M251 一颗风扇叶片上 RGB 灯的灯效控制

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 M251 对一颗风扇叶片上 RGB 灯进行灯效控制
BSP 版本	M251_Series_BSP_CMSIS_V3.02.007
开发平台	NuMaker-M251SD-V1.1

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 M251 对风扇一颗风扇叶片上 RGB 灯进行控制以得到不同的灯效。

风扇所使用的是扇叶上自带三色 LED 灯的款式，该风扇主要用于显卡上的散热单元，该风扇除了原本的散热功能外，还可以通过不同灯效来增强视觉感受。显卡上的针对灯效做的视觉效果可以大致分成三种，第一种是在风扇槽的外圈放置灯光，通过灯条实现灯光效果。第二种是将风扇本身视作一个灯罩，通过在轴心放置灯光来实现灯效。最后一种也是本范例程序介绍的，在风扇的其中一个叶片上镶有 RGB LED 灯，通过风扇旋转速度和 LED 灯的互相配合来达成灯光效果。

1.1 此范例程序会用到以下 MCU 功能来运作:

1. PWM: 用作直接控制 RGB LED 灯的颜色，因为有三个风扇，共需要九个输出脚位。
2. QSPI: 用作仿真 PWM 信号以控制三个风扇的转速。
3. PDMA: 提高数据传输效率，以保证 QSPI 能维持传输速度。
4. Timer: 有两个功能，Timer Capture 用于侦测风扇转速，以校准灯效位置，Timer Time-out 用于更改 RGB LED 灯。
5. SysTick: 与 Timer 搭配用于更改 RGB LED 灯。
6. UART: 不用于风扇相关控制，用于 PC 端与 MCU 做数据传输。

1.2 RGB 灯风扇原理

该风扇上的 LED 灯使用红、绿、蓝(色光三原色)三路信号最终结合为灯光呈现的颜色。此处控制灯光使用 PWM 信号，利用短时间内不同占空比的 PWM 信号，用不同平均电压的三原色 LED 灯混合来使灯光输出想要的颜色。

除了混合 LED 灯外，还会用到人眼视觉暂留。人眼对于连续的灯有视觉暂留的现象，只要灯光更新的足够快，实际非同一时间出现的灯光，在旋转一整圈后和下一圈灯光重叠，就好像一整圈灯光同时持续在运作，不会有不连续的感觉。这也说明同一种灯效的灯光在相邻数圈的出现位置需要一致，才能得到我们想要呈现的效果，不然灯光效果会混杂在一起造成达不到想要的效果。基于上述，灯光输出需要配合风扇转速来使灯光出现在想要的位置。

想要知道风扇的位置信息，需要接收并分析风扇的 TACH 信号。风扇旋转时会回传 TACH 信号，该信号连接上拉电阻(10k ohm)后连接于 TM_EXT 脚位。TACH 信号每旋转 90 度会发生一次信号转折，借由这些信号转折可以知道风扇马达在 0 度、90 度、180 度、270 度位置的时间点，也可以通过计算知道当前风扇的转速。

风扇可以通过 PWM 的不同占空比来控制风扇转速。但即便给予相同的 PWM 信号控制风扇转速，风扇转速也不会是恒定值，因此需要每旋转一圈就更新一次当前的转速值，可以用来让灯光在指定位置显示想要的颜色。若没有持续更新风扇的转速值，也会造成灯光没办法在指定位置显示想要的颜色，造成灯光效果混乱。

有些需要散热的设备上可能同时有三个风扇在运作。要想同时控制三个转速不同的风扇实现灯光效果，需要节省 Timer 的使用。同时在一个 Timer 中开启 Capture mode 和 Continuous counting mode 功能，可以做到一个 Timer 对应一个风扇的灯光控制。Timer capture mode 利用 TM_EXT(Timer external)脚位将 TACH 信号做解析，判断信号转折点并记录相邻两个信号转折点的时间。利用风扇转 90 度会发生一次 TACH 信号转折的时间来推算风扇的转速。Timer 除了使用 Timer capture mode 外，还用 Continuous counting mode 控制灯光。要使用 Timer 控制灯光，需要知道如何配合 Capture mode 作为灯光控制的方式，图 1-1 为灯效运作底层的说明。

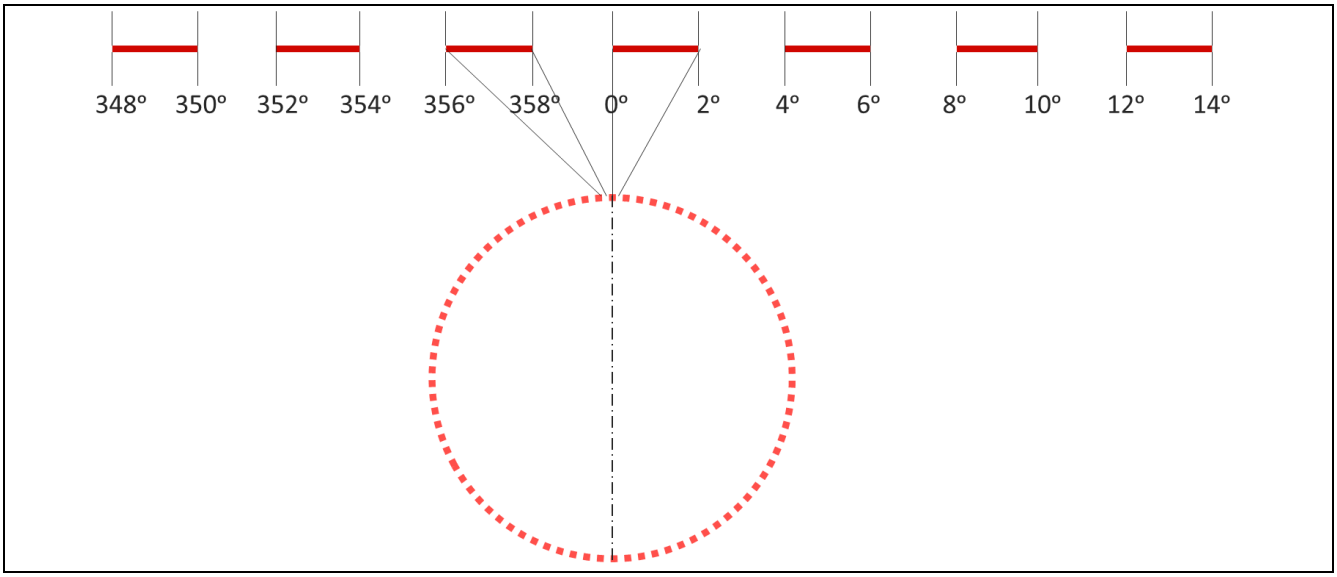


图 1-1 风扇上灯效显示原理

以每 2 度为一个单位做一段显示效果(图 1-1 使用红白相间来表示)。换句话说，风扇每旋转 2 度即进行一次灯光的显示，用 Capture mode 计算的转速可进一步计算旋转 2 度花费的时间，此值可以设定 Continuous counting mode 的下一次 Time-out 值。用 Timer continuous counting

mode 使风扇每旋转 2 度就进入一次 Timer 中断更新灯光内容(因同一个 Timer 需要搭配 Timer capture mode 使用，所以不可使用 Periodic mode 控制灯光效果，使用 Periodic mode 会清除 TIMER_CNT(Timer counter) 寄存器的值，造成 Capture mode 值错误)，并更新 TIMER_CMP(Timer compare) 寄存器的值 (Continuous counting mode 下一次 Time-out 值) 以让 Timer 持续动作。

在风扇旋转完一圈后在 12 点钟位置的灯光效果由 Timer capture mode 控制而不由 Timer continuous mode 的 Time-out 控制，在此点重新开始一圈的灯效显示、更新风扇转速、清除 TIF(Timer continuous counting mode 的 Time-out flag) 并更新 TIMER_CNT 值，这样可以让前后两圈在 12 点钟位置的灯光效果不会因为位置不同而产生混杂。

知道灯光的产生原理后即可按照想要的方式实现想要的效果。以下有三种灯光效果作为范例:

- 彩虹: 一整圈圆周为彩虹色，并往一个方向旋转。
- 颜色流动: 随着时间由一整圈的 A 颜色逐渐变成 B 颜色。
- 三爪:将圆分成三等分(每 120 度为一个区间)，其中每个区间只有前面 60 度发光，隔一段时间灯效往一个方向旋转。

同一段程序会因为风扇的旋转方向导致呈现出来的灯效会有所不同。为了方便说明，本文件关于灯效的说明统一用风扇顺时针旋转为范例进行图片示例。

1.3 灯效原理-彩虹

此灯效为显示一整圈的彩虹色，并且持续向一个方向进行旋转，旋转动作是持续进行的，计算平均每 500ms 会旋转 90 度，如图 1-2 所示。

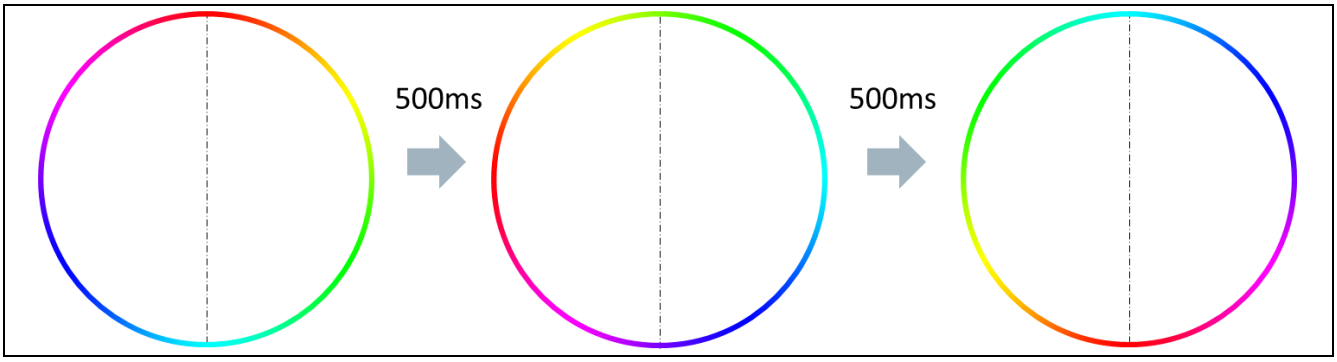


图 1-2 彩虹灯效的图片说明

为避免降低执行效率，在进入灯效控制前创建此灯效需要用到的数组。因为显示原理是以每 2 度为单位进行灯效更新，一个圆周会被切割为 180 份。因此前述数组内容为 180 个 PWM duty 的元素，涵盖了实现彩虹灯效的所有 PWM duty 值(在章节：代码介绍中呈现)。PWM 作为三色 LED 灯输出在圆周中持续的循环，即构成了一整圈彩虹灯效。

在执行 Timer continuous counting mode 时每进入一次中断就往数组中下一个元素做为 PWM duty 的输出，要使彩虹灯效持续旋转而非每一圈彩虹都固定在圆周上的同一个位置，需要搭配 SysTick ，每旋转一圈时检查是否 SysTick 计时已到，若 SysTick 已经到达设定时间，将输出起始的对应值加 1，这样就会朝一个方向做旋转的动作。

1.4 灯效原理-颜色流动

此灯效为显示不同的整圈颜色，并且在中间颜色切换过渡时每 40ms 向旋转方向增加 30 度的下一个颜色，共花费 480ms 来完成一次颜色的切换，如图 1-3 所示。

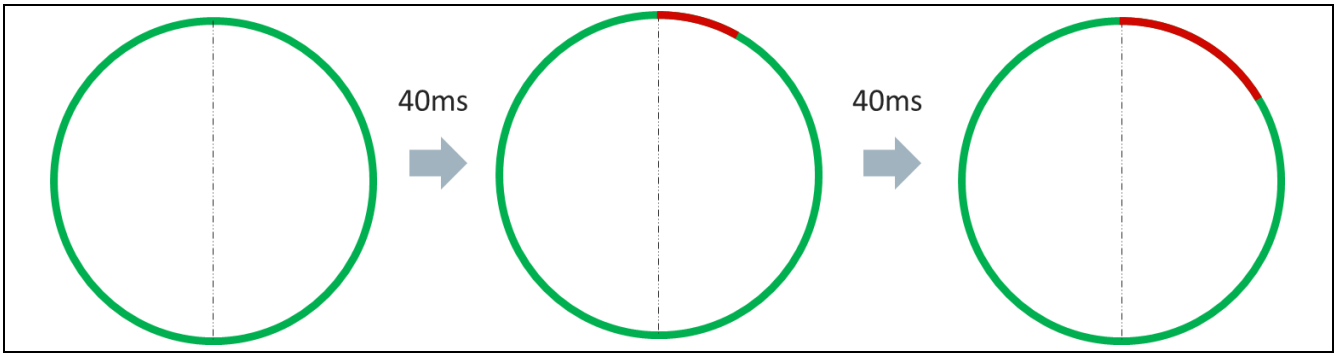


图 1-3 颜色流动灯效的图片说明

此图为由绿色过渡到红色的显示，颜色朝一个方向。每次颜色切换完成后有 1.4 秒的时间维持同一个颜色，接着进行下一次的颜色过渡。此处需要搭配 SysTick clock 做颜色过渡和显示同一个颜色的计时。

1.5 灯效原理-三爪

此灯效为一整圈有三个区域有颜色显示，以十二点钟方向为 0 度，分别为 0~60, 120~180, 240~300 度显示颜色，并且每过 500ms 在圆周旋转 40 度，共旋转三次后即达成一次显示循环，如图 1-4 所示。

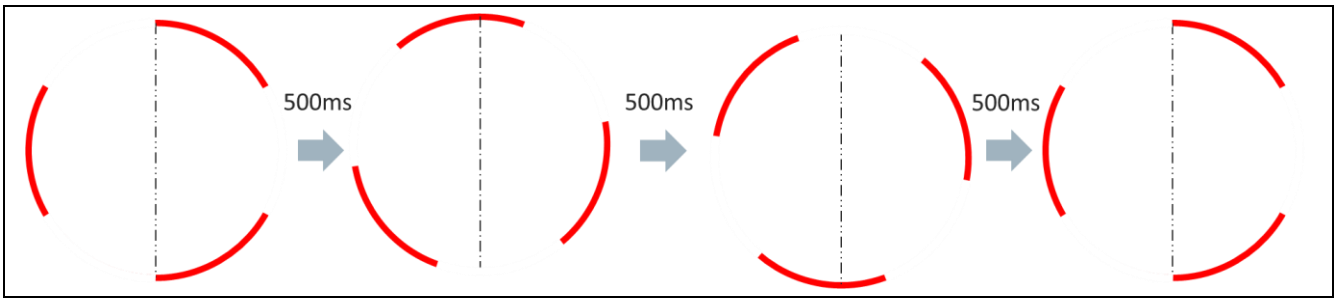


图 1-4 三爪灯效的图片说明(1)

在此之上加入色彩的转换，每旋转一次即更换一次颜色，如图 1-5 所示。

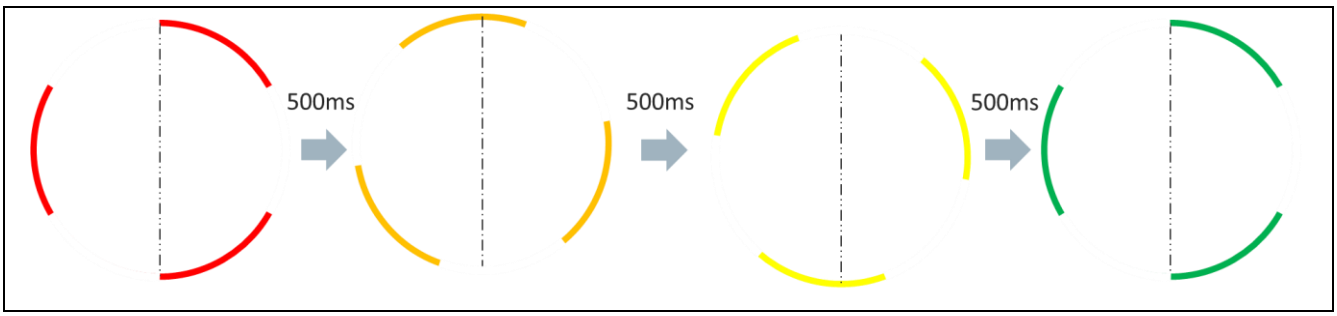


图 1-5 三爪灯效的图片说明(2)

1.6 执行结果

本范例代码执行编译和烧录的结果如图 1-6 所示。

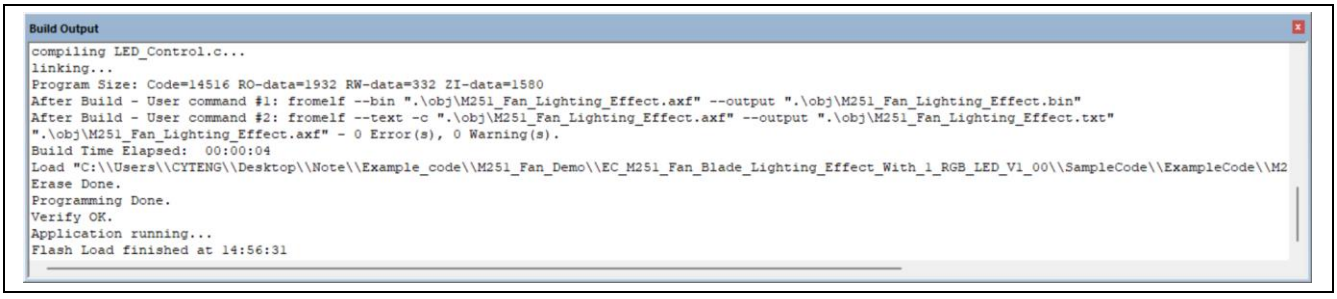


图 1-6 执行编译和烧录结果

此风扇中央是顺时针旋转，左右两侧是逆时针旋转，因此在执行时两侧灯效会不同于中央的风扇，会是相反方向，风扇照片如图 1-7。

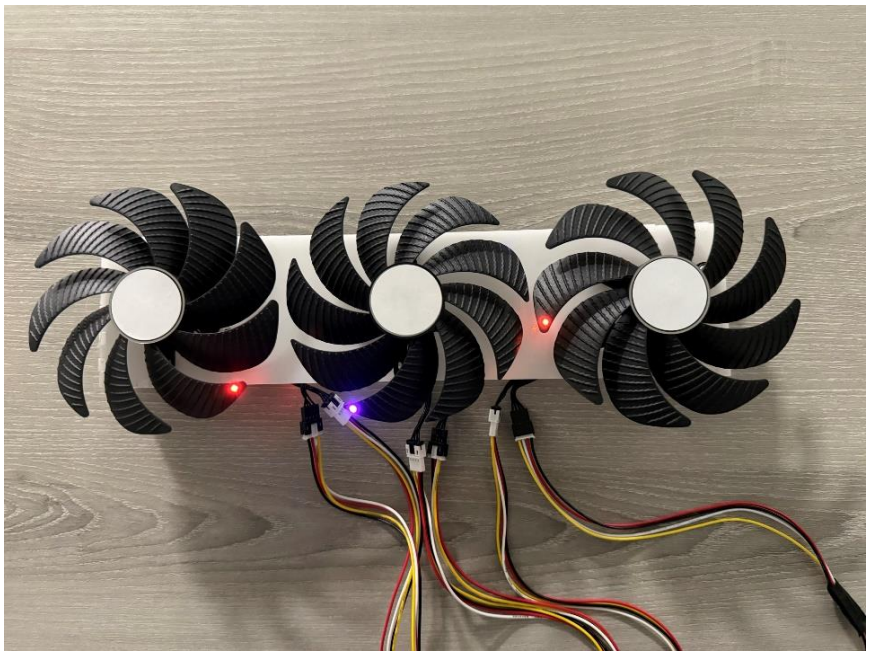


图 1-7 风扇照片

此范例程序会在风扇旋转的时候显示灯效。然而相机拍摄时对影像的保存跟人眼视觉不相同，相机拍摄时会撷取一个极短时间的画面信息进行储存，因此在拍摄风扇时，拍出来的影像会有灯效不满一圈的情况，如图 1-8。

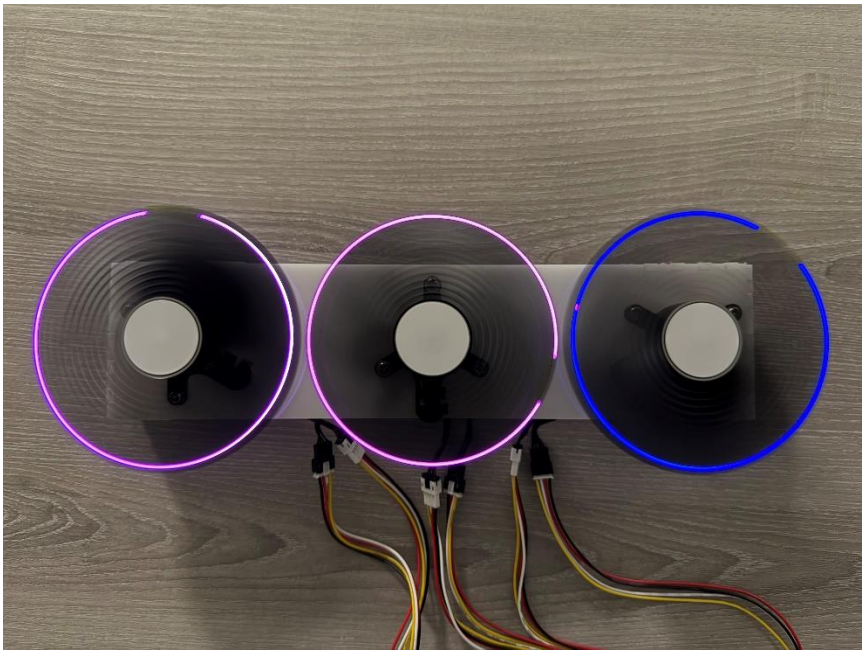


图 1-8 一般拍摄时的相片

这时候可以利用增加曝光时间来改善拍摄的效果。执行结果如图 1-9 至图 1-13 所示。

虽然增加曝光时间可以改善拍摄的灯效，但是因三个风扇的转速并不完全相同，转速慢的风扇在旋转完一圈时，转速快的风扇已经多旋转几度了，这会造成相片在同个曝光时间下，有些风扇部分位置相邻两圈会发生颜色重迭，这个现象在拍摄彩虹灯效时可以明显看出。

彩虹灯效如图 1-9 所示。中央的风扇转速较左右两侧快，造成中间风扇有白色出现。正常情况下想要显示白色，可以让三个颜色的 LED 灯用同个 PWM duty 输出，在这里拍摄出现了白色即代表相邻两圈的颜色在该位置出现了重迭。

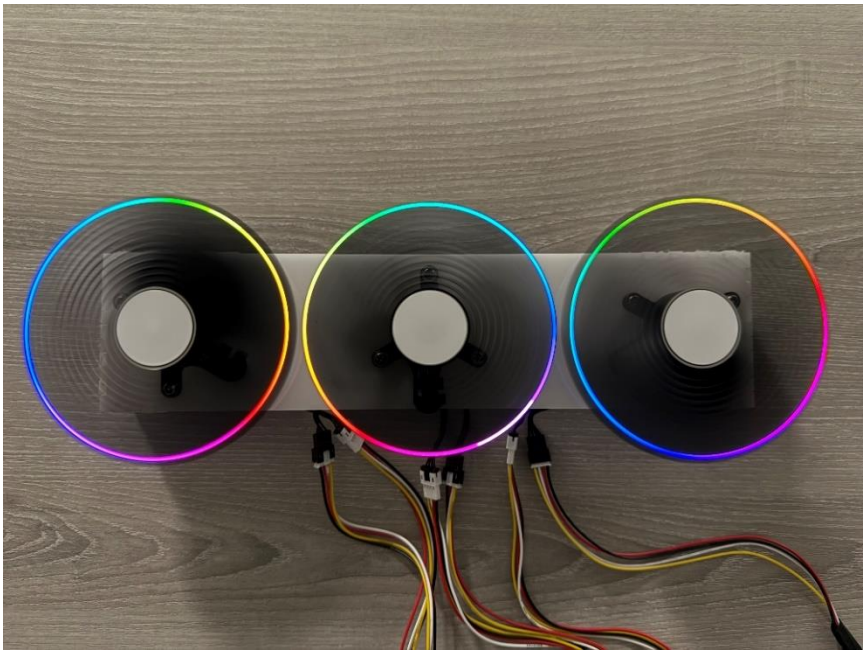


图 1-9 彩虹灯效

颜色流动灯效如图 1-10 和图 1-11 所示。此为一次灯效改变所拍摄的前后对比照片。

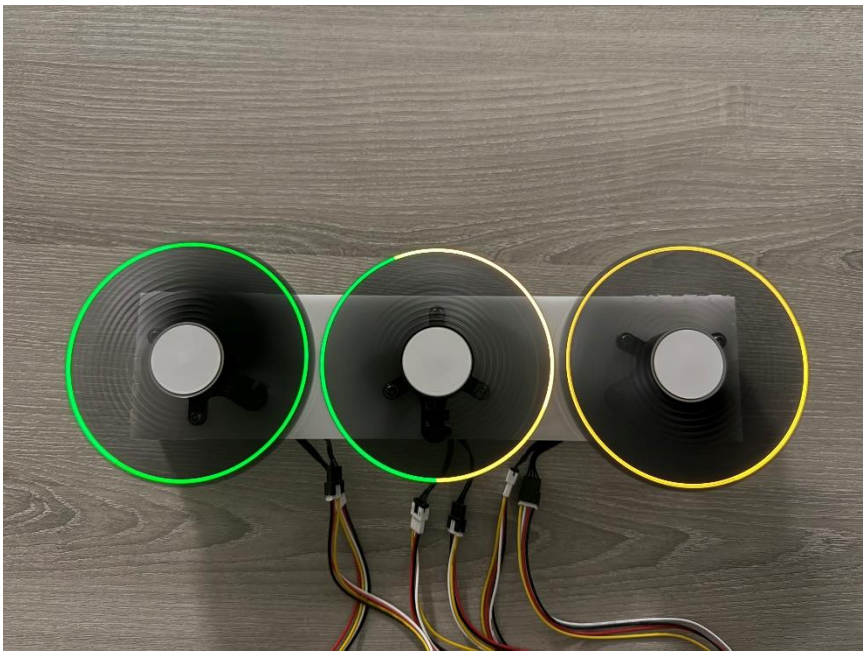


图 1-10 颜色流动灯效(1)

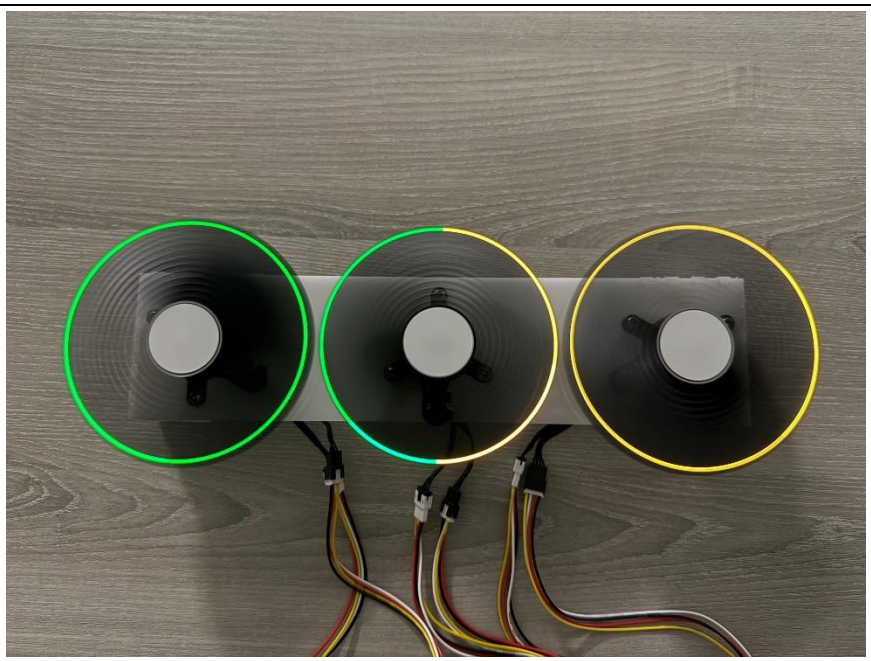


图 1-11 颜色流动灯效(2)

三爪灯效如图 1-12 和图 1-13 所示。此为一次灯效改变所拍摄的前后对比照片。

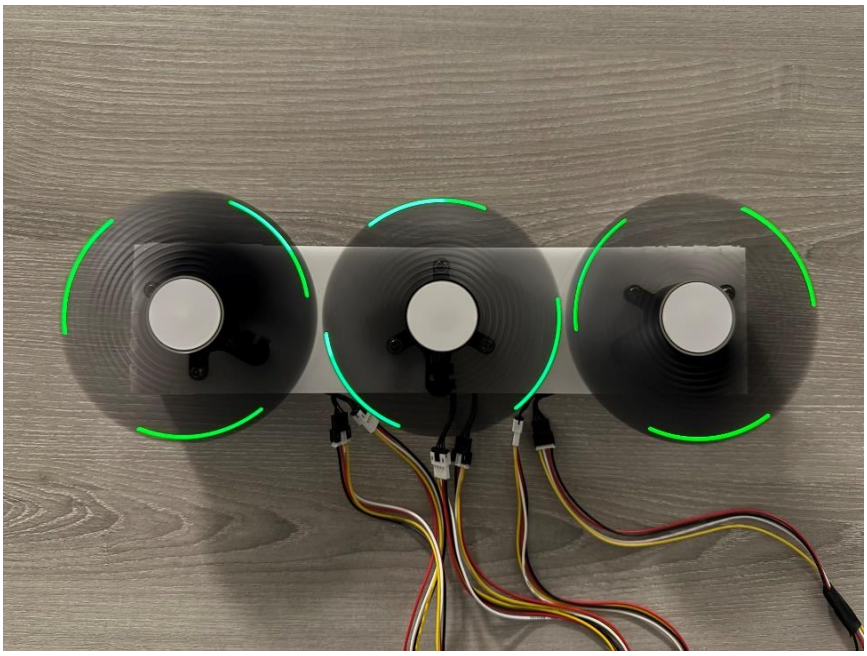


图 1-12 三爪灯效(1)

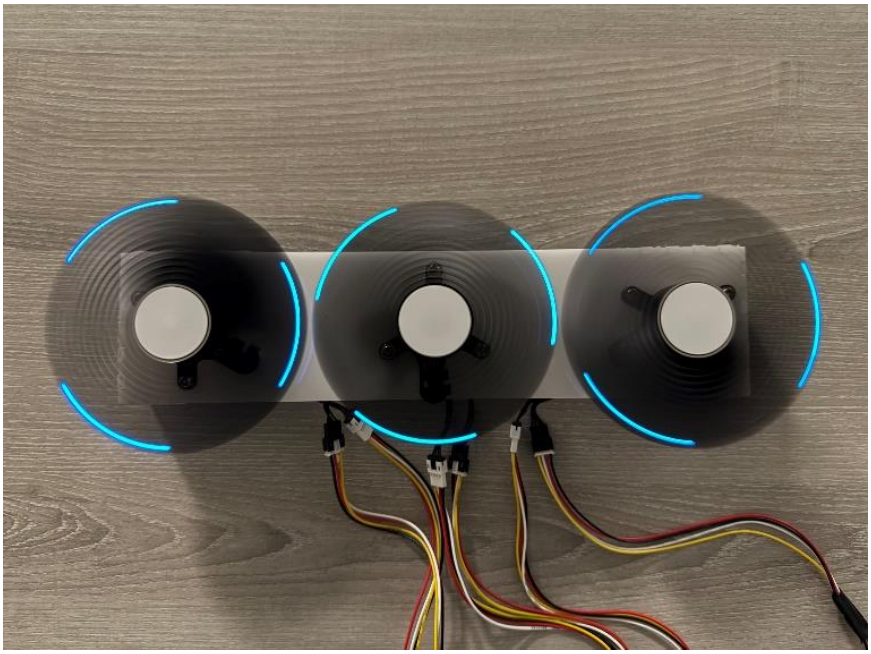


图 1-13 三爪灯效(2)

2. 代码介绍

初始化系统后，对每个需要用到的外设进行初始化的动作。代码位于 main.c。

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

    /* Init System, IP clock and multi-function I/O */
    SYS_Init();

    /* Lock protected registers */
    SYS_LockReg();

    /* Init UART */
    UART0_Init();

    /* Init Timer0, Timer1, Timer2 */
    TIMER0_Init();
    TIMER1_Init();
    TIMER2_Init();

    /* Init All Timer Capture function & enable interrupt */
    TIMER_Capture_Init();

    /* Init Systick */
    SysTick_Initial();

    /* Start SysTick */
    SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;
```

函数 TIMER0_Init 内容，其他 Timer 的初始化内容相似，设定 Timer 为 Continuous mode，并且设定除频器为 1(该数值 0 会在进行除频时以(0+1)进行除频动作)。

```
void TIMER0_Init()
{
    /* Set TIMER0 continuous time-out frequency is 1Hz */
    TIMER_Open(TIMER0, TIMER_CONTINUOUS_MODE, 1);

    /* Set TIMER0 prescale = 1 */
    TIMER_SET_PRESCALE_VALUE(TIMER0, 0);
}
```

函数 TIMER_Capture_Init 内容，Timer 在进行灯光控制同时也兼做风扇马达 TACH 信号的 Capture 动作，此函数即在做 Timer capture 和 Interrupt 启用设定。由于 Timer 需要同时做 Capture 的动作，因此 Timer 必须使用 Continuous mode 来使 Timer 在 Time-out 发生后不重置计数器的值。

```
void TIMER_Capture_Init()
{
    //Enable Timer capture, capture interrupt, interrupt, NVIC
    TIMER_EnableCapture(TIMER0, TIMER_CAPTURE_FREE_COUNTING_MODE,
TIMER_CAPTURE_EVENT_RISING_FALLING);
    TIMER_EnableCaptureInt(TIMER0);
```

```
    TIMER_EnableInt(TIMER0);
    TIMER_EnableCapture(TIMER1, TIMER_CAPTURE_FREE_COUNTING_MODE,
TIMER_CAPTURE_EVENT_RISING_FALLING);
    TIMER_EnableCaptureInt(TIMER1);
    TIMER_EnableInt(TIMER1);
    TIMER_EnableCapture(TIMER2, TIMER_CAPTURE_FREE_COUNTING_MODE,
TIMER_CAPTURE_EVENT_RISING_FALLING);
    TIMER_EnableCaptureInt(TIMER2);
    TIMER_EnableInt(TIMER2);
    NVIC_EnableIRQ(TMR0_IRQn);
    NVIC_EnableIRQ(TMR1_IRQn);
    NVIC_EnableIRQ(TMR2_IRQn);
}
```

函数 SysTick_Initial 内容，对 SysTick 做初始化，动作包含选择 Clock source、更新计数目标值(这个值会在灯效初始化时再被设定一次)、清除计数器值、启用 SysTick 中断。

```
void SysTick_Initial(void)
{
    /* Set SysTick clock source to HCLK */
    SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk;

    /* Set load value to make SysTick period is 250ms */
    SysTick->LOAD = 48000000 / 4 ;

    /* Clear SysTick counter */
    SysTick->VAL = 0;

    /* Enable SysTick interrupt */
    SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;

    /* Enable SysTick NVIC */
    NVIC_EnableIRQ(SysTick_IRQn);
}
```

接续 main 内容，为了控制风扇的转速，需要 MCU 提供一个 PWM 信号来控制风扇转速。此程序使用 QSPI 来仿真 PWM 信号。要使用 QSPI 仿真 PWM 信号，首先须建立风扇转速的 PWM duty 数组以提供 QSPI 输出信号的来源，并使用 PDMA(Peripheral Direct Memory Access)来进行数据传输。如果要理解详细原理，建立控制风扇转速的 PWM duty 数组，这里使用 QPSI 来模拟 PWM 输出信号，可以在 NuForum 参考另一个 Nuvoton Example code [“[M251] Using QSPI with PDMA to Simulate PWM”] 来了解其原理以及应用，在此范例中，利用 QSPI 模拟的 PWM 信号占空比值固定为 50%。相似的程序代码在此不再重述。

```
    /* Setting PDMA Structure for QSPI */
    /* DSCT0 Description*/
    DMA_DESC_QSPI[0].u32Ctl =
        ((40 - 1) << PDMA_DSCT_CTL_TXCNT_Pos) | /* Transfer count is 40 */ \
        PDMA_WIDTH_32 |   /* Transfer width is 32 bits(one word) */ \
        PDMA_SAR_INC  |   /* Source increment size is 32 bits(one word) */ \
        PDMA_DAR_FIX  |   /* Destination increment size is 32 bits(one word) */ \
        PDMA_REQ_SINGLE |  /* Transfer type is burst transfer type */ \
        PDMA_BURST_128 |  /* Burst size is 128. No effect in single transfer type */ \
```

```
        PDMA_TBINTDIS_DISABLE |   /* Disable transfer done and table empty interrupt */ \
        PDMA_OP_SCATTER;  /* Operation mode is scatter-gather mode */

    /* Configure source address */
    DMA_DESC_QSPI[0].u32Src = (uint32_t)Duty_array;

    /* Configure destination address */
    DMA_DESC_QSPI[0].u32Dest = (uint32_t)&QSPI0->TX;

    /* Configure next descriptor table address */
    DMA_DESC_QSPI[0].u32Offset = (uint32_t)&DMA_DESC_QSPI[1] - (PDMA->SCATBA); /* next
descriptor table is table 2 */
    /* next descriptor table is table 2 */

    /* DSCT1 Description*/
    DMA_DESC_QSPI[1].u32Ctl =
        ((40 - 1) << PDMA_DSCT_CTL_TXCNT_Pos) | /* Transfer count is 40 */ \
        PDMA_WIDTH_32 |   /* Transfer width is 32 bits(one word) */ \
        PDMA_SAR_INC |    /* Source increment size is 32 bits(one word) */ \
        PDMA_DAR_FIX |    /* Destination increment size is 32 bits(one word) */ \
        PDMA_REQ_SINGLE |  /* Transfer type is burst transfer type */ \
        PDMA_BURST_128 |  /* Burst size is 128. No effect in single transfer type */ \
        PDMA_TBINTDIS_DISABLE |   /* Disable transfer done and table empty interrupt */ \
        PDMA_OP_SCATTER;  /* Operation mode is scatter-gather mode */

    /* Configure source address */
    DMA_DESC_QSPI[1].u32Src = (uint32_t)Duty_array;

    /* Configure destination address */
    DMA_DESC_QSPI[1].u32Dest = (uint32_t)&QSPI0->TX;

    /* Configure next descriptor table address */
    DMA_DESC_QSPI[1].u32Offset = (uint32_t)&DMA_DESC_QSPI[0] - (PDMA->SCATBA); /* next
descriptor table is table 1 */
    /* next descriptor table is table 1 */

    /* Calculate PWM output duty array */
    Calculate_Duty_array(Duty_right_now);

    /* Init QSPI & PDMA, Start QSPI output */
    QSPI_and_PDMA_Init();
```

接续 main 内容，初始化控制 LED 灯的 PWM，并创建彩虹灯效会用到的数组，数组内容为跑一整轮彩虹效果需要用到的全部 PWM duty 值。并且启动 Timer 和 PWM、指定三个风扇的灯效并做灯效初始化动作。

```
    /* Init PWM output */
    PWM_Init();

    /* Create Rainbow lighting effect array */
    for (uint8_t i = 0; i < 180; i++)
    {
        if (i < 40)
        {
            Rainbow_array[i] = 100;
        }
        else if ((i > 39) && (i < 80))
        {
            Rainbow_array[i] = 100 - 2.5 * (i - 40);
        }
        else if ((i > 79) && (i < 140))
```

```
        {
            Rainbow_array[i] = 0;
        }
        else if ((i > 139) && (i < 180))
        {
            Rainbow_array[i] = 2.5 * (i - 140);
        }
    }

    //Timer start
    TIMER_Start(TIMER0);
    TIMER_Start(TIMER1);
    TIMER_Start(TIMER2);

    /* Start PWM counter */
    PWM_Start(fan_left.PWM_generator, fan_left.PWM_RMSK);
    PWM_Start(fan_left.PWM_generator, fan_left.PWM_GMSK);
    PWM_Start(fan_left.PWM_generator, fan_left.PWM_BMSK);
    PWM_Start(fan_middle.PWM_generator, fan_middle.PWM_RMSK);
    PWM_Start(fan_middle.PWM_generator, fan_middle.PWM_GMSK);
    PWM_Start(fan_middle.PWM_generator, fan_middle.PWM_BMSK);
    PWM_Start(fan_right.PWM_generator, fan_right.PWM_RMSK);
    PWM_Start(fan_right.PWM_generator, fan_right.PWM_GMSK);
    PWM_Start(fan_right.PWM_generator, fan_right.PWM_BMSK);

    /* Init lighting effect: Rainbow lighting mode */
    Lighting_Mode_Setting = RAINBOW_LIGHTING_MODE;
    fan_left.light_mode_choose = Lighting_Mode_Setting ;
    fan_middle.light_mode_choose = Lighting_Mode_Setting ;
    fan_right.light_mode_choose = Lighting_Mode_Setting ;
    Rainbow_initial(&fan_left);
    Rainbow_initial(&fan_middle);
    Rainbow_initial(&fan_right);
```

根据上述所创建的 Rainbow_array 表如下，在彩虹灯效套用该数组时，红色的 Duty 值为 Rainbow_array[i]，绿色的 Duty 值为 Rainbow_array[i+60]，蓝色的 Duty 值为 Rainbow_array[i+120]，套用数组的元素项若大于 179 则取对 180 的余数。这样就变成表 2-1 的对照表，刚好在经过 180 个元素会变成一轮彩虹颜色的循环。

i	0	20	40	60	80	100	120	140	160
Rainbow_array[i] (R)	100	100	100	50	0	0	0	0	50
G	0	0	50	100	100	100	50	0	0
B	50	0	0	0	0	50	100	100	100

表 2-1 Rainbow_array 对照表

函数 PWM_Init 内容，设定 PWM 的初始频率与 Duty 值，并将输出信号做反向(Inverse)方便后续计算。请注意，建议 PWM 频率要高于风扇旋转 1 度时间去换算的频率，原因为若 PWM 频

率不够高，风扇在旋转时 PWM 控制的灯光不够跑完一轮周期，会造成灯光显示分层，达不到想要的灯光效果。

举例说明，欲在 0 度~2 度的位置显示灯光紫色(R,G,B)=(128,0,255)，若 PWM 输出频率换算的周期为风扇马达旋转 2 度的时间，由 0 度位置开始新一轮的 PWM 输出，该灯光会在 0 度~1 度显示亮紫色(R,G,B)=(255,0,255)，在 1 度~2 度显示蓝色(R,G,B)=(0,0,255)，这即是因为 PWM 频率不够快所导致的结果，如图 2-1。

```
void PWM_Init()
{
    //For PWM Initial Setting
    PWM_ConfigOutputChannel(fan_left.PWM_generator, fan_left.PWM_Rchannel, 50000,
100);//PWM0 ch0
    PWM_ConfigOutputChannel(fan_left.PWM_generator, fan_left.PWM_Gchannel, 50000,
100);//PWM0 ch1
    PWM_ConfigOutputChannel(fan_left.PWM_generator, fan_left.PWM_Bchannel, 50000,
100);//PWM0 ch2
    PWM_ConfigOutputChannel(fan_middle.PWM_generator, fan_middle.PWM_Rchannel, 50000,
100);//PWM0 ch3
    PWM_ConfigOutputChannel(fan_middle.PWM_generator, fan_middle.PWM_Gchannel, 50000,
100);//PWM0 ch4
    PWM_ConfigOutputChannel(fan_middle.PWM_generator, fan_middle.PWM_Bchannel, 50000,
100);//PWM0 ch5
    PWM_ConfigOutputChannel(fan_right.PWM_generator, fan_right.PWM_Rchannel, 50000,
100);//PWM1 ch3
    PWM_ConfigOutputChannel(fan_right.PWM_generator, fan_right.PWM_Gchannel, 50000,
100);//PWM1 ch4
    PWM_ConfigOutputChannel(fan_right.PWM_generator, fan_right.PWM_Bchannel, 50000,
100);//PWM1 ch5

    /* Enable output of PWM0, PWM1 */
    PWM_EnableOutput(fan_left.PWM_generator, fan_left.PWM_RMSK);
    PWM_EnableOutput(fan_left.PWM_generator, fan_left.PWM_GMSK);
    PWM_EnableOutput(fan_left.PWM_generator, fan_left.PWM_BMSK);
    PWM_EnableOutput(fan_middle.PWM_generator, fan_middle.PWM_RMSK);
    PWM_EnableOutput(fan_middle.PWM_generator, fan_middle.PWM_GMSK);
    PWM_EnableOutput(fan_middle.PWM_generator, fan_middle.PWM_BMSK);
    PWM_EnableOutput(fan_right.PWM_generator, fan_right.PWM_RMSK);
    PWM_EnableOutput(fan_right.PWM_generator, fan_right.PWM_GMSK);
    PWM_EnableOutput(fan_right.PWM_generator, fan_right.PWM_BMSK);

    //PWM Inverse
    PWM0->POLCTL |= 0x3FUL; //ch0~ch5
    PWM1->POLCTL |= 0x38UL; //ch3~ch5
}
```

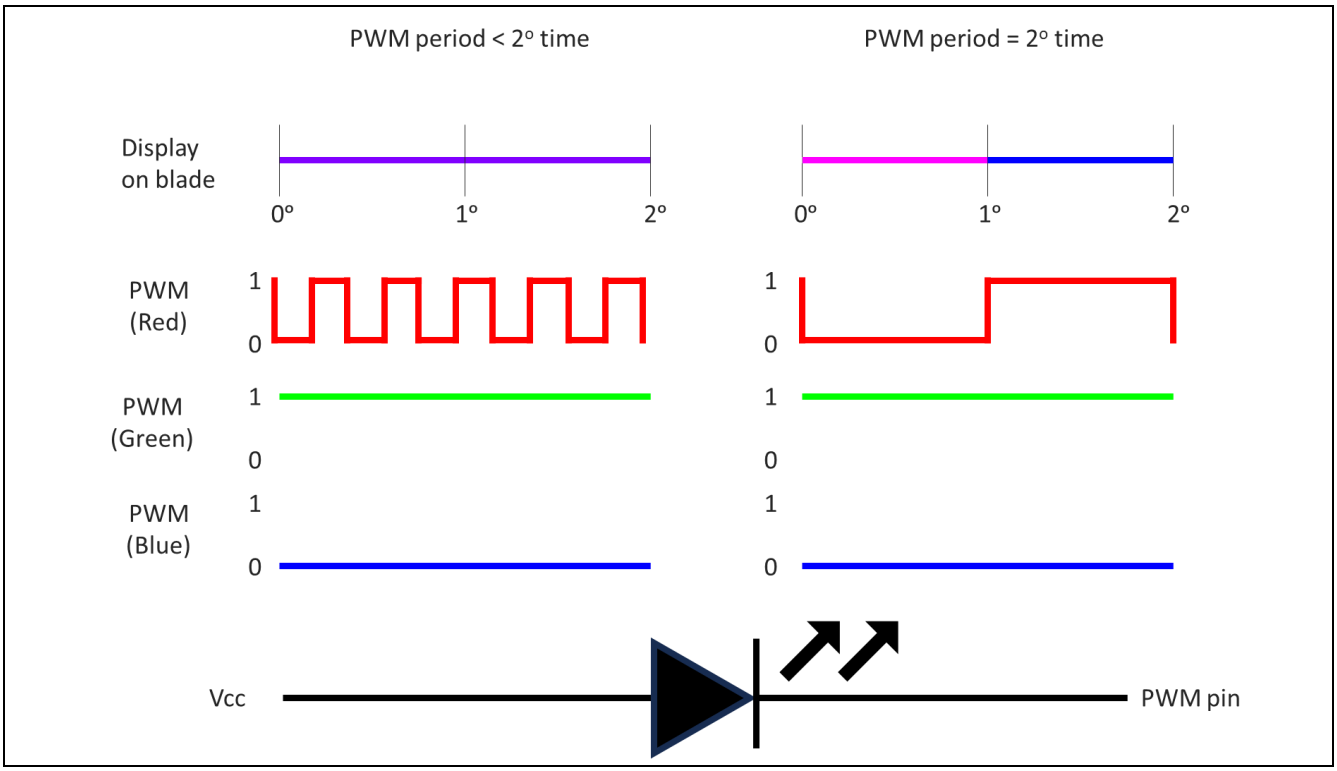


图 2-1 PWM 频率对灯效影响说明

其他对灯效初始化的函数，例如 Rainbow_initial 函数是关于切换不同灯效时需要将一些变量初始化，以确保切换前的灯效变量不会影响到接下来的灯效。函数一开始的 PWM_EnableOutput 是因为三爪灯效可能会关闭当前 PWM 输出，因此需要在初始化的时候做一个固定开启的动作。最后是切换 SysTick 的目标值并重置 SysTick 重新开始计时。

```
void Rainbow_initial(struct fan* fan_setting)
{
    PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_RMSK);
    PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_GMSK);
    PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_BMSK);

    /* Init variable*/
    fan_setting->update_count = fan_setting->Capture_Init_count_Offset;
    fan_setting->fan_count = 0;
    fan_setting->count_initial = 0;
    fan_setting->R_duty = 0;
    fan_setting->G_duty = 0;
    fan_setting->B_duty = 0;
    fan_setting->systick_flag = 0;

    /* Setting SysTick time-out value */
    SysTick->LOAD = RAINBOW_SYSTICK_LOAD_VALUE * fan_setting->Speed;

    /* Reset SysTick */
    SysTick->VAL = 0;
}
```

这边再列出 Color_shift_initial 函数内容，这边相较 Rainbow_initial 函数多了将颜色值填入 Color_R_Duty 等三色的动作，这是因为不同于 Rainbow 灯效是用 Rainbow_array，这边是用 Default_Color 数组去当作过渡的颜色去计算灯效，并且将 RGB 的 0~255 值转换为 PWM Duty 的 0~100 值(后面的亮度可以按照使用者想要的亮度去调整结构体的内容，正常使用时不需要去更动)。Color_Transition 变量是用于表示这个风扇此时是处于变化前的颜色、或是正在处于颜色过渡的过程，三个风扇的 Color_Transition 初始值不一样可以让三个风扇有顺序的进行灯效。

```
void Color_shift_initial(struct fan* fan_setting)
{
    PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_RMSK);
    PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_GMSK);
    PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_BMSK);

    /* Init Basic Color */
    for (uint8_t i = 0; i < fan_setting->Color_Num; i++)
    {
        fan_setting->Color_R_Duty[i] =  Default_Color[i][0] * DUTY_MAX / COLOR_VALUE_MAX *
fan_setting->Brightness / BRIGHTNESS_MAX;
        fan_setting->Color_G_Duty[i] =  Default_Color[i][1] * DUTY_MAX / COLOR_VALUE_MAX *
fan_setting->Brightness / BRIGHTNESS_MAX;
        fan_setting->Color_B_Duty[i] =  Default_Color[i][2] * DUTY_MAX / COLOR_VALUE_MAX *
fan_setting->Brightness / BRIGHTNESS_MAX;
    }

    /* Init variable*/
    fan_left.Color_Transition = 2;
    fan_middle.Color_Transition = 1;
    fan_right.Color_Transition = 0;
    fan_setting->update_count = fan_setting->Capture_Init_count_Offset;
    fan_setting->fan_count = 0;
    fan_setting->count_initial = 0;
    fan_setting->color_select = 0;
    fan_setting->change_count = 0;
    fan_setting->systick_flag = 0;

    /* Setting SysTick time-out value */
    SysTick->LOAD = COLOR_SHIFT_SYSTICK_LOAD_VALUE * fan_setting->Speed;

    /* Reset SysTick */
    SysTick->VAL = 0;
}
```

在介绍灯效之前，需要先说明一下 SysTick 中断和 Timer 中断内容，两个中断都与控制灯效有着密切的关系，前者是用来帮助计时切换灯效到下一帧的时间，后者则是负责对风扇马达的 TACH 进行 Capture 和在风扇上每个位置需要输出控制 LED 灯的 PWM 信号。

SysTick 中断函数内容，每进来两次中断函数就将切换信号的 Flag 设为 1，这些 Flag 会在执行灯效的时候被清除。因为每个风扇转速不一样，flag 不应该三个风扇共享。基于上述原因，不建议在灯效函数中直接检查 SysTick 的 Time-out flag，这样 flag 被清除时可能会有风扇没切换到灯效。

```
void SysTick_Handler(void)
{
    /* Reset SysTick */
    SysTick->VAL = 0;

    /* Set flag = 1 when enter SysTick interrupt twice */
    if (SysTick_count > 1)
    {
        fan_left.systick_flag = 1;
        fan_middle.systick_flag = 1;
        fan_right.systick_flag = 1;
        SysTick_count = 0;
    }

    /* Calculate SysTick count */
    SysTick_count++;
}
```

Timer 中断函数内容，这里以 TMR0_IRQHandler 举例，Timer1 和 Timer2 的中断程序同理。首先检查 Capture flag，该 Flag 用以侦测风扇马达在运作时回传的 TACH 信号，Capture 此信号有两个作用，一个是借由此信号可以得知风扇当前的转速，灯效更新会利用风扇转速作为时间依据来做更新。第二个是借由此信号可以得知风扇现在旋转到的”相对位置”，因为风扇马达每次在上电启动时起始位置不一样，因此 TACH 开始产生的位置也会不一样，没有办法用 TACH 信号来表示风扇马达的绝对位置。得知风扇的相对位置后记录进入 Capture 中断次数可以得知现在风扇是否旋转了一圈。

在旋转了一圈后，更新风扇的转速来给下次 Timer 的灯效更新做参考，与让以一圈为单位进行循环的灯效有一个起始参考点，除了让我们在计算灯效的参数上更方便外，也让风扇马达旋转中相邻两圈的灯效不至于差太多影响最后灯效的呈现。结尾清除可能发生的 Time-out 中断 Flag。

```
void TMR0_IRQHandler(void)
{
    /* Check capture flag=1, otherwise is time out interrupt occur */
    if (TIMER_GetCaptureIntFlag(TIMER0) == 1)
    {
        /* Clear capture interrupt flag */
        TIMER_ClearCaptureIntFlag(TIMER0);

        /* Calculate capture count, if perform four times capture, it means the fan motor
has completed one revolution.  */
        fan_left.Capture_Init_count++;
```

```
        /* Record the capture position when the motor starts rotating after being powered
on.*/
        fan_left.Capture_Init_count = fan_left.Capture_Init_count % (4 +
fan_left.Capture_Init_count_Start_position) + fan_left.Capture_Init_count_Start_position;

        /* If the fan motor has completed one revolution. */
        if ((fan_left.update_count >  3) && (fan_left.Capture_Init_count ==
+fan_left.Capture_Init_count_Offset))
        {
            /* Update fan speed */
            Update_Timer_Capture(&fan_left);

            /* Initial fan_count & determine PWM output  */
            Timer_capture_function(&fan_left);

            /* Init update count(capture count) */
            fan_left.update_count = 0;

            /* Setting next Timer time-out value */
            TIMER0->CMP = TIMER0->CNT + fan_left.CalCMP[3];

            /* Clear time out interrupt flag(TIF) to prevent re-entering the interrupt
immediately after exiting the interrupt  */
            TIMER_ClearIntFlag(TIMER0);
            NVIC_ClearPendingIRQ(TMR0_IRQn);

        }

        /* Calculate update count(capture count) */
        fan_left.update_count++;
    }
```

Update_Timer_Capture 函数内容，通过相邻两次旋转一圈的 Timer capture 值的差值来得出实际的计数值，该值即为旋转 360 度需要的时间，后续灯效以旋转 2 度为一次更新，因此将该值除以 180 并储存，用以后续计算 Continuous mode 下次 Time-out 时间。

```
void Update_Timer_Capture(struct fan* fan_setting)
{
    //Put t-1 count value in CalCMP[1],  Scan Capture and put data in CalCMP[0],
Difference value Calculate in CalCMP[2]
    fan_setting->CalCMP[1] = fan_setting->CalCMP[0];
    fan_setting->CalCMP[0] = TIMER_GetCaptureData(fan_setting->Timer_Generator);
    fan_setting->CalCMP[2] = fan_setting->CalCMP[0] - fan_setting->CalCMP[1];
    fan_setting->CalCMP[2] = fan_setting->CalCMP[2] & 0xffffff;
    fan_setting->CalCMP[3]=(fan_setting->CalCMP[2]/180)& 0xffffff;
}
```

接续 Timer 中断函数内容。在没有 Capture flag 时进入 Timer 中断表示风扇旋转了 2 度，需要切换灯效，这时进入灯运作函数 Timer_Light_active_function，变更 LED 灯显示。结束变更 LED 灯后，由于 Timer 为 Continuous mode，需要重新设定 TIMER_CMP 值，以让 Timer 知道

下次进来中断的时间(风扇再旋转 2 度后的时间)。图 2-2 说明了一整圈圆周上触发 Timer 中断的类型。

```
    /* Time out interrupt */
    else if ((TIMER_GetCaptureIntFlag(TIMER0) == 0))
    {
        /* Clear time out interrupt flag */
        TIMER_ClearIntFlag(TIMER0);

        /* Active lighting effect */
        Timer_Light_active_function(&fan_left);

        /* Setting next Timer time-out value */
        TIMER0->CMP = TIMER0->CMP + fan_left.CalCMP[3];
    }
}
```

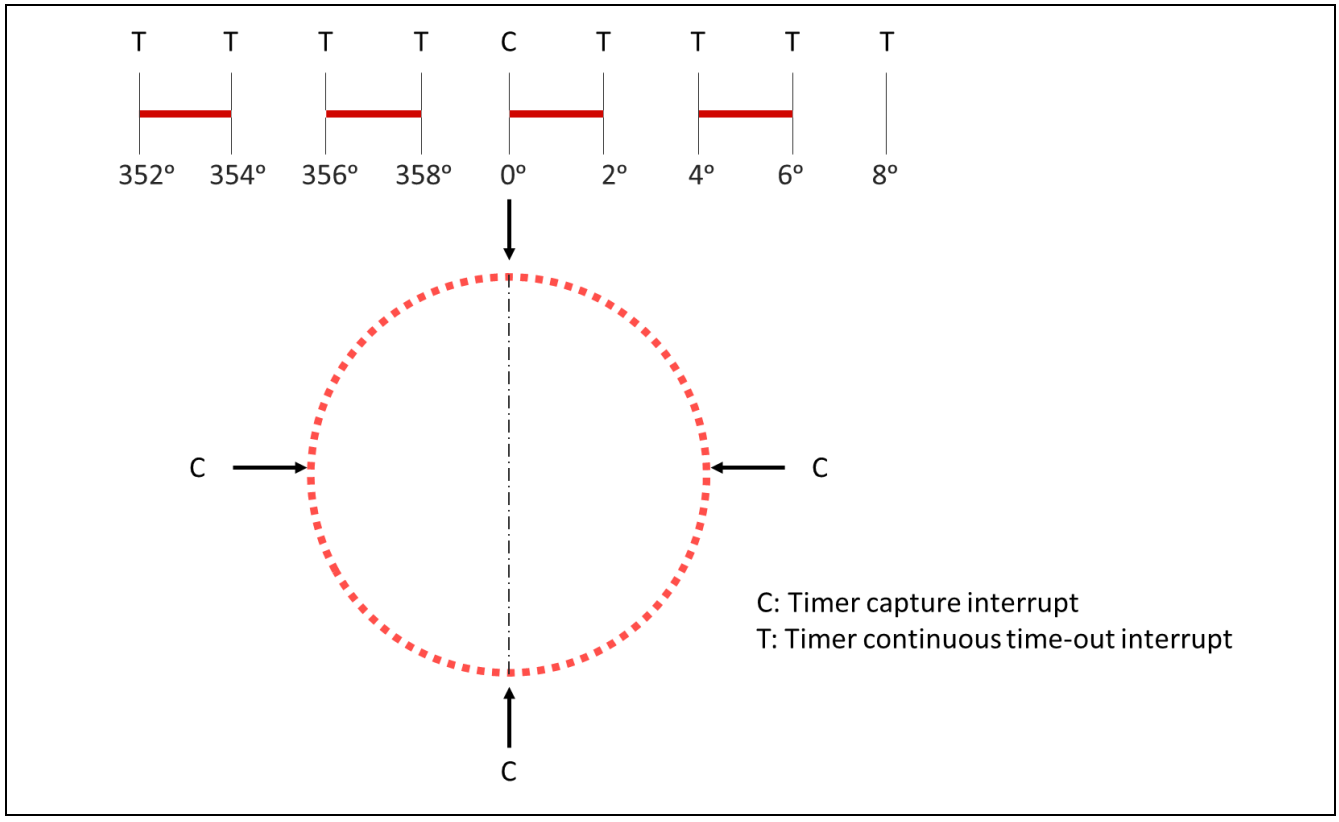


图 2-2 圆周与 Timer Interrupt 说明

最后是位于 LED_Control.c 中的灯效函数，这些函数在上述的 Timer 中断程序里面会用到，分成两个部分，一个是在上述风扇旋转一圈的更新灯效，另一个是风扇旋转 2 度后的更新灯效。

在风扇旋转一圈后，使用 Timer_capture_function 函数将用来持续记录风扇灯效的主要变量:fan_count 做一圈参考点的重置，这样可以不因相邻两圈的 fan_count 值不一样造成灯效严重飘动(若没有做参考点的重置)。

在颜色流动中，除了做参考点的重置外，还有做第一次自参考点重置完的第一次灯效显示。Color_Transition 变量如同 Color_shift_initial 函数的介绍，在该变量值等于 3 时表示正处于颜色过渡中，在重置点时需要显示变化后的颜色;在该变量值小于 3 时表示处于变化前的颜色，在重置点时需要显示变化前的颜色。

```
void Timer_capture_function(struct fan* fan_setting)
{
    switch (fan_setting->light_mode_choose)
    {
        case RAINBOW_LIGHTING_MODE:
        {
            /* Reset fan_count at capture point */
            fan_setting->fan_count = fan_setting->count_initial;
        }
        break;

        case COLOR_SHIFT_LIGHTING_MODE:
        {
            /* Determine the value of the capture point based on the value of
Color_Transition */
            if (fan_setting->Color_Transition == 3)
            {
                fan_setting->u32NewCMR_R = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Rchannel, fan_setting->Color_R_Duty[(fan_setting->color_select + 1) %
fan_setting->Color_Num], 100);
                fan_setting->u32NewCMR_G = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Gchannel, fan_setting->Color_G_Duty[(fan_setting->color_select + 1) %
fan_setting->Color_Num], 100);
                fan_setting->u32NewCMR_B = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Bchannel, fan_setting->Color_B_Duty[(fan_setting->color_select + 1) %
fan_setting->Color_Num], 100);
            }
            else if (fan_setting->Color_Transition < 3)
            {
                fan_setting->u32NewCMR_R = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Rchannel, fan_setting->Color_R_Duty[fan_setting->color_select ], 100);
                fan_setting->u32NewCMR_G = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Gchannel, fan_setting->Color_G_Duty[fan_setting->color_select ], 100);
                fan_setting->u32NewCMR_B = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Bchannel, fan_setting->Color_B_Duty[fan_setting->color_select ], 100);
            }

            /* Reset fan_count at capture point */
            fan_setting->count_initial = 0;
            fan_setting->fan_count = fan_setting->count_initial;
        }
        break;

        case CLAWS_LIGHTING_MODE:
```

```
        {
            /* Reset fan_count at capture point */
            fan_setting->fan_count = fan_setting->count_initial;
        }
        break;

    }

    /* Setting comparator value, to change PWM duty */
    PWM_SET_CMR(fan_setting->PWM_generator, fan_setting->PWM_Rchannel,
fan_setting->u32NewCMR_R);
    PWM_SET_CMR(fan_setting->PWM_generator, fan_setting->PWM_Gchannel,
fan_setting->u32NewCMR_G);
    PWM_SET_CMR(fan_setting->PWM_generator, fan_setting->PWM_Bchannel,
fan_setting->u32NewCMR_B);
}
```

Timer_Light_active_function 函数内容。该函数为大部分进入 Timer 中断时要使用的函数，根据当前为何种灯效来进入对应的计算，接下来会一个一个进行说明。

彩虹灯效内容。在 SysTick 发生 Time-out 后，更改 count_initial 的值用以让进入 Timer_capture_function 后修改 fan_count 为该值。接下来根据 fan_count 值(这边红色、绿色、蓝色的值各不相同)，选择对应的 Rainbow_array 数组的元素作为 PWM Duty，即可在对应的位置显示，一整个圆周连起来即为彩虹色，如图 2-3 所示。

```
void Timer_Light_active_function(struct fan* fan_setting)
{
    switch (fan_setting->light_mode_choose)
    {
        case RAINBOW_LIGHTING_MODE:
        {
            /* When systick_flag =1, change next lighting effect frame */
            if (fan_setting->systick_flag == 1) // flag 0->1 : 11ms
            {
                fan_setting->systick_flag = 0;
                fan_setting->count_initial++;
                fan_setting->count_initial = fan_setting->count_initial % (360 /
COUNT_ANGLE_UNIT);
            }

            /* Change PWM output each time when enter interrupt */
            fan_setting->R_duty = Rainbow_array[fan_setting->fan_count];
            fan_setting->G_duty = Rainbow_array[fan_setting->fan_count_G];
            fan_setting->B_duty = Rainbow_array[fan_setting->fan_count_B];
            fan_setting->u32NewCMR_R = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Rchannel, fan_setting->R_duty, 100);
            fan_setting->u32NewCMR_G = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Gchannel, fan_setting->G_duty, 100);
            fan_setting->u32NewCMR_B = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Bchannel, fan_setting->B_duty, 100);

            fan_setting->fan_count++;
            fan_setting->fan_count = fan_setting->fan_count % (COUNT_ANGLE_360);
```

```
            /* Update fan_count_G & fan_count_B value */
            fan_setting->fan_count_G = fan_setting->fan_count + 120;
            fan_setting->fan_count_G = fan_setting->fan_count_G % (COUNT_ANGLE_360);
            fan_setting->fan_count_B = fan_setting->fan_count + 60;
            fan_setting->fan_count_B = fan_setting->fan_count_B % (COUNT_ANGLE_360);
        }
        break;
    }
```

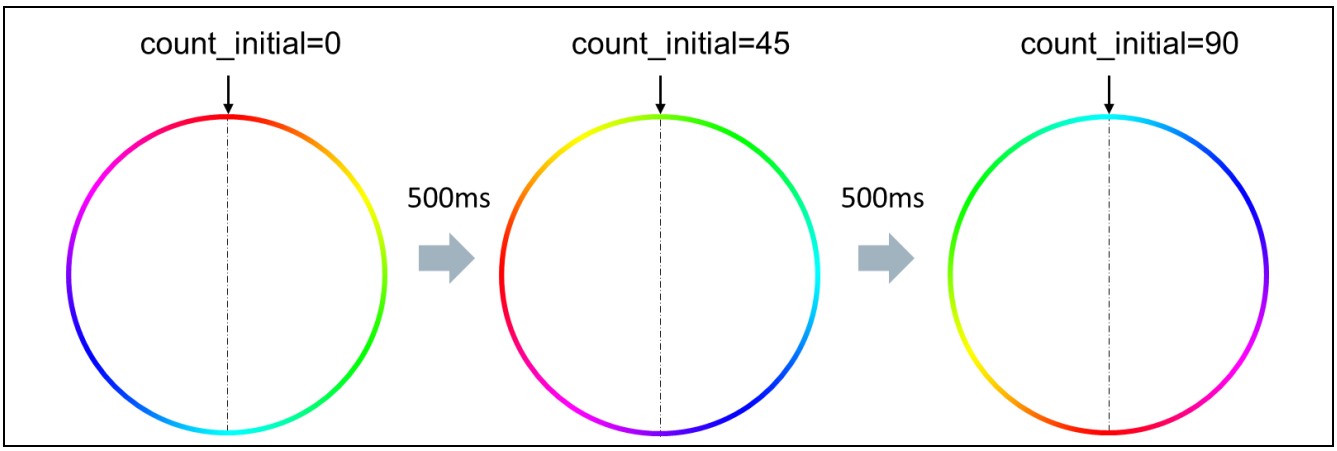


图 2-3 彩虹灯效说明

颜色流动灯效内容，在 SysTick 发生 Time-out 后，更改 change_count 的值用以接下来计算和 Timer_capture_function 的计算。change_count 为每一圈灯效进行时颜色 1 转为颜色 2 的点，因为颜色流动中是每隔一段时间才发生一次颜色的往前过渡，所以要由 change_count 来记录现在已经过渡到圆周中哪个位置。这样连续的进行灯效人眼看起来就好像逐渐由颜色 1 转换到颜色 2。Color_Transition 如同 Timer_capture_function 的介绍。

在进行颜色过渡时(Color_Transition=3)根据 change_count 来判断需要输出颜色 1 还是颜色 2。其余时间(Color_Transition<3)则输出颜色 1。

进行颜色过渡时(Color_Transition=3)如图 2-4 所示，change_count 以 15(COUNT_ANGLE_30)为一个单位增加，表示每一次 SysTick 到 Time-out 时增加 30 度，这样进行一圈就能完成一次灯效。

```
        case COLOR_SHIFT_LIGHTING_MODE:
        {
            /* When systick_flag =1, change next lighting effect frame */
            if (fan_setting->systick_flag == 1)
            {
                fan_setting->systick_flag = 0;
                fan_setting->change_count = fan_setting->change_count + (COUNT_ANGLE_30);
```

```
                if (fan_setting->change_count > (COUNT_ANGLE_30 * 11))
                {
                    fan_setting->change_count = 0;

                    if (fan_setting->Color_Transition < 3)
                    {
                        fan_setting->Color_Transition++;
                    }
                    else
                    {
                        fan_setting->color_select = (fan_setting->color_select + 1) %
fan_setting->Color_Num;
                        fan_setting->Color_Transition = 0;
                    }
                }
            }

            /* Set the lighting effect according to Color_Transition */
            if (fan_setting->Color_Transition == 3)
            {
                if (fan_setting->fan_count < fan_setting->change_count)
                {
                    fan_setting->u32NewCMR_R = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Rchannel, fan_setting->Color_R_Duty[(fan_setting->color_select + 1) %
fan_setting->Color_Num], 100);
                    fan_setting->u32NewCMR_G = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Gchannel, fan_setting->Color_G_Duty[(fan_setting->color_select + 1) %
fan_setting->Color_Num], 100);
                    fan_setting->u32NewCMR_B = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Bchannel, fan_setting->Color_B_Duty[(fan_setting->color_select + 1) %
fan_setting->Color_Num], 100);
                }
                else if (fan_setting->fan_count > (fan_setting->change_count - 1))
                {

                    fan_setting->u32NewCMR_R = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Rchannel, fan_setting->Color_R_Duty[fan_setting->color_select], 100);
                    fan_setting->u32NewCMR_G = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Gchannel, fan_setting->Color_G_Duty[fan_setting->color_select], 100);
                    fan_setting->u32NewCMR_B = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Bchannel, fan_setting->Color_B_Duty[fan_setting->color_select], 100);

                }
            }
            else if (fan_setting->Color_Transition < 3)
            {
                fan_setting->u32NewCMR_R = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Rchannel, fan_setting->Color_R_Duty[fan_setting->color_select], 100);
                fan_setting->u32NewCMR_G = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Gchannel, fan_setting->Color_G_Duty[fan_setting->color_select], 100);
                fan_setting->u32NewCMR_B = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Bchannel, fan_setting->Color_B_Duty[fan_setting->color_select], 100);
            }

            fan_setting->fan_count++;
            fan_setting->fan_count = fan_setting->fan_count % (COUNT_ANGLE_360);
        }
        break;
```

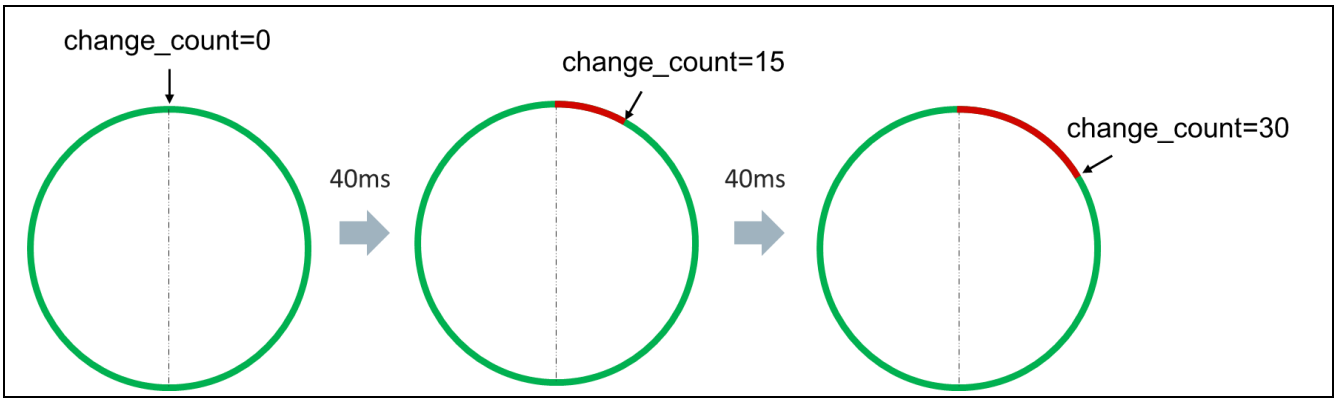


图 2-4 颜色流动灯效说明

三爪灯效内容，在 SysTick 发生 Time-out 后，更改 count_initial 的值用以让进入 Timer_capture_function 后修改 fan_count 为该值。再来修改 LED 灯显示的颜色至下一个颜色，这边用 color_select 表应显示数组内的哪个颜色，Color_Num 为该数组总共有多少颜色，在 color_select 超过数组内颜色时将 color_select 归零。

然后就是呈现三爪的效果，利用在特定区域取消 PWM 输出和特定区域启用 PWM 输出来达成三爪的效果。举例说明，图 2-5 中最左边 count_initial=0，由 12 点钟方向顺时针每 60 度变更一次亮暗。在 count_initial=20 时，整个灯效往逆时针方向移动 40 度。在 count_initial=40 时，整个灯效再往逆时针方向移动 40 度。三次即可达成一次灯效在位置上的循环。

颜色与灯效位置上的搭配即达成此灯效的显示。

```
        case CLAWS_LIGHTING_MODE:
        {
            /* When systick_flag =1, change next lighting effect frame */
            if (fan_setting->systick_flag == 1)
            {
                fan_setting->systick_flag = 0;
                fan_setting->count_initial = (fan_setting->count_initial + 20) % 60;

                fan_setting->color_select = (fan_setting->color_select + 1) %
fan_setting->Color_Num;

                fan_setting->u32NewCMR_R = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Rchannel, fan_setting->Color_R_Duty[fan_setting->color_select], 100);
                fan_setting->u32NewCMR_G = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Gchannel, fan_setting->Color_G_Duty[fan_setting->color_select], 100);
                fan_setting->u32NewCMR_B = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Bchannel, fan_setting->Color_B_Duty[fan_setting->color_select], 100);


                PWM_SET_CMR(fan_setting->PWM_generator, fan_setting->PWM_Rchannel,
fan_setting->u32NewCMR_R);
```

```
                PWM_SET_CMR(fan_setting->PWM_generator, fan_setting->PWM_Gchannel,
fan_setting->u32NewCMR_G);
                PWM_SET_CMR(fan_setting->PWM_generator, fan_setting->PWM_Bchannel,
fan_setting->u32NewCMR_B);
            }

            //Claws bright zone in first frame(angle): 0~60  120~180, 240~300
            if ((fan_setting->fan_count < COUNT_ANGLE_60) | (((COUNT_ANGLE_120 - 1) <
fan_setting->fan_count) & (fan_setting->fan_count < COUNT_ANGLE_180)) | \
                            (((COUNT_ANGLE_240 - 1) < fan_setting->fan_count) &
(fan_setting->fan_count < COUNT_ANGLE_300)))
            {
                PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_RMSK);
                PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_GMSK);
                PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_BMSK);

            }
            //Claws dark zone in first frame(angle): 60~120, 180~240, 300~360
            else if ((((COUNT_ANGLE_60 - 1) < fan_setting->fan_count) &
(fan_setting->fan_count < COUNT_ANGLE_120)) | \
                            (((COUNT_ANGLE_180 - 1) < fan_setting->fan_count) &
(fan_setting->fan_count < COUNT_ANGLE_240)) | (((COUNT_ANGLE_300 - 1) <
fan_setting->fan_count) & (fan_setting->fan_count < COUNT_ANGLE_360)))
            {
                PWM_DisableOutput(fan_setting->PWM_generator, fan_setting->PWM_RMSK);
                PWM_DisableOutput(fan_setting->PWM_generator, fan_setting->PWM_GMSK);
                PWM_DisableOutput(fan_setting->PWM_generator, fan_setting->PWM_BMSK);
            }
            fan_setting->fan_count++;
            fan_setting->fan_count = fan_setting->fan_count % (COUNT_ANGLE_360);
        }
        break;
```

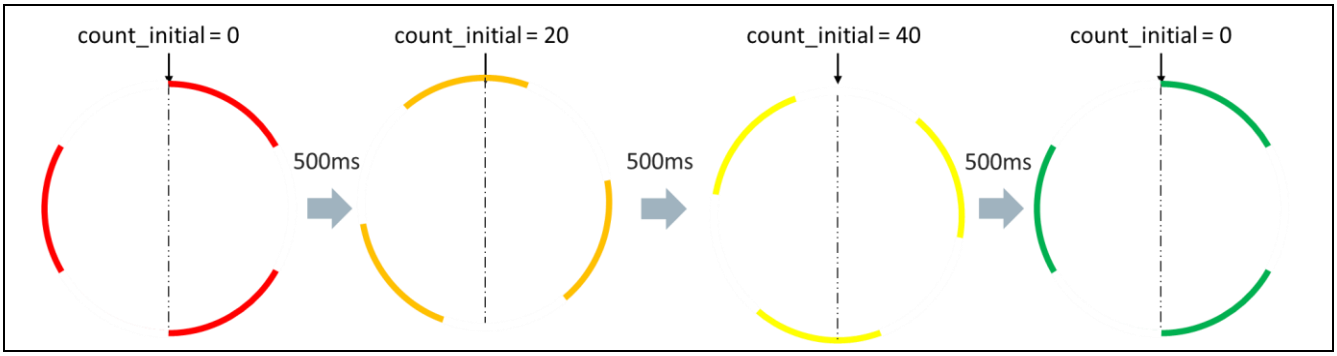


图 2-5 三爪灯效说明

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - M251_Series_BSP_CMSIS_V3.02.007
- IDE 版本
  - Keil uVersion 5.39

3.2 硬件需求

- 电路组件
  - NuMaker-M251SD V1.1
- 其他
  - 三个带有 LED 灯的风扇
- 线路连接

  - 将 UART0 TX (PB.13) pin 脚连接到 PC UART RX、 UART0 RX (PB.12) pin 脚连接到 PC UART TX，并开启 Nu-Link2-Me 上的 TXD 和 RXD switch 以显示范例代码的执行结果。
  - 一个风扇有数条线，每一个风扇的脚位都需要接到指定脚位，风扇数量有限的话，也可以接小于 3 个风扇。

	Fan 1 pin	Fan 2 pin	Fan 3 pin
TACH(FG)	PA.11	PA.10	PA.9
PWM(QSPI MOSI)	PA.0	PA.0	PA.0
LED_R	PB.5	PB.2	PC.2
LED_G	PB.4	PB.1	PC.1
LED_B	PB.3	PB.0	PC.0

- TACH 需要上拉电阻以让风扇能正确收到 Capture 信号

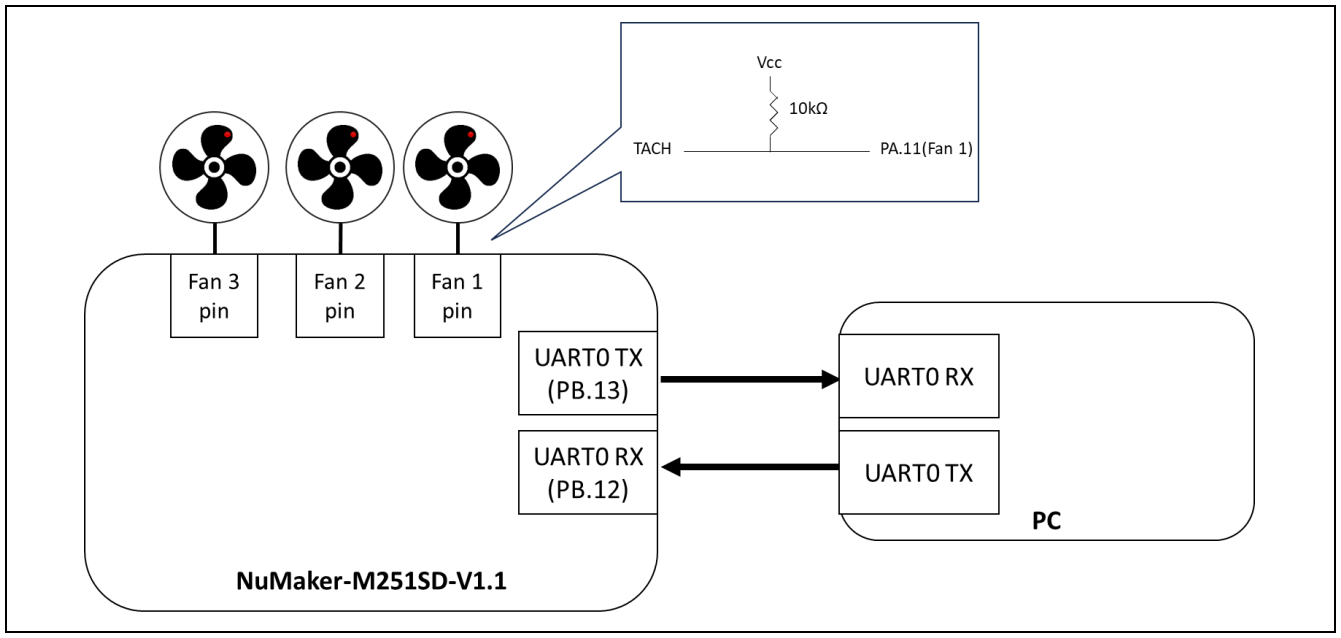


图 3-1 线路示意图

4. 目录信息

目录结构如图 4-1 所示。

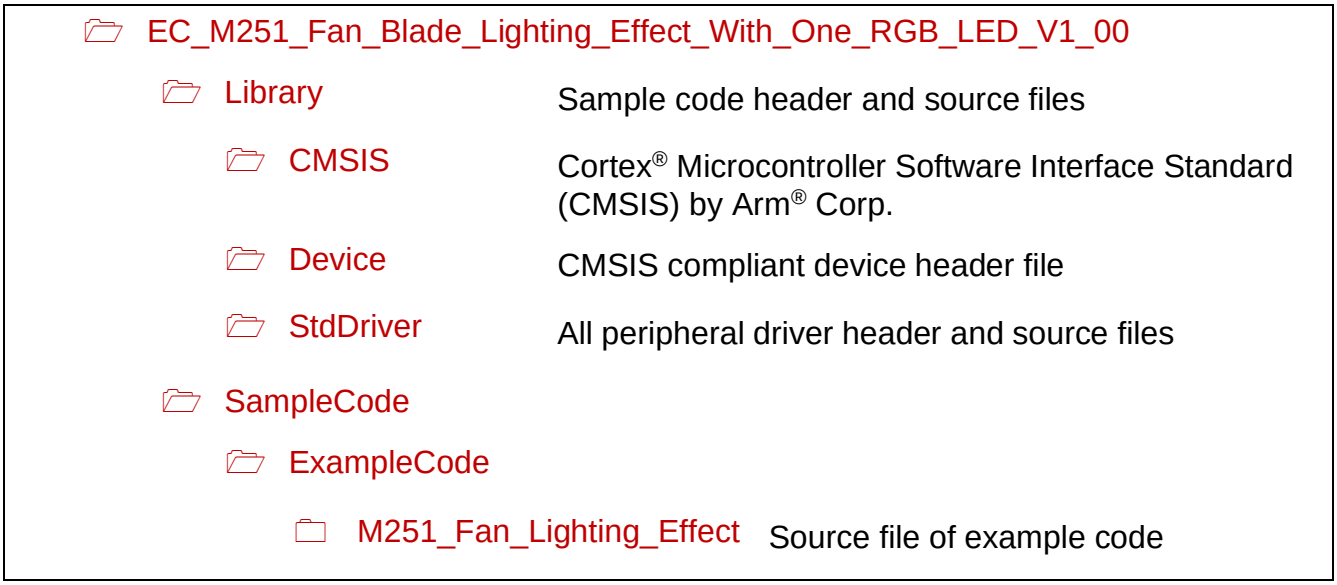


图 4-1 目录信息

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 *M251_Fan_Lighting_Effect.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
2025.07.01	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.