

Application note

Use the MMU to protect FLASH partitions in multi-user scenarios

Introduction

In the process of embedded product development, sometimes there are scenarios in which multiple users are required to develop application software in stages within a single MCU. In this scenario, the codes and data of each user may not be shared with other users due to copyright or security considerations. So how do you solve these problems?

This document is mainly aimed at the above-mentioned application scenarios of the NSING MCU series products, and guides users how to use the NSING MCU, through the built-in memory management unit (Memory Management Unit, MMU) to achieve the multi-user area division and access permissions of the FLASH main storage area Management, so as to solve the problem of code copyright protection and data security in the multi-user development process. Therefore, it can be widely used in various copyright protection, sensitive data and multi-user code protection scenarios. This document is only applicable to NSING MCU products with built-in MMU. Currently, the supported product series include N32G452 series, N32G455 series, N32G457 series, N32G4FR series, N32WB452 series, N32L43x series, N32G43x series and N32L40x series.

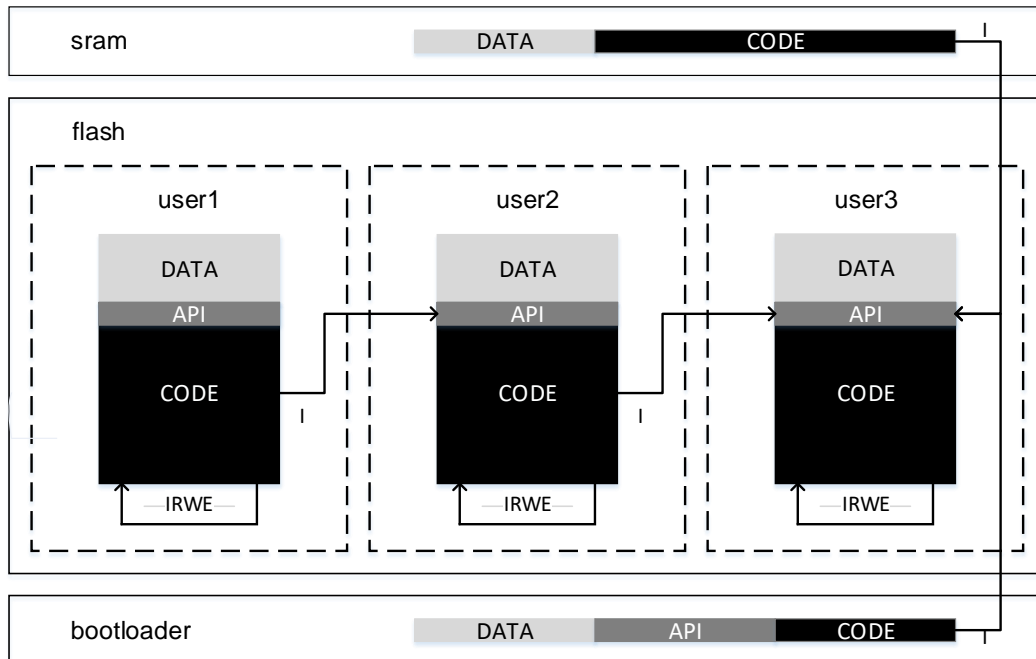
CONTENTS

1 Implementation mechanism of partition protection	1
2 Function description of the MMU	2
2.1 User area division	2
2.2 Access permission management	3
3 Instruction	5
3.1 The operating environment	5
3.2 Operation steps	5
3.2.1 Device enters the Bootloader	5
3.2.2 Device connection tool	5
3.2.3 Configuration partition	6
3.2.4 Program download	7
3.2.4.1 Download program through the debugging interface	7
3.2.4.2 Download program through the built-in bootloader	8
4 The sample project	12
4.1 Section address configuration	12
4.1.1 SCT distributed load file	12
4.2 Generating a bin file	14
4.3 Partition access operation	15
4.3.1 Call API	16
4.3.2 Read and write data MMU abnormal alarm	18
4.3.3 Interrupt handling	18
5 Conclusion	20
6 Version history	21
7 Notice	22

1 Implementation mechanism of partition protection

Generally, the FLASH memory (FLASH) in the MCU chip is connected to the memory bus, and the CPU can access any area in the FLASH without limit. Multiple user areas should be divided and protected for FLASH in a single MCU to avoid different users reading or modifying FLASH contents in other user areas directly by CPU instructions. We can use the Nations MCU built-in MMU, set the FLASH the main storage area partition and access, at the same time can protect all the application code and data storage area from illegal access and tampering, and indicates the memory and the protected registers of illegal access errors, all unauthorized operation will trigger the MMU abnormal alarm, To achieve multi-user FLASH partition protection.

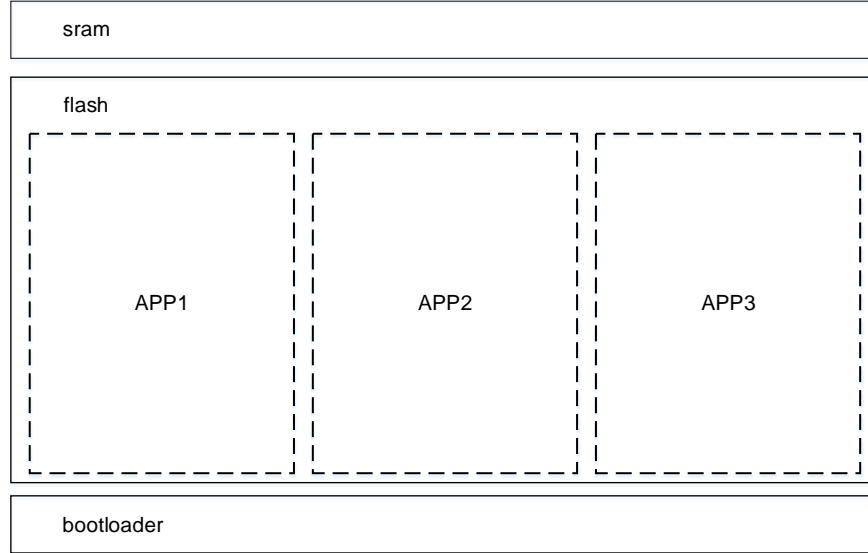
Figure 1-1 MMU partition protection implementation mechanism



2 Function description of the MMU

MMU can realize region division and access permission management of FLASH main storage area, and can divide independent storage space for different applications of MCU (see Figure 2-1), and manage access permission.

Figure 2-1 Memory area division



2.1 User area division

The FLASH main storage area can be divided into USER1 (default), USER2, and USER3 at most. In practice, the user area can be divided into the following situations. For the Settings of each situation, refer to Table 2-1

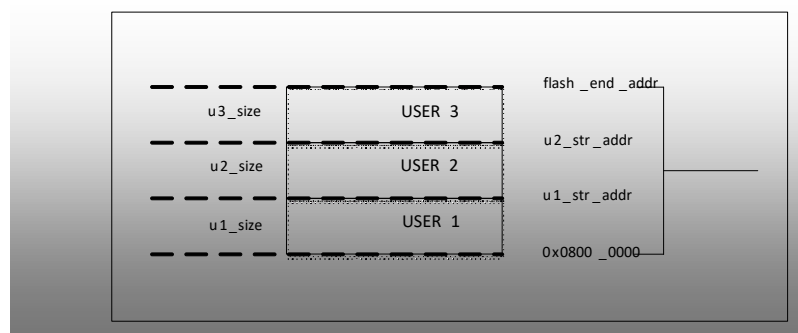
Table 2-1 User Partition Setting Instructions

Description	Partition setup sequence	FLASH space size involving authority management ⁽³⁾			Instructions
		user1	user2	user3	
There are no partitions and the default is the user1 zone	-	-	-	-	The user1 area is flash_size and does not have the access permission management function
Set one partition , do not keep the default user area ¹	user1	flash_size	-	-	
Set one partition, Keep the default user area (user1) ²	user3	-	-	user3_size	The space of user1 and user3 is flash_size. User1 does not have the access permission management function
Set two partitions , do not keep the default user area	user3→user1	user1_size	-	user3_size	The space size of user1 and user3 is flash_size
Set 2 partitions, do not keep the default user area	user3→user2	-	user2_size	user3_size	The space of user1, user2, and user3 is flash_size. User1 does not have the access permission management function

Set up three partitions	user3→user2→user1	user1_size	user2_size	user3_size	User1, user2, and user3 are flash_size
<p>Description:</p> <p>(1) "do not keep the default user area" means that all the space of FLASH main storage area is divided into user areas through partition Settings, and each partition involves access permission management function;</p> <p>(2) " Keep the default user area " means that user1 will not be partitionized, that is, user1 area will be left open and access permission management is not involved;</p> <p>(3) " FLASH space size involving authority management" refers to the FLASH main storage space with partition size set.</p>					

When the FLASH primary storage partition is divided into three regions, USER1 (default), USER2, and USER3, as shown in Figure 2-2. The granularity of the partition is 16KB.

Figure 2-2 FLASH main storage area division relationship



For details about how to set user partitions in the FLASH primary storage area, see Table 2-2. You can divide regions by setting the size of each user partition. Partition Settings are static Settings. Once set, the MCU will automatically load the configuration every time it is powered on. In particular, partition Settings can only be operated once, and the operation is irreversible.

Table 2-2 FLASH main storage area partition setting instructions

Partition users	The storage area	Partition size range
USER1	$0x0800_0000 \sim (0x0800_0000 + u1_size - 1)$	16KB ¹ ~ (flash_size) KB
USER2	$(0x0800_0000 + u1_size) \sim (flash_end_addr - u3_size)$	0 KB ~ (flash_size - 32)KB
USER3	$(flash_end_addr - u3_size + 1) \sim (flash_end_addr)^2$	0 KB ~ (flash_size - 16)KB
<p>Description:</p> <p>(1) The granularity of the partition is 16KB;</p> <p>(2) Flash_end_addr varies according to model, and the corresponding flash_size is also different. Flash_size should be the sum of the size of the flash_area USER1, USER2, and USER3. The size is (flash_end_addr - 0x0800_0000 + 1) KB.</p> <p>Note: User partition Settings cannot be reset</p>		

2. 2 Access permission management

The operation permissions of each area of FLASH main storage area are managed through user area division to realize memory access control. Table 2-3 provides the access permissions of each user area before and after FLASH main storage area division.

Table 2-3 User permission table

	Visited area					
	user1		user2		user3	
Program ownership /	Whether the partitions		Whether the partitions		Whether the partitions	
Access method	N ¹	Y	N	Y	N	Y
user1 code	IRWE ^{2, 3}	IRWE	IRWE	I	IRWE	I
user2 code	IRWE	I	IRWE	IRWE	IRWE	I
user3 code	IRWE	I	IRWE	I	IRWE	IRWE
SRAM1/2 code	IRWE	I	IRWE	I	IRWE	I
DMA	RW	-	RW	-	RW	-
JTAG/SWD	IRWE	I	IRWE	I	IRWE	I
<p>Description:</p> <p>(1) Before the partition, USER1, USER2, and USER3 are regarded as the same area, and all FLASH space is USER1 by default;</p> <p>(2) I represents addressing, R represents reading, W represents writing, and E represents erasing;</p> <p>(3) "Write Protection (WRP) Enable" is at the same level as "Access Rights Management for MMU Partitions".</p> <p>Note: If the USER1 area size is not set (see section 3.2.1 to section 3.2.3 for "Operation step"), the USER1 area does not have access permission management.</p>						

3 Instruction

You can partition the FLASH main memory area in MCU by using the Nations MCU Download Tool on the PC provided by Nations. For details about how to use the Tool, see *Nations MCU Download Tool User Manual*.

3.1 The operating environment

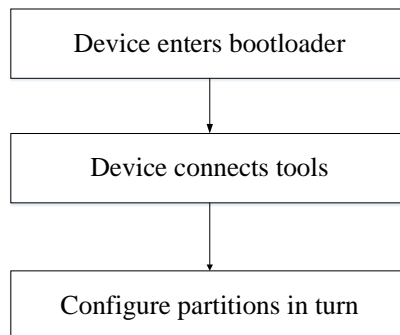
- Hardware environment: PC (Windows XP/7/10), development board N32G43XR-STB V1.0 (including N32G435RBL7 chip)
- Target device: N32G435RBL7 chip
- Software environment: Download tool (Nations MCU Download tool. Exe), USB DFU driver or USB-to-serial driver (optional)

Note: Bootloader supports USB interface or USART interface download, please confirm that the USB DFU driver or USB to serial port driver has been installed before use. At the same time, confirm that the target device has entered the Bootloader state, so that the device can be connected to the download tool normally. For details on how to make the target device enter the Bootloader state, please refer to the user manual of the target device chip. This document uses the N32G435RBL7 chip to download using the USART interface as an example for illustration.

3.2 Operation steps

Figure 3-1 shows the process for dividing user areas in the FLASH main memory area. The following describes how to set partitions.

Figure 3-1 Partition setup steps



3.2.1 Device enters the Bootloader

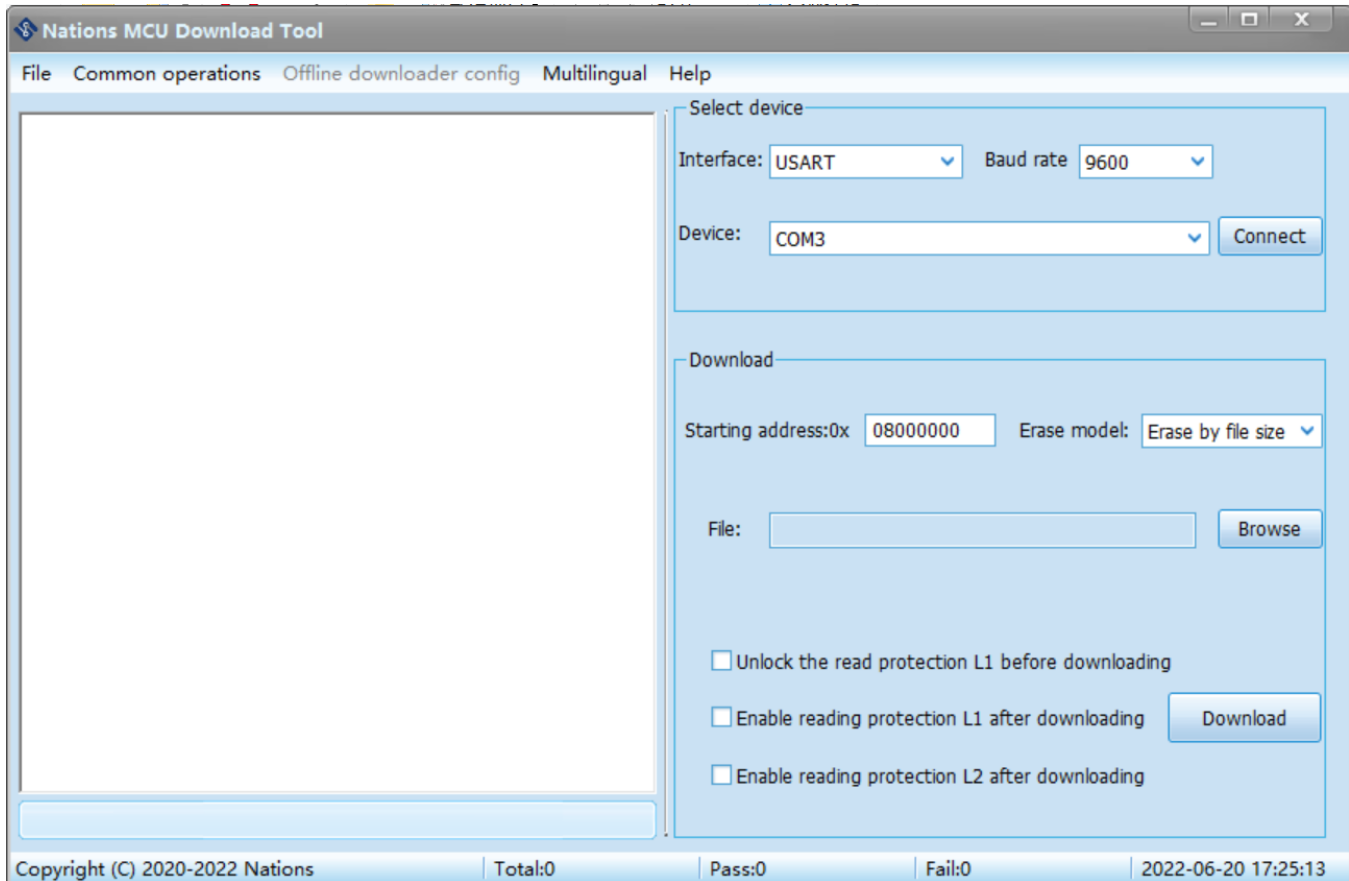
N32G435RBL7 BOOT0(PD0) pin is connected to VDD, the chip is powered on and enters the Bootloader.

Note: For the development board N32G43XR-STB V1.0, use the USART interface, then connect the USB Debug Port interface for power supply, otherwise use the USB COMM interface for power supply.

3.2.2 Device connection tool

Double-click Nations MCU Download Tool.exe to open the download tool. The interface is shown in Figure 3-2. Here, the focus will be on the "Select Device" area. The interface defaults to "USART". Select the matching port number as the device. The "COM port number" can be viewed through the "Device Manager" of the PC. The serial port connected to the MCU in Figure 3-2 is identified as "COM3". At the same time, set the baud rate of USART (the default configuration "9600" can be used), click the "connect" button, the left display interface will prompt "The device is connected!". At this time, the device and the tool have been connected normally. Note: USART1 in the Bootloader of N32G435RBL7 uses PA9 and PA10 as TX and RX respectively. Please ensure that PA9 and PA10 are properly connected to TX and RX of the serial port.

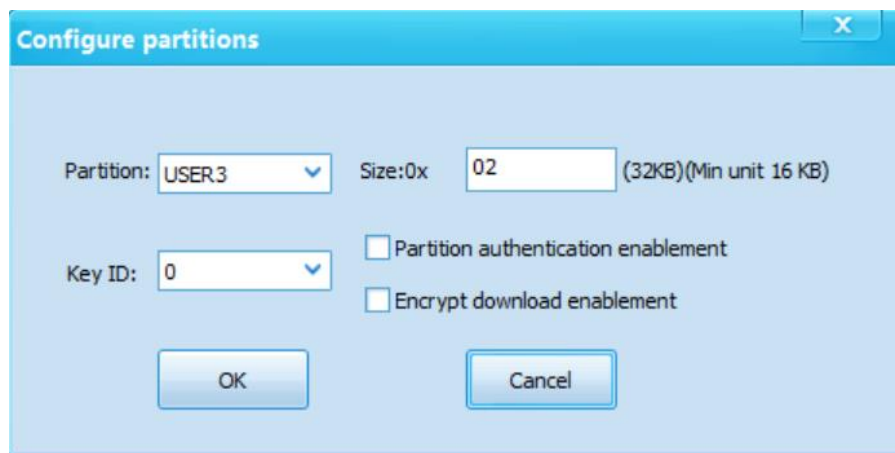
Figure 3-2 Download tool interface



3.2.3 Configuration partition

Click the "Configure partition" button in the "Common operations" area to pop up the Configure Partition dialog box, select the partition user ID (USER1, USER2 or USER3) in turn, and enter the FLASH size of the partition (the value is set in the unit of partition granularity 16KB). As shown in Figure 3-3, suppose you need to partition a 32KB area for USER3, select "USER3" for the partition and enter 0x02 for the size. Click "Configure Partition" to confirm the configuration partition and complete the area division of the current user ID.

Figure 3-3 Interface for configuring partitions



Note:

(1) The partition configuration operation is irreversible, please operate with caution

(2) If multiple partitions need to be configured, each user can enter the Bootloader configuration respectively. For details about the configuration size and sequence, see Table 2-1 and *Nations MCU Download Tool User Manual*. Improper operations may lead to configuration failure.

3.2.4 Program download

After the partition Settings take effect, the user area cannot be accessed using the debugging interface. Therefore, there are two ways to download a user application:

- (1) Before setting the partition, download the program through the debugging interface or Bootloader.
- (2) After setting the partition, download the program through the built-in Bootloader (recommended);

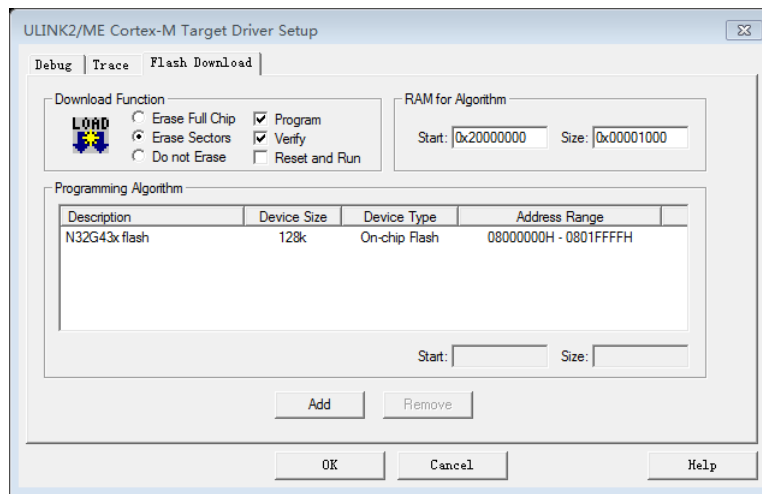
3.2.4.1 Download program through the debugging interface

If the partition is not set, N32G435RBL7 can also use the debugging interface (JTAG or SWD) to download the program of each user. The specific operation steps are the same as the general situation and will not be described again.

The following highlights the precautions for downloading partitioned programs through debugging interfaces:

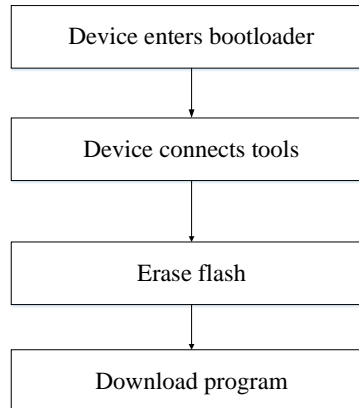
- (1) Ensure that the starting position and size of FLASH and SRAM of the program are correct, and must match with the partition configuration (for setting the starting position and size of FLASH and SRAM, please refer to section 4.1.1 "SCT distributed load file");
- (2) In order to ensure that the debug interface can download multiple programs in batches, in the MDK "Options for Target->Debug ->Use: xx Debugger->Settings->Flash Download" page, "Download Function" must not be selected "Erase Full Chip" "(like Figure 3-4).

Figure 3-4 Interface for Flash download



3.2.4.2 Download program through the built-in bootloader

Figure 3-5 Bootloader download operation steps



In order to ensure the safety of the program update, the built-in Bootloader of the MCU also provides functions such as partition authentication and encrypted download (for the corresponding enabling and downloading procedures, please refer to the *Nations MCU Download Tool User Manual*. Here is the most basic program download process is introduced and described. As shown in Figure 3-5, the Bootloader download operation is roughly divided into 4 steps: the device enters the Bootloader, the device connects to tool, erases the FLASH and downloads the program.

- The specific process of program download is as follows:

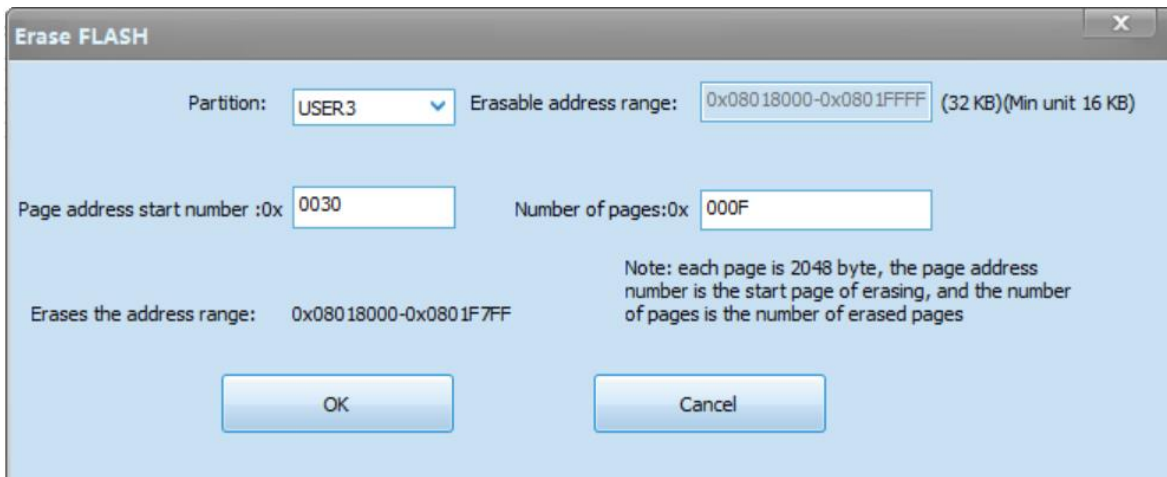
1. Access the Bootloader and connection tool

If the device is connected to a tool, skip this step and go to Step 3 to erase the FLASH. Otherwise, refer to section 3.2.1 and section 3.2.2 to perform Step 1 and Step 2 in sequence to ensure the normal connection between MCU and the download tool.

2. Erase FLASH

If the downloaded FLASH area is erased, go to Step 4. Otherwise, click “Erase sectors” in the “Common operations” area on the main window of the download tool. In the dialog box that is displayed, select the partition and enter the page address number (start page) and page number of the erased area. As shown in Figure 3-6, if the 32KB USER3 area is erased, the partition is selected as "USER3 ", the FLASH page size of N32G435RBL7 is 2KB, and the page number corresponding to the starting address 0x08018000 of USER3 partition is 0x0030 and the page number is 0x000F. The erasing address ranges from 0x08018000 to 0x0801FFFF. After you confirm that the address range is correct, click "OK" to confirm that the FLASH erasure operation is complete and the FLASH erasure is successful. Close the "Erase FLASH" dialog box and return to the main screen of the download tool.

Figure 3-6 Erase FLASH interface



The image shows a software dialog box titled "Erase FLASH". It contains the following fields and controls:

- Partition:** A dropdown menu currently showing "USER3".
- Erasable address range:** A text box showing "0x08018000-0x0801FFFF" with "(32 KB)(Min unit 16 KB)" to its right.
- Page address start number :0x** A text box showing "0030".
- Number of pages:0x** A text box showing "000F".
- Erases the address range:** A text box showing "0x08018000-0x0801F7FF".
- Note:** A text block stating: "Note: each page is 2048 byte, the page address number is the start page of erasing, and the number of pages is the number of erased pages".
- Buttons:** "OK" and "Cancel" buttons at the bottom.

3. Download the program

Click the "Partition download configuration" option in the "Common Operations" drop-down menu, which are USER1, USER2, and USER3, check the partition download enable, select the file path (the path where the program BIN is located), and enter the starting address (the default is each partition), after checking that it is correct, click "OK" to automatically download and return to the main interface of the download tool. (See Figure 3-8).

Figure 3-7 Partition download interface



Partition download

USER1

Starting address: 08000000 ☒ Partition download enable

Download file: IMU example\MMU\user1\MDK-ARM\Objects\user1.bin

☐ Partition authentication enablement

☐ Encrypt download enablement ☐ Download file is ciphertext

USER2

Starting address: 0800C000 ☒ Partition download enable

Download file: IMU example\MMU\user2\MDK-ARM\Objects\user2.bin

☐ Partition authentication enablement

☐ Encrypt download enablement ☐ Download file is ciphertext

USER3

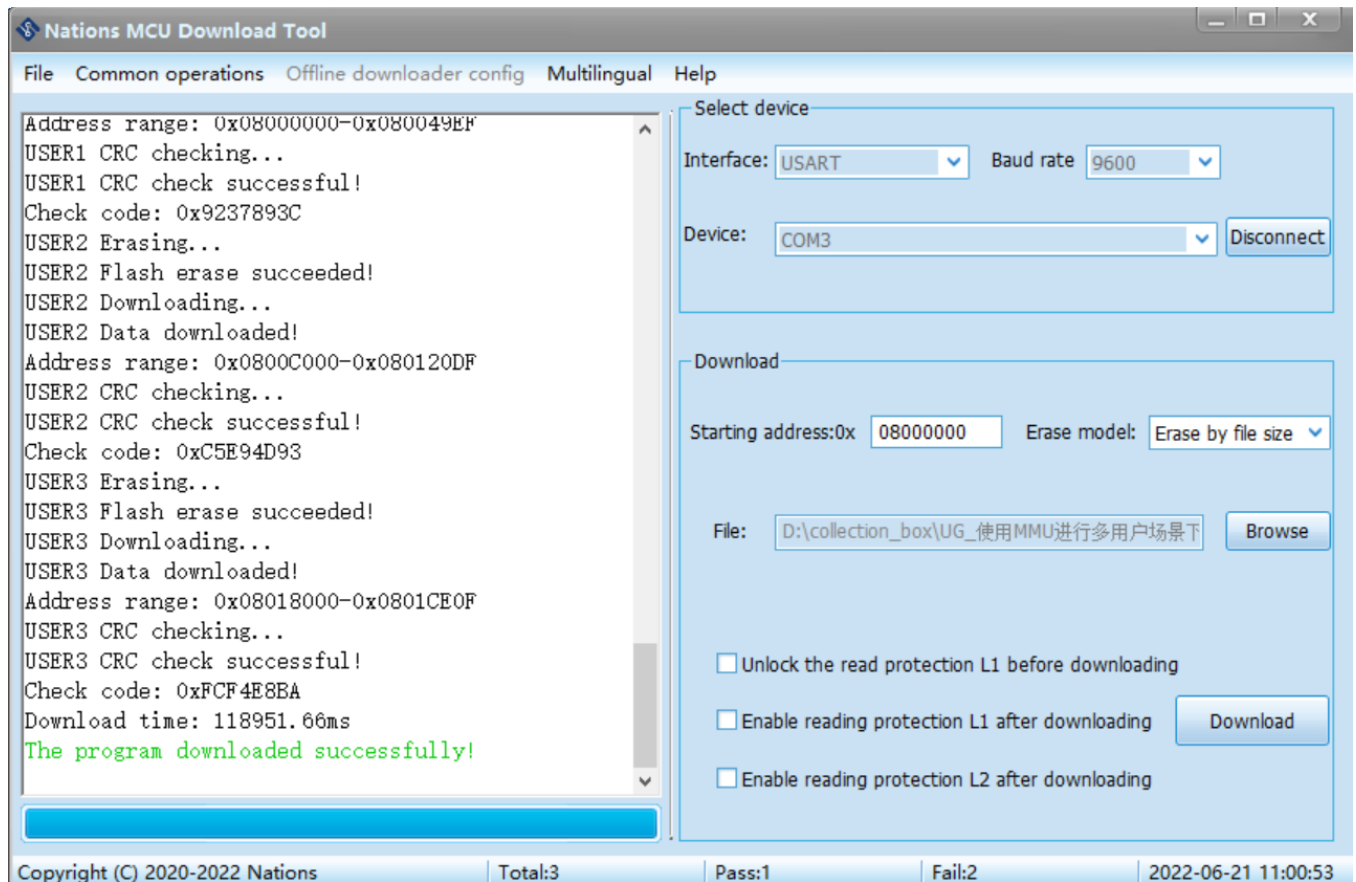
Starting address: 08018000 ☒ Partition download enable

Download file: IMU example\MMU\user3\MDK-ARM\Objects\user3.bin

☐ Partition authentication enablement

☐ Encrypt download enablement ☐ Download file is ciphertext

Figure 3-8 Download interface



Note:

- (1) The sequence between "partition setup" and "Bootloader downloading program" is not mandatory. To ensure the security of the program, it is recommended to set the partition first, and then update the program;
- (2) If no partition is set and the program is downloaded using the Bootloader, there is no need to operate "Partition download". In the "Download" area, select the file path (the path where the program BIN is located), enter the starting address, configure the erasing mode, and click "Download", wait for the download to complete.
- (3) The start address of download must match the start location of FLASH set by the program (refer to section 4.1.1 "SCT Scattered Loading Files"), otherwise the program may run abnormally.

4 The sample project

In order to show the execution mode of the program after the FLASH main storage area is partitioned, such as the function calling method between different partition areas, the different effects of reading data normally or abnormally, and the interrupt processing method, three example projects will be provided as USER1, USER2 and USER3 three partition user ID projects (path: Nationstech.N32G43x_Library.x.x.x\projects\n32g43x_EVAL\examples\MMU).

The following subsections will focus on the section address configuration of the project, the generation of bin files, and the access operations between user partitions.

4.1 Section address configuration

Take the N42G435RBL7 chip as an example. Assume that the USER1, USER2, and USER3 user areas are 64KB, 32KB, and 32KB respectively. In this case, the partition relationship of the FLASH main storage area is shown in Figure 4-1. Each user can negotiate and divide the FLASH main storage area according to the actual code amount of the application in each partition.

Figure 4-1 Flash main storage area partition relationship



In addition to partition the FLASH main storage area, to avoid global variable storage space conflicts among different partitioning programs, you can also partition the 32KB SRAM space of N42G435RBL7. The SRAM of each user can store the global variables in the corresponding program, among which the global variables of USER2 and USER3 need to be manually initialized. Since the starting address of the chip program execution is 0x08000000, USER1, as an end user, is also responsible for processing stacks and interrupt responses, so the SRAM of USER1 can also be used as stack space. The address specified by the global variable needs to avoid the stack (see the project's .map file for the stack top address).

SRAM partition is optional because the MMU of the N32G435RBL7 only manages partition access to the main FLASH storage area. SRAM is actually shared by USER1, USER2, and USER3. Dividing the SRAM into multiple regions is only for the stability of program execution (preventing overlapping of global variable spaces in different partitions) and does not provide the function of "protecting data security in user SRAM". According to the actual application, the space of global variables can be allocated by users through mutual negotiation without dividing SRAM. In this sample, USER1, USER2 and USER3 share 32KB SRAM

After the user area is divided, the application programs of each user need to be downloaded to different address spaces. Therefore, the corresponding projects need to configure their respective section addresses to avoid program download failure or abnormal operation because the address space allocated by the program is inconsistent with the download address.

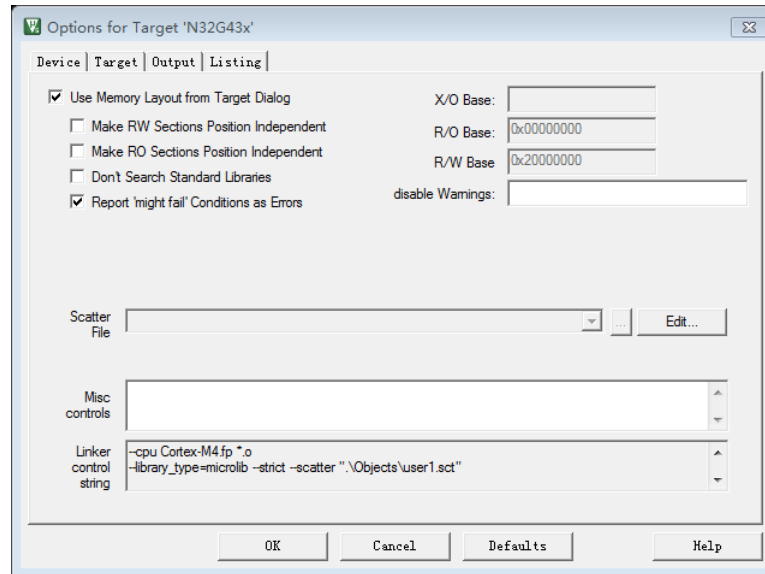
4.1.1 SCT distributed load file

The KEIL linker allocates each section address according to the configuration of the SCT distributed load file and generates the distributed load code, so the location of a section can be customized by modifying the SCT distributed load file.

- Select the SCT file generation method

SCT files can be automatically generated using MDK, or you can use user-defined SCT files. This selection can be configured through the MDK "Options for Target -> Linker->Use Memory Layout from Target Dialog" option, as shown in Figure 4-2

Figure 4-2 Choose how the SCT file is generated



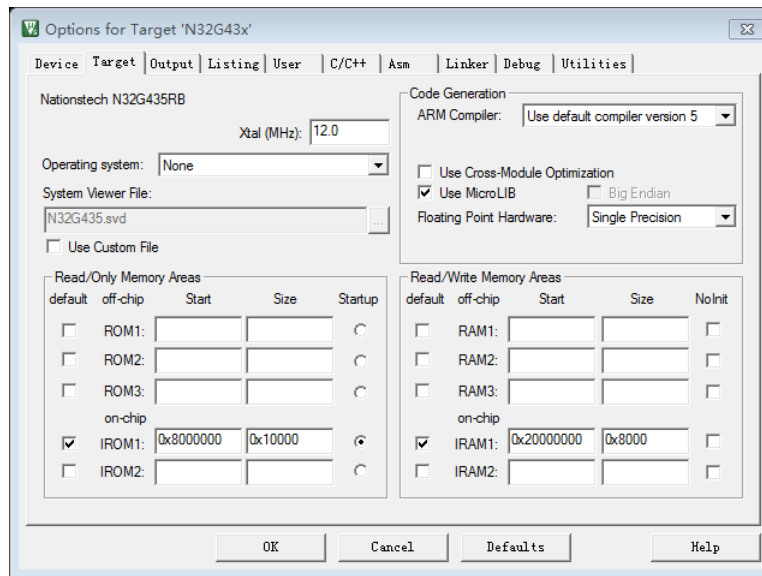
Select the "Use Memory Layout from Target Dialog" option (default for SDK) to generate the SCT file using the memory distribution configuration options of the "Options for Target -> Target" page. In this case, "Options for Target -> Linker-> Scatter File" is invalid. You cannot manually open the generated SCT File for editing. When the project construction is completed, MDK will generate a new SCT file to overwrite the old one.

If you need to manually edit the SCT file, uncheck the "Use Memory Layout from Target Dialog", specify the SCT file path in the options for "Options for Target -> Linker-> Scatter File" box. After that, clicking "Edit" will open the SCT file automatically, and users can edit the file manually.

■ Configure storage distribution through Target control

Select "Options for Target -> Linker->Use Memory Layout from Target Dialog" in MDK. On the "Options for Target -> Target" page, the memory distribution configuration takes effect automatically. The default configuration in the SDK is automatically loaded after selecting the chip model on the "Options for Target -> Device" page. After setting the FLASH partition, reset the memory configuration.

Figure 4-3 Target storage distribution configuration



In this example, USER1 is used as an example. Figure 4-4 shows the storage distribution configuration on the Options for Target -> Target page. In the on-chip part, IROM1 starts at 0x08000000 and its size is 0x10000, which is exactly the start address and size of USER1's FLASH. If IRAM1 has a start address of 0x20000000 and a size of 0x8000, they are the start address and size of USER1's SRAM region respectively. In the figure, IROM1 and IRAM1 are checked by default, indicating that the current configuration information will be used. If this parameter is unchecked, the storage configuration information will not be used.

The projects of USER2 and USER3 can reset the memory configuration in a similar way. For details, refer to the configuration of the corresponding example projects.

The path of the SCT file generated by MDK through the Target memory distribution configuration in Figure 4-3 is ".\Objects\user1.sct" (default setting of SDK), and the content of the SCT file is shown in Figure 4-4. You can manually edit the SCT file by referring to the file format.

Figure 4-4 SCT file content

```

1 ; *****
2 ; *** Scatter-Loading Description File generated by uVision ***
3 ; *****
4
5 LR_IROM1 0x08000000 0x00010000 { ; load region size_region
6 ER_IROM1 0x08000000 0x00010000 { ; load address = execution address
7   *.o (RESET, +First)
8   *(InRoot$$Sections)
9   .ANY (+RO)
10  .ANY (+XO)
11 }
12 RW_IRAM1 0x20000000 0x00008000 { ; RW data
13   .ANY (+RW +ZI)
14 }
15 }
16
17

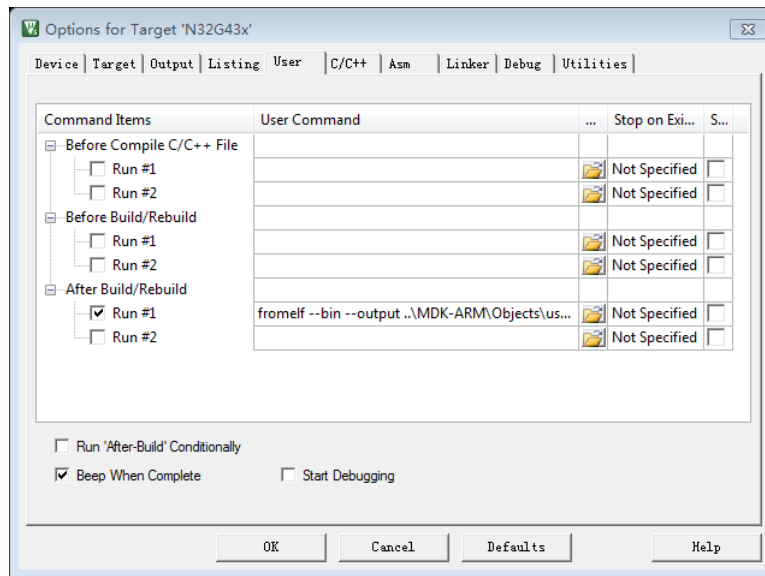
```

4. 2 Generating a bin file

To download the program through *Nations MCU Download Tool*, you need to download the bin file of the program. Here, the fromelf instruction is used to generate a bin file. Users can also write their own Python scripts and enter user instructions to execute the scripts.

On the configuration page of "Options for Target->User" of MDK, the "After Build/Rebuild" column is added to call fromelf tool to form the instruction to generate bin file (generate bin according to axf file), as shown in Figure 4-5

Figure 4-5 Interface for user configuration



The instruction to generate the bin file first calls the fromelf tool, followed by the tool's options, output file name, and input file name. If bin files and axf files are generated in the same folder "..\MDK-ARM\Objects", the user instruction of the sample project can be written as "fromelf --bin --output ..\MDK-ARM\Objects\user1.bin ..\MDK-ARM\Objects\user1.axf ". Therefore, in step 4 "Download program" of section 3.2.4.2 "Download program through the built-in bootloader ", select the bin file in this path.

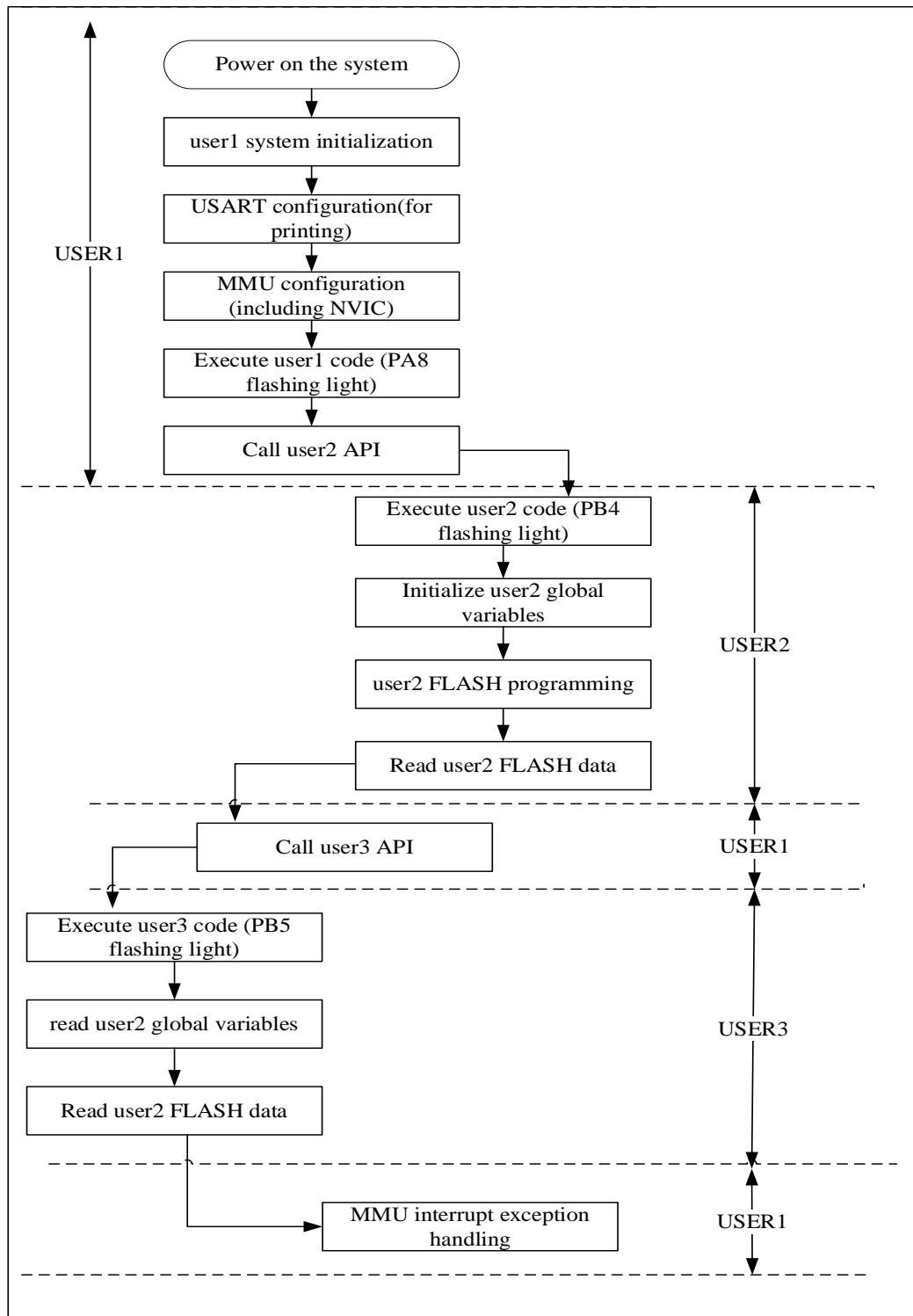
4. 3 Partition access operation

The sample projects for USER1, USER2, and USER3 work together to demonstrate mutual access between different partitions. Download the sample projects of USER1, USER2, and USER3 to the N32G435RBL7 chip respectively. After being powered on again, the chip with three partitions will execute the code according to the flow shown in Figure 4-6 (refer to the partition size configuration in section 4.1 "Section address configuration" for sample project). Chip program execution start address is 0x08000000, so USER1 as the end user is responsible for the control of the entire application process, including system initialization, stack processing, interrupt processing and other operations.

The MMU limits the read and write operations between FLASH partitions. The access between partitions is realized by invoking API. For example, USER2 and USER3 implement applications with certain functions (encapsulated in API form) respectively, and USER1 accesses the application functions of USER2 or USER3 by invoking the API. The MMU also restricts the partition user's relocation interrupt vector table operations (SCB->VTOR). Only the end user USER1 has the permission to set SCB->VTOR, and the address of the interrupt vector table must be in USER1's FLASH space. All unauthorized operations involving MMU (such as debugging interface/program reading or writing unauthorized, of reading or writing unauthorized interrupt vector table address , etc.) will trigger the MMU abnormal alarm and inform the user in time in the way of reset or interruption.

The following highlights three partition access operations: calling API across partitions, reading and writing data across partitions, and interrupt handling.

Figure 4-6 Example project execution flow



4.3.1 Call API

The cross-partition call API is essentially the execution of a program by jumping to a function at a specified location. Functions can be assigned addresses automatically by the compiler (see the project's .map file), or they can be assigned addresses by users of each partition (recommended). In API scenarios that provide multiple cross-partition access,

specifying fixed addresses for functions is clearly advantageous. The "__attribute__" keyword in MDK can specify the address.

In this case, USER1 calls the API of USER2 and USER3, respectively. This section describes how USER1 invokes USER2 API for reference.

The FLASH of USER2 ranges from 0x0801_0000~0x0801_7FFF, and the SRAM ranges from 0x2000_4000~0x2000_6000. In the user2_demo.c of sample project user2, place function "void Test_User2(void)" at address 0x08016000 (see Figure 4-7).

Figure 4-7 Specifies the function address

```

45  /**
46   * @brief FLASH program.
47   */
48  void Test_ProgramFlashWord(uint32_t Address, uint32_t Data)
49  {
93  /**
96  void Test_InitData(void)
97  {
101 void Test_User2(void) __attribute__((section(".ARM.__at_0x08016000")));
102
103  /**
104   * @brief USER2_Demo
105   */
106  void Test_User2(void)
107  {
108  {
109      uint32_t flash_write_data = 0x76543210;
110      Test_InitData();
111      /* USART Configuration */
112      // USART_Configuration();
113      /* Output a message on Hyperterminal using printf function */
114      printf("\n\rHello! Here is USER2 Example!\n\r");
115
116      /* LED2 Blinks */
117      Test_LedBlink(LED2_PORT, LED2_PIN);
118
119      /* Program USER2 FLASH */
120      Test_ProgramFlashWord(0x08017800, flash_write_data);
121
122      /* Read USER2 FLASH */
123      printf("USER2 Get USER2 FLASH *0x08017800 = 0x%X\r\n", *((__IO uint32_t*) (0x08017800)));
124
125      /* Read USER2 FLASH */
126      printf("USER2 Get USER2 SRAM *0x20004A00 = 0x%X\r\n", *((__IO uint32_t*) (0x20004A00)));
127  }

```

USER2 provides the jump address of the function to other partition users so that they can jump to this address and call API functions. To facilitate co-development by multiple users, USER2 can use macros in the user2_demo.h file to define jump addresses and jump operations for functions (see Figure 4-8). After that, different users can get information about the jump to the application through the header file.

Figure 4-8 Jump address and function pointer

```

44  typedef void (*pFunction) (void);
45
46  #define USER2_FUNC_ADDR (0x08016001)
47  #define API_FuncEntry2 ( (pFunction) (USER2_FUNC_ADDR) )

```

For USER1, you can choose to add user2_demo.h to the example project user1. After that, the program of USER1 can jump to the execution function of USER2 by calling the API "API_FuncEntry2()"; to realize operations such as PB4 flashing

Note: The nature of calling the API across partitions does not limit the functions to be jumped. But there is a special

case that needs to be pointed out, it cannot jump to the reset function. The reason is that the Reset_Handler function processing in startup_n32g43x.s (as shown in Figure 4-9) involves cross-partition operations, which will trigger an MMU exception alarm.

Figure 4-9 Reset_Handler function definition

```

170
171; Reset handler
172 Reset_Handler PROC
173     EXPORT Reset_Handler            [WEAK]
174     IMPORT __main
175     IMPORT SystemInit
176     LDR R0, =SystemInit
177     BLX R0
178     LDR R0, =__main
179     BX  R0
180     ENDP
181

```

4.3.2 Read and write data MMU abnormal alarm

After the FLASH partition configuration takes effect, cross-partition data reading, FLASH programming, SRAM code accessing the user partition, DMA1/DMA2 or debugging interface accessing the user partition will trigger the MMU exception alarm (see section 4.3.3 “Interrupt handling” for the interrupt alarm mode and handling method). The example projects for USER2 and USER3 demonstrate normal and abnormal data reading and writing, respectively.

In the user2_demo.c file of the example project user2, the example demo demonstrates that User2 reads and writes data in the owning partition area (SRAM or FLASH), as shown in Figure 4-7. Write the value of the global variable flash_write_data in USER2 SRAM to the position 0x0801_7800 specified by USER2 FLASH, and verify that the data written is correct. The above operations are routine operations, and the specific operation methods will not be described. It is important to note that the initial value of the global variable of USER2 may not be 0 because the example project of USER2 did not execute the startup process. Please initialize the global variable before using it.

USER3 can read and write USER2 SRAM. However, USER3 cannot write to USER2 FLASH or read data from USER2 FLASH due to the partition permission management function of the MMU. In USER3's example project user3, the file user3_demo.c contains the sample demo code that lines 66 in Figure 4-10 will trigger an MMU abnormal reset alarm (the default).

Figure 4-10 USER3 reading data

```

45 void Test_User3(void) __attribute__((section(".ARM.__at_0x08019000")));
46
47 /**
48  * @brief USER3_Demo
49  *
50  */
51 void Test_User3(void)
52 {
53     /* USART Configuration */
54     // USART_Configuration();
55
56     /* Output a message on Hyperterminal using printf function */
57     printf("\n\rHello! Here is USER3 Example\n\r");
58
59     /* LED3 Blinks */
60     Test_LedBlink(LED3_PORT, LED3_PIN);
61
62     /* Read USER2 SRAM */
63     printf("USER3 Get USER2 SRAM *0x20004A00 = 0x%X\r\n", *((__IO uint32_t*) (0x20004A00)));
64
65     /* Read USER2 FLASH */
66     printf("USER3 Get USER2 FLASH *0x08017800 = 0x%X\r\n", *((__IO uint32_t*) (0x08017800)));
67 }

```

4.3.3 Interrupt handling

Since USER2 and USER3 cannot relocate the interrupt direction table (SCB->VTOR), all interrupt operations involving USER2 and USER3 are handled by the end user USER1. So the three users need to collaborate to complete the interrupt handling. USER2 and USER3 need to inform USER1 of their respective interrupt handling and add it to USER1's program for processing. If the operations involved in the interrupt service functions of USER2 and USER3 need to be kept secret, it is recommended that the interrupt handling content be encapsulated as an API assigned to a fixed location in the manner described in section 4.3.1 "Call API", and that USER1 call the API to handle the corresponding interrupt.

There are two ways of MMU alarm: reset (default) or interrupt. In this example, we will demonstrate MMU interrupt handling. In the n32g43x_mmu.c file, "void MMU_Init(MMU_ALARM_MODE mode)" function is provided to configure the MMU alarm mode.

The example project user1 demonstrates how to use the MMU interrupt exception alarm. The user1_mmu_demo.c file provides the configuration method for the MMU interrupt abnormal alarm (sample code shown in Figure 4-11), and the MMU interrupt handler function is defined in the n32g43x_it.c file (see Figure 4-12). Any unauthorized operation by USER1, USER2, or USER3 triggers a call to the MMU interrupt handler function.

Figure 4-11 MMU interruption alarm configuration

```

45 void NVIC_Configuration(void)
46 {
47     NVIC_InitType NVIC_InitStructure;
48
49     /* Enable MMU IRQChannel */
50     NVIC_InitStructure.NVIC_IRQChannel      = MMU_IRQn;
51     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
52     NVIC_InitStructure.NVIC_IRQChannelSubPriority    = 0;
53     NVIC_InitStructure.NVIC_IRQChannelCmd          = ENABLE;
54     NVIC_Init(&NVIC_InitStructure);
55 }
56
57 /**
58  * @brief MMU_Config.
59  */
60 void MMU_Configuration(void)
61 {
62     /* NVIC Configuration */
63     NVIC_Configuration();
64
65     /* Configure mode of MMU alarm */
66     MMU_Init(MMU_INT_EN);
67 }

```

Figure 4-12 MMU interrupt processing example

```

136 /**
137  * @brief This function handles MMU interrupt request.
138  */
139 void MMU_IRQHandler(void)
140 {
141     NVIC->ICPR[2] |= (uint32_t)(1UL << ((uint32_t)16));
142     *((volatile unsigned long *) (MMU_BASE + 0x04)) = 0;
143     while(*((volatile unsigned long *) (MMU_BASE + 0x04)))
144     {
145     }
146
147     printf("MMU Alarms~~~\r\n");
148 }
149
150

```

5 Conclusion

The FLASH can be divided into three regions (USER1, USER2 or USER3) by using the MMU embedded in the MCU chip of Nations, and the access control function is provided for each user region. It can not only protect internal memory attacks (such as mutual access between different user areas, SRAM1/2 access, etc.), but also resist some external attacks (such as debugging interface access, DMA access, etc.).

Users can set partitions and download programs through the Bootloader. Once the partition is successfully set, the user area division and permission management functions take effect immediately. In addition, the partition configuration can only be set once and cannot be reset. The operation is irreversible. These features enable the MMU to prevent unauthorized access to the FLASH and effectively protect data and code stored in the FLASH. Thus, it plays a security role in application scenarios such as copyright protection and sensitive data protection.

6 Version history

Version	Date	Modify
V1.0	2020.11.8	New document
V1.1	2022.6.22	<ol style="list-style-type: none">1. Modify G436 to G43x2. Modify section 3.2.4.23. Modify clerical error, delete watermark, modify notice, adjust document format4. The tool has been updated, corresponding to screenshot of the updated tool

7 Notice

This document is the exclusive property of NSING TECHNOLOGIES PTE. LTD. (Hereinafter referred to as NSING). This document, and the product of NSING described herein (Hereinafter referred to as the Product) are owned by NSING under the laws and treaties of Republic of Singapore and other applicable jurisdictions worldwide. The intellectual properties of the product belong to NSING Technologies Inc. and NSING Technologies Inc. does not grant any third party any license under its patents, copyrights, trademarks, or other intellectual property rights. Names and brands of third party may be mentioned or referred thereto (if any) for identification purposes only. NSING reserves the right to make changes, corrections, enhancements, modifications, and improvements to this document at any time without notice. Please contact NSING and obtain the latest version of this document before placing orders. Although NSING has attempted to provide accurate and reliable information, NSING assumes no responsibility for the accuracy and reliability of this document. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. In no event shall NSING be liable for any direct, indirect, incidental, special, exemplary, or consequential damages arising in any way out of the use of this document or the Product. NSING 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, all types of safety devices, and other applications intended to supporter sustain life. All Insecure Usage shall be made at user's risk. User shall indemnify NSING and hold NSING harmless from and against all claims, costs, damages, and other liabilities, arising from or related to any customer's Insecure Usage Any express or implied warranty with regard to this document or the Product, including, but not limited to. The warranties of merchantability, fitness for a particular purpose and non-infringement are disclaimed to the fullest extent permitted by law. Unless otherwise explicitly permitted by NSING, anyone may not use, duplicate, modify, transcribe or otherwise distribute this document for any purposes, in whole or in part.