GigaDevice Semiconductor Inc.

Arm® Cortex®- M3/M4/M23/M33 32-bit MCU

Application Note
AN018

Table of Contents

List of Figures

List of Tables

1. Introduction

Memory Protection Unit(MPU) is an optional unit in Cortex-M3/M4 controller. It can protect memory and make software system more robust and reliable. If you want to learn more about MPU, you can read Chapter 14 of the Cortex-M3 Authoritative Guide in Chinese or Chapter 11 of the Cortex-M4 Authoritative Guide in English. This application note, based on the GD32F4xx series, describes how to set up MPU according to the application and how to switch between privileged mode and unprivileged mode.

2. Application of MPU

2.1. Development Environment

Evaluation Board：GD32450Z-EVAL

Development tool：GD32F4xx_MPU_example

Keil/IAR

2.2. Development Goal

Define an array named PrivilegedReadOnlyArray in the software, and define the region attribute that the array belongs to as read-only in privilege mode. Verify that a write operation on the region will cause memory management exception.

2.3. Code Interpretation

MPU programming can be divided into four steps：

(a) Check whether the core supports MPU through MPU type register；

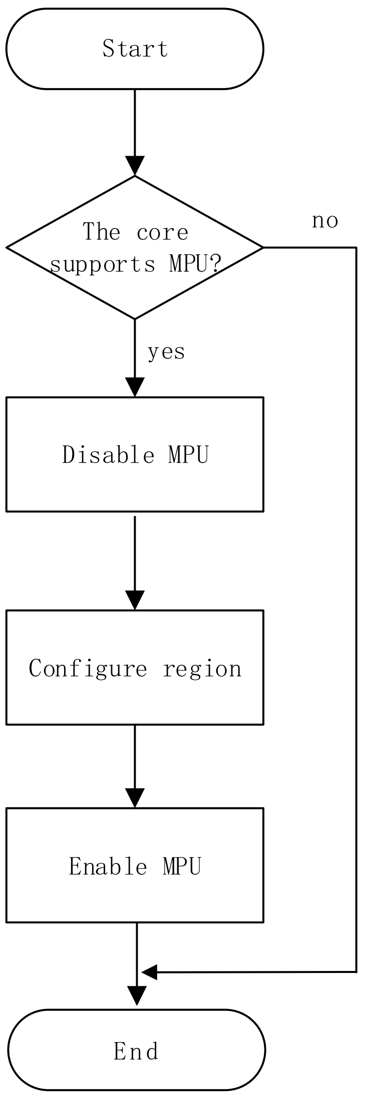
(b) Disable MPU；

(c) Configure the size and attributes of each region according to the application requirement；

(d) Enable MPU。

The flowchart is show in Figure 2.1.

Figure 2-1. Flowchart of MPU



In the main program, configure systick first, then initialize LED1 and LED2, and then determine whether the core supports MPU. If the core does not support, the program enters an infinite loop. Otherwise, configure MPU. Finally, let the LED1 toggle every second. The main program code is show in ***Table 2-1. Code of main program***.

Table 2-1. Code of main program

```
int main(void)
{
    systick_config();
    gd_eval_led_init(LED1);
    gd_eval_led_init(LED2);
    gd_eval_led_off(LED2);

    /* if the core dose not support MPU, go to infinite loop */
    if(FALSE == mpu_test()){
        while(1);
    }

    mpu_setup();
    mpu_access_permission_config();
    while(1){
        /* toggle led1 */
        gd_eval_led_toggle(LED1);
        delay_1ms(1000);
    }
}
```

The properties of RAM、FLASH and the region to which the peripheral belongs are configured in the mpu_setup() function. The properties of the region to which the arrayPrivilegedReadOnlyArray belongs are configured in the mpu_access_permission_config() function. The following is an introduction of mpu_setup() function and mpu_access_permission_config() function.

In the mpu_setup() function, the program first disables the MPU, and then sets the properties for the region in which RAM is located. Set RAM as region 0, starting address is 0x20000000, size is 8KB, readable and writable in privilege mode and unprivileged mode. It can not be shared, not cached, not buffered, and allowed to fetch. Next, set the properties of the region in which FLASH is located. Set FLASH as region 1, starting address is 0x08000000, size is 1MB, readable and writable in privilege mode and unprivileged mode. It can not be shared, not cached, not buffered, and allowed to fetch. Finally, set the properties of the region in which peripheral is located. Set peripheral as region 2, starting address is 0x40000000, size is 512MB, readable and writable in privilege mode and unprivileged mode. It can not be shared, not cached, not buffered, and not allowed to fetch. After setting up, enable MPU. The code of the mpu_setup() function is shown in ***Table 2-2. Code of mpu_setup() function***.

Table 2-2. Code of mpu_setup() function

```
void mpu_setup(void)
{
    mpu_region_init_struct mpu_init_struct;

    /* disable MPU */
    mpu_disable();

    /* configure RAM region as region 0, 8kB of size and R/W region */
    mpu_init_struct.enable                  =   MPU_REGION_ENABLE;
    mpu_init_struct.base_address             =   RAM_BASE_ADDRESS;
    mpu_init_struct.region_size             =   RAM_SIZE;
     mpu_init_struct.access_permission=MPU_REGION_PRIV_READ_WRITE_USER_READ_WRI
       TE;
    mpu_init_struct.bufferable              =   MPU_ACCESS_NOT_BUFFERABLE;
    mpu_init_struct.cacheable                    =   MPU_ACCESS_NOT_CACHEABLE;
    mpu_init_struct.shareable                    =   MPU_ACCESS_NOT_SHAREABLE;
    mpu_init_struct.number                       =   MPU_REGION_NUMBER_0;
    mpu_init_struct.type_extension_field         =   MPU_TEX_LEVEL_0;
    mpu_init_struct.sub_region_disable           =   MPU_SUB_REGION_ALL_DISABLE;
    mpu_init_struct.instruction_accessable =   MPU_INSTRUCTION_ACCESS_ENABLE;

    mpu_region_config(&mpu_init_struct);

    /* configure FLASH region as region 1, 1MB of size and R/W region */
    mpu_init_struct.base_address                 =   FLASH_BASE_ADDRESS;
    mpu_init_struct.region_size                  =   FLASH_SIZE;
    mpu_init_struct.number                       =   MPU_REGION_NUMBER_1;

    mpu_region_config(&mpu_init_struct);

    /* configure peripheral region as region 2, 512MB of size, R/W and execute never region */
    mpu_init_struct.base_address                 =   PERIPH_BASE_ADDRESS;
    mpu_init_struct.region_size                  =   PERIPH_SIZE;
    mpu_init_struct.number                       =   MPU_REGION_NUMBER_2;
    mpu_init_struct.instruction_accessable =   MPU_INSTRUCTION_ACCESS_DISABLE;

    mpu_region_config(&mpu_init_struct);

    /* enable MPU */
    mpu_enable(MPU_HFNMI_DISABLE_PRIVDEF_ENABLE);

}
```

In the mpu_access_permission_config() function, the program first disables the MPU, and then configures the properties of the region where the array PrivilegedReadOnlyArray is located. Set the array as region 3, starting address is 0x20002000, size is 32Byte, only can read in privileged mode. It can not be shared, not cached, not buffered, and allowed to fetch. After setting up, enable MPU. Because it is set to be readable only in privilege mode, after execute read_data = PrivilegedReadOnlyArray[0]; the program will not enter MemManage_Handler. However, if execute write operation PrivilegedReadOnlyArray[0] = 'a'; the program goes into MemManage_Handler and lights up the LED2. The code of the mpu_access_permission() function is shown in ***Table 2-3. Code of mpu_access_permission() function***.

Table 2-3. Code of mpu_access_permission() function

```
void mpu_access_permission_config(void)
{
    mpu_region_init_struct mpu_struct;

    /* disable MPU */
    mpu_disable();

    /* configure region for privileged read only array as region 3, 32 byte and read only in privileged
mode */
    mpu_struct.enable                      =   MPU_REGION_ENABLE;
    mpu_struct.base_address                =   ARRAY_BASE_ADDRESS;
    mpu_struct.region_size                 =   ARRAY_SIZE;
    mpu_struct.access_permission=MPU_REGION_PRIV_READ_ONLY_USER_DISABLE;
    mpu_struct.bufferable                  =   MPU_ACCESS_NOT_BUFFERABLE;
    mpu_struct.cacheable                   =   MPU_ACCESS_NOT_CACHEABLE;
    mpu_struct.shareable                   =   MPU_ACCESS_NOT_SHAREABLE;
    mpu_struct.number                      =   MPU_REGION_NUMBER_3;
    mpu_struct.type_extension_field        =   MPU_TEX_LEVEL_0;
    mpu_struct.sub_region_disable          =   MPU_SUB_REGION_ALL_DISABLE;
    mpu_struct.instruction_accessable      =   MPU_INSTRUCTION_ACCESS_ENABLE;

    mpu_region_config(&mpu_struct);

    /* enable MPU */
    mpu_enable(MPU_HFNMI_DISABLE_PRIVDEF_ENABLE);

    /* read from privileged read only array. This will not generate error */
    read_data = PrivilegedReadOnlyArray[0];

    /* uncomment the following line to write to privileged read only array. This will generate error */
//      PrivilegedReadOnlyArray[0] = 'a';
}
```

3. Privileged Mode and Unprivileged Mode Switching

3.1. Development Environment

Evaluation Board：GD32450Z-EVAL

Development tool：GD32F4xx_privilege_mode_unprivilege_mode_switch

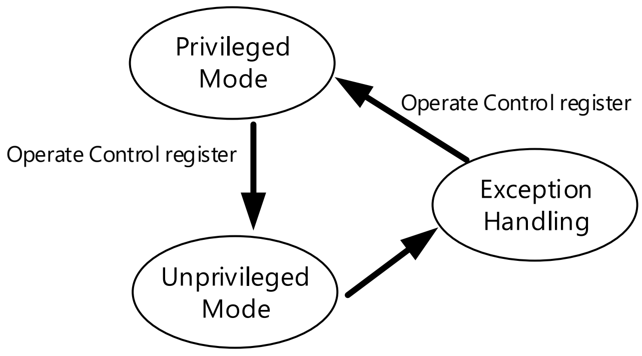
Keil

3.2. Introduction of Privileged Mode and Unprivileged Mode Switching

Privileged Mode and Unprivileged Mode are two modes supported by Cortex-M3/M4 and can be configured by bit 0 of the control register. In privileged mode, you can enter unprivileged mode by setting CONTROL[0]. Code in unprivileged mode can no longer attempt to modify CONTROL[0] to return to privileged mode, it must pass an exception handing operation, such as SVC(Supervisor Calls). Modifying CONTROL[0] in exception handling can return to privileged mode from unprivileged mode. The handover of privileged mode and unprivileged mode is shown in figure 3.1. In different modes, code has different access rights. When code runs in privileged mode, it has all access permissions. However, in unprivileged mode, its access permissions are limited.

Privileged mode and unprivileged mode are not part of MPU, but are associated with MPU. Memory access rules can be set through MPU. These rules including whether the memory space is readable and writable in the privileged mode. Therefore, setting the read-write property of the memory by MPU and operate the memory, user can understand the switch of privileged mode and unprivileged mode better.

Figure 3-1. Privilege Mode and Unprivileged Mode Switching



3.3. Development Goal

User can understand privileged mode and unprivileged mode switching better by learning this example.

3.4. Code Interpretation

In the main program, configure systick and initialize LED1 and LED2 first. Then determine whether the core supports MPU. Next, switch the stack pointer from main stack pointer to process stack pointer. And then configure the MPU. In the mpu_access_permission_config() function, the attributes of the region to which the array PrivilegedReadOnlyArray belongs are defined as readable and writable in privileged mode. Switching thread mode from privileged mode to unprivileged mode, if write to the array now, the code will trigger memory management exception and lights up LED2. It is impossible to switch back to privileged mode by modify the control register directly. It can only be done in the exception. The program enters the SVC exception and switches back to privileged mode. Write to the array will not trigger memory management exception now and the main program enter the while loop to toggle LED1 every second. The main program code is shown in table 3-1.

Table 3-1. Code of main program

```
int main(void)
{
    systick_config();
    gd_eval_led_init(LED1);
    gd_eval_led_init(LED2);
    gd_eval_led_off(LED2);

    /*if the core dose not support MPU, go to infinite loop */
    if(FALSE == mpu_test()){
        while(1);
    }
    /* switch thread mode stack from main to process */
    /* initialize memory reserved for process stack */
    for(Index = 0; Index < SP_PROCESS_SIZE; Index++)
    {
      PSPMemAlloc[Index] = 0x00;
    }

    /* set process stack value */
    __set_PSP((uint32_t)PSPMemAlloc + SP_PROCESS_SIZE);

    /* select process stack as thread mode stack */
    __set_CONTROL(SP_PROCESS);

    /* execute ISB instruction to flush pipeline as recommended by arm */
    __ISB();

    /* get the thread mode stack used */
    if((__get_CONTROL() & 0x02) == SP_MAIN){
        /* main stack is used as the current stack */
        CurrentStack = SP_MAIN;
    }else{
        /* process stack is used as the current stack */
        CurrentStack = SP_PROCESS;

        /* get process stack pointer value */
        PSPValue = __get_PSP();
    }

    mpu_setup();
    mpu_access_permission_config();
    /* switch thread mode from privileged to unprivileged */
    /* thread mode has unprivileged access */
    __set_CONTROL(THREAD_MODE_UNPRIVILEGED | SP_PROCESS);

    /* execute ISB instruction to flush pipeline as recommended by arm */
    __ISB();

    /* unprivileged access mainly affect ability to:
      - use or not use certain instructions such as MSR fields
      - access System Control Space (SCS) registers such as NVIC and SysTick */

    /* check thread mode privilege status */
    if((__get_CONTROL() & 0x01) == THREAD_MODE_PRIVILEGED){
        /* thread mode has privileged access */
        ThreadMode = THREAD_MODE_PRIVILEGED;
    }else{
        /* thread mode has unprivileged access */
        ThreadMode = THREAD_MODE_UNPRIVILEGED;
    }

    /* the thread mode is unprivileged now. It will cause MemManage fault if uncomment the following
       four lines.*/
//      if(PrivilegedReadOnlyArray[0])
//      {
//          PrivilegedReadOnlyArray[0] = 'e';
//      }

    /* switch back thread mode from unprivileged to privileged */
    /* try to switch back thread mode to privileged (Not possible, this can be
       done only in Handler mode) */
    __set_CONTROL(THREAD_MODE_PRIVILEGED | SP_PROCESS);

    /* execute ISB instruction to flush pipeline as recommended by arm */
    __ISB();

    /* generate a system call exception, and in the ISR switch back thread mode to privileged */
    __SVC();

    /* check Thread mode privilege status */
    if((__get_CONTROL() & 0x01) == THREAD_MODE_PRIVILEGED)
    {
        /* Thread mode has privileged access */
        ThreadMode = THREAD_MODE_PRIVILEGED;
    }else{
        /* Thread mode has unprivileged access */
        ThreadMode = THREAD_MODE_UNPRIVILEGED;
    }

    /* the thread mode is privileged now ,write to PrivilegedReadOnlyArray[0] will not cause
       MemManage fault */
    if(PrivilegedReadOnlyArray[0])
    {
        PrivilegedReadOnlyArray[0] = 'e';
    }

    while(1){
        /* toggle led1*/
        gd_eval_led_toggle(LED1);
        delay_1ms(1000);
    }
}
```

4. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Nov.30 2021

Important Notice