Using M0A23 GPIO to Achieve ARGB LED Marquee Effect

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses GPIO to simulate ARGB LED timing sequence and achieve the ARGB LED Marquee effect on the M0A23 series microcontroller (MCU).
BSP Version	M0A21_Series_BSP_CMSIS_V3.01.000
Hardware	Bling Bling Board Ver2.0

1. Overview

This example code uses M0A23 GPIO to simulate ARGB LED timing sequence and produce ARGB LED marquee effect. This code uses one set of timer to schedule ARGB LED to do ARGB LED toning, one set of GPIO to simulate ARGB LED timing sequence, and one set of UART to enter display word.

1.1 Principle

1.1.1 ARGB LED Data Transfer Time (TH + TL = 1200ns ± 160ns)

Figure 1-1 Logic '0' is defined as a high pulse of 300ns ± 80ns followed by a low pulse of 900ns ± 80ns. Figure 1-2 Logic '1' is defined as a high pulse of 900ns ± 80ns followed by a low pulse of 300ns ± 80ns. Figure 1-3 Reset time needs to be greater than 50 us.

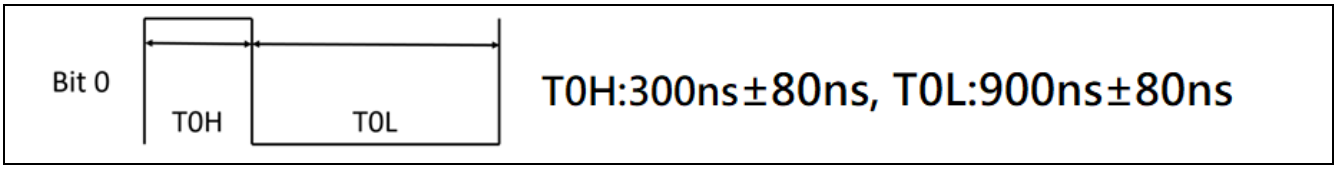

Figure 1-1 Logic '0' (Bit 0)

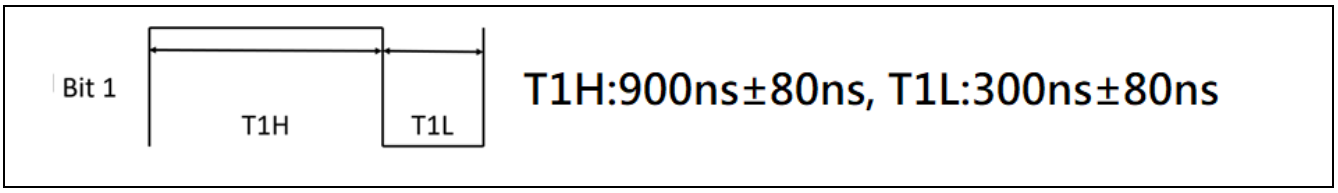

Figure 1-2 Logic '1' (Bit 1)

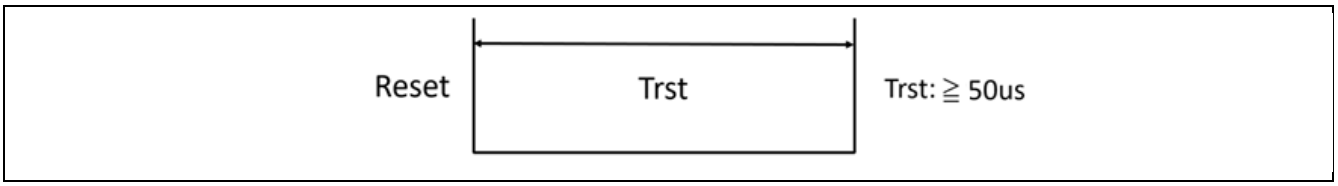

Figure 1-3 Reset (Trst)

1.1.2 ARGB LED Characteristics and Connection Methods

Users can control the color of ARGB LEDs with just one signal line. Compared with traditional RGB LEDs, ARGB LEDs are easier to design and save layout space. The connection is as shown in Figure 1-4, from the GPIO of MCU to the input of LED1 (DIN), and then the output of LED1 (DOUT) will be connected to the input of LED2 (DIN).

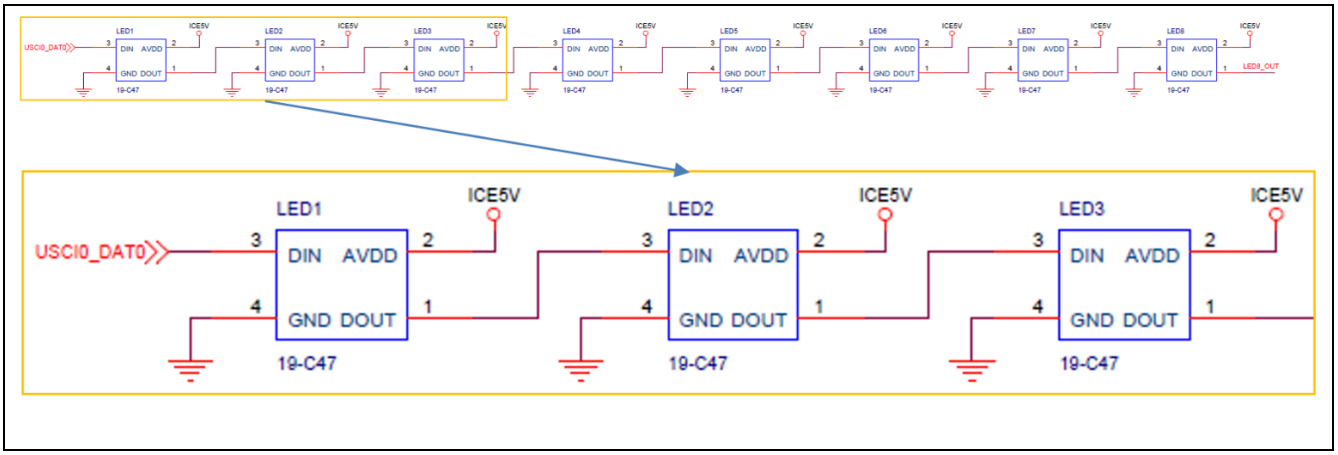

Figure 1-4 ARGB LED Cascading Connection

1.1.3 GPIO Clock Frequency and Logic Bit Design

Use GPIO to control ARGB LED. Delay the high and low levels for a period of time through the NOP() instruction. The system clock HIRC is 48 MHz. Thus, the time to execute a command is about 1 / 48 MHz (20.83ns). It can be obtained that the NOP() instruction needs to be executed several times.

In Figure 1-1 and Figure 1-2, logic '0' is defined as a high level of 300ns ± 80ns (220ns ≤ TH0 ≤ 380ns) and a low level of 900ns ± 80ns (820ns ≤ TL0 ≤ 980ns). Logic '1' is defined as a high level of 900ns ± 80ns (820ns ≤ TH1 ≤ 980ns) and a low level of 300ns ± 80ns (220ns ≤ TL1 ≤ 380ns). In this example code, the subroutine for logic '0' is ARGB_sendBIT0(), and the subroutine for logic '1' is ARGB_sendBIT1(). Within the subroutines for logic '0' and logic '1', the NOP() instruction is used, taking approximately 1 / 48MHz (20.83ns) to execute.

It can be inferred from the above that in the design, logic '0' requires a minimum of approximately 11 NOP() instructions for the high level (220/20.83≈11NOP()), and a maximum of approximately 19 NOP() instructions (380/20.83≈19NOP()). For the low level of logic '0', a minimum of approximately 40 NOP() instructions is needed (820/20.83≈40NOP()), and a maximum of approximately 48 NOP() instructions (980/20.83≈48NOP()). As for logic '0', the high level requires a minimum of around 40 NOP() instructions (820/20.83≈40NOP()), and a maximum of approximately 48 NOP() instructions (980/20.83≈48NOP()). The low level of logic '1' demands a minimum of approximately 11 NOP() instructions (220/20.83≈11NOP()), and a maximum of approximately 19 NOP() instructions (380/20.83≈19NOP()). The clock frequency of the ARGB LED can be designed based on the above conditions.

In Figure 1-5, the high level of logic '0' is at 280ns (280/20.83≈14NOP()), and the low level is at 980ns (980/20.83≈48NOP()). For logic '1', the high level is at 920ns (920/20.83≈45NOP()), and the low level is at 340ns (340/20.83≈17NOP()).

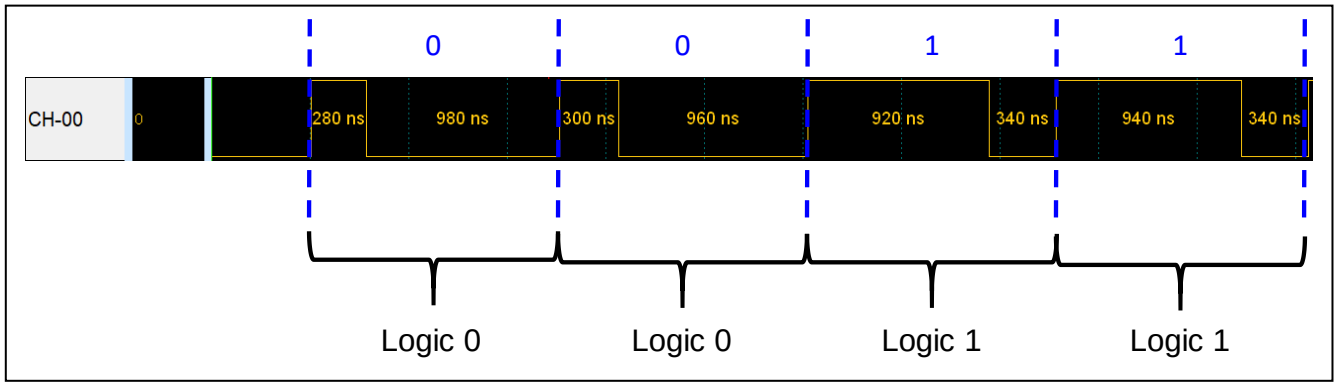

Figure 1-5 Delay High and Low Levels for a Period through NOP Instruction

The example code utilizes Keil C, which translates C language into assembly language during compilation. Due to the execution time of C language instructions such as loops or assigning variable values, there might be a need for several NOP() instructions, requiring adjustments in the actual program design.

In the example code, the ARGB_sendOne() subroutine is used to distinguish logic '0' and logic '1'. The ARGB_sendBIT0() handles logic '0', while ARGB_sendBIT1() handles logic '1'. Within the ARGB_sendBIT0() subroutine, there are 11 NOP() instructions for the high level. Due to additional 7 NOP() instructions introduced during execution, influenced by variable assignments, conditional statements (if...else), and loops (for) within the ARGB_sendOne() subroutine, the actual high-level logic '0' would then be (11 + 7 = 18) 18 NOP() instructions. As shown in Figure 1-6, the real high-level logic '0' takes 357ns (357/20.83≈18 NOP()). In the ARGB_sendBIT0() subroutine, the low level contains 29 NOP() instructions. Similarly influenced by variable assignments, conditional statements (if...else), and loops (for) within the ARGB_sendOne() subroutine, an additional 14 NOP() instructions are introduced during execution. Consequently, the actual low-level logic '0' would then be (29 + 14 = 43) 43 NOP() instructions. As depicted in Figure 1-7, the real low-level logic '0' takes 892ns (892/20.83≈43 NOP()).

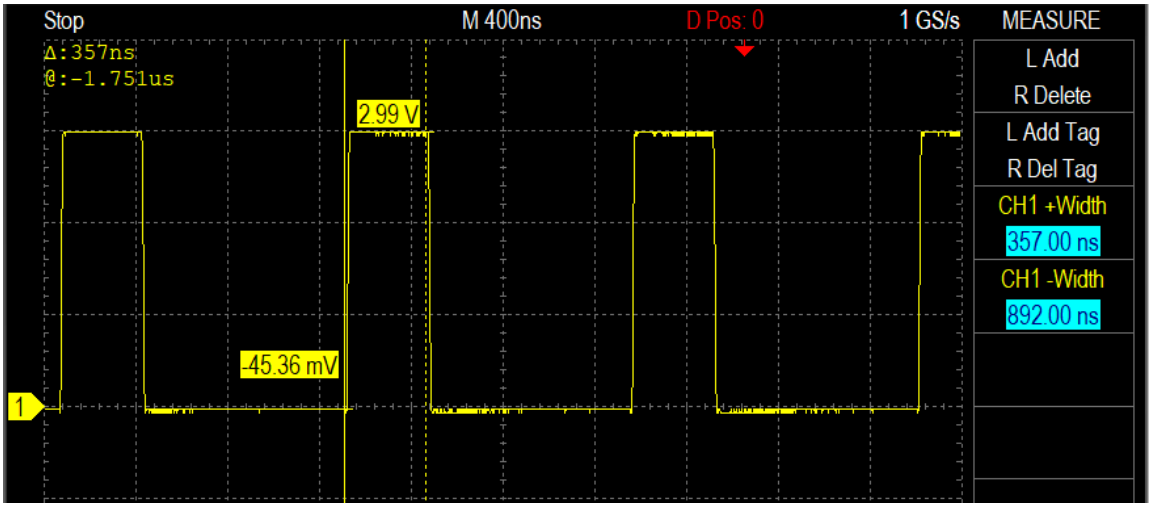

Figure 1-6 Logic '0' High-Level Waveform

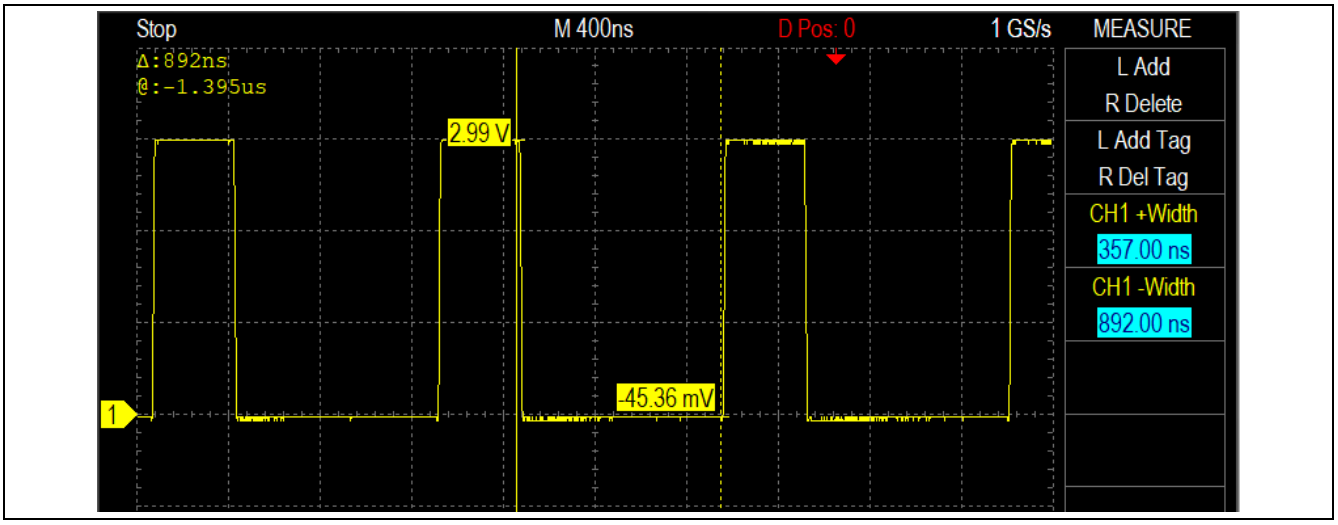

Figure 1-7 Logic '0' Low-Level Waveform

1.2 Demo Result

1.2.1 Logic '1' (Bit 1), Logic '0' (Bit 0), Reset (Trst) Demo Result

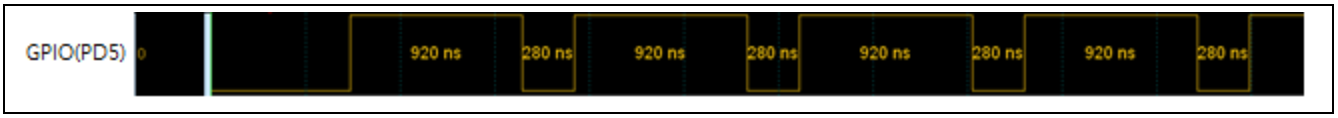

Figure 1-8 Logic '1' Demo Result

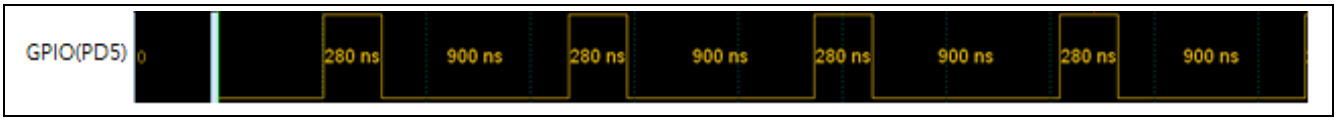

Figure 1-9 Logic '0' Demo Result

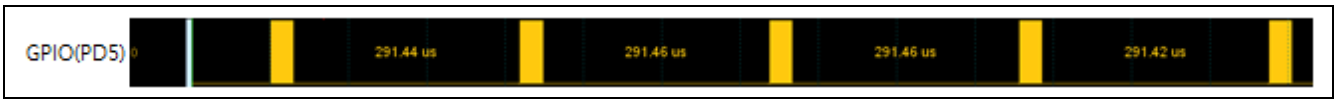

Figure 1-10 Trst Demo Result

1.2.2 Terminal Software Demo Result

Use terminal software to select execution mode through UART. Please set baud rate to 115200. The demo result is shown in Figure 1-11 ARGB LED marquee demo effect is shown in Figure. The mode is as follows:

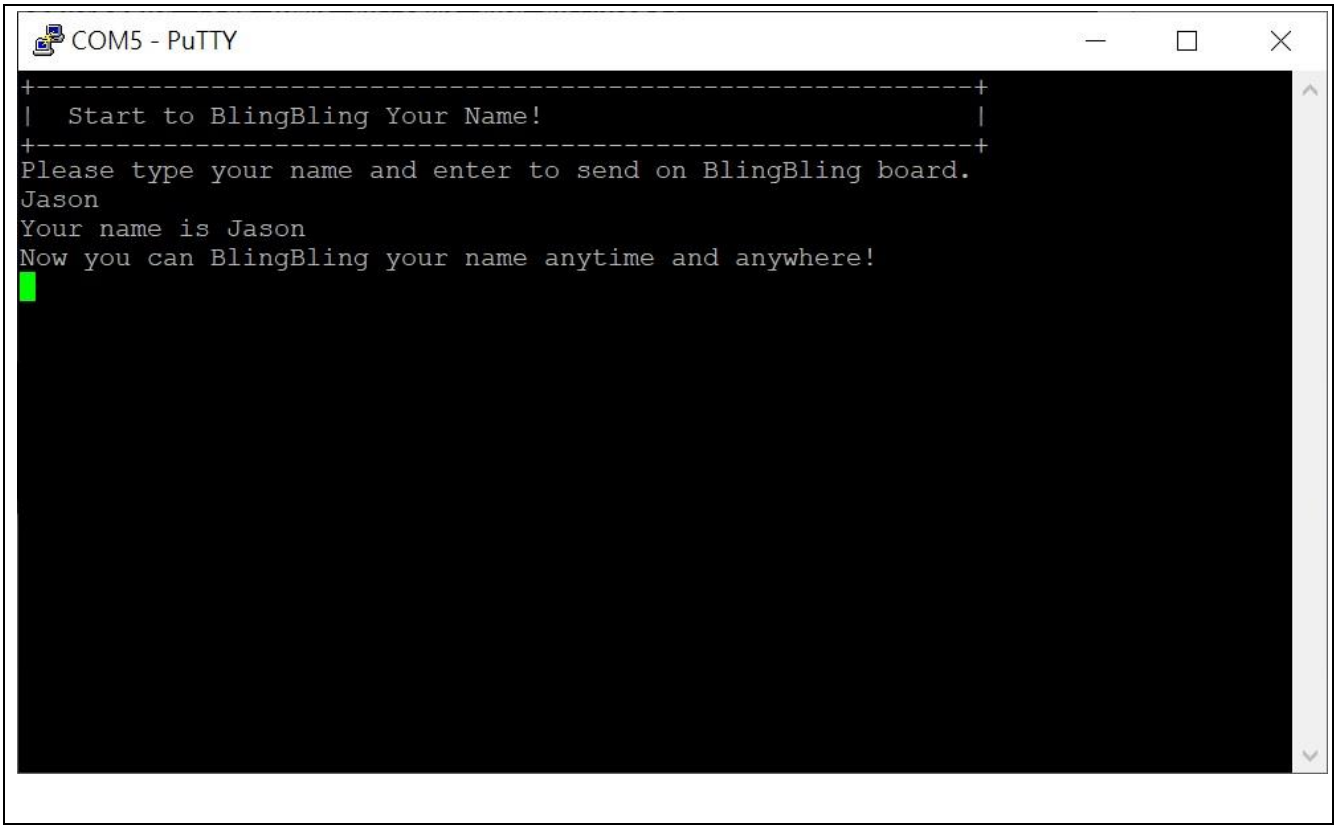

Figure 1-11 Demo Result of Terminal Software

1.2.3 Marquee Demo Result

ARGB LED marquee demo effect is shown in Figure 1-12.

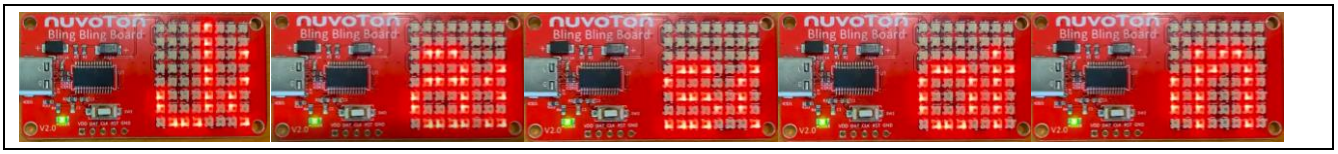

Figure 1-12 ARGB LED Marquee Demo Result

2. Code Description

2.1 Main Code Description

The steps are listed below:

1. Initialize the M0A23 series in *main.c,* including HIRC, UART0 and Timer0.
2. Use ARGB_init() to set GPIO and simulation ARGB timing sequence.
3. Use Read_DataFlash() to decide whether there is data in the Data Flash.
4. Decide whether there is a new string input in the While loop.

```
int32_t main(void)
{
    /* Init System, IP clock and multi-function I/O. */
    SYS_Init();

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

     /* Init ARGB */
    ARGB_init();

    TIMER_Delay(TIMER0, 1000000);

    Read_DataFlash();

    Convert_FlashData();

    printf("+------------------------------------------------------------+\n");
    printf("|  Start to ""BlingBling"" Your Name!                          |\n");
    printf("+------------------------------------------------------------+\n");
    printf("Please type your name and enter to send on BlingBling board.\n");

    while(1)
    {
        if(receiveData == FALSE)
        {
            put_string(String);
        }
        else
        {
            Write_DataFlash();

            receiveData = FALSE;
        }
     }
}
```

When Read_DataFlash() is called, it will read the Flash data at the specified address.

```
void Read_DataFlash()
{
    SYS_UnlockReg();

    /* Enable FMC ISP function */
```

```
    FMC_Open();

    /* Enable Data Flash and set base address. */
    set_data_flash_base(DATA_FLASH_BASE);

    FMC_ENABLE_AP_UPDATE();

    flash_read(DATA_FLASH_BASE, DATA_FLASH_BASE + RX_BUFFER_SIZE);

    FMC_DISABLE_AP_UPDATE();

    FMC_Close();
}
```

When Convert_FlashData() is called, determine whether the string character array has data.

```
void Convert_FlashData()
{
    uint8_t i = 0;

    if((uint8_t)flashData[0] != 0xFF)
    {
        for(i = 0; i < RX_BUFFER_SIZE; i++)
        {
            String[i] = 0;
        }

        for(i = 0; i < RX_BUFFER_SIZE; i++)
        {
            if((uint8_t)flashData[i] != 0xFF)
                String[i] = flashData[i];
            else
                break;
        }

        for(i = 0; i < RX_BUFFER_SIZE; i++)
        {
            flashData[i] = 0;
        }
    }
}
```

When put_string(char inputString[]) is called, the string will be stored in the marquee data array.

```
void put_string(char inputString[])
{
    int8_t count;

    for(count = 0; count < strlen(inputString); count++)
    {
        select_word(inputString[count]);

        put_char(count, strlen(inputString));
    }

    if(showArray_column >8)
        Run_column = showArray_column;
    else
        Run_column = 8;

    ARGB_showString();

    TIMER_Delay(TIMER0, 1000000);
```

```
    color++;

    if(color == 3)
    {
        color = 0;
    }

    showArray_row = 0;
    showArray_column = 0;
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M0A21_Series_BSP_CMSIS_V3.01.000
- IDE version
  - Keil uVersion 5.36

3.2 Hardware Requirements

- Circuit components
  - Bling Bling Board Ver2.0
  - ARGB2 LED
- Pin Connect

	VCC	↔	VCC	
	GPIO (PD.5)	↔	Din	
	GND	↔	GND	
	M0A23 Bling Bling Board		ARGB LED	

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_M0A23_GPIO_ARGB_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	
BlingBling	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 *BlingBling.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2023.12.15	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.