如何调试 Cortx-M 的 HardFault 错误

介绍

本技术说明目的是展示如何使用 IAR Embedded Workbench for ARM 调试 HardFault 错误。

HardFault 是指其他异常处理机制都无法处理的各类 fault。通常，HardFault 用于不可恢复的系统故障。

下面的几个示例描述了不同的 fault 场景。

Example 1：芯片超频

在本例中，Cortex-M3 板上的 CPU 时钟被设置为非常高的频率，这通常会导致 HardFault 异常随机地出现在有效的指令位置。

Call Stack 窗口中可以看到 HardFault 异常发生时执行了哪一行代码。

在 Register 窗口中 NVIC:CFSR(Configurable Fault Status Register)显示发生了不精确的数据访问错误。IMPRECISERR = 1 即表示不精确数据访问错误发生。

由于错误不精确，因此不可能看到违规数据访问的地址。在本例中，很难找到问题的实际原因是它与以不正确的 CPU 频率运行有关。请看下面的图 1:

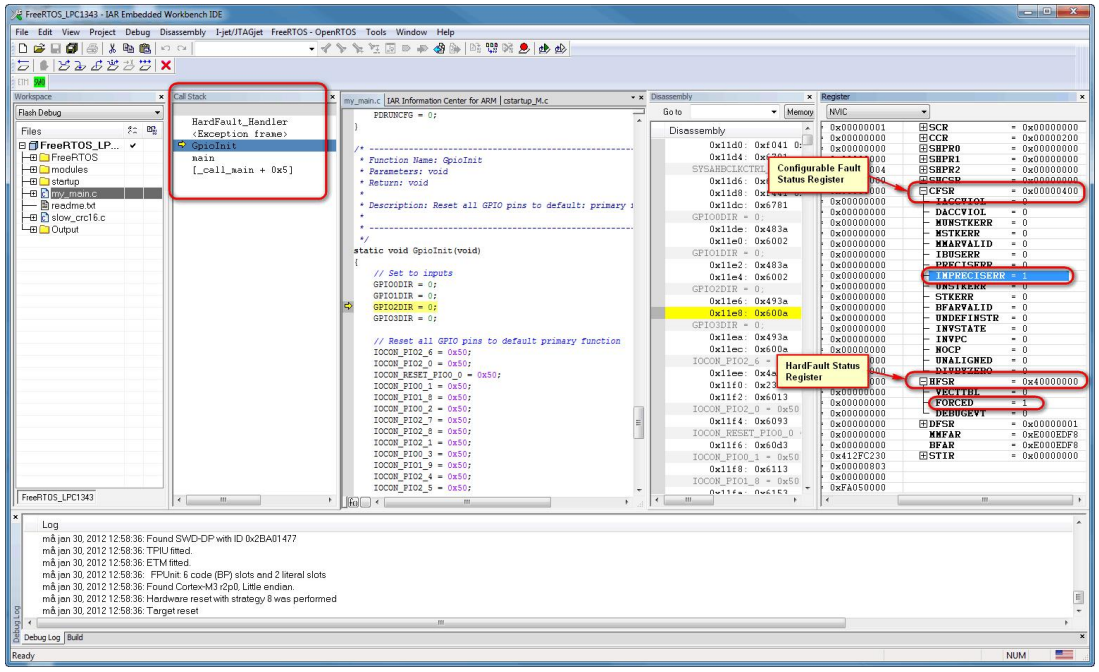

图 1

Example 2：除以零

这个例子展示了如何通过在 CCR 寄存器中启用 DIV_0_TRP 位来捕获除 0 错误。在 Call Stack 窗口中，可以看到发生无效除法的源代码行。查看 Register 窗口，NVIC:CFSR 标志 DIVBYZERO 被置 1。请看下面的图 2:

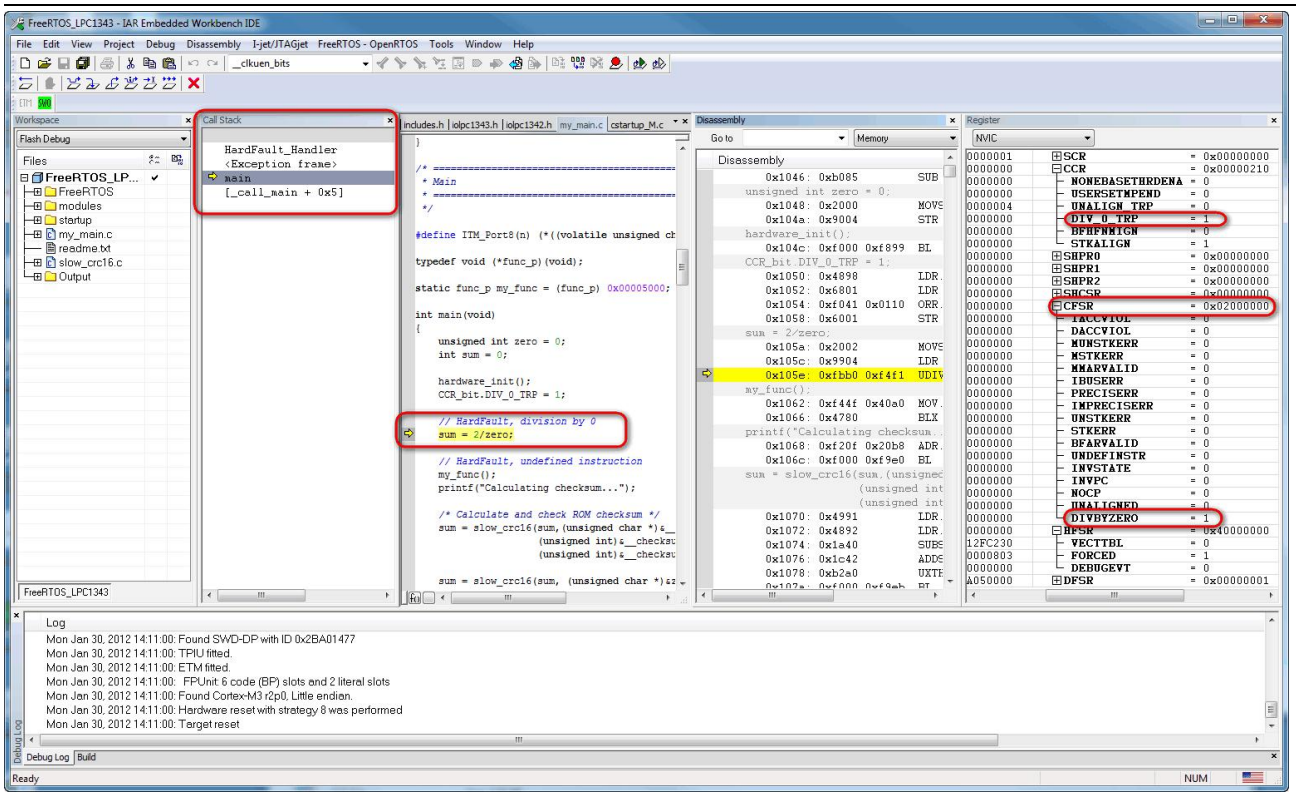

图 2

Example 3：访问无效的地址

在本例中，访问了无效内存。Call Stack 窗口可以看到在何处进行了非法访问。在 Register 窗口中，NVIC:CFSR 标志显示 PRECISERR 被置 1，发生了精确的数据访问错误，处理器已将错误地址写入了 BFAR 寄存器。请看下面的图 3 和图 4：

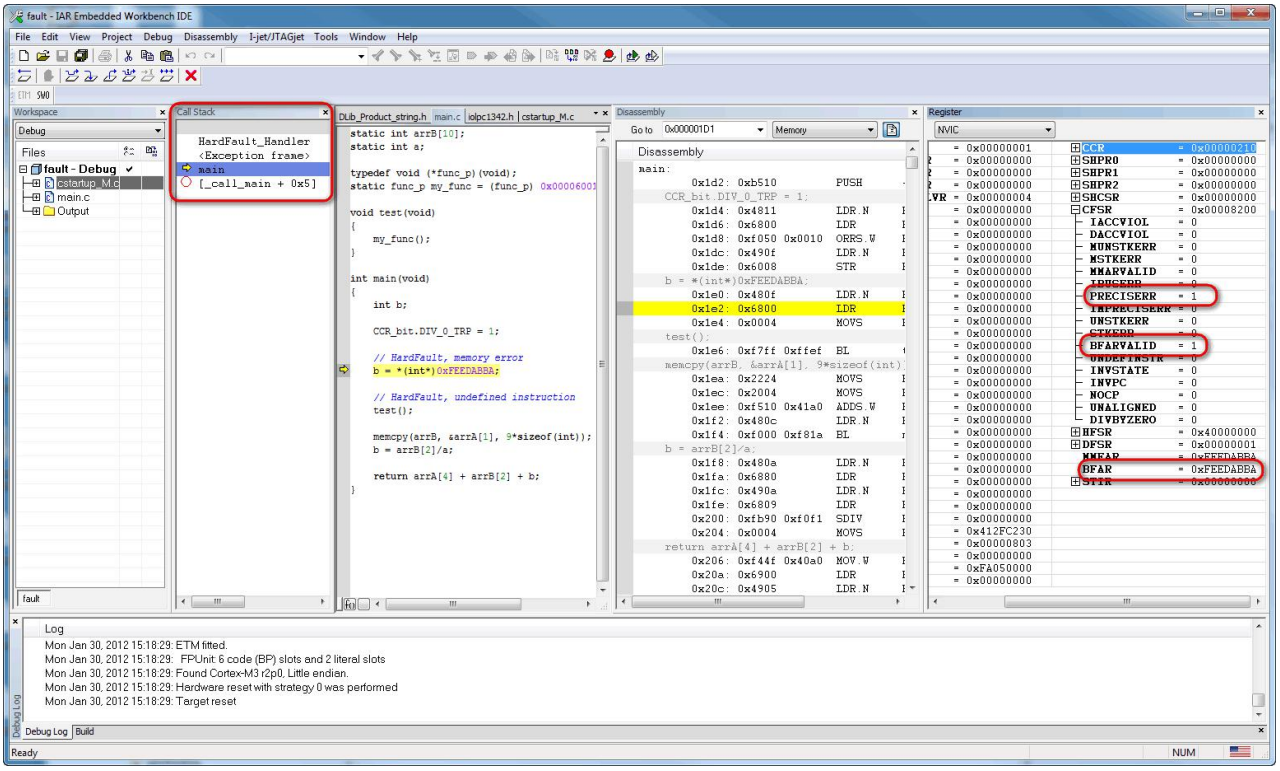

图 3

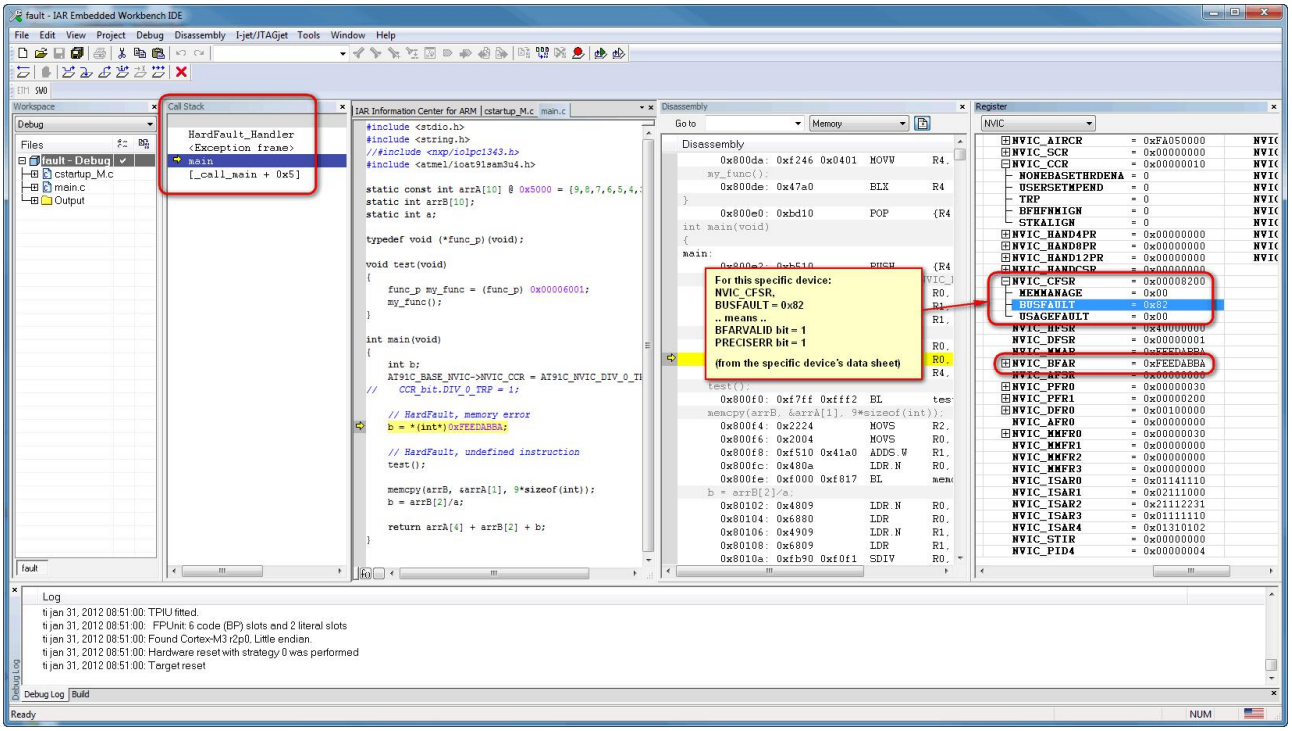

图 4

Example 4：分支到没有代码的地址

在本例中，调用了一个无效的函数指针。在 Register 窗口中，NVIC:CFSR 标志显示 UNDEFINSTR = 1，说明处理器试图执行一条未定义的指令。

在 Call Stack 窗口可以看到在何处调用了非法指令，有几种方法可以继续查找到非法指令所在的函数：

1. 在非法指令上设置断点，再次运行应用程序。当到达断点时，使用 Call Stack 窗口查找调用的函数。

2. 在 Register 窗口中查看 CPU:LR 寄存器，找到前一个调用是在哪里发出的。在反汇编窗口中查看 LR 寄存器中的地址，并“Go to”到该地址，这是上一次调用的地方。请看下面的图 5~图 7：

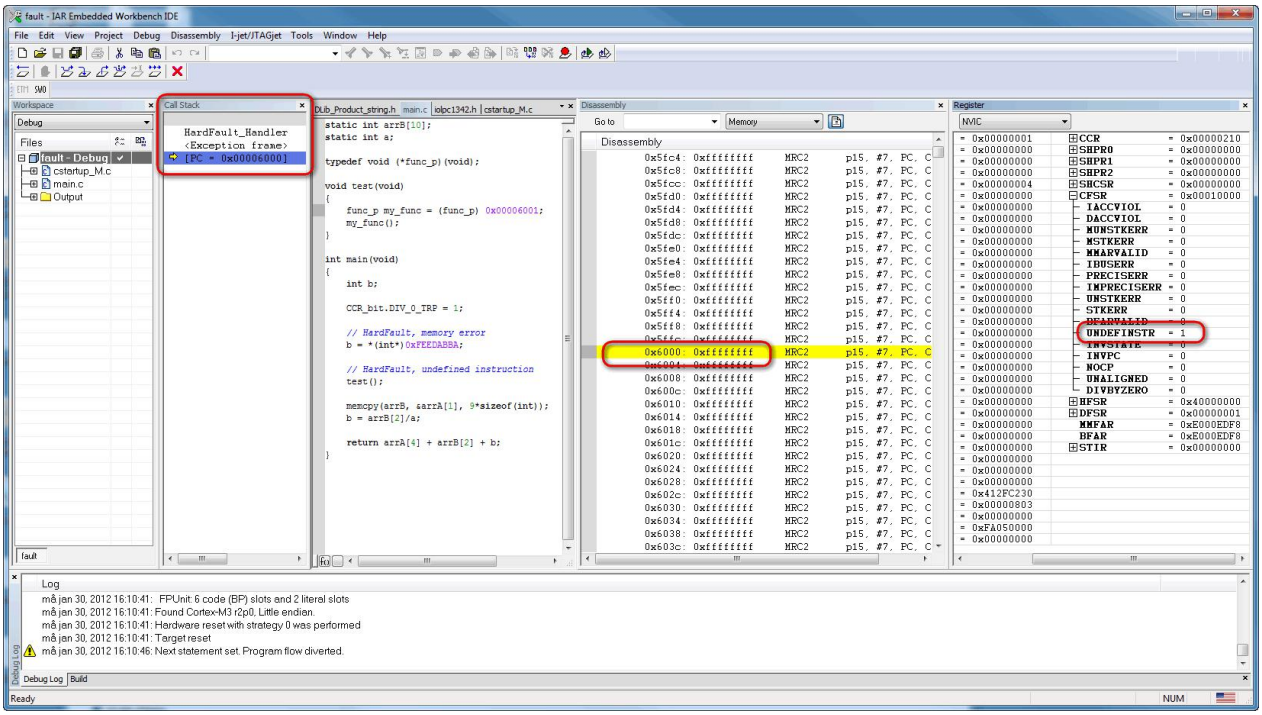

图 5

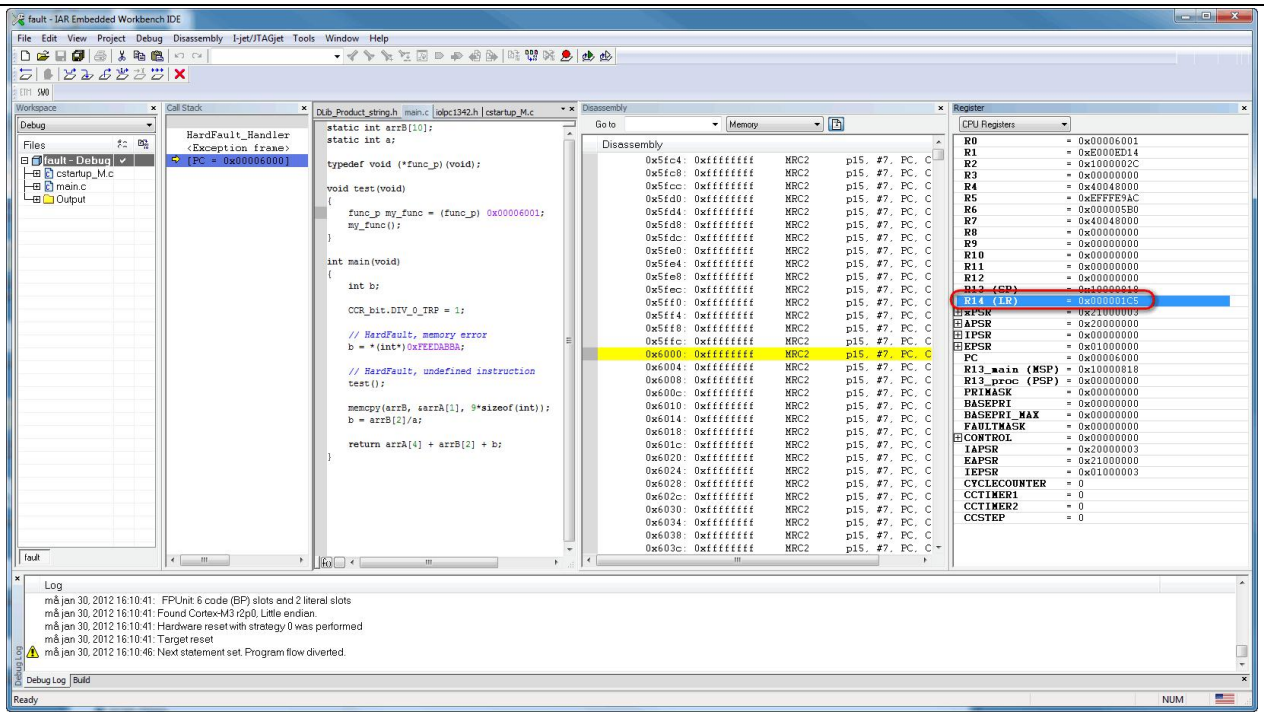
fault - IAR Embedded Workbench IDE
File Edit View Project Debug Disassembly I-jet/JTAGjet Tools Window Help
Workspace
Debug
Files
fault - Debug
cstartup_M.c
main.c
Output
Call Stack
HardFault_Handler
<Exception frame>
[PC = 0x00006000]
DLib_Product_string.h main.c iolpc1342.h cstartup_M.c
static int arrB[10];
static int a;
typedef void (*func_p)(void);
void test(void)
{
func_p my_func = (func_p) 0x00006001;
my_func();
}
int main(void)
{
int b;
CCR_bit.DIV_0_TRP = 1;
// HardFault, memory error
b = *(int*)0xFEEDABBA;
// HardFault, undefined instruction
test();
memcpy(arrB, &arrA[1], 9*sizeof(int));
b = arrB[2]/a;
return arrA[4] + arrB[2] + b;
}
Disassembly
Go to
Memory
0x5fc4: 0xffffffff MRC2 p15, #7, PC, C
0x5fc8: 0xffffffff MRC2 p15, #7, PC, C
0x5fcc: 0xffffffff MRC2 p15, #7, PC, C
0x5fd0: 0xffffffff MRC2 p15, #7, PC, C
0x5fd4: 0xffffffff MRC2 p15, #7, PC, C
0x5fd8: 0xffffffff MRC2 p15, #7, PC, C
0x5fdc: 0xffffffff MRC2 p15, #7, PC, C
0x5fe0: 0xffffffff MRC2 p15, #7, PC, C
0x5fe4: 0xffffffff MRC2 p15, #7, PC, C
0x5fe8: 0xffffffff MRC2 p15, #7, PC, C
0x5fec: 0xffffffff MRC2 p15, #7, PC, C
0x5ff0: 0xffffffff MRC2 p15, #7, PC, C
0x5ff4: 0xffffffff MRC2 p15, #7, PC, C
0x5ff8: 0xffffffff MRC2 p15, #7, PC, C
0x5ffc: 0xffffffff MRC2 p15, #7, PC, C
0x6000: 0xffffffff MRC2 p15, #7, PC, C
0x6004: 0xffffffff MRC2 p15, #7, PC, C
0x6008: 0xffffffff MRC2 p15, #7, PC, C
0x600c: 0xffffffff MRC2 p15, #7, PC, C
0x6010: 0xffffffff MRC2 p15, #7, PC, C
0x6014: 0xffffffff MRC2 p15, #7, PC, C
0x6018: 0xffffffff MRC2 p15, #7, PC, C
0x601c: 0xffffffff MRC2 p15, #7, PC, C
0x6020: 0xffffffff MRC2 p15, #7, PC, C
0x6024: 0xffffffff MRC2 p15, #7, PC, C
0x6028: 0xffffffff MRC2 p15, #7, PC, C
0x602c: 0xffffffff MRC2 p15, #7, PC, C
0x6030: 0xffffffff MRC2 p15, #7, PC, C
0x6034: 0xffffffff MRC2 p15, #7, PC, C
0x6038: 0xffffffff MRC2 p15, #7, PC, C
0x603c: 0xffffffff MRC2 p15, #7, PC, C
Register
CPU Registers
R0 = 0x00006001
R1 = 0xE000ED14
R2 = 0x1000002C
R3 = 0x00000000
R4 = 0x40048000
R5 = 0xEFFFE9AC
R6 = 0x000005B0
R7 = 0x40048000
R8 = 0x00000000
R9 = 0x00000000
R10 = 0x00000000
R11 = 0x00000000
R12 = 0x00000000
R13 (SP) = 0x10000818
R14 (LR) = 0x000001C5
xPSR = 0x21000003
APSR = 0x20000000
IPSR = 0x00000000
EPSR = 0x01000000
PC = 0x00006000
R13_main (MSP) = 0x10000818
R13_proc (PSP) = 0x00000000
PRIMASK = 0x00000000
BASEPRI = 0x00000000
BASEPRI_MAX = 0x00000000
FAULTMASK = 0x00000000
CONTROL = 0x00000000
IAPSR = 0x20000003
EAPSR = 0x21000000
IEPSR = 0x01000003
CYCLECOUNTER = 0
CCTIMER1 = 0
CCTIMER2 = 0
CCSTEP = 0
Log
må jan 30, 2012 16:10:41: FPUnit: 6 code (BP) slots and 2 literal slots
må jan 30, 2012 16:10:41: Found Cortex-M3 r2p0, Little endian.
må jan 30, 2012 16:10:41: Hardware reset with strategy 0 was performed
må jan 30, 2012 16:10:41: Target reset
må jan 30, 2012 16:10:46: Next statement set. Program flow diverted.
Debug Log Build
Ready
NUM

图 6

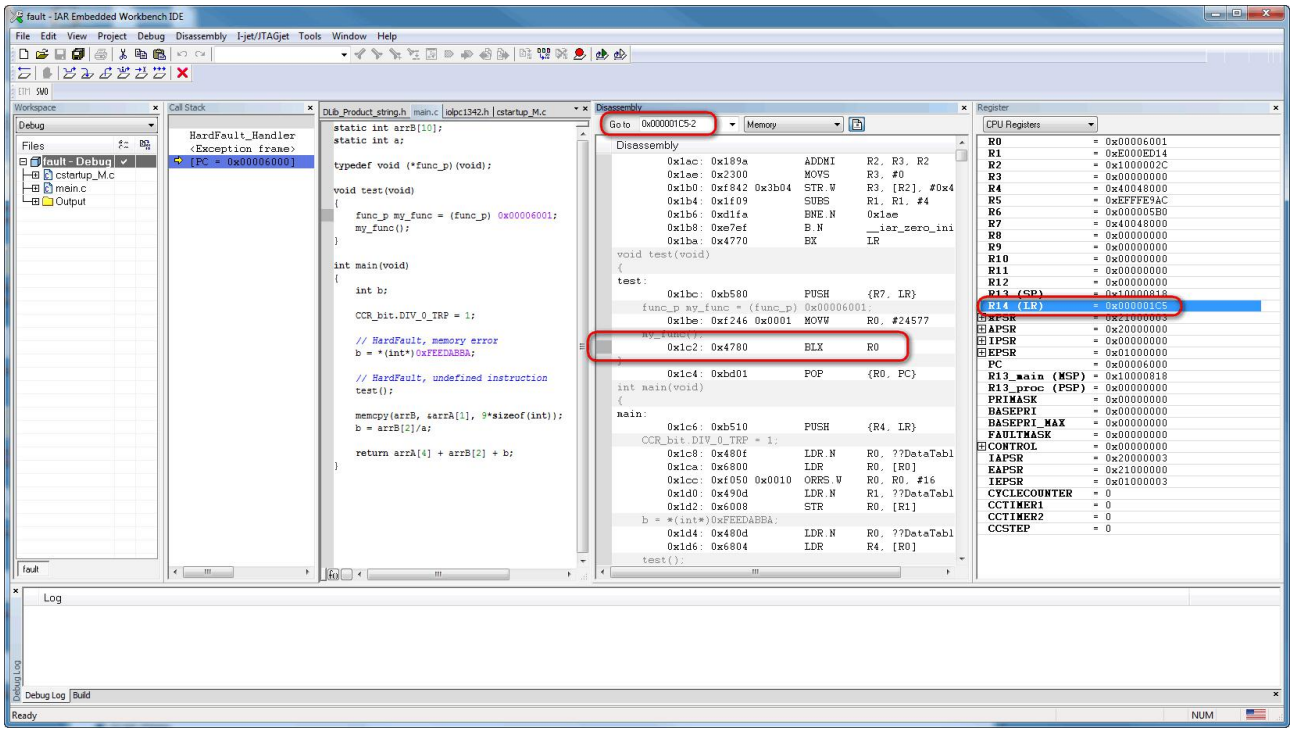
fault - IAR Embedded Workbench IDE
File Edit View Project Debug Disassembly I-jet/JTAGjet Tools Window Help
Workspace
Debug
Files
fault - Debug
cstartup_M.c
main.c
Output
Call Stack
HardFault_Handler
<Exception frame>
[PC = 0x00006000]
DLib_Product_string.h main.c iolpc1342.h cstartup_M.c
static int arrB[10];
static int a;
typedef void (*func_p)(void);
void test(void)
{
func_p my_func = (func_p) 0x00006001;
my_func();
}
int main(void)
{
int b;
CCR_bit.DIV_0_TRP = 1;
// HardFault, memory error
b = *(int*)0xFEEDABBA;
// HardFault, undefined instruction
test();
memcpy(arrB, &arrA[1], 9*sizeof(int));
b = arrB[2]/a;
return arrA[4] + arrB[2] + b;
}
Disassembly
Go to 0x000001C5-2
Memory
0x1ac: 0x189a ADDNI R2, R3, R2
0x1ae: 0x2300 MOVS R3, #0
0x1b0: 0xf842 0x3b04 STR.W R3, [R2], #0x4
0x1b4: 0x1f09 SUBS R1, R1, #4
0x1b6: 0xd1fa BNE.N 0x1ae
0x1b8: 0xe7ef B.N __iar_zero_ini
0x1ba: 0x4770 BX LR
void test(void)
{
test:
0x1bc: 0xb580 PUSH {R7, LR}
func_p my_func = (func_p) 0x00006001;
0x1be: 0xf246 0x0001 MOVW R0, #24577
my_func();
0x1c2: 0x4780 BLX R0
}
0x1c4: 0xbd01 POP {R0, PC}
int main(void)
{
main:
0x1c6: 0xb510 PUSH {R4, LR}
CCR_bit.DIV_0_TRP = 1;
0x1c8: 0x480f LDR.N R0, ??DataTabl
0x1ca: 0x6800 LDR R0, [R0]
0x1cc: 0xf050 0x0010 ORRS.W R0, R0, #16
0x1d0: 0x490d LDR.N R1, ??DataTabl
0x1d2: 0x6008 STR R0, [R1]
b = *(int*)0xFEEDABBA;
0x1d4: 0x480d LDR.N R0, ??DataTabl
0x1d6: 0x6804 LDR R4, [R0]
test();
Register
CPU Registers
R0 = 0x00006001
R1 = 0xE000ED14
R2 = 0x1000002C
R3 = 0x00000000
R4 = 0x40048000
R5 = 0xEFFFE9AC
R6 = 0x000005B0
R7 = 0x40048000
R8 = 0x00000000
R9 = 0x00000000
R10 = 0x00000000
R11 = 0x00000000
R12 = 0x00000000
R13 (SP) = 0x10000818
R14 (LR) = 0x000001C5
xPSR = 0x21000003
APSR = 0x20000000
IPSR = 0x00000000
EPSR = 0x01000000
PC = 0x00006000
R13_main (MSP) = 0x10000818
R13_proc (PSP) = 0x00000000
PRIMASK = 0x00000000
BASEPRI = 0x00000000
BASEPRI_MAX = 0x00000000
FAULTMASK = 0x00000000
CONTROL = 0x00000000
IAPSR = 0x20000003
EAPSR = 0x21000000
IEPSR = 0x01000003
CYCLECOUNTER = 0
CCTIMER1 = 0
CCTIMER2 = 0
CCSTEP = 0
Log
Debug Log Build
Ready
NUM

图 7

调试工具

为了更容易准确地识别应用程序遇到的 HardFault 错误类型，在最近版本的 IAR Embedded Workbench for ARM 中有一个调试器 macro(宏)可用。宏文件位于安装目录中：

arm\config\debugger\ARM\vector_catch.mac

从 View > Macros > Macro Registration 加载宏。当 HardFault 被触发时，宏将在 Debug Log 窗口中产生有用的输出。请看下面的图 8：

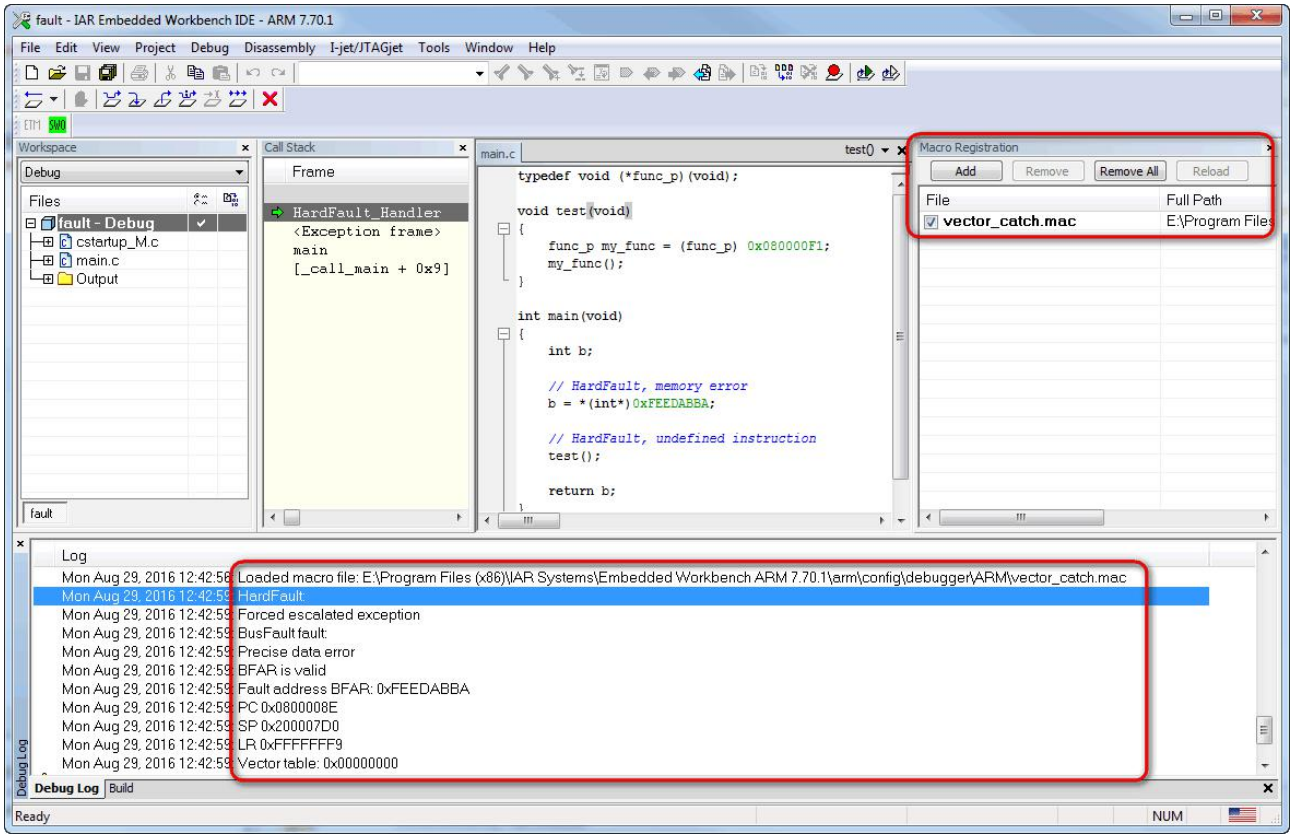

图 8

注意：

1. Register 窗口中的信息可能与上面的屏幕截图不同，这取决于您使用的是哪种 Cortex-M 设备。Cortex-M0 设备也没有提供 Cortex-M 上所有可用的 Fault 状态寄存器。
2. 如果错误处理程序中有复杂的代码，最好在处理程序的早期设置断点，以便在继续执行时寄存器和缓冲区不会丢失重要信息。通过早期设置断点，当调用 fault hander 时，将立即停止执行。
3. 有关 Fault 的更多信息，请参阅“Cortex-M3 Devices Generic User Guide”中的“HardFault”这一章。