



QRD8916 Software Manual

Software Product Document

80-NP616-2 A

July 8, 2014

Submit technical questions at:
<https://support.cdmatech.com/>

Confidential and Proprietary – Qualcomm Technologies, Inc.

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

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

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

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

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

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

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

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

**© 2014 Qualcomm Technologies, Inc.
All rights reserved.**

Contents

Part 1 – From Concept to Production..... 8

1 Product Overview 8

- 1.1 QRD8916 default design, implementation, and testing 8
- 1.2 MSM8916 features 8
- 1.3 QRD8916 software features..... 11

2 Software Customization Guidelines 13

- 2.1 Hardware component changes 13
- 2.2 Camera, LCD, and CTP 13
- 2.3 Component test procedures..... 15
- 2.4 User Interface (UI) customizations..... 15
- 2.5 Commercial release configurations..... 16

3 CMCC Test Guidelines 18

- 3.1 General Settings 18
- 3.2 Wireless Communication..... 20
- 3.3 Field test items 25
- 3.4 Service application test items..... 26
- 3.5 Software reliability test items 38
- 3.6 Hardware reliability test items 38

4 CTA Guidelines..... 39

- 4.1 WCDMA..... 39
- 4.2 TDSCDMA..... 40
- 4.3 TD-LTE 40
- 4.4 USIM NI-UICC 41

5 RF Software Changes 42

- 5.1 RF card..... 42

5.2 Characterizatoin and Calibration	43
5.3 General bringing up steps	45
6 Factory Tools	46
7 NV and EFS Tips	47
7.1 Adding an EFS NV item	47
7.2 Converting QCN to XML file for easy edit	49
7.3 WCNSS_qcom_config.ini	49
Part 2 – Beginner’s Guide to QTI	58
8 Software Installation and Setup	58
8.1 Installation and Setup	58
8.2 Downloading QTI software from ChipCode	66
8.3 Firmware Programming	74
8.4 Operational Guide	87
A Android Device Tree Structure	112
A.1 Android target tree structure	112
A.2 Building Tiny Android	113
A.3 Building the Linux kernel manually	113
A.4 Building Android manually	114
A.5 Other important Android build commands	114
9 Driver Porting Guide	117
9.1 CDT	117
9.2 Device Tree	120
9.3 GPIO	123
9.4 I2C	128
9.5 UART	136
9.6 Serial Peripheral Interface	147
9.7 Display	157
Part 3 – UI/FRWK Changes	170

10 Runtime Feature Control.....	170
10.1 Overview.....	170
10.2 How to work	171
10.3 How to make a system.img	171
10.4 How to use the tools to switch the carrier	173
11 CMCC DSDX New Feature.....	178
11.1 CMCC Dual SIM status bar	178
11.2 CMCC Dual SIM DM feature.....	179
11.3 CMCC Dual SIM SMS/MMS support.....	179
11.4 Supporting to close all running tasks in task management	181
11.5 Manual or automatical configuration of multi-mode dual SIM capability according to PLMN and UICC	182
11.6 CMCC network mode	184
12 CTA New Feature.....	189
12.1 Adding CTA dynamic overlay mechanism.....	189
12.2 CTA DSDX feature	189
12.3 CTA security.....	190
13 SWE Browser	194
13.1 CAF.....	194
13.2 To build SWE from GAF.....	194
13.3 SWE Resource overlay	195
14 Feature Gap of KK and JB	196
14.1 New features related quick search bar	196
15 CT New Feature	198
16 CU New Feature	199

Figures

Figure 6-1 SGLTE Test RF Connection	39
Figure 10-1 Build ID naming convention	63
Figure 10-2 Combined software release packages.....	64
Figure 10-3 QPST configuration.....	80
Figure 10-4 Emergency download port	81
Figure 10-5 Flash CDT with QPST	82
Figure 10-6 QRCT NV tool for generating QCN	88
Figure 11-1 Device tree in boot image.....	120
Figure 12-1 System image component.....	170
Figure 12-2 Carrier configure setting.....	173
Figure 12-3 Changing button of CMCC/DSDA	174
Figure 12-4 Switching the CMCC	175
Figure 13-1 SGLTE dual SIM status bar style.....	179
Figure 13-2 Dual SIM SMS/MMS interface.....	180
Figure 13-3 Task manager icon	181
Figure 13-4 Task manager behavior	182
Figure 13-5 Multi-SIM configuration.....	183
Figure 13-6 Multi-SIM switch behavior	183
Figure 13-7 Test mode interface	185
Figure 13-8 Set network feature interface.....	186
Figure 13-9 Modem test menu interface	187
Figure 13-10 Default mobile network menu interface.....	188
Figure 14-1 Dual SIM UPLMN interface	189
Figure 14-2 New a UPLMN	190
Figure 14-3 DM End User notifications	191
Figure 14-4 DM enable/disable switch	191
Figure 14-5 CTA app security switch.....	192
Figure 14-6 CTA app security behavior	193
Figure 16-1 APP local search interface.....	196
Figure 16-2 Call log search interface.....	197
Figure 17-1 SMS center number editing interface.....	198

Tables

Table 3-1 Mode settings for CMCC wireless communication test items	20
Table 10-1 Required equipment and software	58
Table 10-2 Release packages	63
Table 10-3 Component release build properties	64
Table 10-4 Required software.....	74
Table 10-5 RF configuration.....	87
Table 10-6 WCDMA/GSM + GSM DSDS NV settings.....	88
Table 10-7 CDMA + GSM DSDS NV Settings	89
Table 10-8 LTE + GSM DSDS settings.....	90
Table 1-1 Register table in LK.....	121
Table 1-2 Device tree key API.....	122
Table 1-3 GPIO physical address.....	123
Table 1-4 GPIO CFG table	123
Table 1-5 GPIO interrupt CFG table.....	124
Table 1-6 INT CFG.....	124
Table 1-7 GPIO wakeup interrupt.....	125
Table 1-8 QUP physical address	128
Table 1-9 QUP IRQ	128
Table 1-10 UART physical address	136
Table 1-11 UART interrupt table.....	136
Table 1-12 BLSP BAM physical address	142
Table 1-13 BAM pipe	142
Table 1-14 Base address	147
Table 1-15 BLSP BAM physical address	154
Table 1-16 BAM pipe assignment	154
Table 9-17 Display properties.....	157
Table 12-1 System image component.....	171
Table 13-1 Signal strength level	178

Revision history

Revision	Date	Description
A	July 2014	Initial release

Note: There is no Rev. I, O, Q, S, X, or Z per Mil. standards.

QUALCOMM®
106.37.221.50 2014.11.17 at 17:13:11 PST
xumingtao@hipad.com

Part 1 – From Concept to Production

1 Product Overview

1.1 QRD8916 default design, implementation, and testing

1.2 MSM8916 features

Mobile devices continue to integrate more and increasingly complex functions, and support more operating bands while maintaining performance, board space, and cost

These demands are met by the Qualcomm Technologies, Inc. (QTI).

MSM8916 – with its quad-ARMCortex-A53 application processors – which further expand mass-market chipset capabilities by making 3G and 4G high-speed data and rich multimedia features accessible to more consumers worldwide.

Depending upon the IC variant, this multimode solution supports the latest air interface standards including 1xEV-DO, 1x Advanced, HSPA+ and TD-SCDMA, LTE as well as single SIM, dual SIM dual standby (DSDS), dual SIM dual active (DSDA) and triple SIM triple standby (TSTS).

One baseband receiver port and one baseband transmitter port enable simultaneous voice and data operation for user multitasking. The new MSM8916 leverages QTI airlink and multimedia technology leadership to significantly lower the cost of high-performance mobile devices.

The MSM8916 has a high level of integration that reduces the bill-of-material (BOM), which delivers board-area savings. The cost and time-to-market advantages of this IC will help drive wireless broadband adoption in mass markets around the world.

Wireless products based on the MSM8916 may include:

- Voice and data phones, smartphones
- Support for the latest, most-popular operating systems
- Music player-enabled devices and applications
- Camera phones
- Multimedia phones with gaming, streaming video, and video conferencing features
- GPS, GLONASS, and Beidou for global location-based service. Supports 3 band only.
- Wireless connectivity – NFC (QCA1990), Bluetooth?, WLAN, and FM receiver (with the WCN3620)
- The MSM8916 benefits are applied to each of these product types and include:
 - Higher integration to reduce PCB surface area, time-to-market, and BOM costs while adding capabilities and processing power
 - Integrated quad-application processors and hardware cores eliminate multimedia coprocessors, providing superior image quality and resolution for mobile devices while extending application times
 - Higher computing power for high-end applications, and DC power savings for longer run times
 - Position location and navigation systems are supported via the WTR's global navigation

- satellite system (GNSS) receiver
- The MSM8916 supports Gen 8C operation
- Single platform that provides dedicated support for all market-leading codecs and other
- multimedia formats to support carrier deployments around the world
- DC power reduction using innovative techniques

The MSM8916 is fabricated using the advanced 28 nm LP CMOS process, and is available in the 760 NSP (a 14 x 12 x 0.96 mm package with many ground pins for improved electrical grounding, mechanical strength, and thermal continuity).

The MSM8916 supports high-performance applications worldwide using a variety of wireless networks (depending upon IC variant):

- GSM/GPRS/EDGE
- CDMA20001x, 1x Advanced and 1xEV-DO Rev A
- WCDMA R99, Rel 5 HSDPA, Rel 6 HSUPA, and Rel 7 HSPA+ (42 Mbps)
- TD-SCDMA with 4.2 Mbps DL and 2.2 Mbps UL option
- LTE Cat 4
- GPS, GNSS, and Beidou
- Complementary ICs within the MSM8916 chipset include:
 - Wafer-level RFICs: WTR1605L(80-N5420-x) and WTR2605 (80-N9978-x documents)
 - Power management: PM8916 (80-NK807-x documents)
 - Wireless connectivity: WCN3620 for WLAN, Bluetooth, and FM (80-WL006-x documents)
 - NFC Connectivity: QCA1990 (80-Y0597-x)

The MSM8916 chipset and system software solution supports the Convergence Platform for mobile applications by leveraging the years of systems expertise and field experience with CDMA, WCDMA, GSM, TD-SCDMA, LTE and GNSS technologies that QTI brings. QTI works with its partners to develop products that meet the exact needs of the growing wireless market, providing its customers with complete, verifiable solutions, including fully segmented product families, systems software, testing, and support.

1.3 QRD8916 software features

Feature Name	Comments
Common sensor HAL for SSC and Native sensor driver	In mainline. No way to disable
Crash Dump/Compressed Crash Dump	In mainline. Disable SBL debug/crash dump function or put Logkit to quite mode.
Fast Start for Factory Testing	In mainline. Only will be enabled during flag is set on misc partiiton.
Enable AP and MP upgrade from SD card	In mainline. Only will be enabled during MP images are in update package
Crash Log Record	In mainline. Put Logkit to quite mode to disable it.
Generic sensor HAL for different vendor sensors	In mainline. No way to disable
Kernel Logging	In mainline. Put Logkit to quite mode to disable it.
Log upload	In mainline. Put Logkit to quite mode to disable it.
QXDM Logging	In mainline. Put Logkit to quite mode to disable it.
Support Firmware Over the Air (FOTA)	In mainline. No way to disable
LA HAL Sensor Fuzzing	In mainline. No way to disable
Power Off Alarm in QuickBoot mode	In mainline. No way to disable
Power Off Charging in QuickBoot mode	In mainline. No way to disable
Long Press Power Up	In mainline. No way to disable
Power on Vibration	In mainline. No way to disable
Power Off Charging	In mainline. fastboot oem commands
Support for non standard USB chargers(kernel is impacted team)	In mainline. No way to disable
Add baseband subtype support in Boot Loader	In mainline. No way to disable
Fast power-on	In mainline. Disable the feature on UI.
Alarm Clock when power off (RTC alarm wakeup)	In mainline. No way to disable
Sub platform type support in Linux kernel	In mainline. No way to disable
China CTA security requirement (L1)	In mainline. Disable appops.
Chinese Language support for Fastmmi	In mainline and wait for external release. No way to disable
RTC Alarm driver unit-test and SMP fix	In mainline. No way to disable
Add on device logging for QDSS on Android(QDSS Agent)	In mainline. Put Logkit to quite mode to disable it.

Feature Name	Comments
NFC case implementation for Fastmmi	In mainline. Only will be enabled during flag is set on misc partiiton.
NV operation interface and new diag command implementation in Fastmmi	In mainline. Only will be enabled during flag is set on misc partiiton.
PSensor calibration in Fastmmi	In mainline. Only will be enabled during flag is set on misc partiiton.

2 Software Customization Guidelines

2.1 Hardware component changes

Qualcomm provides detailed component list to customers. It can help customer speed up the development. Beside, we also provide general guideline about driver porting and development. You can refer to Appendix C – Driver Porting Guide. In this document, you can get detail guide about Device Tree, GPIO, I2C, UART, SPI, SSC, Camera, Display and PMIC

QRD Component list

80-VL976-6 QRD HARDWARE COMPONENT VERIFICATION SCHEDULE provide full list of all components, it include sensor, display, camera, CTP, Memory.

You can get component driver from GCDB – Global Component Database: <https://createpoint.qti.qualcomm.com>

80-NC193-10 provide detail guide for GCDB usage.

2.2 Camera, LCD, and CTP

80-NN766-1: Linux Android Display Driver Porting Guide describe detail process to port display driver. It contains steps below:

1. Download verified lcd driver from GCDB web: <https://createpoint.qti.qualcomm.com>.
2. To complete same model merge the driver directly. for different models, download reference driver with same chipset and IC vendor, then modify lcd parameter and save to XML.
3. Use GCDB script generate dtsti and .h.
4. Merge dtsti and .h into kernel code.
5. Rebuild kernel and LK image.
6. Debug kernel lcd driver firstly. and then debug LK and the continuous display in LK + Kernel, at last.

7. Before debugging kernel lcd driver, close LK display and contiouns disply cont_splash.

How to use GCDB script develop lcd driver:

- GCDB script is a Qualcomm provided tool for customers to speed lcd driver development.
- GCDB script locate at [Android Root]/device/qcom/common/display/tools.
- GCDB script command sample: `#perl parser.pl <.xml> panel`
- XML file is a user defined input file. It includes all lcd parameter. You can refer to 80-BA103-1: Display GCDB XML Entries
- Recommend you to downlod reference XML from GCDB and then change it based on your design.

How to use Camera sensor:

1. Confirm whether the sensor is in Qualcomm verification list (refer to Solution 0028472).
2. Download verified driver from GCDB web:<https://createpoint.qti.qualcomm.com>.For any download issues, contact GCDB web.
3. If it is an non-verified sensor, refer to Solution 00028473 to create and debug sensor. The followings are the general debug steps:
 - a. Chek power supply, GPIO setting, clock.
 - b. Check power on/off sequence.
 - c. Check MCLK and GPIO setting.
 - d. Check CCI register.
 - e. Confirm CCI register.
 - f. Check CCI config setting.
 - g. Verify CSIPHY/D.
 - h. Confirm CSIPHY/D is compatible with MIPI connection.
 - i. AF motor debug.
 - j. EEPROM debug.

For more information, refer to 80-NL239-32 : Sensor Module Bringup.

4. To confirm timing have oscilloscope prepared.

2.3 Component test procedures

Functional area	Document	DCN
LCD brightness, Contrast ratio, NTSC ratio, Uniformity, Gamma	QRD Component Lab LCD Modules Test Procedure	80-VL976-1
CTP proximity, Accuracy, Jitter	QRD Component Lab Capacitive Touch Panel Test Procedure	80-VL976-2
Camera	QRD Component Lab Camera Module Test Procedure	80-VL976-3
Native I2C-sensor (ACC, mag, gyro, light, and proximity)	QRD Component Lab I2C-Sensor Validation Process	80-VL976-8
RF	QRD Component Lab – RF Component Test Plan (detailed test items for CDMA/UMTS/GSM)	80-VL976-4
DDR/eMMC	QRD Component Lab Memory Test Procedure	80-VL976-5

2.4 User Interface (UI) customizations

QRD implement many features based on CMCC/CTA requirement. You can find more detail information in Section 12 – 16.

2.5 Commercial release configurations

Some features are intended for use only during the development and debugging stages. These features must be disabled prior to commercial release to improve performance and user experience.

Disabling kernel debug feature to optimize HLOS performance

By default, the kernel configuration file includes many settings that are useful during the development and debugging phase but may affect performance.

To disable the default kernel debugging settings, use `LINUX/android/kernel/arch/arm/configs/msm8226-perf_defconfig` instead of `msm8226_defconfig`.

Enabling Subsystem Restart (SSR)

When a subsystem (mode, wifi, video core, etc.) crashes, the subsystem can be configured to automatically notify the AP. The AP then restarts the applicable subsystem. This provides a better user experience.

1. To enable subsystem restart prior to commercial release, open the `android/device/qcom/msmXXXX/system.prop` file.
2. Modify the `persist.sys.ssr.restart_level` property as follows:
`persist.sys.ssr.restart_level=modem,wcns,adsp,venus or 1`

This enables subsystem restart for modem, wifi, adsp, and video subsystems. This is the recommended setting for commercial release.

During engineering development, this property should be set to `1` as shown below. This enables subsystem panic which is useful for engineering releases.

```
persist.sys.ssr.restart_level=1
```

Subsystem panic means that when each subsystem encounters a problem which triggers a reset, the AP will catch this event and will call kernel panic. As a result, the device can enter download mode to capture ramdump.

Disabling Download Mode

During development and debugging, Download mode is used to get memory dump for crash analysis. For a production device, Download mode is not needed since the device should just reboot.

Use one of the following methods to disable download mode:

- Modify the `arch/arm/mach-msm/restart.c` file as follows:

```
static int download_mode = 1;  change 1 -> 0
```

Modifying the `download_mode` value is the preferred method because you have the ability to dynamically obtain a memory dump for a launched phone. Download mode can be enabled from the shell:

```
echo 1 > /sys/module/restart/parameters/download_mode
```

- Remove `CONFIG_MSM_DLOAD_MODE=y` from
`LINUX/android/kernel/arch/arm/configs/msm8226_defconfig`

Using this method, Download mode is completely disabled. It cannot be enabled dynamically so you will not be able to switch to Download mode for image programming.

If needed, Firehose can be used for image programming. QFIL (QPST 2.7 Build 416 or later) supports Firehose. QFIL uses `adb reboot edl` to switch to emergency download mode.

3 CMCC Test Guidelines

CMCC has the following test categories:

- Wireless communication
- Field test
- Service application
- Software reliability
- Hardware reliability

Different settings for detailed test items are introduced.

Since CTA tests are similar to CMCC test, see Chapter 6 for information on the CTA test requirements that differ from CMCC.

3.1 General Settings

Firstly, general QCN/EFS/Policyman is introduced. With this general setting, most CMCC/CTA test items could be passed.

3.1.1 QCN/EFS setting

1. Set TDSCDMA/TDLTE related NV/EFS. See *TD-SCDMA NV/EFS Configuration* [80-NG442-1]. For MSM8916 (DPM 1.0) and later PLs (refer to the respective release notes for information on one binary feature support status), refer to Chapter Two of [*TD-SCDMA NV/EFS Configuration* [80-NG442-1].
2. For DSDS NV/EFS configuration, refer to 80-NM375-1_L_G_DSDS_NV_Settings_DI3x.
3. See *Coex EFS Dual Technology Operation* [80-NH099-1] for information on reading/writing EFS file.

3.1.2 SGLTE policy file for field testing

Taking single card project policy file as example. For multi card project such as CSFB+G DSDS, find respective policy files from respective folders.

1. Go to modem build:

```
\modem_proc\mmcp\policyman\configurations\Carrier\CMCC\SGLTE\  
CMCC_procured\carrier_policy.xml
```

2. Copy carrier_policy.xml to EFS\policyman\ carrier_policy.xml.

An example of the default SGLTE phone carrier_policy.xml file is shown below. The key components are that need to be modified are shown in **red** font.

```
<mcc_list name="sglte_mccs"> 460 </mcc_list>  
<!-- These are the operators (IMSI PLMNs) for which SGLTE  
will be allowed -->  
<plmn_list name="sglte_operators"> 460-00 460-02 460-07 460-08  
  </plmn_list>  
<!-- Define the OOS timer with a 5 minute interval -->  
<!-- NOTE: Proper functioning of the SGLTE policy requires  
that there be a timer named "oos". Do NOT rename this timer. -->  
<define_timer name="oos" units="min" interval="5" />
```

3.1.3 SGLTE policy file for lab testing

Taking single card project policy file as example. For multi card project such as CSFB+G DSDS, find respective policy files from respective folders.

1. Go to your modem build

```
\modem_proc\mmcp\policyman\configurations\Carrier\CMCC\SGLTE\test
```

2. Copy carrier_policy.xml to EFS\policyman\ carrier_policy.xml.

3. For most lab tests using SGLTE phone mode, the items shown in **red** need to be modified:

```
<mcc_list name="sglte_mccs"> 001 002 003 004 005 006 007 008 009 010 011
012 246 316 440 460 466 </mcc_list>
<!-- These are the operators (IMSI PLMNs) for which SGLTE will be
      allowed -->
<plmn_list name="sglte_operators"> 440-79 001-01 460-00 460-01 460-02
460-03 460-04 460-07 460-09 460-08 460-080 460-71 002-02 002-81 002-91
002-11 466-02 466-09 246-81 246-081 </plmn_list>
<!-- Define the OOS timer with a 20 minute interval -->
<!-- NOTE: Proper functioning of the SGLTE policy requires that there
be a timer named "oos". Do NOT rename this timer. -->
<define_timer name="oos" units="min" interval="20"/>
```

3.2 Wireless Communication

For all lab wireless communication tests, be sure to disable supplementary service.

For wireless communication testing, information on basic mode setting for CMCC cases are shown in the following table.

Table 3-1 Mode settings for CMCC wireless communication test items

	Test Item	CMCC Version	Mode Setting	Comments
Wireless Communication (TD-SCDMA)	2G/3G Protocol Conformance	V2.0-20130219	T+G mode	
	RRM Conformance	V2.0-20130219	T+G mode	
	TD internal protocol Conformance	V2.0-20130219	T+G mode	
	(U)SIM Compatibility	V2.5-20130815	SGLTE mode	Use default mode of submitted samples, no special requirement

	Test Item		CMCC Version	Mode Setting	Comments
	Basic Communication -- DSDS		V2.5-20130815	SGLTE mode	Use default mode of submitted samples, no special requirement
	Basic Communication -- DSDA		V2.5-20130815	SGLTE mode	Use default mode of submitted samples, no special requirement
	Basic Communication		V2.0-20130219	SGLTE mode	Use default mode of submitted samples, no special requirement
	Joint Detection		V2.0-20130219	SGLTE mode->T+G mode	Will first try SGLTE mode to do testing. If there is an issue camping on, they will change to T+G/T mode.
	RF Conformance		V2.0-20130219	SGLTE mode->T+G mode	Will first try SGLTE mode to do testing. If there is an issue camping on, they will change to T+G/T mode.
	IOT (A+F)		V2.0-20130219	SGLTE mode->T+G mode	Will first try SGLTE mode to do testing. If there is an issue camping on, they will change to T+G mode
	International Roaming(lab)		V1.0	default mode	Will not change the mode initially because there is no special requirement. If there is an issue camping on, they will change to T+G mode.
LTE Test	LTE FDD Conformance-V3.0	Protocol-FDD	V3.0	CSFB only	
		RF-FDD	V3.0	CSFB only	All FDD-related TC should be covered only in CSFB mode.
		RRM-FDD	V3.0	CSFB only	All FDD-related TC should be covered only in CSFB mode.
		USIM-FDD	V3.0	CSFB only	Latest TC can be used CSFB mode.
	TD-LTE Conformance-V3.0	Protocol-TDD	V3.0	SGLTE/CSFB	For TDD-LTE only, SGLTE or CSFB will be OK. The IRAT part is test in CSFB-only mode
		RF-TDD	V3.0	Applicable for SGLTE	Both SGLTE and CSFB are applicable.
		RRM-TDD	V3.0	Partly applicable for SGLTE	TDD-FDD inter-mode, L2T CS fallback related and all L2G TCs should be covered only in CSFB mode.
		USIM-TDD	V3.0	SGLTE or CSFB	Latest TC can be used for CSFB or SGLTE mode
	TD-LTE IOT	NV-IOT	V2.0	SGLTE or CSFB	

	Test Item		CMCC Version	Mode Setting	Comments
	(Single USIM Dual Standby Handset)-V2.0	NV-IOT IPv6	V2.0	SGLTE/CSFB mode	For 2G/4G, 2G/3G IRAT TC need test with CSFB only mode. CMCC lab use SGLTE test all IPV6 TC include IRAT, and need have clarification if failed for IRAT about type 2 LTE mobile.
		Network Simulator IOT	V2.0	SGLTE or CSFB	Latest TC can be used for CSFB or SGLTE mode
		NS-IOT Throughput	V2.0	SGLTE or CSFB	Latest TC can be used for CSFB or SGLTE mode

3.2.1 TDSCDMA

USIM and user equipment compatibility

Set lab EHPLMN: 460-00, 460-02, 460-07, 460-08, 466-02, 460-29, 001-01.

TDSCDMA RRM conformance

Use SGLTE phone, set mode pref to **T+G** and **CSFB** mode, Combined attach.

Problem	Solution
TDS RRM, UE fails to camp on network in 4G prefer mode, but succeeds in 2G/3G mode	Lab Setup issue: TE limitation, need UE work in CSFB mode to send combined attach when in TDSCDMA.
TDSCDMA TC4.3.8.8: PS domain, data transfer mode, from UTRAN to GPRS IRAT cell reselection delay, it happens 15 times in 87 times test, the TE does not have the Channel request from UE.	Set DTM ON according to QCN/EFS setting.
TC4.3.11.1: TDSCDMA-GSM(GPRS), the measurement accuracy of carrier RSSI, the UE will have it greater than the upper limitation with 101 times during test2.	RF issue could partially pass with very good calibration. It needs to do calibration ARFCN 20 instead of ARFCN 31 in the calibration tree

TDSCDMA RF conformance

Problem	Solution
TDS RF, TC7.2.5 UL Close loop power control, TPC1 fails. TPC1 step gap is -0.02dB, expected scope is -0.5dB~-1.5dB.	Calibration issue.

WCDMA International Roaming

Set UE mode to CSFB.

3.2.2 LTE

For LTE FDD RF/RRM test case, set UE mode to CSFB.

TD-LTE IOT

The `carrier_policy.xml` file should include **460** in `sglte_mccs`, **460-01** and **460-04** in SGLTE PLMN list.

Problem	Solution
LTE NV-IOT, FTP DL average PEAK rate is lower than expectation 40M bps	No solution finally. Always reproduces within unstable CMCC lab, the network configuration always changes
NV-IOT UE cannot register to 4G test cell. From the log, the NV-IOT lab test APN(IMS2) does not work as UI setting, and then the network rejects this PDN.	In the UI, set APN to IMS2 again.
TD-LTE NV IOT IPV6, TD-LTE&TDS, TDS&GSM network reselection fails	Configuration issue which needs the correct <code>carrier_policy.xml</code> with SGLTE Operator 460-01, and the NAS EFS file <code>eia0_allowed</code> and <code>lte_nas_lsti_config</code> . See Section 3.1.1.
LTE NSIOT, UE fails to camp on network during test cases	Configuration issue which needs the correct <code>carrier_policy.xml</code> with SGLTE Operator 460-080.

01-CMCC_LTE_Terminal_Test_Plan-TD-LTE Conformance

Problem	Solution
LTE RRM, UE fails to camp on network during test cases.	Lab Setup issue: CMCC lab uses an incorrect automation script, always order UE to do something before it confirms to TE.
DUT has no action within 12 mins, the testing terminal should send RRC Connection Request immediately.	Real network interference. Need to execute with a good shielding box or in shielding room.

01-CMCC_LTE_Terminal_Test_Plan-LTE FDD Conformance-V3.0

Test area	Recommendation
RRM-FDD	To cover FDD/WCDMA PSHO test cases (LTE FDD RRM TC 5.2.1/5.2.7, etc.), remove the /nv/item_files/modem/nas/tdscdma_op_plmn_list file.
USIM-FDD	<ol style="list-style-type: none">1. Set CSFB mode.2. Modify the values for the following NV items:<ul style="list-style-type: none">▫ NV 69674 GSTK Feature Bitmask =C6▫ NV 65683 QMI CAT mode =2 Android.

3.2.2.1 Key points for LTE protocol test

For SGLTE tests, you need to prepare two types of phones:

- SGLTE phone, voice centric. Used for EPS only attach test.
- CSFB phone, voice centric. Used for combined attach and IRAT test.

The following tests are not performed for SGLTE phone:

- TDD-LTE combined attach test.
- TDD-LTE data centric test, for example: TC9.2.1.2.4 TDD, TC9.2.3.2.4 TDD, TC9.2.3.2.17 TDD.

3.3 Field test items

For field testing, both QXDM and AP side adb logs should be captured. Check the configuration of LTE product (CSFB or SGLTE) before testing.

CMCC scope	
Module	Item
TDS	Domestic -- Beijing
	Domestic -- Guangzhou
	Domestic -- Hangzhou
	Domestic -- Nanjing
LTE	Domestic -- Guangzhou

CMCC scope	
	Domestic -- Hangzhou
	Domestic -- Shenzhen
HSUPA	HSUPA --Beijing
IR	HK NA EU Japan Korea Not all locations are mandatory, related to phone's price. International Roaming needs video recording before PA close.

3.4 Service application test items

QRD CMCC Pretest scope		Comments
Module	Item	
Service and application	AGPS	The phone which are ≥ 800 RMB (or the screen is larger/equal to 4.5 inch), mandatory request. Push APGS Certification.
	Device Manager	SIM card ready(available for DM test, can apply by CMCC PoC) From 2014/2/1, new solution will be implemented.
	WLAN Function	Close all application which will cost extra power. WLAN RF exposed device with root enabled ready.
	WLAN RF	
	MMS	
	MMS Conformance	Lab pretest in CTTL, display issue is focused by tester
	MMS IOT	
	Local Function	
	Streaming	NV69731 set to 1 will lead the PDP issue, please set to 0 only for such testing. Partial cases only can be tested in CMCC lab, hard to book

QRD CMCC Pretest scope		Comments
	broadband Internet	NV69731 set to 1 will lead the PDP issue, please set to 0 only for such testing. Partial cases only can be tested in CMCC lab, hard to book
	CMMB Mobile TV	
	KJAVA	
	Video Telephone	
	NFC	

3.4.1 WLAN

3.4.1.1 Software configuration for RF test

1. DUT needs to be rooted.
2. In the `/data/misc/wifi/WCNSS_qcom_cfg.ini` file, modify the following values to disable Power Save and Roaming.

```
gEnableImps=0
gEnableBmps=0

FastRoamEnabled=0
```
3. Use adb to push the updated ini file back to the DUT.

```
adb root
adb push WCNSS_qcom_cfg.ini /data/misc/wifi
```
4. Turn Wifi off and on, then execute the WLAN RF test.

3.4.1.2 WAPI release process (IMPORTANT)

WAPI requires an additional license in order to obtain the WAPI-related patches. These patches must be applied in the early stages of development. It cannot be done right before CTA test.

To obtain the WAPI patches:

1. Open a Salesforce case requesting the WAPI patches.
QTI Product Management will provide your regional contact with the applicable WAPI letter for customer signature.

2. After customer signature is obtained, an SBA WAPI release is prepared.
You will receive a notification when the WAPI patch is available for download.

3.4.1.3 Troubleshooting typical WLAN software issues

WLAN does not work

When CMCC runs WLAN RX RF performance test in protocol mode, after running a while and increasing the attenuation (to 61db) at the same time, the instrument will report a test failure.

The CMCC lab uses WIFI instrument from Litepoint instead of M8860C now.)

Root cause and solution

Powersave and roaming functions are enabled by default in the DUT, however, these function may lead to the instrument getting no response from the DUT after it has been running a while. As a result, the instrument will make an incorrect decision to disconnect with the DUT.

1. Disable the powersave and roaming functions in the configuration file.
2. Push WCNSS_qcom_cfg.ini file into phone before testing WLAN Rx sensitivity in signaling mode to close Power save mode. See Appendix 9.3 for an example of the WCNSS_qcom_cfg.ini file.

```
adb root
adb remount
adb push WCNSS_qcom_cfg.ini /data/misc/wifi
adb reboot
```

3. Disable GPS relative scanning by deleting the following files:

```
/system/bin/lowi_test
/system/bin/lowi-server
/system/etc/lowi.conf
```

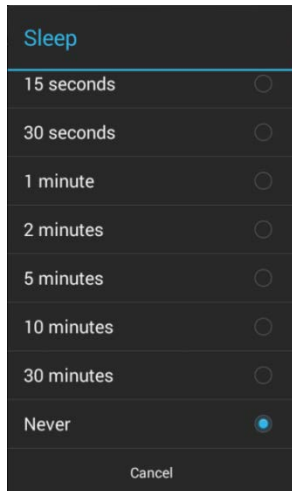
ADB command example:

```
U:\>adb root
restarting adbd as root

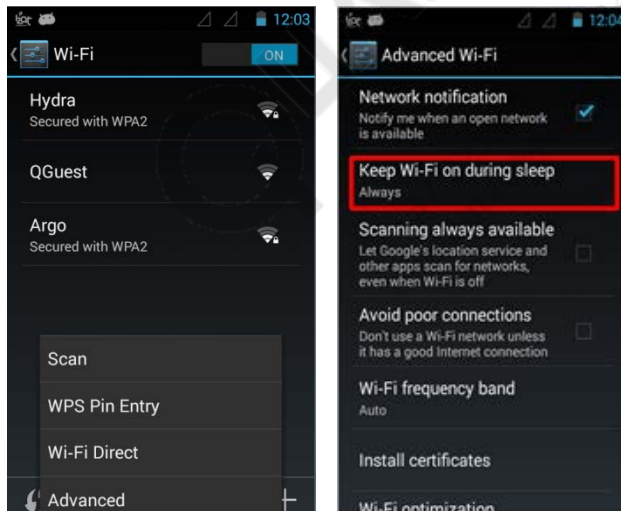
U:\>adb remount
remount succeeded

U:\>adb shell
root@msm8610:/ # cd system/bin
cd system/bin
root@msm8610:/system/bin # rm lowi_test
rm lowi_test
255!root@msm8610:/system/bin # rm lowi-server
rm lowi-server
root@msm8610:/system/bin # cd ../etc
cd ../etc
root@msm8610:/system/etc # rm lowi.conf
rm lowi.conf
!root@msm8610:/system/etc # reboot
reboot
```

4. In the Android phone UI, set the phone in wake mode by selecting **Settings**→ **Display** →**Sleep**.



5. Set WLAN after enabling WLAN function and connecting to AP in UI.



6. Return to the Android main screen.

SoftAP FTP download throughput issue

DUT works as SoftAP and PC client connect to the SoftAP. FTP download throughput could not meet CMCC requirements. This happens in Guangzhou CMCC ONLY

Solution:

This is Guangzhou CMCC network issue which tags the FTP data packets with different DSCP per packet basis. This causes bigger throughput fluctuation and there is a workaround from DUT side:

```
diff --git a/CORE/HDD/src/wlan_hdd_wmm.c b/CORE/HDD/src/wlan_hdd_wmm.c
index a43898b..d2a8b4b 100644
--- a/CORE/HDD/src/wlan_hdd_wmm.c
+++ b/CORE/HDD/src/wlan_hdd_wmm.c
@@ -1785,7 +1785,8 @@ v_VOID_t hdd_wmm_classify_pkt ( hdd_adapter_t*
pAdapter,
    }

    dscp = (tos>>2) & 0x3f;
-   userPri = hddWmmDscpToUpMap[dscp];
+   userPri = SME_QOS_WMM_UP_VI;

#ifdef HDD_WMM_DEBUG
    VOS_TRACE(VOS_MODULE_ID_HDD, WMM_TRACE_LEVEL_INFO,
```

IOT test with some CMCC Guangzhou AP(CR#590602)

DUT could not connect to some AP in Guangzhou CMCC.

Solution:

Root cause is that DUT includes some flag in IE and AP does not honor it. Fix has been identified and would be included in future release. OEM could add the following code if it does not appear in their code base.

```
diff --git a/CORE/MAC/src/pe/lim/limProcessMlmRspMessages.c
b/CORE/MAC/src/pe/lim/limProcessMlmRspMessages.c
index e5974dc..f02e964 100644
--- a/CORE/MAC/src/pe/lim/limProcessMlmRspMessages.c
+++ b/CORE/MAC/src/pe/lim/limProcessMlmRspMessages.c
@@ -706,6 +706,13 @@ limProcessMlmAuthCnf(tpAniSirGlobal pMac, tANI_U32
 *pMsgBuf)
{
    caps &= (~LIM_SPECTRUM_MANAGEMENT_BIT_MASK);
}

+    /* Clear rrm bit if AP doesn't support it */
+    if(!(pSessionEntry->pLimJoinReq->bssDescription.capabilityInfo
+        & LIM_RRM_BIT_MASK))
+    {
+        caps &= (~LIM_RRM_BIT_MASK);
+    }

    pMlmAssocReq->capabilityInfo = caps;
    PELOG3(limLog(pMac, LOG3,
        FL("Capabilities to be used in AssocReq=0x%X, privacy
bit=%x shortSlotTime %x"),
diff --git a/CORE/MAC/src/pe/lim/limUtils.h
b/CORE/MAC/src/pe/lim/limUtils.h
index 72f9c44..526d577 100644
--- a/CORE/MAC/src/pe/lim/limUtils.h
+++ b/CORE/MAC/src/pe/lim/limUtils.h
@@ -68,6 +68,7 @@ typedef enum
#define LIM_STA_ID_MASK                0x00FF
#define LIM_AID_MASK                   0xC000
#define LIM_SPECTRUM_MANAGEMENT_BIT_MASK 0x0100
+define LIM_RRM_BIT_MASK                0x1000
    #if defined (WLAN_FEATURE_VOWIFI_11R) || defined (FEATURE_WLAN_CCX) ||
defined(FEATURE_WLAN_LFR)
#define LIM_MAX_REASSOC_RETRY_LIMIT    2
#endif
```

Roaming test and GUI showed rejected (CR#582883)

DUT could not connect to CMCC AP and Android GUI shows “Rejected”.

Solution:

This is actually expected behavior. Under some weak AP signal area, DUT would keep trying to connect to AP but could not succeed. After several times retry Android framework would disable the profile. When DUT moves closer to AP, it would not reconnect automatically as the profile has been disabled.

We could have the following workaround if CMCC requires:

```
diff --git a/wifi/java/android/net/wifi/SupplicantStateTracker.java
b/wifi/java/android/net/wifi/SupplicantStateTracker.java
index e76eb17..c3b4bf8 100644
--- a/wifi/java/android/net/wifi/SupplicantStateTracker.java
+++ b/wifi/java/android/net/wifi/SupplicantStateTracker.java
@@ -99,8 +99,6 @@ class SupplicantStateTracker extends StateMachine {
    mWifiConfigStore.enableAllNetworks();
    mNetworksDisabledDuringConnect = false;
}
- /* Disable failed network */
- mWifiConfigStore.disableNetwork(netId, disableReason);
```

EAP-SIM connection failure (CR#NA)

Sometimes DUT could not connect to AP with EAP-SIM authentication method.

Solution:

This is a general guide and there is no specific fix.

1. Make sure DUT has the latest patch EAP-SIM patch.
2. Change different card to try.
3. Test reference device and see if it is working normally.
4. If there is QMI response timeout in logcat log as “E/wpa_supplicant(1097): eap_proxy_qmi_response_wait !QMI STATE 1 TIME_OUT”, the following timeout value could be increased to try:

```
diff --git a/src/eap_peer/eap_proxy.h b/src/eap_peer/eap_proxy.h
+#define QMI_RESP_TIME_OUT 650
```

5. If the above test results proves to be DUT issue, file a case for Qualcomm CE.

3.4.2 AGPS

China Mobile has deployed A-GPS SUPL1.0 for China Mobile customized handsets. A-GPS support is mandatory.

Configure AGPS NV items on the modem. For more details, see GPSONE GEN 7 Engine (1X AND UMTS) Nonvolatile Items Description (80-VG439-1).

```
NV 4707= 1 //configured as User Plane(SUPL).
NV 1920 = 7 //standalone and user plane AGPS is supported.
           If it is USA market, set it to 0xFF7F.
NV 3758 =1 //(Enable User Plane Secure Transport), CMCC SUPL server
           is security transport mechanism.
```

1. Inject SUPL Root certificate file. Put SUPL Root certificate file which is got from CMCC at modem EFS/SUPL folder with any name. EFS/SUPL folder may store at most 10 A-GPS SUPL certificates files.

During the testing phase, Customers may use Qualcomm QPST-> EFS explorer to drag certificates into modem EFS/SUPL folder.

During commercial launch, customers also call Qualcomm location API on android to inject certificates files into modem EFS/SUPL folder from android side.

All SUPL Root certificate files must be DER encoding. You need to convert it to DER format if it is CER encoding certificate.

2. Set the SUPL server address setting on the Android side.

Configure SUPL server address (SUPL_HOST and SUPL_PORT) in Android /etc/gps.conf file. This file is located at android build folder: android\hardware\qcom\gps\etc. SUPL server address is configured properly in /etc/gps.conf, which address will be written into NV4703 SUPL Server URL Address automatically while handset powers up.

For example:

In: android\hardware\qcom\gps\etc\gps.conf

⌘ For CMCC live SUPL server in commercial network

```
# FOR SUPL SUPPORT, set the following
SUPL_HOST=221.176.0.55
SUPL_PORT=7275
```

```
# supl version 1.0
SUPL_VER=0x10000
```

⌘ For CMCC lab SUPL server in CMCC BJ lab network(CMCC TA testing is run on the lab server in CMCC lab network)

```
# FOR SUPL SUPPORT, set the following
```

```
SUPL_HOST= 218.206.176.50
SUPL_PORT=7275
```

```
# supl version 1.0
SUPL_VER=0x10000
```

3. Add one SUPL type APN for AGPS in data\etc\Apns-conf.xml, we recommend strongly that adding one SUPL type into existing APN list, do not create one separate APN for SUPL type.

- ✎ For China Mobile live SUPL server, the SUPL type APN should be added into APN cmwap.

For example:

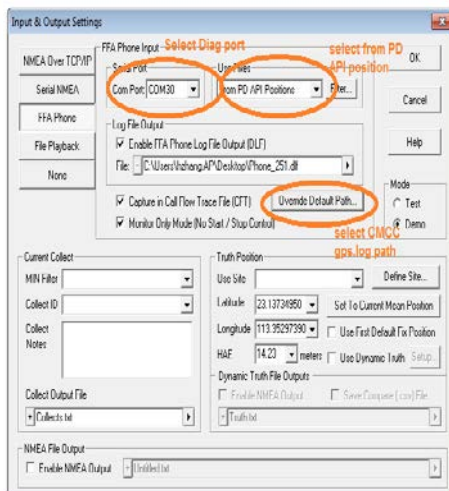
```
<apn carrier="China Mobile" apn="cmwap" mcc="460" mnc="02" user=""
server="" password="" proxy="" port="" mmsproxy="" mmsport="" mmsc=""
type="default,supl"/>
```

```
<apn carrier="China Mobile GPRS" apn="cmwap" mcc="460" mnc="07"
user="" server="" password="" proxy="" port="" mmsproxy="" mmsport=""
mmsc="" type="default,supl"/>
```

- ✎ For CMCC lab SUPL server, OEM should get the right lab APN for AGPS. Add one SUPL type APN for AGPS on the APN in data\etc\Apns-conf.xml.

3.4.2.1 CMCC A-GPS log collection tool

1. Download the latest SnapperHS tool from Documents and Downloads. SnapperHS is one Qualcomm AGPS call flow analysis and performance statistic tool.
2. Open SnapperHS->Setting menu, then follow settings below to collect CMCC A-GPS log “gps.log” while running A-GPS testing (MO/NI MSA and MSB). While A-GPS is started, the A-GPS log will be output into gps.log file. CMCC already has Qualcomm SnapperHS tool.



3.4.2.2 Use Qualcomm GPS /A-GPS test tool ODLT to run set initiated MSA and MSB

Qualcomm GPS tool ODLT is released with CRM build. It is one GPS test tool with rich features, no extra installation needed. ODLT logo is shown in the list of applications if you load one Qualcomm baseline build.

Running cold start MSA

1. Options→Edit config→API selection: Loc API
2. Options→Edit config→fix parameters settings→Recurrence Type: Singleshot
3. Options→Edit config→fix parameters settings→preferred op mode: MS-assisted
4. Options→Edit config→assistance data deletion: Delete All.
5. Go back the menu one by one, select Save Config file as “CMCC_cold_MSA”.
6. Go back the menu one by one, start the testing. use SnapperHS tool to CMCC AGPS gps.log

Running cold start MSB

1. Options→Edit config→API selection: Loc API
2. Options→Edit config→fix parameters settings→Recurrence Type: App Track
3. Options→Edit config→fix parameters settings→preferred op mode: MS-based
4. Options→Edit config→assistance data deletion: Delete All.
5. Go back the menu one by one, select Save Config file as “CMCC_cold_MSB_Track”.
6. Go back the menu one by one, start the testing. use SnapperHS tool to CMCC AGPS gps.log

Running cold start standalone

1. Options→Edit config→API selection: Loc API
2. Options→Edit config→fix parameters settings→Recurrence Type: App Track
3. Options→Edit config→fix parameters settings→preferred op mode: Standalone
4. Options→Edit config→assistance data deletion: Delete All.
5. Go back the menu one by one, select Save Config file as “CMCC_cold_ST_Track”.
6. Go back the menu one by one, start the testing. use SnapperHS tool to CMCC AGPS gps.log

3.4.2.3 CMCC A-GPS known SUPL server issue and workaround on handset side

Qualcomm strictly follow 3GPP and OMA to implement the SUPL A-GPS. But during SUPL IOT between Qualcomm and CMCC NSN SUPL server, Qualcomm finds one CMCC SUPL server issue. The Almanac provided by CMCC SUPL server has wrong PRN numbering. While Qualcomm verifies the consistence of Almanac and ephemeris provided by CMCC, consistence checking fails. So all Almanac and ephemeris provided by CMCC SUPL server will not be used in position calculation. This causes longer TTFF. So on handset side, customers have to apply one workaround. For more information see *Application Note: Interoperability Issue on CMCC (NSN/MOT) Networks (80-NL684-1)*.

For CMCC live SUPL server, due to CMCC SUPL server loading issue, CMCC live SUPL server have very high refusing rate on SSL layer, SUPL request etc. This is known CMCC SUPL server issue , there is no way to solve it on handset side.

3.4.3 MIFI

The CMCC lab at Hangzhou is neither a shielded room nor a clean environment for WLAN. There is a lot of AP interference that affects throughput and response time.

Workarounds for the typical WLAN-related issues encountered at the CMCC Hangzhou lab are described below.

Issue	Solution
Throughput issue – End2end FTP download or upload cannot meet CMCC requirements.	To get the best possible throughput: <ol style="list-style-type: none">1. Disable softAP mode power save: <code>#wmiconfig -i eth0 --power maxperf</code>2. Choose the cleanest available channel. Use a sniffer tool to collect the AP list for each channel then select the cleanest channel to use.3. Select a time when there is not much testing being performed. For example, schedule the testing at noon or at night.
Ping response time cannot meet CMCC requirement – CMCC requires End2end ping response time < 30ms. Since the LTE side needs ~20ms, that leaves only ~10ms for WLAN response time	To avoid long response time: <ol style="list-style-type: none">1. Ensure that the device is using the cleanest possible channel.2. Disable softAP mode power save: <code>#wmiconfig -i teth0 --power maxperf</code>3. Choose the cleanest available channel. Use a sniffer tool to collect the AP list for each channel then select the cleanest channel to use.4. Select a time when there is not much testing being performed. For example, schedule the testing at noon or at night.

3.4.4 BT and FM

Issue	Solution
Power consumption is too high during BT file exchange. Expected result is 150mA while actual power consumption is approximately 203mA.	<ol style="list-style-type: none">1. Change the power class to reduce Bluetooth RF power consumption:<ol style="list-style-type: none">a. From the UI, turn off Bluetooth.b. In adb shell, set the <code>qcom.bt.dev_power_class</code> property to 2 using this command: <code>adb setprop qcom.bt.dev_power_class 2</code>c. Turn on Bluetooth2. Reduce OBEX MTU to 16k. (This reduces power consumption but increases the time required to transfer a file.) <pre>diff --git a/obex/javax/obex/ObexHelper.java b/obex/javax/obex/ObexHelper.java index 8ebd802..97d6d33 100644 --- a/obex/javax/obex/ObexHelper.java +++ b/obex/javax/obex/ObexHelper.java @@ -76,7 +76,7 @@ public final class ObexHelper { * Temporary workaround to be able to push files * to Windows 7. * TODO: Should be removed as soon as Microsoft * updates their driver. */ - public static final int MAX_CLIENT_PACKET_SIZE = 0xFC00; + public static final int MAX_CLIENT_PACKET_SIZE = 0x4000;</pre>

3.5 Software reliability test items

CMCC scope
Item
MTBF Script modification base on OEM's build.
Power Measurement Uninstall all applications which consume extra power.
Stress
Concurrency
Abnormal
Local Performance
Endurance
OTA
Structure
Audio

3.6 Hardware reliability test items

Module	Item
Hardware Reliability	OTA
	Structure
	Audio

4 CTA Guidelines

4.1 WCDMA

4.1.1 Test case 3GPP 34.123-1

The reason for handover failure is no measurement report. The test case is as below.

3GPP 34.123-1, 8.3.7.1、 8.3.11.1、 8.3.11.14 , 8.4.1.35.

Analysis and solution:

SGLTE Test RF Connection:

Normally, SGLTE phone has two RF transceivers and three antennas (not including GPS and WIFI). The first transceiver supports L/W/TDS and it has two antennas for primary TX/RX path and diversity RX path. The second transceiver only supports GSM quad bands and it only has one dedicate antenna for GSM TX/RX. So for testing such X2G IRAT case on SGLTE phone, we need to use power splitter to connect L/W/TDS primary antenna with GSM antenna together. Otherwise, even phone can measure some GSM RSSI, but would fail in FCCH and SCH decoding. Thus, X2G IRAT case would fail.

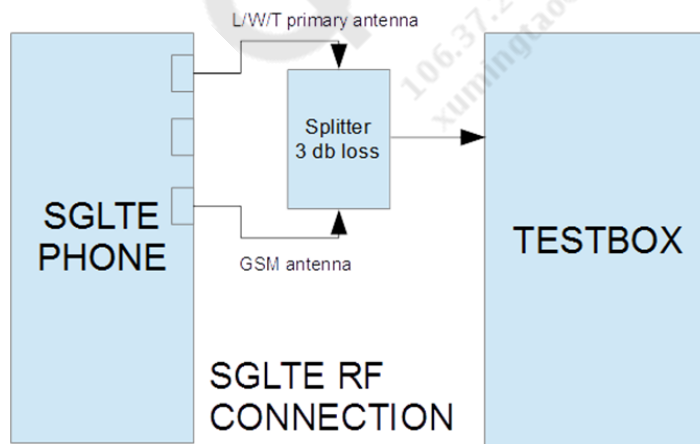


Figure 6-1 SGLTE Test RF Connection

4.2 TDSCDMA

4.2.1 TDSCDMA RRM conformance

CTA lab requires UE set to TDS, TDL, GSM mode + SGLTE mode to test TDS RRM cases.

Solution:

1. Set to TDS, TDL, GSM mode + SGLTE mode
2. In policy file `carrier_policy.xml`, Add **466** to `mcc_list`, add **466-02** to `plmn_list`.

```
<mcc_list name="sglte_mccs"> 001 002 003 004 005 006 007 008 009 010 011  
012 316 440 460 466 </mcc_list>.  
<plmn_list name="sglte_operators"> 466-02 440-79 001-01 460-00 460-01  
460-02 460-03 460-04 460-07 460-09 460-08 460-080 460-71 002-02 002-81  
002-91 002-11 466-09 246-81 246-081 </plmn_list> .
```

4.2.2 TDSCDMA RF – HSDPA test part

This is only applicable to CTA testing.

For TDSCDMA RF lab conformance (HSDPA test part), if the device can't do registration, it may be caused by the test SIM HPLMN (001, 01) and callbox doesn't have integration protection. It needs to set **NV 69731** to **0**, **NV 66011** to **0**, **NV 66013** to **1**. After passing such kind of case, it needs to change these three NVs back to original setting **NV69731** to **2**, **NV 66011** to **1**, **NV 66013** to **0**.

It depends on what kind of SIM card is used and if the callbox has integrity protection check.

For CMCC testing, don't need this setting.

4.3 TD-LTE

4.3.1 TDL protocol

4.3.1.1 TDL protocol 6.1.1.1 fails

Solution:

1. Remove the EFS file `/nv/item_files/modem/nas/ehplmn`. Or add **001-01 PLMN** to EFS file `/nv/item_files/modem/nas/ehplmn`.
2. This case needs to be test with a special-designed SIM card.
3. Set **NV65605=0** and **NV69731=2**.

4.3.1.2 ZUC algorithm

CTA needs to enable ZUC, for details, refer to *Enabling ZUC Algorithms in LTE Security capabilities* (80-NF455-1).

And other lab and live test use default setting and don't need to enable ZUC.

Set **EIA3** and **EEA3** to **1** to enable ZUC algorithm. Please refer to [3.1.1](#), change `efs_lte_nas_ue_sec_capability` to enable ZUC.

4.4 USIM NI-UICC

For CTA USIM NI-UICC part, UE in SGLTE home mode, needs to add **234 244 246** to “`sglte_mccs`” list, and meantime add **234-005 246-81 246-081** to “`sglte_operators`” list.

New **carrier_poily** list of lab testing is as below:

```
<mcc_list name="sglte_mccs"> 001 002 003 004 005 006 007 008 009 010 011 012 316 440 460
466 234 244 246 </mcc_list>
<plmn_list name="sglte_operators"> 466-02 440-79 001-01 460-00 460-01 460-02 460-03 460-
04 460-07 460-09 460-08 460-080 460-71 002-02 002-81 002-91 002-11 466-09 246-81 246-081
234-005 </plmn_list>
```

5 RF Software Changes

5.1 RF card

QRD8916 has two superset reference schematics, and in MSM8916 QRD software PL1.1, the corresponding RF card detail is as follows:

Ref.schematic DCN	RF card	NV1878
DP10-VL716-100	RF_HW_WTR1605_CHILE_RP1	62
DP10-VL717-100	RF_HW_WTR1605_CHILE_RP2	89

Driver code of this RF card is located in these folders,

modem_proc\rfc_dimepm\rf_card\rfc_wtr1605_chile_rp1

modem_proc\rfc_dimepm\rf_card\rfc_wtr1605_chile_rp2

RP1 is a single transceiver CSFB design supporting most of popular bands.

RF_HW_WTR1605_CHILE_RP1
1 st transceiver
WTR1605L
FDD-LTE B1, B3, B7 TDD-LTE B38, B39, B40, B41 TD-SCDMA B34, B39 WCDMA B1, B2, B5, B8 CDMA BC0 GSM 850,900,1800,1900

RP2 is a dual transceiver SVLTE design, with 2-4-2 PCB reference.

RF_HW_WTR1605_CHILE_RP2	
1 st transceiver	2 nd transceiver
WTR1605L	WTR2605
FDD-LTE B1, B3, B7 TDD-LTE B38, B39, B40, B41 TD-SCDMA B34, B39 WCDMA B1, B2, B5, B8 CDMA BC0 GSM 850,900,1800,1900	CDMA BC0

About how to customize RP2 RF card to CMCC CSFB / CT SRLTE / CU CSFB variants in DP10-VL717-100, will add contents in further revision.

Follow the reference schematic / PCB layout to minimize the effort involved in bring-up.

5.1.1 Static QCN

Static QCN XMLs for QRD 8916 RF cards are provided in modem_proc\rftarget_dimepm\common\qcn folder,

RF card	Mode	Static XML subfolder
RP1	APT	wtr1605_chile_rp1
	APT+DPD	wtr1605_chile_rp1_ept
RP2	APT	wtr1605_chile_rp2_apr
	APT+DPD	TBD

MSM8916_<RF card name> _MASTERFILE.xml is the portal XML for a specified RF card.

Use this file when you generate static QCN in QRCT.

In the same folder, static NV items are listed in mode / band / RXTX categorized XMLs for convenient editing. See *Comprehensive MSM8974/MDM9x25/MDM8x26 Family RF NV Items* [80-N5420-171] for information on these NV items.

NOTE: These XMLs only contains static RFNV items. Downloading this static QCN will not make your UE get online.

5.2 Characterization and Calibration

Because QRD8916 uses QPA, 2 new concepts, **Characterization** and **ESC V4 calibration** are mandatory on this platform. As an result, **TX Multi-Lin V3** linearizer NVs are also needed on this platform.

5.2.1 Characterization

QPA works in compressed zone in high gain mode, characterization is a necessary step in RF bringing up, customer needs to go through RGI bias characterization and Pin / Pout characterization over different temperatures to ensure handset's ACLR/ACPR performance.

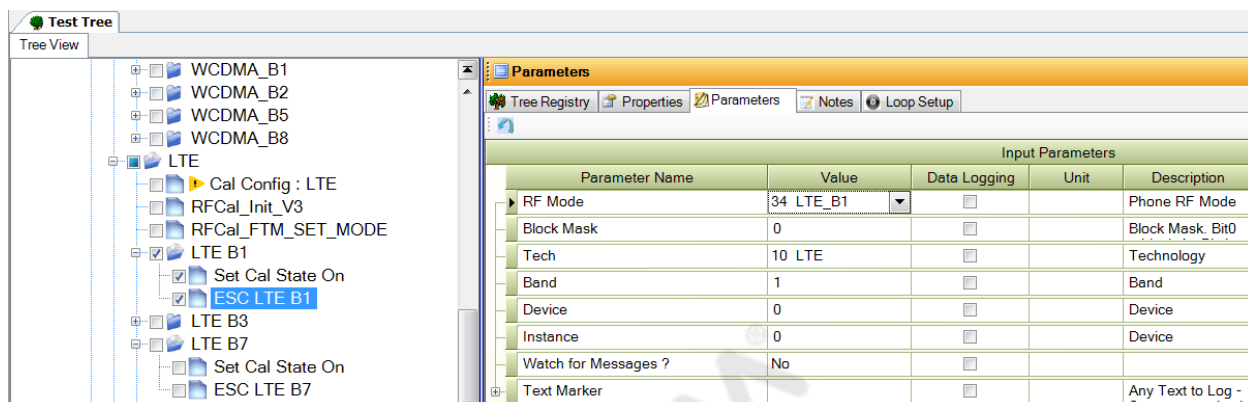
See 80-NK201-6 for detail steps about how to do characterization on 8916.

General introduction about modem overview APT+DPD can be found in 80-NG201-3.

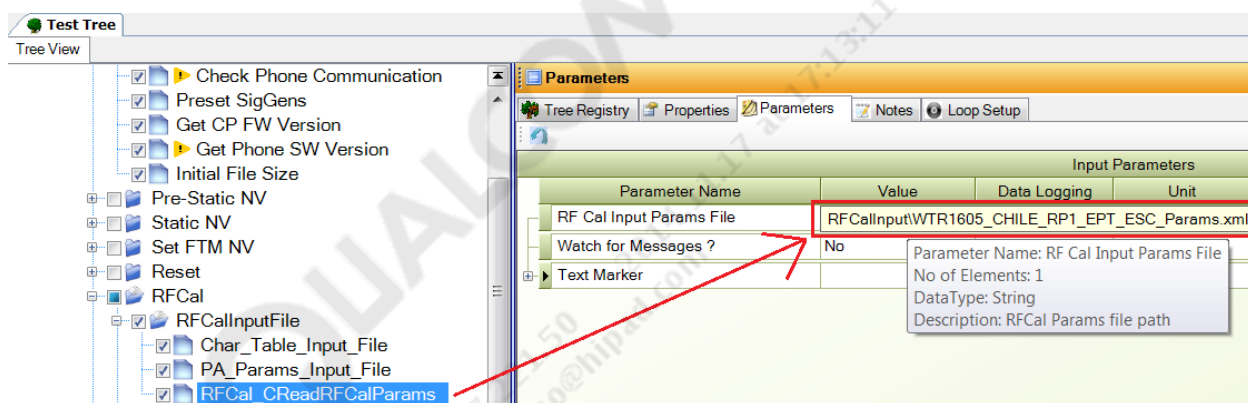
5.2.2 Calibration

ESC V4 calibration is introduced in 8916 platform.

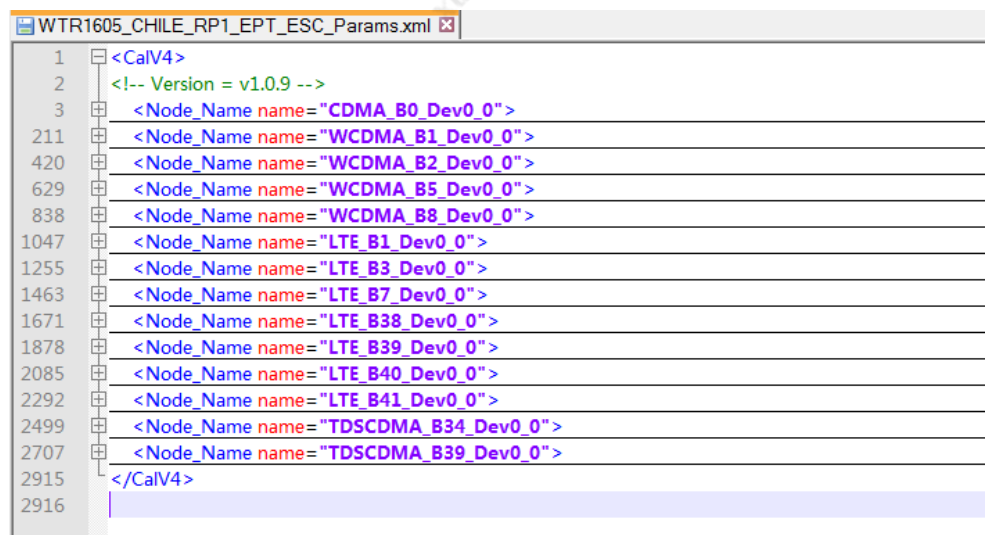
You may notice that for a specified band, calibration parameters are no longer given in calibration XTT test item,



Unlike previous ESC V3 calibration XTT, RF calibration parameters in ESC V4 calibration XTT is not given in the XTT directly, but in a referred ESC V4 calibration parameters XTT,



This XML file referred in ESC V4 calibration XTT in “RFCal_CReadRFCalParams” item contains ESC V4 cal parameters, you may use any XML text editor to read / modify it.



Details of ESC V4 parameters settings can be found in 80-NG201-2.

5.2.3 TX MULTI-LIN V3 NV items

As a result of ESC V4 calibration and APT+DPD characterization, TX MULTI-LIN **V3** NV items replace original TX MULTI-LIN V1 NV items to store TX linearizer calibration data on QRD8916.

	Previous Platforms	MSM8916
Cal XTT	ESC V3 calibration	ESC V4 calibration
NV item	TX Multi-Lin V1	TX Multi-Lin V3
feature	APT	APT, APT+DPD, ET

TX Multi-Lin V3 NV items' name are as below,

RFNV_LTE_B1_TX_MULTI_LIN_V3_DATA_I

RFNV_LTE_B2_TX_MULTI_LIN_V3_DATA_I

RFNV_LTE_B2_TX_MULTI_LIN_V3_DATA_I

...

This new category NV items have variable length and structure, to understand detail about it, see 80-N5420-171, chapter 5.2.2.2.

5.3 General bringing up steps

General bringing up steps on 8916 platform can be described as

1. Determine RF card and generate / download static QCN.
2. Characterization and RF calibration
3. Service programming / Protocol NV setting.

Check 80-NB715-1 for details.

6 Factory Tools

For more information about factory tools, refer to the following documents.

Document Name	DCN
Factory tool adoption	
QDART-MFG User Guide	80-NF136-1
QDART-MFG SWDL tool User Guide (<i>draft</i>)	80-NF136-2
QDART-MFG RFcal & verify tool User Guide	80-NF136-3
QDART-MFG Error Codes	80-NF136-4
QDART-MFG Configuration GUI Editor User Guide	80-NH711-1
QDART-MFG Station Calibration User Guide (<i>draft</i>)	80-NF136-5
Factory tools development and optimization	
8226 RF Test time reduction	80-NG894-24
QTI_SCPI_RF_Calibration_Commands	80-NB700-1
Qualcomm Test Sequence System – QSEQ	80-N2330-2
QDART Multi-DUT Developer's Guide	80-NJ921-1
Application Note: EMMC Downloader	80-NG850-1

7 NV and EFS Tips

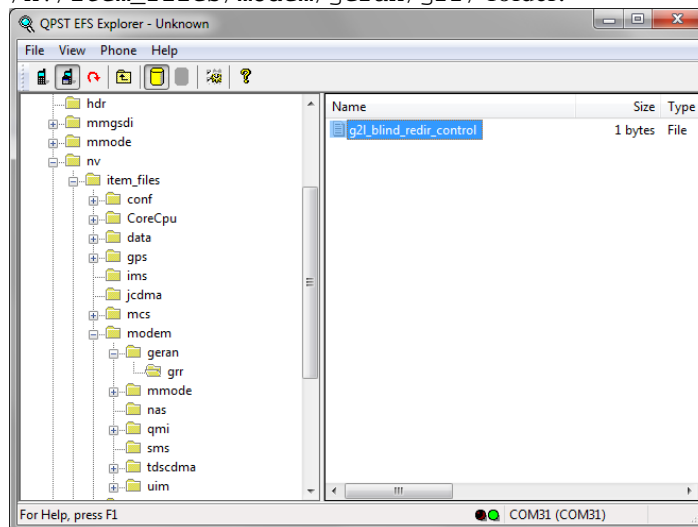
7.1 Adding an EFS NV item


For the NV item that are located in the `nv/item_files` folder, the configuration has two key pointers:

1. Edit the applicable `.conf` file to add the path to the NV item file. For example, to add the GERAN G2L Blind Redirection Control item (NV69747), edit the `/nv/item_files/conf/geran.conf` file and add the line shown in red below:

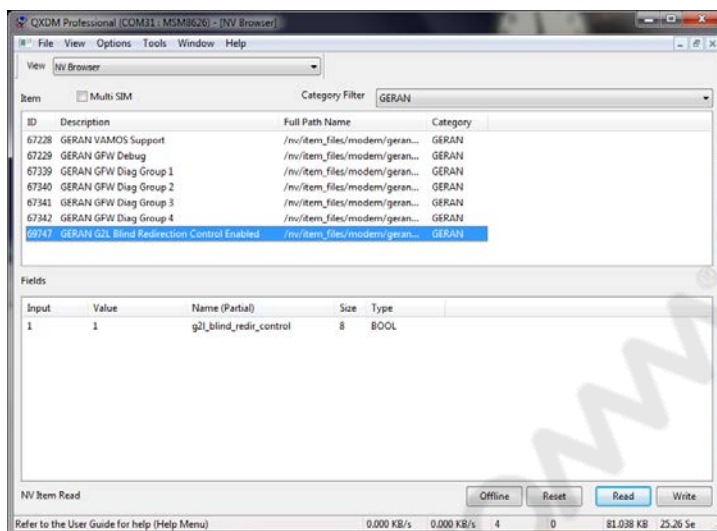
```
/nv/item_files/gsm/gll/gsm_commercial_recovery_restart
/nv/item_files/gsm/l1/l1_debug
/nv/item_files/gsm/l1/l1_q6_clock
/nv/item_files/modem/geran/gfw_debug
/nv/item_files/modem/geran/vamos_support
/nv/item_files/modem/geran/gfw_diag_group1
/nv/item_files/modem/geran/gfw_diag_group2
/nv/item_files/modem/geran/gfw_diag_group3
/nv/item_files/modem/geran/gfw_diag_group4
/nv/item_files/modem/geran/grr/g2l_blind_redir_control
```

2. In QPST Explorer, move the `g2l_blind_redir_control` EFS file into the `/nv/item_files/modem/geran/grr/` folder.



3. Click  to reboot the device.

4. The NV item (in this case, NV69747) is displayed in QXDM. Click **Read**.

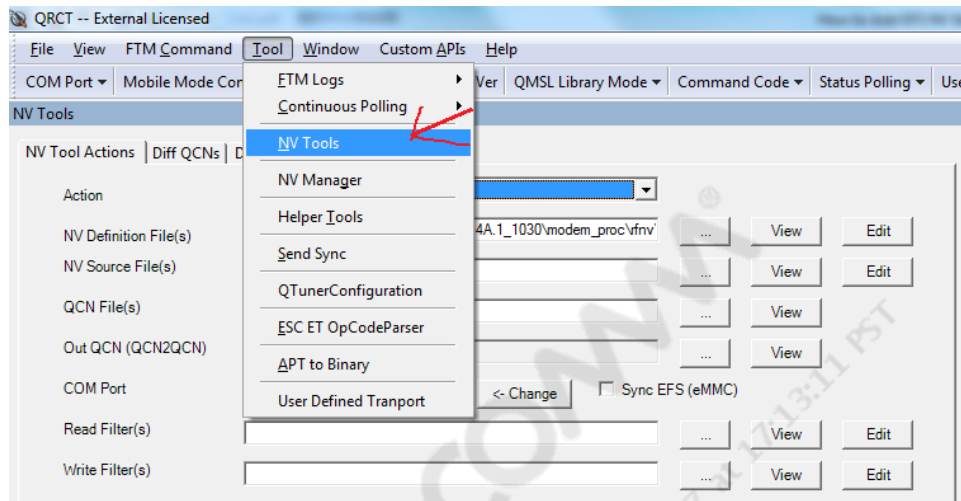


Troubleshooting

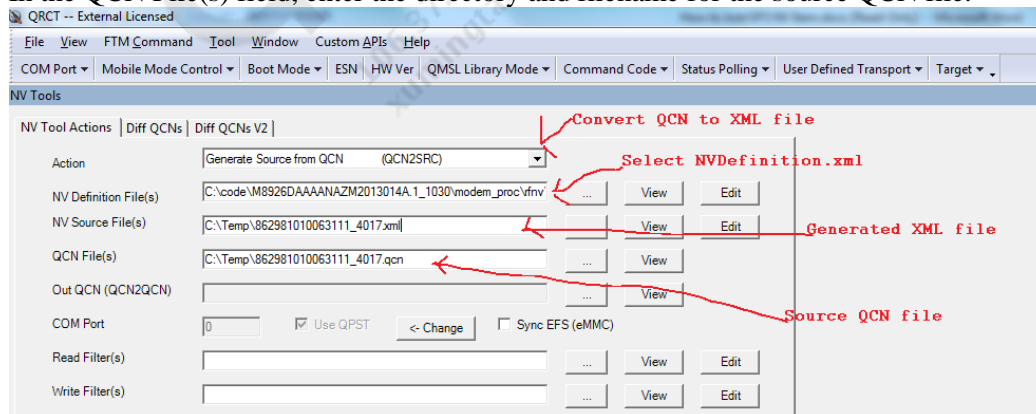
Issue	Solution
<p>QXDM reports an error:</p> <p>NV Status Error Received: No such file or directory</p> <p>Attempting To Read NV Item Failed – EFS Item is not configured</p>	<ol style="list-style-type: none"> 1. Verify that the applicable .conf file has been updated to include the NV item. For example, to add support for NV67229 (GERAN GFW Debug), the following entry must be present in the <code>nv/item_files/conf/geran.conf</code> file: <code>/nv/item_files/modem/geran/gfw_debug</code> 2. Verify that the NV item file is present in the applicable <code>/nv/item_files/modem</code> subfolder. For example, the <code>gfw_debug</code> EFS file must be present in the <code>/nv/item_files/modem/geran</code> folder.

7.2 Converting QCN to XML file for easy edit

1. Open QDART > QRCT > Tool > NV Tools.



2. From the Action dropdown list, select Convert QCN to XML file.
3. In the NV Definition File(s), click the Browse button (...) and select the NVDefinition.xml file.
4. In the NV Source File(s) field, enter the directory and filename for the generated XML file.
5. In the QCN File(s) field, enter the directory and filename for the source QCN file.



If protocol EFS file may not be included in NVDefinition.xml, the EFS/NV item will be missing after conversion. See *QDART NV Tool* 80-VN248-1

NOTE: to add EFS/NV items in NVDefinition.xml to avoid this issue.

7.3 WCNSS_qcom_config.ini

This file allows you to override the factory defaults for the WLAN driver.

```
# Enable IMPS or not
gEnableImps=0
# Enable/Disable Idle Scan
```

```
gEnableIdleScan=0
# Increase sleep duration (seconds) during IMPS
# 0 implies no periodic wake up from IMPS. Periodic wakeup is
# unnecessary if Idle Scan is disabled.
gImpsModSleepTime=0
# Enable BMPS or not
gEnableBmps=0

# Enable suspend or not
# 1: Enable standby, 2: Enable Deep sleep, 3: Enable Mcast/Bcast Filter
gEnableSuspend=3

# Phy Mode (auto, b, g, n, etc)
# Valid values are 0-9, with 0 = Auto, 4 = 11n, 9 = 11ac
gDot11Mode=0

# Handoff Enable(1) Disable(0)

gEnableHandoff=0

# CSR Roaming Enable(1) Disable(0)

gRoamingTime=0

# Assigned MAC Addresses - This will be used until NV items are in place

# Each byte of MAC address is represented in Hex format as XX

Intf0MacAddress=000AF58989FF
Intf1MacAddress=000AF58989FE
Intf2MacAddress=000AF58989FD

Intf3MacAddress=000AF58989FC

# UAPSD service interval for VO,VI, BE, BK traffic

InfraUapsdVoSrvIntv=0

InfraUapsdViSrvIntv=0

InfraUapsdBeSrvIntv=0

InfraUapsdBkSrvIntv=0
```

```
# Make 1x1 the default antenna configuration

gNumRxAnt=1

# Beacon filtering frequency (unit in beacon intervals)

gNthBeaconFilter=50

# Enable WAPI or not

WAPIIsEnabled=1

# Flags to filter Mcast abd Bcast RX packets.

# Value 0: No filtering, 1: Filter all Multicast.

# 2: Filter all Broadcast. 3: Filter all Mcast abd Bcast

McastBcastFilter=0

#Flag to enable HostARPOffload feature or not

hostArpOffload=1

#SoftAP Related Parameters

# AP MAC addr

gAPMacAddr=000AF589dcab

# 802.11n Protection flag

gEnableApProt=1

#Enable OBSS protection

gEnableApOBSSProt=1

#Enable/Disable UAPSD for SoftAP

gEnableApUapsd=0

# Fixed Rate
```

```
gFixedRate=0

# Maximum Tx power
# gTxPowerCap=30

# Fragmentation Threshold

# gFragmentationThreshold=2346

# RTS threshold

RTSThreshold=2347

# Intra-BSS forward

gDisableIntraBssFwd=0

# WMM Enable/Disable

WmmIsEnabled=0

# 802.11d support

g11dSupportEnabled=1

# 802.11h support

g11hSupportEnabled=1

# CCX Support and fast transition
CcxEabled=0
FastTransitionEnabled=0
ImplicitQosIsEnabled=1
gNeighborScanTimerPeriod=200

gNeighborLookupThreshold=76
gNeighborReassocThreshold=81

gNeighborScanChannelMinTime=20
gNeighborScanChannelMaxTime=30
gMaxNeighborReqTries=3

# Legacy (non-CCX, non-802.11r) Fast Roaming Support
```

```
# To enable, set FastRoamEnabled=1
# To disable, set FastRoamEnabled=0
FastRoamEnabled=0

#Check if the AP to which we are roaming is better than current AP in terms
of RSSI.
#Checking is disabled if set to Zero.Otherwise it will use this value as to
how better
#the RSSI of the new/roamable AP should be for roaming
RoamRssiDiff=3

# If the RSSI of any available candidate is better than currently
associated
# AP by at least gImmediateRoamRssiDiff, then being to roam immediately
(without
# registering for reassoc threshold).
# NOTE: Value of 0 means that we would register for reassoc threshold.
gImmediateRoamRssiDiff=10
# SAP Country code

# Default Country Code is 2 bytes, 3rd byte is optional indoor or out door.

# Example

#   US Indoor, USI

#   Korea Outdoor, KRO

#   Japan without optional byte, JP

#   France without optional byte, FR

#gAPCntryCode=USI

#Short Guard Interval Enable/disable

gShortGI20Mhz=1

gShortGI40Mhz=1

#Auto Shutdown Value in seconds. A value of 0 means Auto shutoff is
disabled

gAPAutoShutOff=0
# SAP auto channel selection configuration
```

```
# 0 = disable auto channel selection

# 1 = enable auto channel selection, channel provided by supplicant will be
ignored

gApAutoChannelSelection=0

# Listen Energy Detect Mode Configuration

# Valid values 0-128

# 128 means disable Energy Detect feature

# 0-9 are threshold code and 7 is recommended value from system if feature
is to be enabled.

# 10-128 are reserved.

# The EDET threshold mapping is as follows in 3dB step:

# 0 = -60 dBm

# 1 = -63 dBm

# 2 = -66 dBm

# ...

# 7 = -81 dBm

# 8 = -84 dBm

# 9 = -87 dBm

# Note: Any of these settings are valid. Setting 0 would yield the highest
power saving (in a noisy environment) at the cost of more range. The range
impact is approximately #calculated as:

#

# Range Loss (dB) = EDET threshold level (dBm) + 97 dBm.

#
```

```
gEnablePhyAgcListenMode=128

#Preferred channel to start BT AMP AP mode (0 means, any channel)

BtAmpPreferredChannel=0

#Preferred band (both or 2.4 only or 5 only)

BandCapability=0

#Beacon Early Termination (1 = enable the BET feature, 0 = disable)

enableBeaconEarlyTermination=0

beaconEarlyTerminationWakeInterval=3

#Bluetooth Alternate Mac Phy (1 = enable the BT AMP feature, 0 = disable)

gEnableBtAmp=0

#SOFTAP Channel Range selection

gAPChannelSelectStartChannel=1

gAPChannelSelectEndChannel=11

#SOFTAP Channel Range selection Operating band

# 0:2.4GHZ 1: LOW-5GHZ 2:MID-5GHZ 3:HIG-5GHZ 4: 4.9HZ BAND

gAPChannelSelectOperatingBand=0

#Channel Bonding

gChannelBondingMode5GHz=1

#Enable Keep alive with non-zero period value

#gStaKeepAlivePeriod = 30

#AP LINK MONITOR TIMEOUT is used for both SAP and GO mode.
#It is used to change the frequency of keep alive packets in the AP Link
Monitor period which is by
#default 20s. Currently the keep alive packets are sent as an interval of
3s but after this change
```

```
#the keep alive packet frequency can be changed.

#gApLinkMonitorPeriod = 3

#If set will start with active scan after driver load, otherwise will start
with

#passive scan to find out the domain

gEnableBypass1ld=1

#If set to 0, will not scan DFS channels

gEnableDFSChnlScan=1

gVhtChannelWidth=2
gEnableLogp=1

# Enable Automatic Tx Power control

gEnableAutomaticTxPowerControl=1

# 0 for OLPC 1 for CLPC and SCPC
gEnableCloseLoop=1

#Data Inactivity Timeout when in powersave (in ms)
gDataInactivityTimeout=200

# VHT Tx/Rx MCS values
# Valid values are 0,1,2. If commented out, the default value is 0.
# 0=MCS0-7, 1=MCS0-8, 2=MCS0-9
gVhtRxMCS=2
gVhtTxMCS=2

# Enable CRDA regulatory support by settings default country code
#gCrdaDefaultCountryCode=TW

# Scan Timing Parameters
# gPassiveMaxChannelTime=110
# gPassiveMinChannelTime=60
# gActiveMaxChannelTime=40
# gActiveMinChannelTime=20
gPassiveMaxChannelTimeConc=110
gPassiveMinChannelTimeConc=60
```



```
gActiveMaxChannelTimeConc=27
gActiveMinChannelTimeConc=20
gRestTimeConc=100
gNumChanCombinedConc=1

#If set to 0, MCC is not allowed.
gEnableMCCMode=1

# 1=enable STBC; 0=disable STBC
gEnableRXSTBC=1

# Enable Active mode offload
gEnableActiveModeOffload=1

#Enable Scan Results Aging based on timer
#Timer value is in seconds
#If Set to 0 it will not enable the feature
gScanAgingTime=0

#Enable Power saving mechanism Based on Android Framework
#If set to 0 Driver internally control the Power saving mechanism
#If set to 1 Android Framwrok control the Power saving mechanism
isAndroidPsEn=0

#disable LDPC in STA mode if the AP is TXBF capable
gDisableLDPCWithTxbfAP=1

#Enable thermal mitigation
#gThermalMitigationEnable=1
END

# Note: Configuration parser would not read anything past the END marker
```

Part 2 – Beginner's Guide to QTI

8 Software Installation and Setup

8.1 Installation and Setup

8.1.1 Required equipment and software

Table 10-1 identifies the equipment and software needed for a user to install and run the software.

Table 10-1 Required equipment and software

	Item description	Version	Source/vendor	Purpose
1	Linux development workstation that exceeds minimum requirements for running Ubuntu 64-bit OS	—	—	Android build machine
2	Windows 7 or Windows XP workstation	Windows 7 or Windows XP	Microsoft	Alternate Non-HLOS build machine and Windows-based programming tools
3	Ubuntu 12.0.4 LTS Linux distribution for 64-bit architecture	12.0.4 LTS	Ubuntu Community/ Canonical, Ltd.	Android build host OS
4	Java SE JDK for Linux x64	6	Oracle	Building Android
5	repo	—	Android Open Source Project	Android source management tool
6	ARM [®] toolchain	ARM Compiler Tools 5.01 update 3 (build 94)	ARM Ltd.	Building boot images, RPM, TrustZone (TZ)
7	Hexagon [™] toolchain	6.2.09	Qualcomm/ Gnu	Building Modem Processor Subsystem (MPSS) and Applications Digital Signal Processor Subsystem (ADSP)

	Item description	Version	Source/vendor	Purpose
8	Python	2.6.6	Python.org	Building boot, subsystem, etc. <ul style="list-style-type: none"> ▪ Boot – Ver 2.6.6 ▪ RPM – Ver 2.6.6 ▪ TZ – Ver 2.6.6 ▪ Meta – Ver 2.7.5 ▪ MPSS – Ver 2.7.6
9	SCons Ver 2.0.0 or higher		scons.org	Building all non-HLOS source code releases

8.1.2 Installing Ubuntu

You must be able to log in as root or use sudo to have root permissions during the installation.

1. Create an installation CD and install it onto the computer by following the instructions at <http://releases.ubuntu.com>.
2. After installation, perform a software update using one of the following options:
 - a. Using the GUI, select System→Administration→Update Manager
or
 - b. Using the shell command line
 - i. Edit the source config file directly, as follows:

```
sudo vi /etc/apt/sources.list
```
 - ii. Edit the file to enable the universe and multiverse sources, and disable the Ubuntu installation CD source.
 - iii. From the command line, perform the package list update and package upgrades:

```
sudo apt-get update
sudo apt-get upgrade
```
3. Use apt-get to install the additional required packages.

```
$ sudo apt-get install git-core gnupg flex bison gperf build-essential
zip curl zlib1g-dev libc6-dev lib32ncurses5-dev ia32-libs x11proto-core-
dev libx11-dev lib32readline5-dev lib32z-dev libgl1-mesa-dev g++-
multilib mingw32 tofrodos python-markdown libxml2-utils xsltproc
```
4. **IMPORTANT!** Make bash the default shell (Android build scripts contain bash shell dependencies that require the system default shell /bin/sh to invoke bash) using one of the following options:
 - a. Reconfigure the package:
 - i. Use the command:

```
sudo dpkg-reconfigure dash
```
 - ii. Answer *no*.
 - b. Manually change the symlink /bin/sh→dash to /bin/sh→bash using the following commands:

```
sudo rm /bin/sh
sudo ln -s /bin/bash /bin/sh
```

NOTE: See the Ubuntu Wiki page at <https://wiki.ubuntu.com/DashAsBinSh> for more information.

8.1.3 Configuring Samba for Windows sharing (optional)

1. Use the following command to install the Samba server and configuration manager for Windows sharing:

```
sudo apt-get install samba system-config-samba
```

2. Configure the Samba server using:

```
System->Administration->Samba
preferences->server settings:
vmgroup, security=user authentication
encrypt pw=yes, guest acct=no guest acct
add share directory=/, share name=root, description=root directory
```

8.1.4 Installing JDK

The Sun JDK is no longer in Ubuntu's main package repository. In order to download it, add the appropriate repository and indicate to the system about the JDK being used.

```
sudo add-apt-repository "deb http://archive.canonical.com/ lucid partner"
sudo apt-get update
sudo apt-get install sun-java6-jdk
```

8.1.5 Installing Repo

The repo tool is a source code configuration management tool used by the Android project (see [R5]). It is a frontend to git written in Python that uses a manifest file to aid downloading code organized as a set of projects stored in different git repositories.

To install repo:

1. Create a ~/bin directory in your home directory, or, if you have root or sudo access, install for all system users under a common location, such as /usr/local/bin or somewhere under /opt.

3. Download the repo script.

```
$ curl https://dl-ssl.google.com/dl/googlesource/git-repo/repo
>~/bin/repo
```

4. Set the repo script attributes to executable.

```
$ chmod a+x ~/bin/repo
```

5. Include the installed directory location for repo in your PATH.

```
$ export PATH=~/bin:$PATH
```

6. Run **repo --help** to verify installation; you should see a message similar to the following:

```
$ repo --help
usage: repo COMMAND [ARGS]
repo is not yet installed. Use "repo init" to install it here.
The most commonly used repo commands are:
init          Install repo in the current working directory
```

`help` Display detailed help on a command

NOTE: For access to the full online help, install repo (repo init).

8.1.6 Installing the ARM Compiler Tools

Building the non-HLOS images requires the specific version of the ARM Compiler Tools indicated in Section 10.1.1. Linux is the recommended build environment for building all software images; however, either Windows or Linux-hosted versions work for building the non-HLOS images. For more information about the ARM Developer Suite and toolchains, go to the ARM support website at <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.subset.swdev.coretools/index.html>.

installing on a Linux host

1. Obtain the required ARM toolchain from your ARM vendor.
2. Follow the vendor instructions to install the toolchain and flex license manager onto your Linux build system.

Installing on a Windows host

1. Obtain the required ARM toolchain from your ARM vendor.
2. Follow the vendor instructions to install the toolchain and flex license manager onto your Windows build system.
3. Access the software from <https://silver.arm.com/download/download.tm?pv=1245960>.
4. The default install location is C:\Program Files (x86)\ARM_Compiler5\.
5. If necessary, change the directory where the files are extracted to match the location where you have installed the tools. For example, the installing directory for QTI is C:\Program Files (x86)\ARM_Compiler5\bin.
6. Confirm that the updated tools are installed by opening a DOS command prompt window and checking the versions for the compilers, linker, assembler, and fromelf.
7. To check the versions, run **armcc -vsn**. It must return the following:

```
ARM/Thumb C/C++ Compiler, 5.01 [Build 94]
For support contact support-sw@arm.com
Software supplied by: ARM Limited
armar --vsn
armlink --vsn
armasm --vsn
fromelf --vsn
```

The returned version must be Build 94 for all.

8.1.7 Installing the Hexagon toolchain

Linux is the recommended build environment for building all software images; however, either Windows or Linux-hosted versions work for building the non-HLOS images.

See *HexagonTools Installation Guide* (80-VB419-25) for detailed procedures to download and install the Hexagon toolchain software. For more documentation on using the Hexagon tools, see *Hexagon Development Tools Overview* (80-VB419-74) and *Hexagon LLVM C/C++ Compiler User Guide* (80-VB419-89).

Refer to *Presentation: LLVM Compiler Hexagon™ Processor Deployment Plan* (80-VB419-87) for a detailed explanation of the LLVM compiler in the Hexagon toolchain. LLVM compilers work with Hexagon software development tools and utilities to provide a complete programming system for developing high-performance software.

QUALCOMM®
106.37.221.50 2014.11.17 at 17:13:11 PST
xumingtao@hipad.com

8.1.8 Downloading and Building the Software

Table 10-2 describes the software for this product line divided into the release packages that must be downloaded separately and combined to have a complete product line software set.

Table 10-2 Release packages

From chipcode.qti.qualcomm.com [R6]	From codeaurora.org [R4]
Proprietary non-HLOS software Contains proprietary source and firmware images for all nonapps processors. This software is an umbrella package built from a combined set of individual component releases that have already been integrated.	Open source HLOS software Contains open source for apps processor HLOS
Proprietary HLOS software Contains proprietary source and firmware images for the apps processor HLOS.	—

The proprietary and open source HLOS packages must be obtained from separate sources and then combined according to the downloading instructions given in Section 10.2.2.10. Each package is identified by a unique build identification (build ID) code followed this naming convention:

<PL_Image>-<Version>-<Chipset>

Where:

- PL_Image – LNX.LA.Branch for Linux Android
- Version – Variable number of digits used to represent the build ID version.
- Chipset – 8x16 for MSM8916

LNX.LA.3.7.1-00710-8x16.0

Build is from the 3.7.1 branch

Version number

MSM8x16 chipset

Figure 10-1 Build ID naming convention

Figure 10-2 illustrates the combined software release packages.

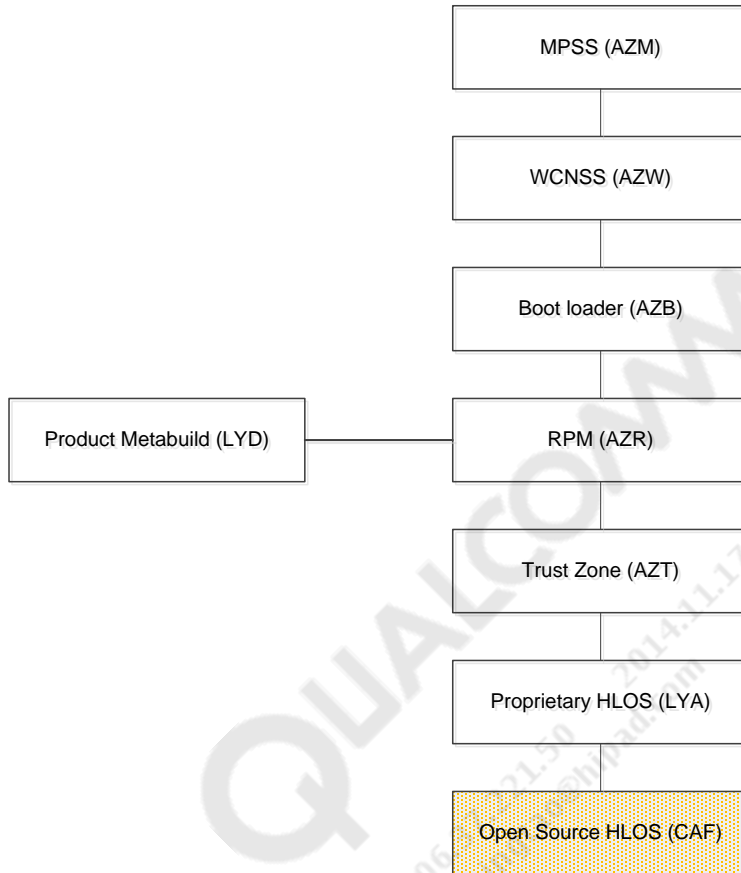


Figure 10-2 Combined software release packages

Table 10-3 gives the component release build properties. The compiler, Python, Perl, and Cygwin version information for each of the non-HLOS build modules is also provided. Make sure the build PC has the correct versions for each tool.

Table 10-3 Component release build properties





















Component build release	Source or binary only	Toolchain required for building source	Python version	Perl version	Cygwin	Supported build hosts
Android HLOS (LYA)	Source	Android gnu toolchain	—	—	—	Linux only
MPSS (AZM)	Source	Hexagon 6.2.09	Python 2.7.6 (64-bit)	Perl 5.10.1	Windows builds only; only needs tee.exe	Linux, Windows XP, and Windows 7
ADSP (AZL)	Binary	—	—	—	—	—

Component build release	Source or binary only	Toolchain required for building source	Python version	Perl version	Cygwin	Supported build hosts
Boot loaders (AZB)	Source	ARM Compiler Tools 5.01 update 3 (build 94)	Python 2.6.6	Perl 5.8.x Linux builds only	Windows builds only; only needs tee.exe	Linux, Windows XP, and Windows 7
RPM (AZR)	Source	ARM Compiler Tools 5.01 update 3 (build 94)	Python 2.6.2	Perl 5.6.1	Windows builds only; only needs tee.exe	Linux, Windows XP and Windows 7 only
TZ (AZT)	Source	ARM Compiler Tools 5.01 update 3 (build 94)	Python 2.6.6	—	Windows builds only; only needs tee.exe	Linux, Windows XP, and Windows 7 only
WCNSS (AZW)	Binary	ARM RVCT 2.2.1.593, ARMCT5.01, Hexagon 5.0 Toolset	Python 2.6.2	Not required unless you are using some tools to dump extraction. Recommended to use 5.8.x and above.	Windows builds only; only needs tee.exe	Linux, Windows XP, and Windows 7 only

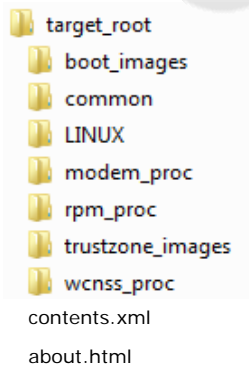
8.2 Downloading QTI software from ChipCode

NOTE: Numerous changes were made in this section.

1. The software distribution package (distro) is composed of multiple subsystem image files. Download the distro components to your build PC.

Name	Last Update	Last Commit > 0dbe4ad7752 – r106410.2 - ES1 1.0.6410.2
 LINUX	about 1 month ago	 QC Publisher r106410.2 - ES1 1.0.6410.2
 boot_images	about 1 month ago	 QC Publisher r106410.2 - ES1 1.0.6410.2
 common	about 1 month ago	 QC Publisher r106410.2 - ES1 1.0.6410.2
 modem_proc	about 1 month ago	 QC Publisher r106410.2 - ES1 1.0.6410.2
 rpm_proc	about 1 month ago	 QC Publisher r106410.2 - ES1 1.0.6410.2
 trustzone_images	about 1 month ago	 QC Publisher r106410.2 - ES1 1.0.6410.2
 wcns_proc	about 1 month ago	 QC Publisher r106410.2 - ES1 1.0.6410.2
 .gitattributes	about 1 month ago	 QC Publisher r106410.2 - ES1 1.0.6410.2
 about.html	about 1 month ago	 QC Publisher r106410.2 - ES1 1.0.6410.2
 contents.xml	about 1 month ago	 QC Publisher r106410.2 - ES1 1.0.6410.2

2. Create a top-level directory on the build PC and unzip each of the subsystem images to generate the following directory structure. In this example, <target_root> is the top-level directory.



3. Be sure to download the `about.html` and `contents.xml` files. These files contain the build ID that you will use to download the open source software from CAF.

8.2.1 Downloading open source Android HLOS software from CAF

NOTE: Numerous changes were made in this section.

2. See the release notes for the link to the open source software from CAF.

Alternatively, go to <https://www.codeaurora.org/xwiki/bin/QAEP/release> and find the release branch containing the matching build ID in the branch releases table.

2. In an empty directory, use the repo init command with the correct branch and manifest as indicated in the branch releases table.

```
$ repo init -u git://codeaurora.org/platform/manifest.git -b release -m 30 [manifest] -repo-url=git://codeaurora.org/tools/repo.git
```

3. Type the repo sync command.

```
$ repo sync
```

4. Copy the vendor/qcom/proprietary directory tree into the open source HLOS source tree contained in your workspace.

```
$cp -r <LYA_build_location>/HY11-<build_id>/LINUX/android/* .
```

Finding the Linux build ID for other releases

If you need to obtain the Linux build ID for another release for which you do not have the release notes, the build ID can be obtained in the following ways:

1. From ChipCode – On the distro page, the about.html section displays the build ID.

Build Components: Build ID

Image	Build/Label	Distro Path	Format
LNx.LA.x.x	LNx.LA.0.0-03620-8x16_32_64.0-1	LINUX	SRC
BOOT.BF.x.x	BOOT.BF.3.0-00060-M8916AAAAANAZB-2	boot_images	SRC
MSM8916.LA.x.x	M8916AAAAANLYD106410.2	common	SRC
MSM8916.LA.x.x	M8916AAAAANLYD106410.2	contents.xml	SRC
MPSS.DPM.x.x	MPSS.DPM.1.0-00369-M8916EAAAANVZM-2	modem_proc	SRC
RPM.BF.x.x	RPM.BF.2.0-00040-M8916AAAAANAZR-1	rpm_proc	SRC
TZ.BF.x.x	TZ.BF.3.0-00028-M8916AAAAANAZT-1	trustzone_images	SRC
CNSS.PR.x.x	M8916AAAAANAZW142008.1	wcnss_proc	BIN

2. From the about.html file that was downloaded to the build PC.
3. From the contents.xml file that was downloaded to the build PC. Search for <name>apps</name>, then locate the build_id.

```
<build>
```

```
<name>apps</name>
```

```
<role>apps</role>
<chipset>msm8926</chipset>
<build_id>LNX.LA.0.0-03620-8x16_32_64.0-1</build_id>
```

8.2.2 Compiling the Non-HLOS software

8.2.2.1 Setting build Windows environment

Before issuing the non-HLOS build commands, certain command environment settings must be set to ensure the correct executable path and toolchain configuration. The specific environment settings vary based on your host software installation, but it is similar to the example “myenviron_amss.cmd” script below (for Windows), which sets the path to point to the ARM toolchain lib, include, bin, and license file configuration.

```
#
# myenviron_amss_8916
#
SET ARMLMD_LICENSE_FILE=<mylicense_file>@<mylicense_server>
set ARM_COMPILER_PATH=C:\apps\ARMCT5.01\94\bin64
set PYTHON_PATH=C:\Python26
set PYTHONPATH=C:\Python26
set MAKE_PATH=C:\apps\ARMCT5.01\94\bin64
set GNUPATH=C:\cygwin\bin
set CRMPERL=C:\Perl64\bin
set PERLPATH=C:\Perl64\bin

set ARMHOME=C:\Apps\ARMCT5.01\94
set ARMINC=C:\Apps\ARMCT5.01\94\include
set ARMLIB=C:\Apps\ARMCT5.01\94\lib
set ARMBIN=C:\Apps\ARMCT5.01\94\bin
set ARMPATH=C:\Apps\ARMCT5.01\94\bin
set ARMINCLUDE=C:\Apps\ARMCT5.01\94\include
set ARMTOOLS=ARMCT5.01
set
PATH=.;C:\Python26;C:\Apps\ARMCT5.01\94\bin;C:\apps\ARMCT5.01\94\bin64;C:\c
ygwin\bin;%PATH%
set HEXAGON_ROOT=C:\Qualcomm\HEXAGON_Tools
set HEXAGON_RTOS_RELEASE=5.0.07
set HEXAGON_Q6VERSION=v4
set HEXAGON_IMAGE_ENTRY=0x08400000
```

8.2.2.2 Building MPSS

NOTE: Numerous changes were made in this section.

To build MPSS (<target_root> is the top-level directory that is created in Section 10.2):

1. For Linux, verify that the paths below have been set by referring to setenv.sh in the build.

```
<target_root>\modem_proc\build\ms\setenv.sh
export ARMLMD_LICENSE_FILE=<LICENSE FILE INFO>
ARM_COMPILER_PATH=/pkg/qct/software/arm/RVDS/2.2BLD593/RVCT/Programs/2.2
/593/linux-pentium
PYTHON_PATH=/pkg/qct/software/python/2.6.6/bin
MAKE_PATH=/pkg/gnu/make/3.81/bin
export ARMTTOOLS=RVCT221
export ARMROOT=/pkg/qct/software/arm/RVDS/2.2BLD593
export ARMLIB=$ARMROOT/RVCT/Data/2.2/349/lib
export ARMINCLUDE=$ARMROOT/RVCT/Data/2.2/349/include/unix
export ARMINC=$ARMINCLUDE
export ARMCONF=$ARMROOT/RVCT/Programs/2.2/593/linux-pentium
export ARMDLL=$ARMROOT/RVCT/Programs/2.2/593/linux-pentium
export ARMBIN=$ARMROOT/RVCT/Programs/2.2/593/linux-pentium
export PATH=$MAKE_PATH:$PYTHON_PATH:$ARM_COMPILER_PATH:$PATH
export ARMHOME=$ARMROOT
export HEXAGON_ROOT=/pkg/qct/software/hexagon/releases/tools
```

2. Navigate to the following directory:

```
cd <target_root>/modem_proc/build/ms
```

3. Depending on your build environment, use one of the following commands:

Build environment	Command
Linux	Build images – ./build.sh 8916.gen.prod -k Clean the build – ./build.sh 8916.gen.prod -k -c Notices: For CMCC – EAAAAANYZM: ./build.sh 8916.lwgn.s.prod -k & EAAAAANWZM: ./build.sh 8916.lwg.prod -k (depending on if customer wants to launch with segment loading on/off) For CU –: EAAAAANWZM: ./build.sh 8916.lwg.prod -k For CT –EAAAAANVZM: ./build.sh 8916.gen.prod -k 1x Compiled out with Segment loading enabled: EAAAAANWZM: ./build.sh 8916.lwg.prod -k 1x Compiled out with Segment loading disabled: EAAAAANYZM: ./build.sh 8916.lwgn.s.prod -k
Windows	Build images – build.cmd 8916.gen.prod -k Clean the build – build.cmd 8916.gen.prod -k -c

8.2.2.3 Building boot loaders

To build the boot loaders:

1. For Linux, verify that the paths below have been set. Refer to setenv.sh in your boot build:

```
<target_root>\boot_images\build\ms\setenv.sh.
export ARMLMD_LICENSE_FILE=<LICENSE FILE INFO>
export ARM_COMPILER_PATH=/<>Path to compiler>/arm/RVDS/5.01bld94/bin64
export PYTHON_PATH=/<>Path to python>/python/2.6.6/bin
export MAKE_PATH=/<>Path to make>/gnu/make/3.81/bin
export ARMTTOOLS=ARMCT5.01
export ARMROOT=/<>Path to compiler>/arm/RVDS/5.01bld94
export ARMLIB=$ARMROOT/lib
export ARMINCLUDE=$ARMROOT/include
export ARMINC=$ARMINCLUDE
export ARMBIN=$ARMROOT/bin64
export PATH=$MAKE_PATH:$PYTHON_PATH:$ARM_COMPILER_PATH:$PATH
export ARMHOME=$ARMROOT
export_arlmd_license
```

2. Navigate to the following directory:

```
cd <target_root>/boot_images/build/ms
```

Where <target_root> is the top-level directory that is created in Section 10.2.

3. Depending on your build environment/release, choose one of the following build command options:

Build environment	Command
Linux	Build boot images \$./build.sh TARGET_FAMILY=8916 BUILD_ID=HAAAAANAZ Cleaning the build \$./build.sh -c TARGET_FAMILY=8916 BUILD_ID=HAAAAANAZ Depending on your configuration, edit the script boot_images/build/ms/build_8916.sh to change if [-e "setenv.sh"]; then - source setenv.sh + source ./setenv.sh fi
Windows	Building boot images build.cmd TARGET_FAMILY=8916 BUILD_ID=HAAAAANAZ Cleaning the build build.cmd -c TARGET_FAMILY=8916 BUILD_ID=HAAAAANAZ

8.2.2.4 Building TrustZone images

To build the MSM8916 TrustZone images:

1. Navigate to the following directory:

```
cd <target_root>/trustzone_images/build/ms
```

- Run the following command to build all images:

Build environment	Command
Linux	Build images – ./build.sh CHIPSET=msm8916 tz hyp sampleapp tzbsp_no_xpu playready widevine isdbtmm securitytest keymaster commonlib Clean the build – ./build.sh CHIPSET=msm8916 tz hyp sampleapp tzbsp_no_xpu playready widevine isdbtmm securitytest keymaster commonlib -c
Windows	Build images – build.cmd CHIPSET=msm8916 tz hyp sampleapp tzbsp_no_xpu playready widevine isdbtmm securitytest keymaster commonlib Clean the build – build.cmd CHIPSET=msm8916 tz hyp sampleapp tzbsp_no_xpu playready widevine isdbtmm securitytest keymaster commonlib -c

8.2.2.5 Building RPM

Use the following commands to build RPM (<target_root> is the top-level directory). Ensure that your tools are the versions specified in [Table 10-3](#).

NOTE: Building on Linux may not work for early ES releases.

- Open a command prompt and change to the following directory:
cd <target_root>\rpm_proc\build
- Depending on your build environment, use one of the following commands:

Build environment	Command
Linux	Build images – ./build_8916.sh 8916 Clean the build – ./build_8916.sh 8916 -c
Windows	Build images – build_8916.bat 8916 Clean the build – build_8916.bat 8916 -c

NOTE: rpm.mbn are found at rpm_proc\build\ms\bin\AAAAANAAR.

8.2.2.6 Building WCNSS

Use the following commands to build WCNSS (<target_root> is the top-level directory). Ensure that your tools are the versions specified in [Table 10-3](#).

- Open a command prompt and change to the following directory:
cd <target_root>\wcns_proc\Pronto\bsp\build\
- Depending on your build environment, use one of the following commands:

Build environment	Command
Linux	Build images – source ./wcns_build.sh 8916 pronto Clean the build – source ./wcns_build.sh 8916 pronto -c
Windows	Build images – wcns_build.cmd 8916 pronto Clean the build – wcns_build.cmd 8916 pronto -c

8.2.2.7 Building ADSP

The release provides prebuilt ADSP images. In cases where prebuilt images are not provided, the following commands can be used to build ADSP images (<target_root> is the top-level directory) from source codes. Ensure that the tools are the versions specified in [Table 10-3](#).

1. Open a command prompt and change to the following directory:

```
cd <target_root>\adsp_proc\build\
```

2. Depending on the build environment, use one of the following commands:

Build environment	Command
Linux	Build images – python build.py Clean the build – python build.py -o clean
Windows	Build images – python build.py Clean the build – python build.py -o clean

8.2.2.8 Updating NON-HLOS.bin

If any MPSS, ADSP, or WCNSS is recompiled, use the following commands to update the NON-HLOS.bin file with the new images (<target_root> is the top-level directory that is created in [Section 10.2](#)):

1. Navigate to the following directory:

```
cd <target_root>/common/build
```

2. Enter the command:

```
python update_common_info.py
```

NOTE: If you are also creating the whole sparse_images (which is required when contents.xml is to be burned to be a whole image using Qualcomm Product Support Tool (QPST)), the image compiled from Linux/Android must be copied into the LINUX/android/out/target/product/msm8916_32_k64/ directory. For details about the required files, refer to apps-related configuration information in the contents.xml file. In Linux, you are advised to copy Linux/android code downloaded from codeaurora.org to the LINUX/android directory downloaded from Qualcomm ChipCode for compilation.

8.2.2.9 Updating single images 8916_msimage.mbn and MPRG8916.mbn

Dedicated compilation commands are required to update single images. If the default single image has an issue, e.g., DDR configuration parameters have changed, it needs to be recompiled.

1. Check if the following images exist based on information in the contents.xml file:

```
sbl1.mbn: <target_root>\boot_images\build\ms\bin\8916\  
rpm.mbn: <target_root>\rpm_proc\build\ms\bin\AAAAANAAR\  
tz.mbn: <target_root>\trustzone_images\build\ms\bin\MAUAANAA\
```

2. Run the following commands to create a single image:

```
python singleimage.py -x singleimage_partition_8974.xml -  
search_path=<target_root>\boot_images\build\ms\bin\8916\ --  
search_path=<target_root>\rpm_proc\build\ms\bin\AAAAANAAR\ --  
search_path=<target_root>\trustzone_images\build\ms\bin\MAUAANAA\
```

3. If the creation is successful, a log similar to the following is printed:


```
SUCCESS - singleimage.bin created  
Filename: 'singleimage.bin' (1.29 MB)
```

4. Rename singleimage.bin and <<target_root>\boot_images\build\ms\bin\8916\emmcblld.mbn as 8916_msimage.mbn and MPRG8916.mbn, respectively.
5. Copy the two files to boot_images/build/ms/bin/EMMCBLD/.

8.2.2.10 Building the Apps processor Android HLOS

1. In a BASH shell, navigate to the Android source tree base directory.
`cd <build id>/LINUX/android`
2. Enter the following command to configure the build environment shell settings:
`source build/envsetup.sh`

NOTE: You must use the source command so the environment settings are defined in the current shell.

3. Enter the lunch command to select the build configuration, or enter with no parameters to see an interactive menu for making selections.
`lunch msm8916-userdebug`
4. Run make to start the build. To run parallel builds for faster build times on a multicore build machine, run the following command:
`make -j4`

8.3 Firmware Programming

8.3.1 Required software

Table 10-4 lists the software required for programming firmware images into a target device.

Table 10-4 Required software

	Item description	Version	Source/vendor	Purpose
1	QPST	2.7.405 or later	Qualcomm Technologies, Inc.	Programming firmware images using QPST
2	QXDM Professional™	3.14.447 or later	Qualcomm Technologies, Inc.	Programming NV Item values, reading diagnostic, etc.
3	Lauterbach T32 ARMv8 License Extension	LA-3743X	Lauterbach GmbH	Programming firmware images using JTAG and applications processor debugging
4	Lauterbach T32 Hexagon License Extension	LA-3741A	Lauterbach GmbH	Modem software processor, firmware processor, ADSP debugging using JTAG
5	Lauterbach T32 Cortex-M3 License Extension	LA-7844X or LA-7844	Lauterbach GmbH	Programming firmware images using JTAG and RPM debugging using JTAG
6	Lauterbach T32 ARM9™ License Extension	LA-7742X	Lauterbach GmbH	Venus and WCNSS debugging using JTAG
7	Lauterbach T32 Windows	Aug 2012 Software Version: R.2012.08.000040902 Build: 38589--40902.	Lauterbach GmbH	Programming firmware images and debugging using JTAG
8	Android SDK tools (Host USB drivers, adb, fastboot)	r10 or higher ADB 1.0.29 or later	Android Open Source Project	Windows host USB driver for adb and fastboot; adb and fastboot tools for Windows
9	Qualcomm USB Network Driver Combo	1.0.80 or later	Qualcomm Technologies, Inc.	Windows host USB drivers for Qualcomm composite devices

8.3.2 Installing T32

QPST must be used for firmware download; however, T32 can be used when QPST download does not work. The Dec 2013 Build 49679 version of T32 is the mandatory minimum revision that is needed for binary download and debugging. The T32 links under common\t32\t32_dap\ should be used for binary download and debugging.

By default, these files assume the T32 installing directory is C:\T32. If T32 is installed in a different directory, the .lnk shortcut files must be modified.

To modify the .lnk shortcut files:

1. Locate the .lnk shortcut files.
2. Right-click the mouse and select **Properties**.

3. In the Target field, change the path to the proper path of t32marm.exe. The default path is C:\t32\t32marm.exe.

8.3.3 Installing Android ADB, Fastboot, and USB driver for Windows

Android CDP support requires the following USB device support:

- Android USB Driver (android_winusb.inf)
 - a. Android ADB Interface
 - b. Android Boot Loader Interface (fastboot)
 - Qualcomm Composite USB Modem/Serial Driver (qcmdm.inf, qcser.inf)
 - a. Qualcomm HS-USB Android DIAG
 - b. Qualcomm HS-USB Android Modem
 - c. Qualcomm HS-USB Android GPS (NMEA)
 - Qualcomm Composite USB Network Combo driver (qcnet.inf)
 - a. Qualcomm Wireless HS-USB Ethernet Adapter
1. Before installing the drivers, edit the qcmdm.inf and qcser.inf files to make sure that they contain support for the Android SURF VID/PID with appropriate entries in each section as indicated in Step 4.
 2. Go to <http://developer.android.com/sdk/win-usb.html>, and follow the instructions to install the SDK and USB driver.
 3. Right-click **My Computer**, and select **Properties**→**Advanced**→**Environment Variables**, and set the path to include the c:\android-sdk-windows\tools directory.
 4. The Android USB driver for ADB and Fastboot needs to add the Qualcomm SURF VID/PID, which supports the connection to the URF. Edit the file android-sdk-windows\usb_driver\android_winusb.inf to add the Qualcomm VID/PID lines to each section.

android_winusb.inf

[Google.NTx86]

;Qualcomm SURF/FFA

%SingleAdbInterface% = USB_Install, USB\VID_05C6&PID_9025

%CompositeAdbInterface% = USB_Install, USB\VID_05C6&PID_9025&MI_01

%SingleBootLoaderInterface% = USB_Install, USB\VID_18D1&PID_D00D

[Google.NTamd64]

;Qualcomm SURF/FFA

%SingleAdbInterface% = USB_Install, USB\VID_05C6&PID_9025

%CompositeAdbInterface% = USB_Install, USB\VID_05C6&PID_9025&MI_01

%SingleBootLoaderInterface% = USB_Install, USB\VID_18D1&PID_D00D

In addition, make sure that there are matching entries under the [Strings] section.

[Strings]

SingleAdbInterface = "Android ADB Interface"

CompositeAdbInterface = "Android Composite ADB Interface"

SingleBootLoaderInterface = "Android Bootloader Interface"

5. The ADB client (adb.exe) supports a built-in list of recognized USB VID/PID devices. To add the SURF or another device to the list of recognized devices, which is not included in the built-in support list, create a %USERPROFILE%\android directory if it does not exist.
6. Navigate to the %USERPROFILE%\android directory.
7. In the %USERPROFILE%\android directory, create /edit the adb_usb.ini file. If the file exists, it contains a DO NOT EDIT message. Disregard this message and edit the file anyway. Add a line containing 0x05C6 to the end of the file.

NOTE: Do not run android update ADB or it resets the contents of this file and overwrite the line just added.

After editing, the adb_usb.ini file must look like the following:

```
# ANDROID 3RD PARTY USB VENDOR ID LIST-DO NOT EDIT.
# USE 'android update adb' TO GENERATE.
# 1 USB VENDOR ID PER LINE.
0x05C6
```

8. Obtain the latest version of the Qualcomm Composite USB driver from Documents and Downloads. (To include network interface support, use the Qualcomm Composite USB Network Combo driver.)

Android debugging is enabled/disabled in user space with composition 9025/9026 respectively.

qcmdm.inf

[Models]

%QUALCOMM90252% = Modem2, USB\VID_05C6&PID_9025&MI_02

%QUALCOMM90261% = Modem2, USB\VID_05C6&PID_9026&MI_01

[Models.NTamd64]

%QUALCOMM90252% = Modem2, USB\VID_05C6&PID_9025&MI_02

%QUALCOMM90261% = Modem2, USB\VID_05C6&PID_9026&MI_01

[Models.NTia64]

%QUALCOMM90252% = Modem2, USB\VID_05C6&PID_9025&MI_02

%QUALCOMM90261% = Modem2, USB\VID_05C6&PID_9026&MI_01

[Strings]

QUALCOMM90252 = "Qualcomm Android Modem 9025"

QUALCOMM90261 = "Qualcomm HS-USB Android Modem 9026"

qcser.inf

[QcomSerialPort]

%QcomDevice90250% = QportInstall00, USB\VID_05C6&PID_9025&MI_00

%QcomDevice90253% = QportInstall00, USB\VID_05C6&PID_9025&MI_03

%QcomDevice90260% = QportInstall00, USB\VID_05C6&PID_9026&MI_00

%QcomDevice90262% = QportInstall00, USB\VID_05C6&PID_9026&MI_02

[QcomSerialPort.NTia64]

%QcomDevice90250% = QportInstall00, USB\VID_05C6&PID_9025&MI_00

%QcomDevice90253% = QportInstall00, USB\VID_05C6&PID_9025&MI_03

```

%QcomDevice90260% = QportInstall00, USB\VID_05C6&PID_9026&MI_00
%QcomDevice90262% = QportInstall00, USB\VID_05C6&PID_9026&MI_02
[QcomSerialPort.NTamd64]
%QcomDevice90250% = QportInstall00, USB\VID_05C6&PID_9025&MI_00
%QcomDevice90253% = QportInstall00, USB\VID_05C6&PID_9025&MI_03
%QcomDevice90260% = QportInstall00, USB\VID_05C6&PID_9026&MI_00
%QcomDevice90262% = QportInstall00, USB\VID_05C6&PID_9026&MI_02
[Strings]
QcomDevice90250 = "Qualcomm HS-USB Android DIAG 9025"
QcomDevice90253 = "Qualcomm HS-USB Android GPS (NMEA)9025"
QcomDevice90260 = "Qualcomm HS-USB Android DIAG 9026"
QcomDevice90262 = "Qualcomm HS-USB Android GPS (NMEA)9026"
qcnet.inf
[QCOM]
qcwwan.DeviceDesc90254 = qcwwan.ndi, USB\VID_05C6&PID_9025&MI_04
qcwwan.DeviceDesc90263 = qcwwan.ndi, USB\VID_05C6&PID_9026&MI_03

[QCOM.NTia64]
qcwwan.DeviceDesc90254 = qcwwan.ndi, USB\VID_05C6&PID_9025&MI_04
qcwwan.DeviceDesc90263 = qcwwan.ndi, USB\VID_05C6&PID_9026&MI_03
[QCOM.NTamd64]
qcwwan.DeviceDesc90254 = qcwwan.ndi, USB\VID_05C6&PID_9025&MI_04
qcwwan.DeviceDesc90263 = qcwwan.ndi, USB\VID_05C6&PID_9026&MI_03
[Strings]
qcwwan.DeviceDesc90254 = "Qualcomm Wireless HS-USB Ethernet Adapter
9025"
qcwwan.DeviceDesc90263 = "Qualcomm Wireless HS-USB Ethernet Adapter
9026"

```

8.3.4 Installing adb and fastboot in Linux

1. Before installing adb, modify the USB driver by navigating to the following directory:
`cd /etc/udev/rules.d/`

2. Enter the command:
`sudo vi 50-android.rules`

The result must be similar to the following:

```

#Sooner low-level bootloader
SUBSYSTEM=="usb", SYSFS{idVendor}=="18d1", SYSFS{idProduct}=="d00d",
MODE="0664", GROUP="plugdev"
# adb composite interface device 9025
SUBSYSTEM=="usb", SYSFS{idVendor}=="05C6", SYSFS{idProduct}=="9025",
MODE="0664", GROUP="plugdev"

```

3. After editing the file, to see the list of target devices connected to the Linux box, type:

Lsusb

4. The adb and fastboot executables for Linux are located in the android\out\host\linux-x86\bin directory in the Android software release after a build is complete. If the android\out\host\linux-x86\bin directory is not in the executable search path, use the following steps to add it. If it is already in the executable search path, skip to Step 6:
 - a. Type the command:
`source build/envsetup.sh`
 - b. Type the command:
`choosecombo 1 msm8916 userdebug`
5. Verify that fastboot has properly flashed the Android images to the target, type the command:

`sudo fastboot devices`
6. Verify that device is displayed by fastboot in response to typing in the fastboot devices command.

NOTE: To run adb or fastboot, sudo or root access on the Linux machine may be required.

8.3.5 Programming procedures

This section describes the procedures to be used for programming and reprogramming each firmware image and device.

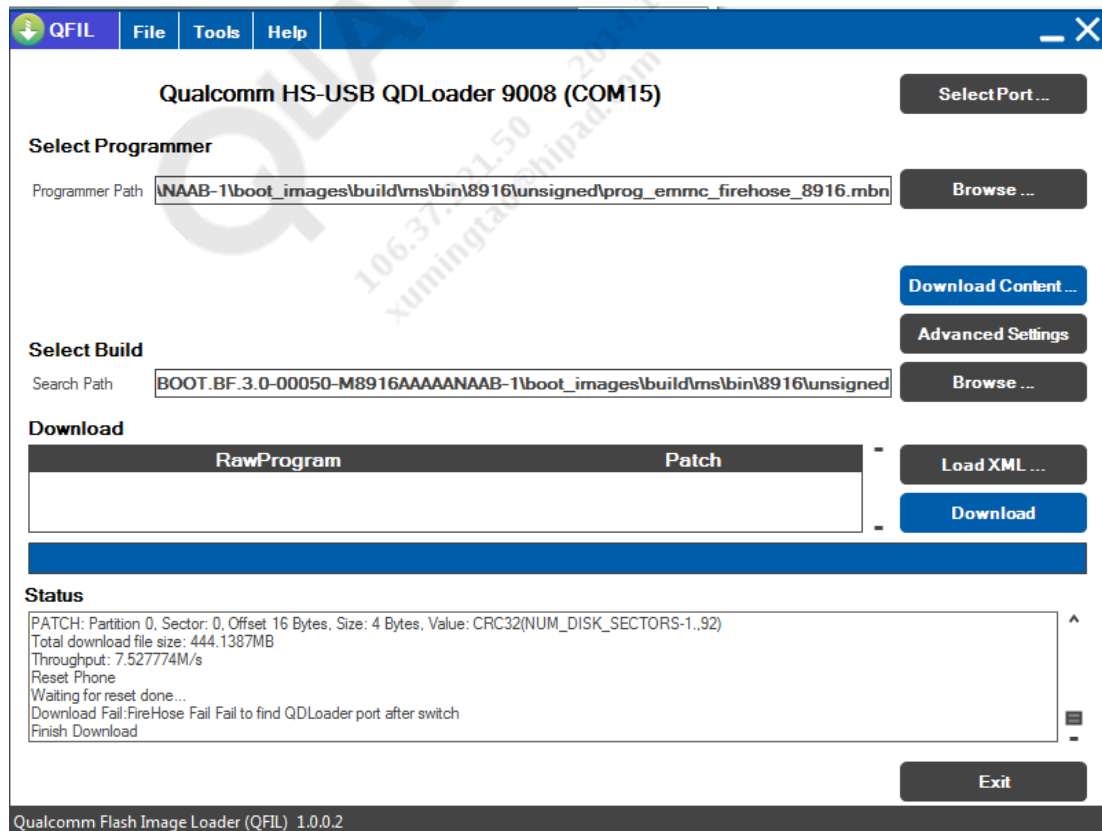
8.3.5.1 Programming eMMC with QPST

If no image is flashed to eMMC (this is the situation for the first-time binary download in the factory line), the PBL enumerates USB as the Qualcomm Sahara download interface waits for the download command from the host PC. In this case, the minimum initial binary is needed to be flashed so that the rest of the Android images can be downloaded.

NOTE: On the MSM8916 CDP, you can force the device into this mode by erasing the device entirely or using the dip switch S7 pin 3.

NOTE: Your version of QPST must meet the minimum requirement specified in Section 10.3.1.

1. Launch QFIL from QPST. If USB 9008 port is detected in Windows Device Manager, it is automatically detected.



2. Select programmer path; found in the following folder:
boot_images\build\ms\bin\8916\unsigned\prog_emmc_firehose_8916.mbn.
3. Click **Download Content** and select metabuild's contents.xml.
4. Click **Download**.

See *Qualcomm Flash Image Loader (QFIL)* 80-NN120-1

NOTE: for more information on QFIL.

NOTE: If you use QPST to burn a board that has already been burned with software, you need to disable the corresponding port on the QPST Configuration interface, as shown in [Figure 10-3](#). For boards that cannot enter the Download mode, try again after installing the batteries.

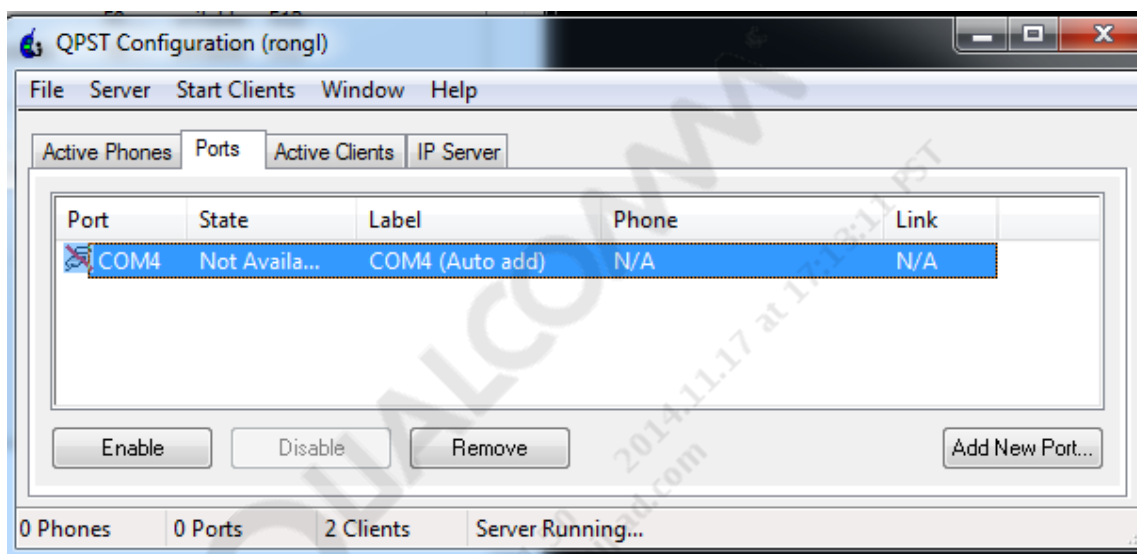


Figure 10-3 QPST configuration

NOTE: It is recommended to flash CDT if it is the first download on a blank eMMC. CDT provides platform/device-dependent data, such as Platform ID, DDR hardware parameters, etc. If sb11 succeeds to read the CDT from eMMC, the default `config_data_table` array is updated which is linked into the build during compile. To program CDT to eMMC boot partition, see [section 10.3.5.2](#).

8.3.5.2 Programming CDT using QPST

Ensure that you have the following to program CDT using the QPST tool:

1. Flash programmer – MPRG8916.mbn
2. XML file – To load the flash programmer, use sahara.xml
3. XML file – To load the configuration file that specifies the search paths and filenames for rawprogram and patch xml files. Use `qrd_prog_cfg.xml` with the following code:

```
<?xml version="1.0"?>
<configuration>
  <options>
    <!-- NOTE: Defaults are shown here -->
    VERBOSE_FEEDBACK      = true
    SEARCH_CWD_LAST       = true
    HEX_PROGRAMMER        = MPRG8916.mbn
```



```

    </options>
    <search_paths>
<!-- NOTE: Search paths IN ORDER, as in first look here, then there,
then there etc -->
    <!-- NOTE: CWD is an implied search path, and it is always last -->
    emmc_cdt_program_script\cdt_prog_xml
    emmc_cdt_program_script\cdt_bin
    </search_paths>
    <rawprogram>
    rawprogram2_qrd.xml
    </rawprogram>
    <patch>
    patch2.xml
    </patch>
</configuration>

```

4. XML file – This file is found in the release folder or applied from customer support services. The rawprogram2_qrd.xml and patch2.xml are used to load the CDT binary that was created by CDT cdt_generator.py and CDT xml:

```

cd boot_images\core\boot\secboot3\scripts
python cdt_generator.py qrd_1.0_platform_jedec_lpddr2_single_channel.xml
qrd_1.0_platform_jedec_lpddr2_single_channel.bin

```

To program CDT using QPST:

1. Ensure that the phone is in Emergency Download mode

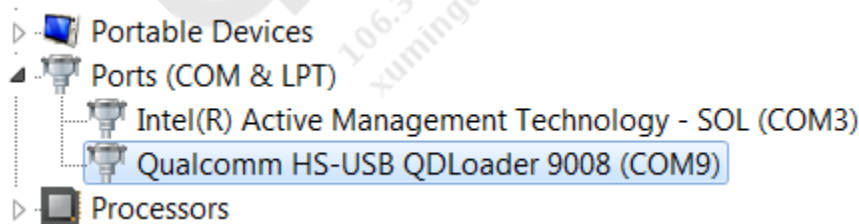


Figure 10-4 Emergency download port

2. Open eMMC software download application (run as administrator), provide the sahara xml file, and uncheck "Program Boot Loaders", "Program MMC device", and "NV Backup".

- Click "Load Configuration and start download..." and select qrd_prog_cfg.xml to program the CDT as shown in Figure 10-5:

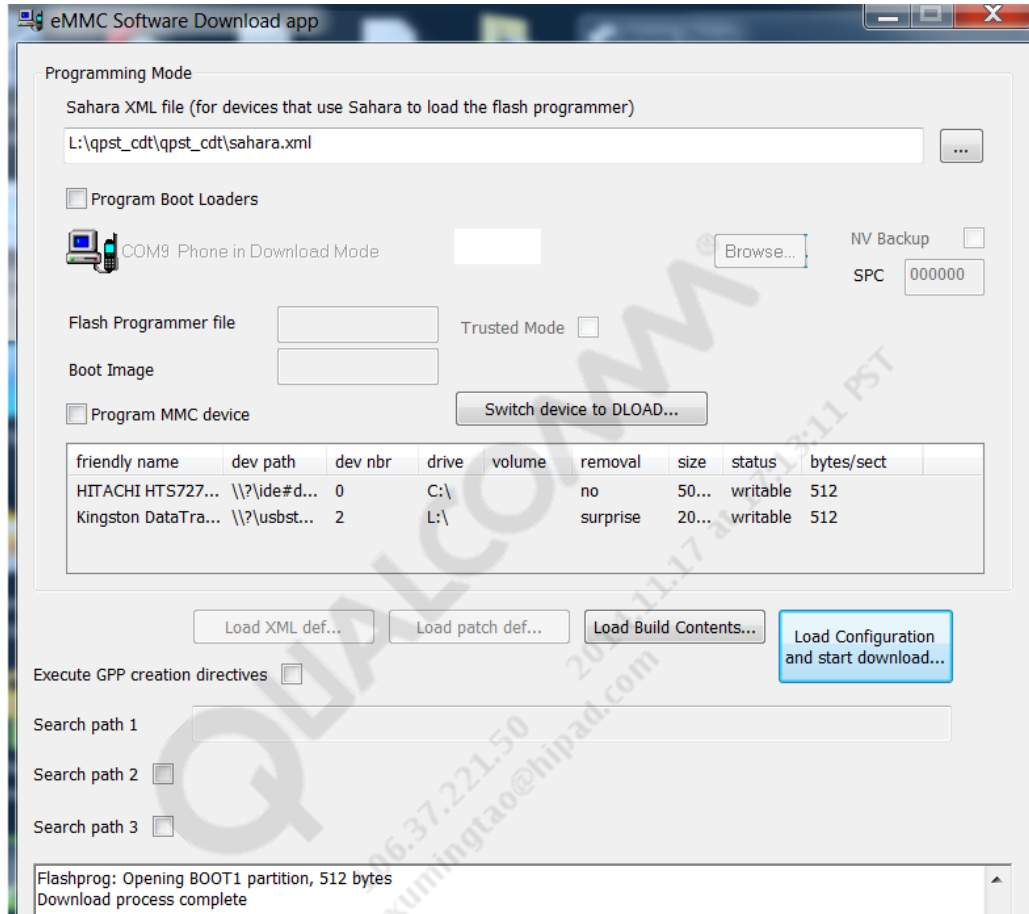
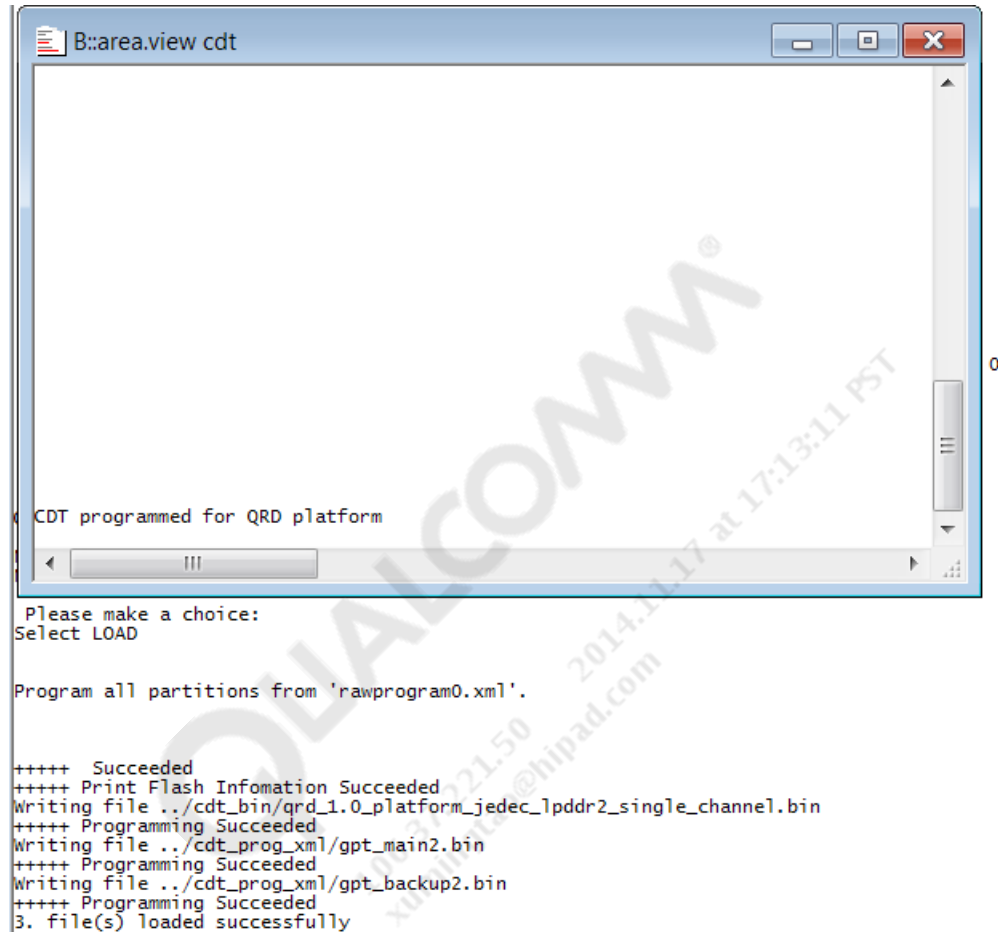


Figure 10-5 Flash CDT with QPST

8.3.5.3 Programming CDT using JTAG:

- Set up T32 debug environment first (see section 10.3.5.4)
- Run the CDT script in CortexA53Core0 session. The following information is shown if the script succeeds.

```
cd.do C:\emmc_cdt_program_script\
qrd_1.0_lpddr2_single_channel_emmc_cdt_program.cmm
```



8.3.5.4 Programming system images using Fastboot

Before programming the system images, use Fastboot. The Android boot loaders must already be flashed on the target:

1. Plug the USB cable into the target.
2. Depending on your build environment, choose one of the following options:
 - a. From the Windows command shell, run:


```
fastboot devices
```
 - From Linux:
 - i. Navigate to the following directory:


```
cd <AndroidRoot>/LINUX/device/out/host/linux-x86/bin
```
 - ii. Run:


```
sudo fastboot devices
```

The list of registered devices is shown.

3. Once the device is detected, Flash the binaries to the target. The following commands run all the Fastboot steps at once.

```
cd <target_root>/common/build
fastboot_all.py
```

Each binary can also be flashed selectively through the following fastboot command options:

```
fastboot flash modem <path to NON-HLOS.bin> or <path to APQ.bin>
fastboot flash sbl1 <path to sbl1.mbn>
fastboot flash rpm <path to rpm.mbn>
fastboot flash QSEE <path to tz.mbn>
fastboot flash about <path to emmc_appsboot.mbn >
fastboot flash boot <path to boot.img>
fastboot flash system <path to system.img>
fastboot flash userdata <path to userdata.img>
fastboot flash persist <path to persist.img>
fastboot flash recovery <path to recovery.img>
```

To derive a list of all fastboot partitions supported by fastboot programming, refer to the source code in LINUX/android/bootable/bootloader/lk/platform/msm_shared/mmc.c.

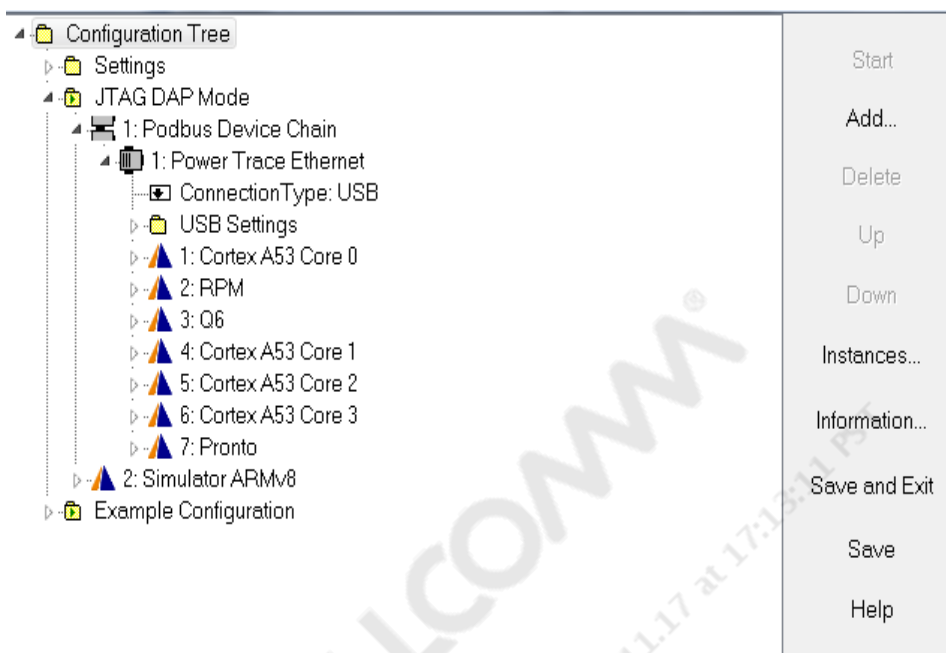
8.3.5.5 Flashing applications to Android using ADB

1. Plug the USB cable into the target.
2. Navigate to the following directory:
`cd <root>/LINUX/device/out/host/linux-x86/bin`
3. Enter the command below to register a device:
`sudo adb devices`
4. Navigate to the following directory:
`cd <root>/LINUX/device/out/target/product/surf/obj/APPS/AppName_intermediates/`
5. Copy the files:
`cp package.apk AppName.apk`
6. Push the files as follows:
`adb push AppName.apk /system/app/.`

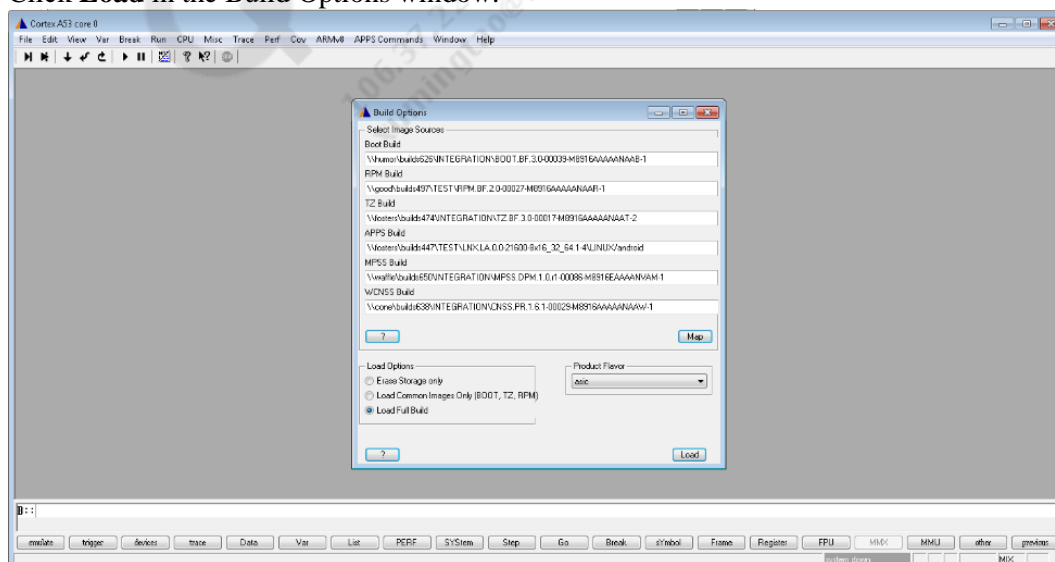
NOTE: In general, the syntax is: `adb push <file_name> <location_on_the_target>`

8.3.5.6 Programming eMMC boot loaders with T32

1. Start T32 by using the t32start.cmd in the <meta build>\common\t32 folder.
2. Navigate to Configuration Tree→JTAG DAP Mode→Podbus Device Chain→Power Trace Ethernet.

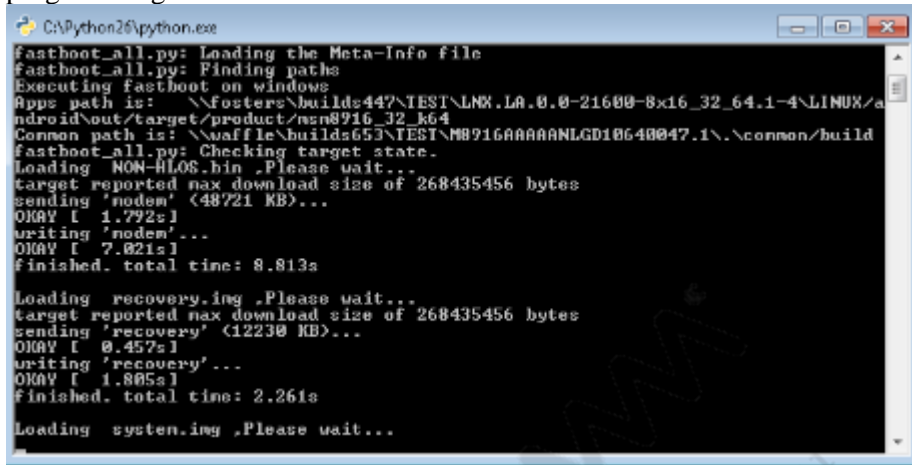


3. Select CortexA53Core0 and click **Start**.
4. In CortexA53Core0 T32 window, click **APPS COMMANDS**→**Build Options**, then select **ASIC** flavor in “Product Flavor” field and click **MAP**.
5. Click **Load** in the Build Options window.



6. Keep your USB connecting to PC, and it uses fastboot to flash **NON-HLOS.bin** and apps binaries. Fastboot download process will start automatically by script after T32 finishes

programming boot loaders.



```
C:\Python26\python.exe
fastboot_all.py: Loading the Meta-Info file
fastboot_all.py: Finding paths
Executing fastboot on windows
Apps path is: \\fosters\builds447\TEST\LNK.LA.0.0-21600-8x16_32_64.1-4\Linux/a
ndroid\out\target\product\msn8916_32_k64
Common path is: \\waffle\builds653\TEST\M8916AAAAANLGD10640047.1\.\common\build
fastboot_all.py: Checking target state.
Loading NON-HLOS.bin .Please wait...
target reported max download size of 268435456 bytes
sending 'nodem' (48721 KB)...
OKAY [ 1.772s]
writing 'nodem'...
OKAY [ 7.021s]
finished. total time: 8.813s

Loading recovery.img .Please wait...
target reported max download size of 268435456 bytes
sending 'recovery' (12230 KB)...
OKAY [ 0.457s]
writing 'recovery'...
OKAY [ 1.805s]
finished. total time: 2.261s

Loading system.img .Please wait...
```

7. After fastboot completes flashing the binaries, power cycle the device.

8.4 Operational Guide

8.4.1 Common NV settings

8.4.1.1 RF NV settings

Static QCN is generated from the XML shipped within the MPSS build. Pick the appropriate xml file from the following path:

<MPSS ROOT>\modem_proc\rftarget_dimepm\common \rftarget_dimepm\common\qcn\

Choose an RF card per your requirements from [Table 10-5](#).

To generate a static QCN, select the RF configuration to be used and then convert it to QCN using QRCT tool:

1. Run QRCT.
2. Navigate to Tool→NV tools and pick the master xml file from the following build path:

<MPSS ROOT>\modem_proc\rftarget_dimepm\common \rftarget_dimepm\common\qcn\

For example, for RF card: RFC_WTR1605_CHILE_SVDSDA, the build path must be as follows:

\modem_proc\rftarget_dimepm\common\qcn\wtr1605_chile_svdsda\etc
MSM8916_WTR1605_CHILE_SVDSDA_MASTERFILE.

Table 10-5 RF configuration

RFC Name	RF hardware ID (NV# 1878)	Comments	Reference QCN
RFC_WTR1605_CHILE_RF360	72	An initial bringup card with HDET on WTR being used	—
RFC_WTR1605_CHILE_RP1	62	QRD Reference Platform - Single WTR solution with CSFB design support	MSM8916_CHILE_RP1_Reference_QCN.qcn
RFC_WTR1605_CHILE_RP2	89	QRD Reference Platform - Two WTR solution with SVLTE support	MSM8916_CHILE_RP2_APT_Reference_QCN.qcn
RFC_WTR1605_CHILE_RF360_TDET	73	An APT-only configuration for internal platform	MSM8916_CHILE_TDET_Reference_QCN.qcn
RFC_WTR1605_CHILE_SVDSDA	90	An EPT configuration, which is a superset of SVLTE/SGLTE/DSDA	MSM8916_CHILE_SVDSDA_Reference_QCN.qcn
RFC_WTR1605_CHILE_SGLTE	91	An initial SGLTE bringup card, which is now deprecated.	MSM8916_CHILE_SGLTE_Reference_QCN.qcn
RFC_WTR1605_CHILE_SXSDA_V2	92	An EPT configuration with QFE2101	MSM8916_CHILE_SxSDA_V2_Reference_QCN.qcn

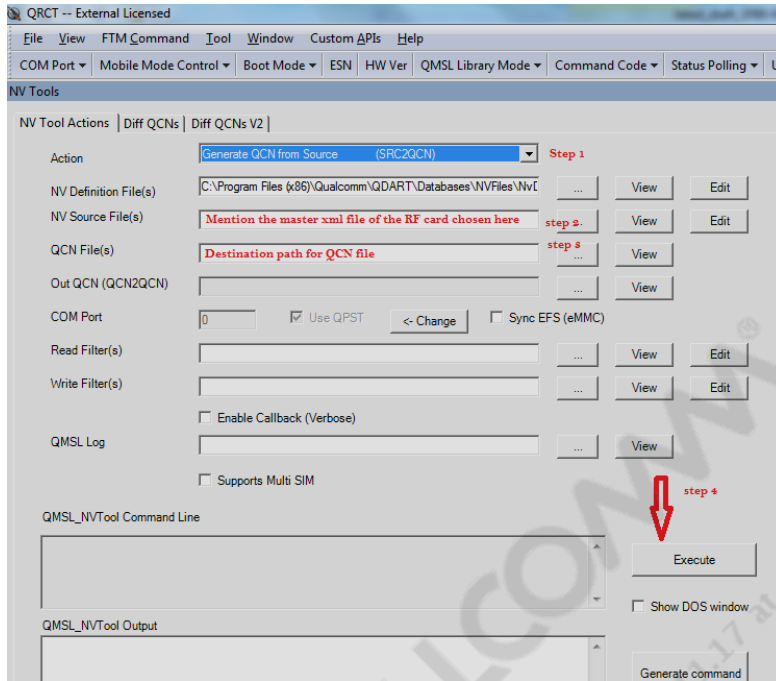


Figure 10-6 QRCT NV tool for generating QCN

NOTE: The MPSS build has a reference QCN as a deliverable. The QCN contains RF-only NV items and does not have XO TRIM NV item in it. Hence, for basic functional testing, this QCN is loaded followed by XO-only calibration. The reference QCNs are found at
 \modem_proc\rftarget_dimepm\common

8.4.1.2 NV configuration for WCDMA/GSM + GSM

Refer to [Table 10-6](#) for WCDMA/GSM + GSM DSDS NV configuration.

Table 10-6 WCDMA/GSM + GSM DSDS NV settings

NV item	Subscription1 (WCDMA/GSM) value	Subscription2 (GSM) value
00010	17: Auto (WCDMA or GSM) 14: WCDMA only	13: GSM only
00441	0xFFFF (0x380 or 0x387 for specific bands)	Same as SUB1
00850	0x02 CS and PS	0x00 CS only
00855	0 RTRE configuration for WCDMA/GSM + GSM (Before setting this item send spc 000000 through QXDM command window)	Same as SUB1
00880	0x01 integrity enable	Same as SUB1
00881	0x01 ciphering enable	Same as SUB1
00882	0 fake security enable	Same as SUB1
00905	0x0000 fatal error option	Same as SUB1
00946	0x0040 IMT band(0CE0 US PCS Band)	0x0040

NV item	Subscription1 (WCDMA/GSM) value	Subscription2 (GSM) value
03649(RRC Version)	Inactive or 0→R99; 1→R5; 2→R6; 3→R7; 4→R8	Same as SUB1
03851	0 RxD control	Same as SUB1
04118 (HSDPA Cat)	Inactive or 24→DC	Same as SUB1
04210 (HSUPA Cat)	Inactive or 6→EUL 2 ms; 5→EUL 10 ms	Same as SUB1
04398	0→DSDS; 1→SS	Same as SUB1
04399	1 detect hardware reset	Same as SUB1
06876	00005→WCDMA/GSM to GSM Tune away 00006→DSDS	Same as SUB1
06907	1 Dual SIM hardware 0 Single SIM hardware	Same as SUB1

For WCDMA/GSM + GSM DSDA NV configuration, the same as [Table 10-6](#), except NV70266 (Dual standby preference) must be set to 2.

8.4.1.3 NV configuration for CDMA + GSM

Refer to [Table 10-7](#) for CDMA + GSM DSDS NV configuration

Table 10-7 CDMA + GSM DSDS NV Settings

NV item	Subscription1 (CDMA+ HDR) value	Subscription2 (GSM) value
10 (Mode Preference)	4->: Automatic mode 19: CDMA and HDR only	13: GSM only
475 (HDR SCP Session Status)	0-> Inactive	Same as SUB1
850 (Service Domine Preference)	0x02 CS and PS	0x00 CS only
905 (Fatal Error Option)	0x0000 fatal error option	Same as SUB1
4204 (HDR SCP Force	0-> HDR SCP Force Release 0 Session Configuration	Same as SUB1
4964 (HDR SCP Force At Configuration)	0-> HDR rev 0, 1->HDR rev A, 3->HDR rev B.	Same as SUB1
03446 (TRM Configuration)	2, 0	Same as SUB1
4398 (UIM Select Default USIM Application	0→DSDS; 1-> SS	Same as SUB1
4399 (detect hardware reset)	1 detect hardware reset	Same as SUB1
6874 (ASID 1 Data)	255	Same as SUB1
6875 (ASID 2 Data)	255	Same as SUB1
562 (preference Hybrid Mode)	1 -> Hybrid operation allowed	0 -> Hybrid operation not allowed
6876(Dual standby config Items	00002 (Dual standby preference)	Same as SUB1
6907 (NV_UIM_HW_SIM_CONFIG)	1-> Dual SIM 0-> Single SIM	Same as SUB1

NV item	Subscription1 (CDMA+ HDR) value	Subscription2 (GSM) value
855 (RTRE configuration)	0 (Before setting this item send spc 000000 through QXDM command window)	Same as SUB1

For CDMA + GSM DSDA NV configuration, the same as [Table 10-7](#), except NV70266 (Dual standby preference) must be set to 2.

8.4.1.4 LTE + GSM DSDS settings

To make sure that the device is correctly configured for LTE + GSM DSDS mode of operation, make the following changes.

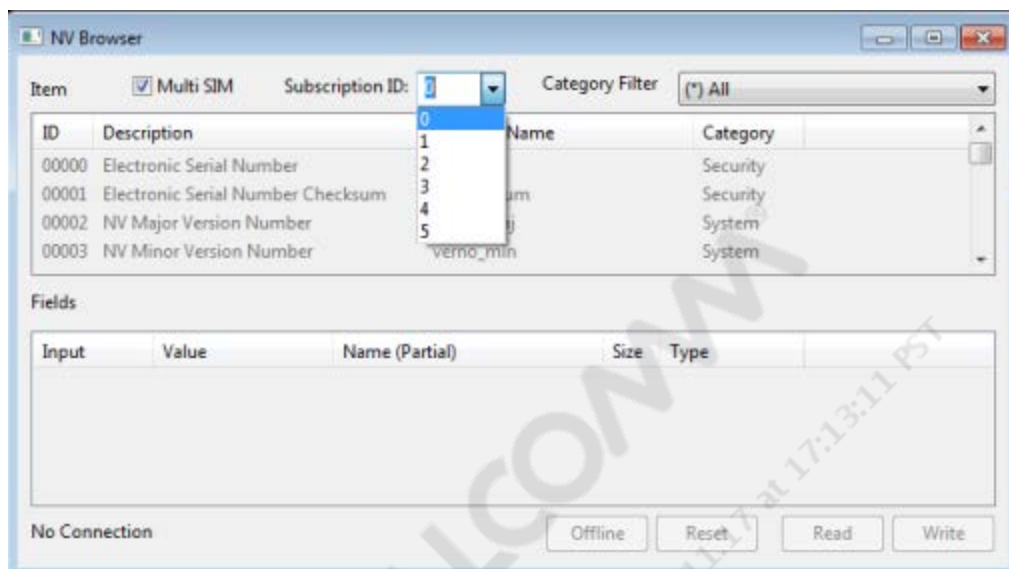
VoLTE/IMS/eMBMS and CSG features are not supported with LTE + GSM DSDS. This section explains all the needed NV items for disabling VoLTE/IMS/eMBMS as well as other parameters for LTE + GSM mode of DSDS operation.

Table 10-8 LTE + GSM DSDS settings

Feature Name	How to disable
VoLTE/IMS	Refer to section 3.6 of the document <i>Application Note: SGLTE Device Configuration</i> (80-NJ017-11).
eMBMS	EFS file embms_feature_status (with value set as 0) must be copied under the path /nv/item_files/modem/lte/rrc
CSG	Create an EFS file viz. csg_control at the path "/nv/item_files/modem/lte/rrc/csg". A 16-byte hexadecimal value is written into it 01 01 01 00 01 00 00 00 E0 93 04 00 00 00 00 00
Carrier Aggregation	It is disabled by default in DSDS mode

Enable DSDS NV settings

1. Set the following NVs in QXDM NV browser



Subscription 0	
NV Item number	NV Item value
00010	31 (GWL)
00850	0x01
65777	1
70210	hw_config.UIM[1].DISABLE_UIM Set to FALSE
06876	5
06907	1
04398	0

2. Perform “Spc 000000” in QXDM and then set the following:

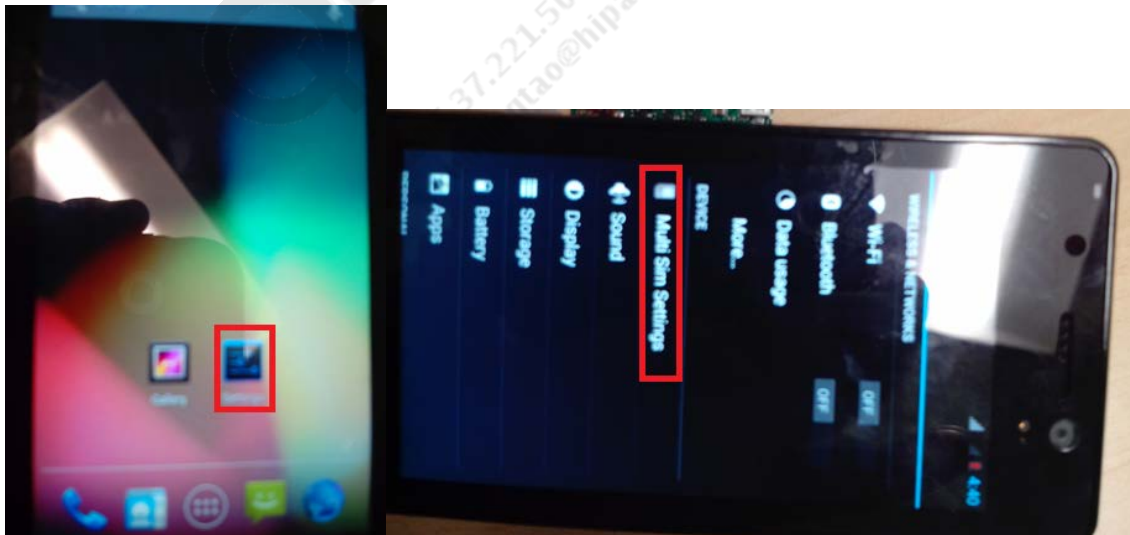
NV Item number	NV Item value
00855	0 (for both Single SIM and Dual SIM)
70266	1 (for Dual SIM)

Subscription 1	
NV Item number	NV Item value
00010	13 (GSM Only)
00850	0x00
65777	0

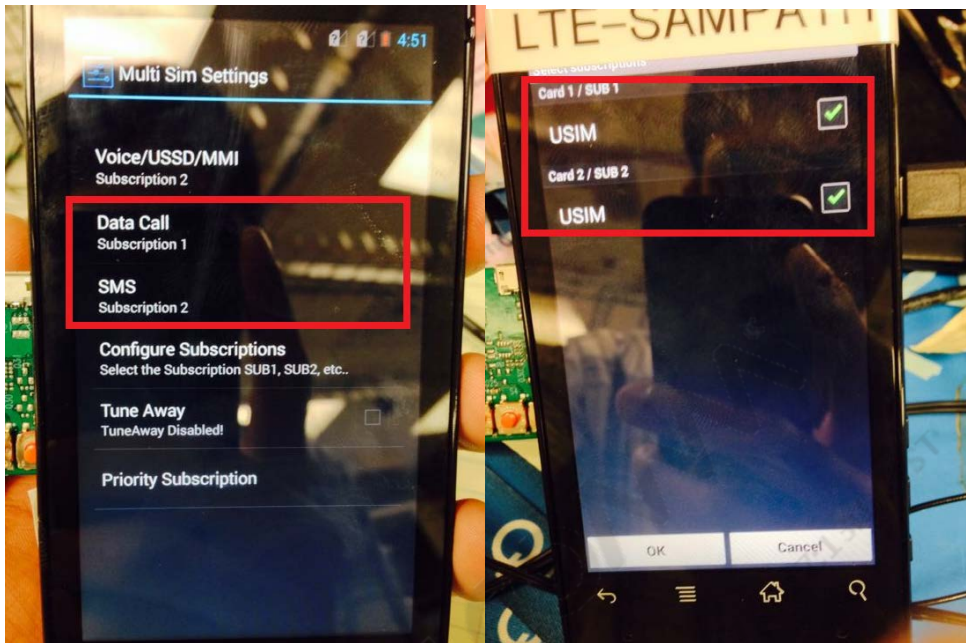
3. For NV settings for IRAT scenario – NV settings are required only on Subscription 0.

Subscription 0	
NV Item number	NV Item value
00010	34 (G+L)
00850	0x02
65777	0
00946	1F
02954	0

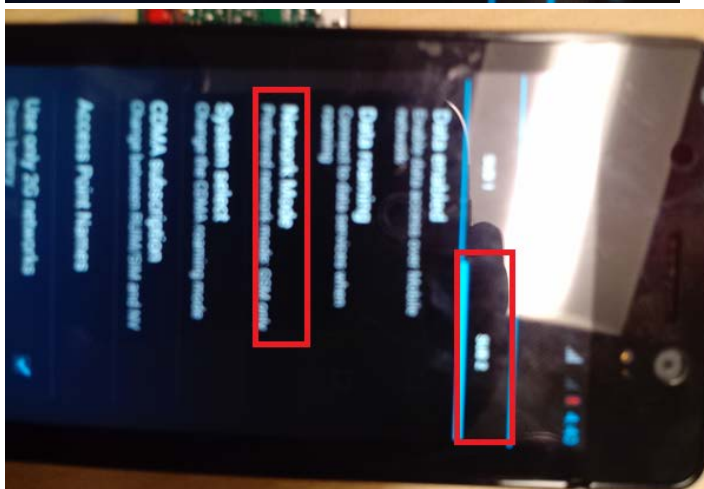
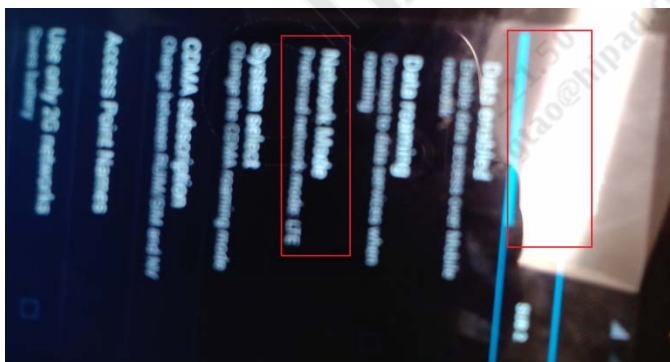
4. Restart the UE.
5. Run the dual SIM commands from ADB shell
 5. adb devices
 6. adb root
 7. adb shell
 8. setprop persist.radio.multisim.config dsds
 9. getprop persist.multisim.config (it must show up as DSDS).
6. Restart the UE.
7. Perform the below setting from UI.
8. Go to Settings on the UI.

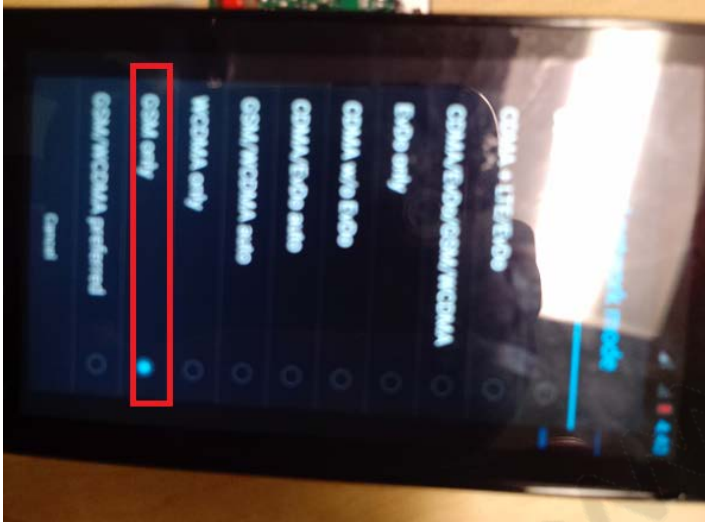


9. The user must be able to see Multi-SIM settings, select it.
10. Select Configure Subscriptions and configure the UIM cards to respective Subs.



11. Go back to settings, select Mobile Networks, select subscriptions set the respective subs to LTE and GSM.





12. You can run the below cmd after power-up, to check if both RIL activated for DSDS.

```
C:\>adb shell ps rild
```

USER	PID	PPID	VSIZE	RSS	WCHAN	PC	NAME
radio	243	1	17968	2900	ffffffff	00000000	S

/system/bin/rild

USER	PID	PPID	VSIZE	RSS	WCHAN	PC	NAME
radio	247	1	17952	2932	ffffffff	00000000	S

/system/bin/rild

13. If below cmd returns only one radio as output then the RIL still is in Single SIM mode.

```
C:\>adb shell ps rild
```

USER	PID	PPID	VSIZE	RSS	WCHAN	PC	NAME
radio	166	1	16928	2828	ffffffff	00000000	S

/system/bin/rild

8.4.1.5 1xSRLTE settings

The following settings need to be done to configure the device in 1xSRLTE mode of operation.

NV Item	Value	Description
72539	1	ESR support/CSFB support for dual Rx UEs 1 – Supported 2 – Not supported
72550	500 (ms)	LTE NAS 1xSRLTE ESR delay timer

8.4.2 Segment loading configuration

As part of memory optimizations, there is segment loading concept assuming that there will not be any use case where the network supports both the modes (WCDMA and TD-SCDMA) at one place. The Segment loading feature is controlled by FEATURE_SEGMENT_LOADING and when enabled in MPPS build and based on the UE supported modes, it supports two segments i.e. (LTE/GSM/WCDMA/CDMA) or (LTE/GSM/TD-SCDMA/CDMA). The segment loading works in two modes i.e. Automatic or Manual and by default it is set to Manual mode.

Manual mode

For manual mode:

1. On a fresh load of the build, the UE is by default configured for Manual Mode of Segment Loading operation.
2. NV72542 – 2 (default) → The UE works in WCDMA Segment mode.
3. NV72542 – 1 → The UE works in TD-SCDMA Segment mode.
4. In Manual mode, there are no switches between WCDMA and TD-SCDMA segments, as the Manual mode means that the UE is set to operate as determined by the NV72542 setting.

Changing the NV72542 value between 1 and 2, the UE can be set to operate with WCDMA segment or TD-SCDMA segment.

At power-up first time and based on the NV 72542 setting. The UE loads the said NV image initially and start searching the network as per the configured RAT priority order. If UE failed to get the coverage on the network in Manual mode, the user must change the NV72542 explicitly to search for the other networks. To avoid the user intervention for NV switch and finally network search, the user can set the segment loading mode to Automatic.

Automatic Mode

1. To take the UE out of Manual Mode and to put it in Automatic mode, load the segment_loadign.xml file as below
 - a. Copy from <build_root>\modem_proc\mmcp\policyman\configurations\SegLoad\segment_loading.xml to \.\policyman\ on the device.
 - b. Change NV 10 to “auto”.
2. While in Automatic mode, UE loads the image based on the NV 72542 setting
 - a. Segment switching occurs as per the logic outlined in the segment_loading.xml file.
 - b. While in service on MCCs is listed in segment_loading.xml file, TDS segment is loaded.
 - c. While in service on MCCs not listed in segment_loading.xml file, WCDMA segment is loaded.
 - d. While in complete OOS, segment switching occurs as determined by the timers listed in segment_loading.xml.
 - e. Logic/comments in “<build_root>\modem_proc\mmcp\policyman\configurations\SegLoad\segment_loading.xml” file offer detailed steps.
 - f. Segment switching occurs with modem subsystem reset.

3. Segment loading segment_loading.xml file works independent of other policy manager configuration files.

NOTE: Do not set NV 72542 to 0. It causes device runtime crash from heap exhaustion

8.4.3 Call configuration

8.4.3.1 1X voice call

Prerequisites

- NV setting (set NV # 10 to 4 for Automatic mode)
- QCN (be sure to perform RF calibration on the device; do not use a golden QCN..., also note that 1X is on the SV chain by default)
- PRL (must match the band/channel being tested, and SID & NID must also match)

Callbox setup

1. Current testing has been done on Agilent 8960
2. Callbox setup instructions
 - a. In System Config/Application Setup, select CDMA 2000 Lab App B (version should be B or above);
 - b. In Call Setup/Call Control screen
 - i. Set the Operating Mode to Active Cell;
 - ii. Set the System Type to IS-2000;
 - iii. Click “**More**” to move to page 2 of 5, select Cell Info/Cell Parameters. Configure the SID/NID(match with PRL, or set it as wildcard: SID=0, NID=65535);
 - c. In Call Setup/Call Params screen
 - i. Set Cell 1 Power to a proper value (between -45 ~ -65 dBm);
 - ii. Set Cell Band and Channel, match the PRL setting;
 - iii. Set Protocol Rev to 6(IS-2000-0);
 - iv. Set Radio Config(RC) to (3,3) - (Fwd3, Rvs3);
 - v. Set FCH Service Option setup for (Fwd3, Rvs3) to SO3 (Voice);

Device setup

To make the call:

1. Use the Call Manager screen in QXDM Professional (QXDM Pro).
2. Set the phone number to something like 1234 and make sure that the service option matches the callbox setting.
3. Click **Call** to start a call. For MT, originate the call from the test box.

NOTE: For Dual SIM device, the default subscription is 0. If subscription 1 is needed, check the **Dual SIM** and then select **Subscription ID**.

Call Manager

Options ☒ Dual SIM Subscription ID: 0

Technology CDMA

Phone Number 1234

Service Type 4 - Automatic

Call Type/Option 3 - Voice (EVRC)

Call Setup Timer (s) 20 ☐ Use Key Emulation

Conversation Timer (s) 10 ☐ Infinite Call

Teardown Timer (s) 20 ☐ Redial Upon Failure

Pause Timer (s) 3 ☐ Require Channel

Calls To Attempt 0 ☐ Attempt MC Support

Required Channel 0

Start

Results

Status	0 Call(s) Attempted
Phone State	-
Phone Link Failures	0
Dropped Calls	0
Call Failures	0
Channel Failures	0

8.4.3.2 1X data call

Prerequisites

- NV setting (set NV #10 to 4 for Automatic mode)
- QCN (be sure to perform RF calibration on the device; do not use a “golden” QCN...; also note that 1X is on the SV chain by default)
- PRL (must match the band/channel being tested, and SID & NID must also match)

Callbox setup

1. Current testing has been done on Agilent 8960 or Anritsu MT8820;
2. In Call Setup/Call Params screen
 - a. Set Radio Config(RC) to (3,3) - (Fwd3, Rvs3);
 - b. Set FCH Service Option Setup for (Fwd3, Rvs3) to SO32 (TDSO);
3. Callbox setup instructions – Make sure band/channel and SID/NID match those specified in PRL; cell power must be set somewhere between -45 and -65 dB

Device setup

To make the call:

1. Use the Call Manager screen in QDXM Pro.
2. Set the phone number to something like 1234 and make sure that the service option matches the callbox setting.
3. Click Call to start a call; for MT, originate the call from the test box.

8.4.3.3 HDR call

NV settings

1. NV setting (set NV # 10 to 4 for Automatic mode)
2. For DO Rev A calls, NV #4964 must be set to Rev A mode and the callbox has to be put in Rev A mode. For example, Set NV #4964 = NV_HDRSCP_REVA_PROTOCOLS_WITH_MFPA.

RF calibration

1. Ensure that the device has been RF-calibrated
2. Roaming List – A preferred roaming list with HDR channels must be loaded and subnet ID in PRL must match the callbox setting
3. Roaming list is loaded via QPST Service Programming; do the following:
 - a. Select the **Roam** tab
 - a. Enter PRL path in the Preferred Roaming area
 - b. Select **Write to Phone**

Callbox setup

For DO Rev A calls, place the callbox in Rev A mode.

Device setup

To make a call:

1. Power cycle the device
2. Enter mode online in the QXDM Pro Command Bar; the device must attempt to acquire HDR channel and negotiate a session

8.4.3.4 GSM voice call

Prerequisites

- NV settings
 - a. Set NV #10(Mode Preference) to 13 for GSM only operation
 - c. Set NV #441(Band Class Preference) to 0x200 for GSM900 band
 - d. For Multimode build, the Mode preference is changed via the UI. Otherwise Multimode uses the default setting in Android (defaults to 1X only) and it uses this to overwrite the NV item, so the modem does not go into GSM.
- QCN (be sure to perform RF calibration on the device; do not use a golden QCN)

Callbox setup

1. Current testing is performed on Agilent 8960 or Anritsu MT8820
2. Set band to GSM900; cell power must be set somewhere between -45 and -65 dBm
3. Set Channel mode to TCH/F (full rate TCH)

Device setup

To make a call:

1. Use the Call Manager screen in QDXM Pro.
2. Set the phone number, e.g., to 1234, and make sure that the service option matches the callbox setting.
3. Click Call to start a call; for MT, originate the call from the test box.

8.4.3.5 GPRS data call

RF calibration

Ensure the device is RF-calibrated

Callbox setup (Agilent 8960)

1. Select call setup screen
2. Callbox setup
 - a. BCCH parameters→cell power = -75 dBm
 - b. BCCH parameters→cell band = EGSM
 - c. BCCH parameters→Broadcast chan = 20
 - d. PDTCH parameters→Multislot config = 1 Down 1 Up
 - e. Operating mode = Active mode GPRS
 - f. Data Conn = Type ETSI Type A

Device setup

To make a call:

1. Insert a SIM (test SIM is good enough)
2. Power up the device; Enter mode online in QXDM Pro Command Bar if necessary
3. UE camps on GPRS cell and ATTACH
4. Initiate test mode A data call; hit Start Data Connection; bottom of screen must display TRANSFERRING

8.4.3.6 WCDMA voice call

Prerequisites

QCN – Ensure device is calibrated

Callbox setup

1. Agilent 8960
 - a. Cal box setup
 - i. Call Control/Security Info/Security Parameters/Security Operations – None
 - ii. Call Params/Cell Power – -50.00 dBm
 - iii. Call Params/Channel Type – 12.2 KRMC
 - iv. Call Params/Paging Service – AMR Voice
 - b. Test results – MO and MT passed 5/5
2. Anritsu 8480C

Callbox setup

 - i. Station Globals – Spec_Release – 3; HSDPA – FALSE; EUL – FALSE
3. Anritsu 8820

Callbox setup

 - i. Call Processing – On
 - ii. Test Loop Mode – Off
 - iii. Signal/Channel Coding – Voice

Device setup

To make a call (MO):

1. After starting firmware and software, attach USB and then do a “mode online” from QXDM Pro Command Bar
2. Allow the mobile to acquire and register to network
3. Start a call using the Call Manager dialog in QXDM Pro
4. Set Technology to WCDMA

5. Set phone number to 1234
6. Check the **infinite call** box
7. Start the call
8. Phone state shows “Originating call” and then “Conversation”

8.4.3.7 WCDMA data call

Prerequisites

QCN – Ensure device is calibrated

Callbox

- Agilent 8960
 - a. Callbox setup
 - i. Call Params/Channel Type – HSPA
 - b. Test results – DUN data calls passed with ~384-kbps throughput; QMI not tried
- Anritsu 8480C
 - a. Callbox setup
 - i. Station Globals – Spec_Release – 5; HSDPA – TRUE; EUL – FALSE
 - b. Test results – DUN data calls was up but crashed later; QMI not tried
- Anritsu 8820
 - Callbox setup
 - i. Call processing – On
 - ii. Test Loop Mode – Mode 1
 - iii. Signal/Chanel Coding – Fixed Reference Channel

Device setup

To make a call:

1. Allow UE register to network
2. Click “**Connect/Dial**” on USB (DUN) or QMICM to initiate data call
3. Run iPerf or FTP applications to test throughput (we tested with FTP)

8.4.3.8 TD-SCDMA call

Prerequisites

- QCN – Ensure device is calibrated
- NV_PREF_MODE_1(10) – Set to 53 TD-SCDMA only
- Set NV 00850 = 0x2 (CS PS)
- TDS RRC integrity protection enabled (66011) – Set to 0; set to 1 for a live network
- TDS RRC ciphering enabled (66012) – Set to 0; set to 1 for a live network
- TDS RRC fake security status (66013) – Set to 0; set to 1 for a live network
- TDS RRC special frequency enabled (66014) – Set to 0
- TDS RRC PDCP disabled (66016) – Set to 1
- TDS RRC version (66017) – Set to 3
- TDS RRC HSDPA category (66020) – Set to 15
- TDS RRC NV version (66023) – Set to 2
- TDS RRC optional feature bitmasks (66024) – Set to 0x3F
- TDS RRC Special Test Setting Enabled (69731) – Set to 0; set to 1 when entering MTNet and CMCC tests

Callbox

- Agilent 8960
 - Callbox setup
 - i. Cell Power – -45.00 dBm
 - ii. Channel Type – 12.2K RMC SC
 - iii. Paging Service – AMR Voice
 - iv. Channel – Desired band/channel
 - v. Screen2 – Cell Info → Cell Parameters → PS Domain Information → Present
- Anritsu 8820
 - Callbox setup
 - i. Call processing – On
 - ii. Test Loop Mode – Mode 1
 - iii. Signal/Channel Coding – Fixed Reference Channel

Device setup

To make a call:

1. Configure the test equipment using the settings mentioned in Section 10.4.3.8 and check that the signal is acquired.
2. MO and MT Calls - Initiate the MO call from device and MT call from the test equipment.
3. Send/Receive SMS from/to device and test equipment.
4. DUN Call Using a Dial Up Connection “PS on USB” setup a DUN data Call.
5. Browse data from web browser to data session.

8.4.3.9 LTE data call

Prerequisites

- QCN – Ensure device is calibrated
- NV Settings - Set NV 00010 = 30 (LTE-only) and NV 00850 = 0x1 (PS-only) and NV 65777 = 0x1

Callbox setup

- Aeroflex

Cal box setup

- i. Under home screen, select Mode as Callbox Mode
- ii. Select Parameter Config Tab.Parameter Group → Select System → Band ID Ex: For TDD Testing:38 & For FDD Tsting:1
- iii. Under Parameter Group select Layer 3 → MCC 001 and MNC 01 based on your SIM configuration.

- Anritsu 8820

Callbox setup

- i. Call Processing – On
- ii. Test Loop Mode – Off
- iii. Signal/Channel Coding – Required LTE band/channel
- iv. Start Call

Device setup

To make a call:

1. Connect primary and secondary RFs to the respective ports.
2. Run mode LPM in QXDM.
3. Click Registry software ICON.
4. Click Application software ICON.
5. In Apps software, open TC and run.

6. Click Analyzer → Run. A message appears, “Please power on the phone” in application software.
7. Run mode online in QXDM.
8. The UE registers to the network with status in conversation state.
9. Verify browsing by opening a web page.

8.4.4 GPS configuration

Prerequisites

GNSS SubSysGNSS DLL Ver 1.0.44 or higher is required to perform offline RF dev.

Operation procedures

Running offline RF Dev requires QPSR, for more details, see *gpsOne Gen 8 Engine Family RF Development Test Procedures* (80-VM522-2).

8.4.5 Multimedia configuration

This section describes the bringup of multimedia features like audio, display, video, and camera. For further details, see *MSM8916 Linux Android Software Debug Manual Software Product Document* (SP80-NL239-5).

8.4.5.1 Audio configuration

Audio device debugging

Use amix command to enable and disable the devices. For example, to enabling Speaker Device for audio playback over I2S:

- Enable sequence:
 - a. `tinymix 'PRI_MI2S_RX Audio Mixer MultiMedia1' 1`
`tinymix 'RX3 MIX1 INP1' 'RX1'`
 - b. `tinymix 'SPK DAC Switch' 1`
- Disable sequence:
 - c. `tinymix 'PRI_MI2S_RX Audio Mixer MultiMedia1' 0`
`tinymix 'RX3 MIX1 INP1' 'ZERO'`
`tinymix 'SPK DAC Switch' 0`

Audio playback

This section describes the procedure to verify audio playback.

To verify audio playback, open the ADB shell, and try the following tinymix and tinyplay commands to start the playback:

1. Start compress playback
 - a. Enable RX Codec Path (speaker device).

- b. `tinymix 'RX3 MIX1 INP1' 'RX1'`
 - c. `tinymix 'SPK DAC Switch' 1`
 - d. Enable DSP AFE for playback over MI2S.
 - e. `tinymix 'PRI_MI2S_RX Audio Mixer MultiMedia1' 1`
2. Play the PCM audio using the following `tinypplay` command:
- `tinypplay <filename.wav>`
3. Stop compress playback
- a. Stop compress Playback (Ctrl +C).
 - b. Disable RX Codec Path (speaker device).
 - c. `tinymix 'RX3 MIX1 INP1' 'ZERO'`
 - d. `tinymix 'SPK DAC Switch' 0`
 - e. Disable DSP AFE for compress Playback over I2S.
 - f. `tinymix 'PRI_MI2S_RX Audio Mixer MultiMedia1' 0`

Audio recording

This section describes the procedure to verify PCM capture headset MIC.

1. To verify PCM capture headset MIC:
- ```
#Enable capless switch for MICBIAS
tinymix 'MICBIAS1 CAPLESS Switch' 1
#Enable DSP AFE for Audio Recording over I2S
tinymix 'MultiMedia1 Mixer TERT_MI2S_TX' 1
Enable Codec TX Path
tinymix 'DEC1 MUX' 'ADC2'
tinymix 'ADC2 MUX' 'INP2'
```
2. Start the audio recording.
- a. `tinycap /data/rec.wav`
  - b. Disable the handset TX device (AMIC1).
- ```
tinymix 'MICBIAS1 CAPLESS Switch' 0
tinymix 'MultiMedia1 Mixer TERT_MI2S_TX' 0
tinymix 'DEC1 MUX' 'ZERO'
tinymix 'ADC2 MUX' 'ZERO'
```

Audio recording through Dual MIC

This section describes the procedure to verify recording through Dual MIC

1. To verify PCM, capture through Dual MIC:

```
#Enable DSP AFE for Audio Recording over I2S
tinymix 'MultiMedial Mixer TERT_MI2S_TX' 1
# Enable Codec TX Path
tinymix 'DEC1 MUX' 'ADC1'
tinymix 'DEC2 MUX' 'ADC2'
tinymix 'ADC2 MUX' 'INP3'
tinymix 'MI2S_TX Channels' 'Two'
```

2. Start the audio recording.

- a. `tinycap/data/adc12.wav -C 2 -R 44100 -T 20`

- b. Disable the handset Tx device (AMIC1).

```
tinymix 'MultiMedial Mixer TERT_MI2S_TX' 0
tinymix 'DEC1 MUX' 'ZERO'
tinymix 'DEC2 MUX' 'ZERO'
tinymix 'ADC2 MUX' 'ZERO'
tinymix 'MI2S_TX Channels' 'One'
```

Voice call

Enable QXDM log with voice log mask. Use QCAT's vocoder playback function to play vocoder packets and PCM separately.

Following are the voice quality issues:

1. For Rx path, if vocoder packet is OK but PCM is not OK, verify it from the voice tuning perspective.
2. For Tx path, if the PCM is OK, but the vocoder packet is not OK, verify it from voice tuning perspective.
3. For Rx path, if the vocoder packet is not OK, consider more from the network perspective.
4. For Tx path, if the vocoder packet is OK, but the far end still hears noise, consider from network perspective.

Mute issues are more likely device issues. Verify if the device has codec widgets connected properly and if the gain is muted or not. Also, consider the voice quality issue based on the vocoder packets and PCM playback.

MBHC

This section describes the process to check if the headset insertion/removal and button press/release are detected by the codec driver using ADB Shell (solution#22064).

Headset jack and button jack are created in MBHC driver (wcd-mbhc-v2.c) by calling `snd_soc_jack_new()`. Event number are found from the kernel logs; look for "Button Jack" and "Headset Jack".

Input – msm8x16-snd-card Button Jack as /devices/platform/soc-audio.0/sound/card0/inputX

Input – msm8x16-snd-card Headset Jack as /devices/platform/soc-audio.0/sound/card0/inputY

X – Headset button jack input event number

Y – Headset jack input event number

- To detect headset insertion and removal:

1. Connect the target device to PC using a USB port.
 - a. Open the adb shell window and run the following adb command:


```
#adb shell getevent /dev/input/eventY
```

Y – Headset jack input device event number

The following messages are printed in the adb shell window, if the headset is inserted:

```
/dev/input/eventY: 0005 0002 00000001
/dev/input/eventY: 0005 0004 00000001 (this line is only present if headset
has a microphone)
/dev/input/eventY: 0000 0000 00000000
```

The following messages print in the adb shell window, if the headset is removed:

```
/dev/input/eventY: 0005 0002 00000000
/dev/input/eventY: 0005 0004 00000000 (this line present only if headset
has a microphone)
/dev/input/eventY: 0000 0000 00000000
```

Headset insertion or removal event are found in the kernel logs. In the kernel logs, look for "Reporting insertion" or "Reporting removal".

1. "Reporting insertion 1" is for headphone without a microphone
2. "Reporting insertion 3" is for headset with microphone

To detect headset button press/release detection:

2. Connect the target device to a PC using a USB port.

10. Open the adb shell window and run the following adb command:

```
#adb shell getevent /dev/input/eventX
```

X – Headset button jack input device event number

The following messages are printed in the adb shell window, if the headset button is pressed:

```
/dev/input/eventX: 0001 0001 00000001
```

```
/dev/input/eventX: 0000 0000 00000000
```

The following messages are printed in the adb shell window, if the headset button released.

```
/dev/input/eventX: 0001 0001 00000000
```

```
/dev/input/eventX: 0000 0000 00000000
```

Headset button press or release event are found in the kernel logs. In the kernel logs, look for "Button pressed" or "Button released".

Enabling kernel MBHC debugging logs

Kernel MBHC debugging logs are enabled by dynamically executing the following adb command:

```
#mount -t debugfs debugfs /sys/kernel/debug
#echo -n "file msm8x16-wcd.c +p" > /sys/kernel/debug/dynamic_debug/control
#echo -n "file msm8x16.c +p" > /sys/kernel/debug/dynamic_debug/control
#echo -n "file wcd-mbhc-v2.c +p" > /sys/kernel/debug/dynamic_debug/control
```

FM

This section describes how to use the AMIX control of FM playback. The tinymix control command listed below sets the audio routing path and enables the codec for FM playback use case. However, for FM tuner, it has to turn on FM tuner and tune the channels before switching the device.

To use the AMIX control of FM playback:

1. Enable codec Rx path.

```
tinymix 'RX1 MIX1 INP1' 'RX1'
tinymix 'RX2 MIX1 INP1' 'RX2'
tinymix 'HPL DAC Switch' 1
```

2. Enable the DSP Loopback between INTERNAL_FM_TX and I2S.

```
tinymix 'SEC_MI2S_RX Port Mixer INTERNAL_FM_TX' 1
```

3. Connect backend and frontend DAI.

```
tinymix ' MI2S_DL_HL Switch' 1
```

4. Start the hostless playback and recording to enable the internal codec Rx and Tx devices (msm8x16-wcd)

```
tinycap -C 2 -D hw:0,6 -P (run it in one console terminal)
```

```
tinyplay -D hw:0,5 -P (run it in the other console terminal)
```

External I2S interface bringup

Details of external I2S bringup are provided in the document *MSM8916 External I2S interface* (80-NL239-42).

8.4.5.2 Display

For display panel bringup and driver porting-related information, see *DSI Programing Guide for B-Family Android Devices* (80-NA157-174).

It describes application usage of the Display Serial Interface (DSI) panel bringup for the Android OS. It also provides sample code and PLL calculation pertaining to the DSI Mobile Industry Processor Interface (MIPI) panel bringup. For details on Linux Android display driver porting, see *Linux Android Display Driver Porting Guide* (80-NN766-1).

8.4.5.3 Camera

Android default camera application is used to verify camera features. For details related to sensor driver porting and migration, refer to *Application Note: Sensor Module Bringup* (80-NL239-32).

For techniques on debugging different kind of camera errors, refer to *Linux Camera Debugging Guide* (80-NL239-33).

Blink detection

This section describes how to verify a blink detection.

Steps to enable:

1. Turn on the Face Detection option from the camera application UI.
2. Introduce a Human Subject (HS) in the camera FOV with the following conditions:
 - a. The HS must look directly at the camera.
 - b. Blink either of the two eyes

The Face and the eye blink get detected appropriately.

Gaze detection

This section describes how to verify a gaze detection.

Steps to enable:

1. Turn On the Face Detection option from the camera application UI
2. Introduce a Human Subject (HS) in the camera FOV with the following conditions
 - a. The HS must look directly at the camera
 - b. Gaze to the Left/right

The Face and the direction of the gaze (left/right) gets detected appropriately.

Smile detection

This section describes how to verify a smile detection.

Steps to enable:

1. Turn On the Face Detection option from the camera application UI
2. Introduce a Human Subject (HS) in the camera FOV with the following conditions
 - a. The HS must look directly at the camera
 - b. With a smile

The Face and the smile get detected appropriately.

Video HDR

NOTE: This section was added to this document revision.

Video HDR option gets populated in the camcorder UI only if the sensor has support for it; otherwise, the option is not populated in the UI. OEM needs to select this option in camera application to enable/verify this feature.

8.4.6 Video

NOTE: Numerous changes were made in this section.

Android default video player application is used to verify the playback of various video formats.

Widevine/CP playback

For successful Widevine playback, keybox must be installed correctly. Refer to *Application Note: Widevine DRM (80-N9340-1)* application document for keybox installation procedure.

Widevine demo application is used for successful secure content playback.

WFD + HDCP + WV playback

OEM needs to get HDCP libraries from third-party vendor and provision the device for playing secure content over HDCP Wi-Fi display session. WFD client application can be used to establish successful WFD session.

8.4.7 WCNSS configuration

WCNSS functionality does not require any specific configuration as everything is built in. Basic functionality is verified using either FTM tool or GUI (default Android settings application). For FTM tool usage, see *QRCT User Guide For WCN36X0 WLAN/BT/FM In FTM Mode* (80-WL300-27).

8.4.8 Subsystem Restart (SSR)

Subsystem Restart is a feature designed to give a seamless end-user experience when restarting after a system malfunction. The SoC is considered to be divided into individual subsystems (for example, Modem, WCNSS, etc.), and a central root, the Applications Processor (AP). The clients of these individual subsystems that are running on the AP receive notification from the kernel about a particular subsystem shutting down. The clients must be able to handle this notification in a graceful manner. The clients can expect to receive another notification when the subsystems are back up. Examples of such clients are EFS sync, remote storage, etc.

The use case for this feature is a catastrophic restart, i.e., a software malfunction on the modem, or any other subsystem that could cause the phone to be dysfunctional. In this instance, the AP is expected to restart the respective subsystems, to restore them to normal operation.

The core restart module, powers up/down registered subsystems when they crash and sends appropriate notifications.

Compile options to enable SSR – CONFIG_MSM_SUBSYSTEM_RESTART

Following are some useful adb commands for SSR:

- To find a specific subsystem

```
cat /sys/bus/msm_subsys/devices/subsysX/name (here X = 0,1,2 for different subsystems
```

```
modem, wcnss etc.)
```

- To know if SSR is enabled on a specific subsystem –

```
cat /sys/bus/msm_subsys/devices/subsysX/restart_level
```

- Enable SSR for various subsystems –

```
echo related > /sys/bus/msm_subsys/devices/subsysX/restart_level
```

- Disable SSR for various subsystems-

```
echo system > /sys/bus/msm_subsys/devices/subsysX/restart_level
```

For further information on SSR, refer Q34, Q35, and Q36.

A Android Device Tree Structure

The Android device tree structure, for example, the <Android device tree root>, is laid out as follows:

- build/ – Build environment setup and makefiles
- bionic/ – Android C library
- dalvik/ – Android JVM
- kernel/ – Linux kernel
- framework/ – Android platform layer (system libraries and Java components)
- system/ – Android system (utilities and libraries, fastboot, logcat, liblog)
- external/ – Non-Android-specific Open Source projects required for Android
- prebuilt/ – Precompiled binaries for building Android, e.g., cross-compilers
- packages/ – Standard Android Java applications and components
- development/ – Android reference applications and tools for developers
- hardware/ – HAL (audio, sensors) and Qualcomm specific hardware wrappers
- vendor/qcom/ – Qualcomm target definitions, e.g., msm7201a_surf
- vendor/qcom-proprietary/ – Qualcomm-proprietary components, for example, MM, QCRIL, etc.
- out/ – Built files created by user
 - out/host/ – Host executables created by the Android build
 - out/target/product/<product> – Target files
 - appsboot*.mbn – Applications boot loader
 - boot.img – Android boot image (Linux kernel + root FS)
 - system.img – Android components (/system)
 - userdata.img – Android development applications and database
 - root/ – Root FS directory, which compiles into ramdisk.img and merged into boot.img
 - system/ – System FS directory, which compiles into system.img
 - obj/ – Intermediate object files
 - include/ – Compiled include files from components
 - lib/
 - STATIC_LIBRARIES/
 - SHARED_LIBRARIES/
 - EXECUTABLES/
 - APPS/
 - symbols/ – Symbols for all target binaries

A.1 Android target tree structure

The Android target tree structure is laid out as follows:

- / – Root directory (ramdisk.img, read-only)
 - a. init.rc – Initialization config files (device config, service startups) init.qcom.rc

- b. `dev/` – Device nodes
- c. `proc/` – Process information
- d. `sys/` – System/kernel configuration
- e. `sbin/` – System startup binaries (ADB daemon; read-only)
- f. `system/` – From `system.img` (read-write)
 - i. `bin/` – Android system binaries
 - ii. `lib/` – Android system libraries
 - iii. `xbin/` – Nonessential binaries
 - iv. `framework/` – Android framework components (Java)
 - v. `app/` – Android applications (Java)
 - vi. `etc/` – Android configuration files
- g. `sdcard/` – Mount point for SD card
- h. `data/` – From `userdata.img` (read-write)
 - i. `app/` – User installed Android applications
 - ii. `tombstones/` – Android crash logs

A.2 Building Tiny Android

Tiny Android, or `TINY_ANDROID`, is a build variant that creates only a superminimal build configuration used for board bringup and low-level debugging. The `TINY_ANDROID` configuration consists of only the Android Linux kernel and a system root file system containing a minimal set of system utilities.

To build Tiny Android, use:

```
$ make BUILD_TINY_ANDROID=true -j4
```

A.3 Building the Linux kernel manually

- Set up the Android build environment (`envsetup.sh/choosecombo`).

1. Change to the kernel directory (`kernel/`).
2. Set up the correct kernel config with the command:


```
make ARCH=arm CROSS_COMPILE=arm-eabi- msm8974_defconfig
```
3. Build the kernel image with the command:


```
make -j3 ARCH=arm CROSS_COMPILE=arm-eabi- zImage
```
4. If desired, build the optional kernel modules with the command:


```
make -j3 ARCH=arm CROSS_COMPILE=arm-eabi- modules
```

The resulting kernel image appears in `kernel/arch/arm/boot/zImage`.

NOTE: In theory, it is possible to use `-jn` as long as ‘n’ is smaller than the number of processors in the server where the build is being created.

5. To start with a clean tree, use the following commands:

- a. To remove object files:
`make clean`
- b. To remove all generated files:
`make distclean`

A.4 Building Android manually

- Set up the Android build environment (envsetup.sh/choosecombo).
 1. Change to the main Android directory.
 2. Build with the command:
`make -j4`
 3. To build individual components, choose one of the following options:
 - To run make from the top of the tree, use the command:
`m <component name> # E.g. m libril-qc-1`
 - To build all of the modules in the current directory, change to the component directory and use the command:
`Mm`
 4. To delete individual component object files, choose one of the following options:
 - To delete a particular module, use the command:
`m clean-<module name>`
 - To delete a module within a given path, use the commands:
`rm -rf out/target/product/*/obj/STATIC_LIBRARIES/`
`<module name>_intermediates`
`rm -rf out/target/product/*/obj/SHARED_LIBRARIES/`
`<module name>_intermediates`
`rm -rf out/target/product/*/obj/EXECUTABLES/`
`<module name>_intermediates`

A.5 Other important Android build commands

Other important Android build commands are:

1. `printconfig` – Prints the current configuration as set by the choosecombo commands.
2. `m` – Runs make from the top of the tree. This is useful because the user can run make from within subdirectories. If you have the TOP environment variable set, the commands use it. If you do not have the TOP variable set, the commands look up the tree from the current directory, trying to find the top of the tree.
3. `- mm` – Builds all of the modules in the current directory.
4. `- mmm` – Builds all of the modules in the supplied directories.
5. `croot` – cd to the top of the tree.
6. `sgrep` – grep for the regex you provide in all .c, .cpp, .h, .java, and .xml files below the current directory.
7. `clean-$(LOCAL_MODULE)` and `clean-$(LOCAL_PACKAGE_NAME)`

- a. Let you selectively clean one target. For example, you can type `make clean-libutils`, and it deletes `libutils.so` and all of the intermediate files, or you can type `make clean-Home` and it cleans just the Home application.

8. `make clean` – Makes `clean` deletes of all of the output and intermediate files for this configuration. This is the same as `rm -rf out/<configuration>/`.

Android makefiles (`Android.mk`) have the following properties:

- 9. Similar to regular GNU makefiles; some differences are:
 - a. Predefined variables to assign for source files, include paths, compiler flags, library includes, etc.
 - b. Predefined action for compiling executables, shared libraries, static libraries, Android packages, using precompiled binaries, etc.

10. Variables

- a. `LOCAL_SRC_FILES` – List of all source files to include
- b. `LOCAL_MODULE` – Module name (used for “m”)
- c. `LOCAL_CFLAGS` – C compiler flags override
- d. `LOCAL_SHARED_LIBRARIES` – Shared libraries to include

11. Action

- `include $(CLEAR_VARS)` – Clears `LOCAL*` variables for the following sections:
 - i. `include $(BUILD_EXECUTABLE)`
 - ii. `include $(BUILD_SHARED_LIBRARIES)`
 - iii. `include $(BUILD_STATIC_LIBRARIES)`

NOTE: Paths in `Android.mk` are always relative to the Android device tree root directory.

To add a new module to the Android source tree, perform the following steps:

1. Create a directory to contain the new module source files and `Android.mk` file.
2. In the `Android.mk` file, define the `LOCAL_MODULE` variable with the name of the new module name to be generated from your `Android.mk`.

NOTE: For Applications modules, use `LOCAL_PACKAGE_NAME` instead.

Local path in your new module is `LOCAL_PATH`. This is the directory your `Android.mk` file is in. You can set it by inserting the following as the first line in your `Android.mk` file:

```
LOCAL_PATH := $(call my-dir).
LOCAL_SRC_FILES
```

The build system looks at `LOCAL_SRC_FILES` to find out which source files to compile, `.cpp`, `.c`, `.y`, `.l`, and/or `.java`. For `.lex` and `.yacc` files, the intermediate `.h` and `.c/.cpp` files are generated automatically. If the files are in a subdirectory of the one containing the `Android.mk` file, it is necessary to prefix them with the directory name:

```
LOCAL_SRC_FILES := \
file1.cpp \
```

1 dir/file2.cpp

2 The new module can be configured with the following:

- 3 ■ LOCAL_STATIC_LIBRARIES – These are the static libraries that you want to include in
4 your module.

5 LOCAL_STATIC_LIBRARIES := \
6 libutils \
7 libtinyxml

- 8 ■ LOCAL_MODULE_PATH – Instructs the build system to put the module somewhere other
9 than what is normal for its type. If you override this, make sure that you also set
10 LOCAL_UNSTRIPPED_PATH if it is an executable or a shared library so the unstripped
11 binary also has somewhere to go; otherwise, an error occurs.

9 Driver Porting Guide

9.1 CDT

9.1.1 Introduction

A CDT provides platform/device-dependent data, such as Platform ID, DDR hardware parameters, etc. Various modules can utilize this information to reduce dependency and perform dynamic initialization.

The CDT is programmed to the EEPROM device at the factory. If the target device does not have an EEPROM or the EEPROM has not been programmed, a default copy of the CDT will be linked into the build at compile time. The bootloader will fetch or link in the CDT, so the whole process will be transparent and seamless to other modules.

From a user perspective, the CDT will simply be a continuous byte array in the memory.

9.1.2 Boot procedure for loading CDT

1. sbl1 checks CDT partition which gets saved in boot partition of eMMC.
If yes, load CDT image. If no, perform step 2.
2. sbl1 load default cdt table (config_data_table[]) in sbl1.mbn.
3. sbl1 pass platform info to lk by SMEM
4. sbl1 - SMEM_HW_SW_BUILD_ID
5. lk - SMEM_BOARD_INFO_LOCATION
6. lk fetch platform info, load dt header, and search compatible dt entry
7. lk pass correct dt entry address to kernel

9.1.3 Key functions

- SBL1
- boot_update_config_data_table (boot_images\core\boot\secboot3\src\boot_config_emmc.c)
- LK
- dev_tree_get_entry_info (bootable\bootloader\lk\platform\msm_shared\dev_tree.c)

9.1.4 Key enumeration

- boot_images\core\api\systemdrivers\DDIChipInfo.h
- DALCHIPINFO_ID_MSM8326 = 205,
- DALCHIPINFO_ID_MSM8916 = 206,

```

1      DALCHIPINFO_ID_MSM8994    = 207,
2      ■ boot_images\core\api\systemdrivers\ PlatformInfoDefs.h
3      ...
4      DALPLATFORMINFO_TYPE_SURF    = 0x01, /**< Target is a SURF device. */
5      DALPLATFORMINFO_TYPE_CDP     = DALPLATFORMINFO_TYPE_SURF,
6      /**< Target is a CDP (aka SURF) device. */
7      DALPLATFORMINFO_TYPE_MTP_MSM = 0x08, /**< Target is a MSM MTP device.*/
8      DALPLATFORMINFO_TYPE_QRD     = 0x0B, /**< Target is a QRD device. */

```

9.1.5 DT header

```

10     kernel\arch\arm\boot\dts\qcom\msm8916-cdp.dts
11         model = "Qualcomm MSM 8916 CDP";
12         compatible = "qcom,msm8916-cdp", "qcom,msm8916", "qcom,cdp";
13         qcom,board-id = <1 0>;
14     kernel\arch\arm\boot\dts\qcom\msm8916-mtp.dts
15         model = "Qualcomm MSM 8916 MTP";
16         compatible = "qcom,msm8916-mtp", "qcom,msm8916", "qcom,mtp";
17         qcom,board-id = <8 0>;
18     kernel\arch\arm\boot\dts\qcom\msm8916-qrd.dts
19         model = "Qualcomm MSM 8916 QRD";
20         compatible = "qcom,msm8916-qrd", "qcom,msm8916", "qcom,qrd";
21         qcom,board-id = <11 0>;

```

For dt.img format, please refer to dtbtool.txt and
bootable\bootloader\lk\platform\msm_shared\smem.h.

9.1.6 CDT description XML

```

25     boot_images\core\boot\secboot3\scripts\jedec_lpddr2_single_channel.xml
26     <device id="cdb0">
27         <props name="platform_id" type="DALPROP_ATTR_TYPE_BYTE_SEQ">
28
29             0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, end
30
31         </props>
32     </device>

```

Example of 8916 device

CDP:

0x03, 0x01, 0x01, 0x00, 0x00, 0x00, end

MTP:

0x03, 0x08, 0x01, 0x00, 0x00, 0x00, end

QRD:

0x03, 0x0B, 0x01, 0x00, 0x00, 0x00, end

Byte 0: platform_id Version

Byte 1: platform_id ID:

0x01: Target is a SURF device

0x08: Target is a MTP device

1 0x0B: Target is a QRD device
2 Byte 2: platform_id Hardware version
3

4 **Requirements**

5 Generate CDT image and boot_cdt_array.c

6 cd boot_images\core\boot\secboot3\scripts

7 modify <jedec_lpddr2_single_channel.xml>

8 python cdt_generator.py jedec_lpddr2_single_channel.xml <cdt image>.bin

9
10 Program CDT image on device

11
12 cd boot_images\core\boot\secboot3\scripts\emmc_cdt_program_scripts

13 python ../../../../storage/tools/ptool/ptool.py -x partition.xml -p 2

14 modify <jedec_lpddr2_single_channel.xml>

15
16 QRD HW V2:

17 0x03, 0x0B, 0x02, 0x00, 0x00, 0x00, end

18
19 python cdt_generator.py jedec_lpddr2_single_channel.xml platform_ddr.bin

20 cp platform_ddr.bin emmc_cdt_program_scripts

21
22 T32

23 run platform_ddr_emmc_cdt_program.cmm

24 QPST

25 platform_ddr_prog_cfg.xml

26
27 Change default CDT in sb11 image

28 copy boot_cdt_array.c to boot_images\core\boot\secboot3\hw\msm8916

29 modify config_data_table[] in boot_cdt_array.c

30 rebuild sb11

31 program sb11 image

32 > fastboot flash sb11 sb1.mbn or use QPST emmcdownload.exe

33
34 Generate DT Binary

35 <modify dts>

36 dtc -p 1024 -O dtb -o msm8916.dtb msm8916.dts

37 cat zImage msm8916.dtb > boot.img

38 program boot image

39 > fastboot flash boot boot.img

40
41
42 EEPROM CDT reference *Application Note: EEPROM Software Configuration Data Table (CDT)*
43 (80-N3411-1).

9.2 Device Tree

9.2.1 Introduction

In order to make life easier for embedded board vendors, device tree can be used to represent devices in the system by declaring some nodes and properties in one simple file (dts). Even for on-chip devices and other buses that don't specifically fit in an existing OF specification, device tree is also recommended.

It creates a great flexibility. In the way, the kernel can probe those and match drivers to device, without having to hard code all sorts of tables. It's also more flexible for board vendors to do minor hardware upgrades without significantly impacting the kernel code or cluttering it with special cases.

Device tree usage helps with:

12. Reducing the effort required to port to a new board
13. Code reuse, by making drivers generic enough to support a set of compatible devices, with the differences being represented in the device tree

9.2.1.1 DTC

To parse device tree script into binary format, dtc (device tree compile) will be used.

You can use this command to generate dtb zImage directly if no kernel changes:

```
dtc -p 1024 -O dtb -o msm8916.dtb msm8916.dts
cat zImage msm.dtb > dtb-zimage
```

9.2.1.2 Bootloader

Fastboot will parse boot image and extract device tree information.

DT address = image address + page_size + kernel_actual + ramdisk_actual + second_actual

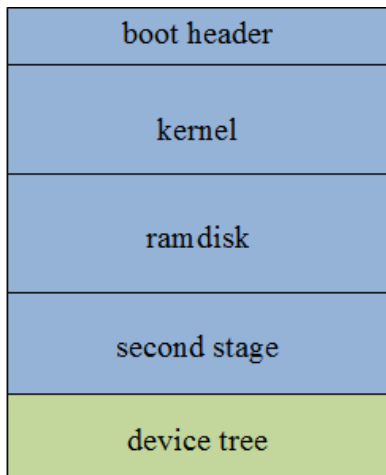


Figure 11-1 Device tree in boot image

To pass dt table to kernel, fastboot uses the same way as atags. Kernel still uses r2 to save dt table ptr.

Table 1-1 Register table in LK

Register	ATAGS	DT
R0	Null	Null
R1	Machine type number	Unique machine ID is no longer required when bootloader passes a proper device tree. The passed-in ID should be 0xFFFFFFFF to the kernel in register r1
R2	Physical address of tagged list in system RAM	Physical pointer to the device-tree block (defined in chapter II) in RAM. Device tree can be located anywhere in system RAM, but it should be aligned on a 64 bit boundary.

The code source path is arch/arm64/boot/ for 64 bits and arch/arm /boot/ for 32 bits kernel.

9.2.1.3 Kernel

Relative codes

```
kernel/drivers/of/device.c  
include/linux/of_gpio.h
```

1 Key functions

2 **Table 1-2 Device tree key API**

Function Name	Description
of_platform_populate()	System init function. It will check dt table which passed by fastboot.
of_match_device()	<p>Key function in driver probe function. You should declare it in driver. For example:</p> <pre>static const struct of_device_id s5k3llyx_dt_match[] = { {.compatible = "qcom,s5k3llyx", .data = &s5k3llyx_s_ctrl}, {} }; static struct platform_driver s5k3llyx_platform_driver = { .driver = { .name = "qcom,s5k3llyx", .owner = THIS_MODULE, .of_match_table = s5k3llyx_dt_match, }, }; static int32_t s5k3llyx_platform_probe(struct platform_device *pdev) { int32_t rc = 0; const struct of_device_id *match; match = of_match_device(s5k3llyx_dt_match, &pdev->dev); rc = msm_sensor_platform_probe(pdev, match->data); return rc; }</pre>

3 More functions information can refer to Documentation/devicetree/.

9.3 GPIO

This chapter describes the Top-Level Mode Multiplexer (TLMM) GPIO and explains how to configure it in the bootloader and kernel.

9.3.1 Hardware overview

MSM8916 has 122 GPIOs.

9.3.1.1 GPIO configuration

This section describes the configure register of GPIO in the MSM8916 chipset.

Table 1-3 GPIO physical address

Hardware Register	Physical address	Reset State
TLMM_GPIO_CFGn, n=[0..121]	0x01000000 + 0x00000000 + 0x1000 * (n)	0x000000E2
TLMM_GPIO_IN_OUTn, n=[0..121]	0x01000000 + 0x00000004 + 0x1000 * (n)	0x00000000
TLMM_GPIO_OE_0	0x01000000 + 0x00200080	0x00000000
TLMM_GPIO_OE_1	0x01000000 + 0x00200084	0x00000000
TLMM_GPIO_OE_2	0x01000000 + 0x00200088	0x00000000
TLMM_GPIO_OE_3	0x01000000 + 0x0020008C	0x00000000

Table 1-4 GPIO CFG table

Bits	Field Name	Description
10	GPIO_HIHYS_EN	Controls the hihys_en for GPIO[n]
9	GPIO_OE	Controls the OE for GPIO[n] when in GPIO mode.
8:6	DRV_STRENGTH	Controls the GPIO pad drive strength. This applies regardless of the FUNC_SEL field selection. 0: DRV_2_MA 1: DRV_4_MA 2: DRV_6_MA 3: DRV_8_MA 4: DRV_10_MA 5: DRV_12_MA 6: DRV_14_MA 7: DRV_16_MA
5:2	FUNC_SEL	Many of the GPIO pads have one or more functional hardware interfaces behind them. This field controls how the pad is used. Set this to the appropriate value for the function desired
1:0	GPIO_PULL	The pad can be configured to employ an internal weak pull up, pull down, or keeper function. This applies regardless of the FUNC_SEL field selection. 0x0: NO_PULL (Disables all pull)

	0x1: PULL_DOWN (Weak Pull-down)	1
	0x2: KEEPER (Weak Keeper)	2
	0x3: PULL_UP (Weak Pull-Up)	3

NOTE: In CFG register, GPIO_OE can enable gpio configuration. Do not write TLMM_GPIO_OE_[0...3].

NOTE: In CFG register, GPIO_HIHYS_EN can enable gpio hysteresis on input mode, which is mostly useful for noisy low frequency signals like sleep clock etc.

9.3.1.2 Interrupt configuration

Table 1-5 GPIO interrupt CFG table

Hardware register	Physical address	Reset state
TLMM_GPIO_INTR_CFGn , n=[0..121]	0x01000000 + 0x00000008 + 0x1000 * (n)	0x000000E2
TLMM_GPIO_INTR_STATUSn , n=[0.. 121]	0x01000000 + 0x0000000C + 0x1000 * (n)	0x00000000

Table 1-6 INT CFG

Bits	Field name	Description
8	DIR_CONN_EN	0: DISABLE 1: ENABLE
7:05	TARGET_PROC	0: WCSS 1: SENSORS 2: LPA_DSP 3: RPM 4: KPSS 5: MSS 6: TZ 7: NONE
4	INTR_RAW_STATUS_EN	0: DISABLE 1: ENABLE
3:02	INTR_DECT_CTL	0: LEVEL 1: POS_EDGE 2: NEG_EDGE 3: DUAL_EDGE
1	INTR_POL_CTL	0: POLARITY_0 1: POLARITY_1
0	INTR_ENABLE	0: DISABLE 1: ENABLE

NOTE: POS_EDGE means rising edge trigger

NOTE: NEG_EDGE means falling edge trigger

NOTE: Both rising edge and falling edge are supported now

Table 1-7 GPIO wakeup interrupt

Hardware register	Physical address	Reset state
TLMM_MPM_WAKEUP_INT_EN_0	0x01000000 + 0x00100008	0x00000000
TLMM_MPM_WAKEUP_INT_EN_1	0x01000000 + 0x0010000C	0x00000000

9.3.2 Software overview

9.3.2.1 Configuring GPIO in the bootloader

You can call `gpio_tlmm_config()` to config GPIO in bootloader.

`bootable/bootloader/lk/platform/msm8916/gpio.c`

```
void gpio_tlmm_config(uint32_t gpio, uint8_t func,
                      uint8_t dir, uint8_t pull,
                      uint8_t drvstr, uint32_t enable)
{
    uint32_t val = 0;
    val |= pull;
    val |= func << 2;
    val |= drvstr << 6;
    val |= enable << 9;
    writel(val, (uint32_t *)GPIO_CONFIG_ADDR(gpio));
    return;
}
```

9.3.2.2 Configuring GPIO in the kernel

This section describes the steps required to configure any of the GPIO in the MSM8916 chipset.

For more information, refer to *Linux Pin Control Migration Guide (Internal Use Only) (80-NK256-1)*, and `Documentation/devicetree/bindings/pinctrl/msm-pinctrl.txt`

1. Add a pin grouping node to the `msm8916-pinctrl.dtsi` file.

```
&soc {
    tlmm_pinmux: pinctrl@1000000{
        client1_pins {
            /* Uses general purpose pins */
            qcom,pins = <&gp 0>, <&gp 1>;
            qcom,num-grp-pins = <2>;
            /* function setting as specified in board-<soc>-gpiomux.c */
            qcom,pin-func = <1>;
            label = "client1-bus";
            /* Active configuration of bus pins */
            client1_default: client1_default {
```

```

1      /* Property names as specified in pinctrl-bindings.txt /
2      drive-strength = <8>; /* 8 MA */
3      bias-disable; /* No PULL */
4      };
5
6      };
7
8      2. Modify the client node in the device tree (msm8916.dtsi or msm8916-<board>.dtsi).
9      Soc {
10         Client1 {
11             /*Please note default state is programmed by the kernel at the time
12             the time of kernel bootup. No driver changes necessary, since at
13             probe time the default state would already be programmed in TLMM */
14             pinctrl-states = "default";
15             /* Use phandle reference to default configuration node */
16             pinctrl-0 = <&Client1_default>;
17         };
18
19         3. Delete the GPIO configuration from the board-<soc>-gpiomux.c file.
20         static struct msm_gpiomux_config msm_client1_configs[] __initdata = {
21             {
22                 .gpio = 0,
23                 .settings = {
24                     [GPIOMUX_SUSPENDED] = &gpio_client_config,
25                 },
26             },
27             {
28                 .gpio = 1,
29                 .settings = {
30                     [GPIOMUX_SUSPENDED] = &gpio_client_config,
31                 },
32             };
33         }
34     }

```

9.3.2.3 Step by step

Check drivers/i2c/muxes/i2c-mux-pinctrl.c

1. Get gpio information:

```
of_property_read_string_index(np, "pinctrl-names", i, out)
```

2. Parse the pin control:

```
devm_pinctrl_get()
```

3. Install a given state by using pinctrl_lookup_state() and pinctrl_select_state():

```

for (i = 0; i < mux->pdata->bus_count; i++) {
    mux->states[i] = pinctrl_lookup_state(mux->pinctrl,
        mux->pdata->pinctrl_states[i]);
}

```

```

1      ...
2  }
3  ...
4  for (i = 0; i < mux->pdata->bus_count; i++) {
5      u32 bus = mux->pdata->base_bus_num ?
6          (mux->pdata->base_bus_num + i) : 0;
7
8      mux->busses[i] = i2c_add_mux_adapter(mux->parent, &pdev->dev,
9          mux, bus, i, 0,
10         i2c_mux_pinctrl_select,
11         deselect);
12
13     }

```

4. Add dts:

```

15     i2c0_active {
16         /* CLK, DATA */
17         qcom,pins = <&gp 7>, <&gp 6>;
18         qcom,num-grp-pins = <2>;
19         qcom,pin-func = <3>;
20         label = "i2c0-active";
21         /* active state */
22         i2c_default: default {
23             drive-strength = <2>; /* 2 MA */
24             bias-disable = <0>; /* No PULL */
25         };
26     };

```

9.4 I2C

This chapter describes the Inter-Integrated Circuit (I2C) and explains how to configure it in the kernel.

9.4.1 Hardware overview

9.4.1.1 Qualcomm universal serial engine (QUP)

The QUP provides a general purpose data path engine to support multiple mini cores. Each mini core implements protocol-specific logic. The common FIFO provides a consistent system IO buffer and system DMA model across widely varying external interface types. For example, one pair of FIFO buffers can support SPI and I2C mini cores independently.

9.4.1.2 I2C core

The I2C core supports I2C Standard (100 kHz) and Fast (400 kHz). The following key features have been added for the MSM8916:

- 14. BAM integration
- 15. Support for I2C tag version

9.4.1.3 QUP base addresses and IRQs

To match the labeling in the software interface manual, each QUP is identified by BLSP core and QUP core (1 to 6).

NOTE: The MSM8916 chipset contains only one BLSP core.

Table 1-8 QUP physical address

BLSP hardware ID	QUP Core	Physical address
BLSP1	BLSP 1 QUP 1	0x78B5000
BLSP1	BLSP 1 QUP 2	0x78B6000
BLSP1	BLSP 1 QUP 3	0x78B7000
BLSP1	BLSP 1 QUP 4	0x78B8000
BLSP1	BLSP 1 QUP 5	0x78B9000
BLSP1	BLSP 1 QUP 6	0x78BA000

Table 1-9 QUP IRQ

BLSP hardware ID	QUP Core	IRQ
BLSP1	BLSP 1 QUP 1	95
BLSP1	BLSP 1 QUP 2	96
BLSP1	BLSP 1 QUP 3	97
BLSP1	BLSP 1 QUP 4	98

BLSP1	BLSP 1 QUP 5	99
BLSP1	BLSP 1 QUP 6	100

9.4.2 Software overview

9.4.2.1 Configuring the QUP core as I2C in the kernel

This section describes the steps required to configure and use any of the QUP cores which is available in the MSM8916 chipset as an I2C device.

By default, Qualcomm Technologies Inc. (QTI) has preconfigured BLSP1_QUP2 and BLSP1_QUP5 as I2C. For more detailed information, refer to
 /kernel/arch/arm/boot/dts/qcom/msm8916.dtsi,
 Documentation/devicetree/bindings/i2c/i2c-msm-v2.txt

9.4.2.2 Code modification

The following examples show how to configure BLSP1_QUP2 as I2C for the MSM8916 chipset.

File to modify: /kernel/arch/arm/boot/dts/msm8916.dts

1. Add a new device tree node.

```
i2c_0: i2c@78b6000 { /* BLSP1 QUP2 */
    cell-index = <5>; //BUS ID can be any # recommend OEM to use large
    # so won't conflict with Qualcomm id
    compatible = "qcom,i2c-msm-v2"; //Manufacturer model
    #address-cells = <1>; // address for slave chips
    #size-cells = <0>; // size for slave chips
    reg-names = "qup_phys_addr", "bam_phys_addr"; //Name of QUP_BASE
    reg = <0x78b6000 0x600>,
        <0x7884000 0x23000>; //BLSP1 QUP5 BASE address and size
    interrupt-names = "qup_irq", "bam_irq"; //Interrupt Name
    interrupts = <0 96 0>, <0 238 0>;
    gpios = <&msmgpio 7 0>, /* SCL */
        <&msmgpio 6 0>; /* SDA */
    //Output Clk frequency (can be 100KHz, or 400KHz)
    qcom,i2c-bus-freq = <100000>;
    qcom,i2c-src-freq = <19200000>; //Source clock frequency
};
```

2. Add a clock node.

This step describes how to modify the clock table.

File to modify:

Project_Root/drivers/clk/qcom/clock-gcc-8916.c

The code for adding a clock node:

```

1
2      /* Clock lookup */
3      static struct clk_lookup msm_clocks_lookup[] = {
4          /*Add node to BLSP1 AHB Clock
5          CLK_LIST(gcc_blsp1_ahb_clk),
6          /*
7          Add a node to QUP Core clock.
8          Note: In clock regime QUP cores are label #1 to #6.
9          */
10         CLK_LIST(gcc_blsp1_qup1_i2c_apps_clk),
11

```

3. Set up the GPIO.

File to modify:

arch/arm/boot/dts/qcom/msm8916-pinctrl.dtsi

```

16     i2c0_active {
17         /* CLK, DATA */
18         qcom,pins = <&gp 7>, <&gp 6>;
19         qcom,num-grp-pins = <2>;
20         qcom,pin-func = <3>;
21         label = "i2c0-active";
22         /* active state */
23         i2c_default: default {
24             drive-strength = <2>; /* 2 MA */
25             bias-disable = <0>; /* No PULL */
26         };
27     };
28
29     i2c0_suspend {
30         /* CLK, DATA */
31         qcom,pins = <&gp 7>, <&gp 6>;
32         qcom,num-grp-pins = <2>;
33         qcom,pin-func = <0>;
34         label = "i2c0-suspend";
35         /*suspended state */
36         i2c_sleep: sleep {
37             drive-strength = <2>; /* 2 MA */
38             bias-disable = <0>; /* No PULL */
39         };
40     };
41

```

4. Registering the GPIOs:

```

1      static int i2c_msm_rsrcs_gpio_pinctrl_init(struct i2c_msm_ctrl *ctrl)
2      {
3          ctrl->rsrcs.pinctrl = devm_pinctrl_get(ctrl->dev);
4          if (IS_ERR_OR_NULL(ctrl->rsrcs.pinctrl)) {
5              dev_err(ctrl->dev, "failed to get pinctrl\n");
6              return PTR_ERR(ctrl->rsrcs.pinctrl);
7          }
8
9          ctrl->rsrcs.gpio_state_active
10             = pinctrl_lookup_state(ctrl->rsrcs.pinctrl,
11                                     PINCTRL_STATE_DEFAULT);
12          if (IS_ERR_OR_NULL(ctrl->rsrcs.gpio_state_active))
13              dev_info(ctrl->dev, "can not get default pinstate\n");
14
15          ctrl->rsrcs.gpio_state_suspend
16             = pinctrl_lookup_state(ctrl->rsrcs.pinctrl,
17                                     PINCTRL_STATE_SLEEP);
18          if (IS_ERR_OR_NULL(ctrl->rsrcs.gpio_state_suspend))
19              dev_info(ctrl->dev, "can not get sleep pinstate\n");
20          return 0;
21      }

```

9.4.2.3 Quick verification

This section describes how to verify the I2C bus.

Ensure that the bus is registered.

If you enter all the information properly, you should see I2C bus registered under **/dev/i2c-**, where cell-index matches the bus number.

```

27 adb shell --> Get adb shell
28 cd /dev/
29 ls i2c* --> to List all the I2C buses
30 root@android:/dev # ls i2c*
31 ls i2c*
32 i2c-0
33 i2c-10
34 i2c-2

```

If bus is properly registered, test it using a simple test program. This example is created in the following directory: **/vendor/qcom/proprietary/kernel-tests/i2c-test/**.

Once you compile and run the program, if the I2C bus is properly programmed and the slave device responds, you will see the following output:

```

39 root@android:/data # ./i2c-test
40 ./i2c-test
41 Device Open successfull [3]
42 I2C RDWR Returned 1

```

If an error occurs, you will see the following output:

```
./i2c-test
```

```
Device Open successfull [3]
```

```
I2C RDWR Returned -1
```

If I2C RDWR returns -1, check the kernel log for the driver error message. If you see the following error message:

```
[ 6131.397699] qup_i2c f9927000.i2c: I2C slave addr:0x54 not connected
```

Typically, the slave device doesn't send an acknowledgment. The bus is properly configured and at least the start bit and address bit are sent from the bus, but the slave refuses it and doesn't acknowledge it.

At this point, the debugging should focus on the slave device to make sure it is properly powered up and ready to accept messages.

If you see the following error message:

```
[ 6190.209880] qup_i2c f9927000.i2c: Transaction timed out, SL-AD = 0x54
```

```
[ 6190.216389] qup_i2c f9927000.i2c: I2C Status: 132100
```

```
[ 6190.221247] qup_i2c f9927000.i2c: QUP Status: 0
```

```
[ 6190.225857] qup_i2c f9927000.i2c: OP Flags: 10
```

This error message could be due to multiple issues:

16. Invalid software configuration

17. Invalid hardware configuration

18. Slave device issues

9.4.3 Debugging tips

If you do a simple read/write, but the I2C always fails, the following sections provide debugging tips.

1. Check SDA/SCL Idling.
2. Scope the BUS to ensure that the SDA/SCL is idling at the high logic level. If it is not idling high, there is either a hardware configuration problem or the GPIO settings are invalid.
3. Set a breakpoint at the line where the error message is coming. For example, at the Transaction timed out message:

```
static int
i2c_msm_frmwrk_xfer(struct i2c_adapter *adap, struct i2c_msg msgs[], int
num)
{
...//Put a breakpoint inside if statement.
if (!timeout) {
uint32_t istatus = readl_relaxed(dev->base +
QUP_I2C_STATUS);
...
}
```

1 }

2 **9.4.4 Step by step**

3 **9.4.4.1 Registering a slave device using the device tree**

4 Once the I2C bus is properly verified, you can create a slave device driver and register it with I2C
5 bus. Refer to the following files for examples.

6 For an I2C slave device, refer to `msm8916-qrd.dts`.

7 For Synaptic Touch Screen driver registration, refer to
8 `kernel/drivers/input/touchscreen/synaptics_i2c_rmi4.c`.

9 The following example shows the minimum requirement for properly registering a slave device
10 using the device tree.

11 **9.4.4.2 Creating a device tree node**

12 File to modify:

13 `/kernel/arch/arm/boot/dts/msm8916-qrd.dts`

15 Add a new device tree node:

```
16 &i2c_0 { /* BLSP1 QUP2 */
17     synaptics@c { //Slave driver and slave Address
18         compatible = "synaptics,rmi4"; //Manufacture, model
19
20         reg = <0x0c>; //Slave Address
21         interrupt-parent = <&msmgpio>; //GPIO handler
22         interrupts = <17 0x2>; //GPIO # will be converted to gpio_irq
23         vdd-supply = <&pwm8916_117>;
24         vio-supply = <&pwm8916_16>;
25         synaptics,reset-gpio = <&msmgpio 16 0x00>; //Pass a GPIO
26         synaptics,irq-gpio = <&msmgpio 17 0x00>;
27         synaptics,button-map = <139 102 158>;
28         synaptics,i2c-pull-up;
29         synaptics,reg-en;
30     };
31 }
```

32 **9.4.4.3 Slave driver**

```
33     #include <linux/module.h>
34     #include <linux/init.h>
35     #include <linux/delay.h>
36     #include <linux/i2c.h>
37     #include <linux/interrupt.h>
38     #include <linux/slab.h>
```

```
1      #include <linux/gpio.h>
2      #include <linux/debugfs.h>
3      #include <linux/seq_file.h>
4      #include <linux/regulator/consumer.h>
5      #include <linux/string.h>
6      #include <linux/of_gpio.h>
7
8      #ifdef CONFIG_OF //Open firmware must be defined for dts usage
9      static struct of_device_id qcom_i2c_test_table[] = {
10         { .compatible = "qcom,i2c-test", }, //Compatible node must match dts
11         { },
12     };
13     #else
14     #define qcom_i2c_test_table NULL
15     #endif
16
17     //I2C slave id supported by driver
18     static const struct i2c_device_id qcom_id[] = {
19         { "qcom_i2c_test", 0 },
20         { }
21     };
22
23     static int i2c_test_test_transfer(struct i2c_client *client)
24     {
25         struct i2c_msg xfer; //I2C transfer structure
26         u8 buf = 0x55; //data to transfer
27         xfer.addr = client->addr;
28         xfer.flags = 0;
29         xfer.len = 1;
30         xfer.buf = &buf;
31
32         return i2c_transfer(client->adapter, &xfer, 1);
33     }
34
35     static int __devinit i2c_test_probe(struct i2c_client *client,
36         const struct i2c_device_id *id)
37     {
38         int irq_gpio = -1;
39         int irq;
40         int addr;
41         //Parse data using dt.
42         if(client->dev.of_node){
43             irq_gpio = of_get_named_gpio_flags(client->dev.of_node,
44 "qcom_i2c_test,irq-gpio", 0, NULL);
```

```
1      }
2      irq = client->irq; //GPIO irq #. already converted to gpio_to_irq
3      addr = client->addr; //Slave Addr
4      dev_err(&client->dev, "gpio [%d] irq [%d] gpio_irq [%d] Slaveaddr [%x]
5      \n", irq_gpio, irq,
6      gpio_to_irq(irq_gpio), addr);
7
8      //You can initiate a I2C transfer anytime using i2c_client *client
9      structure
10     i2c_test_test_transfer(client);
11
12     return 0;
13 }
14
15 //I2C Driver Info
16 static struct i2c_driver i2c_test_driver = {
17     .driver = {
18         .name      = "qcom_i2c_test",
19         .owner      = THIS_MODULE,
20         .of_match_table = qcom_i2c_test_table,
21     },
22     .probe         = i2c_test_probe,
23     .id_table       = qcom_id,
24 };
25
26 //Easy wrapper to do driver init
27 module_i2c_driver(i2c_test_driver);
28
29 MODULE_DESCRIPTION("I2C TEST");
30 MODULE_LICENSE("GPL v2");
```

31 In the kernel log, the following message indicates the Device Tree was successfully configured.

```
32
33 <3>[    2.670731] qcom_i2c_test 2-0052: gpio [61] irq [306] gpio_irq [306]
34                Slaveaddr [52]
```

9.5 UART

This chapter describes the UART and explains how to configure it in the bootloader and kernel.

9.5.1 Hardware overview

9.5.1.1 BLSP

BLSP is a new design that replaces the legacy GSBI core in the MSM8916 chipset families. All legacy GSBI software registers have been removed.

Each BLSP block includes six QUP and six UART cores. Instead of the legacy data mover (ADM), BAM is used as a hardware data mover. Each BLSP peripheral is statically connected to a pair of BAM pipes, consists of 26 pipes that can be used for data move operations and supports both BAM and non-BAM based data transfer.

9.5.1.2 UART core

Key features are added for the chipset:

- BAM support
- Single-character mode

The UART core is used for transmitting and receiving data through a serial interface. It is used for communicating with other UART protocol devices. Configuration of this mode is primarily defined by the UART_DM_MR1 and UART_DM_MR2 registers.

9.5.1.3 Base addresses

To match the labeling in the software interface manual, each UART is identified by the BLSP core 1 and UART core (1 to 2).

Table 1-10 UART physical address

BLSP hardware ID	UART Core	Physical address (UART_DM_BASE_ADDRESS)
BLSP1	BLSP 1 UART 1	0x78AF000
BLSP1	BLSP 1 UART 2	0x78B0000

9.5.1.4 IRQ numbers

To match the labeling in the software interface manual, each UART is identified by the BLSP core 1 and UART core (1 to 2).

Table 1-11 UART interrupt table

BLSP hardware ID	UART Core	IRQ #
BLSP1	BLSP 1 UART 1	107
BLSP1	BLSP 1 UART 2	108

9.5.2 Software overview

9.5.2.1 UART in the bootloader

By default, QTI is configured the BLSP1 UART2 in the bootloader to use. This section describes the changes required to configure a different UART for debugging purpose.

Enabling the UART for debugging

File to modify:

Project_Root/bootable/bootloader/lk/project/msm8916.mk

Set the flag, WITH_DEBUG_UART, to TRUE:

```
DEFINES += WITH_DEBUG_UART=1
```

Setting the base address

File to modify:

Project_Root/bootable/bootloader/lk/target/msm8916/init.c

Set the correct base address:

```
void target_early_init(void)
{
#ifdef WITH_DEBUG_UART
    /*
     * First argument represents the ID (can be any since it's not used)
     * Second argument, if it is a GSBI base, must be 0
     * Third Argument is the physical address for UART CORE defined in
     * /bootable/bootloader/lk/platform/msm8916/include/platform/iomap.h
     */
    uart_dm_init(1, 0, BLSP1_UART2_BASE); //it is uart[0..2] instead of
    uart[1..2]
#endif
}
```

Configuring the clock

Two clocks are required for the UART to properly work on the MSM8916 device:

19. BLSP AHB clock

20. UART core clock

File to modify:

Project_Root/bootable/bootloader/lk/platform/msm8916/acpuclock.c

Enable the clocks:

```
void clock_config_uart_dm(uint8_t id)
```

```

1      {
2          int ret;
3          /*
4              NOTE: In clock regime clocks are # from 1 to 2 so UART0 would
5              be identified as UART1
6          */
7          //iface_clk is BLSP clk
8          ret = clk_get_set_enable("uart2_iface_clk", 0, 1);
9
10         //core_clock is UART clock.
11         ret = clk_get_set_enable("uart2_core_clk", 7372800, 1);
12     }

```

13

14 Register the clocks with the clock regime. The BLSP1_AHB clock is enabled by default.

15 Configuring the GPIO

16 File to modify:

17 Project_Root/bootable/bootloader/lk/platform/msm8916/gpio.c

18 Configure the correct GPIO:

```

19 void gpio_config_uart_dm(uint8_t id)
20 {
21     /*
22     Configure the RX/TX GPIO
23     Argument 1: GPIO #
24     Argument 2: Function (Please see device pinout for more information)
25     Argument 3: Input/Output (Can be 0/1)
26     Argument 4: Should be no PULL
27     Argument 5: Drive strength
28     Argument 6: Output Enable (Can be 0/1)
29     */
30     gpio_tlmm_config(5, 2, GPIO_INPUT, GPIO_NO_PULL,
31                     GPIO_8MA, GPIO_DISABLE);
32     gpio_tlmm_config(4, 2, GPIO_OUTPUT, GPIO_NO_PULL,
33                     GPIO_8MA, GPIO_DISABLE);
34 }

```

35 Debugging tips

36 If the UART is properly configured, the following message is displayed on the serial console:

37 **Android Bootloader - UART_DM Initialized!!!**

38 If you do not see a message, use the following debugging tips.

- 39 ■ Set a breakpoint

Put a breakpoint in the bootloader inside the `uart_dm_init` function line at `msm_boot_uart_dm_write (uart_dm_base, data, 44)` in this file:

`/bootable/bootloader/lk/platform/msm_shared/uart_dm.c`

- Check properly configured GPIOs

Check the GPIO configuration register, `GPIO_CFGn`, to ensure that the GPIO settings valid.

9.5.2.2 Low-speed UART in the kernel

Low-speed UART is a FIFO-based UART driver designed for small data transfer at a slow rate, such as console debugging or IrDA transfer. If a large amount of data is transferred, or for a high-speed transfer, QTI recommends using the high-speed UART driver.

By default, BLSP1 UART2 Base `0x78B0000` is preconfigured as low-speed UART.

Code modification

The following examples describe how to configure BLSP1 UART2 to use the low-speed UART driver on the MSM8916 chipset.

The same methods can be applied to the MSM8916 chipset, but with the following modifications:

1. Device Tree Source -- `/arch/arm/boot/dts/qcom/msm8916.dtsi`
2. Clk Table -- `/drivers/clk/qcom/clock-gcc-8916.c`

In this file, update the `msm_clocks_lookup []` table.

3. GPIO Table -- `/kernel/arch/arm/boot/dts/qcom/msm8916-pinctrl.dtsi`
4. Creating a device tree node

For more details, refer to the device tree source:

`/kernel/arch/arm/boot/dts/qcom/msm8916.dtsi`

NOTE: Currently, there is no ID field to explicitly assign to the UART. Therefore, the first UART registered is `ttyHSL0`, the second one is `ttyHSL1`, etc.

1. File to modify:

`/kernel/arch/arm/boot/dts/qcom/msm8916.dtsi`

Add a new Device Tree node:

```
blsp1_uart2: serial@78b0000 { // 78b0000 is the UART_DM Base Address
    //manufacture, model of serial driver
    compatible = "qcom,msm-lsuart-v14";
    //Base address UART_DM and size
    /*
    First Field: 0 SPI interrupt (Shared Peripheral Interrupt)
    Second Field: Interrupt #
    Third field: Trigger type, keep 0
    information:/kernel/Documentation/devicetree/bindings/arm/gic.txt
```

```

1      */
2      reg = <0x78b0000 0x200>;
3      interrupts = <0 108 0>;
4      status = " ok "; //Status OK enables it
5      clocks = <&clock_gcc clk_gcc_blsp1_uart2_apps_clk>,
6              <&clock_gcc clk_gcc_blsp1_ahb_clk>;
7      clock-names = "core_clk", "iface_clk";
8  };

```

2. Set up the clocks

File to modify:

drivers/clock/qcom/clock-gcc-8916.c

3. Add a clock node:

```

14  /* Clock lookup */
15  static struct clk_lookup msm_clocks_lookup[] = {
16  //Add node to BLSP1 AHB Clock
17
18  CLK_LIST(blsp1_qup1_uart1_apps_clk_src),
19  CLK_LIST(blsp1_qup1_uart2_apps_clk_src),
20  /*
21  Add a node to QUP Core clock.
22  Note: In clock regime UART cores are label #1 to #2.
23  */
24  CLK_LIST(blsp1_uart1_apps_clk_src),
25  CLK_LIST(blsp1_uart2_apps_clk_src),

```

4. Set up the GPIO

File to modify:

kernel/arch/arm/boot/dts/qcom/msm8916-pinctrl.dtsi

```

30  pmx-uartconsole {
31      qcom,pins = <&gp 4>, <&gp 5>;
32      qcom,num-grp-pins = <2>;
33      qcom,pin-func = <2>;
34      label = "uart-console";
35      uart_console_sleep: uart-console {
36          drive-strength = <16>;
37          bias-disable;
38      };
39  };

```

5. Debugging tips

This section describes how to verify the low-speed UART.

1 a. Checking the registration

2 Ensure that the UART is properly registered with the TTY stack. Run the following
3 commands:

4 `adb shell -> start a new shell`
5 `ls /dev/ttyHSL* -> Make sure UART is properly registered`
6

7 If you do not see your device, check your code modification to ensure that all the
8 information is defined and correct.

9 b. Checking the Internal loopback

10 i. Run the following commands to enable loopback:

11 `adb shell`
12 `mount -t debugfs none /sys/kernel/debug -> mount debug fs`
13 `cd /sys/kernel/debug/msm_serial_hsl -> directory for Low`
14 `Speed UART`
15 `echo 1 > loopback.# -> enable loopback. # is`
16 `device #`
17 `cat loopback.# -> make sure returns 1`

18 ii. Open another shell to dump the UART Rx data:

19 `adb shell`
20 `cat /dev/ttyHSL# ->Dump any data UART Receive`

21 iii. Transmit some test data through a separate shell:

22 `adb shell`
23 `echo "This Document Is Very Much Helpful" > /dev/ttyHSL# -`
24 `>Transfer data`

25 ■ If the loopback works:

- 26 □ You will see your test message loop continuously in the command shell until you exit the
- 27 cat program. This is because of the internal loopback and how the cat program opens the
- 28 UART.
- 29 □ You can assume the UART is properly configured and only the GPIO settings need to be
- 30 confirmed.

31 c. Loopback does not work

32 ■ If loopback does not work, check the clock settings.

33 Before checking the clock, ensure that the UART is still in the Active state. Open the
34 UART from the shell:

35 `adb shell`
36 `cat /dev/ttyHSL# ->Dump any data UART Receive`

37 d. Loopback works, but there is no output

38 If the loopback works but you do not see any signal output, check the GPIO settings.

9.5.2.3 High-speed UART in the kernel

UART_DM can be configured as a BAM-based UART. This driver is designed for high-speed, large data transfers such as Bluetooth® communication.

BLSP BAM address and IRQ

Each BLSP core has a master BAM hardware block and one dedicated IRQ.

Table 1-12 BLSP BAM physical address

BLSP hardware ID	UART Core	Physical address (BLSP_BAM, IRQ)
BLSP1	BLSP 1 UART[1:2]	0x7884000, 244

BAM pipe assignment

Each BLSP UART core has two unique pipes (Consumer and Producer) pre-allocated for data mover operations. Table UartPipeAssignment identifies the pipe to use for each BLSP UART_DM core.

Table 1-13 BAM pipe

BLSP hardware ID	UART Core	Pipes (Consumer and Producer)
BLSP1	BLSP 1 UART1	0,1
BLSP1	BLSP 1 UART2	2,3

Code modification

The following examples show how to configure BLSP1_UART2 as a high-speed UART on the MSM8916 chipset.

1. Device Tree Source -- /arch/arm/boot/dts/qcom/msm8916.dtsi

2. Clk Table -- /drivers/clk/qcom/clock-gcc-8916.c

In this file, update the msm_clocks_lookup [] table.

3. GPIO Table -- /kernel/arch/arm/boot/dts/qcom/msm8916-pinctrl.dtsi

4. Creating a device tree node

For more details, refer to the device tree source:

```
/kernel/arch/arm/boot/dts/qcom/msm8916.dtsi
```

5. ARM TrustZone Changes --

```
/trustzone_images/core/hwengines/bam/8916/bamtgtcfgdata_tz.h
```

6. Create a device tree node

For detailed information, refer to the Device Tree documentation:

```
/kernel/Documentation/devicetree/bindings/tty/serial/msm_serial_hs.txt.
```

File to modify:

/kernel/arch/arm/boot/dts/qcom/msm8916.dtsi

Or any dts/dtsi file used.

The following elements are the minimum requirement:

```
uart2: uart@0x78b0000 { //0x78b0000 is the UART_DM Base address for
BLSP1_UART2
    /*
        UART ID, Recommend OEM to use Large ID so does not conflict with
        Qualcomm configured ID.
        cell-index 102 would represent as /dev/ttyHS102
    */
    cell-index = <102>; //UART ID, Recommend OEM to use Large ID # that
< 255
    compatible = "qcom,msm-hsuart-v14"; //manufacture, model (must be
same)
    status = "ok"; //"ok" or "okay" to enable
    /*
        First Row UART_DM Base and Size always 0x1000
        Second Row is BAM address, size always 0x19000
        For BLSP1 UART0:5 Bam Address = 0xF9904000
    */
    reg = <0x78b0000 0x200>;
    interrupts = <0 108 0>;
    reg-names = "core_mem", "bam_mem"; //Keep the same names
    /*
        First Field: 0 SPI interrupt (Shared Peripheral Interrupt)
        Second Field: Interrupt #
        Third field: Trigger type, keep 0
        For more
information:/kernel/Documentation/devicetree/bindings/arm/gic.txt
        First Row UART_DM IRQ #
        Second Row is BAM IRQ
        For BLSP1 UART0:1 IRQ = 238
    */
    interrupts = <0 108 0>, <0 238 0>;
    interrupt-names = "core_irq", "bam_irq"; //Keep same
    qcom,bam-tx-ep-pipe-index = <2>; //BAM Consumer Pipe
    qcom,bam-rx-ep-pipe-index = <3>; //BAM Producer Pipe
};
```

7. Set up the clocks

NOTE: This section is based on the MSM8916-Pre-ES release.

File to modify:

drivers/clk/qcom/clock-gcc-8916.c

Add a node to the UART core clock:

```
/* Clock lookup */
static struct clk_lookup msm_clocks_lookup[] = {
//Add node to BLSP1 AHB Clock

CLK_LIST(blsp1_qup1_uart1_apps_clk_src),
CLK_LIST(blsp1_qup1_uart2_apps_clk_src),
/*
Add a node to QUP Core clock.
Note: In clock regime UART cores are label #1 to #2.
*/
CLK_LIST(blsp1_uart1_apps_clk_src),
CLK_LIST(blsp1_uart2_apps_clk_src),
```

8. Set up the GPIO

NOTE: This section is based on the MSM8916-Pre-ES release;

File to modify:

kernel/arch/arm/boot/dts/qcom/msm8916-pinctrl.dtsi

```
pmx-uartconsole {
    qcom,pins = <&gp 4>, <&gp 5>, <&gp 6>, <&gp 7>;
    qcom,num-grp-pins = <4>;
    qcom,pin-func = <2>;
    label = "uart-console";
    uart_console_sleep: uart-console {
        drive-strength = <16>;
        bias-disable;
    };
};
```

9. Change TrustZone

Each BLSP can be used by any subsystem such as Modem and LPASS. It is important to give ownership of the BAM Pipes to Applications processor.

File to modify:

/trustzone_images/core/hwengines/bam/8916/bamtgtcfgdata_tz.h


```

1      Allocate BLSP BAM pipes with the applications core:
2      /*
3          bam_tgt_blspl1_secconfig = BLSP1 Core
4
5          Each Bit You set represent Pipe #.
6          For example Bit 0 = Pipe # 0
7                      Bit 5 = Pipe # 5
8
9          Any bit set on TZBSP_VMID_AP owns by APPs
10     bam_sec_config_type bam_tgt_cel_secconfig_8916 =
11     {
12         {
13             { 0x800000C3 , TZBSP_VMID_TZ, 0x0, TZBSP_VMID_TZ_BIT}, /* krait tz
14         */
15             { 0x0000003C , TZBSP_VMID_AP, 0x0, TZBSP_VMID_AP_BIT} /* krait
16     apps */
17         },
18         { {0x0} /* SG not supported*/
19         },
20         TZBSP_VMID_TZ_BIT,
21         0x0 /* IRQ will be routed to EE0 */
22     };

```

Debugging tips

This section describes how to verify the high-speed UART.

■ Check the registration

Ensure that the UART is properly registered with the TTY stack. Run the following commands:

```
adb shell -> start a new shell
```

```
ls /dev/ttyHS* -> Make sure UART is properly registered
```

If you do not see your device, check your code modification to ensure that all the information is defined and correct.

■ Check the Internal loopback

a. Run the following commands to enable loopback:

```
adb shell
```

```
mount -t debugfs none /sys/kernel/debug -> mount debug fs
```

```
cd /sys/kernel/debug/msm_serial_hs -> directory for High Speed
UART
```

```
echo 1 > loopback.# -> enable loopback. # is
```

```
device #
```

```
cat loopback.# -> make sure returns 1
```

b. Open another shell to dump the UART Rx data:

```
adb shell
cat /dev/ttyHS# ->Dump any data UART Receive
c. Transmit some test data through a separate shell:
adb shell
echo "This Is A Helpful Document" > /dev/ttyHS# ->Transfer data
```

If the loopback works:

You will see your test message loop continuously in the command shell until you exit the cat program. This is because of the internal loopback and how the cat program opens the UART.

You can assume the UART is properly configured and only the GPIO settings need to be confirmed.

- Loopback does not work

For the latest details, refer to the Device Tree documentation:

/kernel/Documentation/devicetree/bindings/tty/serial/msm_serial_hs.txt.

9.6 Serial Peripheral Interface

This chapter describes the Serial Peripheral Interface (SPI) and explains how to configure it in the kernel.

9.6.1 Hardware overview

9.6.1.1 SPI core

The SPI allows full-half-duplex, synchronous, serial communication between a master and slave. There is no explicit communication framing, error checking, or defined data word length.

Key features:

- 21. Supports up to 48 MHz
- 22. Supports 4 to 32 bits per word of transfer
- 23. Supports a maximum of four Chip Selects (CS) per bus
- 24. Supports BAM (new for MSM8916 chipsets)

Table 1-14 Base address

BLSP hardware ID	QUP Core	Physical address
BLSP1	BLSP 1 QUP 1	0x78B5000
BLSP1	BLSP 1 QUP 2	0x78B6000
BLSP1	BLSP 1 QUP 3	0x78B7000
BLSP1	BLSP 1 QUP 4	0x78B8000
BLSP1	BLSP 1 QUP 5	0x78B9000
BLSP1	BLSP 1 QUP 6	0x78BA000

The SPI core shares the same base address as the QUP core; or the correct values.

9.6.2 Software overview

9.6.2.1 Configuring the QUP core as SPI in the kernel

This section describes the steps required to configure and use any of the QUP cores available in the MSM8916 chipset as an SPI device.

By default, QTI has preconfigured BLSP1_QUP3 as SPI. For more detailed information, refer to `/kernel/arch/arm/boot/dts/qcom/msm8916.dtsi`.

Code modification

The following examples show how to configure BLSP1_QUP3 as SPI for the MSM8916 chipset. The same methods can be applied to the MSM8916 chipset, but with the following modifications:

Device Tree Source -- `/kernel/arch/arm/boot/dts/qcom/msm8916.dtsi`

1 Clk Table -- Project_Root/drivers/clk/qcom/clock-gcc-8916.c

2
3 In this file, update the msm_clocks_lookup[] table.

4 GPIO Table -- arch/arm/boot/dts/qcom/msm8916-pinctrl.dtsi

5
6 TrustZone Changes --

7 /trustzone_images/core/hwengines/bam/8916/bamtgtcfgdata_tz.h

8 1. Creating a device tree

9 File to modify:

10 /kernel/arch/arm/boot/dts/qcom/msm8916.dts

11
12 Add a new Device Tree node:

```
13 spi_0: spi@78b7000 { /* BLSP1 QUP3 */
14     compatible = "qcom,spi-qup-v2";
15     #address-cells = <1>;
16     #size-cells = <0>;
17     reg-names = "spi_physical", "spi_bam_physical";
18     reg = <0x78b7000 0x600>,
19         <0x7884000 0x23000>;
20     interrupt-names = "spi_irq", "spi_bam_irq";
21     interrupts = <0 97 0>, <0 238 0>;
22     spi-max-frequency = <19200000>;
23     pinctrl-names = "default", "sleep", "cs_default";
24     pinctrl-0 = <&spi0_default>;
25     pinctrl-1 = <&spi0_sleep>;
26     pinctrl-2 = <&spi0_cs_sleep>;
27     clocks = <&clock_gcc clk_gcc_blspi_ahb_clk>,
28         <&clock_gcc clk_gcc_blspi_qup1_spi_apps_clk>;
29     clock-names = "iface_clk", "core_clk";
30     qcom,infinite-mode = <0>;
31     qcom,use-bam;
32     qcom,use-pinctrl;
33     qcom,ver-reg-exists;
34     qcom,bam-consumer-pipe-index = <8>;
35     qcom,bam-producer-pipe-index = <9>;
36     qcom,master-id = <86>;
37     status = "ok";
38 };
```

39
40 For latest details follow:

41 /kernel/Documentation/devicetree/bindings/spi/spi_qsd.txt.

42 2. Set the clocks

This step describes how to modify the clock table.

File to modify:

Project_Root/drivers/clk/qcom/clock-gcc-8916.c

Add a clock node:

```
/* Clock lookup */
static struct clk_lookup msm_clocks_lookup[] = {
//Add node to BLSP1 AHB Clock

CLK_LIST(gcc_blspi_ahb_clk),

/*
Add a node to QUP Core clock.
*/
CLK_LIST(blspi_qup3_spi_apps_clk_src),
```

3. Set the GPIO

File to modify:

Project_Root/arch/arm/boot/dts/qcom/msm8916-pinctrl.dtsi

```
spi0_active {
/* MOSI, MISO, CLK */
qcom,pins = <&gp 8>, <&gp 9>, <&gp 11>;
qcom,num-grp-pins = <3>;
qcom,pin-func = <1>;
label = "spi0-active";
/* active state */
spi0_default: default {
drive-strength = <12>; /* 12 MA */
bias-disable = <0>; /* No PULL */
};
};

spi0_suspend {
/* MOSI, MISO, CLK */
qcom,pins = <&gp 8>, <&gp 9>, <&gp 11>;
qcom,num-grp-pins = <3>;
qcom,pin-func = <0>;
label = "spi0-suspend";
/* suspended state */
spi0_sleep: sleep {
drive-strength = <2>; /* 2 MA */
bias-disable = <0>; /* No PULL */
};
};

spi0_cs0_active {
/* CS */
qcom,pins = <&gp 10>;
qcom,num-grp-pins = <1>;
```

```

1          qcom,pin-func = <1>;
2          label = "spi0-cs0-active";
3          spi0_cs0_active: cs0_active {
4              drive-strength = <2>;
5              bias-disable = <0>;
6          };
7      };
8
9
10     spi0_cs0_suspend {
11         /* CS */
12         qcom,pins = <&gp 10>;
13         qcom,num-grp-pins = <1>;
14         qcom,pin-func = <0>;
15         label = "spi0-cs0-suspend";
16         spi0_cs0_sleep: cs0_sleep {
17             drive-strength = <2>;
18             bias-disable = <0>;
19         };
20     };
21
22
23
24

```

For the latest details, refer to the Device Tree documentation:

[/kernel/Documentation/devicetree/bindings/spi/spi_qsd.txt](#).

Quick verification

If you enter all the information properly, you should see the SPI bus registered under `/sys/class/spi_master/spi`, where cell-index matches the bus number.

```

31 adb shell --> Get adb shell
32 cd /sys/class/spi_master to list all the spi master
33 root@android:/sys/class/spi_master # ls
34 ls
35 spi0
36 spi6
37 spi7

```

Registering a slave device using the device tree

Once the SPI bus is registered, you can create a slave device driver and register it with the SPI master. For examples of SPI slave devices, refer to the following files:

```

41 /kernel/arch/arm/boot/dts/qcom/msm8916.dts
42 /kernel/Documentation/devicetree/bindings/spi/spi_qsd.txt
43 /kernel/Documentation/devicetree/bindings/spi/spi-bus.txt
44

```

The following example shows the minimum requirements for registering a slave device.

1. Create a Device Tree Node

File to modify:

/kernel/arch/arm/boot/dts/qcom/msm8916.dts

Add a new Device Tree node:

```
spi_0: spi@78b7000 { /* BLSP1 QUP3 */
    compatible = "qcom,spi-qup-v2";
    #address-cells = <1>;
    #size-cells = <0>;
    reg-names = "spi_physical", "spi_bam_physical";
    reg = <0x78b7000 0x600>,
        <0x7884000 0x23000>;
    interrupt-names = "spi_irq", "spi_bam_irq";
    interrupts = <0 97 0>, <0 238 0>;
    spi-max-frequency = <19200000>;
    pinctrl-names = "default", "sleep", "cs_default";
    pinctrl-0 = <&spi0_default>;
    pinctrl-1 = <&spi0_sleep>;
    pinctrl-2 = <&spi0_cs_sleep>;
    clocks = <&clock_gcc clk_gcc_blspi_ahb_clk>,
        <&clock_gcc clk_gcc_blspi_qup1_spi_apps_clk>;
    clock-names = "iface_clk", "core_clk";
    qcom,infinite-mode = <0>;
    qcom,use-bam;
    qcom,use-pinctrl;
    qcom,ver-reg-exists;
    qcom,bam-consumer-pipe-index = <8>;
    qcom,bam-producer-pipe-index = <9>;
    qcom,master-id = <86>;
    status = "disabled";
};
```

2. Sample Slave Device Driver

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/spi/spi.h>
#include <linux/interrupt.h>
#include <linux/slab.h>
#include <linux/gpio.h>
#include <linux/debugfs.h>
#include <linux/seq_file.h>
#include <linux/regulator/consumer.h>
```

```
1      #include <linux/string.h>
2      #include <linux/of_gpio.h>
3
4      #ifdef CONFIG_OF //Open firmware must be defined for dts useage
5      static struct of_device_id qcom_spi_test_table[] = {
6      { .compatible = "qcom,spi-test",}, //Compatible node must match dts
7      { },
8      };
9      #else
10     #define qcom_spi_test_table NULL
11     #endif
12
13
14     #define BUFFER_SIZE 4<<10
15     struct spi_message spi_msg;
16     struct spi_transfer spi_xfer;
17     u8 *tx_buf; //This needs to be DMA friendly buffer
18     static int spi_test_transfer(struct spi_device *spi)
19     {
20     spi_message_init(&spi_msg);
21
22     spi_xfer.tx_buf = tx_buf;
23     spi_xfer.len = BUFFER_SIZE;
24     spi_xfer.bits_per_word = 8;
25     spi_xfer.speed_hz = spi->max_speed_hz;
26
27     spi_message_add_tail(&spi_xfer, &spi_msg);
28
29     return spi_sync(spi, &spi_msg);
30     }
31
32
33     static int __devinit spi_test_probe(struct spi_device *spi)
34     {
35
36     int irq_gpio = -1;
37     int irq;
38     int cs;
39     int cpha,cpol,cs_high;
40     u32 max_speed;
41
42     dev_err(&spi->dev, "s\n", __func__);
43
44     //allocate memory for transfer
```



```

1      tx_buf = kmalloc(BUFFER_SIZE, GFP_ATOMIC);
2      if(tx_buf == NULL){
3          dev_err(&spi->dev, "s: mem alloc failed\n", __func__);
4          return -ENOMEM;
5      }
6      //Parse data using dt.
7      if(spi->dev.of_node){
8          irq_gpio = of_get_named_gpio_flags(spi->dev.of_node,
9      "qcom_spi_test,irq-gpio", 0, NULL);
10     }
11     irq = spi->irq;
12     cs = spi->chip_select;
13     cpha = ( spi->mode & SPI_CPHA ) ? 1:0;
14     cpol = ( spi->mode & SPI_CPOL ) ? 1:0;
15     cs_high = ( spi->mode & SPI_CS_HIGH ) ? 1:0;
16     max_speed = spi->max_speed_hz;
17     dev_err(&spi->dev, "gpio [d] irq [d] gpio_irq [d] cs [x] CPHA [x] CPOL
18     [x] CS_HIGH [x]\n",
19         irq_gpio, irq, gpio_to_irq(irq_gpio), cs, cpha, cpol, cs_high);
20
21     dev_err(&spi->dev, "Max_speed [d]\n", max_speed );
22
23     //Once you have a spi_device structure you can do a transfer anytime
24     spi->bits_per_word = 8;
25     dev_err(&spi->dev, "SPI sync returned [d]\n",  spi_test_transfer(spi));
26     return 0;
27 }
28
29
30 //SPI Driver Info
31 static struct spi_driver spi_test_driver = {
32     .driver = {
33         .name      = "qcom_spi_test",
34         .owner      = THIS_MODULE,
35         .of_match_table = qcom_spi_test_table,
36     },
37     .probe          = spi_test_probe,
38 };
39
40
41 static int __init spi_test_init(void)
42 {
43     return spi_register_driver(&spi_test_driver);
44 }
45

```

```

1      static void __exit spi_test_exit(void)
2      {
3      spi_unregister_driver(&spi_test_driver);
4      }
5
6
7      module_init(spi_test_init);
8      module_exit(spi_test_exit);
9      MODULE_DESCRIPTION("SPI TEST");
10     MODULE_LICENSE("GPL v2");
11

```

In the kernel log, the following message indicates the Device Tree was successfully configured.

```

13 <3>[ 2.503571] qcom_spi_test spi6.0: spi_test_probe
14 <3>[ 2.507305] qcom_spi_test spi6.0: gpio [61] irq [306] gpio_irq [306]
15          cs [0] CPHA [1] CPOL [1] CS_HIGH [1]
16 <3>[ 2.516825] qcom_spi_test spi6.0: Max_speed [4800000]
17 <3>[ 2.521932] qcom_spi_test spi6.0: SPI sync returned [0]

```

9.6.2.2 Optional – Enable data mover mode (BAM)

The SPI can operate in Data Mover mode (BAM) or FIFO-based mode. If large amounts of data are to be transferred, QTI recommends enabling BAM to offload the CPU.

BLSP BAM address and IRQ

Each BLSP core has a master BAM hardware block and one dedicated IRQ.

Table 1-15 BLSP BAM physical address

BLSP HW ID	QUP Cores	BLSP_BAM, IRQ
BLSP1	BLSP1_QUP[1:6]	0x7884000, 244

Table 1-16 BAM pipe assignment

BLSP HW ID	QUP Cores	PIPE(in, out)
BLSP1	BLSP1_QUP_1	4,5
BLSP1	BLSP1_QUP_2	6,7
BLSP1	BLSP1_QUP_3	8,9
BLSP1	BLSP1_QUP_4	10,11
BLSP1	BLSP1_QUP_5	12,13
BLSP1	BLSP1_QUP_6	14,15

Each BLSP QUP core has two unique pipes (Consumer and Producer) pre-allocated for data mover operations. The MSM8916 chipset contains only one BLSP core.

Code modification

This section describes how to enable BAM mode for the SPI.

1. Device Tree modification:

The following additional settings to the Device Tree are required to enable BAM with SPI core:

For latest detail, please follow

```
/kernel/Documentation/devicetree/bindings/spi/spi_qsd.txt
//Add following additional nodes device tree to enable BAM
spi_0: spi@78b7000 { /* BLSP1 QUP3 */
    compatible = "qcom,spi-qup-v2";
    #address-cells = <1>;
    #size-cells = <0>;
    reg-names = "spi_physical", "spi_bam_physical";
    reg = <0x78b7000 0x600>,
        <0x7884000 0x23000>;
    interrupt-names = "spi_irq", "spi_bam_irq";
    interrupts = <0 97 0>, <0 238 0>;
    spi-max-frequency = <19200000>;
    pinctrl-names = "default", "sleep", "cs_default";
    pinctrl-0 = <&spi0_default>;
    pinctrl-1 = <&spi0_sleep>;
    pinctrl-2 = <&spi0_cs_sleep>;
    clocks = <&clock_gcc clk_gcc_blbsp1_ahb_clk>,
        <&clock_gcc clk_gcc_blbsp1_qup1_spi_apps_clk>;
    clock-names = "iface_clk", "core_clk";
    qcom,infinite-mode = <0>;
    qcom,use-bam;
    qcom,use-pinctrl;
    qcom,ver-reg-exists;
    qcom,bam-consumer-pipe-index = <8>;
    qcom,bam-producer-pipe-index = <9>;
    qcom,master-id = <86>;
    status = "ok";
};
```

2. TrustZone Changes:

Each BLSP can be used by any subsystem such as Modem and LPASS. It is important to give ownership of the BAM pipes to applications processor.

File to modify:

```
/trustzone_images/core/hwengines/bam/8916/bamtgtcfgdata_tz.h
```

3. Allocate BLSP BAM pipes with the applications core:

```
/*
    bam_tgt_blbsp1_secconfig = BLSP1 Core
```

```
1      Each Bit You set represent Pipe #.  
2      For example Bit 0 = Pipe # 0  
3          Bit 5 = Pipe # 5  
4  
5      Any bit set on TZBSP_VMID_AP owns by APPs  
6      */  
7      bam_sec_config_type bam_tgt_cel_secconfig_8916 =  
8      {  
9          {  
10             { 0x800000C3 , TZBSP_VMID_TZ, 0x0, TZBSP_VMID_TZ_BIT}, /* krait  
11      tz */  
12             { 0x0000003C , TZBSP_VMID_AP, 0x0, TZBSP_VMID_AP_BIT} /* krait  
13      apps */  
14             },  
15             { 0x0 } /* SG not supported*/  
16             },  
17             TZBSP_VMID_TZ_BIT,  
18             0x0 /* IRQ will be routed to EE0 */  
19      };
```

9.7 Display

9.7.1 Introduction

This chapter provides information for OEM manufacturers to porting the Display Panel on MSM8916 build. MSM8916 just supports one display panel with MIPI DSI interface, both video mode and command mode are supported, and the maximum resolution is WXGA (up to 1280 x 800 at 60 fps).

This chapter will focus on how to port DSI panel in Linux kernel. Please read panel vendor's specification for more details of the IC driver before you start porting your display panel.

9.7.2 Kernel porting

9.7.2.1 Properties

In MSM8916 build, it is to use device tree to describe hardware, comparing to hardcoding every detail of a device into board specific files in previous chipset. One big change/advantage is to support new target with Global Component Database (GCDB). The purpose is to provide configuration options for XML inputs in display panel configuration on MSM8916, MSM8x26 and MSM8x10 chipsets. For more information about GCDB, refer to *XML Tag Description For Display Module In GCDB* (80-BA103-1).

Display panel configuration is part of device trees as well; a binding is a description of how a device is described in the device tree. You can find Bindings for display panel at `kernel\Documentation\devicetree\bindings\fb\mdss-dsi-panel.txt`.

The below table describe those display properties one by one with suggestion to OEM manufacturers.

Table 9-17 Display properties

Properties	Description	Sample codes	Suggestion for OEMs
qcom,mdss-dsi-panel-name	A string used as the descriptive name of the panel	qcom,mdss-dsi-panel-name = "tianma(otm1283a) 720p video mode dsi panel";	Customize it based on your panel name
qcom,mdss-dsi-panel-controller	Specifies the phandle for the DSIcontroller that this panel will be mapped to.	qcom,mdss-dsi-panel-controller = <&mdss_dsi0>;	Keep unchanged
qcom,mdss-dsi-panel-type	Specifies the panel operating mode. "dsi_video_mode" = enable video mode "dsi_cmd_mode" = enable command mode	qcom,mdss-dsi-panel-type = "dsi_video_mode";	Customize it based on your panel specification

qcom,mdss-dsi-panel-destination	A string that specifies the destination display for the panel. "display_1" = DISPLAY_1 "display_2" = DISPLAY_2	qcom,mdss-dsi-panel-destination = "display_1";	Keep unchanged
qcom,mdss-dsi-bpp	Specifies the panel bits per pixel. 3 = for rgb111 8 = for rgb332 12 = for rgb444 16 = for rgb565 18 = for rgb666 24 = for rgb888	qcom,mdss-dsi-bpp = <24>;	Customize it based on your panel specification
qcom,mdss-dsi-panel-framerate	Specifies the frame rate for the panel. 60 = 60 frames per second (default)	qcom,mdss-dsi-panel-framerate = <60>;	Customize it based on your panel specification
qcom,mdss-dsi-panel-width	Specifies panel width in pixels.	qcom,mdss-dsi-panel-width = <720>;	Customize it based on your panel specification
qcom,mdss-dsi-panel-height	Specifies panel height in pixels.	qcom,mdss-dsi-panel-height = <1280>;	Customize it based on your panel specification
qcom,mdss-dsi-h-back-porch	Horizontal back porch value in pixel.	qcom,mdss-dsi-h-back-porch = <100>;	Customize it based on your panel specification
qcom,mdss-dsi-h-front-porch	Horizontal front porch value in pixel.	qcom,mdss-dsi-h-front-porch = <52>;	Customize it based on your panel specification
qcom,mdss-dsi-h-pulse-width	Horizontal pulse width.	qcom,mdss-dsi-h-pulse-width = <24>;	Customize it based on your panel specification
qcom,mdss-dsi-h-sync-skew	Horizontal sync skew value. 0 = default value.	qcom,mdss-dsi-h-sync-skew = <0>;	Keep unchanged
qcom,mdss-dsi-v-back-porch	Vertical back porch value in pixel.	qcom,mdss-dsi-v-back-porch = <20>;	Customize it based on your panel specification
qcom,mdss-dsi-v-front-porch	Vertical front porch value in pixel.	qcom,mdss-dsi-v-front-porch = <8>;	Customize it based on your panel specification
qcom,mdss-dsi-v-pulse-width	Vertical pulse width.	qcom,mdss-dsi-v-pulse-width = <4>;	Customize it based on your panel specification
qcom,mdss-dsi-h-left-border	Horizontal left border in pixel. 0 = default value	qcom,mdss-dsi-h-left-border = <0>;	Keep unchanged
qcom,mdss-dsi-h-right-border	Horizontal right border in pixel. 0 = default value	qcom,mdss-dsi-h-right-border = <0>;	Keep unchanged
qcom,mdss-dsi-v-top-border	Vertical top border in pixel. 0 = default value	qcom,mdss-dsi-v-top-border = <0>;	Keep unchanged
qcom,mdss-dsi-v-bottom-border	Vertical bottom border in pixel. 0 = default value	qcom,mdss-dsi-v-bottom-border = <0>;	Keep unchanged

qcom,mdss-dsi-border-color	Defines the border color value if border is present. 0 = default value.	qcom,mdss-dsi-border-color = <0>;	Keep unchanged
qcom,mdss-dsi-panel-timings	An array of length 12 that specifies the PHY timing settings for the panel.	qcom,mdss-dsi-panel-timings = [92 1A 12 00 3E 42 16 1E 14 03 04 00];	Keep unchanged when you try to bring up this panel at the beginning, and create case for help for fine tuning parameters if the panel failed to bring up
qcom,mdss-dsi-virtual-channel-id	Specifies the virtual channel identifier. 0 = default value.	qcom,mdss-dsi-virtual-channel-id = <0>;	Keep unchanged
qcom,mdss-dsi-stream	Specifies the packet stream to be used. 0 = stream 0 (default) 1 = stream 1	qcom,mdss-dsi-stream = <0>;	Keep unchanged
qcom,mdss-dsi-on-command	An array of variable length that lists the init commands of the panel. Please refer dsi_cmd_desc structure definition at \kernel\drivers\video\msm\mdss\mdss_dsi.h for the format. A byte stream formed by multiple dcs packets base on qcom dsi controller protocol. byte 0: dcs data type byte 1: set to indicate this is an individual packet(no chain) byte 2: virtual channel number byte 3: expect ack from client (dcs read command) byte 4: wait number of specified ms after dcs command transmitted byte 5, 6: 16 bits length in network byte order byte 7 and beyond: number byte of payload	qcom,mdss-dsi-on-command = [32 01 00 00 00 00 02 00 00 29 01 00 00 10 00 02 FF 99];	Customize it based on your panel specification
qcom,mdss-dsi-off-command	An array of variable length that lists the panel off commands.	qcom,mdss-dsi-off-command = [05 01 00 00 32 02 28 00 05 01 00 00 78 02 10 00];	customize it based on your panel specification
qcom,mdss-dsi-on-command-state	String that specifies the ctrl state for sending ON commands. "dsi_lp_mode" = DSI low power mode (default) "dsi_hs_mode" = DSI high speed mode	qcom,mdss-dsi-on-command-state = "dsi_lp_mode";	Customize it based on your panel specification

qcom,mdss-dsi-off-command-state	String that specifies the ctrl state for sending OFF commands. "dsi_lp_mode" = DSI low power mode (default) "dsi_hs_mode" = DSI high speed mode	qcom,mdss-dsi-off-command-state = "dsi_hs_mode";	Customize it based on your panel specification
qcom,mdss-dsi-color-order	Specifies the R, G and B channel ordering. "rgb_swap_rgb" = DSI_RGB_SWAP_RGB (default value) "rgb_swap_rbg" = DSI_RGB_SWAP_RBG "rgb_swap_brg" = DSI_RGB_SWAP_BRG "rgb_swap_grb" = DSI_RGB_SWAP_GRB "rgb_swap_gbr" = DSI_RGB_SWAP_GBR	qcom,mdss-dsi-color-order = "rgb_swap_rgb";	Customize it based on your panel specification
qcom,mdss-dsi-underflow-color	Specifies the controller settings for the panel under flow color. 0xff = default value.	qcom,mdss-dsi-underflow-color = <0xff>;	Keep unchanged
qcom,mdss-dsi-h-sync-pulse	Specifies the pulse mode option for the panel. 0 = Do not send hsa/he following vs/ve packet(default) 1 = Send hsa/he following vs/ve packet	qcom,mdss-dsi-h-sync-pulse = <1>;	Customize it based on your panel specification
qcom,mdss-dsi-hfp-power-mode	Boolean to determine DSI lane state during horizontal front porch (HFP) blanking period	qcom,mdss-dsi-hfp-power-mode;	Customize it based on your panel specification
qcom,mdss-dsi-hbp-power-mode	Boolean to determine DSI lane state during horizontal back porch (HBP) blanking period.	qcom,mdss-dsi-hbp-power-mode;	Customize it based on your panel specification
qcom,mdss-dsi-hsa-power-mode	Boolean to determine DSI lane state during horizontal sync active (HSA) mode.	qcom,mdss-dsi-hsa-power-mode;	Customize it based on your panel specification
qcom,mdss-dsi-bllp-eof-power-mode	Boolean to determine DSI lane state during blanking low power period (BLLP) EOF mode.	qcom,mdss-dsi-bllp-eof-power-mode;	Customize it based on your panel specification
qcom,mdss-dsi-bllp-power-mode	Boolean to determine DSI lane state during blanking low power period (BLLP) mode.	qcom,mdss-dsi-bllp-power-mode;	Customize it based on your panel specification

qcom,mdss-dsi-traffic-mode	Specifies the panel traffic mode. "non_burst_sync_pulse" = non burst with sync pulses (default). "non_burst_sync_event" = non burst with sync start event. "burst_mode" = burst mode.	qcom,mdss-dsi-traffic-mode = "burst_mode";	Customize it based on your panel specification
qcom,mdss-dsi-lane-0-state	Boolean that specifies whether data lane 0 is enabled.	qcom,mdss-dsi-lane-0-state;	Customize it based on your panel specification
qcom,mdss-dsi-lane-1-state	Boolean that specifies whether data lane 1 is enabled.	qcom,mdss-dsi-lane-1-state;	Customize it based on your panel specification
qcom,mdss-dsi-lane-2-state	Boolean that specifies whether data lane 2 is enabled.	qcom,mdss-dsi-lane-2-state;	Customize it based on your panel specification
qcom,mdss-dsi-lane-3-state	Boolean that specifies whether data lane 3 is enabled.	qcom,mdss-dsi-lane-3-state;	Customize it based on your panel specification
qcom,mdss-dsi-t-clk-post	Specifies the byte clock cycles after mode switch.	qcom,mdss-dsi-t-clk-post = <0x04>;	Keep unchanged when you try to bring up this panel at the beginning, and create case for help for fine tuning parameters if the panel failed to bring up
qcom,mdss-dsi-t-clk-pre	Specifies the byte clock cycles before mode switch.	qcom,mdss-dsi-t-clk-pre = <0x1C>;	Keep unchanged when you try to bring up this panel at the beginning, and create case for help for fine tuning parameters if the panel failed to bring up
qcom,mdss-dsi-dma-trigger	Specifies the trigger mechanism to be used for DMA path. "none" = no trigger "trigger_te" = Tear check signal line used for trigger "trigger_sw" = Triggered by software (default) "trigger_sw_seof" = Software trigger and start/end of frame trigger. "trigger_sw_te" = Software trigger and TE	qcom,mdss-dsi-dma-trigger = "trigger_sw";	Keep unchanged
qcom,mdss-dsi-mdp-trigger	Specifies the trigger mechanism to be used for MDP path. "none" = no trigger "trigger_te" = Tear check signal line used for trigger "trigger_sw" = Triggered by software (default)	qcom,mdss-dsi-mdp-trigger = "none";	Keep unchanged

	"trigger_sw_te" = Software trigger and TE		
qcom,mdss-dsi-bl-pmic-control-type	A string that specifies the implementation of backlight control for this panel. "bl_ctrl_pwm" = Backlight controlled by PWM gpio. "bl_ctrl_wled" = Backlight controlled by WLED. "bl_ctrl_dcs" = Backlight controlled by DCS commands. other: Unknown backlight control. (default)	qcom,mdss-dsi-bl-pmic-control-type = "bl_ctrl_wled";	Customize it based on your panel specification
qcom,mdss-dsi-reset-sequence	An array that lists the sequence of reset gpio values and sleeps Each command will have the format defined as below: --> Reset GPIO value --> Sleep value (in ms)	qcom,mdss-dsi-reset-sequence = <1 2>, <0 10>, <1 10>;	Customize it based on your panel specification
qcom,mdss-dsi-bl-min-level	Specifies the min backlight level supported by the panel. 0 = default value.	qcom,mdss-dsi-bl-min-level = <1>;	Keep unchanged
qcom,mdss-dsi-bl-max-level	Specifies the max backlight level supported by the panel. 255 = default value.	qcom,mdss-dsi-bl-max-level = <255>;	Keep unchanged
qcom,mdss-dsi-lane-map	Specifies the data lane swap configuration. "lane_map_0123" = <0 1 2 3> (default value) "lane_map_3012" = <3 0 1 2> "lane_map_2301" = <2 3 0 1> "lane_map_1230" = <1 2 3 0> "lane_map_0321" = <0 3 2 1> "lane_map_1032" = <1 0 3 2> "lane_map_2103" = <2 1 0 3> "lane_map_3210" = <3 2 1 0>	qcom,mdss-dsi-lane-map = "lane_map_0123";	Customize it based on your panel specification

9.7.2.2 Sample

QRD8916 phone adopts Innolux(otm1283a) 720p video mode panel, the reference display configuration files is kernel/arch/arm/boot/dts/qcom/dsi-panel-innolux-720p-video.dtsi.

The sample codes are copied below:

```
&mdss_mdp {
    dsi_innolux_720p_video: qcom,mdss_dsi_innolux_720p_video {
        qcom,mdss-dsi-panel-name = "innolux(otm1283a) 720p video mode dsi
panel";
        qcom,mdss-dsi-panel-controller = <&mdss_dsi0>;
```

```

1      qcom,mdss-dsi-panel-type = "dsi_video_mode";
2      qcom,mdss-dsi-panel-destination = "display_1";
3      qcom,mdss-dsi-panel-framerate = <60>;
4      qcom,mdss-dsi-virtual-channel-id = <0>;
5      qcom,mdss-dsi-stream = <0>;
6      qcom,mdss-dsi-panel-width = <720>;
7      qcom,mdss-dsi-panel-height = <1280>;
8      qcom,mdss-dsi-h-front-porch = <52>;
9      qcom,mdss-dsi-h-back-porch = <100>;
10     qcom,mdss-dsi-h-pulse-width = <24>;
11     qcom,mdss-dsi-h-sync-skew = <0>;
12     qcom,mdss-dsi-v-back-porch = <20>;
13     qcom,mdss-dsi-v-front-porch = <8>;
14     qcom,mdss-dsi-v-pulse-width = <4>;
15     qcom,mdss-dsi-h-left-border = <0>;
16     qcom,mdss-dsi-h-right-border = <0>;
17     qcom,mdss-dsi-v-top-border = <0>;
18     qcom,mdss-dsi-v-bottom-border = <0>;
19     qcom,mdss-dsi-bpp = <24>;
20     qcom,mdss-dsi-color-order = "rgb_swap_rgb";
21     qcom,mdss-dsi-underflow-color = <0xff>;
22     qcom,mdss-dsi-border-color = <0>;
23     qcom,mdss-dsi-on-command = [29 01 00 00 00 00 02 00 00
24         29 01 00 00 00 00 04 ff 12 83 01
25         29 01 00 00 00 00 02 00 80
26         29 01 00 00 00 00 03 ff 12 83
27         29 01 00 00 00 00 02 00 92
28         29 01 00 00 00 00 02 ff 30
29         29 01 00 00 00 00 02 00 93
30         29 01 00 00 00 00 02 ff 02
31         29 01 00 00 00 00 02 00 80
32         29 01 00 00 00 00 0a c0 00 64 00 10 10 00 64 10 10
33         29 01 00 00 00 00 02 00 90
34         29 01 00 00 00 00 07 c0 00 5c 00 01 00 04
35         29 01 00 00 00 00 02 00 b3
36         29 01 00 00 00 00 03 c0 00 50
37         29 01 00 00 00 00 02 00 81
38         29 01 00 00 00 00 02 c1 55
39         29 01 00 00 00 00 02 00 90
40         29 01 00 00 00 00 02 c4 49
41         29 01 00 00 00 00 02 00 e0
42         29 01 00 00 00 00 02 c0 c8
43         29 01 00 00 00 00 02 00 a0

```

1		29 01 00 00 00 00 0f c4 05 10 04 02 05 15 11 05 10 07 02 05 15
2	11	
3		29 01 00 00 00 00 02 00 b0
4		29 01 00 00 00 00 03 c4 00 00
5		29 01 00 00 00 00 02 00 91
6		29 01 00 00 00 00 03 c5 a6 d0
7		29 01 00 00 00 00 02 00 00
8		29 01 00 00 00 00 03 d8 c7 c7
9		29 01 00 00 00 00 02 00 00
10		29 01 00 00 00 00 02 d9 87
11		29 01 00 00 00 00 02 00 b0
12		29 01 00 00 00 00 03 c5 04 38
13		29 01 00 00 00 00 02 00 bb
14		29 01 00 00 00 00 02 c5 80
15		29 01 00 00 00 00 02 00 82
16		29 01 00 00 00 00 02 c4 02
17		29 01 00 00 00 00 02 00 c6
18		29 01 00 00 00 00 02 b0 03
19		29 01 00 00 00 00 02 00 00
20		29 01 00 00 00 00 02 d0 40
21		29 01 00 00 00 00 02 00 00
22		29 01 00 00 00 00 03 d1 00 00
23		29 01 00 00 00 00 02 00 b2
24		29 01 00 00 00 00 03 f5 00 00
25		29 01 00 00 00 00 02 00 b4
26		29 01 00 00 00 00 03 f5 00 00
27		29 01 00 00 00 00 02 00 b6
28		29 01 00 00 00 00 03 f5 00 00
29		29 01 00 00 00 00 02 00 b8
30		29 01 00 00 00 00 03 f5 00 00
31		29 01 00 00 00 00 02 00 94
32		29 01 00 00 00 00 02 f5 02
33		29 01 00 00 00 00 02 00 ba
34		29 01 00 00 00 00 02 f5 03
35		29 01 00 00 00 00 02 00 b4
36		29 01 00 00 00 00 02 c5 c0
37		29 01 00 00 00 00 02 00 90
38		29 01 00 00 00 00 05 f5 02 11 02 11
39		29 01 00 00 00 00 02 00 90
40		29 01 00 00 00 00 02 c5 50
41		29 01 00 00 00 00 02 00 94
42		29 01 00 00 00 00 02 c5 66
43		29 01 00 00 00 00 02 00 80
44		29 01 00 00 00 00 0c cb 00 00 00 00 00 00 00 00 00 00

1		29 01 00 00 00 00 02 00 90
2		29 01 00 00 00 00 10 cb 00 00 00 00 00 00 00 00 00 00 00 00 00
3	00 00	
4		29 01 00 00 00 00 02 00 a0
5		29 01 00 00 00 00 10 cb 00 00 00 00 00 00 00 00 00 00 00 00 00
6	00 00	
7		29 01 00 00 00 00 02 00 b0
8		29 01 00 00 00 00 10 cb 00 00 00 00 00 00 00 00 00 00 00 00 00
9	00 00	
10		29 01 00 00 00 00 02 00 c0
11		29 01 00 00 00 00 10 cb 05 05 05 05 05 05 05 05 05 00 05 00 00
12	00 00	
13		29 01 00 00 00 00 02 00 d0
14		29 01 00 00 00 00 10 cb 00 00 00 00 05 00 00 05 05 05 05 05 05
15	05 05	
16		29 01 00 00 00 00 02 00 e0
17		29 01 00 00 00 00 0f cb 05 00 05 00 00 00 00 00 00 00 00 05 00
18	00	
19		29 01 00 00 00 00 02 00 f0
20		29 01 00 00 00 00 0c cb ff ff ff ff ff ff ff ff ff ff ff
21		29 01 00 00 00 00 02 00 80
22		29 01 00 00 00 00 10 cc 29 2a 0a 0c 0e 10 12 14 06 00 08 00 00
23	00 00	
24		29 01 00 00 00 00 02 00 90
25		29 01 00 00 00 00 10 cc 00 00 00 00 02 00 00 29 2a 09 0b 0d 0f
26	11 13	
27		29 01 00 00 00 00 02 00 a0
28		29 01 00 00 00 00 0f cc 05 00 07 00 00 00 00 00 00 00 00 01 00
29	00	
30		29 01 00 00 00 00 02 00 b0
31		29 01 00 00 00 00 10 cc 29 2a 13 11 0f 0d 0b 09 01 00 07 00 00
32	00 00	
33		29 01 00 00 00 00 02 00 c0
34		29 01 00 00 00 00 10 cc 00 00 00 00 05 00 00 29 2a 14 12 10 0e
35	0c 0a	
36		29 01 00 00 00 00 02 00 d0
37		29 01 00 00 00 00 0f cc 02 00 08 00 00 00 00 00 00 00 00 06 00
38	00	
39		29 01 00 00 00 00 02 00 80
40		29 01 00 00 00 00 0d ce 89 05 10 88 05 10 00 00 00 00 00 00
41		29 01 00 00 00 00 02 00 90
42		29 01 00 00 00 00 0f ce 54 fc 10 54 fd 10 55 00 10 55 01 10 00
43	00	
44		29 01 00 00 00 00 02 00 a0
45		29 01 00 00 00 00 0f ce 58 07 04 fc 00 10 00 58 06 04 fd 00 10
46	00	

```

1          29 01 00 00 00 00 02 00 b0
2          29 01 00 00 00 00 0f ce 58 05 04 fe 00 10 00 58 04 04 ff 00 10
3      00
4          29 01 00 00 00 00 02 00 c0
5          29 01 00 00 00 00 0f ce 58 03 05 00 00 10 00 58 02 05 01 00 10
6      00
7          29 01 00 00 00 00 02 00 d0
8          29 01 00 00 00 00 0f ce 58 01 05 02 00 10 00 58 00 05 03 00 10
9      00
10         29 01 00 00 00 00 02 00 80
11         29 01 00 00 00 00 0f cf 50 00 05 04 00 10 00 50 01 05 05 00 10
12     00
13         29 01 00 00 00 00 02 00 90
14         29 01 00 00 00 00 0f cf 50 02 05 06 00 10 00 50 03 05 07 00 10
15     00
16         29 01 00 00 00 00 02 00 a0
17         29 01 00 00 00 00 0f cf 00 00 00 00 00 00 00 00 00 00 00 00
18     00
19         29 01 00 00 00 00 02 00 b0
20         29 01 00 00 00 00 0f cf 00 00 00 00 00 00 00 00 00 00 00 00
21     00
22         29 01 00 00 00 00 02 00 c0
23         29 01 00 00 00 00 0c cf 39 39 20 20 00 00 01 01 20 00 00
24         29 01 00 00 00 00 02 00 00
25         29 01 00 00 00 00 11 e1 02 12 18 0e 07 0f 0b 09 04 07 0e 08 0f
26     12 0c 08
27         29 01 00 00 00 00 02 00 00
28         29 01 00 00 00 00 11 e2 02 12 18 0e 07 10 0b 0a 04 08 0e 08 0f
29     12 0c 08
30         29 01 00 00 00 00 02 00 b5
31         29 01 00 00 00 00 07 c5 0b 95 ff 0b 95 ff
32         29 01 00 00 00 00 02 00 00
33         29 01 00 00 00 00 04 ff ff ff ff
34         29 01 00 00 78 00 02 11 00
35         29 01 00 00 00 00 02 29 00
36         29 01 00 00 00 00 02 35 00
37         29 01 00 00 00 00 02 51 7f
38         29 01 00 00 00 00 02 53 2c];
39     qcom,mdss-dsi-off-command = [05 01 00 00 00 00 02 28 00
40         05 01 00 00 78 00 02 10 00];
41     qcom,mdss-dsi-on-command-state = "dsi_lp_mode";
42     qcom,mdss-dsi-off-command-state = "dsi_hs_mode";
43     qcom,mdss-dsi-h-sync-pulse = <1>;
44     qcom,mdss-dsi-traffic-mode = "burst_mode";
45     qcom,mdss-dsi-lane-map = "lane_map_0123";
46     qcom,mdss-dsi-bl1p-eof-power-mode;

```

```

1      qcom,mdss-dsi-bl-lp-power-mode;
2      qcom,mdss-dsi-lane-0-state;
3      qcom,mdss-dsi-lane-1-state;
4      qcom,mdss-dsi-lane-2-state;
5      qcom,mdss-dsi-lane-3-state;
6      qcom,mdss-dsi-panel-timings = [92 1A 12 00 3E 42 16 1E 14 03 04 00];
7      qcom,mdss-dsi-t-clk-post = <0x04>;
8      qcom,mdss-dsi-t-clk-pre = <0x1C>;
9      qcom,mdss-dsi-bl-min-level = <1>;
10     qcom,mdss-dsi-bl-max-level = <255>;
11     qcom,mdss-dsi-dma-trigger = "trigger_sw";
12     qcom,mdss-dsi-mdp-trigger = "none";
13     qcom,mdss-dsi-bl-pmic-control-type = "bl_ctrl_dcs";
14     qcom,mdss-dsi-reset-sequence = <1 1>, <0 50>, <1 50>;
15 };
16 };

```

The display configuration is included into super MSM8916 device tree with the following codes snippets copied from MSM8916-qrd.dts.

```

21     #include "dsi-panel-innolux-720p-video.dtsi"
22
23     &mdss_mdp {
24         qcom,mdss-pref-prim-intf = "dsi";
25     };
26
27     &dsi_innolux_720p_video {
28         qcom,mdss-dsi-bl-pmic-control-type = "bl_ctrl_dcs";
29     };
30
31     &mdss_pinmux {
32         qcom,num-grp-pins = <3>;
33         qcom,pins = <&gp 32>, <&gp 25>, <&gp 97>;
34     };
35
36     &mdss_dsi0 {
37         qcom,dsi-pref-prim-pan = <&dsi_innolux_720p_video>;
38         pinctrl-names = "default", "sleep";
39         pinctrl-0 = <&mdss_dsi_active>;
40         pinctrl-1 = <&mdss_dsi_suspend>;
41
42         qcom,platform-enable-gpio = <&msm_gpio 32 0>;
43         qcom,platform-reset-gpio = <&msm_gpio 25 0>;
44         qcom,platform-bklight-en-gpio = <&msm_gpio 97 0>;

```

};

9.7.2.3 Step by step

1. Create your own panel configuration device tree in kernel/arch/arm/boot/dts/qcom folder, for example, dsi-panel-<vendor>-<res>-video.dtsi.
2. Copy the content from dsi-panel-nt35590-720p-video.dtsi (golden device tree file) into your own panel device tree file, dsi-panel-<vendor>-<res>-video.dtsi.
3. Or according to *XML Tag Description For Display Module In GCDB (80-BA103-1)*.to generate the panel dtsi and panel header using parser script. Parser script is available with /device/qcom/common/display/tools/parser.pl.

4. Modify the panel vendor information, such as.

```
dsi_<vendor>_<res>_video: qcom,mdss_dsi_<vendor>_<res>_video {
    qcom,mdss-dsi-panel-name = "<vendor> <res> video mode dsi panel";
```

5. Modify the panel basic information, such as, panel resolution, V porches, H porches, bpp, based on your panel specification.

```
qcom,mdss-dsi-panel-width = <720>;
qcom,mdss-dsi-panel-height = <1280>;

qcom,mdss-dsi-h-front-porch = <52>;
qcom,mdss-dsi-h-back-porch = <100>;
qcom,mdss-dsi-h-pulse-width = <24>;

qcom,mdss-dsi-v-back-porch = <20>;
qcom,mdss-dsi-v-front-porch = <8>;
qcom,mdss-dsi-v-pulse-width = <4>;
```

```
qcom,mdss-dsi-bpp = <24>;
```

6. Modify the panel power on and power off commands sequence, based on your panel specification.

```
qcom,mdss-dsi-on-command = [05 01 00 00 78 02 11 00
    05 01 00 00 78 02 29 00];
qcom,mdss-dsi-on-command-state = "dsi_lp_mode";
qcom,mdss-dsi-off-command = [05 01 00 00 32 02 28 00
    05 01 00 00 78 02 10 00];
qcom,mdss-dsi-off-command-state = "dsi_hs_mode";
```

7. Modify the panel gpio configuration based on your hardware schematics, you may need change

```
qcom,platform-enable-gpio = <&msm_gpio 32 0>;
qcom,platform-reset-gpio = <&msm_gpio 25 0>;
qcom,platform-bklight-en-gpio = <&msm_gpio 97 0>;
```

8. we use GPIO25 for panel reset, if you change it to other GPIO, you need change arch/arm/boot/dts/qcom/msm8916-pinctrl.dtsi as well.

1 9. Add your own panel device tree into MSM8916 main device tree.

2 #include "dsi-panel-innolux-720p-video.dtsi"

3
4 &mdss_mdp {

5 qcom,mdss-pref-prim-intf = "dsi";

6 };

7
8 &dsi_innolux_720p_video {

9 qcom,mdss-dsi-bl-pmic-control-type = "bl_ctrl_dcs";

10 };

11 10. Rebuild and load your boot image, and check whether the panel can be lighted, if not, please
12 measure the signal about DSI clock lane and data lane to debug the panel driver further.

Part 3 – UI/FRWK Changes

10 Runtime Feature Control

10.1 Overview



Figure 12-1 System image component

The single image solution contains the default property-defined file and the special files which belong to each carrier. And for each carrier, it may contain the apps, libs, resources, special property, overlay resource and some scripts.

Table 12-1 System image component

Content	Used For
Props (default or special props)	Control the features on or off
apps, libs, res,	Required by special carrier
Browser_Res ,	Overlay the default resources for this carrier
scripts or some others	Switch the carrier or required by special carrier

10.2 How to work

1. Carrier config app will set flag(strNewSpec) in
/data/data/com.android.qti.featuresettings/cache/action, which carrier users want to switch.
2. Do the factory reset in recovery mode, erase data partition.
3. Reboot the phone, and loading switch script will copy new carrier apks to data partition.

10.3 How to make a system.img

10.3.1 Making a CMCC/OTA system.img

Making a CMCC & DSDA LTE mode configuration, please follow the steps below to change it from code lever:

This section describes the procedures for making a system.img and an OTA patch for different carriers.

1. Modify file for making CMCC single image

Locate the file named android.mk, change default to carrier name as needed

path: vendor/qcom/proprietary/qrdplus/Extension/config/Android.mk

#####

#Add for proper install for carrier

InstallCarrierMainCarrier := ChinaMobile

InstallCarrierOutsideCarrier := None

InstallCarrierOTACarrier := ChinaUnicom ChinaTelecom CTA

2. DSDA ADDITIONAL_FEATURE_PROPERTIES

Locate the file named vendor.prop, add feature properties as needed

Path:

vendor/qcom/proprietary/qrdplus/ChinaMobile/config/property/vendor.prop

#

ADDITIONAL_FEATURE_PROPERTIES FOR ChinaMobile SPEC

```

1      #
2      persist.radio.multisim.config=dsda
3      -----
4
5  3. Add CMCC preload apks, which can be uninstalled by end users and recovery after factory
6      reset.
7
8  Create folder cmcc in
9  vendor/qcom/proprietary/qrdplus/InternalUseOnly/Carrier/Preload/apps/cmcc
10
11 Locate the file named Android.mk, add feature properties as needed
12
13 path: vendor/qcom/proprietary/qrdplus/ChinaMobile/config/Android.mk
14
15 -----
16 #####
17
18 PRELOAD_APP := cmcc
19
20 $(shell mkdir -p $(TARGET_OUT)/vendor/ChinaMobile/data/app)
21
22 $(shell cp -rf
23 $(LOCAL_PATH)/../../../../InternalUseOnly/CarrierSpecific/PreLoadApp/apps/$(PRELOAD_APP)/*
24 \
25 $(TARGET_OUT)/vendor/ChinaMobile/data/app
26
27 -----

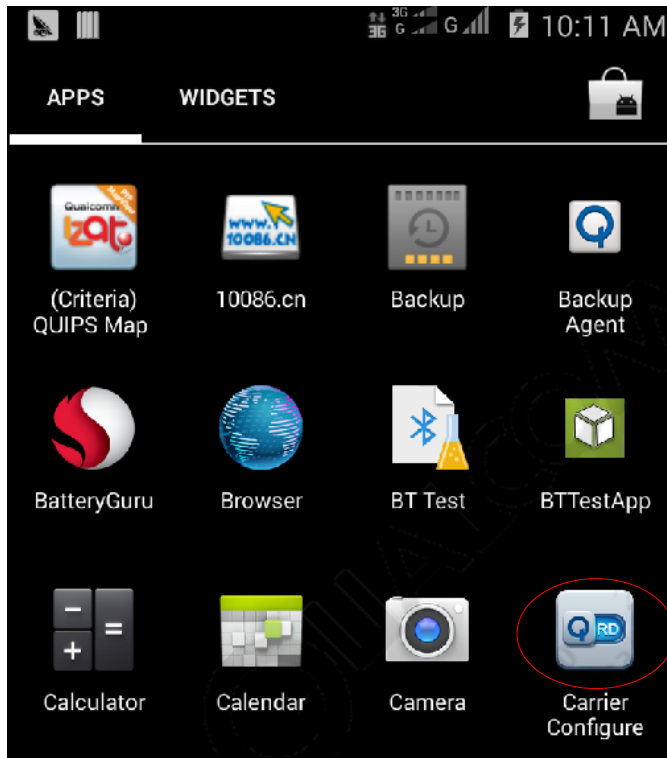
```

4. Use the CD command to enter the path where you sync the code, and input the command to make image.

5. After the image is made, you can find the folder which path like “/out/target/product/msm8226”. As every project is different, the name after the path “/out/target/product/” is different. You can find the `system.img` and OTA patch files in the path.

1 10.4 How to use the tools to switch the carrier

2 10.4.1 Changing the DSDA mode



3
4 **Figure 12-2 Carrier configure setting**

- 1 You could change the DSDA mode.



2
3 **Figure 12-3 Changing button of CMCC/DSDA**

- 4 **NOTE:** All the changes will be valid after you edit them, and when you press the back key to exit this
5 activity, it will alert you if you want to reboot the phone.

10.4.2 Switching the CMCC

You could select one carrier to switch to.



Figure 12-4 Switching the CMCC

Once you select one carrier, it will alert you if you want to switch to this carrier.

- If you press Cancel, the switch will not be implemented.
- If you press OK, it will start to switch carrier. After the switch, the phone will erase all the data from your phone's internal storage.

10.4.3 Adding the overlay resources

Sometimes the resources need overlay as required. For example, China Telecom requires that the Browser icon should be different from the default icon. In this case, we could use this solution.

10.4.3.1 Adding the overlay resources for framework

If you want to add the overlay resources for framework, you could do as follows:

1. Adding the resources to source

For all the framework resources, you need to add them under:

```
vendor/qcom/proprietary/qrdplus/[Carrier Name]/res/Frameworks
```

For example, add the overlay resource for ChinaMobile:

```
vendor/qcom/proprietary/qrplus/Chinamobile/res/Frameworks
```

2. Adding the manifest

Add the manifest to build these resources as one application. Please make sure the package is named as `android.overlay.[Carrier Name]`.

3. Adding the Android.mk

There are some rules that you need to pay attention to:

```
LOCAL_PACKAGE_NAME := [Carrier]FrameworksRes
```

```
LOCAL_MODULE_STEM := framework-res
```

```
LOCAL_MODULE_PATH :=
```

```
$(TARGET_OUT)/vendor/[Carrier]/system/vendor/overlay/framework
```

After building the source code, you could find there is one application named as `framework-res.apk` under:

```
System Out]/vendor/[Carrier]/system/vendor/overlay/framework.
```

It will be linked to

```
/system/vendor/overlay/framework/framework-res.apk
```

4. Adding the module to product

Add the package name which is defined in the `Android.mk` to `product.mk`. For each carrier, there is one `product.mk` to define all the modules belong to this carrier. So you need to add the package name to it, and make sure the application will be built into the user-release version.

10.4.3.2 Adding the overlay resources for system app

1. Adding the resources to source

For all the framework resource, you need to add them under:

```
vendor/qcom/proprietary/qrplus/[Carrier Name]/res/[OriginalModuleName]
```

For example, add the resource for the Browser of ChinaUnicom:

```
vendor/qcom/proprietary/qrplus/ChinaUnicom/res/Browser
```

2. Adding the resources to source

Add the manifest to build the resource as one application. Please make sure the package is named as:

```
[Original package name].overlay.[Carrier Name]
```

For example, add the resource for the Browser of ChinaUnicom, The package name should be:

```
com.android.browser.overlay.cu
```


3. Adding the resources to source

There are some rules that you need to pay attention to:

```
LOCAL_PACKAGE_NAME := [Carrier][Original Module Name]Res
LOCAL_MODULE_STEM := [Original Module Name]-overlay
LOCAL_MODULE_PATH :=
$(TARGET_OUT)/vendor/[Carrier]/system/vendor/overlay/system/app
```

For example, add the resource for the Browser of ChinaUnicom:

```
LOCAL_PACKAGE_NAME := CuBrowserRes
LOCAL_MODULE_STEM := Browser-overlay
LOCAL_MODULE_PATH :=
$(TARGET_OUT)/vendor/ChinaUnicom/system/vendor/overlay/system/app
```

After building the source code, you could find there is one application named as Browser-overlay.apk under:

```
[System Out]/vendor/ChinaUnicom/system/vendor/overlay/system/app
```

It will be linked to:

```
/system/vendor/overlay/system/app/Browser-overlay.apk
```

4. Adding the resources to source

Add the package name which is defined in Android.mk to product.mk.

10.4.3.3 Adding the overlay resources for data app

This step is similar to the step of adding resources for the system app. Please refer to it for details.

10.4.4 Adding related boot/shut animation resource for carrier.

Take CMCC for example:

Create the folder below and put CMCC boot animation into it
platform/vendor/qcom/proprietary/qrdplus/InternalUseOnly/CarrierSpecific/Resource/boot_animation/cmcc

- a. bootanimation.zip
- b. shutdownanimation.zip
- c. boot.wav
- d. shutdown.wav

11 CMCC DSDX New Feature

11.1 CMCC Dual SIM status bar

1. Signal bar level and LTE RSRP mapping:

CMCC requires either 0-4 or 0-5 levels of the signal bar according to the LTE RSRP signal intensity.

QRD UI/APP team plans to follow 0-4 level style which will map 0-4 signal level to <-120, [-120,-113), [-113,-105), [-105,-97), >=-97 dBm.

Table 13-1 Signal strength level

	0 格	1 格	2 格	3 格	4 格	5 格
LTE RSRP (dBm)	<-120	[-120,-115)	[-115,-110)	[-110,-105)	[-105,-97)	>=-97
	<-120	[-120,-113)	[-113,-105)	[-105,-97)	>=-97	N/A

2. LTE dual SIM signal bar style

The horizontal two bars, which indicate two slots normally.

- 1 The vertical two bars, which indicate to register two networks in one slot in SGLTE mode
 2 (the upper one for data: LTE/TD, the lower one for voice: 2G)



3
4 **Figure 13-1 SGLTE dual SIM status bar style**

5 **11.2 CMCC Dual SIM DM feature**

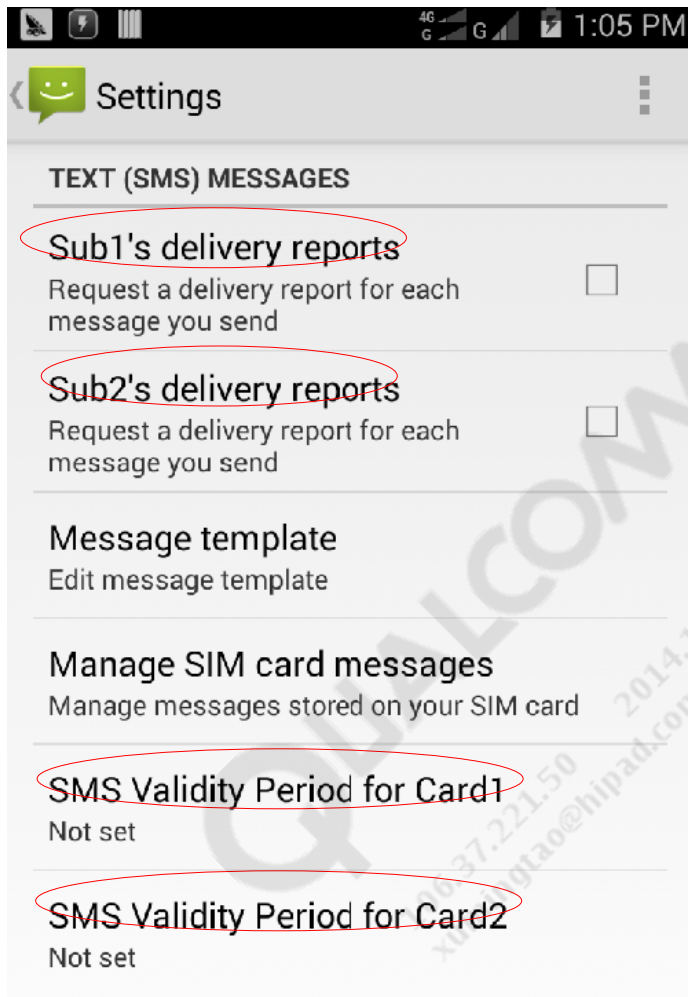
6 Insert two SIM cards with DM capability, using Slot 1 to SMS self-register by default, and no
 7 matter Slot 1 or Slot 2 has SIM card or not.

8 All scenes should be met below DM3.0 cases No.: 10.5.1; 10.5.2

9 **11.3 CMCC Dual SIM SMS/MMS support**

10 Add setting interface for Dual SIM SMS Delivery Report, storage location, validity of MMS,
 11 SMS center.

1 All scenes should be met below local function DSDA test case No.: 8.1.5.8



2
3 **Figure 13-2 Dual SIM SMS/MMS interface**
4

11.4 Supporting to close all running tasks in task management

Open task manager in status bar, click task manager icon:



Figure 13-3 Task manager icon

- Support to check the usage of memory, including free/total memory;
- Support to close one running task;
- Support to close all tasks.

All scenes should be met below local function test cases:

TS-NATIVEFUNC-TASK-BASIC-000001 ~ TS-NATIVEFUNC-TASK-BASIC-000004



Figure 13-4 Task manager behavior

11.5 Manual or automatical configuration of multi-mode dual SIM capability according to PLMN and UICC

NOTE: Only f for CSFB mode.

feature flag: persist.radio.dsdx

Path: platform/vendor/qcom-proprietary/qrdplus/ChinaMobile/config/property/vendor.prop

With CMCC tech spec: LTE/TD-SCDMA/GSM Multi-mode Terminal Technology Specification -Dual-USIM:

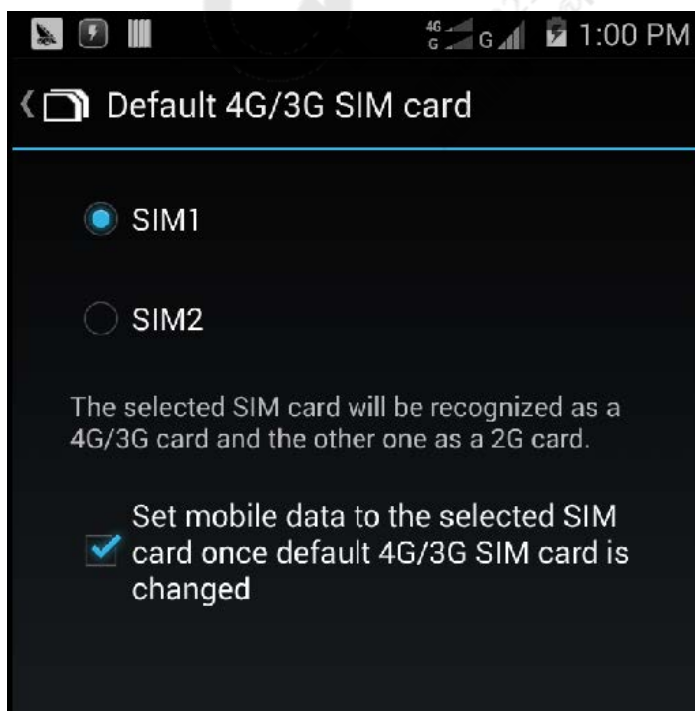
- Chapter 5.6. Slot requirement
- [Requirement No.]: TS-DS-GENRQ-SIMCS-000001

After Power on and enter into Launch, check whether the manual configuration scenario is matched. If so, prompt the Dialog to notify users that you can manually config which SIM works on 4G mode, and now which one is configed as 4G by default. Users can click it to jump into mode configuration menu to config. Otherwise it will keep remaining as current configuration.

In the Configuration Mode Menu, if manual config scenario is matched, then the menu is active to operate, otherwise, it should be disabled.



1
2
Figure 13-5 Multi-SIM configuration



3
4
Figure 13-6 Multi-SIM switch behavior

11.6 CMCC network mode

11.6.1 Change network with CMCC requirement

Default network value 20: L/T/W/G is in

vendor\qcom\proprietary\qrdplus\Extension\config\Android.mk

```

@cat "$(PRODUCT_OUT)/system/vendor/Default/"$(LocalPropFileName)" >>
"$(PRODUCT_OUT)/system/vendor/Default/"$(BuildPropFileName)"

    @for item in $(RefactorCarriers) ; do \
        echo "##### Refactor $$item build.prop
#####" ;\
        -
        cp
        "$(PRODUCT_OUT)/system/vendor/Default/"$(BuildPropFileName)"
        "$(PRODUCT_OUT)/system/vendor/$$item/"$(BuildPropFileName)" ;\
        +
        if [ "$$item" = "ChinaMobile" ]; then \
            +
            echo "ro.telephony.default_network=20" >
            "$(PRODUCT_OUT)/system/vendor/$$item/"$(BuildPropFileName)" ;\
            +
            cat
            "$(PRODUCT_OUT)/system/vendor/Default/"$(BuildPropFileName)" >
            "$(PRODUCT_OUT)/system/vendor/$$item/"$(BuildPropFileName)" ;\
            +
            fi ;\
            +
            if [ "$$item" != "ChinaMobile" ]; then \
                +
                cp
                "$(PRODUCT_OUT)/system/vendor/Default/"$(BuildPropFileName)"
                "$(PRODUCT_OUT)/system/vendor/$$item/"$(BuildPropFileName)" ;\
                +
                fi ;\
                if [ -f
                "$(PRODUCT_OUT)/system/vendor/$$item/"$(LocalPropFileName)" ] ; then \
                    echo -e "\n" >>
                    "$(PRODUCT_OUT)/system/vendor/$$item/"$(BuildPropFileName)" ;\
                    cat
                    "$(PRODUCT_OUT)/system/vendor/$$item/"$(LocalPropFileName)" >>
                    "$(PRODUCT_OUT)/system/vendor/$$item/"$(BuildPropFileName)" ;\

```

Enable Modem test Menu:

1. Put *****4636***** in dialer pad.

- 1
2. Click phone information.



2
3 **Figure 13-7 Test mode interface**

- 4
3. Choose **NETWORK_MODE_LTE** in Set network feature pull-down menu list.



Figure 13-8 Set network feature interface

- 1
- 2
- 3
4. Check modem menu list in setting->Mobile data->Preferred network type.



Figure 13-9 Modem test menu interface

NOTE: Default is NETWORK_MODE_CMCC in Set network feature pull-down menu list.



Figure 13-10 Default mobile network menu interface

12 CTA New Feature

12.1 Adding CTA dynamic overlay mechanism

Put dedicated CTA resource in the path:

vendor/qcom-proprietary/qrdplus/Extension/CTA/res.

12.2 CTA DSDX feature

UE updates the User controlled PLMN selector list, for single SIM and dual SIM.

All scenes should be met below CTA test cases No.:7.2.1& 7.2.5, UE updating the User controlled PLMN selector list for E-UTRAN.

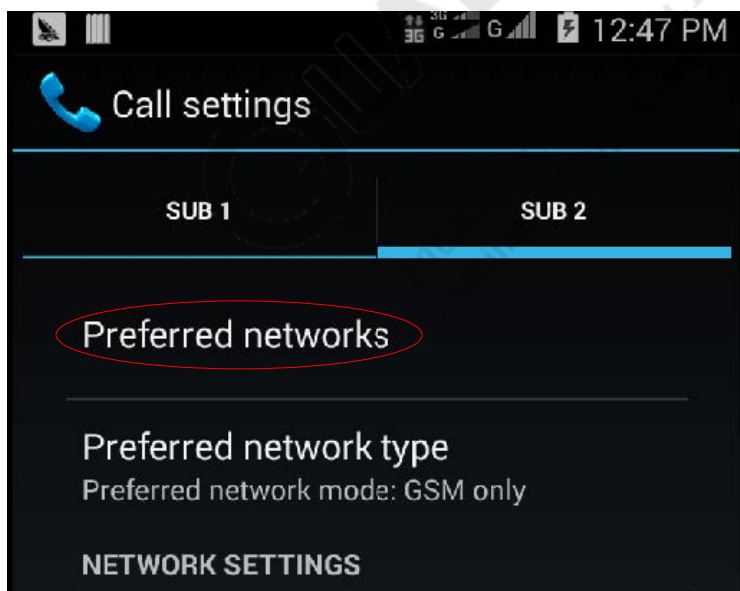


Figure 14-1 Dual SIM UPLMN interface



Figure 14-2 New a UPLMN

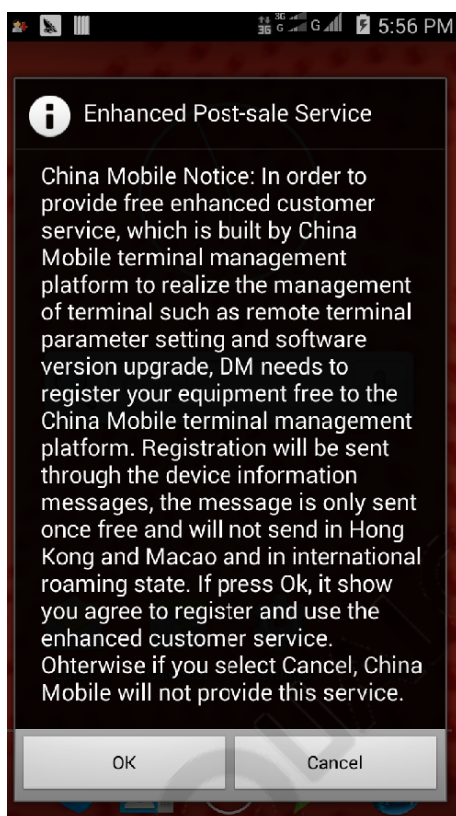
12.3 CTA security

1. DM security change

Adding DM End User notifications (MIIT security lever 1)

- a. Informing users to conform when SMS self-register.

- 1 b. Informing users to conform when open data connection.



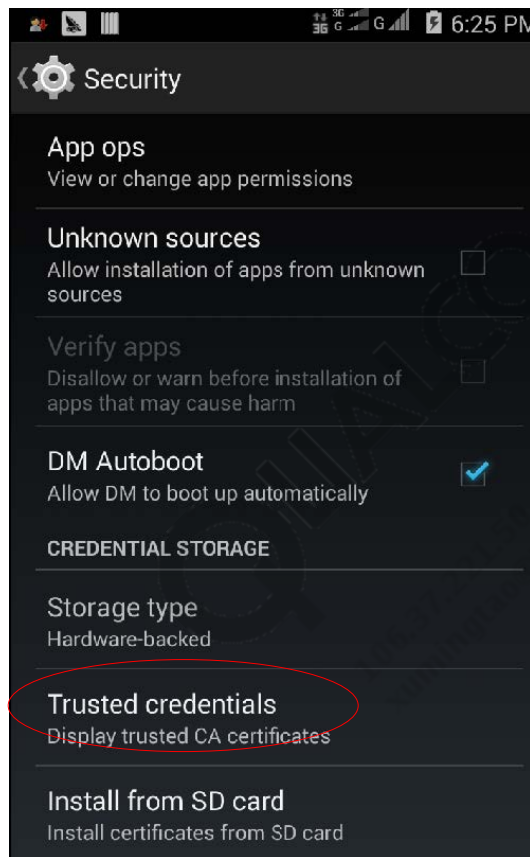
2
3 **Figure 14-3 DM End User notifications**

- 4 2. Adding DM switch in setting (MIIT security lever 3)



5
6 **Figure 14-4 DM enable/disable switch**

- 1 3. Meeting CTA security (MIIT security lever 3)
- 2 a. HW security
- 3 b. OS security
- 4 c. Peripheral interface security
- 5 d. App security
- 6 e. User data security



7
8 **Figure 14-5 CTA app security switch**

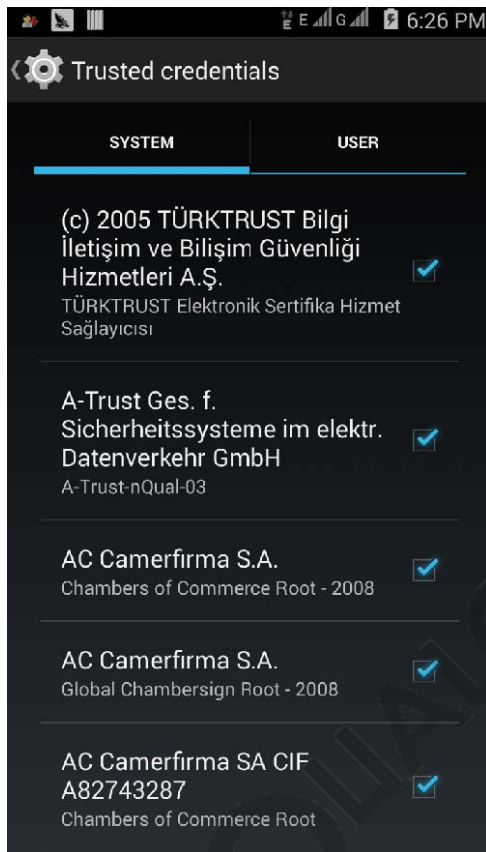


Figure 14-6 CTA app security behavior

13 SWE Browser

13.1 CAF

CAF link:

<https://www.codeaurora.org/xwiki/bin/Chromium+for+Snapdragon/Build>

13.2 To build SWE from GAF

13.2.1 Building Machine requirements

- Need to have at least 8GB swap space to successfully link the code.
- Need to be connected to the Internet.

13.2.2 Preparing the Machine to build SWE

1. Install Ubuntu 12.04. Select release appropriate to the hardware from <http://releases.ubuntu.com/12.04/>

2. Install Git on the machine.

```
sudo apt-get install git-core
```

3. Install depot_tools

```
a. git clone git://codeaurora.org/quic/chrome4sdp/chromium/  
tools/depot_tools.git
```

```
b. $ export PATH="$PATH":`pwd`/depot_tools
```

You may want to add this to your .bashrc file or your shell's equivalent, so that you don't need to reset your \$PATH manually each time you open a new shell.

4. Install build dependencies.

```
a. git clone git://codeaurora.org/quic/chrome4sdp/chromium/src.git
```

```
b. cd src
```

```
c. sudo build/install-build-deps.sh
```

```
d. sudo build/install-build-deps-android.sh
```

5. Install Java.

13.2.3 Building the code

1. Select a directory where you desire to make a build. We will call this directory <swe-root>

2. Create a file named `gclient` in `<swe-root>` directory.
3. Open the file and add the following contents to it.


```
solutions = [
    { "name"      : "src",
      "url"       :
        "git://codeaurora.org/quic/chrome4sdp/chromium/src.git@refs/remotes/o
        rigin/1599-qrd",
      "deps_file" : ".DEPS.git",
      "managed"   : True,
      "safesync_url": "",
    },
  ]
  target_os = ["android"]
```
4. `gclient sync -v -n --no-nag-max`
5. `cd src`
6. `. build/android/envsetup.sh`
7. `GYP_DEFINES="$GYP_DEFINES clang=0" gclient runhooks -v`
8. `ninja -C out/Release swe_system_package`
9. Find the built package at: `<swe-root>/src/out/Release/swe_system_package.zip`

13.2.4 Integrating the `swe_system_package` in Android Build with binary format

1. Remove existing Android Browser if already present in the Android Build Tree. We will call the root of Android Build `<android-build-root>`

```
rm -rf <android-build-root>/packages/apps/Browser
```
2. Copy the `swe_system_package.zip` which is created in Step 8 of Chapter 15.2.3 to `<android-build-root>`.
3. Extract the `swe-system-package` as follows:


```
cd <android-build-root>
unzip swe_system_package.zip -d packages/apps/Browser/
```

Follow normal Android Build Process to make the build. Browser will be part of the system image.

13.3 SWE Resource overlay

- Put related preloads bookmark icon into
- ```
Proprietary/qrdplus/ChinaMobile/apps/BrowserRes/res/raw.
```

# 14 Feature Gap of KK and JB

## 14.1 New features related quick search bar

Previously Google supports local search in quick search bar, but in android 4.4 KK, the function has been removed, now quick search bar already become a pure network search engine.

1. Supporting search app in Manager apps' setting

All scenes should be met below local function test cases: TS-NATIVEFUNC-LOCALSEARCH-SEARCH-000004

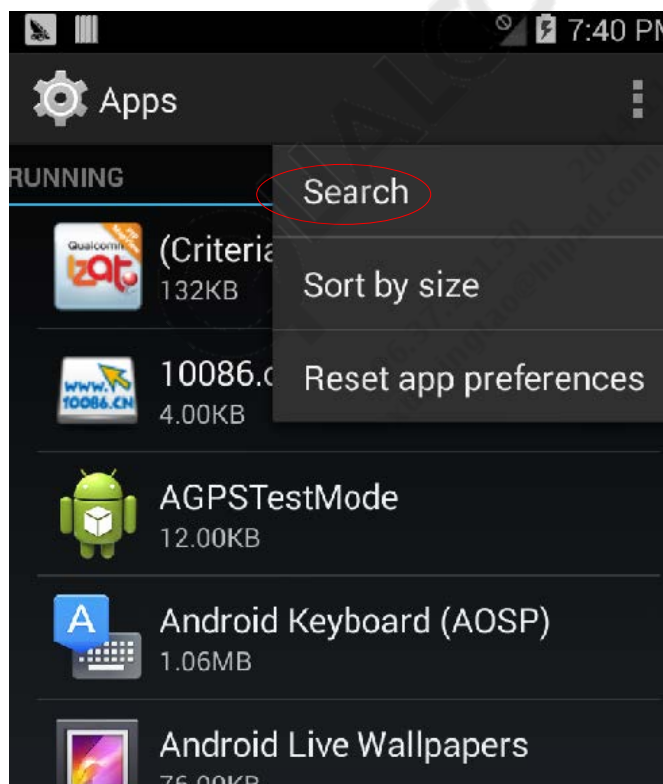


Figure 16-1 APP local search interface

2. Supporting search call log in dial pad interface.

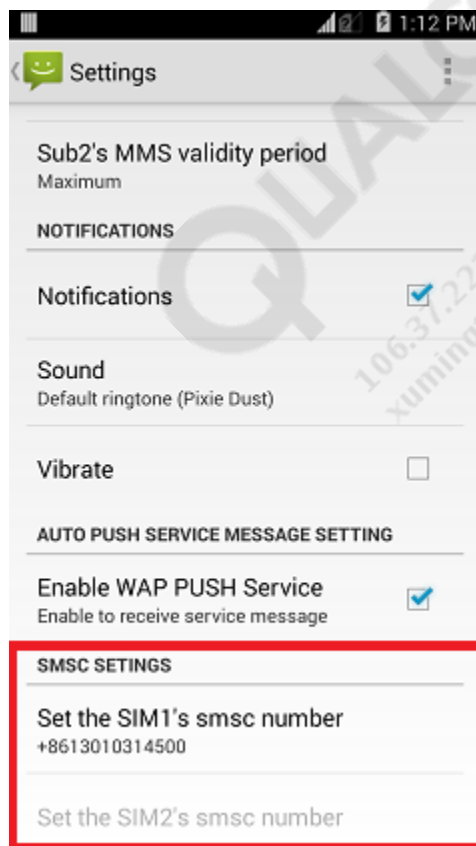
All scenes should be met below local function test cases: TS-NATIVEFUNC-LOCALSEARCH-SEARCH-000006



Figure 16-2 Call log search interface

# 15 CT New Feature

- Remove "international data roaming switch", use android data switch to turn on/off data service for both roaming and non-roaming case.
- The data switch is turned ON by default. When roaming status is changed from non-roaming to roaming and if data switch is enabled, device should notify user "The phone is in roaming. Data roaming may result in significant data charges". The switch should keep unchanged even in roaming.
- Allow user to edit the SMS center number for WCDMA/GSM mode



**Figure 17-1 SMS center number editing interface**

# 16 CU New Feature

---

Remove “**Emergency Only**” in LockScreen/Statusbar:

- If the network is available, Do not show “Emergency call only” or tips like that in LockScreen/Status bar if there is no Card in UE (Card is absent from UE), using/only show “NO Card” .
- Display "No service" when UE in out of service state.