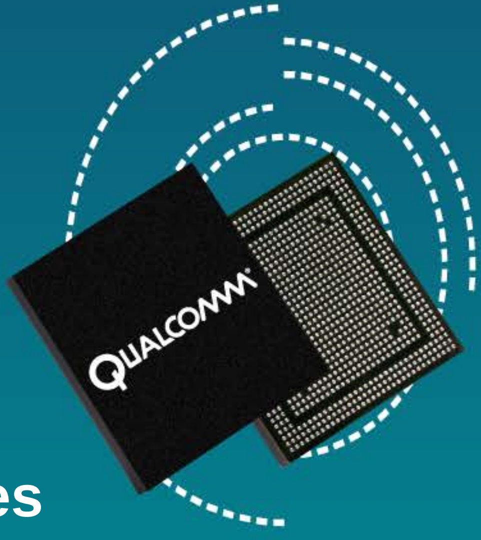
Resource Power Manager (RPM.BF) Features

80-NF597-1 E



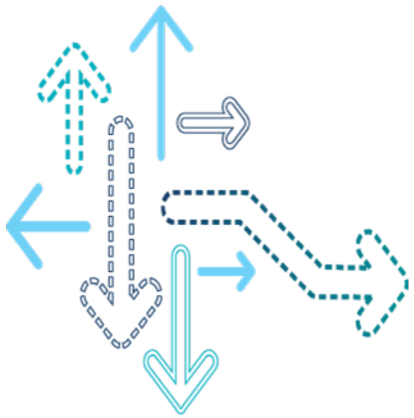
Revision History

Revision	Date	Description
A	Apr 2013	Initial release
B	May 2013	Updated title and SWEvent Overview slide; added Introduction section
C	Mar 2014	Updated title, Supported ASICs slide, and RPM Memory Map slide
D	Apr 2014	Updated title and Supported ASICs slide
E	Apr 2014	Updated Supported ASICs slide

Contents

- Introduction
- Cortex-M3 Core
- Nested Vectored Interrupt Controller (NVIC)
- Messaging v2.0
- Railway
- On-Chip Memory (OCMEM)
- RapidBridge Core Power Reduction (RBCPR)
- Software Events
- References
- Questions?

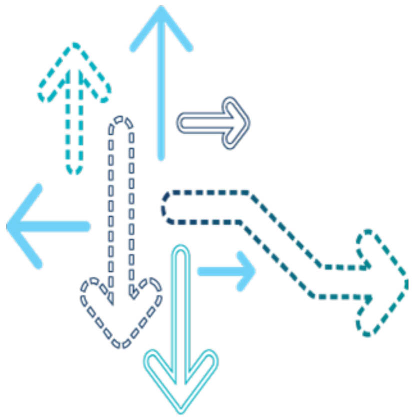
Introduction



Supported ASICs

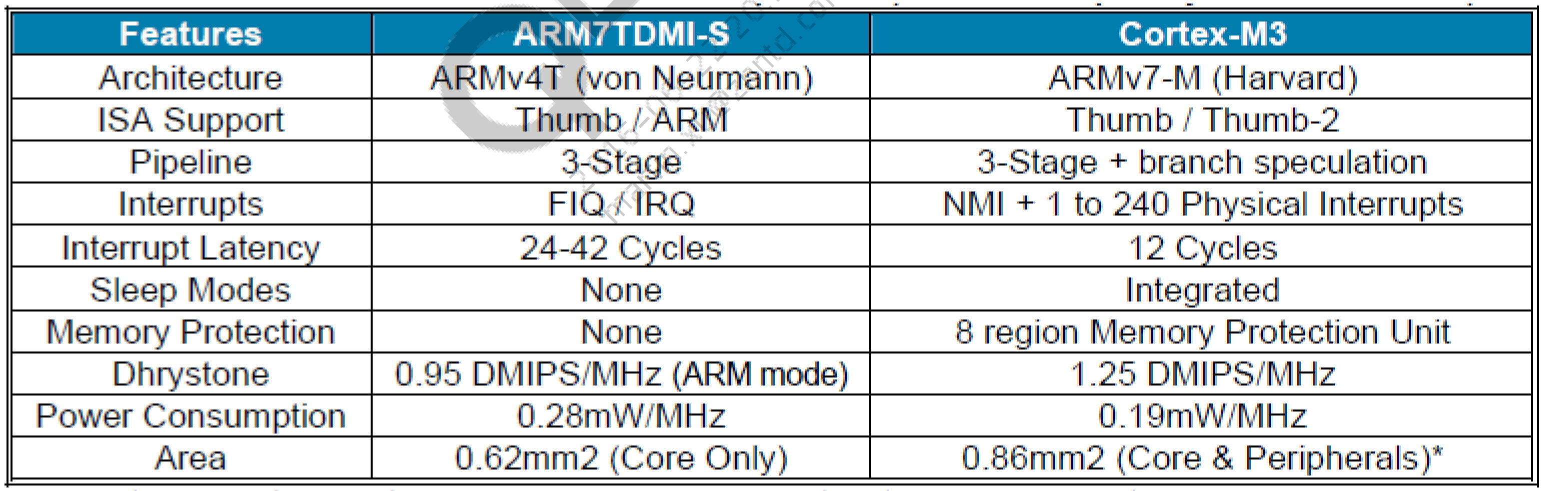
- This presentation applies to chipsets that use the RPM.BF subsystem.

Cortex-M3 Core



Cortex-M3 Core

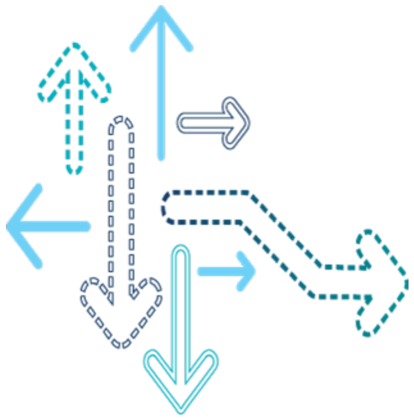
- RPM uses an ARM® Cortex-M3 processor. Compared to the ARM7TDMI® processor in MSM8960:
  - Higher MIPS and lower power
  - Harvard architecture with simultaneous instruction fetch with data load/store
  - Low interrupt latency
  - Thumb2, hardware multiply and divide



Features	ARM7TDMI-S	Cortex-M3
Architecture	ARMv4T (von Neumann)	ARMv7-M (Harvard)
ISA Support	Thumb / ARM	Thumb / Thumb-2
Pipeline	3-Stage	3-Stage + branch speculation
Interrupts	FIQ / IRQ	NMI + 1 to 240 Physical Interrupts
Interrupt Latency	24-42 Cycles	12 Cycles
Sleep Modes	None	Integrated
Memory Protection	None	8 region Memory Protection Unit
Dhrystone	0.95 DMIPS/MHz (ARM mode)	1.25 DMIPS/MHz
Power Consumption	0.28mW/MHz	0.19mW/MHz
Area	0.62mm2 (Core Only)	0.86mm2 (Core & Peripherals)*

* Does not include optional system peripherals (MPU & ETM) or integration level components

Nested Vectored Interrupt Controller (NVIC)



NVIC Overview

- The Nested Vectored Interrupt Controller (NVIC) is an integral part of the Cortex-M3 processor
  - Provides outstanding interrupt latency (12 cycles vs 24 to 42 cycles on ARM7TDMI)
  - Can be configured to 1 and 240 physical interrupts with up to 256 levels of priority (RPM uses 16+64 interrupts)

NVIC Vector Table

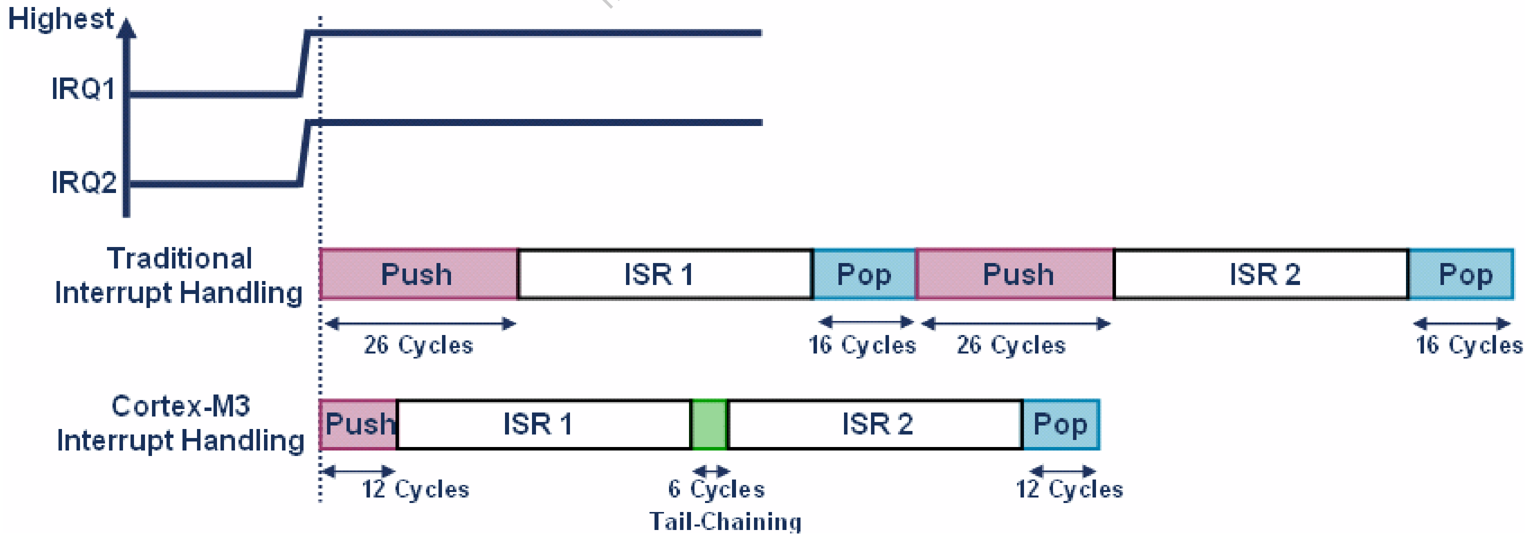
- Uses a relocatable vector table that contains the addresses of the interrupt handlers
- Fetches the address from vector table through the instruction bus interface
- Vector table is located at address 0 at reset, but can be relocated

M3 Exception Model

- Cortex-M3 migrates from the banked register exception model of the ARM7 to a stack-based exception model.
- On exception, PC, PSR, LR, and R0-R3, R12 are pushed onto the stack.
- Data bus stacks the registers while the instruction bus reads the vector table and fetches the first instruction.
- The stacked registers are automatically restored after execution of ISR.
- By handling the stack operations in hardware, the Cortex-M3 processor removes the need to write assembler wrappers to perform stack manipulation, making apps development significantly easier.

NVIC Tail-Chaining

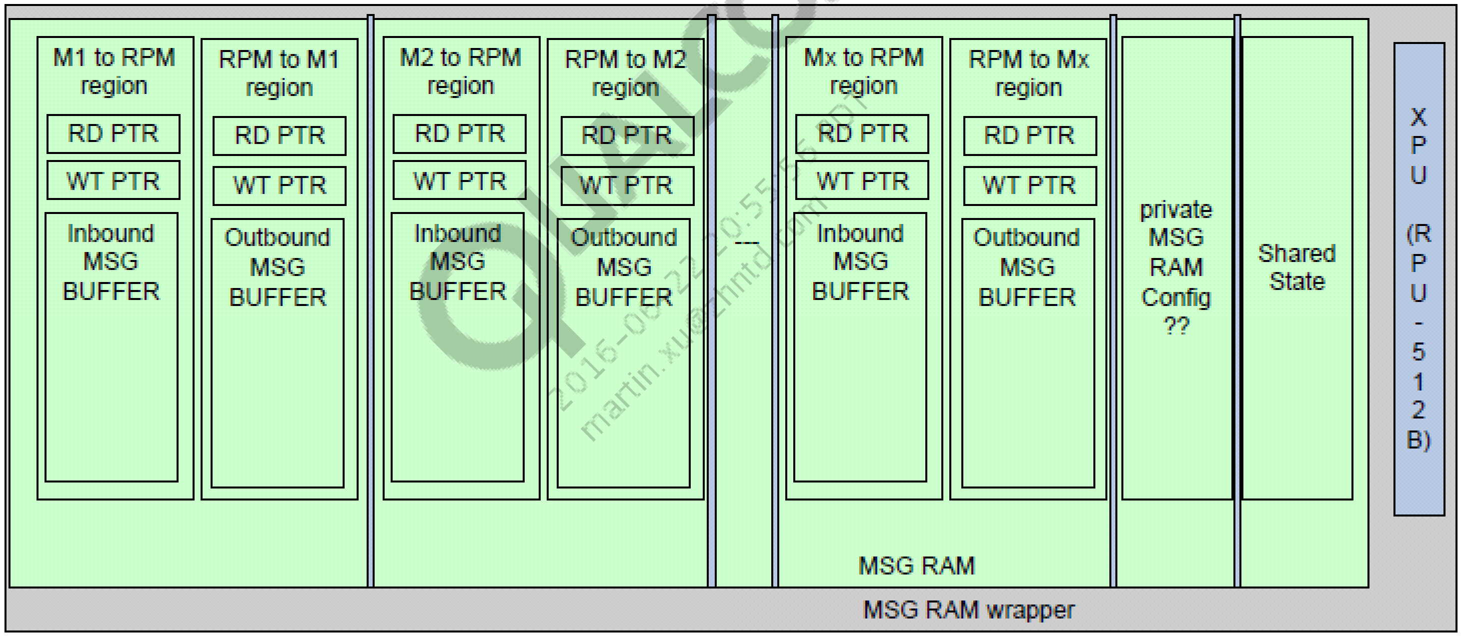
- In the case of back-to-back interrupts:
  - Traditional systems repeat the state save and restore cycle twice, resulting in higher latency. Cortex-M3 implements tail-chaining in NVIC hardware to simplify the handling.
  - Tail-chaining achieves much lower latency by replacing serial stack pop and push actions that normally take over 30 clock cycles with a simple 6-cycle instruction fetch.
  - The processor state is automatically saved on interrupt entry, and restored on interrupt exit, in fewer cycles than a software implementation, significantly enhancing system performance.



Messaging v2.0

Message RAM Partition

- Memory is partitioned and protected for each processor/execution environment.



RPM Memory Map

	RPM.AF	RPM.BF
RPM_CODE_START_ADDR	0x20000	0x100000
RPM_CODE_SIZE	0x24000 (MSM), 0x28000 (APQ)	0x20000
RPM_DATA_START_ADDR	NA	0x190000
RPM_DATA_SIZE	NA	0x10000
RPM_MSG_RAM_ADDR	0x108000	0xFC428000
RPM_MSG_RAM_SIZE	0x6000	0x4000
RPM_SWVERSION_ADDR	0x108008	STR at 0x190040

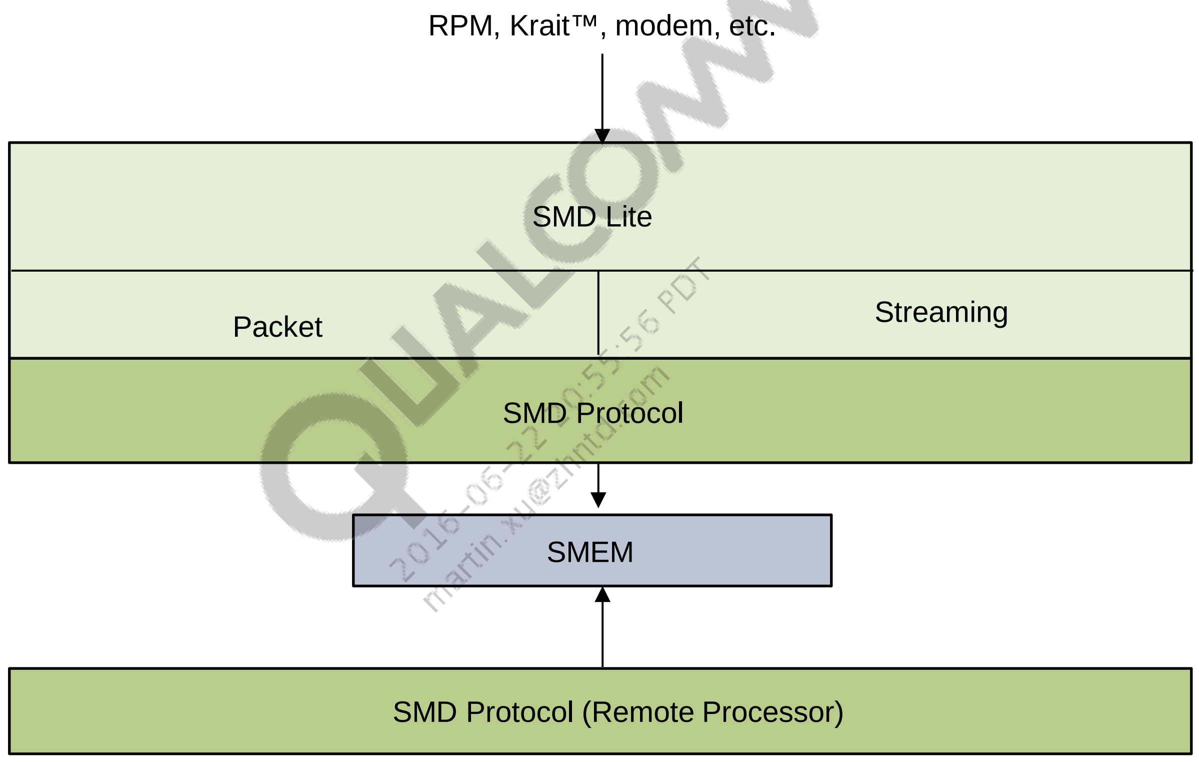
Transport – Register Emulation vs SMD Lite

- Limitation of register emulation
  - Model creates unnecessary inter-target work
  - Performance is undesirable
  - Resource usage is not optimal
- Shared Memory Driver (SMD) Lite is a message-oriented FIFO transport
  - It is a standard, well-understood QTI technology
  - It allows for symmetric full-duplex communication

SMD Lite

- SMD Lite is a lightweight implementation of SMD
  - Taskless, giving clients the control to read, write, and process data in their own context (priority)
  - Provides simple and intuitive APIs
    - smdl_read, smdl_write, etc.
    - Client pushes a nonblocking read or write and returns right away vs SMD buffer pulls from clients
  - Lets clients manage their own buffers directly
  - Fully interoperable with SMD APIs
    - An SMD connection can have SMD on one end and SMD Lite on the other
- SMD is only a transport, still need to layer a protocol on top

SMD APIs and Layers



Message Structure – Key Value Pair (KVP)

- SMD Lite provides messages and formats them as KVP.
- KVP is 2+n words of data:
  - One word for key – Generally used as a 4-byte string (uv\0\0, clk\0, etc.)
  - One word for length – Describes how many bytes follow
  - Blob of data
- This structuring is repeated recursively to build full messages.

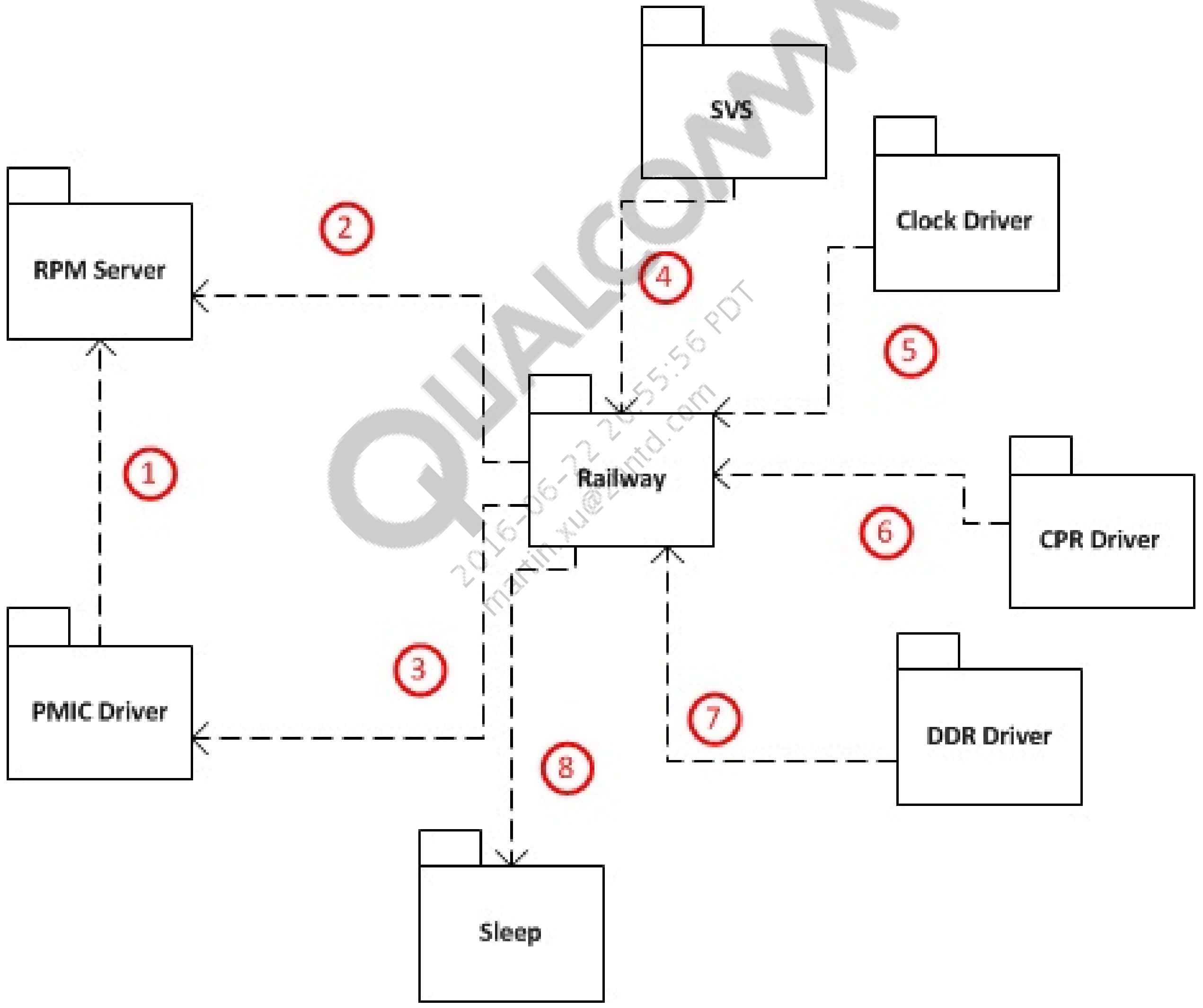
Message Structure – Key Value Pair (KVP) (cont.)

- Example of a request to change LDO3’s voltage and current

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

Railway

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. This overrides the registration done by the PMIC driver—the PMIC driver receives no notification for votes on these resources. For these resources, the RPM server routes both the xlate and apply functions to the overrider, i.e., railway.
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.)

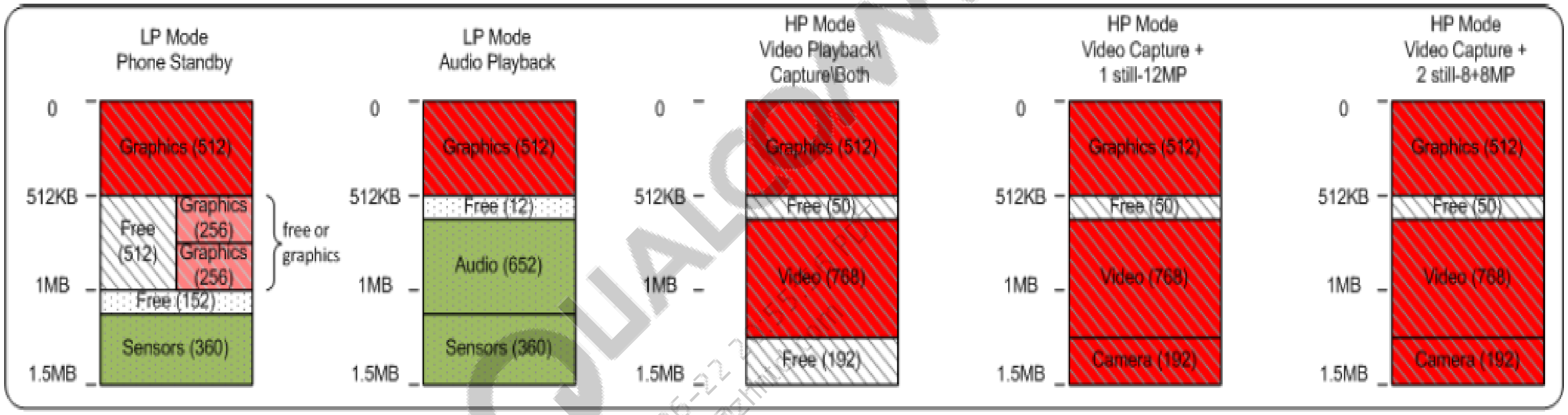
5. Clock driver registers with Railway for notifications on when Vdd_Cx or Vdd_Gfx change. It also votes on /pmic/client/clk_regime_dig and /pmic/client/gfx nodes for when it wants to change voltages on Vdd_Cx and Vdd_Gfx respectively; these nodes are implemented by the railway component.
6. CPR registers 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 nonsuppressible votes for Vdd_Cx, which would prevent Vdd minimization.

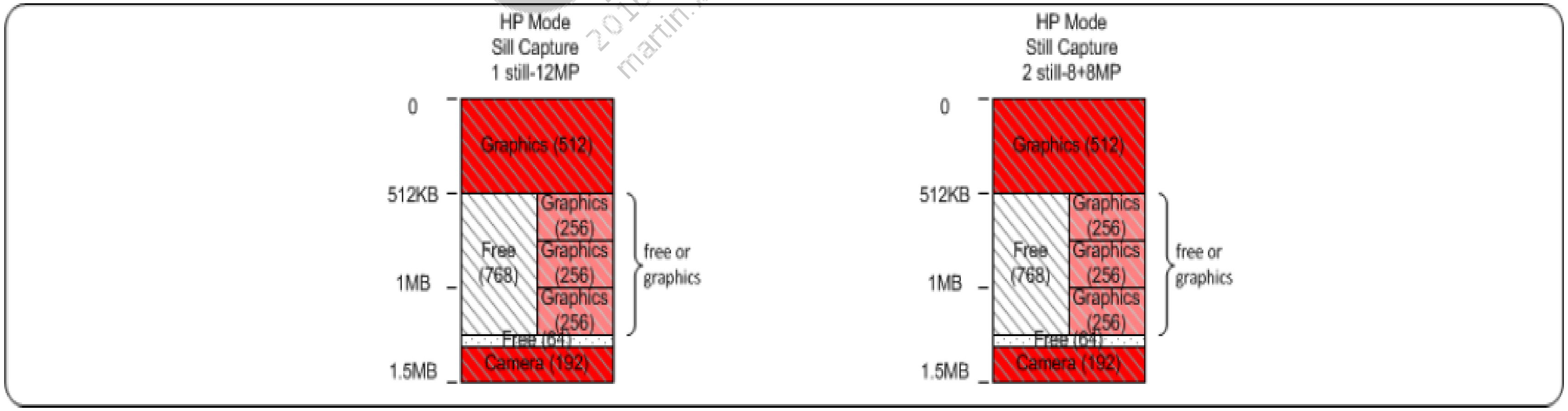
On-Chip Memory (OCMEM)

OCMEM

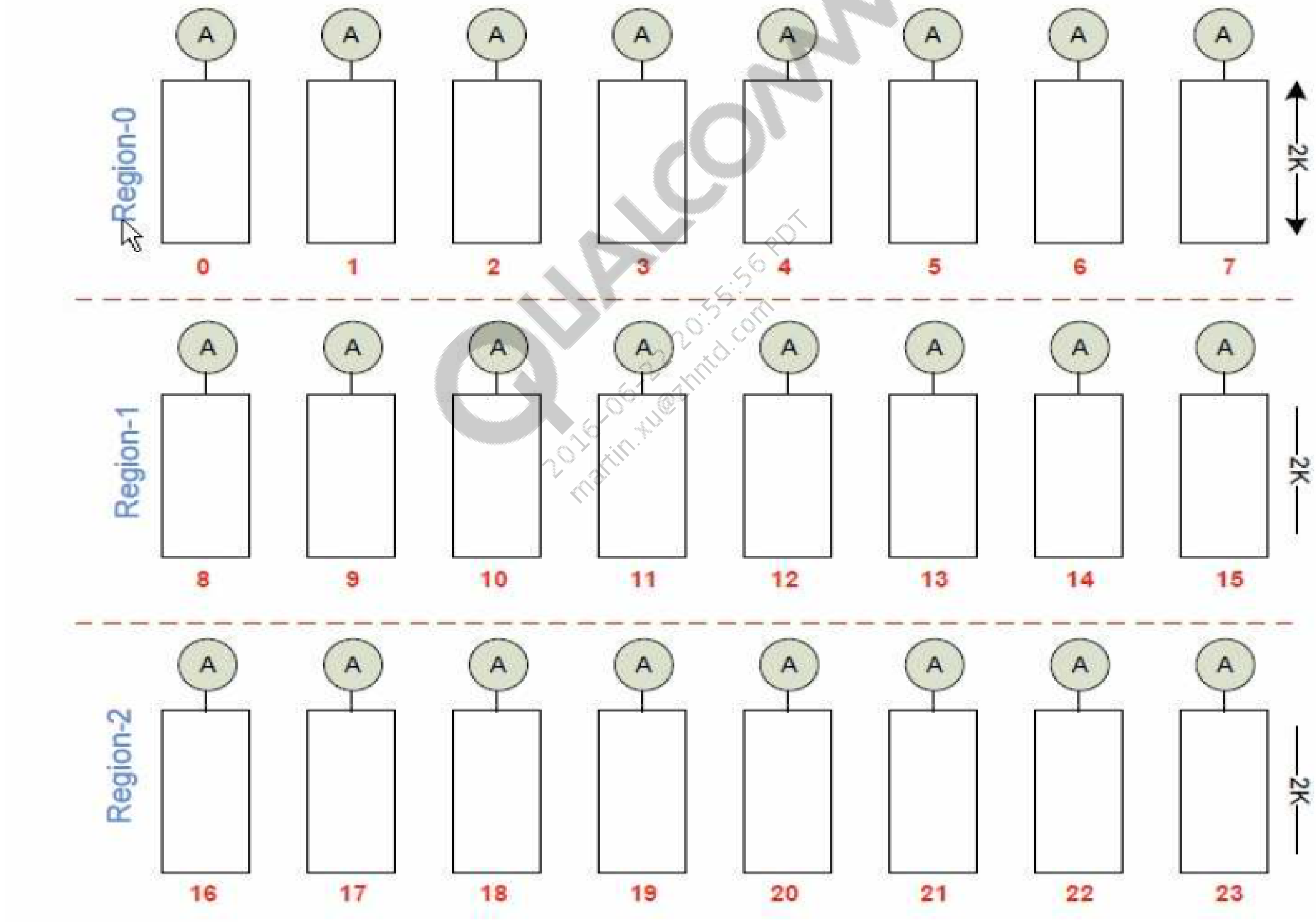
- OCMEM driver in HLOS is the centralized entity that maintains allocation states and corresponding power states
- Sensors running in OCMEM
- Background low-power use case cannot involve the apps processor
- OCMEM to go into retention similar to DDR
- Both apps and sensors vote to RPM, RPM determines aggregated power state

OCMEM Allocation





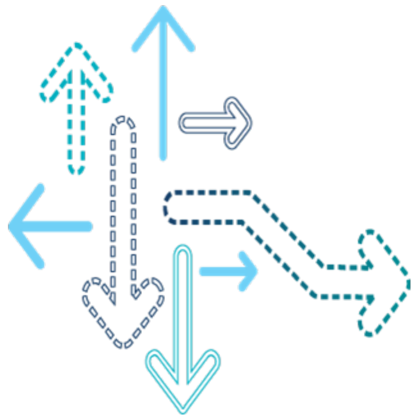
OCMEM Hardware Overview



OCMEM Driver Implementation

- OCMEM is comprised of three regions with 8 memory macros
- Resource type – 0x706d636f
  - This is 'ocmp' in little endian, to be added to rpm_resource_type as RPM_OCMEM_POWER_REQ in rpm.h
- Resource IDs – 0, 1, 2 (integers)
  - These refer to the three regions on *OCMEM Hardware Overview* slide
- Keys accepted by each resource – 0, 1, 2, 3
  - 0 = banks 0 and 4, 1 = banks 1 and 5, etc. due to the hardware requirement that the macros are controlled in pairs
- Data accepted by each key – 4-byte integer, values are enum
  - { OFF = 0, RETENTION =1, ACTIVE = 2 }

RapidBridge Core Power Reduction (RBCPR)

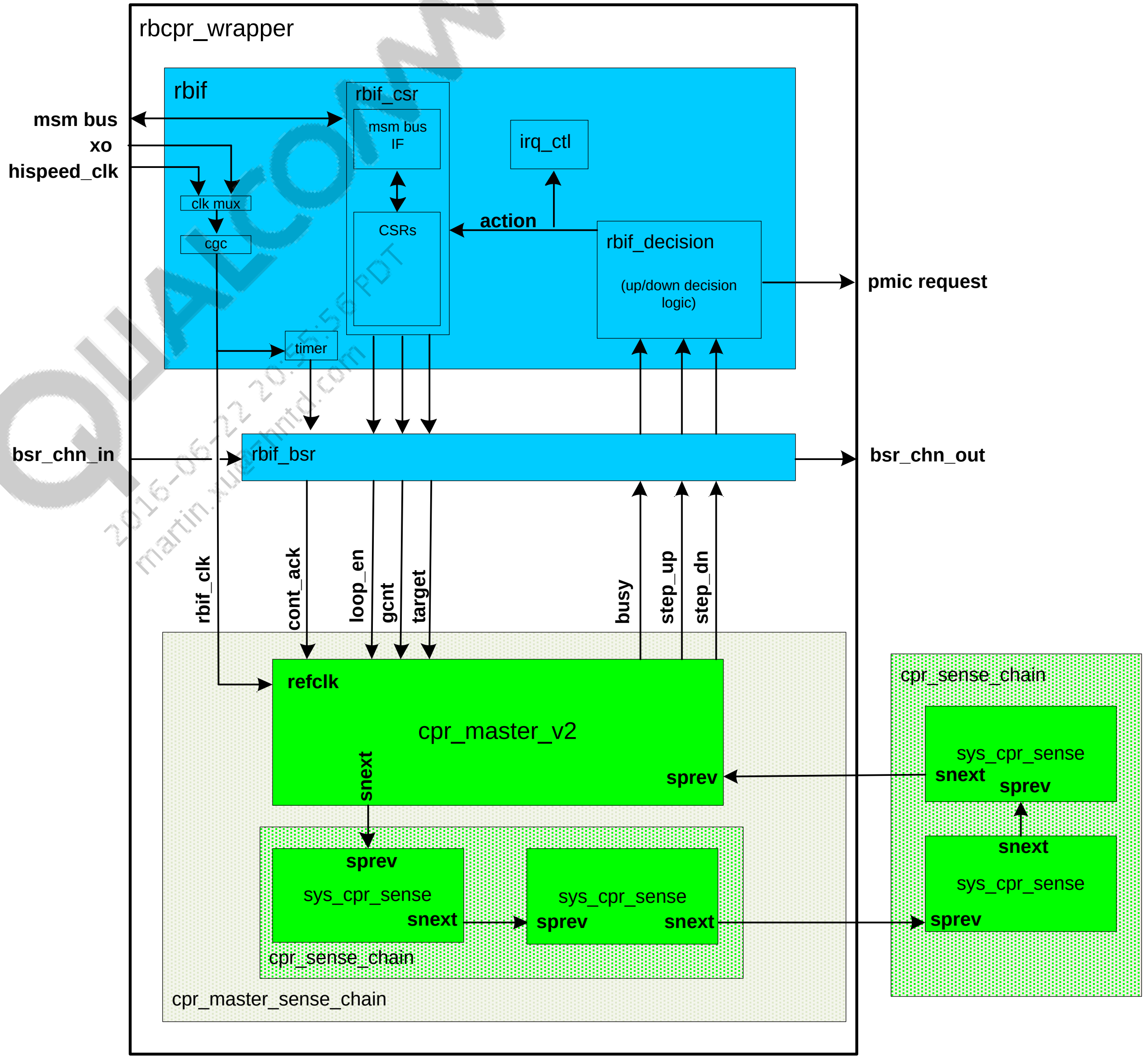


RBCPR Overview

- RBCPR technology provides a feedback loop to optimize the voltage setting.
- Two main use cases of RBCPR:
  - Setup – When the voltage corner is changed and the RBCPR configuration must be changed
  - Adjustment – When the RBCPR hardware senses a voltage change is necessary and the RBCRP software adjusts the voltage
- RBCPR is expected to apply to the following domains and be controlled by the indicated execution environment:
  - VDD Dig – Controlled by RPM
  - VDD GPU – Controlled by RPM

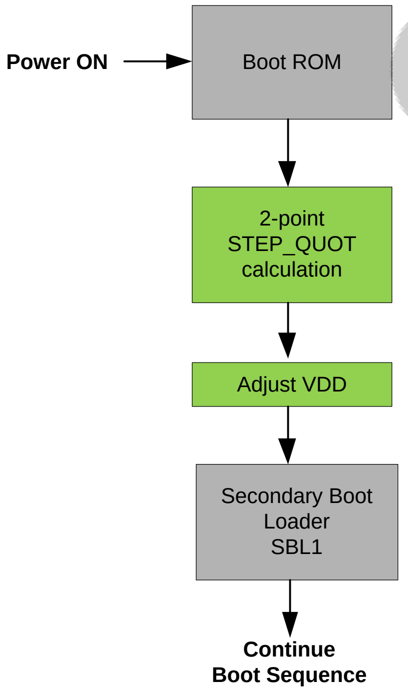
RBCPR Block Diagram

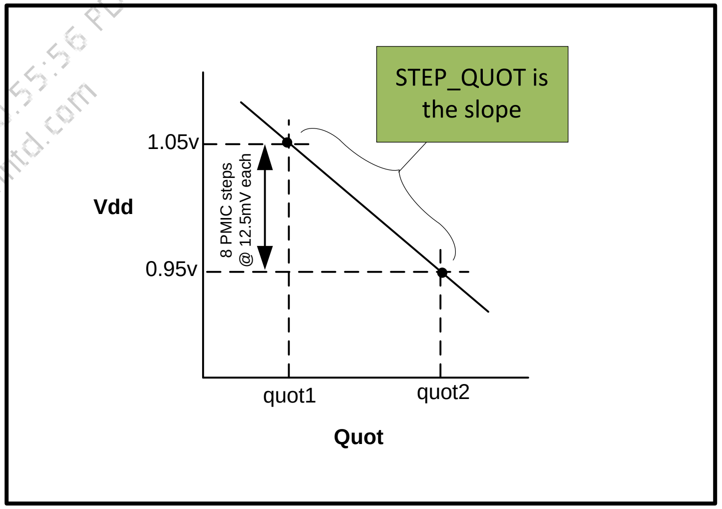
- The RBCPR core from Rapid Bridge consists of one master and a number of sensors, shown in the figure in green.
- There is an additional QTI wrapper logic called rbif, shown in the figure in blue.



Two-Point STEP_QUOT Calculation

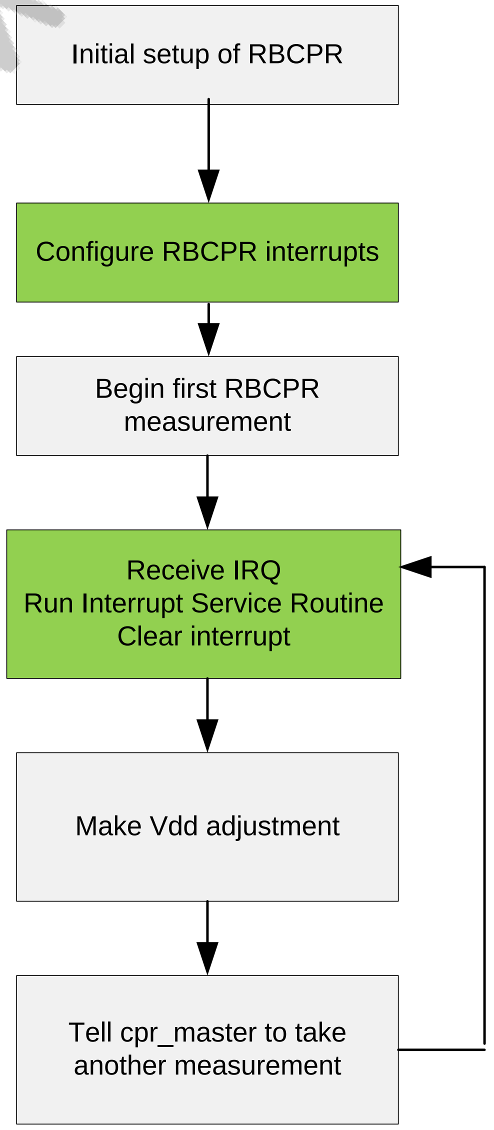
- STEP_QUOT is how many QUOT units per PMIC step
- STEP_QUOT = (quot2 - quot1)/8
- 8 = (1.05 V - 0.95 V)/0.0125 V





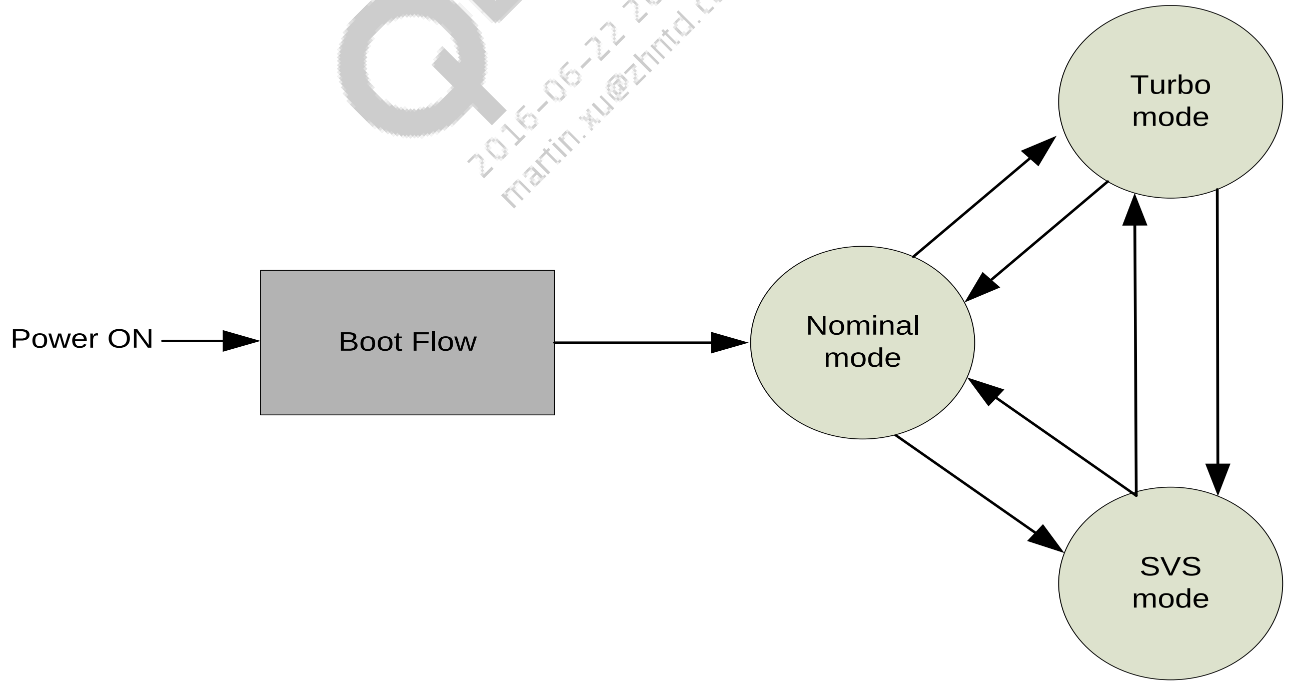
Adjustment

1. Enable CPR interrupts (up_flag_en/ down_flag_en/rbcpr_done_en).
2. Receive CPR interrupt, look at RBCPR_STATUS and make PMIC adjustment.
   a. If step_up is 1, increase PMIC VDD by one step.
   b. If step_down is 1, decrease PMIC VDD by one step.
3. Tell CPR to take another RBCPR measurement (write to RBCPR_CONT_ACK_CMD or BCPR_CONT_NACK_CMD).


Initial setup of RBCPR
Configure RBCPR interrupts
Begin first RBCPR measurement
Receive IRQ
Run Interrupt Service Routine
Clear interrupt
Make Vdd adjustment
Tell cpr_master to take another measurement

RPM Mode (Corner)

- RPM transitions from one mode to another
  - Nominal
  - Turbo
  - SVS
- CPR gets notified of the mode transition and changes its configuration (gcnt/target pairs)



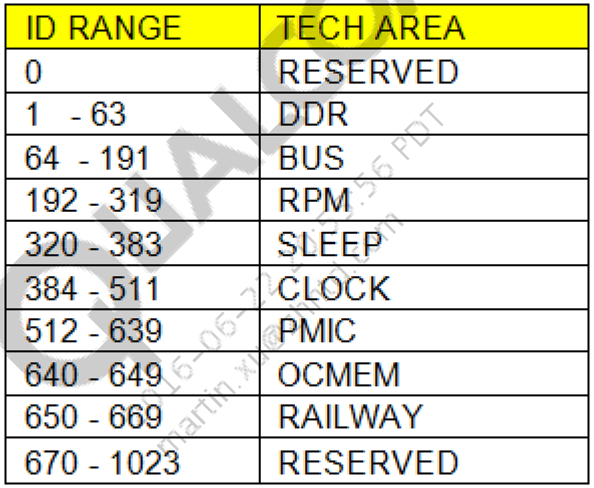
Software Events

SWEvent Overview

- Software events are used to replace RPM log event in earlier chipsets
- Default output is the RAM, which can be changed to QDSS

SWEvent Range Table

- The enum is defined in core/api/debugtrace/tracer_event_ids.h



ID RANGE	TECH AREA
0	RESERVED
1 - 63	DDR
64 - 191	BUS
192 - 319	RPM
320 - 383	SLEEP
384 - 511	CLOCK
512 - 639	PMIC
640 - 649	OCMEM
650 - 669	RAILWAY
670 - 1023	RESERVED

Adding RPM Software Events

1. Add new SWEvent to tech area SConscript.
   - Add the new SWEvent to the bottom of the list of SWEvents in the SConscript, but before the last event.
   - The format is ['NEW_SWEVENT', 'description of new software event: param1 %d'].
   - The following is an example of adding RPM_MASTER_SET_TRANSITION_COMPLETE to core/power/rpm/build/Sconscript:

```
if 'USES_QDSS_SWE' in env:
  QDSS_IMG = ['QDSS_EN_IMG']
  events = [['RPM_BOOT_STARTED=192','rpm boot started'],
     ['RPM_BOOT_FINISHED','rpm boot finished'],
     ['RPM_BRINGUP_REQ','rpm_bringup_req: (master %d) (core %d)'],
     ['RPM_BRINGUP_ACK','rpm_bringup_ack: (master %d) (core %d)'],
     ['RPM_SHUTDOWN_REQ','rpm_shutdown_req: (master %d) (core %d)'],
     ['RPM_SHUTDOWN_ACK','rpm_shutdown_ack: (master %d) (core %d)'],
     ['RPM_TRANS_QUEUED','rpm_trans_queued: (master %d) (status %d) (deadline: %d)'],
     ['RPM_MASTER_SET_TRANS','rpm_master_set_trans:(master %d)(fromSet %d)(toSet: %d)'],
     ['RPM_MASTER_SET_TRANS_COMPLETE','rpm_set_trans_complete: (master %d)'],
     ['RPM_LAST=319','rpm last placeholder'],
  ]
  env.AddSWEInfo(QDSS_IMG, events)
```

Adding RPM SWEvents (cont.)

2. Add SWEVENT calls for new SConscript.
   - Example

```
#include "swevent.h"
...
SWEVENT(RPM_MASTER_SET_TRANS_COMPLETE, master_id);
```

3. Add SWEVENT RAM postparsing.
   - Add parsing details to core/power/rpm/debug/scripts/<tech area>_parser.py
   - ID is a hex value which matches the swevent ID. RPM_MASTER_SET_TRANS_COMPLETE = 200 = 0xc8.
   - The class should return a string describing the SWEvent.
   - Data is parseable with various functions, described in core/power/rpm/debug/scripts/ target_data.py. The following is an example of adding a parsing class for RPM_MASTER_SET_TRANSITION_COMPLETE:

```
class RPMTransitionComplete:
   __metaclass__ = Parser
   id = 0xC8
   def parse(self, data):
     return 'rpm_master_set_transition_complete (master: %s)' % get_master_name(data[0])
```

References

Ref.	Document	
Qualcomm Technologies		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*Resource Power Manager (RPM.BF) User Guide*	80-NA157-15

Questions?

https://support.cdmatech.com