Wi-Fi 软件用户手册

关于本手册

使用范围和目的

本文档概述了搭建 Linux 802.11 系统的过程，便于用户根据自身主机、实际应用便捷使用 Wi-Fi 模块。

适用人群

本文档适用于使用 Linux 主机，且使用英飞凌 Wi-Fi 方案的用户。用户最好有 Linux 内核网络搭建经验，或了解 Linux 主机处理器启动流程相关知识。

Note: ***本文档中所有术语和缩写说明见 Wi-Fi 术语表。***

目录

目录

1 Wi-Fi 软件介绍

英飞凌 Wi-Fi 软件能无线接收和发送 802.11 帧，是搭建 Wi-Fi 使用环境中至关重要的一环。本指南帮助用户了解 Linux 内核网络子系统及用户空间设置 WLAN 所需的 wpa_supplicant, hostapd, iw 等工具。本指南涵盖用户空间特点、内核空间特点，以及修改或配置 Wi-Fi 设备的驱动程序。

1.1 Wi-Fi 软件块架构框架

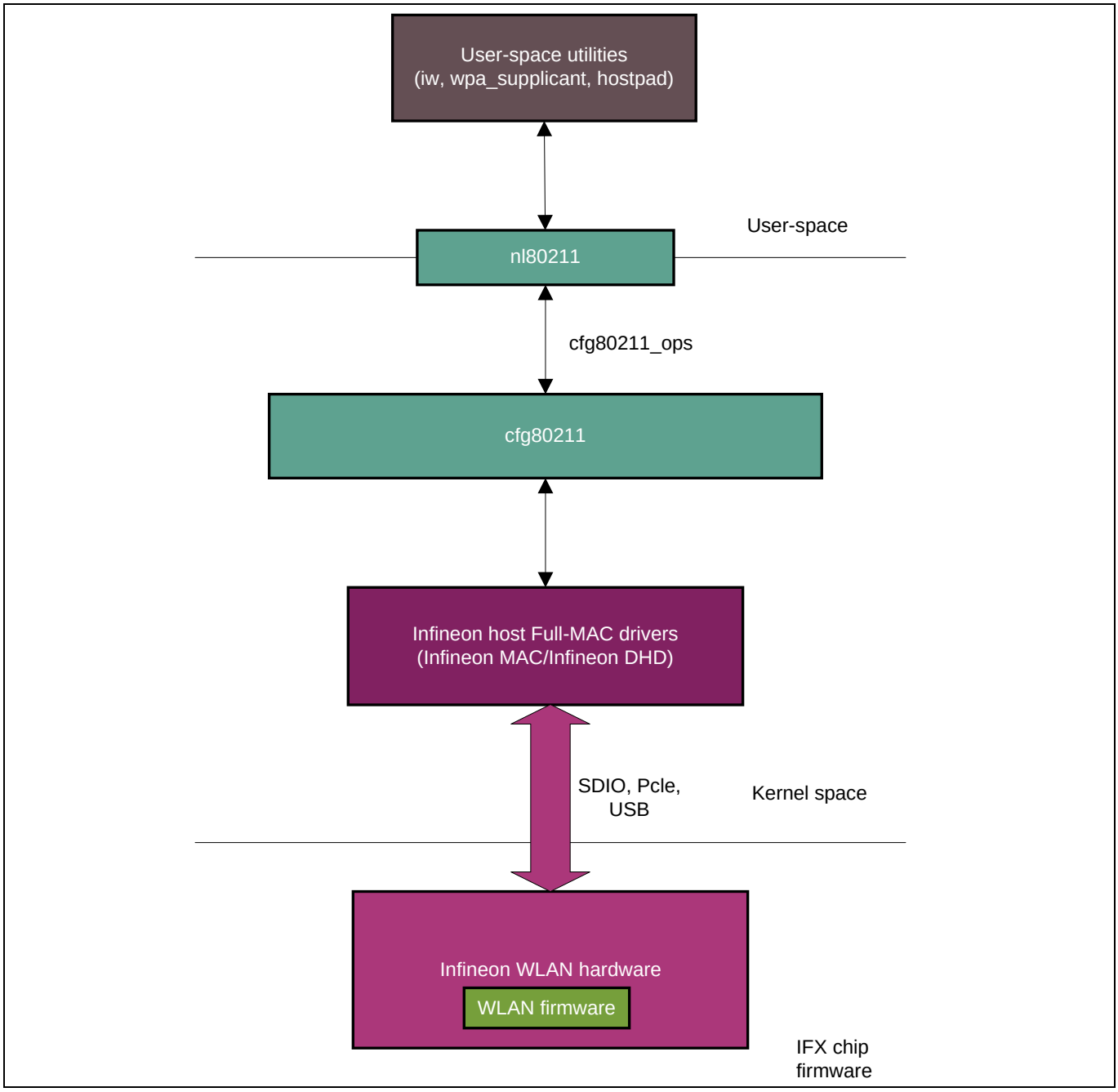

Figure 1 Linux 802.11 架构- 有删节

Wi-Fi 软件包含用户空间应用（最顶层内含 iw, wpa_supplicant, 以及 hostapd）和固件层（嵌入于英飞凌 Wi-Fi 芯片中），两者中间还包括发送数据、接收数据和事件路径等。中间层支持各种数据流：发送数据，接

Wi-Fi 软件介绍

收数据和事件机制。基于该架构，设备驱动程序（每季度由英飞凌打包并释放）可在主机与从设备间进行流量控制或事件队列机制管理。最底层负责核心 802.11 协议运行，且包含部分总线硬件。该层内置于英飞凌 Wi-Fi 固件，无需额外设备驱动程序，显著缩短上市周期。

2 平台接口，启动流程，及设备树二进制文件（DTB）

2.1 硬件连接

主机处理器与目标 Wi-Fi 设备可以通过以下几种方式连接:

- 总线连接：通过双向总线连接主处理器与目标 Wi-Fi 设备。
- SDIO：通过 D0，D1，D2，和 D3 连接； PCIe：通过 TDN，TDP， RDN，和 RDP 连接；USB：通过 DP 和 DN 等连接。蓝牙连接通常使用 UART，连接通过 RX，TX，RTS，和 CTS 等。

Table 1 主处理器与目标 Wi-Fi/蓝牙 设备总线连接方式表

连接种类	总线	相应引脚（Pin）
主机处理器 <--> Wi-Fi 连接	SDIO	D0-D3
主机处理器 <--> Wi-Fi 连接	PCIe	TDN, TDP, RDN, and RDP
主机处理器 <--> Wi-Fi /蓝牙连接	USB	DP, DN
主机处理器 <--> 蓝牙连接	UART	RTS, CTS, TX, RX

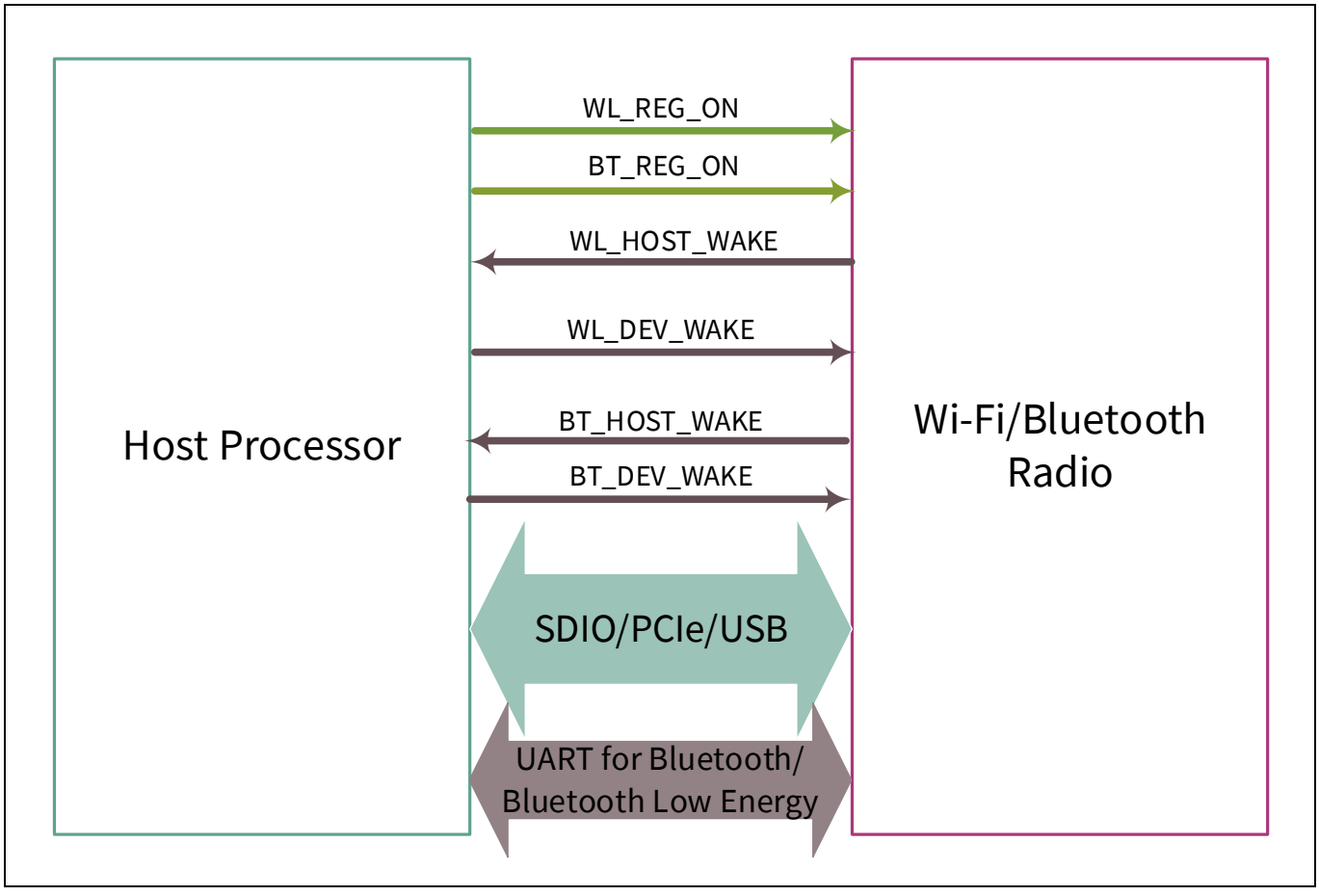

Figure 2 主机处理器连接 Wi-Fi/蓝牙/低功耗蓝牙块

平台接口，启动流程，及设备树二进制文件（DTB）

- 电源相关连接: 用于 Wi-Fi 模块上电的通用接口，包括 WL_REG_ON， BT_REG_ON 。
- 中断信号连接：用于唤醒主机处理器或 Wi-Fi 模块的通用接口，包括 WL_HOST_WAKE，WL_DEV_WAKE，BT_HOST_WAKE，和 BT_DEV_WAKE。

2.2 设备树二进制文件（DTB）

设备树用来描述平台硬件信息，如 I2C 和 SPI 设备数量等。设备树二进制文件（DTB）常以可读文件格式编译和保存，源文件扩展名常见为*.dts*，头文件扩展名常为*.dtsi*。*.dts* 文件用于描述板级的设备信息，*.dtsi* 文件用于描述 SoC 信息。设备树源文件需使用设备树编译工具（DTC）进行编译，源文件存储在内核目录： *<kernel_base>/scripts/dtc*。DTC 编译生成后缀为*.dtb* 的二进制文件，也即扁平设备树（FDT）。通过设备树数据，Linux 操作系统可以注册和查找设备信息。系统启动的初期阶段， FDT 文件被解析成展开设备树（EDT）文件，大大提升了后续启动阶段的数据解析效率。数据树中，信息通常包括设备 I/O 端口和中断线寄存器相关信息。每一个设备节点代表设备树中的一个设备，其节点名用于内核扫描识别设备。驱动中的“兼容性”属性明确了内核需要在设备树中寻找的设备，一旦找到对应的名字，相应的设备便会与该驱动绑定。

英飞凌 Wi-Fi Linux 驱动**包**附有 *devicetree* 文件夹，其中包括 iMX6SX 和 iMX6UL 设备树二进制文件（DTB）。如果主机平台无法使用该设备树包，在 *arch/arm/boot/dts/* 目录中找到现有源文件（后缀为*.dts*, *.dtsi*），将该源文件传输到目标主机平台。以下为*.dts*, *.dtsi* 文件设置：

- wlreg_on：该引脚(WL_REG_ON)用于 Wi-Fi 设备上电，必须以高电平模式启动。具体参见相应的 I 芯片数据表中电压要求。
- 带内架构/带外架构：通过 SDIO 连接至主机处理器的 Wi-Fi 设备有两种方式管理设备与主处理器间的中断。带内架构中，SDIO 的 DATA1 信号线复用于中断线。带外架构（out-of-band , OOB)则需要使用单独的 GPIO 引脚。请注意不要引脚多路复用，该引脚只能作为 GPIO 引脚使用。可根据实际应用情况选择带内架构或带外架构。为实现最低功耗，推荐使用 OOB 信号。在该模式下，SDIO 总路处于挂起状态，直到 Wi-Fi 收到数据包后 WLAN_HOST_WAKE 产生中断唤醒 CPU。若主机处理器上无多余 GPIO 接口，可使用带内架构法，将 DATA1 信号线复用为中断线，从而阻止总路进入挂起模式。该方法会增加功耗。

以下为 linux-imx/arch/arm/boot/dts/:目录中文件在 iMX 平台的中断管理设置具体应用：

- imx6ul-evk-btwifi-oob.dtsi 适用于 OOB 中断引脚分布与配置
- imx6ul-evk-btwifi.dtsi 适用于 WL_REG_ON 引脚相关设置

对于 FullMAC (FMAC)， 兼容字段扩展为 `of_device_is_compatible(np, "brcm,bcm4329-fmac"))`. 有关 Linux 设备树的更多详细信息，请参阅**此博客文章** 。

2.3 内核设置

内核编译请参照厂商的说明设置源代码和工具链。具体说明见各厂商官网专区，也可到 GIT 本地库进行下载（**https://source.codeaurora.org/external/imx/linux-imx**）。根据目标架构（arm64, arm, x86, mips 等）选择 *arch/arm/configs* 中可用的默认 defconfig。使用*.config* 文件配置内核，输入以下指令：

```
$ make defconfig
```

编辑*.config* 文件，将 cfg80211 编译为模块：

```
# CONFIG_CFG80211=m
```

如果你的驱动不是 DHD，按以下设置更改:

```
# CONFIG_BCMDHD=n
```

此外，在*.config* 文件中启用以下配置:

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

配置完成后，便可为目标 host 构建内核镜像。i.MX 平台构建内核镜像指令如下：

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

构建完成的内核镜像存储于：*arch/arm/boot/zImage*。

英飞凌软件包每季度发布，支持多个平台(MMC/SDIO: NXP iMX6, NXP iMX8； PCIe: iMX8, Intel NUC)，适用于两大主要生态系统：

- 最新 Google Android 开源平台（AOSP），可衍生支持 Android TV, Wear OS 等。
  - 当前版本：Android 10，内核版本：4.19LTS
- 最新发布的 Linux 内核长期支持版本
  - 当前版本：5.4.18LTS

在基于 FMAC 驱动的芯片组中，英飞凌内核支持方案使用 **Backports** 项目。因此，老版本内核也能运行最新软件，如果内核版本不是最新的 LTS（当前版本为 5.4.18），请在开发环境中执行 Backports 包，以启用您设备中最新连接软件（固件和驱动程序）。至此，您可以使用修改后的 DTB，内核镜像并加载（带 Backport）内核模块了(KM)。

2.4 WLAN host 接口

本节将介绍连接 Wi-Fi 设备与主机处理器的可用接口选项。

2.4.1 多媒体卡

多媒体卡（MMC）是一种低成本的数据存储卡，在此基础之上发展出了安全数字（SD）标准。由 MMC 演变而来的 I/O 卡在具备高速串行数据输入/输出线的同时，还拥有低功耗的特点，非常适合以电池供电的电子设备。这类型设备常见于物联网设备中。本产品已将 Wi-Fi 解决方案与设备驱动程序预打包，您只需要在 SDIO 接口主机中安装驱动（SDHC，MMC 接口需映射到 SDIO 接口才能与 Wi-Fi 芯片通信。**Table 2** 列出了 IFX 系列芯片对 MMC/SDIO 接口的支持信息。更多信息，请参照每季度发布的最新支持系列，或 Linux 以及 Android **技术简介**。

Table 2 SDIO 主机接口类 Wi-Fi 设备

天线配置	802.11 协议	IFX Wi-Fi 芯片	是否支持(是/否)?
1*1 SISO	802.11n	CYW43362	是
		CYW43364	是
		CYW43340	是
		CYW4343W	是
		CYW43438	是
1*1 SISO	802.11n	CYW43439	是
		CYW43012	是
	802.11ac	CYW43455	是
		CYW4373	是
2*2 MIMO		CYW4354	是
		CYW4356	是
		CYW43570	是
		CYW54591	否
2*2 MIMO	802.11ax/6	CYW55572	是

2.4.2 PCIe

高速外围组件互连（PCIe）是一种高速串行总线，通常用作 SSD、Wi-Fi、以太网等接口。具体版本和规格，参阅相应的芯片数据表。**Table 3** 列出了 IFX 系列芯片对 PCIe 接口的支持信息。

Table 3 PCIe 主机接口类 Wi-Fi 设备

天线配置	802.11 协议	IFX Wi-Fi 芯片	是否支持(是/否)?
1*1 SISO	802.11n	CYW43362	否
		CYW43364	否
		CYW43340	否
		CYW4343W	否
		CYW43438	否
		CYW43439	否

平台接口，启动流程，及设备树二进制文件（DTB）

天线配置	802.11 协议	IFX Wi-Fi 芯片	是否支持(是/否)?
		CYW43012	否
1*1 SISO	802.11ac	CYW43455	否
		CYW4373	否
2*2 MIMO	802.11ac	CYW4354	否
		CYW4356	是
		CYW43570	是
		CYW54591	是
2*2 MIMO	802.11ax	CYW55572	是

2.4.3 USB

USB 是一种事实上的通信介质，用于将设备插入或连接至 PC。此类设备的功能应用范围涵盖存储、Wi-Fi、以太网加密狗等。IFX Wi-Fi 芯片系列中，CYW4373 (1*1 802.11ac) 和 CYW43569 (2*2 802.11ac) 支持 USB 接口。

3 Linux 内核 802.11 子系统

3.1 NL80211

Netlink (nl80211) 协议是基于套接字的 IPC 通信方式，用于用户空间和内核空间的通信，或用户空间进程之间的通信。该协议是 ioctls 之后提出的，意在提供更灵活的内核相关网络配置和监控网络接口。Table 4 比较了传统基于 ioctls 的系统调用和基于套接字的 netlink 协议。

Table 4 基于 ioctls 的系统调用和 netlink 协议比较

性能	Netlink 套接字	系统调用
谁发起通信?	用户空间内应用程序以及内核模块	用户空间内应用程序
能否组播?	是	否
是否需要轮询?	否	是
调用是否异步?	是（提供消息队列）	否

Netlink 套接字运行方式很简单：在用户空间中打开并注册一个套接字，它将同内核 netlink 套接字处理所有通信。相比其他通信方式，Netlink 在内核与用户空间通信上有多个优势。它无需轮询，用户空间应用程序打开套接字，调用 `recvmsg()`，若内核未发送通信，则进入被动接收状态，如 iproute2 工具包 (`lib/libnetlink.c`)使用的 API，`rtnl_listen()`。

另一个优点是，netlink 套接字支持组播传输。使用 `socket()` 系统调用从用户空间中创建 netlink 套接字，可以是 `SOCK_RAW` 或 `SOCK_DGRAM`。套接字可创建于内核或用户空间中。调用 `netlink_kernel_create()`，可在内核中创建 netlink 套接字；调用 `socket()` 系统回调的方法，可在用户空间中创建 netlink 套接字。内核套接字设置 `NETLINK_KERNEL_SOCKET` 作为其标识。最后，两种方法都需要调用`__netlink_create()`函数。

0 为 netlink 在内核或用户空间中的创建流程。

Linux 内核 802.11 子系统

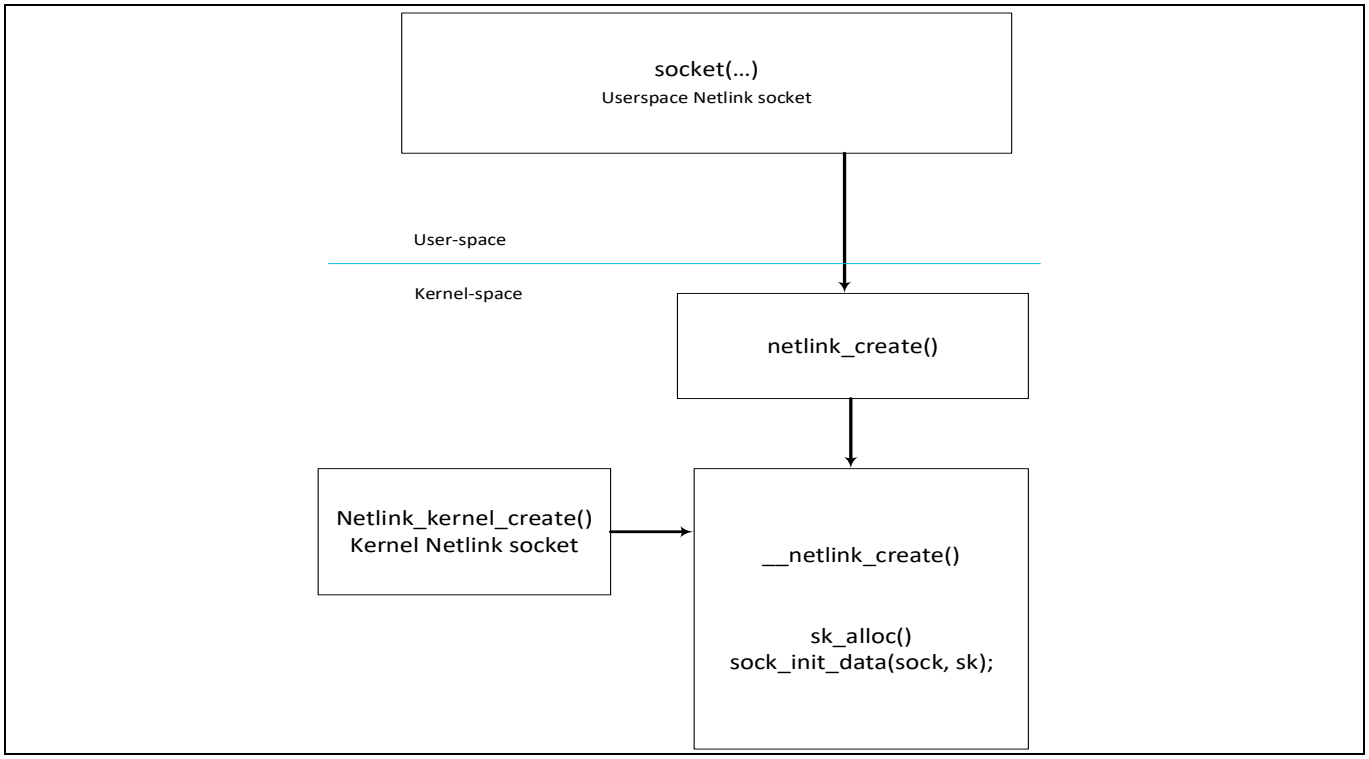

Figure 3 Netlink 创建流程

libnl 包是一个集合库，为基于 netlink 协议的内核接口提供 API。iproute2 包使用 libnl 库。除了核心库 libnl 以外，该软件包还支持 generic netlink 系列（libnl-genl），routing 系列（ libnl-route）以及 netfilter 系列（libnl-nf）。

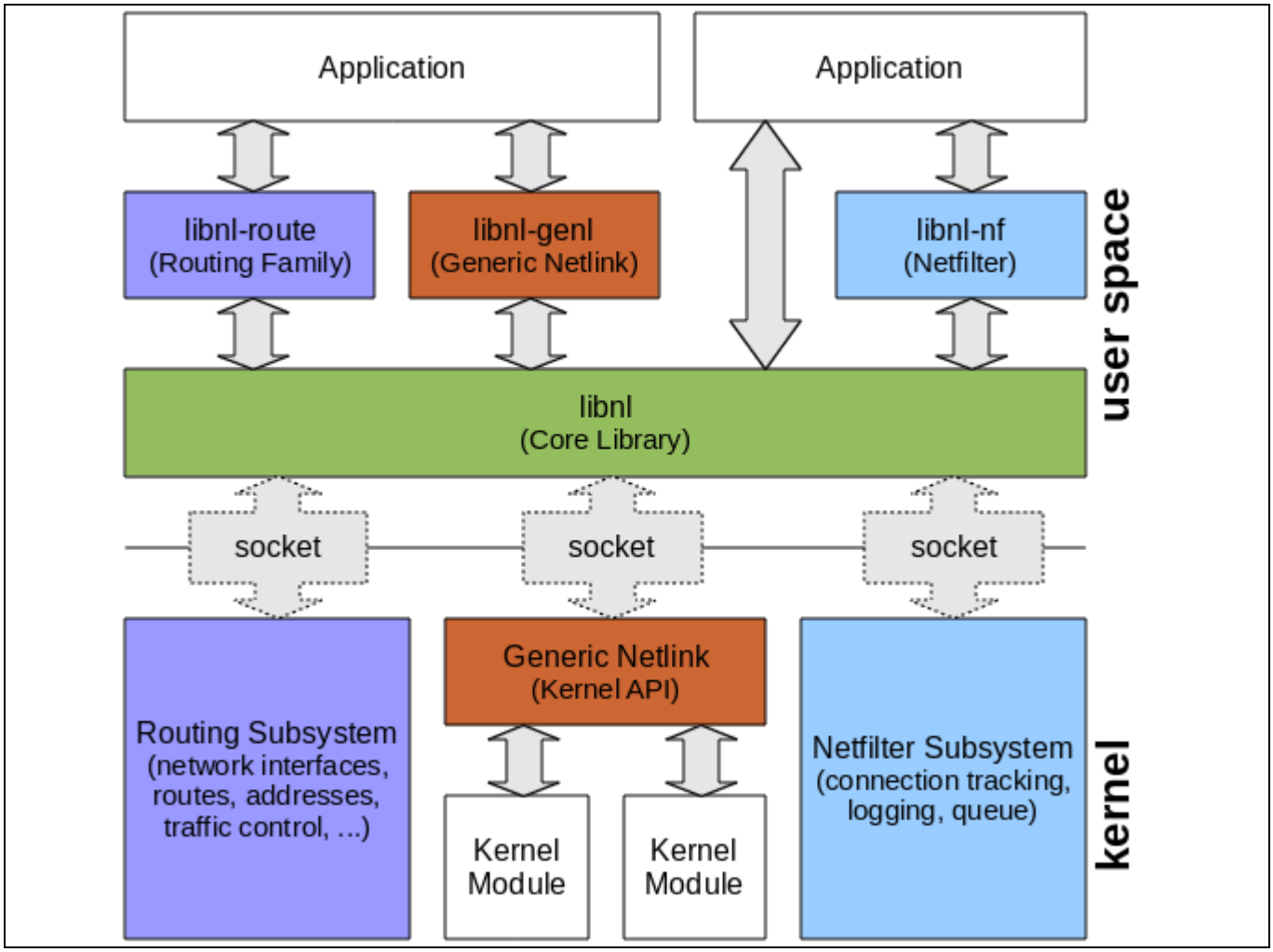

Figure 4　　Netlink 系列

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

3.2　　CFG80211

CFG80211 主要负责 Linux 中 802. 11 设备配置 API。它通过 nl80211 在内核与用户空间之间创建管理接口。为方便向后兼容，CFG80211 向用户空间提供无线扩展接口（WEXT），但其完全基于驱动层实现。除此之外，CFG80211 还能实现监管功率限制以及频谱管控。

驱动程序要使用 CFG80211，必须要在 CFG80211 中注册硬件设备。实现注册需要使用一系列硬件功能描述的结构体下，本节将对此进行详细阐释。每个设备的基本结构是“wiphy”，每个 wiphy 都定义了与系统相连的一个物理无线设备。每个 wiphy 可以有零个、一个或多个虚拟接口与之相关联。关联的虚拟接口需要通过将网络接口的 `ieee80211_ptr` 指向 `struct wireless_dev` 来识别，该结构体描述了接口的无线部分。通常情况下，这个结构体将嵌入于网络接口的私有数据区。驱动程序可以选择性地允许在运行中创建或销毁虚拟接口，但如果没有虚拟接口或没有创建虚拟接口的能力，无线设备就会失效。每个 wiphy 结构体都包含设备信息，也配有一个指向驱动程序提供的各种操作的指针。Wi-Fi 驱动厂商有责任提供 cfg80211 操作回调并配上准确描述设备信息的 wiphy 结构体。英飞凌发布的驱动程序包，已预先安

Linux 内核 802.11 子系统

装好所有与 cfg80211 相关的操作，你无需了解与 cfg80211 相关的复杂问题，直接进行应用开发。如果你想用其他的 cfg80211_ops 来定制驱动程序，请参阅 `cfg80211.h` 了解具体操作细节。

3.3 FMAC 激活

FMAC 描述了一种使用 mac 子层管理实体(**MLME**)进行硬件管理的无线网卡。Wi-Fi 组合中的所有芯片都属于这一类别。FMAC 驱动最初是在 Linux 内核 2.6+中引入的。由于这个驱动程序是 Linux 内核的一部分，任何人都可以在上游进行修改。以下是 FMAC 驱动的一些关键属性。

- 支持 SDIO、PCIe、USB 接口的单个二进制文件
- 支持 softAP、P2P、TDLS 等主要功能（与各芯片相关的具体参数请参考英飞凌 FMAC 驱动发布包中提供的 README）
- 由于 FMAC 是内核的一部分，轻松支持不同的内核版本

1. 从英飞凌 github 中下载最新支持的 Linux 内核源代码.

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

2. 修改默认的内核文件*.config*，运行下列指令，编译内核镜像:

```
#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. Wi-Fi 芯片固件版本有以下两个选项:

   a) 使用*/lib/firmware/cypress* 中的原始固件文件

   b) 通过英飞凌的季度发布包，更新到最新的固件。更新请运行以下指令:

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

用新编译的内核镜像重启设备，即可体验最新的 Wi-Fi 功能。

重启后，可以用 `dmesg` 检查芯片 ID 和其他信息，如固件版本、固件 ID、编译日期等等。

- 调用 "`modprobe cfg80211` "和 "`modprobe sdhci-pci` "来插入 brcmfmac 需要的所有依赖模块。
- 在插入驱动程序时，可以进行模块参数传递给 LKM，具体参数见 **Table 5**，如:

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

Table 5 FMAC 模块参数

模块参数	功能	模块参数类型
p2pon	启用传统的 p2p 管理功能	int
txglomsz	最大 tx 数据包链尺寸[SDIO]	int
debug	调试输出的级别。见 **调试说明**.	int
feature_disable	禁用功能	int
alternative_fw_path	替代固件路径；如果固件存在于 /lib/firmware/cypress 以外的其他路径。	string
fcmode	固件控制流量模式	int
roamoff	关闭内部漫游引擎	int
iapp	对旧 AP 进行部分支持	int
ignore_probe_fail	持续调试，直到 probe 成功	int

3.3.1 Backports

Backports 由 Linux 官方推出，能使旧内核得以运行最新的驱动程序。例如，它使 Linux 5.4 fmac 驱动能够在 4. 14 甚至 3.x Linux 内核上运行。

Backports 包含脚本、补丁和源代码。Backport 中输入了较新版本的内核树，输出生成 "Backports package "。使用 Backports package，便可编译在旧内核上运行的驱动程序。

Figure 5 是 Backports package 提供的 cfg80211 修改版本的大致路径，通过单独的 compat 模块实现向后兼容。注意，在构建内核时，需要在.config 中禁用原来的 cfg80211 和 brcmfmac（旧内核中）。

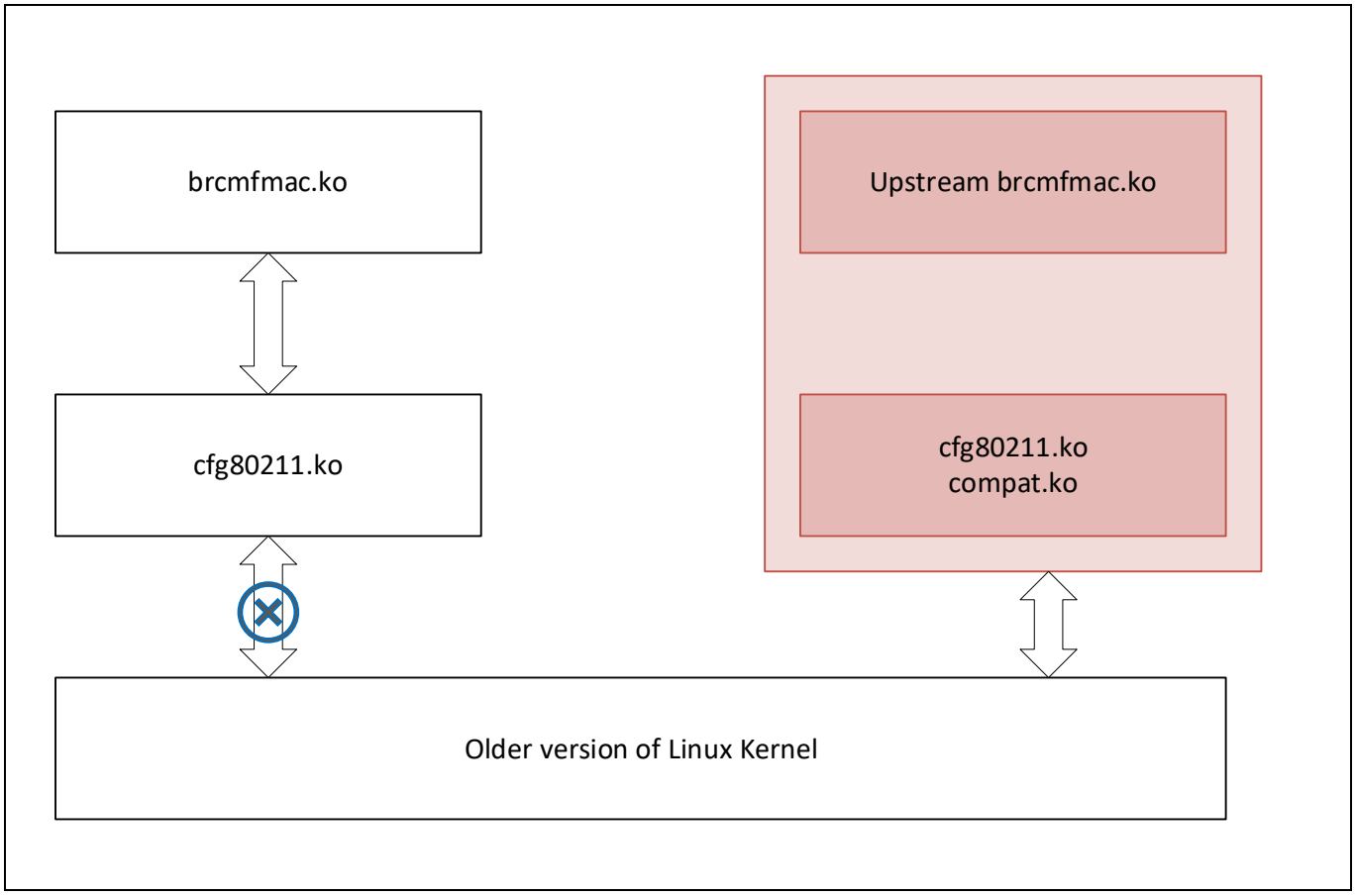

Figure 5 Backports package

英飞凌支持打包发布 backports package，即以可加载模块格式发布，而不是集成于内核，这样可以保证内核源不受污染，支持多个内核版本，消除板卡支持包（BSP）的特定依赖性，等等。可以通过以下命令将 Backports 的 **发布包**与内核集成。

```
$ 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
```

要启用调试，请修改*.config* 文件。：

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

3.3.2 交叉编译

FMAC 交叉编译，例如在 Android 主机上，需要获得适合目标平台的 Android 工具链。获取工具链的方法是 **GNU Toolchain | GNU-A Downloads - Arm Developer** 并把它放在任何目录下（例如，*$HOME/imx8mq/*）。

Table 6 标准架构的工具链前缀

架构	工具链名称
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>

按以下步骤交叉编译 FMAC 驱动器（以 i.MX8 平台为例）：

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. 按 **Backports** 所述给 FMAC 驱动打补丁
9. 调用以下命令配置 FMAC:

```
$ 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. 编译 FMAC 驱动模块:

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

11. 编译后的内核模块存储在以下目录

```
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 加载 FMAC 驱动

按以下步骤在内核中加载 FMAC 驱动:

- 确保固件，clm_blob 和 nvram 存储在 the */lib/firmware/cypress* 文件夹.
- 确保所有二进制文件前缀为 cyfmac<chip_name>-<bus_name>.bin/clm_blob/txt.
- 按以下顺序执行 insmod:

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

3.3.4 调试说明

如果内核因 Wi-Fi 驱动或固件出现崩溃，可在 FMAC 驱动中启用调试打印功能。启用调试打印功能，具体内核模块编译步骤如下：

1. 针对系统上运行的内核构建 brcmfmac 内核模块:

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

   按如下指令修改.config 内核源文件，建立新的内核镜像:

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

2. 要在运行的内核中将 brcmfmac 编译为 LKM，执行以下命令:

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

   如:

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

3. 启动 brcmfmac 调试日志:

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

4. 插入驱动模块，将所需的信息级别作为模块参数:

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

   以下为 *debug.h* 文件中定义的信息级别 (见 */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
```

如,

启用 Wi-Fi 固件控制台（环形缓冲区，旨在保存 Wi-Fi 固件内部的调试打印）日志:

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

设置控制台轮询间隔(250ms),

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

启用追踪:

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

更多 FMAC 调试相关功能信息，参见源代码*/v4.14.52-backports/drivers/net/wireless/broadcom/brcm80211/brcmfmac/debug.c.*

3.3.5 常见问题

1. 无效模块格式

出现以下问题，使用 dmesg 命令查看详细报错原因。

```
# 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 '
```

根本原因:

LKM 可能为特定内核版本编译，与当前运行内核版本不一致。另外，编译后的内核模块架构和主机平台的架构可能不同。

解决方案:

下载匹配的内核版本，重新安装正确的内核镜像。

2. 不明符号

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

使用 `dmesg` 命令查看详细报错原因。

```
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)
```

根本原因:

brcmfmac insmod 之前，部分模块缺失。

解决方案:

- 加载模块之前，检查所有模块依赖性。
- 调用 grep 命令查找未知符号，根据内核 dmesg 命令输出结果找到缺失的模块。

3. 调用“iw reg get”命令后，未显示信道，国家码显示为#n

使用 `dmesg` 命令查看详细报错原因。

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

根本原因:

/lib/firmware/cypress 文件夹中的 clm_blob 文件可能属于 cyw4373 芯片。

dmesg 中显示“无法找到对应 clm 版本”表示该补丁为 Cypress 专用， brcmfmac clm_blob 不能下载。

解决方案:

- 从英飞凌季度发布包 中复制 clm_blob 文件至/lib/firmware/cypress 文件夹。
- 参阅 **AN225347** 了解 clm_blob 工作流程后，向英飞凌申请该产品的 clm_blob，替换上一步骤中复制的通用 clm_blob。
- 使用 Cypress 指定的 brcmfmac ， 不用 Linux 本地 FMAC。

4. 使用 modprobe sdhci-pci 命令后， brcmfmac 或 btsdio 模块自动加载

根本原因:

内核在模块 ID 表中查找到设备 ID 时，自动加载模块。

解决方案:

在 `/etc/modprobe.d/blacklist.conf` 中添加模块黑名单

Linux 内核 802.11 子系统

```
"blacklist btsdio"
"blacklist brcmutil"
"blacklist brcmfmac"
```

5. 使用 insmod brcmfmac 后，USB 便携 WIFI 适配器未被激活

   根本原因:

   NVRAM 参数可能未包含在固件镜像中. 在 */lib/firmware/cypress/cyfmac4373-usb.bin* 中执行字符串命令，以检查固件镜像的尾部是否包含了 nvram 参数。

   解决方案:

   如果没有 NVRAM 参数，联系英飞凌技术支持部门，创建带有 NVRAM 参数的新固件，替换原有固件。使用英飞凌季度发布包中的固件，可避免出现这类问题。

4 用户空间 Wi-Fi 工具

4.1 wpa_supplicant

WPA_SUPPLICANT 是一种跨平台程序，支持 WEP, WPA, WPA2, WPA3 (IEEE 802.11i) 以及 WPA-EAP。它使用验证器实现密钥协商，并控制 STA 设备的漫游和关联。wpa_supplicant 的部分主要功能如下:

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

4.1.1 wpa_supplicant 依赖

- Libnl

  libnl 包是一个集合库，为基于 netlink 协议的内核接口提供 API。Netlink (nl80211) 协议是基于套接字的 IPC 通信方式，用于用户空间和内核空间的通信，或用户空间进程之间的通信。该协议是 IOCTL 之后提出的，意在提供更灵活的内核相关网络配置和监控网络接口。IOCTL 有可能污染内核，破坏系统的稳定性。Netlink socket 很简单，只需要在 *netlink.h* 中添加一个常量（协议类型），然后，内核模块和应用程序之间便可通过 socket API 通信。

- OpenSSL

  OpenSSL 是一个加密工具箱，可实现安全套接字层（SSL v2/v3）和传输层安全（TLS v1）网络协议并满足它们所要求的相关密码学标准。OpenSSL 程序是一个命令行工具，用于从 shell 中调用 OpenSSL 密码库的各种密码学功能。

- Dbus (可选)

  D-Bus 是一个消息接发总线系统，便于实现进程间通信。wpa_supplicant 有两个控制接口： dbus API 和一个目录，该目录通常是 /var/run/wpa_supplicant/ 或 /run/wpa_supplicant/，具体取决于发行版，其中包含一个为 wpa_supplicant 管理的每个 Wi-fi 接口命名的套接字。默认情况下，控制接口是不活跃的。您需要使用 -u 命令行选项来获取 dbus，并使用 -O /var/run/wpa_supplicant（或任何目录）来获取套接字。

4.1.2 编译

- 从 wpa_supplicant **官方网站**中下载最新源文件。
- 复制到 `wpa_supplicant-2.9/wpa_supplicant` 目录中，根据系统要求配置 `defconfig` 。基于 Android 的主机推荐将调试打印重路由到 logcat，为此，需取消注释 `CONFIG_ANDROID_LOG=y`。根据不同要求，还可以取消注释其他调试专用的宏。设置完成后，使用 `cp defconfig .config` 进行后续处理。
- 编译 wpa_supplicant 源文件，使用以下命令:

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

- 在特定位置安装已编译的二进制文件，使用以下命令:

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

Note: *可能会出现缺少 openssl 的头文件或版本不匹配等情况。请确保已经按照 wpa_supplicant 依赖章节安装了相应的版本。例如，在基于 ubuntu 的系统中，对于 libssl.so 或 libcrypto.so，通常只需 sudo apt-get install libssl-dev，但有时版本要求可能是 1.0.2 或 1.1，而不是默认安装的版本（默认安装有时是 1.0.0）。遇到这种情况，需要将系统的 libssl 版本升级到 wpa_supplicant 所要求的特定版本。虽然这些情况比较少见，但仍可能出现在运行旧版本内核的主机处理器。*

4.1.3 配置 wpa_supplicant

配置 wpa_supplicant 需要使用一个文本文件，该文件列出了所有接受的网络和安全策略，包括预共享密钥。该配置文件中的所有文件路径应使用完整的（绝对路径，而不是相对于工作目录的）路径，以允许修改工作目录。

例:

```
# 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"
}
```

关于 wpa_supplicant.conf 中其他相关的具体宏的细节，请访问这个**链接**.

完成配置后，启动 wpa_supplicant，连接到你的办公室或家庭接入点，继续配置，运行以下命令

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

在这行命令中,

-B 用来在后台运行 wpa_supplicant 守护进程。

-c 选项用于提供配置文件，在本例中是，example.conf。

-i 选项用于选择网络接口。

如果需要调试打印，在运行命令时添加-d 参数。更多细节，见 **man**。

Note: *如果出现以下错误：“初始化控制界面 '/var/run/wpa_supplicant' 失败”，可能有因为有另一个 wpa_supplicant 进程在运行中，或者该文件是由 wpa_supplicant 的异常终止生成的。需要先手动删除该文件，再重新启动 wpa_supplicant。命令如下:*

```
$killall wpa_supplicant
```

4.1.4 wpa_cli

Wpa_cli 是一个基于文本的前端程序，用于与 wpa_supplicant 交互。它用于查询当前状态、更改配置、触发事件和请求交互式用户输入。此外，该程序可以配置 EAPoL 状态机参数和触发事件，如重新关联和 IEEE 802.1X 注销/登录等。

wpa_cli 支持两种模式：交互模式和命令行模式。这两种模式共享相同的命令集，主要区别在于，交互模式可以访问未经请求的消息（事件信息、用户名/密码请求）。

用户空间 Wi-Fi 工具

使用交互模式：在串口终端输入 wpa_cli，不使用命令作为命令行参数，响应后，在 wpa_cli 提示符下输入命令。在命令行模式下，将同样的命令作为命令行参数输入。

4.1.4.1 配置选项

Table 7 为可行 wpa_cli 配置选项。

Table 7 wpa_cli 启动参数

命令	描述
-p path	控制套接字路径，应与 wpa_supplicant.conf 中的 ctrl_interface 匹配。默认路径是 /var/run/wpa_supplicant。
-i ifname	配置接口。默认情况下，使用在套接字路径中找到的第一个接口。
-h	显示帮助。
-v	显示版本信息。
-B	后台运行守护进程。
-a action_file	在守护进程中执行事件，具体事件内容参见 **wpa_supplicant。**
-P pid_file	提供 PID 文件的位置。
-g global_ctrl	使用一个全局控制接口连接 wpa_supplicant，而不是默认的 Unix 域套接字。
-G ping_interval	在向 wpa_supplicant 发送每个 ping 之前，等待 ping 间隔（以秒为单位）。参见 Table 8 中 Ping 命令。
command	列出所有可用命令。

4.1.4.2 WPA_CLI 命令

可在交互模式提示符下，或命令行模式中发布 **Table *8* WPA_CLI 命令** Table 8 中的命令。

Table 8 WPA_CLI 命令

命令	描述
help	显示帮助。
status	报告当前接口的 WPA/EAPOL/EAP 状态。
ifname	显示当前的接口名称。默认接口是在套接字路径中找到的第一个接口。
ping	对 wpa_supplicant 调用 ping 命令。此命令可用于测试 wpa_supplicant 守护进程的状态。
mib	报告当前接口的 MIB 变量（dot1x, dot11）。

用户空间 Wi-Fi 工具

命令	描述
help	显示帮助。
interface [ifname]	显示可用的接口，设置当前的接口，若有多个接口可用，可同时使用两个功能。
level *debug_level*	更改 wpa_supplicant 的调试级别。级别越高，生成信息越多。
license	显示 wpa_cli 的完整许可证。
logoff	IEEE 802.1X EAPOL 状态机进入 "注销 "状态。
logon	IEEE 802.1X EAPOL 状态机进入"登录 "状态。
set [settings]	设置变量。如果没有提供参数，显示已知变量及其设置。
pmksa	显示 PMKSA 缓存的内容。
reassociate	强制重新关联当前接入点。
reconfigure	强制 wpa_supplicant 重新读取其配置文件。
preauthenticate *BSSID*	强制对指定的 *BSSID* 进行预认证。
identity *network_id identity*	配置 SSID 标识。
password *network_id password*	配置 SSID 密码。
new_password *network_id password*	改变 SSID 密码。
PIN *network_id pin*	给 SSID 配置 PIN。
passphrase *network_id passphrase*	给 SSID 配置私钥口令。
bssid *network_id bssid*	为 SSID 设置首选 BSSID。
blacklist [*bssid* | *clear*]	将 BSSID 添加到黑名单中。如果调用时没有任何其他参数，显示黑名单。使用 *clear* 清除 wpa_cli 黑名单。
list_networks	列出已配置的网络。
select_network *network_id*	选择一个网络并禁用其他网络。
enable_network *network_id*	启用网络。
add_network	添加网络。
remove_network *network_id*	移除网络。
set_network *[network_id variable value]*	设置网络变量。如果运行时没有参数，显示变量列表。
get_network *network_id variable*	获取网络变量。
disconnect	断开连接，收到重新关联/重新连接命令后再进行连接。
reconnect	类似于重新关联，但只在已经断开连接的情况下生效。

用户空间 Wi-Fi 工具

命令	描述
help	显示帮助。
scan	进行新的 BSS 扫描。
scan_results	获取最新的 BSS 扫描结果。这个命令可以在用 scan 运行 BSS 扫描后调用。
bss [*idx* | *bssid*]	获取带有 "bssid "或 "idx "标识的网络的详细 BSS 扫描结果。
otp *network_id password*	给 SSID 配置一次性密码。
terminate	强制终止 wpa_supplicant。
interface_add *ifname* [*confname driver ctrl_interface driver_param bridge_name*]	使用给定参数添加新接口。
interface_remove *ifname*	移除接口
interface_list	列出可用的接口
quit	退出 wpa_cli。

Note: *如果运行任何 wap_cli 或 wpa_supplicant 命令时出现"RFkill Soft blocked "错误，可以使用以下命令：*

```
$sudo rfkill unblock all
```

4.1.4.3 典型 STA/AP 用例

STA/AP 组合方式	wpa_cli 命令
STA 接入到开放安全认证 AP	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 接入到 WPA2 认证 AP	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

STA/AP 组合方式	wpa_cli 命令
	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 接入到 WPA2_WPA3 认证 AP	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 接入到 WPA3 认证 AP	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
开放安全认证 2G/5G softAP	wpa_cli> IFNAME=wlan0 remove_net all wpa_cli> IFNAME=wlan0 add_net wpa_cli> IFNAME=wlan0 set_net 0 ssid ‘”CYP_5GAP”’

STA/AP 组合方式	wpa_cli 命令
	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
WPA2 认证 2G/5G softAP	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 (搭建 2.4 GHz AP，把频率 5180 修改为 2437) 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 无线认证和隐私基础设施 (WAPI)

WAPI 是中国国家标准，用于保障 WLANs 的安全。它的安全程度比 WEP 或 WPA 更高。关于 WAPI 相关的支持，请联系您当地的英飞凌销售办或英飞凌代表，他们可以为您提供支持 WAPI 的 wpa_supplicant 版本。

Note: *Linux 驱动需支持 nl80211/cfg80211。如果设备版本较老，不支持 netlink 驱动，需要回滚到旧 wext 驱动:*

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

4.2 Hostapd

Hostapd 是一个用户空间守护进程，用于设置 AP 和授权服务器，可精细地控制大多数参数。 它能实现 IEEE 802.11 的 AP 管理、IEEE 802.1X/WPA/WPA2/WPA3/EAP 授权、RADIUS 客户端、EAP 服务器和 RADIUS 授权服务器。通过不同配置，hostapd 可在以上任一模式中执行。Hostapd 在设计之初是一个守护进程，现也支持前端程序，如子程序 hostapd_cli。

4.2.1 Hostapd 依赖

- Libnl
- Openssl

4.2.2 Hostapd 编译

按以下步骤编译 hostapd：

1. 下载 hostapd 源代码 **软件包**并迁移到根文件夹中。
   ```
   $ cd hostapd-2.9/hostapd
   ```
2. 将现有的 defconfig 文件复制到.config。设置以下标志。
   ```
   $ cp defconfig .config
   $ vi .config
   ```
3. 设置以下参数，
   ```
   CONFIG_DRIVER_NL80211=y
   CFLAGS += -I/usr/include/libnl3
   CONFIG_LIBNL32=y
   ```

编译 hostapd 工具，使用下列 make 命令。

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

安装 hostapd 工具，发出以下命令，并把它放在/usr/sbin 目录下（主流选择）。

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

4.2.3 配置文件

Hostapd 使用文本文件进行配置，该文件设置了接入点（AP）的安全协议（802.11i、802.1X 等）、国家码、口令等。创建 *hostapd.conf* 文件，如下：

```
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 用于启用调试打印，可在平台建立结束后删除。*

4.2.3.1 Hostapd 使用

Hostpad 使用	设置
设置 WPA2 认证的 softAP	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
设置 WPA3 认证的 softAP	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 配置

DHCP 守护进程有多种类型可选；如 udhcp、dhcp、dnsmasq 等。其中大部分用作 dhcp 服务器，一部分还可作 dns 服务器。Udhcp 等默认安装在核心操作系统，也可以手动安装 dnsmasq 等软件包。下面以 dnsmasq 为例，演示如何在 AP 接口上设置 dhcp 和 dns 服务器。对 *dhcpd.conf* 文件做如下修改：

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

默认 dnsmasq 配置文件提供了配置 dhcp 服务器的多种选项。因此，与其编辑默认的.*conf* 文件，不如备份该文件，创建并使用一个新的.config 文件：

```
$ 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
```

通过上述代码，将 192.168.0.11 和 192.168.0.30 之间的 IP 地址分配给 wlan0 接口。再对网络路由做如下修正，便可以启动 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
```

即使选择使用现成的 dhcp 服务器守护进程，也要参阅相应的文档（比如路径修改等）来设置 dhcp 和 dns 服务器。

4.3 IW

IW 是基于 nl80211 的用户空间命令行工具，用于配置无线设备。它支持 Wi-Fi 设备所使用的两种 Wi-Fi 驱动。旧 iwconfig 工具已被淘汰，强烈建议改用 iw 和 nl80211。

4.3.1 依赖

iw 的基本要求是要有 libnl。要使用 pkg-config，应该满足以下的依赖:

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

4.3.2 编译

可以使用系统的软件包管理工具来安装上述软件包，然后继续安装 iw。如果选择从源代码编译，可以在这个 **链接**获取已发布的 tarball.

4.3.3 典型用途

Iw 使用起来非常简单。 `iw list` 命令可列出系统中的无线设备的信息。该命令还会显示你的 Wi-Fi 设备所支持的命令列表。在此基础上，你可以使用 iw help <cmd_name>来发布适合你使用情况的命令。

4.3.3.1 具有 WPA/WPA2/WPA3 认证的 SoftAP

按以下命令设置一个安全的 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
```

对于与 AP 相关的安全配置，可以使用 hostapd.conf，其中接口参数为 softap，具体命令参阅**配置文件**，按所需的安全级别（WPAx）设置 AP。

4.3.3.2 STA 接入到开放/ wep 认证 AP

iw 只能处理 STA 接入到开放认证或 WEP 认证 AP 的连接过程。对于 WPAx 认证 AP，建议使用 wpa_supplicant 代替。

开放认证:

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

如果有多个具有相同 SSID 的 AP，但想与频率为 2432（频道 5）的 AP 连接，请运行以下命令:

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

WEP 已被淘汰，取而代之的是 802. 11i 中更强大的安全措施。如果你有一个支持 WEP 的 AP，并且想用 iw 连接，请使用以下命令:

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

5 附录

5.1 如何使用默认/yocto 版本

Yocto 工具广泛用于开发嵌入式产品的定制化 Linux 系统。Yocto project 提供了一系列灵活的工具以及开源社区，全世界的嵌入式开发者可以在这里分享技术、软件、配置和经验，帮助创建定制 Linux 镜像和嵌入式产品，可应用于物联网设备等任何需要定制 Linux 操作系统的设备。 本版本支持在 Yocto 项目中启用 Wi-Fi 连接软件，便于快速开始连接软件。使用 Yocto 版本，请按下列步骤构建程序：

1. 提取构建脚本 tarball

```
$ tar zxvf cypress-yocto-scripts-v5.4.18-2020_0925.tar.gz
```

2. 创建一个工作目录。例如：cypress-imx-bsp

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

3. 将以下数据复制到工作目录中。

```
 * 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. 运行 *setup_host_env.sh* 脚本进行首次构建，为主机设置构建环境。

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

5. 在工作目录中运行 *build_yocto_wireless.sh* 脚本，生成 Cypress 定制的 Yocto 镜像。

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

如果脚本无法按用户权限运行，请使用：

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

5.2 如何使用非 yocto 版本

本季度发布包中的补丁文件基于最新的 Linux 稳定版内核（v5.4.18），所以旧的内核需要使用 backports。下面的例子展示了如何在旧版本内核或 Linux 稳定版内核（v5.4.18）中使用该包。如果使用带有 backports 的旧版本内核，需要单独构建 Linux 内核镜像和 cypress wifi 驱动模块。

构建内核镜像的方法参见**设备树二进制文件（DTB）**。构建 cypress Wi-Fi 驱动的 Backports 模块，参见 **Backports**。

5.3 固件升级

通常情况下，每季度发布的版本都包含最新固件。要升级到最新的固件，请运行以下命令：

```
$ modprobe brcmfmac.ko
```

参考资料

[1] **Device Tree Structure**

[2] **Linux Wireless**

[3] **Linux Device Drivers**

[4] **Linux MMC Subsystem**

[5] **Linux PCI Bus Subsytem**

[6] **Linux USB Subsytem**

修改历史

文件版本	发布日期	变更内容
**	2021-04-26	首发
	2023-03-08	英到中翻译

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

Edition 2023-03-08
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. **

IMPORTANT NOTICE
The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

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

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

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

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