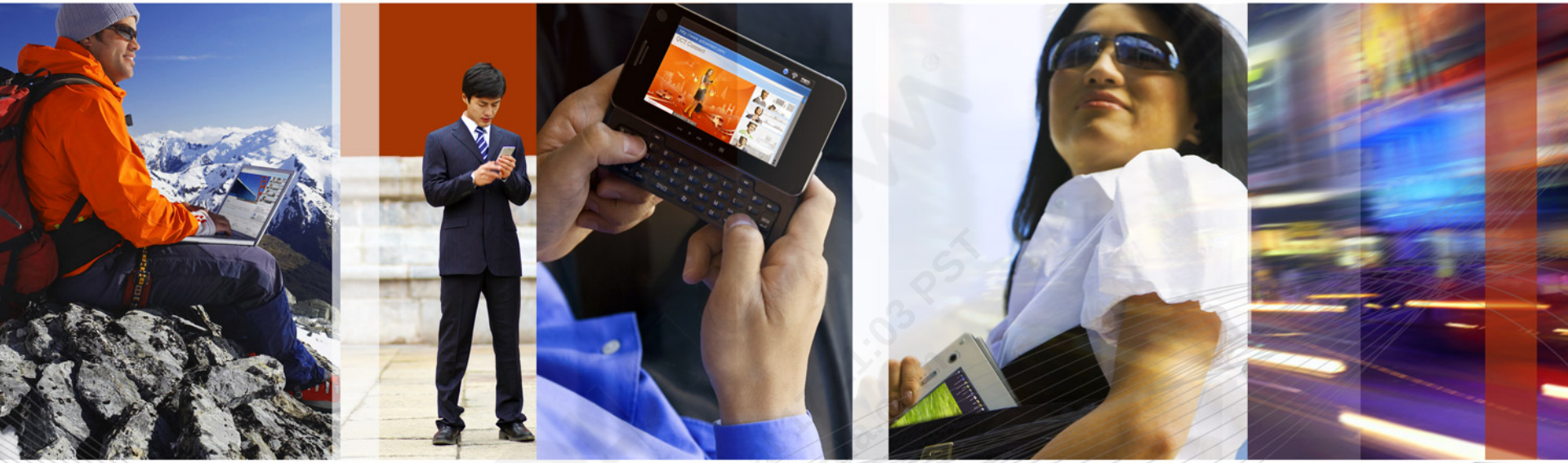


Qualcomm ATHEROS

REDEFINING MOBILITY

QCA WCN36x0 WLAN Power Optimization Guide

80-Y0513-3 Rev. F

Qualcomm Atheros, Inc.
1700 Technology Drive
San Jose, CA 95110
U.S.A.

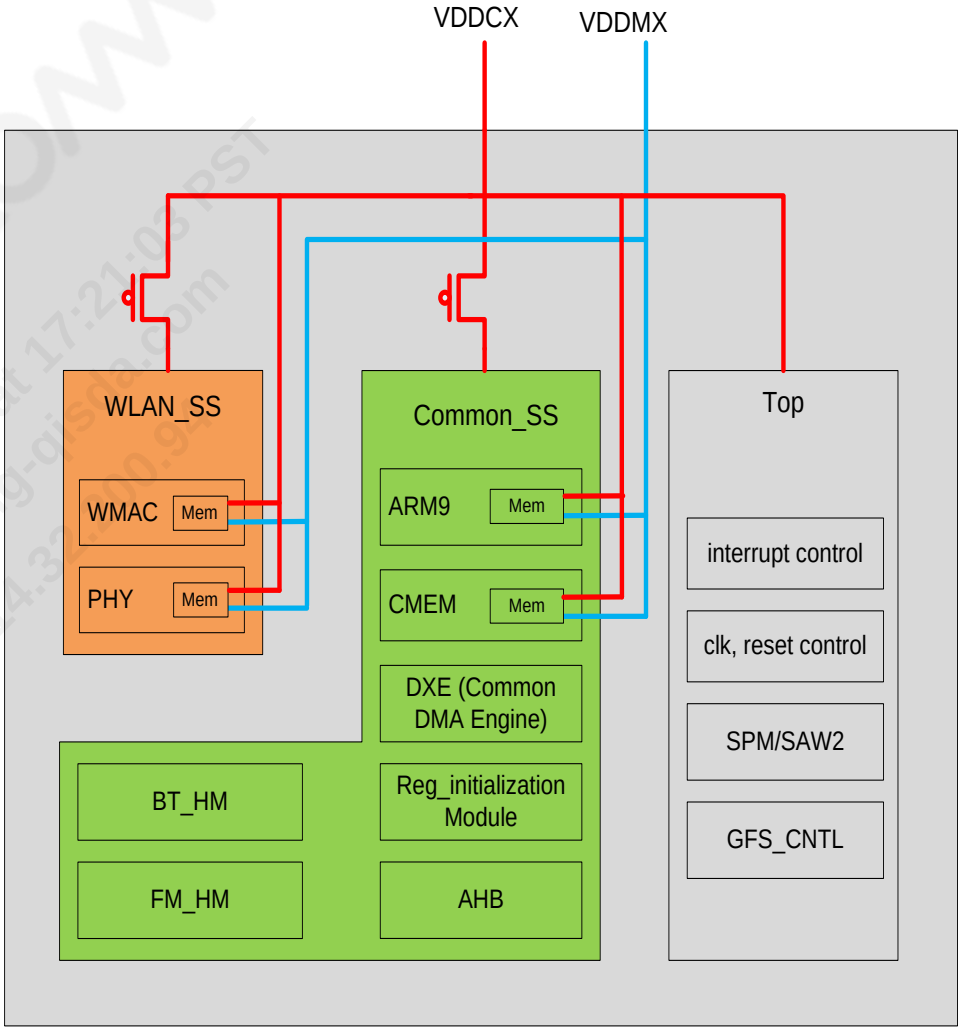
Revision History

Revision	Date	Description
A	Aug 2012	Initial release
B	Oct 2012	Updated WCN-SS Power Rails and Headswitches figure; expanded on description for Modulated DTIM and missing beacon; updated testing
C	Oct 2012	Added WCNSS configuration guide; added references to WCNSS for Android and Windows
D	Dec 2012	Added PNO slides, MSM8974 WCN-SS Power Optimizations
E	Jan 2013	Updated Packet Filtering and ARP Offload and Driver Implementation sections
F	Feb 2013	Updated graphics on slides 6, 11, and 28; added information on WPM and descriptions of BET and miss beacon; changed Riva to WCNSS

Table of Contents

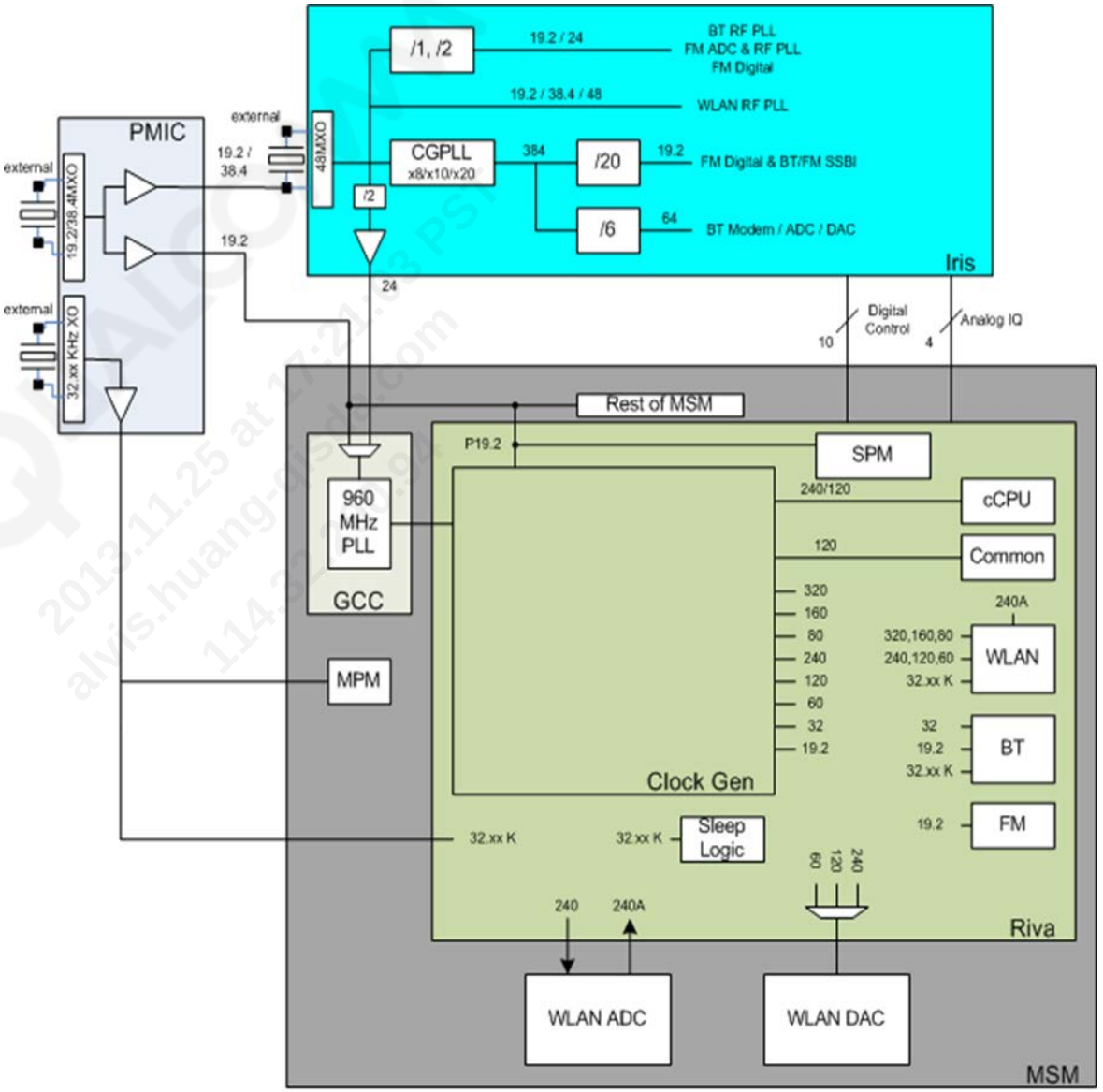
WCN-SS Power Rails and Headswitches

- VDDmx is the memory voltage and powers RAM blocks
- VDDcx is the chip voltage and powers digital (nonmemory) blocks
- RPM controls the voltage of VDDcx and VDDmx to satisfy the dynamic requirements of all subsystems that share access to these rails
- Three blocks with isolated power control – WLAN, Common, and Top (Always-On); headswitches control power to WLAN and Common
- WLAN software controls the headswitch to WLAN block; shuts it down whenever WLAN is idle
- Subsystem Power Manager (SPM) hardware block controls the headswitch to common block



WCN36x0 Clock

- WCN36x0 has an optional 48 MHz reference clock that is required for WLAN 5 GHz operation.
- WCNSS has its own 960 MHz PLL which is on the GCC and provides the clock source for much of the WCNSS subsystem. 19.2 MHz clock from the PMIC XO or 24 MHz clock derived from an optional crystal that is attached to Iris - The PLL outputs both a 960 MHz clock and a 480 MHz clock.
- CGPLL is needed for various Bluetooth and FM analog components; it is controlled by Power Manager by Clock Regime driver.



WCN36x0 Clock *(cont.)*

- 48MHz XO on Iris
  - The 48MHz XO on Iris is an optional crystal that, if present, can supply a 48MHz clock to the WLAN RF. It is needed in order to support WLAN 80MHz mode.
- CGPLL
  - This PLL on Iris provides clocks to various Bluetooth and FM analog components; it will be enabled and disabled by the Power Manager as needed when entering and exiting an active BT or FM mode of operation; Power Manager will control the CGPLL via the Clock Regime driver
- cCPU Clock
  - The ARM9 (cCPU) clock is programmable to run at 240 MHz, 120 MHz, or 60 MHz; it is controlled via the Clock Regime driver; Power Manager will dynamically change the cCPU clock when it is notified of a change in the active wireless technology state
  - In order to achieve MIPS requirement targets, the CoreBSP sleep driver must restore the cCPU clock frequency during wake to the same value it was when going to sleep

WCN-SS HW – Power Rails

- All rails to WCN3660 and WCNSS are controlled by RPM
  - No PMIC driver in WCNSS, i.e., no way for WCNSS to talk directly to PMIC
  - We use NPA Remoting to forward PMIC control via RPM
- PMIC NPA nodes will be provided by the PMIC driver team for each supply, and control will be managed through these NPA nodes
- All rails will be turned ON in the Wake Set from WCNSS and turned OFF in the Sleep Set
  - Including the 2.9V rail for the Transmit PA; very low leakage if PA is not used
- WCN3660 requires that 1.8V SMPS I/O supply must come up before 1.3V
  - All the other supply domains (1.8V LDO, 1.2V and 2.9V) protect their inputs until 1.3V supply comes up
  - Once 1.3V supply comes up, we release isolation by S/W control, assuming S/W knows all the necessary supplies have been up

RAM Leakage Reduction Modes

RAM Mode	Description
ACTIVE	RAM block is being actively used (read/write)
DORMANT	• Standard Cell Power-Collapsed, Memory Periphery Footswitched • RAM contents are retained, but the RAM may not be accessed (no read/write)
OFF	All contents of RAM are lost; leakage current is minimized

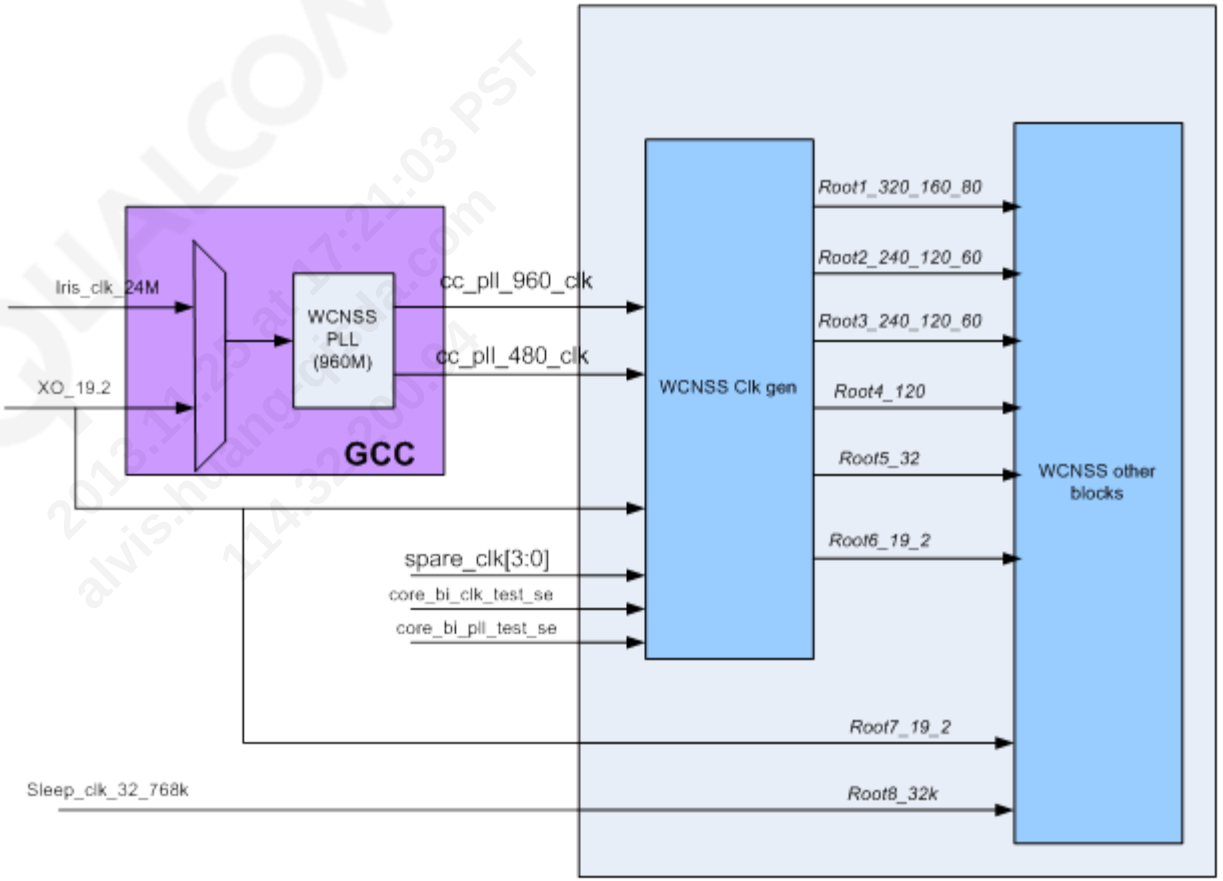
- There are four memory blocks in WCNSS
  - cMEM and Cache on Common_SS
  - WMAC and PHY memories on WLAN_SS

WCN-SS HW – GDHS and RAM Supply Rails

- WCNSS SW will directly control the WLAN GDHS via register writes
- The COMMON GDHS is controlled by SAW2 (SPM); SAW2 is configured by register writes in the Sleep Driver
- TOP is always on, but subject to non-functionality due to VDDmin
  - Note: QGIC is actually in COMMON; requires save and restore of QGIC registers by the Sleep Driver around power collapse
- All RAM blocks support for low-power modes of operation
  - WLAN RAM will be directly controlled by WCNSS SW via register writes
  - COMMON RAM will be controlled by SAW2, which is configured by register writes in the Sleep Driver
    - ACTIVE during wake
    - OFF during sleep (assumes that cMEM and cache contents are always lost during power collapse)

WCN-SS HW – Clocks

- All WCNSS clocks are derived from the 960 MHz PLL in the GCC, except during sleep or the first steps of the wakeup cycle before the PLL is enabled
- For WLAN, the cCPU clock frequency setting is independent of the phy clock setting; any combination can be used, as long as the data rate requirements are met
- For BT, only the PLL generated 32 MHz clock is used for the link controller
- For FM, only the PLL generated 19.2 MHz clock is supported



WCN-SS HW – Clocks *(cont.)*

- PMIC supplies 19.2 MHz reference clock and 32 kHz sleep clock
- IRIS has an optional 48 MHz reference clock that is required for WLAN 5 GHz operation
- 960 MHz PLL output is used to drive the WCNSS clock tree in operational mode
  - Until PLL output is stable, WCNSS is clocked directly off the 19.2 MHz ref clk
- 32 kHz sleep clock drives all the WCNSS sleep logic and low-frequency timers
- CGPLL on IRIS is used to drive modem, ADC/DAC, and WCN3660 digital blocks for BT and FM; controlled by Power Manager SW by sending an SSBI command to IRIS

WCN-SS HW – Clocks *(cont.)*

- 960 MHz PLL will not be reprogrammed dynamically depending on connectivity mode of operation
  - All required clocks are derived from this single PLL freq source
- List of system clocks generated from PLL (Note: Most clocks also run temporarily off the 19.2 MHz XO during boot and wake)
  - **WLAN_PHY_CLK** (Root1_320_160_80)
    - Divider selects from 320/160/80 MHz depending on WLAN BW Mode (20 MHz/ 40 MHz/ 80 MHz)
    - Controlled by Power Manager
  - **WLAN_ADC_DAC_CLK** (Root2_240_120_60)
    - Divider selects between 240/120/60 MHz depending on WLAN BW Mode (20 MHz/ 40 MHz/ 80 MHz)
    - Controlled by Power Manager
  - **CPU_CLK** (Root3_240_120_60)
    - Used for cCPU and AHB (AHB is always div2)
    - Divider selects between frequencies depending on MIPS requirements
    - Can change cCPU freq independent of all other technology clocks
    - Controlled by Power Manager
  - **BT_CLK** (Root5_32)
    - Also shared with Coexistence; controlled by Power Manager
  - **BT_FM_19_2_CLK** (Root6_19_2)
    - Controlled by Power Manager
  - **CLK_XO_19_2** (Root7_19_2)
    - This is the main clock that everything comes up on from boot; SPM also uses this
  - **SLEEP_CLK** (Root8_32_768K)
    - Used for a WLAN sleep timer and for BT sleep clock calibration (if needed)

WCN-SS HW – Clocks *(cont.)*

- The 48 MHz XO and GCC PLL are controlled by the sleep driver SW
- The clock tree settings that exist on entry to sleep are restored upon exit from sleep by the sleep driver
- All requests for clocks are accomplished by function calls to the Clock Regime driver
  - Clock Regime maintains and enforces dependencies
    - Example: will control VDD NPA node to raise VDD to Nominal before switching cCPU to 240 MHz
  - Function calls to Clock Regime are blocking until complete
- PM requests clocks that are shared between wireless technologies
  - WLAN_RFIF_CLK if it controls common HW (UART baud rate), need feedback
  - BT Clock (32 MHz) because it is shared with WLAN due to Coexistence
  - BT/FM Clock (19.2 MHz) because it is shared by BT and FM
  - cCPU Clock (120 MHz/240 MHz) because it is shared by everything
- Other clocks that are technology-specific (i.e., all WLAN clocks) are requested by the respective stack SW

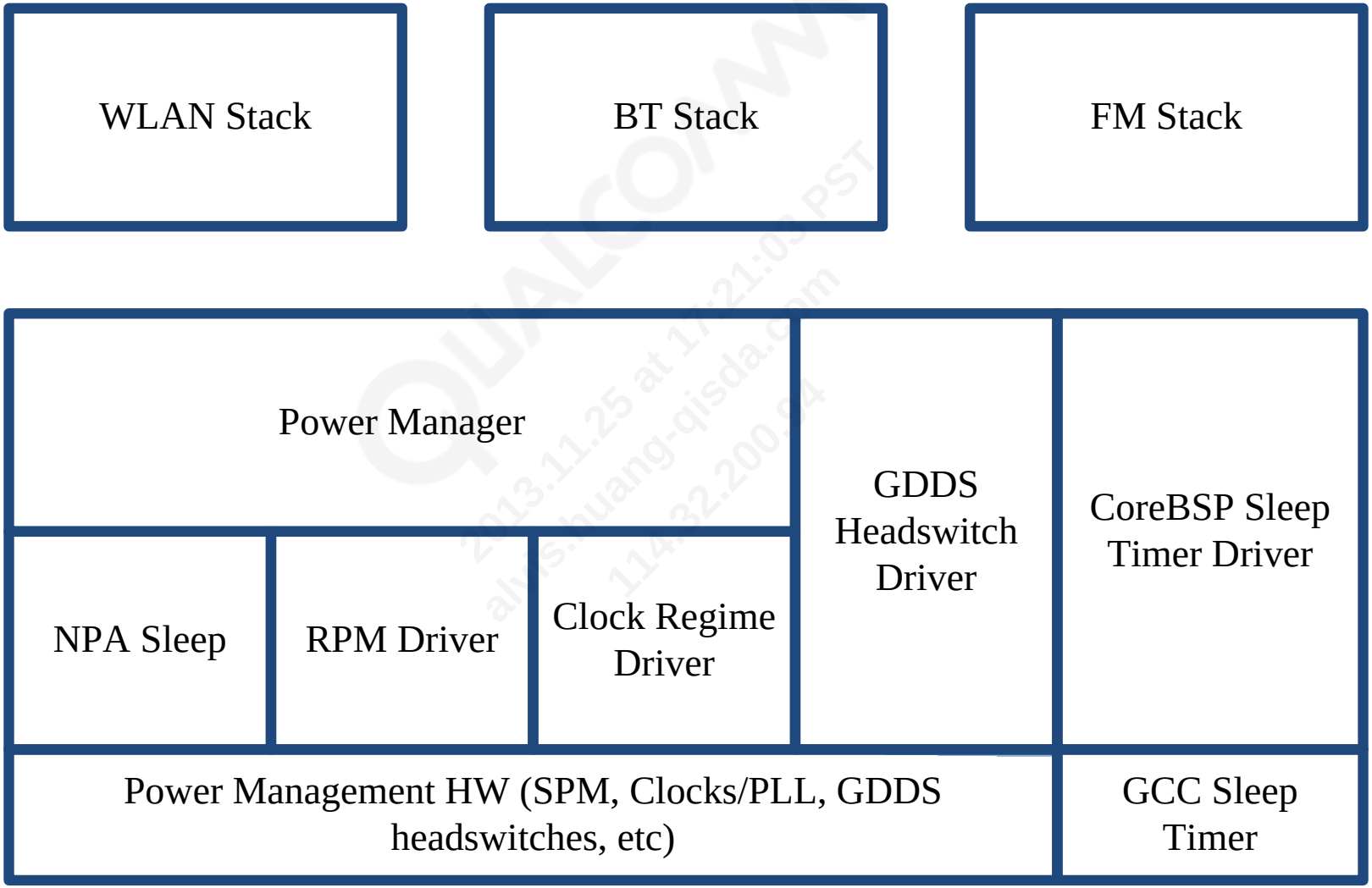
WCN-SS HW – Timers

- There are 32 kHz local timers on both the collapsible COMMON and the always-on TOP domains
- BT_CLK needs to be saved and restored around power collapse
  - Specific HW was added to accurately account for exact number of 32 kHz ticks spent in sleep mode, used to resynchronize BT_CLK by SW
  - Configured in Power Manager since BT_CLK is also used in WLAN as part of Coexistence
- WLAN_CLK needs to be saved and restored around power collapse
  - Also uses specific HW; is done in WLAN stack SW

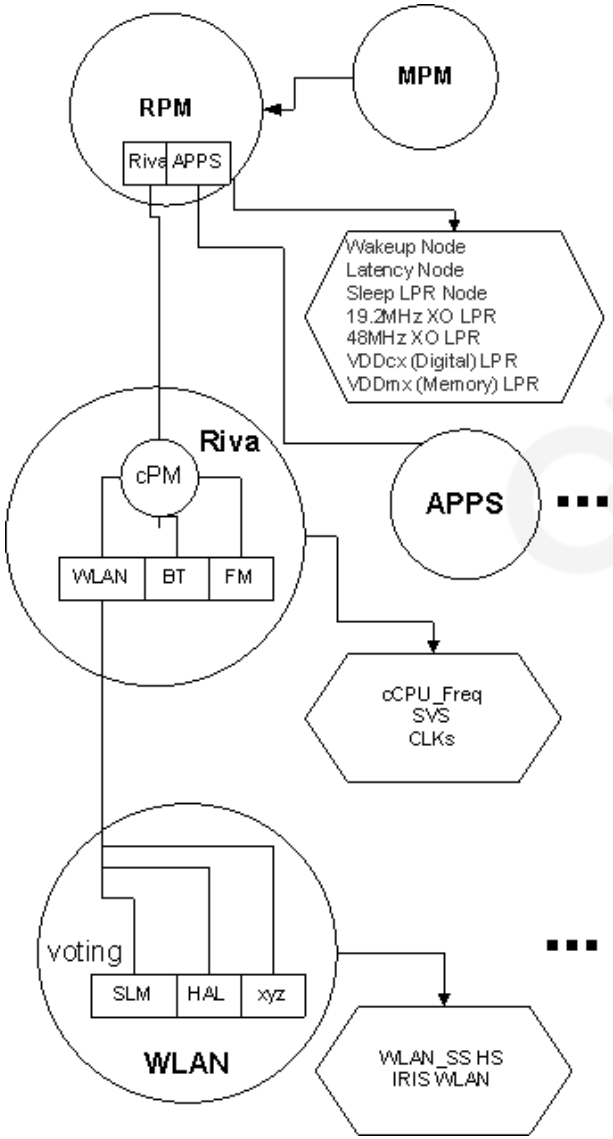
WCN36x0 Power Management

REDEFINING MOBILITY

Power Management SW Architecture



SW Component Hierarchy for Power Save



- Individual stacks (WLAN, BT, FM) decide when they can go idle for a period of time
  - Set a CoreBSP timer
  - Idle the HW specific to that technology (including IRIS)
  - Notify the Connectivity Power Manager of the mode change
- Power Manager (PM) aggregates the modes of all technologies, **controls shared HW resources,** like cCPU clock, and in doing so, configures NPA for sleep
- RPM aggregates control over shared HW resources across all MSM™ subsystems

Power Manager

- Provides an API for WLAN/BT/FM modules to indicate a change in mode
  - PM_WLAN_DISABLED
  - PM_WLAN_80MHZ_ENABLED_ACTIVE
  - PM_WLAN_80MHZ_ENABLED_INACTIVE
    - etc.
  - PM_BT_DISABLED
  - PM_BT_ENABLED_ACTIVE
  - PM_BT_ENABLED_INACTIVE
  - PM_BT_VOICE_ENABLED_ACTIVE
    - etc.
  - PM_FM_DISABLED
  - PM_FM_ENABLED_ACTIVE
  - PM_FM_AUDIO_ENABLED_INACTIVE
    - etc.
- Abstracts the power management details from the WLAN/BT/FM modules
  - Co-locates all power management details in one module
  - Finds the least common denominator among all currently enabled modes
  - Sets LPRMs for LPRs registered with Sleep driver
  - Controls cCPU clock

cCPU Clock Scaling

- cCPU clock frequency is required to be 240 MHz for all BT Active modes and WLAN Active mode
- VDDcx must be at nominal level for cCPU to operate at 240 MHz
- Clock Regime Driver includes NPA request to VDD_Dig node to ensure VDDcx is at the correct level before setting cCPU clock to requested frequency
- To minimize the RPM messages on a cCPU clock frequency change, the Power Manager makes its own NPA request to the VDD_Dig node; this is bundled with the request to the Internal Bus Driver in an NPA Transaction, which results in a single message to the RPM that includes all requests generated by the VDD_Dig request and Internal Bus Driver requests

SVS Support

- The default VDDcx voltage level is SVS
- WCNSS will request a nominal voltage level if it requires the cCPU to operate at 240 MHz; cCPU operates at 240 MHz for all Bluetooth and WLAN Active modes
- WCNSS will request a return to SVS voltage level once the mode that required the 240 MHz cCPU clock frequency is changed

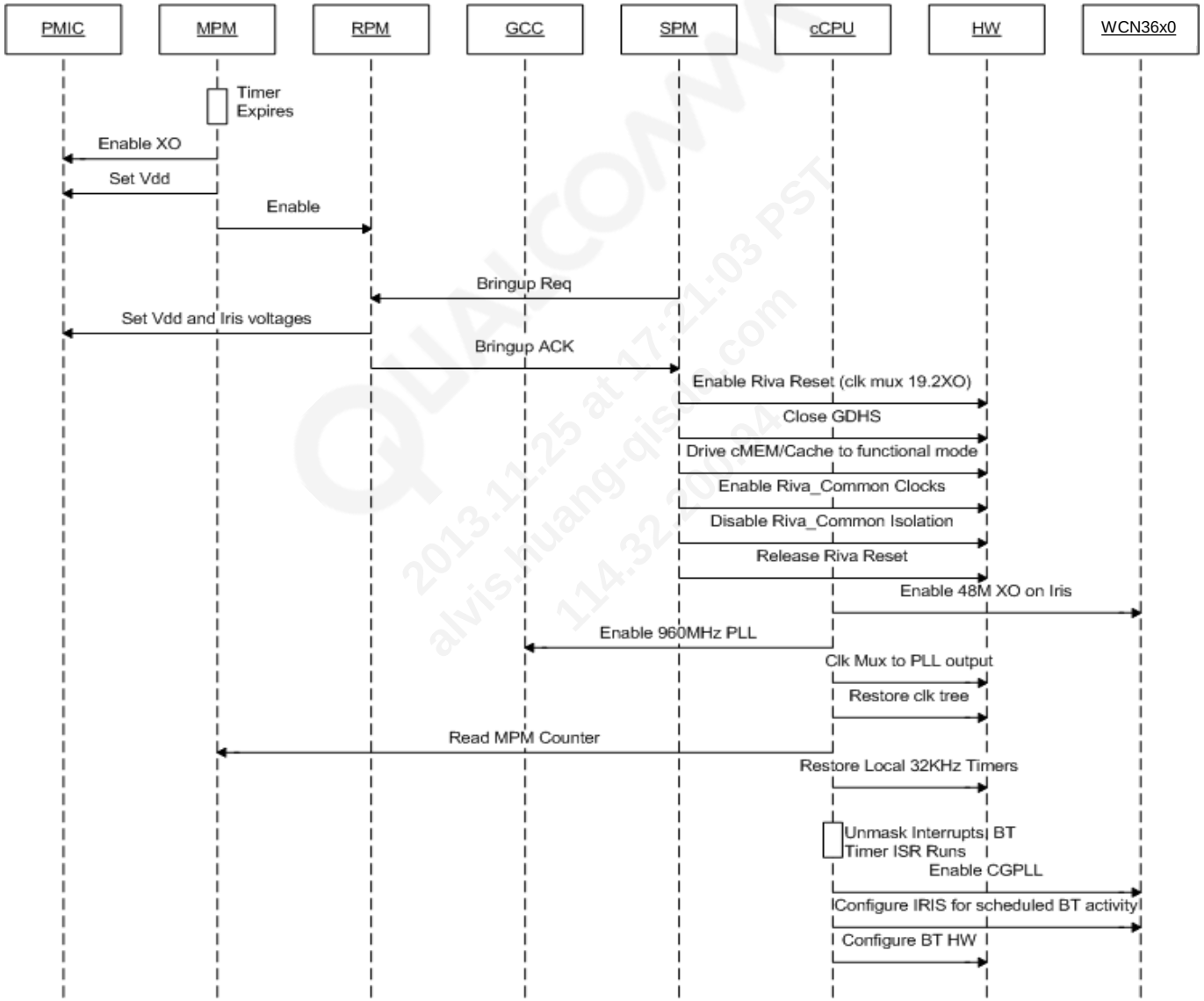
ARM9 Clock Gating and Power Collapse with VDD MIN

- On cCPU entering Idle thread, there are only two expected options for the state to enter
- The first of these is ARM9 Clock Gating; this state is entered when the next expected wake time is too close to the current time, or when the PM has disabled GDHS; ARM9 Clock Gating mode gates the cCPU clock to the ARM9; no other hardware is turned off
- The PM uses an NPA node provided by the Sleep Driver to disable GDHS; this is done based on the Power Manager mode set by each technology
- If there is enough time until the next expected wakeup, the Power Collapse with VDD min state is entered

Core BSP Resources Required

- Clock Regime Driver
  - CPU clock frequency changes are requested through /clk/cpu NPA node
  - All other clocks needed by WCNSS are configured using the DAL Clock API
  - Handles the switch to/from the IRIS 48 MHz XO on power collapse entry/exit
- Sleep Driver
  - GDHS Disable requests are made using the /core/cpu/vdd NPA node
  - Configures RPM's Sleep set for WCNSS/Iris voltage supplies
- Internal Bus Driver
  - Bus bandwidth requests are made through the icbarb API, which uses the /icb/arbiter NPA node
  - Bus Bandwidth requests are made based on PM mode changes
- PMIC NPA
  - The PM uses PMIC NPA nodes to request the voltage supplies to be placed in the RPM's active set
  - PMIC Driver used to communicate directly with PMIC does not exist on WCNSs

Wakeup Sequence - BT Scheduled Activity



Enter Sleep Sequence

Enter Sleep	
cCPU	CoreBSP Sleep Driver sends RPM resource set change, and RPM sleep timer value for scheduled wake time
	CoreBSP Sleep Driver configures SPM for sleep
	CoreBSP Sleep Driver uses Clock Regime driver to set clock gen MUX to 19.2M output
	CoreBSP Sleep Driver uses Clock Regime driver to disable 960 MHz PLL (clears bit in GCC)
	CoreBSP Sleep Driver uses Clock Regime driver to configure Iris clock plan and disable 48M XO (if used)
	CoreBSP Sleep Driver saves state of cCPU in DDR RAM, executes SWFI
SPM	SPM asserts Common reset
	SPM enables Common isolation
	SPM disables Common clocks
	SPM drives cMEM into retention mode (if required)
	SPM disables headswitch to Common
	SPM notifies RPM is in sleep state
RPM/MPM	RPM grants sleep request
	RPM takes down Iris voltage sources
	RPM instructs MPM to set VDDcx & VDDmx to proper levels
	MPM changes VDDcx and VDDmx from nominal to retention (or off) levels
	MPM disables 19.2M PMIC XO buffer

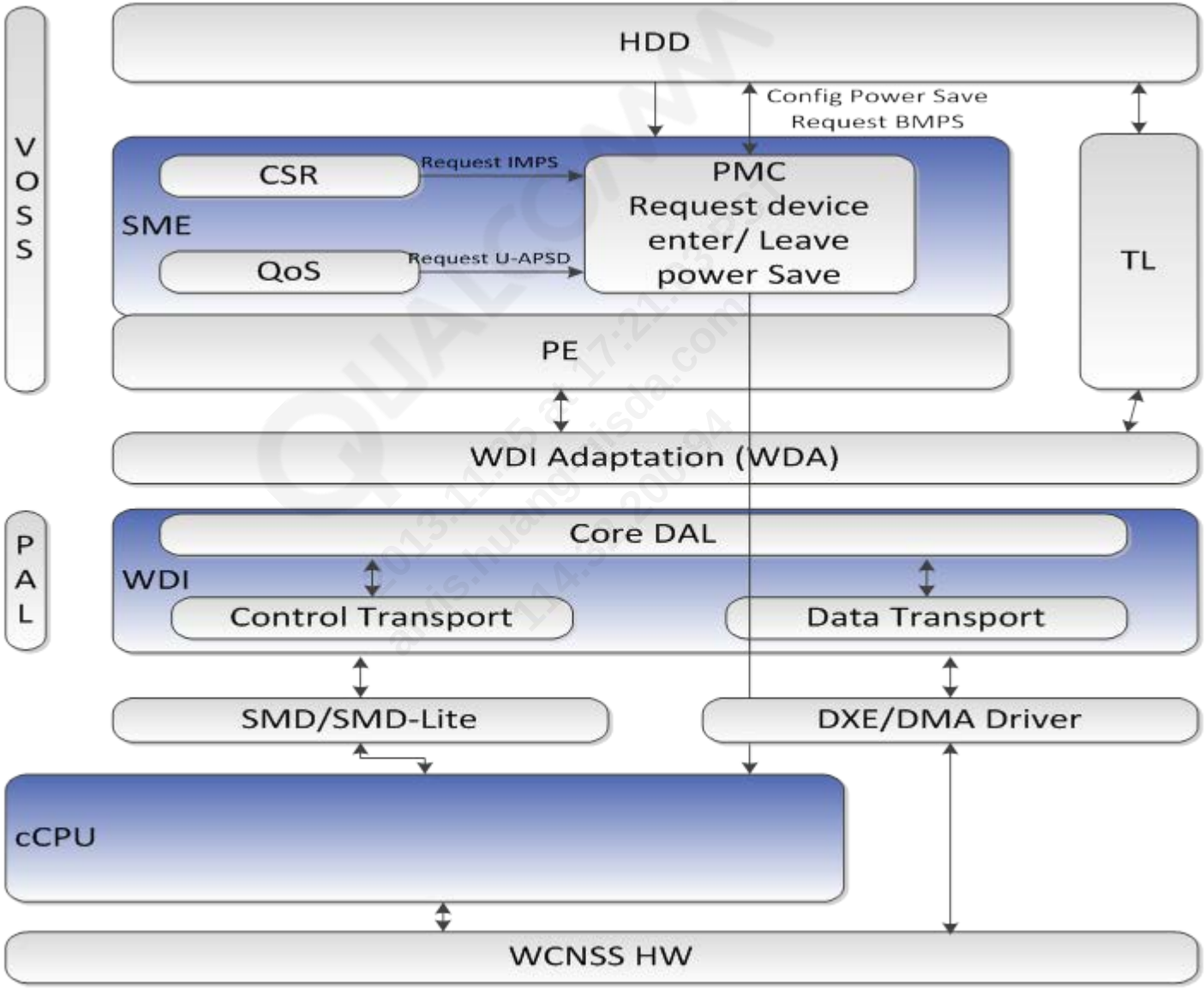
Exit Sleep Sequence

Exit Sleep	
MPM	MPM Timer Expires (t = 0)
	MPM Enables 19.2M PMIC XO buffer
	MPM Changes VDDcx and VDDmx from retention/off to nominal levels (1.05V)
	MPM Enables RPM
RPM/SPM	RPM Sends wakeup to SPM
	SPM issues bringup_req to RPM
	RPM brings up Iris voltage sources
	RPM issues bringup_ack to SPM
	SPM enables Common headswitches
	SPM drives cMEM into functional mode
	SPM enables Common clocks
	SPM disables Common isolation
	SPM de-asserts Common reset
cCPU	cCPU executes warm boot code in CoreBSP, OS restores context from DDR, and returns to the SWFI instruction
	CoreBSP Sleep Driver uses Clock Regime driver to configure Iris clock plan and enable 48M XO (if used)
	CoreBSP Sleep Driver uses Clock Regime driver to enable 960MHz PLL (sets bit in GCC)
	CoreBSP Sleep Driver uses Clock Regime driver to set clock gen MUX to PLL output
	CoreBSP Sleep Driver restores local Sleep Timer by reading MPM counter
	CoreBSP Sleep Driver umasks interrupts; cCPU processes interrupt

WLAN Power Save Mode (BMPS and IMPS)

REDEFINING MOBILITY

Android Power Save Mode



Power Management Control

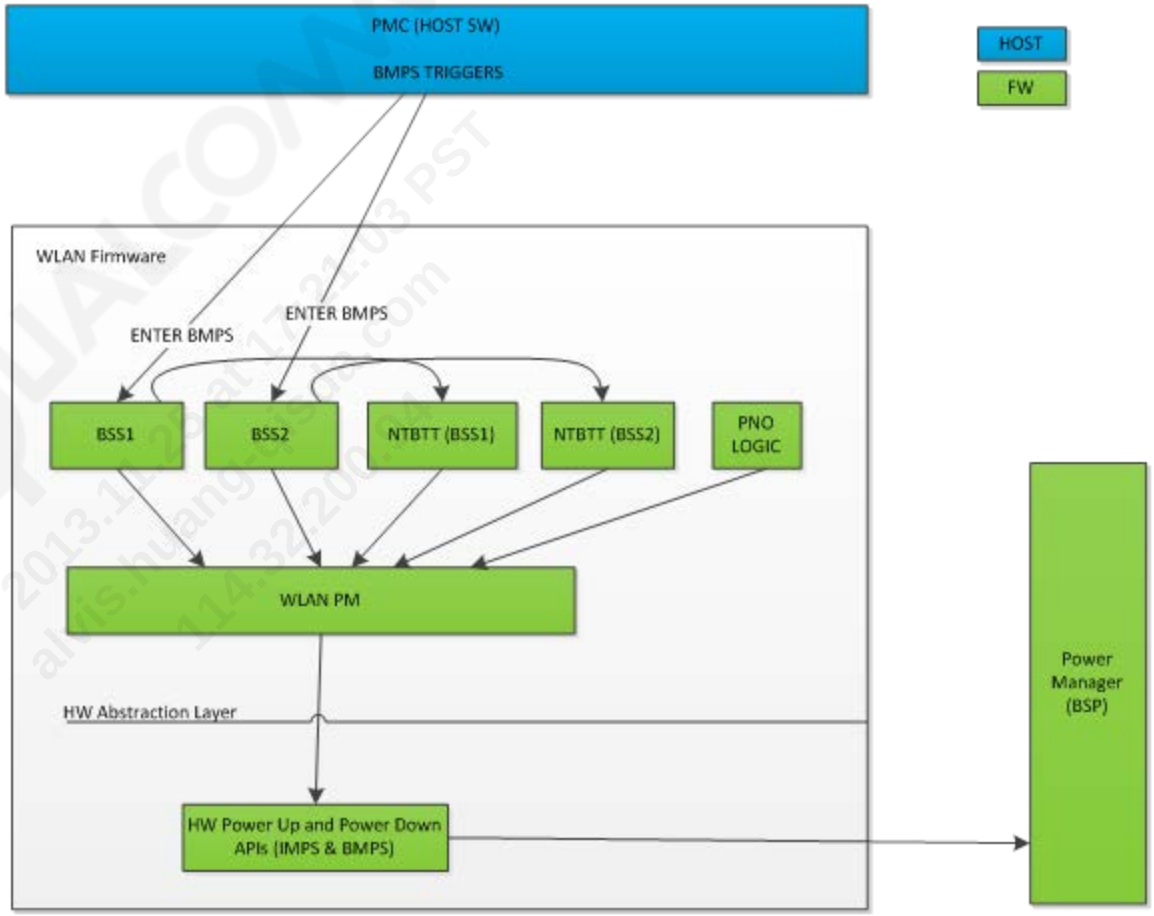
- PMC features
  - Part of the Station Management Entity (SME)
  - Designed to be independent of the OS platform
  - Independent of a target hardware device to accommodate future products
  - Provides services to HDD, CSR, QoS and other driver modules
  - Uses vOSS and PE/HAL services to accomplish power management-related tasks

PMC Services

- Driver modules can use these PMC services
  - Query the current Power Save state of the system
  - Request to enter a specific Power Save mode, e.g., IMPS and BMPS
  - Request Full Power
  - Disallow entry into Power Save modes
  - Control Power Save modes, wakeup cycles, etc. by updating the config
  - Signal power-related events, e.g., hibernate, etc.

Power-Save FW

- WPM (WLAN Power Manager)
  - Aggregate votes (from the clients using API) and their modes of operation
  - Four states exist in WPM:
    - RXONLY
    - ACTIVE
    - Low_pwr1: BMPS
    - Low_pwr2: IMPS or no vote from any clients



Name	Description	Default
gEnableImps	Enable Idle Mode Power Save 1 – Enable 0 – Disable	1
gEnableBmps	Enable Beacon Mode Power Save 1 – Enable 0 – Disable	1
gBmpsMinListenInterval **gBmpsModListenInterval** **gBmpsMaxListenInterval**	Default Configure DTIM 1 – Min 65535 – Max	1
gEnableModulatedDTIM	Use Modulated DTIM 0 – Disable 5 – Max	0
gEnableDynamicDTIM	Use Dynamic DTIM 0 – Disable 5 – Max	0
gTelescopicBeaconWakeupEn	Use Tele-DTIM 0 – Disable 1 – Enable	0
telescopicBeaconTransListenInterval **telescopicBeaconTransListenIntervalNumIdleBcns** **telescopicBeaconMaxListenInterval** **telescopicBeaconMaxListenIntervalNumIdleBcns**	Configure Tele-DTIM For Interval: 1 – Min 7 – Max For NumIdleBcns 5 – Min 255 – Max	3 10 5 15

- See Q2 and Q3 for more information.

BMPS

REDEFINING MOBILITY

BMPS (Beacon Mode Power Save)

- In WCNSS_qcom_cfg.ini
  - gEnableBmps = 1
- Associated with an access point and no traffic in Tx or Rx
- WLAN digital domain is power-collapsed along with the corresponding domain in the WCN3660 chipset; if no other connectivity technology (BT or FM) is active, the entire WCN subsystem is power-collapsed
- BMPS Enter trigger can come from two places
  - HDD - WLAN host device driver
  - PMC – WCNSS FW autonomously may decide to put the system in BMPS mode
- Wake up every Listen/DTIM interval to listen to Beacon to check DTIM information to retrieve packets buffered at AP and check capability information
- BMPS Exit trigger can come from
  - Explicit Exit indication from HDD
  - WCNSS FW can decide to bring the chip out of BMPS
  - WCNSS FW is responsible for putting the chip back to BMPS

BMPS Listen Interval Negotiation

- If gIgnoreDtim isn't set, set the LI using
  - gBmpsMinListenInterval
  - gBmpsModListenInterval
  - gBmpsMaxListenInterval
- Use gPowerUsage to identify which listen interval to use

- DTIM = AP interval
- Listen Interval Calculation
  - 1) If **DTIMs ≤ LI** then **LI = DTIMs.**
  - 2) ELSE
    - (1) If **DTIMs is divisible by LI** then **LI will not be changed.**
      - Example: a. LI=4 and DTIMs=8, LI=4. b. LI=3 and DTIMs=6, LI=3
    - (2) If **DTIMs is not divisible by LI**, then from **GCD to LI, get the biggest value that can divide the DTIMs.**
      - Example: a. LI=4 and DTIMs=9, LI=3 b. LI = 7 and DTIMs = 12, LI = 6

BMPS Listen Interval Negotiation *(cont.)*

- Set gEnableModulatedDTIM = MDTIM

- Modulate DTIM
  - When the system is in suspend (maximum beacon will be at 1s == 10)
    - If maxModulatedDTIM ((MAX_LI_VAL = 10) / DTIMs) equal or larger than MDTIM (configured in WCNSS_qcom_cfg.ini)
      - Set LI to MDTIM * DTIM
      - If Dtim = 2 and Mdtim = 2 then LI is 4
    - Else
      - Set LI to maxModulatedDTIM * DTIMs
  - When the system wakes up
    - Set LI to DTIM

BMPS Listen Interval Negotiation *(cont.)*

- gIgnoreDtim = 1
- ignoreDtim
  - Ignore DTIM interval; set the current LI

- Use the gEnableDynamicDTIM = a
- Dynamic DTIM
  - When the system into the suspend – LCD off
    - Reconfigure power parameters (DTIMa)
    - Exit BMPS
    - Re-enter BMPS so that WCNSS takes into account the DTIMa
  - When the system wakes up
    - Reconfigure power parameters (DTIM1)
    - Exit BMPS
    - Re-enter BMPS so that WCNSS takes into account the DTIMa

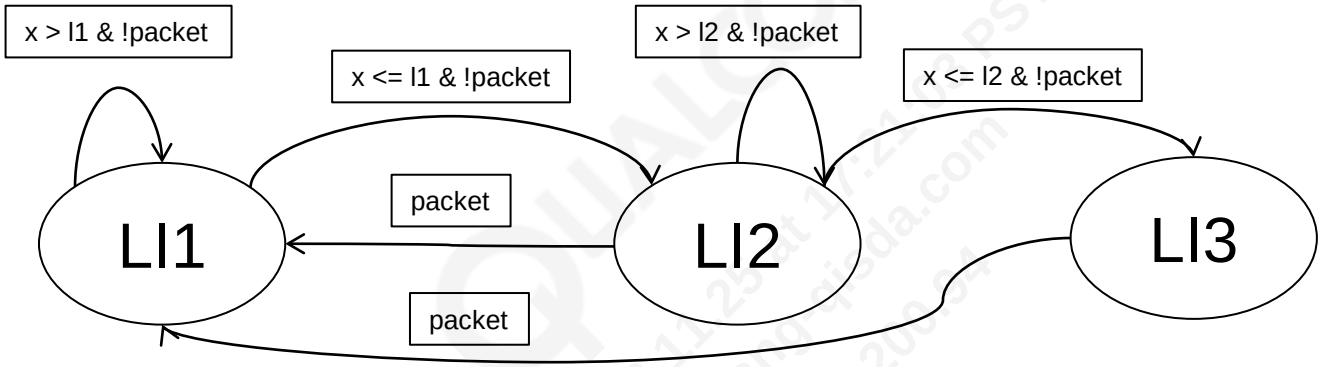
BMPS Listen Interval Negotiation *(cont.)*

- Set gTelescopicBeaconWakeupEn = 1

x = count how long it stays in that node without packet

l1 = telescopicBeaconTransListenIntervalNumIdleBcns

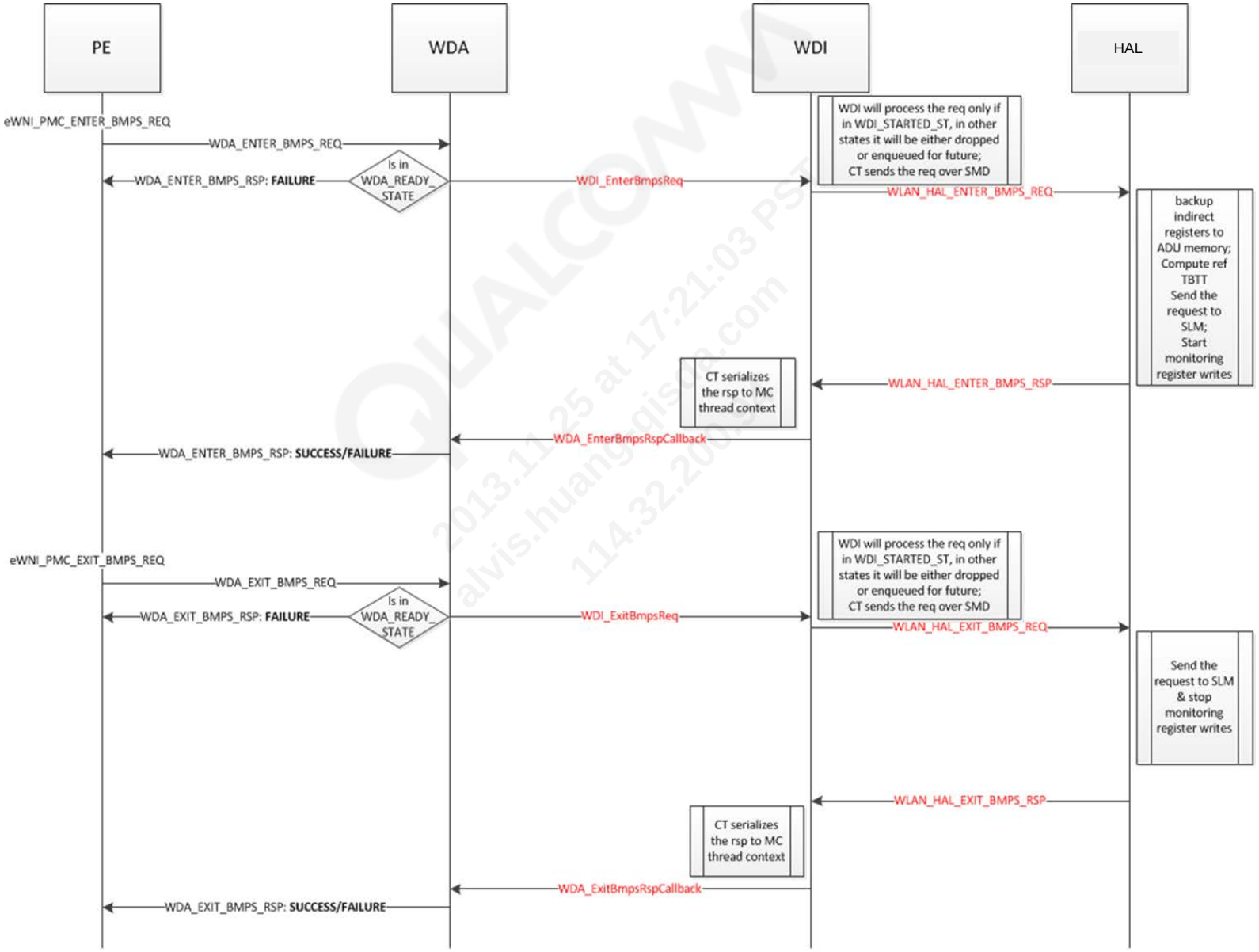
l2 = telescopicBeaconMaxListenIntervalNumIdleBcns



- Telescopic DTIM
  - Start with interval of 1
  - Stay in interval of 1 for telescopicBeaconTransListenIntervalNumIdleBcns amount
    - If no data during that period move to telescopicBeaconTransListenInterval interval
      - Stay in this interval for telescopicBeaconMaxListenIntervalNumIdleBcns amount
        - If no data during that period move to telescopicBeaconMaxListenInterval
  - If data is presented, move back to interval of LI1
  - If two consecutive beacons are missed, move back to interval of LI1
    - After first beacon miss, it tries to receive the next earliest arriving beacon

- See Q1 for more information.

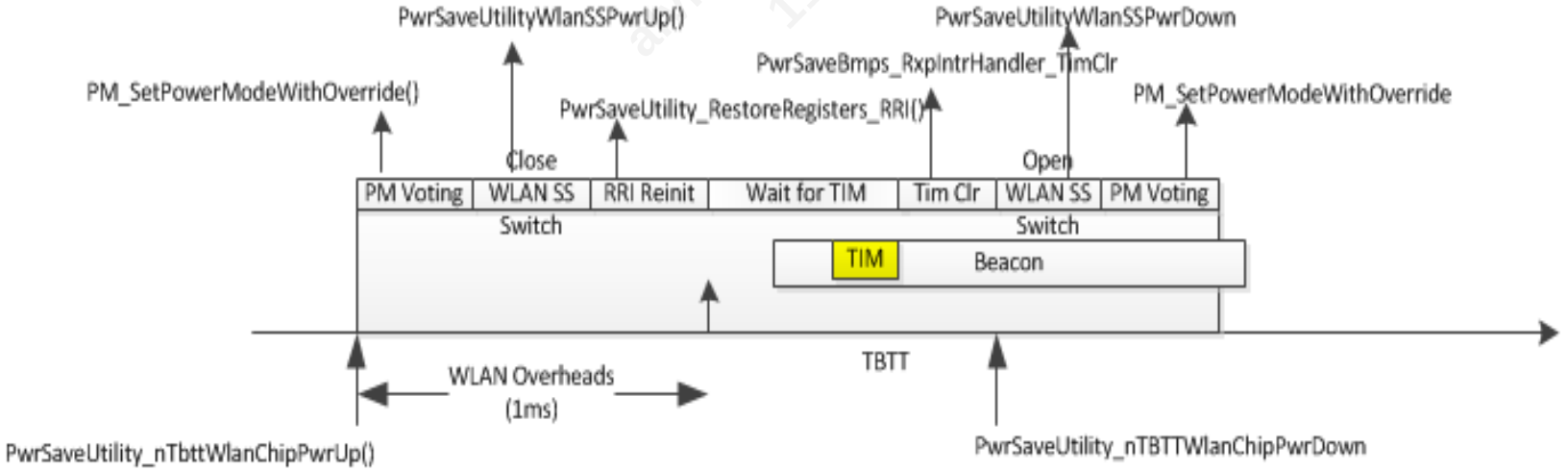
BMPS Sequence



Beacon Early Termination (BET)

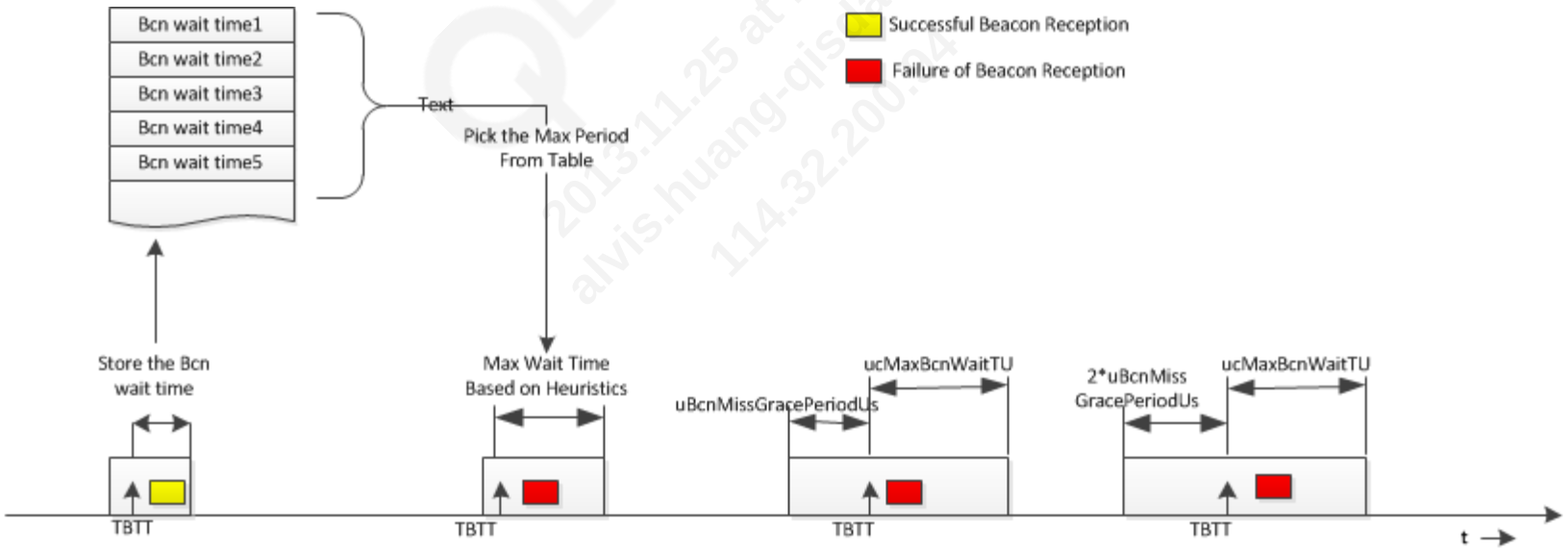
- Terminate the reception of beacon if the TIM element is clear for the power saving
- BET can be configured in a WCNSS configuration file
- BET is supported only in 2.4 Ghz for Infra STA case
- For P2P Client BET is disabled as FW has to parse NOA attributes in Beacon

Beacon Early Termination (BET) Config ('enableBeaconEarlyTermination=1')	Behavior
beaconEarlyTerminationWakeInterval=3	Every 2nd Beacon is a Non-BET beacon and hence has longer active duration
beaconEarlyTerminationWakeInterval=9	Every 4th Beacon is a Non-BET beacon and hence has longer active duration
beaconEarlyTerminationWakeInterval=10	Every 5th Beacon is a Non-BET beacon and hence has longer active duration



Beacon Miss Detection

- Maintains a circular buffer with Beacon wait times for last 50 beacons
- Pick a maximum value from the buffer and use it as next timeout period for receiving beacon (range: ucMinBcnWaitTU, ucMaxBcnWaitTU)
- Beacon Wait Window extends with consecutive beacon misses
- Beacon window is flexible for APs with good and poor connectivity



Beacon window extension with consecutive bcn misses

IMPS

REDEFINING MOBILITY

IMPS (Idle Mode Power Save)

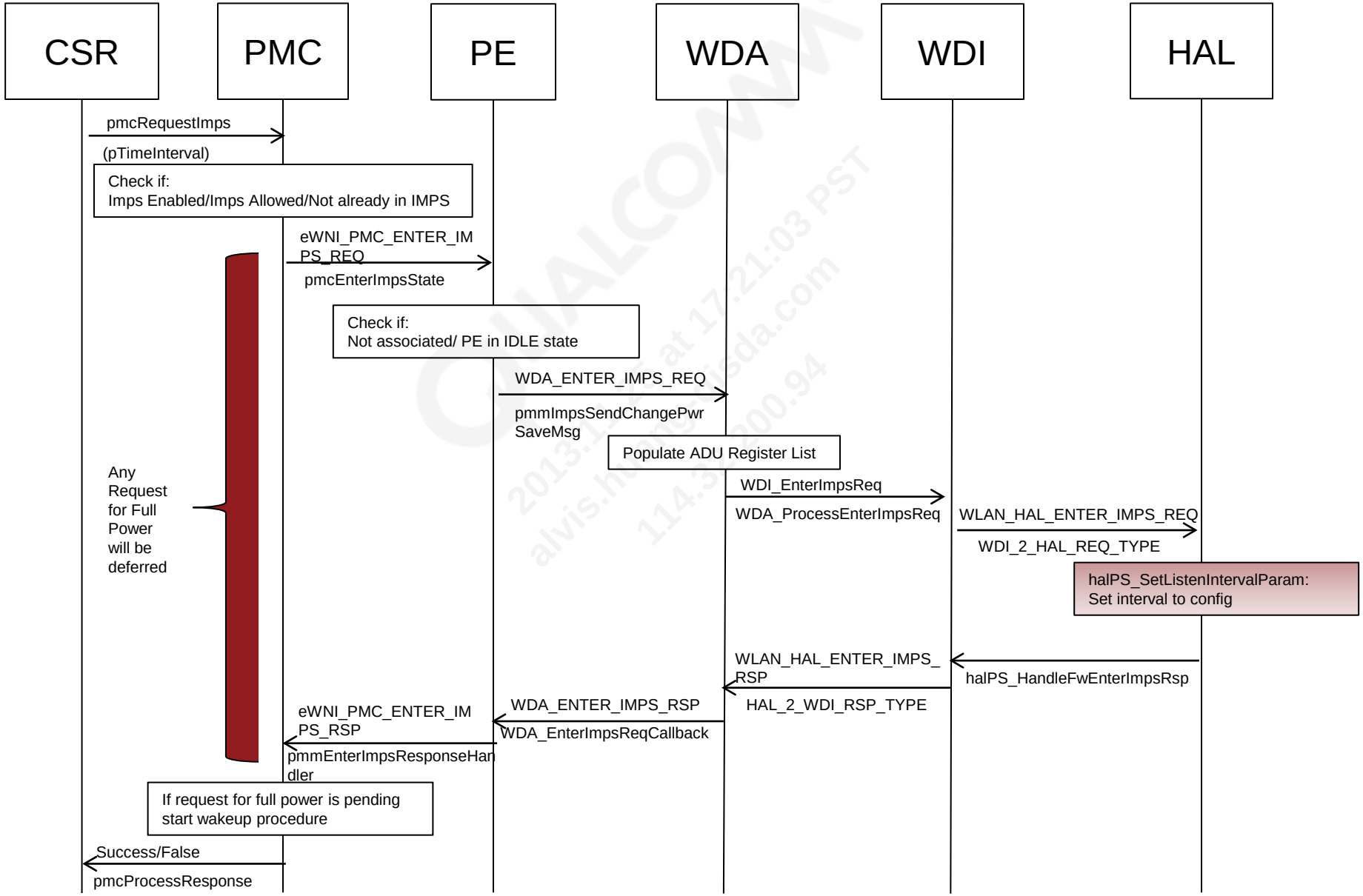
- To enable, in WCNSS_qcom_cfg.ini
  - gEnableImps = 1

- Station is not connected to AP
- CSR requests PMC enter IMPS between scans
- IMPS exit or does not start during
  - Idle scan is disabled
  - Device stays in IMPS until it is explicitly requested to exit that mode
- Entry/exit from this mode is triggered by host software
- WLAN register contents need to be restored upon exiting this mode
- Hardware state
  - WLAN hardware domain is power-collapsed
  - WCNSS common may or may not be power-collapsed based on the scan

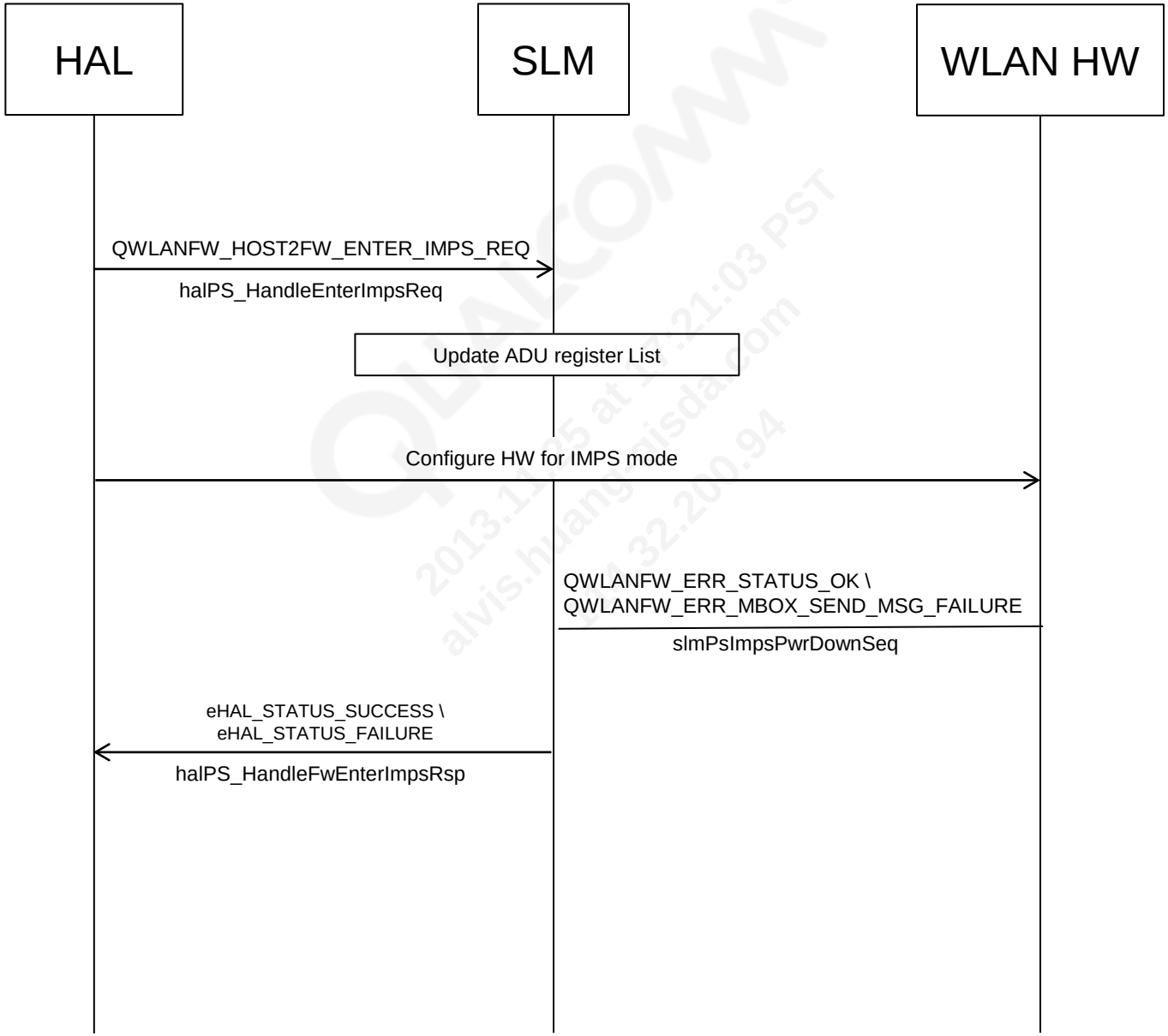
IMPS *(cont.)*

- Station looks in its existing profile list and sends a probe request, with SSID specified in profile
- Station repeats process and continues to repeat steps until it is associated with an AP

IMPS Sequence in Host



IMPS Sequence in WCNSS



Testing

- Almost all the target numbers are calculated on screen room
  - Only one client connected to AP
  - No noise generated
  - Broadcast/multicast filter is on
  - It is performance-configured (many apps and debug features are turned off)

- During BMPS and IMPS
    - In default, Android wakes up the host every 5 min in IMPS
    - In default, Android wakes up the host every 30 min in BMPS
  - During those modes, if the current consumption goes up to 70mA and stays there for more than 1 sec, it could be app processor not going to suspend mode
  - WCNSS should not exceed 2 % of usage during those modes

- Disconnect JTAG and USB during your current measurement

- DTIMx
  - x indicates what interval it is. x = 1 then 100 ms each beacon during BMPS

Testing *(cont.)*

- Capture kernel log with iwpriv enabled
  - This will display the extra log message
    - SME: Iwpriv wlan0 setwlandbg 6 8 1
    - WDA: Iwpriv wlan0 setwlandbg 8 8 1
    - HDD: Iwpriv wlan0 setwlandbg 5 4 1

- Capture powertop log
  - To see what kind of process is running at that moment and processor speed

- Capture tcpdump
  - To debug what kinds of packets trigger apps processor to wake up
  - In ADB shell
    - tcpdump -i any -s 0 -w /data/ip2.pcap

Packet Filtering and ARP Offload

REDEFINING MOBILITY

Packet Filtering and ARP Offload

- Packet Filtering
  - There are two different types of Packet Filtering
    - Filter received Multicast/Broadcast packets completely without passing them to host driver
    - Filter received Multicast/Broadcast packets selectively based on set rules
- ARP Offload
  - Host offloads the process of sending ARP response messages
  - WCNSS sends ARP response messages
- WCNSS_qcom_cfg.ini contains config params for Packet Filtering and ARP Offload
  - Whenever the system is going into early suspend (screen off), wlan driver will apply the filters
  - Whenever the system is moving out of suspend (late resume) these filters will be cleared

ARP Offload and Packet Filtering during Concurrency

- WCNSS stays in Active mode during concurrency (STA + P2P Client or STA + P2PGo)

- WCNSS Active mode Offloads is Disabled
  - Config param gEnableActiveModeOffload=0
  - ARP offload and MC filtering is done in WCNSS when phone enters Suspend mode and when WCNSS is in BMPS mode of operation
  - This implies that ARP offload and MC filtering will not happen during concurrency if WCNSS Active mode Offloads is disabled

- WCNSS Active mode Offloads is Enabled
  - Config param gEnableActiveModeOffload=1
  - ARP offload and MC filtering becomes independent of the BMPS mode
  - ARP offload and MC filtering works even in WCNSS Active mode
  - This implies that ARP offload and MC filtering will happen during concurrency if WCNSS Active mode Offloads is enabled

Packet Filtering and ARP Offload – Config Params

- The following Config Params are supported in WCNSS_qcom_cfg.ini

Config param in ini	Functionality	Allowed Values
mcastBcastFilter	To filter Mcast / Bcast Rx packets completely	0: No filtering 1: Filter all Multicast. 2: Filter all Broadcast. 3: Filter all Mcast and Bcast
hostArpOffload	To enable HostARPOffload feature so that ARP response will be sent	0 – Disable 1 – Enable
isMcAddrListFilter	To allow Mcast RX packets from registered addresses only and filter remaining Mcast and Bcast Rx packets	0 – Disable 1 - Enable

- If gEnableActiveModeOffload=1 following will be the behavior during concurrency

Concurrent Mode	ARP Offload hostArpOffload=1	MCPacket Filtering isMcAddrListFilter=1	MCBC Filtering McastBcastFilter=0
STA + P2P Client	ARP Offload is enabled for STA and P2P Client modes.	Packet Filtering is enabled for both STA and P2P Client modes	Currently Not Supported in concurrency.
STA + P2P GO	ARP Offload is enabled for STA mode Only.	Packet Filtering is enabled for STA mode ONLY.	Currently Not Supported in Concurrency.

Packet Filtering and ARP Offload – Config Params *(cont.)*

Config param in ini	Allowed Values
mcastBcastFilter	0: No filtering 1: Filter all Multicast. 2: Filter all Broadcast. 3: Filter all Mcast and Bcast
hostArpOffload	0 – Disable 1 – Enable
isMcAddrListFilter	0 – Disable 1 - Enable

mcastBcastFilter	isMcAddrListFilter	hostArpOffload	Expected Behavior		
			Multicast Packets Filtered by FW/HW	Broadcast Packets Filtered by FW /HW	ARP Rsp
0	0	0	NO	NO	NO
1	0	0	YES	NO	NO
2	0	0	NO	YES	NO
3	0	0	YES	YES	NO
0	1	0	YES – Except for the registered Addresses	YES	NO
1	1	0	YES	YES	NO
2	1	0	YES – Except for the registered Addresses	YES	NO
3	1	0	YES	YES	NO
0	0	1	NO	NO	YES
1	0	1	YES	NO	YES
2	0	1	NO	NO	YES
3	0	1	YES	NO(filter internally set to MC only)	YES
0	1	1	YES – Except for the registered Addresses	YES	YES
1	1	1	YES	YES	YES
2	1	1	YES – Except for the registered Addresses	YES	YES
3	1	1	YES	YES	YES

Packet Filtering and ARP Offload - IOCTLs

- The following IOCTLs are available to set Packet Filtering and ARP Offload features

IOCTL	Functionality
WLAN_PRIV_SET_MCBC_FILTER	To filter Mcast/Bcast RX packets completely
WLAN_PRIV_CLEAR_MCBC_FILTER	To clear the filters for Mcast/Bcast RX packets
WLAN_PRIV_SET_HOST_OFFLOAD	To enable HostARPOffload feature so that ARP response will be sent. This IOCTL can also be used for Neighbor Discovery Offload.
WLAN_SET_PACKET_FILTER_PARAMS	To filter Mcast and Bcast RX packets selectively based on filter params

IOCTL - WLAN_PRIV_SET_HOST_OFFLOAD

- IOCTL **WLAN_PRIV_SET_HOST_OFFLOAD** can be used for ARP Offload or Neighbor Discovery Offload

- Input Params
  - Offload Type
    - 0 - WLAN_IPV4_ARP_REPLY_OFFLOAD
    - 1 - WLAN_IPV6_NEIGHBOR_DISCOVERY_OFFLOAD
  - Enable Or Disable Flag
    - 0 - WLAN_OFFLOAD_DISABLE
    - 0x1 - WLAN_OFFLOAD_ENABLE
    - WLAN_OFFLOAD_ARP_AND_BC_FILTER_ENABLE
      - It is valid only in the context of ARP Offload type
      - Its value is (WLAN_OFFLOAD_ENABLE | WLAN_OFFLOAD_BC_FILTER_ENABLE)
      - 0x2 - WLAN_OFFLOAD_BC_FILTER_ENABLE
  - IPV4 Address
    - If Offload Type is ARP and the request is for Enable
  - IPV6 Address
    - If Offload Type is Neighbor Discovery and the request is for Enable

- Refer wlan_hdd_host_offload.h for more details

IOCTL - WLAN_SET_PACKET_FILTER_PARAMS

- WLAN_SET_PACKET_FILTER_PARAMS
  - Packet Filtering feature enables FW to filter multicast and broadcast packets and send only those packets that match the configured filter rules up to the host SW even when the Host is active
  - Provides flexibility to filter intended multicast and broadcast packets at reception in STA Active mode
  - Received packets that do not match the filter rules set will be dropped and not sent to host
  - Unicast frames will unconditionally be sent to host
  - Parameters are configured dynamically through a private IOCTL

IOCTL - WLAN_SET_PACKET_FILTER_PARAMS *(cont.)*

- Packet Filtering IOCTL requires input parameters
  - filterAction – Enum Type To Set and Clear filters
  - filterId – ID of the filter
  - numParams – Number of parameters/frame headers to add for filtering
  - paramsData – Array of structures with protocol layer header, comparison flags & data fields corresponding to the header as below
    - protocolLayer – Type of protocol layer header to which the data being configured correspond
    - cmpFlag – Comparison type
    - dataOffset – Offset of the data to compare from the respective protocol layer header start (as per the respective protocol specification) in terms of bytes
    - dataLength – Length of data to compare
    - compareData – Array of 8 bytes
    - dataMask – Mask to be applied on the received packet data (array of 8 bytes)

IOCTL - WLAN_SET_PACKET_FILTER_PARAMS *(cont.)*

- Possible values for the input parameters
  - filterAction can have 2 valid values
    - filterAction = 1 means to set the filter
    - filterAction = 2 means to clear the filter
  - filterId can have the following possible values
    - filterId = 0 to 9
  - Elements of paramsData structure can take the following values
    - protocolLayer = 1 for MAC header
    - protocolLayer = 2 for ARP header
    - protocolLayer = 3 for IP header
  - cmpFlag can have the following values
    - cmpFlag = 0 means comparison is invalid
    - cmpFlag = 1 means compare for equality of the data present in received packet to the corresponding configured data
    - cmpFlag = 2 means for equality of the data present in received packet to the corresponding configured data after applying the mask
    - cmpFlag = 3 means compare for non-equality of the data present in received packet to the corresponding configured data
    - cmpFlag = 4 means compare for non-equality of the data present in received packet to the corresponding configured data after applying the mask
    - cmpFlag can be programmed with any of the above values in such a way that received frames can selectively be allowed to host or dropped at FW

IOCTL - WLAN_SET_PACKET_FILTER_PARAMS *(cont.)*

- **Example (1) –** Add a filter for IPv6 multicast packets (multicast address starting with 0x33 33) at FW even when host is in Active mode and FW will allow/push the packets matching with this filter to host. All other data packets received which do not fall into this filter category will be dropped by FW and will not be allowed to host
  - With filterAction = 1, filterId = 2, numParams = , 2paramsData [0]. protocolLayer = 1, paramsData [0].cmpFlag = 3, paramsData [0].compareData = 0x333300000000, paramsData [0].dataMask = 0xFFFF00000000, paramsData [0].dataOffset = 4, paramsData [0]. dataLength = 6 ; paramsData [1]. protocolLayer = 1, paramsData [1].cmpFlag = 1, paramsData [1].compareData = 0x86DD, paramsData [1]. dataLength = 2 & paramsData [1].dataOffset = 34,

IOCTL - WLAN_SET_PACKET_FILTER_PARAMS *(cont.)*

- Example (2) – To drop IPv6 multicast frames with IP address range as FFxx::\16 at FW, the filter need to be configured as follows
  - filterAction = 1
  - filterId = 7 (can be any value from 0 to 9)
  - numParams = 4
  - paramsData [0]. protocolLayer = 2, paramsData [0].cmpFlag = 3, paramsData [0].dataOffset = 6, paramsData [0]. dataLength = 2 , paramsData [0].compareData = 0x86DD, paramsData [0].dataMask = 0x0
  - paramsData [1]. protocolLayer = 3, paramsData [1].cmpFlag = 4, paramsData [1].dataOffset = 24, paramsData [1]. dataLength = 2, paramsData [1].compareData = 0xFF, paramsData [1]. dataMask = 0xFF
  - paramsData [2]. protocolLayer = 3, paramsData [2].cmpFlag = 3, paramsData [2].dataOffset = 28, paramsData [2]. dataLength = 8, paramsData [2].compareData = 0x0, paramsData [2].dataMask = 0x0
  - paramsData [3]. protocolLayer = 3, paramsData [3].cmpFlag = 3, paramsData [3].dataOffset = 36, paramsData [3]. dataLength = 4 , paramsData [3].compareData = 0x22, paramsData [3].dataMask = 0x0

IOCTL - WLAN_SET_PACKET_FILTER_PARAMS - Test with iwpriv

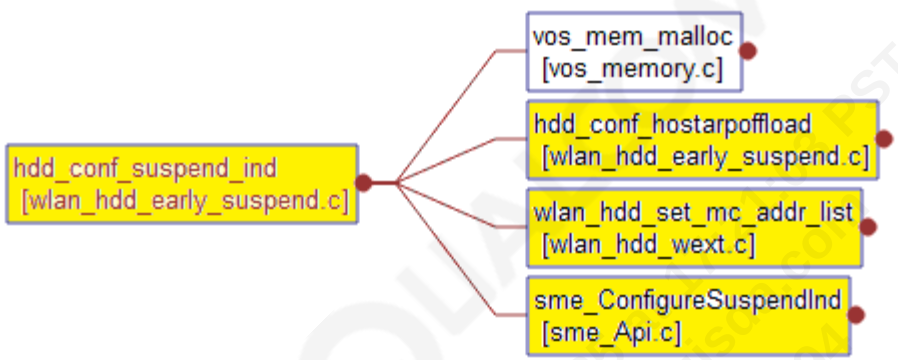
- Command to set filter to drop ipv4 MC (address range :- 224.0.0.0 to 239.255.255.255) by using mask not equal(4)
  - adb shell iwpriv wlan0 setPktFilter 1 8 1 3 4 16 1 224 0 0 0 0 0 0 0 240 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- Command that clears above ipv4 MC filter
  - adb shell iwpriv wlan0 setPktFilter 2 8 1 3 4 16 1 224 0 0 0 0 0 0 0 240 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- Command to set filter to drop ipv6 MC (address range :- FFxx::\16) by using not equal(3) and mask not equal(4)
  - adb shell iwpriv wlan0 setPktFilter 1 7 4 2 3 6 2 134 221 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 4 24 2 255 0 0 0 0 0 0 0 255 0 0 0 0 0 0 0 3 3 28 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 36 4 0 0 0 22 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- Command that clears the ipv6 MC filter
  - adb shell iwpriv wlan0 setPktFilter 2 7 4 2 3 6 2 134 221 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 4 24 2 255 0 0 0 0 0 0 0 255 0 0 0 0 0 0 0 3 3 28 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 36 4 0 0 0 22 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Driver Implementation

REDEFINING MOBILITY

Packet Filtering and ARP Offload – Driver Implementation

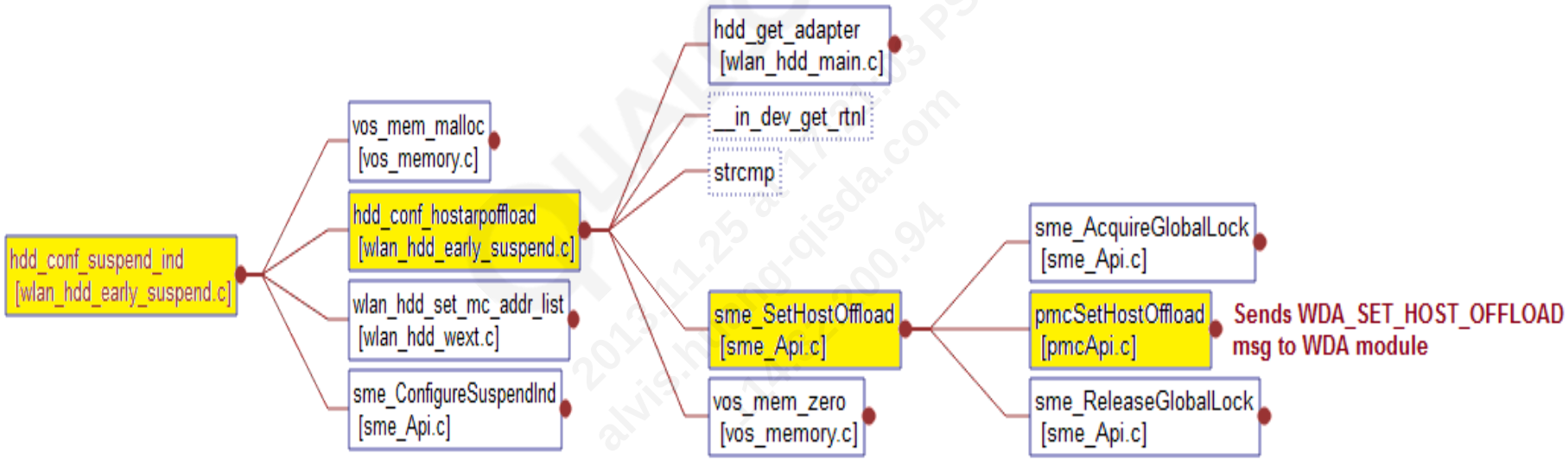
- Whenever the system goes into early suspend (screen off), wlan driver will apply the filters based on config params



- Based on ini params appropriate function is called in early suspend to filter packets
  - hdd_conf_hostarpoffload() - To enable ARP Offload feature
  - wlan_hdd_set_mc_addr_list() - To allow Mcast RX packets from registered addresses only and filter remaining Mcast and Bcast Rx packets
  - sme_ConfigureSuspendInd() - To filter Mcast / Bcast Rx packets completely

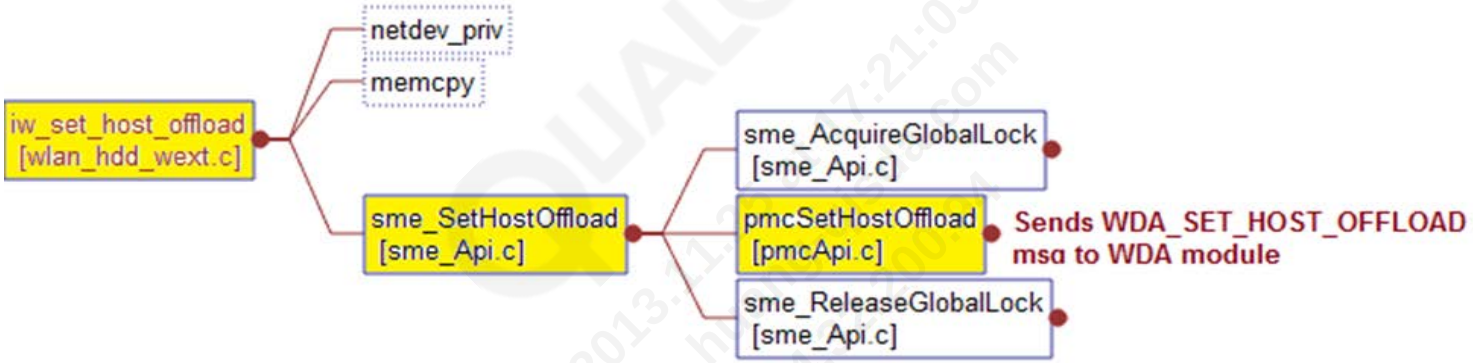
Call Graph - ARP Offload (config param)

- Call Graph for setting ARP Offload feature through ini param **hostArpOffload**



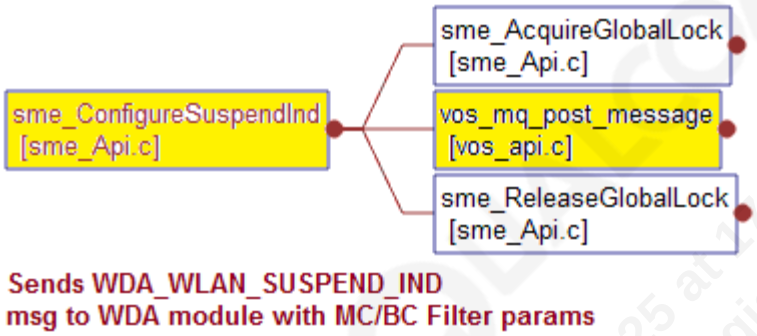
Call Graph - ARP Offload (IOCTL)

- Call Graph for setting ARP Offload feature through IOCTL



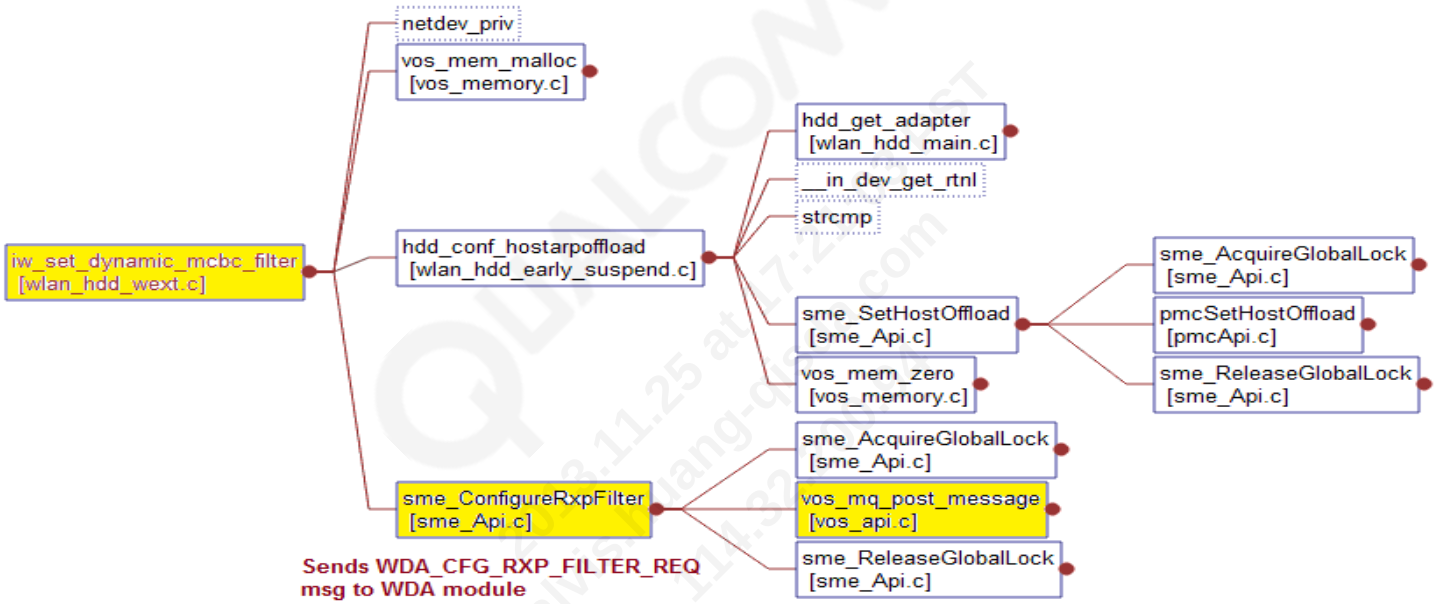
Call Graph - MC/BC Packet Filtering (Config Param)

- Call Graph for filtering Mcast/Bcast pkts completely through ini param **mcastBcastFilter**



Call Graph - MC/BC Packet Filtering (IOCTL)

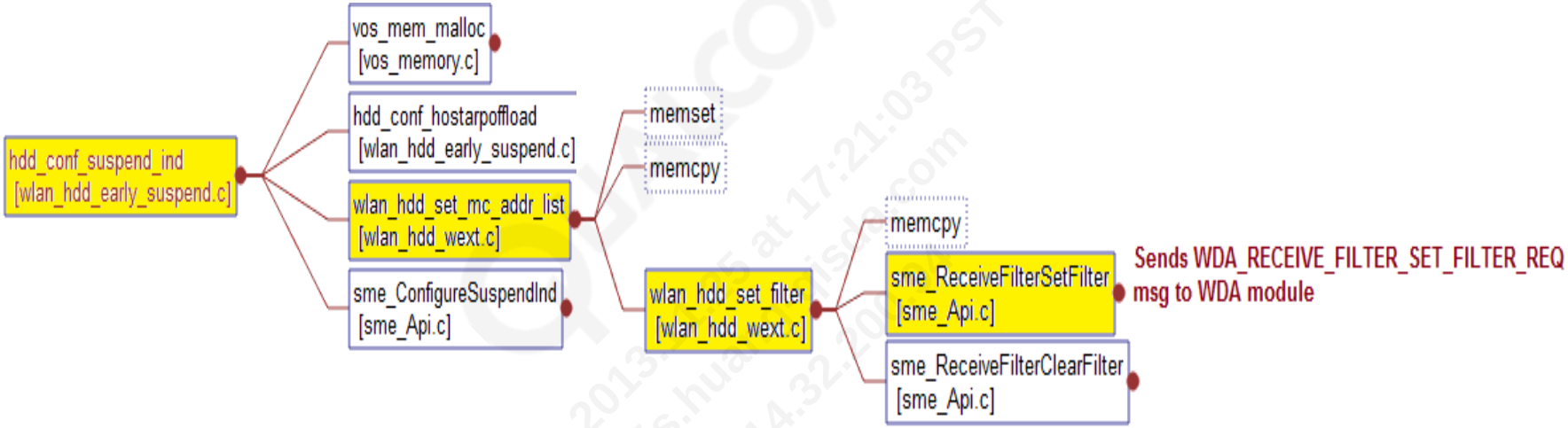
- Call Graph for filtering Mcast/Bcast pkts completely through IOCTL



- The above function call flow will happen if WLAN is already in suspended state
- Otherwise the filter is set as part of next suspend event
  - Similar to the call graph corresponding to ini param **mcastBcastFilter**

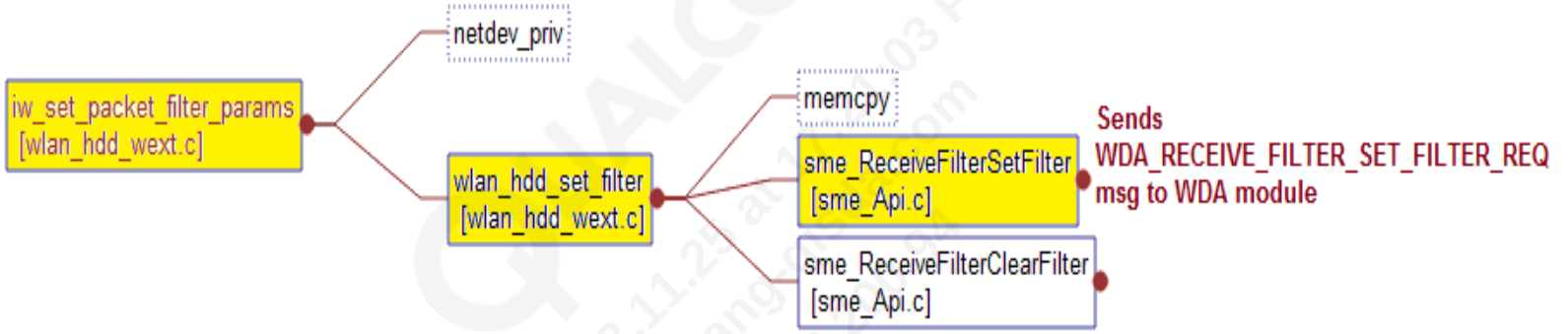
Call Graph - Packet Filter ALL and allow only from Registered Addresses

- Call Graph for filtering ALL Mcast/Bcast pkts except the Mcast pkts from registered addresses when **isMcAddrListFilter** is set



Call Graph - Packet Filtering Based on Set Rules (IOCTL)

- Call Graph for filtering Mcast/Bcast pkts based on set rules through IOCTL



Preferred Network Offloading (PNO)

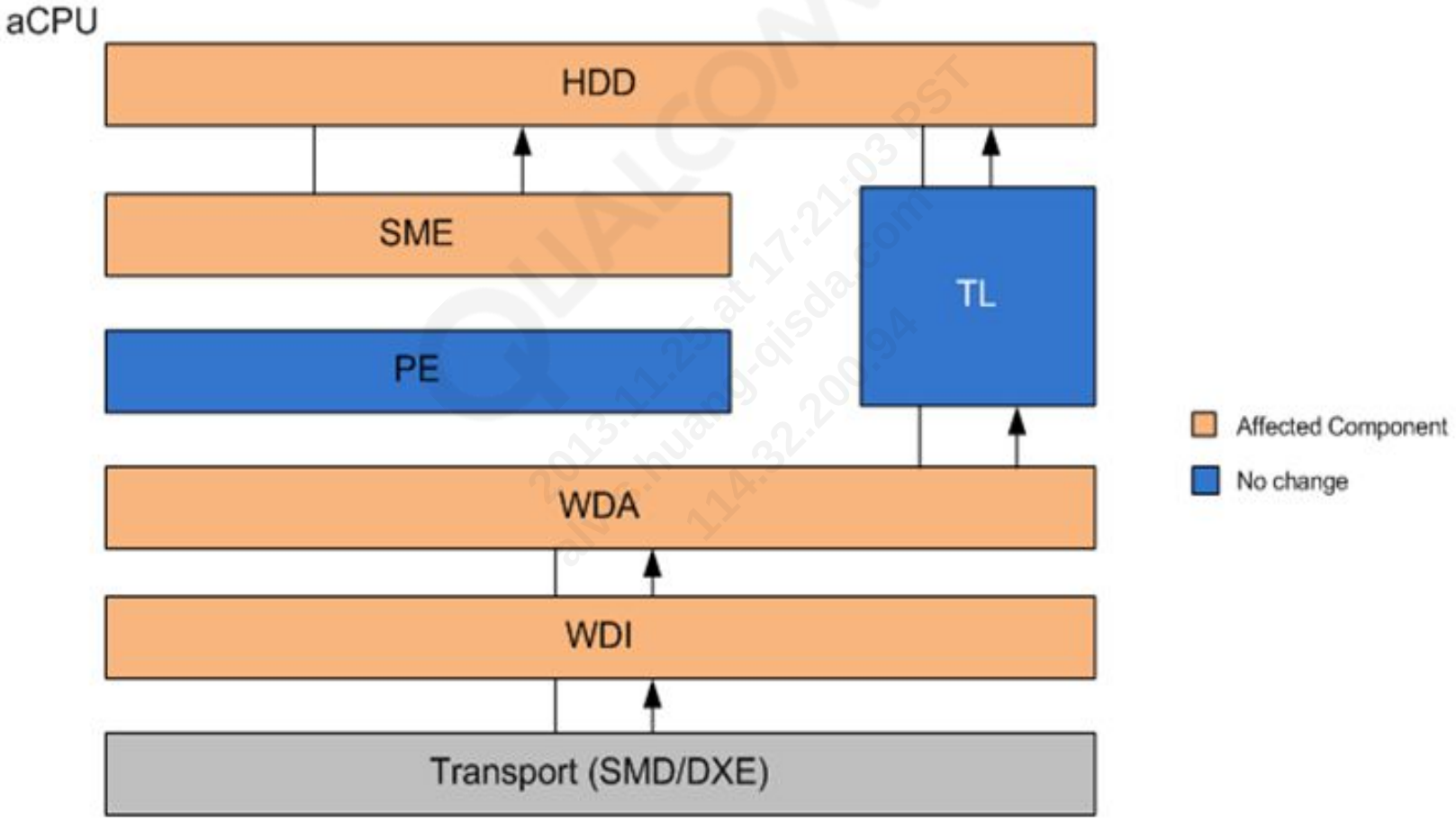
REDEFINING MOBILITY

PNO

- PNO enables a high-level application to request the WLAN firmware to look for networks of choice in an efficient manner.
- The user application can save a list of preferred SSIDs.
- WCNSS FW software scans for such SSIDs based on a higher level application request, provided that the system is in the early suspended state, which enables aCPU to preserve power while WCNSS looks for a preferred network.
- WCNSS will notify aCPU if any of the saved SSIDs are found OTA.
- Private IOCTL is exposed from driver to enable/disable PNO.
- The PNO default scan interval is 5 seconds if the PNO scan timer is not set.
- PNO scans for preferred networks start at aCPU early suspend state and stop as soon as aCPU in late resume state.

Host-Affected Components

- PNO code is featurized with the FEATURE_WLAN_SCAN_PNO flag



Host-Affected Components *(cont.)*

- HDD
  - Exposes configuration IOCTL to set up preferred network list either via private IOCTL with “pno” command or via extended private IOCTL, WLAN_SET_PNO (SIOCIWFIRSTPRIV + 24)
  - Sends IWEVCUSTOM event to supplicant to notify preferred network found
  - wlan_hdd_wext.c
    - iw_set_pno function handle PNO requests
    - found_pref_network_cb function sends IWEVCUSTOM event to supplicant
- SME
  - Updates scan parameters to WCNSS initially and as soon as changes occur
    - csrApiScan.c, pmcUpdateScanParams in csrApplyChannelPowerCountryInfo when changes occur
  - Posts WDA_SET_PNO_REQ message to WDA module with PNO settings
  - Callback HDD when getting indication eWNI_SME_PREF_NETWORK_FOUND_IND with preferred network found
- WDA/WDI
  - Sends PNO request to WCNSS via SMD
  - Handles WDI_PREF_NETWORK_FOUND_IND indication from WCNSS when preferred network found and sends message eWNI_SME_PREF_NETWORK_FOUND_IND to SME

PNO Parameters

- <enabled>: 0- disable, 1- enable
- <network count>: Number of networks in preferred list
- For each network
  - <ssid length>: SSID length of the preferred network
  - <ssid>: Preferred network SSID
  - <authentication>: Authentication (0 – any authentication). Please refer to wlan_hal_msg.h, tAuthType type
  - <encryption>: Encryption (0 – any encryption). Please refer to wlan_hal_msg.h, tEdType type
  - <ch_num>: Number of channels. If 0 is mentioned, then all 2.4G/5G channels will be scanned.
  - <channel_list optional>: Channel number to scan. If 0 is set for <ch_num>, then this can be skipped
  - <bcast_type>: preferred SSID is broadcast or hidden.
    - 0: Don't know: Broadcast probe will be send first and if no probe response is received then a directed Probe request will be send
    - Broadcast: Only broadcast Probe requests will be send
    - Hidden: Only directed Probe requests for that SSID will be send
  - <rssi_threshold>: RSSI threshold, beyond this RSSI value the network will not be reported to APPS
- <scan_timers>: Number of scan timers. Max number of scan timers is 10.
- For each scan timer
  - <scan_time>: scan interval in secs
  - <scan_repeat>: Number of times to scan using the scan interval. 0 means scan continuously till PNO disabled
- <PNO mode>: 1 – to start PNO when aCPU in early suspend state and stop when in late resume state

PNO Command Line Test

- adb shell iwpriv wlan0 setpno "1 1 5 Prima 0 0 3 1 6 11 0 90 2 45 8 300 0 1"
  - ‘1’: Enable PNO
  - ‘1’ :1 network in preferred list
  - ‘5’: SSID “Prima” length is 5
  - ‘4’:‘Prima’ as preferred network SSID
  - ‘0’: any authentication
  - ‘0’: any encryption
  - ‘3’: Number of channels is 3 as channel 1, 6 and 11.
  - ‘1 6 11’: scan channel 1, 6 and 11
  - ‘0’: Don’t know: Broadcast probe is sent first, and if no probe response is received, then a directed Probe request is sent
  - ‘90’: RSSI threshold as -90
  - ‘2’: Number of scan timers is 2
  - ‘45’: First timer scan interval is 45 seconds
  - ‘8’: scan 8 times every 45 seconds
  - ‘300’: Second timer scan interval is 300 seconds
  - ‘0’: scan every 300 seconds until PNO is disabled
  - ‘1’: PNO scan to work in aCPU late suspend mode

Integrate PNO in Android

- Android framework has background scan implementation which relies on Wifi PNO or RTC timer itself to scan periodically for preferred network
  - Set config_wifi_background_scan_support flag in frameworks/base/core/res/res/values/config.xml to define using Wifi PNO or RTC timer periodic wakeup scan
    - **true**: wifi chipset supports background scanning mechanism (PNO); this mechanism allows the host to remain in suspend state and the wlan firmware to actively scan and wake the host when a configured SSID is detected; set to true to use Wifi PNO
    - **false**: default value; use RTC timers to do periodic wakeup scan
- wpa_supplicant has android_pno_start and android_pno_stop function attached to framework to enable and disable PNO
  - In external/wpa_supplicant_8/src/drivers/driver_nl80211.c, customers have to change these functions' implementation for WCN solution PNO parameters
- wpa_supplicant has to handle custom event received from driver
  - Customers have to modify wpa_supplicant to handle the event accordingly

MSM8974 WCN-SS Power Optimizations

REDEFINING MOBILITY

Power Management Changes for MSM8974 WCN-SS

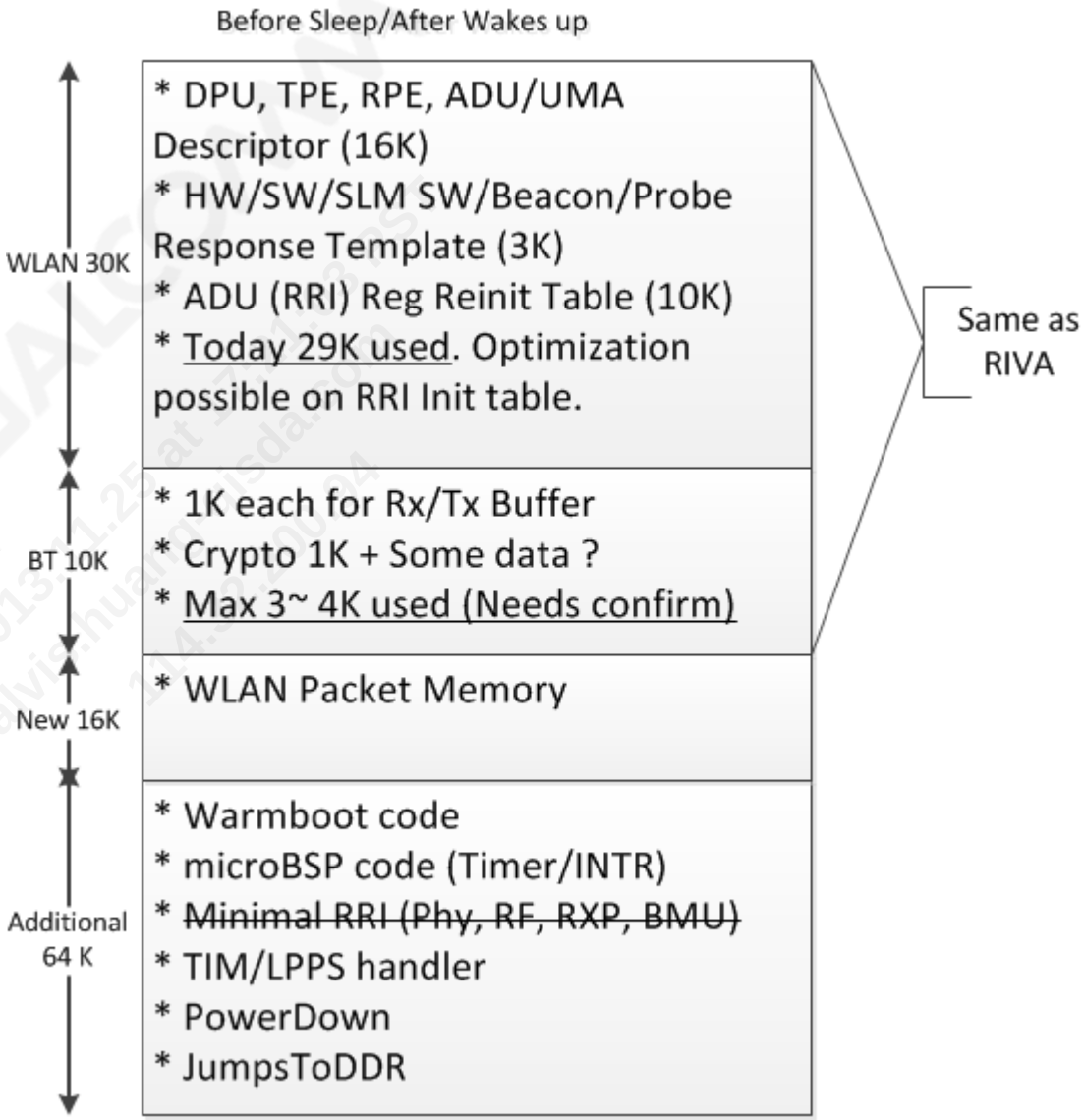
- SPM (Subsystem Power Manager)
  - SPM controls the IRIS XO, PLL
- cMEM
  - Limit access to the MSM DDR during low-power mode of operation
  - Increase the cMEM size by an additional 64 KB
    - Total cMEM is 120 KBytes (56 KB + 64 KB)
    - uBSP running from cMEM to support WLAN BMPS and BT LPPS operations

SPM (Subsystem Power Manager) Updates for MSM8974 WCN-SS

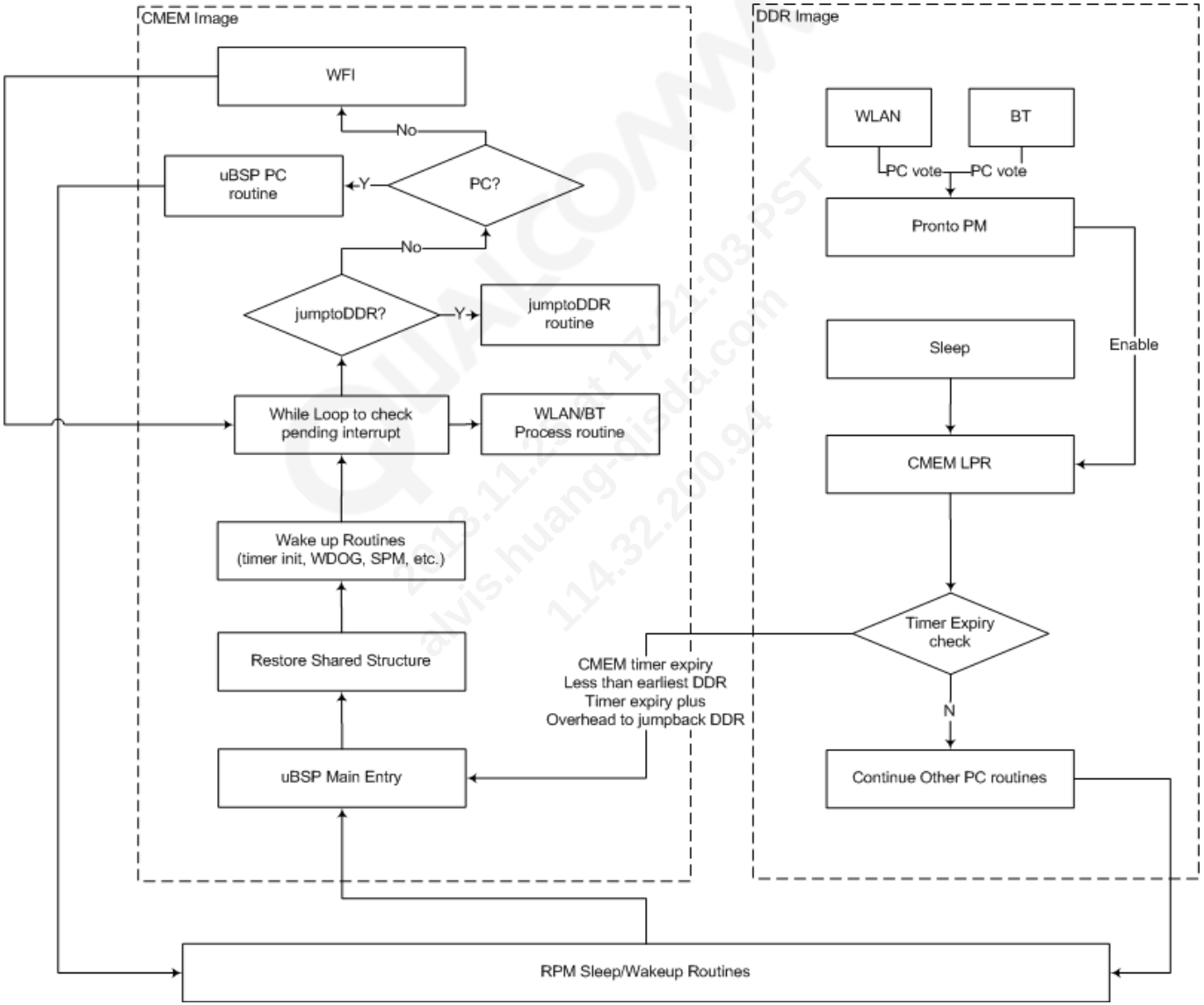
- Change to the warm-boot sequence
  - Wakeup from RPM/aCPU triggers SPM
  - SPM starts the power-up sequence
    - SPM performs handshake with RPM to request resources
    - RPM enables WCN resources and acknowledges the request
    - SPM configures 5-wire WLAN GPIOs to prepare the 48 MHz Iris XO
    - SPM configures WCN36x0 clock plan and enables the 48 MHz XO (if used)
    - SPM enables WCN-SS 960 MHz PLL
  - ARM9 starts fetching instructions from DDR/cMEM

cMEM (additional 64KB) Updates for MSM8974 WCN-SS

- Low-Power Mode
  - Minimize DDR access to reduce the power consumption in the lower power mode (WLAN BMPS and BT LPPS)
  - uBSP code will run in cMEM space in the lower-power mode



uBSP in Low-Power Mode for MSM8974 WCN-SS



WLAN/BT Low-Power Mode for MSM8974 WCN-SS

- WLAN Low-power mode
  - WLAN BMPS mode handling in cMEM
    - TIM SET (Unicast Data Pending) – Fetch the instructions from DDR
    - DTIM SET (B/Mcast Data Pending) – Fetch the instructions from DDR
    - TIM CLEAR – Power Down (no instruction fetching from DDR)
- BT Low-power mode
  - BT LPPS (Low-Power Page Scan) handling in cMEM
    - 1.28 Seconds Timer Interrupt

References

Ref.	Document	
Qualcomm		
Q1	*Application Note: Telescopic Beacon Wakeup*	80-N7461-1
Q2	*WCNSS Android™ Configuration Guide*	80-N8140-1
Q3	*WCNSS Windows Configuration Guide*	80-N5047-23

Questions?

REDEFINING MOBILITY