

GigaDevice Semiconductor Inc.

Arm[®] Cortex[®]-M3 32-bit MCU

应用笔记

AN028

目录

目录.....	2
图索引	3
表索引	4
1. Hard fault 产生原因.....	5
1.1 硬件方面常见原因:	5
1.2 软件方面常见原因:	5
2. Cortex-M3 内核 HardFault 错误定位方法	6
2.1 更改 startup.s 的启动文件.....	6
2.2 hard_fault_handler_c 函数	6
2.3 Cortex-M3 内核错误寄存器说明	8
3. 通过 Jlink 排查 keil 程序的 Hard fault 错误	11
3.1 排查问题使用到的工具:	11
3.2 排查步骤.....	11
3.2.1 使用 keil 生成 map 和 lst 文件	11
3.2.2 保存出问题时候的 RAM	11
3.2.3 分析问题.....	12
3.3 Jlink Command 使用方法:	13
4. 历史版本	17

图索引

图 3-1. Keil 生成 map 和 lst 文件	11
图 3-2. 输入 halt 来停住内核	12
图 3-3. 保存 RAM 内容	12
图 3-4. 从 map 文件中查看栈	13
图 3-5. Bin 文件分析	13
图 3-6. f 命令	14
图 3-7. h 命令	14
图 3-8. s 命令	14
图 3-9. st 命令	14
图 3-10. 读取指令	15
图 3-11. 写指令	15
图 3-12. 擦除指令	16

表索引

表 4-1. 历史版本	17
-------------------	----

1. Hard fault 产生原因

1.1 硬件方面常见原因：

- 电源设计有错误，造成器件供电不稳；
- 电源质量不好，纹波，噪声过大；
- 器件接地不良；
- 对于带有 Vcap 引脚的器件，管脚处理不当；
- 电路中有强干扰源，对器件造成干扰；

1.2 软件方面常见原因：

- 使用了空指针；
- 对地址偏移量的计算有误；
- 数组越界导致程序出错；
- 动态内存使用不当，导致访问了已释放的内存地址；
- 通过地址访问了已失效的局部变量；

一般因为硬件造成 Hard Fault 错误的可能性较低，大多数都是软件原因造成的。所以遇到硬件中断错误，基本就是通过软件来排查。

2. Cortex-M3 内核 HardFault 错误定位方法

2.1 更改 startup.s 的启动文件

首先更改 startup.s 的启动文件，把里面的 HardFault_Handler 代码段换成下面的代码：

```
HardFault_Handler\  
  
        PROC  
  
        IMPORT hard_fault_handler_c  
  
        TST LR, #4  
  
        ITE EQ  
  
        MRSEQ R0, MSP  
  
        MRSNE R0, PSP  
  
        B hard_fault_handler_c  
  
        ENDP
```

2.2 hard_fault_handler_c 函数

然后把 hard_fault_handler_c 函数放在 c 文件的代码中。代码如下：

```
void hard_fault_handler_c(unsigned int * hardfault_args)  
{  
    static unsigned int stacked_r0;  
    static unsigned int stacked_r1;  
    static unsigned int stacked_r2;  
    static unsigned int stacked_r3;  
    static unsigned int stacked_r12;  
    static unsigned int stacked_lr;  
    static unsigned int stacked_pc;  
    static unsigned int stacked_psr;  
    static unsigned int SHCSR;  
    static unsigned char MFSR;  
    static unsigned char BFSR;
```

```

static unsigned short int UFSR;

static unsigned int HFSR;

static unsigned int DFSR;

static unsigned int MMAR;

static unsigned int BFAR;

stacked_r0 = ((unsigned long) hardfault_args[0]);

stacked_r1 = ((unsigned long) hardfault_args[1]);

stacked_r2 = ((unsigned long) hardfault_args[2]);

stacked_r3 = ((unsigned long) hardfault_args[3]);

stacked_r12 = ((unsigned long) hardfault_args[4]);

/*异常中断发生时，这个异常模式特定的物理寄存器 R14,即 lr 被设置成该异常模式将要
返回的地址*/

stacked_lr = ((unsigned long) hardfault_args[5]);

stacked_pc = ((unsigned long) hardfault_args[6]);

stacked_psr = ((unsigned long) hardfault_args[7]);

SHCSR = *((volatile unsigned long *) (0xE000ED24)); //系统 Handler 控制及状态寄存
器

MFSR = *((volatile unsigned char *) (0xE000ED28)); //存储器管理 fault 状态寄存器

BFSR = *((volatile unsigned char *) (0xE000ED29)); //总线 fault 状态寄存器

UFSR = *((volatile unsigned short int *) (0xE000ED2A)); //用法 fault 状态寄存器

HFSR = *((volatile unsigned long *) (0xE000ED2C)); //硬 fault 状态寄存器

DFSR = *((volatile unsigned long *) (0xE000ED30)); //调试 fault 状态寄存器

MMAR = *((volatile unsigned long *) (0xE000ED34)); //存储管理地址寄存器

BFAR = *((volatile unsigned long *) (0xE000ED38)); //总线 fault 地址寄存器

while (1);
}

```

执行程序后，若发生内核错误，则程序会运行到最后的 `while(1)`处。此时观察相应的堆栈和故

障寄存器值，stacked_lr 即为故障发生时进入故障中断前 pc 的值，在 MDK 软件调试状态下，假如 stacked_lr 的值为 0x1A002D08，在左下方的命令窗口输入“pc = 0x1A002D08”，回车，即可定位发生错误的代码位置。

2.3 Cortex-M3 内核错误寄存器说明

根据内核错误状态寄存器的值，对应下面的说明，也可以看出是发生了何种内核错误。

附录：Cortex-M3 内核错误寄存器说明

表D.17 系统Handler控制及状态寄存器SHCSR 0xE000_ED24

位段	名称	类型	复位值	描述
18	USGFAULTENA	R/W	0	用法 fault 服务例程使能位
17	BUSFAULTENA	R/W	0	总线 fault 服务例程使能位
16	MEMFAULTENA	R/W	0	存储器管理 fault 服务例程使能位
15	SVCALLPENDEDED	R/W	0	SVC 悬起中。本来已经要 SVC 服务例程，但是却被更高优先级异常取代
14	BUSFAULTPENDEDED	R/W	0	总线 fault 悬起中，细节同上。
13	MEMFAULTPENDEDED	R/W	0	存储器管理 fault 悬起中，细节同上
12	USGFAULTPENDEDED	R/W	0	用法 fault 悬起中，细节同上
11	SYSTICKACT	R/W	0	SysTick 异常活动中
10	PENDSVACT	R/W	0	PendSV 异常活动中
9	-	-	-	-
8	MONITORACT	R/W	0	Monitor 异常活动中
7	SVCALLACT	R/W	0	SVC 异常活动中
6:4	-	-	-	-
3	USGFAULTACT	R/W	0	用法 fault 异常活动中
2	-	-	-	-
1	BUSFAULTACT	R/W	0	总线 fault 异常活动中
0	MEMFAULTACT	R/W	0	存储器管理 fault 异常活动中

表 D.18 存储器管理 fault 状态寄存器(MFSR) 0xE000_ED28

位段	名称	类型	复位值	描述
7	MMARVALID	-	0	=1 时表示 MMAR 有效
6:5	-	-	-	-
4	MSTKERR	R/Wc	0	入栈时发生错误
3	MUNSTKERR	R/Wc	0	出栈时发生错误
2	-	-	-	-
1	DACCVIOL	R/Wc	0	数据访问违例
0	IACCVIOL	R/Wc	0	取指访问违例

表 D.19 总线 fault 状态寄存器(BFSR) 0xE000_ED29

位段	名称	类型	复位值	描述
7	BFARVALID	-	0	=1 时表示 BFAR 有效
6:5	-	-	-	-
4	STKERR	R/Wc	0	入栈时发生错误
3	UNSTKERR	R/Wc	0	出栈时发生错误
2	IMPRECISERR	R/Wc	0	不精确的数据访问违例 (violation)
1	PRECISERR	R/Wc	0	精确的数据访问违例
0	IBUSERR	R/Wc	0	取指时的访问违例

表 D.20 用法 fault 状态寄存器(UFSR), 地址: 0xE000_ED2A

位段	名称	类型	复位值	描述
9	DIVBYZERO	R/Wc	0	表示除法运算时除数为零 (只有在 DIV_0_TRP 置位时才会发生)
8	UNALIGNED	R/Wc	0	未对齐访问导致的 fault
7:4	-	-	-	-
3	NOCP	R/Wc	0	试图执行协处理器相关指令
2	INVPC	R/Wc	0	在异常返回时试图非法地加载 EXC_RETURN 到 PC。包括非法的指令, 非法的上下文以及非法的值。The return PC 指向的指令试图设置 PC 的值 (要理解此位的含义, 还需学习后面的讨论中断级异常的章节)
1	INVSTATE	R/Wc	0	试图切入 ARM 状态
0	UNDEFINSTR	R/Wc	0	执行的指令其编码是未定义的——解码不能

表 D.21 硬 fault 状态寄存器 0xE000_ED2C

位段	名称	类型	复位值	描述
31	DEBUGEVT	R/Wc	0	硬 fault 因调试事件而产生
30	FORCED	R/Wc	0	硬 fault 是总线 fault, 存储器管理 fault 或是用法 fault 上访的结果
29:2	-	-	-	-
1	VECTBL	R/Wc	0	硬 fault 是在取向量时发生的
0	-	-	-	-

表 D.22 调试 fault 状态寄存器(DFSR) 0xE000_ED30

位段	名称	类型	复位值	描述
4	EXTERNAL	R/Wc	0	EDBGREQ 信号有效
3	VCATCH	R/Wc	0	发生向量加载
2	DWTTRAP	R/Wc	0	发生 DWT 匹配
1	BKPT	R/Wc	0	执行到 BKPT 指令
0	HALTED	R/Wc	0	在 NVIC 中请求 HALT

表 D.23 存储管理地址寄存器(MMAR) 0xE000_ED34

位段	名称	类型	复位值	描述
31:0	MMAR	R	-	触发存储管理 fault 的地址

表 D.24 总线 fault 地址寄存器(BFAR) 0xE000_ED38

位段	名称	类型	复位值	描述
31:0	BFAR	R	-	触发总线 fault 的地址

3. 通过 Jlink 排查 keil 程序的 Hard fault 错误

3.1 排查问题使用到的工具：

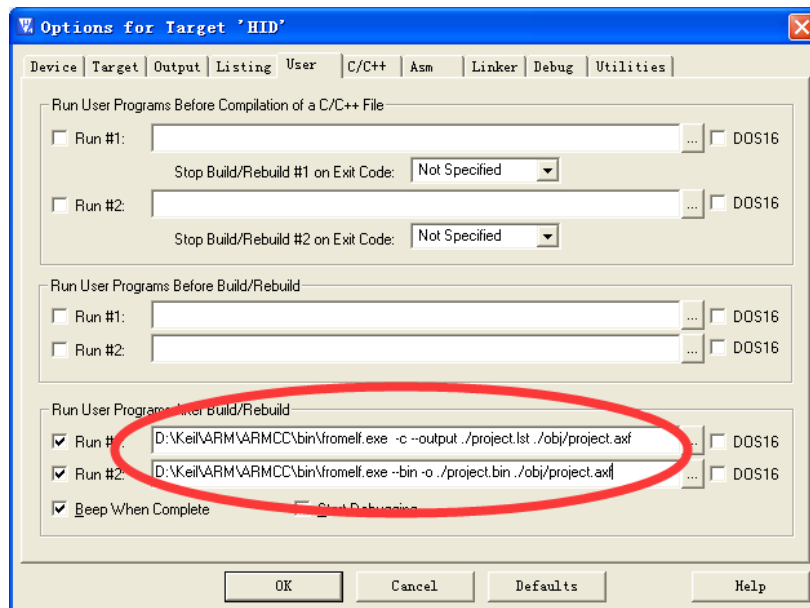
Jlink, Segger (Jlink 上位机), Keil。

3.2 排查步骤

3.2.1 使用 keil 生成 map 和 lst 文件

Map 文件是 keil 自动生成的，他被放在工程路径下。里面能标明每个函数、每个变量的位置。
lst 文件反映的是每一个函数，每一条指令的 PC 指针，在 keil 中需要调用 USER 命令生成。如 [图 3-1. Keil 生成 map 和 lst 文件](#) 所示。

图 3-1. Keil 生成 map 和 lst 文件



```
D:\Keil\ARM\ARMCC\bin\fromelf.exe -c --output ./project.lst ./obj/project.axf
```

D:\Keil\ARM\ARMCC\bin\fromelf.exe 表示的是 fromelf.exe 的路径；

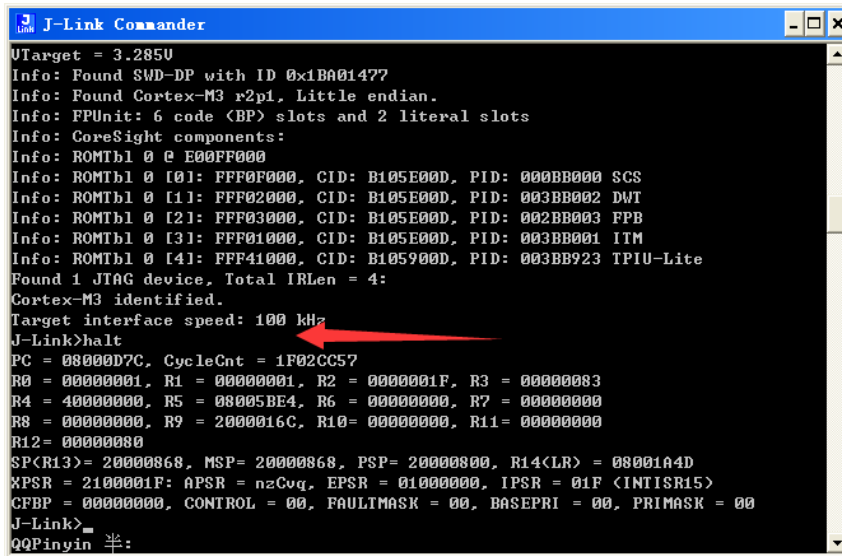
./obj/project.axf 表示生成的 axf 文件位置，可能需要根据实际情况调整；

3.2.2 保存出问题时候的 RAM

出问题的时候调用。别断电，接上 Jlink，调用 Segger 里面的 Jlink command 来获取现场：

1、先输入一个“USB”让 Jlink 接上设备，然后输入 halt 来停住内核，如 [图 3-2. 输入 halt 来停住内核](#) 所示；

图 3-2. 输入 halt 来停住内核



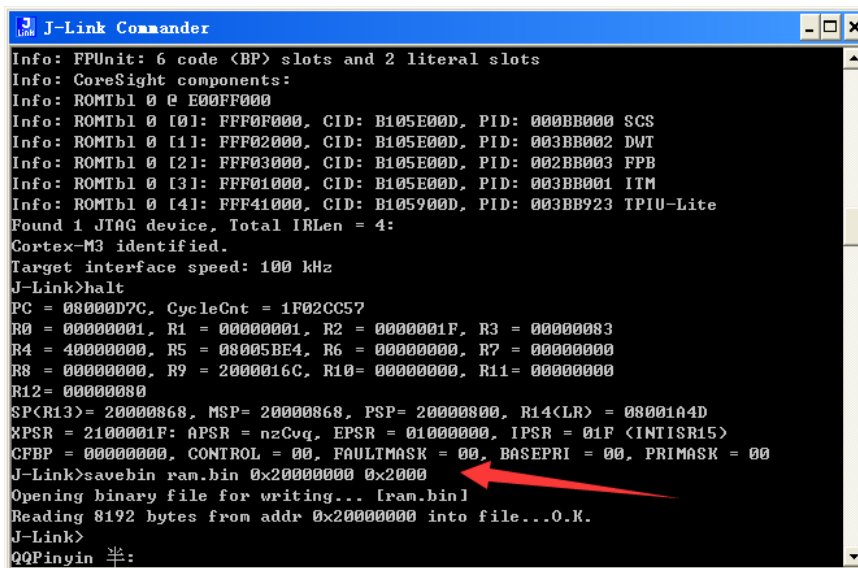
```

J-Link Commander
UTarget = 3.285U
Info: Found SWD-DP with ID 0x1BA01477
Info: Found Cortex-M3 r2p1, Little endian.
Info: FPUUnit: 6 code (BP) slots and 2 literal slots
Info: CoreSight components:
Info: ROMTbl 0 @ E00FF000
Info: ROMTbl 0 [0]: FFF0F000, CID: B105E00D, PID: 000BB000 SCS
Info: ROMTbl 0 [1]: FFF02000, CID: B105E00D, PID: 003BB002 DWT
Info: ROMTbl 0 [2]: FFF03000, CID: B105E00D, PID: 002BB003 FPB
Info: ROMTbl 0 [3]: FFF01000, CID: B105E00D, PID: 003BB001 ITM
Info: ROMTbl 0 [4]: FFF41000, CID: B105900D, PID: 003BB923 TPIU-Lite
Found 1 JTAG device, Total IRLen = 4:
Cortex-M3 identified.
Target interface speed: 100 kHz
J-Link>halt
PC = 08000D7C, CycleCnt = 1F02CC57
R0 = 00000001, R1 = 00000001, R2 = 0000001F, R3 = 00000083
R4 = 40000000, R5 = 08005BE4, R6 = 00000000, R7 = 00000000
R8 = 00000000, R9 = 2000016C, R10 = 00000000, R11 = 00000000
R12 = 00000080
SP(R13) = 20000868, MSP = 20000868, PSP = 20000800, R14(LR) = 08001A4D
XPSR = 2100001F: APSR = nzCvq, EPSR = 01000000, IPSR = 01F (INTISR15)
CFBP = 00000000, CONTROL = 00, FAULTMASK = 00, BASEPRI = 00, PRIMASK = 00
J-Link>
QQPinyin 半:
  
```

2、调用 `savebin ram.bin 0x20000000 0x2000` 将 RAM 中的内容全部保存下来，如 [图 3-3. 保存 RAM 内容](#) 所示；

保存下来的东西被存在放 Segger 的安装目录中。

图 3-3. 保存 RAM 内容



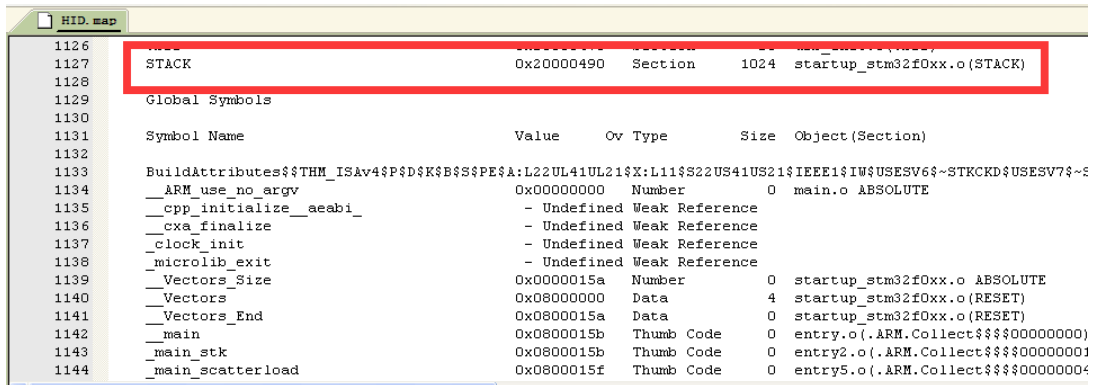
```

J-Link Commander
Info: FPUUnit: 6 code (BP) slots and 2 literal slots
Info: CoreSight components:
Info: ROMTbl 0 @ E00FF000
Info: ROMTbl 0 [0]: FFF0F000, CID: B105E00D, PID: 000BB000 SCS
Info: ROMTbl 0 [1]: FFF02000, CID: B105E00D, PID: 003BB002 DWT
Info: ROMTbl 0 [2]: FFF03000, CID: B105E00D, PID: 002BB003 FPB
Info: ROMTbl 0 [3]: FFF01000, CID: B105E00D, PID: 003BB001 ITM
Info: ROMTbl 0 [4]: FFF41000, CID: B105900D, PID: 003BB923 TPIU-Lite
Found 1 JTAG device, Total IRLen = 4:
Cortex-M3 identified.
Target interface speed: 100 kHz
J-Link>halt
PC = 08000D7C, CycleCnt = 1F02CC57
R0 = 00000001, R1 = 00000001, R2 = 0000001F, R3 = 00000083
R4 = 40000000, R5 = 08005BE4, R6 = 00000000, R7 = 00000000
R8 = 00000000, R9 = 2000016C, R10 = 00000000, R11 = 00000000
R12 = 00000080
SP(R13) = 20000868, MSP = 20000868, PSP = 20000800, R14(LR) = 08001A4D
XPSR = 2100001F: APSR = nzCvq, EPSR = 01000000, IPSR = 01F (INTISR15)
CFBP = 00000000, CONTROL = 00, FAULTMASK = 00, BASEPRI = 00, PRIMASK = 00
J-Link>savebin ram.bin 0x20000000 0x2000
Opening binary file for writing... [ram.bin]
Reading 8192 bytes from addr 0x20000000 into file...O.K.
J-Link>
QQPinyin 半:
  
```

3.2.3 分析问题

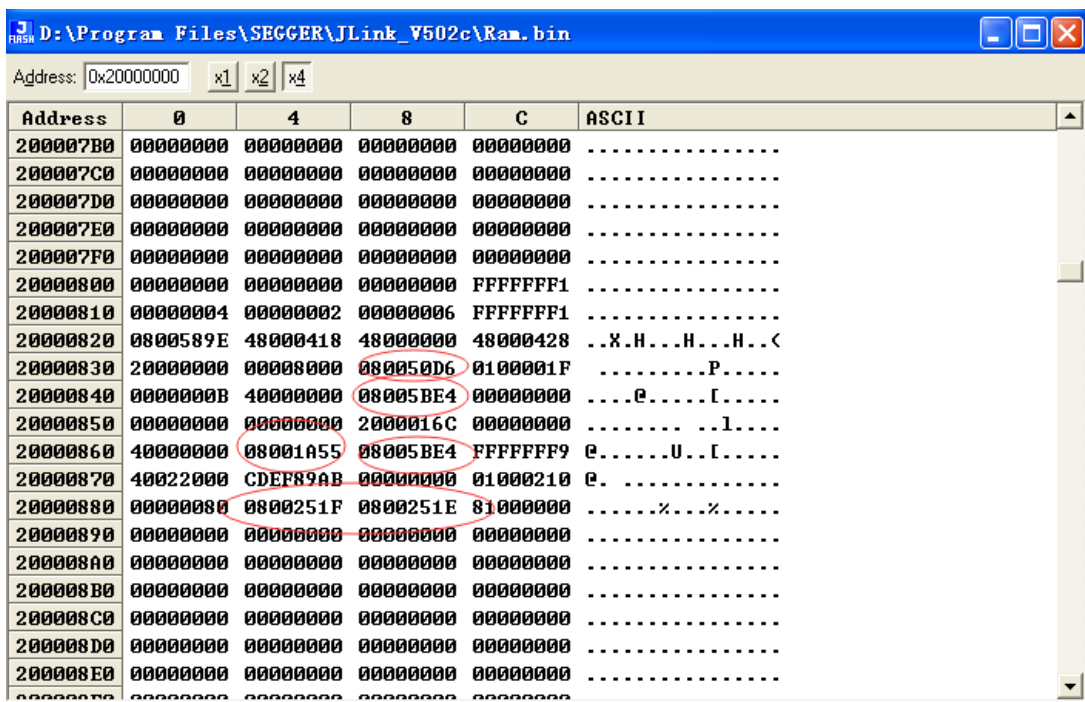
1、查看 map 文件，找到栈的位置。如 [图 3-4. 从 map 文件中查看栈](#) 所示。

图 3-4. 从 map 文件中查看栈



2、打开保存的 bin 文件进行分析，找到进入硬件中断前调用了哪些函数，在使用哪个变量。

图 3-5. Bin 文件分析



从栈的底部往上看，确定哪个地方的值是函数指针，然后对照 lst 文件去逐一查看，分析，就能大致知道是在执行哪个函数，哪一条指令，或者是调用某个参数导致的硬件中断错误的。

通过 map 文件可以知道每个变量的位置，可以直接去查看我们保存下来的 ram 中变量的当前情况来分析程序逻辑。

3.3 Jlink Command 使用方法:

f Firmware info 用来查看 Jlink 的硬件版本

图 3-6. f 命令

```
J-Link>f
Firmware: J-Link V9 compiled Sep 18 2015 19:53:12
Hardware: V9.20
```

h halt 用来停止 MCU 内核，可以查看内核的 PC 指针等特殊寄存器

图 3-7. h 命令

```
J-Link>h
PC = 08000D7C, CycleCnt = F39B01CC
R0 = 00000001, R1 = 00000001, R2 = 0000001F, R3 = 00000083
R4 = 40000000, R5 = 08005BE4, R6 = 00000000, R7 = 00000000
R8 = 00000000, R9 = 2000016C, R10= 00000000, R11= 00000000
R12= 00000080
SP<R13>= 20000868, MSP= 20000868, PSP= 20000800, R14<LR> = 08001A4D
XPSR = 2100001F: APSR = nzCvq, EPSR = 01000000, IPSR = 01F <INTISR15>
CFBP = 00000000, CONTROL = 00, FAULIMASK = 00, BASEPRI = 00, PRIMASK = 00
```

g go 用来激活被 halt 的内核

Sleep Waits the given time (in milliseconds). Syntax: Sleep <delay>用来延时

s Single step the target chip 单步调试代码，可以先执行 halt，然后再来单步调试

图 3-8. s 命令

```
J-Link>halt
PC = 08000D7C, CycleCnt = A970C614
R0 = 00000001, R1 = 00000001, R2 = 0000001F, R3 = 00000083
R4 = 40000000, R5 = 08005BE4, R6 = 00000000, R7 = 00000000
R8 = 00000000, R9 = 2000016C, R10= 00000000, R11= 00000000
R12= 00000080
SP<R13>= 20000868, MSP= 20000868, PSP= 20000800, R14<LR> = 08001A4D
XPSR = 2100001F: APSR = nzCvq, EPSR = 01000000, IPSR = 01F <INTISR15>
CFBP = 00000000, CONTROL = 00, FAULIMASK = 00, BASEPRI = 00, PRIMASK = 00
J-Link>s
08000D7C: FE E7 B #0x04
```

st Show hardware status 显示 Jlink 当前状态

图 3-9. st 命令

```
J-Link>st
VTarget=3.285V
ITarget=0mA
TCK=0 TDI=0 TDO=0 TMS=1 TRES=1 TRST=0
Supported target interface speeds:
- 12 MHz/n, (n)=1). => 12000kHz, 6000kHz, 4000kHz, ...
- Adaptive clocking
J-Link>
```

hwinfo Show hardware info 显示 Jlink 的硬件信息

mem Read memory. Syntax: mem [<Zone>:]<Addr>, <NumBytes> (hex)

- mem8 Read 8-bit items. Syntax: mem8 [<Zone>:]<Addr>, <NumBytes> (hex)
- mem16 Read 16-bit items. Syntax: mem16 [<Zone>:]<Addr>, <NumItems> (hex)
- mem32 Read 32-bit items. Syntax: mem32 [<Zone>:]<Addr>, <NumItems> (hex)

■ 读取指令:

图 3-10. 读取指令

```
J-Link>mem 0x8000000 20
08000000 = 90 08 00 20 71 01 00 08 5F 0E 00 08 7D 0D 00 08
08000010 = 5D 0E 00 08 15 02 00 08 FB 1D 00 08 00 00 00 00
J-Link>mem8 0x8000000 2
08000000 = 90 08
J-Link>mem16 0x8000000 2
08000000 = 0890 2000
J-Link>mem32 0x8000000 2
08000000 = 20000890 08000171
J-Link>
```

- w1 Write 8-bit items. Syntax: w1 [<Zone>:]<Addr>, <Data> (hex)
- w2 Write 16-bit items. Syntax: w2 [<Zone>:]<Addr>, <Data> (hex)
- w4 Write 32-bit items. Syntax: w4 [<Zone>:]<Addr>, <Data> (hex)

■ 写指令:

图 3-11. 写指令

```
J-Link>w2 0x20000000 55
Writing 0055 -> 20000000
J-Link>mem16 0x20000000 2
20000000 = 0055 0017
J-Link>w4 0x20000000 5566
Writing 00005566 -> 20000000
J-Link>mem32 0x20000000 2
20000000 = 00005566 00040000
J-Link>
```

- erase Erase internal flash of selected device. Syntax: Erase

擦除指令，先选定器件然后再来执行擦除

图 3-12. 擦除指令

```

J-Link>device stm32f100c8
Info: Device "STM32F100C8" selected.
Reconnecting to target...
Info: Found SWD-DP with ID 0x1BA01477
Info: Found SWD-DP with ID 0x1BA01477
Info: Found Cortex-M3 r2p1, Little endian.
Info: FPUUnit: 6 code <BP> slots and 2 literal slots
Info: CoreSight components:
Info: ROMTbl 0 @ E00FF000
Info: ROMTbl 0 [0]: FFF0F000, CID: B105E00D, PID: 000BB000 SCS
Info: ROMTbl 0 [1]: FFF02000, CID: B105E00D, PID: 003BB002 DWT
Info: ROMTbl 0 [2]: FFF03000, CID: B105E00D, PID: 002BB003 FPB
Info: ROMTbl 0 [3]: FFF01000, CID: B105E00D, PID: 003BB001 ITM
Info: ROMTbl 0 [4]: FFF41000, CID: B105900D, PID: 003BB923 TPIU-Lite
J-Link>erase
Erasing device <STM32F100C8>...
Info: J-Link: Flash download: Only internal flash banks will be erase
To enable erasing of other flash banks like QSPI or CFI, it needs to
via "exec EnableEraseAllFlashBanks"
Info: J-Link: Flash download: Total time needed: 4.503s <Prepare: 1.0
e: 0.000s, Erase: 1.797s, Program: 0.000s, Verify: 0.000s, Restore: 1
Erasing done.
J-Link>_

```

loadfile Load data file into target memory.

Syntax: loadfile <filename>, [<addr>]

Supported extensions: *.bin, *.mot, *.hex, *.srec

<addr> is needed for bin files only. //用来下载文件

loadbin Load *.bin file into target memory.

Syntax: loadbin <filename>, <addr> //用来下载 bin 文件

savebin Saves target memory into binary file. //用来保存 bin 文件

Syntax: savebin <filename>, <addr>, <NumBytes>

SetPC Set the PC to specified value. Syntax: SetPC <Addr> //用来设置 PC 指针，可以让程序从某个地方开始执行

4. 版本历史

表 4-1. 版本历史

版本号.	描述	日期
1.0	首次发布	2021 年 04 月 30 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 its 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 its 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.