N9H20
SPI Loader
Reference Guide

V1.0

Publication Release Date: May. 2018

Support Chips:
N9H20 Series

Support Platforms:
Non-OS

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

N9H20 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 N9H20 microprocessor.

2. SPI Loader Overview

N9H20 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 Figure 2-1.

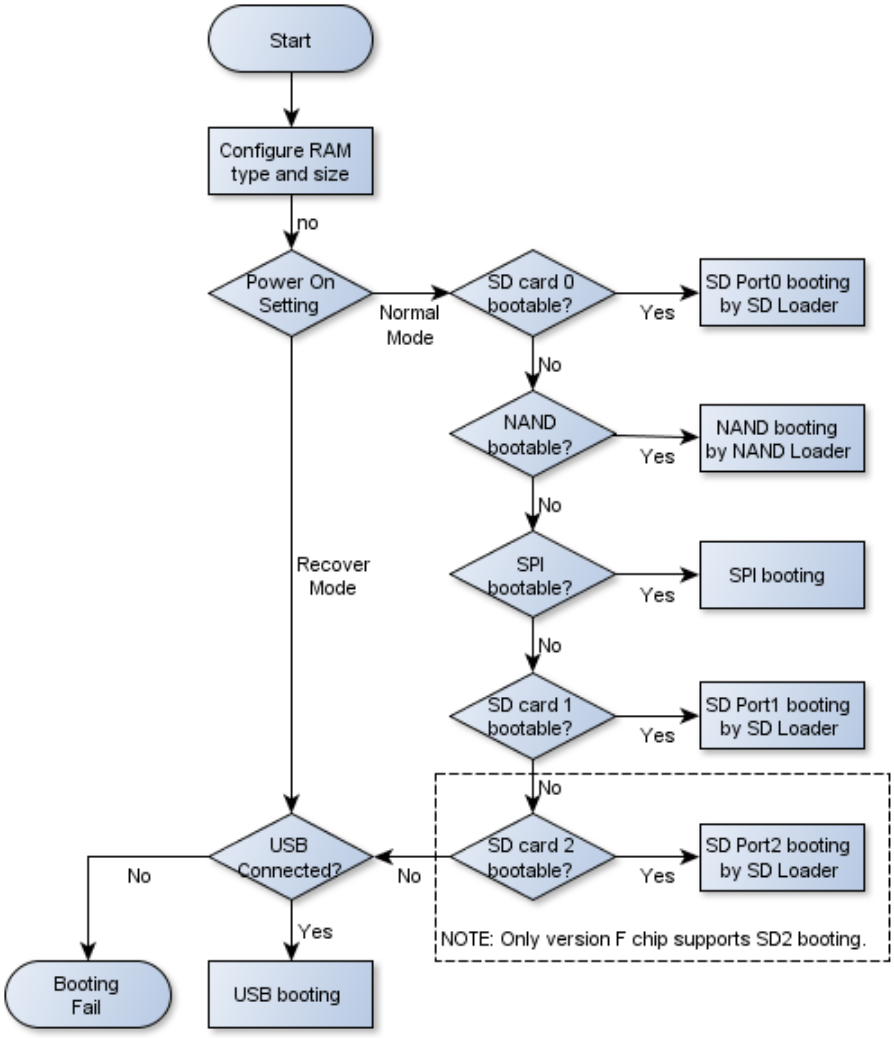

Figure 2-1 IBR Booting Flow

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 192MHz
- 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 N9H20K5

	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*

For N9H20K3

	SPI Loader	**Logo Image**	**Execute Image**
Image No.	0	1	2
Image Type	System Image	Logo	Execute
Image execute address	***0x700000***	**0x500000**	**Depend on firmware**
Image start block	*Default value (0)*	*Behind Spi Loader*	*Behind Logo Image*

For N9H20K1

	SPI Loader	**Logo Image**	**Execute Image**
Image No.	0	*Not support*	1
Image Type	System Image	*Not support*	Execute
Image execute address	***0x180000***	*Not support*	**Depend on firmware**
Image start block	*Default value (0)*	*Not support*	*Behind Spi Loader*

Burn images for SPI Solution

Take N9H20K5 for example

- ✧ Loader image – SpiLoader_192MHz_GPM1006_QVGA_1002.bin
  - ■ Choose the type “SPI”
  - ■ Set Image type “System Image”
  - ■ Browse the file “SpiLoader_192MHz_GPM1006_QVGA_1002.bin”
  - ■ Press the button “Burn”

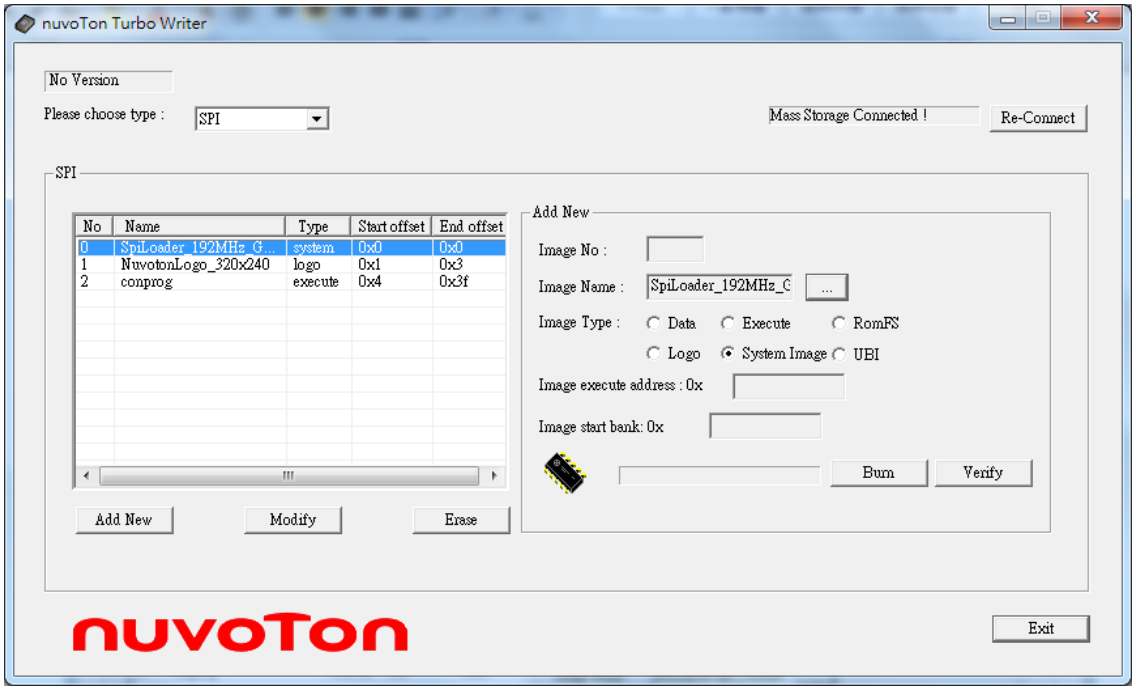

Figure 2-2 SpiLoader_192MHz_GPM1006_QVGA_1002.bin

- ✧ 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”.

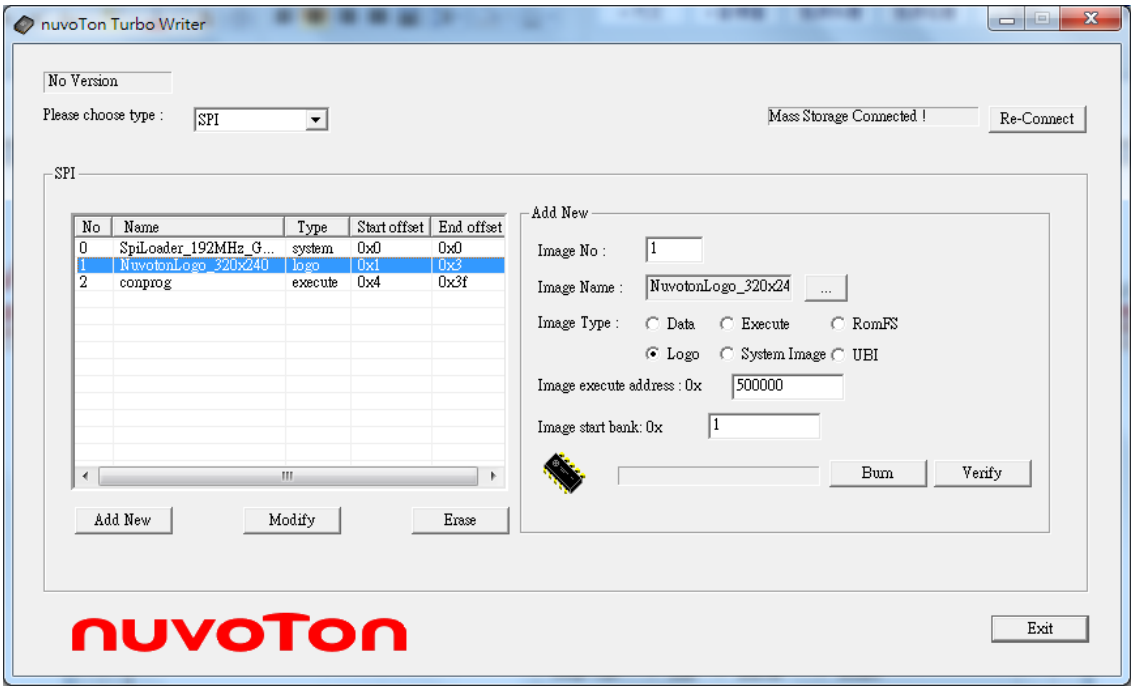

Figure 2-3 NuvotonLogo_320x240.bin

✧ 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”.

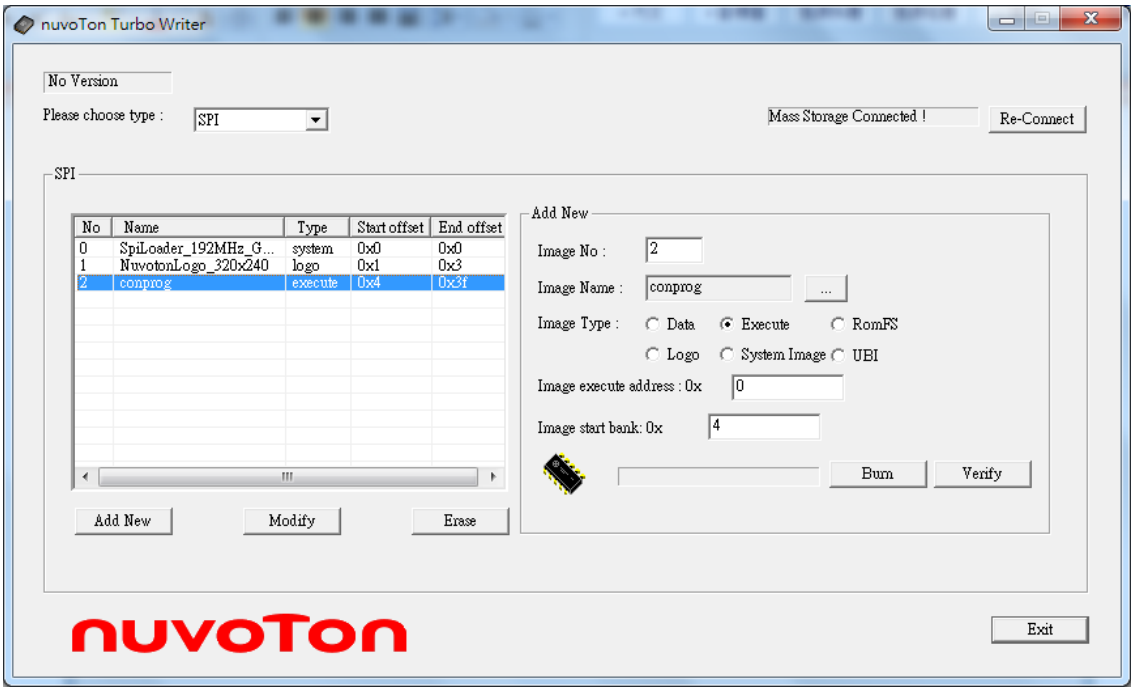

Figure 2-4 Conprog.bin

Boot Up flow for SPI Solution

Init RTC
DDR2 32MB
SD Port0 Booting Fail – No Device [No Device]
NAND Booting 2K-Page (4) Fail - Not for Boot [Not for Boot]
SPI Booting Success [SPI Booting Success]
Clock Skew [IBR operation]
- B0003030 = 00001010
- B0003034 = 00888800
Code Executes at 0x00900000 [Jump to SpiLoader]
* Not Config RTC
SPI Loader start (20151002).
Load Image Load file length 0x400, execute address 0x80906F08 [Load Image List]
Load file length 0x25800, execute address 0x500000 [Load Logo Image]
Load file length 0x3BFB00, execute address 0x0 [Load Execute Image]
Jump to kernel Linux version 2.6.35.4 (root@ccchang.nuvoton) (gcc version 4.2.1) #490 PREEMPT Wed Jan 21 15:06:08 CST 2015

[Note] The IBR operations are different between different IBR version

2.3. SpiLoader with gzip

SpiLoader_gzip flow

- Initial system clock. The default system clock is 192MHz
- 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 execute 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 N9H20K5

	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*

For N9H20K3

	SPI Loader_gzip	Logo Image	Execute Image
Image No.	0	1	2
Image Type	System Image	Logo	Execute
Image execute address	***0x700000***	**0x500000**	**Depend on firmware**
Image start block	*Default value (0)*	*Behind Spi Loader*	*Behind Logo Image*
Compressed image address	*Not support*	*Not support*	*0x300000*

For N9H20K1

	SPI Loader	Logo Image	Execute Image
Image No.	0	*Not support*	1
Image Type	System Image	*Not support*	Execute
Image execute address	***0x180000***	*Not support*	**Depend on firmware**
Image start block	*Default value (0)*	*Not support*	*Behind Spi Loader*
Compressed image address	*Not support*	*Not support*	*0x130000*

Burn images for SPI Solution

Take N9H20K5 for example

✧ Loader image – SpiLoader_gzip_192MHz_GPM1006_QVGA_1002.bin

- Choose the type “SPI”
- Set Image type “System Image”
- Browse the file “SpiLoader_gzip _192MHz_GPM1006_QVGA_1002.bin”
- Press the button “Burn”

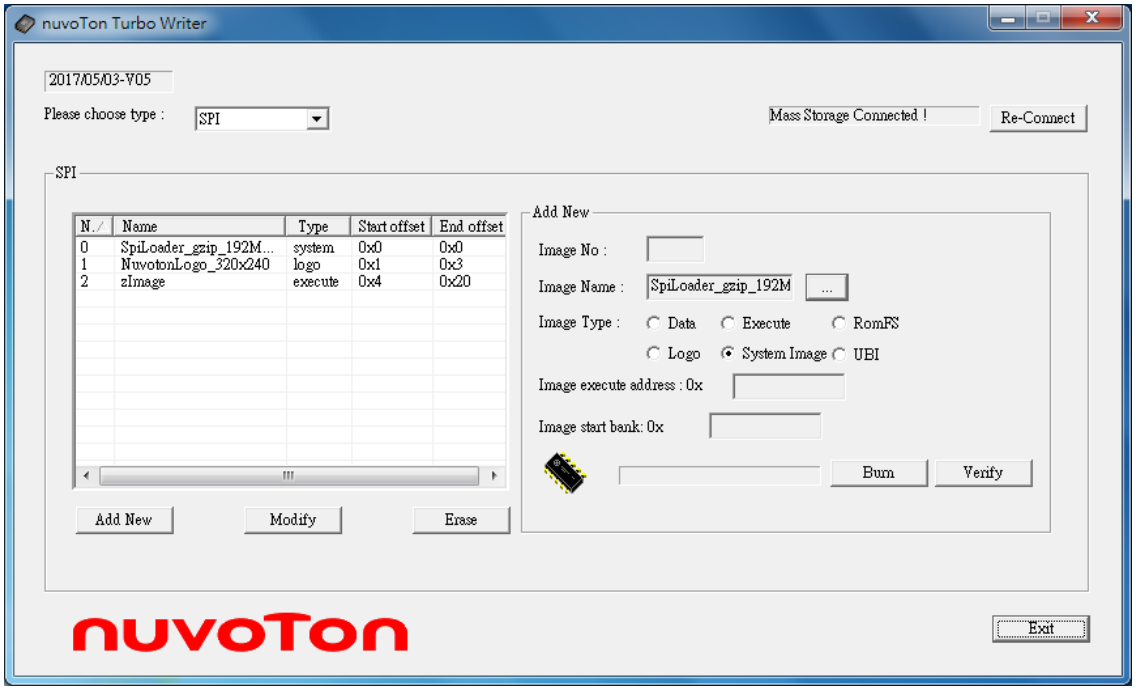

Figure 2-5 SpiLoader_gzip_192MHz_GPM1006_QVGA_1002.bin

✧ 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”.

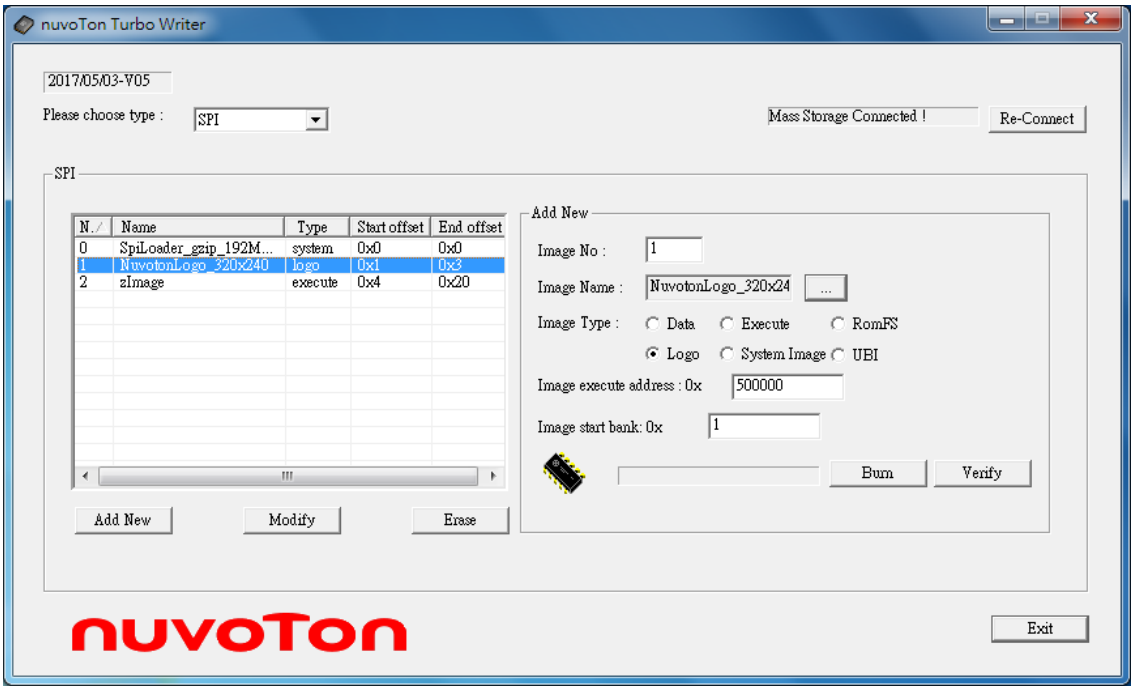

Figure 2-6 NuvotonLogo_320x240.bin

✧ Execute 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”.

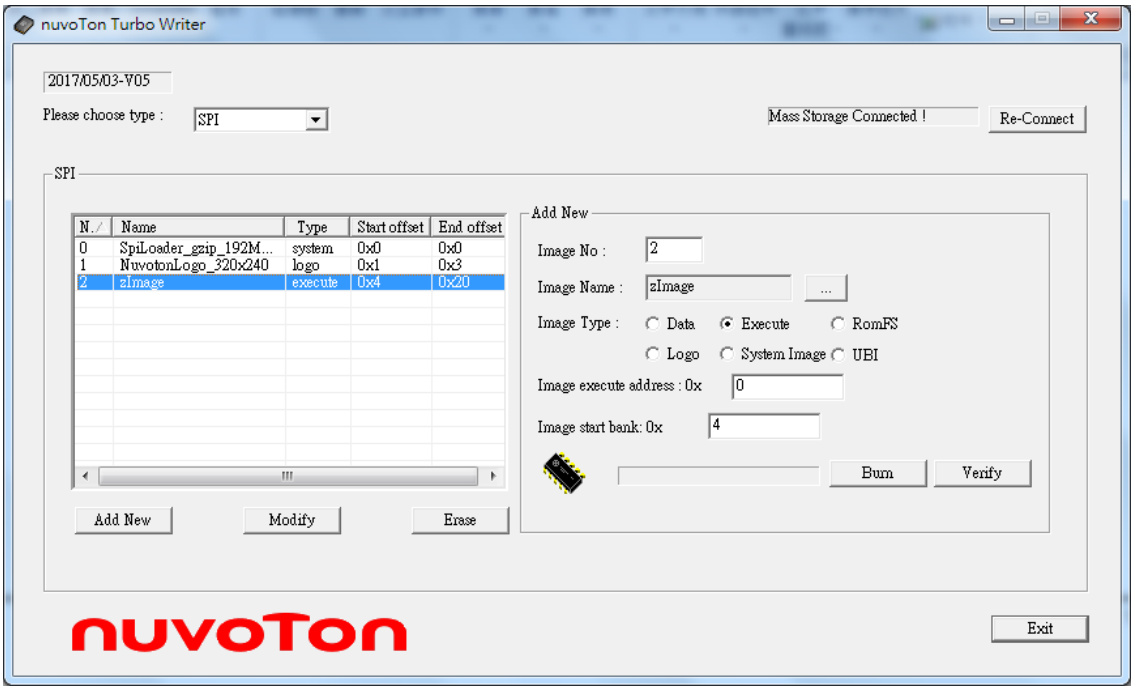

Figure 2-7 zImage.bin

Boot Up flow for SPI Solution with compressed image

Init RTC
DDR2 32MB
SD Port0 Booting Fail – No Device [No Device]
NAND Booting 2K-Page (4) Fail - Not for Boot [Not for Boot]
SPI Booting Success [SPI Booting Success]
Clock Skew [IBR operation]
- B0003030 = 00001010
- B0003034 = 00888800
Code Executes at 0x00900000 [Jump to SpiLoader]
* Not Config RTC
SPI Loader start (20151002 - gzip).
Booting image at 0x00A00000 ... [Check u-Boot Header]
Get Magic Number
Booting image at 0x00A00000 ...
Get Magic Number
Gzip Uncompressing to 0x0 ... OK [Decompressed image]
Jump to kernel Linux version 2.6.35.4 (root@CentOS.Server) (gcc version 4.2.1) #182 PREEMPT Tue May 20 09:52:46 CST 2014

[Note] The IBR operations are different between different IBR version

Boot Up flow for SPI Solution without compressed image

```
Init RTC
DDR2 32MB
SD Port0 Booting Fail – No Device                    [No Device]
NAND Booting 2K-Page (4) Fail - Not for Boot         [Not for Boot]
SPI Booting Success                                  [SPI Booting Success]
Clock Skew                                           [IBR operation]
  - B0003030 = 00001010
  - B0003034 = 00888800
Code Executes at 0x00900000                          [Jump to SpiLoader]
* Not Config RTC
SPI Loader start (20151002 - gzip).
## Booting image at 0x00A00000 ...                   [Check u-Boot Header]
Bad Magic Number
Jump to kernel    Linux version 2.6.35.4 (root@CentOS.Server) (gcc version 4.2.1) #182 PREEMPT
Tue May 20 09:52:46 CST 2014
```

[Note] The IBR operations are different between different IBR version

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. 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	17.8KB (18240Bytes)
spiLoader with gzip	29.0KB (29784Bytes)

Table 2 Spend time of SpiLoader & SpiLoader_gzip example (192MHz)

	Total Time	Un-Compressed time	Load image time	Image Size
Un-compressed image	2.156 s	N/A	1.828 s	2.62 MB (2748280Bytes)
gzip-compressed image	2.203 s	0.703 s	1.141 s	1.64 MB (1725536Bytes)

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

3. Source Code Review

3.1. Build SpiLoader Image

SpiLoader project supports Keil uVision IDE. Each project file provides several targets for different panel. Please select "**GAINTPLUS_ GPM1006**" as the standard target for N9H20 demo board.

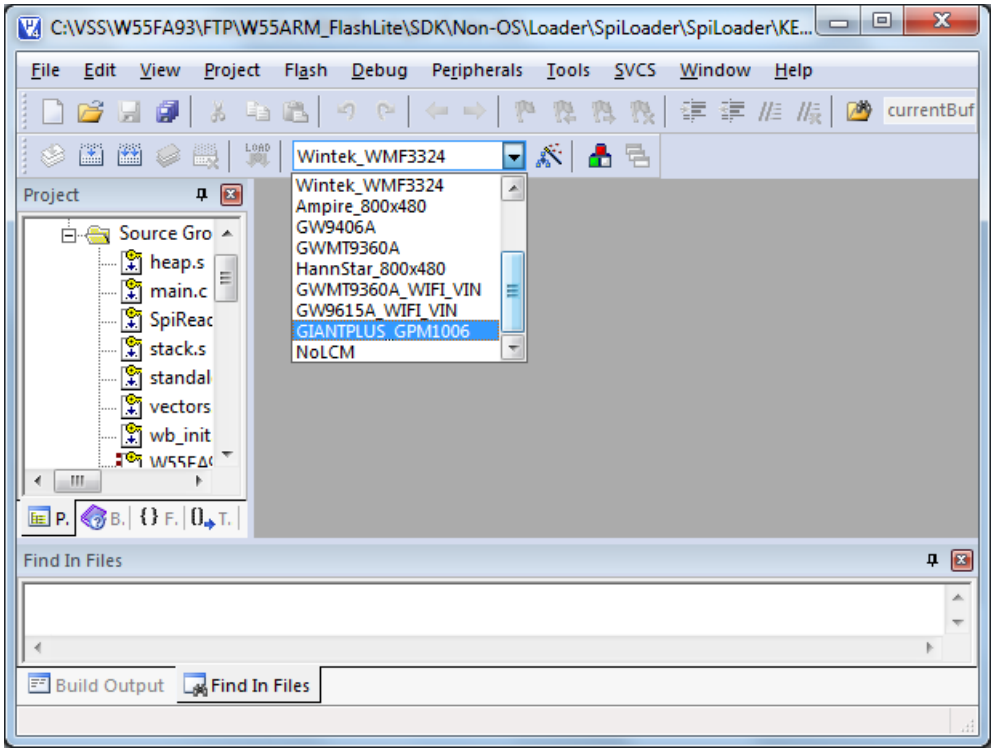

Figure 3-1 SpiLoader project in Keil

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 **init**(). The proposed system clock for N9H20 is **192MHz**. It can be divided to 48MHz for USBD engine and make N9H20 run stably. If you want to run N9H20 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 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 )|SICRST );
    outp32(REG_AHBIPRST, 0);
    outp32(REG_APBIPRST, TMR0RST | TMR1RST);
    outp32(REG_APBIPRST, 0);
    outp32(REG_AHBCLK,u32Cke);
    sysEnableCache(CACHE_WRITE_BACK);

    u32ExtFreq = sysGetExternalClock();     /* KHz unit */
    outp32(REG_DQSODS, 0x1010);
    outp32(REG_CKDQSDS, E_CLKSKEW);
    if(u32ExtFreq==12000)
    {
        outp32(REG_SDREF, 0x805A);
    }
    else
    {
        outp32(REG_SDREF, 0x80C0);
    }
#ifdef __UPLL_192__
    if((inp32(REG_CHIPCFG) & SDRAMSEL) == 0x20)     /* Power On Setting SDRAM type is
DDR2 */
    {
        outp32(REG_SDMR, 0x432);
        outp32(REG_DQSODS, 0x00001010);
        outp32(REG_MISCPCR,0x00000001);            /* Driving strength */
        outp32(REG_SDTIME, 0x21667525);
    }
    else if((inp32(REG_CHIPCFG) & SDRAMSEL) == 0x30)     /* Power On Setting SDRAM type
is DDR */
    {
```

```
    #ifdef __DDR_75__
        outp32(REG_SDTIME, 0x098E7549); /* DDR Speed grade-75 */
    #endif
    #ifdef __DDR_6__
        outp32(REG_SDTIME, 0x094E7425); /* DDR Speed grade-6 */
    #endif
    #ifdef __DDR_5__
        outp32(REG_SDTIME, 0x094E6425); /* DDR Speed grade-5 */
    #endif
        outp32(REG_SDMR, 0x22);         /* Cas Latency = 2 */
    }

    sysSetSystemClock(eSYS_UPLL,    //E_SYS_SRC_CLK eSrcClk,
                    192000,         //UINT32 u32PllKHz,
                    192000,         //UINT32 u32SysKHz,
                    192000,         //UINT32 u32CpuKHz,
                    192000/2,       //UINT32 u32HclkKHz,
                    192000/4);      //UINT32 u32ApbKHz
#endif

    /* enable UART */
    sysUartPort(1);
    uart.uiFreq = u32ExtFreq*1000;    /* Hz unit */
    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("SPI Loader start (%s).\n", DATE_CODE);
    sysSetLocalInterrupt(ENABLE_IRQ);
    sysFlushCache(I_D_CACHE);
}
```

RTC Initial

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

```
//#define __No_RTC__
…
int main(void)
{
    …
#ifdef __No_RTC__
    sysprintf("* Not Config RTC\n");
    outp32(REG_APBCLK, inp32(REG_APBCLK) & ~RTC_CKE);
#else
    if(inp32(INIR) & 0x1)
    {
        sysprintf("* Enable HW Power Off\n");

        count = 0;

        outp32(REG_APBCLK, inp32(REG_APBCLK) | RTC_CKE);

        outp32(AER,0x0000a965);

        while(1)
        {
            if((inp32(AER) & 0x10000) == 0x10000)
                break;
            if(count > 1000000)
            {
                sysprintf("Write RTC Fail!!\n");
                break;
            }
            count++;
        }
        outp32(PWRON, 0x60005);   /* Press Power Key during 6 sec to Power off
(0x'6'0005) */
        outp32(RIIR,0x4);
    }
    else
    {
        if((inp32(INIR) & 0x1) == 0)
        sysprintf("RTC is in-active!!\n");
    }

#endif
```

SPU Initial

The Spi Loader also needs to initialize SPU to avoid pop noise and set SPU during the SPI read operaion instead of delay.

```
int main(void)
{
   …
   spuDacOnLoader(2);
   …
#ifdef __DAC_ON__

               outpw(REG_SPU_DAC_VOL, inpw(REG_SPU_DAC_VOL) & ~0x0800000); /* P7 */

               …

               for(j=0;j<4;j++)
               {
                   SPIReadFast(0, start_addr + size * j, size, (UINT32*) (executeAddr +
size * j));

                   switch(j)
                   {
                       case 0:;
                           outpw(REG_SPU_DAC_VOL, inpw(REG_SPU_DAC_VOL) & ~0x0400000);
/* P6 */
                           break;
                       case 1:
                           outpw(REG_SPU_DAC_VOL, inpw(REG_SPU_DAC_VOL) & ~0x01e0000);
/* P1-4 */
                           break;
                       case 2:
                           outpw(REG_SPU_DAC_VOL, inpw(REG_SPU_DAC_VOL) & ~0x0200000);
/* P5 */
                           break;
                   }
               }
               fileLen = fileLen - size * 4;
               if(fileLen)
                   SPIReadFast(0, start_addr + size * 4, fileLen, (UINT32*) (executeAddr
+ size * 4));

               outpw(REG_SPU_DAC_VOL, inpw(REG_SPU_DAC_VOL) & ~0x00010000);  /* P0 */

               outpw(REG_SPU_DAC_VOL, inpw(REG_SPU_DAC_VOL) | 0x00001F1F);

               outp32(REG_AHBCLK, inp32(REG_AHBCLK) | ADO_CKE | SPU_CKE | HCLK4_CKE);
/* enable SPU engine clock */
               /* Initial SPU in advance for linux set volume issue */
               spuOpen(eDRVSPU_FREQ_8000);
#else
               SPIReadFast(0, startBlock * 0x10000, fileLen, (UINT32*)executeAddr);
#endif
   …
}
```

Security function

The security function is to provide anti-copy function to prevent the reproduction. The security function 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 the 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 */
    SPIReadFast(0, 63*1024, 1024, (UINT32*)imagebuf);  /* offset, len, address */
```

Load Image from SPI Flash to DRAM

After got Image Information Table, SpiLoader will found out the Logo image first and then copy it from SPI Flash to DRAM. 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);
                        SPIReadFast(0, startBlock * 0x10000, fileLen,
(UINT32*)executeAddr);
                        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);
                        SPIReadFast(0, startBlock * 0x10000, fileLen,
(UINT32*)executeAddr);
                        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);

                        sysSetGlobalInterrupt(DISABLE_ALL_INTERRUPTS);
                        sysSetLocalInterrupt(DISABLE_FIQ_IRQ);
```

```
                        SPIReadFast(0, startBlock * 0x10000, fileLen,
(UINT32*)executeAddr);

                        sysSetGlobalInterrupt(DISABLE_ALL_INTERRUPTS);
                        sysSetLocalInterrupt(DISABLE_FIQ_IRQ);
                        /* Invalid and disable cache */
                        sysDisableCache();
                        sysInvalidCache();

                        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\n");


                        //lcmFill2Dark((char *)(FB_ADDR | 0x80000000));
                        outp32(REG_AHBIPRST, JPGRST | SICRST |UDCRST | EDMARST);
                        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;
        }
   }

   return(0); /* avoid compilation warning */
}
```

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);

                 SPIReadFast(0, startBlock * 0x10000, 64, (UINT32*)IMAGE_BUFFER);
```

```
                u32Result = do_bootm(IMAGE_BUFFER, 0, CHECK_HEADER_ONLY);
                sysSetGlobalInterrupt(DISABLE_ALL_INTERRUPTS);
                sysSetLocalInterrupt(DISABLE_FIQ_IRQ);


                if(u32Result)         /* Not compressed */
                        SPIReadFast(0, startBlock * 0x10000, fileLen,
(UINT32*)executeAddr);
                else                  /* compressed */
                {
                        SPIReadFast(0, startBlock * 0x10000, fileLen,
(UINT32*)IMAGE_BUFFER);
                        do_bootm(IMAGE_BUFFER, executeAddr, LOAD_IMAGE);
                }

                sysSetGlobalInterrupt(DISABLE_ALL_INTERRUPTS);
                sysSetLocalInterrupt(DISABLE_FIQ_IRQ);
                /* Invalid and disable cache */
                sysDisableCache();
                sysInvalidCache();

                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\n");

                //lcmFill2Dark((char *)(FB_ADDR | 0x80000000));
                outp32(REG_AHBIPRST, JPGRST | SICRST |UDCRST | EDMARST);
                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, 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.