
Introduction to Qualcomm Specific Debugging Features



Qualcomm Technologies, Inc.

80-P7139-3 A

Confidential and Proprietary – Qualcomm Technologies, Inc.

Restricted Distribution: Not to be distributed to anyone who is not an employee of either Qualcomm Technologies, Inc. or its affiliated companies without the express approval of Qualcomm Configuration Management.



Confidential and Proprietary – Qualcomm Technologies, Inc.

QUALCOMM
2017-08-07 18:31:51 PDT
lilubao@gionee.com

NO PUBLIC DISCLOSURE PERMITTED: Please report postings of this document on public servers or websites to: DocCtrlAgent@qualcomm.com.

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies, Inc.

Qualcomm is a trademark of Qualcomm Incorporated, registered in the United States and other countries. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer (“export”) laws. Diversion contrary to U.S. and international law is strictly prohibited.

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

© 2016 Qualcomm Technologies, Inc. and/or its affiliated companies. All rights reserved.

Revision History

Revision	Date	Description
A	July 2016	Initial release

QUALCOMM
2017-08-07 18:31:51 PDT
lilubao@gionee.com

Contents

- Introduction
- RTB Log
- IPC Log
- TZBSP Diag Buffer
- Qualcomm Access Control (QuAC)
- NOC Error
- Non-Secure Dog Bite
- Secure Dog Bite
- TZ Counters
- CPU State Machine and TZ counters
- How to Check TZ Counters
- RPM Log
- References
- Questions?

Introduction

- This presentation provides an overview of Qualcomm-specific debugging features, and is intended to be used by software engineers who are debugging stability issues.

QUALCOMM
2017-08-07 18:31:51 PDT
lilubao@gionee.com

RTB Log

- RTB log is used to log different events to a small non-cached region, and aids to debug reset issues where caches may not be properly flushed before the target resets
- Enable RTB log
 - CONFIG_MSM_RTB=y
 - CONFIG_MSM_RTB_SEPARATE_CPUS=y
 - Set msm_rtb.enabled and msm_rtb.filter in BOARD_KERNEL_CMDLINE, or change their settings in /sys/module/msm_rtb/parameters when runtime.
- Select RTB log filters
 - See following slides; also see msm_rtb.h to know how bits are mapped to events
- Get RTB logs
 - QCAP reports , or Linux Ramdump parser reports

RTB Log (cont.)

- What can be seen from RTB log
 - Timestamp is aligned to dmesg log
 - I/O memory access: LOGK_READL/WRITEL
 - Function calls into I/O memory access primitives defined in arch/arm64/include/asm/io.h
 - Context switching: LOGK_CTXID
 - Record logs before actual thread switching in switch_to()
 - IRQ happens: LOGK_IRQ
 - Record HWIRQ number read from GIC IAR
 - CPU hotplug: LOGK_HOTPLUG
 - Records logs when
 - CPU_STARTING (CPU will be running soon)
 - CPU_DYING (CPU will not be running any tasks or handling any interrupts, and will soon die)
 - L2 register read/write: LOGK_L2CPWRITE/LOGK_L2CPREAD
 - Records the register address to be accessed.

IPC Log

- IPC log consists of MSM-specific kernel log buffers, which are initially used for interprocessor communications (IPC) modules, and applied to some other modules, like PCIe
 - A similar buffer as kernel printk; it has a head structure for each record
- Enable IPC log
 - CONFIG_IPC_LOGGING
- Get IPC log
 - QCAP reports
 - Type the following command in adb shell:
 - `cat /d/ipc_logging/*/log_con &`

IPC Log (cont.)

- What can be seen from IPC log
 - Timestamp is aligned to dmesg log/ RTB log
 - General logs from following drivers, like smd/smем/smsm/glink/rpm_smd/sps

```
IPC
... smem
... smd
... smsm
... glink
... dsps_smd_trans
... lpass_smd_trans
... mpss_smd_trans
... wcnss_smd_trans
... rpm_smd_trans
... sps_ipc_log0
... sps_ipc_log1
... sps_ipc_log2
... sps_ipc_log3
... sps_ipc_log4
... mpss_smем
... lpass_smем
... dsps_smем
... rpm_smем
... sps_bam_0x0000000000684000_0
... sps_bam_0x0000000000684000_1
... sps_bam_0x0000000000684000_2
... sps_bam_0x0000000000684000_3
... sps_bam_0x0000000000684000_4
... pcie0-short
... pcie0-long
... pcie0-dump
... pcie1-short
... pcie1-long
... pcie1-dump
... pcie2-short
... pcie2-long
... pcie2-dump
... glink_ssr
... glink_lbsrv
... smp2p
```

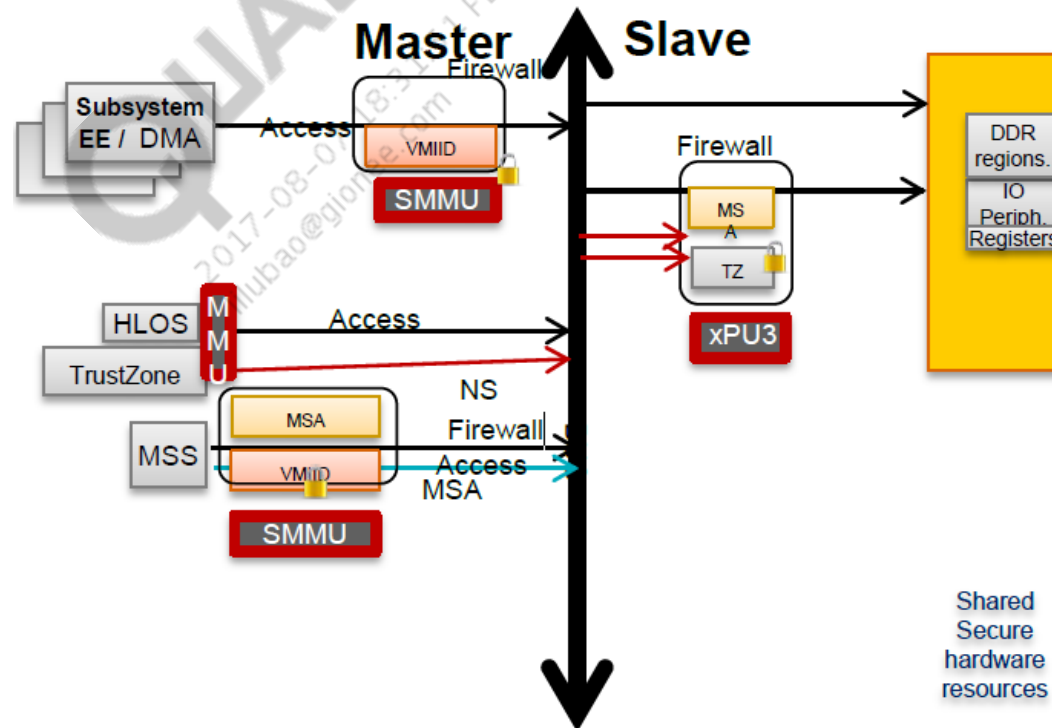
```
kqmi_req_resp
kqmi_ind
glink_pkt
7570000.uart
sps_bam_0x00000000007544000_0
sps_bam_0x00000000007544000_1
sps_bam_0x00000000007544000_2
sps_bam_0x00000000007544000_3
sps_bam_0x00000000007544000_4
smd_tty
smd_pkt
diag
sps_bam_0x0000000000644000_0
sps_bam_0x0000000000644000_1
sps_bam_0x0000000000644000_2
sps_bam_0x0000000000644000_3
sps_bam_0x0000000000644000_4
devfreq_spdm
local_IPCRTR
sps_bam_0x00000000006b04000_0
sps_bam_0x00000000006b04000_1
sps_bam_0x00000000006b04000_2
sps_bam_0x00000000006b04000_3
sps_bam_0x00000000006b04000_4
91c0000.slim
lpass_IPCRTR
sps_bam_0x00000000009184000_0
sps_bam_0x00000000009184000_1
sps_bam_0x00000000009184000_2
sps_bam_0x00000000009184000_3
sps_bam_0x00000000009184000_4
dsps_IPCRTR
sps_bam_0x00000000007584000_0
sps_bam_0x00000000007584000_1
sps_bam_0x00000000007584000_2
sps_bam_0x00000000007584000_3
sps_bam_0x00000000007584000_4
mpss_IPCRTR
```

TZBSP Diag Buffer

- TZBSP diag buffer is a circular buffer saved in IMEM, logged by TZBSP
- Enable TZBSP
 - Enabled by default
- Get TZBSP diag log
 - QCAP report -- From crash dump
 - Cat /d/tzdbg/log -- During runtime
- Source code from both TZ side and HLOS side
- What can be seen from TZBSP diag buffer
 - SMMU failure, XPU violation (QuAC, Qualcomm Access Control)
 - NOC error
 - Non-secure dog bite
 - Secure dog triggered from non-secure world (from Hypervisor)

Qualcomm Access Control (QuAC)

- Inside the SoC, if one master wants to access certain slaves
 - SMMU unit checks the access permission for any memory access
 - xPUs are present at the slave sides; also controls the access using VMID generated by the master



NOC Error

- Inside the SoC, if one master was trying to access non-clock registers, or accessing DDR that had been in a bad or self-refresh state, an NOC error would be generated (see following example)

TZ diag log:

Target = 8996v3

NoC = SNOC

ERRLOG0 = 0x80030308

ERRLOG1 = 0x84054001

ERRLOG2 = 0x0

ERRLOG3 = 0x3d0040

ERRLOG4 = 0x0

Decoded logs :

opcode = 0x4 = Write

errcode = **0x3 = Unclocked access**

InitFlow = 0x4 = qxm_aggr0_noc

TargFlow = 0x4 = qhs0_hmss_0

BID = 0x5

PID = 0x2

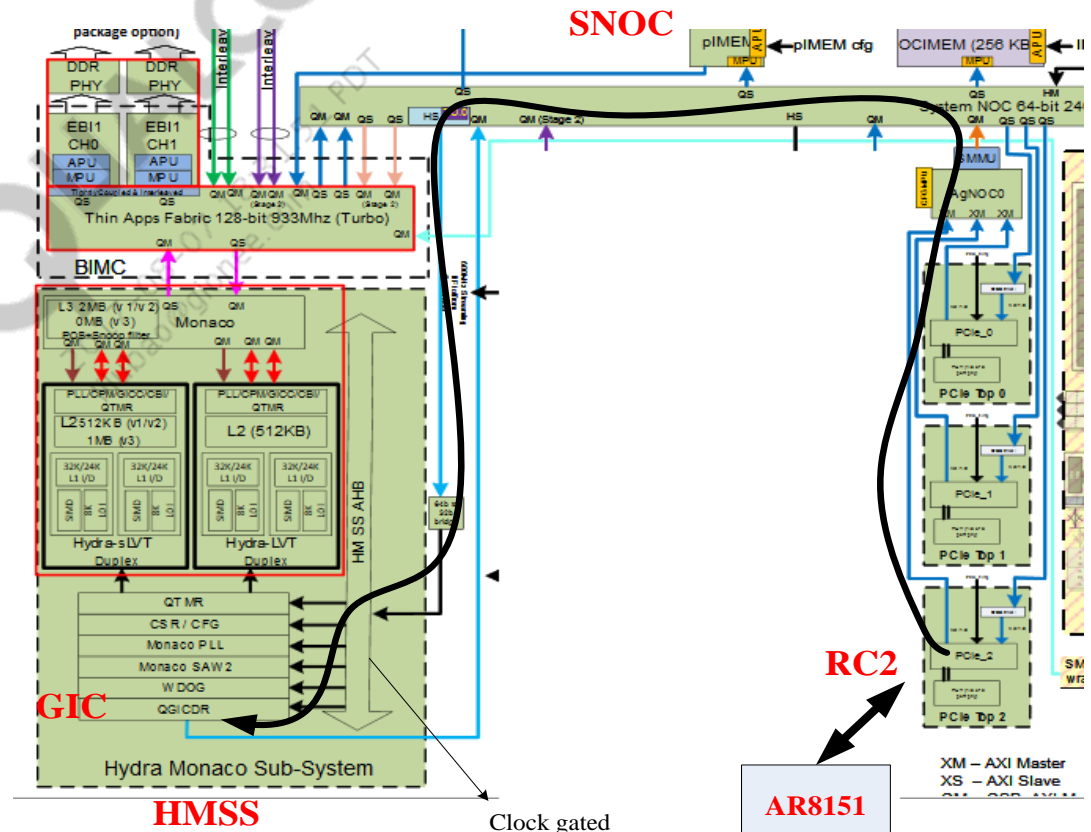
MID = 0x0

BID/PID/MID = ANOC0 PCIE 2

Address offset = 0x003d0040

Address base = 0x09800000

Address = **0x09bd0040**



Non-Secure Dog Bite

- Case #1: Linux kernel ought to pet dog periodically, but if it fails, dog bark (IRQ) occurs; if kernel is able to handle the IRQ, kernel intentionally triggers non-secure dog bite
 - Kernel prints when last pet happens and current time
 - Pet/bark time is configurable in device tree
- Case #2: If kernel is not able to handle dog bark IRQ in time, dog would automatically bite (FIQ); TZ would service it and save the context
 - TZ log says non-secure dog bite happens

Note: All non-secure dog bites would finally trigger secure dog bite, because always relying on secure dog bite to reset chip

Secure Dog Bite

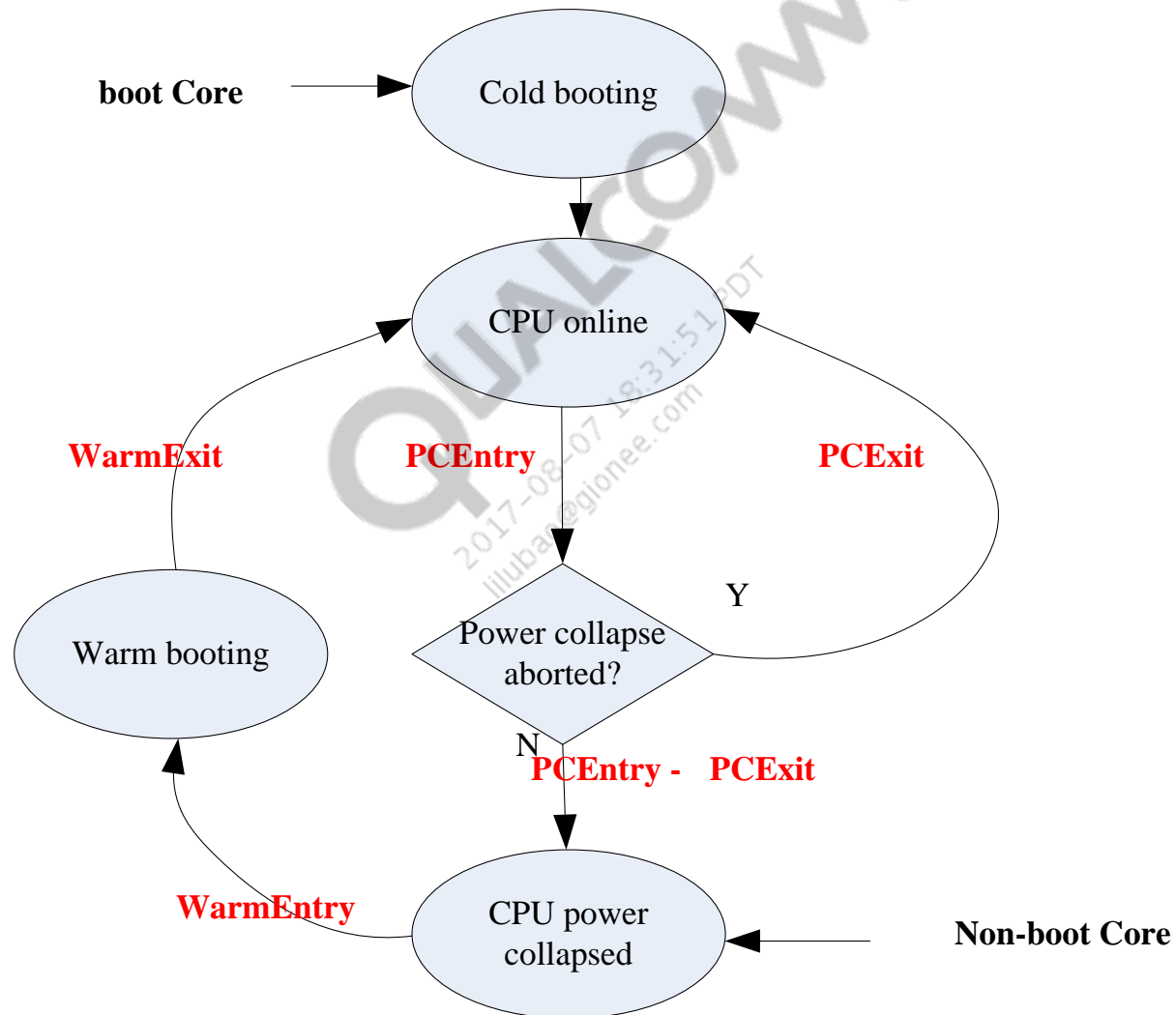
- Case #3: If a Hypervisor runs into fatal errors, it can make SMC call and trigger secure dog bite
 - Hyp log identifies the fatal error type
 - TZ log states that secure dog triggered from non-secure world
- Case #4: After entering TZ, TZ pets secure dog when it barks, but if TZ fails to do so, secure dog biting finally happens and chip resets
 - There is no useful log from TZ diag log
 - Time when bark/bite happens for secure dog is also configurable
 - See Oem_config.xml: OEM_sec_wdog_bark_time, OEM_sec_wdog_bite_time.

TZ Counters

- TZ counters are saved in IMEM, and increased when cores entering and exiting power collapse
- TZ counters are in QCAP logs

Boot Status / TZ Counter								
=====								
CPU	WarmEntry	WarmExit	PCEnter	PCExit	Warm JumpAddr	JumpInstr	PSCIEntry	PSCIExit
0	0x00B7257C	0x00B7257C	0x00CC43C6	0x00151E4A	0x0000000000000000	0x00000000	0x00CD59B2	0x00163436
1	0x0098D42D	0x0098D42D	0x00A7AC4C	0x000ED820	0x0000000000000000	0x00000000	0x00A84FCD	0x000F7BA1
2	0x00868808	0x00868808	0x00931EDA	0x000C96D2	0x0000000000000000	0x00000000	0x00937E06	0x000CF5FE
3	0x007C500F	0x007C500F	0x0086DFCC	0x000A8FBD	0x0000000000000000	0x00000000	0x00872884	0x000AD875
4	0x0037E9CC	0x0037E9CC	0x0086C318	0x004ED94D	0x0000000000000000	0x00000000	0x00877C4A	0x004F927F
5	0x00034AC9	0x00034AC9	0x00041C34	0x0000D16B	0x0000000000000000	0x00000000	0x0004307A	0x0000E5B1
6	0x000046A0	0x000046A0	0x0000529D	0x00000BFD	0x0000000000000000	0x00000000	0x0000534B	0x00000CAB
7	0x00003AF4	0x00003AF4	0x000044D9	0x000009E5	0x0000000000000000	0x00000000	0x0000455D	0x00000A69

CPU State Machine and TZ counters



How to Check TZ Counters

- What can be learned from TZ counters
 - Core may be stuck in warm boot
 - If WarmEntry = WarmExit counter, then it enters and exits TZ the same number of times; if there is a mismatch, the core is still in TZ
 - Core may be stuck in power collapse
 - For boot core , if $PCEnter - (PCExit + WarmEntry) = 0$, then the core is online; if $PCEnter - (PCExit + WarmEntry) = 1$, core 0 is power collapsed
 - For non-boot core , if $PCEnter - (PCExit + (WarmEntry - 1)) = 0$, the core is online; [For non-boot cores, their first powering up is implemented as a warm boot]

boot core (logical core0)	warmEntry = WarmExit	$PCEnter - (PCExit + WarmEntry) = 0$	Not stuck in warm boot, core online
		$PCEnter - (PCExit + WarmEntry) = 1$	Not stuck in warm boot, core power collapsed
	warmEntry = WarmExit+1	no need to check	stuck in warm boot
non-boot core	WarmEntry = WarmExit	$PCEnter - (PCExit + (WarmEntry - 1)) = 0$	Not stuck in warm boot, core online
		$PCEnter - (PCExit + (WarmEntry - 1)) = 1$	Not stuck in warm boot, core power collapsed
	warmEntry = WarmExit+1	no need to check	stuck in warm boot

RPM Log

- RPM log is a circular buffer saved in DataRAM, logged by RPM subsystem in ULog format
- Get RPM log
 - QCAP report -- From crash dump
 - Hansei

```
rpm_proc\core\bsp\rpm\scripts\hansei\
```

 - See 80-NA157-9 for usage
- What can be seen from RPM log
 - RPM fatal error, caused by APPS NON SECURE WD BITE
 - RPM fatal error, caused by other reasons
 - If device is entering vdd-min or exiting vdd-min when crash happens

Note: To know whether AP cores are offline, RPM dump needs to be loaded via T32 simulator

```
rpm.ees[master_id].subsystem_status
```

References

Title	Number
Qualcomm Technologies, Inc.	
<i>RPM Debug Overview</i>	80-NA157-9

Acronym or term	Definition
ITC	interprocessor communications
NOC	Network On a Chip
RTB	Register Trace Buffer
RPM	Remote Power Manager
SoC	System on Chip
TZBSP	TrustZone Boot Services Platform

QUALCOMM®
2017-08-07 18:31:51 PDT
lilubao@gionee.com

Questions?

<https://createpoint.qti.qualcomm.com>

