Wi-Fi ソフトウェア ユーザーガイド

本書について

適用範囲と目的

本書では、Linux 802.11 エコシステムの構成ブロックの概要を説明します。本書は、選択したホストで Wi-Fi モジュールを便利に使用し、アプリケーションに基づいて設定するのに役立ちます。

対象者

本書は、インフィニオン Wi-Fi ソリューションと任意の Linux ホストを使用している方を対象とします。Linux カーネル ネットワークの経験があるか、Linux ホスト プロセッサのブート フローに関する知識があることが推奨されます。

Note: *本書で使用する略語と頭字語については、Wi-Fi glossary を参照してください。*

目次

本書について 1
目次 2
1 Wi-Fi ソフトウェアの紹介 4
1.1 Wi-Fi ソフトウェア アーキテクチャのブロック図 4
2 プラットフォーム インターフェース, ブート プロセス, およびデバイス ツリーブロブ 5
2.1 ハードウェア接続 5
2.2 カーネル設定 6
2.2.1 デバイス ツリーブロブ (DTB) 設定 7
2.3 WLAN ホスト インターフェース 8
2.3.1 マルチメディア カード (MMC) 8
2.3.2 PCIe 8
2.3.3 USB 8
3 Linux カーネル 802.11 サブシステム 9
3.1 nl80211 9
3.2 cfg80211 11
3.3 FMAC の立ち上げ 11
3.3.1 カーネル 6.1+ のサポート 13
3.3.2 カーネル 6.6+ のサポート 16
3.3.3 Backports 17
3.3.4 クロスコンパイル 18
3.3.5 FMAC ドライバのロード 19
3.3.6 デバッグに関する注意事項 19
3.3.7 よくある問題 21
4 ユーザースペース Wi-Fi ユーティリティ 23
4.1 wpa_supplicant 23
4.1.1 wpa_supplicant の依存関係 23
4.1.2 wpa_supplicant のコンパイル 24
4.1.3 wpa_supplicant の設定 24
4.1.4 wpa_cli 26
4.1.4.1 設定オプション 26
4.1.4.2 wpa_cli コマンド 27
4.1.4.3 STA/AP の一般的な使用例 29
4.1.4.4 ワイヤレス認証およびプライバシーインフラストラクチャ (WAPI) 31
4.2 hostapd 31
4.2.1 hostapd の依存関係 31
4.2.2 hostapd のコンパイル 31
4.2.3 Conf files 32
4.2.3.1 hostapd の使用法 33
4.2.4 DHCP 設定 33
4.3 iw 34
4.3.1 依存関係 34
4.3.2 iw のコンパイル 34
4.3.3 通常の使用法 35
4.3.3.1 WPA/WPA2/WPA3 セキュリティを備えた SoftAP 35
4.3.3.2 オープン/WEP セキュリティを備えた AP に接続する STA 35
5 付録 36
5.1 デフォルト/Yocto リリースにコネクティビティを追加するためのチェックリスト 36

目次

5.2 Yocto リリース以外にコネクティビティを追加するためのチェックリスト 36
5.3 ファームウェアのアップグレード 37
5.4 i.MX8 用ツールチェーン 37
5.4.1 Linux ホストのセットアップ 37
5.4.2 必要なパッケージ 37
5.4.3 Yocto レシピのダウンロード 38
5.4.4 ビルドの初期化 38
5.4.5 カスタム i.MX ツールチェーン 39
5.4.6 イメージのビルド 39
参考資料 40
改訂履歴 41
免責事項 42

1 Wi-Fi ソフトウェアの紹介

Wi-Fi ソフトウェアは、Wi-Fi デバイスを動作可能にするために必要な基本的なコンポーネントを提供します。つまり、802.11 フレームを無線で送受信します。本書は、Linux カーネル ネットワーキング サブシステムと、wpa_supplicant, hostapd, および iw のようなユーザー空間からの WLAN の設定に関連するコンポーネントを理解するのに役立ちます。本書では、ユーザー空間とカーネル空間の機能、および Wi-Fi デバイスで使用するために作成または設定されたデバイスドライバについて説明します。

1.1 Wi-Fi ソフトウェア アーキテクチャのブロック図

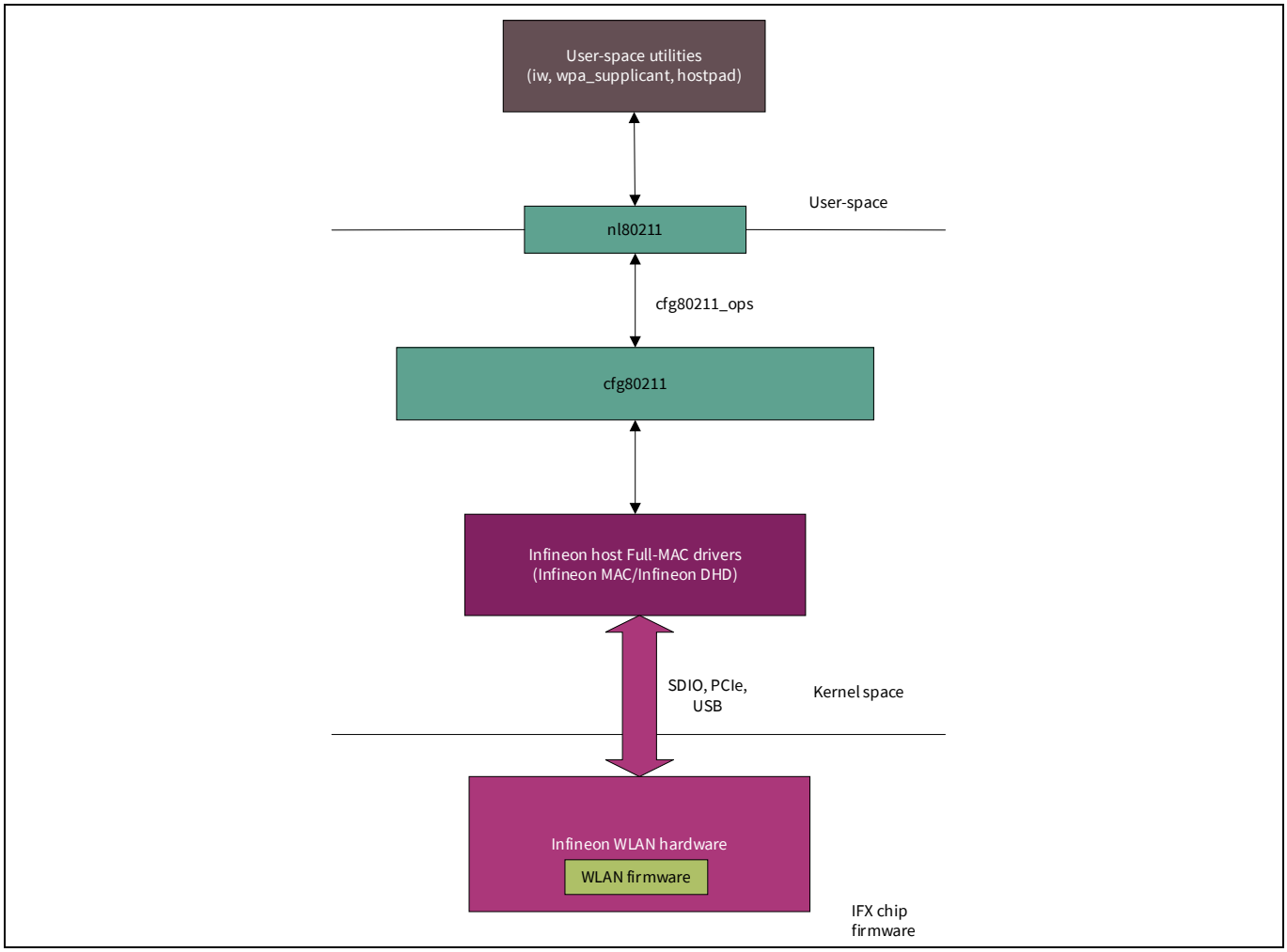


Figure 1 Linux 802.11 アーキテクチャ – 要約

Wi-Fi ソフトウェア アーキテクチャのアプリケーション (*iw*, *wpa_supplicant*, および *hostapd* が属する最上位のレイヤー) とファームウェアレベル (インフィニオン Wi-Fi チップに組み込まれている) の間には、送信 (TX), 受信 (RX), およびイベントパスがあります。中間層は、TX/RX またはイベントのいずれかフローのタイプごとに条件を使用します。それに基づいて、ホストとデバイス間のフロー制御またはイベント キューメカニズムもデバイスドライバに実装されます (インフィニオンリリーストレインにてパッケージ化リリースされる)。ハードウェア層は、バス ハードウェアの一部とともにコア 802.11 動作を実装します。このレイヤーはインフィニオンの Wi-Fi ファームウェア内に実装されるため、個別のデバイスドライバを作成する必要がなくなり、市場投入までの時間が大幅に短縮されます。

2 プラットフォーム インターフェース, ブート プロセス, およびデバイス ツリーブロブ

2.1 ハードウェア接続

ホストプロセッサとターゲット Wi-Fi 無線との間の接続は、次のように分類できます。

- バス接続: ホスト プロセッサとターゲット Wi-Fi 無線は、双方向バスを介して接続されます。これにより、次の接続が提供されます。
  - SDIO: D0, D1, D2, および D3
  - PCIe: TDN, TDP, RDN, および RDP
  - USB: DP, DN など

Bluetooth®の場合、トランスポートは通常、SDIO または UART のいずれかになります (接続は RX, TX, RTS, CTS などになります)。

Table 1 ホストプロセッサと Wi-Fi/Bluetooth®無線間のトランスポートバスの組み合わせ

接続カテゴリ	バス	各バスに対応するピン
ホスト <--> Wi-Fi 接続	SDIO	D0-D3
ホスト <--> Wi-Fi 接続	PCIe	TDN, TDP, RDN, および RDP
ホスト <--> Wi-Fi/Bluetooth®接続	USB	DP, DN
ホスト <---> Bluetooth®接続	UART	RTS, CTS, TX, RX

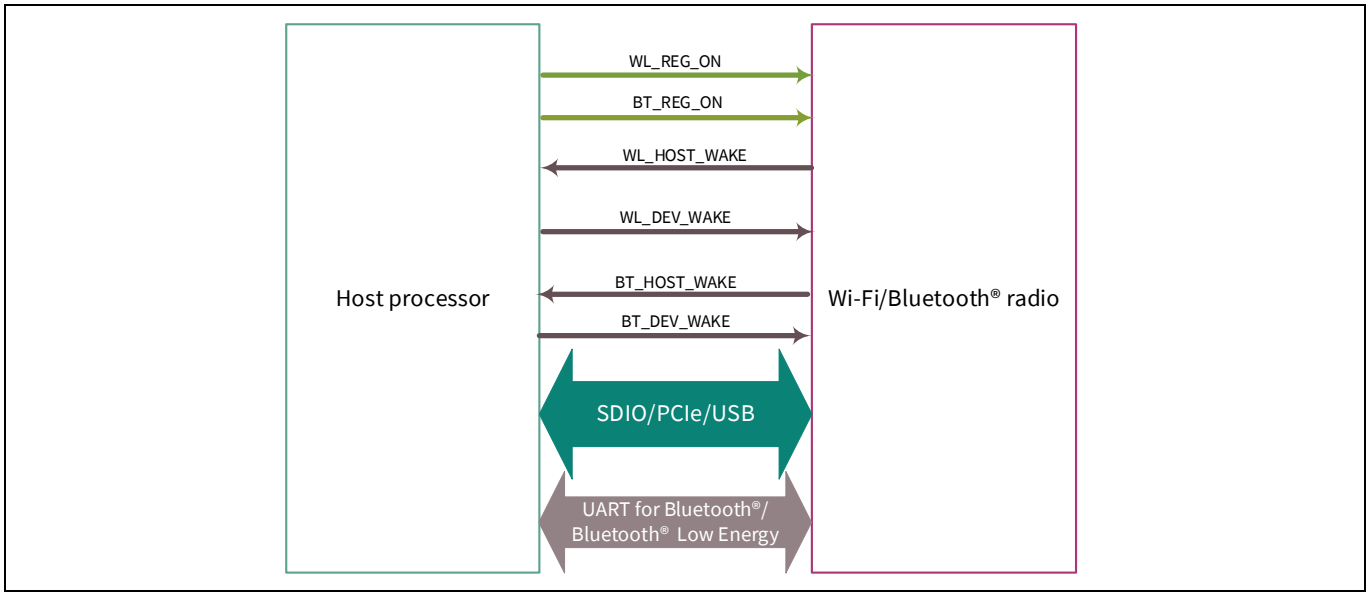


Figure 2 ホストプロセッサから Wi-Fi/Bluetooth®/Bluetooth® Low Energy 無線接続ブロック

- **電源関連の接続**: Wi-Fi モジュールの電源を入れるために必要な GPIO が含まれています。WL_REG_ON と BT_REG_ON はこのカテゴリに属します。
- **シグナリング接続**: ホスト プロセッサまたは Wi-Fi モジュールをウェイクアップするために必要な GPIO が含まれます。担当するピンは、WL_HOST_WAKE, WL_DEV_WAKE, BT_HOST_WAKE, および BT_DEV_WAKE です。

2.2 カーネル設定

カーネルのコンパイルについては、ベンダーの指示に従って、ソースとツールチェーンを設定します。これらは、ベンダーの配布媒体で見つけることができます。

1. ターゲットのアーキテクチャ (ARM64, Arm®, x86, MIPS など) に基づいて、*arch/arm/configs* で利用可能なデフォルト設定を選択します。
2. 次のコマンドを発行して、カーネルを *.config* ファイルで設定します。

```
$ make defconfig
```

3. *.config* ファイルを編集し、*cfg80211* をモジュールとしてビルドします。

```
# CONFIG_CFG80211=m
```

4. 従来の Dongle Host Driver (DHD) をドライバとして使用していない場合は、次の設定を変更します。

```
# CONFIG_BCMDHD=n
```

5. カーネル バージョン 5.4.18 以降では、以下の *cfg80211 regdb* 設定を使用不可にします。

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

6. さらに、*.config* ファイルで次の設定を有効にします。

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

7. ターゲットホストの Linux カーネルイメージをビルドします。次のコマンドは、i.MX 用です。

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

カーネルイメージが *arch/arm/boot/zImage* ディレクトリで利用可能になりました。

四半期ごとのトレインリリースから入手可能なインフィニオンのソフトウェア作成物は、次の 2 つの主要なエコシステムのさまざまなプラットフォーム (MMC/SDIO: NXP i.MX6, NXP i.MX8, PCIe: i.MX8, Intel NUC) でサポートされ、検証されています。

- 最新の Google Android オープンソース プラットフォーム (AOSP) バージョン (Android TV, Wear OS などの派生サポート付き)
- 最新の長期 Linux カーネルリリース

インフィニオンのカーネルサポートポリシーでは、FMAC ドライバベースのチップセットに Backports Project を使用します。これにより、古いカーネルで最新のソフトウェアを実行できます。カーネル バージョンが最新の LTS でない場合は、開発環境で Backports パッケージを実行して、設計で最新の接続ソフトウェア (ファームウェアおよびドライバ) をイネーブルにします。次に、変更されたデバイス ツリー ブロブ (DTB) とカーネル イメージを使用してホスト プロセッサをフラッシュし、(バックポートされた) カーネル モジュール (KM) をロードします。

2.2.1 デバイス ツリーブロブ (DTB) 設定

デバイス ツリーは、I2C デバイスや SPI デバイスなど、本質的に検出できないハードウェアの platform_data または pdata を記述する方法を提供します。DTB は通常、*.dts* ソース ファイルや *.dtsi* インクルード ファイルなど、人間が判読できる形式で作成および管理されます。*.dts* ファイルはボードレベルの定義を提供し、*.dtsi* は SoC レベルの定義を提供します。デバイスツリーのソースファイルは、デバイス ツリーコンパイラ (DTC) を使用してコンパイルされます。ソースファイルは *<kernel_base>/scripts/dtc* フォルダにあります。DTC は、フラット化されたデバイス ツリー (FDT) とも呼ばれる*.dtb* ファイルを生成します。

Linux は、デバイス ツリー データを使用して、システム内のデバイスを検索して登録します。FDT は、ブートの非常に初期のフェーズでは生の形式でアクセスされますが、ブートの後のフェーズやシステムのブート完了後には、より効率的にアクセスするために "拡張デバイスツリー" (EDT) と呼ばれるカーネル内部データ構造に拡張されます。通常、デバイスツリーには、デバイスが使用する必要がある I/O ポートと割込みラインに関する情報が含まれます。各デバイスノード (デバイスのツリー内のプラットフォームデバイスを表す) には、カーネルがデバイスツリーをスキャンするときにデバイスを識別するために使用される name プロパティがあります。ドライバでは、"compatible" プロパティは、カーネルがデバイス ツリーで検索する必要がある名前フィールドを指定します。カーネルが名前を見つけると、対応するデバイスがインスタンス化され、ドライバと照合されます。

Wi-Fi Linux ドライバ パッケージ には、i.MX6SX および i.MX6UL デバイスツリーブロブを含むデバイスツリーフォルダがあります。ホストプラットフォームがデバイスツリーパッケージにない場合は、*arch/arm/boot/dts/* にある既存のソースファイル (*.dts* および *.dtsi* ファイル) を参照し、ファイルをターゲットホストプラットフォームに移植します。*.dts* ファイルと *.dtsi* ファイルの各フィールドでは、次の設定を使用できます。

- **wlreg_on**: このピン (WL_REG_ON) は、Wi-Fi デバイスの電源投入を担当します。ピンはアクティブ HIGH に設定する必要があります。電圧要件については、対応するチップのデータシートを参照してください。
- **In-band/out-of-band**: Wi-Fi デバイスが SDIO 経由でホスト プロセッサに接続されている場合、Wi-Fi デバイスからホストに割込みをルーティングするには 2 つの方法があります。
  - 割込みのインバンド メカニズムでは、SDIO DATA1 ラインを使用して割込みを通知します。
  - アウトオブバンドメカニズムには、専用の GPIO ピンが必要です。ピンの多重化が処理され、ピンが GPIO としてのみ機能していることを確認します。

アプリケーションに応じて、in-band または out-of-band (OOB) を選択できます。最適な低電力数値を実現するには、OOB シグナリング方式を使用します。これは、このモードでは、パケットが受信されたときに WLAN_HOST_WAKE ラインで割込みがトリガーされない限り、SDIO バスがサスペンドモードになるためです。ホストプロセッサにスペアの GPIO がない場合は、DATA1 ラインを割込みとして再利用する in-band 割込み方式を使用します。これにより、電力負担が増えるバスの一時停止を防ぎます。

NXP Semiconductor の i.MX プラットフォーム用の次の実装例は、次の場所にあります。

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

- *imx6ul-evk-btwifi-oob.dtsi* (OOB 割込みピンの割り当てと設定用)
- *imx6ul-evk-btwifi.dtsi* (WL_REG_ON ピン関連の設定用)

フル MAC (FMAC) の場合、compatible フィールドは `of_device_is_compatible(np, "brcm,bcm4329-fmac"))` に拡張されます。Linux デバイス ツリーの詳細については、Linux Device Tree - FMAC の Web ページを参照してください。

2.3 WLAN ホスト インターフェース

ここでは、Wi-Fi デバイスを選択したホストプロセッサに接続するために使用できるインターフェースオプションについて説明します。

2.3.1 マルチメディア カード (MMC)

マルチメディアカード (MMC) は、セキュアデジタル (SD) 規格が発展した低コストのデータストレージメディアです。I/O カードは、高速シリアルデータ入出力ラインと低消費電力を兼ね備えているため、IoT デバイスなどのバッテリー駆動の電子機器に適しています。Wi-Fi ソリューションは、デバイス ドライバに事前にパッケージ化されています。したがって、SDIO ホストのドライバの実装を処理するだけで済みます。ホスト内の SDHC インターフェースと MMC インターフェースは、Wi-Fi チップと通信するために SDIO ホスト インターフェースに慎重にマッピングする必要があります。Table 2 に、MMC/SDIO インターフェースをサポートする Wi-Fi チップセットを示します。

Table 2 ホストインターフェースとして SDIO を搭載した Wi-Fi デバイス

アンテナ設定	802.11 プロトコル	インフィニオン Wi-Fi デバイス
1x1 SISO	802.11n	AIROC™ CYW43439
1x1 SISO	802.11ac	AIROC™ CYW43012
1x1 SISO	802.11ac	AIROC™ CYW43022
1x1 SISO	802.11ac	AIROC™ CYW4373
1x1 SISO	802.11ac	AIROC™ CYW43455
1x1 SISO	802.11ax	AIROC™ CYW55500
2×2 MIMO	802.11ac	AIROC™ CYW5459x
2×2 MIMO	802.11ax	AIROC™ CYW5557x

2.3.2 PCIe

PCIe (Peripheral Component Interconnect Express) は、SSD, Wi-Fi, Ethernet などのインターフェースとして一般的に使用される高速シリアルバスです。バージョンまたはレーン関連の仕様については、対応するチップデータシートを参照してください。Table 3 に、PCIe インターフェースをサポートするチップ マトリックスを示します。

Table 3 PCIe をホストインターフェースとする Wi-Fi デバイス

アンテナ設定	802.11 プロトコル	インフィニオン Wi-Fi デバイス
1x1 SISO	802.11ac	AIROC™ CYW43455
2×2 MIMO	802.11ac	AIROC™ CYW5459x
2×2 MIMO	802.11ax	AIROC™ CYW5557x

2.3.3 USB

このクラスのデバイスの USB 機能は、ストレージメディアから Wi-Fi、またはイーサネットドングルまで多岐にわたります。Wi-Fi ポートフォリオでは、AIROC™ CYW4373 (1×1 802.11ac) が USB インターフェースをサポートしています。

無線 LAN 機器の詳細については、AIROC™ Wi-Fi + Bluetooth®コンボのページをご覧ください。

3 Linux カーネル 802.11 サブシステム

3.1 nl80211

ネットリンク (nl80211) プロトコルは、ユーザー空間とカーネル空間の間、またはユーザー空間プロセス間の通信に使用されるソケットベースの IPC メカニズムです。これは、主にカーネル関連のネットワーク設定を提供し、ネットワークインターフェースを監視するために、ioctls のより柔軟な後継として設計されました。Table 4 は、従来の ioctl ベースのシステム コールと netlink を比較したものです。

Table 4 syscall と netlink の比較

プロパティ	Netlink sockets	Syscalls
通信を開始できるユーザー	ユーザー空間アプリケーションとカーネルモジュール	ユーザー アプリケーション
マルチキャスト?	有	無
ポーリングが必要ですか?	無	有
非同期?	Yes (メッセージキューを提供します)	無

netlink ソケットの操作は単純で、カーネルネットリンクソケットとの通信を処理するソケットをユーザースペースで開いて登録します。netlink プロトコルは、ユーザー空間とカーネル間の他の通信方法に比べていくつかの利点があります。例えば、netlink ソケットを操作するときにポーリングは必要ありません。ユーザー空間のアプリケーションは、ソケットを開いてから `recvmsg()` を呼び出し、カーネルからメッセージが送られてこない場合 (例えば、iproute2 パッケージ (*lib/libnetlink.c*) の `rtnl_listen()` メソッド) はブロッキング状態になります 。

別の利点は、netlink ソケットがマルチキャスト伝送をサポートしていることです。netlink ソケットは、`socket()`システムコールを使用してユーザースペースから作成します。netlink ソケットは、`SOCK_RAW` ソケットまたは `SOCK_DGRAM` ソケットのいずれかです。netlink ソケットは、カーネルまたはユーザー空間に作成できます。カーネルネットリンクソケットは `netlink_kernel_create()`メソッドによって作成され、ユーザー空間ネットリンクソケットは `socket()`システムコールによって作成されます。ユーザースペースまたはカーネルからネットリンクソケットを作成すると、`netlink_sock` オブジェクトが作成されます。

ソケットがユーザー空間から作成されると、`netlink_create()`メソッドによって処理されます。ソケットがカーネルに作成されると、`__netlink_kernel_create()`によって処理されます。このメソッドは `NETLINK_KERNEL_SOCKET` フラグを設定します。最終的に、両方のメソッドが`_netlink_create()`を呼び出して、ソケットを一般的に割り当て (`sk_alloc()`メソッドを呼び出すことによって) ソケットを初期化します。

Figure 3 に、カーネル空間とユーザー空間で netlink ソケットがどのように作成されるかを示します。

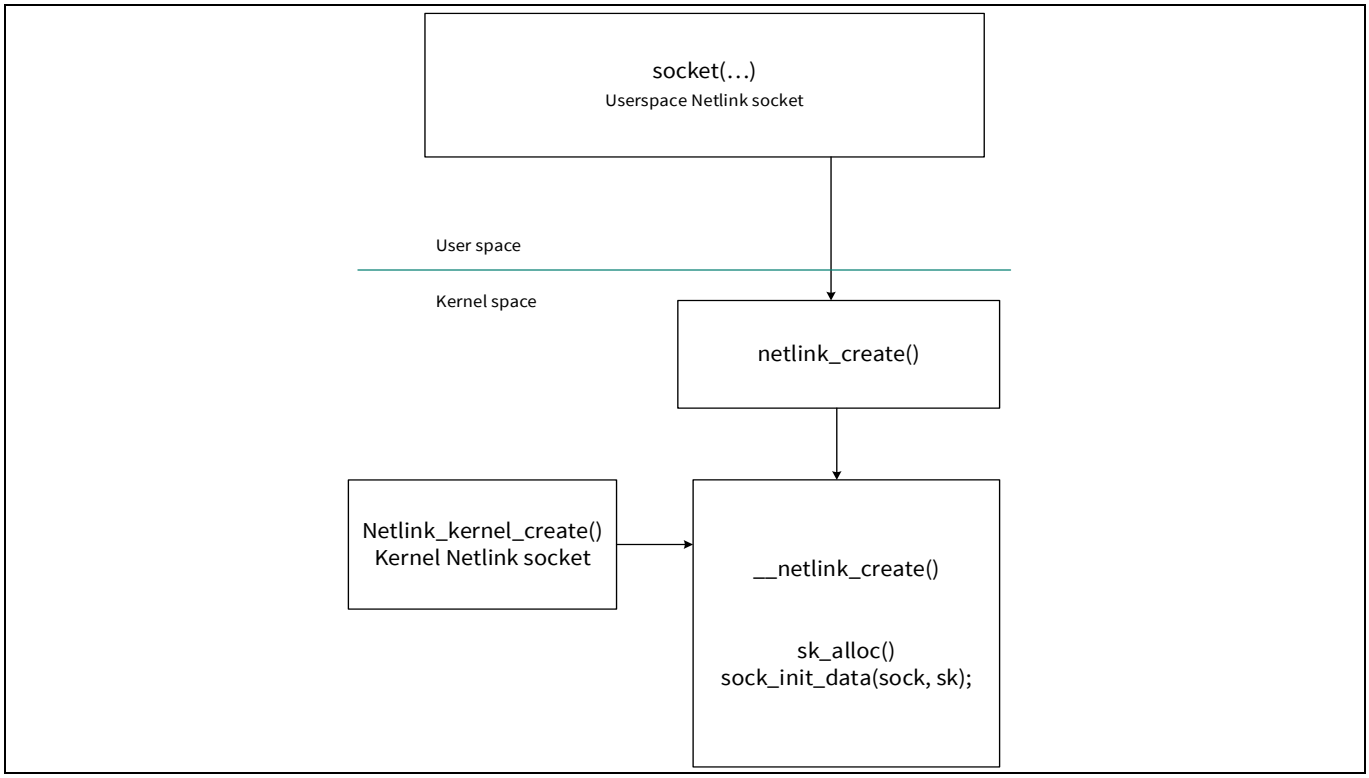


Figure 3　Netlink のプロセスフロー

libnl パッケージは、netlink プロトコルベースの Linux カーネルインターフェースに API を提供するライブラリのコレクションです。*iproute2* パッケージは libnl ライブラリを使用します。コアライブラリ (*libnl*) の他に、このパッケージには汎用 netlink ファミリー (*libnl-genl*), ルーティングファミリー (*libnl-route*), および netfilter ファミリー (*libnl-nf*) のサポートが含まれます。

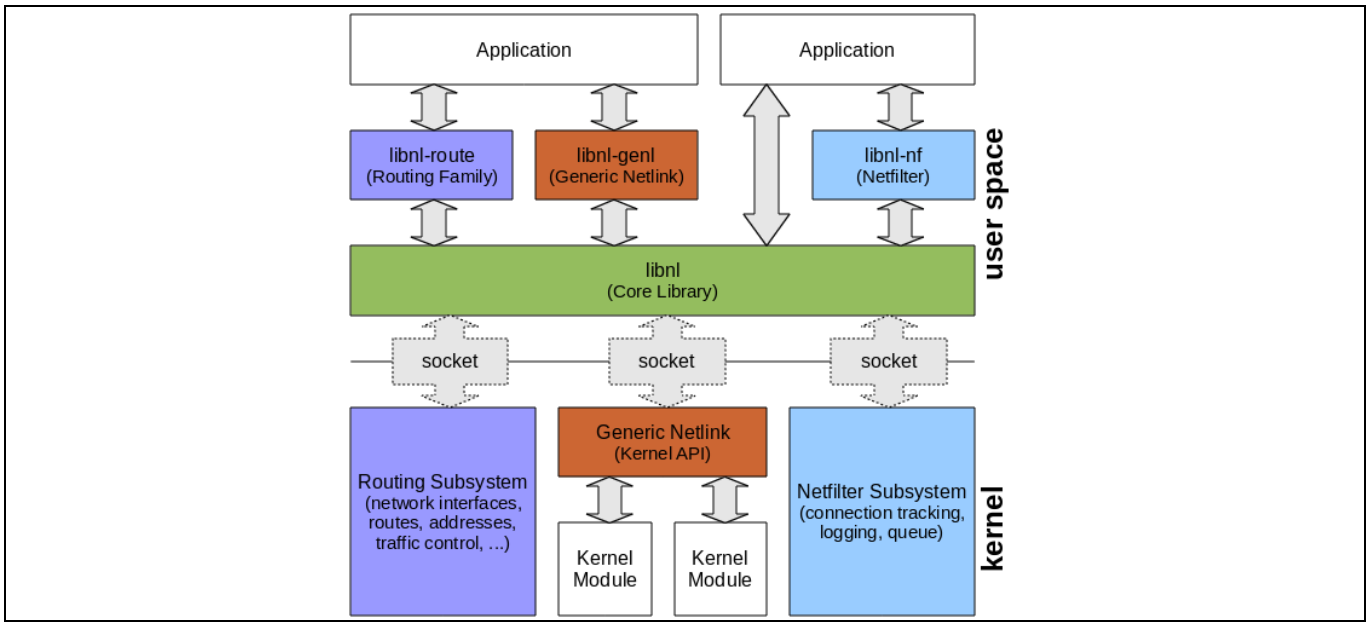


Figure 4　Netlink ファミリー

画像ソースは、Netlink Protocol Library Suite (libnl) にあります。

3.2 cfg80211

CFG80211 は、主に Linux の 802.11 デバイスの API の設定を担当しています。これは、nl80211 を介してカーネルとユーザー空間の間の管理インターフェースを提供します。下位互換性のために、cfg80211 はユーザー空間にワイヤレス拡張 (WEXT) も提供しますが、ドライバ レイヤーから完全に抽象化します。さらに、cfg80211 には、規制電力の制約とスペクトルに関する考慮事項を確立するのに役立つコードが含まれています。

ドライバで cfg80211 を使用するには、ハードウェア デバイスを cfg80211 に登録する必要があります。これは、ここで説明するいくつかのハードウェア機能構造体を通じて行われます。各デバイスの基本的な構造は「wiphy」であり、各インスタンスはシステムに接続された物理的なワイヤレスデバイスを表します。

各 wiphy には、0 個, 1 個, または多数の仮想インターフェースを関連付けることができます。関連付けられている仮想インターフェースは、ネットワーク インターフェースの `ieee80211_ptr` ポインターを、インターフェースのワイヤレス部分を記述する `struct wireless_dev` にポインティングすることで識別する必要があります。通常、この構造体はネットワークインターフェースのプライベートデータ領域に埋め込まれています。ドライバは必要に応じて、仮想インターフェースをその場で作成または破棄できますが、少なくとも 1 つの仮想インターフェースまたはいくつかの仮想インターフェースを作成する機能がない場合、ワイヤレス デバイスは役に立ちません。

各 wiphy 構造体には、デバイスの機能情報が含まれており、ドライバが提供するさまざまな操作へのポインターがあります。Wi-Fi ドライバは、cfg80211 操作コールバックを提供し、デバイスの機能を正確に示すために wiphy 構造体に入力する必要があります。

インフィニオンのドライバ リリース パッケージでは、cfg80211 に関連するすべての操作がすでに処理されているため、cfg80211 に関連する複雑な操作を省いて、アプリケーション開発を続けることができます。追加の cfg80211_ops を使用してドライバをカスタマイズする場合は、*cfg80211.h* で各操作の詳細を参照してください。

3.3 FMAC の立ち上げ

FMAC は、MAC サブレイヤー管理エンティティ (MLME) がハードウェアで管理されるタイプのワイヤレスカードを表します。Wi-Fi ポートフォリオのすべてのチップは、このカテゴリに分類されます。FMAC ドライバは、もともと Linux Kernel 2.6+で導入されました。このドライバは Linux カーネルの一部であるため、誰でも変更をアップストリームできます。FMAC ドライバの主な属性を次に示します。

- SDIO, PCIe, および USB インターフェースを 1 つのバイナリでサポート
- SoftAP, P2P (Peer-to-Peer), TDLS (Tunneled Direct Link Setup) などの主要な機能をサポートします (各チップに関連する仕様については、インフィニオン FMAC ドライバリリースパッケージに付属の README ファイルを参照してください)。
- FMAC はカーネルの一部であるため、異なるカーネルバージョンのサポートが容易になります。

インフィニオン FMAC をビルドするには、次の手順を実行してください。

1. サポートされている最新の Linux カーネルソースをインフィニオン GitHub からダウンロードします。

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

2. デフォルトのカーネル *.config* を変更し、次のオプションを有効にしてから、カーネルイメージをコンパイルします。

```
#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. カーネル バージョン 5.4 以上の場合は、次の追加オプションを追加します。

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

4. Wi-Fi チップのファームウェアには、次の 2 つのオプションがあります。
   a) */lib/firmware/cypress* ディレクトリにあるオリジナルのファームウェアファイルを使用します。
   b) インフィニオンの四半期ごとのリリースを通じて入手可能な最新のファームウェアにアップグレードするには、次のコマンドを使用します。

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

5. 新しくコンパイルされたカーネルイメージでデバイスを再起動し、最新の Wi-Fi 機能を使用します。
6. 再起動後、`dmesg` コマンドでチップ ID と、ファームウェアのバージョン, ファームウェア ID, コンパイル日などの追加情報を確認します。
7. 次のコマンドを使用して、*brcmfmac* が必要とするすべての依存モジュールを挿入します。

```
modprobe cfg80211
```

8. ドライバをロードする際に、Table 5 にリストされているパラメーターを引数としてローダブルカーネルモジュール (LKM) に渡します。例えば、以下を参照してください。

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

Table 5　FMAC モジュール パラメーター

モジュールパラメーター名	機能	モジュールパラメータータイプ
p2pon	従来の P2P 管理機能を有効にします	int
txglomsz	最大 Tx パケット チェーン サイズ [SDIO]	int
debug	デバッグ出力のレベル。デバッグに関する注意事項を参照してください。	int
feature_disable	機能を無効にします	int
alternative_fw_path	代替ファームウェアパス。ファームウェアが */lib/firmware/cypress* 以外の別のパスに存在する場合。	string
fcmode	ファームウェア制御のフロー制御のモード	int
roamoff	内部ローミングエンジンを使用しない	int
Iapp	廃止されたアクセスポイント間プロトコルの部分的なサポートを有効にします	int
ignore_probe_fail	デバッグのためのプローブは常に成功します	int

3.3.1　カーネル 6.1+ のサポート

カーネルのフォワードポーティングには、新しいバージョンで動作するようにカーネルの FMAC ソースコードを更新することが含まれます。これには、カーネルとオペレーティングシステムのインターフェースを変更したり、新しいハードウェアまたはソフトウェア構成で動作するようにドライバやその他のカーネルモジュールを更新したりすることが含まれる場合があります。

新しいバージョンのカーネルで FMAC をクロスコンパイルするには、ビルドシステムにプラットフォームに適したツールチェーンが必要です (通常は BSP で提供されます)。前提条件は、FMAC パッチをダウンロードすることです (インフィニオンコミュニティから FMAC ドライバをダウンロードするか、最寄りのインフィニオン FAE/営業担当者に連絡してリリースパッケージを入手してください)。

新しいカーネルバージョンでのセットアップを有効にするには、次の手順を実行します。

1. 使用するプラットフォームに固有の 6.1 + カーネル ソース コードをダウンロードしてください。 例えば、i.MX8 では、次のようになります。

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

2. プラットフォームのカーネルバージョンに固有のツールチェーンを設定します。例えば、i.MX8 では、i.MX8 用ツールチェーンを取得する手順に従ってください。
3. FMAC パッチをダウンロードして適用してください。
4. defconfig に FMAC に必要なフラグを追加し、カーネルのアップグレードによって発生する API 互換性の問題を修正してください。例えば、i.MX8 の場合は以下です。

```
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. 次のコマンドを使用して、cfg80211 を使用不可にします。

```
#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. プラットフォームに固有のビルドソースを設定します。例えば、i.MX8 の場合は以下です。

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

7. リンカー エラーを回避するには、次のコマンドを実行します。

```
$ unset LDFLAGS
```

8. プラットフォームのカーネルをビルドします。例えば、i.MX8 の場合は以下です。

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

ビルドされたカーネルは *arch/arm64/boot/Image* で利用可能です。出力モジュールは、次のファイルにあります。

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

Note: *コンパイル中の問題については、コミュニティを参照してください。*

3.3.2 カーネル 6.6+ のサポート

カーネルをインフィニオンがサポートするバージョンに更新します。Fedora リリース カーネルをダウンロードし、特定のバージョン v6.6.15 (SHA ID: 9670bf06) にチェックアウトします。これは、次のテスト環境を使用します。

- ハードウェア機器: INTEL NUC x86_64
- ソフトウェアツール: Fedora fc38 と v6.6.15 カーネル

以下の手順を実施してください。

1. 6.1 + カーネルのソースコードコードをダウンロードします。
```
$ git clone https://gitlab.com/cki-project/kernel-ark.git
$ git checkout fedora-6.6
#or
$ git checkout 9670bf06
```
2. 既存の *.config* ファイル の内容に基づいて、すべてのオプションのデフォルト値を保持し、新しい設定シンボルについて尋ねます
```
$ make oldconfig
```
3. カーネルイメージとモジュールをビルドします。
```
$ make bzImage
$ make modules
```
4. モジュールとカーネルをインストールします。
```
$ make modules_install
$ make install
```
5. FMAC コンパイルの場合は、コミュニティから FMAC パッチをダウンロードします。Linux のルートフォルダで、以下の bash コマンドでパッチのリリースを untar/apply します。
```
$ tar zxvf<release patches package> .tar.gz
#apply script for i in <release patches package>/*.patch; do patch –p1 < $i; done
```
6. プラットフォームのデフォルト設定をロードします。
```
$ make defconfig
```
7. 次の設定に示すように、モジュールを有効/無効/ロードするには、`make menuconfig` コマンドを使用するか、*.config* ファイルを編集します。
```
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. デバッグのために次の設定を有効にします。
   ```
   CONFIG_BRCMDBG=y
   CONFIG_BRCM_TRACING=y
   ```
9. カーネルモジュールをビルドします。

Note: *マルチプロセッサシステムでのコンパイルを高速化し、シングルプロセッサシステムでいくらか改善するには、"-j n" (n はプロセッサの数* 1.5) を使用してください。nproc コマンドを使用すると、搭載されているプロセッサの数を確認できます。*

```
$ make modules
```

10. 生成されたモジュールを参照用に新しいディレクトリにコピーします。
```
$ 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 は、古いカーネルで最新のドライバを実行できるようにする Linux 公式プロジェクトです。たとえば、Linux 5.15 FMAC ドライバを v4.4 から v5.15.58 まで実行できます。

Backports プロジェクトには、一連のスクリプト, パッチ, およびソース コードが含まれています。このプロジェクトは、新しいバージョンのカーネルツリーを入力として受け取り、出力として "backports パッケージ" を生成します。backports パッケージを使用して、古いカーネルで実行するためのドライバをコンパイルできます。

Figure 5 に、backports パッケージが cfg80211 の修正バージョンを提供する場所を示します。下位互換性を有効にするために、追加の compat モジュールがあります。カーネルをビルドするときに、元の *cfg80211* と *brcmfmac* (古いカーネル内) を *.config* ファイルで無効にする必要があることに注意してください。

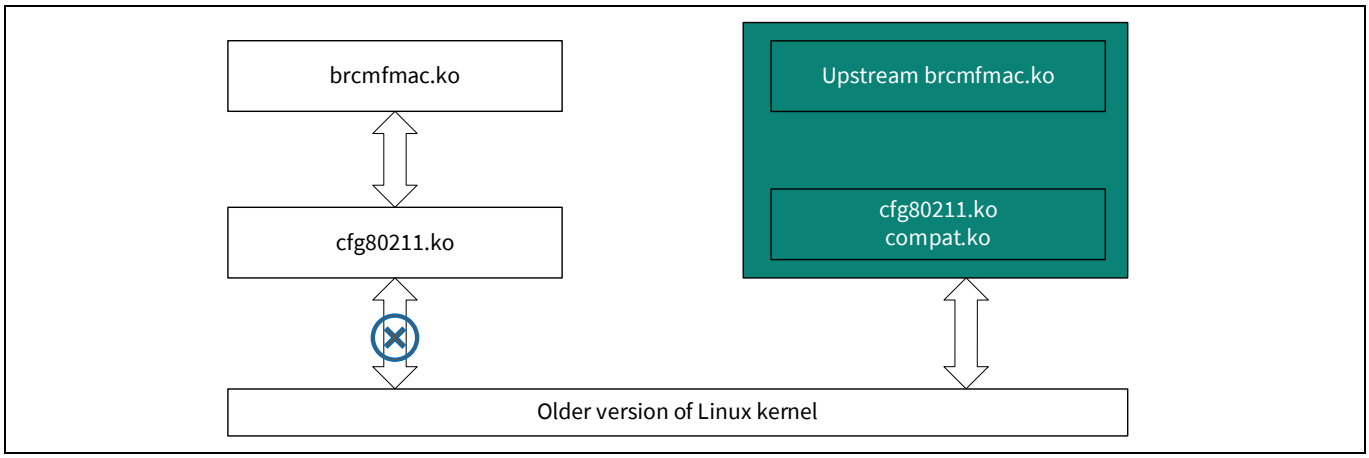


Figure 5 Backports パッケージ

インフィニオンは、バックポートパッケージのパッケージリリースモードをサポートしており、ターゲットタイプは kernel-integration ではなく loadable modules です。したがって、カーネル ソースは汚染されず、カーネルの複数のバージョンをサポートし、BSP 固有の依存関係を排除します。backports リリースパッケージをカーネルに統合するには、次の操作を行います。

```
$ 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
```

デバッグプリントを有効にするには、*.config* ファイルを変更します。

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

3.3.4 クロスコンパイル

FMAC をクロスコンパイルするには、例えば Linux ホストで、ターゲットプラットフォームに適したツールチェーンを取得する必要があります (通常は BSP で提供されます)。例えば、GNU-A Downloads からツールチェーンを取得し、任意のディレクトリに展開します。

FMAC ドライバをクロスコンパイルするには、次の手順を実行します。

1. 次のコマンドを使用して、適切なカーネルヘッダーとコンパイラを設定します。

   ```
   export MY_KERNEL=<PATH_TO_KERNEL_OBJ>
   export CROSS_COMPILE=<PATH_TO_COMPILER>
   ```

2. Backports セクションで説明されているように、FMAC ドライバにパッチを適用します。
3. 次のコマンドを使用して、FMAC ドライバを設定します。

   ```
   $ make KLIB=$MY_KERNEL KLIB_BUILD=$MY_KERNEL ARCH=arm64 COMPILE=$CROSS_COMPILE
   defconfig-brcmfmac
   ```

4. 次のコマンドを使用して、FMAC ドライバ モジュールをコンパイルします。

   ```
   $ make KLIB=$MY_KERNEL KLIB_BUILD=$MY_KERNEL ARCH=arm64 COMPILE=$CROSS_COMPILE
   modules
   ```

コンパイルされたカーネル モジュールは以下のディレクトリにあります。

- *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 FMAC ドライバのロード

FMAC ドライバをカーネルにロードするには、次の手順を実行します。

1. 次のことを確認してください。
   - ファームウェア, clm_blob, および nvram は、*/lib/firmware/cypress* フォルダにあります。
     - IFX GitHub_firmware から最新のファームウェアと clm_blob を入手してください。
     - モジュールベンダーから最新の NVRAM を入手してください。
   - すべてのバイナリには、プレフィックス *cyfmac<chip_name>-<bus_name>.bin/clm_blob/txt* が付いています。
2. 次の一連の `insmod` コマンドを使用します。

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

Note: SDIO インターフェースの場合は、次のモジュール パラメータを insmod brcmfmac.ko に追加して、最大スループットを実現します。

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

3.3.6 デバッグに関する注意事項

Wi-Fi ドライバまたはファームウェアの問題が原因でカーネルクラッシュの問題が発生した場合は、FMAC ドライバでデバッグプリントを有効にします。デバッグプリントを有効にしてカーネルモジュールをコンパイルするには、次の手順を実行します。

1. システム上で実行されているカーネルに対して *brcmfmac* カーネルモジュールをビルドする場合は、次のコマンドを使用します。

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

2. 新しいカーネルイメージをビルドするには、カーネルソースの *.config* ファイルを次のように変更します。

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

3. *brcmfmac* を実行中のカーネルに対して LKM としてコンパイルするには、次のコマンドを実行します。

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

例:

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

4. brcmfmac デバッグ ログを使用可能にするには、次のコマンドを使用します。

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

5. 必要なメッセージレベルをモジュール パラメーターとしてドライバ モジュールをロードします。

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

debug.h ファイルでは、次のメッセージ レベルが定義されています (*/v4.14.52-backports/drivers/net/wireless/broadcom/brcm80211/brcmfmac/debug.h* で利用可能):

```
#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
```

例:

- Wi-Fi ファームウェア コンソール (Wi-Fi ファームウェア内のデバッグプリントを保持するためのリング バッファ) ログを有効にするには、次のコマンドを使用します。

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

- コンソールのポーリング間隔 (250 ms) を設定するには、次のコマンドを使用します。

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

- トレースを有効にするには、次のコマンドを使用します。

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

FMAC のデバッグに関連する機能の詳細については、
drivers/net/wireless/broadcom/brcm80211/brcmfmac/debug.c にあるソース コードを参照してください。

3.3.7 よくある問題

1. モジュール形式が無効です。

   次のエラーが発生した場合は、dmesg で詳細なエラーを確認してください。

```
# 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:

   特定のカーネル バージョン用にビルドされた LKM と現在のシステムの LKM との間に不一致がある可能性があります。さらに、コンパイルされたカーネルモジュールとホストプラットフォームでは、アーキテクチャが異なる場合があります。

   解決策:

   正しいカーネルをダウンロードし、正しいカーネルイメージを再インストールしてください。

2. モジュール内のシンボルが不明です。

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

   dmesg を使用して、このエラーの根本原因を確認します。

```
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 ドライバをロードする前に、一部のモジュールが見つかりませんでした。

   解決策:

   - モジュールをロードする前に、モジュールのすべての依存関係を確認してください。
   - Linux カーネルの `dmesg` によって出力された不明なシンボルを grep して、欠落しているモジュールを特定してください。

3. `iw reg get` にチャネルがなく、国コードは `#n` です。

   `dmesg` で詳細なエラーを確認してください。

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

根本原因:

clm_blob は、*/lib/firmware/cypress* フォルダ内の CYW4373 用である可能性があります。

dmesg の「cannot find clm version」というメッセージは、インフィニオン固有のパッチが brcmfmac clm_blob に適用されていないか、ダウンロード機能がないことを示します。

解決策:

- 四半期ごとのリリーストレインから clm_blob ファイルを */lib/firmware/cypress* にコピーしてください。
- clm_blob の流れを理解するためには、AN225347 を参照してください。インフィニオンに製品固有の clm_blob をリクエストしてください。四半期リリーストレインからコピーした汎用 clm_blob を製品固有の clm_blob に置き換えてください。
- Linux ネイティブの FMAC ではなく、インフィニオン固有の brcmfmac に移行してください。

4. USB ドングルは `insmod brcmfmac` の後に起動しません。

根本原因:

NVRAM パラメーターは、ファームウェアイメージに含まれている場合があります。*/lib/firmware/cypress/cyfmac4373-usb.bin* で strings コマンドを実行して、ファームウェア イメージの末尾に NVRAM パラメーターが含まれているかどうかを確認します。

解決策:

NVRAM パラメーターが存在しない場合は、インフィニオン サポートに連絡して、 NVRAM を使用してファームウェアを構築し、既存のファームウェアを交換してください。このエラーは、四半期リリーストレインのファームウェアを使用している場合には発生しません。

4 ユーザースペース Wi-Fi ユーティリティ

4.1 wpa_supplicant

wpa_supplicant は、WEP, WPA, WPA2, WPA3 (IEEE 802.11i), および WPA-EAP をサポートするクロスプラットフォーム サプリカントです。オーセンティケータとのキーネゴシエーションを実装し、STA デバイスのローミングと関連付けも制御します。これには、次の主要な機能があります。

- WPA および完全な IEEE 802.11i, RSN, WPA2
- WPA-PSK および WPA2-PSK
- WPA-EAP (WPA – 例えば RADIUS サーバーを使用するエンタープライズ)
- CCMP, TKIP, および WEP (104/128 ビットと 40/64 ビットの両方) のキー管理
- RSN: PMKSA キャッシュ、事前認証
- IEEE 802.11r
- IEEE 802.11w
- Wi-Fi Protected Setup (WPS)

4.1.1 wpa_supplicant の依存関係

- Netlink プロトコル ライブラリ スイート (libnl)

libnl スイートは、netlink ベースの Linux カーネルインターフェースに API を提供するライブラリのコレクションです。Netlink は、主にカーネルとユーザー空間のプロセス間のソケットベースの IPC メカニズムです。これは、IOCTL の後継としてより柔軟に設計され、主にネットワーク関連のカーネル設定および監視インターフェースを提供します。IOCTL は、カーネルを汚染し、システムの安定性を損なうリスクがあります。Netlink ソケットはシンプルで、定数 (プロトコルタイプ) を *netlink.h* に追加するだけで済みます。その後、カーネル モジュールとアプリケーションは、ソケットスタイルの API を使用してすぐに通信できます。

- OpenSSL

OpenSSL は、Secure Sockets Layer (SSL v2/v3) と Transport Layer Security (TLS v1) ネットワーク プロトコル、およびそれらに必要な関連暗号化標準を実装する暗号化ツールキットです。OpenSSL プログラムは、OpenSSL の暗号ライブラリのさまざまな暗号化機能をシェルから使用するためのコマンドラインツールです。

- dbus (オプション)

D-Bus は、他のプロセスと通信するための簡単な方法を提供するメッセージバスシステムです。最新の *wpa_supplicant* バージョンには、dbus API とディレクトリ (通常はディストリビューションに応じて */var/run/wpa_supplicant/* または */run/wpa_supplicant/*) の 2 つの制御インターフェースがあり、*wpa_supplicant* が管理する各 Wi-Fi インターフェースに名前が付けられたソケットが含まれます。制御インターフェースは、デフォルトではアクティブではありません。dbus を取得するには `-u` コマンドラインオプションが必要であり、ソケットには `-O /var/run/wpa_supplicant` (または任意のディレクトリ) が必要です。

4.1.2 wpa_supplicant のコンパイル

1. ifx-hostap GitHub repo から 最新の wpa_supplicant ソースを入手してください。

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

2. *wpa_supplicant-2.10/wpa_supplicant* に移動し、システム要件に応じて *defconfig* を変更します。

   Android ベースのホストの場合は、`CONFIG_ANDROID_LOG=y` のコメントを解除して 、デバッグ出力を logcat に再ルーティングすることができます。要件に基づいてコメントを解除できるデバッグ固有のマクロが他にもいくつかあります。

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

3. 次のコマンドを使用して、*wpa_supplicant* ソースファイルをコンパイルします。

```
$ make -j4
```

Note: *クロスコンパイルする場合は、次のコマンドを使用します。*

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

4. コンパイルされたバイナリを */usr/sbin* ディレクトリ (一般的な選択肢) にインストールするには、次のコマンドを使用します。

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

Note: *生成されたバイナリを以下から手動でコピーすることもできます。*

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

Note: *openssl にヘッダーファイルが見つからない、またはバージョンの不一致などの問題が発生する場合があります。依存関係のセクションに従い、正確なバージョン要件に従ってインストールしたことを確認してください。例えば、Ubuntu ベースのシステムでは、libssl.so または libcrypto.so の場合、ほとんどの場合、`sudo apt-get install libssl-dev` で十分です。ただし、バージョン要件は、デフォルトでインストールされているバージョン(場合によっては 1.0.0) ではなく、1.0.2 または 1.1 である場合があります。その場合は、システムの libssl バージョンを wpa_supplicant の特定のリリースで必要なバージョンにアップグレードする必要があります。これらのエラーはまれですが、古いバージョンのカーネルを実行している一部のホストプロセッサでは、このようなエラーが発生する可能性があります。*

4.1.3 wpa_supplicant の設定

wpa_supplicant は、事前共有キーを含む、受け入れられたすべてのネットワークとセキュリティ ポリシーを一覧表示するテキスト ファイルを使用して構成されます。この設定ファイル内のすべてのファイル パスは、作業ディレクトリを変更できるように、フルパス (作業ディレクトリからの相対パスではなく絶対パス) を使用する必要があります。

例:

```
# 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"
}
```

wpa_supplicant.conf に関連する他の特定のマクロの詳細については、wpa_supplicant 設定ファイルを参照してください。

設定が完了したら、オフィスまたは自宅のアクセスポイントに接続するための *wpa_supplicant* を開始し、アプリケーションを開始します。次のコマンドを実行します。

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

ここで、

- `-B`: wpa_supplicant デーモンをバックグラウンドで実行します。
- `-c`: 設定ファイル (この場合は example.conf) を提供します。
- `-i`: 使用するネットワークインターフェースを選択します。

追加のデバッグプリントを行うには、コマンドの実行中に `-d` パラメーターを追加できます。詳細については、wpa_supplicant(8) Linux のメインページを参照してください。

Note: *「Failed to initialize control interface '/var/run/wpa_supplicant'」というエラーが表示された場合は、別の wpa_supplicant プロセスが既に実行されているか、wpa_supplicant のクリーンでない終了によってファイルが残された可能性があります。`$killall wpa_supplicant` コマンドを使用して、wpa_supplicant を再起動する前にこのファイルを手動で削除します。*

4.1.4 wpa_cli

wpa_cli ユーティリティは、*wpa_supplicant* と対話するためのテキストベースのフロントエンドプログラムです。これは、現在のステータスのクエリ, 設定の変更, イベントのトリガー, および対話型ユーザー入力の要求に使用されます。さらに、このユーティリティは、Extensible Authentication Protocol over LAN (EAPoL) ステートマシンパラメーターを設定し、再アソシエーションや IEEE 802.1x のログオフおよびログオンなどのイベントをトリガーできます。

wpa_cli ユーティリティは、対話型とコマンドラインの 2 つのモードをサポートします。どちらのモードも同じコマンドセットを共有します。主な違いは、インタラクティブモードでは、未承諾のメッセージ (イベントメッセージとユーザー名/パスワード要求) にアクセスできることです。

対話モードは、コマンドラインでパラメーターなしで wpa_cli を実行すると開始されます。次に、制御端末から、wpa_cli プロンプトに応答してコマンドが入力されます。コマンドラインモードでは、同じコマンドがコマンドライン引数として入力されます。

4.1.4.1 設定オプション

Table 6 にリストされているオプションは、wpa_cli を設定するための引数として使用できます。

Table 6 wpa_cli を開始するオプション

コマンド	説明
`-p path`	ソケットのパスを制御します。これは、*wpa_supplicant.conf* の *ctrl_interface* と一致する必要があります。デフォルトのパスは */var/run/wpa_supplicant* です。
`-i ifname`	インターフェースを設定します。デフォルトでは、ソケットパスで見つかった最初のインターフェースが使用されます。
`-h`	ヘルプを表示します
`-v`	バージョン情報を表示します
`-B`	デーモンをバックグラウンドで実行します
`-a action_file`	デーモンモードで実行され、wpa_supplicant からのイベントに基づいてアクションファイルを実行します
`-P pid_file`	PID ファイルの場所を提供します
`-g global_ctrl`	デフォルトの Unix ドメインソケットではなく、*wpa_supplicant* へのグローバル制御インターフェースを使用します
`-G ping_interval`	各 ping を *wpa_supplicant* に送信する前に、ping 間隔 (秒単位) を待機します。`ping` コマンドを参照してください
`command`	使用可能なコマンドを一覧表示します

4.1.4.2 wpa_cli コマンド

コマンドは、Table 7 に示すように、コマンドラインまたは対話式に操作する場合はプロンプトで発行されます。

Table 7 wpa_cli コマンド

コマンド	説明
`add_network`	ネットワークを追加します
`blacklist [bssid | clear]`	BSSID をブラックリストに追加します。追加の引数を指定せずに呼び出された場合は、ブラックリストを表示します。`clear` を指定すると、wpa_cli はブラックリストをクリアします。
`bss [idx | bssid]`	`bssid` または `idx` で識別されたネットワークの詳細な BSS スキャン結果を取得します。
`bssid network_id bssid`	SSID の優先 BSSID を設定します。
`Disconnect`	接続を切断し、再アソシエーションまたは再接続コマンド待ちます。
`enable_network network_id`	ネットワークを有効にします。
`get_network network_id variable`	ネットワーク変数を取得します。
`Help`	使用情報を表示します。
`identity network_id identity`	SSID の ID を設定します。
`ifname`	現在のインターフェース名が表示されます。デフォルトのインターフェースは、ソケットパスで最初に見つかったインターフェースです。
`interface [ifname]`	使用可能なインターフェースを表示するか、現在のインターフェースを設定するか、または複数のインターフェースが使用可能な場合に両方を表示します。
`interface_add ifname [confname driver ctrl_interface driver_param bridge_name]`	与えられたパラメーターで新しいインターフェースを追加します。
`interface_list`	使用可能なインターフェースを一覧表示します。
`interface_remove ifname`	インターフェースを削除します。
`level debug_level`	wpa_supplicant のデバッグレベルを変更します。数値が大きいほど、生成されるメッセージの数が多くなります。
`license`	wpa_cli のフルライセンスを表示します。
`list_networks`	設定済みのネットワークを一覧表示します。
`logoff`	IEEE 802.1X EAPoL ステートマシンを「ログオフ」状態にします。
`logon`	IEEE 802.1X EAPoL ステートマシンを「ログオン」状態にします。
`mib`	現在のインターフェースの MIB 変数 (dot1x および dot11) をレポートします。
`new_password network_id password`	SSID のパスワードを変更します。

コマンド	説明
`otp network_id password`	SSID のワンタイムパスワードを設定します。
`passphrase network_id passphrase`	SSID の秘密キーパスフレーズを設定します。
`password network_id password`	SSID のパスワードを設定します。
`PIN network_id pin`	SSID の PIN を設定します。
`ping`	*wpa_supplicant* ユーティリティを ping します。このコマンドを使用して、*wpa_supplicant* デーモンの状況をテストできます。
`pmksa`	PMKSA キャッシュの内容が表示されます。
`preauthenticate BSSID`	指定した BSSID の事前認証を強制します。
`quit`	wpa_cli を終了します
`reassociate`	現在のアクセスポイントへの再アソシエーションを強制します。
`reconfigure`	*wpa_supplicant* に設定ファイルの再読み込みを強制します。
`reconnect`	reassociate と似ていますが、既に切断されている場合にのみ有効になります。
`remove_network network_id`	ネットワークを削除します。
`scan`	新しい BSS スキャンを要求します
`scan_results`	最新の BSS スキャン結果を取得します。このコマンドは、スキャンとともに BSS スキャンを実行した後に呼び出すことができます。
`select_network network_id`	ネットワークを選択し、他のネットワークを無効にします。
`set [settings]`	変数を設定します。引数が指定されていない場合は、既知の変数とその設定が表示されます。
`set_network [network_id variable value]`	ネットワーク変数を設定します。引数なしで実行した場合の変数のリストを表示します。
`status`	現在のインターフェースの現在の WPA/EAPoL/EAP ステータスを取得します。
`terminate`	*wpa_supplicant* を終了します。

Note: wpa_cli コマンドまたは wpa_supplicant コマンドの実行中に「Rfkill Soft blocked」というエラーが発生した場合は、`$sudo rfkill unblock all` コマンドを使用してください。

4.1.4.3 STA/AP の一般的な使用例

STA/AP の組み合わせ	wpa_cli コマンド
オープンセキュリティで AP に接続する STA	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
WPA2 セキュリティで AP に接続する STA	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
WPA3 セキュリティで AP に接続する STA	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/AP の組み合わせ	wpa_cli コマンド
WPA3_WPA2 セキュリティで AP に接続する STA	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
オープンセキュリティを備えた 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"' 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 セキュリティを備えた 2G/5G SoftAP	wpa_cli> IFNAME=wlan0 remove_network 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 ワイヤレス認証およびプライバシーインフラストラクチャ (WAPI)

WAPI は、WLAN のセキュリティに関する中国の国家標準です。WEP や WPA よりも安全です。

WAPI 関連のサポートについては、最寄りのインフィニオン営業所または WAPI 対応バージョンの *wpa_supplicant* を提供できるインフィニオンの担当者にお問い合わせください。

Note: Linux ドライバは nl80211/cfg80211 をサポートしている必要があります。デバイスが古く、netlink ドライバをサポートしていない場合は、レガシーの wext ドライバにロールバックする必要があります。

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

4.2 hostapd

hostapd は、アクセスポイントや認証サーバーを設定するためのユーザースペースデーモンで、ほとんどのパラメーターを細かく制御できます。IEEE 802.11 アクセスポイント管理, IEEE 802.1x/WPA/WPA2/WPA3/EAP オーセンティケータ, RADIUS クライアント, EAP サーバー, および RADIUS 認証サーバーを実装します。hostapd は、これらのモードのいずれかで機能するように設定できます。もともとデーモンプログラムとして設計された hostapd は、*hostapd_cli* などのフロントエンドプログラムをサポートしています。

4.2.1 hostapd の依存関係

- libnl
- openssl

4.2.2 hostapd のコンパイル

1. ifx-hostap GitHub から最新の *wpa_supplicant* ソースを入手してください。

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

2. *wpa_supplicant-2.10/hostapd* に移動し、システム要件に応じて defconfig を変更します。

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

3. hostapd ソースファイルをコンパイルするには、次のコマンドを使用します。

```
$ make -j4
```

Note: クロスコンパイルする場合は、次のコマンドを使用します。

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

4. コンパイルされたバイナリを */usr/sbin* ディレクトリ (一般的な選択肢) にインストールするには、次のコマンドを使用します。

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

Note: 生成されたバイナリを以下から手動でコピーすることもできます。

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

4.2.3 Conf files

hostapd は、AP のセキュリティ ポリシー (802.11i,802.1x など), 国コード, パスフレーズなどを設定するテキストファイルを使用して設定されます。

次の例のような *hostapd.conf* ファイルを作成します。

```
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 は、デバッグプリントを有効にするために使用されます。プラットフォームの起動が終了した後で取り外すことができます。*

4.2.3.1 hostapd の使用法

hostapd の使用法	設定

WPA2 セキュリティを使用した SoftAP のセットアップ	`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`

WPA3 セキュリティを使用した SoftAP のセットアップ	`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 設定

DHCP デーモンには、*udhcp*, *dhcp*, および *dnsmasq* など、いくつかのオプションが用意されています。それらのほとんどは DHCP サーバーとして使用されます。それらのいくつかは、DNS サーバー機能を追加で提供します。これらは、*udhcp* などのコア OS にデフォルトで含まれている場合もありますが、それ以外の場合は、手動でパッケージをインストールできます (*dnsmasq* など)。この場合、*dnsmasq* は、AP インターフェースで DHCP サーバーと DNS サーバーを設定する方法を示す例と見なされます。

dhcpd.conf ファイルに次の変更を加えます。

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

デフォルトの *dnsmasq* 設定ファイルには、DHCP サーバーを設定するためのオプションが用意されています。したがって、デフォルトの*.conf* ファイルを編集する代わりに、ファイルをバックアップし、新しい .config ファイルを作成して、次のファイルを使用します。

```
$ 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
```

これらの行は、192.168.0.11 から 192.168.0.30 までの IP アドレスを *wlan0* インターフェースに割り当てます。これで、ネットワークルーティングの特定の修正により、*dnsmasq* を開始できます。

```
$ 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
```

使用可能な DHCP サーバーデーモンツールの一部を使用することを選択した場合でも、対応するドキュメント (パス固有の変更など) を参照して、DHCP サーバーと DNS サーバーを設定してください。

4.3 *iw*

iw ユーティリティは、ワイヤレスデバイスの設定に使用される *nl80211* ベースのユーザースペースコマンドラインユーティリティです。Wi-Fi デバイスで使用される両方の Wi-Fi ドライバをサポートします。古い *iwconfig* ツールは非推奨であり、*iw* と *nl80211* に切り替えることを強く推奨します。

4.3.1 依存関係

iw の基本的要件は、*libnl* を持つことです。*pkg-config* では、以下の依存関係を満たす必要があります。

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

4.3.2 *iw* のコンパイル

お使いのシステムのパッケージマネージャーツールを使用して、これらのパッケージをインストールし、*iw* のインストールを続行できます。ソースからコンパイルすることを選択した場合、tar balls はカーネルページで入手できます。

4.3.3 通常の使用法

`iw list` コマンドは、システム内のワイヤレスデバイスの機能を提供します。このコマンドは、Wi-Fi デバイスでサポートされているコマンドのリストも表示します。それに基づいて、`iw help <cmd_name>`を使用して、ユースケースに適したコマンドを発行できます。

4.3.3.1 WPA/WPA2/WPA3 セキュリティを備えた SoftAP

次のコマンドを使用して、セキュアな SoftAP をセットアップします。

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

AP 関連のセキュリティ設定では、hostapd.conf を使用し、*Conf* files に記載されているように `interface` パラメーターを `softap` にして、必要なセキュリティレベル (WPAx) で AP をセットアップできます。

4.3.3.2 オープン/WEP セキュリティを備えた AP に接続する STA

iw は、オープンセキュリティまたは WEP セキュリティのいずれかで接続プロセスを処理できます。WPAx セキュリティの場合は、代わりに *wpa_supplicant* を使用してください。

オープンセキュリティ:

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

同じ SSID を持つ複数の AP があり、周波数 2432 (Channel 5) の AP に接続する場合は、次のコマンドを実行します。

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

WEP は廃止され、802.11i でより堅牢なセキュリティ対策が利用できるようになりました。WEP をサポートする AP がまだあり、*iw* を使用して接続する場合は、次のコマンドを使用します。

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

5 付録

5.1 デフォルト/Yocto リリースにコネクティビティを追加するためのチェックリスト

Yocto は、広く使用されているカスタム組込み Linux ディストリビューション作成ツールです。Yocto プロジェクトは、世界中の組込み開発者が、組込みデバイスや IoT デバイスに仕立てられた Linux イメージを作成するためのテクノロジ, ソフトウェアスタック, 設定, ベストプラクティスを共有するための、またはカスタマイズされた Linux OS が必要な場合のための柔軟なツールセットとスペースを提供します。このリリースでは、Yocto プロジェクトで Wi-Fi コネクティビティソフトウェアを有効にできるため、コネクティビティソフトウェアをすぐに簡単に使い始めることができます。Yocto リリースを使用する場合は、次のビルド手順に従ってください。

1. ビルドスクリプトの tar ball 抽出します。

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

2. 作業ディレクトリを作成します。例えば、*cypress-imx-bsp* です。

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

3. 次のデータを作業ディレクトリにコピーします。

```
* 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. 初回ビルドの *setup_host_env.sh* スクリプトを実行します。これは、ホストのビルド環境を設定するのに役立ちます。

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

5. 作業ディレクトリで *build_yocto_wireless.sh* スクリプトを実行して、インフィニオンがカスタマイズした Yocto イメージを生成します。

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

6. スクリプトをユーザーのアクセス許可で実行できない場合は、次のコマンドを使用します。

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

5.2 Yocto リリース以外にコネクティビティを追加するためのチェックリスト

この四半期リリースパッケージのパッチファイルは、最新の安定版 Linux カーネルリリース (v5.15.58) に基づいています。したがって、古いカーネルは backports パッケージを使用する必要があります。このパッケージを古いカーネルまたは Linux 安定版 v5.4.18 で使用する方法の例をいくつか紹介します。

古いバージョンのカーネルで backports プロジェクトを使用している場合は、Linux カーネルイメージとインフィニオン Wi-Fi ドライバ モジュールを別々にビルドする必要があります。

カーネルイメージのビルドは、カーネル設定に記載されている手順に従って行います。インフィニオン Wi-Fi ドライバ バックポートモジュールの場合は、カーネル 6.1+ のサポートセクションに記載されている手順に従います。

5.3 ファームウェアのアップグレード

通常、四半期ごとのリリースには更新されたファームウェアが含まれます。最新のファームウェアにアップグレードするには、次のようにします。

1. コミュニティページからファイルを取得します。
2. ホストファイルシステム (*/lib/firmware/cypress/**) 内のファイルを置き換えます。
3. 次のコマンドを実行:
   ```
   $ modprobe brcmfmac.ko
   ```

5.4 i.MX8 用ツールチェーン

5.4.1 Linux ホストのセットアップ

Yocto ビルドシステムには Linux ホストマシンが必要です。使用可能なハード ディスクの最小容量は 50 GB です。ただし、ホストマシンに最大のイメージまたはディストリビューションをビルドできるようにするには、少なくとも 120 GB の容量があることが推奨されます。

本書の手順は、Ubuntu 18.04 でテストされています。

5.4.2 必要なパッケージ

1. Yocto プロジェクトのホストマシンに次のパッケージをインストールしてください。
   ```
   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. 次のコマンドを使用して、ホームフォルダに *bin* という名前のディレクトリを作成してください。
   ```
   mkdir ~/bin
   ```
3. 次のコマンドを使ってツールをダウンロードしてください。
   ```
   curl http://commondatastorage.googleapis.com/git-repodownloads/repo> ~/bin/repo
   ```

4. ツールを実行するには次のコマンドを使用してください。

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

5. */bin* ディレクトリを PATH 変数に追加してください。次の行を*.bashrc* ファイルに追加して、起動した各シェルまたはターミナルでパスを使用できるようにします。

```
echo "export PATH=~/bin:$PATH" >> ~/.bashrc
source ~/.bashrc
```

5.4.3 Yocto レシピのダウンロード

Yocto プロジェクトは、イメージをビルドする際に使用される多くのレシピで構成されています。これらのレシピは、いくつかのリポジトリから取得されます。repo ツールは、これらのリポジトリをダウンロードするために使用されます。*ea-yocto-base* リポジトリからブランチを選択する必要があります。

1. ダウンロードしたファイルのディレクトリを作成してください (次の例では *ea-bsp*)。

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

2. Git をまだ設定していない場合は設定します。「Your name」を実際の名前に、「Your e-mail」をメールアドレスに変更してください。

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

3. repo を初期化してください。この手順では、必要なすべてのリポジトリを含むファイルがダウンロードされます。それに応じて**<selected branch>**をブランチ名に変更します。

   この例では、*ea-6.1.36* ブランチを使用します。

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

4. ファイルのダウンロードを開始してください。

```
repo sync
```

これで、すべてのファイルが *ea-bsp* ディレクトリにダウンロードされました。ほとんどのファイルは、実際には *sources* サブディレクトリにあります。

5.4.4 ビルドの初期化

1. 次のコマンドを入力して、Yocto ビルドを設定してください。ディレクトリが自動的に *build-imx8mqea-com-wayland* に変更されます。

```
DISTRO=fsl-imx-wayland MACHINE=imx8mnea-ucom source ea-setup-release.sh -b
build-imx8mnea-ucom-wayland
```

2. *conf/local.conf* を編集して SSH サーバーを有効にしてください。

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

5.4.5 カスタム i.MX ツールチェーン

1. イメージをビルドしてツールチェーンを作成してください。次の例を参照してください。

```
bitbake meta-toolchain
```

   これにより、*build-imx8mnea-ucom-xwayland/tmp/deploy/sdk* ファイルが作成されます。

2. *build-imx8mnea-ucom-xwayland/tmp/deploy/sdk* の場所に移動して、ツールチェーンをインストールしてください。
3. 次のコマンドを使用して、ツールチェーンをインストールしてください。

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

   これにより、ツールチェーンが */opt/fsl-imx-wayland/6.1-mickledore/* にインストールされます。

4. 次のコマンドを使用して環境変数をソースにしてください。

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

5.4.6 イメージのビルド

セットアップが完了したので、ビルドを開始します。*ea-image-base* イメージのビルドは、次の例で示されます。

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

イメージの作成には、ホストコンピューターの機能によっては数時間かかる場合があります。

参考資料

[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

改訂履歴

版数	発行日	変更内容
**	2025-04-16	本版は英語版 002-32689 Rev. *B について、インフィニオン DEVELOPER COMMUNITY の参画者によって日本語に翻訳されたドキュメントです。
*A	2025-09-09	これは英語版 002-32689 Rev. *C を翻訳した日本語版 002-41476 Rev. *A です。

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

Edition 2025-09-09
Published by
Infineon Technologies AG
81726 Munich, Germany

© 2025 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
002-41476 Rev. *A

重要事項
本手引書に記載された本製品の使用に関する手引きとして提供されるものであり、いかなる場合も、本製品における特定の機能性能や品質について保証するものではありません。本製品の使用の前に、当該手引書の受領者は実際の使用環境の下であらゆる本製品の機能及びその他本手引書に記された一切の技術的情報について確認する義務が有ります。インフィニオンテクノロジーズはここに当該手引書内で記される情報につき、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

警告事項
技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。
インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。