Using SPIM to Read External SPI Flash Image

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses the NuMaker-M467HJ to implement reading image files stored in the microcontroller's external SPI Flash via SPIM and displaying them on the NuMaker-M467HJ's LCD. It also compares the reading speed difference between SPIM's two-wire (Dual) and four-wire (Dual) modes.
BSP Version	M460_Series_BSP_CMSIS_V3.00.004 emWin Software Package for M460 V7.1
Hardware	NuMaker- M467HJ V1.1

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 demonstrates using the NuMaker-M467HJ evaluation board to implement a solution where image files, stored in external SPI Flash memory, are read via the SPIM (SPI Synchronous Serial Interface) and displayed on the NuMaker-M467HJ's LCD. It also provides a performance comparison between the two-wire (Dual) and four-wire (Quad) SPIM read modes.

In Human-Machine Interface (HMI) applications or other projects requiring rich graphical displays, designers are often constrained by the limited internal SRAM capacity of the microcontroller (MCU). Consequently, it is common practice to store large graphical assets (such as UI icons, background images, etc.) in external SPI Flash. When these assets are needed for display, they are transferred at high speed to the MCU via the SPIM interface and subsequently pushed to the LCD driver.

1.1 Principle

The core of this example is to leverage the Multi-I/O capability of the SPIM (SPI Synchronous Serial Interface) to efficiently read image data stored in external SPI Flash and display it on the LCD screen.

1.1.1 SPIM and Direct Memory Mapping Function

The SPIM module of the NuMicro M467 series microcontroller is a high-performance interface designed for high-speed access to external SPI Flash. It not only fully supports standard SPI mode but also provides Dual and Quad Multi-I/O modes, while also featuring a powerful Direct Memory Mapping mode (DMM mode) function.

The entire process begins with the MCU acting as the Master, issuing a read command and the target address of the image to the external SPI Flash. Utilizing the SPIM's DMM mode, the operating principle is that the SPIM Controller virtually maps the external SPI Flash memory space into the MCU's internal address space, as shown in Table 1-1. From the CPU's perspective, the external Flash is treated as part of the internal memory, and its access address is unified into the MCU's memory map.

Memory Address	Size	Functions
0x40007000-0x40007FFF	4 Kbytes	SPIM control registers: Store parameters related to SPIM configuration.
0x100000-0x020FFFFF	32 Mbytes	The internal MCU address space where the external SPI Flash is mapped. (**Note**: This address space is only applicable to models with a Flash size of 1024 Kbytes.)

Table 1-1 SPIM Memory Mapped Address

Once in this mode, when the CPU attempts to perform a read operation from this mapped memory address (0x100000), the SPIM Controller automatically executes the complete SPI Flash read timing sequence. This includes sending the read command and address, and initiating high-speed data transfer, all without CPU intervention or additional function calls. This greatly simplifies the image data acquisition process, especially for applications requiring frequent image data reads.

For SPIM-related software configurations, please refer to the Code Description section.

1.1.2 Dual Mode and Quad Mode

To enhance performance, SPI Flash read efficiency is a critical factor. The standard SPI interface utilizes a pair of data lines (MISO and MOSI) for data input and output. To increase efficiency, most SPI Flash devices support Multi-I/O modes, specifically Dual mode and Quad mode.

In Dual mode, the traditional MOSI and MISO pins are reconfigured as bidirectional IO0 and IO1 data lines, allowing the SPIM to transmit 2 bits (2-bit) in parallel per clock cycle, thereby doubling the data throughput. Furthermore, Quad mode redefines the Write Protect (WP#) and Hold (HOLD#) pins on the Flash chip as additional data lines, IO2 and IO3. Once the Quad Enable (QE) bit is set in the Flash chip, the SPIM can utilize these four data lines to transmit 4 bits (4-bit) in parallel per clock cycle, theoretically boosting the bandwidth to twice that of Dual mode.

When configuring the SPIM to enter DMM mode with Dual/Quad mode enabled, it is essential to ensure command compatibility between the MCU and the external SPI Flash. Different Flash chip vendors or models may use different high-speed continuous read commands (Fast Read Opcodes).

The evaluation board used in this example is the NuMaker-M467HJ V1.1. As seen from the circuit schematic, the selected Winbond SPI Flash is the W25Q32JVSSIQ. This model supports the read commands configured by the M467 series SPIM, such as command 0x3B (Fast Read Dual Output) and command 0xEB (Fast Quad Read). Additionally, parameters like Dummy clocks must be set according to the waveform configurations in the SPI Flash specification (see Figure 1-1) to meet timing requirements.

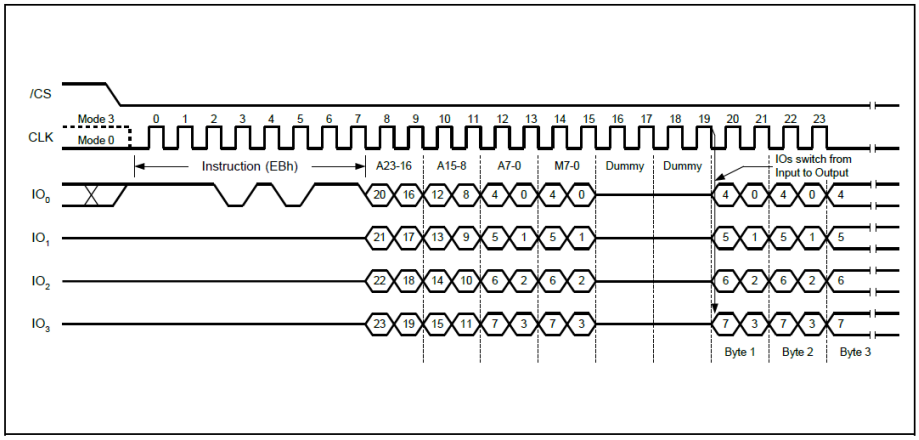

Figure 24. Fast Read Quad I/O Instruction (M7-M0 should be set to Fxh)

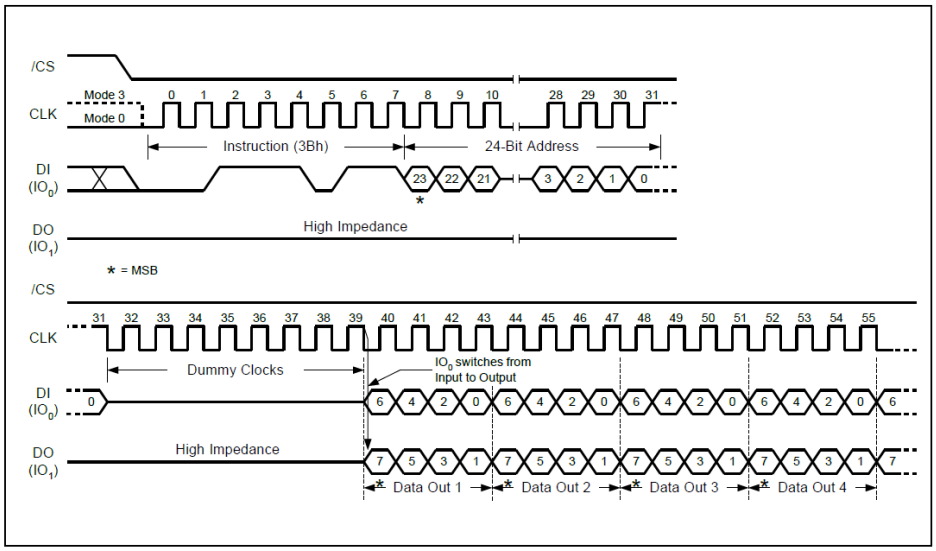

Figure 18. Fast Read Dual Output Instruction

Source: Winbond W25Q32JV Datasheet

Figure 1-1 SPI Flash Specification

The hardware pin configuration is shown in Table 1-2, and the circuit connection is shown in Figure 1-2.

Pin Number	I/O Function
PI12	SPIM_SS
PI13	SPIM_MISO
PI14	SPIM_D2
PI15	SPIM_D3
PJ0	SPIM_CLK
PJ1	SPIM_MOSI

Table 1-2 Pin Configuration

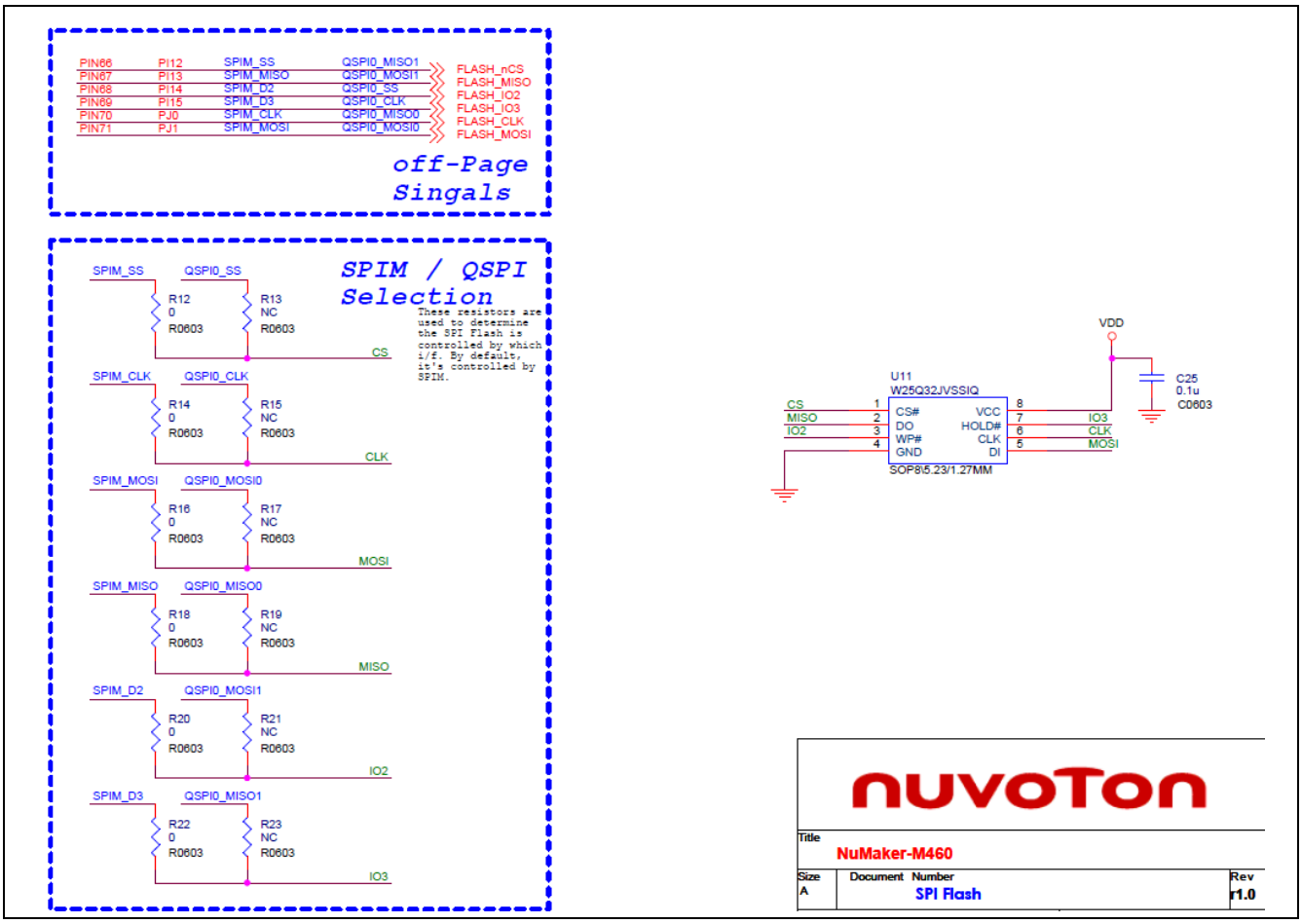

Figure 1-2 SPI Flash Circuit Connection Diagram

1.2 Demo Result

1.2.1 Programming

When programming the SPI Flash, it is necessary to specify the programming pins in the programmer settings. Using Keil MDK as an example, refer to Figure 1-3 for the configuration and select the corresponding pins according to the pin layout in Table 1-2.

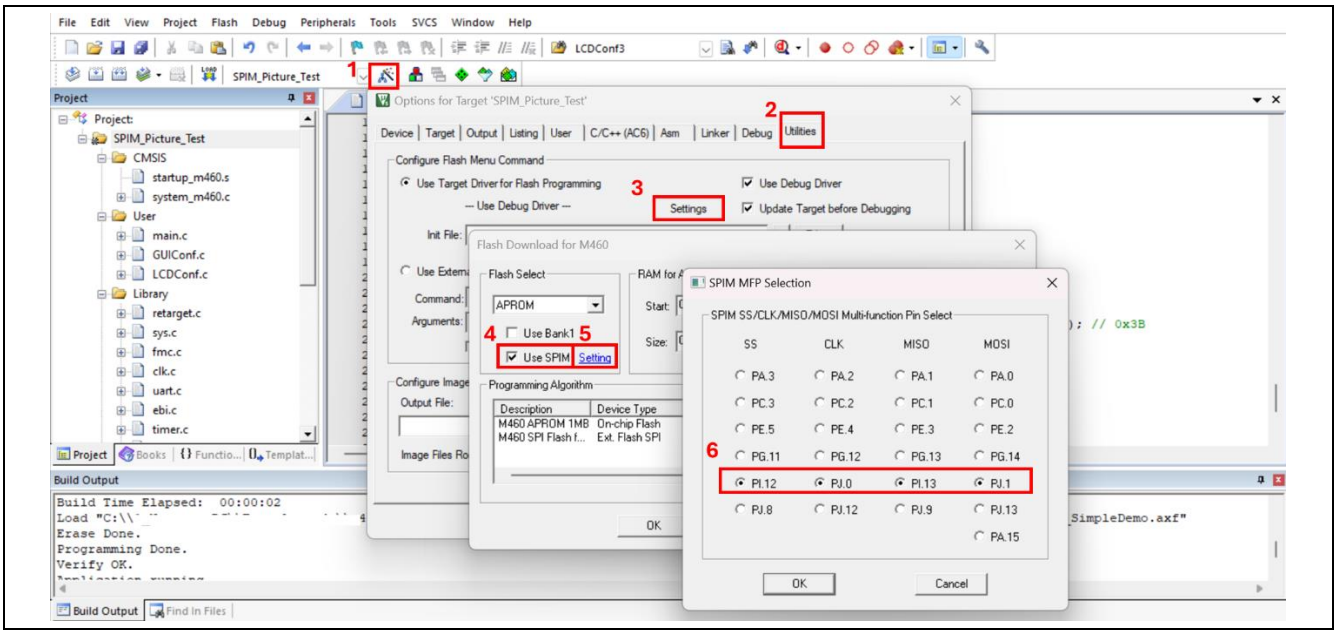

Figure 1-3 SPI Flash Programming Settings

1.2.2 Execution

The results of compiling and programming are shown in Figure 1-4 and Figure 1-5. After initialization is complete, use a PC UART terminal to input '0' or '1' to enter the Quad mode or Dual mode test modes, respectively. Upon execution, the image will be successfully displayed on the LCD, and the time required to load one 480x272 resolution image from the SPI Flash can be viewed via the printed information.

Based on the software time measurement, it can be observed that loading one 480x272 resolution image takes only about 18 ms in Quad mode, whereas Dual mode requires about 35 ms. This confirms the data bandwidth advantage of Quad mode, which significantly reduces image loading time and has a decisive impact on the smoothness of the HMI.

```
+-------------------------------------------------------------------+
  emWin AppWizard SPIM Read Mode Test
  This example project is based on emWin AppWizard.
  It demonstrates the performance and initialization
  differences between SPIM Fast Quad Read mode and
  Fast Dual Read mode.

  Please select the test mode:
    Enter 0 -> Test Fast 'Quad' Read mode
    Enter 1 -> Test Fast 'Dual' Read mode
+-------------------------------------------------------------------+

Setting up Fast 'Quad' Read mode...
== Test completed in [18] ms with Mode 0. Picture showed on LCD. ==
```

Figure 1-4 Compile and Program Results – Quad Mode

```
+-----------------------------------------------------------------+
  emWin AppWizard SPIM Read Mode Test
  This example project is based on emWin AppWizard.
  It demonstrates the performance and initialization
  differences between SPIM Fast Quad Read mode and
  Fast Dual Read mode.

  Please select the test mode:
    Enter 0 -> Test Fast 'Quad' Read mode
    Enter 1 -> Test Fast 'Dual' Read mode
+-----------------------------------------------------------------+

Setting up Fast 'Dual' Read mode...
SPIM get JEDEC ID=0xEF, 0x40, 0x16
== Test completed in [35] ms with Mode 1. Picture showed on LCD. ==
```

Figure 1-5 Compile and Program Results – Dual Mode

Additionally, a commercially available logic analyzer can be used to perform an actual measurement of the image read time. This example uses the Acute MSO3124V instrument, as shown in Figure 1-6. By selecting the 'Serial Flash Bus Decode' mode for analysis, the measurement results can be obtained, as shown in Figure 1-7. Besides measuring the time, the Bus Decode results can also be used to verify whether the corresponding mode has been set correctly.

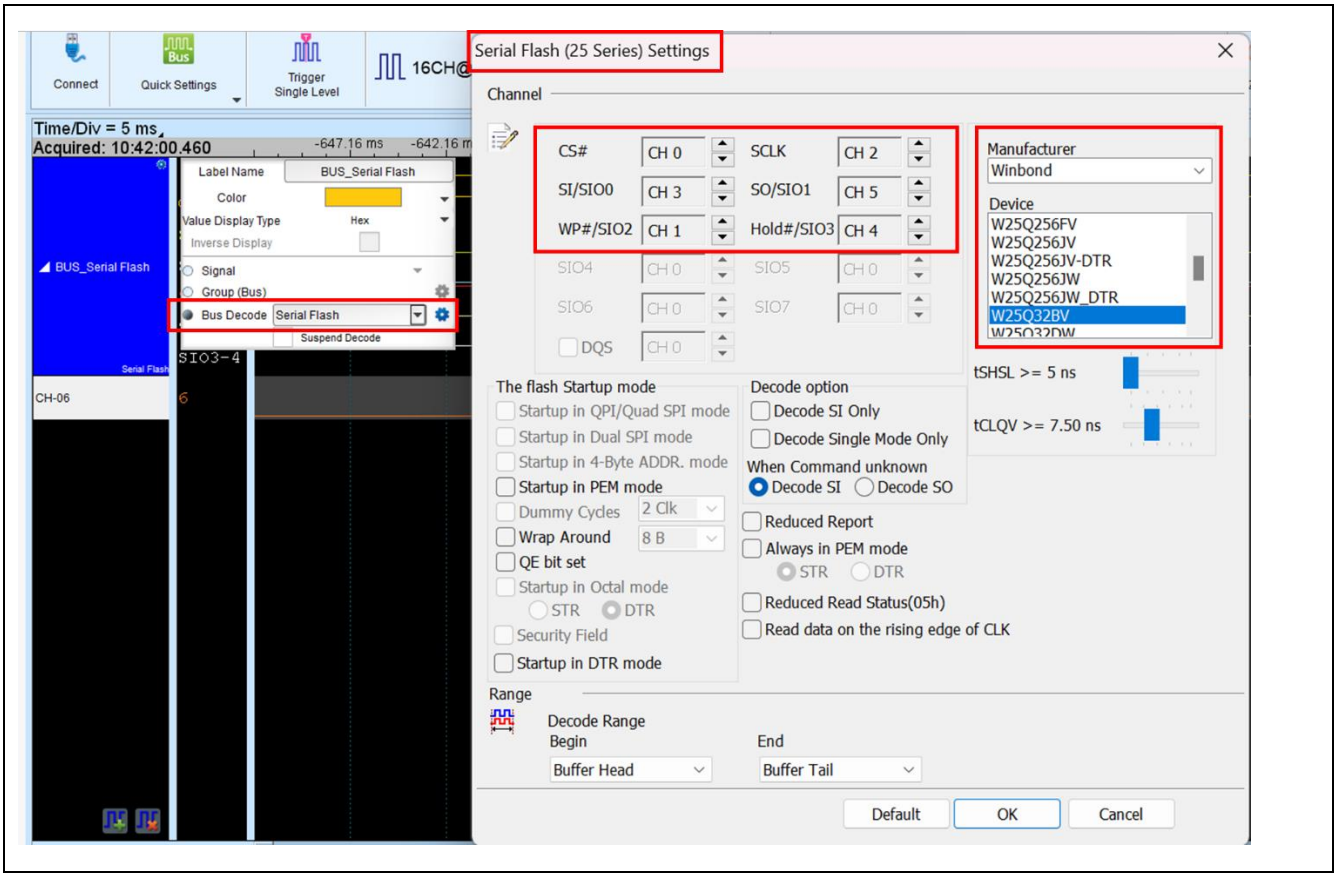

Figure 1-6 Logic Analyzer Bus Decode Settings

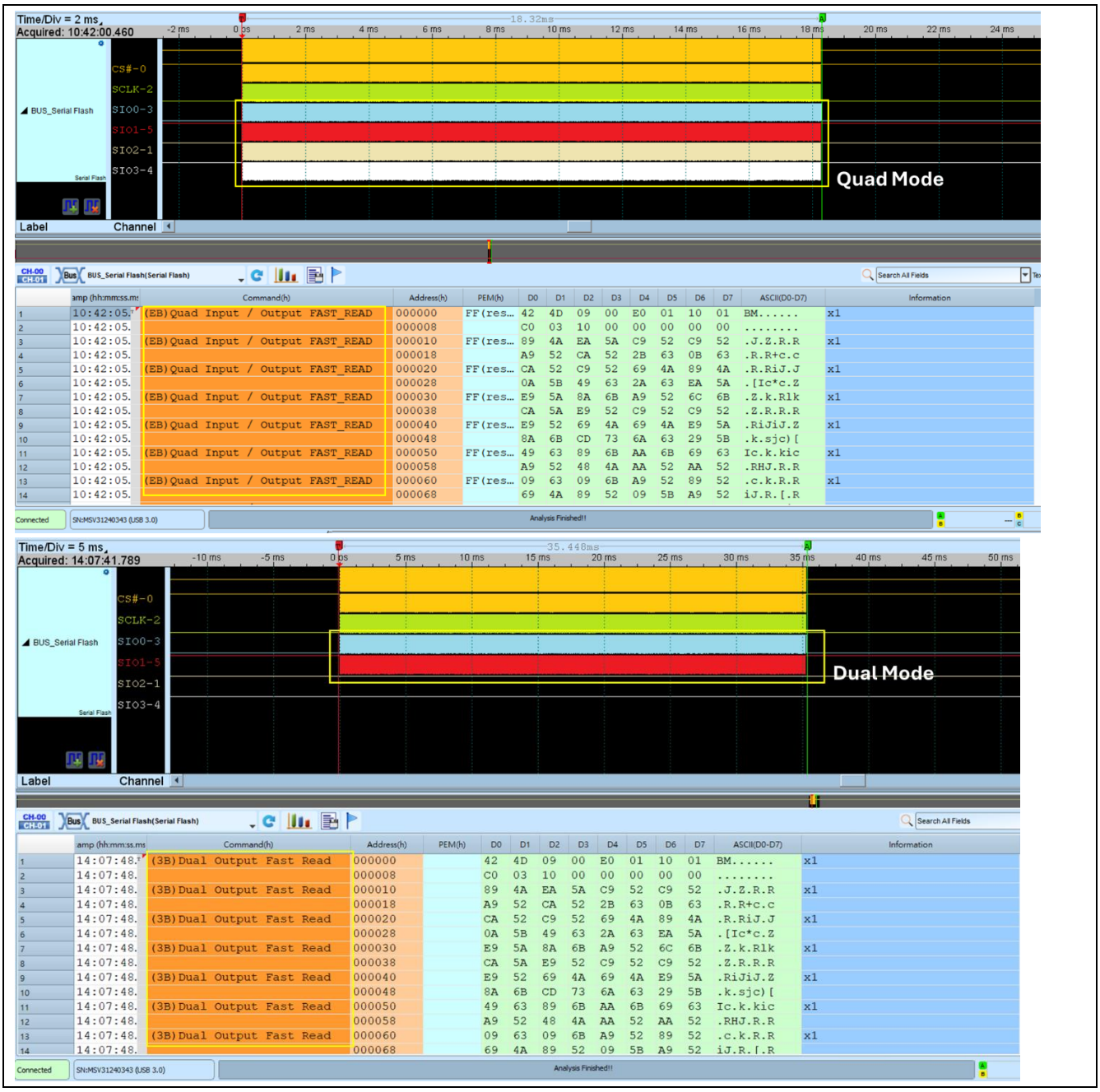

Figure 1-7 Logic Analysis Results

2. Code Description

The main configuration code for the SPIM is located in *main.c*.

SPIM initialization is divided into pin setup and parameter setup. The pin setup is located in the sys_init() function, where the SPIM multi-function pins are configured according to Table 1-2. Within this setup, the clock pin (PJ0) is set as a Schmitt trigger to enhance noise immunity and ensure clock signal stability. Furthermore, the SPIM I/O pins are set to high slew rate mode, which allows for faster voltage transitions to maintain signal integrity during high-speed communication.

```
void sys_init(void)
{
    …
    /* Init SPIM multi-function pins, MOSI(PJ.1), MISO(PI.13), CLK(PJ.0), SS(PI.12),
D3(PI.15), and D2(PI.14) */
    SET_SPIM_MOSI_PJ1();
    SET_SPIM_MISO_PI13();
    SET_SPIM_CLK_PJ0();
    SET_SPIM_SS_PI12();
    SET_SPIM_D3_PI15();
    SET_SPIM_D2_PI14();
    PJ->SMTEN |= GPIO_SMTEN_SMTEN0_Msk;

    /* Set SPIM I/O pins as high slew rate up to 80 MHz. */
    PI->SLEWCTL = (PI->SLEWCTL & 0x0CFFFFFF) | (0x1<<GPIO_SLEWCTL_HSREN12_Pos) |
                  (0x1<<GPIO_SLEWCTL_HSREN13_Pos) | (0x1<<GPIO_SLEWCTL_HSREN14_Pos) |
                  (0x1<<GPIO_SLEWCTL_HSREN15_Pos) ;
    PJ->SLEWCTL = (PJ->SLEWCTL & 0xF3FFFFF0) | (0x1<<GPIO_SLEWCTL_HSREN0_Pos) |
                  (0x1<<GPIO_SLEWCTL_HSREN13_Pos) ;
    …
}
```

The SPIM parameter setup is located in the main() function, with settings described as follows:

1. SPIM Clock**:** SPIM_SET_CLOCK_DIVIDER is used to set the clock divider. The calculation formula is HCLK / (Divider)^2.In this example, the HCLK frequency is 200 MHz, so setting the value to 2 means the SPIM frequency is 50 MHz.

2. Receive Clock Delay: SPIM_SET_RXCLKDLY_RDDLYSEL and SPIM_SET_RXCLKDLY_RDEDGE are the receive clock delay settings. Here, they are set for no delay.

3. Dummy Cycles: SPIM_SET_DCNUM defines the buffer time after the MCU sends the command and before the SPI Flash begins to output valid data. According to the specification waveform in Figure 1-1, 4 dummy cycles are required for command 0xEB (Fast Read Quad Output), while 8 dummy cycles are required for command 0x3B (Fast Read Dual Output).

4. Initialize SPI Flash and Verify ID: SPIM_InitFlash is the API used to initialize the SPI Flash. SPIM_ReadJedecId is also used to print the JEDEC ID of the SPI Flash.

5. SPIM Internal Cache: In DMM mode, the SPIM's internal 16 Kbytes cache can be enabled. This cache block serves as a buffer between the MCU and the SPI Flash. If nearby data is accessed consecutively, these accesses will hit directly in the internal cache, further reducing the number of accesses to the SPI Flash.
6. Encryption Bit (Cipher Bit): The Chiper Disable Bit determines whether to perform real-time encryption/decryption on the transferred data. As encryption/decryption requires additional processing time, this example disables the function by setting the macro SPIM_CIPHER_ON to 0.
7. 4-byte mode: The W25Q32JVSSIQ does not support 4-byte mode; therefore, the macro USE_4_BYTES_MODE is defined as 0 to use 3-byte mode.
8. SPIM DMM mode: SPIM_EnterDirectMapMode is the API for SPIM to enter DMM mode. The u32RdCmd parameter within this API is used to specify the I/O mode to be adopted, such as command 0xEB for Fast Read Quad Output or command 0x3B for Fast Read Dual Output.

```
int main(void)
{
   …
   if(mode == '0')
   {
      printf("Setting up Fast 'Quad' Read mode...\n");
      uint8_t idBuf[3];
      SYS_UnlockReg();
      SPIM_SET_CLOCK_DIVIDER(2);
      SPIM_SET_RXCLKDLY_RDDLYSEL(0);
      SPIM_SET_RXCLKDLY_RDEDGE();
      SPIM_SET_DCNUM(4);

      if (SPIM_InitFlash(1) != 0)
      {
        printf("SPIM flash initialize failed!\n");
      }
      SPIM_ReadJedecId(idBuf, sizeof (idBuf), 1);
      printf("SPIM get JEDEC ID=0x%02X, 0x%02X, 0x%02X\n", idBuf[0], idBuf[1], idBuf[2]);

      SPIM_ENABLE_CACHE();

      if (SPIM_CIPHER_ON)
        SPIM_ENABLE_CIPHER();
      else
        SPIM_DISABLE_CIPHER();

      if (SPIM_Enable_4Bytes_Mode(USE_4_BYTES_MODE, 1) != 0)
      {
        printf("SPIM_Enable_4Bytes_Mode failed!\n");
      }
      /* 0xEB */
      SPIM_EnterDirectMapMode(USE_4_BYTES_MODE, CMD_DMA_FAST_QUAD_READ, 0);
   }
   else if(mode == '1')
   {
```

```
      printf("Setting up Fast 'Dual' Read mode...\n");
      uint8_t idBuf[3];
      SYS_UnlockReg();
      SPIM_SET_CLOCK_DIVIDER(2);
      SPIM_SET_RXCLKDLY_RDDLYSEL(0);
      SPIM_SET_RXCLKDLY_RDEDGE();

      SPIM_SET_DCNUM(8);

      if (SPIM_InitFlash(1) != 0)
      {
        printf("SPIM flash initialize failed!\n");
      }
      SPIM_ReadJedecId(idBuf, sizeof (idBuf), 1);
      printf("SPIM get JEDEC ID=0x%02X, 0x%02X, 0x%02X\n", idBuf[0], idBuf[1], idBuf[2]);

      SPIM_ENABLE_CACHE();

      if (SPIM_CIPHER_ON)
        SPIM_ENABLE_CIPHER();
      else
        SPIM_DISABLE_CIPHER();

      if (SPIM_Enable_4Bytes_Mode(USE_4_BYTES_MODE, 1) != 0)
      {
         printf("SPIM_Enable_4Bytes_Mode failed!\n");
      }
      /* Disable quad mode bit */
      SPIM_SetQuadEnable(0, 2);
      /* 0x3B */
      SPIM_EnterDirectMapMode(USE_4_BYTES_MODE, CMD_DMA_FAST_READ_DUAL_OUTPUT, 0);
   }
   …
}
```

Finally, the GUI_DrawStreamedBitmapAuto() API from the emWin Library is used to draw an image with a 480x272 resolution. Timestamps are recorded before and after the drawing operation, and the calculated time difference is printed via UART.

```
int main(void)
{
   …
   GUI_MULTIBUF_Begin();
   tStart = OS_TimeMS;
   GUI_DrawStreamedBitmapAuto(acEbike_Demo_BG0, 0, 0);
   tEnd = OS_TimeMS;
   GUI_MULTIBUF_End();
   tElapsed = tEnd - tStart;
   printf("== Test completed in [%d] ms with Mode %c. Picture showed on LCD. ==\r\n",
   tElapsed, mode);
   GUI_Delay(2000);
   …
}
```

As described previously, the SPI Flash memory space is virtually mapped to the MCU's address space at 0x100000. In Keil MDK, a Scatter File can be used to precisely define memory regions.

The steps to open the Scatter File are shown in Figure 2-1, and an example configuration is as follows.

In this example, a region named LOAD_ROM_2 is defined with a starting address of 0x100000, which is the virtually mapped address of the SPI Flash. The image file *Ebike_Demo_BG0.o* is explicitly allocated to this region. This allows the MCU to access this image directly at 0x100000, achieving high-efficiency external data access.

```
LOAD_ROM_1  0x0
{
    APROM.bin  0x0
    {
     startup_M460.o (RESET, +FIRST)
     *(+RO)
    }

    SRAM  0x20000000 0x80000
    {
     * (+RW, +ZI)
    }
}

LOAD_ROM_2  0x100000
{
    SPIM.bin  0x100000
    {
     Ebike_Demo_BG0.o (+RO)
    }
}
```

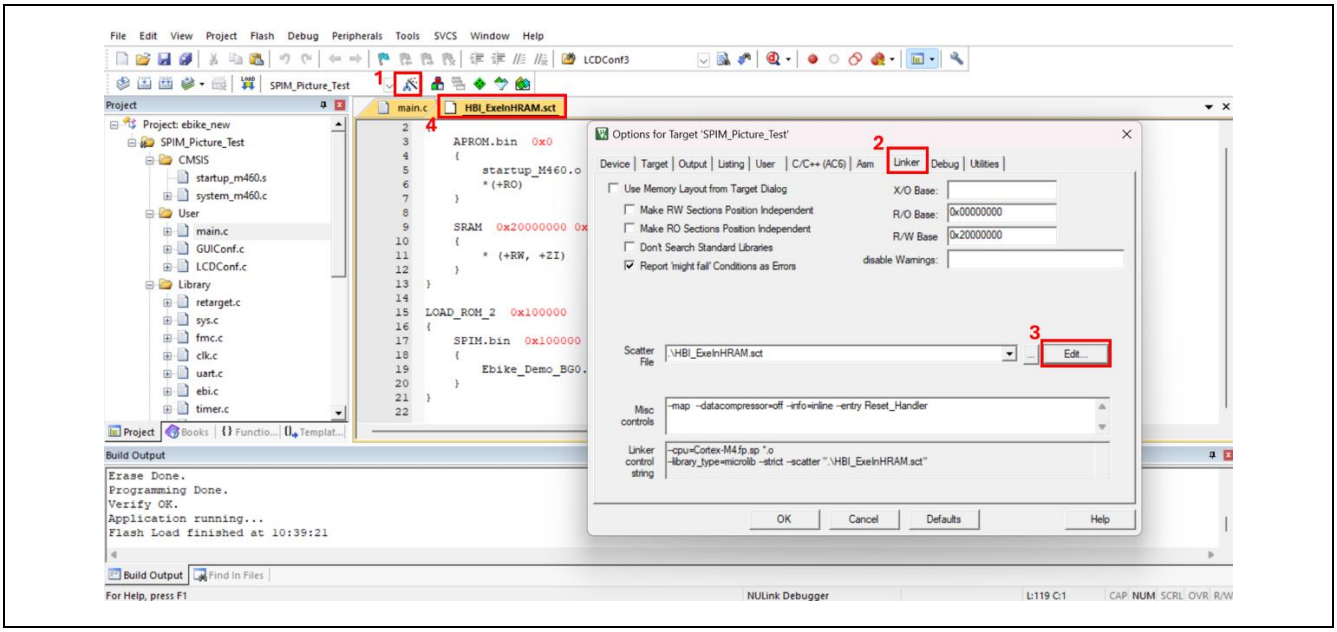

Figure 2-1 How to Open the Scatter File

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M460_Series_BSP_CMSIS_V3.00.004
  - emWin Software Package for M460 V7.1
- IDE version
  - Keil uVision 5.43

3.2 Hardware Requirements

- Circuit components
  - NuMaker- M467HJ V1.1

4. Directory Information

The directory structure is shown below.

EC_ M467_SPIM_Image_Quad_Dual_Test_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
ThirdParty	Header files for emWin project
SampleCode	
ExampleCode	
SPIM_Image_Quad_Dual_Test	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 *SPIM_Image_Quad_Dual_Test.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2025.11.01	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.