Embedded Xinu 上での Raspberry Pi 3 オンボード WiFi (BCM43455) ドライバのゼロからの実装

—- スキャンから WPA2 接続・DHCP・ping 疎通まで —-

実機検証レポート（arm-rpi3 platform）

2026 年 6 月 2 日

概要

本レポートは、教育用 OS である Embedded Xinu を Raspberry Pi 3 B+（BCM2837）へ移植した `arm-rpi3` プラットフォーム上に、オンボード WiFi チップ Broadcom/Cypress BCM43455 のドライバを**ゼロから実装**し、アクセスポイントのスキャン・一覧表示から WPA2-PSK 接続、DHCP による IP 取得、そして ping 疎通までを実機で達成した記録である。Linux の `brcmfmac` ドライバおよび plan9/circle の `ether4330` ドライバをリファレンスとしつつ、Xinu の限られた環境（MMU 有効・D-cache 無効・標準 C ライブラリ僅少）に合わせて SDIO バスの bring-up、ファームウェアのロード、SDPCM/BDC フレーミング、ファームウェア内蔵サプリカント (FWSUP) による 4-way ハンドシェイク、最小限の IP スタック (ARP/ICMP) を実装した。本実装で突破した **6 つの技術的な壁**と、その診断・解決の過程を詳述する。最終的に `ping 192.168.3.52` は定常状態で**パケットロス 0%・平均 RTT 15 ms** を達成した。

1　背景と目標

Embedded Xinu は小規模で読みやすい教育用 OS であり、本プロジェクトではこれを Raspberry Pi 3 B+ の実機（BCM2837、Cortex-A53、32-bit）へ移植している。Pi 3 B+ のオンボード WiFi は BCM43455 で、SoC とは SDIO バス（Arasan SDHCI/EMMC コントローラ、物理アドレス `0x3F300000`）で接続されている。

目標は単純である：**「WiFi アクセスポイントの一覧を表示し、その中から選んで接続する」**。ただし Xinu には WiFi スタックが一切存在しないため、SDIO の最下層からファームウェアのロード、無線制御プロトコル、暗号化 (WPA2)、上位の IP 疎通までを自前で実装する必要があった。

2　システム構成

実装は単一の自己完結したファイル `apps/wifi.c`（約 2,000 行）に集約した。HTTP 経由の操作・診断インタフェースは `apps/webactor.c` が提供する。レイヤ構成を表 1 に示す。

ファームウェア（`cyfmac43455-sdio-minimal.bin`, 約 548 KB）、NVRAM、CLM(規制) ブロブは `.incbin` でカーネルイメージに直接埋め込み、SDIO 経由でチップ RAM(`0x198000`) へストリーム転送する。MMU は有効化しているが D-cache は USB-ethernet/フレームバッファの DMA を壊すため意図的に無効、命令キャッシュ (I-cache) のみ有効化している。

表 1　WiFi ドライバのレイヤ構成

レイヤ	内容	主な関数
SDIO host	Arasan SDHCI 初期化, CMD52/53	`emmc_*`, `sdio_cmd52/53`
バックプレーン	EROM コアスキャン, ARM CR4 制御	`wifi_sbmem`, `wifi_corescan`
firmware ロード	fw/nvram/clm を RAM へ転送	`wifi_fwload`, `wifi_clmload`
SDPCM/BDC	制御 (ioctl)・データのフレーミング	`wifi_wlcmd`, `wifi_data_tx`
無線制御	scan, join, 暗号設定	`wifi_scan`, `wifi_do_join`
IP 応答	DHCP クライアント, ARP/ICMP 応答	`wifi_dhcp`, `wifi_net_service`

3　実装の道のり：6 つの壁

スキャン（AP 一覧の取得）までは比較的順調に到達したが、**WPA2 接続から先**には段階的に 6 つの壁が立ちはだかった。各々を実機ログによる診断で特定し、解決した（表 2）。

表 2　突破した 6 つの技術的な壁

#	症状	原因と解決
1	`sup_wpa` が `-23`	使用中の *standard* ファームウェアに在荷サプリカント (FWSUP) が無い。`idsup` を持つ *minimal* ファームウェア (7.45.241) へ差し替え。
2	`WSEC_PMK` が `-2`	PMK 構造体 `brcmf_wsec_pmk_le` は `key[128]` を含む **132 バイト固定**。36 バイトで送っていたため拒否されていた。
3	接続しても assoc せず	`wsec`/`wpa_auth`/`auth` をコマンド (WLC_SET_*) で設定していたが、join/FWSUP が読むのは **bsscfg iovar**。全て iovar 設定に変更。
4	join が無反応 (scan のみ)	ext_join 構造体 `brcmf_ext_join_params_le` は `__packed` ではなく**自然アラインメントで 70 バイト**。packed の 65 バイトは `-14`(BUFTOOSHORT)、114 バイトはフィールドがズレて誤読されていた。
5	ping 不通 (0% 応答)	SDPCM の**送信クレジット window/フロー制御を無視**。fw は各フレームに `max_seq`(byte 9) と `fcmask`(byte 8) を載せ、window 超過の送信は黙って破棄する。RX から追従し、開くまで待って送信。
6	接続が不安定	association が AUTH で停滞し再スキャンに戻ることがある。join を最大 4 回リトライ。

3.1　壁 1・2：FWSUP ファームウェアと PMK 構造体

WPA2-PSK の 4-way ハンドシェイクは、ホスト側サプリカントを書く代わりに、ファームウェア内蔵サプリカント (FWSUP) に任せる方針とした。`brcmfmac` は `sup_wpa` iovar の GET が `-23`(UNSUPPORTED) を返さないことで FWSUP を判定する（`feature.c:349`）。RPi 配布の *standard* ファームウェアはこれを返した。capability 文字列を比較すると、*minimal* ファームウェアのみ `idsup idauth`(在荷サプリカント) を持ち、*standard* は代わりに external-SAE(`extsae-dpp-sr-okc`) を採用

していた。*minimal* へ差し替えると `sup_wpa=1` が `rc=0` で通り、PMK 構造体を 132 バイトに直すと `WSEC_PMK` も `rc=0` となった。

3.2 壁 4：ext_join 構造体のアラインメント（最大の難所）

最も解決に時間を要したのがこの壁である。`join` iovar に渡す構造体について、当初は `brcmf_scan_params_le`(入れ子の SSID を含む大きな構造体) を仮定して 114 バイトで送っていたが、ファームウェアは長さは受理するものの scan/assoc フィールドを誤読し、probe だけして association に進まなかった。新しい `brcmfmac` の `brcmf_join_scan_params_le` を仮定して 65 バイトに縮めると今度は `-14`(BUFTOOSHORT)。

決め手は構造体の**パッキングの再確認**であった。`fwil_types.h` 全体で `__packed` は 1 箇所のみで、join 関連の構造体には付いていない。すなわち**自然アラインメント**であり、実サイズは

$$\underbrace{\texttt{offsetof(assoc\_le)}}_{56} + \underbrace{\texttt{offsetof(chanspec\_list)}}_{12} + \underbrace{\texttt{sizeof(u16)}}_{2} = 70 \text{ バイト}$$

である。これは plan9 の `ether4330` が使う `wl_extjoin_params_t` とバイト単位で一致した（`scan_type@36`, `bssid@56`, `chanspec@68`）。70 バイトの正しいレイアウトに直すと、スキャンで得た実 chanspec を指定した directed join で初めて association が成立した。

Listing 1　壁 4 を解いた 70 バイト join 構造体（抜粋）

```
/* wl_extjoin_params_t (自然アラインメントで 70B):
 *   [0..3] SSID_len  [4..35] SSID[32]
 *   [36] scan_type(+3pad)  [40] nprobes  [44] active  [48] passive  [52] home
 *   [56..61] assoc bssid  [62..63] pad  [64..67] chanspec_num  [68..69] chanspec */
jp[0] = sl;                                   /* SSID_len */
for (i = 0; i < sl; i++) jp[4 + i] = ssid[i];
jp[36] = 0xFF;                                /* scan_type */
jp[40] = 2;                                   /* nprobes = 2 */
jp[44] = 120;                                 /* active_time = 120 ms */
jp[48] = 0x86; jp[49] = 0x01;                 /* passive_time = 390 ms */
jp[52]=jp[53]=jp[54]=jp[55]=0xFF;             /* home_time = -1 */
for (i = 0; i < 6; i++) jp[56 + i] = bssid[i];   /* assoc bssid */
jp[64] = 1;                                   /* chanspec_num = 1 */
jp[68] = chanspec & 0xFF; jp[69] = chanspec >> 8;
jsz = 70;
```

3.3 壁 5：SDPCM 送信クレジットとフロー制御（ping 不通の真因）

association・DHCP まで成功した後、`ping` が 100% ロスした。受信フレームのログを追加して診断したところ、Pi は **ARP 要求も ICMP echo 要求も正しく受信し、応答を生成・送出していた**にもかかわらず、ホストに届いていなかった。原因は SDPCM の送信フロー制御を無視していたことにあった。ファームウェアは送出する全フレームの SDPCM ヘッダに送信クレジット窓 `max_seq`(byte 9) とチャネル別フロー制御マスク `fcmask`(byte 8) を載せており、`txseq == txwindow` の状態やデータチャネルの FC が立った状態で送信したフレームは**黙って破棄**される。早期に送った ARP 応答はたまたま窓内で届いたが、後続の ICMP 応答は窓を超過して破棄されていた。受信フレームから窓とマスクを常時追従し、窓が開くまで待ってから送信するように修正して解決した。

4　接続シーケンスと実機検証

最終的な接続シーケンスを以下に示す。

1. `/api/wifi/scan` —- escan により AP を列挙し JSON で返す
2. `/api/wifi/join?ssid=...&pass=...` —- DOWN/INFRA/UP → scan で対象の BSSID/chanspec を特定 → `wpa_auth/auth/wsec` を iovar 設定 → `sup_wpa=1` → `WSEC_PMK`(PMK) → 70B `join` iovar → ファームウェアが 4-way を実行 → `E_LINK` up
3. `/api/wifi/dhcp` —- DISCOVER/OFFER/REQUEST/ACK で IP を取得し、ARP/ICMP 応答スレッドを起動

実機での join イベント列(`X-Wifi-EvSeq` ヘッダ)は `124(scan), 46(PSK_SUP, status 6=KEYED), 1(JOIN), 0(SET_SSID)` となり、ファームウェアサプリカントによる 4-way ハンドシェイク完了が確認できた。`GET_BSSID` は AP の BSSID `0c:73:29:8e:f8:2c` を返し association を裏付けた。DHCP は `IP=192.168.3.52, mask=255.255.255.0, gw=192.168.3.1, dns=192.168.3.1` を取得した。

4.1　2.4 GHz / 5 GHz 両バンドでの検証

同一ルータの 2.4 GHz(`0C73298EF82D-2G`) と 5 GHz(`0C73298EF82D-5G`、パスフレーズ共通) の双方で、接続・DHCP・ping を確認した（表 3)。5 GHz 側の AP はチャネル 106（`chanspec=0xe06a`）という **DFS チャネル**に在ったが、スキャンで得たファームウェアの chanspec をそのまま directed join に渡す方式のため、chanspec を手で組み立てることなく 5 GHz DFS チャネルへもそのまま接続できた。なお本ドライバの join は毎回 `WLC_DOWN` で張り直すため、バンドの切り替えに明示的な切断操作は不要である（2.4 GHz から 5 GHz へは join するだけで切り替わる)。以前 5 GHz が接続できなかったのは第 4・5 の壁（構造体・フロー制御）が原因であり、5 GHz 固有の問題ではなかったことも確認された。

表 3　実機 ping 結果（WiFi 経由、両バンド、定常状態）

バンド	チャネル (chanspec)	受信/送信 (ロス)	RTT 平均 (最小/最大)
2.4 GHz	ch6 (`0x1006`)	10/10 (0.0 %)	15.3 (11.5/28.0) ms
5 GHz	ch106 DFS (`0xe06a`)	7/8 (12.5 %)†	18.4 (15.6/22.2) ms

† 5 GHz の欠損 1 パケットは ARP 解決中の settling（`Host is down`）によるもので、解決後は連続応答（7/7)。2.4 GHz も初回試行では同様に最初の数発が settling で欠ける。

Listing 2　実機 ping 出力（2.4 GHz、定常 10 パケット）

```
64 bytes from 192.168.3.52: icmp_seq=1 ttl=64 time=11.533 ms
 ... (10/10) ...
10 packets transmitted, 10 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 11.533/15.338/27.991/4.674 ms
```

Listing 3　実機 ping 出力（5 GHz DFS ch106、抜粋）

```
Request timeout for icmp_seq 0          # ARP 解決中の settling
64 bytes from 192.168.3.52: icmp_seq=1 ttl=64 time=15.583 ms
 ... (icmp_seq 1..7 すべて応答) ...
8 packets transmitted, 7 packets received, 12.5% packet loss
round-trip min/avg/max/stddev = 15.583/18.399/22.173/2.008 ms
```

初回試行では最初の数パケットが `Host is down`（ARP 解決中の settling）で落ちるが、解決後は 0% ロスで安定する。

5　クライアント側ネットワーク：ping と NTP 時刻取得

応答（responder）だけでなく、能動的なクライアント機能も実装した。`wifi_arp_resolve()` で次ホップの MAC を解決し（宛先が同一サブネットならそのホストを、サブネット外ならデフォルトゲートウェイを ARP）、その上に ICMP echo クライアント `wifi_ping()` と NTP クライアント `wifi_ntp()`（UDP/123）を載せた。サブネット外宛はゲートウェイ経由でルーティングするため、家庭用ルータや iPhone テザリングの NAT を越えて**インターネットへ到達**できる。

実機（自宅ルータ 5 GHz 接続）での結果を表 4 に示す。NTP は Cloudflare の公開サーバ（`162.159.200.123`）へ問い合わせ、応答の送信タイムスタンプ（NTP 1900 年起点）を Unix 時刻（1970 年起点）へ変換し、JST（UTC+9）の暦日時へ整形した。

表 4　クライアント側ネットワーク（自宅ルータ 5 GHz 経由）

操作	結果
ping 192.168.3.1（ゲートウェイ）	3/3 応答
ping 8.8.8.8（インターネット）	4/4 応答（NAT 越えでインターネット到達）
NTP 162.159.200.123	unix=1780366848
→ JST 整形	**2026-06-02 11:20:48**

Listing 4　NTP 時刻取得（実機トレース）

```
[wifi] === NTP 162.159.200.123 ===
[wifi] *** NTP unix=1780366848  JST=2026-06-02 11:20:48 ***
```

iPhone テザリング経由でも ping 8.8.8.8 は 3/4 応答でインターネット到達を確認したが、NTP（UDP/123）は携帯キャリアでフィルタされる可能性があり、自宅ルータ経由が確実であった。なお NTP の初回問い合わせはゲートウェイの ARP 解決が間に合わず取りこぼすことがあるため、ARP がキャッシュされた後に成功する。本ドライバの WLAN データ経路は UDP/ICMP/ARP までで TCP は未実装のため、TLS を要する HTTPS（例：一般的なポータルサイト）の取得は範囲外である。

6　診断手法上の工夫

長時間 (約 150 秒) の join 処理の後では Xinu の TCP が大きな応答ボディを配送できず (write がブロック)、従来の「全ログを HTTP ボディで返す」方式が機能しなかった。そこで、(1) 主要な診断値を `X-Wifi-*` レスポンス**ヘッダ**に載せる（小さなヘッダは確実に届く）、(2) 即時応答する独立エンドポイント `/api/wifi/trace` で完全ログを別取得する、という二段構えで診断を進めた。イベント列を `X-Wifi-EvSeq` に CSV で載せたことが、壁 4・5 の特定に決定的に寄与した。

7　結論と残課題

Embedded Xinu 上に BCM43455 ドライバをゼロから実装し、**スキャン → AP 選択 → WPA2-PSK 接続 → DHCP → ping 疎通 → インターネット到達 → NTP による現在時刻取得**までを実

機で達成した。鍵となったのは、(i) 在荷サプリカント付きファームウェアの選択、(ii) ワイヤ構造体のサイズ・アラインメントの厳密な一致、(iii)SDPCM 送信クレジット制御の遵守、の 3 点であり、いずれも実機ログによる地道な診断で特定した。最終的に、自作ドライバは家庭用ルータ／ iPhone テザリングの NAT を越えてインターネットに到達し、NTP サーバから **JST 2026-06-02 11:20:48** を取得した。

残課題として、TCP を含む上位プロトコル（および TLS ／ HTTPS)、再接続・ローミング、複数 AP の永続設定、スキャン一覧からブラウザで選択・接続する UI などが挙げられる。

リファレンス：Linux `brcmfmac`（`drivers/net/wireless/broadcom/brcm80211`)、plan9/circle `ether4330`。ビルド：`arm-rpi3-port` ブランチ、コミット `2c4f783`。デプロイは SD カードの差し替えと完全電源断による。