QM215 Linux Android Software User Manual

SP80-PK881-4 A

October 17, 2018

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

Revision history

Revision	Date	Description
A	October 2018	Initial release

Contents

Tables

1 Introduction

1.1 Purpose

This document describes how to obtain, build, and program software applicable to the QM215 Linux Android Software Product Family (SPF) “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, and so on

1.2 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `cp armcc armcpp`.

Code variables appear in angle brackets, for example, `<number>`.

Commands to be entered appear in a different font, for example, **`copy a:*.* b:`**.

Button and key names appear in bold font, for example, click **Save** or press **Enter**.

1.3 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://createpoint.qti.qualcomm.com/.

If you do not have access to the CDMATech Support website, register for access or send email to support.cdmatech@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	Python	2.7.6	Python.org	Building boot, subsystem, and so on. Versions are identified for each subsystem in Table 3-2.
7	SCons v2.0.0 or higher	–	scons.org	Building all non-HLOS source code releases

2.2 Install Ubuntu

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

1. Create an installation CD and install it on 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**
   - 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
     ```

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

2.3 Configure 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 Install JDK

The Sun JDK is no longer available in Ubuntu’s main package repository. 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 Install Repo

The Repo tool is a source code configuration management tool used by the Android project, see *Installing Repo*. It is a front-end to git written in Python, which uses a manifest file to help download 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: To access online help, install Repo (repo init).

2.6 Install the Arm compiler tools

Building the non-HLOS images requires the specific version of the Arm Compiler Tools indicated in Table 3-2. 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, see Arm support website.

Install 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.

Install 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. The default install location is C:\Program Files (x86)\ARM_Compiler5\.

   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.
4. 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.

5. 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 Install 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 the *Hexagon Tools Installation Guide* (80-N2040-32) for detailed procedures to download and install the Hexagon toolchain software. See the *Hexagon Development Tools* Overview (80-N2040-12) for more documentation on using the Hexagon tools.

See *Qualcomm Hexagon LLVM C/C++ Compiler User Guide* (80-VB419-89) and *LLVM Compiler Hexagon Processor Deployment Plan* (80-VB419-87), for a detailed explanation of the LLVM compiler in the Hexagon toolchain. LLVM compilers work with Hexagon software development tools and utilities to provide a complete programming system for developing high-performance software.

3 Download and build 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 https://chipcode.qti.qualcomm.com/	From https://www.codeaurora.org/
▪ Proprietary non-HLOS software ▫ Contains proprietary source and firmware images for all non application processors ▫ An umbrella package built from a combined set of integrated individual component releases	
▪ Proprietary HLOS software ▫ Contains proprietary source and firmware images for the application processor HLOS	▪ Open source HLOS software ▫ Contains open source for application processor HLOS

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

<PL Image>-<Version>-<Chipset>

- <PL_Image> – LA.Branch for Linux Android
- <Version> – Variable number of digits used to represent the build ID version
- <Chipset> – MSM8937

For example, LA.UM.5.1-01010-8x37.0

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. Ensure that 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	Source	Android GNU toolchain Ubuntu 14.04 arm-linux-androideabi-gcc (GCC) 4.9	–	–	–	Linux only
MPSS	Source	Hexagon 6.4.06	Python 2.7.5	Perl 5.18.2 (Install XML:: Parser module) sudo apt-get install aptitude sudo aptitude install libxml-simple-per	Windows builds only; needs tee.exe	Linux, Windows XP, and Windows 7
Boot loaders	Source	Arm Compiler Tools 5.01 update 3 (build 94)	Python 2.7.5	Perl 5.8.x Linux builds only	Windows builds only; needs tee.exe	Linux, Windows XP, and Windows 7
RPM	Source	Arm Compiler Tools 5.01 update 3 (build 94)	Python 2.7.6	Perl 5.6.1	Windows builds only; needs tee.exe	Linux, Windows XP and Windows 7 only
TZ	Source	Snapdragon LLVM Arm compiler 3.5.2.4 gcc-linaro-arm-linux-gnueabihf-4.8-2014.02_linux	Python 2.7.5	–	Windows builds only; needs tee.exe	Linux, Windows XP, and Windows 7 only
CNSS	Binary	–	–	–	–	–
aDSP	Source	Hexagon Tools Version: 8.0.07	Python 2.7.6	Perl 5.6.1	Windows builds only; needs tee.exe	Linux, Windows XP, and Windows 7 only
CPE	Binary	–	–	–	–	–
Video	Binary	–	–	–	–	–

3.1 Download QTI proprietary software from Qualcomm ChipCode™ Portal

NOTE: QTI software can be downloaded from the Qualcomm ChipCode. Designated points of contact in your organization can download the licensed software. The software is organized into distribution packages (distros) composed of subsystem image files. Each distro has a corresponding Git project. The Git tree includes revisions for previous builds that allow you to diff the changes between releases.

1. If you are new to Qualcomm ChipCode, review the following link for up-to-date documentation and a set of tutorial videos:

   https://chipcode.qti.qualcomm.com/projects/help/wiki

2. Create a top-level directory on the build PC and unzip each of the subsystem images to generate the following directory structure for Software Product Family (SPF) distribution. In this example, <target_root> is the top-level directory.

   <target_root>

   /ADSP.VT.3.0/adsp_proc

   /BOOT.BF.3.3.2/boot_images

   /CNSS.PR.4.0.3/wcnss_proc

   /CPE.TSF.3.0/cpe_proc

   /LA.UM.7.6.2/LINUX

   /MPSS.JO.3.1/modem_proc

   /MPSS.TA.3.0/modem_proc

   /MSM8917.LA.3.2.1/common

   /SDM439.LA.1.0.1/common

   /SDM429.LA.1.0.1/common

   /SDM450.LA.3.2.1/common

   /SDM632.LA.1.0.1/common

   /MSM8953.LA.3.2.1/common

   /RPM.BF.2.2/rpm_proc

   /RPM.BF.2.4/rpm_proc

   /TZ.BF.4.0.5/trustzone_images

   /VIDEO.VE_ULT.3.1/venus_proc

   /VIDEO.VE.4.4/venus_proc

   /WLAN_ADDON.HL.1.0/addon <**Applicable only if you have a WAPI license**>

3. Ensure that the contents.xml file is located in the root folder as shown in Step 2.

4. For various metabuild components and IDs, see the about.html file in the root of the repository on Qualcomm ChipCode. For example, see Table 3-3.

Table 3-3 Example of metabuild components

Name	Build ID	SPF image dir
META build	MSM8917.LA.3.2.1/common	–
aDSP	ADSP.VT.3.0-00109-00000-1	ADSP.VT.3.0/adsp_proc
Boot	BOOT.BF.3.3.2-00056-M8953JAAAANAZB-1	BOOT.BF.3.3/boot_images
CNSS	CNSS.PR.4.0.3-00165-M8953BAAAANAZW-1	CNSS.PR.4.0.3/wcnss_proc
CPE	CPE.TSF.3.0-00002-W9335AAAAAAAZQ-4	CPE.TSF.3.0/cpe_proc
Android	LA.UM.7.6.2.r1-03200-89xx.0-1	LA.UM.7.6.2/LINUX
MPSS	MPSS.JO.3.1-00158-SDM439_GENNS_PACK-1	MPSS.JO.3.1/modem_proc
RPM	RPM.BF.2.2-00244-M8917AAAAANAZR-2	RPM.BF.2.2/rpm_proc
TZ	TZ.BF.4.0.5-00134-M8937AAAAANAZT-1	TZ.BF.4.0.5/trustzone_images
Video	VIDEO.VE_ULT.3.1-00035-PROD-1	VIDEO.VE_ULT.3.1/venus_proc
WLAN.ADDON_PR	WLAN_ADDON.HL.1.0-00031-CNSS_RMZ_WAPI-1	WLAN_ADDON.HL.1.0/addon

3.2 Download open source HLOS software

The Linux board support package (BSP) release is obtained in two parts, a proprietary release from Qualcomm ChipCode and an open source release from the code aurora forum (CAF) site.

1. Open the contents.xml file that was downloaded from Qualcomm ChipCode (see Section 1.5 and search for the <name>apps</name> tag.
2. Below this tag, locate the <build_id> tag. This identifies the corresponding open source HLOS software build similar to LA.UM.6.6.r1-02701-89xx.0.
3. 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. For example, see Table 3-4.

Table 3-4 Example of APSS build ID

Tag / Build ID	Chipset	Manifest	Android Version
LA.UM.7.6.2.r1-03200-89xx.0	MSM8953_64	LA.UM.7.6.2.r1-03200-89xx.0.xml	09.00.00

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

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

5. Type the following Repo sync command:

```
$ repo sync
```

6. 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/*
make –j4
```

3.3 Compile the non-HLOS software

3.3.1 Set build Windows environment

Before issuing the non-HLOS build commands, certain command environment settings are 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 myenviron_amss.cmd script (for Windows), which sets the path to point to the Arm toolchain lib, include, bin, and license file configuration.

```
#
# myenviron_amss
#
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:\cygwin\bin; %PATH %
set HEXAGON_ROOT=C:\Qualcomm\HEXAGON_Tools
set HEXAGON_RTOS_RELEASE=5.0.07
set HEXAGON_Q6VERSION= v4
set HEXAGON_IMAGE_ENTRY=0*x08400000
```

3.3.2 Build boot loaders

Compiler version: ARMCT501B94. For correct Tool versions, see the Table 3-2 and set the export paths appropriately.

1. For Linux, verify that the following paths are set. See the setenv.sh in your boot build <target_root>/BOOT.BF.3.3.2/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/2.7.5/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.BF.3.3.2/boot_images/build/ms
```

Where <target_root> is the top-level directory created in Section 3.1

Depending on the build environment or release, use one of the command options described in Table 3-5.

Table 3-5 Build boot loaders

Build environment	Build command

Linux	▪ Build images –
	`./build.sh TARGET_FAMILY=8917 --prod`
	▪ Clean the Build –
	`./build.sh TARGET_FAMILY=8917 --prod –c`

Windows	▪ Build images –
	`build.cmd TARGET_FAMILY=8917 --prod`
	▪ Clean the Build –
	`build.cmd TARGET_FAMILY=8917 --prod –c`

3.3.3 Build TZ images

Compiler – LLVM3521LGCC

1. Navigate to the following directory:

```
cd <target_root>/TZ.BF.4.0.5/trustzone_images/build/ms
```

2. Run the commands described in Table 3-6 to build all images.

Table 3-6 TZ build commands

Build environment	Build command

Linux	▪ Build images –
	`build.sh CHIPSET=msm8937 devcfg sampleapp`
	▪ Clean the build –
	`build.sh CHIPSET=msm8937 devcfg sampleapp -c`

Windows	▪ Build images –
	`build.cmd CHIPSET=msm8937 devcfg sampleapp`
	▪ Clean the build –
	`build.cmd CHIPSET=msm8937 devcfg sampleapp –c`

3.3.4 Build RPM

Use the following commands to build RPM (<target_root> is the top-level directory). Ensure that the tool versions are as specified in Table 3-7.

The Compiler version is ARMCT501B94.

1. Open a command prompt and change to the following directory:
   ```
   cd <target_root>/RPM.BF.2.2/rpm_proc/build
   ```
2. Use the appropriate command from the Table 3-7 based on the build environment.

Table 3-7 RPM build commands

Build environment	Build command

Linux	▪ Build images –
	`./build_pukeena.sh`
	▪ Clean the Build –
	`./build_pukeena.sh –c`

Windows	▪ Build images –
	`build_pukeena.bat`
	▪ Clean the build –
	`build_pukeena.bat -c`

3.3.5 Build aDSP

The aDSP images are being released as binaries. Hexagon access OEMs can have their own compilation. Below build commands are the used by QTI.
Hexagon Tools Version: 8.0.07

1. Open a command prompt and change to the following directory:
   ```
   cd <target_root>/ADSP.VT.3.0/adsp_proc (For MSM*.LA.3.2.1 SP)
   ```
2. Use the appropriate command from Table 3-8 based on the build environment.

Table 3-8 aDSP build commands

Build environment	Build command

Linux	▪ Build images –
	`python ./build/build.py –c msm8937 –o all`
	▪ Clean the build –
	`python ./build/build.py –c msm8937 –o clean`

Windows	▪ Build images –
	`python build/build.py –c msm8937 –o all`
	▪ Clean the build –
	`python build/build.py –c msm8937 –o clean`

3.3.6 Build MPSS image

Use the compiler as – Hexagon v6.4.06 or later. For correct tool versions, see the Table 3-2 and set the export paths appropriately.

1. For Linux, verify that the following paths are set by referring to setenv.sh in the build.

```
<target root>/MPSS.JO.3.1/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/2.7.5/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>/MPSS.JO.3.1/modem_proc/build/ms
Perl: install XML: Parser module
```

sudo apt-get install aptitude

sudo aptitude install libxml-simple-perl

3. Depending on your build environment, use one of the following commands described in Table 3-9.

Table 3-9 MPSS build commands

Build variant	Build environment	Build commands	Comments
Kitchen sink (ALL RATs) Segment Loading Disabled.	Linux	▪ Build images – `./build.sh 8937.genns.prod -k` ▪ Clean the Build – `./build.sh 8937.genns.prod -c`	Main flavor including all RATs (W+T+G+C+L).
	Windows	▪ Build images – `build.cmd 8937.genns.prod -k` ▪ Clean the Build – `build.cmd 8937.genns.prod -c`	

3.3.7 Build WCNSS image

NOTE: WCNSS image is released as a binary and no build compilation is needed.

3.3.8 Codec processing engine (CPE) image

The CPE was introduced from WCD9330 Audio Codec to support Ultra-low power Qualcomm® Voice Activation solution. The CPE is a non-HLOS image and is stored in the application processor file system.

The CPE image is delivered as a binary and no build instructions or build loading is required as it is downloaded to WCD93xx CPE on-chip memory by codec driver using the SLIMbus interface.

3.3.9 Video image

NOTE: The VIDEO image is released as a binary and no build compilation is required.

4 Firmware programming

4.1 Required software

Table 4-1 lists the software required to program firmware images onto a target device.

Table 4-1 Required software

	Item description	Version	Source/vendor	Purpose
1	QPST	2.7.431 or later	QTI	Programming firmware images using QPST
2	QXDM Professional^{TM} (QXDM Pro)	3.14.447 or later	QTI	Programming NV Item values, reading diagnostic, and so on
3	Lauterbach TRACE32 ARMv8 License Extension	LA-3743X	Lauterbach GmbH	Programming firmware images using JTAG and applications processor debugging
4	Lauterbach TRACE32 QDSP6 License Extension	LA-3741A	Lauterbach GmbH	Modem software processor, firmware processor
5	Lauterbach TRACE32 Cortex-M3 License Extension	LA-7844X or LA-7844	Lauterbach GmbH	Programming firmware images using JTAG and RPM debugging using JTAG
6	Lauterbach TRACE32 ARM9 License Extension	LA-7742X	Lauterbach GmbH	Venus and WCNSS debugging using JTAG
7	Lauterbach TRACE32 Windows	August 2012 Software version – S.2014.12.000058805X Build – 58805. December 2, 2014 Podbus (58805)	Lauterbach GmbH	Programming firmware images and debugging using JTAG
8	Android SDK tools (Host USB drivers, adb, fastboot)	r10 or higher ADB 1.0.31 or later	Android Open Source Project	Windows host USB driver for adb and fastboot; adb and fastboot tools for Windows
9	QTI USB network driver combo	1.00.37 or later	QTI	Windows host USB drivers for QTI composite devices

4.2 Install TRACE32

QPST must be used for firmware download; however, TRACE32 can be used when QPST download does not work. Build 58805 version of TRACE32 is the revision required for binary download and debugging. The TRACE32 links under common\t32\t32_dap\ must be used for binary download and debugging.

By default, these files assume that the TRACE32 installing directory is C:\T32. If TRACE32 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 Install Android ADB, Fastboot, and USB driver for Windows

1. From non-HLOS build, copy the drivers from \LINUX\android\vendor\qcom\proprietary\usb\host\windows\prebuilt folder (all items from prebuilt folder) to Windows C:\Windows\System32\ and C:\Windows\SysWOW64" folders.
2. Go to Device Manager and follow the instructions under the following section *Troubleshooting for Windows* to load the drivers for ADB.
   a. Reboot the machine.
   b. Connect the device after reboot and wait till Windows finishes installing drivers for the new device.
   c. Ensure that the adb interface is up on the Windows Device Manager (right-click **My computer > Manage > Device Manager > ADB Interface > Android Composite ADB Interface > Modems > Network adapters > Ports**).

If driver installation procedures are performed correctly, there should not be any yellow mark on the ADB interface (Android Composite ADB Interface), under modems, network adapters, and ports.

Troubleshooting for Windows

If you see a yellow mark on the ADB interface in Windows Device Manager, do the following:

1. Double-click **Android Composite ADB Interface** and go the **Driver** tab.
2. Click **Update Driver**, select **Install from a list or specific location (Advanced)**, and then click **Next.**
3. Click **Don't search, I will choose the driver to install**, and then click **Next.**
4. Select **My Computer** and click **Next**.

5. Click **Have Disk** and then **Browse**. Specify the inf file location as the build path (for example, /LINUX/android /vendor/qcom/proprietary/usb/host/windows/prebuilt/android-drivers/i386/).
6. Click **OK** and **Next** in the installer window, which installs the driver for the ADB interface.

Other than ADB and fastboot, QTI USB Host drivers can be installed from the following path: https://createpoint.qti.qualcomm.com/tools/#

4.4 Install adb and fastboot in Linux

From non-HLOS build, copy the drivers from
\LINUX\android\vendor\qcom\proprietary\usb\host\windows\prebuilt
to /system/bin of the Linux machine.

1. Verify that the fastboot has properly flashed the Android images to the target, and then type the following command:

```
sudo fastboot devices
```

2. Verify that the device is displayed by fastboot.

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

4.5 Program procedures

This section describes the procedures to be used to program and reprogram each firmware image and device.

4.5.1 Program eMMC with QFIL

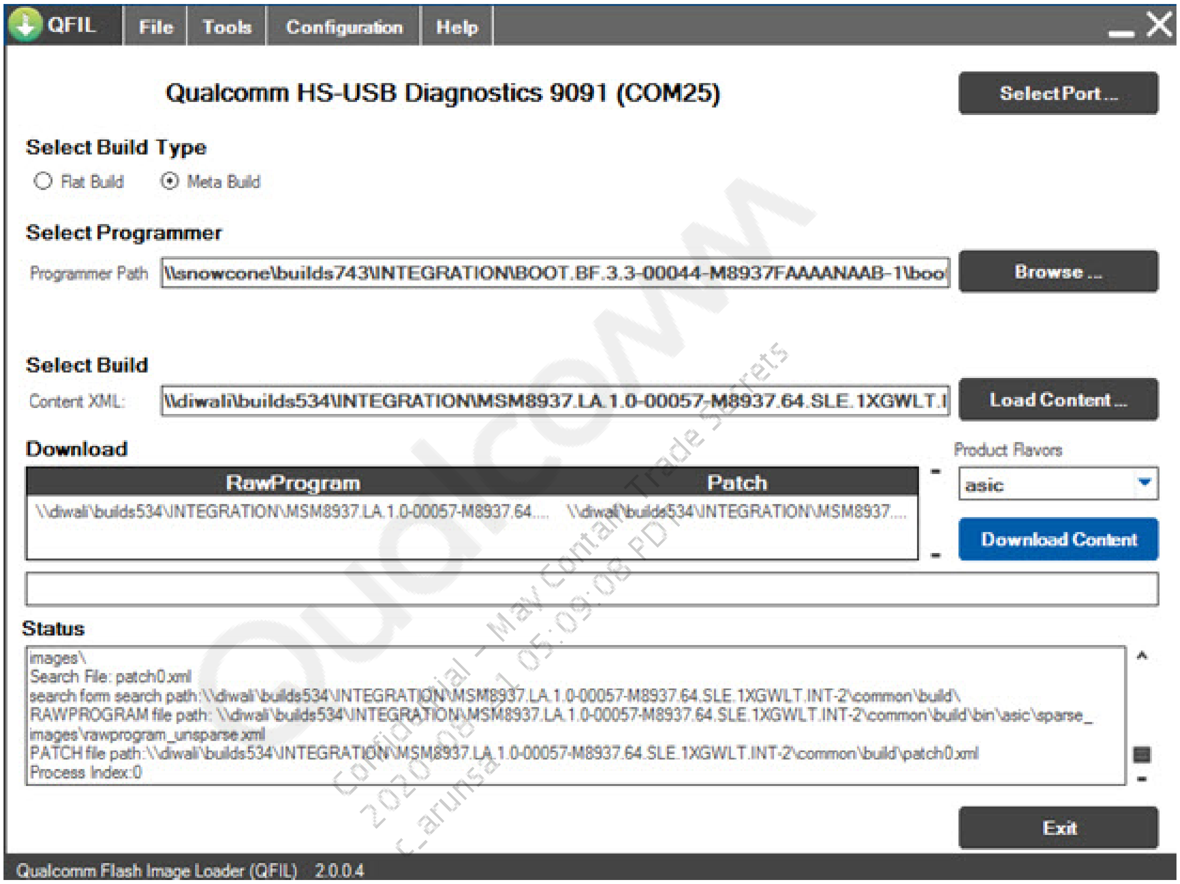
Qualcomm Flash Image Loader (QFIL) tool is used to download the software on to the reference device. The Details about the build loading procedure is mentioned below:

1. Ensure QPST, QXDM Pro are closed and J-Tag or TRACE32 is disconnected from the setup.
2. Launch QFIL from the Start menu. QFIL automatically places the device in EDL (Emergency Download Mode) and USB 9008 port is enumerated in Windows Device Manager. If the USB 9008 port is not enumerated, manually place the device in EDL using one of the steps below:
   - Erase eMMC using TRACE32

     (or)
   - Open the command prompt and run the following commands:

```
C:\>adb reboot bootloader
C:\>fastboot devices
     65144579 fastboot
C:\>fastboot erase sbl1
     erasing 'sbl1'...
```

```
            OKAY [ 0.031s]
            finished. total time: 0.047s
        C:\>fastboot reboot
            rebooting...
            finished. total time: 0.016s
```



Qualcomm HS-USB Diagnostics 9091 (COM25)

Select Build Type

○ Flat Build ● Meta Build

Select Programmer

Programmer Path: \\snowcone\builds743\INTEGRATION\BOOT.BF.3.3-00044-M8937FAAAANAAB-1\boo

Select Build

Content XML: \\diwali\builds534\INTEGRATION\MSM8937.LA.1.0-00057-M8937.64.SLE.1XGWLT.I

Download

Product Flavors: asic

RawProgram	Patch
\\diwali\builds534\INTEGRATION\MSM8937.LA.1.0-00057-M8937.64....	\\diwali\builds534\INTEGRATION\MSM8937....

Status

```
images\
Search File: patch0.xml
search form search path:\\diwali\builds534\INTEGRATION\MSM8937.LA.1.0-00057-M8937.64.SLE.1XGWLT.INT-2\common\build\
RAWPROGRAM file path: \\diwali\builds534\INTEGRATION\MSM8937.LA.1.0-00057-M8937.64.SLE.1XGWLT.INT-2\common\build\bin\asic\sparse_
images\rawprogram_unsparse.xml
PATCH file path:\\diwali\builds534\INTEGRATION\MSM8937.LA.1.0-00057-M8937.64.SLE.1XGWLT.INT-2\common\build\patch0.xml
Process Index:0
```

Qualcomm Flash Image Loader (QFIL) 2.0.0.4

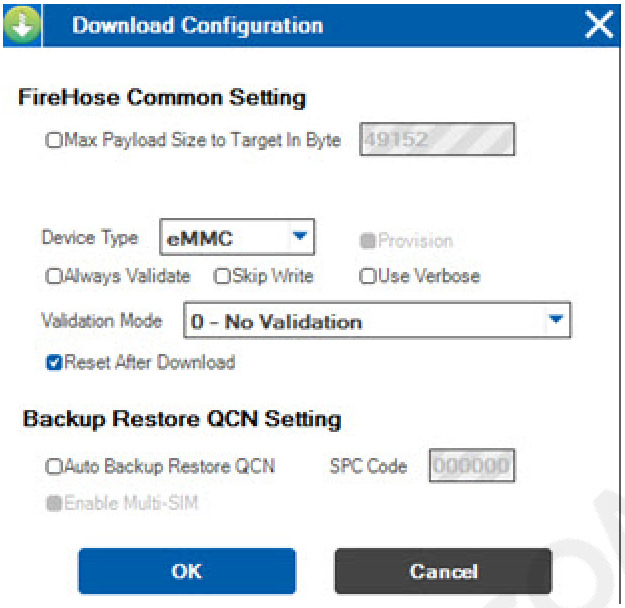
3. Click **Load Content**… and select meta build's contents.xml. The **Programmer Path** gets automatically detected;

   (Or)

   If the programmer path does not populate automatically, click **Browse** and select programmer path found in the following folder: boot_images\build\ms\bin\FAADANAZ\prog_emmc_firehose_8937_ddr.mbn

4. Click **Download Content**.

NOTE: In order to auto reset device after flashing the build, navigate to "Configuration Tab → FireHose Configuration" and Enable "Reset After Download option" as shown below:



FireHose Common Setting

□Max Payload Size to Target In Byte 49152

Device Type eMMC ● Provision

□Always Validate □Skip Write □Use Verbose

Validation Mode 0 - No Validation

☑Reset After Download

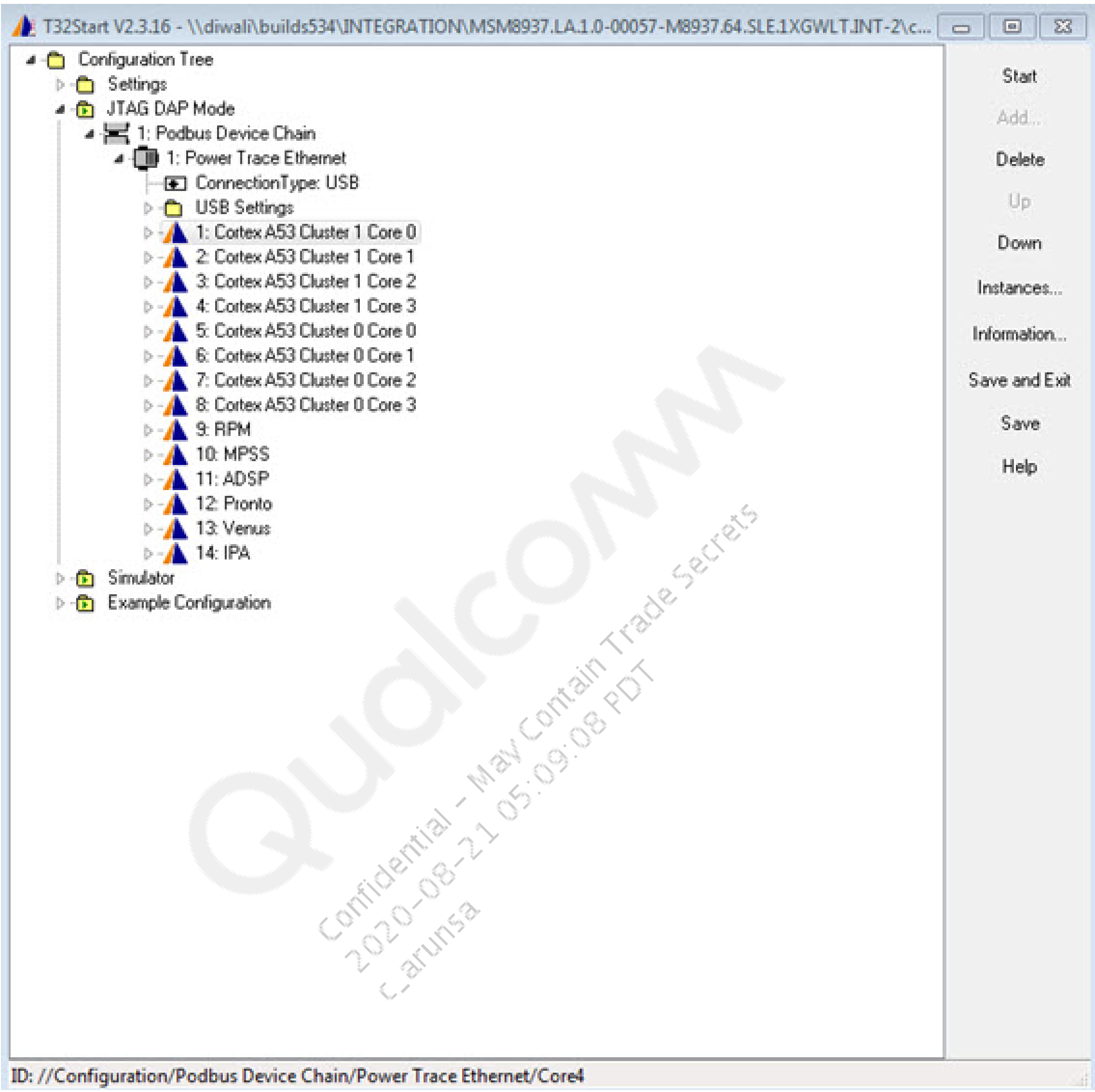
Backup Restore QCN Setting

□Auto Backup Restore QCN SPC Code 000000

● Enable Multi-SIM

4.5.2 Program eMMC with TRACE32

1. Open build location <META ROOT>\common\Core\t32\msm8937\t32start.cmd
2. Under JTAG DAP mode → Podbus device chain → Power Trace Ethernet, select : 1. CortexA53 Cluster1 Core0 and click **Start**.

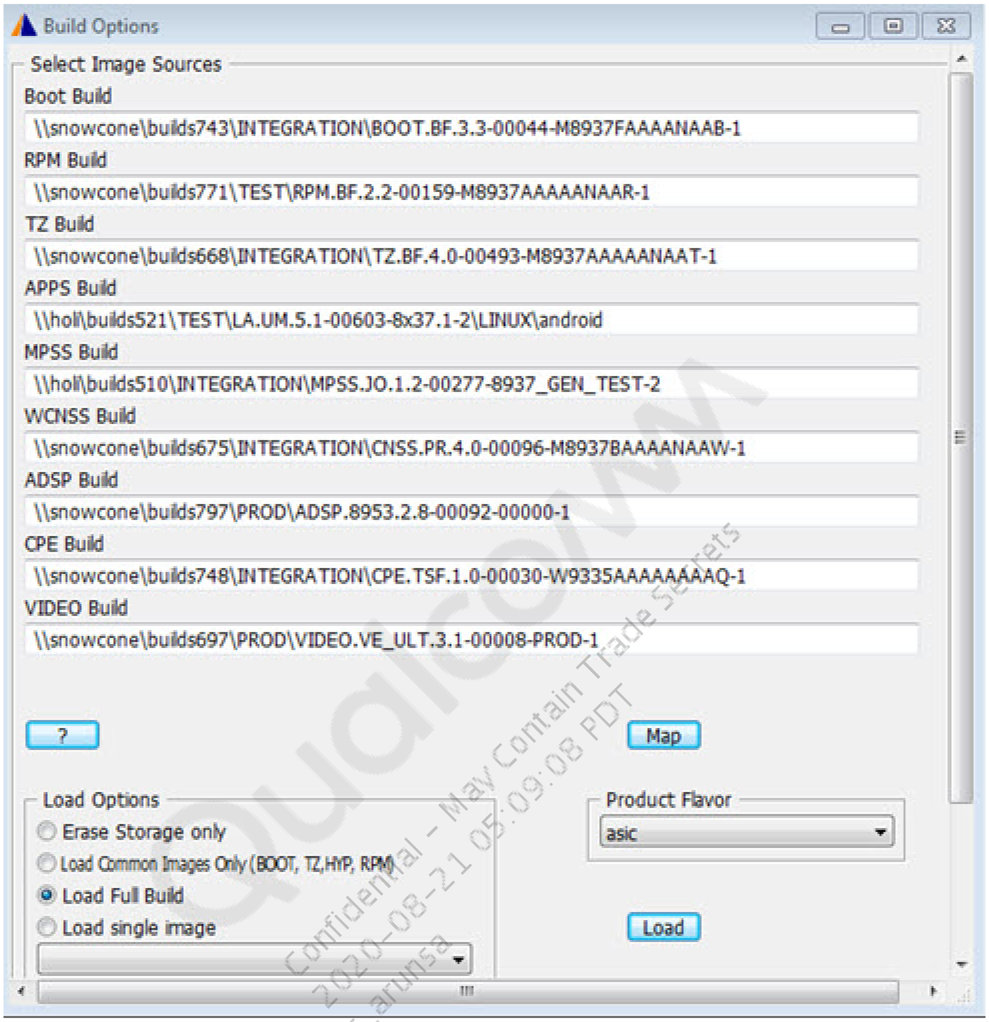


T32Start V2.3.16 - \\diwali\builds534\INTEGRATION\MSM8937.LA.1.0-00057-M8937.64.SLE.1XGWLT.INT-2\c...

Configuration Tree
Settings
JTAG DAP Mode
1: Podbus Device Chain
1: Power Trace Ethernet
ConnectionType: USB
USB Settings
1: Cortex A53 Cluster 1 Core 0
2: Cortex A53 Cluster 1 Core 1
3: Cortex A53 Cluster 1 Core 2
4: Cortex A53 Cluster 1 Core 3
5: Cortex A53 Cluster 0 Core 0
6: Cortex A53 Cluster 0 Core 1
7: Cortex A53 Cluster 0 Core 2
8: Cortex A53 Cluster 0 Core 3
9: RPM
10: MPSS
11: ADSP
12: Pronto
13: Venus
14: IPA
Simulator
Example Configuration

ID: //Configuration/Podbus Device Chain/Power Trace Ethernet/Core4

3. Click the "APPS COMMANDS" tab → Build Options.
4. Select Product Flavor as “**ASIC**” and click **MAP**.
5. Click **Load**.



Select Image Sources

Boot Build
\\snowcone\builds743\INTEGRATION\BOOT.BF.3.3-00044-M8937FAAAANAAB-1

RPM Build
\\snowcone\builds771\TEST\RPM.BF.2.2-00159-M8937AAAAANAAR-1

TZ Build
\\snowcone\builds668\INTEGRATION\TZ.BF.4.0-00493-M8937AAAAANAAT-1

APPS Build
\\holi\builds521\TEST\LA.UM.5.1-00603-8x37.1-2\LINUX\android

MPSS Build
\\holi\builds510\INTEGRATION\MPSS.JO.1.2-00277-8937_GEN_TEST-2

WCNSS Build
\\snowcone\builds675\INTEGRATION\CNSS.PR.4.0-00096-M8937BAAAANAAW-1

ADSP Build
\\snowcone\builds797\PROD\ADSP.8953.2.8-00092-00000-1

CPE Build
\\snowcone\builds748\INTEGRATION\CPE.TSF.1.0-00030-W9335AAAAAAAAQ-1

VIDEO Build
\\snowcone\builds697\PROD\VIDEO.VE_ULT.3.1-00008-PROD-1

Load Options
○ Erase Storage only
○ Load Common Images Only (BOOT, TZ,HYP, RPM)
● Load Full Build
○ Load single image

Product Flavor
asic

6. After TRACE32 finishes programming/flashing boot loaders to eMMC the phone enters fastboot mode and binaries like NON-HLOS.bin and APSS binaries are pushed onto the target device using the script **`fastboot_all.py.`**

   Note: Keep the USB connected to PC as it uses fastboot to flash non-HLOS.bin and application processor binaries. Fastboot download process starts automatically using a script after TRACE32 finishes programming boot loaders.

7. After fastboot completes flashing binaries, power cycle the device using command `fastboot reboot.`

You should see adb devices in the device manager (Android boots up on the phone) or device will be listed with **adb devices** (Android boots up on the phone). The phone will be visible in QPST/QXDM.

4.5.3 Program eMMC using Fastboot

NOTE: Before programming the system images using Fastboot, the device should be flashed at least once using the procedure mentioned in Section 4.5.1 or 4.5.2:

1. Plug the USB cable into the target. Ensure the phone is in fastboot mode
2. Depending on your environment, choose one of the following options:
   - From Windows, in command shell, run:

     ```
     fastboot devices
     ```

   - From Linux, run the following command:

     ```
     sudo fastboot devices
     ```

   A 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_complete.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 tz <path to tz.mbn>
fastboot flash devcfg <path to devcfg.mbn>
fastboot flash dsp <path to adspso.bin>
fastboot flash adsp <path to dsp2.mbn>
fastboot flash aboot <path to emmc_appsboot.mbn >
fastboot flash cmnlib <path to cmnlib.mbn >
fastboot flash cmnlib64 <path to cmnlib64.mbn >
fastboot flash keymaster <path to keymaster.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>
fastboot flash cache <path to cache.img>
```

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

4.5.4 Flash applications to Android using ADB

1. Plug the USB cable into the target.
2. Enter the following command to register a device:
   - □ Linux – `sudo adb devices`
   - □ Windows – `adb devices`
3. Navigate to the directory containing the application apk:
   - □ Linux – Copy the files as follows:

     ```
     cp package.apk AppName.apk
     ```

   - □ Windows – 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>. Else, `adb install` command can also be used.

5 Operational guide

For common NV settings (such as RF NV settings, WCDMA/GSM + GSM, CDMA + GSM) and Call configuration, see *MSM89x7 RF Software Overview* (80-P2485-3).

GPS configuration

GNSS SubSysGNSS DLL v1.0.44 or higher is required to perform offline RF development. Running offline RF Dev requires QPSR, see *IZat Gen 8 Engine Family RF Development Test Procedures* (80-VM522-2) for more details.

Multimedia configuration

For audio configuration and debugging, see:

- *QM215 Audio Bringup* Guide (80-PK881-51)
- Salesforce solutions:
  - 00031062
  - 00031061
  - 00031105

Display

For display panel bringup and driver porting-related information, see *DSI Programming Guide for B-Family Android Devices* (80-NA157-174). It describes application usage of the Display Serial Interface (DSI) panel bringup for the Android OS. It also provides sample code and PLL calculation pertaining to the DSI Mobile Industry Processor Interface (MIPI) panel bringup.

- For details on Linux Android display driver porting, see *Linux Android Display Driver Porting Guide* (80-NN766-1).
- For details on display bring-up and debug, see:
  - *Android Display Debug Guide* (80-NP925-1)
  - *Multimedia Driver Development and Bringup Guide – Display* (80-NU323-3)

Camera

Android default camera application is used to verify the camera features. For details related to sensor driver porting and migration, see *Multimedia Driver Development and Bringup Guide – Camera* (80-NU323-2).

For techniques on debugging different kind of camera errors, see *Linux Camera Debugging Guide* (80-NL239-33). Additional debugging steps are covered in *Multimedia Driver Development and Bringup Guide – Camera* (80-NU323-2)

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.

Additional bring up and debugging steps are covered in:

- *Multimedia Driver Development and Bringup Guide – Video* (80-NU323-5)
- *Android Video Debug Guide* (80-NU339-1)

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 *WCN36X0 WLAN/BT/FM In FTM Guide With QRCT Test Example* (80-WL300-27).

5.1 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, and so on), and a central root, the Applications Processor (AP). The clients of these individual subsystems that run 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,and so on.

The use case for this feature is a catastrophic restart, that is, 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:

On Android targets where SSR is enabled, the restart status and statistics for a subsystem are located here

```
ls /sys/bus/msm_subsys/devices/
```

- To find a specific subsystem

  ```
  cat /sys/bus/msm_subsys/devices/subsysX/name  (here X = 0,1,2 for
  different subsystems modem, wcnss and so on)
  ```

- To know if SSR is enabled on a specific subsystem. SSR for a subsystem is enabled if the restart_level is set to RELATED

```
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, see the following documents:

- *Subsystem Restart User Guide* (80-N5609-2)
- *MSM8x10 Android Subsystem Restart Overview* (80-NC839-21)
- *MSM8974 Android Subsystem Restart* (80-NA157-31)

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 *QRD BRF User Guide* (80-NF136-1).

6.2 FactoryKit

FactoryKit is an Android application used for MMI test. It functions similar to FastMMI but with a longer bootup time. FactoryKit is used on customer devices and all Qualcomm® Reference Design (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, for example, cross-compilers
packages/ – Standard Android Java applications and components
development/ – Android reference applications and tools for developers
hardware/ – HAL (audio, sensors) and QTI specific hardware wrappers
vendor/qcom/ – QTI target definitions, for example, msm7201a_surf
vendor/qcom-proprietary/ – QTI proprietary components, for example, MM, QCRIL,
and so on.
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 Build Linux kernel manually

1. Change directory to the main Android directory.
2. Set up the Android build environment:

```
source build/envsetup.sh
lunch msm8937_64-userdebug (64-bit kernelspace and 64-bit user space)
```

or

```
lunch msm8937_32-userdebug (32-bit kernelspace and 32-bit user space)
```

3. Build the kernel image with the following command:

```
make kernel
```

The resulting kernel image appears in out/target/product/msm8952_64/boot.img

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

a. To remove object files:

```
make clean
```

b. To remove all the generated files:

```
make distclean
```

A.3 Build Android manually

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

     ```
     m clean-<module name>
     ```

   - □ To delete a module within a given path, use the following 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, and so on.
  - Predefined action for compiling executables, shared libraries, static libraries, Android packages, using precompiled binaries, and so on.
- 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:

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 must 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, ensure that you also set LOCAL_UNSTRIPPED_PATH if it is an executable or a shared library, so that the unstripped binary also has somewhere to go; otherwise, an error occurs.

B References

B.1 Related documents

NOTE: Few of the documents listed as references are not yet published and will be released between ES to CS timeframe of the QM215 chipset.

Title	Number
Qualcomm Technologies, Inc.	
Hexagon Tools Installation Guide	80-N2040-32
Hexagon Development Tools Overview	80-N2040-12
USB Host Driver for Windows 2000/Windows XP User Guide	80-V4609-1
USB Host Driver Installation Instructions for Microsoft Windows	80-VP092-1
IZat Gen 8 Engine Family RF Development Test Procedures	80-VM522-2
Qualcomm CreatePoint User Guide	80-NC193-2
Qualcomm Flash Image Loader (QFIL) User Guide	80-NN120-1
Qualcomm Hexagon LLVM C/C++ Compiler User Guide	80-VB419-89
LLVM Compiler Hexagon Processor Deployment Plan	80-VB419-87
DSI Programing Guide for B-Family Android Devices	80-NA157-174
Linux Android Display Driver Porting Guide	80-NN766-1
WCN36X0 WLAN/BT/FM In FTM Guide With QRCT Test Example	80-WL300-27
QRD BRF User Guide	80-NF136-1
Linux Camera Debugging Guide	80-NL239-33
Widevine DRM	80-N9340-1
Subsystem Restart User Guide	80-N5609-2
MSM8x10 Android Subsystem Restart Overview	80-NC839-21
MSM8974 Android Subsystem Restart	80-NA157-31
SGLTE Device Configuration	80-NJ017-11
Segment Loading Feature	80-NL239-46
Android Display Debug Guide	80-NP925-1
Multimedia Driver Development and Bringup Guide – Display	80-NU323-3
Multimedia Driver Development and Bringup Guide – Camera	80-NU323-2
Multimedia Driver Development and Bringup Guide – Video	80-NU323-5
Android Video Debug Guide	80-NU339-1
Snapdragon Software Product Family	80-P3255-31
QM215 Linux Android Software Thermal Management Overview	80-PK881-35
QM215 TrustZone and Security Overview	80-PK881-40

Title	Number
QM215 Audio Bringup Guide	80-PK881-51
QM215 External MI2S Interface	80-PK881-54
QM215 Modem Software Overview	80-PK881-96
QM215 RF Software Overview	80-PK881-XX
Resources	
Android Open Source Project Page	
Android Developer Resources	
Android Source Download and System Setup	
Code Aurora Forum	
Installing Repo	
Qualcomm ChipCode website	

B.2 Acronyms and terms

Acronym or term	Definition
CDT	Configuration Data Table
CPE	Code processing engine
QPST	Qualcomm Product Support Tool
SSR	Subsystem Restart