GigaDevice Semiconductor Inc.

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

Application Note
AN023

Table of Contents

List of Figures

List of Tables

1. Introduction

Letter shell is an embedded shell written in C language that can be embedded in the program. It is mainly for embedded devices with C language functions as the operating unit, and meanwhile, it can be called through the command line to run the functions in the program.

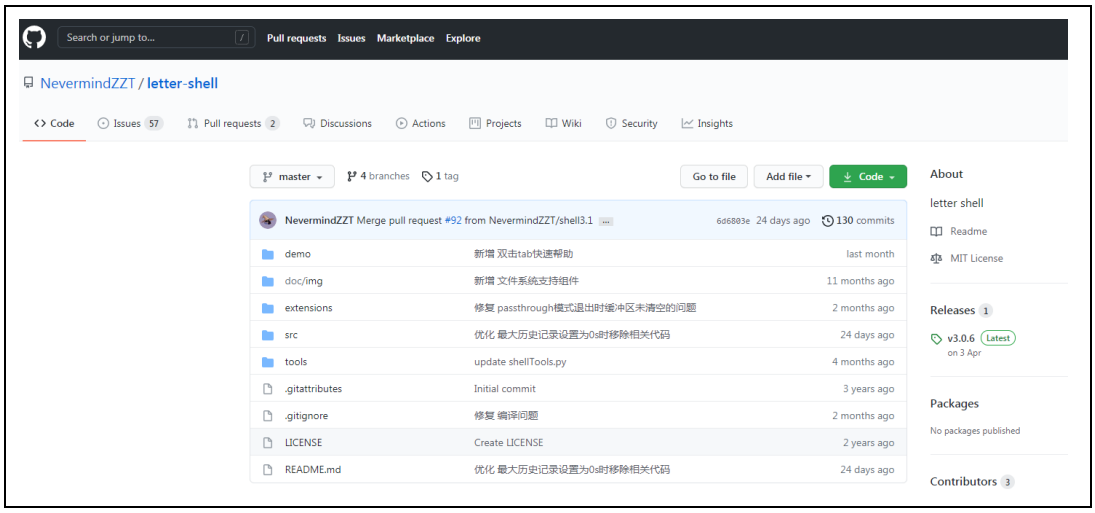
1.1. Main features

- Command auto completion
- Shortcut key function definition
- Command authority management
- User management
- Variable support
- Proxy function and parameter proxy analysis

2. Get the Letter shell source code

The github repository address of Letter shell is: https://github.com/NevermindZZT/letter-shell, as shown in ***Figure 2-1. Github repository address of Letter shell***.

Figure 2-1. Github repository address of Letter shell



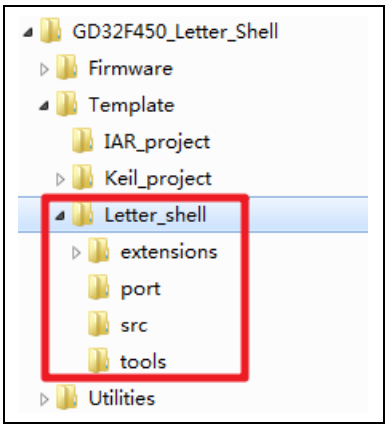
The Letter shell version used by this Application Note is 3.0.6.

3. GD32F450 Letter shell porting

3.1. Stucture of project files

This AN is built on the basis of the default Template project of GD32F4xx_Firmware_Library. Create a new Letter_shell folder, import Letter_shell core files, and copy the extensions, src and tools folders directly to the Letter_shell folder. Create a new port folder to store the porting interface files which are shell_cfg.h, shell_port.h and shell_port.c files. The details are shown in ***Figure 3-1. Stucture of project files***.

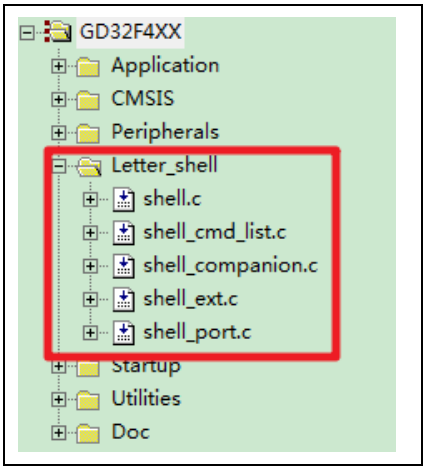
Figure 3-1. Stucture of project files



3.2. Build framework of Keil project

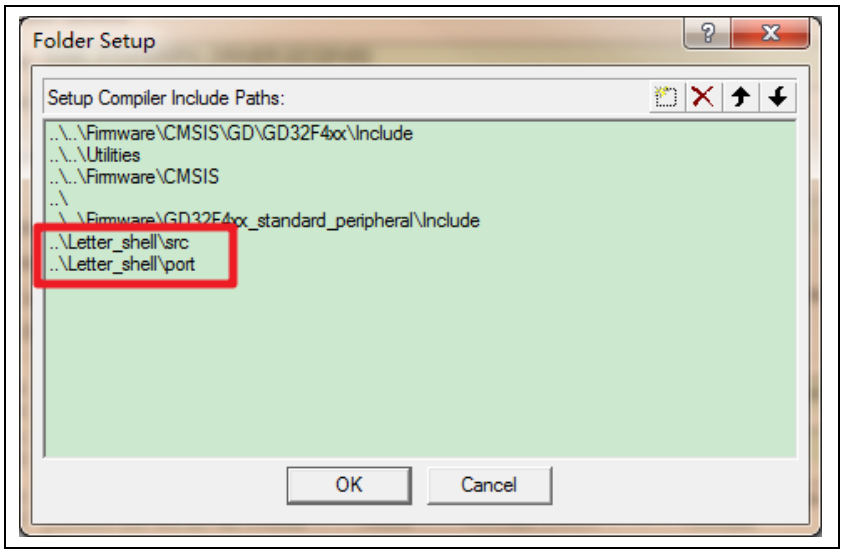
Take the Keil IDE as an example (other IDE projects are similar in construction, which will not be described here), add Letter_shell to the project, and add the c files under the src folder and port folder, as shown in ***Figure 3-2. Stucture of Keil project***.

Figure 3-2. Stucture of Keil project



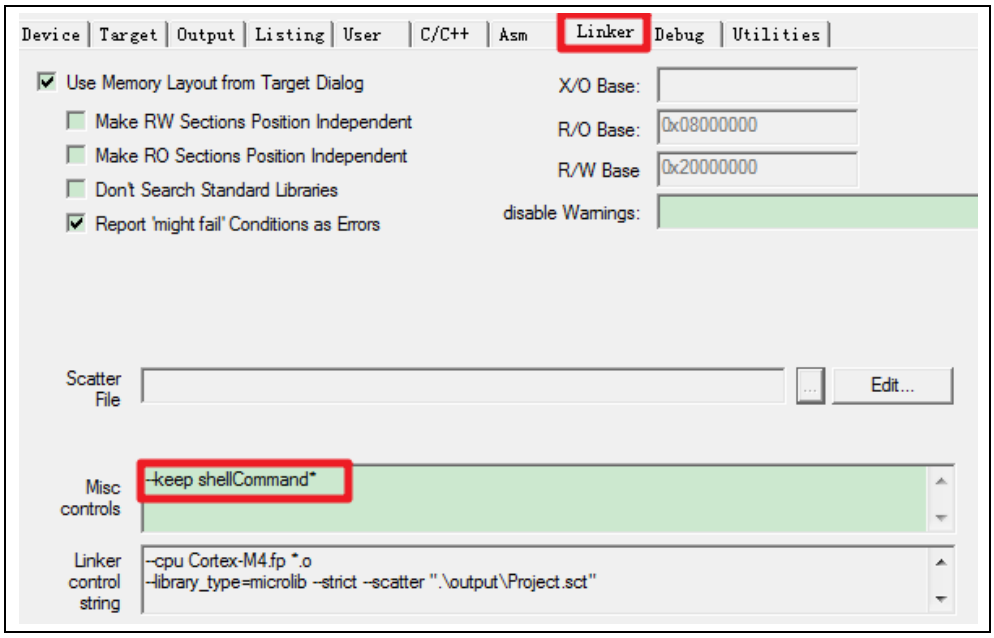
In the Setup Compiler Include Paths, include the header files in the src and port folders, as shown in ***Figure 3-3. Settings of Keil Folder Setup***.

Figure 3-3. Settings of Keil Folder Setup



In the Linker tab of the Keil project, add --keep shellCommand* to prevent it from being compiled and optimized, as shown in ***Figure 3-4. Settings of Keil Linker***.

Figure 3-4. Settings of Keil Linker



3.3. Coding the porting interface files

The porting interface file is mainly to implement shell_port.c, as shown in ***Table 3-1. Stucture of Shell***.

Table 3-1. Stucture of Shell

```
typedef struct shell_def
{
    struct
```

```
    {
        const struct shell_command *user;
        int activeTime;
        char *path;
    #if SHELL_USING_COMPANION == 1
        struct shell_companion_object *companions;
    #endif
    } info;
    struct
    {
        unsigned short length;
        unsigned short cursor;
        char *buffer;
        char *param[SHELL_PARAMETER_MAX_NUMBER];
        unsigned short bufferSize;
        unsigned short paramCount;
        int keyValue;
    } parser;
    struct
    {
        char *item[SHELL_HISTORY_MAX_NUMBER];
        unsigned short number;
        unsigned short record;
        signed short offset;
    } history;
    struct
    {
        void *base;
        unsigned short count;
    } commandList;
    struct
    {
        unsigned char isChecked : 1;
        unsigned char isActive : 1;
        unsigned char tabFlag : 1;
    } status;
    signed char (*read)(char *);
    void (*write)(const char);
} Shell;
```

From the definition of Shell structure, shell read and write functions need to be implemented, as shown in ***Table 3-2. Implementation of userShellRead function*** and ***Table 3-3. Implementation of userShellWrite function***.

Table 3-2. Implementation of userShellRead function

```
signed char userShellRead(char *data)
{
    *data = 0;

    if (usart_flag_get(EVAL_COM0, USART_FLAG_RBNE) != RESET) {
        *data = usart_data_receive(EVAL_COM0);
    }

    if (*data == 0) {
        return -1;
    }
    return 0;
}
```

Table 3-3. Implementation of userShellWrite function

```
void userShellWrite(char data)
{
    while (RESET == usart_flag_get(EVAL_COM0,USART_FLAG_TC));
    usart_data_transmit(EVAL_COM0, (uint8_t) data);
}
```

After implementing the userShellRead and userShellWrite functions, register them in the shell structure, as shown in ***Table 3-4. Register userShellRead and userShellWrite functions***.

Table 3-4. Register userShellRead and userShellWrite functions

```
……
shell.write = userShellWrite;
shell.read = userShellRead;
shellInit(&shell, shellBuffer, sizeof(shellBuffer)/sizeof(shellBuffer[0]));
……
```

3.4. Function call of Letter shell

There are not many interfaces for letter shell. The following two points should be noted:

1. Call userShellInit in the main program to complete Letter shell initialization;

2. Call the shellTask task periodically.

Specific as shown in ***Table 3-5. Function call of Letter shell***.

Table 3-5. Function call of Letter shell

```
……
userShellInit();
……
while (1){
    shellTask(&shell);
    delay_1ms(50);
}
……
```

4. Simple application of Lettle shell

4.1. Serial terminal software

For porting based on serial port, letter shell recommends using secureCRT software. The related key mapping in letter shell is designed according to secureCRT. When using other serial port software, you may need to modify the key value.

Open the secureCRT, after the correct configuration, download the program to the GD32F450i-EVAL development board, the execution result is shown in ***Figure 4-1. Print results of successful Letter shell porting***.

Figure 4-1. Print results of successful Letter shell porting

4.2. Add custom functions

Letter shell supports adding custom functions, here are 3 custom functions, as shown in ***Table 4-1. reboot function***, ***Table 4-2. led control function*** and ***Table 4-3. get random data function***.

Table 4-1. reboot function

```
int reboot(int argc, char *agrv[])
{
    printf(" %dparameter(s)\r\n", argc);
    for (char i = 1; i < argc; i++)
    {
        printf("%s\r\n", agrv[i]);
    }
    NVIC_SystemReset();
    return 0;
}
SHELL_EXPORT_CMD(SHELL_CMD_PERMISSION(0)|SHELL_CMD_TYPE(SHELL_TYPE_CMD_
MAIN), reboot, reboot, reboot);
```

Table 4-2. led control function

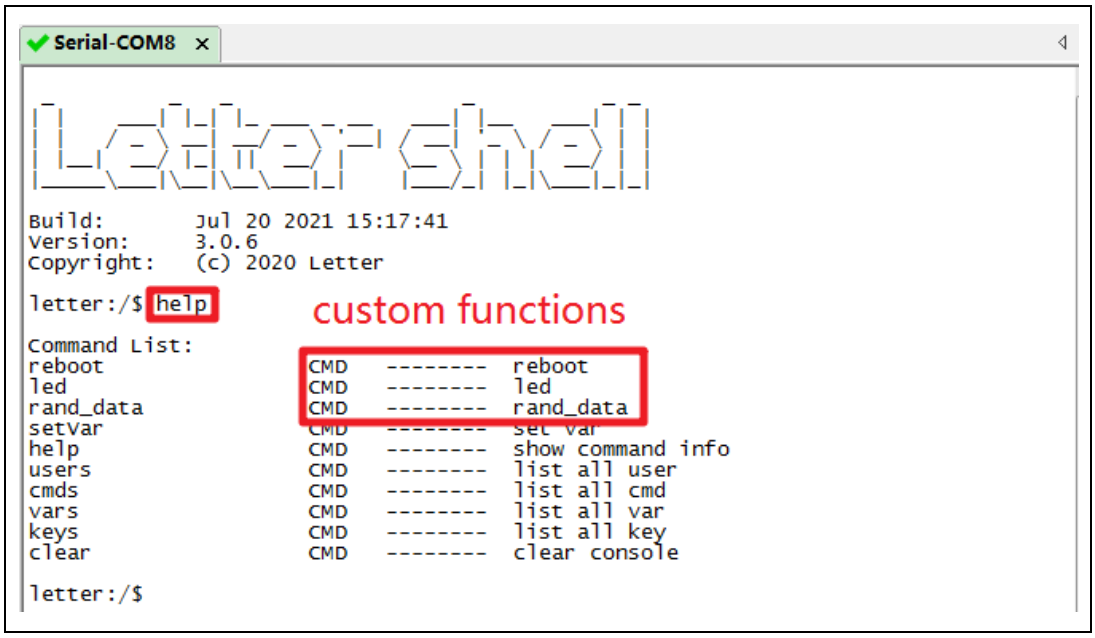
```
int led(int argc, char *agrv[])
{
    uint32_t temp, rtn, i;
    if(argc == 2){
        rtn = sscanf(agrv[1],"%d", &temp);
        if(rtn == 1){
            if(temp == 0){
                gd_eval_led_off(LED2);
                printf("LED2 is off!\r\n");
            }else if(temp == 1){
                gd_eval_led_on(LED2);
                printf("LED2 is on!\r\n");
            }else if(temp == 2){
                gd_eval_led_toggle(LED2);
                printf("LED2 is toggled!\r\n");
            }else if(temp == 3){
                for(i = 0; i < 6; i++){
                    gd_eval_led_toggle(LED2);
                    delay_1ms(400); // cannot be used in interrupt.
                }
                printf("LED is blinked!\r\n");
            }
        }
    }
    return 0;
}
SHELL_EXPORT_CMD(SHELL_CMD_PERMISSION(0)|SHELL_CMD_TYPE(SHELL_TYPE_CMD_
MAIN)|SHELL_CMD_DISABLE_RETURN, led, led, led);
```

Table 4-3. get random data function

```
int rand_data(int argc, char *agrv[])
{
    uint32_t temp;
    if(SUCCESS == trng_ready_check()){
        temp = trng_get_true_random_data();
        printf("Randon Data = 0x%08x\r\n", temp);
    }
    return 0;
}
SHELL_EXPORT_CMD(SHELL_CMD_PERMISSION(0)|SHELL_CMD_TYPE(SHELL_TYPE_CMD_
MAIN)|SHELL_CMD_DISABLE_RETURN, rand_data, rand_data, rand_data);
```

The user can type help to see which commands are supported, as shown in ***Figure 4-2. Result when tpying help***.

Figure 4-2. Result when tpying help



4.3. Execution result of reboot function

Type reboot in the secureCRT software to achieve the effect of printing and soft reset the chip, as shown in ***Figure 4-3. Result when tpying reboot***.

Figure 4-3. Result when tpying reboot

```
letter:/$ reboot
 1parameter(s)
 _          _   _                          _          _ _
| |      ___| |_| |_ ___ _.__      ___| |__     ___| | |
| |    / _ \ __| __/ _ \ '__|   / __| '_ \   / _ \ | |
| |__| (_) | |_| || (_) |      \__ \ | | | (_) | | |
|_____\___|\__|\__\___|_|      |___/_| |_|\___|_|_|

Build:       Jul 20 2021 15:17:41
Version:     3.0.6
Copyright:   (c) 2020 Letter

letter:/$
```

4.4. Execution result of led control function

Type led+number (0~3) in the secureCRT software to achieve the control effect of the LED, as shown in ***Figure 4-4. Result when tpying led***.

Figure 4-4. Result when tpying led

```
letter:/$ led 0
LED2 is off!

letter:/$ led 1
LED2 is on!

letter:/$ led 2
LED2 is toggled!

letter:/$ led 3
LED is blinked!

letter:/$
```

4.5. Execution result of get random data function

Type rand_data in the secureCRT software to achieve the effect of obtaining random numbers, as shown in ***Figure 4-5. Result when tpying rand_data***.

Figure 4-5. Result when tpying rand_data

```
letter:/$ rand_data
Randon Data = 0x83a5a48d

letter:/$ rand_data
Randon Data = 0x8e525d5c

letter:/$ rand_data
Randon Data = 0x4c85d0ee

letter:/$ rand_data
Randon Data = 0x24d0c816

letter:/$ rand_data
Randon Data = 0xaa1c9c20

letter:/$
```

Note: All commands can be automatically completed by pressing Tab key.

5. Revision history

Table 5-1. Revision history

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

Important Notice