# B5 Cover Page

(182x257mm or 516x728pt)

# Rust x Raspberry Pi Pico 2W で組込み開発を深く学ぶ

— PC やスマートフォンから操作できる IoT 入門 —

［著］J-IMPACT

技術書典 19（2025 年冬）新刊
2025 年 11 月 16 日　 ver 1.0

**■免責**

本書は情報の提供のみを目的としています。
本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

**■商標**

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。
また本書では、™、Ⓡ、Ⓒ などのマークは省略しています。

# まえがき / はじめに

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊ **注意** ＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

このドキュメントは、Re:VIEW のサンプル原稿を兼ねています。自分の原稿を書くときは、サンプルの原稿ファイルと画像ファイルを消してください。

# 目次

# 第 1 章
# RP2350 のメモリと内部構造

Re:VIEW Starter を使っていただき、ありがとうございます。
この章では、Re:VIEW Starter を使うために必要となる「Ruby」と「TeXLive」のインストール方法を説明します。

## 1.1 はじめに ― RP2350 の中で何が起きているのか

マイコン開発の世界では、「**LED を点滅させる**」こと（**L チカ**）が最初の一歩とされています。しかし、LED が一瞬光る裏では、**CPU、メモリ、クロック、入出力回路（GPIO）**といった多数の要素が連携して動いています。

この章では、その「マイコンの中で起きていること」を、**RP2350** という具体的なチップを例に理解していきます。

### ♣ 1.1.1 RP2350 とは何か

RP2350 は、Raspberry Pi 社が開発した **MCU（マイクロコントローラ、マイコン）**です。「Raspberry Pi Pico 2」、「**Raspberry Pi Pico 2W**」などの小型ボードに搭載され、**低消費電力・拡張性**を両立した設計が特徴です。

同社の初代マイコン **RP2040** の後継として登場し、内部構造がより洗練されています。具体的には次のような進化がありました。

- CPU コアが Cortex-M0+ から **Cortex-M33** へ
  - より高性能で、セキュリティ機能（TrustZone）を搭載
- SRAM の大容量化と分割構造化
  - より多くのタスクや無線処理を同時に実行可能
- DMA・PIO の改良
  - 入出力の高速制御を CPU に頼らず行える

現時点で、SRAM, DMA, PIO などの用語の意味がわからなくても安心してください。それぞれの詳細は、この章の中で解説します。

このように、RP2350 は「**組込みプログラミングの学習用にも、実用製品開発にも使える**」という位置づけにあります。

### ♣ 1.1.2　「LED が点滅する」までに起きていること

MCU（マイクロコントローラ、マイコン）とは、CPU・RAM・ROM・I/O ポートなどを 1 つの集積回路（IC）にまとめた極小コンピュータです。

ここでは MCU の 1 つである **RP2350** が持つさまざまな機能と、それらの動作を理解するため、リスト 1.1 のような Rust コードを考えてみましょう。

▼リスト 1.1: L チカコードの中心部

```
led.set_high(); // LED点灯
delay_ms(500);
led.set_low();  // LED消灯
delay_ms(500);
```

これは L チカをするコードの一部であり、以下のような処理をしています。

1. LED を点灯する
2. 500 ミリ秒待機する
3. LED を消灯する
4. 500 ミリ秒待機する

この数行の裏では、RP2350 の内部で次のような一連の動作が発生しています。

これらの仕組みを理解することで、単なる「LED が光る」という現象が、「どのように CPU やメモリが協力して実現されているのか」という視点から見えるようになります。

#### ① プログラム（命令）を外部フラッシュメモリに書き込む

リスト 1.1 の L チカコードを MCU に実行させるためには、このプログラムを MCU に読める形で**書き込む**必要があります。

RP2350 の場合、書き込み先は MCU 内部ではなく、**外部のフラッシュメモリ**です。書き込まれたプログラムは**命令**の羅列として保存されます。また、 `delay_ms` のような**関数**を定義して、複数の命令群をパッケージ化することもできます。

#### ② 内部メモリ（SRAM）上にデータを配置

`led` という**変数**は、MCU 内部のメモリ（**SRAM**）に配置されます。

外部フラッシュメモリ・SRAM ともに、メモリ上の情報が配置される場所には**アドレス**という数値が割り振られ、その名のとおり「情報の番地」として管理されます。

#### ③ CPU が命令を実行

プログラム（＝命令）を実行するのは **CPU** です。CPU は**レジスタ**という独自の小さなメモリ

群を持ち、そこに外部フラッシュメモリや SRAM にある情報をコピーしながら命令を処理します。

CPU が正しく処理を行うためには、ほしい情報があるアドレスを正確に把握しておく必要があります。

#### ④ GPIO レジスタへの書き込み

`led` 変数に続く `.set_high` や `.set_low` は、**メソッド**と呼ばれる関数の一種です。これらは LED の点滅と消灯を実行します。

RP2350 のいくつかのピンには汎用入出力（**GPIO**）の機能が割り当てられており、ピンにかかる電圧の高/低（**High**/**Low**）を制御します。GPIO も独自のレジスタを持っており、それらの値によってピン電圧を操作できます。

GPIO のように、MCU に搭載され、様々な機能を実現する周辺装置のことを、**ペリフェラル**といいます。

#### ⑤ タイマで時間を待つ

`delay_ms` 関数では、MCU を一定時間待機させます。そのために MCU が持つタイマの機能を利用しますが、これも各タイマが持つレジスタの操作によって実行します。

#### まとめ

これらはすべて、**メモリ空間上の異なる領域が協調して動く**ことで成立しています。RP2350 を理解するとは、この「データと命令がどこを通って動くか」を知ることにほかなりません。

次章からは、ここで登場した各要素について詳しく見ていきます。

## 1.2　メモリ — 命令とデータの居場所

情報を保存する場所を**メモリ**といいます。

RP2350 がプログラムを実行するとき、**命令**と**データ**は異なる種類のメモリに置かれます。L チカの例では、命令（プログラムの本体）は外部の**フラッシュメモリ**に、実行中に変化するデータ（変数など）は **SRAM** に配置されていると説明しました。

それぞれのメモリの特性を理解することが、RP2350 の動作を正確に把握する第一歩です。

### ♣ 1.2.1　メモリの種類とその役割

RP2350 には複数のメモリが存在します。それぞれの性質と用途をまとめると、表 1.1 のようになります。

つまり、

- フラッシュメモリ：プログラムを長期的に保存する場所
- SRAM：実行中に CPU が利用する一時的な作業領域
- ブート ROM：起動時に最初に動作する制御コード

▼表 1.1: 代表的なメモリの一覧

| 名称 | 主な用途 | 容量 | 電源を切ると情報が消えるか |
|---|---|---|---|
| フラッシュメモリ | プログラムや定数の長期保存 | 大 | 消えない（不揮発） |
| SRAM | 実行中のデータの一時保存 | 小 | 消える（揮発） |
| ブート ROM | 起動時の制御コード | 極小 | 消えない（不揮発） |

という役割分担になっています。

### ♣ 1.2.2　フラッシュメモリ ― 命令の「保管庫」

RP2350 にはプログラムを保存するためのメモリが内蔵されておらず、代わりにボード上の**外部フラッシュメモリ**にプログラムを書き込みます。本書で使用する **Raspberry Pi Pico 2W** のボード上には、4MB のフラッシュメモリが搭載されています[*1]。

RP2350 内部の SRAM にプログラムを直接書き込まない理由は、メモリの**性質とコスト構造**にあります。

- SRAM は非常に高速だが容量が小さく、電源を切ると内容が消える。プログラムを保持するには不向き。
- フラッシュメモリは書き換えは遅いが、電源を切っても消えず、安価に大容量を確保できる。

SRAM のように、電源を切ると情報が消えてしまう性質を**揮発性**といいます。それに対し、電源を切っても情報が消えないフラッシュメモリは**不揮発性**です。

容量の面でも、RP2350 が内蔵する SRAM は 520KB しかありません。これでは、多くのアプリケーションに必要な数百 KB〜数 MB 規模のプログラムを格納するには容量が不足します。

したがって、命令を長期間保持できるフラッシュメモリの外付けが必要となります。

#### 定数の保存

命令の他にも、プログラムの実行を通して値が変化しない**定数**（リスト 1.2）もフラッシュメモリに保存されます。

▼リスト 1.2: 定数の例

```
const DELAY_DEFAULT: u32 = 250;
```

### ♣ 1.2.3　SRAM ― 実行時データの作業領域

SRAM（Static RAM）は揮発性ですが、高速に読み書きでき、CPU が最も頻繁にアクセスするメモリです。命令やデータを「本」、フラッシュメモリを「**本棚**」であるとすると、SRAM は「**作業机**」であると考えることができます。本棚より容量は小さいですが、作業時にすぐ使う数冊だけを一時的に置いておける場所、というイメージです。

RP2350 には 520KB の SRAM があり、バンクという複数の独立した単位に分かれています。

[*1] RP2354 という派生チップは、プログラム保存用のフラッシュメモリをチップ内に積層した構成になっています。

これにより、CPU とペリフェラル（DMA など）が同時にアクセスしても速度低下が起きにくくなっています。

SRAM には次のような情報が配置されます。

- **変数**や**配列**など、プログラム実行中に変化するデータ
- **スタックフレーム**（関数呼び出し時の一時領域）
- **DMA バッファ**[*2]など、ペリフェラルが利用する一時的データ

L チカの例では、`led` 変数などの制御情報が SRAM に確保され、CPU はその値を参照しながら GPIO レジスタへ出力を行います。

#### スタック — 一時的な情報を積み上げる場所

### ♣ 1.2.4　ROM とブート ROM — 起動を支える小さな制御プログラム

RP2350 のチップ内部には、書き換えできない ROM 領域が存在します。この領域は製造段階で固定されたプログラムを含み、電源投入直後に CPU が最初に実行します。

とくに重要なのが**ブート ROM** です。ブート ROM は、RP2350 が起動した瞬間に実行される制御プログラムであり、主に次のような処理を担当します。

- as

#### QSPI と XIP — フラッシュメモリ上の命令を直接実行する仕組み

RP2350 は、外部フラッシュメモリを **QSPI**（Quad Serial Peripheral Interface）という高速な通信方式で接続しています。

1 本の通信線で、データを 1 ビットずつ順番に送る通信方式を**シリアル通信**といいます。代表的なものに **SPI** 方式がありますが、QSPI はその改良版で、4 本のデータ線を使用して並列に通信を行うので高速なデータ転送が可能です。

QSPI の重要な特徴は、外部フラッシュメモリを CPU の**アドレス空間**に直接マッピング（配置）できることです。この仕組みにより、CPU は、外部フラッシュメモリ上の命令を内部 SRAM 上にあるかのように読み出して実行できます。この方式を **XIP**（Execute In Place） と呼びます。

#### XIP がある場合とない場合の違い

XIP を採用していない場合、CPU がプログラムを実行するためには以下の手順を踏みます。

1. 起動時にブート ROM が外部フラッシュメモリから命令を読み出す。
2. それらを SRAM にすべてコピーする。
3. CPU が SRAM 上のプログラムを実行する。

XIP によって、CPU はフラッシュメモリの内容を直接読み出して命令を実行できるため、起動時にプログラムを SRAM へコピーする必要がありません。その結果、**起動が高速化され、SRAM を実行時データのみに使える**という利点があります。

[*2] DMA については後述します。

一方、XIP がなければ次のような処理が必要になります。

1. 起動時にブート ROM がフラッシュから命令を読み出す。
2. それらを SRAM にすべてコピーする。
3. CPU が SRAM 上のプログラムを実行する。

この方法では起動時間が長くなり、SRAM の多くをコードが占有します。RP2350 が採用する XIP 構成は、**外部フラッシュの大容量をそのまま活かし、SRAM を動的な処理専用に温存できる**という点で合理的です。

**④ GPIO レジスタへの書き込み**

この関数では、LED につながったピンの電位を **High** または **Low** にし、LED を点灯または消灯状態に切り替えます。このような単純なピンの使い方を **GPIO**（General-Purpose Input/Output、汎用入出力）といいます。また、GPIO のように、CPU の外部で様々な機能を提供する周辺回路のことをペリフェラルといいます。

CPU 同様、それぞれのペリフェラルは固有のレジスタを持っており、その値によって設定や状態確認を行うことができます。たとえば、GPIO 出力のレジスタに書き込むことで High または Low のレベルを変更したり、GPIO 入力のレジスタを読むことで現在入ってきている信号の High・Low を確認したりできます。

RP2350 をはじめとする多くの MCU では、外部フラッシュや SRAM と同様に、各ペリフェラルのレジスタにもアドレスが割り振られています。そのため、CPU からは各ペリフェラルのレジスタがメモリ空間の一部であるように見えます。このような機能を**メモリマップド I/O** といい、メモリの読み書きを通してペリフェラルを制御することができます。

詳しいアドレス構造や、どの範囲が外部フラッシュ・SRAM・GPIO に割り当てられているかは、次節で図を使って詳しく説明します。

**⑤ タイマで時間を待つ**

`delay_ms` 関数では、一定時間待機する処理を実行します。そのために MCU が持つ各タイマの機能を利用しますが、これもメモリマップド I/O の効果によって、アドレスへの読み書きとして表現できます。

## 1.3　RP2350 の内部構成とアドレス空間

この節では、**プログラム（命令）とデータが RP2350 内部のどこを通って動くか**を、メモリマップ（アドレス空間）という“地図”で確認します。ここを押さえると、後の「`memory.x`でどこに何を置くか」を迷わず決められるようになります。

### ♣ 1.3.1　全体像：何がどこにある？

RP2350 は、ユーザプログラムを実行するデュアルコア（Cortex-M33 あるいは RISC-V Hazard3

を選択）と、それらを支えるオンチップ SRAM、外部メモリ (XIP)、各種ペリフェラルで構成されています。最重要ポイントだけ先にまとめます。

RP2040 との違い — 性能と構造の進化点　初心者が混乱しやすい RP2040 との相違を整理し、RP2350 の新機能（Cortex-M33・SRAM 拡張・TrustZone など）を理解。

RP2350 の内部構成とアドレス空間　マイコンの「地図」としてのメモリマップを紹介。BootROM、SRAM、Flash、ペリフェラルなどの関係を図解。

SRAM — 実行中の作業机　変数・スタック・DMA バッファなどの実体を、実行中の視点から理解。　複数バンク構成・アクセス制御・SRAM と Flash の使い分けを説明。

Flash（XIP）— プログラムの住む場所　プログラムがどこに保存され、どのように実行されるか（XIP 動作、読み込み速度、SRAM 展開など）を解説。

GPIO・PIO・DMA — ハードウェアを動かすための仕組み　LED 点滅を例に、RP2350 で GPIO を操作する流れを説明。　PIO と DMA がどのように CPU の負荷を減らし、タイミングを正確に保つかを解説。

タイマと非同期処理 — 時間を扱うための基本　Timer や Duration による時間管理の仕組みを、ハードウェアタイマや SysTick との関連から説明。　async/await モデルの利点を述べる。

BLE 制御の基盤 — 無線機能とメモリ構成の関係　Pico 2W との接続は後の章で扱うが、ここでは RP2350 内部で CYW43 通信（PIO+DMA）と BLE スタックがどう支えられているかを整理。

共有データと同期 — Atomic 変数とタスク間通信　AtomicU32 を例に、スレッドセーフにデータを共有する仕組みを理解。マルチコア化や割り込み安全との関係も補足。

RP2350 のメモリマップと memory.x への橋渡し

CPU は外部フラッシュや SRAM のアドレスを走り回りながら処理を実行します。

#### ② CPU コアが命令を実行

外部フラッシュに書き込まれたプログラムは命令として読み出され、**CPU コア**（処理を実行する部分）で順次処理されます。

さらに、CPU は命令処理を複数段階に分け、並列に実行して効率化することもできます。このような仕組みを**パイプライン**といいます。

#### ③ 内部メモリ（SRAM）上の変数やスタックにアクセス

リスト 1.1 に登場する `led` という**変数**は、MCU 内部のメモリ（**SRAM**）に配置され、CPU が読み書きできるようになります。

メモリ上の情報が配置される場所はアドレスと呼ばれ、その名のとおり「番地」として扱われます。CPU は外部フラッシュや SRAM のアドレスを走り回りながら処理を実行します。

CPU は**レジスタ**と

`delay_ms` は**関数**です。関数とは、一連の命令をまとめたパッケージで、他の命令と同様、外部フラッシュに情報が保存されています。

CPU が外部フラッシュを読んでプログラム（命令）を順次処理していくと、「次の処理は関数

『A』を実行」と書かれた命令に行き当たります。すると CPU は「関数『A』」が書かれているアドレスにジャンプして、そこに書かれた処理を行います。

CPU は「関数『A』」の実行中に、必要な変数を作ることがあります。ただし、この変数は「関数『A』」の中でのみ必要なものなので、関数の実行後に破棄されます。このような一時的な変数を**ローカル変数**といいます。

また、「関数『A』」の実行後は、CPU はジャンプ前に実行していた命令のアドレスまで戻ってきて、あとの処理を続ける必要があります。そのために、CPU は「関数『A』」のあどれすにジャンプする前に、現在地に相当する**戻りアドレス**を保存しておきます。「関数『A』」の実行後、CPU がそこに戻ってきた後は、戻りアドレスは不要になるため破棄されます。

ローカル変数や戻りアドレスなどの一時的な情報は、SRAM 内部の**スタック**と呼ばれる領域に積み上げられ、不要になったら削除されます。

④

# 第 2 章

# L チカ

Rust のインストール
https://rust-lang.org/ja/learn/get-started/
Git のインストール
https://git-scm.com/downloads

# 付録 A
# RP2040 と RP2350 の比較

RP2350 は、Raspberry Pi 社が開発した初代マイコン **RP2040** の後継にあたります。外観や開発環境は似ていますが、内部構造や動作仕様には大きな進化があります。この補章では、RP2040 で学んだ知識を RP2350 でどう活かせるのか、という視点から両者の違いを整理します。

## A.1 概要：互換性を保ちながら進化した RP シリーズ

RP2040 と RP2350 は、いずれも「低コストで高性能なデュアルコア MCU」という設計思想を共有しています。GPIO 番号、PIO の構造、外部フラッシュブートなどの基本設計は共通です。しかし、RP2350 では**性能・セキュリティ・通信機能・アーキテクチャの柔軟性**が大きく向上しました。

▼表 A.1: RP2040 と RP2350 の比較

| 項目 | RP2040 | RP2350 | RP2350 での変更点 |
|---|---|---|---|
| メイン CPU | Dual-core Arm Cortex-M0+ | Dual-core Arm Cortex-M33 / Dual-core RISC-V Hazard3 | Core0, Core1 それぞれでアーキテクチャを選択可能 |
| 最大クロック | 133MHz | 150MHz | 動作周波数の向上 |
| オンチップ SRAM | 264KB（6 バンク） | 520KB（10 バンク） | バンク構成拡張 |
| パッケージ（ピン数） | QFN-56 | QFN-60 (RP2350A), QFN-80 (RP2350B) | ピン数の増加 |
| DMA | 12 チャネル、割り込み数 2 | 16 チャネル、割り込み数 4 | 転送機能強化 |

表 A.1 で示したのは代表的な変更点のみです。他にも電源周りやクロック端子周りなども

RP2040 から変化していますので、RP2350 を使ってボード設計をする場合には、公式マニュアルを確認してください。（ただし、公式マニュアルは 1300 ページ以上あり、RP2040 のマニュアルと比較すると、約 2 倍の長さになっています）

## A.2 CPU アーキテクチャの選択性

RP2040 は Cortex-M0+ を固定で 2 コア（Core 0, Core 1）搭載していました。Cortex-M0+ は、Arm Cortex-M シリーズの中でも低グレードの位置づけです。

これに対して RP2350 は、Arm コアとして Cortex-M33 を搭載しました。Cortex-M0+ と比較するとかなりグレードアップし、以下のような特徴を持ちます。

/ノンセキュア分離
計算ユニット）搭載
信号処理）にも対応

また、RP2350 は RISC-V Hazard3 も搭載しています。このコアの特徴は以下のとおりです。

32bit IMAC 命令セット（整数＋乗除算＋圧縮命令対応）
て拡張・解析が容易

簡単にいうと、Arm コアは高性能ですがライセンスで権利がガチガチに守られているのに対して、RISC-V コアは比較的簡素な作りであるもののオープンソース、という違いがあります。実際、RISC-V Hazard3 は以下のリポジトリでソースが公開されています。

[外部リンク] RISC-V Hazard3 の Github リポジトリ
https://github.com/Wren6991/Hazard3

RP2350 は RP2040 のデュアルコア構造を受け継ぎ、Core 0, Core 1 のそれぞれで Arm または RISC-V を選択できるようになっています。つまり、合わせて 4 個のコアを搭載していますが、ここから 2 個選んで並列動作させることができます。

この「選べるアーキテクチャ」が最大の変更点です。

## A.3 メモリ構成の拡張

RP2040 の 264KB SRAM (6 バンク) に対し、RP2350 は 520KB SRAM (10 バンク) を内蔵します。さらに、

外部 QSPI 経由で 最大 16 MB までの XIP Flash および PSRAM をサポート。

DMA 専用領域を確保しやすくなった

無線スタックや画像処理などの大容量処理に対応

Secure/Non-Secure 領域を分離可能

この多バンク構造は、DMA 転送やマルチコア実行時のメモリアクセス競合を低減します。

## A.4 DMA・PIO の改良

RP シリーズ独自の **PIO (Programmable I/O)** は 3 ブロック（計 12 ステートマシン）へ拡張されました。DMA チャネルも 16 に増え、PIO ⇔ メモリ間の高速転送を CPU 負荷ゼロで実行できます。

これにより、SPI/$I^2$ S のソフト実装や LED 制御、無線モジュール（CYW43）との通信を効率的に行えます。

また、DMA には、セキュリティ対応・メモリプロテクション対応機能も追加されています。

# あとがき / おわりに

いかがだったでしょうか。感想や質問は随時受けつけています。

## ♣ 著者紹介

**カウプラン機関極東支部** (@_kauplan)

“賞味期限が 1 年にも満たないようなツールやフレームワークに振り回されるのを許しておけるほど、我々の人生は長くはない。”

- https://kauplan.org/
- 『パンプキン・シザーズ』推し
- 『ワールド・トリガー』推し
- 『プリンセス・プリンシパル』推し

## ♣ 既刊一覧

- 『SQL 高速化 in PostgreSQL』（技術書典 2）
- 『オブジェクト指向言語解体新書』（技術書典 3）
- 『jQuery だって複雑なアプリ作れるもん！ 』（技術書典 4）
- 『Shell スクリプトでサーバ設定を自動化する本』（技術書典 5）
- 『Ruby のエラーメッセージが読み解けるようになる本』（技術書典 6）
- 『わかりみ SQL』（技術書典 7）
- 『Python の黒魔術』（技術書典 8）
- 『カウプランレポート vol.01』（技術書典 10）

# Rust x Raspberry Pi Pico 2W で組込み開発を深く学ぶ

PC やスマートフォンから操作できる IoT 入門

---

2025 年 11 月 16 日　ver 1.0 （技術書典 19）

著　者　J-IMPACT
発行者　大野 駿太郎
連絡先　sohno@ushitora.net
https://j-impact.jp/
@j_impact_jp (https://twitter.com/j_impact_jp)

---



*(powered by Re:VIEW Starter)*