9026在珠江数码出现的频繁挂死问题，经过对死机文件的分析，发现问题由主控和线卡之间的板间通讯导致。

20140105

上午主要考虑如何将异常的log保存下来，以及搭建验证环境。下午通过分析串口打印，发现异常log会write到flash上，再在代码中找对read函数的调用，往上跟踪，发现debug模式下有命令可以查看异常log。之后查看前方现网的9026，查看log，发现出问题的点。
void do_tt(int tid)可以打印进程的堆栈信息。
startTaskSuspend 20可以手动构建一个空指针
将do_tt移到map平台不太可行。虽然它依赖的底层函数可以移到平台，但是在调用过程中，需要先ts这个线程，使用ptrace等获取其堆栈，再tr这个线程。依赖关系较多。
tWriteFlash 异常打印通过该函数写到/dev/mtd2下
excLogDump 可打印异常信息。
showExcInFlashOam
system manage-terminal enable
con system debug-user enable
system-monitor task-suspend-reboot函数，可以控制进程挂起后要做的事情，可重启。
system-monitor watchdog函数，可以控制看门狗

20140106

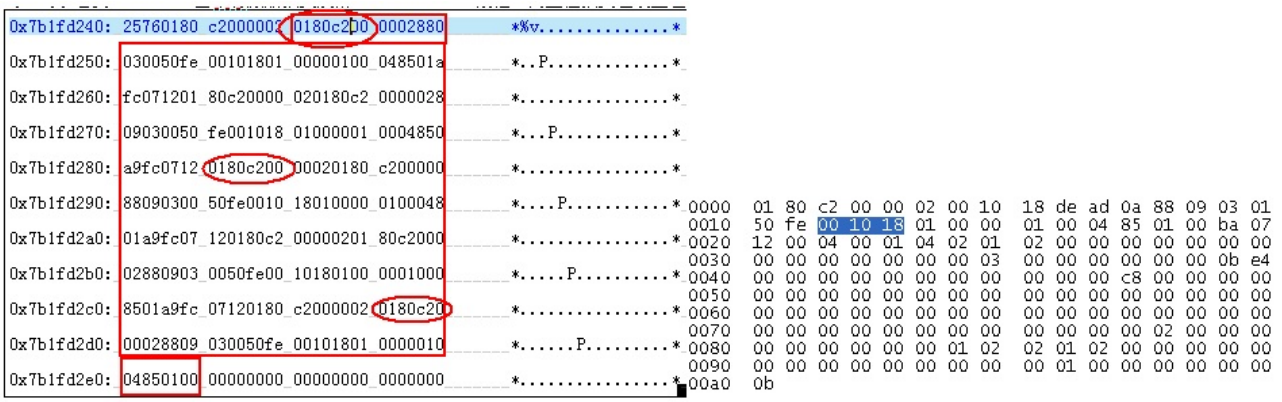
首先反汇编出问题的o文件，但o文件中只有偏移，没有地址，不太好看。另外，确认前方出问题的log对应的软件版本，并从软件版本中提取app.bin，反汇编。进而将代码和异常堆栈中的地址对应起来。
下午，重点分析getQos_ServiceFlowConfigData函数中局部变量和入参的值。其中入参正常，有两个service flow。但serviceFlowCount不正常，其值为0x0180。另外，还从堆栈中找到出问题时recvMsg的消息内容，发现oamtask回给cmcController线程的报文全乱掉了。
之后，在分析堆栈时发现，rcvMsg的实际长度只有156个字节，明显少于实际的长度。那么这段内存是被谁破坏的呢？如果是oam通信过程中被破坏还好说，毕竟是自己的问题。但如果是被的线程破坏就无从查起了。
想到，我们可以抓个包，对比包的内容，看下正常情况下，调用这个函数时rcvMsg的值应该是多少。

总结：
1. 分析故障现场，再怎么强调都不为过。比如严成安分析现场log，发现cmcController线程suspend了。我通过分析tshell异常时的打印，发现异常log有记录到flash中，进而发现MAP平台提供的死机文件功能。田明通过分析异常时的堆栈，结合抓包，完整呈现故障出现时cmcController的现场信息。包括可以通过telnet 9999监控前方的CMC，也是在试图获取现场。
在跟踪故障过程中，要时刻考虑，是否充分了解了现场情况？
1）异常log。show log可看。
2）coredump文件。
3）抓包文件。
4）死机文件。
5）可否远程监控。

2. 从管理的角度，有以下经验：
1）故障负责人需要汇总故障相关信息。进而整理：信息是否完整？现场了解是否充分？应该如何去加速复现？等等。
2）组织有经验的人来头脑风暴。集思广益，为故障寻找突破口。
3）了解全局，考虑所有可能出问题的地方。之前一直怀疑线卡挂死，没有怀疑主控有问题。但是线卡又是一个黑盒子

今天通过分析故障现场的异常日志，一些关键点：
1. 故障由getQos_ServiceFlowConfigData触发。该函数会调用RSSEND发送消息给oamtask线程，后者负责发消息给线卡查询service flow信息，并将线卡发回来的OAM报文中提取payload信息，拼接为一个完整的rspmsg，回给cmccontroller线程。如果正常情况下，rspmsg的最前面两个字节会标记service flow的个数，我们再依赖这个service flow的个数来遍历msg内容。

2. 通过分析CmcController的堆栈，如下左图，是oamtask线程回给CmcController线程的rspmsg。
在这种情况下，rspmsg的内容包含了原始OAM报文的目的mac（0180c2000002）,报文类型（8809，死机文件打印有点问题，最后一个数字没有显示出来，所以只有880）。右图为一个正常的OAM报文。
对比可以看到，oamtask基本将原始报文发给CmcController线程，而且在报文组装上也存在重大问题，导致rcvmsg内容异常。如图，buffer内容出现很多重复。



3. 通过上面的分析，得到初步结论，BRCM的OAM通信框架存在严重问题，在某些情况下会组装报文出错。我们之前没有发现，是因为正常情况下，线卡和主控之间超过150个字节的OAM报文较少，不存在报文组装，所以看不出来。但是我们不能保证，今天在getQos_ServiceFlowConfigData中添加了防护，在其他位置就没有类似错误。

4. 由于我们只有CmcController线程的堆栈信息，完全无法分析现场oamtask线程为什么会出错。而今天在所内尝试复线失败。所以只能通过走查brcm的oam通信框架去分析问题。如何验证，也是个麻烦。总不能拿现场环境去验证。

从前期和BRCM沟通来看，他们对于OAM模式也很少维护，一般用TCP模式较多。理论上，TCP模式也会更稳定。比如至少省去了报文组装的工作。只改通信通道，也不会影响上层业务。

因此，建议尽快启动前方TCP版本的切换。

如下是我们这次出错的关键函数。
ServiceFlowCount就是从oamtask线程回给cmccontroller线程的第一个short变量中读取的，其值为0x0180。然后这个for循环会将srvFlowConfig全局变量后大约100k的内存覆盖。

因此我们今天的规避版本，需要重点检查，是否还有类似的OAM消息相关的，不受控的for循环内存读写。付MM初步走查，这是第一个。

```
for(i = 0 ; i < serviceFlowCount; i++)
{
   /*service flow id*/
   srvFlowConfig[i].serviceFlowId = ntohl(*((WORD32 *)(pSrvFlowConfigEntry+offset)));   /*Index*/
   offset += 4;

   /*direction*/
   srvFlowConfig[i].serviceFlowDirection= *(pSrvFlowConfigEntry + offset);
   offset += 1;

   /*flowType*/
   srvFlowConfig[i].flowType  = *(pSrvFlowConfigEntry + offset);
   offset += 1;

   /*Qos Type*/
   srvFlowConfig[i].serviceFlow_QosType = *(pSrvFlowConfigEntry + offset);
   offset += 1;

   memcpy(srvFlowConfig[i].requiredAttrMask, (pSrvFlowConfigEntry+offset), 4);
   offset += 4;
```

两点想法：
1. 关键进程，如Cmctroller、oamtask等进程suspend时重启单板。上周日晓筱有验证过，但当时考虑suspend后立即重启，没有使用平台现有的suspend后重启机制。
2. 关键log，记录到flash中。目前大家很多log都是使用MPRINT打印到tshell中，但这种打印对现场故障没有帮助。因此建议梳理，哪些关键log，在MPRINT的同时，还要记录到flash。