GigaDevice Semiconductor Inc.

Arm® Cortex®- M3/M4/M33 32-bit MCU

Application Note
AN027

Table of Contents

List of Figures

List of Tables

1. Introduction

The ISP (In-system Programming) protocol allows programmable devices to be programmed with user code without removing it from the circuit board, and programmed devices can be erased or reprogrammed by ISP. Commonly used ISP protocols are based on USART, USB, I2C, etc. The method introduced in this paper is based on the USART ISP protocol.

Using the ISP host to update program requires the help of a computer. It is inconvenient in actual production process. An embedded ISP host is implemented to update the program of target MCU. Embedded host communicates with BootLoader through USART according to the ISP protocol, so as to update the program of the target MCU.

Note: For details, refer to the ISP serial port protocol description.

2. Hardware connection

Use the GD32F303ZET6 as the embedded host and GD32F407VGT6 as the target MCU, this app note introduces how to update program of target MCU by serial port ISP.

The GD32F4xx devices provide three kinds of boot sources which can be selected by the BOOT0 and BOOT1 pins. The details are shown in ***Table 2-1. Boot modes***. The value on the two pins is latched on the 4th rising edge of CK_SYS after a reset. It is up to the user to set the BOOT0 and BOOT1 pins after a power-on reset or a system reset to select the required boot source. Once the two pins have been sampled, they are free and can be used for other purposes.

Table 2-1. Boot modes

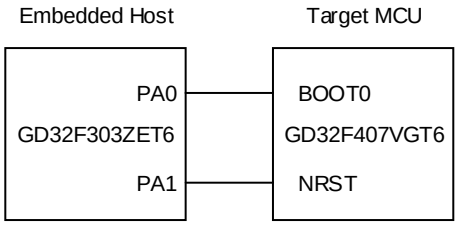
Selected boot source	Boot mode selection pins	
	Boot1	Boot0
Main Flash Memory	x	0
Boot loader	0	1
On-chip SRAM	1	1

Connect the BOOT1 of embedded host with GND, two GPIO pins are selected to control the level of BOOT0 and NRST pin on the target board respectively. The function of BOOT0 and NRST with different levels are shown in ***Table 2-2. Pin function of BOOT0 and NRST***. For example, PA0 controls BOOT0 and PA1 controls NRST, as shown in ***Figure 2-1. Hardware connection***.

Table 2-2. Pin function of BOOT0 and NRST

Function	Pin level	Description
BOOT0	High	Boot from BootLoader
	Low	Boot from flash
NRST	High	No effect
	Low	Reset

Figure 2-1. Hardware connection



3. Update program through USART

3.1. Initialization of USART

The initialization of USART is shown in ***Table 3-1. Initialization of USART***.

Table 3-1. Initialization of USART

```
/*!
    \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. Configuration of control pin

Connect the PA0 pin and PA1 pin on GD32F303ZET6 with BOOT0 pin and NRST pin on GD32F407VGT6 respectively.

The configuration of PA0 and PA1 are shown in ***Table 3-2. Configuration of PA0 and PA1***.

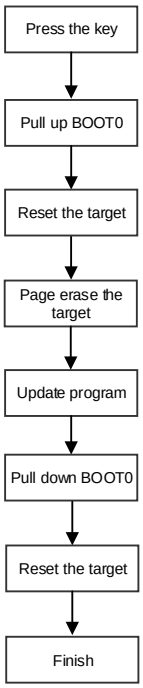
Table 3-2. Configuration of PA0 and PA1

```
/* 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. Update program

bin2 to be downloaded is stored at the address 0x08004000 of the embedded host before downloading. The format of bin2 is bin size (4 bytes) followed by bin1. The bin file can be stored in other locations as required, and the bin format can be set freely. The flow chart of updating program of target MCU by embedded host is shown in ***Figure 3-1. Main program flow chart***.

Figure 3-1. Main program flow chart



The specific operation steps are as follows:

1. Pull up the BOOT0 of the target MCU.

Table 3-3. Pull up BOOT0

```
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. Send the handshake command, if the embedded host receives 0x79, the handshake is successful.

Table 3-4. Handshake command

```
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. Send page erase command.

First index the sectors to be erased based on GD32F4xx flash structure and bin file size.

Table 3-5. Get the number of pages to be erased

```
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;
}
```

Execute the page erase command.

Table 3-6. Erase command

```
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. Send program command.

The bin file is smaller than or equal to 252 bytes:

Table 3-7. Program command (smaller than or equal to 252 bytes)

```
/*!
    \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;
}
```

The storage of BootLoader is limited. So if the bin file is larger than 252 bytes, the bin file must be unpacked.

Table 3-8. Program command (greater than 252 bytes)

```
/*!
    \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. Pull down the BOOT0 of the target MCU and reset it.

Table 3-9. Pull down the BOOT0

```
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);
}
```

So far, the program in GD32F407VGT6 can run normally.

4. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Dec.14, 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.