GigaDevice Semiconductor Inc.

Arm® Cortex®-M3 32-bit MCU

Application Note
AN028
# Table of Contents

Table of Contents ................................................................................................................. 2  
List of Figures ......................................................................................................................... 3  
1. The causes of Hard fault The causes of Hard fault......................................................... 5  
   1.1 Common causes of hardware......................................................................................... 5  
   1.2 Common causes of software......................................................................................... 5  
2. Debug location method of kernel HardFault................................................................. 6  
   2.1 change the startup file of startup.s ............................................................................ 6  
   2.2 hard_fault_handler_c function .................................................................................. 6  
   2.3 description of the Cortex-M3 kernel error register .................................................... 8  
3. Check hard fault error of Keil program through JLINK ............................................. 10  
   3.1 Tools used for troubleshooting.................................................................................... 10  
   3.2 Troubleshooting steps ................................................................................................. 10  
   3.2.1 Generate map file and lst file using Keil ................................................................. 10  
   3.2.2 Save the RAM when a problem occurs ................................................................. 10  
   3.2.3 Analysis problems ................................................................................................. 11  
   3.3 Usage method of Jlink Command .............................................................................. 12  
4. Revision history ............................................................................................................... 16
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>The map and LST files generated by Keil</td>
<td>10</td>
</tr>
<tr>
<td>3-2</td>
<td>Input halt to stop the kernel</td>
<td>11</td>
</tr>
<tr>
<td>3-3</td>
<td>Save the RAM content</td>
<td>11</td>
</tr>
<tr>
<td>3-4</td>
<td>View the stack from the map file</td>
<td>12</td>
</tr>
<tr>
<td>3-5</td>
<td>Analyze the BIN file</td>
<td>12</td>
</tr>
<tr>
<td>3-6</td>
<td>f command</td>
<td>13</td>
</tr>
<tr>
<td>3-7</td>
<td>h command</td>
<td>13</td>
</tr>
<tr>
<td>3-8</td>
<td>s command</td>
<td>13</td>
</tr>
<tr>
<td>3-9</td>
<td>st command</td>
<td>13</td>
</tr>
<tr>
<td>3-10</td>
<td>Read instruction</td>
<td>14</td>
</tr>
<tr>
<td>3-11</td>
<td>Write instruction</td>
<td>14</td>
</tr>
<tr>
<td>3-12</td>
<td>Erase instruction</td>
<td>15</td>
</tr>
</tbody>
</table>
List of Tables

Table 4-1. Revision history .............................................................................................................................................. 16
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 $V_{\text{cap}}$ 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.

```assembly
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.

```c
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>
<thead>
<tr>
<th>Table D.20</th>
<th>System Handler Control and State Register (0xE000ED24)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>Name</td>
</tr>
<tr>
<td>18</td>
<td>USGFaultEn</td>
</tr>
<tr>
<td>17</td>
<td>BusFaultEn</td>
</tr>
<tr>
<td>16</td>
<td>MemFaultEn</td>
</tr>
<tr>
<td>15</td>
<td>SVCALLPended</td>
</tr>
<tr>
<td>14</td>
<td>BusFaultPended</td>
</tr>
<tr>
<td>13</td>
<td>MemFaultPended</td>
</tr>
<tr>
<td>12</td>
<td>USGFaultPended</td>
</tr>
<tr>
<td>11</td>
<td>SYSTICKACT</td>
</tr>
<tr>
<td>10</td>
<td>PENDSVCT</td>
</tr>
<tr>
<td>8</td>
<td>MONITORACT</td>
</tr>
<tr>
<td>7</td>
<td>SVCALLCT</td>
</tr>
<tr>
<td>3</td>
<td>USGFaultTact</td>
</tr>
<tr>
<td>1</td>
<td>BusFaultTact</td>
</tr>
<tr>
<td>0</td>
<td>MemFaultTact</td>
</tr>
</tbody>
</table>

*Bit 12 (USGFaultPended) is not available on revision 0 of Cortex-M3 processor.

<table>
<thead>
<tr>
<th>Table D.21</th>
<th>Memory Management Fault Status Register (0xE000ED28; byte size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>Name</td>
</tr>
<tr>
<td>7</td>
<td>MMARVALID</td>
</tr>
<tr>
<td>6:5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>MSTKERR</td>
</tr>
<tr>
<td>3</td>
<td>MUNSTKERR</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>DACCVIOL</td>
</tr>
<tr>
<td>0</td>
<td>IACCVIOL</td>
</tr>
</tbody>
</table>
### Table D.22  Bus Fault Status Register (0xE000ED29; byte size)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Type</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>BFARVALID</td>
<td>—</td>
<td>0</td>
<td>Indicates BFAR is valid</td>
</tr>
<tr>
<td>6:5</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>4</td>
<td>STKFRR</td>
<td>R/Wc</td>
<td>0</td>
<td>Stacking error</td>
</tr>
<tr>
<td>3</td>
<td>UNSTKERR</td>
<td>R/Wc</td>
<td>0</td>
<td>Unstacking error</td>
</tr>
<tr>
<td>2</td>
<td>IMPREISERR</td>
<td>R/Wc</td>
<td>0</td>
<td>Imprecise data access violation</td>
</tr>
<tr>
<td>1</td>
<td>PRECISERR</td>
<td>R/Wc</td>
<td>0</td>
<td>Precise data access violation</td>
</tr>
<tr>
<td>0</td>
<td>BUSERR</td>
<td>R/Wc</td>
<td>0</td>
<td>Instruction access violation</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Type</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>DIVBYZERO</td>
<td>R/Wc</td>
<td>0</td>
<td>Indicates divide by zero takes place (can only be set if DIV_0_TRP is set)</td>
</tr>
<tr>
<td>6</td>
<td>UNALIGNED</td>
<td>R/Wc</td>
<td>0</td>
<td>Indicates unaligned access takes place (can only be set if UNALGN_TRP is set)</td>
</tr>
<tr>
<td>7:4</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>NOPC</td>
<td>R/Wc</td>
<td>0</td>
<td>Attempts to execute a coprocessor instruction</td>
</tr>
<tr>
<td>2</td>
<td>INVCPC</td>
<td>R/Wc</td>
<td>0</td>
<td>Attempts to do exception with bad value in EXC_RETURN number</td>
</tr>
<tr>
<td>1</td>
<td>INVSATE</td>
<td>R/Wc</td>
<td>0</td>
<td>Attempts to switch to invalid state (e.g., ARM)</td>
</tr>
<tr>
<td>0</td>
<td>UNDEFINSTR</td>
<td>R/Wc</td>
<td>0</td>
<td>Attempts to execute an undefined instruction</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Type</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>DEBUG_EVT</td>
<td>R/Wc</td>
<td>0</td>
<td>Indicates hard fault is triggered by debug event</td>
</tr>
<tr>
<td>30</td>
<td>FORCED</td>
<td>R/Wc</td>
<td>0</td>
<td>Indicates hard fault is taken because of bus fault/memory management fault/usage fault</td>
</tr>
<tr>
<td>29:2</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>VECTBL</td>
<td>R/Wc</td>
<td>0</td>
<td>Indicates hard fault is caused by failed vector fetch</td>
</tr>
<tr>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Type</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>EXTERNAL</td>
<td>R/Wc</td>
<td>0</td>
<td>EDBGFRQ signal asserted</td>
</tr>
<tr>
<td>3</td>
<td>Vcatch</td>
<td>R/Wc</td>
<td>0</td>
<td>Vector fetch occurred</td>
</tr>
<tr>
<td>2</td>
<td>DWTRAP</td>
<td>R/Wc</td>
<td>0</td>
<td>DWT match occurred</td>
</tr>
<tr>
<td>1</td>
<td>BKPT</td>
<td>R/Wc</td>
<td>0</td>
<td>BKPT instruction executed</td>
</tr>
<tr>
<td>0</td>
<td>HALTED</td>
<td>R/Wc</td>
<td>0</td>
<td>Halt requested in NVIC</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Type</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>MMAR</td>
<td>R</td>
<td>—</td>
<td>Address that caused memory manage fault</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Type</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>BFAR</td>
<td>R</td>
<td>—</td>
<td>Address that caused bus fault</td>
</tr>
</tbody>
</table>
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

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 “halt” to stop the kernel.

Figure 3-2. Input halt to stop the kernel

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

3.2.3 Analysis problems

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

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
```

The `f` command 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 = 00000000, CycleCnt = 0393B1CC
R0 = 00000000, R1 = 00000000, R2 = 00000000, R3 = 00000083
R4 = 40000000, R5 = 00000004, R6 = 00000000, R7 = 00000000
R8 = 00000000, R9 = 2000000C, R10 = 00000000, R11 = 00000000
R12 = 00000000
SP<R13> = 20000068, MSP = 20000068, PSP = 20000000, R14<LR> = 0800104D
XPSR = 0100001F, APSR = 0+x0Q, EPSR = 01000000, IPSR = 01F <INTISR15>
CFPR = 00000000, CONTROL = 00, FAULTMASK = 00, BASEPRI = 00, PRIMASK = 00
```

The `h` command goto. Used to activate the kernel that is halt.

Figure 3-8. s command

```
J-Link>s
PC = 00000000, CycleCnt = 03970C14
R0 = 00000000, R1 = 00000000, R2 = 00000000, R3 = 00000083
R4 = 40000000, R5 = 00000004, R6 = 00000000, R7 = 00000000
R8 = 00000000, R9 = 2000000C, R10 = 00000000, R11 = 00000000
R12 = 00000000
SP<R13> = 20000068, MSP = 20000068, PSP = 20000000, R14<LR> = 0800104D
XPSR = 0100001F, APSR = 0+x0Q, EPSR = 01000000, IPSR = 01F <INTISR15>
CFPR = 00000000, CONTROL = 00, FAULTMASK = 00, BASEPRI = 00, PRIMASK = 00
J-Link>s
0000000C: PE E2 B # 0x04
```

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

Figure 3-9. st command

```
J-Link>st
J-Link>st
```

The `st` command show hardware status. Display the current state of Jlink.

Figure 3-10. hwinfo command

```
J-Link>hwinfo
```

The `hwinfo` command Show hardware info. Display the hardware information of Jlink.
Cortex-M3 kernel hardfault error debugging and locating method

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 0x80000000 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 00 00 00 00
J-Link>mem8 0x80000000 2
00000000 = 90 00
J-Link>mem16 0x80000000 2
00000000 = 0090 2000
J-Link>mem32 0x80000000 2
00000000 = 20000000 00000001
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 00000000
J-Link>

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

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

<table>
<thead>
<tr>
<th>Revision No.</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Initial Release</td>
<td>Apr.30, 2021</td>
</tr>
</tbody>
</table>
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.

© 2021 GigaDevice – All rights reserved