GigaDevice Semiconductor Inc.

Arm$^{®}$ Cortex$^{®}$-M3 32-bit MCU

Application Note
AN028

Table of Contents

List of Figures

List of Tables

1. The causes of Hard fault The causes of Hard fault

1.1 Common causes of hardware

- Power design error, resulting in device power supply instability.
- The quality of power supply is not very well, too much noise.
- The device is not grounded properly.
- For devices with V_{cap} pin, the handling is not proper.
- There is a strong interference source in the circuit, causing interference to the device.

1.2 Common causes of software

- Null pointer was used.
- The calculation of address offset is incorrect.
- Program error caused by array bound.
- The improper use of the dynamic memory, leading to access memory address has been released.
- Visit to the local variable by address which has already invalidated.

Generally, the possibility of Hard Fault caused by hardware is low, most are caused by software. Therefore, if the hardware interrupt error occurs, the error investigation will be through software.

2. Debug location method of kernel HardFault

2.1 change the startup file of startup.s

First, change the startup file of startup.s, replace the HardFault_Handler code in the following code.

```
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 function

Then put the hard_fault_handler_c function in the code of the C file. The code shows as below.

```
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]);

    /* When an abnormal interrupt occurs, the abnormal mode register R14 (LR register) is
set as the address which the exception mode will return. */

    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))); // System Handler control and
status register

    MFSR = (*((volatile unsigned char *)(0xE000ED28)));     // Memory management fault
status register

    BFSR = (*((volatile unsigned char *)(0xE000ED29)));     // Bus fault status register

    UFSR = (*((volatile unsigned short int *)(0xE000ED2A)));// Use fault status register

    HFSR = (*((volatile unsigned long *)(0xE000ED2C)));     // Hard fault status register

    DFSR = (*((volatile unsigned long *)(0xE000ED30)));     // Debug fault status register

    MMAR = (*((volatile unsigned long *)(0xE000ED34)));     //  Memory  management
address register

    BFAR = (*((volatile unsigned long *)(0xE000ED38))); // Bus fault address register

    while (1);
```

```
}
```

If a kernel error occurs after executing the program, the program runs to the last while (1). At this moment, the corresponding stack and fault register values are observed. stacked_lr is the value of PC before the fault occurs when the fault is interrupted. In the debugging state of MDK, if the value of stacked_lr is 0x1A002D08, enter "PC = 0x1A002D08" in the command window at the bottom left" will locate the code location where the error occurred

2.3 description of the Cortex-M3 kernel error register

According to the values of kernel error status register show as below, it is also possible to see what kernel error has occurred.

Appendix: description of the Cortex-M3 kernel error register

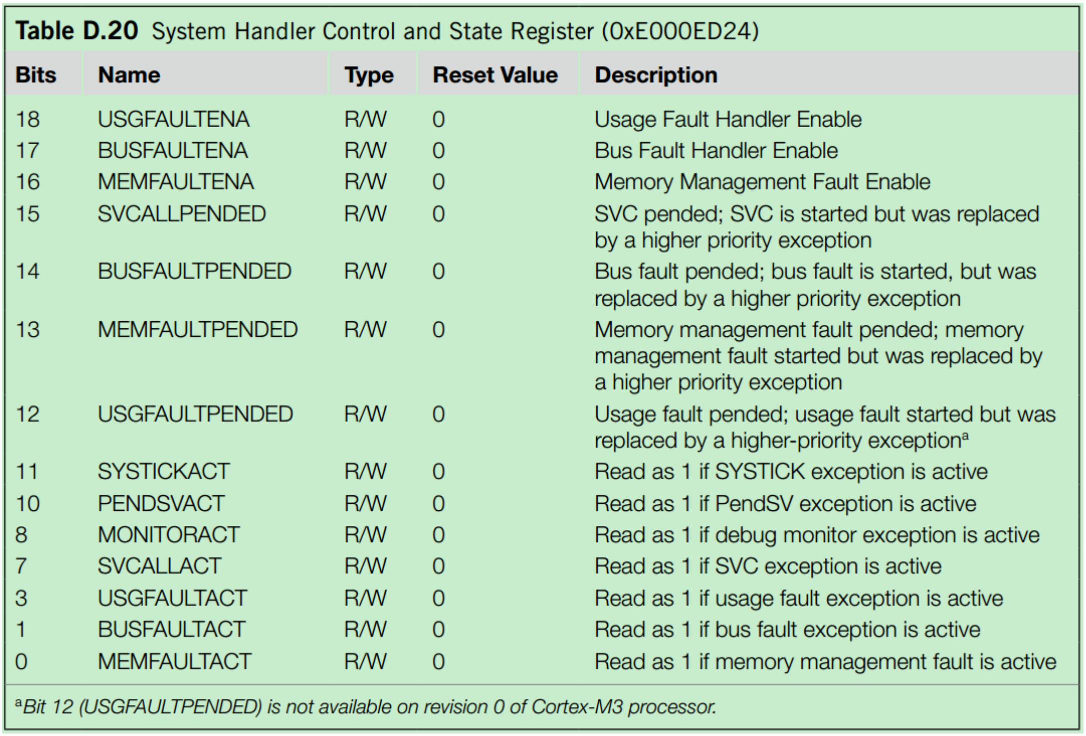


Table D.20 System Handler Control and State Register (0xE000ED24)

Bits	Name	Type	Reset Value	Description
18	USGFAULTENA	R/W	0	Usage Fault Handler Enable
17	BUSFAULTENA	R/W	0	Bus Fault Handler Enable
16	MEMFAULTENA	R/W	0	Memory Management Fault Enable
15	SVCALLPENDED	R/W	0	SVC pended; SVC is started but was replaced by a higher priority exception
14	BUSFAULTPENDED	R/W	0	Bus fault pended; bus fault is started, but was replaced by a higher priority exception
13	MEMFAULTPENDED	R/W	0	Memory management fault pended; memory management fault started but was replaced by a higher priority exception
12	USGFAULTPENDED	R/W	0	Usage fault pended; usage fault started but was replaced by a higher-priority exception[a]
11	SYSTICKACT	R/W	0	Read as 1 if SYSTICK exception is active
10	PENDSVACT	R/W	0	Read as 1 if PendSV exception is active
8	MONITORACT	R/W	0	Read as 1 if debug monitor exception is active
7	SVCALLACT	R/W	0	Read as 1 if SVC exception is active
3	USGFAULTACT	R/W	0	Read as 1 if usage fault exception is active
1	BUSFAULTACT	R/W	0	Read as 1 if bus fault exception is active
0	MEMFAULTACT	R/W	0	Read as 1 if memory management fault is active

[a]*Bit 12 (USGFAULTPENDED) is not available on revision 0 of Cortex-M3 processor.*

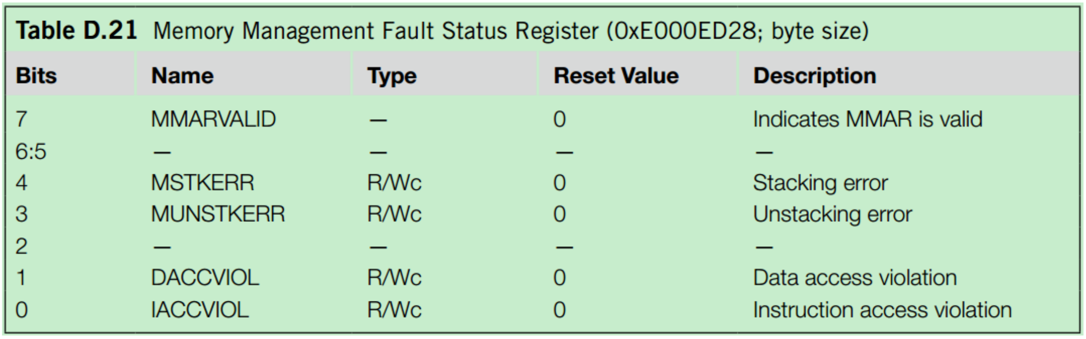


Table D.21 Memory Management Fault Status Register (0xE000ED28; byte size)

Bits	Name	Type	Reset Value	Description
7	MMARVALID	—	0	Indicates MMAR is valid
6:5	—	—	—	—
4	MSTKERR	R/Wc	0	Stacking error
3	MUNSTKERR	R/Wc	0	Unstacking error
2	—	—	—	—
1	DACCVIOL	R/Wc	0	Data access violation
0	IACCVIOL	R/Wc	0	Instruction access violation

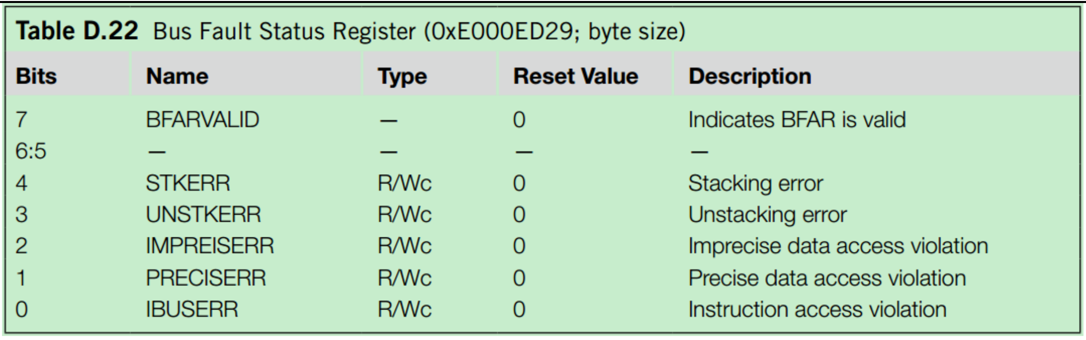


Table D.22 Bus Fault Status Register (0xE000ED29; byte size)

Bits	Name	Type	Reset Value	Description
7	BFARVALID	—	0	Indicates BFAR is valid
6:5	—	—	—	—
4	STKERR	R/Wc	0	Stacking error
3	UNSTKERR	R/Wc	0	Unstacking error
2	IMPREISERR	R/Wc	0	Imprecise data access violation
1	PRECISERR	R/Wc	0	Precise data access violation
0	IBUSERR	R/Wc	0	Instruction access violation

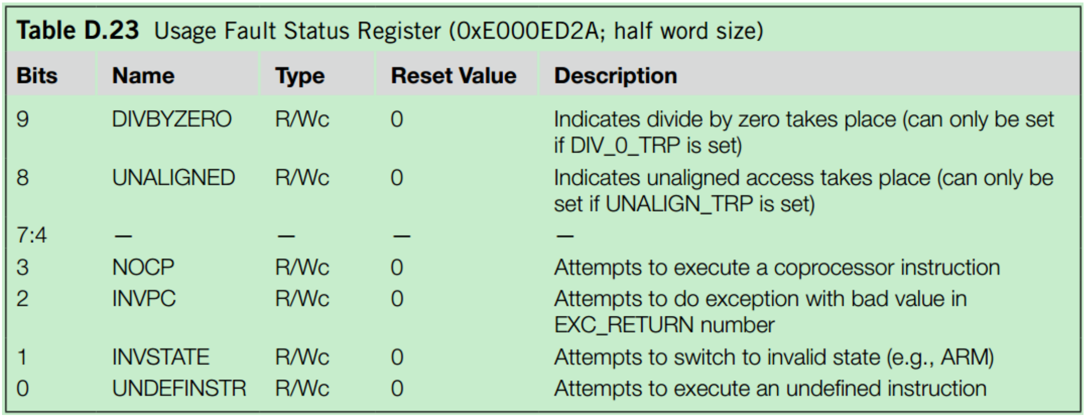


Table D.23 Usage Fault Status Register (0xE000ED2A; half word size)

Bits	Name	Type	Reset Value	Description
9	DIVBYZERO	R/Wc	0	Indicates divide by zero takes place (can only be set if DIV_0_TRP is set)
8	UNALIGNED	R/Wc	0	Indicates unaligned access takes place (can only be set if UNALIGN_TRP is set)
7:4	—	—	—	—
3	NOCP	R/Wc	0	Attempts to execute a coprocessor instruction
2	INVPC	R/Wc	0	Attempts to do exception with bad value in EXC_RETURN number
1	INVSTATE	R/Wc	0	Attempts to switch to invalid state (e.g., ARM)
0	UNDEFINSTR	R/Wc	0	Attempts to execute an undefined instruction

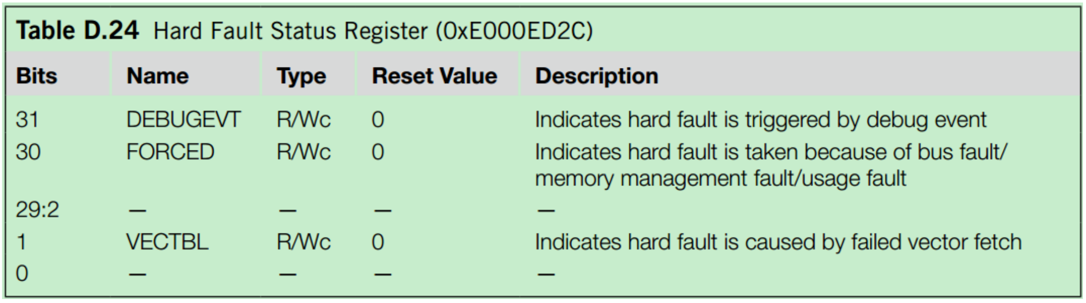


Table D.24 Hard Fault Status Register (0xE000ED2C)

Bits	Name	Type	Reset Value	Description
31	DEBUGEVT	R/Wc	0	Indicates hard fault is triggered by debug event
30	FORCED	R/Wc	0	Indicates hard fault is taken because of bus fault/ memory management fault/usage fault
29:2	—	—	—	—
1	VECTBL	R/Wc	0	Indicates hard fault is caused by failed vector fetch
0	—	—	—	—

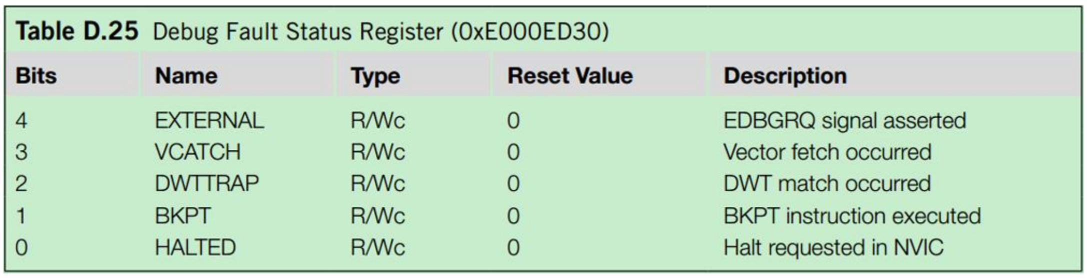


Table D.25 Debug Fault Status Register (0xE000ED30)

Bits	Name	Type	Reset Value	Description
4	EXTERNAL	R/Wc	0	EDBGRQ signal asserted
3	VCATCH	R/Wc	0	Vector fetch occurred
2	DWTTRAP	R/Wc	0	DWT match occurred
1	BKPT	R/Wc	0	BKPT instruction executed
0	HALTED	R/Wc	0	Halt requested in NVIC

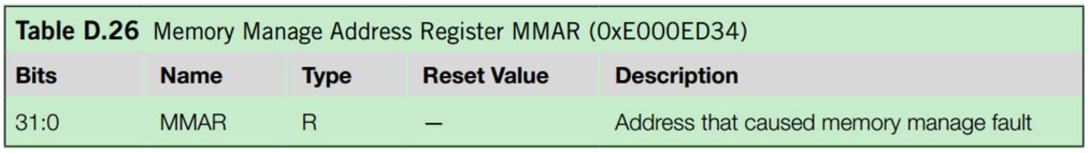


Table D.26 Memory Manage Address Register MMAR (0xE000ED34)

Bits	Name	Type	Reset Value	Description
31:0	MMAR	R	—	Address that caused memory manage fault

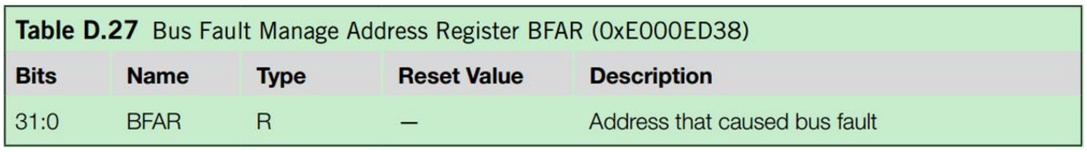


Table D.27 Bus Fault Manage Address Register BFAR (0xE000ED38)

Bits	Name	Type	Reset Value	Description
31:0	BFAR	R	—	Address that caused bus fault

3. Check hard fault error of Keil program through JLINK

3.1 Tools used for troubleshooting

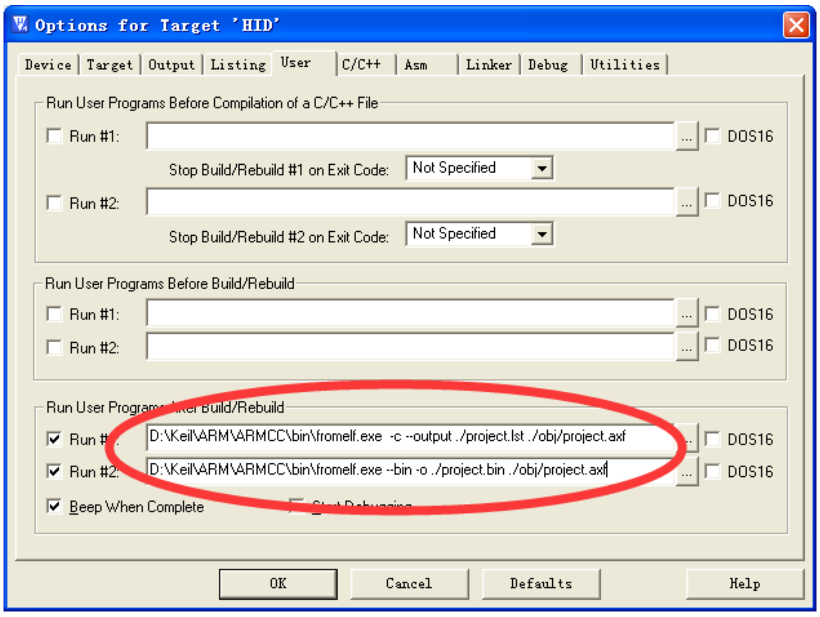
Jlink, Segger (the upper computer of Jlink), Keil.

3.2 Troubleshooting steps

3.2.1 Generate map file and lst file using Keil

The Map file is generated automatically by Keil and placed in the path of the engineering. It can indicate the location of each function and each variable. The lst file reflects the PC pointer of each function and each instruction. It is generated by USER command in Keil as shown in ***Figure 3-1. The map and LST files generated by Keil***

Figure 3-1. The map and LST files generated by Keil



Options for Target 'HID'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Run User Programs Before Compilation of a C/C++ File
- [ ] Run #1: — [ ] DOS16
- Stop Build/Rebuild #1 on Exit Code: Not Specified
- [ ] Run #2: — [ ] DOS16
- Stop Build/Rebuild #2 on Exit Code: Not Specified

Run User Programs Before Build/Rebuild
- [ ] Run #1: — [ ] DOS16
- [ ] Run #2: — [ ] DOS16

Run User Programs After Build/Rebuild
- [✓] Run #1: `D:\Keil\ARM\ARMCC\bin\fromelf.exe  -c --output ./project.lst ./obj/project.axf` — [ ] DOS16
- [✓] Run #2: `D:\Keil\ARM\ARMCC\bin\fromelf.exe --bin -o ./project.bin ./obj/project.axf` — [ ] DOS16
- [✓] Beep When Complete
- [ ] Start Debugging

OK | Cancel | Defaults | Help

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

D:\Keil\ARM\ARMCC\bin\fromelf.exe represents the path of fromelf.exe.

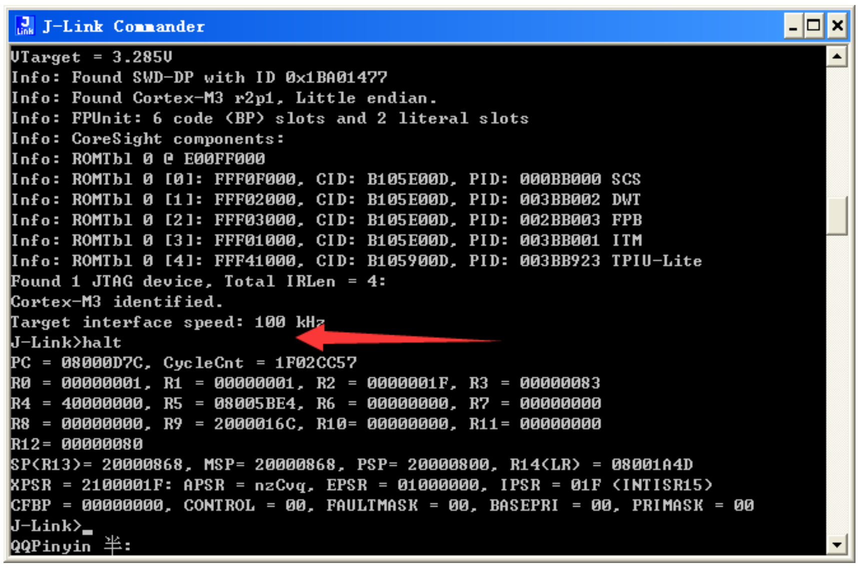
./obj/project.axf represents the location of the AXF file. It may need to be adjusted according to the actual situation.

3.2.2 Save the RAM when a problem occurs

Use this function when a problem occurs. Don't power off and connect to Jlink. Call the Jlink command in Segger to get the current information.

1. First, enter “USB” to let the Jlink connect to the device. Then enter “hatl” to stop the kernel.

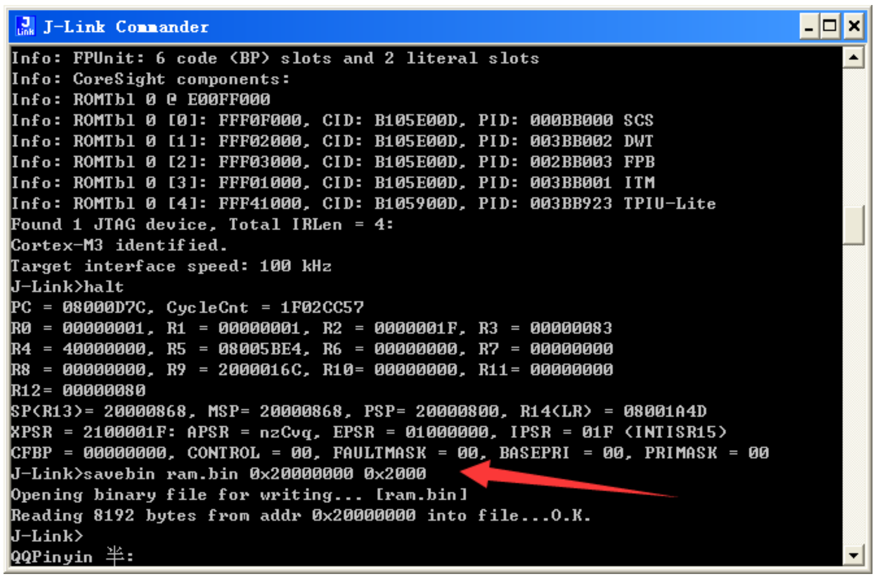
Figure 3-2. Input halt to stop the kernal



```
J-Link Commander
VTarget = 3.285V
Info: Found SWD-DP with ID 0x1BA01477
Info: Found Cortex-M3 r2p1, Little endian.
Info: FPUnit: 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. Call “savebin ram.bin 0x20000000 0x2000” to save all the content in RAM. The saved items are present in the installation directory of the Segger.

Figure 3-3. Save the RAM content

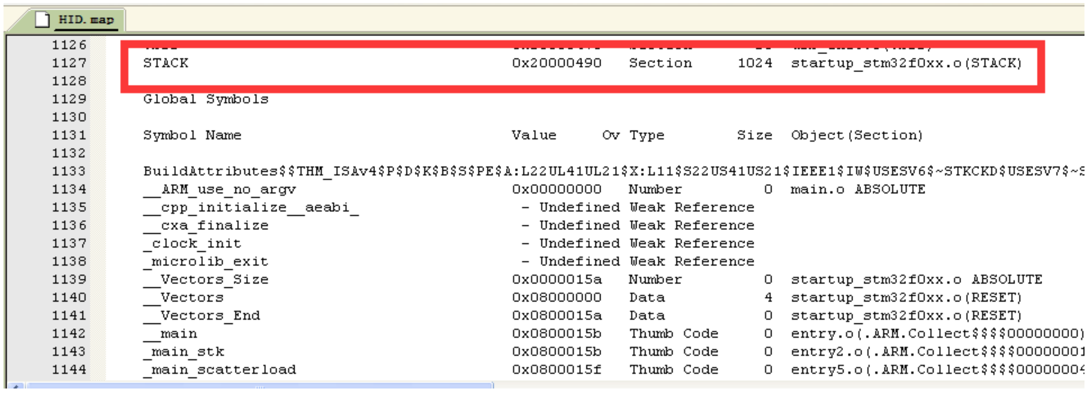


```
J-Link Commander
Info: FPUnit: 6 code (BP) slots and 2 literal slots
Info: CoreSight components:
Info: ROMTbl 0 @ E00FF000
Info: ROMTbl 0 [0]: FFF0F000, CID: B105E00D, PID: 000BB000 SCS
Info: ROMTbl 0 [1]: FFF02000, CID: B105E00D, PID: 003BB002 DWT
Info: ROMTbl 0 [2]: FFF03000, CID: B105E00D, PID: 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 Analysis problems

1. Find the location of the stack through check the map file.

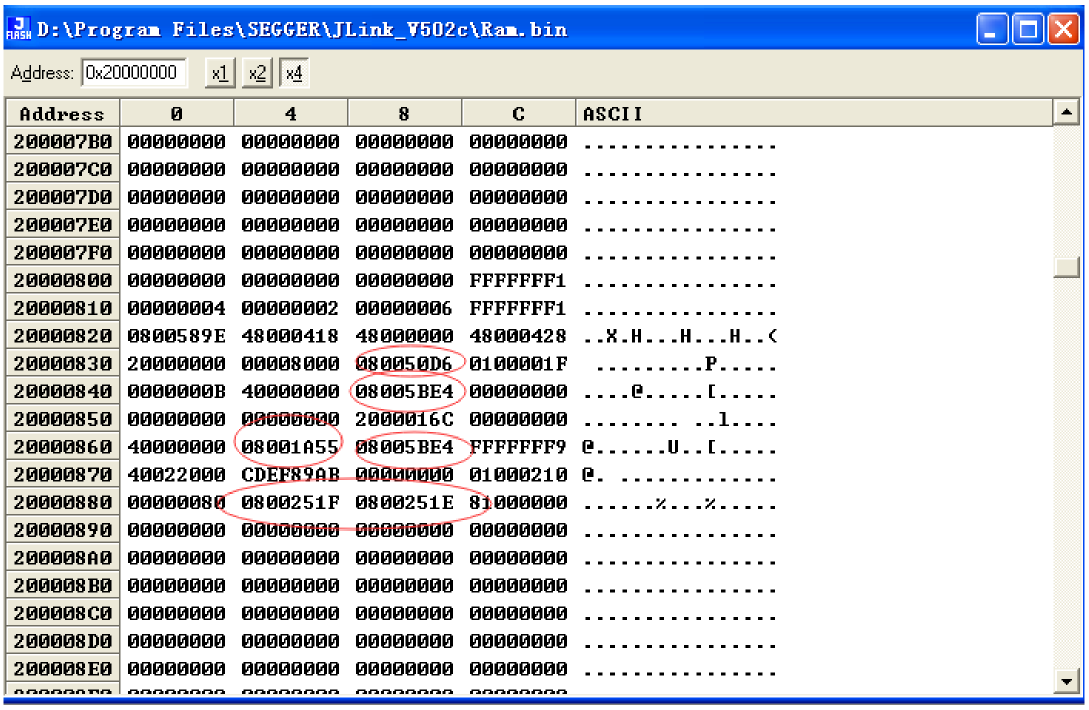
Figure 3-4. View the stack from the map file



```
STACK                                   0x20000490   Section      1024  startup_stm32f0xx.o(STACK)

Global Symbols

Symbol Name                             Value     Ov Type        Size  Object(Section)

BuildAttributes$$THM_ISAv4$P$D$K$B$S$PE$A:L22UL41UL21$X:L11$S22US41US21$IEEE1$IW$USESV6$~STKCKD$USESV7$~S
__ARM_use_no_argv                       0x00000000   Number         0  main.o ABSOLUTE
__cpp_initialize__aeabi_                 - Undefined Weak Reference
__cxa_finalize                           - Undefined Weak Reference
_clock_init                              - Undefined Weak Reference
_microlib_exit                           - Undefined Weak Reference
__Vectors_Size                          0x0000015a   Number         0  startup_stm32f0xx.o ABSOLUTE
__Vectors                               0x08000000   Data           4  startup_stm32f0xx.o(RESET)
__Vectors_End                           0x0800015a   Data           0  startup_stm32f0xx.o(RESET)
__main                                  0x0800015b   Thumb Code     0  entry.o(.ARM.Collect$$$$00000000)
_main_stk                               0x0800015b   Thumb Code     0  entry2.o(.ARM.Collect$$$$00000001
_main_scatterload                       0x0800015f   Thumb Code     0  entry5.o(.ARM.Collect$$$$00000004
```

2. Open the saved bin file for analysis. Find which functions are called and which variables are used before the hardware interrupt.

Figure 3-5. Analyze the BIN file



```
Address   0        4        8        C        ASCII
200007B0  00000000 00000000 00000000 00000000 ................
200007C0  00000000 00000000 00000000 00000000 ................
200007D0  00000000 00000000 00000000 00000000 ................
200007E0  00000000 00000000 00000000 00000000 ................
200007F0  00000000 00000000 00000000 00000000 ................
20000800  00000000 00000000 00000000 FFFFFFF1 ................
20000810  00000004 00000002 00000006 FFFFFFF1 ................
20000820  0800589E 48000418 48000000 48000428 ..X.H...H...H..(
20000830  20000000 00008000 080050D6 0100001F  .........P.....
20000840  0000000B 40000000 08005BE4 00000000 ....@.....[.....
20000850  00000000 00000000 2000016C 00000000 ........ ..l....
20000860  40000000 08001A55 08005BE4 FFFFFFF9 @......U..[.....
20000870  40022000 CDEF89AB 00000000 01000210 @. .............
20000880  00000080 0800251F 0800251E 81000000 ......%...%.....
20000890  00000000 00000000 00000000 00000000 ................
200008A0  00000000 00000000 00000000 00000000 ................
200008B0  00000000 00000000 00000000 00000000 ................
200008C0  00000000 00000000 00000000 00000000 ................
200008D0  00000000 00000000 00000000 00000000 ................
200008E0  00000000 00000000 00000000 00000000 ................
```

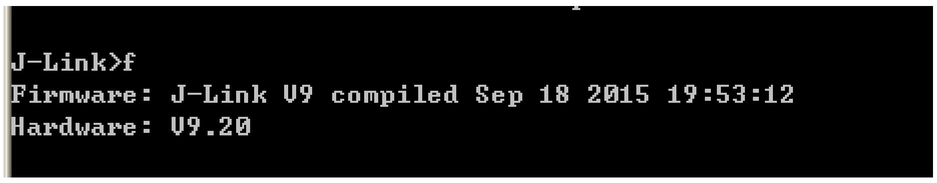
Look up from the bottom of the stack to confirm the function pointer. Comparing the lst file one by one and analyzing them, you can generally know which function, which instruction, or which parameter caused the hardware interrupt error.

The location of each variable can be knew through the map file. You can analyze the program logic by looking directly at the current status of the variables in the RAM.

3.3 Usage method of Jlink Command

f   Firmware info. Used to view the hardware version of Jlink.

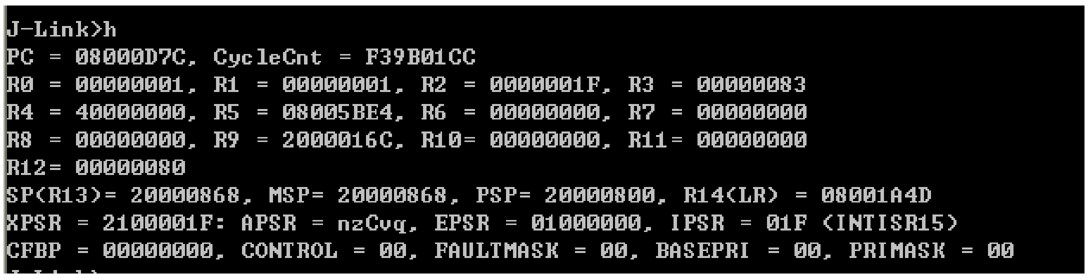
Figure 3-6. f command



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

h   halt. Used to stop the MCU kernel, the PC pointer or other special registers can be viewed.

Figure 3-7. h command



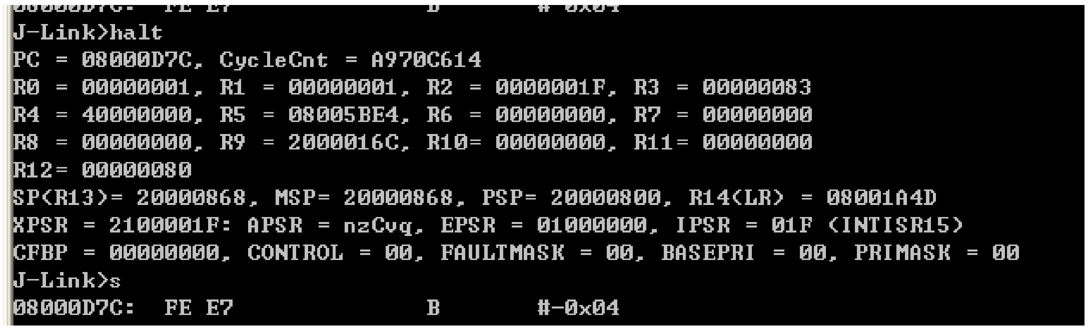
```
J-Link>h
PC = 08000D7C, CycleCnt = F39B01CC
R0 = 00000001, R1 = 00000001, R2 = 0000001F, R3 = 00000083
R4 = 40000000, R5 = 08005BE4, R6 = 00000000, R7 = 00000000
R8 = 00000000, R9 = 2000016C, R10= 00000000, R11= 00000000
R12= 00000080
SP(R13)= 20000868, MSP= 20000868, PSP= 20000800, R14(LR) = 08001A4D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

g   go. Used to activate the kernel that is halt.

Sleep   Waits the given time (in milliseconds). Syntax: Sleep <delay> for delay

s   Single step the target chip. Debug the code in single step. The halt can be execute first and then debugged in single step.

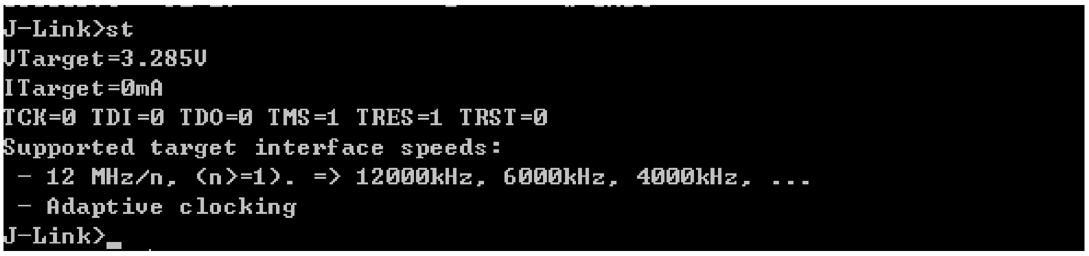
Figure 3-8. s command



```
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, FAULTMASK = 00, BASEPRI = 00, PRIMASK = 00
J-Link>s
08000D7C:  FE E7              B       #-0x04
```

st   Show hardware status. Display the current state of Jlink.

Figure 3-9. st command



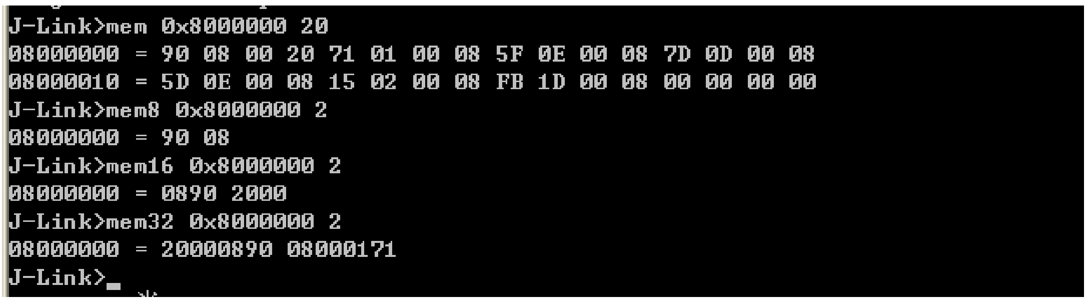
```
J-Link>st
VTarget=3.285V
ITarget=0mA
TCK=0 TDI=0 TDO=0 TMS=1 TRES=1 TRST=0
Supported target interface speeds:
 - 12 MHz/n, (n>=1). => 12000kHz, 6000kHz, 4000kHz, ...
 - Adaptive clocking
J-Link>_
```

hwinfo   Show hardware info. Display the hardware information of Jlink.

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)

- Read instruction：

Figure 3-10. Read instruction

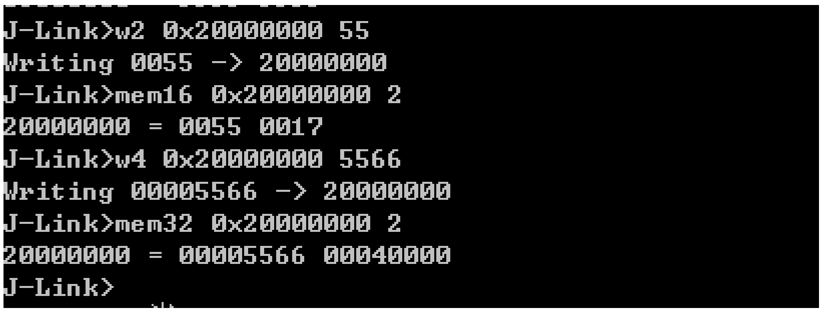


```
J-Link>mem 0x8000000 20
08000000 = 90 08 00 20 71 01 00 08 5F 0E 00 08 7D 0D 00 08
08000010 = 5D 0E 00 08 15 02 00 08 FB 1D 00 08 00 00 00 00
J-Link>mem8 0x8000000 2
08000000 = 90 08
J-Link>mem16 0x8000000 2
08000000 = 0890 2000
J-Link>mem32 0x8000000 2
08000000 = 20000890 08000171
J-Link>_
```

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)

- Write instruction：

Figure 3-11. Write instruction

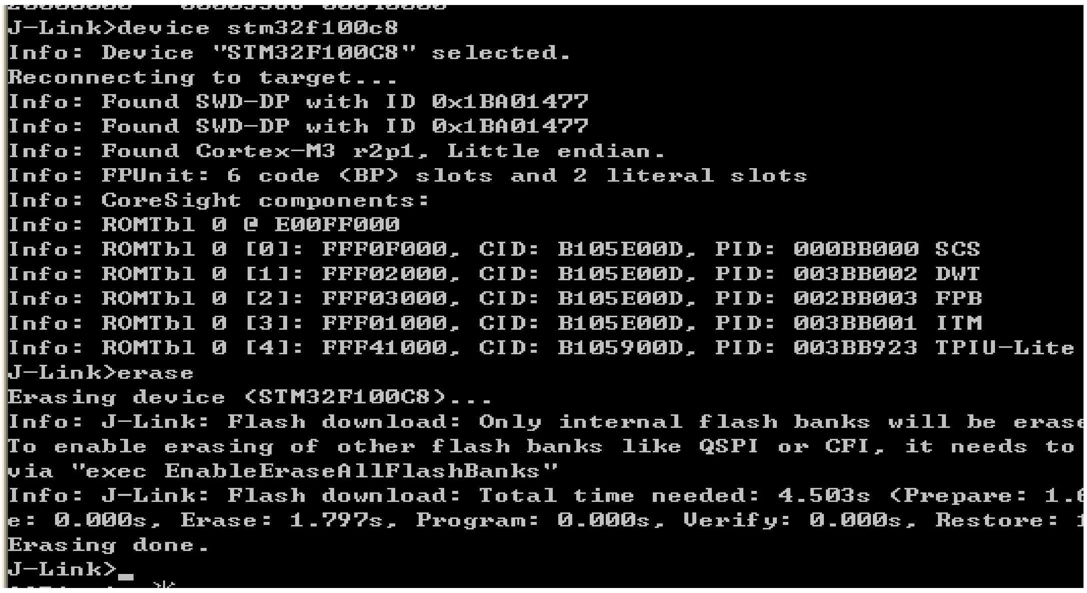


```
J-Link>w2 0x20000000 55
Writing 0055 -> 20000000
J-Link>mem16 0x20000000 2
20000000 = 0055 0017
J-Link>w4 0x20000000 5566
Writing 00005566 -> 20000000
J-Link>mem32 0x20000000 2
20000000 = 00005566 00040000
J-Link>
```

erase Erase internal flash of selected device. Syntax: Erase

Erase instruction, select the device first and then perform the erase.

Figure 3-12. Erase instruction



```
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: FPUnit: 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.
e: 0.000s, Erase: 1.797s, Program: 0.000s, Verify: 0.000s, Restore: 
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. // Used to download files.

loadbin Load *.bin file into target memory.

Syntax: loadbin <filename>, <addr> // Used to download bin files.

savebin Saves target memory into binary file. // Used to save bin files.

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

SetPC Set the PC to specified value. Syntax: SetPC <Addr> // Used to set the PC pointer. The program can start execute from a specified location.

4. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Apr.30, 2021