Using PDMA to Move Data from Flash to TFT Display

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses PDMA of the M031 series microcontroller (MCU) to move the picture data in SPI Flash to TFT display.
BSP Version	M031_Series_BSP_CMSIS_V3.06.000
Hardware	NuMaker-M031KI V1.0 NuTFT-SPI_320x240 Daughter Board_V1.3

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 uses the PDMA function of SPI and USPI in M031KIAAE to move the picture data in SPI Flash to TFT display.

First, the picture data is written to Flash through SPI, and then the PDMA function of SPI RX is used to transfer the data in Flash to USPI TX, and the TFT display is driven by TX.

1.1 Principle

The 320x240 TFT is displayed in RGB565 mode and requires 153600 (320x240x2) bytes of data to display whole screen picture. The storage capacity of the Flash (W25Q16) is 2 MB. After power-on, the image data is saved to the Flash. The M031KIAAE communicates with the Flash via SPI and the TFT control chip (ILI9341) via USPI. Both SPI and USPI support the PDMA function. By enabling the SPI RX PDMA function, the image data in Flash is directly transmitted to USPI TX, and the TFT display is driven by TX. In addition to setting the Flash read data start address and the TFT display initial location, other data transmission is completed by the PDMA, thereby improving the CPU efficiency.

1.2 Demo Result

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

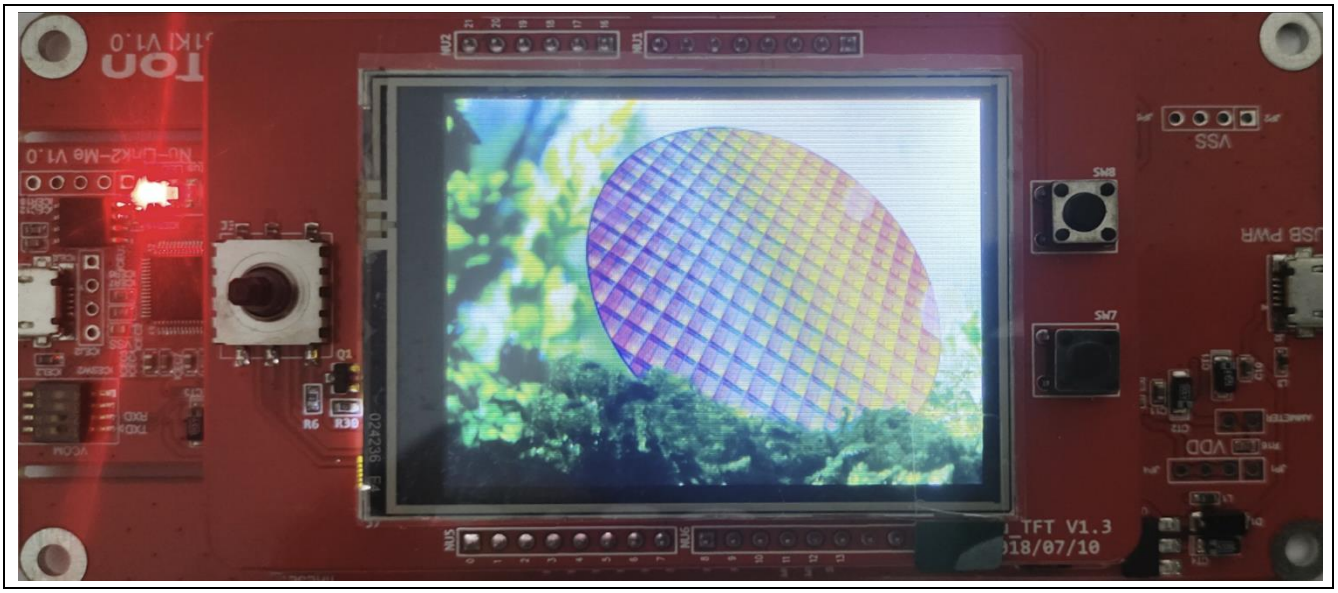


Figure 1-1 Compilation and Programming Results

The message printed by the serial port is shown Figure 1-2.

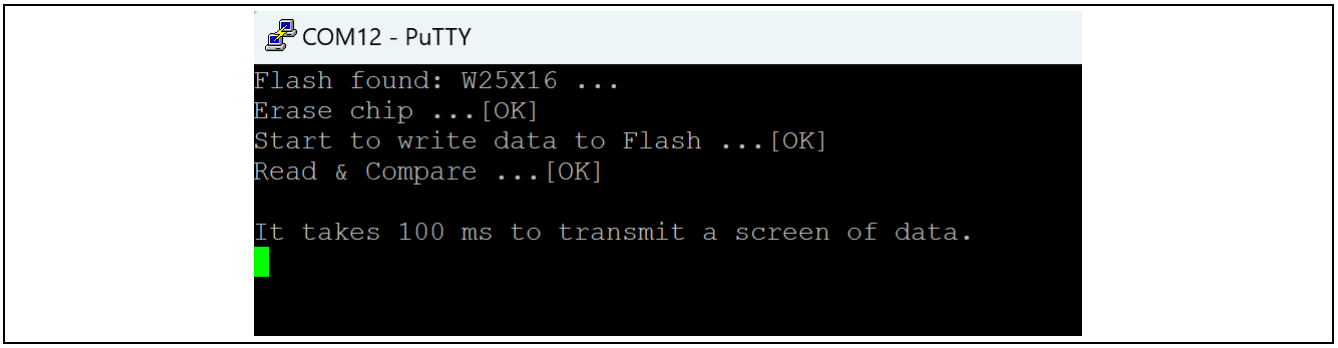


Figure 1-2 Message Printed by Serial Port

2. Code Description

Image data array definition. The code is located in *Image.c*.

```
const uint8_t u8Image[DATA_PER_SCREEN] =
{
    0XF7,0XBE,0XF7,0XDE,0XF7,0XDE,0XF7,0XDE,0XF7,0XDE,0XF7,0XBE,0XF7,0XBE,0XF7,0XBE,
    0XEF,0X9D,0XE7,0X7C,0XE7,0X7C,0XE7,0X7C,0XE7,0X7C,0XE7,0X7C,0XE7,0X7C,0XE7,0X7C,
    0XE7,0X7C,0XE7,0X7C,0XE7,0X7C,0XE7,0X9C,0XE7,0X9D,0XE7,0X9D,0XE7,0X9C,0XE7,0X9C,
    0XE7,0X7C,0XE7,0X9C,0XE7,0X9C,0XE7,0X9D,0XE7,0X9D,0XE7,0X9D,0XE7,0X9D,0XEF,0X9D,
    0XEF,0X9D,0XEF,0X9E,0XEF,0XBD,0XEF,0XBE,0XEF,0XBE,0XEF,0XBD,0XEF,0XBD,0XEF,0X9D,
    ……
    0X3B,0X67,0X32,0XC6,0X2A,0X24,0X19,0X83,0X19,0X83,0X21,0XA4,0X2A,0X05,0X2A,0X25,
    0X21,0XE4,0X21,0XC3,0X21,0XC3,0X21,0X83,0X19,0X43,0X21,0X63,0X2A,0X04,0X32,0X45,
    0X2A,0X04,0X32,0X65,0X3A,0XC7,0X2A,0X25,0X19,0X83,0X21,0XC4,0X29,0XE5,0X21,0XA4,
    0X19,0X43,0X19,0X43,0X21,0X83,0X21,0X63,0X19,0X22,0X19,0X02,0X10,0XE2,0X10,0XC2,
    0X10,0XC2,0X10,0XC2,0X10,0XC2,0X10,0XC2,0X10,0XC2,0X10,0XC2,0X10,0XC2,0X10,0XA2,
};
```

Image data array size definition. The code is located in *Image.h*.

```
#define DATA_PER_SCREEN   (153600UL)
extern const uint8_t u8Image[DATA_PER_SCREEN];
```

The main function includes the initialization of the system, UART0, USCI_SPI0, PWM, SPI0, TFT driver chip, SPI Flash, Timer0, and finally executing of the main loop. The code is located at the main function.

```
    int main()
    {
        /* System initialization */
        SYS_Init();
        /* Initial UART0 to 115200-8n1 for print message */
        UART_Open(UART0, 115200);
        /* Initial USCI_SPI0 */
        USCI_SPI_Init();
        /* Initial PWM for backlight brightness control */
        PWM_Init();
        /* Initial SPI0 */
        SPI_Init();
        /* Initial ILI9341 LCD Panel Hardware */
        Drv_ILI9341_LCD_Init();
        /* Identify flash type, erase flash, write picture datas to flash,
        check if write successful */
        SpiFlash_Setting();
        /* Initial Timer0 */
        Timer0_Init();
        while (1)
        {
            /* Data read from FLASH is sent to TFT via PDMA */
            Copy_FlashData_To_TFT_WithPDMA();
        }
    }
```

The main loop function is used to move data in Flash to TFT display. After data of one screen is transmitted, serial port will output the time taken for transmission. The code is located at Copy_FlashData_To_TFT_WithPDMA function.

```
/* Data read from FLASH is sent to TFT via PDMA */
void Copy_FlashData_To_TFT_WithPDMA(void)
{
    uint32_t u32RegValue, u32Abort;

    printf("\nStart refresh TFT.... \n");
    /* Data Width is swtich to 8 Bits */
    Switch_DataWidth_Is_Byte(TRUE);
    /* Reset LCD display position */
    LCD_Set_Pos();
    /* Reset read start address */
    SpiFlash_SetReadAddress(0x000000);
    /* Data Width is swtich to 16 Bits */
    Switch_DataWidth_Is_Byte(FALSE);
    /* Init PDMA */
    PDMA_Init();
    /* CS = 0 : select chip */
    USPI_CS_CLR;
    /* DC = 1 : write data */
    LCM_DC_SET;
    /* Enable SPI RX PDMA function */
    SPI_TRIGGER_RX_PDMA(SPI0);
    while (1)
    {
        /* Get interrupt status */
        u32RegValue = PDMA_GET_INT_STATUS(PDMA);
        /* Check the PDMA transfer done interrupt flag */
        if (u32RegValue &PDMA_INTSTS_TDIF_Msk)
        {
            /* Check the PDMA transfer done flags */
            if ((PDMA_GET_TD_STS(PDMA) & (0x01 << SPI_MASTER_RX_DMA_CH)) ==
                    (0x01 << SPI_MASTER_RX_DMA_CH))
            {
                /* Clear the PDMA transfer done flags */
                PDMA_CLR_TD_FLAG(PDMA, (0x01 << SPI_MASTER_RX_DMA_CH));
                /* Enable SPI RX PDMA function */
                SPI_TRIGGER_RX_PDMA(SPI0);
                u8TransferCnt++;
                if (u8TransferCnt >= TRANSFER_TIMES)
                {
                    u8TransferCnt = 0;
                    printf("\nIt takes %d ms to transmit a screen of data. \n",
                           (u8TimerCnt*10));
                    /* Disable Timer0 NVIC */
                    NVIC_DisableIRQ(TMR0_IRQn);
                    while (1);
                }
            }
        }
        /* Check the DMA transfer abort interrupt flag */
        if (u32RegValue &PDMA_INTSTS_ABTIF_Msk)
        {
            /* Get the target abort flag */
            u32Abort = PDMA_GET_ABORT_STS(PDMA);
            /* Clear the target abort flag */
            PDMA_CLR_ABORT_FLAG(PDMA, u32Abort);
```

```
            printf("\nabort happen\n");
        }

        /* Check the DMA time-out interrupt flag */
        if (u32RegValue &(PDMA_INTSTS_REQTOF0_Msk | PDMA_INTSTS_REQTOF1_Msk))
        {
            /* Clear the time-out flag */
            PDMA->INTSTS = u32RegValue &(PDMA_INTSTS_REQTOF0_Msk |
                                         PDMA_INTSTS_REQTOF1_Msk);
            printf("\ntime-out happen\n");
        }
    }
}
```

Set the transmission bit width to 8 bits when setting the command, and set the transmission bit width to 16 bits when transferring data. A transmission bit width of 16 bits is more efficient than a transmission bit width of 8 bits. The code is located at the Switch_DataWidth_Is_Byte function.

```
void Switch_DataWidth_Is_Byte(_Bool bFlag)
{
    if (bFlag)
    {
        /* USPI0 Transfer Data width is 8 Bits */
        USPI0->LINECTL &= ~USPI_LINECTL_DWIDTH_Msk;
        USPI0->LINECTL |= (0x08 << USPI_LINECTL_DWIDTH_Pos);
        /* SPI0 Transfer Data width is 8 Bits*/
        SPI0->CTL &= ~SPI_CTL_DWIDTH_Msk;
        SPI0->CTL |= (0x08 << SPI_CTL_DWIDTH_Pos);
    }
    else
    {
        /* USPI0 Transfer Data width is 16 Bits */
        USPI0->LINECTL &= ~USPI_LINECTL_DWIDTH_Msk;
        USPI0->LINECTL |= (0 << USPI_LINECTL_DWIDTH_Pos);
        /* SPI0 Transfer Data width is 16 Bits */
        SPI0->CTL &= ~SPI_CTL_DWIDTH_Msk;
        SPI0->CTL |= (0x10 << SPI_CTL_DWIDTH_Pos);
    }
}
```

The Timer0 initialization function is set to interrupt every 10 milliseconds and the counter is initialized to 0. The code is located at Switch_DataWidth_Is_Byte function. The Timer0 interrupt function adds 1 to the counter once an interruption occurs. The code is located at the TMR0_IRQHandler function.

```
void Timer0_Init(void)
{
    /* Open Timer0 in periodic mode, enable interrupt
    and 1 interrupt tick per 10 millisecond */
    TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, 100);
    TIMER_EnableInt(TIMER0);
    /* Start Timer0 counting */
    TIMER_Start(TIMER0);
    /* Clear timer counter */
    u8TimerCnt = 0;
    /* Enable Timer0 NVIC */
    NVIC_EnableIRQ(TMR0_IRQn);
}

void TMR0_IRQHandler(void)
{
    if (TIMER_GetIntFlag(TIMER0) == 1)
```

```
    {
        /* Clear Timer0 time-out interrupt flag */
        TIMER_ClearIntFlag(TIMER0);
        /* Timer counter plus one */
        u8TimerCnt++;
    }
}
```

Because TFT is displayed in RGB565 mode, a pixel is represented by 16 bits of data, and the basic unit of transmission between MCU and TFT control chip is also 16 bits, so 76800x16 bits of data are required to transmit a screen picture, that is, 320x240 pixels. Because the transmission number of a PDMA table is defined by 16 bits, the maximum transmission data is 65536 x16 bits, so the transmission of a screen picture needs to be divided into two transfers, respectively transmitting 38400 x16 bits of data. The PDMA operation mode is set to Scatter-Gather. The number of transfers is set to 38400. The transmission bit width is set to 16 bits. The transmission type is set to signal. The source address is fixed to the SPI0 RX register and the destination address is fixed to the USPI0 TX register. The code is located at the PDMA_Init function.

```
void PDMA_Init(void)
{
    /* Descriptor table 1 configuration */
    DMA_DESC[0].ctl = ((TRANSFER_LEN - 1) << PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_16 |
                      PDMA_SAR_FIX | PDMA_DAR_FIX | PDMA_REQ_SINGLE | PDMA_OP_SCATTER;
    DMA_DESC[0].src = (uint32_t) &SPI0->RX;
    DMA_DESC[0].dest = (uint32_t) &USPI0->TXDAT;
    DMA_DESC[0].offset = (uint32_t) &DMA_DESC[1] - (PDMA->SCATBA);
    /* Descriptor table 2 configuration */
    DMA_DESC[1].ctl = ((TRANSFER_LEN - 1) << PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_16 |
                      PDMA_SAR_FIX | PDMA_DAR_FIX | PDMA_REQ_SINGLE | PDMA_OP_BASIC;
    DMA_DESC[1].src = (uint32_t) &SPI0->RX;
    DMA_DESC[1].dest = (uint32_t) &USPI0->TXDAT;
    DMA_DESC[1].offset = 0;
    /* Open PDMA channel 1 for SPI RX */
    PDMA_Open(PDMA, (1 << SPI_MASTER_RX_DMA_CH));
    /* Configure PDMA transfer mode */
    PDMA_SetTransferMode(PDMA, SPI_MASTER_RX_DMA_CH, PDMA_SPI0_RX, 1, (uint32_t)
                         &DMA_DESC[0]);
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M031_Series_BSP_CMSIS_V3.06.000
- IDE version
  - Keil uVersion 5.36

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M031KI V1.0
  - NuTFT-SPI_320x240 Daughter Board_V1.3
- Pin Connect
  - Flash and TFT Control Chip are located on NuTFT-SPI_320x240 Daughter Board.
  - M031KIAAE is located on NuMaker-M031KI board.

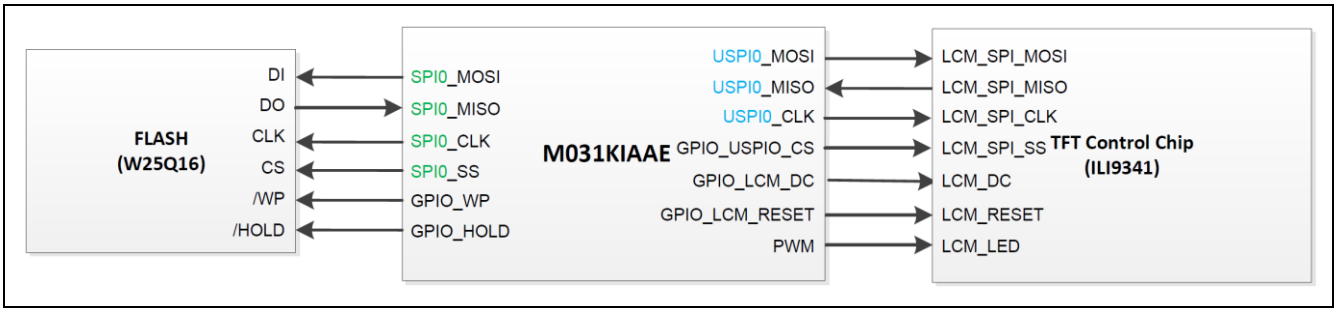


Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_ M031_PDMA_FLASH_TO_TFT_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 *M031_PDMA_FLASH_TO_TFT.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2024.07.10	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.