MS51 驱动 RGB 7 段 LED

NuMicro® 1T 8051 系列微控制器范例代码介绍

文件信息

应用简述	使用 MS51 32K 系列驱动 RGB 彩色 7 段 LED 显示
BSP 版本	MS51_Series_BSP_Keil_V1.00.006
开发平台	NuMaker-MS51PC V1.1 & PWM_ARGB_Control_Board V1.0

1. 概述

1.1 原理

传统 7 段显示为单一颜色，使用者只需把正极接电，不同的负极接地就能控制七段显示不同的数字。

RGB 是色光的色彩模式。R 代表红色，G 代表绿色，B 代表蓝色，三种色彩迭加形成了其它的色彩。因为三种颜色都有 256 个亮度水平级，所以三种色彩迭加就形成 1670 万种颜色。

所以当 RGB 应用于 7 段 LED 显示时，不仅需要点亮，还需要控制 RGB 三色不同的色阶，达成混合颜色的效果。

本范例使用 ADH-Tech 的 ACD8143 RGB LED 模块来实现，LED 脚位定义与内部驱动线路如下：共点亮 2 个 RGB LED 分别定义为 DIG.1、DIG.2 。

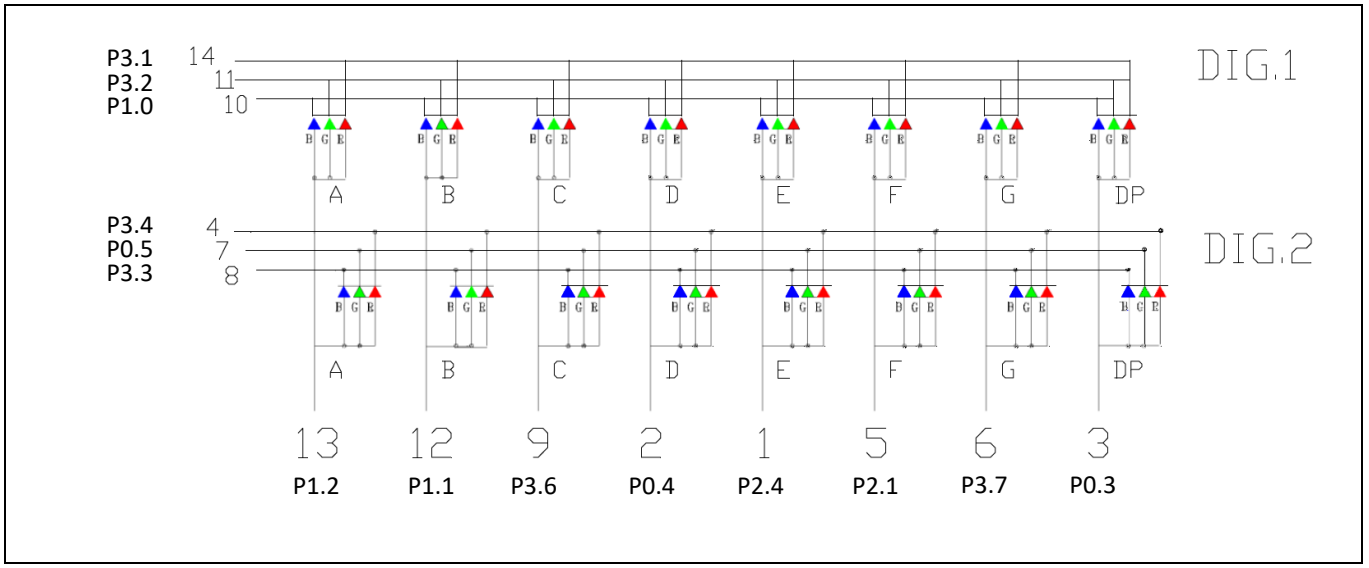

图 1-1 ACD8143 脚位定义与内部驱动线路

脚位 10\11\14 分别代表需要点亮 DIG.1 的 PWM 色阶 B\G\R 控制， 脚位 4\7\8 用于点亮 DIG.2， PWM 输出控制 255 级色阶。

其余脚位 13\12\9\2\1\5\63 直接以 GPIO 控制点亮，程序内定义名称为 SEG, 分别对应 SEG_A ~ SEG_G 以及 SEG_DP。

例如当需要点亮 DIG.1 的 A 段绿色时，设定 LED 11 脚 PWM 输出 Duty 定义色阶，13 脚 GPIO 定为 1 来进行 Enable SEG 配置。

依据上述定义，配合 NuMaker-MS51PC，设计控制板线路如下

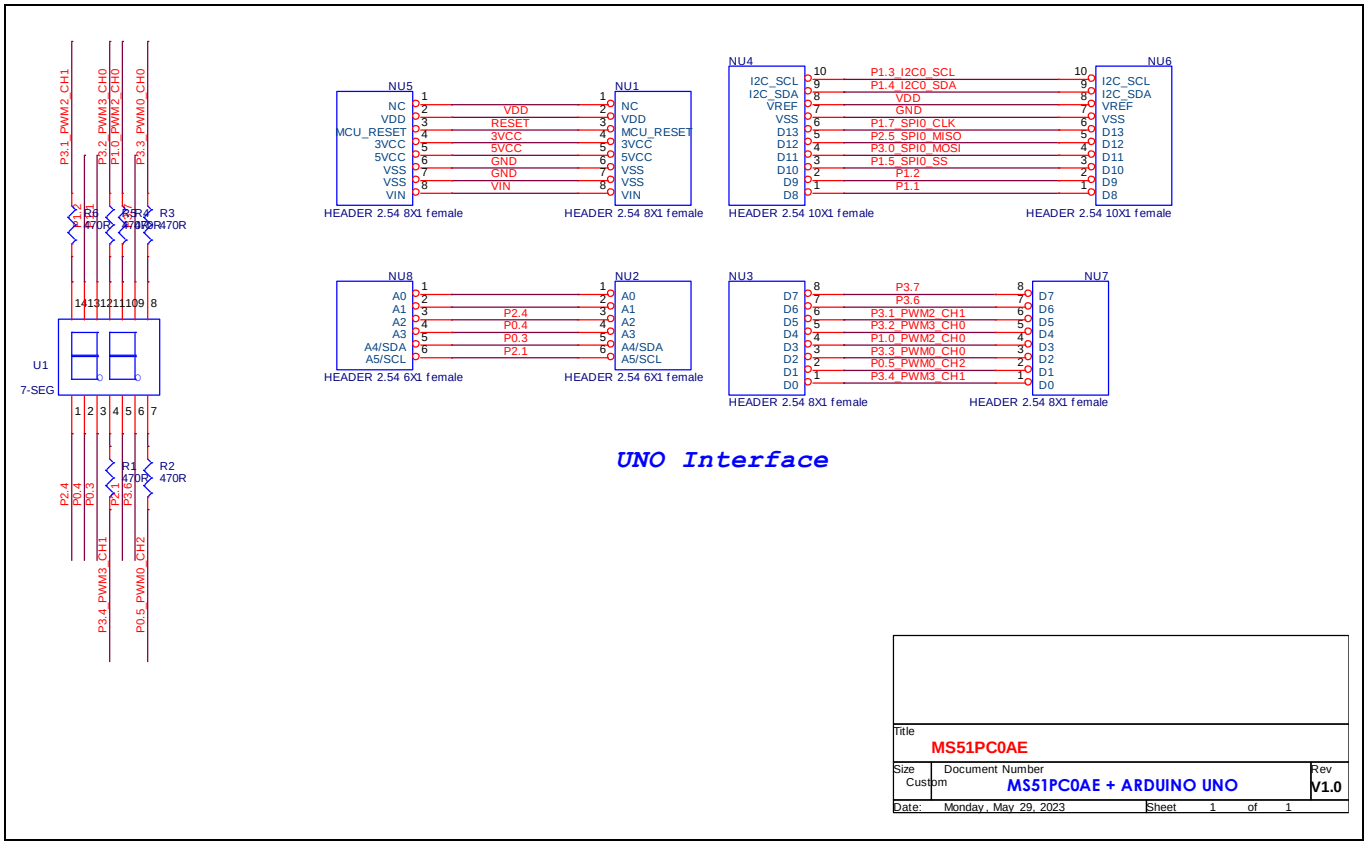

图 1-2 ACD8143 控制板原理图

1.2 执行结果

通过 Access Port UART 传输固定格式 RGB LED 显示对应显示的数字与颜色

UART 传输格式如下，每次传输定义一个 7 段显示

Target: 0 = DIG.1, 1 = DIG.2;

Digit = 0~9, A~F;

Dot Point: 0 = Disable, 1 = Enable;

Color R/G/B = 0x00 ~ 0xFF

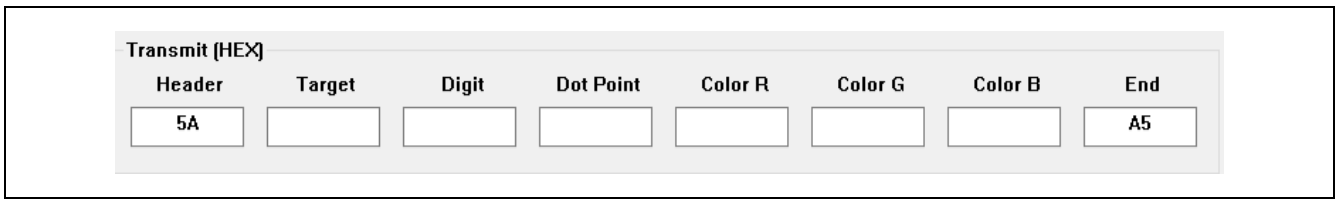

图 1-3 UART 传输格式定义

输入范例，

第一组输入 5A 00 03 00 69 AA 00 A5 控制 DIG.1 显示数字 [3] ，颜色为嫩绿色

第二组输入 5A 01 0A 00 FF AF 30 A5 控制 DIG.2 显示字符 [A] ，颜色为嫩黄色

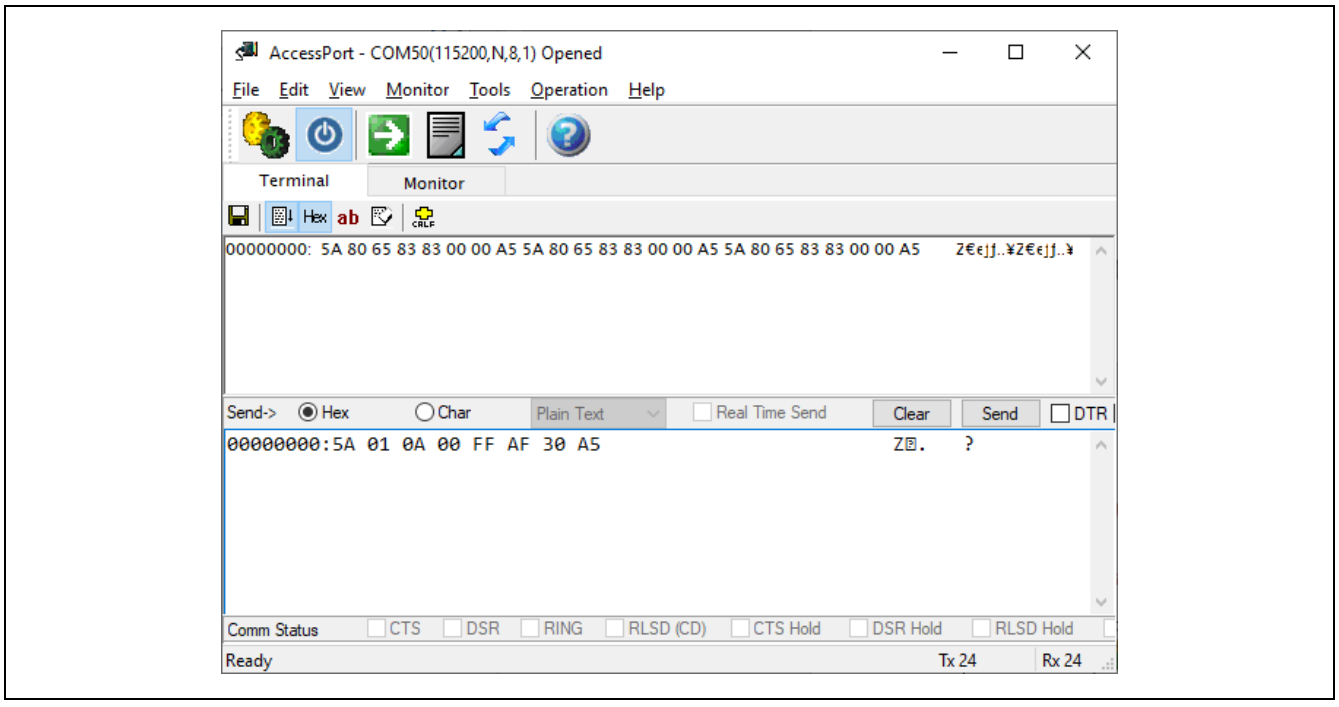

图 1-4 UART 输入与返回值

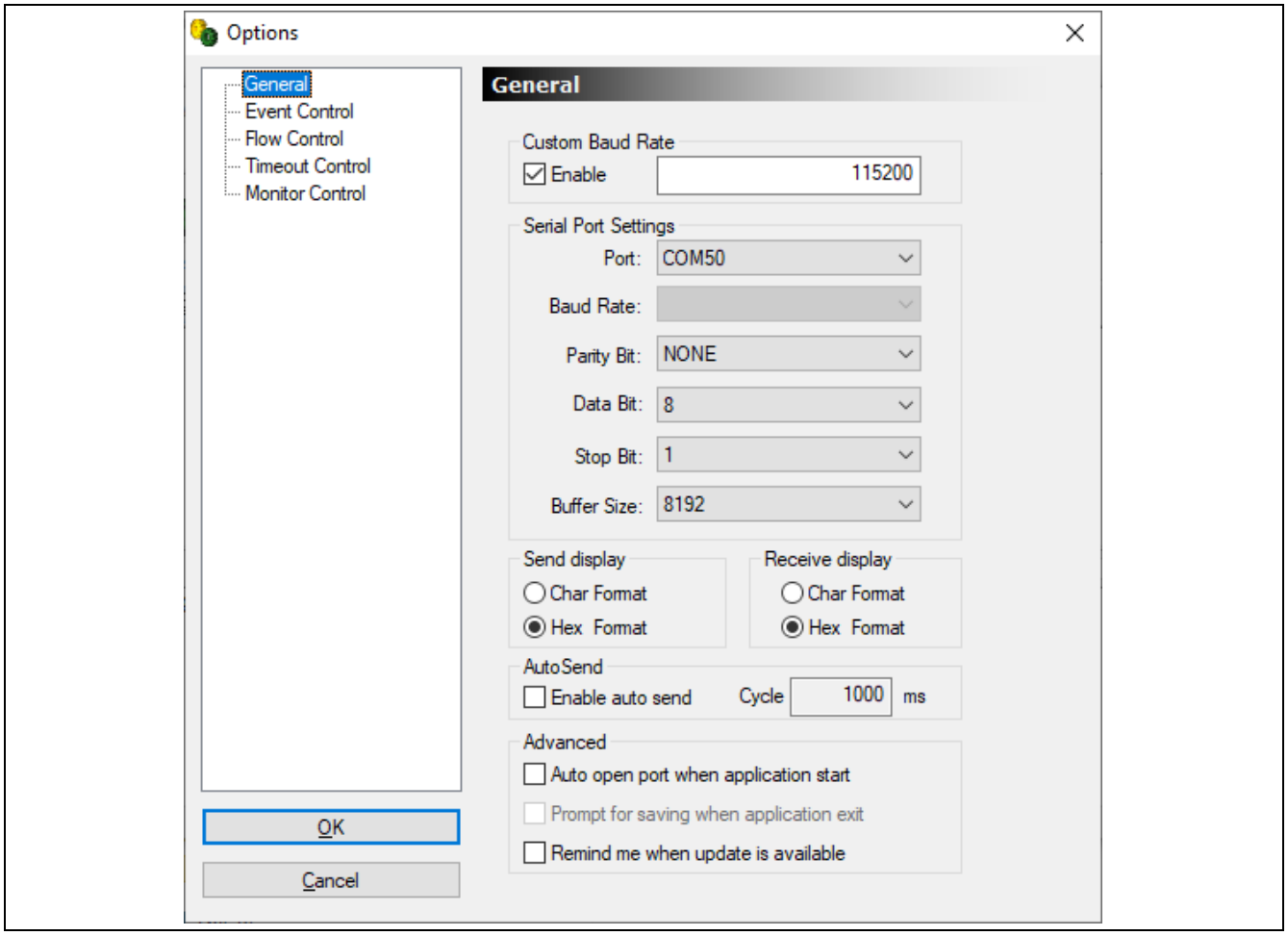

图 1-5 Access Port 配置示意图

2. 配置示意图代码介绍

LED 驱动点亮代码位于 *main.c*。

定义 7 段显示器不同显示字符的 SEG。SEG_DP 在每个 PWM 更新的周期判断是否需要点亮。

```
uint8_t Show_Char[16][7] =
{
    /* SEG_A, SEG_B, SEG_C, SEG_D, SEG_E, SEG_F, SEG_G */
    {      1,     1,     1,     1,     1,     1,     0},    // 0
    {      0,     1,     1,     0,     0,     0,     0},    // 1
    {      1,     1,     0,     1,     1,     0,     1},    // 2
    {      1,     1,     1,     1,     0,     0,     1},    // 3
    {      0,     1,     1,     0,     0,     1,     1},    // 4
    {      1,     0,     1,     1,     0,     1,     1},    // 5
    {      1,     0,     1,     1,     1,     1,     1},    // 6
    {      1,     1,     1,     0,     0,     0,     0},    // 7
    {      1,     1,     1,     1,     1,     1,     1},    // 8
    {      1,     1,     1,     1,     0,     1,     1},    // 9
    {      1,     1,     1,     0,     1,     1,     1},    // A
    {      0,     0,     1,     1,     1,     1,     1},    // B
    {      1,     0,     0,     1,     1,     1,     0},    // C
    {      0,     1,     1,     1,     1,     0,     1},    // D
    {      1,     0,     0,     1,     1,     1,     1},    // E
    {      1,     0,     0,     0,     1,     1,     1},    // F
};
```

SEG 所用到的 GPIO 初始配置。

```
    ……
void Segment_Initial(void)
{
    /* Set all segment pins to output low */
    SEG_A = SEG_B = SEG_C = SEG_D = \
    SEG_E = SEG_F = SEG_G = SEG_DP = 0;
    /* Set all segment pins to output mode */
    P12_PUSHPULL_MODE;    // SEG_A
    P11_PUSHPULL_MODE;    // SEG_B
    P36_PUSHPULL_MODE;    // SEG_C
    P04_PUSHPULL_MODE;    // SEG_D
    P24_PUSHPULL_MODE;    // SEG_E
    P21_PUSHPULL_MODE;    // SEG_F
    P37_PUSHPULL_MODE;    // SEG_G
    P03_PUSHPULL_MODE;    // SEG_DP

    /* Initial value */
    *(uint8_t *)&Scan_Pos = 0x01;
}
```

色阶所用到的 PWM 初始配置。

```
void PWM_Initial(void)
{
    /* Set all PWM pins to output mode */
    P31_PUSHPULL_MODE;    // Dig1_R
    P32_PUSHPULL_MODE;    // Dig1_G
    P10_PUSHPULL_MODE;    // Dig1_B
    P34_PUSHPULL_MODE;    // Dig2_R
    P05_PUSHPULL_MODE;    // Dig2_G
    P33_PUSHPULL_MODE;    // Dig2_B

    /* Set PWM clock source */
    PWM0_ClockSource(PWM_FSYS, 1);
    PWM123_ClockSource(PWM2, 1);
    PWM123_ClockSource(PWM3, 1);

    /* Set PWM period value */
    /* PWM0 for Dig2_G, Dig2_B */
    SFRS = 0;
    PWM0PH = PWM_FEQ_H;
    PWM0PL = PWM_FEQ_L;
    /* PWM2 for Dig1_R, Dig1_B */
    SFRS = 2;
    PWM2PH = PWM_FEQ_H;
    PWM2PL = PWM_FEQ_L;
    /* PWM3 for Dig1_G, Dig1_G */
    PWM3PH = PWM_FEQ_H;
    PWM3PL = PWM_FEQ_L;

    /* Reset SFRS */
    SFRS = 0;

    /* Enable PWM output */
    ENABLE_PWM2_CH1_P31_OUTPUT;    // Dig1_R
    ENABLE_PWM3_CH0_P32_OUTPUT;    // Dig1_G
    ENABLE_PWM2_CH0_P10_OUTPUT;    // Dig1_B
    ENABLE_PWM3_CH1_P34_OUTPUT;    // Dig2_R
    ENABLE_PWM0_CH2_P05_OUTPUT;    // Dig2_G
    ENABLE_PWM0_CH0_P33_OUTPUT;    // Dig2_B
};
```

每次更新色阶，需要重新定义 PWM 的 duty。

```
void Set_PWM_Duty(void)
{
    /* Digi_1 */
    if(((g_u8Scan_Pos == 0x07) && g_Dig_Setting[0].DP) || \
       ((g_u8Scan_Pos < 0x07) && (*(Show_Char[g_Dig_Setting[0].Char] + g_u8Scan_Pos))))
    {
        /* Enable segment */
        SFRS = 2;
        /* Dig1_R */
        PWM2C1H = ((255 - g_Dig_Setting[0].R) >> 4);
        PWM2C1L = ((255 - g_Dig_Setting[0].R) << 4);
        /* Dig1_G */
        PWM3C0H = ((255 - g_Dig_Setting[0].G) >> 4);
```

```
        PWM3C0L = ((255 - g_Dig_Setting[0].G) << 4);
        /* Dig1_B */
        PWM2C0H = ((255 - g_Dig_Setting[0].B) >> 4);
        PWM2C0L = ((255 - g_Dig_Setting[0].B) << 4);
    }
    else
    {
        /* Disable segment */
        SFRS = 2;
        /* Dig1_R */
        PWM2C1H = PWM_FEQ_H;
        PWM2C1L = PWM_FEQ_L;
        /* Dig1_G */
        PWM3C0H = PWM_FEQ_H;
        PWM3C0L = PWM_FEQ_L;
        /* Dig1_B */
        PWM2C0H = PWM_FEQ_H;
        PWM2C0L = PWM_FEQ_L;
    }

    /* Digi_2 */
    if(((g_u8Scan_Pos == 0x07) && g_Dig_Setting[1].DP) || \
       ((g_u8Scan_Pos < 0x07) && (*(Show_Char[g_Dig_Setting[1].Char] + g_u8Scan_Pos))))
    {
        /* Enable segment */
        SFRS = 2;
        /* Dig2_R */
        PWM3C1H = ((255 - g_Dig_Setting[1].R) >> 4);
        PWM3C1L = ((255 - g_Dig_Setting[1].R) << 4);
        SFRS = 0;
        /* Dig2_G */
        PWM0C2H = ((255 - g_Dig_Setting[1].G) >> 4);
        PWM0C2L = ((255 - g_Dig_Setting[1].G) << 4);
        /* Dig2_B */
        PWM0C0H = ((255 - g_Dig_Setting[1].B) >> 4);
        PWM0C0L = ((255 - g_Dig_Setting[1].B) << 4);
    }
    else
    {
        /* Disable segment */
        SFRS = 2;
        /* Dig2_R */
        PWM3C1H = PWM_FEQ_H;
        PWM3C1L = PWM_FEQ_L;
        SFRS = 0;
        /* Dig2_B */
        PWM0C0H = PWM_FEQ_H;
        PWM0C0L = PWM_FEQ_L;
        /* Dig2_G */
        PWM0C2H = PWM_FEQ_H;
        PWM0C2L = PWM_FEQ_L;
    }

    /* Reload new setting */
    set_PWM0CON0_LOAD;
    set_PWM2CON0_LOAD;
    set_PWM3CON0_LOAD;

    /* Wait new setting reload */
```

```
    while(PWM3CON0 & BIT6);

    /* Reset SFR */
    SFRS = 0;
}
```

主程序设定每 1ms 更新一次所有 GPIO 的状态，达成循环点亮所有 LED。

完成 LED 轮询点亮后，重置 Timer 0 定时并等候下一次定时中断。

```
    while(1)
    {
        if(TF0 == 1)
        {
            /* Stop Timer0 */
            clr_TCON_TR0;

            /* Set all segment pins to output low */
            SEG_A = SEG_B = SEG_C = SEG_D = \
            SEG_E = SEG_F = SEG_G = SEG_DP = 0;

            /* Set PWM duty */
            Set_PWM_Duty();

            /* Scan segment pins */
            SEG_A  = Scan_Pos.A;
            SEG_B  = Scan_Pos.B;
            SEG_C  = Scan_Pos.C;
            SEG_D  = Scan_Pos.D;
            SEG_E  = Scan_Pos.E;
            SEG_F  = Scan_Pos.F;
            SEG_G  = Scan_Pos.G;
            SEG_DP = Scan_Pos.DP;
            (Scan_Pos.DP == 0x01)?(*(uint8_t *)&Scan_Pos = 0x01):(*(uint8_t *)&Scan_Pos
             <<= 1);
            (g_u8Scan_Pos == 0x07)?(g_u8Scan_Pos = 0):(g_u8Scan_Pos++);

            /* Set next time-out */
            TH0 = TH0_INIT;
            TL0 = TL0_INIT;

            /* Clear Timer0 overflow flag */
            TF0 = 0 ;

           /* Start Timer0 */
           set_TCON_TR0;
        }
```

若有 UART 传输要求更改显示内容，判定 UART 接受字符串是否完整。接受完成后重新定义 PWM 色阶并 UART 回传更改结果。

```
            /* Check UART0 RX */
            if(UART0_RX_Flag)
            {
                /* Start UART RX */
                if(UART0_RX_TempCount == 0)
```

```
                UART0_RX_TempCount = UART0_RX_Count;
            else
            {
                /* UART stops */
                if(UART0_RX_TempCount == UART0_RX_Count)
                {
                    UART0_Timeout_Count++;
                    if(UART0_Timeout_Count == 5)
                    {
                        UART0_RX_Flag = 0;
                        UART0_TX_Flag = 1;
                        UART0_Timeout_Count = 0;
                    }
                }
                else
                {
                    UART0_RX_TempCount = UART0_RX_Count;
                    UART0_Timeout_Count = 0;
                }
            }
        }

    /* Set/Get command */
    if(UART0_TX_Flag)
    {
        /* Check valid data */
        if((UART0_RX_Data[0] == 0x5A) && (UART0_RX_Data[UART0_RX_Ptr-1] == 0xA5))
        {
            /* Set Command */
            if(UART0_RX_Data[1] < DIGI_COUNT)
            {
                g_Dig_Setting[UART0_RX_Data[1]].Char = UART0_RX_Data[2];
                g_Dig_Setting[UART0_RX_Data[1]].DP   = UART0_RX_Data[3];
                g_Dig_Setting[UART0_RX_Data[1]].R    = UART0_RX_Data[4];
                g_Dig_Setting[UART0_RX_Data[1]].G    = UART0_RX_Data[5];
                g_Dig_Setting[UART0_RX_Data[1]].B    = UART0_RX_Data[6];

                /* Transmit reply */
                DISABLE_UART0_INTERRUPT;
                for(i = 0; i < 8; i++)
                    UART_Send_Data(UART0, Set_Reply[i]);
                ENABLE_UART0_INTERRUPT;
            }
        }
        UART0_RX_Count = UART0_RX_Ptr = UART0_RX_TempCount = UART0_TX_Flag = 0;
    }
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - MS51_Series_BSP_Keil_V1.00.006
- IDE 版本
  - Keil uVision5 and PK51 Development Kit V9.52

3.2 硬件需求

- 电路组件
  - NuMaker-MS51PC V1.1
  - PWM_ARGB_Control_Board V1.0
- 线路示意图
  - 通过 NuMaker-MS51PC Arduino 接口连接，以显示范例代码的执行结果。

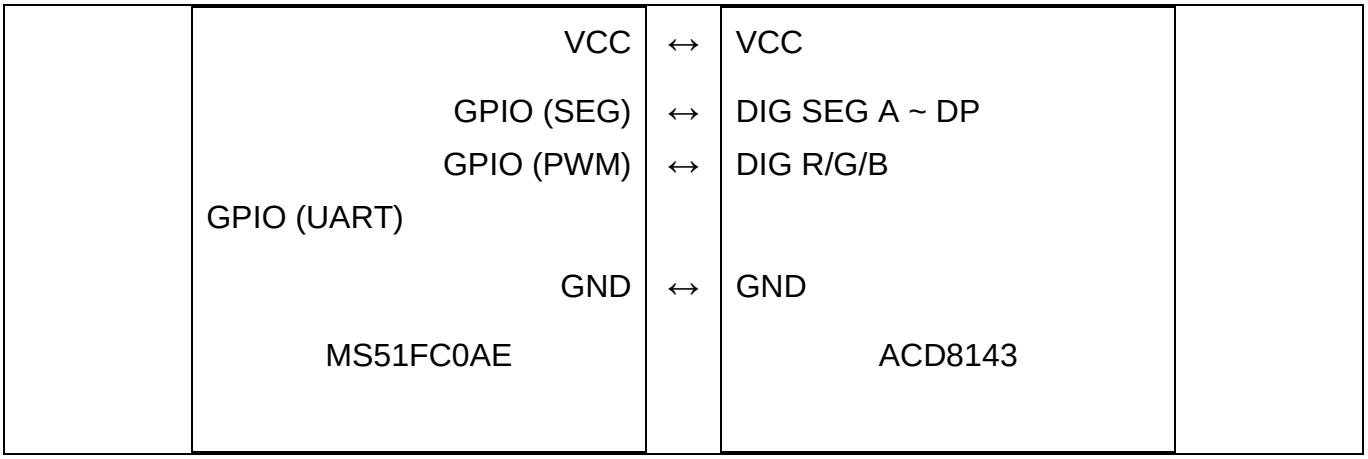

图 3-1 线路示意图

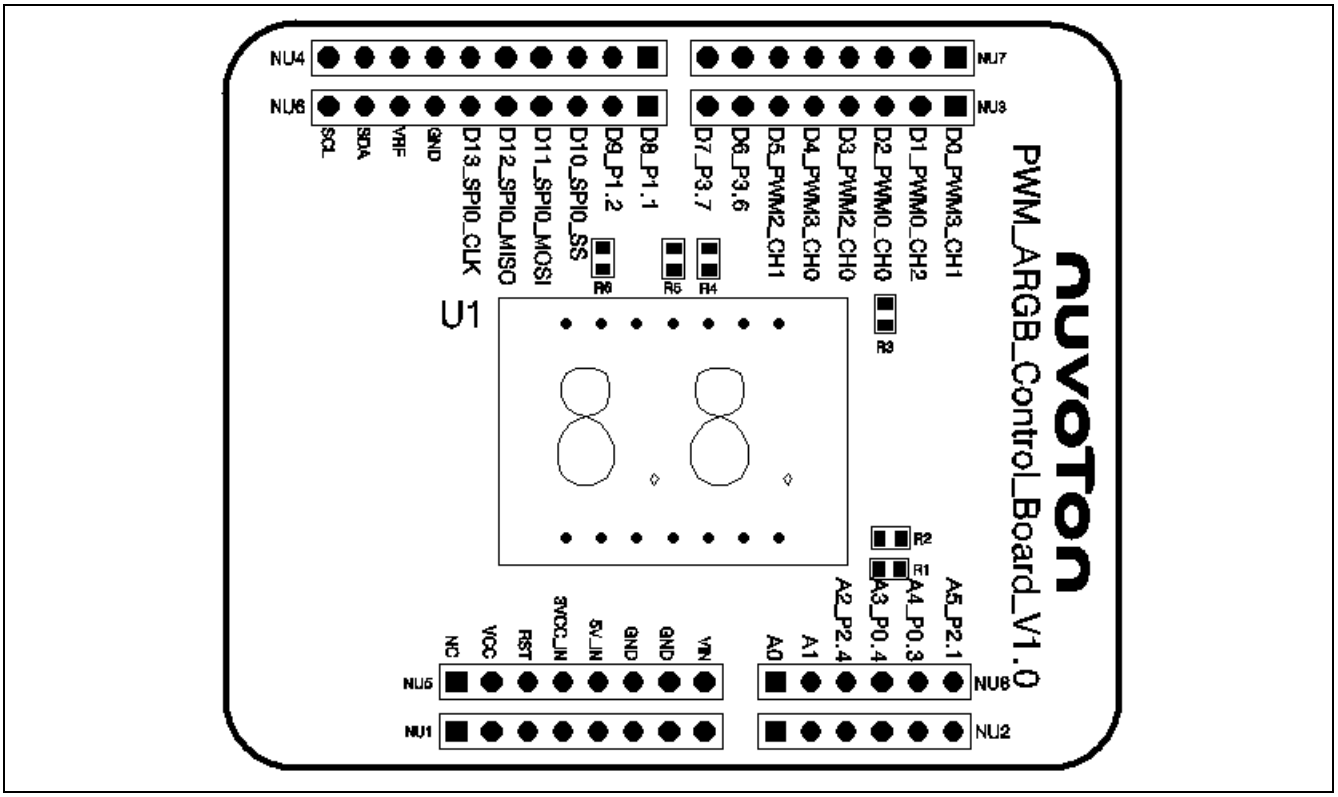

图 3-2 PWM_ARGB_Control_Board PCB

4. 目录信息

EC_MS51_RGB_7Segment_UART_V1.00	
Library	Sample code header and source files
Device	MS51 compliant device header file
Startup	Startup file for MS51
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 *RGB_7Segment_UART.uvproj*。
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
2023.08.11	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.