N9H26 SpiWriter User Guide

Document Information

Abstract	Introduce the steps to build and launch SpiWriter for the N9H26 series microprocessor (MPU).
Apply to	N9H26 series

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 Introduction

SpiWriter is the firmware stored in the SD card, which is used to write the necessary firmware to the SPI Flash for mass production. After the SpiWriter writes the necessary firmware, the device should become a complete product.

The SpiWriter can write following firmware to the SPI Flash.

- SPI loader
- Logo image
- Execute image
- User image

Figure 1 and **Error! Reference source not found.** describe the working environment and workflow of SpiWriter.

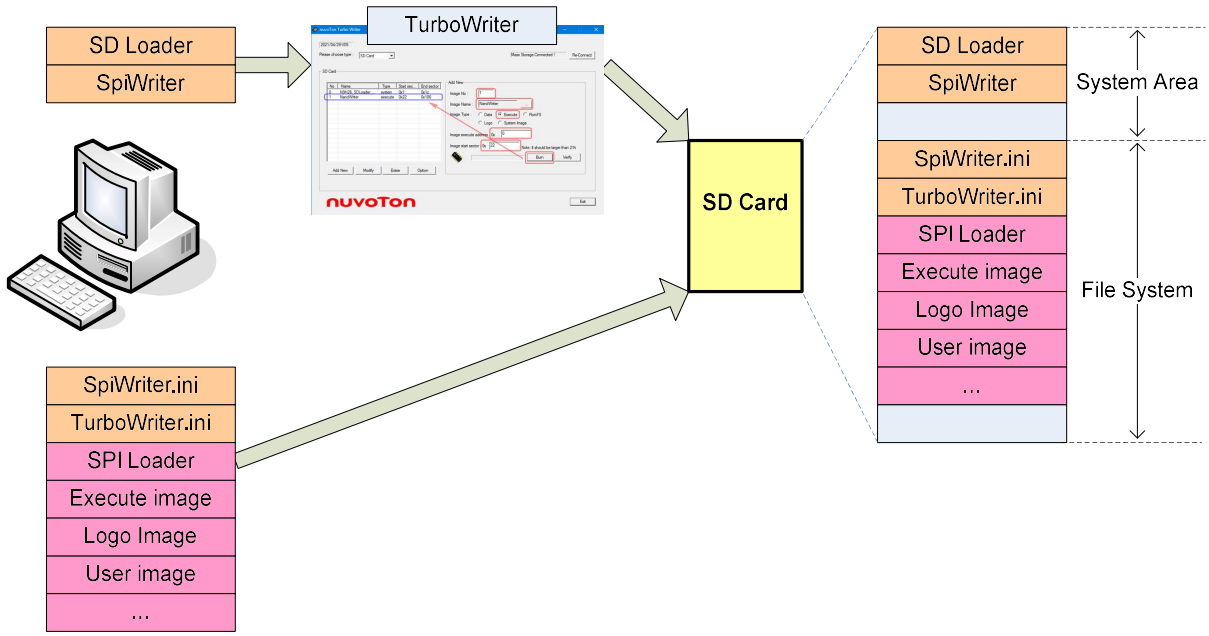

Figure 1 Prepare SD Card for SpiWriter

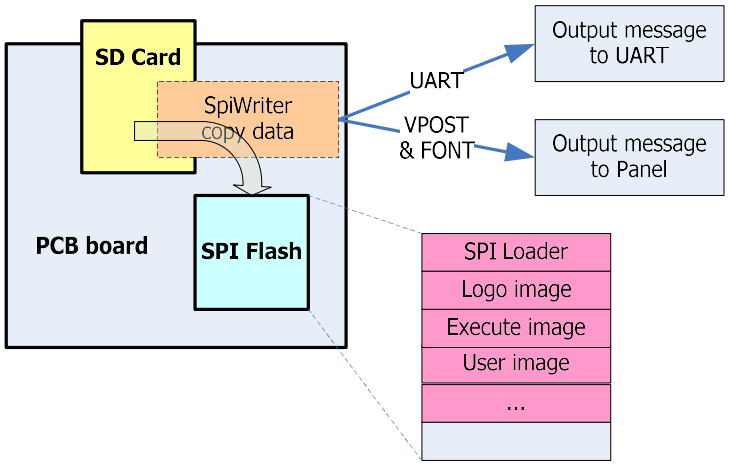

Figure 2 Execute SpiWriter to Copy Firmware to SPI Flash

Nuvoton provides SpiWriter source code within the N9H26 series microprocessor (MPU) BSP.

2 SpiWriter BSP Directory Structure

This chapter introduces the SpiWriter related files and directories in the N9H26 BSP.

2.1 Tools\MassProduction_Tools\SpiWriter

Doc	The document of the SpiWriter.
Ini	The sample for the SpiWriter initial file.
Src	The source code of the SpiWriter.
PCM_Pattern	The audio file of the SpiWriter.

2.2 Tools\MassProduction_Tools\SpiWriter\Src

GCC	The GCC project files for the SpiWriter.
KEIL	The KEIL project files for the SpiWriter.
Writer.c	The main function for the SpiWriter.
Writer.h	The header file for the SpiWriter.
ProcessIni.c	The function to process *NandWriter.ini* initial file.
ProcessOptionalIni.c	The function to process *TurboWriter.ini* initial file.
DrawFont.c	The functions to draw font on panel.
pcm.h	The header file for the audio function.
playsound.c	The functions to play sound.
spiflash.c	Special NAND Flash driver for the SpiWriter.
Rootkey.c	Root key Generator for authentication Flash

2.3 Tools\MassProduction_Tools\SpiWriter\Ini

SpiWriter.ini	The sample initial file for SpiWriter.

2.4 Binary file

NandWriter.bin	The binary file of the NAND writer.

3 SpiWriter Source Code

Complete source code is included in the *N9H26 BSP Tools\MassProduction_Tools\SpiWriter* directory:

3.1 Development Environment

Keil IDE and Eclipse are used as Non-OS BSP development environment, which uses J-Link ICE or ULINK2 ICE (optional) for debugging. This document uses Keil IDE to describe the project structure. To support ARM9, MDK Plus or Professional edition shall be used.

Note that Keil IDE and ICE need to be purchased from vendor sources.

Feature	MDK Edition			
	Professional	Plus	Essential	Lite
	All-in-one solution including Middleware	Supports all microcontroller cores and Middleware	Supports selected Cortex-M	Free with code size limit: 32 KBytes
Device Support				
Arm Cortex-M0/M0+/M3/M4/M7	✔	✔	✔	✔
Arm Cortex-M23/M33 Non-secure only	✔	✔	✔	✘
Arm Cortex-M23/M33 Secure and non-secure	✔	✔	✘	✘
Armv8-M Architecture Models including FastModel	✔	✘	✘	✘
Arm SecurCore®	✔	✔	✘	✘
Arm7™, Arm9™, Arm Cortex-R4	✔	✔	✘	✘

Figure 3 Keil MDK License Chart

3.2 Project Structure

The SpiWriter project includes one main function file and some driver files of N9H26.

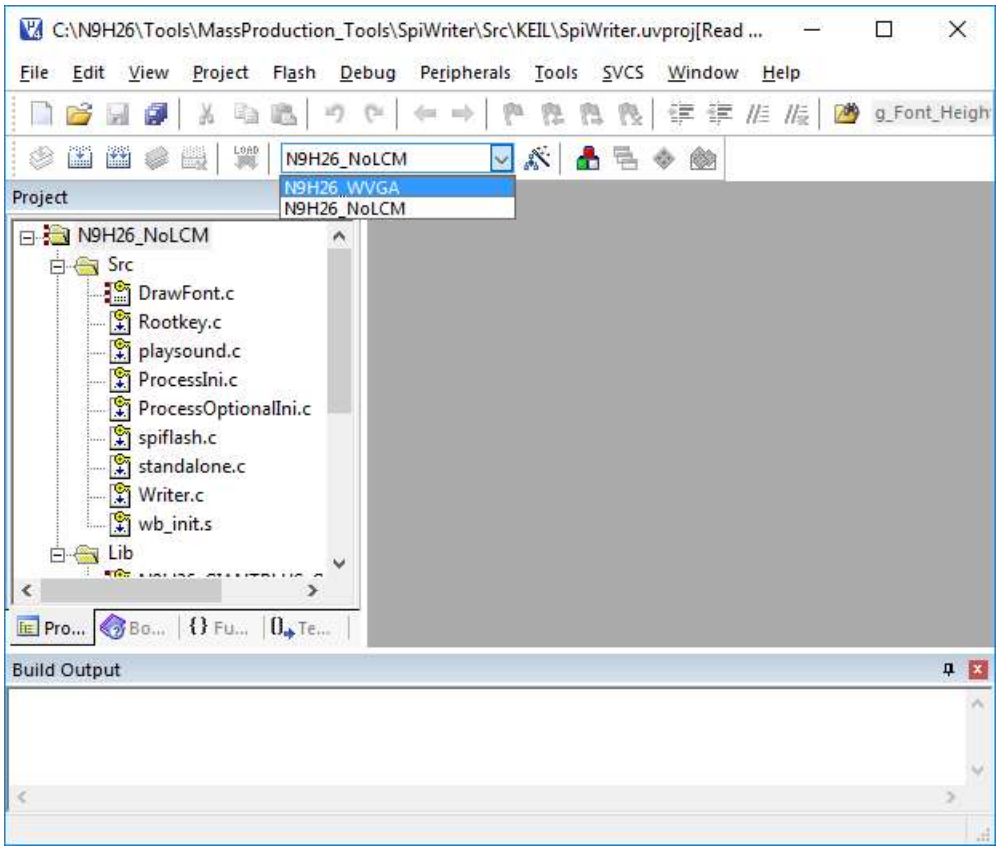

Figure 4 SPI Loader Project Tree on Keil MDK

The SpiWriter project includes some targets that can be used in different situations.

- **N9H26_NoLCM**: Only show the writing result by UART.
- **N9H26_WVGA**: Initialize VPOST for WVGA panel. This is the official standard target.

3.3 Build SpiWriter Project

Normally, the SpiWriter doesn't need to modify. If the SpiWriter is modified, clicking the **Rebuild** icon as shown below or press **F7** function key to rebuilt it in Keil MDK.

Figure 5 Shortcut Icon to Rebuild the SpiWriter on Keil MDK

The binary file of SpiWriter will be created with the file name *SpiWriter_xxx.bin*. The "*xxx*" is depend on the project target. For the **N9H26_WVGA** project target, the binay file name is *N9H26_SpiWriter_WVGA.bin*.

4 Settings

4.1 SD Card 0

The SD card 0 must reserve some space to store the SDLoader and SpiWriter before usage. The procedure is as below step:

1. Launch TurboWriter in recovery mode and set the system Reserved Area Size if this SD card does not do it before.
2. Burn the SDLoader binary file – *N9H26_SDLoader_240MHz_Fast.bin* as system image.
3. Burn the SpiWriter binary file – *N9H26_SpiWriter_WVGA.bin* as execute image with "Image execute address" 0.

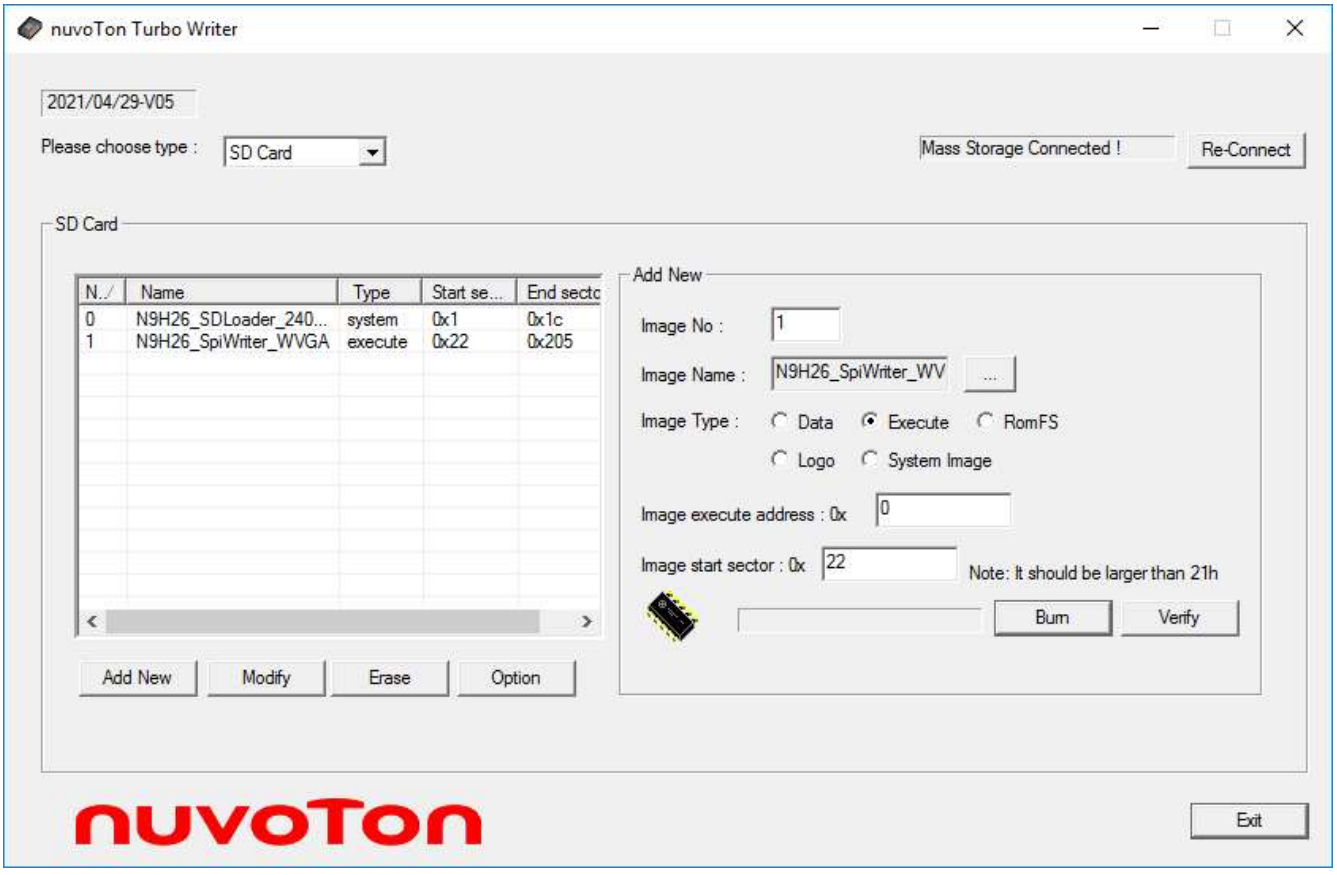

Figure 6 Burn SpiWriter

4. Reserve System area 2 MB and Enable SD format.
   These two files are burned in System reserved area and unable to read from card reader.

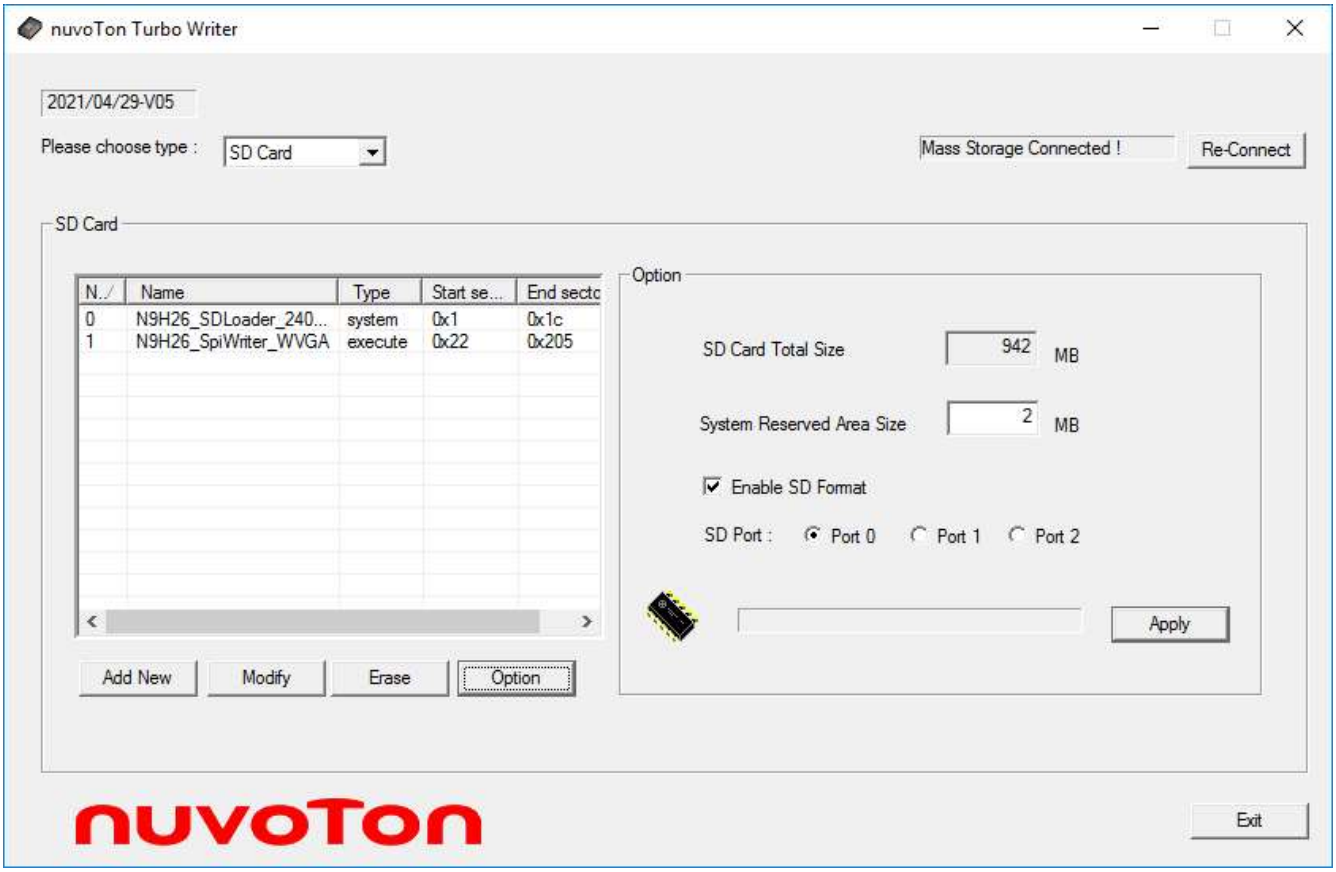

Figure 7 Set System Reserved Area Size and Format SD Card

Put this SD card to another card reader and copy SpiWriter.ini and related files that want to burn to SPI Flash from this SD card.

This SD card content structure is as below figure. The root directory contains the SpiWriter.ini (must), SpiLoader binary file/Raw data Image file (must), Logo binary file (option), execute binary file (option). It also provides some option in SpiWriter.ini for user. Further information about file setting can be found at INI File section.

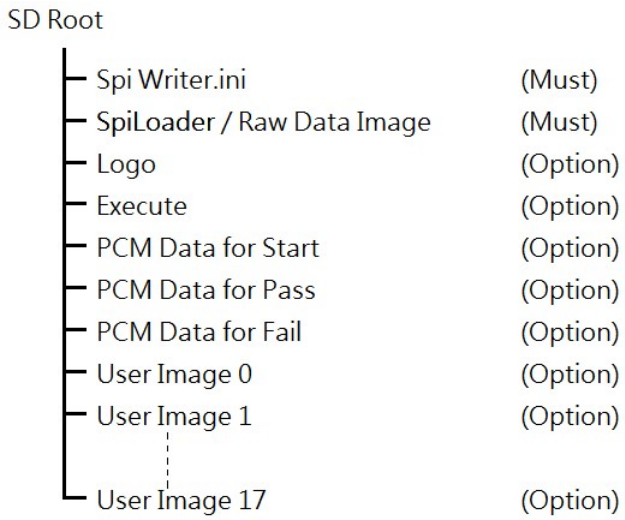

Figure 8 Files in SD card

4.2 *SpiWriter.ini*

The INI file means *SpiWriter.ini* file that provides user a flexible way to change program setting without modifying the source code of *SpiWriter.bin*. It describes the files user wants to write to SPI Flash in the SD Card.

The *SpiWriter.ini* file provides some sections as below):

```
[Raw Data Mode File Name]
//Format:File name
//If SpiWriter gets the file name, it ignores other files and only write the file.
//RawDataImage.bin

[SpiLoader File Name]
//Format:File name, execute address
N9H26_SpiLoader_240MHz_FW050TFT_800x480_24B.bin, 0x900000

[Logo File Name]
// Format:File name, Start Block (0 is decided by writer), execute address
Logo.dat, 0x0, 0x500000

[Execute File Name]
// Format:File name, Start Block (0 is decided by writer), execute address
conprog.bin, 0x0, 0x0000000

[Data Verify]
```

```
// 1 to verify data, 0 to not to verify.
1

[Chip Erase]
// 1 to erase whole SPI Flash, 0 to erase the sector which will be used.
0

[Sound Play]
// 1 to enable sound play, 0 to disable sound play.
1

[Sound Play Volume]
// Maximum is 63
32

[Write Root]
//Index is from 1 to 4, 0 to disable Write Root Key
0

[Flow Status Sound File Name - Start]
//StartBurn.pcm

[Flow Status Sound File Name - Pass]
//BurnComplete.pcm

[Flow Status Sound File Name - Fail]
//BurnFail.pcm

// Format:GPIO Pin, Active Level, Example:GPA6, 1
[GPIO Flow Status - Start]

[GPIO Flow Status - Pass]

[GPIO Flow Status - Fail]

// Format for User Image:File name, Start Block (0 is decided by writer)
// Ex.Image0.bin, 0x46 (Start block is 70)
// Maximum User Image Files is 18.
[User Image File name]
Image0.bin, 0x0
```

```
[User Image File name]
Image1.bin, 0x0

[User Image File name]
Image2.bin, 0x0

[User Image File name]
Image3.bin, 0x0

[User Image File name]
Image4.bin, 0x0
```

Due to its limited parsing ability of *SpiWriter.bin*, there are some constraints in SpiWriter.ini as below:

- No space is allowed to precede the option for each line.
- Only "//" comment is allowed at the beginning of each line
- String in "[ ]" is not allowed to changed.
- Only "[SpiLoader File Name]" is necessary. The other sections are optional.

If the logo file is not necessary for the SpiWriter, below two methods are to skip burning *Logo.dat* into the target SPI Flash.

```
[Logo File Name]
//Logo.dat, 0x0, 0x500000
```

Or

```
[Logo File Name]
```

It also allows changing the file name for burning. Below sample changes the file name from *N9H26_SpiLoader_240MHz_FW050TFT_800x480_24B.bin* to *Nuvoton.bin* for "[SpiLoader File Name]" section.

```
[SpiLoader File Name]
Nuvoton.bin, 0x900000
```

4.2.1 Raw Data Mode File Name

The format for this setting is:

[Raw Data Mode File Name]

File name

User can modify the file name by this setting. For example,

```
[Raw Data Mode File Name]
//Format:File name
//If SpiWriter gets the file name, it ignores other files and only write the file.
RawDataImage.bin
```

4.2.2 Raw Data Mode File Name

The format for this setting is:

[SpiLoader File Name]

File name, execute address

User can modify loader file name and execute address by this setting. For example, Loader execute address is 0x900000

```
[SpiLoader File Name]
//Format:File name, execute address
N9H26_SpiLoader_240MHz_FW050TFT_800x480_24B.bin, 0x900000
```

4.2.3 Logo File Name

The format for this setting is:

[Logo File Name]

File name, start block, execute address

User can modify logo file name, SPI Flash start block, and execute address by this setting. If user want to burn logo binary file right after the previous image, user needs to set the start block to 0. For example, Logo execute address is 0x500000

```
[Logo File Name]
// Format:File name, Start Block (0 is decided by writer), execute address
Logo.dat, 0x0, 0x500000
```

For example, Logo execute address is 0x500000 and start block is 2.

```
[Logo File Name]
// Format:File name, Start Block (0 is decided by writer), execute address
Logo.dat, 0x2, 0x500000
```

4.2.4 Execute File Name

The format for this setting is:

[Execute File Name]

File name, start block, execute address

User can modify logo file name, SPI Flash start block, and execute address by this setting. If user wants to burn execute binary file right after the previous image, user needs to set the start block to 0. For example, Burn execute image – *demo.bin* right after the previous image and execute address is 0x100000

```
[Execute File Name]
// Format:File name, Start Block (0 is decided by writer), execute address
demo.bin, 0x0, 0x1000000
```

For example, Burn execute image – *conprog.bin* from block 9 and execute image is 0x00000000

```
[Execute File Name]
// Format:File name, Start Block (0 is decided by writer), execute address
conprog.bin, 0x9, 0x0000000
```

4.2.5 Data Verify

The format for this setting is:

[Data Verify]

0/1

User can enable or disable Data verify operation by setting. Disable Data verify operation can speed up the SPI writing operation. (If there is no such setting, the default setting is disabled). For example, Disable Data verify operation.

```
[Data Verify]
// 1 to verify data, 0 to not to verify.
0
```

4.2.6 Chip Erase

The format for this setting is:

[Chip Erase]

0/1

User can enable or disable Chip Erase operation by setting. If Chip Erase Operation is disabled, SpiWriter will do sector erase to erase the spi sectors that will be wrote. (If there is no such setting, the default setting is disabled). For example, Disable Data verify operation.

```
[Chip Erase]
// 1 to erase whole SPI Flash, 0 to erase the sector which will be used.
0
```

4.2.7 Sound Play

The format for this setting is:

[Sound Play]

0/1

SpiWriter can play PCM data when Burn Start/Burn Pass/Bun Fail and it has three built-in PCM data. User can enable or disable Sound Play operation by setting. (If there is no such setting, the default setting is disabled). For example, Enable Sound Play operation.

```
[Sound Play]
// 1 to enable sound play, 0 to disable sound play.
1
```

4.2.8 Sound Play Volume

The format for this setting is:

[Sound Play Volume]

0/1

SpiWriter can control Sound Play operation Volume by setting. The minimum value is 0 and maximum is 63. (If there is such setting, the default setting is 31). For example, Set Sound Play operation Volume to maximum Volume

```
[Sound Play Volume]
// Maximum is 63
63
```

4.2.9 Flow Status Sound File Name

The format for this settings are:

[Flow Status Sound File Name - Start]

File name

[Flow Status Sound File Name - Pass]

File name

[Flow Status Sound File Name - Fail]

File name

SpiWriter allows user to use their own PCM data when Burn Start/Burn Pass/Bun Fail. User set the PCM fiel name for Burn Start/Burn Pass/Bun Fail stage by setting. (If there is no such setting, it will play default PCM data when Sound Play is enabled). For example, Enable Sound Play operation.

```
[Flow Status Sound File Name - Start]
StartBurn.pcm

[Flow Status Sound File Name - Pass]
BurnComplete.pcm

[Flow Status Sound File Name - Fail]
BurnFail.pcm
```

4.2.10 GPIO Flow Status

The format for this settings are:

[GPIO Flow Status - Start]

GPIO pin, Active Level

[GPIO Flow Status - Pass]

GPIO pin, Active Level

[GPIO Flow Status - Fail]

GPIO pin, Active Level

SpiWriter allow user to control GPIO pin for status check when Burn Start/Burn Pass/Bun Fail. SpiWriter will set the pin function setting for the setting and set the GPIO output level to inactive level after getting the setting. User can select GPIO pin and active level by this setting. (If threre is no such setting, the default setting is disabled). For example, Set GPB11/GPA6/GPA7 for Status Check

```
// Format:GPIO Pin, Active Level, Example:GPA6, 1
[GPIO Flow Status - Start]
GPB11, 0

[GPIO Flow Status - Pass]
GPA6, 1

[GPIO Flow Status - Fail]
GPA7, 1
```

4.2.11 User Image File name

The format for this setting is:

[User Image File name]

File name, start block

User can set User image file name and SPI Flash start block by this setting. SpiLoader allow user to set 18 User image files. If user wants to burn binary file right after the previous image, user needs to set the start block to 0. For example, Burn 4 User Image files, and set the start block to 0x46, 0x4B, 0x4D, and 0x52 for them.

```
// Format for User Image:File name, Start Block (0 is decided by writer)
// Ex.Image0.bin, 0x46 (Start block is 70)
// Maximum User Image Files is 18.
[User Image File name]
Image0.bin, 0x46

[User Image File name]
Image1.bin, 0x4B
```

```
[User Image File name]
Image2.bin, 0x4D

[User Image File name]
Image3.bin, 0x52
```

4.3 *TurboWriter.ini*

SpiWriter supports the same function as Turbowriter to provide writing some user configuration setting in the boot code header to adjust the setting of N9H26. Please use the same *TurboWriter.ini* file that used by Turbowriter for SpiWriter. Please obtain the *TurboWriter.ini* file from the *TurboWriter* folder for the target N9H26 platform.

The *TurboWriter.ini* file provides some sections as below:

```
 [ADDRESS]
  ADDRESS = 00900000
[CLOCK_SKEW]
  DQSODS = 00001010
  CKDQSDS = 00888800

[N9H26 USER_DEFINE]
  b0003010 = 00000006
  b0000204 = FFFFFFFF
  b0000208 = FFFFFFFF
  b0000228 = 00001150
  b0000224 = 000011D0

  /* Omit some source code in document. */

  b0003000 = 05230476
```

5 Operation

When the SD card 0 is prepared successfully and booting from Normal mode, it will show the target SD card burning status on the panel as below:

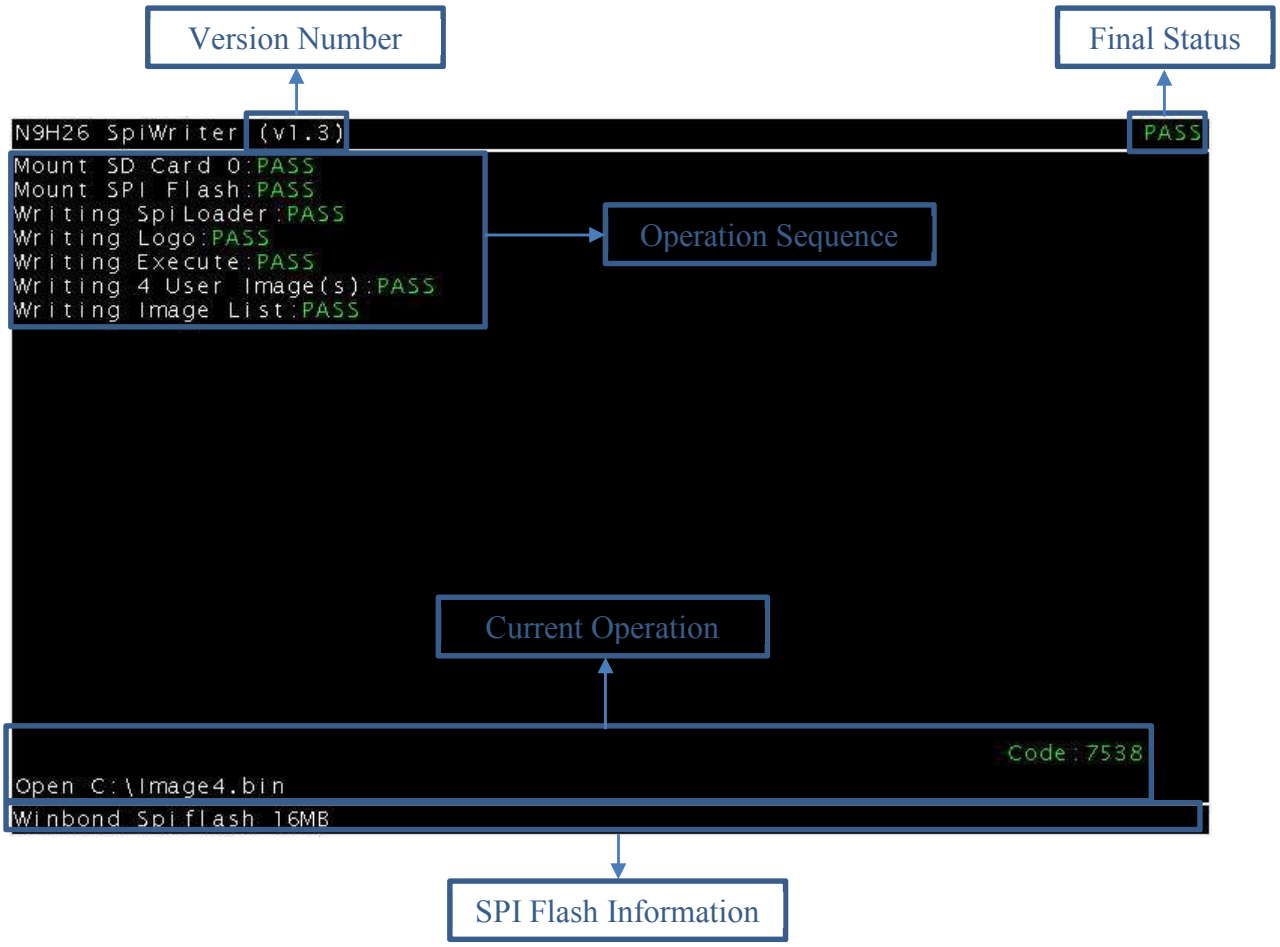

Figure 9 SpiWriter Display message

It can divide into several parts:

- Version Number: show this version number.
- Final Status: show the final operation status. If there is any fail items in the operation sequence, the final Status will be “FAIL”.
- Operation Sequence: show the current operation progress.
- Current Operation: show more detail information for current operation. For example, it fails for some function, the code will show the return code for this.
- The flash and Fail operation Information: shows SPI Flash size and Fail operation status.

Figure 10 SpiWriter Fail Case

6 Sample

Here is the result of *SpiWriter.ini* in section 4.2

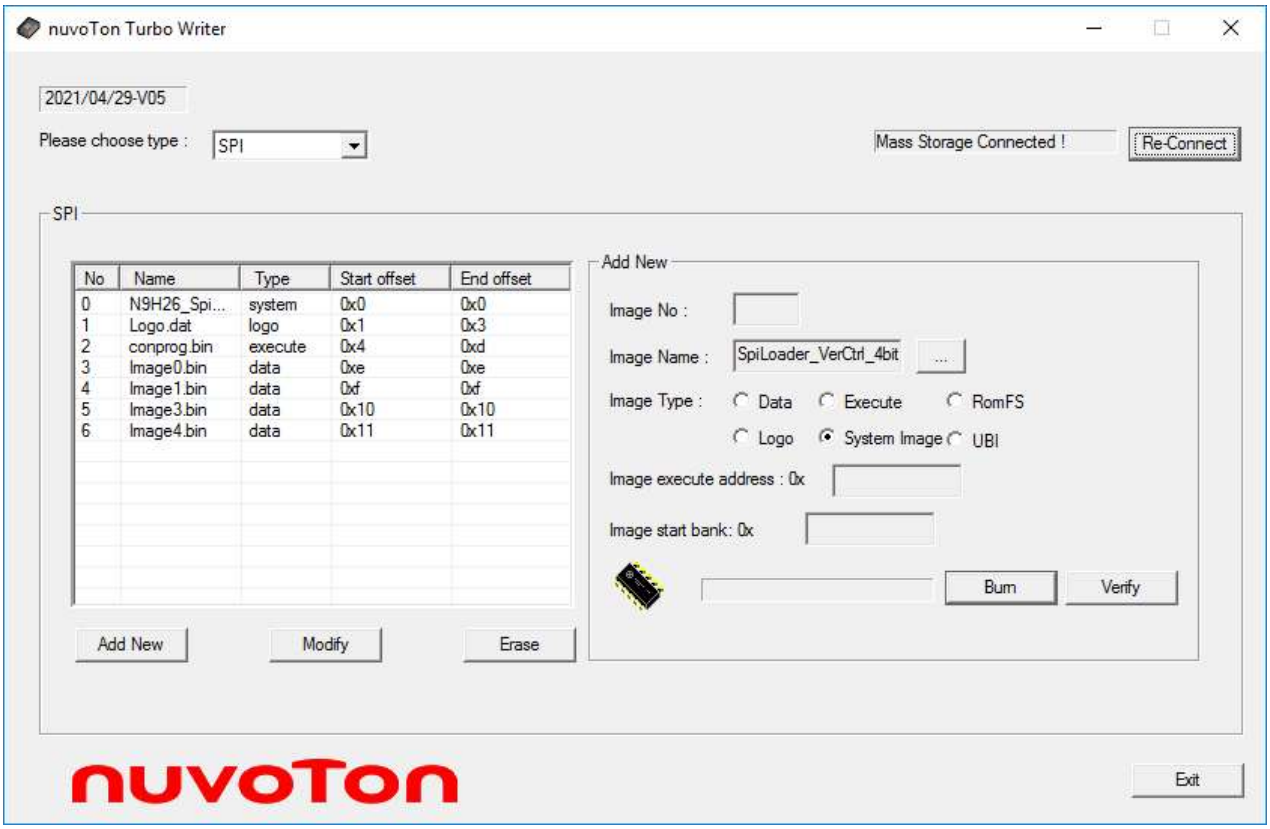

Figure 11 SpiWriter Result

7 Supporting Resources

The N9H26 system related issues can be posted in Nuvoton's forum:

- ARM7/9 forum at: http://forum.nuvoton.com/viewforum.php?f=12.

Revision History

Date	Revision	Description
2021.6.8	1.01	1. Modify document structure.
2013.8.5	1.00	1. Initially issued.

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.