

在 TFT 屏显示动画并叠加字符

NuMicro® 32 位系列微控制器范例代码介绍

文件信息

应用简述	使用 QSPI Quad 模式搭配 PDMA 从 FLASH 中读取动画数据，然后叠加字符数据，最后利用 SPI 接口驱动 TFT 控制器进行显示。
BSP 版本	M3331_Series_BSP_CMSIS_V3.00.002
开发平台	NuMaker-M3334KI V2.0 & NuFun-M3334KI 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. 概述

本范例代码利用 QSPI Quad 模式搭配 PDMA，依次从 FLASH 中读取由动画生成的多张图片数据，然后再分别叠加字符数据，最后利用 SPI 搭配 PDMA，依次把数据发送到 TFT 控制器进行显示。

1.1 原理

320x240 TFT 以 RGB565 模式显示，一个像素需要 16 Bit 数据，满屏显示需要 153,600 (320 x 240 x 2) 个字节数据。评估板上的 FLASH (W25Q32) 存储容量为 4 Mbytes，最多可存储 27 张图片数据。执行「Tool」文件夹下「*ImagesToBin_Uncompress.exe*」（该 Tool 仅供学习参考），将当前路径下 27 张分辨率为 320 x 240，位深度为 16 的 bmp 格式图片转换为 RGB565 格式数据（高位在前），并依序合并，最终在当前路径下生成一个「*SPI_Flash.bin*」文件。每张图片的扫描方式如图 1-1 所示。



图 1-1 图片扫描示意图

利用 ICP Programming Tool，将 Bin 文件编程到 NuFun-M3334KI V2.0 板上的 FLASH 芯片。操作如下：

1. 双击「Tool」文件夹下「*NuMicro ICP Programming Tool 3.22.7959a.exe*」进行工具安装（安装后在安装目录下才会有「*SPI_Flash.ini*」文件）。
2. 在安装目录下找到「*SPI_Flash.ini*」并打开，参考图 1-2 进行修改（根据图示步骤 2 设定待烧录 Bin 文件路径；烧录完成后，根据图示步骤 3 设定从 FLASH 中读出用于校验的 Bin 文件路径）。

```
[Version]
Nu_LinkVersion=V1.00
[M031]
ProgramAlgorithm=FlashAlgorithm_M031_SPI.a
ProgramOffset=0x0
ProgramFile=C:\Program Files (x86)\Nuvoton Tools\ICPTool\SPI_Flash.bin
OutputFile=C:\Program Files (x86)\Nuvoton Tools\ICPTool\SPI_Flash_output.bin
[M3331]
ProgramAlgorithm=FlashAlgorithm_M3331_SPI.a
ProgramOffset=0x0
ProgramFile=C:\Program Files (x86)\Nuvoton Tools\ICPTool\SPI_Flash.bin
OutputFile=C:\Program Files (x86)\Nuvoton Tools\ICPTool\SPI_Flash_output.bin
```

1. Set the starting address for programming
2. Set the source file path
3. Set the path for the verification file

图 1-2 编程参数设定

3. 使用 USB 线连接 EVB 与 PC，并参考图 1-3 进行操作（在进入步骤 1 的过程中，若提示固件与驱动版本不一致，则根据提示进行固件升级）。注意在「Programming」选项中因为没有 Internal Flash 需要烧录，因此必须勾选「Chip Setting」。

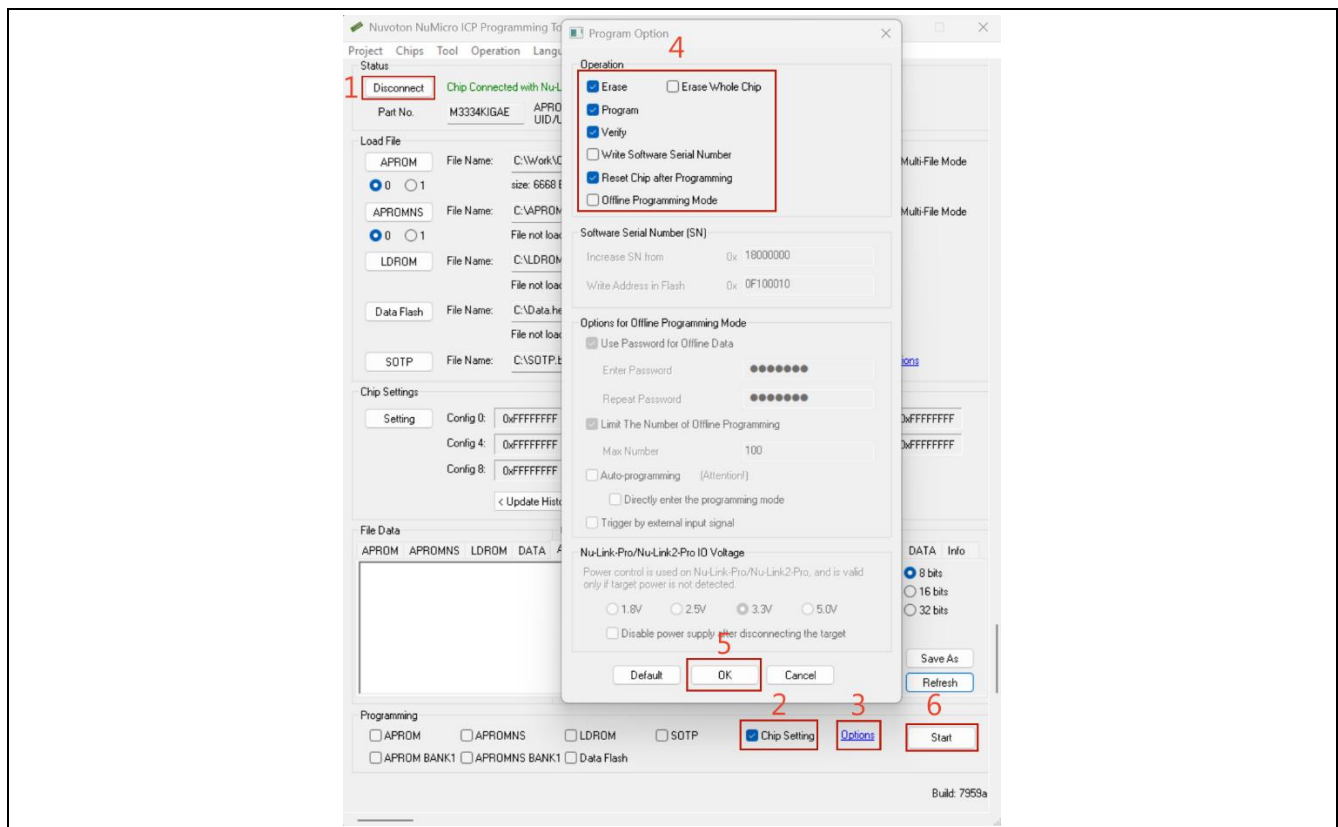


图 1-3 编程选项设定

MCU 利用 QSPI 接口与外部 FLASH 进行通信。为了提高数据传输速率，启用了 Quad 模式，并使用了 RX PDMA，数据宽度为 32 Bit。MCU 利用 SPI 接口与 TFT 控制器相连，在此使用了 TX PDMA 和双传输缓冲区，数据宽度为 16 Bit。

MCU 上电后先初始化相关接口，然后依序读取外部 FLASH 中每张图片数据，并分别叠加字符数据，最后依次发送到 TFT 控制器进行动画显示。

此示例的整体软件流程如下：

1. 上电后初始化 MCU 系统时钟、GPIO、UART、Timer0、QSPI、SPI、以及外部 LCD 等。
2. MCU 将外部 FLASH 配置为 QSPI Quad 模式，并启用接收端 PDMA 以高效读取图像数据。
3. 动画帧数据按顺序从外部 FLASH 中读取并传入显示缓冲区。
4. 诸如实时帧率之类的字符数据会逐行与图像数据在显示缓冲区中进行合并。
5. 合并后的帧数据利用 SPI 接口使用 TX PDMA 并结合双缓冲机制传输至 TFT 控制器。
6. Timer0 用于测量显示屏的刷新频率，该频率的计算结果会通过 UART 接口进行输出。
7. 上述过程 3~6 会针对每一帧重复进行，从而在 TFT 显示屏上生成连续的动画效果。

1.2 执行结果

本范例代码执行编译和烧录后，会显示动画并叠加字符，其中一个画面如图 1-4 所示。

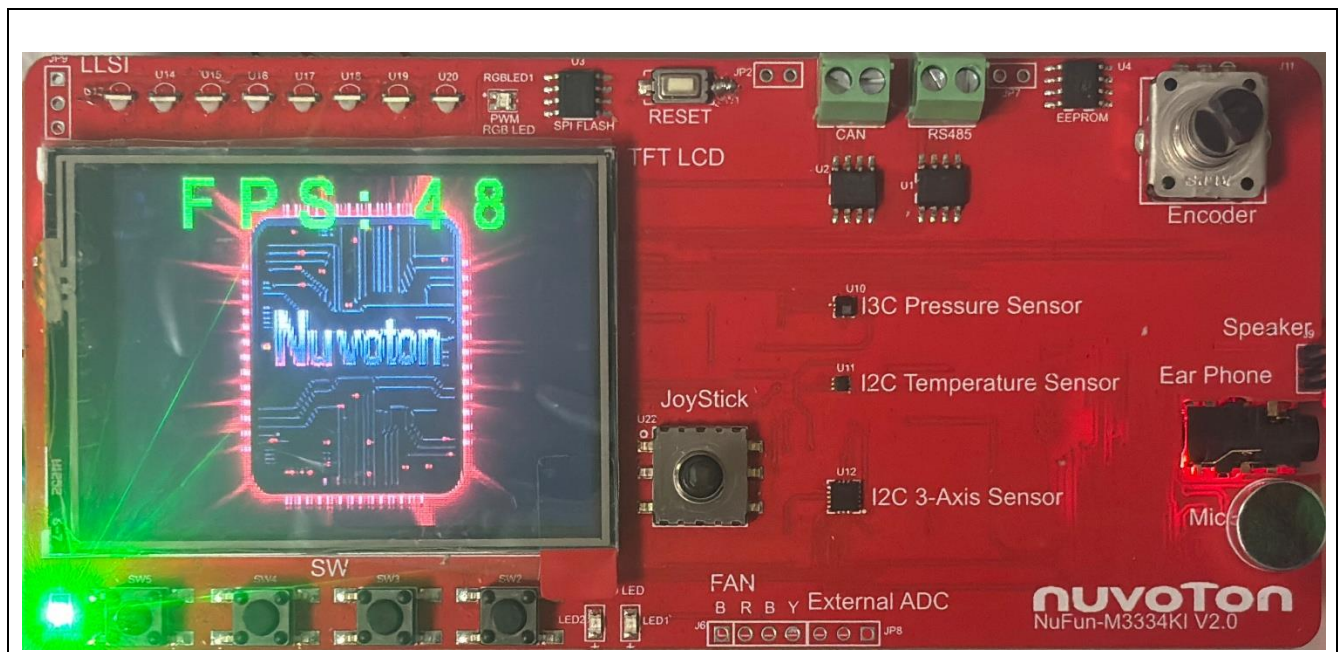


图 1-4 执行编译和烧录结果

同时 PC UART 会打印实时屏幕刷新率，如下：

FPS : 48.
FPS : 48.
FPS : 48.
FPS : 48.
FPS : 48.
FPS : 48.
FPS : 48.
FPS : 48.
FPS : 48.
FPS : 48.
FPS : 48.
FPS : 48.

2. 代码介绍

以下将介绍此示例的关键代码部分。除非另有说明，所有代码均位于 *main.c* 文件中。

以下代码定义了此示例中所使用的宏常量。

```
#define PDMA_FOR_LCD_CH0    0
#define PDMA_FOR_FLASH_CH1 1
#define STRING_WIDTH      (5*8) /* String width */
#define STRING_HEIGHT     46    /* String height */
#define LCD_WIDTH         320
#define LCD_HEIGHT        240
#define PIC_WIDTH          320   /* Picture width */
#define PIC_HEIGHT        240   /* Picture height */
#define LV_HOR_RES_MAX     320
#define LV_VER_RES_MAX     240
#define FLASH_QSPI_CLK_FREQ 60000000 /* QSPI clock speed */
#define LCD_SPI_CLK_FREQ   60000000  /* SPI clock speed */
#define COLOR_WHITE       0xFFFF
#define COLOR_BLACK       0x0000
#define COLOR_BLUE        0x001F
#define COLOR_BLUE2       0x051F
#define COLOR_RED         0xF800
#define COLOR_MAGENTA     0xF81F
#define COLOR_GREEN       0x07E0
#define COLOR_CYAN        0x7FFF
#define COLOR_YELLOW      0xFFE0
#define LCD_RST           PD14
#define LCD_DC             PH10
#define LCD_EN             PH11
#define LCD_DC_CLR()      PH10 = 0
#define LCD_DC_SET()      PH10 = 1
#define QSPI_FLASH_PORT   QSPI0
#define SPI_LCD_PORT      SPI1
```

系统初始化包括 UART、Timer0、QSPI、SPI 和 LCD 的配置。在初始化过程中，受保护的寄存器会被解锁，系统时钟和 I/O 会被配置，LCD 会以预设颜色清零。Timer0 被配置用于测量 LCD 的刷新率。设置 PDMA 中断优先级为最高，确保数据高效传输。

```
uint8_t i;
uint32_t Flash_Addr;
uint16_t Rate = 0;

/* Unlock protected registers */
SYS_UnlockReg();

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

/* Configure PD14[LCD_RST] PH10[LCD_DC] PH11[LCD_EN] as push-pull output */
GPIO_SetMode(PD, BIT14, GPIO_MODE_OUTPUT);
```

```

GPIO_SetMode(PH, BIT10 | BIT11, GPIO_MODE_OUTPUT);

/* Configure UART0: 115200, 8-bit word, no parity bit, 1 stop bit. */
UART_Open(UART0, 115200);

/* Timer0 initialization. The main purpose is to measure the LCD refresh rate. */
Timer0_Init();

/* QSPI0 initialization, used for driving the Flash */
QSPI_FLASH_Init();

/* SPI1 initialization, used for driving the LCD */
SPI_LCD_Init();

/* LCD initialization */
LCD_Init();

/* Set full-screen display with green color. */
LCD_Color_Set(0, 0, (LCD_WIDTH - 1), (LCD_HEIGHT - 1), COLOR_GREEN);

/* Set interrupt priority */
NVIC_SetPriority(PDMA0_IRQn, 0);
NVIC_SetPriority(TMR0_IRQn, 1);

/* Delay 200ms */
CLK_SysTickLongDelay(200000);

```

主循环负责显示动画并叠加字符数据。

对于每一帧：

重置刷新计时器计数器。

从外部 FLASH 中读取图像数据，并将其与字符数据合并。

通过 SPI 结合 PDMA 将合并后的数据发送至 LCD。

计算刷新频率，并通过 UART 输出显示。

```

/* Set the starting address of Flash */
Flash_Addr = 0;

for (i = 0; i < 27; i++)
{
    /* Reset the time counter */
    u8TimerCnt = 0;

    /* Display the image and superimpose the characters */
    Flash_Addr = LCD_Display_Picture_And_String(Flash_Addr,
                                                0, 0,
                                                PIC_WIDTH, PIC_HEIGHT,
                                                COLOR_BLACK, Rate,
                                                40, 0,

```

```

                                STRING_WIDTH, STRING_HEIGHT,
                                COLOR_RED);

    /* Calculate the refresh rate */
    Rate = (uint16_t)(10000.0/(u8TimerCnt*1));

    /* Print the refresh rate */
    printf("FPS = %d.\n", Rate);
}

```

函数 `LCD_Display_Picture_And_String()` 从外部 FLASH 中指定地址读取图像数据，将其显示在预定义的 LCD 区域内，并叠加诸如帧率值之类的文本字符。

```

/* Read the image data from a specific address in the flash memory and display it in a
specific area of the image, while also overlaying numbers on the image. */
uint32_t LCD_Display_Picture_And_String(uint32_t Addr, /* Flash address */
    uint16_t Pic_x, uint16_t Pic_y, /* picture display coordinates */
    uint16_t Pic_Width, uint16_t Pic_Height, /* picture size */
    uint16_t String_Backcolor, uint16_t Rate, /* string background color, display numerical */
    uint16_t String_x, uint16_t String_y, /* string display coordinates */
    uint16_t String_Width, uint16_t String_Height, /* string size */
    uint16_t String_Color) /* string color */
{
    uint32_t i;

    /* Initialize the pointer */
    uint16_t *Display_Buffer = LCD_Display_Buffer0;
    uint8_t Y_Number = 0;

    if (Addr == 0)
    {
        QSPI_Close(QSPI0);

        while ((QSPI0->STATUS & QSPI_STATUS_SPIENSTS_Msk) != 0)
        {}

        /* Disable MCU quad I/O mode. */
        QSPI_DISABLE_QUAD_MODE(QSPI0);

        /* Set the QSPI data length to 8 */
        QSPI_SET_DATA_WIDTH(QSPI0, 8);

        QSPI0->CTL |= (QSPI_CTL_SPIEN_Msk);

        /* Enable Flash quad mode */
        SpiFlash_Enable_QSPI_Quad();

        /* Set the starting address for Flash reading */
        SpiFlash_SetReadAddress(Addr);

        QSPI_Close(QSPI0);

        while ((QSPI0->STATUS & QSPI_STATUS_SPIENSTS_Msk) != 0)
        {}
    }
}

```

```

    /* Set the QSPI data length to 32 */
    QSPI_SET_DATA_WIDTH(QSPI0, 32);

    QSPI0->CTL |= (QSPI_CTL_SPIEN_Msk);
}

SPI_Close(SPI_LCD_PORT);

while ((SPI_LCD_PORT->STATUS & SPI_STATUS_SPIENSTS_Msk) != 0)
{}

/* Set the SPI data width to 8 */
SPI_SET_DATA_WIDTH(SPI_LCD_PORT, 8);

SPI_LCD_PORT->CTL |= (SPI_CTL_SPIEN_Msk);

/* Set the display area of the LCD */
LCD_Display_Zone_Set(Pic_x, Pic_y, (Pic_x + Pic_Width - 1), (Pic_y + Pic_Height - 1));

SPI_Close(SPI_LCD_PORT);

while ((SPI_LCD_PORT->STATUS & SPI_STATUS_SPIENSTS_Msk) != 0)
{}

/* Set the SPI data width to 16 */
SPI_SET_DATA_WIDTH(SPI_LCD_PORT, 16);

SPI_LCD_PORT->CTL |= (SPI_CTL_SPIEN_Msk);

/* Initialization flag */
PDMA_For_LCD_Send_End_Flag = 1;

for (i = 0; i < Pic_Height; i++)
{
    /* QSPI master PDMA RX Initialization */
    PDMA_For_FLASH_Trigger((uint8_t *)Display_Buffer, LCD_WIDTH / 2);

    /* Clear RX Buffer */
    QSPI_ClearRxFIFO(QSPI_FLASH_PORT);

    /* Enable RX PDMA transfer */
    QSPI_TRIGGER_RX_PDMA(QSPI0);

    /* Initialization flag */
    PDMA_For_Flash_Read_End_Flag = 0;

    /* Waiting for the reading to be completed */
    while (PDMA_For_Flash_Read_End_Flag == 0);

    /* Finished reading */
    PDMA_For_Flash_Read_End_Flag = 0;

    /* Merge the image data with the character data */
    if ((Y_Number >= String_y) && (Y_Number < (String_y + String_Height)))
    {
        /* Merge "FPS:" character data with the image data. */

```

```

String_Line_Fill(Display_Buffer, (uint8_t *)(&XL_Word[10]),
String_x, String_Width, (Y_Number - String_y),
String_Color, String_Backcolor, 0);
String_Line_Fill(Display_Buffer, (uint8_t *)(&XL_Word[11]),
String_x + String_Width * 1, String_Width, (Y_Number - String_y),
String_Color, String_Backcolor, 0);
String_Line_Fill(Display_Buffer, (uint8_t *)(&XL_Word[12]),
String_x + String_Width * 2, String_Width, (Y_Number - String_y),
String_Color, String_Backcolor, 0);
String_Line_Fill(Display_Buffer, (uint8_t *)(&XL_Word[13]),
String_x + String_Width * 3, String_Width, (Y_Number - String_y),
String_Color, String_Backcolor, 0);

/* Merge "Rate" value character data with the image data. */
String_Line_Fill(Display_Buffer, (uint8_t *)(&XL_Word[Rate / 10]),
String_x + String_Width * 4, String_Width, (Y_Number - String_y),
String_Color, String_Backcolor, 0);
String_Line_Fill(Display_Buffer, (uint8_t *)(&XL_Word[Rate % 10]),
String_x + String_Width * 5, String_Width, (Y_Number - String_y),
String_Color, String_Backcolor, 0);
}

Y_Number++;

/* Send the merged data */
Display_Buffer = PDMA_For_LCD_Data_Send(Display_Buffer, Pic_Width);
}

/* Waiting for the sending to be completed */
PDMA_For_LCD_Wait_Send_End();

/* Return the address of the next image data in Flash */
return (Addr + Pic_Width * Pic_Height * 2);
}

```

函数 **String_Line_Fill()** 通过将字符位图数据与图像数据合并的方式更新显示缓冲区。像素值是根据字体位图定义和颜色参数进行更新的。

```

/* Update the character data to the buffer. */
void String_Line_Fill(uint16_t *Buff, uint8_t *String, uint8_t x, uint16_t String_Width,
uint8_t Line_Num, uint16_t String_Color, uint16_t Back_Color,
uint8_t Backcolor_Enable)
{
uint16_t i = 0;
uint8_t Bit_Number = 0;
uint16_t Pixel_Cnt = 0;
uint16_t Width = (String_Width / 8);

for (i = 0; i < Width; i++)
{
for (Bit_Number = 0; Bit_Number < 8; Bit_Number++)
{
if ((String[Line_Num * Width + i] >> (7 - Bit_Number)) & 0x01)
{
Buff[x+Pixel_Cnt] = String_Color;
}
}
}
}

```

```

    }
    else
    {
        if (Backcolor_Enable == 1)
        {
            Buff[x+Pixel_Cnt] = Back_Color;
        }
    }

    Pixel_Cnt++;
}
}
}

```

函数 PDMA_For_LCD_Data_Send() 和 PDMA_For_LCD_Trigger() 实现了使用 PDMA 进行的双缓冲式 SPI 数据传输，以确保液晶显示屏的持续且高效更新。

```

/* Use PDMA to send data of a specific length from the cache. */
uint16_t *PDMA_For_LCD_Data_Send(uint16_t *buffer, uint16_t count)
{
    while (1)
    {
        /* Wait for the data transmission to be completed. */
        if (PDMA_For_LCD_Send_End_Flag == 1)
        {
            PDMA_For_LCD_Send_End_Flag = 0;

            /* Trigger data transmission */
            PDMA_For_LCD_Trigger(buffer, count);

            /* Switch the sending buffer */
            if (buffer == LCD_Display_Buffer0)
            {
                buffer = LCD_Display_Buffer1;
            }
            else
            {
                buffer = LCD_Display_Buffer0;
            }

            break;
        }
    }

    return buffer;
}

void PDMA_For_LCD_Trigger(uint16_t *buffer, uint16_t count)
{
    PDMA_Open(PDMA0, (1<<PDMA_FOR_LCD_CH0));

    /*=====
    SPI master PDMA TX channel configuration:
    -----
    Word length = 16 bits
    */
}

```

```

    Transfer Count = count
    Source = buffer
    Source Address = Increasing
    Destination = SPI1->TX
    Destination Address = Fixed
    Burst Type = Single Transfer

    =====*/
    PDMA_SetTransferCnt(PDMA0, PDMA_FOR_LCD_CH0, PDMA_WIDTH_16, count);
    PDMA_SetTransferAddr(PDMA0, PDMA_FOR_LCD_CH0, (uint32_t)(buffer), PDMA_SAR_INC,
        (uint32_t)&SPI1->TX, PDMA_DAR_FIX);
    PDMA_SetTransferMode(PDMA0, PDMA_FOR_LCD_CH0, PDMA_SPI1_TX, FALSE, 0);
    PDMA_SetBurstType(PDMA0, PDMA_FOR_LCD_CH0, PDMA_REQ_SINGLE, PDMA_BURST_4);

    /* Enable PDMA channel 0 interrupt */
    PDMA_EnableInt(PDMA0, PDMA_FOR_LCD_CH0, PDMA_INT_TRANS_DONE);

    NVIC_EnableIRQ(PDMA0_IRQn);

    SPI_TRIGGER_TX_PDMA(SPI1);
}

```

Timer0 以周期性模式进行配置，具有固定的中断间隔。中断服务程序会递增一个计数器值，该计数器用于计算显示刷新率。

```

void Timer0_Init(void)
{
    /* Open Timer0 in periodic mode, enable interrupt
    and 1 interrupt tick per 100 microsecond */
    TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, 10000);
    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++;
    }
}

```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
 - M3331_Series_BSP_CMSIS_V3.00.002
- IDE 版本
 - Keil uVision 5.41
 - VS Code 1.1111.0

3.2 硬件需求

- 电路组件
 - NuMaker-M3334KI V2.0
 - NuFun-M3334KI V2.0
- 线路示意图
 - 将 NuMaker-M3334KI V2.0 板与 NuFun-M3334KI V2.0 板，通过板载连接器对插连接起来。如图 3-2 所示，两组黄色框起的连接器对插，标识「A」方向对齐；两组蓝黑色框起的连接器对插，标识「B」方向对齐。MCU 位于 NuMaker-M3334KI V2.0 上，TFT 及 FLASH 位于 NuFun-M3334KI V2.0 上，接线示意图如图 3-1 所示。注意：为避免 FLASH 发生写保护，「/WP、/HOLD、/CS」引脚建议接上拉电阻。

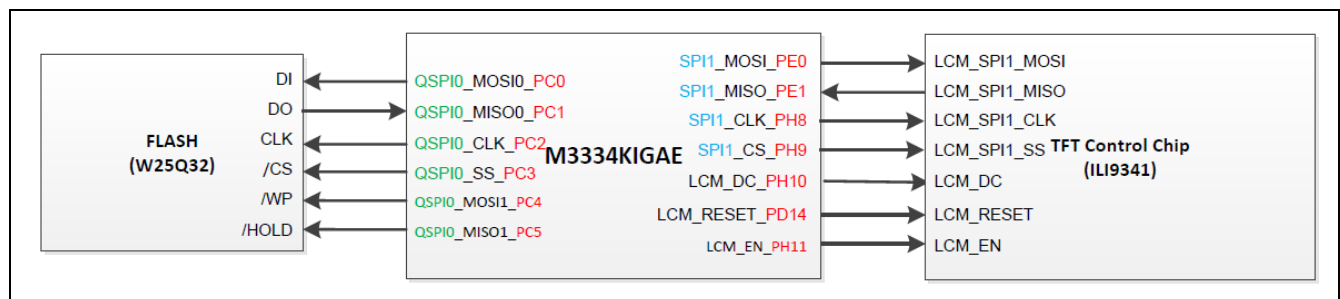


图 3-1 接线示意图

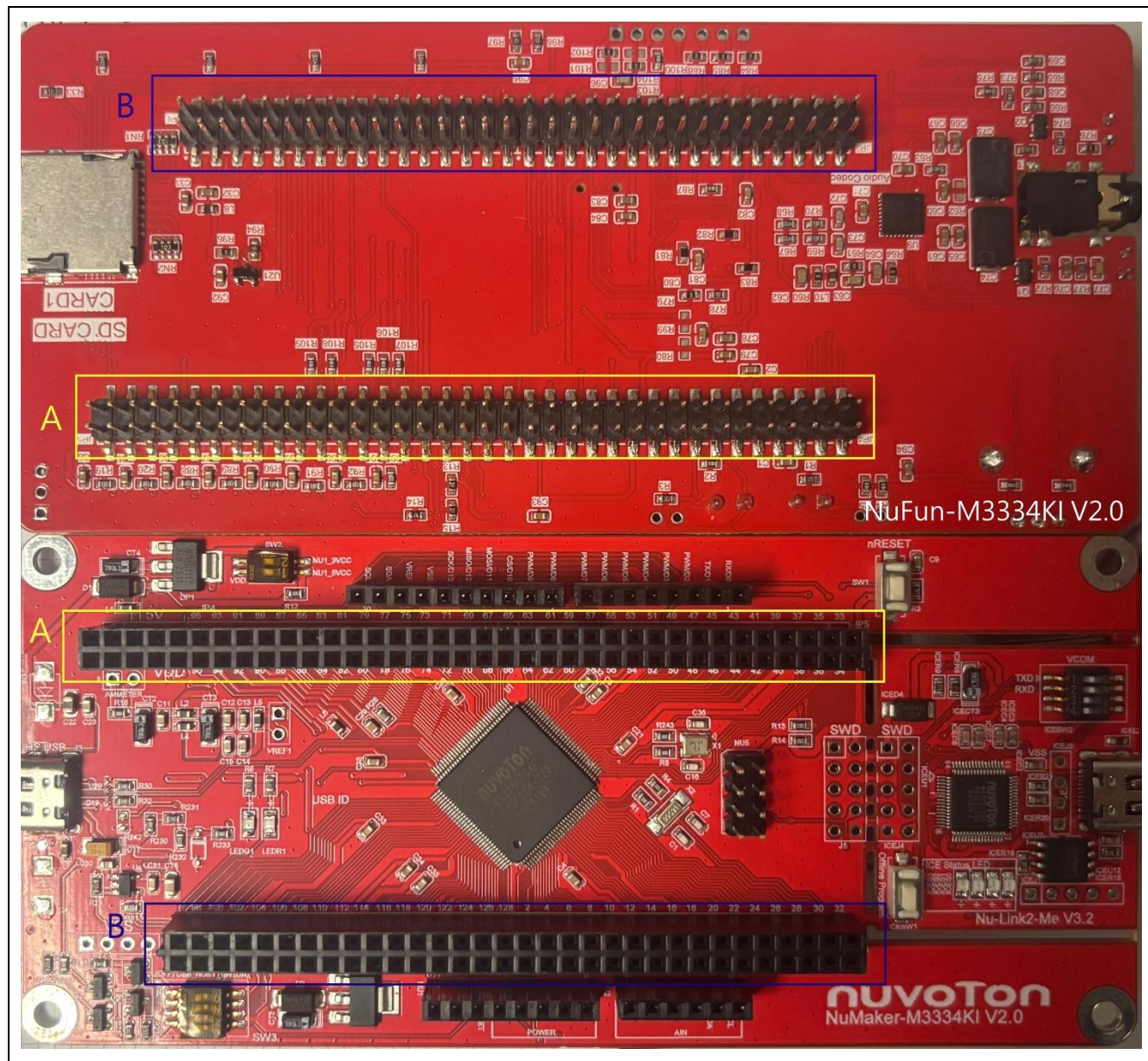


图 3-2 实物接线图

4. 目录信息

目录结构如下图所示。

 EC_M3331_Displaying_Animations_and_Overlaid_Characters_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
 Tool	ICP & ImagesToBin_Uncompress

图 4-1 目录信息

5. 范例程序执行

5.1 Keil 平台

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 *Display.uvprojx*。
2. 进入编译模式界面
 - 编译
 - 下载代码至内存
 - 进入 / 离开仿真模式
3. 进入仿真模式界面
 - 执行代码

5.2 VS Code 平台

1. 选择 **File**
 - 单击 **Open Folder**
 - 根据目录信息章节选择 Display_Animation 路径下的 VSCode 文件夹
2. 选择 **CMSIS**
 - 单击 **Build solution** 编译工程
 - 单击 **Load & Run application** 下载程序到目标芯片
 - 单击 **Load & Debug application** 进入仿真模式

6. 修订纪录

Date	Revision	Description
2026.03.31	1.00	初始发布。
2026.08.13	1.01	修改LCD_RST宏定义及添加Tool 源码。
2026.09.20	1.02	1.在修改SPiX_CTL、SPiX_CLKDIV、 SPiX_SSCTL、SPiX_FIFOCTL寄存器前，先禁用SPI，修改完后，再使能SPI。 2.在修改QSPiX_CTL、QSPiX_CLKDIV、 QSPiX_SSCTL、QSPiX_FIFOCTL寄存器前， 先禁用QSPI，修改完后，再使能QSPI。

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.