Using MS51 to Drive 7-segment RGB LED

Example Code Introduction for NuMicro® 1T 8051

Document Information

Application	This example code uses MS51 32K series to drive RGB color 7-segment LED display.
BSP Version	MS51_Series_BSP_Keil_V1.00.006
Hardware	NuMaker-MS51PC V1.1 & PWM_ARGB_Control_Board V1.0

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. Overview

1.1 Principle

The classic 7-segment display typically features a single color configuration, and users only need to connect the positive terminal to a power source and various negative terminals to ground (GND) to manipulate the 7-segment display to exhibit different numerical digits.

The RGB color model is based on an additive process, where the primary colors of light — red, green, and blue are combined. When applied to a 7-segment LED display, RGB serves the dual purpose of illuminating the display and allowing precise control of the intensity levels of the three primary colors, each with 256 levels. This enables the creation of a staggering 16.7 million mixed colors.

In this particular demonstration, ADH-Tech's ACD8143 RGB LED module has been employed for implementation. The LED pin configuration and internal drive circuitry are outlined as follows: Two RGB LEDs are activated, designated as DIG.1 and DIG.2.

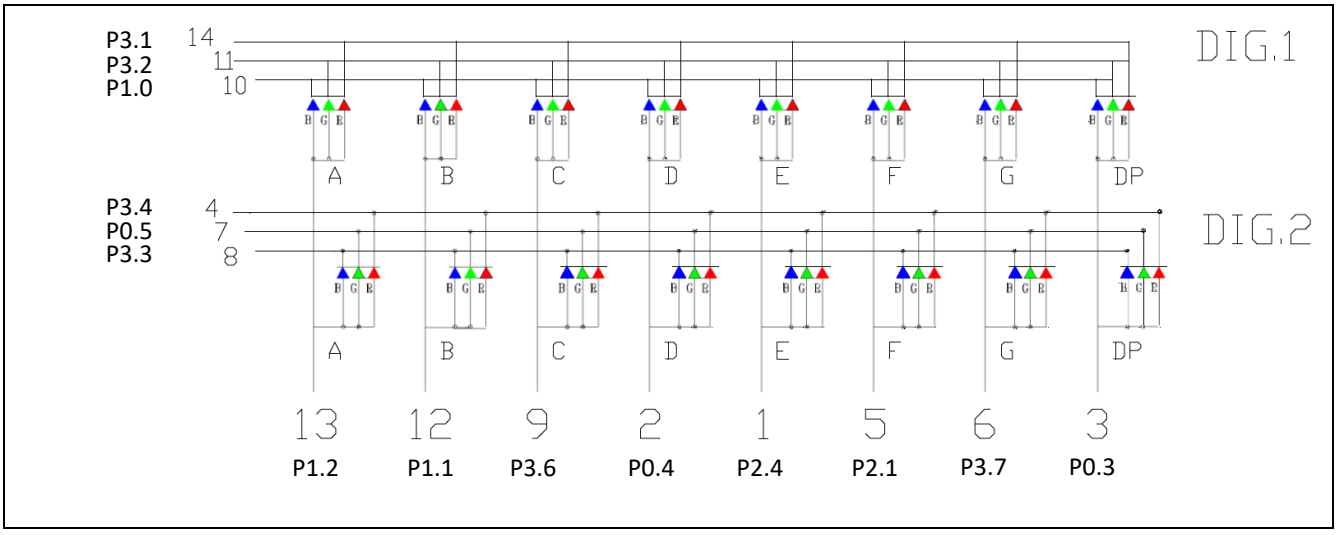

Figure 1-1 ACD8143 Pin Definition and Internal Drive Circuit

ACD8143 pins 10, 11, and 14 correspond to the PWM color channels for Blue (B), Green (G), and Red (R) control, responsible for illuminating DIG1. Meanwhile, pins 4, 7, and 8 are utilized to illuminate DIG2. The PWM (Pulse Width Modulation) output allows precise control over 255 levels of color intensity within the color scale.

The remaining pins are directly controlled via GPIO. In the example code program, these pins are designated as 'SEG' for easy reference. Specifically, 13, 12, 9, 2, 1, 5, 6 and 3 means SEG_A ~ SEG_G and SEG_DP.

For instance, if the requirement is to illuminate segment A of DIG1 in green, you can achieve this by configuring the LED 11-pin PWM output Duty to define the desired shade within the color scale. Additionally, set the 13-pin GPIO to 1 to enable the SEG configuration.

By leveraging the PWM and GPIO control mechanisms, the control board schematic is aligned with the Arduino interface of the NuMaker-MS51PC.

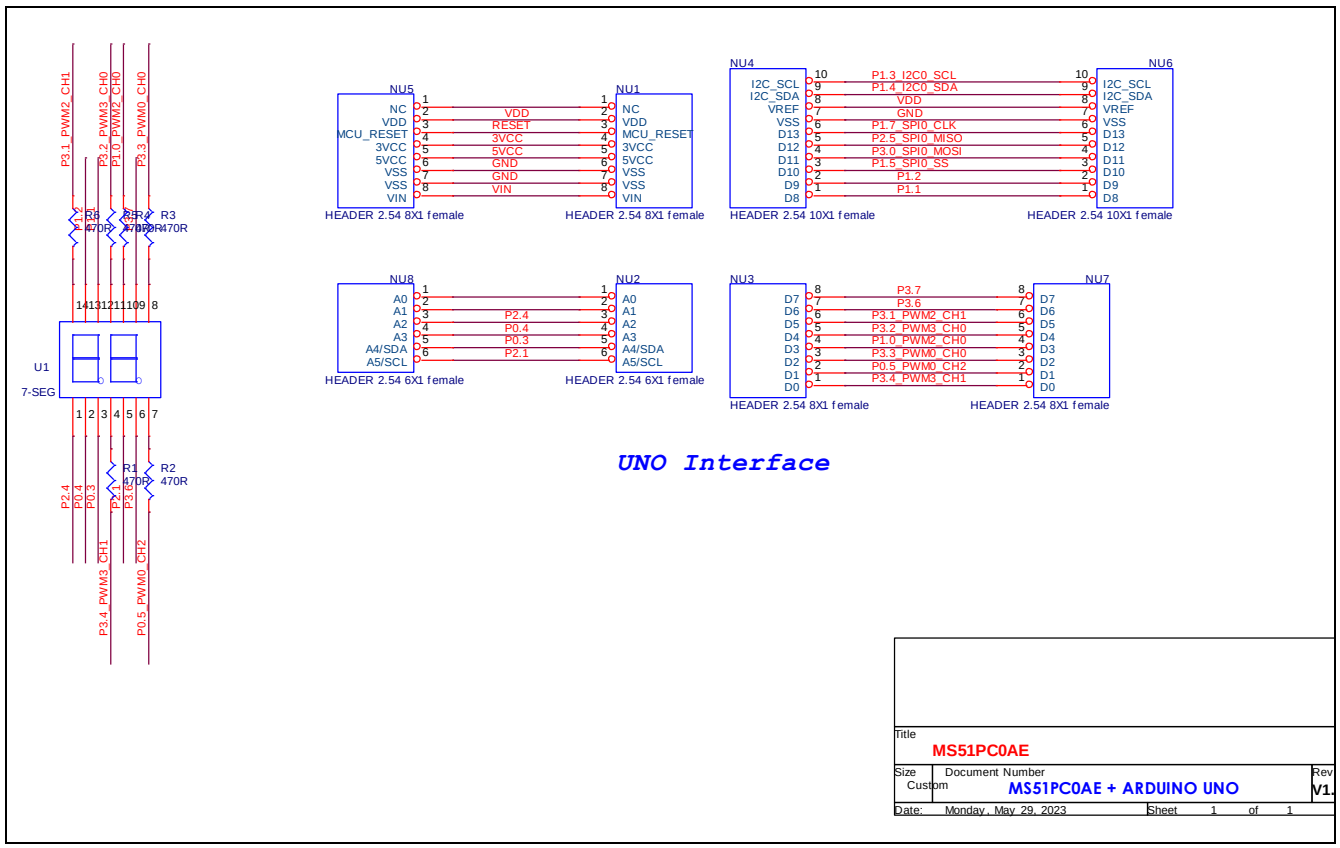

Figure 1-2 ACD8143 Control Board Schematic

1.2 Demo Result

Transmit the fixed format RGB LED through Access Port UART to display corresponding numbers and colors.

The UART transmission format is as follows, where each transmission defines a 7-segment.

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

Digit = 0 ~ 9, A~F;

Dot Point: 0 = Disable, 1 = Enable;

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

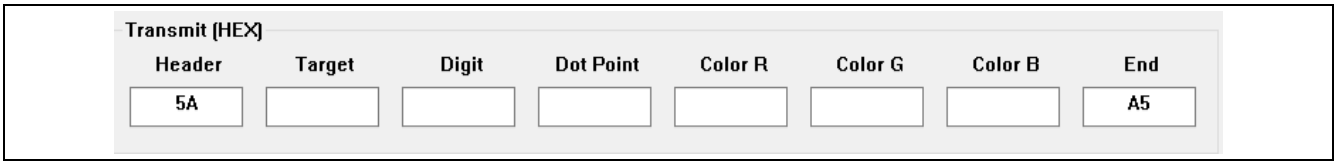

Figure 1-3 UART Transmission Format Definition

For example,

The first group of input 5A 00 03 00 69 AA 00 A5 controls DIG.1 to display the number [3], the color is light green.

The second group of input 5A 01 0A 00 FF AF 30 A5 controls DIG.2 to display the character [A], the color is bright yellow.

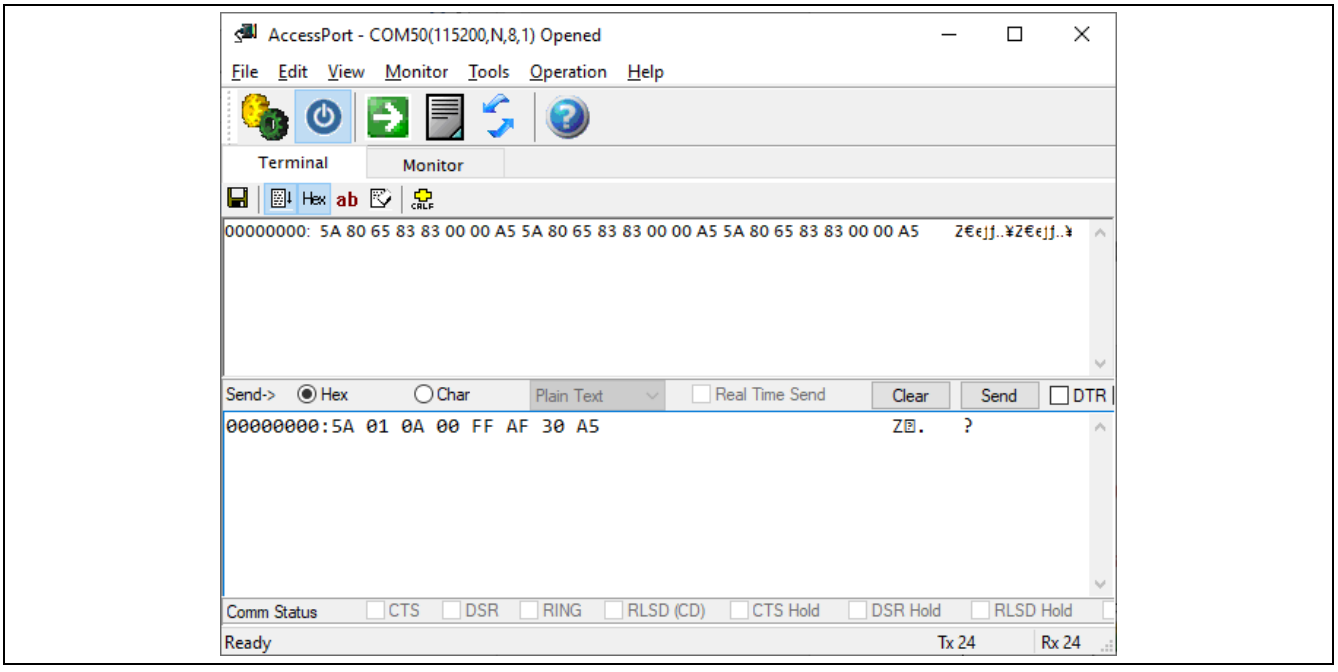

Figure 1-4 UART Input and Return Value

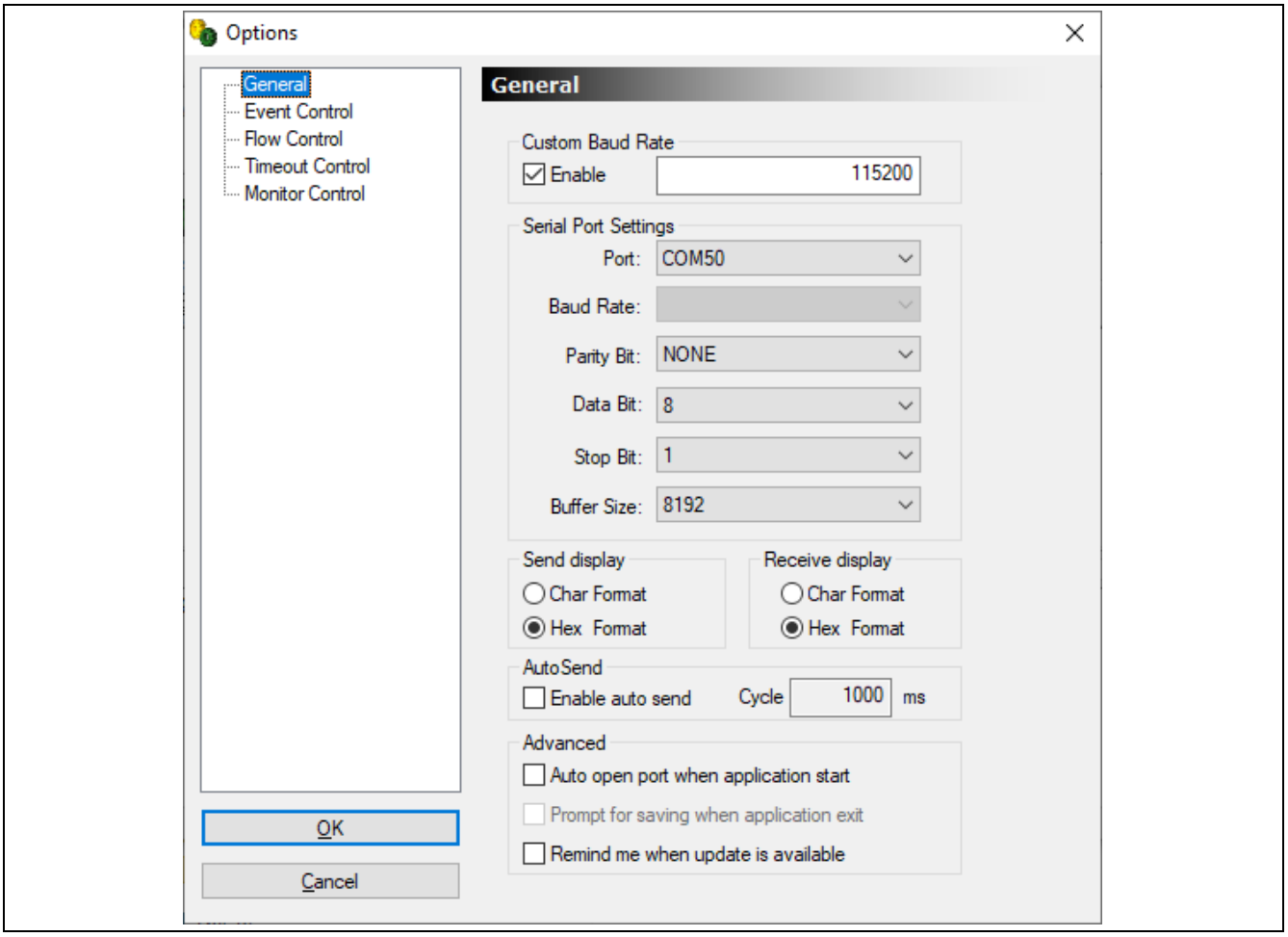

Figure 1-5 Access Port Configuration

2. Code Description

The function to drive uses PWM in the original code which is located in main.c.

It defines the SEG configurations for the various display characters on the 7-segment display, with DP indicating special lighting during PWM duty refresh cycles.

```
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
};
```

GPIO initial configuration is used by SEG.

```
    ……
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;
}
```

The initial PWM configuration is used for the RGB color scale.

```
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
};
```

Every time if the color scale needs to be updated, the PWM duty should be defined again.

```
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;
}
```

The main loop of this example code program is set to update the status of all GPIOs every 1ms to achieve a cycle of lighting all LEDs.

After completing LED polling, reset Timer 0 timing and wait for the next timing interrupt.

```
    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;
        }
```

If there is a UART transmission request to change the display content, determine whether the UART accepting string is complete. After accepting, redefine the PWM level and UART returns the change result.

```
            /* 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. Software and Hardware Requirements

3.1 Software Requirements

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

3.2 Hardware Requirements

- Circuit components
  - NuMaker-MS51PC V1.1
  - PWM_ARGB_Control_Board V1.0
- Pin Connect
  - Connect through the NuMaker-MS51PC Arduino interface to display the execution results of the sample code

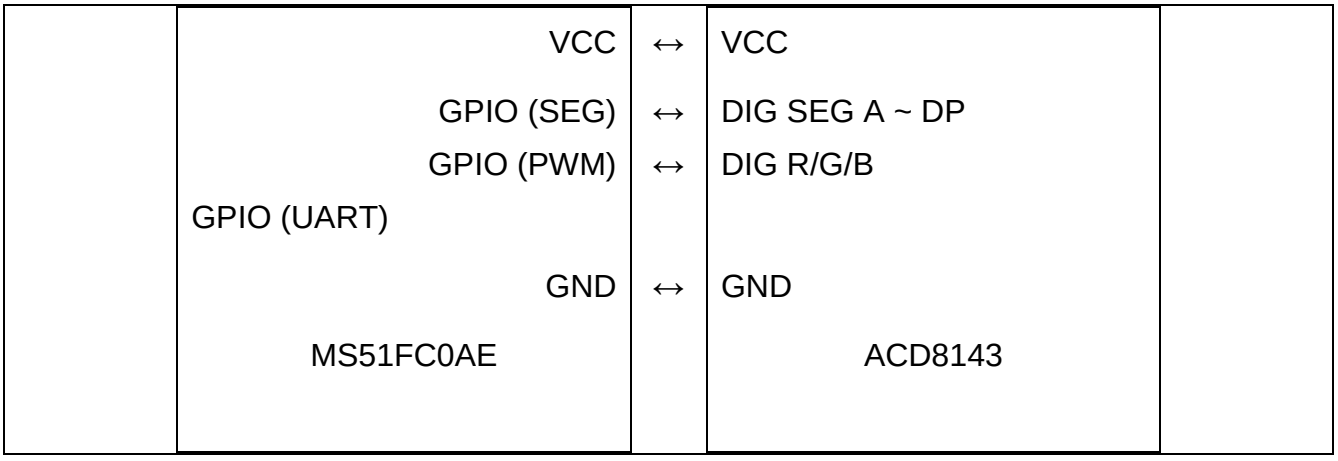

Figure 3-1 Pin Connect

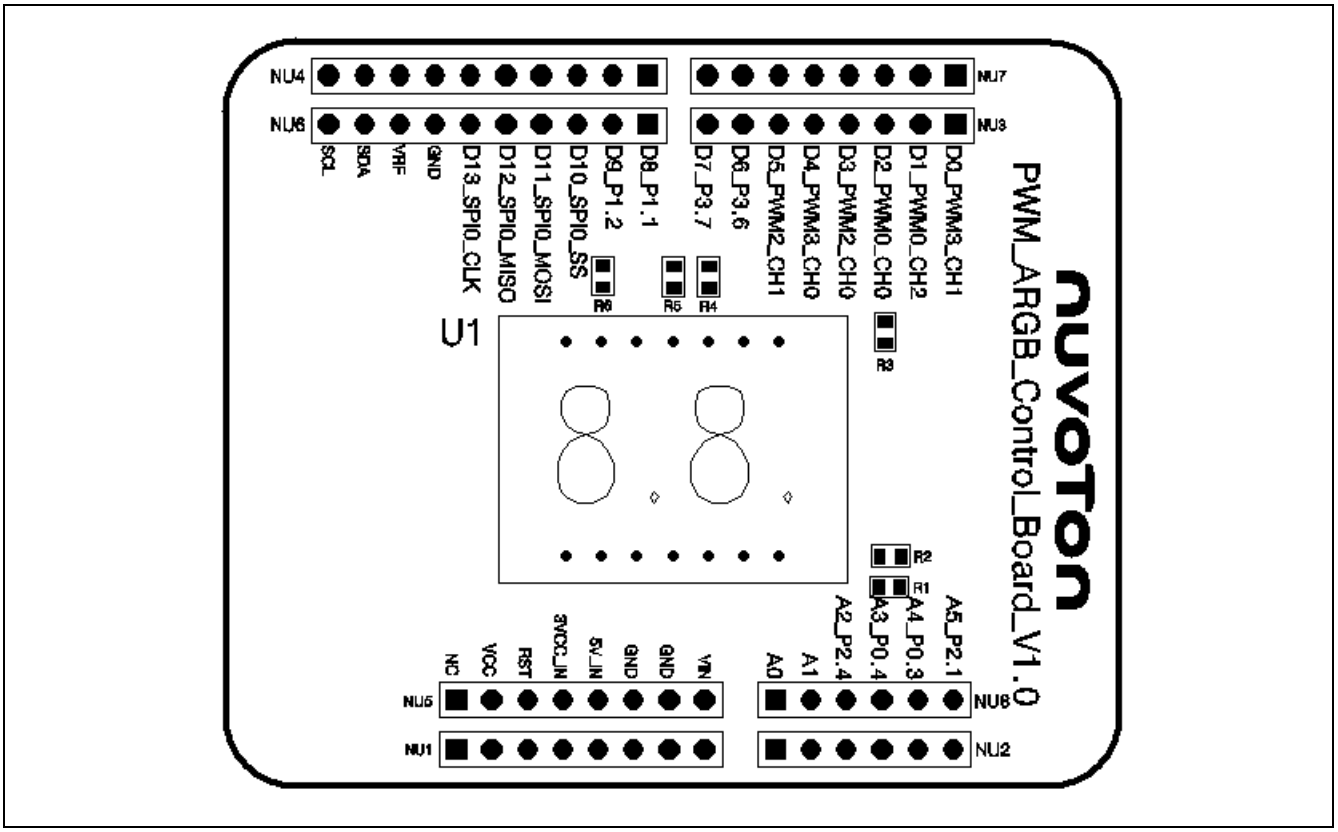

Figure 3-2 PWM_ARGB_Control_Board PCB

4. Directory Information

The directory structure is shown below.

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

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 *RGB_7Segment_UART.uvproj*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2023.08.11	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.