Wi-Fi software user guide

About this document

Scope and purpose

This document provides an overview of the building blocks of 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 primarily 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

Table of contents

1 Introduction to Wi-Fi software

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 subsytems and the components involved in configuring the WLAN from user-space, for example, wpa_supplicant, hostapd, iw, and so on. This document will cover the user-space, kernel space features and device drivers written or configured to be used by Wi-Fi devices.

1.1 Block diagram of Wi-Fi software architecture

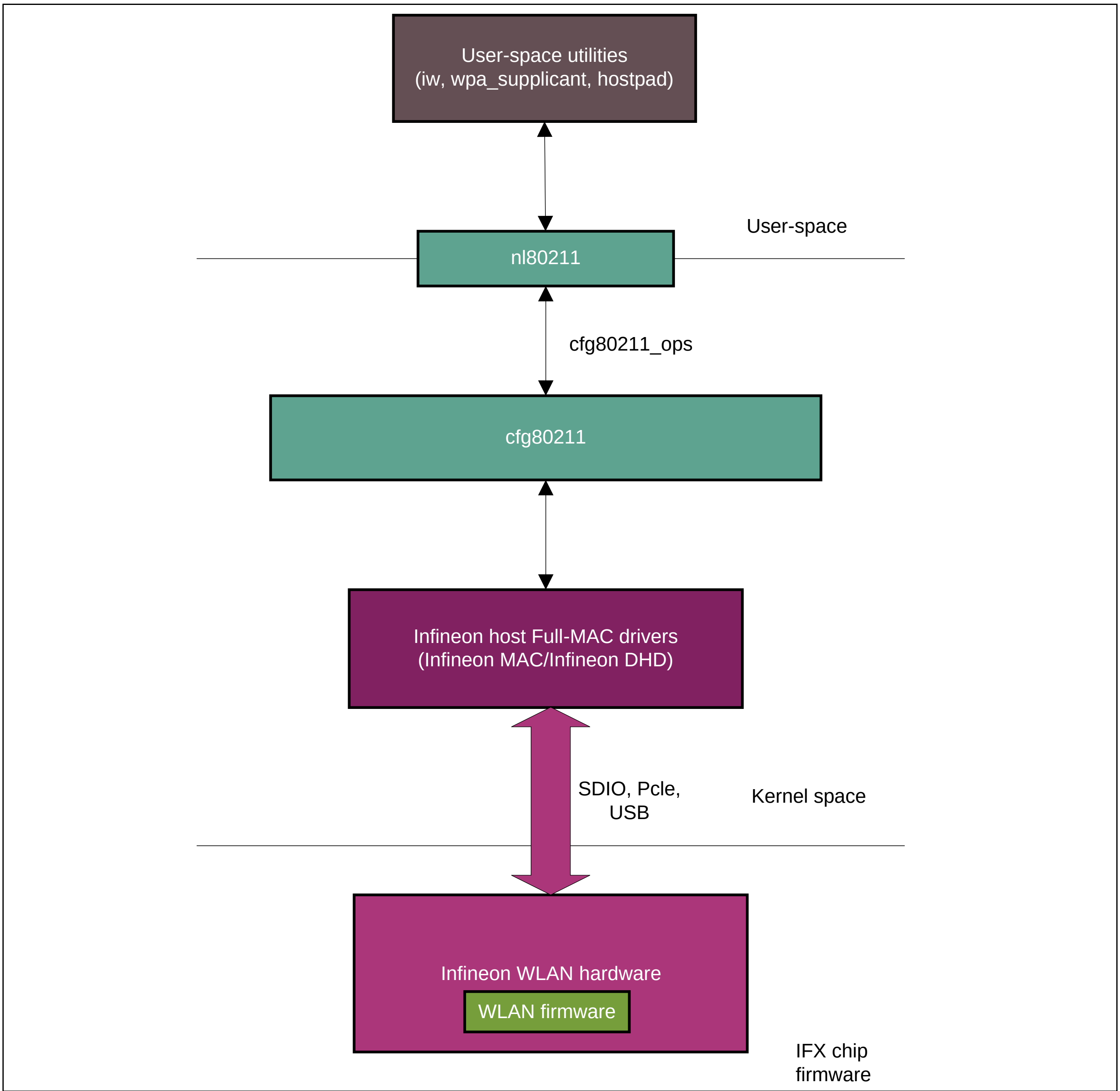

Figure 1 Linux 802.11 architecture - abridged

There is a transmit (TX), receive (RX), and event paths between the applications (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 quarterly release trains**). The lowermost layer implements the core 802.11 operations along with a part of the bus hardware. This layer is implemented inside the IFX Wi-Fi firmware, and packaged with eliminating the need for writing separate device driver, hence reducing 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: In this category, the host processor and the target Wi-Fi radio are connected through bi-directional bus.
- For SDIO, the connections are D0, D1, D2, and D3. Similarly, for PCIe the corresponding connections will be TDN, TDP, RDN, and RDP, and for USB it is DP, DN, and so on. Usually, for Bluetooth the transport will be through UART and the 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/BT connection	USB	DP, DN
Host <---> BT connection	UART	RTS, CTS, TX, RX

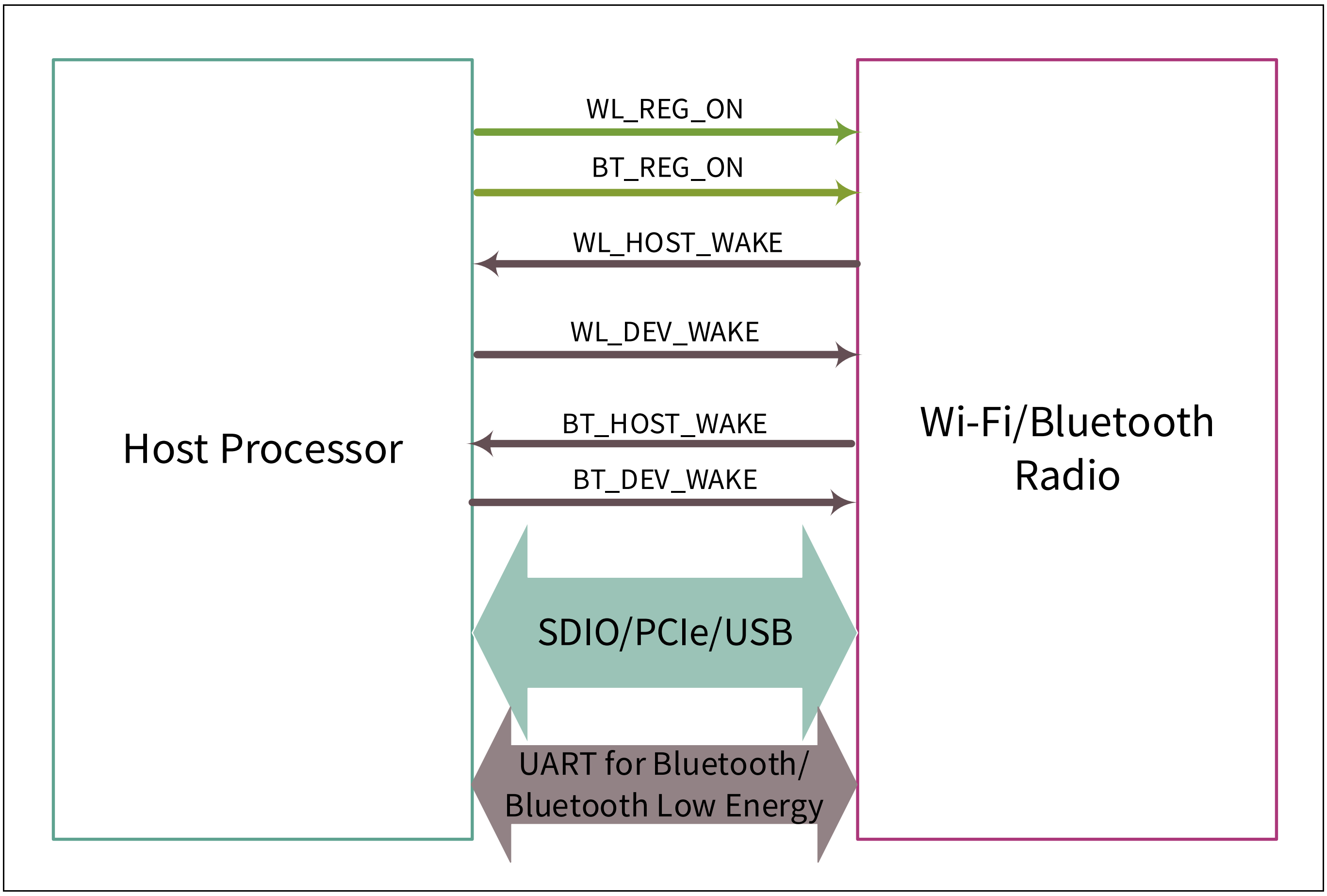

Figure 2 Host Processor to Wi-Fi/Bluetooth/Bluetooth Low Energy radio connection blocks

- Power-related connection: This category includes the GPIOs required to power up the Wi-Fi module. WL_REG_ON, BT_REG_ON belongs to this category.

- Signalling connection: This category 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 Device tree blob

Device Tree provides a way to describe platform_data or pdata of hardware that is not inherently discoverable for instance, I2C and SPI devices. Device Tree Blob 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 definition. The device tree source files are compiled using Device Tree Compiler (DTC); source files can be found in the *<kernel_base>/scripts/dtc* folder. DTC generates the *.dtb* , which is also known as Flattened Device Tree (FDT). The Linux operating system 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. Usually, the device tree contains information regarding the I/O port and interrupt lines that the device is supposed to use. Each device node (representing a platform device in a tree of devices) has a name property, which is used to identify the device when the kernel scans though 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 will be instantiated and matched with a driver.

The Wi-Fi Linux driver **package** has a *devicetree* folder, which includes iMX6SX and iMX6UL device tree blobs. If the host platform is not available in the device tree package, refer to the existing source files (*.dts*, *.dtsi* files) available in *arch/arm/boot/dts/,* and port the files to your target host platform. Following are the settings for each field of the *.dts, .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 l 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. In-band mechanism of interrupt uses SDIO DATA1 line to signal the interrupts. Out-of-band mechanism requires a dedicated GPIO pin. Make sure that the ping 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 best low power numbers, it is recommended to use OOB signaling methods, since in that mode the SDIO bus will be put into a suspended mode unless an interrupt triggered is on the WLAN_HOST_WAKE line when a packet is received. If no spare GPIOs are available in the host processor, you might choose to use the in-band interrupt method where the DATA1 line is repurposed to work as the interrupt, thereby preventing the bus from being suspended which adds on to the power burden.

Following are example implementations for iMX platforms 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 FullMAC (FMAC), the .compatible field expands to `of_device_is_compatible(np, "brcm,bcm4329-fmac"))`. For more details on Linux device tree, see this **blog post**.

2.3 Configuring kernel

For your kernel compilation, follow the vendor's instructions and set up the source and toolchain. You can find the in the vendor's distribution medium, mostly the GIT repository, for instance, **https://source.codeaurora.org/external/imx/linux-imx**). Based on the target's architecture (arm64, arm, x86, mips etc), you can select a default defconfig available in *arch/arm/configs*, and issue the following command to configure the kernel with the *.config* file:

```
$ make defconfig
```

Now, edit the *.config* file and build cfg80211 as module:

```
# CONFIG_CFG80211=m
```

If are not using legacy DHD as your driver, change the following:

```
# CONFIG_BCMDHD=n
```

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
```

Now, you can build the Linux kernel image for your target host. Here is an example for i.MX:

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

The kernel image is now available here: *arch/arm/boot/zImage*.

Infineon software artifacts available from quarterly train releases are supported, validated in a variety of platforms (for MMC/SDIO: NXP iMX6, NXP iMX8, for PCIe: iMX8, Intel NUC) in two primary ecosystems:

- Latest Google Android Open Source Platform (AOSP) version (with derivative support for Android TV, Wear OS etc)
  - Current version: Android 10, Kernel version 4.19LTS
- Latest Long-term Linux kernel release
  - Current version: 5.4.18LTS

Infineon's kernel support policy uses the **Backports** Project for FMAC driver-based chipset. This enables older kernels to run newest software. If the kernel version is not the latest LTS (currently 5.4.18), execute the Backports package in your development environment to enable the latest connectivity software (firmware and drivers) in your design. Now, you are ready to flash your host processor with the modified dtb, kernel image, and load the (backported) kernel modules (KM).

2.4 WLAN host interface

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

2.4.1 Multimedia card

The Multimedia Card (MMC) is a low-cost data storage medium, from which Secure Digital (SD)standard evolved. The I/O card variant combines high-speed serial data input/output lines with low-power consumption making it very suitable for battery-powered electronic devices; a typical use case being IoT devices. The Wi-Fi solution is pre-packaged with the device-driver, so you only need to take care of the driver implementation in SDIO Host (SDHC, MMC interface in the host needs to be carefully mapped to a SDIO host interface to communicate with Wi-Fi chip). **Table 2** lists the Wi-Fi chipsets which support MMC/SDIO interface. In addition to the latest support status with every quarterly release trains, see the Linux and Android **technical brief**.

Table 2 Wi-Fi devices with SDIO as host interface

Antenna Configuration	802.11 protocol	IFX Wi-Fi Chip	Supported (Y/N)?
1*1 SISO	802.11n	CYW43362	Y
		CYW43364	Y
		CYW43340	Y
		CYW4343W	Y
		CYW43438	Y
1*1 SISO	802.11n	CYW43439	Y
		CYW43012	Y
	802.11ac	CYW43455	Y
		CYW4373	Y
2*2 MIMO		CYW4354	Y
		CYW4356	Y
		CYW43570	Y
		CYW54591	N
2*2 MIMO	802.11ax/6	CYW55572	Y

2.4.2 PCIe

Peripheral Component Interconnect Express (PCIe), is a high-speed serial bus which 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 PCIe interface.

Table 3 Wi-Fi Devices with PCIe as host interface

Antenna Configuration	802.11 protocol	IFX Wi-Fi Chip	Supported (Y/N)
1*1 SISO	802.11n	CYW43362	N
		CYW43364	N
		CYW43340	N
		CYW4343W	N
		CYW43438	N
		CYW43439	N
		CYW43012	N
1*1 SISO	802.11ac	CYW43455	N
		CYW4373	N
2*2 MIMO	802.11ac	CYW4354	N
		CYW4356	Y
		CYW43570	Y
		CYW54591	Y
2*2 MIMO	802.11ax	CYW55572	Y

2.4.3 USB

USB is a de-facto communication medium for plugging or connecting a device to PC. Functionality of this class of devices can range from a storage medium to a Wi-Fi, Ethernet dongle even. In the Wi-Fi portfolio, CYW4373 (1*1 802.11ac) and CYW43569 (2*2 802.11ac) support USB interface.

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 userspace processes. It was designed to be 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
Does it provide multicast?	Yes	No
Does it require polling?	No	Yes
Is it asynchronous?	Yes (It provides message queues)	No

The way netlink sockets operate is quite simple; you open and register a socket in user-space and that handles all sorts of communications with a kernel netlink socket. Netlink has some advantages over other ways of communication between the userspace and the kernel. For example, there is no need for polling when working with netlink sockets. A userspace 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 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 in the user-space; kernel netlink sockets are created by the `netlink_kernel_create()` method; and userspace netlink sockets are created by the `socket()` system call. Creating a netlink socket from userspace or from the kernel creates a `netlink_sock` object. When the socket is created from userspace, 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 in the common way (by calling the `sk_alloc()` method) and initialize it.

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

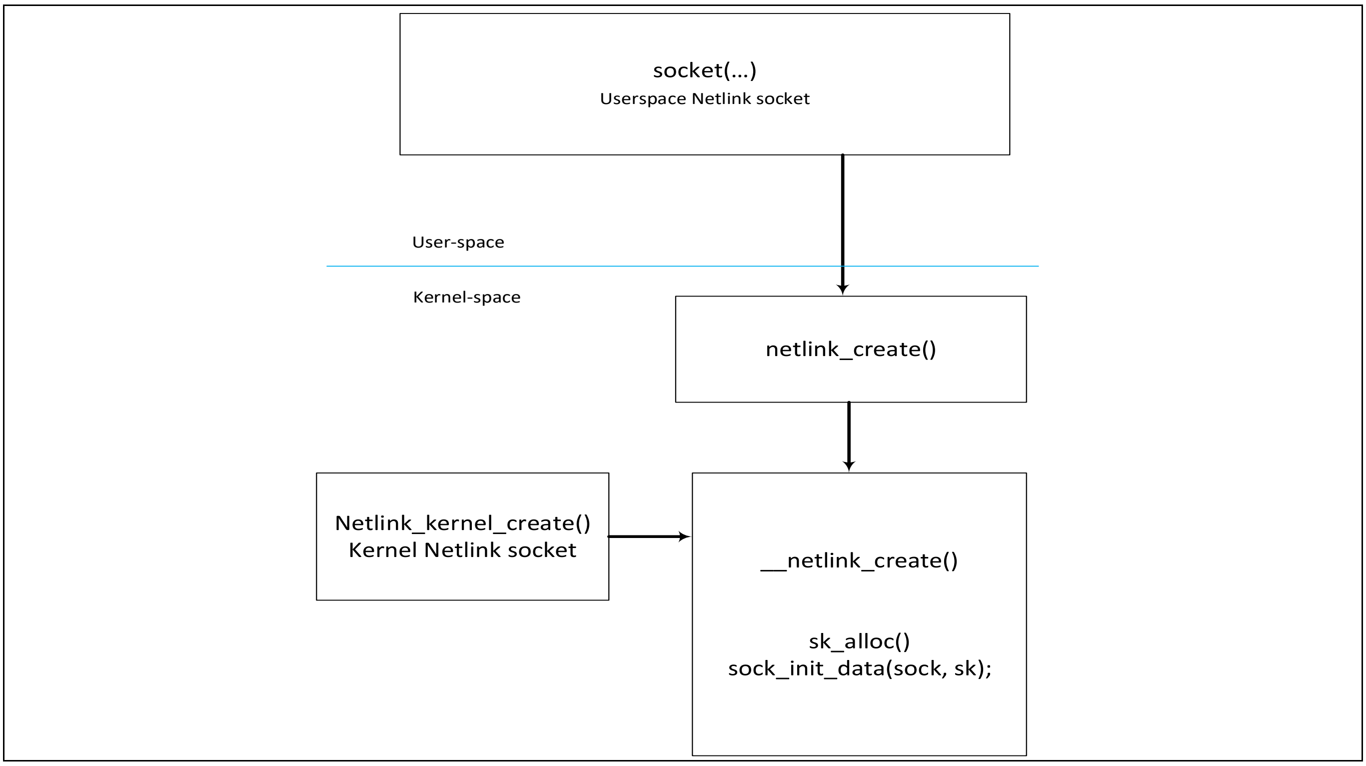

Figure 3 Netlink process flow

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).

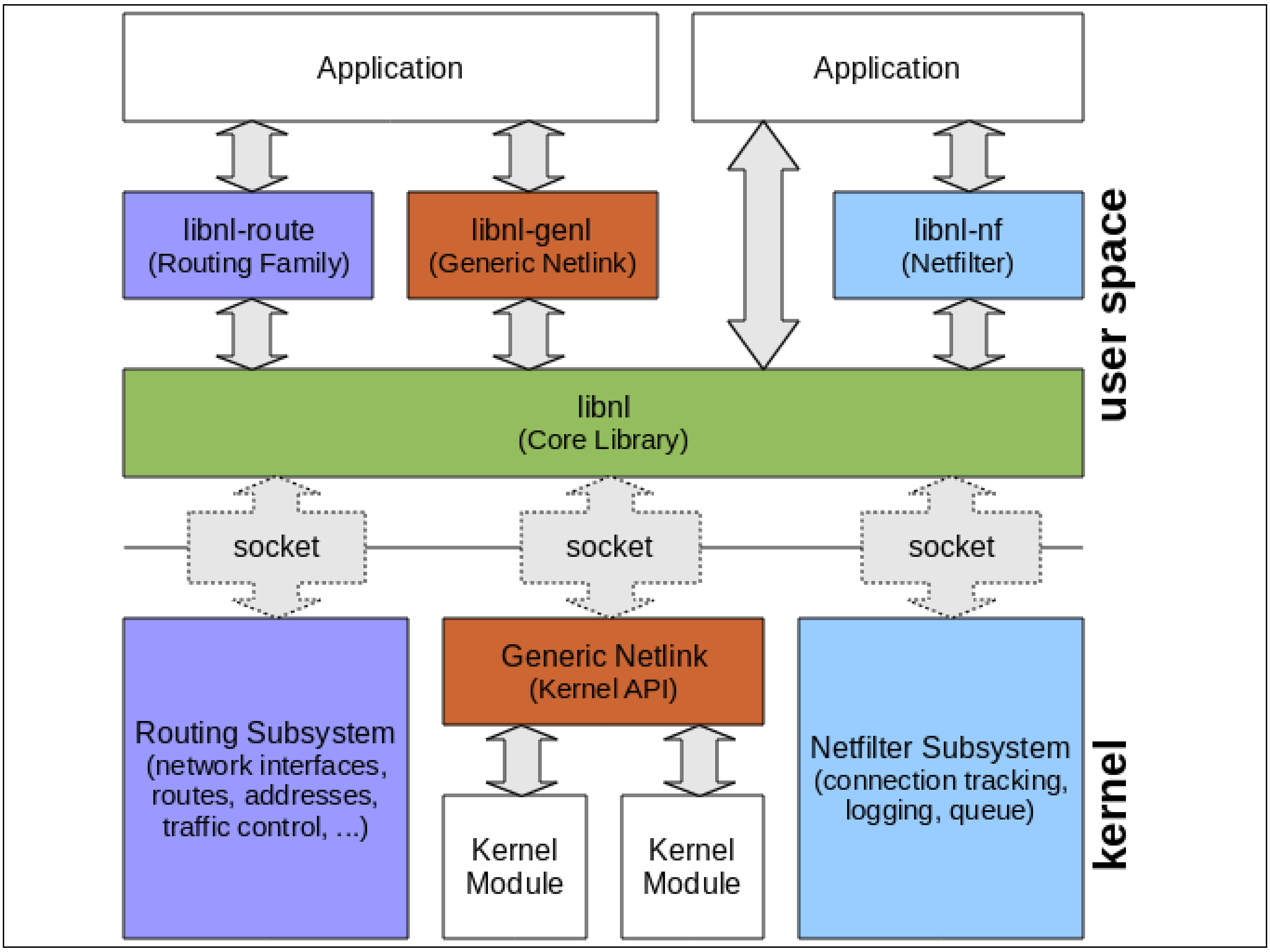

Figure 4 Netlink family

Image Source: **libnl - Netlink Protocol Library Suite (infradead.org)**

3.2 CFG80211

CFG80211 is primarily responsible for the configuration APIs for 802.11 devices in Linux. It provides management interface between kernel and userspace via nl80211. For backward compatibility, cfg80211 also offers wireless extensions (WEXT) to userspace, but abstracts them out from the driver layer completely. Additionally, cfg80211 contains code to help 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 a number of hardware capability structs, which is 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` pofinter 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 device capability information, and also 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 and you can actually skip knowing about the complexities related to cfg80211 and continue with application development. If you want to customize the driver with some additional cfg80211_ops, see *cfg80211.h* for further 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+. Since, this driver is a part of Linux kernel, anyone can upstream changes. Following are some key attributes of the FMAC driver:

- Supports SDIO, PCIe, USB interfaces with single binary
- Supports major features like softAP, P2P, TDLS, and so on (please refer to README provided with Infineon FMAC driver release package for specifics related to each chip).
- Since FMAC is part of kernel, supporting different kernel version becomes easy

1. Download the latest supported Linux kernel source from Infineon github.

```
$ git clone -b latest-v5.4 https://github.com/cypresssemiconductorco/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. There are two options for the firmware of the Wi-Fi chips:

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

b) Update to the latest firmware available through quarterly releases from Infineon. For updating the firmware, issue the following commands:

```
$ git clone -b latest-v5.4 https://github.com/cypresssemiconductorco/ifx-linux-firmware.git
$ cp ifx-linux-firmware/firmware/* /lib/firmware/cypress
```

Now, reboot your device with the freshly compiled kernel image and use the latest Wi-Fi features.

After rebooting, you can use `dmesg` to check the chip ID and additional information such as firmware version, firmware id, compilation date, and so on.

- Use "`modprobe cfg80211`" and "`modprobe sdhci-pci`" to insert all dependent modules that brcmfmac needs.
- While insmoding the driver, you can pass the parameters, listed in Table 5, as arguments to LKM. Here is an example:

```
$ 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 **Debug notes**.	int
feature_disable	Disable features	int
alternative_fw_path	alternative firmware path; i.e if firmware present in different path other than /lib/firmware/cypress.	string
fcmode	Mode of firmware-controlled flow control	int
roamoff	Do not use 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 Backports

Backports is a Linux official project that enables old kernels to run the latest drivers. For example, it enables Linux 5.4 FMAC driver to run on 4.14 or even 3.x Linux kernel.

Backports project contains a set of scripts, patches, and source code. Backport 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 drivers for running on 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 *.config* when building kernel.

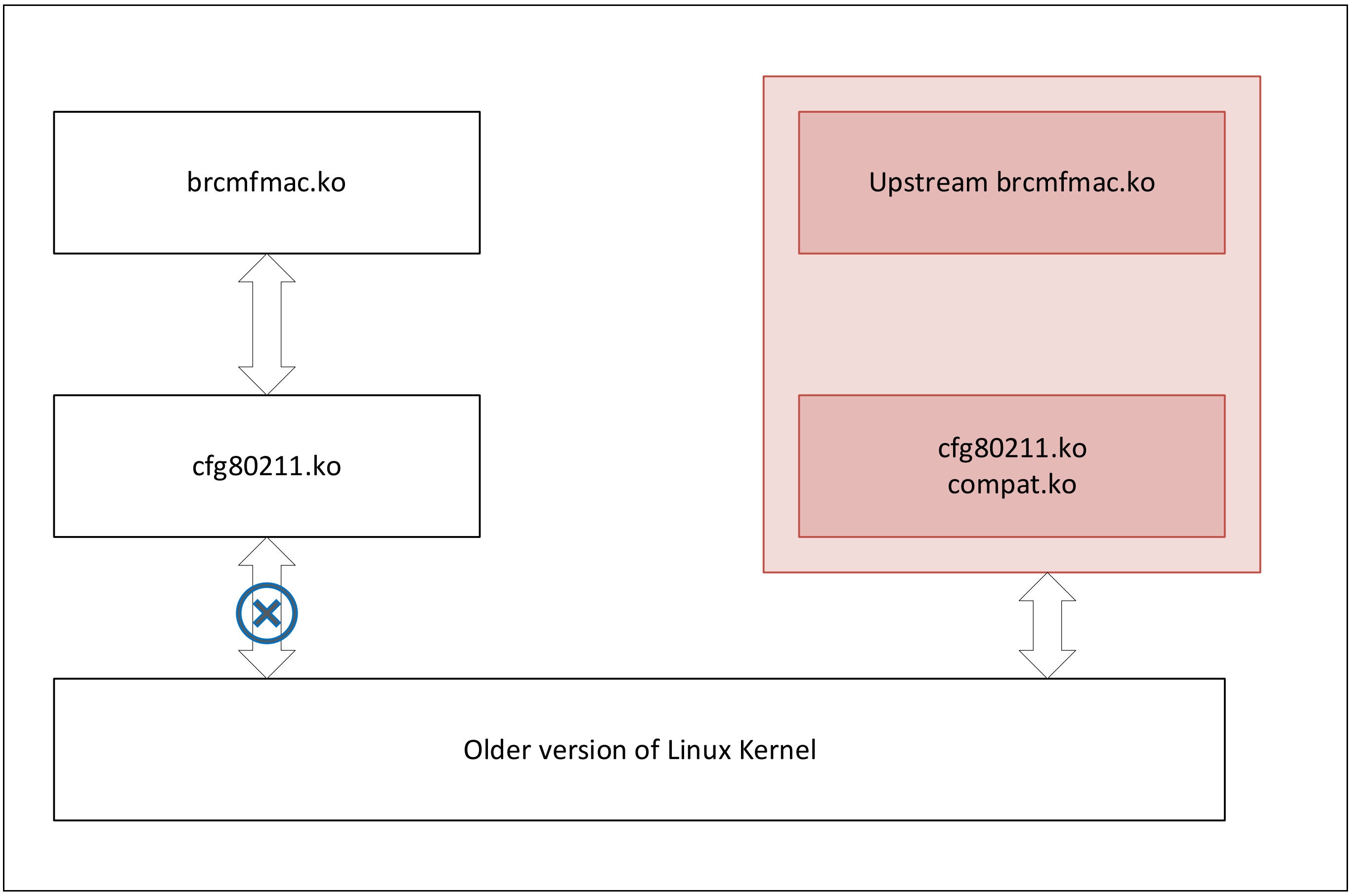

Figure 5 Backports package

Infineon supports the package release mode of the backports package, that is, the target type is loadable modules instead of kernel-integration, so that the kernel source remains untainted, supports multiple versions of kernel, eliminates Board support Package (BSP) specific dependencies, and so on. You can integrate the backports **release** package, with your kernel by following through the steps mentioned below.

```
$ git clone -b latest-v5.4 https://github.com/cypresssemiconductorco/ifx-backports.git

$ cd ifx-backports/v5.4.18-backports
$ cp brcmfmac defconfigs/.
$ make KLIB=$MY_KERNEL KLIB_BUILD=$MY_KERNEL defconfig-brcmfmac
# For cross-compilation, you need to source the toolchain before running make commands (modify the folder based on your host processor).
$ source /opt/poky/1.8/environment-setup-cortexa7hf-vfp-neon-poky-linux-gnueabi
$ 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
```

3.3.2 Cross-compilation

To cross-compile FMAC, for example on an Android host, you need to get the Android toolchain suitable for your target platform. Get the toolchain from **GNU Toolchain | GNU-A Downloads – Arm Developer** and place it in any directory (for instance, *$HOME/imx8mq/*)

Table 6 Toolchain prefixes for standard architecture

Architecture	Toolchain Name
ARM-based	armv7a-linux-android-<clang-version>
x86-based	x86-<clang-version>
MIPS-based	mipsel-linux-android-<clang-version>
ARM64-based	aarch64-linux-android-<clang-version>
x86-64-based	x86_64-<clang-version>
MIPS64-based	mips64el-linux-android-<clang-version>

Do the following to cross-compile the FMAC driver (i.MX8 is the reference platform here):

1. `export MY_ANDROID=$HOME/imx8mq/android_build`
2. `export MY_KERNEL=$HOME/imx8mq/android_build/out/target/product/evk_8mq/obj/KERNEL_OBJ`
3. `export AARCH64_GCC_CROSS_COMPILE=$HOME/imx8mq/gcc-arm-8.3-2019.03-x86_64-aarch64-linux-gnu/bin/aarch64-linux-gnu-`
4. `export PATH=${MY_ANDROID}/prebuilts/clang/host/linux-x86/clang-r353983d/bin:$PATH`
5. `export PATH=${MY_ANDROID}/prebuilts/gcc/linux-x86/aarch64/aarch64-linux-android-4.9/bin:$PATH`
6. `export PATH=$HOME/imx8mq/android_build/out/target/product/evk_8mq:$PATH`
7. `export _JAVA_OPTIONS="-Xmx4g"`
8. Patch the FMAC driver as mentioned in the **Backports**
9. Configure FMAC using the following command:

```
$ make KLIB=$MY_KERNEL KLIB_BUILD=$MY_KERNEL ARCH=arm64 CC=clang
CLANG_TRIPLE=aarch64-linux-gnu- CROSS_COMPILE=aarch64-linux-android- defconfig-
brcmfmac
```

10. Now, compile the FMAC driver modules:

```
$ make KLIB=$MY_KERNEL KLIB_BUILD=$MY_KERNEL ARCH=arm64 CC=clang
CLANG_TRIPLE=aarch64-linux-gnu- CROSS_COMPILE=aarch64-linux-android- modules
```

11. The compiled ernel modules are available here

```
compat/compat.ko, net/wireless/cfg80211.ko,
drivers/net/wireless/broadcom/brcm80211/brcmutil/brcmutil.ko,
drivers/net/wireless/broadcom/brcm80211/brcmfmac/brcmfmac.ko
```

3.3.3 Loading the FMAC driver

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

- Make sure you have firmware, clm_blob, and nvram present in the */lib/firmware/cypress* folder.
- Make sure that all binaries have the prefix cyfmac<chip_name>-<bus_name>.bin/clm_blob/txt.

- Follow this sequence of insmod:

```
$ insmod compat.ko
$ insmod cfg80211.ko
$ insmod brcmutil.ko
$ insmod brcmfmac.ko
```

3.3.4 Debug notes

In case you have run-into some kernel crash issue due to some problem with the Wi-Fi driver or firmware, you would want to enable the debug prints in FMAC driver. For that purpose, you can follow the steps mentioned below to compile the kernel modules with debug prints enabled.

1. If you are building the brcmfmac kernel modules against the kernel running on the system:

```
CPTCFG_BRCM_TRACING=y
CPTCFG_BRCMDBG=y
CPTCFG_BRCMFMAC_PROTO_BCDC=y
CPTCFG_BRCMFMAC_PROTO_MSGBUF=y
CPTCFG_CFG80211_WEXT=y
```

   To build a new kernel image, modify the *.config* file of kernel source with the following:

```
CONFIG_BRCMDBG=y
CONFIG_DEBUG_FS=y
```

2. To compile brcmfmac as LKM against the running kernel, run the command:

```
make -C <path_to_kernel_src> M=<fmac_source_dir>
```

   For instance:

```
$ make -C /lib/modules/`uname -r`/build M=$PWD
```

3. Enable the brcmfmac debug log:

```
$ echo 8 > /proc/sys/kernel/printk
```

4. Insert the driver module with the required message level as the module parameter.

```
$insmod brcmfmac.ko debug=${BRCMF_Message_Level}
```

   Following are the message levels 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 instance,

To enable Wi-Fi firmware console (ring buffers meant to hold debug prints inside Wi-Fi firmware) log:

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

To set console polling interval (250ms),

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

To enable Trace:

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

For further details on the functions associated with debugging FMAC, see the source code available in */v4.14.52-backports/drivers/net/wireless/broadcom/brcm80211/brcmfmac/debug.c*.

3.3.5 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. Also, the architecture might vary between the compiled kernel module and the host platform.

   Solution:

   Download the correct kernel and reinstall the correct kernel image.

2. Unknown Symbol:

```
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 brcmfmac insmod.

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”, 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 Cypress-specific patches were not applied for brcmfmac clm_blob might not have the download facility.

Solution:

- Copy the clm_blob file from quarterly release train to /lib/firmware/cypress
- See **AN225347** to understand the clm_blob flow. Then, 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 Cypress-specific brcmfmac instead of Linux native FMAC .

4. brcmfmac or btsdio modules are loaded automatically after “modprobe sdhci-pci”

Root Cause:

Modules are loaded if the kernel finds the device IDs in modules ID table.

Solution:

Add the modules in `/etc/modprobe.d/blacklist.conf`

`“blacklist btsdio”`

`“blacklist brcmutil”`

`“blacklist brcmfmac”`

5. USB dongle is not brought up after “insmod brcmfmac”

Root Cause:

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

Solution:

If nvram parameters are not present, contact Support to build a firmware with nvram and replace the pre-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. Following are some of the key features of wpa_supplicant:

- 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**, **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

- 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 the various cryptography functions of OpenSSL's crypto library from the shell.

- Dbus (optional)

  D-Bus is a message bus system, 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 is managing. The control interfaces are not active. by default. You need the -u commandline option to get dbus, and -O /var/run/wpa_supplicant (or whichever directory) for the sockets.

4.1.2 Compilation

- Get the latest source from wpa_supplicant from their **website**
- Migrate to `wpa_supplicant-2.9/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 and for that uncomment `CONFIG_ANDROID_LOG=y`. There are several other debug-specific macros that you can uncomment based on your requirements. Now, `cp defconfig .config` for further processing.
- To compile the wpa_supplicant source files, use the following command:

```
$ make <CC=arm-linux-gnueabi-gcc>
```

- To install the compiled binaries at a particular location, use the following command:

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

Note: *You might come across such as missing the header file in openssl or version mismatch. Make sure that you have followed the dependency section and installed them according to the exact version requirement. For example, in a ubuntu-based system, for libssl.so or libcrypto.so, most times, sudo apt-get install libssl-dev should be sufficient but sometimes 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 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 full (absolute, not relative to working directory) path to allow the working directory to be changed.

Example:

```
# allow frontend (e.g., wpa_cli) to be used by all users in '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 details on other specific macros relevant in wpa_supplicant.conf, visit this **link**.

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

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

Where,

-B is used to run the wpa_supplicant daemon in the background.

-c option is used to provide the configuration file, in this case, example.conf.

-i option is used to select the network interface to be used.

For additional debug prints, you can add -d parameter while running the command. For more details, see the **man** page of wpa_supplicant.

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. You need to manually remove this file before restarting wpa_supplicant. The command to do that is:*

```
$killall wpa_supplicant
```

4.1.4 wpa_cli

The Wpa_cli utility is a text-based frontend program for interacting with wpa_supplicant. It is used to query the current status, change configuration, trigger events, and request interactive user input. Additionally, the utility can configure EAPoL state machine parameters and trigger events such as reassociation and IEEE 802.1X logoff/logon.

The **wpa_cli** utility supports two modes: interactive and command line. Both modes share the same command set and the main difference is that the interactive mode provides access to unsolicited messages (event messages, 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 Options for configuration

The options listed in **Table 7** are available as argument to configure wpa_cli.

Table 7 Options to start wpa_cli

Command	Description
-p path	Controls 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.

Command	Description
-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

You can issue the commands listed in **Table 8** on the command line or at a prompt when operating interactively.

Table 8 WPA_CLI commandsi

Command	Description
help	Shows usage help.
status	Reports the current WPA/EAPOL/EAP status for the current interface.
ifname	Shows the current interface name. The default interface is the first interface found in the socket path.
ping	Pings the wpa_supplicant utility. This command can be used to test the status of the wpa_supplicant daemon.
mib	Reports MIB variables (dot1x, dot11) for the current interface.
interface [ifname]	Shows the available interfaces, sets the current interface, or does both when multiple interfaces are available.
level *debug_level*	Changes the debugging level in wpa_supplicant. Larger numbers generate more messages.
license	Displays the full license for wpa_cli.
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.
set [settings]	Sets variables. When no arguments are supplied, the known variables and their settings are displayed.
pmksa	Shows the contents of the PMKSA cache.
reassociate	Forces a reassociation to the current access point.
reconfigure	Forces wpa_supplicant to re-read its configuration file.
preauthenticate *BSSID*	Forces preauthentication of the specified *BSSID*.
identity *network_id identity*	Configures an identity for an SSID.
password *network_id password*	Configures a password for an SSID.
new_password *network_id password*	Changes the password for an SSID.

Command	Description
help	Shows usage help.
PIN *network_id pin*	Configures a PIN for an SSID.
passphrase *network_id passphrase*	Configures a private key passphrase for an SSID.
bssid *network_id bssid*	Sets a preferred BSSID for an SSID
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.
list_networks	Lists the configured networks.
select_network *network_id*	Selects a network and disable others.
enable_network *network_id*	Enables a network.
add_network	Adds a network.
remove_network *network_id*	Removes a network
set_network *[network_id variable value]*	Sets network variables. Shows a list of variables when run without arguments.
get_network *network_id variable*	Gets network variables.
disconnect	Disconnects and waits for reassociate/reconnect command before connecting.
reconnect	Similar to reassociate, but only takes effect if already disconnected
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.
bss [*idx* | *bssid*]	Gets a detailed BSS scan result for the network identified by "bssid" or "idx".
otp *network_id password*	Configures a one-time password for an SSID.
terminate	Forces wpa_supplicant to terminate.
interface_add *ifname* [*confname driver ctrl_interface driver_param bridge_name*]	Adds a new interface with the given parameters.
interface_remove *ifname*	Removes the interface
interface_list	Lists the available interfaces
quit	Exits wpa_cli.

Note: *If you encounter the error "RFkill Soft blocked" while running any of the wap_cli or wpa_supplicant commands, you can use the following command.*

```
$sudo rfkill unblock all
```

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`

STA/AP combinations	wpa_cli commands
	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" 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

STA/AP combinations	wpa_cli commands
	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 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 security of WLANs. It is more secured than WEP or WPA. For WAPI-related support, contact your local Infineon Sales office or your 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 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 userspace daemon for setting up access point or authentication servers with fine granular control over most of the parameters. It implements 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 frontend programs like hostapd_cli.

4.2.1 Dependencies of hostapd

- Libnl
- Openssl

4.2.2 Compilation of hostapd

Do the following to compile the hostapd

1. Download the hostapd source **package** and migrate to the root folder.
   ```
   $ cd hostapd-2.9/hostapd
   ```
2. Copy the existing defconfig file to .config. Ensure that the following flags are set.
   ```
   $ cp defconfig .config
   ```
3. `$ vi .config`
4. Now, make sure that the following parameters are set,
   ```
   CONFIG_DRIVER_NL80211=y
   CFLAGS += -I/usr/include/libnl3
   CONFIG_LIBNL32=y
   ```

To compile the hostapd utility, you can use the make command as mentioned below.

```
$ make CC=arm-linux-gnueabi-gcc
```

To install the hostapd utility, you can issue the below command and place it in say /usr/sbin directory (a popular choice)

```
$ make install DESTDIR=<target directory >
```

4.2.3 Conf files

Hostapd is configured using text file that sets up the Access Pont’s (AP’s) security policies (802.11i, 802.1X etc), country code, passphrase, and so on. Create a *hostapd.conf* file. Here is an example conf file:

```
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. Can be removed once bring-up of a platform is over.

4.2.3.1 Hostapd usage

Hostpad 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
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 choices available for DHCP daemon; such as udhcp, dhcp, dnsmasq, and so on. Most of them are used as a dhcp server and some of them additionally provide the DNS server functionality. Sometimes, they are included by default with core OS like udhcp; otherwise, you can manually install the packages. (for example, dnsmasq). In this case, dnsmasq is considered as an example to demonstrate 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 options to configure the dhcp server. So, 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 nl80211 based userspace command line utility used to configure wireless devices. It supports both Wi-Fi drivers used by the 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. 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 Compilation

You can use the package manager tool for your system to install the above packages and then proceed with installation of iw. If you choose to compile from source, release tarballs are available in this **link**.

4.3.3 Typical usage

Using iw is really easy. The command, `iw list,` provides you with the capabilities of your wireless device(s) 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

The following commands help you 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

# udhcpd ./udhcpd_wlan0.conf
```

For the AP-related security configuration, you can use *hostapd.conf*, with the `interface` parameter as `softap` as mentioned in **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 only handle the connection process with either open security or WEP security. For WPAx security, it is recommended to 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 favour of the more robust security measures as 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 Yocto release, follow this build procedure:

1. Extract the build scripts tarball

```
$ tar zxvf cypress-yocto-scripts-v5.4.18-2020_0925.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-v5.4.18-2020_0925.zip
* build_yocto_wireless.sh
* meta-cywlan
* nvram.zip
* bt-firmware.tar.gz
$ cp cypress-fmac-v5.4.18-2020_0925.zip cypress-imx-bsp
  $ cp -r cypress-yocto-scripts-v5.4.18-2020_0925/meta-cywlan cypress-yocto-scripts-v5.4.18-2020_0925/build_yocto_wireless.sh cypress-yocto-scripts-v5.4.18-2020_0925/nvram.zip cypress-yocto-scripts-v5.4.18-2020_0925/bt-firmware.tar.gz cypress-imx-bsp
```

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

```
$ cypress-yocto-scripts-v5.4.18-2020_0925/setup_host_env.sh
```

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

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

If the scripts are unable to be run by user permission, use:

```
$ 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 (v5.4.18), so older kernels need to use backports package. Here are some examples on 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 kernel, Linux kernel image and cypress wifi driver modules need to be built separately.
Building the kernel image is done by following the steps mentioned in **Device tree blob**. For cypress wifi driver backports modules, you can follow the steps mentioned in **Backports**.

5.3 Upgrading firmware

Usually, the quarterly releases contain the updated firmware. If you decide to upgrade to the latest firmware, run the following command.

```
$ modprobe brcmfmac.ko
```

References

[1] **Device Tree Structure**

[2] **Linux Wireless**

[3] **Linux Device Drivers**

[4] **Linux MMC Subsystem**

[5] **Linux PCI Bus Subsytem**

[6] **Linux USB Subsytem**

Revision history

Document version	Date of release	Description of changes
**	2021-04-16	Initial release

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2021-04-16

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Go to www.cypress.com/support

Document reference

002-32689 Rev. **