How to boot from SPI ROM

V1.0

Publication Release Date: Sept. 2008

Support Chips:

NUC501

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.

Table of Contents

1. Introduction

NUC501 supports to boot from SPI ROM/Flash with 32KB internal SRAM included. The SPI ROM/Flash could be mapping to the memory space start form 0x400000000 and could be accessed by CPU directly. Whenever the CPU fetches the code or data in the mapping memory of SPI ROM/Flash, the NUC501 will translate the fetching command to SPI bus to get the data form SPI ROM/Flash. By the way, it is supported to program the boot code and applications into SPI ROM/Flash and boot form SPI to execute them.

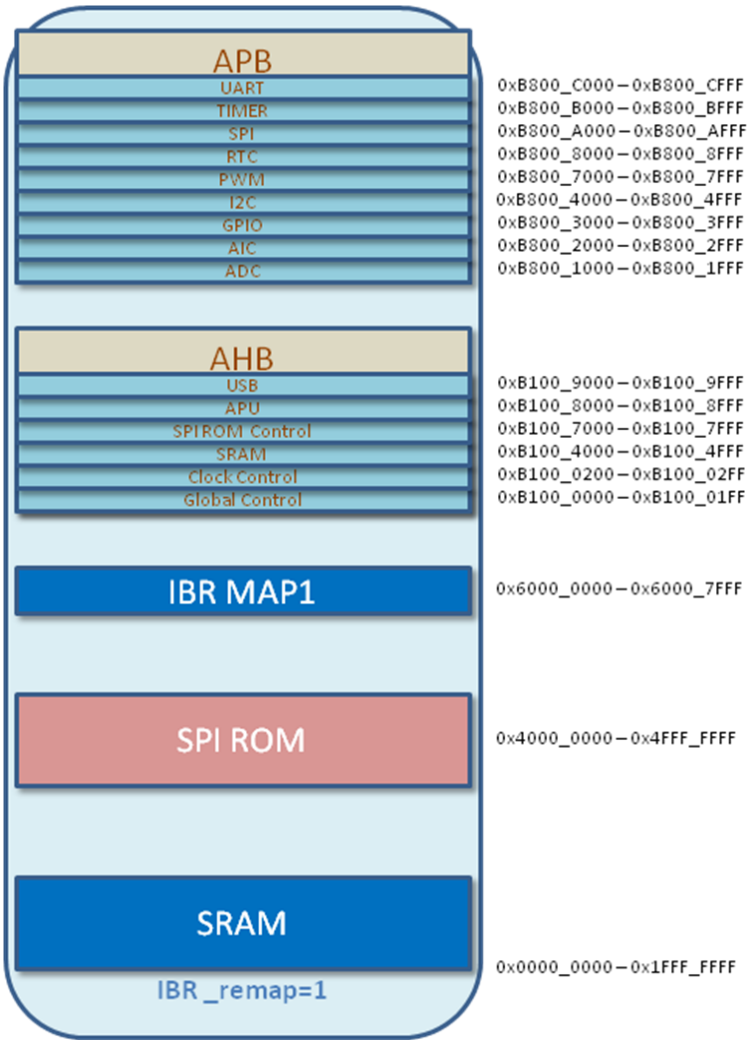
NUC501 didn't include internal flash ROM to store the code; however, the SPI ROM/Flash could be included to do the same thing. Furthermore, the memory size of embedded flash IC is fixed but the SPI ROM/Flash size used by NUC501 can be changed according to the applications.

Besides the function of booting form SPI, NUC501 also includes 32KB SRAM, thus it is possible to speed up the application by executing the critical functions in SRAM. By well define the application execution flow, it is possible to full use the 81MIPS computation power of NUC501.

2. System Memory Map

2.1. Internal Boot ROM

There is an internal boot ROM (IBR) embedded in NUC501 and it is the first code to be executed by CPU. After booting by internal boot ROM, the memory map of NUC501 is shown as following figure:



IBR will map the 32KB SRAM to address 0x0 and detect the boot jump setting to boot from SPI, SRAM or USB. If the jump setting is set to boot from SPI, IBR will copy 16KB data from SPI ROM/Flash into the front of the SRAM then set the program counter to 0x0 to execute the code in SRAM.

2.2. Direct Memory Mode

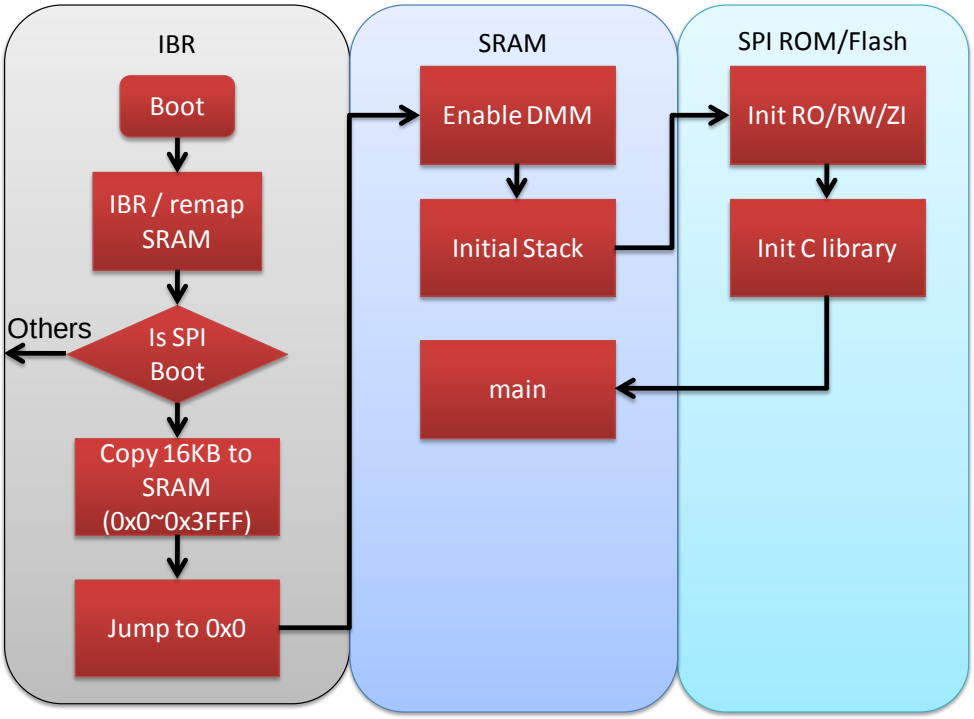
The SPI ROM/Flash will be mapped to 0x40000000 by SPIM module of NUC501 when direct memory mode (DMM) is enabled. In other words, once the DMM mode is enabled, CPU can fetch the code or data in SPI ROM/Flash as memory and this is what called XIP @ SPI ROM/Flash (Execute in Place @ SPI ROM/Flash). However, it is necessary to enable the DMM mode before we can execute code in SPI ROM/Flash directly. If the DMM is disabled, the data in 0x40000000 ~ 0x4FFFFFFF would be all 0xFFFFFFFF.

3. An Example of Boot from SPI

In this section, an example of boot from SPI ROM/Flash is provided to show how to write a ROM code for NUC501. The tool chain used in this example is ARM ADS or Keil MDK, however, the concept could be applied to any tool chain. In this example, the most code is placed in SRAM and only necessary codes are placed in SPI ROM/Flash. It would be ok to modify the linker script of the example to place the code to SPI ROM/Flash if necessary.

3.1. Boot Flow

The following figure shows the boot flow of NUC501 when boot from SPI is enabled:



The IBR is the first code to be executed when booting. If the SPI boot is enabled, it will copy 16KB data from SPI ROM/Flash to SRAM and jump to 0x0 to execute the code in SRAM. The code in the front of SPI ROM/Flash must contain the startup code and it enables the DMM mode firstly to map the SPI ROM/Flash to 0x40000000. After the DMM mode enabled, the startup code start to initial the programming execution environment including stack, read only region, read write region, zero initial region, and all necessary initialization for C library. If the entire program execution environment is ready, the startup code calls the main function to start user's application. Because the startup code must be executed before the main function, it must be placed in SPI ROM/Flash

in this example. However, the main function is possible to be place in SRAM or SPI ROM/Flash by modifying the linker script.

3.1.1. Enable DMM

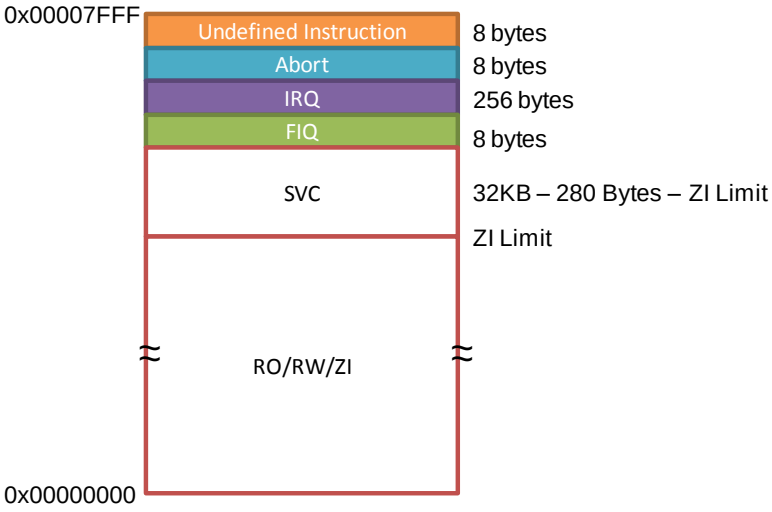
It is recommended to enable the DMM mode at the beginning of the startup code, and the following code segment is the assemble code to enable DMM mode:

```
;/*---------------------------------------------------------------------------*/
;/* Init Direct Memory Mode                                                   */
;/*---------------------------------------------------------------------------*/
InitDMM
    LDR r0, =0xb1007008     ; Auto chip select
    LDR r1, =0x00000009     ;
    STR r1, [r0, #0x00]

    LDR r0, =0xb1007000     ; DMM mode
    LDR r1, =0x0B321344     ;
    STR r1, [r0, #0x00]

    LDR r0, =0xb1007004     ; Divider
    LDR r1, =0x007F0000     ;
    STR r1, [r0, #0x00]

    LDR r0,=0xb1000034
    LDR r1,=0x140
    STR r1, [r0, #0x00]
```

These codes are used to enable SPIM and configure the SPIM to enable the DMM mode by setting the relative control registers. Therefore, It is also possible to set these registers by ICE to enable DMM mode.

3.1.2. Initial Stack

The embedded 32KB SRAM of NUC501 is able to be the stack. The stack in this example is start from the top of 32KB SRAM and the size of each exception mode is as follows:



Because the exception handlers of undefined instruction, Abort and FIQ are just forever loops, their stack sizes are all 8 bytes. The IRQ stack size is 256 bytes and SVC stack size is dependent on ZI limit of the program. The stack size definitions are in the startup code and they are as follows:

```
RAM_Limit       EQU     0x00008000


UND_Stack_Len   EQU     8
Abort_Stack_Len EQU     8
IRQ_Stack_Len   EQU     256
FIQ_Stack_Len   EQU     8
SVC_Stack_Len   EQU     128


UND_Stack       EQU     RAM_Limit
Abort_Stack     EQU     UND_Stack   - UND_Stack_Len
IRQ_Stack       EQU     Abort_Stack - Abort_Stack_Len
FIQ_Stack       EQU     IRQ_Stack   - IRQ_Stack_Len
SVC_Stack       EQU     FIQ_Stack   - FIQ_Stack_Len
USR_Stack       EQU     SVC_Stack   - SVC_Stack_Len
```

The user stack is not used in the example, thus it is ok to overlap with SVC stack.

3.1.3. Initial RO/RW/ZI and C library

The C library of ARM compiler will initial the RO (Read-Only region), RW (Read-Write region), and ZI (Zero-Initial region) by calling __main function. The __main function will initial these regions according to the linker script file , thus the startup code also call this function to initial these regions. The __main function also does the necessary initializations for C library and calls the user's main function after all initializations.

3.1.4. The linker script – scatter loading file

The ARM ADS and Keil MDK both use scatter loading file to be their linker script. In this example, the scatter loading file is “ApplicationROM.scf” and the contents of the scatter loading file are as follows:

```
SPI_ROM 0x40000000
{
   ROM 0x40000000
   {
        AppInitROM.o(app_init, +First)
        anon$$obj.o
        __main.o
    }
   RAM 0x58 0x7E00
   {
        * (+RO)
        * (+RW, +ZI)
   }
}
```

The “SPI_ROM 0x40000000” indicates the load region is called SPI_ROM and its base address is 0x40000000. The “ROM 0x40000000” indicates an execute region is called ROM and its base address is 0x40000000. Furthermore, because the base address of ROM is the same with SPI_ROM, it is called “root region” and the startup code, anon$$obj.o, and __main.o must be in the root region. As mentioned above, the startup code must be executed firstly in the user’s application, thus the “+First” attrib is used.

The “RAM 0x58 0x7E00” indicates a memory region called RAM, its start address is 0x58 and size is 0x7E00. The memory spaces before 0x58 are reserved for exceptions. “* (+RO)” indicates all code and data with “RO” attrib to be placed from 0x58, next is all data with “RW” attrib and finally, all data with “ZI” attrib.

After link the program according to the previous scatter loading file, all codes from user’s application will be executed in SRAM. If the user wants to execute most code in SPI ROM/Flash and execute main function in SRAM, the scatter loading file may be able to be modified as follows:

```
SPI_ROM 0x40000000
{
   ROM 0x40000000
   {
        AppInitROM.o(app_init, +First)
        anon$$obj.o
        __main.o
        * (+RO)
    }
   RAM 0x58 0x7E00
   {
       Smpl_DrvGPIO.o
        * (+RW, +ZI)
   }
}
```

The new scatter loading file indicates all codes are executed in SPI ROM/Flash except the code in Smpl_DrvGPIO.o.

Please refer to the linker documents of ARM for more information of scatter loading file.

3.1.5. Writing Code for ROM

The ARM C library default to use semihosting for standard I/O and that needs to be retarget to the hardware of NUC501, this is what called retarget.

The file “NVT_Platform.c” includes the retarget functions to retarget the I/O to NUC501, and then the code could be executed standalone.

For more detail information about how to write code for ROM, please refer to ARM Developer Guide.

4. Revision History

Version	Date	Description
V1.0	Mar. 23, 2009	● 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.