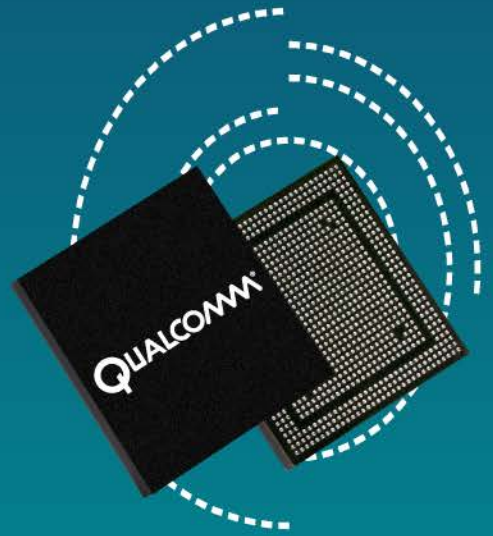
Qualcomm
TECHNOLOGIES, INC



MSM8x09 Resource Power Management (RPM) Debug Overview

80-NR964-9 A

Confidential and Proprietary – Qualcomm Technologies, Inc.

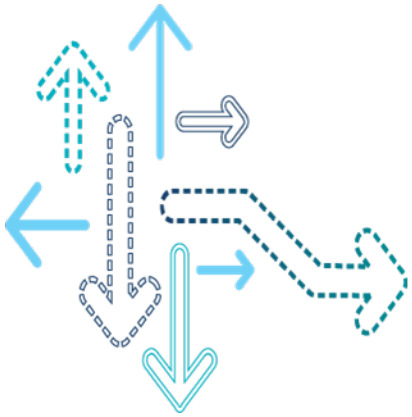
Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision History

Revision	Date	Description
A	Oct 2014	Initial release

Contents

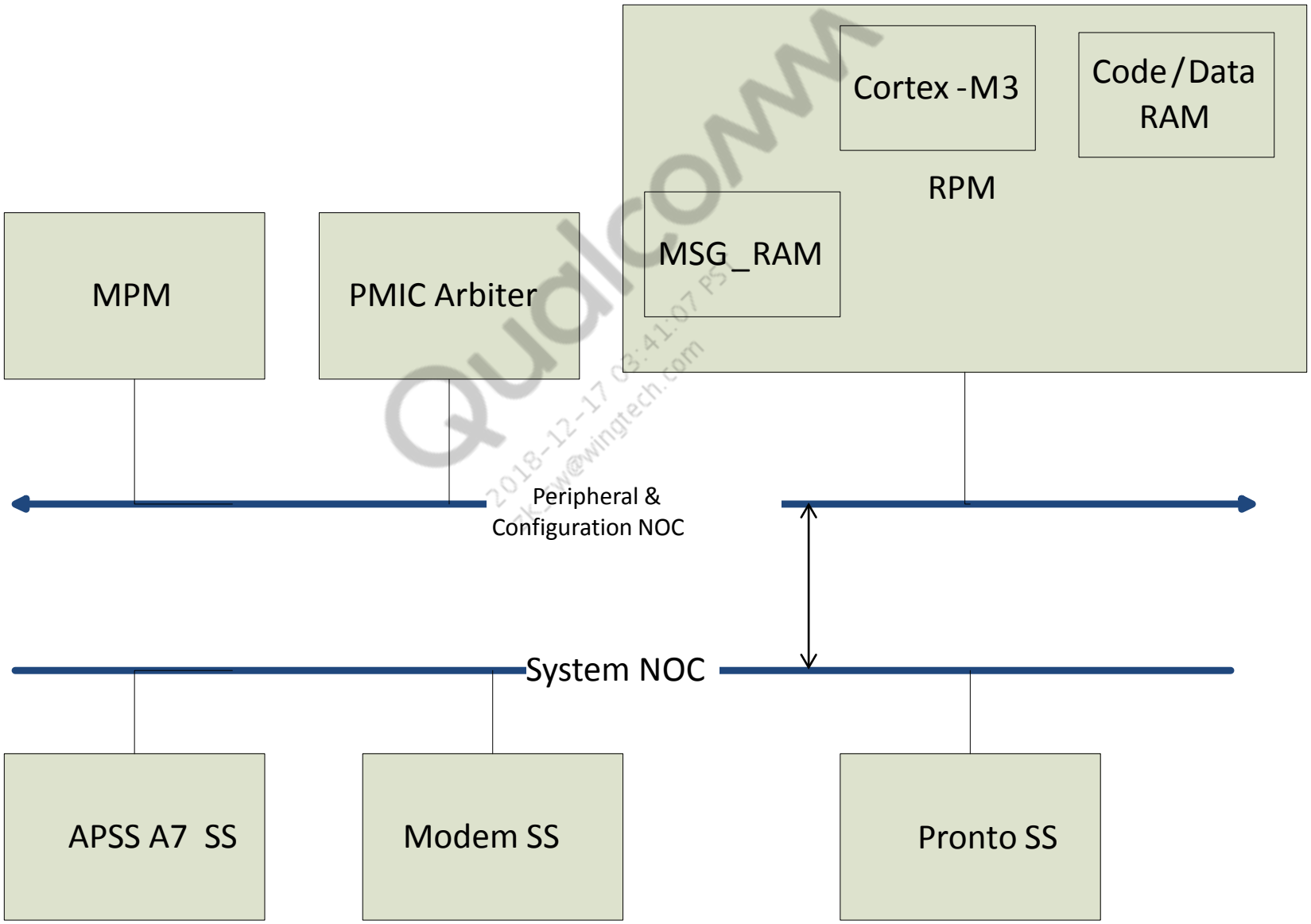
Overview



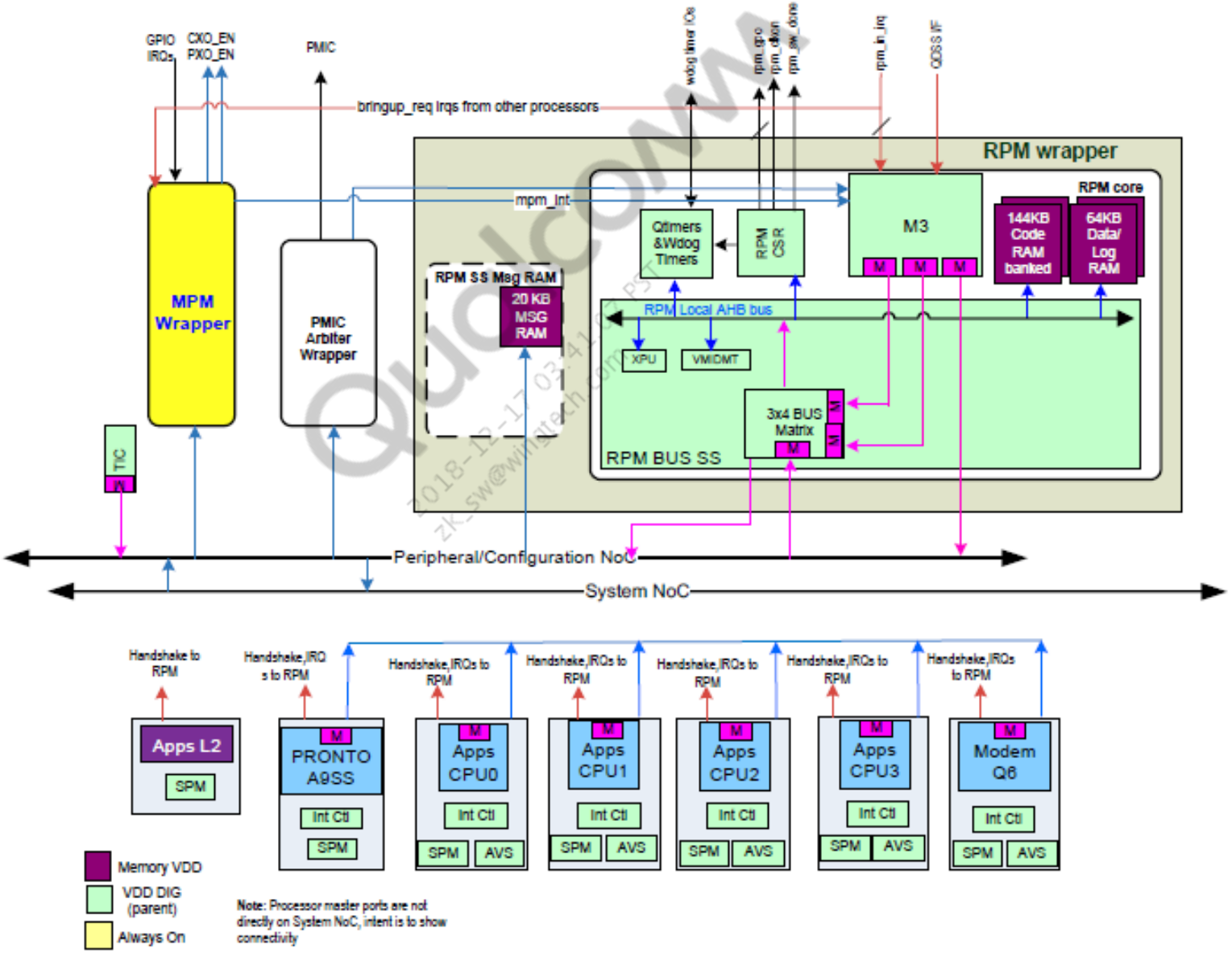
Resource Power Manager (RPM) Hardware Details

- RPM is an engine to manage SOC shared resources and attain the lowest dynamic and static power profile of that SOC
- Cortex-M3 – Harvard architecture processor
  - Higher MIPS and lower power
  - Harvard architecture with simultaneous instruction fetch with data load/store
  - Low-interrupt latency
- Code image exclusively from internal RAM
- (128 + 16) KB code RAM, 64 KB data RAM, and 20 KB msg RAM
- Supports frequency “Fast” – 150 MHz, and “Slow” (SVS) – 100 MHz
- Supports RBCPR for Vdd_Cx
- Supports open loop RBCPR for Vdd_Mx
- SPMI-based interface to PMIC
- Qtimer global counter – 56-bit @ XO clock speed
- Dedicated XPU protection for MPM

RPM High-Level Block Diagram



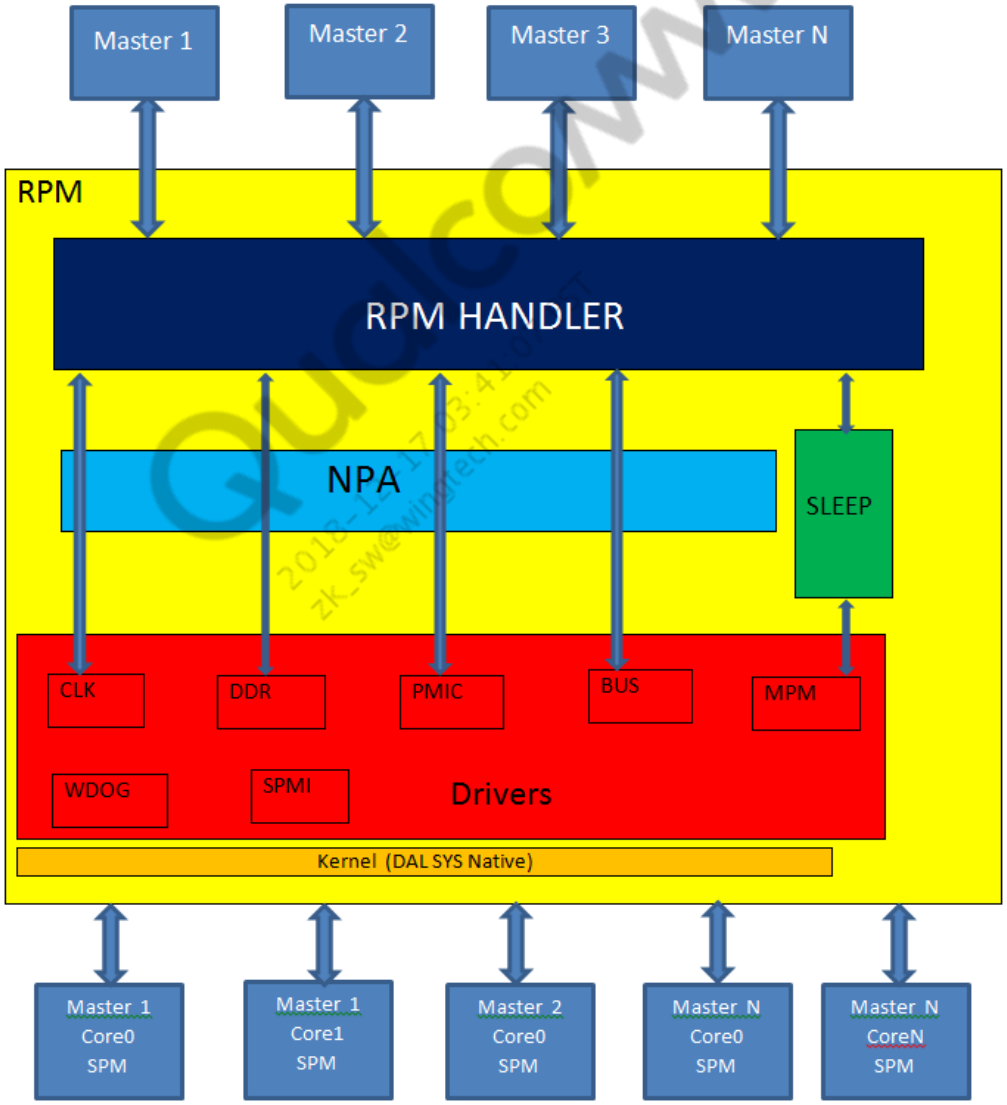
RPM Wrapper Block Diagram



Hardware Overview

- The RPM directly communicates with processors and/or hardware accelerators in each subsystem to process and coordinate shared power/resource requests
- The RPM code, RAM, provides storage for the RPM's code
- RPM SS MSG RAM is used for other masters in the system to communicate their resource requests to RPM
- RPM uses PMIC Arbiter to manage shared PMIC resources
- The RPM core has a local AHB bus connecting the Cortex M3 to its code and data/MSG RAM, and all its shared peripherals (timers, GPIOs, status, etc.)
- The RPM processor interrupt controller accepts IRQs from MPM, SPMs, and various masters
- MPM is always on domain to handle wake-up from different Sleep states, such as CLK_OFF, VDD_MIN, or power collapse
- The clock for the RPM subsystem (core and bus) can be configured at chip-level clock block from a PLL or directly from the XO clock. The frequency targets are 100 MHz at SVS and 150 MHz at nominal

High-Level Software Component Block Diagram – RPM



Software Components – RPM

- Kernel – DALSys-based lightweight kernel
- RPM handler – RPM handler abstracts the RPM message protocol away from other softwares
- Drivers – Drivers for each of the resources supported by the RPM register with RPM handler to request notification when requests are received for the resource which the driver controls
- Node Power Architecture (NPA) – A driver may use the NPA to represent resources controlled by the driver (example: /sleep/uber, /clk/bimc)
- Clock driver – RPM clock driver directly handles aggregating requests from each of the masters for any of the system-wide clock resources controlled by the RPM
- Bus arbitration driver – RPM bus arbiter driver takes bus arbiter settings as requests from different masters in the system and aggregates them to represent the frequency-independent system settings

Software Components – RPM (cont.)

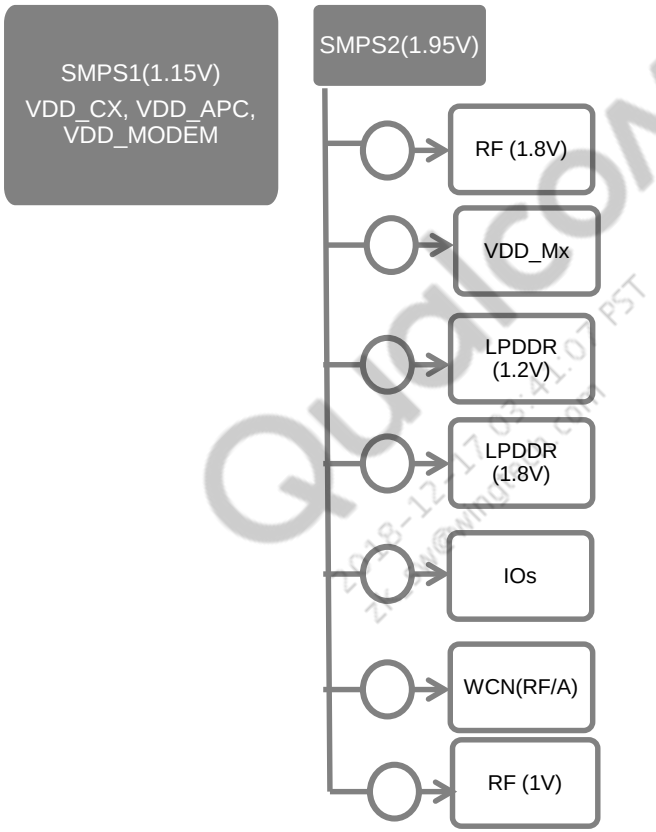
- PMIC – RPM PMIC driver directly aggregates requests from each of the masters for any of the system-wide PMIC resources controlled by RPM
- Watchdog driver – Watchdog driver is a fail-safe mechanism for incorrect or stuck conditions
- MPM driver – MPM driver programs the MPM hardware block during system-wide sleep to monitor the wakeup capable interrupts and to achieve voltage retention for CX and MX

Hardware Delta with MSM8916

- APSS A7 subsystem shares the voltage rail from the Vdd_Cx
- Open loop RBCPR for the Vdd_mx is enabled. Right voltage for the Vdd_Mx is applied based on MSM™ parts by reading fuse bits of the register in RPM software
- Please refer to the voltage table below:

Voltage corner	MMS8909	MSM8916	MSM8909	MSM8916	MSM8909	MSM8916
	VDD CX		VDD MX		VDD APC	
Turbo	1.250	1.2	1.2875	1.2875	1.250	1.2
Nominal	1.125	1.075	1.15	1.15	1.125	1.075
SVS	0.9625	0.95	1.05	1.05	1.05	1.05
Retention	0.5	0.5	0.7	0.7	GDHS	GDHS

MSM8909 Power Grid



MSM8909 + PM8909 + WTR2955

○----- Represents LDO

RPM Services

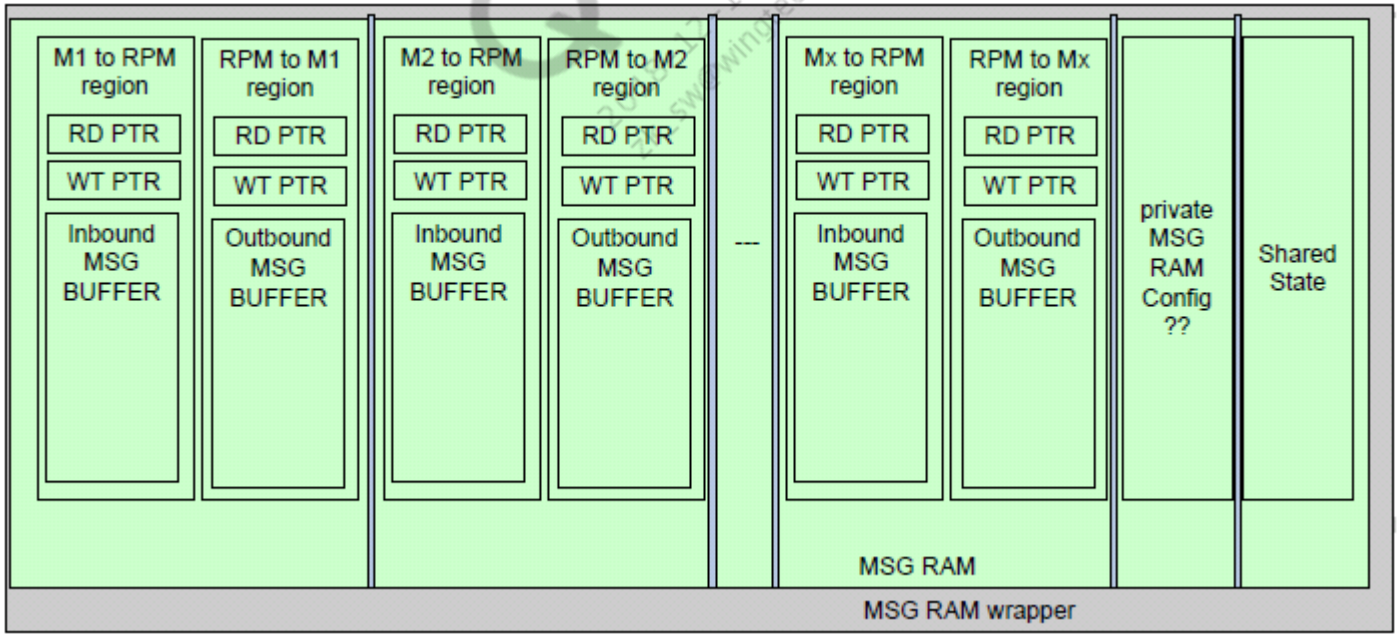
- RPM provide services that control:
  - Clocks
  - Bus arbitration
  - PMIC
  - DDR (DMM)
  - Chip sleep wake-up interrupts
  - Processor wake-up deadlines
- RPM allows control to vary across the following sets:
  - Active set
  - Sleep set
  - Next active set

Execution Environment (EE)

- EEs include:
  - APSS (A7)
  - Modem
  - Pronto

Message RAM Partition

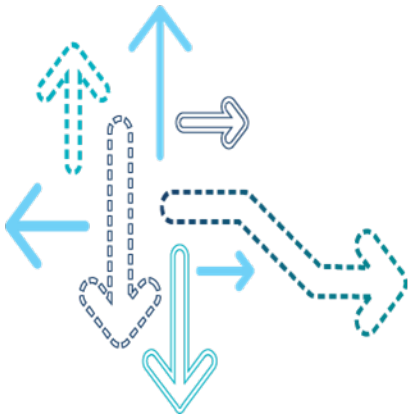
- Memory partitioned and protected for each processor/execution environment
- The message RAM is partitioned into several regions. A majority of the partitions are dedicated for master/RPM pairs. These partitions are only accessible by that master and the RPM
- No XPU protection for message RAM. Each master needs to have SMMU entry to access the designated MSG RAM area



RPM Memory Map

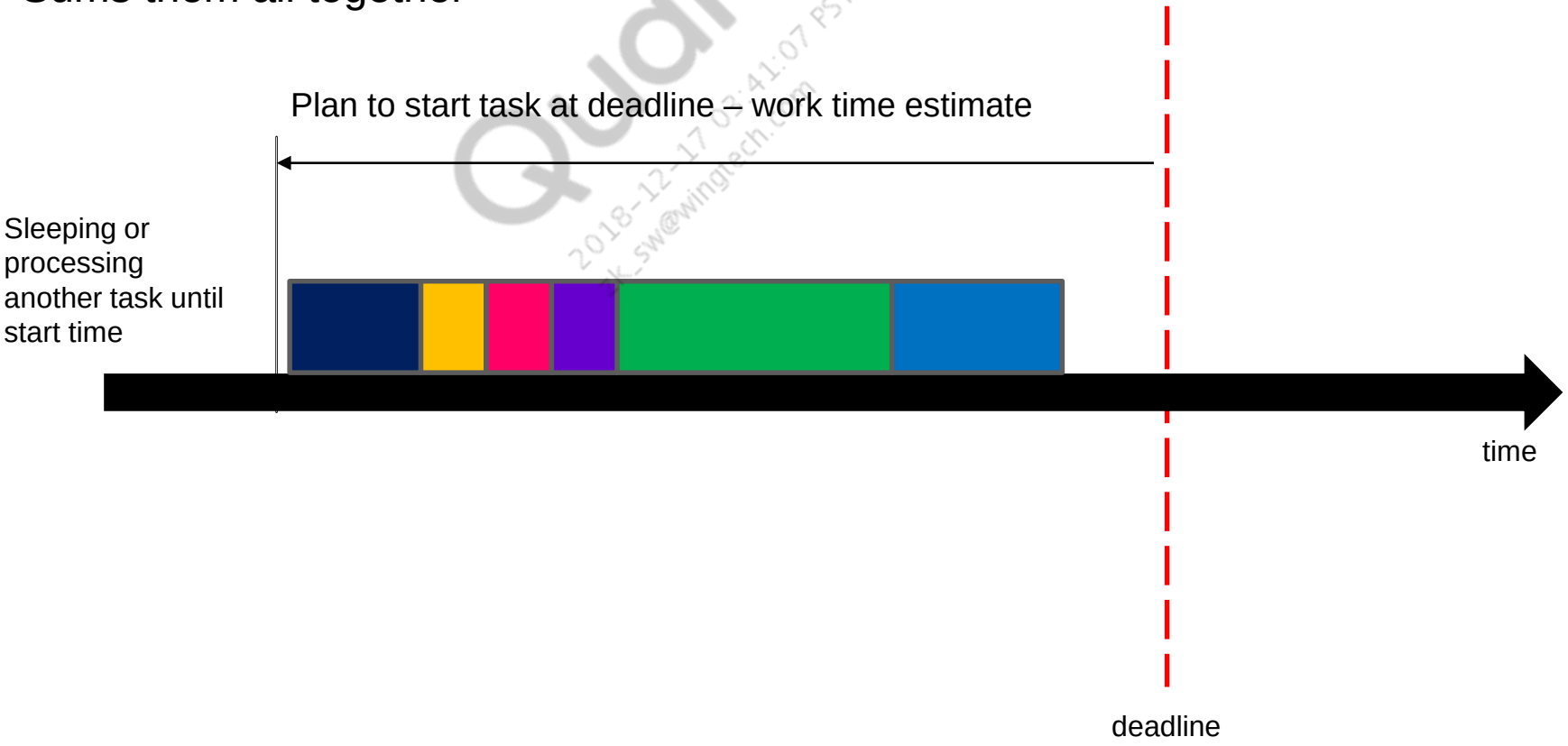
	RPM view	System view
RPM_CODE_START_ADDR	0x0000	0x200000
RPM_CODE_SIZE	0x24000	0x24000
RPM_DATA_START_ADDR	0x90000	0x290000
RPM_DATA_SIZE	0x10000	0x10000
RPM_MSG_RAM_ADDR	0x60060000	0x60000
RPM_MSG_RAM_SIZE	0x5000	0x5000

Scheduling

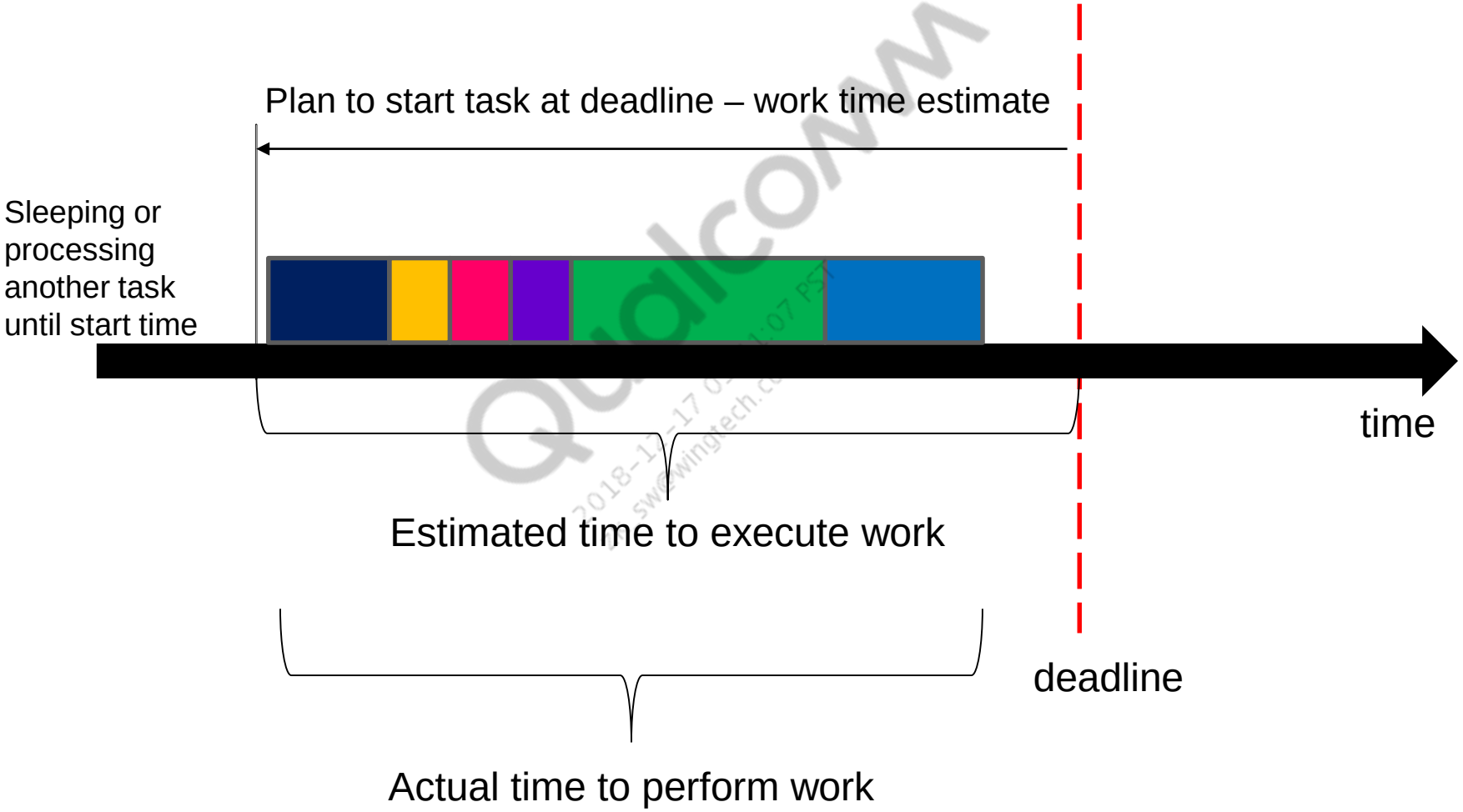


Scheduled Sleep Wake-Up

- RPM is a single-thread, time-based scheduler
- Current estimation algorithm
  - Determines which resources need to be updated
  - Looks up worst-case transition time for that resource, based on historical data
  - Sums them all together

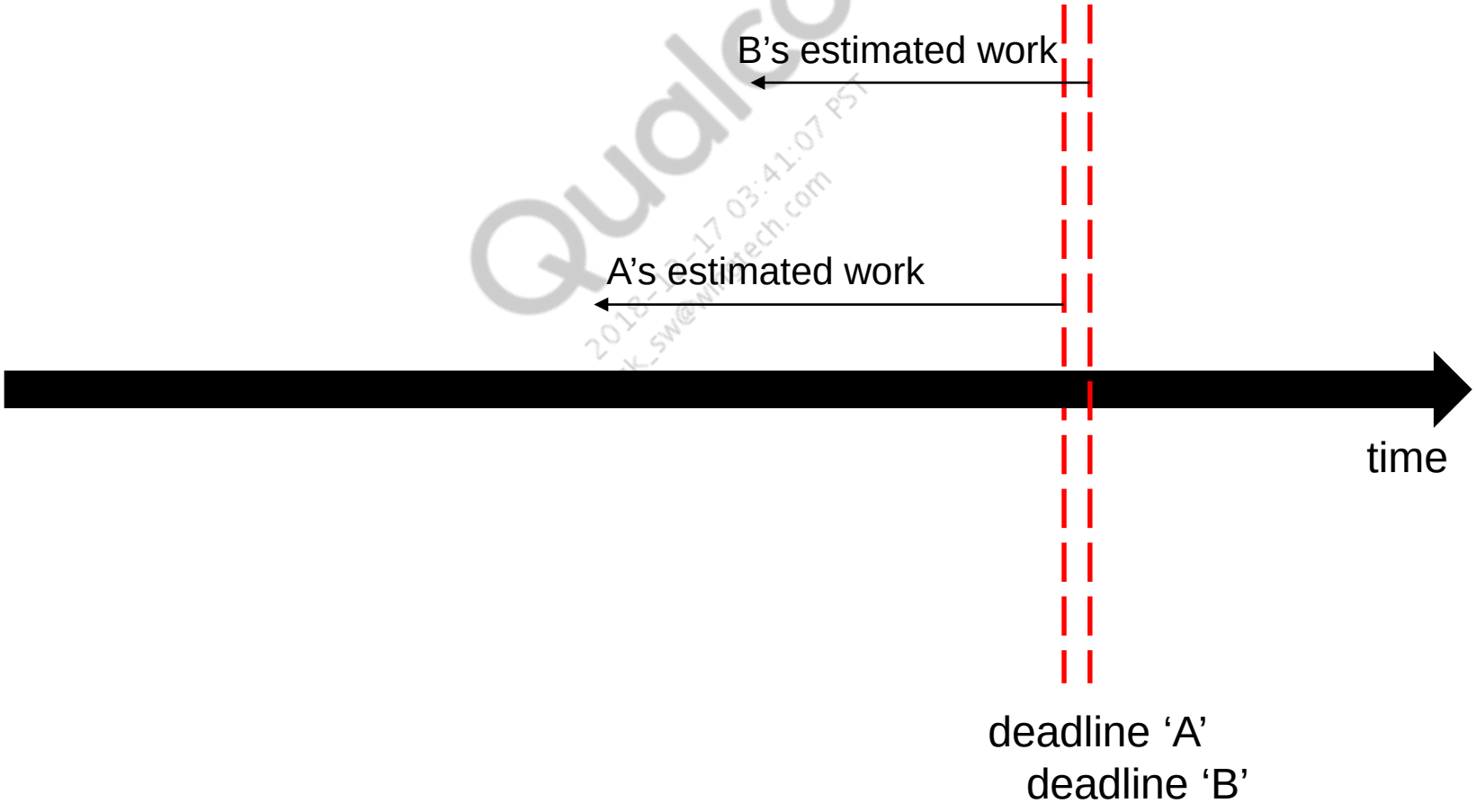


Scheduled Sleep Wake-Up (cont.)



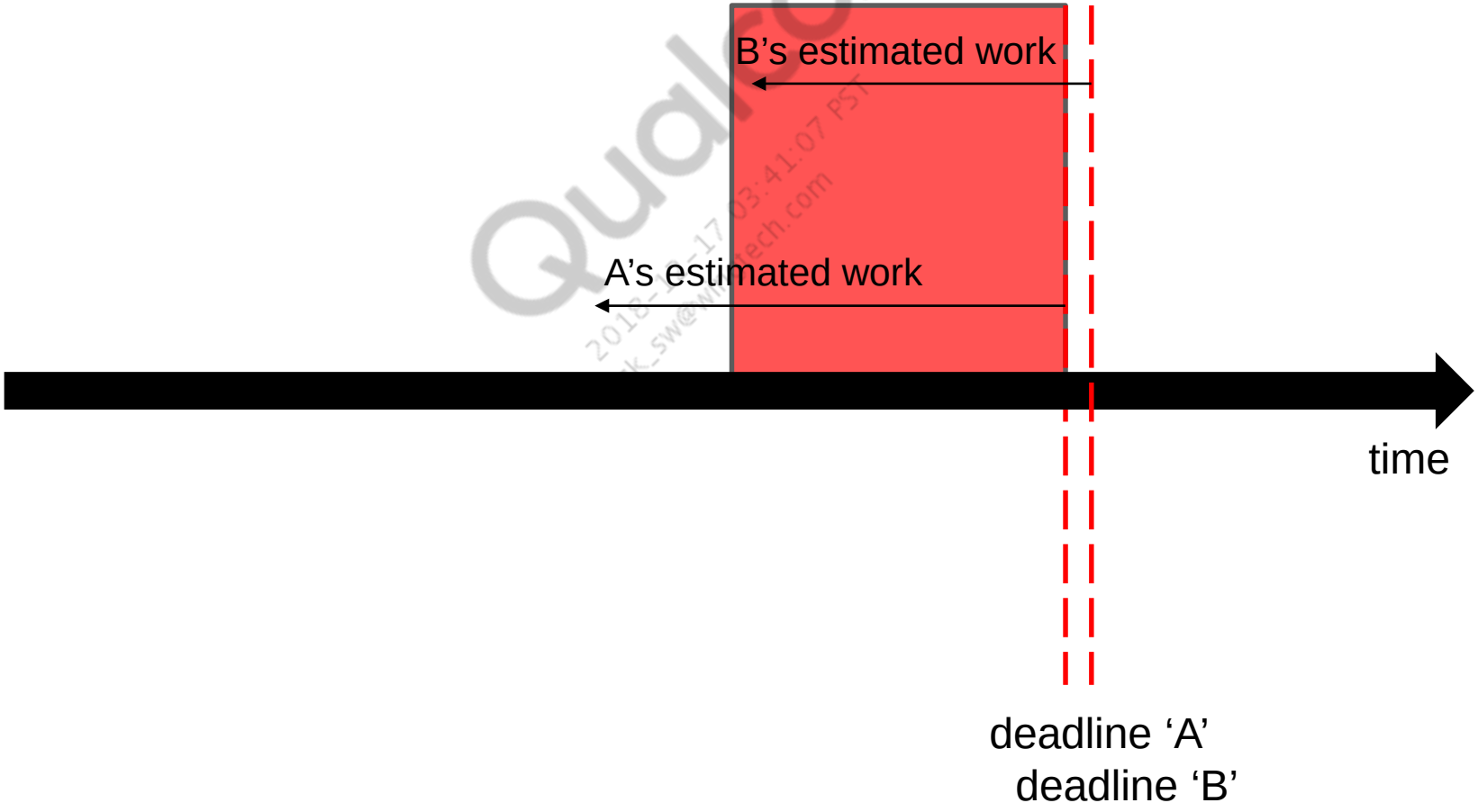
Schedule Collision – Two Scheduled

- Task A and task B are on schedule, each with their own deadline – deadline A and deadline B



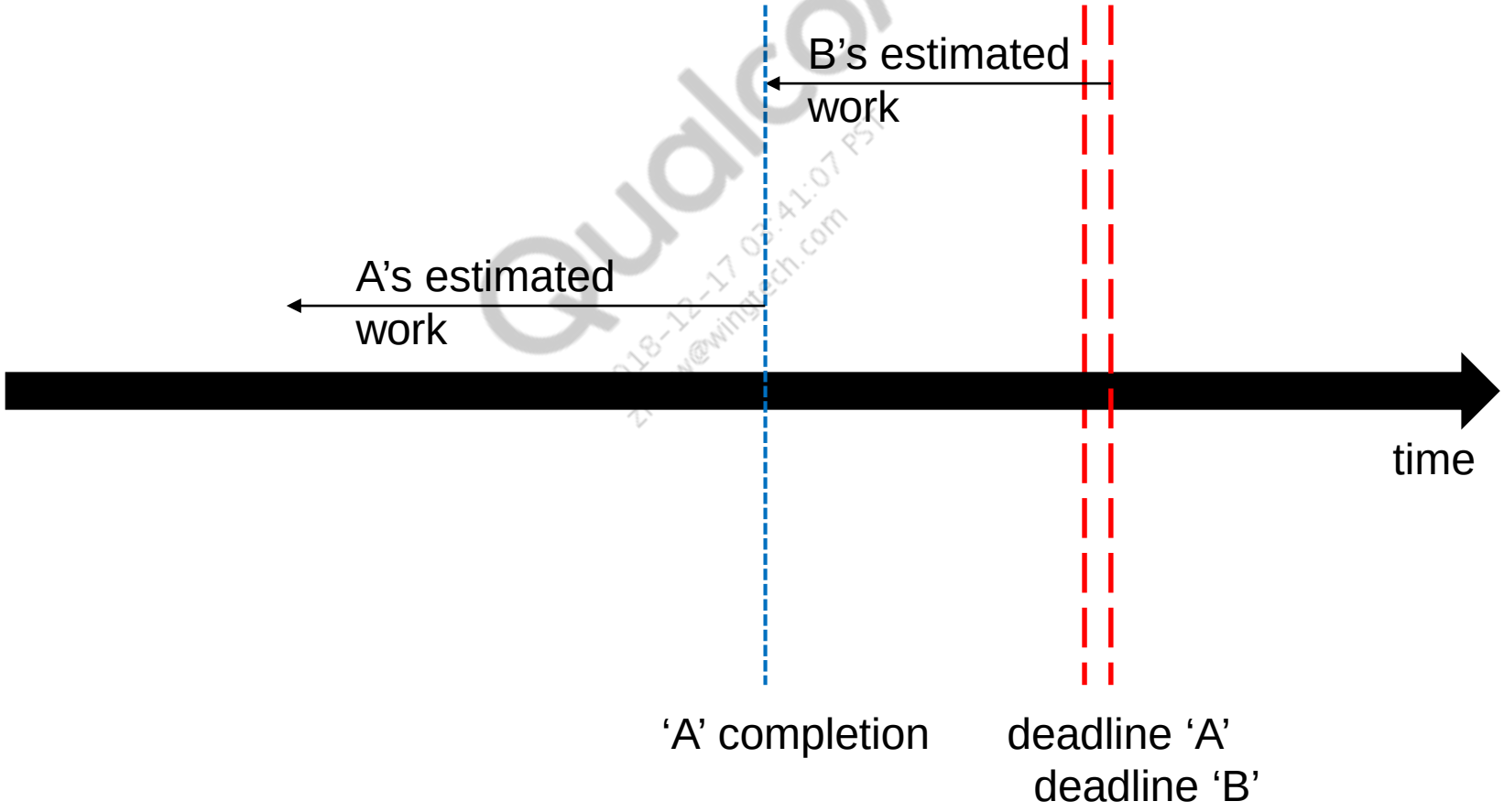
Schedule Collision – Two Scheduled (cont.)

- Conflict – Need to work on two things at once
- Resolution?
- Inspired by OS scheduling approaches known as “earliest deadline first”

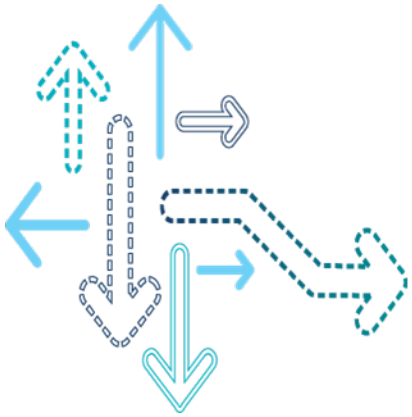


Schedule Collision – Two Scheduled (cont.)

- Based on the deadline, task A now conflicts with task B in the queue, so the execution time for task A is pushed out

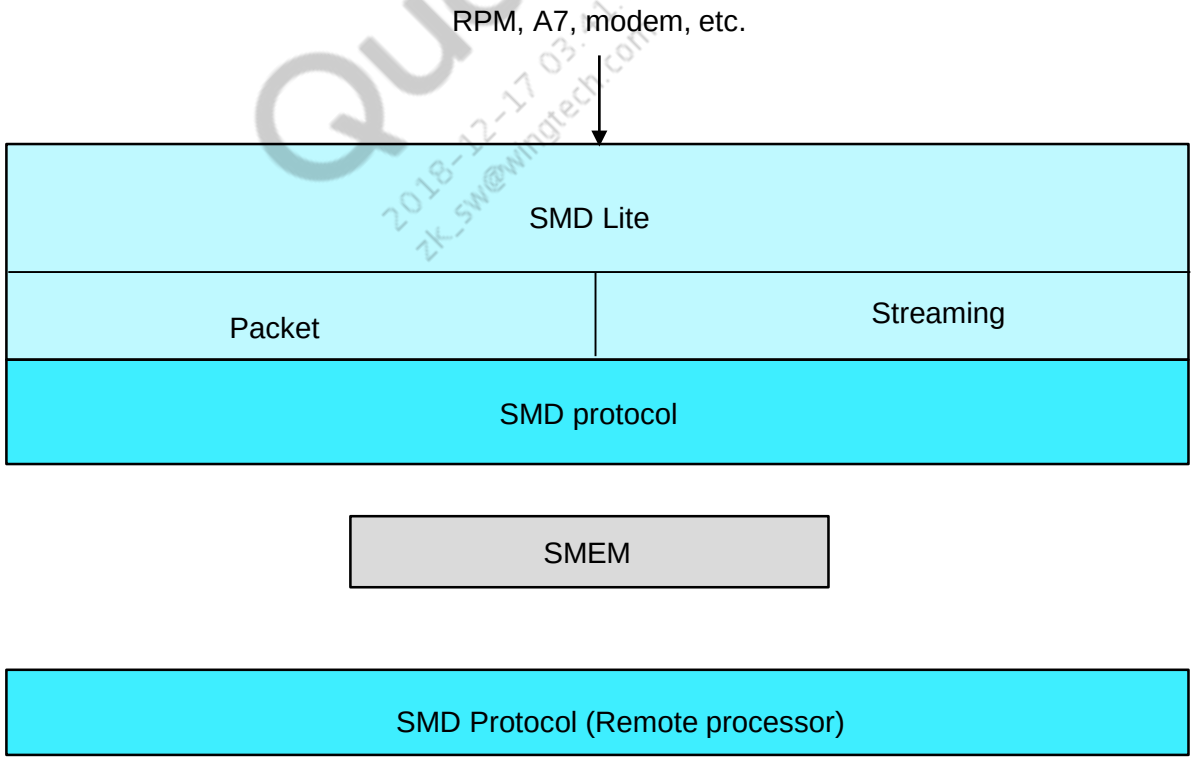


RPM Messaging Shared Memory Driver (SMD)/Key Value Pairs (KVP)



SMD Lite (SMDL)

- SMDL is a lightweight interprocessor communication layer
  - It is focused on the mechanism of moving data between processors. It remains unaware of the policy of what that data means and of higher-level constructs, such as queuing
- SMD is only a transport and still needs to layer a protocol on top



Message Structure – KVP

- SMDL understands messages formatted as KVP
- At a high level, a KVP structure is simply:
  - What you are talking about
  - Blob of data that you are saying
- At a low level, KVP is 2+n words of data
  - 1 word for “key” – Generally used as a 4-byte string (“uv\0\0”, “clk\0”, etc.)
  - 1 word for “length” – Describes how many bytes follow
  - Blob of data
- This structuring is repeated recursively to build full messages
- Internally, a request to an RPM is a concatenated array of KVP objects. KVP objects are simple in their layout in memory.

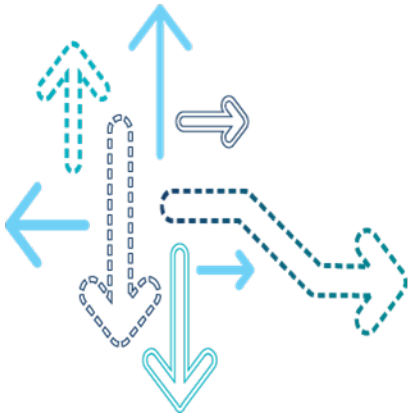
  <4 bytes of key> <4 bytes of length> <0..* bytes of value>

Message Structure – KVP (cont.)

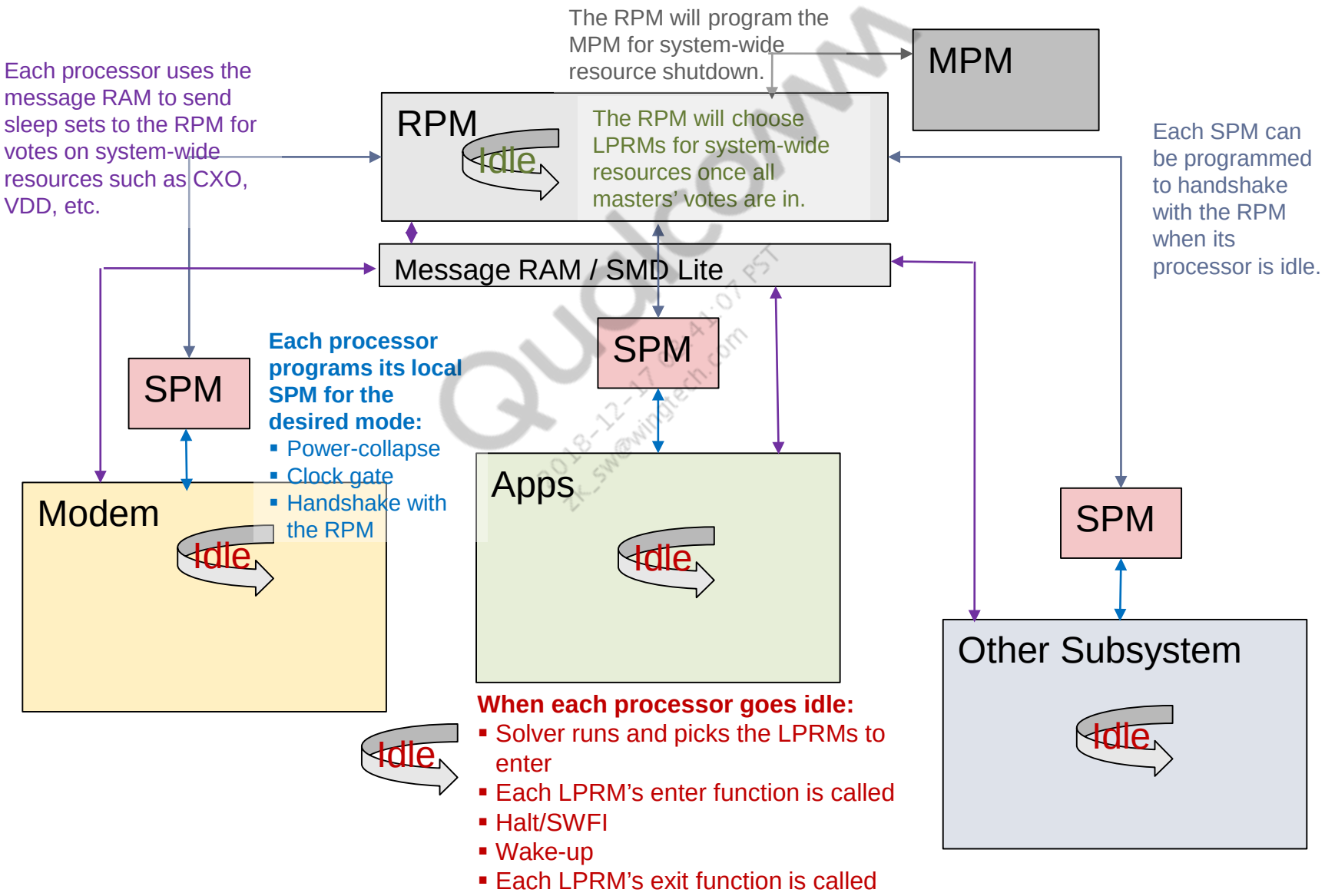
- Example of a request to change LDO3 voltage and current

```
{
    “req\0” : {
        {“rsrc” : “ldo\0”}
        {“id”   : 3}
        {“set”  : 0}
        {“data” : {
              {“uv\0\0” : 1100000}
              {“mA\0\0” : 130}
          }
        }
    }
}
```


System Sleep



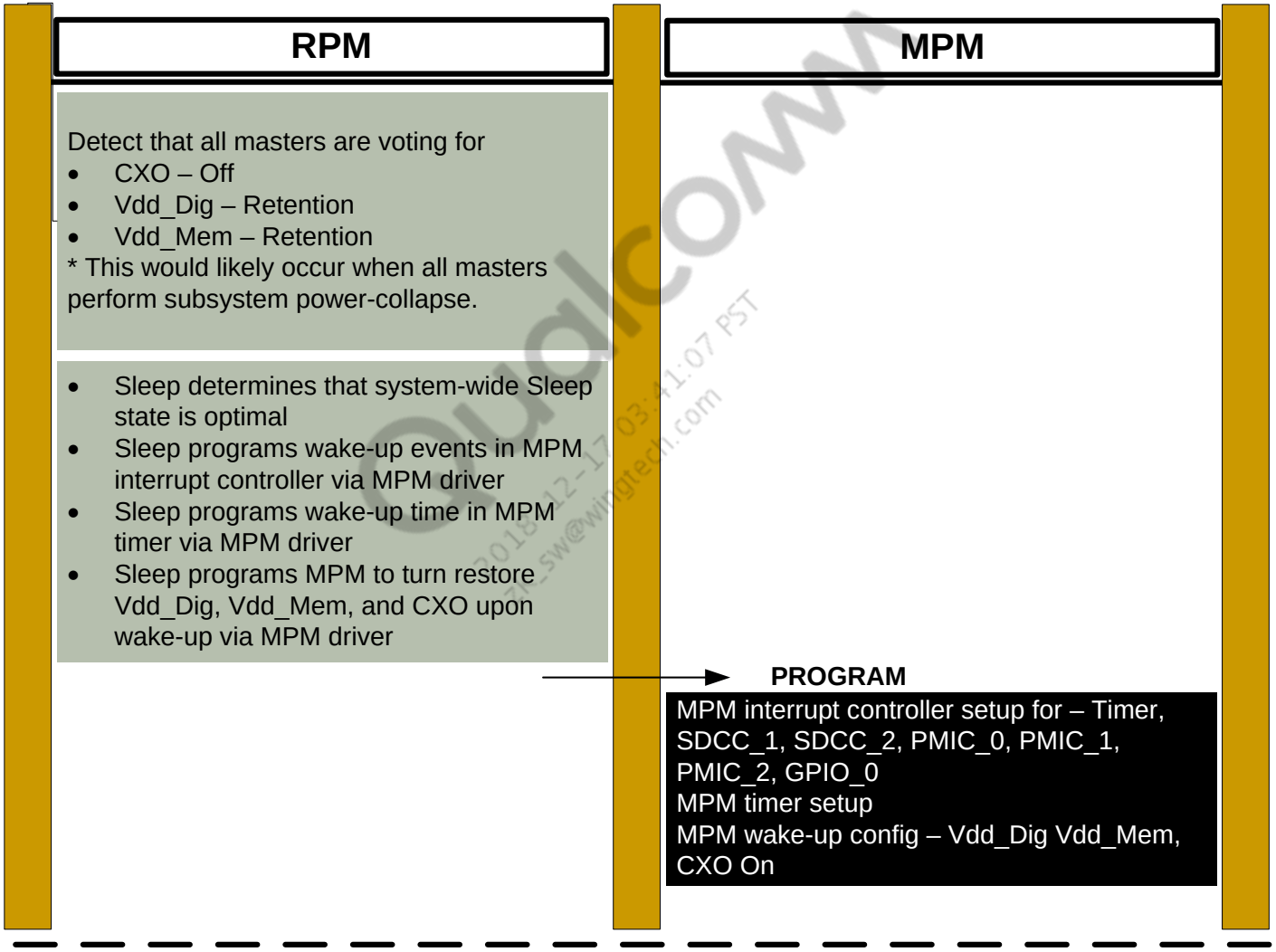
System Sleep Overview



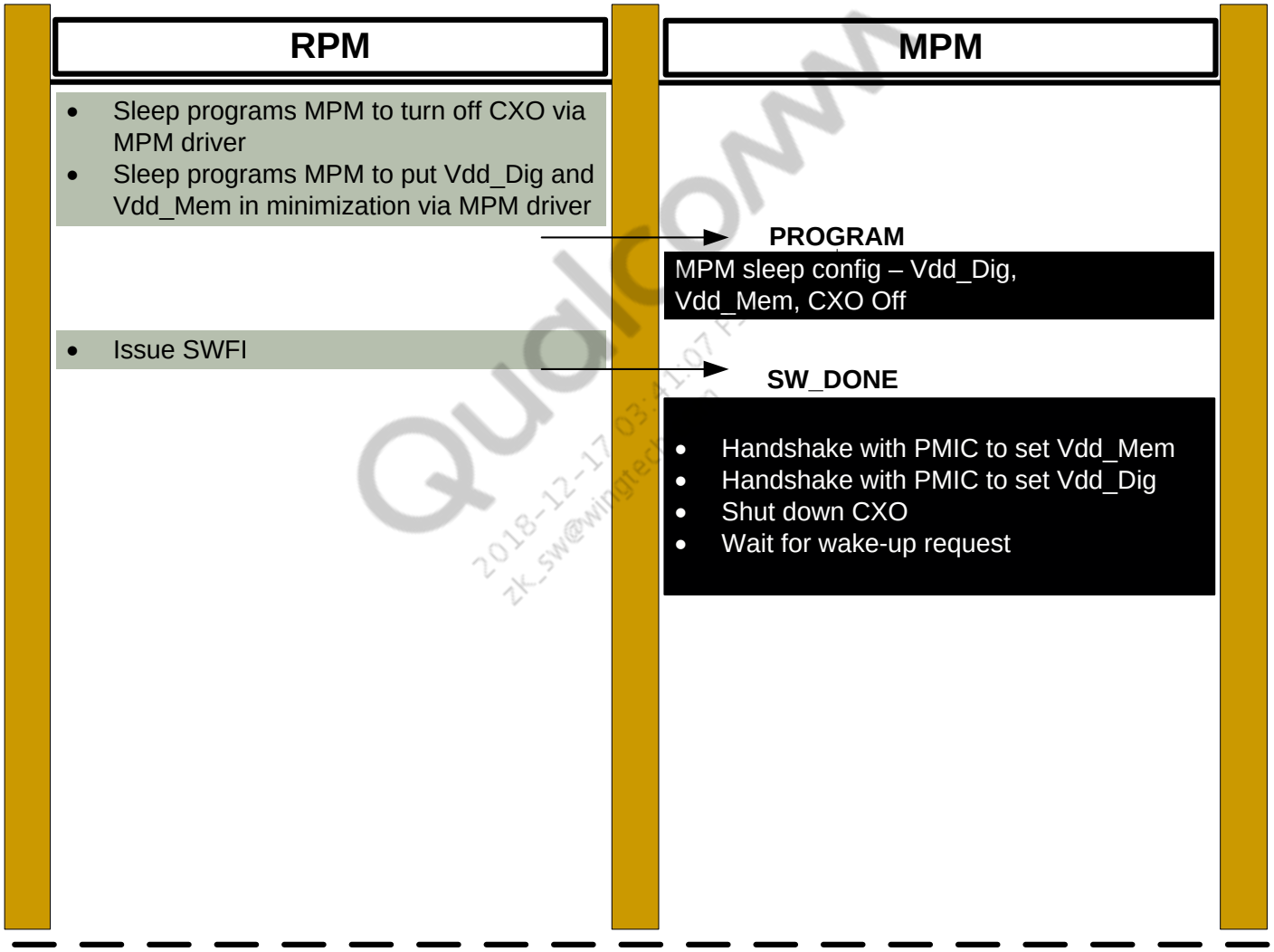
System Sleep Overview (cont.)

- XO shutdown – The first form of system sleep gates off (shuts down) the reference clocks in the system
- VDD minimization – The second form of system sleep puts the system power rails in a Retention state. This form can be used concurrently with the first form.

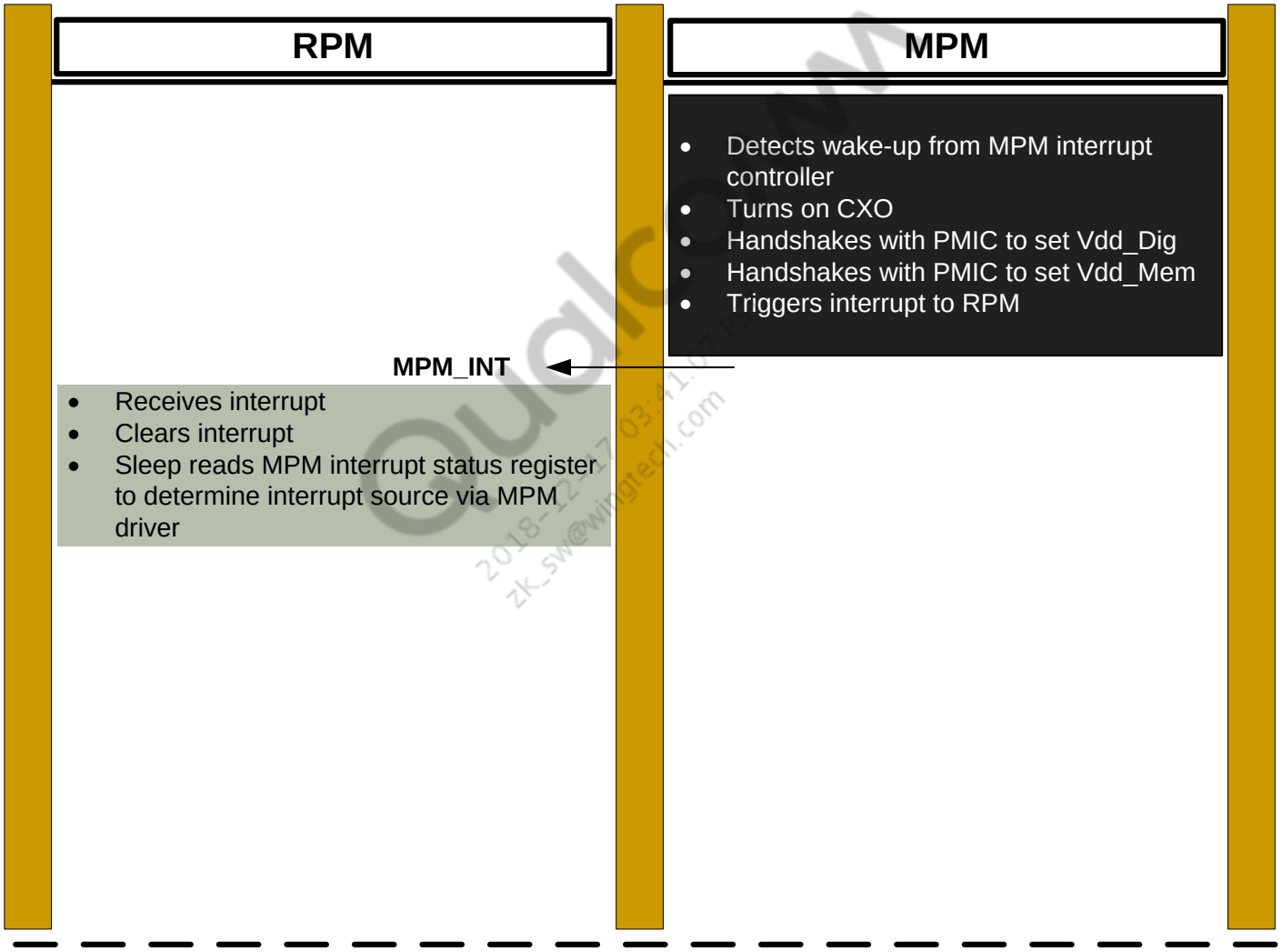
XO Shutdown/System Vdd_Min Flow



XO Shutdown/System Vdd_Min Flow (cont.)

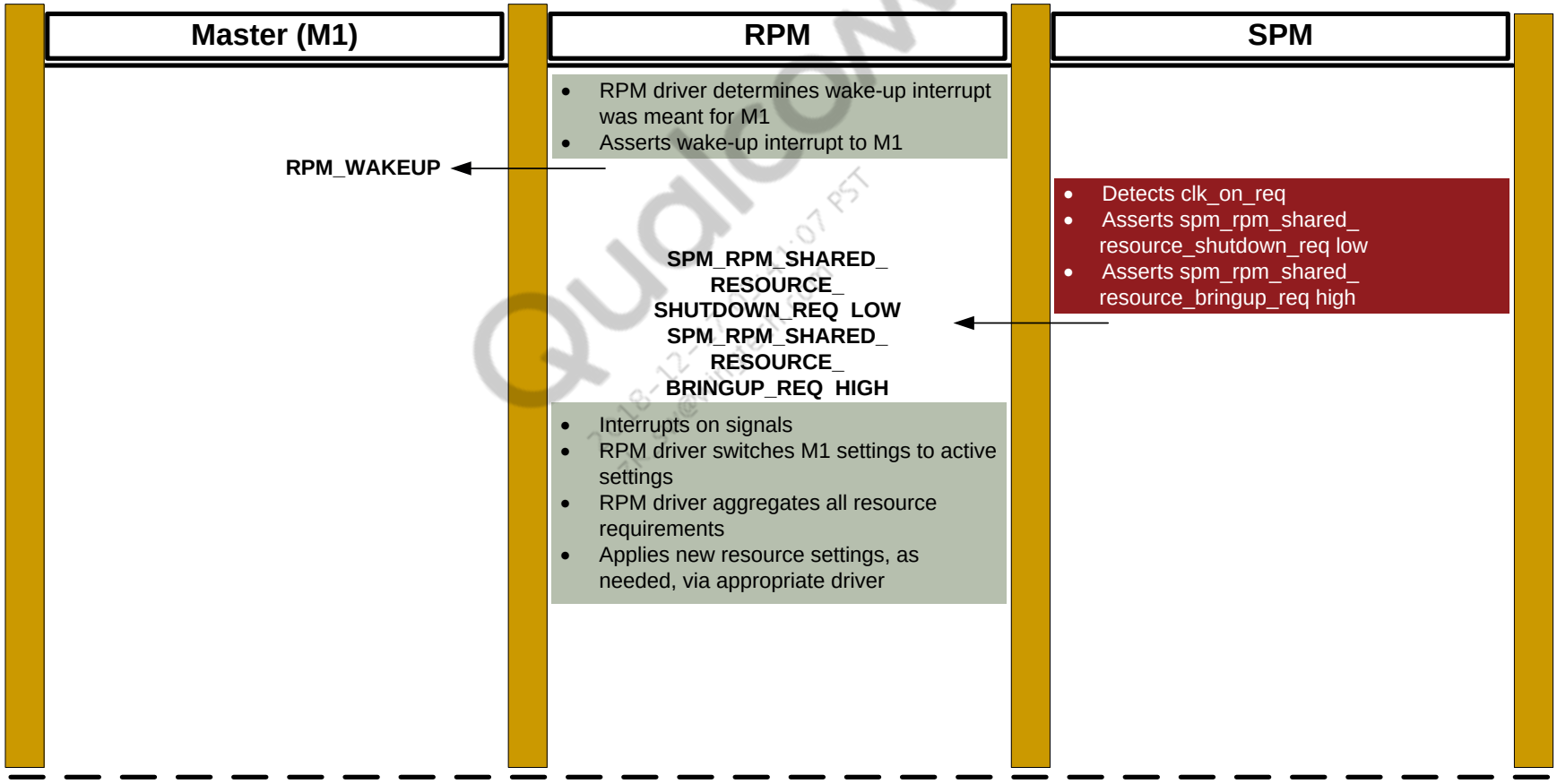


XO Restore/System VDD Restore Flow

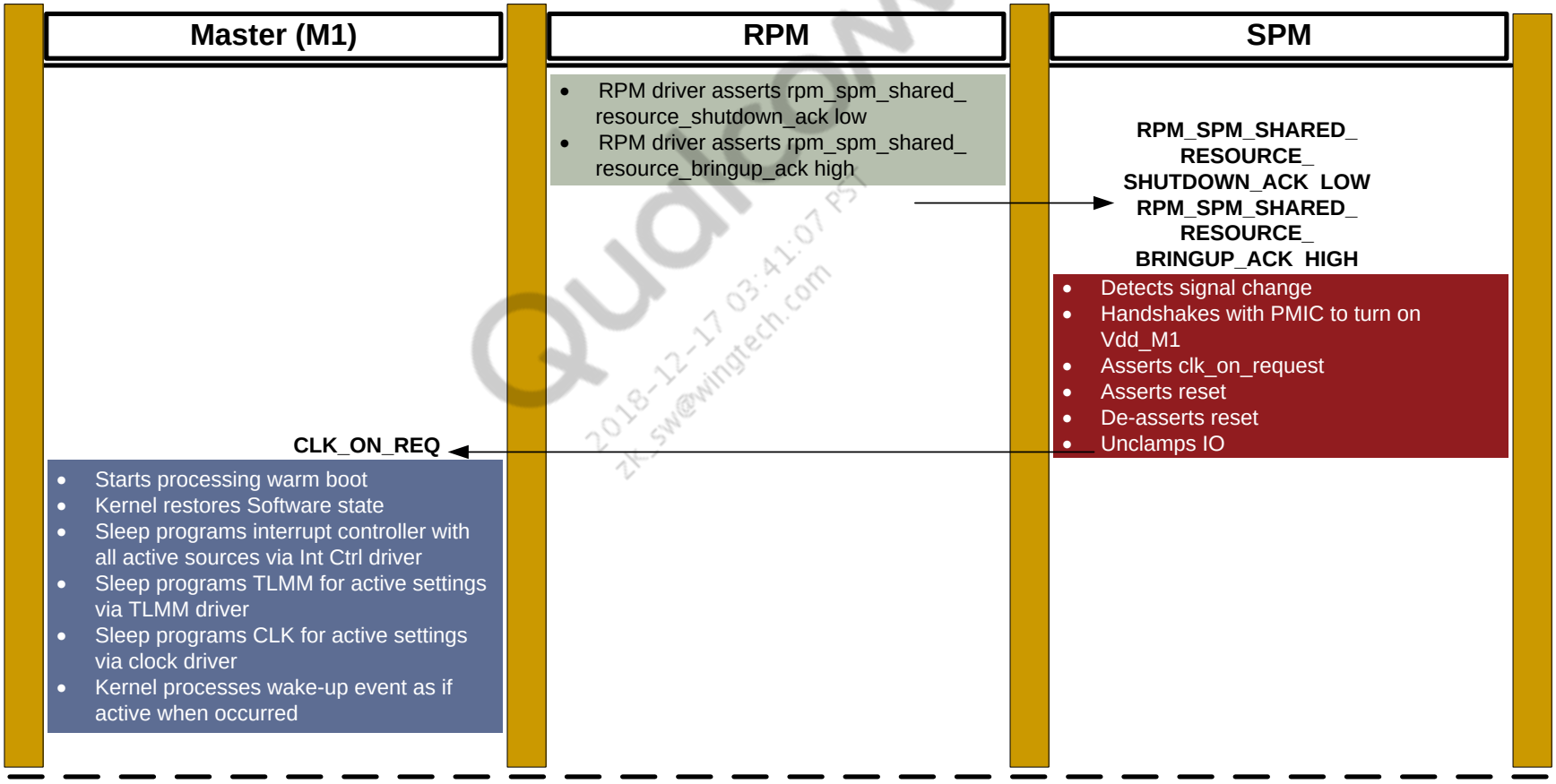


Subsystem Core Power Restore Flow

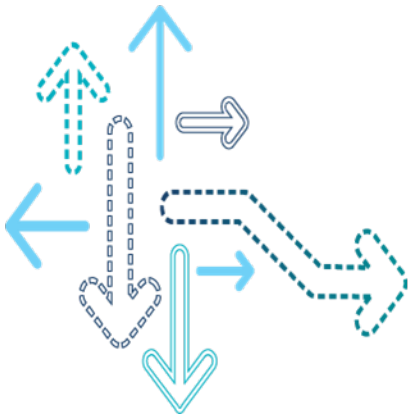
- Triggered from the end of XO restore/system Vdd_Min flow



Subsystem Core Power Restore Flow (cont.)



Rapid Bridge Core Power Reduction (RBCPR)

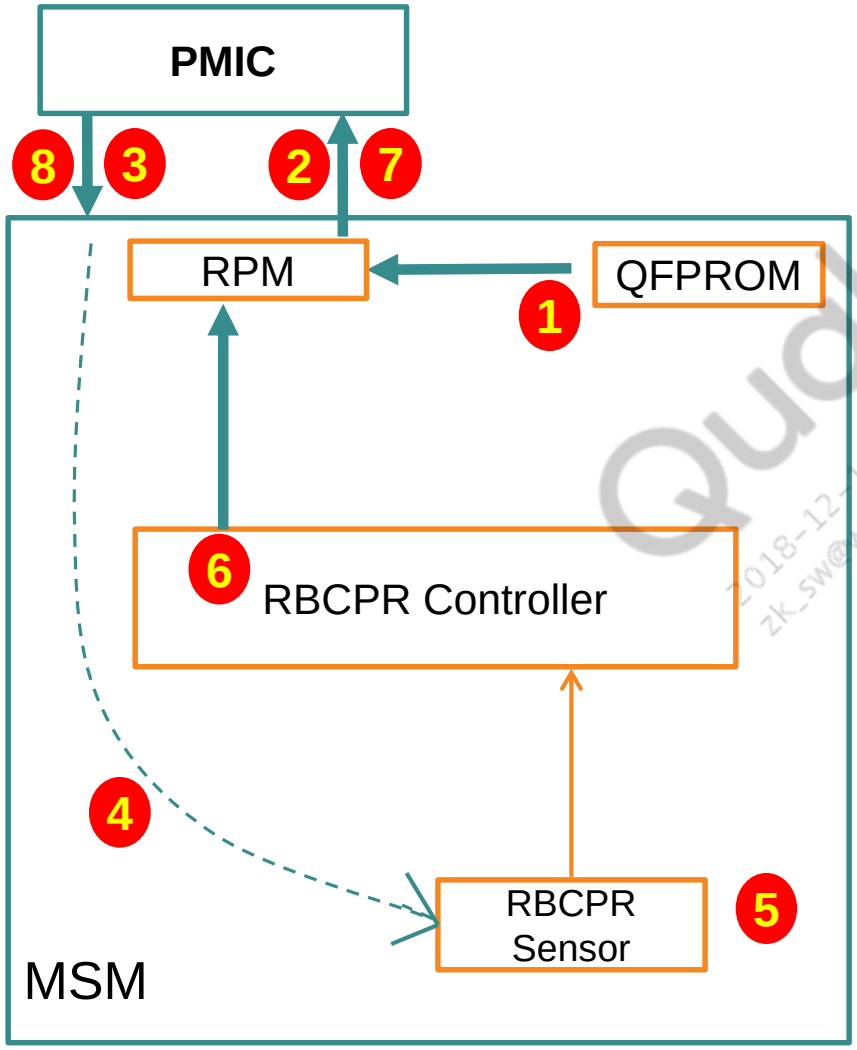


RBCPR Overview

- RBCPR technology provides a feedback loop to optimize the voltage setting
- Two main use cases of RBCPR are:
  - Setup – When the voltage corner is changed and the RBCPR configuration must be changed
  - Adjustment – When the RBCPR hardware senses that a voltage change is necessary and the RBCRP software adjusts the voltage
- RBCPR is expected to be applied to the following domains and be controlled by the indicated execution environment:
  - VDD Dig(Vdd_Cx) – Controlled by RPM
  - Vdd Memory(Vdd_Mx) – This is the open loop RBCPR. Voltage for the Vdd_mx is applied based on the MSM part by reading the fuse bits in RPM software.
- CPR driver source code
  - The source code resides in the folder rpm_proc\core\power\rbcpr\
  - Floor and ceiling Voltage settings for each corner can be found in the file:

    rpm_proc/core/power/rbcpr/src/target/8909/rbcpr_bsp.c
- Cx rail voltage is adjusted by the CPR driver
- The recommendation provided by the CPR is in terms of PMIC steps; one step = 12.5 mV

RBCPR Block Diagram



Open-loop operation (Boot-up):

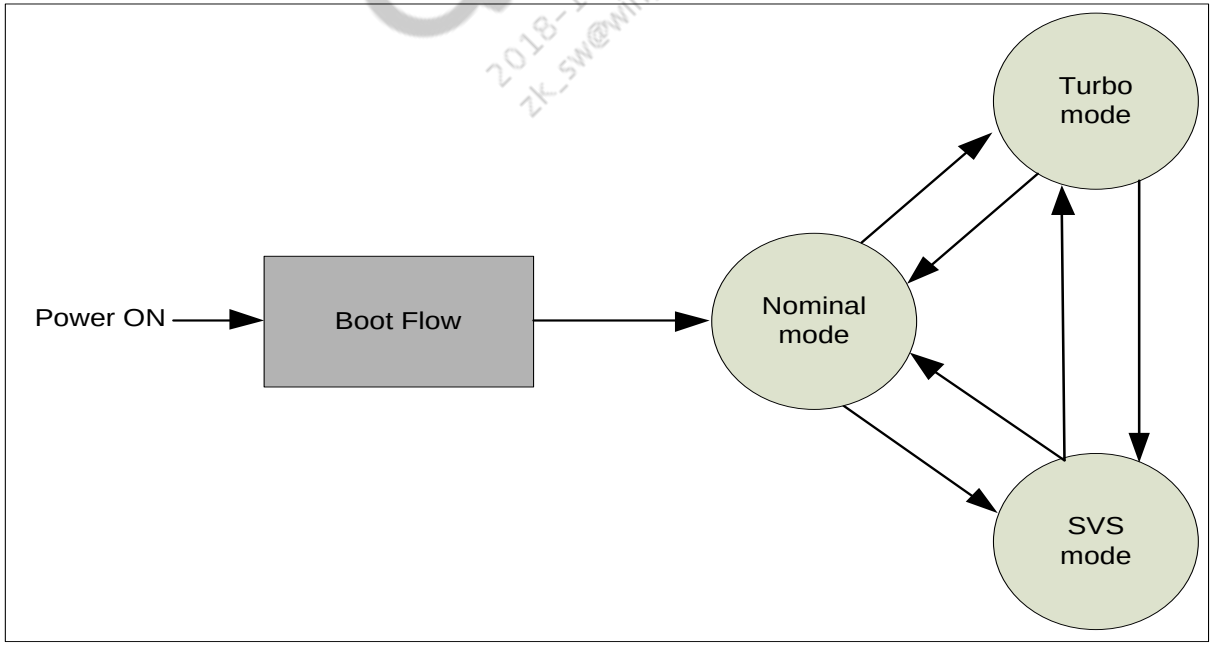
1. RPM reads e-fuses
2. RPM sets initial voltages in the PMIC (In case of Closed loop, voltages are read from CPR table)
3. PMIC outputs updated voltage to MSM

Closed-loop operation:

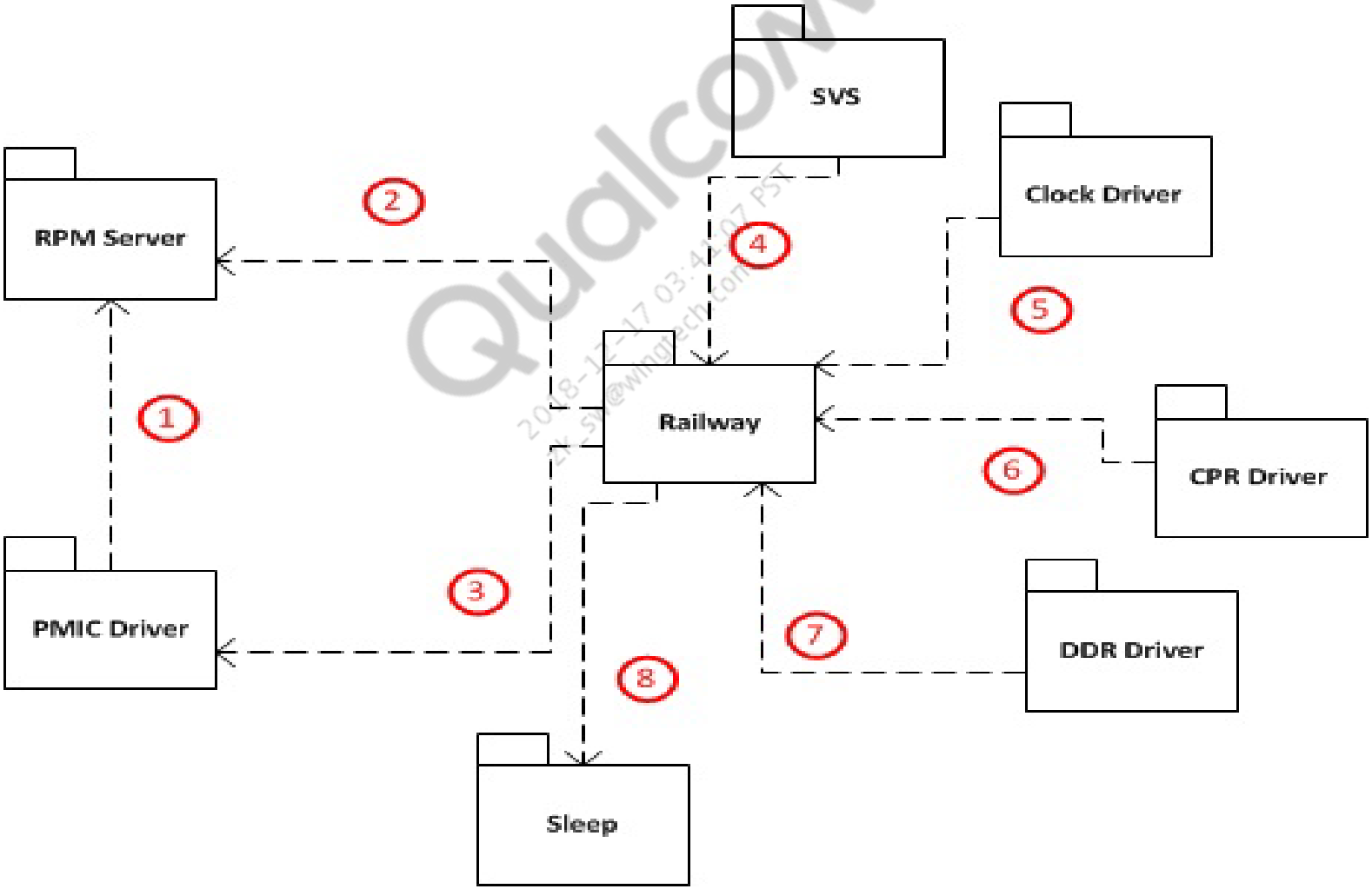
4. Updated voltage affects RBCPR sensors
5. Sensors generate RBCPR data
6. RBCPR controller computes voltage re-adjustment and interrupts the RPM
7. RPM writes voltage re-adjustment to PMIC
8. PMIC outputs updated voltage to MSM

RPM Mode (Voltage Corners)

- RPM transitions in between below voltage corners are based on the following subsystem requests:
  - SVS
  - Nominal
  - Turbo
- The CPR driver gets notified of the mode transition and changes its configuration. When there is a requirement to switch the core voltage for a mode change, RBCPR callbacks are called.



Railway Software Architecture



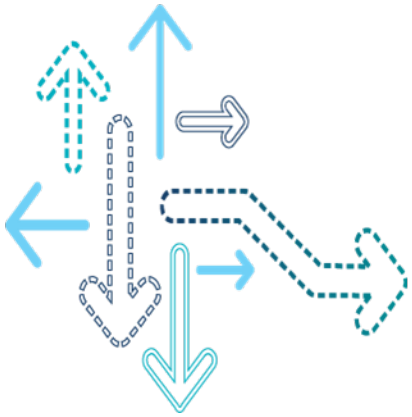
Railway Software Architecture (cont.)

1. The PMIC driver registers with the RPM server to receive all PMIC requests (RPM_SMPS_A_REQ, RPM_LDO_A_REQ, etc.)
2. Railway registers with the RPM server to receive PMIC requests for specific resources, i.e., Vdd_Cx, Vdd_Mx, Vdd_Gfx
3. Railway makes requests for voltages on the rails it manages directly to the PMIC driver. The PMIC driver is responsible for understanding the power grid of the specific target being run on and adjusting the parent regulators as required.
4. SVS registers with Railway for notifications on when Vdd_Cx changes. It also uses the Railway API for voting for Vdd_Cx up when it wants to boost the CPU speed.

Railway Software Architecture (cont.)

5. Clock driver registers with Railway for notifications on when Vdd_Cx or Vdd_Gfx change. It also votes on /pmic/client/clk_regime_dig and /pmic/client/gfx nodes for when it wants to change voltages on Vdd_Cx and Vdd_Gfx respectively; these nodes are implemented by the Railway component.
6. CPR registers with Railway for notifications when either Vdd_Cx or Vdd_Gfx changes, so that it can reinitialize the CPR hardware to the new corner. It also uses an API on Railway to initiate a voltage change when the dynamic CPR-derived voltage for the current corner has been updated.
7. The DDR driver registers with Railway for notifications on when Vdd_Cx and Vdd_Mx change.
8. Railway is responsible for updating the /sleep/uber node depending on whether there are any non suppressible votes for Vdd_Cx and Vdd_Mx which would prevent Vdd minimization.
9. Voltage settings for each corner can be found in file:
   rpm_proc/core/power/railway_v2/src/8909/railway_config.c
10. CX voltage settings for each corner in the above file is considered if RBCPR is disabled.

Debug



Saving RPM Dumps

- OEMs can save the RPM dumps in the following manner and send for further analysis:
  1. Create an issue scenario where the RPM dumps are needed.
  2. Open ARM7 Trace32 (T32), and attach (sys.m.a).
  3. Break T32 and perform the following to save the RPM memory dump:
     - d.save.binary CODERAM.bin RPM_CODE_START_ADDR++(RPM_CODE_SIZE - 1)
     - d.save.binary MSGRAM.bin RPM_MSG_RAM_ADDR++(RPM_MSG_SIZE - 1)
     - d.save.binary DATARAM.bin RPM_DATA_START_ADDR++(RPM_DATA_SIZE - 1)

- OEM should use the system View Memory map shown in the slide Message RAM Partition, if RPM memory needs to be accessed from outside of RPM space

Loading RPM Dumps onto T32 and Extracting Logs

- To load the RPM dumps received from OEMs onto a T32 simulator:
  1. Use RPM View Memory map, shown in slide Message RAM Partition, to load the RPM dumps onto a T32 simulator.
  2. Open a T32 simulator and perform sys.up.
  3. Navigate to the dumps directory and load the same as shown below:
     - d.load.binary CODERAM.bin RPM_CODE_START_ADDR
     - d.load.binary MSGRAM.bin RPM_MSG_RAM_ADDR
     - d.load.binary DATARAM.bin RPM_DATA_START_ADDR

Extract RPM Logs with rpm_log_bfam.py

- Extracting logs:
  1. Extract an RPM external log
     - Perform rpm_proc\core\power\ulog\scripts\ULogDump.cmm <path to your directory>
  2. Extract an NPA log
     - Perform rpm_proc\core\power\npa\scripts\NPADump.cmm <path to your directory>
  3. Execute the Python script for post processing
     - python rpm_proc\core\power\rpm\debug\scripts\rpm_log_bfam.py -f “RPM External Log.ulog” -n “NPA Log.ulog” > rpm_parsed.txt
  - Additional switches are –r, which print raw (hex sclk value) timestamps

RPM External Log – Analysis

- The RPM publishes a small log into a very limited area of data RAM.
- The physical format of the log is the ULog format used for various other logs.
  - Circular buffer, currently sized at 8 KB
  - Raw log, using a set of IDs and a variable number of parameters per message
- Message request contents – Timestamp, operation, data

0x000000002f83de3e: rpm_message_received (master: "APSS") (message id: 2230)
0x000000002f83deb9: rpm_svs (mode: RPM_SVS_FAST) (reason: imminent processing)
0x000000002f83e003: rpm_process_request (master: "APSS") (resource type: bslv) (id: 45) (full name: ICBID_SLAVE_PCNOC_SNOC)
0x000000002f83e046: rpm_xlate_request (resource type: bslv) (resource id: 45) (full name: ICBID_SLAVE_PCNOC_SNOC)
0x000000002f83e091: icb_slaves_translate_cb (slave: ICBID_SLAVE_PCNOC_SNOC) (bandwidth: 0)
0x000000002f83e0c0: rpm_apply_request (resource type: bslv) (resource id: 45) (full name: ICBID_SLAVE_PCNOC_SNOC)
0x000000002f83e0ff: icb_slaves_apply_cb (slave: ICBID_SLAVE_PCNOC_SNOC) (bandwidth: 0)
0x000000002f83e219: Clock: gcc_pcnoc_ahb_clk Frequency = 19MHz
0x000000002f83e293: rpm_send_message_response (master: "APSS")

RPM External Log – Analysis (cont.)

- Message request contents – Entering Low Power mode

0x000000002f8da287: deep_sleep_enter: (mode: "VDD Minimization") (count: 1)
0x000000002f8da385: Clock Processor Collapse: Enter
0x000000002f8da8b7: Clock: gcc_bimc_clk Frequency = 100MHz
0x000000002f8da9e7: Clock: gcc_apss_tcu_async_clk Frequency = 100MHz
0x000000002f8daa3e: Clock: gcc_dehr_clk Requested State = Enable. Actual State = 1
0x000000002f8daa8f: Clock: gcc_dehr_clk Requested State = Disable. Actual State = 0
0x000000002f8db0ed: Clock: gcc_bimc_clk Requested State = Disable. Actual State = 0
0x000000002f8db111: Clock Processor Collapse: Done
0x000000002f8dbb51: Clock Processor Collapse: Enter
0x000000002f8dbba1: Clock: gcc_spmi_ser_clk Requested State = Disable. Actual State = 0
0x000000002f8dbbe5: Clock: gcc_imem_axi_clk Requested State = Disable. Actual State = 0
0x000000002f8dbc29: Clock: gcc_msg_ram_ahb_clk Requested State = Disable. Actual State = 0
0x000000002f8dbcd7: Clock: gcc_apss_tcu_async_clk Frequency = 19MHz
0x000000002f8dbe84: Clock PLL: GPLL0 Status = Disable
0x000000002f8dc025: Clock: gcc_rpm_proc_fclk Frequency = 19MHz
0x000000002f8dc1d5: Clock Processor Collapse: Done
0x000000002f8dc302: deep_sleep_enter_complete: (mode: "VDD Minimization") →System is in Low power mode

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 other processors, so using NPADump.cmm to retrieve full NPA system state is suggested
  - NPADump.cmm – Retrieves the resource and client names
- The following is the snippet to view the XO votes from subsystems:

```
npa_resource (name: "/xo/cxo") (handle: 0x95890) (units: Enable)
(resource max: 1) (active max: 1) (active state: 1)  (active
headroom: 0) (request state: 1)
npa_client (name: WCSS) (handle: 0x993F0) (resource: 0x95890)
(type: NPA_CLIENT_LIMIT_MAX) (request: 1)
npa_client (name: WCSS) (handle: 0x993B0) (resource: 0x95890)
(type: NPA_CLIENT_REQUIRED) (request: 1)
npa_client (name: MPSS) (handle: 0x959A8) (resource: 0x95890)
(type: NPA_CLIENT_LIMIT_MAX) (request: 1)
npa_client (name: MPSS) (handle: 0x959E0) (resource: 0x95890)
(type: NPA_CLIENT_REQUIRED) (request: 1)
npa_client (name: APSS) (handle: 0x95B30) (resource: 0x95890)
(type: NPA_CLIENT_REQUIRED) (request: 1)
npa_change_event (name: sleep) (handle: 0x96020) (resource:
0x95890)
end npa_resource (handle: 0x95890)
```

Hansei RAM Dump Parser

- This is a tool to parse debug information out of the RAM dump. It generates RPM logs, NPA logs, master status, resource states, etc.
- Installation
  1. Install Python 2.7.x (not 2.6.x).
     Check version – python –V.
  2. Install pyelftools library that supports the ARM compiler. If the mainline version does not work, use:
     https://bitbucket.org/pplesnar/pyelftools-pp
     Install command – python setup.py install
- Hansei script release
  1. Released since RPM 100
  2. Location – rpm_proc\core\bsp\rpm\scripts\hansei\

Hansei RAM Dump Parser (cont.)

- Usage

  hansei.py [-h] --elf rpm.elf --output <OutPut Path> dumpfile <dump path>

  Dump path should contain rpm_code_ram.bin rpm_data_ram.bin rpm_msg_ram.bin
- Example:

  python hansei.py –elf rpm.elf -o . dumpfile <dump path containing rpm_code_ram.bin rpm_data_ram.bin rpm_msg_ram.bin>
- Output:
  - rpm-summary.txt – Contains general information about the health of the RPM, including the Core Dump state and various fault information
  - rpm-log.txt – The post processed “RPM external log”
  - npa-dump.txt – The standard NPA dump format, though without the (inaccurate) timestamps
  - ee-status.txt – Contains information about the subsystems (and their cores) that are active or sleeping
  - reqs_by_master/* – A folder containing a file for each execution environment, detailing all the current requests that EE has in place with the RPM
  - reqs_by_resource/* – A folder structure containing a folder for each of the resource “types” registered with the RPM server, and under that folder, a file containing all of the requests to each resource of that type

Key Debugging Structures and Variables

- Information has been reorganized in the RPM structure

```
typedef struct
{
unsigned num_ees; // The number of EEData structures.
unsigned supported_classes; // The number of ResourceClassData
structures.
unsigned supported_resources; // The number of ResourceData structures.
EEData *ees;
ResourceClassData *classes; // Stored in order of registration.
ResourceData *resources; // Indexed by perfect hash lookup.
} SystemData;
extern SystemData * const rpm;
```

- SPM state for each master
  rpm.ees[master_id].subsystem_status

- Sleep counts
  sleep_stats[0] / sleep_stats[1]

- How to disable deep sleep
  Set sleep_allow_low_power_modes = FALSE

Debugging Railways

- Railway configuration
  Railway_config.c
- Check rail votes Railway.rail_state[x]; x= rail# (mx=0/cx=1)
  - Follow voter_list_head and voter_link to see all voters
  - Voter ID
    master number (0: APPS, 1: MODEM, 2: RIVA)

0x0 (0)| (
current_active = (mode = RAILWAY_SUPER_TURBO, microvolts = 1287500), →Represents active corner mode and it's voltage
| unconstrained_target = (mode = RAILWAY_SUPER_TURBO, microvolts = 1287500),
| constrained_target = (mode = RAILWAY_SUPER_TURBO, microvolts = 1287500),
| sm_mode_voter = 0x0,
| cbs = (((callback_fn = 0xA8A5, callback_cookie = 0x0), (callback_fn = 0x0, callback_cookie = 0
| sleep_handle = 0x00095C48,
| voter_list_head = 0x00098F00 -> (
| voltage_corner = RAILWAY_SUPER_TURBO, -→Represents subsystem Corner Vote
| active_floor = RAILWAY_NO_REQUEST,
| explicit_voltage_in_uv = 1287500, -→Subsystem requested voltage.
| suppressible = FALSE,
| id = 1, ----------→Represents the subsystem Id
| rail = 0,
| sw_enable = TRUE ,
| voter_link = 0x000988A8),
| corner_uvs = (0, 650000, 1050000, 1050000, 1150000, 1287500, 1287500, 1287500, 1287500),
| voltage_settling_cb = 0x0,
| voltage_settling_cb_cookie = 0x0),

RPM T32 Scripts for RPM Crash Debugging

- The scripts are located at <rpm build id>/rpm_proc/core/bsp/rpm/scripts
  - **rpm_load_dump.cmm –** Script used to load the binaries and restore the state of a debug session**.**
  - **rpm_m3_unstack.cmm –** Script used to examine a pre-empted process

    On examining a dead RPM, it may be that an executing process was preempted (by a software fault handler or interrupt). Since the core dump generally occurs at the top of the fault handler, running this script immediately after rpm_restore_core.cmm will usually place you at the faulting instruction.
  - **rpm_parse_faults.cmm –** Script provides the details about exceptions that have occured. Refer to the file exception.c for arm exceptions, located at /rpm_proc/core/bsp/rpm/src

RPM T32 Scripts for RPM Crash Debugging (cont.)

- Upon RPM crash, the variable rpm_core_dump is used to dump rpm core context

```
rpm_core_dump = (
cookie = 0x0BAFF1ED,
dumped_at = 0x0AB8341C,
registers = (
  gp_regs = (0x0009D7E4, 0x00222D08, 0x4C, 0x10001000, 0x6044C000, 0x60000000, 0x0, 0x60000000, 0x0, 0x03E8, 0x00061A80, 0x00
  SP_main = 0x0009D7C0,
  SP_process = 0x0,
  LR = 0x9B13,
  PC = 0x61,
  xPSR = 0x60000006,   ----------→First byte which provides information about the kind of exception that occurred
  PRIMASK = 0x0,
  FAULTMASK = 0x1,
  BASEPRI = 0x10,
  CONTROL = 0x0),
fault_detail = (
  SystemHandlerCtrlAndState = 0x00070008,
  ConfigurableFaultStatus = 0x00010000,
  HardFaultStatus = 0x0,
  DebugFaultStatus = 0x0,
  MemManageAddress = 0xE000ED34,  → Provides the address at which invalid memory access exception occurred      BusFaultAddress =
0xE000ED38,        →Provides the address at which BusFault exception occurred
  AuxFaultStatus = 0x0),
```

- Refer to the exception.c (/rpm_proc/core/bsp/rpm/src) for exception types
- The above example 0x06 represents the Usage Fault exception

References

Ref.	Document	
Qualcomm Technologies		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*Resource Power Manager (RPM.AF) User Guide*	80-N6955-1

Questions?

https://support.cdmatech.com