Using Timer-Triggered PDMA to Drive LEDs while Updating Flash

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code implements two segment LEDs to be driven by the timer-triggered PDMA for normal display even when the M032 series microcontroller is in the process of updating Flash.
BSP Version	M031_Series_BSP_CMSIS_V3.07.000
Hardware	NuMaker-M032SE V1.3 NuEdu-Basic01 V2.0

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

This example code implements two segment LEDs display to be driven even when the M032 series microcontroller is in the process of updating Flash.

In some products on the market, during the operation of the application, Flash Data needs to be updated. If the amount of data to be updated is large or the update is frequent, it will cause other program functions to be in a blocked state. For example, if an application needs to update Flash and also requires driving the segment LED display, when the time for updating Flash exceeds 20 ms, it will result in flickering because the refresh interval of LED exceeds 20 ms. This reference program can solve this problem.

1.1 Principle

The segment LED has one bit-select pins. When the bit-select pin is at a high level (or low level), the LED will light up; otherwise, it will be turned off. This reference design assumes that the bit-select pin is effective at a high level, meaning that a high level will light up the LED.

The segment LED is usually composed of 8 light-emitting diodes (A~G and DOT). The on or off state is achieved by controlling the level of each diode cathode. This reference design assumes that the diode cathode is effective at low levels, meaning that a low level will light up the corresponding segment. The principle of two segment LEDs and control is shown in Figure 1-1.

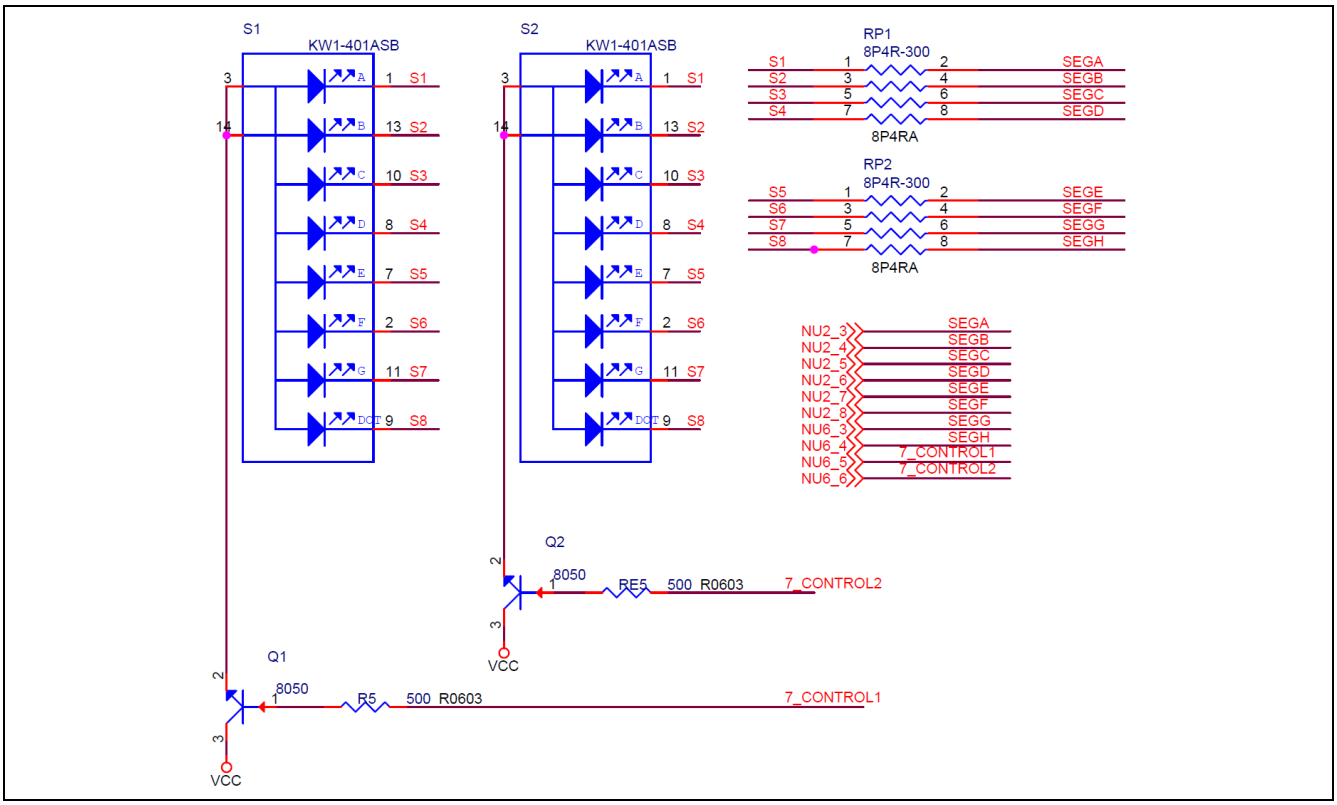

Figure 1-1 Principle of Two Segment LEDs and Control

By scanning every segment LED in turn and taking advantage of the human eye's persistence of vision effect, continuous display is achieved.

If the CPU needs to read the execution instruction and the Flash being updated are in the same bank, the CPU will be pending until the Flash update is completed before it can read the instruction. If the LED driver code is still executed through the CPU at this time, flashing will definitely occur. However, the operation of the PDMA controller does not require CPU intervention. If the relevant data for LED display is transmitted by the PMDA at this time, the problem of flashing can be avoided.

The Timer0 triggers the PDMA. The PDMA is set to the Scatter-Gather mode. The first descriptor table transmits the data for the first LED display, and the second descriptor table transmits the data for the second LED display. It is set to the Ping-Pong mode, meaning that after the first descriptor table finishes transmission, the second descriptor table is transmitted; after the second descriptor table finishes transmission, the first descriptor table is transmitted again, and this cycle continues.

1.2 Demo Result

The compilation and programming results of this example code are shown in Figure 1-2.

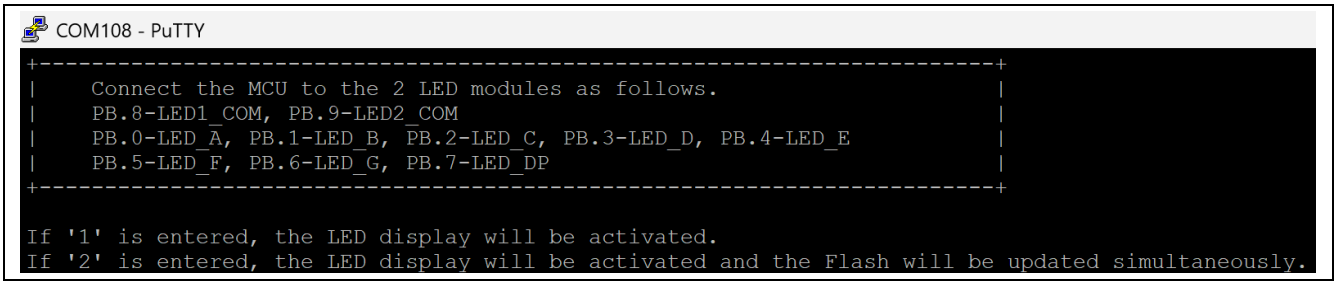

Figure 1-2 Compilation and Programming Results

Input the character "1" via the serial port, and the two LEDs will display "1" and "2" respectively. There will be a 900 ms delay, then display "3" and "4" respectively, followed by a 900 ms delay, then display "5" and "6" respectively, and so on, until "E" and "F" are displayed. The serial port output is shown in Figure 1-3.

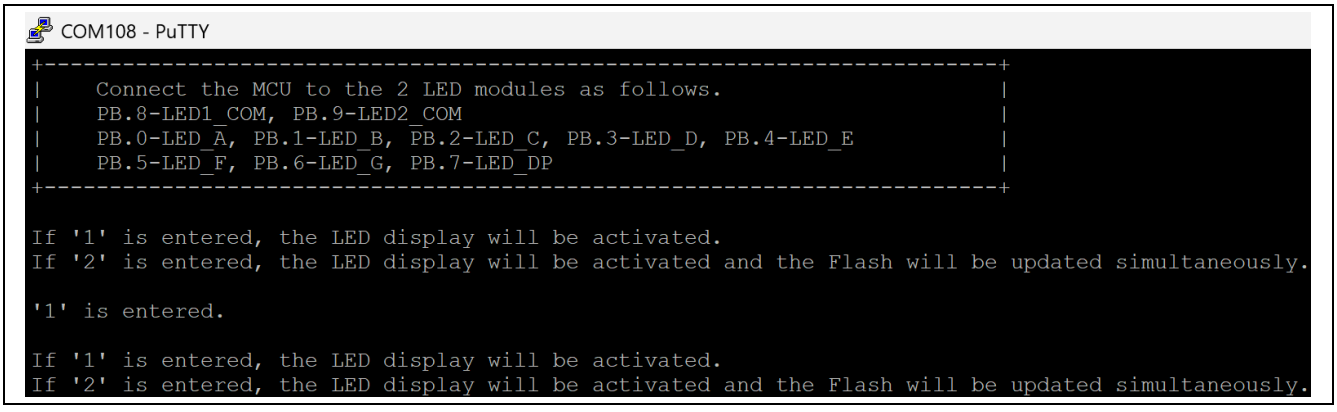

Figure 1-3 Input Character "1" via Serial Port

Then input the character "2", and the two LEDs will display "1" and "2" respectively. At the same time, the Flash erasing, writing and reading operations will be performed, taking approximately 516 ms. The waveform of PA0 pin is captured using the logic analyzer as shown in Figure 1-4; then "3" and "4" will be displayed respectively, and the Flash erasing, writing and reading operations will be performed simultaneously, taking approximately 516 ms; then "5" and "6" will be displayed respectively, and the Flash erasing, writing and reading operations will be performed simultaneously, taking approximately 516 ms; and so on, until "E" and "F" displayed. Serial port output is shown in Figure 1-5.

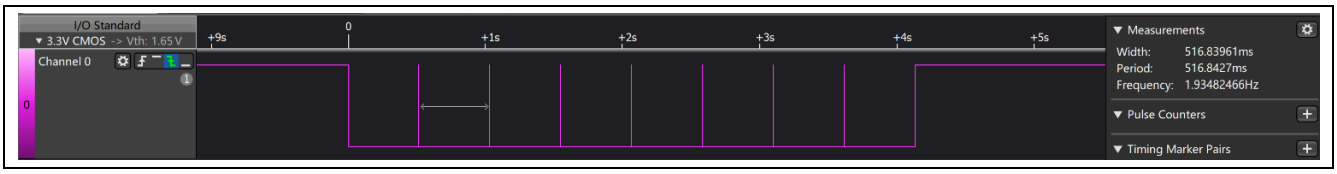

Figure 1-4 Flash Erasing, Writing and Reading Time

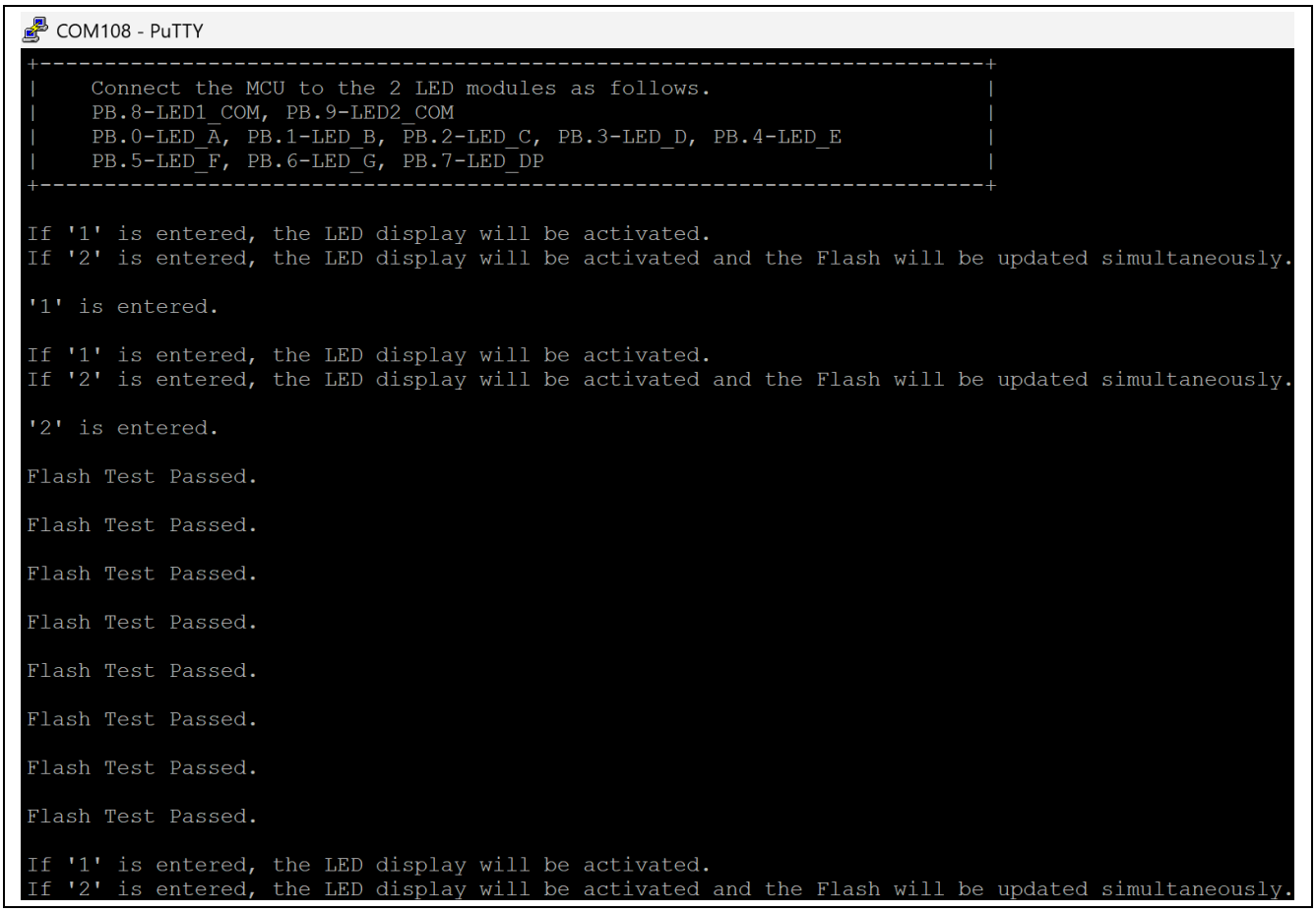

Figure 1-5 Input Character "2" via Serial Port

2. Code Description

Macro definitions and definitions for digital tube display numerical codes are located in *main.c*.

```
#define UART0_Baudrate  115200
#define TIMER0_Freq     1000            // Set Timer0 interrupt frequency
#define LED1_COM        0x01            // Select LED1
#define LED2_COM        0x02            // Select LED2
#define TEST_PATTERN    0x5A5A5A00      // Flash programming value
#define PDMA_CHN        1
#define APROM_TEST_BASE 0x00008000UL    // Flash programming start address
#define APROM_TEST_LEN  0x00002000UL    // Flash programming length

uint8_t const g_u8Value[16] =
{
    0xC0,  //"0"
    0xF9,  //"1"
    0xA4,  //"2"
    0xB0,  //"3"
    0x99,  //"4"
    0x92,  //"5"
    0x82,  //"6"
    0xF8,  //"7"
    0x80,  //"8"
    0x90,  //"9"
    0x88,  //"A"
    0x83,  //"B"
    0xC6,  //"C"
    0xA1,  //"D"
    0x86,  //"E"
    0x8E,  //"F"
};
```

The initialization program, including system initialization, UART initialization, display initial value initialization, PDMA initialization and timer initialization is located in *main.c*.

```
/* Variable initialization */
char u8Option;
uint16_t i;

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

/* Init UART0 to 115200-8n1 for print message */
UART_Open(UART0, UART0_Baudrate);

printf("+------------------------------------------------------------------------+ \n");
printf("|    Connect the MCU to the 2 LED modules as follows.                    | \n");
printf("|    PB.8-LED1_COM, PB.9-LED2_COM                                        | \n");
printf("|    PB.0-LED_A, PB.1-LED_B, PB.2-LED_C, PB.3-LED_D, PB.4-LED_E          | \n");
printf("|    PB.5-LED_F, PB.6-LED_G, PB.7-LED_DP                                 | \n");
printf("+------------------------------------------------------------------------+ \n");
```

```
/* Set PB[9:0] as output mode */
GPIO_SetMode(PB, 0x03FF, GPIO_MODE_OUTPUT);
/* Set PA0 as output mode */
GPIO_SetMode(PA, BIT0, GPIO_MODE_OUTPUT);
PA0  = 1;

/* Variable initialization */
u8Index = 0;
u8Times = 1;

/* Display value initialization */
g_u16LED1[0] = ((uint16_t)LED1_COM << 8) + g_u8Value[u8Index];
g_u16LED2[0] = ((uint16_t)LED2_COM << 8) + g_u8Value[u8Index + 1];

/* PDMA Init */
PDMA_Init();

/* Set Timer0 frequency to 1 kHz */
TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, TIMER0_Freq);

/* Select Timer0 as PDMA trigger source */
TIMER_SetTriggerTarget(TIMER0, TIMER_TRG_TO_PDMA);

/* Start Timer0 */
TIMER_Start(TIMER0);
```

PDMA initialization function is located in *main.c*.

```
void PDMA_Init(void)
{
    /* Open Channel 1 */
    PDMA_Open(PDMA, 1 << PDMA_CHN);

    PDMA_SetTransferMode(PDMA, PDMA_CHN, PDMA_TMR0, TRUE, (uint32_t)&DMA_DESC[0]);

    g_u32DMAConfig = \
                (0 << PDMA_DSCT_CTL_TXCNT_Pos) | /* Transfer count is 1 */ \
                PDMA_WIDTH_16 |  /* Transfer width is 16 bits(one word) */ \
                PDMA_SAR_FIX |   /* Source increment size is fixed(no increment) */ \
                PDMA_DAR_FIX |   /* Destination increment size is fixed(no increment */ \
                PDMA_REQ_BURST | /* Transfer type is burst transfer type */ \
                PDMA_BURST_1 |   /* Burst size is 1. No effect in single transfer type */ \
                PDMA_OP_SCATTER; /* Operation mode is scatter-gather mode */

    DMA_DESC[0].ctl = g_u32DMAConfig;
    /* Configure source address */
    DMA_DESC[0].src = (uint32_t)g_u16LED1;
    /* Configure destination address */
    DMA_DESC[0].dest = (uint32_t) &(PB->DOUT);
    /* Configure next descriptor table address */
    DMA_DESC[0].offset = (uint32_t)&DMA_DESC[1] - (PDMA->SCATBA);

    DMA_DESC[1].ctl = g_u32DMAConfig;
    /* Configure source address */
    DMA_DESC[1].src = (uint32_t)g_u16LED2;
    /* Configure destination address */
```

```
    DMA_DESC[1].dest = (uint32_t) &(PB->DOUT);
    /* Configure next descriptor table address */
    DMA_DESC[1].offset = (uint32_t)&DMA_DESC[0] - (PDMA->SCATBA);
}
```

Flash programming and checking function is located in *main.c*.

```
int32_t Flash_Programming_Verify(uint32_t u32StartAddr, uint32_t u32EndAddr, uint32_t
u32Pattern)
{
    uint32_t    u32Addr;

    for (u32Addr = u32StartAddr; u32Addr < u32EndAddr; u32Addr += FMC_FLASH_PAGE_SIZE)
    {
        /* Erase page */
        if (FMC_Erase(u32Addr) != 0)
        {
            printf("FMC_Erase address 0x%x failed!\n", u32Addr);
            return -1;
        }

        /* Write test pattern to fill the whole page */
        if (Fill_Data_Pattern(u32Addr, u32Addr + FMC_FLASH_PAGE_SIZE, u32Pattern) < 0)
        {
            printf("Failed to write page 0x%x!\n", u32Addr);
            return -1;
        }

        /* Verify if page contents are all equal to test pattern */
        if (Verify_Data(u32Addr, u32Addr + FMC_FLASH_PAGE_SIZE, u32Pattern) < 0)
        {
            printf("\nData verify failed!\n ");         /* error message */
            return -1;                                  /* Program verify failed */
        }
    }

    printf("\nFlash Test Passed.\n");                   /* information message */
    return 0;                                           /* Flash test passed */
}
```

Main loop code is located in *main.c*.

```
while (1)
{
    printf("\nIf '1' is entered, the LED display will be activated.\n");
    printf("If '2' is entered, the LED display will be activated and the Flash will be
     updated simultaneously.\n");

    u8Option = getchar();

    if (u8Option == '1')
    {
        printf("\n'1' is entered.\n");

        while (1)
        {
            /* Delay 900 ms */
```

```
            for (i = 0; i < 3; i++)
            {
                CLK_SysTickDelay(300000);
            }

            /* Update display data */
            u8Index = 2 * u8Times;
            g_u16LED1[0] = ((uint16_t)LED1_COM << 8) + g_u8Value[u8Index];
            g_u16LED2[0] = ((uint16_t)LED2_COM << 8) + g_u8Value[u8Index + 1];

            /* cycle count */
            u8Times++;

            if (u8Times >= (sizeof(g_u8Value)/2))
            {
                u8Times = 0;
                break;
            }
        }
    }

    if (u8Option == '2')
    {
        printf("\n'2' is entered.\n");

        while (1)
        {
            __NOP();
            SYS_UnlockReg();
            FMC_Open();

            FMC_ENABLE_AP_UPDATE();

            PA0 = 0; /* Testing the programming time */

            /* Program and check the data */
            if (Flash_Programming_Verify(APROM_TEST_BASE,
            (APROM_TEST_BASE+APROM_TEST_LEN), (TEST_PATTERN+u8Times)) < 0)
            {
                printf("\n\nAPROM test failed!\n");
            }

            PA0 = 1; /* Testing the programming time */

            FMC_DISABLE_AP_UPDATE();

            /* Disable FMC ISP function */
            FMC_Close();
            SYS_LockReg();

            /* Update display data */
            u8Index = 2*u8Times;
            g_u16LED1[0] = ((uint16_t)LED1_COM << 8) + g_u8Value[u8Index];
            g_u16LED2[0] = ((uint16_t)LED2_COM << 8) + g_u8Value[u8Index + 1];

            /* cycle count */
```

```
            u8Times++;

            if (u8Times >= (sizeof(g_u8Value)/2))
            {
                u8Times = 0;
                break;
            }
        }
    }
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M031_Series_BSP_CMSIS_V3.07.000
- IDE version
  - Keil uVision 5.41

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M032SE V1.3
  - NuEdu-Basic01 V2.0
- Pin Connect
  - Connect the corresponding pins of NuMaker-M032SE and NuEdu-Basic01 as shown in Figure 3-1.

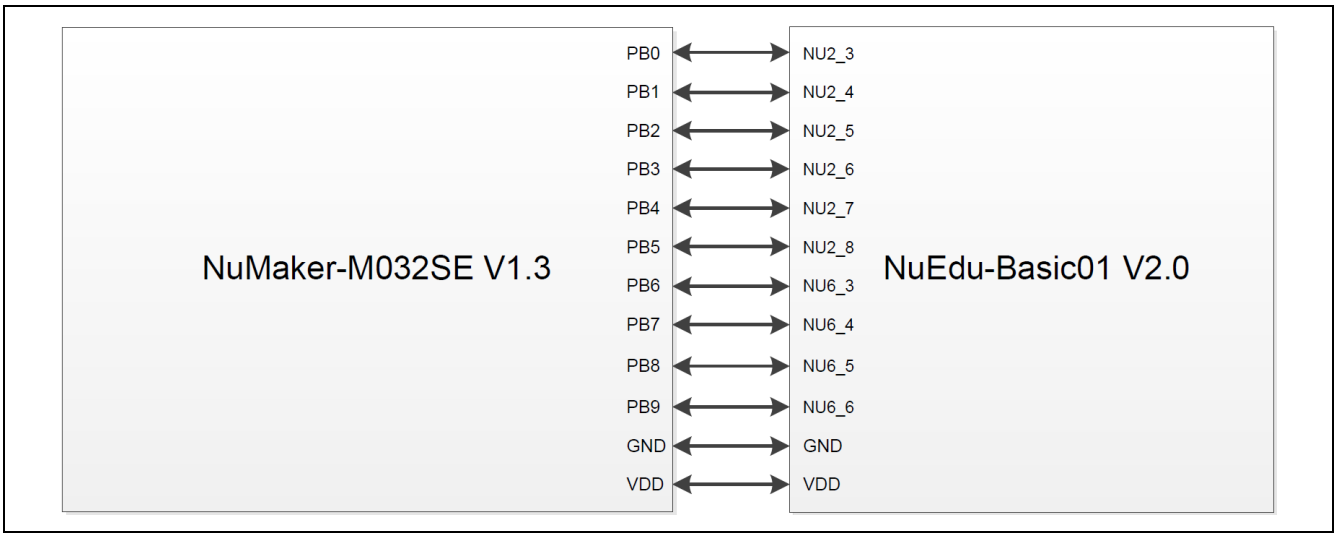

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_M032_Timer_Trigger_PDMA_To_Drive_LED_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	
Project	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 *Timer_Trigger_PDMA_To_Drive_LED.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2025.10.15	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.

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.