

Displaying Animations and Overlaid Characters

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example demonstrates using QSPI Quad mode in combination with PDMA to read image data from external Flash memory. Character data is then overlaid on each image frame, and the final data is sent to the TFT controller through the SPI interface for display.
BSP Version	M3331_Series_BSP_CMSIS_V3.00.002
Hardware	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. Overview

This example code uses QSPI Quad mode together with PDMA to sequentially read multiple animation frame images stored in external Flash memory. Character data is overlaid on each image frame. Finally, the processed frame data is transmitted through the SPI interface together with PDMA to the TFT controller for display.

1.1 Principle

The 320×240 TFT display operates in RGB565 format, where each pixel requires 16 bits (2 bytes) of data. A full-screen frame therefore requires 153,600 bytes ($320 \times 240 \times 2$). The external Flash memory (W25Q32) on the evaluation board has a capacity of 4 MB, allowing storage of up to 27 animation frames. Run the *ImagesToBin_Uncompress.exe* file in the Tool folder. **Note:** This tool is provided solely for learning and reference purposes. This tool will convert 27 BMP format images (with a resolution of 320×240 pixels and a color depth of 16 bits) in the current directory into RGB565 format data (high byte first). The converted data is merged sequentially to generate a single *SPI_Flash.bin* file. The image scanning method is illustrated in Figure 1-1.



Figure 1-1 Image Scanning Process

Using the ICP programming tool, the BIN file is programmed into the external Flash device on the NuFun-M3334KI board. The procedure is described as follows:

1. Install the programming tool by double-clicking the *NuMicro ICP Programming Tool 3.22.7959a.exe* located in the Tool folder. The *SPI_Flash.ini* will only be available in the installation directory after the installation is complete.
2. Locate and open *SPI_Flash.ini* in the installation directory. Refer to Figure 1-2 and set the path of the BIN file to be programmed (Step 2). After programming is completed, set the BIN file path used for verification readback from Flash (Step 3).

```
[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

Figure 1-2 Programming Parameter Setting

- First, connect the evaluation board to the PC using a USB cable. Then, follow the instructions in Figure 1-3. During the process of moving on to Step 1, if it is indicated that the firmware version does not match the driver version, then follow the prompt to upgrade the firmware. Note that in the **Programming** option, since there is no Internal Flash to be programmed, the **Chip Setting** must be checked.

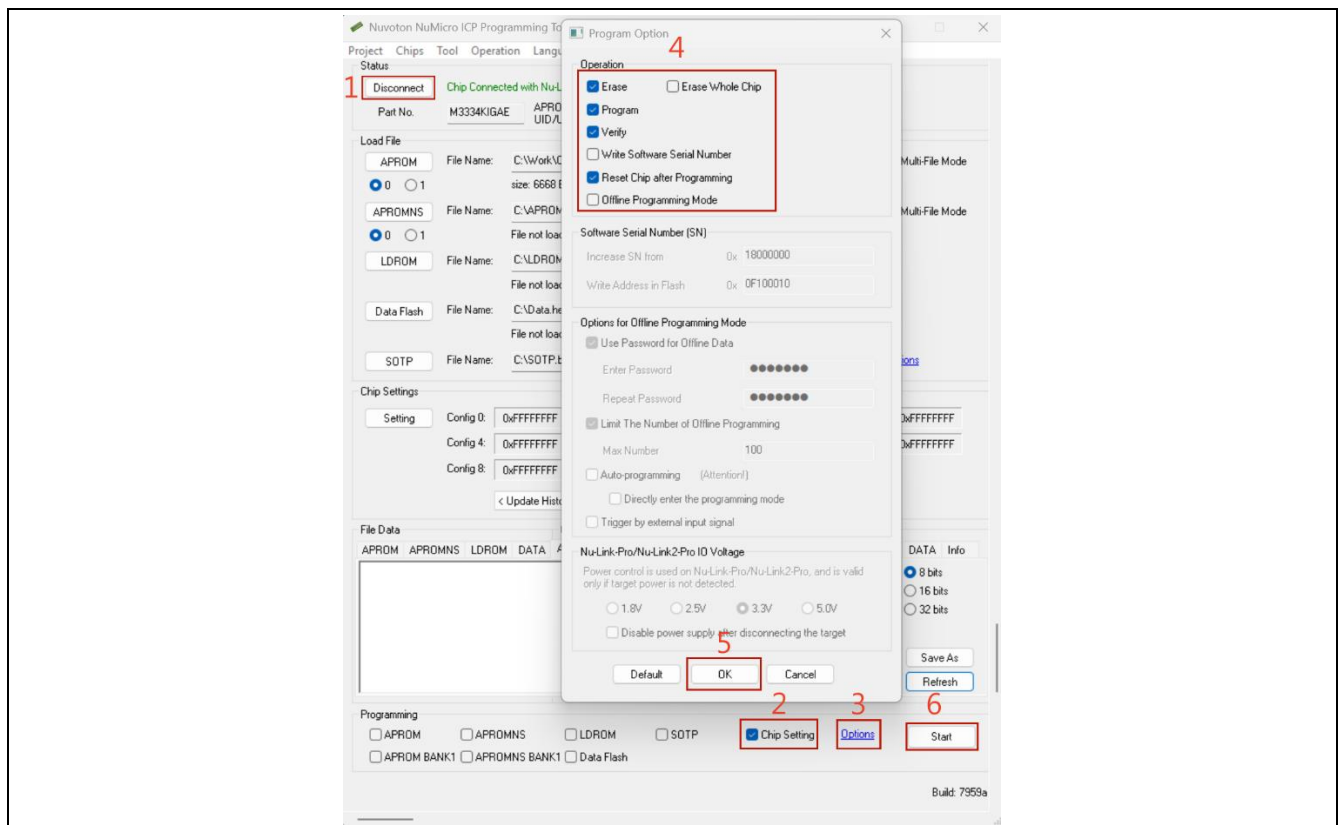


Figure 1-3 Programming Options Setting

The microcontroller (MCU) communicates with the external Flash through the QSPI interface. To improve data throughput, Quad 4-line mode is enabled and RX PDMA is used with a data width of 32 bits. The MCU connects to the TFT controller through the SPI interface, where TX PDMA and a dual transmission buffer are employed with a data width of 16 bits.

After power-on, the MCU initializes all required peripherals. In the main loop, image data is read sequentially from external Flash memory, character data is overlaid on each frame, and the resulting data is transmitted to the TFT controller to generate the animation.

The overall software flow of this example is summarized as follows:

1. The system clock, GPIO, UART, Timer0, QSPI, SPI, PDMA, and LCD peripherals are initialized after power-on.
2. The MCU configures the external Flash to operate in QSPI Quad mode and enables RX PDMA to read image data efficiently.
3. Animation frame data is read sequentially from external Flash memory into the display buffer.
4. Character data, such as the real-time frame rate, is merged with the image data line by line in the display buffer.
5. The merged frame data is transmitted to the TFT controller through the SPI interface using TX PDMA with a double-buffer mechanism.
6. Timer0 is used to measure the display refresh rate, which is calculated and reported through the UART interface.
7. Steps 3 to 6 are repeated for each frame, thereby generating a continuous animation effect on the TFT display screen.

1.2 Demo Result

After the example code is compiled and programmed into the MCU, animation frames with overlaid characters are displayed on the TFT screen. One example display is shown in Figure 1-4.

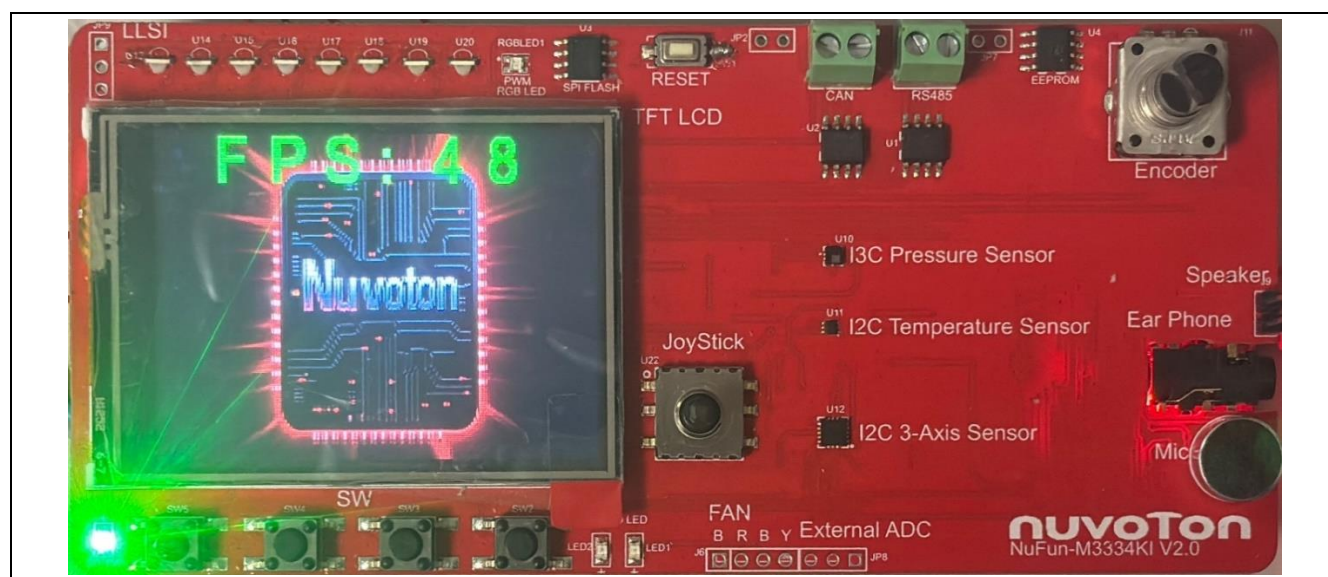


Figure 1-4 Display Output After Programming

Meanwhile, the PC UART terminal displays the real-time display refresh rate, for example:

FPS : 48.
FPS : 48.
FPS : 48.
FPS : 48.
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. Code Description

The following section describes the key code components of this example. All code snippets are located in *main.c* unless stated otherwise.

The following code defines the macro constants used in this example.

```
#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
```

The system initialization includes UART, Timer0, QSPI, SPI, and LCD configuration. During initialization, protected registers are unlocked, system clocks and I/O are configured, and the LCD is cleared with a predefined color. Timer0 is configured to measure the refresh rate of the LCD. The PDMA interrupt priority is set to the highest level to ensure efficient data transmission.

```
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);

```

The main loop is responsible for displaying the animation and overlaying character data.

For each frame:

The refresh timer counter is reset.

Image data is read from external Flash memory and merged with character data.

The merged data is sent to the LCD via SPI with PDMA support.

The refresh rate is calculated and printed through UART.

```

/* 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);
    }

```

The function `LCD_Display_Picture_And_String()` reads image data from external Flash memory at a specified address, displays it in a defined LCD region, and overlays text characters such as the frame rate value.

```

/* 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);
}

```

The function String_Line_Fill() updates the display buffer by merging character bitmap data with image data. Pixel values are updated based on font bitmap definitions and color parameters.

```

/* 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++;
}
}
}

```

The functions PDMA_For_LCD_Data_Send() and PDMA_For_LCD_Trigger() implement double-buffered SPI data transmission using PDMA to ensure continuous and efficient LCD updates.

```

/* 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 is configured in periodic mode with a fixed interrupt interval. The interrupt service routine increments a counter used to calculate the display refresh rate.

```

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. Software and Hardware Requirements

3.1 Software Requirements

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

3.2 Hardware Requirements

- Circuit components
 - NuMaker-M3334KI V2.0
 - NuFun-M3334KI V2.0
- Pin Connect
 - Connect the connectors on the NuMaker-M3334KI board to those on the NuFun-M3334KI board as shown in Figure 3-2. The connector groups marked with identifier A and B must be aligned accordingly. The MCU is located on the NuMaker-M3334KI board, while the TFT display and Flash memory are located on the NuFun-M3334KI board.

Note: To prevent FLASH from entering a write-protected state, it is recommended to connect pull-up resistors to the "/WP", "/HOLD" and "/CS" pins.

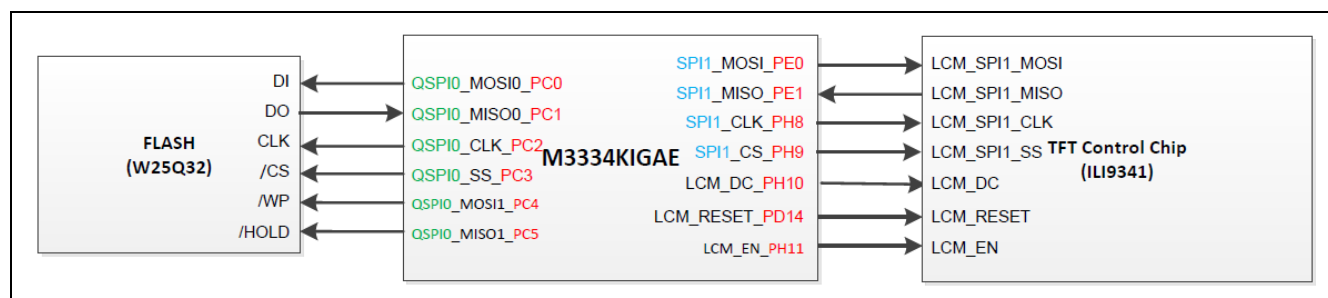


Figure 3-1 Pin Connect

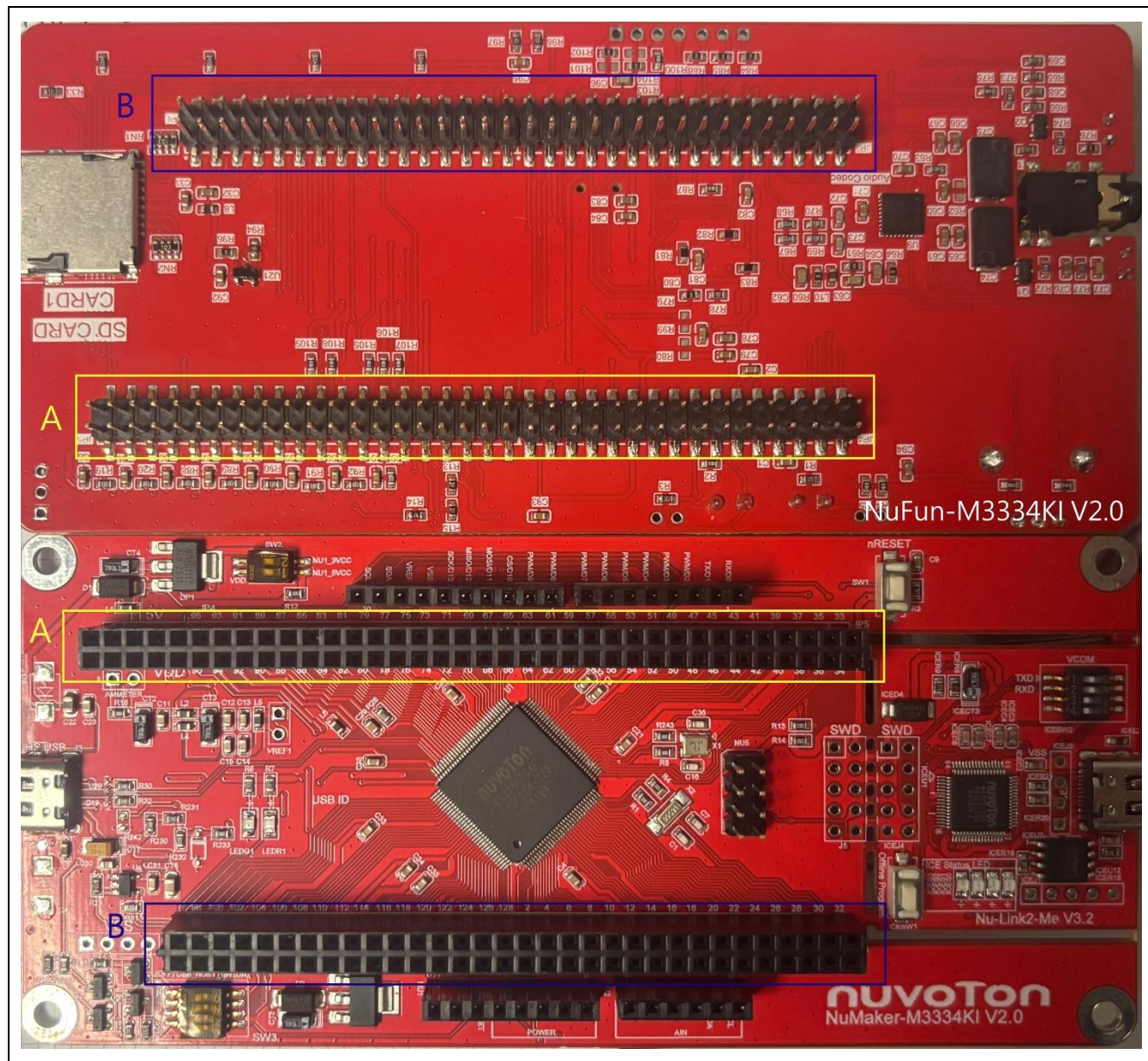


Figure 3-2 Connection Diagram

4. Directory Information

The directory structure is shown below.

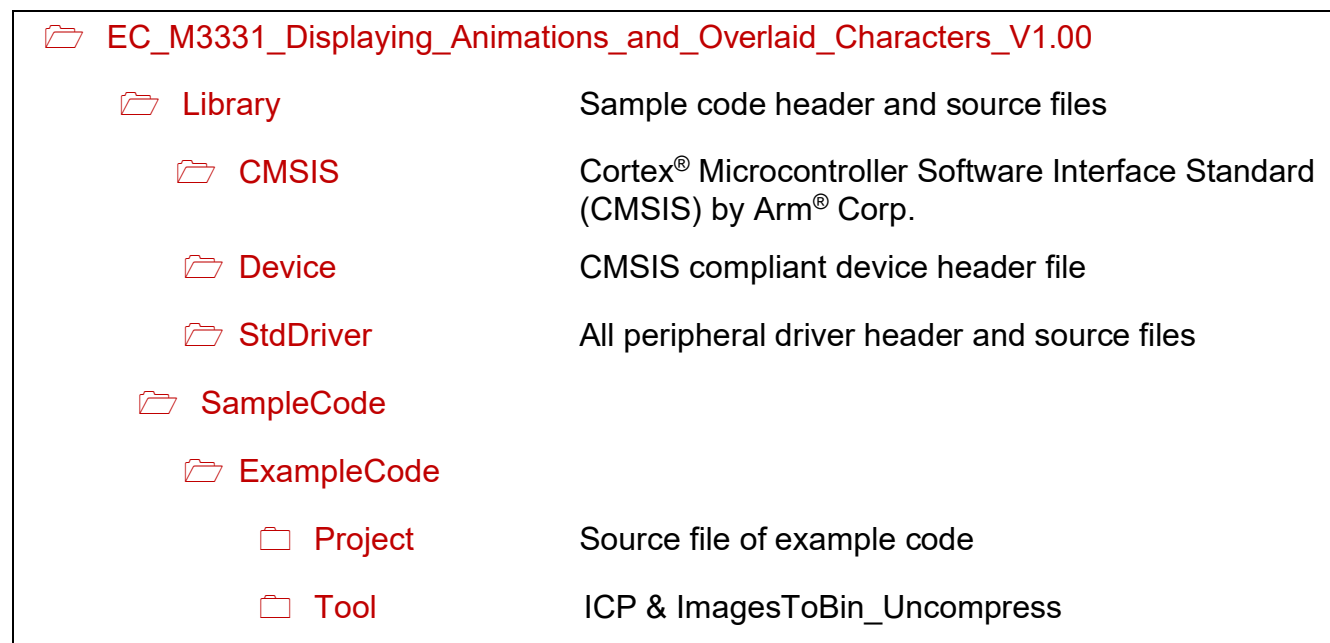


Figure 4-1 Directory Structure

5. Example Code Execution

5.1 Keil MDK

1. Browse the sample code folder as described in the Directory Information section and double-click *Display.uvprojx*.
2. Enter Keil compile mode.
 - Build
 - Download
 - Start/Stop debug session
3. Enter debug mode.
 - Run

5.2 VS Code

1. Select **File**.
 - Click **Open Folder**.
 - According to the Directory Information section, select the VSCode folder under the Display_Animation path.
2. Select **CMSIS**.
 - Click **Build solution** to compile the project.
 - Click **Load & Run application** to program the target.
 - Click **Load & Debug application** to enter debug mode.

6. Revision History

Date	Revision	Description
2026.03.31	1.00	Initial version.
2026.08.13	1.01	Modify the LCD_RST macro definition and add Tool source code.
2026.09.20	1.02	1.Before changing the configurations of SPIx_CTL, SPIx_CLKDIV, SPIx_SSCTL and SPIx_FIFCTL registers, first clear the SPIEN and confirm the SPIENSTS is 0. 2.Before changing the configurations of QSPIx_CTL, QSPIx_CLKDIV, QSPIx_SSCTL and QSPIx_FIFCTL registers, first clear the SPIEN and confirm the SPIENSTS is 0.

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.*