Embedded Xinu 移植研究計画書

Raspberry Pi Zero 2 W ／ Raspberry Pi Pico (RP2040)

Xinu-rpi プロジェクト

2026 年 6 月

目次

1　概要と背景

本書は、既存の Embedded Xinu の Raspberry Pi 移植群（32-bit `xinu-rpi`/`arm-rpi`、AArch32 Cortex-A53 の `xinu-rpi3`、AArch64 の `xinu-rpi4` ／ `xinu-rpi5`）を土台に、**Raspberry Pi Zero 2 W** および **Raspberry Pi Pico (RP2040)** への移植を検討する研究計画書である。両者は移植の難易度が大きく異なる:

- **Pi Zero 2 W** — SoC は Pi 3 と同一ダイ（BCM2837 / Cortex-A53 / ペリフェラル基底 `0x3F000000`）。`xinu-rpi3` の `arm-rpi3` プラットフォームの**アーキ層がそのまま流用**でき、差分は WiFi・有線 Ethernet・RAM 容量に集約される。**工数：中**。
- **Pico (RP2040)** — Cortex-M0+ の**マイクロコントローラ**（MMU 無し、Thumb のみ、SRAM 264 KB、外部フラッシュ XIP）。Xinu の**アーキ層（boot ／コンテキストスイッチ／割り込み）を Cortex-M 用に新規実装**する新アーキポートになる。**工数：大**。

ターゲット	種別	流用度	工数（最小核）
Pi Zero 2 W	A53/AArch32（Pi3 同ダイ）	アーキ丸ごと流用	小〜中（1〜2 週）
Pico (RP2040)	Cortex-M0+ マイコン	上位 OS は流用・アーキ層は新規	大（3〜5 週）

■Xinu 移植の前提（共通）．Xinu は**単一アドレス空間 OS** であり、仮想記憶（MMU）を必須としない。実際 Pi 系の移植も MMU off ／恒等マップで動作している。したがって Pico の「MMU 無し」は障害ではなく、むしろ素直である。アーキ依存面はコンテキストスイッチ・割り込み・ブート・タイマに限られ、プロセス管理・スケジューラ・セマフォ・メールボックス・シェル等の上位層は**アーキ非依存 C** としてそのまま移植できる。

第 I 部　Raspberry Pi Zero 2 W 移植計画

2　前提と結論（Zero 2 W）

Pi Zero 2 W の SoC は **RP3A0 = BCM2837 ダイの再パッケージ**（Cortex-A53, AArch32, ペリフェラル基底 `0x3F000000`、全 MMIO オフセットが Pi 3 と同一、RAM 512 MB）。`xinu-rpi3` の `arm-rpi3` プラットフォームと**アーキ・基底・MMIO が完全一致**するため、CPU・割り込み・タイマ・MMU・UART・SD・HDMI・USB-DWC2 は**そのまま流用**できる。新アーキ対応は不要で、**プラットフォーム派生 + ドライバ差し替え**で到達できる。

3　ハードウェア差分（Pi 3 B+ → Zero 2 W）

項目	Pi 3 B+（現 arm-rpi3）	Pi Zero 2 W	対応
SoC/コア/基底	BCM2837 / A53 / `0x3F000000`	同一	流用
UART/Timer/IRQ/Mailbox/PM	同	同一	流用

項目	Pi 3 B+（現 arm-rpi3）	Pi Zero 2 W	対応
MMU・ctxsw・boot	同	同一	流用
HDMI	mailbox FB	同一（mini-HDMI）	流用（framebuffer_rpi）
SD(EMMC)	SDHCI +0x300000	同一	流用
RAM	1 GB	512 MB	ATAG 自動検出で吸収
有線 Ethernet	内蔵 LAN78xx(USB)	**無し**	**smsc9512/ETH0 を外す**
USB	DWC2 →内蔵ハブ (4 ポート +LAN)	DWC2 OTG 1 ポート直結	DWC2 流用・ハブ前提調整
WiFi	CYW43455(dual-band, id 0x4345)	単バンド 2.4GHz（43436/43438 系）	**FW/NVRAM/chip-id 差替**

4 移植戦略（Zero 2 W）

arm-rpi3 を壊さず、**派生プラットフォーム arm-rpi-zero2w を新設**して Pi 3 と共存させる。

- compile/platforms/arm-rpi-zero2w/（platformVars・xinu.conf・ld.script を arm-rpi3 から複製）
- system/platforms/arm-rpi-zero2w/（同上。差分のみ調整）
- DEFS += -D_XINU_PLATFORM_ARM_RPI_ -D_XINU_PLATFORM_ARM_RPI3_ -D_XINU_PLATFORM_ZERO2W_
- ビルド：make PLATFORM=arm-rpi-zero2w

補足：手早く確認するだけなら make PLATFORM=arm-rpi3 のイメージを Zero 2 W に載せても、**シリアルシェルまでは動く可能性が高い**（ETH0 のブート待ちでハングしなければ）。本番は派生プラットフォームで行う。

5 フェーズ別作業計画（Zero 2 W）

■P0 — 足場（半日）. compile/ と system/ の arm-rpi-zero2w を arm-rpi3 から複製。platformVars の DEVICES から **smsc9512 を削除**。xinu.conf から **ETH0 デバイス定義を削除** → system/main.c・apps/abcl_xinu_net.c の #ifdef ETH0 ブロックが無効化され、**ブート時の etherOpen() 無限待ちハングを回避**。**検証**：リンクまで通る。

■P1 — シリアルシェル起動（1〜2 日）. 基本流用（boot・platforminit.c・mmu.c・timer.c・割り込み）。RAM は platforminit.c が ATAG から platform.maxaddr を取得（512 MB を自動反映）。config.txt を Zero 2 W 用に（arm_64bit=0, kernel=kernel.img/kernel7.img, dtoverlay=disable-bt, enable_uart=1）。シリアルは GPIO14/15。**検証**：実機で xsh$ プロンプト到達、HDMI に framebuffer 出力。

■P2 — WiFi（**本丸**, 3〜6 **日**）. `wifi-fw/` に **Zero 2 W チップ用** `brcmfmac43436-sdio.bin` ＋ **nvram** を投入し `device/wifi/wifi_fw.c`（焼き込み）を差し替え。`apps/wifi.c` の `BCM43455_CHIP_ID 0x4345` 固定を実機の backplane chip-id 読み出し値に合わせて分岐。WL_ON 投入（firmware GPIO expander 経由）と GPCLK2 リファレンスクロックの配線を確認。上位（scan/WPA2/DHCP/ARP/ICMP）はチップ非依存で流用。**検証**：`wifi probe` → `wifi scan` → `wifi on <ssid> <pass>`→ DHCP → Mac から ping 応答。

■P3 — **ネット**/HTTP **を** WiFi **で（1〜2 日）**. 有線 ETH0 が無いので、HTTP ゲートウェイ／レスポンダを WiFi インタフェースに束ねる（Pi 3 で WiFi 越し ARP/ICMP/HTTP 実績あり）。**検証**：`curl http://<wifi-ip>:8080/...`、`wifi adhoc/aodv` でメッシュも確認。

■P4 — USB **入力(任意**, 2〜3 **日)**. DWC2 をホストモードで単一 OTG ポートに。`usb/usbkbd/usbmouse` 流用。Pi 3 のハブ前提列挙を直結用に調整。遠隔操作が効くため後回し可。

■P5 — **ウィンドウマネージャ（任意**, 1 **日）**. `apps/gwm.c`・`apps/gvideo.c`・`device/gwincon` 流用（HDMI 同一）。

■P6 — **ビルド/デプロイ整備（半日）**. `make PLATFORM=arm-rpi-zero2w` → `xinu.boot`。SD に `kernel.img`+firmware+`config.txt`+WiFi FW。

6 リスク・工数（Zero 2 W）

- **WiFi チップ FW/chip-id/WL_ON が最大の不確実性**。実機の chip-id 採取と正しい `brcmfmac43436` FW/nvram 入手が鍵。ここが取れれば上位は流用で動く。
- ATAG メモリ検出が 512 MB を正しく返すか（要実測）。
- DWC2 のハブ無し直結列挙、mini-HDMI ／ GPIO ヘッダ個体差。

工数感：最小目標（シリアルシェル +HDMI+WiFi 接続）で **1〜2 週間**、フル（USB 入力 +gwm+HTTP/メッシュ）で **3〜4 週間**。律速は P2（WiFi FW）。

第 II 部 Raspberry Pi Pico (RP2040) 移植計画

7 前提と結論（Pico）

Pico は **マイクロコントローラ**：RP2040 = Cortex-M0+ ×2, ARMv6-M, MMU 無し, SRAM 264 KB, コードは外部 QSPI フラッシュ (XIP)。Pi 系（A53/AArch32, MMU, GB 級 RAM）とは別物である。ただし Xinu はアーキ依存面が小さいため、**新アーキ** `cortex-m` ＋ **プラットフォーム** `rp2040-pico` **を作り、最小核から積み上げる**方針が成立する。本丸は**例外/コンテキスト/ブートのアーキ再実装**（ARMv7-A 用は流用不可）と **SRAM 制約**。**工数：大（新アーキポート）**。

8 アーキ/ハード差分（arm-rpi3 → RP2040）

項目	arm-rpi3 (A53/AArch32)	RP2040 (Cortex-M0+)	対応
ISA	ARM+Thumb（`-marm`）	Thumb のみ（ARMv6-M）	`-mthumb -mcpu=cortex-m0plus`
例外モデル	IRQ/FIQ バンクモード, CPSR	NVIC + HW スタッキング, MSP/PSP, EXC_RETURN	割り込み層を全面書換
ctxsw	r4-r11+lr+pc+CPSR, SYS（`arch/arm/ctxsw.S`）	r4-r11 退避 + SP 切替（PendSV/直接）	新規実装
割り込み配送	コントローラ読取 → `dispatch.c`	ベクタテーブルで各 IRQ 直行	NVIC 化
タイマ tick	BCM system timer	SysTick（M 標準）	小さな新ドライバ
MMU	恒等マップ	無し（任意で MPU）	`mmu.c` 不要
RAM	1 GB	264 KB	最小構成に絞る
ブート	firmware → kernel.img(ATAG)	bootROM → boot2(256B,XIP) → ベクタ	boot2+ ベクタ +crt0（UF2 配布）
UART	PL011	PL011（RP2040 も PrimeCell）	`uart-pl011` ほぼ流用
ディスプレイ	HDMI mailbox FB	無し	対象外
USB	DWC2	RP2040 独自 USB	別ドライバ（任意）
WiFi	SDIO (CYW43455)	Pico W: CYW43439 を PIO 駆動 gSPI	別バス・別ドライバ（後回し）

9 移植戦略（Pico）

- **新アーキ層**：`compile/arch/cortex-m/platformVars` と `system/arch/cortex-m/{ctxsw.S, setupStack.c, irq_handler.S, intutils.S, halt.S, pause.S, memory_barrier.S}`（`arch/arm` の対応物を Cortex-M 用に新規実装）。
- **新プラットフォーム**：`compile/platforms/rp2040-pico/` と `system/platforms/rp2040-pico/{start.S(ベクタ +crt0), boot2, platforminit.c, clocks.c, systick.c, nvic.c, uart-pl011}`。
- **最小デバイス構成**：`uart・tty・ramdisk・loopback・null` のみ。`network/usb/framebuffer/gwincon` は外す。
- **そのまま流用する Xinu 上位層（移植価値の核心）**：プロセステーブル・`resched`/ready list・セマフォ・メールボックス・メッセージ送受信（send/receive）・キュー・`getmem/freemem`・シェル（xsh）と組込みコマンド・tty。これらはアーキ非依存 C で無改造に近い。
- 第一弾は **RP2040 / Pico**（最も枯れている）。RP2350/Pico 2 は発展課題。

10　フェーズ別作業計画（Pico）

■P0 — ビルド骨格(1〜2 日)． cortex-m アーキと rp2040-pico プラットフォームの雛形作成。CFLAGS += -mthumb -mcpu=cortex-m0plus -mfloat-abi=soft、 DEFS += -D_XINU_ARCH_CORTEXM_ -D_XINU_PLATFORM_RP2040_、BOOTIMAGE を .uf2/.bin 出力に。リンカスクリプトは FLASH(XIP) 0x10000000(先頭 256B が boot2、続いてベクタ +.text)と SRAM 0x20000000(264 KB、.data/.bss/heap/stacks)
検証：make PLATFORM=rp2040-pico がリンクまで通る。

■P1 — Boot + UART "hello" (2〜4 日)． start.S に Cortex-M ベクタテーブル（初期 MSP=SRAM 上端, Reset, NMI, HardFault, SVC, PendSV, SysTick …）。Reset で .data コピー/.bss ゼロ→ nulluser()。boot2 は Pico SDK 由来の W25Q XIP 設定（先頭 256B + CRC)。最初は **SRAM 実行（UF2/bootrom）**で boot2 を回避してもよい。uart-pl011 は基底 0x40034000、clk_peri とボーレート除数、IRQ を NVIC に。**検証**：UF2 書込み→ UART に起動メッセージ。

■P2 — コンテキストスイッチ + スケジューラ (3〜5 日, 山場)． system/arch/cortex-m/ctxsw.S：r4-r11 退避 + lr + SP 切替（特権・MSP 共有の協調 ctxsw。Xinu の ctxsw() は関数呼び出しモデルなので直接切替が可能)。setupStack.c：新スレッド初回 ret がエントリへ args(r0-r3) 付きで飛ぶようフレーム構築（CPSR は無いので簡素化)。**検証**：2 プロセスが協調的に交互動作（procdemo 相当)。

■P3 — 割り込み + プリエンプション + 同期 (3〜5 日)． nvic.c/irq_handler：NVIC 有効化・優先度、各デバイス IRQ ハンドラ登録（ベクタ直行)。systick.c：SysTick を 100 Hz tick → clkhandler → プリエンプティブスケジューラ。排他 disable()/restore() は PRIMASK で実装。**検証**：セマフォ／メールボックス／ send/receive ／ sleep のテスト（testsuite 一部)。

■P4 — シェル (xsh) (2〜3 日)． shell+tty(UART)+ 主要コマンド（help/ps/memstat/...）を最小デバイス構成で。任意で ramdisk+ 簡易 FS、cc/abclc（SRAM 余裕次第。RP2040 では厳しめ → RP2350 向き)。**検証**：UART 端末で xsh$ 対話。

■P5 — 任意拡張（各 1〜3 週）． USB-CDC コンソール（RP2040 独自 USB device スタック)、WiFi (Pico W の CYW43439 を PIO 駆動 gSPI で。第 11 節参照)、SPI SD + FS、PIO 応用。

11　Pico の WiFi について

WiFi の有無は**モデル次第**である。

モデル	WiFi
Pico（RP2040, 無印）	無し
Pico W / Pico WH	あり：Infineon CYW43439（2.4GHz b/g/n + BT 5.2）
Pico 2（RP2350, 無印）	無し
Pico 2 W	あり（同じく CYW43439）

ただし**繋ぎ方が Pi 系と全く異なる**。Pi のオンボード WiFi は **SDIO** 接続だが、Pico W の CYW43439 は**独自 SPI（"gSPI"）を RP2040 の PIO で駆動**する方式である（クロック/データ線が一部共有、

オンボード LED も同チップ経由)。そのため **Pi の SDIO WiFi ドライバ(`device/wifi/wifi.c`)はそのまま使えず**、bare-metal Xinu で動かすには gSPI/PIO トランスポート + CYW43439 ファームウェア DL + lwIP 相当のスタックを新規実装する必要がある(**重い後続フェーズ、+3〜6 週**)。WiFi が要るなら **Pico W** ／ **Pico 2 W** を選定する前提となる。

12 マルチタスクについて

(a) 1 コアでのプリエンプティブ・マルチタスク:確実に可能で、Xinu 移植の中心的成果物。Cortex-M は NVIC + SysTick + PendSV という RTOS 向けの仕組みを備え、本計画の P2(ctxsw)+ P3(SysTick プリエンプション)でプロセス・スケジューラ・セマフォ・メールボックス・メッセージ送受信が得られる。SRAM 264 KB でもスレッドのスタックは各数百 B〜数 KB 程度で、多数の軽量プロセスを並行できる。

(b) 2 コア同時(デュアルコア SMP):追加実装が必要。RP2040/RP2350 は 2 コアだが、Xinu は標準でユニプロセッサ(Pi3/4/5 移植も 2 個目以降のコアは `wfe` で停止する単コア運用)。現実的には「1 コアで時分割マルチタスク(もう 1 コアは停止 or 専用タスク)」が第一段階で、真の SMP はスケジューラ拡張を要する別作業である。

13 リスク・工数(Pico)

- **ctxsw/例外モデルの再実装が中核リスク**(P2/P3)。ただし Cortex-M RTOS の定石(PendSV / PSP-MSP / SysTick)があり設計は確立。
- **SRAM 264 KB**:核 + シェルは収まるが、TCP/IP・cc・XFS の同時搭載は厳しく機能選別が要る(RP2350 で緩和)。
- boot2/イメージ形式(CRC, UF2)。最初は SRAM 実行で回避可。
- WiFi(gSPI/PIO) は別物で重く、第一弾スコープ外。
- ARMv6-M 制約(ハード除算なし等)は `libgcc`(`-lgcc`)で吸収。

工数感:最小核(UART シェル + プリエンプティブ・スケジューラ + セマフォ/ メールボックス)で **3〜5 週間**、+USB-CDC で +1〜2 週、+WiFi(Pico W)+ ネットでさらに **3〜6 週**。Pi Zero 2 W(最小 1〜2 週)より明確に大きく、**研究/ 教育テーマ級**である。

14 RP2040 vs RP2350(Pico 2)

RP2040/Pico を第一弾推奨(最も枯れ・資料豊富・安価)。**RP2350/Pico 2** は Cortex-M33(ARMv8-M, ハード除算/DSP/任意 FPU/MPU/TrustZone)+ SRAM 520 KB で RAM とコア性能が楽(cc/FS/ネットを載せやすい)。アーキ層は `cortex-m` を共有しつつ `-mcpu=cortex-m33` 差分を足す形で発展できる。ブート形式はやや複雑。

15 結び(共通の第一歩)

- **Pi Zero 2 W**:`arm-rpi3` を複製して `arm-rpi-zero2w` を作り(`smsc9512/ETH0` を外す)、シリアルシェル到達 → `wifi probe` で実機 chip-id を採取 → WiFi FW 整備、の順。

- **Pico (RP2040)**：`arch/cortex-m` + `platforms/rp2040-pico` の雛形（P0）→ `start.S`+`uart-pl011` で UART “hello”（P1）→ `ctxsw`/`setupStack` で 2 プロセス協調動作（P2）。ここが通れば移植成功の見通しが立つ。