



## **AX620 平台 应用与内核 Debug 使用说明**

文档版本：V1.0

发布日期：2021/12/28

AXERA CONFIDENTIAL FOR Sipeed

前 言 .....	3
<b>目 录</b>	
修订历史 .....	4
1 Linux 内核 ramdoops .....	5
1.1 简介 .....	6
1.2 使用方法 .....	6
2 Linux memory 转储 .....	7
2.1 使用方法 .....	8
3 Gdb 调试应用程序 .....	12
3.1 简介 .....	13
4 strace 调试应用程序 .....	14
4.1 简介 .....	15

## 权利声明

爱芯元智半导体（上海）有限公司或其许可人保留一切权利。

非经权利人书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## 注意

您购买的产品、服务或特性等应受商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非商业合同另有约定，本公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

AXERA CONFIDENTIAL FOR Speed

## 适用产品



爱芯 AX620

## 前言

## 适读人群

- 软件开发工程师
- 技术支持工程师

## 符号与格式定义

符号/格式	说明
<code>xxx</code>	表示您可以执行的命令行。
<i>斜体</i>	表示变量。如，“安装目录/AX620A_SDK_Vx.x.x/build 目录”中的“安装目录”是一个变量，由您的实际环境决定。
 说明/备注：	表示您在使用产品的过程中，我们向您说明的事项。
 注意：	表示您在使用产品的过程中，需要您特别注意的事项。

文档版本	发布时间	修订说明
V1.0	2021/12/28	文档初版

### 修订历史

AXERA CONFIDENTIAL FOR Sipeed

本章节包含：

## 1 Linux 内核 ramdoops

[错误!未找到引用源。](#) [简介](#)

[错误!未找到引用源。](#) [使用方法](#)

AXERA CONFIDENTIAL FOR Sipeed

## 1.1 简介

ramoops 是采用 ram 保存 oops 信息的一种技术，该技术使用 pstore 机制实现，在内核开关中用 3 个开关控制：PSTORE\_CONSOLE 控制是否保存控制台输出，PSTORE\_FTRACE 控制是否保存函数调用序列，PSTORE\_RAM 控制是否保存 panic/oops 信息。

## 1.2 使用方法

AX620 的内核已经集成了 ramoops 功能当内核崩溃后重启系统会看到在 /sys/fs/pstore 目录下有节点 console-ramoops-0 与 dmesg-ramoops-0 使用 cat 命令就可以读出 log 信息，注意因为 log 保存在 ram 中的所以当系统崩溃后只能选着让系统热重启即复位。

```
/sys/fs/pstore
# ls
console-ramoops-0  dmesg-ramoops-0
```

图 1

本章节包含：

## 2 Linux memory 转储

错误!未找到引用源。 使用方法

AXERA CONFIDENTIAL FOR Sipeed



## 2.1 使用方法

1. ax620 会将 dump 文件保存的 sd card 中，将 dump 的文件和 wmlinux 放在一起执行命令 `./crash_arm32 wmlinux vmcore.dmp.20211227204526`，crash\_arm32 放在 sd 中的 tools/crash\_tools 目录下。级联方案获取从机的 log 的方法是 `axdl20 -p AX620_slave_V0.20.0_20211227174945.pac --dmp=1`（注意是两个“-”）

```
GNU gdb (GDB) 7.6
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-unknown-linux-gnu --target=aarch64-elf-linux"...
```

```
KERNEL: wmlinux
DUMPFILE: vmcore
CPUS: 4
DATE: Thu Oct 22 11:03:21 CST 2020
UPTIME: 00:03:26
LOAD AVERAGE: 0.02, 0.04, 0.01
TASKS: 85
NODENAME: aichip
RELEASE: 4.19.125
VERSION: #10 SMP PREEMPT Thu Oct 22 10:39:55 CST 2020
MACHINE: aarch64 (unknown Mhz)
MEMORY: 4 GB
PANIC: "Unable to handle kernel NULL pointer dereference at virtual address 0000000000000000"
PID: 1146
COMMAND: "sh"
TASK: ffff80002f2d8000 [THREAD_INFO: ffff80002f2d8000]
CPU: 0
STATE: TASK_RUNNING (PANIC)
```

crash> █

图 5

2. 执行 help 查看当前 crash 支持的命令，支持的命令比较多简单介绍几个

```
crash> help
```

*	extend	log	rd	task
alias	files	mach	repeat	timer
ascii	foreach	mod	runq	tree
bpf	fuser	mount	search	union
bt	gdb	net	set	vm
bttop	help	p	sig	vtop
dev	ipcs	ps	struct	waitq
dis	irq	pte	swap	whatis
eval	kmem	ptob	sym	wr
exit	list	ptov	sys	q

图 6

3. Log 命令查看内核 dmesg log

```

[ 206.630139] x5 : 0000000000000000 x4 : 0000000000000001
[ 206.635442] x3 : 0000000000000007 x2 : 0000000000000006
[ 206.640745] x1 : a39e9d43e7a70800 x0 : 0000000000000000
[ 206.646048] Call trace:
[ 206.648491] machine_kexec+0x44/0x2a8
[ 206.652149] __crash_kexec+0x7c/0x128
[ 206.655805] crash_kexec+0x6c/0x80
[ 206.659203] die+0x11c/0x1e0
[ 206.662081] die_kernel_fault+0x60/0x70
[ 206.665910] __do_kernel_fault+0x88/0xa8
[ 206.669826] do_page_fault+0x6c/0x480
[ 206.673482] do_translation_fault+0x58/0x60
[ 206.677658] do_mem_abort+0x54/0x100
[ 206.681228] ell_da+0x20/0x80
[ 206.684190] sysrq_handle_crash+0x20/0x30
[ 206.688193] __handle_sysrq+0x9c/0x198
[ 206.691936] write_sysrq_trigger+0x64/0x80
[ 206.696025] proc_reg_write+0x60/0xd8
[ 206.699682] __vfs_write+0x30/0x168
[ 206.703165] vfs_write+0xa4/0x1a8
[ 206.706475] ksys_write+0x64/0xe8
[ 206.709785] __arm64_sys_write+0x18/0x20
[ 206.713702] el0_svc_common+0x6c/0x178
[ 206.717446] el0_svc_handler+0x24/0x80
[ 206.721188] el0_svc+0x8/0xc
[ 206.724062] ---[ end trace b6f951f944193ff2 ]---
[ 206.728673] Bye!

```

图 7

#### 4. bt 命令查看内核 crash 栈

```

crash> bt
PID: 1146 TASK: ffff80002f2d8000 CPU: 0 COMMAND: "sh"
#0 [a39e9d43e7a70800] machine_kexec at ffff00000809ade4
PC: 0000ffffbef58048 LR: 00000000004099f4 SP: 0000ffff8bd990
X29: 0000ffff8bdfaf X28: 000000001088b525 X27: 000000000048a6a0
X26: 00000000004c0000 X25: 0000000000000000 X24: 0000000000000020
X23: 000000001088b140 X22: 00000000004c0000 X21: 0000000000000002
X20: 000000001088b140 X19: 0000000000000001 X18: 0000000000000836
X17: 0000ffffbef58060 X16: 0000000000000000 X15: 0000000000000008
X14: 0000ffffbeeabc20 X13: 726567676972742d X12: 0101010101010101
X11: 0000000000000000 X10: 0101010101010101 X9: ffffffff8000ffff0
X8: 0000000000000000 X7: 0000000000000021 X6: 0080808080808080
X5: 0000000000000000 X4: 0000000000000063 X3: 000000001088b141
X2: 0000000000000002 X1: 000000001088b140 X0: 0000000000000001
ORIG X0: 0000000000000001 SYSCALLNO: 40 PSTATE: 80000000

```

图 8

#### 5. ps 命令查看系统的系统进程和内核线程

1039	2	0	ffff80002ebadcc0	ID	0.0	0	0	[mmc_complete]
1042	2	1	ffff80002ebacf80	ID	0.0	0	0	[kworker/1:1H]
1043	2	0	ffff80002eb14f80	ID	0.0	0	0	[kworker/0:1H]
1053	2	3	ffff80002eb05cc0	ID	0.0	0	0	[kworker/3:1H]
1054	2	2	ffff80002eb04240	IN	0.0	0	0	[jbd2/mmcblk0p3-]
1055	2	2	ffff80002ebac240	ID	0.0	0	0	[ext4-rsv-conver]
1057	2	3	ffff80002eba8d40	ID	0.0	0	0	[kworker/3:2H]
1068	2	2	ffff80002eba8000	ID	0.0	0	0	[kworker/2:1H]
1073	1	1	ffff80002ebc6a00	IN	0.0	2596	1688	syslogd
1077	1	2	ffff80002ebc27c0	IN	0.0	2596	1684	klogd
1092	2	0	ffff80002eb04f80	ID	0.0	0	0	[kworker/0:2]
1093	1	2	ffff80002ebc4f80	IN	0.0	2596	1564	udhcpc
1107	1	3	ffff80002ebaa7c0	IN	0.0	2728	1752	crond
1130	2	2	ffff80002eef27c0	ID	0.0	0	0	[kworker/2:2H]
1131	2	0	ffff80002eb11a80	ID	0.0	0	0	[kworker/0:2H]
1146	1	0	ffff80002f2d8000	RU	0.0	2728	1944	sh
1180	2	3	ffff80002f2db500	ID	0.0	0	0	[kworker/3:2]
1190	1	0	ffff80002ecf1a80	IN	0.0	6144	2424	sshd
1201	1	2	ffff80002f2d9a80	IN	0.0	2596	672	udhcpc

图 9

#### 6. Mod 查看内核加载了哪些模块

crash> mod					
MODULE	NAME	SIZE	OBJECT	FILE	
ffff0000008b2200	ax_isp	16384	(not loaded)	[CONFIG_KALLSYMS]	
ffff0000008bb2c0	ax_uirq	16384	(not loaded)	[CONFIG_KALLSYMS]	
ffff0000008c6280	ax_sys	28672	(not loaded)	[CONFIG_KALLSYMS]	
ffff0000008d2240	ax_venc	20480	(not loaded)	[CONFIG_KALLSYMS]	
ffff0000008db0c0	uio_pdrv_genirq	16384	(not loaded)	[CONFIG_KALLSYMS]	
ffff0000008ea700	ax_vdec	53248	(not loaded)	[CONFIG_KALLSYMS]	
ffff0000008fb2c0	ax_jpegenc	40960	(not loaded)	[CONFIG_KALLSYMS]	
ffff00000090a280	ax_jpegdec	40960	(not loaded)	[CONFIG_KALLSYMS]	

图 10

#### 7. sym -l <模块的名字> 查看模块的符号表

```
crash> sym -l ax_isp
ffff000008080000 (t) .head.text
ffff000008080000 (t) _head
ffff000008080000 (T) _text
ffff000008080800 (t) .text
ffff000008080800 (T) __exception_text_start
ffff000008080800 (T) _stext
ffff000008080800 (T) do_undefinstr
ffff000008080a98 (T) do_sysinstr
ffff000008080b08 (T) do_mem_abort
ffff000008080c08 (T) do_el0_irq_bp_hardening
ffff000008080c58 (T) do_el0_ia_bp_hardening
ffff000008080cf0 (T) do_sp_pc_abort
ffff000008080e10 (T) do_debug_exception
ffff000008080fa0 (t) gic_handle_irq
ffff000008081048 (t) gic_handle_irq
ffff0000080811c8 (T) __do_softirq
ffff0000080811c8 (T) __exception_text_end
ffff0000080811c8 (T) __irqentry_text_end
ffff0000080811c8 (T) __irqentry_text_start
ffff0000080811c8 (T) __softirqentry_text_start
ffff0000080813d8 (T) __entry_text_start
ffff0000080813d8 (T) __softirqentry_text_end
ffff000008081800 (T) vectors
```

图 11

crash 工具功能很强大这里只介绍了几个命令的功能，读者可以根据需要查看 crash 的相关手册。

章节包含:

## 3 Gdb 调试应用程序

错误!未找到引用源。 简介

AXERA CONFIDENTIAL FOR Sipeed

### 3.1 简介

应用程序一般使用 gdb 来调试需要在应用程序编译的选项中加-g，保存应用程序的 core 文件是非常有必要的，要生成 core 文件设置保存路径需要一系列的设置，如下设置是在 AX620 平台下的一些设置。ulimit -c 用来查询当前 core dump 是否打开，查询结果是 0 表示 core dump 是关闭的。可以使用 ulimit -c unlimited 打开，当应用程序发生 Segmentation fault 是就会生成 core 文件。echo "/opt/core-%e-%p-%t" > /proc/sys/kernel/core\_pattern 设置 core dump 的保存路径。直接使用 gdb <应用程序的名字> <core 文件的名字> 即可调试。

AXERA CONFIDENTIAL FOR Sipeeta

章节包含:

## 4 strace 调试应用程序

错误!未找到引用源。 简介

AXERA CONFIDENTIAL FOR Sipeed

## 4.1 简介

strace 是一个强大的应用层调试工具，他的特点是简单好用，局限是只能跟踪到系统调用层，以下是一些经常使用的例子更多的使用请大家网上搜索。

1.跟踪应用程序中一些线程的状态。首先使用 ps -T 显示进程的线程号，使用 strace -p <线程号> 就可以跟踪线程当前的状态

```
# strace -p 1201
strace: Process 1201 attached
ppoll([{fd=3, events=POLLIN}, {fd=-1}], 2, {tv_sec=43038, tv_nsec=434339673}, NULL, 0
```

图 12

2.查看某个程序或者命令执行过程中做了什么调用了哪些库文件

strace date -s "2020-10-20" 可以看到使用系统调用 settimeofday 设置了系统时间

```
getuid() = 0
brk(NULL) = 0x3dfdc000
brk(0x3dffd000) = 0x3dffd000
openat(AT_FDCWD, "/etc/localtime", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/localtime", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
settimeofday({tv_sec=1603152000, tv_usec=0}, NULL) = 0
fstat(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(0x5, 0x1), ...}) = 0
ioctl(1, TCGETS, {B115200 opost isig icanon echo ...}) = 0
write(1, "Tue Oct 20 00:00:00 UTC 2020\n", 29Tue Oct 20 00:00:00 UTC 2020
) = 29
```

图 13

3.跟踪系统调用执行所需要的时间 strace -tt -T date -s "2020-10-20"

```
00:00:41.335625 getuid() = 0 <0.000015>
00:00:41.335778 brk(NULL) = 0x366e9000 <0.000015>
00:00:41.335844 brk(0x3670a000) = 0x3670a000 <0.000020>
00:00:41.335930 openat(AT_FDCWD, "/etc/localtime", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory) <0.000020>
00:00:41.336075 openat(AT_FDCWD, "/etc/localtime", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory) <0.000019>
00:00:41.336159 settimeofday({tv_sec=1603152000, tv_usec=0}, NULL) = 0 <0.000030>
00:00:00.000058 fstat(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(0x5, 0x1), ...}) = 0 <0.000017>
00:00:00.000134 ioctl(1, TCGETS, {B115200 opost isig icanon echo ...}) = 0 <0.000023>
00:00:00.000225 write(1, "Tue Oct 20 00:00:00 UTC 2020\n", 29Tue Oct 20 00:00:00 UTC 2020
```

图 14