Qualcomm
TECHNOLOGIES, INC

MSM8936/MSM8939 Linux Android Software User Manual

Software Product Document

SP80-NM846-4 H

October 9, 2014

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

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

Contents

1 Introduction 6

1.1 Conventions 6
1.2 References 6

2 Installation and Setup 8

2.1 Required equipment and software 8
2.2 Installing Ubuntu 9
2.3 Configuring Samba for Windows sharing (optional) 10
2.4 Installing JDK 10
2.5 Installing Repo 10
2.6 Installing the ARM Compiler Tools 11
2.7 Installing the Hexagon toolchain 12

3 Downloading and Building the Software 13

3.1 Downloading QTI proprietary software 15
3.2 Downloading open source HLOS software 16
3.3 Compiling the Non-HLOS software 16

4 Firmware Programming 30

4.1 Required software 30
4.2 Installing T32 30
4.3 Installing Android ADB, Fastboot, and USB driver for Windows 31
4.4 Installing adb and fastboot in Linux 34
4.5 Programming procedures 35

5 Operational Guide 46

5.1 CDP and MTP SIM slots, primary antenna, and USB ports 46
5.2 Common NV settings 50
5.3 Segment loading configuration 58
5.4 Call configuration 59
5.5 GPS configuration 67
5.6 Multimedia configuration 67
5.7 WCNSS configuration 71
5.8 Subsystem Restart (SSR) 72

6 Factory Tools 73

6.1 QDART-MFG and TPP 73
6.2 FactoryKit 73

Tables

Table 1-1 Reference documents and standards 6
Table 2-1 Required equipment and software 8
Table 3-1 Release packages 13
Table 3-2 Component release build properties 14
Table 3-3 MPSS build commands 19
Table 3-4 Build boot loaders 22
Table 3-5 Build TZ images 22
Table 3-6 RPM build commands 23
Table 3-7 WCNSS build commands 24
Table 3-8 Supported build flavors 25
Table 4-1 Required software 30
Table 5-1 RF configuration 50
Table 5-2 WCDMA/GSM + GSM DSDS NV settings 52
Table 5-3 CDMA + GSM DSDS NV Settings 52
Table 5-4 LTE + GSM DSDS settings 53

Figures

Figure 3-1 Build ID naming convention 13
Figure 3-2 Combined software release packages 14
Figure 4-1 QPST configuration 36
Figure 4-2 Emergency download port 37
Figure 4-3 Flash CDT with QPST 38
Figure 5-1 DUAL SIM slots on the baseband card 46
Figure 5-2 UIM3 on the baseband card 47
Figure 5-3 DUAL SIM slots on the baseband card 47
Figure 5-4 DUAL SIM slots (UIM1/UIM2) on the baseband card 48
Figure 5-5 CDP antenna configuration 49
Figure 5-6 CDP JTAG configuration 49
Figure 5-7 CDP USB configuration 50
Figure 5-8 QRCT NV tool for generating QCN 51

Revision history

Revision	Date	Description
A	Jun 2014	Initial release
B	Jun 2014	▪ Updated Table 3-4 to Table 3-7 – Added commands for merged MSM8916 and MSM8936/MSM8939 ▪ Added Section 5.6.1.3 to Section 5.6.1.6
C	Jul 2014	▪ Updated Table 3-3, Table 3-4, and Table 3-7 ▪ Updated Section 3.3.7 and Section 4.5.6 ▪ Added Section 3.3.9 and Section 3.3.11
D	Jul 2014	Updated Table 3-3 and Section 4.5.1
E	Aug 2014	Updated Section 3.3.7, Table 3-3, and Table 3-8
F	Oct 2014	Updated Section 3.3.5 and Table 3-4
G	Sep 2014	Updated commands in Section 3.3.2
H	Oct 2014	Updated Table 2-1 and Table 3-2.

1 Introduction

This document describes how to obtain, build, and program software applicable to the MSM8936/8939 Linux Android™ Software Product (SP) "as-is" into a reference platform including:

- Setting up a development environment and installing the software.
- Building the software and flashing it onto a reference platform.
- Configuration and bringup of call, GPS, multimedia, etc.

1.1 Conventions

Function declarations, function names, type declarations, and code samples appear in a different font, e.g., `#include`.

Code variables appear in angle brackets, e.g., `<number>`.

If you are viewing this document using a color monitor, or if you print this document to a color printer, red typeface indicates data types, blue typeface indicates attributes, and green typeface indicates system attributes.

Shading indicates content that has been added or changed in this revision of the document.

1.2 References

Reference documents are listed in Table 1-1. Reference documents that are no longer applicable are deleted from this table; therefore, reference numbers may not be sequential.

Table 1-1 Reference documents and standards

Ref.	Document	
Qualcomm Technologies		
Q2	*Hexagon Tools Installation Guide*	80-VB419-25
Q3	*Hexagon Development Tools Overview*	80-VB419-74
Q11	*USB Host Driver for Windows 2000/Windows XP User Guide*	80-V4609-1
Q12	*USB Host Driver Installation Instructions for Microsoft Windows*	80-VP092-1
Q13	*gpsOne Gen 8 Engine Family RF Development Test Procedures*	80-VM522-2
Q15	*Qualcomm ChipCode and Qualcomm ChipCenter User Guide*	80-NC193-2
Q16	*Qualcomm Flash Image Loader (QFIL)*	80-NN120-1
Q17	*Hexagon LLVM C/C++ Compiler User Guide*	80-VB419-89
Q18	*Presentation: LLVM Compiler Hexagon Processor Deployment Plan*	80-VB419-87

Ref.	Document	
Q22	*MSM8936/MSM8939 Linux Android Software Debug Manual Software Product Document*	SP80-NM846-5
Q23	*MSM8936/MSM8939 Linux Android Software Porting Manual Software Product Document*	SP80-NM846-6
Q24	*MSM8916+WTR1605 RF Calibration Software/XTT/DLL Revision Guide*	80-NK201-10
Q25	*DSI Programing Guide for B-Family Android Devices*	80-NA157-174
Q26	*Linux Android Display Driver Porting Guide*	80-NN766-1
Q27	*QRCT User Guide For WCN36X0 WLAN/BT/FM In FTM Mode*	80-WL300-27
Q28	*QDART-MFG Installer Application Note*	80-NH711-3
Q29	*QDART-MFG User Guide*	80-NF136-1
Q30	*MSM8916 External I2S interface*	80-NL239-42
Q31	*Application Note: Sensor Module Bringup*	80-NL239-32
Q32	*Linux Camera Debugging Guide*	80-NL239-33
Q33	*Application Note: Widevine DRM*	80-N9340-1
Q34	*Subsystem Restart User Guide*	80-N5609-2
Q35	*Presentation: MSM8x10 Android Subsystem Restart Overview*	80-NC839-21
Q36	*Presentation: MSM8974 Android Subsystem Restart*	80-NA157-31
Q37	*Application Note: SGLTE Device Configuration*	80-NJ017-11
Q38	*Presentation: MSM8936/MSM8939 Thermal Management Overview*	80-NM846-6
Q39	*Presentation: Security TrustZone QSEE Overview*	80-NL239-5
Q40	*Presentation: MSM8936/MSM8939 Software Architecture Overview*	80-NM846-14
Q41	*Application Note: Segment Loading Feature*	80-NL239-46
Resources		
R1	*Android Open Source Project Page*	http://source.android.com/
R2	*Android Developer Resources*	http://developer.android.com/index.html
R3	*Android Source Download and System Setup*	http://source.android.com/source/index.html
R4	*Code Aurora Forum*	https://www.codeaurora.org/
R5	*Installing Repo*	http://source.android.com/source/downloading.html
R6	*Qualcomm ChipCode Website*	https://chipcode.qti.qualcomm.com/

2 Installation and Setup

2.1 Required equipment and software

Table 2-1 identifies the equipment and software needed for a user to install and run the software.

Table 2-1 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	▪ DPM v1.0 – Hexagon tool version 6.2.09 ▪ DPM v2.0 released before 7/30/2014 – Hexagon tool version 5.03 ▪ DPM v2.0 released on or after 7/30/2014 – Hexagon tool version 6.2.09	QTI/ Gnu	Building Modem Processor Subsystem (MPSS)
8	Python	2.6.6	Python.org	Building boot, subsystem, etc. ▪ 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

2.2 Installing Ubuntu

You must be able to log in as root or use pseudo 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:
   - Using the GUI, select System→Administration→Update Manager

     Or
   - 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:
   - Reconfigure the package:
     i. Use the command:

     ```
     sudo dpkg-reconfigure dash
     ```

     ii. Answer *no*.
   - 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.

2.3 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 accnt=no guest accnt
   add share directory=/, share name=root, description=root directory
   ```

2.4 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
```

2.5 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 pseudo access, install for all system users under a common location, such as /usr/local/bin or somewhere under /opt.
2. Download the repo script.
   ```
   $ curl https://dl-ssl.google.com/dl/googlesource/git-repo/repo
   >~/bin/repo
   ```
3. Set the repo script attributes to executable.
   ```
   $ chmod a+x ~/bin/repo
   ```
4. Include the installed directory location for repo in your PATH.
   ```
   $ export PATH=~/bin:$PATH
   ```
5. 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).

2.6 Installing the ARM Compiler Tools

Building the non-HLOS images requires the specific version of the ARM Compiler Tools indicated in Section 2.1. 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.

2.7 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 [Q2] for detailed procedures to download and install the Hexagon toolchain software. See [Q3] for more documentation on using the Hexagon tools.

See [Q17] and [Q18], 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.

3 Downloading and Building the Software

Table 3-1 describes the software for this product line divided into the release packages that must be downloaded separately and combined to have complete product line software set.

Table 3-1 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 3.3.10. Each package is identified by 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 – 8x36 for MSM8936/8939

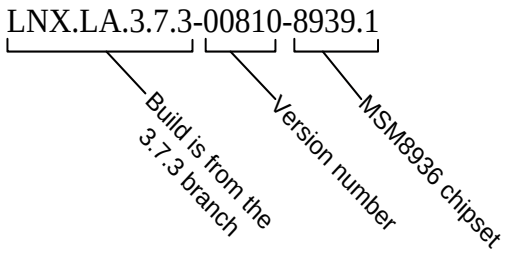


Figure 3-1 Build ID naming convention

Figure 3-2 illustrates the combined software release packages.

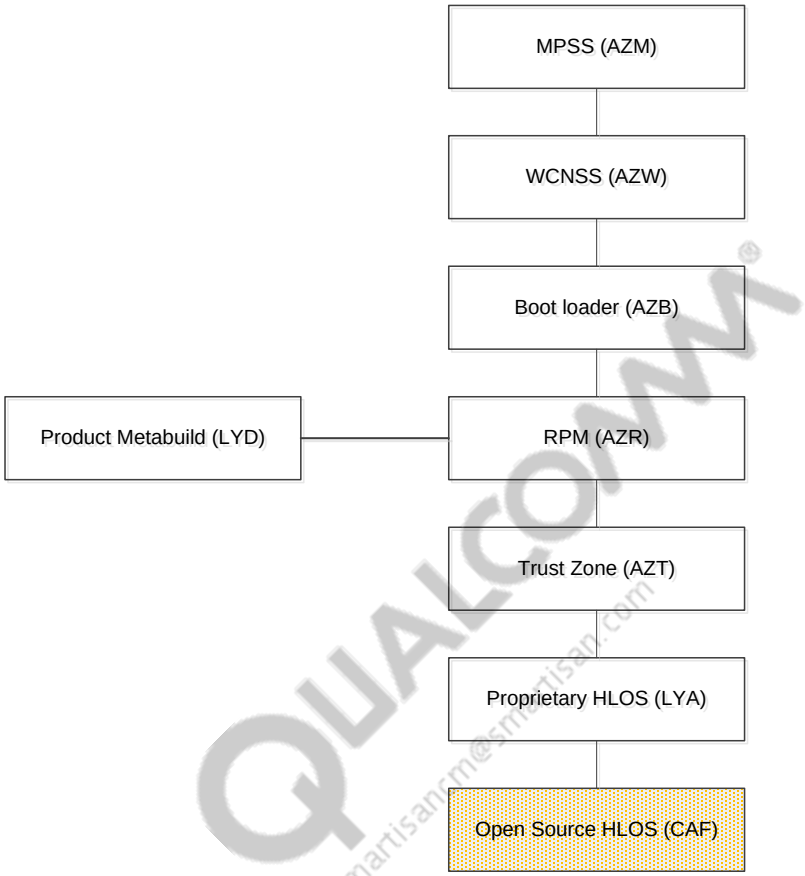


Figure 3-2 Combined software release packages

Table 3-2 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 3-2 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
DPM 1.0 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
DPM 2.0 MPSS (AZM) released before 7/30/2014	Source	Hexagon 5.03	Python 2.7.6 (64-bit)	Perl 5.10.1	Windows builds only; only needs tee.exe	Linux, Windows XP, and Windows 7

Component build release	Source or binary only	Toolchain required for building source	Python version	Perl version	Cygwin	Supported build hosts
DPM 2.0 MPSS (AZM) released before 7/30/2014	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
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

3.1 Downloading QTI proprietary software

The non-HLOS and proprietary HLOS software releases are distributed on the Qualcomm ChipCode™ website [R6]. Designated points of contact at customer sites are given access to download software for which the customer has a current license. Each customer has access to a set of distros for the chipsets for which they are licensed. For each distro, there is a Git project and the Git tree includes revisions for previous builds, allowing OEMs to diff the changes between releases via Git.

Up-to-date documentation and a set of tutorial videos available at: https://chipcode.qti.qualcomm.com/projects/help/wiki.

To make the build process go smoothly, create a top-level directory in the build PC and unzip each archive file to generate the following directory structure. In this example, <target_root> is the top-level directory.

```
<target_root>          /common/
                       /boot_images/
                       /modem_proc/
                       /rpm_proc/
                       /trustzone_images/
                       /wcnss_proc/Q
                       /LINUX/
                       contents.xml
```

NOTE: Ensure that the contents.xml file is located in the root folder as shown.

3.2 Downloading open source HLOS software

1. Find the contents.xml file for your build at chipcode.qti.qualcomm.com. This file is in the root directory of the release build downloaded from Qualcomm ChipCode. Look for the apps build ID tag for the APSS build (found under the <name>apps</name> tag). It is similar to the sample tag shown below:

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

2. Follow the instructions listed at https://www.codeaurora.org/xwiki/bin/QAEP/release (look under the Branch releases section) to find the APSS build ID in the released software. The release area lists all builds in CAF.

3. In an empty directory, issue 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
   ```

4. Type the repo sync command.

   ```
   $ repo sync
   ```

5. After the repo sync finishes, copy the vendor/qcom/proprietary directory tree from the proprietary HLOS release into the open source HLOS source tree contained in your workspace.

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

3.3 Compiling the non-HLOS software

3.3.1 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_8936
#
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
```

3.3.2 Building MPSS

To build MPSS (<target_root> is the top-level directory that is created in Section 3.1):

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 ARMTOOLS=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 described in Table 3-3.

Table 3-3 MPSS build commands

NOTE: For MPSS build flavors, refer section 3.3.7.

Build Flavor	Build environment	Build MSM8916 and MSM8939 (Builds both flavors mentioned in Column1)	Build MSM8916-only (Builds respective flavor mentioned in Column1)	Build MSM8939-only (Builds respective flavor mentioned in column 1)	Comments
FAAAANVZ – For MSM8939 EAAAANVZ – For MSM8916	Linux	Build images – ./build.sh 8936.gen.prod 8916.gen.prod -k Clean the build – ./build.sh 8936.gen.prod 8916.gen.prod -k -c	Build images – ./build.sh 8916.gen.prod -k Clean the build – ./build.sh 8916.gen.prod -k –c	Build images – ./build.sh 8936.gen.prod -k Clean the build – ./build.sh 8936.gen.prod -k –c	This build includes all RATs (W/T+G+C+L). Segment loading for W/T is enabled.
FAAAANVZ – For MSM8939 EAAAANVZ – For MSM8916	Windows	Build images – build.cmd 8936.gen.prod 8916.gen.prod -k Clean the build – build.cmd 8936.gen.prod 8916.gen.prod -k -c	Build images – build.cmd 8916.gen.prod -k Clean the build – build.cmd 8916.gen.prod -k -c	Build images – build.cmd 8936.gen.prod -k Clean the build – build.cmd 8936.gen.prod -k -c	This build includes all RATs (W/T+G+C+L). Segment loading for W/T is enabled.
FAAAANUZ – For MSM8939 EAAAANUZ – For MSM8916	Linux	Build images – ./build.sh 8936.genns.prod 8916.genns.prod -k Clean the build – ./build.sh 8936.genns.prod 8916.genns.prod -k -c	Build images – ./build.sh 8916.genns.prod -k Clean the build – ./build.sh 8916.genns.prod -k –c	Build images – ./build.sh 8936.genns.prod -k Clean the build – ./build.sh 8936.genns.prod -k –c	This build includes all RATs (W+T+G+C+L). Segment loading for W/T is disabled.
FAAAANUZ – For MSM8939 EAAAANUZ – For MSM8916	Windows	Build images – build.cmd 8936.genns.prod 8916.genns.prod -k Clean the build – build.cmd 8936.genns.prod 8916.genns.prod –k -c	Build images – build.cmd 8916.genns.prod -k Clean the build – build.cmd 8916.genns.prod -k -c	Build images – build.cmd 8936.genns.prod -k Clean the build – build.cmd 8936.genns.prod -k -c	This build includes all RATs (W+T+G+C+L). Segment loading for W/T is disabled.

Build Flavor	Build environment	Build MSM8916 and MSM8939 (Builds both flavors mentioned in Column1)	Build MSM8916-only (Builds respective flavor mentioned in Column1)	Build MSM8939-only (Builds respective flavor mentioned in column 1)	Comments
FAAAANYZ – For MSM8939 EAAAANYZ – For MSM8916	Linux	Build images – ./build.sh 8936.lwgns.prod 8916.lwgns.prod -k Clean the build – ./build.sh 8936.lwgns.prod 8916.lwgns.prod -k -c	Build images – ./build.sh 8916.lwgns.prod -k Clean the build – ./build.sh 8916.lwgns.prod -k -c	Build images – ./build.sh 8936.lwgns.prod -k Clean the build – ./build.sh 8936.lwgns.prod -k -c	This variant has 1x/DO compiled out. This build includes RATs (W+T+G+L). Segment loading for W/T is disabled.
FAAAANYZ – For MSM8939 EAAAANYZ – For MSM8916	Windows	Build images – ./build.cmd 8936.lwgns.prod 8916.lwgns.prod -k Clean the build – ./build.cmd 8936.lwgns.prod 8916.lwgns.prod -k -c	Build images – ./build.cmd 8916.lwgns.prod -k Clean the build – ./build.cmd 8916.lwgns.prod -k -c	Build images – ./build.cmd 8936.lwgns.prod -k Clean the build – ./build.cmd 8936.lwgns.prod -k -c	This variant has 1x/DO compiled out. This build includes RATs (W+T+G+L). Segment loading for W/T is disabled.

3.3.3 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 ARMTOOLS=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_armlmd_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 3.1.

3. Choose one of the build command options described Table 3-4 in based on the build environment/release.

Table 3-4 Build boot loaders

Build environment	Command
Linux	Build boot images for both 8916 and 8936. No option in boot builds to compile images for single target. $ ./build.sh TARGET_FAMILY=8936 --prod Cleaning the build: $ ./build.sh -c TARGET_FAMILY=8936 --prod Depending on your configuration, edit the script boot_images/build/ms/build_8936.sh to change if [-e “setenv.sh”]; then - source setenv.sh + source ./setenv.sh Fi
Windows	Build boot images for both 8916 and 8936. No option in boot builds to compile images for single target. build.cmd TARGET_FAMILY=8936 --prod Cleaning the build build.cmd -c TARGET_FAMILY=8936 --prod

3.3.4 Building TZ images

To build the MSM8936/8939 TZ images:

1. Navigate to the following directory:

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

2. Run the commands described in Table 3-5 to build all images.

Table 3-5 Build TZ images

Build environment	To compile MSM8916 and MSM8939	To compile MSM8916 only	To Compile MSM8939 only
Linux	cleanpack_8916_8936.sh : Cleans and Compile.	**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	**Build images** – ./build.sh CHIPSET=msm8936 tz hyp sampleapp tzbsp_no_xpu playready widevine isdbtmm securitytest keymaster commonlib **Clean the build** – ./build.sh CHIPSET=msm8936 tz hyp sampleapp tzbsp_no_xpu playready widevine isdbtmm securitytest keymaster commonlib –c

Build environment	To compile MSM8916 and MSM8939	To compile MSM8916 only	To Compile MSM8939 only
Windows	cleanpack_8916_8936.cmd : Cleans and Compile.	**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	**Build images** – build.cmd CHIPSET=msm8936 tz hyp sampleapp tzbsp_no_xpu playready widevine isdbtmm securitytest keymaster commonlib **Clean the build** – build.cmd CHIPSET=msm8936 tz hyp sampleapp tzbsp_no_xpu playready widevine isdbtmm securitytest keymaster commonlib -c

NOTE: Issues have been observed with Windows build commands provided in the release build.

3.3.5 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 3-2.

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

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

```
cd <target_root>\rpm_proc\build
```

2. Use the appropriate command from Table 3-6 based on the build environment.

Table 3-6 RPM build commands

Build environment	Build MSM8916 and MSM8939	Build MSM8916-only	Build MSM8939-only
Linux	**Build images** – ./build_8936_8916.sh **Clean the build** – ./build_8936_8916.sh -c	**Bulid images** – ./build_8916.sh **Clean the build** – ./build_8916.sh -c	**Bulid images** – ./build_8936.sh **Clean the build** – ./build_8936.sh -c
Windows	**Build images** – build_8936_8916.bat **Clean the build** – build_8936_8916.bat -c	**Build images** – build_8916.bat **Clean the build** – build_8916.bat -c	**Build images** – build_8936.bat **Clean the build** – build_8936.bat -c

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

3.3.6 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 3-2.

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

```
cd <target_root>\wcnss_proc\Pronto\bsp\build\
```

Use the appropriate command from Table 3-7 based on the build environment.

Table 3-7 WCNSS build commands

Build environment	Build MSM8916 and MSM8936	Build MSM8916-only	Build MSM8936-only
Linux	Build images – source ./wcnss_build.sh 8916 pronto && source ./wcnss_build.sh 8936 pronto	**Build images** – source ./wcnss_build.sh 8916 pronto **Clean the build** – source ./wcnss_build.sh 8916 pronto -c	**Build images** – source ./wcnss_build.sh 8936 pronto **Clean the build** – source ./wcnss_build.sh 8936 pronto -c
Windows	Build images – wcnss_build.cmd 8916 pronto && wcnss_build.cmd 8936 pronto	**Build images** – wcnss_build.cmd 8916 pronto **Clean the build** – wcnss_build.cmd 8916 pronto -c	**Build images** – wcnss_build.cmd 8936 pronto **Clean the build** – wcnss_build.cmd 8936 pronto -c

3.3.7 Update Non-HLOS.bin

Table 3-8 describes the supported build flavors.

NOTE: The content XMLs mentioned in the Table 3-8 (apart from the default contents.xml) are now placed in the common/build folder. Customer's can pick and copy the required contents file to the root folder, and rename it as contents.xml.

Table 3-8 Supported build flavors

Meta contents.xml	Images needed to be compiled for this (prerequisite)	META compilation procedure	Comments
Contents.xml (Which comes with Build By default)	MPSS: Refer Table 3-3: Row 3 or 4 (Depending on Linux or Windows). Column3 (i.e. MSM8939 and MSM8916 should be compiled). All the remaining images should be compiled for both MSM8916 and MSM8939 as mentioned in respective tables. Apps is common for MSM8916 and MSM8939.	1. Meta root directory should now have respective MPSS as in column2. 2. CD <target_root>\common\build 3. Run python update_common_info.py	Generates NON HLOS for MSM8916 and MSM8939 which supports all RATs. Segment loading for W/T disabled.
contents_VG.xml	MPSS: Refer Table 3-3: Row 1 or 2 (depending on Linux or Windows). Column 3 (i.e. MSM8939 and MSM8916 should be compiled). All the remaining images should be compiled for both MSM8916 and MSM8939 as mentioned in respective tables. Apps is common for MSM8916 and MSM8939.	1. Delete contents.xml file from Meta root directory and rename contents_VG.xml as contents.xml file 2. Meta root directory should now have respective MPSS as in column2. 3. CD <target_root>\common\build 4. Run python update_common_info.py	Generates NON HLOS for MSM8916 and MSM8939 which supports all RATs. Segment loading for W/T Enabled.
contents_VG_8939.Xml	MPSS: Refer Table 3-3: Row 1 or 2 (depending on Linux or Windows). Column 5 (i.e. MSM8939 only should be compiled). All the remaining images should be compiled for MSM8939 only.	1. Delete contents.xml file from Meta root directory and rename contents_VG_8939.Xml as contents.xml file 2. Meta root directory should now have respective MPSS as in column2. 3. CD <target_root>\common\build 4. Run python update_common_info.py	Generates NON HLOS for MSM8939 only which supports all RATs. Segment loading for W/T enabled.

Meta contents.xml	Images needed to be compiled for this (prerequisite)	META compilation procedure	Comments
contents_UG_8939.Xml	MPSS: Refer Table 3-3: Row 3 or 4 (depending on Linux or Windows). Column 5 (i.e. MSM8939 only should be compiled). All the remaining images should be compiled for MSM8939.	1. Delete contents.xml file from Meta root directory and rename contents_UG_8939.Xml as contents.xml file 2. Meta root directory should now have respective MPSS as in column2. 3. CD <target_root>\common\build 4. Run python update_common_info.py	Generates NON HLOS for MSM8939 only which supports all RATs. Segment loading for W/T disabled.
contents_YG_8939.Xml	MPSS: Refer Table 3-3: Row 5 or 6 (Depending on Linux Or windows). Column 5 (i.e. MSM8939 only should be compiled). All the remaining images should be compiled for MSM8939.	1. Delete contents.xml file from Meta root directory and rename contents_YG_8939.Xml as contents.xml file 2. Meta root directory should now have respective MPSS as in column2. 3. CD <target_root>\common\build 4. Run python update_common_info.py	Generates NON HLOS for MSM8939 only which supports RATs (W+T+G+L). 1x/DO compiled out. Segment loading for W/T disabled.
contents_VG_8916.Xml	MPSS: Refer Table 3-3: Row 1 or 2(depending on Linux or Windows). Column4(i.e. MSM8916 only should be compiled) All the remaining images should be compiled for MSM8916.	1. Delete contents.xml file from Meta root directory and rename contents_VG_8916.Xml as contents.xml file 2. Meta root directory should now have respective MPSS as in column2. 3. CD <target_root>\common\build 4. Run python update_common_info.py	Generates NON HLOS for MSM8916 only which supports all RATs. Segment loading for W/T enabled.
contents_UG_8916.Xml	MPSS: Refer Table 3-3: Row 3 or 4 (depending on Linux or Windows). Column 4 (i.e. MSM8916 only should be compiled). All the remaining images should be compiled for MSM8916.	1. Delete contents.xml file from Meta root directory and rename contents_UG_8916.Xml as contents.xml file 2. Meta root directory should now have respective MPSS as in column2. 3. CD <target_root>\common\build 4. Run python update_common_info.py	Generates NON HLOS for MSM8916 only which supports all RATs. Segment loading for W/T disabled.

Meta contents.xml	Images needed to be compiled for this (prerequisite)	META compilation procedure	Comments
contents_YG_8916.Xml	MPSS: Refer Table 3-3: Row 5 or 6 (depending on Linux or Windows). Column 4 (i.e. MSM8916 only should be compiled). All the remaining images should be compiled for MSM8916.	1. Delete contents.xml file from Meta root directory and rename contents_YG_8916.Xml as contents.xml file 2. Meta root directory should now have respective MPSS as in column2. 3. CD <target_root>\common\build 4. Run python update_common_info.py	Generates NON HLOS for MSM8916 only which supports RATs (W+T+G+L). 1x/DO compiled out. Segment loading for W/T disabled.

If any MPSS 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 3.1):

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/msm8936_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.

3.3.8 Update single images 8936_msimage.mbn and MPRG8936.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\8936\
rpm.mbn: <target_root>\rpm_proc\build\ms\bin\8936\
tz.mbn: <target_root>\trustzone_images\build\ms\bin\MAWAANAA\
```

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\8936\ --
search_path=<target_root>\rpm_proc\build\ms\bin\8936\ --
search_path=<target_root>\trustzone_images\build\ms\bin\MAWAANAA\
```

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\8936\emmcbld.mbn as 8936_msimage.mbn and MPRG8936.mbn, respectively.
5. Copy the two files to boot_images/build/ms/bin/EMMCBLD/.

3.3.9 Update 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\8916\
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\8916\ --
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\emmcbld.mbn as 8916_msimage.mbn and MPRG8916.mbn, respectively.
5. Copy the two files to boot_images/build/ms/bin/EMMCBLD/.

3.3.10 Build 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_32-userdebug (for 32-bit kernel space and 32-bit user space)
   - lunch msm8916_32_k64-userdebug (for 64-bit kernel space and 32-bit user space)

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
```

```
Single release supporting MSM8916 and MSM8939
```

4 Firmware Programming

4.1 Required software

Table 4-1 lists the software required for programming firmware images into a target device.

Table 4-1 Required software

	Item description	Version	Source/vendor	Purpose
1	QPST	2.7.405 or later	Qualcomm, Inc.	Programming firmware images using QPST
2	QXDM Professional™ (QXDM Pro)	3.14.447 or later	Qualcomm, 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 QDSP6 License Extension	LA-3741A	Lauterbach GmbH	Modem software processor, firmware processor
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, Inc.	Windows host USB drivers for Qualcomm composite devices

4.2 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.

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

Android CDP support requires the following USB device support:

- Android USB Driver (android_winusb.inf)
  - Android ADB Interface
  - Android Boot Loader Interface (fastboot)
- Qualcomm Composite USB Modem/Serial Driver (qcmdm.inf, qcser.inf)
  - Qualcomm HS-USB Android DIAG
  - Qualcomm HS-USB Android Modem
  - Qualcomm HS-USB Android GPS (NMEA)
- Qualcomm Composite USB Network Combo driver (qcnet.inf)
  - 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”
```

4.4 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 5:

   a. Type the command:

      ```
      source build/envsetup.sh
      ```

   b. Type the command:

      ```
      choosecombo 1 msm8936 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, pseudo or root access on the Linux machine may be required.

4.5 Programming procedures

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

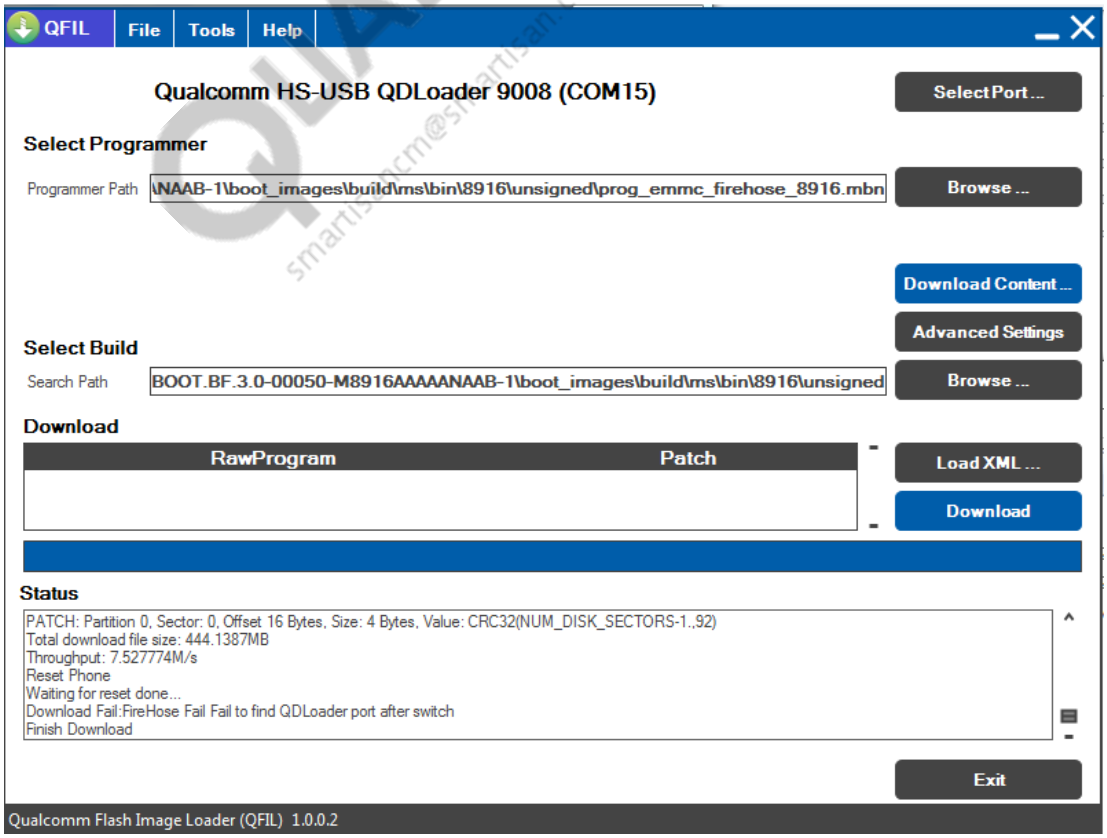
4.5.1 Programming eMMC with QPST

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 MSM8936/8939 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 4.1.

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\8936\unsigned\prog_emmc_firehose_8936.mbn.

3. Click **Download Content** and select metabuild's contents.xml.

NOTE: If the MSM8916 and MSM8939 are built with the single contents mentioned in section 3.3.7 , do not use the same content for loading the build but use the individual contents files (i.e. MSM8916 or MSM8939) for build loading using the QFIL tool.

4. Click **Download**.

NOTE: See [Q16] 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 4-1. For boards that cannot enter the Download mode, try again after installing the batteries.

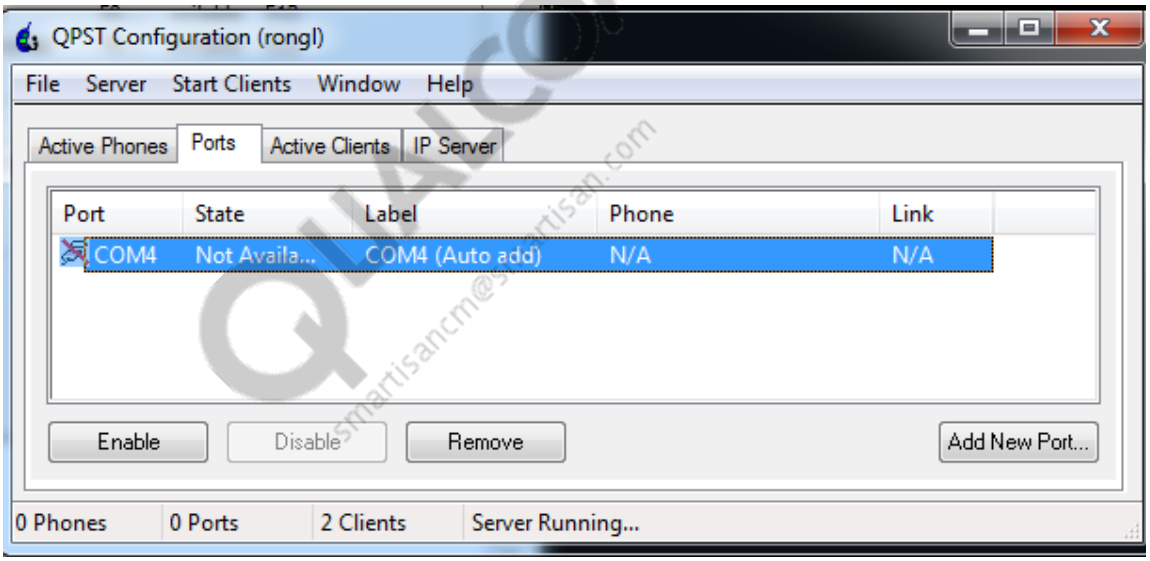


Figure 4-1 QPST configuration

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 sbl1 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 4.5.2.

4.5.2 Programming CDT using QPST

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

- Flash programmer – MPRG8936.mbn
- XML file – To load the flash programmer, use sahara.xml
- 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          = MPRG8936.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>
```

- 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

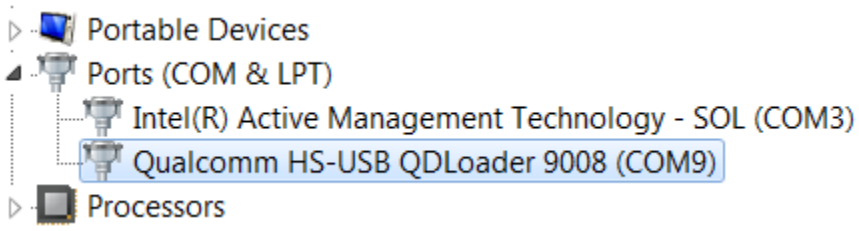


Figure 4-2 Emergency download port

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".
3. Click "Load Configuration and start download..." and select qrd_prog_cfg.xml to program the CDT as shown in Figure 4-3:

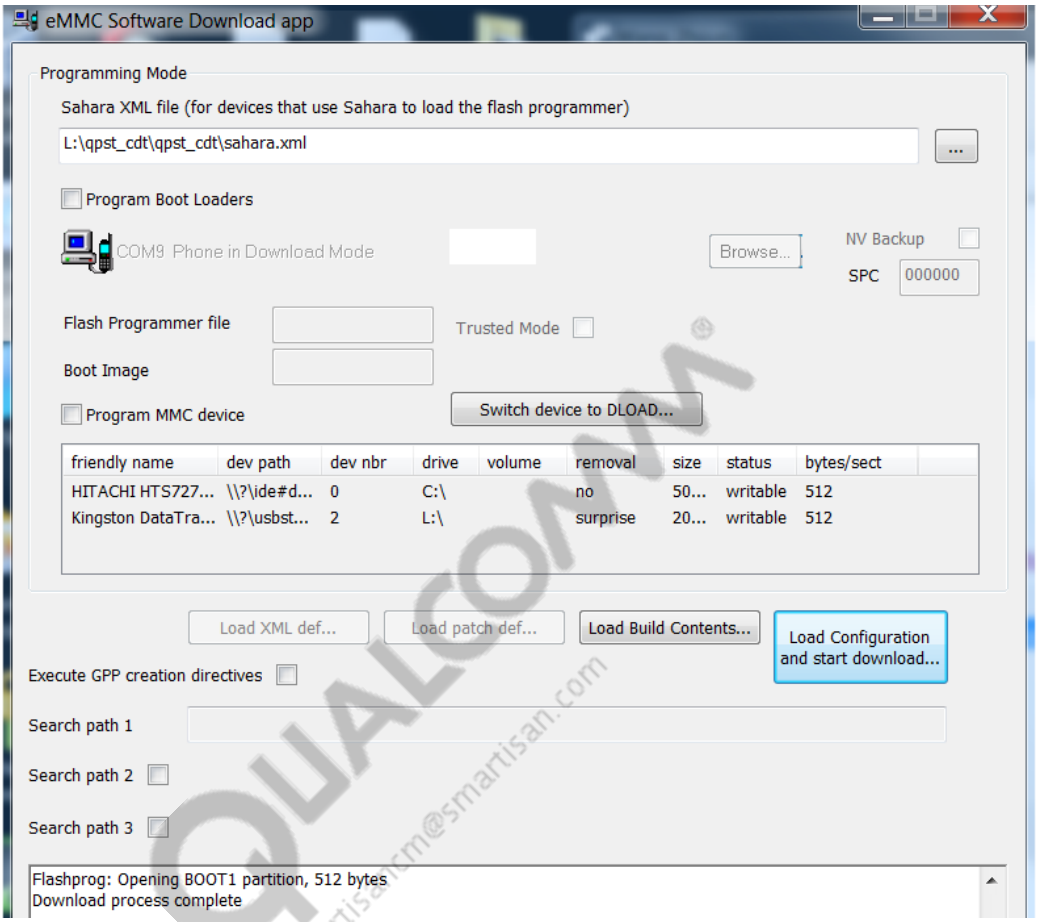
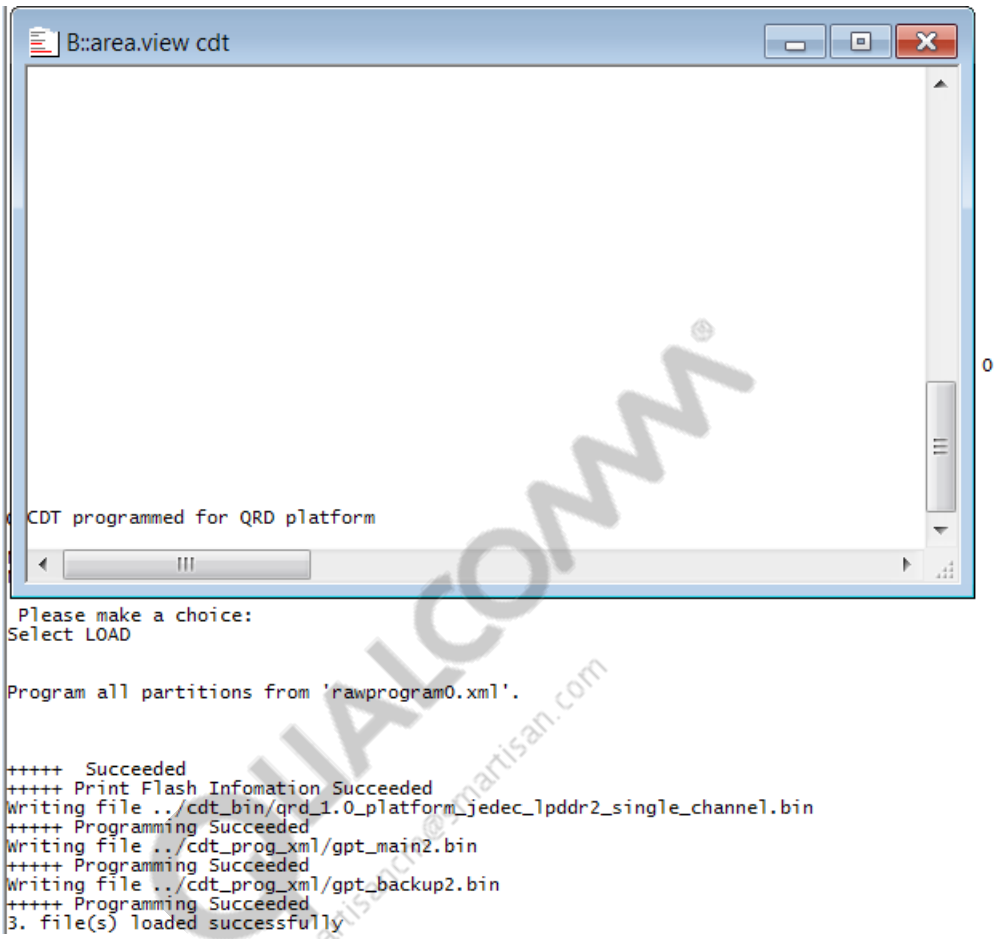


Figure 4-3 Flash CDT with QPST

4.5.3 Programming CDT using JTAG:

1. Set up T32 debug environment first (see section 4.5.4)
2. 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
```



4.5.4 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:
   - 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
     ```

     iii. 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 aboot <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.

4.5.5 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
```

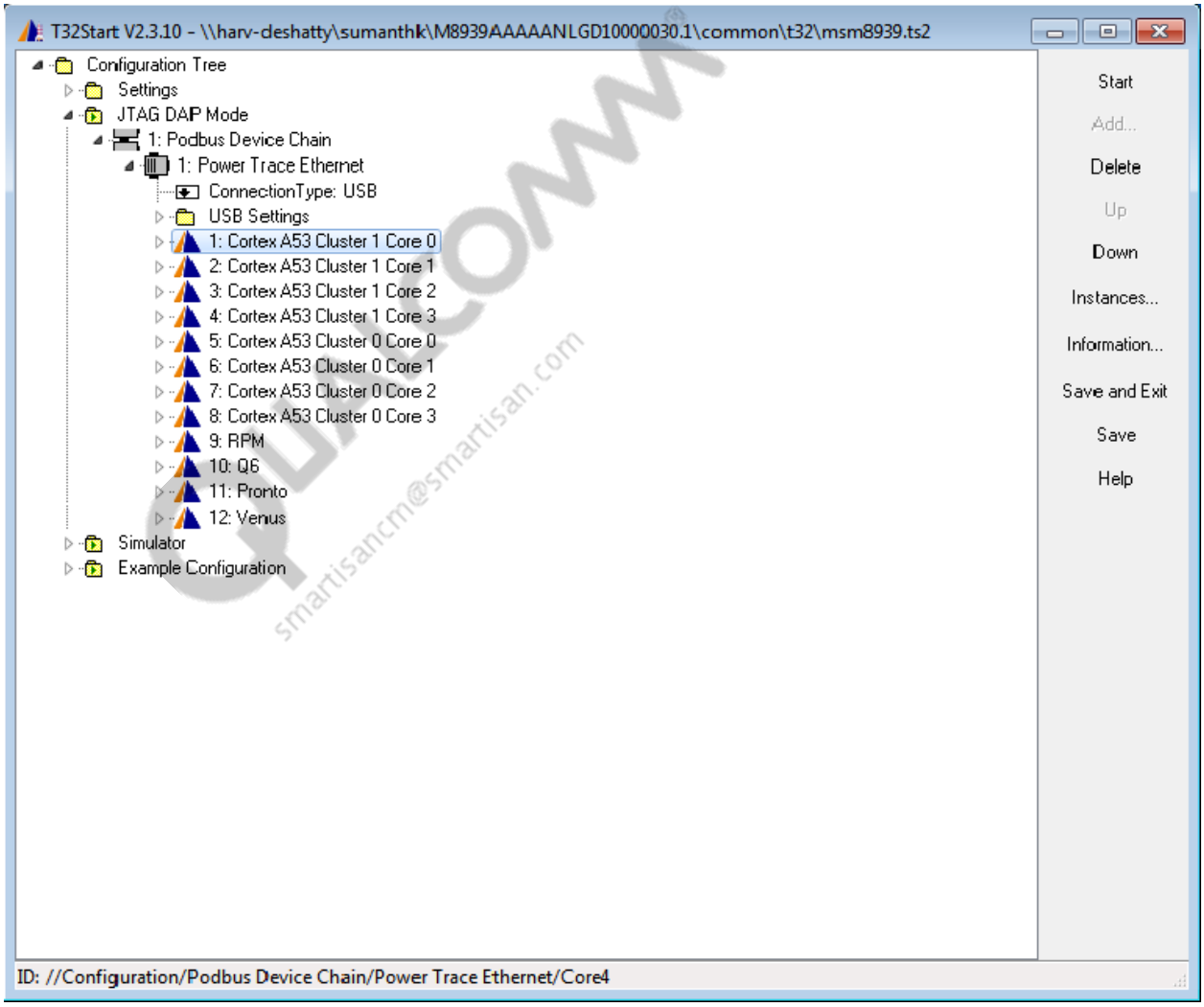
6. Push the files as follows:

```
adb push AppName.apk /system/app/.
```

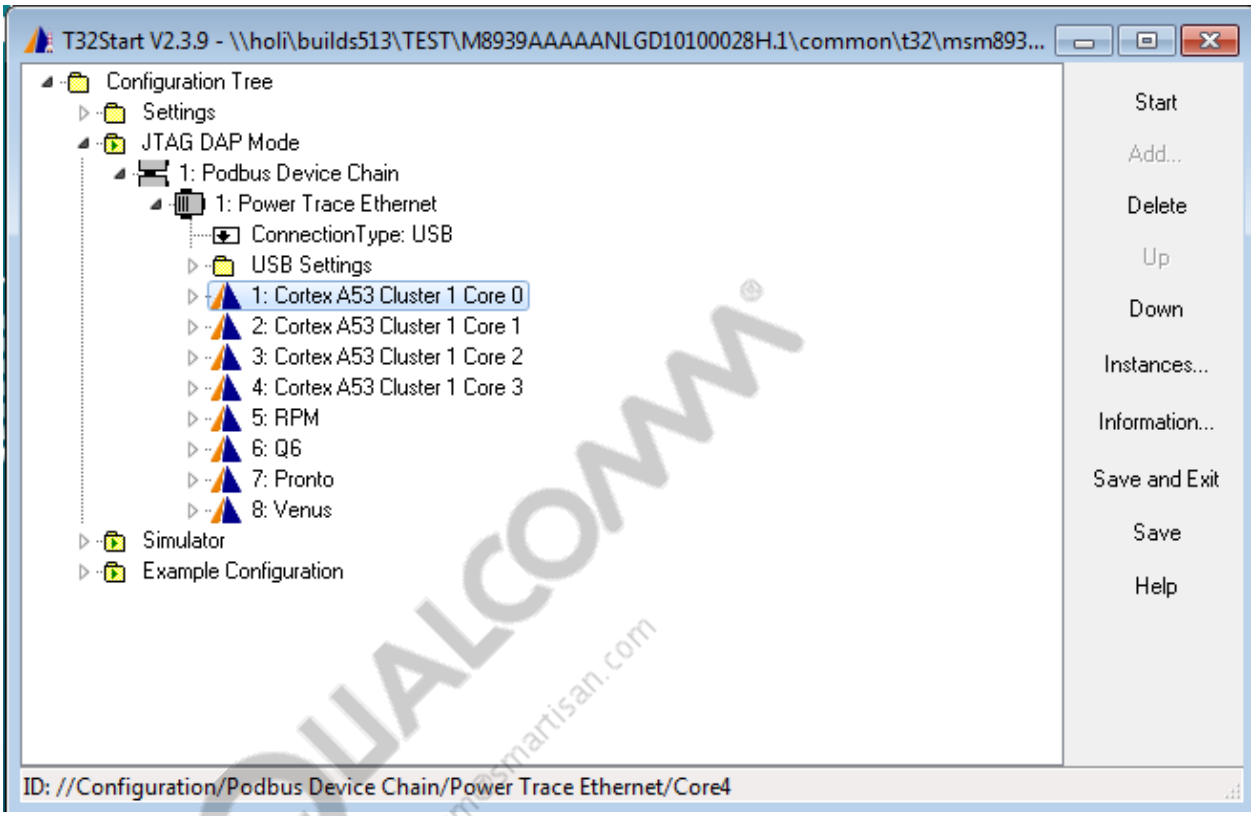
NOTE: In general, the syntax is: adb push <file_name> <location_on_the_target>

4.5.6 Programming eMMC boot loaders with T32

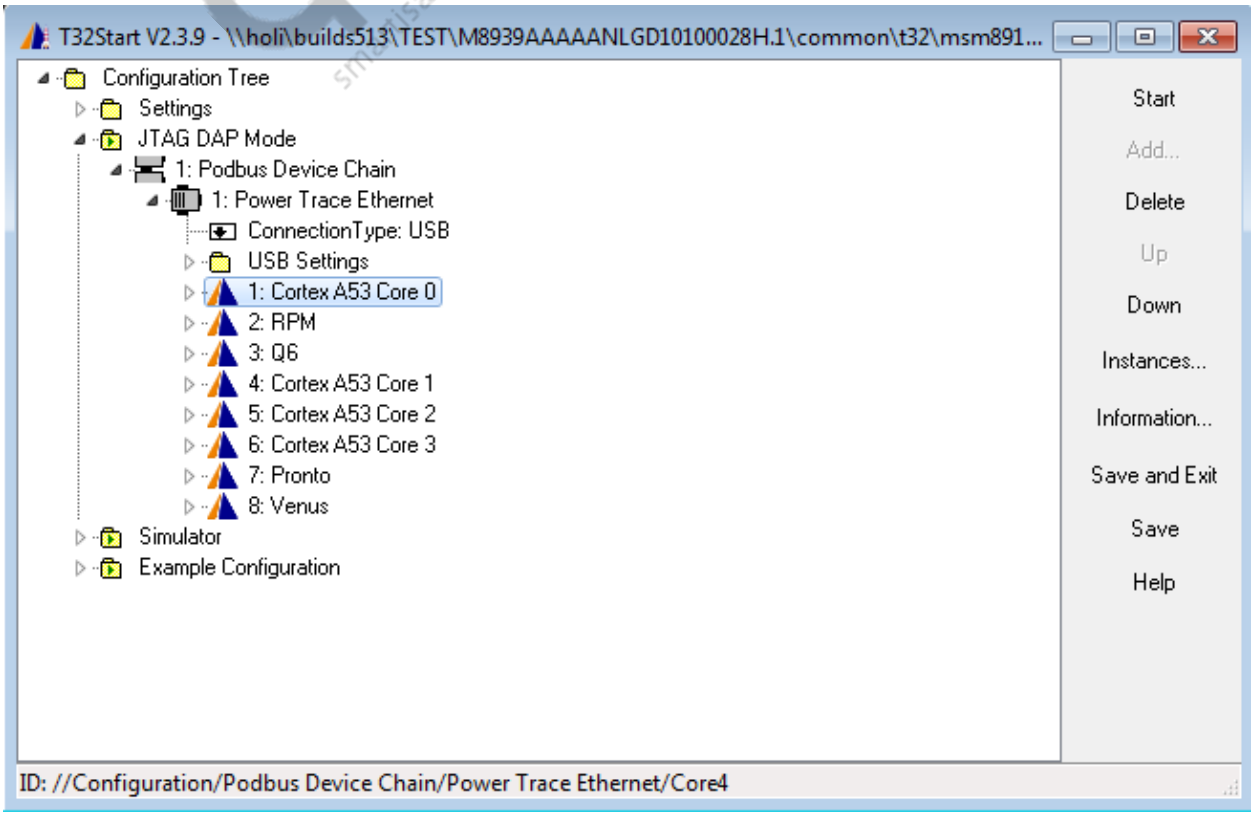
1. Start T32 by using the t32start.cmd in the <meta build>\common\msm8939\t32 folder.
2. For msm 8936 start t32start.cmd in the <meta build>\common\msm8936\t32 folder.
3. For msm 8916 start t32start.cmd in the <meta build>\common\msm8916\t32 folder.
4. Navigate to Configuration Tree→JTAG DAP Mode→Podbus Device Chain→Power Trace Ethernet.
   MSM8939:



MSM8936:

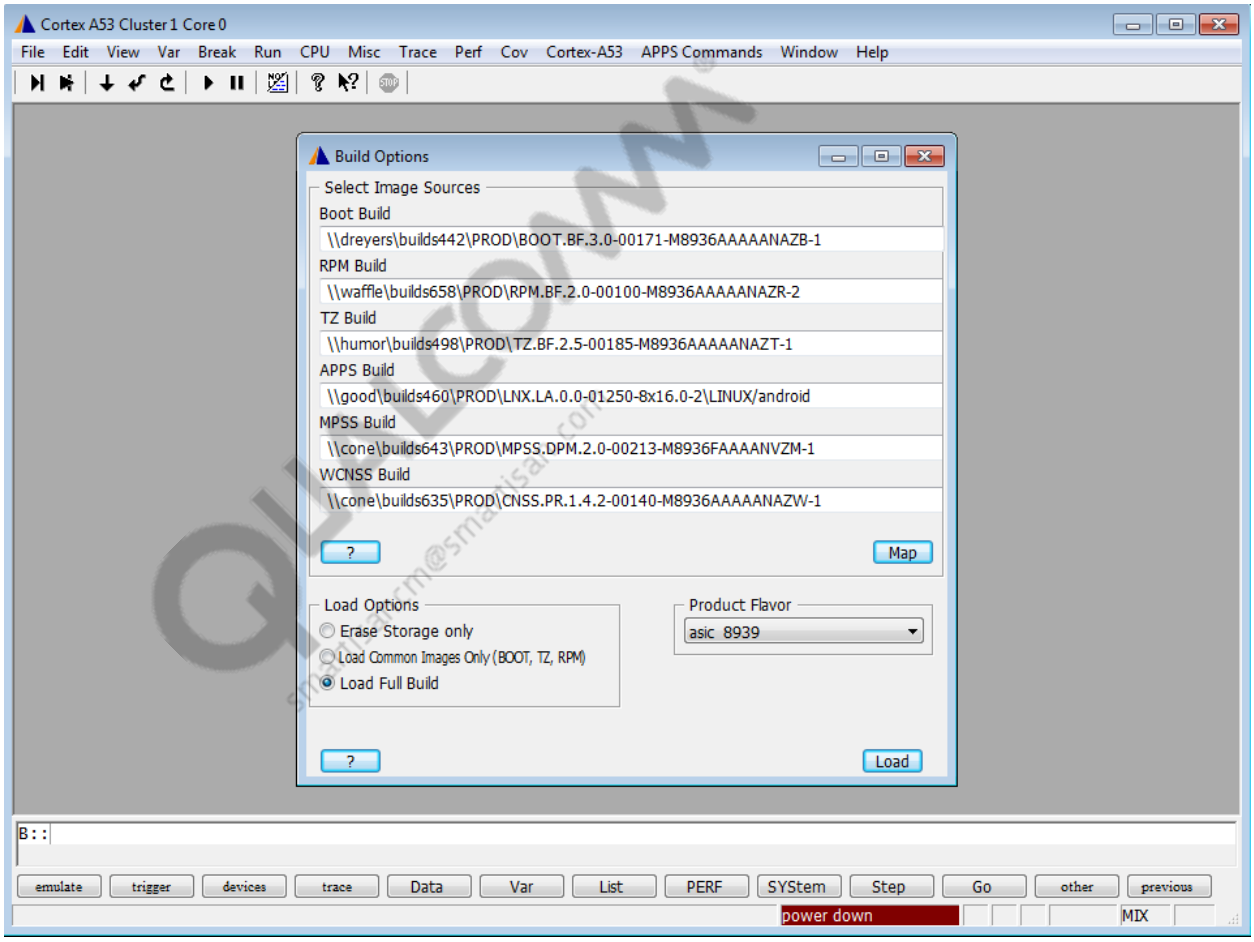


MSM8916:

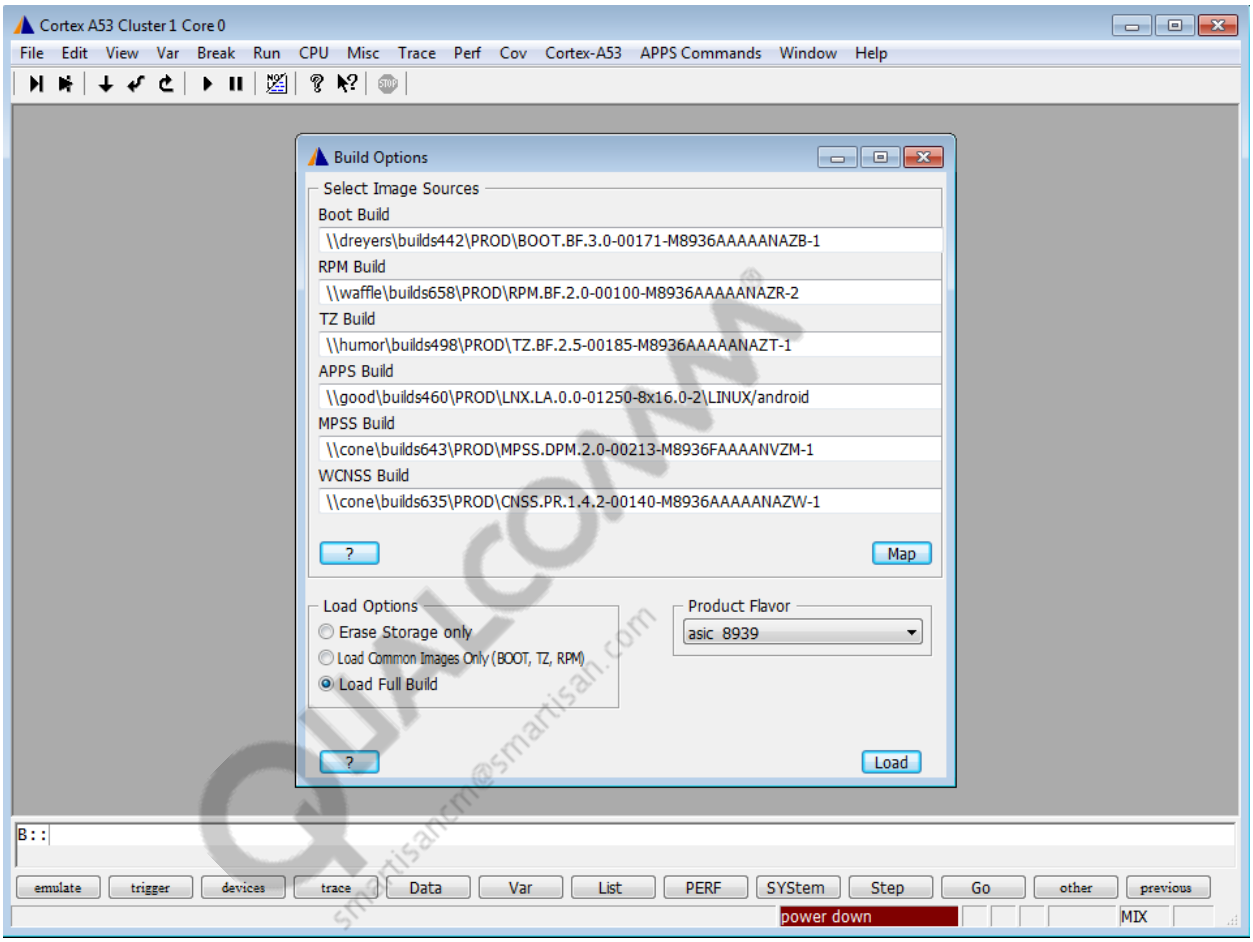


5. Select CortexA53 Cluster 1 Core0 and click **Start**.
6. In CortexA53Core0 T32 window, click **APPS COMMANDS→Build Options**, then select **ASIC** flavor in "Product Flavor" field and click **MAP**.
7. Click **Load** in the Build Options window.

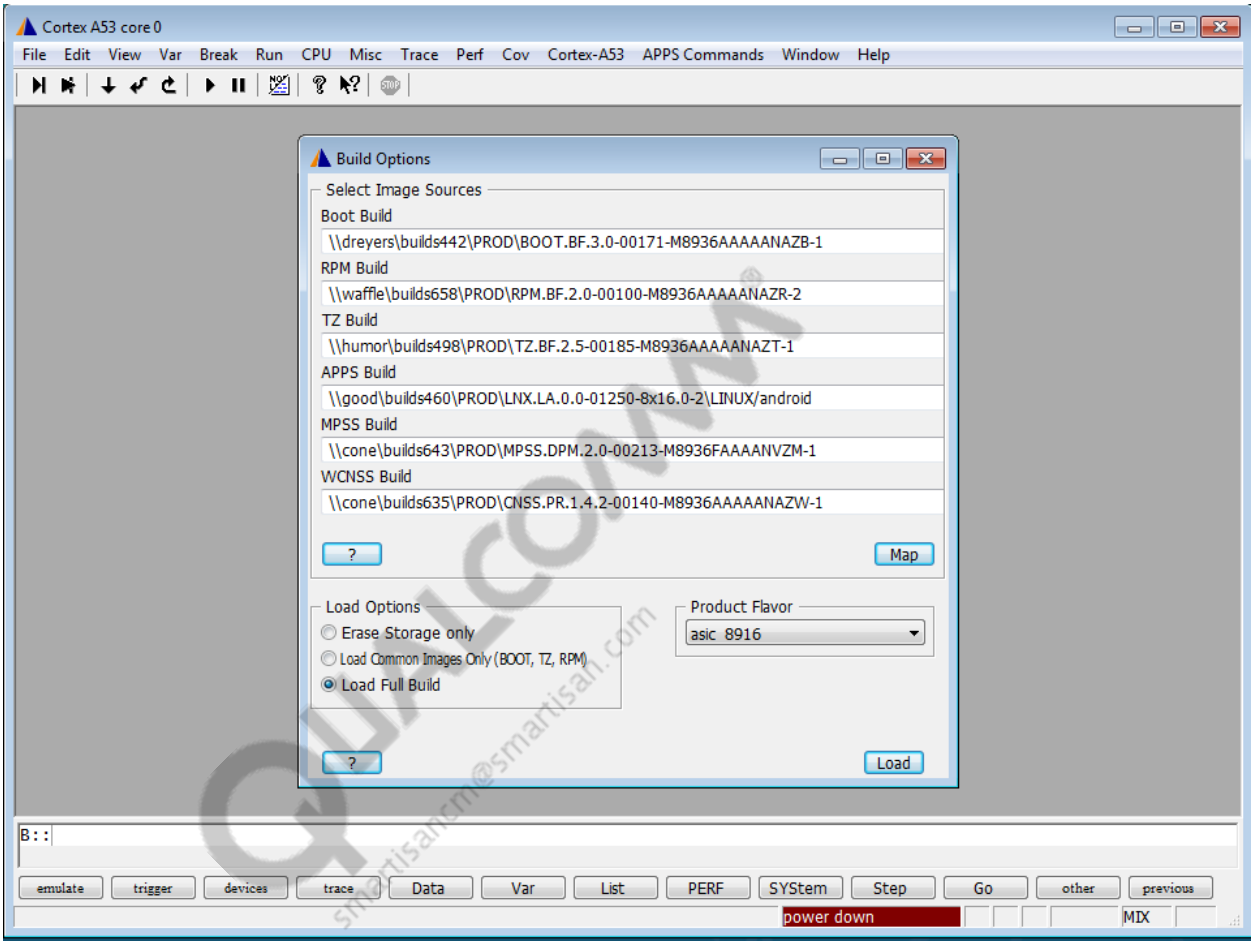
MSM8939:



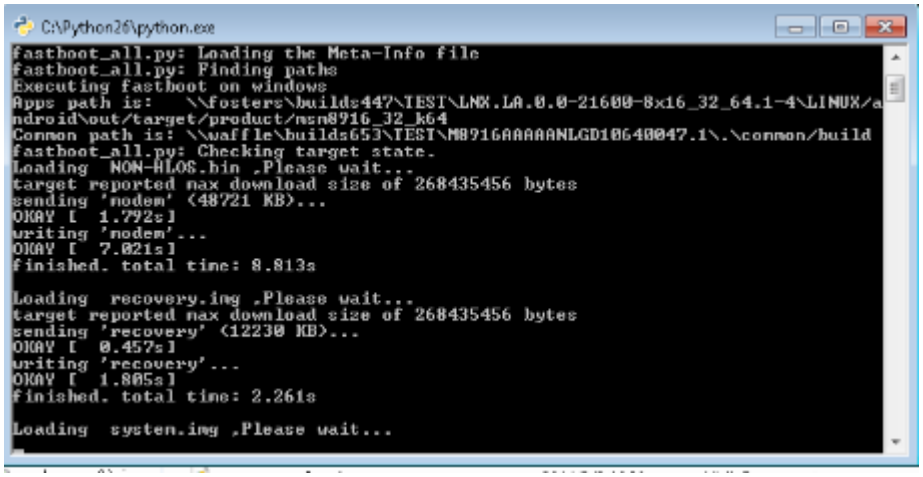
MSM8936:



MSM8916:



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.



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

5 Operational Guide

5.1 CDP and MTP SIM slots, primary antenna, and USB ports

5.1.1 CDP UIM configuration

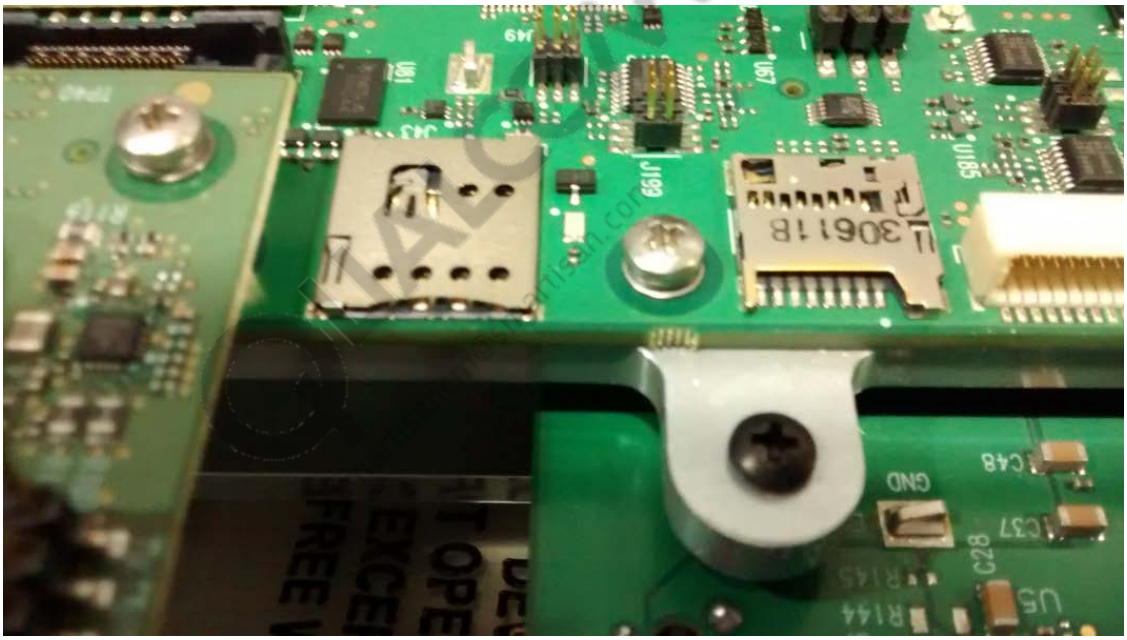


Figure 5-1 DUAL SIM slots on the baseband card

Figure 5-2 UIM3 on the baseband card

5.1.2 MTP UIM configuration

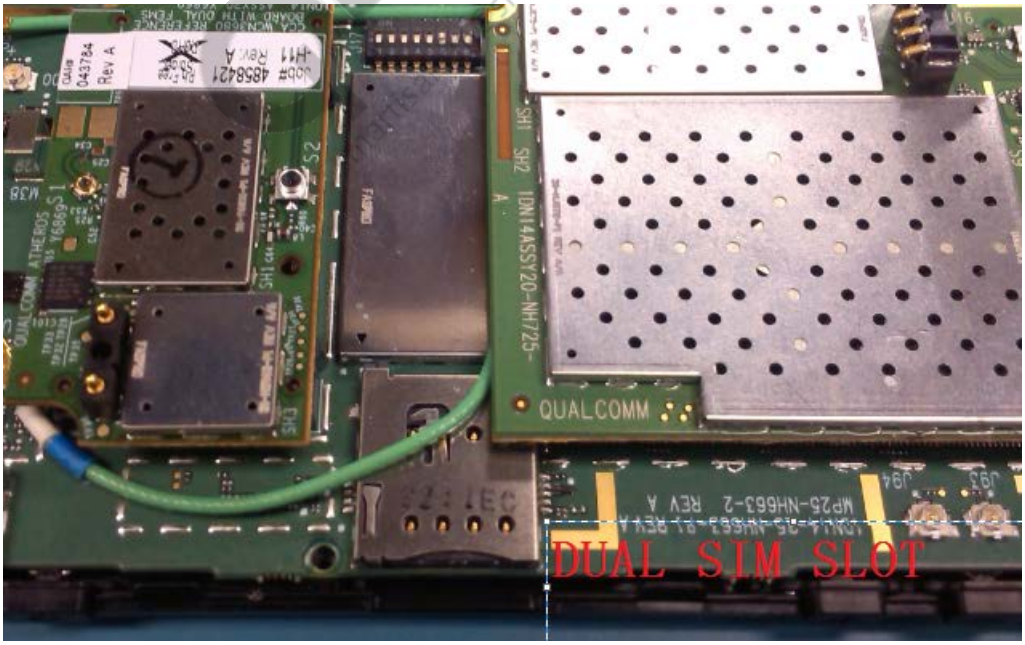


Figure 5-3 DUAL SIM slots on the baseband card

Figure 5-4 DUAL SIM slots (UIM1/UIM2) on the baseband card

5.1.3 CDP antenna configuration

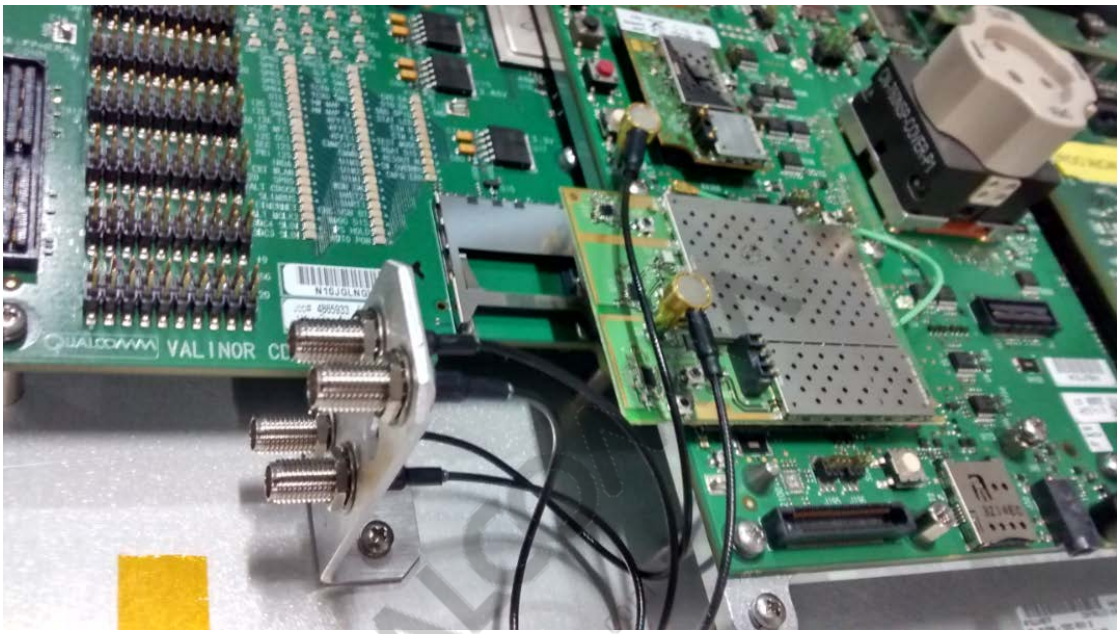


Figure 5-5 CDP antenna configuration

5.1.4 CDP USB/JTAG configuration

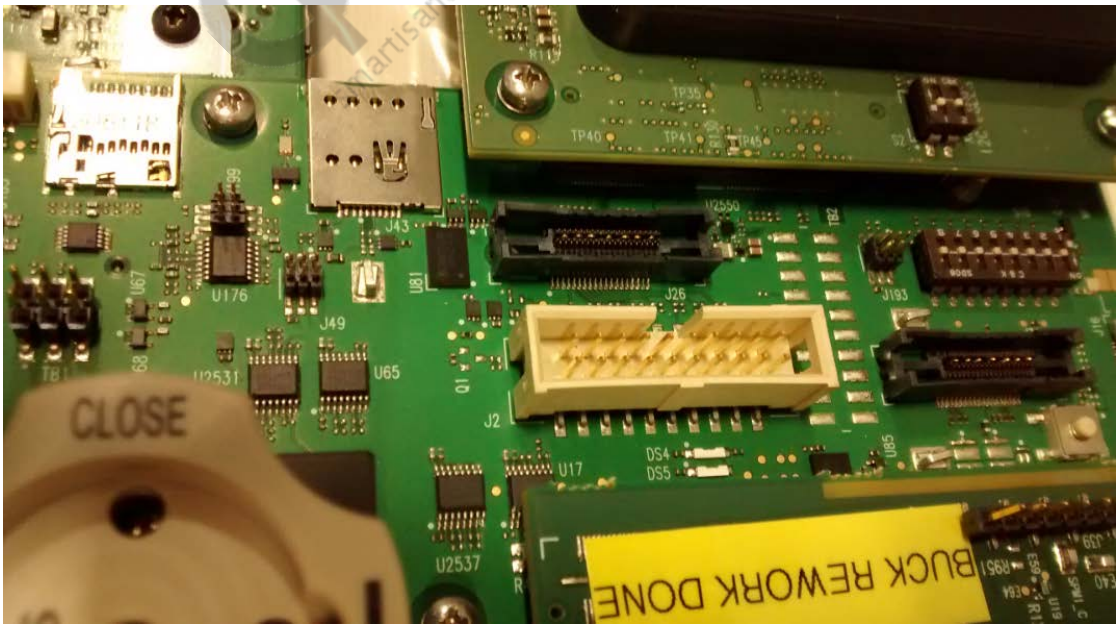


Figure 5-6 CDP JTAG configuration

Figure 5-7 CDP USB configuration

5.2 Common NV settings

5.2.1 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 5-1.

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

9. Run QRCT.
10. 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 5-1 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 Name	RF hardware ID (NV# 1878)	Comments	Reference QCN
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_SXDSDA_V2	92	An EPT configuration with QFE2101	MSM8916_CHILE_SxDSDA_V2_Reference_QCN.qcn

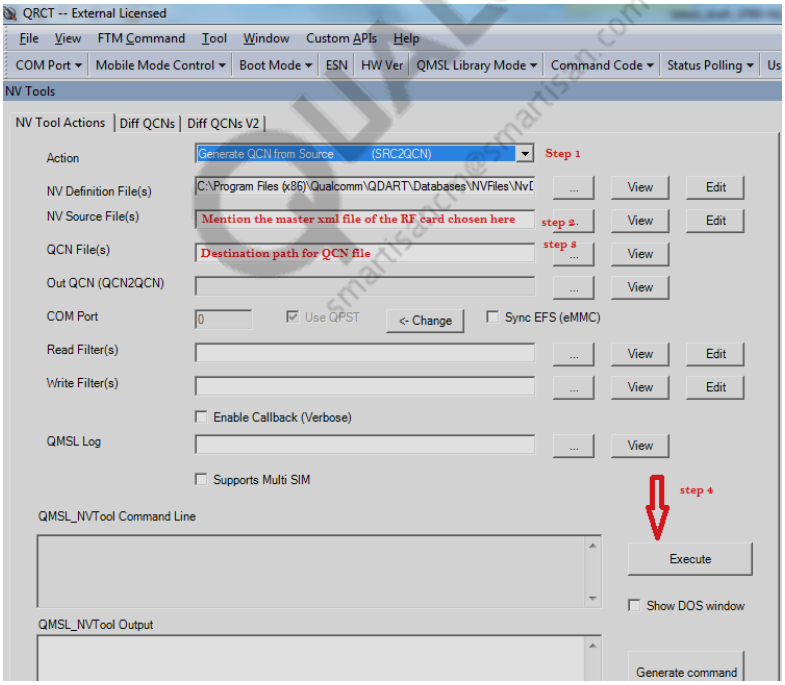


Figure 5-8 QRCT NV tool for generating QCN

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

5.2.2 NV configuration for WCDMA/GSM + GSM

Refer to Table 5-2 for WCDMA/GSM + GSM DSDS NV configuration.

Table 5-2 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
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 5-2, except NV70266 (Dual standby preference) must be set to 2.

5.2.3 NV configuration for CDMA + GSM

Refer to Table 5-3 for CDMA + GSM DSDS NV configuration

Table 5-3 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 Domain 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

NV item	Subscription1 (CDMA+ HDR) value	Subscription2 (GSM) value
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
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 5-3, except NV70266 (Dual standby preference) must be set to 2.

5.2.4 LTE + GSM DSDS settings

User must make the following changes to make sure that the device is correctly configured for LTE + GSM DSDS mode of operation.

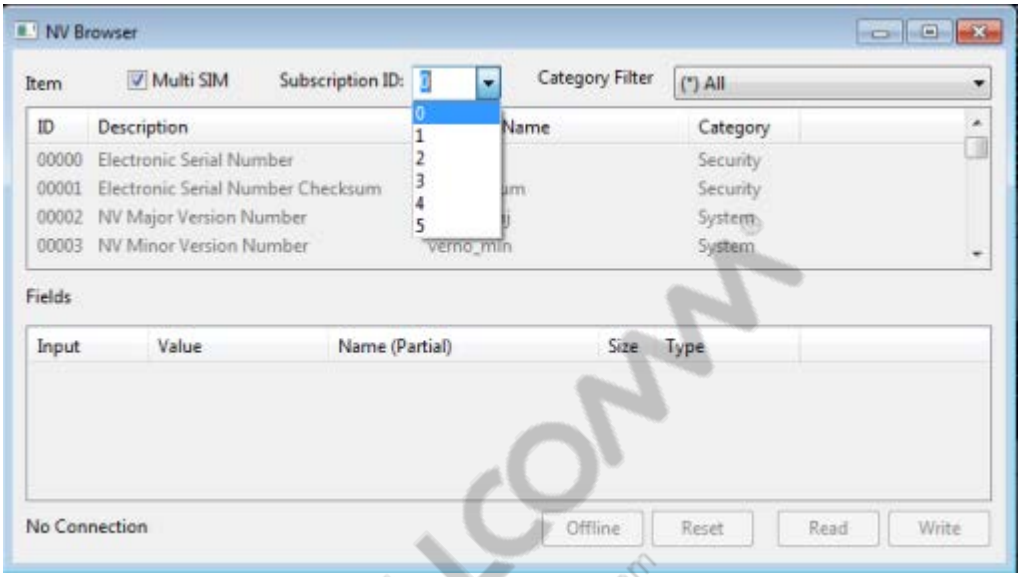
Volte/IMS/eMBMS and CSG features are not supported with LTE + GSM DSDS. This section explains all the needed NV items for disabling VoLTE/IMS/eMBMS as well as other parameters for LTE + GSM mode of DSDS operation.

Table 5-4 LTE + GSM DSDS settings

Feature Name	How to disable
VoLTE/IMS	Refer to section 3.6 of the application note in [Q37]
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

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:

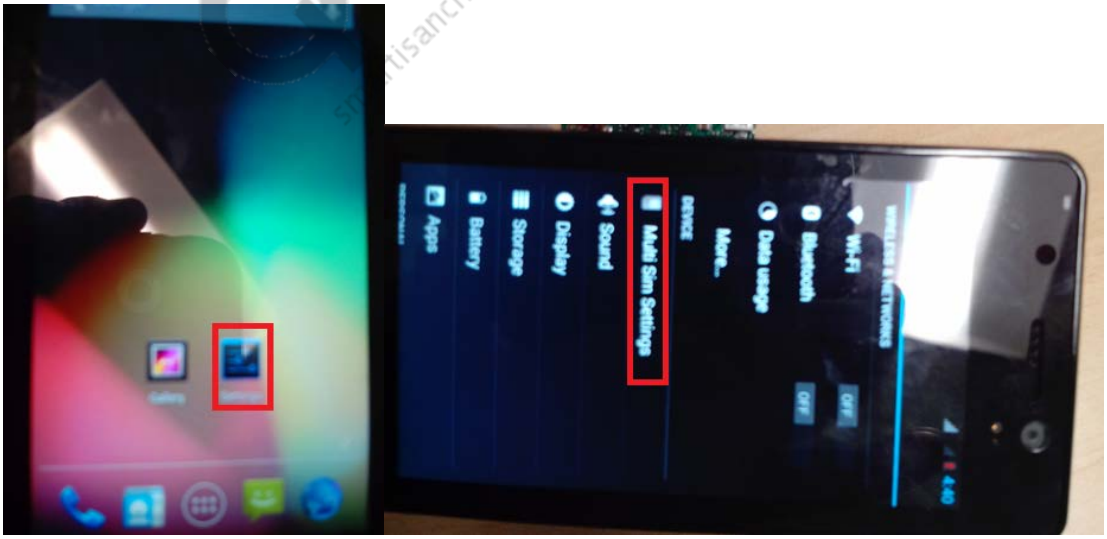
NV Item number	NV Item value
00855	0 (for both Single SIM and Dual SIM)
70266	1 (for Dual SIM)

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

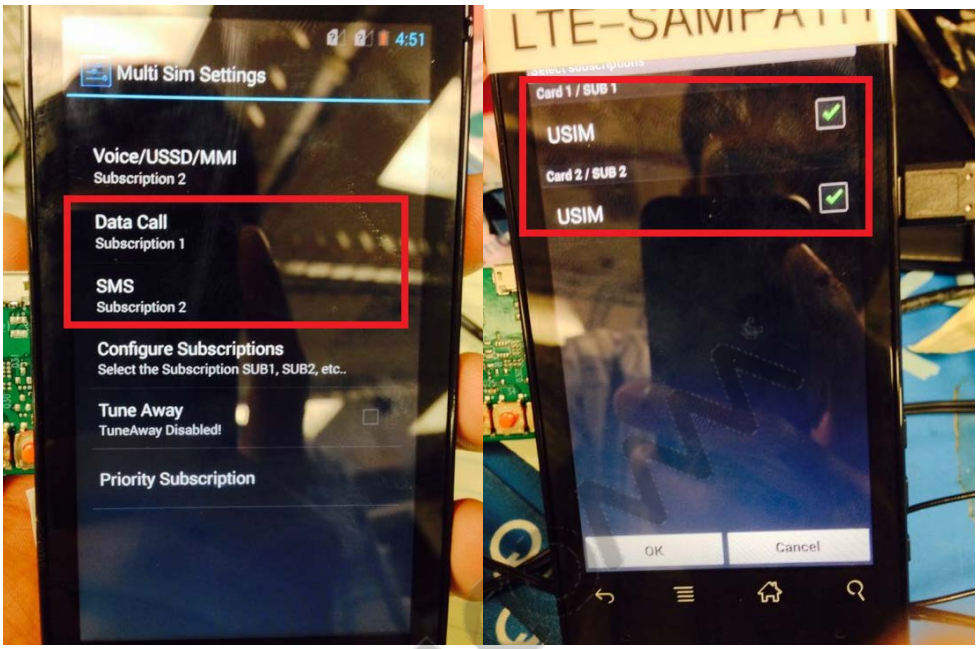
3. For NV settings for IRAT scenario – NV settings are required only on Subscription 0.

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

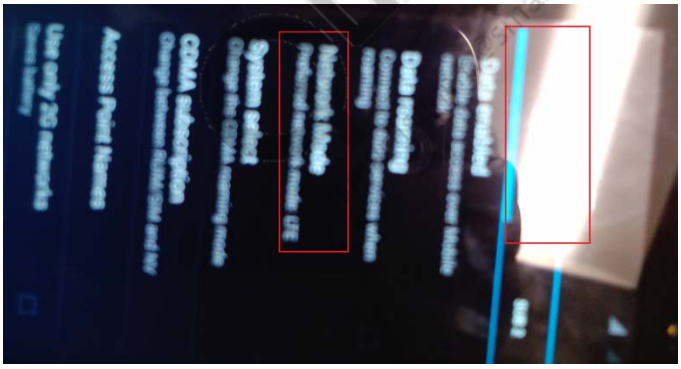
4. Restart the UE.
5. Run the dual SIM commands from ADB shell
   - adb devices
   - adb root
   - adb shell
   - setprop persist.radio.multisim.config dsds
   - getprop persist.multisim.config (it must show up as DSDS).
6. Restart the UE.
7. Perform the below setting from UI.
8. Go to Settings on the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

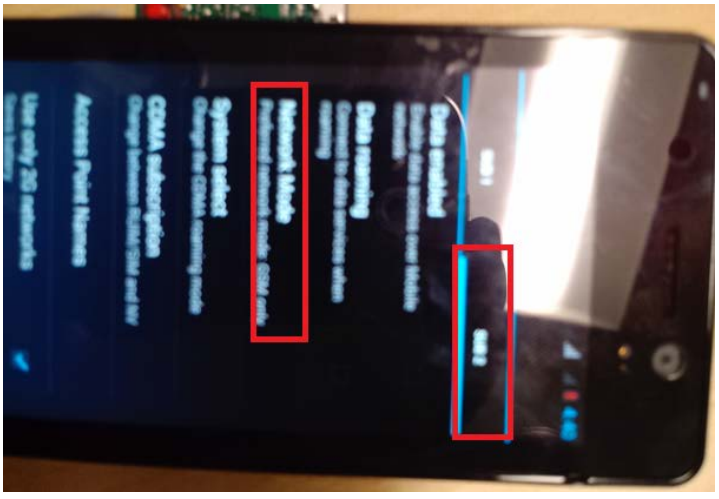


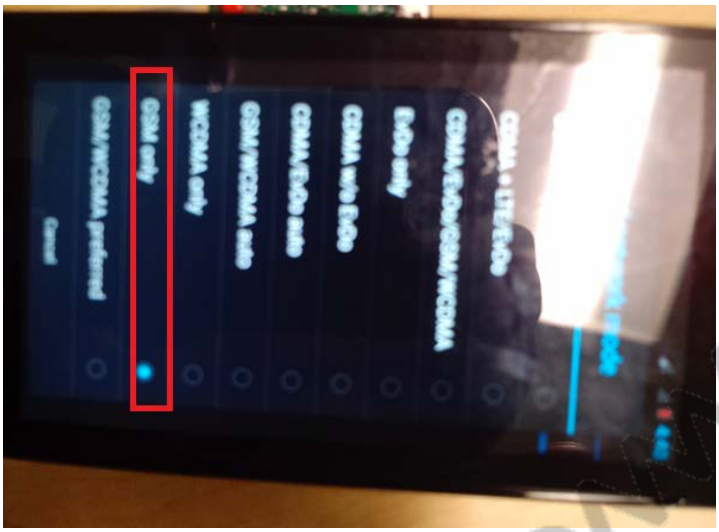
9. The user must be able to see Multi-SIM settings, select it.
10. Select Configure Subscriptions and configure the UIM cards to respective Subs.



11. Go back to settings, select Mobile Networks, and 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
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
```

5.2.5 1X SRLTE settings

The following settings need to be done to configure the device in 1X SRLTE 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

5.3 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 TDSCDMA) 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:

- On a fresh load of the build, the UE is by default configured for Manual Mode of Segment Loading operation.
- NV72542 – 2 (default) → The UE works in WCDMA Segment mode.
- NV72542 – 1 → The UE works in TD-SCDMA Segment mode.
- 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 NV72452 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

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

- 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

5.4 Call configuration

5.4.1 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

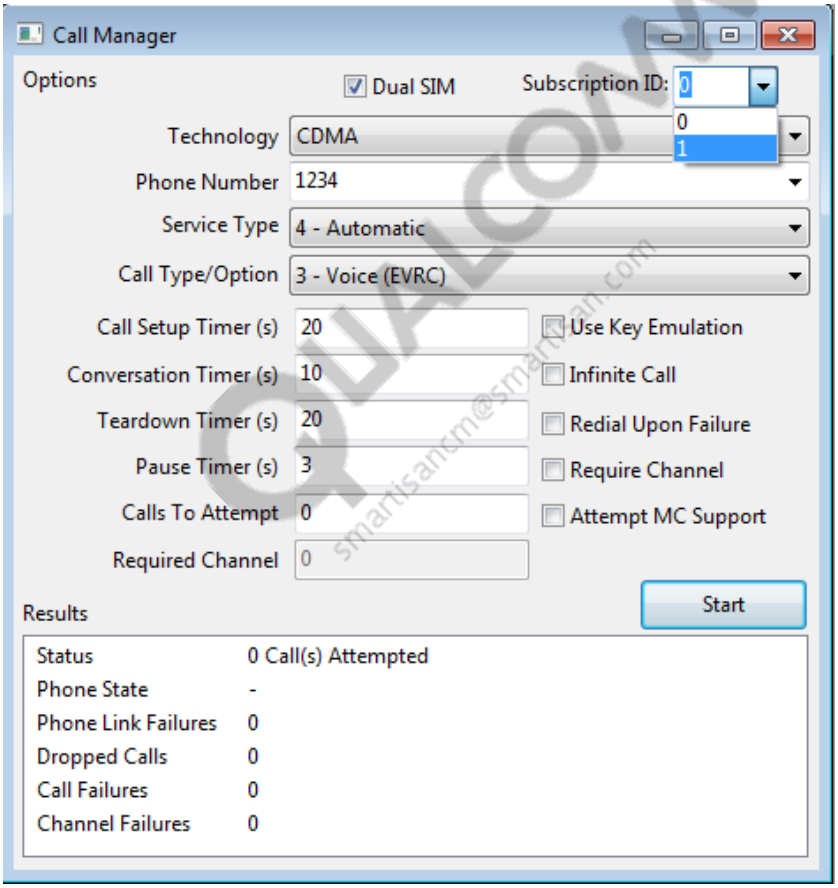
- Current testing has been done on Agilent 8960
- Callbox setup instructions
  - In System Config/Application Setup, select CDMA 2000 Lab App B (version should be B or above);
  - In Call Setup/Call Control screen
    - Set the Operating Mode to Active Cell;
    - Set the System Type to IS-2000;
    - 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);
  - In Call Setup/Call Params screen
    - Set Cell 1 Power to a proper value (between -45 ~ -65 dBm);
    - Set Cell Band and Channel, match the PRL setting;
    - Set Protocol Rev to 6(IS-2000-0);
    - Set Radio Config(RC) to (3,3) - (Fwd3, Rvs3);
    - 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 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**.



5.4.2 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

- Current testing has been done on Agilent 8960 or Anritsu MT8820;
- In Call Setup/Call Params screen
  - Set Radio Config(RC) to (3,3) - (Fwd3, Rvs3);
  - Set FCH Service Option Setup for (Fwd3, Rvs3) to SO32 (TDSO);
- 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.

5.4.3 HDR call

NV settings

- NV setting (set NV # 10 to 4 for Automatic mode)
- 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

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

Callbox setup

- For DO Rev A calls, the callbox must be placed 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

5.4.4 GSM voice call

Prerequisites

- NV settings
  a. Set NV #10(Mode Preference) to 13 for GSM only operation
  b. Set NV #441(Band Class Preference) to 0x200 for GSM900 band
  c. 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

- Current testing is performed on Agilent 8960 or Anritsu MT8820
- Set band to GSM900; cell power must be set somewhere between -45 and -65 dBm
- 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.

5.4.5 GPRS data call

RF calibration

- Ensure the device is RF-calibrated

Callbox setup (Agilent 8960)

- Select call setup screen
- Callbox setup
  - BCCH parameters→cell power = -75 dBm
  - BCCH parameters→cell band = EGSM
  - BCCH parameters→Broadcast chan = 20
  - PDTCH parameters→Multislot config = 1 Down 1 Up
  - Operating mode = Active mode GPRS
  - 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

5.4.6 WCDMA voice call

Prerequisites

- QCN – Ensure device is calibrated

Callbox setup

- Agilent 8960
  - Cal box setup
    - Call Control/Security Info/Security Parameters/Security Operations – None
    - Call Parms/Cell Power – -50.00 dBm
    - Call Parms/Channel Type – 12.2 KRMC
    - Call Parms/Paging Service – AMR Voice
  - Test results – MO and MT passed 5/5
- Anritsu 8480C
  - Callbox setup
    - Station Globals – Spec_Release – 3; HSDPA – FALSE; EUL – FALSE
- Anritsu 8820
  - Callbox setup
    - Call Processing – On
    - Test Loop Mode – Off
    - 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”

5.4.7 WCDMA data call

Prerequisites

- QCN – Ensure device is calibrated

Callbox

- Agilent 8960
  - Callbox setup
    - Call Parms/Channel Type – HSPA
  - Test results – DUN data calls passed with ~384-kbps throughput; QMI not tried
- Anritsu 8480C
  - Callbox setup
    - Station Globals – Spec_Release – 5; HSDPA – TRUE; EUL – FALSE
  - Test results – DUN data calls was up but crashed later; QMI not tried
- Anritsu 8820
  - Callbox setup
    - Call processing – On
    - Test Loop Mode – Mode 1
    - 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)

5.4.8 TDSCDMA 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
    - Cell Power – -45.00 dBm
    - Channel Type – 12.2K RMC SC
    - Paging Service – AMR Voice
    - Channel – Desired band/channel
    - Screen2 – Cell Info → Cell Parameters → PS Domain Information → Present
- Anritsu 8820
  - Callbox setup
    - Call processing – On
    - Test Loop Mode – Mode 1
    - Signal/Chanel Coding – Fixed Reference Channel

Device setup

To make a call:

1. Configure the test equipment using the settings mentioned in Section 5.4.8 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.

5.4.9 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
    - Under home screen, select Mode as Callbox Mode
    - Select Parameter Config Tab.Parameter Group → Select System → Band ID Ex: For TDD Testing:38 & For FDD Tsting:1
    - Under Parameter Group select Layer 3 → MCC 001 and MNC 01 based on your SIM configuration.
- Anritsu 8820
  - Callbox setup
    - Call Processing – On
    - Test Loop Mode – Off
    - Signal/Channel Coding – Required LTE band/channel
    - 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.

5.5 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, see [Q13] for complete details

5.6 Multimedia configuration

This section describes the bringup of multimedia features like audio, display, video, and camera. For further details, see [Q22].

5.6.1 Audio configuration

5.6.1.1 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

5.6.1.2 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 tinyplay command:
   - tinyplay <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`

5.6.1.3 Audio recording

This section describes the procedure to verify a 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
```

1. Start the audio recording.

```
tinycap /data/rec.wav
```

2. 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'
```

5.6.1.4 Voice call

The QXDM log is enabled with the voice log mask and QCAT's vocoder playback function is used to play vocoder packets and PCM separately.

The voice quality issues are listed below along with their troubleshooting leads.

- For Rx path, if vocoder packet is OK but PCM is not OK – Analyze voice tuning.
- For Tx path, if the PCM is OK, but the vocoder packet is not OK – Analyze voice tuning.
- For Rx path, if the vocoder packet is not OK – Analyze the network.
- For Tx path, if the vocoder packet is OK, but the far end still hears noise – Analyze the network.

Mute issues – These are likely due to device issues. Verify that the device codec widgets are correctly connected and the gain is muted. The voice quality issue can be due to the vocoder packets and the PCM playback.

5.6.1.5 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 can be found from the kernel logs; look for "Button Jack" and "Headset Jack".

Input – `msm8939-snd-card Button Jack as /devices/platform/soc-audio.0/sound/card0/inputX`

Input – `msm8939-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 an USB port.
2. 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 will be 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 will printed 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 can be found in the kernel logs. In the kernel logs, look for "Reporting insertion" or "Reporting removal".

- “Reporting insertion 1” is for headphone without a microphone
- “Reporting insertion 3” is for headset with microphone

To detect headset button press/release detection:

1. Connect the target device to a PC using an USB port.
2. 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 will be printed in the adb shell window if the headset button is pressed:

```
/dev/input/eventX: 0001 0001 00000001
/dev/input/eventX: 0000 0000 00000000
```

Below messages will 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 can be 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 can be 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
```

5.6.1.6 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 'HPHL 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)f
   f.a
   ```

5.6.2 Display

For display panel bringup and driver porting-related information, see [Q25]. [Q25] 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 [Q26].

5.6.3 Camera

Android default camera application is used to verify camera features. For details related to sensor driver porting and migration, refer to [Q31].

For techniques on debugging different kind of camera errors, refer to [Q32].

5.6.4 Video

Android default video player application is used to verify the playback of various video formats. Default camera application can be used to verify video recording use case.

5.7 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 [Q27].

5.8 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). Th**e** 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.

6 Factory Tools

6.1 QDART-MFG and TPP

QDART-MFG installer is a set of factory tools designed to manufacture, reduce the setup steps, and optimize installation size and installation time. It provides GoNoGo UI, QSPR test framework, test solution for all test stations, including: software download and upgrade, RF calibration and verify, Bluetooth® and WLAN, MMI, radiated, and service programming. For more details, see [Q28] and [Q29]

6.2 FactoryKit

FactoryKit is an Android application used for MMI test. It functions similar to FastMMI but with longer bootup time. FactoryKit is used on customer devices and all QRD SKU devices.

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)
  - init.rc – Initialization config files (device config, service startups) init.qcom.rc

- dev/ – Device nodes
- proc/ – Process information
- sys/ – System/kernel configuration
- sbin/ – System startup binaries (ADB daemon; read-only)
- system/ – From system.img (read-write)
  - bin/ – Android system binaries
  - lib/ – Android system libraries
  - xbin/ – Nonessential binaries
  - framework/ – Android framework components (Java)
  - app/ – Android applications (Java)
  - etc/ – Android configuration files
- sdcard/ – Mount point for SD card
- data/ – From userdata.img (read-write)
  - app/ – User installed Android applications
  - tombstones/ – Android crash logs

A.2 Building the Linux kernel manually

1. Set up the Android build environment (envsetup.sh/choosecombo).
2. Change to the kernel directory (kernel/).
3. Set up the correct kernel config with the command:

   ```
   make ARCH=arm CROSS_COMPILE=arm-eabi- msm8974_defconfig
   ```

4. Build the kernel image with the command:

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

5. 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.

6. 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.3 Building Android manually

1. Set up the Android build environment (envsetup.sh/choosecombo).
2. Change to the main Android directory.
3. Build with the command:

   ```
   make -j4
   ```

4. 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
     ```

5. 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.4 Other important Android build commands

Other important Android build commands are:

- printconfig – Prints the current configuration as set by the choosecombo commands.
- 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.
- - mm – Builds all of the modules in the current directory.
- - mmm – Builds all of the modules in the supplied directories.

- croot – cd to the top of the tree.
- sgrep – grep for the regex you provide in all .c, .cpp, .h, .java, and .xml files below the current directory.
- clean-$(LOCAL_MODULE) and clean-$(LOCAL_PACKAGE_NAME)
  - 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.
- 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:

- Similar to regular GNU makefiles; some differences are:
  - Predefined variables to assign for source files, include paths, compiler flags, library includes, etc.
  - Predefined action for compiling executables, shared libraries, static libraries, Android packages, using precompiled binaries, etc.
- Variables
  - LOCAL_SRC_FILES – List of all source files to include
  - LOCAL_MODULE – Module name (used for "m")
  - LOCAL_CFLAGS – C compiler flags override
  - LOCAL_SHARED_LIBRARIES – Shared libraries to include
- Action
  - include $(CLEAR_VARS) – Clears LOCAL* variables for the following sections:
    - include $(BUILD_EXECUTABLE)
    - include $(BUILD_SHARED_LIBRARIES)
    - 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 \
dir/file2.cpp
```

The new module can be configured with the following:

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

  ```
  LOCAL_STATIC_LIBRARIES := \
  libutils \
  libtinyxml
  ```

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