Wi-Fi software user guide

About this document

Scope and purpose

This document provides an overview of the building blocks of the Linux 802.11 ecosystem. This document helps you to use Wi-Fi modules conveniently with a host of your choice and configure it based on your application.

Intended audience

This document is intended for those using Infineon Wi-Fi solutions with the Linux host of their choice. It is recommended that you have prior experience with Linux kernel networking or knowledge of the boot flow of a Linux host processor.

Note: See Wi-Fi glossary for terms and acronyms used in this document.

Table of contents

1 Introduction to Wi-Fi software

The Wi-Fi software provides the essential components required to set a Wi-Fi device operational; that is, sending and receiving 802.11 frames over the air. This document helps you to understand the Linux kernel networking subsystems and the components involved in configuring the WLAN from user space such as wpa_supplicant, hostapd, and iw. This document covers the user-space and kernel-space features, and device drivers written or configured to be used by Wi-Fi devices.

1.1 Wi-Fi software architecture block diagram

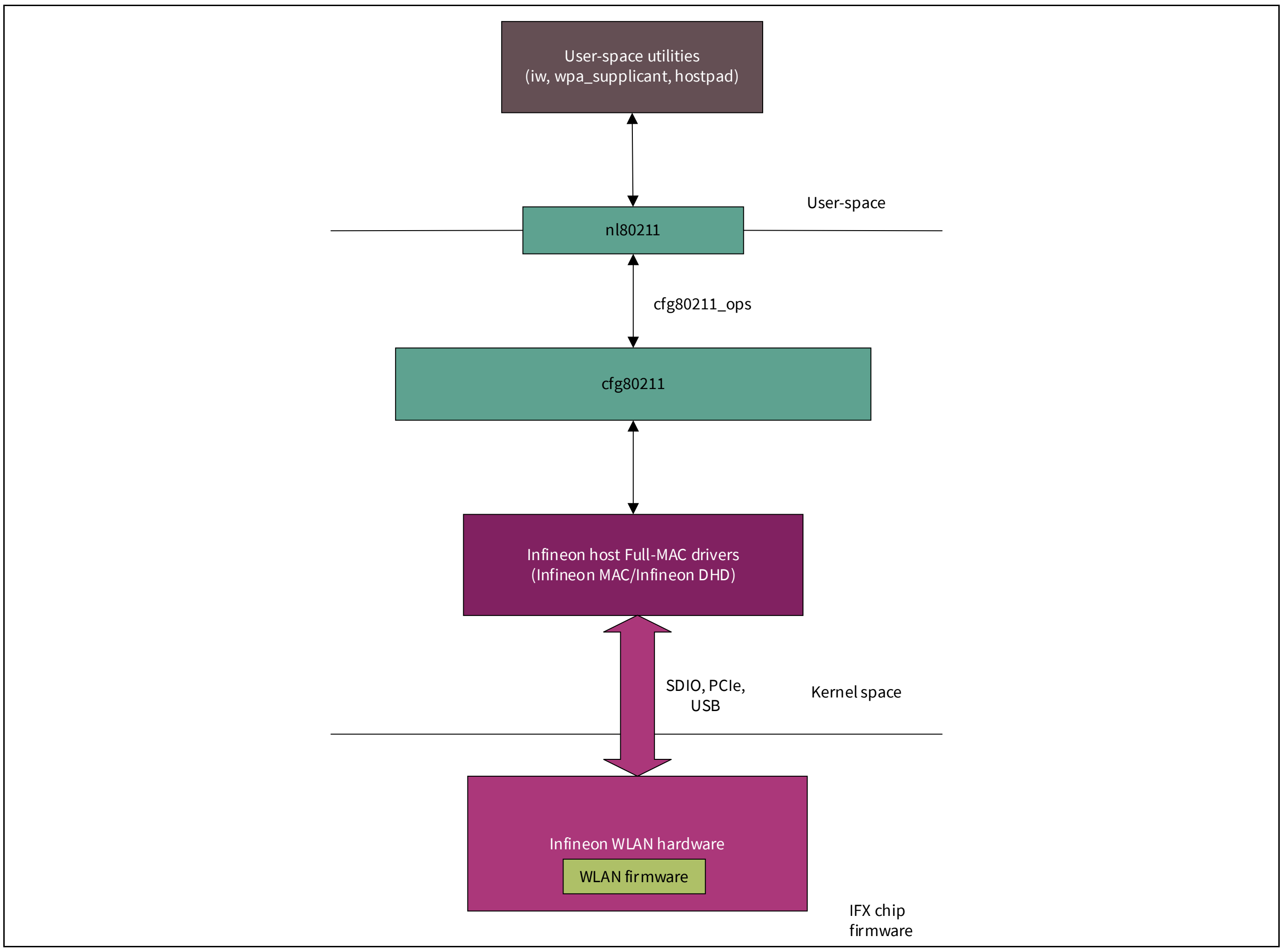


Figure 1 Linux 802.11 architecture – abridged

There is a transmit (TX), receive (RX), and event paths between the applications (the top-most layer where *iw*, *wpa_supplicant*, and *hostapd* belong) and firmware level (embedded within the Infineon Wi-Fi chip) of Wi-Fi software architecture. The intermediate layers employ conditionals for each type of flow; either TX/RX or event. Based on that, the flow control or event queue mechanism between the host and the device is also implemented in the device driver (packaged and provided by Infineon release trains). The hardware layer implements the core 802.11 operations along with a part of the bus hardware. This layer is implemented inside the Infineon Wi-Fi firmware, eliminating the need for writing separate device drivers, thereby reducing the time to market significantly.

2 Platform interface, boot process, and device tree blob

2.1 Hardware connection

The connection between the host processor and the target Wi-Fi radio can be categorized into the following:

- **Bus connection:** The host processor and the target Wi-Fi radio are connected through a bidirectional bus. This provides connections for the following:
  - SDIO: D0, D1, D2, and D3
  - PCIe: TDN, TDP, RDN, and RDP
  - USB: DP, DN, and so on

  For Bluetooth®, the transport usually will be either through SDIO or through UART (connections will be RX, TX, RTS, CTS, and so on).

Table 1 Transport bus combinations between host processor and Wi-Fi/Bluetooth® radios

Connection category	Bus	Pins corresponding to each bus
Host <--> Wi-Fi connection	SDIO	D0-D3
Host <--> Wi-Fi connection	PCIe	TDN, TDP, RDN, and RDP
Host <--> Wi-Fi/Bluetooth® connection	USB	DP, DN
Host <---> Bluetooth® connection	UART	RTS, CTS, TX, RX

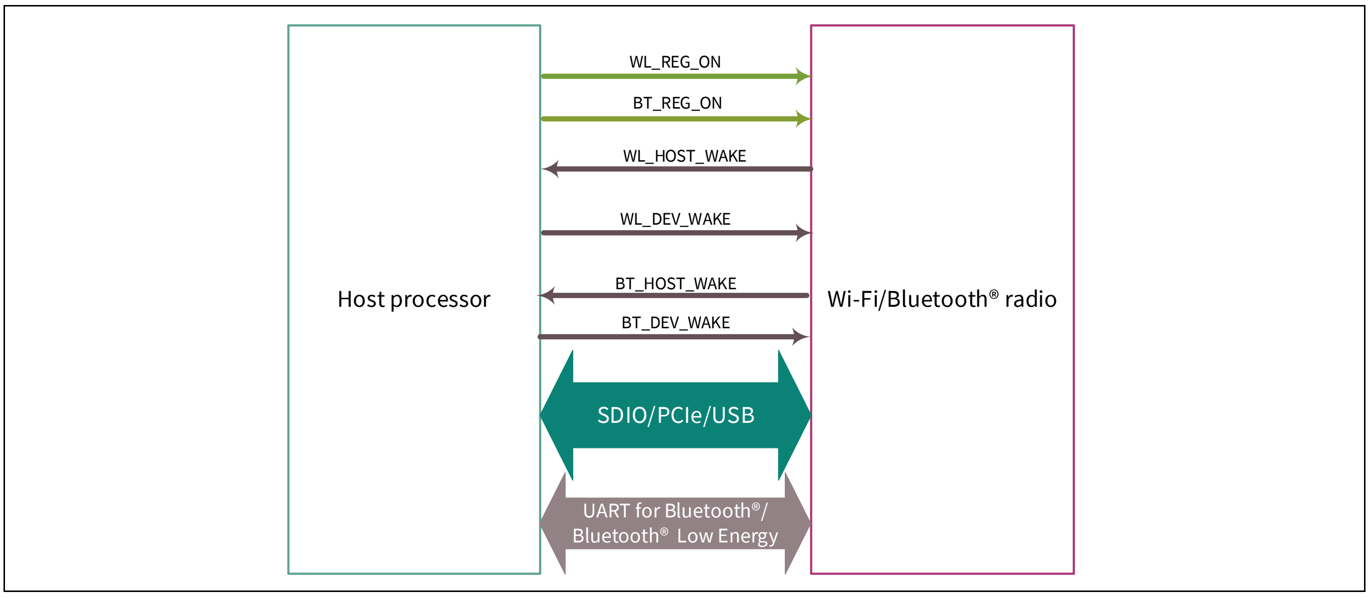


Figure 2 Host processor to Wi-Fi/Bluetooth®/Bluetooth® Low Energy radio connection blocks

- **Power-related connection:** Includes the GPIOs required to power up the Wi-Fi module. WL_REG_ON and BT_REG_ON belong to this category.
- **Signaling connection:** Includes the GPIOs needed to wake the host processor or Wi-Fi module. The responsible pins are WL_HOST_WAKE, WL_DEV_WAKE, BT_HOST_WAKE, and BT_DEV_WAKE.

2.2 Kernel configuration

For kernel compilation, follow the vendor's instructions and set up the source and toolchain. You can find them in the vendor's distribution medium.

1. Based on the target's architecture (ARM64, Arm®, x86, MIPS, and so on), select a default configuration available in *arch/arm/configs.*
2. Issue the following command to configure the kernel with the *.config* file:

```
$ make defconfig
```

3. Edit the *.config* file and build *cfg80211* as a module:

```
# CONFIG_CFG80211=m
```

4. If you are not using legacy Dongle Host Driver(DHD) as the driver, change the following configuration:

```
# CONFIG_BCMDHD=n
```

5. For kernel versions 5.4.18 and above, disable the following *cfg80211 regdb* configurations:

```
# CONFIG_CFG80211_REQUIRE_SIGNED_REGDB=n
# CONFIG_CFG80211_USE_KERNEL_REGDB_KEYS=n
```

6. Additionally, enable the following configurations in the *.config* file:

```
# CONFIG_ASYMMETRIC_KEY_TYPE=y
# CONFIG_ASYMMETRIC_PUBLIC_KEY_SUBTYPE=y
# CONFIG_X509_CERTIFICATE_PARSER=y
# CONFIG_PKCS7_MESSAGE_PARSER=y
```

7. Build the Linux kernel image for the target host. The following command is for i.MX:

```
$ make oldconfig
$ make zImage -j8
```

The kernel image is now available in the *arch/arm/boot/zImage* directory.

Infineon software artifacts available from quarterly train releases are supported, and validated in a variety of platforms (for MMC/SDIO: NXP i.MX6, NXP i.MX8, for PCIe: i.MX8, Intel NUC) in two primary ecosystems:

- Latest Google Android Open Source Platform (AOSP) version (with derivative support for Android TV, Wear OS, and so on)
- Latest Long-term Linux kernel release

Infineon's kernel support policy uses the Backports Project for FMAC driver-based chipset. This enables older kernels to run the newest software. If the kernel version is not the latest LTS, execute the Backports package in the development environment to enable the latest connectivity software (firmware and drivers) in the design. Then, flash the host processor with the modified device tree blob (DTB) and kernel image, and load the (backported) kernel modules (KM).

2.2.1 Device tree blob (DTB) configuration

Device tree provides a way to describe the platform_data or pdata of the hardware that is not inherently discoverable such as I2C and SPI devices. The DTB is typically created and maintained in human-readable formats such as *.dts* source files and *.dtsi* include files. The *.dts* file provides board-level definitions while the *.dtsi* provides SoC-level definitions. The device tree source files are compiled using the Device Tree Compiler (DTC); source files can be found in the *<kernel_base>/scripts/dtc* folder. DTC generates the *.dtb* file, which is also known as the Flattened Device Tree (FDT).

Linux uses the device tree data to find and register the devices in the system. The FDT is accessed in the raw form during the very early phases of boot but is expanded into a kernel internal data structure known as the "expanded device tree" (EDT) for more efficient access during the later phases of the boot and after the system has completed booting. The device tree usually contains the information regarding the I/O port and interrupt lines that the device is required to use. Each device node (representing a platform device in a tree of devices) has a name property that is used to identify the device when the kernel scans through the device tree. In the driver, the "compatible" property specifies the name field that the kernel should look for in the device tree. Once the kernel finds the name, the corresponding device is instantiated and matched with a driver.

The Wi-Fi Linux driver package has a *device tree* folder, which includes i.MX6SX and i.MX6UL device tree blobs. If the host platform is not available in the device tree package, see the existing source files (*.dts* and *.dtsi* files) available in *arch/arm/boot/dts/,* and port the files to your target host platform. The following settings are available for each field of *.dts and .dtsi files*:

- **wlreg_on:** This pin (WL_REG_ON) is responsible for powering up the Wi-Fi device. The pin must be set to active HIGH. See the corresponding chip datasheet for the voltage requirement.
- **In-band/out-of-band:** When the Wi-Fi device is connected over SDIO to the host processor, there are two ways to route the interrupts from the Wi-Fi device to the host:
  - The in-band mechanism of the interrupt uses the SDIO DATA1 line to signal the interrupts.
  - The out-of-band mechanism requires a dedicated GPIO pin. Make sure that the pin multiplexing has been taken care of and the pin is working as a GPIO only.

You can opt for in-band or out-of-band (OOB) depending on the application. To achieve the best low-power numbers, use OOB signaling methods. This is because in this mode, the SDIO bus is put into a suspended mode unless an interrupt is triggered on the WLAN_HOST_WAKE line when a packet is received. If no spare GPIOs are available in the host processor, use the in-band interrupt method where the DATA1 line is repurposed to work as the interrupt. This prevents the bus from being suspended which adds to the power burden.

The following example implementations for i.MX platforms from NXP Semiconductor are available in

linux-imx/arch/arm/boot/dts/:

- *imx6ul-evk-btwifi-oob.dtsi* for the OOB interrupt pin allocation and configuration
- *imx6ul-evk-btwifi.dtsi* for the WL_REG_ON pin-related configuration

For Full MAC (FMAC), the compatible field expands to `of_device_is_compatible(np, "brcm,bcm4329-fmac"))`. For more details on the Linux device tree, see the Linux Device Tree - FMAC webpage.

2.3 WLAN host interface

This section explains the interface options available for connecting a Wi-Fi device to a host processor of your choice.

2.3.1 Multimedia Card (MMC)

The Multimedia Card (MMC) is a low-cost data storage medium, from which the Secure Digital (SD) standard evolved. The I/O card variant combines high-speed serial data input/output lines with low power consumption, making it suitable for battery-powered electronic devices such as IoT devices. The Wi-Fi solution is pre-packaged with the device driver; therefore, you only need to handle the driver implementation in the SDIO Host. SDHC and MMC interfaces in the host must be carefully mapped to an SDIO host interface to communicate with the Wi-Fi chip. Table 2 lists the Wi-Fi chipsets that support the MMC/SDIO interface.

Table 2 Wi-Fi devices with SDIO as host interface

Antenna configuration	802.11 protocol	Infineon Wi-Fi device
1×1 SISO	802.11n	AIROC™ CYW43439
1×1 SISO	802.11ac	AIROC™ CYW43012
1×1 SISO	802.11ac	AIROC™ CYW43022
1×1 SISO	802.11ac	AIROC™ CYW4373
1×1 SISO	802.11ac	AIROC™ CYW43455
1×1 SISO	802.11ax	AIROC™ CYW55500
2×2 MIMO	802.11ac	AIROC™ CYW5459x
2×2 MIMO	802.11ax	AIROC™ CYW5557x

2.3.2 PCIe

Peripheral Component Interconnect Express (PCIe) is a high-speed serial bus that is commonly used as an interface for SSDs, Wi-Fi, Ethernet, and so on. For version or lane-related specifications, see the corresponding chip datasheet. Table 3 lists the chip matrix that supports the PCIe interface.

Table 3 Wi-Fi devices with PCIe as host interface

Antenna configuration	802.11 protocol	Infineon Wi-Fi device
1×1 SISO	802.11ac	AIROC™ CYW43455
2×2 MIMO	802.11ac	AIROC™ CYW5459x
2×2 MIMO	802.11ax	AIROC™ CYW5557x

2.3.3 USB

The USB functionality of this class of devices can range from a storage medium to a Wi-Fi, or Ethernet dongle. In the Wi-Fi portfolio, AIROC™ CYW4373 (1×1 802.11ac) supports the USB interface.

For more details on the WLAN device, see the AIROC™ Wi-Fi + Bluetooth® Combos web page.

3 Linux kernel 802.11 subsystem

3.1 nl80211

The netlink (nl80211) protocol is a socket-based IPC mechanism used for communicating between user space and kernel space or between the user-space processes. It was designed to be a more flexible successor to ioctls to provide mainly kernel-related networking configuration and monitor network interfaces. Table 4 compares the legacy ioctl-based system calls with netlink.

Table 4 Comparison between syscall and netlink

Properties	Netlink sockets	Syscalls
Who can initiate the communication	User-space application and kernel module	User-space application
Multicast?	Yes	No
Require polling?	No	Yes
Asynchronous?	Yes (It provides message queues)	No

The netlink socket operation is simple: you open and register a socket in the user space that handles communications with a kernel netlink socket. The netlink protocol has some advantages over other ways of communication between the user space and kernel. For example, there is no need for polling when working with netlink sockets. A user-space application opens a socket and then calls `recvmsg()`, and enters a blocking state if no messages are sent from the kernel (for example, the `rtnl_listen()` method of the iproute2 package (*lib/libnetlink.c*).

Another advantage is that netlink sockets support multicast transmission. You create netlink sockets from the user space with the `socket()` system call. The netlink sockets can either be `SOCK_RAW` or `SOCK_DGRAM` sockets. Netlink sockets can be created in the kernel or the user space. Kernel netlink sockets are created by the `netlink_kernel_create()` method; user-space netlink sockets are created by the `socket()` system call. Creating a netlink socket from the user space or from kernel creates a `netlink_sock` object.

When the socket is created from user space, it is handled by the `netlink_create()` method. When the socket is created in the kernel, it is handled by `__netlink_kernel_create()`; this method sets the `NETLINK_KERNEL_SOCKET` flag. Eventually, both methods call `__netlink_create()` to allocate a socket commonly (by calling the `sk_alloc()` method) and initializing it.

Figure 3 shows how a netlink socket is created in the kernel and user space.

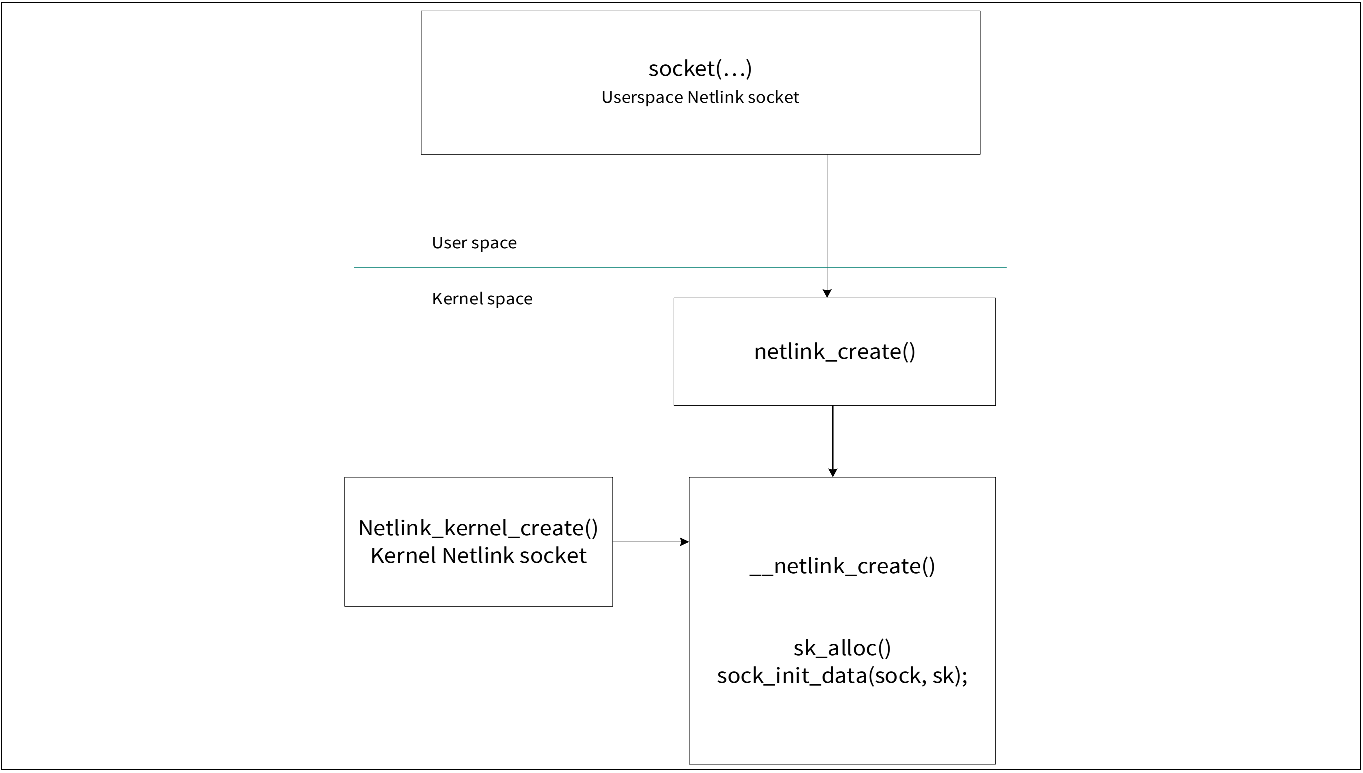


Figure 3 Netlink process flow

The *libnl* package is a collection of libraries providing APIs to the netlink protocol-based Linux kernel interfaces. The *iproute2* package uses the libnl library. Besides the core library (*libnl*), the package includes support for the generic netlink family (*libnl-genl*), routing family (*libnl-route*), and netfilter family (*libnl-nf*).

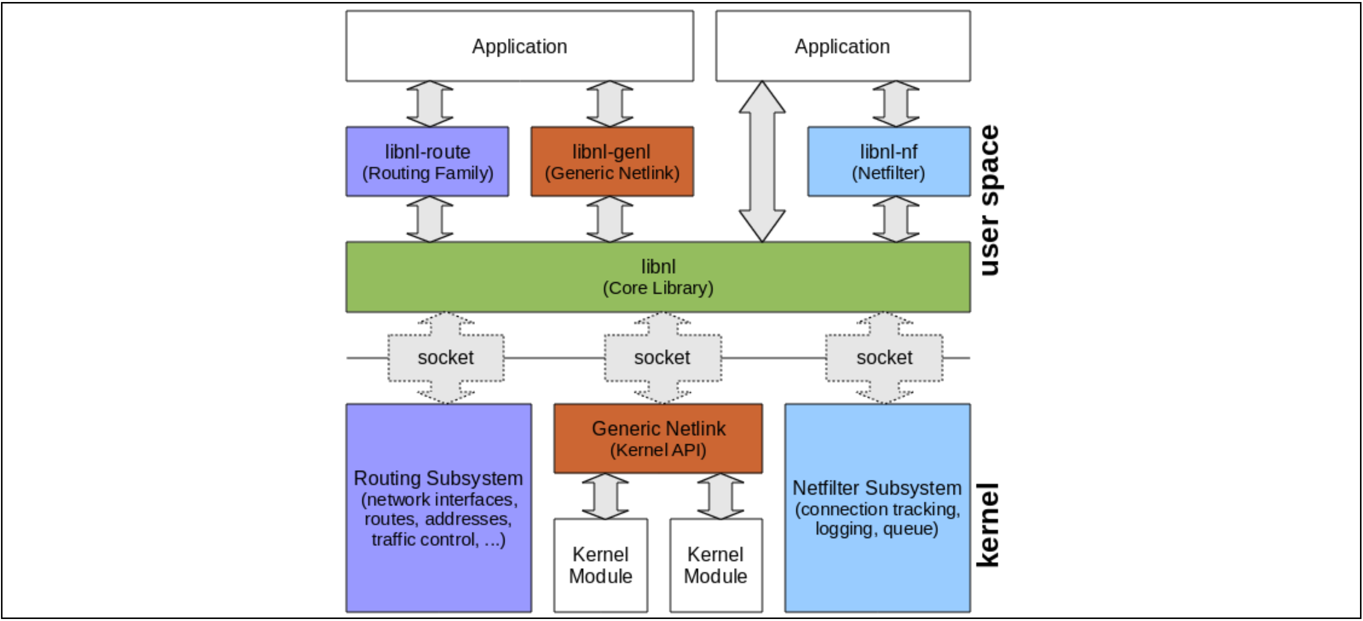


Figure 4 Netlink family

The image source is available at Netlink Protocol Library Suite (libnl).

3.2 cfg80211

CFG80211 is primarily responsible for the configuration of APIs for 802.11 devices in Linux. It provides a management interface between the kernel and user space via nl80211. For backward compatibility, cfg80211 also offers wireless extensions (WEXT) to the user space but abstracts them out from the driver layer completely. Additionally, cfg80211 contains the code that helps to establish the regulatory power constraints and spectrum considerations.

For a driver to use cfg80211, it must register the hardware device with cfg80211. This happens through several hardware capability structs, which are explained in this section. The fundamental structure for each device is the 'wiphy', of which each instance describes a physical wireless device connected to the system.

Each wiphy can have zero, one, or many virtual interfaces associated with it. The associated virtual interface needs to be identified by pointing the network interface's `ieee80211_ptr` pointer to a `struct wireless_dev`, which describes the wireless part of the interface. Normally, this struct is embedded in the network interface's private data area. Drivers can optionally allow creating or destroying virtual interfaces on-the-fly, but without at least one virtual interface or the ability to create some, the wireless device is not useful.

Each wiphy structure contains the device capability information and has a pointer to the various operations the driver offers. It is the Wi-Fi drivers' responsibility to provide the cfg80211 operation callbacks and fill in the wiphy struct to accurately indicate the device's capability.

With Infineon's driver release package, all cfg80211-related operations are already taken care of; you can skip learning about the complexities related to cfg80211 and continue with application development. If you want to customize the driver with additional cfg80211_ops, see *cfg80211.h* for more details on each operation.

3.3 FMAC bringup

FMAC describes a type of wireless card where the mac sublayer management entity (MLME) is managed in hardware. All chips in the Wi-Fi portfolio fall under this category. The FMAC driver was originally introduced in Linux Kernel 2.6+. Because this driver is a part of the Linux kernel, anyone can upstream changes. The following are some key attributes of the FMAC driver:

- Supports SDIO, PCIe, and USB interfaces with a single binary
- Supports major features such as SoftAP, peer-to-peer (P2P), tunneled direct link setup (TDLS), and so on (see the README file provided with the Infineon FMAC driver release package for specifications related to each chip).
- Because FMAC is part of the kernel, supporting different kernel versions becomes easy.

Do the following to build the Infineon FMAC:

1. Download the latest supported Linux kernel source from Infineon GitHub:

```
$ git clone https://github.com/Infineon/ifx-wireless-drivers.git
```

2. Modify the default kernel *.config* and enable the following options, and then compile the kernel image:

```
#CONFIG_BRCMUTIL=y
#CONFIG_BRCMFMAC=y
#CONFIG_BRCMFMAC_SDIO=y
#CONFIG_BRCMFMAC_PROTO_BCDC=y
#CONFIG_BRCMFMAC_PCIE=y
#CONFIG_BRCMFMAC_PROTO_MSGBUF=y
```

3. For the kernel versions above 5.4, add the following extra options:

```
#CONFIG_CFG80211_REQUIRE_SIGNED_REGDB=n
#CONFIG_CFG80211_USE_KERNEL_REGDB_KEYS=n
```

4. There are two options for the firmware of the Wi-Fi chips:

   a) Use the original firmware files in the */lib/firmware/cypress* directory.

   b) Upgrade to the latest firmware available through quarterly releases from Infineon using the following commands:

```
$ git clone https://github.com/Infineon/ifx-linux-firmware.git
$ cp ifx-linux-firmware/firmware/* /lib/firmware/cypress
```

5. Reboot the device with the freshly compiled kernel image and use the latest Wi-Fi features.
6. After rebooting, the `dmesg` command to check the chip ID and additional information such as firmware version, firmware ID, and compilation date.
7. Use the following command to insert all dependent modules that *brcmfmac* needs:

```
modprobe cfg80211
```

8. While loading the driver, pass the parameters listed in Table 5 as arguments to the loadable kernel module (LKM). For example, see the following:

```
$ insmod brcmfmac.ko alternative_fw_path=/etc/firmware/cypress
```

Table 5 FMAC module parameters

Module parameter name	Functionality	Module parameter type
`p2pon`	Enable legacy P2P management functionality	int
`txglomsz`	Maximum Tx packet chain size [SDIO]	int
`debug`	Level of debug output; see Notes on debugging.	int
`feature_disable`	Disable features	int
`alternative_fw_path`	Alternative firmware path. If the firmware is present in a different path other than */lib/firmware/cypress*.	string
`fcmode`	Mode of firmware-controlled flow control	int
`roamoff`	Do not use an internal roaming engine	int
`Iapp`	Enable partial support for the obsoleted inter-access point protocol	int
`ignore_probe_fail`	Always succeed probe for debugging	int

3.3.1 Support for kernels 6.1+

Forward-porting of a kernel involves updating the FMAC source code of the kernel to work with the new version. This may involve making changes to the kernel's interface with the operating system, as well as updating drivers and other kernel modules to work with new hardware or software configurations.

To cross-compile the FMAC on newer versions of the kernel, the build system should have a toolchain suitable for the platform (usually provided in the BSP). A prerequisite is downloading FMAC patches (download the FMAC driver from Infineon Community or contact the local Infineon FAE/sales representative to get a release package).

Do the following to enable the setup with the newer kernel version:

1. Download the 6.1 + kernel source code specific to the platform used. For example, on the i.MX8:

```
$ git clone https://github.com/embeddedartists/linux-imx.git
$ cd linux-imx
$ git checkout ea_6.1.y
```

2. Set up the tool chain specific to the platform's kernel version. For example, on i.MX8, follow the procedure to get the Toolchain for i.MX8.
3. Download the FMAC patches and apply them.
4. Add the flags necessary for the FMAC in defconfig and fix the API compatibility issues caused due to kernel upgrade. For example, on i.MX8:

```
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;

diff --git a/arch/arm64/configs/ea_imx8_defconfig
b/arch/arm64/configs/ea_imx8_defconfig
index 2a3fa3d95531..6917a8014d4c 100644
--- a/arch/arm64/configs/ea_imx8_defconfig
+++ b/arch/arm64/configs/ea_imx8_defconfig
@@ -242,6 +242,21 @@ CONFIG_MTD_NAND_DENALI_DT=y
 CONFIG_MTD_NAND_GPMI_NAND=y
 CONFIG_MTD_NAND_FSL_IFC=y
 CONFIG_MTD_SPI_NOR=y
+CONFIG_CFG80211=m
+CONFIG_BCMDHD=n
+CONFIG_BRCMUTIL=m
+CONFIG_BRCMFMAC=m
+CONFIG_BRCMFMAC_SDIO=y
```

```
+CONFIG_BRCMFMAC_PROTO_BCDC=y
+CONFIG_BRCMFMAC_PCIE=y
+CONFIG_BRCMFMAC_PROTO_MSGBUF=y
+CONFIG_CFG80211_REQUIRE_SIGNED_REGDB=n
+CONFIG_CFG80211_USE_KERNEL_REGDB_KEYS=n
+CONFIG_ASYMMETRIC_KEY_TYPE=y
+CONFIG_ASYMMETRIC_PUBLIC_KEY_SUBTYPE=y
+CONFIG_X509_CERTIFICATE_PARSER=y
+CONFIG_PKCS7_MESSAGE_PARSER=y
+CONFIG_MMC_BUS_CLOCK_GATE=y
 # CONFIG_MTD_SPI_NOR_USE_4K_SECTORS is not set
 CONFIG_MTD_UBI=y
 CONFIG_BLK_DEV_LOOP=y
@@ -1106,3 +1121,5 @@ CONFIG_ASYMMETRIC_PUBLIC_KEY_SUBTYPE=y
 CONFIG_X509_CERTIFICATE_PARSER=y
 CONFIG_PKCS7_MESSAGE_PARSER=y
 CONFIG_PWM_IMX_TPM=y
+CONFIG_CFG80211_REQUIRE_SIGNED_REGDB=n
+CONFIG_CFG80211_USE_KERNEL_REGDB_KEYS=n

diff --git
a/drivers/net/wireless/broadcom/brcm80211/brcmfmac/sdio.c
b/drivers/net/wireless/broadcom/brcm80211/brcmfmac/sdio.c
--- a/drivers/net/wireless/broadcom/brcm80211/brcmfmac/sdio.c
+++ b/drivers/net/wireless/broadcom/brcm80211/brcmfmac/sdio.c
@@ -4810,13 +4811,15 @@
-                     netif_rx_ni(skb);
+                     netif_rx(skb);
+                     skb = skbnext;
+              }
diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index b23dcbeacdf3..b0bca07a5c5e 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -7732,7 +7732,7 @@ int sched_setattr_nocheck(struct task_struct *p, const struct
sched_attr *attr)
   return __sched_setscheduler(p, attr, false, true);
 }
 EXPORT_SYMBOL_GPL(sched_setattr_nocheck);
+EXPORT_SYMBOL_GPL(sched_setscheduler);
```

5. Use the following commands to disable cfg80211:

```
#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. Set up the build source-specific to the platform. For example, on i.MX8:

```
$ source /opt/fsl-imx-wayland/6.1-mickledore/environment-setup- armv8a-poky-linux
```

7. Run the following command to avoid linker errors:

```
$ unset LDFLAGS
```

8. Build the kernel for the platform. For example, on i.MX8:

```
$ make clean
$ unset LDFLAGS
$ make ea_imx8_defconfig
$ make
```

The built kernel is available in *arch/arm64/boot/Image.* The output modules can be found in the following files:

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

Note: For any issues during the compilation, see the Community.

3.3.2 Support for Kernels 6.6+

Update the kernel to Infineon-supported version. Download the Fedora releasing kernel and check out to the specific version v6.6.15 (SHA ID: 9670bf06). This uses the following test environment:

- Hardware equipment: INTEL NUC x86_64
- Software tool: Fedora fc38 with v6.6.15 kernel

Do the following:

1. Download the 6.1 + kernel source code:

```
$ git clone https://gitlab.com/cki-project/kernel-ark.git
$ git checkout fedora-6.6
#or
$ git checkout 9670bf06
```

2. Keep the default values for all options based on the contents of your existing *.config* file and ask about new config symbols

```
$ make oldconfig
```

3. Build the kernel image and modules:

```
$ make bzImage
$ make modules
```

4. Install the module and kernel:

```
$ make modules_install
$ make install
```

5. For FMAC compilation, download the FMAC patches from community .In Linux root folder, untar/apply releasing patches with below bash commands
```
$ tar zxvf <release patches package>.tar.gz
#apply script for i in <release patches package>/*.patch; do patch -p1 < $i; done
```
6. Load the platform default configuration:
```
$ make defconfig
```
7. Enable/disable/load the module as shown in the following configuration by using the `make menuconfig` command or by editing the *.config* file:
```
CONFIG_CFG80211=m
CONFIG_BCMDHD=n
CONFIG_BRCMUTIL=m
CONFIG_BRCMFMAC=m
CONFIG_BRCMFMAC_SDIO=y
CONFIG_BRCMFMAC_PROTO_BCDC=y
CONFIG_BRCMFMAC_PCIE=y
CONFIG_BRCMFMAC_PROTO_MSGBUF=y
CONFIG_CFG80211_REQUIRE_SIGNED_REGDB=n
CONFIG_CFG80211_USE_KERNEL_REGDB_KEYS=n
CONFIG_ASYMMETRIC_KEY_TYPE=y
CONFIG_ASYMMETRIC_PUBLIC_KEY_SUBTYPE=y
CONFIG_X509_CERTIFICATE_PARSER=y
CONFIG_PKCS7_MESSAGE_PARSER=y
CONFIG_MMC_BUS_CLOCK_GATE=y
```
8. Enable the following configuration for debug:
```
CONFIG_BRCMDBG=y
CONFIG_BRCM_TRACING=y
```
9. Build kernel modules.

Note: *To speed up compilation on multiprocessor systems, and get some improvement on single-processor ones, use "`-j n`", where `n` is the number of processors * 1.5. You can use the `nproc` command to see how many processors you have.*

```
$ make modules
```
10. Copy the generated modules to new directory for reference:
```
$ cp drivers/mmc/core/mmc_core.ko <Target Folder>
$ cp drivers/mmc/core/mmc_block.ko <Target Folder>
$ cp drivers/mmc/host/sdhci.ko <Target Folder>
$ cp drivers/mmc/host/sdhci-pci.ko <Target Folder>
$ cp drivers/mmc/host/cqhci.ko <Target Folder>
$ cp drivers/net/wireless/broadcom/brcm80211/brcmfmac/brcmfmac.ko <Target Folder>
$ cp drivers/net/wireless/broadcom/brcm80211/brcmfmac/brcmfmac-cyw.ko <Target
Folder>
$ cp drivers/net/wireless/broadcom/brcm80211/brcmutil/brcmutil.ko <Target Folder>
$ cp net/wireless/cfg80211.ko <Target Folder>
```

3.3.3 Backports

Backports is a Linux official project that enables old kernels to run the latest drivers. For example, it enables the Linux 5.15 FMAC driver to run on v4.4 up to v5.15.58.

The Backports project contains a set of scripts, patches, and source code. This project takes the newer-version kernel tree as its input and generates the "backports package" as its output. You can take the backports package and compile the drivers for running on an older kernel.

Figure 5 shows where the backports package provides a modified version of *cfg80211*. There is an extra *compat* module to enable backward compatibility. Note that the original *cfg80211* and *brcmfmac* (in the old kernel) need to be disabled in the *.config* file when building the kernel.

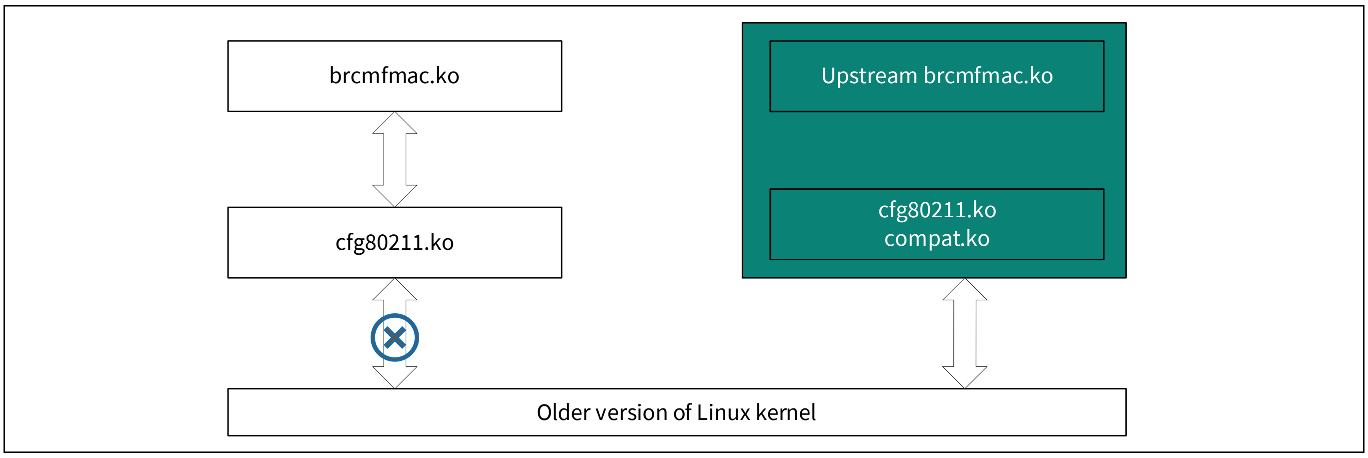


Figure 5 Backports package

Infineon supports the package release mode of the backports package, that is, the target type is loadable modules instead of kernel-integration. Therefore, the kernel source remains untainted, supports multiple versions of the kernel, and eliminates BSP-specific dependencies. To integrate the backports release package with your kernel, do the following.

```
$ git clone https://github.com/Infineon/ifx-backports.git
$ cd ifx-backports/v5.15.58-backports
$ cp brcmfmac defconfigs/.
# Choose the appropriate linux headers
$ MY_KERNEL=/usr/src/linux-headers-*
$ make KLIB=$MY_KERNEL KLIB_BUILD=$MY_KERNEL defconfig-brcmfmac
$ make KLIB=$MY_KERNEL KLIB_BUILD=$MY_KERNEL modules
```

To enable the debug prints, modify the *.config* file:

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

3.3.4 Cross-compilation

To cross-compile FMAC, for example on a Linux host, you need to get the toolchain suitable for your target platform (usually provided in the BSP). For example, get the toolchain from the GNU-A Downloads and extract it in any directory.

Do the following to cross-compile the FMAC driver:

1. Use the following commands to set the appropriate Kernel headers and compilers:
```
export MY_KERNEL=<PATH_TO_KERNEL_OBJ>
export CROSS_COMPILE=<PATH_TO_COMPILER>
```
2. Patch the FMAC driver as mentioned in the Backports section.
3. Configure the FMAC driver using the following command:
```
$ make KLIB=$MY_KERNEL KLIB_BUILD=$MY_KERNEL ARCH=arm64 COMPILE=$CROSS_COMPILE
defconfig-brcmfmac
```
4. Compile the FMAC driver modules using the following command:
```
$ make KLIB=$MY_KERNEL KLIB_BUILD=$MY_KERNEL ARCH=arm64 COMPILE=$CROSS_COMPILE
modules
```

The compiled kernel modules are available in the following directories:

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

3.3.5 Loading the FMAC driver

Do the following to load the FMAC driver to the kernel.

1. Ensure the following:
   - Firmware, clm_blob, and nvram present in the */lib/firmware/cypress* folder.
     - Get the latest firmware and clm_blob from IFX GitHub_firmware.
     - Get the latest NVRAM from your module vendor.
   - All binaries have the prefix *cyfmac<chip_name>-<bus_name>.bin/clm_blob/txt*.
2. Use the following sequence of `insmod` commands:
```
$ insmod compat.ko
$ insmod cfg80211.ko
$ insmod brcmutil.ko
$ insmod brcmfmac.ko
```

Note: *For SDIO interface, add the following module parameters to* `insmod brcmfmac.ko` *to achieve the maximum throughput:*

```
fcmode=2
sdio_in_isr=1
```

```
sdio_rxf_thread=1
```

3.3.6 Notes on debugging

If you ran into kernel crash issues because of problems with the Wi-Fi driver or firmware, enable debug prints in the FMAC driver. To compile the kernel modules with the debug prints enabled, do the following:

1. If you are building the *brcmfmac* kernel modules against the kernel running on the system, use the following commands:
   ```
   CPTCFG_BRCM_TRACING=y
   CPTCFG_BRCMDBG=y
   CPTCFG_BRCMFMAC_PROTO_BCDC=y
   CPTCFG_BRCMFMAC_PROTO_MSGBUF=y
   CPTCFG_CFG80211_WEXT=y
   ```
2. To build a new kernel image, modify the *.config* file of the kernel source with the following:
   ```
   CONFIG_BRCMDBG=y
   CONFIG_DEBUG_FS=y
   ```
3. To compile *brcmfmac* as an LKM against the running kernel, run the following command:
   ```
   make -C <path_to_kernel_src> M=<fmac_source_dir>
   ```
For example:
   ```
   $ make -C /lib/modules/`uname -r`/build M=$PWD
   ```
4. To enable the brcmfmac debug log, use the following command:
   ```
   $ echo 8 > /proc/sys/kernel/printk
   ```
5. Load the driver module with the required message level as the module parameter.
   ```
   $insmod brcmfmac.ko debug=${BRCMF_Message_Level}
   ```
   The following message levels are defined in the *debug.h* file
   (available at */v4.14.52-backports/drivers/net/wireless/broadcom/brcm80211/brcmfmac/debug.h*):
   ```
   #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
#define BRCMF_ULP_VAL 0x00200000
```

For example:

- To enable the Wi-Fi firmware console (ring buffers meant to hold debug prints inside Wi-Fi firmware) log, use the following command:

```
$ insmod brcmfmac.ko debug=0x00100006 (TRACE, INFO and WIFI_FW_LOG)
```

- To set the console polling interval (250 ms), use the following command:

```
$ echo 250 > /sys/kernel/debug/brcmfmac/${mmc slot}/console_interval
```

- To enable trace, use the following command:

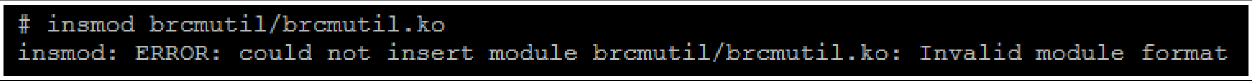
```
$ insmod brcmfmac.ko debug=0x6 (TRACE and INFO )
```

For further details on the functions associated with debugging the FMAC, see the source code available in *drivers/net/wireless/broadcom/brcm80211/brcmfmac/debug.c*.

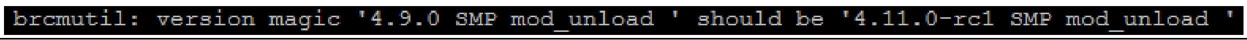
3.3.7 Frequently encountered issues

1. Invalid module format.

   If you get the following errors, see the dmesg for the detailed error.



```
# insmod brcmutil/brcmutil.ko
insmod: ERROR: could not insert module brcmutil/brcmutil.ko: Invalid module format
```



```
brcmutil: version magic '4.9.0 SMP mod_unload ' should be '4.11.0-rc1 SMP mod_unload '
```

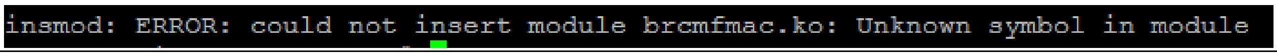
   Root cause:

   There could be a mismatch between the LKMs built for a particular kernel version and those in the current system. Additionally, the architecture might differ between the compiled kernel module and the host platform.

   Solution:

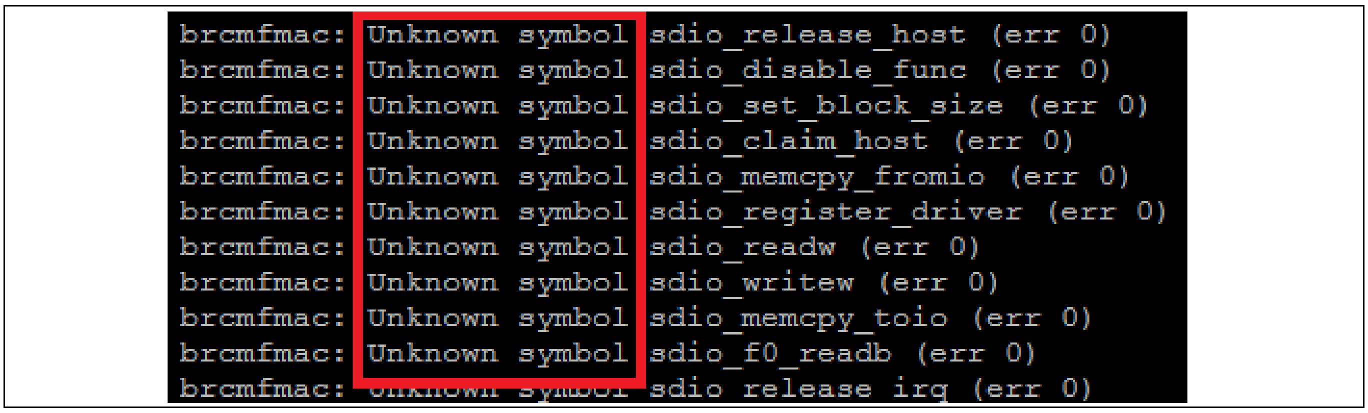
   Download the correct kernel and reinstall the correct kernel image.

2. Unknown symbol in module.



```
insmod: ERROR: could not insert module brcmfmac.ko: Unknown symbol in module
```

   Use dmesg to check the root cause of this error.



```
brcmfmac: Unknown symbol sdio_release_host (err 0)
brcmfmac: Unknown symbol sdio_disable_func (err 0)
brcmfmac: Unknown symbol sdio_set_block_size (err 0)
brcmfmac: Unknown symbol sdio_claim_host (err 0)
brcmfmac: Unknown symbol sdio_memcpy_fromio (err 0)
brcmfmac: Unknown symbol sdio_register_driver (err 0)
brcmfmac: Unknown symbol sdio_readw (err 0)
brcmfmac: Unknown symbol sdio_writew (err 0)
brcmfmac: Unknown symbol sdio_memcpy_toio (err 0)
brcmfmac: Unknown symbol sdio_f0_readb (err 0)
brcmfmac: Unknown symbol sdio_release_irq (err 0)
```

Root cause:

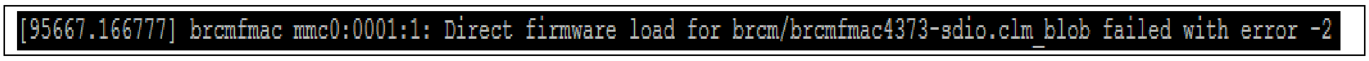
Some modules were missed before loading the brcmfmac driver.

Solution:

- Check all dependencies of the module before loading it.
- Grep the unknown symbols, as printed by `dmesg` in the Linux kernel to identify the missing modules.

3. No channels in `iw reg get`, the country code is `#n`.

   Check the detailed error in `dmesg`.



```
[95667.166777] brcmfmac mmc0:0001:1: Direct firmware load for brcm/brcmfmac4373-sdio.clm_blob failed with error -2
```

Root cause:

The clm_blob might be for CYW4373 in the */lib/firmware/cypress* folder.

The message “cannot find clm version” in dmesg indicates that the Infineon-specific patches were not applied for brcmfmac clm_blob or might not have the download facility.

Solution:

- Copy the *clm_blob* file from the quarterly release train to */lib/firmware/cypress*.
- See AN225347 to understand the clm_blob flow. Request a product-specific clm_blob from Infineon. Replace the generic clm_blob copied from the quarterly release train with the product-specific clm_blob.
- Move to Infineon-specific brcmfmac instead of Linux-native FMAC.

4. The USB dongle is not brought up after `insmod brcmfmac`.

Root cause:

NVRAM parameters might be included in the firmware image. Run the strings command in */lib/firmware/cypress/cyfmac4373-usb.bin* to check whether NVRAM parameters are included in the tail of the firmware image.

Solution:

If NVRAM parameters are not present, contact the Infineon support to build a firmware with NVRAM and replace the existing one. This error does not occur if you use the firmware from the quarterly release train.

4 User-space Wi-Fi utils

4.1 *wpa_supplicant*

wpa_supplicant is a cross-platform supplicant providing support for WEP, WPA, WPA2, WPA3 (IEEE 802.11i), and WPA-EAP. It implements the key negotiation with an authenticator and also controls the roaming and association of STA devices. It has the following key features:

- WPA and full IEEE 802.11i, RSN, WPA2
- WPA-PSK and WPA2-PSK
- WPA-EAP (WPA – Enterprise, for example, with RADIUS server)
- Key management for CCMP, TKIP, and WEP (both 104/128- and 40/64-bit)
- RSN: PMKSA caching, pre-authentication
- IEEE 802.11r
- IEEE 802.11w
- Wi-Fi Protected Setup (WPS)

4.1.1 Dependencies of *wpa_supplicant*

- Netlink Protocol Library Suite (libnl)

  The libnl suite is a collection of libraries providing APIs to netlink-based Linux kernel interfaces. Netlink is a socket-based IPC mechanism primarily between the kernel and user-space processes. It was designed to be a more flexible successor to IOCTL to provide mainly networking-related kernel configuration and monitoring interfaces. The IOCTL risks polluting the kernel and damaging the stability of the system. Netlink socket is simple, only a constant (protocol type) needs to be added to *netlink.h*. Then, the kernel module and application can communicate using socket-style APIs immediately.

- OpenSSL

  OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) network protocols and related cryptography standards required by them. The OpenSSL program is a command-line tool for using various cryptography functions of OpenSSL's crypto library from the shell.

- dbus (optional)

  D-Bus is a message bus system, which provides a simple way to communicate with other processes. Modern *wpa_supplicant* versions have two control interfaces: a dbus API and a directory, normally */var/run/wpa_supplicant/* or */run/wpa_supplicant/* depending on the distro, containing a socket named for each Wi-Fi interface that *wpa_supplicant* manages. The control interfaces are not active by default. You need the `-u` command-line option to get dbus, and `-O /var/run/wpa_supplicant` (or whichever directory) for the sockets.

4.1.2 Compiling *wpa_supplicant*

1. Get the latest source of *wpa_supplicant* from ifx-hostap GitHub repo.
```
$ git clone https://github.com/Infineon/ifx-hostap.git
```

2. Navigate to *wpa_supplicant-2.10/wpa_supplicant* and modify the *defconfig* according to your system requirements.
   For an Android-based host, you might want to re-route the debug prints to *logcat* by uncommenting `CONFIG_ANDROID_LOG=y`. There are several other debug-specific macros that you can uncomment based on your requirements.
```
$ cd ifx-hostap/wpa_supplicant
$ cp defconfig_base .config
```

3. Compile the *wpa_supplicant* source files using the following command:
```
$ make -j4
```

Note: *If you are cross-compiling, use the following command:*

```
$ export CROSS_COMPILE=<PATH_TO_COMPILER>
$ make -j4 <cc=$CROSS_COMPILE>
```

4. To install the compiled binaries at the */usr/sbin* directory (a popular choice), use the following command:
```
$ make install DESTDIR=<your_target_directory>
```

Note: *You can also manually copy the generated binaries from the following:*

- *hostap/wpa_supplicant/wpa_supplicant*
- *hostap/wpa_supplicant/wpa_cli*

Note: *You might come across issues such as a missing header file in openssl or a version mismatch. Make sure that you have followed the dependency section and installed them according to the exact version requirement. For example, in an Ubuntu-based system, for libssl.so or libcrypto.so, most times, `sudo apt-get install libssl-dev` should be sufficient. However, the version requirement might be 1.0.2 or 1.1 instead of the default installed version (1.0.0 in some cases). In that case, you would need to upgrade your system's libssl version to the one required by the particular release of wpa_supplicant. Though these errors are uncommon, some host processors running an older version of the kernel can introduce such errors.*

4.1.3 Configuring *wpa_supplicant*

wpa_supplicant is configured using a text file that lists all accepted networks and security policies, including pre-shared keys. All file paths in this configuration file should use a full path (absolute, not relative to the working directory) to allow the working directory to be changed.

Example:

```
# allow frontend (e.g., wpa_cli) to be used by all users in the 'wheel' group
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=wheel
#
# home network; allow all valid ciphers
network= {
        ssid="home"
        scan_ssid=1
        key_mgmt=WPA-PSK
        psk="very secret passphrase"
}
#
# work network; use EAP-TLS with WPA; allow only CCMP and TKIP ciphers
network= {
        ssid="work"
        scan_ssid=1
        key_mgmt=WPA-EAP
        pairwise=CCMP TKIP
        group=CCMP TKIP
        eap=TLS
        identity="user [at] example.com"
        ca_cert="/etc/cert/ca.pem"
        client_cert="/etc/cert/user.pem"
        private_key="/etc/cert/user.prv"
        private_key_passwd="password"
}
```

For more details on other specific macros relevant in *wpa_supplicant.conf*, see the wpa_supplicant configuration file.

After completing the configuration, start *wpa_supplicant* for connecting to your office or home access point and get on with the application. Run the following command:

```
$ wpa_supplicant -B -i wlan0 -c /etc/wpa_supplicant/example.conf
```

where,

- `-B`: Runs the *wpa_supplicant* daemon in the background.
- `-c`: Provides the configuration file; in this case, *example.conf*.
- `-i`: Selects the network interface to be used.

For additional debug prints, you can add the `-d` parameter while running the command. For more details, see the wpa_supplicant(8) Linux main page.

Note: *If you get the following error: "Failed to initialize control interface '/var/run/wpa_supplicant'", you may have another wpa_supplicant process already running or the file was left by an unclean termination of wpa_supplicant. Use the* `$killall wpa_supplicant` *command to manually remove this file before restarting wpa_supplicant.*

4.1.4 wpa_cli

The wpa_cli utility is a text-based front-end program for interacting with *wpa_supplicant*. It is used to query the current status, change the configuration, trigger events, and request interactive user input. Additionally, the utility can configure the Extensible Authentication Protocol over LAN (EAPoL) state machine parameters and trigger events such as reassociation and IEEE 802.1x logoff and logon.

The wpa_cli utility supports two modes: interactive and command-line. Both modes share the same command set; the main difference is that the interactive mode provides access to unsolicited messages (event messages and username/password requests).

Interactive mode is started when wpa_cli is executed without any parameters on the command line. Commands are then entered from the controlling terminal in response to the wpa_cli prompt. In command-line mode, the same commands are entered as command-line arguments.

4.1.4.1 Configuration options

The options listed in Table 6 are available as an argument to configure wpa_cli.

Table 6 Options to start wpa_cli

Command	**Description**
`-p path`	Controls the sockets path. This should match the *ctrl_interface* in *wpa_supplicant.conf*. The default path is */var/run/wpa_supplicant.*
`-i ifname`	Configures the interface. By default, the first interface found in the socket path is used.
`-h`	Shows help
`-v`	Shows version information
`-B`	Runs the daemon in the background
`-a action_file`	Runs in daemon mode, executing the action file based on events from wpa_supplicant
`-P pid_file`	Provides the PID file location
`-g global_ctrl`	Uses a global control interface to *wpa_supplicant* rather than the default Unix domain sockets
`-G ping_interval`	Waits for the ping interval (in seconds) before sending each ping to *wpa_supplicant*. See the `ping` command
`command`	Lists the available commands

4.1.4.2 wpa_cli commands

The commands are issued as listed in Table 7 on the command line or at a prompt when operating interactively.

Table 7 wpa_cli commands

Command	**Description**
`add_network`	Adds a network
`blacklist [bssid | clear]`	Adds a BSSID to the blacklist. When invoked without any extra arguments, display the blacklist. Specifying `clear` causes wpa_cli to clear the blacklist.
`bss [idx | bssid]`	Gets a detailed BSS scan result for the network identified by `bssid` or `idx`
`bssid network_id bssid`	Sets a preferred BSSID for an SSID
`Disconnect`	Disconnects and waits for reassociate or reconnect command before connecting
`enable_network network_id`	Enables a network
`get_network network_id variable`	Gets network variables
`Help`	Displays usage information
`identity network_id identity`	Configures the identity for an SSID
`ifname`	Shows the current interface name. The default interface is the first interface found in the socket path.
`interface [ifname]`	Shows the available interfaces, sets the current interface, or does both when multiple interfaces are available.
`interface_add ifname [confname driver ctrl_interface driver_param bridge_name]`	Adds a new interface with the given parameters
`interface_list`	Lists the available interfaces
`interface_remove ifname`	Removes the interface
`level debug_level`	Changes the debugging level in *wpa_supplicant*. Larger numbers generate more messages.
`license`	Displays the full license for wpa_cli
`list_networks`	Lists the configured networks
`logoff`	Sends the IEEE 802.1X EAPoL state machine into the "logoff" state
`logon`	Sends the IEEE 802.1X EAPoL state machine into the "logon" state.
`mib`	Reports MIB variables (dot1x and dot11) for the current interface
`new_password network_id password`	Changes the password for an SSID
`otp network_id password`	Configures a one-time password for an SSID
`passphrase network_id passphrase`	Configures a private key passphrase for an SSID
`password network_id password`	Configures a password for an SSID

Command	Description
`PIN network_id pin`	Configures a PIN for an SSID
`ping`	Pings the *wpa_supplicant* utility. This command can be used to test the status of the *wpa_supplicant* daemon.
`pmksa`	Shows the contents of the PMKSA cache
`preauthenticate BSSID`	Forces preauthentication of the specified `BSSID`
`quit`	Exits wpa_cli
`reassociate`	Forces a reassociation to the current access point
`reconfigure`	Forces *wpa_supplicant* to reread its configuration file
`reconnect`	Similar to reassociate, but only takes effect if already disconnected
`remove_network network_id`	Removes a network
`scan`	Requests a new BSS scan
`scan_results`	Gets the latest BSS scan results. This command can be invoked after running a BSS scan with scan
`select_network network_id`	Selects a network and disable others
`set [settings]`	Sets variables. When no arguments are supplied, the known variables and their settings are displayed.
`set_network [network_id variable value]`	Sets network variables. Shows a list of variables when run without arguments.
`status`	Gets the current WPA/EAPoL/EAP status for the current interface
`terminate`	Terminates *wpa_supplicant*

Note: *If you encounter the error "Rfkill Soft blocked" while running any of the wpa_cli or wpa_supplicant commands, use the `$sudo rfkill unblock all` command.*

4.1.4.3 Typical STA/AP use cases

STA/AP combinations	wpa_cli commands

STA connecting to an AP with open security	`wpa_cli>IFNAME=wlan0 remove_n all`
	`wpa_cli>IFNAME=wlan0 add_network`
	`IFNAME=wlan0 wpa_cli>set_network 0 ssid "wireless_test_2"`
	`wpa_cli>IFNAME=wlan0 set_network 0 key_mgmt NONE`
	`wpa_cli>IFNAME=wlan0 enable_network 0`
	`wpa_cli>IFNAME=wlan0 select_network 0`
	`wpa_cli>IFNAME=wlan0 status`

STA connecting to an AP with WPA2 security	`wpa_cli> IFNAME=wlan0 remove_n all`
	`wpa_cli> IFNAME=wlan0 add_network`
	`wpa_cli> IFNAME=wlan0 set_network 0 ssid "wireless_test_2"`
	`wpa_cli> IFNAME=wlan0 set_network 0 proto WPA2`
	`wpa_cli> IFNAME=wlan0 set_network 0 key_mgmt WPA-PSK`
	`wpa_cli> IFNAME=wlan0 set_network 0 pairwise CCMP`
	`wpa_cli> IFNAME=wlan0 set_network 0 psk "12345678"`

STA/AP combinations	wpa_cli commands
	wpa_cli> IFNAME=wlan0 enable_network 0 wpa_cli> IFNAME=wlan0 select_network 0 wpa_cli> IFNAME=wlan0 status
STA connecting to an AP with WPA3 security	wpa_cli> IFNAME=wlan0 disconnect wpa_cli> IFNAME=wlan0 list_network wpa_cli> IFNAME=wlan0 remove_network 0 wpa_cli> IFNAME=wlan0 add_network wpa_cli> IFNAME=wlan0 set_network 0 ssid '"localap3"' wpa_cli> IFNAME=wlan0 set_network 0 ieee80211w 2 wpa_cli> IFNAME=wlan0 set_network 0 proto RSN wpa_cli> IFNAME=wlan0 set_network 0 key_mgmt SAE wpa_cli> IFNAME=wlan0 set_network 0 pairwise CCMP wpa_cli> IFNAME=wlan0 set_network 0 sae_password '"12345678"' wpa_cli> IFNAME=wlan0 save_config wpa_cli> IFNAME=wlan0 enable_network 0 wpa_cli> IFNAME=wlan0 select_network 0 wpa_cli> IFNAME=wlan0 status
STA connecting to an AP with WPA3_WPA2 security	wpa_cli> IFNAME=wlan0 disconnect wpa_cli> IFNAME=wlan0 list_network wpa_cli> IFNAME=wlan0 remove_network 0 wpa_cli> IFNAME=wlan0 add_network wpa_cli> IFNAME=wlan0 set_network 0 ssid '"localap3"' wpa_cli> IFNAME=wlan0 set_network 0 ieee80211w 1 wpa_cli> IFNAME=wlan0 set_network 0 proto RSN wpa_cli> IFNAME=wlan0 set_network 0 key_mgmt SAE wpa_cli> IFNAME=wlan0 set_network 0 pairwise CCMP wpa_cli> IFNAME=wlan0 set_network 0 sae_password '"12345678"' wpa_cli> IFNAME=wlan0 set_network 0 psk ‘”12345678”’ wpa_cli> IFNAME=wlan0 save_config wpa_cli> IFNAME=wlan0 enable_network 0 wpa_cli> IFNAME=wlan0 select_network 0 wpa_cli> IFNAME=wlan0 status
2G/5G SoftAP with open security	wpa_cli> IFNAME=wlan0 remove_net all wpa_cli> IFNAME=wlan0 add_net wpa_cli> IFNAME=wlan0 set_net 0 ssid ‘”CYP_5GAP”’ wpa_cli> IFNAME=wlan0 set_net 0 key_mgmt NONE wpa_cli> IFNAME=wlan0 set_net 0 frequency 5180 (Change 5180 to 2437 for setting up 2.4 GHz AP) wpa_cli> IFNAME=wlan0 set_net 0 mode 2 wpa_cli> IFNAME=wlan0 select_net 0
2G/5G SoftAP with WPA2 security	wpa_cli> IFNAME=wlan0 remove_network all wpa_cli> IFNAME=wlan0 add_network wpa_cli> IFNAME=wlan0 set_network 0 ssid ‘"CYP_wpa2psk_5GAP"’ wpa_cli> IFNAME=wlan0 set_network 0 proto WPA2 wpa_cli> IFNAME=wlan0 set_network 0 key_mgmt WPA-PSK wpa_cli> IFNAME=wlan0 set_network 0 pairwise CCMP

STA/AP combinations	wpa_cli commands

	`wpa_cli> IFNAME=wlan0 set_network 0 psk '"9876543210"'`
	 `wpa_cli> IFNAME=wlan0 set_net 0 frequency 5745`
	 `(Change 5180 to 2437 for setting up 2.4 GHz AP)`
	 `wpa_cli> IFNAME=wlan0 set_net 0 mode 2 wpa_cli> IFNAME=wlan0 select_network 0 wpa_cli> IFNAME=wlan0 status`

4.1.4.4 Wireless authentication and privacy infrastructure (WAPI)

WAPI is a Chinese national standard for the security of WLANs. It is more secure than WEP or WPA.

For WAPI-related support, contact the local Infineon sales office or an Infineon representative who can provide you with the WAPI-enabled version of *wpa_supplicant*.

Note: *The Linux driver should support nl80211/cfg80211. If the device is old and does not support the netlink driver, you would need to roll back to the legacy wext driver:*

```
# wpa_supplicant -B -i wlan0 -D wext -c /etc/wpa_supplicant/example.conf
```

4.2 hostapd

hostapd is a user-space daemon for setting up the access point or authentication servers with fine granular control over most of the parameters. It implements the IEEE 802.11 access point management, IEEE 802.1x/WPA/WPA2/WPA3/EAP authenticators, RADIUS client, EAP server, and RADIUS authentication server. You can configure hostapd to function in any of those modes. Originally designed to be a daemon program, hostapd supports front-end programs, such as *hostapd_cli*.

4.2.1 hostapd dependencies

- libnl
- openssl

4.2.2 Compiling hostapd

1. Get the latest source of *wpa_supplicant* from the ifx-hostap GitHub.

```
$ git clone https://github.com/Infineon/ifx-hostap.git
```

2. Navigate to *wpa_supplicant-2.10/hostapd* and modify the defconfig according to your system requirements:

```
$ cd ifx-hostap/hostapd
$ cp defconfig_base .config
```

3. To compile the hostapd source files, use the following command:

```
$ make -j4
```

Note: *If you are cross-compiling, use the following command:*

```
$ export CROSS_COMPILE=<PATH_TO_COMPILER>
$ make -j4 <cc=$CROSS_COMPILE>
```

4. To install the compiled binaries at the */usr/sbin* directory (a popular choice), use the following command:

```
$ make install DESTDIR=<your_target_directory>
```

Note: You can also manually copy the generated binaries from the following:

- *hostap/hostapd/hostapd*
- *hostap/hostapd/hostapd_cli*

4.2.3 Conf files

hostapd is configured using a text file that sets up the AP's security policies (802.11i, 802.1x, and so on), country code, passphrase, and so on.

Create a *hostapd.conf* file such as the following example:

```
interface=wlan0
driver=nl80211
ctrl_interface=/tmp/hostapd
ssid=test_ssid
hw_mode=g
channel=1
macaddr_acl=0
auth_algs=1
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_passphrase=test_ssid
rsn_pairwise=CCMP
wpa_pairwise=CCMP
To set the device up as a softap, n run the following command:
$ sudo hostapd ./hostapd.conf -B -dd
```

Note: `-dd` *is used to enable debug prints. It can be removed after the platform bringup is over.*

4.2.3.1 hostapd usage

hostapd usage	Settings

Setting up a SoftAP with WPA2 security	`interface=wlan0`
	`driver=nl80211`
	`ctrl_interface=/tmp/hostapd`
	`ssid=test_ssid`
	`hw_mode=g`
	`channel=1`
	`macaddr_acl=0`
	`auth_algs=1`
	`wpa=2`
	`wpa_key_mgmt=WPA-PSK`
	`wpa_passphrase=test_ssid`
	`rsn_pairwise=CCMP`
	`wpa_pairwise=CCMP`

hostapd usage	Settings

Setting up a SoftAP with WPA3 security	`interface=wlan0`
	`driver=nl80211`
	`ctrl_interface=/tmp/hostapd`
	`ssid=hostap_sae`
	`channel=6`
	`hw_mode=g`
	`auth_algs=3`
	`wpa=2`
	`wpa_key_mgmt=SAE`
	`sae_password=12345678`
	`ieee80211w=2`
	`rsn_pairwise=CCMP`
	`group_cipher=CCMP`

4.2.4 DHCP configuration

There are a few options available for the DHCP daemon, such as *udhcp*, *dhcp*, and *dnsmasq*. Most of them are used as a DHCP server; some of them additionally provide the DNS server functionality. Sometimes, they are included by default with a core OS, such as *udhcp*; otherwise, you can manually install the packages (for example, *dnsmasq*). In this case, *dnsmasq* is considered an example to demonstrate how to setup DHCP and DNS servers on the AP interface.

Do the following changes to the *dhcpd.conf* file:

```
$ sudo nano /etc/dhcpd.conf
interface wlan0
static ip_address=192.168.0.10/24
```

The default *dnsmasq* configuration file provides the options to configure the DHCP server. Therefore, instead of editing the default *.conf* file, back up the file, create a new .config file, and use the file:

```
$ sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig
$ sudo nano /etc/dnsmasq.conf
  interface=wlan0
  dhcp-range=192.168.0.11,192.168.0.30,255.255.255.0,24h
  dhcp-option=3,192.168.1.1  #Gateway IP
  dhcp-option=6,192.168.1.1         #DNS
  server=8.8.8.8                    #DNS Server
  log-queries
  log-dhcp
  listen-address=127.0.0.1
```

These lines allocate IP addresses between 192.168.0.11 and 192.168.0.30 to the *wlan0* interface. Now, with certain amendments in network routing you can start *dnsmasq*:

```
$ ifconfig wlan0 up 192.168.1.1 netmask 255.255.255.0
$ route add -net 192.168.1.0 netmask 255.255.255.0 gw 192.168.1.1
$ dnsmasq -C dnsmasq.conf -d
```

Even if you choose to use some of the available DHCP server daemon tools, see the corresponding documentation (for path-specific changes and so on) to setup the DHCP and DNS servers.

4.3 *iw*

The *iw* utility is an *nl80211*-based user-space command-line utility used to configure the wireless devices. It supports both Wi-Fi drivers used by Wi-Fi devices. The old *iwconfig* tool is deprecated and it is strongly recommended to switch to *iw* and *nl80211*.

4.3.1 Dependencies

The basic requirement for *iw* is to have *libnl*. The following dependencies should be met for *pkg-config*:

- *libnl >= libnl-1*
- *libnl-devel >=libnl-devel-1*
- *libnl-genl >= libnl-genl-1*
- *crda*
- *wireless-regdb*

4.3.2 Compiling *iw*

You can use the package manager tool for your system to install these packages and then proceed with the installation of *iw*. If you choose to compile from source, release tar balls are available in the kernel page.

4.3.3 Typical usage

The `iw list` command provides you with the capabilities of the wireless devices in your system. This command also displays the list of supported commands for your Wi-Fi device. Based on that, you can use `iw help <cmd_name>` to issue the command for your use case.

4.3.3.1 SoftAP with WPA/WPA2/WPA3 security

Use the following commands to set up a secured SoftAP:

```
$ iw dev wlan0 interface add softap type __ap
$ ifconfig wlan0 hw ether 00:90:4c:12:d0:05
$ ifconfig wlan0 192.168.10.1
```

For the AP-related security configuration, you can use *hostapd.conf*, with the `interface` parameter as `softap` as mentioned in the *Conf* files, and set up the AP with the desired level of security (WPAx).

4.3.3.2 STA connecting to an AP with open/WEP security

iw can handle only the connection process with either open security or WEP security. For WPAx security, use *wpa_supplicant* instead.

Open security:

```
iw wlan0 connect <target_ap_ssid>
```

If there are multiple APs with the same SSID, and you want to connect with the AP that is on frequency 2432 (Channel 5), run the following command:

```
iw wlan0 connect <target_ap_ssid> 2432
```

WEP is deprecated in favor of the more robust security measures available in 802.11i. If you still have a WEP-supported AP and want to connect using *iw*, use the following command:

```
iw wlan0 connect <target_wep_ap_ssid> keys 0:abcde d:1:0011223344
```

5 Appendix

5.1 Checklist to add connectivity to a default/Yocto release

Yocto is a widely used custom embedded Linux distribution creation tool. The Yocto project provides a flexible set of tools and a space where embedded developers worldwide can share technologies, software stacks, configurations, and best practices to create tailored Linux images for embedded and IoT devices, or anywhere a customized Linux OS is needed. This release allows you to enable the Wi-Fi connectivity software in your Yocto projects, making it easier to get started quickly with the connectivity software. For the Yocto release, follow this build procedure:

1. Extract the build scripts tar ball:

```
$ tar zxvf cypress-yocto-scripts-v5.15.58-*.tar.gz
```

2. Create a working directory. For example, *cypress-imx-bsp*:

```
$ mkdir cypress-imx-bsp
```

3. Copy the following data into the working directory:

```
* cypress-fmac-v5.15.58-*.zip
* build_yocto_wireless.sh
* meta-cywlan
* nvram.zip
* bt-firmware.tar.gz
$ cp cypress-fmac-v5.15.58-*.zip cypress-imx-bsp
$ cp -r cypress-yocto-scripts-v5.15.58-2023_0901/meta-cywlan cypress-yocto-scripts-
v5.15.58-2023_0901/build_yocto_wireless.sh cypress-yocto-scripts- v5.15.58-
2023_0901/nvram.zip cypress-yocto-scripts- v5.15.58-2023_0901/bt-firmware.tar.gz
cypress-imx-bsp
```

4. Run the *setup_host_env.sh* script for the first-time build. This will help to set up the build environment for your host.

```
$ cypress-yocto-scripts- v5.15.58-2023_0901/setup_host_env.sh
```

5. Run the *build_yocto_wireless.sh* script in the working directory to generate the Infineon customized Yocto image.

```
$ cd cypress-imx-bsp
$ ./build_yocto_wireless.sh
```

6. If the scripts are unable to be run by user permission, use the following command:

```
$ chmod a+x *.sh
```

5.2 Checklist to add connectivity to a non-Yocto release

The patch files in this quarterly release package are based on the latest stable Linux kernel release (v5.15.58); therefore, older kernels need to use the backports package. Here are some examples of how to use this package with an older kernel or Linux-stable v5.4.18.

If you are using the backports project with an older version of the kernel, you need to build the Linux kernel image and Infineon Wi-Fi driver modules separately.

Building the kernel image is done by following the steps mentioned in the Kernel configuration. For Infineon Wi-Fi driver backports modules, you can follow the steps mentioned in the Support for kernels 6.1+ section.

5.3 Upgrading the firmware

Usually, the quarterly releases contain the updated firmware. To upgrade to the latest firmware, do the following:

1. Get the files from the community page.
2. Replace the files in your host filesystem(*/lib/firmware/cypress/**).
3. Run the following command:
   ```
   $ modprobe brcmfmac.ko
   ```

5.4 Toolchain for i.MX8

5.4.1 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.

5.4.2 Required packages

1. Install the following packages on the host machine for the Yocto project:
   ```
   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
   ```

2. Use the following command to create a directory named *bin* in the home folder.
   ```
   mkdir ~/bin
   ```

3. Download the tool using the following command:
   ```
   curl http://commondatastorage.googleapis.com/git-repodownloads/repo > ~/bin/repo
   ```

4. Use the following command to execute the tool:

```
chmod a+x ~/bin/repo
```

5. 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
```

5.4.3 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.

1. Create a directory for the downloaded files (*ea-bsp* in the following example):

```
mkdir ea-bsp
cd ea-bsp
```

2. Configure Git if not configured already. Change “Your name” to your actual name and "Your e-mail" to your email address.

```
git config --global user.name "Your name"
git config --global user.email "Your e-mail"
```

3. Initialize the repo. The file containing all the required repositories is downloaded in this step. Change **<selected branch>** to a branch name accordingly.

   This example uses the *ea-6.1.36* branch.

```
repo init -u https://github.com/embeddedartists/ea-yocto-base -b ea-6.1.36
```

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 *sources* subdirectory.

5.4.4 Initializing 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-wayland MACHINE=imx8mnea-ucom source ea-setup-release.sh -b build-
imx8mnea-ucom-wayland
```

2. Edit *conf/local.conf* to enable the SSH server:

```
- EXTRA_IMAGE_FEATURES = "debug-tweaks"
+ EXTRA_IMAGE_FEATURES = "debug-tweaks ssh-server-openssh"
```

5.4.5 Custom i.MX toolchain

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

```
bitbake meta-toolchain
```

   This creates the *build-imx8mnea-ucom-xwayland/tmp/deploy/sdk* file.

2. Go to the *build-imx8mnea-ucom-xwayland/tmp/deploy/sdk* location to install the toolchain.
3. Use the following command to install the toolchain:

```
sudo ./fsl-imx-wayland-glibc-x86_64-meta-toolchain-armv8a-imx8mnea-ucom-toolchain-
6.1-mickledore.sh
```

   This results in the toolchain being installed in */opt/fsl-imx-wayland/6.1-mickledore/*.

4. Source the environment variables using the following command:

```
source /opt/fsl-imx-wayland/6.1-mickledore/environment-setup-armv8a-poky-linux
```

5.4.6 Building the image

Now that the setup is completed, 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 several hours depending on the host computer's capability.

References

[1] Device Tree Structure

[2] Linux Wireless

[3] Linux Device Drivers

[4] Linux MMC Subsystem

[5] Linux PCI Bus Subsystem

[6] Linux USB Subsystem

[7] Linux Backports project

Revision history

Document revision	Date	Description of changes
**	2021-04-16	Initial release.
*A	2024-02-15	Updated Cypress links to Infineon. Removed the NRND chip. Changed the Android-specific steps as generic. Updated Table 2 and Table 3. Updated Multimedia Card and Support for kernels 6.1+. Added Toolchain for i.MX8.
*B	2024-05-21	Updated Support for Kernels 6.6+ Added CYW43022 support
*C	2024-11-19	Added CYW55500 support in Table 2.

Edition 2024-11-19

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-32689 Rev. *C