使用 PSIO 实现 ARGB2 LED 灯效

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 M252 PSIO 实现 ARGB2 LED 时序并实现彩虹灯、跑马灯等灯效
BSP 版本	M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.02.001
开发平台	NuMaker-M252KG Ver1.0

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

PSIO (Programmable Serial I/O) 提供简易的方法达到各种串行传输的接收与传送，例如: HDQ、DMX512、1-wire、IR、PS/2、Microwire、Wiegand、LED 等。本文描述如何使用 PSIO 实现 ARGB2 LED 灯效，并演示 ARGB2 LED 彩虹灯、跑马灯效果，此范例代码使用一组 Timer 来定时更新 ARGB2 LED 彩虹灯、跑马灯的颜色，一组 UART 来选择执行模式；其内容包含 PSIO 与 ARGB2 LED 的基本介绍、范例程序的说明以及运行结果。

1.1 原理

1.1.1 ARGB2 LED 数据传输时间 (TH + TL = 1.2 µs ± 600 ns)

ARGB2 LED 时序图如图 1-1 与图 1-2 所示。逻辑 '0' 定义高电平 0.3 µs ± 0.15 µs 与低电平 0.9 µs ± 0.15 µs，逻辑 '1' 定义为高电平 0.9 µs ± 0.15 µs 与低电平 0.3 µs ± 0.15 µs。

在使用 PSIO 控制 ARGB2 LED 灯条，必须先设定好时序规则(slot)，按照预期的时序运作，将时间基本单位划分约 0.3 µs 可以造出所需的波形。时序计算公式如下：

- PSIO clock source 选择 PCLK1，PCLK1 clock 计算方式 HIRC (48 MHz) / 4 = 12 MHz
- PSIO clock 使用 PCLK1 除频 4 = 3MHz
- PSIO one slot cycle time = 1 / 3MHz = 0.33 µs

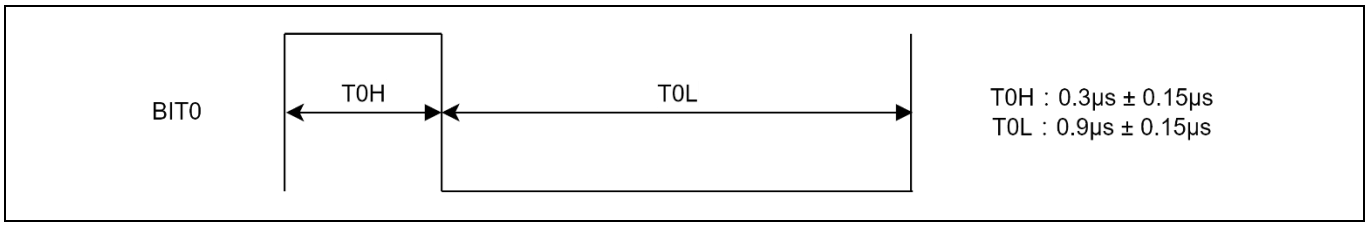

图 1-1 逻辑 '0' (bit 0)

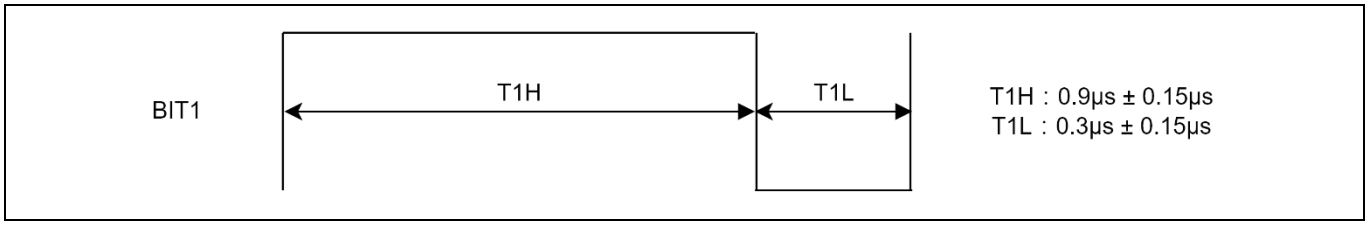

图 1-2 逻辑 '1' (bit 1)

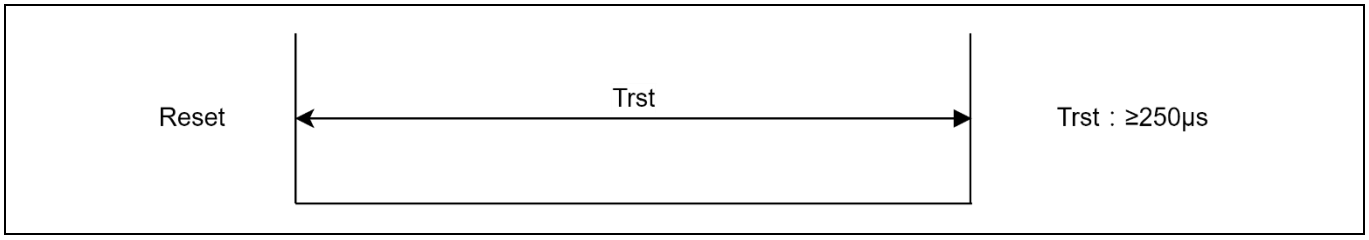

图 1- 3 复位 (Trst)

1.2 高级功能模式

1.2.1 Feedback 模式

ARGB2 LED 灯条除了能透过 MCU 传递信号做灯效，也可以回传信息给 MCU，内容有两种：

- 回传灯条上 LED 串联颗数与目前灯条被设定的 ID 号
- 回传灯条上 LED 串联颗数与 LED 消耗电流大小

Feedback 模式指令波形如图 1-4，下达 Feedback 模式指令给 ARGB2 LED 灯条后，ARGB2 LED 灯条会回传 High Pulse。PSIO Control PIN 由 Output 切换成 Input 读取 Pulse，藉由读取 Pulse 的数目及宽度便可得到该灯条上有多少颗 LED 及所使用的最大电流、目前灯条被设定的 ID。若 160 μs 内没有收到 High Pulse，MCU 要恢复到 Normal Mode，即 PSIO Control PIN 由 Input 切回 Output。

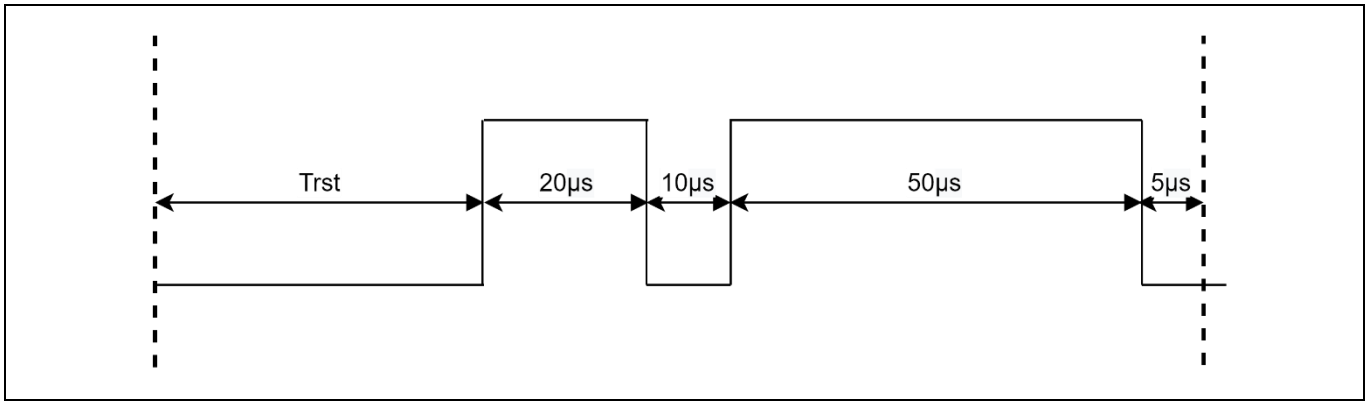

图 1- 4 进入 Feedback 模式指令波形

Pulse 宽度代表意义：

- 收到 20 μs High Pulse：LED 耗电流为 12mA
- 收到 10 μs High Pulse：LED 耗电流为 5mA

1.2.2 TH20 SET 模式

由 MCU 传送 20 µs (高电平) + 250 µs (低电平) 信号至 ARGB2 LED 灯条即可进入 TH20 SET 模式，后面再接续 24 bits，如图 1-5。接续在 TH20 SET 模式后是 24 bits 设定数据，可参考表 1-1、表 1-2。

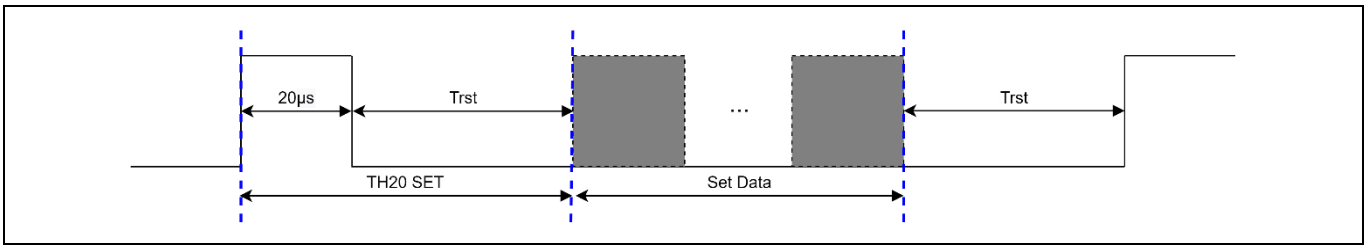

图 1-5 TH20 SET 模式波形

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SS <5>	SS <4>	0	G <4>	G <3>	G <2>	G <1>	G <0>	SS <3>	SS <2>	SS <6>	R <4>	R <3>	R <2>	R <1>	R <0>	SS <1>	SS <0>	0	B <4>	B <3>	B <2>	B <1>	B <0>

表 1-1 TH20 SET 模式 24 bits 数据格式

SS[0]	设置接收新资料后，PWM 是否重新计数，default：0 0：不重新计数 1：PWM 一律会由 0 开始计数
SS[2:1]	Imax (请勿更动)，default：00
SS[3]	为数据更新时机设置，default：0 0：接收到 250 µs Trst 信号才同步更新 1：收满 24 bits 立即更新
SS[5:4]	设置 PWM 频率，default：11 00：1.25 kHz 01：2.5 kHz 10：10 kHz 11：20 kHz

SS[6]	设置 Feedback 模式回传内容，default：0 0：回传 LED 颗数与 LED 消耗电流大小 1：回传 LED 颗数与目前灯条被设定的 ID 号
G[4:0]	G LED Dimming 功能，共 31 阶，default：0x1F
R[4:0]	R LED Dimming 功能，共 31 阶，default：0x1F
B[4:0]	B LED Dimming 功能，共 31 阶，default：0x1F

表 1-2 TH20 SET 模式设置功能说明

Dimming 功能可把原最大电流设置更细致，达到最佳灯效。控制输出电流公式如下，以 G LED 为例：

$$Iout = \frac{\mathrm{Im} \times (15.5 + 0.5 \times \mathrm{G}[0] + 1 \times \mathrm{G}[1] + 2 \times \mathrm{G}[2] + 4 \times \mathrm{G}[3] + 8 \times \mathrm{G}[4])}{31}$$

1.2.3 并联模式 Y Cable 功能

透过 Y Cable 功能可以对 ARGB2 LED 灯条做 ID 设定。开启 Y Cable 功能如图 1-6，进入前必需有一个 250 μs (低电平) Pulse 的 Reset 信号。并联模式 Y Cable 功能的 Command 有四种，可参考表 1-3。

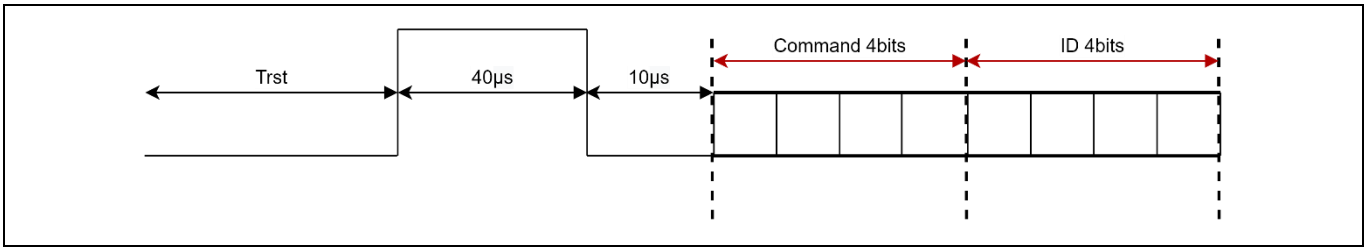

图 1-6 开启 Y Cable 指令波形

Command [7：4]	0001：设置灯条 ID 号 (SET_ID) 0010：清除灯条 ID 号 (CLR_ID) 0011：确认 ID 号是否存在或重复 (CHK_ID) 0100：指定使用 ID 号灯条 (Specify_ID)
ID [3：0]	范围为：0000 ~ 1111

表 1-3 并联模式 Y Cable 指令设定说明

1.2.3.1 设置 ARGB2 LED 灯条 ID (SET_ID)

SET_ID + ID 指令传送完毕后，如设置 ID 成功，ARGB2 LED 灯条会回传 High Pulse，PSIO 需要立刻由 Output 切换成 Input，用以判断 ID 设定情况。依此方法将所有并联的 ARGB2 LED 灯条给予 ID 号，当所有并联灯条都有 ID 号后再下 SET_ID，ARGB2 LED 灯条就不会回复 High Pulse。[1]

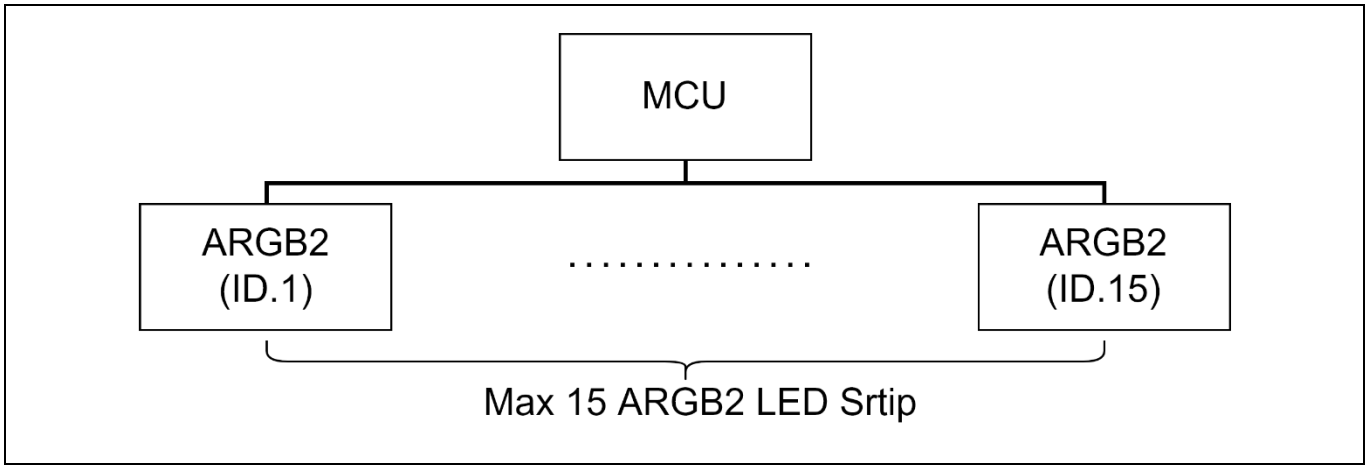

图 1- 7 Set ID 示意图

1.2.3.2 清除 ARGB2 LED 灯条 ID (CLR_ID)

当传送的 ID = 0000 时可以清除所有 ARGB2 LED 灯条 ID，当 ID = 0001 ~ 1111 时可以清除指定 ID 为 0。设置 ID 前建议做一次 CLR_ID + 0000，确保所有并联 ARGB2 LED 灯条 ID 皆为 0。

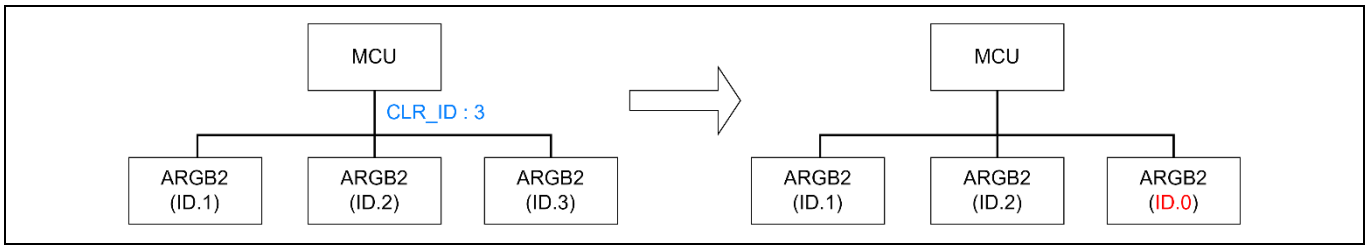

图 1- 8 CLR_ID 示意图

[1] 每次 Power ON 后，ARGB2 灯条的 ID 都会变成 0

1.2.3.3 确认 ARGB2 LED 灯条 ID (CHK_ID)

MCU 传送 CHK_ID 指令后，如有符合指定的 ID，ARGB2 LED 灯条会回传 High Pulse，假如并联的 ARGB2 LED 灯条有 ID 被重复设定，此时发送 CHK_ID 指令，MCU 会收到被重复设定数量的 High Pulse。

1.2.3.4 指定使用 ID 号灯条 (Specify_ID)

只有被指定到的 ID 灯条会有动作，当并联的 ARGB2 LED 灯条 ID 为 0 时则会一起动作。波形示意图，如图 1-9。

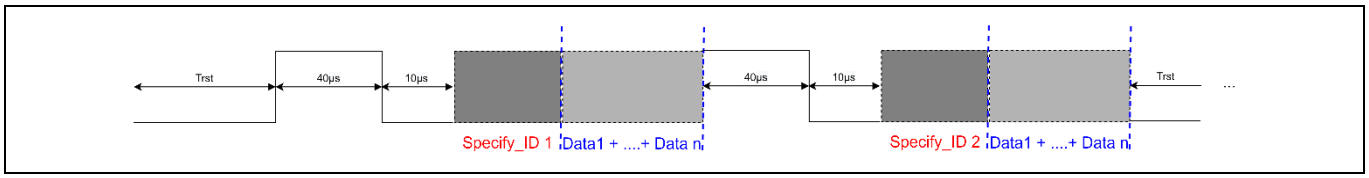

图 1- 9 Specify_ID 示意图

并联模式 Y Cable 功能控制流程如下：

- CLR_ID：把所有并联的 ARGB2 LED 灯条 ID 设定为 0
- SET_ID：分配不同 ID 给所有 ARGB2 LED 灯条，分配 ID 成功后，ARGB2 LED 灯条会回传 Pulse
- CHK_ID：对分配到 ID 的 ARGB2 LED 灯条分别做确认，如分配成功 ARGB2 LED 灯条会回传 Pulse
- Specify_ID：可以指定灯条做控制

1.3 执行结果

使用串口调试助手透过 UART 选择执行模式，波特率设定 115200，串口调试助手执行结果如图 1-10，可选择模式如下，其中 1 ~ 3 为 ARGB2 LED 灯效显示，4 ~ 7 为开启灯条不同模式：

1. 轮流执行显示绿灯、红灯、蓝灯、灯效
2. 执行彩虹灯效果
3. 执行跑马灯效果 (以彩虹方式)
4. 执行 Feedback 模式
5. 执行 TH20 SET 模式
6. 执行 Y Cable 模式
7. 设置 RGB 显示亮度

```
================================================
        Enter the select test function:
================================================
[01] RGB Loop
[02] Rainbow cycle
[03] Theater chase rainbow
[04] Feedback mode
[05] Th20 set mode
[06] Y cable mode
[07] Set RGB Brightness
```

图 1- 10 串口调试助手显示画面

1.3.2 逻辑 ’1’ (bit 1)、逻辑 ’0’ (bit 0)、复位 (Trst) 执行结果

图 1- 11 逻辑’1’和逻辑’0’执行结果

图 1- 12 复位 (Trst) 执行结果

1.3.3 GRB 循环执行结果

图 1- 13 绿灯执行结果

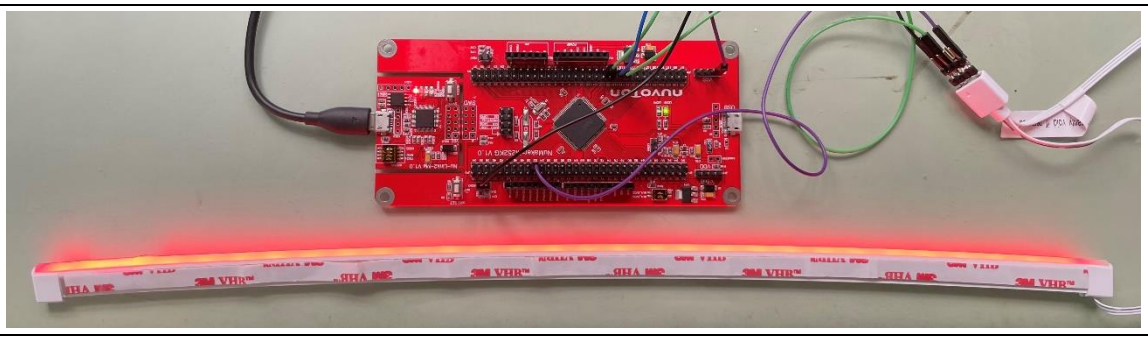

图 1- 14 红灯执行结果

图 1- 15 蓝灯执行结果

1.3.4 彩虹灯执行结果

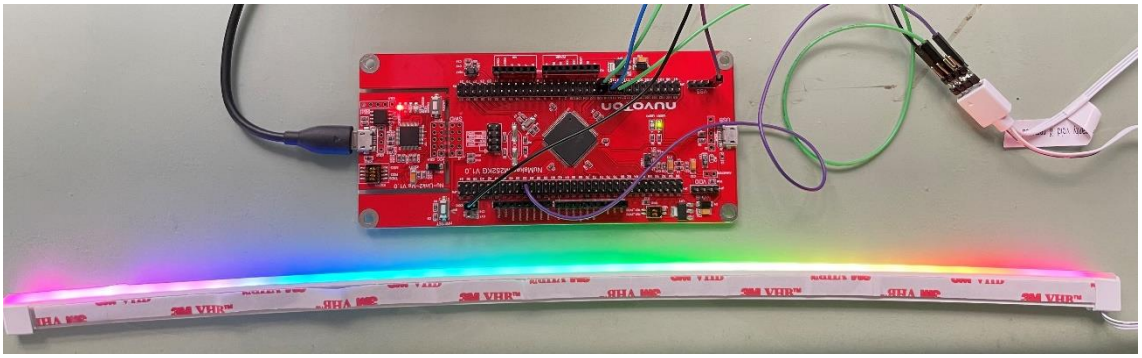

图 1- 16 彩虹灯执行结果

1.3.5 跑马灯执行结果

图 1- 17 跑马灯执行结果

1.3.6 Feedback 模式执行结果

ARGB2 规范进入 Feedback 的时序需要 20 µs (高电平) + 10 µs (低电平) + 50 µs (高电平)。如图 1-18 所示。

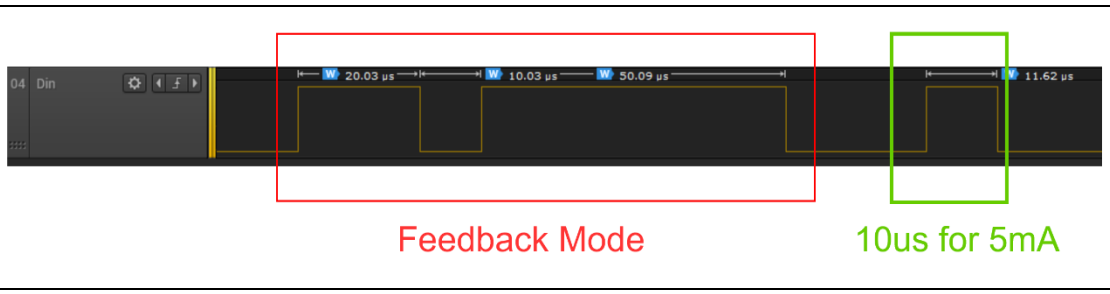

图 1- 18 Feedback 模式波形和 Pulse 宽度代表最大耗电流

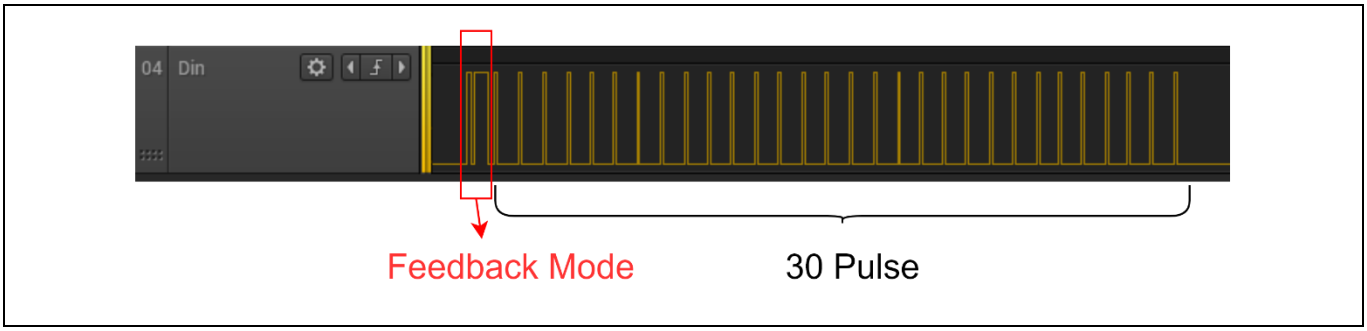

图 1- 19 ARGB2 LED 灯条回传 Pulse

1.3.7 TH20 Set 模式执行结果

由 MCU 传送 20 µs (高电平) + 250 µs (低电平) 信号至 ARGB2 LED 灯条即可进入 TH20 SET 模式，后面再接续 24 bits 数据。

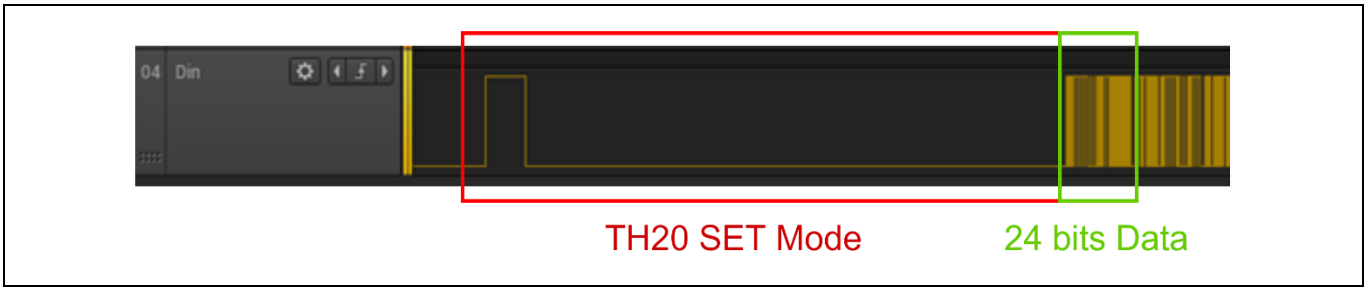

图 1- 20 开启 TH20 模式模式波形

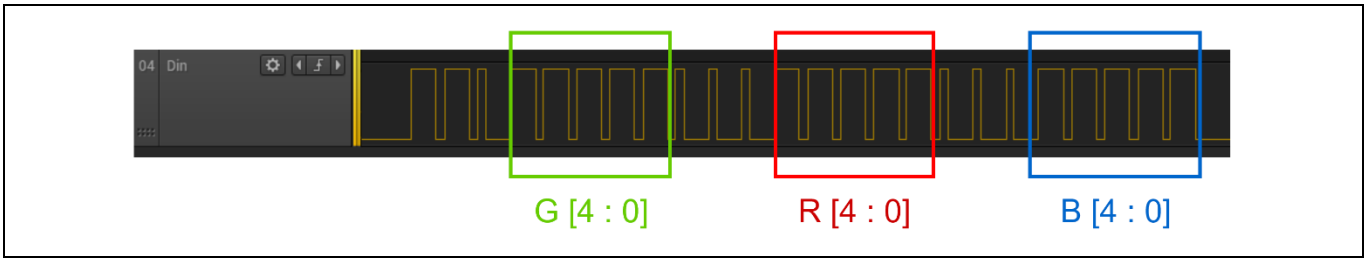

图 1- 21 TH20 设置 24 bits 资料波形

1.3.8 并联模式 Y Cable 执行结果

ARGB2 规范支持多灯条并联控制模式 (最多 15 条) 。使用指定的命令，可以识别每个灯条分配的唯一的灯条动态 ID (通过 set dynamic ID 命令) 。命令完成后，MCU 可以单独控制并将 " 清除 ID " 、 " 检查 ID " 、 " 指定 ID " 命令发送到每个灯条。

MCU 设置输出时序 40 µs + 10 µs + 8 bits (Command + ID) 进入 Y Cable 模式，波形如图 1-22。

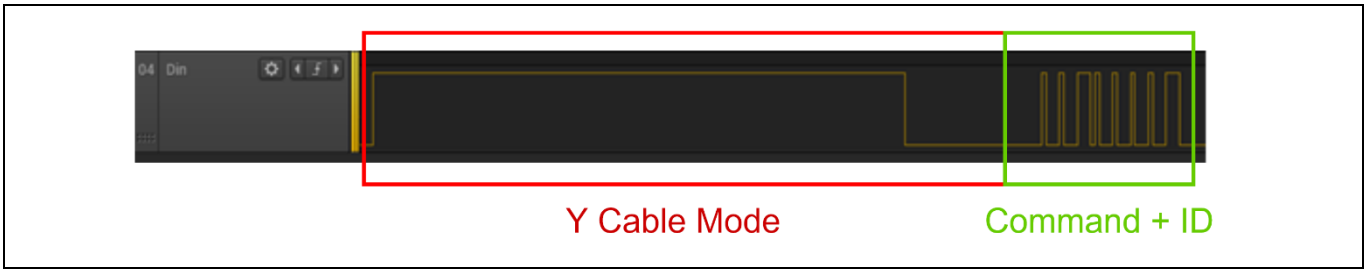

图 1- 22 开启 Y Cable 功能和 8 bits (Command + ID) 波形

- 设置 ARGB2 灯条 ID (SET_ID) 执行结果

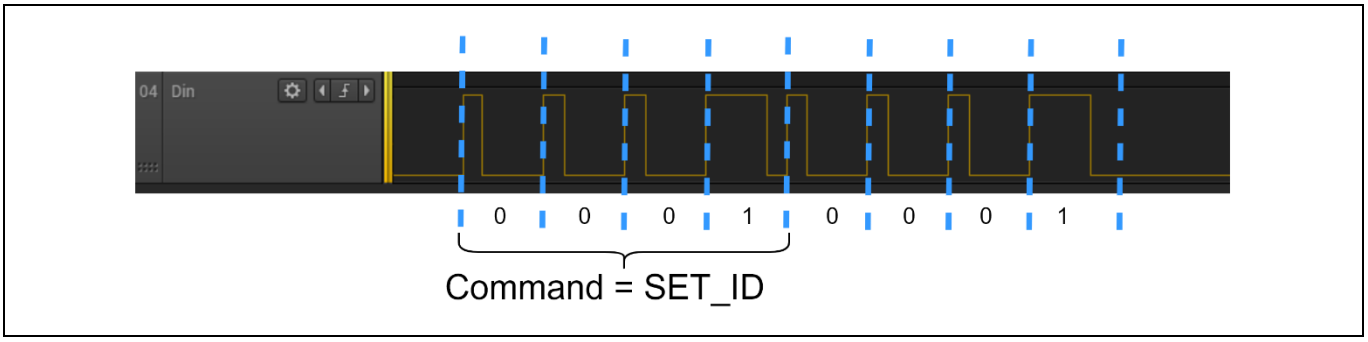

图 1- 23 设置 ARGB2 灯条 (SET_ID) 波形

- 清除 ARGB2 灯条 ID (CLR_ID) 执行结果

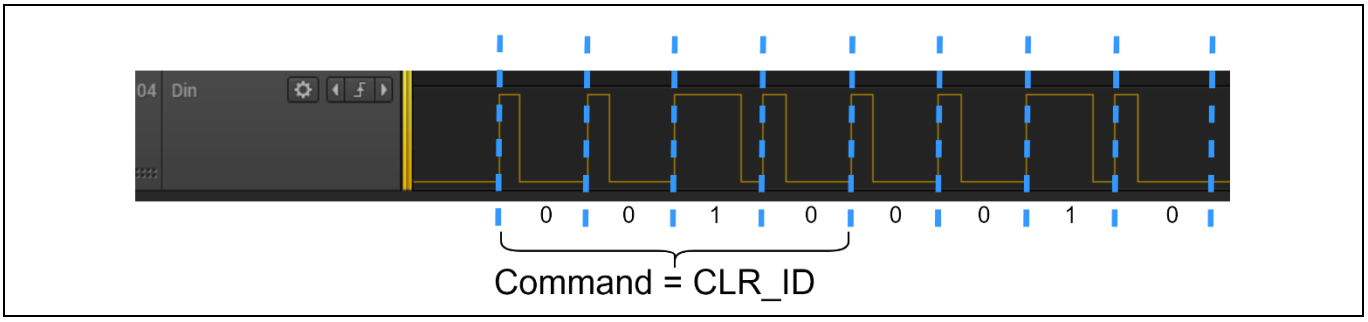

图 1- 24 清除 ARGB2 灯条 (CLR_ID) 波形

- 确认 ARGB2 灯条 ID (CHK_ID) 执行结果

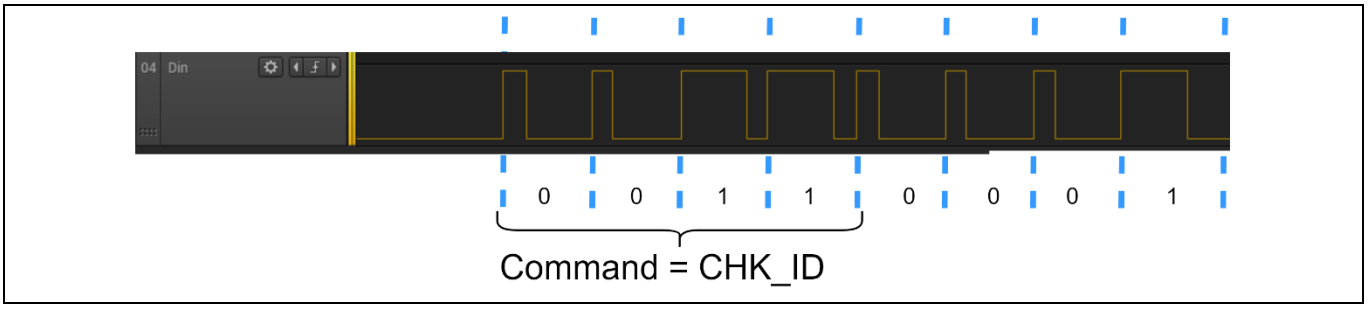

图 1- 25 确认 ARGB2 灯条 (CHK_ID) 波形

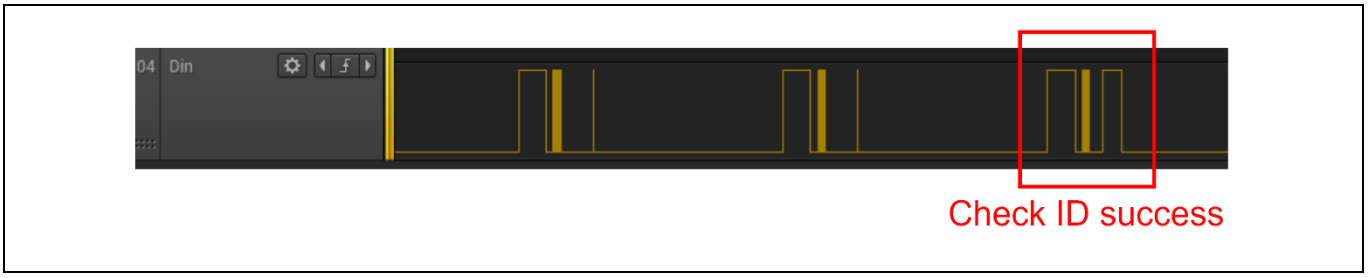

图 1- 26 确认 ARGB2 灯条 ID 成功示意图

- 指定使用 ID 号码 ARGB2 灯条 (Specify_ID) 执行结果

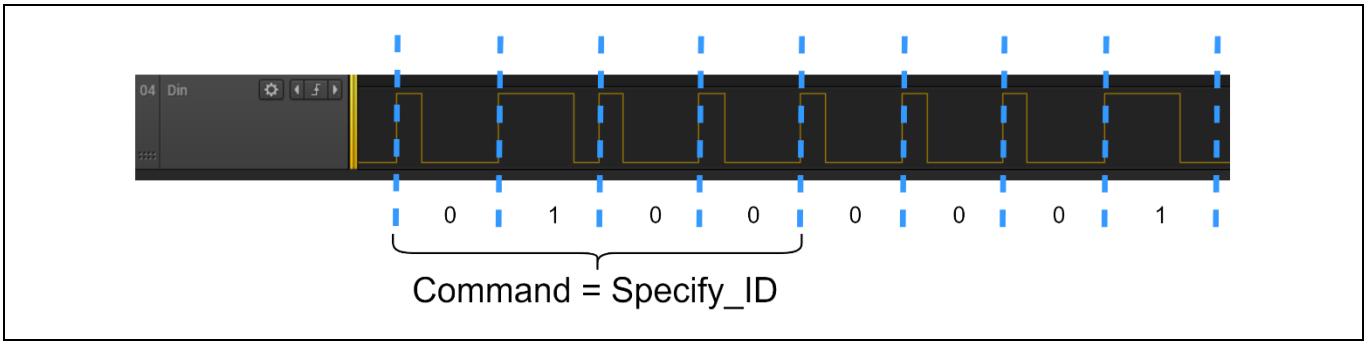

图 1- 27 指定使用 ID 号灯条 (Specify_ID)

2. 代码介绍

2.1 主代码介绍

代码工作流程如下:

1. *main.c* 中初始化 M252 系列相关外设，包含 HIRC、PSIO、TIMER0、UART0
2. ARGB2StripTestMenu() function 中 while() 循环选择执行 GRB 循环、彩虹灯、跑马灯、Feedback 模式、TH20 SET 模式、并联 Y Cable 模式功能
3. 逻辑 ’0’、逻辑 ’1’ 波形使用 PSIO_SendDataWithSize(uint32_t u32DataSize, uint32_t u32Data) 实现

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

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

    /* Init UART0 for printf and test */
    UART0_Init();

    /* Lock protected registers */
    SYS_LockReg();

    Delay_Init();

    /* Use slot control 0 and PIN 0 */
    PSIO_StripInit(PSIO_SC0, PSIO_PIN0);

    PSIO_StripSetBrightness(255);

    ARGB2StripTestMenu();
}
```

main.c PSIO_StripInit() 初始化 PSIO Slot Control 和 PSIO PIN，请参考如下代码。

```
void PSIO_StripInit(uint8_t u8SlotCtrl, uint8_t u8DataPin)
{
    gsPSIO_ARGB2_Cfg.u8SlotCtrl = u8SlotCtrl;
    gsPSIO_ARGB2_Cfg.u8DataPin = u8DataPin;

    PSIO_Strip_OpenOutput(PSIO_PIN_ENABLE);
}
```

PSIS_ARGB2_Driver.c PSIO_Strip_OpenOutput() 在初始化 Output Mode 时，使用 PSIO_SET_GENCTL() 设定 Pin 的状态，内容为使用的 Pin、Slot controller、模式、初始化状态及结束状态等。PSIO_SET_WIDTH() 与 PSIO_SET_OUTPUT_DEPTH() 则是与数据的格式有关，此范例初始设定输出宽度为 8 bit、输入宽度为 1 bit，深度皆为 1。

```
static void PSIO_Strip_OpenOutput(uint32_t u32Switch)
{
    /* PSIO UART TX PIN general setting */
    PSIO_SET_GENCTL(PSIO,
                    gsPSIO_ARGB2_Cfg.u8DataPin,
                    u32Switch,
                    gsPSIO_ARGB2_Cfg.u8SlotCtrl,
                    PSIO_OUTPUT_MODE,
                    PSIO_LOW_LEVEL,
                    PSIO_LOW_LEVEL);

    /* Set data order ad MSB */
    PSIO_SET_ORDER(PSIO, gsPSIO_ARGB2_Cfg.u8DataPin, PSIO_MSB);

    /* Set output data buffer width as 8 bit, input data width as 1 bit*/
    PSIO_SET_WIDTH(PSIO, gsPSIO_ARGB2_Cfg.u8DataPin, 1, 8);

    /* Set output data buffer depth as 1 */
    PSIO_SET_OUTPUT_DEPTH(PSIO, gsPSIO_ARGB2_Cfg.u8DataPin, PSIO_DEPTH1);

    /* Set slot controller trigger source as software trigger */
    PSIO_SET_TRIGSRC(PSIO, gsPSIO_ARGB2_Cfg.u8SlotCtrl, PSIO_SW_TRIGGER);
}
```

main.c while() 循环实现选择功能执行，请参考如下代码。

```
void ARGB2StripTestMenu(void)
{
    uint8_t u8Index = 0;
    struct
    {
        char *chName;
        int (*funcptr)(void);
    } sMainMenu[] =
    {
        {"", NULL},
        {"RGB Loop", &Strip_RGBLoop},
        {"Rainbow cycle", &Strip_RainbowCycle},
        {"Theater chase rainbow", &Strip_TheaterChaseRainbow},
        {"Feedback mode", &Strip_FeedbackMode},
        {"Th20 set mode", &Strip_Th20SetMode},
        {"Y cable mode", &Strip_YCableSetupMenu},
        {"Set RGB Brightness", &Strip_SetBrightness},
    };

    while (1)
    {
        printf("\r\n\r\n");
```

```
        printf("============================================\r\n");
        printf("\tEnter the select test function: \r\n");
        printf("============================================\r\n");

        for (u8Index = 1; u8Index < sizeof(sMainMenu) / sizeof(sMainMenu[0]); u8Index++)
        {
            printf("[%02d] %s\r\n",
                   u8Index,
                   sMainMenu[u8Index].chName);
        }

        printf("\r\n");

        u8Index = sysGetNum();

        if ((u8Index == 0) || (u8Index >= sizeof(sMainMenu) / sizeof(sMainMenu[0])))
        {
            continue;
        }

        if (sMainMenu[u8Index].funcptr != NULL)
        {
            sMainMenu[u8Index].funcptr();
        }
    }
}
```

輸出邏輯 ‘0’、邏輯 ‘1’ 波形信号函数需要三种不同的动作，分别是输出高电平、从缓冲区输出数据及输出低电平，时序的规划如图 1-1 和图 1-2 所述，并整理于表 1-1。Slot 0 ~ Slot 2 可视为 ARGB2 LED 协议的 1 bit。PSIO_SET_WIDTH() 根据输出的数据大小设定输出宽度，例如设定 ARGB2 LED 的颜色需要设定输出宽度为 24 bit。PSIO_SET_SCCTL() 利用 PSIO 的循环功能，将此行为再复制 23 次，即输出 24 bit 数据。设定完成后，用软件触发 PSIO_START_SC() 的方式，让 slot controller 开始执行。

	Slot0	Slot1	Slot2
Check Point	Check Point 0	Check Point 1	Check Point 2
Engine Clock	1	2	1
PIN Action	Output High	Output Data Form Buffer	Output Low

表 2-1 PSIO 设定输出逻辑’0’、逻辑’1’

```
void PSIO_SendDataWithSize(uint32_t u32DataSize, uint32_t u32Data)
{
    S_PSIO_CP_CONFIG sDataConfig = {0};

    /* Slot 0 ~ 7 */
    sDataConfig.CKPT0SLT = PSIO_SLOT0;
    sDataConfig.CKPT1SLT = PSIO_SLOT1;
    sDataConfig.CKPT2SLT = PSIO_SLOT2;
```

```
    sDataConfig.CKPT3SLT = PSIO_SLOT_DISABLE;
    sDataConfig.CKPT4SLT = PSIO_SLOT_DISABLE;
    sDataConfig.CKPT5SLT = PSIO_SLOT_DISABLE;
    sDataConfig.CKPT6SLT = PSIO_SLOT_DISABLE;
    sDataConfig.CKPT7SLT = PSIO_SLOT_DISABLE;

    /* Actions 0 ~ 7 */
    sDataConfig.CKPT0ACT = PSIO_OUT_HIGH;
    sDataConfig.CKPT1ACT = PSIO_OUT_BUFFER;
    sDataConfig.CKPT2ACT = PSIO_OUT_LOW;
    sDataConfig.CKPT3ACT = PSIO_NO_ACTION;
    sDataConfig.CKPT4ACT = PSIO_NO_ACTION;
    sDataConfig.CKPT5ACT = PSIO_NO_ACTION;
    sDataConfig.CKPT6ACT = PSIO_NO_ACTION;
    sDataConfig.CKPT7ACT = PSIO_NO_ACTION;

    /* Set check point configuration */
    PSIO_SET_CP_CONFIG(PSIO, gsPSIO_ARGB2_Cfg.u8DataPin, &sDataConfig);

    /* Set slot count */
    PSIO_SCSLOT_SET_SLOT(PSIO, gsPSIO_ARGB2_Cfg.u8SlotCtrl, PSIO_SLOT0, 1);
    PSIO_SCSLOT_SET_SLOT(PSIO, gsPSIO_ARGB2_Cfg.u8SlotCtrl, PSIO_SLOT1, 2);
    PSIO_SCSLOT_SET_SLOT(PSIO, gsPSIO_ARGB2_Cfg.u8SlotCtrl, PSIO_SLOT2, 1);

    /* Enable repeat slot0 ~ slot2 u32DataSize - 1 times */
    PSIO_SET_SCCTL(PSIO, gsPSIO_ARGB2_Cfg.u8SlotCtrl, PSIO_SLOT0, PSIO_SLOT2, u32DataSize
- 1, PSIO_REPEAT_DISABLE);

    /* Set out data width as u32DataSize bit */
    PSIO_SET_WIDTH(PSIO, gsPSIO_ARGB2_Cfg.u8DataPin, 0, u32DataSize);

    /* Set out data depth as 1 */
    PSIO_SET_OUTPUT_DEPTH(PSIO, gsPSIO_ARGB2_Cfg.u8DataPin, PSIO_DEPTH1);

    /* Set output data to buffer */
    PSIO_SET_OUTPUT_DATA(PSIO, gsPSIO_ARGB2_Cfg.u8DataPin, u32Data);

    /* Trigger slot controller */
    PSIO_START_SC(PSIO, gsPSIO_ARGB2_Cfg.u8SlotCtrl);

    /* Wait for slot controller is not busy */
    while (PSIO_GET_BUSY_FLAG(PSIO, gsPSIO_ARGB2_Cfg.u8SlotCtrl));

    /* Stop slot controller */
    PSIO_STOP_SC(PSIO, gsPSIO_ARGB2_Cfg.u8SlotCtrl);

    /* Disable repeat slot0 ~ slot2 */
    PSIO_SET_SCCTL(PSIO, gsPSIO_ARGB2_Cfg.u8SlotCtrl, PSIO_SLOT_DISABLE,
PSIO_SLOT_DISABLE, 0, PSIO_REPEAT_DISABLE);
}
```

2.2 灯效代码说明

2.2.1 GRB 循环灯效

PSIO_StripSetRGBColor(uint8_t u8Red, uint8_t u8Green, uint8_t u8Blue) 用来组合单颗灯颜色的 24 bits 数据，可设定单颗灯单一颜色，PIXEL_NUM 代表要设定的 LED 灯数量，利用 Timer 函数 TIMER_Delay() 来调整 LED 灯显示的间隔时间。

```
int Strip_RGBLoop(void)
{
    uint8_t u8i = 0;

    while (kbhit())
    {
        for (u8i = 0; u8i < PIXEL_NUM; u8i++)
        {
            PSIO_StripSetRGBColor(0xFF, 0x00, 0x00);
        }

        TIMER_Delay(TIMER0, 100000);

        for (u8i = 0; u8i < PIXEL_NUM; u8i++)
        {
            PSIO_StripSetRGBColor(0x00, 0xFF, 0x00);
        }

        TIMER_Delay(TIMER0, 100000);

        for (u8i = 0; u8i < PIXEL_NUM; u8i++)
        {
            PSIO_StripSetRGBColor(0x00, 0x00, 0xFF);
        }

        TIMER_Delay(TIMER0, 100000);
    }

    PSIO_StripClear();

    return 0;
}
```

```
void PSIO_StripSetRGBColor(uint8_t u8Red, uint8_t u8Green, uint8_t u8Blue)
{
    int32_t i32Data = 0;

    u8Red   /= (256 / ((uint16_t) gsPSIO_ARGB2_Cfg.u8Brightness + 1));
    u8Green /= (256 / ((uint16_t) gsPSIO_ARGB2_Cfg.u8Brightness + 1));
    u8Blue  /= (256 / ((uint16_t) gsPSIO_ARGB2_Cfg.u8Brightness + 1));

    i32Data |= (u8Green << 16);
    i32Data |= (u8Red << 8);
```

```
    i32Data |= u8Blue;

    PSIO_SendDataWithSize(24, i32Data);
}
```

2.2.2 彩虹灯

PSIO_StripSetPixelColor(uint16_t n, uint32_t GRBColor) 函数用来存放每颗灯的颜色，PIXEL_NUM 为欲设定灯的数量， PSIO_StripWheel(uint8_t wheelPos) 函数用来设定颜色存放位置，用以产生轮转效果，PSIO_StripShow() 函数用来输出 BIT0、BIT1 波形至 ARGB2 灯条，以实现彩虹灯效果。

```
int Strip_RainbowCycle(void)
{
    uint16_t u16i = 0, u16j = 0;

    /* 5 cycles of all colors on wheel */
    for (u16j = 0; u16j < (256 * 5); u16j++)
    {
        for (u16i = 0; u16i < PIXEL_NUM; u16i++)
        {
            PSIO_StripSetPixelColor(u16i, PSIO_StripWheel(((u16i * 256 / PIXEL_NUM) +
u16j) & 255));
        }

        PSIO_StripShow();

        PSIO_StripReset();
    }

    PSIO_StripClear();

    return 0;
}
```

2.2.3 跑马灯

前半段代码与彩虹灯一样，PSIO_StripSetPixelColor(uint16_t n, uint32_t GRBColor) 函数用来存放每颗灯的颜色，PIXEL_NUM 为欲设定灯的数量，PSIO_StripWheel(uint8_t wheelPos) 函数用来设定颜色存放位置，用以产生轮转效果，PSIO_StripShow() 函数用来输出 BIT0、BIT1 波形至 ARGB2 灯条，最后再用一个 for 循环，透过 PSIO_StripSetPixelColor(uint16_t n, uint32_t GRBColor) 函数让其一段时间间隔无颜色，以实现跑马灯效果。

```
int Strip_TheaterChaseRainbow(void)
{
    uint16_t u16i = 0, u16j = 0;
```

```
    uint8_t u8q = 0;

    /* cycle all 256 colors in the wheel */
    for (u16j = 0; u16j < 256; u16j++)
    {
        for (u8q = 0; u8q < 3; u8q++)
        {
            for (u16i = 0; u16i < PIXEL_NUM; u16i = u16i + 3)
            {
                PSIO_StripSetPixelColor(u16i + u8q, PSIO_StripWheel((u16i + u16j) % 255));
            }

            PSIO_StripShow();

            PSIO_StripReset();

            for (u16i = 0; u16i < PIXEL_NUM; u16i = u16i + 3)
            {
                /* turn every third pixel off */
                PSIO_StripSetPixelColor(u16i + u8q, 0);
            }
        }
    }

    PSIO_StripClear();

    return 0;
}
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.02.001
- IDE 版本
  - Keil uVersion 5.28.0.0

3.2 硬件需求

- 电路组件
  - NuMaker-M252KG V1.0
  - ARGB2 LED 灯条
- 示意图
  - 使用 PSIO 脚位来模拟 ARGB2 LED 灯条时序图

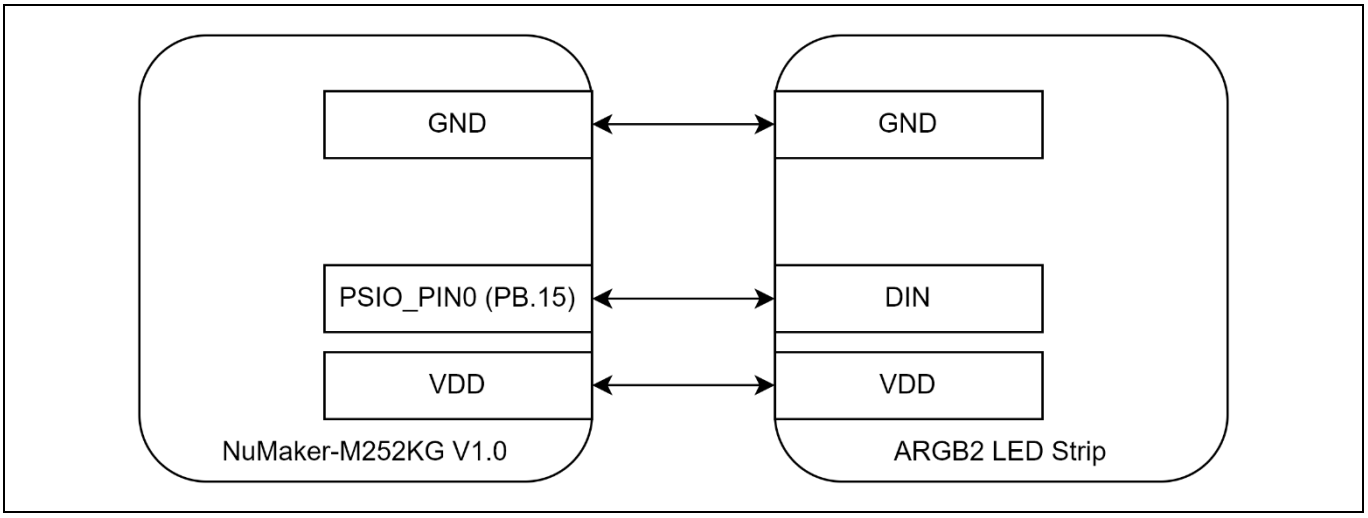

图 3- 1 M252 PSIO_PIN0 和 ARGB2 LED DIN 连接

- ARGB2 LED 灯条工作电压为 5V，故 NuMaker-M252KG 开发板为 ARGB2 LED 灯条供应电压必须更换为 5V。可以改变 Nu-Link2-Me 上的 ICE JPR1 (MCU VCC) 电阻使其输出 5V 电压，或是使用 USB to TTL module 的 5V PIN。

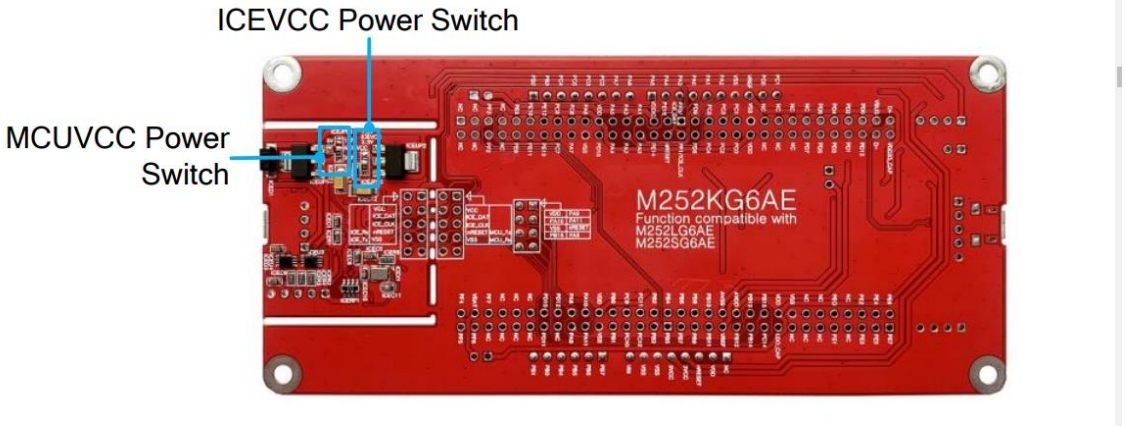

图 3- 2 MCU VCC 切换至 5V

4. 目录信息

EC_M252_PSIO_ARGB2_LED_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	Source file of example code

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 *M252_PSIO_ARGB2_LED_Control.uvprojx*。
2. 进入编译模式界面
   - 编译
   - 下载代码至内存
   - 进入 / 离开仿真模式
3. 进入仿真模式界面
   - 执行代码

6. 修订纪录

Date	Revision	Description
2022.06.02	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.