N9H26 NAND Loader Reference Guide

Document Information

Abstract	Introduce the steps to build and launch NAND Loader 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 NAND loader is a firmware stored at the NAND Flash chip for booting purpose. It will set the system clock, initialize the relevant modules, and then load the next firmware to DRAM to execute.

The NAND loader supports the following features:

- Initialize more modules such as SPU, RTC, and so on.
- Load Logo image to DRAM if it existed at NAND Flash chip.
- Load next firmware to DRAM if it existed at NAND Flash chip.
- Execute next firmware. Normally, it should be NVT loader.

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

2 NAND Loader BSP Directory Structure

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

2.1 Loaders\NANDLoader

GCC	The GCC project files for the NAND loader.
KEIL	The KEIL project files for the NAND loader.
USER_DEFINE	The sample codes supports user defined feature.
NandLoader.c	The main function for the NAND loader.
NandLoader.scf	The scatter file for the NAND loader
NandDrv.c	NAND Flash driver of N9H26.
Other files	System driver of N9H26.

2.2 Loaders\Binary

N9H26_NANDLoader_xxx.bin	The binary file of the NAND loader for different project targets.

3 NAND Loader Source Code

Complete source codes are included in the *N9H26 BSP Loaders\NANDLoader* 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 3-1 Keil MDK License Chart

3.2 Project Structure

The NAND loader project includes one main function file and some driver files of N9H26. It doesn't link any driver library in order to shrink the binary code size.

Please note that the binary code size of the NAND loader MUST less than (page size * (page number per block - 2)) of the NAND Flash chip on board.

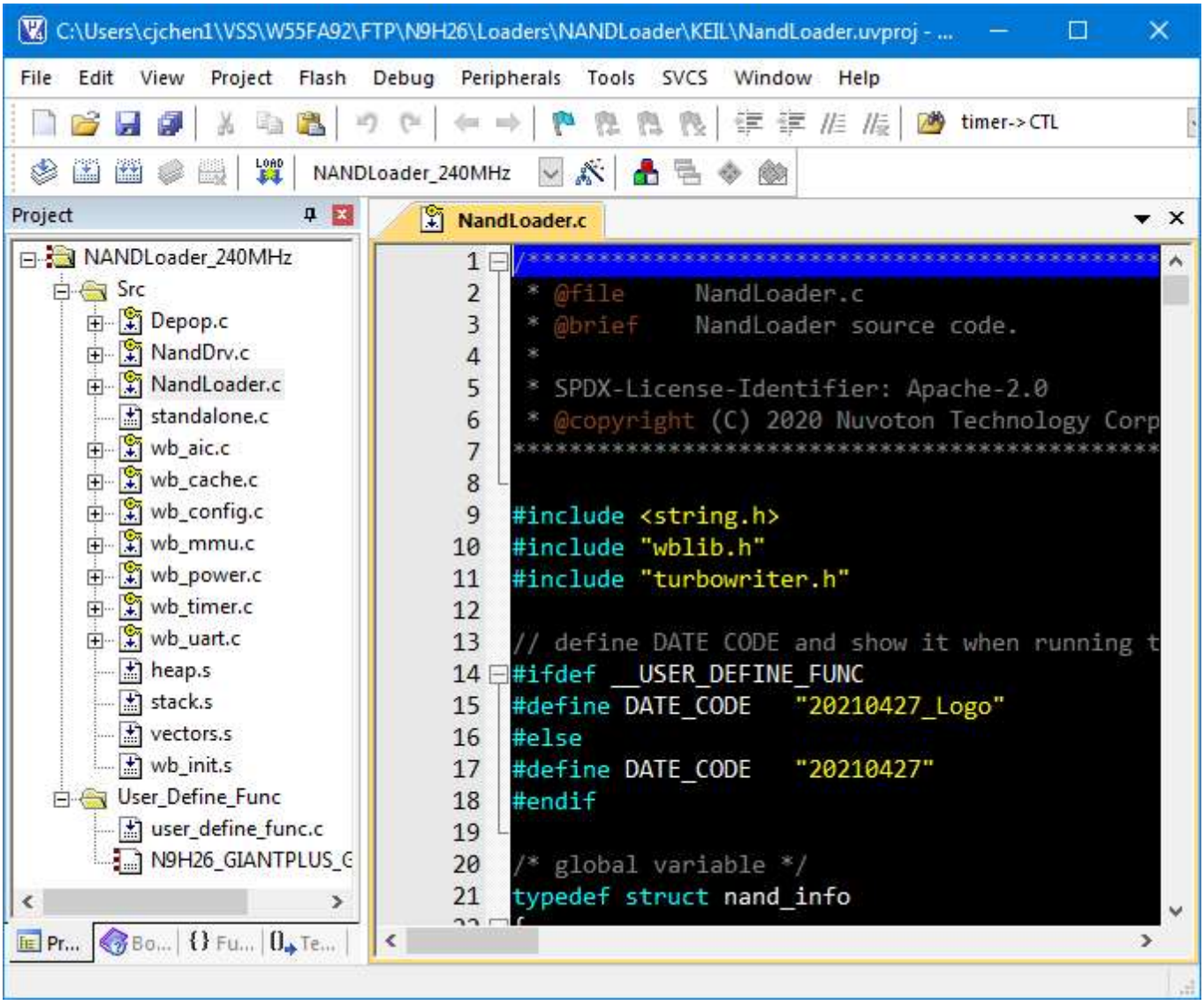

Figure 3-2 NAND Loader Project Tree on Keil MDK

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

- **NANDLoader_240MHz**: Set system core clock to 240MHz. This is the official standard target.
- **NANDLoader_240MHz_Logo**: Set system core clock to 240MHz and support user defined feature. Please refer to section **Error! Reference source not found.** for further information.
- **NANDLoader_192MHz**: Set system core clock to 192MHz.

3.3 System Initialization

The system initialization code is located in main function, including system code clock setting, enable cache feature, and UART debug port setting. It also initializes some necessary peripherals during the boot process.

```
int main()
{
    NVT_NAND_INFO_T image;
    int volatile count, i;

    /* Clear Boot Code Header in SRAM to avoid booting fail issue */
    outp32(0xFF000000, 0);

    //--- enable cache to speed up booting
    sysDisableCache();
    sysFlushCache(I_D_CACHE);
    sysEnableCache(CACHE_WRITE_BACK);

    sysprintf("N9H26 Nand Boot Loader entry (%s).\n", DATE_CODE);

    //--- initial SPU
    DrvSPU_Open();
    spuDacPLLEnable();

    /* PLL clock setting */
    initClock();
    spuDacPrechargeEnable();
    sysprintf("System clock = %dHz\nDRAM clock = %dHz\nREG_SDTIME = 0x%08X\n",
              sysGetSystemClock(), sysGetDramClock(), inp32(REG_SDTIME));

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

3.4 NAND Flash Initialization

One of the major tasks of the NAND loader is to copy the next firmware on the NAND Flash to DRAM for execution. To initialize the NAND Flash driver, both fmiInitDevice() and sicSMInit() must be called in source code.

```
    /* Initial DMAC and NAND interface */
    fmiInitDevice();
    sicSMInit();
    memset((char *)&image, 0, sizeof(NVT_NAND_INFO_T));
```

```
    /* read physical block 0~3 - image information */
    for (i=0; i<4; i++)
    {
        if (!sicSMpread(0, i, pSM0->uPagePerBlock-2, imagebuf))
        {
            if (((*(pImageList+0)) == 0x574255aa) && ((*(pImageList+3)) == 0x57425963))
            {
                sysprintf("Get image information from block 0x%x ..\n", i);
                break;
            }
        }
    }
```

3.5 User Defined Feature

The NAND loader allows the user to execute user defined functions before the NAND loader executes the next firmware. For example, the NAND loader can display a Logo on the LCD panel as soon as possible after booting.

Please select project target "**NANDLoader_240MHz_Logo**" to enable this feature.

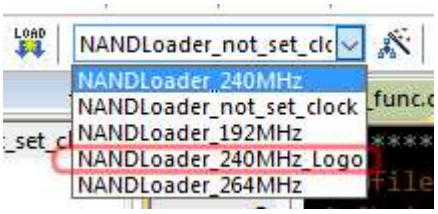

Figure 3-3 The NAND loader project target for user defined feature

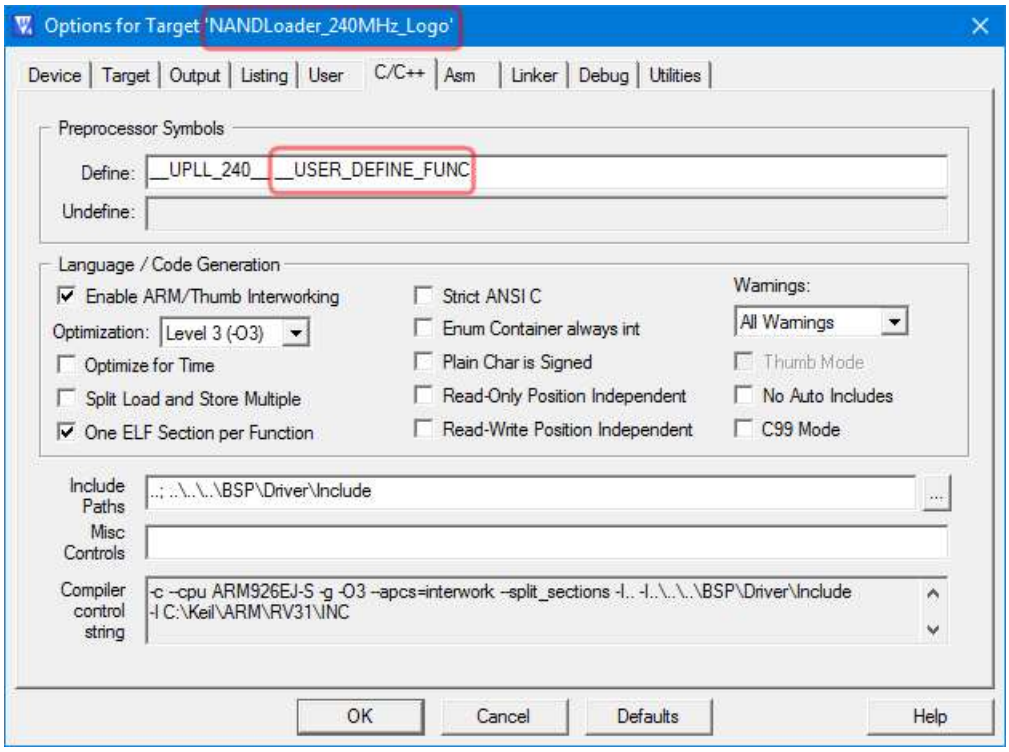

Figure 3-4 The NAND loader project definition for user defined feature

Within main() function, the **user_define_func()** is called to execute user defined function before load next firmware.

```
#ifdef __USER_DEFINE_FUNC
        //--- call user define function before jump to next application.
        user_define_func();
#endif

        /* load execution file */
        pImageList = pImageList+4;
        for (i=0; i<count; i++)
        {
            if (((*(pImageList) >> 16) & 0xffff) == 1)  // execute
            {
                image.startBlock = *(pImageList + 1) & 0xffff;
                image.endBlock = (*(pImageList + 1) & 0xffff0000) >> 16;
                image.executeAddr = *(pImageList + 2);
                // sysprintf("executing address = 0x%x\n", image.executeAddr);
                image.fileLen = *(pImageList + 3);
                MoveData(&image, TRUE);
                break;
```

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

Within source code file *NANDLoader\USER_DEFINE\user_define_func.c*, the function user_define_func() can be implemented to do the user defined function.

```
#ifdef __USER_DEFINE_FUNC
#include "user_define_func.h"

/*---------------------------------------------------------------------------------------*/
/* The entry point of User Define Function that called by NandLoader.                    */
/*---------------------------------------------------------------------------------------*/
void user_define_func()
{
    /* Do something here for user defined function */
}

#else

/*---------------------------------------------------------------------------------------*/
/* The entry point of User Define Function that called by NandLoader.                    */
/*---------------------------------------------------------------------------------------*/
void user_define_func()
{
    //--- Keep empty if user define nothing for User Define Function.
}

#endif  // end of #ifdef __USER_DEFINE_FUNC
```

3.6 Build NAND Loader Project

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

Figure 3-5 Shortcut Icon to Rebuild the NAND Loader on Keil MDK

The binary file of NAND loader will be copied to the *Loaders\Binary* folder with the file name *N9H26_NANDLoader_xxx.bin*. The "*xxx*" is depend on the project target. For the **NANDLoader_240MHz** project target, the binay file name is *N9H26_NANDLoader_240MHz_Fast.bin*.

Please note that the binary code size of the NAND loader MUST less than (page size * (page number per block - 2)) of the NAND Flash chip on board.

4 Download and Run

4.1 Download NAND Loader Binary to NAND Flash

The NAND loader binary on NAND Flash can be programmed by the tool *TurboWriter* and here are the steps. Further information about *TurboWriter* can be found at BSP *Tools/PC_Tools/TurboWriter Tool User Guide.pdf*.

1. Power off device.
2. Plug in USB cable to PC/NB.
3. Power on device under Recovery mode.
4. Run TurboWriter for N9H26 version on PC/NB.
5. Wait for the TurboWriter message to change to "**Mass Storage Connected !**". If not, press the "**Re-Connect**" button to reconnect the device.
6. Select "**NAND**" on the option "**Please choose type**".
7. Select NAND loader binary file on the option "**Image Name**".
8. Select "**System Image**" on the option "**Image Type**".
9. Press "**Burn**" button to burn the NAND loader binary into NAND Flash.
10. After burning completed, check the NAND loader information in the left table.

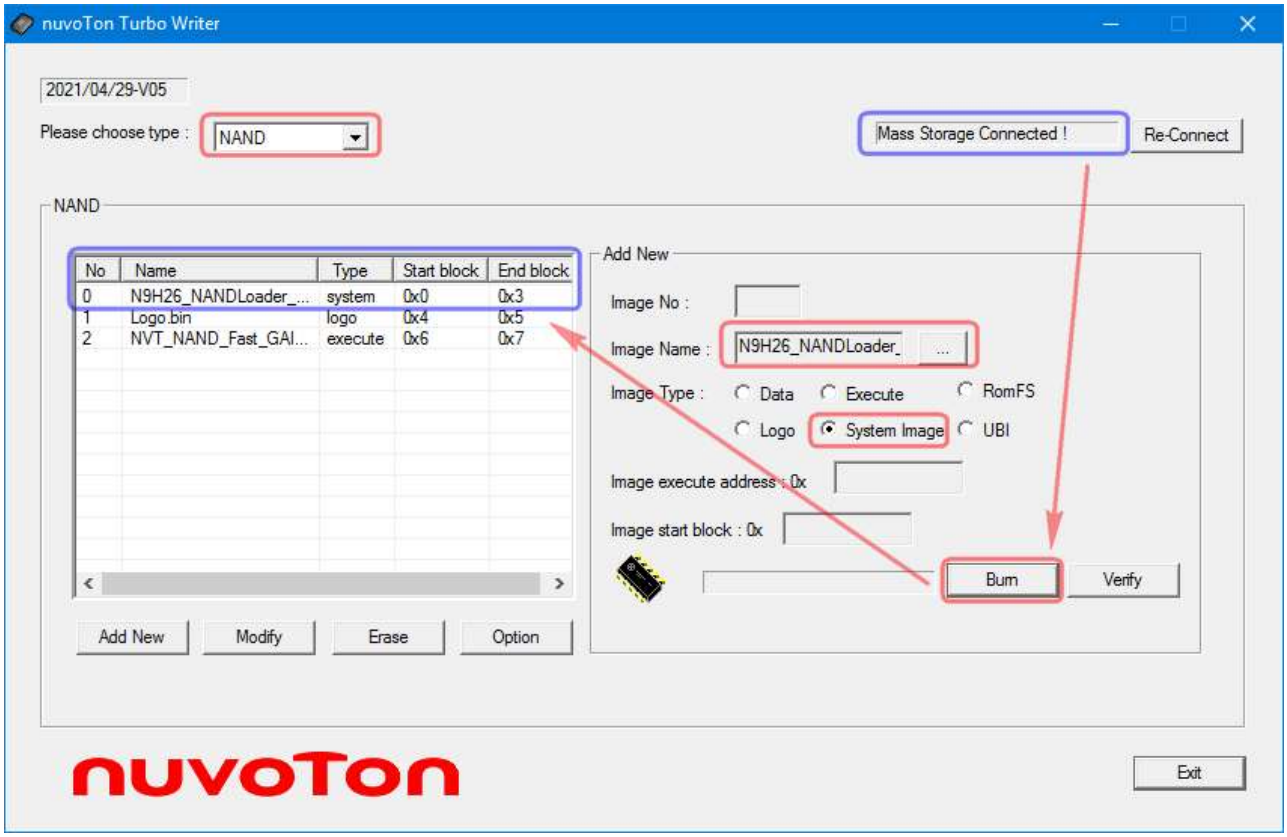

Figure 4-1 Programmed NAND Loader by TurboWriter

11. Remove USB device safely.
12. Plug out USB cable from PC/NB.
13. Reset the device under Normal mode.

4.2 Run NAND Loader

N9H26 has built-in 16K bytes IBR (Internal Booting ROM) where is the boot loader to initialize chip basically when power on, and then try to find out the next stage loader from different type of storage. It could be SD card, NAND Flash, SPI Flash, or USB storage. The booting sequence by the IBR as Figure 4-2.

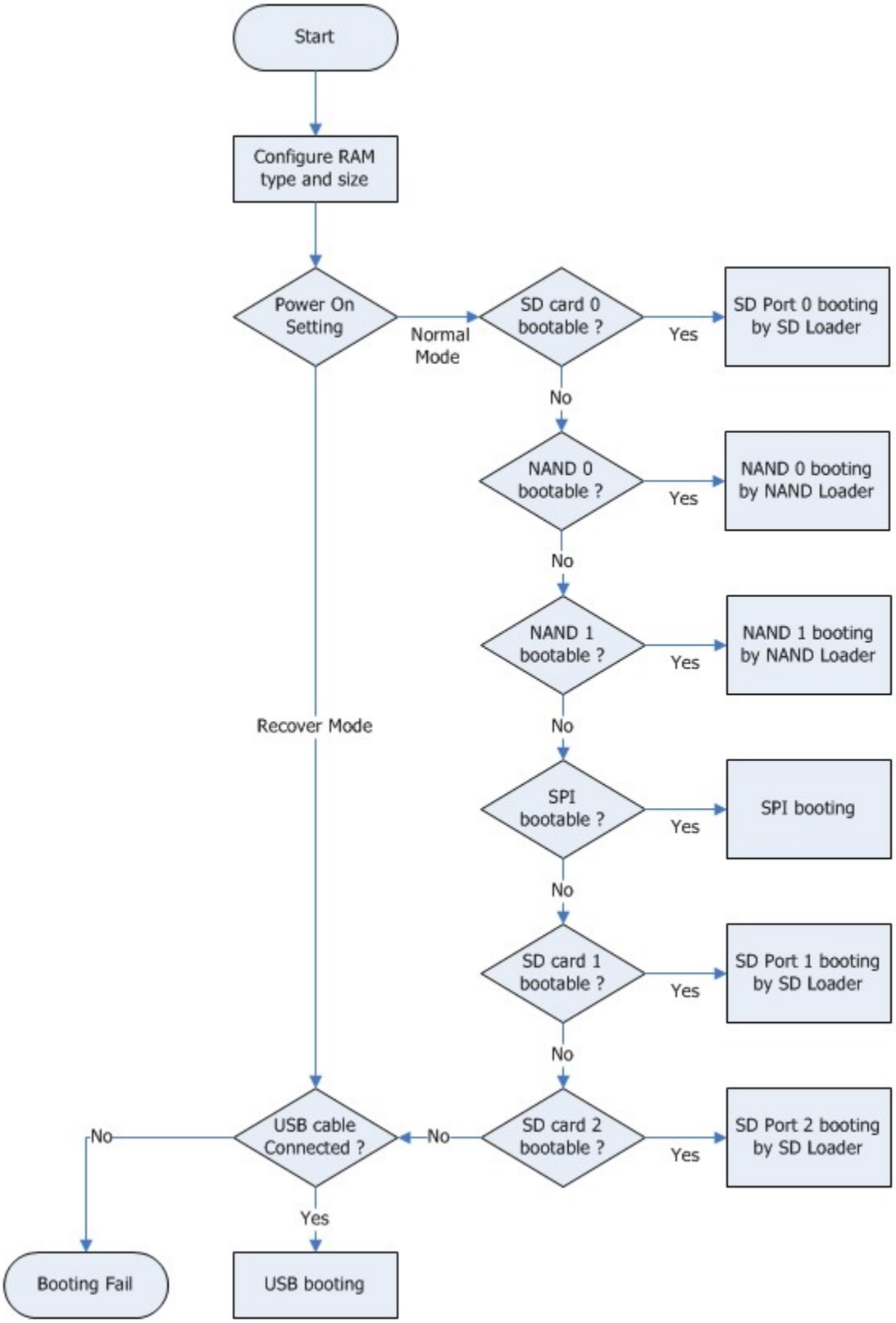

Figure 4-2 IBR Booting Sequence after Power On

The IBR will execute the NAND loader if SD loader on SD card 0 is invalid.

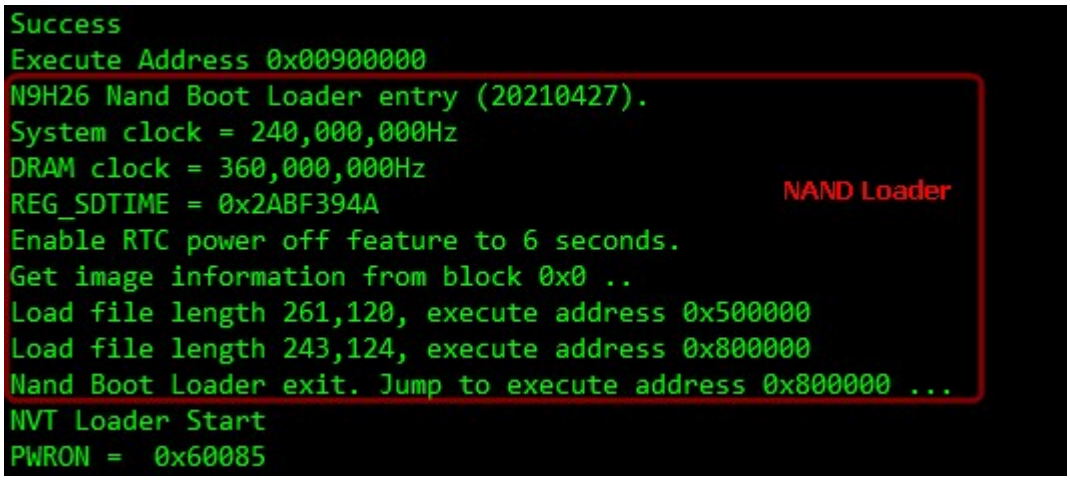

Figure 4-3 The NAND Loader Runs on N9H26

5 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
2021.6.15	1.01	1. Modify document structure.
2018.5.4	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.