# Table of Contents

Table of Contents ........................................................................................................ 2  
List of Figures ............................................................................................................. 3  
List of Tables ............................................................................................................. 4  
1. Introduction ............................................................................................................. 5  
2. CmBacktrace Porting .............................................................................................. 6  
   2.1. Download CmBacktrace .................................................................................. 6  
   2.2. Add CmBacktrace source file ....................................................................... 6  
   2.3. Project configuration of different IDEs ......................................................... 7  
   2.4. CmBacktrace parameter configuration ....................................................... 8  
   2.5. Others .......................................................................................................... 9  
3. Functional test of CmBacktrace ............................................................................. 12  
4. Revision history .................................................................................................. 15
How to use CmBacktrace to track faults in GD32 Cortex-M series

List of Figures

Figure 2-1. Version information of CmBacktrace ................................................................. 6
Figure 2-2. Flowchart of MPU ................................................................................................. 6
Figure 2-3. Project configuration of Keil and IAR ................................................................. 7
Figure 2-4. Comment the original HardFault Handler function .............................................. 7
Figure 2-5. Project configuration of Keil .................................................................................. 8
Figure 2-6. Project configuration of IAR .................................................................................. 8
Figure 2-7. Configuration of cmb_def.h .................................................................................. 9
Figure 2-8. Conditional compilation of cmb_def.h ................................................................. 10
Figure 2-9. Explanation of ARM Compiler Version 6 on __ARMCC_VERSION ....................... 10
Figure 2-10. Modification of compiler using ARM compiler version 6 .................................. 11
Figure 3-1. Fault_test_by_unalign error report generated under Keil ................................. 13
Figure 3-2. According to the axf file generated by Keil, use the addr2line tool to obtain the function call stack information ......................................................................................... 13
Figure 3-3. Fault_test_by_unalign error report generated under IAR ................................. 14
Figure 3-4. According to the axf file generated by IAR, use the addr2line tool to obtain the function call stack information ......................................................................................... 14
List of Tables

Table 2-1. Configuration of CmBacktrace parameter ................................................................. 8
Table 3-1. fault_test_by_unalign ................................................................................................. 12
Table 3-2. fault_test_by_div0 ...................................................................................................... 12
Table 4-1. Revision history ......................................................................................................... 15
1. **Introduction**

CmBacktrace (Cortex Microcontroller Backtrace) is an open source library that automatically tracks and locates error codes for ARM Cortex-M series MCUs, and automatically analyzes the causes of errors. The main features are as follows:

- **Supported errors include:**
  - Assert
  - Fault (Hard Fault, Memory Management Fault, Bus Fault, Usage Fault, Debug Fault)

- **Failure reason** Automatic diagnosis: When a failure occurs, the cause of the failure can be automatically analyzed, and the code location of the failure can be located, without the need to manually analyze the complicated fault registers; -Output the function call stack of the error site (need to cooperate with the addr2line tool for precise positioning), restore the field information when the error occurred, and locate the problem code location and logic more quickly and accurately. You can also use the library under normal conditions to get the current function call stack;

- **Support bare metal and the following operating system platforms:**
  - RT-Thread
  - UCOS
  - FreeRTOS (source code needs to be modified)

- According to the error scene status, output the corresponding thread stack or C main stack;

- The fault diagnosis information supports multiple languages (currently: Simplified Chinese, English);

- Adapt to Cortex-M0/M3/M4/M7 MCU;

- Support IAR, KEIL, GCC compiler;

This document describes how to port CmBacktrace to the GD32 project.
2. Porting CmBacktrace

2.1. Download CmBacktrace

The CmBacktrace transplantation platform introduced in this document is the GD32E507Z-EVAL development board. The IDE platforms ported by CmBacktrace are KEIL5 and IAR.

CmBacktrace source code can be downloaded from [https://github.com/armink/CmBacktrace](https://github.com/armink/CmBacktrace). The currently tested CmBacktrace software version is 1.4.0, as shown in the figure below.

![Version information of CmBacktrace](image)

**Figure 2-1. Version information of CmBacktrace**

```c
/* library.software.version.number */
#define CMB_SW_VERSION ................."1.4.0"
```

![Flowchart of MPU](image)

**Figure 2-2. Flowchart of MPU**

2.2. Add CmBacktrace source file

The migration method introduced in this article is based on the 01_GPIO_Running_LED project in GD32E507Z_EVAL_Demo_Suites. First, copy the CmBacktrace\cm_backtrace library file to the 01_GPIO_Running_LED file. Then open the project and add cm_backtrace.c and cmb_fault.S to the project.
How to use CmBacktrace to track faults in GD32 Cortex-M series

Since cmb_fault.S will use HardFault_Handler, the original HardFault_Handler function should be commented.

Figure 2-4. Comment the original HardFault_Handler function

```c
/*
 * \brief: this function handles HardFault exception
 * \param[in]: none
 * \param[out]: none
 * \retval: none
 *
 * //void HardFault_Handler(void)
 * //{
 *   //.../\* if Hard Fault exception occurs, go to infinite loop */
 *   //...\while (1){
 *   //...
 * }  
 */
```

2.3. Project configuration of different IDEs

CmBacktrace must be configured to support the C99 standard when using the KEIL5 compiler. The engineering configuration of Keil and IAR is shown in the figure below.
2.4. **CmBacktrace parameter configuration**

The configuration options for different platforms and scenarios are defined in `cmb_def.h`.

<table>
<thead>
<tr>
<th>Configuration Name</th>
<th>Function description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cmb_println(...)</code></td>
<td>Error and diagnostic information output</td>
<td>Must be configured</td>
</tr>
<tr>
<td><code>CMB_USING_BARE_METAL_PLATFORM</code></td>
<td>Whether it is used on a bare metal platform</td>
<td>Define this macro if it is used</td>
</tr>
<tr>
<td><code>CMB_USING_OS_PLATFORM</code></td>
<td>Whether it is used on the operating system platform</td>
<td>Operating system and bare metal must choose one of two</td>
</tr>
</tbody>
</table>
How to use CmBacktrace to track faults in GD32 Cortex-M series

<table>
<thead>
<tr>
<th>Configuration Name</th>
<th>Function description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMB_OS_PLATFORM_TYPE</td>
<td>Operating System Platform</td>
<td>RTT/uCOS/FREERTOS</td>
</tr>
<tr>
<td>CMB_CPU_PLATFORM_TYPE</td>
<td>Operating System Platform</td>
<td>M0/M3/M4/M7/M33</td>
</tr>
<tr>
<td>CMB_USING_DUMP_STACK_INFO</td>
<td>Whether to use Dump stack function</td>
<td>Use to define this macro</td>
</tr>
<tr>
<td>CMB_PRINT_LANGUAGE</td>
<td>Language when outputting information</td>
<td>CHINESE/ENGLISH</td>
</tr>
</tbody>
</table>

The configuration in the CmBacktrace GD32E507Z project is shown in the figure below.

**Figure 2-7. Configuration of cmb_def.h**

```c
#ifndef _CMD_CFG_H_
#define _CMD_CFG_H_

/* print line, must config by user */
#define cmbr_println(...) printf(__VA_ARGS__)\printf("
"")

/* enable bare metal(no OS) platform */
#define CMB_USING_BARE_METAL_PLATFORM

/* cpu platform type, must config by user */
#define CMB_CPU_PLATFORM_TYPE .......... CMB_CPU_ARM_CORTEX_M33
/* enable dump stack information */
#define CMB_USING_DUMP_STACK_INFO
/* language of print information */
#define CMB_PRINT_LANGUAGE ................ CMB_PRINT_LANGUAGE_ENGLISH
#endif /* _CMD_CFG_H_ */
```

### 2.5. Others

The cmb_def.h in the original code uses the __CC_ARM macro to distinguish which IDE environment it is. For the ARM Compiler Version 6 compiler, the macro used is __ARMCC_VERSION, as shown in the figure below.
Figure 2-8. Conditional compilation of cmb_def.h

```c
#if defined(__CC_ARM) || defined(__CLANG_ARM)
  /* C stack block name, default is STACK */
#else if defined(CMB_CSTACK_BLOCK_NAME)
  #define CMB_CSTACK_BLOCK_NAME
  /* C stack block name, default is 'CSTACK' */
#endif

#if defined(CMB_CODE_SECTION_NAME)
  #define CMB_CODE_SECTION_NAME STACK
#endif

#define CMB_CODE_SECTION_NAME ER_IROM1

#endif

#endif
#else
define CMB_CSTACK_BLOCK_NAME
#endif

#if defined(CMB_CODE_SECTION_NAME)
  #define CMB_CODE_SECTION_NAME CSTACK"
#endif

#if defined(CMB_CODE_SECTION_NAME)
  #define CMB_CODE_SECTION_NAME "text"
#endif

#endif

```

Figure 2-9. Explanation of ARM Compiler Version 6 on __ARMCC_VERSION

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>When defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>__sizeln</td>
<td>-</td>
<td>Always defined for the ARM compiler, even when you specify the --sizeln option. See also __ARMCC_VERSION.</td>
</tr>
</tbody>
</table>
| __ARMCC_VERSION | \*        | Always defined. It is a decimal number, and is guaranteed to increase between releases. The format is major.minor.build where:  
|             | \*        | - major version  
|             | \*        | - minor version  
|             | \*        | - build is the build number.                                               |

Therefore, the ARM Compiler Version 6 compiler is used, some modifications should be made as follows.
Figure 2-10. Modification of compiler using ARM compiler version 6

```c
#if defined (__ARMCC_VERSION) && (__ARMCC_VERSION >= 6010050)
    /* C stack block name, default is STACK */
    ifndef CMB_CSTACK_BLOCK_NAME
    define CMB_CSTACK_BLOCK_NAME STACK
    endif
    /* code section name, default is ER_IROM1 */
    ifndef CMB_CODE_SECTION_NAME
    define CMB_CODE_SECTION_NAME ER_IROM1
    endif
    #elif defined ( _ICCARM )
    /* C stack block name, default is 'CSTACK' */
    ifndef CMB_CSTACK_BLOCK_NAME
    define CMB_CSTACK_BLOCK_NAME "CSTACK"
    endif
    /* code section name, default is '.text' */
    ifndef CMB_CODE_SECTION_NAME
    define CMB_CODE_SECTION_NAME "".text"
    endif
    #elif defined ( _GNUC )
```
3. **Functional test of CmBacktrace**

This chapter introduces HardFault caused by misalignment and division by zero errors, which are captured by CmBacktrace and printed through the serial port, as shown below.

**Table 3-1. fault_test_by_unalign**

```c
void fault_test_by_unalign(void) {
    volatile int * SCB_CCR = (volatile int *) 0xE000ED14; // SCB->CCR
    volatile int * p;
    volatile int value;

    "SCB_CCR |= (1 << 3); /* bit3: UNALIGN_TRP. */"

    p = (int *) 0x00;
    value = "p;"
    printf("addr:0x%02X value:0x%08X\n", (int) p, value);

    p = (int *) 0x04;
    value = "p;"
    printf("addr:0x%02X value:0x%08X\n", (int) p, value);

    p = (int *) 0x03;
    value = "p;"
    printf("addr:0x%02X value:0x%08X\n", (int) p, value);
}
```

**Table 3-2. fault_test_by_div0**

```c
void fault_test_by_div0(void) {
    volatile int * SCB_CCR = (volatile int *) 0xE000ED14; // SCB->CCR
    int x, y, z;

    "SCB_CCR |= (1 << 4); /* bit4: DIV_0_TRP. */"

    x = 10;
    y = 0;
    z = x / y;
    printf("z:%d\n", z);
}
```

According to the specific operating system of the computer, the addr2line.exe stored in the tools folder of CmBacktrace can be directly copied to C:\\Windows, or the tools folder path of the CmBacktrace warehouse can be added to the environment variable path. This can ensure that the command line tool can use the addr2line command normally.

The fault_test_by_unalign error report generated under Keil and the result printed by
How to use CmBacktrace to track faults in GD32 Cortex-M series

addr2line are as follows.

Figure 3-1. Fault_test_by_unalign error report generated under Keil

| Firmware name: CmBacktrace, hardware version: V1.0.0, software version: V0.1.0 |
| Fault on interrupt or bare metal (no OS) environment |
| ===== Thread stack information ===== |
| addr: 20000928 | data: 20000e84 |
| addr: 2000092c | data: 0000305d |
| addr: 20000930 | data: 0000004f |
| addr: 20000934 | data: 0000010c |
| addr: 20000938 | data: 00000034 |
| addr: 2000093c | data: 00001c1f |
| addr: 20000940 | data: 00000003 |
| addr: 20000944 | data: 000c0d14 |
| addr: 20000948 | data: 00000000 |
| addr: 2000094c | data: 0801a8f5 |
| addr: 20000950 | data: 00000000 |
| addr: 20000954 | data: 00000000 |
| addr: 2000095c | data: 00000000 |
| addr: 20000960 | data: 00000000 |
| addr: 20000964 | data: 00000000 |
| addr: 20000968 | data: 00000000 |
| addr: 2000096c | data: 00000079 |

======================= Registers information ========================
K0 : 0000000c K1 : 00000003 E2 : 00000000 E3 : 40d13800
K12 : 00000001 LK : 0b001011 KC : 0000000c PSE : 23000000

Usage fault is caused by indicates that an unaligned access fault has taken place.
Slew more call stack info by run: addr2line -e CmBacktrace. axf -a -f 080017c4 08001810 08001810 08001a10 080001c0 08001af4 080010d9

Figure 3-2. According to the axf file generated by Keil, use the addr2line tool to obtain the function call stack information

```
_suites\Projects\GPTO\Running_LEDMDK-ARM\output>addr2line -e Project.axf -a -f 080017c4 08001810 08001c0 08001af4 080010d9
fault_test_by_unalign

_Suites\Projects\GPTO\Running_LEDMDK-ARM\..\main.c:119
0x80001810
fputs

_SUP2\Projects\GPTO\Running_LEDMDK-ARM\..\main.c:139
0x80001810
Reset_Handler

_SUP2\Projects\GPTO\Running_LEDMDK-ARM\..\source\startup_gd32e50x_01.c:185
0x80001af4
main

_SUP2\Projects\GPTO\Running_LEDMDK-ARM\..\main.c:25
0x8000079
$4
??
```

The fault_test_by_unalign error report generated under IAR and the result printed by addr2line are as follows.
How to use CmBacktrace to track faults in GD32 Cortex-M series

Figure 3. Fault_test_by_unalign error report generated under IAR

 addr: 0x00 value: 0x20000400
 addr: 0x04 value: 0x00000000
 Firmware name: CmBacktrace, hardware version: V1.0.0, software version: V0.1.0
 Fault on interrupt or bare metal [no OS] environment
 ======= Thread stack information ======
 addr: 200000a3b data: 00000000
 addr: 200000a3c data: 00000000
 addr: 200000a3d data: 00000000
 addr: 200000a3e data: 00000000
 addr: 200000a3f data: 00000000

================ Registers information =================
 R0 : 00000000 C R1 : 00000000 E2 : 20000000 R3 : 00000000
 R12: 000000fe LR : 000025a7 FC : 00002d5a PSR : 00000000

Usage fault is caused by indicates that an unaligned access fault has taken place,
Show more call stack info by run: addr2line -e CmBacktrace.out -a -f 08002d5a 08002d80
08002cb6 08002ffa

Figure 3-4. According to the axf file generated by IAR, use the addr2line tool to obtain
the function call stack information

_Suites\Projects\GPIO_Running_LED\Eval\GD32E50x\Exe>addr2line -e Project.out -a
 -f 08002d5a 08002d80 08002cb6 08002ffa
 0x08002d5a
 forbid
 fault_test_by_unalign
_Suites\Projects\GPIO_Running_LED\main.c:119
 0x08002d80
 for

_Suites\Projects\GPIO_Running_LED\main.c:139
 0x08002cb6
 main
_Suites\Projects\GPIO_Running_LED\main.c:165
 0x08002ffa
_call_main

???:?
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>Nov.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