N9H26
SPI Loader
Reference Guide

V1.0

Publication Release Date: May. 2018

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.

Table of Contents

1. General Description

N9H26 Non-OS library consists of a set of libraries. These libraries are built to access those on-chip functions such as VPOST, APU, SIC, USBH, USBD, GPIO, I2C, SPI and UART, as well as File System (NVTFAT), USB Mass Storage devices (UMAS) and NAND Flash devices (GNAND). This document describes the basic function of SPI Loader. With this introduction, user can quickly understand the SPI Loader on N9H26 microprocessor.

2. SPI Loader Overview

N9H26 built-in 16K bytes IBR (Internal Booting ROM) where stored the boot loader to initial chip basically when power on, and then try to find out the next stage boot loader from different type of storage. It could be SD card, NAND, SPI Flash, or USB storage. The search sequence by IBR is shown in the **錯誤! 找不到參照來源。**.

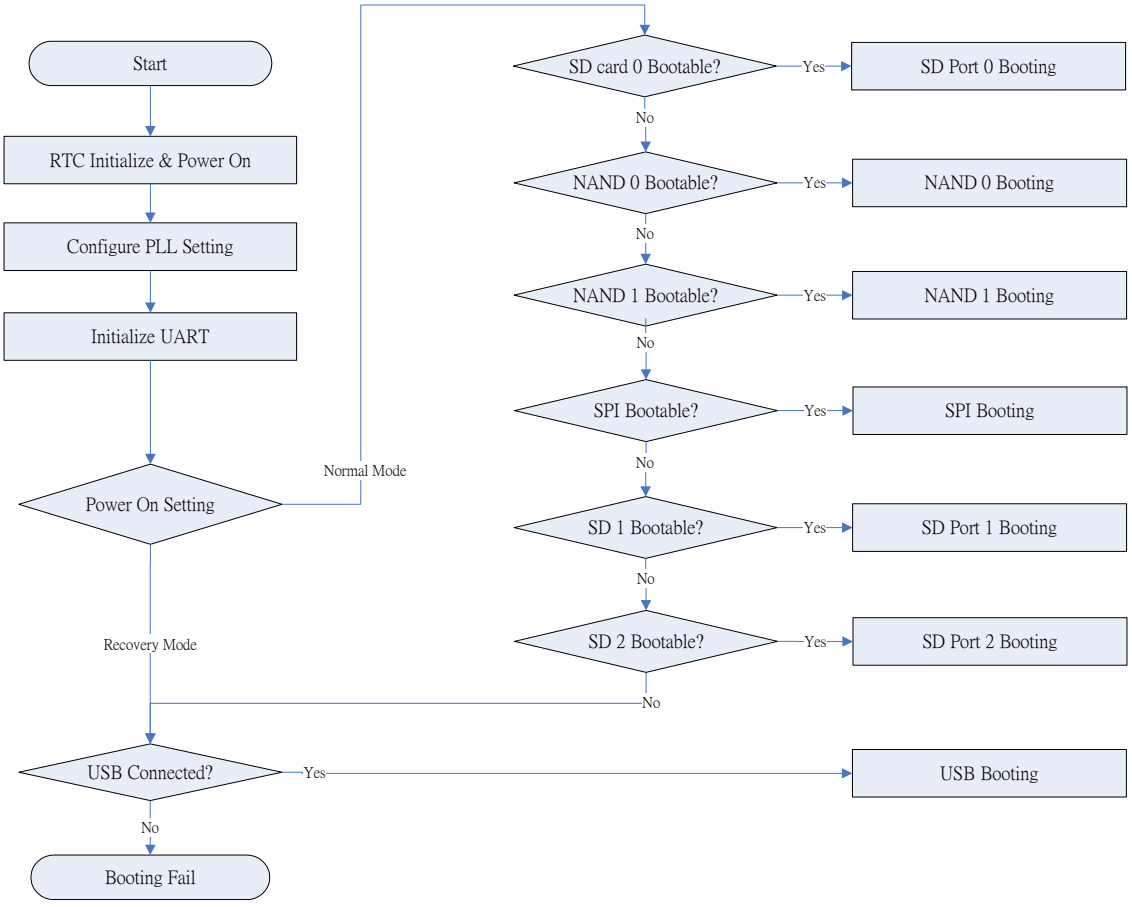

Figure 2-1 IBR Booting Flow

The boot loader in IBR will hand over the chip controlling to SPI Loader if SD card 0 and NAND flash are not for booting.

2.1. SPI Loader Introduction

The SPI Loader has two version – One is SpiLoader & the other is SpiLoader_gzip which has decompression function.

2.2. SpiLoader

SpiLoader flow

- Initial system clock. The default system clock is 240MHz
- Initial more modules such as RTC, SPU, VPOST, and so on if necessary
- Do Security Check if the Security function is enabled (Only for W74M SPI flash)
- Check and load images according to the **Image Information Table** (SPI Flash Offset 63KB)
  - Load Logo image with image type "Logo"
  - Load next firmware with image type "Execute"
- Hand over chip controlling to next firmware.

Images for SPI Solution

For N9H26

	SPI Loader	**Logo Image**	**Execute Image**
Image No.	0	1	2
Image Type	System Image	Logo	Execute
Image execute address	***0x900000***	**0x500000**	**Any valid address**
Image start block	*Default value (0)*	*Behind Spi Loader*	*Behind Logo Image*

Burn images for SPI Solution

Take H9H26 for example

✧ Loader image – SpiLoader_240MHz_GAINTPLUS_QVGA_0313.bin

- Choose the type "SPI"
- Set Image type "System Image"
- Browse the file "SpiLoader_240MHz_GAINTPLUS_QVGA_0313.bin"
- Press the button "Burn"

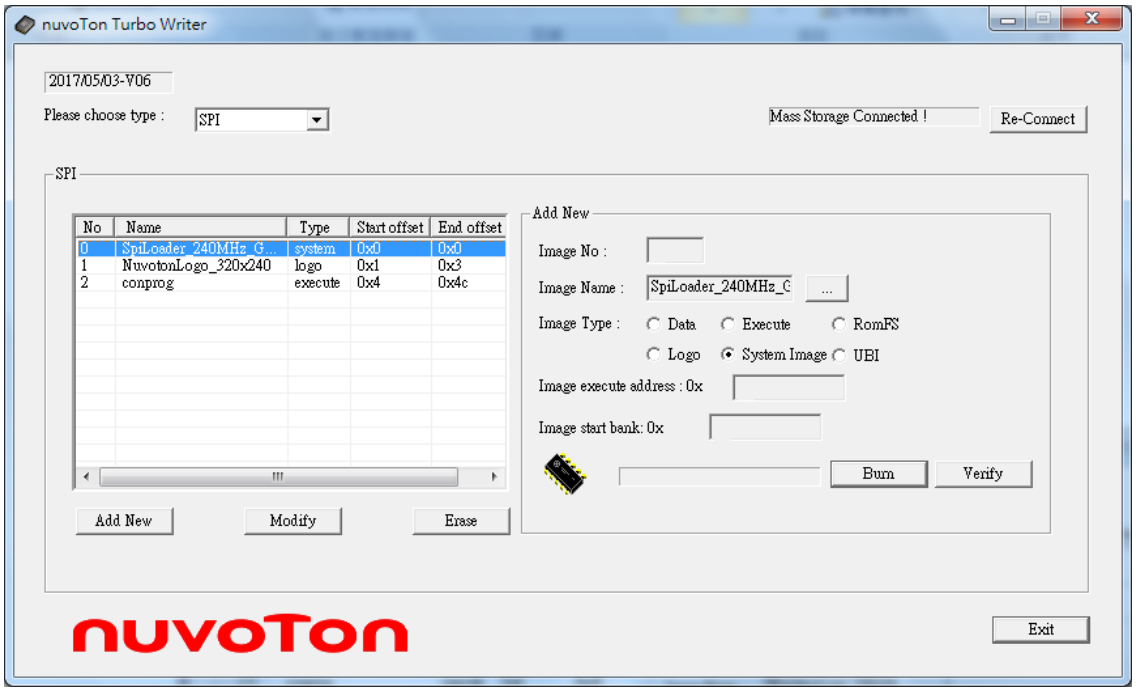

Figure 2-2 SpiLoader_240MHz_GAINTPLUS_QVGA_0313.bin

✧ Logo image – NuvotonLogo_320x240.bin

- Set Image type “Logo”
- Image number “1”
- Browse the file “NuvotonLogo_320x240.bin”
- Set the execute address: **0x500000**
- Set the start block number: **0x1**
- Press the button “Burn”.

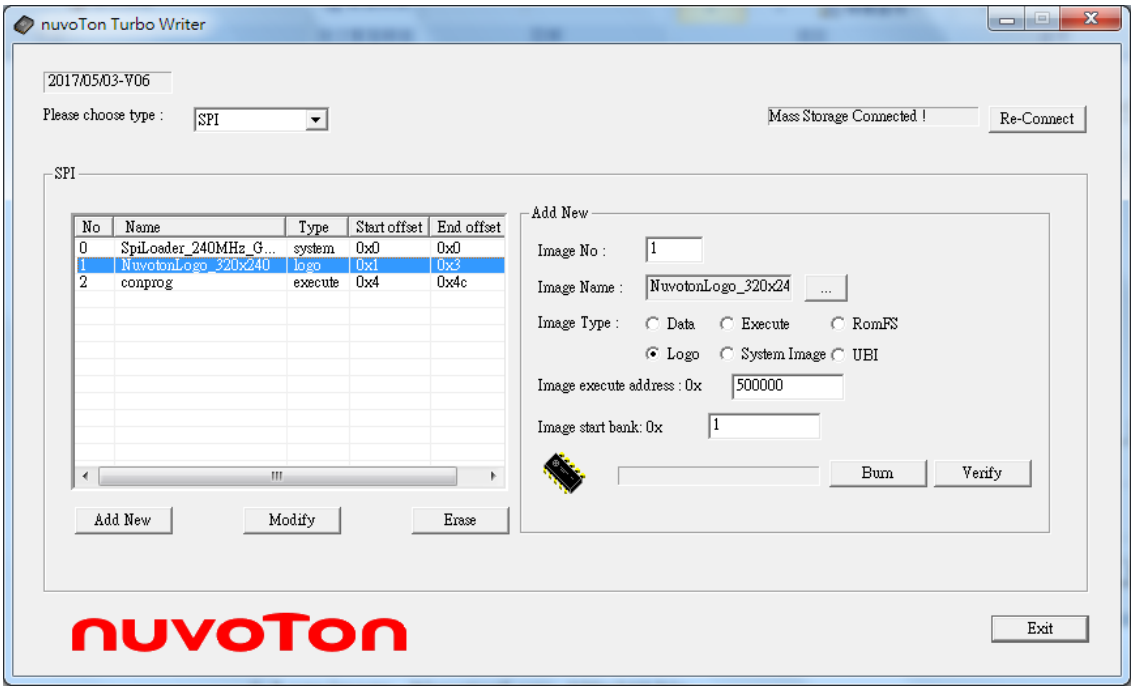

Figure 2-3 NuvotonLogo_320x240.bin

✧ Execture image – Conprog.bin

- Set Image type “Execute”
- Image number “2”
- Browse the file “Conprog.bin”
- Set the execute address: **0x000000**
- Set the start block number: **0x4**
- Press the button “Burn”.

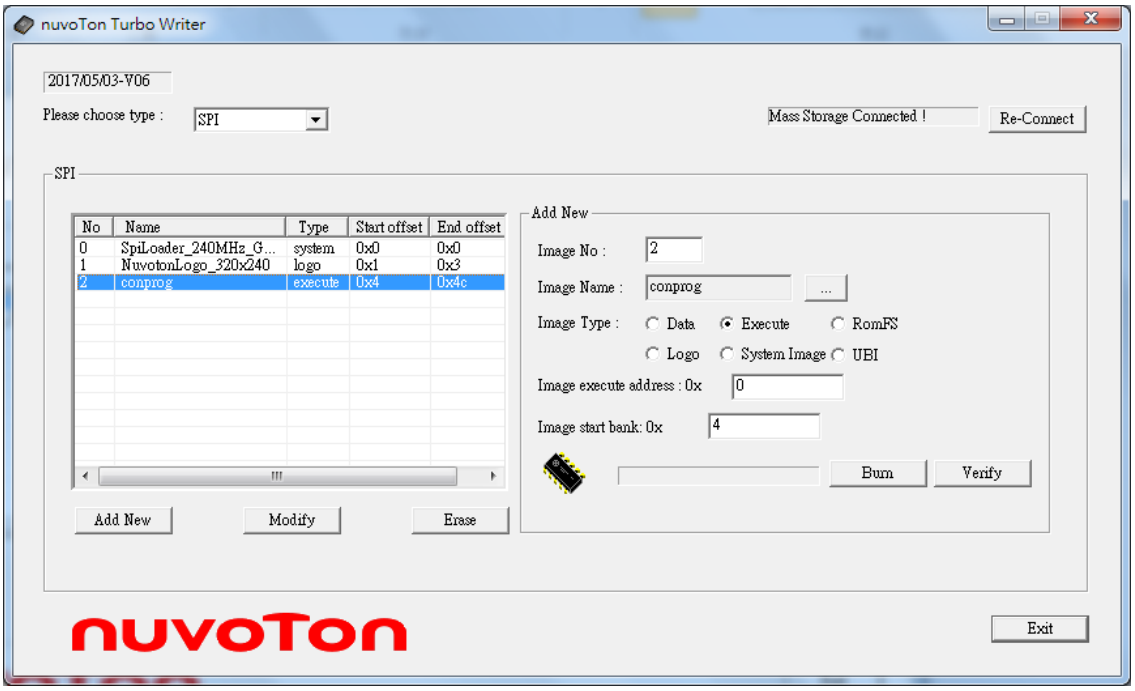

Figure 2-4 Conprog.bin

2.3. SpiLoader with gzip

SpiLoader_gzip flow

- Initial system clock. The default system clock is 240MHz
- Do Security Check if the Security function is enabled (Only for W74M SPI flash)
- Initial more modules such as SPU, RTC, VPOST, and so on if necessary
- Check and load images according to the **Image Information Table** (SPI Flash Offset 63KB)
  - Load Logo image with image type "Logo"
  - Load next firmware with image type "Execute"
- Hand over chip controlling to next firmware.
  - It supports gzip decompression function for execute type image
    - If exxcute image has 64bytes u-Boot header, it will check the Compression type and decompression execute image to the execute address.
    - Execute type image address limitation
      - ✓ Because the compressed image is loaded to the Compressed image address, user needs to make sure that the source data address is not conflict with destination address.

Images for SPI Solution

For N9H26

	SPI Loader_gzip	**Logo Image**	**Execute Image**
Image No.	0	1	2
Image Name	*File name for SPI Loader*	*File name for Logo image*	*File name for Execute Image*
Image Type	System Image	Logo	Execute
Image start block	*Default value (0)*	*Behind Spi Loader*	*Behind Logo Image*
Compressed image address	*Not support*	*Not support*	*0xA00000*

Burn images for SPI Solution

Take N9H26 for example

✧ Loader image – SpiLoader_gzip_240MHz_GAINTPLUS_QVGA_0313.bin

- Choose the type “SPI”
- Set Image type “System Image”
- Browse the file “SpiLoader_gzip_240MHz_GAINTPLUS_QVGA_0313.bin”
- Press the button “Burn”

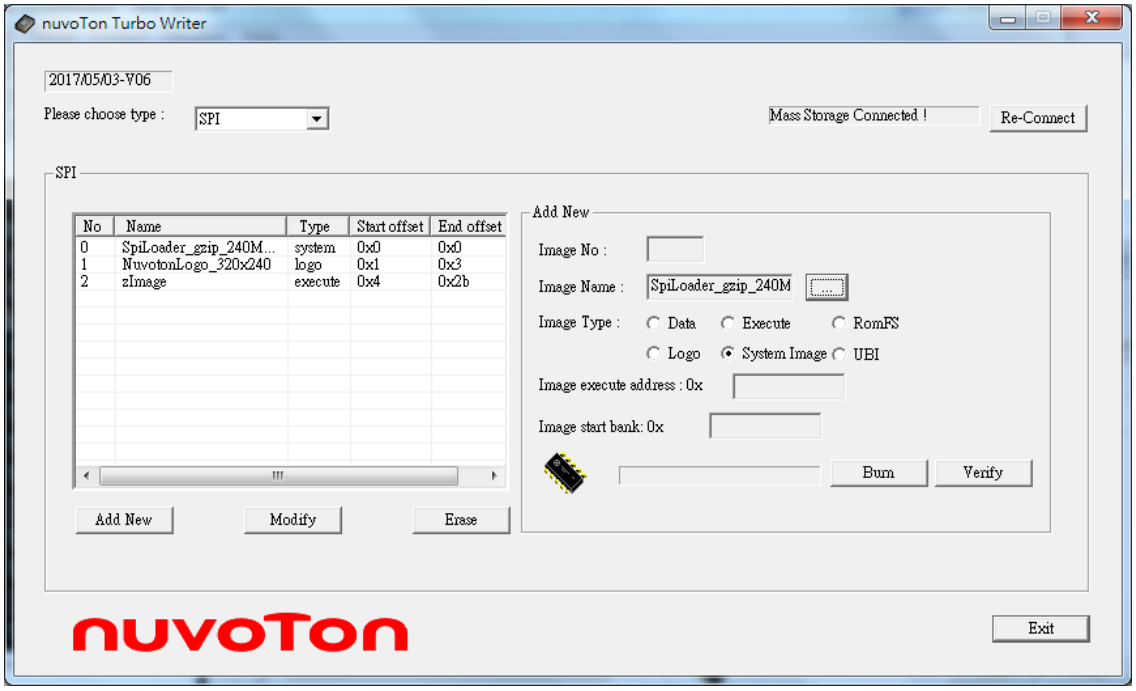

Figure 2-5 SpiLoader_gzip_240MHz_GAINTPLUS_QVGA_0313.bin

✧ Logo image – NuvotonLogo_320x240.bin

- Set Image type “Logo”
- Image number “1”
- Browse the file “NuvotonLogo_320x240.bin”
- Set the execute address: **0x500000**
- Set the start block number: **0x1**
- Press the button “Burn”.

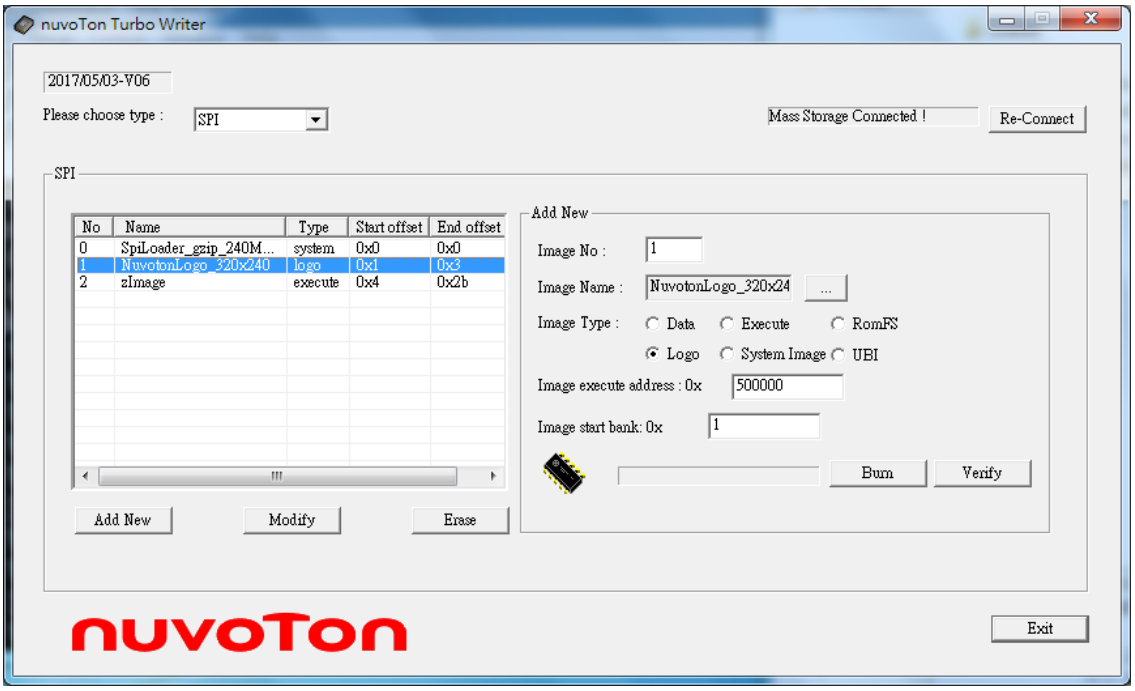

Figure 2-6 NuvotonLogo_320x240.bin

✧ Execture image – zImage.bin

- Set Image type “Execute”
- Image number “2”
- Browse the file “zImage.bin”
- Set the execute address: **0x000000**
- Set the start block number: **0x4**
- Press the button “Burn”.

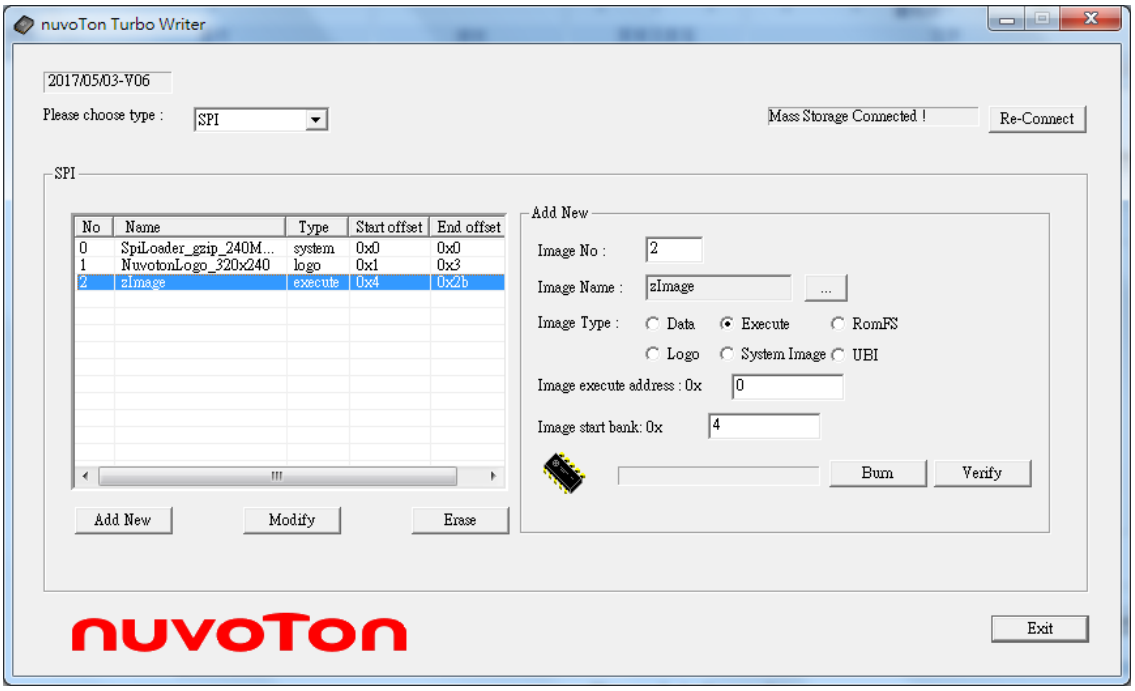

Figure 2-7 zImage.bin

Difference between SpiLoader and SpiLoader_gzip

Because IBR SPI Booting Read operation takes more time than other booting, we hope the code size of SPI loader is as small as possible. We create two project files to build the SpiLoader with/without decompression function. (It takes about 11KB Code to deal with decompression) SpiLoader_gzip is used when code size is critical.

Image format for SpiLoader_gzip

The compressed file must created by gzip and it needs to have u-Boot image header as follows.

```
typedef struct image_header {
        uint32_t        ih_magic;       /* Image Header Magic Number            */
        uint32_t        ih_hcrc;        /* Image Header CRC Checksum            */
        uint32_t        ih_time;        /* Image Creation Timestamp             */
        uint32_t        ih_size;        /* Image Data Size
          */
        uint32_t        ih_load;        /* Data  Load  Address
          */
        uint32_t        ih_ep;          /* Entry Point Address
          */
        uint32_t        ih_dcrc;        /* Image Data CRC Checksum              */
        uint8_t         ih_os;          /* Operating System
          */
        uint8_t         ih_arch;        /* CPU architecture
          */
```

```
    uint8_t         ih_type;        /* Image Type                   */
    uint8_t         ih_comp;        /* Compression Type
      */
    uint8_t         ih_name[IH_NMLEN];      /* Image Name           */
} image_header_t;
```

[Note] SpiLoader only uses the fields ih_magic (0x56190527) and ih_comp (0x01).

Spend time between SpiLoader and SpiLoader_gzip

Although the data SpiLoader_gzip needs to read is less than SpiLoader, it needs to take time to do decompression operation. Here is an example for SpiLoader/SpiLoader_gzip

Table 1 SpiLoader & SpiLoader_gzip size example

	Size
Normal spiLoader	23.7KB (24368Bytes)
spiLoader with gzip	40.3KB (45056Bytes)

Table 2 Spend time of SpiLoader & SpiLoader_gzip example (240MHz)

	Total Time	Un-Compressed time	Load image time	Image Size
Un-compressed image	5.023 s	N/A	4.243 s	4.52 MB (4742848Bytes)
gzip-compressed image	4.306 s	1.17 s	2.309s	2.46 MB (2592768Bytes)

[Note] The total time is from IBR starts to Linux Kernel Start.

3. Source Code Review

3.1. Build SpiLoader Image

SpiLoader project supports both Metrowerks CodeWarrior for ARM Developer Suite (ADS) and Keil uVision IDE. Each project file provides several targets for different panel. Please select "**GAINTPLUS_GPM1006**" as the standard target for N9H26 demo board. The clock setting, SPI mode (1 bit mode /4 bit mode), and RTC setting is defined in SpiLoader.h. Please modify it before you build SpiLoader.

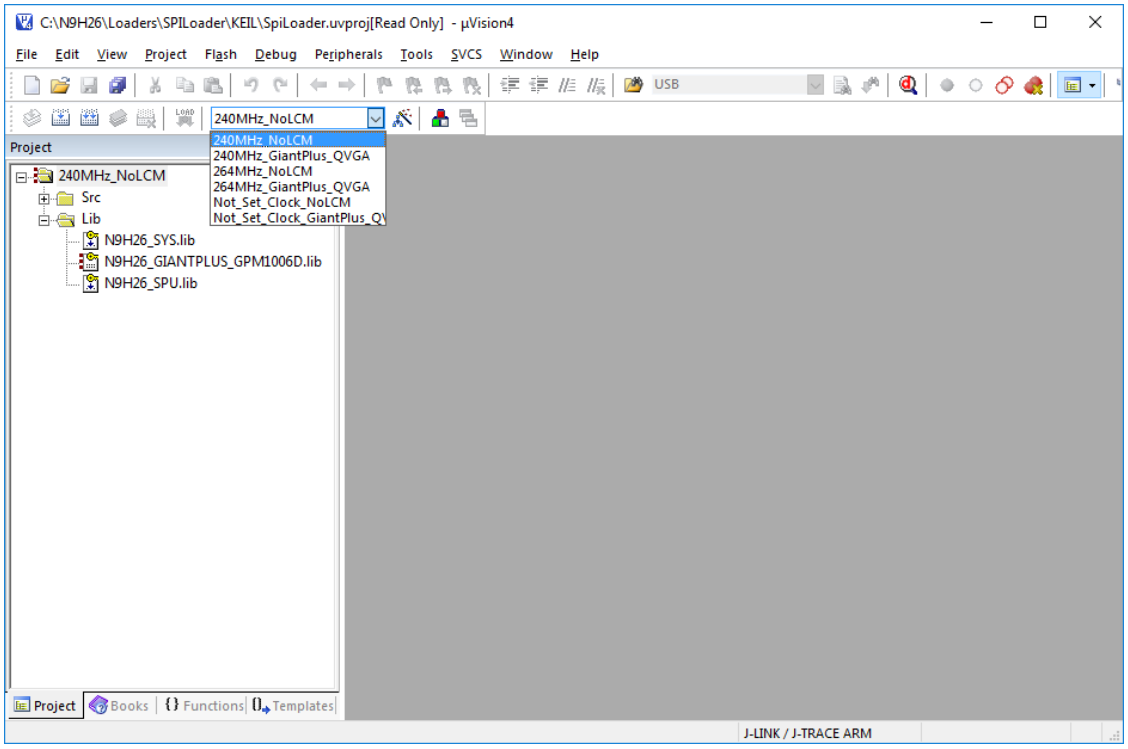

Figure 3-1 SpiLoader project in Keil

3.2. Source Code Review

If you want to modify SpiLoader by yourself, following description about SpiLoader source code could be helpful for you.

System Initial

The first job of SpiLoader is to enable engine clock, set system clock, and configure UART setting. It is implemented by function **initClock**().The standard system clock is 240MHz because the predefined constant **__UPLL_240__** is defined in Spiloader.h file. Select modify Spiloader.h file to build SpiLoader image with different system clock. The proposed system clock for N9H26 is **240MHz**. It can be divided to 48MHz for USBD engine and make N9H26 run stably. If you want to run N9H26 at higher system clock, you have to take risks by yourself. If you don't know how to get correct setting for DRAM, please don't modify it.

```
void initClock(void)
{
    UINT32 u32ExtFreq;
    UINT32 reg_tmp;

    u32ExtFreq = sysGetExternalClock();     /* Hz unit */
    if(u32ExtFreq==12000000)
    {
        outp32(REG_SDREF, 0x805A);
    }
    else
    {
        outp32(REG_SDREF, 0x80C0);
    }
 #ifdef __ABB__
    outp32(REG_SDREF, 0x8016);
 #endif

#ifdef __UPLL_240__
    outp32(REG_CKDQSDS, E_CLKSKEW);
    #ifdef __DDR2__
        outp32(REG_SDTIME, 0x2ABF394A);     /* REG_SDTIME for 360MHz SDRAM clock */
        outp32(REG_SDMR, 0x00000432);
        outp32(REG_MISC_SSEL, 0x00000155);  /* set MISC_SSEL to Reduced Strength to
improve EMI */
    #endif

    /* initial DRAM clock BEFORE inital system clock since we change it from low (216MHz
by IBR) to high (360MHz) */
    sysSetDramClock(eSYS_MPLL, 360000000, 360000000);   /* change from 216MHz (IBR) to
360MHz, */

    /* initial system clock */
    sysSetSystemClock(eSYS_UPLL,
                      240000000,      /* Specified the APLL/UPLL clock, unit Hz */
                      240000000);     /* Specified the system clock, unit Hz */
```

```
    sysSetCPUClock (240000000);     /* Unit Hz */
    sysSetAPBClock ( 60000000);     /* Unit Hz */
#endif  /* __UPLL_240__ */

    /* always set APLL to 432MHz */
    sysSetPllClock(eSYS_APLL, 432000000);

    /* always set HCLK234 to 0 */
    reg_tmp = inp32(REG_CLKDIV4) | CHG_APB;     /* MUST set CHG_APB to HIGH when
configure CLKDIV4 */
    outp32(REG_CLKDIV4, reg_tmp & (~HCLK234_N));

#ifdef __UPLL_NOT_SET__
    sysprintf("Spi Loader DONOT set anything and follow IBR setting !!\n");
    sysprintf("  REG_SDTIME = 0x%08X\n", inp32(REG_SDTIME));
#endif  /* __UPLL_NOT_SET__  */
}
```

UART Initial

Configure UART setting and enable engine clock.

```
void init(void)
{
    WB_UART_T  uart;
    UINT32  u32ExtFreq;
    UINT32 u32Cke = inp32(REG_AHBCLK);
    /* Reset SIC engine to fix USB update kernel and mvoie file */
    outp32(REG_AHBCLK, u32Cke  | (SIC_CKE | NAND_CKE | SD_CKE));
    outp32(REG_AHBIPRST, inp32(REG_AHBIPRST )|SIC_RST );
    outp32(REG_AHBIPRST, 0);
    outp32(REG_AHBCLK,u32Cke);

    sysDisableCache();
    sysFlushCache(I_D_CACHE);
    sysEnableCache(CACHE_WRITE_BACK);
    u32ExtFreq = sysGetExternalClock();         /* KHz unit */

    /* enable UART */
    sysUartPort(1);
    uart.uiFreq = u32ExtFreq;                   /* Hz unit */
    uart.uiBaudrate = 115200;
    uart.uart_no = WB_UART_1;
    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("SPI Loader start (%s).\n", DATE_CODE);
}
```

RTC Initial

RTC hardware power off function is enabled by default in SpiLoader. If RTC hardware power off function is not required for your solution, please unvomment the definition __RTC_HW_PWOFF__ in SpiLoader.h.
If RTC is not required for your solution, please unvomment the definition __No_RTC__ in SpiLoader.h.

```
#ifdef __No_RTC__
    sysprintf("* Not Config RTC\n");
    outp32(REG_APBCLK, inp32(REG_APBCLK) & ~RTC_CKE);   /* disable RTC clock to save
power */
#else
    #ifdef __RTC_HW_PWOFF__
    sysprintf("Enable HW Power Off\n");
    /* RTC H/W Power Off Function Configuration */
    RTC_Check();    /* waiting for RTC regiesters ready for access */
    outp32(PWRON, (inp32(PWRON) & ~PCLR_TIME) | 0x60005);   /* Press Power Key during 6
sec to Power off (0x'6'0005) */
    RTC_Check();
    outp32(RIIR, 0x4);
    RTC_Check();
```

```
    outp32(REG_APBCLK, inp32(REG_APBCLK) & ~RTC_CKE);   /* disable RTC clock to save
power */
    #else
    /* RTC H/W Power Off Function Configuration */
    RTC_Check();    /* waiting for RTC regiesters ready for access */
    outp32(PWRON, (inp32(PWRON) & ~PCLR_TIME) & ~0x4);   /* Press Power Key during 6 sec
to Power off (0x'6'0005) */
    RTC_Check();
    outp32(RIIR, 0x4);
    RTC_Check();
    sysprintf("Disable HW Power Off - 0x%X\n",inp32(PWRON));
    outp32(REG_APBCLK, inp32(REG_APBCLK) & ~RTC_CKE);   /* disable RTC clock to save
power */
    #endif
#endif
```

SPU Initial

The Spi Loader also needs to initialize SPU to avoid pop noise.

```
int main(void)
{
   …
    DrvSPU_Open();
    spuDacPLLEnable();
    spuDacPrechargeEnable();
   …
}
```

Security function

The security function is to provide anti-copy functionto prevent the reproduction. The security funciotn is only supported by W74M series – provide an authentication mechanism to ensure the physical authenticity of the attached flash devices. If you want to use te security function, please build SpiLoader by the project file name with security.

```
int main(void)
{
   …

#ifdef __Security__
    UINT8     u8UID[8];
    unsigned char ROOTKey[32];     /* Rootkey array */
    unsigned char HMACKey[32];     /* HMACkey array */
    unsigned char HMACMessage[4];  /* HMAC message data, use for update HMAC key */
    unsigned char Input_tag[12];   /* Input tag data for request conte */
    unsigned char RPMCStatus;
#endif
     …
#ifdef __Security__
    if ((RPMC_ReadUID(u8UID)) == -1)
    {
        sysprintf("read id error !!\n");
        return -1;
```

```
    }

    sysprintf("SPI flash uid [0x%02X%02X%02X%02X%02X%02X%02X%02X]\n",u8UID[0],
u8UID[1],u8UID[2], u8UID[3],u8UID[4], u8UID[5],u8UID[6], u8UID[7]);

    /* first stage, initial rootkey */
    RPMC_CreateRootKey((unsigned char *)u8UID,8, ROOTKey);    /* caculate ROOTKey with
UID & ROOTKeyTag by SHA256 */

    /* Second stage, update HMACKey after ever power on. without update HMACkey, Gneiss
would not function */
    HMACMessage[0] = rand()%0x100;        /* Get random data for HMAC message, it can
also be serial number, RTC information and so on. */
    HMACMessage[1] = rand()%0x100;
    HMACMessage[2] = rand()%0x100;
    HMACMessage[3] = rand()%0x100;

    /* Update HMAC key and get new HMACKey.
       HMACKey is generated by SW using Rootkey and HMACMessage.
       RPMC would also generate the same HMACKey by HW
     */
    RPMCStatus = RPMC_UpHMACkey(KEY_INDEX, ROOTKey, HMACMessage, HMACKey);
    if(RPMCStatus == 0x80)
    {
        /* update HMACkey success */
        sysprintf("RPMC_UpHMACkey Success - 0x%02X!!\n",RPMCStatus );
    }
    else
    {
        /* write HMACkey fail, check datasheet for the error bit */
        sysprintf("RPMC_UpHMACkey Fail - 0x%02X!!\n",RPMCStatus );
    }

    /* Third stage, increase RPMC counter */
    /* input tag is send in to RPMC, it could be time stamp, serial number and so on */
    for(i= 0; i<12;i++)
        Input_tag[i] = u8UID[i%8];

    RPMCStatus = RPMC_IncCounter(KEY_INDEX, HMACKey, Input_tag);
    if(RPMCStatus == 0x80)
    {
        /* increase counter success */
        sysprintf("RPMC_IncCounter Success - 0x%02X!!\n",RPMCStatus );
    }
    else
    {
        /* increase counter fail, check datasheet for the error bit */
        sysprintf("RPMC_IncCounter Fail - 0x%02X!!\n",RPMCStatus );
        while(1);
    }

    if(RPMC_Challenge(KEY_INDEX, HMACKey, Input_tag)!=0)
    {
        sysprintf("RPMC_Challenge Fail!!\n" );
        /* return signature miss-match */
        while(1);
    }
    else
        sysprintf("RPMC_Challenge Pass!!\n" );
```

```
#endif
```

Get Image Information Table

The location of Image Information Table always is **offset 63KB** in the SPI Flash. SpiLoader reads and parses it to found out all images that TurboWriter write into.

```
   sysprintf("Load Image ");
   /* read image information */
#ifndef __OTP_4BIT__
   SPIReadFast(0, 63*1024, 1024, (UINT32*)imagebuf);  /* offset, len, address */
#else
   outpw(REG_GPEFUN1, (inpw(REG_GPEFUN1)  & ~(MF_GPE8 |MF_GPE9)) | 0x44);
#endif
   JEDEC_Probe();
   spiFlashFastReadQuad(63*1024, 1024, (UINT32*)imagebuf);  /* offset, len, address */
#endif
```

Load Image from SPI Flash to RAM

After got Image Information Table, SpiLoader will found out the Logo image first and then copy it from SPI Flash to RAM. Next, SpiLoader will found out the RomFS image, copy it from SPI Flash to RAM and create TAG for Linux kernel. Finally, SpiLoader will found out first image with image type “Execute”, copy it from SPI Flash to RAM, and then hand over chip controlling to it.

```
   if (((*(pImageList+0)) == 0xAA554257) && ((*(pImageList+3)) == 0x63594257))
   {
       count = *(pImageList+1);

       pImageList=((unsigned int*)(((unsigned int)image_buffer)|0x80000000));
       startBlock = fileLen = executeAddr = 0;

       /* load logo first */
       pImageList = pImageList+4;
       for (i=0; i<count; i++)
       {
           if (((*(pImageList) >> 16) & 0xffff) == 4)    /* logo */
           {
               startBlock = *(pImageList + 1) & 0xffff;
               executeAddr = *(pImageList + 2);
               fileLen = *(pImageList + 3);
#ifndef __OTP_4BIT__
               SPIReadFast(0, startBlock * 0x10000, fileLen, (UINT32*)executeAddr);
#else
               spiFlashFastReadQuad(startBlock * 0x10000, fileLen,
(UINT32*)executeAddr);
#endif
               break;
           }
           /* pointer to next image */
           pImageList = pImageList+12;
       }
```

```
        pImageList=((unsigned int*)(((unsigned int)image_buffer)|0x80000000));
        startBlock = fileLen = executeAddr = 0;

        /* load romfs file */
        pImageList = pImageList+4;
        for (i=0; i<count; i++)
        {
            if (((*(pImageList) >> 16) & 0xffff) == 2)    /* RomFS */
            {
                startBlock = *(pImageList + 1) & 0xffff;
                executeAddr = *(pImageList + 2);
                fileLen = *(pImageList + 3);
#ifndef __OTP_4BIT__
                SPIReadFast(0, startBlock * 0x10000, fileLen, (UINT32*)executeAddr);
#else
                spiFlashFastReadQuad(startBlock * 0x10000, fileLen,
(UINT32*)executeAddr);
#endif
                tag_flag = 1;
                tagaddr = executeAddr;
                tagsize = fileLen;

                break;
            }
            /* pointer to next image */
            pImageList = pImageList+12;
        }

        pImageList=((unsigned int*)(((unsigned int)image_buffer)|0x80000000));
        startBlock = fileLen = executeAddr = 0;

        /* load execution file */
        pImageList = pImageList+4;
        for (i=0; i<count; i++)
        {
            if (((*(pImageList) >> 16) & 0xffff) == 1)    /* execute */
            {
                startBlock = *(pImageList + 1) & 0xffff;
                executeAddr = *(pImageList + 2);
                fileLen = *(pImageList + 3);
#ifndef __OTP_4BIT__
                SPIReadFast(0, startBlock * 0x10000, fileLen, (UINT32*)executeAddr);
#else
                spiFlashFastReadQuad(startBlock * 0x10000, fileLen,
(UINT32*)executeAddr);
#endif

                sysSetGlobalInterrupt(DISABLE_ALL_INTERRUPTS);
                sysSetLocalInterrupt(DISABLE_FIQ_IRQ);
//                  Invalid and disable cache
//                 sysDisableCache();
//                 sysInvalidCache();
//                 memcpy(0x0, kbuf, CP_SIZE);

                if(tag_flag)
                {
                    sysprintf("Create Tag - Address 0x%08X, Size
0x%08X\n",tagaddr,tagsize );
```

```
                TAG_create(tagaddr,tagsize);
            }

            /* JUMP to kernel */
            sysprintf("Jump to kernel\n\n");

            //lcmFill2Dark((char *)(FB_ADDR | 0x80000000));
            outp32(REG_AHBIPRST, JPG_RST | SIC_RST |UDC_RST | EDMA_RST);
            outp32(REG_AHBIPRST, 0);
            outp32(REG_APBIPRST, UART1RST | UART0RST | TMR1RST | TMR0RST );
            outp32(REG_APBIPRST, 0);
            sysFlushCache(I_D_CACHE);

            fw_func = (void(*)(void))(executeAddr);
            fw_func();
            break;
        }
        /* pointer to next image */
        pImageList = pImageList+12;
    }
}
```

3.2.1. Difference between SpiLoader and SpiLoader_gzip

SpiLodaer_gzip will load first 64byte of execute image to check if there is an u-Boot header.

```
    /* load execution file */
    pImageList = pImageList+4;
    for (i=0; i<count; i++)
    {
        if (((*(pImageList) >> 16) & 0xffff) == 1)    /* execute */
        {
            UINT32 u32Result;
            startBlock = *(pImageList + 1) & 0xffff;
            executeAddr = *(pImageList + 2);
            fileLen = *(pImageList + 3);

#ifndef __OTP_4BIT__
            SPIReadFast(0, startBlock * 0x10000, 64, (UINT32*)IMAGE_BUFFER);
#else
            spiFlashFastReadQuad(startBlock * 0x10000, 64, (UINT32*)IMAGE_BUFFER);
#endif

            u32Result = do_bootm(IMAGE_BUFFER, 0, CHECK_HEADER_ONLY);

            if(u32Result)       /* Not compressed */
            {

#ifndef __OTP_4BIT__
                SPIReadFast(0, startBlock * 0x10000, fileLen, (UINT32*)executeAddr);
#else
                spiFlashFastReadQuad(startBlock * 0x10000, fileLen,
(UINT32*)executeAddr);
#endif
            }
            else              /* compressed */
```

```
                {
#ifndef __OTP_4BIT__
                    SPIReadFast(0, startBlock * 0x10000, fileLen, (UINT32*)IMAGE_BUFFER);
#else
                    spiFlashFastReadQuad(startBlock * 0x10000, fileLen,
(UINT32*)IMAGE_BUFFER);
#endif
                    do_bootm(IMAGE_BUFFER, executeAddr, LOAD_IMAGE);
                }
                sysSetGlobalInterrupt(DISABLE_ALL_INTERRUPTS);
                sysSetLocalInterrupt(DISABLE_FIQ_IRQ);
//                  Invalid and disable cache
//                 sysDisableCache();
//                 sysInvalidCache();
//                 memcpy(0x0, kbuf, CP_SIZE);

                if(tag_flag)
                {
                    sysprintf("Create Tag - Address 0x%08X, Size
0x%08X\n",tagaddr,tagsize );
                    TAG_create(tagaddr,tagsize);
                }

                /* JUMP to kernel */
                sysprintf("Jump to kernel\n\n");

                //lcmFill2Dark((char *)(FB_ADDR | 0x80000000));
                outp32(REG_AHBIPRST, JPG_RST | SIC_RST |UDC_RST | EDMA_RST);
                outp32(REG_AHBIPRST, 0);
                outp32(REG_APBIPRST, UART1RST | UART0RST | TMR1RST | TMR0RST );
                outp32(REG_APBIPRST, 0);
                sysFlushCache(I_D_CACHE);

                fw_func = (void(*)(void))(executeAddr);
                fw_func();
                break;
            }
            /* pointer to next image */
            pImageList = pImageList+12;
        }
    }
```

4. Revision History

Version	Date	Description
V1.0	May. 4, 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.