How to Mount FAT in SPI Flash

V1.1

Publication Release Date: May. 2018

Support Chips:

Support Platforms:

The information in this document is subject to change without notice.

The Nuvoton Technology Corp. shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

This documentation may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from the Nuvoton Technology Corp.

Nuvoton Technology Corp. All rights reserved.

Table of Contents

1. How to generate the SPI image for FAT 4

1.1. Generate FAT image for SPI flash 4
1.2. Burn FAT image in SPI flash 5
1.3. SPI flash Layout 6
1.4. Code example 7
1.5. The other solution 9

2. Mass Storage on SPI 11

2.1. Partition SPI flash by Mass Storage 11

3. SPI Flash Support 12

3.1. How to add the SPI Flash support list 12
3.2. Exported DISK SIZE in Mass Storage 13
3.3. Additional note for SPI flash 14

4. Q & A 16

5. Revision History 17

1. How to generate the SPI image for FAT

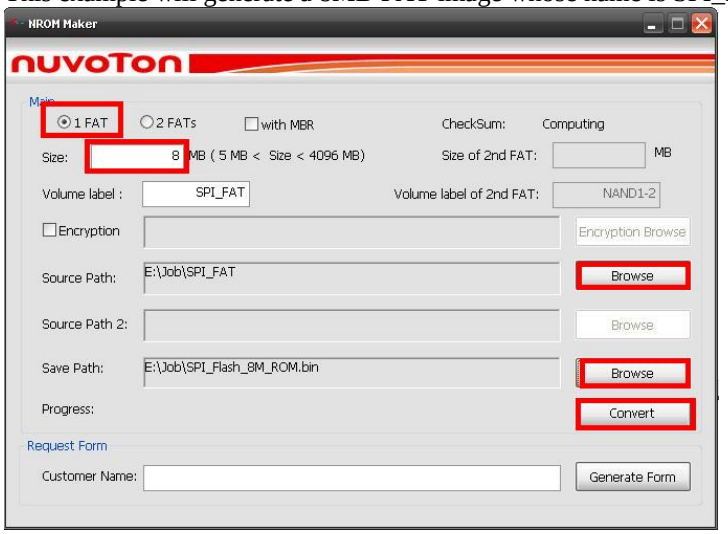
1.1. Generate FAT image for SPI flash

The FAT image for SPI flash is generated by NROMMakder.

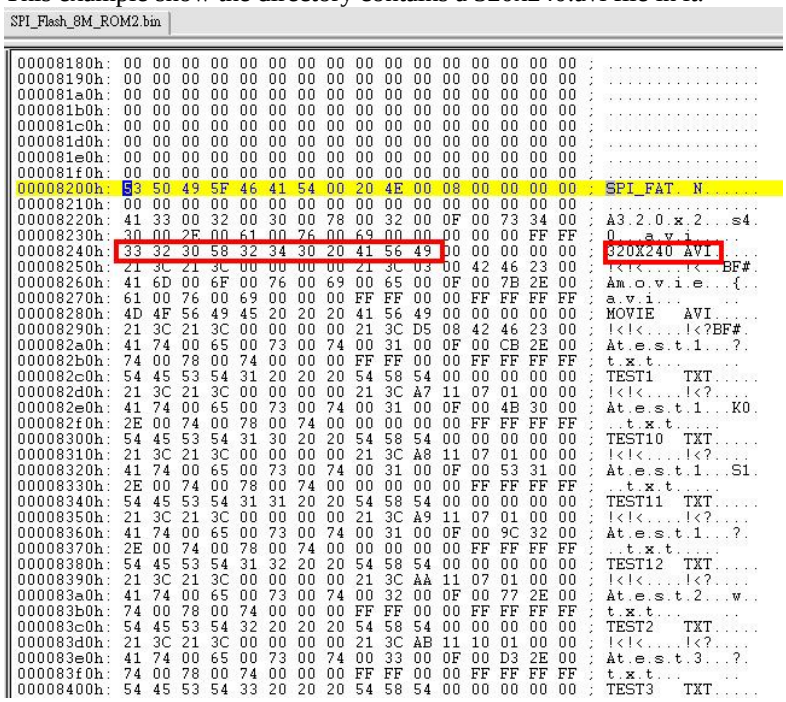
Step 1. Put all the files which you need in one folder. This example puts all the files at SPI_FAT folder.

Step 2 : Specify the source path and save paths to generate the FAT image. Specify correct size for your FAT image. It will depend on you SPI flash and total file size.

This example will generate a 8MB FAT image whose name is SPI_Flash_8M.bin



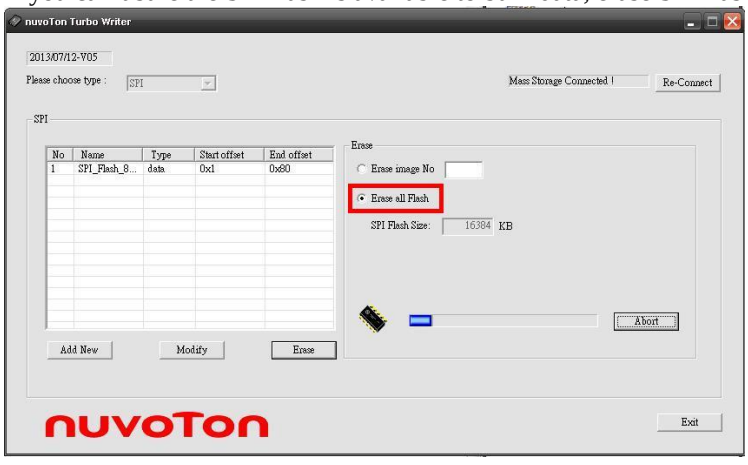
When the FAT image is generated, you can find the root directory will like below picture which contains all file name. This example show the directory contains a 320x240.avi file in it.



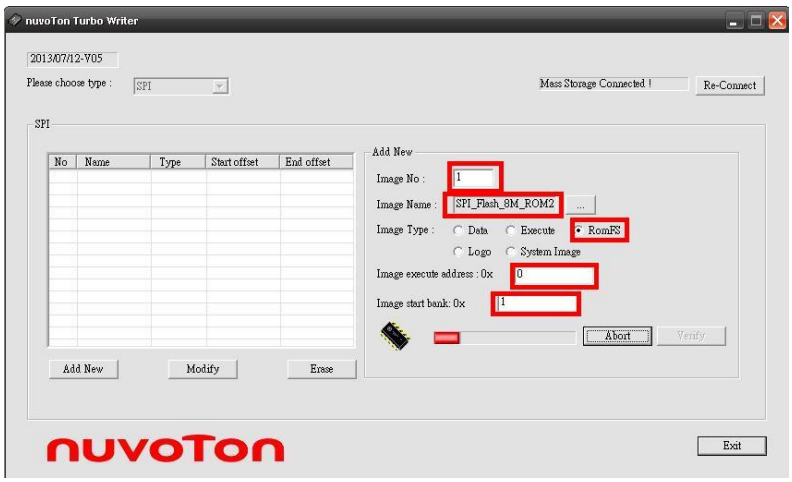
1.2. Burn FAT image in SPI flash

Use TurboWriter to burn the FAT image in SPI flash.

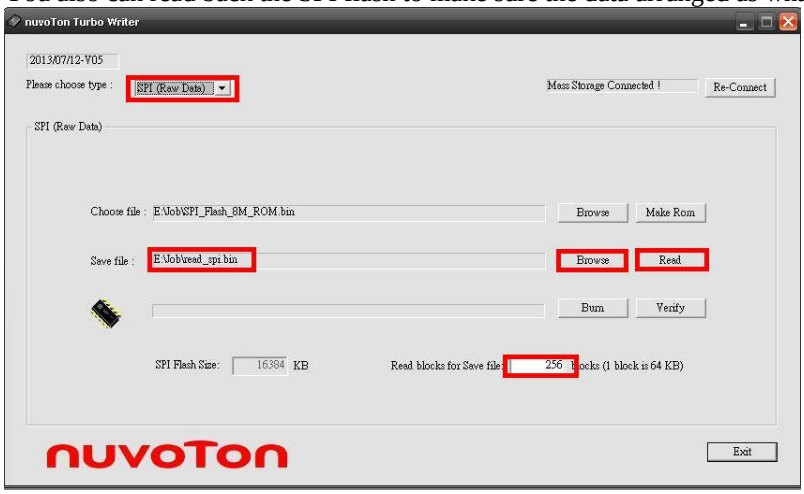
If you can't sure the SPI flash is available to burn data, erase SPI flash is recommend before you use it.



Specify the FAT image generated in section 1.1 on "Image Name". "Image Type" can select "Data" or "RomFS". Don't care the "Image execution address.". Any value for "Image execute address" is available.
Be care of the "Image start bank". One bank size is **64KB** in TurboWriter. It specifies the FAT offset from SPI address 0.



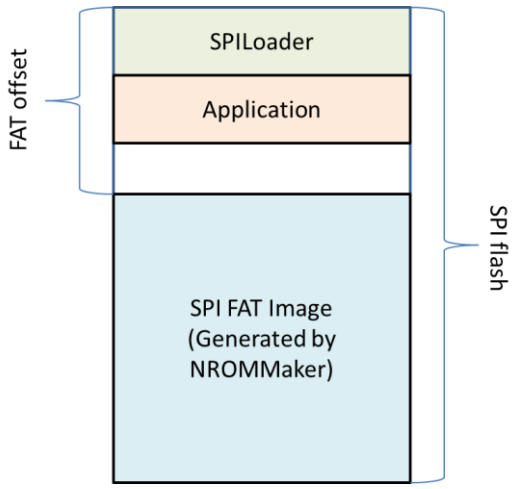
You also can read back the SPI flash to make sure the data arranged as what you want.



The other code, for example SPILoader or others, is burned according to normal operation for SPI flash.

1.3. SPI flash Layout

The SPI flash layout is as below.



1.4. Code example

```
int main()
{
    WB_UART_T     uart;
    LCDFORMATEX lcdformatex;
    CHAR     suFileName[128];
    INT           nStatus;

    /* CACHE_ON   */
    sysEnableCache(CACHE_WRITE_BACK);
    /*-------------------------------------------------------------------
*/
    /*  CPU/AHB/APH:  200/100/50                                        */
    /*-------------------------------------------------------------------
*/
    sysSetSystemClock(eSYS_UPLL,    //E_SYS_SRC_CLK eSrcClk,
                           192000,       //UINT32 u32PllKHz,
                           192000,       //UINT32 u32SysKHz,
                            96000,       //UINT32 u32CpuKHz,
                            96000,       //UINT32 u32HclkKHz,
                            48000);      //UINT32 u32ApbKHz

    /*-------------------------------------------------------------------
*/
    /*  Init UART, N,8,1, 115200                                        */
```

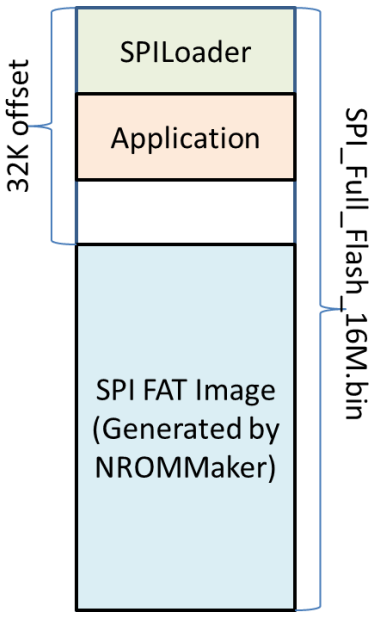
```
    /*---------------------------------------------------------------------
*/
    sysUartPort(1);
    uart.uiFreq = 12000000;                      //use XIN clock
   uart.uiBaudrate = 115200;
   uart.uiDataBits = WB_DATA_BITS_8;
   uart.uiStopBits = WB_STOP_BITS_1;
   uart.uiParity = WB_PARITY_NONE;
    uart.uiRxTriggerLevel = LEVEL_1_BYTE;
   sysInitializeUART(&uart);
    sysprintf("UART initialized.\n");

    _VpostFrameBuffer = (UINT8 *)((UINT32)_VpostFrameBufferPool | 0x80000000);

    /*---------------------------------------------------------------------
*/
    /*  Init timer                                                      */
    /*---------------------------------------------------------------------
*/
    sysSetTimerReferenceClock (TIMER0, 60000000);
    sysStartTimer(TIMER0, 100, PERIODIC_MODE);

    /*---------------------------------------------------------------------
*/
    /*  Init FAT file system                                            */
    /*---------------------------------------------------------------------
*/
    sysprintf("fsInitFileSystem.\n");
    fsInitFileSystem();

    /*---------------------------------------------------------------------
*/
    /*  Init SPI Flash                                                  */
    /*---------------------------------------------------------------------
*/
    SpiFlashOpen(64*1024);// 1 bank = 64K in TurboWriter

    fsAssignDriveNumber('C', DISK_TYPE_SD_MMC, 0, 1);

    spuOpen(eDRVSPU_FREQ_8000);
    spuDacOn(1);
```

```
    /*-------------------------------------------------------------------
*/
    /*                                                                 */
    /*  Direct RGB565 AVI playback                                         */
    /*                                                                 */
    /*-------------------------------------------------------------------
*/
    lcdformatex.ucVASrcFormat = DRVVPOST_FRAME_RGB565;
    lcdformatex.nScreenWidth = 320;
    lcdformatex.nScreenHeight = 240;
   vpostLCMInit(&lcdformatex, (UINT32 *)_VpostFrameBuffer);

    fsAsciiToUnicode("C:\\320x240.AVI", suFileName, TRUE);

    if (aviPlayFile(suFileName, 0, 0, DIRECT_RGB565, avi_play_control) < 0)
        sysprintf("Playback failed, code = %x\n", nStatus);
    else
        sysprintf("Playback done.\n");

    while(1);
}
```

1.5. The other solution

If the customer thinks the reserved area before FAT image being tool large, customer must generate the whole image and burn it as data in section 1.2.

For example, if the SPI flash is 16MB. Customer can reserve only 32KB before FAT image. However, customer must change the argument of SpiFlashOpen(…) accordingly.


SPILoader
Application
SPI FAT Image
(Generated by
NROMMaker)
32K offset
SPI_Full_Flash_16M.bin

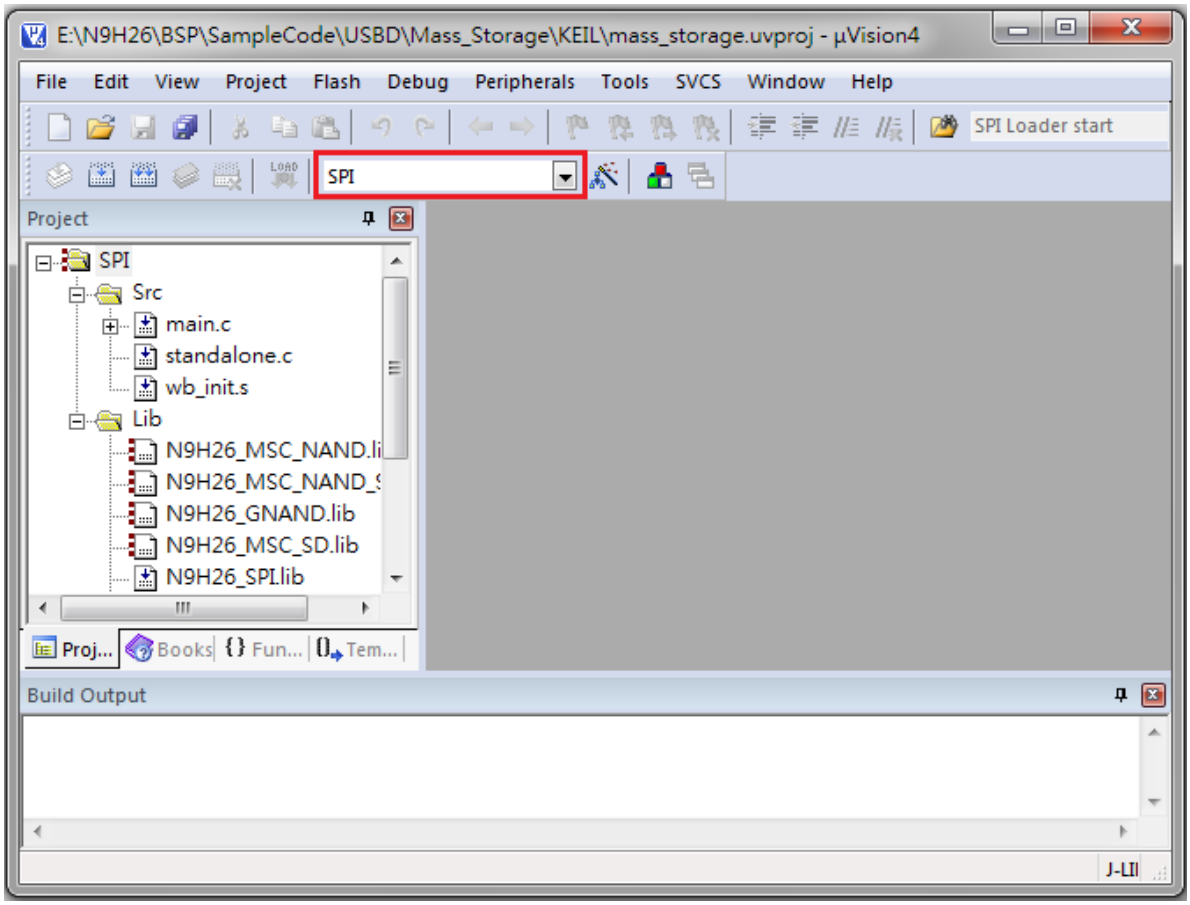
nuvoTon Turbo Writer
2013/07/12-V05
Please choose type :
SPI (Raw Data)
Mass Storage Connected !
Re-Connect
SPI (Raw Data)
Choose file
E:\Job\SPI_Full_Flash_16M.bin
Browse
Make Rom
Save file :
E:\Job\read_spi.bin
Browse
Read
Burn
Verify
SPI Flash Size:
16384
KB
Read blocks for Save file:
256
blocks (1 block is 64 KB)
nuvoTon
Exit

2. Mass Storage on SPI

2.1. Partition SPI flash by Mass Storage

If the SPI flash is solid for first time, you can run the SPI_Only target in mass_storage example under non-OS\UDC\example folder. This is another way to partition SPI flash besides writing SPI image mentioned in Chapter 1.

This example will partition SPI flash which size is provided by SPI library. If it is partitioned before, it will export to PC as removable disk when plug-in to PC. User can operate it as usually removable disk.



3. SPI Flash Support

3.1. How to add the SPI Flash support list

Current SPI flash ID is read by command 0x9F. Please check what the return value of command 0x9F is from the SPI flash spec. Then, adding the ID in MapFlashID(…) function and providing the capacity for this SPI flash. Below "case NewID" is where the new SPI flash needs to modify.

```
// Added SPI list here to expand support list
INT MapFlashID(UINT32 spiID, DISK_DATA_T* pInfo)
{
    switch(spiID)
    {
        case 0xC84016:
            strcpy(pInfo->vendor,"Giga Device");
            pInfo->totalSectorN = 4*1024*2;// Unit : K (4MB)
            pInfo->diskSize = pInfo->totalSectorN/2 - (spi_FAT_offset / 1024);

        case 0xC84017:
            strcpy(pInfo->vendor,"Giga Device");
            pInfo->totalSectorN = 8*1024*2;// Unit : K (8MB)
            pInfo->diskSize = pInfo->totalSectorN/2 - (spi_FAT_offset / 1024);
            break;
        case 0xC84018:
            strcpy(pInfo->vendor,"Giga Device");
            pInfo->totalSectorN = 16*1024*2;    // Unit : K (16MB)
            pInfo->diskSize = pInfo->totalSectorN/2 - (spi_FAT_offset / 1024);
            break;
        case 0xC22014:
            strcpy(pInfo->vendor,"Macronix");
            pInfo->totalSectorN = 1*1024*2;// Unit : K (1MB)
            pInfo->diskSize = pInfo->totalSectorN/2 - (spi_FAT_offset / 1024);
            break;
        case 0xEF3013:
```

```
            strcpy(pInfo->vendor,"Winbond");
            pInfo->totalSectorN = 8*1024*2;// Unit : K (8MB)
            pInfo->diskSize = pInfo->totalSectorN/2 - (spi_FAT_offset / 1024);
            break;
        case NewID:
            strcpy(pInfo->vendor,"NewSPI");
            pInfo->totalSectorN = 8*1024*2;// Unit : K (8MB)
            pInfo->diskSize = pInfo->totalSectorN/2 - (spi_FAT_offset / 1024);
            break;
        default :
            sysprintf("Unknown SPI ID\n");
            return 0;
            break;
    }
    sysprintf("Vendor = %s, Total size = %dK, FAT offset =%dK\n",pInfo->vendor,
pInfo->diskSize,spi_FAT_offset/1024);


    return 0;
```

After the modification, the SPI library needs to re-build. Then, it needs to re-build mass_storage example which links new SPI library to support new SPI flash.

3.2. Exported DISK SIZE in Mass Storage

Currently, the exported disk size is specified by the diskSize filed of DISK_DATA_T structure in MapFlashID(…) function. If user doesn't want to export all SPI flash to Mass Storage, user can adjust it. The default exported size is total capacity of SPI flash subtract FAT offset which is specified in SpiFlashOpen(FATOffset) function.

```
INT MapFlashID(UINT32 spiID, DISK_DATA_T* pInfo)
{
    switch(spiID)
    {
        case 0xC84016:
            strcpy(pInfo->vendor,"Giga Device");
            pInfo->totalSectorN = 4*1024*2;// Unit : K (4MB)
            pInfo->diskSize = pInfo->totalSectorN/2 - (spi_FAT_offset / 1024);
        break;
```

```
        ...
    }
```

```
INT main(VOID)
{
    ...
    #ifdef __SPI_ONLY__
    {
        UINT32 block_size, free_size, disk_size, reserved_size;
        PDISK_T  *pDiskList;

        sysSetTimerReferenceClock (TIMER0, u32ExtFreq*1000);
        sysStartTimer(TIMER0, 100, PERIODIC_MODE);

        fsInitFileSystem();
        fsAssignDriveNumber('C', DISK_TYPE_SD_MMC, 0, 1);

        reserved_size = 64*1024;       // SPI reserved size before FAT = 64K
        status = SpiFlashOpen(reserved_size);

        ...
    }

    ...
}
```

3.3. Additional note for SPI flash

The SPI library uses command 0x20 to erase 4K SPI sector size. If new SPI flash doesn't support such kind of command, it needs to check what kind of command is supported in this SPI flash and to modify the SPIGlue.c for the new command.

```
//Default use command 0x20 to erase 4K Sector
#define      SPI_SECTOR_ERASE_SIZE  4*1024
#define      SPI_ERASE_CMD          0x20
```

Due to this implement doesn’t handle bad sector on SPI flash, it may work abnormal if there is any bad sector in the SPI flash.

4. Q & A

Q1 : Does the SPI FAT support write function?
A1 : SPI FAT only support read function before Nov/2013. Write function supports in SPI library after Oct/2014.

Q2 : Does it can work with Linux SPI?
A2 : Linux SPI driver doesn't support FAT now.

Q3: Why calling the SpiFlashOpen(…) function will show "Unknown SPI ID" on Uart message?
A3: At this moment, it only lists some Flash ID which read through command 0x9F. Because there are new SPI flash device each year, you can add the mapping function for this Flash on MapFlashID(…) function of SPIGlue.c file.

Q4: Can it support 2 partitions in the solution?
A4 : If you want to support 2 partitions in SPI flash, you must generate the correct FAT image with 2 partitions and burn it into SPI flash correctly.

Q5: Why application can't find the file after it mounts the FAT image?
A5: Please check the argument of SpiFlashOpen(…) function. It is the FAT offset from the SPI flash address 0 and the unit is byte. It must match with the FAT image burned in SPI flash as section 1.3

Q6 : Is there any limitation for the FAT offset?
A6 : The FAT offset must be 512B alignment.

Q7: Is there any limitation for the SPI FAT application?
A7: This SPI FAT application doesn't handle bad sector. If there is any SPI bad sector, it can't be used.

Q8: What command must be supported for SPI flash?
A8: Command 0x20 (Erase sector) and 0x9F (Read ID) must be supported. If not, the SPI library must be modified as Chapter 3.

5. Revision History

Version	Date	Description
V1.0	May. 2018	● Created

Important Notice

Nuvoton products are not designed, intended, authorized or warranted for use as components in equipment or systems intended for surgical implantation, atomic energy control instruments, aircraft or spacecraft instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for any other applications intended to support or sustain life. Furthermore, Nuvoton products are not intended for applications whereby failure could result or lead to personal injury, death or severe property or environmental damage.

Nuvoton customers using or selling these products for such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from their improper use or sales.