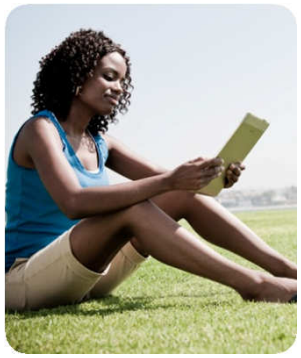
Debugging and Triaging Linux Kernel Issues

Linux Kernel Team
05/20/2011

WELCOME

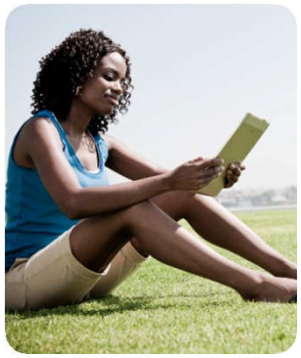


QuIC
QUALCOMM INNOVATION CENTER, INC.

Agenda

- Workflow
- Architecture Basics
- Hands on Demos
- Break
- Debugging Sleep Issues
- Watch dog/Reset Debugging
- Fusion Debugging
- Kernel Debug Options
- GPIOs
- Subsystem Restart

WORKFLOW

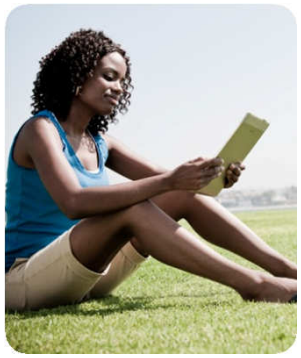


QuIC
QUALCOMM INNOVATION CENTER, INC.

Kernel Stability Workflow

- Failure Analysis Flow
  1. Open DRIODBUG go/droidbug
  2. Android Stability Team Triage
  3. Kernel Stability Team (Label: ANDROID_KERNEL_STABILITY_ISSUE)
     - Issues are distributed across the Kernel Stability Team (Daily)
     - Determine the correct POC, request tester to open a CR, or mark issues as a duplicate. (Goal of 1-2 day turn around)
     - Close DROIDBUG if CR is created or if issue is a Duplicate.
     - Remove Label if POC is determine to be outside the team
- kernel.stability should **not** be added to all kernel failures
- Kernel Stability Team is **not** 1st level of triage for kernel failures
- DROIDBUG is not meant to get CR status or communicate requests for debugging strategy

ARCH BASICS



QuIC
QUALCOMM INNOVATION CENTER, INC.

Architecture Cheat Sheet

- SP = R13
- LR = R14
- PC = R15
  - ARM (32)/Thumb (16)
  - Word/Halfword aligned
- PSR
  - cpsr/spsr

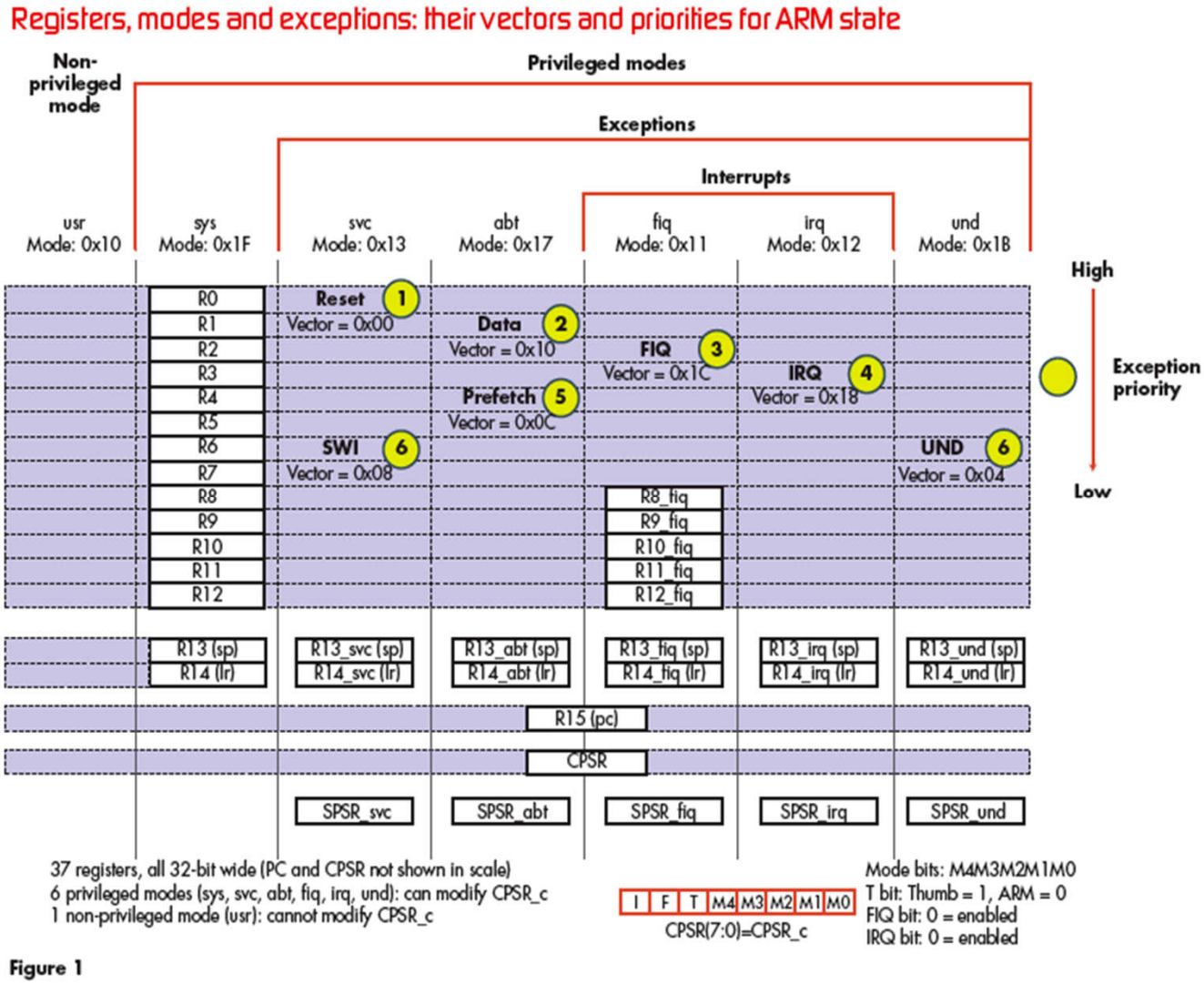


Figure 1

Exceptions

- Events that alters the normal sequence of execution and force the processor to execute special instructions in a privileged state

- Synchronous
  - Ex. Page fault, divide by zero
- Asynchronous
  - Interrupts

- An interrupt or an exception handler is not a process
  - It is a kernel control path

Synchronous Exceptions

- FAULT
  - Program allowed to restart after condition is corrected
  - No loss of continuity
    - E.g. Page FAULT
- TRAPS
  - Triggered when there is no need to re-execute the terminated instruction
- ABORT
  - Serious Error, Program terminated

Exception Handling

- ARMv7 has 8 Different Exceptions
  - reset, 0x00
    undefined instruction, 0x04
    supervisor call (svc), 0x08
    pre-fetch abort, 0x0C
    data abort, 0x10
    not used, 0x14
    irq 0x18
    fiq 0x1C

- V bit of SCTLR register
  - Base Address:
    - V = 0 exception base address = 0x00000000
    - V = 1 exception base address = 0xffff0000 (Virtual Address)

Exception Handling

- Linux implements all Exceptions
- All exception **handlers** run in **SVC** Mode
- Two step Process in Linux
  - Exception Stub – Stack Maintenance and Switch to SVC Mode
  - Exception Specific Op (ASM) -> Exception Handler (C Code)

Common Exceptions

- Data Abort:
  - Data Access Transaction Failed
  - Load/Store Instructions
  - Precise: Internal Aborts from the core
    - MMU Protection Fault
  - Imprecise: typically External Aborts
    - Unrecoverable
- Pre-fetch Abort:
  - Unable to fetch instruction from Memory
  - Architecture returns information in
    - FSR (Fault Status Register) of MMU
    - FAR (Fault Address Register)

Virtual Memory Layout

<5>[0.000000] Virtual kernel memory layout:

vector : 0xffff0000 - 0xffff1000 (4 kB)

fixmap : 0xfff00000 - 0xfffe0000 (896 kB)

DMA : 0xff000000 - 0xffe00000 (14 MB)

vmalloc : 0xe0000000 - 0xfa000000 (416 MB)

Lowmem : 0xc0000000 - 0xdfe00000 (510 MB)

.init : 0xc0008000 - 0xc0040000 (224 kB)

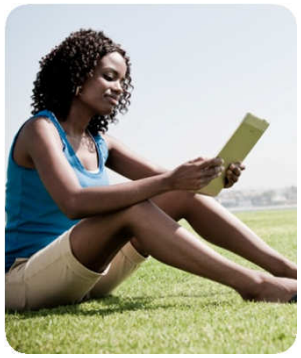
.text : 0xc0100000 - 0xc0925000 (8340 kB)

.data : 0xc0a00000 - 0xc0ab3460 (718 kB)

Kernel Log and Sanity Checks

- Quick Way to extract without vmlinux or symbols
  - $strings <ramdump> under Linux
  - search for “Linux version 2.6.”
- Machine ID
  - Check that Machine ID is same as Target HW
    - <4>[0.000000] Machine: QCT MSM8X55 SURF
    - Typically wrong/missing boot loader images
- Check Memory Initialization
  - “Total Memory Size = MB”
  - Errors ?
    - <1>[0.000732] BUG: Bad page state in process swapper pfn:008c0
    - Typically defective hardware
- Check socinfo initialization
  - Incompatible Modem Image flashed

HANDS ON



QuIC
QUALCOMM INNOVATION CENTER, INC.

HANDS-ON

- Kernel Panic
  - What is useful and what is not
  - Register Dump
  - PC, LR
  - Objdump
  - Backtrace (the 1st one)
  - Code Walkthrough
  - git log –decorate --raw
  - git blame

HANDS-ON

- <1>[41.987497] Unable to handle kernel NULL pointer dereference at virtual address 0000005f
- Register Dump
- PC – Program Counter (Crash Instruction)
- LR – Link Register

- <4>[41.987670] PC is at free_vm_area+0x8/0x30
- <4>[41.987702] LR is at seemingly_unmalicious_func+0x20/0x38
- <4>[41.987730] pc : [<c014e428>] lr : [<c0329014>] psr: 80000193
- <4>[41.987746] sp : d8e4fd70 ip : c08c84b8 fp : c0880ef0
- <4>[41.987767] r10: c0d38f29 r9 : 0000002d r8 : 00000000
- <4>[41.987790] r7 : 0000002d r6 : c08db2a4 r5 : c08dafd0 r4 : 0000005b
- <4>[41.987816] r3 : 00000064 r2 : d8e4fd7c r1 : c07a4ff3 r0 : 0000005b

HANDS-ON

- Lets make sense out of the numbers

- Objdump
- arm-eabi-objdump –D <object_file>
- Location: - android/out/target/product/msm8660_surf/obj/KERNEL_OBJ/

- Sample output:
- **Address** **Content** **Meaning**
- 158c: e2844001 add r4, r4, #1

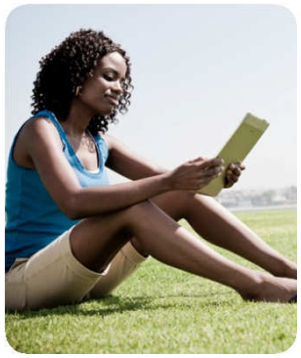
HANDS-ON

- So which instruction actually caused this crash ?
  - Why?
  - Proof ?
- Where did this call come from ?
  - Stack-dump
- Million $$ Question - Who's at fault ?
  - Code walkthrough
- Who did this – git log
  - git log –decorate –raw
- Really, who did this ? - git blame

HANDS-ON

- Useful GUI Tools
- go/grok – Source Code Search
- go/kernel – GIT Web

BACKTRACES



QuIC
QUALCOMM INNOVATION CENTER, INC.

Outline

- Introduction
- Types of backtraces
  - kernel panic
  - WARN_ON
  - BUG_ON
  - scheduling while atomic
  - Other backtraces

Backtraces

- Should ONLY be printed out if there is a bug
  - If you see a backtrace someone, somewhere is doing something incorrect
- Designed to give an idea of the code path flow
- Not all backtraces are kernel panics!
  - Does not mean backtraces should be ignored

Kernel Panic

- Occurs when the system has hit some fatal condition
  - Generally the system could not access memory
- Will see many backtraces
  - <0>[15553.953014] Kernel panic - not syncing: Fatal exception
  - <4>[15553.957628] [<c0107a84>] (unwind_backtrace+0x0/0x164) from [<c068c124>] (panic+0x6c/0xe8)
  - <4>[15553.965722] [<c068c124>] (panic+0x6c/0xe8) from [<c0105390>] (die+0x17c/0x1bc)
  - <4>[15553.972915] [<c0105390>] (die+0x17c/0x1bc) from [<c010b2b0>] (__do_kernel_fault+0x64/0x84)
  - <4>[15553.981170] [<c010b2b0>] (__do_kernel_fault+0x64/0x84) from [<c010b564>] (do_page_fault+0x294/0x2b4)
  - <4>[15553.990287] [<c010b564>] (do_page_fault+0x294/0x2b4) from [<c010044c>] (do_PrefetchAbort+0x34/0x94)
  - <4>[15553.999312] [<c010044c>] (do_DataAbort+0x34/0x94) from [<c0100df0>] (__pabt_svc+0x50/0xa0)
  - This backtrace or a variation (do_PrefetchAbort) appears in every panic
  - Just sending this to a tech team is not enough to diagnose the problem
- <4>[15553.857215] [<c01ee2ac>] (__do_fault+0x50/0x3cc) from [<c01ef400>] (handle_mm_fault+0x348/0x704)
- <4>[15553.865983] [<c01ef400>] (handle_mm_fault+0x348/0x704) from [<c010b3c4>] (do_page_fault+0xf4/0x2b4)
- <4>[15553.875011] [<c010b3c4>] (do_page_fault+0xf4/0x2b4) from [<c010044c>] (do_PrefetchAbort+0x34/0x94)
- <4>[15553.883946] [<c010044c>] (do_PrefetchAbort+0x34/0x94) from [<c0101120>] (ret_from_exception+0x0/0x10)
- This backtrace actually shows where the problem occured

WARN_ON

- System has detected something bad but not fatal
  - Should not be ignored!
- <4>[82.856316] ------------[cut here]------------
- <4>[82.860949] WARNING: at /local/mnt/workspace/lauraa/gingerbread/kernel/arch/arm/mach-msm/sdio_al.c:2039 sdio_al_wake_up+0x368/0x3bc()
- <4>[82.873343] Modules linked in:
- <4>[82.875972] [<c00522d4>] (unwind_backtrace+0x0/0x128) from [<c00c0d68>] (warn_slowpath_common+0x4c/0x64)
- <4>[82.886360] [<c00c0d68>] (warn_slowpath_common+0x4c/0x64) from [<c00c0d98>] (warn_slowpath_null+0x18/0x1c)
- <4>[82.896551] [<c00c0d98>] (warn_slowpath_null+0x18/0x1c) from [<c005ef78>](sdio_al_wake_up+0x368/0x3bc)
- <4>[82.905981] [<c005ef78>] (sdio_al_wake_up+0x368/0x3bc) from [<c005f4ac>] (sdio_al_subsys_notifier_cb+0x174/0x2d0)
- <4>[82.916261] [<c005f4ac>] (sdio_al_subsys_notifier_cb+0x174/0x2d0) from [<c00dea00>] (notifier_call_chain+0x2c/0x70)
- <4>[82.926696] [<c00dea00>] (notifier_call_chain+0x2c/0x70) from [<c00deb60>] (__srcu_notifier_call_chain+0x40/0x58)
- <4>[82.936953] [<c00deb60>] (__srcu_notifier_call_chain+0x40/0x58) from [<c00deb8c>] (srcu_notifier_call_chain+0x14/0x18)
- <4>[82.947645] [<c00deb8c>] (srcu_notifier_call_chain+0x14/0x18) from [<c009e2c0>] (subsystem_restart_thread+0xac/0x250)
- <4>[82.958252] [<c009e2c0>] (subsystem_restart_thread+0xac/0x250) from [<c00d9628>] (kthread+0x80/0x88)^M
- <4>[82.967370] [<c00d9628>] (kthread+0x80/0x88) from [<c004c7a0>] (kernel_thread_exit+0x0/0x8)
- <4>[82.975754] ---[end trace 1b75b31a2719ed1f]---
- Generally give the location where the warning occurred

BUG_ON

- Variation on kernel panic
  - Programmer put some checks in to detect bad state
  - If bad state occurs, kernel intentionally dereferences NULL pointer to crash system
- Will see __bug in the backtrace

<4>[17.175276] [<c0031718>] **(__bug+0x18/0x24**) from [<c02361f4>] (kgsl_yamato_idle+0xdc/0x100)
<4>[17.175319] [<c02361f4>] (kgsl_yamato_idle+0xdc/0x100) from [<c0233468>] (kgsl_ringbuffer_start+0x8f8/0xa60)
<4>[17.175361] [<c0233468>] (kgsl_ringbuffer_start+0x8f8/0xa60) from [<c0235f68>] (kgsl_yamato_start+0x380/0x4a8)
<4>[17.175403] [<c0235f68>] (kgsl_yamato_start+0x380/0x4a8) from [<c022fa14>] (kgsl_open+0x2ec/0x420)
<4>[17.175451] [<c022fa14>] (kgsl_open+0x2ec/0x420) from [<c0105448>] (chrdev_open+0x1dc/0x1f8)
<4>[17.175489] [<c0105448>] (chrdev_open+0x1dc/0x1f8) from [<c0100d44>] (__dentry_open+0x17c/0x294)
<4>[17.175526] [<c0100d44>] (__dentry_open+0x17c/0x294) from [<c0100f18>] (nameidata_to_filp+0x3c/0x50)
<4>[17.175566] [<c0100f18>] (nameidata_to_filp+0x3c/0x50) from [<c010c8e0>] (do_last+0x508/0x65c)
<4>[17.175604] [<c010c8e0>] (do_last+0x508/0x65c) from [<c010e548>] (do_filp_open+0x17c/0x508)
<4>[17.175638] [<c010e548>] (do_filp_open+0x17c/0x508) from [<c0100ae8>] (do_sys_open+0x58/0x10c)
<4>[17.175674] [<c0100ae8>] (do_sys_open+0x58/0x10c) from [<c002de80>] (ret_fast_syscall+0x0/0x30)

- Right before the kernel panic will give file line number info of the check failed

- <2>[17.170371] kernel BUG at /local/mnt/workspace/lauraa/froyo/kernel/drivers/gpu/msm/kgsl_yamato.c:1026!
- <1>[17.170406] Unable to handle kernel NULL pointer dereference at virtual address 00000000

Scheduling while atomic

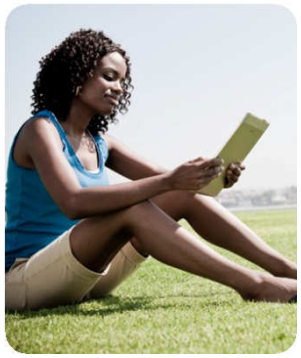
- When handling and IRQ or interrupts are disabled can't call schedule
  - IRQs and interrupts are supposed to be quick, scheduling is long
  - Kernel gets very noisy

- <3>[43469.022412] **bad: scheduling from the idle thread!**
- <4>[43469.022480] [] (unwind_backtrace+0x0/0x164) from [] (dequeue_task_idle+0x1c/0x2c)
- <4>[43469.022535] [] (dequeue_task_idle+0x1c/0x2c) from [] (dequeue_task+0x48/0x54)
- <4>[43469.022580] [] (dequeue_task+0x48/0x54) from [] (deactivate_task+0x34/0x44)
- <4>[43469.022629] [] (deactivate_task+0x34/0x44) from [] (schedule+0x10c/0x528)
- <4>[43469.022675] [] (schedule+0x10c/0x528) from [] (schedule_timeout+0x1d0/0x210)
- <4>[43469.022719] [] (schedule_timeout+0x1d0/0x210) from [] (io_schedule_timeout+0x54/0x84)
- <4>[43469.022780] [] (io_schedule_timeout+0x54/0x84) from [] (congestion_wait+0x6c/0x90)
- <4>[43469.022831] [] (congestion_wait+0x6c/0x90) from [] (shrink_inactive_list+0x244/0x4cc)
- <4>[43469.022879] [] (shrink_inactive_list+0x244/0x4cc) from [] (shrink_zone+0x354/0x41c)
- <4>[43469.022927] [] (shrink_zone+0x354/0x41c) from [] (try_to_free_pages+0x1e4/0x3f4)
- <4>[43469.022973] [] (try_to_free_pages+0x1e4/0x3f4) from [] (__alloc_pages_nodemask+0x350/0x564)
- <4>[43469.023022] [] (__alloc_pages_nodemask+0x350/0x564) from [] (__slab_alloc+0x220/0x4f8)
- <4>[43469.023068] [] (__slab_alloc+0x220/0x4f8) from [] (__kmalloc_track_caller+0xb0/0x108)
- <4>[43469.023120] [] (__kmalloc_track_caller+0xb0/0x108) from [] (__alloc_skb+0x50/0xe0)
- <4>[43469.023179] [] (__alloc_skb+0x50/0xe0) from [] (rmnet_rx_submit.clone.0+0x20/0x88)
- <4>[43469.023230] [] (**rmnet_rx_submit.clone**.0+0x20/0x88) from [] (rmnet_set_alt+0x1e4/0x2a0)
- <4>[43469.023287] [] (rmnet_set_alt+0x1e4/0x2a0) from [] (composite_setup+0x9f8/0xe8c)
- <4>[43469.023337] [] (composite_setup+0x9f8/0xe8c) from [] (usb_interrupt+0x678/0x960)
- <4>[43469.023385] [] (usb_interrupt+0x678/0x960) from [] (handle_IRQ_event+0x24/0xc4)
- <4>[43469.023431] [] (handle_IRQ_event+0x24/0xc4) from [] (handle_level_irq+0xc8/0x144)
- <4>[43469.023482] [] (handle_level_irq+0xc8/0x144) from [] (asm_do_IRQ+0x8c/0xcc)
- <4>[43469.023529] [] (asm_do_IRQ+0x8c/0xcc) from [] (__irq_svc+0x4c/0xe4)

Other backtraces

- The kernel can print backtraces for many other reasons
- Things to keep in mind
  - If a backtrace is printed out someone is doing something wrong
  - Backtraces != kernel panic
    - but they should always be investigated
- Look for context around the backtrace as well. most bactraces are printed out with a message of why they are being printed out

BREAK

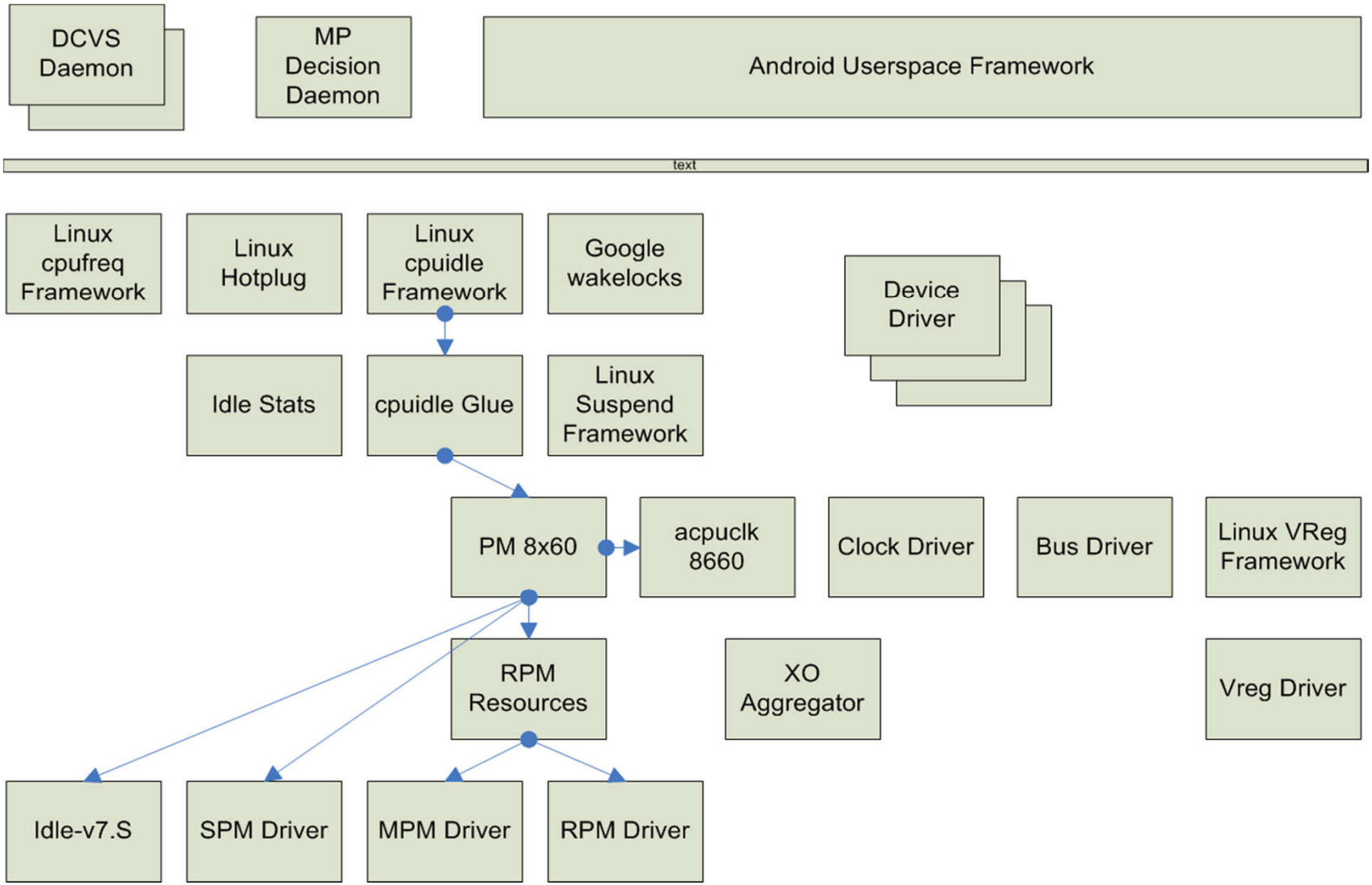


QuIC
QUALCOMM INNOVATION CENTER, INC.

Debugging Sleep Issues

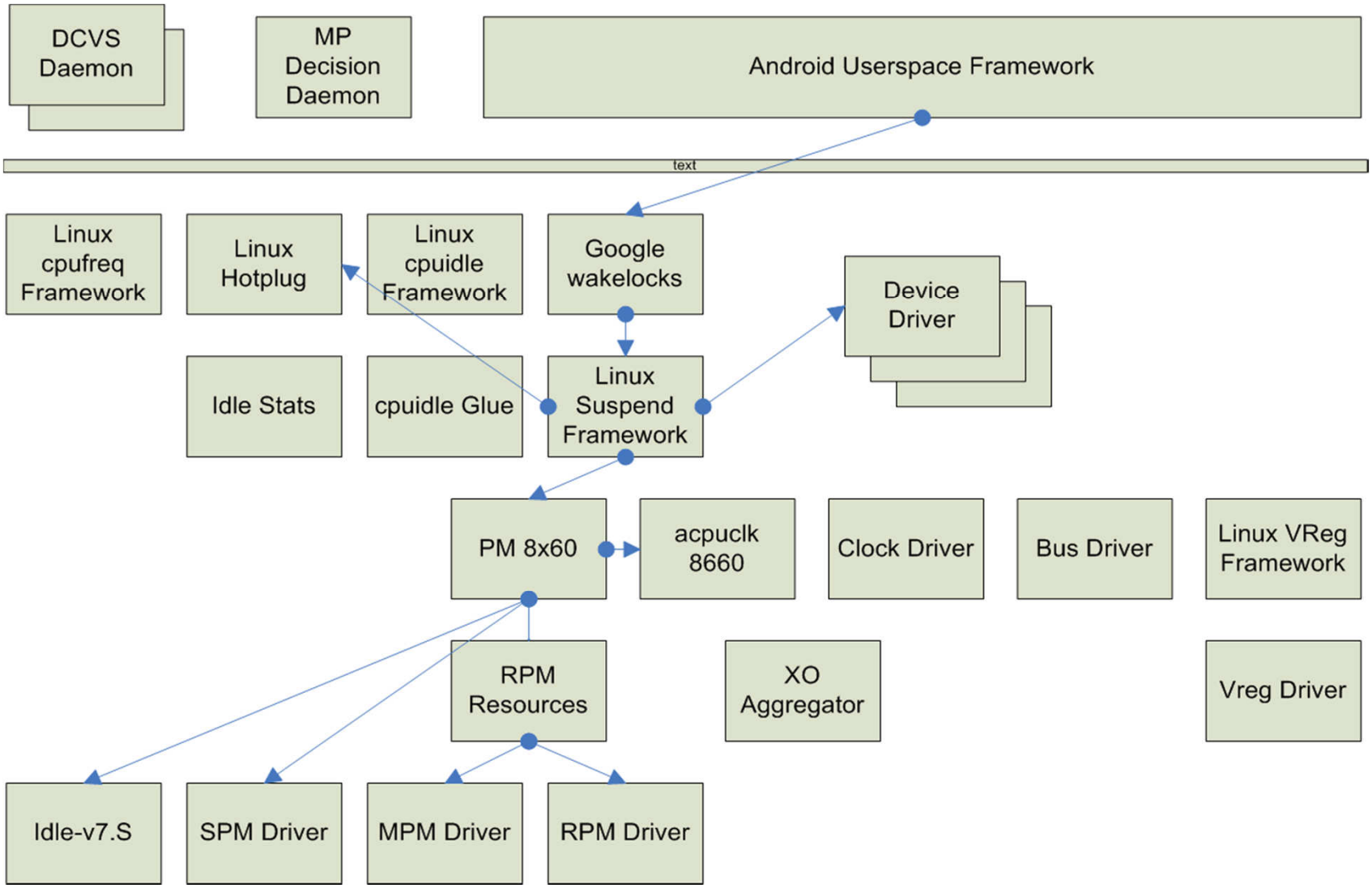
QuIC
QUALCOMM INNOVATION CENTER, INC.

Idle Sleep Use Case



Courtesy: 80-VG968-1C

Suspend Sleep Use Case



Courtesy: 80-VG968-1C

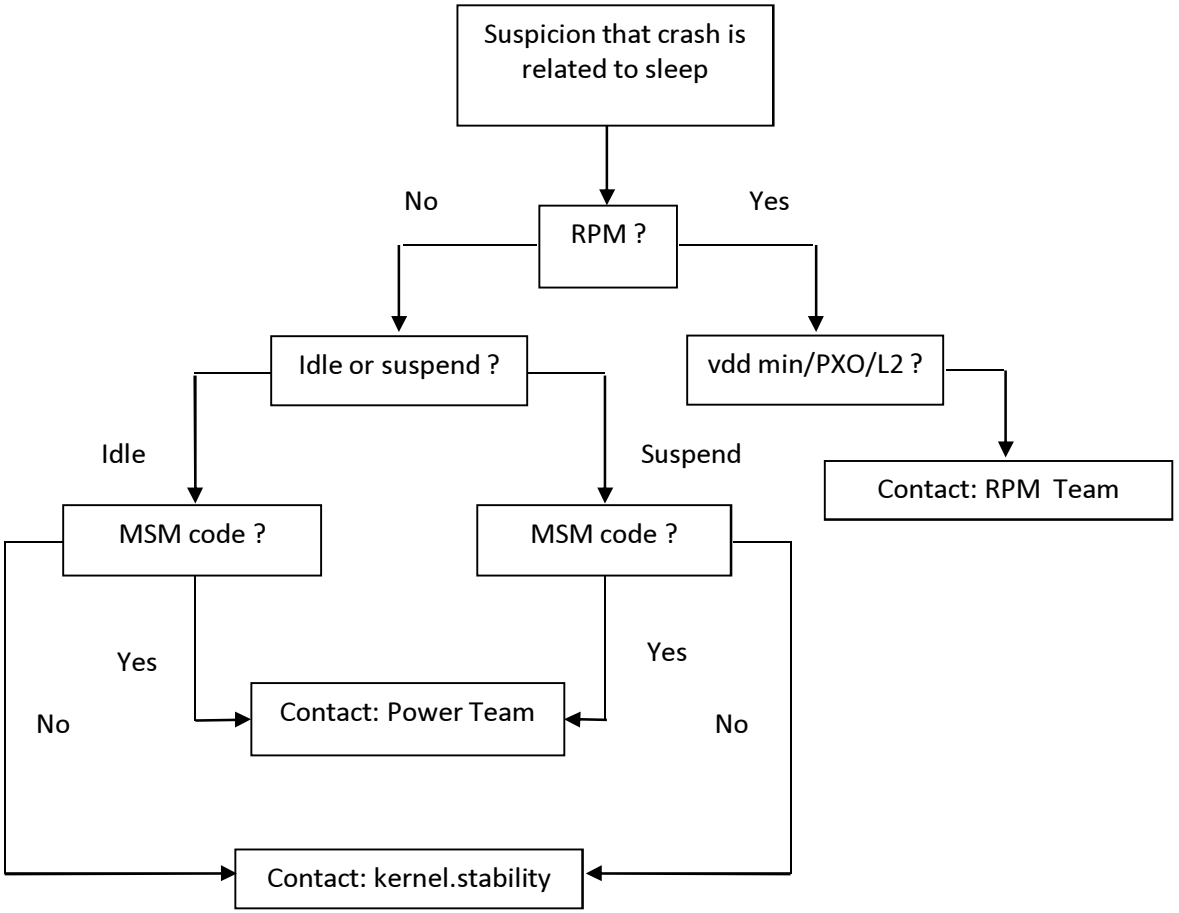
Reasons to believe crash maybe related to Sleep

- Issue occurs only when:
  - usb cable is not connected
  - screen is off
  - easy to reproduce when there is no activity on the phone
  - as an isolation tool when nothing else is obvious

- Of course no isolation/further triaging required if the point of failure is obvious from kernel logs, Jtag, etc

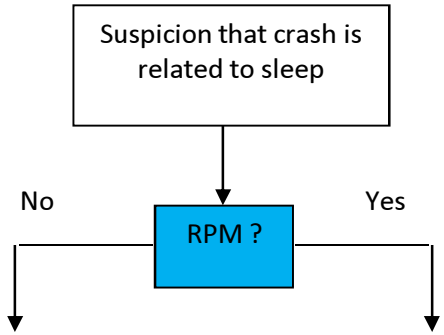
- Sleep = Power Collapse (we don't consider WFI here)

General Isolation Steps – 8660



Beware - Each issue is different and sometimes symptoms might be misleading

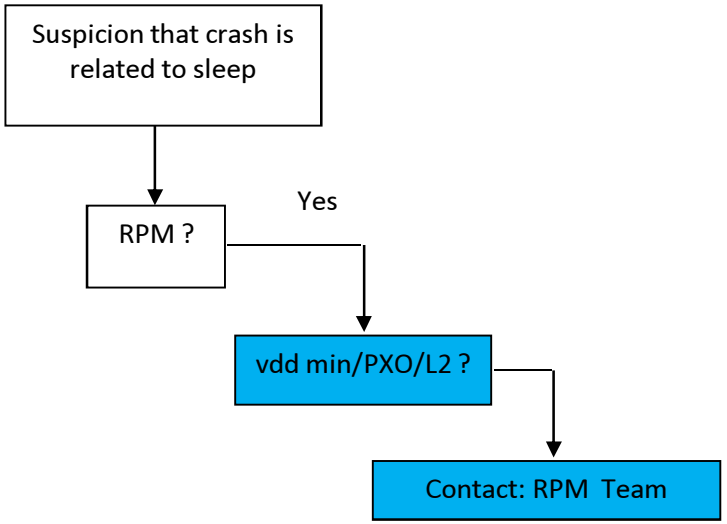
Rule out RPM



Rule out RPM

- Disable idle and suspend power-collapse with RPM notification but leave standalone idle and suspend power-collapses enabled
  - echo 0 > /sys/module/pm_8x60/modes/cpu0/power_collapse/idle_enabled
  - echo 0 > /sys/module/pm_8x60/modes/cpu1/power_collapse/idle_enabled
  - echo 0 > /sys/module/pm_8x60/modes/cpu0/power_collapse/suspend_enabled
  - echo 0 > /sys/module/pm_8x60/modes/cpu1/power_collapse/suspend_enabled
- Disable standalone idle and suspend power-collapses but leave idle and standalone power-collapses with RPM notification enabled:
  - echo 0 > /sys/module/pm_8x60/modes/cpu0/standalone_power_collapse/idle_enabled
  - echo 0 > /sys/module/pm_8x60/modes/cpu1/standalone_power_collapse/idle_enabled
  - echo 0 > /sys/module/pm_8x60/modes/cpu0/standalone_power_collapse/suspend_enabled
  - echo 0 > /sys/module/pm_8x60/modes/cpu1/standalone_power_collapse/suspend_enabled
- If the issue goes away only when disabling power-collapses with RPM notification, then it might be related to RPM

If RPM, rule out vdd min/PXO/L2 ?



If RPM, rule out vdd min/PXO/L2 ?

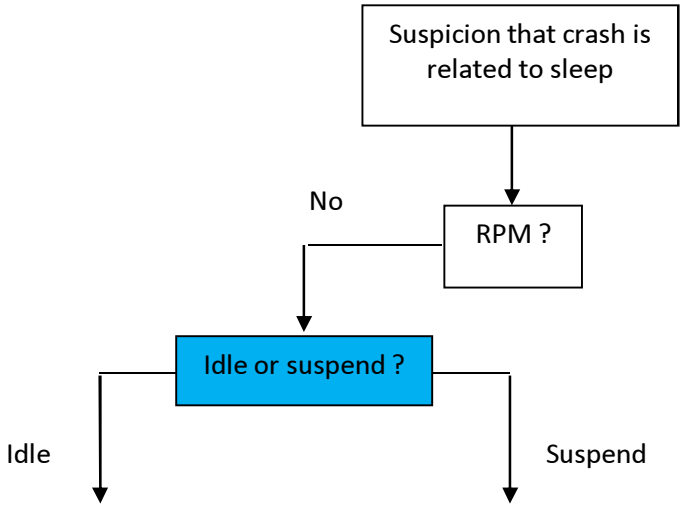
- If it seems to be related to RPM:

- Disable vdd min
- echo 0 > /sys/module/rpm_resources/enable_low_power/vdd_dig
- echo 0 > /sys/module/rpm_resources/enable_low_power/vdd_mem
- This makes sure apps votes for at least 1.0 volts (0.75 volts disabled)

- Disable low power mode for PXO and L2
- echo 0 > /sys/module/rpm_resources/enable_low_power/pxo
- echo 0 > /sys/module/rpm_resources/enable_low_power/L2_cache

- Turn on more apps debug for vdd min and pxo:
- echo 3 > /sys/module/rpm_resources/parameters/debug_mask
- echo 5 > /sys/module/pm_8x60/parameters/debug_mask

Is it specific to Idle or suspend power-collapse ?



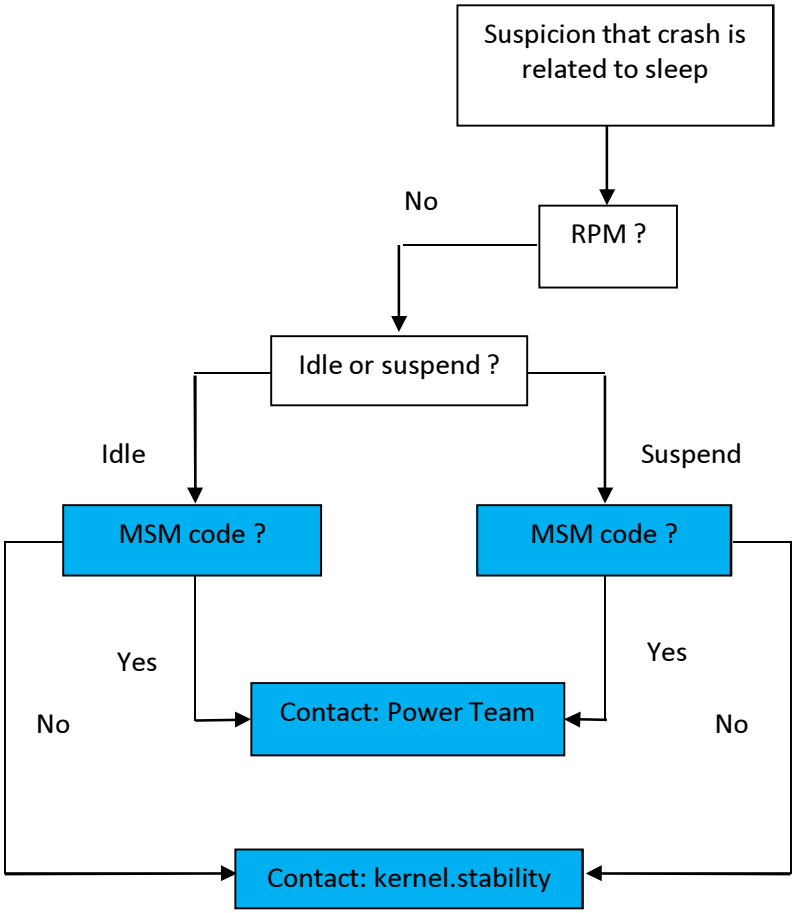
Is it specific to Idle or suspend power-collapse ?

- Disable suspend power-collapses but leave idle power collapses enabled:
- echo 0 > /sys/module/pm_8x60/modes/cpu0/standalone_power_collapse/suspend_enabled
- echo 0 > /sys/module/pm_8x60/modes/cpu1/standalone_power_collapse/suspend_enabled
- echo 0 > /sys/module/pm_8x60/modes/cpu0/power_collapse/suspend_enabled
- echo 0 > /sys/module/pm_8x60/modes/cpu1/power_collapse/suspend_enabled

- Disable idle power-collapses but leave suspend power-collapses enabled:
- echo 0 > /sys/module/pm_8x60/modes/cpu0/standalone_power_collapse/idle_enabled
- echo 0 > /sys/module/pm_8x60/modes/cpu1/standalone_power_collapse/idle_enabled
- echo 0 > /sys/module/pm_8x60/modes/cpu0/power_collapse/idle_enabled
- echo 0 > /sys/module/pm_8x60/modes/cpu1/power_collapse/idle_enabled

- If the issues goes away only when suspend power-collapse is disabled, then we might want to focus on suspend-resume triaging

Is the issue in MSM code ?



Is the issue in MSM code ?

- Enable MSM debug messages
- echo 511 > /sys/module/pm_8x60/parameters/debug_mask

- Ability to selectively enable debug messages. Check enum in arch/arm/mach-msm/pm-8x60.c file:

```
enum {
        MSM_PM_DEBUG_SUSPEND = BIT(0),
        MSM_PM_DEBUG_POWER_COLLAPSE = BIT(1),
        MSM_PM_DEBUG_SUSPEND_LIMITS = BIT(2),
        MSM_PM_DEBUG_CLOCK = BIT(3),
        MSM_PM_DEBUG_RESET_VECTOR = BIT(4),
        MSM_PM_DEBUG_IDLE = BIT(6),
        MSM_PM_DEBUG_IDLE_LIMITS = BIT(7),
        MSM_PM_DEBUG_HOTPLUG = BIT(8),
};
```

Other useful Resources

- Display who holds the wakelocks
- cat /proc/wakelocks

- Enable all wakelock debug messages
- echo 15 > /sys/module/wakelock/parameters/debug_mask

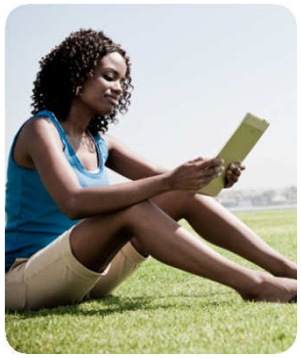
- Display statistics on WFI and Power-Collapses
- cat /proc/msm_pm_stats

- Interrupt that woke us up
- echo 1 > /sys/module/msm_show_resume_irq/parameters/debug_mask

- Remove core 1:
- echo 0 > /sys/devices/system/cpu/cpu1/online

- Wiki with all the power related sysfs knobs
- go/8660-pc-faq

Watchdog Debugging



QuIC
QUALCOMM INNOVATION CENTER, INC.

The MSM Watchdogs

What is a watchdog?

- A hardware timer – counts to X cycles. If it's not reset within that time, triggers an interrupt or some other hardware signal (usually a reset).
- Not a new concept.
- http://www.ganssle.com/watchdogs.pdf - How a watchdog could have saved a spacecraft and other interesting notes.
- Go/blackbird (find the hdd)

What?- Bark and Bite

- Bark
  - A warning that a bite is about to happen. You should NOT pet the watchdog when a bark is received – bad design! Usually configured as an interrupt – the corresponding handler should try to record debugging info.
- Bite
  - Usually configured to cause a reset. Good design says reset system, don't guess/make assumptions of wha went wrong.

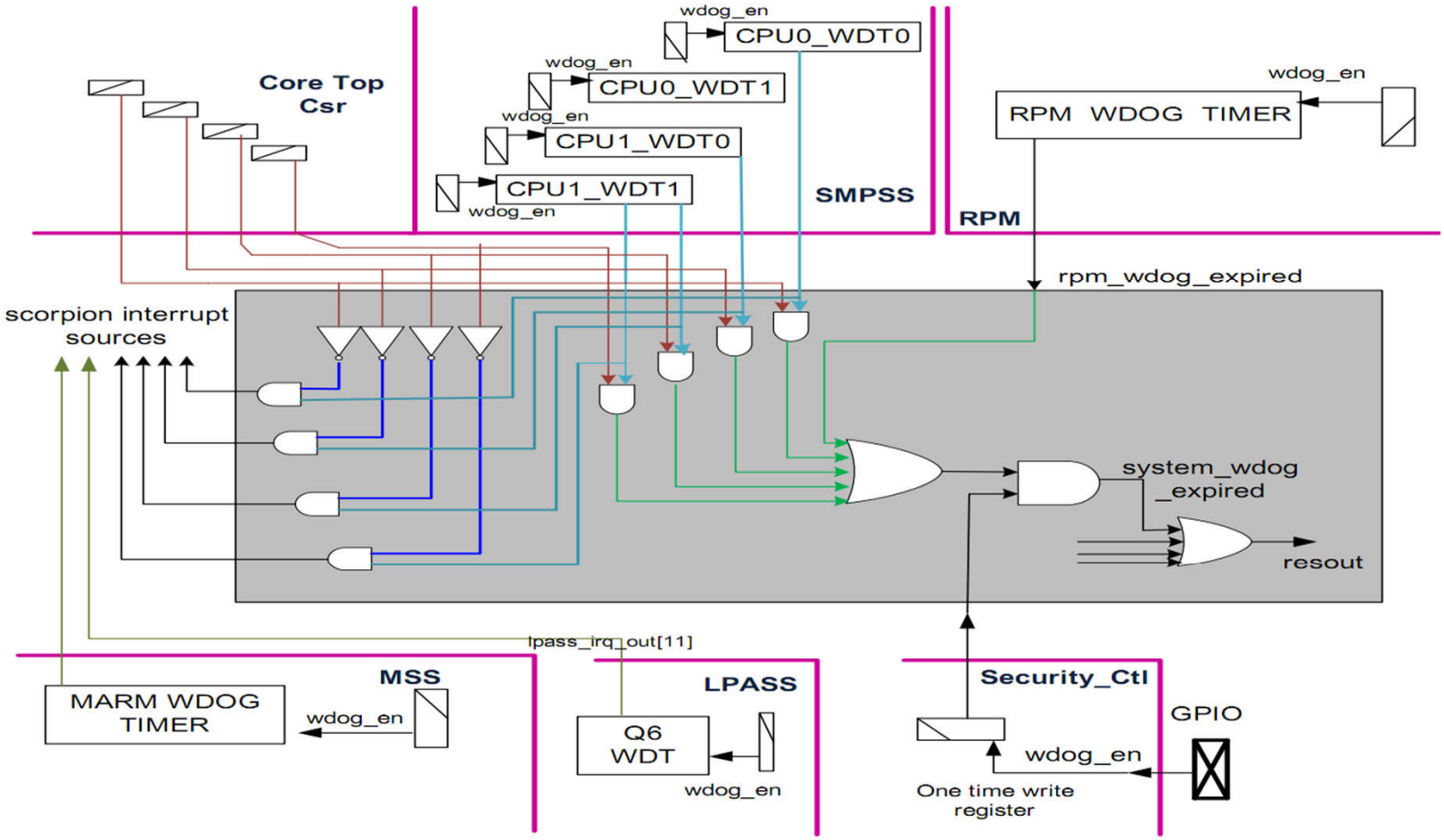
Why?

- For internal testing, primarily to catch lockup/stall conditions that software cannot.
  - Do a complete hardware reset, leaving debugging breadcrumbs behind if possible.
- For production phones, mainly to enhance user experience by randomly resetting the user's phone without warning.
  - Arguably better than having the user reset the device themselves.

Blackbird Watchdogs

- Scorpions
  - Two hardware watchdog timers per core
  - Currently, only one watchdog on Core-0 is used.
- Modem/Q6/DSPS/PMIC
  - Watchdog bite interrupt lines going from each to the Scorpions – used in subsystem restart
- RPM
  - One watchdog timer, causes system reset.

Blackbird Wdogs system level impl.



Scorpion watchdog

- From here on, will refer to the non-secure watchdog timer on core ‘0’ as ‘The Watchdog’.
  - Because it’s the only one that’s being used and is ‘pet’ by the kernel.

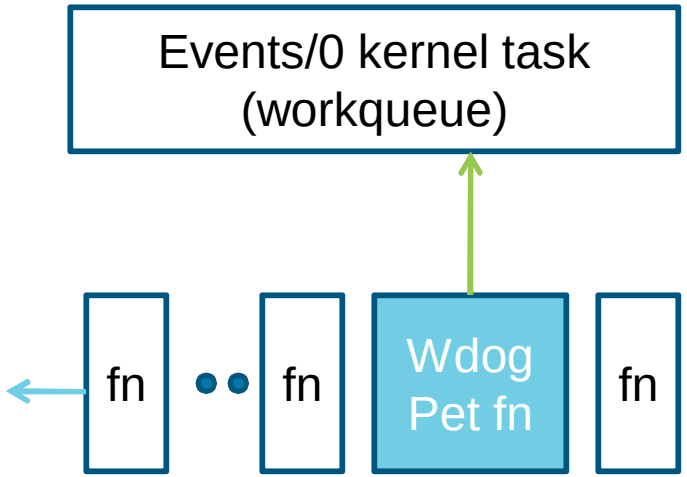
- Two signals, bark and bite – Bark generates an interrupt that can be configured as an FIQ/IRQ. Bite is a reset signal.

Scorpion watchdog part 2

- Bark
  - Configured as either an IRQ handled by the kernel (which means you're assuming interrupts are enabled at the time of the lockup/stall)
  - Or as an FIQ – handled by TZ – no assumption about IRQ state.
  - msm_watchdog.appsbark=0/1 decides which.
- Bite
  - Drives a reset signal (Resout), which will only happen if a bit in a secure register is enabled. (**TCSR_SMPSS_WDOG_CFG)**

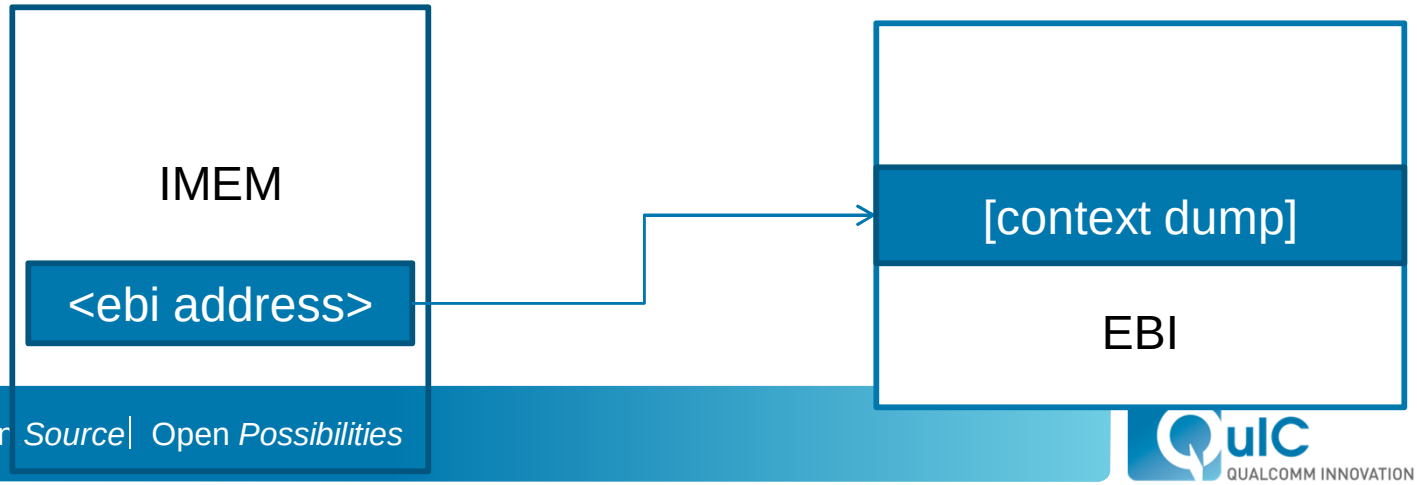
Kernel design

- Petting the watchdog (Resetting the timer)
  - Performed in a workqueue function
  - A workqueue is basically a process in which you can assign functions to be run one by one in a FIFO manner.
  - The kernel watchdog pet function is run inside the shared system workqueue on core 0.
  - **A function must complete before the next one can be run inside a workqueue!**



Scorpion watchdog bark

- Bark
  - Kernel dumps process list, current process backtrace in the bark handler, OR
  - TZ dumps current scorpion context into EBI, provides a pointer to that dump in IMEM.



How do I know an 8660 watchdog fired?

- Scorpions
  - Bark: Either a panic in the kernel log with the message “"Apps watchdog bark received!”
    or based on the output of the wdog-get-regs tool
  - Bite: No indication, deducible via elimination.
- RPM
  - Bark: RPM Error fatal in logs
  - Bite: No indication, deducible via elimination
- Peripherals – Modem/Q6 etc.
  - Bite: Kernel Panic with a “Watchdog bite from X” message.

Debugging a scorpion wdog event

- Debugging starts with a process of elimination.
  - Check kernel logs, if there's a bark panic you've got a bark. Ensure that it was the **first** panic.
  - Run the wdog-get-reg tool
    - If the tool shows a valid scorpion context dump pointer in IMEM, you've got yourself a bark.
  - Get someone to check RPM logs just in case!

Bark Debugging

- Check list of process stacks, generally safe to ignore those with schedule() as their most recent function call.
- Focus on stacks with preempt_schedule – this means something had to be forcefully pre-empted – didn't call a sleeping API.
- Focus on the events/0 backtrace (in 2.6.35 kernels) – If there's a schedule_timeout or preempt_schedule, see the stack and contact PoC.
- Focus on mutex_lock calls
- Focus on softirq threads & threaded ISRs

Bark Debugging- Example

- WLAN code calling into lower layer networking code which waits for too long for a mutex:
  - <6>[57.682708] PID: 7, Name: events/0
  - <4>[57.682723] [<c05aa300>] (schedule+0x434/0x528)
  - <4>[57.682739] [<c05aaccc>] (__mutex_lock_slowpath+0x164/0x1e0)
  - <4>[57.682760] [<c05aad68>] (mutex_lock+0x20/0x3c)
  - <4>[57.682782] [<c046f98c>] (linkwatch_event+0x8/0x34)
  - <4>[57.682799] [<c00d4464>] (worker_thread+0x15c/0x1e8)
  - <4>[57.682813] [<c00d7fd0>] (kthread+0x78/0x80)

Bark Debugging - Example 2

<6>[1887.998703] PID: 7, Name: events/0
<4>[1888.003646] [<c056815c>] (schedule+0x434/0x528)
<4>[1888.012583] [<c05683f0>] (preempt_schedule_irq+0x4c/0x70)
<4>[1888.021530] [<c003fcec>] (svc_preempt+0x8/0x1c)
<4>[1888.030814] [<c00a19c0>] (msm_bus_fabric_rpm_commit+0x74/0x184)
<4>[1888.040976] [<c00a0940>] (msm_bus_commit_fn+0x18/0x1c)
<4>[1888.049998] [<c02d9ef8>] (bus_for_each_dev+0x48/0x84)
<4>[1888.060942] [<c00a12cc>] (msm_bus_scale_client_update_request+0x1dc/0x20c)
<4>[1888.071875] [<c02cf1e8>] (kgsl_pwrctrl_axi+0x58/0xc8)
<4>[1888.081421] [<c02cf4d4>] (kgsl_pwrctrl_sleep+0xfc/0x154)
<4>[1888.090795] [<c02cf558>] (kgsl_idle_check+0x2c/0x5c)
<4>[1888.099736] [<c00ca124>] (worker_thread+0x15c/0x1e8)
<4>[1888.107983] [<c00cdc90>] (kthread+0x78/0x80)

Talking to the RPM may take a long time!

Bark Debugging - Example 3

- Interrupt overrun
  - CORE 0 PC: read_current_timer_delay_loop <0xc029a7d8+0x1c>
  - CORE 0 LR: read_current_timer <0xc00527a4+0x14>
  -
  - Arm unwind stack:
  - [<C029A780>] __delay+0x10
  - [<C03E856C>] i2c_ssbi_pa_read_bytes+0x8c
  - [<C03E7FCC>] i2c_ssbi_transfer+0xa4
  - [<C03E3B44>] i2c_transfer+0x9c
  - [<C033C1FC>] pm8901_isr_thread+0x290
  - [<C01082F0>] irq_thread+0x1b8
  - [<C00E1044>] kthread+0x7c
  - [<C00425EC>] kernel_thread_exit+0x0

- In this case, spurious PMIC interrupts caused an overrun, scheduling the ISR thread above way too often – preventing the kernel workqueue from being scheduled enough to pet the wdog.

Bite (no bark) Debugging

- Wouldn't wish this on my enemies
  - Easily one of the more complex things to debug as many of you already know
- Causes:
  - Bus lockups – which means a bus transaction was stalled, eg. Accessing DDR memory when it's in self refresh. Also eg. Registers need clocks, accessing one when it's clock is turned off.
    - http://www.ee.ic.ac.uk/pcheung/teaching/ee1_digital/Lecture11-Counters.pdf
  - Other conditions?

Bite (no bark) Debugging I

- Disable apps watchdog, reproduce problem
  - How? A few ways:
    - Command-line parameter “msm_watchdog.enable=0”
    - Adb command:
      echo 1 > /sys/module/msm_watchdog/parameters/runtime_disable
    - Hardware dip-switch on FFA.
  - Try to get scorpion context (snoop.pc in T32)
  - Try to get ETM trace – everyone should know this!
- Disable watchdogs one by one – RPM, apps are the only two to try.

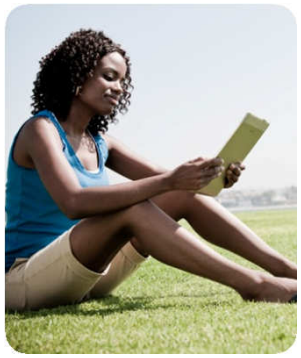
Bite (no bark) Debugging II

- Check which T32 sessions are attachable.
- If RPM T32 is attachable/breakable - see what memories you can access with RPM:
  - D.dump <address>
    - EBI (0x40000000 – 0x7FFFFFFF)
    - SMI (0x38000000 – 0x3BFFFFFF)

Bite (no bark) Debugging
Talk to CoreBSP/VI/HW teams!

- Ask CoreBSP folks to check reset status if possible – confirm that it's actually a watchdog initiated reset!
- Find out about the testclock.cmm script – dump the state of all clocks.
- Find out about hwioreg.per script – run on RPM to get DDR/SMI/other state
- Get everyone (preferably except _me_) involved – this is serious!

Reset Debugging



QuIC
QUALCOMM INNOVATION CENTER, INC.

Sources of reset

- RESIN_N: Normal reset driven by PM8901
- WDOG_EXPIRED: Watchdog Reset
- SRST_N: JTAG reset
- MPM_PU_RESET (warmboot) – coming out of power collapse sleep.

- Others: Check the Blackbird HDD (go/blackbird)

What is ‘Download Mode’?

- A device resets (think of it as a power-off/power-on) and the bootloaders (SBL3) check certain locations in IMEM. If those locations contain certain magic numbers, the normal bootup sequence is abandoned, and the scorpion sits in a loop in SBL3, with USB client to PC host active (QPST!)
- So download mode is actually an interrupted bootup sequence!
- Why download mode? To get ram dumps.
- SBL3 lives in the 0x47XXXXXX address range in EBI.

When to go into download mode?

- Abnormal, fatal conditions
  - Kernel panic – the flags in IMEM are set, and the PS_HOLD is driven low to reset the system – if this fails the watchdog is reset, the petting sequence stopped to reset the system.
  - Watchdog timeout
    - Bark
      - » Kernel Handled: Resets the system by calling panic.
      - » TZ handled: Resets the system using watchdog, download mode imem flags should have been previously set?
    - Bite
      - » Is itself effectively a reset signal, imem flags should have been previously set.

How to get into dload mode?

- Write those magic nos. in IMEM
  - General idea is to write the magic nos. on bootup, and clear them if it is known that a reset is normal and expected.
  - Used to be controlled by NV items, is now handled in the kernel.
  - Reset_detection flag (on by default)
    - /sys/modules/restart/parameters/reset_detection

Debugging

- Logically, only 2 things can cause a reset:
  - Watchdog – already covered.
  - Kernel panic – already covered!
- Disabling resets:
  - Disable the watchdog (covered)
  - Disable reset-on-panic
    - echoing 0 to /proc/sys/kernel/panic on the target at runtime
    - adding panic=0 to the kernel command line
    - going into the kernel configuration (make kernelconfig), general setup submenu, and setting default panic timeout to 0.

Ok, disabled reset – now what?

- Kernel panic
  - Figure out why it panic'ed. Attach T32 and debug.
    - Check for modem/q6 watchdog bite messages.
- Watchdog
  - Covered.

- T32 – Keep the JTAG connected, but may not want to attach in t32 during the test, may affect the test config itself

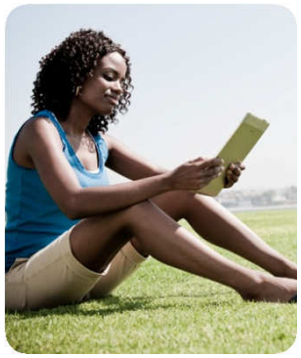
Reset reason

- There is a RESET_STATUS register that tells you what caused the reset – PMIC/Wdog etc. IF this needs to be checked, get CoreBSP help to do so.
- There are also restart reason codes that the kernel writes in IMEM. For example, adb shell reboot bootloader causes a specific code to be written.

Tips..

- If someone shows you an attached T32 session and says they suspect a reset, you should know where each software component lies in memory (bootloader, sbls, kernel, TZ) so that you can tell where the scorpion is executing.
- Reset and shutdown sequences should be studied in the HDD
- If a device is in download mode, it definitely **reset!**

Fusion Debugging

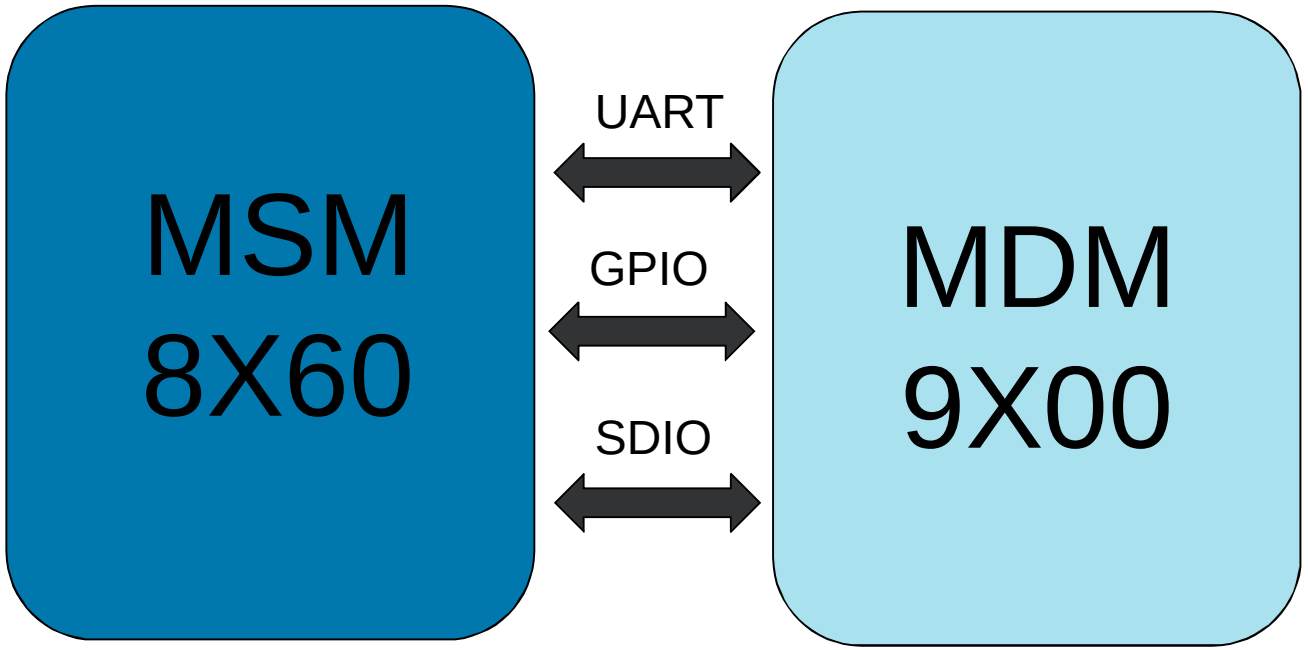


QuIC
QUALCOMM INNOVATION CENTER, INC.

Outline

- Brief architecture overview
- Booting up
- Crashes

What is Fusion?



- No shared memory
- Each runs independently with own RAM
- 9k has no NAND, depends on 8k to load images on bootup

The 9k flashless boot process - userspace

- Userspace program called kickstart responsible for loading images
  - POC for kickstart is Aseem Brahma (abrahma)
- Kickstart brings the 9k out of reset via ioctl call to kernel
- Kickstart loads first set of bootloaders via UART
  - “Dload completed successfully”
  - May see several failures before dload completes – that’s okay!
- Kickstart loads the rest of the images via SDIO
  - “Sahara transfer completed successfully”
- If RAM dumps are collected because of 9k crash, the target will load via UART (dload), collect RAM dumps and then reboot
- **Please contact Aseem for any questions or messages coming from kickstart**

The 9k flashless boot process - kernel

- <6>[2.403992] charm_modem_probe: Registering charm modem
  - If you don't see this you are probably not booting a charm target

- <6>[19.925072] sdio_al: sdio_dld_print_info, FLASHLESS BOOT - DURATION IN MSE C = 4680
  <6>[19.931690] sdio_al: sdio_dld_print_info, FLASHLESS BOOT - BYTES WRITTEN O N SDIO BUS = 24431480...BYTES SENT BY TTY = 24431480
  - This indicates that flashless boot should have completed.

The 9k never came up!

- Check the kickstart logs
  - Did dload ever complete successfully?
  - Did sahara ever complete successfully?
  - Contact Aseem for anything suspicious
- Check the kernel logs
  - grep for ‘charm’
    - any suspicious message contact Laura Abbott (lauraa)
  - grep for ‘sdio_al’
    - any suspicious messages contact Maya Erez or Erez Tsidon

errfatal and status change

- Two GPIOS from 9k to scorpion are used to indicate error conditions
- MDM2AP_ERRFATAL
  - Exactly like it sounds, a fatal condition
  - Message of “Reseting the charm due to an errfatal” in the kernel log
- MDM2AP_STATUS
  - Some other software couldn’t handle the fatal condition
  - Message of “Reseting the charm because status changed” in the kernel log
- What happens on these GPIO will vary with subsystem restart
  - Currently the behavior is to panic. Will be changing in the future
- **If you see any message indicating a status or errfatal GPIO change please contact the 9k triage team for further assistance**

KERNEL DEBUG OPTIONS

QuIC
QUALCOMM INNOVATION CENTER, INC.

Outline

- Overview of options
- List of options
  - CONFIG_DEBUG_LIST
  - CONFIG_PAGE_POISON
  - CONFIG_SLUB_DEBUG_ON
  - user_debug=31
  - initcall_debug
  - CONFIG_DEBUG_SPINLOCK
  - CONFIG_DEBUG_SG
  - DEBUG_SPINLOCK_SLEEP

Every developer has to debug

- Lots of developers face the same problems
- Linux has many built in options to make catching these problems easier
- bigger list of debug options can be found in lib/Kconfig.debug

CONFIG_DEBUG_LIST

- What this does
  - Adds in checks to standard linked list manipulation functions (list_add, list_del)
  - If the pointers aren't consistent (next and prev don't match) a warning will be printed out
- What it's good for
  - finding if a linked list is getting corrupted
- Example

```
WARN(next->prev != prev,
        "list_add corruption. next->prev should be "
        "prev (%p), but was %p. (next=%p).\n",
        prev, next->prev, next);
```

CONFIG_PAGE_POISON

- What this does
  - When unmaping pages in the kernel, fills the memory with 0xAA
- What it's good for
  - Finding out if someone is accessing memory after it is mapped
- Example

<4>[17.061183] PC is at memcpy+0x50/0x330
<4>[17.061191] LR is at 0xaaaaaaaa
<4>[17.061200] pc : [<c03328f0>] lr : [<aaaaaaaa>] psr: 20000113
<4>[17.061205] sp : d8e77edc ip : aaaaaaaa fp : 000020a1
<4>[17.061213] r10: 00003fff r9 : 00000000 r8 : aaaaaaaa
<4>[17.061221] r7 : aaaaaaaa r6 : aaaaaaaa r5 : aaaaaaaa r4 : aaaaaaaa
<4>[17.061230] r3 : aaaaaaaa r2 : 00000660 r1 : d9188020 r0 : 00000000

If you are seeing 0xaa appear in your registers, you have probably hit this case

CONFIG_SLUB_DEBUG_ON

- What this does
  - Lots!
  - Adds extra checks to memory allocation functions
  - If you go outside your memory in certain circumstances it will get very very noisy
- What it's good for
  - Finding use after free bugs
    - If you see 0x6b everywhere with this patch you have a use after free bug
  - Finding use uninitialized bugs
    - If you see 0x5a everywhere with this patch you have a use uninitialized bug
  - Writing off the end of allocated memory
  - Double freeing
  - Freeing a bogus pointer

CONFIG_SLUB_DEBUG_ON

- What this isn't good for
  - Writing off into non adjacent addresses
  - Caching issues
  - Timing issues (may get lucky)
- Example

```
<3>[  390.633333] =============================================
<3>[  390.640485] BUG kmalloc-96: Poison overwritten
<3>[  390.644904] ---------------------------------------------------------------------
<3>[  390.644911]
<3>[  390.654552] INFO: 0xd127f7b0-0xd127f7b0. First byte 0x6a instead of 0x6b
<3>[  390.661243] INFO: Allocated in msm_control+0x2c/0x4bc age=1 cpu=0 pid=127
<3>[  390.668000] INFO: Freed in msm_get_stats+0x7ec/0x948 age=3 cpu=0 pid=3648
<3>[  390.674777] INFO: Slab 0xc12921dc objects=25 used=9 fp=0xd127f780 flags=0x00c3
<3>[  390.681970] INFO: Object 0xd127f780 @offset=1920 fp=0xd127f1e0
```

CONFIG_SLUB_DEBUG_ON

- Another example

<1>[10987.232222] Unable to handle kernel paging request at virtual address 6b6b6b57

<1>[10987.239310] pgd = deb70000

<1>[10987.241524] [6b6b6b57] *pgd=00000000

<4>[10987.265378] PC is at find_vma+0x40/0x7c

<4>[10987.269189] LR is at find_extend_vma+0x14/0x70

<4>[10987.273617] pc : [<c01ef2d4>] lr : [<c01eff90>] psr: 20000013

<4>[10987.273622] sp : d5aa9da8 ip : 6b6b6b4f fp : 00000000

<4>[10987.285072] r10: d5aa9ea8 r9 : d5aa9dec r8 : d5aa9e8c

<4>[10987.290281] r7 : 00000001 r6 : 42234000 r5 : 42234000 r4 : 6b6b6b6b

<4>[10987.296793] r3 : df1b0960 r2 : 6b6b6b6b r1 : 42234000 r0 : daf2d220

The 0x6b6b6b6b marked the memory as free but someone still held a reference to it!

boot with user_debug=31

- What this does
  - If a userspace application dies due to a SIGSEGV or a SIGBUS a message will be logged in the kernel
- What it's good for
  - Helping to correlate a userspace crash with a kernel crash
- Example

```
if (user_debug & UDBG_SEGV) {
        printk(KERN_DEBUG "%s: unhandled page fault (%d) at 0x%08lx, code 0x%03x\n",
            tsk->comm, sig, addr, fsr);
        show_pte(tsk->mm, addr);
        show_regs(regs);
    }
```

This message will be appended with __do_user_fault

initcall_debug

- What this does
  - echo 1 > /sys/modules/kernel/initcall_debug
  - initcall debugging
  - Adds timing information to driver power management code
  - Shows how long suspend resume functions are taking
- What it’s good for
  - Debugging suspend resume timeouts

- WARNING
  - This option can be very noisy
  - Disable watchdog before enabling this option

CONFIG_DEBUG_SPINLOCK / CONFIG_DEBUG_MUTEX

- What this does
  - Adds checks to spinlocks and mutexes
- What it's good for
  - Making sure mutexes / spinlocks are initialized before use
    - "It crashed in mutex lock?"
  - Checks for proper use

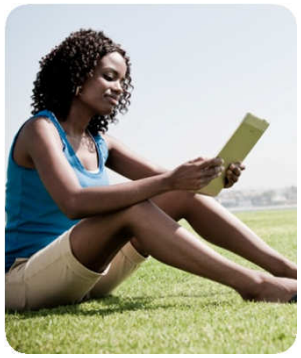
CONFIG_DEBUG_SG

- What this does
  - Adds checks to scatter gather lists
- What it's good for
  - Helping to find bugs in drivers that use scatter gather lists
  - If the a driver uses DMA, they might use scatter gather lists
    - Ask a tech team if they use sglists

DEBUG_SPINLOCK_SLEEP

- What this does
  - "If you say Y here, various routines which may sleep will become very noisy if they are called with a spinlock held.“
- What it’s good for
  - Finding delays
  - Bad use of spinlocks

GPIO



QuIC
QUALCOMM INNOVATION CENTER, INC.

GPIO

- Errors – kernel/include/asm-generic/errno-base.h
- Msmgpio
  - Gpiomux – arch/arm/mach-msm/gpiomux-8x60.c
- Expander Gpios – board file
- I2C – Gpio Errors
- go/gpiostuff

GPIO

- Errors – kernel/include/asm-generic/errno-base.h

- <7>[2.166627] gpio_request: gpio-308 (ov9726) status -22

GPIO

- Errors – kernel/include/asm-generic/errno-base.h
- Msmgpio
  - Gpiomux – arch/arm/mach-msm/gpiomux-8x60.c
- Expander Gpios – board file
- I2C – Gpio Errors
- go/gpiostuff

GPIO

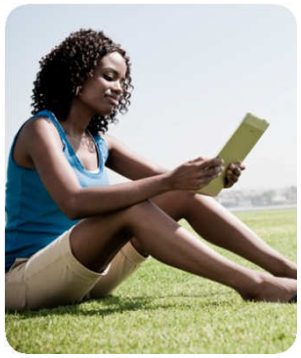
- I2C – Gpio Errors

- <3>[1.996736] qup_i2c qup_i2c.1: QUP: I2C status flags :0x1343c8, irq:185
- <3>[2.002323] qup_i2c qup_i2c.1: I2C slave addr:0x40 not connected
- <3>[2.009866] msm_sync_init: failed to initialize qs_s5k4e1
- <4>[2.014233] msm_camera_qs_s5k4e1: probe of msm_camera_qs_s5k4e1.0 failed with error -5
- <3>[2.055685] qup_i2c qup_i2c.1: QUP: I2C status flags :0x1343c8, irq:185
- <3>[2.061275] qup_i2c qup_i2c.1: I2C slave addr:0x1a not connected
- <3>[2.068868] msm_sync_init: failed to initialize imx074
- <4>[2.072973] msm_camera_imx074: probe of msm_camera_imx074.0 failed with error -5
- <7>[2.166627] gpio_request: gpio-308 (ov9726) status -22

GPIO

- Errors – kernel/include/asm-generic/errno-base.h
- Msmgpio
  - Gpiomux – arch/arm/mach-msm/gpiomux-8x60.c
- Expander Gpios – board file
- I2C – Gpio Errors
- go/gpiostuff

SUBSYSTEM RESTART



QuIC
QUALCOMM INNOVATION CENTER, INC.

Outline

- Subsystem restart
  - What it is
  - What it looks like

The idea behind subsystem restart

- Qualcomm chips have many different subsystems
  - a couple of modems, LPASS etc.
- A crash on one of them generally brings down the entire system
  - This means a reboot of the entire phone
  - rebooting is slow
  - Other subsystems are probably just fine
- Better idea: log the failure and restart the failing subsystem!
  - What this entails depends on each subsystem

Subsystem restart phases

- Phase 1: Subsystem restart code is in place but reboot the whole SOC
  - This is what is in place now

<0>[30.653902] Kernel panic - not syncing: Resetting the SOC
<4>[30.658373] [<c0107a84>] (unwind_backtrace+0x0/0x164) from [<c068bc50>] (panic+0x6c/0xe8)
<4>[30.666568] [<c068bc50>] (panic+0x6c/0xe8) from [<c01677d8>] (subsystem_restart+0x1b0/0x1e8)
<4>[30.674963] [<c01677d8>] (***subsystem_restart***+0x1b0/0x1e8) from [<c01a1ee4>] (worker_thread+0x15c/0x1e8)
<4>[30.684238] [<c01a1ee4>] (worker_thread+0x15c/0x1e8) from [<c01a5a50>] (kthread+0x78/0x80)
<4>[30.692616] [<c01a5a50>] (kthread+0x78/0x80) from [<c010238c>] (kernel_thread_exit+0x0/0x8)
<0>[30.700773] Rebooting in 5 seconds..

  - Look at the line above to see what subsystem crashed
    - external modem = 9k (contact 9k triage team)
    - modem = 8k (contact 8k triage team)
    - lpass = qdsp (contact Hiren Bhagatwala)

Subsystem restart phases

- Phase 2: Restart individual subsystems together
  - Restart modems together
  - Currently in the testing phase
  - Crash will be logged in kernel message
    - <6>[30.636114] Subsystem Restart: Restart sequence requested for external_modem
- Phase 3: Restart each subsystem completely independently
  - Sometime in the future
  - Logging message will still be the same as above

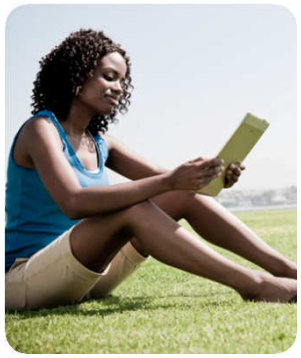
- All phases will involve RAM dump collection
- **For any subsystem crash, please contact the triage team for that particular subsystem**

Q&A

Feedback

QuIC
QUALCOMM INNOVATION CENTER, INC.

Thank You



QuIC
QUALCOMM INNOVATION CENTER, INC.