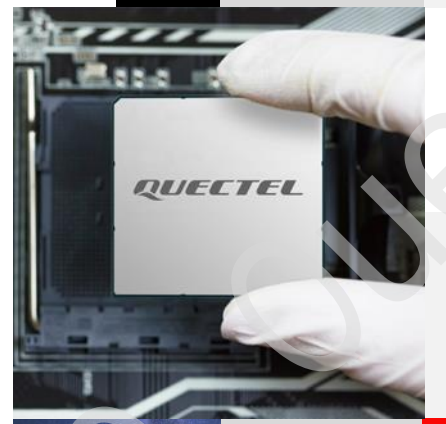
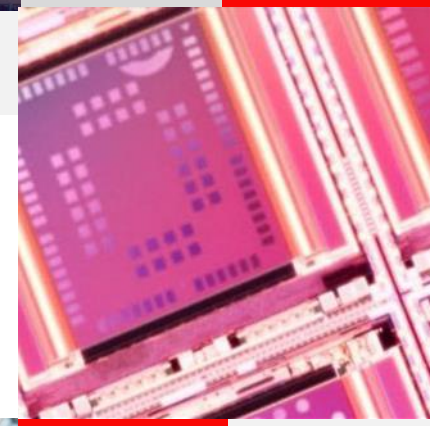
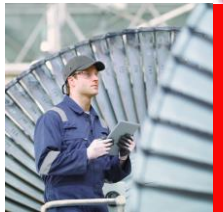

QUECTEL
trace32死机内存分析
副标题
Build a Smarter World



一 memdump简介

二 触发memdump的异常分类和流程

三 memdump可查看的信息

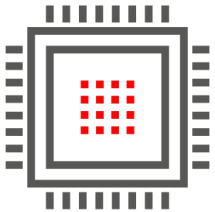
四 memdump分析实例

Build a Smarter World

Trace32 simulator简介

当系统crash后通过异常中断处理来抓取ram中的数据，然后利用Trace32进行故障现场的查看来排查问题。这实际上用到的就是trace32的simulator功能，也就是仿真器功能，我们只需要获取到设备的内存快照来进行指令集的仿真，以此查看故障现场，而不用真实的连接目标板来实时调试。

Memdump如何导出



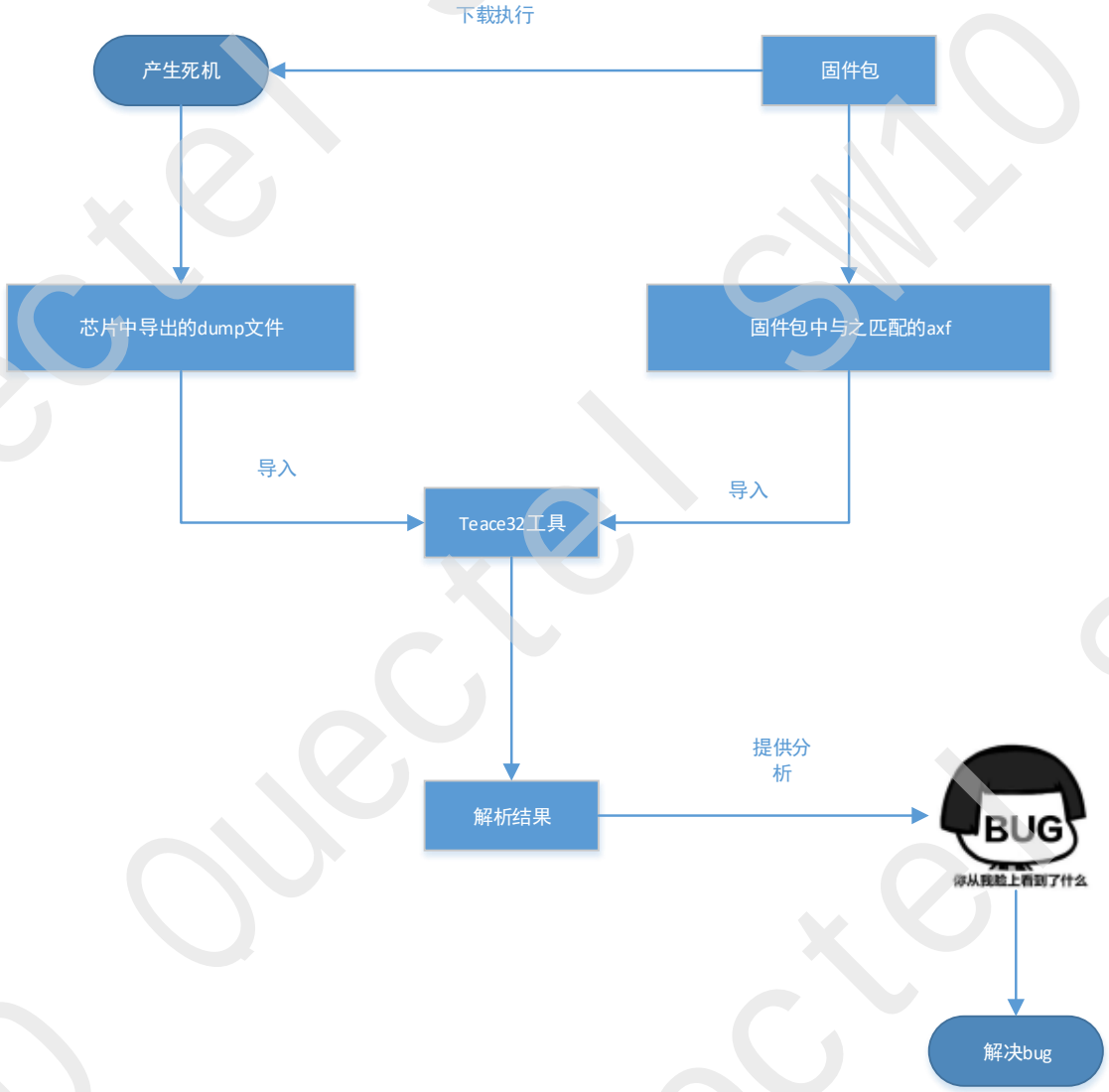
Usb或其他总线

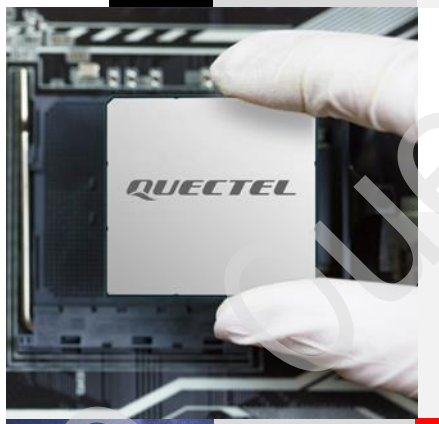
com_compmde.bin	2022/6/14 21:24	UltraEdit Docum...	1 KB
com_CustVer.bin	2022/6/14 21:24	UltraEdit Docum...	1 KB
com_DDR_RW.bin	2022/6/14 21:24	UltraEdit Docum...	7,744 KB
com_DSP_DDR.bin	2022/6/14 21:24	UltraEdit Docum...	448 KB
com_DSPdtcm.bin	2022/6/14 21:24	UltraEdit Docum...	256 KB
com_DTCM.bin	2022/6/14 21:24	UltraEdit Docum...	64 KB
com_dtcSpy.bin	2022/6/14 21:24	UltraEdit Docum...	2 KB
com_EE_Hbuf.bin	2022/6/14 21:24	UltraEdit Docum...	1 KB
com_EE_Hbuf.cmm	2022/6/14 21:25	CMM 文件	6 KB
com_EE_Hbuf.xdb	2022/6/14 21:25	XDB 文件	7 KB
com_errlog.bin	2022/6/14 21:24	UltraEdit Docum...	3 KB
com_ITCM.bin	2022/6/14 21:24	UltraEdit Docum...	64 KB
com_L1aSpy.bin	2022/6/14 21:24	UltraEdit Docum...	4 KB
com_L2_sram.bin	2022/6/14 21:24	UltraEdit Docum...	128 KB
com_PS_DAT.bin	2022/6/14 21:24	UltraEdit Docum...	448 KB
com_rti_tsk.bin	2022/6/14 21:24	UltraEdit Docum...	3 KB
com_SQU.bin	2022/6/14 21:24	UltraEdit Docum...	32 KB
com_usbIntD.bin	2022/6/14 21:24	UltraEdit Docum...	1 KB
com_usbLog.bin	2022/6/14 21:24	UltraEdit Docum...	4 KB
com_usbRese.bin	2022/6/14 21:24	UltraEdit Docum...	1 KB
com_wdtKICK.bin	2022/6/14 21:24	UltraEdit Docum...	1 KB
com_wifiSpy.bin	2022/6/14 21:24	UltraEdit Docum...	2 KB

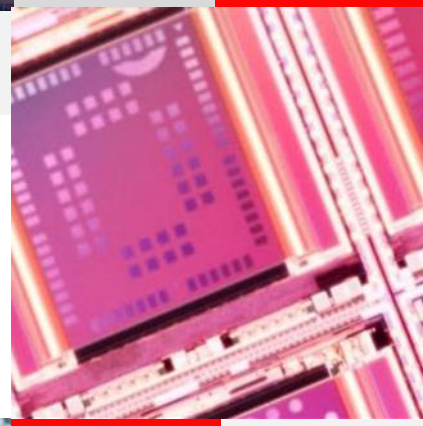
芯片内存信息导出到pc上的文件

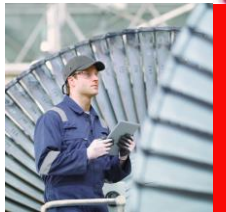
Memdump就是触发死机后，芯片将死机现场的内存导入到个人电脑上，使用trace32工具我们可以回溯芯片死机时的寄存器，和代码执行的步骤，进而分析死机原因

Trace32解析dump文件步骤









一 memdump简介

二 触发memdump的异常分类和触发流程

三 memdump可查看的信息

四 memdump解析实战

Build a Smarter World

Dump的分类-arm异常

data_abort（数据中止访问异常）：
1.内存属性：修改了ro属性的内存内容在arm芯片中一般存在mpu或者mmu，这两个单元可以配置内存的读写属性，若某段内存的属性配置为readony，但是在实际代码运行时修改此段内存内容就会触发此种异常。
2.地址访问越界：在使用mpu单元的芯片中，代码访问的地址超出了实际物理地址会触发此种异常，在使用mmu单元的芯片中，mmu的分页地址变换机构先将逻辑地址转换为页号和页内地址，然后再将页号与页表寄存器中的页表长度进行比较，如果页号大于或者等于页表长度，则会触发此种异常

Prefetch_abort（预取指令中止异常）：
异常发生在CPU流水线取指阶段，当R15寄存器指向地址是非法地址时（例如r15指向的地址不存在）存储器会向处理器发出中止信号；当预取的指令被执行时才会产生预取指中止异常。

Undefine instruction（未定义指令异常）：
异常发生在流水线技术里的译码阶段，如果当前指令不能被识别为芯片平台规定的有效指令，就会产生未定义指令异常

以上异常产生后，芯片会产生异常中断，在异常中断处理中会进入memdump流程

Dump的分类-代码逻辑主动ASSERT

在编写代码时，程序员为防止代码异常执行导致整个系统的崩溃，会对代码加一些异常状态判断，正常情况下代码是不会进入这些异常分支，如果代码进入这些异常状态就会触发主动ASSERT
ASSERT会触发memdump流程，同时将死机代码所在文件和行号信息以及一些状态值保存到内存中。

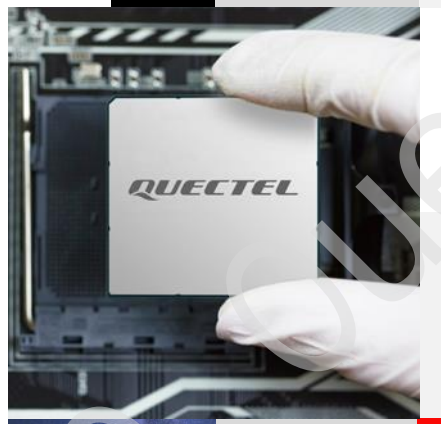
```
299: OSA_STATUS OsaTaskCreate( OsaRefT *pOsaRef, OsaTaskCreateParamsT *pParams )
300: {
301:     static UINT8
302:         objectCnt = 0 ;
303:
304:     CHAR
305:         *pName ;
306:
307:     OsaRefT
308:         OsaRef ;
309:
310:     TX_THREAD
311:        *pTxTask ;
312:
313:     UINT
314:         priority,
315:         status ;
316:
317:     ULONG
318:         size ;
319:
320:     VOID
321:         *stackPtr ;
322:
323:     ULONG
324:         malloc_flag = 0;
325:
326:     OSA_ASSERT(pOsaRef) ;
327:     OSA_ASSERT(pParams) ;
328:
329:     MALLOC_REF( OsaRef, sizeof(TX_THREAD) ) ;
330:     pTxTask = (TX_THREAD *)OsaRef ;
331:
332:     size = pParams->stackSize ;
333:     priority = pParams->priority + 3; //priority 0-2 is used for HISR, so all of the thread's priority is plu
334:
```

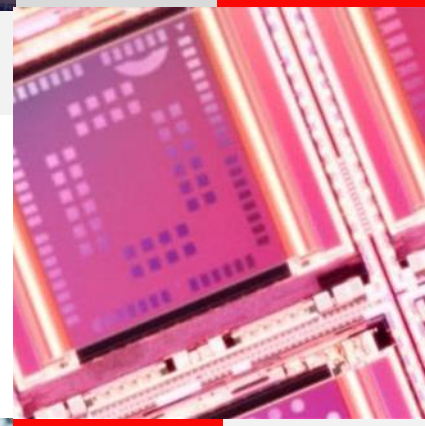
一 memdump简介

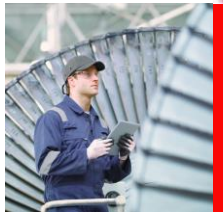
二 触发memdump的异常分类和流程

三 memdump可查看的信息

四 memdump解析实战







Build a Smarter World

www.quectel.com

Memdump所包含的信息

一个正在运行的代码至少有哪几部分构成?

站在C语言角度，一个程序大致分为以下几个部分（这里只列举主要的段）

1. 代码段
2. 数据段
3. bss段
4. 栈
5. 堆

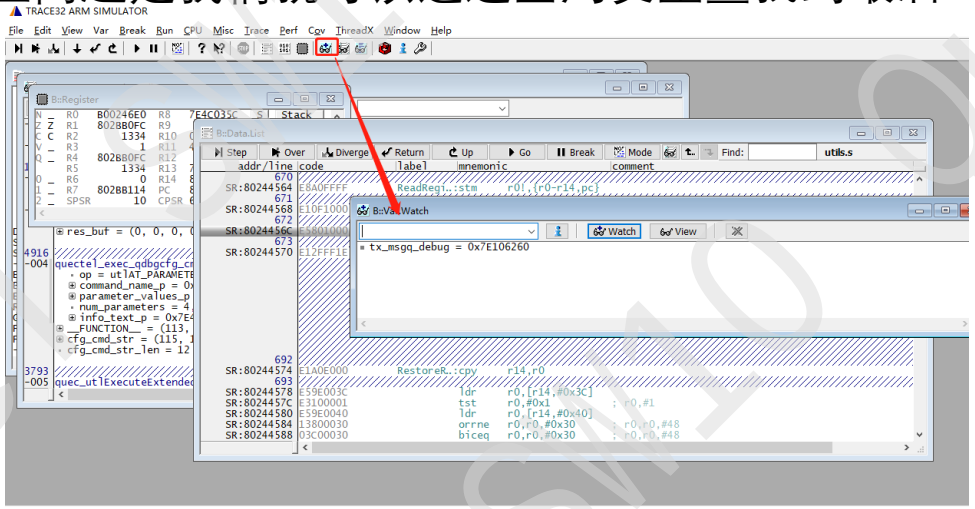
以上这5部分是分析死机的关键信息，都被包含在memdump的文件中

Memdump的作用

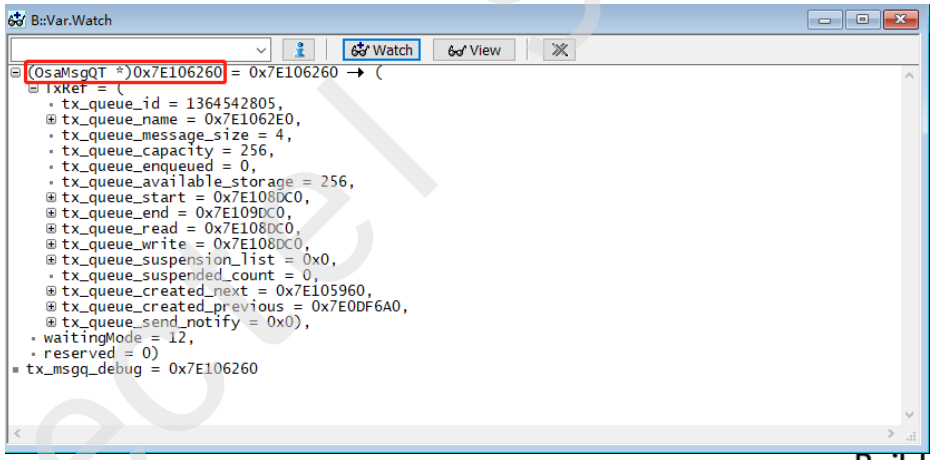
1.查看内存中全局变量值

在代码中我们可以增加一些全局变量用于debug，当发生问题是我们就可以通过全局变量查找到最后一次赋值

```
838: OSA_STATUS OsaMsgQRecv( OsaRefT OsaRef, OsaMsgQRecvParamsT *pParams )
839: {
840:     UINT
841:         status ;
842:
843:     OsaMsgQT
844:         *pMsgQRef = (OsaMsgQT *)OsaRef;
845:
846:     OSA_ASSERT(pParams) ;
847:     OSA_ASSERT(pMsgQRef) ;
848:     if(EEHandlerFlag != 1)
849:     {
850:         tx_msgq_debug = (UINT32)OsaRef;
851:     }
852:     status = tx_queue_receive( (TX_QUEUE *)&pMsgQRef->TxRef, (VOID *)pParams->msgPtr,
853:                                Osa_TimeoutValue(pParams->timeout) ) ;
854:
855:     RETURN_STATUS( status, TX_SUCCESS ) ;
856: }
857: #pragma arm section code
858:
```



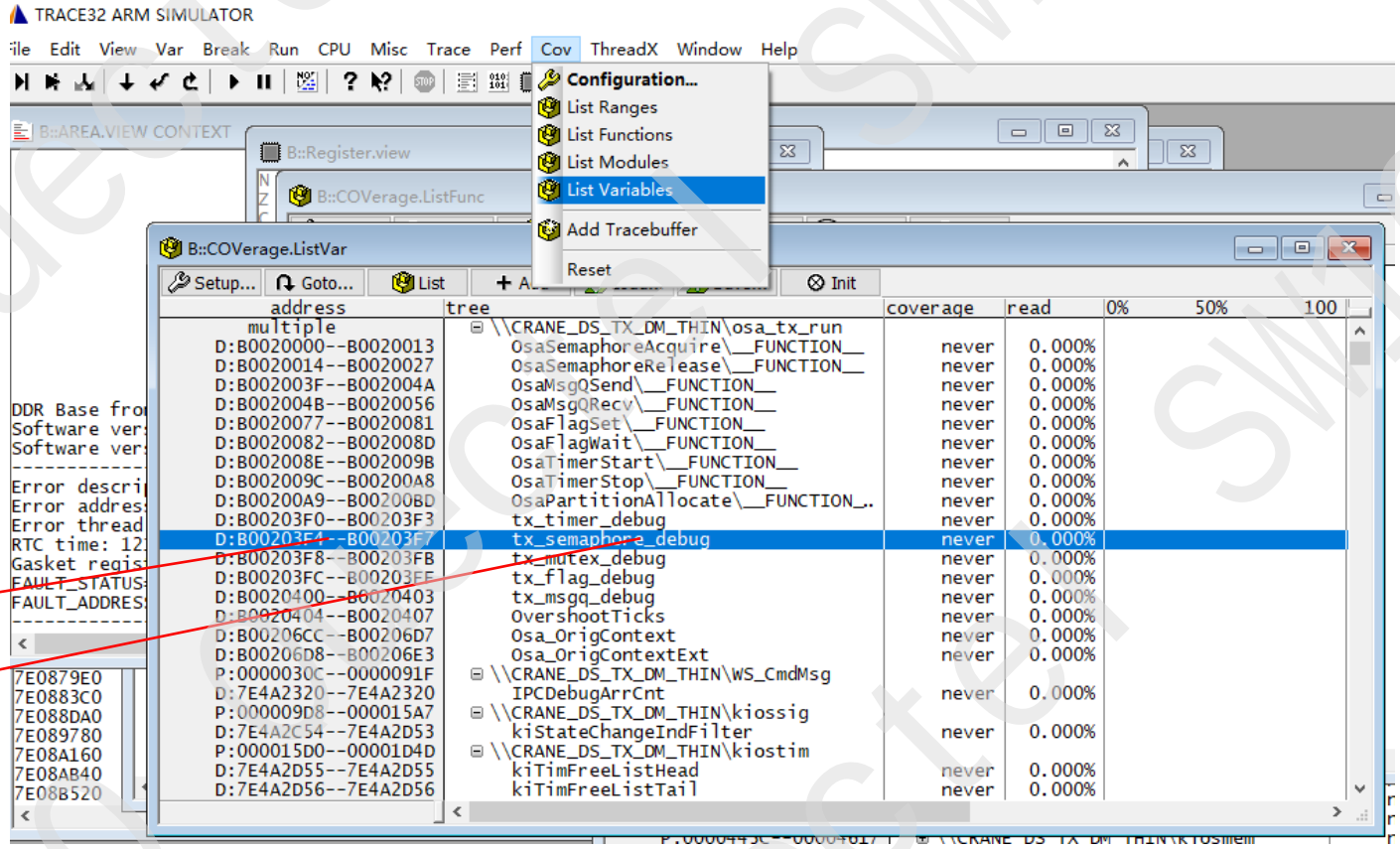
通过给地址强制类型，可以获取到结构体变量的详细信息



Memdump的作用

2.查看内存中全局变量的分布

Memdump中可以看到全局变量在内存中的排布，通过这个我们可以很容易的找到全局变量的越界访问问题

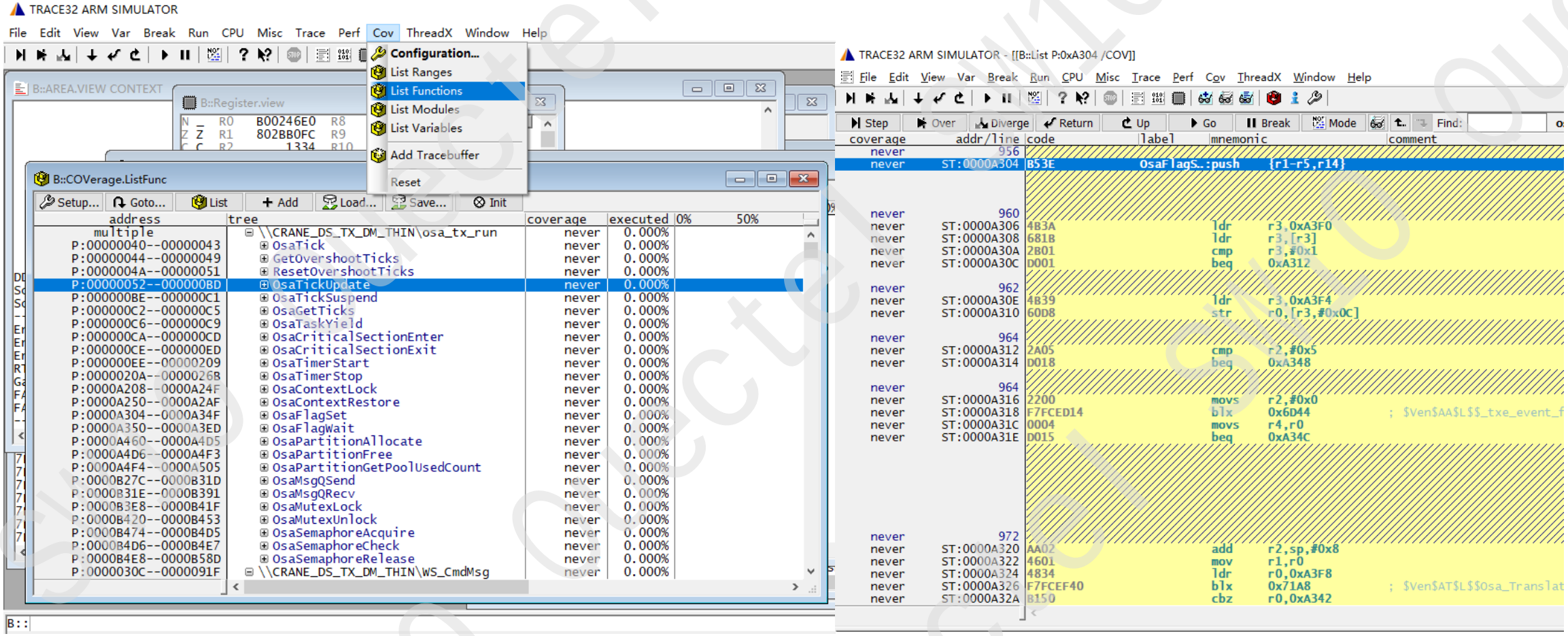


全局变量的地址范围

全局变量的符号

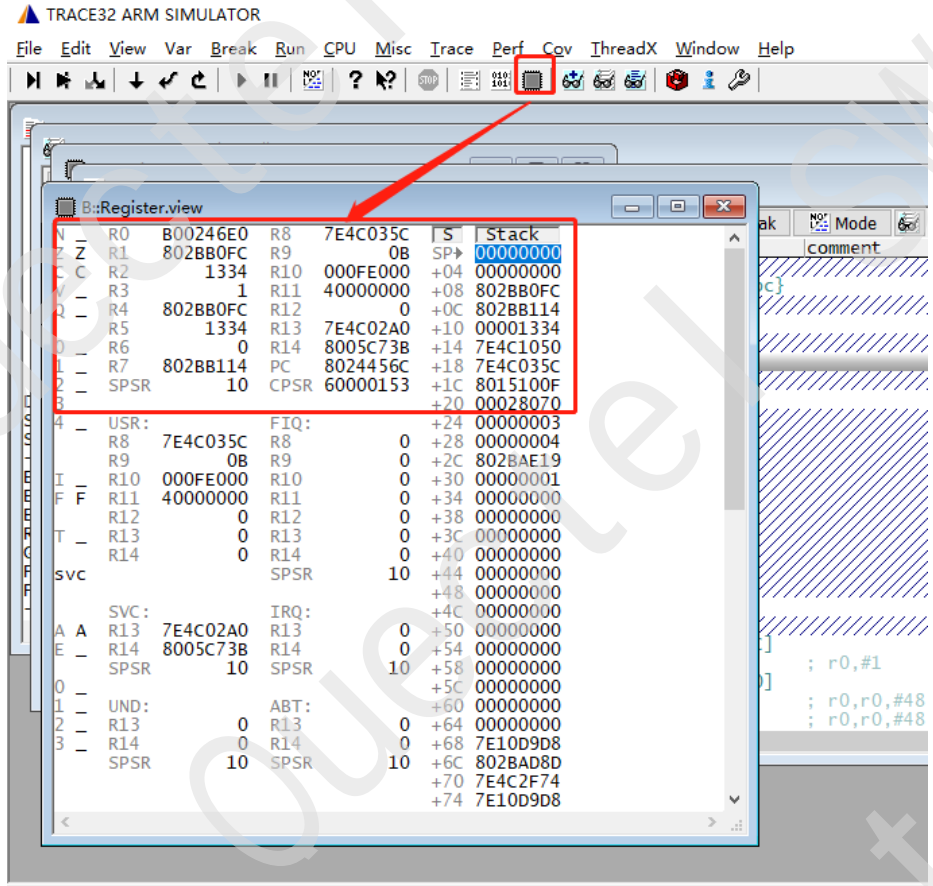
Memdump的作用

3.查看函数在代码段的地址以及反汇编代码，产生死机时可以通过汇编代码反推代码的执行步骤，即使在没有源码的情况下也是可以推算出死机的原因



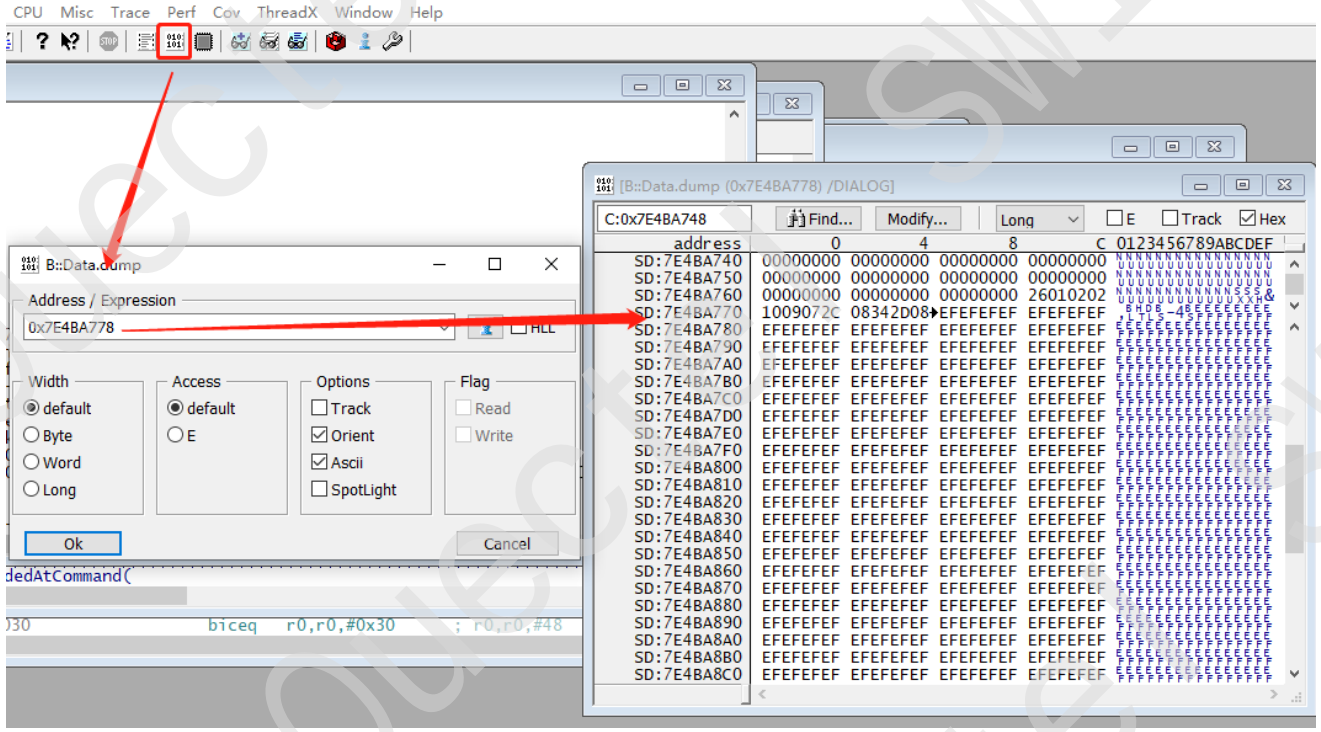
Memdump的作用

4.查看死机时的r0-r15，cpsr，spsr寄存器的值，再结合反汇编代码我们可以反推代码参数运算和传递是否存在错误，找到错误原因



Memdump的作用

5.查看芯片内存
在分析heap，栈溢出，数组越界，或者是汇编里直接操作内存的语句（如LDR,STR）等我们可以通过直接查看内存分析问题



Memdump的作用

6.查看任务各项指标

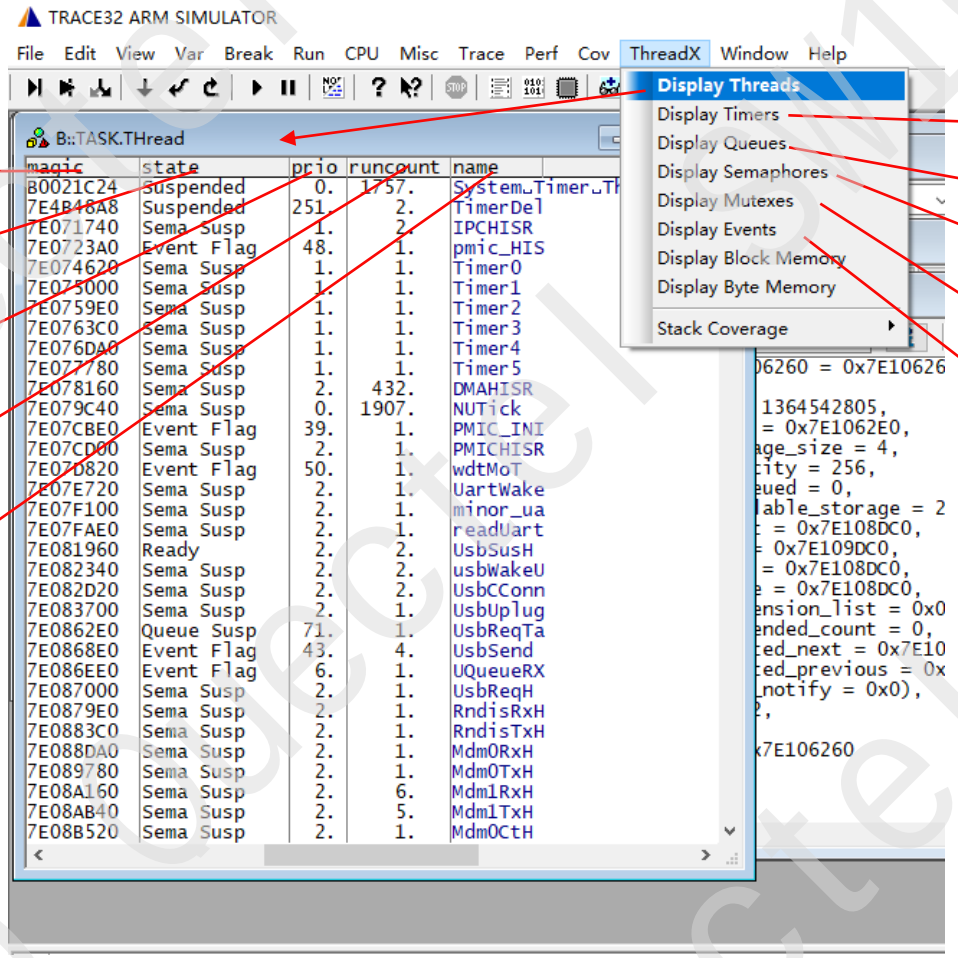
任务tcb的地址

任务状态

任务优先级

任务运行次数

任务名



定时器

消息队列

信号量

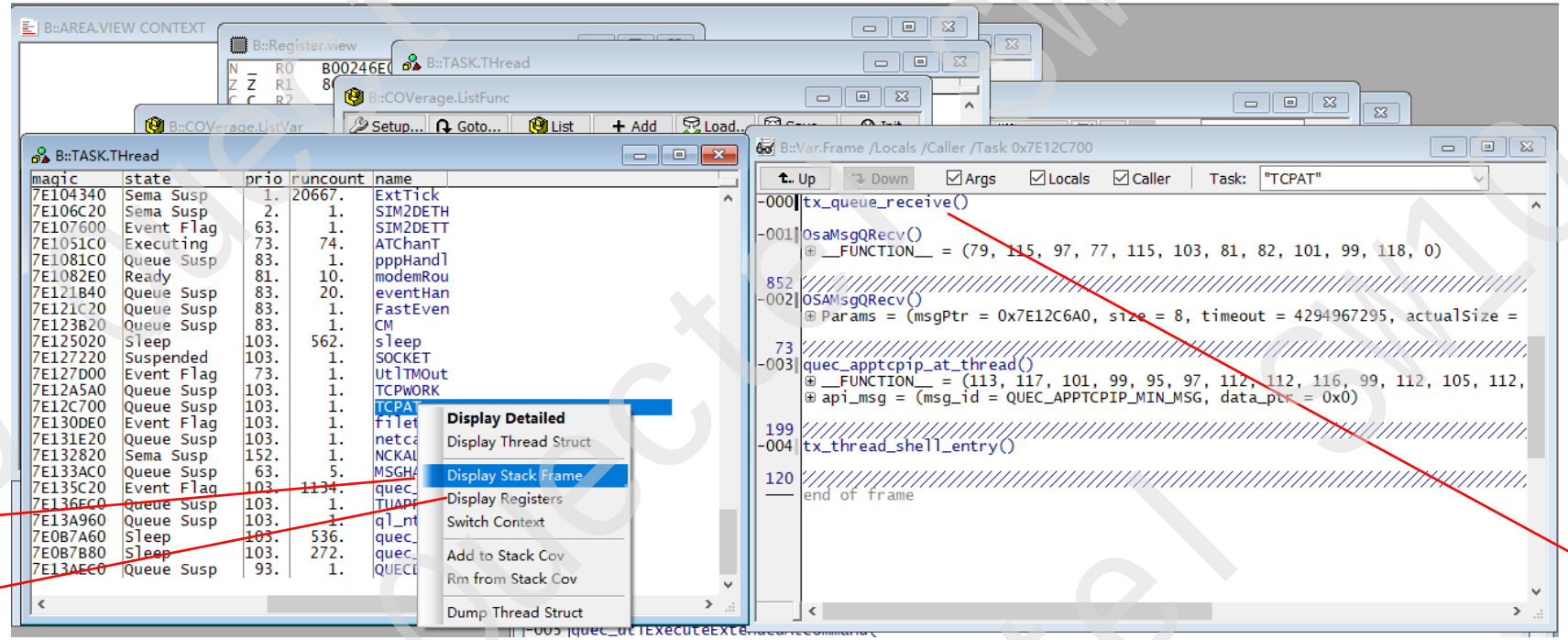
锁

事件

Memdump的作用

6.查看任务各项指标

将鼠标放在任务名上右击，点击Display Stack Frame可以先线程在切出前执行了哪些语句
点击Display Registers 可以看到线程当前的各个寄存器的值。



线程最后的调用栈

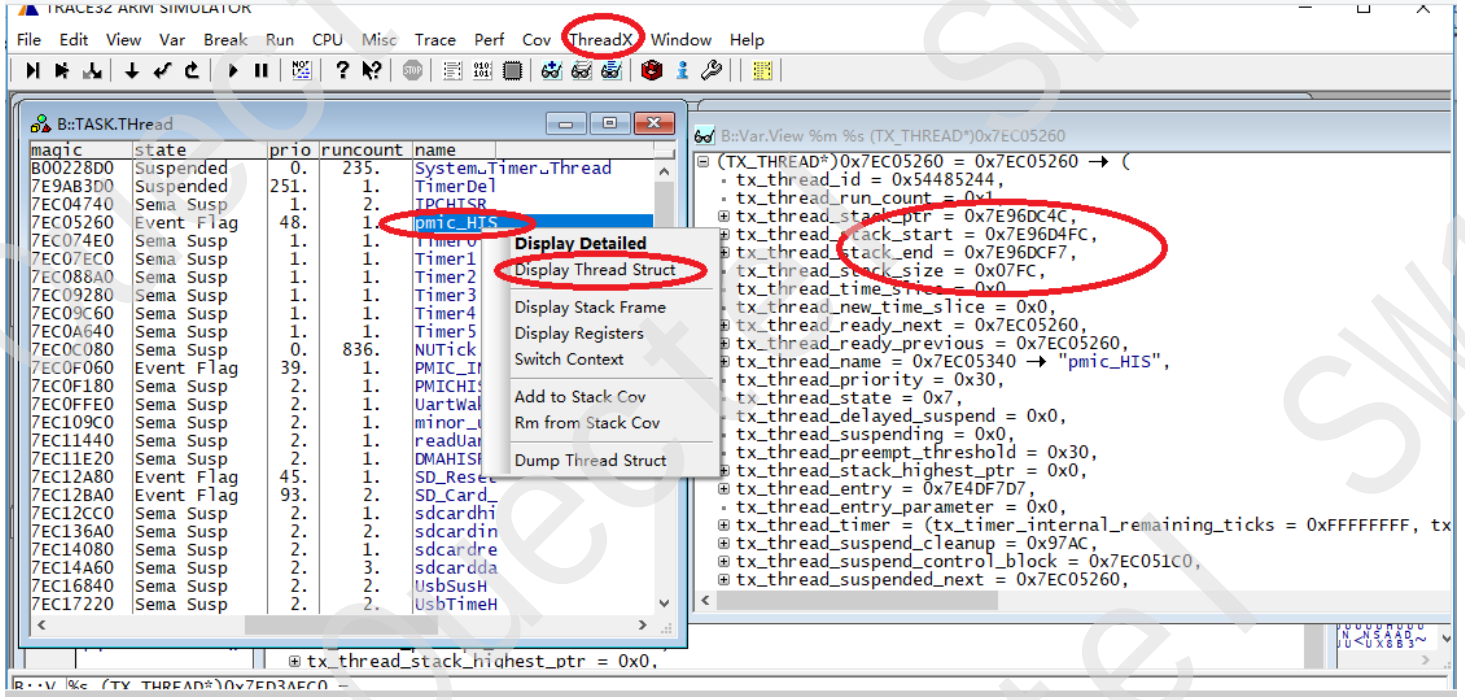
线程当前的寄存器值

导致线程切出的函数

Memdump的作用

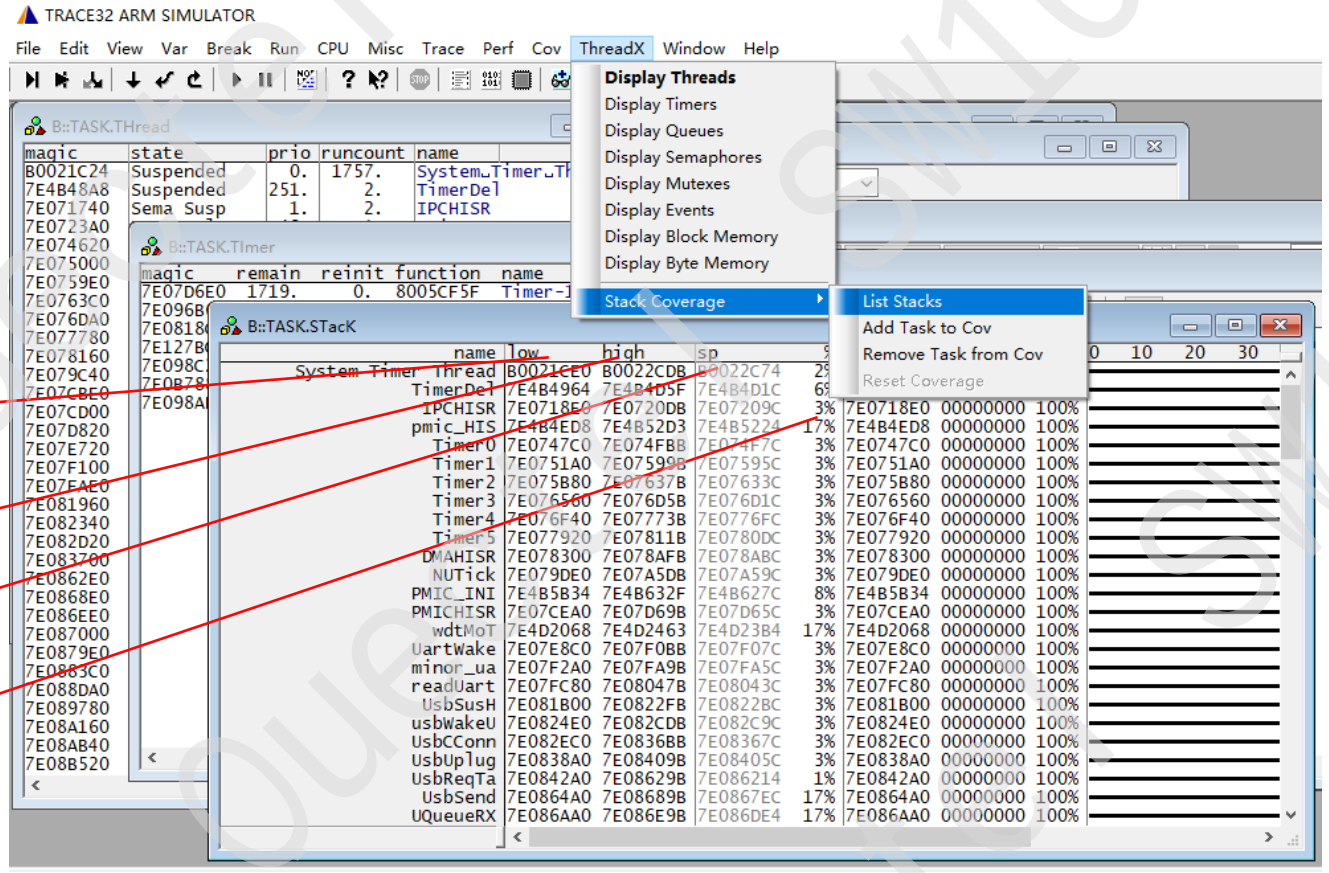
6.查看任务各项指标

将鼠标放在任务名上右击，点击Display Thread Struct可以查看task tcb信息，例如优先级/任务状态/运行次数/栈空间/栈指针等



Memdump的作用

6.查看任务各项指标---任务栈参数

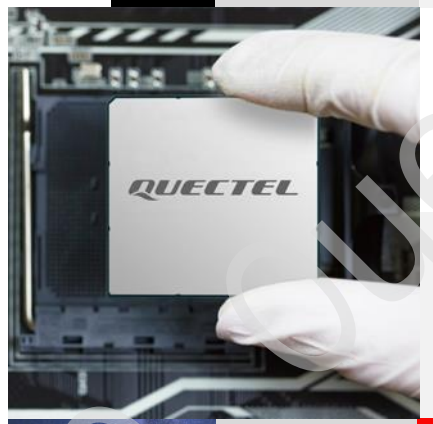


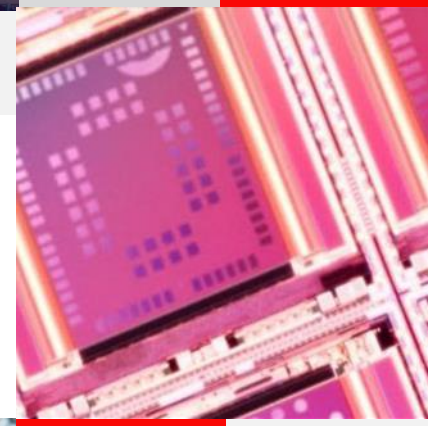
一 memdump简介

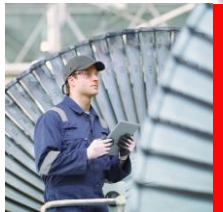
二 触发memdump的异常分类和流程

三 memdump可查看的信息

四 memdump分析实例







Build a Smarter World

www.quectel.com

memdump分析实例-学习心得

学习memdump解析方法1—循序渐进

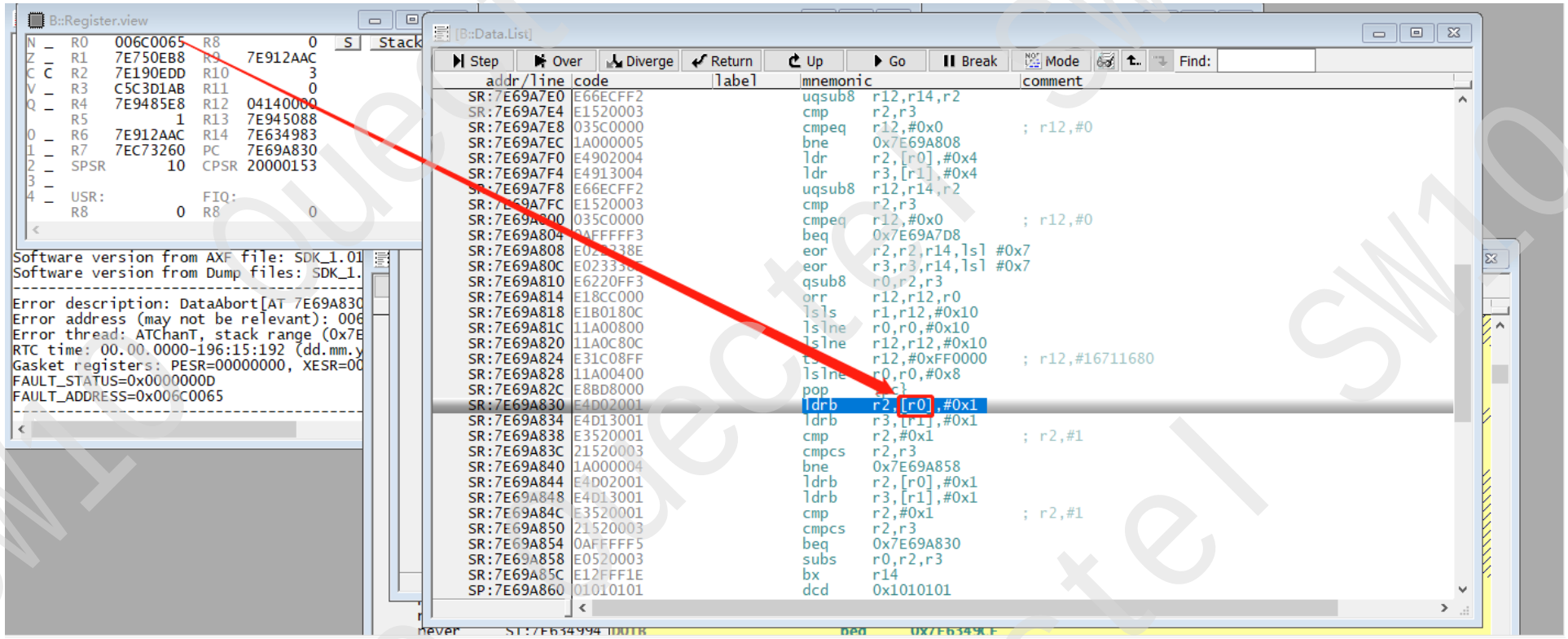
Dump分析所需的知识并非是使用dump工具本身，他需要芯片架构，操作系统，网络，编程语言，编译器等各方面知识，我们无法一次性讲清，这里仅列举出一些常见dump分析的场景，随着其他知识的掌握，bug分析经验增多，对于memdump的理解会更为深刻。
同时等到能熟练分析的那时，我们会对芯片架构，操作系统，网络，编程语言，编译器等这些方面知识，理解更为透彻

学习memdump解析方法2—戒骄戒躁，不抛弃不放弃

遇到复杂问题时，要相信没有什么问题解决不了，不要太看重结果，要注重过程中学习到的知识

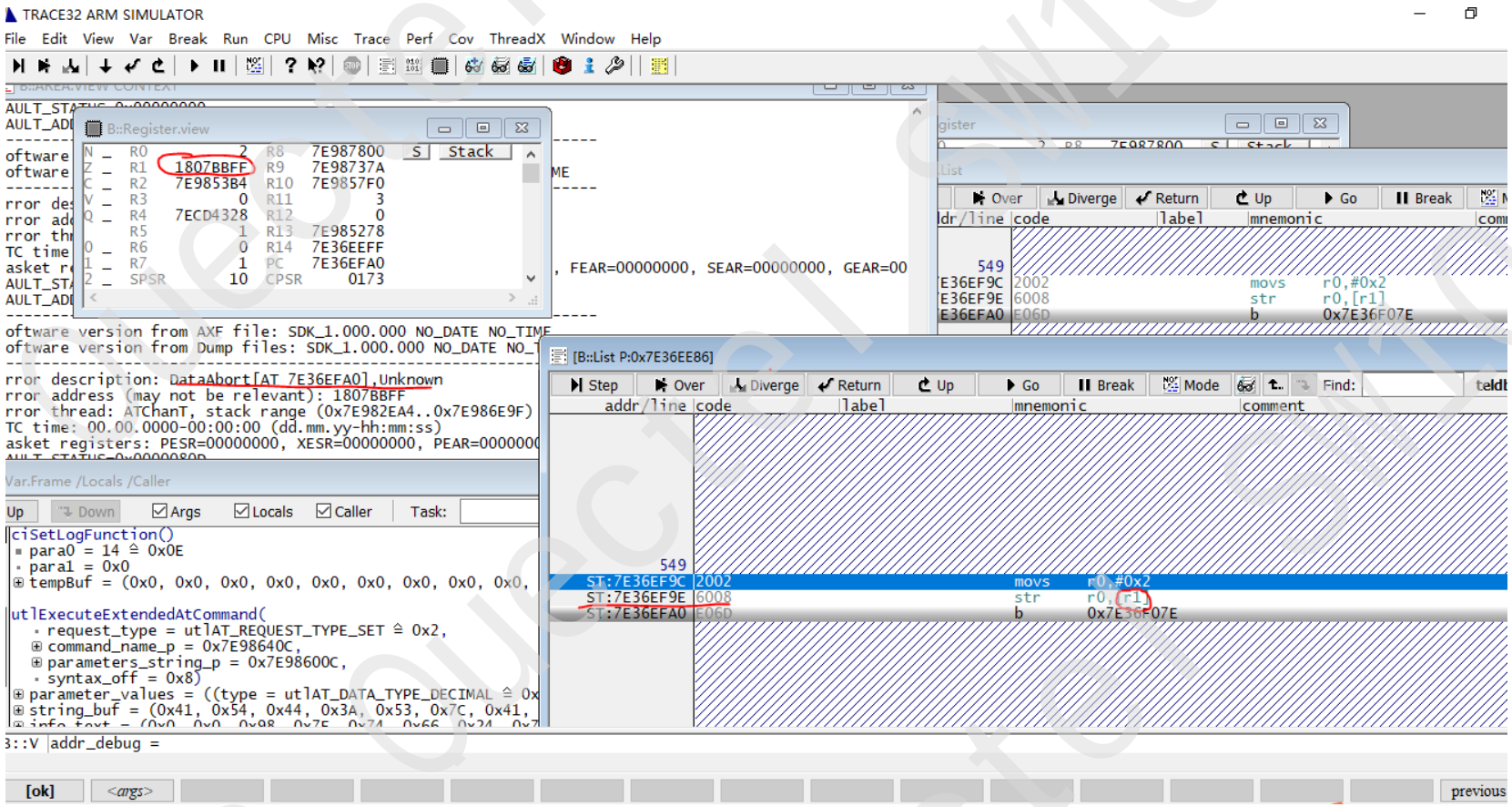
memdump分析实例-data_abort

ldrb　r2,[r0],#0x1，产生了data_abort异常，此指令是将r0指向地址存放的数据存入r2后，r0内地址自动加1，此处的r0的值 0x006C0065是个不存在的地址。



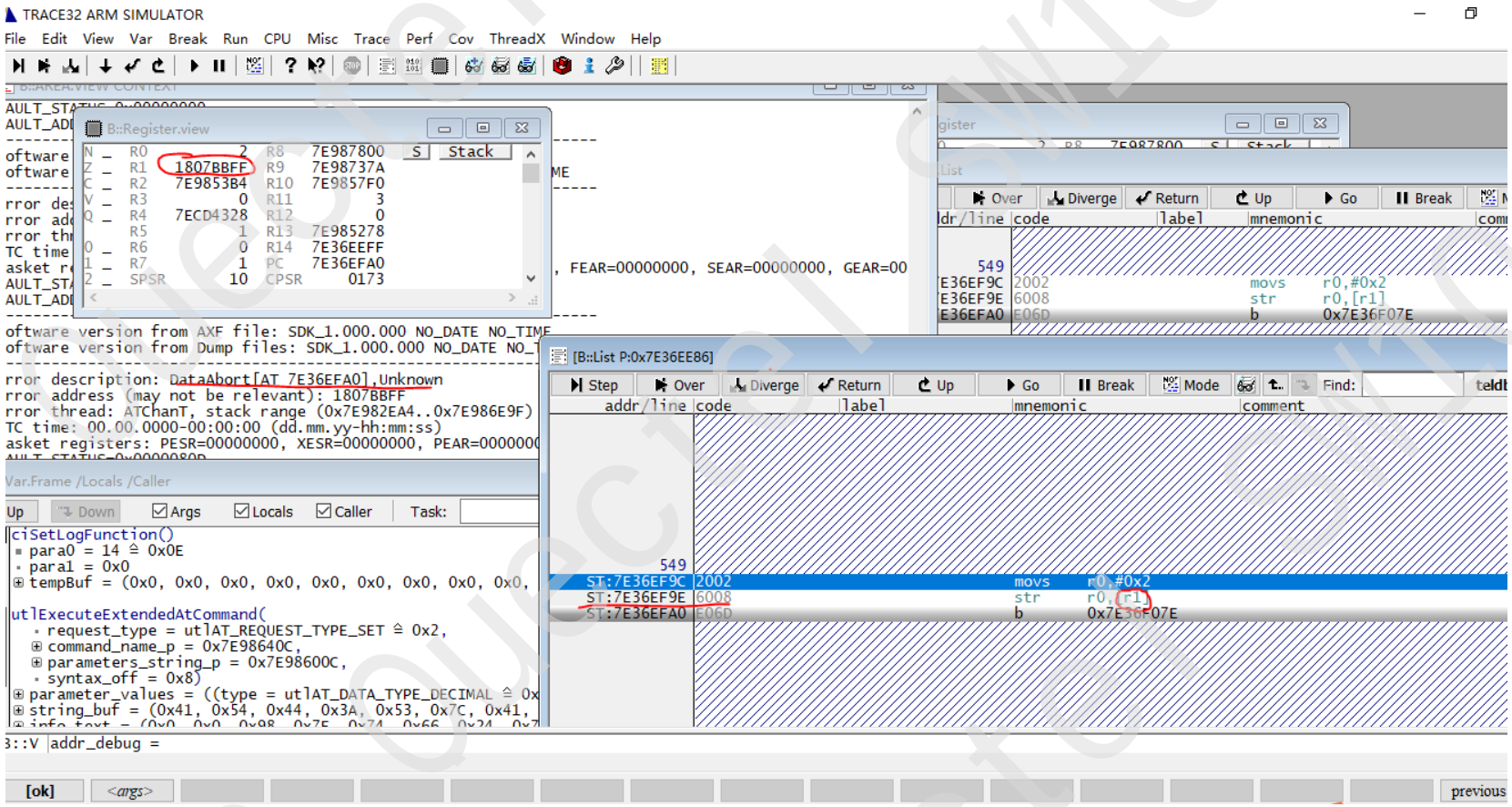
memdump分析实例-data_abort

往r1(0x1807BBFF)非法地址写值触发DUMP。



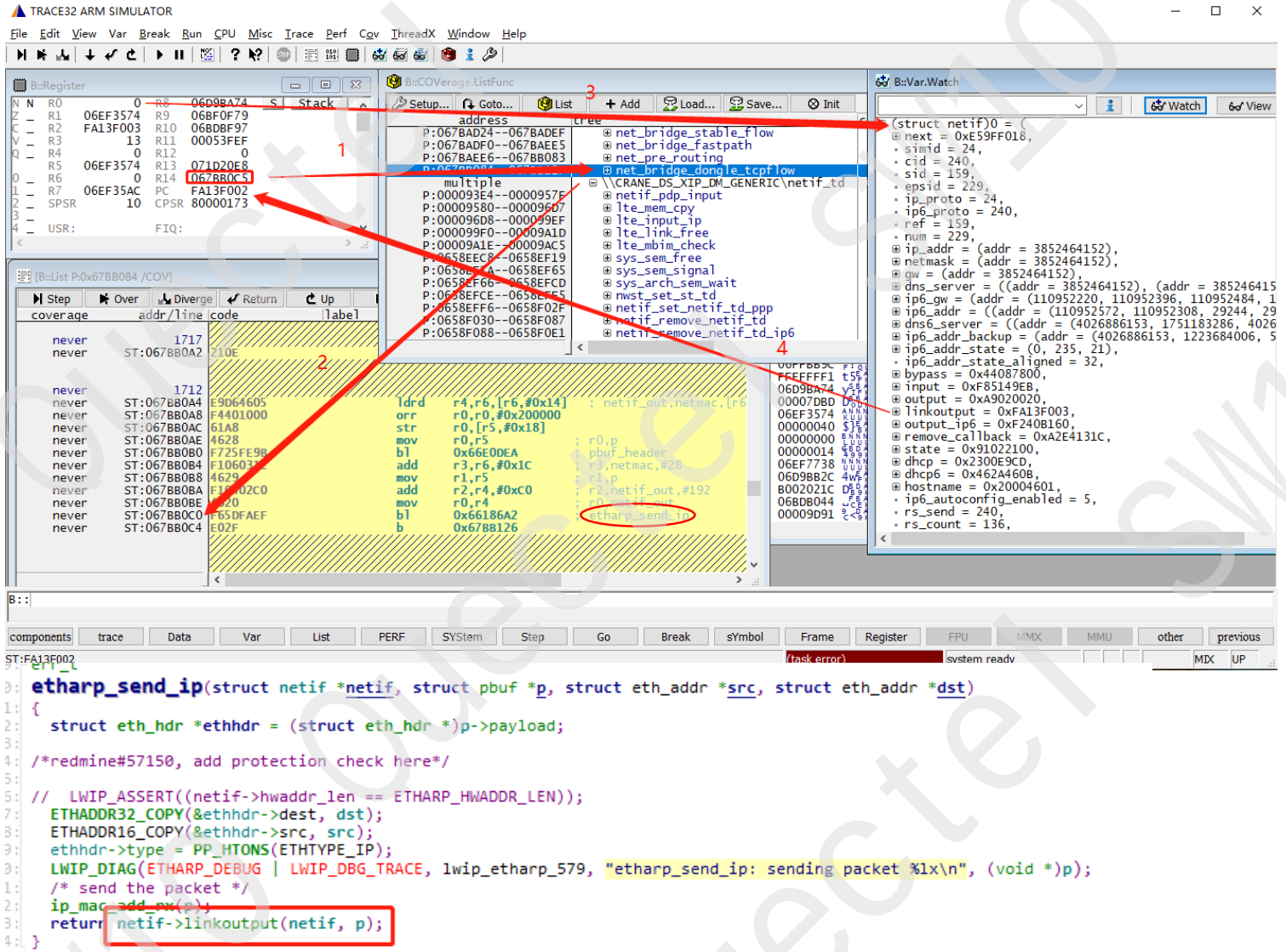
memdump分析实例-data_abort

往r1(0x1807BBFF)非法地址写值触发DUMP。



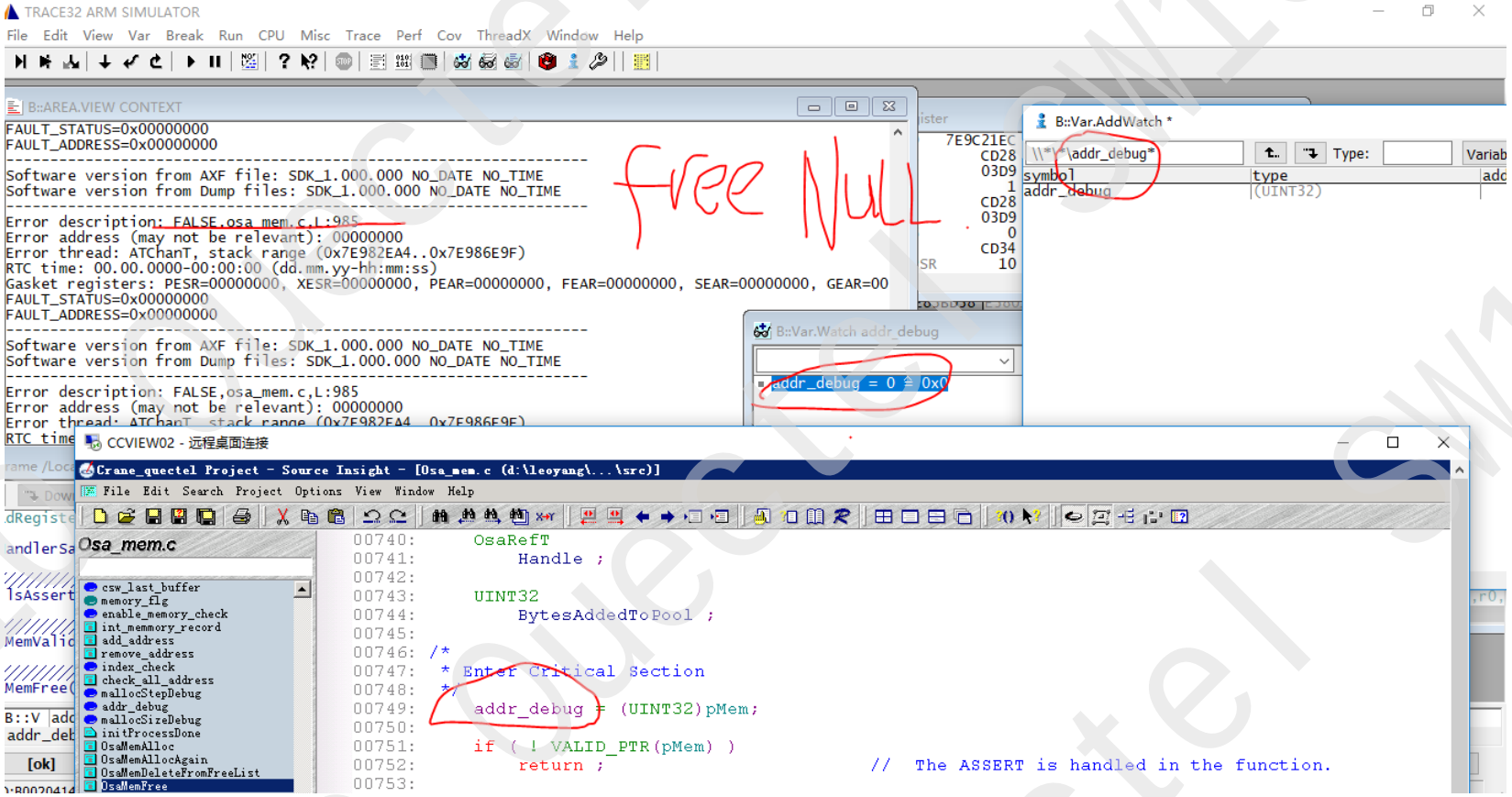
memdump分析实例-PrefetchAbort

PC指针指向0xFA13F002,这个地址并不存在



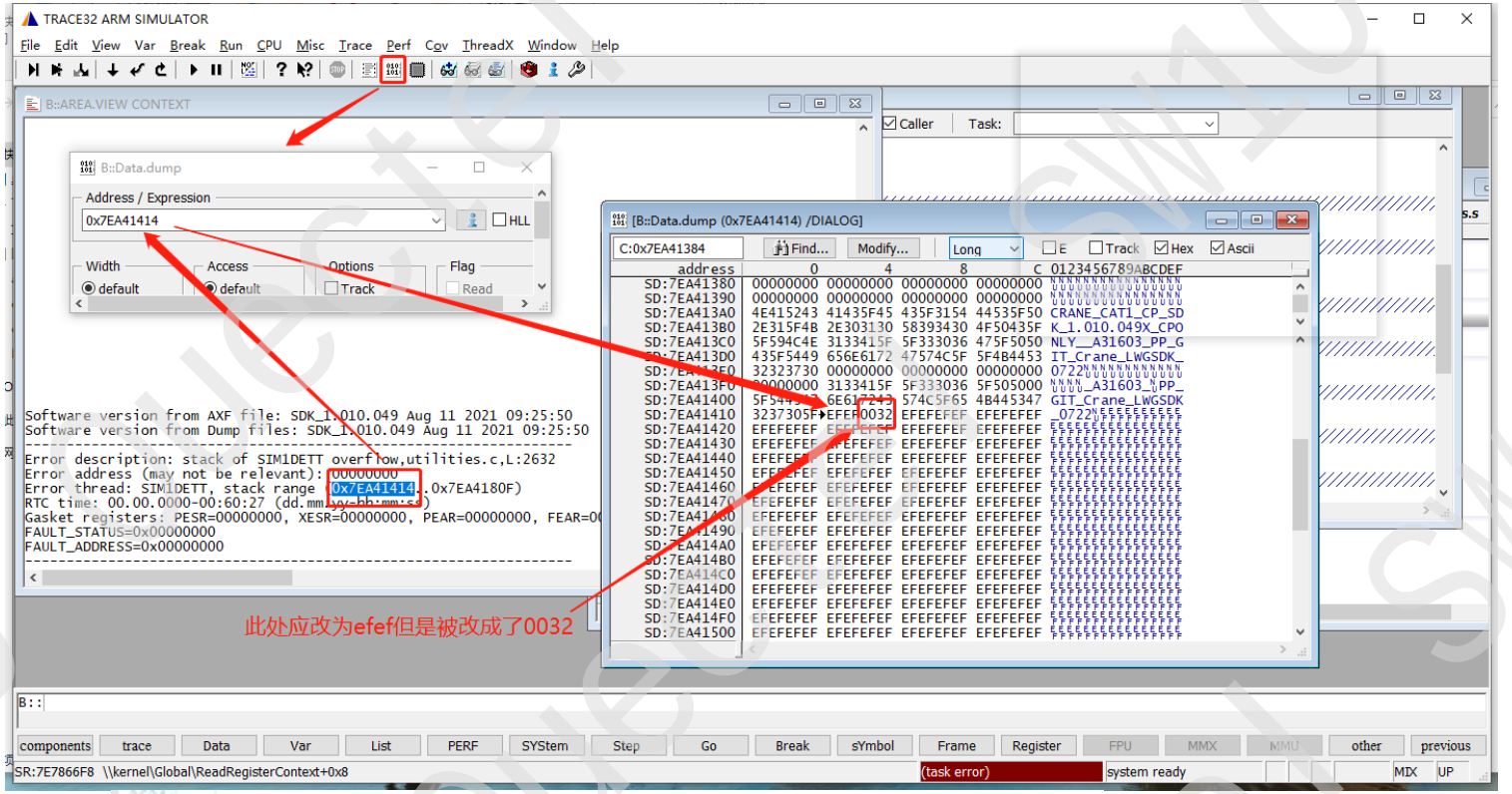
memdump分析实例-ASSERT

通过全局变量可以查到free了一个空指针



memdump分析实例-ASSERT

系统检测到os task栈溢出触发的主动dump:




```
if(thread->tid->tx_thread_id != TX_THREAD_ID)
{
    ASSERT_EXT(0, "Thread 0x%x, Invalid ID 0x%x", thread->tid, thread->tid->tx_thread_i
}

if((stack_magic != 0xEFEFEFEF) || (stack_magic_ != 0xEFEFEFEF))
{
    ASSERT_EXT(0, "stack of %s overflow", thread->tid->tx_thread_name);
}
#endif
```

QUECTEL

移联万物，志高行远

- 拥有更丰富、完整的物联网模组产品组合，一站式满足不同场景和地区的需求
- 推出类型丰富的天线产品，具备全定制天线设计、集成和制造的能力
- 提供云平台管理服务，满足从硬件接入到软件应用的全流程解决方案需求
- 全自动化生产线、测试线，保障产品质量始终如一，同时具有超高性价比
- 建立了业内规模领先的研发团队，为客户提供及时、专业、贴心的技术支持服务
- 持续研发新技术、新产品，率先发布5G、安卓智能、LPWA、C-V2X等产品
- 富有创新精神的国际化专业团队，始终保证“客户第一”

Thank You

Build a Smarter World

全国热线：400 960 7678

上海市闵行区田林路1016号科技绿洲3期（B区）5号楼 200233
联系电话：+86 21 5108 6236 销售支持：sales@quectel.com
技术支持：support@quectel.com 电子邮件：info@quectel.com

移远微信公众号