AN0028

Application Note

Quickly Trace HardFaultHardler

Introduction

This application note describes how to use CmBacktrace library to quickly trace the HardFault and fix it.

Note: The codes in this application note are based on Artyer's V2.x.x BSP (board support package). Therefore, attention should be paid to the differences between the versions of BSP when in use

Applicable parts:

MCU	AT32 Family

Contents

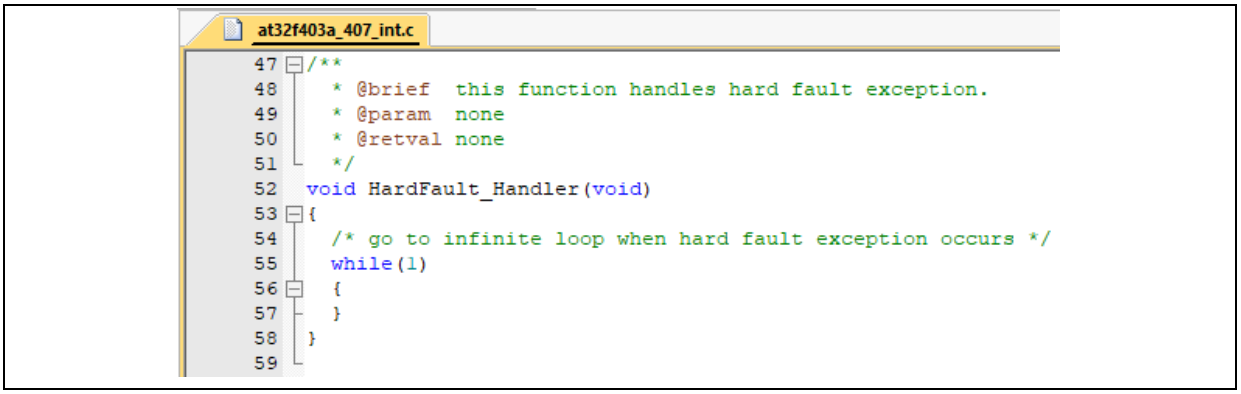
List of tables

List of figures

1 Overview

Sometimes program execution failure may occur during the use of ARM Cortex-M-based MCU (such as AT32 MCU). When we attempt to look into the cause of this problem in Debug mode through compiler, we might find that the program jumps to the HardFault_Handler function, and thus generates a HardFault.

Figure 1. HardFault_Handler function



This application note demonstrates how to quickly track and find the root cause of HardFault through the CmBacktrace-based library.

2 Causes of HardFault generation

Here are the possible factors generating HardFault.

- Data array is handled out of the boundary
- Memory overflow causes access outside the boundary
- Stack overflow causes program crash
- Interrupt handle error

Data array out of boundary

The program uses static array but value overflow occurs during dynamic parameter transfer. It is also possible that the allocated interntal memory is very low to cause program failure.

Memory overflow

Check the RAM area to confirm whether the RAM data count executed after compiling is out of the boundary. It is not recommended to make extreme value configuration to avoid error during dynamic parameter transfer of data array.

Stack overflow

This problem often occurs when using operating system code. As in operating system, the variables of tasks are allocated and placed in a stack space where the tasks applie for.

For example, the xTaskCreate function is called in FreeRTOS to create a task. This function uses the parameter usStackDepth to assign task stack. If the assigned stack size is too small or not big enough, this may cause program to enter HardFault.

Interrupt handling error

Although users enable some interrupts such as USART, TIMER, RTC, the conditions for interrupt generation are met during program execution but some of the interrupt service routine functions cannot be identified, this may lead to error.

3 How to analyze HardFault

When it comes to HardFault probem, the first step in most cases is to check the value in the LR register to determine whether the current stack you are using is MSP or PSP, find the corresponding stack pointer, and then check the content of the stack in the memory. When an error is detected, the core would place R0~R3, R12 Returnaddress, PSR and LR registers in the stack in sequence. The Return address refers to the next instruction to be executed by PC before the occurrence of the error.

However, this is a tedious process requiring the engineer to be familiar with ARM core.

The subsequent section will introduce an open force CmBacktrace library to make quick analysis of an error.

3.1 CmBacktrace library

CmBacktrace (Cortex Microcontroller Backtrace) is an open source library that is capable of automatically tracking and locating error codes for ARM Cortex-M-based MCUs, and analyzing the causes of errors.

Main features:

- Error type that can be identified

1) Assert
2) Fault (Hard Fault, Memory Management Fault, Bus Fault, Usage Fault, Debug Fault)

- Failure cause 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 needing to analyze the complicated fault registers;
- Applicable to Cortex-M0/M3/M4/M7 MCU
- Support IAR, KEIL, GCC compiler
- Support FreeRTOS, UCOSII, RT-Thread, etc

3.2 How to use MDK-based CmBacktrace

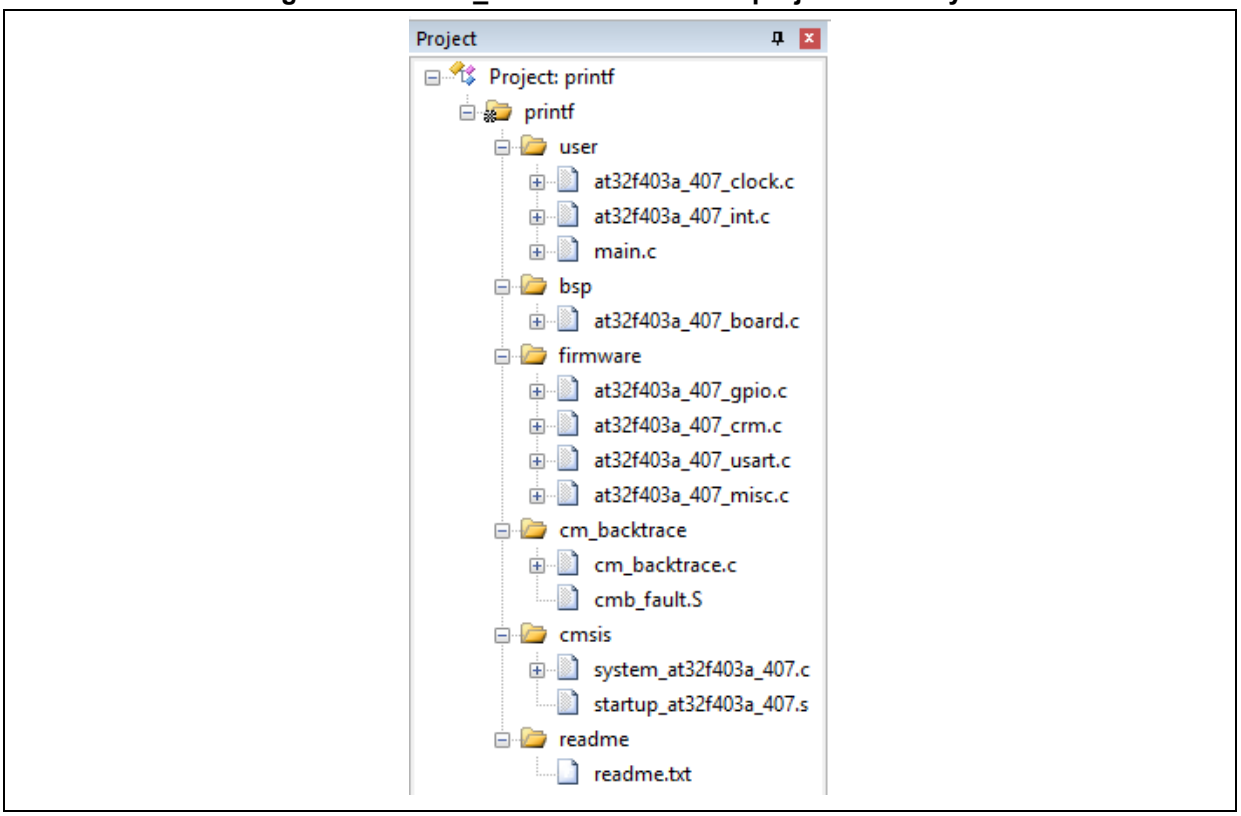
Follow the procedures below:

Step 1: Add *cm_backtrace* file to the MDK

Figure 2. cm_backtrace folder

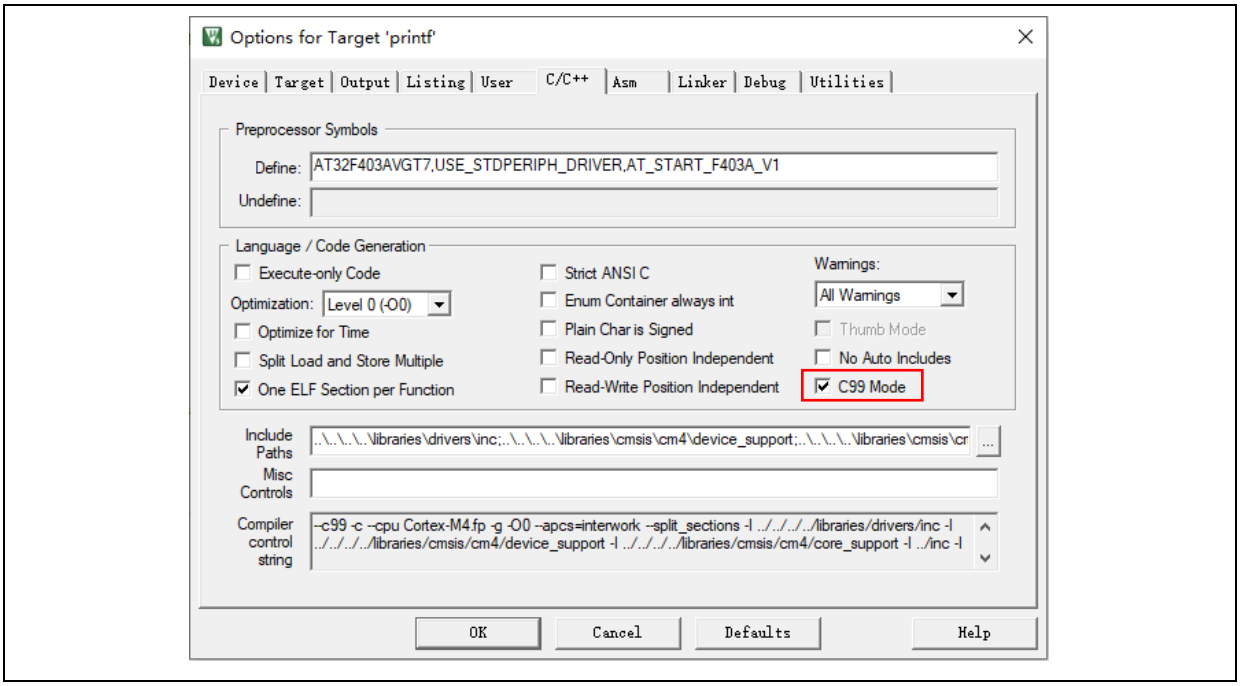
Copy the *cm_backtrace* folder and add it to the keil project directory.

Figure 3. Add cm_backtrace to the keil project directory



Step 2: Add the header file, and tick *C99 Mode*

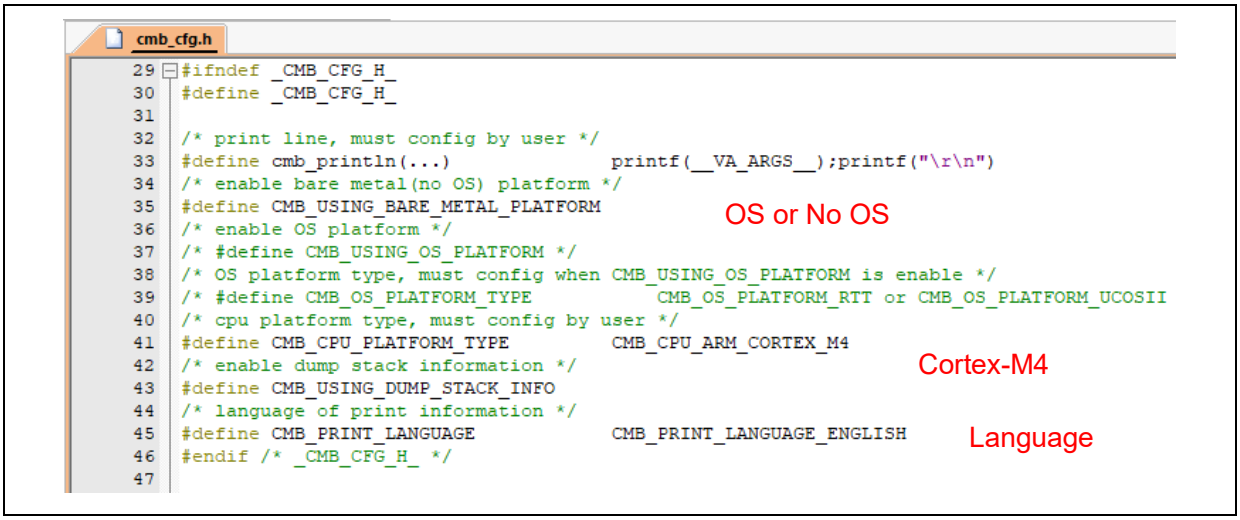
Figure 4. Configure C99 Mode and header file in Keil



Step 3: Compile and debug

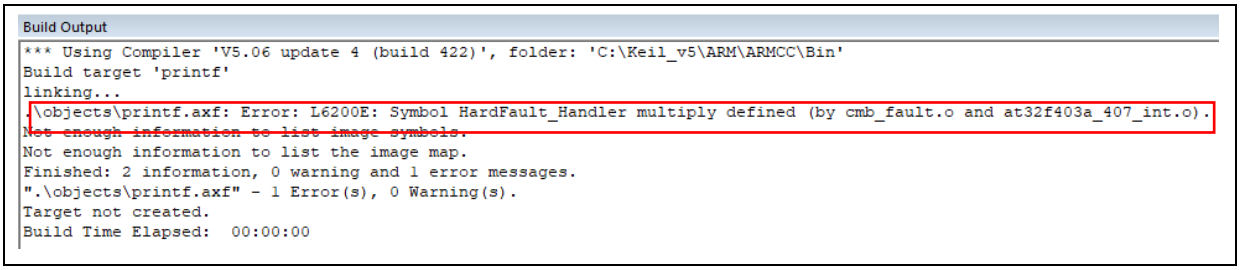
First, follow the prompts below to modify the cmb_cfg.h file.

Figure 5. Configure cmb_cfg.h file



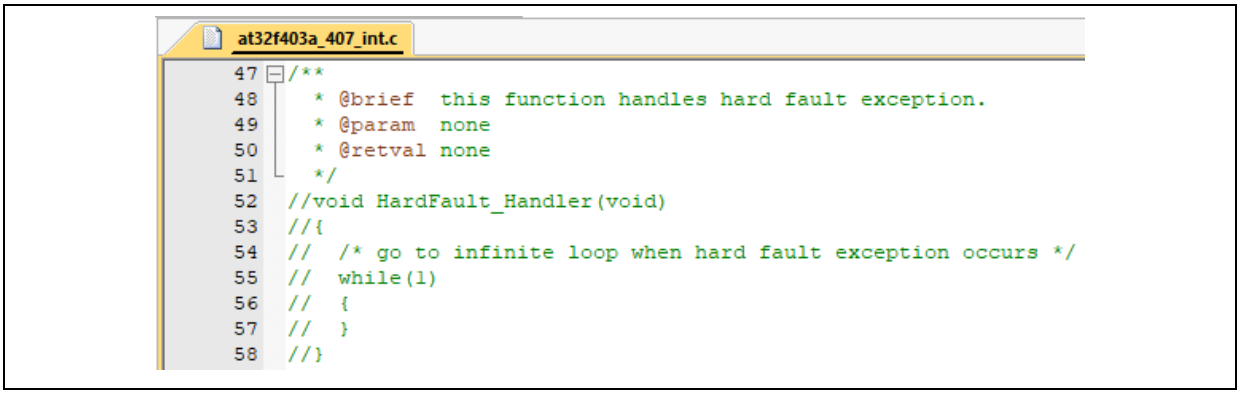
At this point, a compiling error occurs. This is because that the HardFault_Handler is not only defined in the cmb_fault.c but also in the at32f4xx_it.c. In other words, this function is repeatedlyl defined.

Figure 6. at32f4xx_it.c compiling error



Delete the HardFault_Handler function defined in the at32f4xx_it.c.

Figure 7. Delete the HardFault_Handler function



Step 4: Test and view

After successful compilation, go and test it.

Figure 8. Write the division by 0 fault function

```
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\r\n", (int) p, value);

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

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

Then call the cm_backtrace_init(); in the main function to initialize the cm_backtrace, and call the test function:

Figure 9. Call the divided-by-zero error function in main

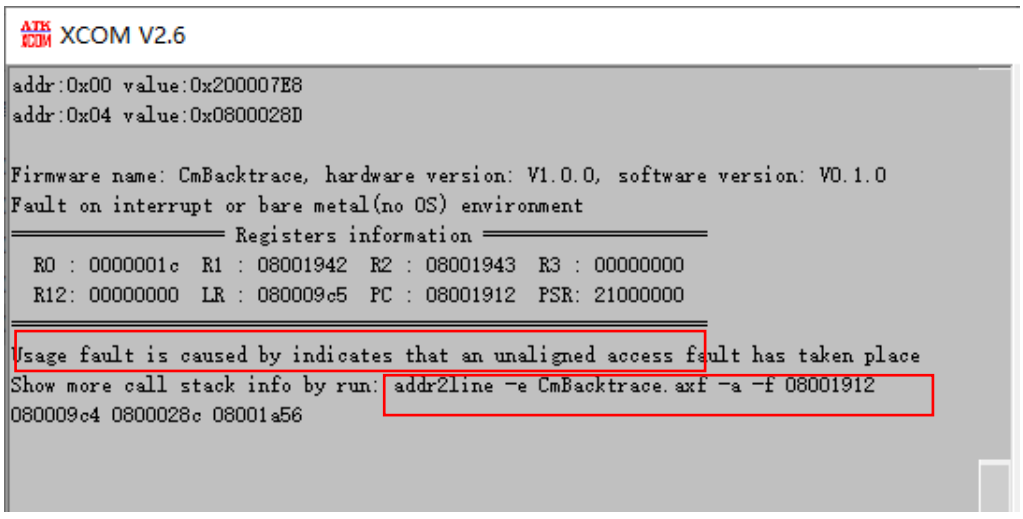
```
int main(void)
{
  system_clock_config();
  at32_board_init();
  uart_print_init(115200);

  cm_backtrace_init("CmBacktrace", HARDWARE_VERSION, SOFTWARE_VERSION);
  fault_test_by_unalign();

  while(1)
  {
    printf("usart printf counter: %u\r\n",time_cnt++);
    delay_sec(1);
  }
}
```

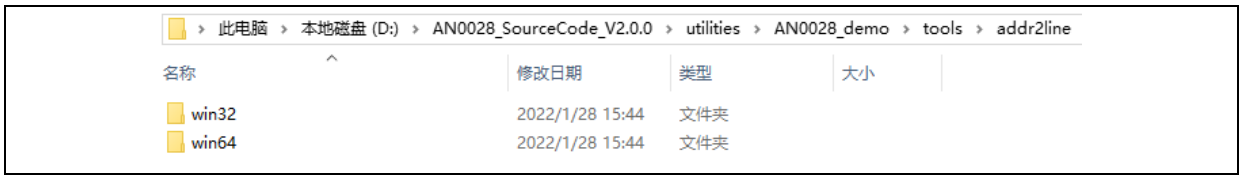
Download and run the program, the following information will be received on PC.

Figure 10. Error information is displayed



We can see that the cause of error (divided by zero) and a command line are displayed. To run this command, you need to use the addr2line.exe tool, which is located in tool folder.

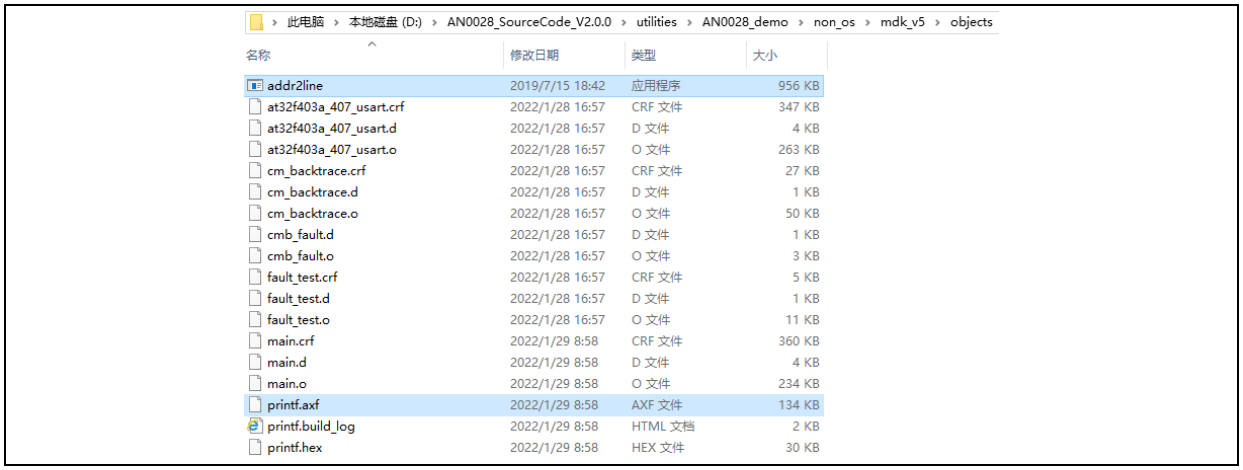
Figure 11. Locate addr2line.exe



There are two versions available for this tool, 32 bit and 64 bit. Select the desired version according to your needs and copy it to the .axf folder under keil project directory:

In this example, it is copied into the AN0028_SourceCode_V2.0.0\utilities\AN0028_demo\non_os\mdk_v5\objects

Figure 12. Copy addr2line.exe

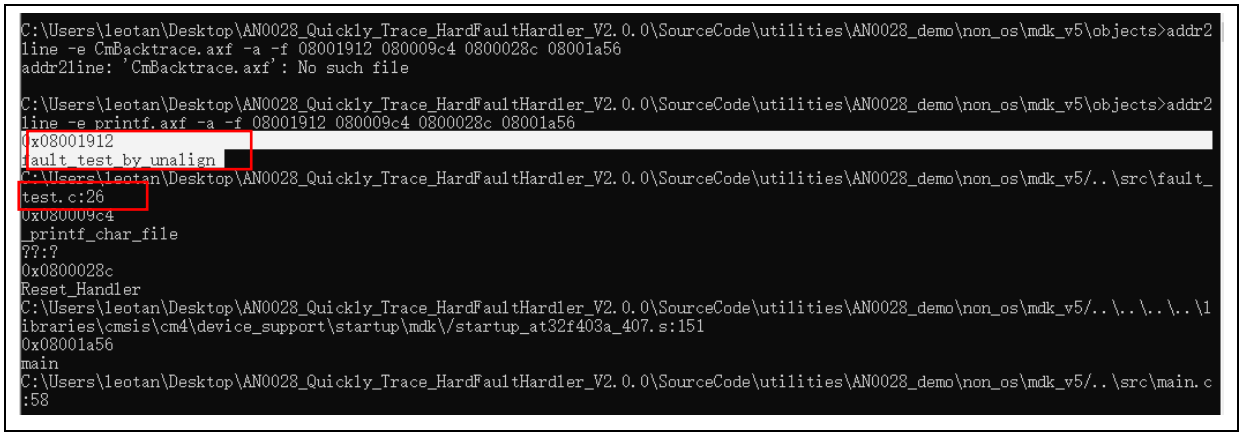


Enter the cmd window, go to the above folder location and run the command in the serial interface assistant window:

addr2line -e CmBacktrace(This name should be changed according to user's project name).axf -a -f 080019c6 08001ae9

For example, if the project name in demo is printf, then the command should be addr2line -e printf.axf -a -f 080019c6 08001ae9

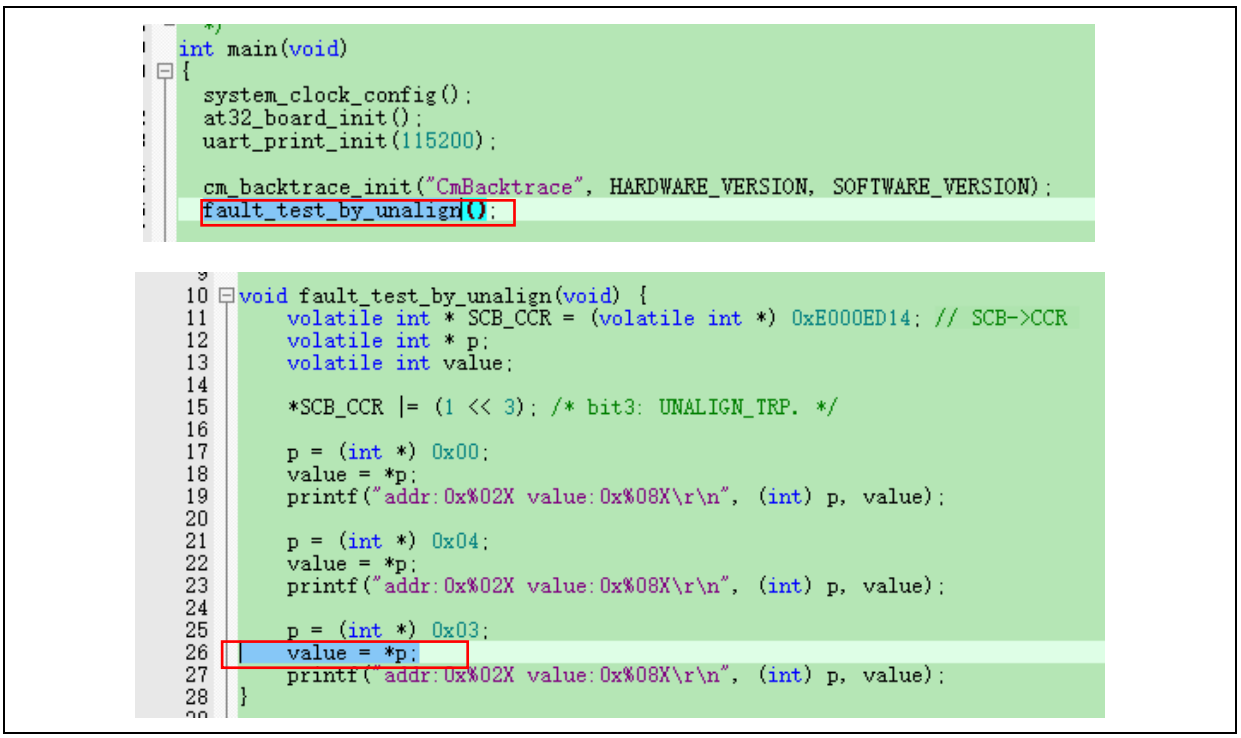
Figure 13. Call CMD to run addr2line.exe



We can see that the addr2line.exe tool has located the line number at which the error code is.

Figure 14. Check the error code area

As shown in the figure below, the error code is pointed at the No.60 line in main.c, and the No.38 in fault_test.c.



It is found that this is the very line number where the error occurs.

The CmBacktrace library can help users quickly locate HardFault error.

3.3 Example cases

Case 1: Division by 0 exception on AT32 bare machine

Project location: AN0028_SourceCode_V2.0.0\utilities\AN0028_demo\non_os

Test item: division by 0 exception on bare machine

Case 2: Division by 0 exception on FreeRTOS

Project location: AN0028_SourceCode_V2.0.0\utilities\AN0028_demo\os\freertos

Test item: Division by 0 exception on FreeRTOS. It should be noted that there are three locations marked with notes /*< Support For CmBacktrace >*/ in tasks.c in an indication of the modifications based on CmBacktrace.

Case 3: Non-aligned access error on USOCⅡ

Project location: AN0028_SourceCode_V2.0.0\utilities\AN0028_demo\os\ucosiii

Test item: Non-aligned access error on USOC Ⅱ. It should be noted that the #define OS_CFG_DBG_EN in os_cfg.h represents 1u.

4 Revision history

Table 1. Document revision history

Date	Revision	Changes
2022.2.7	2.0.0	Initial release
2024.4.8	2.0.1	Modified AC6 comiling error descriptions, and upgraded the version of cm_backtrace to V1.4.1

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Purchasers are solely responsible for the selection and use of ARTERY's products and services; ARTERY assumes no liability for purchasers' selection or use of the products and the relevant services.

No license, express or implied, to any intellectual property right is granted by ARTERY herein regardless of the existence of any previous representation in any forms. If any part of this document involves third party's products or services, it does NOT imply that ARTERY authorizes the use of the third party's products or services, or permits any of the intellectual property, or guarantees any uses of the third party's products or services or intellectual property in any way.

Except as provided in ARTERY's terms and conditions of sale for such products, ARTERY disclaims any express or implied warranty, relating to use and/or sale of the products, including but not restricted to liability or warranties relating to merchantability, fitness for a particular purpose (based on the corresponding legal situation in any unjudicial districts), or infringement of any patent, copyright, or other intellectual property right.

ARTERY's products are not designed for the following purposes, and thus not intended for the following uses: (A) Applications that have specific requirements on safety, for example: life-support applications, active implant devices, or systems that have specific requirements on product function safety; (B) Aviation applications; (C) Aerospace applications or environment; (D) Weapons, and/or (E) Other applications that may cause injuries, deaths or property damages. Since ARTERY products are not intended for the above-mentioned purposes, if purchasers apply ARTERY products to these purposes, purchasers are solely responsible for any consequences or risks caused, even if any written notice is sent to ARTERY by purchasers; in addition, purchasers are solely responsible for the compliance with all statutory and regulatory requirements regarding these uses.

Any inconsistency of the sold ARTERY products with the statement and/or technical features specification described in this document will immediately cause the invalidity of any warranty granted by ARTERY products or services stated in this document by ARTERY, and ARTERY disclaims any responsibility in any form.