WEEK 1 ／ 組込みソフトウェア講義

マイコンの基礎と Lチカの深追い

LEDを点滅させるコードをマイコンの仕組みまでたどって理解し、設計・デバッグ時の手札を増やす

目的 ①

困ったときに公式の資料（データシート）から仕様を読み解けるようになる

目的 ②

深く仕組みを知っておくことで、
問題が起きたときの対処の流れが掴める
（デバッグそのものはまた今度）

目的 ③

仕組みを知ることで、設計の手札（解空間）を広げて最適な選択ができるようになる
（今日の宿題で体感してもらう）

前回の補講で、LEDをチカチカさせる回路の要素を学んだ

- **回路図**

  接続のみを表現する。線で繋がっていなくてもラベルで繋がる。

- **電子部品の機能**

  LEDは光る部品、抵抗は電流を絞る部品。

- **アナログとデジタル**

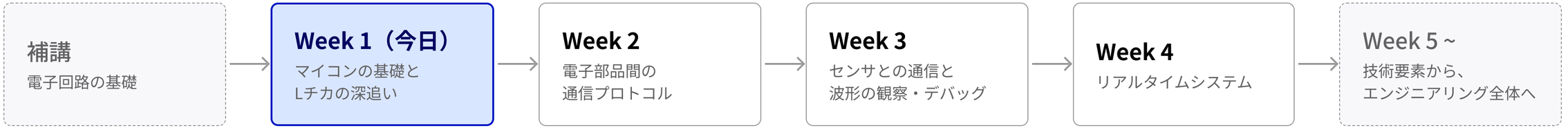
  電気回路は連続的だが、マイコンはデジタルで動く。

- **マイコン**

  CPU・メモリ・入出力が1つに収まり、ソフトとハードを橋渡しする。

0.2 今回の位置づけ

Week 1〜4では、模擬衛星の技術要素、電子工作の基礎を学ぶ



- **「根拠を持ったエンジニアリング」を学ぶ前に、そのフィールドの技術要素を学ぶ**

0.3 今日の流れ

SDKのソース、データシート、原理からの探索の3つを使って、Lチカの1行を説明し、手札を広げる

Lチカの回路・ソフト
まずは雑にAIで知る

→

① マイコンの仕組み
全ては数値の読み書き

→

② コードの深追い
SDKのソースの中へ

→

③ データシート
読み書きするレジスタの中身

→

④ 手札を広げる
原理を設計・デバッグに生かす

- **この4ステップで「動かす」から「原理を知って選べる」へ進む**

すでに学だ LED・抵抗・マイコンを繋げる

- **回路シミュレータ Wokwi で実験**

  https://wokwi.com/projects/468718947173297153

- **LED**

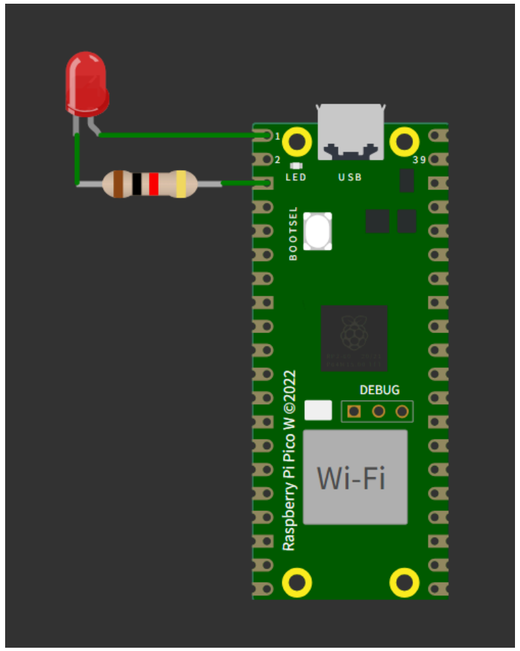
  電気を流すと光る

- **抵抗**

  LEDに電流が流れすぎて壊れないようにする

- **Raspberry Pi Pico**

  回路の動作をマイコンからソフトで制御する



GPIOから抵抗・LEDにつないだLチカ回路（Wokwiで作成）

最初はAIでざっくり始める

このコードを書いたClaudeとの会話：https://claude.ai/share/4c39ff64-aa30-4863-855f-49f0816b049c

```c
#include "pico/stdlib.h"
#define LED_PIN 0  // GP0
int main() {
    gpio_init(LED_PIN);
    gpio_set_dir(LED_PIN, GPIO_OUT);
    while (true) {
        gpio_put(LED_PIN, 1);   // 点灯
        sleep_ms(500);
        gpio_put(LED_PIN, 0);   // 消灯
        sleep_ms(500);
    }
    return 0;
}
```

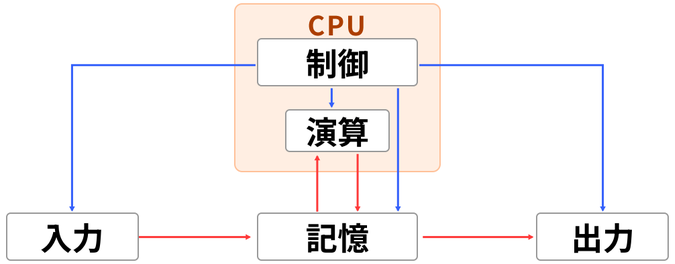
ネットの入門記事やAIの回答はここで完結する。今日はここから「なぜ動いたのか」へ進む。

CHAPTER 01

マイコンの動作をプログラム視点で理解する

マイコンの動作は全て数値の読み書き

補講のマイコン像



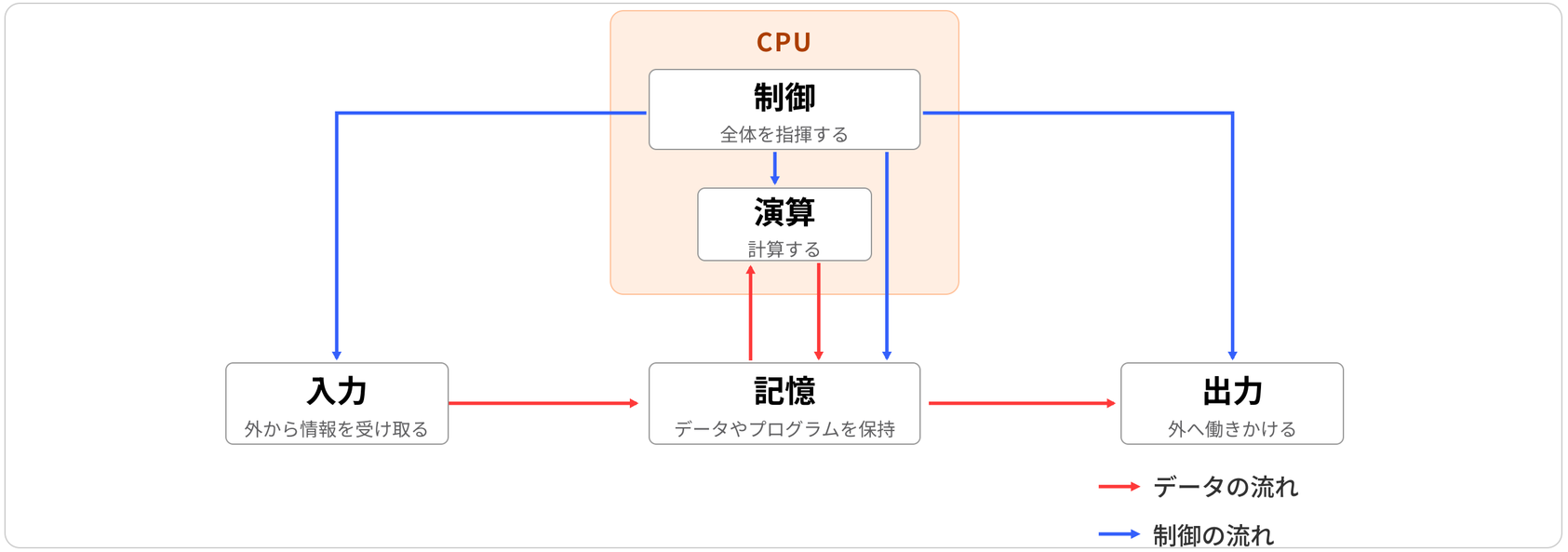
プログラムから
見ると

五大装置の動作全て

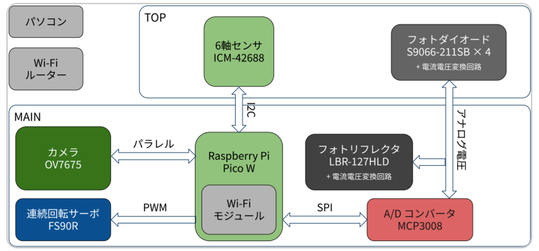
ほぼ全部が「数値の読み書き」

＝ 何番地に、何を ― の1種類に一本化して見える

入力・記憶・演算・制御・出力
コンピュータはこの5つで説明できる。

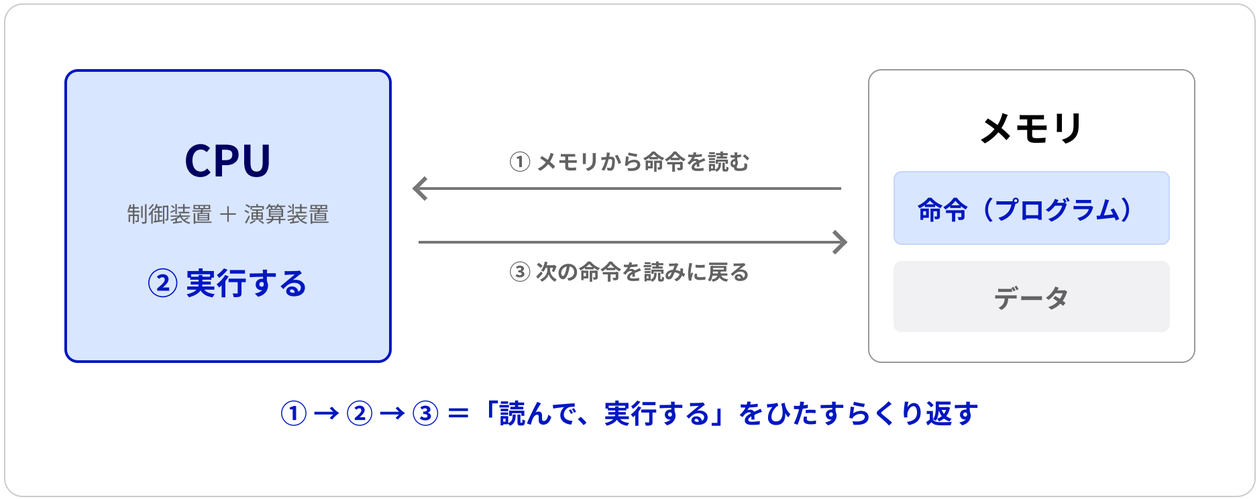


データは 入力 → 記憶 ⇄ 演算 → 出力 と流れ、**制御がすべてを指揮する**



マイコンの中身だけでなく、模擬衛星全体も骨組みは同じ ― 光センサが**入力**、Pico Wの計算が**演算**、モーターやLEDが**出力**にあたる。

CPUの仕事は「メモリから命令を読んで実行する」の繰り返し



- **メモリ**

  プログラムもデータも数値として保存される

- **読む → 実行 → 次を読む**

  普通のコンピュータ（参考：ノイマン型と言う）は
  このサイクルをひたすら回す

- **「実行」も実は数値の読み書き**

  計算も、制御も、入出力も全部

「ソフトの実行 ＝ 数値の読み書きの繰り返し」
—— これだけ覚えればOK。

計算はイメージ通り、メモリの読み書きで実現

```
int a,b,c;  // a,b,cという領域をメモリに確保する
a = 1;      // メモリのaの領域に数値を書き込む
b = 2;      // メモリのbの領域に数値を書き込む
c = a + b;  // メモリからaとbを読み込み、足し算した結果をcに書き込む
```

- **変数とメモリ**

  計算に使う「変数」はメモリに保存する。

- **変数名は人間向け、アドレスはコンピュータ向け**

  変数をメモリのどこに置くか、「アドレス」という数値で管理する。

- **計算の実体はメモリへの書き込み**

  加算結果の数値を書き込む。

数値計算なんだから、メモリに数値を読み書きするのは当たり前では…？

入出力も数値の読み書きでできる

- **入出力はCPU外の周辺回路が担う**

  これを**ペリフェラル**という。

- **周辺回路とCPUをつなぐ掲示板がある**

  これをペリフェラルの**レジスタ**という。

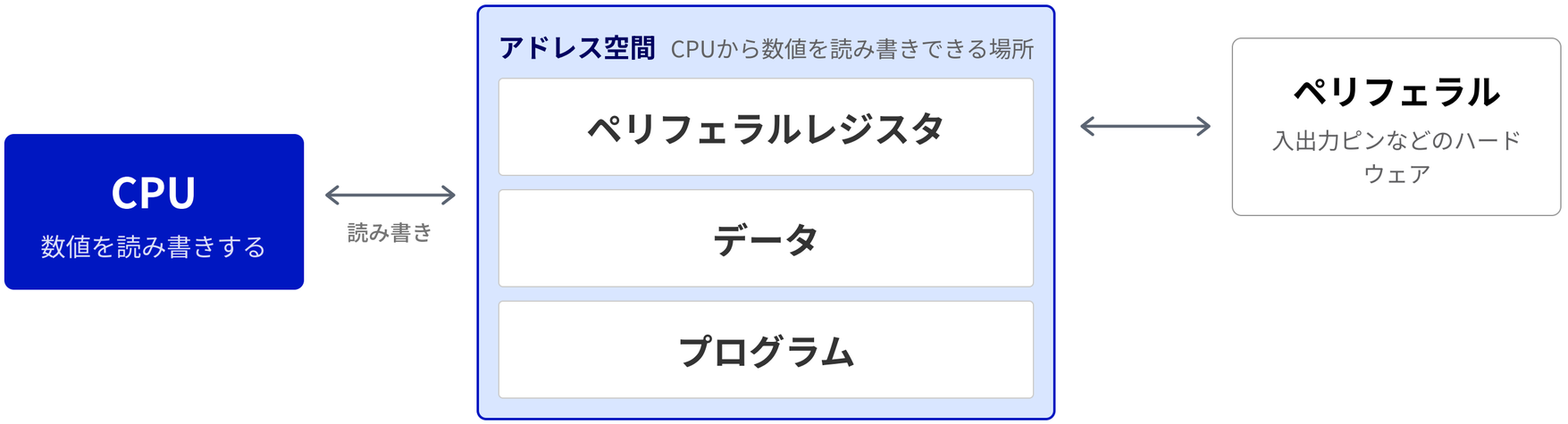
- **掲示板を読んで回路の状態を知る**

  レジスタからCPUが数値を読み込む。

- **掲示板に書いて回路に指示を出す**

  レジスタへCPUが数値を書き込む。

ペリフェラルレジスタも、普通のメモリも、CPUから見たら同じ！



発展：興味あったら調べてみてね

- 五大装置の残り一つ、「制御」は数値の読み書きでどう実現する？
- if文やfor文はどうやって動くのか？

キーワード：**プログラムカウンタ**

1.6 この章のまとめ

マイコンはCPUから数値を読み書きして、演算/入出力/制御する

ここは覚えて次へ

①

CPUは読み書きの繰り返し

プログラムもデータも番地付きの数値。CPUは「読む→実行→次を読む」だけ（ノイマン型）。

②

演算も入出力も読み書き

演算はメモリへ、入出力はペリフェラルの掲示板＝レジスタへ、数値を読み書きすること。

③

CPUには全部アドレス空間

メモリもレジスタもプログラムも一つの番地空間。だから制御（発展）も同じ仕組みで実現できる。

CHAPTER 02

Lチカの深追い：ラズピコの具体例で学ぶ

gpio_put(LED_PIN, 1) の1行からLED点灯までをたどる

```
gpio_put(LED_PIN, 1);    // 指定したピンの出力を H にする
```

gpio_put の1行
Pico SDK の関数

→

SDKのソース
関数の中へ

→

レジスタ（番地）
SIOへの書き込み

→

ピンの電圧
H（約3.3V）

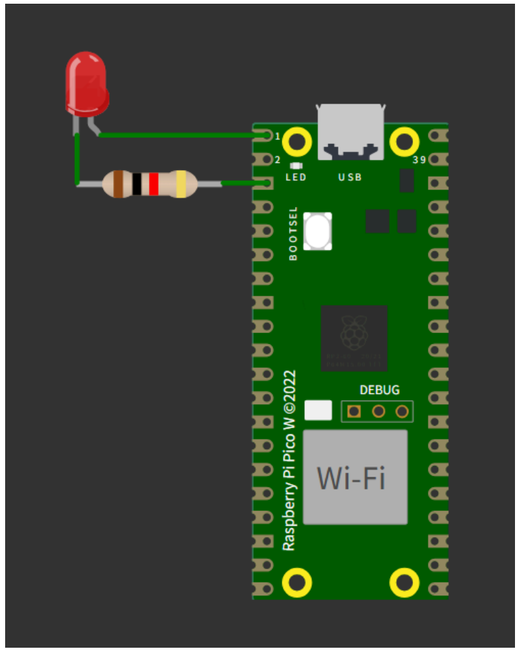
→

LED 点灯
物理の結果

■ この中身を知るのが目的ではなく、「中身を知るプロセス」を知ることが目的

ソフトウェアからマイコンの出力ピンを制御する流れを知る

```c
#include "pico/stdlib.h"
#define LED_PIN 0  // GP0
int main() {
    gpio_init(LED_PIN);
    gpio_set_dir(LED_PIN, GPIO_OUT);
    while (true) {
        gpio_put(LED_PIN, 1);   // 点灯
        sleep_ms(500);
        gpio_put(LED_PIN, 0);   // 消灯
        sleep_ms(500);
    }
    return 0;
}
```



GPIO 0番 → 抵抗 → LED の回路（Wokwiで作成）

最終的に信じるのは一次資料、二次資料も使いこなす

公式ドキュメント 最も信頼

公式リファレンスとも ／ ＝一次資料

- ✓ その技術や製品**そのもの**について書いてある
- ✓ 仕様や使い方を**ちゃんと学べる**
- ✓ **信頼性が最も高い**
  外部サイトは情報が古いかも

他の資料 AI・ブログ・書籍・動画など ／ ＝二次資料

① **公式の記述を要約しているもの**（AIは基本ここ）
分かりやすいけど、結局**公式のトップページが優秀**

② **公式に書いてないことを書いたもの**
- **バグ**は外部サイトに書いてあることが多い
- 性能検証や比較などもありがち

③ **それを外部の視点から書いたもの**
- 他の技術や製品と比べて書いてある
- 外から見るので**全体像が掴める**

一次資料を調べるときも、入口はAIでOK

AIに聞いてみる：https://claude.ai/share/d4423767-271f-4df3-9213-740db517235a

AIに聞いたうえで、一次資料を深ぼる

① **調査対象の「役割」**（SDKのドキュメント）

② **実装**（ソースコード）

③ **ソフトが最終的に叩くレジスタ**（ソースコード）

④ **物理レジスタの意味**（ハードウェアのデータシート）

参考資料 すべて一次資料

① SDK ドキュメント

Raspberry Pi Pico-series C/C++ SDK ドキュメント

pip-assets.raspberrypi.com/…/RP-009085-KB-2-raspberry-pi-pico-c-sdk.pdf

②③ ソースコード

raspberrypi / pico-sdk

github.com/raspberrypi/pico-sdk

④ データシート

RP2040 データシート

pip-assets.raspberrypi.com/…/RP-008371-DS-1-rp2040-datasheet.pdf

gpio_put の中身は数行で、ビットマスクを作ってset系かclr系のレジスタへ書き分けている

```
// gpio_put の実装の骨格（抜粋）
static inline void gpio_put(uint gpio, bool value) {
    uint32_t mask = 1ul << gpio; // 対象ピンのビットだけを立てた値
    if (value)
        gpio_set_mask(mask); // 1にする側
    else
        gpio_clr_mask(mask); // 0にする側
}
```

- **mask ＝ 操作するピンだけ選択する値**

  ここの文法は分からなくても今日は一旦OK

- **value で set / clr に振り分け**

  1ならset系、0ならclr系へ渡す

- **まだ関数呼び出しが残る**

  次に gpio_set_mask() を調べる

gpio_put をたどった先は、sio_hw への代入文1つ

```
// gpio_set_mask の骨格（抜粋）
static inline void gpio_set_mask(uint32_t mask) {
    sio_hw->gpio_set = mask;
}
```

- **見つかったのは「レジスタへの書き込み」**

  sio_hw->gpio_set というものに、
  mask（Hにするピンの位置）を書き込んでいる

- **さらにソースをたどると、sio.h に gpio_set がある。**

  これが GPIO_OUT_SET というレジスタだと書いてある。

- **これは 1.4 で学んだペリフェラルレジスタ**

  ここに数値を書くと、周辺回路が勝手に動く

補足 static inline void や ->（アロー演算子）といった文法は、あとでAIに聞いて調べてみてください。
今は、mask という変数を = で sio_hw->gpio_set というものに代入する、ということが分かればOK。

ソースから分かるのは書き込み先の番地までで、その番地の意味はデータシートが定義する

① 呼び出し ✓ 済 — `gpio_put(LED_PIN, 1)`

↓

② SDK内部 ✓ 済 — `gpio_set_mask(mask)`

↓

③ ソフトの最下段 — **SIOの番地への書き込み**
ソースから分かるのはここまで

↓

④ ハードの仕様 — **この番地に書くとピンが H になる？**
これからデータシートで確認

- **分かったこと。**
  gpio_put の実態はSIOの特定番地への書き込み。
- **まだ分かっていないこと。**
  その番地に書くと、なぜピンが H になるのか。
- **次はデータシートで裏を取る。**
  レジスタの先、ペリフェラルの仕様は
  ハード側の一次情報にあたって調べる。

2.7 この章のまとめ

Lチカの深追いで「覚えて次に進んでほしい」核はこの3つ

ここは覚えて次へ

①

最下段は代入1つ

gpio_put を潜ると、最後は sio_hw への代入＝「SIOの決まった番地へ mask を書け」だった。

②

便利な関数は包装

その正体は、番地への書き込みをポインタで隠した包みにすぎない。

③

意味はデータシート

ソースから分かるのは番地まで。番地の意味はデータシートが定義する（照合先は一次情報）。

Pico Wのオンボード LEDは、RP2040ではなく無線チップにつながっている

- **Pico（無印）は GPIO 25。**

  オンボードLEDがRP2040のGPIO 25につながっている。

- **Pico W は無線チップ側。**

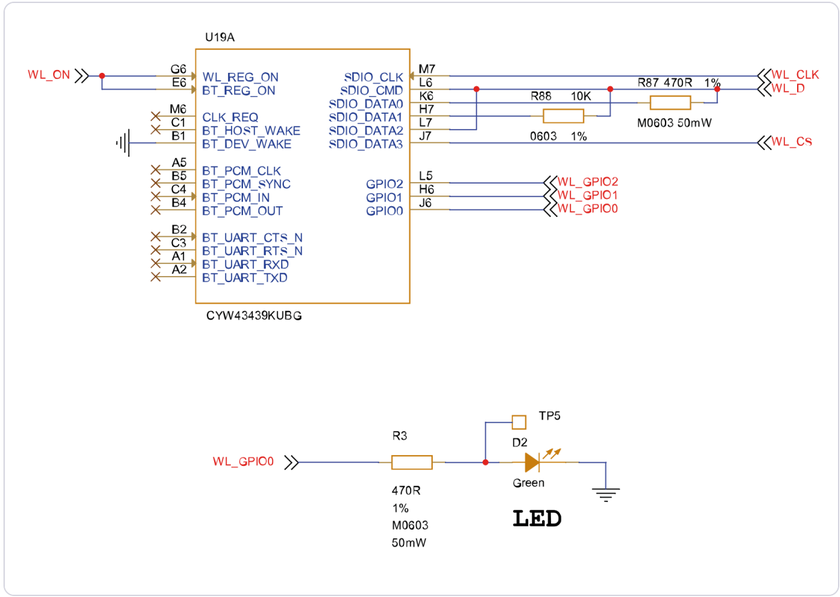
  同じ位置のLEDが無線チップ CYW43439 のGPIOにつながる。

- **Pico WではSIOを経由しない。**

  だから今日たどりたいRP2040のSIOの道には乗らない。

- **回路図の確認を怠ると迷子に。**

  LED 1つでも、間違ったデータシートを読んで時間を溶かす。



Pico W 回路図：オンボードLED（D2）は無線チップ CYW43439 の WL_GPIO0 に接続

CHAPTER 03

データシートで裏を取る

番地の意味はチップの設計が決めており、その仕様書がデータシートである

- **数百ページの英語PDF。**

  普段頭から読む文書ではない。（暇なときに目次ぐらいは全部読むと良い）

- **辞書として引く。**

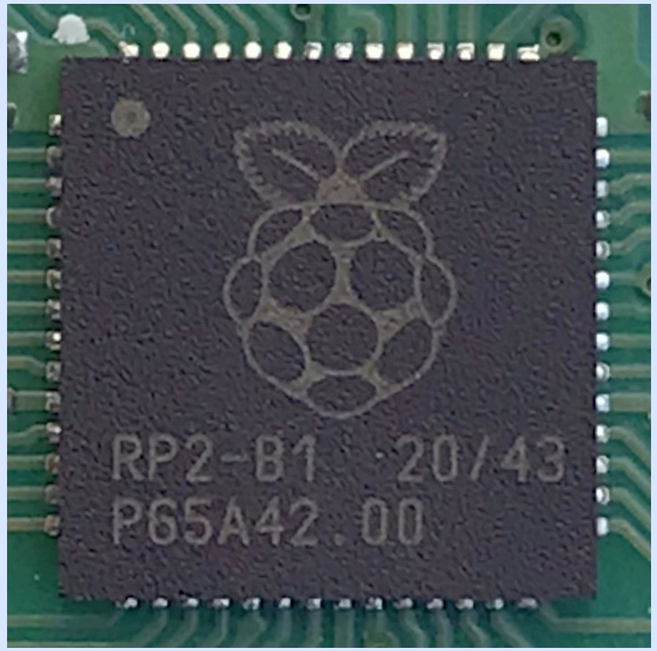
  目次と一覧表から必要な箇所だけを引く。

- **今日調べるのは GPIO だけ。**

  SIOのgpio_setは、どの番地で、書くと何が起きるか。

- **今日学ぶのはデータシートの読み方。**

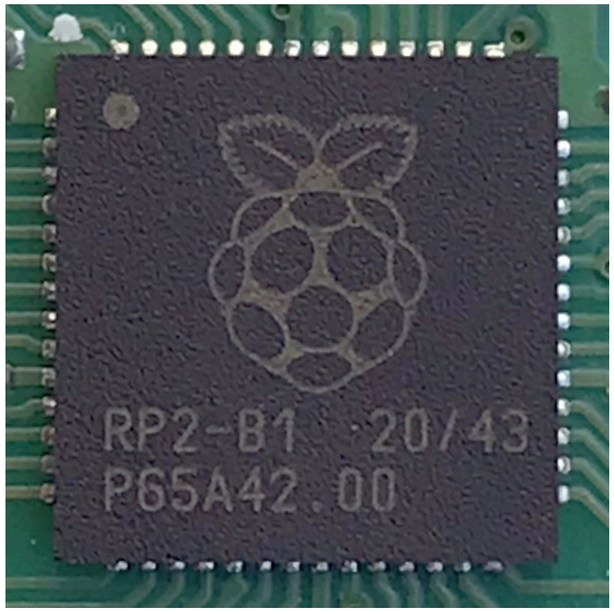
  調べたことの内容は忘れてOK。



RP2040 — この中身の仕様がデータシートに書いてある

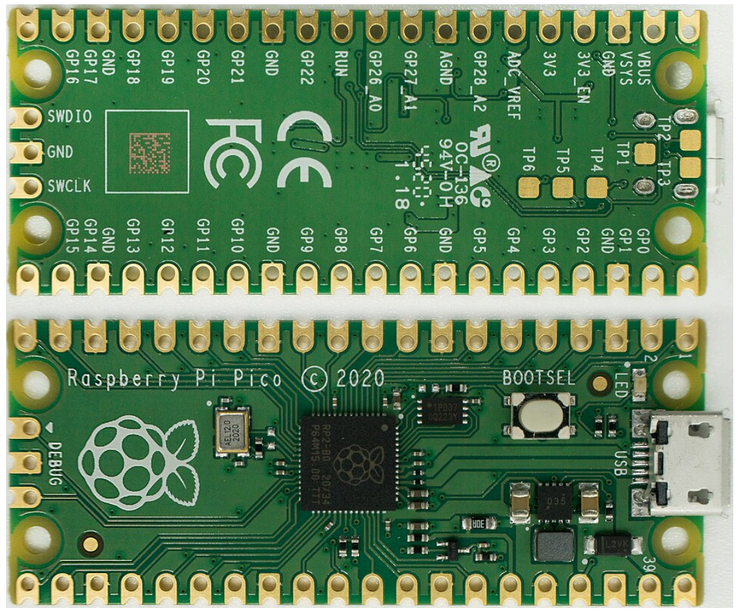
レジスタの仕様はRP2040のデータシートに、電源や周辺部品はPicoのデータシートに載っている

チップ：RP2040 のデータシート ← 今日開くのはこちら



CPU・アドレスマップ・ペリフェラルとレジスタの仕様。
今日調べたい SIO のレジスタはここ。

ボード：Raspberry Pi Pico のデータシート



ピン配置・電源回路・オンボードLEDの配線。基板の上の話はこちら。

GPIO_OUT_SET の詳細を読む

- **対応するピンが「atomic bit-set」されることが分かる**

  よく分からないのでAIに聞いてみると：https://claude.ai/share/9f3f8f48-fc4e-4e20-b734-524c3df13a80

- **しかし、どんな回路が動作してこの機能が実現されるのか分からない**

  もうすこしデータシート全体、関連する部分を読む必要がある

SIO: GPIO_OUT_SET Register

Offset: 0x014

Description

GPIO output value set

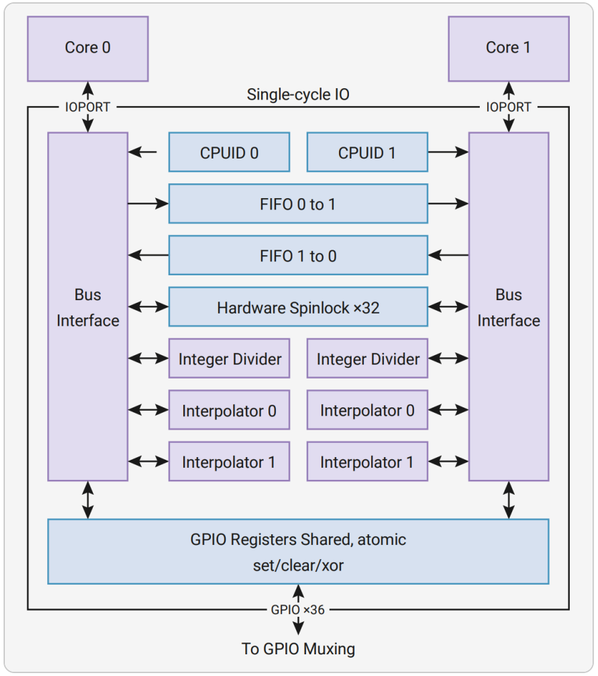
Table 21. GPIO_OUT_SET Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Perform an atomic bit-set on GPIO_OUT, i.e. `GPIO_OUT |= wdata`	WO	0x00000000

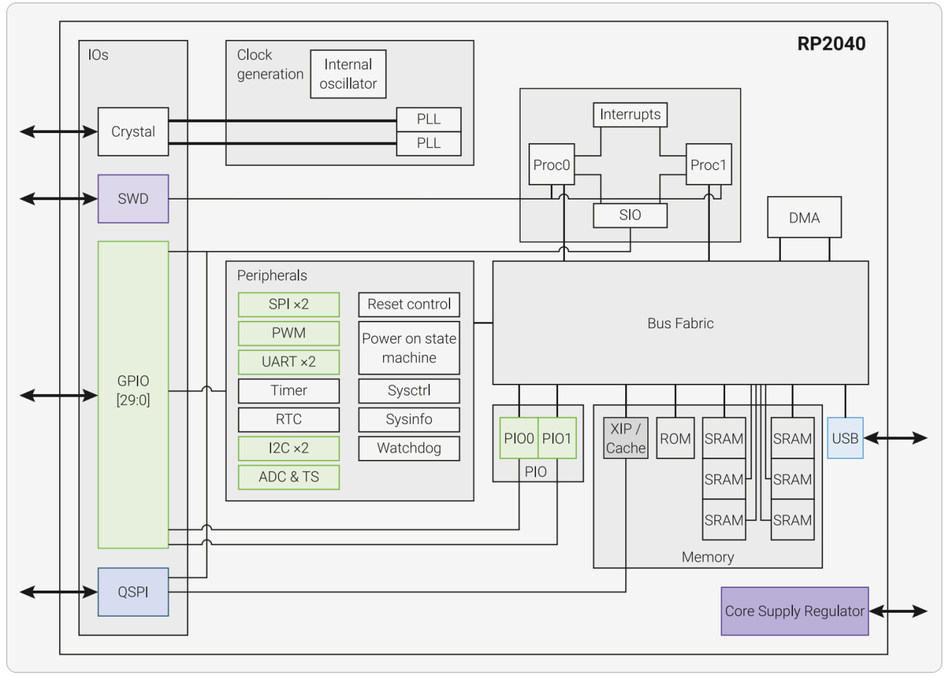
RP2040 データシート：GPIO_OUT_SET レジスタの説明

レジスタが所属するペリフェラル SIO を調べると、全体像がつかめる

マイコンの中で、CPU からどういう経路を通ってピンに出力されるのかが分かる。



SIO の内部 — 共有 GPIO レジスタ（atomic set/clear/xor）→ ピン



RP2040 全体 — SIO は2つのコアと GPIO の間に位置する

CHAPTER 04

原理から「手札（解空間）」を広げる

「動いた」で立ち止まらず、原理を基点に関連する手札を洗い出すエンジニアの姿勢を学ぶ

これまでの原理
32個のスイッチが並んだレジスタの1ビットを操作

→

疑問・仮説
複数いっぺんに操作できるのでは？

→

解空間の探索
SDKのドキュメントやAIで洗い出す

- 原理を知っていると、手札が増える

SDK ドキュメントを見ると、他にも関数が見つかる

- **使った gpio_put の他にも、さまざまな便利な関数がありそう**

  書くソフトウェアの目的により、最適なものがあるはず

- **これが解空間を広げる、手札を増やすということ**

  AI に調べさせるにしろ、この姿勢は自分が持つ必要がある

```
static void gpio_xor_mask (uint32_t mask)
    Toggle every GPIO appearing in mask.

static void gpio_xor_mask64 (uint64_t mask)
    Toggle every GPIO appearing in mask.

static void gpio_xor_mask_n (uint n, uint32_t mask)
    Toggle every GPIO appearing in mask.

static void gpio_put_masked (uint32_t mask, uint32_t value)
    Drive GPIOs high/low depending on parameters.

static void gpio_put_masked64 (uint64_t mask, uint64_t value)
    Drive GPIOs high/low depending on parameters.

static void gpio_put_masked_n (uint n, uint32_t mask, uint32_t value)
    Drive GPIOs high/low depending on parameters.

static void gpio_put_all (uint32_t value)
    Drive all pins simultaneously.
```

SDK ドキュメントの関数一覧（抜粋）。gpio_put の周りに、まだ見ぬ関数が並ぶ。

SDK ドキュメントで見る（p.167〜）

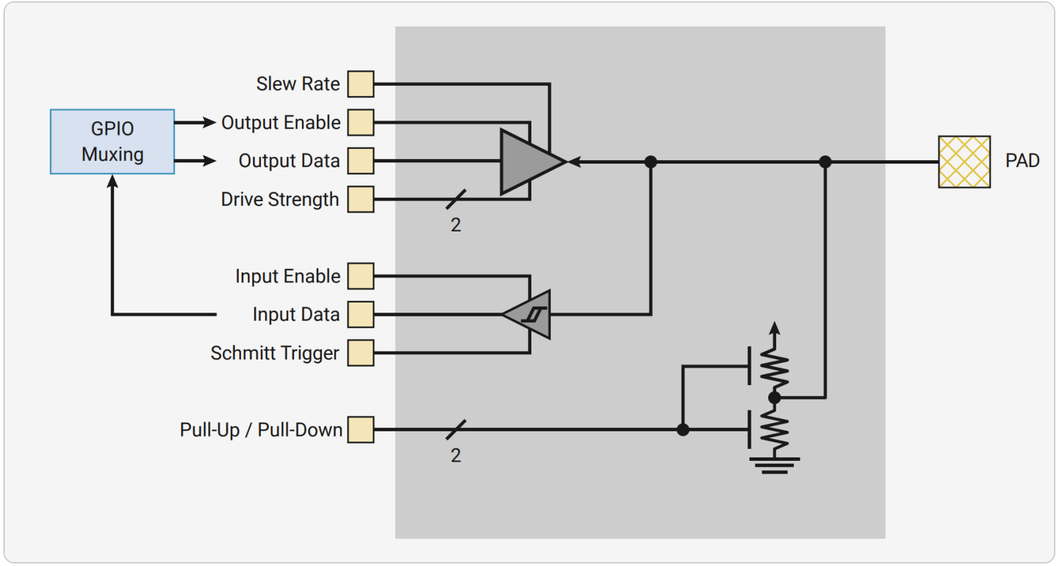
チップのデータシートを見ると、結局何が行われるのか分かる

- **SDKだけだと、ぼんやりと機能しか分からない**

  機能は分かるが、実際の動作はハードウェアのデータシートにある

- **解空間／手札の中で、最適なモノを探すには原理を知る必要がある**

  一番高速にピンを操作するにはどうすればいいか？
  高速に動かすために電気的に調整できるところはないか？



RP2040 データシートの GPIO 図。Drive Strength・Slew Rate など電気的な調整項目まで載っている。

RP2040 データシートで見る（p.241〜）

まとめ ／ 今日たどった往復

gpio_put の1行の正体を、SDKソースとデータシートまで根拠付きで説明できるようになった

① マイコンの仕組み
✓数値の読み書きが全てと分かった

→

② コードの深追い
✓Lチカで何のレジスタを書くか分かった

→

③ データシート
✓ レジスタからどう回路が動くか分かった

→

④ 手札を広げる
これは宿題やキャンプで実践

■ 4ステップを踏破 — 「動かす」から「原理を知って選べる」へ

完了条件は「AIコードを原理とデータシートで評価した、その判断と根拠」の提出

お題 ― これをそのままAIに投げる

「2台のラズピコを8本のGPIOで繋いで、片方から片方へデータを1byteずつ送受信するコードを pico-sdk で書いて」

手順1：そもそもどんな実装の選択肢があるか把握する

自分でデータシート・ドキュメントを見てもよいが、ここも AI と一緒に調べてよい。

手順2：その中で良さそうな選択肢を自分で選ぶ

宿題では明確に要求・制約を出していないので、選ぶ理由は自分なりに決めてよい（ただし説明できること）。
例：めっちゃ速くデータを送れる／コードが読みやすく分かりやすい …

5.2 次回予告：Week 2

マイコンとセンサーをつなぐ方法を学ぶ

Week 2：ハードウェア間通信プロトコル

- **今日の宿題が出発点**

  パラレル通信の課題を認識し、シリアル通信を学ぶ。

- **「通信プロトコル」の捉え方**

  規格ごとのルールだけではなく、情報伝達のための通信規格のレイヤーから理解する。

- **模擬衛星で使う通信規格**

  UART・SPI・I2C のコードを書けるようになる。