WEEK 2 ／ UART・SPI・I2C

ハードウェア間通信プロトコル

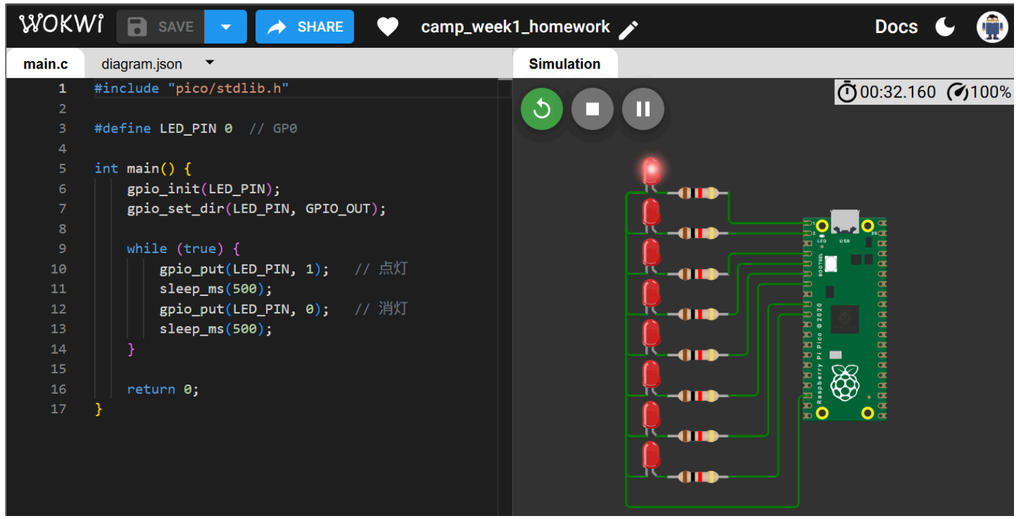
マイコンの中の数値の読み書きから、マイコンと部品間の情報伝達(数値の読み書き)へ。
データが流れるシーケンスと、そのハードウェアの仕組み・ソフトウェアとしての機能を学ぶ。

電子回路の基礎とマイコンの原理が分かっていればOK

- **補講（電子回路）：**H/L のデジタル信号、MOSFETという電気のスイッチ
- **Week 1：**マイコンの動作は数値の読み書きの繰り返し。
  特殊な周辺回路(ペリフェラル)もソフトから見れば数値の読み書き。
- チップとチップの間で情報(数値)をやりとりする方法を学ぶ。

8本の信号線でデータを送ることはできるが…

- **Week 1 の宿題：**
  2台の Pico を8本の GPIO でつなぎ、1バイトずつ送信するコードを書いた。
- **面倒だったこと：**
  配線が多い。データを読むタイミングを合わせるのが難しい。
- **今日学ぶこと：**
  実際の部品間の通信規格はどうなっているのか？

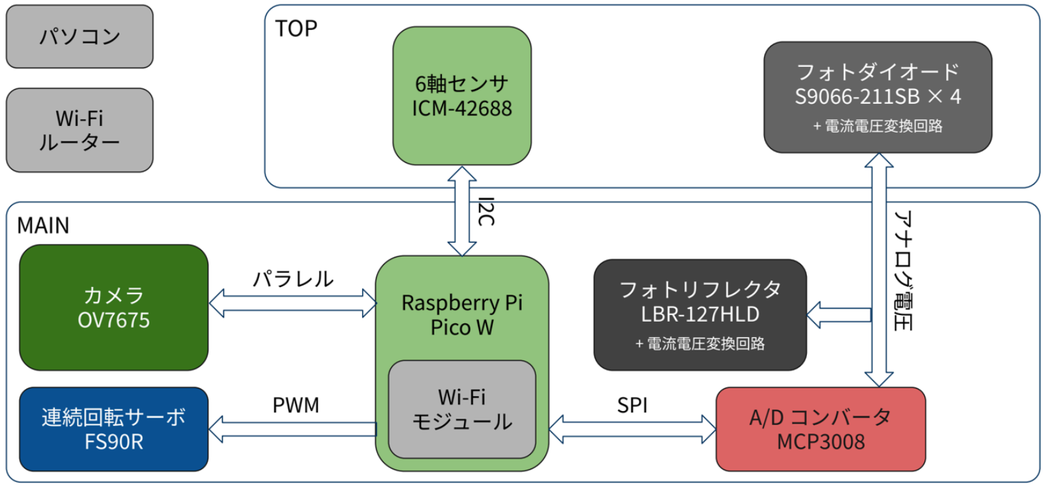


前回の宿題の Wokwi プロジェクト
Pico は1台しか置けないため、送る1バイトを LED 8個で表示して確かめた

0.3 今回の位置づけ

模擬衛星は Pico W・IMU・ADC・カメラという別々のチップの集まり

- **構成：**Pico W、IMU、ADC、カメラ。どれも独立したチップ
- 制御を行うにはこれらのチップと通信する必要がある。
- そのルールである**通信規格**が今日のテーマ。



模擬衛星のシステム構成。マイコンのまわりに独立したチップが並ぶ

補講
電子回路の基礎
→
Week 1
マイコンの基礎
→
Week 2（今日）
ハードウェア間通信規格
→
Week 3
センサとの通信と波形観測
→
Week 4
リアルタイムシステム
→
Week 5〜
エンジニアリングの方法

第 1 章

シリアル通信をレイヤーで捉える

- センサ値やコマンドという**「伝えたい情報（数値）」**がある
- 用途に合わせ、**UART・SPI・I2C** など、1層・2層の規格が存在する

どの層で壊れたかを先に決めて、調べる先を絞り込む

- 「動かない」を「どの層が壊れたか」に**絞り込む**のがデバッグの第一歩
- Week 3 の波形観測で、この切り分けを実際にやる

層で分ける発想は、TCP/IP や OSI 参照モデルとして体系化されている

- 層で分ける発想は、通信の世界全体で使われている戦術
- インターネットでは **TCP/IP** や **OSI 参照モデル**という体系に
- デバッグ時に問題部分を絞り込むという目的だけでなく、**責任を分離した適切な設計**として非常に重要
- 詳細は補講（Wi-Fi 通信）で扱う。今日は2階建てで十分

第 2 章

なぜ通信規格が複数あるのか

- データを運ぶだけなのに、パラレル・UART・SPI・I2C と複数の規格がある。
- ピン数・速度・通信相手の数のトレードオフ。

たくさんピンを使ってデータをたくさん運ぶ

メリット

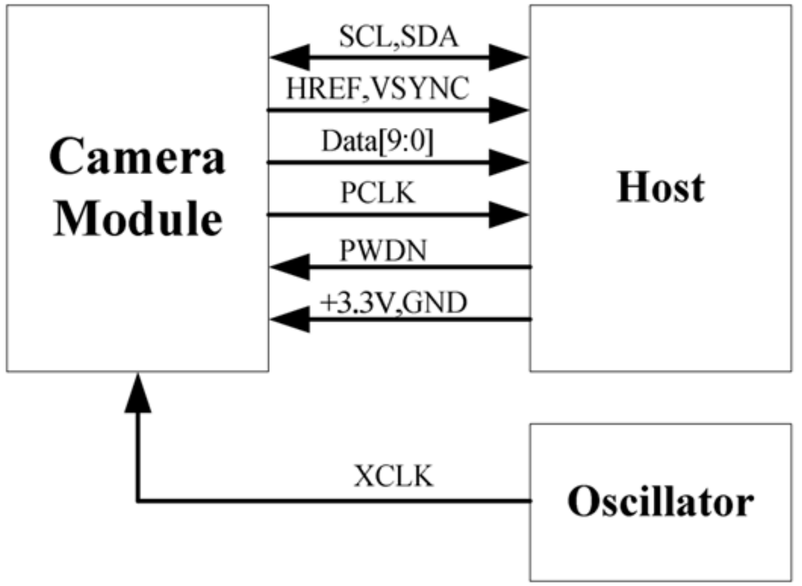
高速。8本のデータ線に1ビットずつ、1回の合図で1バイト転送可能。

デメリット

ピン数を大量に消費する。マイコンのピン数は有限。

模擬衛星での採用例

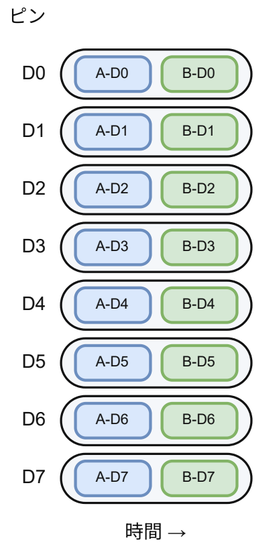
カメラ：D0〜D7 の8本＋同期・クロック4本（PCLK・HS・VS・XCLK）



OV7675 カメラのデータバス（データシートより抜粋）。多数のデータ線を束で使う

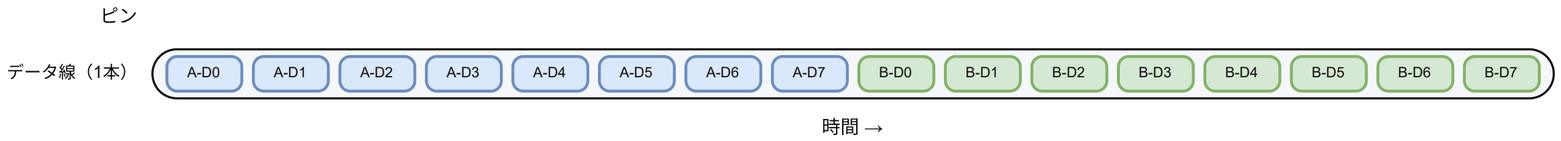
1本の線にビットを時間で区切って順に流し、ピンを抑える

パラレル（8本の線）



速いがピンを8本消費
2バイト＝**2クロック**

シリアル（1本の線）



遅いがピンは1本 ／ 同じ2バイト＝**16クロック**

- 1本のデータ線に、ビットを時間で区切って1つずつ流す
- 同じ量なら遅くなる。代わりに**ピンを大幅に節約**
- UART・SPI・I2C はすべてシリアル

第 3 章

個別規格の前に：共通の語彙

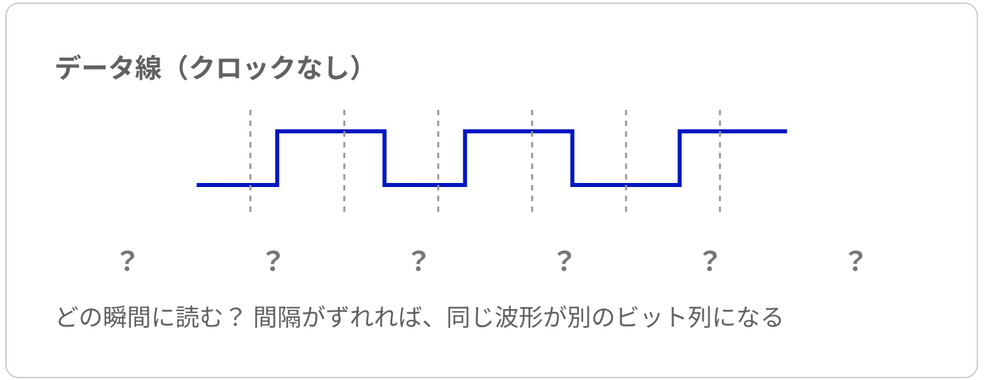
- 誰が通信を主導するか
- 通信線の H/L から、データをいつ読むか

主導権を1つに固定すると、送信の衝突を設計で防げる

- **マスタ**：会話を開始し進行を仕切る側。模擬衛星では Pico
- **スレーブ**：求められたときに応じる側。センサや ADC
- 主導権を1つに固定すると、衝突を設計の段階で防げる
- **注記**：コントローラ／ターゲットという言い換えも

データ線だけでは、いつ H/L を読めばよいか分からない

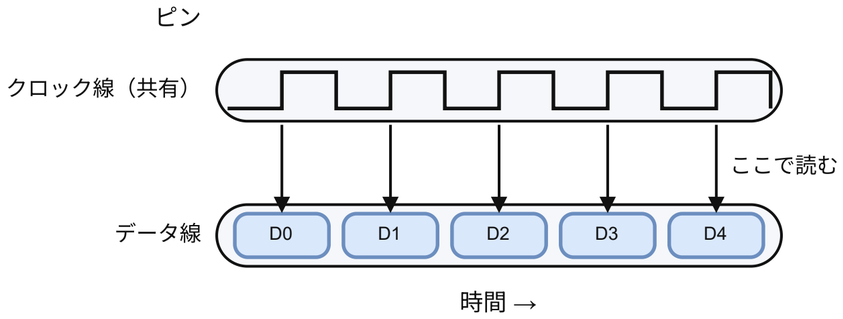
- 送り手は H と L を送るが、受け手は**「いつ読むか」**を知らない
- 同じ波形でも、読むタイミング次第でビット数も値も変わる
- だから読む瞬間をそろえる基準＝**クロック（一定の拍子）**が要る
- 次で、その合わせ方（同期・非同期）を見る



同期式（SPI・I2C）はクロックの合図で、非同期式（UART）は速度の事前合意で

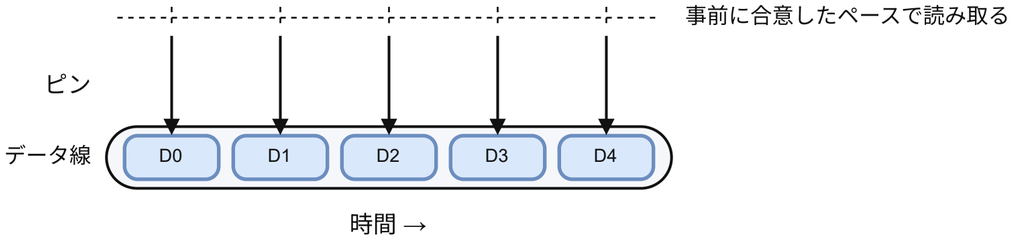
同期式（SPI・I2C）

- クロック線を追加する



非同期式（UART）

- ペースを事前合意する

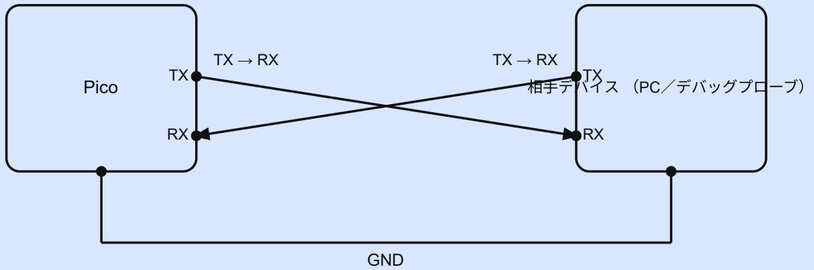


第 4 章

UART

Universal Asynchronous Receiver / Transmitter

クロック線なしで、事前に合意した速度で送り合う非同期の1対1。

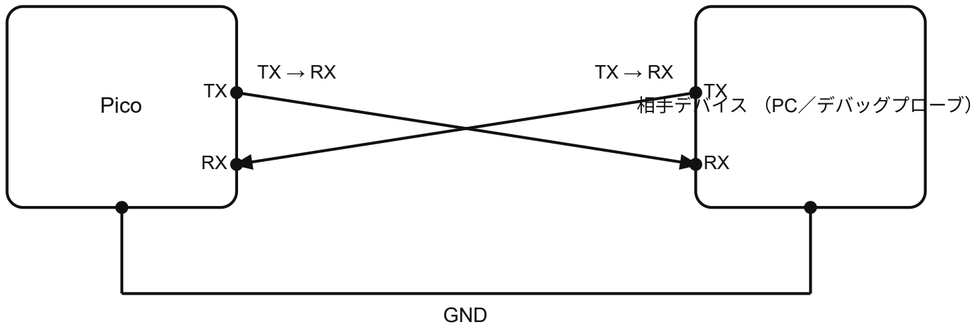


※ 行きと帰りが別の線（全二重）

TX と RX をたすき掛け（クロス）でつなぐ1対1

TX は相手の RX へたすき掛け。TX 同士が定番の配線ミス

- **TX（送信）と RX（受信）**の2本。それぞれ一方通行
- 自分の TX は相手の RX へ。**たすき掛け（クロス）**
- 行きと帰りが別の線なので、双方向に同時に送れる（**全二重**）



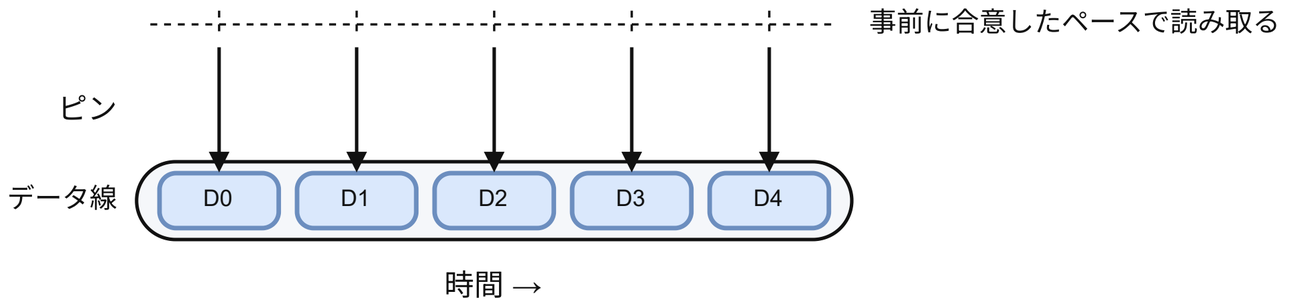
※ 行きと帰りが別の線（全二重）

自分の TX を相手の RX へ、たすき掛けにつなぐ。GND は共通

同じボーレートにして初めて成立する

- **ボーレート**
  1秒間に送るビット数の約束
  単位は **bps**（bit per second）

- **速度は事前設定**
  クロック線を張らない代わりに、テンポを事前に決めておく
  受け手はこの約束を頼りに、自分の時計で数えながら読む

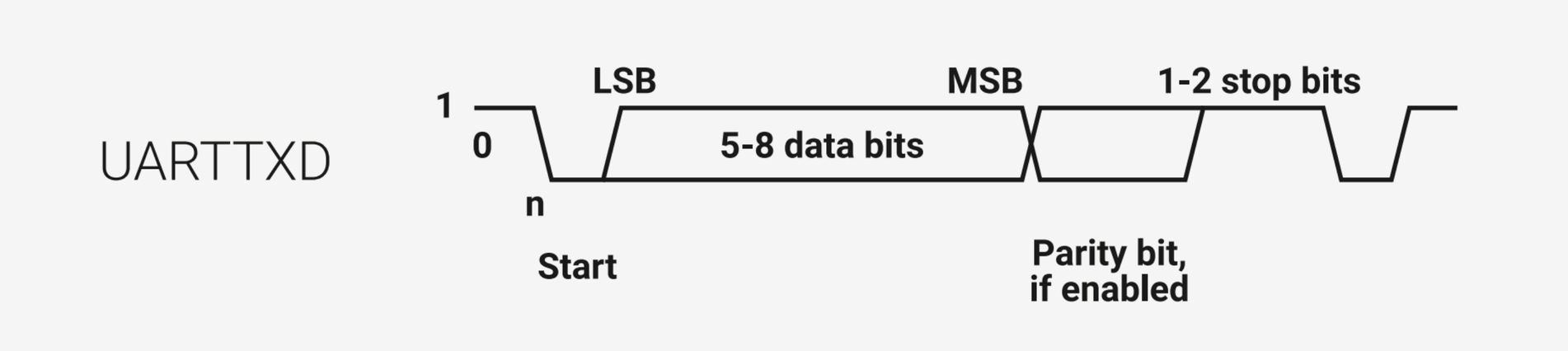


クロック線はなく、合意した 115200 の目盛りで各自が数えて読む

シリアルモニタでも同じ 115200 を選んで速度合意に参加する

次へ → ペースは一定でも、データの開始タイミングを知る必要がある

データビットを送るための"フレーム"



RP2040 データシートの UART フレーム

- **スタートビット**：待機中の信号線は H。L に切り替えた瞬間が送信開始の合図
- **データビット**：5〜8bit のデータをまとめて送る。通常は 8bit
- **パリティビット**：通信データが化けていないか検算するビット。あまり使わない
- **ストップビット**：通信終了を示すため H に戻す。通常は 1bit

ソフトウェアからは uart_write_blocking から叩く。その中身は？

```
uint8_t buf[] = { 'H', 'i' };
uart_write_blocking(uart0, buf, 2);  // uart0 に 2 バイトを送る
```

- 最も簡単なのが **uart_write_blocking**（バイト列をそのまま UART へ）
- Week 1 で gpio_put を追ったのと同じように、仕組みを調べてみる
- （参考）**HAL**：Hardware Abstraction Layer

5.1.32.9.26. uart_write_blocking

`static void uart_write_blocking (uart_inst_t * uart, const uint8_t * src, size_t len) [inline], [static]`

Write to the UART for transmission.

This function will block until all the data has been sent to the UART transmit buffer hardware. Note: Serial data transmission will continue until the Tx FIFO and the transmit shift register (not programmer-accessible) are empty. To ensure the UART FIFO has been emptied, you can use uart_tx_wait_blocking()

Parameters

`uart`	UART instance. uart0 or uart1
`src`	The bytes to send
`len`	The number of bytes to send

どういうこと？

この関数は、すべてのデータが UART 送信バッファ・ハードウェアに送信されるまで処理をブロックします。

注：シリアルデータの送信は、Tx FIFO および送信シフトレジスタ（プログラマからは直接アクセス不可）が空になるまで継続します。UART FIFO が空になったことを確認するには、`uart_tx_wait_blocking()` を使用できます。

FIFO に数値を書き込むまでがソフトの仕事、後の通信はペリフェラルの仕事

① 送信したいデータ
HAL がUARTのペリフェラルに数値を書く

→

② Tx FIFO
送信の順番待ちの列に貯める

→

③ 送信シフトレジスタ
フレームを組み立てる

→

④ UARTTXD（物理的なピン）
1ビットずつデータが送信される

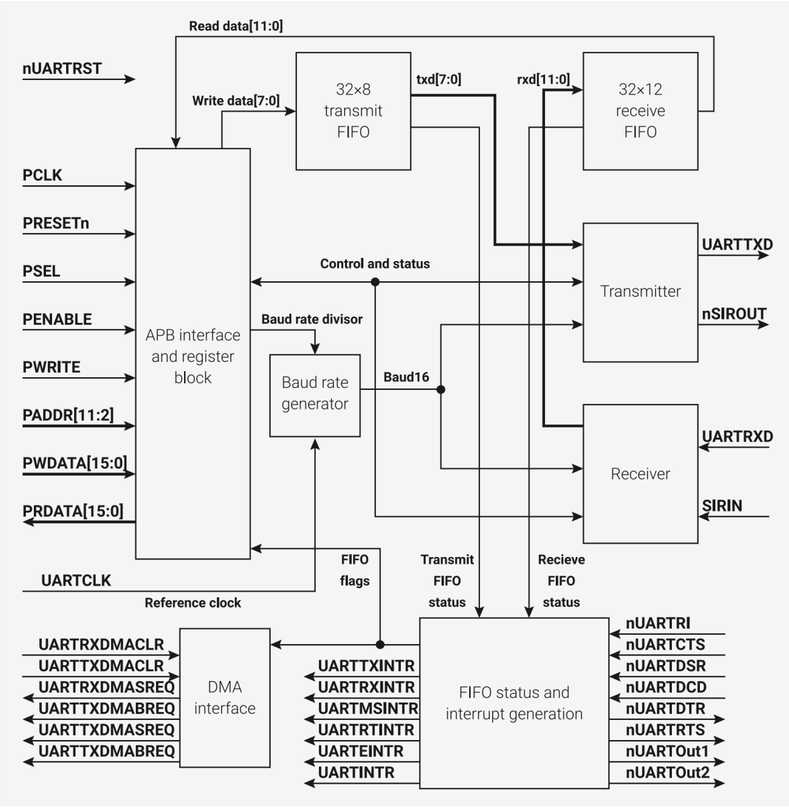
- **ソフトとハードの境界を知らないと良いソフトは書けない。**

SDKの記述：

この関数は、すべてのデータが UART 送信バッファ・ハードウェアに送信されるまで処理をブロックします。

注：シリアルデータの送信は、Tx FIFO および送信シフトレジスタ（プログラマからは直接アクセス不可）が空になるまで継続します。UART FIFO が空になったことを確認するには、`uart_tx_wait_blocking()` を使用できます。

FIFO（First In, First Out）＝ 順番待ちの列



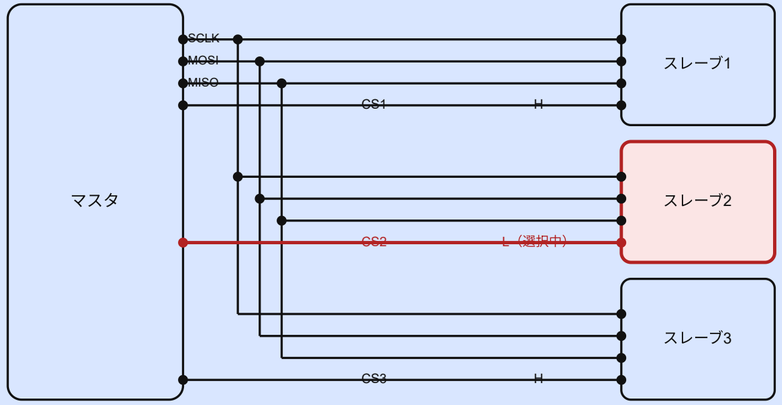
RP2040 データシート UART 機能ブロック図。32×8 送信 FIFO・Transmitter・Baud rate generator

第 5 章

SPI

Serial Peripheral Interface

クロックを共有する同期式。CS で相手を1つ選び、1対多にもできる。

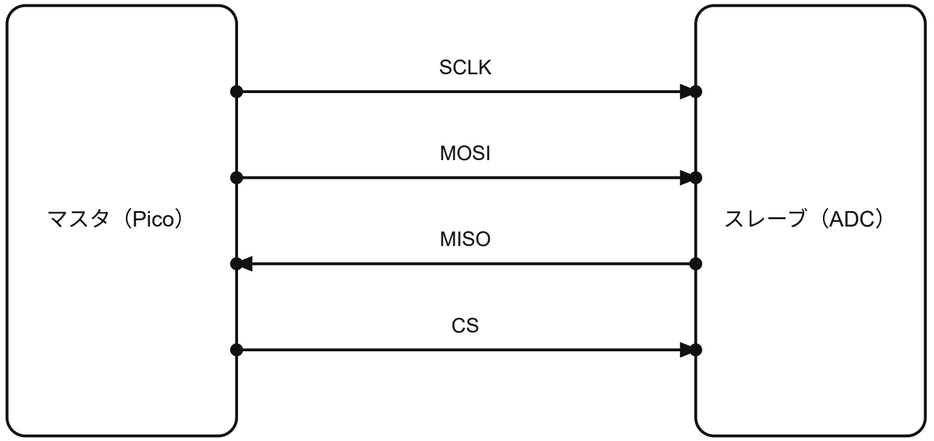


※ 共有3本＋相手ごとの CS

3本を共有し、CS は相手ごとに1本。選ばれた1台と通信

名前が信号の向きを示す。MOSI 同士・MISO 同士をつなぐ

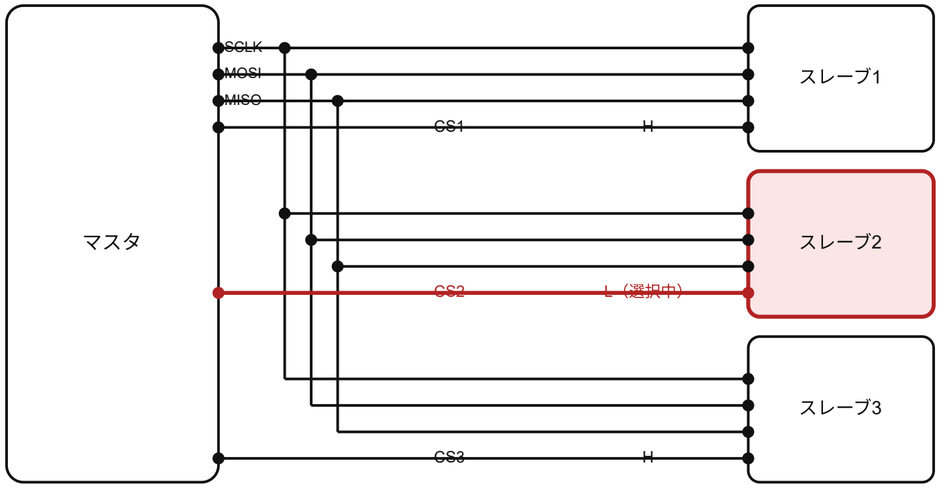
- **SCLK**：Serial Clock。マスタがクロックを決める
- **MOSI**：Master Out Slave In。マスタ → スレーブ
- **MISO**：Master In Slave Out。スレーブ → マスタ
- **CS**：Chip Select。通信相手を選択する



SCLK・MOSI・CS はマスタから、MISO はスレーブから

3本は共有、CS を L にされた1つのスレーブが通信に応答する

- スレーブが複数でも、SCLK・MOSI・MISO の3本は全員で共有
- CS だけは相手ごとに1本ずつ。L にした相手だけが参加
- **デメリット：**相手が1つ増えるたびに CS が1本増える



※ 共有3本＋相手ごとの CS

3本は共有、CS は相手ごとに1本。CS2 だけが L で選択中

"モード"という設定もある。

- **CPOL（極性）**：クロックが休んでいるときＨかＬか
- **CPHA（位相）**：クロックのどのエッジでデータを読むか
- 組み合わせで4モード。部品のデータシートが指定する。

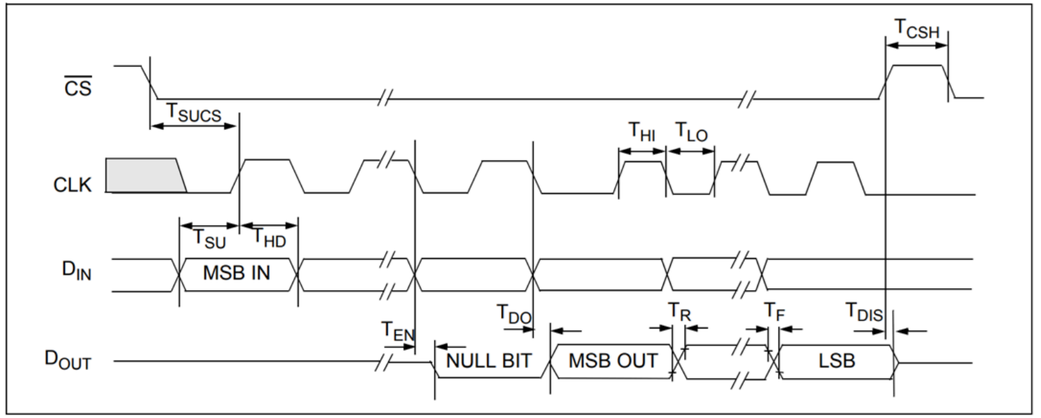
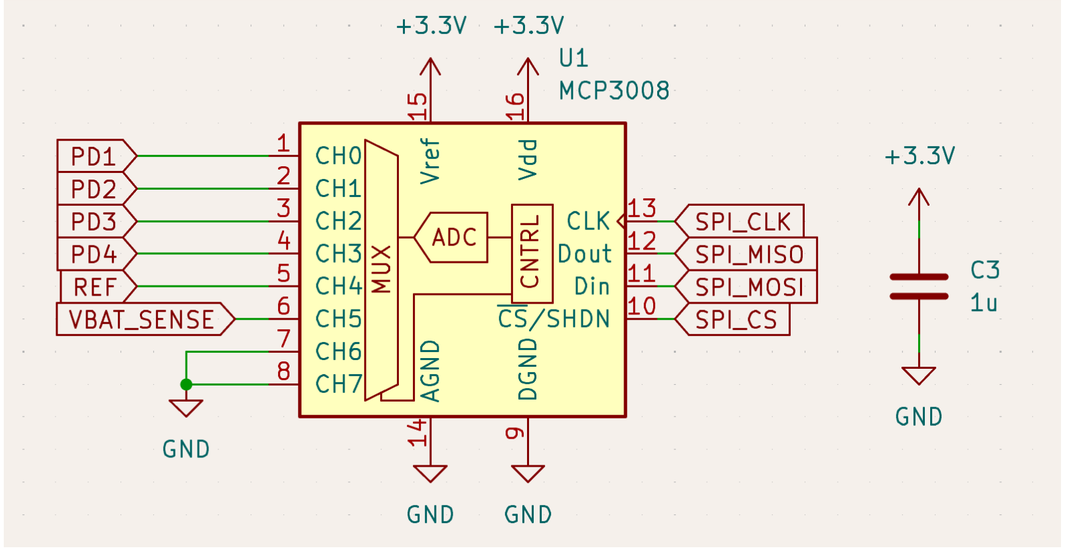


FIGURE 1-1: *Serial Interface Timing.*

MCP3008 データシートの SPI タイミング図（抜粋）

回路図に SPI_CLK・MOSI・MISO・CS の4本がある

- **MCP3008**：明るさセンサ・回転数計のアナログ値をデジタルに変換
- 回路図のネット：SPI_CLK・SPI_MOSI・SPI_MISO・SPI_CS の4本



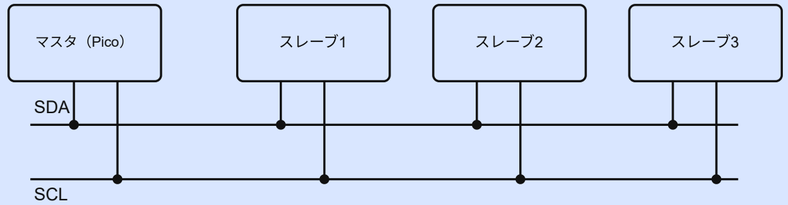
模擬衛星の回路図の MCP3008 まわり。SPI の4本のネットが見える

第 6 章

I2C

Inter-Integrated Circuit

SDA・SCL の2本だけを全員で共有し、
相手は通信データの先頭に置くアドレスで選ぶバス方式。

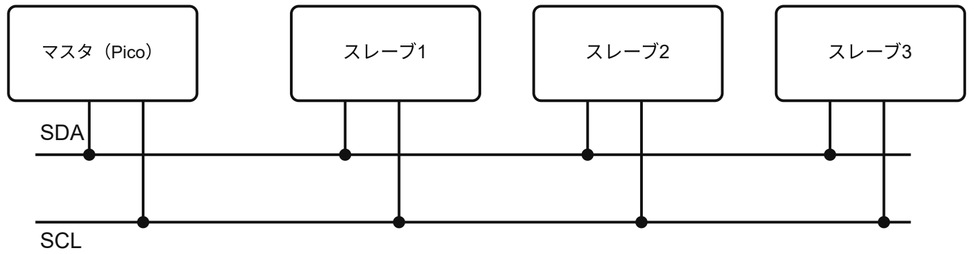


※ 線は全員で共有（バス）

SDA・SCL の2本に、マスタとスレーブ全員がぶら下がる

1本の線を全員で共有しても壊れないのはなぜ？

- SDA・SCL は、マスタもスレーブも全員がぶら下がる共有線（**バス**）
- 普通の出力（H も L も出せる）どうしだと、H と L がぶつかった瞬間に**電源と GND が綱引き**
- 綱引き＝大電流。ピンが壊れかねない危険な状態
- I2C は**出力の形**を工夫し、この衝突を原理的に防ぐ（次で回収）

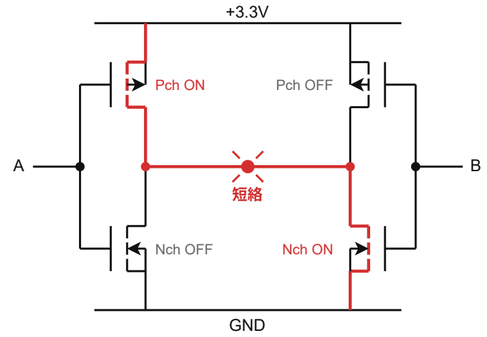


※ 線は全員で共有（バス）

SDA・SCL の2本に、マスタとスレーブ全員がぶら下がる

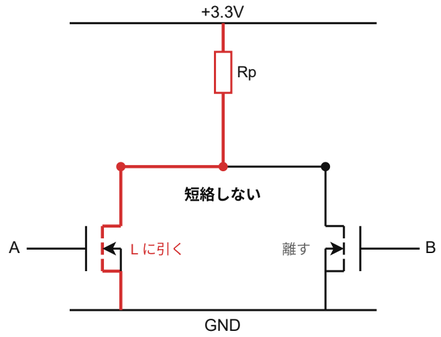
衝突すると短絡するプッシュプルを避け、L に引くだけのオープンドレインを使う

✕ プッシュプル（普通の出力）



H と L がぶつかると、電源→GND が直結して**短絡**。大電流でピンが壊れる

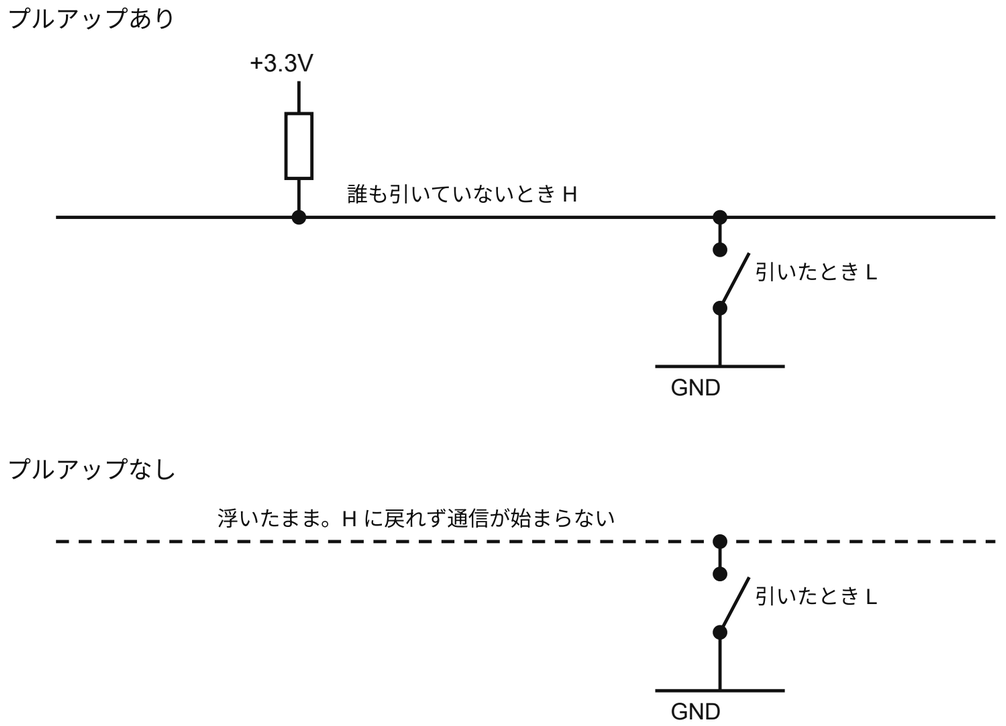
○ オープンドレイン（I2C が採用）



L に引くだけで H を出せない。綱引きが起きず、短絡しない

L にしか引けないから、線の H は外部プルアップ抵抗が作る

- オープンドレインは L に引くだけ → 待機時の H は自然には出てこない
- **外部プルアップ抵抗**が線を電源側へ弱く引き、待機状態を H にする
- 「弱く」がポイント：誰かが L に引けば、そちらが勝つ（**ワイヤードAND**）。抵抗経由なので電流も小さく安全
- 抵抗を忘れると線は H に戻れず通信不能。内蔵プルアップは設定依存の救済、規格の約束は**外部プルアップ**



プルアップがあれば放置で H、外すと線は浮いたまま H に戻れない

基本は競合しない。ただしマルチマスタや ACK では競合が起こりうる

通常時は競合しない

全員が SCL のクロックに同期し、ある瞬間に線を駆動するのは基本1台だけ。話す順番が決まっているので、普段はぶつからない。

マルチマスタ：アービトレーション

マスタが複数いて同時に開始しても、ワイヤードANDで**L が勝つ**。自分は H を出したのに線が L なら負けと検知し、静かに降りる。勝った側の通信は壊れない。

ACK / NACK の送信タイミング

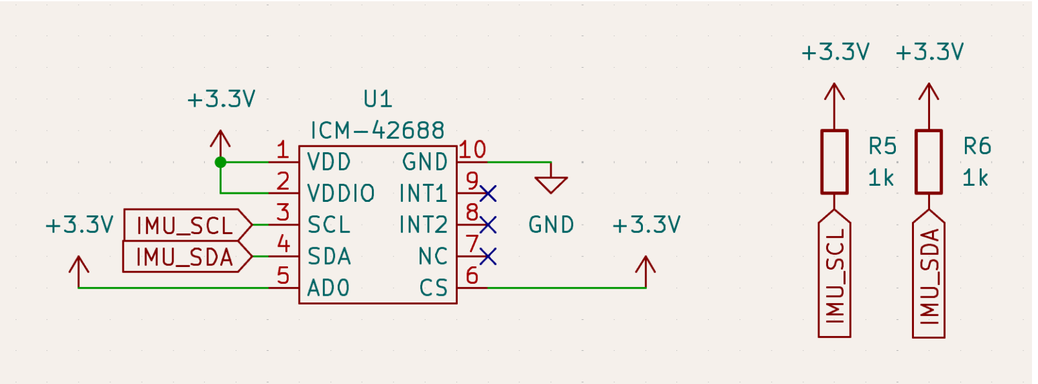
各バイトの**9クロック目**だけ、受信側が SDA を L に引けば ACK、引かなければ NACK。送信の主が一瞬入れ替わる、決められた受け渡し。

クロックストレッチ

スレーブが **SCL を L に保持**し続け、処理が追いつくまでマスタを待たせる。合図の線を止めて時間を稼ぐ。

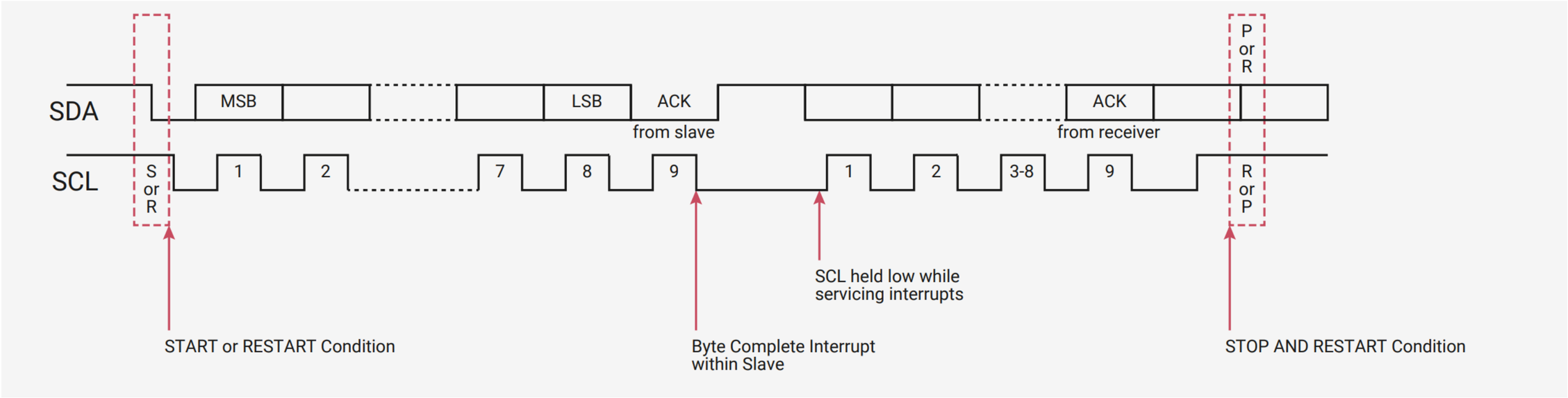
IMU の SDA・SCL に、1kΩ プルアップ R5・R6 が実装されている

- IMU（ICM-42688）は I2C 接続：ネット名 IMU_SDA・IMU_SCL
- R5・R6（1kΩ）が +3.3V へつながる。これがプルアップ抵抗
- 「外部プルアップ必須」の約束が、部品2個として写っている
- 逆読み：回路図でプルアップを見たら共有バスかもしれない



模擬衛星の IMU まわりの回路図。SDA・SCL の 1kΩ プルアップが見える

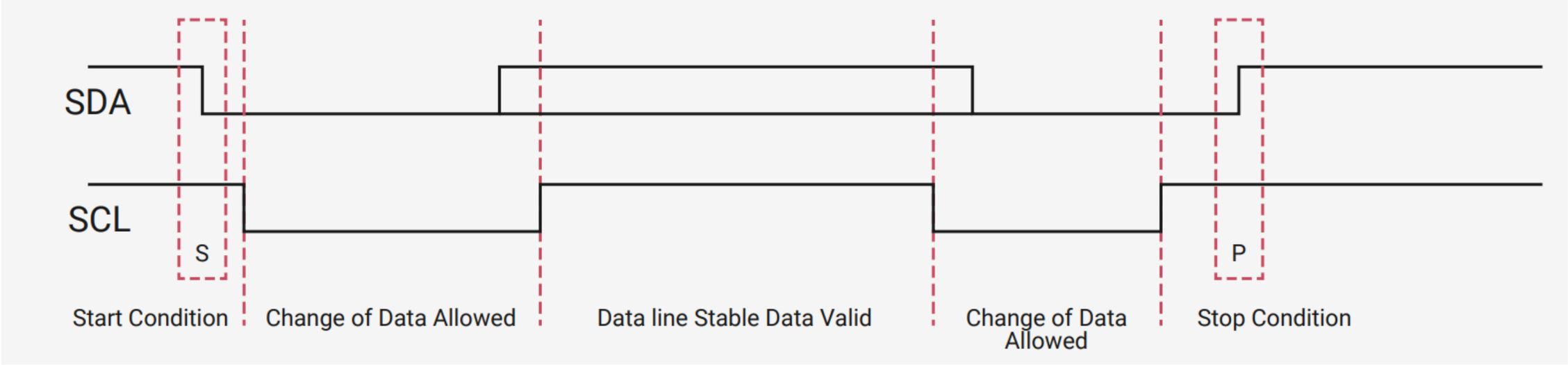
転送はアドレス→ACK→データ→ACK の往復で進む



I2C バス上のデータ転送の実波形（RP2040 データシート）。アドレスに ACK が返り、データが続く

- **約束：**1バイト受け取るごとに受け手が ACK（受け取った）を返す
- **NACK**（返事なし）＝相手がいない・名前が違う・相手が不調
- 転送はアドレス→ACK→データ→ACK の往復で進む

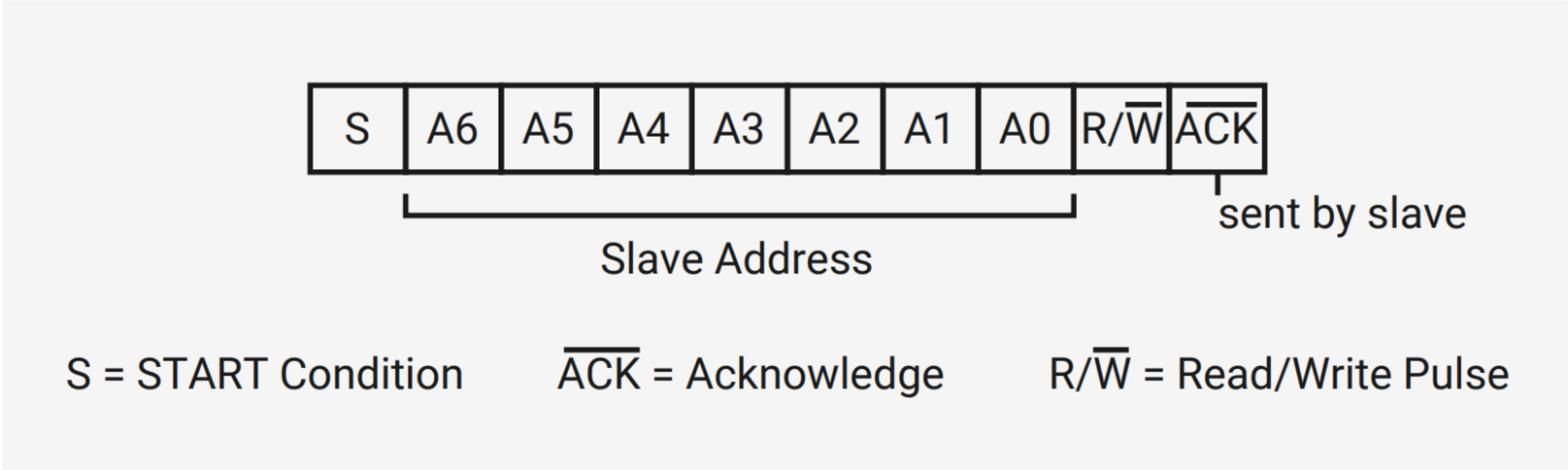
SCL が H の間に SDA を動かす例外が、開始・終了の合図になる



START・STOP コンディション（RP2040 データシート）。S・P の合図と、データを変えてよい区間

- 通常ルール：SDA を変えてよいのは SCL が L の間だけ
- **START**＝SCL が H のまま SDA が H→L ／ **STOP**＝L→H
- 通常データでは起きない動きなので、会話の切れ目を全員が見分ける

会話の先頭に宛先を流して選ぶ。相手選択を意味の層で解く



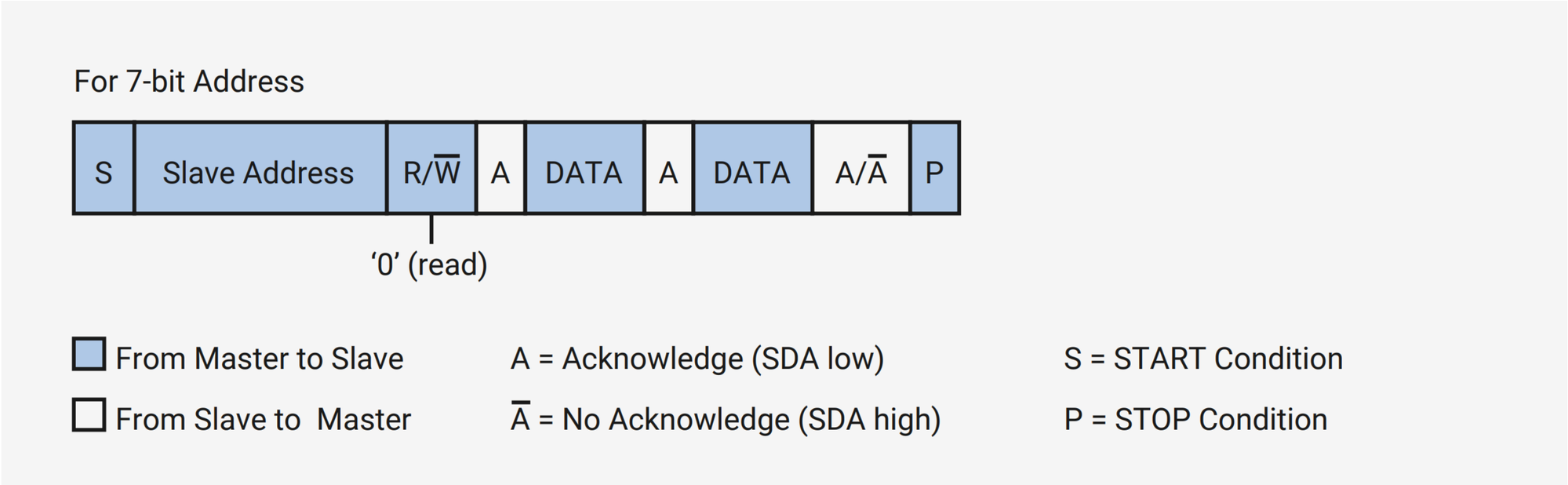
7ビットアドレスのフォーマット：S・A6〜A0・R/W・ACK（RP2040 データシート）

- 各スレーブは固有のアドレス（7ビットが代表）を持つ
- START の直後にアドレスが流れ、自分宛のスレーブだけが ACK を返す
- アドレスの直後の1ビットが、読み書きの向き（R/W）

同じ「相手の選択」でも

SPI＝電気の層（専用線 CS）／I2C＝意味の層（宛先ビット）

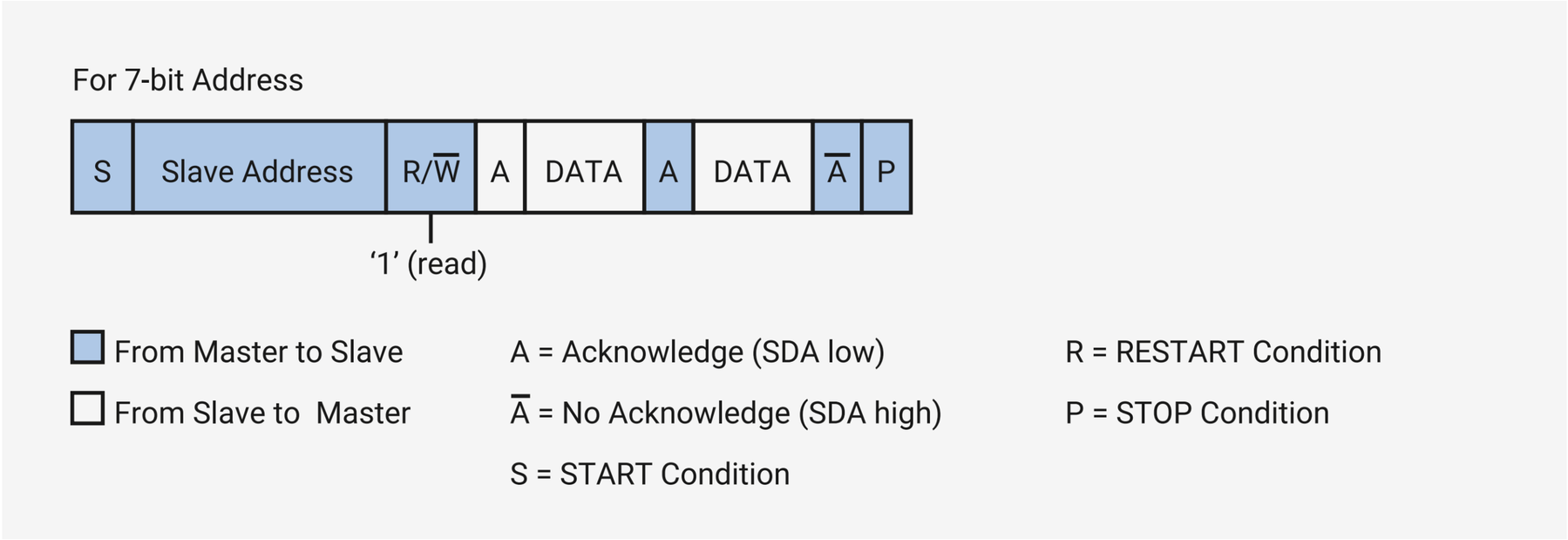
アドレスとデータをマスタが送り、1バイトごとの ACK をスレーブが返す



書き込み（Master-Transmitter）の手順（RP2040 データシート）。A=ACK・Ā=NACK

- S → アドレス＋向き（書き込み）→ ACK → データ → ACK → … → P
- 図の塗り分け：塗り＝マスタ→スレーブ、白＝スレーブ→マスタ
- データはすべてマスタ側、ACK だけがスレーブ側

データの向きが逆になり、最後は NACK で「終わり」を伝える



読み出し（Master-Receiver）の手順（RP2040 データシート）。最後の1バイトだけ NACK（Ā）

- S → アドレス＋向き（読み出し）→ ACK → データ → … → 最後は NACK → P
- 塗り分けが反転：データは白（スレーブ→マスタ）、ACK は塗り（マスタ→スレーブ）
- 最後の NACK は故障ではなく「これで終わりにする」の合図

第 7 章

まとめ：規格の整理と模擬衛星での利用

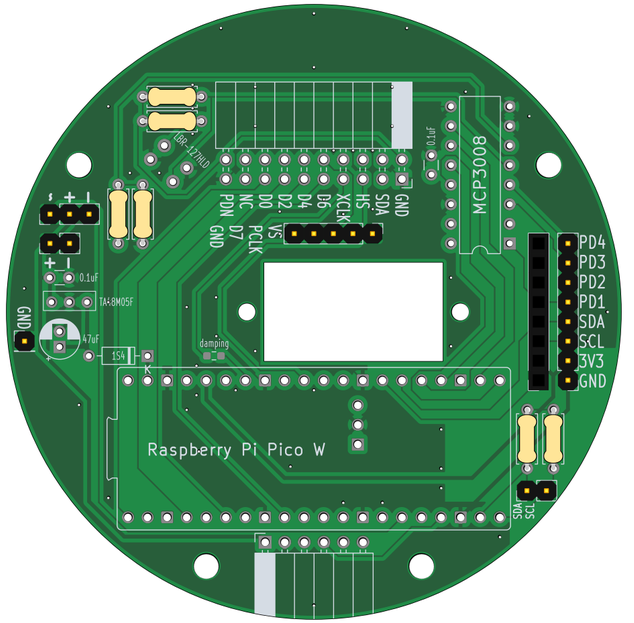
今日学んだ規格を整理し、模擬衛星の基板でどこにどれが使われているかを確認する。

暗記や選定の道具ではなく、配線・実装で迷ったときの索引

規格	線の数	タイミング	相手の数と選び方	模擬衛星での役割
UART	2（TX/RX）	非同期（ボーレート合意）	1対1	デバッグ出力（シリアルログ）
SPI	4＋相手が増えたら CS 追加	同期	複数。CS の線で選ぶ	ADC（MCP3008）
I2C	2（SDA/SCL）	同期	複数。アドレスで選ぶ	IMU、カメラの設定
カメラのパラレルバス	8＋同期線	同期	1対1	カメラの画像データ

IMU=I2C、ADC=SPI、デバッグ=UART、カメラ=パラレル＋I2C

- **IMU：I2C**（IMU_SDA・IMU_SCL）。少量を2本で。プルアップ実装済み
- **ADC：SPI**（SPI_CLK・MOSI・MISO・CS）。4ch の周期的な読み出し
- **デバッグ：UART**（TX・RX）。PC と1対1。今日その下まで追った道
- **カメラ：**画像はパラレル12本、設定は I2C（CAM_SDA・CAM_SCL）の併用

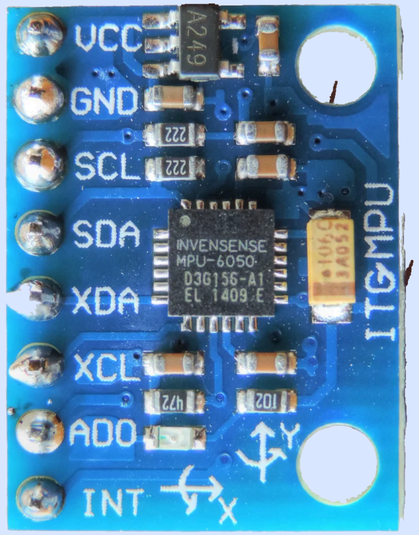


模擬衛星のメイン基板。今日の4規格が同居している

第 8 章 次回予告：Week 3 センサとの通信と波形観測

MPU6050 を題材に、存在確認・設定・読み出しの3段で通信する

- 題材は IMU（MPU6050）。初見のセンサと通信を始める「3段の型」
- **①存在確認 ②設定 ③データ読み出し。**センサが変わっても通用する型
- 見えない信号を波形として観測（ロジックアナライザとオシロ）
- 正常な波形を見た後、その場で壊す。今日の層の視点で切り分ける



Week 3 の題材になる MPU6050 搭載モジュール