## 一、 Hard fault 产生原因

硬件方面常见原因：

1. 电源设计有错误，造成器件供电不稳；
2. 电源质量不好，文波，噪声过大；
3. 器件接地不良；
4. 对于带有 Vcap 引脚的器件，管脚处理不当；
5. 电路中有强干扰源，对器件造成干扰；

软件方面常见原因：

1. 使用了空指针；
2. 对地址偏移量的计算有误；
3. 数组越界导致程序出错；
4. 动态内存使用不当，导致访问了已释放的内存地址；
5. 通过地址访问了已失效的局部变量；

一般因为硬件造成 Hard Fault 错误的可能性较低，90%都是软件原因造成的。所以遇到硬件中断错误，基本就是通过软件来排查

## 二、 排查问题使用到的工具：

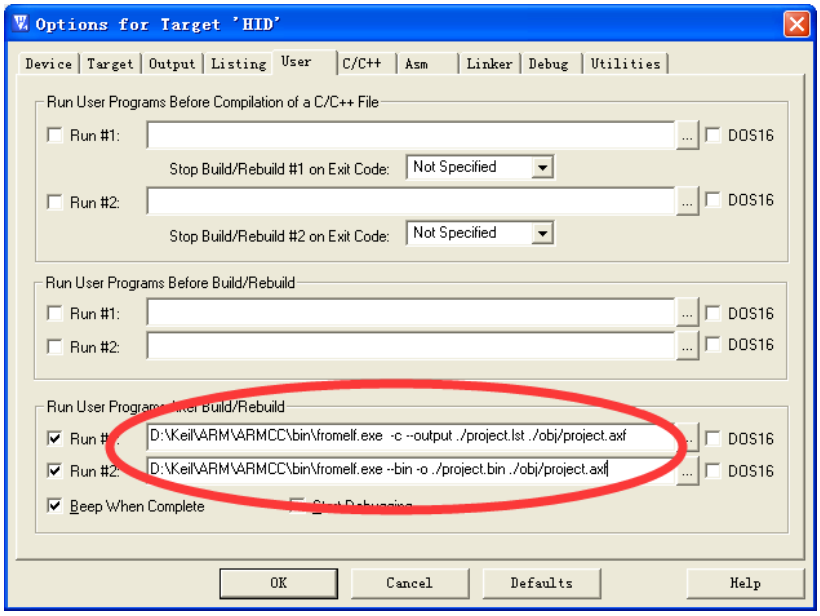
Jlink，Segger（Jlink 上位机），Keil

## 三、 排查步骤

### 1. 使用 keil 生成 map 文件，生成 lst 文件。

Map 文件是 keil 自动生成的，里面能标明每个函数、每个变量的位置。他被放在工程路径下。

lst 文件反映的是每一个函数，每一条指令的 PC 指针，在 keil 中需要调用 USER 命令生成：



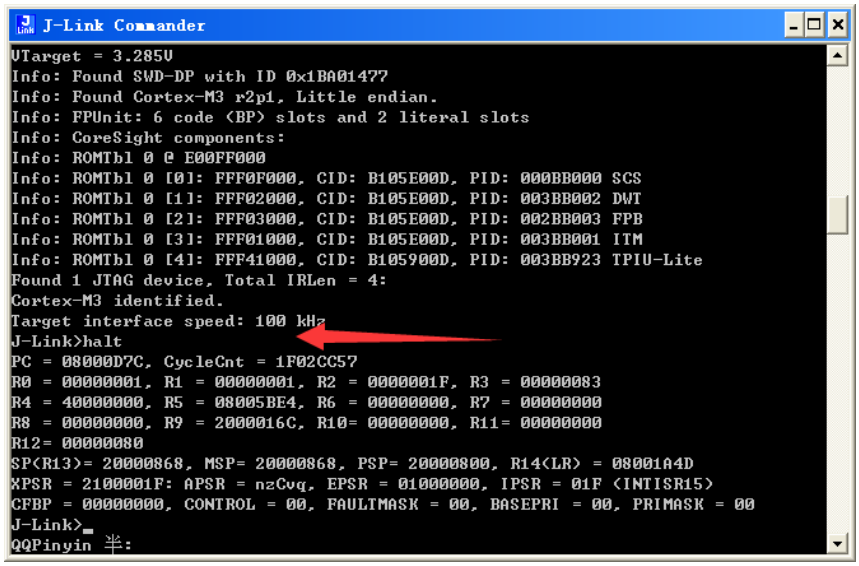

D:\Keil\ARM\ARMCC\bin\fromelf.exe -c --output ./project.lst ./obj/project.axf

D:\Keil\ARM\ARMCC\bin\fromelf.exe 表示的是 fromelf.exe 的路径；
./obj/project.axf 表示生成的 axf 文件位置，可能需要根据实际情况调整；

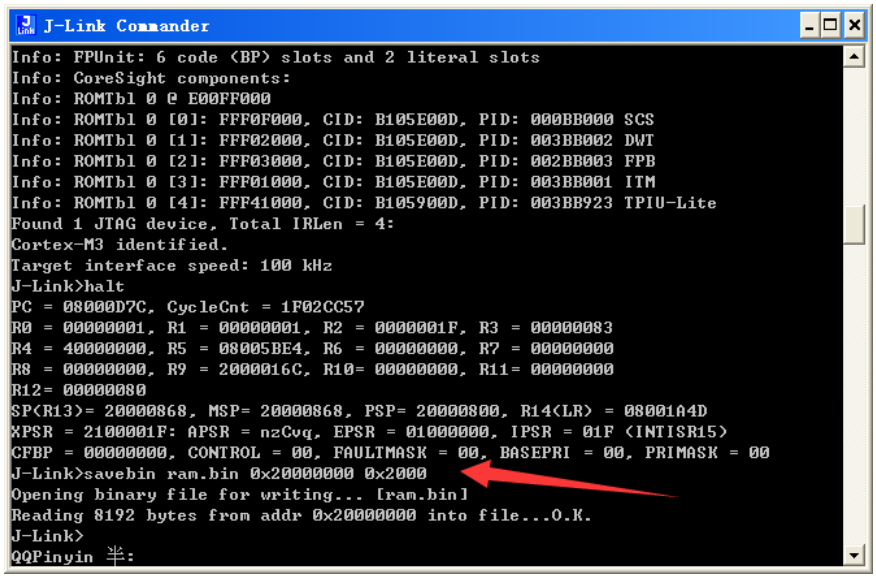
**2.保存出问题时候的 RAM；**

出问题的时候调用别断电，接上 Jlink，调用 Segger 里面的 Jlink command 来获取现场：

a.先输入一个“USB”让 Jlink 接上设备，然后输入 halt 来停住内核；

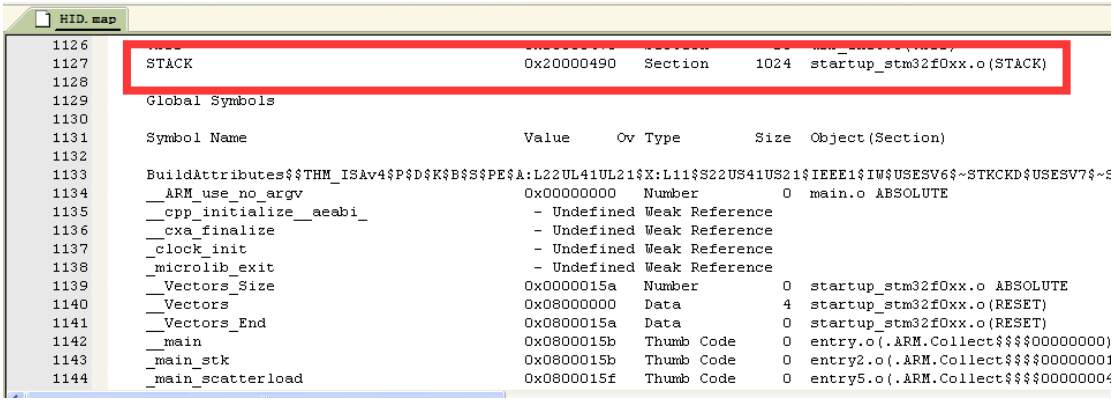



b.调用 savebin ram.bin 0x20000000 0x2000 将 RAM 中的内容全部保存下来；保存下来的东西被存在放 Segger 的安装目录中。

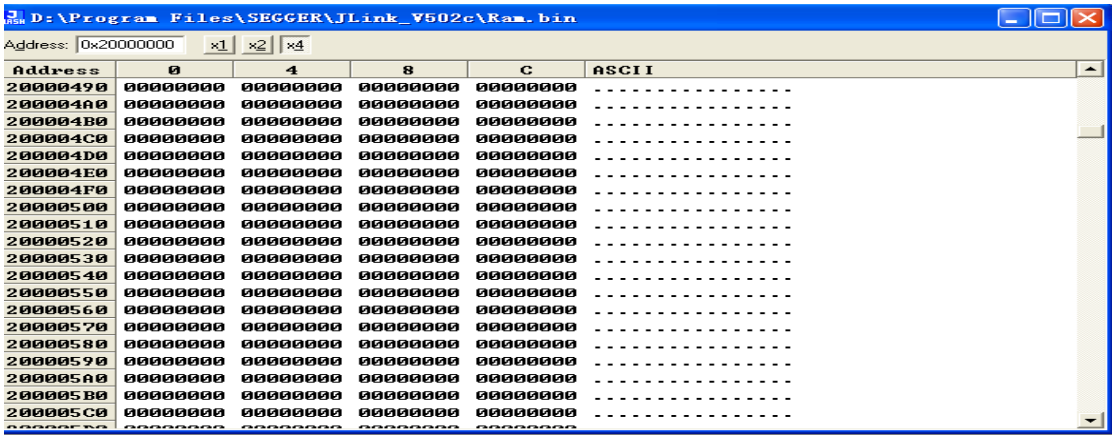


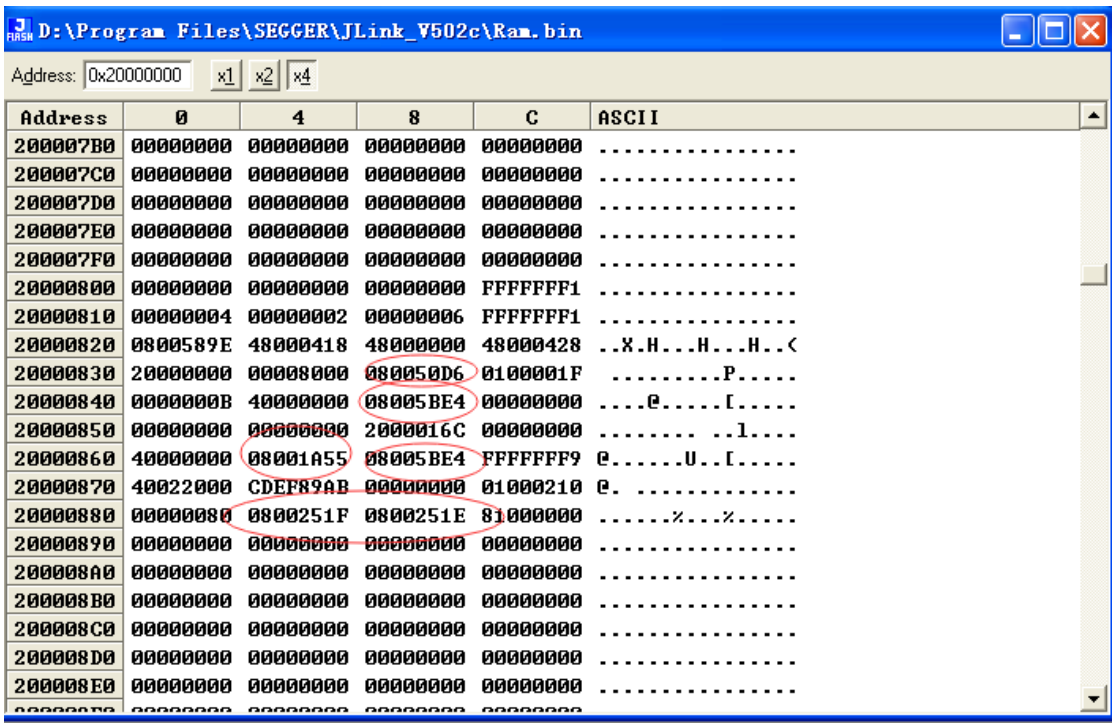
**3.分析问题**

查看 map 文件找到栈的位置。




打开保存的 bin 文件，找到进入硬件中断前调用了哪些函数，在使用哪个变量，然后逐一分析。



从栈的底部往上看，哪个地方的值是函数指针，然后对应 lst 文件去逐一查看，分析，就能大致知道是在执行哪个函数，哪一条指令，或者是调用某个参数导致的硬件中断错误的。通过 map 文件可以知道每个变量的位置，可以直接去查看我们保存下来的 ram 中变量的当前情况来分析程序逻辑。

**Jlink Command 使用方法：**

**f** **Firmware info 用来查看 Jlink 的硬件版本**

```
J-Link>f
Firmware: J-Link V9 compiled Sep 18 2015 19:53:12
Hardware: V9.20
```

**h** **halt 用来停止 MCU 内核，可以查看内核的 PC 指针等特殊寄存器**

```
J-Link>h
PC = 08000D7C, CycleCnt = F39B01CC
R0 = 00000001, R1 = 00000001, R2 = 0000001F, R3 = 00000083
R4 = 40000000, R5 = 08005BE4, R6 = 00000000, R7 = 00000000
R8 = 00000000, R9 = 2000016C, R10= 00000000, R11= 00000000
R12= 00000080
SP(R13)= 20000868, MSP= 20000868, PSP= 20000800, R14(LR) = 08001A4D
XPSR = 2100001F: APSR = nzCvq, EPSR = 01000000, IPSR = 01F (INTISR15)
CFBP = 00000000, CONTROL = 00, FAULTMASK = 00, BASEPRI = 00, PRIMASK = 00
```

**g** **go 用来激活被 halt 的内核**

**Sleep** **Waits the given time (in milliseconds). Syntax: Sleep <delay>用来延时**

**s** **Single step the target chip 单步调试代码，可以先执行 halt，然后再来单步调试**

```
J-Link>halt
PC = 08000D7C, CycleCnt = A970C614
R0 = 00000001, R1 = 00000001, R2 = 0000001F, R3 = 00000083
R4 = 40000000, R5 = 08005BE4, R6 = 00000000, R7 = 00000000
R8 = 00000000, R9 = 2000016C, R10= 00000000, R11= 00000000
R12= 00000080
SP(R13)= 20000868, MSP= 20000868, PSP= 20000800, R14(LR) = 08001A4D
XPSR = 2100001F: APSR = nzCvq, EPSR = 01000000, IPSR = 01F (INTISR15)
CFBP = 00000000, CONTROL = 00, FAULTMASK = 00, BASEPRI = 00, PRIMASK = 00
J-Link>s
08000D7C:  FE E7              B       #-0x04
```

**st** **Show hardware status 显示 Jlink 当前状态**

```
J-Link>st
VTarget=3.285V
ITarget=0mA
TCK=0 TDI=0 TDO=0 TMS=1 TRES=1 TRST=0
Supported target interface speeds:
 - 12 MHz/n, (n>=1). => 12000kHz, 6000kHz, 4000kHz, ...
 - Adaptive clocking
J-Link>
```

**hwinfo** **Show hardware info 显示 Jlink 的硬件信息**

**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)**
读取指令：

```
J-Link>mem 0x8000000 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 08 00 00 00 00
J-Link>mem8 0x8000000 2
08000000 = 90 08
J-Link>mem16 0x8000000 2
08000000 = 0890 2000
J-Link>mem32 0x8000000 2
08000000 = 20000890 08000171
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)**
写指令：

```
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 00040000
J-Link>
```

**erase** **Erase internal flash of selected device. Syntax: Erase**
擦除指令，先选定器件然后再来执行擦除

```
J-Link>device stm32f100c8
Info: Device "STM32F100C8" selected.
Reconnecting to target...
Info: Found SWD-DP with ID 0x1BA01477
Info: Found SWD-DP with ID 0x1BA01477
Info: Found Cortex-M3 r2p1, Little endian.
Info: FPUnit: 6 code (BP) slots and 2 literal slots
Info: CoreSight components:
Info: ROMTbl 0 @ E00FF000
Info: ROMTbl 0 [0]: FFF0F000, CID: B105E00D, PID: 000BB000 SCS
Info: ROMTbl 0 [1]: FFF02000, CID: B105E00D, PID: 003BB002 DWT
Info: ROMTbl 0 [2]: FFF03000, CID: B105E00D, PID: 002BB003 FPB
Info: ROMTbl 0 [3]: FFF01000, CID: B105E00D, PID: 003BB001 ITM
Info: ROMTbl 0 [4]: FFF41000, CID: B105900D, PID: 003BB923 TPIU-Lite
J-Link>erase
Erasing device (STM32F100C8)...
Info: J-Link: Flash download: Only internal flash banks will be erase
To enable erasing of other flash banks like QSPI or CFI, it needs to
via "exec EnableEraseAllFlashBanks"
Info: J-Link: Flash download: Total time needed: 4.503s (Prepare: 1.6
e: 0.000s, Erase: 1.797s, Program: 0.000s, Verify: 0.000s, Restore: 1
Erasing done.
J-Link>
```

**loadfile** **Load data file into target memory.**

**Syntax: loadfile <filename>, [<addr>]**

**Supported extensions: *.bin, *.mot, *.hex, *.srec**

**<addr> is needed for bin files only. //用来下载文件**

**loadbin** **Load *.bin file into target memory.**

**Syntax: loadbin <filename>, <addr>//用来下载 bin 文件**

**savebin** **Saves target memory into binary file.//用来保存 bin 文件**

**Syntax: savebin <filename>, <addr>, <NumBytes>**

**SetPC** **Set the PC to specified value. Syntax: SetPC <Addr>//用来设置 PC 指针，可以让程序从某个地方开始执行**