AN0028

Application Note

Quickly Trace HardFaultHardler

Introduction

This application note is written to describe how to use CmBacktrace library to quickly trace and locate the causes of HardFault.

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

Applicable parts:

MCU	AT32F series

Contents

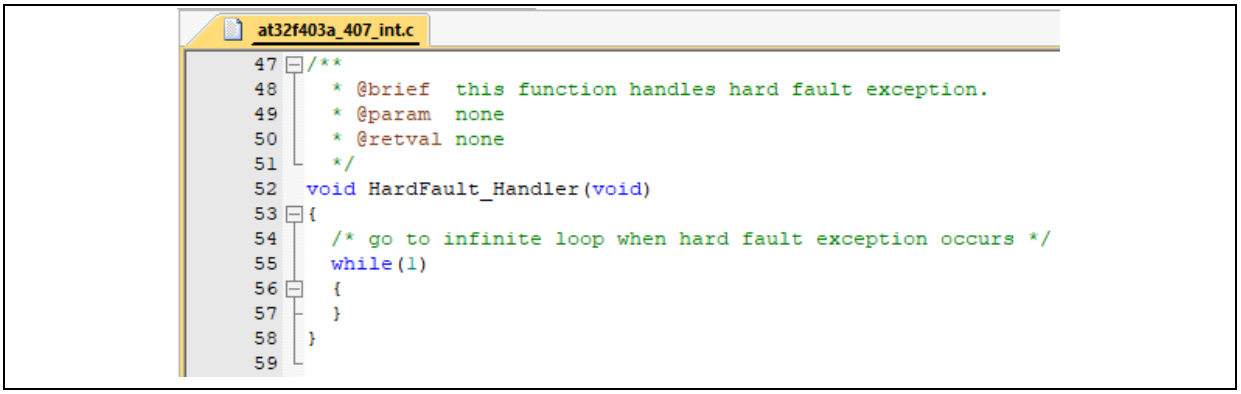
List of tables

List of figures

1 Overview

Sometimes we may be encountered with program running exception during the use of ARM Cortex-M MCU (such as AT32 MCU). When making an attempt to locate the cause of the error through a compiler in debug mode, we may find that the program jumps to HardFault_Handler function, generating a HardFault.

Figure 1. HardFault_Handler function



This application note gives an introduction of CmBacktrace-based library that quickly tracks and locates the causes of HardFault.

2 HardFault causes

Common causes that could trigger HardFault are shown as follows:

- Array out of bounds
- Memory overflow
- Stack overflow and program crash
- Interrupt handling error

Array out of bounds

The value range of an array is defined in the program. Array access is only allowed within the defined range, otherwise, array bound error will occur.

Memory overflow

Check the RAM data size executed after compiling and whether it is out of bounds.

Stack overflow

This issue often occurs during the use of operating system code. In operating systems, the variables of tasks are allocated and placed in a stack space where the tasks applie for.

One of the examples is to create a task by calling xTaskCreate function in the FreeRTOS. The parameter usStackDepth in this function is used to define the size of a task stack. If it is configured too small, it will lead to insufficient stack space, triggering HardFault.

Interrupt handling error

Though some interrupts such as USART, TIMER, and RTC have been enabled in the program, the interrupt service function cannot be detected during the program running even though the interrupt conditions are met.

3 How to analyze HardFault

What we usually do in the event of a HardFault: first check the value in the LR register to confirm whether the current stack used is MSP or PSP, find out the pointer of the corresponding stack and view the contents in the stack. When an error occurs, the core puts the R0~R3, R12 Returnaddress, PSR and LR registers in the stack in sequence, so the Return address is the next instruction to be executed by PC before the occurrence of an exception.

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

The subsequent sections focus on CmBacktrace, an open source library used for quick analysis of an error.

3.1 CmBacktrace introduction

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

- Supported errors include:
  1) Assert
  2) 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 analyze the complicated fault registers;
- Applicable to Cortex-M0/M3/M4/M7 MCU
- Support IAR, KEIL, GCC compiler
- Support FreeRTOS, UCOSII, RT-Thread and so on.

3.2 How to use MDK-based CmBacktrace

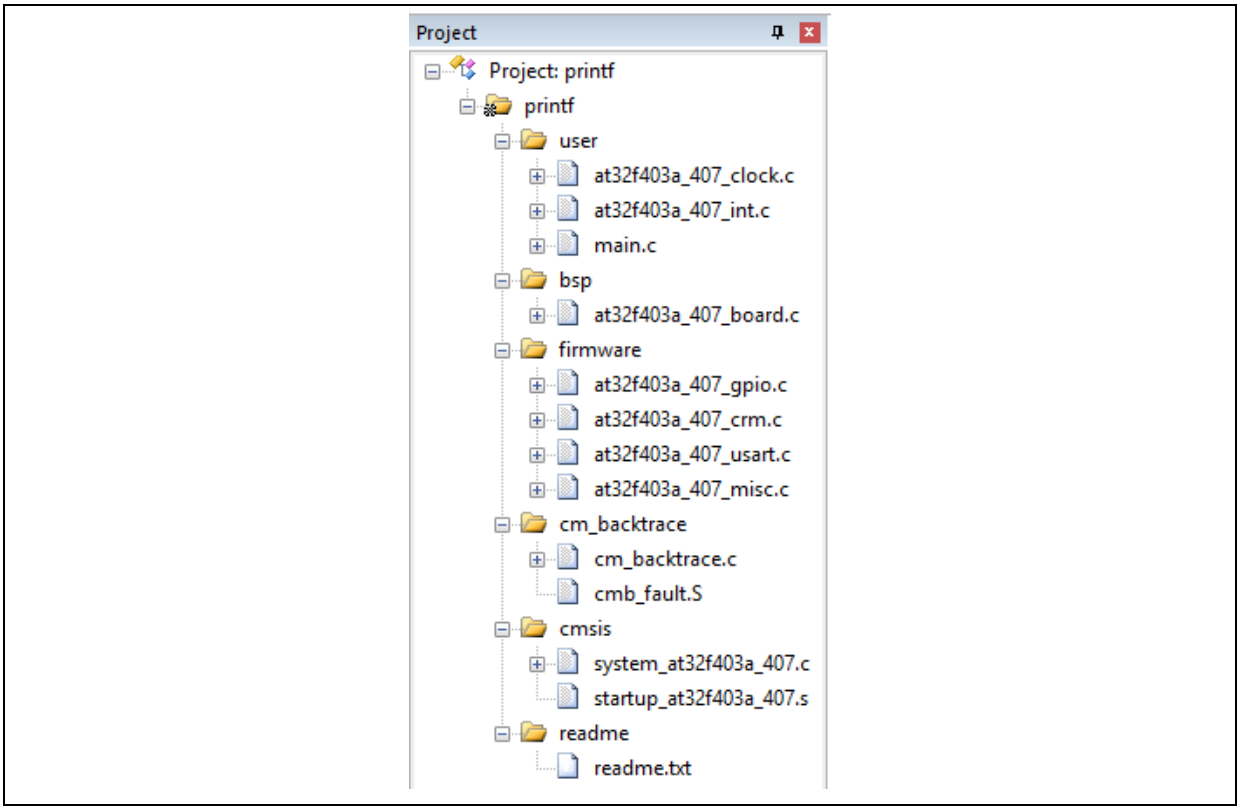
MDK-based porting instructions are as follows:

Step 1: Add *cm_backtrace* file to the MDK

Figure 2. cm_backtrace folder

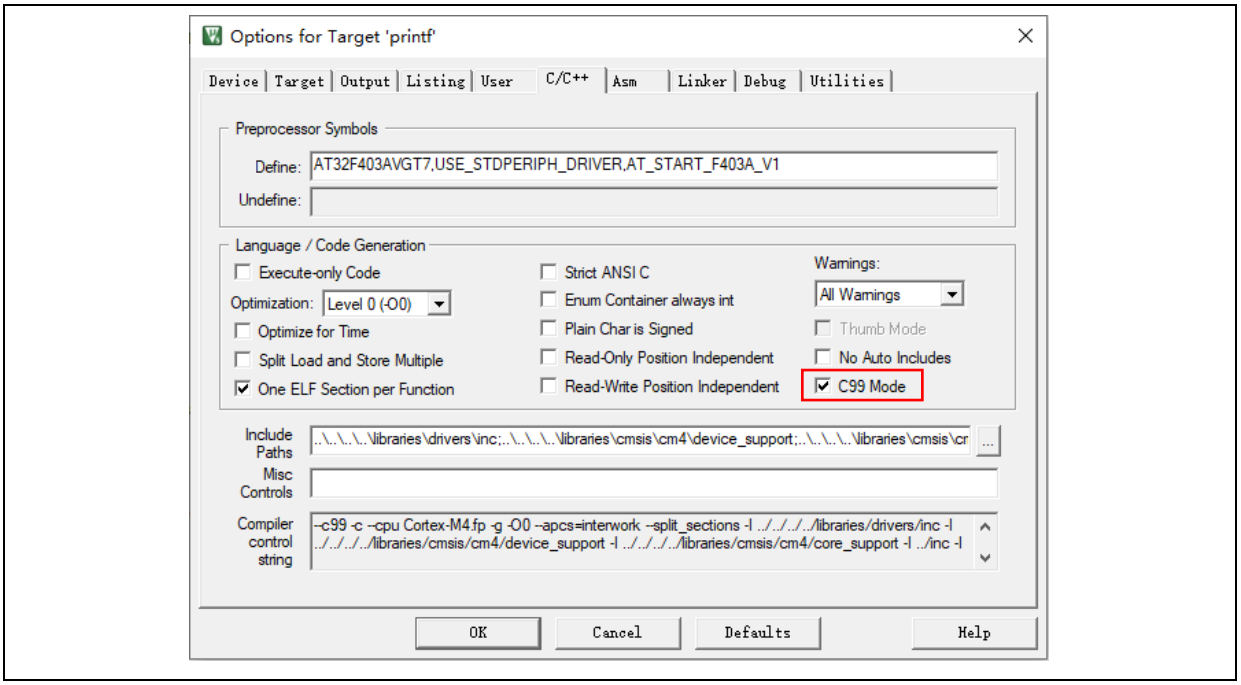
Copy the *cm_backtrace* folder and add it to the keil project under our project directory.

Figure 3. Add cm_backtrace to the keil project



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

cmb_cfg.h

```
29 #ifndef _CMB_CFG_H_
30 #define _CMB_CFG_H_
31
32 /* print line, must config by user */
33 #define cmb_println(...)              printf(__VA_ARGS__);printf("\r\n")
34 /* enable bare metal(no OS) platform */
35 #define CMB_USING_BARE_METAL_PLATFORM
36 /* enable OS platform */
37 /* #define CMB_USING_OS_PLATFORM */
38 /* OS platform type, must config when CMB_USING_OS_PLATFORM is enable */
39 /* #define CMB_OS_PLATFORM_TYPE          CMB_OS_PLATFORM_RTT or CMB_OS_PLATFORM_UCOSII
40 /* cpu platform type, must config by user */
41 #define CMB_CPU_PLATFORM_TYPE         CMB_CPU_ARM_CORTEX_M4
42 /* enable dump stack information */
43 #define CMB_USING_DUMP_STACK_INFO
44 /* language of print information */
45 #define CMB_PRINT_LANGUAGE            CMB_PRINT_LANGUAGE_ENGLISH
46 #endif /* _CMB_CFG_H_ */
47
```

OS or No OS

Cortex-M4

Language

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

Build Output

```
*** Using Compiler 'V5.06 update 4 (build 422)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'printf'
linking...
.\objects\printf.axf: Error: L6200E: Symbol HardFault_Handler multiply defined (by cmb_fault.o and at32f403a_407_int.o).
Not enough information to list image symbols.
Not enough information to list the image map.
Finished: 2 information, 0 warning and 1 error messages.
".\objects\printf.axf" - 1 Error(s), 0 Warning(s).
Target not created.
Build Time Elapsed:  00:00:00
```

Delete the HardFault_Handler function defined in the at32f4xx_it.c.

Figure 7. Delete the HardFault_Handler function

at32f403a_407_int.c

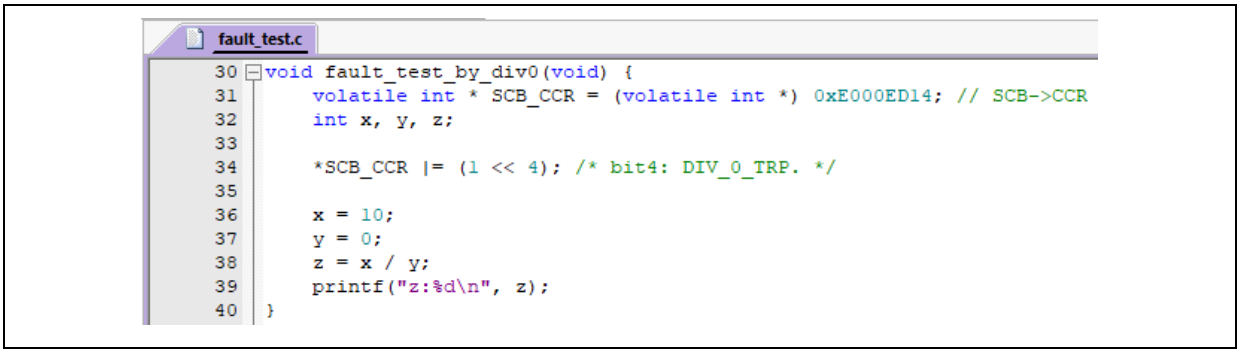
```
47 /**
48   * @brief  this function handles hard fault exception.
49   * @param  none
50   * @retval none
51   */
52 //void HardFault_Handler(void)
53 //{
54 //  /* go to infinite loop when hard fault exception occurs */
55 //  while(1)
56 //  {
57 //  }
58 //}
```

Step 4: Test and view

It can be compiled successfully at this time. Let's take a look at how the CmBacktrace works.

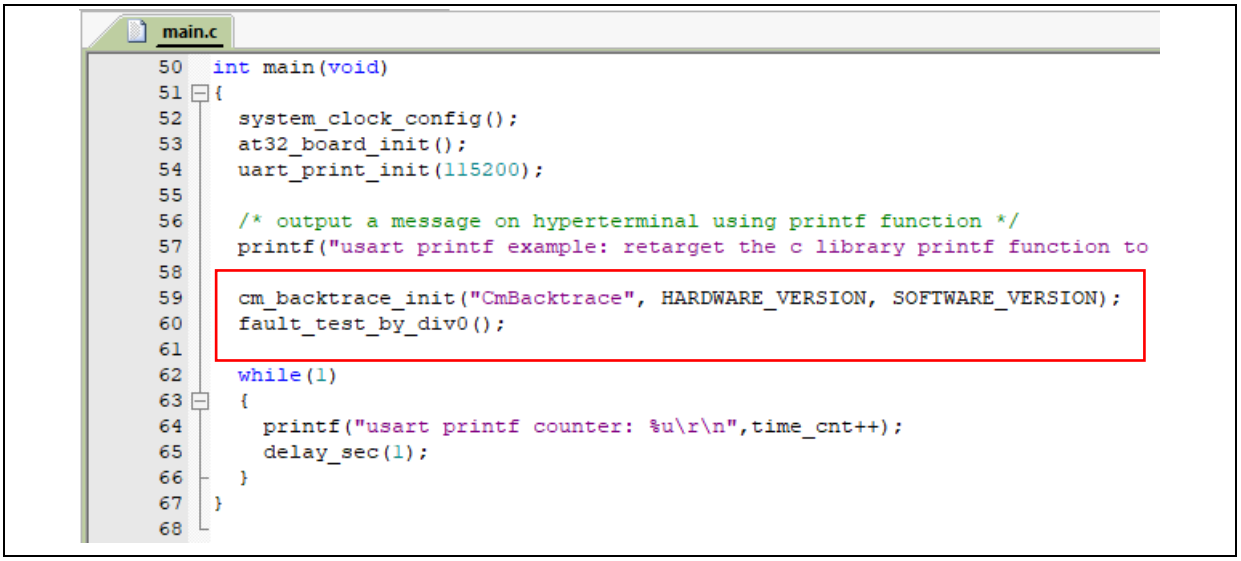
Test function is as follows:

Figure 8. Write the division by 0 fault function



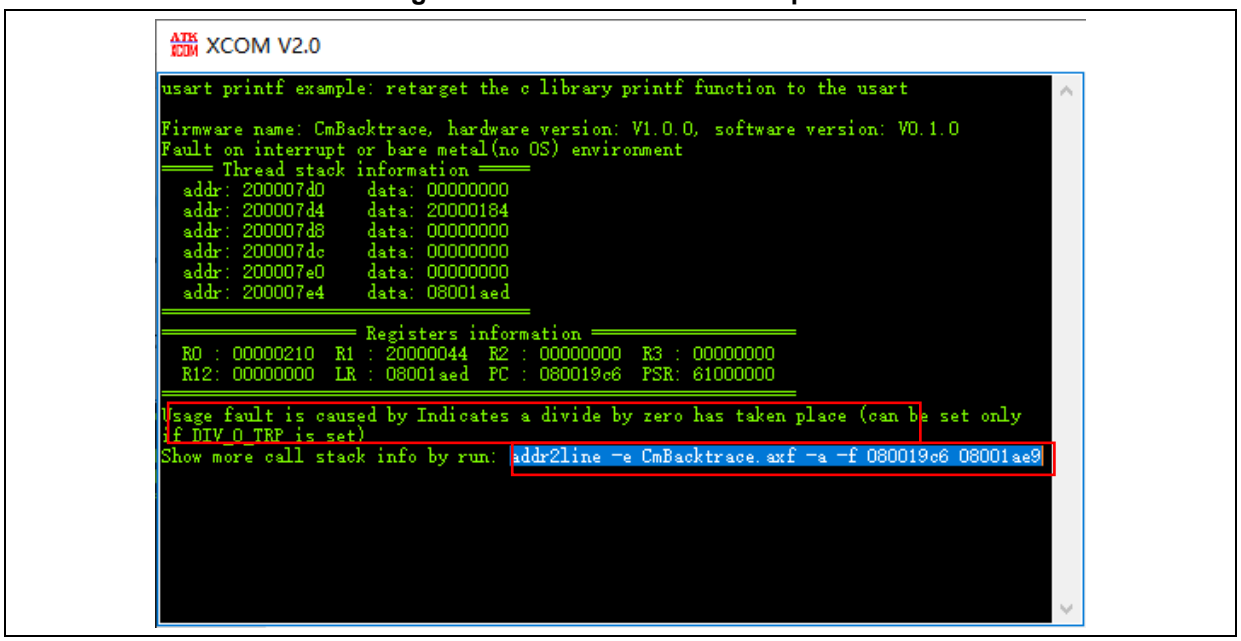
Then call the cm_backtrace_init(); in the main function to initialize the cm_backtrace, and call the test function:

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



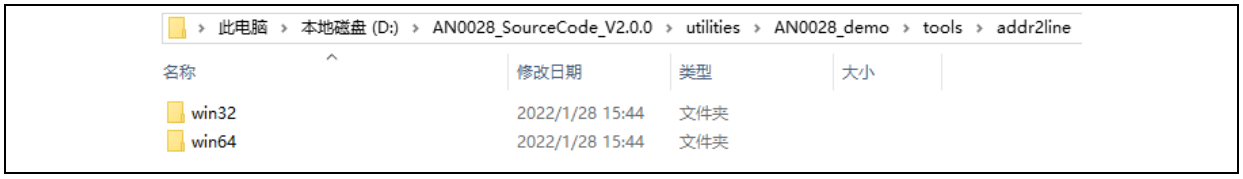
Download and run the program

Figure 10. Error information output



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

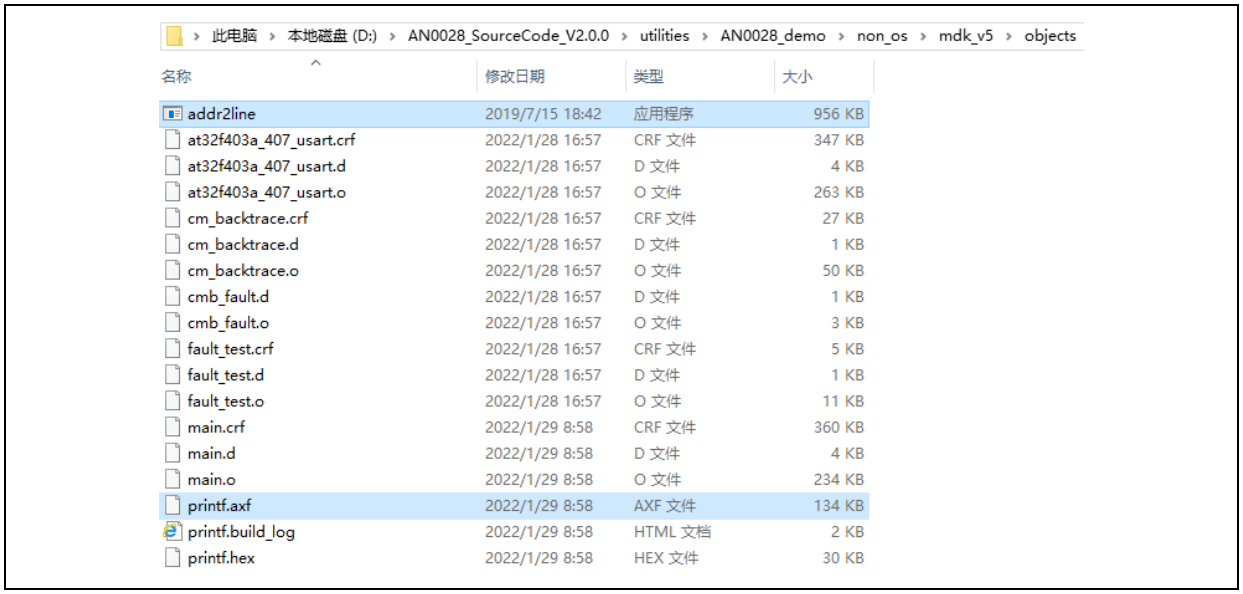
Figure 11. Locate addr2line.exe



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

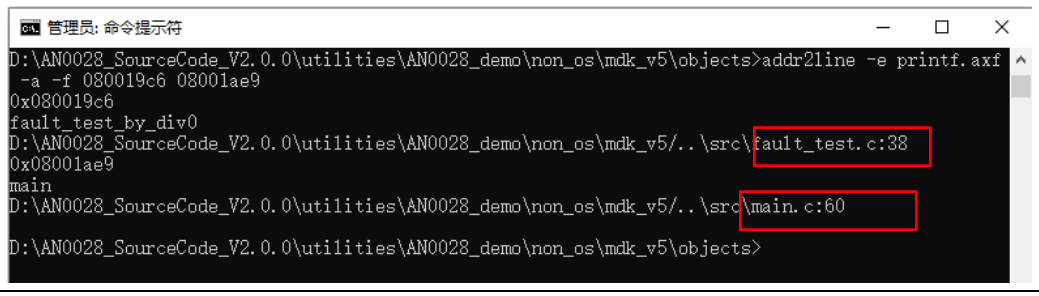


Enter the cmd window, go to the above folder location and run the command in the serial interface assistant:

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

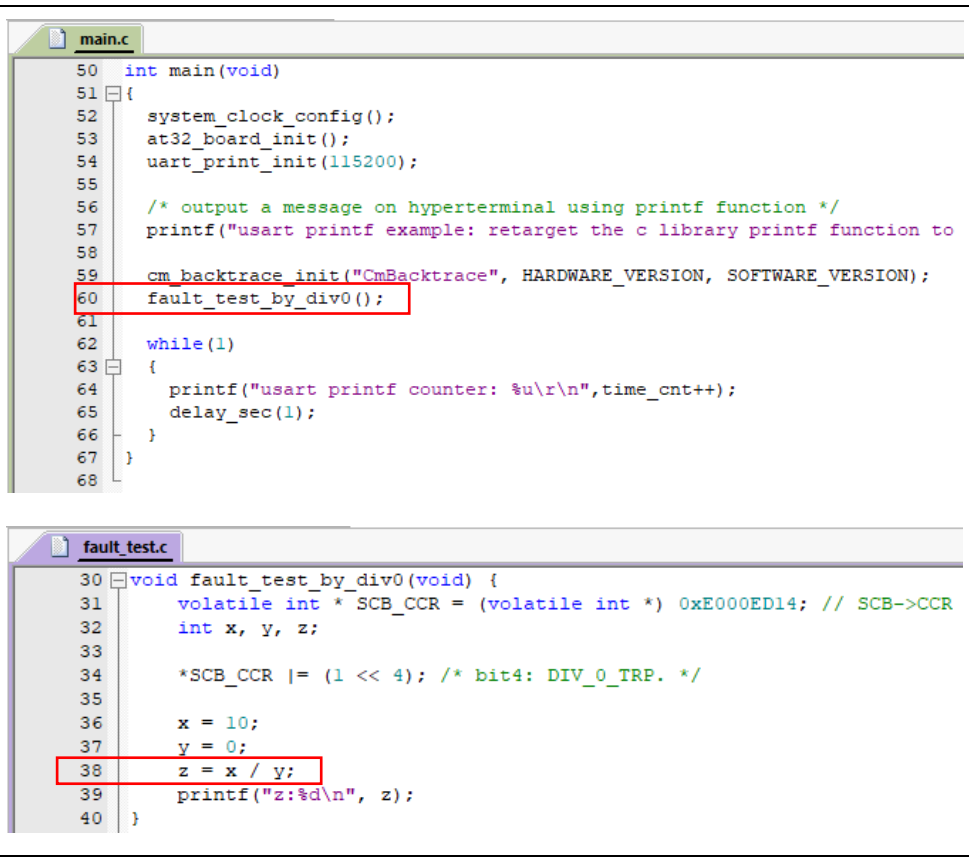
Figure 13. Call CMD to run addr2line.exe



We can see that the addr2line.exe tool locates 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 and other errors.

3.3 Demo

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 with notes /*< Support For CmBacktrace >*/ in tasks.c, which indicates 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 USOCⅡ. 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

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Purchasers are solely responsible for the selection and use of ARTERY's products and services, and ARTERY assumes no liability whatsoever relating to the choice, selection or use of the ARTERY products and services described herein.

No license, express or implied, to any intellectual property rights is granted under this document. If any part of this document deals with any third party products or services, it shall not be deemed a license grant by ARTERY for the use of such third party products or services, or any intellectual property contained therein, or considered as a warranty regarding the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

Unless otherwise specified in ARTERY's terms and conditions of sale, ARTERY provides no warranties, express or implied, regarding the use and/or sale of ARTERY products, including but not limited to any implied warranties of merchantability, fitness for a particular purpose (and their equivalents under the laws of any jurisdiction), or infringement of any patent, copyright or other intellectual property right.

Purchasers hereby agrees that ARTERY's products are not designed or authorized for use in: (A) any application with special requirements of safety such as life support and active implantable device, or system with functional safety requirements; (B) any air craft application; (C) any automotive application or environment; (D) any space application or environment, and/or (E) any weapon application. Purchasers' unauthorized use of them in the aforementioned applications, even if with a written notice, is solely at purchasers' risk, and is solely responsible for meeting all legal and regulatory requirement in such use.

Resale of ARTERY products with provisions different from the statements and/or technical features stated in this document shall immediately void any warranty grant by ARTERY for ARTERY products or services described herein and shall not create or expand in any manner whatsoever, any liability of ARTERY.