GigaDevice Semiconductor Inc.

Arm® Cortex®-Mx 32-bit MCU

Application Note
AN040

Table of Contents

List of Figures

List of Tables

1. Introduction

IAP (In application programming) program can be used for MCU APP function upgrading by a pre-writing bootloader program, increasing the flexibility of code. After upgrading the APP code, the program needs to jump to APP program from Bootloader, this application note is writing for introduction of how to realize the program jumping from Bootloader to APP, basing on GD32F10x series.

2. IAP program

IAP program is commonly included by two parts: Bootloader and APP. Bootloader and APP are two projects, placing in different area of Main Flash, which is started from 0x08000000.

2.1. Program structure

2.1.1. Bootloader

Bootloader code structure is shown as below.

Table 2-1. Bootloader code

```
/*!
    \brief      main function
    \param[in]  none
    \param[out] none
    \retval     none
*/
int main(void)
{
    /* init modules … */
   ……
    /* if no need to update APP */
    if(……){
        /* Check if valid stack address (RAM address) then jump to user application */
        if (0x20000000 == ((*(__IO uint32_t*)USER_FLASH_BANK0_FIRST_PAGE_ADDRESS) &
0x2FFE0000)){
            /* disable all interrupts */
            nvic_irq_disable(EXTI0_IRQn);
            …
            /* Jump to user application */
            JumpAddress = *(__IO uint32_t*) (USER_FLASH_BANK0_FIRST_PAGE_ADDRESS
+ 4);
            Jump_To_Application = (pFunction) JumpAddress;
            /* Initialize user application's Stack Pointer */
            __set_MSP(*(__IO uint32_t*) USER_FLASH_BANK0_FIRST_PAGE_ADDRESS);
            Jump_To_Application();
        } else {
            /* LED2 ON to indicate bad software (when not valid stack address) */
            gd_eval_led_on(LED2);
            /* do nothing */
            while(1){
```

```
            }
        }

    /* Bootloader codes for update APP areas */
    } else {
        /* Bootloader realizing codes */
        /* including commands of operating flash */
        ……
        while (1){
            /* Bootloader realizing codes */
        }
    }
}
```

2.1.2. APP

APP code structure is shown as below.

Table 2-2. APP code

```
/*!
    \brief        main function
    \param[in]    none
    \param[out]   none
    \retval       none
*/
int main(void)
{
   /* set the NVIC vector table base address to APP code area */
   nvic_vector_table_set(NVIC_VECTTAB_FLASH, APP_OFFSET);
   /* enable global interrupt, the same as __set_PRIMASK(0) */
   __enable_irq();
   /* init modules … */
   ……

   while (1){
       /* APP realizing codes */
   }
}
```

2.2. Project configuration

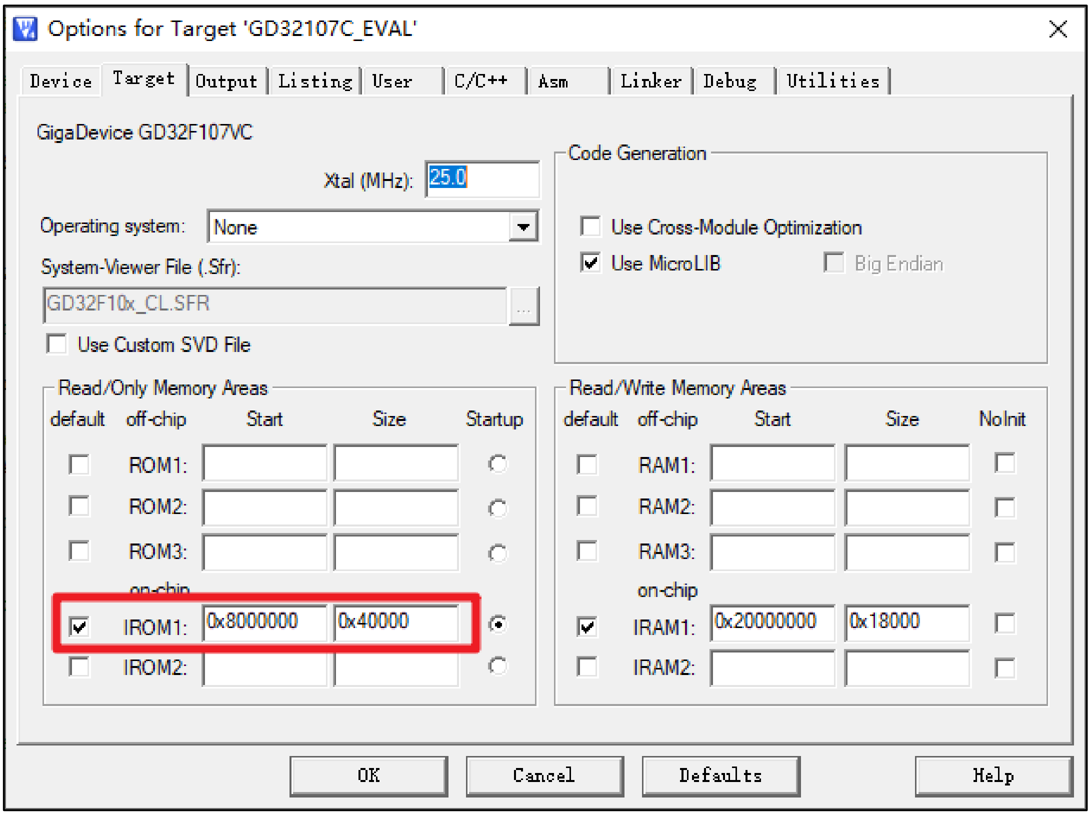
To upgrade APP code, the Bootloader code should be downloaded to Flash area started from 0x08000000 in advance, and the APP code must not overlap the Bootloader code. Follow the steps shown as below, take GD32F107VC as an example.

2.2.1. Bootloader

To ensure the APP code not overlap the Bootloader code, the Bootloader project should be configured in the following steps:

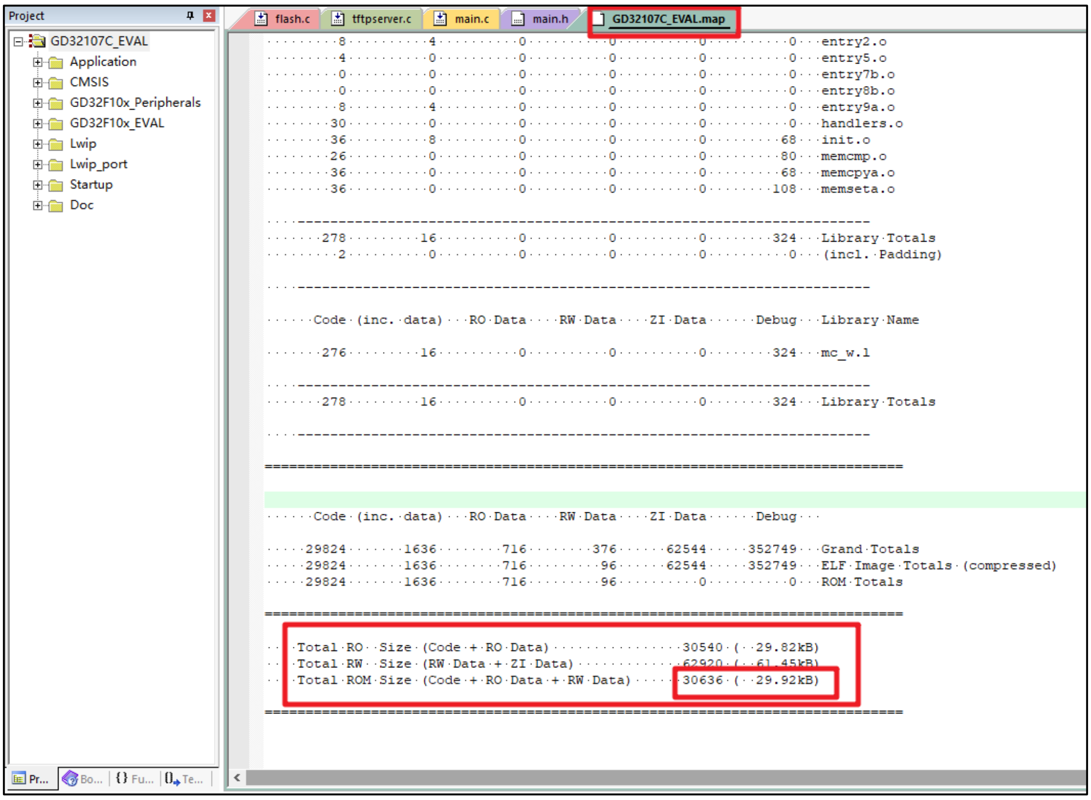
1. First, check the datasheet, flash size is 256KB, so the project configuration is shown as below, confirm it is started from 0x08000000.

Figure 2-1. Bootloader project configuration



2. Check the map file which is produced by Bootloader building, confirm the code size, and we get 29.92KB, it is 0x77AE equally.

Figure 2-2. Bootloader project map file



3. Modify the related commands of writing or erasing APP Flash area in Bootloader code, modification is mainly about changing the start address of Flash area which is to be written or erased: the address changes to 0x08010000, which is defined by the macro USER_FLASH_BANK0_FIRST_PAGE_ADDRESS, it means there are 0x10000 bytes used for Bootloader code storage, and the size is larger than the Bootloader size 0x77AE bytes.

Figure 2-3. Commands of writing or erasing APP Flash in Bootloader project

```c
static int IAP_tftp_process_write(struct udp_pcb *upcb, const ip_addr_t *to, int to_port)
{
    tftp_connection_args *args = NULL;
    /* This function is called from a callback,
    * therefore interrupts are disabled,
    * therefore we can use regular malloc   */
    args = mem_malloc(sizeof *args);
    if (!args){
        IAP_tftp_cleanup_wr(upcb, args);
        return 0;
    }

    args->op = TFTP_WRQ;
    args->to_ip.addr = to->addr;
    args->to_port = to_port;
    /* the block # used as a positive response to a WRQ is _always_ 0!!! (see RFC1350)  */
    args->block = 0;
    args->tot_bytes = 0;

    /* set callback for receives on this UDP PCB (Protocol Control Block) */
    udp_recv(upcb, IAP_wrq_recv_callback, args);
  
    total_count =0;

    /* init flash */
    FLASH_If_Init();
  
    /* erase user flash area */
    FLASH_If_Erase(USER_FLASH_BANK0_FIRST_PAGE_ADDRESS);

    Flash_Write_Address = USER_FLASH_BANK0_FIRST_PAGE_ADDRESS;
    /* initiate the write transaction by sending the first ack */
    IAP_tftp_send_ack_packet(upcb, to, to_port, args->block);
    return 0;
}
```

Figure 2-4. Macro of APP program address in Bootloader project

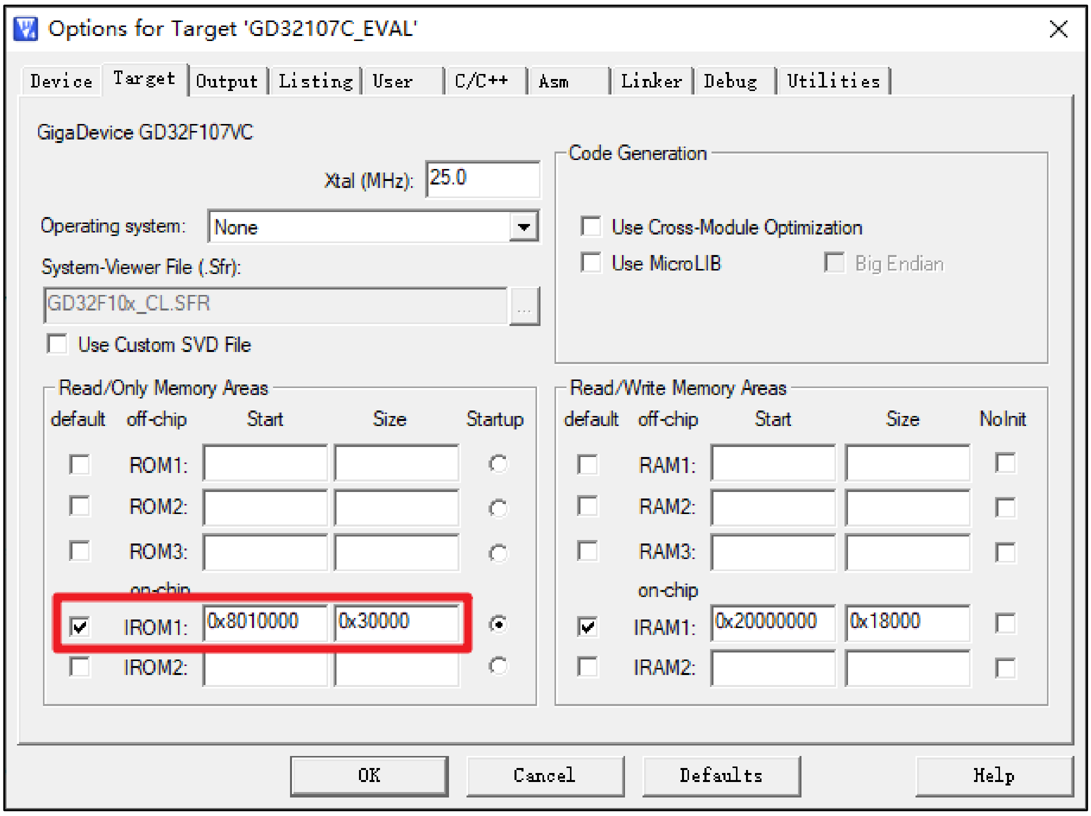
flash.c | tftpserver.c | main.c | main.h | GD32107C_EVAL.map

```c
*/

#ifndef __MAIN_H
#define __MAIN_H

#include "gd32f10x.h"
#include "stdint.h"
#include "gd32f10x_enet_eval.h"

//#define USE_DHCP       /* enable DHCP, if disabled static address is used */

//#define USE_ENET_INTERRUPT
//#define TIMEOUT_CHECK_USE_LWIP

#define USER_FLASH_BANK0_FIRST_PAGE_ADDRESS 0x08010000       //user Flash address start from
#define USER_FLASH_BANK0_LAST_PAGE_ADDRESS  0x0807F800       //user Flash address start from

#define USER_FLASH_BANK1_LAST_PAGE_ADDRESS  0x082FF000       //user Flash address start from
#define USER_FLASH_END_ADDRESS              0x082FFFFF
```

2.2.2. APP

APP project configuration is shown as below. Project code start address is set to 0x08010000, the same as the address which is to be written or erased in Bootloader code.

Figure 2-5. APP project configuration



2.3. Code explaination

There is a special code in Bootloader code and APP code, which is about how to jump from Bootloader to APP, and the code gives in chapter 2.1.1 is applied for Arm Cortex-M core jumping situation, detailed explaination is shown as below.

- if (0x20000000== ((*(__IO uint32_t*)USER_FLASH_BANK0_FIRST_PAGE_ADDRESS) & 0x2FFE0000))

  Here the macro USER_FLASH_BANK0_FIRST_PAGE_ADDRESS is stored the APP program start address, while the APP program start address is stored the stack pointer (check the address before the vectors table in startup.s file). If the APP program is already downloaded, then the APP program start address is surely stored with the stack pointer, thus we can get the downloading situation of APP program by checking whether the stack pointer is located in SRAM area. SRAM size can be obtained by checking the MCU datasheet, in this case it is 96K, that is 0x18000 bytes, so we can check whether SP is among 0x20000000~0x20017FFF, to reach this goal, we can check bit 17-31 of SP, by logically 'and' with 0x2FFE0000, but this is not precisely correct, we can also use direct address comparing method. After judging, if the result is APP program already downloaded, the jumping action will be done continuously.

- nvic_irq_disable(EXTI0_IRQn);

Before jumping to APP program, it is necessary to disable all interrupts to avoid APP program running error or stuck at. One reason is that when running __main function of Reset_Handler function, the RAM area data will be initialized, if some interrupts are not disabled and just at this time, an interrupt occurs and changes some data of RAM, because the interrupt is the Bootloader program interrupt, the data changement is not predicable and may lead to an running error of APP program. Another reason is that after jumping to APP program, due to no reset action of all modules, all module registers maintain configurations as before in Bootloader except the clock parameters which is reconfigured, if there are some interrupts are not disabled, while the corresponding module is still running as before in Bootloader program, and automatically triggers an interrupt, and there are no corresponding interrupt service routine in APP program for flag clearing, then the APP program may stuck at interrupt and running abnormally. So it is necessary to disable all interrupts.

- JumpAddress = *(__IO uint32_t*) (USER_FLASH_BANK0_FIRST_PAGE_ADDRESS + 4);

  Jump_To_Application = (pFunction) JumpAddress;

  The address of USER_FLASH_BANK0_FIRST_PAGE_ADDRESS + 4 stores the Reset_Handler vector, which is the entry address of Reset_Handler function. Because pFunction is already self-defined as a void typed function pointer, the next command is to make the Jump_To_Application pointer point to the entry address of Reset_Handler function.

- __set_MSP(*(__IO uint32_t*) USER_FLASH_BANK0_FIRST_PAGE_ADDRESS);

  Execute the first command of APP program, that is to set the main stack pointer point to the APP program start address USER_FLASH_BANK0_FIRST_PAGE_ADDRESS, prepare MSP to get ready for running APP program, for case of NMI or other fault before running the first command of Reset_Handler function, at this time, MSP is needed to provide stack.

- Jump_To_Application();

  Execute the Reset_Handler function which the Jump_To_Application pointer point at, after executing __main function in Reset_Handler function, the program will jump to main() function, the APP main program part.

3. Revision history

Table 3-1. Revision history

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

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the “Product”), is owned by the Company under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. 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. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.

© 2021 GigaDevice – All rights reserved