How to Mount FAT in SPI Flash

Document Information

Abstract	Introduce how to mount FAT in SPI Flash 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

The document describes how to mount FAT in SPI Flash.

2 How to generate the SPI image for FAT

2.1 Generate FAT Image for SPI Flash

The FAT image for SPI Flash is generated by NROMMaker. It can be found at BSP *Tools/MassProduction_Tools/NRomMaker*.

- Put all the files which user need in one folder. This example puts all the files at SPI_FAT folder.

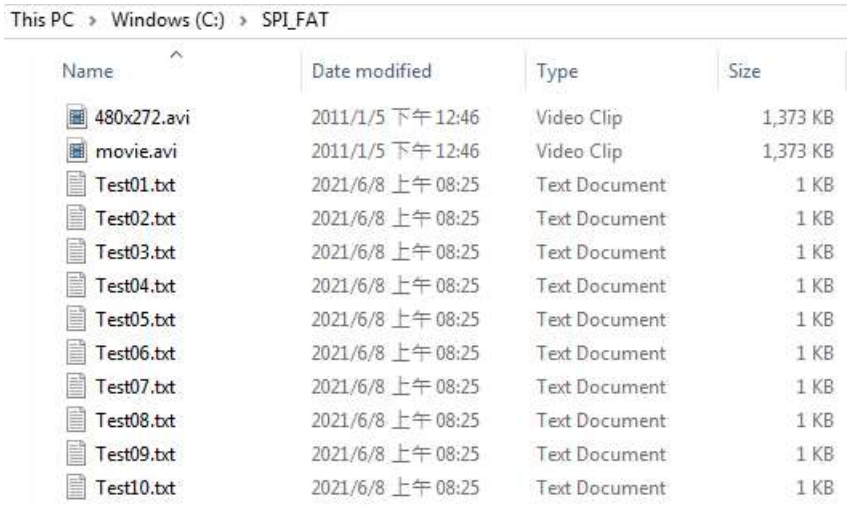

Figure 1 SPI_FAT folder

- This example will generate an 8 MB FAT image whose name is *SPI_Flash_8M_ROM.bin*.
  1. Set FAT partition number.
  2. Specify correct size for the FAT image (depend on the SPI Flash).
     - Less than SPI Flash size.
     - Larger than total file size.
  3. Set Volume Label.
  4. Specify the source path.
  5. Specify the save paths to generate the FAT image.
  6. Press "**Convert**" to generate the FAT image.

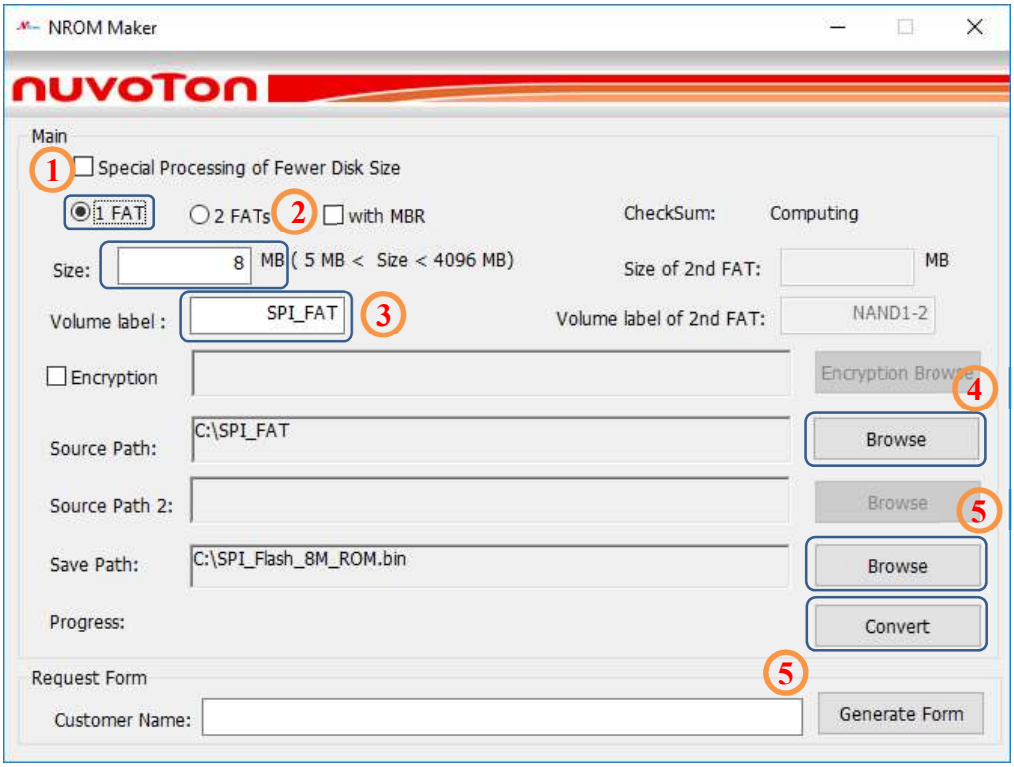

Figure 2 NROMMaker

When the FAT image is generated, user can find the root directory will like below picture which contains all file name. This example show the directory contains a *480x227.avi* file in it.

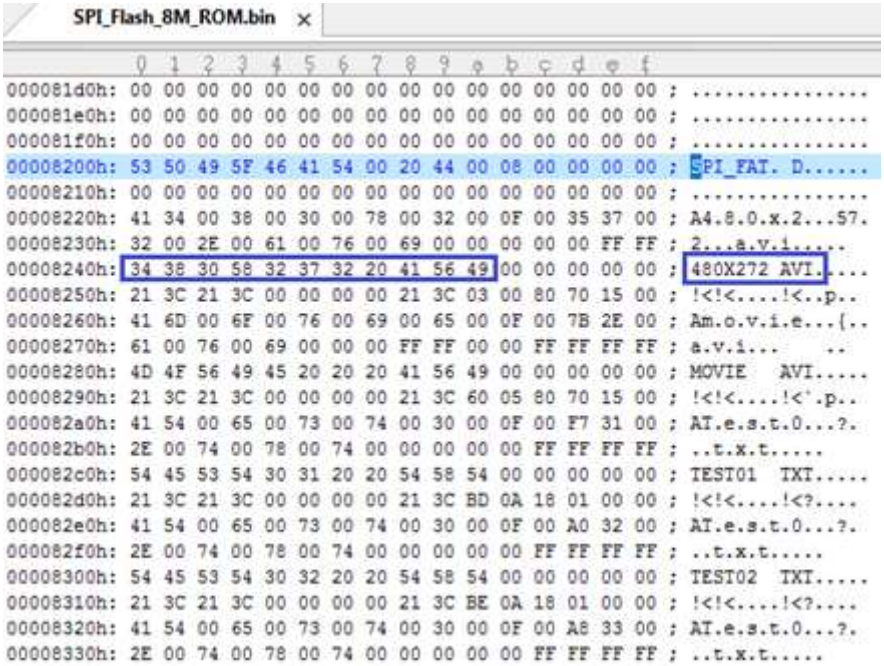

Figure 3 *SPI_Flash_8M_ROM.bin*

2.2 Burn FAT image in SPI Flash

User can burn the FAT image to SPI Flash by TurboWriter. If user can't sure the SPI Flash is available to burn data, erase SPI Flash is recommended before you use it.

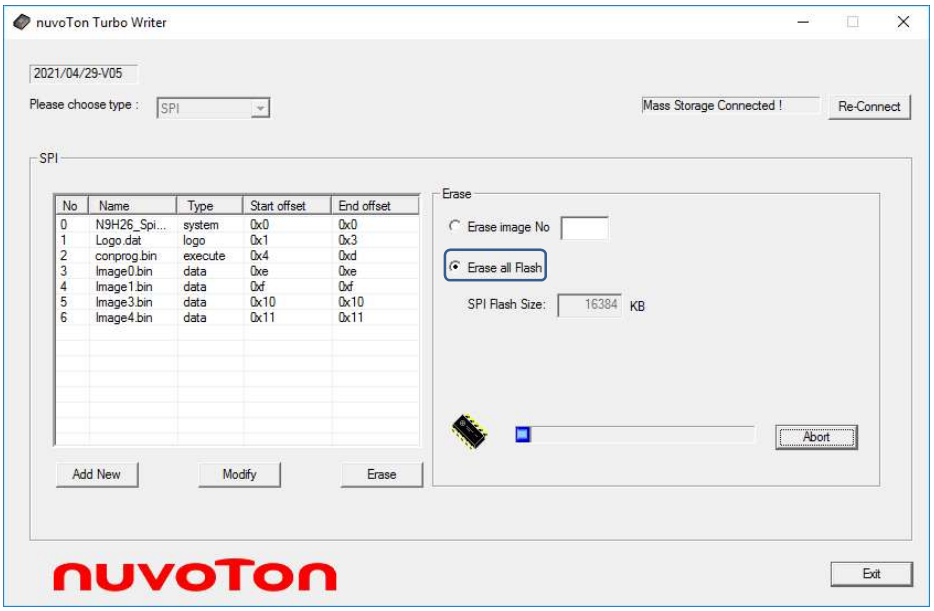

Figure 4 Erase all Flash

Burn the FAT image generated in section 2.1.

1. Press the button "**Add New**"
2. Browse the file "*SPI_Flash_8M_ROM.bin*"
3. Set Image type "**RomFS**
4. Set the image execute address: Any value
5. Set the start block number: 0x1
   - It specifies the FAT offset in SPI Flash
6. Press the button "**Burn**"

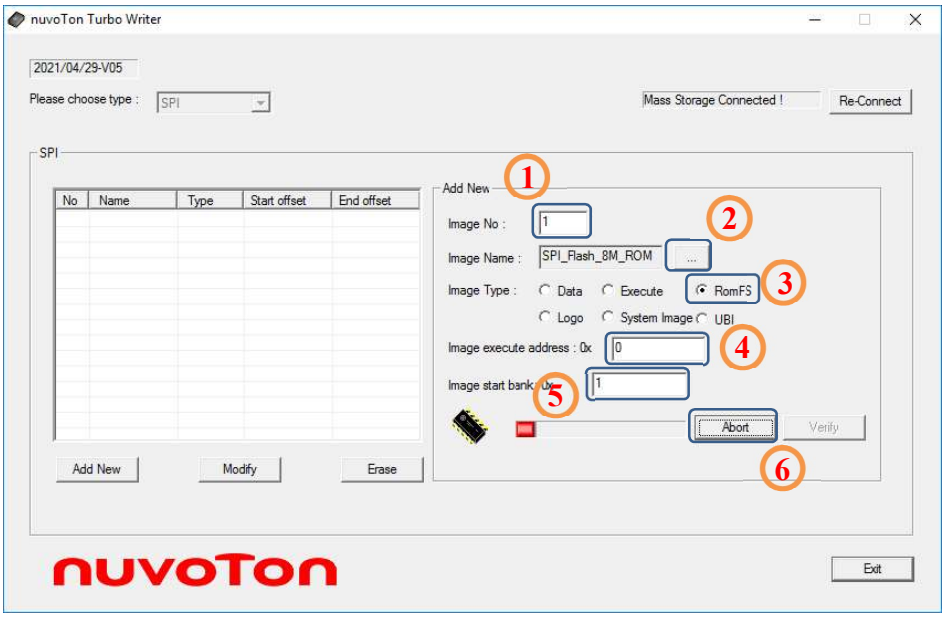

Figure 5 Burn the FAT image

The other code, for example SPI Loader or others, is burned according to normal operation for SPI Flash. User also can read back the SPI Flash to make sure the data arranged correct.

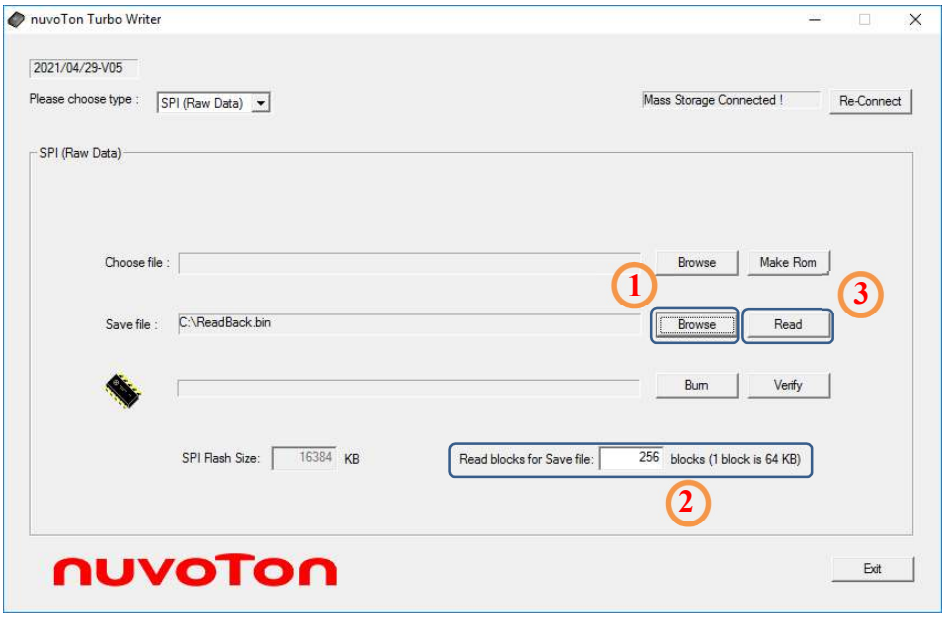

Figure 6 Read back from SPI Flash

3 SPI Flash Layout

The SPI Flash layout is as below.

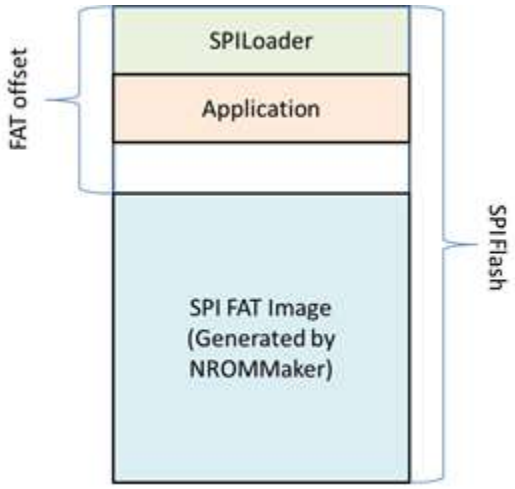

Figure 7 SPI Flash Layout

4 Code Example

4.1 Read files in SPI FAT

The following code is to read file in SPI FAT burned in section 2.2. User can change FAT image offset in SPI Flash. Please make sure that the argument of SpiFlashOpen(…) is the same as the offset.

```
int main()
{
    /* Omit some source code in document. */

    /*---------------------------------------------------------------------*/
    /*  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);

    /* Omit some source code in document. */

    fsAsciiToUnicode("C:\\480x272.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);
}
```

4.2 Mass Storage on SPI Flash

User can run the mass_storage example in *BSP\SampleCode\USBD\Mass_Storage* folder with **SPI** target. This example works as removable disk on PC and report SPI Disk size which is assigned by SPI library to Host. Otherwise, user can operate it as usually removable disk.

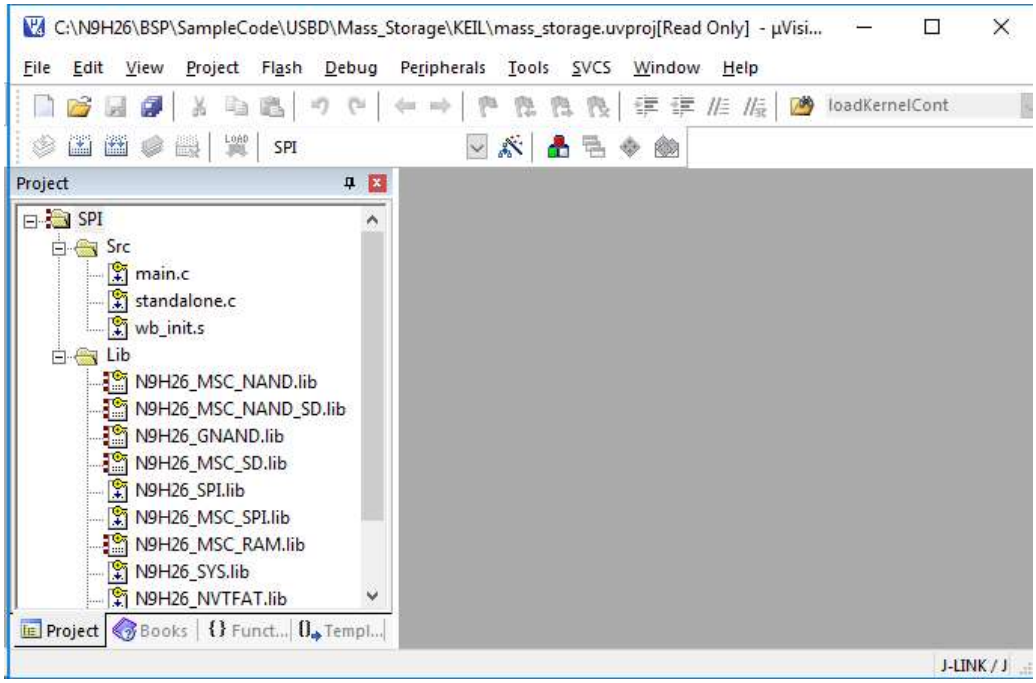

Figure 8 Mass_Storage project

4.3 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 0x1C3018:
            strcpy(pInfo->vendor,"Eon Device");         // EN25Q128
            pInfo->totalSectorN = 16*1024*2;            // Unit : K (16MB)
            pInfo->diskSize = pInfo->totalSectorN/2 - (spi_FAT_offset / 1024);
            break;
        case 0x1C7019:
            strcpy(pInfo->vendor,"Eon Device");         // EN25QH256
```

```
            pInfo->totalSectorN = 32*1024*2;            // Unit : K (32MB)
            pInfo->diskSize = pInfo->totalSectorN/2 - (spi_FAT_offset / 1024);
            break;

        /* Omit some source code in document. */

        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;
    }

    pInfo->totalSectorN = pInfo->totalSectorN - spi_FAT_offset/512;
    sysprintf("Vendor = %s, Total sector = %d for Disk, Total size = %dK, FAT offset
     =%dK\n",pInfo->vendor, pInfo->totalSectorN, 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.

4.4 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. The default exported size is total capacity of SPI Flash subtract FAT offset which is specified in SpiFlashOpen(FATOffset) function.

```
// Added SPI list here to expand support list
INT MapFlashID(UINT32 spiID, DISK_DATA_T* pInfo)
{
    switch(spiID)
    {
        case 0x1C3018:
            strcpy(pInfo->vendor,"Eon Device");         // EN25Q128
```

```
            pInfo->totalSectorN = 16*1024*2;            // Unit : K (16MB)
            pInfo->diskSize = pInfo->totalSectorN/2 - (spi_FAT_offset / 1024);
            break;
        /* Omit some source code in document. */
    }

    pInfo->totalSectorN = pInfo->totalSectorN - spi_FAT_offset/512;
    sysprintf("Vendor = %s, Total sector = %d for Disk, Total size = %dK, FAT offset
       =%dK\n",pInfo->vendor, pInfo->totalSectorN, pInfo->diskSize,spi_FAT_offset/1024);

    return 0;
}
```

If user doesn't want to export all SPI Flash to Mass Storage, user can adjust it by the parameter of SpiFlashOpen function.

```
INT main(VOID)
{
    /* Omit some source code in document. */
    {
        UINT32 block_size, free_size, disk_size, reserved_size;
        fsInitFileSystem();
        fsAssignDriveNumber('C', DISK_TYPE_SD_MMC, 0, 1);

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

        if (fsDiskFreeSpace('C', &block_size, &free_size, &disk_size) < 0)
        {
            UINT32 u32BlockSize, u32FreeSize, u32DiskSize;
            PDISK_T  *pDiskList;
            fsSetReservedArea(reserved_size/512);
            pDiskList = fsGetFullDiskInfomation();
            fsFormatFlashMemoryCard(pDiskList);
            fsReleaseDiskInformation(pDiskList);
            fsDiskFreeSpace('C', &u32BlockSize, &u32FreeSize, &u32DiskSize);
            sysprintf("block_size = %d\n", u32BlockSize);
            sysprintf("free_size = %d\n", u32FreeSize);
            sysprintf("disk_size = %d\n", u32DiskSize);
        }
    }
```

```
    mscdFlashInitExtend(NULL,NULL,NULL, 0,0,0,0);

    /* Omit some source code in document. */
}
```

4.5 Additional note for SPI Flash

The SPI library uses command **0x20** to erase 4 K 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.

5 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 Chapter

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 section 4.3.

6 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
2021.6.10	1.01	1. Modify document structure.
2018.4.5	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.