Getting started with AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip on iMX8 Nano Developer Kit V3

About this document

Scope and purpose

This guide provides step-by-step instructions to configure the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip. This guide provides an overview of setting up the host Embedded Artists iMX8 Nano, loading the WLAN driver, measuring the power consumption in Deep Sleep mode, establishing a Wi-Fi connection between an Access Point (AP) and station (STA). Configuring Bluetooth® and Bluetooth® Low Energy is also explained step-by-step.

Security is an integral part of AIROC™ devices, and the guide describes the security features embedded in the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip.

Intended audience

This document is intended for customers using the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip on iMX8 Nano Developer's Kit V3 as the host.

Table of contents

About this document **1**
Table of contents **2**
1.1 Document structure 4
2 Overview of hardware components **5**
2.1 What's inside the box (included in kit) 5
2.2 Required hardware for power measurement (not included in the kit) 5
3 CYW43022 rework for power measurement **7**
3.1 CYW43022 rework for power measurement 7
4 Wi-Fi bringup **10**
4.1 Wi-Fi image compatibility 10
4.2 Load the FMAC drivers and firmware 12
4.3 Configure the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip as STA 13
4.3.1 wpa_supplicant 13
4.3.2 Using wpa_cli 14
4.3.3 Using wpa_supplicant configuration 15
4.4 Configuring CYW43022 in Deep Sleep mode 16
4.4.1 Measure the power 16
4.4.2 Set up current measurement 17
5 Bluetooth® bringup **20**
5.1 Bring up AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip for Bluetooth® 20
5.2 Application software support for Bluetooth® development 20
5.2.1 Application layer – code examples 20
5.2.2 AIROC™ BTSTACK 21
5.2.3 BTSTACK porting layer 21
5.3 Bluetooth® firmware 22
5.3.1 Programing Bluetooth® firmware using MBT 22
5.3.2 Antenna configuration 23
5.4 BlueZ support 24
6 AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip security **25**
6.1 Overview 25
6.2 Secure boot 25
6.2.1 Access restriction 25
6.2.1.1 Wi-Fi subsystem 25
6.2.1.2 Bluetooth® subsystem 26
6.2.2 Preventing unauthorized firmware execution 26
6.2.2.1 Wi-Fi subsystem 26
6.2.2.2 Bluetooth® subsystem 27
6.3 Device lifecycle 27
6.4 Root of Trust (RoT) and bootloader flow (Wi-Fi) 28
6.5 Root of Trust (RoT) and bootloader flow (Bluetooth®) 30
7 Appendix **31**
7.1 iMX8 Nano image download mode 31
7.2 Flashing the pre-build package 31
7.3 Bring up AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip toolchain for custom image 33
7.3.1 Linux host setup 33
7.3.2 Required packages 34
7.3.3 Download Yocto recipes 35

Table of contents

7.3.4 Initialize the build 35
7.3.5 Custom iMX toolchain 36
7.3.6 Building the image 36
7.3.7 Building the kernel zImage 36
7.4 Building the FMAC driver from backports 42
7.4.1 Debugging the FMAC driver 43
Glossary 44
Revision history 45
Disclaimer 46

1.1 Document structure

Section 2, Overview of hardware components provides an overview of the various hardware components provided in the kit.

Section 3, CYW43022 rework for power measurement provides the hardware rework to be done for power measurement.

Section 4, Wi-Fi bringup provides the bringup instructions to use the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip such as configuring it as a station with the WL tool and measuring the power consumption (VBAT/VDDIO) in Deep Sleep mode.

Section 5, Bluetooth® bringup provides instructions to use the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip for Bluetooth® functionality.

Section 6, AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip security describes the security of the Wi-Fi and Bluetooth® subsystems and defense mechanisms within the subsystems to protect valuable data, information, or compute elements.

Section 7, Appendix explains the process of flashing an image on the host platform, setting up a toolchain to create custom images, and the steps to build drivers as well as how to debug.

2 Overview of hardware components

2.1 What's inside the box (included in kit)

- Host platform – iMX8M Nano Developer's Kit V3 from Embedded Artists (see Figure 1)
- AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip
- Micro-USB to USB-A cable
- CYW960PADM2-SDIO
- Antennas
- Power adapter 12 V, 2.5 A, 30 W

2.2 Required hardware for power measurement (not included in the kit)

- PC with Windows OS
- Power monitor (measures two channels) or two ammeters with µA ratings
- Ethernet LAN cable

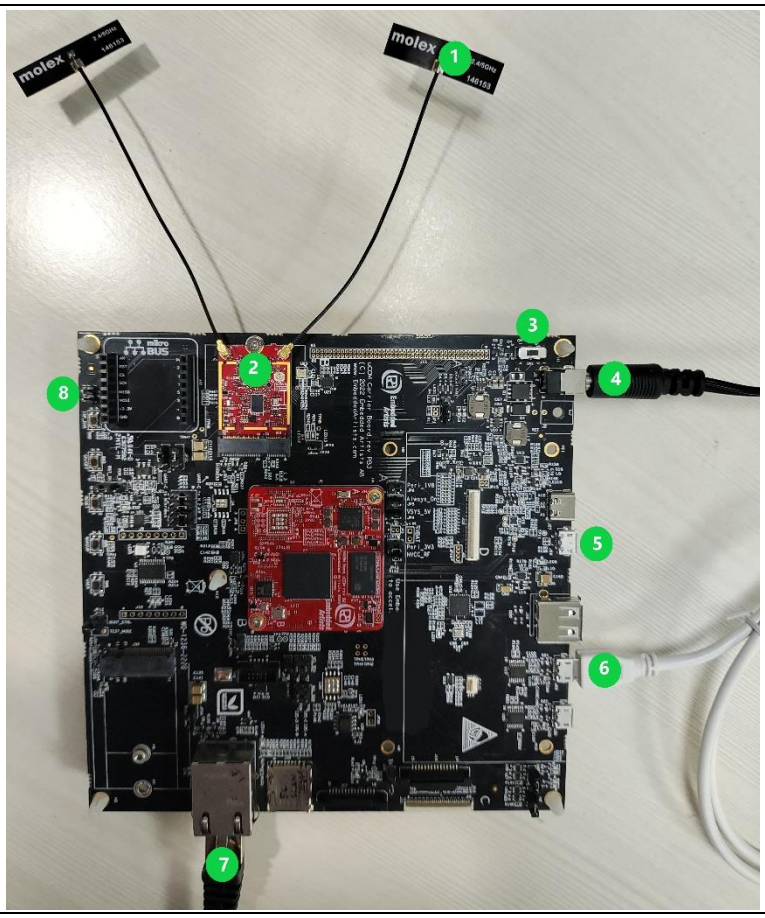


Figure 1 iMX8M Nano Developer Kit V3 board

The following are the hardware components of the setup:

1. Antennas
2. AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip
3. Power ON/OFF switch
4. Power supply DC Jack
5. Micro-USB to USB A cable for Image flashing
6. Micro-USB to USB A cable for serial console
7. Network cable connected to RJ45 connector
8. ISP enable jumper (OTG boot mode)

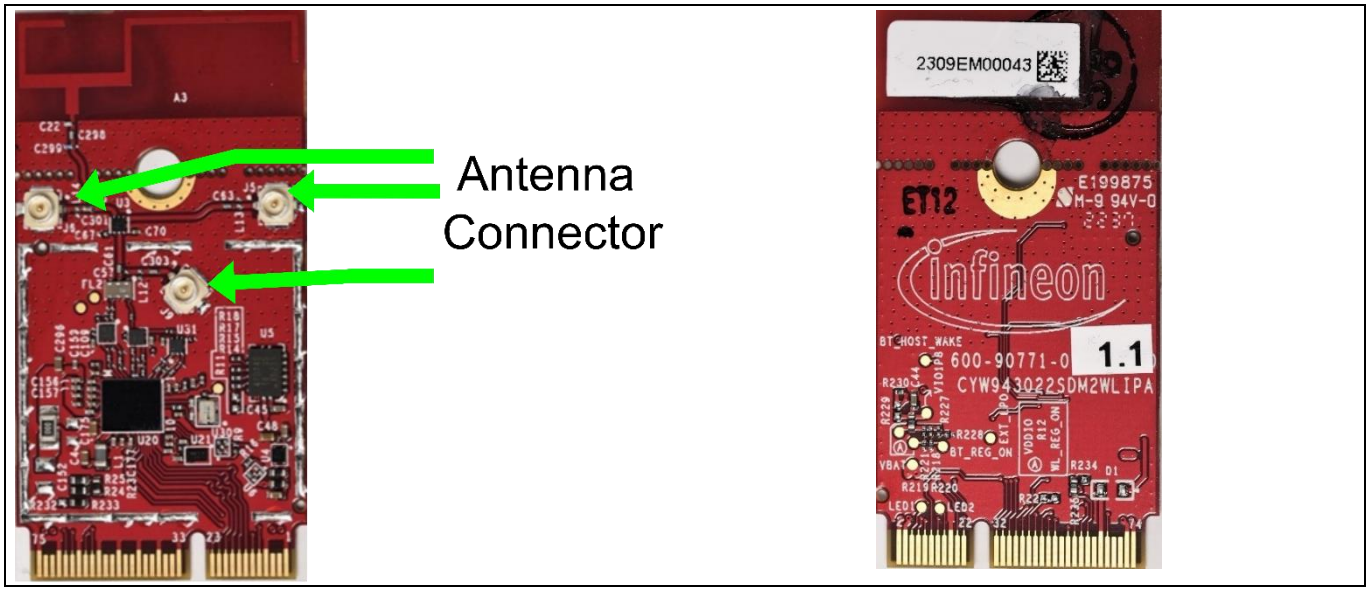


Figure 2 AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip

3 CYW43022 rework for power measurement

Note: This section explains the hardware rework to be done for power measurement. Disable JTAG and Bluetooth® functionality to measure accurate power for Wi-Fi subsystem. If you do not want to measure the power, you can skip this section and go to the Wi-Fi bringup section to explore the Wi-Fi and Bluetooth® functionality.

3.1 CYW43022 rework for power measurement

The CYW43022 M.2 module is powered by VBAT (3.3 V) and VDDIO (1.8 V). To measure the total power consumed by CYW43022, measure the individual currents (I_{VBAT} and I_{VDDIO}).

Do the following changes to measure the power consumption:

- **VBAT changes:** Remove R231 0Ω resistor to measure VBAT current (I_{VBAT}) individually (the red and black wires highlighted in Figure 3 are connected at the VBAT pad) and connect both wires across the pad of the resistor where the power monitor is in series (ammeter mode) connection.

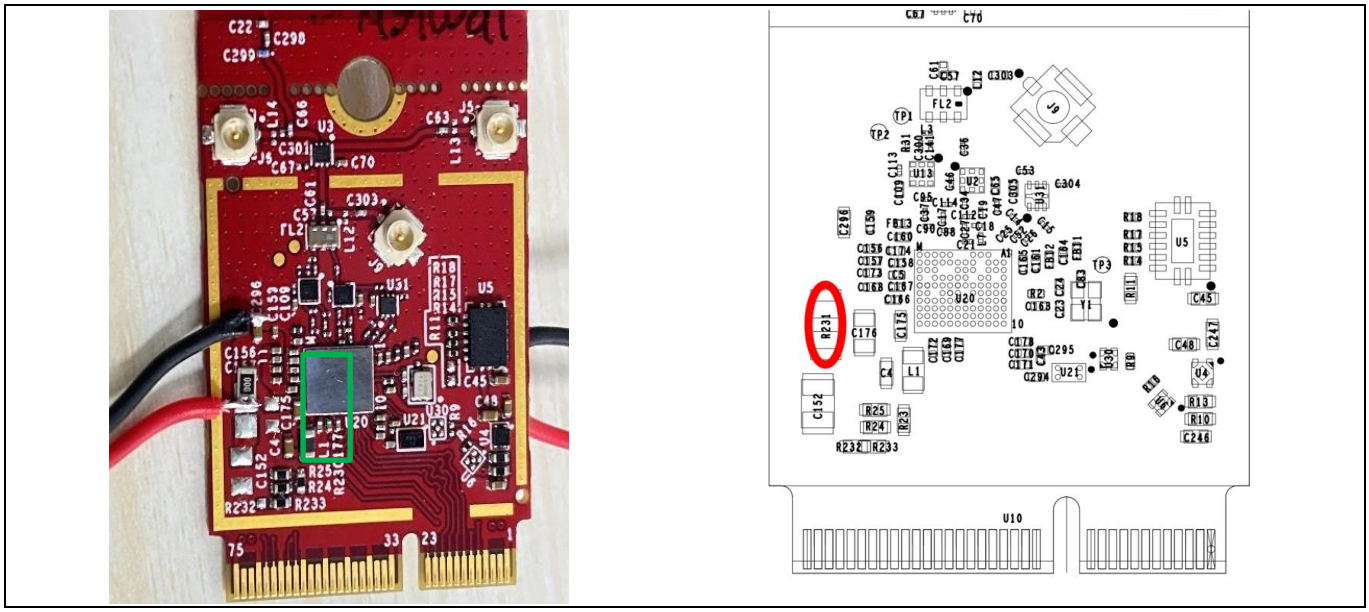


Figure 3 CYW43022 – VBAT rework

- **VDDIO changes**: Remove R12 0Ω resistor to measure VDDIO current (I_{VDDIO}) individually (the red and black wires highlighted in Figure 4 are connected at the VDDIO pad) and connect both wires across the pad of the resistor where the power monitor is in series (ammeter mode) connection.

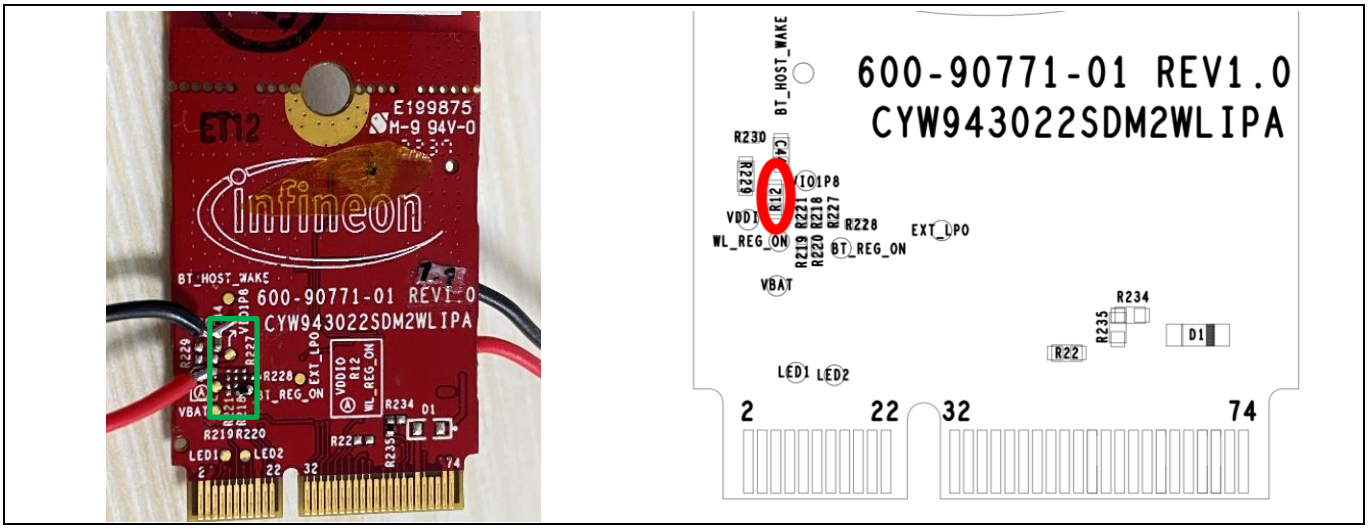


Figure 4 CYW43022 – VDDIO rework

- **BT_REG_ON changes**: The AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip contains both Wi-Fi and Bluetooth® functionalities. To measure an accurate power supply in the Wi-Fi subsystem, switch OFF the BT_REG_ON signal by removing the R15 resistor from the CYW43022 M.2 card. Mount a 0 Ohm resistor at R220. If you do not mount the resistor at R220, the Bluetooth® functionality works.

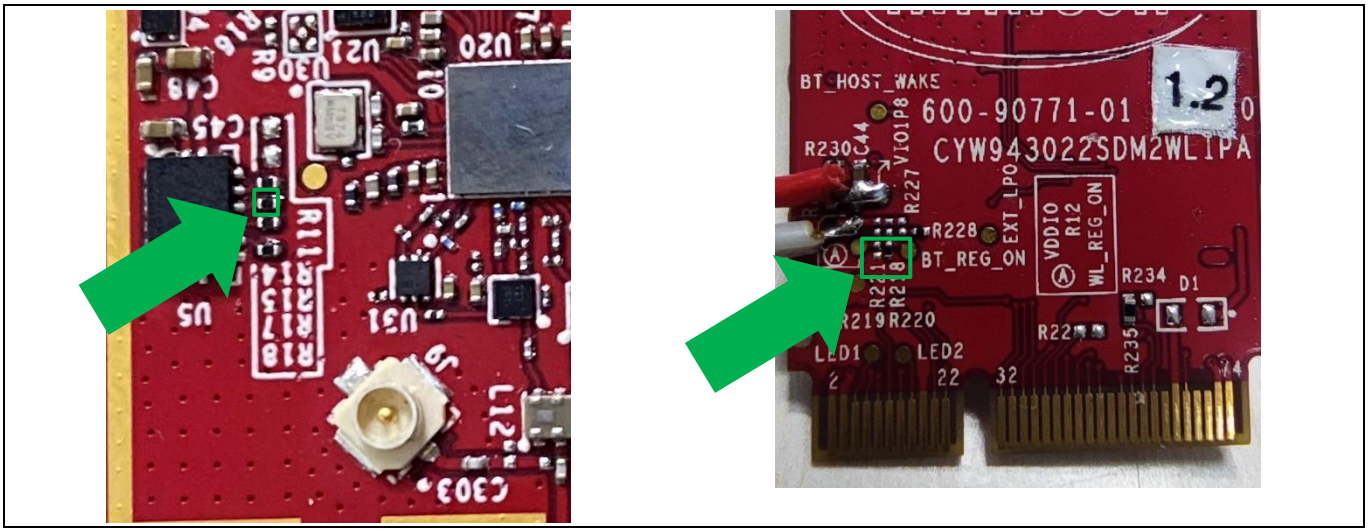
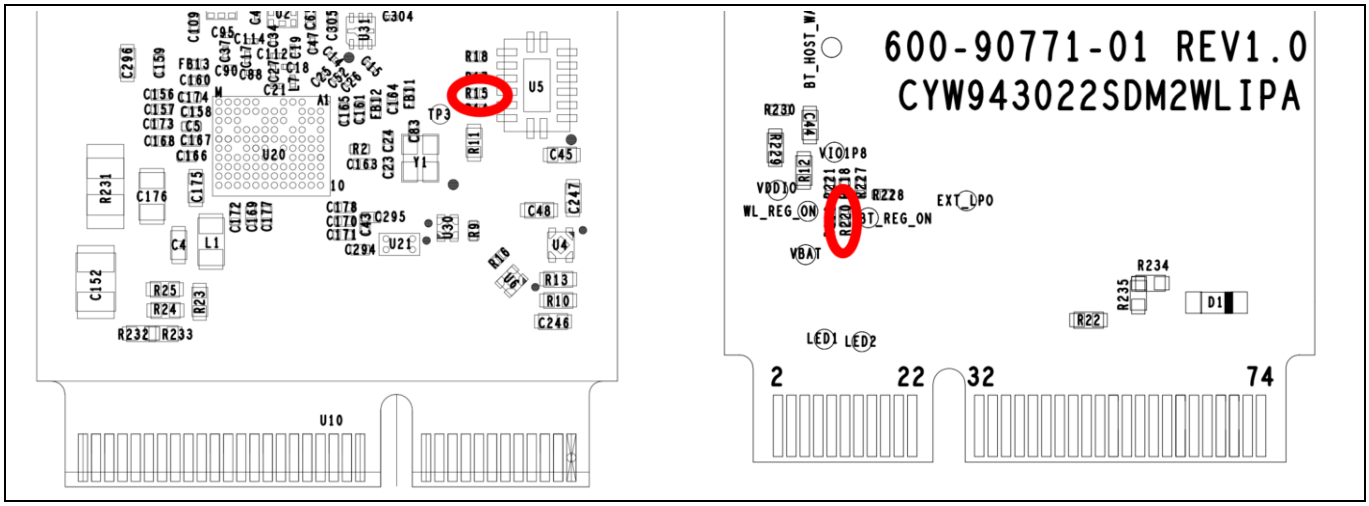
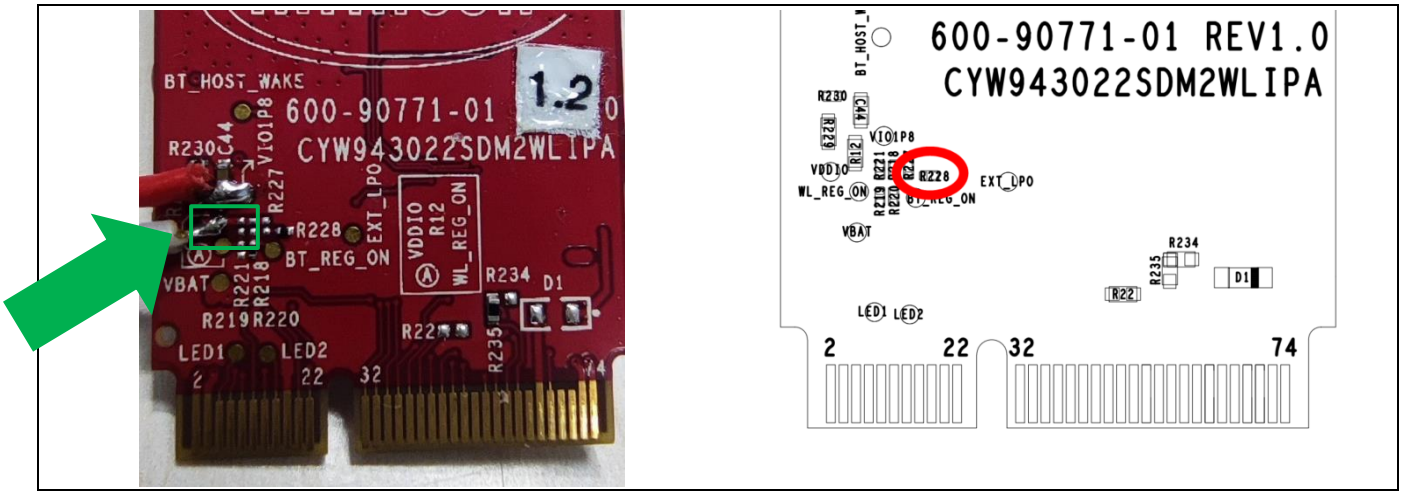


Figure 5 CYW43022– BT_REG_ON resistor



JTAG changes : Mount R228 (JTAG pulled LOW) to disable JTAG.



4 Wi-Fi bringup

4.1 Wi-Fi image compatibility

Connect CYW43022 to the M.2 E-key connector (J33) on the iMX8 Nano Developer's Kit V3 board, as shown in Figure 6.

Connect the serial port of the iMX8 Nano Developer's Kit V3 to the USB port of the laptop to make a serial connection (see Figure 6).

a) Locate the driver for Windows from the FTDI official website, download, and install the driver.

b) Choose an appropriate COM port in any serial terminal application such as PuTTY.

c) Set the baud rate to "115200".

d) Click **Open**.

Note: *If the COM port is not detected, follow the instructions on the iMX Developer's Kit V3 webpage to install the driver. If the computer does not recognize FTDI, update the FTDI USB to a serial driver to fix the issue.*

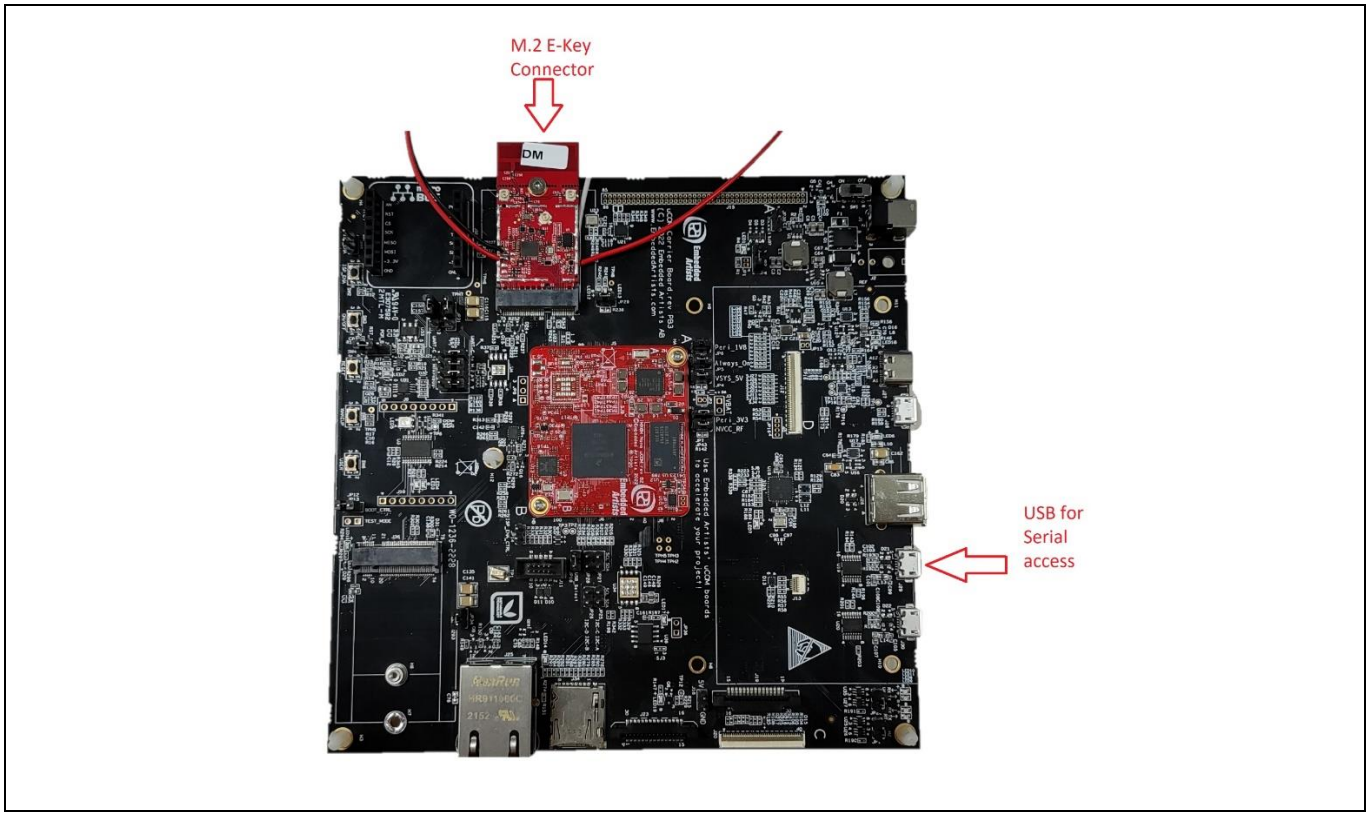


Figure 6 iMX8 Nano Developer Kit V3 – Serial port access

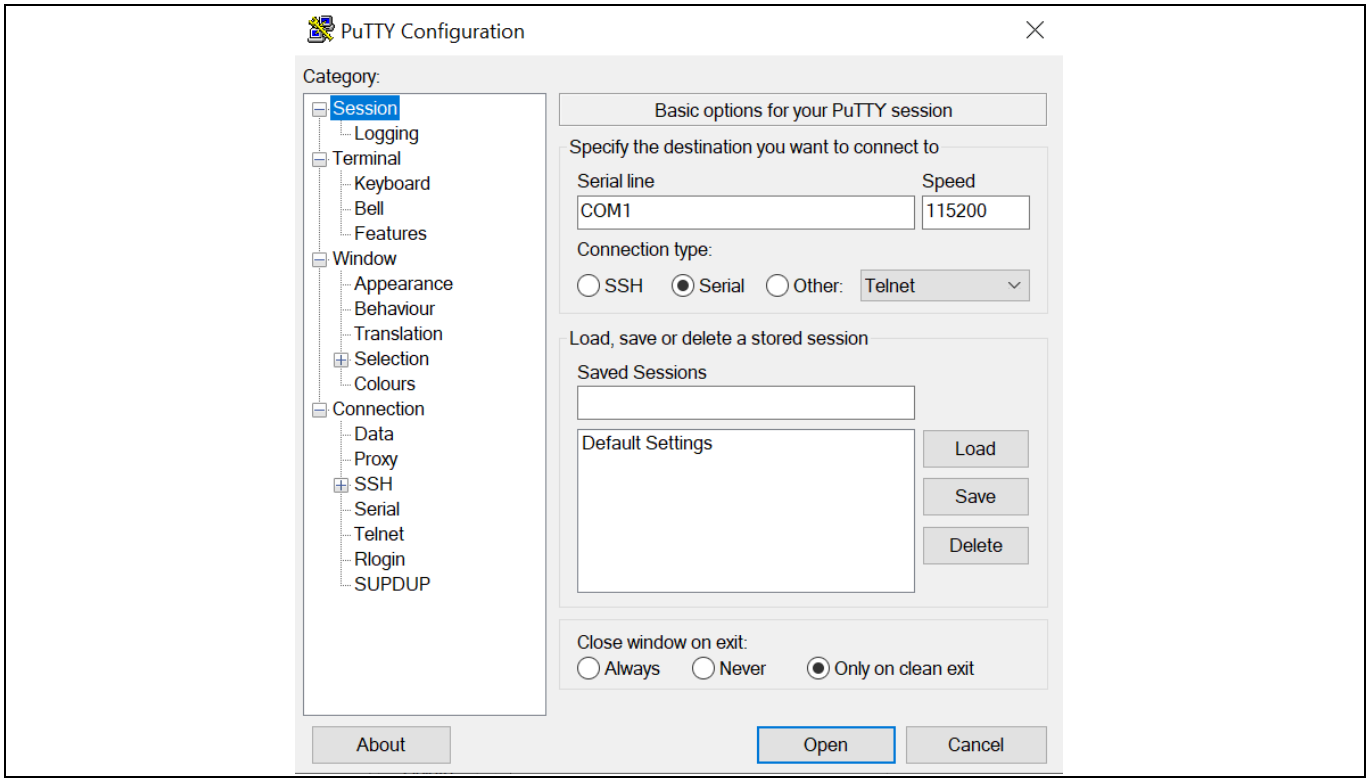


Figure 7 Serial Access Application-PuTTy

Now, power on the host. It will show kernel logs on the terminal and will prompt you to add a username and password for login. Use the following username and password to sign into the host platform:

- Username: **root**
- Password: **pass**

After powering up, to check the kernel version, use the following command:

```
uname -a
```

See the following example:

```
root@imx8mnea-ucom:~# uname -a
Linux imx8mnea-ucom 5.15.32IFX_FMAC-756198-ge9502013779c-dirty #19 SMP
PREEMPT Wed Mar 22 11:36:14 IST 2023 aarch64 GNU/Linux
```

If you have the preconfigured image already flashed on the iMX8 Nano Developer Kit V3, the Linux version will be `5.15.32IFX_FMAC-756198-ge9502013779c-dirty`. If you are not using a preconfigured board or you are using other images, the kernel version may be different. The AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip will work on a different kernel version as well. However, use `5.15.32IFX_FMAC-756198-ge9502013779c-dirty` to get the best performance and low power. See the Flashing the pre-build package section to flash the image.

Check the kernel logs using the `dmesg` command to confirm that the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip is detected:

```
root@imx8mnea-ucom:~ $ dmesg | grep SDIO
[    0.978232] mmc1: new high-speed SDIO card at address 0001
```

4.2 Load the FMAC drivers and firmware

1. Contact the local Infineon FAE or sales representative to get the *CYW43022.zip* release package.
2. Connect LAN to the iMX8 Nano Developer Kit V3 to copy the files via SSH connection and unzip the *CYW43022.zip* release package in the host platform.
3. Navigate to the *CYW43022/Wi-Fi* directory.

Note: *If you have a pre-flashed host from Infineon Technologies, the firmware files are already present in the /lib/firmware/cypress directory. If not and you flash the image using the Flashing the pre-build package section, you need to manually copy the firmware files using the following command:*

```
root@imx8mnea-ucom:~/CYW43022/wifi $ cp cyfmac43022-sdio*
/lib/firmware/cypress
```

4. To load the drivers, use the *load.sh* script in the CYW43022/Wi-Fi directory.
   `$ sh -x load.sh`

Note: Additionally, *you can use the following commands to load the drivers:*

```
root@imx8mnea-ucom:~ $ cd CYW43022/Wi-Fi
root@imx8mnea-ucom:~/CYW43022/wifi $ rmmod brcmfmac cfg80211 compat brcmutil
root@imx8mnea-ucom:~/CYW43022/wifi $ insmod ./brcmutil.ko
root@imx8mnea-ucom:~/CYW43022/wifi $ insmod ./compat.ko
root@imx8mnea-ucom:~/CYW43022/wifi $ insmod ./cfg80211.ko
root@imx8mnea-ucom:~/CYW43022/wifi $ insmod ./brcmfmac.ko
sdio_idleclk_disable=1
```

5. Check the loaded FMAC version. See the following example:

```
root@imx8mnea-ucom:~/ CYW43022# modinfo brcmfmac.ko | grep version
version: backported from Linux (v5.15.58-fafnir-6-43022-18-0-g51dfc6ba2397)
using backports v5.15.58-1-0-g42a95ce7
srcversion: 07BCCA37F3E26960070B4F1
vermagic: 5.15.32IFX_FMAC-756198-ge9502013779c-dirty SMP preempt mod_unload
modversions aarch64
```

Use the `ifconfig -a` command to verify that the driver is loaded and the interface is successfully up. In the output, see the new interface `wlan0`.

```
root@imx8mnea-ucom:~/CYW43022/wifi $ ifconfig -a
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 28  bytes 2833 (2.7 KiB)
```

```
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 28  bytes 2833 (2.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 00:90:4c:2f:a2:eb  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Note: *Dynamic Host Configuration Protocol (DHCP) is enabled by default; therefore, the wlan0 interface gets the IP address automatically after connecting to an AP. However, to configure the IP address of the interface manually to a static IP address, use* `$ ifconfig wlan0 192.168.1.7/24`.

Note: *The MAC address for the wireless interface can be changed via the NVRAM file (cyfmac43022-sdio.txt) available in the package.*

4.3 Configure the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip as STA

4.3.1 wpa_supplicant

Use the wpa_supplicant user-space Linux daemon to handle the WLAN authentication, association, disassociation, and deauthentication processes for STAs. The wpa_supplicant is a cross-platform utility that provides the support for WEP, WPA, WPA2, WPA3 (IEEE 802.11i), and WPA-EAP. It implements the key negotiation with an authenticator and controls the roaming and association of STA devices. The wpa_supplicant version 2.10 or above with Infineon patches is required.

/etc/wpa_supplicant.conf is the default path for the WPA supplicant configuration file.

Use the following Linux SystemD commands to control the operation of the WPA supplicant:

1. Use the following command to initialize the *wpa_supplicant* tool:
`./wpa_supplicant -B -c <location of .conf file > -i <interface name> -B`

For example, root@imx8mnea-ucom: /CYW43022/Wi-Fi#/wpa_supplicant -B -c/etc/wpa_supplicant.conf -iwlan0

Note: *You can use **wpa_cli or wpa_supplicant.conf** to connect to a wireless network.*

2. When connecting to an AP with wpa2-psk security, use the following command:
`./wpa2_associate.sh <SSID> <PSK>`

For example, root@imx8mnea-ucom: sh -x wpa2_associate.sh MY_SSID MY_PASSWORD

4.3.2 Using wpa_cli

1. Use the following command to initialize *wpa_cli*:
   ```
   root@imx8mnea-ucom:~/CYW43022/Wi-Fi# ./wpa_cli
   ```
   This command can be used to interact with *wpa_supplicant*. It can be used to query current status, change configuration, trigger events, and request interactive user input.
2. Scan for available Wi-Fi networks using the following commands:
   - `scan`
   - `scan_results`
3. Use the following wpa_cli sequence to connect to a Wi-Fi network with open security. The following example uses an SSID “ifx-open”. The SSID can be modified to any value.

```
Remove_n all
add_network
set_network 0 ssid "ifx-open"
set_network 0 key_mgmt NONE
enable_network 0
select_network 0
status
```

4. Use the wpa_cli sequence to connect an authenticated SSID with wpa2-psk security. The following example uses an SSID “ifx-wpa2”. The SSID can be modified to any value.

```
remove_n all
add_network
set_network 0 ssid "ifx-wpa2"
set_network 0 proto WPA2
set_network 0 key_mgmt WPA-PSK
set_network 0 pairwise CCMP
set_network 0 psk "password"
enable_network 0
select_network 0
```

5. Use the wpa_cli sequence to connect an authenticated SSID with wpa3-psk security. The following example uses an SSID “ifx-wpa3”. The SSID can be modified to any value.

```
remove_n all
scan
scan_results
add_network
set_network 0 ssid "ifx-wpa3"
set_network 0 proto WPA2
set_network 0 key_mgmt SAE
set_network 0 ieee80211w 2
set_network 0 pairwise CCMP
set_network 0 sae_password "password"
enable_network 0
select_network 0
```

For more information, see wpa_supplicant.

4.3.3 Using wpa_supplicant configuration

Use one of the following sample contents depending on the required security in the conf file available at *etc/wpa_supplicant.conf*:

```
# WPA Supplicant with open networksecurity configuration.
ctrl_interface=/var/run/wpa_supplicant
driver_param=use_p2p_group_interface=1p2p_device=1
update_config=1
device_name=EA-iMX8-LINUX
config_methods=virtual_push_button physical_display keyboard
interworking=1
sae_pwe=2
sae_groups=19
network={
    ssid="ifx-open"
    key_mgmt=NONE
}
```

Or

```
# WPA Supplicant with WPA2PSK-AES network security configuration.
ctrl_interface=/var/run/wpa_supplicant
driver_param=use_p2p_group_interface=1p2p_device=1
update_config=1
device_name=EA-iMX8-LINUX
config_methods=virtual_push_button physical_display keyboard
interworking=1
sae_pwe=2
sae_groups=19
network={
    ssid="ifx-wpa2psk"
    key_mgmt=WPA-PSK
    proto=WPA2
    pairwise=CCMP
    psk="password"
}
```

Or

```
# WPA Supplicant with WPA3-SAE network security configuration.
ctrl_interface=/var/run/wpa_supplicant
driver_param=use_p2p_group_interf
```

```
ace=1p2p_device=1
update_config=1
device_name=EA-iMX8-LINUX
config_methods=virtual_push_button physical_display keyboard
interworking=1
sae_pwe=2
sae_groups=19
network={
    ssid="ifx-wpa3"
    scan_ssid=1
    ieee80211w=2
    key_mgmt=SAE
    proto=RSN
    pairwise=CCMP
    sae_password="password"
}
```

4.4 Configuring CYW43022 in Deep Sleep mode

Wi-Fi or IoT devices should remain associated with an access point (AP) and need to consume less power. Deep Sleep is one such feature that can be used for low-power activities on the Wi-Fi chip.

4.4.1 Measure the power

To measure the CYW43022 power consumption, connect the power monitor in series with the AIROC™ 43022 device (see Figure 11).

To measure the power, use the reworked AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip. Connect the power monitor configured in ammeter mode or µA rating ammeters in series with the reworked AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip device (see Figure 8).

To measure the current, connect the VBAT (3.3 V) and VDDIO (1.8 V) power lines in series with the power monitor (measures two channels) or two ammeters.

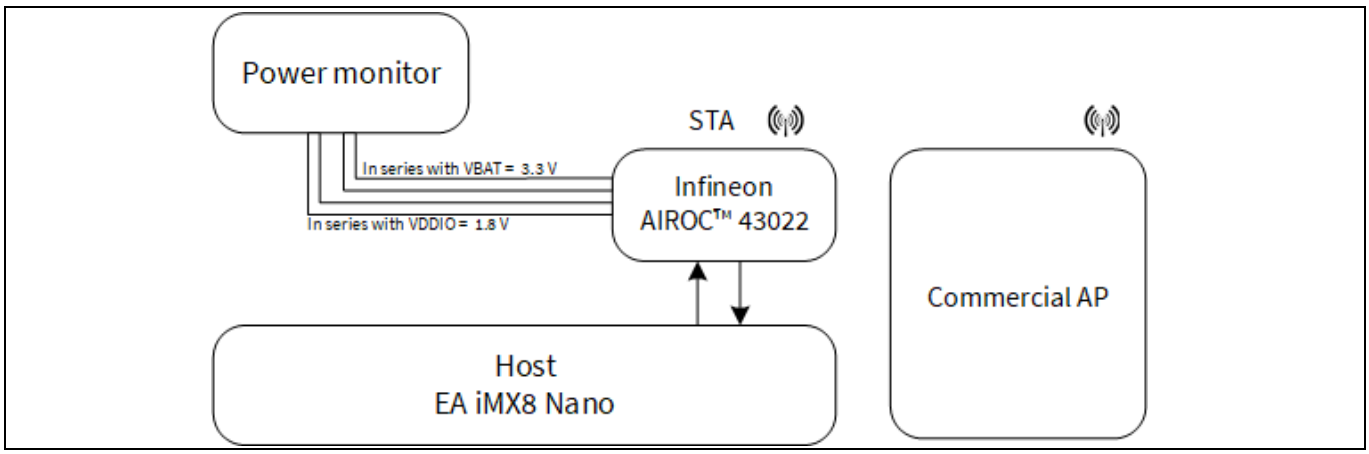


Figure 8 Power monitor setup topology

4.4.2 Set up current measurement

1. Power OFF the iMX8 Nano Developer Kit V3 board before making the series connections.
2. For VBAT (3.3 V), do the following (see Figure 11):
   a) Connect J4 of the reworked CYW43022 board in series with the power monitor.
3. For VDDIO (1.8 V), do the following (see Figure 11):
   b) Connect J3 of the CYW43022 board in series with the power monitor.
4. For current measurements:
   c) Set the power monitor emulation to the current measure configuration (ammeter configuration).
   d) Connect both channels.

Note: Supply VBAT and VDDIO before powering on the iMX8 Nano Developer Kit V3.

5. Power ON the iMX8 Nano Developer Kit V3 and load the drivers as explained in the Load the FMAC drivers and firmware section. By default, the Sleep state is enabled. Confirm it using the following command:

```
root@imx8mnea-ucom:~/CYW43022/wifi $ ./wl ulp
```

Note: This command gives the output as '0' for Deep Sleep mode.

6. Connect to an AP as explained in the Using wpa_cli section. After associating with the AP, CYW43022 remains in Deep Sleep mode. Confirm it using the `wl ulp` command.
7. On loading the drivers, by default, the Sleep state is enabled. Use the following commands to switch to a Deep Sleep state:

```
root@imx8mnea-ucom:~/CYW43022/wifi $ ./prepare_deep_sleep.sh
root@imx8mnea-ucom:~/CYW43022/wifi $ ./wl ulp 2
```

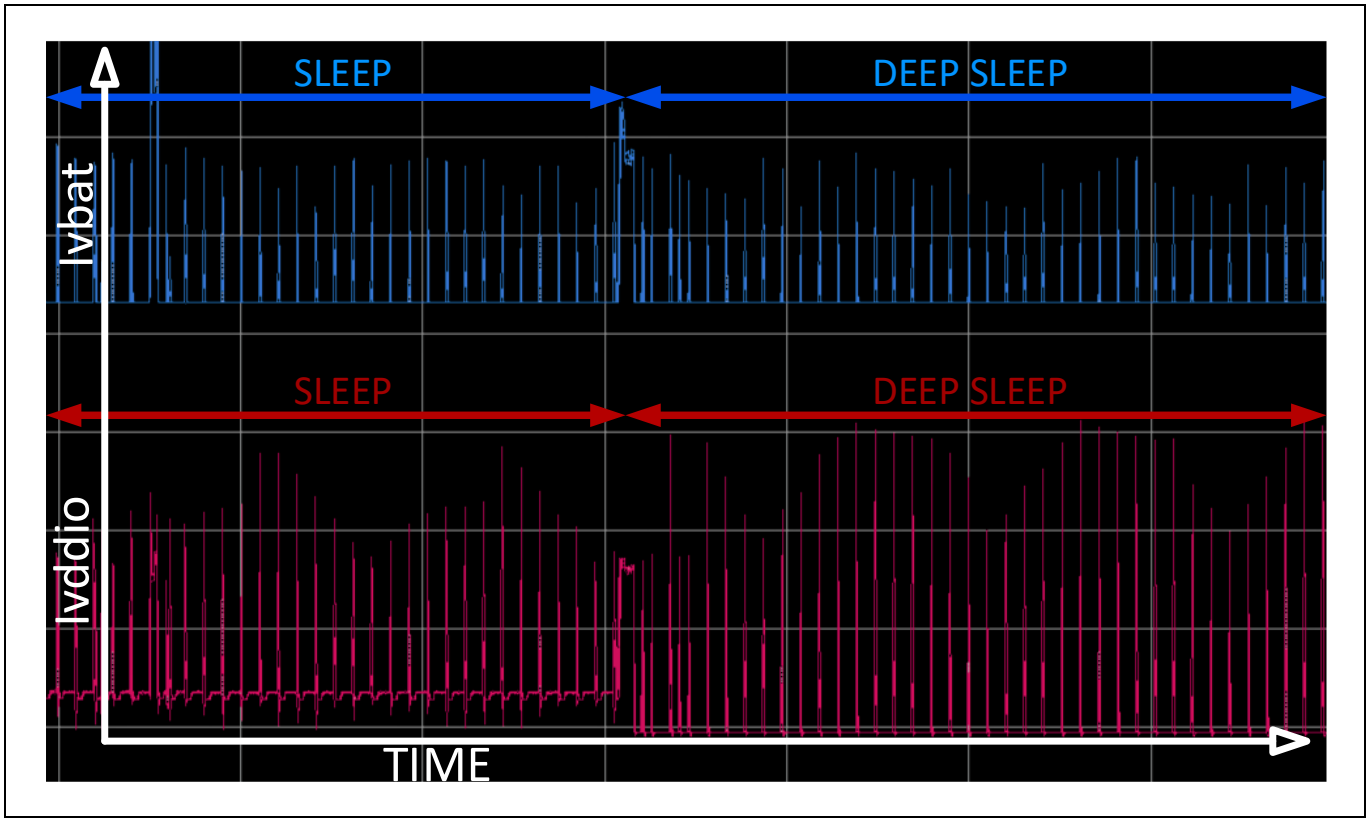


Figure 9 Waveform of the CYW43022 chip entering the Deep Sleep mode

Figure 9 shows the waveform of Ivbat and Ivddio transitioning from the Sleep state to the Deep Sleep state.

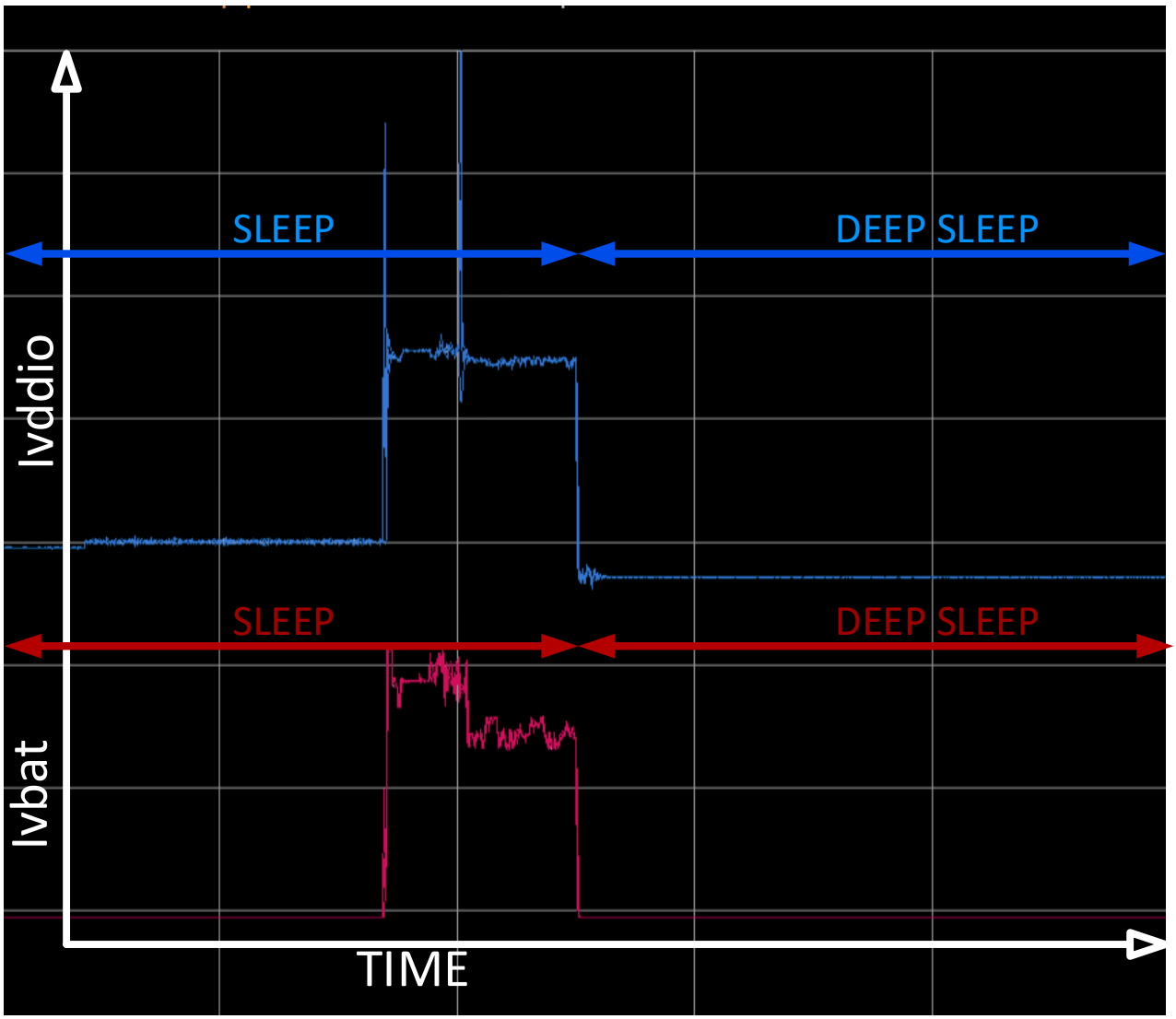


Figure 10 Waveform zoomed

Table 1 Current measurement in VBAT and VDDIO

DTIM	Sleep		Deep Sleep	
	VBAT	VDDIO	VBAT	VDDIO
DTIM 1	219.691 µA	79.727 µA	171.571µA	75.313 µA
DTIM 3	113.063 µA	77.741 µA	81.582 µA	76.227 µA
DTIM 10	55.912 µA	78.726 µA	54.907 µA	22.048 µA

Note: Table 1 lists the sample of the current consumption tested in an Infineon development environment; actual results may vary.

5 Bluetooth® bringup

5.1 Bring up AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip for Bluetooth®

Verify whether the BT_REG_ON pin is configured properly using the following command (see Figure 11):

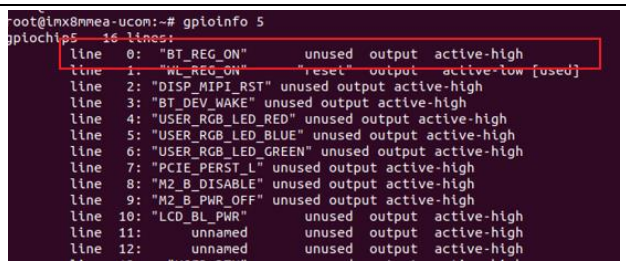


Figure 11 BT_REG_ON pin GPIO status

Use the following software for Bluetooth® development with the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip:

- AIROC™ BTSTACK. For more details, see Application software support for Bluetooth® development.
- Open source BTSTACK. For more details, see BlueZ support.

5.2 Application software support for Bluetooth® development

The following are the software layers:

- Application layer – code examples
- AIROC™ BTSTACK
- BTSTACK porting layer
- MBT - Manufacturing Bluetooth® Test Tool

5.2.1 Application layer – code examples

Note: These code examples only work with AIROC™ BTSTACK.

The application layer contains code examples with a basic structure that demonstrate how to use the AIROC™ BTSTACK APIs to use the Bluetooth® functionality of the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip. The Linux Bluetooth® code example has both Bluetooth® classic and Bluetooth® Low Energy code examples.

The following code examples are available on GitHub:

- AIROC™ BTSTACK: Bluetooth® A2DP Sink for Linux host
- AIROC™ BTSTACK: Bluetooth® Handsfree Audio Gateway for Linux host
- AIROC™ BTSTACK: Bluetooth® Handsfree Unit for Linux host
- AIROC™ BTSTACK: Bluetooth® SPP server for Linux host
- AIROC™ BTSTACK: Bluetooth® LE hello sensor for Linux host
- AIROC™ BTSTACK: Bluetooth® Alert Server(ANS) for Linux host
- AIROC™ BTSTACK: Bluetooth® Alert Client(ANC) for Linux host
- AIROC™ BTSTACK: Bluetooth® Battery Server (BAS) for Linux host

For more information on how to compile and use the code example, see the *README.md* file present in each code example directory.

The following are the changes for the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip in each *README.md*:

- Make sure to use `-f` (FW download baud rate) to 115200 (instead of existing 921600) in the "Operation" section while executing the compiled application.
- Use the AIROC™ BTSTACK v3.6.1 in each code example. Use the following command (instead of the existing btstack clone command) to clone the AIROC™ BTSTACK v3.6.1 in Step-3 of "Using the code example" section:

```
$ git clone https://github.com/Infineon/btstack.git --branch release-v3.6.1
```

5.2.2 AIROC™ BTSTACK

AIROC™ BTSTACK is optimized to work with Infineon controllers. It implements the WICED Bluetooth® APIs and supports Bluetooth® BR/EDR and Bluetooth® LE core protocols. AIROC™ BTSTACK is built with two variants: Bluetooth® LE only and Dual-Mode. The Bluetooth® LE library has APIs that only support Bluetooth® LE features, while the Dual-Mode library supports both Bluetooth® LE and Bluetooth® classic features.

AIROC™ BTSTACK is used as a library that can be pulled in to build a specific Bluetooth® demo or customer applications. AIROC™ BTSTACK works with the WICED library and porting layer.

For more information on AIROC™ BTSTACK, see the following:

- AIROC™ BTSTACK release notes
- Bluetooth® LE API reference manual
- DUAL MODE API reference manual

AIROC™ BTSTACK and AIROC™ BTSTACK porting layer on the GitHub repository:

- AIROC™ BTSTACK
- AIROC™ BTSTACK porting layer

5.2.3 BTSTACK porting layer

AIROC™ BTSTACK runs on a Linux host, while Bluetooth® firmware runs on the Bluetooth® controller (AIROC™ combo chip). To communicate with each other, the host and controller use the HCI protocol. The AIROC™ BTSTACK library integrates with the porting layer. The porting layer implements the OS functionalities required by the stack.

The porting layer has the following functionalities:

- Firmware programing
- HCI communication with the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip
- Timer
- GPIO operations

The porting layer recognizes the packets using the HCI packet type indicator (PTI) field. From the Bluetooth® core specification, the HCI protocol supports the following five types of packets:

- Command
- Asynchronous data
- Synchronous data

- Event
- ISO data

The PTI is a one-byte value that precedes the HCI packet.

The transmitting path (Tx) packets can be directly sent to the controller by the AIROC™ BTSTACK. The receiving path (Rx) packets are received from the Bluetooth® thread and transmitted directly to AIROC™ BTSTACK.

HCI command packets are sent to the controller via the Tx path. HCI event packets are sent from the controller via the Rx path. HCI asynchronous connection link (ACL), synchronous connection oriented (SCO), and isochronous (ISO) data packets are sent both to the Tx path and from the Rx path to the controller.

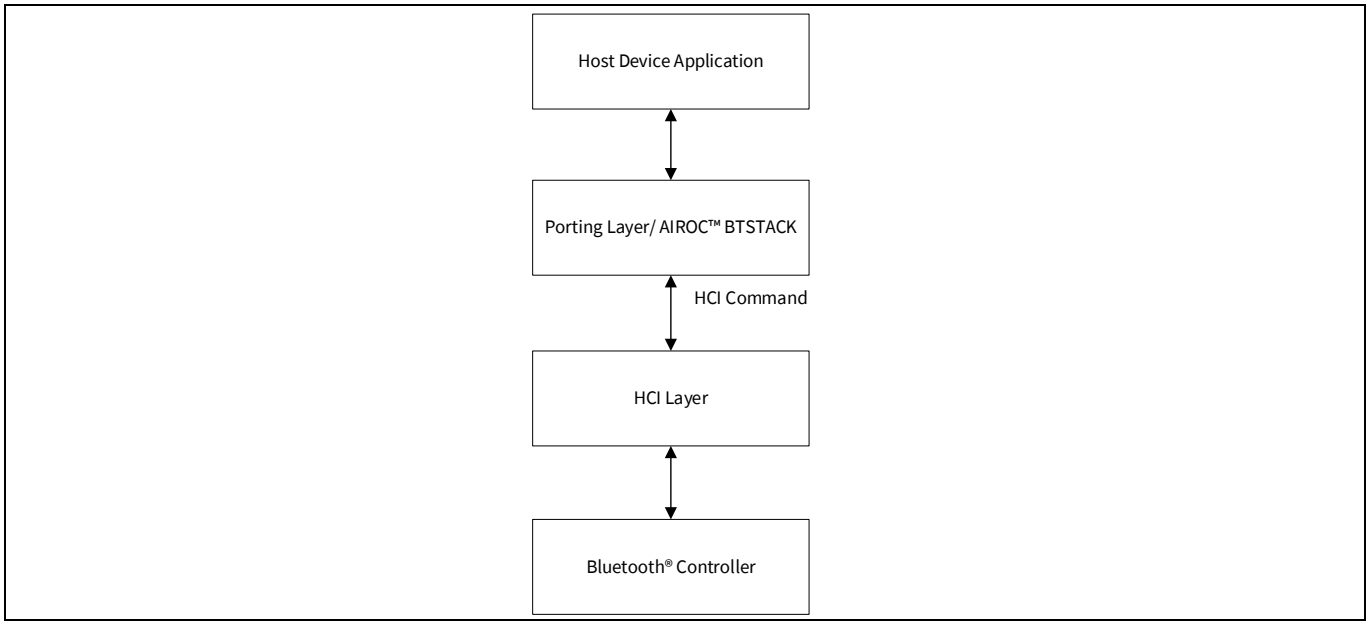


Figure 12 Software layer flow

5.3 Bluetooth® firmware

5.3.1 Programing Bluetooth® firmware using MBT

Note: *If you are using a code example, skip this section because the code example downloads the Bluetooth® firmware automatically.*

Use the Manufacturing Bluetooth® (MBT) test tool to:

- Send and receive command to and from the Bluetooth® chip to test few Bluetooth® features during manufacturing.
- Test the RF performance of the Infineon Bluetooth® classic and Bluetooth® Low Energy devices. MBT sends an HCI command to the device and then waits for HCI complete events.

Follow these steps to program the Bluetooth® firmware using MBT:

1. Download the latest mbt source from the github.com/Infineon/mbt repo to */home/root* directory on iMX8M Nano.
2. Navigate to the *CYW43022/BT/mbt/cyw43022/scripts_for_imx8* directory that contains all necessary script files required to download the Bluetooth® firmware files.

3. Copy the Bluetooth® firmware patch file from the release package (*CYW43022/BT/FW/CYW43012C1_wlbga_iPA_dLNA.hcd*) to the *CYW43022/BT/mbt/cyw43022/scripts_for_imx8* directory and rename it to "*FW.hcd*".
4. Navigate to the */home/ifx/scripts_for_imx8* directory.
5. Run the following command:
   ```
   $ chmod +x mbt; chmod +x bt_autobaud.sh
   ```
6. Run the following command:
   ```
   $ export MBT_TRANSPORT=/dev/ttymxc0
   ```
7. Run the following command:
   ```
   $ ./mbt download FW.hcd -minidriver
   ```

```
root@imx8mnea-ucom:~/scripts_for_imx8# chmod +x mbt; chmod +x bt_autobaud.sh
root@imx8mnea-ucom:~/scripts_for_imx8# export MBT_TRANSPORT=/dev/ttymxc0
root@imx8mnea-ucom:~/scripts_for_imx8# ./mbt download FW.hcd --minidriver
[MBT_TRANSPORT: /dev/ttymxc0]
Wait Controller Detect CTS low
Init UART ..........
Download BT firmware. This may take a few moments...
tx: (4 bytes)
 01 03 0c 00
rx: (7 bytes)
 04 0e 04 01 03 0c 00
Enter parse_patchram
Exit parse_patchram

 Sending Download minidriver
tx: (4 bytes)
 01 2e fc 00
rx: (7 bytes)
 04 0e 04 01 2e fc 00
... hcd files ...
.....................................................................
Exit proc_patchram 4
tx: (4 bytes)
 01 03 0c 00
rx: (7 bytes)
 04 0e 04 01 4c fc 12
Current state: Completed successfully
root@imx8mnea-ucom:~/scripts_for_imx8#
```

Figure 13 Programing Bluetooth® firmware from iMX8M Nano using MBT

For more information on different MBT commands or features, see the readme file available in the release package at "*CYW43022/BT/mbt/cyw43022/scripts_for_imx8*" directory.

5.3.2 Antenna configuration

The AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip comes with the WLBGA package.

WLBGA supports the following two different antenna configurations (Bluetooth® firmware files):

- wlbga_iPA_dLNA
- wlbga_iPA_sLNA

Select the CYW43022 combo chip antenna configurations at the time of Bluetooth® firmware programming by selecting the corresponding antenna's configuration hcd file.

Find the CYW43022 Bluetooth® firmware (*.hcd* files) in the release package. Extract the release package and traverse to the "*CYW43022/BT/FW*" directory for the same.

For example, you can use the following Bluetooth® firmware file for dLNA configuration:

CYW43022/BT/FW/CYW43012C1_wlbga_iPA_dLNA.hcd.

5.4 BlueZ support

1. Download the Bluetooth® firmware patch file. For more details, see the Programing Bluetooth® firmware using MBT section.
2. Run the following command to set up BlueZ on a particular HCI port (*/dev/ttymxc0*):
   ```
   $  hciattach -s 115200 /dev/ttymxc0 bcm43xx
   ```
3. Run the following command to configure the port:
   ```
   $  hciconfig hci0 up
   ```

```
root@imx8mnea-ucom:~/scripts_for_imx8# hciattach -s 115200 /dev/ttymxc0 bcm43xx
bcm43xx_init
Patch not found, continue anyway
Set Controller UART speed to 3000000 bit/s
Setting TTY to N_HCI line discipline
Device setup complete
root@imx8mnea-ucom:~/scripts_for_imx8# hciconfig hci0 up
root@imx8mnea-ucom:~/scripts_for_imx8#
```

Figure 14 HCI UART enabling with hciattach command

4. To test some basic BlueZ stack functionality, run the following commands:
   ```
   $ bluetoothctl
   $ power on
   $ agent on
   $ scan on
   ```

```
root@imx8mnea-ucom:~/scripts_for_imx8# bluetoothctl
Agent registered
[CHG] Controller AA:AA:AA:AA:AA:AA Pairable: yes
[bluetooth]# power on
Changing power on succeeded
[bluetooth]# agent on
Agent is already registered
[bluetooth]# scan on
Discovery started
[CHG] Controller AA:AA:AA:AA:AA:AA Discovering: yes
[NEW] Device 5E:5A:B5:16:D7:12 5E-5A-B5-16-D7-12
[NEW] Device 48:B0:2D:11:AF:41 48-B0-2D-11-AF-41
[DEL] Device 5E:5A:B5:16:D7:12 5E-5A-B5-16-D7-12
[NEW] Device 5E:5A:B5:16:D7:12 5E-5A-B5-16-D7-12
[DEL] Device 48:B0:2D:11:AF:41 48-B0-2D-11-AF-41
[NEW] Device 7C:20:DB:F0:4B:E3 7C-20-DB-F0-4B-E3
[NEW] Device 48:B0:2D:11:AF:41 NVIDIA SHIELD Remote
[bluetooth]#
```

Figure 15 Sanity Bluetooth® test using BlueZ

6 AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip security

6.1 Overview

The value of IoT devices is based on being smart, connected, and increasingly secure. Many applications demand secure processing elements and secure connections to the cloud. But security is only as strong as its weakest link. An IoT device that uses a secure MCU, possibly with a secure element for key or credential storage, can still be vulnerable to over-the-air or side-channel attacks over Bluetooth® and Wi-Fi if those sub-systems are not also secure.

The AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip uses the concept of Defense in Depth (DiD), which originates from the military strategy where multiple levels of defense are deployed to delay an incoming attack. This concept is popular in the field of cyber security and will be an integral part of how we look at security and the system architecture of AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip that supports it.

In AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip, DiD is implemented through a mechanism such as secure boot, which is a security standard that ensures a device only boots using software whose authenticity is known and verifiable. This is implemented through access restriction and preventing unauthorized firmware execution. Multiple device life cycles are implemented, which enhances the security of each lifecycle.

In an IoT system, independent defense mechanisms within sub-systems are layered in order to protect valuable data, information, or compute elements. This multi-layered approach increases the overall security of a system and defends it against many different types of attack vectors.

6.2 Secure boot

Secure boot is a security standard that ensures a device only boots using software whose authenticity is known and verifiable.

It prevents the malicious software from being executed and therefore enhances the security of the system.

Secure boot is enabled through:

- Access restriction
- Preventing unauthorized firmware execution

6.2.1 Access restriction

Restricts access from entities outside the chip to entities inside the chip.

6.2.1.1 Wi-Fi subsystem

The lower 384 KB and upper 512 KB RAM are write protected from host/SDIO; the host can only write at 384 KB to 512 KB (128-KB size). This design is intended to ensure that the bootloader and firmware (FW) running in the protected memory area are not tampered with by the host, thereby enhancing the system security.

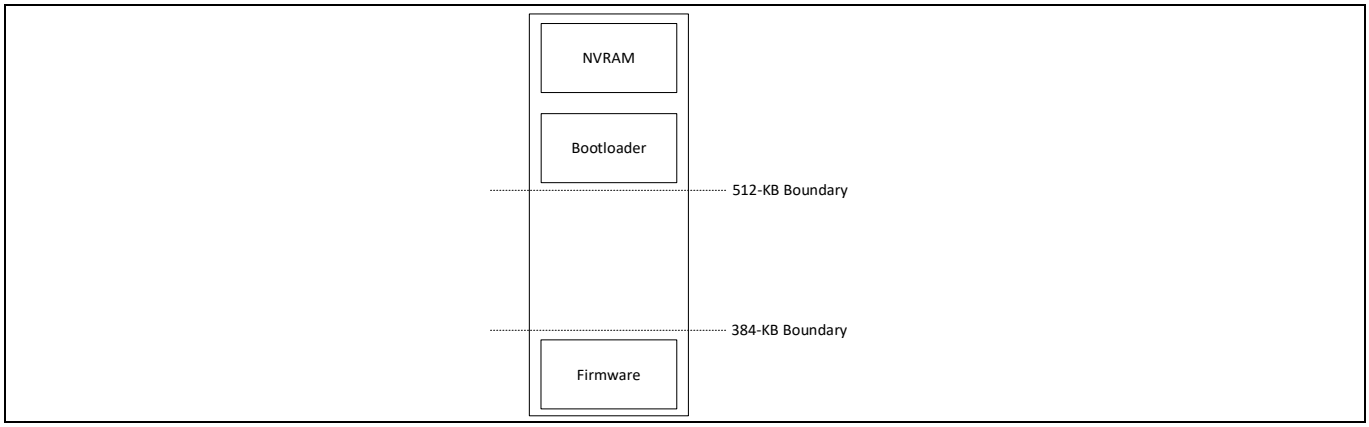


Figure 16 Wi-Fi subsystem RAM (640 KB)

6.2.1.2 Bluetooth® subsystem

During the patch firmware download through UART, Bluetooth® ROM only allows writing to free SRAM and patch RAM areas.

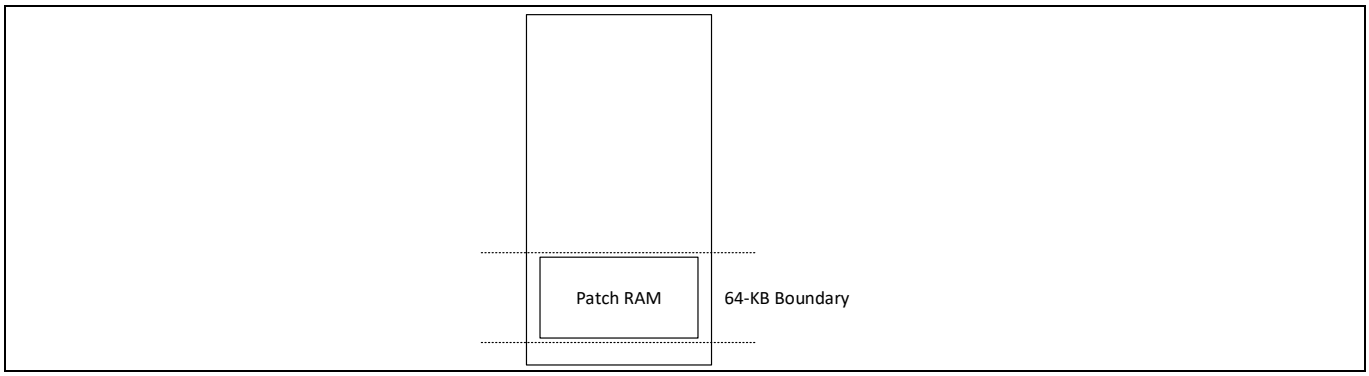


Figure 17 Bluetooth® subsystem RAM (324 KB)

6.2.2 Preventing unauthorized firmware execution

6.2.2.1 Wi-Fi subsystem

- Ensures that the firmware downloaded is authentic and comes from a trusted source. Malicious software can be downloaded from the host; however, it will not be executed until its authenticity is validated.
- Prevents an unauthorized firmware execution by appending a signature validation to the firmware.
- An asymmetric encryption scheme, i.e., a private and public key pair, is used for signature generation. A private key is used to sign the firmware, is confidential, and should not be shared. The public key is openly available, is used to verify the signature of the downloaded firmware, and is present on the OTP.
- After successful firmware signature verification, the downloaded firmware is considered authorized and executed by the bootloader.

Table 2 Security element for Wi-Fi subsystem

Security elements	Value
Signature encryption type	Asymmetric key encryption
Signature validation algorithm	RSA with AES-256 and PKCS1_PSS

6.2.2.2 Bluetooth® subsystem

- When the OTP bit is programmed, Bluetooth® firmware enables secure boot in boot flow.
- When secure boot is enabled, any Patch firmware to be downloaded to the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip should be signed by the remote HSM server.
- In boot sequence, FW will read the secure boot signature in the OTP bit region and verify it with the signature in the Patch FW.
- All the patches should be signed for the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip in secured mode.
- If the patch is not signed or signature verification fails, FW reboots to ROM.

Table 3 Security element for Bluetooth® subsystem

Security elements	Value
Signature encryption type	Asymmetric key encryption
Signature and hash algorithm	ECC256 and SHA256

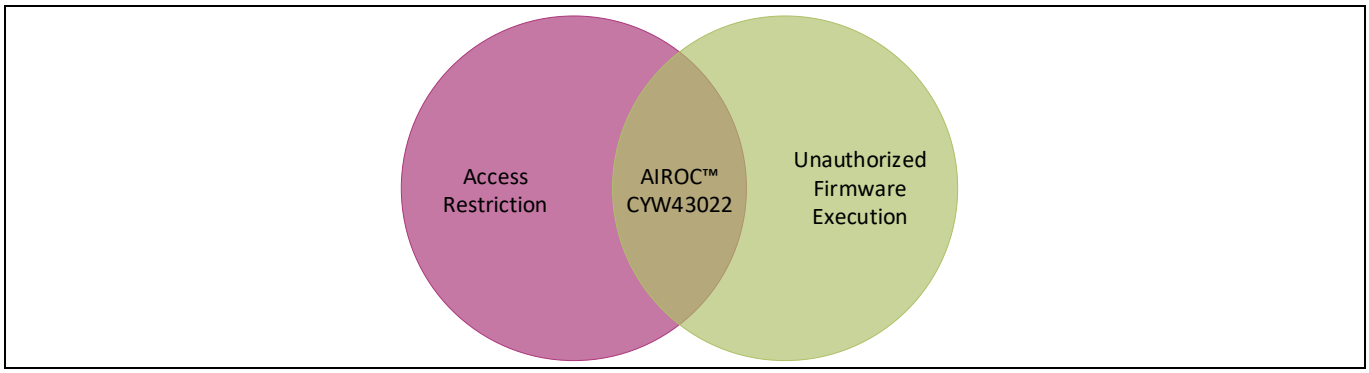


Figure 18 Secure boot

6.3 Device lifecycle

The AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip will employ a set of lifecycle states. These states enhance the level of security and access restrictions. OTP bits indicate the mode of operation, i.e., whether the chip is in secure mode or non-secure mode.

- **Compatibility mode**: Access restriction is absent, i.e., the host writes the whole 640 KB RAM of the Wi-Fi subsystem.
- **Secure mode**: Access restriction is present, i.e., all the lower 384 KB and upper 512 KB of Wi-Fi subsystem RAM are write protected from the host/SDIO. The host can only write at 384 KB to 512 KB (128 KB size).

Various restrictions in secure mode for Bluetooth® and Wi-Fi include:

1. SDIO2WLAN: In compatibility mode, the host can read or write all address regions of the Wi-Fi subsystem RAM, while in secure mode, only part of the Wi-Fi subsystem RAM from 348 KB to 512 KB is accessible to the host to write. This can be configured by setting the OTP bit 22.
2. WLAN2BT: In secure mode, ARMCM3 in the Wi-Fi subsystem cannot read or write to registers in the Bluetooth® subsystem. This can be configured by setting the OTP bit 23.
3. BT2WLAN: In secure mode, ARMCM4 in the Bluetooth® subsystem cannot write to registers in the Wi-Fi subsystem. This can be configured by setting the OTP bit 24.

4. BT SMEM: In compatibility mode, ARMCM4 in the Bluetooth® subsystem can directly access SoCs of the WLAN subsystem, while in secure mode, this shared interface is read or write disabled. This can be configured by setting the OTP bit 25.
5. JTAG: In secure mode, JTAG is disabled, while in compatibility mode, JTAG is enabled for debugging. This can be configured by setting the OTP bit 26.

6.4 Root of Trust (RoT) and bootloader flow (Wi-Fi)

One-time-programmable (OTP) memory is considered the root of trust because it is immutable and has nonvolatile memory; therefore, it can be trusted.

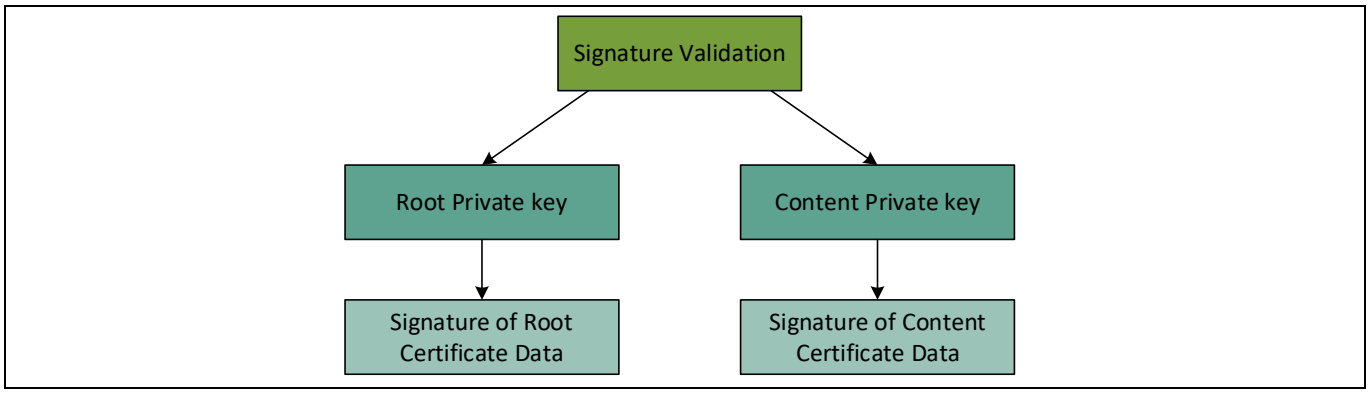


Figure 19 Two level signature mechanism

Instead of a single-level key pair, a 2-level signature mechanism is used, in which:

- The root private key is used to generate the “Signature of Root Certificated Data,” and this signature will be verified by the root public key within the TRXv4 blob at AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip.
- The content private key is used to generate the “Signature of Content Certificate Data,” and this signature will be verified by the content public key within the TRXv4 blob at AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip.
- Signature provides authentication, integrity protection, and non-repudiation for firmware validation.

The following are the steps involved in the secure boot:

1. On Reset, the bootloader entry function is executed.
2. During initialization, the bootloader reads configuration (enable or disable secure boot) from the OTP.
3. The bootloader locks write access to all the Wi-Fi subsystem RAM, except for the 384 KB to 512 KB blocks.
4. The bootloader clears the host handshake registers and indicate that it has started running.
5. The host is expected to download the first block of firmware in a 128 KB window.
6. The bootloader verifies the firmware.
7. If authentication is successful, the bootloader passes control to the firmware.

The following are the steps involved in the firmware validation:

1. The hash is calculated from the root public key that is available in the TRX Header. If this hash of the root public key matches “Hashed Public Key” in the OTP, the root public key in the TRX Header is trusted.

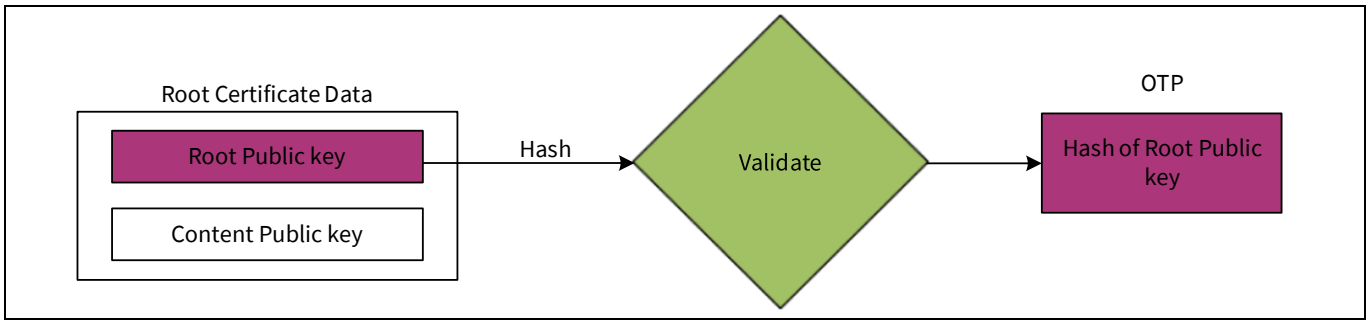


Figure 20 Root public key validation

2. Further, the bootloader calculates the hash value of signature of root certificated data and root public key. This hash value should match the hash value of the root certificated data. If it matches, the contents inside the root certificate data (i.e., root public key and content public key) can be trusted.

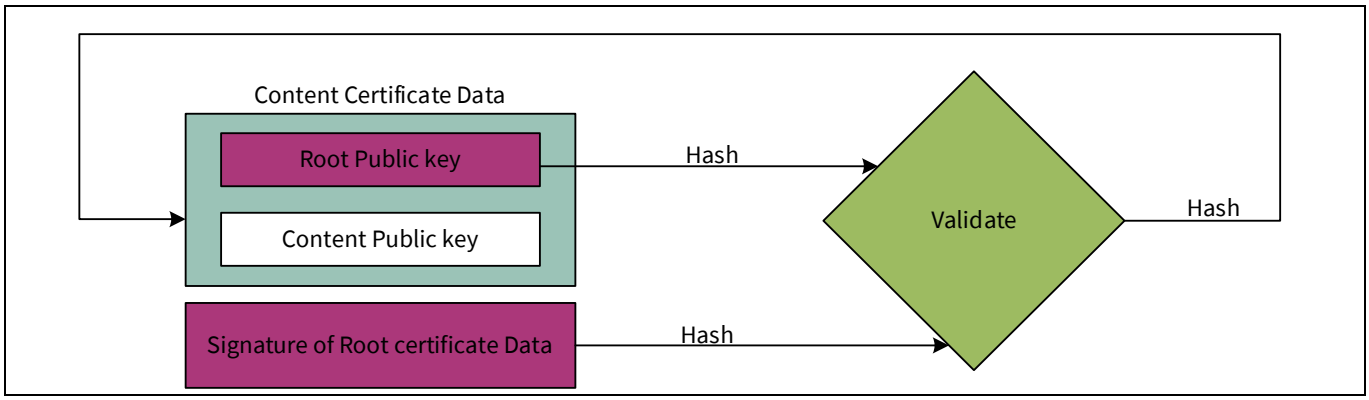


Figure 21 Example figure

3. Because the content public key is verified, we can validate the signature of “Content Certificate Data” by using a cryptographic hash algorithm. After successful validation, “Content Certificate Data” (i.e., Hash of encrypted firmware image, size of encrypted firmware image, and so on.) can be trusted.

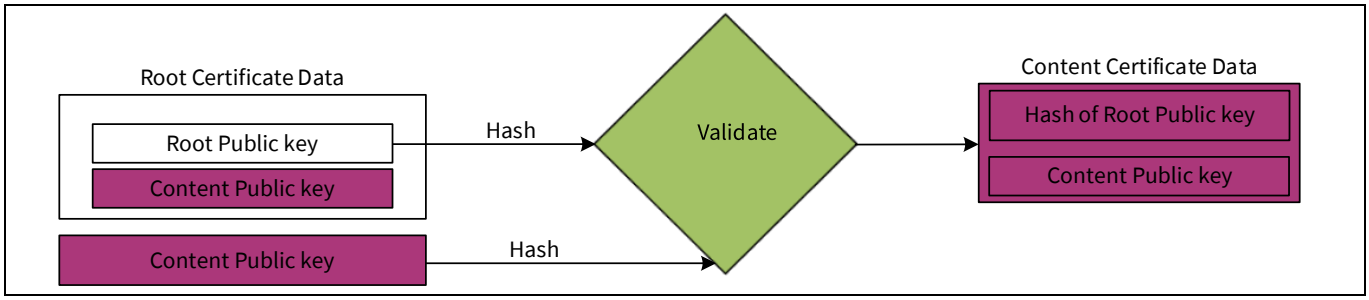


Figure 22 Content certificate data validation

6.5 Root of Trust (RoT) and bootloader flow (Bluetooth®)

RoT is the anchor for all security, and it requires HW support. The Cortex®-M4 core in the Bluetooth® subsystem and the Cortex®-M3 core in the Wi-Fi subsystem provide the necessary encryption services and isolation between system components.

OTP is one-time-programmable memory, which is considered the root of trust because it is immutable and has nonvolatile memory; therefore, it can be trusted.

Bootloader flow in secure boot:

- When the OTP bit 266 is programmed, Bluetooth® FW enables secure boot in boot flow.
- When secure boot is enabled, any patch firmware to be downloaded to the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip should be signed by the remote HSM server.
- In boot sequence, FW will read the secure boot signature in the OTP bit region and verify it with the signature in the patch FW.
- All the patches should be signed for the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip in secured mode.

If the patch is not signed or signature verification fails, FW reboots to ROM.

7 Appendix

7.1 iMX8 Nano image download mode

Do the following to set the iMX8M Nano Developer's Kit V3 board in "Image Download Mode":

1. Short the ISP_ENA jumper to enable the ISP Enabled Mode (see Figure 23).
2. Connect a USB cable between the connector (USB for image download) and the PC.

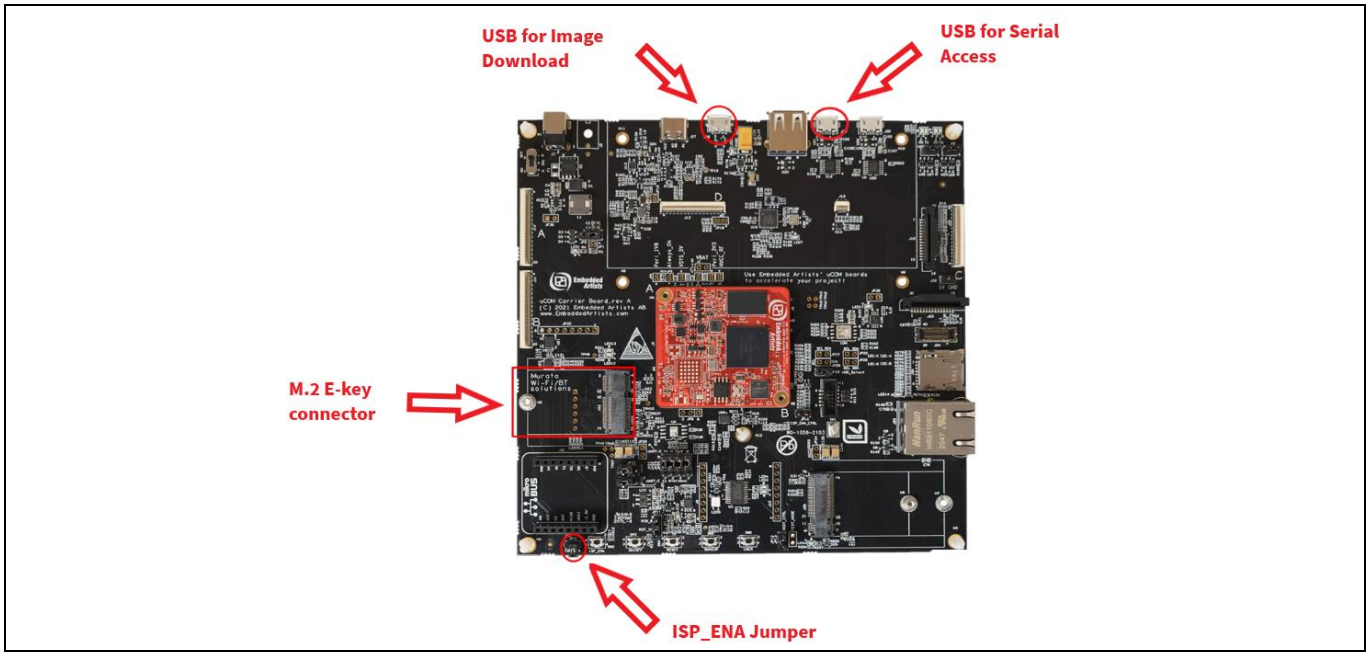


Figure 23 iMX8 connections and ports

7.2 Flashing the pre-build package

1. Download the IMX8 Nano image. This will download *uuu_imx8mn_ucom_5.15.32.zip*.
2. Unzip the *uuu_imx8mn_ucom_5.15.32.zip* image package.
3. Contact the local Infineon FAE or sales representative to get the *CYW43022.zip* release package.
4. Unzip the *CYW43022.zip* release package.
5. Copy the image file and *imx8mn-ea-ucom-kit_v3_ifx_inband.dtb* available in the CYW43022 package folder to *uuu_imx8mn_ucom_5.15.32\files*.
   Copy or replace the *kernel.uuu* file from the CYW43022 package folder to *uuu_imx8mn_ucom_5.15.32*.
6. Put the iMX8 board in "iMX8 Nano image download mode".
7. Power ON the iMX8M Nano Developer's Kit.
8. Use the Universal Update Utility (UUU) tool to download the images to the host board.
   After connecting the hardware to the PC, open the command prompt or terminal and navigate to *uuu_imx8mn_ucom_5.15.32\uuu_imx8mn_ucom_5.15.32*.
9. Use the following command to check whether the board is recognized by the PC (See Figure 24):

```
uuu -lsusb
```

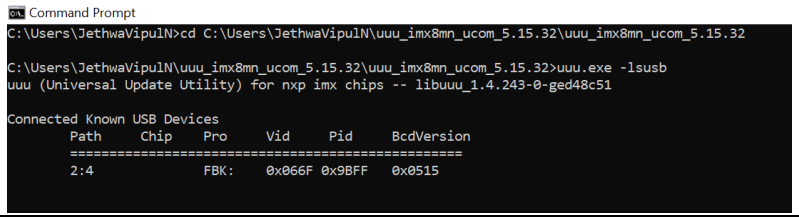


Figure 24 uuu -lsusb command

Note: If the board already has the Linux 5.15.32 kernel, skip step 12 and proceed to step 13.

10. Use the following command to download the full image (see Figure 25):

```
uuu.exe full_wic.uuu
```

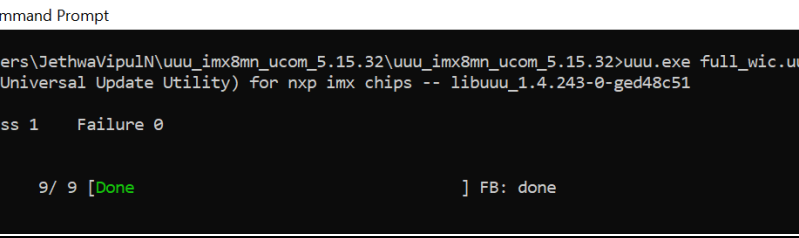


Figure 25 Programing full image using UUU Tool

Note: After running the `uuu.exe` command, power OFF and power ON the board before entering another `uuu.exe` command.

11. Use the following command to download the device tree (see Figure 26):

```
uuu.exe kernel.uuu
```

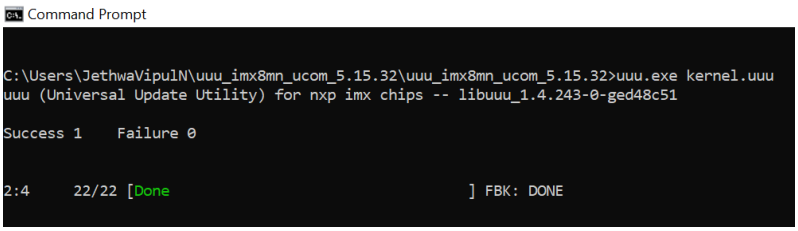


Figure 26 Programing Kernel using UUU tool

12. After the image is downloaded to the iMX8M Nano Developer's Kit, power OFF the device and disconnect the ISP_ENA jumper to bring the board to the normal mode.
13. Connect to the serial port of the hardware and open the COM port in the terminal application at a baud rate of 115200.
14. Power ON the board and press the **Enter** or any key within 3 seconds on the terminal to stop autoboot and enter U-BOOT mode.
15. Enter the following command and confirm that *fdt* is used:

```
u-boot=> printenv fdt_file
fdt_file=imx8mn-ea-ucom-kit_v3.dtb
```

16. Enter the following command to set the dtb file:

    For inbound interrupt:

```
u-boot=> setenv fdt_file imx8mn-ea-ucom-kit_v3_ifx_inband.dtb
```

17. Enter the following command to save the changes:
    ```
    u-boot=> saveenv
    ```
18. Enter the following command to confirm the changes:
    ```
    u-boot=> printenv fdt_file
    ```
19. Enter the following command to continue the booting process:
    ```
    u-boot=> boot
    ```
20. When the booting process is complete, The login window prompts. Enter the following login credentials:
    - Username: **root**
    - Password: **pass**
21. After login, check whether the AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip is connected to the M.2 E-key connector on the iMX8M Nano Developer's Kit V3 board and is recognized by the OS using the `dmesg | grep SDIO` command. It returns the details of the chip with SDIO mode and the multimedia card ID connected.

    For example, use the following command to check that the CYW43022 is connected to mmc0:
    ```
    root@imx8mnea-ucom:/home/CYW43022/Wi-Fi# dmesg | grep SDIO[ 2.687737] mmc0: new high speed SDIO card at address 0001
    ```
22. Use the following command to confirm SDIO settings.
    ```
    root@imx8mnea-ucom:~# cat /sys/kernel/debug/mmc0/ios
    ```

See the following example:

Code Listing 1 SDIO mmc details

```
root@imx8mnea-ucom:CYW43022/Wi-Fi# cat /sys/kernel/debug/mmc0/ios
clock: 50000000 Hz
actual clock: 50000000 Hz
vdd: 21 (3.3 ~ 3.4 V)
bus mode: 2 (push-pull)
chip select: 0 (don't care)
power mode: 2 (on)
bus width: 2 (4 bits)
timing spec: 2 (sd high-speed)
signal voltage: 1 (1.80 V)
driver type: 0 (driver type B)
```

7.3 Bring up AIROC™ CYW43022 Wi-Fi & Bluetooth® combo chip toolchain for custom image

7.3.1 Linux host setup

The Yocto build system requires a Linux host machine. The minimum available hard disk space is 50 GB; however, it is recommended that the host machine has at least 120 GB to be able to build the largest image or distribution.

The instructions in this document are tested on an Ubuntu 18.04.

7.3.2 Required packages

The Yocto project requires the following packages to be installed on the host machine:

```
sudo apt-get update
sudo apt-get install openssh-server

sudo service ssh restart

sudo apt-get update

sudo apt-get install gawk wget git-core diffstat unzip texinfo \
gcc-multilib build-essential chrpath socat \
libsdl1.2-dev xterm sed cvs subversion \
coreutils texi2html docbook-utils python-pysqlite2 help2man make \
gcc g++ desktop-file-utils libgl1-mesa-dev libglu1-mesa-dev \
mercurial autoconf automake groff curl lzop asciidoc u-boot-tools
sudo locale-gen en_US.UTF-8
```

1. Use the following command to create a directory named *bin* in the home folder.
   ```
   mkdir ~/bin
   ```
2. Download the tool using the following command:
   ```
   curl http://commondatastorage.googleapis.com/git-repodownloads/repo > ~/bin/repo
   ```
3. Use the following command to execute the tool:
   ```
   chmod a+x ~/bin/repo
   ```
4. Add the */bin* directory to the PATH variable. Add the following line to the *.bashrc* file so the path is available in each started shell or terminal:
   ```
   echo "export PATH=~/bin:$PATH" >> ~/.bashrc
   source ~/.bashrc
   ```

7.3.3 Download Yocto recipes

The Yocto project consists of many recipes used when building an image. These recipes come from several repositories; the repo tool is used to download these repositories. A branch must be selected of the *ea-yocto-base* repository. Table 4 table lists the available branches.

Table 4 Branches of ea-yocto-base repo

Branch name	Description
ea-5.15.32	u-boot: 2022.04. Linux: 5.15.32
ea-5.10.72	u-boot: 2021.04. Linux: 5.10.72
ea-5.10.35	u-boot: 2021.04. Linux: 5.10.35
ea-5.4.47	u-boot: 2020.04. Linux: 5.4.47
ea-5.4.24	u-boot: 2020.04. Linux: 5.4.24
ea-4.14.98	u-boot: 2018.03. Linux: 4.14.98
ea-4.14.78	u-boot: 2018.03. Linux: 4.14.78
ea-4.9.123	u-boot: 2017.03. Linux: 4.9.123
ea-4.9.11_1.0.0	u-boot: 2016.03. Linux: 4.9.11
ea-4.1.15_2.0.0	u-boot: 2016.03. Linux: 4.1.15

1. Create a directory for the downloaded files (*ea-bsp* in the following example):
   ```
   mkdir ea-bsp
   cd ea-bsp
   ```
2. Configure Git if it is not configured. Change “Your name” to your actual name and "Your e-mail" to your e-mail address.
   ```
   git config --global user.name "Your name"
   git config --global user.email "Your e-mail"
   ```
3. Initialize the repo. The file containing all required repositories is downloaded in this step. Change <*selected branch*> to a branch name according to Table 4.
   This example uses *ea-5.15.32 branch.*
   ```
   repo init -u https://github.com/embeddedartists/ea-yocto-base -b ea-5.15.32
   ```
4. Start to download the files:
   ```
   repo sync
   ```

All files are now downloaded into the *ea-bsp* directory. Most of the files will actually be available in the subdirectory called *sources*.

7.3.4 Initialize the build

1. Enter the following command to set up the yocto build. It automatically changes the directory to *build-imx8mqea-com-wayland:*
   ```
   DISTRO=fsl-imx-xwayland MACHINE=imx8mnea-ucom source ea-setup-release.sh -b build-imx8mnea-ucom-xwayland
   ```
2. Edit *conf/local.conf* to enable the SSH server:
   ```
   - EXTRA_IMAGE_FEATURES = "debug-tweaks"
   + EXTRA_IMAGE_FEATURES = "debug-tweaks ssh-server-openssh"
   ```

7.3.5 Custom iMX toolchain

1. Build an image to create the toolchain. See the following example:

```
bitbake meta-toolchain
```

This creates a file *build-imx8mnea-ucom-xwayland/tmp/deploy/sdk*. Go to the *cd build-imx8mnea-ucom-xwayland/tmp/deploy/sdk* location to install the toolchain.

2. Use the following command to install the toolchain:

```
sudo ./fsl-imx-xwayland-glibc-x86_64-meta-toolchain-armv8a-imx8mnea-ucom-toolchain-5.15-kirkstone.sh
```

This results in the toolchain being installed in */opt/fsl-imx-xwayland/5.15-kirkstone*.

3. Source the environment variables using the following command:

```
source /opt/fsl-imx-xwayland/5.15-kirkstone/environment-setup-armv8a-poky-linux
```

7.3.6 Building the image

Now, the setup is completed and start the build. The building of the ea-image-base image is demonstrated in the following example:

```
bitbake ea-image-base
```

Note that creating an image may take many hours depending on the host computer's capability.

The build output directory appears in *~/ea-bsp/build-imx8mnea-ucom xwayland/tmp/deploy/images/imx8mnea-ucom.*

7.3.7 Building the kernel zImage

1. For the 5.15.32 kernel, use the following commands to download the kernel source code from iMX:

```
$ git clone https://github.com/embeddedartists/linux-imx.git
$ cd linux-imx
$ git checkout ea_5.15.y
```

2. Use the following command to set up the build environment:

```
$ source /opt/fsl-imx-xwayland/5.15-kirkstone/environment-setup-armv8a-poky-linux
```

3. Run the following command to avoid linker errors:

```
$ unset LDFLAGS
```

4. Apply the following patches:

```
diff --git a/arch/arm64/boot/dts/freescale/imx8mn-ea-ucom-kit_v3.dts
b/arch/arm64/boot/dts/freescale/imx8mn-ea-ucom-kit_v3.dts
index ad7479ba7db5..f55c107e1386 100644
--- a/arch/arm64/boot/dts/freescale/imx8mn-ea-ucom-kit_v3.dts
+++ b/arch/arm64/boot/dts/freescale/imx8mn-ea-ucom-kit_v3.dts
@@ -27,13 +27,15 @@
      };

      modem_reset: modem-reset {
-         compatible = "gpio-reset";
+/*         compatible = "gpio-reset";
          reset-gpios = <&gpio_buff 0 GPIO_ACTIVE_LOW>;
          initially-in-reset;
```

```
            reset-delay-us = <2000>;
            reset-post-delay-ms = <100>;
            #reset-cells = <0>;
-       };
+*/
+       };
+

        regulators {
            compatible = "simple-bus";
@@ -275,7 +277,7 @@
        assigned-clocks = <&clk IMX8MN_CLK_UART1>;
        assigned-clock-parents = <&clk IMX8MN_SYS_PLL1_80M>;

-       resets = <&modem_reset>;
+       /* resets = <&modem_reset>;*/
        fsl,uart-has-rtscts;
    };

@@ -309,7 +311,8 @@
        #address-cells = <1>;
        #size-cells = <0>;
        pinctrl-names = "default", "state_100mhz", "state_200mhz";
-       pinctrl-0 = <&pinctrl_usdhc1>, <&pinctrl_usdhc1_gpio>;
+       pinctrl-names = "default";
+           pinctrl-0 = <&pinctrl_usdhc1>, <&pinctrl_usdhc1_gpio>;
        pinctrl-1 = <&pinctrl_usdhc1_100mhz>, <&pinctrl_usdhc1_gpio>;
        pinctrl-2 = <&pinctrl_usdhc1_200mhz>, <&pinctrl_usdhc1_gpio>;
        keep-power-in-suspend;
@@ -321,9 +324,9 @@
        brcmf: bcrmf@1 {
            reg = <1>;
            compatible = "brcm,bcm4329-fmac";
-           interrupt-parent = <&gpio2>;
-           interrupts = <9 IRQ_TYPE_LEVEL_LOW>;  /* WL_HOST_WAKE =
GPIO2_IO09 active low for M.2. */
-           interrupt-names = "host-wake";
+           #interrupt-parent = <&gpio2>;
+           #interrupts = <9 IRQ_TYPE_LEVEL_LOW>;  /* WL_HOST_WAKE =
GPIO2_IO09 active low for M.2. */
+           #interrupt-names = "host-wake";
        };
    };
diff --git a/arch/arm64/configs/ea_imx8_defconfig
b/arch/arm64/configs/ea_imx8_defconfig
index b51cae3c3e24..89f91b6dfa8e 100644
--- a/arch/arm64/configs/ea_imx8_defconfig
+++ b/arch/arm64/configs/ea_imx8_defconfig
@@ -517,7 +517,7 @@ CONFIG_LOGO=y
 CONFIG_SOUND=y
 CONFIG_SND=y
 CONFIG_SND_DYNAMIC_MINORS=y
-CONFIG_SND_USB_AUDIO=m
+CONFIG_SND_USB_AUDIO=y
 CONFIG_SND_SOC=y
```

```
 CONFIG_SND_IMX_SOC=y
 CONFIG_SND_SOC_IMX_AK4458=y
@@ -824,3 +824,18 @@ CONFIG_CORESIGHT_SOURCE_ETM4X=y
 CONFIG_MEMTEST=y
 CONFIG_TOUCHSCREEN_GOODIX=y
 CONFIG_DRM_PANEL_RAYDIUM_RM68200=y
+CONFIG_CFG80211=m
+CONFIG_BCMDHD=n
+CONFIG_USB_SERIAL=y
+CONFIG_USB_SERIAL_CONSOLE=y
+CONFIG_USB_SERIAL_GENERIC=y
+CONFIG_USB_SERIAL_FTDI_SIO=y
+CONFIG_ASYMMETRIC_KEY_TYPE=y
+CONFIG_ASYMMETRIC_PUBLIC_KEY_SUBTYPE=y
+CONFIG_X509_CERTIFICATE_PARSER=y
+CONFIG_PKCS7_MESSAGE_PARSER=y
+#3.1 Disable cfg80211 regdb for the kernel above v5.4.18
+CONFIG_CFG80211_REQUIRE_SIGNED_REGDB=n
+CONFIG_CFG80211_USE_KERNEL_REGDB_KEYS=n
+CONFIG_LOCALVERSION="IFX_FMAC"
+CONFIG_LOCALVERSION_AUTO=y
diff --git a/drivers/mmc/core/core.c b/drivers/mmc/core/core.c
index 368f10405e13..da4258758d4d 100644
--- a/drivers/mmc/core/core.c
+++ b/drivers/mmc/core/core.c
@@ -2333,3 +2333,16 @@ subsys_initcall(mmc_init);
 module_exit(mmc_exit);

 MODULE_LICENSE("GPL");
+/*
+ * Select SD Clock and SD Mode parameters for host.
+ */
+void mmc_set_sdclk(struct mmc_host *host, unsigned int flags)
+{
+    /* PM caps are used for propagation parameters from
+     * MMC to SDHCI Layer
+     */
+        // pr_err("flags = %d\n",flags);
+    host->pm_caps |= flags;
+    /* Call SDHCI interface function from MMC */
+    mmc_set_ios(host);
+}
diff --git a/drivers/mmc/core/core.h b/drivers/mmc/core/core.h
index f5f3f623ea49..bde1650076fc 100644
--- a/drivers/mmc/core/core.h
+++ b/drivers/mmc/core/core.h
@@ -59,6 +59,7 @@ void mmc_power_off(struct mmc_host *host);
 void mmc_power_cycle(struct mmc_host *host, u32 ocr);
 void mmc_set_initial_state(struct mmc_host *host);
 u32 mmc_vddrange_to_ocrmask(int vdd_min, int vdd_max);
+void mmc_set_sdclk(struct mmc_host *host, unsigned int flags);

 static inline void mmc_delay(unsigned int ms)
 {
diff --git a/drivers/mmc/core/sdio_io.c b/drivers/mmc/core/sdio_io.c
```

```
index 79dbf90216b5..a82c43eeb6d7 100644
--- a/drivers/mmc/core/sdio_io.c
+++ b/drivers/mmc/core/sdio_io.c
@@ -812,3 +812,46 @@ void sdio_retune_release(struct sdio_func *func)
     mmc_retune_release(func->card->host);
 }
 EXPORT_SYMBOL_GPL(sdio_retune_release);
+/**
+ *    sdio_set_sdclk - set SD clock enable/disable and sd_mode
+ *    @func: SDIO function attached to host
+ */
+int sdio_set_sdclk(struct sdio_func *func, unsigned int flags)
+{
+    struct mmc_host *host;
+    u8 ctrl;
+    int ret = 0;
+
+    if (WARN_ON(!func))
+        return -EINVAL;
+    host = func->card->host;
+
+        // pr_err("flags =%d\n",flags);
+
+    if ((flags == SDIO_IDLECLOCK_DIS) || (flags == SDIO_IDLECLOCK_EN)) {
+        /* Switch OFF/ON SD CLOCK in sdio Host Controller */
+        mmc_set_sdclk(host, flags);
+    } else if ((flags == SDIO_SDMODE_1BIT) || (flags ==
SDIO_SDMODE_4BIT)) {
+        ret = mmc_io_rw_direct(func->card, 0, 0,
+                SDIO_CCCR_IF, 0, &ctrl);
+        /* Check for Error */
+        if (ret)
+            return ret;
+
+        /* Clear first two bits
+         * 00 - 1 bit wide
+         * 10 - 4 bit wide
+         */
+        ctrl &= ~SDIO_BUS_WIDTH_MASK;
+        /* set as 4-bit bus width */
+        if (flags == SDIO_SDMODE_4BIT)
+            ctrl |= SDIO_BUS_WIDTH_4BIT;
+
+        ret = mmc_io_rw_direct(func->card, 1, 0,
+                SDIO_CCCR_IF, ctrl, NULL);
+        /* Update HOST CTRL register with 1 bit or 4 bit mode */
+        mmc_set_sdclk(host, flags);
+    }
+    return ret;
+}
+EXPORT_SYMBOL_GPL(sdio_set_sdclk);
diff --git a/drivers/mmc/host/sdhci.c b/drivers/mmc/host/sdhci.c
index fc637b9716d4..61be2fd4eab2 100644
--- a/drivers/mmc/host/sdhci.c
+++ b/drivers/mmc/host/sdhci.c
```

```
@@ -2262,6 +2262,37 @@ void sdhci_set_ios(struct mmc_host *mmc, struct
mmc_ios *ios)
     struct sdhci_host *host = mmc_priv(mmc);
     u8 ctrl;

+    /* 43012 WAR: To Support SD Clock to avoid back powering from
+     *  HOST Controller
+     */
+    u16 clk;
+
+    if (mmc->pm_caps & SDIO_IDLECLOCK_EN) {
+        /* Stop SD Clock  */
+        clk = sdhci_readw(host, SDHCI_CLOCK_CONTROL);
+        clk &= ~SDHCI_CLOCK_CARD_EN;
+        sdhci_writew(host, clk, SDHCI_CLOCK_CONTROL);
+        mmc->pm_caps &= ~SDIO_IDLECLOCK_EN;
+        return;
+    } else if (mmc->pm_caps & SDIO_IDLECLOCK_DIS) {
+        /* Start SD Clock  */
+        clk = sdhci_readw(host, SDHCI_CLOCK_CONTROL);
+        clk |= SDHCI_CLOCK_CARD_EN;
+        sdhci_writew(host, clk, SDHCI_CLOCK_CONTROL);
+        mmc->pm_caps &= ~SDIO_IDLECLOCK_DIS;
+        return;
+    } else if (mmc->pm_caps & SDIO_SDMODE_1BIT) {
+        /* Enable 1 bit bus mode  */
+        sdhci_set_bus_width(host, MMC_BUS_WIDTH_1);
+        mmc->pm_caps &= ~SDIO_SDMODE_1BIT;
+        return;
+    } else if (mmc->pm_caps & SDIO_SDMODE_4BIT) {
+        /* Enable 4 bit bus mode  */
+        sdhci_set_bus_width(host, MMC_BUS_WIDTH_4);
+        mmc->pm_caps &= ~SDIO_SDMODE_4BIT;
+        return;
+    }
+
     if (ios->power_mode == MMC_POWER_UNDEFINED)
         return;

diff --git a/include/linux/mmc/pm.h b/include/linux/mmc/pm.h
index 3549f8045784..6b19c21e1438 100644
--- a/include/linux/mmc/pm.h
+++ b/include/linux/mmc/pm.h
@@ -24,4 +24,10 @@ typedef unsigned int mmc_pm_flag_t;
 #define MMC_PM_KEEP_POWER    (1 << 0)    /* preserve card power during
suspend */
 #define MMC_PM_WAKE_SDIO_IRQ    (1 << 1)    /* wake up host system on
SDIO IRQ assertion */

+/* SDIO IDLECLOCK Support - reusing pm_caps */
+#define SDIO_IDLECLOCK_DIS    (1 << 2)    /* Start SDClock */
+#define SDIO_IDLECLOCK_EN    (1 << 3)    /* Stop SDClock */
+#define SDIO_SDMODE_1BIT    (1 << 4)    /* Set 1-bit Bus mode */
+#define SDIO_SDMODE_4BIT    (1 << 5)    /* Set 4-bit Bus mode */
+
```

```
 #endif /* LINUX_MMC_PM_H */
diff --git a/include/linux/mmc/sdio_func.h
b/include/linux/mmc/sdio_func.h
index 478855b8e406..ce99d11e9466 100644
--- a/include/linux/mmc/sdio_func.h
+++ b/include/linux/mmc/sdio_func.h
@@ -175,4 +175,5 @@ extern void sdio_retune_crc_enable(struct sdio_func
*func);
 extern void sdio_retune_hold_now(struct sdio_func *func);
 extern void sdio_retune_release(struct sdio_func *func);

+extern int sdio_set_sdclk(struct sdio_func *func, unsigned int flags);
 #endif /* LINUX_MMC_SDIO_FUNC_H */
```

5. Use the following commands to disable cfg80211:

Note: This command is applicable only for kernels above v5.4.18.

```
#3.1 Disable cfg80211 regdb for the kernel above v5.4.18
CONFIG_CFG80211_REQUIRE_SIGNED_REGDB=n
CONFIG_CFG80211_USE_KERNEL_REGDB_KEYS=n
```

6. Build the kernel using the following commands:

```
$ make clean
$ make ea_imx8_defconfig
$ make
```

The built kernel is available in *arch/arm64/boot/Image.*

7.4 Building the FMAC driver from backports

1. Contact a local Infineon FAE or sales representative to get the *CYW43022.zip* release package.
2. Build the FMAC driver using the following commands:

```
cd ea-bsp/kernel/linux-imx
$ copy the fmac zip file to the machine
$ unzip cypress-fmac-*.zip
$ tar zxvf cypress-backports-*.tar.gz
$ cd v5.15-backports
```

3. Apply the following patch to build the 4-bit driver:

```
--- v5.15.58-
backports/drivers/net/wireless/broadcom/brcm80211/brcmfmac/sdio.c  2023-06-
20 07:48:52.000000000 +0530
+++ sdio_imx8.c 2023-06-26 12:08:50.394292547 +0530
@@ -1183,7 +1183,7 @@
                        if (bus->idleclock == BRCMF_IDLE_STOP)
                                brcmf_sdio_set_sdbus_clk_width(bus,
-                                                              SDIO_SDMODE_1
BIT);
+                                                              SDIO_SDMODE_4
BIT);
                        err = brcmf_sdio_kso_control(bus, false);
```

4. Open a new terminal and run the following command (note that do not source the Yocto toolchain before you run the following command):

```
$ MY_KERNEL=<kernel path>
# Example: MY_KERNEL= ea-bsp/kernel/linux-imx
$ make KLIB=$MY_KERNEL KLIB_BUILD=$MY_KERNEL defconfig-brcmfmac
```

5. Source the Yocto toolchain (for cross compile) and build the FMAC driver:

```
$ source /opt/fsl-imx-xwayland/5.15-kirkstone/environment-setup-armv8a-poky-
linux
$ unset LDFLAGS
$ make KLIB=$MY_KERNEL KLIB_BUILD=$MY_KERNEL modules
```

The output modules can be found in the following files:

- *compat/compat.ko*
- *net/wireless/cfg80211.ko*
- *drivers/net/wireless/broadcom/brcm80211/brcmutil/brcmutil.ko*
- *drivers/net/wireless/broadcom/brcm80211/brcmfmac/brcmfmac.ko*

7.4.1 Debugging the FMAC driver

1. Enable the debug feature in backports source code. Enable CPTCFG_BRCMDBG and CONFIG_DEBUG_FS in the following *brcmfmac* file:

   ```
   $ vi defconfigs/brcmfmac
   ```

2. Add the following lines to the *brcmfmac* file:

   ```
   $ CPTCFG_BACKPORTED_DEBUG_INFO=y
   $ CPTCFG_BRCM_TRACING=y
   $ CPTCFG_BRCMDBG=y
   ```

3. Rebuild the FMAC driver. For more details, see the Building the FMAC driver from backports section.
4. Enable the brcmfmac debug log using the following commands:

   ```
   $ echo 8 > /proc/sys/kernel/printk
   $ insmod brcmfmac.ko debug=0x00100006      (to enable TRACE, INFO, and
   WIFI_FW_LOG, for example)
   ```

   The following message levels are listed in *drivers/net/wireless/broadcom/brcm80211/brcmfmac/debug.h.*

   ```
   /* message levels */
   #define BRCMF_TRACE_VAL 0x00000002
   #define BRCMF_INFO_VAL 0x00000004
   #define BRCMF_DATA_VAL 0x00000008
   #define BRCMF_CTL_VAL 0x00000010
   #define BRCMF_TIMER_VAL 0x00000020
   #define BRCMF_HDRS_VAL 0x00000040
   #define BRCMF_BYTES_VAL 0x00000080
   #define BRCMF_INTR_VAL 0x00000100
   #define BRCMF_GLOM_VAL 0x00000200
   #define BRCMF_EVENT_VAL 0x00000400
   #define BRCMF_BTA_VAL 0x00000800
   #define BRCMF_FIL_VAL 0x00001000
   #define BRCMF_USB_VAL 0x00002000
   #define BRCMF_SCAN_VAL 0x00004000
   #define BRCMF_CONN_VAL 0x00008000
   #define BRCMF_BCDC_VAL 0x00010000
   #define BRCMF_SDIO_VAL 0x00020000
   #define BRCMF_MSGBUF_VAL 0x00040000
   #define BRCMF_PCIE_VAL 0x00080000
   #define BRCMF_FWCON_VAL 0x00100000
   ```

5. Use the following command to print the debug messages:

   ```
   dmesg -w
   ```

Glossary

AP
Access Point

LAN
Local Area Network

MMC
Multimedia card

OTG
On the Go

SDIO
Secure Digital Input Output

STA
Station

ULP
Ultra-low-power

USB
Universal Serial Bus

UUU
Universal update utility

WEP
Wired Equivalent Privacy

WLAN
Wireless Local Area Network

WPA
Wi-Fi Protected Access

Revision history

Document revision	Date	Description of changes
**	2023-09-04	Initial release.
*A	2023-09-29	Added the CYW43022 rework for power measurement section.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.
The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc., and any use of such marks by Infineon is under license.

Edition 2023-09-29
Published by

Infineon Technologies AG
81726 Munich, Germany

© 2023 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?
Email: erratum@infineon.com

Document reference
002-38499 Rev. *A

Important notice
The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenheitsgarantie")

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.