Qualcomm Technologies, Inc.

QCA_Networking_2022.SPF.12.2 CSU1

Release Notes

80-57835-4 Rev. AB

August 24, 2023

For additional information or to submit technical questions, go to: https://createpoint.qti.qualcomm.com

Confidential – Qualcomm Technologies, Inc. and/or its affiliated companies – May Contain Trade Secrets

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

Revision history

Revision	Date	Description
AA	August 2023	Initial release
AB	August 2023	Corrected typos in the commands in 4.4.3.2 and 5.4.3.2 sections

Contents

1 Introduction ... 6

1.1 Purpose ... 7
1.2 Related documentation ... 8
1.3 Frequency spectrum and regulatory domain support ... 11

2 IPQ9574.ILQ.12.2 CSU1 ... 13

2.1 Supported hardware for this SP ... 13
2.2 Build and load the image for IPQ9574.ILQ.12.2 CSU1 ... 14
2.2.1 Download packages available through the Qualcomm ChipCode Portal ... 14
2.2.2 Download packages from external websites ... 15
2.2.3 Generate the Qualcomm® Internet Processor (IPQ) firmware for IPQ9574.ILQ.12.2 CSU1 release ... 24
2.2.4 Generate a complete firmware image ... 33
2.2.5 Flash the complete default software image ... 35
2.2.6 Create customized IPQ9574 + QCN90xx Wi-Fi firmware images ... 36
2.2.7 Create customized IPQ9574 Wi-Fi firmware images ... 37
2.2.8 Create customized QCN9274 Wi-Fi firmware images ... 38
2.3 Flash Wi-Fi firmware image only ... 39

3 IPQ5018.ILQ.12.2 CSU1 ... 41

3.1 Supported features ... 41
3.2 Supported hardware for this SP ... 41
3.3 Build and load the image for IPQ5018.ILQ.12.2 CSU1 ... 42
3.3.1 Download packages available through Qualcomm ChipCode Portal ... 42
3.3.2 Download packages from external websites ... 43
3.3.3 Generate the firmware for IPQ5018.ILQ.12.2 CSU1 ... 43
3.3.4 Generate a complete firmware image ... 51
3.3.5 Flash the complete default software image ... 53
3.3.6 Create customized IPQ50xx Wi-Fi firmware images ... 56
3.3.7 Create customized IPQ50xx + QCN95x4 Wi-Fi firmware images ... 56
3.4 Flash Wi-Fi firmware image only ... 57
3.5 Generate secure boot image ... 58
3.6 Test the serial port profile (SPP) over generic access profile (GAP) (BR/EDR) profile with sample applications with onboard Bluetooth on AP.MP03.1 ... 58

4 IPQ8074.ILQ.12.2 CSU1 ... 62

4.1 Supported features ... 62
4.2 Restrictions on software while using it for testing ... 63

4.3 Supported hardware for this SP 64
4.4 Build and load the image for IPQ8074.ILQ.12.2 CSU1 65
4.4.1 Download packages available through Qualcomm ChipCode 65
4.4.2 Download packages from external websites 65
4.4.3 Generate the firmware for IPQ8074.ILQ.12.2 CSU1 65
4.4.4 Generate a complete firmware image 77
4.4.5 Flash the complete default software image 78
4.4.6 Create customized IPQ807x Wi-Fi firmware images 82
4.4.7 Create customized IPQ807x + QCN90xx Wi-Fi firmware images 84
4.5 U-Boot device tree optimization 85

5 IPQ6018.ILQ.12.2 CSU1 86
5.1 Supported features 86
5.2 Restrictions on software while using it for testing 87
5.3 Supported hardware for this SP 87
5.4 Build and load the image for IPQ6018.ILQ.12.2 CSU1 87
5.4.1 Download packages available through Qualcomm ChipCode Portal 88
5.4.2 Download packages from external websites 88
5.4.3 Generate the firmware for IPQ6018.ILQ.12.2 CSU1 88
5.4.4 Generate a complete firmware image 96
5.4.5 Flash the complete default software image 98
5.4.6 Create customized IPQ60x8 Wi-Fi firmware images 100
5.4.7 Create customized IPQ60x8 + QCN90xx Wi-Fi firmware images 101
5.5 Flash Wi-Fi firmware image only 102
5.6 Test the GATT profile with sample applications with onboard CSR8811 Bluetooth on AP.CP01 103

6 IPQ5322.ILQ.12.2 CSU1 106
6.1 Supported hardware for this SP 106
6.2 Build and load the image for IPQ5322.ILQ.12.2 CSU1 106
6.2.1 Download packages available through the Qualcomm ChipCode Portal 106
6.2.2 Download packages from external websites 107
6.2.3 Generate the IPQ firmware for IPQ5322.ILQ.12.2 CSU1 release 107
6.2.4 Generate a complete firmware image 116
6.2.5 Flash the complete default software image 120
6.3 Flash Wi-Fi firmware image only 121
6.4 IPQ5322 features delivered 122
6.4.1 WLAN deliverables 122
6.5 QCN9160 Scan Radio Support 126

7 QCN9074 scan radio support 127

8 QCN9274 features delivered 128
8.1 WLAN deliverables 128

9 Known issues 133

9.1 IPQ9574.ILQ.12.2 – IPQ9574 + QCN9274 133
9.1.1 Stability issues 133
9.1.2 Performance issues 135
9.1.3 Functional issues 136
9.2 Legacy platforms 139
9.2.1 Stability issues 139
9.2.2 Performance issues 139
9.2.3 Functional issues 140
9.3 IPQ5322.ILQ.12.2 – IPQ5322 + QCN9160 140
9.3.1 Stability issues 140
9.3.2 Performance issues 141
9.3.3 QCN9160 specific issues 143

10 Known limitations 144
10.1 IPQ9574.ILQ.12.2 147
10.2 IPQ5322.ILQ.12.2 148

11 QDART 150
11.1 QDART reference documents 150

12 FTM calibration and verification KPI 152
12.1 Factors impacting KPI 152
12.2 RDP433 153
12.3 RDP419 156
12.4 RDP413 159
12.5 RDP361 161
12.6 RDP432 163
12.7 RDP418 166
12.8 RDP441 168

13 FTM and PHYRF tuning 171

14 BDF updates 173
14.1 IPQ8074 173
14.2 IPQ5018_QCN6122 174
14.3 QCN90xx 174
14.4 IPQ6018 175
14.5 IPQ9574 175
14.6 QCN9274 176
14.7 IPQ5322 177
14.8 QCN9160 178

15 Additional information on WFA certification 179

1 Introduction

This document provides details on the QCA_Networking_2022.SPF.12.2 CSU1 software release. This release supports QSDK on Linux-5.4 and OpenWrt-19.07 based code.

Whether downloaded from the Qualcomm ChipCode™ portal, the Qualcomm® CreatePoint site, or embedded on Equipment received from Qualcomm Atheros, Inc. ("QCA") or its affiliates, the QCA_Networking_2022.SPF.12.2 CSU1 software release (the "SW Package") shall be considered (in order of priority): (i) Evaluation Technology under the terms of the product kit license agreement accompanying the release (the "PKLA"), (ii) Deliverables under the terms of your Limited Use Agreement (the "LUA"), or (iii) Licensed Technology under the terms of your Technology License Agreement (the "TLA"), each with QCA or its affiliate (the LUA, PKLA, or TLA, as applicable, the "Agreement"). The applicable period for which the SW Package is licensed (the "Use Period") starts on the Effective Date of your Agreement or the date you received the SW Package, whichever is later, and expires on August 15, 2024 (unless a different Use Period for the SW Package is specified in the Agreement, in which case the Use Period in the Agreement shall prevail). By receiving and/or using the SW Package, you acknowledge and agree that your use of the SW Package is subject to the terms and conditions of the Agreement. If you do not agree to the terms of the Agreement, have not accepted any such Agreement, or your agreement with QCA or its affiliate does not include Deliverables, Evaluation Technology, or Licensed Technology, you shall immediately delete the SW Package from all storage media and destroy any and all copies made.

Information published by QCA or its affiliates regarding any third-party information does not constitute a license to use such information or endorsement thereof. QCA or its affiliates provides any such third-party information as-is, without any representation, warranty, or indemnity, either express or implied. Use of such information may require a license from a third party under the intellectual property rights of such third party, or a license from QCA or its affiliates under the intellectual property rights of QCA or its affiliates. Users assume all risk of any use of such third-party information.

1.1 Purpose

This document describes new and changed features, download and installation procedures, and known and resolved problems in the hardware and software. This product includes software developed by the University of California, Berkeley, and its contributors.

This release aggregates these SPs:

- IPQ9574.ILQ.12.2.r3
- IPQ5018.ILQ.12.2.r3
- IPQ8074.ILQ.12.2.r3
- IPQ6018.ILQ.12.2.r3
- IPQ5322.ILQ.12.2.r3

This release is:	**QCA_Networking_2022.SPF.12.2 CSU1**
The release version is:	IPQ9574.ILQ.12.2.r3-00015-P-1 IPQ5018.ILQ.12.2.r3-00015-P-1 IPQ8074.ILQ.12.2.r3-00015-P-1 IPQ6018.ILQ.12.2.r3-00015-P-1 IPQ5322.ILQ.12.2.r3-00015-P-1
The open-source label (CLO_TAG) that corresponds to this release is:	AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml
Qualcomm ChipCode™ distribution tag (Use this tag to check out the code from git repository):	r12.2.r3_00009.0

Software products and software image names included with this release:

Software Product Name	Included Software Images
IPQ9574.ILQ.12.2.r3	NHSS.QSDK.12.2.r4-00023-P-1
	WLAN.WBE.1.1.r6-00052-QCAHKSWPL_SILICONZ-1
	WLAN.HK.2.9.r3-00031-QCAHKSWPL_SILICONZ-1
	BOOT.XF.0.3.1.1-00078-IPQ90xxLZB-1
	TZ.WNS.5.3-00264-IPQ90xxAAAAANAZT-1
	TMEL.WNS.1.0-00062-IPQ95xxAAAAANAZT-1
	RPM.BF.2.4.1-00116-IPQ8074AAAAANAZR-6
IPQ8074.ILQ.12.2.r3	NHSS.QSDK.12.2.r4-00023-P-1
	WLAN.HK.2.9.r3-00031-QCAHKSWPL_SILICONZ-1
	TZ.BF.4.0.8-00243-IPQ807xSANAANAZT-1
	BOOT.BF.3.3.1-00172-IPQ8074HAASANAA-1
	NSS.FW.12.2-00149-ALL-1
	RPM.BF.2.4.1-00116-IPQ8074AAAAANAZR-6
	CNSS.PS.3.19-00001-S-1
	WLAN.BL.3.19-00001-S-1
IPQ5018.ILQ.12.2.r3	NHSS.QSDK.12.2.r4-00023-P-1

Software Product Name	Included Software Images
	WLAN.HK.2.9.r3-00031-QCAHKSWPL_SILICONZ-1
	TZ.WNS.4.0-00129-IPQ5018SANAANAZT-1
	BOOT.BF.3.3.1.1-00081-IPQ5018HAASANAA-2
	NSS.FW.12.2-00149-ALL-1
	CNSS.PS.3.19-00001-S-1
	WLAN.BL.3.19-00001-S-1
	BTFW.MAPLE.1.0.0-00101-MPL_ROM_PATCHZ-1
IPQ6018.ILQ.12.2.r3	NHSS.QSDK.12.2.r4-00023-P-1
	WLAN.HK.2.9.r3-00031-QCAHKSWPL_SILICONZ-1
	TZ.WNS.5.1-00183-IPQ60xxAAAAANAZT-1
	BOOT.XF.0.3-00107-IPQ60xxLZB-1
	NSS.FW.12.2-00149-ALL-1
	RPM.BF.2.4.1-00116-IPQ8074AAAAANAZR-6
	CNSS.PS.3.19-00001-S-1
	WLAN.BL.3.19-00001-S-1
IPQ5322.ILQ.12.2.r3	NHSS.QSDK.12.2.r4-00023-P-1
	WLAN.WBE.1.1.r6-00052-QCAHKSWPL_SILICONZ-1
	BOOT.XF.0.3.1.1-00078-IPQ90xxLZB-1
	TZ.WNS.5.3-00264-IPQ90xxAAAAANAZT-1
	TMEL.WNS.1.1-00076-IPQ53xxAAAAANAZT-1

1.2 Related documentation

DCN	Doc Title
80-19274-1	*QCN9274 WI-FI 7 4S/320 MHZ PCIE RADIO WITH FLEXIBLE DUAL RADIO CONFIGURATION DATA SHEET*
80-19275-1	*QCN9272 WI-FI 7 2S/320 MHz PCIe RADIO WITH FLEXIBLE DUAL RADIO CONFIGURATION DATA SHEET*
80-19276-1	*QCN6274 WI-FI 7 4S/320 MHZ PCIE RADIO WITH FLEXIBLE DUAL RADIO CONFIGURATION DATA SHEET*
80-19574-1	*IPQ9574 WI-FI ACCESS POINT SOC DATA SHEET*
80-19576-1	*IPQ9576 WI-FI ACCESS POINT SOC DATA SHEET*
80-19575-1	*IPQ9570 WI-FI ACCESS POINT SOC DATA SHEET*
80-33881-6	*QCN9274 HARDWARE REFERENCE GUIDE*
80-33881-1	*IPQ957X AP.AL02.3 + QCN9274 HARDWARE REFERENCE (RDP0433)*
80-33881-4	*IPQ9574 AP.AL02 + QCN9274 (RDP433) SETUP GUIDE*
80-33881-5	*IPQ9574 AP.AL02 + QCN9274 (RDP454) SETUP GUIDE*
80-49156-4	*IPQ955X AP.AL03.2 + QCN9274 (RDP0458) HARDWARE REFERENCE*
80-49156-7	*IPQ955X AP.AL03.2 + QCN9274 (RDP0458) SETUP GUIDE*
80-33881-20	*IPQ9574 AP.AL02.9 + QCN9274 + QCN9074 SCAN RADIO (RDP459) HARDWARE REFERENCE*
80-33881-8	*IPQ9574 AP.AL02.9 + QCN9274 + QCN9074 SCAN RADIO (RDP459) SETUP GUIDE*
80-19560-8	*IPQ95XX SOC SOFTWARE USER GUIDE*
80-19560-3A	*WIRELESS LAN DRIVER VERSION 12.X FOR ACCESS POINTS: DRIVER ARCHITECTURE PROGRAMMING GUIDE*

DCN	Doc Title
80-19560-3B	*WIRELESS LAN DRIVER VERSION 12.X FOR ACCESS POINTS: PLATFORM COMPONENTS PROGRAMMING GUIDE*
80-19560-3C	*WIRELESS LAN DRIVER VERSION 12.X FOR ACCESS POINTS: FAST PATH AND NETWORKING SERVICES PROGRAMMING GUIDE*
80-19560-3D	*WIRELESS LAN DRIVER VERSION 12.X FOR ACCESS POINTS: MEMORY AND PERFORMANCE OPTIMIZATION PROGRAMMING GUIDE*
80-19560-3E	*WIRELESS LAN DRIVER VERSION 12.X FOR ACCESS POINTS: MESH PROGRAMMING GUIDE*
80-19560-3F	*WIRELESS LAN DRIVER VERSION 12.X FOR ACCESS POINTS: HOST SOFTWARE PROTOCOLS PROGRAMMING GUIDE*
80-19560-3G	*WIRELESS LAN DRIVER VERSION 12.X FOR ACCESS POINTS: DATA PATH FUNCTIONALITIES PROGRAMMING GUIDE*
80-19560-3H	*WIRELESS LAN DRIVER VERSION 12.X FOR ACCESS POINTS: DIAGNOSTIC TOOLS, STATISTICS, AND LOGGING DEBUG GUIDE*
80-19560-3J	*WIRELESS LAN DRIVER VERSION 12.X FOR ACCESS POINTS: AP MODES AND REPEATER FUNCTIONS PROGRAMMING GUIDE*
80-19560-3K	*WIRELESS LAN DRIVER VERSION 12.X FOR ACCESS POINTS: HOST SECURITY FUNCTIONALITIES PROGRAMMING GUIDE*
80-19560-3L	*WIRELESS LAN DRIVER VERSION 12.X FOR ACCESS POINTS: PHY RF TUNING REFERENCE GUIDE*
80-19560-2	*WIRELESS LAN DRIVER VERSION 12.X FOR ACCESS POINTS: COMMAND REFERENCE*
80-33884-6	*ENABLING SECURE BOOT IN QCN92XX CHIPSETS*
80-49340-10	*QCA_NETWORKING_2022.SPF.12.2 QUICK START GUIDE*
80-YB215-1	*FTM RF TEST FOR 802.11AX WLAN AP CHIPSETS USER GUIDE*
80-YB215-4	*BOARD DATA FILE UTILITY FOR 802.11AX WLAN AP CHIPSETS APPLICATION NOTE*
80-YB478-10	*NOISE FLOOR CALIBRATION FOR 802.11AX WLAN AP CHIPSETS USER GUIDE*
80-YB215-7	*FTM QMSL APIS FOR 802.11AX WLAN AP CHIPSETS USER GUIDE*
80-YB215-6	*WLAN TLV2 COMMANDS FOR FTM DIAGNOSTICS OF 802.11AX WLAN AP CHIPSETS REFERENCE MANUAL*
80-YB215-9	*BOARD ID FUSING FOR 802.11AX WLAN AP CHIPSETS APPLICATION NOTE*
80-Y9005-19	*QCN90XX MANUFACTURING TEST RECOMMENDATION APPLICATION NOTE*
80-YB215-8	*CTL ASSISTANT TOOL FOR 802.11AX CHIPSETS USER GUIDE*
80-33882-10	*CTL ENGINE FOR 802.11BE WLAN CHIPSETS QUICK START GUIDE*
80-33882-8	*CTL ENGINE FOR 802.11BE WLAN CHIPSETS USER GUIDE*
80-33882-13	*CTL_Engine_Addendum_802.11be_WLAN_AP_Chipsets_USER GUIDE*
80-YB215-2	*MANUFACTURING TEST RECOMMENDATIONS FOR 802.11AX WLAN AP CHIPSETS APPLICATION NOTE*
80-Y6630-2	*REGULATORY_CONFIGURATION_QCA_WLAN*
80-33881-3	*DIRECT SWITCH CONNECT SUPPORT ON IPQ9574 + QCN9274 PLATFORMS APPLICATION OTE*
80-58156-1	*Fixed Wireless Access (FWA) Application GUIDE*
80-33882-1	*FTM RF TEST FOR 802.11BE WLAN AP CHIPSETS USER GUIDE*
80-33882-2	*QCN92XX/QCN62XX MYFTM TEST USER GUIDE*
80-33882-3	*MANUFACTURING TEST RECOMMENDATIONS FOR 802.11BE WLAN AP CHIPSETS PPLICATION NOTE*
80-33882-4	*WLAN TLV2 COMMANDS FOR FTM DIAGNOSTICS OF 802.11BE WLAN AP CHIPSETS REFERENCE*
80-33882-5	*FTM QMSL APIS FOR 802.11BE WLAN AP CHIPSETS USER GUIDE*

DCN	Doc Title
80-33882-6	*BOARD DATA FILE UTILITY FOR 802.11BE WLAN AP CHIPSETS USER GUIDE*
80-33882-7	*QCN92XX BOARD ID FUSING USER GUIDE*
80-33882-9	*QCN92XX BDF CONTENT CHECKLIST*
80-19559-11	*AFC DEVICE AND CLOUD SERVICES FOR 6 GHZ ON 802.11AX AND 802.11BE WLAN AP HIPSETS USER GUIDE*
80-YA809-1	*FAIL SAFE BOOT FOR WLAN AP CHIPSETS USER GUIDE*
80-41890-3	*QCA_NETWORKING_2022.SPF.12.2 QCN92XX WLAN FIRMWARE BUILD USER GUIDE*
80-YA728-4	*IPQ807X/IPQ807XA/IPQ817X + QCN90XX SOC SOFTWARE USER GUIDE*
80-16052-17	*IPQ50XX SOC SOFTWARE USER GUIDE*
80-19560-43	*IPQ95XX PCM DRIVER API REFERENCE*
80-16052-39	*IPQ50XX PCM DRIVER API REFERENCE*
80-YC609-8	*IPQ60XX SOC SOFTWARE USER GUIDE*
80-19560-4	*CONFIGURATION API (ACFG) FOR 12.X WLAN DRIVERS REFERENCE*
80-36301-2	*Wi-Fi 7 WLAN FIRMWARE PROGRAMMING GUIDE*
80-58135-2	*QCA_NETWORKING_2022.SPF.12.2 WLAN FIRMWARE QUICK START GUIDE*
80-16052-11	*802.11AX WLAN FIRMWARE PROGRAMMING GUIDE*
80-41890-4	*QCA_NETWORKING_2022.SPF.12.2 QCN62XX WLAN FIRMWARE LICENSING AND BUILD SER GUIDE*
80-58134-10	*QCA_NETWORKING_2022.SPF.12.2 FEATURES USER GUIDE*
80-33881-23	*RDP458: QCN92XX AL03.2 FCC PRE-SCAN SUMMARY*
80-19274-16	*QCN92XX/QCN62XX MLO – SINGLE NET DEVICE/WIPHY MODEL FEATURE OVERVIEW*
80-58135-1	*QCA_NETWORKING_2022.SPF.12.2 WLAN FIRMWARE BUILD USER GUIDE*
80-58134-1	*QCA_NETWORKING_2022.SPF.12.2 QUICK START GUIDE*
80-58134-20	*QCN92XX/QCN62XX WI-FI/BLE COEXISTENCE USER GUIDE*
80-19560-44	*IPQ95XX LPASS APPLICATION NOTE*
80-45481-1	*IPQ5332 WI-FI ACCESS POINT SOC DATA SHEET*
80-45482-1	*IPQ5322 WI-FI ACCESS POINT SOC DATA SHEET*
80-45483-1	*IPQ5312 WI-FI ACCESS POINT SOC DATA SHEET*
80-45484-1	*IPQ5302 WI-FI ACCESS POINT SOC DATA SHEET*
80-45481-4	*IPQ53x2 DEVICE REVISION GUIDE*
80-45481-41	*IPQ53XX AP.MI01 DESIGN CHANGE LIST*
80-45481-12	*IPQ53X2 POWER CONSUMPTION APPLICATION NOTE*
80-50180-4	*IPQ53X2 WITH QCN92XX/QCN62XX BOARD DESIGN REFERENCE*
80-50180-8	*IPQ53x2 AP.MI01.X Setup User Guide*
80-54388-1	*IPQ5322 and QCN6274 CoB Paper Design Reference*
80-54389-4	*IPQ53XX.ILQ.12.2 WLAN FIRMWARE SECURE BINARY USER GUIDE*
80-54391-6	*IPQ53XX AP.MI01.2 + QCN9160 SCAN RADIO DESIGN VERIFICATION TEST REPORT*
80-58161-1	*IPQ5332 and QCN9160 Setup User Guide*
80-38405-1	*QCN9160 Tri-Band 2x2 MIMO 802.11ax Data Sheet*
80-38405-4	*QCN9160 DEVICE REVISION GUIDE*
80-45481-20	*AP.MI01.X REFERENCE DESIGN HARDWARE REWORK ERRATA APPLICATION NOTE*
80-19560-2	*Wireless LAN Driver Version 12.x for Access Points: Command Reference*
80-YB478-10	*NOISE FLOOR CALIBRATION FOR 802.11AX WLAN AP CHIPSETS USER GUIDE*

DCN	Doc Title
80-50178-1	*IPQ53XX PPE SWITCH SOFTWARE DEVELOPMENT KIT USER GUIDE*
80-50178-2	*IPQ53XX PPE SWITCH SOFTWARE DEVELOPMENT KIT REFERENCE*
80-50178-3	*IPQ53XX PPE SWITCH SOFTWARE DEVELOPMENT KIT DIAGNOSTIC SHELL USER GUIDE*
80-50178-4	*IPQ53XX PPE SWITCH UCI COMMAND USER GUIDE*
80-50178-5	*IPQ53XX PPE SWITCH SOFTWARE DEVELOPMENT KIT DEVICE TREE CONFIGURATION SER GUIDE*
80-50179-1	*IPQ53XX BDF CONTENT CHECKLIST*
80-45481-10	*IPQ53X2 APPROVED VENDOR LIST REFERENCE*
80-YB714-1	*CDT DEFINITION AND MEMORY CONFIGURATION FOR WLAN AP CHIPSETS USER GUIDE*
80-19560-20	*U-BOOT FOR WLAN AP CHIPSETS USER GUIDE*
80-19560-27	*CRYPTO APIS FOR WLAN AP CHIPSETS USER GUIDE*
80-50186-1	*IPQ53XX SOC SOFTWARE USER GUIDE*
80-50186-2	*IPQ53XX QSDK USER GUIDE*
80-50186-3	*IPQ53XX SECURE BOOT ENABLEMENT USER GUIDE*
80-50186-4	*IPQ53XX SCM API REFERENCE*
80-50186-5	*IPQ53XX CLOCK PLAN REFERENCE MANUAL*
80-50186-6	*IPQ53XX APSS GPIO PIN CONTROL SOFTWARE USER GUIDE*
80-50186-20	*IPQ53XX FEATURES ON DIFFERENT DDR PROFILES APPLICATION NOTE*
80-50186-8	IPQ53XX BOOT AND COREBSP ARCHITECTURE TECHNICAL OVERVIEW
80-50186-9	IPQ53XX STORAGE TECHNICAL OVERVIEW
80-50186-10	IPQ53xx Peripherals (UART, SPI, I2C, I3C) Technical Overview
80-50186-11	IPQ53xx PCIe Technical Overview
80-50186-12	IPQ53XX USB TECHNICAL OVERVIEW
80-50186-14	IPQ53XX TRUSTZONE AND SECURITY TECHNICAL OVERVIEW
80-YA728-10	*LXC AND DOCKER CONTAINERS FOR WLAN AP CHIPSETS USER GUIDE*
80-YA728-24	*SECURE NAND STORAGE FOR WLAN AP CHIPSETS USER GUIDE*
80-YB478-6	*SAFE-BOOT PROTECTION FOR WLAN AP CHIPSETS USER GUIDE*
80-YB478-7	*PERF TOOL FOR WLAN AP CHIPSETS USER GUIDE*
80-YC907-1	*Qualcomm FLASH IMAGE LOADER (QFIL) FOR WLAN AP CHIPSETS USER GUIDE*
80-Y6950-2	*TRUSTZONE APPLICATION FOR WLAN AP CHIPSETS USER GUIDE*
80-Y8045-3	*THERMAL MANAGEMENT ALGORITHM FOR WLAN AP CHIPSETS USER GUIDE*
80-Y8856-82	*QCA_NETWORKING_2022.SPF.12.2 NSS RELEASE NOTES*
80-Y9700-29	*WATCHDOG FOR WLAN AP CHIPSETS USER GUIDE*

1.3 Frequency spectrum and regulatory domain support

This release implements the regulatory version 41. For details about regulatory domains and mapping country codes, see *Regulatory Configuration for QCA WLAN* (80-Y6630-2). Contact Qualcomm® Customer Engineering for information on the 80-Y6630-2 document revision corresponding to the Regulatory data base versions supported in this release.

Contact Qualcomm Customer Engineering for guidance on enabling 6 GHz operation in required countries for evaluation or customer release. A description of 6 GHz support on WLAN AP chipsets is available in the *WLAN Conformance Testing and Compliant Power Configuration for 802.11ax Chipsets*

Information to Share with Test Lab application note (80-YB952-3) and the *Wireless LAN Driver Version 11.0 for Access Point Programming Guide* (80-YA728-6).

Regulatory database updates that have been maintained in 2.4 GHz and 5 GHz firmware are deprecated from the 11.4 release. The firmware database is not updated from 11.1 release onwards. No firmware regulatory database is maintained for 6 GHz, and it is maintained only in BDF and the regdb.bin.

Regulatory database updates are done in BDF for all 802.11ax QLAN chipsets from 11.3 to 12.0 releases. The Regulatory database will be picked up from BDF by default from 11.3 to 11.5 releases.

The Regulatory database will be picked up from regdb.bin by default for 11ax chipsets from 12.0 release.

For 11be chipsets regulatory database information is present over a separate binary named regdb.bin. The 11be chipsets do not have regulatory database information in BDF or FW source code. The binary file regdb.bin is mandatory for the 11be chipsets.

New 6 GHz rules and regulatory domain support will change frequently during 2021-2023. This status is accurate as of the date of publication of these *Release Notes*.

2 IPQ9574.ILQ.12.2 CSU1

2.1 Supported hardware for this SP

This release supports these RDPs:

RDP	RD	Board Name	Chipset number
RDP433	AL02.1	AL02-C4	IPQ9574 + QCN9274 (with 2G, 5G, 6G); single radio 4x4 operation
RDP454	AL02	AL02-C9	IPQ9554 + QCN9274 (with 2G, 5G, 6G); dual radio 2x2 + 2x2 operation
RDP455	AL02.6	AL02-C12	IPQ9574 + QCN9274 + QCN9074 (with 2G, 5G, 6G)
RDP0458	AL03.2	AL03-C2	IPQ9550 + QCN9274 (with 2G, 5G, 6G); dual radio 2x2 + 2x2 operation
RDP0459	AL02.13	AL02-C13	IPQ9574 + QCN9274 + QCN9074 (with 2G, 5G, 6G); single radio 4x4 operation with QCN9074 scan radio
RDP461	AL02.3	AL02.C3	IPQ9574 + QCN9012 (with 2G, 5G, 6G); dual radio 2x2 + 2x2 operation + SDX65
RDP437	AL02.19	AL02.C19	IPQ9574 + QCN9274 + QCN9074 (with 2G, 5G, 6G);); dual radio 2x2 + 2x2 operation + SDX65
RDP456	AL02.8	AL02.8	IPQ9574 + QCN9274 with 2G , 5GL , 5GH , 6GL, 6GH (Pentaband)
RDP467	AL02.11	AL02.11	IPQ9574 + QCN9274 with WKK2.1 2G + WKK01.12 5GL SBS + QCN92xx 01.13 5GH SBS + WK 01.2 6G SBS (Quadband)

RDP433, RDP454, and RDP459 are based on IPQ9574. RDP458 is based on the IPQ9550. They include three QCN9274 based wireless cards, used for 2.4 GHz, 5 GHz, and 6 GHz, respectively. For more details, see the RDP setup guides.

QCN9274 chipset features

- 4x4/320 MHz 802.11be PCIe radio
- 2.4 GHz, 5 GHz, 6 GHz full band support
- Dual Lane PCIe Gen 3
- Package: 11.1 x 12 FCBGA, 0.65 mm ball pitch
- WLAN
  - Dual-synthesizer WLAN radio up to 320 MHz band width,
  - Supports 20/40 MHz in 2.4 GHz
  - Supports 20/40/80/160 MHz in 5 GHz
  - Supports 20/40/80/160/320 MHz in 6 GHz
  - Supports up to 4096 QAM with 4 spatial streams (4SS)
- Supported standards
  - IEEE802.11a/b/g/n/ac/ax/be
- Direct Switch Connect support when attached to IPQ9574/IPQ53xx based Platforms.

IPQ9574 chipset features

- CPU/platform
  - Quad ARM Cortex A73 at 2.2 GHz, 64-bit ISA v8 instruction set
  - 64 KB/32 KB I$/D$ and 1 MB L2$
  - Floating point and NEON SIMD DSP for each core
  - DDR4/DDR3L, USB3, PCIe, all serial interfaces
- One 2.4 GHz radio with four antennas (4x4 40 MHz WLAN 802.11ax)
- Networking
  - 6 Ethernet ports supported including a maximum of two 10G ports
  - USXGMII/XFI, SGMII, SGMII+, PSGMII In-kernel networking offload using ‘SFE” for TCP UDP/UDP flows
  - In-kernel networking offload using ‘SFE” for TCP UDP/UDP flows.
  - L2 hardware switch with PPE for ethernet ports

2.2 Build and load the image for IPQ9574.ILQ.12.2 CSU1

1. Download the Qualcomm Technologies, Inc. (QTI) proprietary code from the Qualcomm ChipCode Portal (see Section 2.2.1).
2. Download other components from external websites for QSDK while building the default configuration (see Section 2.2.2).
3. Generate the firmware:
   a. Reassemble the code and generate the QSDK framework (see Section 2.2.3.2).
   b. Set up and create the QSDK build (see Section 2.2.3.3 and Section 2.2.3.4).
   c. Generate a complete firmware image (see Section 2.2.4).
4. Flash the software image (see Section 2.2.5).

Users should be familiar with directory structures that contain SP images for the different subsystems before downloading the code and building the images for loading. For each SP included in an SPF, SP binary files are generated from the SI binary files of only a subset of the included SIs. In an SPF, some SIs may support multiple SPs while others may only support one SP.

2.2.1 Download packages available through the Qualcomm ChipCode Portal

QTI proprietary code is available from ChipCode. A web/GUI interface and a secure git server both allow access to this code.

- Browse available packages and obtain the download URL at: https://chipcode.qti.qualcomm.com/
- For more information on cloning the code, see: https://chipcode.qti.qualcomm.com/helpki/cloning-code-from-a-repository
- For more information on installation and configuration of the correct version of git and OpenSSL on both Windows and Linux platforms, see: https://chipcode.qti.qualcomm.com/helpki

  These versions are required to support the authentication methods used by QTI ChipCode.

2.2.2 Download packages from external websites

QSDK downloads these components while building the default profile configuration. Further customize QSDK to download more components; this table lists only the components that are necessary for at least one of the QSDK 2.0 default profiles. This list does not include the packages obtained from Qualcomm ChipCode.

Table 2-1 Packages available from external sites

Packages
accel-pptp-0.8.5.tar.bz2
acl-2.2.53.tar.gz
adb-6fe92d1a3fb17545d82d020a3c995f32e6b71f9d.tar.xz
alsa-lib-1.1.8.tar.bz2
alsa-utils-1.1.7.tar.bz2
argp-standalone-1.3.tar.gz
arptables-2015-05-20-f4ab8f63.tar.xz
attr-2.4.48.tar.gz
autoconf-2.69.tar.xz
automake-1.15.1.tar.xz
avahi-0.8.tar.gz
bash-5.1.tar.gz
bc-1.06.95.tar.bz2
binutils-2.31.1.tar.xz
binutils-2.37.tar.xz
bison-3.4.1.tar.xz
breakpad-0.1-c46151db0ffd1a8dae914e45f1212ef427f61ed3.tar.gz
bridge-utils-1.6.tar.xz
busybox-1.35.0.tar.bz2
bzip2-1.0.8.tar.gz
ca-certificates_20200601.tar.xz
c-ares-1.15.0.tar.gz
cgi-io-2022-08-10-901b0f04.tar.xz
cmake-3.15.1.tar.gz

conntrack-tools-2018-05-01-88610abe.tar.xz
coreutils-8.32.tar.xz
cryptodev-linux-1.10.tar.gz
cryptsetup-2.1.0.tar.xz
csstidy-2018-12-20-1d562014.tar.xz
curl-7.80.0.tar.xz
db-4.7.25.NC.tar.gz
dbus-1.13.18.tar.xz
dhcp-4.4.1.tar.gz
dhrystone-2.1.tar.gz
dnsmasq-2.80.tar.xz
dosfstools-4.1.tar.xz
dropbear-2019.78.tar.bz2
e2fsprogs-1.44.5.tar.xz
e2fsprogs-1.45.4.tar.xz
easycwmp-1.6.0.tar.gz
EasyRSA-3.0.4.tgz
ebtables-2018-06-27-48cff25d.tar.xz
elfutils-0.177.tar.bz2
ethtool-5.2.tar.xz
expat-2.2.9.tar.bz2
expat-2.4.9.tar.xz
file-5.38.tar.gz
findutils-4.6.0.tar.gz
firewall-2019-11-22-8174814a.tar.xz
flex-2.6.4.tar.gz
fortify-headers-1.1.tar.gz
fstools-2020-05-12-84269037.tar.xz
gcc-7.5.0.tar.xz

gdb-10.1.tar.xz
gdb-8.3.1.tar.xz
gdbm-1.11.tar.gz
gengetopt-2.23.tar.xz
gettext-0.19.8.1.tar.xz
glib-2.58.3.tar.xz
gmp-6.1.2.tar.xz
gnutls-3.7.1.tar.xz
go1.16.6.linux-amd64.tar.gz
gpsd-3.23.tar.xz
i2c-tools-4.1.tar.xz
iozone3_420.tar
iperf-2.0.13.tar.gz
iperf-3.10.1.tar.gz
iproute2-5.0.0.tar.xz
ipset-7.3.tar.bz2
iptables-1.8.3.tar.bz2
iputils-s20101006.tar.bz2
iw-5.16.tar.xz
jansson-2.12.tar.bz2
json-c-0.12.1-nodoc.tar.gz
jsonfilter-2018-02-04-c7e938d6.tar.xz
kmod-20.tar.xz
libaio-0.3.112.tar.gz
libcap-2.27.tar.xz
libcoap-4.2.0.tar.gz
libdaemon-0.14.tar.gz
libdbi-0.9.0.tar.gz
libelf-0.8.13.tar.gz

libevdev-1.6.0.tar.xz
libevent-2.1.11-stable.tar.gz
libffi-3.3.tar.gz
libgcrypt-1.9.4.tar.bz2
libgpg-error-1.36.tar.bz2
libgpiod-1.3.tar.xz
libiwinfo-2019-10-16-07315b6f.tar.xz
libmicroxml-2015-03-18-caa8d3e6887f5c70e54df555dd78e4e45cfa74cc.zip
libmnl-1.0.4.tar.bz2
libnetfilter_conntrack-2018-05-01-3ccae9f5.tar.xz
libnetfilter_cthelper-1.0.0.tar.bz2
libnetfilter_cttimeout-1.0.0.tar.bz2
libnetfilter_queue-2017-06-27-601abd1c.tar.xz
libnfnetlink-1.0.1.tar.bz2
libnftnl-1.1.4.tar.bz2
libnl-3.4.0.tar.gz
libpcap-1.9.1.tar.gz
libressl-2.9.2.tar.gz
libroxml-3.0.2.tar.gz
libtasn1-4.15.0.tar.gz
libtirpc-1.2.6.tar.bz2
libtool-2.4.tar.xz
libubox-2020-05-25-66195aee.tar.xz
libudev-fbsd-20171216-fa190fdf0b22a41b5f42e3a722f754c08ad7b337.tar.xz
libunwind-1.3.1.tar.gz
libusb-1.0.22.tar.bz2
libuv-v1.40.0.tar.gz
libwebsockets-3.1.0.tar.gz

libxml2-2.9.14.tar.xz
linux-atm-2.5.2.tar.gz
Linux-PAM-1.5.1.tar.xz
linuxptp-3.1.1.tgz
lldpd-1.0.7.tar.gz
lmbench-3.0-a9.tgz
lrzsz-0.12.20.tar.gz
lss-1.0.tar.gz
ltp-20170929.tar.gz
lua-5.1.5.tar.gz
luafilesystem-1.7.0.2.tar.gz
lucihttp-2019-07-05-a34a17d5.tar.xz
LVM2.2.03.02.tgz
lxc-2.1.1.tar.gz
lzma-4.65.tar.bz2
lzo-2.10.tar.gz
m4-1.4.19.tar.xz
make-ext4fs-2020-01-05-5c201be7.tar.xz
mbedtls-2.16.12.tar.gz
mcproxy-2017-08-24-93b5ace42268160ebbfff4c61818fb15fa2d9b99.tar.bz2
mdadm-4.1.tar.xz
memtester-4.1.3.tar.gz
minicom-2.7.1.tar.gz
miniupnpd-2.2.0.tar.gz
mklibs_0.1.35.tar.gz
mm-common-0.9.12.tar.xz
mosquitto-1.6.15.tar.gz
mpc-1.1.0.tar.gz

mpfr-4.0.2.tar.xz
mtd-utils-2.1.1.tar.bz2
mtools-4.0.23.tar.bz2
musl-1.1.24-ea9525c8bcf6170df59364c4bcd616de1acf8703.tar.xz
nat46-2020-06-26-1182f307.tar.xz
ncurses-6.1.tar.gz
netifd-2021-01-09-753c351b.tar.xz
nettle-3.7.3.tar.gz
nftables-0.9.2.tar.bz2
nghttp2-1.41.0.tar.xz
ninja-1.10.2.tar.gz
ntfs-3g_ntfsprogs-2017.3.23.tgz
odhcp6c-2021-01-09-64e1b4e7.tar.xz
odhcpd-2020-05-03-49e4949c.tar.xz
openssh-8.9p1.tar.gz
openssl-1.1.1q.tar.gz
openswan-2.6.51.3.tar.gz
openvpn-2.4.11.tar.xz
openvswitch-2.12.0.tar.gz
openwrt-keyring-2021-02-20-49283916.tar.xz
opkg-2021-01-31-c5dccea9.tar.xz
p910nd-0.97.tar.bz2
Parse-Yapp-1.21.tar.gz
patch-2.7.6.tar.xz
patchelf-0.9.tar.bz2
pciutils-3.6.2.tar.xz
pcre-8.43.tar.bz2
perl-5.28.1.tar.xz
pkg-config-0.29.2.tar.gz

pm-utils-1.4.1.tar.gz
popt-1.16.tar.gz
ppp-2.4.7.git-2019-05-25.tar.xz
procd-2020-03-07-09b9bd82.tar.xz
Python-3.7.13.tar.xz
qdl-1.0.tar.gz
quagga-1.2.4.tar.gz
quilt-0.65.tar.gz
readline-8.0.tar.gz
remserial-1.4.tar.gz
rng-tools-6.6-4ebc21d6f387bb7b4b3f6badc429e27b21c0a6ee.tar.xz
rpcd-2020-05-26-67c8a3fd.tar.xz
rpcsvc-proto-1.4.1.tar.xz
rp-pppoe-3.12.tar.gz
rstp-2011-10-11-434d24bae108dbb21461a13a4abcf014afa8b029.tar.gz
samba-4.14.12.tar.gz
scons-3.0.5.tar.gz
sed-4.7.tar.xz
shadow-4.8.1.tar.xz
six-1.13.0.tar.gz
sqlite-autoconf-3310100.tar.gz
squashfs4.2.tar.gz
squashfskit-v4.14.tar.xz
strace-5.0.tar.xz
stress-1.0.4.tar.gz
stunnel-5.55.tar.gz
sysfsutils-2.1.0.tar.gz
syslog-ng-3.38.1.tar.gz

sysstat-12.0.5.tar.xz
tar-1.32.tar.xz
tayga-0.9.2.tar.bz2
tcpdump-4.9.3.tar.gz
u-boot-2018.03.tar.bz2
u-boot-2022.01.tar.bz2
ubox-2019-06-16-4df34a4d.tar.xz
ubus-2022-02-21-b32a0e17.tar.xz
ucert-2020-05-24-00b921d8.tar.xz
uci-2019-09-01-415f9e48.tar.xz
uClibc++-0.2.5.tar.xz
uclient-2020-06-17-51e16ebf.tar.xz
uhttpd-2020-10-01-3abcc891.tar.xz
unbound-1.13.1.tar.gz
upx-3.96-amd64_linux.tar.xz
urngd-2020-01-21-c7f7b6b6.tar.xz
usb.ids.0.321
usbutils-007.tar.xz
usign-2020-05-23-f1f65026.tar.xz
ustream-ssl-2020-03-13-40b563b1.tar.xz
util-linux-2.34.tar.xz
v0.324.tar.gz
v1.0.0.tar.gz
v5.0.0-master.tar.gz
valgrind-3.18.1.tar.bz2
wireless_tools.29.tar.gz
wsdd2-2020-11-19-e0cf50d5.tar.xz
xl2tpd-1.3.15.tar.gz
xtables-addons-2.14.tar.xz

xz-5.2.4.tar.bz2
xz-5.2.5.tar.xz
zip30.tar.gz
zlib-1.2.11.tar.xz

2.2.3 Generate the Qualcomm® Internet Processor (IPQ) firmware for IPQ9574.ILQ.12.2 CSU1 release

The firmware image can be generated by using a script (see Section 2.1.3.1) or manually (see Section 2.1.3.2).

2.2.3.1 Script to generate complete firmware image

NOTE: The customer build script is applicable only if OpenWRT framework is used.

Meta generation script for 12.2 CSU1 (meta_generation_script.py) aims to simplify the release notes instructions required to be executed by the customers. Follow these steps.

```
$ mkdir $BUILD_WS
```

(Assume that BUILD_WS is the workspace directory)

```
$ cd $BUILD_WS
$ git clone <Chipcode Directory>
$ cd <Chipcode Directory>
$ git checkout r12.2.r3_00009.0
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

Where the clone required deliverable distro is in the same path where the OEM distro is cloned using git clone commands and checkout (git checkout) to the release tag on each deliverable distro.

```
$ rm -rf BOOT.BF.3* TZ.WNS.4.* BTFW.MAPLE.1.0.0 IPQ5018.ILQ.* IPQ6018.ILQ.*
 IPQ8074.ILQ.* IPQ5322.ILQ.* NSS.FW.* CNSS.PS.* WLAN.BL.* TZ.BF.* TZ.WNS.5.1
 BOOT.XF.0.3 BOOT.XF.0.3.1 TMEL.WNS.1.1
$ cp -rf */* .
$ cp -rf TZ.WNS.*/IPQ9574/trustzone_images .
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
 release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml
```

If this command does not work, use this repo init command instead:

```
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
 release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml --repo-
 url=https://git.codelinaro.org/clo/la/tools/repo.git --repo-branch=qc-stable --
 no-clone-bundle
$ repo sync -j8 --no-tags -qc
$ cd common/build
$ sed '/mksquashfs4#/d' -i meta_generation_script.py
$ sed 's/unsquashfs#g/unsquashfs4#g/g' -i meta_generation_script.py
$ sed 's/s#unsquashfs4/s#unsquashfs/g' -i meta_generation_script.py
$ python meta_generation_script.py -c [Chipcode_tag] -s [SP] -p [Profile] -b
 [32/64] -d [Distro_list] -m [MESH] --path $BUILD_WS
```

For example:

```
python meta_generation_script.py -c r12.2.r3_00009.0 -s IPQ9574.ILQ.12.2 -p P -b
 32 -d HYFI,WHC,EZMESH_SRC,EZMESH_BIN,EZMESH_ALG,IFLI_SRC -m EZMESH_FULL --
 path /local/mnt2/workspace/
```

The IPQ9574.ILQ.12.2 CSU1 release supports these configurations:

Software Product	Profile and bit config	Supported Deliverables Combo	Mesh options
IPQ9574.ILQ.12.2	P – 32,64	HYFI, WHC, WAPID, SIGMA-DUT, EZMESH_SRC, EZMESH_BIN, EZMESH_ALG, EZ_ALG_SRC, CTL_APP_SRC, CTL_APP_BIN,EZ_ALG_STA,IFLI_SRC,OEM	EZMESH_FULL, EZMESH_COLOCATED
	LM512 – 32	HYFI, WHC, EZMESH_SRC, EZMESH_BIN, EZMESH_ALG, EZ_ALG_SRC, CTL_APP_SRC, CTL_APP_BIN, OEM	EZMESH_FULL, EZMESH_COLOCATED

NOTE: Distro Name: qca-networking-2022-spf-12-2_qca_oem_ifli-bin
IFLI-BIN customers to directly use BIN on the target (using TFTP).

NOTE: When no additional deliverables are there, use -d OEM while triggering meta_generation_script.

NOTE: If no Mesh options are left, remove -m <Mesh option> while triggering meta_generation_script.

Single images are in these directories:

32-Bit	`$BUILD_WS/A/<Profile>Timestamp/<OEM distro>/common/build/bin` Where <Profile> is P, 512
64-Bit	`$BUILD_WS/A/<Profile>Timestamp/64/<OEM distro>/common/build/bin` Where <Profile> is P

2.2.3.2 Reassemble the code and generate the QSDK framework.

Users who did not use the script to generate an image from Section 2.1.3.1 must follow these steps for **manual** firmware image generation. This example assumes that all packages are obtained using the **git clone** command and placed in the top-level directory.

1. Reassemble the code and generate the QSDK framework:

```
$ git clone <ChipCode Directory>
$ cd <ChipCode Directory>
$ git checkout r12.2.r3_00009.0
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

2. Once *copying* the existing Git repository is complete, change the directories where the files are present as described in this table before running the repo command:

Use git to obtain these files from ChipCode:	Local directory path to files fetched by git from ChipCode:
qsdk-qca-wifi qsdk-qca-wlan	NHSS.QSDK.12.2\apss_proc\out\proprietary\Wifi
qsdk-ieee1905-security qsdk-whc qsdk-whcpy	NHSS.QSDK.12.2\apss_proc\out\proprietary\Hyfi
meta-tools common-tools qca-lib qca-mcs-apps qca-time-services qca-qmi-framework gpio-debug qca-diag qca-cnss-daemon athtestcmd fw-qca-stats qsdk-qca-athdiag minidump btdaemon qsdk-qca-nss qca-nss-fw-eip-al qca-rsrcmgr-bin	NHSS.QSDK.12.2\apss_proc\out\proprietary\QSDK-Base
qca-bluetopia	NHSS.QSDK.12.2\apss_proc\out\proprietary\BLUETOPIA
qca-wifi-fw-src-component-cmn-WLAN.HK.*.tgz qca-wifi-fw-QCA8074_v2.0-WLAN.HK.*.tar.bz2	WLAN.HK*\wlan_proc\src\components\QCA8074_v2.0 WLAN.HK*\wlan_proc\pkg\wlan_proc\bin\QCA8074_v2.0
qca-wifi-fw-QCN9224_v2.0-WLAN.WBE.*.tar.bz2 qca-wifi-fw-src-component-cmn-WLAN.WBE.*.tgz	WLAN.WBE.*/wlan_proc/pkg/wlan_proc/bin/QCN9224_v2.0 WLAN.WBE.*/wlan_proc/src/components/QCN9224_v2.0
qca-afc-bin	NHSS.QSDK.12.2\apss_proc\out\proprietary\RBIN-AFCAgent

3. Execute following commands to copy above deliverables to qsdk directory, and continue generating the QSDK framework:

```
$ rm -rf BOOT.BF.3* TZ.WNS.4.* BTFW.MAPLE.1.0.0 IPQ5018.ILQ.* IPQ6018.ILQ.*
  IPQ8074.ILQ.* IPQ5322.ILQ.* NSS.FW.* CNSS.PS.* WLAN.BL.* TZ.BF.* TZ.WNS.5.1
  BOOT.XF.0.3 BOOT.XF.0.3.1 TMEL.WNS.1.1
$ cp -rf */* .
$ cp -rf TZ.WNS.*/IPQ9574/trustzone_images .
$ repo init \
-u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak.git \
-b release \
-m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml
```

If this command does not work, use this repo init command:

```
$ repo init \
-u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak \
-b release \
-m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml \
--repo-url=https://git.codelinaro.org/clo/la/tools/repo.git \
--repo-branch=qc-stable --no-clone-bundle
$ repo sync -j8 --no-tags -qc
$ mkdir -p qsdk/dl
$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-ieee1905-security/* qsdk
```

```
$ cp -rf apss_proc/out/proprietary/Wifi/qsdk-qca-wifi/* qsdk
$ cp -rf apss_proc/out/proprietary/Wifi/qsdk-qca-wlan/* qsdk
$ cp -rf WLAN.HK*/wlan_proc/pkg/wlan_proc/bin/QCA8074_v1.0/qca-wifi-fw-
 QCA8074_v1.0-WLAN.HK.*.tar.bz2 qsdk/dl/
$ cp -rf WLAN.HK*/wlan_proc/src/components/QCA8074_v2.0/qca-wifi-fw-src-component-
 cmn-WLAN.HK.* qsdk/dl/
$ cp -rf WLAN.HK*/wlan_proc/pkg/wlan_proc/bin/QCA8074_v2.0/qca-wifi-fw-
 QCA8074_v2.0-WLAN.HK.*.tar.bz2 qsdk/dl/
$ tar xvf WLAN.HK*/wlan_proc/src/components/QCA8074_v2.0/qca-wifi-fw-src-component-
 cmn-WLAN.HK.*.tgz -C qsdk/dl
$ cp -rv WLAN.WBE.*/wlan_proc/pkg/wlan_proc/bin/QCN9224_v2.0/qca-wifi-fw-
 QCN9224_v2.0-WLAN.WBE.*.tar.bz2 qsdk/dl/
$ cp -rv WLAN.WBE.*/wlan_proc/src/components/QCN9224_v2.0/qca-wifi-fw-src-
 component-cmn-WLAN.WBE.*.tgz qsdk/dl/

$ cp -rf apss_proc/out/proprietary/QSDK-Base/meta-tools/ .
$ cp -rf apss_proc/out/proprietary/QSDK-Base/common-tools/* qsdk/
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-lib/* qsdk/
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qsdk-qca-nss/* qsdk/
$ cp -rf apss_proc/out/proprietary/BLUETOPIA/qca-bluetopia/* qsdk
$ sed -i 's/PKG_VERSION:=4.2.1.c1_26/PKG_VERSION:=4.2.1.4-00010/g'
 qsdk/qca/feeds/bluetopia/bluetopia/Makefile
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-mcs-apps/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-time-services/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-qmi-framework/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/gpio-debug/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-diag/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-cnss-daemon/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/athtestcmd/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/fw-qca-stats/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qsdk-qca-athdiag/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/nss-prop-user/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/btdaemon/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/minidump/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-rsrcmgr-bin/* qsdk
$ sed -i '/QCAHKSWPL_SILICONZ/c\PKG_VERSION:=WLAN.HK.2.9.r3-00031-
 QCAHKSWPL_SILICONZ-1' qsdk/qca/feeds/qca_hk/net/qca-hk/Makefile
$ cp apss_proc/out/proprietary/QSDK-Base/qca-nss-fw-eip-al/BIN-EIP*.AL.* qsdk/dl/
$ rm -rfv qsdk/qca/feeds/qca/utils/ctrl_app_dut
```

AFC changes:

```
$ cp -rf apss_proc/out/proprietary/RBIN-AFCAgent/qca-afc-bin/* qsdk
$ cd qsdk/qca/feeds/afc
$ find -maxdepth 1 ! -name qti-mfg-provision ! -name qti-encrypt ! -name . -
 exec rm -rv {} \;
```

32-bit:

```
$ cd <ChipCode Directory>
$ mkdir -pv qsdk/staging_dir/target-arm/usr/lib/
```

Premium 32-bit	`$ cd qsdk/prebuilt/ipq95xx_32/ipq_premium`
LM512 32-bit	`$ cd qsdk/prebuilt/ipq95xx_32/ipq_512`

```
$ cp -fv ./libprovision.so ../../../staging_dir/target-arm/usr/lib/
$ cd <ChipCode Directory>
$ mkdir -p qsdk/staging_dir/target-arm/pkginfo/
$ touch qsdk/staging_dir/target-arm/pkginfo/qti-mfg-provision.provides
$ echo libprovision.so > qsdk/staging_dir/target-arm/pkginfo/qti-mfg-
 provision.provides
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

64-bit:

```
$ cd <ChipCode Directory>
$ mkdir -p qsdk/staging_dir/target-aarch64/usr/lib/
```

Premium 64-bit

```
$ cd qsdk/prebuilt/ipq95xx_generic/ipq_premium/
```

```
$ cp -fv ./libprovision.so ../../../staging_dir/target-aarch64/usr/lib/
$ cd <ChipCode Directory>
$ mkdir -p qsdk/staging_dir/target-aarch64/pkginfo/
$ touch qsdk/staging_dir/target-aarch64/pkginfo/qti-mfg-provision.provides
$ echo libprovision.so > qsdk/staging_dir/target-aarch64/pkginfo/qti-mfg-
  provision.provides
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

NOTE: Prebuilt IPKs (such as qca-ezmesh-cmn, qca-ezmesh-ctrl, qca-ezmesh-agent) are in their folders:

`<ChipCode Directory>/qsdk/prebuilt/generic/`	Contains all 64-bit IPK images
`<ChipCode Directory>/qsdk/prebuilt/ipq95xx_32/`	Contains all 32-bit IPK images

	These files are fetched from ChipCode and copied to the working QSDK top-level directory:		
Premium/LM	**Qualcomm® HY-FI™ and WHC**	Hy-Fi qsdk-whc qsdk-whcpy	apss_proc\out\proprietary\Hyfi apss_proc\out\proprietary\Hyfi\qsdk-whc apss_proc\out\proprietary\Hyfi\qsdk-whcpy

4. Following additional files are fetched from ChipCode and copied to the working QSDK top-level directory:

	These files are fetched from ChipCode and copied to the working QSDK top-level directory:		
Premium/ LM512	**EZMESH-SRC**	qsdk-ezmesh-src	apss_proc\out\proprietary\RSRC-EZMESH\qsdk-ezmesh-src
Premium/ LM512	**EZMESH-BIN**	qsdk-ezmesh-bin	apss_proc\out\proprietary\RBIN-EZMESH\qsdk-ezmesh-bin
Premium/ LM512	**EZMESH-ALG**	qsdk-ezmesh-alg-bin	apss_proc\out\proprietary\RBIN-EZMESH-ALG\qsdk-ezmesh-bin
Premium/ LM512	**EZ-ALG-SRC**	qsdk-ez-alg-src	apss_proc\out\proprietary\RSRC-EZMESH-ALG\qsdk-ezmesh-alg-src
Premium	**Sigma-DUT**	sigma-dut.tar.bz2	apss_proc\out\proprietary\sigma-dut\sigma-dut.tar.bz2
Premium	**WAPid**	qsdk-wapid	apss_proc\out\proprietary\Wapid\qsdk-wapid
Premium	**IFLI-SRC**	qca	apss_proc\out\proprietary\RSRC-IFLI

5. Then run the additional code:

Premium/ LM	**HY-FI and WHC**	`$ cp -rf apss_proc/out/proprietary/Hyfi/hyfi/* qsdk`
		`$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-whc/* qsdk`
		`$ mkdir qsdk/qca/feeds/qca-son-mem-debug/qca-son-mem-debug`
		`$ mv qsdk/qca/feeds/qca-son-mem-debug/Makefile qsdk/qca/feeds/qca-son-mem-debug/Config.in qsdk/qca/feeds/qca-son-mem-debug/qca-son-mem-debug`
		`$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-whcpy/* qsdk`
		`$ sed -i "s/@PACKAGE_whc-son/@PACKAGE_whc-map/g" qsdk/qca/feeds/qca-lib/qca-wifison-ext-lib/Makefile`

Premium	**WAPid**	`git clone <wapid ChipCode Directory>`
		`$ cd <wapid ChipCode Directory>`
		`$ git checkout r12.2.r3_00009.0`
		`$ cd ..`
		`$ cp -rf <wapid ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/Wapid/qsdk-wapid/* qsdk`
		Where *wapid ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_wapid-src** for this release.

Premium	**Sigma-DUT**	$ git clone <sigma-dut ChipCode Directory>
		$ cd <sigma-dut ChipCode directory>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ tar xjvf <sigma-dut ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/sigma-dut/sigma-dut.tar.bz2 -C qsdk
		Where *sigma-dut ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_sigma-dut** for this release.

Premium	**IFLI-SRC**	$ git clone <IFLI-SRC ChipCode Directory>
		$ cd <IFLI-SRC ChipCode directory>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <IFLI-SRC ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RSRC-IFLI/* qsdk
		
		Where <***IFLI-SRC***_*ChipCode Directory*> is **qca-networking-2022-spf-12-2_qca_oem_ifli-src** for this release.

Premium/ LM512	**EZMESH-SRC**	EZMESH customers only:
		$ git clone <ezmesh ChipCode Directory>
		$ cd <ezmesh ChipCode Directory>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <ezmesh ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RSRC-EZMESH/qsdk-ezmesh-src/* qsdk
		Where *ezmesh ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ezmesh-src** for this release.
		Skip these sed commands in EZMESH-SRC if using **EZ-ALG-SRC** (that is, both **ezmesh-src** and **ezmesh-alg-src**):
		$ sed -i 's/HYD_MODULE_STRATEGY=y/HYD_MODULE_STRATEGY=n/g' qsdk/qca/src/qca-ezmesh/ezmeshConfig.defs
		$ sed -i '0,/ifeq/{/ifeq/d;}' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '0,/endif/{/endif/d;}' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '/libezmeshalg \\/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '/libezmeshagentalg \\/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '/DUMP/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile

Premium/ LM512	**EZMESH-BIN**	EZMESH customers only:
		$ git clone <ezmesh ChipCode Directory>
		$ cd <ezmesh ChipCode Directory>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <ezmesh ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RBIN-EZMESH/qsdk-ezmesh-bin/* qsdk
		Where *ezmesh ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ezmesh-bin** for this release.

Premium/ LM512	**EZMESH-ALG**	EZMESH customers only:
		$ git clone <ezmesh ChipCode Directory>
		$ cd <ezmesh ChipCode Directory>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <ezmesh ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RBIN-EZMESH-ALG/qsdk-ezmesh-alg-bin/* qsdk

		Where *ezmesh ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ezmesh-alg** for this release.

Premium/ LM512	**EZ-ALG-SRC**	EZMESH customers only:
		`$ git clone <ezmesh ChipCode Directory>`
		`$ cd <ezmesh ChipCode Directory>`
		`$ git checkout r12.2.r3_00009.0`
		`$ cd ..`
		`$ cp -rf <ezmesh ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RSRC-EZMESH-ALG/qsdk-ezmesh-alg-src/* qsdk`
		Where *ezmesh ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ez-alg-src** for this release.

Premium	**EZ-ALG-STA**	**This is applicable only for EZMESH customers that use glibc** :
		`$ git clone <ezmesh ChipCode Directory>`
		`$ cd < ezmesh ChipCode Directory>`
		`$ git checkout r12.2.r3_00009.0`
		Where *ezmesh ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ez-alg-sta** for this release.
		# **EZMESH-SRC is required to compile ezmesh along with EZ-ALG-STA.**

		32 bit

		`$ cd  NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-EZMESH-ALG-STATIC/qca-ezmesh-alg-static-bin/prebuilt/ipq95xx_32/ipq_premium`
		
		`$ tar -xvf qca-ezmesh-alg-static*.ipk`
		`$ tar -xvf data.tar.gz`
		`$ cp ./usr/lib/libezmeshalg.a <Chipcode Directory>/qsdk/staging_dir/target-arm/usr/lib/`
		`$ cd <Chipcode Directory>`

		64 bit

		`$ cd  NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-EZMESH-ALG-STATIC/qca-ezmesh-alg-static-bin/prebuilt/ipq95xx_generic/ipq_premium`
		
		`$ tar -xvf qca-ezmesh-alg-static*.ipk`
		`$ tar -xvf data.tar.gz`
		`$ cp ./usr/lib/libezmeshalg.a <Chipcode Directory>/qsdk/staging_dir/target-aarch64/usr/lib/`
		`$ cd <Chipcode Directory>`

CTL_APP_SRC	WFA customers only:
	`$ git clone <ctl_app_src ChipCode Directory>`
	`$ cd <ctl_app_src ChipCode Directory>`
	`$ git checkout r12.2.r3_00009.0`
	`$ cd ..`
	`$ cp -rf <ctl_app_src ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-SRC-CTRL-APP-DUT/qca-ctrl-app-dut/* qsdk`
	Where ctl_app_src *ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ctlapp-src** for this release.

CTL_APP_BIN	WFA customers only:
	`$ git clone <ctl_app_bin ChipCode Directory>`
	`$ cd <ctl_app_bin ChipCode Directory>`
	`$ git checkout r12.2.r3_00009.0`
	`$ cd ..`
	`$ cp -rf <ctl_app_bin ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-CTRL-APP-DUT/*  <destination directory>`
	Where ctl_app_bin *ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ctlapp-bin** for this release.

* Distro Name: qca-networking-2022-spf-12-2_qca_oem_ifli-bin
IFLI-BIN customers to directly use BIN on the target (using TFTP).

2.2.3.3 Set up the QSDK build environment (one time)

QSDK supports the **Premium** profile for Linux kernel 5.4.213 support**.** The QSDK framework is developed using Ubuntu (from version 16.04 to version 22.04) and Debian. However, QSDK framework regenerates critical tools required to compile firmware at build time. Thus, the framework is independent from the host environment. Although it is developed using the listed distributions, it is expected to work on others such as Red Hat, Mint, or Fedora.

The required tools can be installed manually or using a script which is available in CAF.

Set up the build environment manually

This command is for Ubuntu 16.04 or higher (for older/32-bit Debian/Ubuntu releases; customize it for other distributions:

```
$ sudo apt-get install gcc g++ binutils patch bzip2 flex make gettext \
 pkg-config unzip zlib1g-dev libc6-dev subversion libncurses5-dev gawk \
 sharutils curl libxml-parser-perl ocaml-nox ocaml ocaml-findlib \
 python-yaml libssl-dev libfdt-dev
$ sudo apt-get install device-tree-compiler u-boot-tools
```

For Ubuntu 18.04 build hosts additionally, install:

```
$ sudo apt-get install libssl1.0-dev
```

For Ubuntu 20.04/22.04 build hosts additionally, install:

```
$ wget http://launchpadlibrarian.net/366014597/make_4.1-9.1ubuntu1_amd64.deb
$ sudo dpkg -i make_4.1-9.1ubuntu1_amd64.deb
```

Set up the build environment using script

Use these steps to get the required tools installed in the Ubuntu host for QSDK compilation. Verified on Ubuntu 16 and Ubuntu 18. Once the repo init and sync to the release AU from CAF (Code Aurora Forum) is done successfully, the **setup_qsdk.sh** script is available in **qsdk/scripts/setup_qsdk.sh**.

The script assumes that repo and git are already installed and aborts the execution otherwise.

1. If the repo init was not already done, run these commands:
   ```
   $ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak.git \
    -b release \
    -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml \
    --repo-url=https://git.codelinaro.org/clo/la/tools/repo.git \
    --repo-branch=qc-stable --no-clone-bundle
   $ repo sync -j8 --no-tags -qc
   ```
2. The user running the script should have admin privileges. If not, the script will abort upon unsuccessful verification of sudo/admin access.
3. Run the script:
   ```
   $sudo bash -x setup_qsdk.sh
   ```

NOTE: This is a one-time run script.

2.2.3.4 Create the QSDK build

Because framework automatically downloads the open-source components needed during the build/ make process, ensure that an Internet connection is active on the build host when creating the build.

1. Install the different feeds in the build framework:
   ```
   $ cd qsdk
   $ ./scripts/feeds update -a
   ```

```
$ ./scripts/feeds install -a -f
```

2. Copy the base configuration to use for the build. Choose the **Premium** profile and use it to build with Linux 5.4.213 support.
3. Regenerate a complete configuration file and start the build:

Premium 32-bit

```
$ cp qca/configs/qsdk/ipq_premium.config .config
$ sed -i 's/TARGET_ipq807x_generic/TARGET_ipq95xx_ipq95xx_32/g' .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq95xx/g' .config
$ cp -rf prebuilt/ipq95xx_32/ipq_premium/* prebuilt/ipq95xx_32/
```

Premium 64-bit

```
$ cp qca/configs/qsdk/ipq_premium.config .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq95xx/g' .config
$ mkdir -p prebuilt/generic
$ cp -rf prebuilt/ipq95xx_generic/ipq_premium/* prebuilt/generic
```

LM512 32-bit

```
$ cp qca/configs/qsdk/ipq_512.config .config
$ sed -i 's/TARGET_ipq807x_generic/TARGET_ipq95xx_ipq95xx_32/g' .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq95xx/g' .config
$ cp -rf prebuilt/ipq95xx_32/ipq_512/* prebuilt/ipq95xx_32
```

NOTE: Ignore any error messages such as Error: recursive dependency detected!

For Hy-Fi, Wi-Fi SON customers:

- Starting with the QCA_Networking_2020.SPF.11.3 release, SON and EasyMesh features are compiled as separate build. SON build supports only SON features and EZMESH build supports only EasyMesh features.
- Starting with the QCA_Networking_2021.SPF.11.4 release, MAP package qca-hyd-map is replaced with qca-ezmesh. Enable additional configuration to enable EZMESH packages.
- Use the SON package commands in the following table for a SON features enabled build. Use the EZMESH package commands for an EasyMesh features enabled build.

  Note: These commands must be done before executing `make`.

4. Enable EZMESH configuration (customers with Qualcomm® Wi-Fi SON or EZMESH packages only):

EZMESH package (Full Mode)

```
$ echo "CONFIG_PACKAGE_whc-mesh=n" >> .config
$ echo "CONFIG_PACKAGE_hyfi-mesh=n" >> .config
$ echo "CONFIG_PACKAGE_whc-map=y" >> .config
$ echo "CONFIG_PACKAGE_hyfi-map=y" >> .config
```

Additional configuration to enable EZMESH packages:

```
$ echo "CONFIG_PACKAGE_qca-ezmesh-cmn=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-ctrl=n" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-agent=n" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-agentalg=n" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg-static=n" >> .config
```

EZMESH package (Co-located mode)

```
$ echo "CONFIG_PACKAGE_whc-mesh=n" >> .config
$ echo "CONFIG_PACKAGE_hyfi-mesh=n" >> .config
$ echo "CONFIG_PACKAGE_whc-map=y" >> .config
$ echo "CONFIG_PACKAGE_hyfi-map=y" >> .config
```

Additional configuration to enable EZMESH packages:

```
$ echo "CONFIG_PACKAGE_qca-ezmesh-cmn=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-ctrl=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-agent=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-agentalg=y" >> .config
```

```
$ echo "CONFIG_PACKAGE_qca-ezmesh=n" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg-static=n" >> .config
```

```
$ echo CONFIG_BUILD_SHORTENED_PATH=y >> .config
$ make defconfig
$ make V=s
```

This is applicable only for EZMESH customers that use glibc

EZMESH ALG STATIC

Additional configuration to enable ezmesh alg static

```
$ echo "CONFIG_EZMESH_ADD_STATIC_ALG=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg=n" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg-static=n" >> .config
```

5. For 64-bit firmware, once the premium 32-bit compilation completes, save all the U-Boot executable files OpenWRT-ipq9574*.elf from the directory **qsdk/bin/targets/ipq95xx/ipq95xx_32**. These files are required to generate all the variants of the complete 64-bit firmware image.
6. Download the required packages for the corresponding profile and create the image. Once the build is complete, these files are in the directory **qsdk/bin/targets/ipq95xx/ipq95xx_32** for 32-bit image and **qsdk/bin/targets/ipq95xx/generic** for 64-bit image.

32-bit	ELF files for U-Boot variants	OpenWRT-ipq9574*.elf
	SquashFS	OpenWRT-ipq95xx-ipq95xx_32-squashfs-root.img
	ITB	OpenWRT-ipq95xx-ipq95xx_32-ipq9574-alxx-fit-uImage.itb
64-bit	SquashFS	OpenWRT-ipq95xx-generic-squashfs-root.img
	ITB	OpenWRT-ipq95xx-generic-ipq9574-alxx-fit-uImage.itb

2.2.4 Generate a complete firmware image

IPQ95xx requires flashing multiple images for Bootup, including SBL, RPM, TZ, CDT, MIB images, Kernel, Filesystem, and so on. To simplify both image flashing and device boot, individual images are combined into a single Flattened Image Tree (FIT) image. FIT image components can be flashed into the respective partition based on user configuration. More tools required on the Ubuntu (from version 16.04 to 22.04) 64-bit machine include:

1. Install mkimage:
   ```
   $sudo apt-get install uboot-mkimage
   ```
2. Install DTC:
   ```
   $sudo apt-get install device-tree-compiler
   ```
3. Install libfdt:
   ```
   $sudo apt-get install libfdt-dev
   ```
4. Install Python 2.7.
5. Switch to the Qualcomm ChipCode directory:
   ```
   $ cd <ChipCode Directory>
   ```
   (if the current directory is qsdk, use cd .. to navigate to ChipCode directory)
6. Copy the flash config files to **common/build/ipq**
7. Find the single images in the **common/build/bin** directory. Images are generated for default image content. Fewer images indicate an issue with one of the partition sizes, or that not all required files are present.

32-bit image:

```
$ cd <ChipCode Directory>
$ mkdir -p common/build/ipq
$ mkdir -p apss_proc/out/meta-scripts
$ cp qsdk/qca/src/u-boot-2016/tools/pack.py apss_proc/out/meta-scripts/pack_hk.py
$ sed -i 's#</linux_root_path>#/</linux_root_path>#' contents.xml
$ sed -i 's#</windows_root_path>#\\</windows_root_path>#' contents.xml
$ cp qsdk/bin/targets/ipq95xx/ipq95xx_32/openwrt* common/build/ipq
$ cp -rf apss_proc/out/proprietary/QSDK-Base/meta-tools apss_proc/out/
$ cp -rf qsdk/bin/targets/ipq95xx/ipq95xx_32/dtbs/* common/build/ipq/
$ cp -rf skales/* common/build/ipq/
$ cp qsdk/staging_dir/host/bin/ubinize common/build/ipq/
$ cp qsdk/staging_dir/host/bin/unsquashfs4 common/build/ipq/
$ cp common/build/mksquashfs4 common/build/ipq/
$ cp -rf WLAN.HK*/wlan_proc/build/ms/bin/9574.wlanfw.eval/FW_IMAGES/*
 common/build/ipq/
$ cp -rf WLAN.WBE.*/wlan_proc/build/ms/bin/9224.wlanfw.eval_v2/* common/build/ipq/
$ cp -rvf WLAN.WBE.*/wlan_proc/build/ms/bin/9224.wlanfw.single_dualmac_v2/*
 common/build/ipq/
$ cp -rf TMEL.WNS*/firmware/signed/tmel-ipq95xx-firmware.elf common/build/ipq/
$ cd common/build
$ sed -i 's#unsquashfs#./unsquashfs4#g' update_common_info.py
$ export BLD_ENV_BUILD_ID=<profile-name>
$ python update_common_info.py 32
```

Where <profile-name> is either P or LM512.

64-bit image

```
$ cd <ChipCode Directory>
$ mkdir -p common/build/ipq_x64
$ mkdir -p apss_proc/out/meta-scripts
$ cp qsdk/qca/src/u-boot-2016/tools/pack.py apss_proc/out/meta-scripts/pack_hk.py
$ sed -i 's#</linux_root_path>#/</linux_root_path>#' contents.xml
$ sed -i 's#</windows_root_path>#\\</windows_root_path>#' contents.xml
$ cp qsdk/bin/targets/ipq95xx/generic/openwrt* common/build/ipq_x64
```

Copy the saved openwrt-ipq9574-u-boot*.elf from the 32-bit build to: common/build/ipq_x64

```
$ cp -rf apss_proc/out/proprietary/QSDK-Base/meta-tools apss_proc/out/
$ cp -rf qsdk/bin/targets/ipq95xx/generic/dtbs/* common/build/ipq_x64/
$ cp -rf skales/* common/build/ipq_x64/
$ cp qsdk/staging_dir/host/bin/ubinize common/build/ipq_x64/
$ cp qsdk/staging_dir/host/bin/unsquashfs4 common/build/ipq_x64/
$ cp -rf WLAN.HK*/wlan_proc/build/ms/bin/9574.wlanfw.eval/FW_IMAGES/*
 common/build/ipq_x64/
$ cp -rf WLAN.WBE.*/wlan_proc/build/ms/bin/9224.wlanfw.eval_v2/* common/build/ipq_x64/
$ cp -rvf WLAN.WBE.*/wlan_proc/build/ms/bin/9224.wlanfw.single_dualmac_v2/*
 common/build/ipq_x64/
$ cp -rf TMEL.WNS*/firmware/signed/tmel-ipq95xx-firmware.elf common/build/ipq_x64/
$ cp common/build/mksquashfs4 common/build/ipq_x64
$ cd common/build
$ sed -i 's#unsquashfs#./unsquashfs4#g' update_common_info.py
$ export BLD_ENV_BUILD_ID=<profile-name>
$ python update_common_info.py 64
```

Where <profile-name> is P.

2.2.5 Flash the complete default software image

2.2.5.1 Set up the flash environment

1. Ensure that the board console port is connected to the PC using these RS232 parameters:
   - 115200 bps
   - 8N1
2. Confirm that the PC is connected to the board using one of the Ethernet ports. The PC must have a TFTP server launched and listening on the interface to which the board is connected. The interface must have an IP address configured manually. At this stage power up the board and after a few seconds, press any key during the countdown.

2.2.5.2 Standard board configuration: load the image in flash and boot the platform

1. Copy the desired **xxxx-ipq9574-single.img** to the TFTP server root directory.
2. Check hardware jumper configuration according to reference board and default memory configuration (see the appropriate board setup guide for more information) to verify which flash memory the board is booting from.
3. Confirm the machine ID, Meta version, profile, and selected single image to match the memory type that the board boots from.

Supported Flash	Profile Support	Image Name
NAND eMMC NOR + NAND NOR + eMMC	Premium	eMMC 64 bit: emmc-ipq9574_64-single.img eMMC 32 bit:emmc-ipq9574-single.img NAND 64 bit: nand-ipq9574_64-single.img NAND 32 bit: nand-ipq9574-single.img NOR + eMMC 64 bit: norplusemmc-ipq9574_64-single.img NOR + eMMC 32 bit: norplusemmc-ipq9574-single.img NOR + NAND 64 bit: norplusnand-ipq9574_64-single.img NOR + NAND 32 bit: norplusnand-ipq9574-single.img

4. Set the IP address and server IP using the TFTP process:
   ```
   set ipaddr 192.168.1.11
   set serverip 192.168.1.xx (TFTP server address)
   ping $serverip
   set bootargs console=ttyMSM0,115200n8
   tftpboot xxxx-ipq9574-single.img
   ```
5. Flash the image:
   ```
   imgaddr=$fileaddr && source $imgaddr:script
   ```
6. Power cycle the board after step 5 has completed, as indicated by printing of the U-Boot prompt (may be hundreds of seconds depending on image size and memory technology).
7. To enable MLO support at platform level, set the following bootargs:
   ```
   setenv bootargs 'console=ttyMSM0,115200n8 cnss2.enable_mlo_support=1"
   ```

2.2.5.3 Upgrade the firmware

This release can upgrade board images without the need for a TFTP server. After using U-Boot to flash an initial image and booting the device, use the web interface for future upgrades. After using U-Boot to flash an initial image and boot the device, future upgrades can be done from either the OpenWrt web interface or serial console. Upgrade the existing flash image using either the appropriate single image file or the apps image file generated by **update_common_info.py**.

Upgrading image files via the web interface takes several minutes to complete depending on factors such as memory technology, vendor, image size, browser connectivity, and network load. Completion of the flash upgrade process is signaled by the refresh of the OpenWrt login page, or lack of further messages on the serial console.

- An invalid image remains in the flash memory if the upgrade process is interrupted, such as if system power is lost during upgrade or a key-press event is detected on the serial console port.
- If a board is configured for failsafe boot, and the U-Boot environment variable *bootargs* contains any reference to the rootfs or other partitions managed by the failsafe scheme. In this case, the sysupgrade process may only be partially successful: the board may or may not boot subsequently, and the image may only be partly updated.
- Using the **sysupgrade** command from the Linux console requires the sysupgrade image to be visible:
  - Within the Linux filesystem supplied using development host SSH or TFTPD server, OR
  - On an NFS server that is mounted to the local file system, OR
  - Transferred using a USB storage device
- The sysupgrade feature can only be used to update the image in the memory the board boots from and is used by the running kernel. If the board supports multiple memory technologies, use U-Boot to change the image in memories that the board did not boot from.

See the *IPQ95xx SoC Software User Guide* (80-19560-8) for more information on Sysupgrade support.

2.2.6 Create customized IPQ9574 + QCN90xx Wi-Fi firmware images

Flash memory layout includes a dedicated partition for the Wi-Fi firmware and BDF information. The release includes squashfs and ubifs images in the final image created by the last steps of Section 2.1.5. Use these steps to update or recreate the squashfs/ubifs Wi-Fi firmware image and the complete final image.

NOTE: These steps are not needed if the BDF and PIL files are not customized.

1. Install these tools in the workstation:
   - mtd-utils: sudo apt-get install mtd-utils
   - mksquashfs4 binary from:
     <ChipCode Directory>/IPQ9574.ILQ.12.2/common/build to /usr/bin or /usr/sbin or any location included in $PATH
2. Download the WLAN SquashFS image wifi_fw_ipq9574_qcn9000_squashfs.img from **<ChipCode Directory>/wlan_proc/build/ms/bin/FW_IMAGES/**
3. Extract the squashfs image in a temp directory.
```
mkdir -p fwtemp
cp <download path>/wifi_fw_ipq9574_qcn9000_squashfs.img ./fwtemp
cd fwtemp
unsquashfs wifi_fw_ipq9574_qcn9000_squashfs.img
```
4. Copy/modify any files for ipq9574 and qcn9000 in the folder ./squashfs-root.
5. To generate the firmware squash image for **NOR/eMMC** (this example assumes mksquashfs4 is in $PATH; use the exact installed path of mksquashfs4):
```
mksquashfs4 squashfs-root/ wifi_fw.squashfs -nopad -noappend -root-owned -comp
 xz -Xpreset 9 -Xe -Xlc 0 -Xlp 2 -Xpb 2 -Xbcj arm -b 256k -processors 1
dd if=wifi_fw.squashfs of=wifi_fw_ipq9574_qcn9000_squashfs.img bs=2k conv=sync
```

6. The UBI firmware image for NAND is no longer required because the Wi-Fi image is added to the UBI volume and is ubinized along with Kernel and rootfs
7. To create the final single image with the updated or recreated firmware image:
```
cp <ChipCode Directory>/fwtemp/wifi_fw_ipq9574_qcn9000_squashfs.img <ChipCode Directory>/wlan_proc/build/ms/bin/FW_IMAGES/
cp <ChipCode Directory>/fwtemp/wifi_fw_ipq9574_qcn9000_squashfs.img <ChipCode Directory>/wlan_proc/build/ms/bin/FW_IMAGES/
```
8. Run this command again:
```
export BLD_ENV_BUILD_ID=<profile-name>
python update_common_info.py <arch>
```
Where <*profile-name*> is P and <*arch*> is 32 or 64.

For instructions on how to flash the firmware images separately, see the *IPQ95xx SoC Software User Guide* (80-19560-8).

2.2.7 Create customized IPQ9574 Wi-Fi firmware images

Flash memory layout includes a dedicated partition for the Wi-Fi firmware and BDF information. The release includes squashfs and ubifs images in the final image. Use these steps to update or recreate the squashfs/ubifs Wi-Fi firmware image and the complete final image.

NOTE: These steps are not needed if the BDF and PIL files are not customized.

1. Install these tools in the workstation:
   - mtd-utils: $sudo apt-get install mtd-utils
   - mksquashfs4 binary from:
     <*ChipCode Directory*>/common/build to /usr/bin or /usr/sbin or any location included in $PATH
2. Download the WLAN SquashFS image wifi_fw_ipq9574_squashfs.img from:
<*ChipCode Directory*>/WLAN.HK/wlan_proc\build\ms\bin\9574.wlanfw.eval\FW_IMAGES
3. Extract the squashfs image in a temp directory.
```
mkdir -p fwtemp
cp <download path>/wifi_fw_ipq9574_squashfs.img ./fwtemp
cd fwtemp
unsquashfs wifi_fw_ipq9574_squashfs.img
```
4. To generate the firmware squash image (this example assumes mksquashfs4 is in $PATH; use the exact installed path of mksquashfs4):
```
mksquashfs4 squashfs-root wifi_fw_ipq9574_squashfs.img -nopad -noappend -root-owned -comp xz -Xpreset 9 -Xe -Xlc 0 -Xlp 2 -Xpb 2 -Xbcj arm -b 256k -processors 1
```
5. To create the final single image with the updated or recreated firmware image:
```
cp <download path>/fwtemp/wifi_fw_qcn9224_squashfs.img \
  <ChipCode Directory>/WLAN.WBE.1.0/wlan_proc/build/ms/bin/9224.wlanfw.eval/
```
6. Run this command again:
```
export BLD_ENV_BUILD_ID=<profile-name>
python update_common_info.py <arch>
```
Where <*profile-name*> is P and <*arch*> is 32 or 64.

For instructions on how to flash the firmware images separately, see the *IPQ95xx SoC Software User Guide* (80-19560-8).

2.2.8 Create customized QCN9274 Wi-Fi firmware images

Flash memory layout includes a dedicated partition for the Wi-Fi firmware and BDF information. The release includes squashfs and ubifs images in the final image. Use these steps to update or recreate the squashfs/ubifs Wi-Fi firmware image and the complete final image.

NOTE: These steps are not needed if the BDF and PIL files are not customized.

1. Install these tools in the workstation:
   - mtd-utils: $sudo apt-get install mtd-utils
   - mksquashfs4 binary from:
     ***<ChipCode Directory>*/common/build** to /usr/bin or /usr/sbin or any location included in $PATH
2. Extract the squashfs image in a temp directory.
   - Single-radio QCN9274 (ES silicon)
     ```
     mkdir -p fwtemp
     cp <download path>/wifi_fw_qcn9224_squashfs.img ./fwtemp
     cd fwtemp
     unsquashfs wifi_fw_qcn9224_squashfs.img
     ```
   - Single-radio QCN9274 (production silicon)
     ```
     mkdir -p fwtemp
     cp <download path>/wifi_fw_qcn9224_v2_squashfs.img ./fwtemp
     cd fwtemp
     unsquashfs wifi_fw_qcn9224_v2_squashfs.img
     ```
   - Dual-radio QCN9274 (ES silicon)
     ```
     mkdir -p fwtemp
     cp <download path>/wifi_fw_ipq9574_qcn9224_dualmac_squashfs ./fwtemp
     cd fwtemp
     unsquashfs wifi_fw_ipq9574_qcn9224_dualmac_squashfs.img
     ```
   - Dual-radio QCN9274 (production silicon)
     ```
     mkdir -p fwtemp
     cp <download path>/wifi_fw_ipq9574_qcn9224_v2_dualmac_squashfs ./fwtemp
     cd fwtemp
     unsquashfs wifi_fw_ipq9574_qcn9224_dualmac_v2_squashfs.img
     ```

   All extracted files will be in the directory **./squashfs-root**.
3. Copy/modify any files in the directory **./squashfs-root**.
4. To generate the firmware SquashFS image for NAND/NOR/EMMC (this example assumes mksquashfs4 is in $PATH; use the exact installed path of mksquashfs4):
   - Single-radio QCN9274 (ES silicon)
     ```
     mksquashfs4 squashfs-root wifi_fw_qcn9224_squashfs.img -noappend -root-owned -
     comp xz -Xpreset 9 -Xe -Xlc 0 -Xlp 2 -Xpb 2 -Xbcj arm -b 256k -processors 1
     ```
   - Single-radio QCN9274 (production silicon)
     ```
     mksquashfs4 squashfs-root wifi_fw_qcn9224_v2_squashfs.img -noappend -root-
     owned -comp xz -Xpreset 9 -Xe -Xlc 0 -Xlp 2 -Xpb 2 -Xbcj arm -b 256k -
     processors 1
     ```
   - Dual-radio QCN9274 (ES silicon)
     ```
     mksquashfs4 squashfs-root wifi_fw_qcn9224_dualmac_squashfs.img -noappend -root-
     owned -comp xz -Xpreset 9 -Xe -Xlc 0 -Xlp 2 -Xpb 2 -Xbcj arm -b 256k -
     processors 1
     ```
   - Dual-radio QCN9274 (production silicon)

```
mksquashfs4 squashfs-root wifi_fw_qcn9224_v2_dualmac_squashfs.img -noappend -
  root-owned -comp xz -Xpreset 9 -Xe -Xlc 0 -Xlp 2 -Xpb 2 -Xbcj arm -b 256k -
  processors 1
```

5. To create the final single image with the updated or recreated firmware image:
   - Single-radio QCN9274 (ES silicon)
     ```
     cp <download path>/fwtemp/wifi_fw_qcn9224_squashfs.img \
       <ChipCode Directory>/WLAN.WBE.1.0/wlan_proc/build/ms/bin/9224.wlanfw.eval/
     ```
   - Single-radio QCN9274 (production silicon)
     ```
     cp <download path>/fwtemp/wifi_fw_qcn9224_v2_squashfs.img \
       <ChipCode Directory>/WLAN.WBE.1.0/wlan_proc/build/ms/bin/9224.wlanfw.eval_v2/
     ```
   - Dual-radio QCN9274 (ES silicon)
     ```
     cp <download path>/fwtemp/wifi_fw_qcn9224_squashfs.img \
       <ChipCode Directory>/WLAN.WBE.1.0/wlan_proc/build/ms/bin/
       9224.wlanfw.single_dualmac/
     ```
   - Dual-radio QCN9274 (production silicon)
     ```
     cp <download path>/fwtemp/wifi_fw_qcn9224_v2_squashfs.img \
       <ChipCode Directory>/WLAN.WBE.1.0/wlan_proc/build/ms/bin/
       9224.wlanfw.single_dualmac_v2/
     ```
6. Run this command again:
   ```
   export BLD_ENV_BUILD_ID=<profile-name>
   python update_common_info.py <arch>
   ```
   Where <*profile-name*> is P and <*arch*> is 32 or 64.

For instructions on how to flash the firmware images separately, see the *IPQ95xx SoC Software User Guide* (80-19560-8).

2.3 Flash Wi-Fi firmware image only

The Wi Fi firmware/BDF information stored in the flash image can be updated without rewriting the whole flash image as shown in this section. Regardless of the memory type the system boots from, follow the same basic process of using tftpboot to load the new Wi-Fi firmware image to DDR, then identify the correct location in the flash and write the Wi-Fi firmware image to that location.

Table 2-2 Image file name by radio combination for IPQ9574.ILQ.12.2

Image File Name	Radio Combination
wifi_fw_ipq9574_squashfs.img	IPQ9574
wifi_fw_ipq9574_qcn9000_squashfs.img	IPQ95x4 + QCN90xx
wifi_fw_ipq9574_qcn9224_squashfs.img	IPQ95x4 + QCN92xx V1 (Single Radio)
wifi_fw_ipq9574_qcn9224_v2_squashfs.img	IPQ95x4 + QCN92xx V2 (Single Radio)
wifi_fw_ipq9574_qcn9224_dualmac_squashfs.img	IPQ95x4 + QCN92xx V1 (Dual Radio)
wifi_fw_ipq9574_qcn9224_v2_dualmac_squashfs.img	IPQ95x4 + QCN92xx V2 (Dual Radio)

NOR+NAND boot/NAND boot

- TFTP the image (see Table 2-2 for image file names):
  ```
  # tftpboot <Image File Name>
  ```
- Flash the partition using the flash command:
  ```
  # flash wifi_fw
  ```
  Where wifi_fw is the Wi-Fi UBI volume name. This name is used for flash command.

NOR boot/NOR+eMMC boot/eMMC boot

- TFTP the image (wifi_fw_squashsfs.img) (see Table 2-2 for image file names):
  ```
  # tftpboot <Image File Name>
  ```
- Flash the partition using the flash command:
  ```
  # flash 0:WIFIFW
  ```

NOTE: 0:WIFIFW is the Wi-Fi partition name. This name is used for flash command.

3 IPQ5018.ILQ.12.2 CSU1

3.1 Supported features

- CPU/platform
  - Dual-core Arm Cortex A53 at 1.0 GHz, 64-bit ISA v8 instruction set
  - 32 KB/32 KB I$/D$ and 256 KB L2$
  - Floating point and NEON SIMD DSP for each core
  - DDR3L, USB3, PCIe, all serial interfaces
- Wireless connectivity
  - Wi-Fi radio QCN9000 - 4x4, 5 GHz(RDP404)/6GHz(RDP415) with max bandwidth supported up to 160Mhz
  - Wi-Fi radio QCN61xx - 2x2, 5 GHz/6GHz with max bandwidth supported up to 160Mhz in RDP415, RDP432 and RDP403
  - Wi-Fi internal radio in IPQ5018
    - 2x2, 2.4 GHz with max bandwidth supported up to 40 MHz in RDP404/RDP415/RDP403/RDP432
- Networking
  - Single-core multithreaded network accelerator
  - Hardware crypto offload using CE5 crypto engine
  - SGMII, SGMII+
- Standard programming model
  - Linux 5.4.164 (32-bit and 64-bit)
  - QSDK based on OpenWrt 19.07

3.2 Supported hardware for this SP

NOTE: Existing customers of 16M to stay with 11.5 as a final update and any new designs to be moved to at least a minimum 32 Mb NOR or 128 Mb Flash.

RD	RDP	Chipset part number
MP03.6	RDP439	IPQ5018 QCN9012
MP03.1 (2.4 GHz xFEM (IPQ5018), and 5 GHz iPA (QCN9000))	RDP0404	IPQ5018 QCN9000

RD	RDP	Chipset part number
MP02.1 (2.4 GHz iPA (IPQ5018), 5 GHz iPA (QCN61xx), and 6 GHz iPA (QCN61xx))	RDP403	IPQ5010 QCN61xx
MP03.3 (2.4 GHz iPA (IPQ5018), 5 GHz xPA (QCN61xx), and 6 GHz xPA (QCN9000)	RDP415	IPQ5018 QCN61xx QCN9000
MP03.5 (2.4 GHz iPA (IPQ5018), 5 GHz xPA (QCN61xx), and 6 GHz xPA (QCN61xx)	RDP432	IPQ5018 QCN61xx QCN61xx

3.3 Build and load the image for IPQ5018.ILQ.12.2 CSU1

1. Download the Qualcomm Technologies, Inc. (QTI) proprietary code from the Qualcomm ChipCode Portal (see Section 3.3.1).
2. Download other components from external websites for QSDK while building the default configuration (see Section 2.1.2).
3. Generate the firmware:
   a. Reassemble the code and generate the QSDK framework (see Section 3.3.3.2).
   b. Set up and create the QSDK build (see Section 3.3.3.3).
   c. Generate a complete firmware image (see Section 3.3.4).
4. Flash the software image (see Section 3.3.5).

Users should be familiar with directory structures that contain SP images for the different subsystems before downloading the code and building the images for loading. For each SP included in an SPF, SP binary files are generated from the SI binary files of only a subset of the included SIs. In an SPF, some SIs may support multiple SPs while others may only support one SP.

3.3.1 Download packages available through Qualcomm ChipCode Portal

QTI proprietary code is available from ChipCode. A web/GUI interface and a secure git server both allow access to this code.

- Browse available packages and obtain the download URL at: https://chipcode.qti.qualcomm.com/
- For more information on cloning the code, see: https://chipcode.qti.qualcomm.com/helpki/cloning-code-from-a-repository
- For more information on installation and configuration of the correct version of git and OpenSSL on both Windows and Linux platforms, see: https://chipcode.qti.qualcomm.com/helpki

  These versions are required to support the authentication methods used by QTI ChipCode.

3.3.2 Download packages from external websites

For details on downloading packages from external sites, see Section 2.1.2.

3.3.3 Generate the firmware for IPQ5018.ILQ.12.2 CSU1

The firmware image can be generated by using a script (see Section 3.3.3.1) or manually (see Section 3.3.3.2).

3.3.3.1 Script to generate complete firmware image

NOTE: The customer build script is applicable **only if** OpenWRT framework is used.

Meta generation script for 12.2 CSU1 (meta_generation_script.py) aims to simplify the release notes instructions required to be executed by the customers. Follow these steps:

```
$ mkdir $BUILD_WS
```

(Assume that BUILD_WS is the workspace directory)

```
$ cd $BUILD_WS
$ git clone <Chipcode Directory>
$ cd <Chipcode Directory>
$ git checkout r12.2.r3_00009.0
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

<Clone required deliverable distro in the same path where OEM distro is cloned using Git clone commands and checkout (git checkout) to the release tag on each deliverable distro>

```
$ rm -rf IPQ6018.ILQ.12.* IPQ8074.ILQ.12.* IPQ9574.ILQ.12.* IPQ5322.ILQ.*
 TZ.WNS.5.* TZ.BF.4.0.8 BOOT.XF.* BOOT.BF.3.3.1 RPM.BF.2.4.1 TMEL.WNS.*
 WLAN.WBE.*
$ cp -rf */* .
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
 release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml
```

If this command does not work, use this repo init command instead:

```
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
 release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml --repo-
 url=https://git.codelinaro.org/clo/la/tools/repo.git --repo-branch=qc-stable --
 no-clone-bundle
$ repo sync -j8 --no-tags -qc
$ cd common/build
$ python meta_generation_script.py -c [Chipcode_tag] -s [SP] -p [Profile] -b
 [32/64] -d [Distro_list] -m [MESH] --path $BUILD_WS
```

For example, python meta_generation_script.py -c r12.2.r3_00009.0 -s IPQ5018.ILQ.12.2 -p P -b 32 -d HYFI,WHC,EZMESH_SRC,EZMESH_BIN,EZMESH_ALG -m EZMESH_FULL --path /local/mnt2/workspace/

The IPQ5018.ILQ.12.2 CSU1 release supports these configurations:

Software Product	Profile and bit config	Supported Deliverables Combo	Mesh options
IPQ5018.ILQ.12.2	P – 32,64	HYFI, WHC, WAPID, SIGMA-DUT, EZMESH_SRC, EZMESH_BIN, EZMESH_ALG, EZ_ALG_SRC, CTL_APP_SRC, CTL_APP_BIN, EZ_ALG_STA, OEM	EZMESH_FULL, EZMESH_COLOCATED

Software Product	Profile and bit config	Supported Deliverables Combo	Mesh options
	LM512 – 32	HYFI, WHC, WAPID, SIGMA-DUT, EZMESH_SRC, EZMESH_BIN, EZMESH_ALG, EZ_ALG_SRC, CTL_APP_SRC, CTL_APP_BIN, EZ_ALG_STA, OEM	EZMESH_FULL, EZMESH_COLOCATED
	LM256 – 32	HYFI, WHC, WAPID, SIGMA-DUT, EZMESH_SRC, EZMESH_BIN, EZMESH_ALG, EZ_ALG_SRC, CTL_APP_SRC, CTL_APP_BIN, OEM	EZMESH_FULL, EZMESH_COLOCATED

NOTE: When no additional deliverables are there, use -d OEM while triggering meta_generation_script.

NOTE: If no Mesh options are left, remove -m <Mesh option> while triggering meta_generation_script.

Single images are in these directories:

32-Bit	`$BUILD_WS/M/<Profile>Timestamp/<OEM distro>/common/build/bin` Where <Profile> is P, 512, 256
64-Bit	`$BUILD_WS/M/<Profile>Timestamp/64/<OEM distro>/common/build/bin` Where <Profile> is P

3.3.3.2 Reassemble the code and generate the QSDK framework

This example assumes that all packages listed in Sections 3.3.1 and 2.1.2 are obtained using the **git clone** command and placed in the top-level directory.

NOTE: 64-bit single image generation requires the U-Boot elf files generated by compiling the 32-bit build. Complete the 32-bit compilation before following the 64-bit build generation steps.

1. Re-assemble the code and generate the QSDK framework:

```
$ git clone <ChipCode Directory>
$ cd <ChipCode Directory>
$ git checkout r12.2.r3_00009.0
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

After the copy of the existing git repository is completed, change the directories where the files are present as described in this table before running the Repo command:

Use git to obtain these files from ChipCode:	Local directory path to files fetched by git from ChipCode:
qsdk-qca-wifi qsdk-qca-wlan	NHSS.QSDK.12.2\apss_proc\out\proprietary\Wifi
qsdk-ieee1905-security qsdk-whc qsdk-whcpy	NHSS.QSDK.12.2\apss_proc\out\proprietary\Hyfi

Use git to obtain these files from ChipCode:	Local directory path to files fetched by git from ChipCode:
qca-lib qca-mcs-apps qsdk-qca-nssqca-nss-userspace athtestcmd fw-qca-stats meta-tools common-tools qca-time-services qca-qmi-framework gpio-debug qca-diag qca-cnss-daemon btdaemon qca-rsrcmgr-bin	NHSS.QSDK.12.2\apss_proc\out\proprietary\QSDK-Base
qca-bluetopia	NHSS.QSDK.12.2\apss_proc\out\proprietary\BLUETOPIA
qca-wifi-fw-src-component-cmn-WLAN.HK.*.tgz	WLAN.HK.2.9\wlan_proc\src\components\QCA8074_v1.0
qca-wifi-fw-QCA8074_v1.0-WLAN.HK.*.tar.bz2	WLAN.HK.2.9\wlan_proc\pkg\wlan_proc\bin\QCA8074_v1.0
qca-wifi-fw-src-component-cmn-WLAN.BL.*.tgz qca-wifi-fw-src-component-halphy_tools-WLAN.BL.*.tgz qca-wifi-fw-QCA9888_hw_2-WLAN.BL.*.tar.bz2 qca-wifi-fw-AR900B_hw_2-WLAN.BL.*.tar.bz2 qca-wifi-fw-QCA9984_hw_1-WLAN.BL.*.tar.bz2 qca-wifi-fw-IPQ4019_hw_1-WLAN.BL.*.tar.bz2	WLAN.BL.3.19\cnss_proc\src\components WLAN.BL.3.19\cnss_proc\bin
qca-wifi-fw-AR9887_hw_1-CNSS.PS.*.tar.bz2 qca-wifi-fw-AR9888_hw_2-CNSS.PS.*.tar.bz2	CNSS.PS.3.19
BIN-NSS.FW.*.tar.bz2	NSS.FW.12.2\nss_proc\out\proprietary
qca-afc-bin	NHSS.QSDK.12.2\apss_proc\out\proprietary\RBIN-AFCAgent

Execute following commands to copy above deliverables to qsdk directory, and continue generating the QSDK framework:

```
$ rm -rf IPQ6018.ILQ.12.* IPQ8074.ILQ.12.* IPQ9574.ILQ.12.* IPQ5322.ILQ.*
  TZ.WNS.5.* TZ.BF.4.0.8 BOOT.XF.*  BOOT.BF.3.3.1 RPM.BF.2.4.1 TMEL.WNS.*
  WLAN.WBE.*
$ cp -rf */* .
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
  release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml
```

If this command does not work, use this repo init command:

```
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
  release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml --
  repo-url=https://git.codelinaro.org/clo/la/tools/repo.git --repo-branch=qc-
  stable --no-clone-bundle
$ repo sync -j8 --no-tags -qc
$ mkdir -p qsdk/dl
$ cp -rf apss_proc/out/proprietary/Wifi/qsdk-qca-wifi/* qsdk
$ cp -rf apss_proc/out/proprietary/Wifi/qsdk-qca-wlan/* qsdk
$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-ieee1905-security/* qsdk
$ cp -rf wlan_proc/src/components/QCA8074_v1.0/qca-wifi-fw-src-component-cmn-*
  qsdk/dl/
$ cp -rf wlan_proc/pkg/wlan_proc/bin/QCA8074_v1.0/qca-wifi-fw-QCA8074_v1.0*
  qsdk/dl/
```

```
$ tar xvf cnss_proc/src/components/qca-wifi-fw-src-component-cmn-WLAN.BL.*.tgz
 -C qsdk/dl
$ tar xvf cnss_proc/src/components/qca-wifi-fw-src-component-halphy_tools-
 WLAN.BL.*.tgz -C qsdk/dl
$ cp -rf cnss_proc/src/components/* qsdk/dl
$ cp -rf cnss_proc/bin/QCA9888/hw.2/* qsdk/dl
$ cp -rf cnss_proc/bin/AR900B/hw.2/* qsdk/dl
$ cp -rf cnss_proc/bin/QCA9984/hw.1/* qsdk/dl
$ cp -rf cnss_proc/bin/IPQ4019/hw.1/* qsdk/dl
$ cp -rf qca-wifi-fw-AR988* qsdk/dl
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qsdk-qca-athdiag/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-lib/* qsdk/
$ cp -rf apss_proc/out/proprietary/QSDK-Base/fw-qca-stats/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/meta-tools/ qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/common-tools/* qsdk/
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-mcs-apps/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qsdk-qca-nss/* qsdk/
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-nss-userspace/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-time-services/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-qmi-framework/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/gpio-debug/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-diag/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/athtestcmd/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/btdaemon/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-cnss-daemon/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-rsrcmgr-bin/* qsdk
$ sed -i '/QCAHKSWPL_SILICONZ/c\PKG_VERSION:=WLAN.HK.2.9.r3-00031-
 QCAHKSWPL_SILICONZ-1' qsdk/qca/feeds/qca_hk/net/qca-hk/Makefile
$ cp nss_proc/out/proprietary/* qsdk/dl
$ rm -rfv qsdk/qca/feeds/qca/utils/ctrl_app_dut
```

Premium/ LM512

```
$ cp -rf apss_proc/out/proprietary/BLUETOPIA/qca-bluetopia/* qsdk
$ sed -i '0,/PKG_VERSION:=4.2.1.c1_26/s//PKG_VERSION:=4.2.1.4-00010/'
 qsdk/qca/feeds/bluetopia/bluetopia/Makefile
```

```
$ cp -rf apss_proc/out/proprietary/RBIN-AFCAgent/qca-afc-bin/* qsdk
$ cd qsdk/qca/feeds/afc
$ find -maxdepth 1 ! -name qti-mfg-provision ! -name qti-encrypt ! -name . -
 exec rm -rv {} \;
```

32-bit:

```
$ cd <ChipCode Directory>
$ mkdir -p qsdk/staging_dir/target-arm/usr/lib/
```

Premium 32-bit	`$ cd qsdk/prebuilt/ipq50xx_32/ipq_premium`
LM512 32-bit	`$ cd qsdk/prebuilt/ipq50xx_32/ipq_512`
LM256 32-bit	`$ cd qsdk/prebuilt/ipq50xx_32/ipq_256`

```
$ cp -rf ./libprovision.so ../../../staging_dir/target-arm/usr/lib/
$ cd <ChipCode Directory>
$ mkdir -p qsdk/staging_dir/target-arm/pkginfo/
$ touch qsdk/staging_dir/target-arm/pkginfo/qti-mfg-provision.provides
$ echo libprovision.so > qsdk/staging_dir/target-arm/pkginfo/qti-mfg-
 provision.provides
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

64-bit:

```
$ cd <ChipCode Directory>
$ mkdir -p qsdk/staging_dir/target-aarch64/usr/lib/
```

Premium 64-bit

```
$ cd qsdk/prebuilt/ipq50xx_generic/ipq_premium/
```

```
$ cp -rf ./libprovision.so ../../../staging_dir/target-aarch64/usr/lib/
$ cd <ChipCode Directory>
$ mkdir -p qsdk/staging_dir/target-aarch64/pkginfo/
$ touch qsdk/staging_dir/target-aarch64/pkginfo/qti-mfg-provision.provides
$ echo libprovision.so > qsdk/staging_dir/target-aarch64/pkginfo/qti-mfg-
provision.provides
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

NOTE: Prebuilt ipks (such as Qualcomm® Bluetopia™ or qca-mcs-apps, qca-hyd-son, qca-ezmesh, qca-ezmesh-cmn, qca-ezmesh-ctrl, qca-ezmesh-agent) are in their respective folders.

<ChipCode Directory>/qsdk/prebuilt/ipq50xx_generic/ipq_<profile_name>/	Contains all 64-bit ipk images
<ChipCode Directory>/qsdk/prebuilt/ipq50xx_32/ipq_<profile_name>/	Contains all 32-bit ipk images

Where *ipq_<profile_name>* is ipq_256, ipq_512, or ipq_premium.

2. Customers with Qualcomm premium packages:

	These files are fetched from ChipCode and copied to the working QSDK top-level directory:		
Premium/ LM512/LM256	**Hy-Fi**	hyfi qsdk-whc qsdk-whcpy	apss_proc\out\proprietary\Hyfi apss_proc\out\proprietary\Hyfi\qsdk-whc apss_proc\out\proprietary\Hyfi\qsdk-whcpy
Premium/LM512/ LM256	**WAPid**	qsdk-wapid	apss_proc\out\proprietary\Wapid
Premium/LM256/ LM512	**Sigma-DUT**	sigma-dut.tar.bz2	apss_proc\out\proprietary\sigma-dut
Premium/ LM512/LM256	**EZMESH-SRC**	qsdk-ezmesh-src	apss_proc\out\proprietary\RSRC-EZMESH\qsdk-ezmesh-src
Premium/ LM512/LM256	**EZMESH-BIN**	qsdk-ezmesh-bin	apss_proc\out\proprietary\RBIN-EZMESH\qsdk-ezmesh-bin
Premium/ LM512/LM256	**EZMESH-ALG**	qsdk-ezmesh-alg-bin	apss_proc\out\proprietary\RBIN-EZMESH-ALG\qsdk-ezmesh-bin
Premium/ LM512/LM256	**EZ-ALG-SRC**	qsdk-ez-alg-src	apss_proc\out\proprietary\RSRC-EZMESH-ALG\qsdk-ezmesh-alg-src

Then run the additional code:

Premium/ LM512/LM256 — **Hy-Fi**

```
$ cp -rf apss_proc/out/proprietary/Hyfi/hyfi/* qsdk
$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-whc/* qsdk
$ mkdir qsdk/qca/feeds/qca-son-mem-debug/qca-son-mem-debug
$ mv qsdk/qca/feeds/qca-son-mem-debug/Makefile qsdk/qca/feeds/qca-son-
  mem-debug/Config.in qsdk/qca/feeds/qca-son-mem-debug/qca-son-mem-debug
$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-whcpy/* qsdk
$ sed -i "s/@PACKAGE_whc-son/@PACKAGE_whc-map/g" qsdk/qca/feeds/qca-
  lib/qca-wifison-ext-lib/Makefile
```

Premium/ LM512/LM256/ — **WAPid**

```
$ git clone <wapid ChipCode directory>
$ cd <wapid ChipCode directory>
$ git checkout r12.2.r3_00009.0
$ cd ..
```

		$ cp -rf <wapid ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/Wapid/qsdk-wapid/* qsdk Where *wapid ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_wapid-src** for this release.

Premium/ LM256/LM512	**Sigma-DUT**	$ git clone <sigma-dut ChipCode Directory>
		$ cd <sigma-dut ChipCode directory>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ tar -xjvf <sigma-dut ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/sigma-dut/sigma-dut.tar.bz2 -C qsdk
		Where *sigma-dut ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_sigma-dut** for this release.

Premium/ LM512/LM256	**EZMESH-SRC**	EZMESH customers only:
		$ git clone <ezmesh ChipCode Directory>
		$ cd <ezmesh ChipCode Directory>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <ezmesh ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RSRC-EZMESH/qsdk-ezmesh-src/* qsdk
		Where *ezmesh ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ezmesh-src** for this release.
		Skip these sed commands in EZMESH-SRC if using **EZ-ALG-SRC** (that is, both **ezmesh-src** and **ezmesh-alg-src**):
		$ sed -i 's/HYD_MODULE_STRATEGY=y/HYD_MODULE_STRATEGY=n/g' qsdk/qca/src/qca-ezmesh/ezmeshConfig.defs
		$ sed -i '0,/ifeq/{/ifeq/d;}' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '0,/endif/{/endif/d;}' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '/libezmeshalg \\/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '/libezmeshagentalg \\/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '/DUMP/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile

Premium/ LM512/LM256	**EZMESH-BIN**	EZMESH customers only:
		$ git clone <*ezmesh ChipCode Directory*>
		$ cd <*ezmesh ChipCode Directory*>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <*ezmesh ChipCode Directory*>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RBIN-EZMESH/qsdk-ezmesh-bin/* qsdk
		Where *ezmesh ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ezmesh-bin** for this release.

Premium/ LM512/LM256	**EZMESH-ALG**	EZMESH customers only:
		$ git clone <*ezmesh ChipCode Directory*>
		$ cd <*ezmesh ChipCode Directory*>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <*ezmesh ChipCode Directory*>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RBIN-EZMESH-ALG/qsdk-ezmesh-alg-bin/* qsdk
		Where *ezmesh ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ezmesh-alg** for this release.

Premium/ LM512/LM256	**EZ-ALG-SRC**	EZMESH customers only:
		$ git clone <ezmesh ChipCode Directory>
		$ cd <ezmesh ChipCode Directory>
		$ git checkout r12.2.r3_00009.0

		$ cd ..
		$ cp -rf <ezmesh ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RSRC-EZMESH-ALG/qsdk-ezmesh-alg-src/* qsdk
		Where *ezmesh ChipCode Direectory* is **qca-networking-2022-spf-12-2_qca_oem_ez-alg-src** for this release

Premium/ LM512	**EZ-ALG-STA**	EZMESH customers only:
		$ git clone <ezmesh ChipCode Directory>
		$ cd < ezmesh ChipCode Directory>
		$ git checkout r12.2.r3_00009.0
		Where *ezmesh ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ez-alg-sta** for this release.
		# EZMESH-SRC is required to compile ezmesh along with EZ-ALG-STA.
		
		32 bit
		Premium
		$ cd NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-EZMESH-ALG-STATIC/qca-ezmesh-alg-static-bin/prebuilt/ipq50xx_32/ipq_premium
		LM512
		$ cd NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-EZMESH-ALG-STATIC/qca-ezmesh-alg-static-bin/prebuilt/ipq50xx_32/ipq_512
		
		$ tar -xvf qca-ezmesh-alg-static*.ipk
		$ tar -xvf data.tar.gz
		$ cp ./usr/lib/libezmeshalg.a <Chipcode Directory>/qsdk/staging_dir/target-arm/usr/lib/
		$ cd <Chipcode Directory>
		
		64 bit
		Premium
		$ cd NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-EZMESH-ALG-STATIC/qca-ezmesh-alg-static-bin/prebuilt/ipq50xx_generic/ipq_premium
		
		$ tar -xvf qca-ezmesh-alg-static*.ipk
		$ tar -xvf data.tar.gz
		$ cp ./usr/lib/libezmeshalg.a <Chipcode Directory>/qsdk/staging_dir/target-aarch64/usr/lib/
		$ cd <Chipcode Directory>

CTL_APP_SRC	WFA customers only:
	$ git clone <ctl_app_src ChipCode Directory>
	$ cd <ctl_app_src ChipCode Directory>
	$ git checkout r12.2.r3_00009.0
	$ cd ..
	$ cp -rf <ctl_app_src ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-SRC-CTRL-APP-DUT/qca-ctrl-app-dut/* qsdk
	Where ctl_app_src *ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ctlapp-src** for this release.

CTL_APP_BIN	WFA customers only:
	$ git clone <ctl_app_bin ChipCode Directory>
	$ cd <ctl_app_bin ChipCode Directory>
	$ git checkout r12.2.r3_00009.0
	$ cd ..
	$ cp -rf <ctl_app_bin ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-CTRL-APP-DUT/* <destination directory>
	Where ctl_app_bin *ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ctlapp-bin** for this release.

3.3.3.3 Create the QSDK build

Because framework automatically downloads the open-source components needed during the build/ make process, ensure that an Internet connection is active on the build host when creating the build.

1. Install the different feeds in the build framework:
```
$ cd qsdk
$ ./scripts/feeds update -a
$ ./scripts/feeds install -a -f
```
2. Copy the base configuration to use for the build. Choose the **Premium** or **Enterprise** profile and use it to build with Linux 5.4.213 support.
3. Regenerate a complete configuration file and start the build:

Premium 32-bit
```
$ cp qca/configs/qsdk/ipq_premium.config .config
$ sed -i 's/TARGET_ipq807x_generic/TARGET_ipq50xx_ipq50xx_32/g' .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq50xx/g' .config
$ mv prebuilt/ipq50xx_32/ipq_premium/* prebuilt/ipq50xx_32
```

Premium 64-bit
```
$ cp qca/configs/qsdk/ipq_premium.config .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq50xx/g' .config
$ mkdir -p prebuilt/generic
$ cp -rf prebuilt/ipq50xx_generic/ipq_premium/* prebuilt/generic
```

LM512 32-bit
```
$ cp qca/configs/qsdk/ipq_512.config .config
$ sed -i 's/TARGET_ipq807x_generic/TARGET_ipq50xx_ipq50xx_32/g' .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq50xx/g' .config
$ mv prebuilt/ipq50xx_32/ipq_512/* prebuilt/ipq50xx_32
```

LM256 32-bit
```
$ cp qca/configs/qsdk/ipq_256.config .config
$ sed -i 's/TARGET_ipq807x_generic/TARGET_ipq50xx_ipq50xx_32/g' .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq50xx/g' .config
$ mv prebuilt/ipq50xx_32/ipq_256/* prebuilt/ipq50xx_32/
```

For Hy-Fi, WHC customers:

- Starting with the QCA_Networking_2020.SPF.11.3 release, SON and EasyMesh features are compiled as separate build. SON build supports only SON features and EZMESH build supports only EasyMesh features.
- Starting with the QCA_Networking_2021.SPF.11.4 release, MAP package qca-hyd-map is replaced with qca-ezmesh. Enable additional configuration to enable EZMESH packages.
- Use the SON package commands in this table for a SON features enabled build. Use the EZMESH package commands for an EasyMesh features enabled build.

4. Enable EZMESH configuration:

EZMESH package (Full Mode)
```
$ echo "CONFIG_PACKAGE_whc-mesh=n" >> .config
$ echo "CONFIG_PACKAGE_hyfi-mesh=n" >> .config
$ echo "CONFIG_PACKAGE_whc-map=y" >> .config
$ echo "CONFIG_PACKAGE_hyfi-map=y" >> .config
```
Additional configuration to enable EZMESH packages:
```
$ echo "CONFIG_PACKAGE_qca-ezmesh-cmn=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-ctrl=n" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-agent=n" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-agentalg=n" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg-static=n" >> .config
```

EZMESH package (Co-located mode)
```
$ echo "CONFIG_PACKAGE_whc-mesh=n" >> .config
$ echo "CONFIG_PACKAGE_hyfi-mesh=n" >> .config
```

```
$ echo "CONFIG_PACKAGE_whc-map=y" >> .config
$ echo "CONFIG_PACKAGE_hyfi-map=y" >> .config
```

Additional configuration to enable EZMESH packages:

```
$ echo "CONFIG_PACKAGE_qca-ezmesh-cmn=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-ctrl=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-agent=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-agentalg=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh=n" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg-static=n" >> .config
```

Additional configuration to enable ezmesh alg static

EZMESH ALG STATIC

```
$ echo "CONFIG_EZMESH_ADD_STATIC_ALG=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg=n" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg-static=n" >> .config
```

5. Use the commands in this table to enable qca-mad features.

qca-mad package

```
$ echo "CONFIG_PACKAGE_qca-mad=y" >> .config
```

In QCA_Networking_2021.SPF.11.5 and older releases, the qca-mad package is enabled as loadable modules in LM profiles (LM512, LM256). However, starting with the QCA_Networking_2022.SPF.12.0 CS release qca-mad package is disabled. If the qca-mad package is required, users must enable it explicitly in the build config. This package is made a loadable module for this release.

```
$ echo "CONFIG_BUILD_SHORTENED_PATH=y" >>.config
$ make defconfig
$ make V=s -j5
```

6. For 64-bit firmware, once the LM512 32-bit compilation completes, save all the U-Boot executable files **OpenWRT-ipq5018-u-boot*.elf** from the directory: **qsdk/bin/targets/ipq50xx/ipq50xx_32.** These files are required to generate all variants of the complete 64-bit firmware image.
7. Download the required packages for the corresponding profile and create the image. Once the build is complete, these files are in these directories:
   - 32 bit: qsdk/bin/targets/ipq50xx/ipq50xx_32
   - 64 bit: qsdk/bin/targets/ipq50xx/generic

32-bit

- OpenWRT-ipq5018*.elf (ELF files for U-Boot variants)
- OpenWRT-ipq50xx-ipq50xx_32-squashfs-root.img (SquashFS)
- OpenWRT-ipq50xx-ipq50xx_32-ipq5018-mpxx-fit-uImage.itb (ITB)
- OpenWRT-ipq50xx-ipq50xx_32-ubifs-root.img (IMG)

64-bit

- OpenWRT-ipq50xx-generic-squashfs-root.img (SquashFS)
- OpenWRT-ipq50xx-generic-ipq5018-mpxx-fit-uImage.itb (ITB)
- OpenWRT-ipq50xx-generic-ubifs-root.img (IMG)

3.3.4 Generate a complete firmware image

IPQ50xx requires flashing multiple images for Bootup, including SBL, RPM, TZ, CDT, MIB images, Kernel, Filesystem, and so on. To simplify both image flashing and device boot, individual images are combined into a single Flattened Image Tree (FIT) image. FIT image components can be flashed into the respective partition based on user configuration. More tools required on the Ubuntu (from version 16.04 to 22.04) 64-bit machine include:

1. Install DTC:

```
$sudo apt-get install device-tree-compiler
```

2. Install libfdt:
   ```
   $sudo apt-get install libfdt-dev
   ```
3. Install Python 2.7.

To generate the complete firmware image:

4. Switch to the Qualcomm ChipCode Portal directory:
   ```
   $ cd <ChipCode Directory>
   ```
5. Copy the flash config files to **common/build/ipq**
6. Find the single images in the **common/build/bin** directory. Images are generated for default image content. Fewer images indicate an issue with one of the partition sizes, or that not all required files are present

32-bit image

```
$ cd <ChipCode Directory>
$ mkdir -p common/build/ipq
$ mkdir -p apss_proc/out/meta-scripts
$ cp qsdk/qca/src/u-boot-2016/tools/pack.py apss_proc/out/meta-scripts/pack_hk.py
$ sed -i 's#</linux_root_path>#/</linux_root_path>#' contents.xml
$ sed -i 's#</windows_root_path>#\\</windows_root_path>#' contents.xml
$ cp  qsdk/bin/targets/ipq50xx/ipq50xx_32/openwrt* common/build/ipq
$ cp -r apss_proc/out/proprietary/QSDK-Base/meta-tools apss_proc/out/
$ cp -rf qsdk/bin/targets/ipq50xx/ipq50xx_32/dtbs/* common/build/ipq/
$ cp -rf skales/* common/build/ipq
$ cp -rf wlan_proc/build/ms/bin/5018.wlanfw.eval/* common/build/ipq/
$ cp -rf wlan_proc/build/ms/bin/5018.wlanfw2.map_spruce_eval/* common/build/ipq/
$ cp -rf wlan_proc/build/ms/bin/5018.wlanfw2.map_spr_spr_eval_cs/* common/build/ipq/
$ cp -rf btfw_proc/out/IPQ5018/bin/FW_IMAGES/bt_fw_patch_* common/build/ipq
$ cp qsdk/staging_dir/host/bin/ubinize common/build/ipq/
$ cd common/build/
$ sed "s/'''$/\n'''/g" -i update_common_info.py
$ sed -i "s/os.chdir(ipq_dir)//" update_common_info.py
$ sed '/packages/d;/"ipq5018_64"/d;/t32/d;/ret_prep_64image/d;/Required/d;/skales/d;
  /nosmmu/d;/os.system(cmd)/d;/os.chdir(ipq_dir)/d;/atfdir/d;/noac/d;/single-
  atf/d;/bl31.mbn/d;/bin_atf/d;/ret_pack_64image/d;/list_out_64_single/d;/list_out_64_a
  pps/d;/dict_64_bit_single/d;/dict_64_bit_apps/d;/list_out_debug/d' -i
  update_common_info.py
$ export BLD_ENV_BUILD_ID=<profile-name>
$ python update_common_info.py
```

Where <profile-name> is P/LM512/LM256.

64-bit image

```
$ cd <ChipCode Directory>
$ mkdir -p common/build/ipq_x64
$ mkdir -p apss_proc/out/meta-scripts
$ cp qsdk/qca/src/u-boot-2016/tools/pack.py apss_proc/out/meta-scripts/pack_hk.py
$ sed -i 's#</linux_root_path>#/</linux_root_path>#' contents.xml
$ sed -i 's#</windows_root_path>#\\</windows_root_path>#' contents.xml
$ cp qsdk/bin/targets/ipq50xx/generic/openwrt* common/build/ipq_x64
```

Copy the saved openwrt-ipq5018-u-boot*.elf from the 32-bit build to: common/build/ipq_x64:

```
$ cp -r apss_proc/out/proprietary/QSDK-Base/meta-tools apss_proc/out/
$ cp -rf qsdk/bin/targets/ipq50xx/generic/dtbs/* common/build/ipq_x64/
$ cp -rf skales/* common/build/ipq_x64/
$ cp -rf wlan_proc/build/ms/bin/5018.wlanfw.eval/* common/build/ipq_x64/
$ cp -rf wlan_proc/build/ms/bin/5018.wlanfw2.map_spruce_eval/* common/build/ipq_x64/
$ cp -rf wlan_proc/build/ms/bin/5018.wlanfw2.map_spr_spr_eval_cs/*
  common/build/ipq_x64/
```

```
$ cp -rf btfw_proc/out/IPQ5018/bin/FW_IMAGES/bt_fw_patch_* common/build/ipq_x64
$ cp qsdk/staging_dir/host/bin/ubinize common/build/ipq_x64
$ cd common/build
$ sed "s/'''$/\n'''/g" -i update_common_info.py
$ sed -i "s/os.chdir(ipq_dir)//" update_common_info.py
$ sed
  '/debug/d;/packages/d;/"ipq5018"/d;/t32/d;/ret_prep_32image/d;/Required/d;/nosmmu/d
  ;/os.system(cmd)/d;/skales/d;/os.chdir(ipq_dir)/d;/atfdir/d;/noac/d;/single-
  atf/d;/bl31.mbn/d;/bin_atf/d;/ret_pack_32image/d;/list_out_32_single/d;/list_out_32_a
  pps/d;/dict_32_bit_single/d;/dict_32_bit_apps/d' -i update_common_info.py
$ export BLD_ENV_BUILD_ID=<profile-name>
$ python update_common_info.py
```

Where <profile-name> is P.

3.3.5 Flash the complete default software image

3.3.5.1 Set up the flash environment

1. Ensure that the board console port is connected to the PC using these RS232 parameters:
   - 115200 bps
   - 8N1
2. Confirm that the PC is connected to the board using one of the Ethernet ports. The PC must have a TFTP server launched and listening on the interface to which the board is connected. The interface must have an IP address configured manually. At this stage powerup the board and after a few seconds, press any key during the countdown.

3.3.5.2 Standard board configuration: load the image in flash and boot the platform

1. Copy the intended **xxxx-ipq5018-single.img** (or 64-bit equivalent file) to the TFTP server root directory.
2. Check hardware jumper configuration according to reference board and default memory configuration (see the appropriate board setup guide for more information) to verify which flash memory the board is booting from.
3. Confirm the machine ID, Meta version, profile, and selected single image to match the memory type that the board boots from.
4. Set the IP address and server IP using the TFTP process:
   ```
   setenv ipaddr 192.168.1.11
   setenv serverip 192.168.1.xx (TFTP server address)
   ping ${serverip}
   set bootargs console=ttyMSM0,115200n8
   tftpboot xxxx-ipq5018-single.img
   ```
5. Flash the image:
   ```
   imgaddr=$fileaddr && source $imgaddr:script
   ```
6. Power cycle the board after step 5 has completed, as indicated, by printing of the U-Boot prompt. It may be hundreds of seconds depending on image size and memory technology.

7. For Arithmetic exception issue in the fw_printenv/fw_setenv commands in eMMC boot, run these commands:

```
cat /etc/fw_env.config |  awk '{$NF="";sub(/[ \t]+$/,"")}1'
 >/tmp/fw_env.config
mv /tmp/fw_env.config /etc/fw_env.config
```

NOTE: For Bluetopia, perform Steps 8 and 9.

8. Once the single image has booted, transfer these packages to the AP /tmp directory from ./**qsdk/prebuilt/ipq50xx/** or ./**qsdk/prebuilt/ ipq50xx _64/**, depending on whether the booted image is 32-bit or 64-bit.

```
qsdk/prebuilt/ipq50xx/bluetopia_4.2.1.c1_26-1_arm_cortex-a7_neon-vfpv4.ipk
qsdk/prebuilt/ipq50xx_64/bluetopia_4.2.1.c1_26-1_aarch64_cortex-a53_neon-
 vfpv4.ipk
```

9. Install these packages from the /tmp directory:

```
opkg install bluetopia_4.2.1.c1_26-1_arm_cortex-a7_neon-vfpv4.ipk
For Wi-Fi SON, perform these steps 10, 11, 12 (for all profiles).
```

10. Once the single image has booted, transfer SON packages for SON features or transfer EZMESH packages for EasyMesh features to the AP /tmp directory from ./**qsdk/prebuilt/ipq50xx/** or ./**qsdk/prebuilt/ ipq50xx _64/**, depending on whether the booted image is 32-bit or 64-bit.

SON package (32-bit)	qsdk/prebuilt/ipq50xx/qca-hyd-init_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq50xx/qca-hyd-son_ge474d2e-1_ipq.ipk
SON package (64-bit)	qsdk/prebuilt/ipq50xx_64/qca-hyd-init_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq50xx_64/qca-hyd-son_ge474d2e-1_ipq.ipk
EZMESH package (32-bit)	To support Full mode: qsdk/prebuilt/ipq50xx/qca-ezmesh-cmn_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq50xx/qca-ezmesh_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq50xx/qca-ezmesh-alg_ge474d2e-1_ipq.ipk
EZMESH package (64-bit)	To support Full mode: qsdk/prebuilt/ipq50xx_64/qca-ezmesh-cmn_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq50xx_64/qca-ezmesh_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq50xx_64/qca-ezmesh-alg_ge474d2e-1_ipq.ipk

NOTE: ge474d2e is git commit ID, which may also be changed.

11. Install the packages from the /tmp directory: Choose SON package to enable SON features or choose EZMESH packages for enabling MAP features.

SON package	opkg install qca-hyd-son_ge474d2e-1_ipq.ipk
EZMESH package	To support Full mode: opkg install qca-ezmesh-cmn_ge474d2e-1_ipq.ipk opkg install qca-ezmesh_ge474d2e-1_ipq.ipk opkg install qca-ezmesh-alg_ge474d2e-1_ipq.ipk

NOTE: ge474d2e is the git commit ID, which may also be changed.

12. The HYD daemon application binary name is changed from hyd to hyd-son for the SON package. Similarly, the WSPLCD daemon application binary name is changed from wsplcd to wsplcd-son and wsplcd-map for the SON and EZMESH packages, respectively. The symbolic link to the older binary name must be create after package installation to ensure that scripts and tools using work with the new binary name. Because application binary name has changed, a crash dump is created based on new binary name. Customers must take care of updating any scripts and tools that use or depend on the older binary name.

    Execute these commands to create a symbolic softlink to the older binary name:

SON package	`ln -s /usr/sbin/wsplcd-son /usr/sbin/wsplcd` `ln -s /usr/sbin/hyd-son /usr/sbin/hyd`
EZMESH package	`ln -s /usr/sbin/wsplcd-map /usr/sbin/wsplcd`

3.3.5.3 Upgrade the firmware

This release can upgrade board images without a TFTP server. After using U-Boot to flash an initial image and booting the device, use the web interface for future upgrades. These future upgrades can take place from either the OpenWrt web interface or serial console. Upgrade the existing flash image using either the appropriate single image file or the apps image file generated by **update_common_info.py**.

Upgrading image files via the web interface takes several minutes to complete depending on factors such as memory technology, vendor, image size, browser connectivity, and network load. Completion of the flash upgrade process is signaled by the refresh of the OpenWrt login page, or lack of further messages on the serial console.

- An invalid image remains in the flash memory if the upgrade process is interrupted, such as if system power is lost during upgrade or a keypress event is detected on the serial console port.
- If a board is configured for failsafe boot, and the U-Boot environment variable *bootargs* contains any reference to the rootfs or other partitions managed by the failsafe scheme. In this case, the sysupgrade process may only be partially successful: the board may or may not boot later, and the image may only be partly updated.
- To use the **sysupgrade** command from the Linux console, sysupgrade image must be visible:
  - Within the Linux file system supplied using development host ssh or tftpd server, OR
  - On an NFS server that is mounted to the local file system, OR
  - Transferred using a USB storage device
- The sysupgrade feature can only be used to update the image in the memory the board boots from and is used by the running kernel. If the board supports multiple memory technologies, use U-Boot to change the image in memories that the board did not boot from.

See the *IPQ50xx SoC Software User Guide* (80-16052-17) for more information on sysupgrade support.

3.3.6 Create customized IPQ50xx Wi-Fi firmware images

Flash memory layout includes a dedicated partition for the Wi-Fi firmware and BDF information. The release includes squashfs and ubifs images in the final image use these steps to update or recreate the squashfs/ubifs Wi-Fi firmware image and the complete final image.

NOTE: These steps are not needed if the BDF and PIL files are not customized.

1. Install these tools in the workstation:
   - mtd-utils: sudo apt-get install mtd-utils
   - mksquashfs4 binary from: **<*ChipCode Directory*>/common/build** to /usr/bin or /usr/sbin or any location included in $PATH
2. Download the WLAN IPQ5018 firmware BIN tar ball from the corresponding SP release distro.
3. Extract the firmware binary tarball:
```
mkdir -p <ChipCode Directory>/fwtemp
cp -rfv <ChipCode Directory>/wlan_proc/pkg/wlan_proc/bin/QCA5018_v1.0/qca-wifi-fw-
 QCA5018_v1.0-WLAN.HK.2.9-xxxx-QCAHKSWPL_SILICONZ-X.tar.bz2 fwtemp
cd fwtemp
tar -xvf qca-wifi-fw-QCA5018_v1.0-WLAN.HK.2.9-xxxx-QCAHKSWPL_SILICONZ-
 X.tar.bz2
```
4. Create a staging directory to hold the BD and PIL files.
```
mkdir staging_dir
cp -rfv qca-wifi-fw-WLAN.HK.2.9-xxxx-QCAHKSWPL_SILICONZ-X/PIL_IMAGES/*
 staging_dir
cp -rfv qca-wifi-fw-WLAN.HK.2.9-xxxx-QCAHKSWPL_SILICONZ-X/bdwlan* staging_dir
```
   (Replace the BDF file with the custom versions as needed.)
5. To generate the firmware squash image for NAND/**NOR/EMMC** (this example assumes mksquashfs4 is in $PATH; use the exact installed path of mksquashfs4):
```
mksquashfs4 staging_dir/ wifi_fw.squashfs -nopad -noappend -root-owned -comp
 xz -Xpreset 9 -Xe -Xlc 0 -Xlp 2 -Xpb 2 -Xbcj arm -b 256k -processors 1
dd if=wifi_fw.squashfs of=wifi_fw_squashfs.img bs=2k conv=sync
```
6. To create the final single image with the updated or recreated firmware image:
```
cp <ChipCode Directory>/fwtemp/Wi-Fi_fw_squashfs.img <ChipCode
 Directory>/wlan_proc/build/ms/bin/FW_IMAGES/
cp <ChipCode Directory>/fwtemp/Wi-Fi_fw_ubi.img <ChipCode
 Directory>/wlan_proc/build/ms/bin/FW_IMAGES/
```
7. Run this command again:
```
export BLD_ENV_BUILD_ID=<profile-name>
python update_common_info.py
```
   Where <*profile-name*> is LM512.

NOTE: For instructions on how to flash the firmware images separately, see the *IPQ50xx SoC Software User Guide* (80-16052-17).

3.3.7 Create customized IPQ50xx + QCN95x4 Wi-Fi firmware images

Flash memory layout includes a UBI volume for the Wi-Fi firmware and BDF information. The release includes squashfs images in the final image created by the last steps of Section 3.3.5. Use these steps to update or recreate the squashfs Wi-Fi firmware image and the complete final image.

NOTE: These steps are not needed if the BDF and PIL files are not customized.

1. Install these tools in the workstation:
   - mtd-utils:
     ```
     $sudo apt-get install mtd-utils
     ```
   - mksquashfs4 binary from:
     <ChipCode Directory>/common/build to /usr/bin or /usr/sbin or any location included in $PATH.
2. Download the WLAN SquashFS image wifi_fw_ipq5018_qcn9000_squashfs.img from **<ChipCode Directory>/wlan_proc/build/ms/bin/FW_IMAGES/**
3. Extract the squashfs image in a temp directory.
   ```
   mkdir -p fwtemp
   cp <download path>/wifi_fw_ipq5018_qcn9000_squashfs.img ./fwtemp
   cd fwtemp
   unsquashfs wifi_fw_ipq5018_qcn9000_squashfs.img
   ```
4. Copy/modify any files in the file: ./squashfs-root
5. To generate the firmware SquashFS image for NAND/NOR/EMMC (this example assumes mksquashfs4 is in $PATH; use the exact installed path of mksquashfs4):
   ```
   mksquashfs4 squashfs-root/ wifi_fw.squashfs -nopad -noappend -root-owned -comp
    xz -Xpreset 9 -Xe -Xlc 0 -Xlp 2 -Xpb 2 -Xbcj arm -b 256k -processors 1
   dd if=wifi_fw.squashfs of=wifi_fw_ipq5018_qcn9000_squashfs.img bs=2k conv=sync
   ```
6. The UBI firmware image for NAND is no longer required as the Wi-Fi image is added to the UBI volume and will be ubinized along with Kernel and rootfs.

For instructions on how to the generate ubinized image and flash the firmware images separately, see the *IPQ50xx SoC Software User Guide* (80-16052-17)

3.4 Flash Wi-Fi firmware image only

The Wi-Fi firmware/BDF information stored in the flash image can be updated without rewriting the whole flash image as shown in this section. Regardless of the memory type the system boots from, follow the same basic process of using tftpboot to load the new Wi-Fi firmware image to DDR, then identify the correct location in the flash and write the Wi-Fi firmware image to that location.

Table 3-1 Image file name by radio combination for IPQ9574.ILQ.12.2

Image File Name	Radio Combination
wifi_fw_ipq5018_qcn9000_squashfs.img	IPQ5018 Wi-Fi (internal 2 GHz) and QCN90xx integrated firmware image
wifi_fw_squashfs.img	IPQ5018 (internal 2 GHz) Wi-Fi firmware only image

NOR+NAND boot/NAND boot

The Wi-Fi firmware image is made as UBI volume, along with kernel, rootfs, and BT firmware to save flash space.

- TFTP the image (**wifi_fw_ubi.img**):
  ```
  # tftpboot wifi_fw_ipq5018_qcn9000_squashfs.img
  ```
- Flash the partition using the flash command:
  ```
  # flash wifi_fw
  ```

Where wifi_fw is the Wi-Fi UBI volume name. This name is used for flash command.

NOR boot/NOR+eMMC boot/eMMC boot

- TFTP the image (wifi_fw_squashsfs.img):
  ```
  # tftpboot wifi_fw_ipq5018_qcn9000_squashfs.img
  ```
- Flash the partition using the flash command:
  ```
  # flash 0:WIFIFW
  ```

NOTE: 0:WIFIFW is the Wi-Fi partition name. This name is used for flash command.

3.5 Generate secure boot image

For more information on how to generate secureboot images, see the *IPQ50xx Secure Boot Enablement User Guide* (80-16052-18).

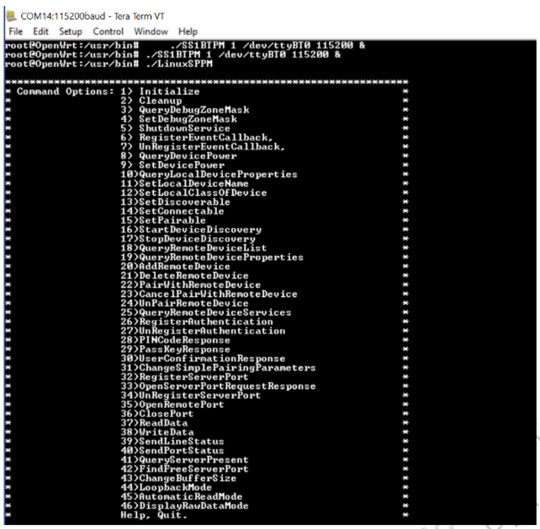
3.6 Test the serial port profile (SPP) over generic access profile (GAP) (BR/EDR) profile with sample applications with onboard Bluetooth on AP.MP03.1

This release contains a preloaded Bluetooth sample application. To test serial port profile (SPP) profile containing sample applications:

1. Launch the Bluetooth sample application from the AP after the 512M profile is built. It is in:
   32-bit: **apss_proc/out/proprietary/BLUETOPIA/qca-bluetopia/prebuilt/ipq50xx**
   64-bit: **apss_proc/out/proprietary/BLUETOPIA/qca-bluetopia/prebuilt/ipq50xx_64**

2. Launch the Bluetooth stack to test the SPP profile with these LinuxSPPM sample applications:
   - The LinuxSPPM application is intended to demonstrate the usage of the Qualcomm Bluetopia Serial Port Profile API and relevant Bluetopia Core APIs. The application supports issuing all the basic commands used by the Serial Port Profile.
   - Launching the BT sample APP, which initializes the BT controller

```
root@openwrt:cd /usr/bin
root@openwrt: ./SS1BTPM 1 /dev/ttyBT0 115200 &
root@openwrt: ./LinuxSPPM
```



3. Initialize the application with the inter-process connection to the SS1BTPM server application. The parameter determines whether the application will display events.

```
SPPM > Initialize 1
BTPM_Initialize()
Success: 0.
DEVM_RegisterEventCall back()  Success: 5.
```

4. Set the Bluetooth radio power state. The parameter defines the state used to the set the power.

```
SPPM>SetDevicePower 1
DEVM_PowerOnDevice() Success: 0.
```

5. Set the device discoverability. The first parameter is the flag to set discoverability. The second parameter is the duration in seconds for the device to remain discoverable.

```
SPPM>SetDiscoverable 1 0
Attempting to set Discoverability Mode: General.

Local Device Properties Changed.
Disc. Mode:   TRUE , 0x00000000
SPPM>DEVM_UpdateLocalDeviceProperties()
Success: 0.
```

6. Set the device connectability. The first parameter is the flag to set connectability. The second parameter is the duration in seconds for the device to remain connectable.

```
SPPM>SetConnectable 1 0
Attempting to set Connectability Mode: Connectable.
DEVM_UpdateLocalDeviceProperties() Success: 0.
SPPM> Local Device Properties Changed
```

7. Set the device pairability. The first parameter is the flag to set pairability.
   The second parameter is the duration in seconds for the device to remain pairable.
```
SPPM>SetPairable 1 0
Attempting to set Pairability Mode: Pairable.
DEVM_UpdateLocalDeviceProperties() Success: 0.
```
8. Register the app to receive authentication notifications. This command takes no parameters
```
SPPM>RegisterAuthentication
DEVM_RegisterAuthentication() Success: 4.
```
9. Register a SPP server port. The first parameter is RFCOMM port handle to register.
   The second parameter is a set of flags to enable authorization, authentication, or encryption.
```
SPPM>RegisterServerPort 5
SPPM_RegisterServerPort(5) Success. Port Handle: 2.
SPPM_RegisterServerPortServiceRecord Success. Record Handle: 65544
 (0x00010008).
```
   The port handle will be used by the client
10. Open an SPP port on a specified remote device. The first parameter is the Bluetooth address of the remote device. The second parameter is the RFCOMM port to use. The third parameter is a flag to use authentication or encryption.
```
SPPM>OpenRemotePort 5CF37082FAE9 5
Remote Device Found.
BD_ADDR:      5CF37082FAE9
COD:           0x000000
Device Name:
Device Flags:  0x80000000
RSSI:          0
Friendly Name:
App. Info:   : 00000000
Paired State : FALSE
Connect State: FALSE
Encrypt State: FALSE
Sniff State  : FALSE
Serv. Known  : FALSE
SPPM>SPPM_OpenRemotePort(1) Success. Port Handle: 1.
SPPM>
Remote Device Properties Changed.
BD_ADDR:      5CF37082FAE9
Device Flags:  0x80400048
Connect State: TRUE
SPPM>
Remote Device Properties Changed.
BD_ADDR:      5CF37082FAE9
Device Name:   SS1BTPM Device
Device Flags:  0x80400049
```
11. Issue a response to a secure simple pairing request. The parameter sets whether the pairing is accepted.
```
SPPM>UserConfirmationResponse 1
DEVM_AuthenticationResponse(), User Confirmation Response Success.
SPPM>
Remote Device Properties Changed.
BD_ADDR:      5CF37082FAE9
Device Flags:  0x804000CD
Paired State : TRUE
SPPM>
Remote Device Properties Changed.
```

```
BD_ADDR:      5CF37082FAE9
Device Flags:  0x804000DD
Encrypt State: TRUE
SPPM>
Remote Port Open Status.
Port Handle: 4
Status:      Remote Port Open Successful
SPPM>
Port Status Changed.
Port Handle: 4
Port Status:
Mask:         0x00000003
BreakSignal:  0
BreakTimeout: 0
```

12. Write data to a remote SPP device. The first parameter is the port handle to write. The second parameter is the data to send. The third parameter is a timeout to try to write the data.

```
SPPM>WriteData 4 100
Success, sending data.
SPPM>Transmit thread started.
Send complete.
Transmit thread stopped.
```

13. The following command reads data that has been sent by the remote SPP device. The first parameter is the port handle to read. The second parameter is the amount of data to read. The third parameter is a timeout to try to read the data.

```
SPPM>ReadData 4 100
SPPM_ReadData(100) Success: 100 bytes read.
Data:0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D
 0x0E 0x0F
0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1A 0x1B 0x1C 0x1D 0x1E
 0x1F
0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x2A 0x2B 0x2C 0x2D 0x2E
 0x2F
0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3A 0x3B 0x3C 0x3D 0x3E
 0x3F
0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4A 0x4B 0x4C 0x4D 0x4E
 0x4F
0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57 0x58 0x59 0x5A 0x5B 0x5C 0x5D 0x5E
 0x5F 0x60 0x61 0x62 0x63
```

Device node name

Board	Kernel version	Device node name
IPQ5018	5.4	/dev/ttyBT0

NOTE: For more information regarding Bluetopia installation and build instructions for IPQ50xx platforms, see *Bluetopia installation and operation for IPQ50xx Bluetooth Application Note* (80-16052-31).

4 IPQ8074.ILQ.12.2 CSU1

4.1 Supported features

- CPU/platform
  - Quad ARM Cortex A53 at 2.2 GHz, 64-bit ISA v8 instruction set, 20 K DMIPS
  - 32 KB/32 KB I$/D$ and 512 KB L2$
  - Floating point and NEON SIMD DSP for each core
  - DDR4/DDR3L, USB3, PCIe, all serial interfaces
- Wireless connectivity
  - For IPQ807x/IPQ817x: 5 GHz antenna configuration
    - 8 x 8/8S-80 MHz or 4x4/4S-160 MHz
    - 2x2/2S-160 MHz
    - 4 x 4/4S-80 MHz
    - 4 x 4/4S-80 MHz + 4 x 4/4S-80 MHz (SBS)
    - 4 x 4/4S-80 MHz + 2 x 2/2S-80 MHz (SBS)
  - 2.4 GHz antenna configuration
    - 4 x 4/4S-40 MHz
- Networking
  - Dual-core multi-threaded network accelerator (NPU)
  - Advanced classification, policing, queuing at wire speed (PPE)
  - USXGMII/XFI, SGMII, SGMII+, PSGMII
  - IPv4/IPv6 passthrough feature
- Standard programming model
  - Linux 5.4.213 (32-bit and 64-bit)
  - QSDK based on OpenWrt 19.07

Platform features

- DC 12 V, 5-A power adapter
- Two x16 DDR3 devices, 1866MT/s 32-bit interface, Total 1 GB (512 MB each)
- SPI NOR
- 8-bit parallel NAND flash
- Two PCIe interfaces
- JTAG
- UART

- One 10G MAC and four 1G MAC
- 1 SFP port

IPQ807x + QCN90xx features

- 4x4/160 MHz 11ax PCIe Radio
- 2.4 GHz, 5 GHz, 6 GHz full band support
- Dual Lane PCIe Gen 3
- Package: 11.1 x 12 FCBGA, 0.65 mm ball pitch
- WLAN
  - Dual-synthesizer WLAN radio up to 160 MHz band width, support aSA, ADFS
  - Supports 20/40 MHz in 2.4 GHz
  - Supports 20/40/80/160 MHz in 5 GHz
  - Supports 20/40/80/160 MHz in 6 GHz
  - Supports up to 1024 QAM (4 Spatial stream (SS)) and 4096 QAM (2 SS)
- Supported standards
  - IEEE802.11a/b/g/n/ac/ax
  - IEEE 802.11d/e/h/i/j/k/r/u/v/w

4.2 Restrictions on software while using it for testing

DL OFDMA Test Tones for FCC Conformance Testing	WLAN Conformance Testing And Compliant Power Configuration for 802.11ax Chipsets Information to Share With Test Lab (80-YB952-3) have been updated with FCC conformance test guidance for DL OFDM transmissions. Test tone XML files to support the FCC testing using QSPR and QRCT are available for the WLAN AP software release at these paths: ▪ In the FW package, the file is available at "/bdfUtil". ▪ In the WLAN source, the file is available at: wlan_proc\wlan\phyrf_svc\tools\bdfExcel\qca8074\CTL_installer_python_scripts\FCC_interim_Guidance_Test_Tones.zip
QCN90xx	▪ Low memory profiles are supported only for 32-bit. Premium/Enterprise profiles supported on both 32-bit & 64-bit.

4.3 Supported hardware for this SP

RDP	RD	Chipset part number
RDP258	AP.V2HK01.1	IPQ8078A QCN5124 QCN5154 PMP8074 QCA8075 Third-party chipset AQR107 (10 Gbps)
RDP385	AP.V2HK01.2	IPQ8074A QCN5024 QCN5054 PMP8074 QCA8075 Third-party chipset AQR107 (10 Gbps)
RDP386	AP.V2HK09.1	IPQ8076A QCN5124 QCN5154 PMP8074 QCA8081
RDP389	AP.V2AC01.1	IPQ8070A QCN5124 QCN5154 PMP8074 QCA8081
RDP390	AP.V2AC02.1	IPQ8071A QCN5124 QCN5154 PMP8074 QCA8081
RDP392	AP.V2OAK02.1	IPQ8173 QCN5024 QCA5054 QCA8075 PMP8074 QCA4024
RDP393	AP.V2OAK03.1	IPQ8174 QCN5024 QCN5054 PMP8074 QCA8075
RDP400	AP.V2AC03.1	IPQ8070A QCN5124 QCN5154 PMP8074 QCA8075
RDP401	AP.V2AC04.1	IPQ8071A QCN5124 QCN5154 PMP8074 QCA8075

4.4 Build and load the image for IPQ8074.ILQ.12.2 CSU1

1. Download the Qualcomm Technologies, Inc. (QTI) proprietary code from Qualcomm ChipCode (see Section 4.4.1).
2. Download other components from external websites for QSDK while building the default configuration (see Section 4.4.2).
3. Generate the firmware:
   a. Re-assemble the code (see Section 2.1.3.2).
   b. Create the QSDK build (see Section 4.4.4).
   c. Generate a complete firmware image (see Section 4.4.3)
4. Install the image in the device flash memory and boot using the image from the flash (see Section 4.4.5).

Users must be familiar with directory structures that contain SP images for the different subsystems before downloading the code and building the images for loading. For each SP included in an SPF, SP binary files are generated from the SI binary files of only a subset of the included SIs. In an SPF, some SIs may support multiple SPs while others may only support one SP.

4.4.1 Download packages available through Qualcomm ChipCode

QTI proprietary code is available from Qualcomm ChipCode. A web/GUI interface and a secure git server both allow access to this code.

- Browse available packages and obtain the download URL at: https://chipcode.qti.qualcomm.com/
- For more information on cloning the code, see: https://chipcode.qti.qualcomm.com/helpki/cloning-code-from-a-repository
- For more information on installation and configuration of the correct version of git and OpenSSL on both Windows and Linux platforms, see: https://chipcode.qti.qualcomm.com/helpki

  These versions are required to support the authentication methods used by Qualcomm ChipCode.

4.4.2 Download packages from external websites

For details on downloading packages from external sites, see Section 2.2.2.

4.4.3 Generate the firmware for IPQ8074.ILQ.12.2 CSU1

The firmware image can be generated by using a script (see Section 4.4.3.1) or manually (see Section 4.4.3.2).

4.4.3.1 Script to generate complete firmware image

NOTE: The customer build script is applicable **only if** OpenWRT framework is used.

Meta generation script for 12.2 CSU1 (meta_generation_script.py) aims to simplify the release notes instructions required to be executed by the customers. Follow these steps:

```
$ mkdir $BUILD_WS
```

(Assume that BUILD_WS is the workspace directory)

```
$ cd $BUILD_WS
$ git clone <Chipcode Directory>
$ cd <Chipcode Directory>
$ git checkout r12.2.r3_00009.0
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

Where the clone required deliverable distro is in the same path where the OEM distro is cloned using git clone commands and checkout (git checkout) to the release tag on each deliverable distro.

```
$ rm -rf IPQ6018.ILQ.12.* IPQ5018.ILQ.12.* IPQ9574.ILQ.12.* IPQ5322.ILQ.* TZ.WNS.*
  BOOT.XF.* BOOT.BF.3.3.1.1 BTFW.MAPLE.* TMEL.WNS.* WLAN.WBE*
$ cp -rf */* .
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b release
  -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml
```

If this command does not work, use this repo init command instead:

```
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b release
  -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml --repo-
  url=https://git.codelinaro.org/clo/la/tools/repo.git --repo-branch=qc-stable --no-
  clone-bundle
$ repo sync -j8 --no-tags -qc
$ cd common/build
$ python meta_generation_script.py -c [Chipcode_tag] -s [SP] -p [Profile] -b [32/64] -
  d [Distro_list] -m [MESH] --path $BUILD_WS
```

For example:

```
python meta_generation_script.py -c r12.2.r3_00009.0 -s IPQ8074.ILQ.12.2 -p P -b
  32 -d HYFI,WHC,EZMESH_SRC,EZMESH_BIN,EZMESH_ALG -m EZMESH_FULL --
  path /local/mnt2/workspace/
```

The IPQ8074.ILQ.12.2 CSU1 release supports these configurations:

Software Product	Profile and bit config	Supported Deliverables Combo	Mesh options
IPQ8074.ILQ.12.2	P – 32,64	HYFI, WHC, WAPID, SIGMA-DUT, EZMESH_SRC, EZMESH_BIN, EZMESH_ALG, EZ_ALG_SRC, CTL_APP_SRC, CTL_APP_BIN, EZ_ALG_STA, OEM	EZMESH_FULL, EZMESH_COLOCATED
	LM512 – 32	HYFI, WHC, EZMESH_SRC, EZMESH_BIN, EZMESH_ALG, EZ_ALG_SRC, CTL_APP_SRC, CTL_APP_BIN, EZ_ALG_STA, OEM	EZMESH_FULL, EZMESH_COLOCATED
	LM256 – 32	HYFI, WHC, EZMESH_SRC, EZMESH_BIN, EZMESH_ALG, EZ_ALG_SRC, CTL_APP_SRC, CTL_APP_BIN,EZ_ALG_STA,OEM	EZMESH_FULL, EZMESH_COLOCATED
	E – 32,64	SIGMA-DUT, CTL_APP_SRC, CTL_APP_BIN, OEM	NA
	LM512E– 32	SIGMA-DUT, CTL_APP_SRC, CTL_APP_BIN, OEM	NA

NOTE: When no additional deliverables are there, use -d OEM while triggering meta_generation_script.

NOTE: If no Mesh options are left, remove -m <Mesh option> while triggering meta_generation_script.

Single images are in these directories:

32-Bit	`$BUILD_WS/H/<Profile>Timestamp/<OEM distro>/common/build/bin` Where <Profile> is P, E, 512, 256, 512E
64-Bit	`$BUILD_WS/H/<Profile>Timestamp/64/<OEM distro>/common/build/bin` Where <Profile> is P, E

4.4.3.2 Manually generate complete firmware image

This example assumes that all packages listed in Sections 4.4.1 and 4.4.2 are obtained using the **git clone** command and placed in the top-level directory.

NOTE: 64-bit single image generation requires one of the self-generated by compiling the 32-bit build. Complete the 32-bit compilation before following the 64-bit build generation steps.

1. Enter these commands to re-assemble the code and generate the QSDK framework:

```
$ git clone <Chipcode Directory>
$ cd <Chipcode Directory>
$ git checkout r12.2.r3_00009.0
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

Use git to obtain these files from ChipCode:	Local directory path to files fetched by git from ChipCode:
qsdk-qca-wifi qsdk-qca-wlan	NHSS.QSDK.12.2\apss_proc\out\proprietary\Wifi
qsdk-ieee1905-security qsdk-whc qsdk-whcpy	NHSS.QSDK.12.2\apss_proc\out\proprietary\Hyfi

Use git to obtain these files from ChipCode:	Local directory path to files fetched by git from ChipCode:
meta-tools common-tools qsdk-qca-nss qca-lib qca-mcs-apps qca-nss-userspace qca-time-services qca-qmi-framework gpio-debug qca-diag qca-cnss-daemon athtestcmd fw-qca-stats minidump qca-nss-fw-eip-hk qca-rsrcmgr-bin	NHSS.QSDK.12.2\apss_proc\out\proprietary\QSDK-Base
qca-bluetopia	NHSS.QSDK.12.2\apss_proc\out\proprietary\BLUETOPIA
qca-wifi-fw-src-component-cmn-WLAN.HK.*.tgz qca-wifi-fw-QCA8074_v1.0-WLAN.HK.*.tar.bz2 qca-wifi-fw-QCA8074_v2.0-WLAN.HK.*.tar.bz2	WLAN.HK.2.9\wlan_proc\src\components\QCA8074_v2.0 WLAN.HK.2.9\wlan_proc\pkg\wlan_proc\bin\QCA8074_v2.0
qca-wifi-fw-src-component-cmn-WLAN.BL.*.tgz qca-wifi-fw-src-component-halphy_tools-WLAN.BL.*.tgz qca-wifi-fw-QCA9888_hw_2-WLAN.BL.*.tar.bz2 qca-wifi-fw-AR900B_hw_2-WLAN.BL.*.tar.bz2 qca-wifi-fw-QCA9984_hw_1-WLAN.BL.*.tar.bz2 qca-wifi-fw-IPQ4019_hw_1-WLAN.BL.*.tar.bz2	WLAN.BL.3.19\cnss_proc\src\components WLAN.BL.3.19\cnss_proc\bin
qca-wifi-fw-AR9887_hw_1-CNSS.PS.*.tar.bz2 qca-wifi-fw-AR9888_hw_2-CNSS.PS.*.tar.bz2	CNSS.PS.3.19
BIN-NSS.FW.*.tar.bz2	NSS.FW.12.2\nss_proc\out\proprietary
qca-afc-bin	NHSS.QSDK.12.2\apss_proc\out\proprietary\RBIN-AFCAgent

Execute following commands to copy above deliverables to qsdk directory, and continue generating the QSDK framework

```
$ rm -rf IPQ6018.ILQ.12.* IPQ5018.ILQ.12.* IPQ9574.ILQ.12.* IPQ5322.ILQ.* TZ.WNS.*
 BOOT.XF.* BOOT.BF.3.3.1.1 BTFW.MAPLE.* TMEL.WNS.* WLAN.WBE*
$ cp -rf */* .
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
 release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml
```

If this command does not work, use this repo init command instead:

```
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
 release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml --
 repo-url=https://git.codelinaro.org/clo/la/tools/repo.git --repo-branch=qc-
 stable --no-clone-bundle
$ repo sync -j8 --no-tags -qc
$ mkdir -p qsdk/dl
$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-ieee1905-security/* qsdk
$ cp -rf apss_proc/out/proprietary/Wifi/qsdk-qca-wifi/* qsdk
$ cp -rf apss_proc/out/proprietary/Wifi/qsdk-qca-wlan/* qsdk
$ cp -rf wlan_proc/src/components/QCA8074_v2.0/qca-wifi-fw-src-component-cmn-WLAN.HK.*
 qsdk/dl/
$ cp -rf wlan_proc/pkg/wlan_proc/bin/QCA8074_v1.0/qca-wifi-fw-QCA8074_v1.0-
 WLAN.HK.*.tar.bz2 qsdk/dl/
$ cp -rf wlan_proc/pkg/wlan_proc/bin/QCA8074_v2.0/qca-wifi-fw-QCA8074_v2.0-
 WLAN.HK.*.tar.bz2 qsdk/dl/
$ tar xvf cnss_proc/src/components/qca-wifi-fw-src-component-cmn-WLAN.BL.*.tgz -C
 qsdk/dl
```

```
$ tar xvf cnss_proc/src/components/qca-wifi-fw-src-component-halphy_tools-
  WLAN.BL.*.tgz -C qsdk/dl
$ cp -rf cnss_proc/src/components/* qsdk/dl
$ cp -rf cnss_proc/bin/QCA9888/hw.2/* qsdk/dl
$ cp -rf cnss_proc/bin/AR900B/hw.2/* qsdk/dl
$ cp -rf cnss_proc/bin/QCA9984/hw.1/* qsdk/dl
$ cp -rf cnss_proc/bin/IPQ4019/hw.1/* qsdk/dl
$ cp -rf qca-wifi-fw-AR988* qsdk/dl
$ cp -rf apss_proc/out/proprietary/QSDK-Base/meta-tools/ .
$ cp -rf apss_proc/out/proprietary/QSDK-Base/common-tools/* qsdk/
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qsdk-qca-nss/* qsdk/
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-lib/* qsdk/
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-mcs-apps/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-nss-userspace/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-time-services/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-qmi-framework/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/gpio-debug/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-diag/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-cnss-daemon/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/athtestcmd/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/fw-qca-stats/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/btdaemon/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/minidump/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-rsrcmgr-bin/* qsdk
$ sed -i '/QCAHKSWPL_SILICONZ/c\PKG_VERSION:=WLAN.HK.2.9.r3-00031-
  QCAHKSWPL_SILICONZ-1' qsdk/qca/feeds/qca_hk/net/qca-hk/Makefile
$ cp apss_proc/out/proprietary/QSDK-Base/qca-nss-fw-eip-hk/BIN-EIP*.HK.* qsdk/dl/
$ cp nss_proc/out/proprietary/* qsdk/dl
$ rm -rfv qsdk/qca/feeds/qca/utils/ctrl_app_dut
```

Premium/ Enterprise/ LM512/LM512E

```
$ cp -rf apss_proc/out/proprietary/BLUETOPIA/qca-bluetopia/* qsdk
$ sed -i '0,/PKG_VERSION:=4.2.1.c1_26/s//PKG_VERSION:=4.2.1.4-00010/'
qsdk/qca/feeds/bluetopia/bluetopia/Makefile
```

```
$ cp -rf apss_proc/out/proprietary/RBIN-AFCAgent/qca-afc-bin/* qsdk
$ cd qsdk/qca/feeds/afc
$ find -maxdepth 1 ! -name qti-mfg-provision ! -name qti-encrypt ! -name . -exec rm -
  rv {} \;
```

32-bit:

```
$ cd <Chipcode Directory>
$ mkdir -p qsdk/staging_dir/target-arm/usr/lib/
```

Premium 32-bit	`$ cd qsdk/prebuilt/ipq807x_32/ipq_premium`
Enterprise 32-bit	`$ cd qsdk/prebuilt/ipq807x_32/ipq_enterprise`
LM512 32-bit	`$ cd qsdk/prebuilt/ipq807x_32/ipq_512`
LM512E 32-bit	`$ cd qsdk/prebuilt/ipq807x_32/ipq_512enterprise`
LM256 32-bit	`$ cd qsdk/prebuilt/ipq807x_32/ipq_256`

```
$ cp -rf ./libprovision.so ../../../staging_dir/target-arm/usr/lib/
$ cd <Chipcode Directory>
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

```
$ mkdir -p qsdk/staging_dir/target-arm/pkginfo/
$ touch qsdk/staging_dir/target-arm/pkginfo/qti-mfg-provision.provides
$ echo libprovision.so > qsdk/staging_dir/target-arm/pkginfo/qti-mfg-
  provision.provides
```

64-bit:

```
$ cd <Chipcode Directory>
$  mkdir -p qsdk/staging_dir/target-aarch64/usr/lib/
```

Premium 64-bit	`$ cd qsdk/prebuilt/ipq807x_generic/ipq_premium`
Enterprise 64-bit	`$ cd qsdk/prebuilt/ipq807x_generic/ipq_enterprise`

```
$ cp -rf ./libprovision.so ../../../staging_dir/target-aarch64/usr/lib/
$ cd <Chipcode Directory>
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

```
$ mkdir -p qsdk/staging_dir/target-aarch64/pkginfo/
$ touch qsdk/staging_dir/target-aarch64/pkginfo/qti-mfg-provision.provides
$ echo libprovision.so > qsdk/staging_dir/target-aarch64/pkginfo/qti-mfg-
 provision.provides
```

NOTE: Prebuilt ipk images (such as Qualcomm Bluetopia or qca-mcs-apps, qca-hyd-init, qca-hyd-son, qca-hyd-map) are in their respective folders:

`<Chipcode Directory>/qsdk/prebuilt/ipq807x_generic/ipq_<profile_name>/`	Contains all 64-bit ipk images
`<Chipcode Directory>/qsdk/prebuilt/ipq807x_32/ipq_<profile_name>/`	Contains all 32-bit ipk images

Where ipq_<profile_name> is ipq_premium, ipq_256 and ipq_512

2. Customers with EZMESH, HY-FI, WHC, Sigma-dut or WAPid packages:

	These files are fetched from ChipCode and copied to the working QSDK top-level directory:		
Premium/LM	**HY-FI and WHC**	Hyfi qsdk-whc qsdk-whcpy	apss_proc\out\proprietary\Hyfi apss_proc\out\proprietary\Wifi\qsdk-whc apss_proc\out\proprietary\Wifi\qsdk-whcpy
Premium	**WAPid**	qsdk-wapid	apss_proc\out\proprietary\Wapid\ qsdk-wapid
Premium/ Enterprise	**Sigma-DUT**	sigma-dut.tar.bz2	apss_proc\out\proprietary\sigma-dut\sigma-dut.tar.bz2
Premium/LM	**EZMESH-SRC**	qsdk-ezmesh-src	apss_proc\out\proprietary\RSRC-EZMESH\qsdk-ezmesh-src
Premium/LM	**EZMESH-BIN**	qsdk-ezmesh-bin	apss_proc\out\proprietary\RBIN-EZMESH\qsdk-ezmesh-bin
Premium/LM	**EZMESH-ALG**	qsdk-ezmesh-alg-bin	apss_proc\out\proprietary\RBIN-EZMESH-ALG\qsdk-ezmesh-alg-bin
Premium/LM	**EZ-ALG-SRC**	qsdk-ez-alg-src	apss_proc\out\proprietary\RSRC-EZMESH-ALG\qsdk-ezmesh-alg-src

Then run the additional code:

Premium/LM	**HY-FI and WHC**	`$ cp -rf apss_proc/out/proprietary/Hyfi/hyfi/* qsdk`
		`$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-whc/* qsdk`
		`$ mkdir qsdk/qca/feeds/qca-son-mem-debug/qca-son-mem-debug`
		`$ mv qsdk/qca/feeds/qca-son-mem-debug/Makefile qsdk/qca/feeds/qca-son-mem-debug/Config.in qsdk/qca/feeds/qca-son-mem-debug/qca-son-mem-debug`
		`$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-whcpy/* qsdk`
		`$ sed -i "s/@PACKAGE_whc-son/@PACKAGE_whc-map/g" qsdk/qca/feeds/qca-lib/qca-wifison-ext-lib/Makefile`

Premium	**WAPid**	`git clone <wapid ChipCode Directory>`
		`$ cd <wapid ChipCode Directory>`
		`$ git checkout r12.2.r3_00009.0`
		`$ cd ..`
		`$ cp -rf <wapid ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/Wapid/qsdk-wapid/* qsdk`
		Where wapid ChipCode Directory is qca-networking-2022-spf-12-2_qca_oem_wapid-src for this release.

Premium/ Enterprise	**Sigma-DUT:**	`$ git clone <sigma-dut ChipCode Directory>`
		`$ cd <sigma-dut ChipCode directory>`
		`$ git checkout r12.2.r3_00009.0`
		`$ cd ..`

		$ tar xjvf <sigma-dut ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/sigma-dut/sigma-dut.tar.bz2 -C qsdk Where sigma-dut ChipCode Directory is qca-networking-2022-spf-12-2_qca_oem_sigma-dut for this release.

Premium/LM	**EZMESH-SRC**	EZMESH customers only:
		$ git clone <*ezmesh ChipCode Directory*>
		$ cd <*ezmesh ChipCode Directory*>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <*ezmesh ChipCode Directory*>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RSRC-EZMESH/qsdk-ezmesh-src/* qsdk
		Where ezmesh ChipCode Directory is qca-networking-2022-spf-12-2_qca_oem_ezmesh-src for this release.
		#Skip these sed commands in EZMESH-SRC if also using **EZ-ALG-SRC** (that is both ezmesh-src and ezmesh-alg-src)
		$ sed -i 's/HYD_MODULE_STRATEGY=y/HYD_MODULE_STRATEGY=n/g' qsdk/qca/src/qca-ezmesh/ezmeshConfig.defs
		$ sed -i '0,/ifeq/{/ifeq/d;}' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '0,/endif/{/endif/d;}' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '/libezmeshalg \\/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '/libezmeshagentalg \\/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '/DUMP/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile

Premium/LM	**EZMESH-BIN**	EZMESH customers only:
		$ git clone <*ezmesh ChipCode Directory*>
		$ cd <*ezmesh ChipCode Directory*>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <*ezmesh ChipCode Directory*>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RBIN-EZMESH/qsdk-ezmesh-bin/* qsdk
		Where ezmesh ChipCode Directory is qca-networking-2022-spf-12-2_qca_oem_ezmesh-bin for this release.

Premium/LM	**EZMESH-ALG**	EZMESH customers only:
		$ git clone <*ezmesh ChipCode Directory*>
		$ cd <*ezmesh ChipCode Directory*>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <*ezmesh ChipCode Directory*>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RBIN-EZMESH-ALG/qsdk-ezmesh-alg-bin/* qsdk
		Where ezmesh ChipCode Directory is qca-networking-2022-spf-12-2_qca_oem_ezmesh-alg for this release

Premium/LM	**EZ-ALG-SRC**	EZMESH customers only:
		$ git clone <ezmesh ChipCode Directory>
		$ cd <ezmesh ChipCode Directory>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <ezmesh ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RSRC-EZMESH-ALG/qsdk-ezmesh-alg-src/* qsdk
		Where ezmesh ChipCode Directory is qca-networking-2022-spf-12-2_qca_oem_ez-alg-src for this release.

Premium/LM

EZ-ALG-STA

EZMESH customers only:

```
$ git clone <ezmesh ChipCode Directory>
$ cd < ezmesh ChipCode Directory>
$ git checkout r12.2.r3_00009.0
```

Where ezmesh ChipCode Directory is qca-networking-2022-spf-12-2_qca_oem_ez-alg-sta for this release.

EZMESH-SRC is required to compile ezmesh along with EZ-ALG-STA.

32-bit

Premium

```
$ cd  NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-EZMESH-ALG-STATIC/qca-ezmesh-alg-static-bin/prebuilt/ipq807x_32/ipq_premium
```

LM512

```
$ cd  NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-EZMESH-ALG-STATIC/qca-ezmesh-alg-static-bin/prebuilt/ipq807x_32/ipq_512
```

LM256

```
$ cd  NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-EZMESH-ALG-STATIC/qca-ezmesh-alg-static-bin/prebuilt/ipq807x_32/ipq_256
```

```
$ tar -xvf qca-ezmesh-alg-static*.ipk
$ tar -xvf data.tar.gz
$ cp ./usr/lib/libezmeshalg.a <Chipcode Directory>/qsdk/staging_dir/target-arm/usr/lib/
$ cd <Chipcode Directory>
```

64-bit

Premium

```
$ cd  NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-EZMESH-ALG-STATIC/qca-ezmesh-alg-static-bin/prebuilt/ipq807x_generic/ipq_premium

$ tar -xvf qca-ezmesh-alg-static*.ipk
$ tar -xvf data.tar.gz
$ cp ./usr/lib/libezmeshalg.a <Chipcode Directory>/qsdk/staging_dir/target-aarch64/usr/lib/
$ cd <Chipcode Directory>
```

CTL_APP_SRC

WFA customers only:

```
$ git clone <ctl_app_src ChipCode Directory>
$ cd <ctl_app_src ChipCode Directory>
$ git checkout r12.2.r3_00009.0
$ cd ..
$ cp -rf <ctl_app_src ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-SRC-CTRL-APP-DUT/qca-ctrl-app-dut/* qsdk
```

Where ctl_app_src *ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ctlapp-src** for this release.

CTL_APP_BIN

WFA customers only:

```
$ git clone <ctl_app_bin ChipCode Directory>
$ cd <ctl_app_bin ChipCode Directory>
$ git checkout r12.2.r3_00009.0
$ cd ..
$ cp -rf <ctl_app_bin ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-CTRL-APP-DUT/* <destination directory>
```

Where ctl_app_bin *ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ctlapp-bin** for this release.

4.4.3.3 Set up the QSDK build environment (one time)

QSDK supports the **premium, enterprise, LM512, LM512E, and LM256** profiles for Linux kernel 5.4.213**.** The QSDK framework is developed using Ubuntu (from version 16.04 to version 18.04) and Debian. However, QSDK framework regenerates critical tools required to compile firmware at build time. Thus, the framework is independent from the host environment. Although it is developed using the listed distributions, it is expected to work on others such as Red Hat, Mint, or Fedora.

The required tools can be installed manually (see Section 4.4.3.3.1). To use a script available in CLO, refer to Section 4.4.3.3.2.

4.4.3.3.1 Set up the build environment manually

This command is for Ubuntu 16.04 to 22.04 (for older/32-bit Debian/Ubuntu releases; customize it for other distributions):

```
$ sudo apt-get install gcc g++ binutils patch bzip2 flex make \
gettext pkg-config unzip zlib1g-dev libc6-dev subversion libncurses5-dev gawk
    sharutils \
curl libxml-parser-perl python-yaml ocaml-nox ocaml ocaml-findlib libssl-dev libfdt-
    dev
$ sudo apt-get install device-tree-compiler u-boot-tools
```

For Ubuntu 18.04 build hosts additionally, install:

```
$ sudo apt-get install libssl1.0-dev
```

For Ubuntu 20.04/22.04 build hosts additionally, install:

```
$ wget http://launchpadlibrarian.net/366014597/make_4.1-9.1ubuntu1_amd64.deb
$ sudo dpkg -i make_4.1-9.1ubuntu1_amd64.deb
```

4.4.3.3.2 Set up the build environment using script

Follow these steps to get the required tools installed in Ubuntu host for QSDK compilation. Verified on Ubuntu 16 and Ubuntu 18.

1. Assuming the repo init and sync to the release AU from CodeLinaro Forum (CLO) is done successfully, the setup_qsdk.sh script should be available in qsdk/scripts/setup_qsdk.sh
2. The script assumes that repo and git are already installed and aborts the execution otherwise.
3. If the repo init is not done, follow the steps:

   ```
   $ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
     release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml
   $ repo sync -j8 --no-tags -qc
   ```

4. The user running the script should have admin privileges. If not, the script will abort upon unsuccessful verification of sudo/admin access.
5. Run the script:

   ```
   $sudo bash -x setup_qsdk.sh
   ```

4.4.3.4 Create the QSDK build

Because the framework automatically downloads the open-source components needed during the build/make processing, ensure that an Internet connection is active on the build host when creating the build

1. Install the different feeds in the build framework:
```
$ cd qsdk
$ ./scripts/feeds update -a
$ ./scripts/feeds install -a -f
```
2. Copy the base configuration to use for the build. Choose **a** profile and use it to build with Linux 5.4.213 support.
3. Regenerate a complete configuration file and start the build:

Premium 32-bit
```
$ cp qca/configs/qsdk/ipq_premium.config .config
$ sed -i "s/TARGET_ipq807x_generic/TARGET_ipq807x_ipq807x_32/g" .config
$ mv prebuilt/ipq807x_32/ipq_premium/* prebuilt/ipq807x_32
```

Premium 64-bit
```
$ cp qca/configs/qsdk/ipq_premium.config .config
$ mkdir -p prebuilt/generic
$ cp -rf prebuilt/ipq807x_generic/ipq_premium/* prebuilt/generic
```

Enterprise 32-bit
```
$ cp qca/configs/qsdk/ipq_enterprise.config .config
$ sed -i "s/TARGET_ipq807x_generic/TARGET_ipq807x_ipq807x_32/g" .config
$ mv prebuilt/ipq807x_32/ipq_enterprise/* prebuilt/ipq807x_32
```

Enterprise 64-bit
```
$ cp qca/configs/qsdk/ipq_enterprise.config .config
$ mkdir -p prebuilt/generic
$ cp -rf prebuilt/ipq807x_generic/ipq_enterprise/* prebuilt/generic
```

LM512E 32-bit
```
$ cp qca/configs/qsdk/ipq_512enterprise.config .config
$ sed -i "s/TARGET_ipq807x_generic/TARGET_ipq807x_ipq807x_32/g" .config
$ mv prebuilt/ipq807x_32/ipq_512enterprise/* prebuilt/ipq807x_32/
```

LM512 32-bit
```
$ cp qca/configs/qsdk/ipq_512.config .config
$ sed -i "s/TARGET_ipq807x_generic/TARGET_ipq807x_ipq807x_32/g" .config
$ mv prebuilt/ipq807x_32/ipq_512/* prebuilt/ipq807x_32
```

LM256 32-bit
```
$ cp qca/configs/qsdk/ipq_256.config .config
$ sed -i "s/TARGET_ipq807x_generic/TARGET_ipq807x_ipq807x_32/g" .config
$ mv prebuilt/ipq807x_32/ipq_256/* prebuilt/ipq807x_32
```

Hy-Fi, WHC customers only:

- Starting with QCA_Networking_2020.SPF.11.3 release, SON and EasyMesh Features are compiled as separate build. SON build supports only SON features and EZMESH build supports only EasyMesh features.
- Starting with QCA_Networking_2021.SPF.11.4 release, MAP package qca-hyd-map is replaced with qca-ezmesh. Enable additional configuration to enable EZMESH packages.
- Use the SON package commands in this table for a SON features enabled build.
  Use the EZMESH package commands for an EasyMesh features enabled build.

4. Enable EZMESH configuration:

EZMESH package (Full Mode)
```
$ echo "CONFIG_PACKAGE_whc-mesh=n" >> .config
$ echo "CONFIG_PACKAGE_hyfi-mesh=n" >> .config
$ echo "CONFIG_PACKAGE_whc-map=y" >> .config
$ echo "CONFIG_PACKAGE_hyfi-map=y" >> .config
Additional configuration to enable EZMESH packages:
$ echo "CONFIG_PACKAGE_qca-ezmesh-cmn=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg=y" >> .config
```

	$ echo "CONFIG_PACKAGE_qca-ezmesh-ctrl=n" >> .config
	$ echo "CONFIG_PACKAGE_qca-ezmesh-agent=n" >> .config
	$ echo "CONFIG_PACKAGE_qca-ezmesh-agentalg=n" >> .config
	$ echo "CONFIG_PACKAGE_qca-ezmesh-alg-static=n" >> .config

EZMESH package (Co-located mode)	$ echo "CONFIG_PACKAGE_whc-mesh=n" >> .config
	$ echo "CONFIG_PACKAGE_hyfi-mesh=n" >> .config
	$ echo "CONFIG_PACKAGE_whc-map=y" >> .config
	$ echo "CONFIG_PACKAGE_hyfi-map=y" >> .config
	Additional configuration to enable EZMESH packages:
	$ echo "CONFIG_PACKAGE_qca-ezmesh-cmn=y" >> .config
	$ echo "CONFIG_PACKAGE_qca-ezmesh-ctrl=y" >> .config
	$ echo "CONFIG_PACKAGE_qca-ezmesh-agent=y" >> .config
	$ echo "CONFIG_PACKAGE_qca-ezmesh-alg=y" >> .config
	$ echo "CONFIG_PACKAGE_qca-ezmesh-agentalg=y" >> .config
	$ echo "CONFIG_PACKAGE_qca-ezmesh=n" >> .config
	$ echo "CONFIG_PACKAGE_qca-ezmesh-alg-static=n" >> .config

Additional configuration to enable ezmesh alg static:

EZMESH ALG STATIC	$ echo "CONFIG_EZMESH_ADD_STATIC_ALG=y" >> .config
	$ echo "CONFIG_PACKAGE_qca-ezmesh-alg=n" >> .config
	$ echo "CONFIG_PACKAGE_qca-ezmesh-alg-static=n" >> .config

- Use the commands in this table to enable qca-mad features.

  In QCA_Networking_2021.SPF.11.5 and older releases, qca-mad package is enabled as loadable modules in LM profiles (LM512, LM256), whereas in QCA_Networking_2022.SPF.12.0 ED1 release qca-mad package is disabled. If qca-mad package is required, then Customer needs to enable it explicitly through build config. This package shall be made as loadable module for QCA_Networking_2022.SPF.12.0 CSU1 release.

qca-mad package	$ echo "CONFIG_PACKAGE_qca-mad=y" >> .config

NOTE: Enabling both SON and EasyMesh Features in a build will increase image size and consume extra memory in Flash storage.

```
$ echo "CONFIG_BUILD_SHORTENED_PATH=y" >>.config
$ make defconfig
$ make V=s -j5
```

5. For 64-bit firmware, once the premium/enterprise 32-bit compilation completes, save the file: OpenWRT-ipq807x-u-boot*.elf, OpenWRT-ipq807x-lkboot*.elf, and OpenWRT-ipq807x_tiny-u-boot*.elf from the directory: **qsdk/bin/targets/ipq807x/ipq807x_32**

   Use this file to generate the complete 64-bit firmware image.

6. Download the required packages for the corresponding profile and create the image. Once the build is complete, the files are in these directories:

   32-bit: qsdk/bin/targets/ipq807x/ipq807x_32

   64-bit: qsdk/bin/targets/ipq807x/generic

32-bit	▪ OpenWRT-ipq807x-u-boot.elf (ELF) ▪ OpenWRT-ipq807x-ipq807x_32-squashfs-root.img (SquashFS) ▪ OpenWRT-ipq807x-ipq807x_32-ipq8074-hkxx-fit-uImage.itb (ITB) ▪ OpenWRT-ipq807x-ipq807x_32-ubi-root.img (IMG)
64-bit	▪ OpenWRT-ipq807x-generic-squashfs-root.img (SquashFS) ▪ OpenWRT-ipq807x-generic-ipq8074-hkxx-fit-uImage.itb (ITB) ▪ OpenWRT-ipq807x-generic-ubi-root.img (IMG)

4.4.4 Generate a complete firmware image

IPQ807x requires multiple images to be flashed for Bootup, including SBL1, SBL2, SBL3, RPM, TZ, CDT, MIBIB, NSS Images, Kernel, Filesystem, and so on. To simplify device loading using the usage from flash memory, the images are combined into a single Flattened Image Tree (FIT) image. The FIT can be flashed into the respective partition based on user configuration. More tools required on the Ubuntu 16.04 to 22.04 64-bit machine include:

1. Install DTC:
   `$sudo apt-get install device-tree-compiler`
2. Install Python 2.7; Install tools required to run packing single image:
   $sudo apt-get install libc6-i386
   $sudo apt-get install libgl1-mesa-dri:i386
3. Switch to the Qualcomm ChipCode directory:
   $ cd <Chipcode Directory>
4. Copy the flash config files to **common/build/ipq**
5. Find the single images under **common/build/bin** directory.

32-bit

```
$ cd <Chipcode Directory>
$ mkdir -p common/build/ipq
$ mkdir -p apss_proc/out/meta-scripts
$ cp qsdk/qca/src/u-boot-2016/tools/pack.py apss_proc/out/meta-scripts/pack_hk.py
$ sed -i 's#</linux_root_path>#/</linux_root_path>#' contents.xml
$ sed -i 's#</windows_root_path>#\\</windows_root_path>#' contents.xml
```

LM256

```
$ cp -rf trustzone_images/build/ms/bin/SANAANAA/devcfg_*.mbn common/build/ipq/
```

```
$ cp qsdk/bin/targets/ipq807x/ipq807x_32/openwrt* common/build/ipq
$ cp -r apss_proc/out/proprietary/QSDK-Base/meta-tools apss_proc/out/
$ cp -rf qsdk/bin/targets/ipq807x/ipq807x_32/dtbs/* common/build/ipq/
$ cp -rf skales/* common/build/ipq/
$ cp qsdk/staging_dir/host/bin/ubinize common/build/ipq/
$ cd common/build
$ sed '/lk/d' -i update_common_info.py
$ export BLD_ENV_BUILD_ID=<profile-name>
$ python update_common_info.py 32
```

(Where <profile-name> is P,E,LM512E, LM256 or LM512)

64-bit

```
$ cd <Chipcode Directory>
$ mkdir -p common/build/ipq_x64
$ mkdir -p apss_proc/out/meta-scripts
$ cp qsdk/qca/src/u-boot-2016/tools/pack.py apss_proc/out/meta-scripts/pack_hk.py
$ sed -i 's#</linux_root_path>#/</linux_root_path>#' contents.xml
$ sed -i 's#</windows_root_path>#\\</windows_root_path>#' contents.xml
$ cp qsdk/bin/targets/ipq807x/generic/openwrt* common/build/ipq_x64
```

Copy the saved openwrt-ipq807x-u-boot*.elf, openwrt-ipq807x-lkboot*.elf, and openwrt-ipq807x_tiny-u-boot*.elf from the 32-bit build to: common/build/ipq_x64

```
$ cp -r apss_proc/out/proprietary/QSDK-Base/meta-tools apss_proc/out/
$ cp -rf qsdk/bin/targets/ipq807x/generic/dtbs/* common/build/ipq_x64/
$ cp -rf skales/* common/build/ipq_x64/
$ cp qsdk/staging_dir/host/bin/ubinize common/build/ipq_x64/
$ cd common/build
```

```
$ sed '/lk/d' -i update_common_info.py
$ export BLD_ENV_BUILD_ID=<profile-name>
$ python update_common_info.py 64
```

(Where <*profile-name*> is P, E).

4.4.5 Flash the complete default software image

4.4.5.1 Set up the flash environment

1. Ensure that the board console port is connected to the PC using these RS232 parameters:
   - 115200 bps
   - 8N1
2. Confirm that the PC is connected to the board using one of the Ethernet ports. The PC must have a TFTP server launched and listening on the interface to which the board is connected. The interface must have an IP address configured manually. At this stage power up the board and after a few seconds, press any key during the countdown.

4.4.5.2 Standard board configuration

1. Copy the **xxxx-ipq807x-single.img** to the TFTP server root directory.
2. Check hardware jumper configuration according to reference board and default memory configuration (see the appropriate board setup guide for more information) to verify which flash memory the board is booting from.
3. Confirm the machine ID, Meta version, profile, and single image.
4. Set the IP address and server IP using the TFTP process:
   ```
   set ipaddr 192.168.1.11
   set serverip 192.168.1.xx (TFTP server address)
   ping ${serverip}
   ```
   (set bootargs console=ttyMSM0,115200n8)
   tftpboot 0x44000000 xxxx-ipq807x-single.img
   ```
   If using a NOR flash, execute this command: sf probe
   ```
5. Flash the image:
   ```
   imgaddr=0x44000000 && source $imgaddr:script
   ```
6. Power cycle the board after step 5 has completed, as indicated by printing of the U-boot prompt (may be hundreds of seconds depending on image size and memory technology).
7. For an Arithmetic exception issue in the fw_printenv/fw_setenv commands in eMMC boot, run these commands:
   ```
   cat /etc/fw_env.config |  awk''{$NF"";sub(/[ \t]+$/"")}'' 
    >/tmp/fw_env.config
   mv /tmp/fw_env.config /etc/fw_env.config
   ```
8. Once the single image has booted, transfer this package to the AP /tmp directory from ./**qsdk/prebuilt/ipq807x/** or ./**qsdk/prebuilt/ipq807x_64/**, depending on whether it is a 32-bit or 64-bit image.
   ```
   bluetopia_4.2.1.c1_26-1_arm_cortex-a7_neon-vfpv4.ipk
   ```
9. Install these packages from the /tmp directory:
   ```
   opkg install bluetopia_4.2.1.c1_26-1_arm_cortex-a7_neon-vfpv4.ipk
   ```

NOTE: For Qualcomm Wi-Fi SON, perform steps 11, 12, and 13.

10. Once the single image has booted, transfer the SON packages for SON features or EZMESH packages for EasyMesh features to the AP /tmp directory from: **./qsdk/prebuilt/ipq80xx/** or **./qsdk/prebuilt/ ipq80xx _64/**, depending on whether the booted image is 32-bit or 64-bit.

SON package (32-bit)	qsdk/prebuilt/ipq807x/qca-hyd-init_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq807x/qca-hyd-son_ge474d2e-1_ipq.ipk
SON package (64-bit)	qsdk/prebuilt/ipq807x_64/qca-hyd-init_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq807x_64/qca-hyd-son_ge474d2e-1_ipq.ipk
EZMESH package (32-bit)	**To support Full mode:** qsdk/prebuilt/ipq807x/qca-ezmesh-cmn_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq807x/qca-ezmesh_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq807x/qca-ezmesh-alg_ge474d2e-1_ipq.ipk **To support Co-located mode:** qsdk/prebuilt/ipq807x/qca-ezmesh-cmn_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq807x/qca-ezmesh-ctrl_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq807x/qca-ezmesh-agent_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq807x/qca-ezmesh-alg_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq807x/qca-ezmesh-agentalg_ge474d2e-1_ipq.ipk
EZMESH package (64-bit)	**To support Full mode:** qsdk/prebuilt/ipq807x_64/qca-ezmesh-cmn_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq807x_64/qca-ezmesh_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq807x_64/qca-ezmesh-alg_ge474d2e-1_ipq.ipk **To support Co-located mode:** qsdk/prebuilt/ipq807x_64/qca-ezmesh-cmn_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq807x_64/qca-ezmesh-ctrl_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq807x_64/qca-ezmesh-agent_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq807x_64/qca-ezmesh-alg_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq807x_64/qca-ezmesh-agentalg_ge474d2e-1_ipq.ipk

NOTE: ge474d2e is git commit ID, which may also be changed.

11. Install these packages from the /tmp directory: Choose the SON package to enable SON features, or choose EZMESH packages to enable MAP features:

SON package	opkg install qca-hyd-init_ge474d2e-1_ipq.ipk opkg install qca-hyd-son_ge474d2e-1_ipq.ipk
EZMESH package	**To support Full mode:** opkg install qca-ezmesh-cmn_ge474d2e-1_ipq.ipk opkg install qca-ezmesh_ge474d2e-1_ipq.ipk opkg install qca-ezmesh-alg_ge474d2e-1_ipq.ipk **To support Co-located mode:** opkg install qca-ezmesh-cmn_ge474d2e-1_ipq.ipk opkg install qca-ezmesh-ctrl_ge474d2e-1_ipq.ipk opkg install qca-ezmesh-agent_ge474d2e-1_ipq.ipk opkg install qca-ezmesh-alg_ge474d2e-1_ipq.ipk opkg install qca-ezmesh-agentalg_ge474d2e-1_ipq.ipk

NOTE: ge474d2e is the git commit ID, which may also be changed.

12. The HYD daemon application binary name is changed from hyd to hyd-son for the SON package. Similarly, the WSPLCD daemon application binary name is changed from wsplcd to wsplcd-son and wsplcd-map for the SON and EZMESH packages, respectively. The symbolic link to the older binary name must be created after package installation to ensure that scripts and tools work with the new binary name. Because application binary name has changed, a crash dump is created based on new binary name. Customers must take care of updating any scripts and tools that use or depend on the older binary name.

    Execute these commands to create a symbolic softlink to the older binary name:

SON package	`ln -s /usr/sbin/wsplcd-son /usr/sbin/wsplcd` `ln -s /usr/sbin/hyd-son /usr/sbin/hyd`
EZMESH package	`ln -s /usr/sbin/wsplcd-map /usr/sbin/wsplcd`

4.4.5.3 Upgrade the firmware

This release can upgrade board images without the need for a TFTP server. After using U-Boot to flash an initial image and booting the device, use the web interface for future upgrades. After using U-Boot to flash an initial image and boot the device, future upgrades can be done from either the OpenWRT web interface or serial console. Upgrade the existing flash image using either the appropriate single image file or the apps image file generated by **update_common_info.py**.

Upgrading image files via the web interface takes several minutes to complete depending on factors such as memory technology, vendor, image size, browser connectivity, and network load. Completion of the flash upgrade process is signaled by the refresh of the OpenWRT login page, or lack of further messages on the serial console.

- An invalid image remains in the flash memory if the upgrade process is interrupted, such as if system power is lost during upgrade or a key-press event is detected on the serial console port.
- If a board is configured for failsafe boot, and the U-boot environment variable *bootargs* contains any reference to the rootfs or other partitions managed by the failsafe scheme. In this case, the sysupgrade process may only be partially successful: the board may boot subsequently, and the image may be partly updated.
- Using the **sysupgrade** command from the Linux console requires the sysupgrade image to be visible:
  - Within the Linux filesystem supplied using development host SSH or TFTPD server, OR
  - On an NFS server that is mounted to the local file system, OR
  - Transferred using a USB storage device
- The sysupgrade feature can only be used to update the image in the memory the board boots from and is used by the running kernel. If the board supports multiple memory technologies, use U-Boot to change the image in memories that the board did not boot from.

See the *IPQ807x/IPQ807xA/IPQ817x SoC Software User Guide* (80-YA728-4) for more information on Sysupgrade support.

See the *QCN90xx Firmware Build Application Note* (80-Y9005-29) for more information.

4.4.5.4 eMMC boot GPT backup partition handling

The location of the GPT backup partition in the final flash image is defined in the file:
./NHSS.QSDK.12.2/apss_proc/out/proprietary/QSDK-Base/meta-tools/ipq807x/config.xml

In the lines:

```
<data type”"EMMC_PARAMETE”">
<total_block>7634944</total_block>
<partition_mbn>gpt_main0.bin</partition_mbn>
<partition_mbn_backup>gpt_backup0.bin</partition_mbn_backup>
```

This file is copied to the **./common/build/ipq** or **./common/build/ipq_x64** directory for 32-bit and 64-bit image builds, respectively. It assumes an eMMC device size of 4 GB; therefore, the GPT backup partition will be incorrectly located for boards that are provisioned with different size eMMC device; for example, HK01.

The configuration file can be corrected by an additional sed command after the ChipCode repository has been checked out: for example, for the 8 GB device provisioned on the HK01 board, use:

```
$sed''s#<total_block>7634944</total_block>#<total_block>15269888</total_block>'' -
 i ./NHSS.QSDK.12.2/apss_proc/out/proprietary/QSDK-Base/meta-
 tools/ipq807x/config.xml
```

Sysupgrade behavior is not guaranteed when the GPT backup partition is not correctly located at the end of the eMMC image.

4.4.5.5 Bring up 802.11ay interface on IPQ8074A platforms

1. After flashing the image, copy the ipk files on the AP to the /root dir and install the IPKs in this sequence using the correct ipk names:
   opkg install wigig-firmware_WIGIG.TLN.*-*-WIGIGTLNZ-1_ipq.ipk
   opkg install wigig-mac80211_1_ipq.ipk
   opkg install kmod-qca-wigig-driver_*.ipk
   opkg install qca-wigig-debug-tools_*.ipk
   opkg install qca-fst-manager_*.ipk
2. Reboot the device.
3. Enable 802.11ay using the UCI commands:
   ```
   $uci set wireless.radio0.disabled=0
   $uci set wireless.radio0.edmg_channel=10
   $uci set wireless.@wifi-iface[0].ssid""xxxxxx"
   $uci commit wireless
   ```
4. Reboot the device.
5. Split WiGig tasks among CPU cores ():
   ```
   $uci set wireless.mac80211=mac80211
   $uci set wireless.mac80211.enable_smp_affinity=1
   $uci commit wireless
   /etc/init.d/network restart
   ```
6. Verify the changes.
   a. Run this shell command to find all wigig N_MSI:
      ```
      cat /proc/interrupts |grep wil6210
      ```
      For example:
      ```
      118: 0 0 0 0 GIC 448 Edge wil6210_tx
      119: 0 0 0 0 GIC 449 Edge wil6210_rx
      ```
   b. Validate the changes using the command:
      ```
      root@OpenWrt:~# cat /proc/irq/xxx/smp_affinity
      8 (core 3)
      root@OpenWrt:~# cat /proc/irq/yyy/smp_affinity
      ```
7. Check CPU use:
   ```
   mpstat -P ALL 1
   ```

4.4.5.5.1 Enable FST on IPQ8074A platforms

1. Enable FST using the UCI commands:
   ```
   $uci set wireless.wifi0.disabled=0
   ```
   For this instruction, assume by default ath0/wifi0 is used; change accordingly if other Wi-Fi interfaces used.
   ```
   $uci set wireless.@wifi-iface[1].ssid""xxxxxx" – same as 802.11ay SSID
   $uci commit wireless
   /etc/init.d/network restart
   ```

2. Get MAC address of the Wi-Fi interface:
```
cat /sys/class/net/ath0/address
```
3. Enable FST manager using UCI commands:
```
# uci set fst.config.interface1=ath0 <by default, configuration uses ath0;
  change accordingly if other wifi interfaces used>
uci set fst.config.interface2_mac=<MAC address from point 11>
uci set fst.config.disabled=0
uci commit fst
```
4. Reboot the device.
5. Check if the FST manager is enabled (the assumption ath0 is used for FST configuration).
   a. Check the command **brctl show** lists bond1 as part of the bridge interface. Both ath0 and wlan0 are no longer part of br-lan.
   b. Check if both ath0 and wlan0 are listed as slave interfaces in **/proc/net/bonding/bond1**.
   c. Check if **ps | grep fst** shows "/usr/sbin/fstman /var/run/hostapd/global -s ath0".

4.4.6 Create customized IPQ807x Wi-Fi firmware images

Flash memory layout includes a dedicated partition for the Wi-Fi firmware and BDF information. The release includes squashfs and ubifs images in the final image created by the last steps of section 4.4.4. Use these steps to update or recreate the squashfs/ubifs Wi-Fi firmware image and the complete final image.

NOTE: These steps are not needed if the BDF and PIL files are not customized.

1. Install these tools in the workstation:
   - mtd-utils: sudo apt-get install mtd-utils
   - mksquashfs4 binary from:
     < ChipCode Directory>**IPQ8074.ILQ.12.2/common/build** to /usr/bin or /usr/sbin or any location included in $PATH
2. Download the WLAN IPQ807x firmware BIN tar ball from the corresponding SP release distro.
3. Download the WLAN SquashFS image wifi_fw_ipq8074_qcn9000_squashfs_v2.img from WLAN.HK.2.3/wlan_proc/build/ms/bin/8074.wlanfw.eval_v2/FW_IMAGES/
4. Extract the firmware binary tarball:
```
mkdir -p <Chipcode Directory>/fwtemp
cp -rfv wlan_proc/pkg/wlan_proc/bin/QCA8074_v2.0/qca-wififw-QCA8074_v2.0-
  WLAN.HK.*-*  fwtemp
cd fwtemp
tar -xvf qca-wifi-fw-QCA8074_v2.0-WLAN.HK.*-*

mkdir -p fwtemp
cp <download path>/wifi_fw_ipq8074_qcn9000_squashfs_v2.img ./fwtemp
cd fwtemp
unsquashfs wifi_fw_ipq8074_qcn9000_squashfs_v2.img
```
5. Create a staging directory to hold the BD and PIL files.
```
mkdir staging_dir
cp -rfv qca-wifi-fw-WLAN.HK.*/PIL_IMAGES/* staging_dir
cp -rfv qca-wifi-fw-WLAN.HK.*/bdwlan* staging_dir
```
   (Replace the BDF file with the custom versions as needed.)

6. To generate the firmware squash image for **NOR/eMMC** (this example assumes mksquashfs4 is in $PATH; use the exact installed path of mksquashfs4):
```
mksquashfs4 staging_dir/ wifi_fw.squashfs -nopad -noappend -root-owned -comp
 xz -Xpreset 9 -Xe -Xlc 0 -Xlp 2 -Xpb 2 -Xbcj arm -b 256k -processors 1
dd if=wifi_fw.squashfs of=wifi_fw_squashfs.img bs=2k conv=sync
```
7. To generate the firmware UBI image for **NAND** (this example assumes mksquashfs4 is in $PATH; use the exact installed path of mksquashfs4):
```
mksquashfs4 staging_dir/ wifi_fw.squashfs -nopad -noappend -root-owned -comp
 xz -Xpreset 9 -Xe -Xlc 0 -Xlp 2 -Xpb 2 -Xbcj arm -b 256k -processors 1
```

NOTE: If the squashfs image is already generated as in the case of NOR/eMMC, ignore these steps and directly ubinize the image:

8. Create the ubinize image with the ubinize utility, which is part of mtd-util.
9. Create a **ubi-wifi.cfg** file with these contents:
```
[wifi_fw]
# Volume mode (other option is static)
mode=ubi
# Source image
image=wifi_fw.squashfs
# Volume ID in UBI image
vol_id=1
# Allow for dynamic resize
vol_type=dynamic
# Volume name
vol_name=wifi_fw
# vol_flags=autoresize
```
   Then run these commands:
```
ubinize -m 2048 -p 128KiB -o wifi_fw.ubi ubi-wifi.cfg
dd if=wifi_fw.ubi of=wifi_fw_ubi.img bs=2k conv=sync
```
10. To create the final single image with the updated or recreated firmware image:
```
cp <Chipcode Directory>/fwtemp/Wi-Fi_fw_squashfs.img <Chipcode
 Directory>/wlan_proc/build/ms/bin/FW_IMAGES/
cp <Chipcode Directory>/fwtemp/Wi-Fi_fw_ubi.img <Chipcode
 Directory>/wlan_proc/build/ms/bin/FW_IMAGES/
```
11. Run this command again:
```
export BLD_ENV_BUILD_ID=<profile-name>
python update_common_info.py
```
   Where <*profile-name*> is E, P.

For instructions on how to flash the firmware images separately, see the *IPQ807x/IPQ807xA/IPQ817x SoC Software User Guide* (80-YA728-4).

4.4.7 Create customized IPQ807x + QCN90xx Wi-Fi firmware images

Flash memory layout includes a dedicated partition for the Wi-Fi firmware and BDF information. The release includes squashfs and ubifs images in the final image created by the last steps of Section 4.4.4. Use these steps to update or recreate the squashfs/ubifs Wi-Fi firmware image and the complete final image.

NOTE: These steps are not needed if the BDF and PIL files are not customized.

1. Install these tools in the workstation:
   - mtd-utils: sudo apt-get install mtd-utils
   - mksquashfs4 binary from: **<C*hipCode Directory*>IPQ8074.ILQ.12.2/common/build** to /usr/bin or /usr/sbin or any location included in $PATH
2. Download the WLAN SquashFS image wifi_fw_ipq8074_qcn9000_squashfs_v2.img from WLAN.HK.2.3/wlan_proc/build/ms/bin/8074.wlanfw.eval_v2/FW_IMAGES/
3. Extract the squashfs image in a temp directory

```
mkdir -p fwtemp
cp <download path>/wifi_fw_ipq8074_qcn9000_squashfs_v2.img ./fwtemp
cd fwtemp
unsquashfs wifi_fw_ipq8074_qcn9000_squashfs_v2.img
```

   All extracted files are in the folder: ./squashfs-root
4. Copy/modify any files in the file: ./squashfs-root
5. To generate the firmware SquashFS image for **NOR/eMMC** (this example assumes mksquashfs4 is in $PATH; use the exact installed path of mksquashfs4):

```
mksquashfs4 squashfs-root/ wifi_fw.squashfs -nopad -noappend -root-owned -comp
   xz -Xpreset 9 -Xe -Xlc 0 -Xlp 2 -Xpb 2 -Xbcj arm -b 256k -processors 1
dd if=wifi_fw.squashfs of=wifi_fw_squashfs.img bs=2k conv=sync
```

6. To generate the firmware UBI image for **NAND** (this example assumes mksquashfs4 is in $PATH; use the exact installed path of mksquashfs4):

```
mksquashfs4 squashfs-root/ wifi_fw.squashfs -nopad -noappend -root-owned -comp
   xz -Xpreset 9 -Xe -Xlc 0 -Xlp 2 -Xpb 2 -Xbcj arm -b 256k -processors 1
```

NOTE: If the squashfs image is already generated as in the case of NOR/eMMC, ignore these steps and directly ubinize the image:

7. Create the ubinize image with the ubinize utility, which is part of mtd-util.
8. Create a **ubi-wifi.cfg** file with these contents:

```
[wifi_fw]
# Volume mode (other option is static)
mode=ubi
# Source image
image=wifi_fw.squashfs
# Volume ID in UBI image
vol_id=1
# Allow for dynamic resize
vol_type=dynamic
# Volume name
vol_name=wifi_fw
# vol_flags=autoresize
```

   Then run these commands:

```
ubinize -m 2048 -p 128KiB -o wifi_fw.ubi ubi-wifi.cfg
dd if=wifi_fw.ubi of=wifi_fw_ubi.img bs=2k conv=sync
```

9. To create the final single image with the updated or recreated firmware image:

```
cp <Chipcode Directory>/fwtemp/Wi-Fi_fw_squashfs.img <Chipcode
   Directory>/wlan_proc/build/ms/bin/FW_IMAGES/
cp <Chipcode Directory>/fwtemp/Wi-Fi_fw_ubi.img <Chipcode
   Directory>/wlan_proc/build/ms/bin/FW_IMAGES/
```

10. Run this command again:

```
export BLD_ENV_BUILD_ID=<profile-name>
python update_common_info.py
```

    Where <*profile-name*> is E, P.

For instructions on how to flash the firmware images separately, see the *IPQ807x/IPQ807xA/IPQ817x SoC Software User Guide* (80-YA728-4).

4.5 U-Boot device tree optimization

To save flash space and remove redundant information. Some of the device tree in U-Boot has been removed/reused.

- HK01.C3 now uses HK01 device tree
- AC02 now uses AC01 device tree
- OAK03 now uses HK01 device tree

If this re-use feature is not needed, revert the following commits:

```
f2b16c5a1e825f59cadd9c0982adf5b71b7d6c07
c8365407da56a1fc67ddb389da12649bd3c2c99b
```

5 IPQ6018.ILQ.12.2 CSU1

5.1 Supported features

- CPU/platform
  - Quad ARM Cortex A53 at 1.8 GHz, 64-bit ISA v8 instruction set
  - 32 KB/32 KB I$/D$ and 512 KB L2$
  - Floating point and NEON SIMD DSP for each core
  - DDR4/DDR3L, USB3, PCIe, all serial interfaces
- Wireless connectivity
  - For IPQ601x: 5 GHz antenna configuration
    - 2 x 2/2S-80 MHz
  - 2.4 GHz antenna configuration
    - 2 x 2/2S-40 MHz
- Networking
  - Single-core multithreaded network accelerator (NPU)
  - Advanced classification, policing, queuing, forwarding at wire speed (PPE)
  - Hardware crypto offload engine
  - USXGMII/XFI, SGMII, SGMII+, PSGMII
- Standard programming model
  - Linux 5.4.213 (32-bit and 64-bit)
  - QSDK based on OpenWRT 19.07

IPQ60xx + QCN90xx features

- 4x4/160 MHz 11ax PCIe Radio
- 2.4 GHz, 5 GHz, 6 GHz full band support
- Dual Lane PCIe Gen 3
- Package: 11.1 x 12 FCBGA, 0.65 mm ball pitch
- WLAN
  - Dual-synthesizer WLAN radio up to 160 MHz band width, support aSA, ADFS
  - Supports 20/40 MHz in 2.4 GHz
  - Supports 20/40/80/160 MHz in 5 GHz
  - Supports 20/40/80/160 MHz in 6 GHz
  - Supports up to 1024 QAM (4 Spatial stream (SS)) and 4096 QAM (2 SS)

- Supported standards
  - IEEE802.11a/b/g/n/ac/ax
  - IEEE 802.11d/e/h/i/j/k/r/u/v/w

5.2 Restrictions on software while using it for testing

DL OFDMA Test Tones for FCC Conformance Testing	*WLAN Conformance Testing and Compliant Power Configuration for 802.11ax Chipsets Information to Share with Test Lab* (80-YB952-3) have been updated with FCC conformance test guidance for DL OFDM transmissions. Test tone XML files to support the FCC testing using QSPR and QRCT are available for the WLAN AP software release at these paths: ▪ In the firmware package, the file is available at "/bdfUtil". ▪ In the WLAN source, the file is available at wlan_proc\wlan\phyrf_svc\tools\bdfExcel\qca8074\CTL_installer_python_scripts\FCC_interim_Guidance_Test_Tones.zip

5.3 Supported hardware for this SP

RD	RDP	Chipset part number
CP01	RDP361 RDP352	IPQ6028 IPQ6018
CP03	RDP365	IPQ6018
CP04	RDP421	IPQ6018
AP.CP01.7	RDP420	IPQ6010

5.4 Build and load the image for IPQ6018.ILQ.12.2 CSU1

1. Download the QTI proprietary code from Qualcomm ChipCode Portal (see Section 5.4.1).
2. Download other components from external websites for QSDK while building the default configuration (see Section 5.4.2).
3. Generate the firmware:Kernel
   a. Re-assemble the code (see Section 5.4.3.1).
   b. Create the QSDK build (see Section 5.4.3.3).
   c. Generate a complete firmware image (see Section 5.4.4).
4. Install the image in the device flash memory and boot using the image from the flash (see Section 5.4.5).

Users should be familiar with directory structures that contain SP images for the different subsystems before downloading the code and building the images for loading. For each SP included in an SPF, SP binary files are generated from the SI binary files of only a subset of the included Sis. In an SPF, some Sis may support multiple SPs while others may only support one SP.

5.4.1 Download packages available through Qualcomm ChipCode Portal

Qualcomm Technologies proprietary code is available from Qualcomm ChipCode Portal. A web/GUI interface and a secure git server both allow access to this code.

- Browse available packages and obtain the download URL at: https://chipcode.qti.qualcomm.com/
- For more information on cloning the code, see: https://chipcode.qti.qualcomm.com/helpki/cloning-code-from-a-repository
- For more information on installation and configuration of the correct version of git and OpenSSL on both Windows and Linux platforms, see: https://chipcode.qti.qualcomm.com/helpki

  These versions are required to support the authentication methods used by Qualcomm ChipCode Portal.

5.4.2 Download packages from external websites

For details on downloading packages from external sites, see Section 2.2.2.

5.4.3 Generate the firmware for IPQ6018.ILQ.12.2 CSU1

The firmware image can be generated by using a script (see Section 5.4.3.1) or manually (see Section 5.4.3.2).

5.4.3.1 Script to generate complete firmware image:

NOTE: The customer build script is applicable **only if** OpenWRT framework is used.

Meta generation script for 12.2 CSU1 (meta_generation_script.py) aims to simplify the release notes instructions required to be executed by the customers. Follow these steps:

```
$ mkdir $BUILD_WS
```

(Assume that BUILD_WS is the workspace directory)

```
$ cd $BUILD_WS
$ git clone <Chipcode Directory>
$ cd <Chipcode Directory>
$ git checkout r12.2.r3_00009.0
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

Where the clone required deliverable distro is in the same path where the OEM distro is cloned using git clone commands and checkout (git checkout) to the release tag on each deliverable distro.

```
$ rm -rf IPQ5018.ILQ.12.* IPQ9574.ILQ.12.* IPQ8074.ILQ.12.* IPQ5322.ILQ.*
 TZ.WNS.4.0 TZ.WNS.5.3 TZ.BF.4.0.8 BOOT.BF.* BOOT.XF.0.3.1*  BTFW.MAPLE.*
 TMEL.WNS.* WLAN.WBE*
$ cp -rf */* .
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
 release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml
```

If this command does not work, use this repo init command instead:

```
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
 release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml --
 repo-url=https://git.codelinaro.org/clo/la/tools/repo.git --repo-branch=qc-
 stable --no-clone-bundle
$ repo sync -j8 --no-tags -qc
```

```
$ cd common/build
$ python meta_generation_script.py -c [Chipcode_tag] -s [SP] -p [Profile] -b
  [32/64] -d [Distro_list] -m [MESH]--path $BUILD_WS
```

For example:

```
python meta_generation_script.py -c r12.2.r3_00009.0 -s IPQ6018.ILQ.12.2 -p P -b
  32 -d HYFI,WHC,EZMESH_SRC,EZMESH_BIN,EZMESH_ALG -m EZMESH_FULL--
  path /local/mnt2/workspace/
```

The IPQ6018.ILQ.12.2 CSU1 release supports these configurations:

Software Product	Profile and bit config	Supported Deliverables Combo	Mesh options
IPQ6018.ILQ.12.2	P – 32,64	HYFI, WHC, WAPID, SIGMA-DUT, EZMESH_SRC, EZMESH_BIN, EZMESH_ALG, EZ_ALG_SRC, CTL_APP_SRC, CTL_APP_BIN, OEM	EZMESH_FULL, EZMESH_COLOCATED
	LM512 – 32	HYFI, WHC, EZMESH_SRC, EZMESH_BIN, EZMESH_ALG, EZ_ALG_SRC, CTL_APP_SRC, CTL_APP_BIN, OEM	EZMESH_FULL, EZMESH_COLOCATED
	E – 32,64	SIGMA-DUT, CTL_APP_SRC, CTL_APP_BIN, OEM	NA
	LM256 – 32	HYFI, WHC, EZMESH_SRC, EZMESH_BIN, EZMESH_ALG, EZ_ALG_SRC, CTL_APP_SRC, CTL_APP_BIN, OEM	EZMESH_FULL, EZMESH_COLOCATED

NOTE: When no additional deliverables are there, use -d OEM while triggering meta_generation_script.

NOTE: If no Mesh options are left, remove -m <Mesh option> while triggering meta_generation_script.

Single images are in these directories:

32-Bit	`$BUILD_WS/C/<Profile>Timestamp/<OEM distro>/common/build/bin` Where <Profile> is P/E/512/256
64-Bit	`$BUILD_WS/C/<Profile>Timestamp/64/<OEM distro>/common/build/bin` Where <Profile> is P/E

5.4.3.2 Manually generate complete firmware image

This example assumes that all packages listed in sections 5.4.1 and 5.4.2 are obtained using the **git clone** command and placed in the top-level directory.

NOTE: 64-bit single image generation requires the U-boot elf files generated by compiling the 32-bit build. Complete the 32-bit compilation before following the 64-bit build generation steps.

1. Re-assemble the code and generate the QSDK framework:

```
$ git clone <Chipcode Directory>
$ cd <Chipcode Directory>
$ git checkout r12.2.r3_00009.0
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

2. After the copy of the existing Git repository is completed, change the directories where the files are present as described in this table before running the Repo command:

Use git to obtain these files from ChipCode:	Local directory path to files fetched by git from ChipCode:
qsdk-qca-wifi qsdk-qca-wlan	NHSS.QSDK.12.2\apss_proc\out\proprietary\Wifi
qsdk-ieee1905-security qsdk-whc qsdk-whcpy	NHSS.QSDK.12.2\apss_proc\out\proprietary\Hyfi
qca-lib qca-mcs-apps qca-nss-userspace qca-time-services qca-qmi-framework gpio-debug qca-diag athtestcmd qca-cnss-daemon fw-qca-stats meta-tools common-tools qsdk-qca-nss qca-nss-fw-eip-cp btdaemon minidump qca-rsrcmgr-bin	NHSS.QSDK.12.2\apss_proc\out\proprietary\QSDK-Base
qca-bluetopia	NHSS.QSDK.12.2\apss_proc\out\proprietary\BLUETOPIA
qca-wifi-fw-src-component-cmn-WLAN.HK.*.tgz qca-wifi-fw-QCA6018_v1.0-WLAN.HK.*.tar.bz2	WLAN.HK.2.9\wlan_proc\src\components WLAN.HK.2.9\wlan_proc\pkg\wlan_proc\bin\QCA6018
qca-wifi-fw-src-component-cmn-WLAN.BL.*.tgz qca-wifi-fw-src-component-halphy_tools-WLAN.BL.*.tgz qca-wifi-fw-QCA9888_hw_2-WLAN.BL.*.tar.bz2 qca-wifi-fw-AR900B_hw_2-WLAN.BL.*.tar.bz2 qca-wifi-fw-QCA9984_hw_1-WLAN.BL.*.tar.bz2 qca-wifi-fw-IPQ4019_hw_1-WLAN.BL.*.tar.bz2	WLAN.BL.3.19\cnss_proc\src\components WLAN.BL.3.19\cnss_proc\bin
qca-wifi-fw-AR9887_hw_1-CNSS.PS.*.tar.bz2 qca-wifi-fw-AR9888_hw_2-CNSS.PS.*.tar.bz2	CNSS.PS.3.19
BIN-NSS.FW.*.tar.bz2	NSS.FW.12.2\nss_proc\out\proprietary
qca-afc-bin	NHSS.QSDK.12.2\apss_proc\out\proprietary\RBIN-AFCAgent

3. Execute following commands to copy above deliverables to qsdk directory, and continue generating the QSDK framework:

```
$ rm -rf IPQ5018.ILQ.12.* IPQ9574.ILQ.12.* IPQ8074.ILQ.12.* IPQ5322.ILQ.*
   TZ.WNS.4.0 TZ.WNS.5.3 TZ.BF.4.0.8 BOOT.BF.* BOOT.XF.0.3.1*  BTFW.MAPLE.*
   TMEL.WNS.* WLAN.WBE*
$ cp -rf */* .
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
   release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml
```

If this command does not work, use this repo init command instead:

```
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
   release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml --
   repo-url=https://git.codelinaro.org/clo/la/tools/repo.git --repo-branch=qc-
   stable --no-clone-bundle
```

```
$ repo sync -j8 --no-tags -qc
$ mkdir -p qsdk/dl
$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-ieee1905-security/* qsdk
$ cp -rf apss_proc/out/proprietary/Wifi/qsdk-qca-wifi/* qsdk
$ cp -rf apss_proc/out/proprietary/Wifi/qsdk-qca-wlan/* qsdk
$ cp -rf wlan_proc/src/components/qca-wifi-fw-src-component-cmn-WLAN.HK.* qsdk/dl/
$ cp -rf wlan_proc/pkg/wlan_proc/bin/QCA6018/qca-wifi-fw-QCA6018_v1.0-WLAN.HK.*
   qsdk/dl/
$ tar xvf cnss_proc/src/components/qca-wifi-fw-src-component-cmn-WLAN.BL.*.tgz -C
   qsdk/dl
$ tar xvf cnss_proc/src/components/qca-wifi-fw-src-component-halphy_tools-
   WLAN.BL.*.tgz -C qsdk/dl
$ cp -rf cnss_proc/src/components/* qsdk/dl
$ cp -rf cnss_proc/bin/QCA9888/hw.2/* qsdk/dl
$ cp -rf cnss_proc/bin/AR900B/hw.2/* qsdk/dl
$ cp -rf cnss_proc/bin/QCA9984/hw.1/* qsdk/dl
$ cp -rf cnss_proc/bin/IPQ4019/hw.1/* qsdk/dl
$ cp -rf qca-wifi-fw-AR988* qsdk/dl
$ cp -rf apss_proc/out/proprietary/QSDK-Base/meta-tools/ qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/common-tools/* qsdk/
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qsdk-qca-nss/* qsdk/
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-lib/* qsdk/
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-mcs-apps/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-nss-userspace/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-time-services/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-qmi-framework/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/gpio-debug/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-diag/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/athtestcmd/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/btdaemon/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/minidump/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-cnss-daemon/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/fw-qca-stats/* qsdk
$ cp apss_proc/out/proprietary/QSDK-Base/qca-nss-fw-eip-cp/BIN-EIP197.CP.*
   qsdk/dl/
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-rsrcmgr-bin/* qsdk
$ sed -i '/QCAHKSWPL_SILICONZ/c\PKG_VERSION:=WLAN.HK.2.9.r3-00031-
   QCAHKSWPL_SILICONZ-1' qsdk/qca/feeds/qca-cp/net/qca-cyp/Makefile
```

Premium/ Enterprise/ LM512

```
$ cp -rf apss_proc/out/proprietary/BLUETOPIA/qca-bluetopia/* qsdk
$ sed -i '0,/PKG_VERSION:=4.2.1.c1_26/s//PKG_VERSION:=4.2.1.4-00010/'
  qsdk/qca/feeds/bluetopia/bluetopia/Makefile
```

```
$ cp nss_proc/out/proprietary/* qsdk/dl
$ rm -rfv qsdk/qca/feeds/qca/utils/ctrl_app_dut
$ cp -rf apss_proc/out/proprietary/RBIN-AFCAgent/qca-afc-bin/* qsdk
$ cd qsdk/qca/feeds/afc
$ find -maxdepth 1 ! -name qti-mfg-provision ! -name qti-encrypt ! -name . -exec
   rm -rv {} \;
```

32-bit:

```
$ cd <Chipcode Directory>
$ mkdir -p qsdk/staging_dir/target-arm/usr/lib/
```

Premium 32-bit

```
$ cd qsdk/prebuilt/ipq60xx_32/ipq_premium
```

Enterprise 32-bit	`$ cd qsdk/prebuilt/ipq60xx_32/ipq_enterprise`
LM512 32-bit	`$ cd qsdk/prebuilt/ipq60xx_32/ipq_512`
LM256 32-bit	`$ cd qsdk/prebuilt/ipq60xx_32/ipq_256`

```
$ cp -rf ./libprovision.so ../../../staging_dir/target-arm/usr/lib/
$ cd <Chipcode Directory>
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

```
$ mkdir -p qsdk/staging_dir/target-arm/pkginfo/
$ touch qsdk/staging_dir/target-arm/pkginfo/qti-mfg-provision.provides
$ echo libprovision.so > qsdk/staging_dir/target-arm/pkginfo/qti-mfg-
  provision.provides
```

64-bit:

```
$ cd <Chipcode Directory>
$ mkdir -p qsdk/staging_dir/target-aarch64/usr/lib/
```

Premium 64-bit	`$ cd qsdk/prebuilt/ipq60xx_generic/ipq_premium`
Enterprise 64-bit	`$ cd qsdk/prebuilt/ipq60xx_generic/ipq_enterprise`

```
$ cp -rf ./libprovision.so ../../../staging_dir/target-aarch64/usr/lib/
$ cd <Chipcode Directory>
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

```
$ mkdir -p qsdk/staging_dir/target-aarch64/pkginfo/
$ touch qsdk/staging_dir/target-aarch64/pkginfo/qti-mfg-provision.provides
$ echo libprovision.so > qsdk/staging_dir/target-aarch64/pkginfo/qti-mfg-
  provision.provides
```

NOTE: Prebuilt ipk images (such as Qualcomm Bluetopia or qca-mcs-apps, qca-hyd-init, qca-hyd-son, qca-hyd-map) are in their respective folders:

`<ChipCode Directory>/qsdk/prebuilt/ipq60xx_64/ipq_<profile_name>/`	Contains all 64-bit ipk images
`<ChipCode Directory>/qsdk/prebuilt/ipq60xx/ipq_<profile_name>/`	Contains all 32-bit ipk images

Where ipq_<profile_name> is ipq_premium, ipq_enterprise, ipq_256, and ipq_512.

4. Customers with Qualcomm EZMESH, HY-FI, WHC, WAPid, or Sigma DUT packages:

	These files are fetched from ChipCode and copied to the working QSDK top-level directory:		
Premium/LM	**HY-FI and WHC:**	Hyfi qsdk-whc qsdk-whcpy	apss_proc\out\proprietary\Hyfi apss_proc\out\proprietary\Hyfi\qsdk-whc apss_proc\out\proprietary\Hyfi\qsdk-whcpy
Premium	**WAPid:**	qsdk-wapid	apss_proc\out\proprietary\Wapid\qsdk-wapid
Premium/ Enterprise	**Sigma-DUT:**	sigma-dut.tar.bz2	apss_proc\out\proprietary\sigma-dut\sigma-dut.tar.bz2
Premium/LM	**EZMESH-SRC**	qsdk-ezmesh-src	apss_proc\out\proprietary\RSRC-EZMESH\qsdk-ezmesh-src
Premium/LM	**EZMESH-BIN**	qsdk-ezmesh-bin	apss_proc\out\proprietary\RBIN-EZMESH\qsdk-ezmesh-bin
Premium/LM	**EZMESH-ALG**	qsdk-ezmesh-alg-bin	apss_proc\out\proprietary\RBIN-EZMESH-ALG\qsdk-ezmesh-alg-bin
Premium/LM	**EZ-ALG-SRC**	qsdk-ezmesh-alg-src	apss_proc\out\proprietary\RSRC-EZMESH-ALG\qsdk-ezmesh-alg-src

Then run the additional code:

Premium/LM	**HY-FI and WHC**	`$ cp -rf apss_proc/out/proprietary/Hyfi/hyfi/* qsdk`
		`$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-whc/* qsdk`
		`$ mkdir qsdk/qca/feeds/qca-son-mem-debug/qca-son-mem-debug`
		`$ mv qsdk/qca/feeds/qca-son-mem-debug/Makefile qsdk/qca/feeds/qca-son-mem-debug/Config.in qsdk/qca/feeds/qca-son-mem-debug/qca-son-mem-debug`
		`$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-whcpy/* qsdk`
		`$ sed -i "s/@PACKAGE_whc-son/@PACKAGE_whc-map/g" qsdk/qca/feeds/qca-lib/qca-wifison-ext-lib/Makefile`

Premium	**WAPid**	`git clone <wapid ChipCode directory>`
		`$ cd <wapid ChipCode directory>`
		`$ git checkout r12.2.r3_00009.0`
		`$ cd ..`
		`$ cp -rf <wapid ChipCode directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/Wapid/qsdk-wapid/* qsdk`
		Where WAPid ChipCode Directory is qca-networking-2022-spf-12-2_qca_oem_wapid-src for this release.

Premium/ Enterprise	**Sigma-DUT**	`git clone <sigma-dut ChipCode Directory>`
		`$ cd <sigma-dut ChipCode Directory>`
		`$ git checkout r12.2.r3_00009.0`
		`$ cd ..`
		`$ tar xjvf <sigma-dut ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/sigma-dut/sigma-dut.tar.bz2 -C qsdk`
		Where sigma-dut ChipCode Directory is qca-networking-2022-spf-12-2_qca_oem_sigma-dut for this release.

Premium/LM	**EZMESH-SRC**	EZMESH customers only:
		`$ git clone <ezmesh ChipCode Directory>`
		`$ cd <ezmesh ChipCode Directory>`
		`$ git checkout r12.2.r3_00009.0`
		`$ cd ..`
		`$ cp -rf <ezmesh ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RSRC-EZMESH/qsdk-ezmesh-src/* qsdk`
		Where EZMesh ChipCode Directory is qca-networking-2022-spf-12-2_qca_oem_ezmesh-src for this release
		`#Skip these sed commands in EZMESH-SRC if also using **EZ-ALG-SRC** (that is, both ezme src and ezmesh-alg-src)`
		`$ sed -i 's/HYD_MODULE_STRATEGY=y/HYD_MODULE_STRATEGY=n/g' qsdk/qca/src/qca-ezmesh/ezmeshConfig.defs`
		`$ sed -i '0,/ifeq/{/ifeq/d;}' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile`
		`$ sed -i '0,/endif/{/endif/d;}' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile`
		`$ sed -i '/libezmeshalg \\/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile`
		`$ sed -i '/libezmeshagentalg \\/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile`
		`$ sed -i '/DUMP/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile`

Premium/LM	**EZMESH-BIN**	EZMESH customers only:
		`$ git clone <ezmesh ChipCode Directory>`
		`$ cd <ezmesh ChipCode Directory>`
		`$ git checkout r12.2.r3_00009.0`
		`$ cd ..`
		`$ cp -rf <ezmesh ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RBIN-EZMESH/qsdk-ezmesh-bin/* qsdk`
		Where ezmesh ChipCode Directory is qca-networking-2022-spf-12-2_qca_oem_ezmesh-bin for this release

Premium/LM	**EZMESH-ALG**	EZMESH customers only:
		$ git clone <*ezmesh ChipCode Directory*>
		$ cd < *ezmesh ChipCode Directory*>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <*ezmesh ChipCode Directory*>/NHSS.QSDK.12.2/apss_proc/out/proprietary/ RBIN-EZMESH-ALG/qsdk-ezmesh-alg-bin/* qsdk
		Where ezmesh ChipCode Directory is qca-networking-2022-spf-12-2_qca_oem_ezmesh-alg for this release

Premium/LM	**EZ-ALG-SRC**	EZMESH customers only:
		$ git clone <*ezmesh ChipCode Directory*>
		$ cd < *ezmesh ChipCode Directory*>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <*ezmesh ChipCode Directory*>/NHSS.QSDK.12.2/apss_proc/out/proprietary/ RSRC-EZMESH-ALG/qsdk-ezmesh-alg-src/* qsdk
		Where ezmesh ChipCode Directory is qca-networking-2022-spf-12-2_qca_oem_ez-alg-src for this release

CTL_APP_SRC	WFA customers only:
	$ git clone <ctl_app_src ChipCode Directory>
	$ cd < ctl_app_src ChipCode Directory>
	$ git checkout r12.2.r3_00009.0
	$ cd ..
	$ cp -rf < ctl_app_src ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-SRC-CTRL-APP-DUT/qca-ctrl-app-dut/* qsdk
	Where ctl_app_src *ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ctlapp-src** for this release.

CTL_APP_BIN	WFA customers only:
	$ git clone <ctl_app_bin ChipCode Directory>
	$ cd < ctl_app_bin ChipCode Directory>
	$ git checkout r12.2.r3_00009.0
	$ cd ..
	$ cp -rf < ctl_app_bin ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-CTRL-APP-DUT/* <destination directory>
	Where ctl_app_bin *ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ctlapp-bin** for this release.

5.4.3.3 Create the 32-bit and 64-bit QSDK build

QSDK supports the **premium, enterprise, LM512, and LM256** profiles for Linux kernel 5.4.213 support**.** The QSDK framework is developed using Ubuntu (from version 16.04 to version 22.04) and Debian. However, QSDK framework regenerates critical tools required to compile firmware at build time. Thus, the framework is independent from the host environment. Although it is developed using the listed distributions, it is expected to work on others such as Red Hat, Mint, or Fedora.

See Section 2.5.3.2.1, to install the pre-requisite tools manually (or)

See Section 2.5.3.2.2, to use the script to get the pre-requisites installed in one go.

Because framework automatically downloads the open-source components needed during the build/ make process, ensure that an Internet connection is active on the build host when creating the build.

1. Install the different feeds in the build framework:

```
$ cd qsdk
$ ./scripts/feeds update -a
$ ./scripts/feeds install -a -f
```

2. Copy the base configuration to use for the build. Choose a profile and use it to build with Linux 5.4.213 support.
3. Regenerate a complete configuration file and start the build:

Premium 32-bit

```
$ cp qca/configs/qsdk/ipq_premium.config .config
$ sed -i 's/TARGET_ipq807x_generic/TARGET_ipq60xx_ipq60xx_32/g' .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq60xx/g' .config
$ mv prebuilt/ipq60xx_32/ipq_premium/* prebuilt/ipq60xx_32
```

Premium 64-bit

```
$ cp qca/configs/qsdk/ipq_premium.config .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq60xx/g' .config
$ mkdir -p prebuilt/generic
$ cp -rf prebuilt/ipq60xx_generic/ipq_premium/* prebuilt/generic
```

Enterprise 32-bit

```
$ cp qca/configs/qsdk/ipq_enterprise.config .config
$ sed -i 's/TARGET_ipq807x_generic/TARGET_ipq60xx_ipq60xx_32/g' .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq60xx/g' .config
$ mv prebuilt/ipq60xx_32/ipq_enterprise/* prebuilt/ipq60xx_32
```

Enterprise 64-bit

```
$ cp qca/configs/qsdk/ipq_enterprise.config .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq60xx/g' .config
$ mkdir -p prebuilt/generic
$ cp -rf prebuilt/ipq60xx_generic/ipq_enterprise/* prebuilt/generic
```

LM512 32-bit

```
$ cp qca/configs/qsdk/ipq_512.config .config
$ sed -i "s/TARGET_ipq807x_generic/TARGET_ipq60xx_ipq60xx_32/g" .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq60xx/g' .config
$ mv prebuilt/ipq60xx_32/ipq_512/* prebuilt/ipq60xx_32
```

LM256 32-bit

```
$ cp qca/configs/qsdk/ipq_256.config .config
$ sed -i "s/TARGET_ipq807x_generic/TARGET_ipq60xx_ipq60xx_32/g" .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq60xx/g' .config
$ mv prebuilt/ipq60xx_32/ipq_256/* prebuilt/ipq60xx_32
```

For Hy-Fi, WHC customers:

- Starting with QCA_Networking_2020.SPF.11.3 release, SON and EasyMesh Features are compiled as separate build. SON build supports only SON features and EZMESH build supports only EasyMesh features.
- Starting with QCA_Networking_2021.SPF.11.4 release, MAP package qca-hyd-map is replaced with qca-ezmesh. Enable additional configuration to enable EZMESH packages.
- Use the SON package commands in this table for a SON features enabled build.
  Use the EZMESH package commands for an EasyMesh features enabled build.

4. Enable EZMESH configuration:

EZMESH package (Full Mode)

```
$ echo "CONFIG_PACKAGE_whc-mesh=n" >> .config
$ echo "CONFIG_PACKAGE_hyfi-mesh=n" >> .config
$ echo "CONFIG_PACKAGE_whc-map=y" >> .config
$ echo "CONFIG_PACKAGE_hyfi-map=y" >> .config
Additional configuration to enable EZMESH packages:
$ echo "CONFIG_PACKAGE_qca-ezmesh-cmn=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-ctrl=n" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-agent=n" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-agentalg=n" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg-static=n" >> .config
```

EZMESH package (Co-located mode)

```
$ echo "CONFIG_PACKAGE_whc-mesh=n" >> .config
$ echo "CONFIG_PACKAGE_hyfi-mesh=n" >> .config
$ echo "CONFIG_PACKAGE_whc-map=y" >> .config
$ echo "CONFIG_PACKAGE_hyfi-map=y" >> .config
Additional configuration to enable EZMESH packages:
$ echo "CONFIG_PACKAGE_qca-ezmesh-cmn=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-ctrl=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-agent=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-agentalg=y" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh=n" >> .config
$ echo "CONFIG_PACKAGE_qca-ezmesh-alg-static=n" >> .config
```

NOTE: Enabling both SON and EasyMesh Features in a build will increase image size and consume extra memory in Flash storage.

```
$ echo CONFIG_BUILD_SHORTENED_PATH=y >> .config
$ make defconfig
$ make V=s -j5
```

5. For 64-bit firmware, once the 32-bit compilation completes, save all the U-boot executable files **OpenWRT-ipq6018*.elf** from the directory: **qsdk/bin/ipq.** These files are required to generate all 14 variants of the complete 64-bit firmware image.
6. Download the required packages for the corresponding profile and create the image. Once the build is complete, these files are in the **qsdk/bin/ipq** directory:

32-bit	▪ OpenWRT-ipq6018*.elf (ELF files for U-Boot variants) ▪ OpenWRT-ipq-ipq60xx-squashfs-root.img (SquashFS) ▪ OpenWRT-ipq-ipq60xx-qcom-ipq60xx-cpxx-fit-uImage.itb (ITB)
64-bit	▪ OpenWRT-ipq-ipq60xx_64-squashfs-root.img (SquashFS) ▪ OpenWRT-ipq-ipq60xx_64-qcom-ipq60xx-cpxx-fit-uImage.itb (ITB)

NOTE: The OpenWRT-ipq-ipq60xx-ubi-root.img generated in qsdk/bin/ipq folder **will no longer be used** for IPQ6018 because the Wi-Fi firmware image must be added as a ubi volume to the UBI image.

5.4.4 Generate a complete firmware image

IPQ60x8 requires flashing multiple images for Bootup, including xBL, RPM, TZ, CDT, MIBIB, NSS Images, Kernel, Filesystem, and so on. To simplify both image flashing and device boot, individual images are combined into a single Flattened Image Tree (FIT) image. FIT image components can be flashed into the respective partition based on user configuration. More tools required on the Ubuntu 16.04 to 22.04 64-bit machine include:

1. Install DTC:
   ```
   $sudo apt-get install device-tree-compiler
   ```
2. Install libfdt:
   ```
   $sudo apt-get install libfdt-dev
   ```
3. Install Python 2.7.
4. Install tools required to run packing single image:
   ```
   $sudo apt-get install libc6-i386
   $sudo apt-get install libgl1-mesa-dri:i386
   ```
5. Switch to the Qualcomm ChipCode Portal directory:
   ```
   $ cd <Chipcode Directory>
   ```
6. Copy the flash config files to **common/build/ipq**

7. Find the single images in the **common/build/bin** directory. Images are generated for default image content. Fewer images indicate an issue with one of the partition sizes, or that not all required files are present.

32-bit image

```
$ cd <Chipcode Directory>
$ mkdir -p common/build/ipq
$ mkdir -p apss_proc/out/meta-scripts
$ cp qsdk/qca/src/u-boot-2016/tools/pack.py apss_proc/out/meta-scripts/pack_hk.py
$ sed -i 's#</linux_root_path>#/</linux_root_path>#' contents.xml
$ sed -i 's#</windows_root_path>#\\</windows_root_path>#' contents.xml
$ cp qsdk/bin/targets/ipq60xx/ipq60xx_32/openwrt* common/build/ipq
$ cp -r apss_proc/out/proprietary/QSDK-Base/meta-tools apss_proc/out/
$ cp -rf qsdk/bin/targets/ipq60xx/ipq60xx_32/dtbs/* common/build/ipq/
$ cp -rf skales/* common/build/ipq/
$ cp -rf wlan_proc/build/ms/bin/FW_IMAGES/* common/build/ipq/
$ cp qsdk/staging_dir/host/bin/ubinize common/build/ipq/
$ cd common/build
$ sed
 '/debug/d;/packages/d;/"ipq6018_64"/d;/t32/d;/ret_prep_64image/d;/Required/d;/sk
 ales/d;
 /nosmmu/d;/os.system(cmd)/d;/os.chdir(ipq_dir)/d;/atfdir/d;/noac/d;/single-
 atf/d;/bl31.mbn/d;/ret_pack_64image/d;/ret_pack_apps_64image/d;/64image/d' -i
 update_common_info.py
$ export BLD_ENV_BUILD_ID=<profile-name>
$ python update_common_info.py
```

(Where <profile-name> is E, P, LM256, or LM512)

64-bit image

```
$ cd <Chipcode Directory>
$ mkdir -p common/build/ipq_x64
$ mkdir -p apss_proc/out/meta-scripts
$ cp qsdk/qca/src/u-boot-2016/tools/pack.py apss_proc/out/meta-scripts/pack_hk.py
$ sed -i 's#</linux_root_path>#/</linux_root_path>#' contents.xml
$ sed -i 's#</windows_root_path>#\\</windows_root_path>#' contents.xml
$ cp qsdk/bin/targets/ipq60xx/generic/openwrt* common/build/ipq_x64
copy the saved openwrt-ipq6018-u-boot*.elf from the 32-bit build to:
   common/build/ipq_x64
$ cp -r apss_proc/out/proprietary/QSDK-Base/meta-tools apss_proc/out/
$ cp -rf qsdk/bin/targets/ipq60xx/generic/dtbs/* common/build/ipq_x64/
$ cp -rf skales/* common/build/ipq_x64/
$ cp -rf wlan_proc/build/ms/bin/FW_IMAGES/* common/build/ipq_x64
$ cp qsdk/staging_dir/host/bin/ubinize common/build/ipq_x64
$ cd common/build
$ sed -i 's/\.\/ipq/\.\/ipq_x64/g' update_common_info.py
$ sed -i 's/\.\/ipq_x64_x64/\.\/ipq_x64/g' update_common_info.py
$ sed
   '/debug/d;/packages/d;/"ipq6018"/d;/t32/d;/ret_prep_32image/d;/Required/d;/nos
   mmu/d
   ;/os.system(cmd)/d;/skales/d;/os.chdir(ipq_dir)/d;/atfdir/d;/noac/d;/single-
   atf/d;/bl31.mbn/d;/ret_pack_32image/d;/ret_pack_apps_32image/d;/32image/d' -i
   update_common_info.py
$ export BLD_ENV_BUILD_ID=<profile-name>
$ python update_common_info.py
```

(Where <profile-name> is E or P)

5.4.5 Flash the complete default software image

5.4.5.1 Set up the flash environment

1. Ensure that the board console port is connected to the PC using these RS232 parameters:
   - 115200 bps
   - 8N1
2. Confirm that the PC is connected to the board using one of the Ethernet ports. The PC must have a TFTP server launched and listening on the interface to which the board is connected. The interface must have an IP address configured manually. At this stage powerup the board and after a few seconds, press any key during the countdown.

5.4.5.2 Standard board configuration

Load the image in flash and boot the platform:

1. Copy the intended **xxxx-ipq6018-single.img** (or 64-bit equivalent file) to the TFTP server root directory.
2. Check hardware jumper configuration according to reference board and default memory configuration (see the appropriate board setup guide for more information) to verify which flash memory the board is booting from.
3. Confirm the machine ID, Meta version, profile, and selected single image to match the memory type that the board boots from.
4. Set the IP address and server IP using the TFTP process:

```
setenv ipaddr 192.168.1.11
setenv serverip 192.168.1.xx  (TFTP server address)
```


For IPQ6018+QCN90xx

```
ping ${serverip}
set bootargs console=ttyMSM0,115200n8
tftpboot xxxx-ipq6018-single.img
```

1. Once the single image has booted, transfer SON packages for SON features or MAP packages for EasyMesh features to the AP /tmp directory from **./qsdk/prebuilt/ipq60xx/** or **./qsdk/prebuilt/ ipq60xx _64/**, depending on whether the booted image is 32-bit or 64-bit.

```
set bootargs console=ttyMSM0,115200n8
tftpboot xxxx-ipq6018-single.img
```

2. Flash the image:

```
imgaddr=$fileaddr && source $imgaddr:script
```

3. Power cycle the board after step 5 has completed, as indicated, by the printing of the U-boot prompt. It may be hundreds of seconds depending on image size and memory technology.

NOTE: For Bluetopia, perform Steps 7 and 8.

4. Once the single image has booted, transfer these packages to the AP /tmp directory from ./**qsdk/prebuilt/ipq60xx/** or ./**qsdk/prebuilt/ ipq60xx _64/**, depending on whether the booted image is 32-bit or 64-bit.

```
qsdk/prebuilt/ipq60xx/bluetopia_4.2.1.c1_26-1_arm_cortex-a7_neon-vfpv4.ipk
qsdk/prebuilt/ipq60xx_64/bluetopia_4.2.1.c1_26-1_aarch64_cortex-a53_neon-
 vfpv4.ipk
```

5. Install these packages from the /tmp directory:
   `opkg install bluetopia_4.2.1.c1_26-1_arm_cortex-a7_neon-vfpv4.ipk For Wi-Fi SON, perform steps 10, 11, and 12:`
6. Once the single image has booted, transfer SON packages for SON Features or EZMESH Packages for EasyMesh Features to the AP /tmp directory from ./**qsdk/prebuilt/ipq60xx/** or ./**qsdk/prebuilt/ ipq60xx _64/**, depending on whether the booted image is 32-bit or 64-bit.

SON package (32-bit)	qsdk/prebuilt/ipq60xx/qca-hyd-init_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq60xx/qca-hyd-son_ge474d2e-1_ipq.ipk
SON package (64-bit)	qsdk/prebuilt/ipq60xx_64/qca-hyd-init_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq60xx_64/qca-hyd-son_ge474d2e-1_ipq.ipk
EZMESH package (32-bit)	**To support Full mode:** qsdk/prebuilt/ipq60xx/qca-ezmesh-cmn_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq60xx/qca-ezmesh_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq60xx/qca-ezmesh-alg_ge474d2e-1_ipq.ipk **To support Co-located mode:** qsdk/prebuilt/ipq60xx/qca-ezmesh-cmn_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq60xx/qca-ezmesh-ctrl_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq60xx/qca-ezmesh-agent_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq60xx/qca-ezmesh-alg_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq60xx/qca-ezmesh-agentalg_ge474d2e-1_ipq.ipk
EZMESH package (64-bit)	**To support Full mode:** qsdk/prebuilt/ipq60xx_64/qca-ezmesh-cmn_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq60xx_64/qca-ezmesh_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq60xx_64/qca-ezmesh-alg_ge474d2e-1_ipq.ipk **To support Co-located mode:** qsdk/prebuilt/ipq60xx_64/qca-ezmesh-cmn_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq60xx_64/qca-ezmesh-ctrl_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq60xx_64/qca-ezmesh-agent_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq60xx_64/qca-ezmesh-alg_ge474d2e-1_ipq.ipk qsdk/prebuilt/ipq60xx_64/qca-ezmesh-agentalg_ge474d2e-1_ipq.ipk

NOTE: ge474d2e is the git commit ID, which may also be changed.

7. Install these packages from the /tmp directory: Choose SON package to enable SON features, or choose MAP package to enable MAP features:

SON package	opkg install qca-hyd-init_ge474d2e-1_ipq.ipk opkg install qca-hyd-son_ge474d2e-1_ipq.ipk
EZMESH package	**To support Full mode:** opkg install qca-ezmesh-cmn_ge474d2e-1_ipq.ipk opkg install qca-ezmesh_ge474d2e-1_ipq.ipk opkg install qca-ezmesh-alg_ge474d2e-1_ipq.ipk **To support Co-located mode:** opkg install qca-ezmesh-cmn_ge474d2e-1_ipq.ipk opkg install qca-ezmesh-ctrl_ge474d2e-1_ipq.ipk opkg install qca-ezmesh-agent_ge474d2e-1_ipq.ipk opkg install qca-ezmesh-alg_ge474d2e-1_ipq.ipk opkg install qca-ezmesh-agentalg_ge474d2e-1_ipq.ipk

NOTE: ge474d2e is the git commit ID, which may also be changed.

The HYD daemon application binary name is changed from hyd to hyd-son for the SON package. Similarly, the WSPLCD daemon application binary name is changed from wsplcd to wsplcd-son and wsplcd-map for the SON and EZMESH packages, respectively. The symbolic link to the older binary name must be create after package installation to ensure that scripts and tools using work with the new binary name. Because application binary name has changed, a crash dump is created

based on new binary name. Customers must take care of updating any scripts and tools that use or depend on the older binary name.

Execute these commands to create a symbolic softlink to the older binary name:

SON package	`ln -s /usr/sbin/wsplcd-son /usr/sbin/wsplcd` `ln -s /usr/sbin/hyd-son /usr/sbin/hyd`
EZMESH package	`ln -s /usr/sbin/wsplcd-map /usr/sbin/wsplcd`

5.4.5.3 Upgrade the firmware

This release can upgrade board images without a TFTP server. After using U-Boot to flash an initial image and booting the device, use the web interface for future upgrades. These future upgrades can take place from either the OpenWRT web interface or serial console. Upgrade the existing flash image using either the appropriate single image file or the apps image file generated by **update_common_info.py**.

Upgrading image files via the web interface takes several minutes to complete depending on factors such as memory technology, vendor, image size, browser connectivity, and network load. Completion of the flash upgrade process is signaled by the refresh of the OpenWRT login page, or lack of further messages on the serial console.

- An invalid image remains in the flash memory if the upgrade process is interrupted, such as if system power is lost during upgrade or a keypress event is detected on the serial console port.
- If a board is configured for failsafe boot, and the U-boot environment variable *bootargs* contains any reference to the rootfs or other partitions managed by the failsafe scheme. In this case, the sysupgrade process may only be partially successful: the board may or may not boot later, and the image may only be partly updated.
- To use the **sysupgrade** command from the Linux console, the sysupgrade image must be visible:
  - Within the Linux file system supplied using development host ssh or tftpd server, OR
  - On an NFS server that is mounted to the local file system, OR
  - Transferred using a USB storage device
- The sysupgrade feature can only be used to update the image in the memory the board boots from and is used by the running kernel. If the board supports multiple memory technologies, use U-Boot to change the image in memories that the board did not boot from.

See the *IPQ60x8 SoC Software User Guide* (80-YC609-8) for more information on sysupgrade support.

5.4.6 Create customized IPQ60x8 Wi-Fi firmware images

Flash memory layout includes a UBI volume for Wi-Fi firmware and BDF information. The release includes squashfs images in the final image created in the final steps.

Use these steps to update or recreate the squashfs Wi-Fi firmware image and the complete final image.

NOTE: If BDF and PIL files are not customized, there is no need to update or recreate the Wi-Fi firmware images.

1. Install these tools in the workstation:
   - mtd-utils: sudo apt-get install mtd-utils
   - mksquashfs4 binary from **<Chipcode Directory> /common/build** to /usr/bin or /usr/sbin or any location included in $PATH

2. Download the WLAN IPQ6018 firmware BIN tar ball from the corresponding SP release distro.
3. Extract the firmware binary tarball:
```
mkdir -p <Chipcode Directory>/fwtemp
cp -rfv <Chipcode Directory>/wlan_proc/pkg/wlan_proc/bin/QCA6018/qca-wifi-fw-
  QCA6018_v1.0-WLAN.HK.*.tar.bz2 fwtemp
cd fwtemp
tar -xvf qca-wifi-fw-QCA6018_v1.0-WLAN.HK.*.tar.bz2
```
4. Create a staging directory to hold the BD and PIL files.
```
mkdir staging_dir
cp -rfv qca-wifi-fw-WLAN.HK.*/PIL_IMAGES/* staging_dir
cp -rfv qca-wifi-fw-WLAN.HK.*/bdwlan* staging_dir
```
(Replace the BDF file with the custom versions as needed.)
5. To generate the firmware squash image for NAND/NOR/EMMC (this example assumes mksquashfs4 is in $PATH; use the exact installed path of mksquashfs4):
```
mksquashfs4 staging_dir/ wifi_fw.squashfs -nopad -noappend -root-owned -comp
  xz -Xpreset 9 -Xe -Xlc 0 -Xlp 2 -Xpb 2 -Xbcj arm -b 256k -processors 1
dd if=wifi_fw.squashfs of=wifi_fw_squashfs.img bs=2k conv=sync
```
6. The UBI firmware image for NAND is no longer required as Wi-Fi image will be added to the UBI volume and will be ubinized along with Kernel and rootfs

For instructions on how to the generate ubinized image and flash the firmware images separately, see the *IPQ6018 SoC Software User Guide* (80-YC609-8).

5.4.7 Create customized IPQ60x8 + QCN90xx Wi-Fi firmware images

Flash memory layout includes an UBI volume for Wi-Fi firmware and BDF information. The release includes squashfs images in the final image created in the final steps.

Use the following steps to update or recreate the squashfs Wi-Fi firmware image and the complete final image.

NOTE: If BDF and PIL files are not customized, there is no need to update or recreate the Wi-Fi firmware images.

1. Install these tools in the workstation:
   - mtd-utils: sudo apt-get install mtd-utils
   - mksquashfs4 binary from:
     <ChipCode Directory> /common/build to /usr/bin or /usr/sbin or any location included in $PATH
2. Download the WLAN SquashFS image wifi_fw_ipq6018_qcn9000_squashfs.img from WLAN.HK.2.3/wlan_proc/build/ms/bin/FW_IMAGES/
3. Extract the squashfs image in a temp directory.
```
mkdir -p fwtemp
cp <download path>/wifi_fw_ipq6018_qcn9000_squashfs.img  ./fwtemp
cd fwtemp
unsquashfs wifi_fw_ipq6018_qcn9000_squashfs.img
```
4. All extracted files are in the folder: ./squashfs-root
5. Copy/modify any files in the file: ./squashfs-root

6. To generate the firmware SquashFS image for NAND/NOR/EMMC (this example assumes mksquashfs4 is in $PATH; use the exact installed path of mksquashfs4):
```
mksquashfs4 squashfs-root/ wifi_fw.squashfs -nopad -noappend -root-owned
 -comp xz -Xpreset 9 -Xe -Xlc 0 -Xlp 2 -Xpb 2 -Xbcj arm -b 256k -
 processors 1
dd if=wifi_fw.squashfs of=wifi_fw_squashfs.img bs=2k conv=sync
```
7. The UBI firmware image for NAND is no longer required as the Wi-Fi image is added to the UBI volume and will be ubinized along with Kernel and rootfs.

For instructions on how to the generate ubinized image and flash the firmware images separately, see the *IPQ6018 SoC Software User Guide* (80-YC609-8).

5.5 Flash Wi-Fi firmware image only

The Wi Fi firmware/BDF information stored in the flash image can be updated without rewriting the whole flash image as shown in this section. Regardless of the memory type the system boots from, follow the same basic process of using tftpboot to load the new Wi-Fi firmware image to DDR, then identify the correct location in the flash and write the Wi-Fi firmware image to that location.

NAND boot

- Copy the image (wifi_fw_squashsfs.img):
```
# tftpboot wifi_fw_squashsfs.img
```
- For IPQ6018+QCN90xx
```
# tftpb wifi_fw_ipq6018_qcn9000_squashfs.img
```
- Flash the partition using the flash command:
```
# flash wifi_fw
```

Where wifi_fw is the UBI volume name and should be used in the flash command.

NOR boot / eMMC boot

- Copy the image (wifi_fw_squashsfs.img):
```
# tftpboot wifi_fw_squashsfs.img
```
- For IPQ6018+QCN90xx
```
# tftpb wifi_fw_ipq6018_qcn9000_squashfs.img
```
- Flash the partition using the flash command:
```
# flash 0:WIFIFW
```

Where 0:WIFIFW is the Wi-Fi partition name. This name is used for flash command.

UBI partition changes

Separate partition for Wi-Fi firmware image is removed from the NAND flash layout and the Wi-Fi firmware image is added as a UBI volume, along with Kernel and rootfs, to save flash space. This feature will be supported in this release. See *IPQ6018 SoC Software User Guide* (80-YC609-8) for more details.

5.6 Test the GATT profile with sample applications with onboard CSR8811 Bluetooth on AP.CP01

This release contains a preloaded Bluetooth sample application. To test GATT profile containing sample applications:

1. Launch the Bluetooth sample application from the AP after the Premium profile is built. It is available at:
   apss_proc/out/proprietary/BLUETOPIA/qca-bluetopia/prebuilt
2. After the complete firmware image is built, launch the Bluetooth stack to test the GATT profile with these LinuxSPPLE sample applications:
   - The LinuxSPPLE application is intended to demonstrate the usage of the Qualcomm Bluetopia Serial Port Profile Low Energy API and relevant Bluetopia Core APIs. The application supports issuing all the basic commands used by the Serial Port Profile Low Energy.
   - Launching the BT sample APP, which enables BT and download the PSKeys to the BT module.
3. Wi-Fi/BT coex PSKeys are enabled after the sample APP is launched.
4. Export the environment variable before launching the sample application.

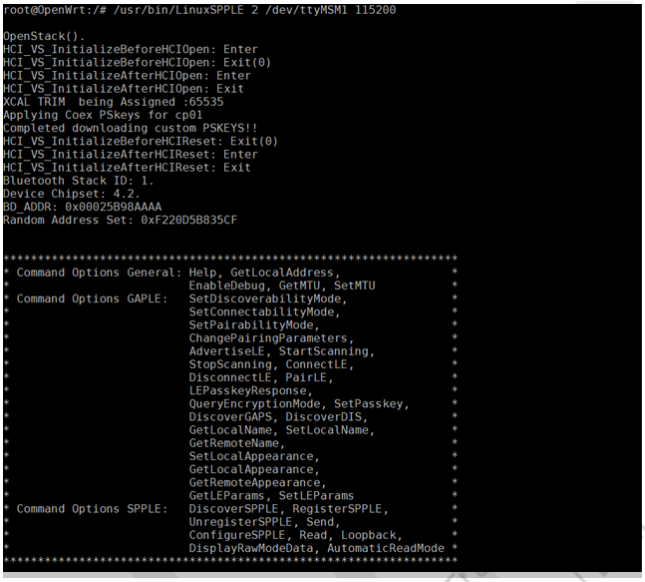
Platform/Module	Environment variable
IPQ6018 platform CP01 or QFN BT module	BTHOST_8311_SOC_TYPE=cp01
custom SOC GPIO settings	BTHOST_8311_SOC_TYPE=custom

5. PSKeys will be read from the text file located in **/usr/bin as PS_KEY_CSR8811.txt.** Contents of the text file must be in this format:

```
# PSKEY_COEX_SCHEME
\x06\xC2\x02\x00\x09\x00\x34\x00\x03\x70\x00\x00\x80\x24\x01\x00\x00\x00\x03\x00
# PSKEY_COEX_PIO_UNITY_3_BT_ACTIVE (BT_active)
\x05\xC2\x02\x00\x0A\x00\x35\x00\x03\x70\x00\x00\x83\x24\x02\x00\x00\x00\x05\x00\x
   01\x00
# PSKEY_COEX_PIO_UNITY_3_BT_STATUS (BT_STATUS)
\x05\xC2\x02\x00\x0A\x00\x36\x00\x03\x70\x00\x00\x84\x24\x02\x00\x00\x00\xff\xff\x
   01\x00
# PSKEY_COEX_PIO_UNITY_3_BT_DENY (WLAN Deny)
\x05\xC2\x02\x00\x0A\x00\x37\x00\x03\x70\x00\x00\x85\x24\x02\x00\x00\x00\x04\x00\x
   01\x00
# WARM RESET
\x00\xC2\x02\x00\x09\x00\x47\x00\x02\x40\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
```

root@OpenWRT:cd /usr/bin
root@OpenWRT: ./LinuxSPPLE 2 /dev/ttyMSM1 115200

6. Note the Bluetooth address of the module on the AP.
   - Bluetooth Stack ID: 1
   - Device Chipset: 4.2.
     BD_ADDR: 0x00025B98AAAA



7. Ignore the HCI_VS print statements; the Bluetooth stack application will launch with the prompt.
8. Run these commands to ensure that the radio is attached to the BT connector is good and can scan surrounding devices:
   - StartScanning – Starts scanning all Bluetooth devices with their device ID and Bluetooth address
   - StopScanning – Stops scanning devices. This command can be run during scanning.
9. Use this command to register a SPPLE service.

   ```
   SPPLE>RegisterSPPLE
   Successfully registered SPPLE Service.
   SPPLE>
   ```

   This function returns zero on successful execution and a negative value on errors. This command takes no parameters.
10. Use this command to turn on LE advertising on the device:

    ```
    AdvertiseLE 1
    GAP_LE_Advertising_Enable success.
    SPPLE>
    ```

    The only parameter required is the advertising flag which is 0 for disable and 1 for enable. This registers the device as SPPLE and clients can connect to this one.
    - Bluetooth clients
      - Bluetooth clients can use a custom App or Light Blue App on an AP.CP01 device with Bluetooth or with the CSR™ USB dongle 8510. LinuxSPPLE must show up in the scanning. Connect to a Bluetooth server with the Bluetooth address and send data.
      - Connecting using Bluetopia stack from another CP01 with On-board CSR 8811

      ```
      SPPLE>ConnectLE 00025B98AAAA 0.
      ```

11. This command performs a SPPLE service discovery operation. This function returns zero on successful execution and a negative value on errors. This command takes no parameters.

    Client side:

```
SPPLE>DiscoverSPPLE
GATT_Start_Service_Discovery_Handle_Range() success.
SPPLE>
Service 0x001E - 0x0028, UUID: 14839AC47D7E415C9A42167340CF2339.
SPPLE>
Service Discovery Operation Complete, Status 0x00.
Valid SPPLE Service Found.
```

12. This command configures a SPPLE service on a remote device. This function returns zero on successful execution and a negative value on errors. This command takes no parameters.

    This function enables notifications of the proper characteristics based on a specified handle. Depending on whether the device role for SPPLE is server or client, either a GATT_Handle_Value_Notification or a GATT_Write_Without_Response_Request API function is called. The called function notifies the receiving credit characteristic or sends a write without response packet to the transmission credit characteristic respectively.

```
SPPLE>ConfigureSPPLE
SPPLE Service found on remote device, attempting to read Transmit Credits, and
   configured CCCDs.

SPPLE>
Write Response.
Connection ID:   1.
```

HSUART node name

Board	Kernel version	Device node name
IPQ6018	5.4	/dev/ttyMSM1

6 IPQ5322.ILQ.12.2 CSU1

6.1 Supported hardware for this SP

This release supports these RDPs:

RDP	RDP ID	FB-HDK	PCB design	DDR	PCIe	Radio	BDF
RDP441	QCARDP441 MI01.2 Tri-band	DP25-57856-200	4 layers	1 GB DDR4 x 16	x1L x2L	2GHz 2x2 xFEM 5GHz 4x4 QCN9274 6GHz 4x4 QCN9274	B12 B0015 B0016
RDP468	QCARDP468 MI01.6 Tri-band	DP25-57856-600	4 layers	512MB DDR4 x 16	X2L	2GHz 2x2 iPA iLNA 5GHz 2x2 QCN9274 6GHz 2x2 QCN9274	B16 B1019 B1019
RDP474	QCARDP474 MI01.9 Dual-band	DP25-57856-900	4 layers	1 GB DDR4 x 16	x1L x2L	2GHz 4x4 QCN9274 5GHz 4x4 QCN9274	B001d B0015

6.2 Build and load the image for IPQ5322.ILQ.12.2 CSU1

1. Download the Qualcomm Technologies, Inc. (QTI) proprietary code from the Qualcomm ChipCode Portal (see Section 6.2.1).
2. Download other components from external websites for QSDK while building the default configuration (see Section 6.2.2).
3. Generate the firmware:
   a. Reassemble the code and generate the QSDK framework (see Section 6.2.3.1).
   b. Set up and create the QSDK build (see Section 6.2.3.3 and Section 6.2.3.4).
   c. Generate a complete firmware image (see Section 6.2.4).
4. Flash the software image (see Section 6.2.5).

Users should be familiar with directory structures that contain SP images for the different subsystems before downloading the code and building the images for loading. For each SP included in an SPF, SP binary files are generated from the SI binary files of only a subset of the included SIs. In an SPF, some SIs may support multiple SPs while others may only support one SP.

6.2.1 Download packages available through the Qualcomm ChipCode Portal

QTI proprietary code is available from ChipCode. A web/GUI interface and a secure git server both allow access to this code.

- Browse available packages and obtain the download URL at: https://chipcode.qti.qualcomm.com/

- For more information on cloning the code, see: https://chipcode.qti.qualcomm.com/helpki/cloning-code-from-a-repository
- For more information on installation and configuration of the correct version of git and OpenSSL on both Windows and Linux platforms, see: https://chipcode.qti.qualcomm.com/helpki

  These versions are required to support the authentication methods used by QTI ChipCode.

6.2.2 Download packages from external websites

For details on downloading packages from external sites, see Section 2.2.2.

6.2.3 Generate the IPQ firmware for IPQ5322.ILQ.12.2 CSU1 release

The firmware image can be generated either using script (see Section 6.2.3.1) or manually (see Section 2.1.3.2)

6.2.3.1 Script to generate complete firmware image

NOTE: The customer build script is applicable only if OpenWrt framework is used.

Meta generation script for IPQ5322 12.2 CSU1 (meta_generation_script.py) aims to simplify the release notes instructions required to be executed by the customers. Follow these steps.

```
$ mkdir $BUILD_WS
```

(Assume that BUILD_WS is the workspace directory)

```
$ cd $BUILD_WS
$ git clone <Chipcode Directory>
$ cd <Chipcode Directory>
$ git checkout r12.2.r3_00009.0
$ rm -rf BOOT.XF.0.3 CNSS.PS.3.19 IPQ6018.ILQ.12.2 TMEL.WNS.1.0 TZ.WNS.4.0
WLAN.BL.3.19 BOOT.BF.3.3.1 IPQ5018.ILQ.12.2 IPQ8074.ILQ.12.2 NSS.FW.12.2
TZ.WNS.5.1 WLAN.HK.2.9 BOOT.BF.3.3.1.1 BTFW.MAPLE.1.0.0 IPQ9574.ILQ.12.2
RPM.BF.2.4.1 TZ.BF.4.0.8
$ cp -rf */* .
$ cp -rf IPQ5322/trustzone_images/ .
```

ChipCode Directory for this release:	qca-networking-2022-spf-12-2_qca_oem

Where the clone required deliverable distro is in the same path where the OEM distro is cloned using git clone commands and checkout (git checkout) to the release tag on each deliverable distro.

```
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
 release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml
```

If this command does not work, use this repo init command instead:

```
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak -b
 release -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml --
 repo-url=https://git.codelinaro.org/clo/la/tools/repo.git --repo-branch=qc-
 stable --no-clone-bundle

$ repo sync -j8 --no-tags -qc
$ cd common/build
```

```
$ python meta_generation_script.py -c [Chipcode_tag] -s [SP] -p [Profile] -b
  [32/64] -d [Distro_list] -m [MESH]--path $BUILD_WS
```

For example:

```
python meta_generation_script.py -c r12.2.r3_00009.0 -p P -b 64 -d HYFI,WHC,WAPID,SIGMA-DUT\
,EZMESH_SRC,EZMESH_BIN,EZMESH_ALG,EZ_ALG_SRC,CTL_APP_SRC,CTL_APP_BIN,IFLI_SRC,YORK \
-m EZMESH_FULL --path /local/mnt2/workspace/
```

The IPQ5322.ILQ.12.2 CSU1 release supports these configurations:

Profile and bit config	Supported Deliverables Combo	Mesh options
P – 32,64	HYFI, WHC, WAPID, SIGMA-DUT, EZMESH_SRC, EZMESH_BIN, EZMESH_ALG, EZ_ALG_SRC, CTL_APP_SRC, CTL_APP_BIN,EZ_ALG_STA, YORK, IFLI_SRC, OEM	EZMESH_FULL, EZMESH_COLOCATED
LM512 – 32	HYFI, WHC, EZMESH_SRC, EZMESH_BIN, EZMESH_ALG, EZ_ALG_SRC, CTL_APP_SRC, CTL_APP_BIN, YORK, OEM	EZMESH_FULL, EZMESH_COLOCATED

NOTE: Distro Name: qca-networking-2022-spf-12-2_qca_oem_ifli-bin

IFLI-BIN customers to directly use BIN on the target (using TFTP).

NOTE: When no additional deliverables are there, use -d OEM while triggering meta_generation_script. 64-bit builds generate 32-bit images by default.

NOTE: Requisite for EZ-ALG-STA is EZMESH-SRC (ONLY). Please ensure no EZMESH options are provided.

NOTE: If no Mesh options are left, remove -m <Mesh option> while triggering meta_generation_script.

Single images are in these directories:

32-Bit	`$BUILD_WS/M/<Profile>Timestamp/<OEM distro>/common/build/bin` Where <Profile> is P, LM
64-Bit	`$BUILD_WS/M/<Profile>Timestamp/64/<OEM distro>/common/build/bin` Where <Profile> is P

6.2.3.2 Reassemble the code and generate the QSDK framework

This example assumes that all packages are obtained using the **git clone** command and placed in the top-level directory.

1. Reassemble the code and generate the QSDK framework:

```
$ git clone <ChipCode Directory>
$ cd <ChipCode Directory>
$ git checkout r12.2.r3_00009.0
$ rm -rf BOOT.XF.0.3 CNSS.PS.3.19 IPQ6018.ILQ.12.2 TMEL.WNS.1.0 TZ.WNS.4.0
  WLAN.BL.3.19 BOOT.BF.3.3.1 IPQ5018.ILQ.12.2 IPQ8074.ILQ.12.2 NSS.FW.12.2
  TZ.WNS.5.1 WLAN.HK.2.9 BOOT.BF.3.3.1.1 BTFW.MAPLE.1.0.0 IPQ9574.ILQ.12.2
  RPM.BF.2.4.1 TZ.BF.4.0.8
$ cp -rf */* .
$ cp -rf IPQ5322/trustzone_images/ .
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

Once *copying* the existing Git repository is complete, change the directories where the files are present as described in this table before running the repo command:

Use git to obtain these files from ChipCode:	Local directory path to files fetched by git from ChipCode:
qsdk-qca-wifi qsdk-qca-wlan	apss_proc\out\proprietary\Wifi
meta-tools minidump common-tools qca-lib qca-mcs-apps qca-qmi-framework gpio-debug qca-diag qca-cnss-daemon athtestcmd fw-qca-stats qsdk-qca-athdiag nss-prop-user qca-rsrcmgr-bin	apss_proc\out\proprietary\QSDK-Base
qsdk-ieee1905-security	apss_proc\out\proprietary\Hyfi

2. Execute following commands to copy above deliverables to qsdk directory, and continue generating the QSDK framework:

```
$ repo init \
-u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak.git \
-b release \
-m  AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml
```

If this command does not work, use this repo init command:

```
$ repo init \
-u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak \
-b release \
-m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml  \
--repo-url=https://git.codelinaro.org/clo/la/tools/repo.git \
--repo-branch=qc-stable --no-clone-bundle
$ repo sync -j8 --no-tags -qc
$ mkdir -p qsdk/dl
$ cp -rf apss_proc/out/proprietary/Wifi/qsdk-qca-wifi/* qsdk
$ cp -rf apss_proc/out/proprietary/Wifi/qsdk-qca-wlan/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/meta-tools/ .
$ cp -rf apss_proc/out/proprietary/QSDK-Base/common-tools/* qsdk/
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-lib/* qsdk/
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-mcs-apps/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-qmi-framework/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/gpio-debug/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-diag/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-cnss-daemon/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/athtestcmd/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/fw-qca-stats/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qsdk-qca-athdiag/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/nss-prop-user/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/qca-rsrcmgr-bin/* qsdk
$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-ieee1905-security/* qsdk
$ cp -rf apss_proc/out/proprietary/QSDK-Base/minidump/* qsdk
$ rm -rfv qsdk/qca/feeds/qca/utils/ctrl_app_dut
```

AFC changes:

```
$ cp -rf apss_proc/out/proprietary/RBIN-AFCAgent/qca-afc-bin/* qsdk
$ cd qsdk/qca/feeds/afc
$ find -maxdepth 1 ! -name qti-mfg-provision ! -name qti-encrypt ! -name . -
  exec rm -rv {} \;
```

32-bit:

```
$ cd <ChipCode Directory>
$ mkdir -pv qsdk/staging_dir/target-arm/usr/lib/
```

Premium 32-bit	`$ cd qsdk/prebuilt/ipq53xx_32/ipq_premium`
LM512 32-bit	`$ cd qsdk/prebuilt/ipq53xx_32/ipq_512`

```
$ cp -fv ./libprovision.so ../../../staging_dir/target-arm/usr/lib/
$ cd <ChipCode Directory>
$ mkdir -p qsdk/staging_dir/target-arm/pkginfo/
$ touch qsdk/staging_dir/target-arm/pkginfo/qti-mfg-provision.provides
$ echo libprovision.so > qsdk/staging_dir/target-arm/pkginfo/qti-mfg-
  provision.provides
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

64-bit:

```
$ cd <ChipCode Directory>
$ mkdir -p qsdk/staging_dir/target-aarch64/usr/lib/
```

Premium 64-bit	`$ cd qsdk/prebuilt/ipq53xx_generic/ipq_premium/`

```
$ cp -fv ./libprovision.so ../../../staging_dir/target-aarch64/usr/lib/
$ cd <ChipCode Directory>
$ mkdir -p qsdk/staging_dir/target-aarch64/pkginfo/
$ touch qsdk/staging_dir/target-aarch64/pkginfo/qti-mfg-provision.provides
$ echo libprovision.so > qsdk/staging_dir/target-aarch64/pkginfo/qti-mfg-
  provision.provides
```

ChipCode Directory for this release:	**qca-networking-2022-spf-12-2_qca_oem**

NOTE: Prebuilt IPKs (such as qca-ezmesh-cmn, qca-ezmesh-ctrl, qca-ezmesh-agent) are in their folders:

`<ChipCode Directory>/qsdk/prebuilt/generic/`	Contains all 64-bit IPK images
`<ChipCode Directory>/qsdk/prebuilt/ipq53xx_32/`	Contains all 32-bit IPK images

These files are fetched from ChipCode and copied to the working QSDK top-level directory:			
Premium/LM	**Qualcomm® HY-FI™ and WHC**	Hy-Fi qsdk-whc qsdk-whcpy	apss_proc\out\proprietary\Hyfi apss_proc\out\proprietary\Hyfi\qsdk-whc apss_proc\out\proprietary\Hyfi\qsdk-whcpy

3. (Customers with Qualcomm EZMESH packages only) these files are fetched from ChipCode and copied to the working QSDK top-level directory:

These files are fetched from ChipCode and copied to the working QSDK top-level directory:			
Premium/ LM512	**EZMESH-SRC**	qsdk-ezmesh-src	NHSS.QSDK.12.2\apss_proc\out\proprietary\RSRC-EZMESH\qsdk-ezmesh-src
Premium/ LM512	**EZMESH-BIN**	qsdk-ezmesh-bin	NHSS.QSDK.12.2\apss_proc\out\proprietary\RBIN-EZMESH\qsdk-ezmesh-bin
Premium/ LM512	**EZMESH-ALG**	qsdk-ezmesh-alg-bin	NHSS.QSDK.12.2\apss_proc\out\proprietary\RBIN-EZMESH-ALG\qsdk-ezmesh-bin
Premium/ LM512	**EZ-ALG-SRC**	qsdk-ez-alg-src	NHSS.QSDK.12.2\apss_proc\out\proprietary\RSRC-EZMESH-ALG\qsdk-ezmesh-alg-src

These files are fetched from ChipCode and copied to the working QSDK top-level directory:			
Premium/LM512	**EZ-ALG-STA**	qsdk-ez-alg-sta	NHSS.QSDK.12.2\apss_proc\out\proprietary\FEAT-BIN-EZMESH-ALG-STATIC\qsdk-ezmesh-alg-sta
Premium	**Sigma-DUT**	sigma-dut.tar.bz2	NHSS.QSDK.12.2\apss_proc\out\proprietary\sigma-dut\sigma-dut.tar.bz2
Premium	**WAPid**	qsdk-wapid	NHSS.QSDK.12.2\apss_proc\out\proprietary\Wapid\qsdk-wapid
Premium	**IFLI-SRC**	qca	NHSS.QSDK.12.2\apss_proc\out\proprietary\RSRC-IFLI

4. Then run the additional code:

Premium/ LM	**HY-FI & WHC**	`$ cp -rf apss_proc/out/proprietary/Hyfi/hyfi/* qsdk`
		`$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-whc/* qsdk`
		`$ mkdir qsdk/qca/feeds/qca-son-mem-debug/qca-son-mem-debug`
		`$ mv qsdk/qca/feeds/qca-son-mem-debug/Makefile qsdk/qca/feeds/qca-son-mem-debug/Config.in qsdk/qca/feeds/qca-son-mem-debug/qca-son-mem-debug`
		`$ cp -rf apss_proc/out/proprietary/Hyfi/qsdk-whcpy/* qsdk`
		`$ sed -i "s/@PACKAGE_whc-son/@PACKAGE_whc-map/g" qsdk/qca/feeds/qca-lib/qca-wifison-ext-lib/Makefile`

Premium	**WAPid**	`git clone <wapid ChipCode Directory>`
		`$ cd <wapid ChipCode Directory>`
		`$ git checkout r12.2.r3_00009.0`
		`$ cd ..`
		`$ cp -rf <wapid ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/Wapid/qsdk-wapid/* qsdk`
		
		Where *wapid ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_wapid-src** for this release.

Premium	**Sigma-DUT**	`$ git clone <sigma-dut ChipCode Directory>`
		`$ cd <sigma-dut ChipCode directory>`
		`$ git checkout r12.2.r3_00009.0`
		`$ cd ..`
		`$ tar xjvf <sigma-dut ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/sigma-dut/sigma-dut.tar.bz2 -C qsdk`
		
		Where *sigma-dut ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_sigma-dut** for this release.

Premium	**IFLI-SRC**	`$ git clone <IFLI-SRC ChipCode Directory>`
		`$ cd <IFLI-SRC ChipCode directory>`
		`$ git checkout r12.2.r3_00009.0`
		`$ cd ..`
		`$ cp -rf <IFLI-SRC ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RSRC-IFLI/* qsdk`
		
		Where **<*IFLI-SRC*_**ChipCode Directory**>** is **qca-networking-2022-spf-12-2_qca_oem_ifli-src** for this release.

Premium/ LM512	**EZMESH-SRC**	EZMESH customers only:
		`$ git clone <ezmesh ChipCode Directory>`
		`$ cd <ezmesh ChipCode Directory>`
		`$ git checkout r12.2.r3_00009.0`
		`$ cd ..`
		`$ cp -rf <ezmesh ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RSRC-EZMESH/qsdk-ezmesh-src/* qsdk`
		
		Where *ezmesh ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ezmesh-src** for this release.
		Skip these sed commands in EZMESH-SRC if using **EZ-ALG-SRC** (that is, both **ezmesh-src** and **ezmesh-alg-src**):
		`$ sed -i 's/HYD_MODULE_STRATEGY=y/HYD_MODULE_STRATEGY=n/g' qsdk/qca/src/qca-ezmesh/ezmeshConfig.defs`

		$ sed -i '0,/ifeq/{/ifeq/d;}' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '0,/endif/{/endif/d;}' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '/libezmeshalg \\/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '/libezmeshagentalg \\/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile
		$ sed -i '/DUMP/d' qsdk/qca/feeds/qca-ezmesh/qca-ezmesh/Makefile

Premium/ LM512	**EZMESH-BIN**	EZMESH customers only:
		$ git clone <ezmesh ChipCode Directory>
		$ cd <ezmesh ChipCode Directory>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <ezmesh ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RBIN-EZMESH/qsdk-ezmesh-bin/* qsdk
		
		Where ezmesh ChipCode Directory is **qca-networking-2022-spf-12-2_qca_oem_ezmesh-bin** for this release.

Premium/ LM512	**EZMESH-ALG**	EZMESH customers only:
		$ git clone <ezmesh ChipCode Directory>
		$ cd <ezmesh ChipCode Directory>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <ezmesh ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RBIN-EZMESH-ALG/qsdk-ezmesh-alg-bin/* qsdk
		
		Where ezmesh ChipCode Directory is **qca-networking-2022-spf-12-2_qca_oem_ezmesh-alg** for this release.

Premium/ LM512	**EZ-ALG-SRC**	EZMESH customers only:
		$ git clone <ezmesh ChipCode Directory>
		$ cd <ezmesh ChipCode Directory>
		$ git checkout r12.2.r3_00009.0
		$ cd ..
		$ cp -rf <ezmesh ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/RSRC-EZMESH-ALG/qsdk-ezmesh-alg-src/* qsdk
		
		Where ezmesh ChipCode Directory is **qca-networking-2022-spf-12-2_qca_oem_ez-alg-src** for this release.

Premium	**EZ-ALG-STA**	**NOTE:** Pre-requisite for EZ-ALG-STA is EZMESH-SRC (ONLY). Please ensure no other EZMESH distros are included.
		
		This is applicable only for EZMESH customers that use glibc :
		$ git clone <ezmesh ChipCode Directory>
		$ cd < ezmesh ChipCode Directory>
		$ git checkout r12.2.r3_00009.0
		
		Where ezmesh ChipCode Directory is **qca-networking-2022-spf-12-2_qca_oem_ez-alg-sta** for this release.
		
		32-bit
		
		$ cd NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-EZMESH-ALG-STATIC/qca-ezmesh-alg-static-bin/prebuilt/ipq53xx_32/ipq_premium
		$ tar -xvf qca-ezmesh-alg-static*.ipk
		$ tar -xvf data.tar.gz
		$ cp ./usr/lib/libezmeshalg.a <Chipcode Directory>/qsdk/staging_dir/target-arm/usr/lib/

		$ cd <Chipcode Directory>
		
		64-bit
		
		$ cd NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-EZMESH-ALG-STATIC/qca-ezmesh-alg-static-bin/prebuilt/ipq53xx_generic/ipq_premium
		$ tar -xvf qca-ezmesh-alg-static*.ipk
		$ tar -xvf data.tar.gz
		$ cp ./usr/lib/libezmeshalg.a <Chipcode Directory>/qsdk/staging_dir/target-aarch64/usr/lib/
		$ cd <Chipcode Directory>

CTL_APP_SRC	WFA customers only:
	$ git clone <ctl_app_src ChipCode Directory>
	$ cd <ctl_app_src ChipCode Directory>
	$ git checkout r12.2.r3_00009.0
	$ cd ..
	$ cp -rf <ctl_app_src ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-SRC-CTRL-APP-DUT/qca-ctrl-app-dut/* qsdk
	
	Where ctl_app_src *ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ctlapp-src** for this release.

CTL_APP_BIN	WFA customers only:
	$ git clone <ctl_app_bin ChipCode Directory>
	$ cd <ctl_app_bin ChipCode Directory>
	$ git checkout r12.2.r3_00009.0
	$ cd ..
	$ cp -rf <ctl_app_bin ChipCode Directory>/NHSS.QSDK.12.2/apss_proc/out/proprietary/FEAT-BIN-CTRL-APP-DUT/* <destination directory>
	
	Where ctl_app_bin *ChipCode Directory* is **qca-networking-2022-spf-12-2_qca_oem_ctlapp-bin** for this release.

***** Distro Name: qca-networking-2022-spf-12-2_qca_oem_ifli-bin

IFLI-BIN customers to directly use BIN on the target (using TFTP).

6.2.3.3 Set up the QSDK build environment (one time)

QSDK supports the **Premium and LM512** profile for Linux kernel 5.4.213 support**.** The QSDK framework is developed using Ubuntu (from version 16.04 to version 22.04) and Debian. However, QSDK framework regenerates critical tools required to compile firmware at build time. Thus, the framework is independent from the host environment. Although it is developed using the listed distributions, it is expected to work on others such as Red Hat, Mint, or Fedora.

The required tools can be installed manually or using a script which is available in CAF.

6.2.3.3.1 Set up the build environment manually

This command is for Ubuntu 16.04 or higher (for older/32-bit Debian/Ubuntu releases; customize it for other distributions:

```
$ sudo apt-get install gcc g++ binutils patch bzip2 flex make gettext \
 pkg-config unzip zlib1g-dev libc6-dev subversion libncurses5-dev gawk \
 sharutils curl libxml-parser-perl ocaml-nox ocaml ocaml-findlib \
 python-yaml libssl-dev libfdt-dev
$ sudo apt-get install device-tree-compiler u-boot-tools
```

For Ubuntu 18.04 build hosts additionally, install:

```
$ sudo apt-get install libssl1.0-dev
```

For Ubuntu 20.04/22.04 build hosts additionally, install:

```
$ wget http://launchpadlibrarian.net/366014597/make_4.1-9.1ubuntu1_amd64.deb
$ sudo dpkg -i make_4.1-9.1ubuntu1_amd64.deb
```

6.2.3.3.2 Set up the build environment using script

Use these steps to get the required tools installed in the Ubuntu host for QSDK compilation. Verified on Ubuntu 16 and Ubuntu 22. Once the repo init and sync to the release AU from CAF (Code Aurora Forum) is done successfully, the **setup_qsdk.sh** script is available in **qsdk/scripts/setup_qsdk.sh**.

The script assumes that repo and git are already installed and aborts the execution otherwise.

1. If the repo init was not already done, run these commands:

```
$ repo init -u https://git.codelinaro.org/clo/qsdk/releases/manifest/qstak.git \
 -b release \
 -m AU_LINUX_QSDK_NHSS.QSDK.12.2.R4_TARGET_ALL.12.2.04.2049.023.xml \
 --repo-url=https://git.codelinaro.org/clo/la/tools/repo.git \
 --repo-branch=qc-stable --no-clone-bundle
$ repo sync -j8 --no-tags -qc
```

2. The user running the script should have admin privileges. If not, the script will abort upon unsuccessful verification of sudo/admin access.
3. Run the script:

```
 $sudo bash -x setup_qsdk.sh
```

NOTE: This is a one-time run script.

6.2.3.4 Create the QSDK build

Framework automatically downloads the open-source components needed during the build/ make process, ensure that an Internet connection is active on the build host when creating the build.

1. Install the different feeds in the build framework:

```
$ cd qsdk
$ ./scripts/feeds update -a
$ ./scripts/feeds install -a -f
```

2. Copy the base configuration to use for the build. Choose the **Premium/LM512** profile and use it to build with Linux 5.4.213 support.
3. Regenerate a complete configuration file and start the build:

Premium 32-bit

```
$ cp qca/configs/qsdk/ipq_premium.config .config
$ sed -i 's/TARGET_ipq807x_generic/TARGET_ipq53xx_ipq53xx_32/g' .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq53xx/g' .config
$ cp -rf prebuilt/ipq53xx_32/ipq_premium/* prebuilt/ipq53xx_32/
```

Premium 64-bit

```
$ cp qca/configs/qsdk/ipq_premium.config .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq53xx/g' .config
$ mkdir -p prebuilt/generic
$ cp -rf prebuilt/ipq53xx_generic/ipq_premium/* prebuilt/generic
```

LM512 32-bit

```
$ cp qca/configs/qsdk/ipq_512.config .config
$ sed -i 's/TARGET_ipq807x_generic/TARGET_ipq53xx_ipq53xx_32/g' .config
$ sed -i 's/TARGET_ipq807x/TARGET_ipq53xx/g' .config
$ cp -rf prebuilt/ipq53xx_32/ipq_512/* prebuilt/ipq53xx_32/
```

NOTE: Ignore any error messages such as Error: recursive dependency detected!

For Hy-Fi, Wi-Fi SON customers:

- Starting with the QCA_Networking_2020.SPF.11.3 release, SON and EasyMesh features are compiled as separate build. SON build supports only SON features and EZMESH build supports only EasyMesh features.
- Starting with the QCA_Networking_2021.SPF.11.4 release, MAP package qca-hyd-map is replaced with qca-ezmesh. Enable additional configuration to enable EZMESH packages.
- Use the SON package commands in the following table for a SON features enabled build. Use the EZMESH package commands for an EasyMesh features enabled build.

NOTE: These commands must be done before executing make.

4. Enable EZMESH configuration (customers with Qualcomm® Wi-Fi SON or EZMESH packages only):

EZMESH package (Full Mode)	`$ echo "CONFIG_PACKAGE_whc-mesh=n" >> .config`
	`$ echo "CONFIG_PACKAGE_hyfi-mesh=n" >> .config`
	`$ echo "CONFIG_PACKAGE_whc-map=y" >> .config`
	`$ echo "CONFIG_PACKAGE_hyfi-map=y" >> .config`
	Additional configuration to enable EZMESH packages:
	`$ echo "CONFIG_PACKAGE_qca-ezmesh-cmn=y" >> .config`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh=y" >> .config`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh-alg=y" >> .config`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh-ctrl=n" >> .config`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh-agent=n" >> .config`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh-agentalg=n" >> .config`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh-alg-static=n" >> .config`

EZMESH package (Co-located mode)	`$ echo "CONFIG_PACKAGE_whc-mesh=n" >> .config`
	`$ echo "CONFIG_PACKAGE_hyfi-mesh=n" >> .config`
	`$ echo "CONFIG_PACKAGE_whc-map=y" >> .config`
	`$ echo "CONFIG_PACKAGE_hyfi-map=y" >> .config`
	`Additional configuration to enable EZMESH packages:`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh-cmn=y" >> .config`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh-ctrl=y" >> .config`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh-agent=y" >> .config`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh-alg=y" >> .config`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh-agentalg=y" >> .config`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh=n" >> .config`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh-alg-static=n" >> .config`

```
$ echo CONFIG_BUILD_SHORTENED_PATH=y >> .config
$ make defconfig
$ make V=s
```

This is applicable only for EZMESH customers that use glibc Additional configuration to enable ezmesh alg static.

EZMESH ALG STATIC	`$ echo "CONFIG_EZMESH_ADD_STATIC_ALG=y" >> .config`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh-alg=n" >> .config`
	`$ echo "CONFIG_PACKAGE_qca-ezmesh-alg-static=n" >> .config`

5. For 64-bit firmware, once the premium 32-bit compilation completes, save all the U-Boot executable files openwrt-ipq5322*.elf from the directory **qsdk/bin/targets/ipq53xx/ipq53xx_32**. These files are required to generate all the variants of the complete 64-bit firmware image.
6. Download the required packages for the corresponding profile and create the image. Once the build is complete, these files are in the directory **qsdk/bin/targets/ipq53xx/ipq53xx_32** for 32-bit image and **qsdk/bin/targets/ipq53xx/generic** for 64-bit image.

32-bit	**ELF files for U-Boot variants**	openwrt-ipq5322*.elf
	SquashFS	openwrt-ipq53xx-ipq53xx_32-squashfs-root.img
	ITB	openwrt-ipq53xx-ipq53xx_32-ipq5322-alxx-fit-uImage.itb
64-bit	**SquashFS**	openwrt-ipq53xx-generic-squashfs-root.img
	ITB	openwrt-ipq53xx-generic-ipq5322-alxx-fit-uImage.itb

6.2.4 Generate a complete firmware image

IPQ53xx requires flashing multiple images for Bootup, including SBL, TZ, CDT, MIB images, Kernel, Filesystem, and so on. To simplify both image flashing and device boot, individual images are combined into a single Flattened Image Tree (FIT) image. FIT image components can be flashed into the respective partition based on user configuration. More tools required on the Ubuntu (from version 16.04 to 22.04) 64-bit machine include:

1. Install mkimage:
   ```
   $sudo apt-get install uboot-mkimage
   ```
2. Install DTC:
   ```
   $sudo apt-get install device-tree-compiler
   ```
3. Install libfdt:
   ```
   $sudo apt-get install libfdt-dev
   ```
4. Install Python 2.7.
5. Switch to the Qualcomm ChipCode directory:
   ```
   $ cd <ChipCode Directory>
   ```
   (if the current directory is qsdk, use cd ../ to navigate to ChipCode directory)
6. Copy the flash config files to **common/build/ipq**
7. Find the single images in the **common/build/bin** directory. Images are generated for default image content. Fewer images indicate an issue with one of the partition sizes, or that not all required files are present.

32-bit image:

```
$ cd <ChipCode Directory>
$ mkdir -p common/build/ipq
$ mkdir -p apss_proc/out/meta-scripts
$ cp qsdk/qca/src/u-boot-2016/tools/pack_v2.py apss_proc/out/meta-scripts/
$ sed -i 's#</linux_root_path>#/</linux_root_path>#' contents.xml
$ sed -i 's#</windows_root_path>#\\</windows_root_path>#' contents.xml
$ cp qsdk/bin/targets/ipq53xx/ipq53xx_32/openwrt* common/build/ipq
$ cp -rf apss_proc/out/proprietary/QSDK-Base/meta-tools apss_proc/out/
$ cp -rf qsdk/bin/targets/ipq53xx/ipq53xx_32/dtbs/* common/build/ipq/
$ cp -rf skales/* common/build/ipq/
$ cp qsdk/staging_dir/host/bin/ubinize common/build/ipq/
$ cp common/build/mksquashfs4 common/build/ipq/
$ cp -rf WLAN.WBE*/wlan_proc/build/ms/bin/*/FW_IMAGES/* common/build/ipq/
$ cp -rf TMEL.WNS*/patch/signed/tmel-ipq53xx-patch.elf common/build/ipq/
$ cd common/build
```

IPQ53xx+QCN92xx customers:

```
$ sed -i.bak '/<wififw_name>/d' ../../apss_proc/out/meta-
tools/ipq5332/config.xml
$ sed -i.bak
's#filename="wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img
"#filename="wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img"
#g' ../../apss_proc/out/meta-tools/ipq5332/flash_partition/emmc-
partition.xml
$ sed -i.bak
's#filename="wifi_fw_ipq5332_squashfs.img"#filename="wifi_fw_ipq533
2_qcn9224_v2_single_dualmac_squashfs.img"#g' ../../apss_proc/out/me
ta-tools/ipq5332/flash_partition/emmc-partition.xml
$ sed -i.bak
's#image=wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img#ima
ge=wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img#g' ../../
apss_proc/out/meta-tools/ipq5332/flash_partition/ipq5332-
ubinize.cfg
```

IPQ53xx+QCN92xx+QCN9160 customers:

```
$ git clone <chipcode_bin_york>
Where, <chipcode_bin_york> is qca-networking-2022-spf-12-
2_qca_oem_bin-york

$ cd <chipcode_bin_york_directory>
$ git checkout r12.2.r3_00009.0
$ cd ..

$ cp -rf
<chipcode_bin_york>/WLAN.WBE*/wlan_proc/build/ms/bin/*/FW_IMAGES/*
./ipq/

$ sed -i.bak '/<wififw_name>/d' ../../apss_proc/out/meta-
tools/ipq5332/config.xml
$ sed -i.bak
's#filename="wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img
"#filename="wifi_fw_ipq5332_qcn9224_v2_single_dualmac_qcn9160_squas
hfs.img"#g' ../../apss_proc/out/meta-
tools/ipq5332/flash_partition/emmc-partition.xml
$ sed -i.bak
's#filename="wifi_fw_ipq5332_squashfs.img"#filename="wifi_fw_ipq533
2_qcn9224_v2_single_dualmac_qcn9160_squashfs.img"#g' ../../apss_pro
c/out/meta-tools/ipq5332/flash_partition/emmc-partition.xml
$ sed -i.bak
's#image=wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img#ima
ge=wifi_fw_ipq5332_qcn9224_v2_single_dualmac_qcn9160_squashfs.img#g
' ../../apss_proc/out/meta-tools/ipq5332/flash_partition/ipq5332-
ubinize.cfg
```



```
$ export BLD_ENV_BUILD_ID=<profile-name>
```

```
$ python update_common_info.py 32
```

Where <profile-name> is either P or LM512.

64-bit image:

```
$ cd <ChipCode Directory>
$ mkdir -p common/build/ipq_x64
$ mkdir -p apss_proc/out/meta-scripts
$ cp qsdk/qca/src/u-boot-2016/tools/pack_v2.py apss_proc/out/meta-scripts/
$ sed -i 's#</linux_root_path>#/</linux_root_path>#' contents.xml
$ sed -i 's#</windows_root_path>#\\</windows_root_path>#' contents.xml
$ cp qsdk/bin/targets/ipq53xx/generic/openwrt* common/build/ipq_x64
```

Copy the saved openwrt-ipq53*-u-boot*.elf from the 32-bit build to common/build/ipq_x64

```
$ cp -rf apss_proc/out/proprietary/QSDK-Base/meta-tools apss_proc/out/
$ cp -rf qsdk/bin/targets/ipq53xx/generic/dtbs/* common/build/ipq_x64/
$ cp -rf skales/* common/build/ipq_x64/
$ cp qsdk/staging_dir/host/bin/ubinize common/build/ipq_x64/
$ cp -rf WLAN.WBE*/wlan_proc/build/ms/bin/*/FW_IMAGES/* common/build/ipq_x64/
$ cp -rf TMEL.WNS*/patch/signed/tmel-ipq53xx-patch.elf common/build/ipq_x64/
$ cp common/build/mksquashfs4 common/build/ipq_x64
$ cd common/build
```

IPQ53xx+QCN92xx customers:

```
$ sed -i.bak '/<wififw_name>/d' ../../apss_proc/out/meta-
tools/ipq5332/config.xml
$ sed -i.bak
'S#filename="wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img
"#filename="wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img"
#g' ../../apss_proc/out/meta-tools/ipq5332/flash_partition/emmc-
partition.xml
$ sed -i.bak
's#filename="wifi_fw_ipq5332_squashfs.img"#filename="wifi_fw_ipq533
2_qcn9224_v2_single_dualmac_squashfs.img"#g' ../../apss_proc/out/me
ta-tools/ipq5332/flash_partition/emmc-partition.xml
$ sed -i.bak
's#image=wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img#ima
ge=wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img#g' ../../
apss_proc/out/meta-tools/ipq5332/flash_partition/ipq5332-
ubinize.cfg
$ sed -i.bak
's#image=wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img#ima
ge=wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img#g' ../../
apss_proc/out/meta-tools/ipq5332/flash_partition/ipq5332-
ubinize_64.cfg
```

IPQ53xx+QCN92xx+QCN9160 customers:

```
$ git clone <chipcode_bin_york>
Where, <chipcode_bin_york> is qca-networking-2022-spf-12-
2_qca_oem_bin-york

$ cd <chipcode_bin_york_directory>
$ git checkout r12.2.r3_00009.0
$ cd ..

$ cp -rf
<chipcode_bin_york>/WLAN.WBE*/wlan_proc/build/ms/bin/*/FW_IMAGES/*
./ipq_x64/

$ sed -i.bak '/<wififw_name>/d' ../../apss_proc/out/meta-
tools/ipq5332/config.xml
$ sed -i.bak
's#filename="wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img
"#filename="wifi_fw_ipq5332_qcn9224_v2_single_dualmac_qcn9160_squas
hfs.img"#g' ../../apss_proc/out/meta-
tools/ipq5332/flash_partition/emmc-partition.xml
$ sed -i.bak
's#filename="wifi_fw_ipq5332_squashfs.img"#filename="wifi_fw_ipq533
2_qcn9224_v2_single_dualmac_qcn9160_squashfs.img"#g' ../../apss_pro
c/out/meta-tools/ipq5332/flash_partition/emmc-partition.xml
$ sed -i.bak
's#image=wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img#ima
ge=wifi_fw_ipq5332_qcn9224_v2_single_dualmac_qcn9160_squashfs.img#g
' ../../apss_proc/out/meta-tools/ipq5332/flash_partition/ipq5332-
ubinize_64.cfg
$ sed -i.bak
's#image=wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img#ima
ge=wifi_fw_ipq5332_qcn9224_v2_single_dualmac_qcn9160_squashfs.img#g
' ../../apss_proc/out/meta-tools/ipq5332/flash_partition/ipq5332-
ubinize.cfg
```

```
$ export BLD_ENV_BUILD_ID=<profile-name>
$ python update_common_info.py 64
```

Where <profile-name> is P.

6.2.5 Flash the complete default software image

6.2.5.1 Set up the flash environment

1. Ensure that the board console port is connected to the PC using these RS232 parameters:
   - 115200 bps
   - 8N1
2. Confirm that the PC is connected to the board using one of the Ethernet ports. The PC must have a TFTP server launched and listening on the interface to which the board is connected. The interface must have an IP address configured manually. At this stage power up the board and after a few seconds, press any key during the countdown.

6.2.5.2 Standard board configuration: load the image in flash and boot the platform

1. Copy the desired **xxxx-ipq5332-single.img** to the TFTP server root directory.
2. Check hardware jumper configuration according to reference board and default memory configuration (see the appropriate board setup guide for more information) to verify which flash memory the board is booting from.
3. Confirm the machine ID, Meta version, profile, and selected single image to match the memory type that the board boots from.

Supported Flash	Profile Support	Image Name
NAND eMMC NOR + NAND NOR + eMMC	Premium LM512	eMMC 64 bit: emmc-ipq5332_64-single.img eMMC 32 bit:emmc-ipq5332-single.img NAND 64 bit: nand-ipq5332_64-single.img NAND 32 bit: nand-ipq5332-single.img NOR + eMMC 64 bit: norplusemmc-ipq5332_64-single.img NOR + eMMC 32 bit: norplusemmc-ipq5332-single.img NOR + NAND 64 bit: norplusnand-ipq5332_64-single.img NOR + NAND 32 bit: norplusnand-ipq5332-single.img

4. Set the IP address and server IP using the TFTP process:
   ```
   set ipaddr 192.168.1.11
   set serverip 192.168.1.xx (TFTP server address)
   ping $serverip
   set bootargs console=ttyMSM0,115200n8
   tftpboot xxxx-ipq5332-single.img
   ```
5. Flash the image:
   ```
   imgaddr=$fileaddr && source $imgaddr:script
   ```
6. Power cycle the board after step 5 has completed, as indicated by printing of the U-Boot prompt (maybe hundreds of seconds depending on image size and memory technology).1.

6.2.5.3 Upgrade the firmware

This release can upgrade board images without the need for a TFTP server. After using U-Boot to flash an initial image and boot the device, use the web interface for future upgrades. After using U-Boot to flash an initial image and boot the device, future upgrades can be done from either the OpenWrt web interface or the serial console. Upgrade the existing flash image using either the appropriate single image file or the apps image file generated by **update_common_info.py**.

Upgrading image files via the web interface takes several minutes to complete depending on factors such as memory technology, vendor, image size, browser connectivity, and network load.

Completion of the flash upgrade process is signaled by the refresh of the OpenWrt login page or lack of further messages on the serial console.

- An invalid image remains in the flash memory if the upgrade process is interrupted, such as if system power is lost during the upgrade or a key-press event is detected on the serial console port.
- If a board is configured for failsafe boot, and the U-Boot environment variable *bootargs* contains any reference to the rootfs or other partitions managed by the failsafe scheme. In this case, the sysupgrade process may only be partially successful: the board may or may not boot subsequently, and the image may only be partly updated.
- Using the **sysupgrade** command from the Linux console requires the sysupgrade image to be visible:
  - Within the Linux filesystem supplied using development host SSH or TFTPD server, OR
  - On an NFS server that is mounted to the local file system, OR
  - Transferred using a USB storage device
- The sysupgrade feature can only be used to update the image in the memory the board boots from and is used by the running kernel. If the board supports multiple memory technologies, use U-Boot to change the image in memories that the board did not boot from.

6.3 Flash Wi-Fi firmware image only

The Wi Fi firmware/BDF information stored in the flash image can be updated without rewriting the whole flash image as shown in this section. Regardless of the memory type the system boots from, follow the same basic process of using tftpboot to load the new Wi-Fi firmware image to DDR, then identify the correct location in the flash and write the Wi-Fi firmware image to that location.

Table 2-1 Image file name by radio combination for IPQ5322.ILQ.12.2

Image File Name	Radio Combination
wifi_fw_ipq5332_qcn9224_v2_single_dualmac_squashfs.img	IPQ53xx + QCN92xx V2 (Dual Radio)
wifi_fw_ipq5332_qcn9224_v2_single_dualmac_qcn9160_squashfs.img	IPQ5332 + QCN9224 + QCN9160

NOR+NAND boot/NAND boot

- TFTP the image (see Table 2-2 for image file names):
  ```
  # tftpboot <Image File Name>
  ```
- Flash the partition using the flash command:
  ```
  # flash wifi_fw
  ```
  Where wifi_fw is the Wi-Fi UBI volume name. This name is used for flash command.

NOR boot/NOR+eMMC boot/eMMC boot

- TFTP the image (wifi_fw_squashsfs.img) (see Table 2-2 for image file names):
  ```
  # tftpboot <Image File Name>
  ```
- Flash the partition using the flash command:
  ```
  # flash 0:WIFIFW
  ```

NOTE: 0:WIFIFW is the Wi-Fi partition name. This name is used for flash command.

6.4 IPQ5322 features delivered

6.4.1 WLAN deliverables

The hardware features in the IPQ5322 chip family data sheets meet Qualcomm CS metrics. This software release enables the features listed in this section. Future releases plan to enable additional features.

- Boot Software
  - Emergency Download through USB
  - Boot from NOR
  - Boot from NAND
  - Boot from eMMC
  - Boot from NOR + eMMC
  - Boot from NOR + NAND
  - Fail Safe Boot for upgrade
  - DDR4
  - eMMC Write Protect
  - Crypto – ARM ISA
  - Turbo mode support. (NOM not supported)
- Platform Security / Trust Zone
  - QSEE/Trust Zone
  - Secure Boot
  - Access Control
  - Stack/Heap/Data Execution protection
  - Crypto – ARM ISA
- U-boot
  - Tftpboot
  - Crashdump
  - U-boot diagnostics
  - Ethernet switch Driver(Manhattan)
  - Ethernet PHY Driver-NAPA SFP
  - Ethernet PHY Driver – 10G SFP
- OS Kernel : Linux
  - Quad Core A53
  - 32-bit Native Kernel
  - 64-bit Native Kernel
  - Pre-emption
  - Perf tools
  - Kernel text protection

- Interface Drivers
  - UART /HS-UART
  - SPI
  - I2C
  - USB
  - LED - GPIO
  - MDIO
  - USB attached SCSI (UAS)
  - PCIe 3.0 root complex
  - PCIE, MHI
- QSDK platform
  - OpenWRT 19.07
  - Premium Profile
  - 512M Profile (Not Applicable for RDP441)
  - Software/Firmware upgrade
  - 32-bit apps on 64-bit kernel in compatibility mode
  - Secure Firmware Upgrade
  - Linux module signing
  - Openwrt sysupgrade support HW root of trust
- Tools
  - Ram Parser
  - QxDM
  - Single Image Packing
- Debug
  - JTAG Scripts
  - Diag
  - QDSS
- QDSP6 Subsystem
  - QuRT™
  - UART
  - Copy Engine
- Other
  - USB Storage
- TME-L
  - Secure boot
  - QTI Provisioning
- Soft SKU
  - License Manager

- Networking
  - EDMA Base Driver
  - Enhanced Connection Manager (SFE support)
  - Base PPE driver – interface configuration
  - Base PPE tunnel driver
- SSDK
  - QCA8081 (Napa) Driver
  - MACSEC (Napa Only)
  - IEEE1588 (Napa Only)
  - 1G/2.5G/10G SFP
  - SGMII/SGMII+/USXGMII/XFI
  - Manhattan switch driver
  - S17c switch driver
  - FDB Bridging and VLAN translation
  - LAG/Trunk
  - ACL and Pre-ACL
  - QoS priority mapping, SP/WRR scheduler
  - Buffer management
  - Queue management
  - Ingress port policing, virtual port policing, flow & ACL based policing
  - Egress port/queue/flow base shaper
  - Virtual port FDB bridging and VLAN translation
  - Management Control packets
  - Service code, Port Mirroring and STP.
  - L3 routing and PPPoE.
  - Tunnel(MAP-E,MAP-T, DS-Lite, VxLAN and GRE).
  - Atheros header.
- FTM
  - Bands: 2GHz
  - Chain Masks: 0x3. (2 chains)
  - Spatial Streams: 1SS across modes, 2SS across modes.
  - SU / Single PHY only.
  - MCS:0-13
  - MCS15
  - BW:
    - 20/40M for 2GHz.
    - CS KPIs.
  - 11be (upto MCS13, MCS15)
  - XTAL Cal
  - Rx Gain Calibrations
  - NF Characterizations

- TPC + CLPC functional enablement QDART enablement
- Channel walkthrough
- IPQ5322+QCN9xxx Calibration validated.
- List Mode
- OLPC Temperature Compensation

■ MM

- Bands: 2GHz
- Chain Mask: 0x3.
- Spatial Streams: 2SS across modes
- SU / Single PHY only.
- MCS:0-11
- BW:
  - 20/40 for 2GHz.
- Basic association and ping in 11be mode
- OFDMA – Functionality
- MLO – Functionality
- CALDB
- Online Cals:
  - RxDCO
  - NF CAL
  - Tx IQ
  - Rx IQ
  - Tx CL
  - eDPD
  - BW Filter CAL

6.5 QCN9160 Scan Radio Support

This software release supports QCN9160 scan radio on RDP 441.

Supported Platform	RDP 441 (MCN# 65-35731-100)
Supported scan radio	QCN9160 (MCN # 30-35983-200)
Supported profile	1G memory profile
Machine ID	8060101 When using RDP441 with QCN9610 for very first time. Please update the machine ID in uboot and flash the image.# To set the machine ID. setenv machid 8060101 tftpb nand-ipq5332_64-single.img imgaddr=$fileaddr && source $imgaddr:script # Disable the MLO setenv bootargs 'console=ttyMSM0,115200n8 cnss2.enable_mlo_support=0' saveenv
Necessary Bring-up Steps	To validate QCN9160 FTM mode, it is important to disable the MLO feature. Steps to disable MLO, append the following in boot args in uboot env. 'cnss2.enable_mlo_support=0'
QCN9160 Test Tree	QCN9160 Test Tree D10-36059-101_MI_2G_2X2_YORK_2G5G6G_2x2_WKK_6G_4x4_RDP441_RevC.cxtt
FTM Capabilities	Supported
MCS support	MCS 0-6
Support for scan radio features	Supports scan features. Feature complete and delivered with L0 validation. Please note that the following features are **not** supported in York Scan. ▪ CFR and AoA ▪ RTT Initiator and Responder ▪ Blanking

7 QCN9074 scan radio support

This software release supports scan radio features. Details for FTM and basic scan feature support are included here:

Supported platform	RDP0459
RDP MCN	RDP: 65-37825-300 (DEV-RDP)
Machine ID	8050c01
Test Tree	\\tdcfileserver\HW_TDC\Pine\30-12944-100_BRF\Test_Tree
Supported Scan Radio	QCN9074 scan radio QCN 9070
DCN No	NB-HDK DCN: DP25-31824-60
Support for FTM	Exists
Supported Scan features	▪ Special VAP ▪ Support 2GHz, 5GHz, 6GHz channel settings ▪ Wireless scan ▪ Monitor mode support ▪ MGMT Transmit support ▪ MGMT Transmit in DFS channel ▪ Spectral Scan support ▪ RF Sensing and AoA in Special VAP mode. ▪ Scan Radio – Host specific channel switch time optimization ▪ Spectral Analysis – CPU optimization ▪ Scan Radio – Target specific channel switch time optimization

8 QCN9274 features delivered

8.1 WLAN deliverables

The hardware features in the QCN9274 chip family data sheets meet Qualcomm CS metrics. This software release enables the features listed in this section. Future releases plan to enable additional features.

- Platform Configuration
  - 1 GB, 2G, 3G (64-bit Host Datapath, Direct switch)
  - 512 MB (32-bit Host Datapath, Direct switch)
  - QCN9274 boots with root of trust (Secure Boot)
- Product Configuration / Modes
  - QWRAP/ProxySTA
  - DBDC Repeater
  - Non-Associated WDS
  - Promiscuous Mode
  - AP Mode
  - WDS/Repeater AP
  - Extender AP
  - Extender STA
  - DBS
  - TBS
  - QBS
  - Penta band
  - Split-Radio
  - Raw Mode
- 11ac optional and enhancement features
  - STBC
  - LDPC
  - Short Guard Interval
  - 11ac features in 2.4GHz (256QAM, 40MHz, 4x4 4ss)
  - SU TxBF
  - SU TxBF beamformee
  - DL MU-MIMO
- 11ax Features
  - LDPC

- 11ax association
- 11ax SU MIMO
- 1024 QAM
- 11ax TxBF
- 11ax TxBF beamformee
- DL-OFDMA (up to 37 users func, 18 user KPI)
- DL MU-MIMO
- UL MU-MIMO
- UL-OFDMA (up to 37 users func, 18 user KPI)
- MU RTS/CTS
- Extended Range (DCM, ER_SU)

■ 11be features
- 2 GHz up to 40 MHz
- 6 GHz up to 320 MHz
- 5 GHz up to 240 MHz
- 4K QAM
- Async MLMR – TxBF
  - SU, OFDMA/MU-MIMO (MU+SU)
  - Upto 4 links
- 1024 BA
- 11be TxBF
- 11be TxBF Beamformee
- DL-OFDMA (up to 16 users)
- UL-OFDMA (up to 16 users)
- DL MU-MIMO
- UL MU-MIMO
- MU RTS/CTS
- Puncturing (Static)
- eMLSR
- MED35
- Multi-RU
- Provisioned MLO (Single AP FH, Mesh FH)
- P-MLO(single link-primary) w DL/UL MU-MIMO
- P-MLO(single link-primary) w DL/UL MU-MIMO

■ Client Device Management
- Quick STA kick out
- Receive Operating Mode Indication (ROMI)
- RSSI-based Association Rejection
- RSSI of STA
- Isolate Wi-Fi STAs within a BSS
- De-authentication On Demand

- Multi-Band Operation (MBO)
- 512 clients (per QCN9274 maximum)
- 16 VAPs per Radio
- WFA Optimized Connective Experience (OCE)

- Locationing
  - 11mc RTT
  - 11az RTT NTBR
  - 11be CFR
- BSS/MBSS
  - Standard Beacon
  - Staggered beacons
  - Burst beacons
  - Beacon Offload
  - Beacon control
  - Hidden SSID
  - STA intra-BSS communication
  - MBSSID (8 VAPs)
  - EMA
  - Minimum Probe Response/Beacon Tx Rate
  - 11ax - Multiple BSSID IE (single transmitting beacon for Multiple VAPs)
  - BSS Color Advertisement
  - OBSS PD for special reuse
- Packet Management
  - A-MSDU/A-MPDU Tx/Rx
  - A-MPDUs with A-MSDUs
  - Linux/NAPI
  - Tx/Rx Meta Data
  - WMM
  - API to configure queue size per target
  - NoDS Mesh Features
  - WMM-Access Class Override
  - VLAN
  - Management Frame Filtering and Forwarding - 802.11h
  - Multicast to Unicast Conversion
  - Airtime Fairness (ATF)
  - Video Over Wireless (VOW)
  - ETSI EN 301 893 (10ms TXOP)
  - P-MLO(single link-primary) w DL/UL MU-MIMO
- Power and Thermal Management
  - U-APSD
  - Green AP

- MIMO Power Save
- Thermal Throttling by Reducing DC
- Smart Antenna (Serial Mode)-QCN90xx parity
- TWT (Legacy)
- TWT (Restricted)

- Run-time Calibrations
  - Closed loop power control
  - DC offset correction
  - Carrier leakage correction
  - CalDB in 512MB profile and above
  - 320MHz IQ Cal
  - DPD
- Channel Management
  - Channel Range Selection
  - Manual Channel Selection
  - RSSI, NF and RX gain calibration
  - Per-subband NF CAL
  - ACS
  - ICM
  - DCS
  - Off-Channel TX/RX
  - 6 GHz LPI/VLP/SP
  - AFC
- Security
  - Open Security
  - WPA, WPA2, WPA-WPA2 Hybrid, WPA3
  - CCMP-128/256
  - GCMP-128/256
  - BIP-CMAC-128/256
  - WAPI 1.0
  - WPS2.0
  - MLO Security
- Service Aware Wi-Fi
  - SAWF API
- Regulatory
  - Country Code Configuration
  - Set Regulatory Domain
  - Regulatory Domain & CTL Engine
  - DFS
  - aDFS
  - DFS with puncturing (functional)

  - Spectral (upto 320MHz)
  - Regulatory Universal Mode
  - LPI Supported Regions
- EasyMesh
  - Band Steering and AP Steering
  - Daisy Chain
  - Smart Monitor Based Steering
  - Up to EasyMesh r4 Features and Certification
  - DE Commercialization
  - AFC
  - E@Home
  - MLO – 2 Link FH and BH support
  - Provisioned MLO
  - Pentaband with 2 Link MLO
- WFA Pre-Scan
  - WFA – Basic
  - WFA – WPA/WPA2 Personal
  - WFA – WPA/WPA2 Enterprise
  - WFA – Protected Management Frame
  - WFA – 11n, 11ac up to Rel 2, 11ax up to Rel 2
  - WFA – WPS 2.0
  - WFA – WPA3
  - WFA – WMM
  - WFA – Voice-Enterprise
  - WFA – WNM Network Power Save
  - WFA – Hot Spot 2.0 – Passpoint
  - WFA – Vantage 2 Carrier Grade Certification
  - WFA – Wi-Fi6E
  - WFA – 11be Plug fest features (PF2 support)
  - WFA – 11be Plug fest features (functional)
- Factory Calibration
  - Transmit Power Calibration – Full Point CAL
  - Transmit Power Calibration – Golden Bin
  - Transmit Power Calibration – 1pt CAL
  - Crystal Calibration
  - Receive Gain Calibration
  - Calibration storage in Flash/EEPROM
  - Noise floor CAL
  - List mode

9 Known issues

9.1 IPQ9574.ILQ.12.2 – IPQ9574 + QCN9274

9.1.1 Stability issues

Sl. No.	Title	Area / Group	Impact
1.	Q6 Crash : wal_monitor.c:1195 Assertion qcache_peer->ml_peer_ptr && !(peer_delete_monitor_time_delta >= g_wall ppe_drv_v4_stats_callback_register+0x38c/0x4b4 Q6 Crash : wal_rx.c:416 Exception detected Q6 Crash : ar_wal_mlo_ipc.c:1552 ExIPC: Exception recieved tid=1d func: wal_mlo_ipc_get_tqm_cmdq	MLO / Multiclient	Customer may observe stability issue in Multinode MLO scenarios along with BW change, Puncturing, Channel change with multiple clients connected across nodes and running traffic.
2.	Hostapd crash observed in MLO AP-STA scenario - Endurance Test		Customer may observe hostapd crash occasionally leading to MLO Station disconnect while running UDP Bi-Di traffic between them.

Sl. No.	Title	Area / Group	Impact
3.	APSS Crash : Kernel Panic : ModuleOrFile = qca_ol PC = hal_qcn9224v1_attach+0x4d64/0x11770 LR = dp_rx_mon_process_2_ APSS Crash : Kernel Panic : ModuleOrFile = nf_conntrack PC = nf_ct_unconfirmed_destroy+0x314/0x3c4 LR = nf_ct_unconfirmed_destr APSS Crash : Kernel Panic : ModuleOrFile = umac PC = ieee80211_add_basic_mlo_rnr_ie+0x680/0x10d8 LR = ieee80211_add_basic_mlo_r APSS Crash : Kernel Panic : ModuleOrFile = umac PC = mlme_restart_req_timeout+0x11c/0x1a0 APSS Crash : Kernel Panic : ModuleOrFile = wifi_3_0 PC = dp_rx_ppe_del_flow_entry+0x48/0x26c LR = dp_rx_ppe_del_flow_entry+0x2c/0x26		Customer may observe stability issue sporadically in Multinode MLO scenarios along with BW change, puncturing, channel change, endurance or wifi restart tests when multiple clients are connected across nodes and running traffic.
4.	Memory degradation observed on a daisy agent (RE2) after 6 hours of run of endurance traffic testing.	EzMesh	Customer may randomly observe memory degradation after 6 hours on daisy agent in an endurance testing with 30 clients connected across and running YT, multicast, conf calls, FTP.
5.	MLO link gets disconnected during reboot test on the controller. Agents switches and stays in SLO.		Customer might observe MLO-SLO switch when in Backhaul Link performing reboot test on controller in multi node EM scenario with clients connected across all nodes and running multi TiD traffic.

Sl. No.	Title	Area / Group	Impact
6.	Backhaul disconnect over the weekend reboot test on daisy agent		Customer might observe backhaul disconnection on EM agents on a multi node multi-VAP EM testing with MLO config and running traffic across multi clients connected across all nodes and running AP reboot test on agents.
7.	GW is not reachable in daisy agent (RE3) after 30 hours of run during endurance traffic testing.		Customer may observe GW unreachable issue in a long run(30+ hours) Multinode MLO - EzMesh scenario with 30 clients connected across nodes and running real world traffic (YT, video conference, multicast, FTP).
8.	Spirent Clients disconnecting with GTK rekeying option set in AP	Spirent C50	Customer may observe Client disconnection with 300s GTK rekeying interval set and more than 400 clients connected to AP on the VAP with keying interval set.

9.1.2 Performance issues

Sl. No.	Title	Test Area	Description
1.	Throughput imbalance observed between DL and UL in bidi traffic with HMT eMLSR client. The failure is not localized if its AP or HMT.	eMLSR	Customer will observe peak throughput imbalance between DL and UL when running BiDirectional traffic with HMT eMLSR client.
	eMLSR TP not meeting KPI by 16% in TCP and 25% in UDP. The failure is not localized if its AP or HMT.		Customer may observer lower throughout with HMT eMLSR client.

Sl. No.	Title	Test Area	Description
2.	Sudden fall in RVR for 2G DL MUMIMO 2SS+2SS mode	MUMIMO	Mid-Range MU Gain will be lower than max possible in 2SS+2SS 2G mode
	Low Tput and MU gain is observed because of 35% MU BRP Errors occured while doing 11AX DL MU test in 8USER UDP/TCP 6G/5G/2G HE 160/80/40.		Customers may observer lower or negative gain in 6G320Mhz.
3.	Observing high latency of around 20 ms in Downlink direction TCPDL_Throughput+Latency test in MLO mode when compared with SLO mode	OTA Perf	In a multi client environment - 5G&6G (Connected 15 clients and 3 active clients) Customer will observe high latency of 10-20 ms in TCP downlink direction when having an EMLSR + one MLMR + one legacy client in MLO mode, compared to SLO.
	Observing low DL MLO throughput compared to SLO throughput in congested OTA performance test		In an assymtetric loading condition with 2G+5G as MLD AP, when the 2G link is highly congested, customer might observe lower throughout in MLO client compared to smilar SLO client.

9.1.3 Functional issues

Sl. No.	Title	Test Area	Product Impact
1.	When GPS is updating location parameters in the location config file, height is not updated based on the response from the barometer	AFC	When GPS is updating location parameters in the location config file, height is not updated based on the response from the barometer. Customer may observe incorrect height than present in that Geolocation
	AP tx power is wrongly updated as 25dbm after reboot.		Medium Impact: AP tx power is wrongly updated as 25dbm after reboot when skyhook & gps functionality is enabled.
	AP power mode is updated to 0 (LPI) when CSA channel change command is issued for a punctured 80Mhz BW channel impacting Dynamic pucturing.		Afer puncturing, when CSA is issued, AP would switch to LPI mode instead of applying the punctured pattern in the targtet channel & operate in SP mode.
	Very Random observation of AP not moving to SP Channel when AWGN detection has happended - when AFC is enabled. (1 / 10 iterations)		Customers may observe sometimes AP does not move to SP Channel when AWGN detection has happended - when AFC is enabled. (1 / 10 iterations)

Sl. No.	Title	Test Area	Product Impact
	AP not selecting the best power after channel change when negative EIRP values are sent in the response		AP does not select the best power after channel change if negative EIRP values are sent in the response (error case test)
	Puncturing is not getting enabled by default for SLO Outdoor + AFC		Puncturing is not working for SLO Outdoor + AFC
	If invalid payload is sent to the AP from dummy server, request ID mismatch error is observed even when there is no mismatch in the request ID.		If invalid payload is sent to the AP from dummy/test server (error case test), request ID mismatch error is observed even when there is no mismatch in the request ID.
2.	Metrics calculation is not happening properly and hence impacting BW allocation for clients connected to BH	EzMesh	BW allocated to clients connected in daisy will have an impact due to metrics average.
	SNR / Date rate of few packets are changing drastically which is impacting FH/BH steering.		Inconsistent behaviour seen once in 12 hour run. Fronthaul & BH steering might not happen due incorrect data rate used in steering logic.
	Wrong updates of primary link in the station database while EzMesh application is restarted and hence primary link is wrong in MLO connection		stadb table wrongly updates the primary link after ezmesh restart.
	Ezmesh assertion is observed during path capacity estimate calculation.		Ezmesh assertion seen while performing path capacity calculation when client RSSI details w.r.t Agent are not updated in stadb s bss table.
	If fast transtion is enabled, AP Steering is failing due to assoc time out		Customers may see failure in steering if FT is enabled due to assoc failure during steering.
	Credential cloning is not triggered in Penta band Agent with DPP+SLO configuration		Clients and Repeaters will not be able to get connected to Agent onboarded with DPP+SLO method as credential cloning fails.
	With Additional FH enabled, initial cloning is not hapenning in SLO+DPP feature		Customers cannot use additional Fronthauls vaps with DPP enablement.
	MLO client forcefully disconnected by AP with offload steering		MLO client getting disconnect forcefully by AP during channel overload situation
	ezmeshCtrl is not running in DPP+Colocated Controller (feature not implemented yet)		ezmeshCtrl will not run in colocated controller when user configure DPP+Colocated mode
	AFC based dynamic puncturing is not applied with EIRP threshold configured		AFC based puncturing not working in Easy mesh config

Sl. No.	Title	Test Area	Product Impact
	When channel selection feature is enabled, on manual BW selection, the best BW is not getting updated.		Channel supervision does not work in radar hit scenario due to which AP does not operate in best channel/bw when channel is manually changed in preference mode. This is negative use-case for channel supervision.
	With Co Located MLO mode , when Channel BW selection happens with global preference enbaled, EM application restarts		In MLO collocated mode, assertion happens when there is channel selection.
	Measuring state timer is reduced from 45 secs to ~3secs in Colocated mode - Backhaul steering fails		Customers will see backhaul steering is not initiated in Colocated controller mode. Agent will not be able to choose best uplink node to operate.
	AP steering is not happening between controller and agents in colocated MLO mode		Customers will see failure in steering when colocated mode with MLO is used.
	Randomly ARP is taking long time to resolve and leading to Ping failure between the stations connected to CAP and RE in MLO enabled MAP R2. (Seen 2 of 10 iterations in automated test cases)		Customer would see delay in ARP response when running traffic between wireless clients where one STA associated with CAP of 2G and other STA associated to RE of 5G.
3.	Multicast traffic stopped after 2Ghz link become active link via Broadcast T2LM feature	MLMR	Customer will observe Multicast traffic is stopped whenever 2Ghz link become active link when Broadcast T2LM enabled
4.	Single Occurance crash seen in performance when jffs2reset -yr is given	Stability	Customer will observe crash with jffs2reset -yr (factory reset)
	Random one-time crash seen in Functional QWRAP mode tests		Customer may see stability issue when testing in QWRAP mode with multiple clients connected to AP running multi TiD traffic. Low impact.. Occurance 1/5
	Single Occurance crash seen in IOT 3 Link MLO tests with 20 clients connected to 2G & 5G each on Ext AP		Crash seen once while running MLO tests using IOT clients

9.2 Legacy platforms

9.2.1 Stability issues

S.No	Title	Target	Impact
1.	Q6 Crash - M3 Assert in Core : PhyA OPCODE : WRITE - MASTER : DMA3 - SLAVE : ROBE/PMI	IPQ9574	Customer may see crashes in WDS/ EasyMesh setups running with TCP/UDP Flood traffic between CAP and RE together with Wi-Fi restart, reboot, repacd restart scripts running in parallel.
2.	Spirent Clients disconnecting with GTK rekeying option set in AP	IPQ8074	Customer may see client disconnections on 6Ghz radio in Spirent setup with more than 375 clients during rekey interval. Only a subset of clients would disconnect.
3.	Q6 Crash : cmnos_thread.c:3804 Assertion 0 failed	IPQ8074	Customer may observe crash when we run SBS to DBS switch tests in multiclient scenario or with interruptions (WiFi restart at frequent interval) test with SpirentC50.
4.	Crash : wal_monitor.c:1073 Assertion !(peer_delete_monitor_time_delta >= g_wal_mac_core_ctrl.param.peer_delete_timeout) failed	IPQ5018	Customer may observe this inconsistent crash, when the frequent channel change tests are run on the 5GHz radio in multiclient setups.
5.	Low Memory - Free Mem : 50640KB - UnaccountedMemory : 40.174%	IPQ6018	Customer may observe low memory in IPQ6018, when 512 clients are connected across all bands and run traffic for overnight with monitor mode ON. This issue is an expected behavior, and it is due to Frag memory usage on monitor mode in Spirent multiclient scenarios.
6.	SMMU Error : PCIe AccessViolation	IPQ8074	Customer may observe inconsistent Crash (1 out of 5 iterations with continous traffic tests) on Root AP connected to a Extender STA when hidden SSID enabled on Root.

9.2.2 Performance issues

Sl. No.	Title	Target	Customer Impact
1.	Observed throughput drop of ~6% for 5GHz and ~8% deviation for 6GHz compared from previous CS(12.1) release with multiple Spirent clients	IPQ8074	

Sl. No.	Title	Target	Customer Impact
2.	Observed throughput drop of ~10-15% when compared from previous CS(12.1) release with multiple Spirent clients	IPQ9574	Customer would observe upto ~15% deviation in 200(Spirent) client UDP DL Throughput test from previous CS(12.1) release
3.	Observing low tput of ~20% deviated from Standalone throughput when Wifi change/channel change done during Green AP enabled mode in 5G HE80 TCP DL/UL.	IPQ8074	Customer would observe upto ~20% deviated from Standalone throughput when Wifi change/channel change done during Green AP enabled mode in 5G HE80 TCP DL/UL.
4.	Observing High Delay Factor with spike upto ~90ms in 2Clients at certain instances (STA hidden node) VoW test scenarios.	IPQ9574	Customer will notice high Delay Factor of upto ~90ms in 2Clients at certain instances (STA hidden node) VoW test scenarios.
5.	Observing failure in MLR(MediaLossRate) with more than 1% deviation if Unicast RTP Video traffic is tested with any simulated tools like Veriwave,etc.	IPQ9574	Customer would observe failure in MLR(MediaLossRate) with more than 1% deviation if Unicast RTP Video traffic is tested with any simulated tools like Veriwave,etc. Otherwise, the impact is not seen with video streaming applications

9.2.3 Functional issues

Sl. No.	Title	Target	Customer Impact
1.	Enabling AP VAP using "ap_set_wireless,APUT,radio,on" sigma_dut CAPI command is not working as part of MBO Certification	IPQ9574, IPQ8074, IPQ6018, and IPQ5018	Customer observe Enabling AP VAP using "radio on" sigma_dut CAPI command is not working as part of MBO Certification

9.3 IPQ5322.ILQ.12.2 – IPQ5322 + QCN9160

9.3.1 Stability issues

Sl. No.	Title	Feature Area	Impact
1.	Q6 Crash - M3 Assert in Core : PhyA0 ErrEvnt: MSGE_ERR_MACTX_CBF_PER_USR_TLV_TAG ErrMsg: 240 (fatal_msg_e)	MultiClient	Impact: **High** Customer will observe crash with multiple Clients connected on all bands and running DL/UL real world traffic

Sl. No.	Title	Feature Area	Impact
	Q6 Crash : platform_allocram.c:142 Assertion FALSE failed		Impact: **High** Customer will observe Crash while running 16K flows with Spirent on connecting clients across the bands(240 Clients on 2G,256 Clients on 5G/6G)
	Q6 Crash : cmnos_assert.c:242 PHY0IU:		Impact: **High** Customer will observe crash with multiple Clients connected on all bands and running DL/UL real world traffic
2.	APSS Crash : Kernel Panic : ModuleOrFile = monitor PC = dp_rx_mon_process_tlv_status+0x15c/0xdf8 LR = dp_rx_mon_process_tlv_status+0x13c/0xdf8	FW Recovery	Impact: **High** Customer will observe crash while doing FW recovery test every 600sec in loop after 4 hours
3.	APSS Crash : Host Asserted : FileName = dp_peer.c FunctionName = dp_peer_unlink_ast_entry at LineNo: 1953	MLO	Impact: **Low** Customer will observe crash with Multinode MLO usecase with clients connected on all bands (11be/11ax/11ac) and doing interruptions like Channel change, BW change

9.3.2 Performance issues

Sl. No.	Title	Feature Area	Impact
1.	Observed upto 21% Throughput drop in HE80 and upto 20% deviation and Negative gain in HE160/EHT160/EHT320 in UL MU-MIMO.	MU MIMO	Impact: **High** Customer will observe the low throughput of upto 20% in 80MHz and negative gain in 160MHz UL MU-MIMO 5G/6G(11ax/11be) 2+2 client test
2.	Video hang/Glitch is noticed for long time/sometimes not recovered during channel switching(station reconnect fails sometimes) Expected: No Hang/glitch Observed: Glitch/Hang noticed for long time sometime failed to recover.	VoW	Impact: **High** Customer will notice hang/glitch for long time along with station disconnects in 5G Unicast/Multicast VoW tests during Channel switch.
3.	Observing low throughput dip of 18% across the traffic run from the KPI commit while pumping traffic from AP with 2.5Gig+10Gig combination to STA with 10Gig [Observed/KPI : UDP_Dl: 5954/7200]	SU	Impact: **High** Customer will observe Throughput Fluctuations upto ~18% from expectations while using Multiple ethernet ports(2.5G + 10G) for TBTC tests
	Observing ~9% Throughput deviation against expectations in TCP BiDi in 6G EHT320 DS mode [Expected/Observed: 6738/7400]		Impact: **High** Customer will observe ~9% Throughput deviation against expectations in TCP BiDi in 6G EHT320 DS mode

Sl. No.	Title	Feature Area	Impact
4.	Observing low Throughput of 45 ~ 8 % deveation against expectations [Expected/observed: [6G_EHT320: UDP_DL: 8500/7861, TCP-DL: 7400/4120,][5G_EHT160: UDP_DL:4800/2521 , TCP-DL:4200/2248]]	MU MIMO	Impact: **High** Customer would observe upto ~45% low throughput against expectations in DL MU when both AP and Sta are used as IPQ5322 AP
5.	Throughput not meeting expectations in UDP UL, TCP UL [Expected\Observed: 1958/1605(82%), TCP UL:1778/1038(58%)]	MLO	Impact: **High** Customer will observe throughput deviation upto 18% in UDP UL and 42% in TCP UL against expectations in eMLSR 5G160+6G160 mode
	Throughput not meeting the expectations in UDP DL, TCP DL, UDP and TCP BiDi [Expected\Observer: 1958/1723(88%), TCP UL:1778/1266(71%)]		Impact: **High** Customer will observe throughput deviation upto 12% in TCP/UDP DL & BiDi and 30% deviation in TCP UL against expectations in eMLSR 5G160+6G160 mode
6.	Observed low throughput of TBTC_TCP_DL when compared with the TBTC_TCP_UL [TBTC_TCP_DL: 7732, TBTC_TCP_UL: 9507]	SU	Impact: **High** Customer will observe low DL throughput of ~20% against UL/BiDi throughput in TBTC TCP mode
7.	Random Video Glitch/Hang is noticed while Channel Scanning(CBS) and Switching Expected: No Glitch Observed: Glitch/Hangis noticed atleast once in 2 mins	VoW	Impact: **High** Customer will notice random Video glitch/Hang in at least once in 2 Mins in 5G Unicast/Multicast Vow tests while Channel Scanning(CBS) and switching.
8.	Observed 6~14% deviation from the CS expectations. [Cabled_Results: Observed/expectations: [2G+6G: UDP-DL:6859/7200, UDP-UL: 6789/7200, TCP-DL:5628/6400, TCP_UL:5594/6400] [5G+6G: UDP-DL: 7219/7200, UDP-UL:7089/7200, TCP_DL:5773/6400, TCP-UL: 5521/6400]]	MLO	Impact: **High** Customer will observe low DL & UL throughput of upto 14% against expectations in MLO 2G(EHT40)+6G(EHT320) and 5G(EHT160)+6G(EHt320) in SFE modes. [Throughput met the expectations DS mode]
9.	observed low throughput in short - mid range with MacBook Pro 11ax compared to competitor AP TPlink. [MacBook Pro 11ax Low Tput details : RDP468-TCPUL-94 Mbps;TPlink-TCPUL-121 Mbps@12dB]	SU	Impact: **High** Customer will observe 10-20% low TCP UL throughput in short-Far range with MacBook Pro 11ax compared to competitor AP
10.	Low Performance by 25-30 Mbps in UDPUL seen with Intel clients in OOB when compared with Competitor AP Huwaei AX3Pro. [RDP441_OOB -UDP_UL: 103 Mbps][Huwaei_AX3Pro-UDP_UL: 125 Mbps]	MU	Impact: **High** Customer will observe 25-30MBPS Low Throughput in UDP UL and DL in multiclient environment when 40 clients are connected on IPQ5322 2GHz compared to competitor AP
11.	CCK spur-channel sensitivity off from KPI 1 ~1.5dB	RXS	Impact: **Low** Customer will observe CCK spur-channel sensitivity off from KPI 1 ~1.5dB

Sl. No.	Title	Feature Area	Impact
12.	Random bump could be found during 11AX BW40 waterfall test for 2.4G band, this issue can be fixed with SBA.	WF	Impact: **Low** Customer will observe Random bump during 11AX BW40 waterfall test for 2.4G band

9.3.3 QCN9160 specific issues

SN	Title	Feature Area	Impact
1.	Band edge feature Garbage offset values printed in 2GHz QCN9160 scan radio	QCN9160 Scan Radio	Impact: **Medium** Customer observe Garbage offset value for band age feature on 2GHz QCN9160 scan radio

10 Known limitations

1. On QCN6122: 15~20ms high latency seen with HSP client in 6GHz due to new client driver limitation.
2. Traffic stall is observed in 10 out of 20 iterations with more than 16 clients due to IntelAX210 client limitation.
3. Zero gain in 6GHz observed for TXBF 6GHz 11AXAHE160 mode 2x2 to 1x1 due to LPI limitation.
4. UL-MU-MIMO throughput has ~12% deviation with IntelAX210 client due to client limitation when compared to HSP.
5. Multi-user SU\OFDMA throughput with AFC (SP mode) is lower than LPI mode with HSP STAs (Max of 35%) (HSP STA limitation).
6. Unicast video test would have packet loss in some iterations with VeriWave test with 5GHz / 2GHz 11AC and 11n clients 4ss test case able to have zero packet loss till 5clients, whereas the expectation is to meet >=9 clients. Ixia requires 3-5dB better EVM compared to IEEE spec to meet Peak throughput and other SLA. QCA RDs are positioned for competitiveness with higher target powers.
7. Repeater is getting associated in 2nd attempt after channel switch on root.
8. AFC server will reject very low height values in location config file.
9. The AP will associate in 11ac/11ng mode instead of 11ax mode whenever vap_diff_mode = 0 if station is not issuing new scan and follows old entry.
10. QWRAP Proxy STA creation failed during the repeater client reconnection when hidden SSID enabled on the root.
11. During CPU Core-0 100% use with control and data traffic (SFE accelerated, 74B packets), control packets such as arp/protocol handshakes would not resume after an interface restart/Firewall stop.
12. IOT stations gets disconnected when raw mode is configured with rekey interval of less than 10mins.
13. iOS clients are not selecting fast transmission authentication for association with WPA3+11r configured AP.
14. Repeater tries to ping GW of old Controller (brought down) even though it has switched its connection to controller-2 with same credentials.
15. hwmode change needs ezmesh restart, else it will fall back to pre-configured hwmode after channel selection supervision timeout.
16. With opclass feature enabled, mixed hwmode support/ Advanced CAC/ Co-ordinated CAC are not supported.
17. Packet size greater than 1500 should be used to avoid random toggling between MU and SU signal in Dl-OFDMA tests in 20Mhz BW with 2 User in FTM mode.

18. Customer must follow below sequence to bring up Wi-Fi interfaces post Wi-Fi restart on IPQ9574 chip. ECM Stop > Wi-Fi > ECM Start.
19. In Spirent C-50 test bed, for 1 Gbps traffic with 1500 clients & 16 VAPs + 1 Monitor VAP, IPQ9574 needs 1.1 GB memory to pass endurance (100 MB more than IPQ8074A)
20. Devmem not enabled by default on 256LM profile on QCN6122.
21. KASAN is not supported on 256M profiles. 1 GB DDR is needed for supporting the 512M profiles.
22. The support of 256 multicast groups only applies to the WLAN module. The maximum multicast group size supported by the switch device can vary from 16 to 32 depending on different SoCs. A switch can only populate up to its maximum number of groups.
23. Q6 crashes might be seen during bulk association/dissociation of multiple clients during SBS to DBS switch tests on HKv2 RDPs.
24. DHCP IP issues will be seen when both ipaddr and proto fields in network file are set to 'dhcp'.
25. Q6 crashes might be seen during channel change tests with IPQ50xx RDPs in multiclient scenarios. Corner case handled in QCN92xx platforms.
26. TKIP security feature is not supported for 6G radio and should not be configured. Hence Auth types which has TKIP security is not supported for 6G radio.
27. To retrieve caldata accurately, after calibration Customer need to delete caldata.bin prior to reboot.
28. Traffic Failure seen whenever Intel station connected with WFA driver version-22.60.0.75 to non-TxVAP when AP enabled with MU-RTS.
29. In Easymesh configuration, during credential cloning if the credential is updated on a per VAP basis, If we change the cred on the txvaps on 6ghz radio and bring up the updated config, the non tx vaps should be brought up manually. Else the non tx vaps will be down and this will lead to ezmesh daemon kill as all vaps are not up.
30. Out of memory call trace might be observed during large file (<1GB) transfers using Samba4 application with Linux client and not with any windows client due to the high memory consumption limitation of ntfs-3g driver which can be workaround using disable caching with iozone - I command.
31. Legacy Security Auth type (TKIP-3, 4, 5, 7, 8, 10) will not be supported with 6GHz for Easymesh.
32. Outdoor VLP Repeater wouldn't associate to Indoor SP Root which is a known limitation due to design implementation.
33. ~18% regression would be observed in Spirent 200 client Tput test compared to 12.1 CS which is known limitation due to the real client throughput optimization.
34. Bond Interface not able to associate to the bridge when Vlan tagged interfaces are slaves.
35. Traffic on Bridge + VLAN interface will be forwarded through host data path instead of NSS/SFE.
36. NTFS file sharing using Samba4 ipk with filesystem caching enabled caused a call trace. Use -I along with iozone to disable file system caching.
37. SAMBA4 package has a performance degradation compared to SAMBA3.6 ipk, new KPI defined (240Mbps in Read function)

38. Decap of CPU based tunnels. Since the outer header for all tunnel packets will be the same in the WAN DL direction, all packets will be routed to the same ARM core based on the PPE hash – thus leading to asymmetric performance.
39. Fluctuation in Peak throughput measurement may be observed rarely/inconsistently with 3 -Core RPS settings in an OOKLA speed test which is a known limitation. For better consistency of the speedtest result, it's recommended that customer uses 4 cores and >4 (recommended 8) threadCount in ookla configuration.
40. Performance acceleration wouldn't happen in IPv6 DL(WAN-LAN) traffic with same subnet scenario due to known limitation of NDP daemon does not support on RMNET.
41. Bridge traffic acceleration (PPE/SFE) failure when DNAT rule is pre-configured for the same LAN-Host as ECM/SFE Does not support this.
42. L2TP(L2 tunneling protocol) session over PPPoE WAN will disconnect as this is a limitation (not supported) for this release. But still L2TP over WAN will work.
43. The support for offload bridging through two vlans on same interface in PPE not supported.
44. Multiple VLAN on same physical interface is not supported in PPE mode.
45. APUT in idle state, configuring MTU<1280B on Wan or Wan6 causes CPU utilization upto 50%.
46. LAG: Linux bonding driver configuration supported in fast path through the bonding driver sysfs IF:
    a. Mode: 802.3ad (4): xmit_hash_policy: layer2 (0), layer2+3 (2)
    b. Mode: Balance-xor (2): xmit_hash_policy: layer2 (0), layer3+4 (1), layer2+3 (2)
47. Only iPv4/iPv6 + TCP/UDP flows are accelerated through fast path.
48. Tunnel support (6RD, DS-Lite) expects at least the initial few packets in the LAN > WAN direction to push the base fast path rule; further flows for these tunnels can be initiated on either the WAN or LAN network.
49. DHCP relay over PPPoE is not supported.
50. SAMBA is not turned on by default in this release and must be enabled manually.
51. L2TPv2 supported for acceleration. L2TPv3 not supported for acceleration.
52. Limitations specific to Networking on HK/CP/MP SPs.
    a. Flows that require ALG support are not accelerated by NSS (except for TFTP (port 69))
    b. Interfaces configured with NSS qdiscs must have a leaf node configured as the default node for enqueue (using the set_default qdisc parameter); if qdisc structures are created without a default, no packets are transmitted; this includes management packets such as ARP.
    c. PPPoE-relay packets in NSS is not supported.
    d. HNAT on the s17c switch will be supported only on the 4.4 kernel and not on 5.4 kernel.
    e. IGMPv3 and MLDv2 traffic over PPPoE WAN interface is not accelerated.
    f. Qualcomm provides a software implementation for IGMP/MLD snooping.
    g. Inline IPsec flows deviate from current NSS behavior for upstream flows; during NSS iPsec acceleration, fragmentation (when required) is performed pre-encapsulation while inline iPsec (hardware) acceleration fragmentation is done post encapsulation
    h. EDMA can fail to queue the packets to eth_rx; the eth_rx drop counter increments even though EDMA holds on to these packets and tries queueing later

i. NSS connection manager expects to see at least one packet in outgoing direction to push tunnel 3/5-Tuple offload rule

j. L2TP tunnel must brought up and down explicitly using ifup and ifdown commands.

k. If LCP timeout or AVP packets timeout occurs between link plugout and plugin, L2TP tunnel will go down and must be brought up using ifup command.

l. Only iPv4 encapsulation is supported in L2TP for outer encapsulation. However, iPv4/iPv6 is supported as inner payloads

m. Tunnel Interface[Virtual] on NAPA phy[WAN] remains in UP state even if physical interface is down [ifconfig ethx down].

10.1 IPQ9574.ILQ.12.2

1. On IPQ9574-2G: 3~27% deviated throughput in MU-MIMO multiclient tests (4 to 100 clients) when compared to IPQ8074A due to IPQ9574 platform limitation.
2. User cannot change from 11be to any other non-11be mode vaps when MLO is configured. Same works fine without MLO
3. Customer May see higher boot time with Single VAP - SLO configuration (when compared to Single VAP - Non SLO configuration)
4. VAP & MLD will be down if addition of existing interface to MLD is tried.
5. Customer wont be able to use “vap priority” on wds sta in MLO Mode
6. AFC : Outdoor enabled VLP capable repeater can't associate with Indoor SP ROOT
7. Customers using RCC mode will not be able to capture the AP Beacons while oeprating in 320MHz
8. VAP is not coming up when we use vap_diff_mode=1 in SLO / MLO mode
9. When the iGen is running in the adjacent channel and APUT is operating in CH 149, Customer will see low TPUT on WKK at mid- to far range compared to the competing AP or Pine
10. Free Memory is below the customer requirement of 400MB in Bells Waikiki 433
11. Low throughput difference of upto ~15%-~20% is seen in in IPQ9574.ILQ.12.2 when compared legacy in RVR Far Range with 2dBm switch
12. CSA command is not working on quarter rate 4.9Ghz
13. Ping takes ~1 minute after CSA on 4.9Ghz Half Rate
14. Observed throughput degradation of 100 Mbps in OTA SLO 80MHz 5GHz OOB Mode with mix of 10 Intel AX Clients
15. EHT capabilities information is not showing in EazyMesh dataElement file
16. When GPS is not reachable, AP fails to update location parameters from skyhook
17. Few Latency test case couldn't meet the SLA i.e avg Latency should be <=50m
18. In MultiClient OTA MU SLO Short Range Test , More than 50% Throughput Imbalance observed from average throughput for more than 40% of clients allocated within the same range for TCP UL
19. Observed throughput drop of ~37% on 1518 bytes Frame Size in 6GHz Spirent MultiClient UDP UL

20. Observing more than 200Mbps low UDP DL throughput in ETSI CH40 compared to FCC CH40 with Intel AX210 client.n peak throughput region both ETSI and FCC cases are able to sustain peak rates but we are seeing around 60Mbps lower throughput in ETSI case when compared to FCC case, this is an expected behaviour which is mainly due to the 2ms max PPDU duration limit in ETSI case.
21. Tx Bytes stats on the net dev interface will not be updated in DS mode, only Txpackets will show the valid counts.
22. CPU & Performance of PPPOE bridge Traffic over WLAN with DS mode will be impacted since the offload failed (Since PPoE pkts are not supported in FSE).
23. PPE failed to offload MAPT unidi TCP multiflow [10 flows] traffic hence impact in performance degradation when they try to send multiple TCP flows.
24. With ECM mode "auto" (ppe-sfe), QoS traffic would failed. User is advised to use Global ECM mode set to "sfe" for QDISC flows to fallback from ppe to sfe.
25. With default fq_codel multicore QDISC on bridge slave physical Lan interface causes QDISC traffic failure on br-lan.
26. In the presence of multicore fq_codel qdisc on physical interface and creating qdisc for QoS on bridge br-lan would lead to QoS feature unusable. User is recommended to create QDISC on the phy interfaces.
27. In the presence of logical interfaces [VxLAN, Bond Interfaces] and VLAN tagged configurations the traffic would not get accelerated with L2 payload[1496B-1500B].User is recommended to use L2 payload upto 4bytes lesser than interface MTU [1500B] or to reconfigure the MTU to 1504B.
28. When Same source and destination port number used over GRE Tunnel SFE data path throughput degradation is expected.

10.2 IPQ5322.ILQ.12.2

1. Low throughput with 8 User OFDMA is expected due to limitation of 4 User OFDMA only for IPQ5322 2GHz.
2. **QCN9160 Scan Radio:** Customer Use cases cannot be validated with the current RDP441 configuration, and we would be able to validate only the standalone QCN9160 Scan Radio function. 5GHz WKK Radio is not supported, and MLO is disabled on this internal configuration
3. KASAN Build need more than 1GB DDR to incorporate multiclient with multiple flows .
4. encap_type change is not supported in Run time. It needs to be change through UCI
5. Whenever there is burst of small packet traffic sent from AP , CPU is reaching to 100% and due that it not processing any Wi-Fi commands
6. Minidump Feature is not supported in Kasan Builds. Minidump should be disabled with KASAN build.
7. FW Recovery Mode2 is not supported in IPQ5322 platform for CS.
8. M+W needs more than 1GB DDR when multiple flows connected with Spirent max client.
9. RAW mode with Direct Switch enabled is not supported.
10. The limitation in RDP441 due to 16-bit CBT where low throughput is expected with a lower frame size.
11. For IPQ5322 2GHz, OFDMA is supported only for 4 users and OFDMA will not happen beyond 4 users.

12. IPQ5322 has duty cycle limitation and associated overheads like 16-bit DDR, lower number of PPE internal SRAM buffers, and some system NOC clocks running at lower speeds which results in low throughput
13. Comparing short packets KPI numbers between RDP 441 and RDP 433 is not valid due to capability retractions in IPQ5322 AP.
14. SFE Module on IPQ5322 RDP-441 WAN doesn't support auto negotiation at 1G link speed due to HW Limitation and it always requires 10G partner link
15. SFP port on RDP 441 does not support a 1G link, which may result in low throughput when a 1G Ethernet connection is connected to the SFP port.
16. When same source and destination port number used over GRE Tunnel SFE data path throughput degradation is expected.
17. While running traffic between MHT ports with MC proxy enabled causes packet flooding on WAN Port.
18. With ECM mode "auto" (ppe-sfe), QoS traffic would failed. The user is advised to use Global ECM mode set to "sfe" for QDISC flows to fallback from ppe to sfe.
19. In the presence of multicore fq_codel qdisc on the physical interface and creating qdisc for QoS on bridge br-lan would lead to QoS feature unusable. User is recommended to create QDISC on the phy interfaces.
20. Tx Bytes stats on the net dev interface will not be updated in DS mode, only Txpackets will show the valid counts
21. CPU & Performance of PPPOE bridge Traffic over WLAN with DS mode will be impacted since the offload failed (Since PPPoE pkts are not supported in SFE)
22. PPE failed to offload MAPT unidi TCP multiflow [10 flows] traffic hence impact in performance degradation when they try to send multiple TCP flows
23. Back-to-Back Multiple MHT switch ports between APUTs causes loop. To avoid loop, user recommended to use Peer device with non MHT ports.

11 QDART

11.1 QDART reference documents

See these documents for the use of QDART_Connectivity with this release:

- For 802.11be and 802.11ax chipsets, see:
  - FTM RF Test for 802.11ax WLAN AP Chipsets User Guide (80-YB215-1)
  - *QCN90xx FTM RF Test User Guide* (80-Y9005-20)
  - FTM QMSL APIs for 802.11be WLAN AP Chipsets User Guide (80-33882-5)
  - Wireless LAN Driver Version 12.x for Access Points: PHYRF Tuning Reference Guide (80-19560-3L)
  - FTM RF TEST FOR 802.11BE WLAN AP CHIPSETS USER GUIDE (80-33882-1)
- For legacy chipsets, see:
  - *QCA99xx QDART Connectivity User Guide* (80-Y8050-1)
  - *IPQ4018/IPQ4019/IPQ4028/IPQ4029 RF Test User Guide* (80-Y9700-3)
  - *Use QDART Connectivity to Test QCA99XX /QCA98XX/QCA95XX Application Note* (80-Y8050-46)

  Version of QDART Connectivity from QPM for this release
  - WLAN QSPR Subsystem : 1.0.00041.1
  - WLAN QRCT Module : 4.0.00175.1

Notes for using QDART_Connectivity with this release

- QDART is recommended to be downloaded from QPM
- BDF conversion Python utilities (such as bin2txt.py and txt2bin.py) require Python version 2.7
- From QDART Connectivity v87 and above, as part of implementing PBW < CBW support, rateBW 1 'RateBW_LegacyOFDM' which was used for nonHT DUP transmission would perform 'LegacyOFDM' transmission in corresponding channel bonding frequency (cbstate value) in the channel's maximum BW. Enum 50 'RateBW_NON_HT_DUP40', Enum 51 'RateBW_NON_HT_DUP80' and Enum 52 'RateBW_NON_HT_DUP160' would be used to perform nonHT DUP mode Transmission from the QDART version mentioned. Similarly, for Qcatestcmd and myFTM the rateBW argument needs to be passed with appropriate value for non HT DUP mode.

- List mode configuration:

  Configuration to connect through QUTS is enabled in SEQ_DeviceConfig_WLAN_LP-Combined.xml

  CONNECT=QUTS:192.168.1.1:TIMEOUT_SEC:60:TARGET_TYPE:1

  - WLAN_PHYRFMODE must be set to 7 for IPQ8074/IPQ6018 and 0 for other targets
  - QSEQ_CONNVFS_WLAN_Init-LP-Combined.xml:

BASE_DUT_PHY_ID	0	PHYA0
	1	PHYB
	2	PHYA1
BASE_DUT_Instance	0	WLAN0
	1	WLAN1
	2	WLAN2
BASE_DUT_TX_MeasOffsetMsec		
BASE_DUT_TX_MeasDurationMsec		
BASE_DUT_TX_SegmentDurationMsec		
BASE_DUT_TX_BSSID	Ex: 00:03:7F:44:55:71	
BASE_DUT_TX_TXSTATION	Ex: 00:03:7F:44:55:72	
BASE_DUT_TX_RXSTATION	Ex: 00:03:7F:44:55:73	
BASE_DUT_RX_MeasOffsetMsec		
BASE_DUT_RX_SegmentDurationMsec		
BASE_DUT_RX_NUMPACKETS		
BASE_DUT_RX_MODE	0	Promiscuous
	1	Filter
BASE_DUT_RX_BSSID: BASE_DUT_RX_STAADDR		
BASE_DUT_RX_STAADDR	Ex: 00:00:00:C0:FF:EE	
BASE_DUT_TX_LDPC	0	Disable LDPC
	1	Enable LDPC
BASE_DUT_TX_STBC	0	Disable STBC
	1	Enable STBC
BASE_DUT_TX_PAPRD	0	Disable DPD
	1	Enable DPD

12 FTM calibration and verification KPI

This section defines the KPI time taken for calibration and verification procedure. Below numbers are from the 12.2 CS.

This section captures per radio per band KPI which could be present across multiple RDPs in below data. Per radio per band data should be applicable across different RDPs (i.e RDPs which are not explicitly called out) which have the same radio.

12.1 Factors impacting KPI

After Tx transmission from the DUT, QDART requests the equipment to measure the signal. QDART and the DUT interact via TLV commands (TLV2). QDART interacts with the tester via QTI SCPI. Hence, there can be variations in the timing.

These factors can impact the KPI timing:

- HW configurations of the test machine (where QDART running)
- Load on the test machine, i.e., CPU load, RAM etcetra., and multiple devices connected to it
- Equipment type i.e., vendor specific equipment
- SW version running on the equipment side
- QDART version
- Plugins enabled on QDART
- QDART test tree configurations and settings
- QPST or QUTS software running on the test machine
- QCA tools vs customer tools

Due to these factors, time taken for calibration/verification varies with each setup. However, timing remains comparable when tests are run back-to-back in the same setup with an older release.

The measured time captured in the following section is the time taken for the completion of the entire test tree.

Note: From 12.2 CS Release, List mode data for Cal KPI FR for 802.11ax is not captured.

NOTE: The recommended test tree configuration is provided. Variation in KPIs from run to run can be from 5-10%.

KPI data capturing the time taken (in seconds) for these cases:

- Calibration overall time (XTAL, FPC or OPC, NF, RxGain)
- Calibration FW time (TX: XTAL, FPC or OPC; RX: NF, RxGain)
- Verification Tx time

- Verification Rx time
- Verification firmware (Tx time only) –calculated from the difference b/w request TLVs received and response TLV sent out from firmware
- Verification firmware (Rx time only) –calculated from the difference b/w request TLVs received and response TLV sent out from firmware

RDPs covered

- RDP433 (IPQ9574)
- RDP419 (IPQ8074 2 GHz/5 GHz 4x4, QCN90xx 6 GHz 4x4)
- RDP413 (QCN90xx 2 GHz 4x4, 5 GHz 4x4)
- RDP361 (IPQ6018 2 GHz/5 GHz 2x2)
- RDP432 (IPQ5018 2 GHz 2x2, QCN61xx 5 GHz/6 GHz 2x2)
- RDP418 (IPQ9574 2 GHz 4x4)
- RDP441 (IPQ5322 2 GHz 2x2, QCN92xx 5 GHz/6 GHz 4x4)

Tools version

- QDART Connectivity Version = 1.0.99

12.2 RDP433

Regular mode Test equipment details

- Instrument ID= LitePoint,IQXEL-M2X,IQ1702A4486,1.13.0SCPI Firmware Rev.= QTI_WLAN_SCPI_revision_2018-08-04
- System Ver.= 1.97;
- WLAN Ver.= 2.2;
- Instrument Ver.= 1.13.0, 3.0.3

List mode Test equipment details

- Instrument ID= LitePoint,IQXEL-M2X,IQ1523A2193,1.10.0
- SCPI Firmware Rev.= QTI_WLAN_SCPI_revision_2018-08-04
- System Ver.= 1.97
- WLAN Ver.= 2.2
- Instrument Ver.= 1.10.0, 4.0.4_support_fetch_PPBM_feature

Table 12-2.1 Calibration overall time

Band	Cal type	Combination used	
		Total time W/ CalDB (in Secs)	Total W/O CalDB (in Secs)
2G	FPC + other cals	197.07	204.14
	OPC + other cals	15.45	24.11
5G	FPC + other cals	583.64	603.68
	OPC + other cals	21.14	43.21
6G	FPC + other cals	585.01	588.16
	OPC + other cals	21.47	41.83

Table 12-2.2 Calibration firmware time

Band	Cal type	Combination used			
		Total time W/ CalDB (in Secs)		Total W/O CalDB (in Secs)	
		TX	RX	TX	RX
2G	FPC + other cals	7.31	0.36	10.72	3.82
	OPC + other cals	0.70	0.36	5.89	3.82
5G	FPC + other cals	27.33	0.79	35.72	12.30
	OPC + other cals	1.00	0.79	11.98	12.30
6G	FPC + other cals	42.05	0.83	34.38	11.44
	OPC + other cals	1.10	0.83	10.83	11.44

Table 12-2.3 Verification Tx time: Overall time for Tx verification KPI

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
TX						
2G	Regular FTM	40	12.91	15.04	23.97	26.61
	List Mode	40	6.20	8.82	6.62	13.01
5G	Regular FTM	80 & 160	60.53	69.57	62.56	74.28
	List Mode	80 & 160	17.24	24.00	18.64	29.24
6G	Regular FTM	80 & 160	41.53	51.18	46.64	56.09
	List Mode	80 & 160	17.24	24.00	18.64	29.24

Table 12-2.4 Verification Rx time: Overall time for Rx verification KPI

Band	Mode	BW	W CalDB(in secs)	W/O CalDB(in secs)
RX				
2G	Regular FTM	20	14.31	16.98
	List Mode	20	22.08	32.40
5G	Regular FTM	20 & 80	66.65	79.80
	List Mode	20 & 80	65.52	97.92
6G	Regular FTM	20 & 80	41.87	51.13
	List Mode	20 & 80	60.48	96.48

Table 12-2.5 Verification firmware (Tx time only): Firmware only time for Tx verification KPI

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
Tx						
2G	Regular FTM	40	0.81	3.24	0.44	2.86
	List Mode	40	6.00	8.60	6.40	12.08
5G	Regular FTM	80 & 160	2.25	13.17	0.92	11.87

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
	List Mode	80 & 160	17.04	23.76	18.24	29.04
6G	Regular FTM	80 & 160	2.24	11.78	0.93	10.46
	List Mode	80 & 160	17.04	23.52	18.48	29.04

Table 12-2.6 Verification firmware (Rx time only): Firmware-only time for Rx verification KPI

Band	Mode	BW	W CalDB(in secs)	W/O CalDB(in secs)
2G	Regular FTM	20	0.27	2.57
	List Mode	20	12.00	22.80
5G	Regular FTM	20 & 80	0.83	11.14
	List Mode	20 & 80	36.00	72.00
6G	Regular FTM	20 & 80	0.78	9.89
	List Mode	20 & 80	36.00	72.00

12.3 RDP419

Test equipment details

- Instrument ID= LitePoint,IQXEL-M2X,IQ1702A4486,1.13.0SCPI Firmware Rev.= QTI_WLAN_SCPI_revision_2018-08-04
- System Ver.= 1.97;
- WLAN Ver.= 2.2;
- Instrument Ver.= 1.13.0, 3.0.3

Table 12-3.1 Calibration overall time

Band	Cal type	Combination Used	
		Total time W/ CalDB (in Secs)	Total W/O CalDB (in Secs)
2G	FPC + other cals	190	191

Band	Cal type	Combination Used	
		Total time W/ CalDB (in Secs)	Total W/O CalDB (in Secs)
	OPC + other cals	8	9
5G	FPC + other cals	543.55	546.71
	OPC + other cals	26.00	32.16
6G	FPC + other cals	577.06	571.73
	OPC + other cals	26.00	38.00

Table 12-3.2 Calibration firmware time

Band	Cal type	Total time W/ CalDB (in Secs)		Total W/O CalDB (in Secs)	
		TX	RX	TX	RX
2G	FPC + other cals	6.417	0.75	6.205	1.62
	OPC + other cals	0.637	0.75	0.887	1.62
5G	FPC + other cals	22.53	1.34	18.58	5.84
	OPC + other cals	1.54	1.34	2.978	5.84
6G	FPC + other cals	34.01	0.73	19.44	9.92
	OPC + other cals	1.05	0.73	3.97	9.92

Table 12-3.3 Verification Tx time: Overall time for Tx verification KPI

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
TX						
2G	Regular FTM	40	14	16	14	14
5G	Regular FTM	80	30	30	32	31
6G	Regular FTM	80 & 160	52	49	48	49

Table 12-3.4 Verification Rx time: Overall time for Rx verification KPI

Band	Mode	BW	W CalDB(in secs)	W/O CalDB(in secs)
RX				
2G	Regular FTM	20	33	35
5G	Regular FTM	20 & 80	90	95
6G	Regular FTM	20 & 80	95.09	99.37

Table 12-3.5 Verification firmware (Tx time only): Firmware only time for Tx verification KPI

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
Tx						
2G	Regular FTM	40	0.37	0.55	0.48	0.64
5G	Regular FTM	80	0.83	1.11	0.84	1.47
6G	Regular FTM	80 & 160	3.38	4.78	3.39	5.36

Table 12-3.6 Verification firmware (Rx time only): Firmware-only time for Rx verification KPI

Band	Mode	BW	W CalDB(in secs)	W/O CalDB(in secs)
2G	Regular FTM	20	0.27	0.52
5G	Regular FTM	20 & 80	0.8	1.55
6G	Regular FTM	20 & 80	1.91	3.04

12.4 RDP413

Test equipment details

- Instrument ID= LitePoint,IQXEL-M2X,IQ1702A4486,1.13.0
- SCPI Firmware Rev.= QTI_WLAN_SCPI_revision_2018-08-04
- System Ver.= 1.97
- WLAN Ver.= 2.2
- Instrument Ver.= 1.13.0, 3.0.3

Table 12-4.1 Calibration overall time

Band	Cal type	Combination Used	
		Total time W/ CalDB (in Secs)	Total W/O CalDB (in Secs)
2G	FPC + other cals	225.72	228.61
	OPC + other cals	20.82	27.41
5G	FPC + other cals	567.56	569.43
	OPC + other cals	24.96	38.94

Table 12-4.2 Calibration firmware time

Band	Cal type	Total time W/ CalDB (in Secs)		Total W/O CalDB (in Secs)	
		TX	RX	TX	RX
2G	FPC + other cals	9.81	0.38	8.31	4.92
	OPC + other cals	0.67	0.38	2.83	4.92
5G	FPC + other cals	27.32	0.71	18.12	10.91
	OPC + other cals	0.89	0.71	3.88	10.91

Table 12-4.3 Verification Tx time

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
TX						
2G	Regular FTM	40	14	16	15	17
5G	Regular FTM	80	48	57	49	50

Table 12-4.4 Verification Rx time

Band	Mode	BW	W CalDB(in secs)	W/O CalDB(in secs)
RX				
2G	Regular FTM	20	34	36
5G	Regular FTM	20 & 80	111	134

Table 12-4.5 Verification firmware (Tx time only)

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
Tx						
2G	Regular FTM	40	0.47	0.89	1.78	2.22
5G	Regular FTM	80	1.72	2.83	3.98	5.44

Table 12-4.6 Verification firmware (Rx time only)

Band	Mode	BW	W CalDB (in secs)	W/O CalDB(in secs)
RX				
2G	Regular FTM	20	0.35	1.12
5G	Regular FTM	20 & 80	1.6	3.1

12.5 RDP361

Test equipment details

- Instrument ID= LitePoint,IQXEL-M2X,IQ1702A4486,1.13.0
- SCPI Firmware Rev.= QTI_WLAN_SCPI_revision_2018-08-04
- System Ver.= 1.97
- WLAN Ver.= 2.2
- Instrument Ver.= 1.13.0, 3.0.3

Table 12-5.1 Calibration overall time

Band	Cal type	Combination Used	
		Total time W/ CalDB (in Secs)	Total W/O CalDB (in Secs)
2G	FPC + other cals	90.98	91.61
	OPC + other cals	4.00	5.00
5G	FPC + other cals	238.32	242.49
	OPC + other cals	17.07	17.27

Table 12-5.2 Calibration firmware time

Band	Cal type	Total time W/ CalDB (in Secs)		Total W/O CalDB (in Secs)	
		TX	RX	TX	RX
2G	FPC + other cals	2.62	0.10	2.419	0.63
	OPC + other cals	0.26	0.10	0.423	0.63
5G	FPC + other cals	14.082	0.906	8.582	1.67
	OPC + other cals	1.33	0.91	1.744	1.67

Table 12-5.3 Verification Tx overall time

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
TX						
2G	Regular FTM	40	7	8	9.8	11
5G	Regular FTM	80	29.35	32.92	29	30

Table 12-5.4 Verification Rx overall time

Band	Mode	BW	W CalDB (in secs)	W/O CalDB(in secs)
RX				

Band	Mode	BW	W CalDB (in secs)	W/O CalDB(in secs)
2G	Regular FTM	20	9.11	12.87
5G	Regular FTM	20 & 80	26	22

Table 12-5.5 Verification firmware (Tx time only)

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
Tx						
2G	Regular FTM	40	0.1	0.22	0.2	0.39
5G	Regular FTM	80	0.651	0.921	0.7	1.02

Table 12-5.6 Verification firmware (Rx time only)

Band	Mode	BW	W CalDB (in secs)	W/O CalDB(in secs)
RX				
2G	Regular FTM	20	0.16	0.233
5G	Regular FTM	20 & 80	0.514	0.669

12.6 RDP432

Test equipment details

- Instrument ID= LitePoint,IQXEL-M2X,IQ1702A4486,1.13.0
- SCPI Firmware Rev.= QTI_WLAN_SCPI_revision_2018-08-04
- System Ver.= 1.97
- WLAN Ver.= 2.2
- Instrument Ver.= 1.13.0, 3.0.3

Table 12-6.1 Calibration overall time

Band	Cal type	Combination Used	
		Total time W/ CalDB (in Secs)	Total W/O CalDB (in Secs)
2G	FPC + other cals	100	98
	OPC + other cals	12.8	15.17
5G	FPC + other cals	418	417
	OPC + other cals	15	24
6G	FPC + other cals	501	499
	OPC + other cals	21.27	29.54

Table 12-6.2 Calibration firmware only time

Band	Cal type	Total time W/ CalDB (in Secs)		Total W/O CalDB (in Secs)	
		TX	RX	TX	RX
2G	FPC + other cals	5.48	0.13	5.15	1.54
	OPC + other cals	0.4	0.13	2.3	1.54
5G	FPC + other cals	26.75	0.32	19.2	5.18
	OPC + other cals	0.57	0.32	4.53	5.18
6G	FPC + other cals	34.67	0.27	22.68	4.54
	OPC + other cals	0.94	0.27	4.85	4.54

Table 12-6.3 Verification Tx overall time

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
TX						
2G	Regular FTM	40	8	8	10	8
5G	Regular FTM	80 & 160	27	30	27	30
6G	Regular FTM	80 & 160	28	28	26	28

Table 12-6.4 Verification Rx overall time

Band	Mode	BW	W CalDB(in secs)	W/O CalDB(in secs)
RX				
2G	Regular FTM	20	10	10
5G	Regular FTM	20 & 80	28	29
6G	Regular FTM	20 & 80	30	34

Table 12-6.5 Verification firmware (Tx time only)

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
Tx						
2G	Regular FTM	40	0.13	0.35	0.9	1.13
5G	Regular FTM	80 & 160	1.62	1.98	5.77	7.54
6G	Regular FTM	80 & 160	0.44	1.58	4.9	6.13

Table 12-6.6 Verification firmware (Rx time only)

Band	Mode	BW	W CalDB(in secs)	W/O CalDB(in secs)
2G	Regular FTM	20	0.12	0.32
5G	Regular FTM	20 & 80	0.43	1.51
6G	Regular FTM	20 & 80	0.38	1.38

12.7 RDP418

Test equipment details

- Instrument ID= LitePoint,IQXEL-M2X,IQ1702A4486,1.13.0
- SCPI Firmware Rev.= QTI_WLAN_SCPI_revision_2018-08-04
- System Ver.= 1.97;
- WLAN Ver.= 2.2;
- Instrument Ver.= 1.13.0, 3.0.3

Table 12-7.1 Calibration overall time

Band	Cal type	Combination Used	
		Total time W/ CalDB (in Secs)	Total W/O CalDB (in Secs)
2G	FPC + other cals	232.48	234.15
	OPC + other cals	19.78	21.24

Table 12-7.2 Calibration firmware only time

Band	Cal type	Total time W/ CalDB (in Secs)		Total W/O CalDB (in Secs)	
		TX	RX	TX	RX
2G	FPC + other cals	5.95	0.53	5.34	1.14
	OPC + other cals	0.31	0.522	0.55	1.13

Table 12-7.3 Verification Tx overall time

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
TX						

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
2G	Regular FTM	40	13	12	12	14

Table 12-7.4 Verification Rx overall time

Band	Mode	BW	W CalDB(in secs)	W/O CalDB(in secs)
RX				
2G	Regular FTM	20	29	30

Table 12-7.5 Verification firmware (Tx time only)

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
Tx						
2G	Regular FTM	40	0.23	0.32	0.44	0.53

Table 12-7.6 Verification firmware (Rx time only)

Band	Mode	BW	W CalDB(in secs)	W/O CalDB(in secs)
RX				
2G	Regular FTM	20	0.29	0.43

12.8 RDP441

Regular mode Test equipment details

- Instrument ID= LitePoint,IQXEL-M2X,IQ1702A4486,1.13.0
- SCPI Firmware Rev.= QTI_WLAN_SCPI_revision_2018-08-04
- System Ver.= 1.97
- WLAN Ver.= 2.2
- Instrument Ver.= 1.13.0, 3.0.3

List mode Test equipment details

- Instrument ID= LitePoint,IQXEL-M2X,IQ1523A2193,1.10.0
- SCPI Firmware Rev.= QTI_WLAN_SCPI_revision_2018-08-04
- System Ver.= 1.97
- WLAN Ver.= 2.2
- Instrument Ver.= 1.10.0, 4.0.4_support_fetch_PPBM_feature

Table 12-8.1 Calibration overall time

Band	Cal type	Combination Used	
		Total time W/ CalDB (in Secs)	Total W/O CalDB (in Secs)
2G	FPC + other cals	110.42	110.62
	OPC + other cals	12.80	13.29

Table 12-8.2 Calibration firmware only time

Band	Cal type	Total time W/ CalDB (in Secs)		Total W/O CalDB (in Secs)	
		TX	RX	TX	RX
2G	FPC + other cals	1.84	0.11	1.91	0.35
	OPC + other cals	0.13	0.11	0.48	0.35

Table 12-8.3 Verification Tx overall time

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
TX						
2G	Regular FTM	40	25.57	25.92	27.45	27.35
	List Mode	40	2.96	3.60	6.80	8.40

Table 12-8.4 Verification Rx overall time

Band	Mode	BW	W CalDB(in secs)	W/O CalDB(in secs)
RX				

Band	Mode	BW	W CalDB(in secs)	W/O CalDB(in secs)
2G	Regular FTM	20	6.85	10.49
	List Mode	20	7.56	11.60

Table 12-8.5 Verification firmware (Tx time only)

Band	Mode	BW	W/ CalDB W/O DPD (in secs)	W/O CalDB W/O DPD (in secs)	W/ DPD W/ CalDB (in secs)	W/ DPD W/O calDB (in secs)
Tx						
2G	Regular FTM	40	0.17	0.21	0.17	0.21
	List Mode	40	2.50	3.20	6.40	8.00

Table 12-8.6 Verification firmware (Rx time only)

Band	Mode	BW	W CalDB(in secs)	W/O CalDB(in secs)
RX				
2G	Regular FTM	20	0.03	0.11
	List Mode	20	3.60	7.20

13 FTM and PHYRF tuning

- Due to the current design in the OFDMA scheduling thread and TQM thread especially in IPQ9574, smaller PPDU packets (about 1500 bytes) were unable to schedule MPDU commands quickly, increasing the chances of the STA TID becoming ineligible for next MU sequence and reverting to SU. Thus, the recommended work around is to use bigger PPDUs more than 15000 bytes to avoid this SU/MU toggling issue.
- To get proper AoA Cal Results in IPQ9574 FTM, WlanDut ID must be configured as IPQ9574 and refDesign in ConnectDutUsingQUTS and WlanLoadDut nodes. (In the interim, IPQ8074A was used for IPQ9574). Support to configure IPQ9574 as WlanDut ID and refDesign was added in QDART v93 (1.0.00093). Thus, test trees have been updated in repository with this new value. The new test tree changes are not backward compatible.
- Qualcomm default BDF supports only FCC, ETSI and MKK CTL regions in CTL table due to the current CTL design and available memory to accommodate different modes for Qualcomm Reference designs. As part of Regulatory database version 39, default CTL region for Korea was changed from ETSI to KOREA and default Qualcomm BDF will not have KOREA region in CTL table. BDF should be modified to remove any existing CTL region and replace it with KOREA, to test Korea country for Qualcomm HW Reference designs. Customers can continue to use the documents 80-YB215-8 and 80-19560-3L for more on updating the CTL table using the CTL assistant tool.
- For countries and 6Ghz power modes not formally approved, Qualcomm BDF will not enable the mapping to **Reg domain** and CST/customer can enable it for their testing purpose, as they have been doing it until now.
- Starting with the SPF.12.0 release, the regulatory database is also supported in a separate binary (regdb.bin) apart from what exists in the BDF. The SPF.12.0 release supports the regulatory database in both the BDF as well as the new regdb binary. The changes done in 12.0 are backward compatible as well. As we foresee new regulatory requirements expanding, future regulatory updates will be done only in Regulatory database binary and not in BDF due to memory constraints in BDF. So, the recommendation for customers is to transition to Regulatory database binary from SPF.12.0 onwards. Tools and guide to update BDF/ regdb.bin are available. Refer Regulatory section in Agile document 80-19560-3L for more details.
- RegDB_excel2bin.py tool is compatible with Python version 3.10 and after. RegDB_excel2bin.py is available only on Windows.
- Regulatory database is present outside of BDF for QCN9610 as regdb.bin. The file needs to be included as part of squashfs image.
- In IPQ9574, we see overall FTM calibration time is increased slightly compared to HK09 2G. This is due to FEM gain linearity differences between the AL02 and HK09. HK09 with QPF4288 FEM has an overshoot on the FEM gain as 1.2dB whereas in AL02 with Sky85340 FEM it is 1dB.
- Because FTM does not have provision to select country, ACR setting needs to be manually programmed in BDF to work properly. In MM, the code is updated to fetch ACR value based on country configuration.

- For QCN9074 scan radio, DPD is not applicable. Hence, in the test tree, DPD has been disabled in SetupDutTxDetails test node. In line with these changes, property 'ToolTest' for dpdComplete test node has been updated to True so that this test node would be bypassed.
- For QCN6122 platforms, if DPD calibration fails or DPD is disabled in BDF, the fixed backoff will be applied on target powers. This backoff power values are obtained from BDF fields (targetPowerR2PBackOffTable5G6G). This is applicable only in Mission mode. In case if DPD is disabled in BDF, backoff target power tables must be made zero in BDF.
- Chipsets can sense the environmental noise and adjust the Rx desense level dynamically in mission mode which is known as adaptive noise immunity (ANI). ANI feature can be enabled in FTM mode while running Rx sweep test case or during any Rx testnode execution. Because FTM operations are in a controlled environment, only static ANI mode needs to be used.
- For QCN9610, CalData save/restore is not supported.
- In FTM, Auto power mode(TxPowerAuto) is supported only for 802.11be chipsets. User to ensure parameters configured through QDART are valid for the specific radio interface as the case for other Tx pwrMode.
- For QCN92xx, If AoA phase cal data varies between board to board then its recommended to perform per board calibration in single channel instead of using Golden cal data.
- For QCN92xx, heEhtxxRatesSuSetA4x4 BDF fields are being used to populate target power value for SU + TxBF / OFDMA + TxBF cases and heEhtxxRatesDlOfdma4x4 BDF fields are being used to populate target power values for SU + OL / OFDMA + OL cases

IPQ9574/QCN92xx IBF feature enablement

Template/Value Changes	BDF Field Changes	Description	
Added IBF enablement flag for IPQ9574/QCN92xx	ibfEnable	Since IBF increases the channel switch timing, the calibration by default is not enabled	
		0	IBF disabled (Default)
		1	Enable IBF calibration

14 BDF updates

This section shows a high-level delta for the BDF between SPF.12.1 CS and SPF.12.2 CSU1. For a description of the new fields or a list of changes, see the files as listed.

For information on BDF details, see these documents:

- *FTM RF Test For 802.11ax WLAN AP Chipsets User Guide* (80-YB215-1)
- *Board Data File Utility for 802.11ax WLAN AP Chipsets Application Note* (80-YB215-4)
- FTM RF TEST FOR 802.11BE WLAN AP CHIPSETS USER GUIDE (80-33882-1)
- Board Data File Utility for 802.11be Chipsets User Guide (80-33882-6)

  Note: Default Values in BDF are based on internal RDP designs. Some fields require tuning based on customer design

14.1 IPQ8074

- For a description of the BDF fields, see **BDF_QCA8074_Description.xlsx**
- For a list of changes, see **NvData_QCA8074.xlsm**

Changes	NvSection
Added new BDF fields for extended Dfs Rssi Threshold	COMMON_BDF_HEADER
Added new BDF fields for expanded Heavy Clip Offset feature in CTLE. Added control flags for feature enable/disable too. By default, it is disabled	HEAVYCLIP_CTLMCS_TABLE

Feature details of changes

This section details the listed changes, and the value change section captures the default values compared to previous BDF version.

Template/Value Changes	BDF Field Changes	Description
Added new BDF fields for extended Dfs Rssi Threshold	baseBdfHeader.DfsRssiThresholdExtFlag baseBdfHeader.DfsRssiThreshold40 baseBdfHeader.DfsRssiThreshold80 baseBdfHeader.DfsRssiThreshold160	These fields are added as a extension to DfsRssiThreshold field.
Value Change	baseBdfHeader.radarDetConfig[0].radar_pd_ma_th_high baseBdfHeader.radarDetConfig[0].radar_pd_jump_th	Values updated for improving radar detection only in bdwlan.b294

Template/Value Changes	BDF Field Changes	Description
Template Change	HEAVYCLIP_CTLMCS_TABLE	New NV section to hold CTL offset power values based on rate and packet types

14.2 IPQ5018_QCN6122

- For a description of the new fields, see **BDF_IPQ5018_QCN6122_Description.xlsx**
- For a list of changes, see **NvData_IPQ5018_QCN6122.xlsm**

Changes	NvSection
Added new BDF fields for offsetting CCA threshold to compensate for Spur channel	FREQMODAL
Added new BDF fields for expanded Heavy Clip Offset feature in CTLE. Added control flags for feature enable/disable too. By default, it is disabled	HEAVYCLIP_CTLMCS_TABLE

Feature details of changes

This section details the listed changes, and the value change section captures the default values compared to previous BDF version.

Template/Value Changes	BDF Field Changes	Description
Offsetting CCA threshold to compensate for Spur channel	spurMitRssiCompOffset_5G6G spurMitRssiCompOffset_2G spurMitRssiCompOffset_CCK spurMitRssiCompOffset_Enable	These fields are added to improve throughput in Spur channels.
Template Change	HEAVYCLIP_CTLMCS_TABLE	New NV section to hold CTL offset power values based on rate and packet types
Value change for QCN6122	CTL_TABLES_6G	CTL Power updates for 6G. based on 6G SP Array gain Regrule update

14.3 QCN90xx

- For a description of the new fields, see **BDF_QCN90_Description.xlsx**
- For a list of changes, see **NvData_QCN90xx.xlsm**

Changes	NvSection
Added new BDF fields for expanded Heavy Clip Offset feature in CTLE. Added control flags for feature enable/disable too. By default, it is disabled	HEAVYCLIP_CTLMCS_TABLE
Added new field as a extension to commonBoardFlags	COMMON_BDF_HEADER

Feature details of changes

This section details the listed changes, and the value change section captures the default values compared to previous BDF version.

Template/Value Changes	BDF Field Changes	Description
Template Change	HEAVYCLIP_CTLMCS_TABLE	New NV section to hold CTL offset power values based on rate and packet types
Value change for 6G	CTL_TABLES_6G	CTL Power updates for 6G. based on 6G SP Array gain Regrule update
Template change	COMMON_BDF_HEADER.baseBdfHeader.commonBoardFlags_EXT	New field as extension to commonBoardFlags used to enable/disable specific features

14.4 IPQ6018

- For a description of the new fields, see **BDF_IPQ6018_Description.xlsx**
- For a list of changes, see **NvData_IPQ6018.xlsm**

Changes	NvSection
Added new BDF fields for expanded Heavy Clip Offset feature in CTLE. Added control flags for feature enable/disable too. By default, it is disabled	HEAVYCLIP_CTLMCS_TABLE

Feature details of changes

This section details the listed changes, and the value change section captures the default values compared to previous BDF version.

Template/Value Changes	BDF Field Changes	Description
Template Change	HEAVYCLIP_CTLMCS_TABLE	New NV section to hold CTL offset power values based on rate and packet types

14.5 IPQ9574

- For a description of the BDF fields, see **BDF_IPQ9574_Description.xlsx**
- For a list of changes, see **NvData_IPQ9574.xlsm**

Changes	NvSection
Added new BDF fields for expanded Heavy Clip Offset feature in CTLE. Added control flags for feature enable/disable too. By default, it is disabled	HEAVYCLIP_CTLMCS_TABLE
Added new field as a extension to commonBoardFlags	COMMON_BDF_HEADER

Feature details of changes

This section details the listed changes, and the value change section captures the default values compared to previous BDF version.

Template/Value Changes	BDF Field Changes	Description
Value Change	CAL_DB_SECTION.TempRecalThreshold	Temperature recal threshold has been updated
Template Change	HEAVYCLIP_CTLMCS_TABLE	New NV section to hold CTL offset power values based on rate and packet types
Template change	COMMON_BDF_HEADER.baseBdfHeader.commonBoardFlags_EXT	New field as extension to commonBoardFlags used to enable/disable specific features

14.6 QCN9274

- For a description of the BDF fields, refer to the *QCN9224 BDF Content Check List* (80-33882-9).
- For a list of changes, see **NvData_QCN9244.xlsm**

This section captures the major changes in BDF	NvSection
Added new fields to support 11BE rates in 2G	HWS_TARGET_POWER_TABLES_2G
Added function selection field for different usage of GPIO Added gpioConfigSecondary fields for supporting additional GPIO entries Added primaryGpioBpf field for configuring default RF path selection during bootup	HWS_GPIO
Added nfCalPerChain5G_EXT to support 5G NF cal for PHYA1 in split phy configuration	HWS_NF_CAL_PER_CHAIN_5G
Added nfCalPerChain6G_EXT to support 6G NF cal for PHYA1 in split phy configuration	HWS_NF_CAL_PER_CHAIN_6G
Added fields for DPD to support Non-linear FEMs Added fields for DPD and eDPD configurations	HWS_DPD_CONFIG
Added new BDF files for new RDs	
Updated values for xtalTempComp sub-section	HWS_XTAL_CAL_SECTION
Added new BDF section for PA Droop calibration	HWS_PDC_CONFIG

This section captures the major changes in BDF	NvSection
Added BDF fields to hold DPD backoff power values Added targetPowerR2PBackOffTable to apply target power backoff when DPD fails Update Target power values for 6G subband 1 b0016 Target power updates for OL and BF	HWS_TARGET_POWER_TABLES_xG
Added BDF fields for holding CBW offset, Offchannel bias, Per channel bias	HWS_RTT_DELAY_TABLES_xG
Added EnableDisableBits2 for 2G band edge improvements towards new SRRC Added ibfEnable bit for enabling/disabling IBF Added dpdTgtPwrBackOffControlFlags field	HWS_BDF_ENABLE_DISABLE_FLAGS
Added BDF fields for startChannelxG_EXT/ endChannelxG_EXT for holding values of start and end channel for XOR band RD	HWS_PHY_SECTION
Updated projectId for RDP441	COMMON_BDF_HEADER
Updated Template for AoA Feature	HWS_AOA_CAL_DATA_xG
Updated coexFlags values	HWS_COMMON_BDF_HEADER
Disabled DPD ratemask to MCS0-3 for linear FEMs DPD config updates on 2G_5GL.b101a	HWS_DPD_CONFIG
Added new BDF fields for expanded Heavy Clip Offset feature in CTLE. Added control flags for feature enable/disable too. By default, it is disabled 6G SP Array gain Regrule update	CTL_ENGINE
RTT Digital delay value updates	HWS_RTT_DELAY_TABLES_xG
RSSI Temperature compensation value updates	HWS_BAND_SPECIFIC_xG
Spur changes to improve PER bumps near sensitivity point in spur channels	HWS_SPURMIT_SECTION

14.7 IPQ5322

- For a description of the fields, see **BDF_IPQ5332_QCN6432_Description.xlsx/** IPQ53XX BDF CONTENT CHECKLIST(80-50179-1)

For a list of changes, see NvData_IPQ5332_QCN6432.xlsmSPF 12.2CS is the initial BDF release for IPQ5322 and following captures the updates to BDF between 12.2 CS and 12.2 CSU1 releasesThis section captures the major changes in BDF	NvSection
ctlExceptionEntries value updates for xPA designs	CTL_ENGINE
TPC Golden bin value update	HWS_TPC_DATA_2G
DacBo config updates for xPA designs	HWS_TPC_CONFIG_2G
RxGainCal Golden bin value update for xPA designs	HWS_RX_GAIN_TABLES_2G
NF Cal Golden bin value update for xPA designs	HWS_NF_CAL_PER_CHAIN_2G
Value update for BW40 in xPA designs	HWS_CFR_DATA_SECTION_2G

14.8 QCN9160

- For a description of the fields, see **BDF_QCN9160_Description.xlsx**
- For a list of changes, see **NvData_QCN9160.xlsm**
- SPF12.2 CS is the initial BDF release for QCN9160 and there is no update in BDF between SPF 12.2 CS and 12.2 CSU1 release

15 Additional information on WFA certification

The following workarounds are suggested for passing WFA certification:

802.11ax Cert Test Cases ID	Comments
4.36.1_24 GHz/5 GHz	Customer might face orientation/throughput related issues. To address this, a workaround has been introduced in FW for all platforms. If customer faces low throughput issue with Marvell station, it is recommended to reboot the Marvell test bed station and to reduce the AP Tx power to 5. Once the AP VAP is up, use the command “iwconfig ath0 txpower 5” and continue the test
4.37.1	Customer might face orientation/throughput related issues. To address this, a workaround has been introduced in FW for all platforms. If customer faces low throughput issue with Marvell station, it is recommended to reboot the Marvell test bed station. Recommended RSSI is -40 to -50 dB
4.40.2	If broadcast trigger is not observed with MCS 7 to all 4 users in mixed NSS configuration, skip NSS 1 step and rerun only the mixed NSS steps
4.40.4	If Marvell station hang issue is observed, try with alternate test bed station for passing the test case.
4.41.1	If Marvell station hang issue is observed, try with alternate test bed station for passing the test case.
4.60.1	If Marvell station hang issue is observed, try with alternate test bed station for passing the test case.
General waiver from WFA	Broadcom 75 test bed station may get crash during association. This is randomly seen across 802.11ax test cases. WFA has shared waiver for such issue and alternate testbed station not involved in that particular cases can be replaced instead of broadcom 75.
General waiver from WFA	If TWT (4.56.X series) 92% PPDUs are not within TWT service period, reboot the QCOM test bed station and APUT for retrying the test case
General waiver from WFA	MAUT BH WIFI onboarding issues with Marvell controller after FH VAPs are up; WFA recommended to use testbed Mediatek reference controller
General waiver from WFA	WFA 11ac Certification test, 4.2.56 having a waiver from WFA due to WTS script limitation

802.11ax Cert Test Cases ID	Comments
11ax_R2_4.2.5.3_6G_MBO	Need manual intervention to pass the test case. When radio off CAPI is called out, Wi-Fi down needs to be given. At the time of radio on CAPI called out while executing WFA test suite, Wi-Fi up needs to be issued followed by the below on the fly commands. cfg80211tool athX bintval 100 wifitool athX setUnitTestCmd 0x47 2 42 0 wifitool athX setUnitTestCmd 0x47 2 64 1 wifitool athX setUnitTestCmd 0x48 2 142 1 wifitool athX setUnitTestCmd 0x47 2 141 1 wifitool athX setUnitTestCmd 0x48 2 186 0 cfg80211tool wifiX he_muedca_mode 0 cfg80211tool athX he_ul_ofdma 0 cfg80211tool athX he_dl_ofdma 0 cfg80211tool athX he_ul_mimo 0 wifitool athX setUnitTestCmd 0x47 2 194 0 cfg80211tool athX he_sounding_mode 1 wifitool athX setUnitTestCmd 0x48 2 2 1 wifitool athX setUnitTestCmd 0x47 2 7 1 wifitool athX setUnitTestCmd 0x47 2 47 1 wifitool athX setUnitTestCmd 0x47 2 7 0 wifitool athX setUnitTestCmd 0x48 2 129 0