QUALCOMM® CDMA Technologies

REDEFINING MOBILITY

MSM8960™ RPM Power Debugging Details

80-N8454-3 A

Qualcomm Confidential and Proprietary

Restricted Distribution. Not to be distributed to anyone who is not an employee of either Qualcomm or a subsidiary of Qualcomm without the express approval of Qualcomm's Configuration Management.

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

Qualcomm reserves the right to make changes to the product(s) or information contained herein without notice. No liability is assumed for any damages arising directly or indirectly by their use or application. The information provided in this document is provided on an "as is" basis.

This document contains Qualcomm confidential and proprietary information and must be shredded when discarded.

QUALCOMM is a registered trademark of QUALCOMM Incorporated in the United States and may be registered in other countries. Other product and brand names may be trademarks or registered trademarks of their respective owners. CDMA2000 is a registered certification mark of the Telecommunications Industry Association, used under license. ARM is a registered trademark of ARM Limited. QDSP is a registered trademark of QUALCOMM Incorporated in the United States and other countries.

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 Incorporated
5775 Morehouse Drive
San Diego, CA 92121-1714
U.S.A.

Revision History

Version	Date	Description
A	Nov 2011	Initial release

Contents

- RPM debugging concepts
  - RPM Halt and the JTAG daisy chain
  - Finding the RPM build version used
  - Saving RPM dumps
  - Loading RPM dumps onto TRACE32 (T32) and extracting logs
  - Key locations storing system state
  - Configuration set inheritance
  - Aggregation across masters
  - Time for full example – Modem vdd_min voting debug example
- RPM logs for debugging
  - RPM external log
  - RPM NPA log/dump
  - Putting it all together – vdd_min issue debug steps
  - How to check power collapse and vdd_min happening and attach the stack

RPM Debugging Concepts

REDEFINING MOBILITY

RPM Halt and the JTAG Daisy Chain

- Legacy ARM® cores (ARM7™, ARM11™) resynchronize the JTAG RTCK signal to their respective processor clocks.
- The result is that halting the RPM clock, e.g., during sleep, causes a “no RTCK” condition on the JTAG daisy chain.
- Therefore, if reliable JTAG is required for any master, the RPM cannot halt, as the RPM is on the daisy chain.
- Several methods of working around this are built into the system.
  - In a command window connect to the Linux side and enter:
    - echo 0 > /sys/module/rpm_resources/enable_low_power/rpm_cpu
  - Create a /nv/items_files/sleep/core0/sleep_config.ini file.
    - A file must be placed at this location (using EFS Explorer)
    - The contents of the file (this will disable XO shutdown and Vdd min) are:
      - [rpm.handshake]
      - disable=1
  - In sleep_os.c, the variable “be_gentle_to_the_daisy_chain” exists that will disable all forms of RPM sleep and allow unimpeded JTAG usage.
  - The various master processors can be configured to disallow RPM sleep via the disabling of the dynamic sleep LRPM named rpm.hs.

Finding the RPM Build Version Used

- Before debugging anything interesting with the RPM in T32, the respective RPM symbols must be loaded. This can be accomplished by checking which RPM version is used in the build. Two methods to find this information are:
  - First method
    - Copy the <Modem_build>\modem_proc\core\bsp\rpm\RPM.bin to C:\RPM
    - Copy which_rpm.py to C:\RPM
    - Open Cygwin and cd c:\RPM, issue the command “python which_rpm.py RPM.bin”
    - This will return the version number for the RPM.
  - Second method
    - Open ARM7 and attach to the running target by sys.m.a
    - d.in 0x00104008/LONG

    In a build this prints, e.g.,
    - SD:00104008 = 00000027
    - Which is in hex, so we interpret 0x27 = 39. Load symbols from the respective RPM reference build, as: d.load.elf <RPM.00.00.39>\build\rpm\8660\build\RPM.elf /nocode

Saving RPM Dumps

- RPM dumps can be saved in the following manner:
  1. Create the issue scenario where RPM dumps are needed
  2. Open ARM7 T32 and attach (sys.m.a)
  3. Break T32 and do the following for saving the RPM Memory Dump
     - d.save.binary C:\Temp\CODERAM.bin 0x20000++0x24000
     - d.save.binary C:\Temp\MSGRAM.bin 0x108000++0x5FFF

Loading RPM Dumps onto T32 and Extracting Logs

- To load RPM dumps onto the T32 simulator:
  1. Open the T32 simulator and do “sys.up”
  2. Jump to the dumps directory and load:
     - d.load.binary MSGRAM.bin 0x108000
     - d.load.binary CODERAM.bin 0x20000
  3. Find out the RPM build being used and load the respective RPM ELF (see previous slides)
  4. d.load.elf <RPM.00.00.xx>\build\rpm\8960\build\RPM.elf /nocode
- Extracting logs
  1. Do <RPM.00.00.xx>\core\power\ulog\scripts\ULogDump.cmm C:\Directory\
  2. Do <RPM.00.00.71>\core\power\npa\scripts\NPADump.cmm C:\Directory\
  3. Execute the python script for postprocessing:
     - python rpm_log.py -f "RPM External Log.ulog" -n "NPA Log.ulog”

Key Locations Storing System States

- Which version of RPM is loaded
  - d.in 0x00104008 /LONG
  - Example – 0x245→0x0.0x2.0x45→00.02.69
- When debugging RPM issues, it can be useful to see what request the RPM firmware driver has saved for a given master/set
- A large repository of useful information is accessible via the global variable gpRPMFWMaster
  - This is an array of DAL_rpmfw_MasterDataType and stores all of the states related to each master processor
  - Array indices are:
    - 0 – Apps processor
    - 1 – Modem processor
    - 2 – LPASS processor
    - 3 – RIVA processor
    - 4 – Sensors processor

gpRPMFWMaster Fields

```
typedef struct
{
  DAL_rpmfw_MasterType         master_id;           /* basic information
                                                       required at various points */
  DAL_rpm_ConfigSetType        selected_set;        /* configuration set DATA*/

  DAL_rpmfw_ConfigSetDataType  set[DAL_RPM_CONFIG_SET_COUNT];
  uint32                       defer_count;         /* defer engine data */
  uint32                       defer_actions;
  DALSYSEventHandle            defer_timer;
  DAL_rpm_ResourceIndType      notify_pending;      /* notification 'driver' data*/

  DALBOOL                      notify_in_queue;
  DAL_rpmfw_SPMStatusType      spm_subsystem_status; /* spm handshake data */
  uint32                       spm_num_active_cores;
  uint32                       spm_pending_bringups;
  DALSYSEventHandle            triggers_timer;      /* trigger 'driver' data */

  DALBOOL                      triggers_timer_expired;
  DAL_rpm_ResourceType resource;                    /* request handler data */
} DAL_rpmfw_MasterDataType;
```

gpRPMFWMaster Fields (cont.)

- Most interesting fields and their meanings
  - Selected_set
    - Indicates which configuration set the RPM is using for the master
  - Set
    - Indicates the request (if any) for each resource
    - Basically, an array indexed by enumeration values of type DAL_rpm_ResourceType, e.g., 6 is PXO
      - » data_valid indicates if any request at all has been made on that master/set pair
      - » Data is either the literal data (if data_size ≤ 4) or a pointer to the data buffer
      - » Data requests are inherited from the active set requests when data_valid = 0 for the sleep set and data_valid = 1 for the active set
    - Resource enumeration defined in core\api\power\RPMTypes.h

gpRPMFWMaster Fields (cont.)

- Most interesting fields and their meanings (cont.)
  - spm_subsystem_status
    - The status of the SPM for that master (going to sleep/sleeping/waking/awake)
  - spm_num_active_cores
    - For apps, how many cores are awake
    - For other processors, either 1 or 0, based on the single processor

gpRPMFWMaster Fields (cont.)

- Resource settings – Master sets

```
gpRPMFWMaster = (
    [0] = 0x00038548,
    [1] = 0x00039040 -> (
      master_id = DAL_RPMFW_MASTER_1 = 0x1,
      selected_set = DAL_RPM_CONFIG_SET_SLEEP = 0x1,
      set = (
       [0] = 0x00039080,
       [1] = 0x000395DC -> (
         resource = (
          [0] = (data_length = 0x40, data_valid = 0x0, data = 0x0003987C),
          [1] = (data_length = 0x0C, data_valid = 0x1, data = 0x000398BC),
          [2] = (data_length = 0x8, data_valid = 0x1, data = 0x000398C8),
          [3] = (data_length = 0x4, data_valid = 0x0, data = 0x0),
          [4] = (data_length = 0x0, data_valid = 0x0, data = 0x0),
          [5] = (data_length = 0x4, data_valid = 0x1, data = 0x0),
```

gpRPMFWMaster Fields (cont.)

- Resource settings – Master sets

```
gpRPMFWMaster = (
  [0] = 0x00039B4C -> (
    master_id = DAL_RPMFW_MASTER_0 = 0 = 0x0,
    …,
    spm_subsystem_status = DAL_RPMFW_SPM_AWAKE = 0 = 0x0,
    spm_num_active_cores = 2 = 0x2,
    spm_pending_bringups = 0 = 0x0,
    triggers_timer = 0x000381C4,
    triggers_timer_expired = 0 = 0x0),
  [1] = 0x0003A68C,
  [2] = 0x0003B1CC -> (
    master_id = DAL_RPMFW_MASTER_2 = 2 = 0x2,
    …,
    spm_subsystem_status = DAL_RPMFW_SPM_SLEEPING = 2 = 0x2,
    spm_num_active_cores = 0 = 0x0,
    spm_pending_bringups = 0 = 0x0,
    triggers_timer = 0x0003E364,
    triggers_timer_expired = 0 = 0x0))
```

gpRPMFWMaster Fields (cont.)

- Resource settings – Active set

```
gpRPMFWMaster = (
 [0] = 0x00039B4C -> (
 master_id = DAL_RPMFW_MASTER_0 = 0 = 0x0,
 selected_set =DAL_RPM_CONFIG_SET_PRIMARY=DAL_RPM_CONFIG_SET_ACTIVE_0
  =0x0,
   set = (
    [0] = 0x00039BD4 -> (
     resource = (
      [0] = (data_length = 64 = 0x40, data_valid = 0 = 0x0, data = 0x00039E74),
      [1] = (data_length = 12 = 0x0C, data_valid = 0 = 0x0, data = 0x00039EB4),
      [2] = (data_length = 8 = 0x8, data_valid = 0 = 0x0, data = 0x00039EC0),
      [3] = (data_length = 4 = 0x4, data_valid = 0 = 0x0, data = 0x0),
      [4] = (data_length = 0 = 0x0, data_valid = 0 = 0x0, data = 0x0),
      [5] = (data_length = 4 = 0x4, data_valid = 0 = 0x0, data = 0x0),
      [6] = (data_length = 4 = 0x4, data_valid = 1 = 0x1, data = 0x1),
      [7] = (data_length = 4 = 0x4, data_valid = 1 = 0x1, data = 0x0),
      [8] = (data_length = 4 = 0x4, data_valid = 1 = 0x1, data = 0x0001944C),
       …
```

gpRPMFWMaster Fields (cont.)

- Resource settings – Sleep set

```
gpRPMFWMaster = (
    [0] = 0x00039B4C -> (
      master_id = DAL_RPMFW_MASTER_0 = 0 = 0x0,
      selected_set = DAL_RPM_CONFIG_SET_SLEEP = 1 = 0x1,
      set = (
        [0] = 0x00039BD4,
        [1] = 0x0003A130 -> (resource = (
            [0] = (data_length = 64 = 0x40, data_valid = 0 = 0x0, data = 0x0003A3D0),
             …
            [4] = (data_length = 0 = 0x0, data_valid = 0 = 0x0, data = 0x0),
            [5] = (data_length = 4 = 0x4, data_valid = 0 = 0x1, data = 0x0),
            [6] = (data_length = 4 = 0x4, data_valid = 0 = 0x0, data = 0x0),
            [7] = (data_length = 4 = 0x4, data_valid = 0 = 0x0, data = 0x0),
          …
```

gpRPMFWMaster Fields (cont.)

- MSM8660™ resource specifics
  - CXO = index 5
  - PXO = index 6
  - Vdd Mem (Vdd Mx) = index 47 (0x2F)
  - Vdd Dig (Vdd Cx) = index 48 (0x30)

gpRPMFWMaster Fields (cont.)

- PMIC resources (voltage regulators) data is stored in a buffer, and buffer contents are the actual voltages being voted for.
- Because PMIC resources are not straightforward, it is good to know when checking the regulator voltages that their data has structure to it. T32 must be convinced to parse this structure, as in:

  v.v (pm_npa_vreg_smps_type*) gpRPMFWMaster[a].set[b].data[c]
- Typical buffer contents (actual voltages being voted for) are:
  - 0x44C => 1100 mV
  - 0x2EE => 750 mV
  - 0x1F4 => 500 mV

Configuration Set Inheritance

Selected Set

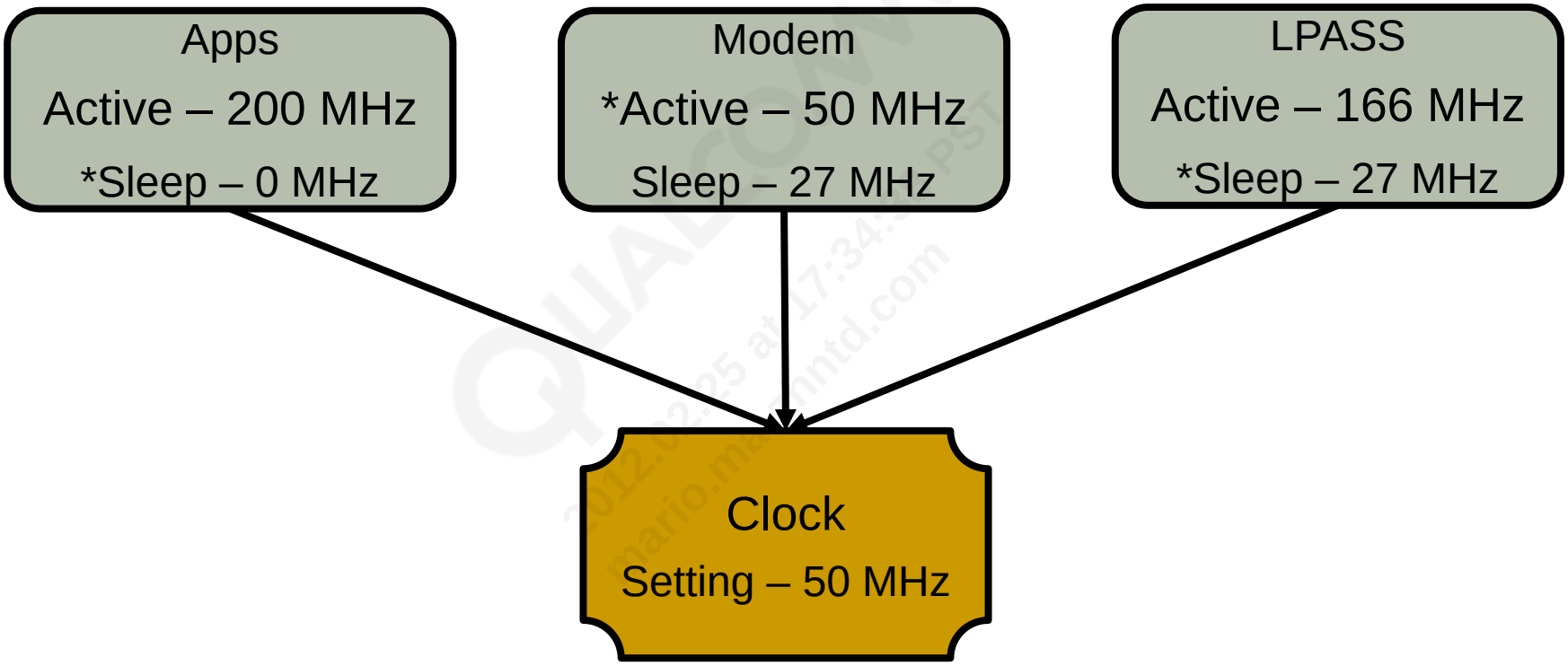
Resource	Active set	Sleep set	Effective request
Clock A	200 MHz	0 MHz	200 MHz
Regulator B	1100 mV	(No request)	1100 mV
Oscillator C	(No request)	(No request)	Driver default (generally “off”)

Configuration Set Inheritance (cont.)

Selected Set

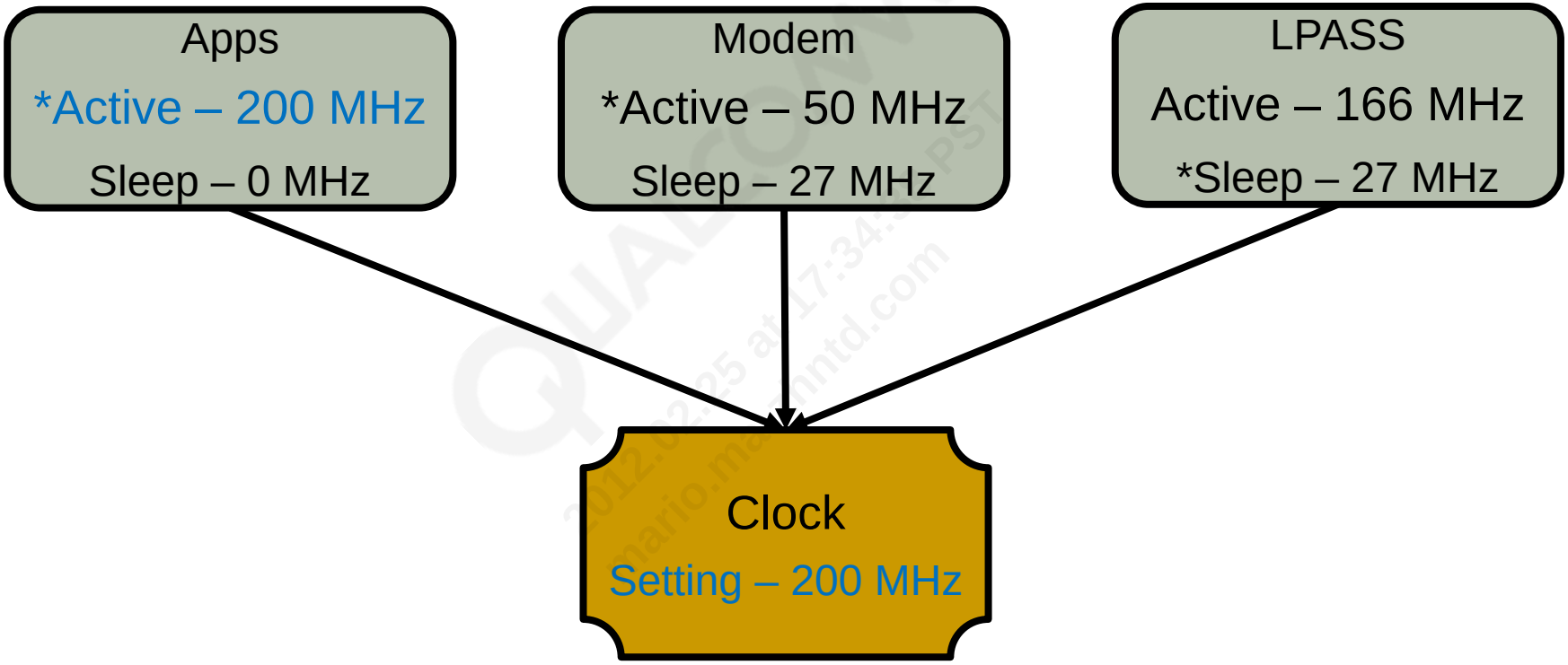
Resource	Active set	Sleep set	Effective request
Clock A	200 MHz	0 MHz	0 MHz
Regulator B	1100 mV	(No request)	1100 mV (inherited)
Oscillator C	(No request)	(No request)	Driver default (generally “off”)

Aggregation Across Masters



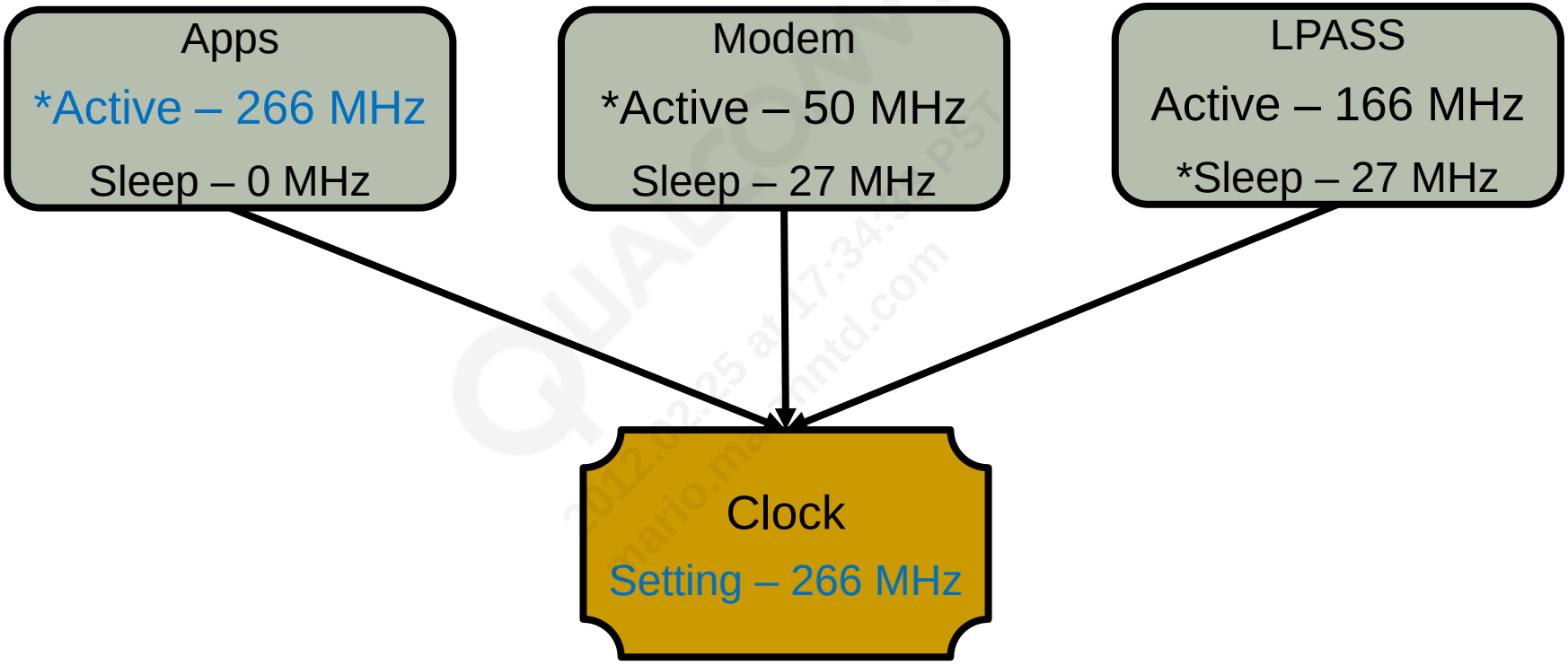
Note: (*) denotes selected set

Aggregation Across Masters (cont.)



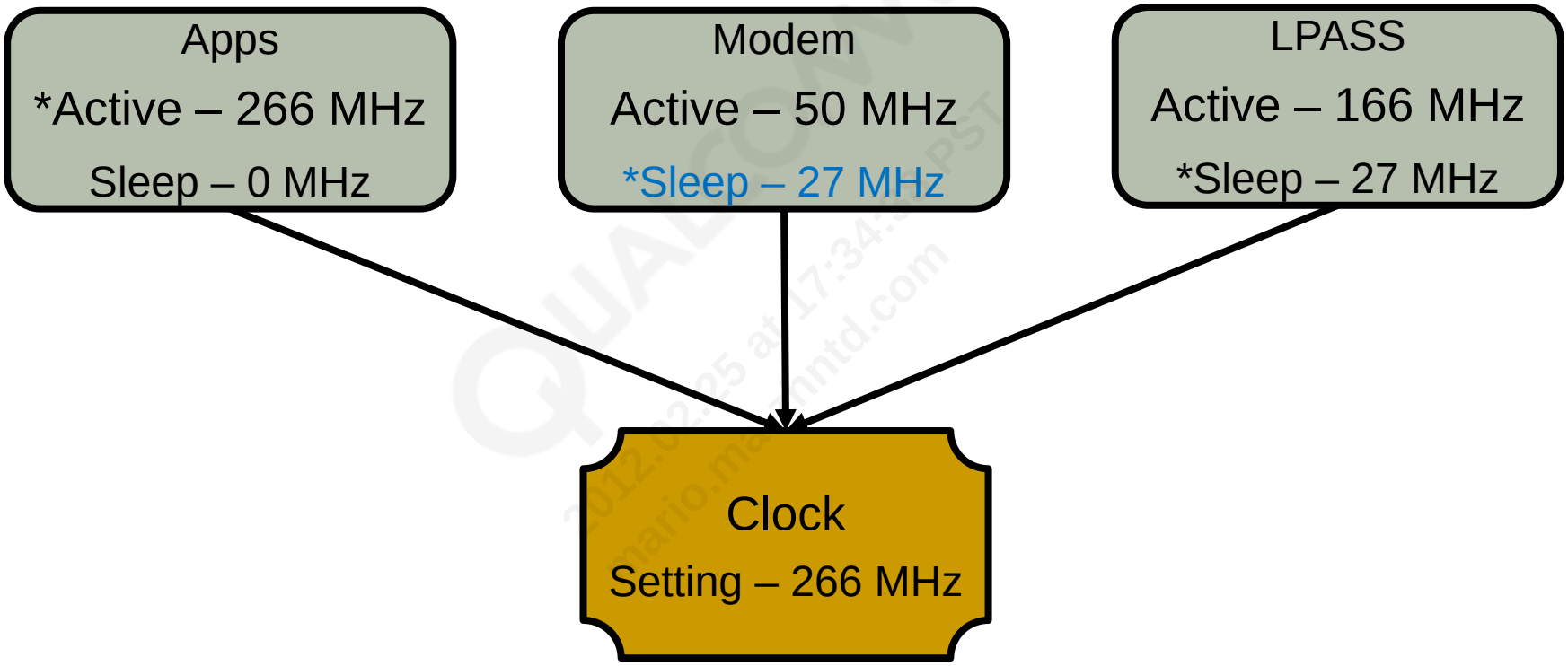
Note: (*) denotes selected set

Aggregation Across Masters (cont.)



Note: (*) denotes selected set

Aggregation Across Masters (cont.)



Note: (*) denotes selected set

Time for Full Example – Modem Vdd_min Voting Debug Example

- To find the voltage that the modem processor is voting for Vdd_Dig while it is asleep, check the variable in T32 by the following:

v.v (pm_npa_vreg_smps_type*) gpRPMFWMaster[1].set[1].data[0x30]
above
gpRPMFWMaster[1] : '1' references to Modem
set[1] : '1' references to Sleep Set
data[0x30] : '0x30' -> 48, references to "Vdd_Dig" resource

(pm_npa_vreg_smps_type*)gpRPMFWMaster[1].set[1].resource[0x30].data =
0x0003D8D8 -> (
mvol = 500 = 0x01F4,
ip = 0x0,
fm = 0x0,
pc = 0x0,
pf_low = 0x0,
pd = 0x0,
ia = 0x0,
freq = 0x0,
freq_clk_src = 0x0,
reserved = 0x0)

In this case, the modem is voting for Vdd_Dig minimization (500 mv).

RPM Logs for Debugging

REDEFINING MOBILITY

RPM External Log

- The RPM publishes a small log into a very limited area of spare message RAM.
- The physical format of the log is the ULog format used for various other logs.
  - It is a circular buffer, currently sized at around 4 kB.
  - It is a raw log, using a set of IDs and a variable number of parameters per message.

RPM External Log – Using T32

- Dumping the log
- NPA log and the RPM external log have been combined by following procedure. Benefit is that more NPA log entries appear, messages are properly interleaved and new high-resolution timestamp is available that can be used for better profiling.
- While attached to the RPM and in the Break state, run the following commands in T32:
  - Do \\<RPM build location>\core\power\ulog\scripts\ULogDump.cmm \\path\to\your\log\directory
  - Do \\<RPM build location>\core\power\npa\scripts\NPADump.cmm \\path\to\your\log\directory
- Run the following script for postprocessing:
  - Python rpm_log.py -f "RPM External Log.ulog" -n "NPA Log.ulog" > parsed_output.txt
  - rpm_log.py –h provides a list of supported options

RPM External Log (Alternate Way – Using ADB)

- The log can also be dumped during the operation by executing the following in the adb shell window while USB is attached:
  - adb shell mount -t debugfs none /sys/kernel/debug/
  - adb shell cat /sys/kernel/debug/rpm_log > <file>
- This log needs to be postprocessed similarly to the T32 logs.
- It is not advised to dump the log continuously but to use this for debugging issues when it is desired to capture the log without halting for T32 dumps.
  - Continuous dumping will affect performance.

RPM External Log – Analysis

- Message request contents
  - Timestamp, operation, data
    - 0x4b4eac: rpm_message_int_received (master: "MSS")
    - 0x4b4eae: rpm_servicing_master (master: "MSS")
    - 0x4b4eb2: rpm_driver_dispatch (resource: "Apps Fabric Clock")
    - 0x4b4eb5: rpm_driver_complete (rejected: 0)
    - 0x4b4eb6: rpm_driver_dispatch (resource: "System Fabric Clock")
    - 0x4b4ec1: rpm_driver_complete (rejected: 0)
    - 0x4b4ec3: rpm_driver_dispatch (resource: "EBI1_CLK")
    - 0x4b4ec8: rpm_driver_complete (rejected: 0)
    - 0x4b4ec9: rpm_sending_message_int (master: "MSS")

RPM External Log – Analysis (cont.)

- Message request contents – Entering Low Power mode

  0x4b4fb4: rpm_shutdown_req (master: "MSS") (core: 0)

  0x4b4fb5: rpm_shutdown_ack (master: "MSS") (core: 0)

  0x4b4fb8: rpm_master_set_transition (master: "MSS") (leaving: "Active Set") (entering: "Sleep Set")

  0x4b4ff9: rpm_status_updating (resource: "CXO_BUFFERS") (in_flux: 1)

  0x4b4ffb: rpm_status_updating (resource: "CXO_BUFFERS") (in_flux: 0)

  0x4b5000: rpm_status_updating (resource: "CXO") (in_flux: 1)

  0x4b5001: rpm_status_updating (resource: "CXO") (in_flux: 0)

  0x4b5006: rpm_status_updating (resource: "PXO") (in_flux: 1)

  0x4b5007: rpm_status_updating (resource: "PXO") (in_flux: 0)

  0x4b5009: rpm_status_updating (resource: "System Fabric Clock") (in_flux: 1)

  0x4b500b: rpm_status_updating (resource: "System Fabric Clock") (in_flux: 0)

  0x4b500c: rpm_status_updating (resource: "EBI1_CLK") (in_flux: 1)

  0x4b500e: rpm_status_updating (resource: "EBI1_CLK") (in_flux: 0)

  0x4b5017: rpm_vdd_min_enter (count: 11)

  ……. System is in Low Power Mode

RPM External Log – Analysis (cont.)

- Message request contents – Entering Low Power Mode (cont.)

  0x4beb5c: rpm_vdd_min_exit

  0x4beb5c: rpm_mpm_wakeup_ints (interrupts: "[timetick] | [modem bringup]")

  0x4beb5f: rpm_bringup_req (master: "MSS") (core: 0)

  0x4beb62: rpm_master_set_transition (master: "MSS") (leaving: "Sleep Set") (entering: "Active Set")

  0x4bebba: rpm_bringup_ack (master: "MSS") (core: 0)

Extending the RPM External Log

- The interface to the RPM external log is provided in core\power\rpm\inc\rpm_log.h
- Adding a log message is easy:
  - #define a new log ID to use in rpm_log.h.
  - Wherever you need to log the message, #include rpm_log.h and call the following macro:
    - RPM_LOG_EVENT(YOUR_LOG_ID, your_data1, your_data2);
    - The number of data elements can be completely variable, but each argument has a cost, so <4 arguments is recommended.
    - A best practice is to have the arguments in each position always mean the same thing for a given ID, i.e., the first argument is always the master ID, the second argument is always the resource, etc.
  - rpm_log.py can then be easily extended to parse the new ID as required.
    - core\power\rpm\dal\scripts\rpm_log.py

RPM NPA Log

- The RPM also contains an NPA log similar to those found on other processors.
- The RPM NPA log is considerably smaller than on other processors, so using NPADump.cmm to retrieve the full NPA system state is generally suggested.
- This information is incorporated into the RPMLog, however, still need to run NPADump to get names.
  - ULOGDump.cmm – Gets the logs
  - NPADump.cmm – Gets the resource and client names
  - RPM_Log.py – Parses the log to a readable format

NPA Log Deciphering

- NPA log deciphering basics
  - Each entry contains:
    - HANDLE – Unique identifier to be used to see when requests start and complete
    - CLIENT – Master or RPM-based resource making the request
    - REQUEST – Resource state being requested
    - RESOURCE – Resource to which the client would like the request to be applied
- Entries in the log are paired
  - 0x4D351C – npa_issue_required_request (handle: 0x0003D7B0) (client: "Modem") (request: 155000) (resource: "/clk/bus/afab")
  - 0x4D3524 – Request complete (handle: 0x0003D7B0) (sequence: 0x001A6000) (request state:155000) (active state:174667)

NPA Log Deciphering Example 1 – Bus-Related Entries

0x4D351C: npa_issue_required_request (handle: 0x0003D7B0) (client: "Modem") (request: 155000) (resource: "/clk/bus/afab")

0x4D351D: npa_issue_required_request (handle: 0x0003D200) (client: "/clkregim/vdd_dig") (request: 1) (resource: "/pmic/client/clk_regime_dig")

0x4D351E: request complete (handle: 0x0003D200) (sequence: 0x0001B501) (request state:1) (active state:1)

0x4D3522: npa_assign_resource_state (handle: 0x0003CD08) (resource: "/clk/bus/afab") (active state: 174667)

0x4D3523: npa_assign_resource_state (handle: 0x0003CCC0) (resource: "/clk/bus/sfab") (active state: 87333)

0x4D3524: request complete (handle: 0x0003D7B0) (sequence: 0x001A6000) (request state:155000) (active state:174667)

NPA Log Deciphering Example 2 – Sleep-Related Entries

- RPM contains 1 low power node = /node/sleep/uber
  - 0x4D34A5: npa_issue_required_request (handle: 0x0003ED64) (client: "sleep") (request: 13) (resource: "/sleep/uber")
    - Requests are determined by looking at what bits are set: 0x13 => 1101 => mem dig pxo cxo
    - In this example:
      - Vdd Mem is required by system
      - Vdd Dig is required by system
      - PXO is not required by system
      - CXO is required by system
- Requests on the /node/sleep/uber usually result in a request to /sleep/lpr to enable/disable
  - 0x4D34A6: npa_issue_required_request (handle: 0x0003ED1C) (client: "/node/sleep/uber") (request: 1) (resource: "/sleep/lpr")
    - Requests values are:
      - 0x1 = RPM Halt only
      - 0x2 = XO Shutdown
      - 0x4 = Vdd minimization

NPA Log Deciphering Example 2 – Sleep-related Entries (cont.)

- Requests on the /node/sleep/uber usually result in a request to /sleep/lpr to enable/disable (cont.)

  0x4D34A7: request complete (handle: 0x0003ED1C) (sequence: 0x00013F01) (request state:1) (active state:1)

  0x4D34A7: request complete (handle: 0x0003ED64) (sequence: 0x0002D500) (request state:13) (active state:13)

- Usually suffices to look at /node/sleep/uber

Putting It All Together – Vdd Min Issue Debug Steps

- System is not entering Vdd minimization
- With T32
  - Step 1 – Look at contents of gpRPMFWMaster
    - Verify all Masters are in the sleep set and both Vdd Dig and Vdd Mem are voted for Vdd minimization values.
  - Step 2 – Correlate RPM Log entry for xo_shutdown_enter with NPA Log dump and examine the /node/sleep/uber requests that occurred directly before the xo_shutdown_enter
  - Step 3 – Look at contents of gpRPMFWMaster for all masters for all bus clocks and bus FABRIC-related requests
    - Watch out for inherited “on” requests from the active set.
- Without T32
  - Dump RPM log from the apps

Putting It All Together – Vdd Min Call Stack

rex_idle_task()

sleep_perform_lpm()

vdd_min_enter(latency = (enter = 0x1B, exit = 0x0141)

clk_regime_swfi(mode = CLKRGM_SWFI_MODE_DEEP_SLEEP = 0x4, flags = 0x0)

Extending the RPM External Log

- The interface to the RPM external log is provided in core\power\rpm\inc\rpm_log.h
- Adding a log message is easy
  - #define a new log ID to use in rpm_log.h.
  - Wherever you need to log the message, #include “rpm_log.h” and call the following macro:
    - RPM_LOG_EVENT(YOUR_LOG_ID, your_data1, your_data2);
    - The number of data elements can be completely variable, but each argument has a cost, so <4 arguments is recommended.
    - A best practice is to have the arguments in each position always mean the same thing for a given ID, i.e., the first argument is always the master ID, the second argument is always the resource, etc.
  - rpm_log.py can then be easily extended to parse the new ID as required
    - core\power\rpm\dal\scripts\rpm_log.py

References

Ref.	Document	
Qualcomm		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1

Questions?

https://support.cdmatech.com

REDEFINING MOBILITY