

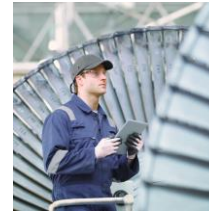
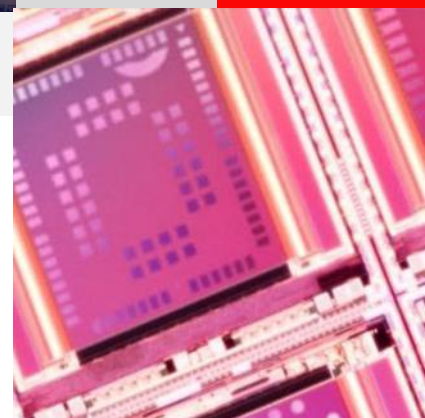
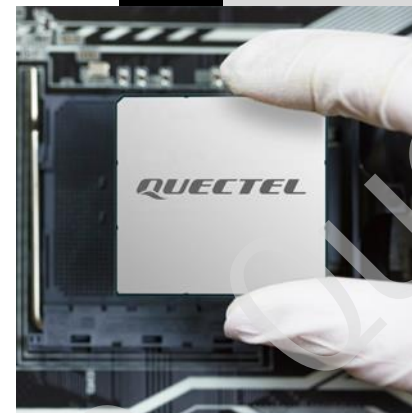


trace32死机内存 分析

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

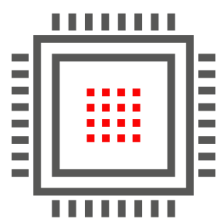


Trace32 simulator简介



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

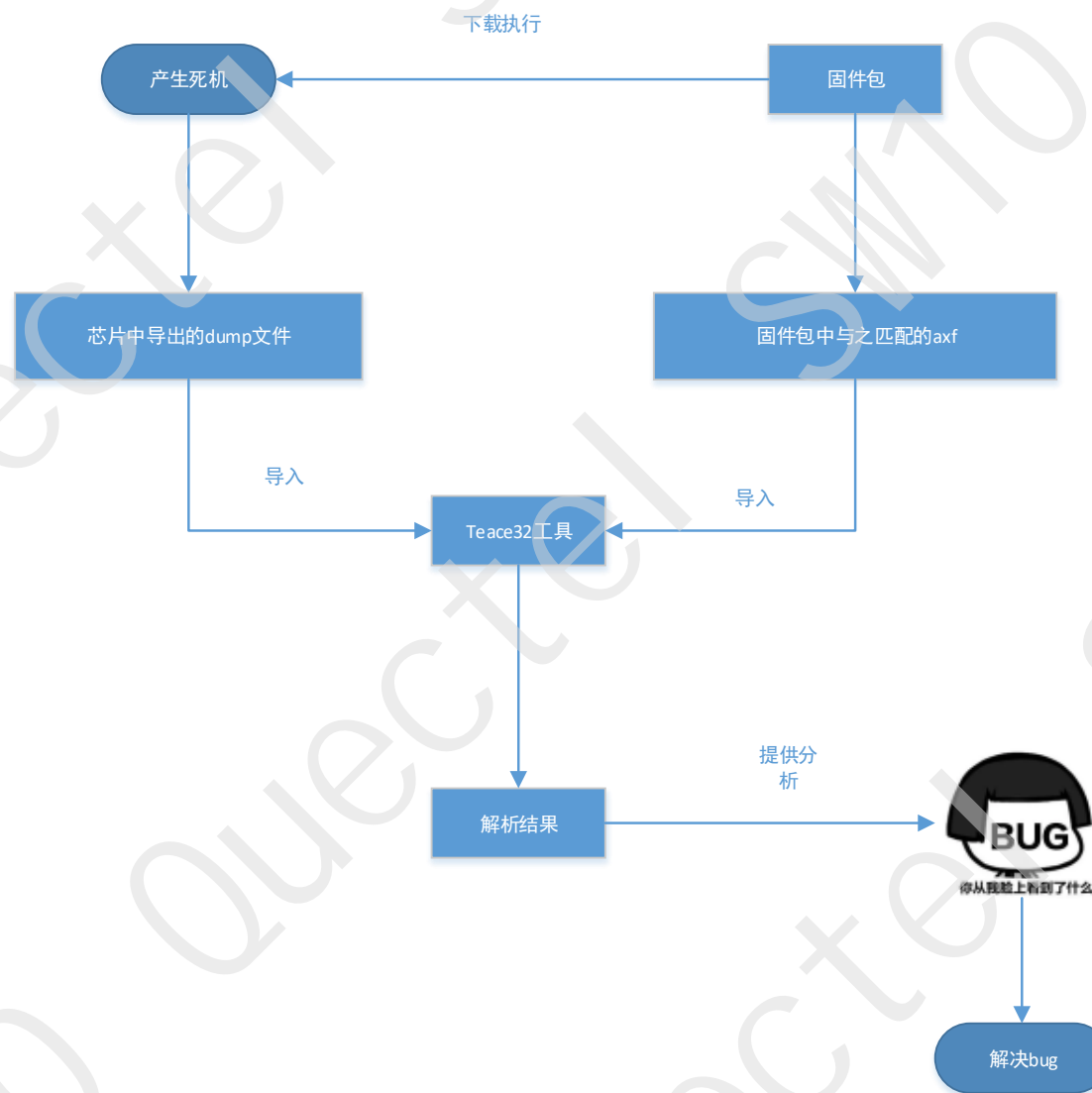


com_compmde.bin	2022/6/14 21:24	UltraEdit Docum...	1 KB
com_CustVer.bin	2022/6/14 21:24	UltraEdit Docum...	1 KB
com_DDR_RW.bin	2022/6/14 21:24	UltraEdit Docum...	7,744 KB
com_DSP_DDR.bin	2022/6/14 21:24	UltraEdit Docum...	448 KB
com_DSPdtcm.bin	2022/6/14 21:24	UltraEdit Docum...	256 KB
com_DTCM.bin	2022/6/14 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_wifiSrv.bin	2022/6/14 21:24	UltraEdit Docum...	2 KB

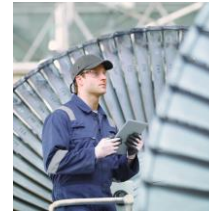
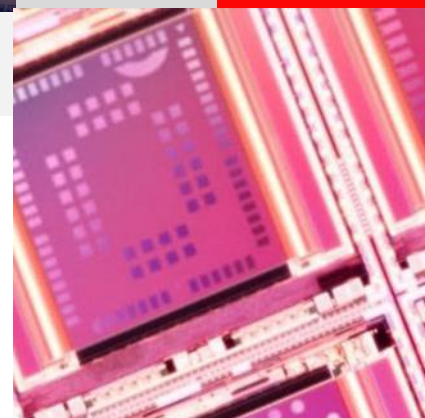
芯片内存信息导出到pc上的文件

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



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readonly, 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。

Undefined 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

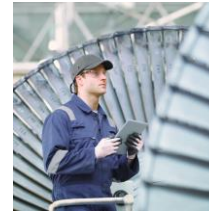
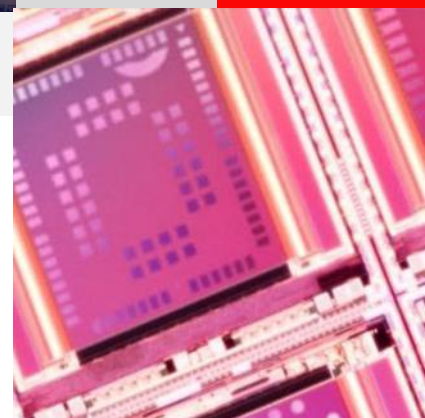
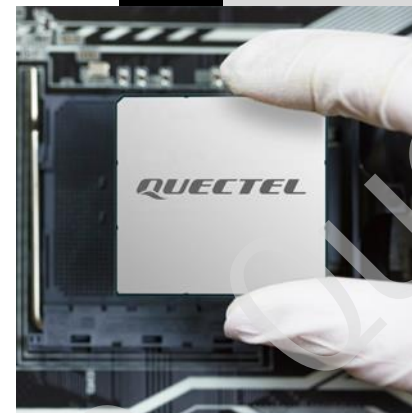
在编写代码时，程序员为防止代码异常执行导致整个系统的崩溃，会对代码加一些异常状态判断，正常情况下代码是不会进入这些异常分支，如果代码进入这些异常状态就会触发主动ASSERT
ASSERT会触发memdump流程，同时将死机代码所在文件和行号信息以及一些状态值保存到内存中。

```

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



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

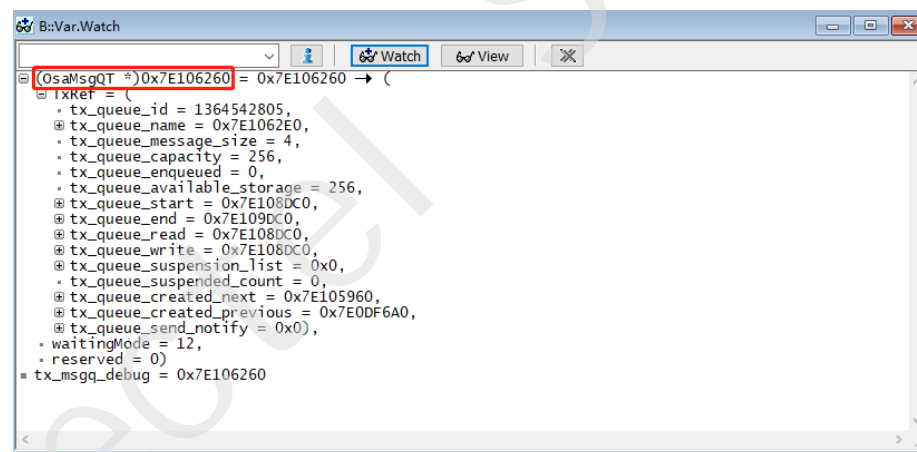
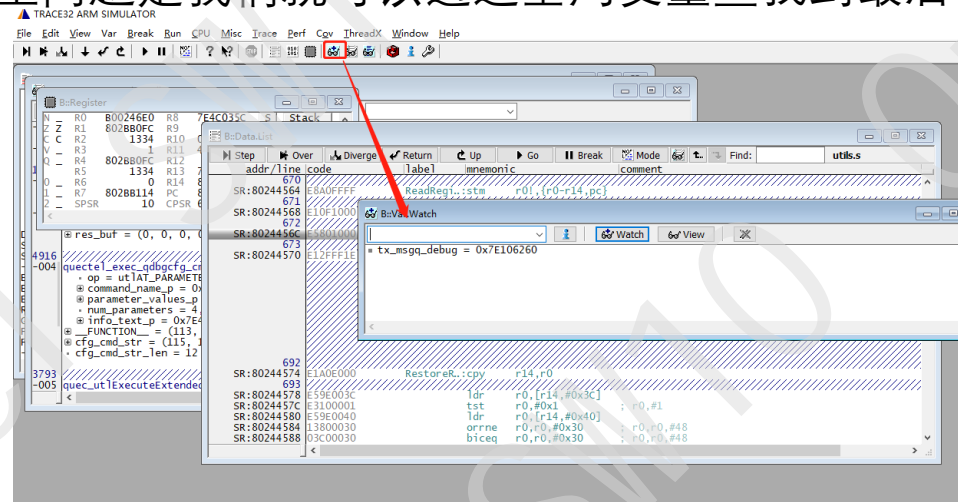
Memdump的作用



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(EHandlerFlag != 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:
```

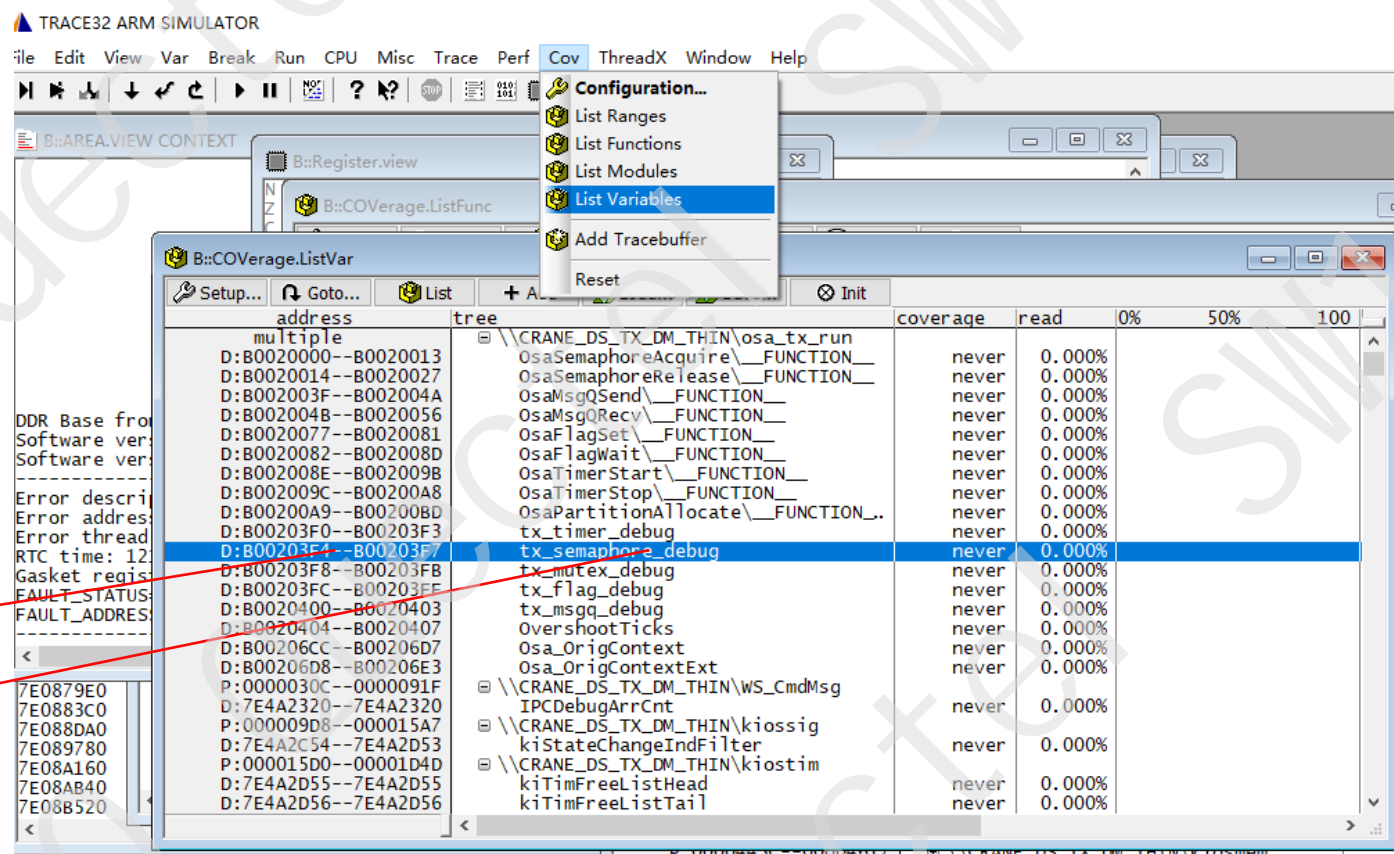


通过给地址强制类型, 可以获取到结构体变量的详细信息

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



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

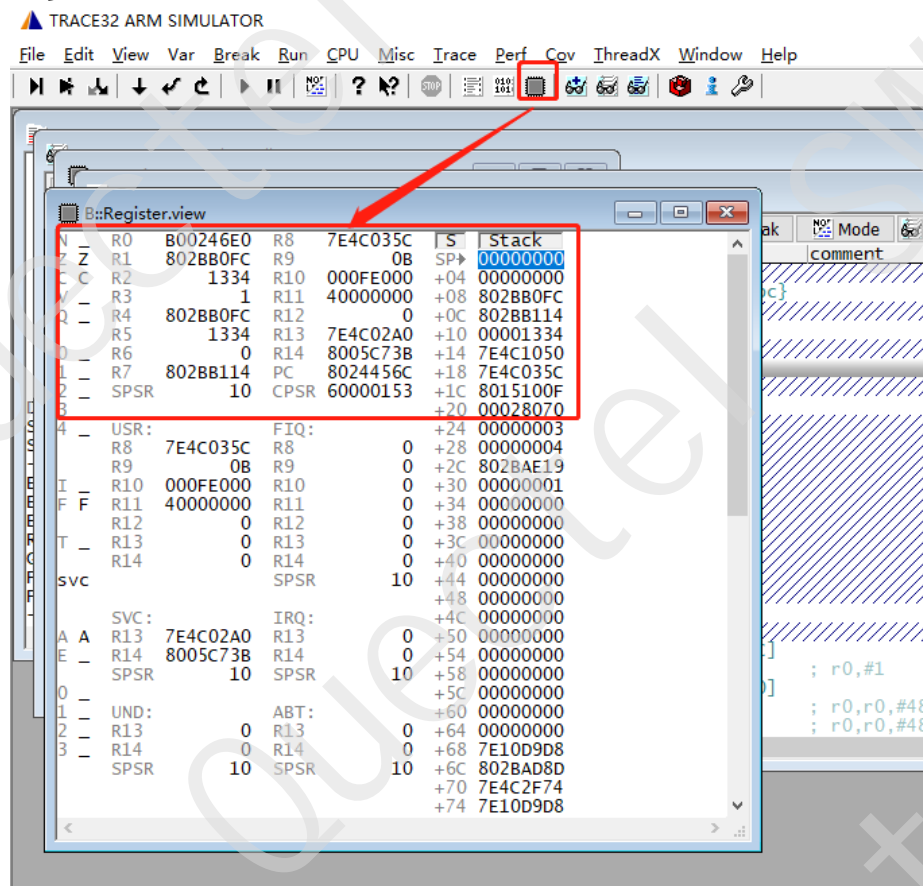
The screenshot shows the TRACE32 ARM SIMULATOR interface. The 'Configuration...' menu is open, and the 'List Functions' dialog box is displayed. The dialog box lists functions with their addresses, coverage, and execution status. The main window displays assembly code for the function OsaFlagSet.

address	tree	coverage	executed
multiple	\\CRANE_DS_TX_DM_THIN\osa_tx_run	never	0.000%
P:00000040--00000043	OsaTick	never	0.000%
P:00000044--00000049	GetOvershootTicks	never	0.000%
P:0000004A--00000051	ResetOvershootTicks	never	0.000%
P:00000052--000000BD	OsaTickUpdate	never	0.000%
P:000000BE--000000C1	OsaTickSuspend	never	0.000%
P:000000C2--000000C5	OsaGetTicks	never	0.000%
P:000000C6--000000C9	OsaTaskYield	never	0.000%
P:000000CA--000000CD	OsaCriticalSectionEnter	never	0.000%
P:000000CE--000000D0	OsaCriticalSectionExit	never	0.000%
P:000000D1--000000D9	OsaTimerStart	never	0.000%
P:000000DA--000000E8	OsaTimerStop	never	0.000%
P:000000E9--000000F0	OsaContextLock	never	0.000%
P:000000F1--000000F4	OsaContextRestore	never	0.000%
P:000000F5--000000F8	OsaFlagSet	never	0.000%
P:000000F9--000000FA	OsaFlagWait	never	0.000%
P:000000FB--000000FC	OsaPartitionAllocate	never	0.000%
P:000000FD--000000FE	OsaPartitionFree	never	0.000%
P:000000FF--00000100	OsaPartitionGetPoolUsedCount	never	0.000%
P:00000101--00000104	OsaMsgQSend	never	0.000%
P:00000105--00000108	OsaMsgQRecv	never	0.000%
P:00000109--00000112	OsaMutexLock	never	0.000%
P:00000113--00000116	OsaMutexUnlock	never	0.000%
P:00000117--00000120	OsaSemaphoreAcquire	never	0.000%
P:00000121--00000124	OsaSemaphoreCheck	never	0.000%
P:00000125--00000128	OsaSemaphoreRelease	never	0.000%
P:00000129--00000132	\\CRANE_DS_TX_DM_THIN\WS_CmdMsg	never	0.000%

coverage	addr/line	code	label	mnemonic	comment
never	956	B53E	OsaFlagSet::push	{r1-r5,r14}	
never	ST:0000A304				
never	960	4B3A		ldr r3,0xA3F0	
never	ST:0000A306	681B		ldr r3,[r3]	
never	ST:0000A308	2B01		cmp r3,#0x1	
never	ST:0000A30C	D001		beq 0xA312	
never	962	4B39		ldr r3,0xA3F4	
never	ST:0000A30E	60D8		str r0,[r3,#0x0C]	
never	964	2A05		cmp r2,#0x5	
never	ST:0000A312	D018		beq 0xA348	
never	964	2200		movs r2,#0x0	
never	ST:0000A316	F7CED14		blx 0x6D44	; \$Ven\$AA\$LS\$txe_event_f
never	ST:0000A318	0004		movs r4,r0	
never	ST:0000A31C	D015		beq 0xA34C	
never	972	AA02		add r2,sp,#0x8	
never	ST:0000A320	4601		mov r1,r0	
never	ST:0000A322	4834		ldr r0,0xA3F8	
never	ST:0000A324	F7CFE40		blx 0x71A8	; \$Ven\$AT\$LS\$osa_Translat
never	ST:0000A326	B150		cbz r0,0xA342	
never	ST:0000A32A				

Memdump的作用

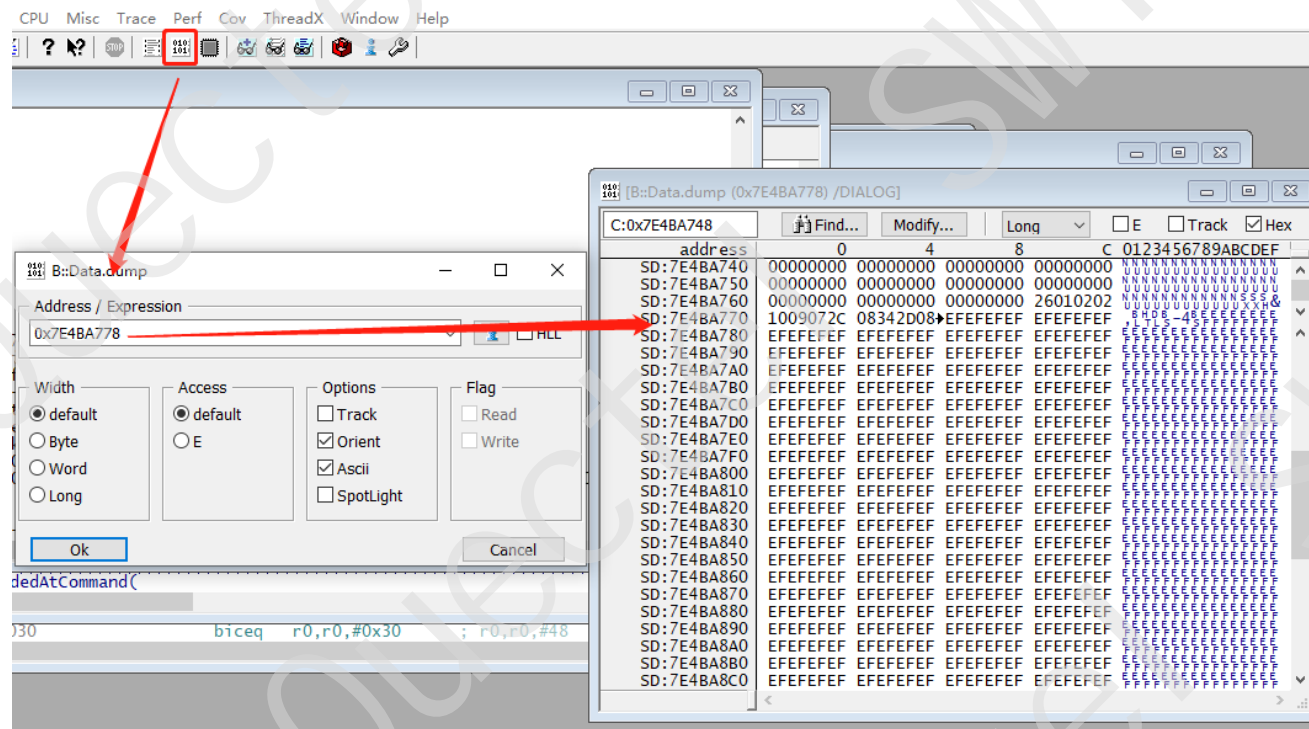
4.查看死机时的r0-r15, cpsr, spsr寄存器的值, 再结合反汇编代码我们可以反推代码参数运算和传递是否存在错误, 找到错误原因



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

TRACE32 ARM SIMULATOR

magic	state	prio	runcount	name
B0021C24	Suspended	0.	1757.	System.Timer.Th
7E4B48A8	Suspended	251.	2.	TimerDel
7E071740	Sema Susp	1.	2.	IPCHISR
7E0723A0	Event Flag	48.	1.	pmic_HIS
7E074620	Sema Susp	1.	1.	Timer0
7E075000	Sema Susp	1.	1.	Timer1
7E0759E0	Sema Susp	1.	1.	Timer2
7E0763C0	Sema Susp	1.	1.	Timer3
7E076DA0	Sema Susp	1.	1.	Timer4
7E077780	Sema Susp	1.	1.	Timer5
7E078160	Sema Susp	2.	432.	DMAHISR
7E079C40	Sema Susp	0.	1907.	NUTick
7E07CBE0	Event Flag	39.	1.	PMIC_INI
7E07CD00	Sema Susp	2.	1.	PMICHISR
7E07D820	Event Flag	50.	1.	wdtMoT
7E07E720	Sema Susp	2.	1.	UartWake
7E07F100	Sema Susp	2.	1.	minor_u
7E07FAE0	Sema Susp	2.	1.	readUart
7E081960	Ready	2.	2.	UsbSush
7E082340	Sema Susp	2.	2.	usbWakeU
7E082D20	Sema Susp	2.	2.	UsbCConn
7E083700	Sema Susp	2.	1.	UsbUplog
7E0862E0	Queue Susp	71.	1.	UsbReqTa
7E0868E0	Event Flag	43.	4.	UsbSend
7E086EE0	Event Flag	6.	1.	UQueueRX
7E087000	Sema Susp	2.	1.	UsbReqH
7E0879E0	Sema Susp	2.	1.	RndisRxH
7E0883C0	Sema Susp	2.	1.	RndisTxH
7E088DA0	Sema Susp	2.	1.	MdmORxH
7E089780	Sema Susp	2.	1.	MdmOTxH
7E08A160	Sema Susp	2.	6.	Mdm1RxH
7E08AB40	Sema Susp	2.	5.	Mdm1TxH
7E08B520	Sema Susp	2.	1.	MdmOCTH

Display Threads

- Display Timers
- Display Queues
- Display Semaphores
- Display Mutexes
- Display Events
- Display Block Memory
- Display Byte Memory
- Stack Coverage

定时器

消息队列

信号量

锁

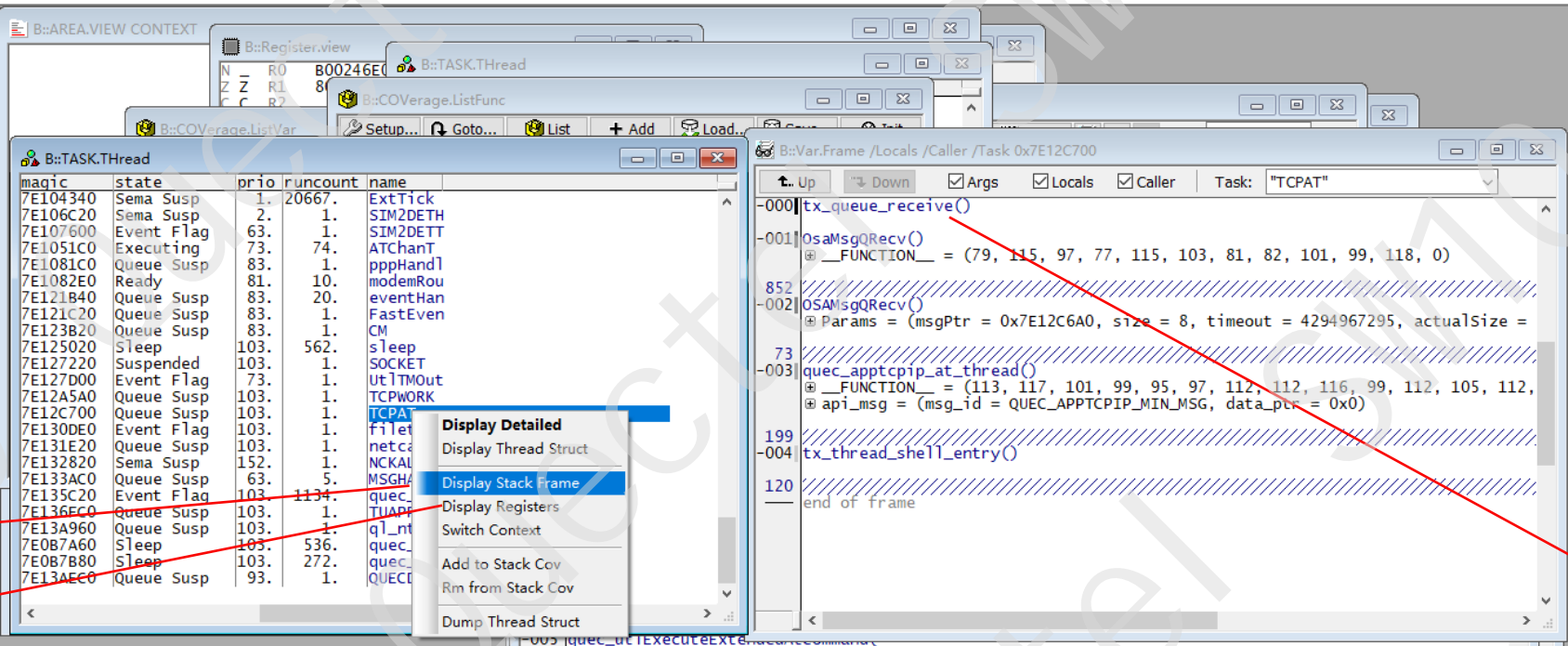
事件

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

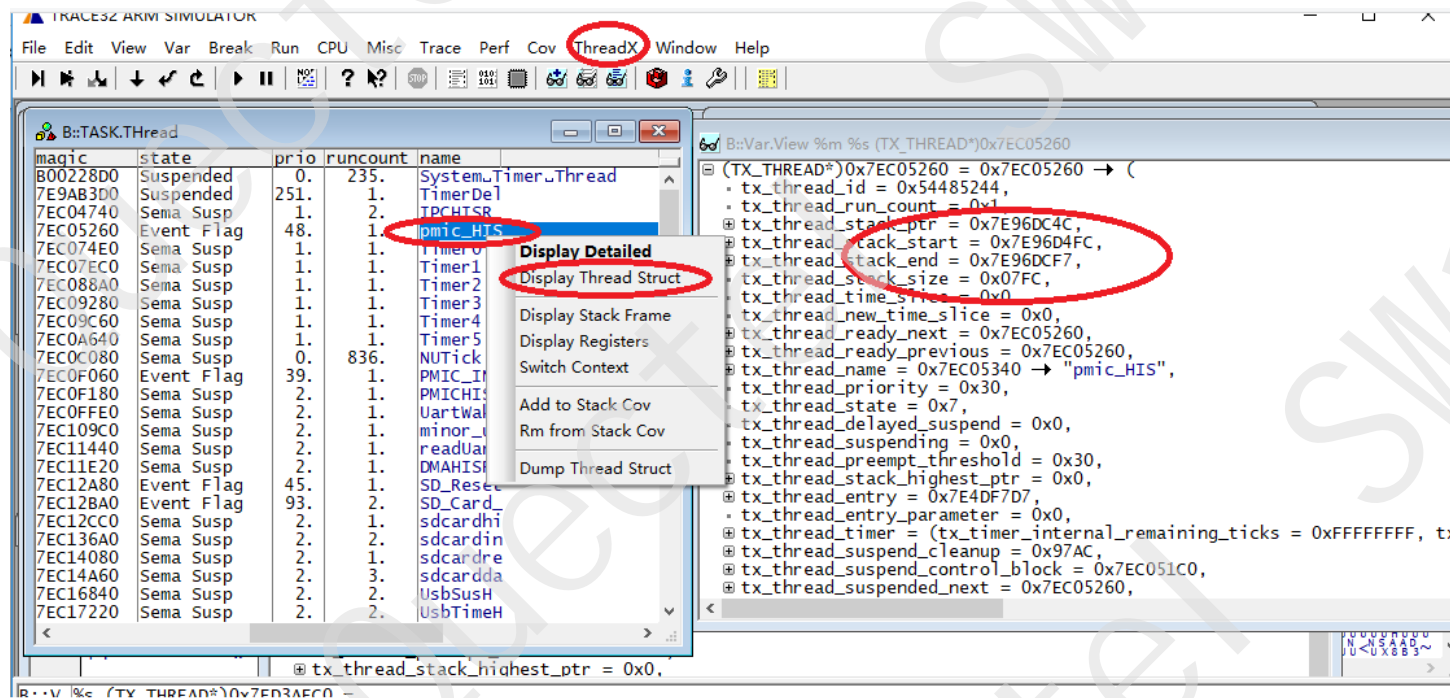
线程当前的寄存器值

导致线程切出的函数

Memdump的作用

6.查看任务各项指标

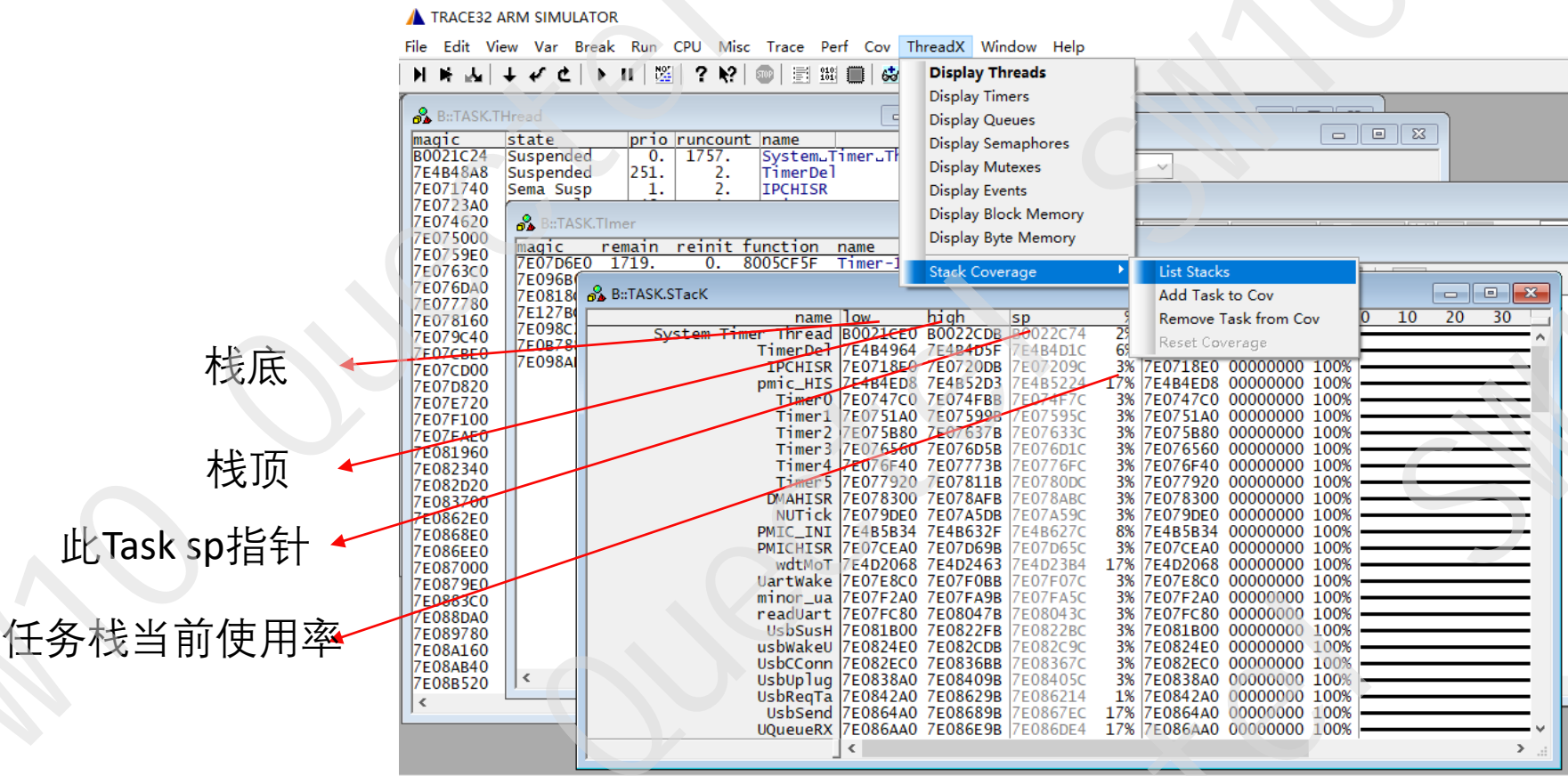
将鼠标放在任务名上右击，点击Display Thread Struct可以查看task tcb信息，例如优先级/任务状态/运行次数/栈空间/栈指针等



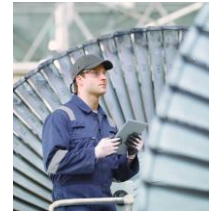
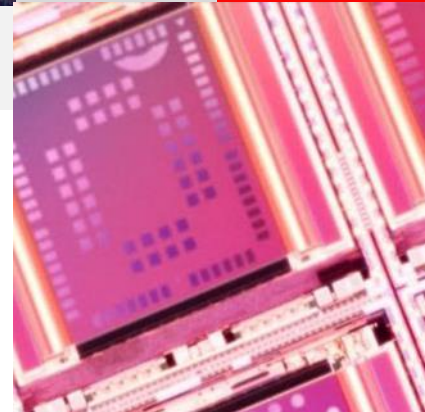
Memdump的作用



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



memdump分析实例-学习心得



学习memdump解析方法1—循序渐进

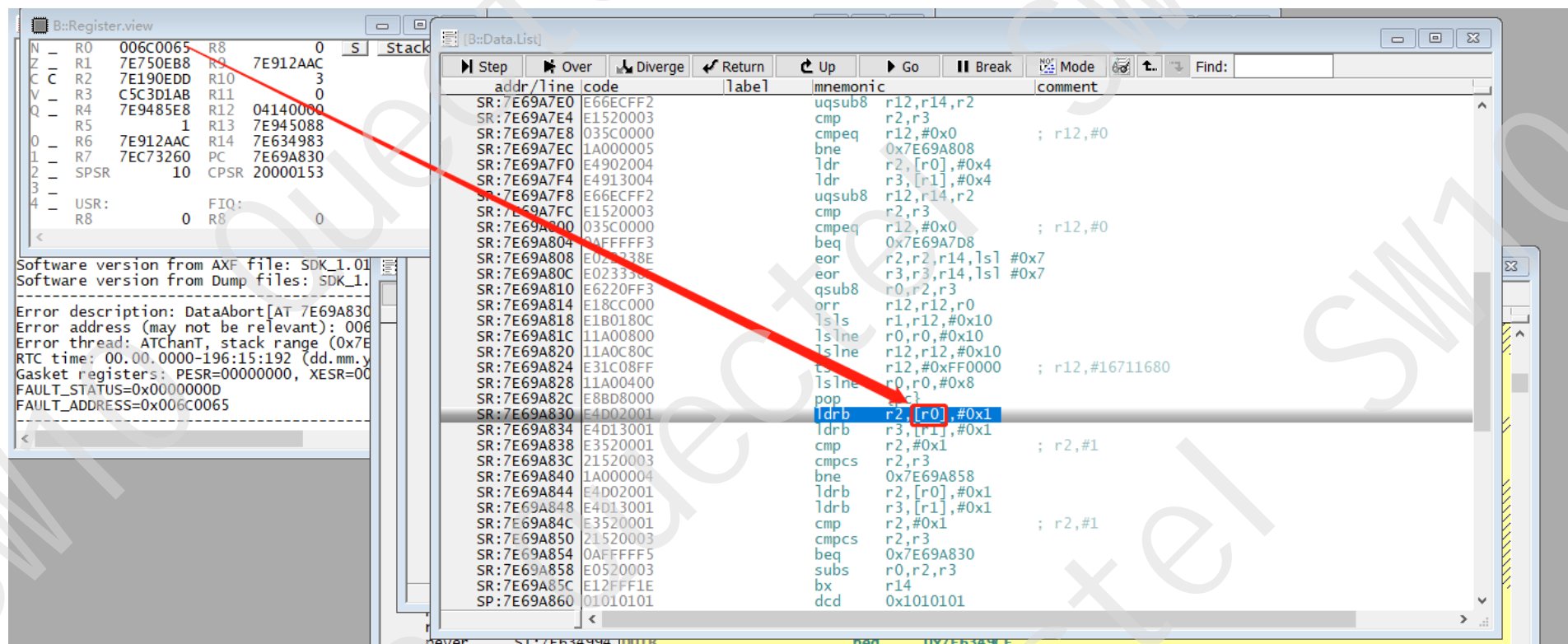
Dump分析所需的知识并非是使用dump工具本身，他需要芯片架构，操作系统，网络，编程语言，编译器等各方面知识，我们无法一次性讲清，这里仅列举出一些常见dump分析的场景，随着其他知识的掌握，bug分析经验增多，对于memdump的理解会更为深刻。同时等到能熟练分析的那时，我们会对芯片架构，操作系统，网络，编程语言，编译器等这些方面知识，理解更为透彻

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

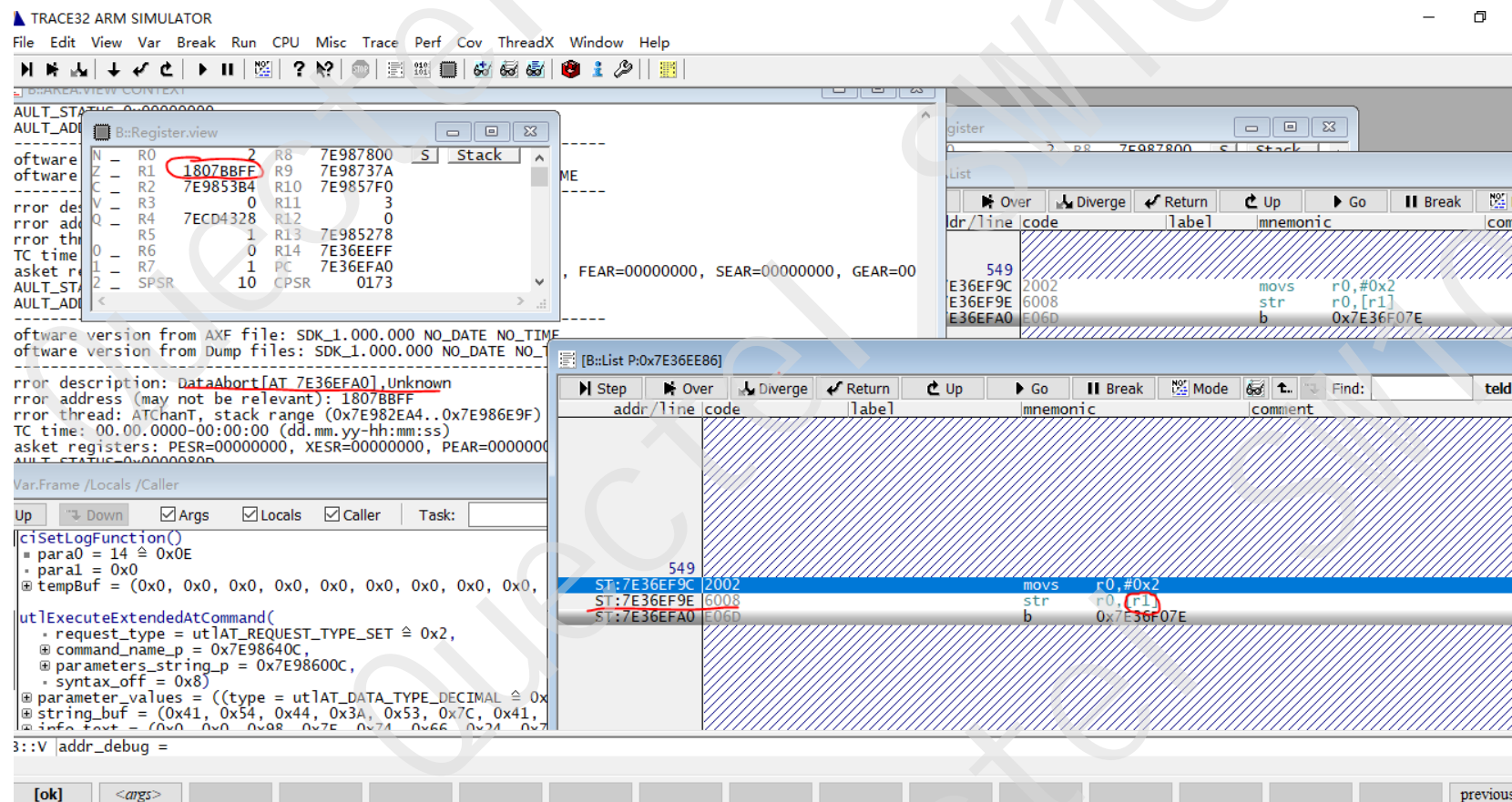
ldrb r2,[r0],#0x1, 产生了data_abort异常, 此指令是将r0指向地址存放的数据存入r2后, r0内地址自动加1, 此处的r0的值 0x006C0065是个不存在的地址。



memdump分析实例-data_abort



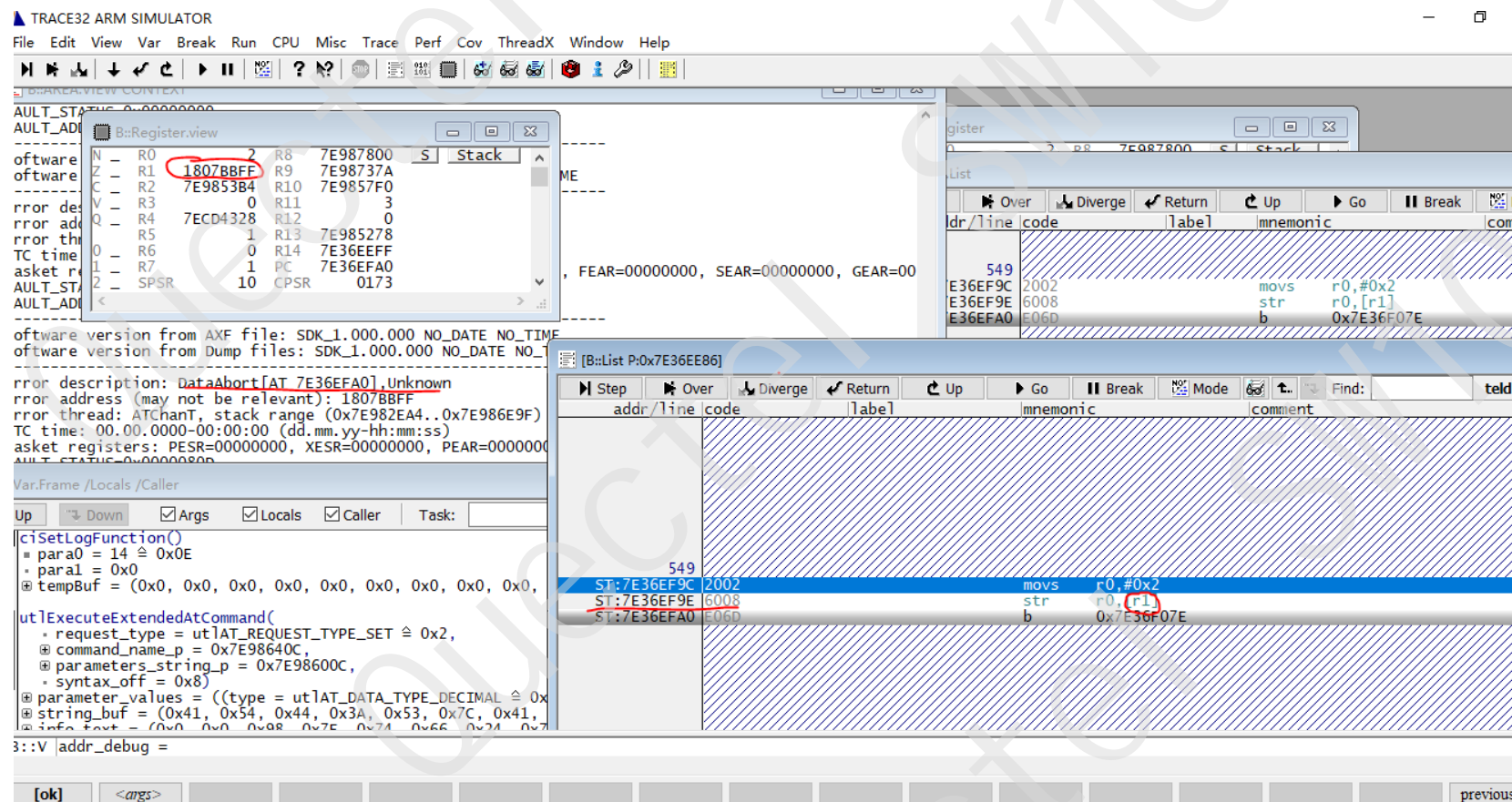
往r1(0x1807BBFF)非法地址写值触发DUMP。



memdump分析实例-data_abort



往r1(0x1807BBFF)非法地址写值触发DUMP。



www.quectel.com



memdump分析实例-ASSERT



通过全局变量可以查到free了一个空指针

The screenshot displays two windows from a development environment. The top window, 'TRACE32 ARM SIMULATOR', shows a memory dump with the following details:

- FAULT_STATUS=0x00000000
- FAULT_ADDRESS=0x00000000
- Software version from AXF file: SDK_1.000.000 NO_DATE NO_TIME
- Software version from Dump files: SDK_1.000.000 NO_DATE NO_TIME
- Error description: FALSE,osa_mem.c,L:985
- Error address (may not be relevant): 00000000
- Error thread: ATChanT, stack range (0x7E982EA4..0x7E986E9F)
- RTC time: 00.00.0000-00:00:00 (dd.mm.yy-hh:mm:ss)
- Gasket registers: PESR=00000000, XESR=00000000, PEAR=00000000, FEAR=00000000, SEAR=00000000, GEAR=00
- FAULT_STATUS=0x00000000
- FAULT_ADDRESS=0x00000000

Handwritten red text 'free NULL' is overlaid on the error description. The bottom window, 'Source Insight', shows the source code for 'Osa_mem.c'. The following code is visible:

```
00740: OsaRefT  
00741: Handle ;  
00742:  
00743: UINT32  
00744: BytesAddedToPool ;  
00745:  
00746: /*  
00747: * Enter Critical Section  
00748: *  
00749: addr_debug = (UINT32)pMem;  
00750:  
00751: if ( ! VALID_PTR(pMem) )  
00752: return ;  
00753:
```

The variable 'addr_debug' is circled in red in the source code. The 'B::Var.Watch addr_debug' window shows the value '0x00000000'.

memdump分析实例-ASSERT



系统检测到os task栈溢出触发的主动dump:

Software version from AXF file: SDK_1.010.049 Aug 11 2021 09:25:50
Software version from Dump files: SDK_1.010.049 Aug 11 2021 09:25:50

Error description: stack of SIMIDETT overflow,utilities.c,L:2632
Error address (may not be relevant): 00000000
Error thread: SIMIDETT, stack range: 0x7EA41414..0x7EA4180F
RTC time: 00.00.0000-00:60:27 (dd.mm.ss.ss)
Gasket registers: PESR=00000000, XESR=00000000, PEAR=00000000, FEAR=00000000
FAULT_STATUS=0x00000000
FAULT_ADDRESS=0x00000000

Address / Expression: 0x7EA41414

Width: default Access: default Options: Track Flag: Read

Memory Dump (0x7EA41414) / DIALOG

address	0	4	8	C	0123456789ABCDEF
SD:7EA41380	00000000	00000000	00000000	00000000	00000000
SD:7EA41390	00000000	00000000	00000000	00000000	00000000
SD:7EA413A0	4E415243	41435F45	435F3154	44535F50	CRANE_CAT1_CP_SD
SD:7EA413B0	2E315F48	2E303130	58393430	4F50435F	K_1_010_049X_CPO
SD:7EA413C0	5F594C4E	3133415F	5F333036	475F5050	NLY_A31603_PP_G
SD:7EA413D0	435F5449	656E6172	47574C5F	5F484453	IT_Crane_LWGSDK
SD:7EA413E0	32323730	00000000	00000000	00000000	07220000
SD:7EA413F0	00000000	3133415F	5F333036	5F505000	0000_A31603_LPP
SD:7EA41400	5F540000	6E617244	574C5F65	4B445347	GIT_Crane_LWGSDK
SD:7EA41410	3237305F	EF0032	EF0032	EF0032	07220000
SD:7EA41420	EF0032	EF0032	EF0032	EF0032	07220000
SD:7EA41430	EF0032	EF0032	EF0032	EF0032	07220000
SD:7EA41440	EF0032	EF0032	EF0032	EF0032	07220000
SD:7EA41450	EF0032	EF0032	EF0032	EF0032	07220000
SD:7EA41460	EF0032	EF0032	EF0032	EF0032	07220000
SD:7EA41470	EF0032	EF0032	EF0032	EF0032	07220000
SD:7EA41480	EF0032	EF0032	EF0032	EF0032	07220000
SD:7EA41490	EF0032	EF0032	EF0032	EF0032	07220000
SD:7EA414A0	EF0032	EF0032	EF0032	EF0032	07220000
SD:7EA414B0	EF0032	EF0032	EF0032	EF0032	07220000
SD:7EA414C0	EF0032	EF0032	EF0032	EF0032	07220000
SD:7EA414D0	EF0032	EF0032	EF0032	EF0032	07220000
SD:7EA414E0	EF0032	EF0032	EF0032	EF0032	07220000
SD:7EA414F0	EF0032	EF0032	EF0032	EF0032	07220000
SD:7EA41500	EF0032	EF0032	EF0032	EF0032	07220000

此处应改为efef但是被改成了0032

```
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_id);
}

if((stack_magic != 0xEF0032) || (stack_magic_ != 0xEF0032))
{
    ASSERT_EXT(0, "stack of %s overflow", thread->tid->tx_thread_name);
}
```




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