系统稳定性分析
-Trace32 手册 V2.0

Jack.Zhao

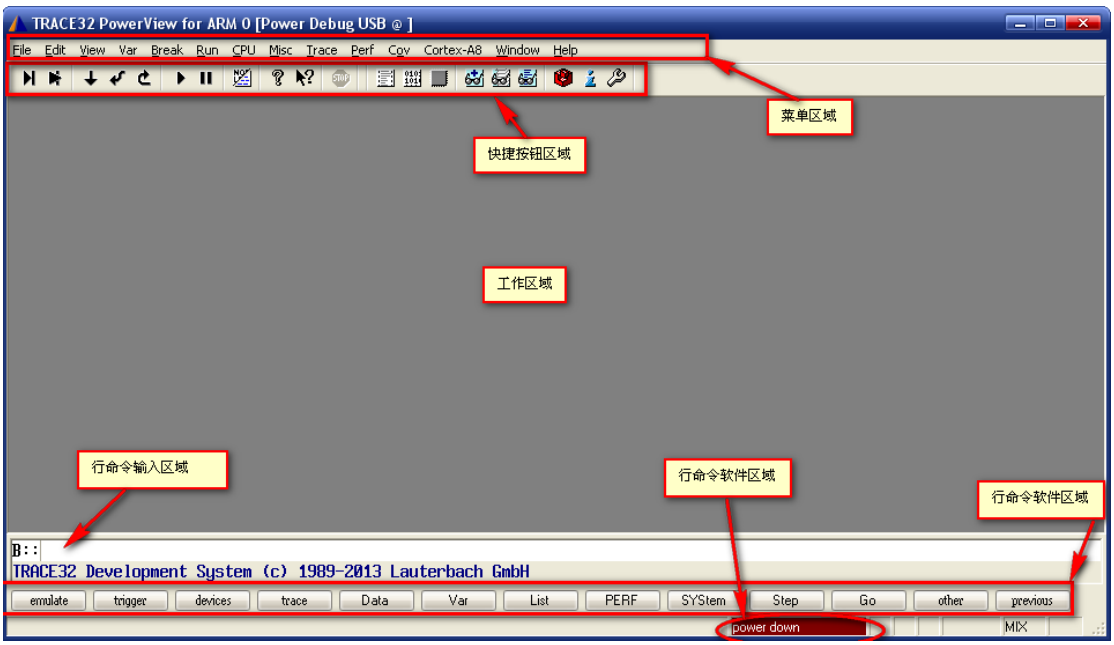
2015.09.15

目录

1 Trace32 分析

手机出现问题时的现场，如定屏、黑屏或其它异常情况，先由研发人员看一下现场，再进行手动 Dump。
现场可以借助于 Trace32 和 ADB 等工具进行现场问题的详细分析。
现场问题主要关注于定屏、黑屏、停在开机画面、花屏、白屏等问题。

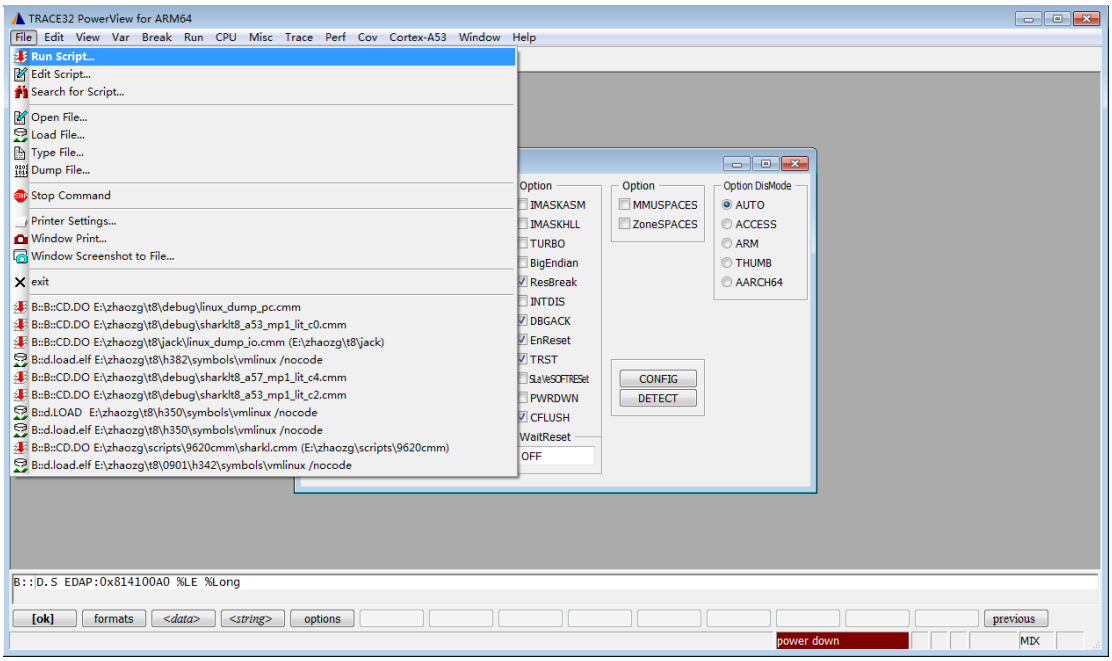
2 Trace32 菜单



界面分成五个区域，用户可以自定义主菜单和快捷按钮。

主菜单区	各种菜单命令的入口区域，用户可以自定义
快捷按钮区	各种常用命令的快捷使用按钮，用户可以自定义
工作区	各种对话框窗口的显示区域
行命令输入区	各种命令通过手动输入执行的区域
行命令软键区	协助用户输入行命令的区域，它提供所有行命令的软键输入方法。
状态显示区	指示当前的调试状态。红圈中的“system down”指示目标板已经供电， 如果目标板电源电压低或没有的话，红圈的区域会显示“POWER DOWN”。

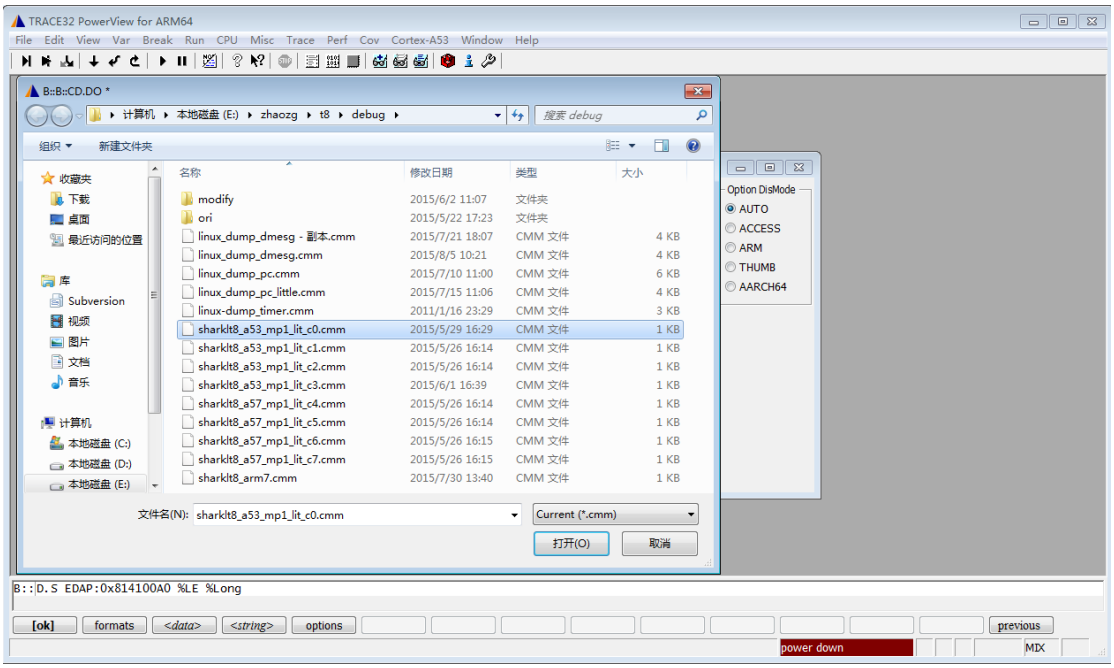
2.1 File 功能



2.1.1 Run Script

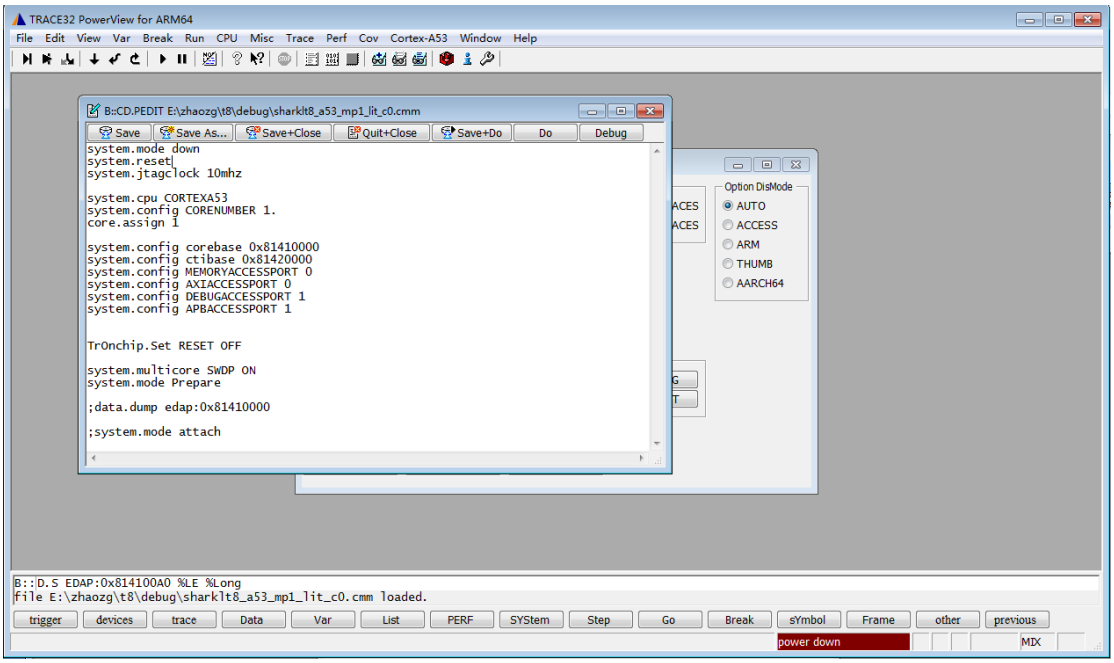
从主菜单区点击"File->Run Script"打开脚本文件选择对话框。如下图所示。

点击“run Batchfile”菜单，会弹出选择脚本的对话框，找到对应脚本的路径，如下图所示。

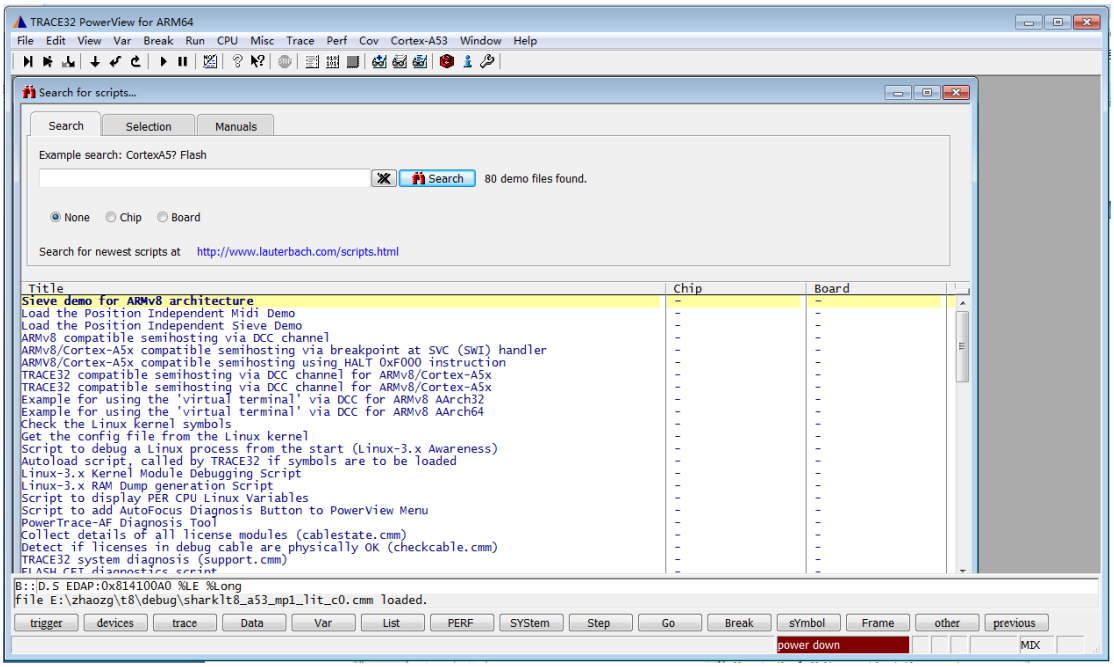


在上图所示的对话框中选择要执行的脚本文件，用户可以选择任意目录下的脚本文件。脚本文件的内容主要以调试命令为主。运行脚本后的界面如下图所示.

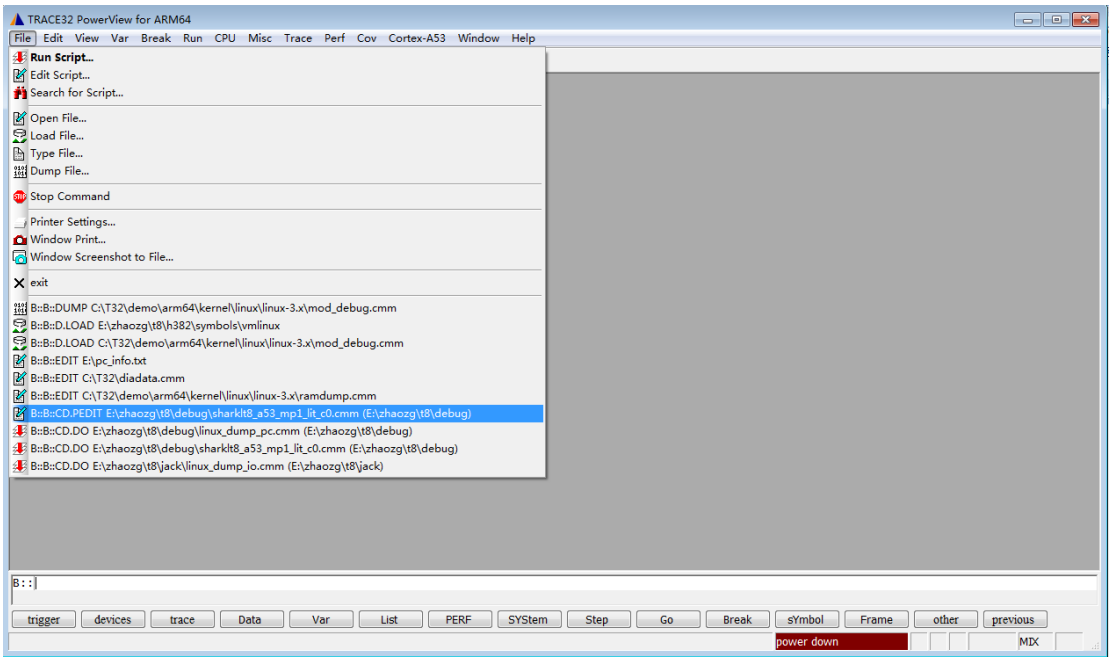
2.1.2 Edit Script



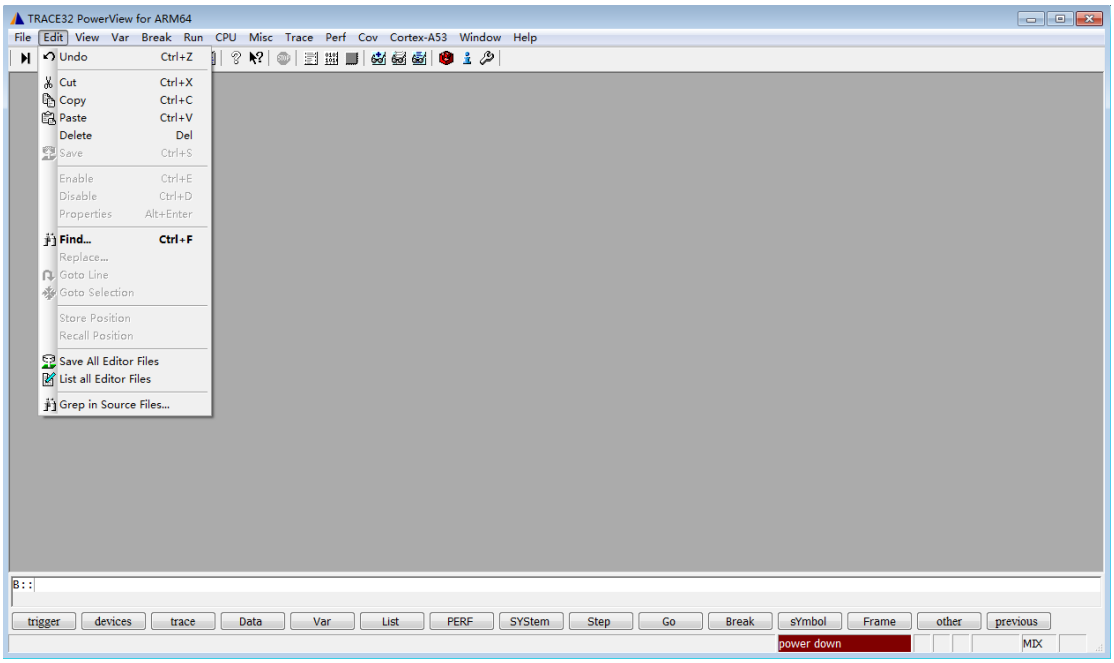
2.1.2 Search for Script



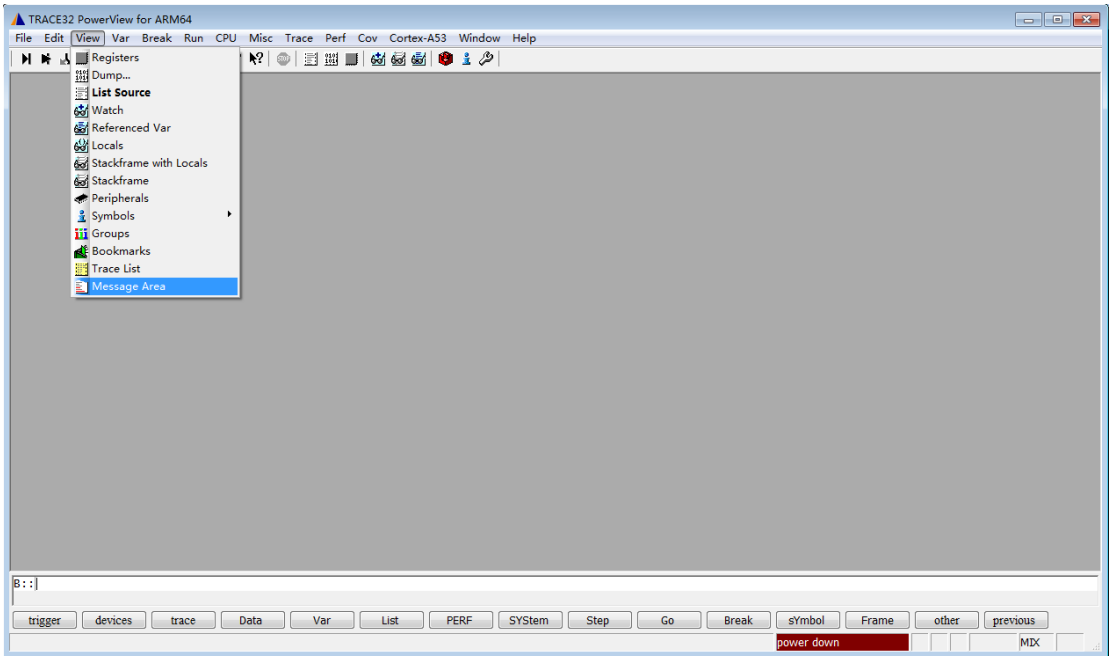
2.1.2 History Script



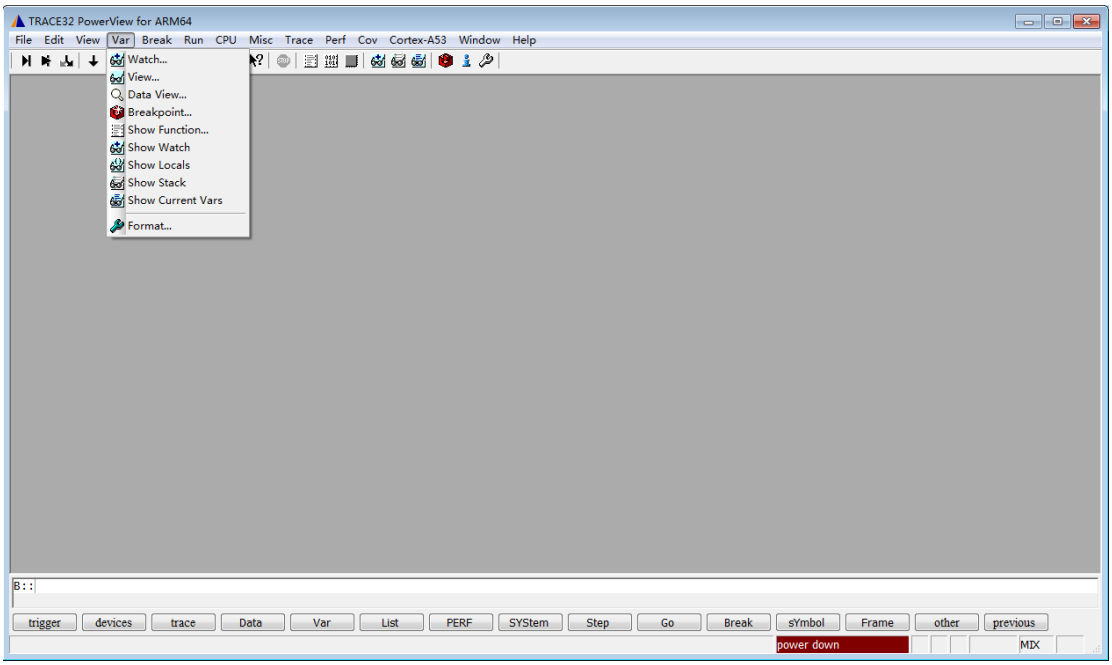
1.1 Edit 功能



1.2 View 功能



1.3 Var 功能

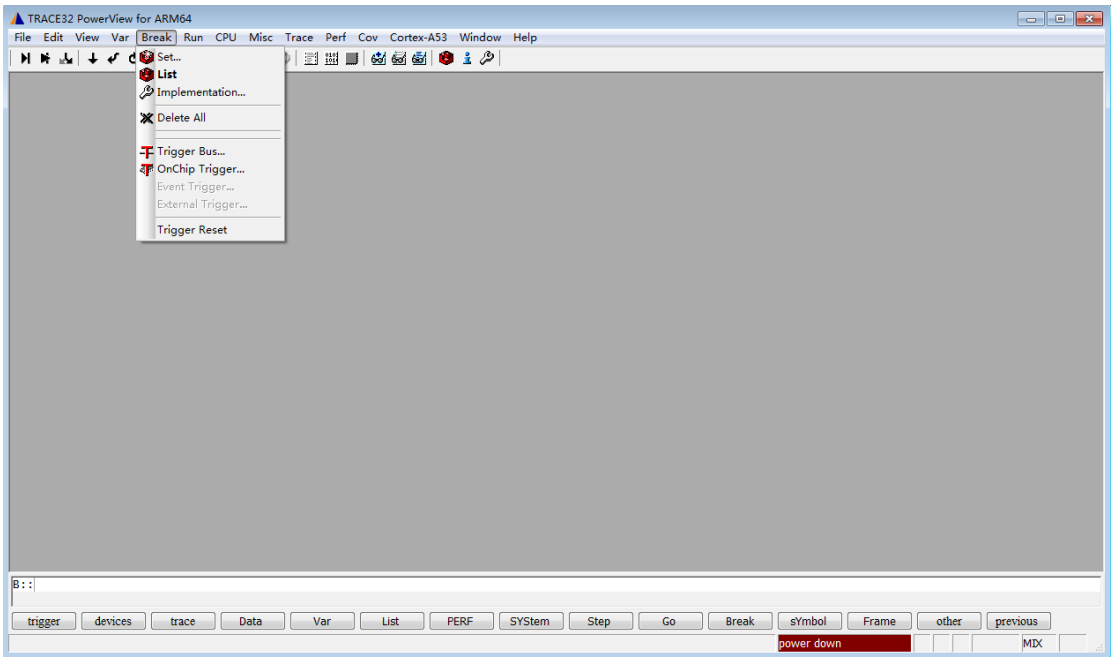


查看函数栈，点击“var->Show Stack”菜单，如下图。弹出函数栈窗口。

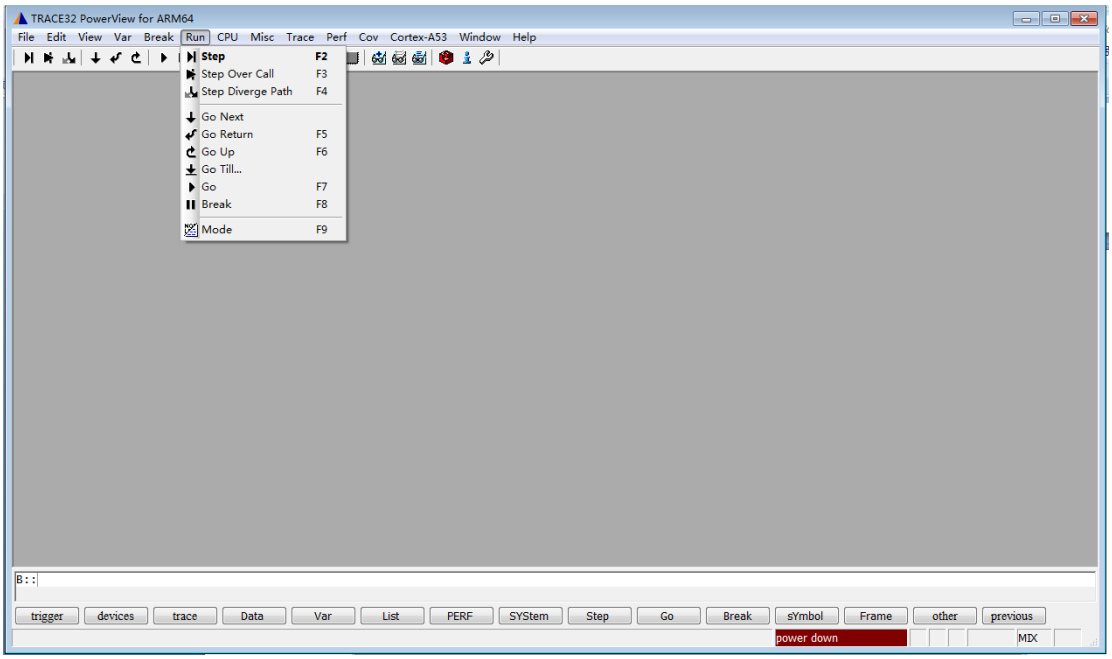
每个cpu都对应一套函数栈。右击状态显示框的“0”，可以选择不同的cpu，在函数栈对话框中也会显示对应cpu的函数栈。

这里最好将各个cpu的函数栈都记录下来，以便后面分析。

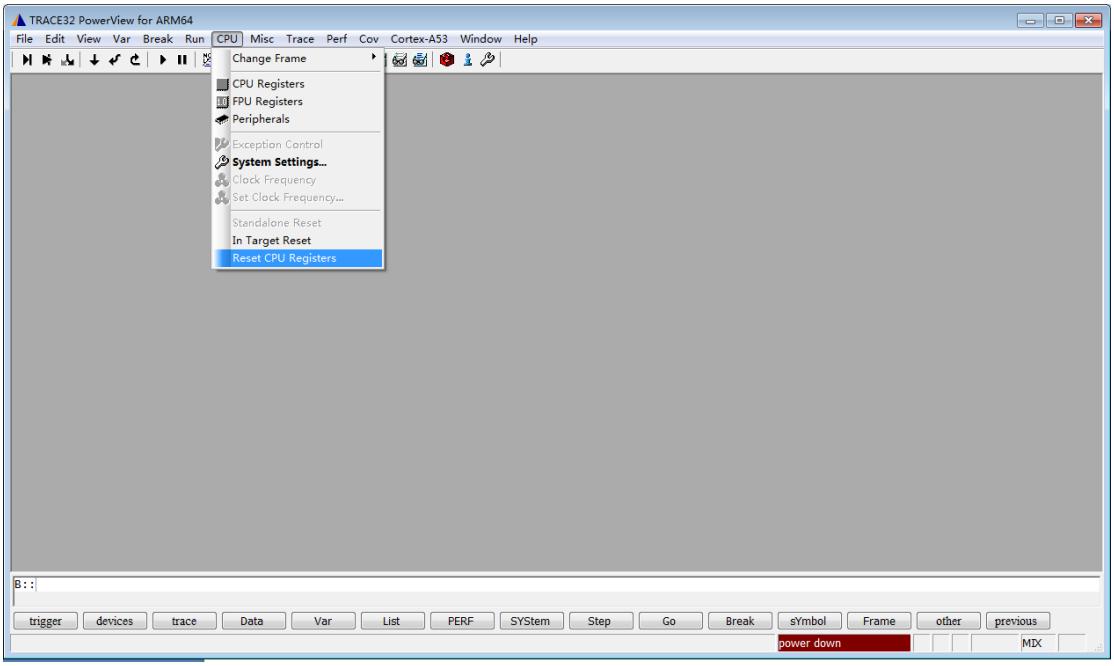
1.4 Break 功能



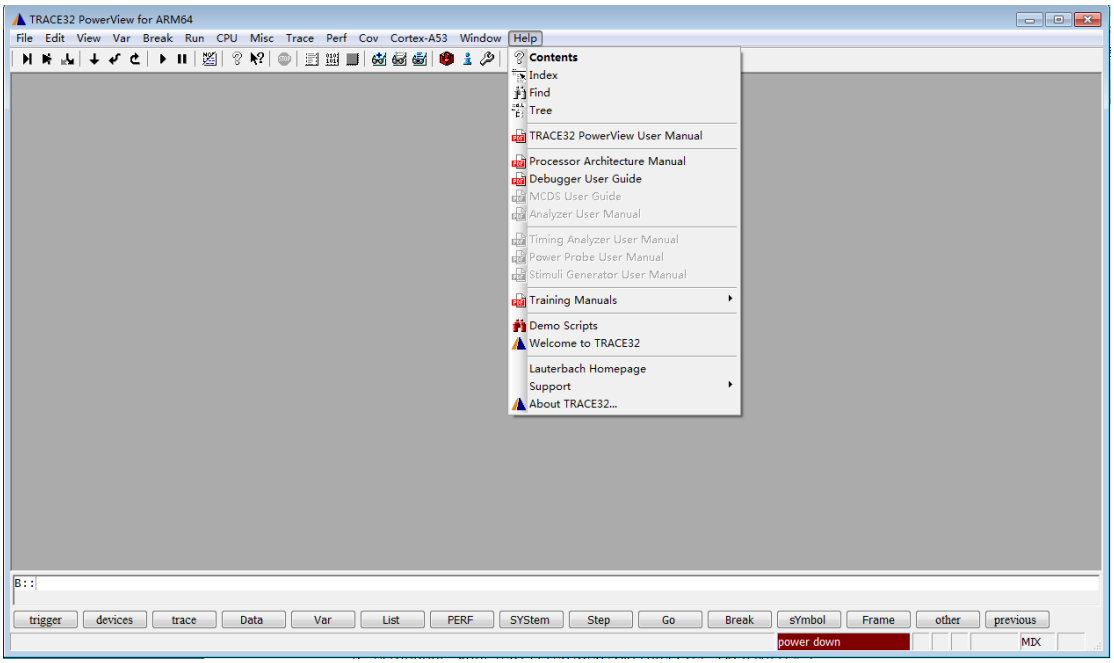
1.5 Run 功能



1.6 Cpu 功能



1.7 help 功能



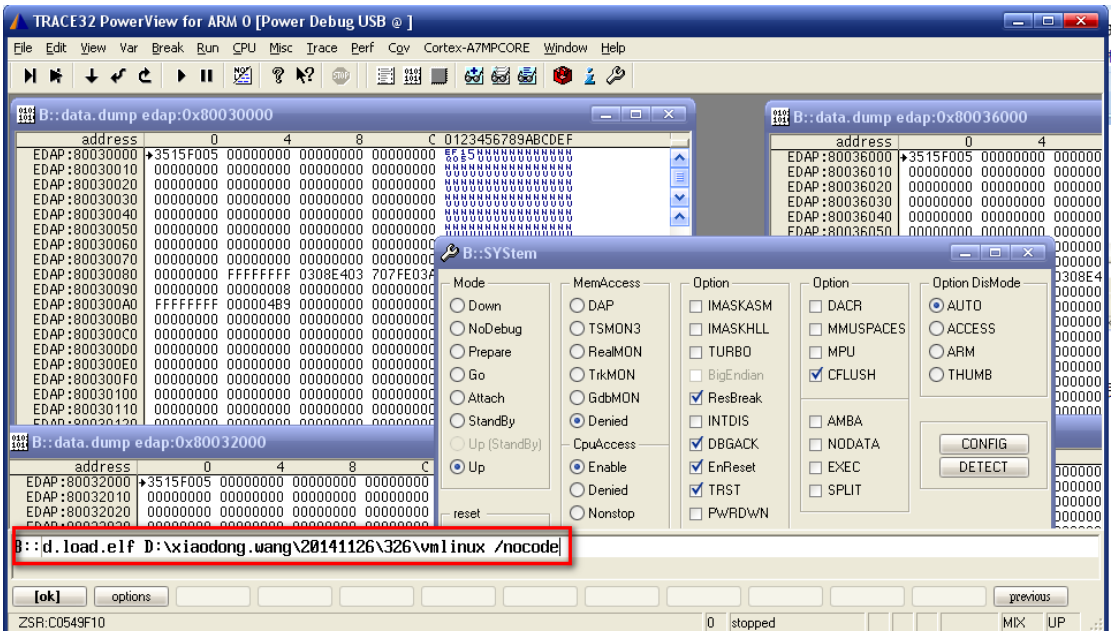
2 Trace32 命令

2.1 Data 命令

系统停止之后，查看函数调用栈，在查看函数调用栈之前，需要向trace32中导入符号表（vmlinux），导入符号表的命令如下：

```
d.load.elf D:\xiaodong.wang\20141126\326\vmlinux /nocode
```

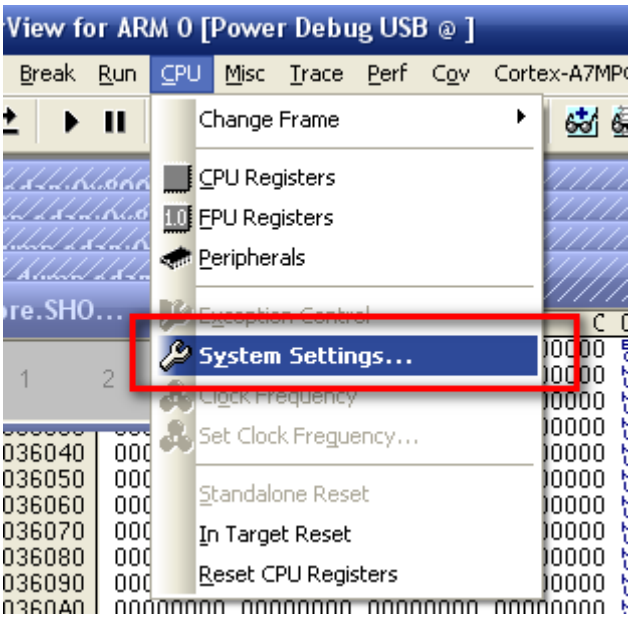
D:\xiaodong.wang\20141126\326\vmlinux为符号表的路径。



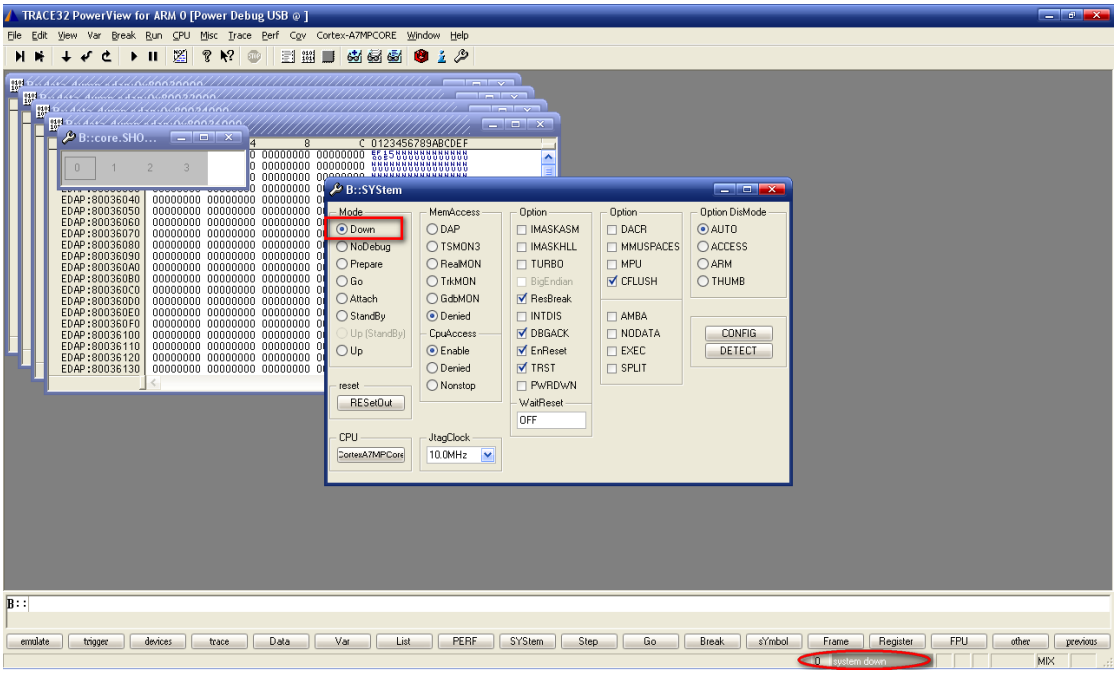
2.2 Var 命令

2.3 System 命令

从主菜单区点击“File->System Settings…”，如图所示。

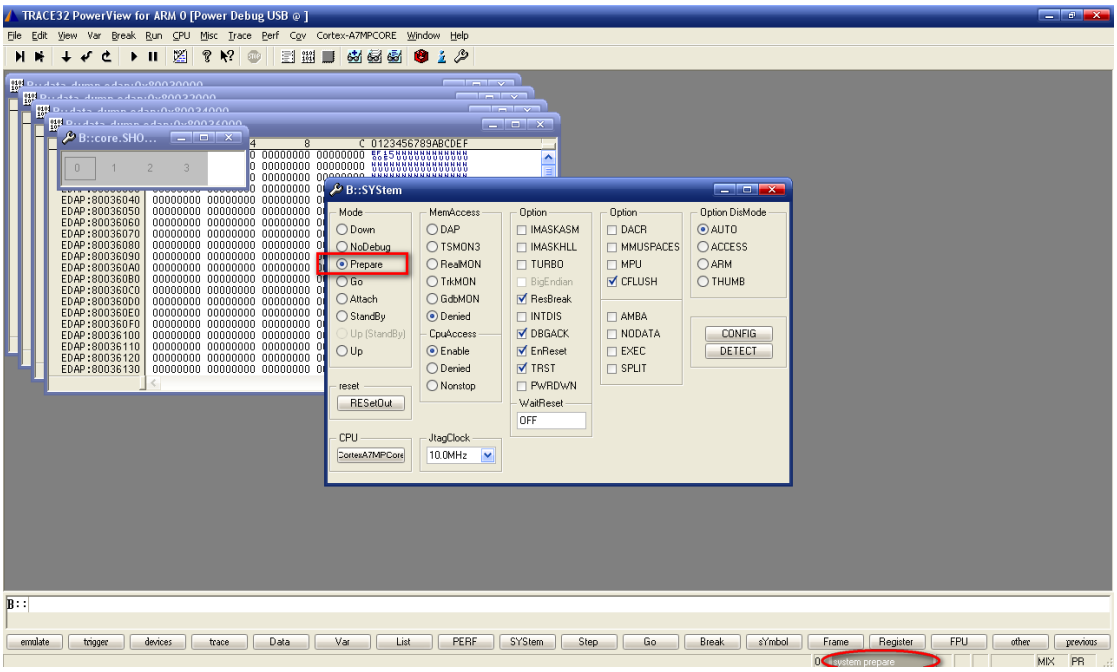


打开脚本文件选择对话框，如下图所示。

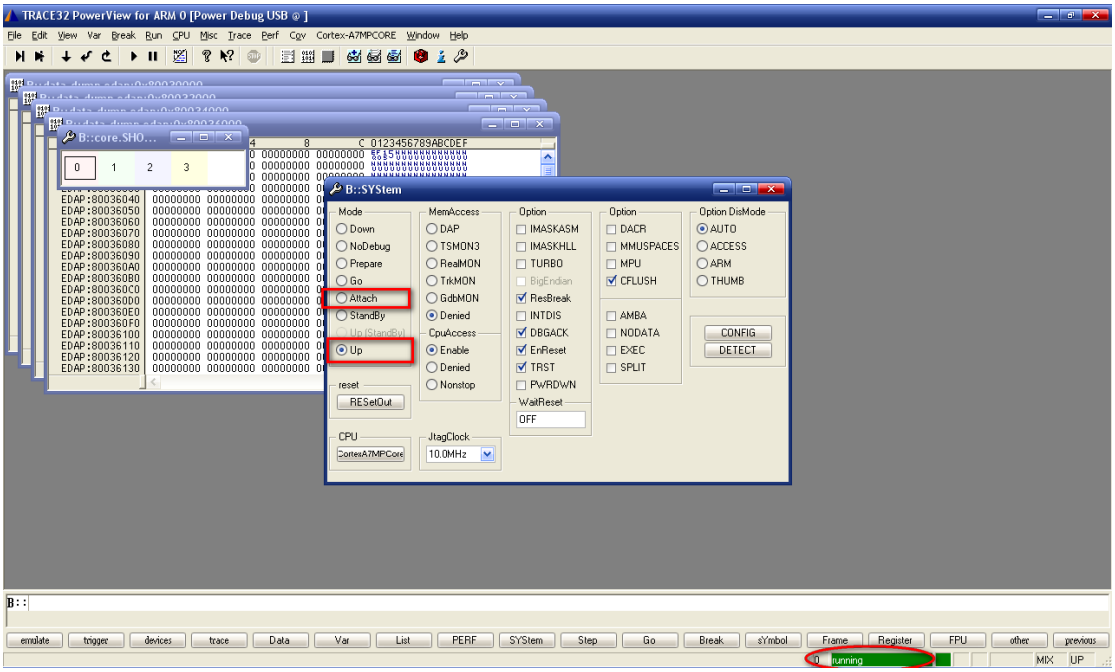


刚启动时，system处于down状态，状态显示区域，显示为“system down”。

然后需要点击“Prepare”进入prepare状态，状态显示区域，显示为“system Prepare”



然后点击“Attach”单选钮，单选框的绿点会跳到“Up”前的小圆框上，在“Up”前面的小园框中有一个绿色园点，表明JTAG通讯已经连接成功，如果选择“Attach”按钮并且目标处理器正在运行的话，在界面的状态显示区会有一个绿色的“Running”条显示，如下图所示。



JTAG通讯已经连接成功后，需要暂停系统。点击下图快捷键区域红框的按钮。

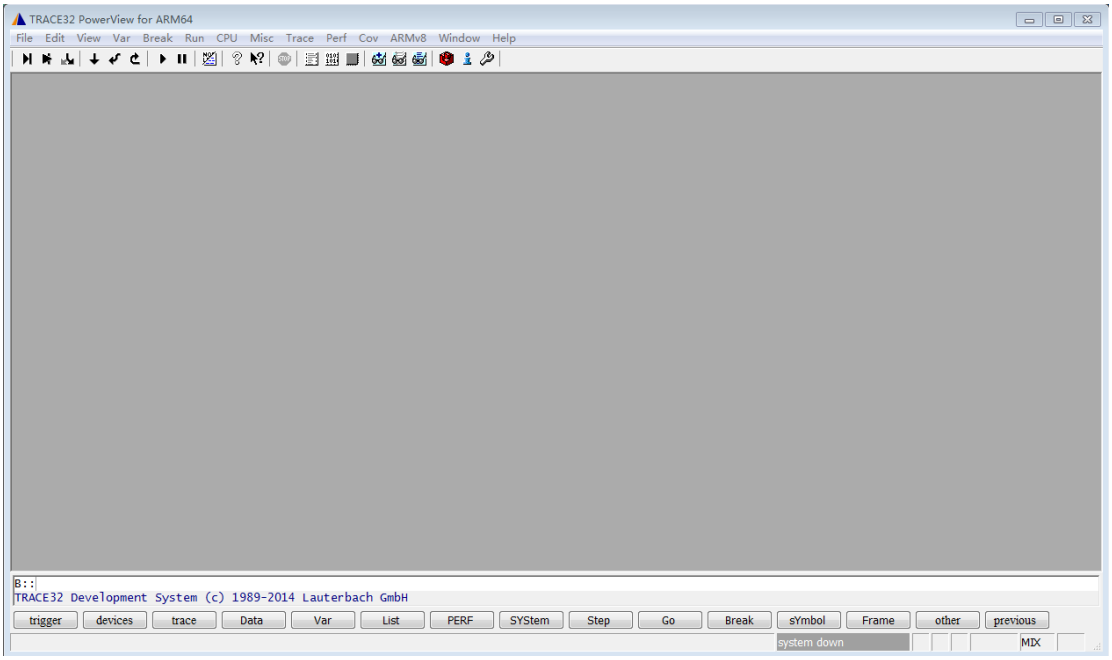
2.4 Break 命令

2.5 Symbol 命令

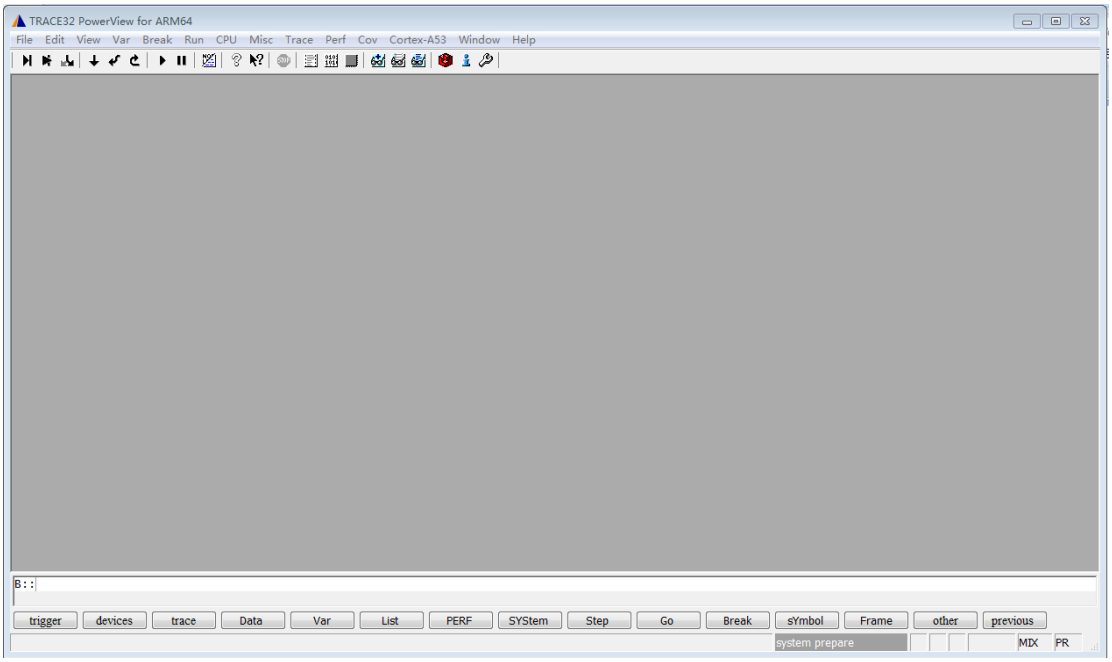
2.6 Frame 命令

3 Trace32 状态类型

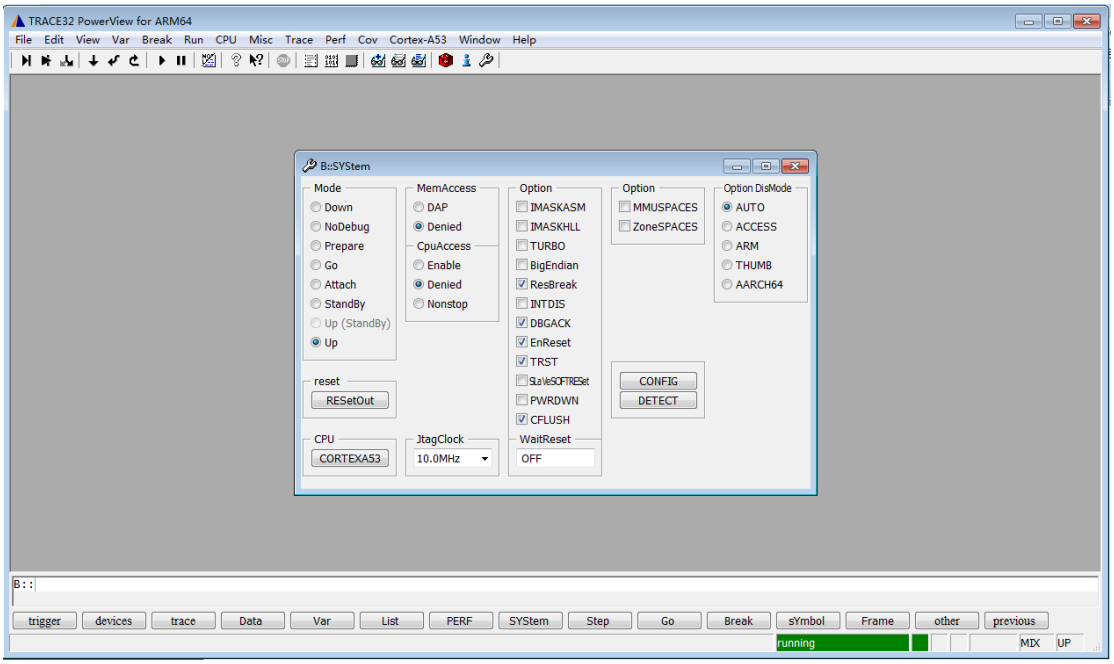
3.1 System Down

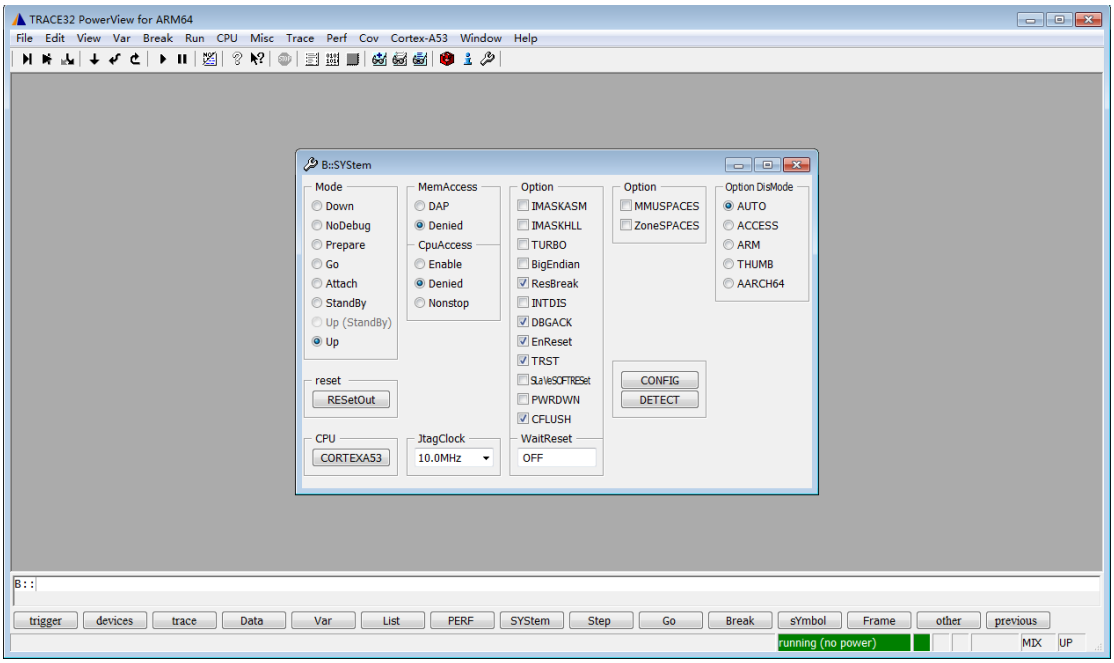


3.2 System Prepare

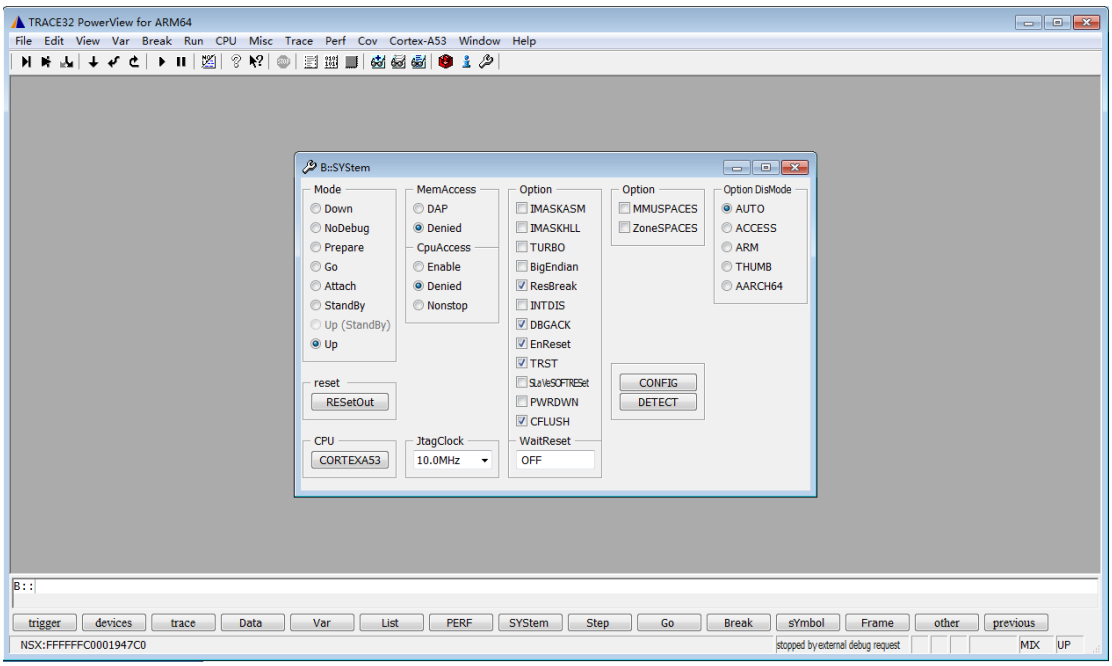


3.3 System Running

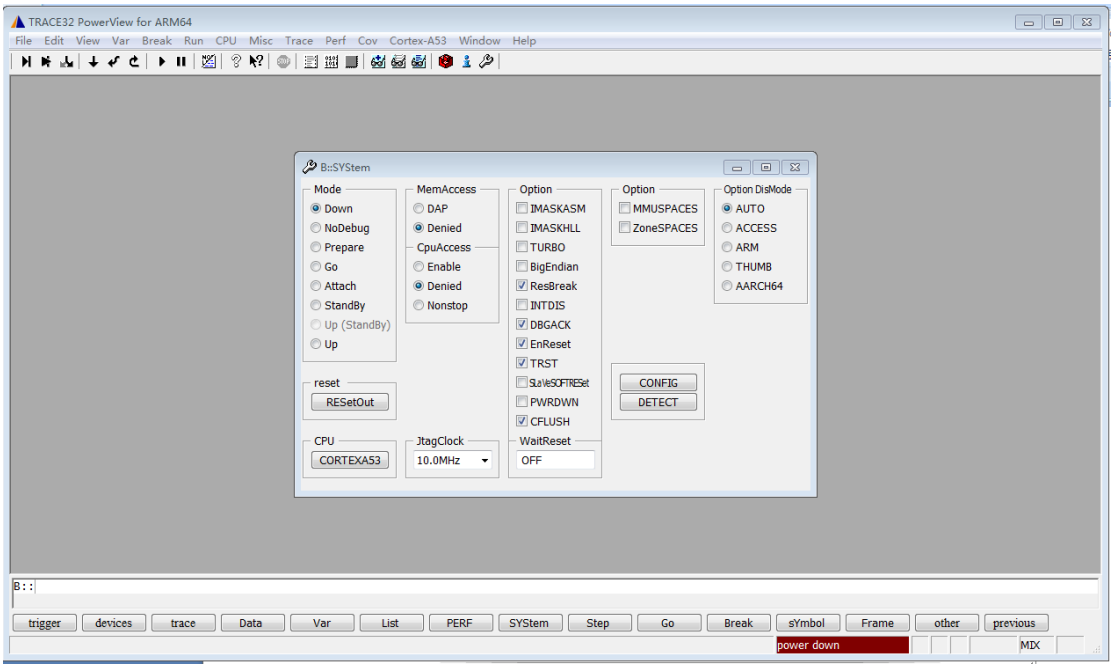




3.4 System Stop



3.5 System PowerDown



4 Trace32 脚本

Trace32 提供了强大的脚本控制功能，可以做到所有的 GUI 的工作,还包括 GUI 无法完成的工作。

正因为其脚本提供的功能过于强大，导致很多功能在 Trace32 软件上你找不到，这部分得用

脚本来驱动使用。更多的 trace32 使用请参考 http://www.jtag1000.com/
Trace32 使用的脚本类似于 pascal 语法，但又有些 C 语言的感觉，采用扩展名为.cmm 的文件，其实就是文本文件，
也可以用记事本打开进行编辑。
好了，下面进入正题，Trace32 的脚本基本语法：
一、先来个输出 Hello World!实例
1、新建一个文本文件，改扩展名为.cmm 文件如：test.cmm
2、打开 Trace32 软件，选择第二个，Edit file，并打开 test.cmm 文件
3、在文件中输入 print "Hello World!",点击上面的 Save 保存按钮，或点击 Save&Run 也行，即运行了第一个 Trace32 程序。
4、输出的内容在左下角，可以看到 Hello World!字样。

4.1 变量申明

二、脚本变量申明

1、本地变量申明：

```
LOCAL &a &b &c//关健字为 LOCAL,变量以“&”开头
ENTRY &a &b
&c=&a*&b
RETURN &c
```

2、全局变量申明：

```
GLOBAL &State &Level//关健字为 GLOBAL,变量以“&”开头
```

4.2 控制语句

三、Trace32 使用的基本控制语句

1、条件控制最基本的 IF-ELSE(注意 Trace32 关健字不区分大小写，也可以是 if else)
如下：

```
IF "a"=="a"
(
 PRINT "true"
)
ELSE IF "a"=="b"
(
 PRINT "false"
)
ELSE
```

```
(
 PRINT "这里不会运行(test)"
)
```

解释一下，Trace32 里面没有 then 关健字，多行语句请使用括号"()"括起来。

2、循环语句 while 和 RePeaT

```
&true=0!=1
&count=1
WHILE &true
(
 DO mem_test
 PRINT "MEMTEST " &count
 &count=&count+1
)
ENDDO

------------
RePeaT [<count>]
 <block>
[WHILE [<condition>]]
or
RePeaT <count> <command>
```

3、GOTO 跳转语句

```
GOTO endloop//关健字为 GOTO，endloop 为标号，如：
 endloop:
  print "这里为 GOTO 执行地"
GOTO 102.//102.为行号
```

4.3 执行命令

四、Trace32 使用的其它执行命令

1、执行其它脚本文件

```
DO <filename> [<parlist>]
比如有二个文件：
a.cmm
b.cmm
a.cmm 内容为：do b
b.cmm 内容为: print "a call b"
运行 a.cmm 文件
```

4.4 调用命令

调用子函数

```
 GOSUB subr1 0x100 10. "abc"//调用子函数，关健字 GOSUB, subr1 为子函数标号，后面为调用的参数 0x100 10. "abc"
 subr1://这里是子函数，以变量名加冒号，标号形式。
  ENTRY &address &len &string
  Data.Set &address++(&len-1) &string
  RETURN
```

4.5 字符输入

3、ENTER 和 ENTRY 语句

```
 enter &x
 print "x=&x"

 ENTRY &address
 GOSUB func1 &address 1.
 ENTRY &result
 PRINT “Result=” &result
 ENDDO
 func1:
  LOCAL &addr &size
  ENTRY &addr &size
  Data.Set &addr++&size 0x0
  &retval=Data.Byte(&addr)
  RETURN &retval
```

INKEY 字符输入

```
 INKEY
 INKEY &key
 IF &key==0x0d
  print "正确的输入"
 else
  print "错误的字符"
```

5 Trace32 步骤

5.1 Trace32 检查手机状态

用 Trace32 检测手机是否掉电如果连接 Trace32 后，显示“power down”,则说明手机已经在长时间的测试过程中掉电关机了。用 USB 线连接手机，可能出现电池充电的画面。

5.2 ADB 检查手机系统状态

先使用 ADB 通过电脑连接手机

sudo adb devices && sudo adb root && sudo adb shell

（1）如果 ADB 可以连接到手机，说明 kernel 还处于运行，并可以响应 ADB 的连接请求；

（2）关注一下 input 事件是否可以上报；

busybox hexdump /dev/input/event0

（3）如果可以看到上报的数据，则看一下 inputservice 服务是否正常。

（4）也可以看一下 kernel 的日志，或将 slog 导出。

cat /proc/sys/kernel/printk

echo 8 > /proc/sys/kernel/printk

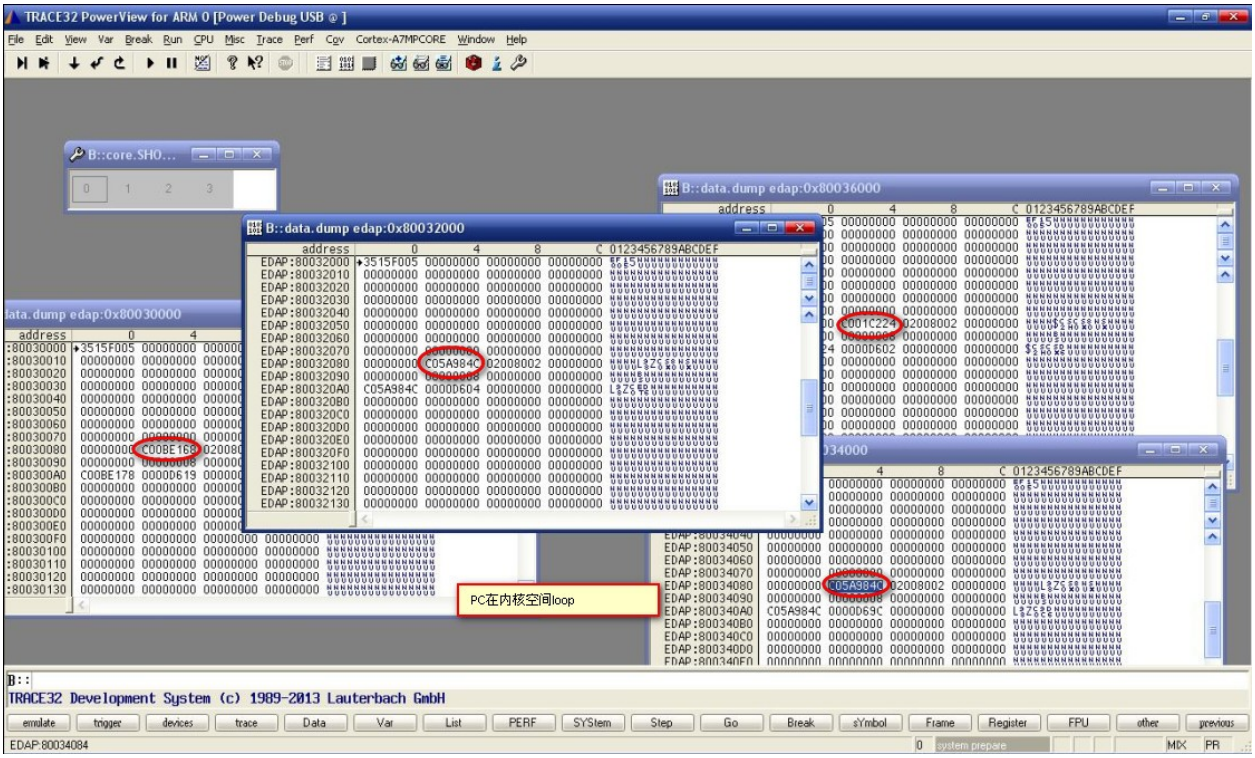
cat /proc/kmsg

adb pull /storage/sdcard0/slog .

如果 ADB 无法连接手机，使用 Trace32 连接手机

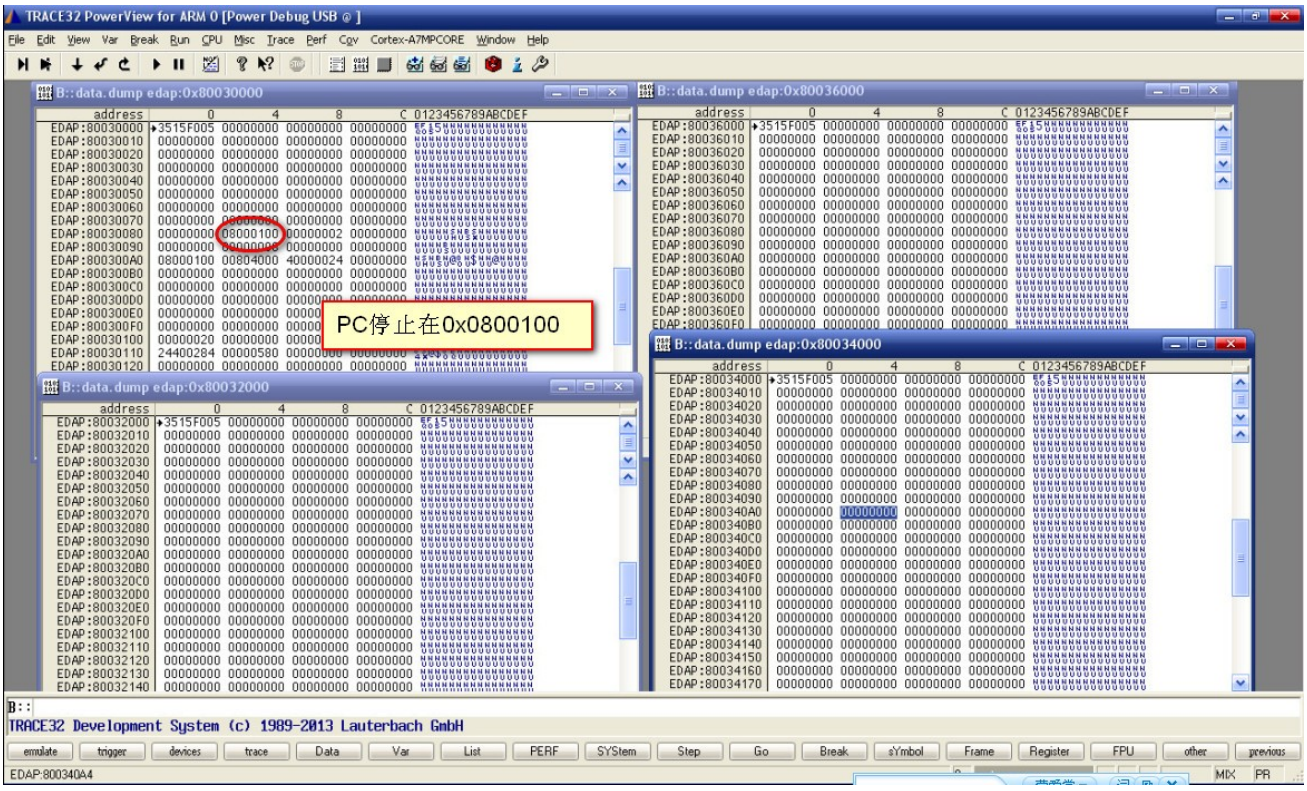
5.3 Trace32 连接手机

设置 Trace32 为 prepare，看 PC 的情况
一般的黑屏和定屏问题，PC 处于几处地址的 loop 状态或 PC 处于停止状态。
PC 处于 loop 状态
4 个窗口为 4 个 core 的 PC 地址处情况



PC 处于某一地址不动

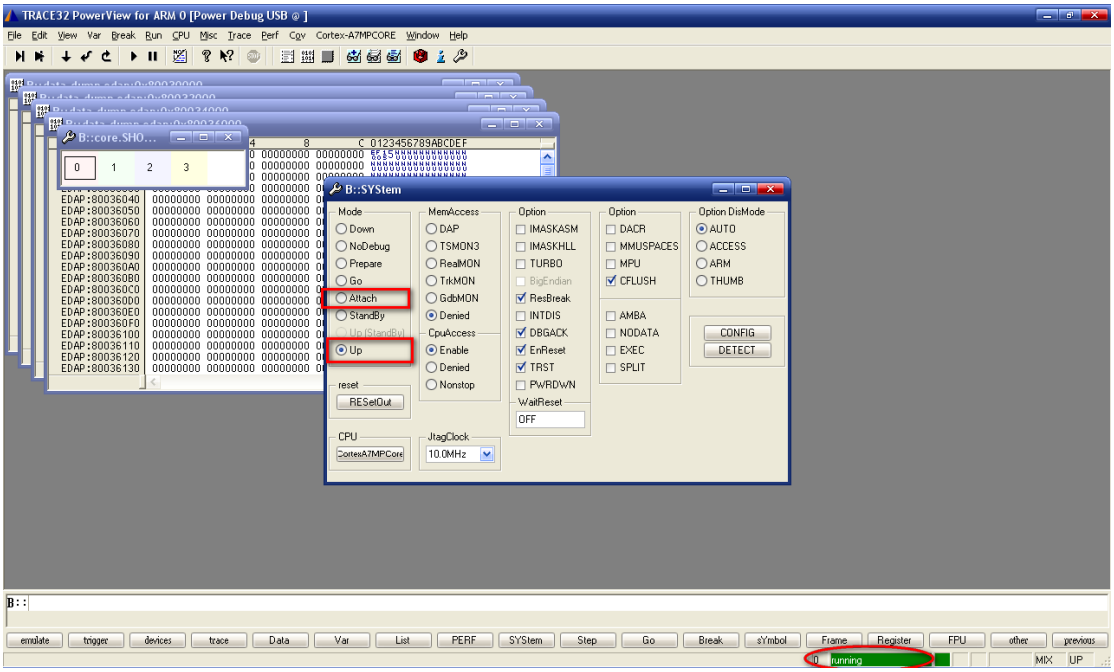
5.4 Trace32 脚本运行



PC 停止的时候，可以看一下 AON 寄存器：0 x402b0000。如果是 PC 停止于某一地址（如在 IRam 中），可以让 DDR 同事来看一下。

这时，我们对问题有了一个大致的定位， 下一步用 Trace 32 暂停 CPU 的执行。
使用 Trace 32 暂停 CPU
先让 Trace32 的 CPU 设置为，attch，然后暂停 CPU。

5.5 Trace32 查看 PC 信息



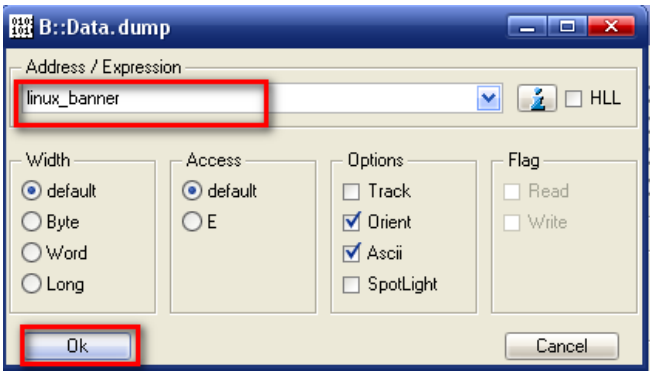
5.6 Trace32 查看 Core 栈信息

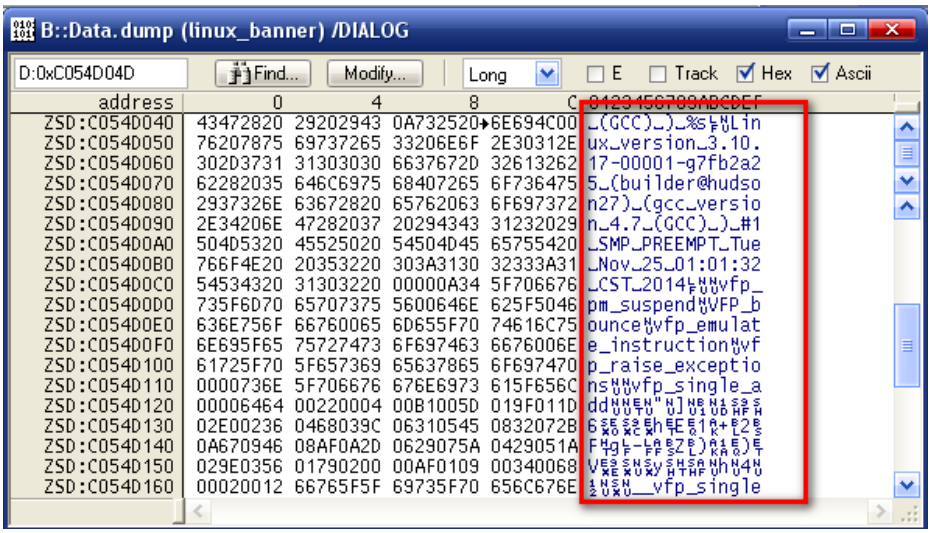
导入符号表

```
d.load.elf Y:\bugs\315438\vmlinux. /nocode
```

查看符号表是否正确

5.7 Trace32 查看变量信息





5.8 Trace32 保存内存信息

通过 Trace 32 看 4 个 CPU core 的 stack 信息
如：

```
-000|mspin_unlock(lock = 0xC084022C, node = 0xDE92DE6C)

-001|__mutex_lock_slowpath(lock_count = 0xC0840218)

-002|mutex_lock(lock = 0xC0840218)

-003|clk_prepare_lock()

-004|clk_prepare(clk = 0xDF414F80)

-005|sprdfb_dispc_clk_enable.constprop.9()

-006|sprdfb_dispc_resume(dev = 0xDF760270)

-007|sprdfb_late_resume(es = 0xDF7602FC)

-008|late_resume(?)

-009|process_one_work(worker = 0xDED27080, work = 0xC082394C)

-010|worker_thread(__worker = 0xDED27080)

-011|kthread(_create = 0xDF46FEB0)

-012|ret_from_fork(asm)

 -->|exception

 ---|end of frame
```

这个问题是 prepare_lock 拿不到，导致 CPU 一直在 loop。
使用 Trace32 导出手机的 memory

```
d.save.binary D:\xiaodongwang\20140923\781.dump eahb:0x80000000++0x20000000
```

再增加一下头信息，就可以通过 crash 来分析现场的 memory。

如看一下 log_buffer 或 CPU 停止和 loop 的指令信息。

```
crash> dis 0xC084022C

0xc084022c <prepare_lock+20>:   andeq   r0, r0, r0
```

如果总线挂住，不能通过 Trace 32 导出 memory，可以通过常按手机 power 键 7s 来 dump memory。

通过 crash 工具可以看一下 curent 进程，也可以看一下中断或 sprd_debug_last_regs_access 等信息。

6 Trace32 案例

Edap地址80030000是cpu0的状态显示窗口，3515F005为特征码，用于调试，后面空余一些保留位用于扩展，c001BD84位置保存cpu的指针值。此时我们可以观察4个cpu的指针是如何变化：

- pc某个地址范围内循环，说明kernel可能处于死循环状态。
- PC定在某个地址处，分为两种情况：
  - 部分现场是停在wfi或wfe指令上，推测可能是等待不到中断或事件（这类现场分析的很少，还不能得出明确的结论）。
  - PC跑飞，系统找不到正确的指令。
- Cpu的某个pc指针为0，说明该cpu被拔掉。

这时候最好截图记录 PC 指针的值，以备后续查看。

6.1 【定屏】Spin_lock 锁问题

Bugid:434774

http://bugzilla.spreadtrum.com/bugzilla/show_bug.cgi?id=434774

6.1.1 现场 PC 信息

Monkey 测试 407#机出现定屏，连接 Trace32 获取各 Core 的信息

Trace32 获取的 Core0 的栈信息示意图

>>现场的一些重要信息

raw_spin_lock 锁结构体信息

-000|raw_spin_lock_irqsave(

```
|    lock = 0xC0B410E8 -> (
|      raw_lock = (
|        slock = 728116068,
|        tickets = (
|          owner = 11108,
|          next = 11110))))
|  flags = 2148401171
|  tmp = 0
```

CPU0 调用栈信息：

```
-000|raw_spin_lock_irqsave(lock = 0xC0B410E8)
-001|dcam_reg_isr(id = DCAM_PATH0_DONE, user_func = 0x0, user_data = 0x2B65)
-002|sprd_img_reg_isr(param = 0xF11DF000)
-003|sprd_img_k_ioctl(?, ?, ?)
-004|vfs_ioctl(?, ?, ?)
-005|do_vfs_ioctl(filp = 0xE1DEACC0, fd = 3277946288, ?, ?)
-006|sys_ioctl(fd = 33, cmd = 1074027022, arg = -1578279836)
-007|ret_fast_syscall(asm)
 -->|exception
-008|ZUR:0xB6EEB374(asm)
-009|ZUT:0xB6F02704(asm)
-010|ZUT:0xB5484B1A(asm)
-011|ZUT:0xB547B4A0(asm)
-012|ZUT:0xB548C552(asm)
-013|ZUT:0xB5491186(asm)
-014|ZUT:0xB5482942(asm)
-015|ZUT:0xB6EC6FEC(asm)
 ---|end of frame
```

6.1.2 现场 PC 分析

Core1,Core2,Core3 都处于 idle 状态

Core0 处于 loop 循环状态，怀疑内核进入死循环。从栈信息看，应该是 DCAM 发生中断，加锁时发生的。

next 值比 owner 值大 2，需要确认是否有位翻转等异常信息。

CPU0 调用栈信息：

```
-000|raw_spin_lock_irqsave(lock = 0xC0B410E8)
-001|dcam_reg_isr(id = DCAM_PATH0_DONE, user_func = 0x0, user_data = 0x2B65)
-002|sprd_img_reg_isr(param = 0xF11DF000)
```

对照代码看，栈信息的参数有问题。user_func 应该为 sprd_img_tx_done， user_data 应该和 param 一样 0xF11DF000 才对

```
LOCAL int sprd_img_reg_isr(struct dcam_dev* param)
{
    dcam_reg_isr(DCAM_PATH0_DONE,    sprd_img_tx_done,        param);
    dcam_reg_isr(DCAM_PATH1_DONE,    sprd_img_tx_done,        param);
    dcam_reg_isr(DCAM_PATH0_OV,      sprd_img_tx_error,       param);
    dcam_reg_isr(DCAM_PATH1_OV,      sprd_img_tx_error,       param);

int32_t  dcam_reg_isr(enum  dcam_irq_id  id,  dcam_isr_func  user_func,  void*
user_data)
```

解决方案：
拍照后启动预览挂在 spinlock,优化 dcam_update_path 里 spinlock 的使用,和快停功能。
优化 dcam_update_path 里 spinlock 的使用

6.2 【黑屏】DDR 循环等待问题

Bugid
http://bugzilla.spreadtrum.com/bugzilla/show_bug.cgi?id=440190

6.2.1 现场 PC 信息
Shark1 待机出现黑屏出现黑屏，连接 USB 依然无反应，无法连接 ADB，长按电源 7s 无效，无法 dump 内存
连接 Trace32 在 prepare 下 Core0 的信息
C0： PC 处于非内核地址空间，在【50006b58 和 50006b64】之间跳
C1,C2,C3：未运行。

```
ENAHB:50006B04|E3A02203          mov     r2,#0x30000000    ; r2,#805306368
ENAHB:50006B08|E59F40B0          ldr     r4,0x50006BC0
ENAHB:50006B0C|E3A03000          mov     r3,#0x0           ; r3,#0
ENAHB:50006B10|E3A00032          mov     r0,#0x32          ; r0,#50
ENAHB:50006B14|E58D3004          str     r3,[r13,#0x4]
ENAHB:50006B18|E58231B0          str     r3,[r2,#0x1B0]
ENAHB:50006B58|E3003F01          mov     r3,#0xF01
ENAHB:50006B5C|E59F505C          ldr     r5,0x50006BC0
ENAHB:50006B60|E0013003          and     r3,r1,r3
ENAHB:50006B64|E1530002          cmp     r3,r2
ENAHB:50006B68|1AFFFFF9          bne     0x50006B54
```

6.2.2 现场 PC 分析

从汇编指令可以看到，

```
mov     r2,#0x30000000
str     r3,[r2,#0x1B0]
```

指令在操作 DDR 寄存器，后续再获取 DDR 状态时，条件未满足，因此 C0 一直处于 poll 循环等待状态。

和地址 50006***地址相关的代码有 DFS,SPL，确认 5.1 5M 上没有 DFS 操作 DDR，因此操作 DDR 的代码和 SPL 有关。

操作 DDR 过程中，陷于死循环，因此长按 power 7s 等 dump 功能都无效

解决方案：
是 ddr training 的时候，等 ddr training 完成标志，等不到，导致程序进行不下去，后续把 training not done 最新的处理方式给更新上，以解决该问题。

6.3 【黑屏】Mutex_lock 问题

Bugid
http://bugzilla.spreadtrum.com/bugzilla/show_bug.cgi?id=451392

6.3.1 现场信息

C0 挂在 mutex_lock
C1,C2,C3 未运行

```
000|owner_running(inline)
-000|mutex_spin_on_owner(lock = 0xEE6C0D78, owner = 0xD066C480)
-001|__mutex_lock_common(inline)
-001|__mutex_lock_interruptible_slowpath(lock_count = 0xEE6C0D78)
-002|mutex_lock_interruptible(lock = 0xEE6C0D78)
-003|rtc_read_time(rtc = 0xEE6C0C00, tm = 0xD17A3ADC)
-004|alarmtimer_fired(?)
-005|static_key_false(inline)
-005|trace_hrtimer_expire_exit(inline)
-005|__run_hrtimer(timer = 0xC0C3A148, now = 0xD17A3B98)
-006|hrtimer_interrupt(?)
-007|__gptimer_interrupt(?, dev_id = 0xD066C480)
-008|static_key_false(inline)
-008|trace_irq_handler_exit(inline)
-008|handle_irq_event_percpu(desc = 0xC09CAE80, action = 0xD066C480)
-009|handle_irq_event(desc = 0xC09CAE80)
-010|handle_fasteoi_irq(?, desc = 0xC09CAE80)
-011|generic_handle_irq(irq = 60)
-012|handle_IRQ(irq = 60, ?)
-013|gic_handle_irq(regs = 0xEE6C0D78)
-014|__irq_svc(asm)
```

```
 -->|exception
-015|__raw_spin_unlock_irqrestore(inline)
-015|raw_spin_unlock_irqrestore(?, flags = 537853971)
-016|spin_unlock_irqrestore(inline)
```

6.3.2 现场分析

Core0 从栈信息参数传递看，调用没有问题，确实是从__run_hrtimer 中调用了定时器 fn 函数 alarmtimer_fired

rtc_read_time 中调用了__mutex_lock，和 alarmtimer 有关

6.4 【黑屏】Core Boot 问题

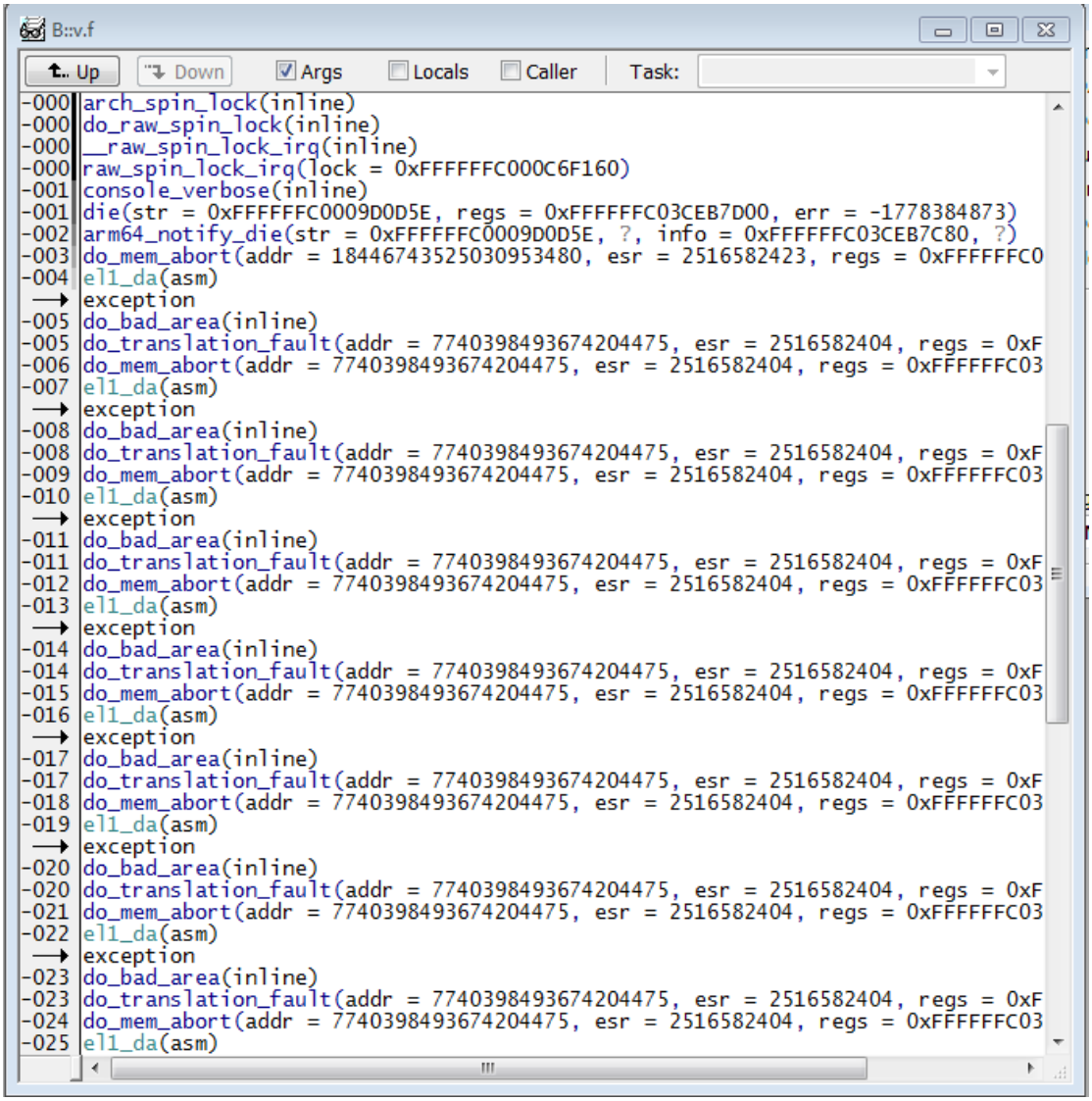
6.4.1 现场信息

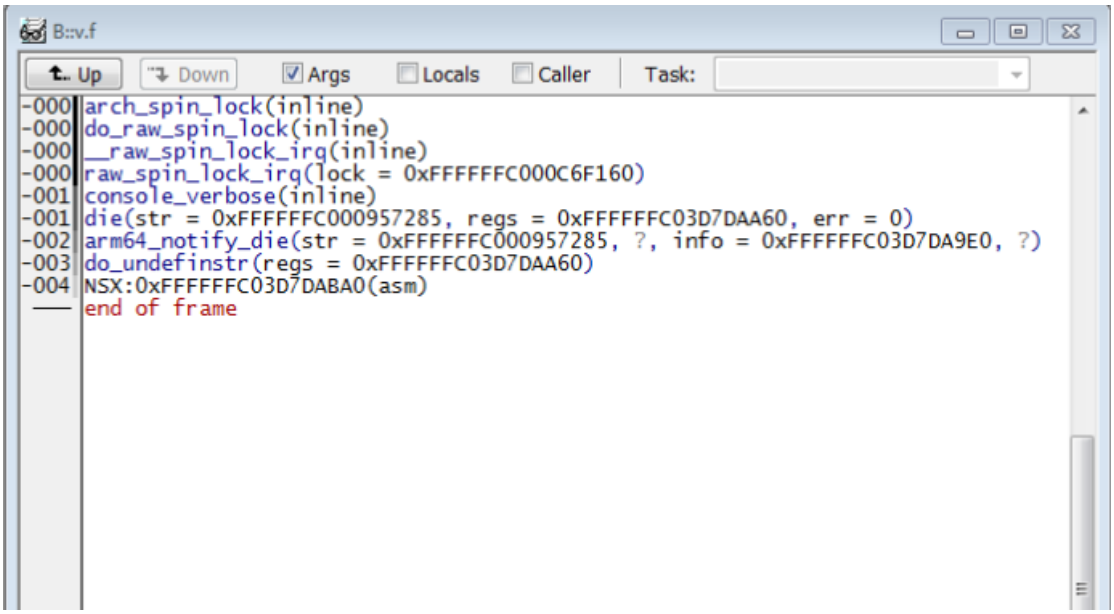
```
================Print the A53-Core0 PC value loop 0x20 times

A53-Core0 PC value 0x0 times : 0xFFFFFFFF
A53-Core0 PC value 0x1 times : 0xFFFFFFFF
A53-Core0 PC value 0x2 times : 0xFFFFFFFF
A53-Core0 PC value 0x3 times : 0xFFFFFFFF
A53-Core0 PC value 0x4 times : 0xFFFFFFFF
A53-Core0 PC value 0x5 times : 0xFFFFFFFF
A53-Core0 PC value 0x6 times : 0xFFFFFFFF
A53-Core0 PC value 0x7 times : 0xFFFFFFFF
A53-Core0 PC value 0x8 times : 0xFFFFFFFF
A53-Core0 PC value 0x9 times : 0xFFFFFFFF
A53-Core0 PC value 0x0A times : 0xFFFFFFFF
================Print the A53-Core1 PC value loop 0x20 times

A53-Core1 PC value 0x0 times : 0xFFFFFFFF
A53-Core1 PC value 0x1 times : 0xFFFFFFFF
A53-Core1 PC value 0x2 times : 0xFFFFFFFF
A53-Core1 PC value 0x3 times : 0xFFFFFFFF
A53-Core1 PC value 0x4 times : 0xFFFFFFFF
A53-Core1 PC value 0x5 times : 0xFFFFFFFF
A53-Core1 PC value 0x6 times : 0xFFFFFFFF
A53-Core1 PC value 0x7 times : 0xFFFFFFFF
A53-Core1 PC value 0x8 times : 0xFFFFFFFF
A53-Core1 PC value 0x9 times : 0xFFFFFFFF
A53-Core1 PC value 0x0A times : 0xFFFFFFFF
```

6.4.2 现场分析

6.5 【黑屏】Core Plug 问题

6.5.1 现场信息





6.5.2 现场分析

6.6 【黑屏】SML Panic 问题

Bugid

6.6.1 现场信息

422#,426# 黑屏 C0:PC 停在 0x000000000x5080d368，进入了 spl 的 crash_panic
C1,C2,C3，C4,C5,C6,C7:未运行
==========Print the A53-Core0 PC value loop 0x20 times

A53-Core0 PC value 0x0 times : 0x5080D368
A53-Core0 PC value 0x1 times : 0x5080D368
A53-Core0 PC value 0x2 times : 0x5080D368
A53-Core0 PC value 0x3 times : 0x5080D368
A53-Core0 PC value 0x4 times : 0x5080D368
A53-Core0 PC value 0x5 times : 0x5080D368
A53-Core0 PC value 0x6 times : 0x5080D368
A53-Core0 PC value 0x7 times : 0x5080D368
A53-Core0 PC value 0x8 times : 0x5080D368
A53-Core0 PC value 0x9 times : 0x5080D368

6.6.2 现场分析

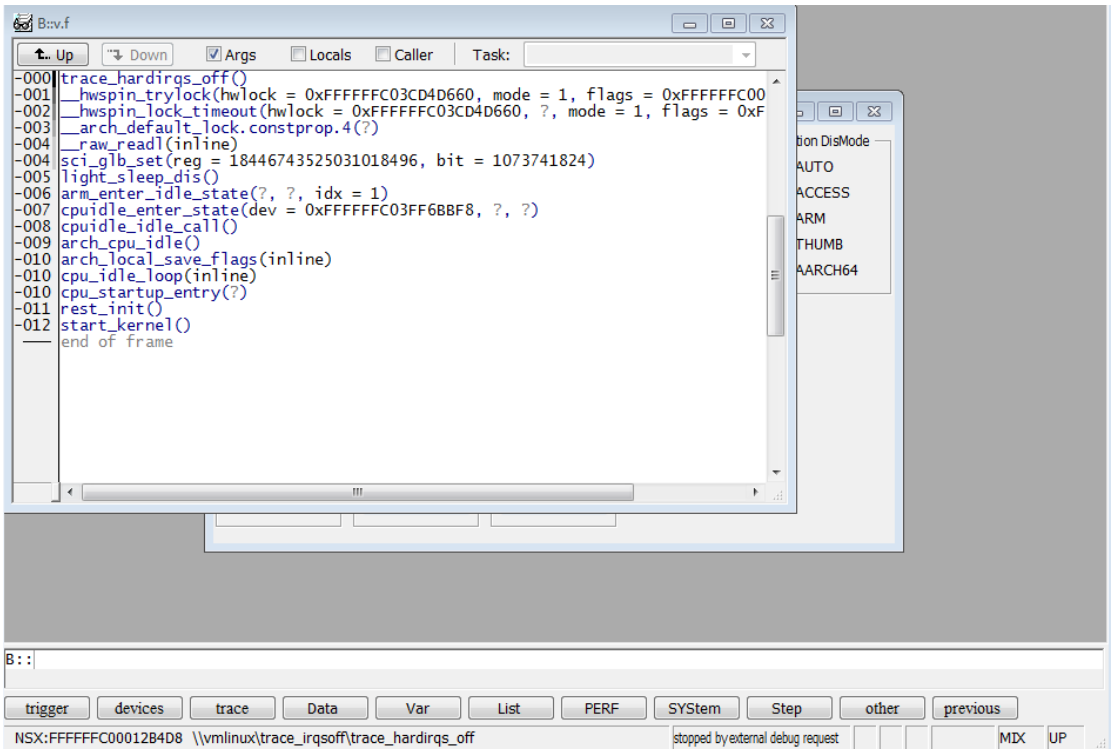
sml.bin，sml.elf，sml.sym

```
0000000050808200 <bl31_entrypoint>:
    50808200:   d53e1000    mrs x0, sctlr_el3
    50808204:   9266f800    and x0, x0, #0xfffffffffdffffff
    50808208:   d51e1000    msr sctlr_el3, x0
    5080820c:   d5033fdf    isb
    50808210:   94001457    bl  5080d36c <reset_handler>
    50808214:   d2820141    mov x1, #0x100a                 // #4106
    50808218:   d53e1000    mrs x0, sctlr_el3
    5080821c:   aa010000    orr x0, x0, x1
    50808220:   d51e1000    msr sctlr_el3, x0

000000005080d368 <crash_panic>:
    5080d368:   14000000    b   5080d368 <crash_panic>
```

6.7 【黑屏】HW_SPINLOCK 问题

Bugid

6.7.1 现场信息




```
================Print the A53-Core1 PC value loop 0x20 times

A53-Core1 PC value 0x0 times : 0xFFFFFFC0002FB6FC
A53-Core1 PC value 0x1 times : 0xFFFFFFC0002FB6D8
A53-Core1 PC value 0x2 times : 0xFFFFFFC0001317FC
A53-Core1 PC value 0x3 times : 0xFFFFFFC0002FB6E0
```

```
A53-Core1 PC value 0x4 times : 0xFFFFFFC000131924
A53-Core1 PC value 0x5 times : 0xFFFFFFC000566DF8
A53-Core1 PC value 0x6 times : 0xFFFFFFC00013191C
A53-Core1 PC value 0x7 times : 0xFFFFFFC00013191C
A53-Core1 PC value 0x8 times : 0xFFFFFFC000759FC4
A53-Core1 PC value 0x9 times : 0xFFFFFFC0000CC590
A53-Core1 PC value 0x0A times : 0xFFFFFFC0000CC56C
A53-Core1 PC value 0x0B times : 0xFFFFFFC0000CC5C8
A53-Core1 PC value 0x0C times : 0xFFFFFFC000566CBC
A53-Core1 PC value 0x0D times : 0xFFFFFFC0000CC590
A53-Core1 PC value 0x0E times : 0xFFFFFFC000566C0C
A53-Core1 PC value 0x0F times : 0xFFFFFFC000566CB8
```

6.7.2 现场分析

查看软件的 hwspinlock 全局数组 local_hwlocks_status 查看是否为 AP 拿到锁没有释放。

```
crash64> local_hwlocks_status
local_hwlocks_status = $1 =
"\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\0
00\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\
000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\000\000\000\000"
crash64>
```

查看 hwspinlock 硬件状态寄存器 0x40060008(注意一定要将对话框缩小到只能看一个寄存器，否则会误读上锁寄存器)，查看该寄存器是否已经被异常拿锁。

6.8 【黑屏】IRAM 循环问题

Bugid

6.8.1 现场 PC 信息

C0 在 IRAM 里 loop,Loop 地址范围在【1004F4-10051C】，查看汇编代码，可以看到条件未满足，导致无法跳到 0x100520，需要 IRAM 同事帮忙看下。

```
ZSX:00000000001004F4
......
ZSX:000000000010051C
--------------------------------------------------
C0: PC loop
A53-Core0 PC value 0x0 times : 0x100510
A53-Core0 PC value 0x1 times : 0x100510
A53-Core0 PC value 0x2 times : 0x100510
```

A53-Core0 PC value 0x3 times : 0x100510
A53-Core0 PC value 0x4 times : 0x100510
A53-Core0 PC value 0x5 times : 0x10050C
A53-Core0 PC value 0x6 times : 0x100510
A53-Core0 PC value 0x7 times : 0x100510
A53-Core0 PC value 0x8 times : 0x100510
A53-Core0 PC value 0x9 times : 0x100510
其余 core 未运行

6.8.2 现场 PC 分析

以下条件未满足，导致无法跳到 0x100520

```
ZSX:0000000000100500|7101009F          cmp     w4,#0x40          ; w4,#64
ZSX:0000000000100504|540000E0          b.eq    0x100520
```

汇编代码:

```
_____________addr/line|code_____|label____|mnemonic_____________|comment
   ZSX:00000000001004DC|00000000          undef   0x0
   ZSX:00000000001004E0|52800000          mov     w0,#0x0         ; w0,#0
   ZSX:00000000001004E4|58000423          ldr     x3,0x100568
   ZSX:00000000001004E8|B9400060          ldr     w0,[x3]
   ZSX:00000000001004EC|7100041F          cmp     w0,#0x1         ; w0,#1
   ZSX:00000000001004F0|54FFFFA1          b.ne    0x1004E4
>>   ZSX:00000000001004F4|52800002          mov     w2,#0x0         ; w2,#0
   ZSX:00000000001004F8|B9000062          str     w2,[x3]
   ZSX:00000000001004FC|11000484          add     w4,w4,#0x1      ; w4,w4,#1
   ZSX:0000000000100500|7101009F          cmp     w4,#0x40        ; w4,#64
   ZSX:0000000000100504|540000E0          b.eq    0x100520
   ZSX:0000000000100508|D2800801          mov     x1,#0x40        ; x1,#64

ZSX:000000000010050C|D1000421__________sub_____x1,x1,#0x1_______;_x1,x1,#1
   ZSX:0000000000100510|F100003F          cmp     x1,#0x0         ; x1,#0
___ZSX:0000000000100514|54FFFFC1__________b.ne____0x10050C
   ZSX:0000000000100518|B9400060          ldr     w0,[x3]
>>  ZSX:000000000010051C|17FFFFF6          b       0x1004F4
   ZSX:0000000000100520|58000280          ldr     x0,0x100570
   ZSX:0000000000100524|B9400002          ldr     w2,[x0]
   ZSX:0000000000100528|B9400001          ldr     w1,[x0]
   ZSX:000000000010052C|12080C42          and     w2,w2,#0xF000000  ; w2,w2,#
   ZSX:0000000000100530|12080C21          and     w1,w1,#0xF000000  ; w1,w1,#
   ZSX:0000000000100534|7100005F          cmp     w2,#0x0         ; w2,#0
   ZSX:0000000000100538|54FFFF61          b.ne    0x100524
```

6.9 【定屏】Suspend MMU 问题

Bugid

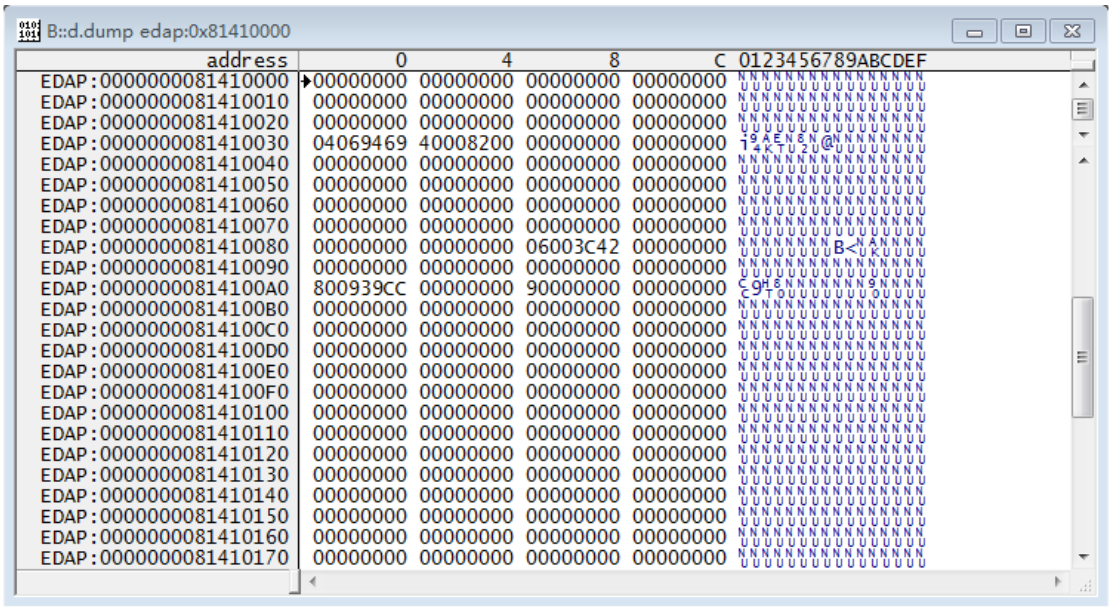
http://bugzilla.spreadtrum.com/bugzilla/show_bug.cgi?id=466860

6.9.1 现场 PC 信息

基于 286#进行降压（小核 0.9/大核 1.0）， 执行相关 Reboot / 大应用切换 / SleepWakeup / Monkey 测试，验证对稳定性的影 响，结果大面积发生定屏或黑屏。

C0：PC 停在 0x800939C8 或 C0：PC 停在 0x800939CC. 偶尔 PC 停在 FFFFFFC000093BB8。
C1,C2,C3 的小 core 的 PC 全部为 0
C4,C5,C6,C7 的大 core 的 PC 停在不同的位置，有的停在内核，有的停在上层，有的停在内核模块。

没有降压的版本，在 Reboot 测试时也发生 一例，从现象上来看，降压后，该问题出现几率非常高
C0 的 PC 停在固定地址处，0x00000000800939cc,如下图所示



6.9.2 现场 PC 分析

0x800939CC 应 该是 DDR 中存放内核代码的空间地址。内核运行后，PC 跑 的地址应该是虚拟后的内核空间地址，如 0xFFFFFFC0000962C4。 或者是 SML 中 的代码，以及一些特殊 PC 值， 如中断向量等， 或 PC 跑飞到异常地 址，怀疑内核当前没有执行 MMU 映 射或者 MMU 异常。内核虚拟地址应该是 DDR 地 址处的一个简单的物理映射。0x800939CC 对 应的虚拟 PC 应该是 FFFFFFC0000939CC。

从下面可以看到系统会通过调用 cpu_resume_mmu， 唤醒 MMU 表，映射内核虚拟空间。
NSX:FFFFFFC0000939C0|58000183 cpu_resume_mmu: ldr x3,0xFFFFFFC0000939F0

```
114|NSX:FFFFFFC0000939C4|D5181000        msr     #0x3,#0x0,c1,c0,#0x0,x0   ; #3,#0,c1,c0,#0,x0
115|NSX:FFFFFFC0000939C8|D5033FDF        isb     sy
116|NSX:FFFFFFC0000939CC|D61F0060_____________________________br_____x3
```

对应内核代码如下

```
ENTRY(cpu_resume_mmu)
        ldr     x3, =cpu_resume_after_mmu
        msr  sctlr_el1, x0 // restore sctlr_el1
        isb
        br      x3      // global jump to virtual address
```

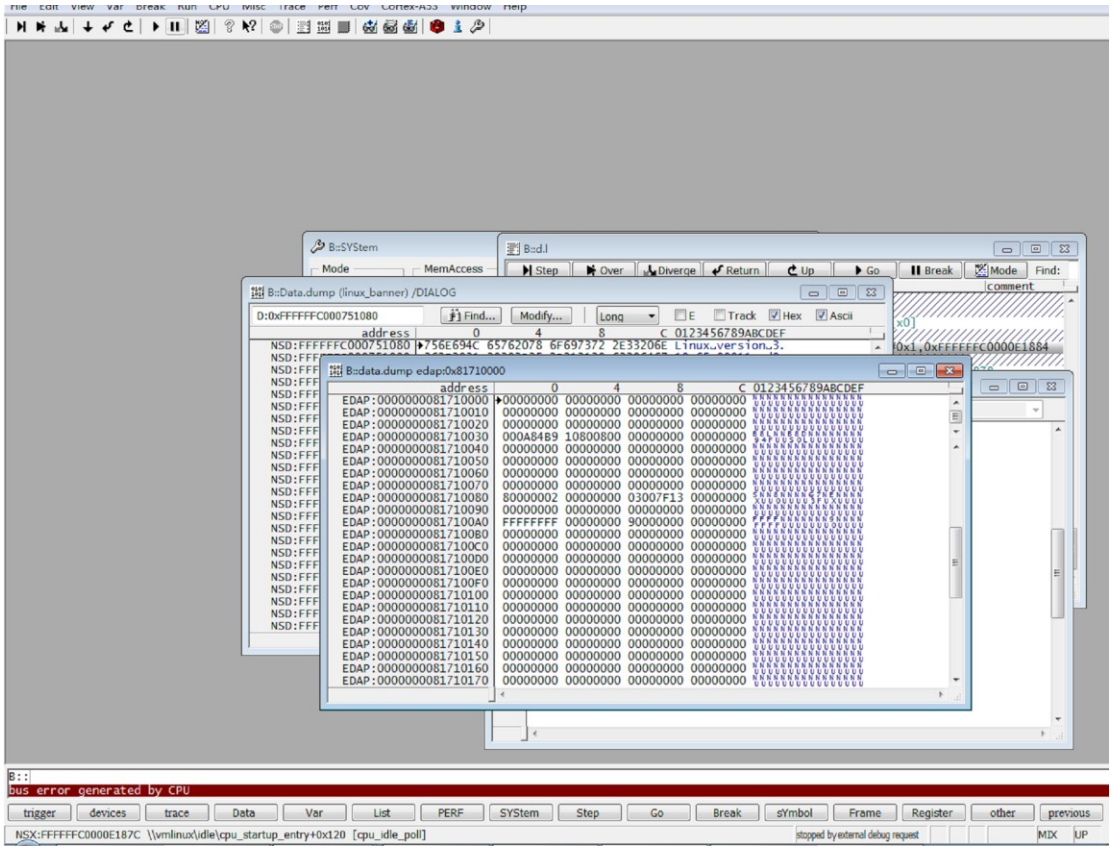
通过进行关大 core，关小 core，关 core pd,cluster pd，调节 PD 的 dts 值等方式确认，定性为小 core 在 cluster 掉电时，大 Core 在 PD 时，问题就会出现。解决办法关掉小 Core 的 cluster pd。

7 Trace32 问题

7.1 Bus error 问题

当用命令 data.dump edap:0x81710000
打开数据 dump 功能时，此时在 running 状态下，如果点击 stop，就会报出下面的 bus error 错误，但是依然可以实现停止功能。

建议在 running 状态下，关闭和 data.dump 相关的窗口，则不会显示上述 bus error 的错误



7.2 Port Fail 问题