# RM00297

## Linux Software Reference Manual for NXP Wireless Connectivity

Rev. 1.0 — 24 April 2025 Reference manual

**Document information**

| Information | Content |
|---|---|
| Keywords | connectivity, software features, software architecture, software integration, certification, PSIRT management, vulnerability management, Wi-Fi, Bluetooth, 802.15.4, Matter |
| Abstract | Describes the software features, software architecture, software integration, and certification for wireless connectivity. |

# 1 Introduction

NXP wireless system-on-chips (SoCs) support Wi-Fi, Bluetooth/Bluetooth LE, 802.15.4 radios for connectivity. To function, the devices rely on software components like the host-driver and firmware. This document is focused on Linux OS based software, covering:

- Section "Connectivity software architecture": NXP unified Wi-Fi drivers are the Multi-Chip and Multi-Interface driver where the same Wi-Fi driver can be used to load any wireless SoC firmware over any host interface. For Bluetooth/Bluetooth LE/802.15.4, an open source driver is built into the Linux Kernel. The section provides an overview of the driver architecture.
- Section "What you need" – The wireless SoCs are paired with a host platform. The host platform can be an NXP i.MX platform or any third-party processors. The section explains how to flash the pre-built Linux images and cross-compile the drivers.
- Section "Connectivity software integration": the standalone software package including the firmware, tools, and more, is available for download on github and NXP website. The section describes how to download the firmware onto the wireless SoC to bring up the Wi-Fi, Bluetooth/Bluetooth LE, and 802.15.4 interfaces. Error handling, fatal error recovery, and debugging methods are also explained.
- Section "Connectivity features": the section covers how to utilize Wi-Fi, Bluetooth/Bluetooth LE, and 802.15.4 radios. For instance, how to configure a WPA3 mobile access point (uAP), create an AD2P connection, start a Thread network, and measure the throughput performance.
- Section "Regulatory certifications": regulatory, standard-based, and security certifications are required for the wireless SoCs used in commercial applications. The section details the certification process and NXP pro support services.
- Section "PSIRT and vulnerability management": NXP established a vulnerability management process in case a wireless SoC malfunctions. The section details the process to protect the customer's interests.

## 1.1 Supported products

The following wireless SoCs are supported:

- 88W8801 ref.[21]
- 88W8987 ref.[22]
- 88W8997 ref.[23]
- 88W9098 ref.[24]
- IW416 ref.[25]
- IW610 ref.[26]
- IW611 ref.[27]
- IW612 ref.[28]

For the list of supported features and release notes, refer to ref.[10].

***Note:*** *IW612 and IW610 are tri-radio solutions that support Wi-Fi, Bluetooth/Bluetooth LE, and 802.15.4. 88W8801 is a single radio solution that supports Wi-Fi only. The other wireless SoCs are dual radio solutions that support Wi-Fi and Bluetooth/Bluetooth LE.*

# 2 Connectivity software architecture

## 2.1 Wi-Fi software architecture

### 2.1.1 Wi-Fi driver architecture

This section describes the architecture of the Wi-Fi Linux driver which supports multiple bus interfaces and Wi-Fi devices. The architecture is divided into two modules:

- **MOAL module:** OS-dependent module that includes:
  - The upper interface to the kernel/protocol stack.
  - The lower interface that registers the Wi-Fi driver to the bus driver (SDIO/PCIE) when loaded, so an SDIO/PCIE device can be detected.
  - One SHIM layer to MLAN module.
- **MLAN module:** OS-independent module that includes most of the driver handling and interface to the firmware. The module also enables applicable interface operations based on the card type conveyed by the MOAL module. When customizing or integrating the Wi-Fi driver, you can use the MLAN module "as is" to reduce the porting work.

Figure 1 shows the Wi-Fi driver architecture.

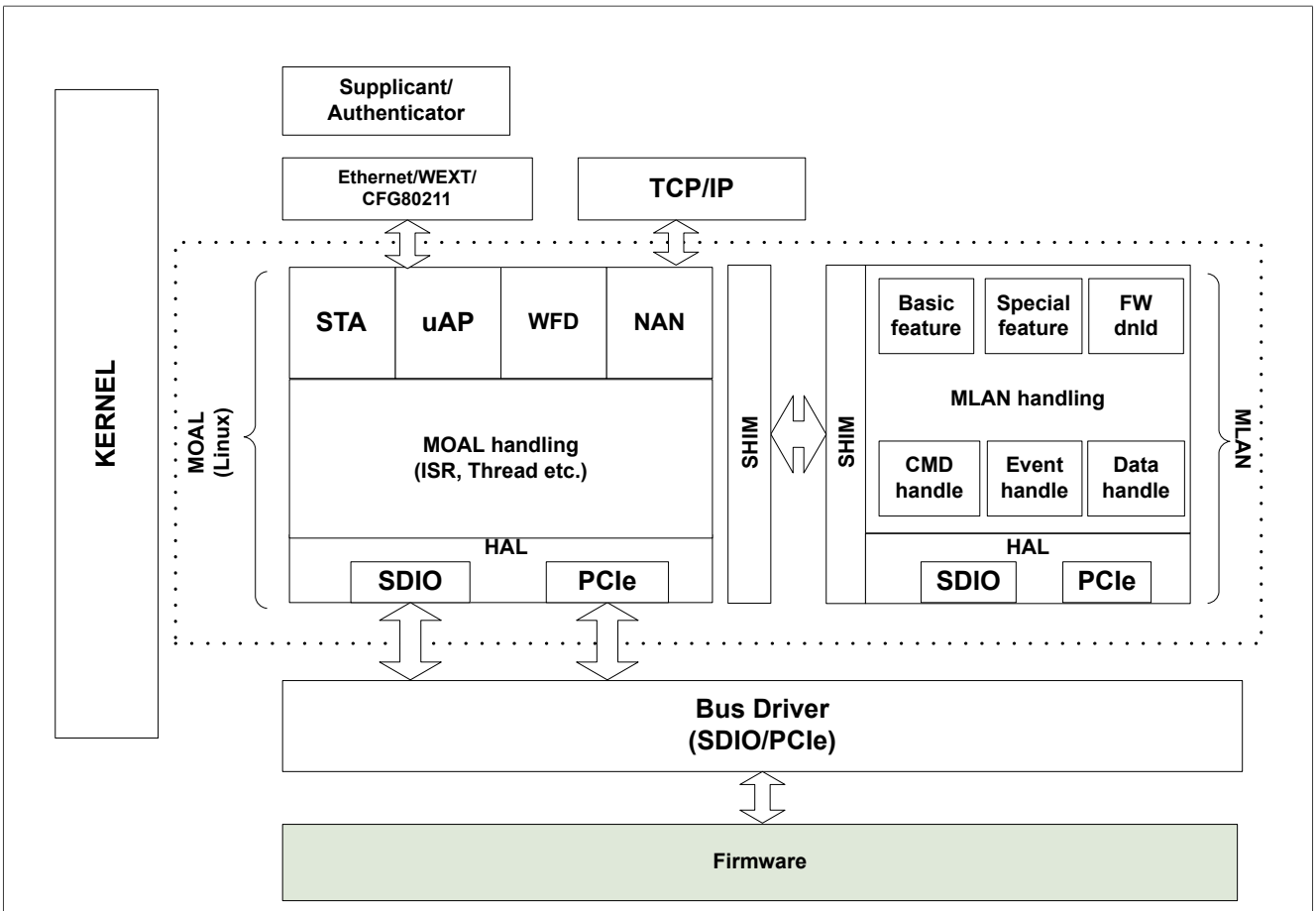



**Figure 1. Wi-Fi driver architecture**

#### 2.1.1.1 MOAL module

**Upper interface to kernel/protocol stack:**

- Ethernet handling: standard ethernet driver module in Linux with similar functionality as an Ethernet NIC card to the kernel. The driver module handles packet transmission and reception as well as the control.
- 802.11 extensions (WE) handling: interface to the wireless extension stack provided by Linux kernel. The interface includes 802.11 specific features like scanning, association, 802.11d, 802.11h, and WMM.
- Cfg80211 handling: interface to cfg80211 stack provided by Linux kernel. This interface is a replacement for WE handling, although it can also work alongside WE handling. Multiple Wi-Fi network interfaces are supported, such as STA, uAP, WFD and NAN. The interface list is created during the driver initialization and can be adjusted according to the device capability.

**Lower interface to the bus driver:**

- Bus driver interfacing: component that provides an interfacing layer between the MOAL module and the low-level bus driver module. The component registers the Wi-Fi driver to the (SDIO/PCIE) bus driver and call the APIs provided by the bus driver to access the registers and data ports.
- Independent operation of SDIO/PCIE: When a device is detected, the driver enables the applicable interface operations for the device interface.

**SHIM layer to MLAN**: used to convert OS-specific APIs to MLAN APIs and vice versa.

#### 2.1.1.2 MLAN module

The MLAN module is used for firmware downloading, command handling, data handling, event handlings, power-save handling, and mux/demux handling, and more.

***Note:*** *The driver or firmware API version up to 18 is supported in MLAN.*

The data sent to or received from the firmware is little-endian based. The driver is configured as little-endian by default.

### 2.1.2 Wi-Fi layer interfaces

The wireless module requires a kernel driver loaded on the host system and a firmware running on NXP-based wireless modules. When the interface bus driver detects the NXP-based wireless module, the MLAN module downloads the firmware binary via the interface adapter. NXP Wi-Fi driver is loaded between the bus driver and the network stack from the cfg80211 subsystem in the kernel. NXP kernel driver includes a set of controls and configurations to communicate with the user space through one of the following interfaces:

- Input/output control (IOCTL)
- Wireless Extension (Wext)
- CFG80211

The IOCTL provides a path to the user space applications *iwconfig* and *iwpriv* whereas cfg80211 interface provides a different path to the user space applications *wpa_supplicant*, *hostapd* and *iw*.

Figure 2 illustrates the Wi-Fi layer interface.

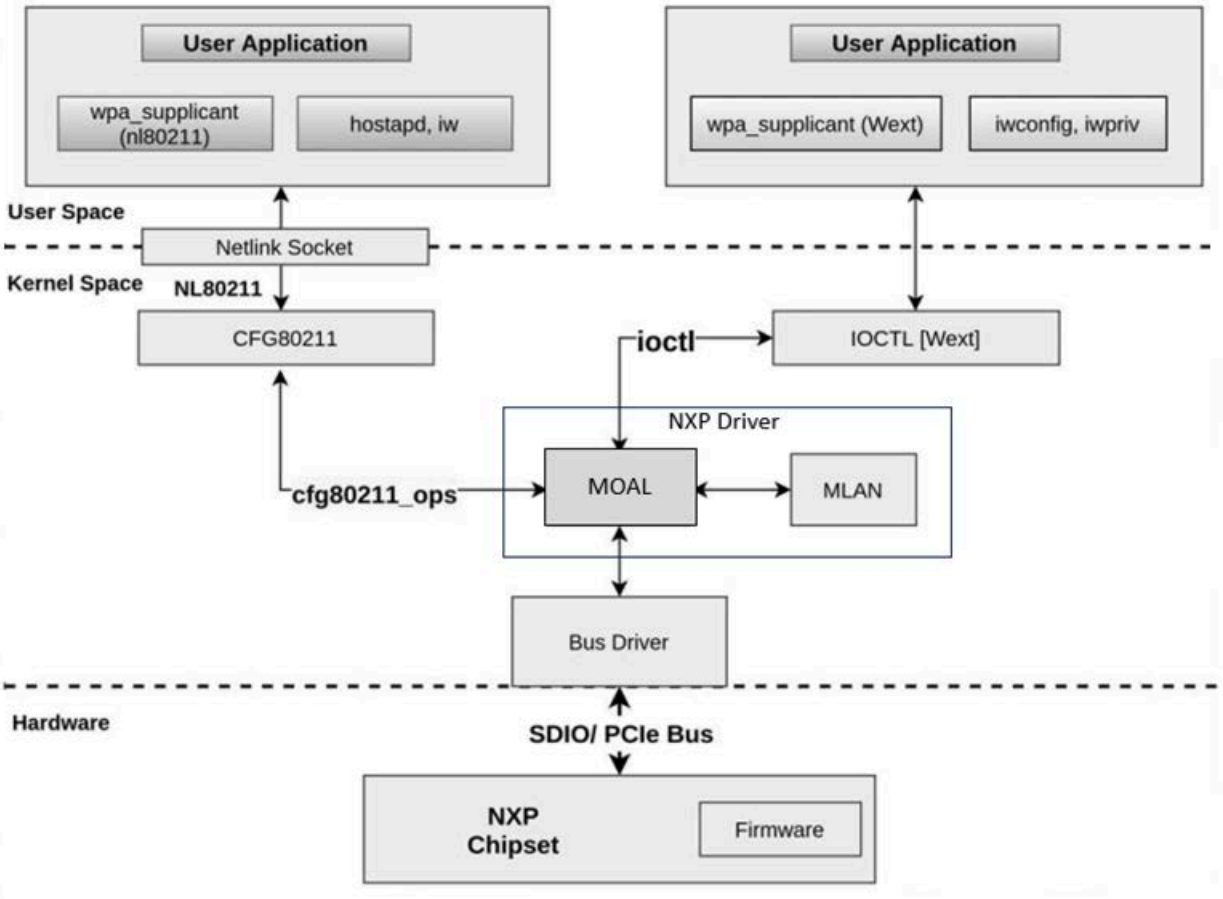



**Figure 2. Wi-Fi layer interface**

## 2.2 Bluetooth software architecture

### 2.2.1 Bluetooth layer interfaces

Figure 3 illustrates the layers between the user applications and the NXP-based Bluetooth module. The NXP-based wireless module requires a kernel driver loaded on the host system and a firmware running on NXP wireless SoC. The Wi-Fi driver loads the combo firmware, or the Bluetooth driver loads the Bluetooth only firmware. The *btnxpuart* driver provides the HCI interface between the firmware and user application.

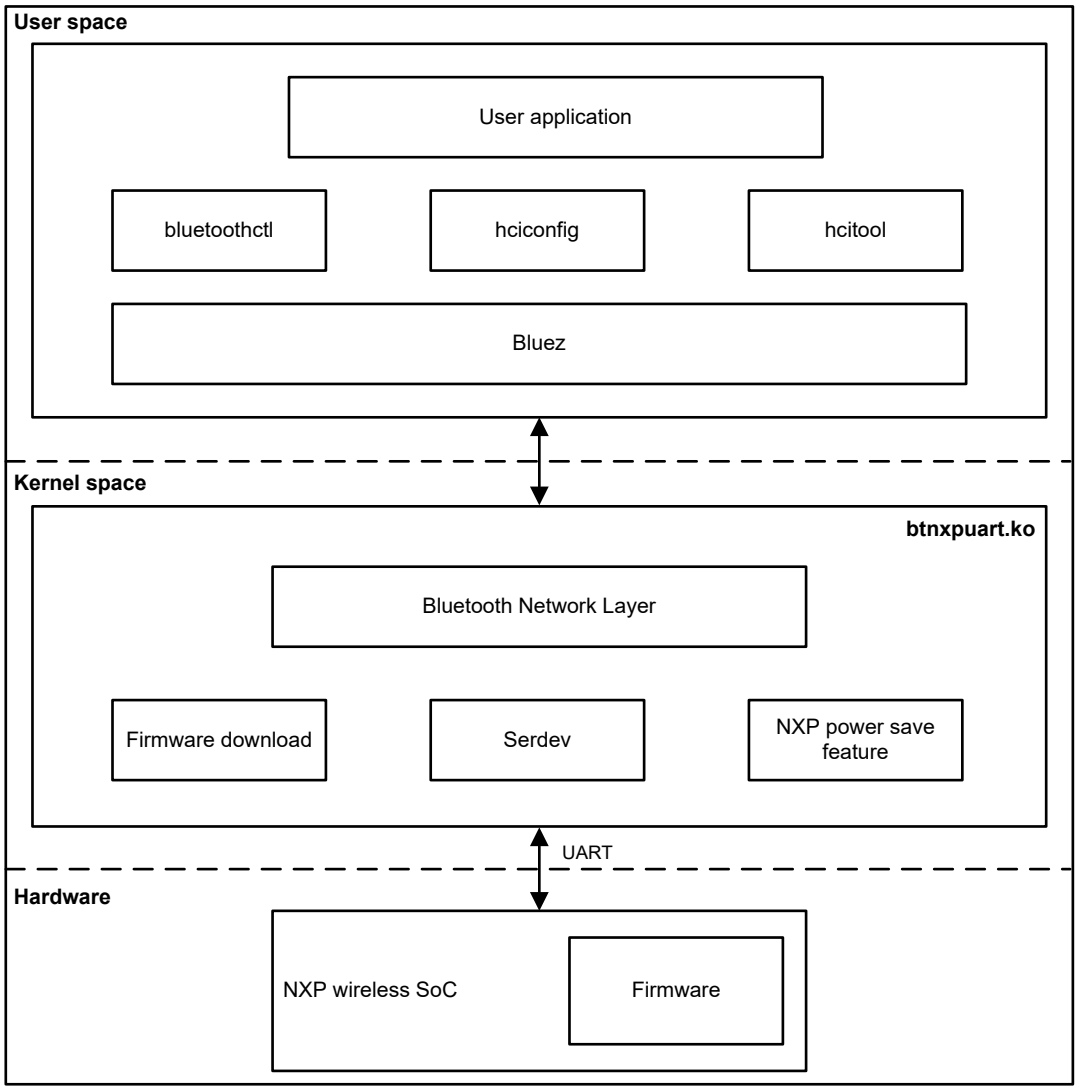



**Figure 3. Bluetooth layer interface**

## 2.3 802.15.4 software architecture

### 2.3.1 OpenThread radio coprocessor (RCP) software architecture

Thread is an IPv6-based networking protocol designed for low-power Internet of Things devices in an IEEE802.15.4-2015 wireless mesh network (ref.[11]). OpenThread released by Google is an open-source implementation of Thread (ref.[31] and ref.[32]).

The 802.15.4 subsystem of the wireless SOC works as controller in OpenThread RCP design as illustrated in Figure 4.

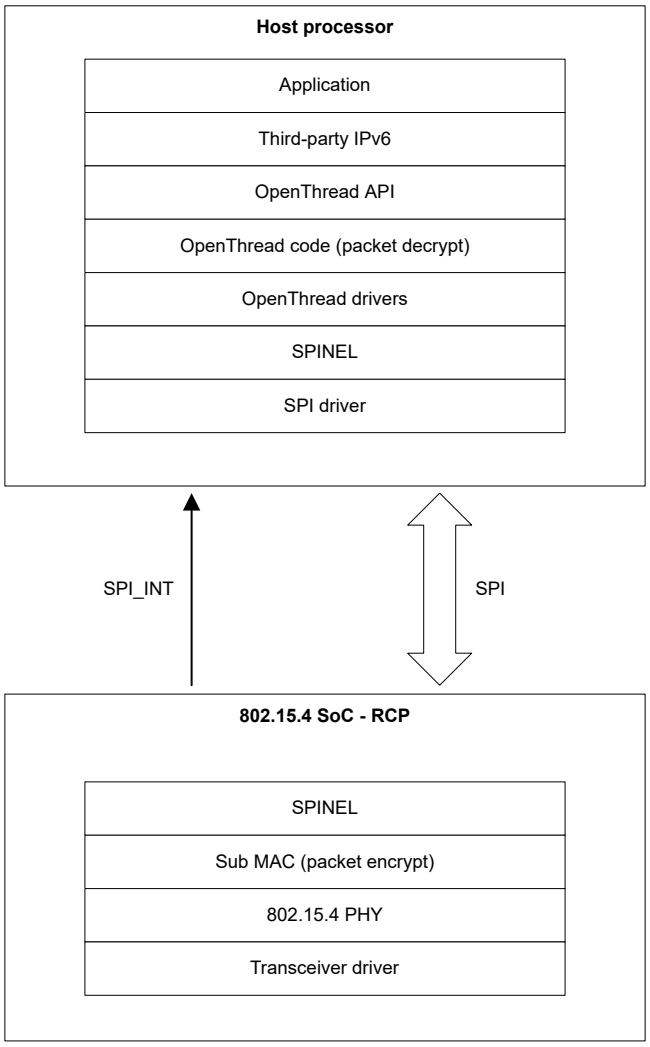



**Figure 4. 802.15.4 software architecture**

#### 2.3.1.1 Host

The OpenThread core runs on the host processor and communicates with the controller via OpenThread Daemon (Ot-daemon) through SPI interface over the Spinel protocol ref.[14].

Ot-daemon main features:

- Used in the radio coprocessor (RCP) design
- OpenThread POSIX build mode that runs OpenThread as a service
- UNIX socket as input and output

Clients can connect to the UNIX socket and communicate using OpenThread CLI as a protocol.

#### 2.3.1.2 Controller

The controller supports:

- Spinel over SPI
- 10 MHz maximum SPI clock speed
- IEEE 802.15.4-2015 sub-MAC and PHY features as required by Thread 1.4
- TX and RX PHY PSDU

### 2.3.1.3 Communication between host and controller

The SPI bus is used for 802.15.4 data communication.

**Table 1. SPI interface related signals for IW612 wireless SoC**

| Signal name | Type | GPIO pin[1] | Description |
|---|---|---|---|
| SPI_FRM | I | 13 | SPI frame input signal |
| SPI_CLK | I | 12 | SPI clock input signal |
| SPI_RXD | I | 14 | SPI receiver input signal |
| SPI_TXD | O | 15 | SPI transmitter output signal |
| SPI_INT | O | 20 | SPI interrupt output signal |

[1] The GPIO pin numbers in the table are specific to IW612 product. To know which GPIO pin number is used for the SPI interface signals, refer to the other product data sheets.

The Host configures SPI_INT pin as interrupt and the wireless SoC asserts SPI_INT when data is available for transfer to the host.

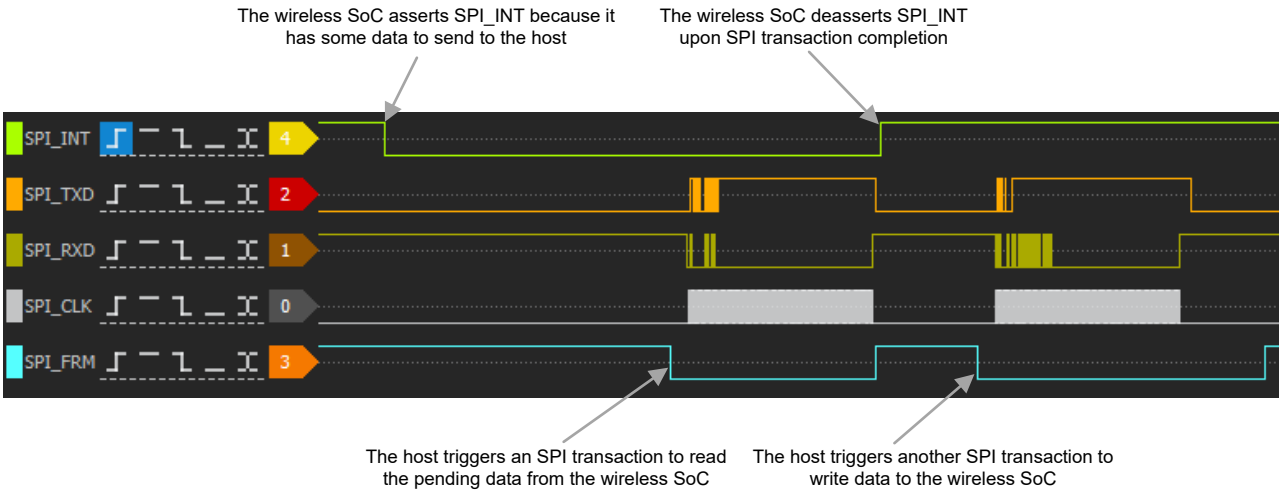



**Figure 5. SPI_INT pin behavior in SPI data communication**

Spinel is a standardized host-controller protocol used for the data communication between the host and controller over SPI interface. Figure 6 shows the Spinel over SPI data format in OpenThread network.

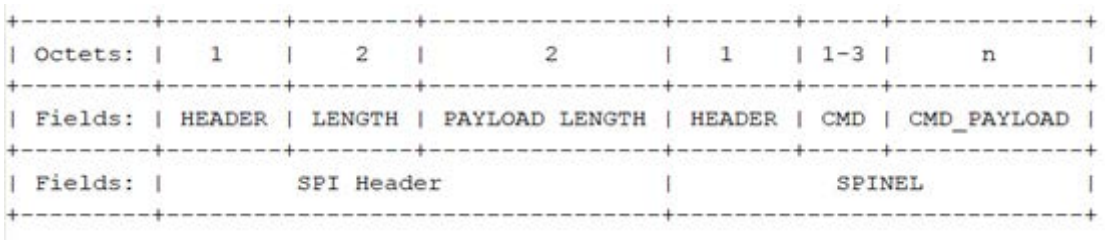



**Figure 6. SPI data format in OpenThread network**

# 3 What you need

## 3.1 Connectivity hardware

For information on NXP-based wireless modules, refer to ref.[18].

Find partner module information on NXP website for the wireless SoCs:

- 88W8801
- 88W8987
- 88W8997
- 88W9098
- IW416
- IW610
- IW611
- IW612

## 3.2 Host platforms

### 3.2.1 NXP i.MX host processors

#### 3.2.1.1 FRDM development board

Refer to ref.[29] and ref.[30].

#### 3.2.1.2 i.MX 8M Quad evaluation kit (EVK)

Refer to the section *i.MX 8M Quad evaluation kit (EVK)* in ref.[18].

#### 3.2.1.3 i.MX 93 EVK

Refer to ref.[17].

### 3.2.2 Integration with i.MX host processors

This section provides the *how to* information on:

- Setting up the Linux OS host.
- Running and configuring a Yocto project ref.[15].
- Generating an image.
- Generating a rootfs.
- Implementing the required changes in the device tree structure to bring up the wireless SoC on the PCIe interface.

#### 3.2.2.1 Set up the build environment

This section describes the steps to set up the host machine to build the Yocto image. The steps provided in this document is tested on Ubuntu 16.04. The recommended minimum Ubuntu version is 16.04 or newer.

##### 3.2.2.1.1 Set up the host

Refer to the steps and specification of the host machine that the board manufacturer provides in the platform specific Yocto build guide. To set up an i.MX host machine, refer to ref.[17], ref.[18], ref.[29], and ref.[30].

##### 3.2.2.1.2 Install *Repo* tool

*Repo* is a tool built on top of Git that makes it easier to manage projects that contain multiple repositories, which do not need to be on the same server.

Perform the following steps to install the *repo* utility:

- Create a bin directory in the home directory

```
$ mkdir ~/bin (this step may not be needed if the bin directory already exists)
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```

- Add the following line to the *.bashrc* file to ensure that the *~/bin* directory is in your PATH variable.

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

#### 3.2.2.1.3 Set up Yocto project

During the project setup, the layers are downloaded to the sources directory; *repo init* repository is used to set the recipes, and `repo sync` command is used to synchronize the repository with the latest code.

The board manufacturer uses the following options for the `repo init` command:

- `-u` for the manifest repository location
- `-b` for the manifest branch
- `-m` for the initial manifest file

The following steps are for i.MX8QM MEK board.

Commands to setup the git:

```
$ git config --global user.name "Your Name"
$ git config --global user.email "Your Email"
$ git config -list
```

The following example shows how to download the i.MX Yocto Project Community BSP recipe layers. In this example we first create the repository *imx-yocto-bsp* for the project.

```
$ mkdir imx-yocto-bsp
$ cd imx-yocto-bsp
$ repo init -u https://github.com/nxp-imx/imx-manifest -b imx-linux-scarthgap -m imx-
X.Y.Z-V.V.V.xml
$ repo sync
```

***Note:*** *In the above command `X, Y, Z, V` refers to i.MX version, for example imx-6.6.36-2.1.0.xml*

When this process is completed, the source code is checked out into *imx-yocto-bsp/sources* repository.

#### 3.2.2.1.4 Set up the build directory and configuration files

Run the commands or scripts provided by the board manufacturer to set up the *build* directory and configuration files.

The command for i.MX8QM MEK board is as follows:

```
$ DISTRO=fsl-imx-wayland MACHINE=imx8qmmek source imx-setup-release.sh -b build
```

***Note:*** *Read more in ref.[15].*

#### 3.2.2.1.5 Build an image

The Yocto Project build uses the `bitbake` command. For example, `bitbake <component>` builds the named component. Each component build has multiple tasks, such as fetching, configuration, compilation, packaging, and deploying to the target rootfs. The *bitbake image build* gathers all the components required by the image and build in order of the dependency per task.

Command to build an image:

```
$ bitbake core-image-base //A console-only image that fully supports the target device
 hardware.
```

#### 3.2.2.2 Add the Wi-Fi utilities and drivers

This section explains how to add the Wi-Fi utilities and/or kernel modules in case they are not already included in the yocto build.

*Wpa_supplicant*, *hostapd utilities* and *cfg80211 subsystem* are required to configure the interface as an AP or STA.

##### 3.2.2.2.1 Add Wi-Fi utilities

To append the DISTRO feature in the local configuration for yocto, add the following lines to *local.conf* file.

```
$ vim imx-yocto-bsp/<build dir>/conf/local.conf

DISTRO_FEATURES_append += " wifi iw"

IMAGE_INSTALL_append += " hostapd wpa-supplicant"
```

Build the above configurations into the image using the command:

```
$ bitbake core-image-base
```

##### 3.2.2.2.2 Enable the features in *hostapd* and *wpa_supplicant*

This section shows how to enable the supported features in *hostapd* and *wpa_supplicant* utilities. Use the following steps to enable IEEE 802.11ax features in the version 2.11 of these utilities. Use the same steps to enable other features in these utilities.

### Steps for hostapd

- Append hostapd recipe by creating *hostapd.bbappend* file.

```
$ vim imx-yocto-bsp/sources/meta-openembedded/meta-oe/recipes-connectivity/hostapd/
hostapd_2.11.bbappend
# Customization of hostapd

# Files directory
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
SRC_URI += " \
file://hostapd.cfg \
"

do_configure_append() {
cat ${WORKDIR}/*.cfg >> ${B}/.config
}
```

- Create *hostapd.cfg* file in *imx-yocto-bsp/sources/meta-openembedded/meta-oe/recipes-connectivity/hostapd/hostapd* with the required flags.

```
$ vim imx-yocto-bsp/sources/meta-openembedded/meta-oe/recipes-connectivity/hostapd/
hostapd/hostapd.cfg

CONFIG_IEEE80211AX=y
```

- Build the hostapd

```
$ bitbake hostapd -f
$ bitbake core-image-base
```

### Steps for wpa_supplicant

- Append *wpa_supplicant* recipe by creating *wpa_supplicant.bbappend* file with below given data.

```
$ vim imx-yocto-bsp/sources/poky/meta/recipes-connectivity/wpa-supplicant/wpa-
supplicant_2.11.bbappend
# Customization of wpa_supplicant
# Files directory
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
do_configure_append() {
   echo "CONFIG_IEEE80211AX=y" >> wpa_supplicant/.config
}
```

- Add the following line to *imx-yocto-bsp/sources/poky/meta/conf/layer.conf* file just after `BBFILES += "${LAYERDIR}/recipes-*/*/*.bb"` line

```
BBFILES += " ${LAYERDIR}/recipes-*/*/*.bbappend"
```

- Build the *wpa_supplicant* utility

```
$ bitbake wpa-supplicant -f
$ bitbake core-image-base
```

#### 3.2.2.2.3 Modify the Kernel configuration for Wi-Fi

Use the following steps to enable *cfg80211*, *nl80211* and *mac80211* subsystem support in kernel to run *wpa_supplicant* and hostapd with *nl80211*.

- Open the `menuconfig` option for the board-specific kernel:

```
$ bitbake linux-imx -c menuconfig
```

- Enable the cfg80211 subsystem:

```
[*]   Networking Support
-*-   Wireless  --->
<*>   cfg80211 - wireless configuration API
[*]   nl80211 testmode command
[*]   cfg80211 wireless extensions compatibility
<*>   Generic IEEE 802.11 Networking Stack (mac80211)
```

- Build the image:

```
$ bitbake linux-imx
$ bitbake core-image-base
```

#### 3.2.2.3 Add the utilities and drivers for Bluetooth/Bluetooth LE

This section shows how to add the packages and kernel modules for Bluetooth/Bluetooth LE.

##### 3.2.2.3.1 Add packages for Bluetooth/Bluetooth LE

The following steps are not required if the yocto build includes BlueZ, PipeWire, and Ofono packages.

PipeWire is used to configure the Bluetooth audio profile and Ofono is used to configure the Bluetooth Headsfree profile.

Use the following steps to install the packages if these are not available in the yocto Linux image.

- Update *local.conf* file with the following lines of code to append the `DISTRO` features in the local configuration for yocto:

```
$ vim imx-yocto-bsp/build/conf/local.conf

DISTRO_FEATURES_append += " bluez5 bluetooth PipeWire"

IMAGE_INSTALL_append += " bluez5 pipewire pipewire-audio pipewire-media-session
 wireplumber
 alsa-utils pipewire-module-bluetooth pipewire-module-bluez5 pipewire-module-rtp ofono
 ofono-tests"
```

- Build the updated configuration into the image:

```
$ bitbake core-image-base
```

##### 3.2.2.3.2 Kernel configuration for Bluetooth/Bluetooth LE

Use the following steps to add the support of HCI UART driver as a module if it is a part of kernel and to enable HID driver.

- Open the `menuconfig` option for the board-specific kernel

```
$ bitbake linux-imx -c menuconfig
```

- Make the HCI UART driver as a part of modules if it is a part of kernel:

```
[*] Networking Support
    <*> Bluetooth subsystem support
       Bluetooth device drivers
       <M> HCI UART driver
```

- Enable the Generic HID driver:

```
Device Drivers
   HID support
     HID bus support
     <*> Generic HID driver
```

- If the external USB to UART interface is used, enable the FTDI serial converter driver:

```
Device Drivers
 [*] USB support
      <M> USB Serial Converter support
          <M> USB FTDI Single Port Serial Driver
```

- Build all the updates into the image:

```
$ bitbake linux-imx
$ bitbake core-image-base
```

#### 3.2.2.4 Modify the DTS file for PCIe interface

This section shows how to change the DTS file to add the support of NXP device on a non-NXP SoC.

Addressable devices use the following properties to encode the address information into the device tree. Look for the address information in the SoC datasheet or ask the SoC vendor.

- reg
- #address-cells
- #size-cells

**PCI address translation**

Similar to the local bus, the PCI address space is completely separate from the CPU address space. Thus, the address translation is needed to get from a PCI address to a CPU address. The address translation uses the range, #address-cells, and #size-cells properties.

```
pci@0x10180000 {
           compatible = "arm,versatile-pci-hostbridge", "pci";
            reg = <0x10180000 0x1000>;
            interrupts = <8 0>;
            bus-range = <0 0>;
            #address-cells = <3>
            #size-cells = <2>;
            ranges = <0x42000000 0 0x80000000 0x80000000 0 0x20000000
                      0x02000000 0 0xa0000000 0xa0000000 0 0x10000000
                      0x01000000 0 0x00000000 0xb0000000 00x01000000>;
        }
```

Example of i.MX8QM MEK board:

```
pcie@0x5f000000 {
                compatible = "fsl,imx8qm-pcie", "snps,dw-pcie";
                reg = <0x5f000000 0x10000 0x6ff00000 0x80000 0x5f110000 0x60000>;
                reg-names = "dbi", "config", "hsio";
                #address-cells = <0x3>;
                #size-cells = <0x2>;
                device_type = "pci";
                bus-range = <0x0 0xff>;
                ranges = <0x81000000 0x0 0x0 0x6ff80000 0x0 0x10000 0x82000000 0x0 0x60000000
 0x60000000 0x0 0xff00000>;
                num-lanes = <0x1>;
                num-viewport = <0x4>;
                interrupts = <0x0 0x46 0x4 0x0 0x48 0x4>;
                interrupt-names = "msi", "dma";
                #interrupt-cells = <0x1>;
                interrupt-map-mask = <0x0 0x0 0x0 0x7>;
                interrupt-map = <0x0 0x0 0x0 0x1 0x1 0x0 0x49 0x4 0x0 0x0 0x0 0x2 0x1 0x0 0x4a 0x4
 0x0 0x0 0x0 0x3 0x1 0x0 0x4b 0x4 0x0 0x0 0x0 0x4 0x1 0x0 0x4c 0x4>;
                clocks = <0x91 0x0 0x91 0x1 0x91 0x2 0x92 0x0 0x93 0x0 0x94 0x0 0x8f 0x0>;
                clock-names = "pcie", "pcie_bus", "pcie_inbound_axi", "pcie_phy", "phy_per",
 "pcie_per", "misc_per";
                power-domains = <0x19 0x98 0x19 0x99 0x19 0xac>;
                power-domain-names = "pcie", "pcie_phy", "hsio_gpio";
                fsl,max-link-speed = <0x3>;
                hsio-cfg = <0x2>;
                local-addr = <0x40000000>;
                status = "okay";
                pinctrl-names = "default";
                pinctrl-0 = <0x95>;
                reset-gpio = <0x20 0x1d 0x1>;
                disable-gpio = <0x20 0x9 0x1>;
                ext_osc = <0x1>;
                epdev_on-supply = <0x96>;
                reserved-region = <0x97>;
               };
```

#### 3.2.2.4.1 Steps to modify the DTS file

The *devtool* sets up an environment to enable changes in the source of an existing component. The commands in this section are for i.MX platform. Change the path and recipe name for a non-NXP SoC.

- Enable the kernel modification:

```
$ devtool modify -x linux-imx
```

The kernel source locally resides in *imx-yocto-bsp/build/workspace/sources/linux-imx*

- Modify the device tree file *arch/arm64/boot/dts/freescale/<dts file>* per SoC
- Build the changes into the image:

```
$ bitbake linux-imx
$ bitbake core-image-base
```

#### 3.2.2.5 Cross compile the wireless driver/utilities and flash the image

Download the driver package from NXP website and follow the steps in this section to cross compile the driver and utilities, and to flash the image.

##### 3.2.2.5.1 Build a toolchain

Use the following steps to build a toolchain for an i.MX EVK board. Note that the steps slightly differ for the different i.MX platforms.

- Build the toolchain

```
$ bitbake meta-toolchain
```

- Install the toolchain. The following commands refer to i.MX platform.

```
$ cd <build dir>/tmp/deploy/sdk
$ sudo ./fsl-imx-wayland-glibc-x86_64-meta-toolchain-aarch64-toolchain-5.4-zeus.sh
```

The toolchain is installed with the default settings at */opt/fsl-imx-wayland/5.4-zeus* location for i.MX platform. The path may differ depending on the i.MX platform.

- Set up the environment variables

```
$ source /opt/fsl-imx-wayland/5.4-zeus/environment-setup-aarch64-poky-linux
```

- Verify the cross compiler setup and version

```
$ $CC --version
aarch64-poky-linux-gcc (GCC) 9.2.0
```

##### 3.2.2.5.2 Cross compile the drivers and utilities

The source code files used to cross compile the drivers and utilities are included in the software release of the wireless SoC. The required files are:

- *XXXX-app-src.tgz*
- *XXXX-GPL-src.tgz*
- *XXXX-mlan-src.tgz*

**Step 1** – Download the latest software package from the webpage of the Wireless SoC.

**Step 2** – Decompress or unzip the software package.

```
unzip XXX.zip
tar -xvf XXXX-app-src.tgz
tar -xvf XXXX-GPL-src.tgz
tar -xvf XXXX-mlan-src.tgz
```

#### 3.2.2.5.2.1 Wi-Fi

**Step 1** – Go to the directory *XXX-GPL/wlan_src.*

```
cd XXX-GPL/wlan_src
```

**Step 2** – Modify the Makefile to change the kernel build directory.

```
$ vim Makefile
```

**Step 3** – Change the KERNELDIR

```
<absolute-path>/imx-yocto-bsp/<build dir>/tmp/work/imx8qmmek-poky-linux/linux-imx/
<linuxkernel-version>-r0/build/
```

**Step 4** – Build the driver package.

```
$ make clean
$ make build
```

If the compilation is successful, the directory */bin_wlan* is created in *XXX-GPL* directory.

#### 3.2.2.5.2.2 Bluetooth

By default, a Linux kernel version greater than 6.1.22 includes the Bluetooth driver btnxpuart. If the driver is not available in the BSP, to configure the kernel and add btnxpuart driver, refer to ref.[7].

A *.dtb* file configures btnxpuart to load the firmware at the correct baud rate. To update the default *.dtb* file of the host platform, refer to Enabling Wi-Fi host interfaces with dtb files .

## 3.3 Software

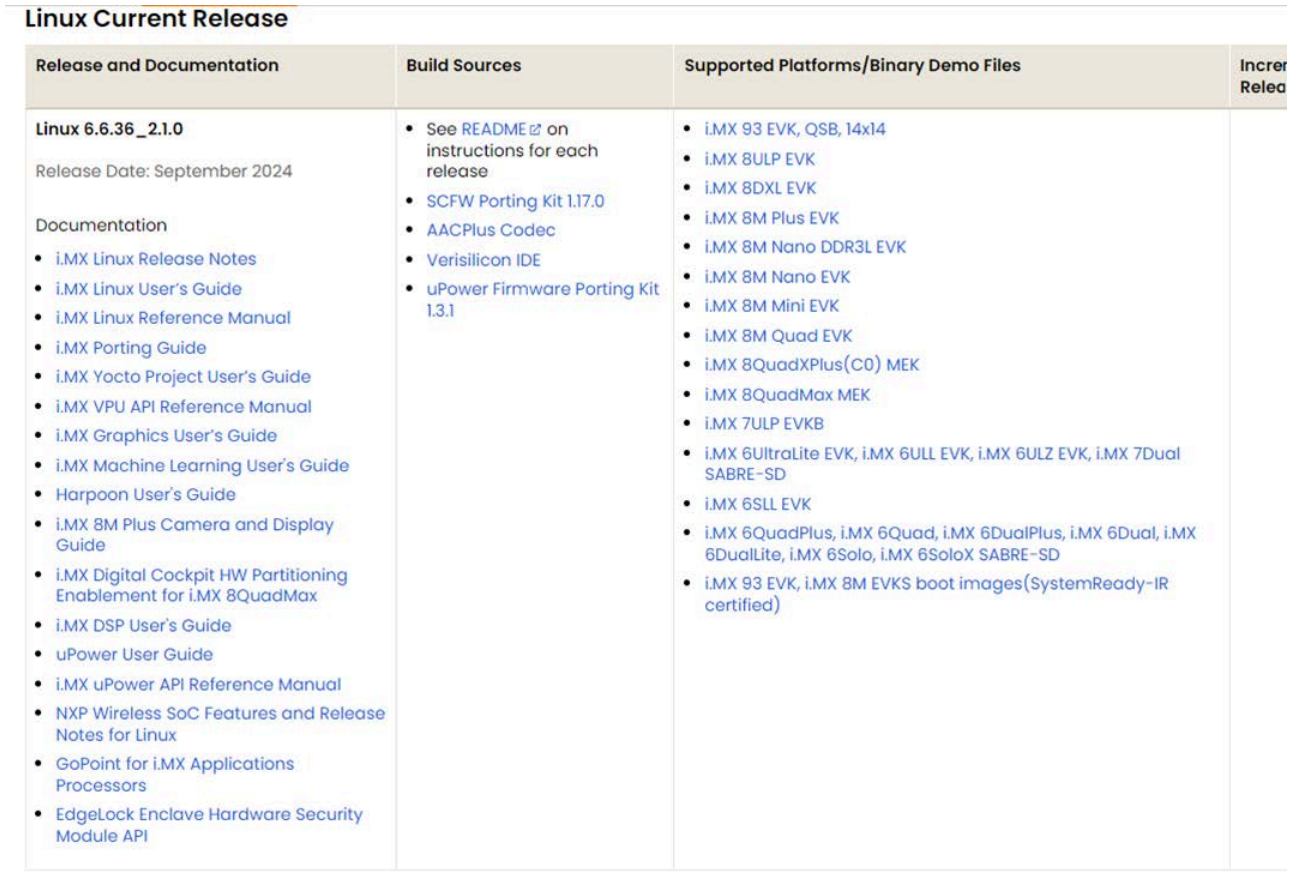
The i.MX Linux BSP is a collection of binary files, source code, and support files used to create a U-Boot bootloader, a Linux kernel image, and a root file system for i.MX development platforms.

All the information on how to set up the Linux OS host, how to run and configure a Yocto Project, generate an image, and generate a rootfs, are covered in Section 3.2.

To use a pre-built image, follow the steps below:

**Step 1** – Download the latest Linux release for the desired host platform from ref.[9]. The release includes a pre-built image that is built specifically for the board configuration.

***Note:*** *A pre-built image contains all the image elements and does not require to build the image using the Yocto setup.*



Linux Current Release

| Release and Documentation | Build Sources | Supported Platforms/Binary Demo Files | Increr Relea |
|---|---|---|---|
| Linux 6.6.36_2.1.0<br>Release Date: September 2024<br>Documentation<br>• i.MX Linux Release Notes<br>• i.MX Linux User's Guide<br>• i.MX Linux Reference Manual<br>• i.MX Porting Guide<br>• i.MX Yocto Project User's Guide<br>• i.MX VPU API Reference Manual<br>• i.MX Graphics User's Guide<br>• i.MX Machine Learning User's Guide<br>• Harpoon User's Guide<br>• i.MX 8M Plus Camera and Display Guide<br>• i.MX Digital Cockpit HW Partitioning Enablement for i.MX 8QuadMax<br>• i.MX DSP User's Guide<br>• uPower User Guide<br>• i.MX uPower API Reference Manual<br>• NXP Wireless SoC Features and Release Notes for Linux<br>• GoPoint for i.MX Applications Processors<br>• EdgeLock Enclave Hardware Security Module API | • See README on instructions for each release<br>• SCFW Porting Kit 1.17.0<br>• AACPlus Codec<br>• Verisilicon IDE<br>• uPower Firmware Porting Kit 1.3.1 | • i.MX 93 EVK, QSB, 14x14<br>• i.MX 8ULP EVK<br>• i.MX 8DXL EVK<br>• i.MX 8M Plus EVK<br>• i.MX 8M Nano DDR3L EVK<br>• i.MX 8M Nano EVK<br>• i.MX 8M Mini EVK<br>• i.MX 8M Quad EVK<br>• i.MX 8QuadXPlus(C0) MEK<br>• i.MX 8QuadMax MEK<br>• i.MX 7ULP EVKB<br>• i.MX 6UltraLite EVK, i.MX 6ULL EVK, i.MX 6ULZ EVK, i.MX 7Dual SABRE-SD<br>• i.MX 6SLL EVK<br>• i.MX 6QuadPlus, i.MX 6Quad, i.MX 6DualPlus, i.MX 6Dual, i.MX 6DualLite, i.MX 6Solo, i.MX 6SoloX SABRE-SD<br>• i.MX 93 EVK, i.MX 8M EVKS boot images(SystemReady-IR certified) | |

**Figure 7. Example of Linux release for i.MX**

**Step 2** – Extract the downloaded release.

Example of release content:

```
EULA.txt
fsl-image-mfgtool-initramfs-imx_mfgtools.cpio.zst.u-boot
GPLv2
Image-imx8mmevk.bin
Image-imx8_all.bin
imx-boot-imx8mm-ddr4-evk-nand.bin-flash_ddr4_evk
imx-boot-imx8mm-ddr4-evk-sd.bin-flash_ddr4_evk
imx-boot-imx8mm-lpddr4-evk-fspi.bin-flash_evk_flexspi
imx-boot-imx8mm-lpddr4-evk-sd.bin-flash_evk
imx-boot-imx8mmevk-sd.bin-flash_evk
imx-image-full-imx8mmevk.manifest
imx-image-full-imx8mmevk.spdx.tar.zst
imx-image-full-imx8mmevk.tar.zst
imx-image-full-imx8mmevk.wic
imx8mm-ddr4-evk-pcie-ep.dtb
imx8mm-ddr4-evk-revb-rm67191-cmd-ram.dtb
imx8mm-ddr4-evk-revb-rm67191.dtb
imx8mm-ddr4-evk-revb-rm67199-cmd-ram.dtb
…
imx8mm-evk-usd-wifi.dtb
imx8mm-evk.dtb
SCR-6.6.23-2.0.0.txt
uuu.auto
uuu.auto-imx8mmevk
```

### 3.3.1 Flash the image

**Step 1** – Connect power and USB-C cable to the host platform. Refer to the host platform's webpage for the hardware connections.

**Step 2** – Switch the host platform boot configurations to download mode.

***Note:*** *The boot configurations are written in a table on the host platform.*

**Step 3** – Download Universal Update Utility (UUU) to download images to different devices on an i.MX board. uuu runs on Linux or Windows OS.

**Step 4** – (Linux OS) – Set executable permissions to the uuu binary.

```
sudo chmod +x /usr/bin/uuu
```

(Windows OS) – Place the downloaded *uuu.exe* in the same directory as the downloaded release.

**Step 5** – Open a command prompt terminal.

**Step 6** – Navigate to the download Linux image and uuu.

**Step 7** – Flash the host platform.

Command for Linux OS:

```
sudo uuu -b emmc_all imx-boot-XXXX-sd.bin-flash_evk imx-image-full-XXXX.wic
```

Command for Windows OS:

```
uuu.exe -b emmc_all imx-boot-XXXX-sd.bin-flash_evk imx-image-full-XXXX.wic
```

**Step 8** – Switch the host platform boot configurations to eMMC mode.

***Note:*** *The boot configurations are written in a table on the host platform.*

### 3.3.2 Serial console setup

For Linux OS, minicom is used to access the serial console of the host platform.

**Step 1** – Connect the micro-USB or USB-C cable into the debug slot on the host platform and the other end to the PC. Refer to the host platform's webpage for the hardware connections.

**Step 2** – Open the serial console and login into the device.

```
ubuntu@ubuntu-desktop:/# sudo minicom -s -D /dev/ttyUSBx
```

`ttyUSB0` or `ttyUSB01` are the serial devices. The minicom setup configuration is as follows:

```
A - Serial Device      : /dev/ttyUSBx
E - Bps/Par/Bits       : 115200 8N1
F - Hardware Flow Control : No
G - Software Flow Control : No
```

**Step 3** – Save and exit.

For Windows OS, any serial terminal, such as TeraTerm or PuTTy is used to access the host platform.

**Step 1** – Connect the micro-USB or USB-C cable into the debug slot on the host platform and the other end to the PC. Refer to the host platform's webpage for the hardware connections.

**Step2** – Open a serial port console with the baud rate of 115200 and other serial port settings on the PC (Figure 8).

**Step 3** – Enter the login information on the console:

```
imx8mmevk login: root
```

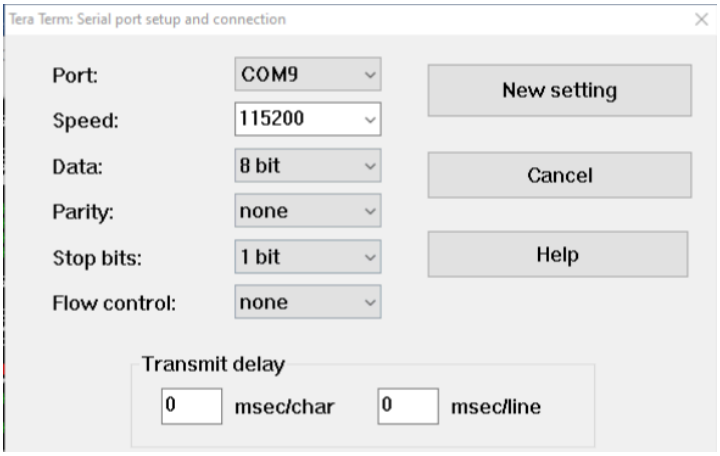



**Figure 8. Serial port settings**

### 3.3.3 Enabling Wi-Fi host interfaces with dtb files

The device tree file (*.dtb*) is used to configure the Wi-Fi host interface for each wireless SoC connected to an i.MX platform. By default, the Linux BSP contains precompiled *dtb* files to enable SDIO and PCIe host interfaces. The precompiled *dtb* files are located in */run/media/boot-mmcblk2p1* directory.

Figure 9 shows the *.dtb* files in */run/media/boot-mmcblk2p1* directory.

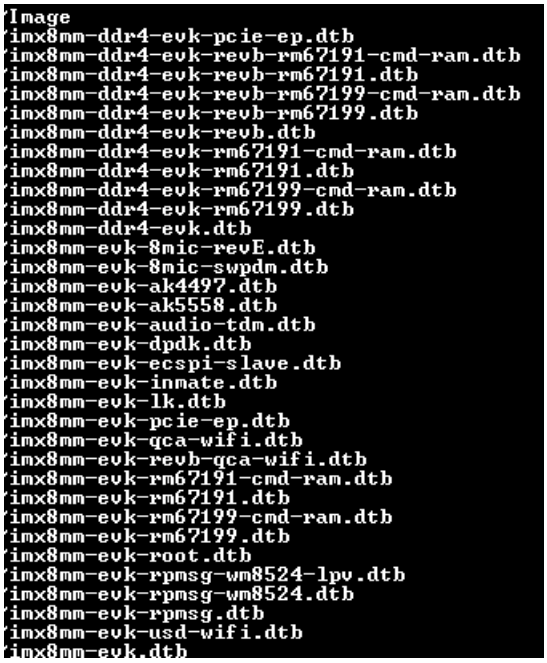



**Figure 9. *.dtb* files in */run/media/boot-mmcblk2p1* directory**

***Note:*** *Each host platform has a specific set of .dtb files. To compile a custom .dtb file, refer to ref.[16].*

To enable a host interface, follow the steps below.

**Step 1** – Power-on the host platform board. Hit any key within 3 seconds to halt u-boot from booting and to enter the uboot console.

```
u-boot=>
```

**Step 2** – Set `fdtfile` parameter to the *.dtb* file of the host interface.

Command syntax:

```
u-boot=> setenv fdtfile <host interface>.dtb
```

Example of command to enable SDIO host interface:

```
u-boot=> setenv fdtfile imx8mm-evk-usd-wifi.dtb
```

**Step 3** – Save the host interface environment.

```
u-boot=> saveenv
```

Command output example:

```
Saving Environment to MMC... Writing to MMC(0)... OK
```

**Step 4** – Reset the host platform board.

```
u-boot=> reset
```

Command output example:

```
Resetting…
```

### 3.3.4 Linux OS login

The default login username for the i.MX Linux OS is *root*. There is no password.

# 4 Connectivity software integration

This section explains how to integrate the software release package onto the i.MX host platform and load onto the wireless SoC. The software release package is available on the webpage of the wireless SoC. Find information about the latest release in the release notes ref.[10].

## 4.1 Start-up and initialization

For NXP-based wireless modules, refer to ref.[18].

### 4.1.1 Download the software package

The pre-compiled Linux BSP includes the required drivers and firmware. Additional software is not required. However, standalone software release packages are available on the webpage of the supported wireless SoC (Section 1.1).

The following instructions apply to IW612 standalone software release as an example.

IW612 software is available for download in the Design resource section of IW612 webpage at nxp.com.

- Go to the product page (ref.[28]).
- Click **Design Resources**.


**Figure 10. Design Resources on IW612 product page**

- Click **Software** in **Design Resources**.
- Select Secure files
- Click **reenter your password** to access the secure files.

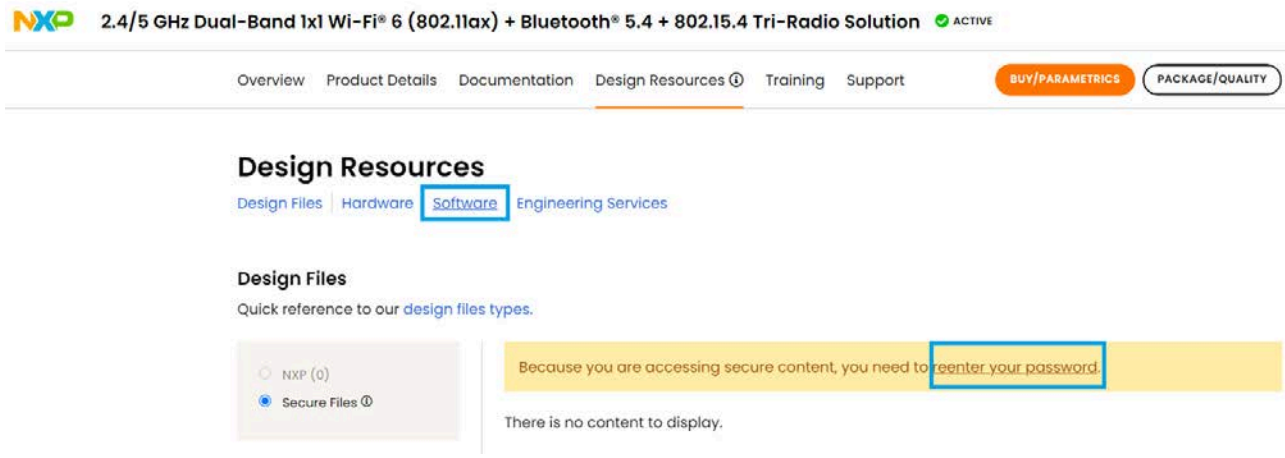



**Figure 11. Software secure files in Design Resources section**

- Select **BSP and Device Drivers** in Embedded Software section.
- Look for the latest software release and click **Download**.

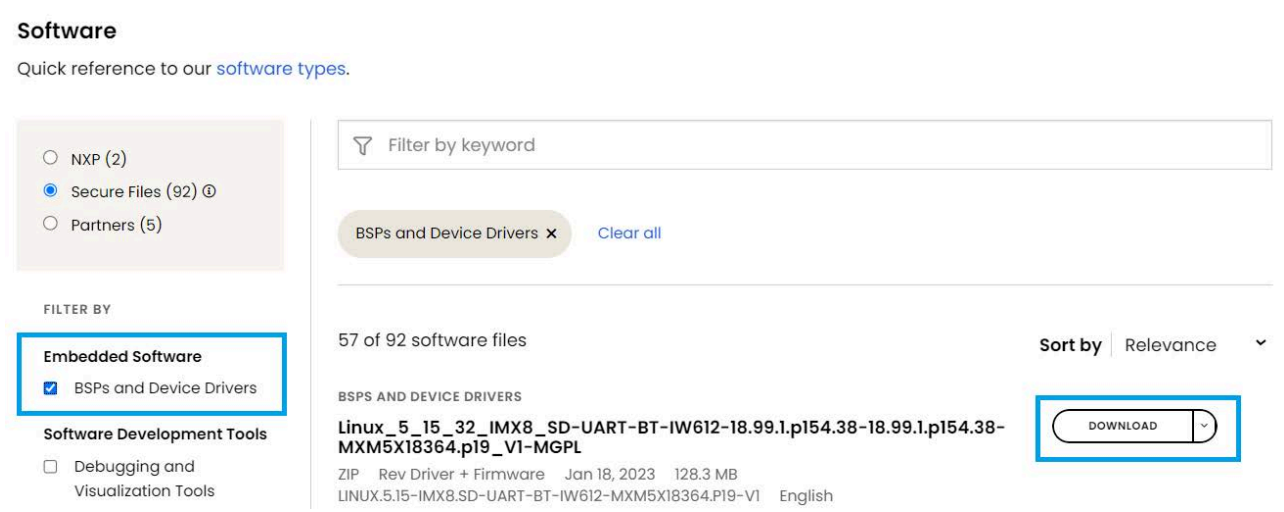



**Figure 12. Software files on IW612 webpage**

#### 4.1.1.1 Firmware types

Figure 13 shows the default */lib/firmware/nxp* directory content for i.MX 8M Quad host platform.

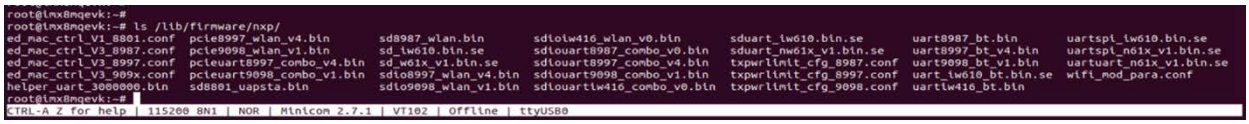



**Figure 13. Default */lib/firmware/nxp* directory content for i.MX 8M Quad host platform**

The */lib/firmware/nxp* directory contains:

- Combo firmware
  - A single firmware binary download activates Wi-Fi and Bluetooth/Bluetooth LE/802.15.4 radios.
  - The firmware is loaded through the Wi-Fi host interface.
- Standalone firmware
  - Separate firmware binary downloads for Wi-Fi and for Bluetooth/Bluetooth LE/802.15.4 radios.
  - The Wi-Fi firmware and the Bluetooth/Bluetooth LE/802.15.4 firmware can be downloaded independently.

The same binaries as for the combo and standalone firmware can be used for regulatory testing.

The naming convention for the firmware binaries is as follows:

- Wi-Fi host interface (IF) – PCIe, SDIO
- Bluetooth/Bluetooth LE IF – UART
- 802.15.4 IF – SPI
- Device identifier – for example 8997, 9098, IW612, and IW416
- Firmware type – combo, bt, wlan
- version (vX) where X is 0 - 4
- Binary security (*.bin* or *.bin.se*)
  - For *.bin.se*, the wireless SoC supports a EdgeLock® Secure (ELS) subsystem to verify the digital NXP signature of the firmware and decrypt the secure firmware.
  - The boot ROM is set to secure mode, so the security unit must verify all the software.
  - The access to JTAG is blocked.
  - During the firmware download, the security unit verifies the digital signature. If the verification is successful, the secure firmware downloads. The digital verification of unauthorized software fails and the download is blocked.

**Table 2. Firmware binary files included in *FwImage* directory**

| Firmware type | Firmware file |
|---|---|
| Wi-Fi and Bluetooth/Bluetooth LE/802.15.4 combo firmware | Syntax:<br>`<Wi-Fi IF><Bluetooth IF><802.15.4 IF><device identifier>.<binary security>`<br>Example for IW612: *sduart_nw61x_v1.bin.se* |
| Standalone Wi-Fi firmware | Syntax:<br>`<Wi-Fi IF><device identifier>.<binary security>`<br>Example for IW612: *sd_nw61x_v1.bin.se* |

**Table 2. Firmware binary files included in *FwImage* directory** ***...continued***

| Firmware type | Firmware file |
|---|---|
| Standalone Bluetooth/Bluetooth LE/802.15.4 firmware | Syntax:<br>`<Bluetooth IF><802.15.4 IF><device identifier>.<binary security>`<br>Example for IW612: *uartspi_nw61x_v1.bin.se* |

### 4.1.2 Transfer the files to i.MX host platform

The software is downloaded on the host PC and copied to i.MX host platform running Linux OS. This section explains the two methods to transfer files from the host PC to the i.MX host platform.

- One method uses file copy using an USB mass storage device (USB-C to USB adapter included)
- The other method uses file copy using Ethernet connection and secure copy (SCP).

#### 4.1.2.1 USB-C to USB adapter

The procedure to use USB-C to USB adapter to transfer files is as follows:

**Step 1** – Hardware connections

- Plug the USB-C to USB adapter to the “Download” slot on i.MX 8M Mini EVK board
- Plug the USB flash drive containing the downloaded software to the other USB end of the adapter

***Note:*** *Linux supports FAT32 and exFAT USB formats. Other formats may run into an error.*

Figure 14 shows USB-C to USB adapter connections.

**Figure 14. USB-C to USB adapter connections**

**Step 2** – Debug access

- Refer to the section *Board set-up* in ref.[29] and ref.[30].

**Step 3** – File transfers

- Issue the command to locate the USB flash drive content

```
lsblk
```

Example of command output:

```
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda             8:0    1 59.6G  0 disk
`-sda1          8:1    1 59.6G  0 part /run/media/sda1
mtdblock0      31:0    0   32M  0 disk
mmcblk2       179:0    0 14.7G  0 disk
|-mmcblk2p1   179:1    0 83.2M  0 part /run/media/mmcblk2p1
`-mmcblk2p2   179:2    0  4.5G  0 part /
mmcblk2boot0 179:32   0    4M  1 disk
mmcblk2boot1 179:64   0    4M  1 disk
```

In the above example, the content of the USB flash drive is located in the directory */run/media/sda1*.

- Move to USB flash drive directory

```
cd <USB directory>
```

- Copy the files to */home/root*

```
cp <software_release_package>.tar /home/root
```

#### 4.1.2.2 SCP utility (optional)

This section explains how to use SCP utility to transfer files from the host PC to the i.MX host platform board via Ethernet.

**Step 1** – Set up the Ethernet interface.

To connect the two interfaces on the same network:

- Attach one end of the ethernet cable to the i.MX host platform Ethernet port and the other end to the host PC

***Note:*** *The ethernet port is used either to transfer files or for debug access (SSH).*

- Get the IP address of the connected PC.

```
ifconfig
```

- Assign the IP address of the i.MX host platform on eth0 interface.

```
ifconfig eth0 192.168.1.100 up
```

- Check if eth0 interface is up.

```
ifconfig eth0
```

Example of output with matching IP addresses for the connected PC and the i.MX host platform board:

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
inet 192.168.1.100  netmask 255.255.255.0  broadcast 192.168.1.255
inet6 fe80::204:9fff:fe07:1230  prefixlen 64  scopeid 0x20<link>
ether 00:04:9f:07:12:30  txqueuelen 1000  (Ethernet)
RX packets 1817  bytes 2605025 (2.4 MiB)
RX errors 0  dropped 0  overruns 0  frame 0
TX packets 272  bytes 32259 (31.5 KiB)
```

**Step 2** – Copy the compressed software release files to the i.MX host platform board.

```
scp -r <sofwtare_release_package>.tar root:@192.168.1.100:/home/root
```

#### 4.1.2.3 Extract the files

This section explains how to decompress files from the downloaded software package.

To decompress a tar file:

**Step 1** – Open a command prompt or a terminal on the host system.

**Step 2** – Issue the command:

```
tar -xvf <sofwtare_release_package>tgz
```

The command creates a directory of the same name with:

- FwImage directory
- Calibration data files
- Source and driver code for Wi-Fi, Bluetooth, 802.15.4 tgz files to decompress

### 4.1.3 Load the firmware

As a recommendation, use the driver default parameters when loading driver and firmware on the wireless SoC. In case any changes are needed, configure the driver parameters using one of the following methods:

- *wifi_mod_para.conf*: default configuration file in the Linux Kernel.
  - Located in */lib/firmware/nxp* directory with all the firmware binaries.
  - Contains the configuration blocks that define the driver and firmware parameters for each wireless SoCs.
- insmod commands are manually executed in the same directory as the drivers

This section explains how to load the combo, standalone Wi-Fi, standalone Bluetooth/Bluetooth LE/802.15.4 firmware binaries using both methods.

**Table 3. wifi_mod_para.conf configuration blocks for the supported wireless SoCs**

| Wireless SoC | Configuration block |
|---|---|
| 88W8987 | <Wi-Fi IF>8987 = {<br>cfg80211_wext=0xf<br>max_vir_bss=1<br>cal_data_cfg=none<br>ps_mode=1<br>auto_ds=1<br>host_mlme=1<br>fw_name=nxp/<firmware>.bin<br>} |
| 88W8997 | <Wi-Fi IF>8997 = {<br>cfg80211_wext=0xf<br>max_vir_bss=1<br>cal_data_cfg=none<br>ps_mode=1<br>auto_ds=1<br>host_mlme=1<br>fw_name=nxp/<firmware>.bin<br>} |
| 88W9098 | <Wi-Fi IF>9098_0 = {<br>cfg80211_wext=0xf<br>max_vir_bss=1<br>cal_data_cfg=none<br>ps_mode=1<br>auto_ds=1<br>host_mlme=1<br>fw_name=nxp/<firmware>.bin<br>} |
| IW416 | SDIW416 = {<br>cfg80211_wext=0xf<br>max_vir_bss=1<br>cal_data_cfg=none<br>ps_mode=1<br>auto_ds=1<br>host_mlme=1<br>fw_name=nxp/<firmware>.bin<br>} |

**Table 3. wifi_mod_para.conf configuration blocks for the supported wireless SoCs** ***...continued***

| Wireless SoC | Configuration block |
|---|---|
| IW612 | `SDIW612 = {`<br>`  cfg80211_wext=0xf`<br>`  wfd_name=p2p`<br>`  max_vir_bss=1`<br>`  cal_data_cfg=none`<br>`  drv_mode=7`<br>`  ps_mode=2`<br>`  auto_ds=2`<br>`  host_mlme=1`<br>`  fw_name=nxp/<firmware>.bin.se`<br>`}` |
| IW610 | `SDIW610 = {`<br>`  cfg80211_wext=0xf`<br>`  max_vir_bss=1`<br>`  cal_data_cfg=none ps_mode=1`<br>`  auto_ds=1`<br>`  host_mlme=1`<br>`  fw_name=nxp/<firmware>.bin.se`<br>`}` |

#### 4.1.3.1 Combo firmware

To load the combo firmware, follow the step below.

**Step 1** – Configure `fw_serial=1` parameter in */lib/firmware/nxp/wifi_mod_para.conf* for the wireless SoC combo firmware download.

. Refer to Section 4.1.1.1 for firmware types and Section 4.1.3 for more information on the load parameters.

Combo firmware syntax:

```
<Wi-Fi IF><Bluetooth IF><802.15.4 IF><device number>_combo_<version>.<binary security>
```

**Step 2** – Load the drivers and firmware.

***Note:*** *Modprobe uses the default mlan.ko and moal.ko drivers located in /lib/modules/<Linux_Kernel>/updates. If other drivers are used, run `insmod <driver>.ko` in the same directory as the drivers.*

Command when using *wifi_mod_para.conf* method:

```
modprobe moal mod_para=/nxp/wifi_mod_para.conf
```

Example of command output for 88W9098:

```
mlan: loading out-of-tree module taints kernel.
wlan: Loading MWLAN driver
wlan: Driver loaded successfully
wlan: Register to Bus Driver...
vendor=0x02DF device=0x914D class=0 function=1
Attach moal handle ops, card interface type: 0x106
SD9098: init module param from usr cfg
card_type: SD9098, config block: 0
cfg80211_wext=0xf
max_vir_bss=1
cal_data_cfg=none
ps_mode = 1
auto_ds = 1
host_mlme=enable mac_addr=00:50:43:20:12:34
fw_name=nxp/sdiouart9098_combo_v1.bin
SDIO: max_segs=128 max_seg_size=65535
rx_work=1 cpu_num=4
Attach mlan adapter operations.card_type is 0x106.
wlan: Enable TX SG mode
wlan: Enable RX SG mode
Request firmware: nxp/sdiouart9098_combo_v1.bin
Wlan: FW download over, firmwarelen=631336 downloaded 631336
WLAN FW is active
fw_cap_info=0x81c3fa3, dev_cap_mask=0xffffffff
max_p2p_conn = 8, max_sta_conn = 64
wlan: version = SD9098----17.92.1.p91-MM5X17293-GPL-(FP92)
Set REG 0x90002328: 0x3000 slew_rate=3
vendor=0x02DF device=0x914E class=0 function=2
…
```

**Step 3** – Load btnxpuart.

```
modprobe btnxpuart
```

**Step 4** – Bring up Bluetooth interface, hci0.

```
hciconfig hci0 up
```

**Step 5** – Verify that hci0 is up.

```
hciconfig -a
```

Command output example:

```
hci0: Type: Primary Bus: UART
BD Address: 88:88:88:88:88:88 ACL MTU: 1021:7 SCO MTU: 120:6
UP RUNNING
RX bytes:1513 acl:0 sco:0 events:92 errors:0
TX bytes:1282 acl:0 sco:0 commands:92 errors:0
```

Where BD Address: 88:88:88:88:88:88 is the default address of the Bluetooth device.

#### 4.1.3.2 Standalone Wi-Fi firmware

To load the standalone firmware, follow the steps below.

**Step 1** – Configure `fw_name` parameter in */lib/firmware/nxp/wifi_mod_para.conf* for standalone Wi-Fi firmware download. Refer to Section 4.1.1.1 for firmware types and Section 4.1.3 for more information on the load parameters.

Standalone Wi-Fi firmware syntax:

```
<Wi-Fi IF><device number>_wlan_<version>.<binary security>
```

**Step 2** – Load the drivers and firmware.

***Note:*** *Modprobe uses the default mlan.ko and moal.ko drivers located in /lib/modules/<Linux_Kernel>/updates. If other drivers are used, run* `insmod <driver>.ko` *in the same directory as the drivers.*

Command when using wifi_mod_para.conf method:

```
modprobe moal mod_para=/nxp/wifi_mod_para.conf
```

Example of command output:

```
wlan: Loading MWLAN driver
…
Request firmware: nxp/sd_w61x_v1.bin.se
Wlan: FW download over, firmwarelen=492724 downloaded 492724
…
WLAN FW is active
wlan: version = SD9177----18.99.1.p154.23-MXM5X18312.p7_V1-MGPL ).
wlan: Driver loaded successfully
```

#### 4.1.3.3 Standalone Bluetooth/802.15.4 firmware via btnxpuart

Linux Kernel versions greater than 6.1.22 include the built-in Bluetooth driver btnxpuart. The driver loads the Bluetooth and 802.15.4 firmware and brings up the Bluetooth interface hci0. For more information, refer to ref. [4].

To load the standalone Bluetooth and 802.15.4 firmware, the preferred method is with btnxpuart.

**Step 1** – Load btnxpuart.

```
modprobe btnxpuart
```

**Step 2** – Bring up Bluetooth interface, hci0.

```
hciconfig hci0 up
```

**Step 4** – Verify that hci0 is up.

```
hciconfig -a
```

Command output example:

```
hci0: Type: Primary Bus: UART
BD Address: 88:88:88:88:88:88 ACL MTU: 1021:7 SCO MTU: 120:6
UP RUNNING
RX bytes:1513 acl:0 sco:0 events:92 errors:0
TX bytes:1282 acl:0 sco:0 commands:92 errors:0
```

Where BD Address: 88:88:88:88:88:88 is the default address of the Bluetooth device.

## 4.2 Bring-up and basic operation

### 4.2.1 Bring-up of Wi-Fi

This section describes the steps for the Wi-Fi interfaces.

#### 4.2.1.1 Bring up the Wi-Fi interface

Use the following steps to bring up the Wi-Fi interfaces

- Invoke the command to initialize *mlan0* interface

```
ifconfig mlan0 up
ifconfig mlan0
```

Command output example:

```
mlan0   Link encap:Ethernet HWaddr 00:50:43:24:83:c4
        UP BROADCAST MULTICAST MTU:1500 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

- Invoke the command to initialize the *uap0* interface

```
ifconfig uap0 up
ifconfig uap0
```

Command output example:

```
uap0   Link encap:Ethernet HWaddr 00:50:43:24:84:c4
       UP BROADCAST MULTICAST MTU:1500 Metric:1
       RX packets:0 errors:0 dropped:0 overruns:0 frame:0
       TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
```

### 4.2.2 Bring-up of Bluetooth

This section describes two different methods to bring up the Bluetooth interfaces:

- Bring up using NXP Bluetooth UART driver on the i.MX Linux BSP version L6.1.22 and later.
- Bring up using Linux Open Source UART driver on i.MX Linux BSP version earlier than L6.1.22.

#### 4.2.2.1 Bring-up of Bluetooth using btnxpuart

Refer to Standalone Bluetooth/802.15.4 firmware via btnxpuart .

### 4.2.3 Bring-up of 802.15.4 (IW610 and IW612 only)

Refer to the section *Bring-up of 802.15.4* in ref.[18].

## 4.3 Error handling and fatal error recovery

### 4.3.1 Wi-Fi independent reset

Refer to ref.[6].

### 4.3.2 Bluetooth independent reset

This section describes the Bluetooth/Bluetooth LE independent reset (IR) feature and its verification.

The independent reset feature allows the host to reset the Bluetooth controller and re download the Bluetooth only firmware through UART without powering OFF the Bluetooth controller. Where the host resets the controller and re downloads the firmware:

- To initialize new operations
- When the host detects unresponsiveness of the Bluetooth controller

In addition, IR feature allows the Wi-Fi driver or the Bluetooth driver to reset and re download their own firmware without depending on each other. For example if the Wi-Fi and Bluetooth combo firmware has been downloaded initially, and the Bluetooth firmware is not responding to host, the host uses the independent reset feature to reset and re download the Bluetooth only firmware without affecting the Wi-Fi operations.

#### 4.3.2.1 Bluetooth independent reset using in-band method

This section describes the steps to verify the independent reset feature with the in-band method.

#### 4.3.2.2 Bluetooth independent reset using NXP Bluetooth UART driver

This section describes the in-band method of independent reset feature verification using NXP Bluetooth UART driver for i.MX Linux BSP kernel version 6.1.22 and later.

**Step 1** – Download the firmware and initialize the Bluetooth interface

- To load the btnxpuart driver and to initialize the Bluetooth interface, refer to the section *Bring up of Bluetooth interface* in ref.[18].

**Step 2** – Set the in-band independent reset mode

```
# hcitool -I hci0 cmd 3f 0d 02 ff
```

**Step 3** – Trigger in-band independent reset

```
# hcitool -I hci0 cmd 3f fc 00
```

The command allows the driver to re download the Bluetooth only firmware over UART.

### 4.3.3 802.15.4 software recovery process example

This sub-section shows an example of recovery process on the i.MX host platform with out-of-band (OOB) IR using Linux shell script.

**Step 1** – Start ot-daemon.

```
ot-daemon "spinel+spi:///dev/spidev1.0?gpioreset-device=/dev/gpiochip5&gpio-int-device=/dev/
gpiochip5&gpio-intline=12&gpio-reset-line=13&spi-mode=0&spi-speed=1000000&spi-resetdelay=500" &
```

***Note:*** *Refer to OpenThread to rebuild ot-daemon with the following change if running this example on i.MX host platform platform. Other platforms do not need this change if the corresponding pin for ot-daemon gpio-reset-line parameter is not connected with OpenThread RCP controller 802.15.4 OOB IR pin.*

```
void SpiInterface::SetGpioValue(int aFd, uint8_t aValue)
{
   OT_UNUSED_VARIABLE(aFd);
   OT_UNUSED_VARIABLE(aValue);
}
```

**Step 2** – Create or join a Thread network.

Follow the instructions in Section 5.3.1.1.

**Step 3** – Enable OOB IR feature.

```
ot-ctl ircfg 1
```

**Step 4** – Copy *ot-host-monitor.sh* script from the wireless SoC software package to the i.MX host platform.

**Step 5** – Run *ot-host-monitor.sh* to monitor ot-daemon status and recover 802.15.4/Bluetooth from 802.15.4 unexpected exception status.

```
chmod 777 ot-host-monitor.sh
./ot-host-monitor.sh 60 S &
```

Figure 15 shows an example of recovery scenario using *ot-host-monitor.sh* script.

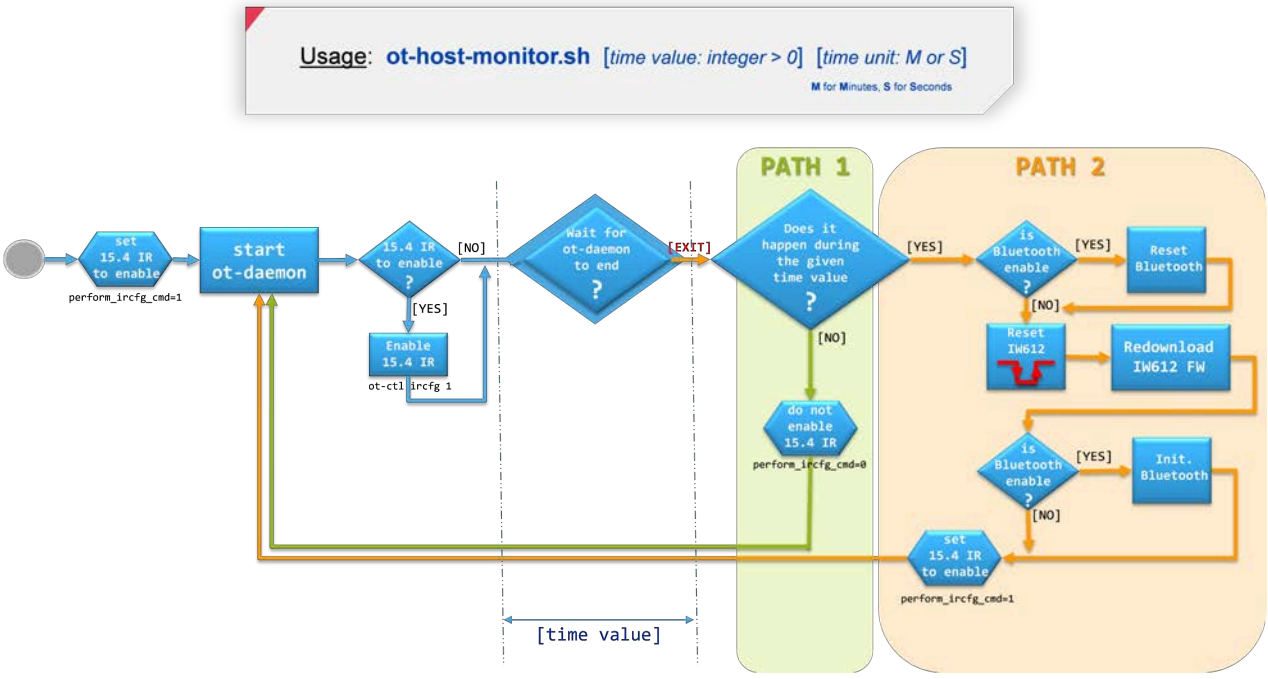



**Figure 15. Example of recovery scenario using *ot-host-monitor.sh* script**

The scenario illustrated in Figure 15 proposes two recovery paths named PATH 1 and PATH 2:

- PATH 1: no recent ot-daemon stop is detected, recovery attempt by just restarting ot-daemon
- PATH 2: ot-daemon unexpectedly stopped two successive times within less than [time value]. Need to trigger OOB IR to re-download the firmware and restart ot-daemon to recover.

***Note:***

1. *The script ot-host-monitor.sh auto exits if ot-daemon is not running first.*
2. *When a firmware redownload is mandatory to recover the functionality, ot-daemon takes longer than 20 seconds to exit again in error state. So, more than 20 seconds is suggested to prevent an endless loop of ot-daemon restart without a firmware download.*
3. *While ot-host-monitor.sh is running, an intentional stop/kill of ot-daemon triggers the recovery process. Stop the running ot-host-monitor.sh before stopping/killing ot-daemon.*
4. *The ECHO function within ot-host-monitor.sh automatically logs messages into /var/log/syslog. If the value of `DEBUG` variable is set to 1 in the script, the ECHO function also outputs messages directly in the Shell which started ot-host-monitor.sh script.*

## 4.4 Logging and debugging

### 4.4.1 Wi-Fi driver debugging

This section explains how to enable driver debug logs and access debug information. Some debug logs can be enabled at runtime and driver load time. Other logs require to first rebuild the driver with a specific configuration. The firmware dump methods to generates dump file to debug on the firmware front are also detailed in this section.

#### 4.4.1.1 Enable the driver debug logs

This section provides the guidelines on how to enable the driver debug logs using either the module parameter or `/proc` at runtime.

##### 4.4.1.1.1 Use the module parameter

Use the `drvdbg` module parameter to enable driver debug logs while the driver is loading. Refer to Driver debug log types for the module parameter values.

Load the module with `drvdbg` parameter:

```
modprobe moal mod_para=nxp/wifi_mod_para.conf drvdbg=<bit masks of driver debug message
 control>
```

***Note:*** *For the module parameters, refer to the Readme file in the software release.*

##### 4.4.1.1.2 Use /proc at runtime

Use the following command to enable driver debug logs when the driver is already loaded:

```
echo "drvdbg=<bit masks of driver debug message control>" >> /proc/mwlan/<adapterX>/
<interface>/debug
```

Where `adapterX` = adapter0 for MAC0, and `adapter#` = adapter1 for MAC1

#### 4.4.1.2 Driver debug log types

Table 4 lists the debug logs types exposed by the NXP driver for `drvdbg` parameter. The debug information can be enabled or disabled using either the module parameter ( Use the module parameter ) or /proc at runtime ( Use /proc at runtime ).

**Table 4. NXP driver debug log types**

| Bit | Message type | Log format | Description |
|---|---|---|---|
| Bit 0 | MMSG | PRINTM(MMSG,...) | Set bit 0 to enable all driver logs with log level MMSG. |
| Bit 1 | MFATAL | PRINTM(MFATAL,...) | Set bit 1 to enable all driver logs with log level MFATAL. |
| Bit 2 | MERROR | PRINTM(MERROR,...) | Set bit 2 to enable all driver logs with log level MERROR. |
| Bit 3 | MDATA | PRINTM(MDATA,...) | Set bit 3 to enable all driver logs with log level MDATA. |
| Bit 4 | MCMND | PRINTM(MCMND,...) | Set bit 4 to enable all driver logs with log level MCMND. |
| Bit 5 | MEVENT | PRINTM(MEVENT,...) | Set bit 5 to enable all driver logs with log level MEVENT. |
| Bit 6 | MINTR | PRINTM(MINTR,...) | Set bit 6 to enable all driver logs with log level MINTR. |
| Bit 7 | MIOCTL | PRINTM(MIOCTL,...) | Set bit 7 to enable all driver logs with log level MIOCTL. |
| Bit 16 | MDAT_D | PRINTM(MDAT_D,...), DBG_HEXDUMP(MDAT_D,...) | Set bit 16 to enable all driver logs with log level MDAT_D and provide the corresponding hexdump in dmesg logs. |
| Bit 17 | MCMD_D | PRINTM(MCMD_D,...), DBG_HEXDUMP(MCMD_D,...) | Set bit 17 to enable all driver logs with log level MCMD_D and provide the corresponding hexdump in dmesg logs. |
| Bit 18 | MEVT_D | PRINTM(MEVT_D,...), DBG_HEXDUMP(MEVT_D,...) | Set bit 18 to enable all driver logs with log level MEVT_D and provide the corresponding hexdump in dmesg logs. |
| Bit 19 | MFW_D | PRINTM(MFW_D,...), DBG_HEXDUMP(MFW_D,...) | Set bit 19 to enable all driver logs with log level MFW_D and provide the corresponding hexdump in dmesg logs. |
| Bit 20 | MIF_D | PRINTM(MIF_D,...), DBG_HEXDUMP(MIF_D,...) | Set bit 20 to enable all driver logs with log level MIF_D and provide the corresponding hexdump in dmesg logs. |
| Bit 28 | MENTRY | PRINTM(MENTRY,...), ENTER(), LEAVE() | Set bit 28 to enable all driver logs with API entry and exit. |
| Bit 29 | MWARN | PRINTM(MWARN,...) | Set bit 29 to enable all driver logs with log level MWARN. |
| Bit 30 | MINFO | PRINTM(MINFO,...) | Set bit 30 to enable all driver logs with log level MINFO. |

#### 4.4.1.3 Firmware dump

This section shows how to retrieve the firmware memory from the device and dump it into a file for debugging purposes. A firmware dump can be triggered from the /proc. The driver also dumps the firmware memory in the auto recovery handling when a fatal error occurs for example due to command timeout or TX timeout.

Command to trigger the firmware dump:

```
echo "debug_dump" >> /proc/mwlan/adapter0/config
```

Command output example:

```
[   98.967103] func0: Wakeup device...
[   98.967561] func1: Wakeup device...
[   98.975362] mmlan0:
[   98.975369] |
[   99.927436] wlan: Notify FW dump complete event
[   99.933588] vendor event :0x1
[   99.937144] vendor event :0x0
[  104.162655] vendor event :0x0
```

Command for dual-MAC devices like 88W9098:

```
echo "debug_dump" >> /proc/mwlan/adapter1/config
```

Command output example:

```
[  157.778776] func0: Wakeup device...
[  157.983260] func1: Wakeup device...
[  158.187669] func1: Wakeup device...
[  158.202021] wlan: Notify FW dump complete event
[  158.206572] vendor event :0x1
[  158.209607] vendor event :0x0
```

Command to collect the firmware dump in a file:

```
cat /proc/mwlan/adapter0/fw_dump > file_fw_dump
```

Use the following commands to collect driver dumps in files:

```
cat /proc/mwlan/adapter0/drv_dump > file_drv_dump
cat /proc/mwlan/adapter1/drv_dump > file_drv_dump_2
```

### 4.4.2 Bluetooth debugging

This section details the Bluetooth Protocol and Driver Debugging using *hcidump*, *btmon* and *dmesg* logs.

#### 4.4.2.1 Bluetooth protocol debugging

##### 4.4.2.1.1 Capture the HCI logs using *hcidump*

Follow these steps to capture the HCI logs between the i.MX 8M Quad EVK and NXP-based module using *hcidump* utility.

**Step 1** – Start capturing HCI logs.

Use the following command to start the HCI logs and store these in the file. Use Frontline Bluetooth Software tool to open the log file:

```
hcidump -i hci0 -w sample_hci.log &
[1] 770
HCI sniffer - Bluetooth packet analyzer ver 5.65
btsnoop version: 1 datalink type: 1002
device: hci0 snap_len: 1500 filter: 0x0
```

**Step 2** – Start capturing HCI logs and store in text format.

```
hcidump -i hci0 -Xt | tee sample_hci.txt &
[3] 825
```

**Step 3** – Connect with a Bluetooth Classic/Bluetooth LE device.

Refer to Scan, pair and connect to Bluetooth classic/Bluetooth LE .

**Step 4** – Stop capturing HCI logs.

```
killall hcidump
[1]+  Terminated              hcidump -i hci0 -w sample_hci.log
```

**Step 5** – Copy the *sample_hci.log* file to the host PC.

***Note:*** *hcidump utility may crash during testing. If this occurs, use btmon utility.*

#### 4.4.2.1.2 Capture the HCI Logs using btmon

Follow these steps to capture the HCI logs between the i.MX host platform and wirless SoC using *btmon* utility.

Use the following command to start the HCI logs and store these in the file. You can use Wireshark tool and Wireshark Software to open the log file:

```
btmon -w sample_hci.log &
[1] 951
Bluetooth monitor ver 5.65
= Note:  5.15.32-lts-next+g4c9269301068 (aarch64) 0.329450
= Note: Bluetooth subsystem version 2.22 0.329459
= New Index: 34:6F:24:C0:CA:3A (Primary,UART,hci0) [hci0] 0.329462
= Open Index: 34:6F:24:C0:CA:3A [hci0] 0.329463
= Index Info: 34:6F:24:C0:CA:3A [hci0] 0.329465
@ MGMT Open: bluetoothd (privileged) version 1.21 {0x0001} 0.329473
```

**Step 1** – Start capturing the HCI logs and store the logs in text format.

```
btmon | tee sample_hci.txt &
[1] 780
```

**Connect with a Bluetooth Classic/LE device**

Refer to Scan, pair and connect to Bluetooth classic/Bluetooth LE .

**Step 2** – Stop capturing the HCI logs.

```
killall btmon
[1]+  Done                    btmon -w sample_hci.log
```

**Step 3** – Copy the *sample_hci.log* file to the host PC.

#### 4.4.2.1.3 Extract the Link Key for remote Bluetooth Classic/Bluetooth LE devices

Follow these steps to get the link key for paired Bluetooth Classic/Bluetooth LE devices:

**Step 1** – Get the information for Bluetooth Classic and/or Bluetooth LE devices.

```
cat /var/lib/bluetooth/<DUT BD Address>/<Remote BD Address>/info
```

**Example 1**: Get the Link Key for a Bluetooth device.

```
cat /var/lib/bluetooth/00:50:43:24:34:F7/B4:F5:00:31:CB:4E/info
        [LinkKey]
        Key=7CC2A6C9AA14F799F9E596B90FC973BC
        Type=8
        PINLength=0
```

**Example 2**: Get the Long Term Key for a Bluetooth LE device.

```
cat /var/lib/bluetooth/00:50:43:24:34:F7/D4:18:20:A0:48:5F/info
        [LongTermKey]
        Key=AA3E7062B5B9415F9F27A5010690412A
        Authenticated=0
        EncSize=16
        EDiv=10551
        Rand=730759945871301339
```

#### 4.4.2.2 Bluetooth driver debugging

Command to get and analyze the Bluetooth logs using *dmesg* utility:

```
dmesg | grep Bluetooth
    [    0.225403] Bluetooth: Core ver 2.22
    [    0.225431] Bluetooth: HCI device and connection manager initialized
    [    0.225440] Bluetooth: HCI socket layer initialized
    [    0.225446] Bluetooth: L2CAP socket layer initialized
    [    0.225458] Bluetooth: SCO socket layer initialized
    [    1.775437] Bluetooth: HCI UART driver ver 2.3
    [    1.779894] Bluetooth: HCI UART protocol H4 registered
    [    1.785041] Bluetooth: HCI UART protocol BCSP registered
    [    1.790390] Bluetooth: HCI UART protocol LL registered
    [    1.795534] Bluetooth: HCI UART protocol ATH3K registered
    [    1.800958] Bluetooth: HCI UART protocol Three-wire (H5) registered
    [    1.807322] Bluetooth: HCI UART protocol Broadcom registered
    [    1.813003] Bluetooth: HCI UART protocol QCA registered
    [    2.284553] Bluetooth: RFCOMM TTY layer initialized
    [    2.289447] Bluetooth: RFCOMM socket layer initialized
    [    2.294614] Bluetooth: RFCOMM ver 1.11
    [    2.298396] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
    [    2.303714] Bluetooth: BNEP filters: protocol multicast
    [    2.308948] Bluetooth: BNEP socket layer initialized
    [    2.313918] Bluetooth: HIDP (Human Interface Emulation) ver 1.2
    [    2.319849] Bluetooth: HIDP socket layer initialized
```

## 4.5 Reporting an issue or posting a query

If you need additional support or if you encounter an issue, open a case or community post on the website.

- Create a case for a project. Only the project assignees and NXP can view the case.
- Create a community post that anyone can access.

NXP addresses both cases and community posts.

### 4.5.1 Opening a case

To open a case:

**Step 1** – Click **Sign in/Register** on the top banner of NXP website (Figure 16).


**Figure 16. Click Sign in/Register on NXP website**

**Step 2** – Capture our credentials or click **Create an account** (Figure 17).

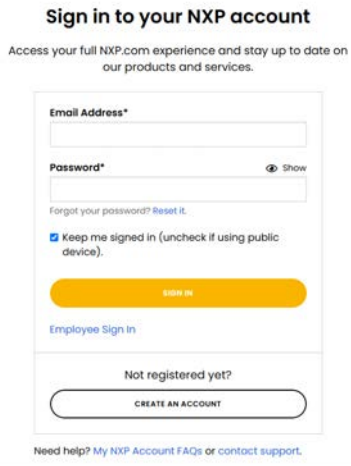



**Figure 17. Capture your credential or create an account**

**Step 2** – Go to NXP Support page (ref.[43]).

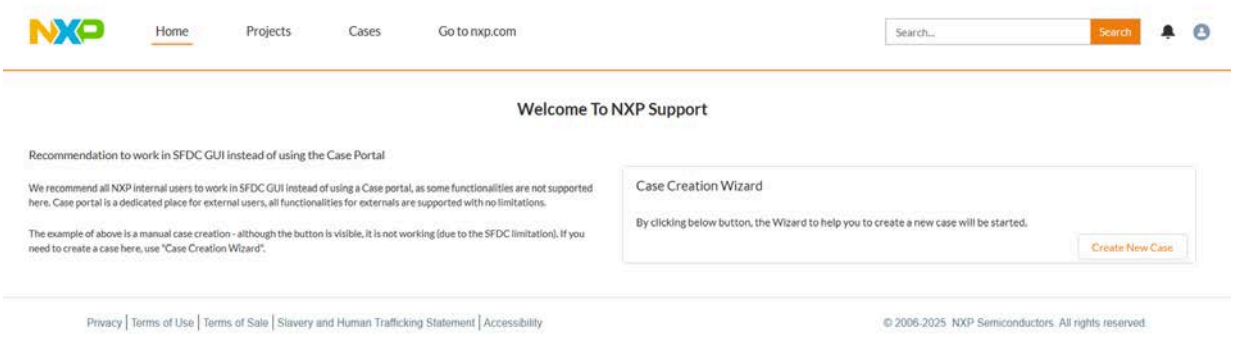



**Figure 18. NXP support page**

**Step 3** – Create a case (Figure 19).

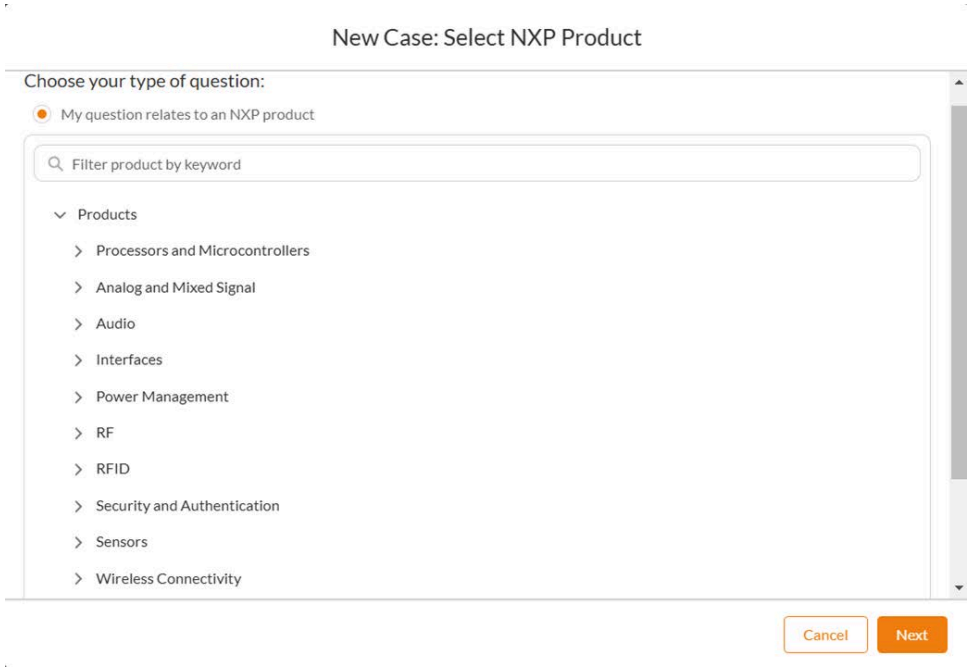



**Figure 19. Create a case**

**Step 4** – Capture information such as product number, topic, and priority (Figure 20).

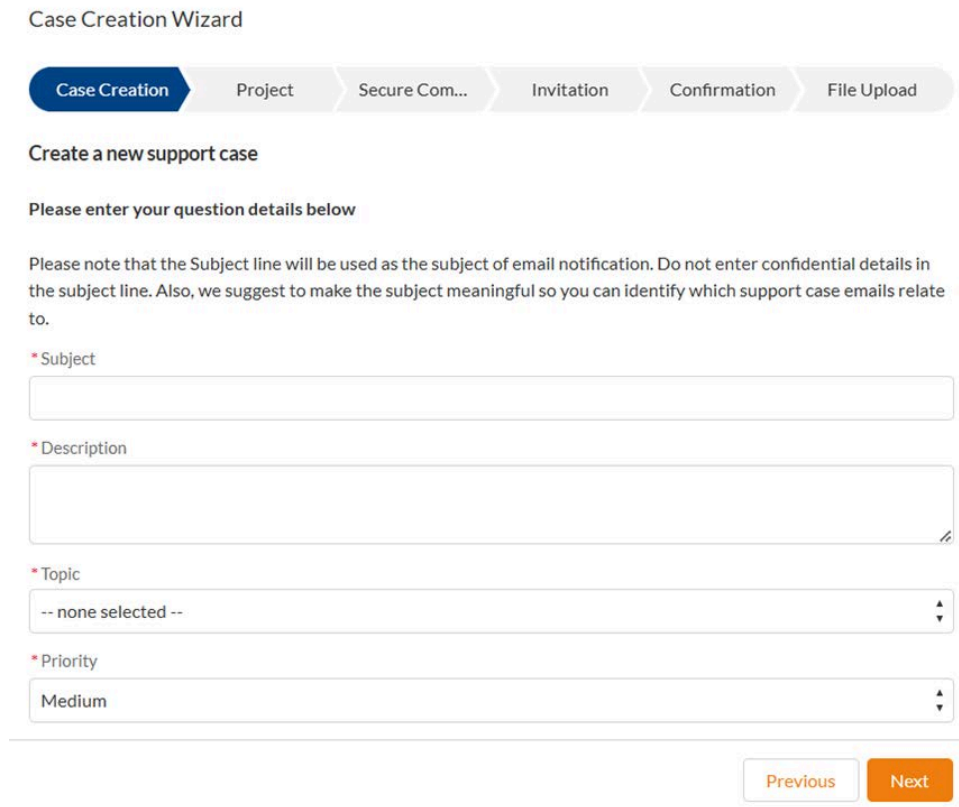



**Figure 20. Capture information about the case**

**Step 5** – Select an existing project or create a project. A project consists of the end application, expected annual volume, end customer name, and more.

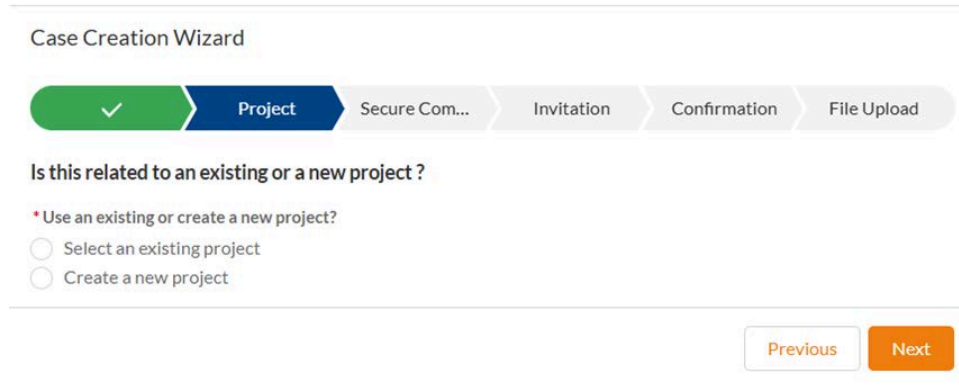



**Figure 21. Select or create a project**

**Step 6** – Capture more details about the case. Provide as much information as possible to speed up the process. Examples of helpful information are:

- Hardware set up
- Host platform
- Wireless SoC
- Driver version
- Linux Kernel, Android Kernel, or FreeRTOS SDK version
- Firmware dumps
- Wireless sniffer logs
- Dmesg logs
- Steps to recreate the issues

NXP sends a response within 24 hours. Depending on the issue complexity or priority, an NXP representative may be assigned to your case.

### 4.5.2 Community post

To open a community post:

**Step 1** – Click **Sign in/Register** on the top banner of NXP website (Figure 22).


**Figure 22. Click Sign in/Register on NXP website**

**Step 2** – Capture our credentials or click **Create an account** (Figure 23).

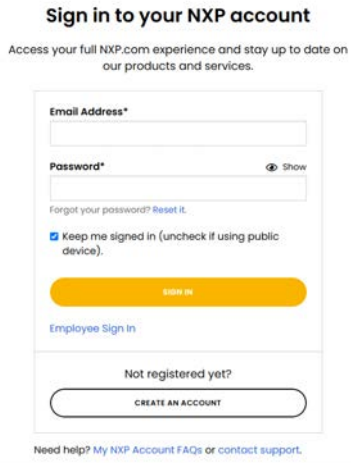



**Figure 23. Capture your credential or create an account**

**Step 3** – Go to **SUPPORT > NXP Community** (ref.[42]).

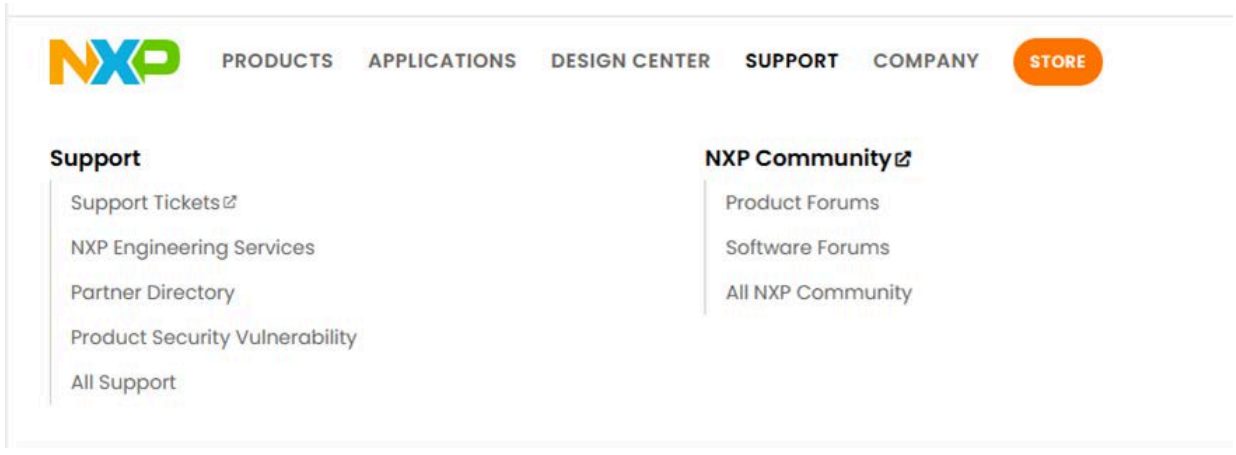



**Figure 24. Access to NXP Community page**

**Step 4** – Click **ASK A QUESTION** (Figure 25).

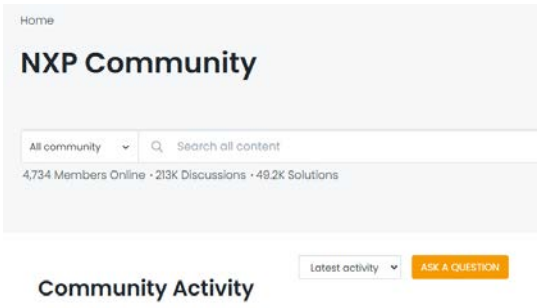



**Figure 25. ASK A QUESTION button on NXP Community page**

**Step 5** – Set the location to *NXP community* > *Forums* > *Product Forums* > *Wireless Connectivity* > *Wi-Fi + Bluetooth + 802.15.4.* (Figure 26).

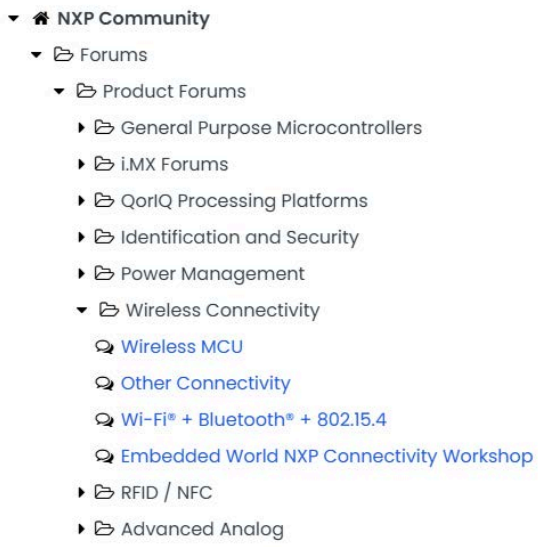



**Figure 26. Set the location**

**Step 6** – Capture more details about your community post. Provide as much information as possible to speed up the process. Examples of helpful information:

- Hardware set up
- Host platform
- Wireless SoC
- Driver version
- Linux Kernel, Android Kernel, or FreeRTOS SDK version
- Firmware dumps
- Wireless sniffer logs
- Dmesg logs
- Steps to recreate the issues

# 5 Connectivity features

## 5.1 Wi-Fi

### 5.1.1 Scan

This section describes the scanning of the visible Access Points using the NXP-based wireless module. Linux operating system provides the standard utilities to show/manipulate wireless devices and their configuration.

#### 5.1.1.1 Using `iw` command

`iw` command is one of the methods used to scan visible access points.

Execute the command to start the scan

```
iw dev mlan0 scan
```

Command output example:

```
[11204.740176] wlan: mlan0 START SCAN
[11207.573646] wlan: SCAN COMPLETED: scanned AP count=1
BSS 44:82:e5:10:2c:38(on mlan0)
    TSF: 11207445411 usec (0d, 03:06:47)
    freq: 2427
    beacon interval: 100 TUs
    capability: ESS Privacy ShortPreamble ShortSlotTime RadioMeasure (0x1431)
    signal: -56.00 dBm
    last seen: 0 ms ago
    SSID: destin
    Supported rates: 1.0* 2.0* 5.5* 6.0 9.0 11.0* 12.0 18.0
    DS Parameter set: channel 4
    Country: in   Environment: Indoor/Outdoor
                  Channels [1 - 13] @ 21 dBm
    TPC report: TX power: 20 dBm
    ERP: NonERP_Present Use_Protection
    RSN:     * Version: 1
             * Group cipher: TKIP
             * Pairwise ciphers: CCMP TKIP
             * Authentication suites: PSK
             * Capabilities: 1-PTKSA-RC 1-GTKSA-RC (0x0000)
    WPA:     * Version: 1
             * Group cipher: TKIP
             * Pairwise ciphers: CCMP TKIP
             * Authentication suites: PSK
    WMM:     * Parameter version 1
             * BE: CW 15-1023, AIFSN 3
             * BK: CW 15-1023, AIFSN 7
             * VI: CW 7-15, AIFSN 2, TXOP 3008 usec
             * VO: CW 3-7, AIFSN 2, TXOP 1504 usec
    Extended supported rates: 24.0 36.0 48.0 54.0
    BSS Load:
              * station count: 4
              * channel utilisation: 52/255
              * available admission capacity: 0 [*32us]
    HT capabilities:
              Capabilities: 0x11ed
                  RX LDPC
                  HT20
                  SM Power Save disabled
                  RX HT20 SGI
                  RX HT40 SGI
                  TX STBC
                  RX STBC 1-stream
                  Max AMSDU length: 3839 bytes
                  DSSS/CCK HT40
         Maximum RX AMPDU length 65535 bytes (exponent: 0x003)
         Minimum RX AMPDU time spacing: 4 usec (0x05)
```

```
         HT TX/RX MCS rate indexes supported: 0-15
    HT operation:
              * primary channel: 4
              * secondary channel offset: no secondary
              * STA channel width: 20 MHz
              * RIFS: 0
              * HT protection: nonmember
              * non-GF present: 1
              * OBSS non-GF present: 1
    * dual beacon: 0
              * dual CTS protection: 0
              * STBC beacon: 0
              * L-SIG TXOP Prot: 0
              * PCO active: 0
              * PCO phase: 0
     Overlapping BSS scan params:
               * passive dwell: 20 TUs
               * active dwell: 10 TUs
               * channel width trigger scan interval: 300 s
               * scan passive total per channel: 200 TUs
               * scan active total per channel: 20 TUs
               * BSS width channel transition delay factor: 5
               * OBSS Scan Activity Threshold: 0.25 %
     Extended capabilities:
               * HT Information Exchange Supported
               * BSS Transition
```

#### 5.1.1.2 Get wpa_supplicant version

Command to get wpa_supplicant version

```
wpa_supplicant -v
```

Command output example:

```
wpa_supplicant v2.11
Copyright (c) 2003-2024, Jouni Malinen <j@w1.fi> and contributors
```

#### 5.1.1.3 Using `wpa_supplicant` and `wpa_cli` commands

You can also use `wpa_supplicant` and `wpa_cli` to scan the visible access points.

Edit the configuration file:

```
nano /etc/wpa_supplicant.conf
```

Content of the configuration file:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1
network={
      key_mgmt=NONE
}
```

Start `wpa_supplicant` in the background

```
killall wpa_supplicant
wpa_supplicant -B -i mlan0 -c /etc/wpa_supplicant.conf
```

Start `wpa_cli` for the scan

```
wpa_cli
```

Command output:

```
wpa_cli v2.11
    Copyright (c) 2004-2024, Jouni Malinen <j@w1.fi> and contributors
    This software may be distributed under the terms of the BSD license.
    See README for more details.
    Selected interface 'mlan0'
    Interactive mode
```

Start the scanning:

```
 > scan
```

Command output example showing the scanned AP count and the SCAN COMPLETE event:

```
OK
[  253.835605] wlan: mlan0 START SCAN
 <3>CTRL-EVENT-SCAN-STARTED
 > [  258.910760] wlan: SCAN COMPLETED: scanned AP count=1
 <3>CTRL-EVENT-SCAN-RESULTS
 <3>CTRL-EVENT-NETWORK-NOT-FOUND
```

Get the scan results:

```
> scan_results
```

Command output example showing one SSID:

```
bssid / frequency / signal level / flags / ssid
44:82:e5:10:2c:38  2442  -93  [WPA-PSK-CCMP+TKIP][WPA2-PSK-CCMP+TKIP][ESS]  destin
```

### 5.1.2 Association

This section shows how to define the configurations for wpa_supplicant to connect with the Access Point using NXP-based wireless module. Linux operating system provides the wpa_supplicant user space utility to connect to the access point using various security settings like WPA2, WPA3, and 802.1x enterprise security.

#### 5.1.2.1 Create the configuration file

Create and edit *wpa_supplicant.conf* configuration file in *etc* directory.

```
nano /etc/wpa_supplicant.conf
```

Configuration file content

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1
network={
      ssid="<value-for-your-network>"
      key_mgmt=<value-for-your-network>
}
```

***Note:*** *Add the value of `ssid` parameter for your network, and the value of `key_mgnt` parameter based on the authentication key management protocols of your network. Refer to wpa_supplicant.conf file (link) for information about the configuration format and supported fields.*

Example of configuration file content:

```
ctrl_interface=/var/run/wpa_supplicant
      ctrl_interface_group=0
      update_config=1
      network={
      ssid="NXP_Demo"
      key_mgmt=NONE
      }
```

#### 5.1.2.2 Use wpa_supplicant to connect with the AP

Connect the device with the Access Point:

```
killall wpa_supplicant
killall hostapd
wpa_supplicant -i mlan0 -Dnl80211 -c /etc/wpa_supplicant.conf
```

Command output example:

```
Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL contro[25602.796098] IPv6: ADDRCONF(NETDEV_UP): mlan0: link is
 not ready
    [25602.922052] wlan: mlan0 START SCAN
    [25607.566402] wlan: SCAN COMPLETED: scanned AP count=18
mlan0: Trying to associate with 1a:2d:8c:55:73:56 (SSID='NXP_Demo' freq=2427 MHz)
    [25607.598754] wlan: Connected to bssid 1a:XX:XX:XX:73:56 successfully
mlan0: Associated with 1a:2d:8c:5[25607.606587] IPv6: ADDRCONF(NETDEV_CHANGE): mlan0:
 link becomes ready
    5:73:56
mlan0: CTRL-EVENT-CONNECTED - Connection to 1a:2d:8c:55:73:56 completed [id=0 id_str=]
```

### 5.1.3 STA security features

This section describe the Wi-Fi security features including the WPA2 and WPA3 SAE security modes. Both the configuration and setup use the open source supplicants.

#### 5.1.3.1 WPA2 security

This section shows the hostapd configuration used to configure WPA2 security and start the Access Point using open source supplicant.

**Create the configuration file**

Configure WPA2 for open source supplicant:

```
nano /etc/hostapd-wpa2.conf
```

Configuration file content:

```
interface=uap0
hw_mode=g
channel=0
country_code=US
ssid=NXP_Demo
auth_algs=1
wpa=2
wpa_key_mgmt=WPA-PSK
rsn_pairwise=CCMP
wpa_passphrase=123456789
```

### Start the Access Point

Run the following command to start the Access Point:

```
killall wpa_supplicant
killall hostapd
ifconfig uap0 192.168.1.2 netmask 255.255.255.0 up
route add default gw 192.168.1.1
hostapd /etc/hostapd-wpa2.conf
```

Command output example:

```
Configuration file: /etc/hostapd-wpa2.conf
rfkill: Cannot open RFKILL control device
uap0: interface state UNINITIALIZED->COUNTRY_UPDATE
ACS: Automatic channel selection started, this may take a bit
[20039.122775] wlan: SCAN COMPLETED: scanned AP count=0
uap0: interface state COUNTRY_UPDATE->ACS
uap0: ACS-STARTED
[20039.603743] wlan: SCAN COMPLETED: scanned AP count=0
[20040.087711] wlan: SCAN COMPLETED: scanned AP count=0
[20040.571527] wlan: SCAN COMPLETED: scanned AP count=0
[20041.051179] wlan: SCAN COMPLETED: scanned AP count=0
uap0: ACS-COMPLETED freq=2412 channel=1
Using interface uap0 with hwaddr 00:50:43:24:84:c4 and ssid "NXP_Demo"
random: Cannot read from /dev/random: Resource temporarily unavai[20041.091736] wlan:
 Starting AP
lable
random: Only 0/20 bytes of strong random data available
r[20041.100179] Get ht_cap from beacon ies: 0xc
andom: Not enough entropy pool available for secure operations
WPA: Not enough entropy in random pool for secure operations - update keys later when the
 first station connects
[20041.120927] wlan: AP started
[20041.130905] Set AC=3, txop=47 cwmin=3, cwmax=7 aifs=1
[20041.138053] Set AC=2, txop=94 cwmin=7, cwmax=15 aifs=1
[20041.145411] Set AC=0, txop=0 cwmin=15, cwmax=63 aifs=3
[20041.152528] Set AC=1, txop=0 cwmin=15, cwmax=1023 aifs=7
uap0: interface state ACS->ENABLED
    uap0: AP-ENABLED
```

#### 5.1.3.1.1 Configure WPA2 to connect with the AP using an open source supplicant

This section describes the configuration for wpa_supplicant to configure WPA2 security and connect with the Access Point.

**Edit the configuration**

Command to configure WPA2 to connect with the AP for an open source supplicant:

```
nano /etc/wpa_supplicant.conf
```

Configuration file content:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1

ap_scan=1
network={
      ssid="NXP_Demo"
      key_mgmt=WPA-PSK
      proto=RSN
      pairwise=CCMP
      group=CCMP
      psk="123456789"
}
```

**Connect with the AP**

Run the following commands to connect the device with the WPA2-based AP:

```
killall wpa_supplicant
killall hostapd
wpa_supplicant -i mlan0 -Dnl80211 -c /etc/wpa_supplicant.conf
```

Command output example:

```
Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL control device
[33831.070886] wlan: mlan0 START SCAN
[33835.714782] wlan: SCAN COMPLETED: scanned AP count=19
mlan0: Trying to associate with 44:82:e5:10:2c:38 (SSID='NXP_Demo' freq=2427 MHz)
[33835.735938] wlan: Connected to bssid 44:XX:XX:XX:2c:38 successfully
mlan0: Associated with 44:82:e5:10:2c:38
mlan0: CTRL-EVENT-SUBNET-STATUS-UPDATE status=0
mlan0: CTRL-EVENT-REGDOM-CHANGE init=COUNTRY_IE type=COUNTRY alpha2=IN
mlan0: WPA: Key negotiation compl[33835.760667] IPv6: ADDRCONF(NETDEV_CHANGE): mlan0:
 link becomes ready
eted with 44:82:e5:10:2c:38 [PTK=[33835.770113] woal_cfg80211_set_rekey_data return:
 gtk_rekey_offload is DISABLE
CCMP GTK=TKIP]
mlan0: CTRL-EVENT-CONNECTED - Connection to 44:82:e5:10:2c:38 completed [id=0 id_str=]
```

### 5.1.3.2 WPA3 security

WPA3 is the next generation of Wi-Fi security with new capabilities that enhance the Wi-Fi protection in personal and enterprise networks. Built on the widely adopted WPA2™, WPA3 adds new features to simplify Wi-Fi security, enable more robust authentication, and deliver increased cryptographic strength for highly sensitive data markets.

All WPA3 networks use the latest security methods, disallow outdated legacy protocols, and require the use of Protected Management Frames (PMF) to maintain resiliency of mission critical networks.

This section describes the configurations for WPA3 SAE and SAE transition mode (backward compatibility with WPA2-PSK) using hostapd and wpa_supplicant for Access Point and Station.

**Configure WPA3 SAE mode for the Access Point**

Create the configuration file:

```
nano /etc/hostapd-wpa3-sae.conf
```

Configuration file content:

```
interface=uap0
driver=nl80211
ssid=NXP_Demo
hw_mode=g
channel=6
wmm_enabled=1
ieee80211n=1
auth_algs=1
wpa=2
wpa_pairwise=CCMP
wpa_passphrase=1234567890
wpa_key_mgmt=SAE
wpa_group_rekey=1800
rsn_pairwise=CCMP
ieee80211w=2
sae_groups=19 20 21
sae_require_mfp=1
sae_anti_clogging_threshold=10
```

### Configure WPA3 SAE transition mode for the Access Point

Create the configuration file:

```
nano /etc/hostapd-wpa3-sae.conf
```

Configuration file content:

```
interface=uap0
driver=nl80211
ssid=NXP_Demo
hw_mode=g
channel=6
wmm_enabled=1
ieee80211n=1
auth_algs=1
wpa=2
wpa_pairwise=CCMP
wpa_passphrase=1234567890
wpa_key_mgmt=WPA-PSK SAE
wpa_group_rekey=1800
rsn_pairwise=CCMP
ieee80211w=1
sae_groups=19 20 21 25 26
sae_require_mfp=1
sae_anti_clogging_threshold=10
```

### Start the Access Point

Run the following command to start the Access Point:

```
killall wpa_supplicant
killall hostapd
ifconfig uap0 192.168.1.2 netmask 255.255.255.0 up
route add default gw 192.168.1.1
hostapd /etc/hostapd-wpa3-sae.conf
```

Command output example:

```
Configuration file: /etc/hostapd-wpa3-sae.conf
rfkill: Cannot open RFKILL control device
[  758.651665] wlan: HostMlme uap0 send deauth/disassoc
Using interface uap0 with hwaddr 70:66:55:26:8b:6b and ssid "NXP_Demo"
random: Only 19/20 bytes of stron[  758.671118] wlan: Starting AP
g random data available
random: [  758.677273] Get ht_cap from beacon ies: 0xc
Not enough entropy pool available[  758.683524] fw doesn't support 11ax
for secure operations
WPA: Not enough entropy in random pool for secure operations - update key[  758.698627]
 wlan: AP started
s later when the first station co[  758.703624] Set AC=3, txop=47 cwmin=3, cwmax=7 aifs=1
nnects
[  758.711651] Set AC=2, txop=94 cwmin=7, cwmax=15 aifs=1
[  758.718678] Set AC=0, txop=0 cwmin=15, cwmax=63 aifs=3
[  758.725701] Set AC=1, txop=0 cwmin=15, cwmax=1023 aifs=7
uap0: interface state UNINITIALIZED->ENABLED
uap0: AP-ENABLED
```

### Configure WPA3 SAE mode to connect with the AP configured with WPA3 SAE

Create the configuration file

```
nano /etc/wpa_supplicant.conf
```

Configuration file content:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1
network={
        ssid="NXP_Demo"
        scan_ssid=1
        key_mgmt=SAE
        proto=RSN
        pairwise=CCMP
        group=CCMP
        psk="1234567890"
        ieee80211w=2
}
```

### Connect with the AP

Commands to connect the device with the AP

```
killall wpa_supplicant
killall hostapd
wpa_supplicant -i mlan0 -Dnl80211 -c /etc/wpa_supplicant.conf
```

Command output example

```
Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL control device
[  145.646509] wlan: mlan0 START SCAN
[  150.026650] wlan: SCAN COMPLETED: scanned AP count=2
mlan0: SME: Trying to authenticate with 00:50:43:23:3c:0e (SSID='NXP_Demo' freq=2437 MHz)
[  150.056535] mlan0:
[  150.056539] wlan: HostMlme Auth received from 00:XX:XX:XX:3c:0e
mlan0: SME: Trying to authenticate with 00:50:43:23:3c:0e (SSID='NXP_Demo' freq=2437 MHz)
[  150.073767] mlan0:
[  150.073771] wlan: HostMlme Auth received from 00:XX:XX:XX:3c:0e
mlan0: PMKSA-CACHE-ADDED 00:50:43:23:3c:0e 0
mlan0: Trying to associate with 00:50:43:23:3c:0e (SSID='NXP_Demo' freq=2437 MHz)
[  150.097756] wlan: HostMlme mlan0 Connected to bssid 00:XX:XX:XX:3c:0e successfully
mlan0: Associated with 00:50:43:23:3c:0e[  150.107410] mlan0:
mlan0: CTRL-EVENT-SUBNET-STATUS-UPDATE status=0
[  150.107416] wlan: Send EAPOL pkt to 00:XX:XX:XX:3c:0e
[  150.126490] mlan0:
[  150.126494] wlan: Send EAPOL pkt to 00:XX:XX:XX:3c:0e
mlan0: WPA: Key negotiation completed with 00:50:43:23:3c:0e [PTK[  150.135343] IPv6:
 ADDRCONF(NETDEV_CHANGE): mlan0: link becomes ready
=CCMP GTK=CCMP]
mlan0: CTRL-EVENT-CONNECTED - Connection to 00:5[  150.148453]
 woal_cfg80211_set_rekey_data return: gtk_rekey_offload is DISABLE
0:43:23:3c:0e completed [id=0 id_str=]
```

***Note:*** *When WPA3-SAE is configured on AP, WPA3 SAE is used on STA to connect with the AP. When WPA3-SAE transition mode is configured on AP, then either WPA3 SAE or WPA2-PSK can be used on STA to connect with the AP.*

#### 5.1.3.3 Wi-Fi Enhanced Open

This section describes the testing procedure of Wi-Fi Enhanced Open security feature. The Wi-Fi Enhanced Open is a new WFA security standard for public networks based on opportunistic wireless encryption (OWE) which provides protections against passive eavesdropping without requiring a password or extra steps to join the network. It integrates established cryptography mechanisms to provide each user with unique individual encryption that protects data exchange between a user device and the Wi-Fi network.

**Configure OWE mode for the Access Point**

***Note:*** *Add the value of* `ssid` *parameter for your network.*

Create the configuration file:

```
nano /etc/hostapd-owe.conf
```

Configuration file content:

```
ctrl_interface=/var/run/hostapd
interface=uap0
driver=nl80211
ssid=<value for your network>
hw_mode=g
channel=6
beacon_int=100
wmm_enabled=1
max_num_sta=10
ieee80211n=1
rts_threshold=2347
fragm_threshold=2346
ht_capab=[HT20+][SHORT-GI-20][RX-STBC1]
wpa=2
wpa_pairwise=CCMP
ieee80211w=2
wpa_key_mgmt=OWE
wpa_group_rekey=60
```

Start the Access Point

Run the following command to start the Access Point:

```
killall wpa_supplicant
killall hostapd
ifconfig uap0 192.168.1.2 netmask 255.255.255.0 up
route add default gw 192.168.1.1
hostapd /etc/hostapd-owe.conf
```

Command output example:

```
rfkill: Cannot open RFKILL control device
uap0: interface state UNINITIALIZED->ENABLED
uap0: AP-ENABLED
```

### Configure OWE mode to connect with the AP configured with OWE

***Note:*** *Add the value of* `ssid` *parameter for your network.*

Create the configuration file

```
nano /etc/wpa_supplicant.conf
```

Configuration file content:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1
network={
        ssid="<value for a network>”
        key_mgmt=OWE
        pairwise=CCMP
        scan_ssid=1
        ieee80211w=2
}
```

### Connect with the AP

Commands to connect the device with the AP:

```
killall wpa_supplicant
killall hostapd
wpa_supplicant -i mlan0 -Dnl80211 -c /etc/wpa_supplicant.conf
```

Command output example with `SSID` = NXP_Demo

```
Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL control device
mlan0: CTRL-EVENT-REGDOM-CHANGE init=BEACON_HINT type=UNKNOWN
mlan0: CTRL-EVENT-REGDOM-CHANGE init=BEACON_HINT type=UNKNOWN
mlan0: CTRL-EVENT-REGDOM-CHANGE init=BEACON_HINT type=UNKNOWN
mlan0: CTRL-EVENT-REGDOM-CHANGE init=BEACON_HINT type=UNKNOWN
mlan0: SME: Trying to authenticate with 36:6f:24:c0:cb:a1 (SSID='NXP_Demo' freq=2437 MHz)
mlan0: Trying to associate with 36:6f:24:c0:cb:a1 (SSID='NXP_Demo' freq=2437 MHz)
mlan0: PMKSA-CACHE-ADDED 36:6f:24:c0:cb:a1 0
mlan0: Associated with 36:6f:24:c0:cb:a1
mlan0: CTRL-EVENT-SUBNET-STATUS-UPDATE status=0
mlan0: WPA: Key negotiation completed with 36:6f:24:c0:cb:a1 [PTK=CCMP GTK=CCMP]
mlan0: CTRL-EVENT-CONNECTED - Connection to 36:6f:24:c0:cb:a1 completed [id=0 id_str=]
```

### Verification from sniffer traces

Association request:

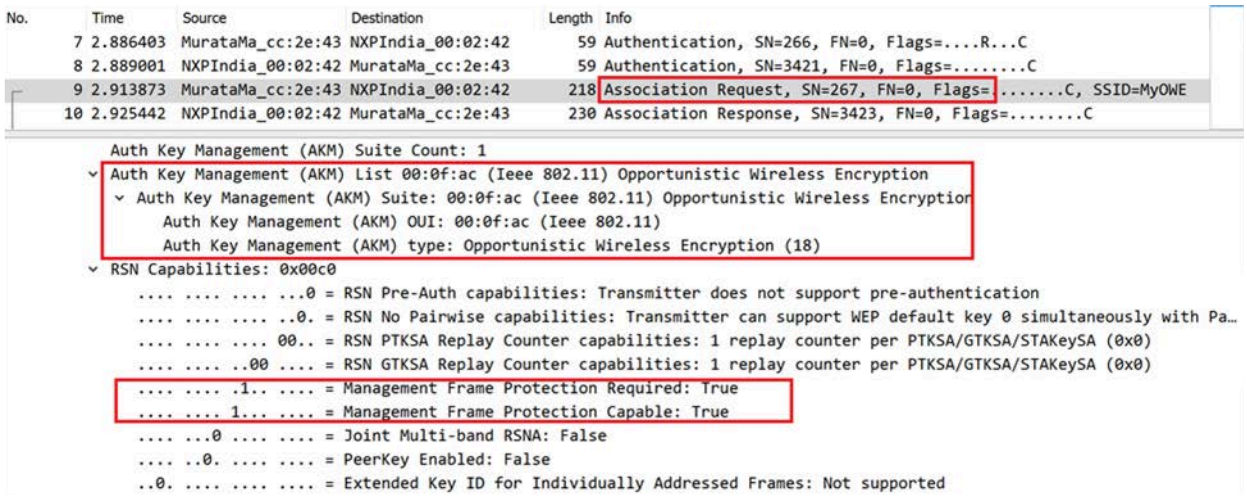



**Figure 27. Association request example**

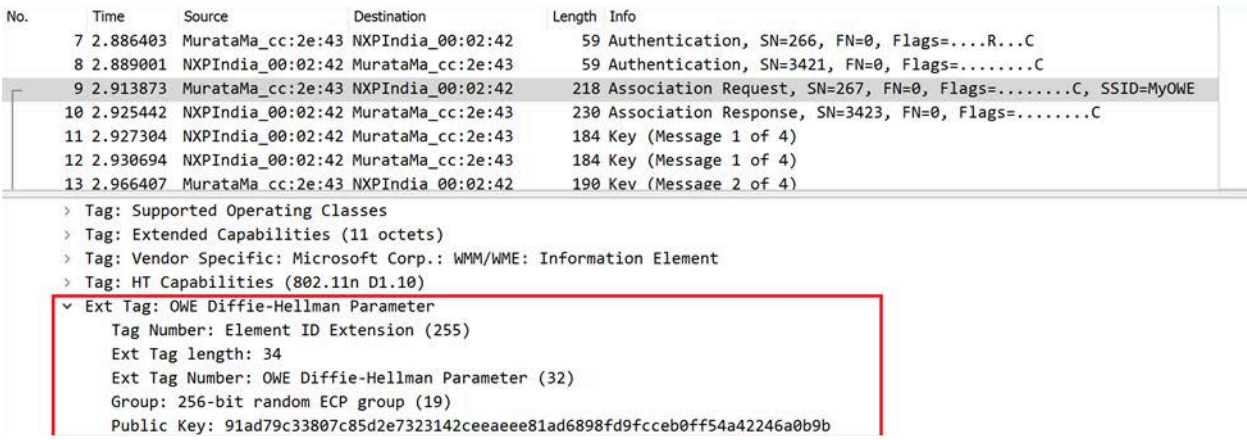



**Figure 28. Association request example (continued)**

Association response:

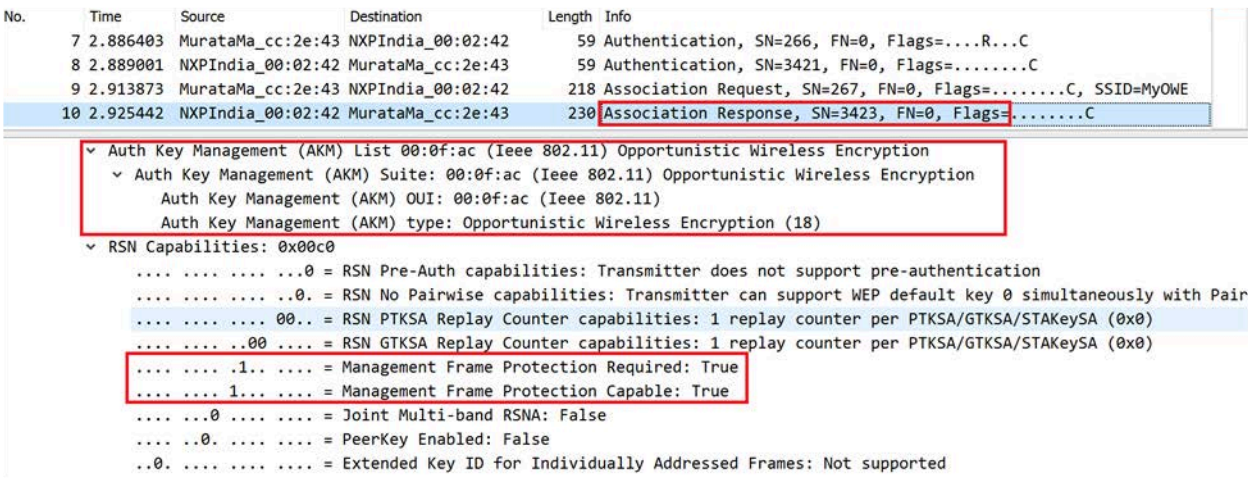



**Figure 29. Association response example**

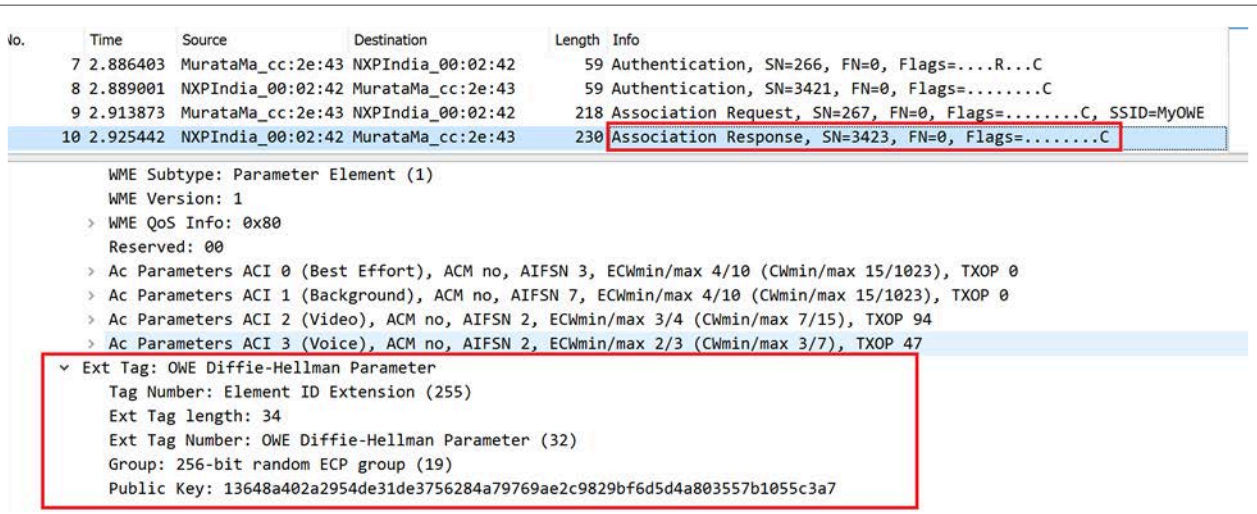



**Figure 30. Association response example (continued)**

### 5.1.4 STA roaming and configuration

This section describes the IEEE 802.11r roaming configurations and enablement. The feature is based on the Fast Transition (FT). FT is faster than normal roaming as it avoids a 4-way handshake when transitioning from one AP to another. Read more about 802.11kvr roaming in ref.[5].

Refer to the following steps to configure and enable the 802.11r roaming.

***Note:*** *wpa_supplicant version 2.7 or above is required for Fast Transition.*

**Step 1** – Enable IEE80211R

- Edit the configuration file located in *hostap/wpa_supplicant/* directory from *wpa_supplicant* source.

```
nano hostap/wpa_supplicant/.config
```

- Set `CONFIG_IEEE80211R` parameter to `y`.

```
CONFIG_IEEE80211R=y
```

- To enable the configuration, re-compile the supplicant binary.

**Step 2** – Create *wpa_supplicant.conf* configuration file.

***Note:*** *Add the value of `ssid` parameter for your network.*

```
nano /etc/wpa_supplicant.conf
```

Configuration file content:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1
ap_scan=1
network={
ssid="<your network>"
key_mgmt=FT-PSK    #Fast Transition Key Management
proto=RSN
pairwise=CCMP
group=CCMP
psk="1234567890"
bgscan="simple:30:-75:120"          #Background scan settings
}
```

**Step 3** - Set the background scanning parameter `bgscan`.

```
bgscan=”simple:<short-scan-interval>:<signal-strength-threshold>:<long-scan-interval>”
```

Where:

**Table 5. Command parameters**

| Parameter | Description |
|---|---|
| `short-scan-interval` | Perform a scan every user defined number of seconds when the signal strength is weaker than the threshold |
| `signal-strength-threshold` | Signal strength from AP (dBm) |
| `long-scan-interval` | Perform a scan every user defined number of seconds when the signal strength is higher than the threshold |

Example of command:

```
bgscan="simple:30:-75:120"
```

In the example above, the scan starts every 30 seconds when the signal strength from the current AP is below -75 dBm. If the signal strength is above -75dBm, the scan starts every 120 seconds.

**Step 4** – Run `wpa_supplicant`.

```
wpa_supplicant -B -Dnl80211 -imlan0 -c/etc/wpa_supplicant.conf
```

Command output example:

```
Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL contr device
mlan0: SME: Trying to authenticate with 5c:df:89:0f:32:7c (SSID='NXP_V10' freq=5180 MHz)
mlan0: Trying to associate with 5c:df:89:0f:32:7c (SSID='NXP_V10' freq=5180 MHz)
mlan0: Associated with 5c:df:89:0f:32:7c
mlan0: CTRL-EVENT-SUBNET-STATUS-UPDATE status=0
mlan0: CTRL-EVENT-REGDOM-CHANGE init=COUNTRY_IE type=COUNTRY alpha2=IN
mlan0: WPA: Key negotiation completed with 5c:df:89:0f:32:7c [PTK=CCMP GTK=CCMP]
mlan0: CTRL-EVENT-CONNECTED - Connection to 5c:df:89:0f:32:7c completed [id=0 id_str=]
```

### 5.1.5 STA provisioning protocols

#### 5.1.5.1 Wi-Fi easy connect (DPP)

This section describes:

- The test procedure of Wi-Fi easy connect (DPP) using *wpa_supplicant*
- How to configure the Wi-Fi devices in STA and AP modes
- How to use DPP feature to connect the STA and AP devices

The Wi-Fi easy connect feature provides a simple and secure method to provision and connect Wi-Fi devices to a network without entering a password.

Refer to the following steps to configure, enable and test Wi-Fi easy connect feature.

**Step 1** – Enable DPP

- Edit the configuration file located in *hostap/wpa_supplicant/* directory from *wpa_supplicant* source

```
nano hostap/wpa_supplicant/.config
```

- Set `CONFIG_DPP` parameter to `y`

```
CONFIG_DPP=y
```

- Edit the configuration file located in *hostap* directory

```
nano hostap/.config
```

- Set `CONFIG_DPP` parameter to `y`

```
CONFIG_DPP=y
```

- Set `CONFIG_INTERWORKING` parameter to `y`

```
CONFIG_INTERWORKING=y
```

- To enable the configuration, re-compile the supplicant and hostapd binaries

**Step 2** – Create the configuration files for hostapd and wpa_supplicant

Create the configuration file for wpa_supplicant (Wi-Fi device set as STA)

```
nano /etc/wpa_supplicant.conf
```

Configuration file content:

```
ctrl_interface=/var/run/wpa_supplicant
update_config=1
dpp_config_processing=2
```

Create the configuration file for hostapd (Wi-Fi device set as STA)

```
nano /etc/hostapd.conf
```

Configuration file content:

```
ctrl_interface=/var/run/hostapd
interface=uap0
driver=nl80211
hw_mode=g
channel=1
ieee80211n=1
ssid=unconfigured
```

**Step 3** – Test procedure

**Test - DUT as Enrollee, Initiator(Authentication), enrolled as STA**

Test setup:

- The DUT acts as initiator and enrollee
- The DUT is configured as STA and STA using wpa_supplicant
- The compliance test tool CCT1 acts as configurator
- The compliance test tool CCT2 acts as responder and enrollee
- CCT2 is configured as AP and AP using hostapd

During the test, the QR code test steps (values in ()) represent the command returned value. The command in <> represents the optional value.

- Start `wpa_supplicant` on DUT and CTT1 (configurator)

```
wpa_supplicant -i mlan0 -D nl8021 -c /etc/wpa_supplicant.conf -B
```

Command output:

```
Successfully initialized wpa_supplicant
```

- Start the hostapd on CTT2 (AP)

```
hostapd /etc/hostapd.conf -B
```

Command output:

```
uap0: interface state UNINITIALIZED->ENABLED
uap0: AP-ENABLED
```

- Start wpa_cli in interactive mode on DUT(STA) and CTT1 (configurator)

```
wpa_cli
```

Command output:

```
wpa_cli v2.11
Copyright (c) 2004-2024, Jouni Malinen <j@w1.fi> and contributors
This software may be distributed under the terms of the BSD license.
See README for more details.
Selected interface 'mlan0'
Interactive mode
```

- Start hostapd_cli in interactive mode on CTT2 (AP)

```
hostapd_cli
```

Command output:

```
hostapd_cli v2.11
Copyright (c) 2004-2024, Jouni Malinen <j@w1.fi> and contributors
This software may be distributed under the terms of the BSD license.
See README for more details.
Selected interface 'uap0'
Interactive mode
```

- Add a Configurator and generate QR code on CTT1 (configurator).

```
> DPP_CONFIGURATOR_ADD
1
```

Returned `1` in above command is config id.

```
> SET dpp_configurator_params " conf=sta-dpp configurator=1"
OK
> DPP_BOOTSTRAP_GEN type=qrcode chan=81/1 mac=34:6f:24:c0:cc:36
1
```

***Note:*** *MAC address of CTT1 should input in above command and returned `1` is bootstrap info id which require to QR code*

```
> DPP_BOOTSTRAP_GET_URI 1
DPP:C:81/1;M:346f24c0cc36;K:MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgAC2/+qun/54bZKrB2cQCGGJKZ/
tdn2mU7VURZumqrik0s=;;
```

***Note:*** *This QR code will be use on DUT with command `DPP_QR_CODE`.*

```
> DPP_LISTEN 2412 role=configurator
OK
```

- Authenticate the DUT on DUT(STA)

```
> DPP_QR_CODE DPP:C:81/1;M:346f24c0cc36;K:MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgAC2/
+qun/54bZKrB2cQCGGJKZ/tdn2mU7VURZumqrik0s=;;
1
```

***Note:*** *On successfully adding QR Code, a bootstrapping info id is returned as shown `1` in above command and should input in below command `DPP_AUTH_INIT`*

```
> DPP_AUTH_INIT peer=1 role=enrollee
OK
```

Command output on the DUT:

```
<3>DPP-TX dst=34:6f:24:c0:cc:36 freq=2412 type=0¶
<3>DPP-TX-STATUS dst=34:6f:24:c0:cc:36 freq=2412 result=SUCCESS¶
<3>DPP-RX src=34:6f:24:c0:cc:36 freq=2412 type=1¶
<3>DPP-AUTH-DIRECTION mutual=0¶
<3>DPP-TX dst=34:6f:24:c0:cc:36 freq=2412 type=2¶
<3>DPP-TX-STATUS dst=34:6f:24:c0:cc:36 freq=2412 result=SUCCESS¶
<3>DPP-AUTH-SUCCESS init=1¶
<3>GAS-QUERY-START addr=34:6f:24:c0:cc:36 dialog_token=214 freq=2412¶
<3>GAS-QUERY-DONE addr=34:6f:24:c0:cc:36 dialog_token=214 freq=2412
status_code=0 result=SUCCESS¶
<3>DPP-CONF-RECEIVED ¶
<3>DPP-CONFOBJ-AKM dpp¶
<3>DPP-CONFOBJ-SSID test¶
<3>DPP-CONNECTOR
eyJ0eXAiOiJkcHBDb24iLCJraWQiOiJpUFFIMzJ2dEpFZkowWWJCd0k4SE5FNWxOOUkzNGJ
RUi04aFJyLW1valBZIiwiYWxnIjoiRVMyNTYifQ.eyJncm91cHMiOlt7Imdyb3VwSWQiOiI
qIiwibmV0Um9sZSI6InN0YSJ9XSwibmV0QWNjZXNzS2V5Ijp7Imt0eSI6IkVDIiwiY3J2Ij
oiUC0yNTYiLCJ4IjoiTWpGLUJINElPcGs3RmNUS1RyN3RpakVNVndSQ0JxLWl3WWJyVXFmL
WxEYyIsInkiOiJIMFVsS0ViamV6X1BFVHVXTHR2TkRpeDJONnRITDNiejRHdHBxQ3doRGww
In19.vUM8zVNdBCJ2vgmsDnI6zvOkZZYKp0xFFOxLYR7ahb5do_s0UqjiCGYdWI-
q1hgAXzXfBV6KsKIwK59LlTgQQA¶
<3>DPP-C-SIGN-KEY
3039301306072a8648ce3d020106082a8648ce3d03010703220002c580f4e7c54239f95
5443bae83cca24d978f73e73608680aa7cedeaa2cb20912¶
<3>DPP-NET-ACCESS-KEY
307702010104209373 75ff78d21376831fa1c6aa9391aa57214cf6d6ecd3b83b1a8ad40
0c52296a00a06082a8648ce3d030107a1440342000432317e047e083a993b15c4ca4ebe
ed8a310c57044206afa2c186eb52a7fe94371f45252846e37b3fcf113b962edbcd0e2c7
637ab472f76f3e06b69a82c210e5d¶
<3>CTRL-EVENT-NETWORK-ADDED 0¶
<3>DPP-NETWORK-ID 0¶
<3>CTRL-EVENT-SCAN-STARTED ¶
```

**Figure 31. Command output on the DUT**

Command output on CTT1:

```
<3>DPP-RX src=34:6f:24:c0:ca:a1 freq=2412 type=0
<3>DPP-TX dst=34:6f:24:c0:ca:a1 freq=2412 type=1
<3>DPP-TX-STATUS dst=34:6f:24:c0:ca:a1 freq=2412 result=SUCCESS
<3>DPP-RX src=34:6f:24:c0:ca:a1 freq=2412 type=2
<3>DPP-AUTH-SUCCESS init=0
```

Stop scanning on the DUT:

```
> DPP_STOP_LISTEN
OK
```

- Generate the QR Code and get URI on CTT2

```
> DPP_BOOTSTRAP_GEN type=qrcode chan=81/1 mac=4a:e7:da:25:61:c2
1
```

***Note:*** *MAC address of CTT2 should input in above command and returned `1` is bootstrap info id which require to get QR code in below command.*

```
> DPP_BOOTSTRAP_GET_URI 1
DPP:C:81/1;M:4ae7da2561c2;K:MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgACVx+FI/
m3+14cb4jx2tc66gopHv44bY2Q65W0OZJtgDI=;;>
```

- Enter the QR Code in the configurator CTT1 and authenticate:

```
> DPP_QR_CODE DPP:C:81/1;M:4ae7da2561c2;K:MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgACVx+FI/
m3+14cb4jx2tc66gopHv44bY2Q65W0OZJtgDI=;;
2
```

***Note:*** *On successfully adding QR Code, a bootstrapping info id is returned as shown 2 in above command and should input in below command DPP_AUTH_INIT*

```
> DPP_AUTH_INIT peer=2 conf=ap-dpp configurator=1
OK
```

Command output on CTT2:

```
<3>DPP-AUTH-SUCCESS·init=0¶
<3>GAS-QUERY-START·addr=34:6f:24:c0:cc:36·dialog_token=0·freq=2412¶
<3>GAS-QUERY-DONE·addr=34:6f:24:c0:cc:36·dialog_token=0·freq=2412·
status_code=0·result=SUCCESS¶
<3>DPP-CONF-RECEIVED·¶
<3>DPP-CONFOBJ-AKM·dpp¶
<3>DPP-CONFOBJ-SSID·test¶
<3>DPP-CONNECTOR·
eyJ0eXAiOiJkcHBDb24iLCJraWQiOiJpUFFIMzJ2dEpFZkowWWJCd0k4SE5FNWxOOUkzNGJ
RUi04aFJyLW1valBZIiwiYWxnIjoiRVMyNTYifQ.eyJncm91cHMiOlt7Imdyb3VwSWQiOiI
qIiwibmV0Um9sZSI6ImFwIn1dLCJuZXRBY2Nlc3NLZXkiOnsia3R5IjoiRUMiLCJjcnYiOi
JQLTI1NiIsIngiOiJ2WkJLejFLdy1JbVRyY2pIQTJ1bVgzX2t3VEJEM2ZqU2hSVjVQWldtQ
3NVIiwieSI6IkpJUC1qWWlNLV9LRFVFUVI5U2xRVlFJcEZBb2M3V29Nd2JsSnQ1bmt2TEUi
fX0.pm1t8DVJQJJhrtaX5xOlOVPHJDIit8iBjZpGhnbhVVTYRf-
vYQIPRC80nB3zf9UehxlNgBcaxM1dlu9_R6AV_w¶
<3>DPP-C-SIGN-KEY·
3039301306072a8648ce3d020106082a8648ce3d03010703220002c580f4e7c54239f95
5443bae83cca24d978f73e73608680aa7cedeaa2cb20912¶
<3>DPP-NET-ACCESS-KEY·
30770201010420b7e039bc5a524c4db98d9ac6e7817ee9a32e955e4fd4014227b7cfe67
acb3a02a00a06082a8648ce3d030107a14403420004bd904acf52b0f88993adc8c7036b
a65f7fe4c13043ddf8d28515793d95a60ac52483fe8d888cfbf283504411f5295055022
9140a1ced6a0cc1b949b799e4bcb1¶
```

**Figure 32. Command output on CCT2**

- Update AP configuration on CTT2:
  First disable AP:

```
> disable
<3>AP-DISABLED
OK
```

Update AP parameters:

```
> set ssid test
OK
> set wpa 2
OK
> set wpa_key_mgmt DPP
OK
> set ieee80211w 2
OK
> set rsn_pairwise CCMP
OK
```

Set the `connector key`, `csign`, and `netaccess` values dumped on CTT2 console:

```
>·set·dpp_connector·
eyJ0eXAiOiJkcHBDb24iLCJraWQiOiJpUFFIMzJ2dEpFZkowWWJCd0k4SE5FNWxOOUkzNGJ
RUi04aFJyLW1valBZIiwiYWxnIjoiRVMyNTYifQ.eyJncm91cHMiOlt7Imdyb3VwSWQiOiI
qIiwibmV0Um9sZSI6ImFwIn1dLCJuZXRBY2Nlc3NLZXkiOnsia3R5IjoiRUMiLCJjcnYiOi
JQLTI1NiIsIngiOiJ2WkJLejFLdy1JbVRyY2pIQTJ1bVgzX2t3VEJEM2ZqU2hSVjVQWldtQ
3NVIiwieSI6IkpJUC1qWWlNLV9LRFVFUVI5U2xRVlFJcEZBb2M3V29Nd2JsSnQ1bmt2TEUi
fX0.pm1t8DVJQJJhrtaX5xOlOVPHJDIit8iBjZpGhnbhVVTYRf-
vYQIPRC80nB3zf9UehxlNgBcaxM1dlu9_R6AV_w¶
OK¶
¶
>·set·dpp_csign·
3039301306072a8648ce3d020106082a8648ce3d03010703220002c580f4e7c54239f95
5443bae83cca24d978f73e73608680aa7cedeaa2cb20912¶
OK¶
¶
>·set·dpp_netaccesskey·
30770201010420b7e039bc5a524c4db98d9ac6e7817ee9a32e955e4fd4014227b7cfe67
acb3a02a00a06082a8648ce3d030107a14403420004bd904acf52b0f88993adc8c7036b
a65f7fe4c13043ddf8d28515793d95a60ac52483fe8d888cfbf283504411f5295055022
9140a1ced6a0cc1b949b799e4bcb1¶
OK¶
```

**Figure 33. Set `connector key`, `csign`, and `netaccess` values**

Re-enable the AP after updates:

```
> enable
<3>AP-ENABLED
OK
> <3>DPP-RX src=34:6f:24:c0:ca:a1 freq=2412 type=5
<3>DPP-TX dst=34:6f:24:c0:ca:a1 freq=2412 type=6 status=0
<3>DPP-TX-STATUS dst=34:6f:24:c0:ca:a1 result=SUCCESS
<3>AP-STA-CONNECTED 34:6f:24:c0:ca:a1
<3>EAPOL-4WAY-HS-COMPLETED 34:6f:24:c0:ca:a1
```

The connection between the DUT (STA) and CTT2 (AP) is successful.

#### 5.1.5.2 Wi-Fi protected setup (WPS)

Wi-Fi protected setup (WPS) is used to establish fast connections between wireless devices and a router. WPS supports two methods (pushing a button or entering a PIN) to configure a network and enable security. This section describes how to verify WPS functionality in STA mode.

**Method 1 – Entering a PIN**

The method requires to enter or to generate a PIN: enter a PIN at the "representant" of the network (usually the AP), or generate a PIN from the client.

**Step 1** - Configure the external AP in the router homepage.

Example of configuration:

```
Security: WPA2-AES
PSK:12345678
WPS PIN: 86146685
```

**Step 2** - Load the drivers.

```
modprobe moal mod_para=nxp/wifi_mod_para.conf ps_mode=1 auto_ds=1
```

**Step 3** - Create *wpa_supplicant.conf.*

Example of wpa_supplicant.conf file:

```
ctrl_interface=/var/run/wpa_supplicant
```

**Step 4** - Start wpa_supplicant on STA (DUT).

```
wpa_supplicant -i mlan0 -D nl80211 -c <path_to_file>/wpa_supplicant.conf
```

**Step 5** - Start wpa_cli.

```
wpa_cli
```

**Step 6** - Scan and display the results of the external AP.

```
> scan
> scan_result
```

Command output example:

```
> scan
OK
[  310.177436] wlan: mlan0 START SCAN
<3>CTRL-EVENT-SCAN-STARTED
> [  313.340574] wlan: SCAN COMPLETED: scanned AP count=19
<3>CTRL-EVENT-REGDOM-CHANGE init=BEACON_HINT type=UNKNOWN
<3>CTRL-EVENT-REGDOM-CHANGE init=BEACON_HINT type=UNKNOWN
<3>CTRL-EVENT-SCAN-RESULTS
<3>WPS-AP-AVAILABLE
<3>CTRL-EVENT-NETWORK-NOT-FOUND
>
> scan_results
bssid / frequency / signal level / flags / ssid
bc:a5:11:a2:b3:4f       5180    -50     [WPA2-PSK-CCMP][WPS][ESS]       Roaming_NXP
b0:39:56:93:86:5d       5180    -32     [WPA2-PSK-CCMP][WPS][ESS]
7c:10:c9:02:da:48       2437    -41     [WPA2-PSK-CCMP][WPS][ESS]       ASUS_AP_2G
dc:33:3d:ab:e9:f8       2437    -50     [WPA2-PSK-CCMP][WPS][ESS]       DRCS_EX_AP_2G
```

**Step 7** - Select the PIN method on the STA (DUT).

***Note:*** *Use and match the pin configured in the external AP.*

```
> wps_pin any <PIN>
```

Command output example:

```
> wps_pin any 86146685
[  419.040560] wlan: mlan0 START SCAN
86146685
<3>CTRL-EVENT-NETWORK-ADDED 0
<3>WPS-PIN-ACTIVE
[  422.482880] wlan: HostMlme mlan0 send auth to bssid 7c:XX:XX:XX:da:48
<3>WPS-AP-AVAILABLE-AUTH
<3>SME: Trying to authenticate with 7c:10:c9:02:da:48 (SSID='ASUS_AP_2G' freq=2437 MHz)
> [  422.497937] mlan0:
[  422.497941] wlan: HostMlme Auth received from 7c:XX:XX:XX:da:48
<3>Trying to associate with 7c:10:c9:02:da:48 (SSID='ASUS_AP_2G' freq=2437 MHz)
> [  422.518977] wlan: HostMlme mlan0 Connected to bssid 7c:XX:XX:XX:da:48 successfully
<3>Associated with 7c:10:c9:02:d[  422.528139] mlan0:
<3>CTRL-EVENT-EAP-STATUS status='started' parameter=''
<3>CTRL-EVENT-REGDOM-CHANGE init=COUNTRY_IE type=COUNTRY alpha2=US
<3>EAPOL-RX 7c:10:c9:02:da:48 42
<3>CTRL-EVENT-EAP-PROPOSED-METHOD vendor=14122 method=1
<3>CTRL-EVENT-EAP-STATUS status='accept proposed method' parameter='WSC'
<3>CTRL-EVENT-EAP-METHOD EAP vendor 14122 method 1 (WSC) selected
<3>EAPOL-RX 7c:10:c9:02:da:48 469
> [  422.600180] mlan0:
[  422.600188] wlan: Send EAPOL pkt to 7c:XX:XX:XX:da:48
<3>EAPOL-RX 7c:10:c9:02:da:48 21[  422.616101] mlan0:
0
> [  422.616106] wlan: Send EAPOL pkt to 7c:XX:XX:XX:da:48
<3>EAPOL-RX 7c:10:c9:02:da:48 13[  422.635341] mlan0:
8
> [  422.635347] wlan: Send EAPOL pkt to 7c:XX:XX:XX:da:48
<3>EAPOL-RX 7c:10:c9:02:da:48 170
<3>WPS-CRED-RECEIVED
<3>WPS-SUCCESS
[  422.677504] mlan0:
[  422.677511] wlan: Send EAPOL pkt to 7c:XX:XX:XX:da:48
<3>EAPOL-RX 7c:10:c9:02:da:48 42[  422.706640] wlan: Received disassociation request on
 mlan0, reason: 3
<3>CTRL-EVENT-EAP-STATUS sta[  422.715775] wlan: REASON: (Deauth) Sending STA is leaving
 (or has left) IBSS or ESS
tus='completion' parameter='failure'
freq=2437 MHz)
<3>Trying to associate with 7c:10:c9:02:da:48 (SSID='ASUS_AP_2G' freq=2437 MHz)
> [  422.780991] wlan: HostMlme mlan0 Connected to bssid 7c:XX:XX:XX:da:48 successfully
 [PTK=CCM[  422.849995] woal_cfg80211_set_rekey_data return: gtk_rekey_offload is DISABLE
P GTK=CCMP]
<3>CTRL-EVENT-CONNECTED - Connection to 7c:10:c9:02:da:48 completed [id=0 id_str=]
<3>EAPOL-RX 7c:10:c9:02:da:48 13[  424.870780] mlan0:
1
> [  424.870795] wlan: Send EAPOL pkt to 7c:XX:XX:XX:da:48
<3>WPA: Group rekeying completed[  424.880877] woal_cfg80211_set_rekey_data return:
 gtk_rekey_offload is DISABLE
 with 7c:10:c9:02:da:48 [GTK=CCMP]
```

***Note:*** *Registration with the external AP shows as* `WPA: Group rekeying completed` *in the command output.*

**Step 8** - Verify the status.

```
gt; status
```

Command output example:

```
> status
bssid=7c:10:c9:02:da:48
freq=2437
ssid=ASUS_AP_2G
id=0
mode=station
pairwise_cipher=CCMP
group_cipher=CCMP
key_mgmt=WPA2-PSK
wpa_state=COMPLETED
p2p_device_address=2c:4c:c6:f4:67:9e
address=2c:4c:c6:f4:67:9e
uuid=25d271cf-1787-5d3a-9903-595488876c53
```

### Method 2 - Using a push button

The user has to push a virtual or actual button on the DUT client and on the external AP.

**Step 1** – Configure the external AP in the router homepage.

Example of configuration:

```
Security: WPA2-AES
PSK:12345678
WPS: Push-Button method
```

**Step 2** - Load the drivers.

```
modprobe moal mod_para=nxp/wifi_mod_para.conf ps_mode=1 auto_ds=1
```

**Step 3** - Create *wpa_supplicant.conf*.

Content of *wpa_supplicant.conf* file:

```
ctrl_interface=/var/run/wpa_supplicant
```

**Step 4** - Start wpa_supplicant on the STA.

```
wpa_supplicant -i mlan0 -D nl80211 -c <path_to_file>/wpa_supplicant.conf
```

**Step 5** - Start *wpa_cli*.

```
wpa_cli
```

**Step 6** - Scan and display the results of the external AP.

```
> scan
> scan_result
```

Command output example:

```
> scan
OK
[   29.163406] wlan: mlan0 START SCAN
<3>CTRL-EVENT-SCAN-STARTED
> [   32.327461] wlan: SCAN COMPLETED: scanned AP count=19
<3>WPS-AP-AVAILABLE
<3>CTRL-EVENT-NETWORK-NOT-FOUND
>
> scan[   33.764308] VGEN1: disabling
[   33.769255] VGEN6: disabling
_results
bssid / frequency / signal level / flags / ssid
dc:33:3d:ab:e9:fe       5180    -59     [WPA2-PSK-CCMP][WPS][ESS]
7c:10:c9:02:da:48       2437    -38     [WPA2-PSK-CCMP][WPS][ESS]       ASUS_AP_2G
dc:33:3d:ab:e9:f8       2437    -55     [WPA2-PSK-CCMP][WPS][ESS]       DRCS_EX_AP_2G
bc:0f:9a:70:1f:69       2412    -44     [WPA2-PSK-CCMP][WPS][ESS]       Roaming_NXP
```

**Step 7** - Select PBC method on the STA (DUT). Push the WPS button on the external AP.

```
> wps_pbc any
```

Command output example:

```
> wps_pbc any
OK
<3>CTRL-EVENT-NETWORK-[   58.737389] wlan: mlan0 START SCAN
ADDED 0
<3>WPS-PBC-ACTIVE
<3>CTRL-EVENT-SCAN-STARTED
> [   62.169134] wlan: SCAN COMPLETED: scanned AP count=14
<3>CTRL-EVENT-REGDOM-CHANGE init=BEACON_HINT type=UNKNOWN
<3>CTRL-EVENT-SCAN-RESULTS
>[   62.181751] wlan: HostMlme mlan0 send auth to bssid 7c:XX:XX:XX:da:48
<3>WPS-AP-AVAILABLE-PBC
<3[   62.191727] mlan0:
>SME: Trying to authenticate with[   62.191732] wlan: HostMlme Auth received from
 7c:XX:XX:XX:da:48
 7c:10:c9:02:da:48 (SSID='ASUS_AP_2G' freq=2437 MHz)
<3>Trying to associate with 7c:10:c9:02:da:48 (SSID='ASUS_AP_2G' freq=2437 MHz)
> [   62.229604] wlan: HostMlme mlan0 Connected to bssid 7c:XX:XX:XX:da:48 successfully
<3>Associated with 7c:10:c9:02:d[   62.238715] mlan0:
a:48
<3>CTRL-EVENT-SUBNET-STA[   62.238727] wlan: Send EAPOL pkt to 7c:XX:XX:XX:da:48
TUS-UPDATE status=0
<3>EAPOL-RX 7c:10:c9:02:da:48 42
<3>CTRL-EVENT-EAP-STARTED EAP auth[   62.258414] mlan0:
entication started
<3>CTRL-EV[   62.258430] wlan: Send EAPOL pkt to 7c:XX:XX:XX:da:48
ENT-EAP-STATUS status='started' parameter=''
<3>CTRL-EVENT-REGDOM-CHANGE init=COUNTRY_IE type=COUNTRY alpha2=US
<3>EAPOL-RX 7c:10:c9:02:da:48 42
<3>CTRL-EVENT-EAP-PROPOSED-METHOD vendor=14122 method=1
<3>CTRL-EVENT-EAP-STATUS status='accept proposed method' parameter='WSC'
<3>CTRL-EVENT-EAP-METHOD EAP vendor 14122 method 1 (WSC) selected
<3>EAPOL-RX 7c:10:c9:02:da:48 469
> [   62.312782] mlan0:
[   62.312790] wlan: Send EAPOL pkt to 7c:XX:XX:XX:da:48
<3>EAPOL-RX 7c:10:c9:02:da:48 21[   62.332987] mlan0:
0
> [   62.332994] wlan: Send EAPOL pkt to 7c:XX:XX:XX:da:48
<3>EAPOL-RX 7c:10:c9:02:da:48 13[   62.347806] mlan0:
8
> [   62.347811] wlan: Send EAPOL pkt to 7c:XX:XX:XX:da:48
<3>EAPOL-RX 7c:10:c9:02:da:48 170
<3>WPS-CRED-RECEIVED
<3>WPS-SUCCESS
[   62.386464] mlan0:
> [   62.386471] wlan: Send EAPOL pkt to 7c:XX:XX:XX:da:48
<3>EAPOL-RX 7c:10:c9:02:da:48 42[   62.402443] wlan: Received disassociation request on
 mlan0, reason: 3
<3>CTRL-EVENT-EAP-STATUS sta[   62.411547] wlan: REASON: (Deauth) Sending STA is leaving
  (or has left) IBSS or ESS
tus='completion' parameter='failure'
<3>CTRL-EVENT-EAP-FAILURE EAP authentication failed
<3>CTRL-EVENT-DISCONNECTED bssid=7c:10:c9:02:da:48 reason=3 locally_generated=1
<3>CTRL-EVENT-DSCP-POLICY clear_all
<3>CTRL-EVENT-REGDOM-CHANGE init=CORE type=WORLD
<3>SME: Trying to authenticate[   62.447378] wlan: HostMlme mlan0 send auth to bssid
 7c:XX:XX:XX:da:48
 with 7c:10:c9:02:da:48 (SSID='ASUS_AP_2G' freq=2437 MHz)
> [   62.459310] mlan0:
[   62.459318] wlan: HostMlme Auth received from 7c:XX:XX:XX:da:48
```

```
<3>Trying to associate with 7c:10:c9:02:da:48 (SSID='ASUS_AP_2G' freq=2437 MHz)
> [   62.485813] wlan: HostMlme mlan0 Connected to bssid 7c:XX:XX:XX:da:48 successfully
<3>Associated with 7c:10:c9:02:d[   62.496744] mlan0:
a:48
<3>CTRL-EVENT-SUBNET-STA[   62.496760] wlan: Send EAPOL pkt to 7c:XX:XX:XX:da:48
TUS-UPDATE status=0
<3>CTRL-EVENT-REGDOM-CHANGE init=COUNTRY_IE type=COUNTRY alpha2=US
<3>EAPOL-RX 7c:10:c9:02:da:48 99
> [   62.518592] mlan0:
<3>EAPOL-RX 7c:10:c9:02:da:48 15[   62.518598] wlan: Send EAPOL pkt to 7c:XX:XX:XX:da:48
5
<3>WPA: Key negotiation complete[   62.533798] IPv6: ADDRCONF(NETDEV_CHANGE): mlan0: link
 becomes ready
d with 7c:10:c9:02:da:48 [PTK=CCM[   62.543432] woal_cfg80211_set_rekey_data return:
 gtk_rekey_offload is DISABLE
P GTK=CCMP]
<3>CTRL-EVENT-CONNECTED - Connection to 7c:10:c9:02:da:48 completed [id=0 id_str=]
```

***Note:*** *Registration with the external AP shows as* `WPA: Key negotiation complete` *in the command output.*

**Step 8** - Verify the status.

```
> status
```

Command output example:

```
> status
bssid=7c:10:c9:02:da:48
freq=2437
ssid=ASUS_AP_2G
id=0
mode=station
pairwise_cipher=CCMP
group_cipher=CCMP
key_mgmt=WPA2-PSK
wpa_state=COMPLETED
p2p_device_address=2c:4c:c6:f4:67:9e
address=2c:4c:c6:f4:67:9e
uuid=25d271cf-1787-5d3a-9903-595488876c53
```

### 5.1.6 uAP configuration

This section shows how to configure and start the Access Point from the NXP-based wireless module. Linux operating system provides the hostapd user space utility for the access point and authentication servers.

The configurations for hostapd are stored in the *hostapd.conf* file located in */etc/* directory.

#### 5.1.6.1 Get the hostpad version

Execute the command:

```
hostapd -v
```

Command output example:

```
hostapd v2.11
User space daemon for IEEE 802.11 AP management,
IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator
Copyright (c) 2002-2024, Jouni Malinen <j@w1.fi> and contributors
```

#### 5.1.6.2 Configure the 2.4 GHz Access Point

Edit the configuration file:

```
nano /etc/hostapd-2.4g.conf
```

Parameter values in the configuration file:

```
interface=uap0
hw_mode=g
channel=0
country_code=US
ssid=NXP_Demo
ieee80211n=1
```

#### 5.1.6.3 Configure the 5 GHz Access Point

Edit the configuration file:

```
nano /etc/hostapd-5g.conf
```

Parameter values in the configuration file:

```
interface=uap0
hw_mode=a
channel=0
country_code=US
ssid=NXP_Demo
ieee80211n=1
```

#### 5.1.6.4 Set up udhcp server

The udhcp server is used to assign the IP to the client devices that are associated with the access point.

**Step 1** - Create the configuration file for udhcp server

- Create the configuration file *udhcpd.conf* in *etc* repository

```
nano /etc/udhcpd.conf
```

- Add the following content to *udhcpd.conf* file

```
start 192.168.1.10
end 192.168.1.20
interface uap0
```

**Step 2** - Create udhcp client lease database file

udhcp server uses the database file to store all the leases that is has assigned.

- Create the lease database file *udhcpd.leases* in */var/lib/misc/* repository so udhcp server can start
- The newly created file does not require any content.

```
touch /var/lib/misc/udhcpd.leases
```

### 5.1.6.5 Start the Access Point

Run the following command to start the Access Point:

```
killall wpa_supplicant
killall hostapd
ifconfig uap0 192.168.1.2 netmask 255.255.255.0 up
route add default gw 192.168.1.1
```

Start udhcp server to assign the IP address to the client devices that are connected to the access point

```
udhcpd /etc/udhcpd.conf
```

Command to start the 2.4 GHz Access Point:

```
hostapd /etc/hostapd-2.4g.conf
```

Command output example:

```
    Configuration file: /etc/hostapd-2.4g.conf
    rfkill: Cannot open RFKILL control device
uap0: interface state UNINITIALIZED->COUNTRY_UPDATE
    ACS: Automatic channel selection started, this may take a bit
    [ 9618.998539] wlan: SCAN COMPLETED: scanned AP count=1
uap0: interface state COUNTRY_UPDATE->ACS
uap0: ACS-STARTED
    [ 9619.479337] wlan: SCAN COMPLETED: scanned AP count=1
    [ 9619.959539] wlan: SCAN COMPLETED: scanned AP count=0
    [ 9620.439382] wlan: SCAN COMPLETED: scanned AP count=0
    [ 9620.919360] wlan: SCAN COMPLETED: scanned AP count=1
uap0: ACS-COMPLETED freq=2412 channel=1[ 9620.933098] wlan: Starting AP
    Using interface uap0 with hwaddr 00:50:43:24:84:c4 and ssid "NX[ 9620.939164] Get
 ht_cap from beacon ies: 0xc
    P_Demo"
    [ 9620.960100] wlan: AP started
    [ 9620.964230] Set AC=3, txop=47 cwmin=3, cwmax=7 aifs=1
    [ 9620.971787] Set AC=2, txop=94 cwmin=7, cwmax=15 aifs=1
    [ 9620.979077] Set AC=0, txop=0 cwmin=15, cwmax=63 aifs=3
    [ 9620.986466] Set AC=1, txop=0 cwmin=15, cwmax=1023 aifs=7
uap0: interface state ACS->ENABLED
uap0: AP-ENABLED
```

Command to start the 5 GHz Access Point:

```
hostapd /etc/hostapd-5g.conf
```

Command output example:

```
    Configuration file: /etc/hostapd-5g.conf
    rfkill: Cannot open RFKILL control device
uap0: interface state UNINITIALIZED->COUNTRY_UPDATE
    ACS: Automatic channel selection started, this may take a bit
    [ 9816.085716] wlan: SCAN COMPLETED: scanned AP count=0
uap0: interface state COUNTRY_UPDATE->ACS
uap0: ACS-STARTED
    [ 9816.483935] wlan: SCAN COMPLETED: scanned AP count=0
    [ 9816.883357] wlan: SCAN COMPLETED: scanned AP count=0
    [ 9817.279225] wlan: SCAN COMPLETED: scanned AP count=0
    [ 9817.675171] wlan: SCAN COMPLETED: scanned AP count=0
uap0: ACS-COMPLETED freq=5180 channel=36[ 9817.686492] wlan: Starting AP
    Using interface uap0 with hwaddr 00:50:43:24:84:c4 and ssid "NX[ 9817.692595] Get
 ht_cap from beacon ies: 0xc
    P_Demo"
    [ 9817.702468] fw doesn't support 11ax
    [ 9817.717811] wlan: AP started
    [ 9817.722028] Set AC=3, txop=47 cwmin=3, cwmax=7 aifs=1
    [ 9817.729606] Set AC=2, txop=94 cwmin=7, cwmax=15 aifs=1
    [ 9817.736887] Set AC=0, txop=0 cwmin=15, cwmax=63 aifs=3
    [ 9817.744417] Set AC=1, txop=0 cwmin=15, cwmax=1023 aifs=7
uap0: interface state ACS->ENABLED
uap0: AP-ENABLED
```

### 5.1.7 uAP security features

This section includes how to configure the Mobile AP security with the configuration file *hostapd.conf*. The Mobile AP and the client interface can run simultaneously and operate on the same band and channel.

#### 5.1.7.1 Hostapd configuration file

The configuration file *hostapd.conf* is located in *etc* directory. Each security mode requires specific parameters such as `hw_mode`, `AP modes` (i.e 802.11ac, 802.11n, 802.11ax), `wpa_key_mgmt`, `SSID`, `channel`, `psk`, and more. To configure Mobile AP security, change the parameter values in the *hostapd.conf* file.

##### 5.1.7.1.1 Open security

Example of *hostapd.conf* content for Open security:

```
interface=uap0
hw_mode=a
ieee80211n=1
channel=36
country_code=US
ssid=NXP_Demo
```

Parameter settings:

- `hw_mode = a`
- `channel = 36`
- `ieee80211n = 1`

#### 5.1.7.1.2 WPA2 security

Example of *hostapd.conf* content for WPA2 security and for 5 GHz:

```
interface=uap0
driver=nl80211
ctrl_interface=/var/run/hostapd
ctrl_interface_group=0
ieee80211d=1
country_code=US
beacon_int=100
dtim_period=1
wmm_enabled=1
uapsd_advertisement_enabled=1
wmm_ac_bk_cwmin=4
wmm_ac_bk_cwmax=10
wmm_ac_bk_aifs=7
wmm_ac_bk_txop_limit=0
wmm_ac_bk_acm=0
wmm_ac_be_aifs=3
wmm_ac_be_cwmin=4
wmm_ac_be_cwmax=10
wmm_ac_be_txop_limit=0
wmm_ac_be_acm=0
wmm_ac_vi_aifs=2
wmm_ac_vi_cwmin=3
wmm_ac_vi_cwmax=4
wmm_ac_vi_txop_limit=94
wmm_ac_vi_acm=0
wmm_ac_vo_aifs=2
wmm_ac_vo_cwmin=2
wmm_ac_vo_cwmax=3
wmm_ac_vo_txop_limit=47
wmm_ac_vo_acm=0
ssid=NXP_Demo
ignore_broadcast_ssid=0
hw_mode=a
channel=36
auth_algs=1
max_num_sta=10
ieee80211n=1
require_ht=0
ht_capab=[SHORT-GI-20][SHORT-GI-40][HT40+]
ieee80211ac=1
require_vht=0
vht_capab=[SHORT-GI-80]
vht_oper_chwidth=1
vht_oper_centr_freq_seg0_idx=42
ieee80211ax=1
he_su_beamformer=1
he_bss_color=1
he_oper_chwidth=1
he_oper_centr_freq_seg0_idx=42
eapol_version=1
wpa_key_mgmt=WPA-PSK
wpa=2
rsn_pairwise=CCMP
wpa_passphrase=1234567890
```

Parameter settings:

- `hw_mode = a`
- `channel = 36`
- `ieee80211n = 1`
- `ieee80211ac = 1`
- `ieee80211ax = 1`
- `wpa_key_mgmt = WPA-PSK`
- `wpa_passphrase = 1234567890`
- `wpa=2`

#### 5.1.7.1.3 WPA3 security

##### 5.1.7.1.3.1 WPA3 transition security

When WPA3-SAE is configured on the AP, WPA3 SAE is used on the STA to connect with the AP. When WPA3-SAE transition mode is configured on the AP, then either WPA3 SAE or WPA2-PSK can be used on STA to connect with the AP.

Example of *hostapd.conf* content for 5 GHz WPA3 transition security:

```
interface=uap0
driver=nl80211
ssid=NXP_Demo
hw_mode=a
channel=36
wmm_enabled=1
ieee80211n=1
ieee80211ac=1
ieee80211ax=1
auth_algs=1
wpa=2
wpa_pairwise=CCMP
wpa_passphrase=1234567890
sae_password=1234567890
wpa_key_mgmt=WPA-PSK SAE
wpa_group_rekey=1800
rsn_pairwise=CCMP
ieee80211w=1
sae_groups=19 20 21
sae_require_mfp=1
sae_pwe=2
sae_anti_clogging_threshold=10
```

Parameter settings:

- hw_mode = a
- Channel = 36
- wpa_pairwise=CCMP
- wpa_passphrase=1234567890
- wpa_key_mgmt=WPA-PSK SAE
- wpa_group_rekey=1800
- rsn_pairwise=CCMP
- ieee80211w=1
- sae_groups=19 20 21
- wpa=2

#### 5.1.7.2 Example

The example shows how to start a Mobile AP, and check for connected clients. The steps are the same for all security modes.

**Step 1** - Edit *hostapd.conf* configuration file for the security mode. See Open security , WPA2 security , and WPA3 transition security .

**Step 2** - Bring up the Mobile AP interface.

```
# ifconfig uap0 192.168.1.2 netmask 255.255.255.0 up
```

**Step 3** - Set up the udhcp server.

The udhcp server is used to assign the IP to the client devices that are associated with the access point.

- Create the configuration file *udhcpd.conf* in *etc* directory.

```
nano /etc/udhcpd.conf
```

- Add the following content to the *udhcpd.conf* file.

```
start 192.168.1.10
end 192.168.1.20
interface uap0
```

**Step 4** - Create udhcp client lease database file.

The udhcp server uses the database file to store all the leases that is has assigned.

- Create the lease database file *udhcpd.leases* in */var/lib/misc/* repository so the udhcp server can start.

```
touch /var/lib/misc/udhcpd.leases
```

- The newly created file does not require any content.

**Step 5** - Start udhcp server to assign the IP address to the client devices that are connected to the access point.

```
udhcpd /etc/udhcpd.conf
```

**Step 7** - Cancel previous `hostapd` processes.

```
# killall hostapd
```

**Step 8** - Run `hostapd` in the background.

```
# hostapd -B /etc/hostapd.conf
```

Command output example:

```
uap0: interface state UNINITIALIZED->COUNTRY_UPDATE
ACS: Automatic channel selection started, this may take a bit
wlan: SCAN COMPLETED: scanned AP count=31
uap0: interface state COUNTRY_UPDATE->ACS
uap0: ACS-STARTED
wlan: SCAN COMPLETED: scanned AP count=9
wlan: SCAN COMPLETED: scanned AP count=17
wlan: SCAN COMPLETED: scanned AP count=13
wlan: SCAN COMPLETED: scanned AP count=15
wlan: Starting AP
wlan: AP started
Set AC=3, txop=47 cwmin=3, cwmax=7 aifs=1
Set AC=2, txop=94 cwmin=7, cwmax=15 aifs=1
Set AC=0, txop=0 cwmin=15, cwmax=63 aifs=3
Set AC=1, txop=0 cwmin=15, cwmax=1023 aifs=7
uap0:
wlan: HostMlme Auth received from 36:XX:XX:XX:02:59
wlan: HostMlme uap0 send Auth
uap0:
wlan: HostMlme MICRO_AP_STA_ASSOC 36:XX:XX:XX:02:59
wlan: UAP/GO add peer station, address =36:XX:XX:XX:02:59
wlan: HostMlme uap0 send assoc/reassoc resp
```

**Step 9** - Display the list of connected clients.

```
# iw uap0 station dump
```

Command output example:

```
Station 36:86:69:d2:02:59
inactive time: 266 ms
signal: -46 dBm
current time: 165191405647 ms
```

### Expected result

The Mobile AP is configured with Open, WPA2, or WPA3 transition security mode. The interface is up and clients can connect. Connect a client device and check connectivity using ping.

### 5.1.8 Configure IEEE 802.11a/b/g/n/ac/ax standards

This section describes the different IEEE 802.11 standard configurations for the Access Point. The standards use different bandwidths and network speed. The IEEE 802.11a and IEEE 802.11ac standards operate only at 5 GHz, IEEE 802.11b/g operate at 2.4 GHz, and both IEEE 802.11n and IEEE802.11ax can operate at 2.4 GHz and 5 GHz bands.

#### 5.1.8.1 Configure IEEE 802.11b standard

Configure the Access Point for IEEE 802.11b standard.

```
root@imx8mqevk:~# nano /etc/hostapd.conf
```

Configuration file content:

```
interface=uap0
hw_mode=b
channel=0
country_code=US
ssid=NXP_Demo
```

#### 5.1.8.2 Configure IEEE 802.11a standard

Configure the Access Point for IEEE 802.11a standard.

```
root@imx8mqevk:~# nano /etc/hostapd.conf
```

Configuration file content:

```
interface=uap0
hw_mode=a
channel=0
country_code=US
ssid=NXP_Demo
```

#### 5.1.8.3 Configure IEEE 802.11g standard

Configure the Access Point for IEEE 802.11g standard.

```
root@imx8mqevk:~# nano /etc/hostapd.conf
```

Configuration file content:

```
interface=uap0
hw_mode=g
channel=0
country_code=US
ssid=NXP_Demo
```

### 5.1.8.4 Configure IEEE 802.11n standard

Use the following to configure the Access Point configuration for IEEE 802.11n standard for 2.4 GHz and 5 GHz.

**Edit the configuration for 2.4 GHz**

```
root@imx8mqevk:~# nano /etc/hostapd.conf
```

Configuration file content:

```
interface=uap0
hw_mode=g
ieee80211n=1
channel=0
country_code=US
ssid=NXP_Demo
```

**Edit the configuration for 5 GHz**

```
root@imx8mqevk:~# nano /etc/hostapd.conf
```

Configuration file content:

```
interface=uap0
hw_mode=a
ieee80211n=1
channel=0
country_code=US
ssid=NXP_Demo
```

### 5.1.8.5 Configure IEEE 802.11ac standard

Configure the Access Point configuration for IEEE 802.11ac standard.

```
root@imx8mqevk:~# nano /etc/hostapd.conf
```

Configuration file content:

```
interface=uap0
hw_mode=a
ieee80211ac=1
channel=0
country_code=US
ssid=NXP_Demo
```

### 5.1.8.6 Configure IEEE802.11ax standard

Use the following to configure the Access Point configuration for IEEE 802.11ax standard for 2.4 GHz and 5 GHz.

**Edit the configuration for 2.4 GHz**

```
root@imx8mqevk:~# nano /etc/hostapd.conf
```

Configuration file content

```
interface=uap0
hw_mode=g
ieee80211ax=1
ieee80211n=1
ieee80211ac=1
channel=0
country_code=US
ssid=NXP_Demo
```

**Edit the configuration for 5 GHz**

```
root@imx8mqevk:~# nano /etc/hostapd.conf
```

Configuration file content

```
interface=uap0
hw_mode=a
ieee80211ax=1
ieee80211n=1
ieee80211ac=1
channel=0
country_code=US
ssid=NXP_Demo
```

### 5.1.9 Configure and test 802.11w (PMF)

IEEE 802.11w is the standard defined by Wi-Fi Alliance for Protected Management Frames (PMF). PMF is used to increase security by providing:

- Data confidentiality of management frames
- Mechanisms that enable data integrity.
- Data origin authenticity
- Replay protection

#### 5.1.9.1 Enable 802.11w (PMF)

**Step 1** - Create *wpa_supplicant.conf*.

Create and edit *wpa_supplicant.conf* configuration file in *etc* directory.

```
root@imx8mqevk:~# nano /etc/wpa_supplicant.conf
```

Content of *wpa_supplicant.conf* file:

```
network={
        ssid="ASUS_2G"
        scan_ssid=1
        key_mgmt=SAE
        proto=RSN
        pairwise=CCMP
        group=CCMP
        psk="12345678"
        ieee80211w=2
        }
```

**Step 2** - Start wpa_supplicant.

```
root@imx8mqevk:~# wpa_supplicant -i mlan0 -Dnl80211 -c /etc/wpa_ supplicant.conf -B
```

Command output example:

```
Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL control device
```

**Step 3** - Check the connection of DUT STA with AP in *dmesg* logs.

```
root@imx8mqevk:~# dmesg -w
```

Command output example:

```
[  229.862260] wlan: HostMlme mlan0 Connected to bssid 7c:XX:XX:XX:da:48 successfully
[  230.959806] WPA: Key negotiation completed with 7c:10:c9:02:da:48 [PTK=CCMP GTK=CCMP]
mlan0: CTRL-EVENT-CONNECTED - Connection to 7c:10:c9:02:da:48 completed
```

**Step 4** - Use Sniffer to verify the settings.

Verify that RSN information shows the correct settings for PMF (Figure 34).

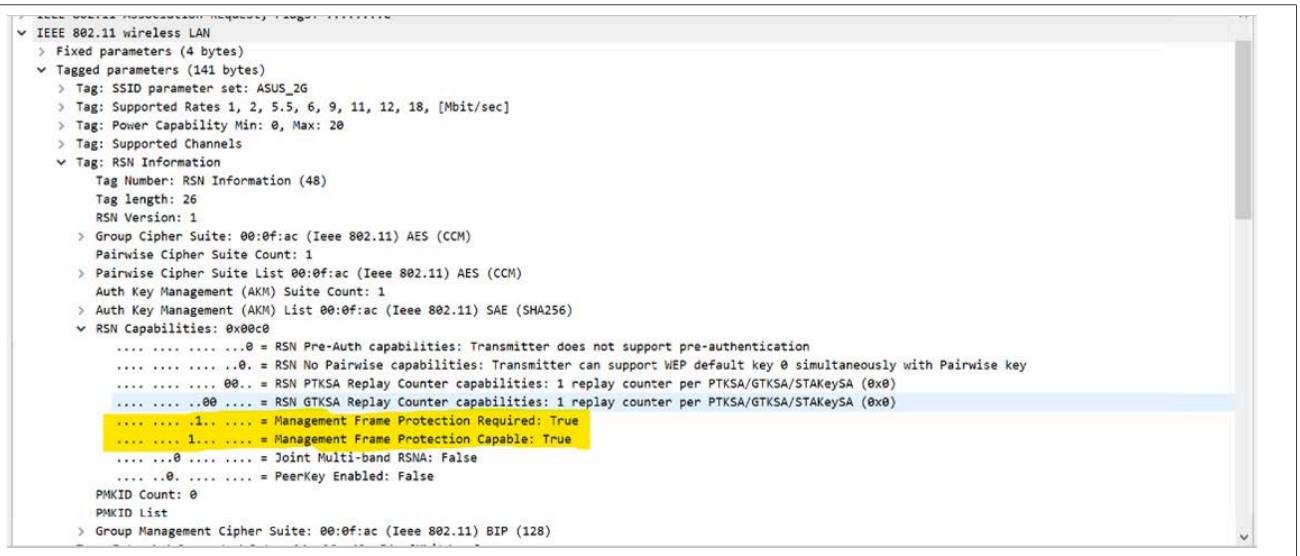



**Figure 34. RSN information setting showing PMF enabled**

### 5.1.9.2 Set 802.11w (PMF) capable

**Step 1** - Create *wpa_supplicant.conf*.

Create and edit *wpa_supplicant.conf* configuration file in *etc* directory.

```
root@imx8mqevk:~# nano /etc/wpa_supplicant.conf
```

Content of wpa_supplicant.conf file:

```
network={
        ssid="ASUS_2G"
        scan_ssid=1
        key_mgmt=SAE
        proto=RSN
        pairwise=CCMP
        group=CCMP
        psk="12345678"
        ieee80211w=1
        }
```

**Step-2: Start wpa_supplicant**

```
root@imx8mqevk:~# wpa_supplicant -i mlan0 -Dnl80211 -c /etc/wpa_ supplicant.conf -B
```

Command output example:

```
Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL control device
```

**Step 3** - Verify the connection of DUT STA with the AP.

```
root@imx8mqevk:~# dmesg -w
```

Command output example:

```
[   80.463167] wlan: HostMlme mlan0 Connected to bssid 7c:XX:XX:XX:da:48 successfully
[   80.925710] WPA: Key negotiation completed with 7c:10:c9:02:da:48 [PTK=CCMP GTK=CCMP]
CTRL-EVENT-CONNECTED - Connection to 7c:10:c9:02:da:48 completed
```

**Step 5** - Use Sniffer to verify the settings.

Verify that RSN information shows the correct settings for PMF (Figure 35).

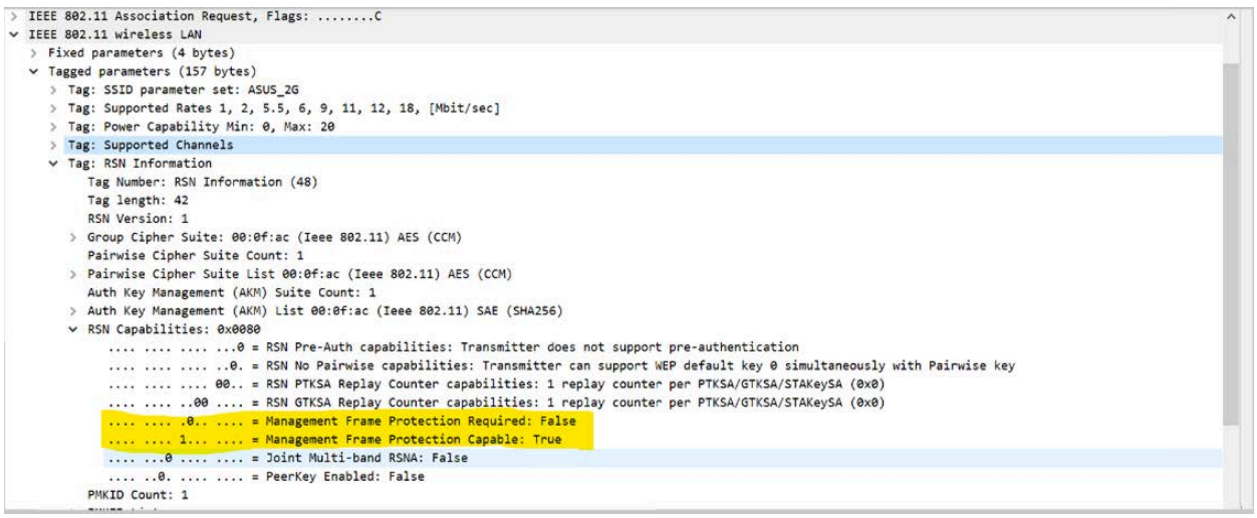



**Figure 35. RSN information settings showing PMF set to capable**

### 5.1.10 Wi-Fi direct

This section describes the Wi-Fi Direct (WFD) mode configuration and procedures. The feature is used to establish a connection for device-to-device or peer-to-peer (P2P) communication, without a nearby centralized network.

**Wi-Fi direct GO mode**

Use the following steps to test the Wi-Fi direct feature in GO mode on WFD interface wfd0.

**Step 1** – Edit *p2p_supplicant.conf*.

Content of *p2p_supplicant.conf* file:

```
ctrl_interface=/var/run/wpa_supplicant
update_config=1
persistent_reconnect=1
p2p_no_group_iface=1
```

**Step 2** – Load the driver/fw.

```
modprobe moal mod_para=nxp/wifi_mod_para.conf
```

**Step 3** – Start *wpa_supplicant*.

```
wpa_supplicant -i wfd0 -Dnl80211 -c <path_to_file>/p2p.conf -B
```

Command output example:

```
Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL control device
```

**Step 4** – Start the *wpa_cli*.

```
wpa_cli
```

Command output example:

```
wpa_cli v2.11
Copyright (c) 2004-2024, Jouni Malinen <j@w1.fi> and contributors
This software may be distributed under the terms of the BSD license.
See README for more details.
Selected interface 'wfd0'
Interactive mode
```

**Step 5** – Start Wi-Fi Direct on the user interface for the peer or mobile device.

**Step 6** – Enable the p2p device discovery.

```
> p2p_find
```

Command output example:

```
OK
[  202.954971] wlan: wfd0 START SCAN
<3>CTRL-EVENT-SCAN-STARTED
> [  205.760257] wlan: SCAN COMPLETED: scanned AP count=2
<3>CTRL-EVENT-REGDOM-CHANGE init=BEACON_HINT type=UNKNOWN
<3>P2P-DEVICE-FOUND 26:18:1d:56:65:0a p2p_dev_addr=26:18:1d:56:65:0a
 pri_dev_type=10-0050F204-5 name='[Phone] Galaxy S8' config1
> [  205.896705] wlan: wfd0 START SCAN
<3>CTRL-EVENT-SCAN-STARTED
```

**Step 7** – Stop the ongoing p2p discovery.

```
> p2p_stop_find
```

**Step 8** - Request provisioning discovery of peer mobile device.

```
> p2p_prov_disc <Peer Device MAC address> pbc
```

Example of command:

```
e> p2p_prov_disc 26:18:1d:56:65:0a pbc
```

Command output example:

```
[  244.076007] wlan: TX P2P Provision Discovery Request, channel=1
OK
> [  244.399980] wlan: TX P2P Provision Discovery Request, channel=1
[  244.720509] wlan: TX P2P Provision Discovery Request, channel=1
[  244.875344] wlan: RX P2P Provision Discovery Response, channel=1
<3>P2P-PROV-DISC-PBC-RESP 26:18:1d:56:65:0a
```

**Step 9** – Connect with the p2p device.

```
> p2p_connect <Peer Device MAC address> pbc go_intent=14 freq=2437
```

Example of command:

```
> p2p_connect 26:18:1d:56:65:0a pbc go_intent=14 freq=2437
```

**Step 10** – Accept the Wi-Fi Direct connection request on the peer or on the mobile device.

Command output example:

```
[  270.284200] wlan: TX P2P Group Owner Negotiation Req Frame, channel=1
OK
> [  270.436918] wlan: RX P2P Group Owner Negotiation Rsp Frame, channel=1
[  273.201217] wlan: RX P2P Group Owner Negotiation Req Frame, channel=11
[  273.208544] wlan: TX P2P Group Owner Negotiation Rsp Frame, channel=11
[  273.226731] wlan: RX P2P Group Owner Negotiation Confirm Frame, channel=11
[  273.907552] wlan: SCAN COMPLETED: scanned AP count=1
<3>CTRL-EVENT-SCAN-RESULTS
<3>WPS-AP-AVAILABLE-PBC
> [  273.918293] Can not set data rate in disconnected state
<3>SME: Trying to authenticate with 26:18:1d:56:e5:0a (SSID='DIRECT-dW-SM-G950F'
 freq=2437 MHz)
> [  274.141469] wlan: HostMlme wfd0 send auth to bssid 26:XX:XX:XX:e5:0a
[  274.192359] wfd0:
[  274.192366] wlan: HostMlme Auth received from 26:XX:XX:XX:e5:0a
<3>Trying to associate with 26:18:1d:56:e5:0a (SSID='DIRECT-dW-SM-G950F' freq=2437 MHz)
> [  274.298586] wlan: HostMlme wfd0 Connected to bssid 26:XX:XX:XX:e5:0a successfully
<3>Associated with 26:18:1d:56:e[  274.307263] wfd0:
<3>CTRL-EVE[  274.327528] wlan: Send EAPOL pkt to 26:XX:XX:XX:e5:0a
NT-EAP-STATUS status='started' parameter=''
<3>CTRL-EVENT-REGDOM-CHANGE init=COUNTRY_IE type=COUNTRY alpha2=IN
<3>EAPOL-RX 26:18:1d:56:e5:0a 18
> [  274.397589] wlan: Send EAPOL pkt to 26:XX:XX:XX:e5:0a
<3>EAPOL-RX 26:18:1d:56:e5:0a 234
<3>SME: Trying to authenticate with 26:18:1d:56:e5:0a (SSID='DIRECT-dW-SM-G950F'
 freq=2437 MHz)
> [  274.723741] wlan: HostMlme wfd0 send auth to bssid 26:XX:XX:XX:e5:0a
[  274.806751] wfd0:
[  274.806758] wlan: HostMlme Auth received from 26:XX:XX:XX:e5:0a
<3>Trying to associate with 26:18:1d:56:e5:0a (SSID='DIRECT-dW-SM-G950F' freq=2437 MHz)
> [  274.912719] wlan: HostMlme wfd0 Connected to bssid 26:XX:XX:XX:e5:0a successfully
<3>Associated with 26:18:1d:56:e5:0a
<3>CTRL-EVENT-SUBNET-S[  274.924725] wfd0:
TATUS-UPDATE status=0
<3>CTRL[  274.924737] wlan: Send EAPOL pkt to 26:XX:XX:XX:e5:0a
-EVENT-REGDOM-CHANGE init=COUNTRY_IE type=COUNTRY alpha2=IN
<3>EAPOL-RX 26:18:1d:56:e5:0a 9[  274.943443] wfd0:
9
<3>EAPOL-RX 26:18:1d:56:e5:[  274.943447] wlan: Send EAPOL pkt to 26:XX:XX:XX:e5:0a
0a 171
<3>WPA: Key negotiation comple[  274.955269] IPv6: ADDRCONF(NETDEV_CHANGE): wfd0: link
 becomes ready
ted with 26:18:1d:56:e5:0a [PTK=C[  274.964998] woal_cfg80211_set_rekey_data return:
 gtk_rekey_offload is DISABLE
CMP GTK=CCMP]
<3>CTRL-EVENT-CONNECTED - Connection to 26:18:1d:56:e5:0a completed [id=0 id_str=]
<3>P2P-GROUP-STARTED wfd0 client ssid="DIRECT-dW-SM-G950F" freq=2437
 psk=aef6fc0c420903c10df110b35efc2d692fef2637d628a63132b2481
```

**Step 11** – Check the connection status.

```
> status
```

Command output example:

```
bssid=26:18:1d:56:e5:0a
freq=2437
ssid=DIRECT-dW-SM-G950F
id=0
mode=station
pairwise_cipher=CCMP
group_cipher=CCMP
key_mgmt=WPA2-PSK
wpa_state=COMPLETED
p2p_device_address=2e:4c:c6:f4:67:9e
address=2e:4c:c6:f4:67:9e
uuid=17ca6821-1587-5ce1-bab7-4aa5716e65fe
```

**Step 12** – Close the *wpa_cli* prompt.

```
> quit
```

### Wi-Fi direct client mode

Use the following steps to test the Wi-Fi direct feature in client mode.

**Step 1** – Create *p2p_supplicant.conf*.

Content of *p2p_supplicant.conf* file content:

```
ctrl_interface=/var/run/wpa_supplicant
update_config=1
device_name=NXP-P2P
device_type=10-0050F204-4
config_methods="keypad push_button virtual display"
persistent_reconnect=1
p2p_no_group_iface=1
```

***Note:*** *The content of the configuration file is an example. Replace the values of parameters like device_name or device_type with your own values.*

**Step 2** - Start the wpa_supplicant.

```
wpa_supplicant -i wfd0 -Dnl80211 -c <path_to_file>/p2p.conf -B
```

Command output example:

```
Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL control device
WPS: Converting display to virtual_display for WPS 2.0 compliance
WPS: Converting push_button to virtual_push_button for WPS 2.0 compliance
```

**Step 3** – Start the wpa_cli.

```
wpa_cli
```

Command output example:

```
wpa_cli v2.11
Copyright (c) 2004-2024, Jouni Malinen <j@w1.fi> and contributors
This software may be distributed under the terms of the BSD license.
See README for more details.
Selected interface 'wfd0'
Interactive mode
```

**Step 4** – Start Wi-Fi Direct on the User Interface for the peer or mobile device.

**Step 5** – Enable the p2p device discovery.

```
> p2p_find
```

Command output example:

```
OK
[  198.289636] wlan: wfd0 START SCAN
<3>CTRL-EVENT-SCAN-STARTED
> [  201.094955] wlan: SCAN COMPLETED: scanned AP count=1
<3>CTRL-EVENT-REGDOM-CHANGE init=BEACON_HINT type=UNKNOWN
> [  201.436453] wlan: wfd0 START SCAN
<3>CTRL-EVENT-SCAN-STARTED
> [  202.889619] wlan: SCAN COMPLETED: scanned AP count=2
<3>P2P-DEVICE-FOUND 26:18:1d:56:65:0a p2p_dev_addr=26:18:1d:56:65:0a
 pri_dev_type=10-0050F204-5 name='[Phone] Galaxy S8' config1
> [  203.124702] wlan: wfd0 START SCAN
<3>CTRL-EVENT-SCAN-STARTED
> [  203.252207] wlan: SCAN COMPLETED: scanned AP count=2
> [  203.487885] wlan: wfd0 START SCAN
<3>CTRL-EVENT-SCAN-STARTED
```

**Step 6** – Stop the ongoing p2p discovery.

```
> p2p_stop_find
```

**Step 7** – Add a new P2P group with a frequency value.

```
> p2p_group_add freq=2437
```

Command output example:

```
OK
> [  235.152677] Can not set data rate in disconnected state
[  235.162729] wlan: Starting AP
[  235.173056] wlan: AP started
[  235.176875] IPv6: ADDRCONF(NETDEV_CHANGE): wfd0: link becomes ready
[  235.184207] Set AC=3, txop=47 cwmin=3, cwmax=7 aifs=1
[  235.189592] Set AC=2, txop=94 cwmin=7, cwmax=15 aifs=1
[  235.194987] Set AC=0, txop=0 cwmin=15, cwmax=63 aifs=3
[  235.200335] Set AC=1, txop=0 cwmin=15, cwmax=1023 aifs=7
<3>AP-ENABLED
<3>CTRL-EVENT-CONNECTED - Connection to 2e:4c:c6:f4:67:9e completed [id=0 id_str=]
<3>P2P-GROUP-STARTED wfd0 GO ssid="DIRECT-lL" freq=2437 passphrase="SNNbuG6t"
 go_dev_addr=2e:4c:c6:f4:67:9e
<3>RX-PROBE-REQUEST sa=26:18:1d:56:65:0a signal=0
<3>RX-PROBE-REQUEST sa=26:18:1d:56:65:0a signal=0
<3>RX-PROBE-REQUEST sa=00:0c:e7:30:90:cb signal=0
<3>RX-PROBE-REQUEST sa=26:18:1d:56:65:0a signal=0
<3>RX-PROBE-REQUEST sa=26:18:1d:56:65:0a signal=0
<3>RX-PROBE-REQUEST sa=26:18:1d:56:65:0a signal=0
<3>RX-PROBE-REQUEST sa=78:af:08:1c:dd:a1 signal=0
<3>RX-PROBE-REQUEST sa=78:af:08:1c:dd:a1 signal=0
<3>RX-PROBE-REQUEST sa=26:18:1d:56:e5:0a signal=0
<3>WPS-ENROLLEE-SEEN 26:18:1d:56:e5:0a 78fa9353-fe52-58aa-9549-c4aeeb7c7f99 0-00000000-0
 0x3148 4 1 [ ]
<3>RX-PROBE-REQUEST sa=26:18:1d:56:e5:0a signal=0
```

**Step 8** – Connect with the P2P device from mobile.

**Step 9** – Establish connection with mobile device.

```
> wps_pbc
```

Command output example:

```
OK
<3>WPS-PBC-ACTIVE
<3>WPS-ENROLLEE-SEEN 26:18:1d:56:e5:0a 78fa9353-fe52-58aa-9549-c4aeeb7c7f99
<3>RX-PROBE-REQUEST sa=00:0c:e7:03:1a:c2 signal=0
<3>WPS-ENROLLEE-SEEN 26:18:1d:56:e5:0a 78fa9353-fe52-58aa-9549-c4aeeb7c7f99 0-00000000-0
 0x3148 4 1 [ ]
<3>RX-PROBE-REQUEST sa=26:18:1d:56:e5:0a signal=0
> [  244.465555] wfd0:
[  244.465570] wlan: HostMlme Auth received from 26:XX:XX:XX:e5:0a
[  244.474209] wlan: HostMlme wfd0 send Auth
[  244.479567] wfd0:
[  244.479572] wlan: HostMlme MICRO_AP_STA_ASSOC 26:XX:XX:XX:e5:0a
[  244.489483] wlan: UAP/GO add peer station, address =26:XX:XX:XX:e5:0a
[  244.496473] wlan: HostMlme wfd0 send assoc/reassoc resp
<3>CTRL-EVENT-EAP-STARTED 26:18:[  244.503445] wfd0:
1d:56:e5:0a
[  244.503465] wlan: Send EAPOL pkt to 26:XX:XX:XX:e5:0a
<3>CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=1
<3>EAPOL-RX 26:18:1d:56:e5:0a 38[  244.536583] wfd0:
[  244.717276] wfd0:
[  244.717280] wlan: EVENT: MICRO_AP_STA_DEAUTH 26:XX:XX:XX:e5:0a
<3>WPS-ENROLLEE-SEEN 26:18:1d:56:e5:0a 78fa9353-fe52-58aa-9549-c4aeeb7c7f99 0-00000000-0
 0x3148 4 1 [ ]
<3>RX-PROBE-REQUEST sa=26:18:1d:56:e5:0a signal=0
> [  244.965520] wlan: hostmlme notify deauth station 26:XX:XX:XX:e5:0a
[  244.971762] wfd0:
[  244.971767] wlan: EVENT: MICRO_AP_STA_DEAUTH 26:XX:XX:XX:e5:0a
<3>RX-PROBE-REQUEST sa=7e:89:3b:2f:cd:0e signal=0
<3>RX-PROBE-REQUEST sa=7e:89:3b:2f:cd:0e signal=0
<3>RX-PROBE-REQUEST sa=c6:ba:d3:16:96:82 signal=0
<3>RX-PROBE-REQUEST sa=7e:89:3b:2f:cd:0e signal=0
<3>WPS-ENROLLEE-SEEN 26:18:1d:56:e5:0a 78fa9353-fe52-58aa-9549-c4aeeb7c7f99 0-00000000-0
 0x3148 4 1 [ ]
<3>RX-PROBE-REQUEST sa=[  245.205445] wfd0:
[  245.205456] wlan: HostMlme Auth received from 26:XX:XX:XX:e5:0a
26:18:1d:56:e5:0a signal=0
> [  245.215770] wlan: HostMlme wfd0 send Auth
[  245.245449] wfd0:
[  245.245455] wlan: HostMlme MICRO_AP_STA_ASSOC 26:XX:XX:XX:e5:0a
[  245.260325] wlan: UAP/GO add peer station, address =26:XX:XX:XX:e5:0a
[  245.268775] wlan: HostMlme wfd0 send assoc/reassoc resp
[  245.275425] wfd0:
[  245.275430] wlan: Send EAPOL pkt to 26:XX:XX:XX:e5:0a
<3>EAPOL-RX 26:18:1d:56:e5:0a 12[  245.321574] wfd0:
1
> [  245.321578] wlan: Send EAPOL pkt to 26:XX:XX:XX:e5:0a
<3>EAPOL-RX 26:18:1d:56:e5:0a 99
<3>AP-STA-CONNECTED 26:18:1d:56:e5:0a p2p_dev_addr=26:18:1d:56:65:0a
<3>EAPOL-4WAY-HS-COMPLETED 26:18:1d:56:e5:0a
```

**Step 11** – Check the connection status.

```
> status
```

Command output example:

```
bssid=2e:4c:c6:f4:67:9e
freq=2437
ssid=DIRECT-lL
id=0
mode=P2P GO
pairwise_cipher=CCMP
group_cipher=CCMP
key_mgmt=WPA2-PSK
wpa_state=COMPLETED
p2p_device_address=2e:4c:c6:f4:67:9e
address=2e:4c:c6:f4:67:9e
uuid=17ca6821-1587-5ce1-bab7-4aa5716e65fe
```

**Step 12** – Close the wpa_cli prompt.

```
> quit
```

### 5.1.11 Suspend and resume

This section introduces the steps to put the i.MX host processor in suspend state.

**Step 1** – Update the configuration file.

Edit *wifi_mod_para.conf* file:

```
nano /lib/firmware/nxp/wifi_mod_para.conf
```

Update `ps_mode` in the configuration file:

```
SD8987 = {
    cfg80211_wext=0xf
    wfd_name=p2p
    max_vir_bss=2
    cal_data_cfg=none
    drv_mode=7
    ps_mode=1
    auto_ds=1
}
```

**Step 2** – Load the modules in the kernel.

```
modprobe moal mod_para=nxp/wifi_mod_para.conf
```

**Step 3** – Connect the client (STA) device to the external access point using wpa_supplicant.

Refer to Association .

**Step 4** – Enable WoWLAN

```
iw phy#0 wowlan enable any
```

**Step 5** – Enable SDIO wakeup

```
echo enabled > /sys/bus/platform/devices/30b50000.mmc/power/wakeup
```

***Note:*** *This step is not needed for PCIe interface.*

**Step 6** – Set host sleep parameters

```
 echo "hssetpara=2 0xff 0xc8 3 400" > /proc/mwlan/adapter0/config
```

**Step 7** – Set the device in suspend state

```
echo mem >> /sys/power/state
```

**Step 8** – Ping from AP backend to STA for host wakeup.

Refer to Section 5.1.12 for more offload features and remote wakeup.

#### 5.1.11.1 Enable dmesg logs for device in supended state

This section explains how to capture the Wi-Fi driver logs even when the i.MX host processor is in suspend state.

**Step 1** – Boot the i.MX 8M Quad EVK to the U-boot console.

```
U-Boot >
```

**Step 2** – Enable dmesg logs while the device is in suspended state.

```
U-Boot > setenv mmcargs $mmcargs no_console_suspend
```

**Step 3** – Boot the i.MX 8M Quad EVK.

```
U-Boot > boot
```

### 5.1.12 Offload during host suspend

#### 5.1.12.1 Neighbor solicitation (NS) offload

NS offload is the ability of network adapter to respond to a Neighbor Discovery Neighbor Solicitation request with a Neighbor Advertisement without waking the host.

This section describes the steps to test the NS offload functionality.

**Step 1** – Load the driver.

```
modprobe moal mod_para=nxp/wifi_mod_para.conf hs_auto_arp=1
```

**Step 2** – Add the SSID and password of external AP to wpa_supplicant.conf.

Content of wpa_supplicant.conf file:

```
network={
         SSID=”<SSID>”
         psk=”<Password>”
         key_mgmt=WPA-PSK
}
```

Example wpa_supplicant.conf file:

```
network={
         SSID="NXP_AP"
         psk="12345678"
         key_mgmt=WPA-PSK
}
```

**Step 3** – Connect with an external AP

```
wpa_supplicant -i mlan0 -D nl80211 -c <path_to_file>/wpa_supplicant.conf -B
```

**Step 4** – Get the IP address for the DUT.

```
udhcpc -i mlan0
```

**Step 5** – Verify IPv4 and IPv6 addresses on mlan0.

```
ifconfig mlan0
```

Command output example:

```
mlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.244  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::2e4c:c6ff:fef4:679e  prefixlen 64  scopeid 0x20<link>
        ether 2c:4c:c6:f4:67:9e  txqueuelen 1000  (Ethernet)
        RX packets 20  bytes 4662 (4.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 52  bytes 7068 (6.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

**Step 6** – Verify IPv4 and IPv6 addresses on the external AP.

Check that both IPv4 and IPv6 addresses are available on eth0 interface on the Linux machine connected to the external AP backend device.

```
nxp@nxp:~$ ifconfig eth0
```

Command output example:

```
nxp@nxp:~$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 28:d2:44:08:90:b2
          inet addr:192.168.0.192  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::ddb8:30e0:6936:e5b6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:18753 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5996 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6383519 (6.3 MB)  TX bytes:954617 (954.6 KB)
          Interrupt:20 Memory:f2500000-f2520000
```

**Step 7** – Check the connectivity between the DUT STA and external AP.

```
ping <ip_address_of_external_AP>
```

Command output example:

```
ping -I mlan0 192.168.0.192 -c 5
PING 192.168.0.192 (192.168.0.192) from 192.168.0.244 mlan0: 56(84) bytes of data.
64 bytes from 192.168.0.192: icmp_seq=1 ttl=64 time=5.68 ms
64 bytes from 192.168.0.192: icmp_seq=2 ttl=64 time=3.80 ms
64 bytes from 192.168.0.192: icmp_seq=3 ttl=64 time=3.46 ms
64 bytes from 192.168.0.192: icmp_seq=4 ttl=64 time=3.49 ms
64 bytes from 192.168.0.192: icmp_seq=5 ttl=64 time=3.05 ms
--- 192.168.0.192 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 3.047/3.894/5.684/0.925 ms
```

**Step 8** – Check IPv6 connectivity from AP back-end to DUT_STA.

```
nxp@nxp:~$ ping6 <IPV6 addr of mlan0 of DUT>%eth0
```

Command output example:

```
nxp@nxp:~$ ping6 fe80::2e4c:c6ff:fef4:679e%eth0 -c 5
PING fe80::2e4c:c6ff:fef4:679e%eth0(fe80::2e4c:c6ff:fef4:679e) 56 data bytes
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=1 ttl=64 time=105 ms
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=2 ttl=64 time=19.6 ms
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=3 ttl=64 time=44.3 ms
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=4 ttl=64 time=64.4 ms
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=5 ttl=64 time=85.9 ms
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=6 ttl=64 time=109 ms
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=7 ttl=64 time=30.6 ms
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=8 ttl=64 time=51.7 ms
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=9 ttl=64 time=74.4 ms
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=10 ttl=64 time=96.9 ms
--- fe80::2e4c:c6ff:fef4:679e%eth0 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9013ms
rtt min/avg/max/mdev = 19.668/68.297/109.205/29.787 ms
```

**Step 9** – Configure host sleep parameters.

- Enable WoWLAN.

```
iw phy#0 wowlan enable any
```

- Set wake-up condition.

Command to set multicast packet wake-up condition:

```
echo "hssetpara=8 0xff 0xc8" > /proc/mwlan/adapter0/config
```

- Enable host sleep.

```
echo mem >  /sys/power/state
```

**Step 10** – Ping from the AP back-end to DUT_STA.

```
nxp@nxp:~ $ ping6 fe80::2e4c:c6ff:fef4:679e%eth0
```

Command output example:

```
nxp@nxp:~ $ ping6 fe80::2e4c:c6ff:fef4:679e%eth0
PING fe80::2e4c:c6ff:fef4:679e%eth0(fe80::2e4c:c6ff:fef4:679e) 56 data bytes
From fe80::ddb8:30e0:6936:e5b6 icmp_seq=26 Destination unreachable: Address unreachable
From fe80::ddb8:30e0:6936:e5b6 icmp_seq=27 Destination unreachable: Address unreachable
From fe80::ddb8:30e0:6936:e5b6 icmp_seq=28 Destination unreachable: Address unreachable
From fe80::ddb8:30e0:6936:e5b6 icmp_seq=29 Destination unreachable: Address unreachable
From fe80::ddb8:30e0:6936:e5b6 icmp_seq=30 Destination unreachable: Address unreachable
From fe80::ddb8:30e0:6936:e5b6 icmp_seq=31 Destination unreachable: Address unreachable
From fe80::ddb8:30e0:6936:e5b6 icmp_seq=32 Destination unreachable: Address unreachable
```

Verify that the host does not wake up when ping6 is started and NS and NA packets are observed on sniffer.

Figure 36 and Figure 37 illustrate the verification from sniffer:

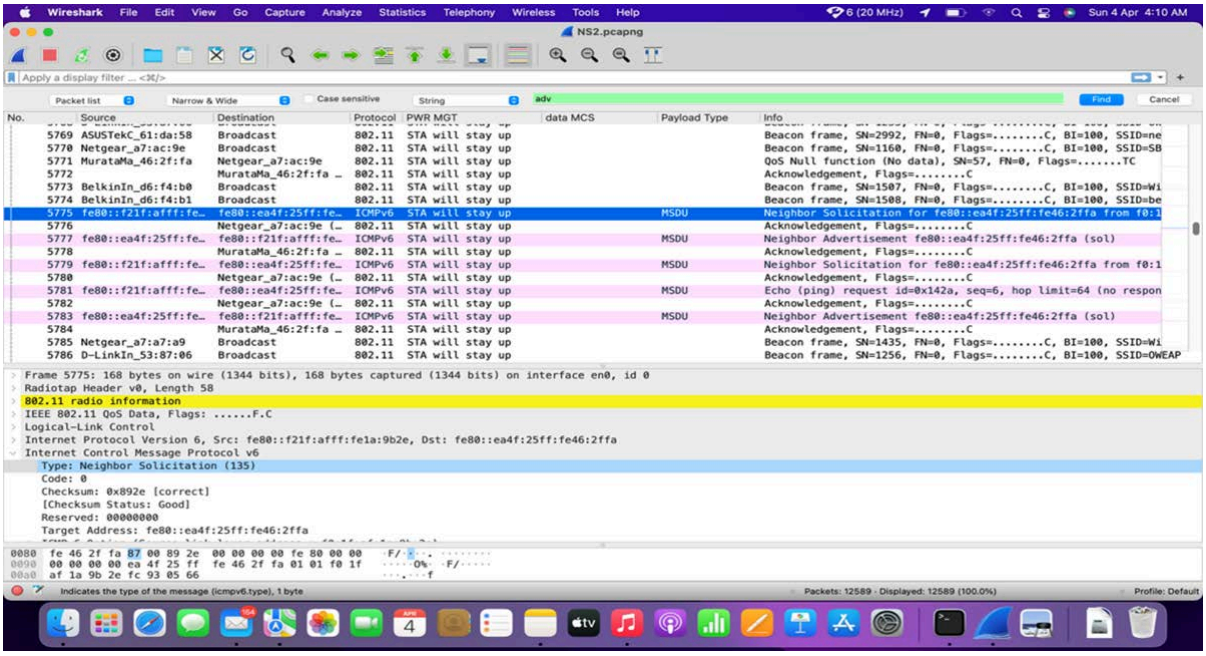



**Figure 36. Neighbor solicitation request**

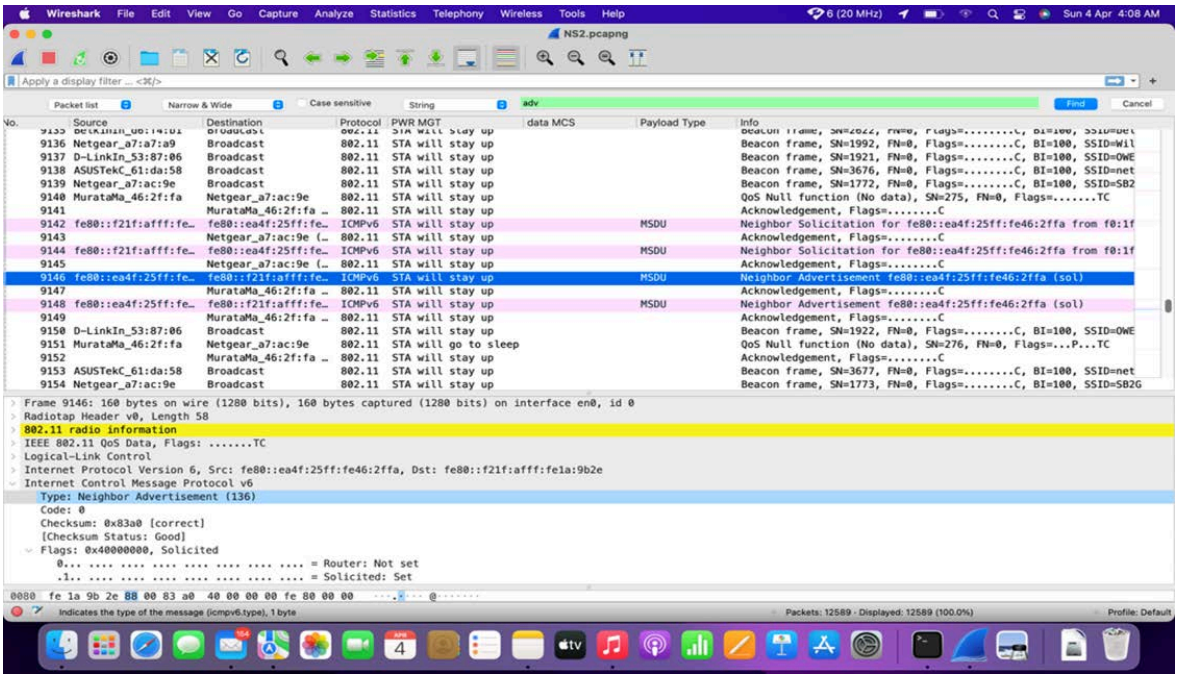



**Figure 37. Neighbor advertisement**

### 5.1.12.2 Multicast DNS (mDNS) wake-up

The Multicast DNS (mDNS) protocol resolves the hostnames to IP addresses within small networks that do not include a local name server. mDNS uses the same programming interfaces, packet formats and operating semantics as unicast domain name system (DNS).

To verify the mDNS wake-up functionality:

**Step 1** - Create *wpa_supplicant.conf* file.

Create and edit *wpa_supplicant.conf* configuration file in *etc* directory.

```
nano /etc/wpa_supplicant.conf
```

Content of wpa_supplicant.conf file:

```
network={
      SSID="ASUS_2G"
      key_mgmt=WPA-PSK
      psk="12345678"
      }
```

**Step 2** - Connect the DUT STA with an external AP.

```
wpa_supplicant -Dnl80211 -i mlan0 -c /etc/wpa_ supplicant.conf -B
```

**Step 4** – Get the IP address for the DUT.

```
udhcpc -i mlan0
```

**Step 5** - Verify IPv4 and IPv6 addresses.

```
ifconfig mlan0
```

Command output example:

```
mlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.244  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::2e4c:c6ff:fef4:679e  prefixlen 64  scopeid 0x20<link>
        ether 2c:4c:c6:f4:67:9e  txqueuelen 1000  (Ethernet)
        RX packets 9  bytes 1448 (1.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 48  bytes 6825 (6.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

**Step 6** - Verify IPv4 and IPv6 addresses on the external AP.

```
ifconfig eth0
```

Command output example:

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.233 netmask 255.255.255.0 broadcast 192.168.0.255
        inet6 fe80::d80:6eb4:869f:f90b  prefixlen 64  scopeid 0x20<link>
        ether 28:d2:44:05:0a:81  txqueuelen 1000  (Ethernet)
        RX packets 171  bytes 165165 (165.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 125  bytes 14394 (14.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 20  memory 0xf2500000-f2520000
```

**Step 7** – Check the connectivity between the DUT STA and external AP.

```
ping <ip_address_of_external_AP>
```

Command output example:

```
ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=5.91 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=64 time=6.49 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=64 time=4.23 ms
```

**Step 8** - Check IPv6 connectivity from the AP to DUT STA.

```
# ping6 <IPV6 addr of mlan0 of DUT>%eth0
```

Command output example:

```
# ping6 -c 20 fe80::2e4c:c6ff:fef4:679e%eth0
PING fe80::2e4c:c6ff:fef4:679e%eth0(fe80::2e4c:c6ff:fef4:679e) 56 data bytes
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=1 ttl=64 time=76.0 ms
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=2 ttl=64 time=97.1 ms
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=3 ttl=64 time=33.3 ms
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=4 ttl=64 time=36.7 ms
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=5 ttl=64 time=57.6 ms
64 bytes from fe80::2e4c:c6ff:fef4:679e: icmp_seq=6 ttl=64 time=78.3 ms
```

**Step 9** - Configure `host_sleep` parameters.

- Enable WoWlan .

```
iw phy#0 wowlan enable any
```

- Set the condition for wake up and multicast packet.

```
echo "hssetpara=8 0xff 0xc8" > /proc/mwlan/adapter0/config
```

- Add multicast route

```
route add -net 224.0.0.0 netmask 240.0.0.0 dev mlan0
```

- Add multicast route at the AP back-end.

```
route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0
```

**Step 10** - Start iPerf UDP server on the DUT.

```
iperf -s -i 1 -u -B 224.0.0.1 &
```

Command output example:

```
------------------------------------------------------------
Server listening on UDP port 5001
Binding to local address 224.0.0.1
Joining multicast group  224.0.0.1
Receiving 1470 byte datagrams
UDP buffer size: 12.0 MByte (default)
```

**Step 11** - Enable host sleep on DUT

```
echo mem > /sys/power/state
```

**Step 12** - Run UDP TX traffic from the external AP to the DUT.

```
iperf -c 224.0.0.1 -i 1 -u -b 5M -B
```

Command output example:

```
------------------------------------------------------------
Client connecting to 224.0.0.1, UDP port 5001
Sending 1470 byte datagrams
Setting multicast TTL to 1
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.0.233 port 60532 connected with 224.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec   612 KBytes  5.01 Mbits/sec
[  3]  1.0- 2.0 sec   610 KBytes  5.00 Mbits/sec
[  3]  2.0- 3.0 sec   610 KBytes  5.00 Mbits/sec
[  3]  3.0- 4.0 sec   610 KBytes  5.00 Mbits/sec
[  3]  4.0- 5.0 sec   610 KBytes  5.00 Mbits/sec
[  3]  5.0- 6.0 sec   610 KBytes  5.00 Mbits/sec
[  3]  6.0- 7.0 sec   612 KBytes  5.01 Mbits/sec
[  3]  7.0- 8.0 sec   610 KBytes  5.00 Mbits/sec
[  3]  8.0- 9.0 sec   610 KBytes  5.00 Mbits/sec
[  3]  9.0-10.0 sec   610 KBytes  5.00 Mbits/sec
[  3]  0.0-10.0 sec  5.96 MBytes  5.00 Mbits/sec
[  3] Sent 4253 datagrams
```

**Step 13** - Verify that the DUT resumes operation when receiving multicast data from the external AP.

### 5.1.13 Configure energy detection (ED) and TX power

This section provides the guidelines to load the energy detection (ED) configuration for the adaptivity test and to load the TX power table.

#### 5.1.13.1 Load ED-MAC configuration

This section shows how to load the converted ED-MAC configuration.

The ED-MAC configuration binary file from the module vendor is activated during the wireless driver load time. In the following example, the driver loads *ed_mac.bin* using `init_hostcmd_cfg` parameter:

```
nano /lib/firmware/nxp/wifi_mod_para.conf
SD8987 = {
      cfg80211_wext=0xf
      wfd_name=p2p
      cal_data_cfg=none
      max_vir_bss=1
     init_hostcmd_cfg=nxp/ed_mac.bin
}
```

Issue the command to load the module:

```
modprobe moal mod_para=nxp/wifi_mod_para.conf
```

#### 5.1.13.2 Load TX power table configuration

This section provides the instructions to load the TX power table configuration.

The TX Power table configuration binary file the from module vendor is activated during the wireless driver load time. In the following example, the driver loads *txpower_US.bin* using `txpwrlimit_cfg` parameter:

```
nano /lib/firmware/nxp/wifi_mod_para.conf
SD8987 = {
      cfg80211_wext=0xf
      wfd_name=p2p
      cal_data_cfg=none
      max_vir_bss=1
     txpwrlimit_cfg=nxp/txpower_US.bin
}
```

Issue the command to load the module:

```
modprobe moal mod_para=nxp/wifi_mod_para.conf
```

### 5.1.14 Operating dynamic frequency selection (DFS) region

A DFS-enabled system goes through the following sequence:

1. The Access Point selects a channel in 5 GHz frequency band and monitors that channel for potential radar interference for a minimum listening time (CAC time). No transmissions can occur during this period.
2. If some interference is detected, the system must go and select another channel and repeat the channel availability check on the new channel (the original channel is added to a list of channels with radar).
3. Once a channel has been selected and no radar signal is detected for CAC time, the network starts to use that channel.
4. While operating on the channel, the device continuously checks for potential radar signal. If some interference is detected, the device adds `CHAN_SW_ANN` IE (channel switch announcement IE) to the action frame, the AP Beacon, or the probe responses. This informs the connected devices that a channel change is pending.
5. The AP stops the communication on the current channel. The channel is added to the list of channels with radar. The access point selects a new channel (one that is not on the list). The sequence starts again with a channel availability check if another DFS channel is selected.
6. Once a channel is included in the list of channels with radar, the access point avoids using that channel again before the expiring NOP time for that channel.

Content of DFS hostapd configuration file:

```
ctrl_interface=/var/run/hostapd
interface=uap0
driver=nl80211
hw_mode=a
ssid=DFS_AP
channel=52
ieee80211d=1
ieee80211n=1
ieee80211ac=1
ieee80211h=1
country_code=US
ht_capab=[HT40-][HT40+][SHORT-GI-20][SHORT-GI-40]
vht_oper_chwidth=1
vht_capab=[SHORT-GI-80][SU-BEAMFORMER][SU-BEAMFORMEE][MU-BEAMFORMEE]
vht_oper_centr_freq_seg0_idx=58
```

### 5.1.15 Software antenna diversity (SAD)

This section explains how to configure and enable/disable the software antenna diversity feature on a 1x1 device with 2 antennas.

**Step 1** - Verify or get the current configurations

```
cat /proc/mwlan/adapter0/config
```

For example:

- If software antenna diversity is enabled, the command output is as follows:

```
hardware_status=0
netlink_num=31
drv_mode=7
hssetpara=7,0xff,200,400
rf_test_mode=0
antcfg=0x303 0x303
```

- If the antenna configuration is set to one antenna, the command output is as follows:

```
hardware_status=0
netlink_num=31
drv_mode=7
hssetpara=7,0xff,200,400
rf_test_mode=0
antcfg=0x301 0x301
```

**Step 2** - Enable Tx/Rx software antenna diversity

```
echo "antcfg=0xffff" > /proc/mwlan/adapter0/config
```

**Step 3** Set antenna evaluate time interval to 6 s

```
echo "antcfg=0xffff 0x1770" > /proc/mwlan/adapter0/config
```

***Note:*** *The software antenna diversity mode must be enabled when setting software antenna diversity evaluate time interval. The default evaluate time interval is 6 s (0x1770)*

**Step 4** - Select antenna 1 for TX/RX and disable software antenna diversity

```
echo "antcfg=1" > /proc/mwlan/adapter0/config
```

**Step 5** - Select antenna 2 for TX/RX and disable software antenna diversity

```
echo "antcfg=2" > /proc/mwlan/adapter0/config
```

### 5.1.16 Tunneled direct link support (TDLS)

TDLS enables Wi-Fi devices to set secure direct links and transfer data directly. IEEE 802.11z amendment extends the direct link support (DLS) feature introduced by 802.11e. With DLS, STAs can establish a direct link without any knowledge of the AP where the STAs encapsulate the direct link signaling protocol into the data frame using a specific ether type.

IEEE 802.11z defines the following mechanism for TDLS:

- TDLS discovery: discovering TDLS capable STAs in the BSS to form direct links
- TDLS setup: ability to set up a direct link (with optional security) with a TDLS capable peer STAs.
- TDLS direct link power save mode (PSM):
  - TDLS peer PSM: scheduled access of direct link on a negotiated wakeup schedule.
  - Peer UAPSD: The TDLS STA buffers the traffic for the peer STA in power save mode and signals the traffic through the AP path. The peer STA receives the traffic by initiating a service period.
- TDLS off-channel operation: the Wi-Fi devices supporting TDLS feature can go off-channel and/or off band while maintaining the association. This can be used to choose a channel with better throughput or latency to meet the application requirements. For example,a 2.4 GHz device can move to 5 GHz band while maintaining direct link to avoid interference. However, in this case the Wi-Fi device must obey regulatory rules and avoid interference with Radars. TDLS does not define DFS algorithm nor the channel selection rules.

**Establish the AP link and TDLS link**

- Connect the STAs to the AP.

Use `wpa_supplicant` to connect the STAs to the AP. Make sure CONFIG_TDLS is enabled in supplicant. Verify that the supports host-based TDLS.

Example of configuration file for the AP-link connection:

```
ctrl_interface=/var/run/wpa_supplicant
update_config=1
network={
    ssid="TestAP"
    psk="1234567890"
    }
```

- Set the security mode of the AP to OPEN/ AES.
- Establish the infra connection to the AP.
  Issue `wpa_supplicant` command:

```
./wpa_supplicant i mlan0 c wpa_supplicant.conf Dnl80211 ddd
```

- Issue `wpa_cli` command.

```
./wpa_cli
```

***Note:*** *Connect both clients to the same AP.*

- Find TDLS device.

```
> TDLS_DISCOVER <Peer MAC address>
```

Example of command:

```
> TDLS_DISCOVER FF:FF:FF:FF:FF:FF
```

The Wi-Fi device which supports TDLS in the same BSS responds to the discovery request and sends the discover response.

- Set up TDLS direct link.

```
> TDLS_SETUP <Peer MAC address>
```

***Note:*** *Only one client-side must set the command `TDLS_SETUP`. If TDLS link setup is a success, the traffic to the peer device transmits via TDLS direct link. The address 1 and address 2 of the packet header are the destination address and the source address respectively.*

Command to tear down TDLS direct link:

```
> TDLS_TEARDOWN <Peer Mac address>
```

If TDLS link is teared down, the traffic restores the transmit operation through the AP path.

**Verify TDLS connection**

- Check */proc/mwlan/adapter0/mlan0/debug*

The `mlan0-debug` parameter is introduced for TDLS. When the link is up, the peer mac address, `snr` and `nf` are present.

```
Tx BA stream table:
tid = 0, ta = 08:d4:2b:13:99:f6 amsdu=0
Rx reorder table:
tid = 0, ta = 08:d4:2b:13:99:f6, start_win = 3210, win_size = 16, amsdu=0
buffer: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
tdls peer : 08:d4:2b:13:99:f6 snr=0 nf=0
htcap: 2d 1a fe 01 1b ff ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00
Extcap: 7f 05 00 00 00 50 20 00 00 00
vhtcap: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

The driver log message `TDLS_ENABLE_LINK/ TDLS_DISABLE_LINK` is present in the driver log when `tdls_setup` and `tdls_teardown` commands are executed respectively.

### 5.1.17 UNII-4 channel support

Wi-Fi 5 standard added the support for 160 MHz channel width to improve the performance. Prior to Wi-Fi 5, the maximum channel width was 80 MHz.

The first three UNII groups can accommodate only two 160 MHz channels, which also include DFS space. As a consequence, the Wi-Fi devices may be disconnected when RADAR signals are observed. To avoid the disconnection, most of the Wi-Fi devices (routers, access points) avoid the use of 160 MHz channel width and instead opt for the more reliable 80 MHz. The performance may be affected.

To help with this scenario, the UNII4 portion of the 5 GHz band, also referred to as the 5.9 GHz band, is newly supported. This extension to the 5 GHz band adds new channels such as channels 169, 173, and 177 (channel 181 is not usable). The new channels can be combined to form a third 160 MHz channel in Wi-Fi 6 standard.

### 5.1.18 Statistics and counters

Many statistics and counters are available for Wi-Fi TX/RX operations, for example frame counts, retransmissions, and error counts. Those counters are available as part of standard /proc interface.

Table 6 lists the statistics and counters shown in /proc.

**Table 6. Statistics and counters**

| Field name | Type | Description |
|---|---|---|
| dot11GroupTransmittedFrameCount | UINT32 | Increments when the multicast bit is set in the destination MAC |
| dot11FailedCount | UINT32 | Increments when an MSDU is not transmitted successfully |
| dot11RetryCount | UINT32 | Increments when an MSDU is successfully transmitted after 1 or more retransmissions |
| dot11MultipleRetryCount | UINT32 | Increments when an MSDU is successfully transmitted after more than 1 retransmission |
| dot11FrameDuplicateCount | UINT32 | Increments when a frame is received that the Sequence Control field is indicating a duplicate count |
| dot11RTSSuccessCount | UINT32 | Increments when a CTS is received in response to an RTS |
| dot11RTSFailureCount | UINT32 | Increments when a CTS is not received in response to an RTS |
| dot11ACKFailureCount | UINT32 | Increments when an ACK is not received when expected |
| dot11ReceivedFragmentCount | UINT32 | Increments for each successfully received MPDU of type Data or Management |
| dot11GroupReceivedFrameCount | UINT32 | Increments when a MSDU is received with the multicast bit set in the destination MAC address |
| dot11FCSErrorCount | UINT32 | Increments when an FCS error is detected in a received MPDU |
| dot11TransmittedFrameCount | UINT32 | Increments for each successfully transmitted MSDU |
| Wepicverrcnt | UINT32[4] | Increment when WEP decryption error for key index 0–3 |
| beaconReceivedCount | UINT32 | Increments when a beacon is received |
| beaconMissedCount | UINT32 | Increments when an beacon is not received when expected |
| dot11TransmittedFragmentCount | UINT32 | Increments for each successfully transmitted MPDU |
| dot11QosTransmittedFragmentCount | UINT32[8] | Increments for each successfully transmitted QOS Data MPDU |
| dot11QosFailedCount | UINT32[8] | increments when an MSDU, for a particular UP, is not transmitted successfully due to the number of transmit attempts exceeding either the dot11ShortRetryLimit or dot11 LongRetryLimit. |
| dot11QosRetryCount | UINT32[8] | increments when an MSDU, of a particular UP, is successfully transmitted after one or more retransmissions |
| dot11QosMultipleRetryCount | UINT32[8] | increments when an MSDU, of a particular UP, is successfully transmitted after more than one retransmissions |
| dot11QosFrameDuplicateCount | UINT32[8] | increments when a frame, of a particular UP, is received that the Sequence Control field indicates is a duplicate |

**Table 6. Statistics and counters***...continued*

| Field name | Type | Description |
|---|---|---|
| dot11QosRTSSuccessCount | UINT32[8] | increments when a CTS frame is received in response to an RTS that has been sent for the transmission of an MPDU of a particular UP |
| dot11QosRTSFailureCount | UINT32[8] | increments when a CTS frame is not received in response to an RTS that has been sent for the transmission of an MPDU of a particular UP |
| dot11QosACKFailureCount | UINT32[8] | increments when an Ack frame is not received in response to an MPDU of a particular UP |
| dot11QosReceivedFragmentCount | UINT32[8] | incremented for each successfully received MPDU with the Type subfield equal to Data of a particular UP |
| dot11QosTransmittedFrameCount | UINT32[8] | increments for each successfully transmitted MSDU of a particular UP |
| dot11QosDiscardedFrameCount | UINT32[8] | increments for each Discarded MSDU of a particular UP |
| dot11QosMPDUsReceivedCount | UINT32[8] | increments for each received MPDU of a particular TID |
| dot11QosRetriesReceivedCount | UINT32[8] | increments for each received MPDU of a particular TID with the Retry subfield equal to 1 |
| dot11RSNAStatsCMACICVErrors | UINT32 | Counts the number of ICV Errors |
| dot11RSNAStatsCMACReplays | UINT32 | The number of received MPDUs discarded by the CMAC replay errors |
| dot11RSNAStatsRobustMgmt CCMPReplays | UINT32 | The number of received robust Management frames discarded due to CCMP replay errors |
| dot11RSNAStatsTKIPICVErrors | UINT32 | Counts the number of TKIP ICV errors encountered when decrypting packets for the STA. |
| dot11RSNAStatsTKIPReplays | UINT32 | Counts the number of TKIP replay errors detected |
| dot11RSNAStatsCCMPDecryptErrors | UINT32 | The number of received MPDUs discarded by the CCMP decryption algorithm |
| dot11RSNAstatsCCMPReplays | UINT32 | The number of received MPDUs discarded by the CMAC replay errors |
| dot11TransmittedAMSDUCount | UINT32 | incremented for an acknowledged A-MSDU frame with an individual address in the Address 1 field or an A-MSDU frame with a group address in the Address 1 field |
| dot11FailedAMSDUCount | UINT32 | incremented when an A-MSDU is not transmitted successfully due to the number of transmit attempts exceeding either the dot11ShortRetryLimit or dot11LongRetry Limit |
| dot11RetryAMSDUCount | UINT32 | incremented when an A-MSDU is successfully transmitted after one or more retransmissions |
| dot11MultipleRetryAMSDUCount | UINT32 | incremented when an A-MSDU is successfully transmitted after more than one retransmission |
| dot11TransmittedOctetsInAMSDUCount | UINT32 | incremented by the number of octets in the frame body of an A-MSDU frame when an A-MSDU frame is successfully transmitted |
| dot11AMSDUAckFailureCount | UINT32 | incremented when an acknowledgment to an A-MSDU is not received when expected. This acknowledgment can be in an Ack or Block-Ack frame |

**Table 6. Statistics and counters***...continued*

| Field name | Type | Description |
|---|---|---|
| dot11ReceivedAMSDUCount | UINT32 | incremented for a received A-MSDU frame with the station's MAC address in the Address 1 field or an A-MSDU frame with a group address in the Address 1 field |
| dot11ReceivedOctetsInAMSDUCount | UINT32 | incremented by the number of octets in the frame body of an A-MSDU frame when an A-MSDU frame is received |
| dot11TransmittedAMPDUCount | UINT32 | incremented when an A-MPDU is transmitted |
| dot11TransmittedMPDUsInAMPDUCount | UINT32 | increment by the number of MPDUs in the A-MPDU when an A-MPDU is transmitted |
| dot11TransmittedOctetsInAMPDUCount | UINT32 | incremented by the number of octets in the A-MPDU frame when an A-MPDU frame is transmitted |
| dot11AMPDUReceivedCount | UINT32 | incremented when the MAC receives an A-MPDU from the PHY |
| dot11MPDUInReceivedAMPDUCount | UINT32 | incremented by the number of MPDUs received in the A-MPDU when an A-MPDU is received |
| dot11ReceivedOctetsInAMPDUCount | UINT32 | incremented by the number of octets in the A-MPDU frame when an A-MPDU frame is received |
| dot11AMPDUDelimiterCRCErrorCount | UINT32 | incremented when an MPDU delimiter has a CRC error when this is the first CRC error in the received A-MPDU or when the previous delimiter has been decoded correctly |

Example of command:

```
cat /proc/mwlan/adapter0/mlan0/log
```

Example of command output:

```
dot11GroupTransmittedFrameCount    0
dot11FailedCount                   0
dot11RetryCount                    1
dot11MultipleRetryCount            1
dot11FrameDuplicateCount           0
dot11RTSSuccessCount               0
dot11RTSFailureCount               0
dot11ACKFailureCount               4
dot11ReceivedFragmentCount         6
dot11GroupReceivedFrameCount       1
dot11FCSErrorCount                 13
dot11TransmittedFrameCount         3
wepicverrcnt-1                     0
wepicverrcnt-2                     0
wepicverrcnt-3                     0
wepicverrcnt-4                     0
beaconReceivedCount                346
beaconMissedCount                  0
dot11TransmittedFragmentCount      3
dot11QosTransmittedFragmentCount   0 0 0 0 0 0 0 3
dot11QosFailedCount                0 0 0 0 0 0 0 0
dot11QosRetryCount                 0 0 0 0 0 0 0 1
dot11QosMultipleRetryCount         0 0 0 0 0 0 0 1
dot11QosFrameDuplicateCount        0 0 0 0 0 0 0 0
dot11QosRTSSuccessCount            0 0 0 0 0 0 0 0
dot11QosRTSFailureCount            0 0 0 0 0 0 0 0
dot11QosACKFailureCount            0 0 0 0 0 0 0 4
dot11QosReceivedFragmentCount      2 0 0 0 0 0 3 0
dot11QosTransmittedFrameCount      0 0 0 0 0 0 0 3
dot11QosDiscardedFrameCount        0 0 0 0 0 0 0 0
dot11QosMPDUsReceivedCount         2 0 0 0 0 0 3 0
dot11QosRetriesReceivedCount       1 0 0 0 0 0 0 0
dot11RSNAStatsCMACICVErrors          0
dot11RSNAStatsCMACReplays            0
dot11RSNAStatsRobustMgmtCCMPReplays  0
dot11RSNAStatsTKIPICVErrors          0
dot11RSNAStatsTKIPReplays            0
dot11RSNAStatsCCMPDecryptErrors      0
dot11RSNAstatsCCMPReplays            0
dot11TransmittedAMSDUCount           0
dot11FailedAMSDUCount                0
dot11RetryAMSDUCount                 0
dot11MultipleRetryAMSDUCount         0
dot11TransmittedOctetsInAMSDUCount   0
dot11AMSDUAckFailureCount            0
dot11ReceivedAMSDUCount              0
dot11ReceivedOctetsInAMSDUCount      0
dot11TransmittedAMPDUCount           35
dot11TransmittedMPDUsInAMPDUCount    35
dot11TransmittedOctetsInAMPDUCount   0
dot11AMPDUReceivedCount              12789179
dot11MPDUInReceivedAMPDUCount        15974027
dot11ReceivedOctetsInAMPDUCount      349
dot11AMPDUDelimiterCRCErrorCount     161520711
```

### 5.1.19 Measuring the throughput

This section describes the throughput test using the iPerf tool. iPerf is an open source tool used for network throughput measurements. The tool can test either TCP or UDP throughput. To perform an iPerf test the user must configure an iPerf server to act as a traffic sink, and an iPerf client to generate network traffic.

Figure 38 shows the test setup with iPerf.

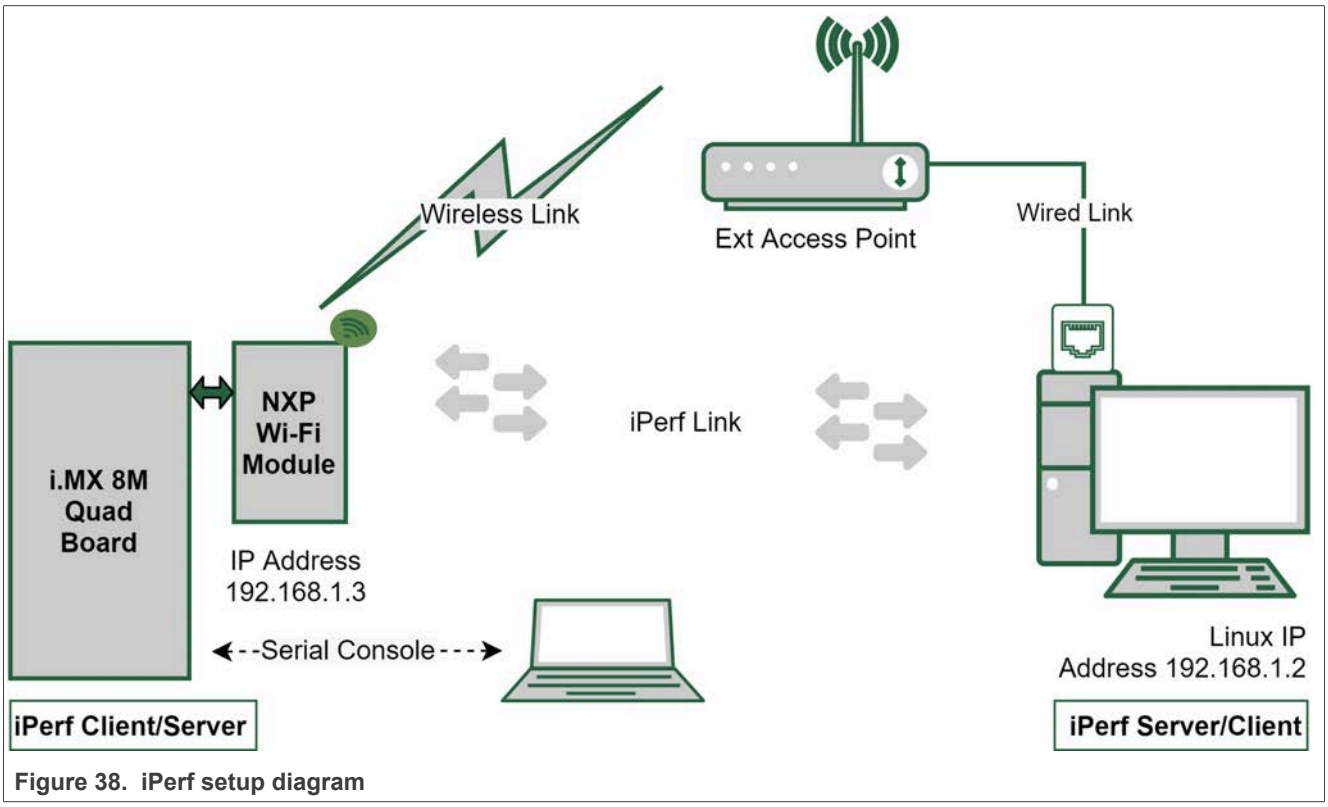



**Figure 38. iPerf setup diagram**

**Step 1** – Connect the NXP Wi-Fi device with the access point.

Follow the instructions given in Association to connect the Wi-Fi device (attached to i.MX platform) with the Access Point (External Wi-Fi router).

Use the following steps to test the throughput of the network established between the AP and the station or client.

**Step 2** – Get the IP address of the access point.

Run the command:

```
udhcpc -i mlan0
```

Command output example:

```
udhcpc -i mlan0
udhcpc: started, v1.31.0
udhcpc: sending discover
udhcpc: sending select for 192.168.1.3
udhcpc: lease of 192.168.1.3 obtained, lease time 600
/etc/udhcpc.d/50default: Adding DNS 192.168.1.1
```

**Step 3** – Start iPerf on the server.

Use the following command to start iPerf on the Ubuntu station connected with the external Wi-Fi Router as a server.

```
root@ubuntu-desktop:/# iperf -s -i 1
```

Command output example on the server after iPerf started on the client:

```
------------------------------------------------------------
Server listening on 5201
------------------------------------------------------------
Accepted connection from 192.168.1.3, port 47590
[  5] local 192.168.1.2 port 5201 connected to 192.168.1.3 port 55286
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  5]  0.00-1.00  sec  XX.XX MBytes  XXX.X Mbits/sec  X.XXX ms  XX/XXXX  (XX%)
[  5]  1.00-2.00  sec  XX.XX MBytes  XXX.X Mbits/sec  X.XXX ms  XX/XXXX  (XX%)
[  5]  2.00-3.00  sec  XX.XX MBytes  XXX.X Mbits/sec  X.XXX ms  XX/XXXX  (XX%)
[  5]  3.00-4.00  sec  XX.XX MBytes  XXX.X Mbits/sec  X.XXX ms  XX/XXXX  (XX%)
[  5]  4.00-5.00  sec  XX.XX MBytes  XXX.X Mbits/sec  X.XXX ms  XX/XXXX  (XX%)
[  5]  5.00-6.00  sec  XX.XX MBytes  XXX.X Mbits/sec  X.XXX ms  XX/XXXX  (XX%)
[  5]  6.00-7.00  sec  XX.XX MBytes  XXX.X Mbits/sec  X.XXX ms  XX/XXXX  (XX%)
[  5]  7.00-8.00  sec  XX.XX MBytes  XXX.X Mbits/sec  X.XXX ms  XX/XXXX  (XX%)
[  5]  8.00-9.00  sec  XX.XX MBytes  XXX.X Mbits/sec  X.XXX ms  XX/XXXX  (XX%)
[  5]  9.00-10.00 sec  XX.XX MBytes  XXX.X Mbits/sec  X.XXX ms  XX/XXXX  (XX%)
[  5] 10.00-10.52 sec  XX.XX MBytes  XXX.X Mbits/sec  X.XXX ms  XX/XXXX  (XX%)
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  5]  0.00-10.52 sec  XXX.X MBytes  XXX.X Mbits/sec  X.XXX ms  XXXX/XXXXX (XX%)
------------------------------------------------------------
```

**Step 4** – Start iPerf on the client.

The iPerf server is running and the system is connected to the external AP.

Run the command to start iPerf on i.MX 8M Quad station as a client. The command takes iPerf server IP address as an argument.

```
iperf -c 192.168.1.2 -u -b 50M -i 1
```

Command output example:

```
Connecting to host 192.168.1.2, port 5201
[  5] local 192.168.1.3 port 55286 connected to 192.168.1.2 port 5201
[ ID] Interval           Transfer     Bitrate          Total Datagrams
[  5]   0.00-1.00   sec  XX.XX MBytes  XXX.X Mbits/sec  XXXX
[  5]   1.00-2.00   sec  XX.XX MBytes  XXX.X Mbits/sec  XXXX
[  5]   2.00-3.00   sec  XX.XX MBytes  XXX.X Mbits/sec  XXXX
[  5]   3.00-4.00   sec  XX.XX MBytes  XXX.X Mbits/sec  XXXX
[  5]   4.00-5.00   sec  XX.XX MBytes  XXX.X Mbits/sec  XXXX
[  5]   5.00-6.00   sec  XX.XX MBytes  XXX.X Mbits/sec  XXXX
[  5]   6.00-7.00   sec  XX.XX MBytes  XXX.X Mbits/sec  XXXX
[  5]   7.00-8.00   sec  XX.XX MBytes  XXX.X Mbits/sec  XXXX
[  5]   8.00-9.00   sec  XX.XX MBytes  XXX.X Mbits/sec  XXXX
[  5]   9.00-10.00  sec  XX.XX MBytes  XXX.X Mbits/sec  XXXX
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-10.00  sec  XXX.X MBytes  XXX.X Mbits/sec  X.XXX ms   XXXX/XXXXX (XX%)  sender
[  5]   0.00-10.00  sec  XXX.X MBytes  XXX.X Mbits/sec  X.XXX ms   XXXX/XXXXX (XX%)  receiver
iperf Done.
```

### 5.1.20 Throughput performance tuning

To fine tune the network stack per the platform and use case, a few parameters are available.

Examples for Linux platform:

```
echo 12582912 > /proc/sys/net/core/rmem_max
echo 12582912 > /proc/sys/net/core/wmem_max
echo 12582912 > /proc/sys/net/core/rmem_default
echo 12582912 > /proc/sys/net/core/wmem_default
echo '10240 87380 12582912' > /proc/sys/net/ipv4/tcp_rmem
echo '10240 87380 12582912' > /proc/sys/net/ipv4/tcp_wmem
echo '12582912 12582912 12582912' > /proc/sys/net/ipv4/tcp_mem
echo '12582912 12582912 12582912' > /proc/sys/net/ipv4/udp_mem
echo 1310720 > /proc/sys/net/ipv4/tcp_limit_output_bytes
echo 1 > /proc/sys/net/ipv4/route/flush
echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
echo performance > /sys/devices/system/cpu/cpu2/cpufreq/scaling_governor
echo performance > /sys/devices/system/cpu/cpu3/cpufreq/scaling_governor
```

#### Trade-off to maintain

Note that these are only suggested starting points. Changing the parameter values from the default value can increase the resources used on the system. Increasing the values can improve the performance but you must experiment with different values to determine what works best for a given system, workload and traffic type.

For the throughput performance that relates to driver parameters, use the default settings (recommendation).

**Table 7. Throughput performance related driver parameters**

| module_param | Description | Default | Comment |
|---|---|---|---|
| tx_work | 0: Disable TX work queue;<br>1: Enable TX work queue | 1 | Use tx_work queue for better performance.<br>Trade-off: CPU utilization |
| tx_skb_clone | 0: Disable TX skb clone;<br>1: Enable TX skb clone | 1 | Use tx_skb_clone for better performance.<br>Trade-off: CPU utilization |
| rx_work | 0: Auto, enable/disable RX work queue on multi-core/single core system;<br>1: Enable RX work queue;<br>2: Disable RX work queue | 0 | Use rx_work queue for better performance.<br>Trade-off: CPU utilization |
| net_rx | 0: Use netif_rx/netif_rx_ni API in RX;<br>1: Use netif_receive_skb API in RX | 1 | Use netif_receive_skb API for better performance in polling case |
| amsdu_deaggr | 0: Buffer copy in A-MSDU de-aggregation;<br>1: Avoid buffer copy in A-MSDU de-aggregation | 1 | Avoid buffer copy for better performance and CPU utilization |
| rps | 0: Disable RPS (Receive Packet Steering);<br>bit0-bit4 (0x1-0xf): Enables RPS on specific CPU | 0 | Use rps for better performance.<br>Trade-off: complexity and stability |
| pmqos | 0: Disable PM QoS (Power Management Quality of Service);<br>1: Enable PM QoS | 1 | Enable pmqos for better performance.<br>Trade-off: complexity and stability |
| napi | 0: Disable napi related APIs<br>1: Enable napi related APIs | 0 | Use napi related APIs for better CPU utilization.<br>Trade-off: complexity and stability |

## 5.2 Bluetooth

### 5.2.1 Bluetooth Classic

#### 5.2.1.1 Scan, pair and connect to Bluetooth classic/Bluetooth LE

This section describes the configuration steps to scan, pair and connect with a Remote Bluetooth device from NXP-based wireless module. BlueZ Stack provides the *bluetoothctl* command line utility to connect with a Remote Bluetooth device.

Use the following steps to scan, pair and connect the remote Bluetooth Classic/Bluetooth Low Energy device using *bluetoothctl* utility.

**Start *bluetoothctl***

Start *bluetoothctl* to interact with the Bluetooth daemon from the command line.

***Note:*** *This section describes the options required to scan, pair and connect the Bluetooth classic/LE device. For more options use "help" command in bluetoothctl.*

```
bluetoothctl
```

Command output example:

```
Agent registered
[bluetooth]# version
Version 5.65
[bluetooth]#
```

**Authenticate**

Since the pairing procedure will involve authentication by PIN, it is required to register with an authentication agent as per the peer device capabilities. The agent handles the PIN prompt.

***Note:*** `agent on` *uses option NoInputNoOutput.*

`Agent on` command:

```
[bluetooth]# agent on
```

`Agent help` command:

```
[bluetooth]# agent help
```

Command output example:

```
Enable/disable agent with given capability
Usage:
agent <on/off/capability>
[bluetooth]#
Agent is already registered
```

`default-agent` command:

```
[bluetooth]# default-agent
```

Command output example:

```
Default agent request successful
```

### Start scanning

Run the following command to start scanning for remote Bluetooth Classic/Bluetooth LE device(s).

```
[bluetooth]# scan on
```

Command output example:

```
Discovery started
[CHG] Controller 00:50:43:24:34:F7 Discovering: yes
[NEW] Device B4:F5:00:31:CB:4E Moto E
```

### Stop scanning

Stop the scanning and check for available remote Bluetooth Classic/Bluetooth LE device(s) for pairing.

```
[bluetooth]# scan off
```

Command output example:

```
Discovery stopped
[CHG] Controller 00:50:43:24:34:F7
Discovering: no
[bluetooth]# devices
Device B4:F5:00:31:CB:4E Moto E
[bluetooth]#
```

### Start pairing

Start pairing with the remote Bluetooth Classic/Bluetooth LE device.

```
[bluetooth]# pair B4:F5:00:31:CB:4E
```

Command output example:

```
Attempting to pair with B4:F5:00:31:CB:4E
[CHG] Device B4:F5:00:31:CB:4E Connected: yes
Request confirmation
[agent] Confirm passkey 666330 (yes/no):
```

### Confirm pairing

If the passkey matches, type yes to complete the pairing procedure and establish a connection:

```
[agent] Confirm passkey 666330 (yes/no): yes
[CHG] Device B4:F5:00:31:CB:4E Modalias: bluetooth:v000Fp1200d1436
[CHG] Device B4:F5:00:31:CB:4E UUIDs: 00001105-0000-1000-8000-00805f9b34fb
[CHG] Device B4:F5:00:31:CB:4E UUIDs: 0000110a-0000-1000-8000-00805f9b34fb
[CHG] Device B4:F5:00:31:CB:4E UUIDs: 0000110c-0000-1000-8000-00805f9b34fb
[CHG] Device B4:F5:00:31:CB:4E UUIDs: 0000110e-0000-1000-8000-00805f9b34fb
[CHG] Device B4:F5:00:31:CB:4E UUIDs: 00001112-0000-1000-8000-00805f9b34fb
[CHG] Device B4:F5:00:31:CB:4E UUIDs: 00001115-0000-1000-8000-00805f9b34fb
[CHG] Device B4:F5:00:31:CB:4E UUIDs: 00001116-0000-1000-8000-00805f9b34fb
[CHG] Device B4:F5:00:31:CB:4E UUIDs: 0000111f-0000-1000-8000-00805f9b34fb
[CHG] Device B4:F5:00:31:CB:4E UUIDs: 0000112f-0000-1000-8000-00805f9b34fb
[CHG] Device B4:F5:00:31:CB:4E UUIDs: 00001132-0000-1000-8000-00805f9b34fb
[CHG] Device B4:F5:00:31:CB:4E UUIDs: 00001200-0000-1000-8000-00805f9b34fb
[CHG] Device B4:F5:00:31:CB:4E UUIDs: 00001800-0000-1000-8000-00805f9b34fb
[CHG] Device B4:F5:00:31:CB:4E UUIDs: 00001801-0000-1000-8000-00805f9b34fb
[CHG] Device B4:F5:00:31:CB:4E ServicesResolved: yes
[CHG] Device B4:F5:00:31:CB:4E Paired: yes
Pairing successful
[CHG] Device B4:F5:00:31:CB:4E ServicesResolved: no
[CHG] Device B4:F5:00:31:CB:4E Connected: no
[bluetooth]#
```

### Connect with the remote Bluetooth Classic/Bluetooth LE device

The remote Bluetooth Classic/Bluetooth LE device is not yet in the connected state. Run the following command to set the connected state.

- Command to trust the device:

```
[bluetooth]# trust B4:F5:00:31:CB:4E
```

Command output example:

```
[CHG] Device B4:F5:00:31:CB:4E Trusted: yes
Changing B4:F5:00:31:CB:4E trust succeeded
```

- Command to connect the Bluetooth device:

```
[bluetooth]# connect B4:F5:00:31:CB:4E
```

Command output example:

```
Attempting to connect to B4:F5:00:31:CB:4E
[CHG] Device B4:F5:00:31:CB:4E Connected: yes
Connection successful
[CHG] Device B4:F5:00:31:CB:4E ServicesResolved: yes
[Moto E]#
```

### Disconnect the remote Bluetooth Classic/Bluetooth LE device

```
[Moto E]# disconnect B4:F5:00:31:CB:4E
```

Command output example:

```
Attempting to disconnect from B4:F5:00:31:CB:4E
[CHG] Device B4:F5:00:31:CB:4E ServicesResolved: no
Successful disconnected
[CHG] Device B4:F5:00:31:CB:4E Connected: no
[bluetooth]#
```

### Remove the remote Bluetooth Classic/Bluetooth LE device

```
[bluetooth]# remove B4:F5:00:31:CB:4E
```

Command output example:

```
[DEL] Device B4:F5:00:31:CB:4E Moto E
Device has been removed
[bluetooth]#
```

### Exit from *bluetoothctl* command line interface

```
[Moto E]# quit
```

#### 5.2.1.2 Advanced audio distribution profile (A2DP)

This section shows how to configure i.MX 8M EVK board with A2DP sink or source profile.

The following PipeWire packages are used to configure A2DP profiles:

- **PipeWire:** a low-latency audio server that serves as a replacement for PulseAudio and JACK.
- **pipewire-audio:** core PipeWire package for handling audio processing and routing.

Command to check the outcome of the server swap:

```
$ pactl info
...
Server Name: PipeWire x.y.z
...
```

- **WirePlumber:** utility used as session manager for PipeWire.

#### 5.2.1.2.1 A2DP sink profile configuration

Steps to configure i.MX 8M EVK board with A2DP sink profile:

**Step 1** – Start PipeWire on i.MX 8M EVK board.

```
$ root@imx8mmevk:~# systemctl --user start pipewire wireplumber
```

**Step 2** – Connect with a peer Bluetooth device that supports Audio Source Profile.

Refer to Section 5.2.1.1.

**Step 3** – Verify the audio source profile of the connected Bluetooth device.

Command using the command line utility for Bluetooth operations (bluetoothctl):

```
$ root@imx8mmevk:~# bluetoothctl
[test_phone]# info 48:74:12:C2:F2:82
Device 48:74:12:C2:F2:82 (public)
        Name: test_phone
        Alias: test_phone
        Class: 0x005a020c (5898764)
        Icon: phone
        Paired: yes
        Bonded: yes
        Trusted: no
        Blocked: no
        Connected: yes
        LegacyPairing: no
        UUID: Vendor specific           (00000000-0000-0000-0000-000000000000)
        UUID: OBEX Object Push          (00001105-0000-1000-8000-00805f9b34fb)
        UUID: Audio Source              (0000110a-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control        (0000110e-0000-1000-8000-00805f9b34fb)
        UUID: Headset AG                (00001112-0000-1000-8000-00805f9b34fb)
        UUID: PANU                      (00001115-0000-1000-8000-00805f9b34fb)
        UUID: NAP                       (00001116-0000-1000-8000-00805f9b34fb)
        UUID: Handsfree Audio Gateway   (0000111f-0000-1000-8000-00805f9b34fb)
        UUID: Phonebook Access Server   (0000112f-0000-1000-8000-00805f9b34fb)
        UUID: Message Access Server     (00001132-0000-1000-8000-00805f9b34fb)
        UUID: PnP Information           (00001200-0000-1000-8000-00805f9b34fb)
        UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
        UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
        UUID: Unknown                   (0000aa15-0000-1000-8000-00805f9b34fb)
        UUID: Vendor specific           (a49eaa15-cb06-495c-9f4f-bb80a90cdf00)
        Modalias: bluetooth:v001Dp1200d1436
[test_phone]#
```

**Step 4** – Check the available sink cards in the audio profile

```
$ root@imx8mmevk:~# wpctl status
        PipeWire 'pipewire-0' [1.0.5, root@imx8mmevk, cookie:815200939]
        └─ Clients:
        32. WirePlumber                         [1.0.5, root@imx8mmevk, pid:559]
        40. WirePlumber [export]                [1.0.5, root@imx8mmevk, pid:559]
        75. wpctl                               [1.0.5, root@imx8mmevk, pid:578]
        Audio
        ├─ Devices: │      41. Built-in Audio                      [alsa]
        │           42. Built-in Audio                    [alsa]
        │           43. Built-in Audio                    [alsa]
        │           44. Built-in Audio                    [alsa]
        │           74. OnePlus Nord CE 2 Lite 5G         [bluez5]
        │
        ├─ Sinks:
        │           46. Built-in Audio Mono               [vol: 0.40]
        │           49. Built-in Audio Stereo             [vol: 0.40]
        │       *   50. Built-in Audio Stereo             [vol: 0.40]
        │
        ├─ Sources:
        │       *   45. Built-in Audio Mono               [vol: 1.00]
        │           47. Built-in Audio Stereo             [vol: 1.00]
        │           48. Built-in Audio Stereo             [vol: 1.00]
        │
        ├─ Filters:
        │
        └─ Streams:

        Video
        ├─ Devices:
        │           54. i.MX6S_CSI                        [v4l2]
        │           55. vsi_v4l2enc                       [v4l2]
        │           56. vsi_v4l2dec                       [v4l2]
        │
        ├─ Sinks:
        │
        ├─ Sources:
        │       *   59. i.MX6S_CSI (V4L2)
        │
        ├─ Filters:
        │
        └─ Streams:

        Settings
        └─ Default Configured Devices:
        0 Audio/Sink    alsa_output.platform-sound-wm8524.stereo-fallback
        1 Audio/Source  alsa_input.platform-sound-bt-sco.mono-fallback
        root@imx8mmevk:~#
```

**Step 5** – Look for `wm8524` as value of `alsa card_name` in the list of sink cards.

```
$ root@imx8mmevk:~# wpctl inspect 50
id 50, type PipeWire:Interface:Node
alsa.card = "1"
alsa.card_name = "wm8524-audio"
alsa.class = "generic"
alsa.device = "0"
alsa.id = "wm8524audio"
alsa.long_card_name = "wm8524-audio"
```

- Set `wm8524` as the default sink to listen to music on the audio jack of the i.MX 8M EVK board.

```
$ root@imx8mmevk:~# wpctl set-default 50
```

- Check the `wpctl` status again and verify that `wm8524` is set as `alsa_output`.

```
$ root@imx8mmevk:~# wpctl status
Settings
└─ Default Configured Devices:
         0. Audio/Sink    alsa_output.platform-sound-wm8524.stereo-fallback
         1. Audio/Source  alsa_input.platform-sound-bt-sco.mono-fallback
```

**Step 6** – Plug the audio jack of the i.MX 8M EVK board to the speaker/headset and play music from the Bluetooth A2DP source device connected to the board.

**Step 7** – Use `bluetooth-player` to play and pause the audio.

```
$ root@imx8mmevk:~# bluetooth-player
[NEW] Media /org/bluez/hci0
[bluetooSupportedUUIDs: 0000110a-0000-1000-8000-00805f9b34fb
[bluetooSupportedUUIDs: 0000110b-0000-1000-8000-00805f9b34fb
[NEW] Player /org/bluez/hci0/dev_48_74_12_C2_F2_82/player0 [default]
[NEW] Endpoint /org/bluez/hci0/dev_48_74_12_C2_F2_82/sep1
[NEW] Transport /org/bluez/hci0/dev_48_74_12_C2_F2_82/sep1/fd0
[bluetooth]# pause
Attempting to pause
Pause successful
[CHG] Transport /org/bluez/hci0/dev_48_74_12_C2_F2_82/sep1/fd0 State: idle
[CHG] Player /org/bluez/hci0/dev_48_74_12_C2_F2_82/player0 Status: paused
[CHG] Player /org/bluez/hci0/dev_48_74_12_C2_F2_82/player0 Position: 0x0000cbbb (52155)
[bluetooth]# play
Attempting to play
Play successful
[CHG] Transport /org/bluez/hci0/dev_48_74_12_C2_F2_82/sep1/fd0 State: pending
[CHG] Player /org/bluez/hci0/dev_48_74_12_C2_F2_82/player0 Status: playing
[CHG] Player /org/bluez/hci0/dev_48_74_12_C2_F2_82/player0 Position: 0x0000cccf (52431)
[CHG] Player /org/bluez/hci0/dev_48_74_12_C2_F2_82/player0 Track.Title: Takin' Back My
 Love
[CHG] Player /org/bluez/hci0/dev_48_74_12_C2_F2_82/player0 Track.TrackNumber: 0x00000000
 (0)
[CHG] Player /org/bluez/hci0/dev_48_74_12_C2_F2_82/player0 Track.NumberOfTracks:
 0x00000000 (0)
[CHG] Player /org/bluez/hci0/dev_48_74_12_C2_F2_82/player0 Track.Duration: 0x0003841a
 (230426)
[CHG] Player /org/bluez/hci0/dev_48_74_12_C2_F2_82/player0 Track.Album: Greatest Hits
[CHG] Player /org/bluez/hci0/dev_48_74_12_C2_F2_82/player0 Track.Artist: Enrique
 Iglesias, Ciara
[CHG] Player /org/bluez/hci0/dev_48_74_12_C2_F2_82/player0 Position: 0x0000ccf6 (52470)
[CHG] Transport /org/bluez/hci0/dev_48_74_12_C2_F2_82/sep1/fd0 State: active
[bluetooth]#
```

**Step 8** – Disconnect the device using bluetoothctl command line interface.

```
[test_phone]# disconnect 48:74:12:C2:F2:82
Attempting to disconnect from 48:74:12:C2:F2:82
[DEL] Player /org/bluez/hci0/dev_48_74_12_C2_F2_82/player0 [default]
[DEL] Transport /org/bluez/hci0/dev_48_74_12_C2_F2_82/sep1/fd1
[DEL] Endpoint /org/bluez/hci0/dev_48_74_12_C2_F2_82/sep1
hci0 48:74:12:C2:F2:82 type BR/EDR disconnected with reason 2
[CHG] Device 48:74:12:C2:F2:82 ServicesResolved: no
Successful disconnected 5G]#
[CHG] Device 48:74:12:C2:F2:82 Connected: no
[bluetooth]#
```

#### 5.2.1.2.2 Configuring A2DP source profile

Steps to configure i.MX 8M EVK board with A2DP source profile:

**Step 1** – Start PipeWire on i.MX 8M EVK board.

```
$ root@imx8mmevk:~# systemctl --user start pipewire wireplumber
$ root@imx8mmevk:~#
```

**Step 2** – Connect with a remote Bluetooth device that supports the Audio Sink Profile. Refer to Section "Scan, pair and connect to Bluetooth classic/Bluetooth LE"

**Step 3** – Verify the Audio Sink Profile capability of the connected device.

Command using the command line utility for Bluetooth operations (bluetoothctl):

```
$ root@imx8mmevk:~# bluetoothctl
$ [JBL Flip 5]# info
Device B8:F6:53:E8:BF:B7 (public)
      Name: JBL Flip 5
      Alias: JBL Flip 5
      Class: 0x00240414 (2360340)
      Icon: audio-card
      Paired: yes
      Bonded: yes
      Trusted: no
      Blocked: no
      Connected: yes
      LegacyPairing: no
      UUID: Serial Port               (00001101-0000-1000-8000-00805f9b34fb)
      UUID: Headset                   (00001108-0000-1000-8000-00805f9b34fb)
      UUID: Audio Sink                (0000110b-0000-1000-8000-00805f9b34fb)
      UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
      UUID: Advanced Audio Distribu.. (0000110d-0000-1000-8000-00805f9b34fb)
      UUID: A/V Remote Control        (0000110e-0000-1000-8000-00805f9b34fb)
      UUID: Handsfree                 (0000111e-0000-1000-8000-00805f9b34fb)
      [DEL] Device AC:C0:48:9F:82:5A Test
      [JBL Flip 5]#
```

**Step 5** – Check the available sink cards.

```
$ root@imx8mmevk:~# bluetoothctl
      $ [JBL Flip 5]# info
      Device B8:F6:53:E8:BF:B7 (public)
      Name: JBL Flip 5
      Alias: JBL Flip 5
      Class: 0x00240414 (2360340)
      Icon: audio-card
      Paired: yes
      Bonded: yes
      Trusted: no
      Blocked: no
      Connected: yes
      LegacyPairing: no
      UUID: Serial Port               (00001101-0000-1000-8000-00805f9b34fb)
      UUID: Headset                   (00001108-0000-1000-8000-00805f9b34fb)
      UUID: Audio Sink                (0000110b-0000-1000-8000-00805f9b34fb)
      UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
      UUID: Advanced Audio Distribu.. (0000110d-0000-1000-8000-00805f9b34fb)
      UUID: A/V Remote Control        (0000110e-0000-1000-8000-00805f9b34fb)
      UUID: Handsfree                 (0000111e-0000-1000-8000-00805f9b34fb)
      [DEL] Device AC:C0:48:9F:82:5A Test
      [JBL Flip 5]#
      Step 5 - Use the following command to check the available sink cards:
      root@imx8mmevk:~# wpctl status
      PipeWire 'pipewire-0' [1.0.5, root@imx8mmevk, cookie:815200939]
      └─ Clients:
      32. WirePlumber                         [1.0.5, root@imx8mmevk, pid:559]
      40. WirePlumber [export]                [1.0.5, root@imx8mmevk, pid:559]
      80. wpctl                               [1.0.5, root@imx8mmevk, pid:665]
      Audio
      ├─ Devices:
      │      41. Built-in Audio                       [alsa]
      │      42. Built-in Audio                       [alsa]
      │      43. Built-in Audio                       [alsa]
      │      44. Built-in Audio                       [alsa]
      │      74. JBL Flip 5                           [bluez5]
      │
      ├─ Sinks:
      │      46. Built-in Audio Mono                  [vol: 0.40]
      │      49. Built-in Audio Stereo                [vol: 0.40]
      │  *   50. Built-in Audio Stereo                [vol: 0.40]
      │      75. JBL Flip 5                           [vol: 0.16]
      │
      ├─ Sources:
      │  *   45. Built-in Audio Mono                  [vol: 1.00]
      │      47. Built-in Audio Stereo                [vol: 1.00]
      │      48. Built-in Audio Stereo                [vol: 1.00]
      │
      ├─ Filters:
      │
      └─ Streams:

      Video
      ├─ Devices:
      │      54. i.MX6S_CSI                           [v4l2]
      │      55. vsi_v4l2enc                          [v4l2]
      │      56. vsi_v4l2dec                          [v4l2]
      │
      ├─ Sinks:
      │
      ├─ Sources:
      │  *   59. i.MX6S_CSI (V4L2)
      │
      ├─ Filters:
      │
      └─ Streams:
      Settings
      └─ Default Configured Devices:
      0. Audio/Sink    alsa_output.platform-sound-wm8524.stereo-fallback
      1. Audio/Source  alsa_input.platform-sound-bt-sco.mono-fallback
root@imx8mmevk:~#
```

**Step 6** – Look for the connected sink device in the list of sink cards.

```
$ root@imx8mmevk:~# wpctl inspect 75
id 75, type PipeWire:Interface:Node
    api.bluez5.address = "B8:F6:53:E8:BF:B7"
    api.bluez5.codec = "sbc"
    api.bluez5.profile = "a2dp-sink"
    api.bluez5.transport = ""
    audio.adapt.follower = ""
    bluez5.loopback = "false"
    card.profile.device = "1"
    * client.id = "40"
    clock.quantum-limit = "8192"
    device.api = "bluez5"
    * device.id = "74"
    device.routes = "1"
    * factory.id = "11"
    factory.mode = "merge"
    factory.name = "api.bluez5.a2dp.sink"
    library.name = "audioconvert/libspa-audioconvert"
  * media.class = "Audio/Sink"
    media.name = "JBL Flip 5"
  * node.description = "JBL Flip 5"
  node.driver = "true"
  * node.name = "bluez_output.B8_F6_53_E8_BF_B7.1"
  node.pause-on-idle = "false"
  * object.serial = "114"
  * priority.driver = "1010"
  * priority.session = "1010"
root@imx8mmevk:~#
```

- Select the connected Bluetooth device as an audio sink device.

```
$ root@imx8mmevk:~# wpctl set-default 75
```

**Step 7** – Play the audio file using `pw-play` utility.

```
$ root@imx8mmevk:~# pw-play -v Makhna.wav &
[1] 715
root@imx8mmevk:~# sndfile: opened file "Makhna.wav" format 00010002 channels:2 rate:44100
sndfile: using default channel map: FL,FR
PCM: fmt:s16 rate:44100 channels:2 width:2
rate:44100 latency:4410 (0.100s)
connecting playback stream; target=(null)
stream state changed unconnected -> connecting
stream param change: Spa:Enum:ParamId:Latency
stream param change: Spa:Enum:ParamId:Tag
stream param change: Spa:Enum:ParamId:Props
stream properties:
        application.name = "pw-play"
        node.name = "pw-play"
        media.title = "Makhna (Drive)(bossmobi)"
        media.software = "Lavf60.3.100"
        media.artist = "wapking.fun"
        media.date = "2019"
        media.format = "WAV (Microsoft)"
        media.name = "'Makhna (Drive)(bossmobi)' / 'wapking.fun'"
        node.rate = "1/44100"
        node.latency = "4410/44100"
        media.type = "Audio"
        media.category = "Playback"
        media.role = "Music"
        media.filename = "Makhna.wav"
        stream.is-live = "true"
```

#### 5.2.1.3 Hands-free profile (HFP)

This section introduces the Ofono package used to dial, hang-up and answer telephone calls, and details the configuration for the hands-free profile.

##### 5.2.1.3.1 Ofono package

**Ofono package**

Ofono provides a mobile telephony application development framework with minimal and easy-to-use APIs. Ofono is controlled through D-Bus.

ofonod is the daemon that provides the Ofono stack for interfacing mobile telephony devices. For example, one can tell ofonod to send AT commands over */dev/rfcomm0* by calling the D-Bus method `org.ofono.at.Manager.Create`.

Table 8 describes the AT commands used to dial, hang-up and answer the calls:

**Table 8. Hands-free profile AT commands**

| command | Description |
|---|---|
| ATA | Command used to answer a call |
| ATD | Command used to place a call to a phone number |
| AT+CHUP | Command used to hang up. This command causes the AG to end an active call |

For more information on AT commands refer to the.

##### 5.2.1.3.2 Hands-free profile configuration

**Step 1** – Enable the host interface.

- **88W8987**: Use *imx8mq-evk-usdhc2-m2.dtb* file to enable SDIO on M.2 connector
- **88W8997**: Use *imx8mq-evk-pcie1-m2.dtb* file to enable PCIe on M.2 connector

***Note:*** *For details on interface enable, refer to ref.[18].*

**Step 2** – Stop PipeWire.

If PipeWire is running and needs to be restarted, stop it using:

```
systemctl --user status
```

Command output example:

```
pipewire
```

**Step 3** – Update the sample rate in the configuration file.

To change the default sample rate to 8000 samples per second, edit the configuration file:

```
nano /etc/pipewire/pipewire.conf
default.clock.rate = 8000
default.clock.quantum = 1024
```

**Step 4** – Start PipeWire on i.MX 8M Quad EVK.

```
systemctl --user start pipewire
```

**Step 5** – List the available sink devices.

```
pw-cli list-objects Node | grep 'id\|name'
```

Command output example:

```
  index: 0
      name: <alsa_output.platform-sound-bt-sco.mono-fallback>
  index: 1
      name: <alsa_output.platform-sound-spdif.stereo-fallback>
* index: 2
      name: <alsa_output.platform-sound-wm8524.stereo-fallback>
root@imx8mqevk:~#
```

**Step 6** – Set external sink device as default sink.

```
pw-cli set-default-sink <index number of external sink device>
```

***Note:*** *As the i.MX 8M Quad EVK does not have a built-in mic, an external USB audio mic adapter is required to enable the two-way audio.*

**Step 7** – List the available source devices.

```
pw-cli list-objects Node | grep 'id\|name'
```

Command output example:

```
index: 0
     name: <alsa_output.platform-sound-bt-sco.mono-fallback.monitor>
* index: 1
     name: <alsa_input.platform-sound-bt-sco.mono-fallback>
index: 2
     name: <alsa_input.platform-sound-hdmi-arc.stereo-fallback>
index: 3
     name: <alsa_output.platform-sound-hdmi.stereo-fallback.monitor>
index: 4
     name: <alsa_output.platform-sound-spdif.stereo-fallback.monitor>
index: 5
     name: <alsa_input.platform-sound-spdif.stereo-fallback>
index: 6
     name: <alsa_output.platform-sound-wm8524.stereo-fallback.monitor>

root@imx8mqevk:~#
```

**Step 8** – Set the external source device as default source.

```
pw-cli set-default-source <index number of external source device>
```

**Step 9** – Enable the audio route.

```
pw-link alsa_input.platform-sound-bt-sco.mono-fallback <External sink device>

pw-link <External source device> alsa_output.platform-sound-bt-sco.mono-fallback
```

**Step 10** – Enable PCM line management by the host software.

The following command controls the PCM lines. This command should be sent when the host wants to control the PCM lines.

The host issues this command to have multiple voices at the same time, and inform the controller that the host software provides the PCM configuration.

```
hcitool -i hci0 cmd 0x3f 0x0070 0x01
```

**Table 9. Command parameters**

| Parameter | Length | Definition |
|---|---|---|
| ogf | 1 | 0x3F |
| ocf | 2 | 0x0070 |
| action | 1 | 0x00 = the controller manages the PCM lines(default)<br>0x01 = the host software manages the PCM lines (enables new use cases) |

Command output example:

```
< HCI Command: ogf 0x3f, ocf 0x0070, plen 1
  01
> HCI Event: 0x0e plen 4
  01 70 FC 00
```

**Step 11** – Connect with a remote Bluetooth device.

Refer to Scan, pair and connect to Bluetooth classic/Bluetooth LE and connect with a remote Bluetooth device that supports HFP Audio Gateway.

**Step 12** – Verify the Hands-free Audio Gateway Profile capability of the connected Bluetooth device.

```
bluetoothctl
[GT-S7560M]# info 04:1B:BA:C7:92:36
```

Output command example:

```
Device 04:1B:BA:C7:92:36 (public)

    Name: GT-S7560M
    Alias: GT-S7560M
    Class: 0x005a020c
    Icon: phone
    Paired: yes
    Trusted: yes
    Blocked: no
    Connected: yes
    LegacyPairing: no
    UUID: OBEX Object Push          (00001105-0000-1000-8000-00805f9b34fb)
    UUID: Audio Source              (0000110a-0000-1000-8000-00805f9b34fb)
    UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
    UUID: Headset AG                (00001112-0000-1000-8000-00805f9b34fb)
    UUID: NAP                       (00001116-0000-1000-8000-00805f9b34fb)
    UUID: Handsfree Audio Gateway   (0000111f-0000-1000-8000-00805f9b34fb)
    UUID: Phonebook Access Server   (0000112f-0000-1000-8000-00805f9b34fb)
    UUID: Message Access Server     (00001132-0000-1000-8000-00805f9b34fb)
    UUID: PnP Information           (00001200-0000-1000-8000-00805f9b34fb)
    Modalias: bluetooth:v0075p0100d0100
[GT-S7560M]#
```

**Step 13** – Set SCO voice data path through PCM interface.

```
hcitool -i hci0 cmd 0x3F 0x001D 0x01
```

**Table 10. Command parameters**

| Parameter | Length | Definition |
|---|---|---|
| OGF | 1 | 0x3F |
| OCF | 2 | 0x001D |
| Voice path | 1 | 0x00 = Host/DMA<br>0x01 = PCM |

Command output example:

```
< HCI Command: ogf 0x3f, ocf 0x001d, plen 1
  01
> HCI Event: 0x0e plen 4
  01 1D FC 00
root@imx8mqevk:~#
```

***Note:*** *Host voice path is not a supported feature with i.MX 8M Quad.*

**Step 14** – Write PCM settings.

```
hcitool -i hci0 cmd 0x3F 0x0007 0x00
```

**Table 11. Command parameters**

| Parameter | Length | Definition |
|---|---|---|
| OGF | 1 | 0x3F |
| OCF | 2 | 0x0007 |
| PCM_Setting | 1 | Bit[4]: PCM clock ON<br>• 0 = PCM clock is terminated after last data bit has been transmitted<br>• 1 = make PCM clock available continuously<br>Bit[3]: Reserved<br>Bit[2]: PCM Sync source<br>• 0 = PCM sync page generated from system clock<br>• 1 = PCM sync page generated from frame clock<br>Bit[1]: Central/peripheral<br>• 0 = PCM I/F peripheral, external PCM clock and synchronization<br>• 1 = PCM I/F central, internal PCM clock and synchronization<br>Bit[0]: PCM direction<br>• 0 = port A receives, port B transmits<br>• 1 = port A transmits, port B receives |

Command output example:

```
< HCI Command: ogf 0x3f, ocf 0x0007, plen 1
  00
> HCI Event: 0x0e plen 4
  01 07 FC 00
root@imx8mqevk:~#
```

**Step 15** – Write PCM sync settings

```
hcitool -i hci0 cmd 0x3F 0x0028 0x03 0x00 0x03
```

**Table 12. Command parameters**

| Parameter | Length | Definition |
|---|---|---|
| OGF | 1 | 0x3F |
| OCF | 2 | 0x0028 |
| PCM Sync Settings 1 | 1 | Default = 0x03<br>ISR (IramSyncRate) only valid if IF = Host:<br>PCM Sync Settings 1 Bit[0]:<br>• 0 = bursts controlled by Tx or Rx of voice packets<br>• 1 = fixed rate of 8 ksample/s<br>ISS (IramSyncSource) only valid if IF = Host and ISR = Fixed Rate:<br>PCM Sync Settings 1 Bit[1]:<br>• 0 = 0 = ISR not aligned to frame tick<br>• 1 = ISR aligned to frame tick (this field should be set to 1) |
| PCM Sync Settings 2 | 2 | Default = 0x0000<br>pcmIfMode in PCM Descriptor<br>PCM Sync Settings 2 Bits[1:0]:<br>• 00 = PCM short sync<br>• 01 = PCM long sync<br>• 10 = I2S audio mode<br>pcmLRCPol in PCM Descriptor<br>PCM Sync Settings 2 Bit[4]:<br>• 0 = LRC is same polarity as PCM sync<br>• 1 = LRC is inverted<br>pcmMClkEn in PCM Descriptor<br>PCM Sync Settings 2 Bit[8]:<br>• 0 = disable generation of PCM main clock<br>• 1 = enable<br>pcm2048MClkSel in PCM Descriptor<br>PCM Sync Settings 2 Bit[9]:<br>• 0 = default<br>• 1 = select 2.048 MHz clock for PCM clock<br>16k Sync in PCM<br>PCM Sync Settings 2 Bit[10]:<br>• 0 = 8k Sync<br>• 1 = 16k Sync |

Command output example:

```
< HCI Command: ogf 0x3f, ocf 0x0028, plen 3
   03 00 03
> HCI Event: 0x0e plen 4
   01 28 FC 00
root@imx8mqevk:~#
```

**Step 16** – Write PCM Link settings

```
hcitool -i hci0 cmd 0x3F 0x0029 0x04 0x00
```

**Table 13. Command parameters**

| Parameter | Length | Definition |
|---|---|---|
| OGF | 1 | 0x3F |
| OCF | 2 | 0x0029 |
| PCM_Link_Setting | 2 | Bits [13:10]:<br>Each bit corresponds to 1 of 4 PCM timeslots (if 0, the slot is used by the BTU)<br>Bits[9:2]:<br>Defines start of PCM slot relative to start of PCM synchronization (must be greater than the size of the PCM slot)<br>Bits[1:0]:<br>Indicates which PCM slots should be used. Default:<br>• 0x0004 = first SCO link<br>• 0x0045 = second SCO link<br>• 0x0086 = third SCO link |

***Note:*** *The PCM Link settings command should be given after HCI reset and before setting up the voice link. Also, if multiple voice links are supported, this command should be given before setting up the voice link with the respective parameters of each voice link.*

Command output example:

```
< HCI Command: ogf 0x3f, ocf 0x0029, plen 2
   04 00
> HCI Event: 0x0e plen 4
   01 29 FC 00
root@imx8mqevk:~#
```

**Step 17** – Write the voice settings.

To write the required voice settings, use `HCI_Write_Voice_Setting` command defined in ref.[12].

```
hcitool -i hci0 cmd 0x03 0x0026 0x60 0x00
```

Command output example:

```
< HCI Command: ogf 0x03, ocf 0x0026, plen 2
   60 00
> HCI Event: 0x0e plen 4
   01 26 0C 00
root@imx8mqevk:~#
```

**Step 18** – Run Ofono test scripts.

Go to the repository where Ofono test scripts are available:

```
cd /usr/lib/ofono/test/
```

Run the dial number test script

```
root@imx8mqevk:/usr/lib/ofono/test# python3 ./dial-number 1234567890
```

Command output example:

```
Using modem /hfp/org/bluez/hci0/dev_04_1B_BA_C7_92_36
/hfp/org/bluez/hci0/dev_04_1B_BA_C7_92_36/voicecall01
root@imx8mqevk:~/test#
```

**Step 19** – Initialize the PCM interface.

The command below initializes and configures PCM. This command should be sent in any of the following situations:

- To initialize PCM after starting voice call on a particular SCO connection
- To switch call from SCO connection 1 to SCO connection 2
- To route SCO connection 1 voice data to SCO connection 2
- To de-initialize PCM once the voice call is over on a particular SCO connection

To initialize the PCM interface:

```
root@imx8mqevk:/usr/lib/ofono/test# hcitool -i hci0 cmd 0x3f 0x006f 0x00 0x00 0x08 0x00
 0x00 0x00
```

Command output example:

```
< HCI Command: ogf 0x3f, ocf 0x006f, plen 6
   00 00 08 00 00 00
> HCI Event: 0x0e plen 4
   01 6F FC 00
```

Command to de-initialize the PCM interface

```
root@imx8mqevk:/usr/lib/ofono/test# hcitool -i hci0 cmd 0x3f 0x006f 0x01 0x00 0x08 0x00
 0x00 0x00
```

**Table 14. Command parameters**

| Parameter | Length | Definition |
|---|---|---|
| OGF | 1 | 0x3F |
| OCF | 2 | 0x006F |
| Action | 1 | 0x00 = PCM will be initialized<br>0x01 = PCM will be de-initialized |
| Operation mode | 1 | 0x00 = normal mode<br>This mode is used when only 1 voice call needs to be active at a time. Depending on the Action parameter, either the PCM will be initialized or de-initialized.<br>0x01 = internal loopback<br>This mode is used when there are 2 HF connections and audio data on first connection needs to be routed to second connection. This can be used only when both SCO links are of the same type (that is, NB to NB, or WB to WB only).<br>0x02 = remote loopback on same link<br>0x03 = local loopback on same link<br>0x04 to 0xFF = reserved |
| SCO Handle1 | 2 | Synchronous connection handle for which PCM configuration is to be done<br>Range from 8 to 10 |
| SCO Handle2 | 2 | Synchronous connection handle for which PCM configuration is to be done<br>Parameter valid only when Operation Mode = 0x01. |

Command output example:

```
< HCI Command: ogf 0x3f, ocf 0x006f, plen 6
   01 00 08 00 00 00
> HCI Event: 0x0e plen 4
   01 6F FC 00
```

**Step 20** – Use Ofono test script to hang up a call.

Ofono hangup-all script uses the AT command `AT+CHUP` to hangup a call.

```
root@imx8mqevk:/usr/lib/ofono/test# python3 ./hangup-all
root@imx8mqevk:/usr/lib/ofono/test#
```

**Step 21** – Use Ofono test script to answer a call.

Ofono answer-calls script uses the AT command `ATA` to answer a call.

```
root@imx8mqevk:/usr/lib/ofono/test#  python3 ./answer-calls
```

Command output example:

```
[ /hfp/org/bluez/hci0/dev_04_1B_BA_C7_92_36 ]
[ /hfp/org/bluez/hci0/dev_04_1B_BA_C7_92_36/voicecall01 ] incoming
root@imx8mqevk:/usr/lib/ofono/test#
```

**Step 22** – Initialize PCM.

Refer to Step 19.

**Step 23** – Use Ofono test script to hang up an active call.

Ofono hangup-active script uses the AT command `AT+CHUP` to hang up an active call

```
root@imx8mqevk:/usr/lib/ofono/test# python3 ./hangup-active
[ /hfp/org/bluez/hci0/dev_04_1B_BA_C7_92_36/voicecall01 ] active
root@imx8mqevk:/usr/lib/ofono/test#
```

#### 5.2.1.4 Object Push Profile

This section explains how to send the file to a remote device over Bluetooth. BlueZ Stack provides the obexctl user space utility to send the files to a Bluetooth device.

Use the following steps to configure Object Push Profile using OBEX Control utility (obexctl)

**Start the OBEX daemon on i.MX 8M Quad**

```
/usr/libexec/bluetooth/obexd &
    [1] 768
root@imx8mqevk:~#
```

**Connect with a remote Bluetooth device that supports Object Push Profile**

Refer to Scan, pair and connect to Bluetooth classic/Bluetooth LE .

Run the following commands to verify the Object Push Profile capability of the connected Bluetooth device.

```
bluetoothctl
    Agent registered
    [Moto E]# info B4:F5:00:31:CB:4E
    Device B4:F5:00:31:CB:4E (public)
            Name: Moto E
            Alias: Moto E
            Class: 0x005a020c
            Icon: phone
            Paired: yes
            Trusted: yes
            Blocked: no
            Connected: yes
    LegacyPairing: no
            UUID: OBEX Object Push          (00001105-0000-1000-8000-00805f9b34fb)
            UUID: Audio Source              (0000110a-0000-1000-8000-00805f9b34fb)
            UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
            UUID: A/V Remote Control        (0000110e-0000-1000-8000-00805f9b34fb)
            UUID: Headset AG                (00001112-0000-1000-8000-00805f9b34fb)
            UUID: PANU                      (00001115-0000-1000-8000-00805f9b34fb)
            UUID: NAP                       (00001116-0000-1000-8000-00805f9b34fb)
            UUID: Handsfree Audio Gateway   (0000111f-0000-1000-8000-00805f9b34fb)
            UUID: Phonebook Access Server   (0000112f-0000-1000-8000-00805f9b34fb)
            UUID: Message Access Server     (00001132-0000-1000-8000-00805f9b34fb)
            UUID: PnP Information           (00001200-0000-1000-8000-00805f9b34fb)
            UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
            UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
    Modalias: bluetooth:v000Fp1200d1436
    [Moto E]#
```

### Start OBEX Control

Use the following commands to start the OBEX Control and connect to the remote Bluetooth device paired using Bluetooth Control.

```
obexctl
    [NEW] Client /org/bluez/obex
    [obex]# connect B4:F5:00:31:CB:4E
    Attempting to connect to B4:F5:00:31:CB:4E
    [NEW] Session /org/bluez/obex/client/session0 [default]
    [NEW] ObjectPush /org/bluez/obex/client/session0
    Connection successful
    [B4:F5:00:31:CB:4E]#
```

Send a file from i.MX 8M Quad to the connected remote Bluetooth device

```
[B4:F5:00:31:CB:4E]# send /home/root/sample_audio.wav
    Attempting to send /home/root/sample_audio.wav to /org/bluez/obex/client/session0
    [NEW] Transfer /org/bluez/obex/client/session0/transfer0
    Transfer /org/bluez/obex/client/session0/transfer0
            Status: queued
            Name: sample_audio.wav
            Size: 1073218
            Filename: /home/root/sample_audio.wav
    Session: /org/bluez/obex/client/session0
    [CHG] Transfer /org/bluez/obex/client/session0/transfer0 Status: active
    [CHG] Transfer /org/bluez/obex/client/session0/transfer0 Transferred: 8030 (@8KB/s
 02:12)
    [B4:F5:00:31:CB:4E]#
```

### Select ACCEPT or DECLINE on the remote Bluetooth device

Select ACCEPT or DECLINE on the remote Bluetooth device depending on the file to be received. The following logs appear once the file transfer is completed.

```
[CHG] Transfer /org/bluez/obex/client/session1/transfer1 Status: complete
    [DEL] Transfer /org/bluez/obex/client/session1/transfer1
    [B4:F5:00:31:CB:4E]#
```

### Disconnect and exit from the obexctl

```
[B4:F5:00:31:CB:4E]# disconnect
    Attempting to disconnect to /org/bluez/obex/client/session2
    [DEL] Session /org/bluez/obex/client/session2 [default]
    [DEL] ObjectPush /org/bluez/obex/client/session2
    Disconnection successful
    [obex]# quit
```

***Note:*** *The OBEX Control receiving is not working for some users with the current version 5.65 of BlueZ. It may be resolved in a later version.*

#### 5.2.1.5 Human Interface Device Profile

This section provides the configuration steps to connect a Bluetooth keyboard and/or mouse. The Human Interface Device Profile is used to establish the connection.

Use the following steps to configure Human Interface Device (HID) Profile to connect with the Bluetooth device.

**Connect with a remote Bluetooth device that supports Human Interface Device profile**

Refer to Scan, pair and connect to Bluetooth classic/Bluetooth LE .

**Verify the Human Interface Device Profile capability of the connected Bluetooth device**

```
[Rapoo E6700]# info 6C:5D:63:20:40:AA
        Device 6C:5D:63:20:40:AA
        Name: Rapoo E6700
        Alias: Rapoo E6700
        Class: 0x002540
        Icon: input-keyboard
        Paired: yes
        Trusted: yes
        Blocked: no
        Connected: yes
        LegacyPairing: no
        UUID: Service Discovery Server (00001000-0000-1000-8000-00805f9b34fb)
        UUID: Human Interface Device    (00001124-0000-1000-8000-00805f9b34fb)
        UUID: PnP Information           (00001200-0000-1000-8000-00805f9b34fb)
        Modalias: usb:v0A5Cp8502d011B
        [Rapoo E6700]#
```

The connected keyboard can be used as a normal keyboard now.

***Note:*** *To enable HID profile, enable the user-space I/O driver support for HID subsystem in i.MX Linux BSP source code.*

#### 5.2.1.6 Serial port profile (SPP)

The serial port profile (SPP) feature is used to set emulated serial cable connections using an RFCOMM connection between two peer devices.

This section describes the procedure to verify the serial port profile feature. It shows how to send and receive a message string to and from the remote device over Bluetooth connection using SPP.

To validate SPP feature with the i.MX host platform, you need an Android mobile phone and SPP Android application *Bluespp*. *BlueSPP* application is available for download on Google play store.

##### 5.2.1.6.1 Enable serial port profile

Steps to enable SPP support on i.MX 8M Quad EVK running with Linux OS:

**Step 1** - Cancel previous *bluetoothd* services.

```
killall bluetoothd
```

**Step 2** - Start *bluetoothd* with`–compat` option.

```
/usr/lib/bluetooth/bluetoothd --compat &
```

**Step 3** - Bring up the HCI interface

Example of command:

```
hciconfig hci0 up
```

**Step 4** - Check which profiles or services are supported by default.

Example of command:

```
sdptool browse local | grep -i name
```

Example of command output:

```
Service Name: Generic Access Profile
Service Name: Generic Attribute Profile
Service Name: Device Information
Service Name: AVRCP CT
Service Name: AVRCP TG
Service Name: hfp_hf
```

***Note:*** *SPP is not supported by default.*

**Step 5** - Add the channel for serial port profile

```
sdptool add --channel=22 SP
```

Command output example:

```
Serial Port service registered
```

**Step 6** - Start `rfcomm` to watch a specific port.

```
rfcomm watch /dev/rfcomm0 22 > /dev/null &
```

Command output example:

```
Serial Port service registered
```

**Step 7** - Check that the serial port profile is enabled.

Example of command:

```
sdptool browse local | grep -i name
```

Example of command output:

```
Service Name: Generic Access Profile
Service Name: Generic Attribute Profile
Service Name: Device Information
Service Name: AVRCP CT
Service Name: AVRCP TG
Service Name: hfp_hf
Service Name: Serial Port
```

#### 5.2.1.6.2 Verify serial port profile

Steps to verify SPP profile on i.MX 8M Quad EVK running with Linux OS after you have enabled SPP ( Enable serial port profile ).

**Step 1** - Check if SPP is enabled.

```
sdptool browse local | grep -i name
```

Example of command output:

```
Service Name: Generic Access Profile
Service Name: Generic Attribute Profile
Service Name: Device Information
Service Name: AVRCP CT
Service Name: AVRCP TG
Service Name: hfp_hf
Service Name: Serial Port
```

**Step 2** - Add the channel for serial port profile.

```
sdptool add --channel=22 SP
```

Example of command output:

```
Serial Port service registered
```

**Step 3** - Start *rfcomm* to watch the connection on a specific port.

```
rfcomm watch /dev/rfcomm0 22 > /dev/null &
```

**Step 4** - Pair the DUT with a mobile phone.

Follow the steps in Scan, pair and connect to Bluetooth classic/Bluetooth LE .

**Step 5** - Establish the serial port profile connection.

- Open *Bluespp* application on an Android phone.
- Select *imx8mqevk* platform device from the list of devices (Figure 39).
- Click *Connect* to initiate the connection (Figure 40).

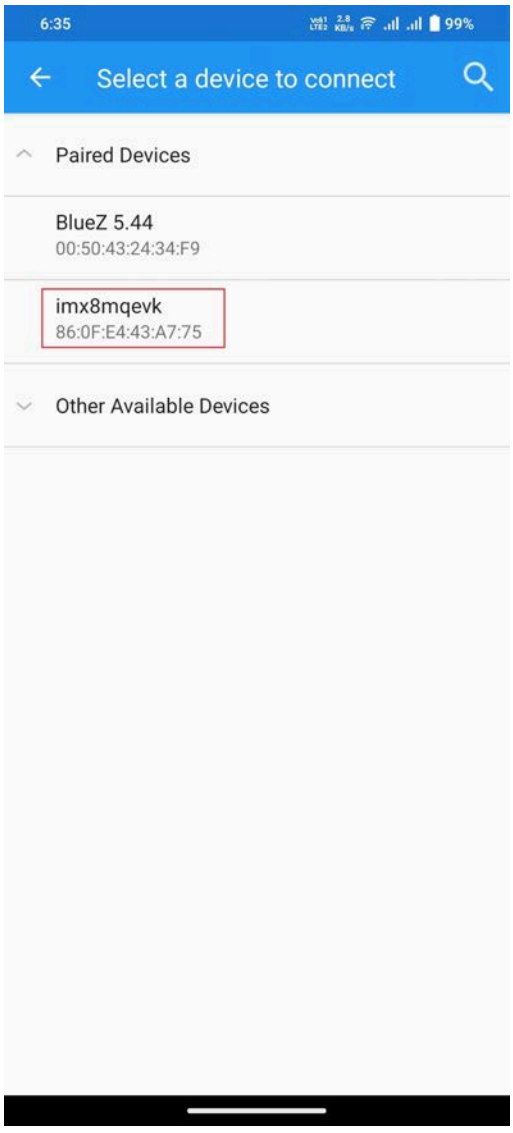



**Figure 39. Select the device to connect**

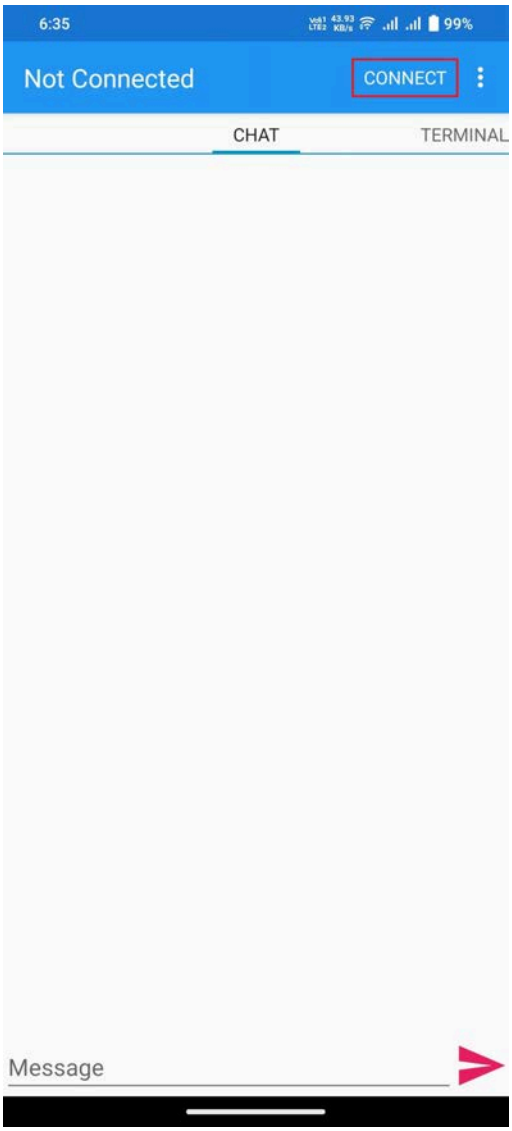



**Figure 40. Click CONNECT**

**Step 6** - Get confirmation that SPP connection is established.

When the connection is successful, *Connected to imx8mqevk* pops up on the mobile application, and *rfcomm0* file is generated in */dev/* directory (Figure 41).

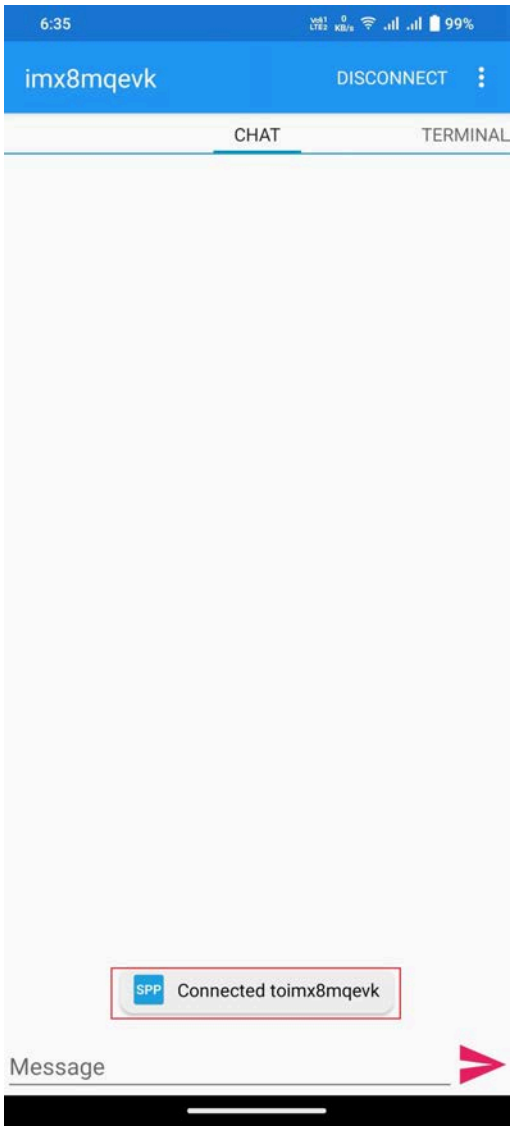



**Figure 41. Confirmation of SPP connection**

**Step 7** - Send a message from the DUT to the mobile phone application.

```
echo “hello from NXP!” > /dev/rfcomm0
```

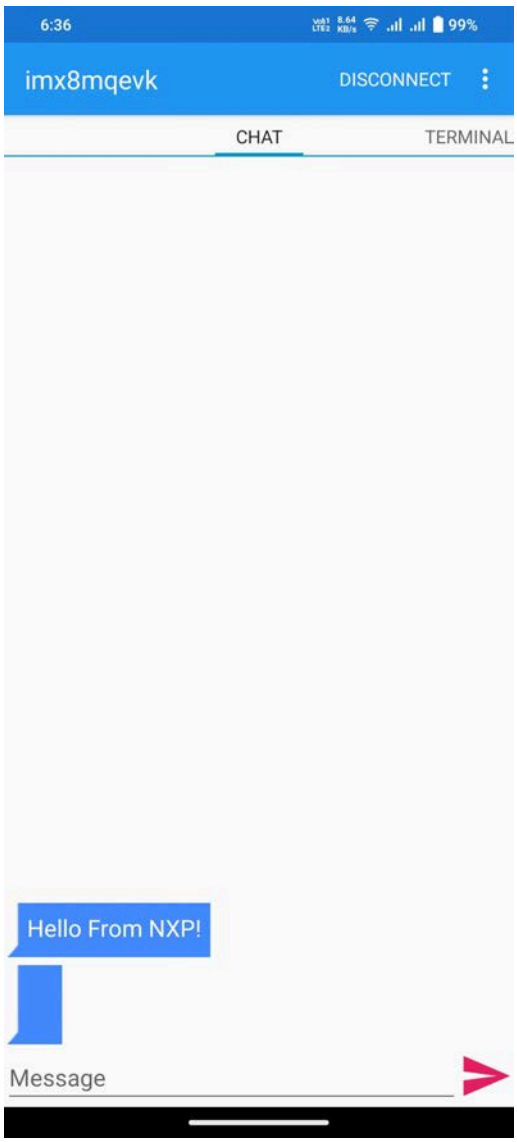



**Figure 42. Sending a message from the DUT**

**Step 8** - Send a message from the mobile phone application to the DUT.

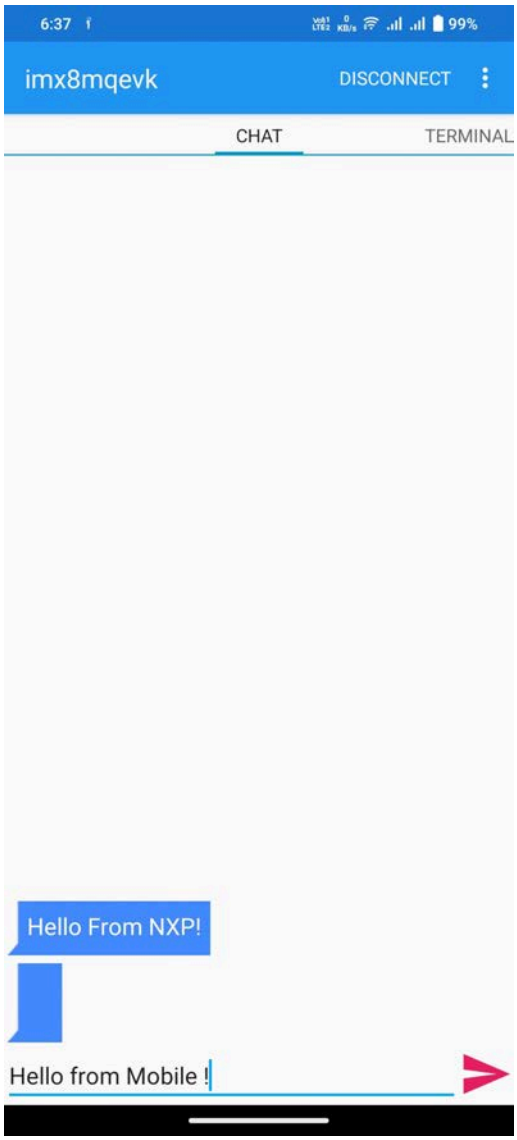



**Figure 43. Send a message to the DUT**

Example of command to read the message:

```
cat /dev/rfcomm0
```

Example of command output:

```
Hello from Mobile
```

### 5.2.2 Bluetooth LE

#### 5.2.2.1 Bluetooth LE device as GATT server

This section shows how to configure the i.MX 8M Quad board as a Bluetooth LE GATT Server using the example of Battery Service registration.

Use the following steps to configure the Battery Service for LE GATT server. To configure more services refer to ref.[13].

**Start *bluetoothctl* to interact with the Bluetooth daemon from the command line**

```
bluetoothctl
    Agent registered
    [bluetooth]#
```

**Register the Battery Service**

```
[bluetooth]# menu gatt
    Menu gatt:
    [bluetooth]# register-service 0x180F
    [NEW] Primary Service
            /org/bluez/app/service0xbf29850
            0x180F
            Battery Service
    [/org/bluez/app/service0xbf29850] Primary (yes/no): yes
[bluetooth]#
```

**Configure the characteristics for Battery Service**

```
[bluetooth]# register-characteristic 0x2A19 notify,read
    [NEW] Characteristic
            /org/bluez/app/service0xbf29850/chrc0xbf32890
            0x2A19
            Battery Level
    [/org/bluez/app/service0xbf29850/chrc0xbf32890] Enter value: 0x64
[bluetooth]#
```

### Register the Battery Service

```
[bluetooth]# register-application 0000180F-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 00001112-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 0000110a-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 00001200-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 0000110e-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 0000110b-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 0000110c-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 00001800-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 00001801-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 0000180f-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 0000111e-0000-1000-8000-00805f9b34fb
    Application registered
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 00001112-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 0000110a-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 00001200-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 0000110e-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 0000110b-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 0000110c-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 00001800-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 00001801-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 0000180f-0000-1000-8000-00805f9b34fb
    [CHG] Controller 00:50:43:24:34:F7 UUIDs: 0000111e-0000-1000-8000-00805f9b34fb
    [bluetooth]#
```

### Check that the Battery Service was added successfully

```
[bluetooth]# back
    Menu main:
[bluetooth]# show
    Controller 00:50:43:24:34:F7 (public)
            Name: imx8mqevk
            Alias: imx8mqevk
            Class: 0x002c0000
            Powered: yes
            Discoverable: no
    DiscoverableTimeout: 0x000000b4
            Pairable: yes
            UUID: Headset AG                (00001112-0000-1000-8000-00805f9b34fb)
            UUID: Audio Source              (0000110a-0000-1000-8000-00805f9b34fb)
            UUID: PnP Information           (00001200-0000-1000-8000-00805f9b34fb)
            UUID: A/V Remote Control        (0000110e-0000-1000-8000-00805f9b34fb)
            UUID: Audio Sink                (0000110b-0000-1000-8000-00805f9b34fb)
            UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
            UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
            UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
            UUID: Battery Service           (0000180f-0000-1000-8000-00805f9b34fb)
            UUID: Handsfree                 (0000111e-0000-1000-8000-00805f9b34fb)
    Modalias: usb:v1D6Bp0246d0532
            Discovering: no
            Roles: central
            Roles: peripheral
    Advertising Features:
            ActiveInstances: 0x00 (0)
            SupportedInstances: 0x04 (4)
            SupportedIncludes: tx-power
            SupportedIncludes: appearance
            SupportedIncludes: local-name
            SupportedSecondaryChannels: 1M
            SupportedSecondaryChannels: 2M
            SupportedSecondaryChannels: Coded
[bluetooth]#
```

### Start Bluetooth LE advertising

```
[bluetooth]# advertise on
    [CHG] Controller 00:50:43:24:34:F7 SupportedInstances: 0x04
    [CHG] Controller 00:50:43:24:34:F7 ActiveInstances: 0x01
    Advertising object registered
    Tx Power: off
    Name: off
    Apperance: off
    Discoverable: off
[bluetooth]#
```

### Connect to the i.MX host platform using the application "nRF connect LE Cell Phone" (ref.[8])

Check for the available Battery Service in the application.

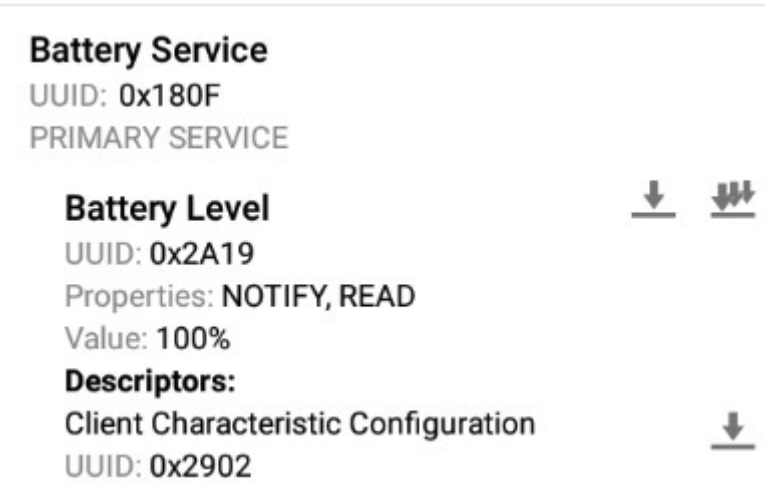



**Figure 44. Battery Service**

Read operation can be performed using the options from the nRF connect application. The following logs will be displayed on the i.MX 8M Quad console:

```
ReadValue: 41:CB:E2:8F:BA:62 offset 0 link LE
```

#### 5.2.2.2 Bluetooth LE device as GATT client

This section describes the procedure to configure the i.MX 8M Quad as a Bluetooth LE client. It also shows the options to list the attributes available in the Bluetooth LE server and access information.

Follow these steps to configure the Bluetooth LE GATT client.

**Connect with a remote Bluetooth LE device**

Refer to Scan, pair and connect to Bluetooth classic/Bluetooth LE.

**List the attributes**

```
[LE Device]# menu gatt
    Menu gatt
    [LE Device]# list-attributes 5F:82:1E:F1:DE:CC
Primary Service
    /org/bluez/hci0/dev_5F_82_1E_F1_DE_CC/service0001
    00001801-0000-1000-8000-00805f9b34fb
    Generic Attribute Profile
Characteristic
/org/bluez/hci0/dev_5F_82_1E_F1_DE_CC/service0001/char0002
    00002a05-0000-1000-8000-00805f9b34fb
    Service Changed
[LE Device]#
```

**Retrieve the attribute information**

Issue the command to retrieve the attribute information

```
[LE Device]# attribute-info /org/bluez/hci0/dev_5F_82_1E_F1_DE_CC/service0001/char0002
```

Command output example:

```
Characteristic - Service Changed
UUID: 00002a05-0000-1000-8000-00805f9b34fb
Service: /org/bluez/hci0/dev_5F_82_1E_F1_DE_CC/service0001
Notifying: no
Flags: indicate
[LE Device]#
```

### 5.2.3 Bluetooth power configurations

This section describes the support for Bluetooth and Bluetooth LE power configuration with vendor specific commands, and with examples using calibration data. The following considerations apply to power configuration:

- At power up, the initial power configuration of the device is based on the calibration data stored in the OTP.
- During runtime, the Bluetooth calibration data can be updated using vendor-specific commands (VSC).
- The max TX power can be updated using a vendor-specific command and the value can be reflected only in the range of the power class set in the calibration data.
- The values of the set and measured transmit power can be different due to various factors, including environmental conditions and interference.

Table 15 shows Bluetooth and Bluetooth LE power class configurations.

**Table 15. Power class and max power levels**

| Type | Max power level | Designed operating range |
|---|---|---|
| Class 1 | 20 dBm | Up to 100 m (328 feet) |
| Class 1.5 | 12 dBm | Up to 30 m (100 feet) |
| Class 2 | 4 dBm | Up to 10 m (33 feet) |

### 5.2.4 Human Interface Device Service

This section describes the configuration steps to connect the NXP-based Bluetooth LE module with a Bluetooth Low Energy device that supports Human Interface Device Profile.

Follow these steps to configure Human Interface Device Service.

**Connect with the Bluetooth LE device**

Refer to Scan, pair and connect to Bluetooth classic/Bluetooth LE.

**Verify the Human Interface Device Service capability of the connected Bluetooth LE device**

```
bluetoothctl
Agent registered
[LE Device]# info D4:18:20:A0:48:5F
Device D4:18:20:A0:48:5F (public)
    Name: LE Device
    Alias: LE Device
    Paired: yes
    Trusted: yes
    Blocked: no
    Connected: yes
    LegacyPairing: no
    UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
    UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
    UUID: Device Information        (0000180a-0000-1000-8000-00805f9b34fb)
    UUID: Battery Service           (0000180f-0000-1000-8000-00805f9b34fb)
    UUID: Human Interface Device    (00001812-0000-1000-8000-00805f9b34fb)
    UUID: Vendor specific           (3dda0001-957f-7d4a-34a6-74696673696d)
    ManufacturerData Key: 0x00df
    ManufacturerData Value:
57 30 46 39 30 32 31 39 59 32                    W0F90219Y2
[LE Device]#
```

The connected keyboard can be used as a normal keyboard now.

***Note:*** *To enable HID profile, enable the user-space I/O driver support for HID subsystem in i.MX Linux BSP source code.*

### 5.2.5 Wake-on Bluetooth/Bluetooth LE

Refer to ref.[4].

### 5.2.6 Bluetooth audio framework

- Start Pipewire on the i.MX host platform.

```
systemctl --user start pipewire wireplumber
```

- Get Pipewire version.

```
 pipewire -version
```

Command output example:

```
pipewire
Compiled with libpipewire 1.2.0
Linked with libpipewire 1.2.0
```

- Check Pipewire/Wireplumber status.

```
wpctl status
```

- Select the default sink/source for sound.

```
wpctl set-default <node id from wpctl status>
```

- Play an audio sample from Pipewire default player.

```
pw-play <--target to specify node> <audio_file.wav>
```

- For advanced options, refer to ref.[34].

#### 5.2.6.1 Classic audio

- Modify */usr/share/wireplumber/wireplumber.conf.d/51-bluez-imx.conf*.

***Note:*** *Look for a similar file if the above path does not match.*

Modification for A2DP source:

```
monitor.bluez.properties = {
  ## These features do not work on all headsets, so they are enabled
.......
  bluez5.hfphsp-backend = "ofono"
  bluez5.roles = <a2dp_source>
}
```

Modification for A2DP sink:

```
monitor.bluez.properties = {
  ## These features do not work on all headsets, so they are enabled
.......
  bluez5.hfphsp-backend = "ofono"
  bluez5.roles = <a2dp_sink>
}
```

- Check that the remote Bluetooth device supports the A2DP profile feature.
- Establish the Bluetooth connection between two devices. Refer to Section 5.2.2.1.
  The corresponding Node pops up in *wpctl status* under *Sinks/Sources*.
- Select the node with remote Bluez address and start streaming the music.

## 5.3 802.15.4

### 5.3.1 OpenThread

#### 5.3.1.1 Create and join a Thread network

This section describes the steps and configurations to create a thread network and join the thread network.

***Note:*** *For details on 802.15.4 interface, refer to section Bring-up of 802.15.4 interface in ref.[18].*

**Step 1** – Create a Thread Network

*ot-ctl* utility commands are used to configure the Thread network

- Stop the existing thread networks

```
root@imx8mmevk:/usr/sbin# ot-ctl thread stop
Done
root@imx8mmevk:/usr/sbin# ot-ctl ifconfig down
Done
```

- Apply a factory reset on 802.15.4 radio

```
root@imx8mmevk:/usr/sbin# ot-ctl factoryreset
```

- Set the channel of operation

```
root@imx8mmevk:/usr/sbin# ot-ctl channel 26
Done
```

- Set the panid and the extpanid for the network

```
root@imx8mmevk:/usr/sbin# ot-ctl panid 0x1234
Done
root@imx8mmevk:/usr/sbin# ot-ctl extpanid dead00beef00cafe
Done
```

- Set networkkey for the network

```
root@imx8mmevk:/usr/sbin# ot-ctl networkkey
```

Command output example:

```
00112233445566778899aabbccddeeff
Done
```

- Set the name for the network

```
root@imx8mmevk:/usr/sbin# ot-ctl networkname ThreadTest
Done
```

- Start the Thread network

```
root@imx8mmevk:/usr/sbin# ot-ctl ifconfig up
Done
root@imx8mmevk:/usr/sbin# ot-ctl thread start
Done
```

- Verify that the state of the device is leader

```
root@imx8mmevk:/usr/sbin# ot-ctl state
```

Command output example:

```
leader
Done
```

**Step 2** – Join network from the end device

Follow the steps listed below on the end device that will join the nThread network created in Step 1.

***Note:*** *The end device can be a Full Thread Device (FTD) or Minimal Thread Device (MTD).*

*ot-ctl* utility/commands are used to join the Thread network.

- Stop the existing thread networks

```
root@imx8mmevk:/usr/sbin# ot-ctl thread stop
Done
root@imx8mmevk:/usr/sbin# ot-ctl ifconfig down
Done
```

- Apply a factory reset on 802.15.4 radio

```
root@imx8mmevk:/usr/sbin# ot-ctl factoryreset
```

- Set the channel of operation

```
root@imx8mmevk:/usr/sbin# ot-ctl channel 26
Done
```

- Set the panid and extpanid of the network to join

```
root@imx8mmevk:/usr/sbin# ot-ctl panid 0x1234
Done
root@imx8mmevk:/usr/sbin# ot-ctl extpanid dead00beef00cafe
Done
```

- Set the networkkey for the network

```
root@imx8mmevk:/usr/sbin# ot-ctl networkkey
```

Command output example:

```
00112233445566778899aabbccddeeff
Done
```

- Set name for the network

```
root@imx8mmevk:/usr/sbin# ot-ctl networkname ThreadTest
Done
```

- Start the Thread network

```
root@imx8mmevk:/usr/sbin# ot-ctl ifconfig up
Done
root@imx8mmevk:/usr/sbin# ot-ctl thread start
Done
```

- Ensure that the device state is leader

```
root@imx8mmevk:/usr/sbin# ot-ctl state
```

Command output example:

```
leader
Done
```

Initially the device state is going to be leader.

Once the End device is accepted as a child by the Parent device, its state changes to Child.

```
root@imx8mmevk:/usr/sbin# ot-ctl state
Child
Done
root@imx8mmevk:/usr/sbin# ot-ctl ipaddr
fdc0:de7a:b5c0:0:0:ff:fe00:0c01 # Routing Locator (RLOC)
fdc0:de7a:b5c0:0:66bf:99b9:24c0:d55f # Mesh-Local EID (ML-EID)
fe80:0:0:0:18e5:29b3:a638:943b # Link-Local Address (LLA)
Done
```

**Step 3** – Ping the child using the mesh-local address

```
root@imx8mmevk:/usr/sbin# ot-ctl ping fdc0:de7a:b5c0:0:66bf:99b9:24c0:d55f
16 bytes from fdc0:de7a:b5c0:0:66bf:99b9:24c0:d55f icmp_seq=1 hlim=64 time=17ms
```

**Step 4** – Run the throughput test using iPerf tool and mesh-local address.

- Start iPerf server at device A (Leader or Child)

```
root@imx8mmevk:/usr/sbin# iperf -s -u -i 1 -w 400k -p5005 -V -B <mesh-local address of device A>
```

- Start iPerf client at device B (Child or Leader)

```
root@imx8mmevk:/usr/sbin# iperf -c <mesh-local address of device A> -B <mesh-local address of device
 B> -u -b 250k -l500 -V -i1 -t 20 -p5005
```

## 5.4 Coexistence

### 5.4.1 Introduction to coexistence

The coexistence architecture has three major components:

- Central hardware Packet Traffic Arbiter (PTA): arbitrates between on-chip Wi-Fi and narrowband radios. Controls the front end components such as RF switches.
- Local hardware arbiter: arbitrates the packets between Bluetooth and Bluetooth Low Energy (LE).
- Coexistence software: configures the central HW PTA and works with the Wi-Fi and narrowband firmware.

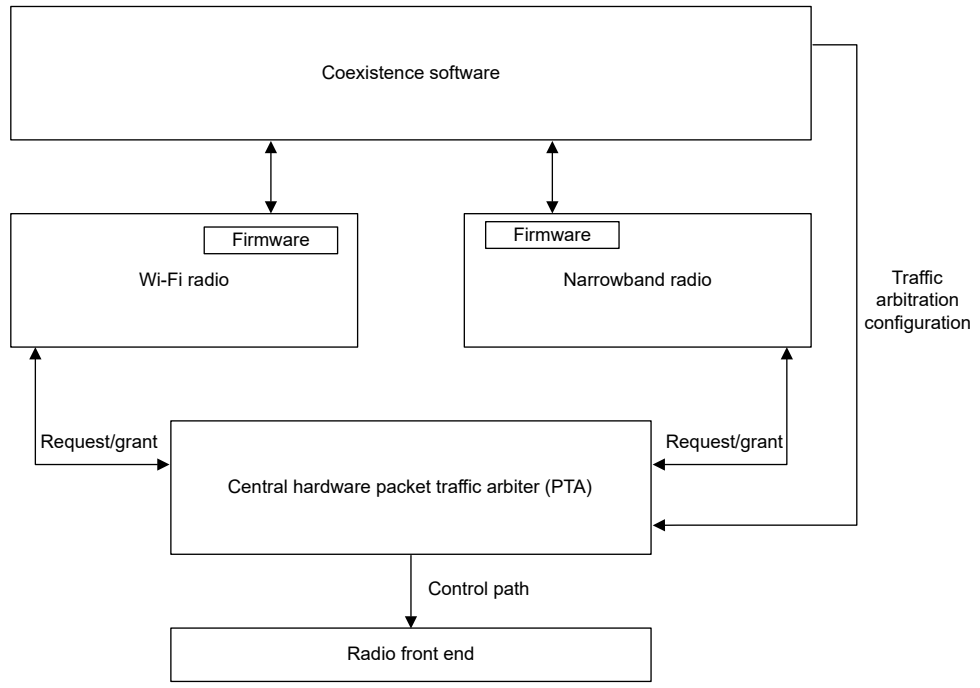



**Figure 45. Coexistence subsystem**

The coexistence mechanism of NXP is a combination of interference avoidance and arbitration between the Wi-Fi and narrowband radios.

Interference avoidance is the coordination between the Wi-Fi and narrowband radios in order to avoid overlapping frequency usage. The coordination helps the narrowband radio to adapt the AFH map and avoid hopping into the Wi-Fi channel, reducing the interference to each other.

In addition to interference avoidance, the central HW PTA provides real-time arbitration between the Wi-Fi and narrowband radios on a per-packet basis. This arbitration can be statically enabled/disabled. The individual radios post a request to the central HW PTA to access the radio front-end. The central HW PTA grants access to the individual radios based on the configured priorities and grant rules.

### 5.4.2 Operating mode – Antenna configuration

#### 5.4.2.1 Shared antenna application

In a shared antenna application, one antenna is shared between the Wi-Fi and narrowband radios. Access to the antenna is through an RF switch. The Wi-Fi and narrowband radios do not have simultaneous access to the antenna. The central HW PTA manages the arbitration between the Wi-Fi and narrowband radio access to the antenna, and controls the switch in real time.

#### 5.4.2.2 Dedicated antenna application

In applications where the Wi-Fi and narrowband radios have a dedicated antenna, the central HW PTA arbitrates between the Wi-Fi and narrowband radios. The need for arbitration between the Wi-Fi and narrowband radios depends on the antenna isolation, the target output powers, and the operating environments. The arbitration setup can be reviewed for each product implementation.

#### 5.4.2.3 Antenna isolation

**Low antenna isolation**

The configuration uses coexistence schemes to balance the throughput performance. Depending on the arbitration time, the throughput is a percentage of a baseline for each radio. Bluetooth audio profiles are protected from glitches.

The central HW PTA controls the simultaneous transmission of the Wi-Fi and narrowband radios for regulatory purposes.

**High antenna isolation**

The configuration enables concurrent operation of the Wi-Fi and narrowband radios. The throughput is close to the baseline.

# 6 Certifications

Wireless devices must go through certifications and compliance testing before being used in commercial applications.

- Regulatory certifications: certifications for different countries / regions with regulatory rules that a device must abide by to be allowed operation in that country / region. The certifications are related with RF, physical layer performance, and conformance rather than specific protocol layer conformance.
- Standards-based certifications and qualification testing: connectivity technology certifications offered by the Wi-Fi alliance, Bluetooth special interest group (SIG), Thread Group, Connectivity Standards Alliance (Zigbee, Matter). These certify that a device complies to standards, and check for device interoperability.
- Security certifications: certifications such as Federal Information Processing Information Standards (FIPS) for security.

The last section introduces NXP pro support services for enhanced certification support.

## 6.1 Regulatory certifications

If using modules, note that:

- The module manufacturer has already gone through regulatory certifications.
- Some countries do not allow modular regulatory certifications, and the regulatory certification testing may be required on the end-product.

Regulatory certification testing refers to RF and physical layer performance and conformance rather than protocol layer testing. Tests items such as dynamic frequency selection (DFS) and adaptivity refer to some aspects of the protocol layer to test the conformance to the standards.

For an overview of compliance and certification, refer to ref.[1].

Contact the module maker vendor for any question about regulatory certification on the module being used.

### 6.1.1 Device calibration and performance optimization

Before any regulatory certification testing, the module should already be characterized and tuned for optimal transmit and receive performance. To check if any additional tuning must be completed in the final application, contact the module maker.

### 6.1.2 Tools in use for regulatory certification testing

The module maker should have completed the regulatory certification testing.

Some countries do not allow modular certification, or require the regulatory certification to be completed on the final product. Confirm with the module maker which countries the module has already been certified in.

For cases where regulatory certification testing must be performed on the final product, use RF test mode in the NXP production software.

To learn how to use RF test mode during certification testing, refer to ref.[3].

### 6.1.3 Guidelines for specific regulatory tests

Module makers should have completed the Dynamic frequency selection (DFS), EU Adaptivity, and Japan carrier sense regulatory tests.

Some detection thresholds may need to be adjusted when the module is paired onto the final platform depending on the antenna gain and cable losses.

Contact your module maker to know if any adjustment is needed when the module is used on the final platform.

### 6.1.4 Creating transmit power tables from regulatory certification passing limits

During the regulatory certification, the targeted transmit output power levels are set and tested for compliance with the regulatory limits for each country and/or region.

Contact your module maker to obtain the regulatory compliant transmit power tables for your module, and to know if any adjustment is required for the antenna gain or path loss on the final product.

## 6.2 Standards-based certifications and qualification testing

In addition to the regulatory certifications required for the country where the device will be used, the standards-based certification or qualification testing allow to advertise the compliance of the device on the website of the standards organizations, and promotes the device interoperability with other device vendors.

### 6.2.1 Wi-Fi alliance (WFA)

Wi-Fi alliance offers certification for a range of Wi-Fi standards and features. Refer to ref.[35], ref.[2], and ref.[19].

### 6.2.2 Bluetooth special interest group (Bluetooth SIG)

The Bluetooth SIG qualification program is used to qualify devices. Refer to ref.[36].

NXP performs the controller certification for each Bluetooth device. To speed up the certification process and reduce costs, re-use NXP qualified design ID (QID) or device number (DN) for a given device. Since the RF and RFPHY part of the controller testing must be performed on each final product, run these specific tests at a Bluetooth qualified test facility (BQTF).

- For Bluetooth classic qualification testing, procedure to enable scan and test mode, refer to ref.[3].
- For Bluetooth LE qualification testing, standard HCI commands and syntax to use to manually perform the qualification testing, refer to ref.[3].

***Note:*** *Contact your module maker or local NXP representative for the list of QDID / DN to refer to for the different NXP Bluetooth devices.*

### 6.2.3 Thread

NXP obtains Thread certification on its devices. If you keep the default Thread system functionality, the Thread Group offers certification via inheritance:

- Derivative products may inherit the certification of a parent product.
- Modules may inherit stack certification.
- End products may inherit component certification.

To learn about Thread certification processes, costs, timeline, and certification via inheritance, refer to ref.[37].

### 6.2.4 Zigbee

The Connectivity Standards Alliance (CSA) maintains Zigbee worldwide open standard. Certification allows the use of certified product logos and lists the product on the CSA website. In addition, certifications show compliance and interoperability in Zigbee networks.

The main types of certification for Zigbee are platform certification and end-product certification. Use an authorized test lab to perform the official testing.

NXP obtains Zigbee certification on its devices. For an example of Zigbee certification on NXP solutions, refer to ref.[20]. And for details about Zigbee certification, refer to ref.[38].

### 6.2.5 Matter

The Matter smart home standard requires all Matter devices to be certified in order to commission into or join a Matter network. The certification for the networking protocol used on the product (Thread, Bluetooth, Bluetooth LE, and/or Wi-Fi) must be completed before the testing for Matter certification.

To test your product for Matter certification, use a CSA approved test lab (ATL), or use a CSA hosted specification validation event (SVE). NXP is a CSA promoter, and one of the leading companies that contribute to new specifications and development of CSA SDK. NXP appointed partner labs are familiar with NXP software operation and NXP boards operation, through SVE and certifications. Customers can benefit from NXP partner lab when arranged through NXP Pro Support Wireless Certification Concierge.

Contact your local NXP representative for more information regarding NXP Matter certification via NXP pro support services for wireless certification.

For more information about matter and certification, refer to ref.[39], and ref.[40].

## 6.3 Security

For security certification, refer to standard and certification programs such as Federal Information Processing Information Standards (FIPS).

### 6.3.1 FIPS

The American Federal Information Processing Standard (FIPS) defines the security requirements for cryptographic modules. NXP wireless SoCs contain an encryption block (AEU) for FIPS grade encryption and decryption.

## 6.4 NXP pro support services

If you have little experience with testing for certification and compliance, or to offload some certification tasks to NXP, you can use NXP pro support services.

- Regulatory certifications
  - NXP monitors regulation updates worldwide.
  - NXP acts as certification lab liaison with global labs to consolidate the testing, certification proposals, and certification options.
  - NXP can oversee and triage issues related to certification.
- Wi-Fi Alliance (WFA)
  - As a WFA approved solutions test lab, NXP can perform WFA approved Quick Track Tests on your platform, and help to complete WFA certification process.
- Bluetooth SIG
  - As a Bluetooth recognized test facility (BRTF), NXP can perform Bluetooth RF qualification pre certification testing on customer designs, and provide support in case of issues.
  - As BRTF, NXP can qualify requests for customized firmware builds from customers.
  - NXP in-house Bluetooth qualified consultant (BQC) can provide initial guidance and discuss with your appointed BQC for a project.
  - NXP performs Bluetooth stack certification testing on customer designs through partner labs, and provides support in case of issues.
  - NXP acts as certification lab liaison with global labs to consolidate the testing, certification proposals, and certification options.
- Zigbee
  - NXP acts as certification lab liaison with global labs to consolidate the testing, certification proposals, and certification options.
- Matter
  - NXP acts as certification lab liaison with global labs to consolidate the testing, certification proposals, and certification options.
- Apple CarPlay
  - NXP performs Apple CarPlay pre-certification tests on customer designs.
  - NXP can arrange an Apple qualified CarPlay certification lab to pre-certify a customer design and provide support in case of issues.

To complete the certification prior to launching your product, contact the certification concierge of NXP pro support services. Refer to your local NXP representative and ref.[41].

# 7 PSIRT and vulnerability management

## 7.1 Introduction

NXP has a well-established vulnerability management process.

Benchmarked against recognized Industry standards (not limited to):

- **CERT®** – Guide to Coordinated Vulnerability Disclosure (August 2017)
- **FIRST** – Guidelines and Practices for Multi-party Vulnerability Coordination (v1.1; Spring 2020)
- **Auto-ISAC** – Best Practice Guide on Incident Response (v1.3; July 2019)
- **ISO/IEC 29147:2018** – Vulnerability disclosure
- **ISO/IEC 30111:2019** – Vulnerability handling processes
- **ISO/SAE 21434:2021** – Road vehicles – Section for ‘Cybersecurity event assessment’ and ‘Cybersecurity incident response’

A cohesive and dedicated team of stakeholders executes NXP process:

- Product security incident response team (PSIRT)
  - Central contact point reachable via email and ref.[33].
- Incident response core team (IRCT)
  - Dedicated team for each incident management
- Security champions in global support teams
  - For all regions
  - Go-to persons for customer communication and questions

## 7.2 Product security incident process (PSIRP)

Figure 46 illustrates the product security incident process (PSIRP).


**Figure 46. Product security incident process (PSIRP)**

**Stage 1** – Receive the report.

- The team receives and acknowledges the report.

**Stage 2** – Evaluate the vulnerability.

The team:

- Determines the affected products and customers.
- Assesses the risk.
- Determines the impact.
- Assigns an initial severity level.

**Stage 3** – Define the solution.

The incident response core team (IRCT):

- Develops the mitigation strategies.
- Prepares the customer communication plan.
- Determines the feasibility of a fix.
- Involves the global sales and support teams.

**Stages 4 to 6** – Communication to customers and closure

- In most cases, NXP issues directed communication to affected customers through the global support channels.
- In certain cases, Auto-ISAC and CERTs may be notified, and in extreme situations, law enforcement may also be involved.
- NXP might issue a Common Vulnerabilities and Exposures (CVE) for the vulnerability.
- NXP follows a coordinated disclosure policy which allows customers to assess and protect their products from the impact of disclosure. See Coordinated disclosure and timely communication.

## 7.3 Coordinated disclosure and timely communication

- NXP believes in responsible and coordinated disclosure to protect the customer's interest
- NXP commits to communicate and act on vulnerabilities in a “timely manner” [1].
- Report receipt
  - NXP strives to acknowledge the receipt of vulnerability reported through PSIRT channel within 24 hours.
- Initial customer communication following a detailed investigation and the definition of a solution.
- Remediation efforts
  - Coordinated, aligned and agreed with the affected customers.

1 A "timely manner" for acting on vulnerabilities varies considerably and is incident-specific. However, the vulnerability process is completed within 12 weeks for a software solution, including availability of patches and notification of the issue. A hardware fix can take considerably longer to address than a software fix. Additionally, a fix that has to be deployed to devices can take time to roll out compared with a server software fix.

# 8 Abbreviations

**Table 16. Abbreviations**

| Abbreviation | Description |
|---|---|
| A2DP | advanced audio distribution profile |
| AFH | adaptive frequency hopping |
| AP | access point |
| API | application programming interface |
| ATL | approved test lab |
| AXIHP | advanced extensible interface high performance |
| BD | Bluetooth device |
| BQC | Bluetooth qualified consultant |
| BRTF | Bluetooth recognized test facility |
| BSSID | basic service set identifiers |
| CLI | command line interface |
| CMD | command |
| CPU | central processor unit |
| CSA | connectivity standards alliance |
| CVE | common vulnerabilities and exposures |
| DDR | double data rate |
| DFS | dynamic frequency selection |
| DN | device number |
| DUT | device under test |
| ED | energy detection |
| EUI-64 | 64-bit extended unique identifier |
| EVK | evaluation kit |
| FED | full end device |
| FIPS | federal information processing information standards |
| FT | fast transition |
| FTD | full thread device |
| FW | firmware |
| GATT | generic attribute profile |
| GPIO | general purpose input output |
| HCI | host controller interface |
| HFP | hands free profile |
| HID | human interface device |
| HW | hardware |
| IEEE | institute of electrical and electronics engineers |

**Table 16. Abbreviations**...*continued*

| Abbreviation | Description |
|---|---|
| IPv6 | internet protocol version 6 |
| IR | independent reset |
| IRCT | incident response core team |
| LE | low energy |
| MAC | media access control |
| MFG | manufacturing |
| MTD | minimal thread device |
| NB | narrow band |
| NCP | network coprocessor |
| OCF | opcode command field |
| OGF | opcode group field |
| OOB | out of band |
| OPP | object push profile |
| OTP | one time programmable |
| OWE | opportunistic wireless encryption |
| PMS | power save mode |
| PSIRP | product security incident process |
| PSIRT | product security incident response team |
| PTA | packet traffic arbitration |
| PTA | Packet Traffic Arbiter |
| QID | qualified design |
| QoS | quality of service |
| RCP | radio coprocessor |
| REED | router enabled end device |
| SDK | software development kit |
| SED | sleepy end device |
| SIG | special interest group |
| SoC | system on chip |
| SPI | serial peripheral interface |
| STA | station |
| STBC | space time block coding |
| SW | software |
| TDLS | tunneled direct link support |
| TP | throughput |
| UART | universal asynchronous receiver transmitter |
| WCI-2 | wireless coexistence interface 2 |

**Table 16. Abbreviations***...continued*

| Abbreviation | Description |
|---|---|
| WFA | Wi-Fi alliance |
| WLAN | wireless local area network |
| WoWLAN | wake on wireless local area network |
| WPA | Wi-Fi protected access |

# 9 References

[1] Application note – AN13006: Compliance and Certification Considerations (link)
[2] Application note – AN12976: Wi-Fi Alliance Derivative Certification Process for Linux OS (link)
[3] Application note – AN14114: RF Test Mode on Linux OS (link)
[4] Application note – AN13958: Host Wake-up over Bluetooth or Bluetooth Low Energy (LE) (link)
[5] Application note – AN14212: 802.11kvr Roaming (link)
[6] Application note – AN14213: Wi-Fi Firmware Automatic Recovery for Linux OS (link)
[7] Application note – AN14310: NXP Bluetooth UART Driver Integration (link)
[8] Nordic Semiconductor - nRF Connect for Desktop (link)
[9] Software – Embedded Linux for i.MX application processors (link)
[10] Release notes – RN00104: NXP Wireless SoC Features and Release Notes for Linux OS (link)
[11] Specification – IEEE802.15.4-2015
[12] Specification – Hands-Free Profile Specification(link)
[13] Specification – Gatt Bluetooth specification (link)
[14] Specification – Spinel protocol (link)
[15] User guide – NXP – i.MX Yocto Project User's Guide (link)
[16] User guide – i.MX Linux user's guide (link)
[17] User manual – IMX93EVKQSG: i.MX 93 EVK Quick Start Guide (link)
[18] User manual – UM11483: Getting Started with NXP-based Wireless Modules on i.MX 8M Quad EVK Running Linux OS (link)
[19] User manual – UM11560: WFA Certification Guide for NXP-based Wireless Modules on i.MX 8M Quad EVK Running Linux OS (link)
[20] User manual – UM12200: Zigbee certification for tri-radio single-chip solutions (link)
[21] Webpage – 88W8801: 2.4 GHz Single-band 1x1 Wi-Fi® 4 (802.11n) Solution (link)
[22] Webpage – 88W8987: 2.4/5 GHz Dual-band 1x1 Wi-Fi® 5 (802.11ac) + Bluetooth® Solution (link)
[23] Webpage – 88W8997: 2.4/5 GHz Dual-band 2x2 Wi-Fi® 5 (802.11ac) + Bluetooth® Solution (link)
[24] Webpage – 88W9098: 2.4/5 GHz Dual-band 2x2 Wi-Fi® 6 (802.11ax) + Bluetooth® (link)
[25] Webpage – IW416: 2.4/5 GHz Dual-band 1x1 Wi-Fi® 4 (802.11n) + Bluetooth® Solution (link)
[26] Webpage – IW610: 2.4/5 GHz Dual-Band 1x1 Wi-Fi® 6 + Bluetooth Low Energy + 802.15.4 Tri-Radio Solution (link)
[27] Webpage – IW611: 2.4/5 GHz Dual-band 1x1 Wi-Fi® 6 (802.11ax) + Bluetooth® Solution (link)
[28] Webpage – IW612: 2.4/5 GHz Dual-band 1x1 Wi-Fi® 6 (802.11ax) + Bluetooth® + 802.15.4 Tri-radio Solution (link)
[29] Webpage – Getting Started with FRDM-IMX91 Development Board (link)
[30] Webpage – Getting Started with FRDM-IMX93 (link)
[31] Webpage – Thread group (link)
[32] Webpage – OpenThread introduction (link)
[33] Webpage – Product Security Vulnerability (link)
[34] Webpage – WirePlumber (link)
[35] Webpage – Certification | Wi-Fi Alliance (link)
[36] Webpage – Develop with Bluetooth, qualify your product (link)
[37] Webpage – Thread certification ( link)
[38] Webpage – Zigbee, the full-stack solution for all smart devices (link)
[39] Webpage – matter, the foundation for connected things (link)
[40] Webpage – Certification, bring credibility to your product (link)
[41] Webpage – NXP Engineering Services – Wireless Connectivity Services (link)
[42] Webpage – NXP Community homepage (link)

[43] Webpage – NXP support (link)

# 10 Note about the source code in the document

The example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# 11 Revision history

**Table 17. Revision history**

| Document ID | Release date | Description |
|---|---|---|
| RM00297 v.1.0 | 24 April 2025 | • Initial version |

# Legal information

## Definitions

**Draft** — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

## Disclaimers

**Limited warranty and liability** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

**Right to make changes** — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Terms and conditions of commercial sale** — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

**Export control** — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

**Suitability for use in non-automotive qualified products** — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

**HTML publications** — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

**Translations** — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

**Security** — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

**NXP B.V.** — NXP B.V. is not an operating company and it does not distribute or sell products.

## Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

**NXP** — wordmark and logo are trademarks of NXP B.V.

**Bluetooth** — the Bluetooth wordmark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

**Matter, Zigbee** — are developed by the Connectivity Standards Alliance. The Alliance's Brands and all goodwill associated therewith, are the exclusive property of the Alliance.

# Tables



# Figures

# Contents