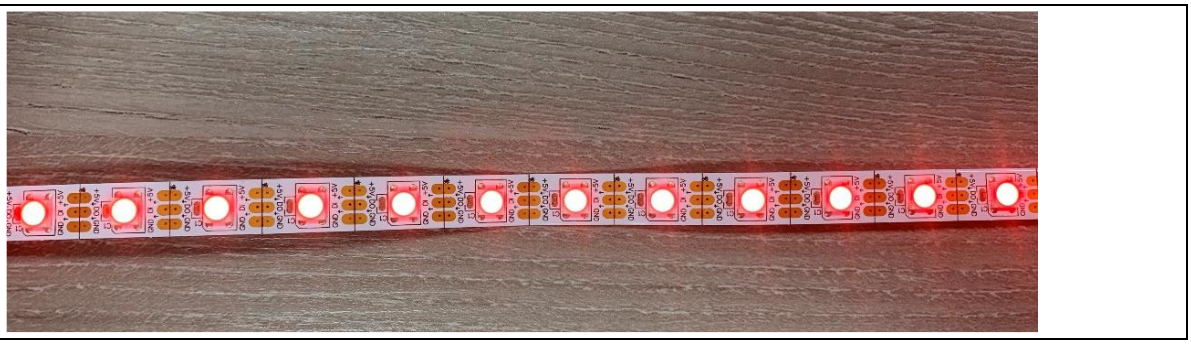

Figure 1-9 The Breath lighting effect on the light strip(2)

Figure 1-10 The Color Cycle lighting effect on the ARGB board(1)

Figure 1-11 The Color Cycle lighting effect on the ARGB board(2)

Figure 1-12 The Color Cycle lighting effect on the light strip(1)

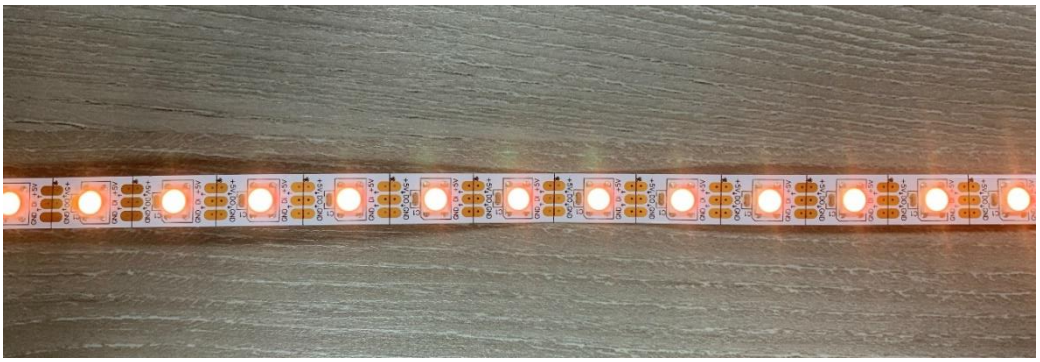

Figure 1-13 The Color Cycle lighting effect on the light strip(2)

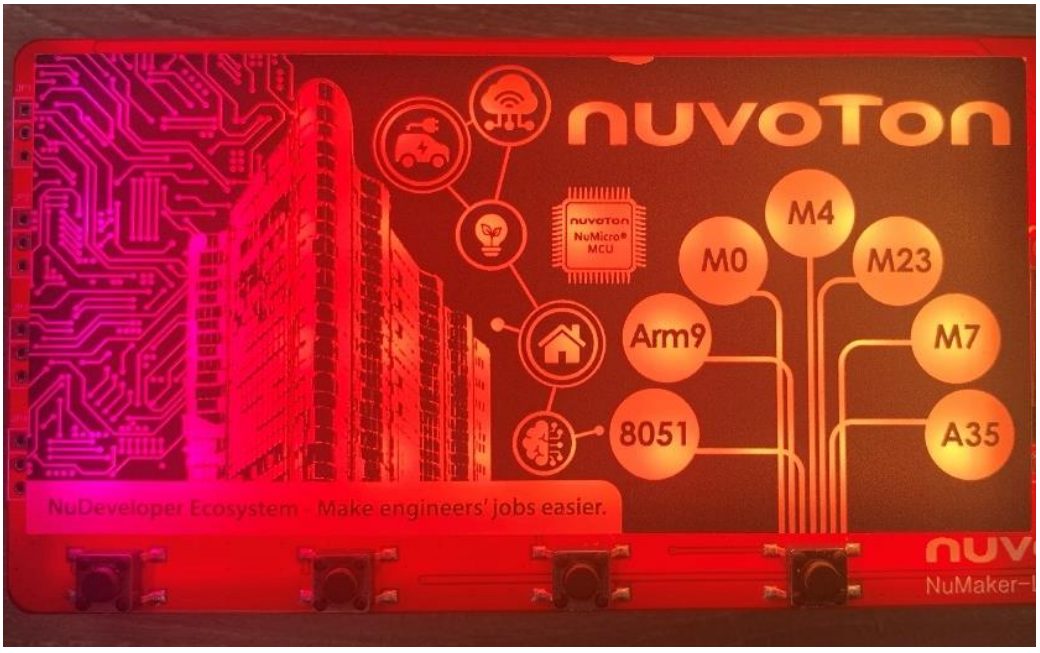

Figure 1-14 The Rainbow Wave lighting effect on the ARGB board(1)

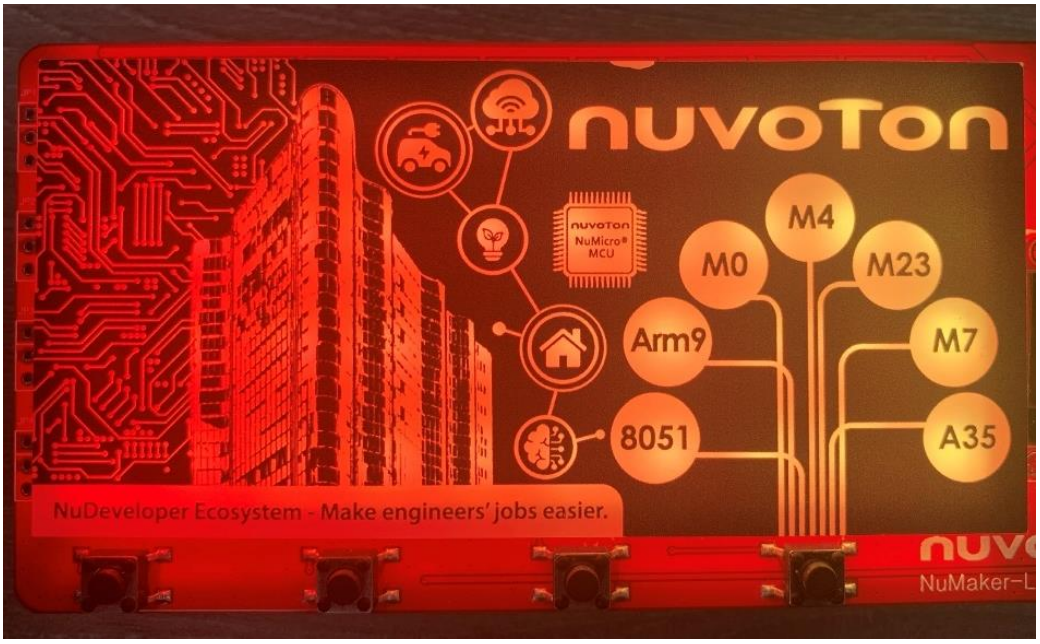

Figure 1-15 The Rainbow Wave lighting effect on the ARGB board(2)

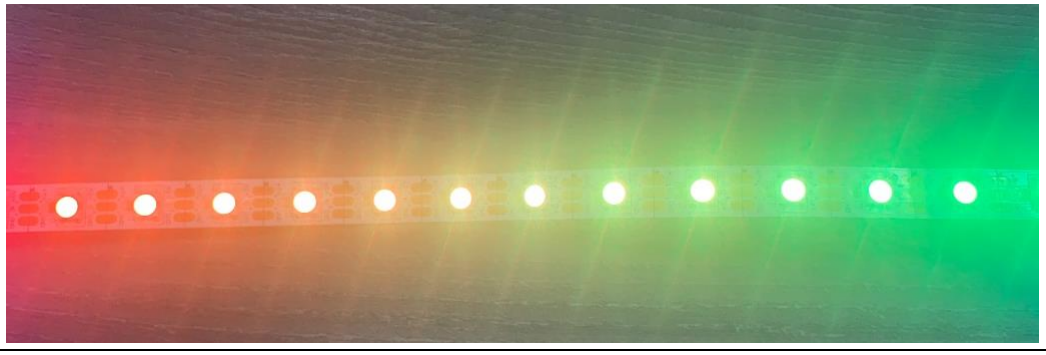

Figure 1-16 The Rainbow Wave lighting effect on the light strip(1)

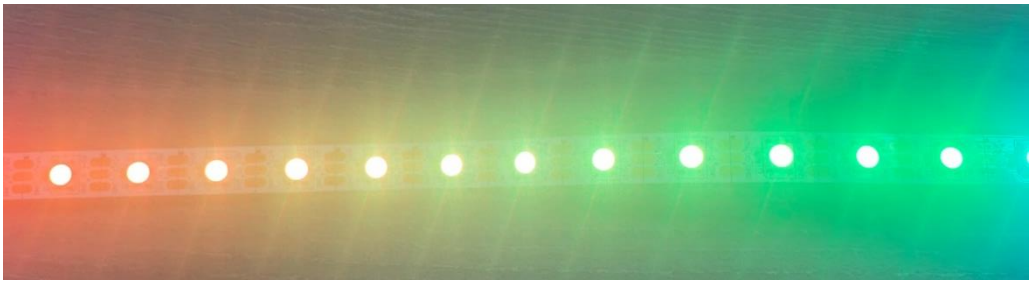

Figure 1-17 The Rainbow Wave lighting effect on the light strip(2)

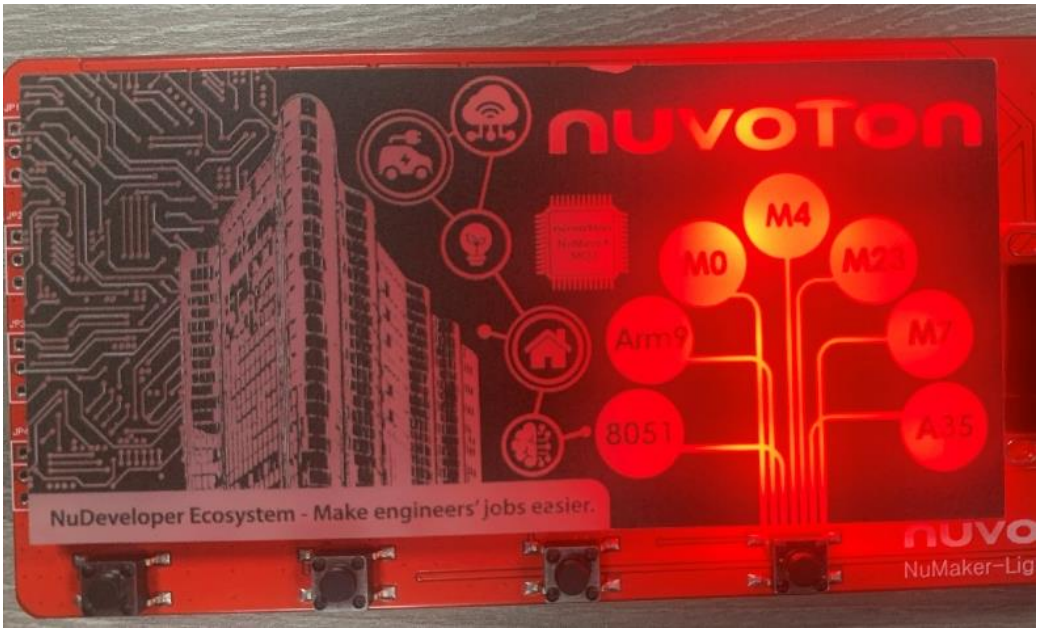

Figure 1-18 The Raindrop lighting effect on the ARGB board(1)

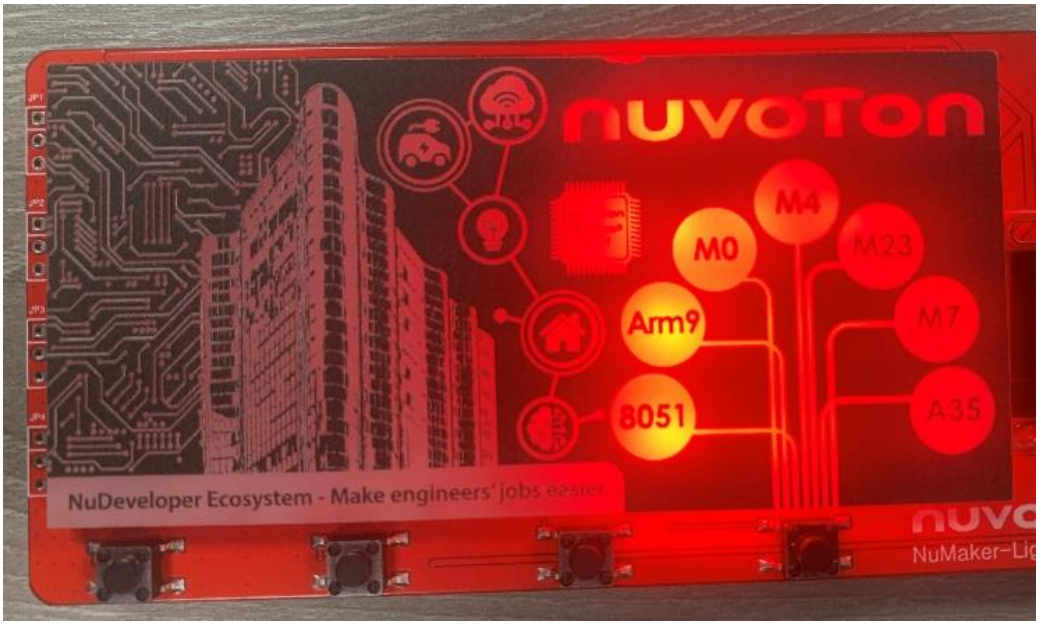

Figure 1-19 The Raindrop lighting effect on the ARGB board(2)

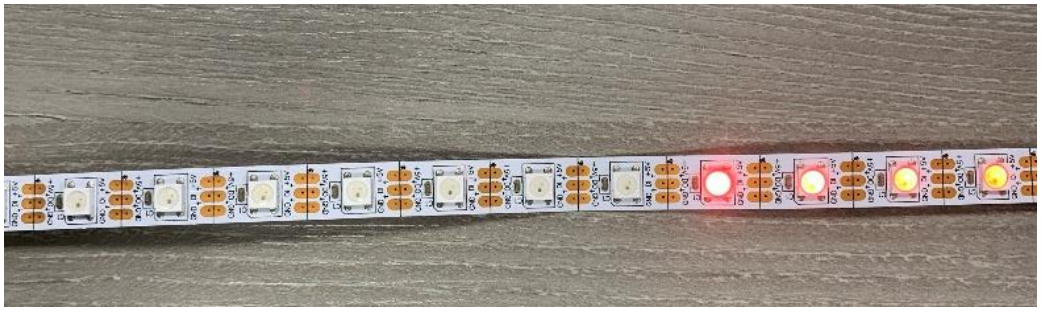

Figure 1-20 The Raindrop lighting effect on the light strip(1)

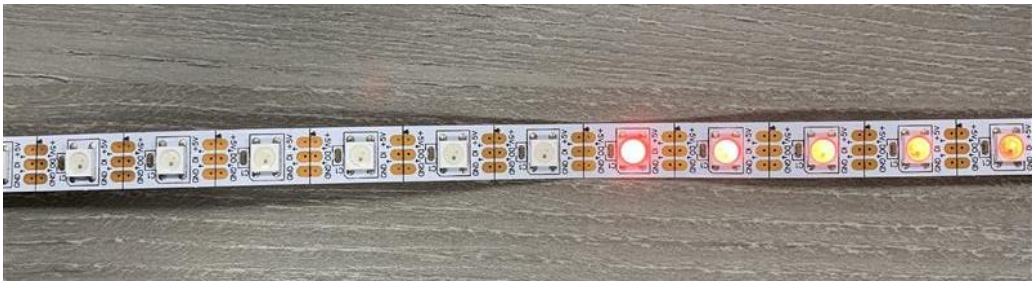

Figure 1-21 The Raindrop lighting effect on the light strip(2)

2. Code Description

Initialize the system. The code is located in *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();
```

Initialize the LLSI and buttons on the NuMaker-Lighting-ARGB board.

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

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

The content of the LLSI_Initial function, located in *LED_Control.c*, involves the configuration of LLSI clock, pins, signal format, interrupts, and PDMA settings. The LLSI_Open function is a part of the NUC1262 LLSI Library and is used to configure the LLSI signal format, including channel, mode, 0 code, 1 code, and more.

```
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();
}
```

The content of the Button_Initial function, located in *Button.c*, involves the configuration of the buttons using GPIO Falling trigger interrupts. This includes setting the GPIO clock, configuring pin modes, interrupts, and debounce functionality.

```
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);
}
```

The main program content involves polling the Systick timer to check if the update time (in this case, 25 ms) has been reached. If the update time has been reached, it checks if the LLSI transmission for the current lighting effect is complete. If the transmission is complete, it re-enters the function and sets the next data to be transmitted by the LLSI.

```
       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;
        }
```

The content of the interrupt function SysTick_Handler includes two functionalities related to the operation of the lighting effects. One is to increment the update flag used by the main program's polling mechanism, and the other is to increment the count for the lighting effect updates.

```
 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;
    }
}
```

The content of the Set_LED_Data function, located in *LED_Control.c*, is used to transmit ARGB data through the LLSI pins using PDMA. Before PDMA can transfer data to LLSI, the transfer settings need to be configured, including data size, data count, the IP connected to PDMA, and the mode of LLSI.

```
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;
}
```

The content of the Breath lighting effect function involves using the Systick timer to track time. The current brightness of the light is calculated using the HDIV_Mod (remainder function) and HDIV_Div (division function). The BreathingBright array represents the brightness variation of the breathing effect over one complete cycle, and BreathingArraySize represents the size of this array. The current brightness is determined by calculating which element in the BreathingBright array corresponds to the current position. When the TimeCounter exceeds the duration of one cycle of the lighting effect, it is reset to zero and moves on to the next color.

Once the calculation is complete, the Set_Single function is used to store the calculated result and send it through LLSI.

```
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);
    }

}
```

The content of the Set_Single function involves storing the data in an array that is set up with LLSI and PDMA. The Set_LED_Data function is used to send the data through LLSI.

```
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;
    }
}
```

The content of the Color Cycle lighting effect function is similar for both forward and reverse directions, with the only difference being the order of the color transitions. To achieve this effect, the colors at the two ends of the transition need to be determined. The Color1 and Color2 arrays determine the RGB values of the colors at the two ends based on the TimeCounter. The final data to be sent through LLSI is stored in the DisplayColor array. The calculation is based on the CurrentTime (the relative time within the transition) and DeltaT (the time required to reach the middle of the transition). These variables determine the proportions of Color1 and Color2. After one complete cycle of the TimeCounter, the variables are reset to zero.

```
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]);
}
```

Figure 2-1 and Figure 2-2 is the visualization explanation for this segment of code. TimeCounter continuously counts time, and DisplayColor is the currently displayed color, which is composed of a mix of two colors at the CurrentTime position (as shown in Figure 2-2). To determine between which two colors it is, you can obtain it by dividing TimeCounter by DeltaT, and Temp records which two colors they are. The proportion of each color can be calculated using CurrentTime and Delta. When TimeCounter exceeds DeltaT multiplied by ColorNum, it resets to zero.

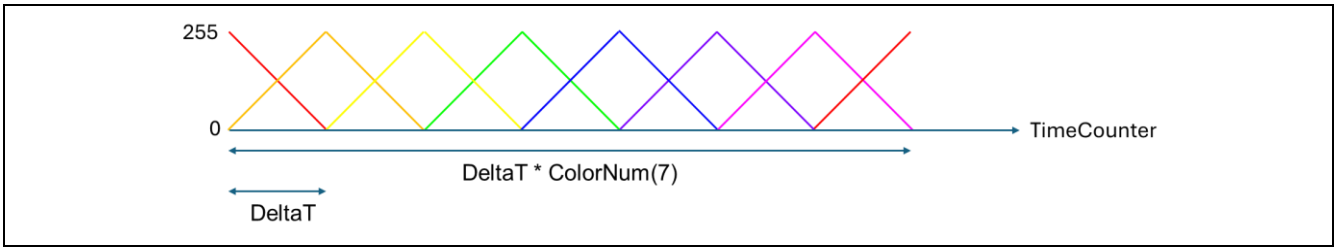

Figure 2-1 Color Cycle description (1)

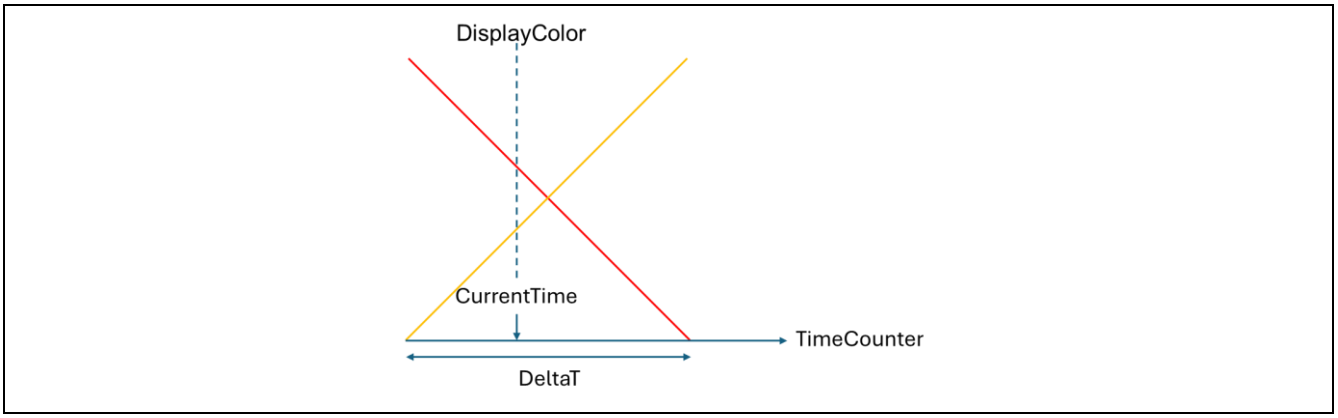

Figure 2-2 Color Cycle description (2)

The content of the Rainbow Wave lighting effect function involves several variables. The TotalDuration variable represents the total time required for a complete cycle of all colors. The NextTime variable is used to calculate the color that each light should display. CurrentTime represents the current time point, and deltaT represents the time span between the two colors in the transition. These variables are used in a for loop to calculate the RGB values for each light, using linear interpolation. Based on the CurrentTime value and the position within deltaT, the proportions of the colors at the two ends of the transition are calculated to determine the final color to be displayed. Unlike the Color Cycle effect, this function calculates the different colors that each light should display, based on the LED_Num variable (the number of lights on the light strip). The calculated values are stored in the DisplayColor array, which is then sent through LLSI. After one complete cycle of the TimeCounter, the variable is reset to zero.

```
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);
}
```

Figure 2-3. Figure 2-4 and Figure 2-5 is the visualization explanation for this segment of code. TimeCounter continuously counts, and DisplayColor shows the current color. Unlike Color

Cycle, this involves creating a two-dimensional array based on the number of lights on the light strip. First, calculate the value of CurrentTime and the count value for the next light update (NextTime), which determines the color displayed by each light on the strip. Therefore, the final DisplayColor value will be the RGB value presented by each light on the strip, not just a single RGB value like in Color Cycle. The color displayed by each light is calculated in the same linear interpolation as in Color Cycle.

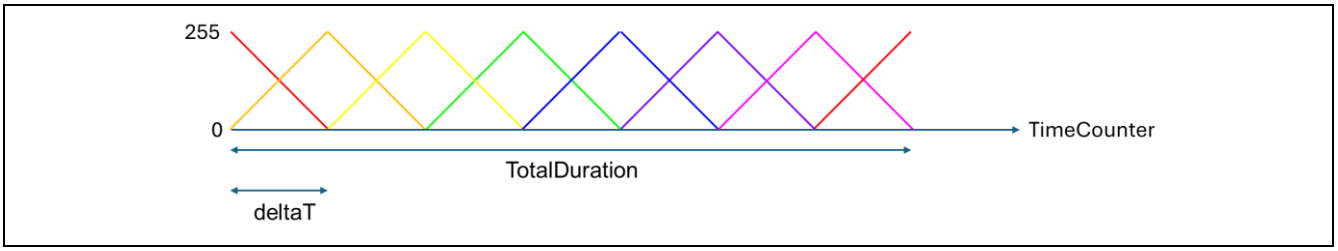

Figure 2-3 Rainbow Wave description (1)

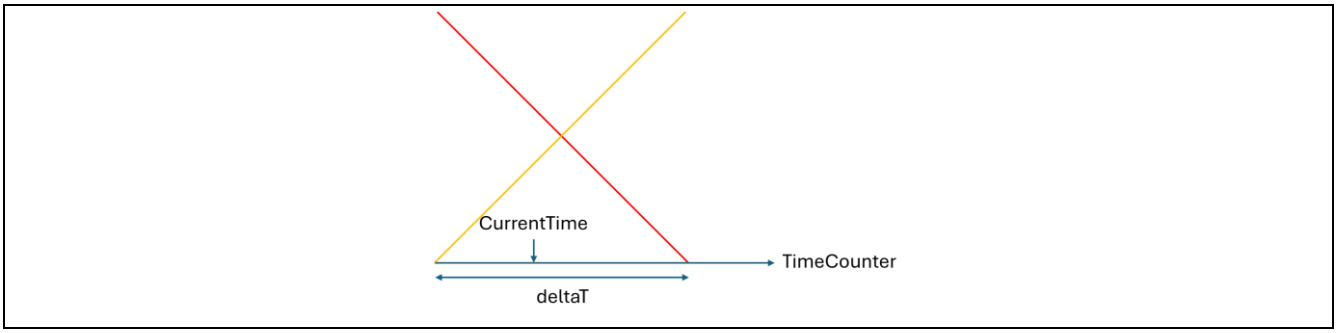

Figure 2-4 Rainbow Wave description (2)

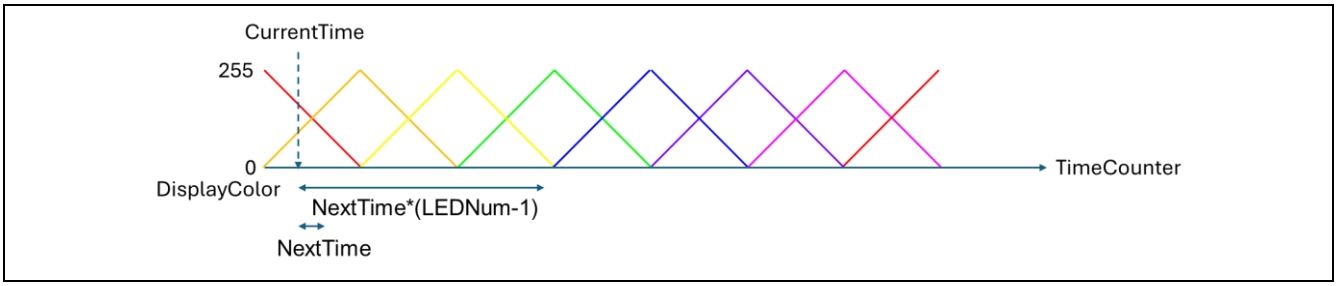

Figure 2-5 Rainbow Wave description (3)

The content of the Set_Array function is similar to the Set_Single function. It involves storing the data in an array that is set up with LLSI and PDMA, and then using the Set_LED_Data function to send the data through LLSI. The difference from the Set_Single function is that in Set_Array, each value can be different, so the data needs to be filled in one by one to correspond to each element.

```
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;
    }
}
```

The content of the Raindrop lighting effect function involves several variables. The DeltaT variable represents the time it takes for a raindrop to move once, while the Duration variable represents the total time it takes for a raindrop to complete one cycle along the light strip. The DelayT variable introduces a delay when all lights are off to prevent continuous appearance of raindrops. The Raindrop_Trailing array is used to create a trailing effect.

The function uses if statements to determine whether the current time is within the raindrop movement period or the period when all lights are off. It then enters a for loop to set which lights should be fully lit, which lights should have trailing brightness, and which lights should be off. By continuously updating the lights on the light strip, the effect of the raindrop moving in one direction can be achieved. After one complete cycle of the TimeCounter, the variable is reset to zero.

```
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. Software and Hardware Requirements

3.1 Software Requirements

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

3.2 Hardware Requirements

- Circuit components
  - NuMaker-NUC1262SE V1.2
- Pin Connect
  - Connect the UART0 TX (PB.13) pin to the PC UART RX pin to display the execution result of the sample code. Then, check if there are any operations performed on the NuMaker-Lighting-ARGB board.
  - If you have the NuMaker-Lighting-ARGB board, simply connect it to the UNO interface of the NuMaker-NUC1262SE (as shown in Figure 3-1). You can use the onboard buttons or control the lighting effect changes from the PC.
  - If you don't have the NuMaker-Lighting-ARGB board, you can use an ARGB light strip with the UNO interface. Connect the 5VCC and VSS power pins on the UNO interface, as well as the signal pin (you can choose from PA.3, PC.2, PC.5, PB.4) (as shown in Figure 3-2). You can control the lighting effect changes from the PC.

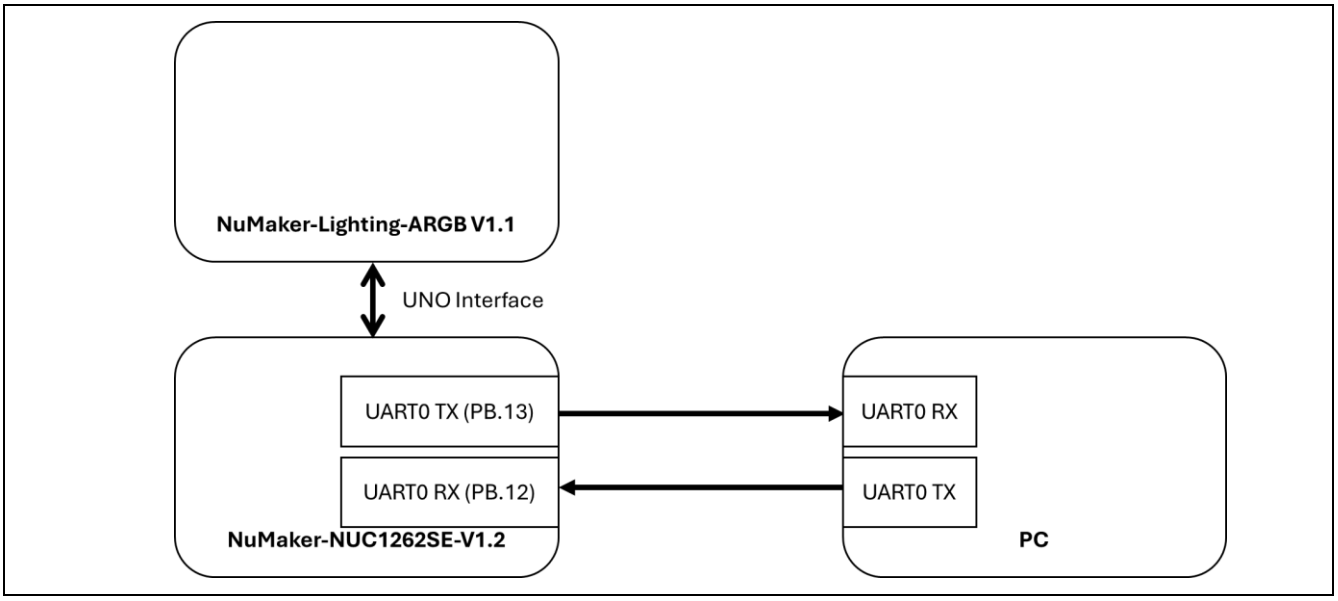

Figure 3-1 Pin Connect(1)

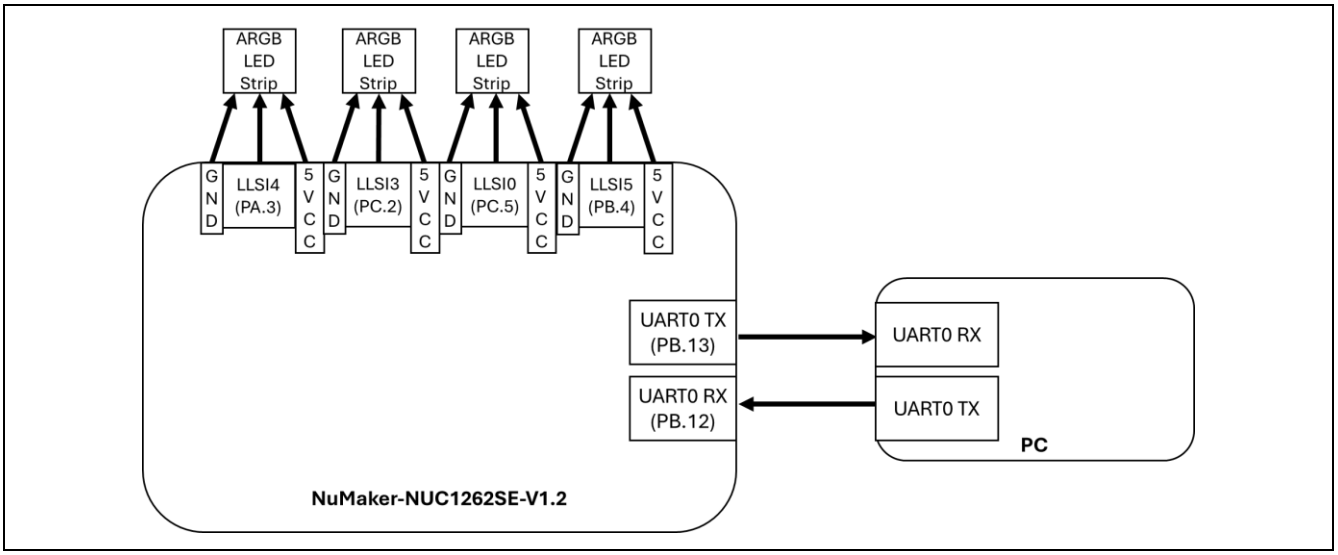

Figure 3-2 Pin Connect(2)

4. Directory Information

The directory structure is shown below.

NUC1262_Use_LLSI_with_PDMA_for_ARGB1_Control_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	
Project	Source file of example code

Figure 4-1 Directory Structure

5. Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *NUC1262_Use_LLSI_with_PDMA_for_ARGB1_Control.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run
4. Use a terminal program (e.g. Tera Term) to send commands based on the UART feedback message to switch between lighting effects.

6. Revision History

Date	Revision	Description
2025.04.03	1.00	Initial version.

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.