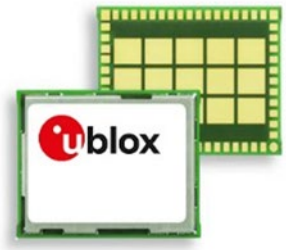
S

Android integration for NXP-based modules

EMMY-W1, LILY-W1 and JODY-W2

Application note



Abstract

This document describes the procedures for integrating u-blox Wi-Fi and Bluetooth modules with Android BSP. Based on an example that describes how to integrate the EMMY-W1 module with Android 9, the same procedures are also relevant for integrating other u-blox modules based on NXP chipsets with different Android versions.

UBX-19035432 - R02
C1-Public

www.u-blox.com

Document information

Title	**Android integration for NXP-based modules**	
Subtitle	EMMY-W1, LILY-W1 and JODY-W2	
Document type	Application note	
Document number	UBX-19035432	
Revision and date	R02	8-Sep-2020
Disclosure restriction	C1-Public	

This document applies to the following products:

Product name	Firmware version (tested)	Product status
EMMY-W1	SD-WLAN-UART-BT-8887-U16-MMC-W15.44.19.p36-15.100.19.p36-C4X15657_A2-GPL SD-WLAN-SD-BT-8887-U16-MMC-W15.68.19.p38-15.26.19.p38-C4X15659_A2-MGPL	Mass production
LILY-W1	SD-UAPSTA-8801-U16-X86-W14.68.36.p146-C4X14666_B0-MGPL USB-UAPSTA-8801-U16-X86-W14.68.36.p146-C4X14666_B0-MGPL	Mass production
JODY-W2	SD-WLAN-UART-BT-8987-U16-MMC-W16.87.10.p33-16.26.10.p33-C4X16651-MGPL SD-WLAN-SD-BT-8987-U16-MMC-W16.68.10.p33-16.26.10.p33-C4X16651-MGPL	Engineering Sample

Contents

1 Introduction

u-blox have integrated and tested several modules, based on NXP[1] chipsets, with different Android versions. This application note describes the essential procedures for integrating NXP-based u-blox modules with Android 9 or above.

Although this document describes how to integrate the EMMY-W1 module with Android 9 using SDIO-UART driver package, the given procedures are also relevant for integrating other NXP-based u-blox modules.

Section 3 and section 4 explain the procedures for integrating Wi-Fi and Bluetooth using SDIO and UART, respectively.

Section 5 describes the additional changes that are required for designs that use the SDIO interface for both radios.

☞ The Android package is provided by the host platform vendor. In this case, NXP semiconductors who manufacture i.MX8 EVK platforms. The package is a customized version of the Android open source package 9.0.0_2.3.4.

☞ The procedures described in this document are applicable to all u-blox modules based on NXP chipsets – regardless of the Android package.

☞ All vendors that manufacture host platforms for Android development must provide the appropriate set of kernel package and Android sources.

☞ Platform vendors normally integrate at least one Wi-Fi module with their manufactured platform. Before starting any integration of u-blox modules, be sure to read the user guide for the EVK [2] and host platform.

The supported module interfaces and the drivers used to test them are shown in Table 1.

Module	Interface	Driver package
EMMY-W1	SDIO-SDIO	SD-WLAN-SD-BT-8887-U16-MMC-W15.68.19.p38-15.26.19.p38-C4X15659_A2-MGPL
	SDIO-UART	SD-WLAN-UART-BT-8887-U16-MMC-W15.44.19.p36-15.100.19.p36-C4X15657_A2-GPL
LILY-W1	USB	USB-UAPSTA-8801-U16-X86-W14.68.36.p146-C4X14666_B0-MGPL
	SDIO	SD-UAPSTA-8801-U16-X86-W14.68.36.p146-C4X14666_B0-MGPL
JODY-W2	SDIO-SDIO	SD-WLAN-SD-BT-8987-U16-MMC-W16.68.10.p33-16.26.10.p33-C4X16651-MGPL
	SDIO-UART	SD-WLAN-UART-BT-8987-U16-MMC-W16.87.10.p33-16.26.10.p33-C4X16651-MGPL

Table 1: Driver packages used for tested interfaces

1.1 Scope

Several regular use cases supported by Wi-Fi and Bluetooth, including station mode, access point, peer to peer and BT connection, are routinely tested against the interfaces supported by u-blox modules.

[1] Earlier chipsets were sold under the banner of Marvell Semiconductors. After the acquisition, the packages and libraries are renamed with NXP as the vendor.

u-blox have used the following hardware and software components for the integration:

- Android BSP: Android_9.0.0_2.3.4
- Host platform: i.MX8 MQ EVK
- Short range host-based module: EMMY-W1 (SDIO-UART)
- Driver: SD-WLAN-UART-BT-8887-U16-MMC-W15.44.19.p36-15.100.19.p36-C4X15657_A2-GPL
- Wireless HAL: NXP Vendor HAL - v005

☞ The main body of this document is primarily targeted towards the integration of the module through the SDIO-UART interface. For information describing integration of the module through the SDIO-SDIO interface, see section 5. When integrating u-blox modules based on NXP chipsets, make sure that you have access to the appropriate System integration manual (SIM) and Data sheet (DS) for your product.

1.2 Workflow

Use the following workflow to integrate the module:

1. Detect the module on host platform.
   Check if the device is detected first on the machine after booting up. This step does not require any change in the source code. Verifying this ensures that hardware integration is correct.
2. Load the driver and firmware followed by interface up.
   Compile the driver and try to load during the initialization sequence. Communication with the module has been established when the interfaces are shown after the firmware download.
3. Initialize Wi-Fi background services and libraries to bring up the interface.
   At this point different layers in Android stack are connected and end-to-end communication is established. The stack then confirms that the vendor HAL is configured properly.
4. Connect the GUI and check for individual use-cases: Station mode, Access point, Wi-Fi Direct, and Bluetooth. For further details see section 7.

2 Components and device setup

2.1 Host platform

The u-blox EVK is evaluated together with the MCIMX8M-EVK host platform from NXP. The configuration setup for the EVK and NXP host platform communication is shown in the Figure 2.

MCIMX8M-EVK supports the following features:

- Industrial grade NXP IMX8M host processor
- Support for both Yocto and Android
- Extensive peripherals for most evaluation scenarios
- Evolving update of BSP for the host platform

For more detailed information, check the specifications of the MCIMX8M-EVK Evaluation Kit for the i.MX 8M Applications Processor [16].

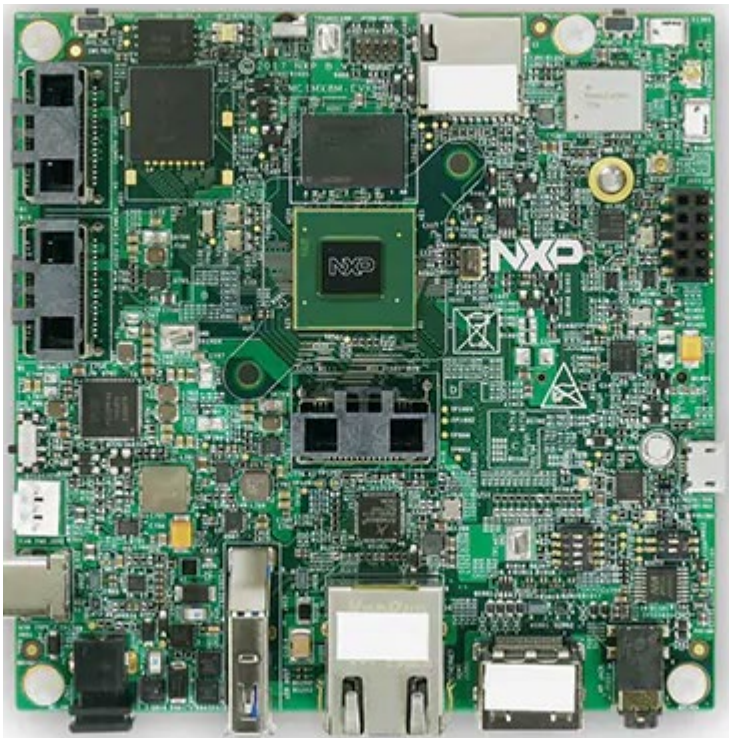


Figure 1: NXP iMX8 MQEVK

2.2 Setup

Figure 2 shows the typical setup configuration for EVK with host platform communication.

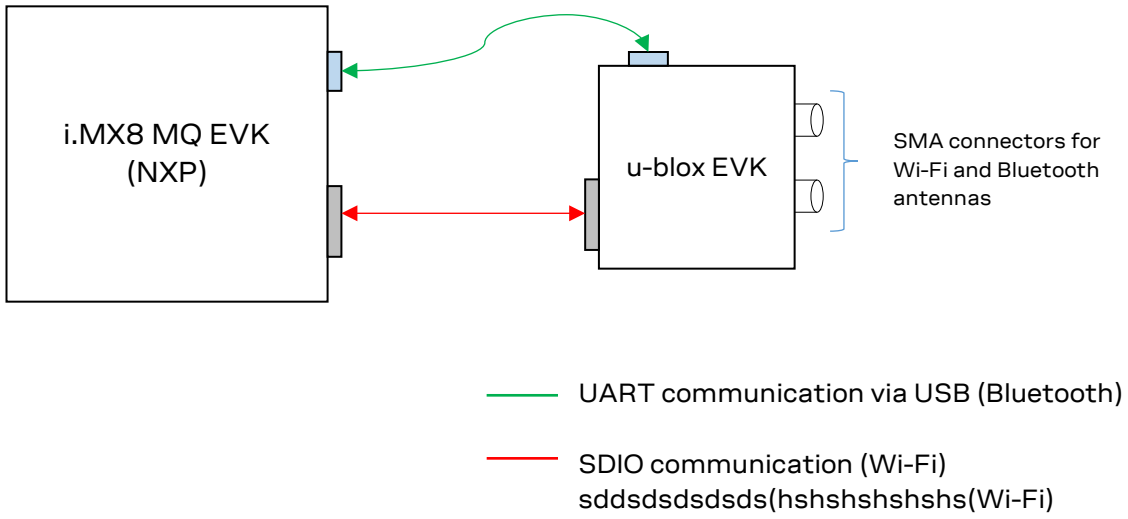


Figure 2: EVK and host platform communication

2.3 Build system

A Linux host (for example Ubuntu 16.04 LTS) is required. You use this host to compile the Android code. The recommended hardware requirements for the host include:

- Intel i7 or Xeon recommended
- 16GB+ RAM
- 200GB+ hard disk (SSD recommended)

2.4 Software packages

Different packages such as the Android open source code, drivers and HAL package required to integrate the u-blox modules based on the NXP chipset with Android 9 or above are explained in brief in the sections below.

2.4.1 Android Open Source Project

This section explains the procedure for building the Android image from scratch on an NXP i.MX8 platform. The following software packages are available for the host platform:

- Android 9.0.0_2.3.4

☞ The following process is generally relevant for the above packages and any future versions of the Android BSP from NXP. The version of the Linux kernel integrated with this image is: *4.14.98.*

Follow the process outlined below to build the hardware image:

1. Install the appropriate packages.

```
$: apt install uuid uuid-dev zlib1g-dev liblz-dev liblzo2-2 liblzo2-dev lzop git-
core curl u-boot-tools mtd-utils android-tools-fsutils openjdk-8-jdk device-tree-
compiler gdisk m4 libz-dev bison flex libssl-dev curl android-tools-fsutils
```

2. Setup the download.

   Install the `repo` tool first when using the manifest repository.

```
$: mkdir ~/bin
$: curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$: chmod a+x ~/bin/repo
$: export PATH=${PATH}:~/bin
```

3. Download the manifest.

   Download and extract the source directory from the link:

   https://www.nxp.com/webapp/Download?colCode=P9.0.0_2.3.4_ANDROID_SOURCE&appType=license

   The directory includes the necessary configuration to fetch packages from the different repositories.

4. Setup the source.

   Run the setup script to clone Android source code. It is a modified version of open-source package to cater the host platforms developed by NXP. It will have changes pertaining to device trees, display drivers etc.. This typically takes 5-6 hours - dependent on the network speed and machine.

   ```
   Path: imx-p9.0.0_2.3.4/
   $: source ./imx-p9.0.0_2.3.4/imx_android_setup.sh
   ```

   At this point, all the files and directories needed for Android image building are downloaded in the folder `android_build`. Before starting the build procedure, integrate the device driver and make the necessary changes in different layers to work with the Wi-Fi and Bluetooth module.

5. Setup the build environment.

   Enter the following terminal command to configure the Android build:

   ```
   Path: imx-p9.0.0_2.3.4/android_build/
   $: source build/envsetup.sh
   ```

6. Enter the Android `lunch` commandthe build has completed successfully. All build artefacts and images can be found in `android_build/out/target/product/evk_8mq/`.

2.4.2 Driver package

The reference driver developed by NXP is distributed exclusively to customers that have signed a Limited Use License Agreement (LULA-M) with u-blox or Nondisclosure Agreement (NDA) with NXP. To obtain the driver package and vendor HAL, contact u-blox support for your area, as shown in the Contact section of this document.

The EMMY-W1 module is based on the NXP 88W8887 chipset. It can be connected to the host in couple of ways using different physical interfaces that use the driver packages, as shown in Table 3. For information about the specific driver packages used for testing each module variant, see Table 1.

Driver package	Interface		Remarks
	Wi-Fi	Bluetooth	
SDIO-UART-XX	SDIO	UART	See chapters 3 and 4.
SDIO-SDIO-XX	SDIO	SDIO	See section 5.

Table 2: Physical interfaces and driver packages

For information about the specific driver packages used for testing each module variant, see Table 1.

The SDIO-UART driver package structure is described below:

```
SD-WLAN-UART-BT-8887-U16-MMC-W15.44.19.p36-15.100.19.p36-C4X15657_A2-GPL (Dec 2019)
├── FwImage                          ┐   sduart8887_combo_a2.bin is used for operating
│   ├── sd8887_wlan_a2.bin           ├-  Wi-Fi and Bluetooth modules in Android. Rest
│   ├── sduart8887_combo_a2.bin      ┘   are needed for individual radios.
│   └── uart8887_bt_a2.bin
├── SD-UAPSTA-8887-U16-MMC-W15.44.19.p36-C4X15657_A2-app-src.tgz
├── SD-UAPSTA-8887-U16-MMC-W15.44.19.p36-C4X15657_A2-GPL-src.tgz
├── SD-UAPSTA-8887-U16-MMC-W15.44.19.p36-C4X15657_A2-mlan-src.tgz
└── UART-BT-8887-U16-X86-15.100.19.p36-2.2-M2614100-GPL-src.tgz
```

2.4.3 Vendor HAL

Both the driver package (SD-WLAN-UART-BT-8887-U16-MMC-W15.44.19.p36-15.100.19.p36-C4X15657_A2-GPL) and vendor HAL (NXP Vendor HAL - v005) are developed by NXP[2]. It is expected that the HAL structure will remain the same for all future packages. Moreover, all extracted library content must be kept in specific path so that the HAL can compile together with future Android builds.

```
NXP_Vendor_hal_005
├── libbt                    → Source code for vendor HAL for Bluetooth
│   └── conf                 → Conf file specific to Bluetooth of the combo chip
└── wlan
    ├── 1.2                  → Service to integrate with native Wi-Fi framework
    ├── config               → Config files needed to start wpa_supplicant
    ├── libwifi-hal          → Library which provides services specific to scanning
    │   └── wifi_hal-mrvl       collection of stats used by the host framework.
    └── wlan_lib             → Glue layer between wpa_supplicant and the framework
```

[2] The driver and vendor HAL have been previously developed by Mavell and some places in the soure code still use Marvell designations.

3 Wi-Fi integration

3.1 Wi-Fi architecture

Android applications use managers to access system services like Wi-Fi. These managers access some of the vendor-specific functionalities through Hardware Abstraction Layer (HAL).

The Wi-Fi manager accesses the kernel through several layers, including HAL, as shown in Figure 3.

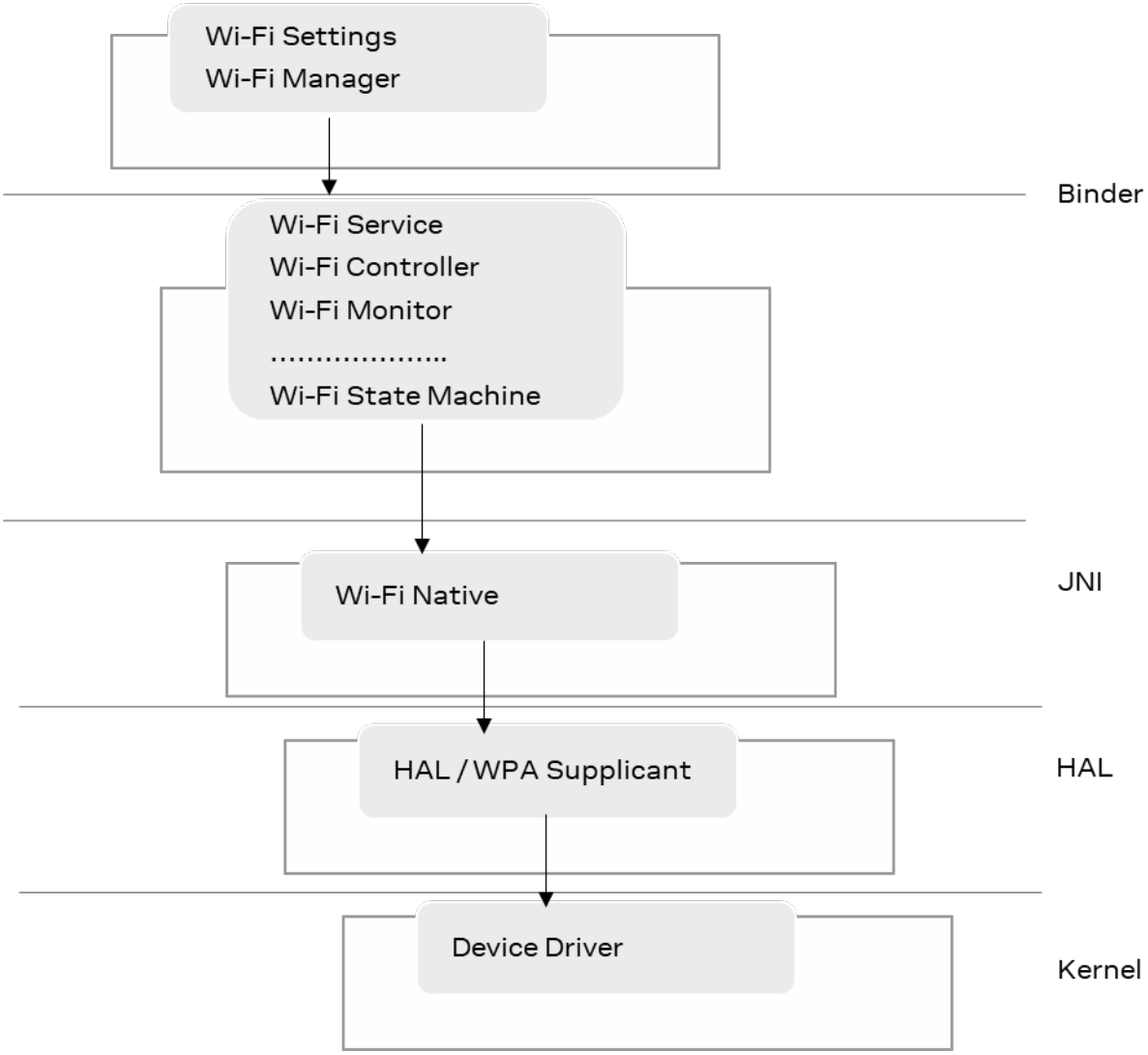


Figure 3: Android Wi-Fi architecture

WifiSettings is an application in the default AOSP build used to control Wi-Fi connections. It uses WifiManager to access Wi-Fi services.

WifiManager provides the following functionalities:

- Provides a list of configured networks
- Monitors the current active Wi-Fi network
- Scans local access points and connects to the network that caters best for the user.

Each action of the Wi-Fi operation is managed in several different logical layers of the module. The initialization process is shown in Figure 4.

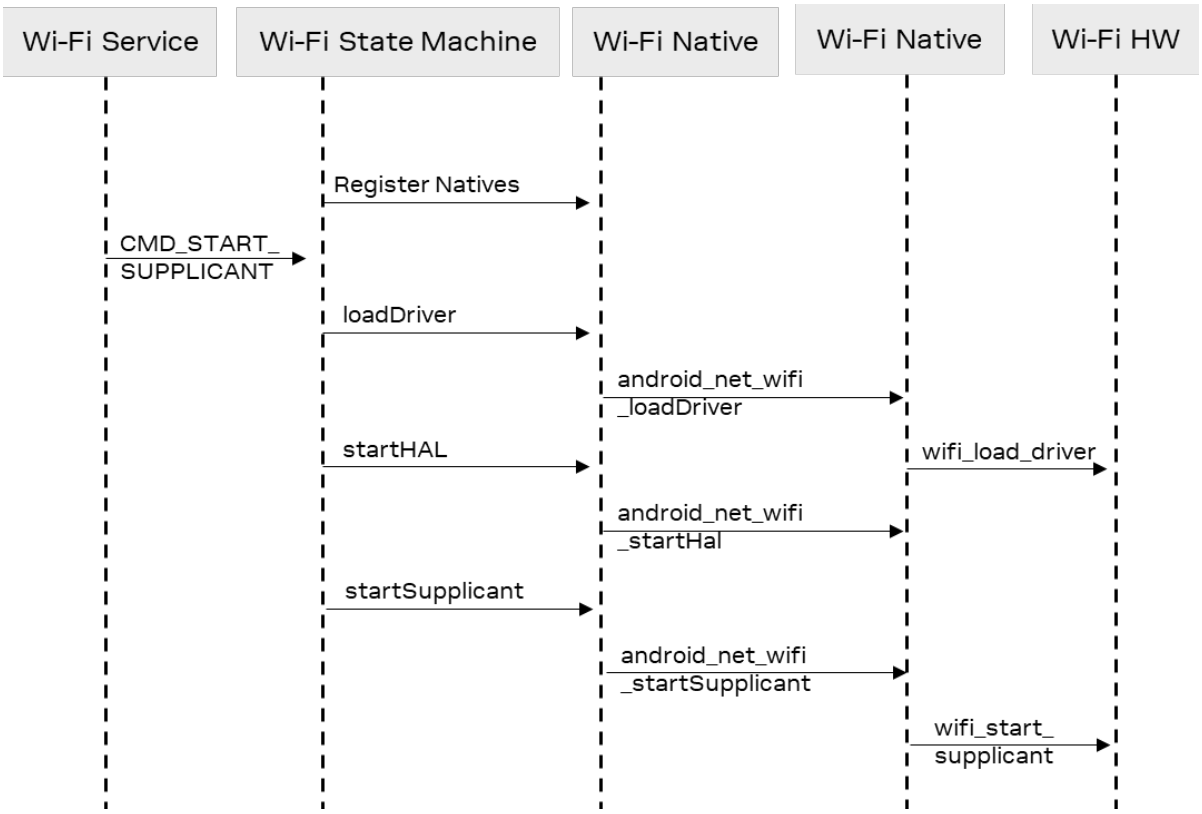


Figure 4: Sequence diagram of Wi-Fi initialization

☞ Most problems in the initialization process are encountered during the loading and activation of Wi-Fi. The Wi-Fi state machine should be of some help in debugging these kind of issues. Once the Wi-Fi is enabled, further operations like scan and join perform smoothly as they do not need any external intervention from the vendor-specific layers.

To get the Wi-Fi driver up and running, several changes must be made to the files and configurations. Various files provided by the chip vendor must be altered to cater for the integration.

3.2 Components

3.2.1 Manifest

To choose the right HIDL interface among the options integrated in AOSP, several changes must be made in the file manifest.

The Wi-Fi HAL chosen in the manifest and shown below uses HIDL interface to communicate with WifiServices. The manifest must contain the tags that are shown and must be added if they are not present already.

☞ If the BSP received from vendor already has these manifest snippets with a different version, then do not alter it. The presence of these snippets indicate the version of the native library that is tested by the vendor.

```
File: android_build/device/fsl/imx8m/evk_8mq/manifest.xml
    <hal format="hidl">
       <name>android.hardware.wifi</name>
       <transport>hwbinder</transport>
       <version>1.2</version>
       <interface>
           <name>IWifi</name>
           <instance>default</instance>
       </interface>
    </hal>
    <hal format="hidl">
       <name>android.hardware.wifi.supplicant</name>
       <transport>hwbinder</transport>
       <version>1.1</version>
       <interface>
           <name>ISupplicant</name>
           <instance>default</instance>
       </interface>
    </hal>
    <hal format="hidl">
        <name>android.hardware.wifi.hostapd</name>
        <transport>hwbinder</transport>
        <version>1.0</version>
        <interface>
            <name>IHostapd</name>
            <instance>default</instance>
        </interface>
    </hal>
```

3.2.2 Driver source

NXP releases driver packages regularly and to obtain updated features and bug fixes we recommend that you use the latest package. The package from NXP requires the contents of different folders to be arranged in a format that reflects the destination of wireless drivers.

```
Destination: android_build/vendor/nxp-
opensource/kernel_imx/drivers/net/wireless/marvell/mrvl8887/
├── muart_src      → Copy the contents of UART-BT-8887-XX into this folder
└── wlan_src       → Copy the contents of SD-UAPSTA-8887-XX into this folder
    ├── Makefile
    ├── mlan
    ├── mlinux
    └── mapp
```

3.2.2.1 Build driver

When building the kernel, select the `MRVL8887` driver for the host platform. In the `BoardConfig.mk` file, check the `TARGET_KERNEL_DEFCONFIG` parameter that defines which kernel configuration file needs to be modified.

```
File: android_build/device/fsl/imx8m/evk_8mq/BoardConfig.mk
TARGET_KERNEL_DEFCONFIG := android_defconfig
```

☞ The exact file that needs to be modified can vary between different Android distributions.

The kernel configuration file shown in this example, `android_defconfig`, is included in the `../arm64/configs/` directory. The path is relative to the architecture of the host platform, such as 32bit arm, 64bit arm, and so on. Add the configuration option to include NXP driver.

```
File: android_build/vendor/nxp-
opensource/kernel_imx/arch/arm64/configs/android_defconfig

CONFIG_WLAN_VENDOR_MARVELL=y
CONFIG_MRVL8887=m
```

The module can be built after the driver has been selected. Select the source code to build the kernel modules by making changes shown below:

```
Path: android_build/vendor/nxp-opensource/kernel_imx/drivers/
/net/wireless/marvell/Makefile
obj-$(CONFIG_MRVL8887) += mrvl8887/wlan_src/
obj-$(CONFIG_MRVL8887) += mrvl8887/muart_src/


Path: android_build/vendor/nxp-
opensource/kernel_imx/drivers/net/wireless/marvell/Kconfig
source "drivers/net/wireless/marvell/mrvl8887/Kconfig"
```

Once the driver is copied correctly, it also needs `Makefile` and `Kconfig` so that the drivers are built along with the Android kernel. The two samples shown below can be used for compiling the drivers.

```
Path: android_build/vendor/nxp-
opensource/kernel_imx/drivers/net/wireless/marvell/mrvl8887/Kconfig

config MRVL8887
        tristate "Marvell WiFi Driver for SD8887"
        default m
        depends on CFG80211
        ---help---
          This adds support for wireless adapters based on Marvell
          802.11n/ac chipsets. If you choose to build it as a module,
          it will be build 2 modules sd8xxx.ko and mlan.ko
```

```
Path: android_build/vendor/nxp-
opensource/kernel_imx/drivers/net/wireless/marvell/mrvl8887/Makefile

ANDROID_CROSS_COMPILE:=$(abspath .)/prebuilts/gcc/linux-x86/arm/arm-linux-androideabi-
4.9/bin/arm-linux-androideabi-
MRVL_ANDROID_SRC_WLAN:= $(LOCAL_PATH)/wlan_src
MRVL_ANDROID_SRC_MUART:= $(LOCAL_PATH)/muart_src

default:
        $(MAKE) -j -C $(KDIR) M=$(MRVL_ANDROID_SRC_WLAN) ARCH=arm
CROSS_COMPILE=$(ANDROID_CROSS_COMPILE) SOURCE_DIR=$(MRVL_ANDROID_SRC_WLAN) modules
        $(MAKE) -j -C $(KDIR) M=$(MRVL_ANDROID_SRC_MUART) ARCH=arm
CROSS_COMPILE=$(ANDROID_CROSS_COMPILE) SOURCE_DIR=$(MRVL_ANDROID_SRC_MUART) modules

clean:
        $(MAKE) -C $(MRVL_ANDROID_SRC_WLAN) clean
        $(MAKE) -C $(MRVL_ANDROID_SRC_MUART) clean
```

☞ Some of the paths shown in the `Makefile` above can vary depending on the Android package supplied from the host platform vendor. Ensure that changes are made accordingly. Several variables, like the path for cross compiler and architecture, might not be needed in the `Makefile`. These variables are handled by default settings in the Android build system. All the compiler options are included.

When all the files are copied, the final directory should look exactly like this:

```
Path: android_build/vendor/nxp-
opensource/kernel_imx/drivers/net/wireless/marvell/mrvl8887
├── Kconfig      } New files as mentioned above for compiling drivers.
├── Makefile     }
├── muart_src
└── wlan_src
    ├── mapp
    ├── mlan
    └── mlinux
```

3.2.3 Firmware binary

Different variants of the firmware are added in the package by default. Choose one of the firmware binaries shown in Table 4.

Firmware	Wi-Fi	Bluetooth
sd8887_wlan_a2.bin	SDIO	N/A
sduart8887_combo_a2.bin	SDIO	UART
uart8887_bt_a2.bin	N/A	UART

Table 3: Firmware images used with the EMMY-W1 module

To copy the final image, firmware files must be copied to the following path:

```
Source: SDUART.8887.U16-MMC-W15.44.19.p36-15.100.19.p36-C4X15657_A2-GPL/FwImage
Destination Path: android_build/vendor/nxp/imx-firmware/mrvl/88W8887/
```

☞ Choose the firmware binary based on the usage of the module for applications. The `sduart8887_combo_a2.bin` file is recommended as it supports both Wi-Fi and Bluetooth functionality.

3.2.4 Vendor HAL

NXP provides a vendor HAL library that is integrated in the vendor-specific implementation. The library includes native commands used in the upper layers.

Copy the entire directory that includes this library from the extracted package, as shown below:

```
Source: NXP Vendor HAL - v005
Destination Path: android_build/hardware/nxp/
```

Once extracted the folder should look like below:

```
├── bt
│   └── libbt
│       ├── conf
│       │   └── bt_vendor_8887.conf
│       └── source_code
└── wlan
    ├── Android.mk
    ├── libwifi-hal
    └── wlan_lib
```

3.3 Integration with Android

3.3.1 Driver and HAL selection

Enable Marvell components to be used with the Android. This requires you to define libraries, drivers, and so on.

```
File: android_build/device/fsl/imx8m/evk_8mq/BoardConfig.mk

BOARD_WLAN_DEVICE                       := mrvl
BOARD_HAVE_BLUETOOTH_MRVL               := true
BOARD_HOSTAPD_PRIVATE_LIB               := lib_driver_cmd_mrvl
BOARD_WPA_SUPPLICANT_PRIVATE_LIB        := lib_driver_cmd_mrvl
BOARD_VENDOR_KERNEL_MODULES += \
       $(KERNEL_OUT)/drivers/net/wireless/marvell/mrvl8887/muart_src/hci_uart.ko \
       $(KERNEL_OUT)/drivers/net/wireless/marvell/mrvl8887/wlan_src/mlan.ko \
       $(KERNEL_OUT)/drivers/net/wireless/marvell/mrvl8887/wlan_src/sd8xxx.ko
```

☞ Along with defining Marvell components, ensure that configurations pertaining to other Wi-Fi and Bluetooth vendors are disabled in this file. For example, `BOARD_HAVE_BLUETOOTH_BCM`, `WIFI_DRIVER_FW_PATH_PARAM` must be disabled to avoid compilation and run-time errors.

3.3.2 Enable HAL library

Enable Marvell HAL implementation library.

```
File: android_build/frameworks/opt/net/wifi/libwifi_hal/Android.mk

LIB_WIFI_HAL := libwifi-hal-mrvl
```

3.3.3 File placement

The host platform specific `Makefile` in the Android source code must include the required changes to copy the files to the right directories.

```
File: android_build/device/fsl/imx8m/evk_8mq/evk_8mq.mk

# Wi-Fi and Bluetooth Firmware
PRODUCT_COPY_FILES += vendor/nxp/imx-
firmware/mrvl/88W8887/sduart8887_combo_a2.bin:$(TARGET_COPY_OUT_VENDOR)/firmware/mrvl/
sduart8887_combo_a2.bin

# WiFi Marvell HAL libraries
PRODUCT_PACKAGES += \
    libbt-vendor \
    lib_driver_cmd_mrvl

# Supplicant config files
PRODUCT_COPY_FILES += \
    device/fsl/common/wifi/wpa_supplicant.conf:vendor/etc/wifi/wpa_supplicant.conf \
  device/fsl/common/wifi/p2p_supplicant.conf:vendor/etc/wifi/p2p_supplicant.conf
```

3.3.4 Additional packages

The Wi-Fi hardware abstraction layer (HAL) provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework. It includes the following HIDL packages:

- Vendor HAL
- Supplicant HAL

When a framework API makes a call to access device hardware interface, the Android system loads the library module for that hardware component. These abstraction layers are already part of the AOSP and are chosen as part of the manifest changes in section 3.2.1.

The following packages are needed for the execution of the complete Wi-Fi stack:

- **android.hardware.wifi@1.0-service**: This is the root of the HAL module and is the service invoked by the native framework.
- **Wifilogd**: This is a logging module used for debug and analysis
- **Wificond**: It communicates with the Wi-Fi driver over standard `nl80211` commands.

Add the appropriate components into the host platform specific file, as shown below:

```
File: android_build/device/fsl/imx8m/evk_8mq/evk_8mq.mk

# Wi-Fi additional packages
PRODUCT_PACKAGES += \
    android.hardware.wifi@1.0-service \
    wifilogd \
    wificond
```

3.3.5 Define interfaces

Set the global interfaces for Wi-Fi operation during host platform initialization.

```
File: android_build/device/fsl/imx8m/evk_8mq/init.rc

    setprop wlan.driver.status ok
    setprop wifi.interface wlan0
    setprop wifi.direct.interface p2p0
```

3.3.6 Starting supplicant

Add a service to invoke the supplicant when triggered by upper layers.

```
File: android_build/device/fsl/imx8m/evk_8mq/init.rc

service wpa_supplicant /vendor/bin/hw/wpa_supplicant \
    -ip2p0 -Dnl80211 -c/vendor/etc/wifi/p2p_supplicant.conf \
    -puse_multi_chan_concurrent=1 \
    -N -iwlan0 -Dnl80211 -c/vendor/etc/wifi/wpa_supplicant.conf \
    -puse_multi_chan_concurrent=1 \
    -C /data/vendor/wifi/wpa/sockets \
    -e/data/misc/wifi/entropy.bin -g@android:wpa_wlan0
    class main
    socket wpa_wlan0 dgram 660 wifi wifi
    user root
    group root
    disabled
    oneshot
```

As shown in the command above, for wpa_supplicant to run, it needs couple of configuration files. Contents of these files are as below. These files are copied into the root file system as specified in the section 3.3.3.

```
Path: android_build/device/fsl/common/wifi/wpa_supplicant.conf

update_config=1
eapol_version=1
ap_scan=1
fast_reauth=1
pmf=1
p2p_add_cli_chan=1
```

```
Path: android_build/device/fsl/common/wifi/p2p_supplicant.conf

update_config=1
eapol_version=1
ap_scan=1
fast_reauth=1
pmf=1
p2p_add_cli_chan=1
p2p_no_group_iface=1
```

3.3.7 Loading the Wi-Fi driver

Drivers are loaded using an `insmod` script which references a configuration file with paths to the kernel modules. Both the script and configuration are added in the file excerpt shown below.

```
Path: android_build/device/fsl/common/init/init.insmod.sh

#! /vendor/bin/sh

# cfg file format:
# [path for modules along with parameters]

cfg_file=$1

if [ -f $cfg_file ]; then
  while IFS=" " read -r line
  do
    insmod $line
  done < $cfg_file
fi

# set property even if there is no insmod config
# as property value "1" is expected in early-boot trigger
setprop $2 1
```

Configuration file for the `insmod` script.

```
Path: android_build/device/fsl/imx8m/evk_8mq/early.init.cfg

vendor/lib/modules/mlan.ko
vendor/lib/modules/sd8xxx.ko fw_name=mrvl/sduart8887_combo_a2.bin cfg80211_wext=0xf
sta_name=wlan wfd_name=p2p p2p_enh=1 drv_mode=0x5 cal_data_cfg=none
vendor/lib/modules/hci_uart.ko
```

With the script and the configuration files now included, they must now be invoked in `init` time.

```
Path: android_build/device/fsl/imx8m/evk_8mq/init.rc

on early-init
    start early_init_sh

service early_init_sh /vendor/bin/init.insmod.sh /vendor/etc/early.init.cfg
sys.all.early_init.ready
    class main
    user root
    group root system
    disabled
    oneshot
```

3.3.8 Patching the SE Linux policy

To edit files on the root file system or use any device node, explicit permissions must be given to the vendor HAL layers. You set these permissions by applying `policy_changes_androidp.patch`, as shown below.

```
Path: android_build/device/fsl/imx8m/

Command: git apply policy_changes_androidp.patch
```

The patch applies the appropriate permissions for both Wi-Fi and Bluetooth.

☞ Contact your local u-blox support team for the patch file.

4 Bluetooth integration

4.1 Bluetooth architecture

To enable Bluetooth, you must execute the appropriate set of configuration and files. The logical functions of the stack in relation to the Bluetooth architecture are shown in Figure 5.

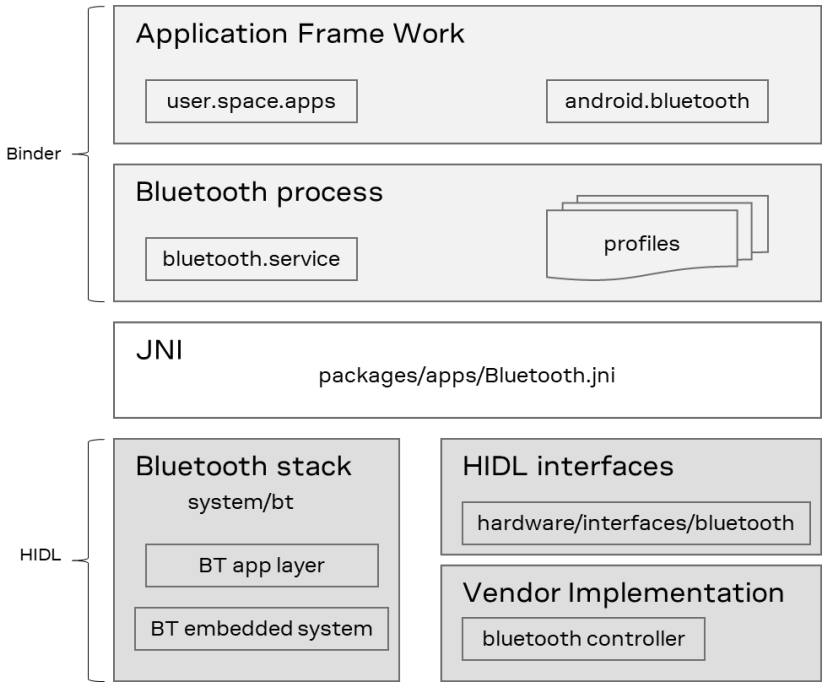


Figure 5: Bluetooth architecture

4.2 Components

4.2.1 Driver source

AOSP includes the HCI UART as an embedded driver in the kernel. NXP provides a modified version of this code, which is needed for the EMMY-W1 series modules. To enable HCI UART driver you need to include some changes in the code parameters. These parameters can include different compile options, like `m` or y, and need to be set as shown in the snippet below. Note that the configuration parameter option CONFIG_USB_SERIAL_FTDI_SIO is only necessary if you are accessing UART interface over FTDI.

```
File: android_build/vendor/nxp-opensource/kernel_imx/arch/arm/configs/
imx_v7_android_defconfig

CONFIG_BT_HCIUART=m
CONFIG_USB_SERIAL=y
CONFIG_USB_SERIAL_FTDI_SIO=y
```

The build and compilation of the proprietary HCI UART driver is described in section 3.2.2.1. Section 3.3.1 describes how the drivers are built into a specific location for loading during the initialization phase of the Android configuration.

4.3 Integration with Android

As described earlier, NXP_Vendor_hal_005 contains the vendor library for Bluetooth. To build this together with the other modules it must be copied to the correct path as shown below.

```
Path: android_build/hardware/marvell/
      ├── libbt ➔ Vendor HAL for Bluetooth
      └── wlan
```

4.3.1 Makefile changes

To specify the vendor for Bluetooth, include the following variables into the `BoardConfig.mk` file.

```
Path: android_build/device/fsl/imx8m/evk_8mq/BoardConfig.mk
BOARD_HAVE_BLUETOOTH := true
BOARD_HAVE_BLUETOOTH_MRVL := true
```

Ensure that Bluetooth configurations from other vendors are removed from `BoardConfig.mk`.

4.3.2 BT interface configuration

Default values needed for BT interface are defined in the libbt. These parameters can be changed in the application design. In which case, these are specified in the `bt_vendor.conf` file pertaining to the module. A file example is shown below.

```
Path: android_build/hardware/marvell/bt/libbt-vendor/conf/bt_vendor_8887.conf
mchar_port = /dev/ttyUSB0
is_uart_port = 1
baudrate_bt = 115200
baudrate_fw_init = 3000000
bd_address = 12:34:56:78:9a:bc
```

Several of the parameters shown above can vary between different u-blox modules, namely:

- `mchar_port` is needed when UART is used as an interface for BT communication. Along with it, is_`uart_port` must also to be set. Configuration of the SDIO interface is described in section 5.2.3.
- `baudrate_bt` and `baudrate_fw_init` must be carefully chosen to suit the design of the module. The values used here should match with the baud rate programmed in the module's OTP memory. Otherwise, communication fails regardless of whether the file descriptor opens successfully.

4.3.3 SE-Linux labelling

Assign permission to the device node /dev/ttyUSB0. Although permissions are modified with `chmod` at the start of the init, Android policy prevents file access.

Explicit file permissions can be configured, as shown below:

```
File: android_build/device/fsl/imx8m/evk_8mq/sepolicy/file_contexts
/dev/ttyUSB0            u:object_r:tty_device:s0
```

4.3.4 Permission settings

Bluetooth is connected over UART using USB. Make the necessary permission changes so that the Android stack can access the port for each read-write.

```
File: android_build/device/fsl/imx8m/evk_8mq/init.rc

setprop hw.bluetooth 1
# /dev/ttyUSB0 is the connected port on i.MX8 EVK
chmod 0666 /dev/ttyUSB0
```

☞ In the above example, `/dev/ttyUSB0` is the port to connect the UART interface over USB. This can be arranged differently depending on the type of connection to the module. The port in this configuration must be in sync with the port configuration file of the vendor

4.3.5 Additional packages

AOSP provides a default Bluetooth stack that supports both Bluetooth BR/EDR and Bluetooth Low Energy. The `android.hardware.bluetooth` package provides the interface required between the Bluetooth controller and stack.

To enable the package, include the variables shown below into the build configuration:

```
File: android_build/device/fsl/imx8m/evk_8mq/evk_8mq.mk
# Bluetooth HAL
PRODUCT_PACKAGES += \
    libbt-vendor \
    android.hardware.bluetooth@1.0-impl \
    android.hardware.bluetooth@1.0-service
```

Check with your system platform provider to ensure that an `rfkill` driver for Bluetooth on/off control has been implemented on your platform.

The platform BSP should implement an `rfkill` driver that allows the Android Bluetooth stack to turn on Bluetooth control through `/sys/class/rfkill/rfkill<id>/state`.

Configure the `rfkill<id>` state to set Bluetooth (on/off) control. Set the `rfkill<id>` permissions in the system `init.rc file` to ensure that the Bluetooth process can access it.

```
File: android_build/device/fsl/imx8m/evk_8mq/init.rc

# Configure bluetooth
chmod 0660 /sys/class/rfkill/rfkill0/state
chown bluetooth bluetooth /sys/class/rfkill/rfkill0/state
```

5 Integrating SDIO-SDIO driver

This section explains the explicit changes required to integrate the SDIO-SDIO variant of the EMMY-W1 driver used with Android 9.

5.1 Driver sources and compilation

To operate Bluetooth over SDIO interface `mbtchar0` device node is used. The `mbtc` (SD-BT-CHAR) driver is part of the `SD-WLAN-SD-BT-8887-U16-MMC-W15.68.19.p38-15.26.19.p38-C4X15659_A2-MGPL.zip` package from NXP.

The various components of this driver are listed below.

```
SD-WLAN-SD-BT-8887-U16-MMC-W15.68.19.p38-15.26.19.p38-C4X15659_A2-MGPL (Mar 2020)
├── FwImage                      sd8887_uapsta_a2.bin is used for operating
│   ├── sd8887_uapsta_a2.bin     Wi-Fi and Bluetooth modules in Android. Rest
│   ├── sd8887_wlan_a2.bin       are needed for individual radios.
│   └── sd8887_bt_a2.bin
├── SD-UAPSTA-8887-U16-MMC-W15.68.19.p38-C4X15659_A2-app-src.tgz
├── SD-UAPSTA-8887-U16-MMC-W15.68.19.p38-C4X15659_A2-MGPL-src.tgz
├── SD-UAPSTA-8887-U16-MMC-W15.68.19.p38-C4X15659_A2-mlan-src.tgz
└── SD-BT-CHAR-8887-U16-MMC-15.26.19.p38-C4X14113_A2-GPL-src.tgz
```

Extract and copy the sources in the directories as mentioned below:

```
Destination Path: android_build/vendor/nxp-
opensource/kernel_imx/drivers/net/wireless/marvell/mrvl8887
├── mbtc_src       → Copy the contents of SD-BT-CHAR-8887-XX into this folder
└── wlan_src       → Copy the contents of SD-UAPSTA-8887-XX into this folder
    ├── Makefile
    ├── mapp       → Applications for configurations
    ├── mlan       ┐ Driver code for Wi-Fi module.
    └── mlinux     ┘ 2 modules are built out of this source sd8xxx, mlan.
```

Once the sources are copied they need to be selected to be compiled as mentioned below.

```
Path: android_build/vendor/nxp-opensource/kernel_imx/drivers/
/net/wireless/marvell/Makefile
obj-$(CONFIG_MRVL8887) += mrvl8887/wlan_src/
obj-$(CONFIG_MRVL8887) += mrvl8887/mbtc_src/

Path: android_build/vendor/nxp-
opensource/kernel_imx/drivers/net/wireless/marvell/Kconfig
source "drivers/net/wireless/marvell/mrvl8887/Kconfig"
```

☞ Note that `CONFIG_MRVL8887` must be enabled before selecting the `mrvl8887` folder in the Android kernel sources, as described in section 3.2.2.1. `Makefile` needs some modification to pick up the right `mbtc` driver path as shown in the snippet below. `Kconfig` remains same.

```
Path: android_build/vendor/nxp-
opensource/kernel_imx/drivers/net/wireless/marvell/mrvl8887/Makefile

ANDROID_CROSS_COMPILE:=$(abspath .)/prebuilts/gcc/linux-x86/arm/arm-linux-androideabi-
4.9/bin/arm-linux-androideabi-
MRVL_ANDROID_SRC_WLAN:= $(LOCAL_PATH)/wlan_src
MRVL_ANDROID_SRC_MBTC:= $(LOCAL_PATH)/mbtc_src

default:
        $(MAKE) -j -C $(KDIR) M=$(MRVL_ANDROID_SRC_WLAN) ARCH=arm
CROSS_COMPILE=$(ANDROID_CROSS_COMPILE) SOURCE_DIR=$(MRVL_ANDROID_SRC_WLAN) modules
        $(MAKE) -j -C $(KDIR) M=$(MRVL_ANDROID_SRC_MBTC) ARCH=arm
CROSS_COMPILE=$(ANDROID_CROSS_COMPILE) SOURCE_DIR=$(MRVL_ANDROID_SRC_MBTC) modules

clean:
        $(MAKE) -C $(MRVL_ANDROID_SRC_WLAN) clean
        $(MAKE) -C $(MRVL_ANDROID_SRC_MBTC) clean
```

5.2 Integration with Android

5.2.1 File placement

Copy the firmware in the designated location. The firmware binaries should be copied in the Android source as mentioned in the section 3.2.3.

```
File: android_build/device/fsl/imx8m/evk_8mq/evk_8mq.mk

# Wi-Fi and Bluetooth Firmware
PRODUCT_COPY_FILES += vendor/nxp/imx-
firmware/mrvl/88W8887/sd8887_uapsta_a2.bin:$(TARGET_COPY_OUT_VENDOR)/firmware/mrvl/sd8
887_uapsta_a2.bin
```

Include the drivers. Additionally mbt8xxx.ko is needed for Bluetooth.

```
File: android_build/device/fsl/imx8m/evk_8mq/BoardConfig.mk

BOARD_VENDOR_KERNEL_MODULES += \
      $(KERNEL_OUT)/drivers/net/wireless/marvell/mrvl8887/mbtc_src/mbt8xxx.ko \
      $(KERNEL_OUT)/drivers/net/wireless/marvell/mrvl8887/wlan_src/mlan.ko \
      $(KERNEL_OUT)/drivers/net/wireless/marvell/mrvl8887/wlan_src/sd8xxx.ko
```

5.2.2 Loading the driver

Follow the instructions in section 3.3.7 for loading the driver. The configuration file for the SDIO drivers changes as below.

```
Path: android_build/device/fsl/imx8m/evk_8mq/early.init.cfg

vendor/lib/modules/mlan.ko
vendor/lib/modules/sd8xxx.ko fw_name=mrvl/sd8887_uapsta_a2.bin cal_data_cfg=none
cfg80211_wext=0xf sta_name=wlan wfd_name=p2p p2p_enh=1 drv_mode=0x5
vendor/lib/modules/mbt8xxx.ko
```

5.2.3 Configuring the Bluetooth interface

The default Bluetooth SD/USB port in `libbt` is `/dev/mbtchar0`. If the query response is not the same as default port, could set `mbt_port` in conf file like below. For further details, see section 4.3.2.

```
Path: android_build/hardware/marvell/bt/libbt-vendor/conf/bt_vendor_8887.conf

mbt_port = /dev/mbtchar0
bd_address = 12:34:56:78:9a:bc
```

5.2.4 SE-Linux labelling

Explicit permission can be set to the device node *mbtchar0,* as shown below:

```
File: android_build/device/fsl/imx8m/evk_8mq/sepolicy/file_contexts
/dev/mbtchar0            u:object_r:tty_device:s0
```

5.2.5 Permission settings

The SDIO driver for Bluetooth creates a separate device node *mbtchar0* for operating with the hardware. For the vendor HAL to communicate to the driver, some permissions in the init.rc file need to be set. Use the chmod command to set the permissions, as shown below.

```
File: android_build/device/fsl/imx8m/evk_8mq/init.rc
# /dev/mbtchar0 is the device node created
chmod 0666 /dev/mbtchar0
```

☞ Before enabling Bluetooth, make sure that this device node is created.

5.2.6 Patch the Bluetooth code

Patch the system Bluetooth using the snippet below.

```
diff --git a/bluetooth/1.0/default/h4_protocol.cc
b/bluetooth/1.0/default/h4_protocol.cc
index df40507..73cb1fd 100644
--- a/bluetooth/1.0/default/h4_protocol.cc
+++ b/bluetooth/1.0/default/h4_protocol.cc
@@ -30,20 +30,13 @@ namespace bluetooth {
 namespace hci {

 size_t H4Protocol::Send(uint8_t type, const uint8_t* data, size_t length) {
-  struct iovec iov[] = {{&type, sizeof(type)},
-                        {const_cast<uint8_t*>(data), length}};
-  ssize_t ret = 0;
-  do {
-    ret = TEMP_FAILURE_RETRY(writev(uart_fd_, iov, sizeof(iov) / sizeof(iov[0])));
-  } while (-1 == ret && EAGAIN == errno);
+  // TODO(bcf): Fix the driver to handle this split into multiple writes.
+  std::vector<uint8_t> tmp;
+  tmp.reserve(1 + length);
+  tmp.push_back(type);
+  tmp.insert(tmp.end(), data, data + length);

-  if (ret == -1) {
-    ALOGE("%s error writing to UART (%s)", __func__, strerror(errno));
-  } else if (ret < static_cast<ssize_t>(length + 1)) {
-    ALOGE("%s: %d / %d bytes written - something went wrong...", __func__,
-          static_cast<int>(ret), static_cast<int>(length + 1));
-  }
-  return ret;
+  return WriteSafely(uart_fd_, tmp.data(), tmp.size());
 }

 void H4Protocol::OnPacketReady() {
```

6 Flashing the image

Use UUU (Universal Update Utility) to flash the image to the eMMC of the i.MX8 MQ EVK.

6.1 Download and build UUU

For UUU source code and other information about this utility, see https://github.com/NXPmicro/mfgtools.

Follow the procedure outlined below to build the utility:

```
git clone https://github.com/NXPmicro/mfgtools.git
cd mfgtools
sudo apt-get install libusb-1.0-0-dev libzip-dev libbz2-dev pkg-config cmake libssl-dev
cmake .
make
```

Once the utility has been built, copy the binary uuu to any default binary path used in the system like `/usr/sbin/` for example.

6.2 Flashing the eMMC using Linux machine

Set the DIP switches to select the eMMC device to flash, as shown below.

Switch	D1	D2	D3	D4
SW801	OFF	OFF	ON	OFF
SW802	OFF	ON	N/A	N/A

Table 4: eMMC DIP pre-flash device switch settings

Connect the EVK to the Linux machine using a type-C USB cable. The required script for flashing is present in the build folder.

```
Path: android_build/out/target/product/evk_8mq/uuu_imx_android_flash.sh
```

Run the following `sh` command to flash the eMMC :

```
Command: sh uuu_imx_android_flash.sh -f imx8mq -a -e
```

Some of the arguments to the script can vary in later versions of Android BSP or UUU. To display the help menu, simply run the script without defining any input.

In some cases the script might get stuck during execution. This typically happens while flashing the eMMC for the first time. In these instances, run the following command to restart the process.

```
Command: fastboot reboot
```

When the flash is complete, set the DIP switches as they are shown below and switch on the EVK.

Switch	D1	D2	D3	D4
SW801	OFF	OFF	ON	OFF
SW802	ON	OFF	N/A	N/A

Table 5: eMMC DIP post-flash device switch settings

6.3 Flashing the eMMC on Windows PC

Connect the EVK to the Windows machine using a type-C USB cable.

```
Path: android_build/out/target/product/evk_8mq/uuu_imx_android_flash.bat
```

Run the batch script to flash the eMMC:

```
Command: .\uuu_imx_android_flash.bat -f imx8mq -a -e
```

Before running this command, some more utilities are needed on Windows. For more information, see the WIN7 User Guide for MFGTools [13].

7 Validation

After the image has been flashed and the host platform has booted, check that the following files, required for Wi-Fi, are present in the path locations shown below:

Type	File name	Path
Firmware	sduart8887_combo_a2.bin	/vendor/firmware/mrvl/
Drivers	mlan.ko, sd8xxx.ko, hci_uart.ko	/vendor/lib/modules/
SUPPLICANT	wpa_supplicant	/vendor/bin/hw/
CLIENT	wpa_cli	/vendor/bin/
WPA CONF file	wpa_supplicant.conf	/vendor/etc/wifi/
P2P CONF file	p2p_supplicant.conf	/vendor/etc/wifi/
HOSTAPD	hostapd	/vendor/bin/hw/

Table 6: List of files

7.1 Wi-Fi bring up

Wi-Fi can be used in the following modes:

- Station mode
- Access point mode
- Wi-Fi Direct mode

All the modes have been tested and verified to work with Android 9 (Android Pie) and the EMMY-W1 module. After all components have been successfully built and installed, as described in sections 3 and 4, the following kernel messages are output in the kernel log during the host boot.

Check the post-boot log copy of the message buffer in the `/var/log/dmesg` log to ensure that the driver and firmware are loaded correctly.

```
evk_8mq:/ # dmesg
mlan: module license 'Marvell Proprietary' taints kernel.
Disabling lock debugging due to kernel taint
Registered swp emulation handler
Console: switching to colour dummy device 80x25
watchdogd: watchdogd started (interval 10, margin 20)!
wlan: Loading MWLAN driver
vendor=0x02DF device=0x9135 class=0 function=1
SDIO: max_segs=128 max_seg_size=65535
rx_work=1 cpu_num=4
wlan: Enable TX SG mode
wlan: Enable RX SG mode
Request firmware: mrvl/sduart8887_combo_a2.bin
random: crng init done
Wlan: FW download over, firmwarelen=621692 downloaded 621692
usb 1-1.4: new low-speed USB device number 6 using xhci-hcd
WLAN FW is active
fw_cap_info=0x187bff03, dev_cap_mask=0xffffffff
SDIO rx aggr: 1 block_size=412
wlan: Enable RX SG mode
mpa_rx_buf_size=65280
wlan: version = SD8887-15.44.19.p36-C4X15C657-GPL-(FP44)
wlan: Driver loaded successfully
HCI UART driver ver 2.2-M2614100
```

Check that all the drivers are loaded correctly using `lsmod` command.

```
evk_8mq:/ # lsmod
Module                  Size  Used by
hci_uart               49152  0
sd8xxx                626688  0
mlan                  471040  1 sd8xxx
```

The expected interfaces after the host platform boots are shown below:

```
evk_8mq:/ # ifconfig
wlan0     Link encap:Ethernet  HWaddr d4:ca:6e:00:1b:16  Driver wlan_sdio
          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 TX bytes:0
p2p0      Link encap:Ethernet  HWaddr d6:ca:6e:00:1b:16  Driver wlan_sdio
          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 TX bytes:0
```

Enable the Wi-Fi using the command below. This translates to switching on the Wi-Fi in the GUI.

```
svc wifi enable
```

Use the `logcat` command to verify the driver and firmware (shown as bold in the log example below) are accessible to the android framework.

```
evk_8mq:/ # logcat
WificondControl: Setting up interface for client mode
wificond: subscribe scan result for interface with index: 7
hwservicemanager: getTransport: Cannot find entry
SupplicantStaIfaceHal: Can't call setupIface, ISupplicantStaIface is null
android_os_HwBinder: HwBinder: Starting thread pool for
default::android.hardware.wifi.supplicant@1.0::ISupplicant
WifiNative: Interface state changed on Iface:{Name=wlan0,Id=0,Type=STA}, isUp=true
WifiNative: Successfully setup Iface:{Name=wlan0,Id=0,Type=STA}
WifiClientModeManager: sending scan available broadcast: false
WifiClientModeManager: Wifi is ready to use for client mode
WifiClientModeManager: sending scan available broadcast: true
WifiStateMachinePrime: State changed from client mode. state = 3
WifiStateMachinePrime: client mode active
WifiStateMachine: entering ConnectModeState: ifaceName = wlan0
WifiStateMachine: setupClientMode() ifacename = wlan0
WifiP2pService: Wifi enabled=true, P2P Interface availability=true, Number of clients=0
WifiP2pService: Wifi enabled=true, P2P Interface availability=true
wpa_supplicant: the nl80211 driver cmd is MACADDR
wpa_supplicant: the nl80211 driver cmd len is 7
WifiScanningService: wifi driver loaded with scan capabilities: max buckets=16
WifiStateMachine: Setting OUI to DA-A1-19
WifiVendorHal: Driver: 15.44.19.p36 Firmware: 15.44.19.p36
...
```

7.2 Station mode

Using the system settings on the host platform, scan and connect to the APs in the environment.

Settings > Wi-Fi & internet > Wi-Fi

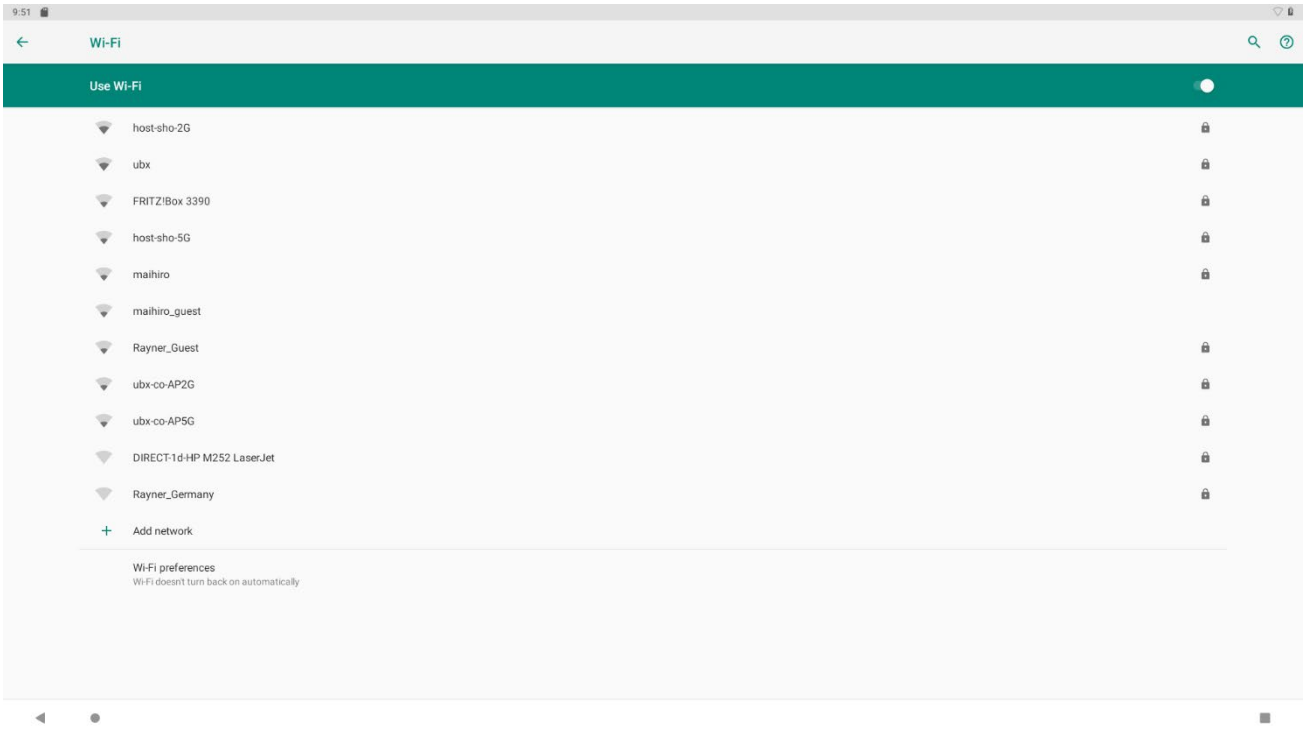


Figure 6: Wi-Fi scanning

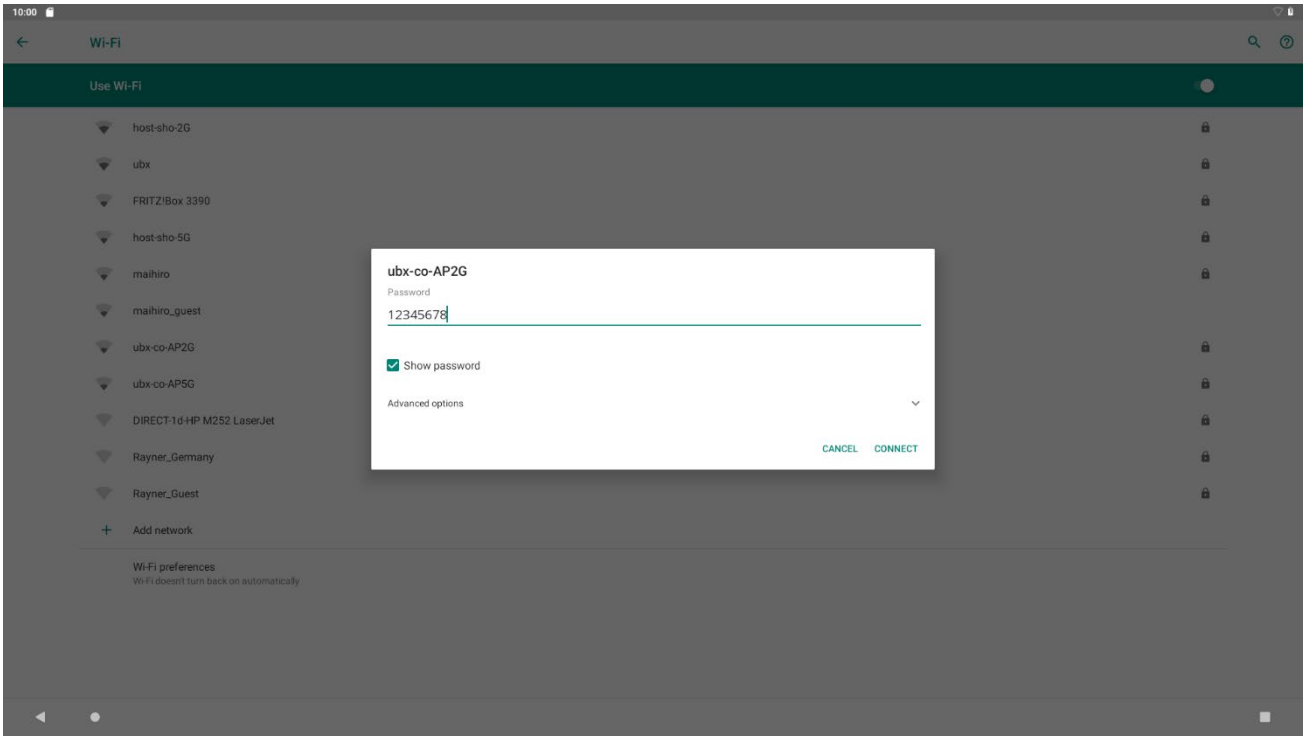


Figure 7: Wi-Fi connecting to a secured AP (WPA2-PSK)

7.3 Access point mode

Access point mode (AP) is used for tethering the internet bandwidth from cellular with the other peers in the environment. This mode is also used for sharing big data files between the devices.

By default, AP operates at 2.4 GHz. You can choose for AP operation at 5 GHz in the location settings. To enable 5 GHz operation with Android enable `CFG80211_INTERNAL_REGDB`, as shown below:

```
Path: android_build/vendor/nxp-opensource/kernel_imx/net/wireless/Kconfig
config CFG80211_INTERNAL_REGDB
        bool "use statically compiled regulatory rules database" if EXPERT
        default y
        depends on CFG80211
```

Copy the `db.txt` file from the online repository [14] to the same location. A country database must also be provided with this when building the kernel. The database file, `db.txt` contains the list of channels supported in different countries. For further information about the `wireless-regdb` database, check the official Linux Wireless Wiki [7].

To configure the Wi-Fi hotspot settings shown in Figure 8, select:

Settings > Connections > Network & internet> Hotspot & tethering > Wi-Fi hotspot

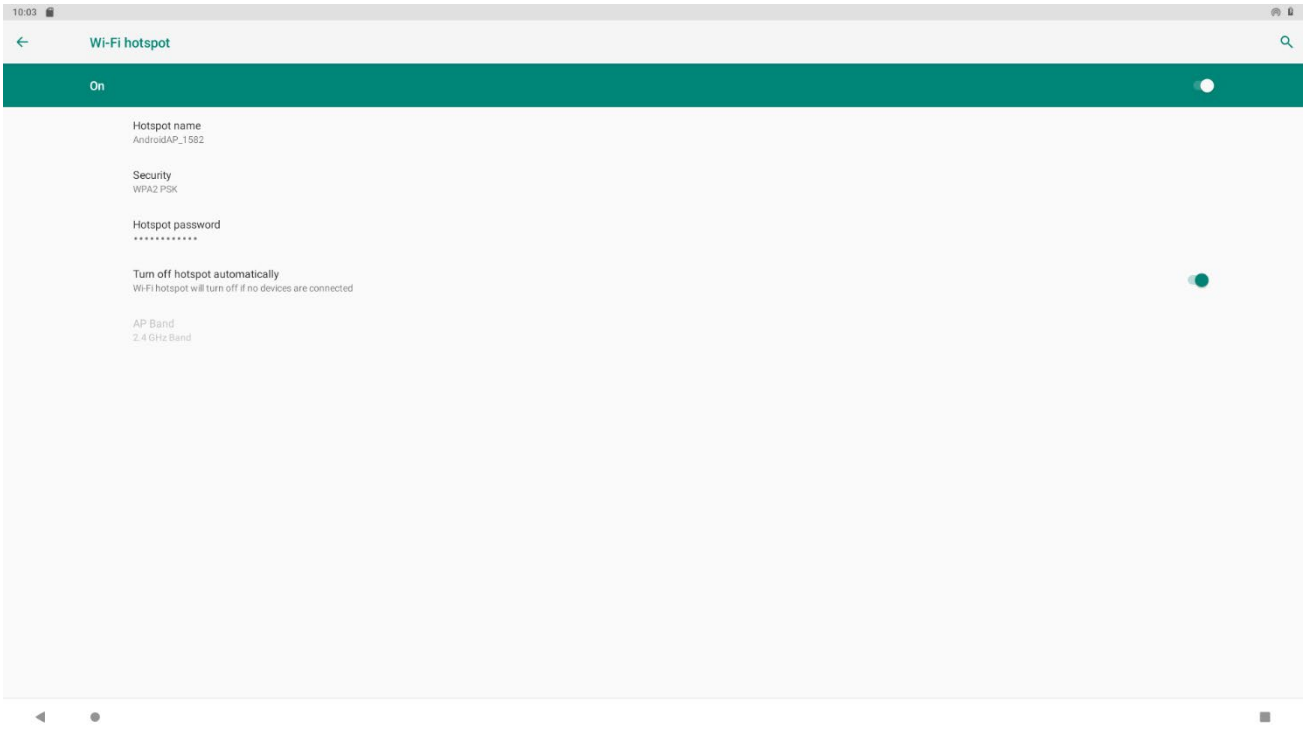


Figure 8: Wi-Fi hotspot creation

7.4 Throughput measurements

In general `iperf` is a reliable utility for measuring network throughput. On Android, an application can be used for the purpose of measurement. “iPerf for Android” is a tool used for running different versions of `iperf(2/3)`. For further information about the `iperf` tool and APK, see reference [15].

7.5 Wi-Fi Direct

The Wi-Fi Direct standard used to connect Wi-Fi devices to one another is used in in a variety of different use cases, like Wi-Fi display that works between two peers.

To enable Wi-Fi Direct, select:

Settings > Connections > Wi-Fi > Wi-Fi preferences > Wi-Fi Direct

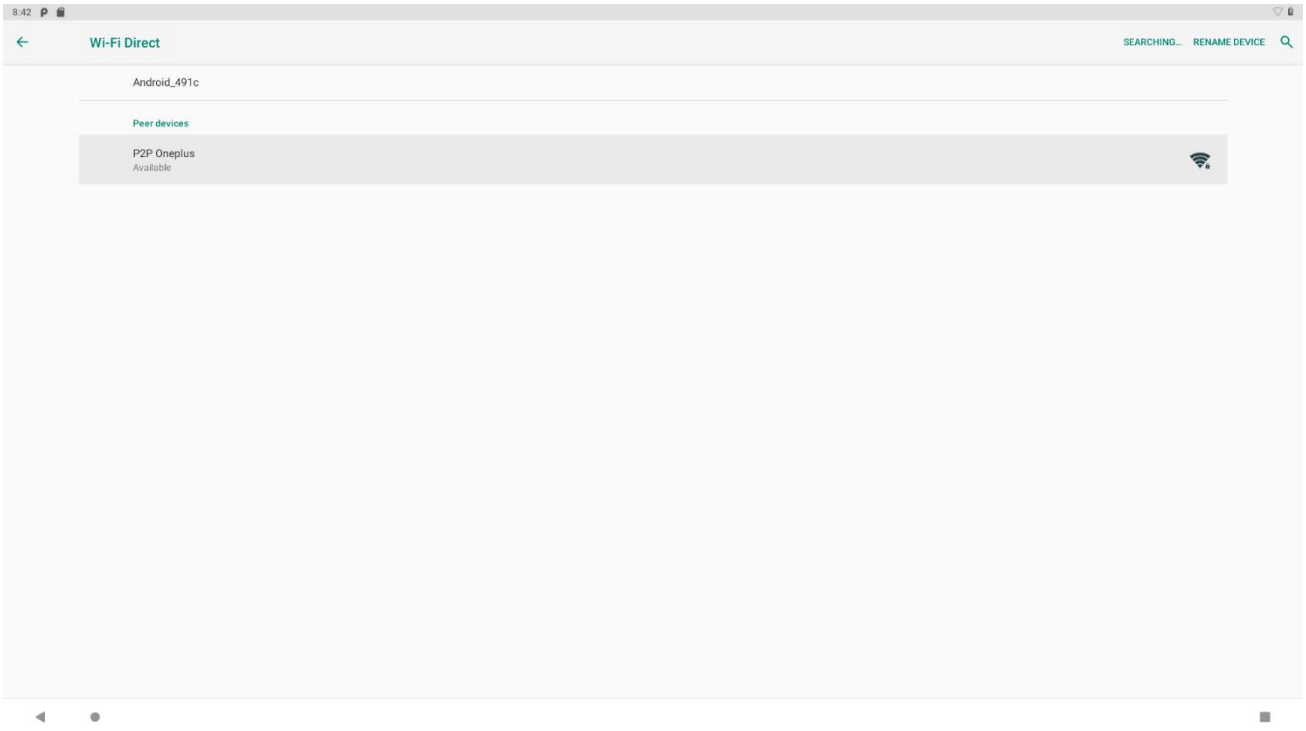


Figure 9: Wi-Fi Direct scanning

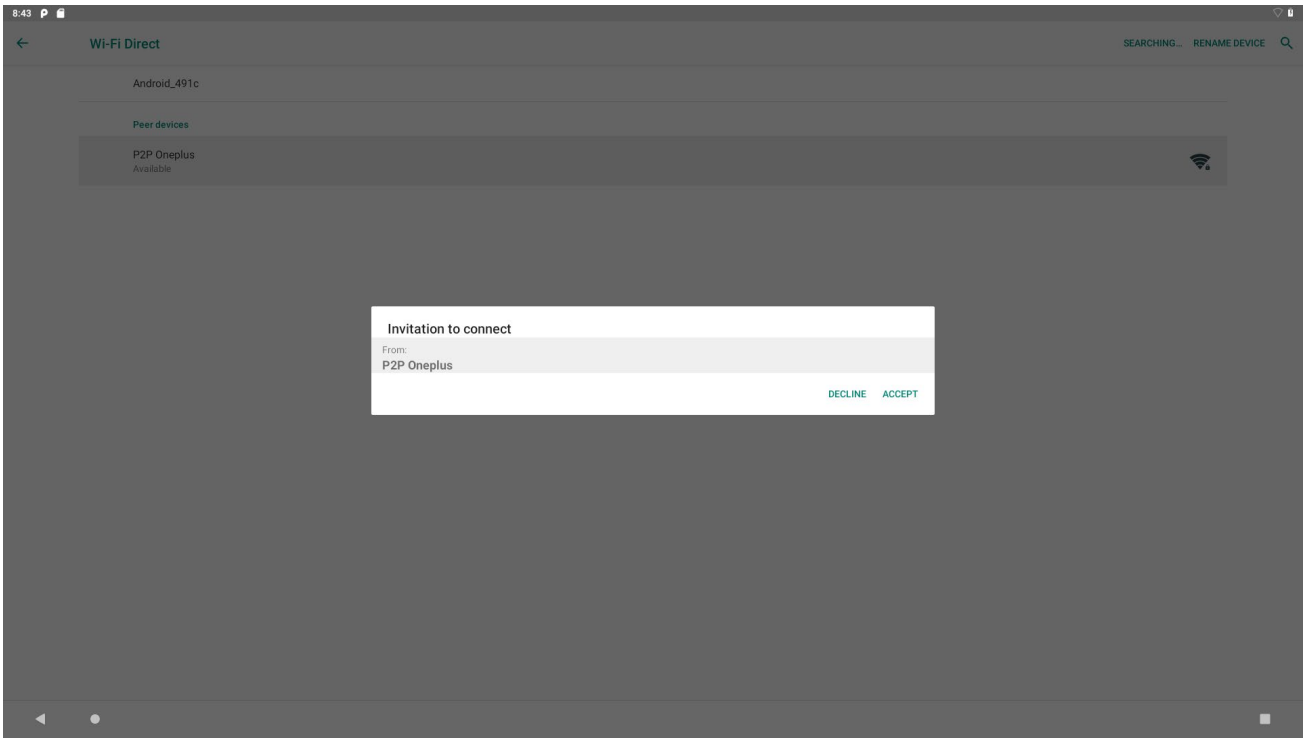


Figure 10: Wi-Fi Direct accept

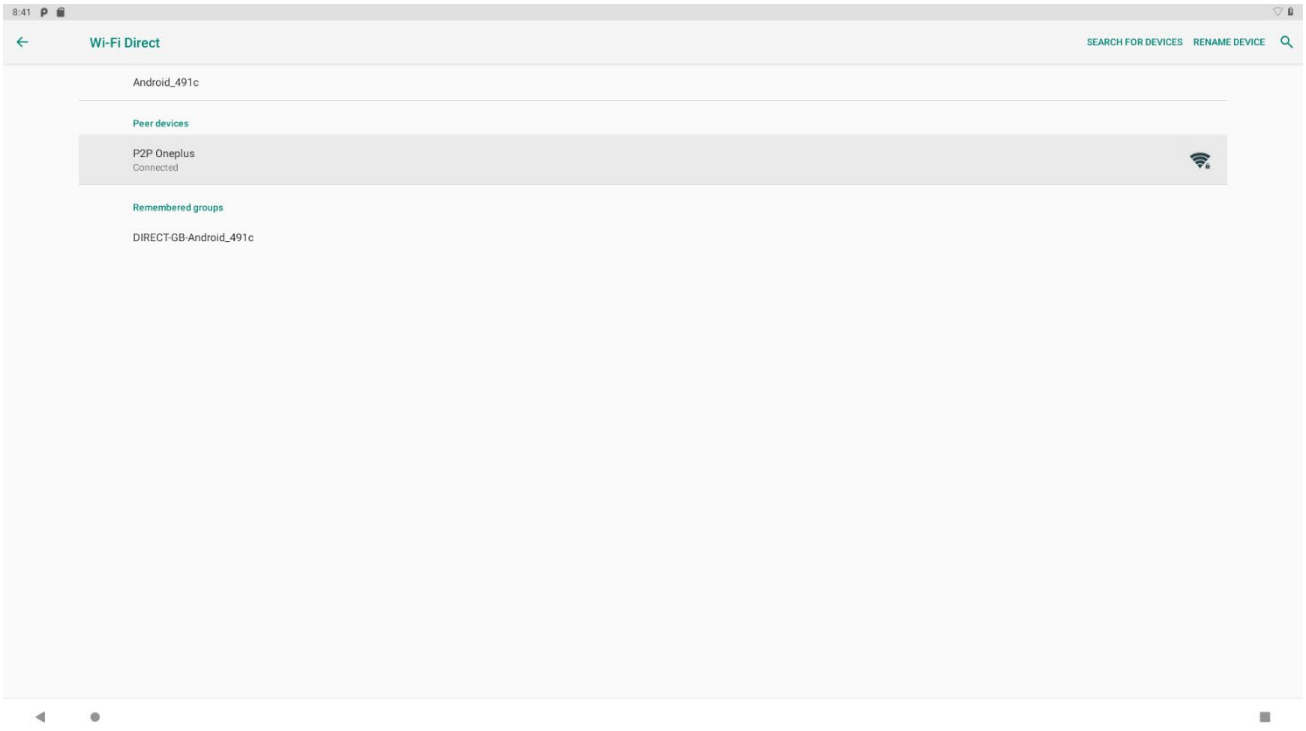


Figure 11: Wi-Fi Direct connected state

☞ Wi-Fi display functionality is not yet supported in the Android image supplied by NXP. At the moment, this functionality requires some additional application level libraries with the Wi-Fi direct protocol.

7.6 Bluetooth bring up

Once the files are placed in the expected paths, Bluetooth will be enabled. Android services can also be used to enable Bluetooth using the below-mentioned command.

```
# Enable Bluetooth via command line
service call bluetooth_manager 6
# Disable Bluetooth via command line
service call bluetooth_manager 8
```

To otherwise enable Bluetooth from the GUI, select:

Settings > Bluetooth & device connection > Bluetooth

Some logs from the Android system when Bluetooth is successfully enabled are provided below:

```
bt_stack_manager: event_start_up_stack is bringing up the stack
bt_core_module: module_start_up Starting module "btif_config_module"
bt_core_module: module_start_up Started module "btif_config_module"
bt_core_module: module_start_up Starting module "btsnoop_module"
bt_core_module: module_start_up Started module "btsnoop_module"
bt_core_module: module_start_up Starting module "hci_module"
bt_hci  : hci_module_start_up
bt_osi_thread: run_thread: thread id 1896, thread name hci_thread started
bt_hci  : hci_initialize
bt_hci  : hci_module_start_up starting async portion
bt_hci  : hci_initialize: IBluetoothHci::getService() returned 0xa4086ce0 (remote)
android.hardware.bluetooth@1.0-impl: BluetoothHci::initialize()
android.hardware.bluetooth@1.0-impl: Open vendor library loaded
bt-vnd-mrvl: bt_vnd_mrvl_if_op(L414): opcode = 0
bt-vnd-mrvl: bt_vnd_mrvl_if_op(L426): power on ---------------------------------------
bt-vnd-mrvl: bt_vnd_mrvl_if_op(L414): opcode = 3
bt-vnd-mrvl: bt_vnd_mrvl_if_op(L443): open serial port--------------------------------
bt-vnd-mrvl: bt_vnd_mrvl_if_op(L449): baudrate_bt 3000000
bt-vnd-mrvl: bt_vnd_mrvl_if_op(L450): baudrate_default 115200
android.hardware.bluetooth@1.0-impl: OnFirmwareConfigured result: 0
android.hardware.bluetooth@1.0-impl: Firmware configured in 0.000s
bt-vnd-mrvl: bt_vnd_mrvl_if_op(L414): opcode = 5
android.hardware.bluetooth@1.0-impl: OnFirmwareConfigured: lpm_timeout_ms 0
bt-vnd-mrvl: bt_vnd_mrvl_if_op(L414): opcode = 6
android.hardware.bluetooth@1.0-impl: low_power_mode_cb result: 0
android.hardware.bluetooth@1.0-impl: OnFirmwareConfigured Calling
StartLowPowerWatchdog()
bt_hci  : event_finish_startup
bt_core_module: module_start_up Started module "hci_module"
```

7.7 Bluetooth verification

Basic Bluetooth involves a discovery scan followed by connection between the two devices. The same connection can be used for different applications such as file sharing or audio streaming. File transfer using images between the devices works successfully and has been verified.

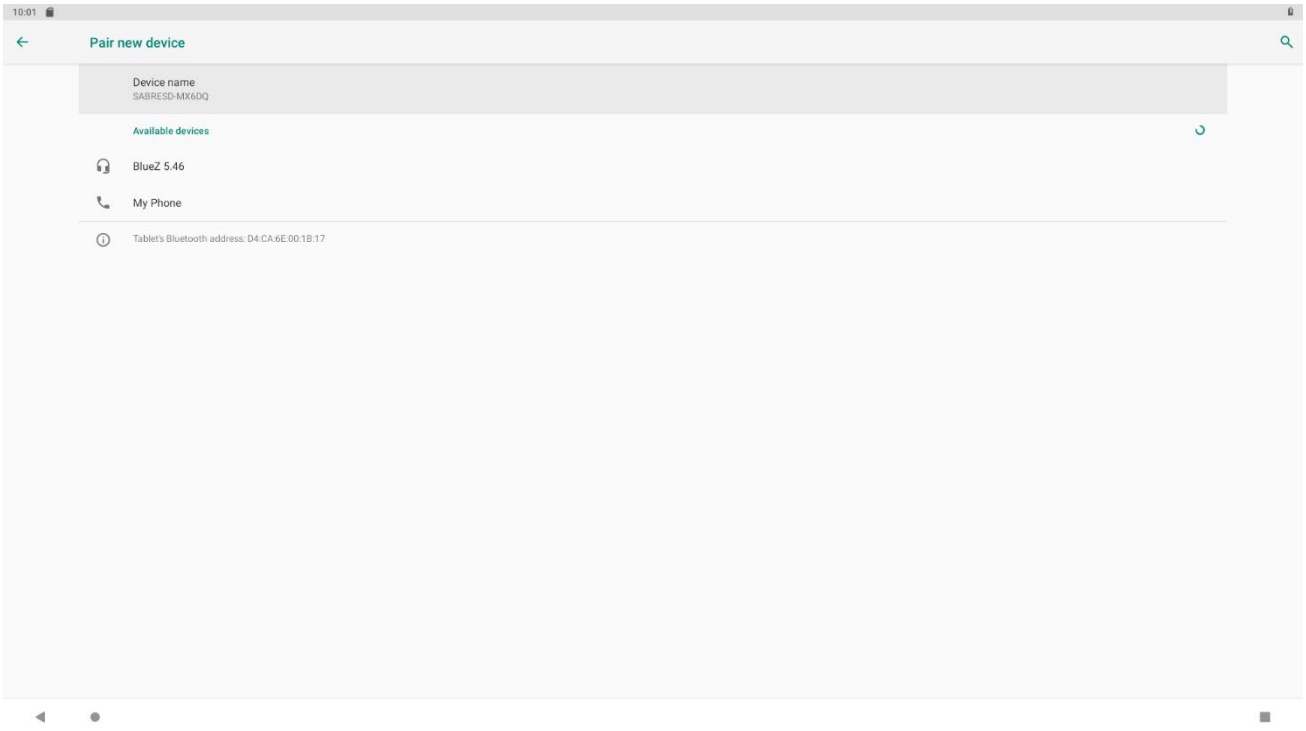


Figure 12: Bluetooth scanning

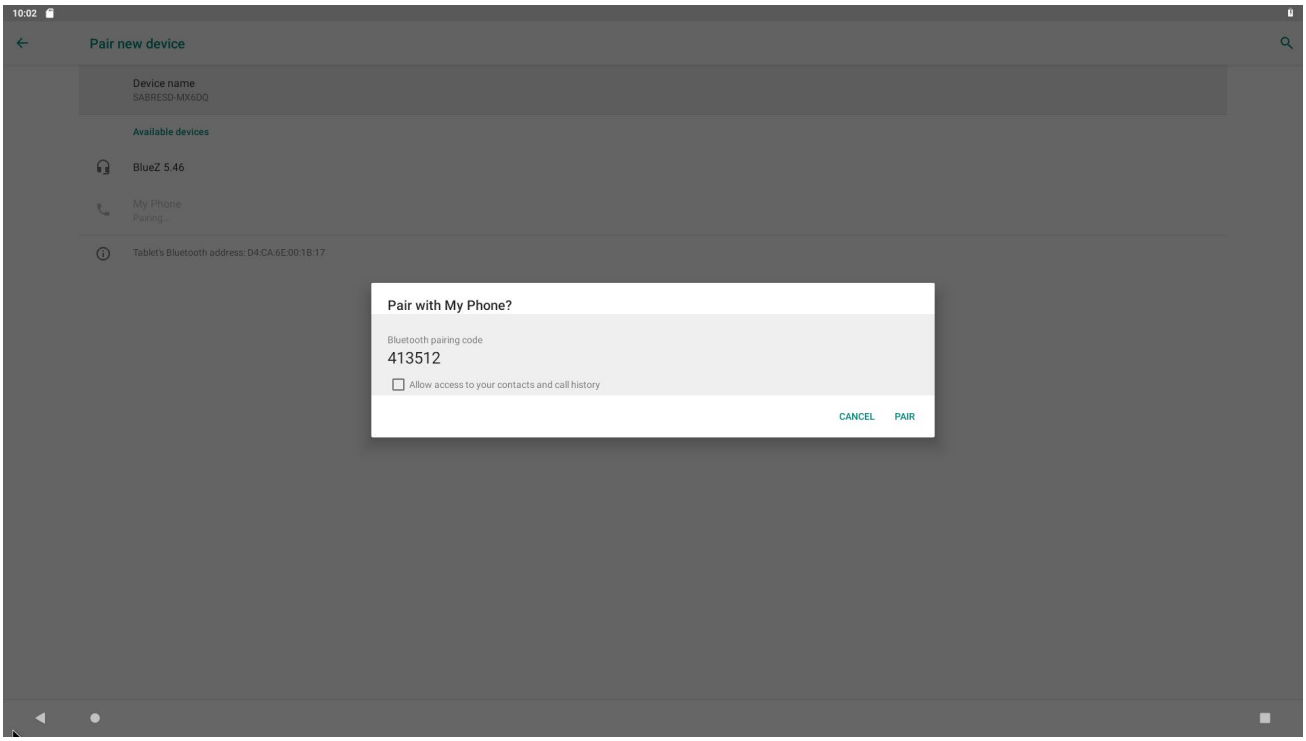


Figure 13: Bluetooth connect

Appendix

A Glossary

Abbreviation	Definition
AOSP	Android Open Source Project
ASCII	American Standard Code for Information Interchange
ARM	Arm (Advanced RISC Machines) Holdings
AEC	Automotive electronics council
BBR	Battery backed RAM
BER	Bit error rate
CPU	Central processing unit
UTC	Coordinated Universal Time
DTE	Data terminal equipment
DC	Direct current
DRX	Discontinuous reception
DDC	Display data channel
DL	Down link (Reception)
BT	Bluetooth
Wi-Fi	Wireless Fidelity
AP	Access Point

Table 7: Explanation of the abbreviations and terms used

Related documentation

[1] EMMY-W1 series system integration manual, UBX-15024929
[2] EVK-EMMY-W1 user guide, UBX-15012713
[3] EVK-LILY-W1 user guide, UBX-15030290
[4] EVK-JODY-W2 user guide, UBX-19027118
[5] EMMY-W1 series data sheet, UBX-15011785
[6] Cross reference platform for Android code browsing: http://aosp.opersys.com/xref/android-9.0.0_r47/
[7] Documentation of internal regulatory domain usage in kernel: https://wireless.wiki.kernel.org/en/developers/regulatory/wireless-regdb
[8] Android support forum for open source code, building and any other queries: https://source.android.com/setup
[9] Implementing SELinux: https://source.android.com/security/selinux/implement
[10] Android Pie integration guide from Marvell: MV-S302867-00B_Android P Vendor HAL porting
[11] Makefile architecture and reference: https://developer.android.com/ndk/guides/android_mk
[12] System architecture of Wi-Fi for Android: https://source.android.com/devices/tech/connect/wifi-overview
[13] WIN7 User Guide for using MFGTools: https://github.com/NXPmicro/mfgtools/wiki/WIN7-User-Guide
[14] Wireless regulatory database for CRDA: https://git.kernel.org/pub/scm/linux/kernel/git/sforshee/wireless-regdb.git/tree/
[15] iPerf for Android: https://apkpure.com/iperf-for-android/com.magicandroidapps.iperf
[16] MCIMX8M-EVK Evaluation Kit for the i.MX 8M Applications Processor: https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/evaluation-kit-for-the-i-mx-8m-applications-processor:MCIMX8M-EVK

For product change notifications and regular updates of u-blox documentation, register on our website, www.u-blox.com.

Revision history

Revision	Date	Name	Comments
R01	30-Oct-2019	aheg	Initial release.
R02	8-Sep-2020	aheg	Extended the document scope to include other modules based on NXP chipsets LILY-W1 and JODY-W2. Included updates to reflect the change in host platform from i.MX6 SABRE-SD to i.MX8 MQ EVK.

Contact

For complete contact information, visit us at www.u-blox.com.

u-blox Offices

North, Central and South America

u-blox America, Inc.

Phone: +1 703 483 3180
E-mail: info_us@u-blox.com

Regional Office West Coast:

Phone: +1 408 573 3640
E-mail: info_us@u-blox.com

Technical Support:

Phone: +1 703 483 3185
E-mail: support@u-blox.com

Headquarters
Europe, Middle East, Africa

u-blox AG

Phone: +41 44 722 74 44
E-mail: info@u-blox.com
Support: support@u-blox.com

Asia, Australia, Pacific

u-blox Singapore Pte. Ltd.

Phone: +65 6734 3811
E-mail: info_ap@u-blox.com
Support: support_ap@u-blox.com

Regional Office Australia:

Phone: +61 2 8448 2016
E-mail: info_anz@u-blox.com
Support: support_ap@u-blox.com

Regional Office China (Beijing):

Phone: +86 10 68 133 545
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Chongqing):

Phone: +86 23 6815 1588
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Shanghai):

Phone: +86 21 6090 4832
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Shenzhen):

Phone: +86 755 8627 1083
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office India:

Phone: +91 80 405 092 00
E-mail: info_in@u-blox.com
Support: support_in@u-blox.com

Regional Office Japan (Osaka):

Phone: +81 6 6941 3660
E-mail: info_jp@u-blox.com
Support: support_jp@u-blox.com

Regional Office Japan (Tokyo):

Phone: +81 3 5775 3850
E-mail: info_jp@u-blox.com
Support: support_jp@u-blox.com

Regional Office Korea:

Phone: +82 2 542 0861
E-mail: info_kr@u-blox.com
Support: support_kr@u-blox.com

Regional Office Taiwan:

Phone: +886 2 2657 1090
E-mail: info_tw@u-blox.com
Support: support_tw@u-blox.com