GigaDevice Semiconductor Inc.

Arm® Cortex®- M3/M4/M33 32-bit MCU

应用笔记

AN027

目录

图索引

表索引

1. 简介

ISP（在系统可编程）协议是可编程设备可以编程写入用户代码，而不需要从电路板上取下来，已经编程的器件也可以用 ISP 方式擦除或再编程。常用的 ISP 协议是基于 USART、USB、I2C 等。本文介绍的方法基于串口 ISP 协议。

由于使用 ISP 上位机升级程序需要借助电脑，在实际生产过程中会带来不便。因此实现了一种嵌入式 ISP 主机用于对目标 MCU 的程序进行升级。嵌入式主机根据 ISP 协议通过串口与 BootLoader 进行通信，从而实现对目标 MCU 程序的升级。

注意：具体协议可参考 ISP 串口协议说明。

2. 硬件连接

将 GD32F303ZET6 作为嵌入式主机，GD32F407VGT6 为目标 MCU，本文介绍嵌入式主机如何通过串口 ISP 对目标 MCU 进行程序升级。

GD32F4xx 系列芯片提供了三种引导源，可以通过 BOOT0 和 BOOT1 引脚来进行选择，详细说明见*表 2-1. 引导模式*。该两个引脚的电平状态会在复位后的第四个 CK_SYS（系统时钟）的上升沿进行锁存。用户可自行选择所需要的引导源，通过设置上电复位和系统复位后的 BOOT0 和 BOOT1 的引脚电平。一旦这两个引脚电平被采样，它们可以被释放并用于其他用途。

表 2-1. 引导模式

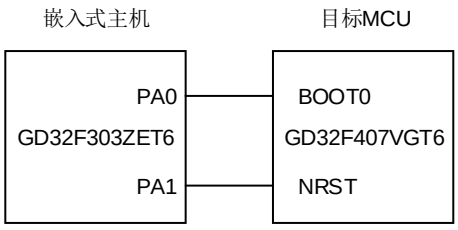
引导源选择	启动模式选择引脚	
	BOOT1	BOOT0
主 FLASH 存储器	x	0
系统存储器	0	1
片上 SRAM	1	1

将嵌入式主机 BOOT1 接地，并选两个 GPIO 引脚分别用于控制目标板 BOOT0 及 NRST 引脚电平。BOOT0 及 NRST 引脚电平不同时功能如*表 2-2. BOOT0 及 NRST 引脚功能*所示。例如 PA0 控制 BOOT0，PA1 控制 NRST，如*图 2-1. 硬件连接*所示。

表 2-2. BOOT0 及 NRST 引脚功能

功能	引脚电平	说明
BOOT0	高	从 BootLoader 启动
	低	从 flash 启动
NRST	高	无影响
	低	复位

图 2-1. 硬件连接



3. 串口升级程序

3.1. 串口初始化

串口初始化如*表 3-1. 串口初始化*所示。

表 3-1. 串口初始化

```
/*!
    \brief        configure COM port
    \param[in]    com: COM on the board
      \arg          EVAL_COM1: COM1 on the board
      \arg          EVAL_COM2: COM2 on the board
    \param[out] none
    \retval       none
*/
void gd_eval_com_init(uint32_t com)
{
    uint32_t com_id = 0U;
    if(EVAL_COM1 == com){
        com_id = 0U;
    }else if(EVAL_COM2 == com){
        com_id = 1U;
    }
    /* enable GPIO clock */
    rcu_periph_clock_enable(COM_GPIO_CLK[com_id]);
    /* enable USART clock */
    rcu_periph_clock_enable(COM_CLK[com_id]);
    /* connect port to USARTx_Tx */
    gpio_init(COM_GPIO_PORT[com_id],    GPIO_MODE_AF_PP,    GPIO_OSPEED_50MHZ,
COM_TX_PIN[com_id]);
    /* connect port to USARTx_Rx */
    gpio_init(COM_GPIO_PORT[com_id], GPIO_MODE_IN_FLOATING, GPIO_OSPEED_50MHZ,
COM_RX_PIN[com_id]);
    /* USART configure */
    usart_deinit(com);
    usart_parity_config(com, USART_PM_EVEN);
    usart_word_length_set(com, USART_WL_9BIT);
    usart_baudrate_set(com, 57600U);
    usart_receive_config(com, USART_RECEIVE_ENABLE);
    usart_transmit_config(com, USART_TRANSMIT_ENABLE);
    usart_enable(com);
}
```

```
/* USART interrupt configuration */
nvic_irq_enable(USART0_IRQn, 0, 0);
/* enable the USART RBNE interrupt */
usart_interrupt_enable(USART0, USART_INT_RBNE);
```

3.2. 控制引脚配置

将 GD32F303ZET6 开发板的 PA0 和 PA1 分别连接 GD32F407VGT6 开发板的 BOOT0 和 NRST 引脚。

PA0 和 PA1 引脚配置如*表 3-2. PA0 和 PA1 引脚配置*所示。

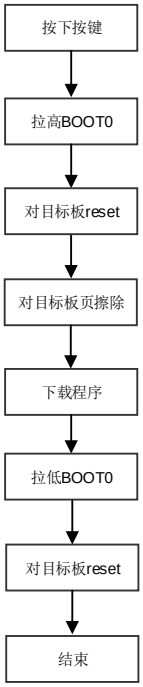
表 3-2. PA0 和 PA1 引脚配置

```
/* enable the clock */
rcu_periph_clock_enable(RCU_GPIOA);
/* configure BOOT0_CTL port */
gpio_init(GPIOA, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_0);
gpio_bit_reset(GPIOA, GPIO_PIN_0);
/* configure NRST_CTL port */
gpio_init(GPIOA, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_1);
gpio_bit_set(GPIOA, GPIO_PIN_1);
```

3.3. 升级程序

待下载 bin2 在下载之前存储到嵌入式主机 0x08004000 地址处，格式为 bin2=bin 文件大小（4 字节）+bin1 文件。可根据需要将 bin 文件存储在其他位置，bin 格式也可以自由设置。嵌入式主机对目标 MCU 进行程序升级流程图如*图 3-1. 主程序流程图*所示。

图 3-1. 主程序流程图



具体操作步骤如下：

1、 将目标 MCU 的 BOOT0 拉高。

表 3-3. BOOT0 拉高

```
void boot0_high(void)
{
    gpio_bit_set(GPIOA, GPIO_PIN_0);
    /* pull up the target reset pin */
    gpio_bit_reset(GPIOA, GPIO_PIN_1);
    gpio_bit_set(GPIOA, GPIO_PIN_1);
    /* wait for the target to reset */
    delay_1ms(300);
}
```

2、 发送握手命令，当嵌入式主机收到 0x79，说明握手成功。

表 3-4. 握手命令

```
ErrStatus usart_handshake(void)
{
    uint16_t count = 0;
    while(count++<10){
        usart_data_transmit(USART0, 0x7F);
        while(RESET == usart_flag_get(USART0, USART_FLAG_TC));
        while((ERROR == ack_flag) && (timeout--)){
        }
        if(0 != timeout){
```

```
                ack_flag = ERROR;
                return SUCCESS;
            }
        }
        return ERROR;
}
```

3、 发送页擦除命令。

首先根据 GD32F4xx flash 结构以及 bin 文件的大小索引需要擦除的扇区。

表 3-5. 获取需要擦除的页数

```
static uint32_t f4_get_sector_number(uint32_t bin_size)
{
    uint32_t sector_size[32] = {16, 16, 16, 16, 64, 128, 128, 128, 128, 128, 128, 128,
                                16, 16, 16, 16, 64, 128, 128, 128, 128, 128, 128, 128,
                                256, 256, 256, 256, 256, 256, 256};
    uint32_t i,j;
    uint32_t sector_number = 0;
    uint32_t sector_address;
    sector_address = 0x08000000 + bin_size;
    for(i=0; i<31; i++){
        uint32_t start_address = 0x08000000;
        uint32_t end_address = 0x08000000;
        for(j=0; j<=i; j++){
            start_address += sector_size[j] * 1024;
            end_address += sector_size[j] * 1024;
        }
        start_address -= sector_size[i] * 1024;
        end_address -= 1;
        if(sector_address>=start_address && sector_address<=end_address){
            sector_number = i;
            break;
        }
    }
    return sector_number;
}
```

执行页擦除命令。

表 3-6. 擦除命令

```
ErrStatus usart_send_erase_command(void)
{
    uint32_t bin_size;
    uint16_t sector_number = 0;
    uint32_t i, temp;
```

```
    /* extend erase command */
    buffer_cmd[0] = 0x44;
    buffer_cmd[1] = 0xFF - buffer_cmd[0];
    /* get the bin size */
    bin_size = *(uint32_t *)(0x8004000);
    bin_size = bin_size;
    sector_number = f4_get_sector_number(bin_size);

    usart_buffer_send(buffer_cmd, 2);
    if(SUCCESS != wait_ack()){
        return ERROR;
    }

    buffer_cmd[0] = (sector_number>>8) & 0xff;
    buffer_cmd[1] = sector_number & 0xff;

    for(i=0; i<sector_number+1; i++){
        temp = 2+2*i;
        buffer_cmd[temp] = (i>>8) & 0xff;
        buffer_cmd[temp+1] = i & 0xff;
    }

    buffer_cmd[temp+2] = data_xor(buffer_cmd, temp+2);
    usart_buffer_send(buffer_cmd, temp+3);

    if(SUCCESS != wait_ack()){
        return ERROR;
    }
    return SUCCESS;
}
```

4、 发送编程命令。

bin 文件小于等于 252 字节：

表 3-7. 编程命令（小于等于 252 字节）

```
/*!
    \brief        send the program command
    \param[in]    pro_addr: the address of the target to be programmed
    \param[in]    bin_addr: the address where stored the bin file
    \param[in]    pro_size: the size to be programmed
    \param[out] none
    \retval       ErrStatus
*/
ErrStatus usart_send_program_command(uint32_t pro_addr, uint32_t bin_addr, uint32_t pro_size)
```

```
{
    uint8_t i;

    buffer_cmd[0] = USART_CMD_PROGRAM;
    buffer_cmd[1] = 0xFF - buffer_cmd[0];

    usart_buffer_send(buffer_cmd, 2);

    if(SUCCESS != wait_ack()){
        return ERROR;
    }
    buffer_cmd[0] = (uint8_t)((pro_addr >> 24) & 0xff);
    buffer_cmd[1] = (uint8_t)((pro_addr >> 16) & 0xff);
    buffer_cmd[2] = (uint8_t)((pro_addr >> 8) & 0xff);
    buffer_cmd[3] = (uint8_t)(pro_addr & 0xff);
    buffer_cmd[4] = data_xor(buffer_cmd, 4);

    usart_buffer_send(buffer_cmd, 5);
    if(SUCCESS != wait_ack()){
        return ERROR;
    }
    memset(buffer_cmd, 0, sizeof(buffer_cmd));
    buffer_cmd[0] = pro_size-1;
    for(i=0; i<pro_size; i=i+4){
        *((uint32_t *)&buffer_cmd[1+i]) = *(uint32_t *)(bin_addr+4+i);
    }
    buffer_cmd[pro_size + 1] = data_xor(buffer_cmd, pro_size + 1);

    usart_buffer_send(buffer_cmd, pro_size + 2);
    if(SUCCESS != wait_ack()){
        return ERROR;
    }
    memset(buffer_cmd, 0, sizeof(buffer_cmd));
    return SUCCESS;
}
```

由于 BootLoader 存储空间有限，当 bin 文件大于 252 字节时，需要对 bin 文件进行拆包。

表 3-8. 编程命令（大于 252 字节）

```
/*!
    \brief          send the program command
    \param[in]    pro_addr: the address of the target to be programmed
    \param[in]    bin_addr: the address where stored the bin file
    \param[out] none
```

```
    \retval       ErrStatus
*/
ErrStatus usart_program_bin(uint32_t pro_addr, uint32_t bin_addr)
{
    uint32_t bin_size;
    uint32_t i;
    uint32_t pack_num, pack_size, res_size;
    /* 4 bytes aligned */
    pack_size = 252;

    bin_size = (*(uint32_t *)(bin_addr));
    pack_num = bin_size/pack_size;
    res_size = bin_size % pack_size;
    /* Unpacking the bin and then program to the flash of the target MCU */
    for(i=0; i<pack_num; i++){
        usart_send_program_command(pro_addr + i*pack_size, bin_addr + i*pack_size,
pack_size);
    }
    if(0 != res_size){
        usart_send_program_command(pro_addr + pack_num*pack_size, bin_addr +
pack_num*pack_size, res_size);
    }
    return SUCCESS;
}
```

5、 将目标 MCU 的 BOOT0 拉低，并对其进行复位。

表 3-9. BOOT0 拉低

```
void boot0_low(void)
{
    gpio_bit_reset(GPIOA, GPIO_PIN_0);
    /* pull down the target reset pin */
    gpio_bit_reset(GPIOA, GPIO_PIN_1);
    gpio_bit_set(GPIOA, GPIO_PIN_1);
    /* wait for the target to reset */
    delay_1ms(300);
}
```

至此，GD32F407VGT6 中程序已经可以正常运行。

4. 版本历史

表 4-1. 版本历史

版本号.	说明	日期
1.0	首次发布	2021 年 12 月 14 日

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.