M251 DMX512 Control Demo

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses M251 to implement DMX512 control of ARGB2 LED.
BSP Version	M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.02.007
Hardware	NuMaker-M251KG 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. Overview

DMX512 protocol is a data dimming protocol proposed by the United States Institute for Theatre Technology (USITT). It provides a protocol standard for communication between lighting controllers and lighting fixtures. By using this protocol, it is possible to digitally control dimmers and other control devices in places such as stages, theaters, and studios. It is suitable for a master-slave control system and offers simple and reliable connectivity. This article describes how to use the M251 UART to decode DMX512 data packets and control ARGB2 LED lights.

1.1 Principle

1.1.1 ARGB2 LED Data Transfer Time (TH + TL = 1.2 μs ± 600 ns)

The ARGB2 LED timing sequence is shown in Figure 1-1 and Figure 1-2. Logic '0' defines high level 0.3 μs ± 0.15 μs and low level 0.9 μs ± 0.15 μs, and logic '1' is defined as high level 0.9 μs ± 0.15 μs and low level 0.3 μs ± 0.15 μs.

To use PSIO to control ARGB2 LED strips, the user must first set the timing rules (slot) and operate according to the expected timing. Divide the basic unit of time into about 0.3 μs to create the desired waveform. The timing calculation formula is as follows:

- PCLK1 is selected as PSIO clock source; PCLK1 is calculated as HIRC (48 MHz) / 4 = 12 MHz.
- PSIO clock uses PCLK1 divided by 4 = 3 MHz.
- PSIO one slot cycle time = 1 / 3 MHz = 0.33 μs.

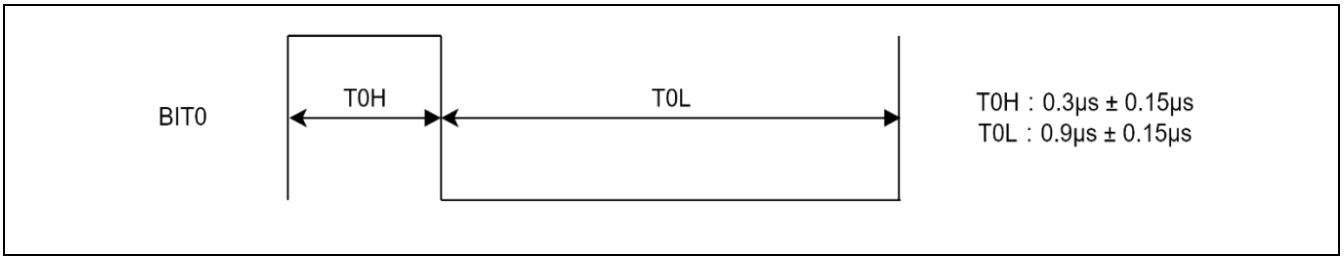

Figure 1-1 Logic ‘0’ (bit 0)

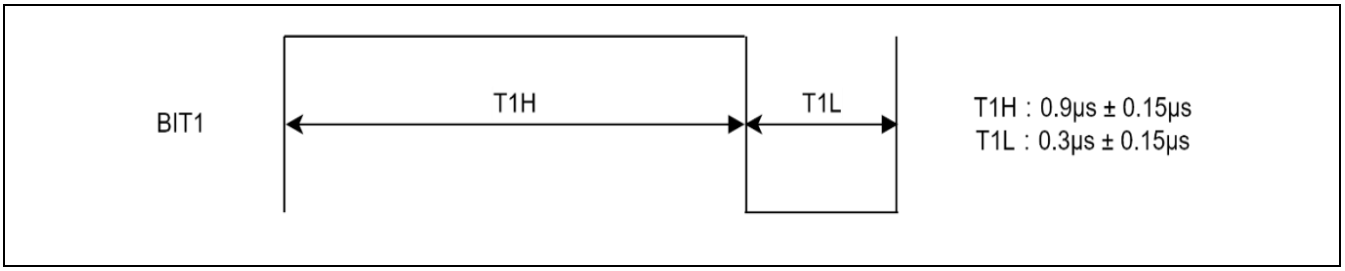

Figure 1-2 Logic ‘1’ (bit 1)

1.1.2 DMX512 Data Frame

A byte is a data frame, also known as a control channel, which can control one or multiple functions of lighting devices. A data frame consists of 1 start bit (S), 8 data bits (D0~D7), and 2 stop bits (E), totaling 11 bits. It adopts unidirectional asynchronous serial transmission, as shown in Figure 1-3.

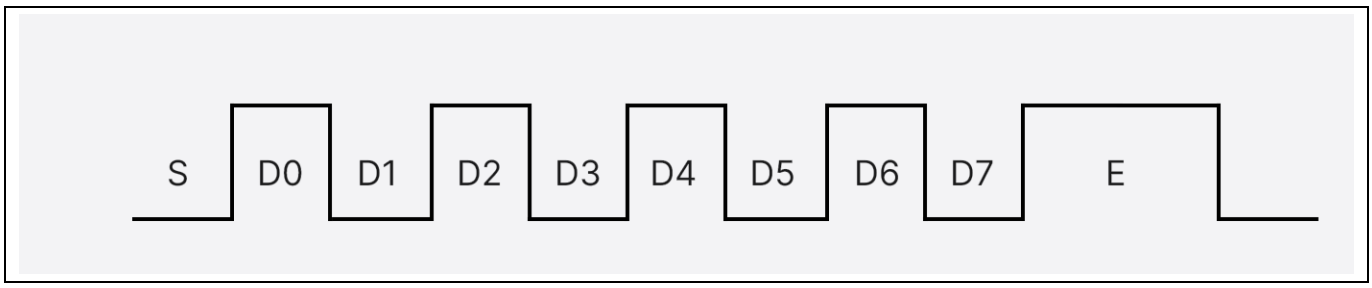

Figure 1-3 DMX512 Data Frame

D0~D7 are 8-bit dimming value, which can generate 256 brightness levels. 0x00 corresponds to the darkest, and 0xff corresponds to the brightest. The bit width is 4 us, with 11 bits per frame, frame width is 44 us, and the transmission rate is 1/44= 250 kbps.

1.1.3 DMX512 Data Packet

A DMX512 data packet consists of one MTBP bit, one Break bit, one MAB bit, one SC frame, and 512 data frames, as shown in Figure 1-4.

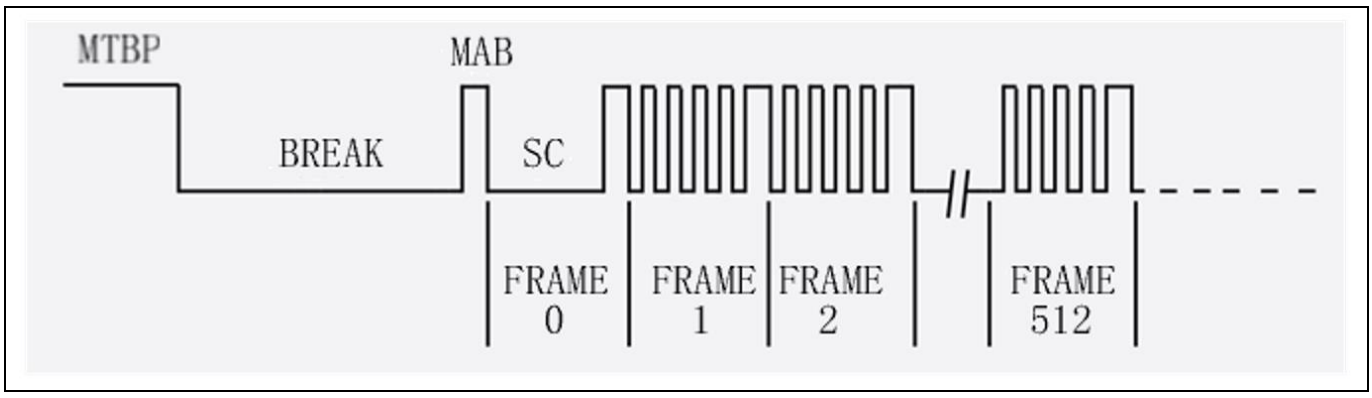

Figure 1-4 DMX512 Data Packet

- MTBP (Mark Time Between Packets) bit indicates a data packet has been sent.
- BREAK bit indicates the start of a new packet, with a value not less than 88 us (22 bits).
- MAB (Mark After Break) bit separates the BREAK and SC frame.
- SC (Start Code) is also known as a NULL START Code, value is 0x00, identifies the function of subsequent data frames in the packet.
- Data Frames (1-512) represent the dimming levels for each receiving device are placed.

1.1.4 UART Decoding DMX512 Data

1. Configure UART baud rate to 250 kbps, 8-bit data length, 2 STOP bits.
2. Enable RDA/RLS interrupt.
3. After receiving the BREAK bit and SC frame, start receiving data frames (1-512).
4. Control channel data used to dim LED lights.

2. Code Description

Use PSIO_StripInit() to initialize PSIO Slot Control and PSIO pin.

```
void PSIO_StripInit(void)
{
    uint8_t i;

    gsPSIO_ARGB2_Cfg[0].u8SlotCtrl = PSIO_SC0;
    gsPSIO_ARGB2_Cfg[0].u8DataPin = PSIO_PIN0;

    gsPSIO_ARGB2_Cfg[1].u8SlotCtrl = PSIO_SC1;
    gsPSIO_ARGB2_Cfg[1].u8DataPin = PSIO_PIN1;

    for (i = 0; i < 2; i++)
    {
        PSIO_Strip_OpenOutput(gsPSIO_ARGB2_Cfg[i], PSIO_PIN_ENABLE);
    }
}
```

Use PSIO_Strip_OpenOutput() when initializing the output mode. Use PSIO_SET_GENCTL() to set the pin controller, mode, start state and end state. PSIO_SET_WIDTH() and PSIO_SET_OUTPUT_DEPTH() are related to the data format. In this example, the output width is initially set to 8 bits, the input width is 1 bit, and the depth is both 1.

```
static void PSIO_Strip_OpenOutput(S_PSIO_ARGB2_STRIP_CFG_T gsPSIO_Cfg, uint32_t u32Switch)
{
    /* PSIO UART TX PIN general setting */
    PSIO_SET_GENCTL(PSIO,
                    gsPSIO_Cfg.u8DataPin,
                    u32Switch, //PSIO_PIN_ENABLE
                    gsPSIO_Cfg.u8SlotCtrl,
                    PSIO_OUTPUT_MODE,
                    PSIO_LOW_LEVEL,
                    PSIO_LOW_LEVEL);

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

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

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

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

The waveform signal function of output logic '0' and logic '1' requires three different actions, namely output high level, output data from the buffer and output low level. The sequence diagram is shown in Figure 1-1 and Figure 1-2 Slot 0 ~ Slot 2 can be regarded as 1 bit of ARGB2 LED protocol. PSIO_SET_WIDTH() sets the output width according to the size of the output data. For example, to set the color of the ARGB2 LED, users must set the output width to 24 bits. PSIO_SET_SCCTL() uses the loop function of PSIO to replicate this behavior 23 times, that is, output 24 bit data. After the setting is completed, use the software triggers PSIO_START_SC() to start the slot controller execution.

```
void PSIO_SendDataWithSize(S_PSIO_ARGB2_STRIP_CFG_T gsPSIO_Cfg, 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_Cfg.u8DataPin, &sDataConfig);

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

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

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

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

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

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

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

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

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

Use DMX512_UART_Init() to initialize UART function.

```
void DMX512_UART_Init(void)
{
    /* Configure DMX512_UART_nRTS as OUTPUT mode */
    GPIO_SetMode(DMX512_UART_nRTS_PORT, DMX512_UART_nRTS_BIT, GPIO_MODE_OUTPUT);
    DMX512_UART_nRTS_PIN = 0;

    /* Set multi-function pins for DMX512_UART RX and TX */
    DMX512_UART_RX_MFP &= ~(DMX512_UART_RX_MFP_MSK | DMX512_UART_TX_MFP_MSK);
    DMX512_UART_TX_MFP |= (DMX512_UART_RX_MFP_FUNC | DMX512_UART_TX_MFP_FUNC);

    /* Configure DMX512_UART and set DMX512_UART baud rate */
    UART_Open(DMX512_UART, DMX512_UART_BAUDRATE);
    DMX512_UART->LINE = UART_WORD_LEN_8 | UART_PARITY_NONE | UART_STOP_BIT_2;

    /* Enable RDA\RLS Interrupt */
    NVIC_EnableIRQ(DMX512_UART_IRQn);
    UART_EnableInt(DMX512_UART, (UART_INTEN_RDAIEN_Msk | UART_INTEN_RLSIEN_Msk));
}
```

Use DMX512_UART_IRQHandler() to handle UART interrupts and receive DMX512 packets data.

```
void DMX512_UART_IRQHandler(void)
{
    uint32_t u32IntSts = DMX512_UART->INTSTS;
    uint8_t u8InChar;

    if (u32IntSts & UART_INTSTS_RLSIF_Msk)
    {
        /* Break Interrupt Flag */
        /* This bit is set to logic "1" whenever the received data input (RX) is held in
           the "spacing state" (logic 0) for longer than a full word transmission time
           (that is, the total time of "START bit" + data bits + parity + STOP bits). */
        if (DMX512_UART->FIFOSTS & UART_FIFOSTS_BIF_Msk)
        {
```

```
            u8BreakFlag = 1;
            u16RecvLen = 0;
            u8ScFlag = 0;
        }

        /* Framing Error Flag */
        /* This bit is set to logic "1" whenever the received character does not have a
           valid "STOP bit" (that is, the STOP bit following the last data bit or PARITY
           bit is detected as logic 0). */
        else if (DMX512_UART->FIFOSTS & UART_FIFOSTS_FEF_Msk)
        {
            u8BreakFlag = 0;
            u16RecvLen = 0;
            u8ScFlag = 0;
        }

        u8InChar = UART_READ(DMX512_UART);
    }

    else if (u32IntSts & UART_INTSTS_RDAINT_Msk)
    {
        /* Get all the input characters */
        while (UART_GET_RX_EMPTY(DMX512_UART) == 0)
        {
            /* Get the character from UART Buffer */
            u8InChar = UART_READ(DMX512_UART);

            if (u8BreakFlag)
            {
                if (u8ScFlag == 0)
                {
                    if (u8InChar == 0x00)
                    {
                        u8ScFlag = 1;
                        g_u8Dmx512Buff[u16RecvLen++] = u8InChar;
                    }
                    else
                        u8BreakFlag = 0;
                }
                else
                {
                    g_u8Dmx512Buff[u16RecvLen++] = u8InChar;
                }
            }

            if (u16RecvLen>512)
                u16RecvLen = 0;
        }

        /*Forces a write of all user-space buffered data for the given output*/
        fflush(stdout);
    }

    if (DMX512_UART->FIFOSTS & (UART_FIFOSTS_BIF_Msk | UART_FIFOSTS_FEF_Msk |
                                UART_FIFOSTS_PEF_Msk | UART_FIFOSTS_RXOVIF_Msk))
    {
```

```
        DMX512_UART->FIFOSTS = (UART_FIFOSTS_BIF_Msk | UART_FIFOSTS_FEF_Msk |
                                UART_FIFOSTS_PEF_Msk | UART_FIFOSTS_RXOVIF_Msk);
    }
}
```

Use Strip_PSIO_DMX512_Test() to process DMX512 data frame and dim the ARGB2 LED through the PSIO module. The data frame from g_u8Dmx512Buff[1] to g_u8Dmx512Buff[512] corresponds to the brightness of the first to 512th lights respectively. The variable g_u8Addr represents the device address of the DMX512 control board.

```
void Strip_PSIO_DMX512_Test(void)
{
    uint8_t i, j;
    uint8_t u8addr;

    u8addr = g_u8Addr;

    for (i = 0; i < sizeof(g_u8TransferTbl); i++)
    {
        g_u8TransferTbl[i] = 0;
    }

    for (i = 0; i < sizeof(LED_POS_TBL); i++)
    {
        g_u8TransferTbl[(LED_POS_TBL[i]-1)*3] = g_u8Dmx512Buff[i*3+1+u8addr];
        g_u8TransferTbl[(LED_POS_TBL[i]-1)*3+1] = g_u8Dmx512Buff[i*3+2+u8addr];
        g_u8TransferTbl[(LED_POS_TBL[i]-1)*3+2] = g_u8Dmx512Buff[i*3+3+u8addr];
    }

    for (i = 0; i < STRIP_NUM; i++)
    {
        uint8_t channel = (i == 0) ? 1 : 0;

        for (j = 0; j < LED_NUM; j++)
        {
            PSIO_StripSetRGBColor(channel, g_u8TransferTbl[i * 36+j*3],
                 g_u8TransferTbl[i * 36+j*3+1], g_u8TransferTbl[i * 36+j*3+2]);
        }
    }
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.02.007
- IDE version
  - Keil uVersion V5.41.0.0

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M251KG V1.1
  - NuMaker-Lighting-ARGB
- USB to DMX512 Bridge

  User can change the Board Address by using two dip switches, shown in below, where the Board Address (0-255) correspond to the DMX512 data frames (1-256).

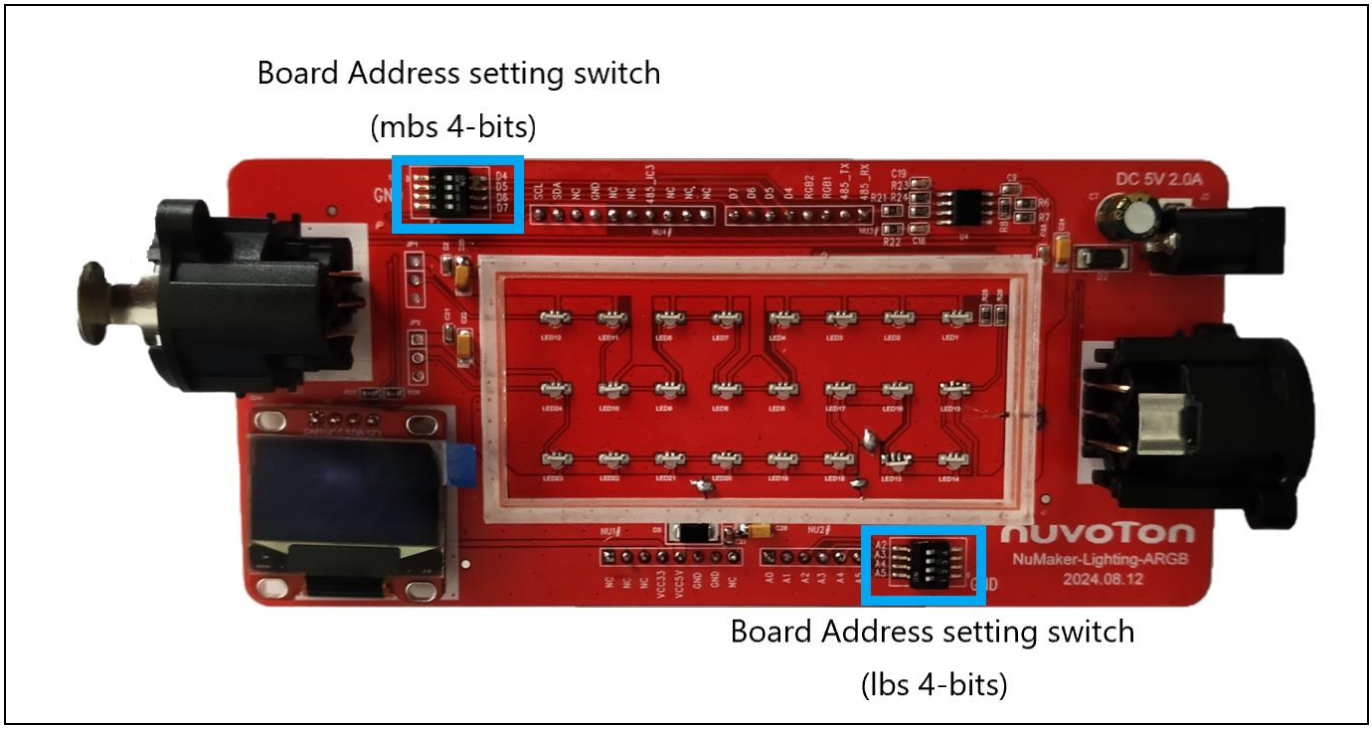

Figure 3-1 Board Address Setting Switch

4. Directory Information

The directory structure is shown below.

EC_M251_DMX512_Control_V1.00	
Tool	Tool
DMX512_Controller	Source file of DMX512_Controller
Library	Sample code header and source file
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	
Project	Source file of example code

Figure 4-1 Directory Structure

5. Example Code Execution

1. Insert the USB to DMX512 Bridge into the USB port of the computer, and connect the XLR connector to the NuMaker-Lighting-ARGB board, as shown in Figure 5-1

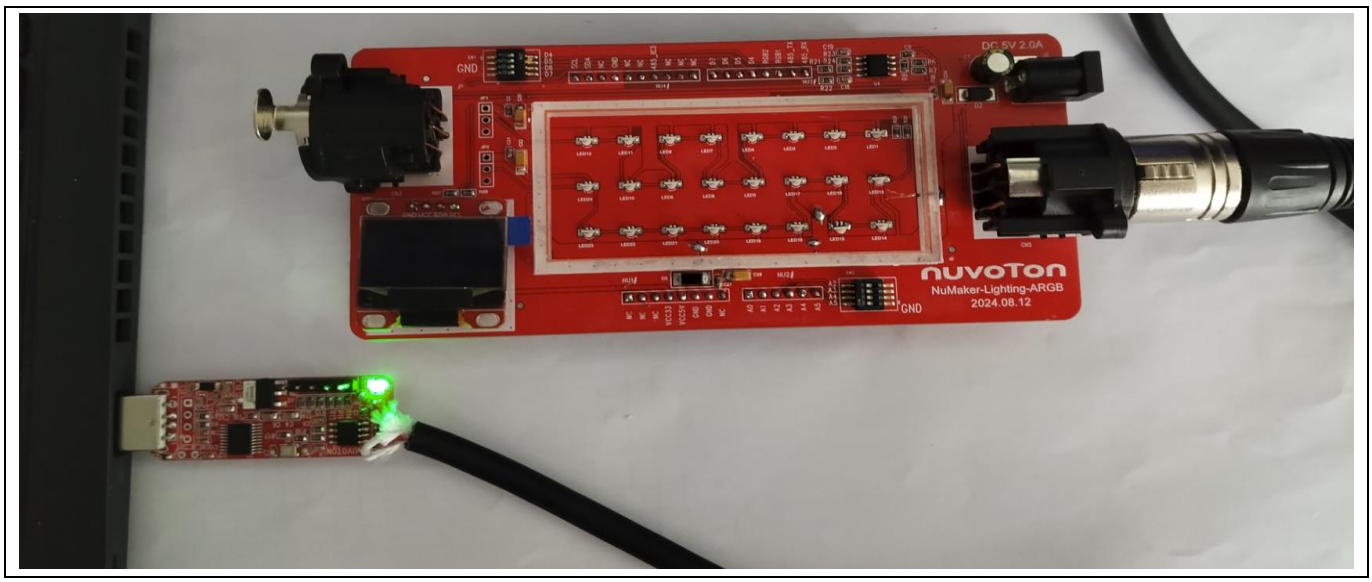

Figure 5-1 USB to DMX512 Bridge Connection

2. Browse the DMX512_Controller folder as described in the Directory Information section and double-click DMX512_Controller. Click on “Search” -> Select“PID:5020” from the dropdown menu and click “Connect”. According to the Board Address of NuMaker-Lighting-ARGB, click on the dropdown menu on the right side of “Selection” and choose the corresponding Board.

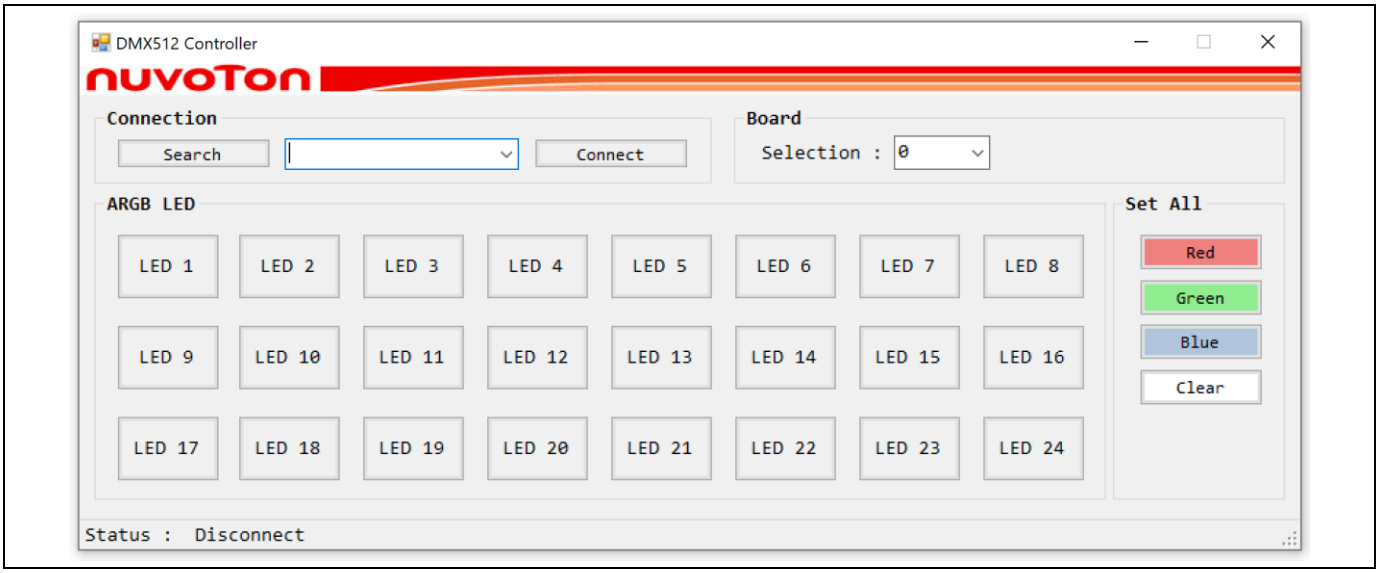

Figure 5-2 DMX512_Controller GUI

3. Browse the sample code folder as described in the Directory Information section and double-click *M251_DMX512_Control.uvprojx*.
4. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
5. Enter debug mode.
   - Run
   - Click on the DMX512 Controller GUI “LED1…LED24” and adjust the value to dim the selected ARGB LED.

6. Revision History

Date	Revision	Description
2024.10.25	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.