

WEEK 3

センサとの通信とデバッグ

6軸 IMU「MPU6050」を AI のコードで動かし、AI が注意した2箇所（プルアップ・アドレス）をわざと間違えて、問題を特定するプロセスを身につける。

■ 動かす

AI と公式サンプルから始める

■ 間違える

注意書きの2箇所を踏み外す

■ 特定する

二分探索・観測（printf / オシロスコープ / ロジックアナライザ）・対照実験

0.1 前回の復習

今日使うのは I2C

- **2本のバスに複数の相手。**

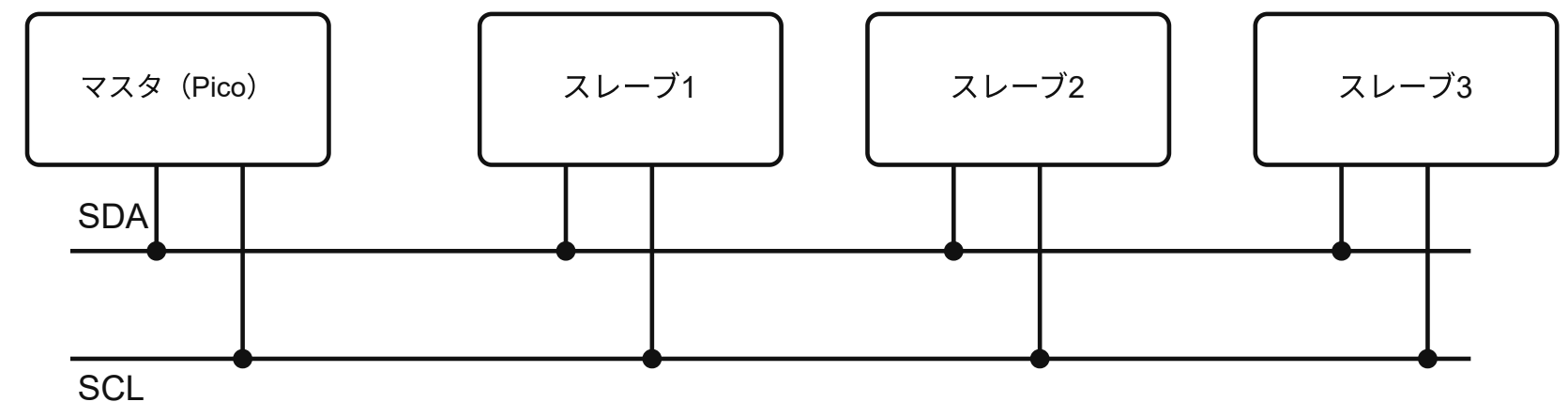
SDA（データ）と SCL（クロック）を全員で共有する。

- **アドレスで相手を選ぶ。**

応答は ACK（Acknowledge）で確認する。

- **バスの H はプルアップ抵抗が作る。**

オープンドレイン出力は L に引くことしかできない。



※ 線は全員で共有（バス）

I2C のバス構成（Week 2 の再掲）

センサと通信＝シリアル通信でセンサのレジスタを読み書き

Week 0

電子工作の基礎

Week 1

マイコンの原理・レジスタ

Week 2

通信の規格

Week 3

センサと波形観測

Week 4

リアルタイムシステム

第1段階「要素技術と根拠のある設計」(Week 1～4) — 今日はその3回目

- 合宿にもそのまま通用する。

模擬衛星の IMU (ICM-42688) が相手でも、今日の進め方は変わらない。

センサの中にも、番地の付いたレジスタが並んでいる

Week 1 の「レジスタ＝CPUと周辺回路との間の掲示板」というモデルは、通信の先にあるセンサの内部にもそのまま通用する。



変わるのは経路だけ — 「番地に値を読み書きする」という形は同じ

通信の仕事は、センサのレジスタへ「番地」と「値」を運ぶこと

- I2C の1回のやり取り＝依頼と応答。

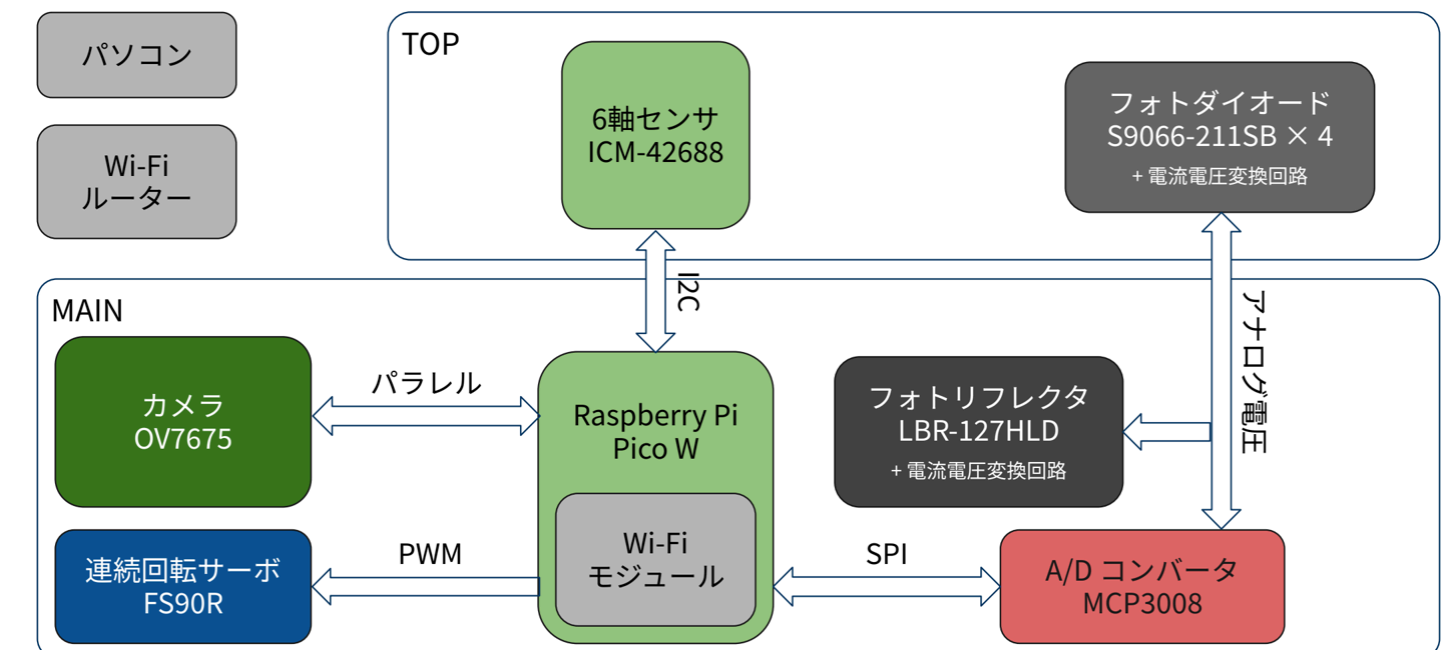
「どの番地を」「読む／書く」の依頼を運び、応答を受け取る。

- 模擬衛星も同じ。

IMU（ICM-42688）のレジスタは I2C で、ADC（Analog-to-Digital Converter）の MCP3008 は SPI で読み書きする。

- 運ばれているのは「番地と値」。

経路が何であれ、この形は変わらない。



模擬衛星のシステム構成図。ICM-42688 は I2C、MCP3008 は SPI で Pico W とつながる

とりあえず動かす — 2つの方法で「動くコード」を手に入れる

- 題材は MPU6050

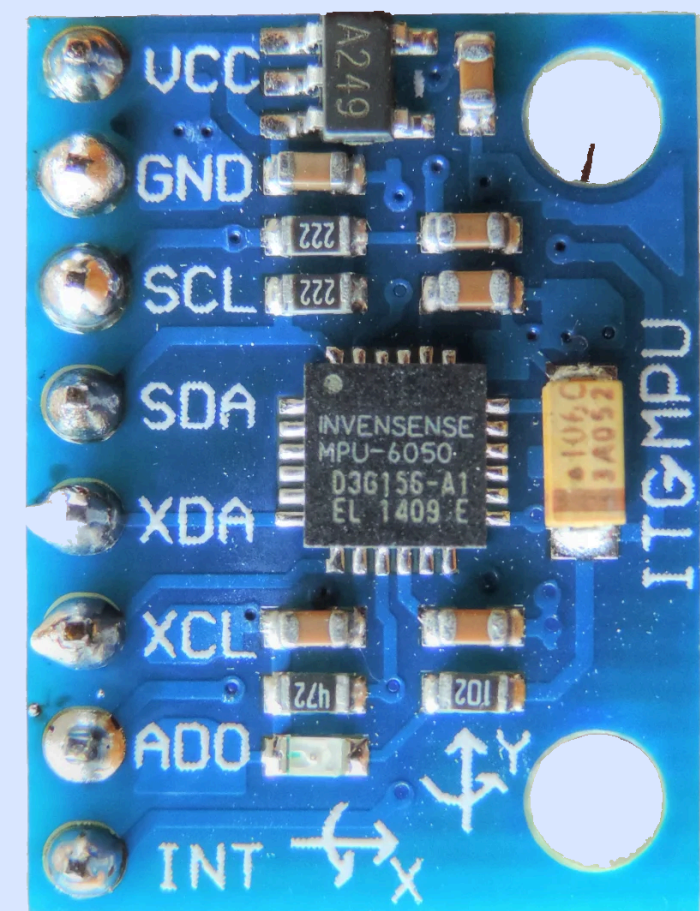
3軸ジャイロ+3軸加速度の6軸 IMU。資料とサンプルコードが豊富な定番。

- 方法は2つ

①：AI に書かせたコード。②：公式のサンプルコード。

- 模擬衛星搭載品とは別の型番

本番は ICM-42688 という別のセンサ。基本は同じ。

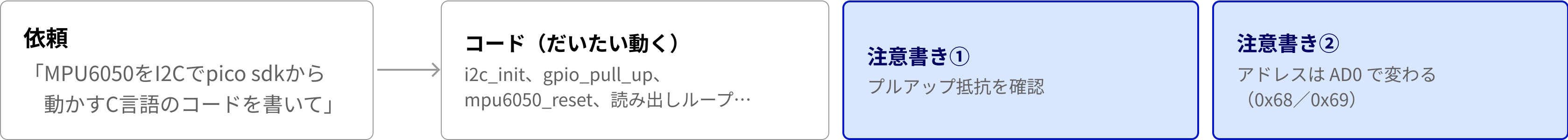
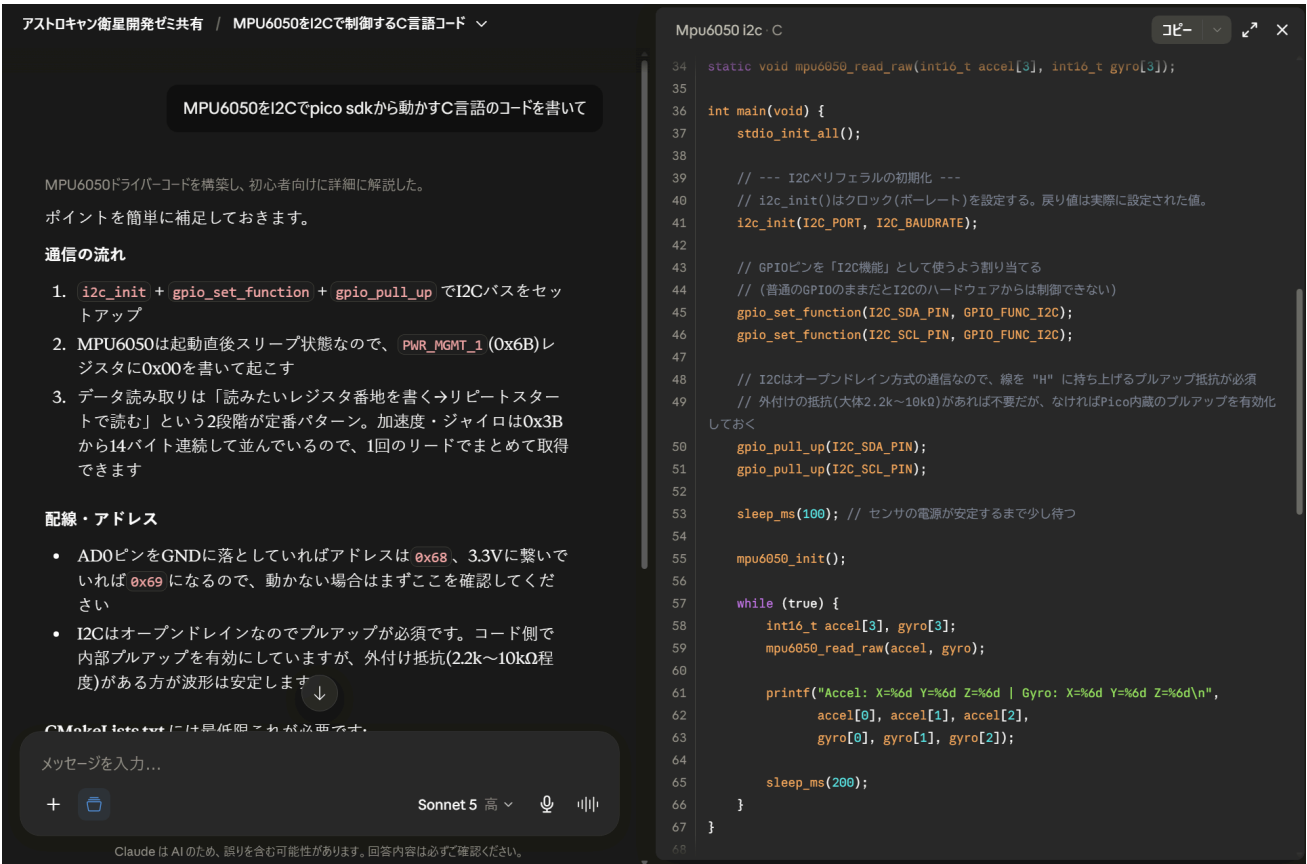


今日の題材となる MPU6050 搭載モジュール

2.1 方法①：AI に書かせる

AI は、ほぼ動くコードと「注意書き」を返してくる

実際の会話：claude.ai/share/d543dcbd-ec47-445b-9d15-1811179875a6（共有のため対話型の画面だが、開発時はエージェント型を使う）



pico-examples にも MPU6050 のサンプルがある

```
mpu6050_reset();                // 起こして
while (1) {
    mpu6050_read_raw(acceleration, gyro, &temp);    // 読んで
    printf("Acc. X = %d, ...", acceleration[0]);    // 表示する
}
```

Raspberry Pi 公式サンプル集 pico-examples の i2c/mpu6050_i2c（概要）：github.com/raspberrypi/pico-examples/tree/master/i2c/mpu6050_i2c

- 骨格は3つだけ。

「リセットして、読み続けて、表示する」。AI のコードもほぼ同じ構造になる。

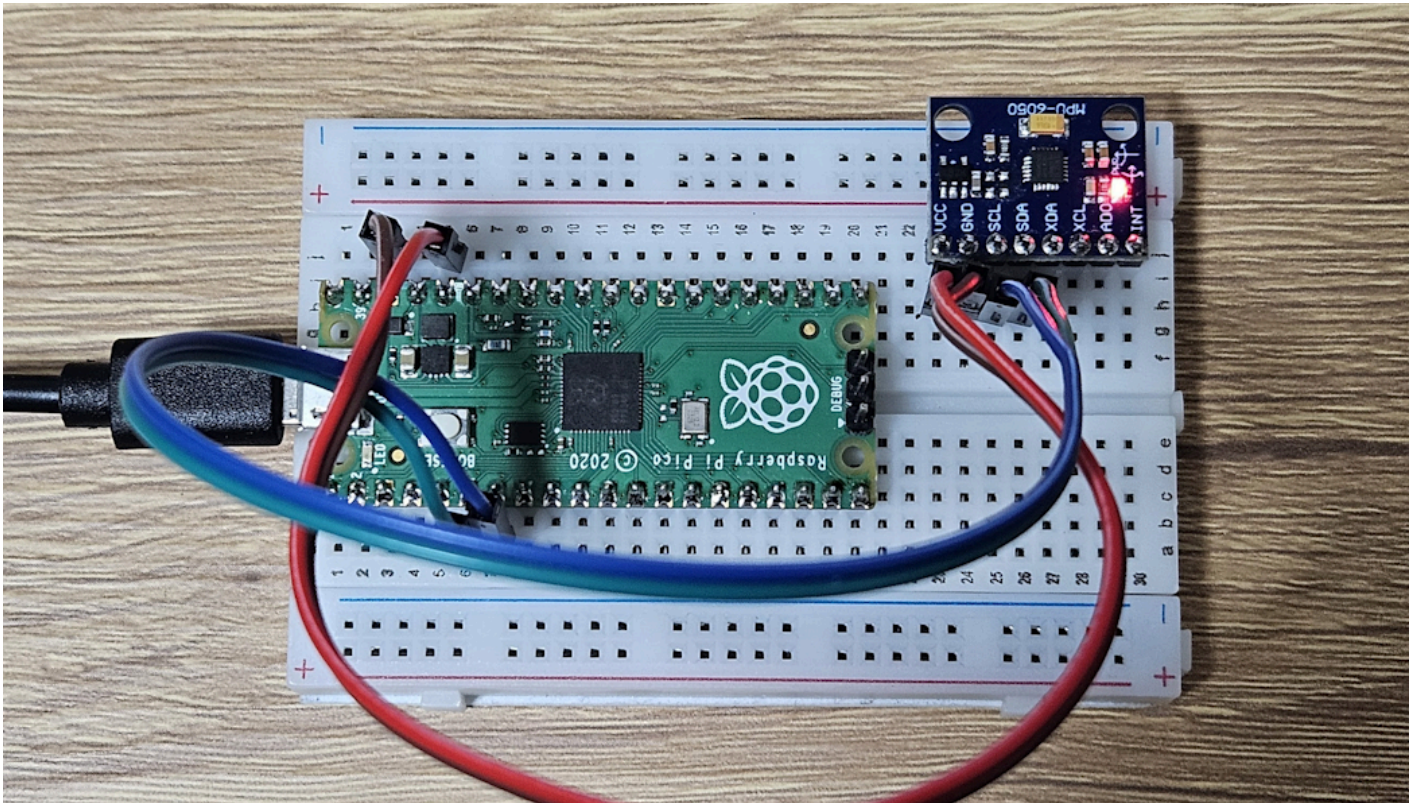
- 今回は方法①で進める。

サンプルは複数種(Pico・Pico W・Pico 2・Pico 2 W)で動くように少し複雑なので、分かりやすいAIのコードの方で

2.3 動かす

一発で動いた

- **コードは1行も変えていない**
ビルドして書き込むだけ。
- **数値が流れ、傾けると反応する**
シリアルモニタに計測値が流れ続ける。



実機の配線 — Pico と MPU6050 をブレッドボードで接続

```
Accel: X= -52 Y= 528 Z= 16584 | Gyro: X= 104 Y= 113 Z= 60
Accel: X= -104 Y= 564 Z= 16456 | Gyro: X= 121 Y= 122 Z= 73
Accel: X= -116 Y= 556 Z= 16560 | Gyro: X= 135 Y= 63 Z= 73
Accel: X= -36 Y= 536 Z= 16560 | Gyro: X= 166 Y= 105 Z= 82
Accel: X= -92 Y= 576 Z= 16468 | Gyro: X= 163 Y= 112 Z= 111
Accel: X= -108 Y= 536 Z= 16496 | Gyro: X= 152 Y= 141 Z= 62
Accel: X= -112 Y= 544 Z= 16496 | Gyro: X= 168 Y= 145 Z= 56
Accel: X= -64 Y= 496 Z= 16428 | Gyro: X= 144 Y= 149 Z= 79
Accel: X= -108 Y= 608 Z= 16476 | Gyro: X= 144 Y= 124 Z= 94
Accel: X= -60 Y= 552 Z= 16540 | Gyro: X= 144 Y= 126 Z= 96
Accel: X= 104 Y= 508 Z= 16520 | Gyro: X= 163 Y= 126 Z= 75
```

AI のコードを動かしたシリアルモニタの出力（正常に動作）

2.4 手元で再現入する：Wokwi

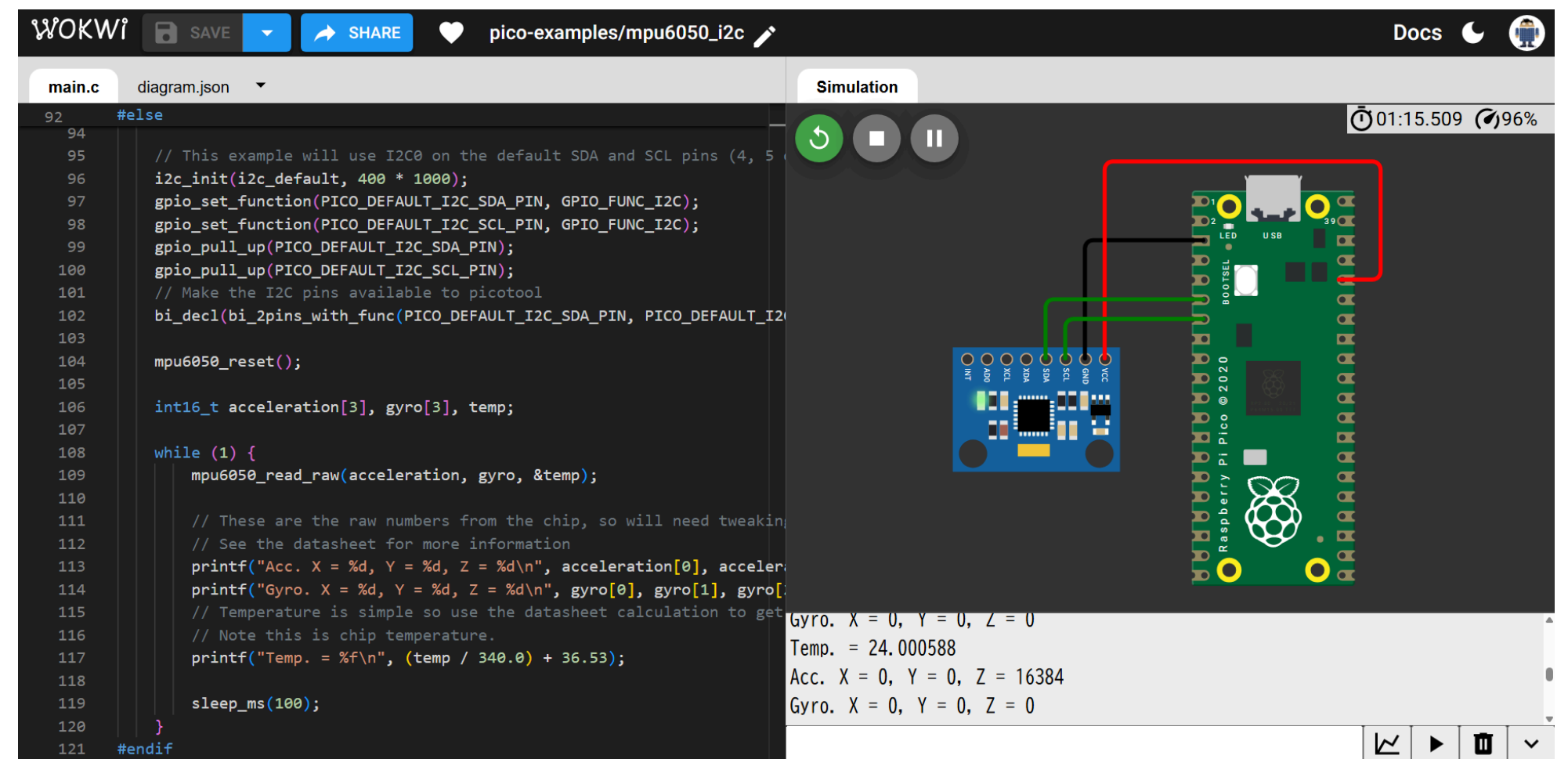
シミュレータでも実行可能

- ブラウザだけで配線と実行

Pico と MPU6050 を Wokwi 上で接続して動かせる。

- ジャイロの値をスライダーで動かせる

出力の変化を手元で確かめられる。



Wokwi で Pico と MPU6050 を接続したシミュレーション画面：wokwi.com/projects/469939527447668737

とりあえず動いた。 では、注意された箇所を間違えていたら？

全で一発で動作させられるとは限らない。

AIも人間も、やってみて動かなかったら修正する、というプロセスを踏む。

一発で動いたが…

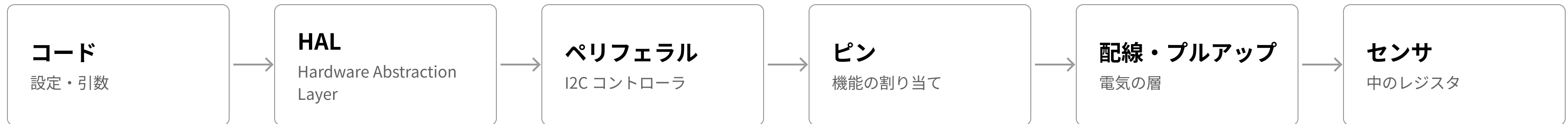
AI の注意書き2つは未回収のまま

実験① プルアップを間違える

実験② アドレスを間違える

3.1 準備①：疑う場所

疑う場所は、理解した経路の上に並んでいる



この経路が、そのまま故障時に「疑う場所の一覧」になる — Week 2 で深掘りした理由の答え

- 経路を知らないと、「動かない」は1個の大きな謎のまま。
- 経路を知っていれば、疑う場所を具体化できる。

3.2 準備②：二分探索

中点で割って、候補を半分に減らす

線形探索 — 端から1個ずつ潰す



二分探索 — 中点で「そこまでは正しいか」を判定する



- 二分探索に必要なのは2つ

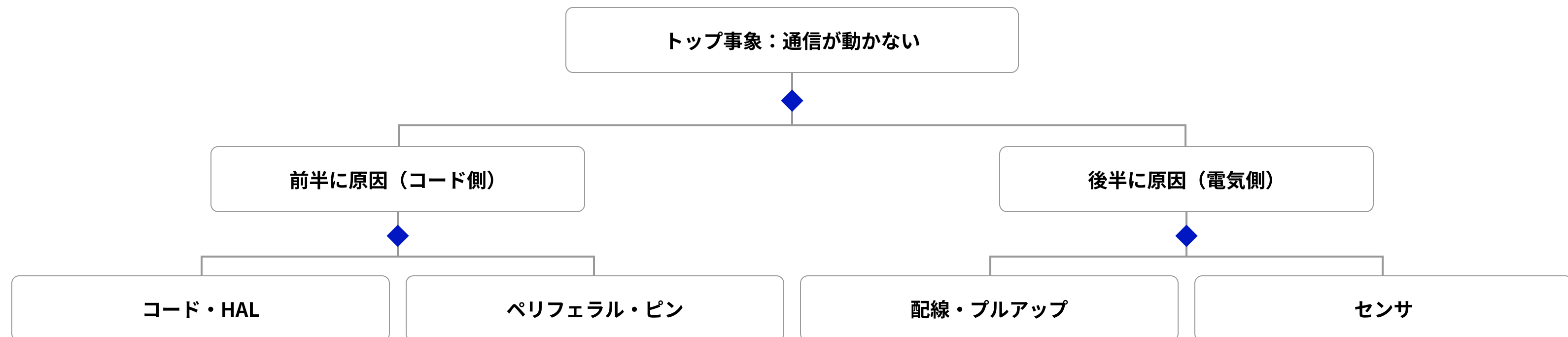
経路の順序を知っていること（対象の仕組みを知っていること）。中点で「分ける」手段があること。

- 当てずっぽうとの違いは「計画」

「どことどこを切り分けるか」を決めてから、手を動かす。

3.3 準備③：分ける手段

分割点に注目する「測定」、ノードに注目する「対照実験」



◆ 測定 — 分割点に注目

分割点で「ここまでは正しいか」を道具で直接観測し、どちら側か決める。
紹介する道具：printf・デバッガ・オシロスコープ・ロジックアナライザ

□ 対照実験 — ノードに注目

仮説を1つ持ち、分割されたノードを1箇所だけ変えて比べる。
直れば原因、直らなければ他に原因。

実験①：プルアップを間違えてみる

内蔵プルアップの有効化をコードから外す（外部プルアップは外してある）。
バスを H に戻す部品が1つも無くなる。

`gpio_pull_up(...);` を外す
仕込み：バスの H を作る部品がゼロに



ターミナルに何も出てこない
症状：手元にあるのはこれだけ

4.1 最初の分割

読み出しの失敗か、通信の失敗か

仮説A 読み出しの失敗

通信は生きている。

加速度の設定やデータの扱いに失敗している。

→ 疑うのはコードの設定・値の変換。

仮説B 通信そのものの失敗

通信自体が成立していない。

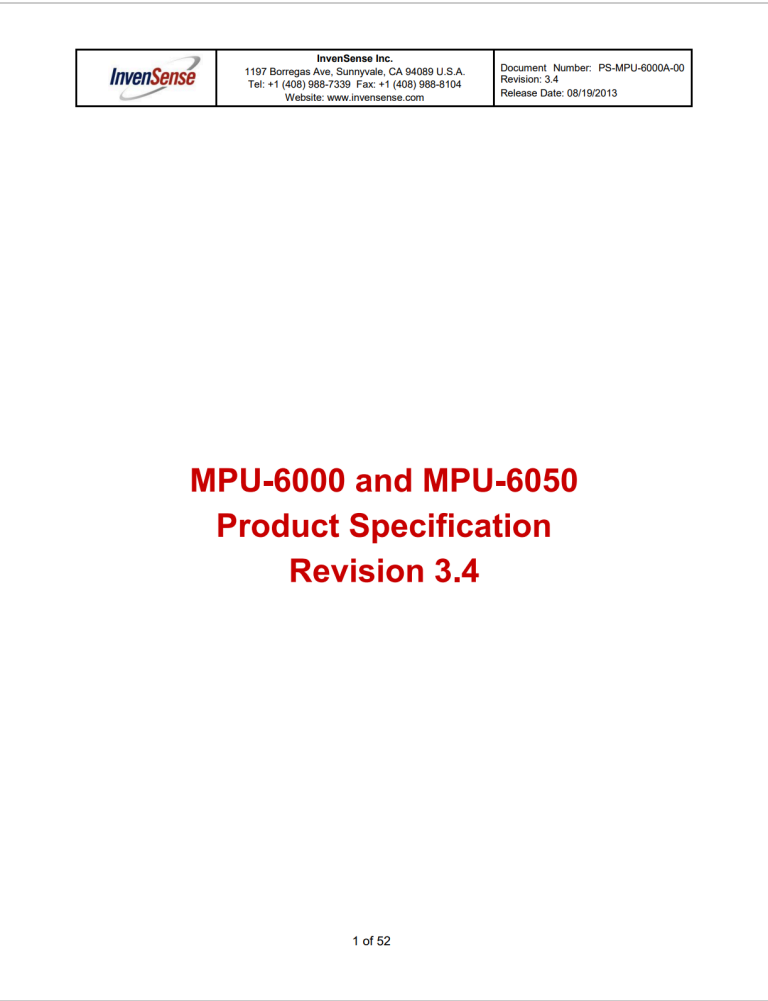
経路の根本が壊れている。

→ 疑うのは配線・電気信号・アドレス。

この2つを、1回で分けられる観測は何か？

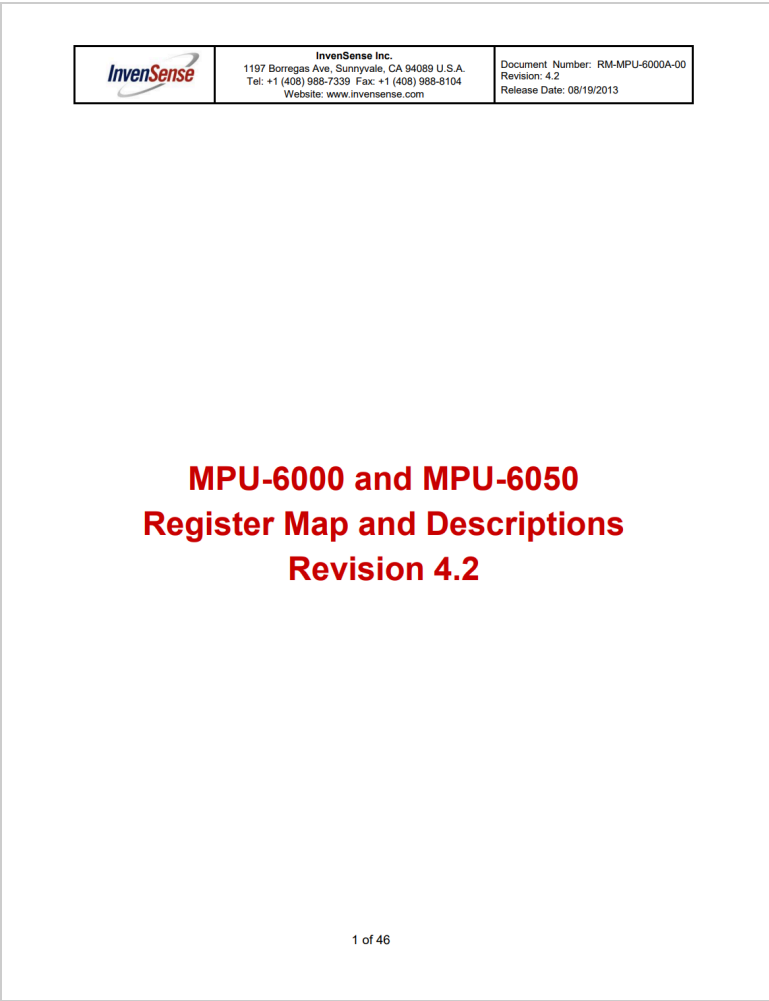
MPU6050 の公式資料は2分冊 — 探し物は「レジスタ」

Product Specification



概要、電氣的条件、ピン配置、通信方法が載っている。

Register Map and Descriptions



レジスタの番地と各ビットの意味が載っている。今日開くのはこちら。

4.3 分割点

WHO_AM_I（0x75） は、読むと必ず 0x68 を返す

① 一覧表で当たりをつける

		MPU-6000/MPU-6050 Register Map and Descriptions				Document Number: RM-MPU-6000A-00 Revision: 4.2 Release Date: 08/19/2013									
3 Register Map The register map for the MPU-60X0 is listed below.															
Addr (Hex)	Addr (Dec.)	Register Name	Serial #	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0				
0D	13	SELF_TEST_X	RW	XA_TEST[4:0]				X0_TEST[4:0]							
0E	14	SELF_TEST_Y	RW	YA_TEST[4:0]				Y0_TEST[4:0]							
0F	15	SELF_TEST_Z	RW	ZA_TEST[4:0]				Z0_TEST[4:0]							
10	16	SELF_TEST_A	RW	RESERVED		XA_TEST[1:0]		YA_TEST[1:0]		ZA_TEST[1:0]					
19	25	SMPRT_DIV	RW									SMPRT_DIV[7:0]			
1A	26	CONFIG	RW	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]						
1B	27	GYRO_CONFIG	RW	-	-	-	FS_SEL[1:0]		-	-	-				
1C	28	ACCEL_CONFIG	RW	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]								
23	35	FFO_EN	RW	TEMP_FIFO_EN	X0_FIFO_EN	Y0_FIFO_EN	Z0_FIFO_EN	ACCEL_FIFO_EN	SLV2_FIFO_EN	SLV1_FIFO_EN	SLV0_FIFO_EN				
24	36	IC_MST_CTRL	RW	MULT_MST_EN	WAIT_FOR_SS	SLV_3_FIFO_EN	IC_MST_P_NSR	IC_MST_CLK[3:0]							
25	37	IC_SLV0_ADDR	RW	IC_SLV0_RW		IC_SLV0_ADDR[6:0]									
26	38	IC_SLV0_REG	RW	IC_SLV0_REG[7:0]											
27	39	IC_SLV0_CTRL	RW	IC_SLV0_EN	IC_SLV0_BYTE_SW	IC_SLV0_REG_DIS	IC_SLV0_GRP	IC_SLV0_LEN[3:0]							
28	40	IC_SLV1_ADDR	RW	IC_SLV1_RW		IC_SLV1_ADDR[6:0]									
29	41	IC_SLV1_REG	RW	IC_SLV1_REG[7:0]											
2A	42	IC_SLV1_CTRL	RW	IC_SLV1_EN	IC_SLV1_BYTE_SW	IC_SLV1_REG_DIS	IC_SLV1_GRP	IC_SLV1_LEN[3:0]							
2B	43	IC_SLV2_ADDR	RW	IC_SLV2_RW		IC_SLV2_ADDR[6:0]									
2C	44	IC_SLV2_REG	RW	IC_SLV2_REG[7:0]											
2D	45	IC_SLV2_CTRL	RW	IC_SLV2_EN	IC_SLV2_BYTE_SW	IC_SLV2_REG_DIS	IC_SLV2_GRP	IC_SLV2_LEN[3:0]							
2E	46	IC_SLV3_ADDR	RW	IC_SLV3_RW		IC_SLV3_ADDR[6:0]									
2F	47	IC_SLV3_REG	RW	IC_SLV3_REG[7:0]											
30	48	IC_SLV3_CTRL	RW	IC_SLV3_EN	IC_SLV3_BYTE_SW	IC_SLV3_REG_DIS	IC_SLV3_GRP	IC_SLV3_LEN[3:0]							
31	49	IC_SLV4_ADDR	RW	IC_SLV4_RW		IC_SLV4_ADDR[6:0]									
32	50	IC_SLV4_REG	RW	IC_SLV4_REG[7:0]											
33	51	IC_SLV4_CTRL	RW	IC_SLV4_CTRL[7:0]											
34	52	IC_SLV4_DO	RW	IC_SLV4_EN	IC_SLV4_INT_EN	IC_SLV4_REG_DIS	IC_MST_CLK[4:0]								
35	53	IC_SLV4_DI	R	IC_SLV4_DATA[7:0]											
36	54	IC_MST_STATUS	R	PASS_THROUGH	IC_SLV4_DONE	IC_LOST_ADR	IC_SLV4_NACK	IC_SLV3_NACK	IC_SLV2_NACK	IC_SLV1_NACK	IC_SLV0_NACK				
37	55	INT_PIN_CFG	RW	INT_LEVEL	INT_OPEN	LATCH_INT_EN	INT_RD_CLEAR	FSYNC_INT_LEVEL	FSYNC_INT_EN	IC_BYPASS_EN	-				
38	56	INT_ENABLE	RW	-	-	-	FFO_OVERFLOW_INT	IC_MST_INT_EN	-	-	DATA_RDY_EN				
3A	58	INT_STATUS	R	-	-	-	FFO_OVERFLOW_INT	IC_MST_INT	-	-	DATA_RDY_INT				

6 of 46

頭から読まず、目次 → レジスタ一覧表とたどる（Week 1 と同じ読み方）。
一覧の末尾近くに WHO_AM_I（0x75）。

② 該当ページで確かめる

4.32 Register 117 – Who Am I WHO_AM_I

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
75	117	-	WHO_AM_I[6:1]						-

Description:

This register is used to verify the identity of the device. The contents of *WHO_AM_I* are the upper 6 bits of the MPU-60X0's 7-bit I²C address. The least significant bit of the MPU-60X0's I²C address is determined by the value of the AD0 pin. The value of the AD0 pin is not reflected in this register.

The default value of the register is 0x68.

Bits 0 and 7 are reserved. (Hard coded to 0)

Parameters:

WHO_AM_I Contains the 6-bit I²C address of the MPU-60X0.
The Power-On-Reset value of Bit6:Bit1 is 110 100.

読むと固定値 0x68 が返る「答えの分かっている読み出し」。期待どおりなら経路の生存が1回で確定する。

4.4 道具①：printf

中の値と進行を、文字で外に出す

```
// 最初にWHO_AM_Iレジスタを読み、センサと正しく通信できているか確認する
uint8_t whoami = mpu6050_read_whoami();
printf("WHO_AM_I = 0x%02X (expected 0x68)\n", whoami);
```

- **組み込みでもよく使われる観測手段**

ソフトウェアの実行状況をログに出せば二分探索が進む。

- **printf の中身は実はかなり複雑**

マイコンにおける printf の経路は UART だったり USB だったりする。ソフトウェアでの"文字列"の扱いも難しい。

- **発展：デバッガ**

SWD（Serial Wire Debug）をつなぎ、実行を止めて中を覗く道具がある。今日は紹介まで。

4.5 観測結果

答えが返ってこない — 通信そのものが失敗している

- ターミナルに何も出てこない

WHO_AM_I の結果の行が届かない。これ自体が観測情報。

- 「呼び出しから戻らない」と考えられる

エラー値が返るだけが失敗ではない。
来ない応答を待ち続けて止まっている。

- 原因が絞り込めた

「読み出しと設定の扱い」側ではなく、通信側に原因があることが分かる。

- 「ここまでは動く」を示す。

今回は省略するが、本来は printf() が動作することは単体で検証が必要。
printf() の動作不良か、応答待ちで到達しないのか切り分けも必要。

mpu6050_read_whoami()

読み出しの呼び出し



応答を待ち続けて戻らない

1バイトの往復すら成立していない



printf(...)

この行に到達しない — 何も表示されない

「読む」のシーケンスは、書くことから始まる

- 1段目は WRITE。

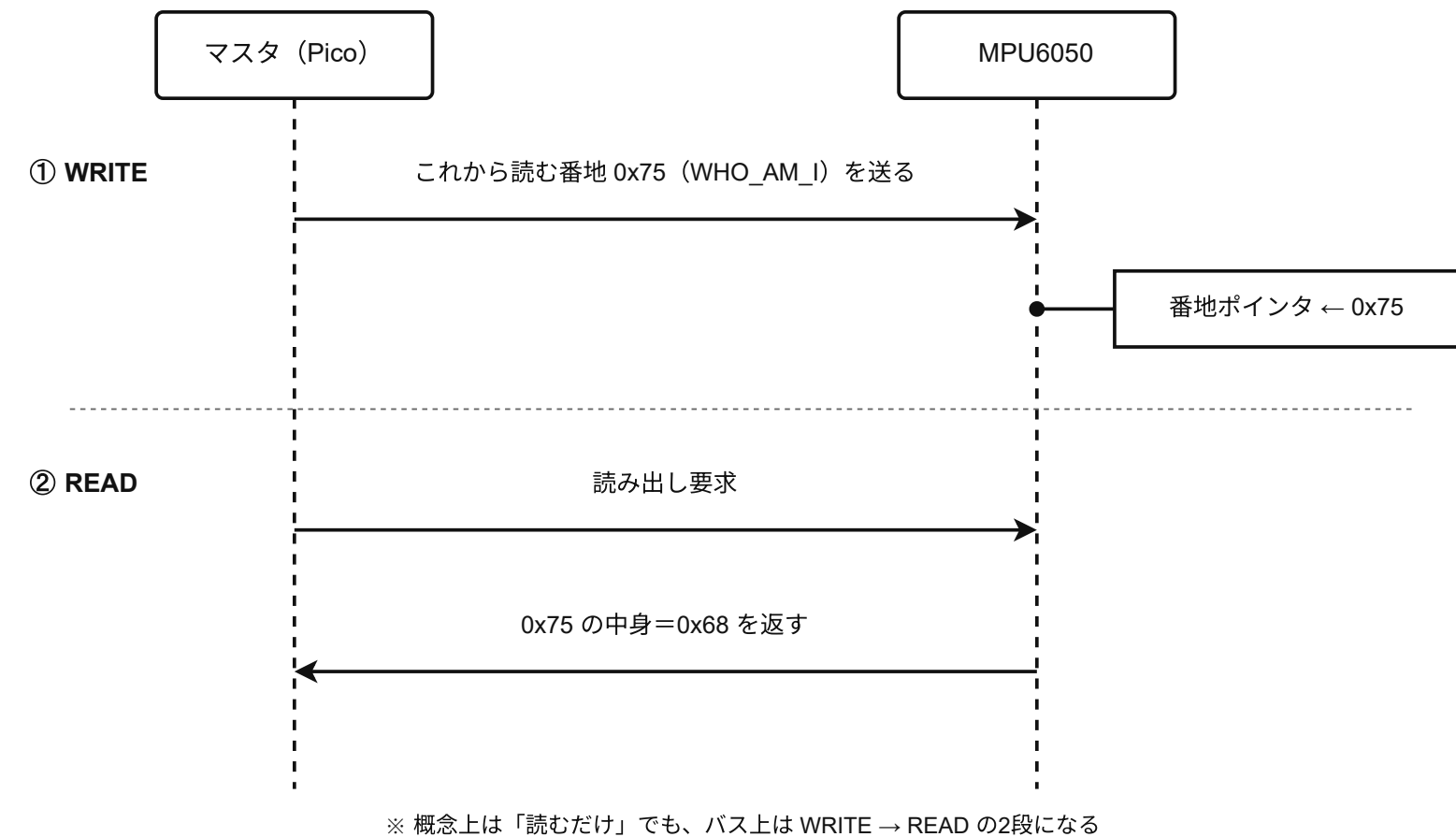
「これから読むのは 0x75 です」と番地を送る。

- 電気信号を見る

— ソフトの道具では、ここから先が写らない

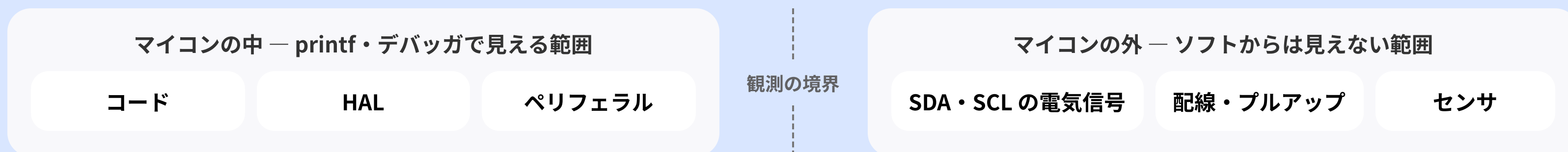
- コードの並びの理由。

i2c_write_blocking と i2c_read_blocking が並ぶのは、このため。



WHO_AM_I の番地 0x75 を WRITE してから、データ 0x68 を READ する2段のシーケンス

電気信号を見る — ソフトの道具では、ここから先が分からない



- ソフト側の手がかりは「呼び出しから戻らない」だけ。

コードにもログにも、SDA・SCL の上の信号は現れない。

- 「はず」を確かめるには、信号を直接見るしかない。

次の中点はマイコンの外 — 電気信号を測る道具を使う。計測器はキット外、今日はデモで見せる。

5.1 道具②：オシロスコープ

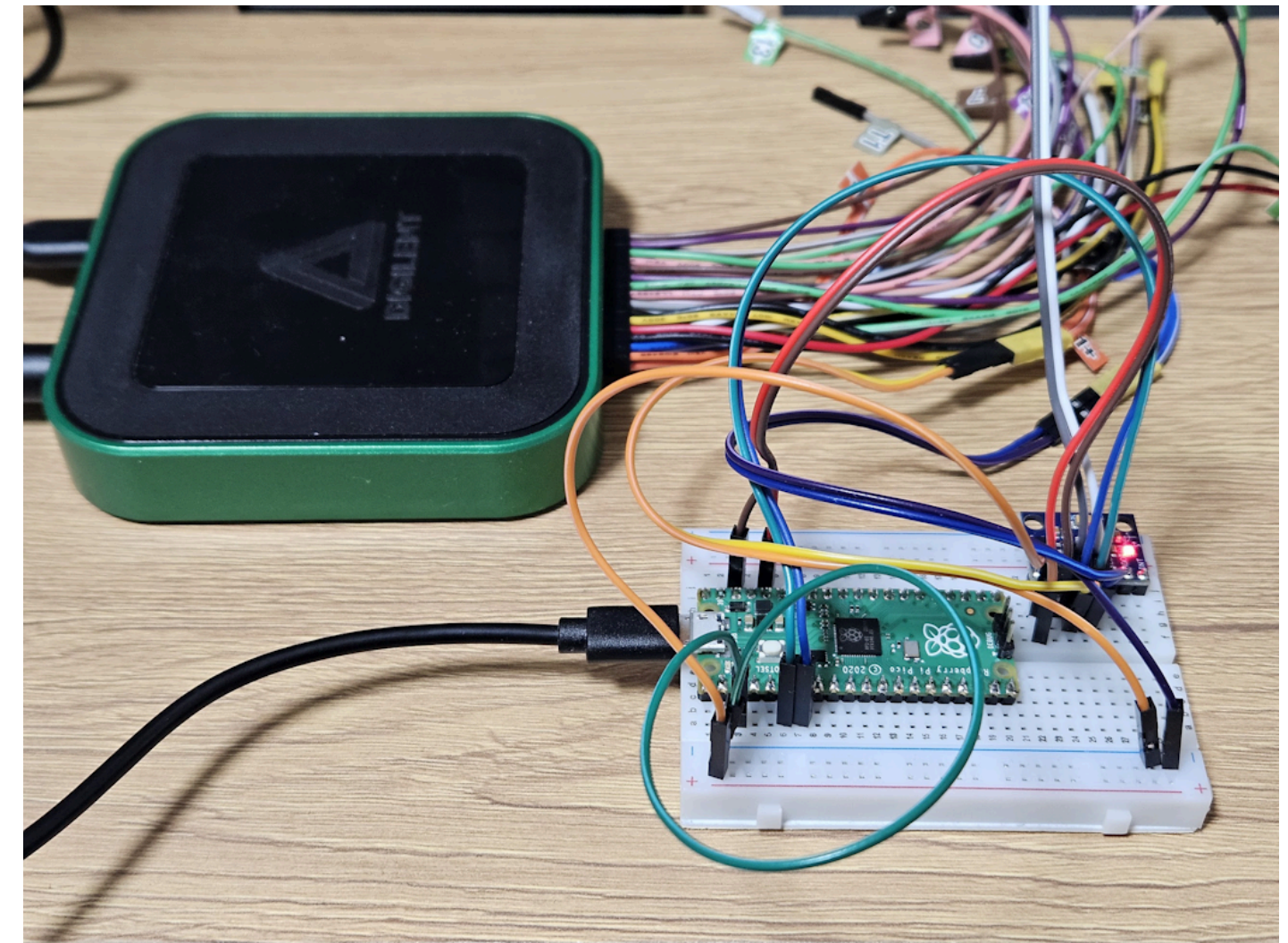
電圧の変化を、波形として描画する

- 見えるのは電圧の形

波形の形、電圧レベル、立ち上がりのなまり。

- デジタル信号の実体もアナログの電圧

通信中なら、H と L を行き来する矩形波が見えるはず。

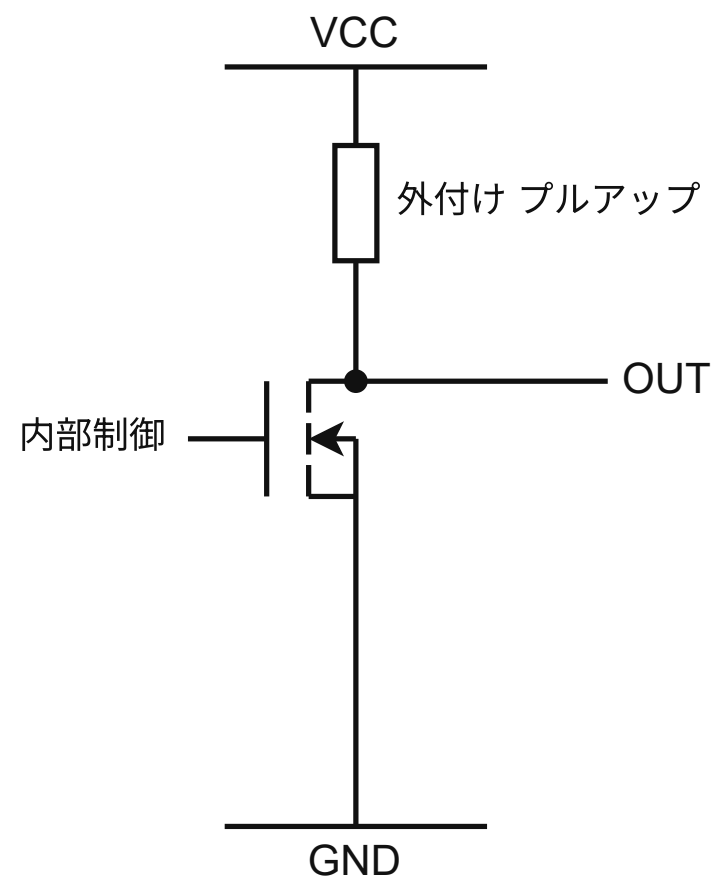


デモ機材の構成 — Pico と MPU6050 の間の SDA ・ SCL にプローブを当てる

5.2 観測

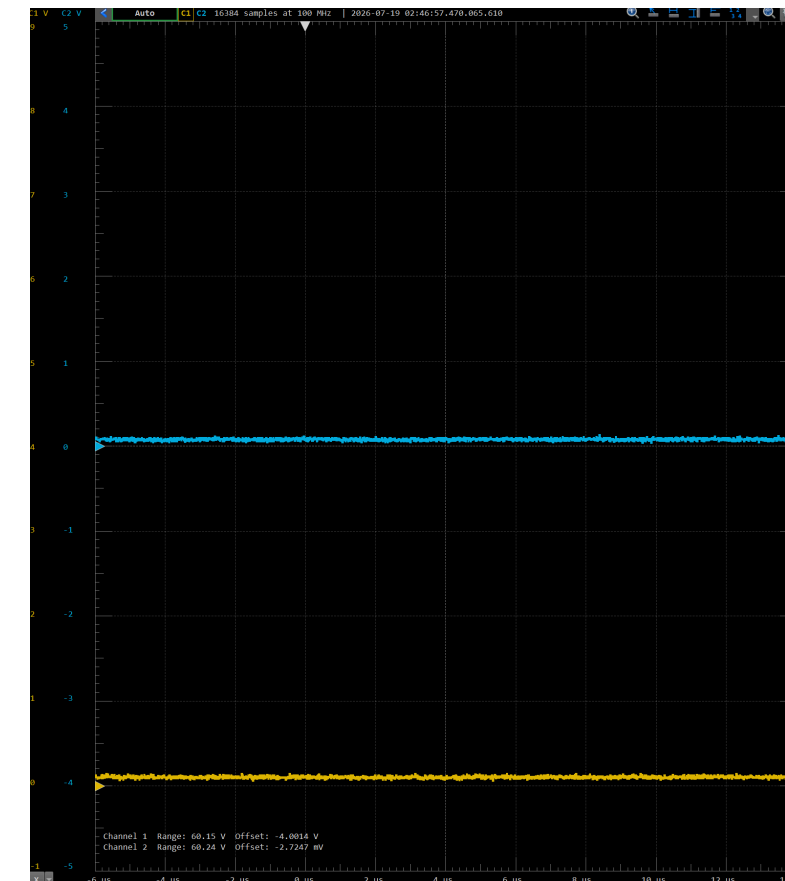
波形が全く立ち上がっていない

復習：オープンドレインとプルアップ（Week 2）



どの部品も線を L に引くことしかできない。
H に戻す仕事は、プルアップ抵抗だけが担う。

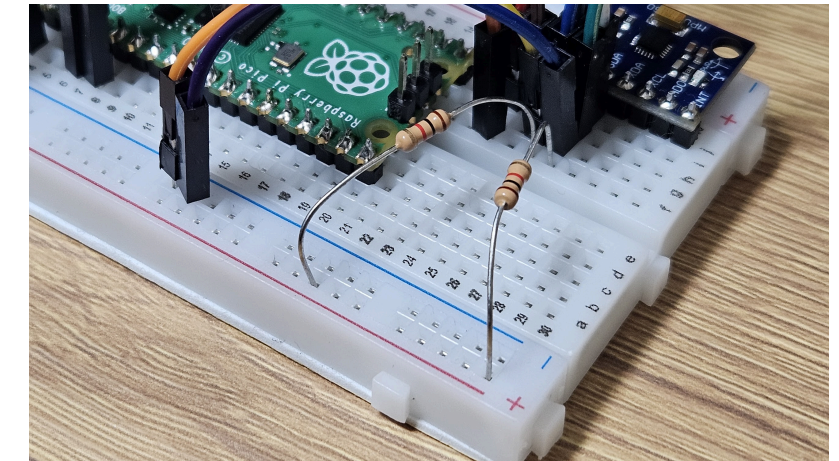
実測：プルアップ無しの SDA・SCL



立ち上がりが無く、通信ができていない。
「H を作る部品が無い」ことが波形にそのまま現れている。

5.3 対処

プルアップを戻すと波形が立つ



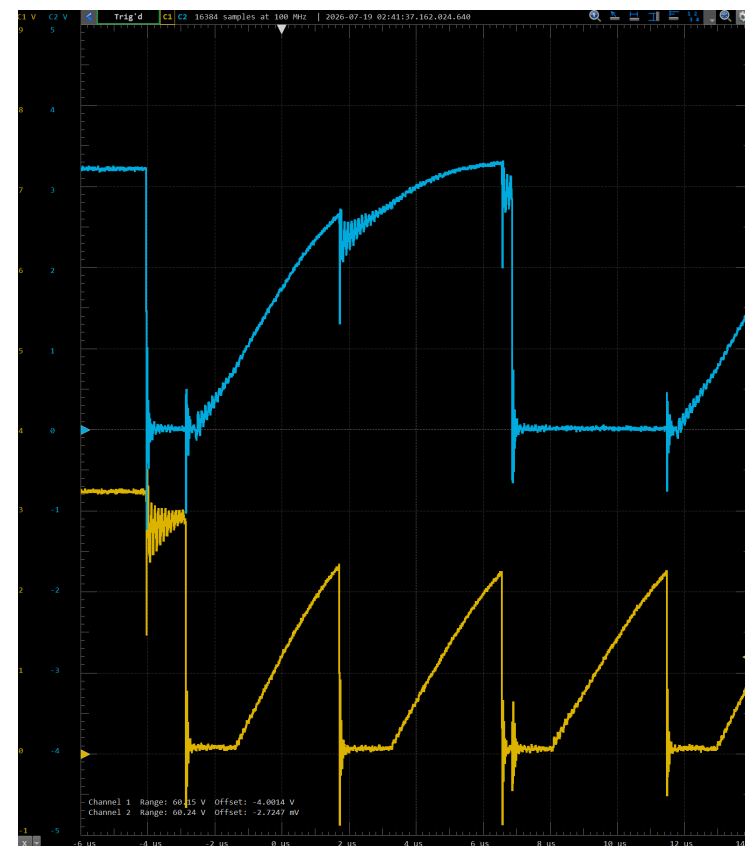
SDA・SCL に外部プルアップ 1kΩ を追加

無し



全く立ち上がらない。
通信は成立しない。

内蔵のみ



動くが、立ち上がりが遅く余裕が無い。
AIのコードはこの状態だった。

外部 1kΩ



H へ引き上げる力が強い。
理想的なクロック波形になった。

復旧した — ただし WHO_AM_I は 0x70、中身は MPU6500 だった

- 通信は復旧

固定値が返り、計測値も流れている。経路の生存確認としては成立。

- 部品が偽物：WHO_AM_I = 0x70（期待は 0x68）

0x70 は後継 MPU6500 の値。刻印は MPU-6050 でも、中身は別品種。
Amazon のレビューにも同じ報告があった。

- まとめ：デバッグの流れ

症状 → 分割（WHO_AM_I・printf） → 分割（波形） → 原因 → 復旧。
中点を決めて、観測して、半分ずつ捨てた。

```
WHO_AM_I = 0x70 (expected 0x68)
Accel: X= -520 Y= 332 Z= 16472 | Gyro: X= 173 Y= 133 Z= 35
Accel: X= -520 Y= 352 Z= 16564 | Gyro: X= 161 Y= 69 Z= 24
Accel: X= -504 Y= 344 Z= 16488 | Gyro: X= 130 Y= 101 Z= 28
Accel: X= -448 Y= 312 Z= 16424 | Gyro: X= 149 Y= 101 Z= 29
Accel: X= -532 Y= 324 Z= 16524 | Gyro: X= 159 Y= 116 Z= 63
Accel: X= -520 Y= 320 Z= 16456 | Gyro: X= 170 Y= 106 Z= 19
Accel: X= -520 Y= 376 Z= 16424 | Gyro: X= 134 Y= 121 Z= 13
```

復旧後の出力 — WHO_AM_I = 0x70 (expected 0x68)

★★★★☆ 中身が6500でした

2026年7月9日に日本でレビュー済み

Amazonで購入

Arduinoで接続しようとしたましたが、MPU-6050用のライブラリでは動かず。

結論、チップの中身がMPU-6500で、

MPU9250_WEライブラリを使ったら認識しました。

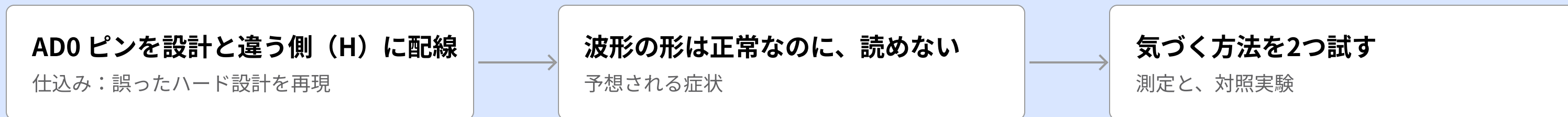
基盤にもチップにもMPU-6050としっかり書いてあったので、見事に引っかかりましたが、

動くには動いたのでまあOKです。

Amazon の商品レビューにも同じ報告

実験②：アドレスを間違えてみる

AI の注意書きの2つ目「アドレスは AD0 で変わる」を、誤ったハード設計を想定して試してみる。



6.1 アドレスとは何か

0x68 はレジスタではなく、センサを呼ぶ「アドレス」

マイコン
0x68 宛に依頼を送る

← I2C バス — 住所で相手を選ぶ →

センサ MPU6050 — 住所：スレーブアドレス 0x68 (AD0=L) / 0x69 (AD0=H)

WHO_AM_I
部屋番号 0x75

設定レジスタ
部屋番号 0x6B など

計測値レジスタ
部屋番号 0x3B～

- I2Cバス内の住所と、センサ内の部屋番号の2階層

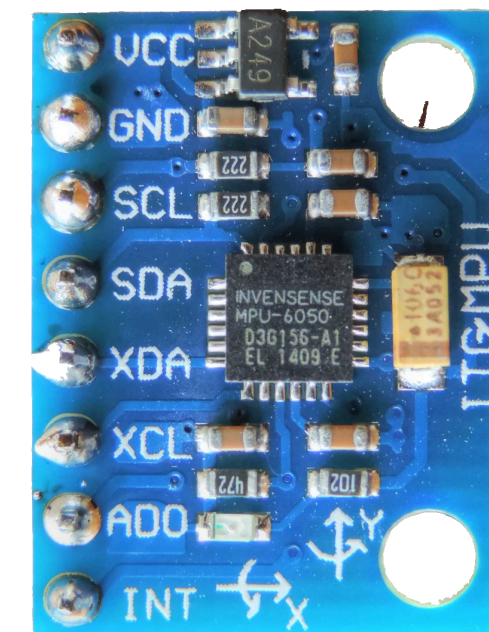
スレーブアドレス (0x68) = 建物の住所。

レジスタの番地 (0x75 など) = 建物の中の部屋番号。

- AD0 ピンでアドレスを変更できる

同じセンサを1本のバスに2個つなぐための仕組み。

配線が想定と違えば、センサのアドレスが整合しない。



MPU6050 搭載モジュール — AD0 ピンの配線でアドレスが決まる

AD0 を H にすると、センサのアドレスは 0x69 になる

- 想定はハード側のミス

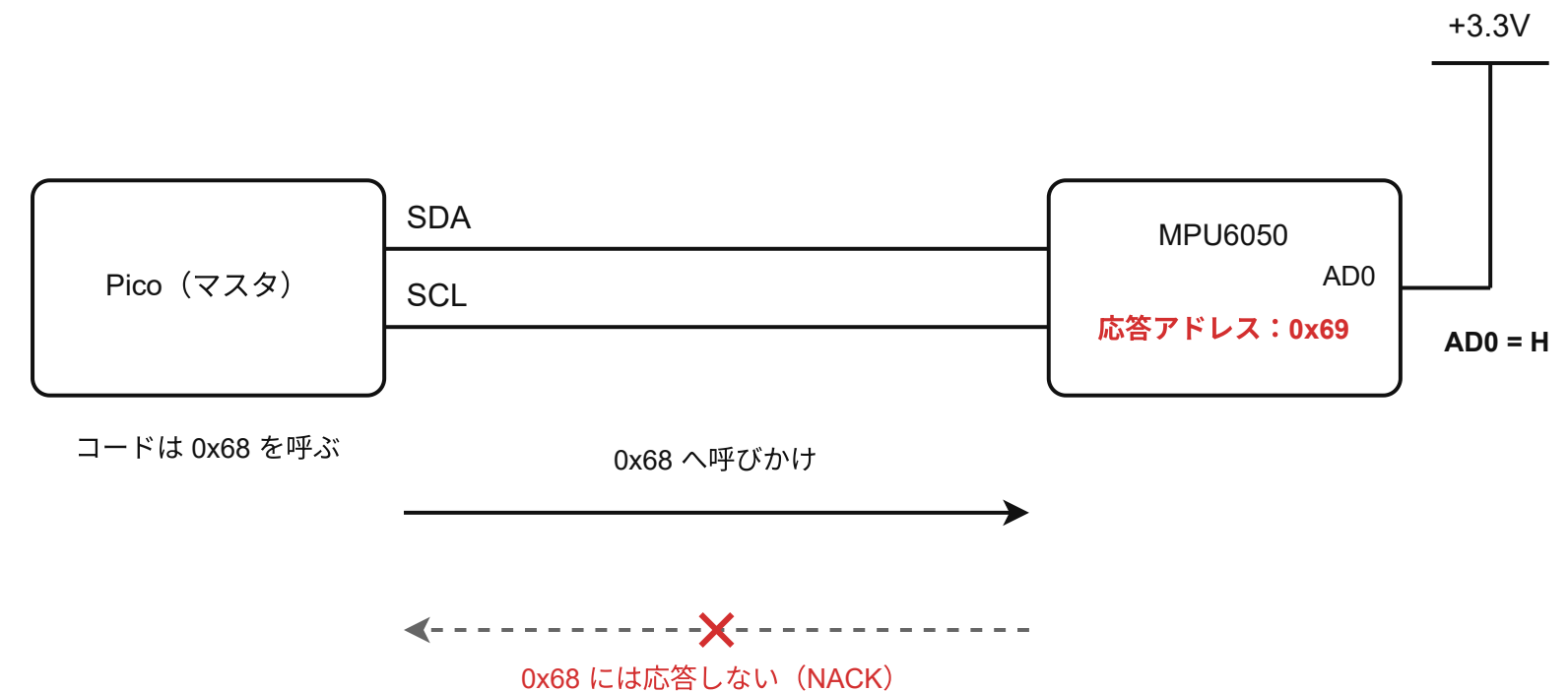
AD0 の配線が設計と違っていた、という状況を再現する。

- 症状はまた「何も出ない」

0x68 を呼び続けるコードは、WHO_AM_I を読めなくなる。

- ソフト側からは区別がつかない

プルアップ無しのとくと、見える症状はまったく同じ。



AD0=H で応答アドレスは 0x69 に。0x68 への呼びかけには誰も応答しない

デジタル信号の意味を解読する装置

- I2Cのシーケンス（Week 2 の復習）

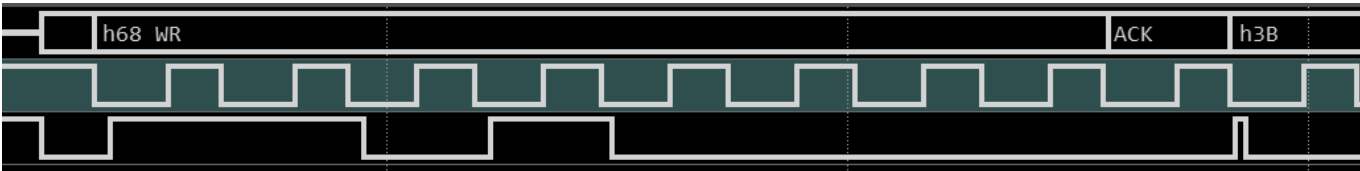
START の後にアドレス7ビット＋読み書き1ビット。
9クロック目に受け手が ACK、応答が無ければ NACK。

- ロジアナのプロトコルデコード機能

ビット列を意味の層に変換して文字で並べる。

- オシロはアナログ、ロジアナはデジタル

電圧の形（なまり）は見えない。H/L の列と意味だけを見る。



ACK の拡大 — 9クロック目で受け手が SDA を L に引く



正常時のデコード — 0x68 WR に ACK、番地、Re-Start、0x68 RD に ACK。WRITE → READ の2段がそのまま現れている

0x68 へ呼びかけて、NACK — そのアドレスには誰もいない

- **NACK=9クロック目で誰も SDA を L に引かなかった**

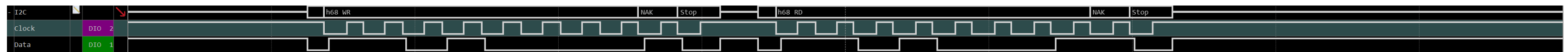
「0x68 へ呼びかけ→NACK」が確認できる。



NACK の拡大 — 9クロック目で誰も SDA を引かない

- **波形は正常＝電気の層は健全**

クロックは見えていて、アドレスが間違っているからACKが来ない。



AD0=H の実機に 0x68 で呼びかけたデコード — WR も RD も NAK で Stop

アドレスを1箇所だけ変えて、結果を比べる

コードA そのまま

```
#define MPU6050_ADDR    0x68
```

現状の再現。動かないことを確かめる基準の側。

コードB 1箇所だけ変える

```
#define MPU6050_ADDR    0x69
```

仮説「アドレスが違うのではないか」に対応する変更のみ。

結果を比べる — 直れば「アドレスの不一致」と確定。直らなければ「原因は他にある」と分かる。どちらに転んでも切り分けが進む

- 確かめ終わったら、設計としての正解に合わせて確定する。

今回はハード側の誤りの想定なので、配線を直してコードは 0x68 に戻す。「動いたからそのまま」とは違う。

整理：切り分けの手段は2系統 — 測定と、対照実験

測定 — 中点で実際に起きていることを、道具で直接見る

printf

マイコンの中の値と進行

オシロスコープ

電気信号の形（電圧・なまり）

ロジックアナライザ

信号の意味（アドレス・ACK/NACK）

対照実験 — 道具が無くてもできる

1箇所変えて比べる

仮説と計画が前提

どちらも目的は同じ — 二分探索の中点で、候補を半分に減らす

- **順序は観測が先、修正は後。**

対照実験の変更も「直すための変更」ではなく「分けるための変更」。当てずっぽうで修正するわけではない。

症状から、壊れた層を逆引きする



Week 2 の「電気の層／意味の層」の2階建てが、そのまま切り分けの軸になる — デバッグでもレイヤー(仕組みの層)の理解が必要

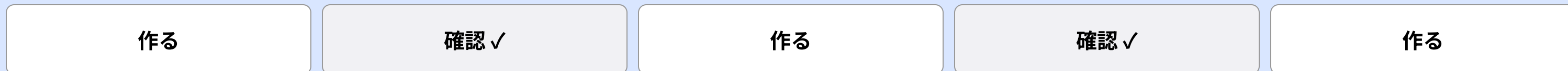
進め方の教訓：少し作っては、動かして確かめる

一気に全部作って、最後に確かめる



動かなかったとき、疑う範囲は「全部」

少し作るごとに、動作確認して足場を築く



疑う範囲は、常に「直前の一步」だけ

- 確認が通った地点は「ここまでは正しい」という足場。

配線1本、設定1つ、機能1つ。単位を小さく刻む。

壊れてから観測点を探すのが二分探索なら、作りながら観測点を残すのがこの進め方。

8.1 ただし AI は一気に作れる

「一つずつ」と「一気に」 — 任せ幅は対象の素性で決める

定番部品

今日の MPU6050。
情報が厚く、一発でも動きやすい。

← 一気に任せる

小さく確かめながら進む →

どこに置くかは設計判断 — 対象の素性で決める「つまみ」

初物・自作基板

合宿の実機。
外れたら絞り込みは自分の仕事。

- **一気に任せて動けば最速。**

動かなければ、今日のように切り分ける。

- **初物ほど、小さく確かめる価値が上がる。**

AI の当たる確率が下がるぶん、観測点を残しながら進む。

まとめ：特定のプロセスと「動く ≠ 適切な設計」

■ 特定のプロセス

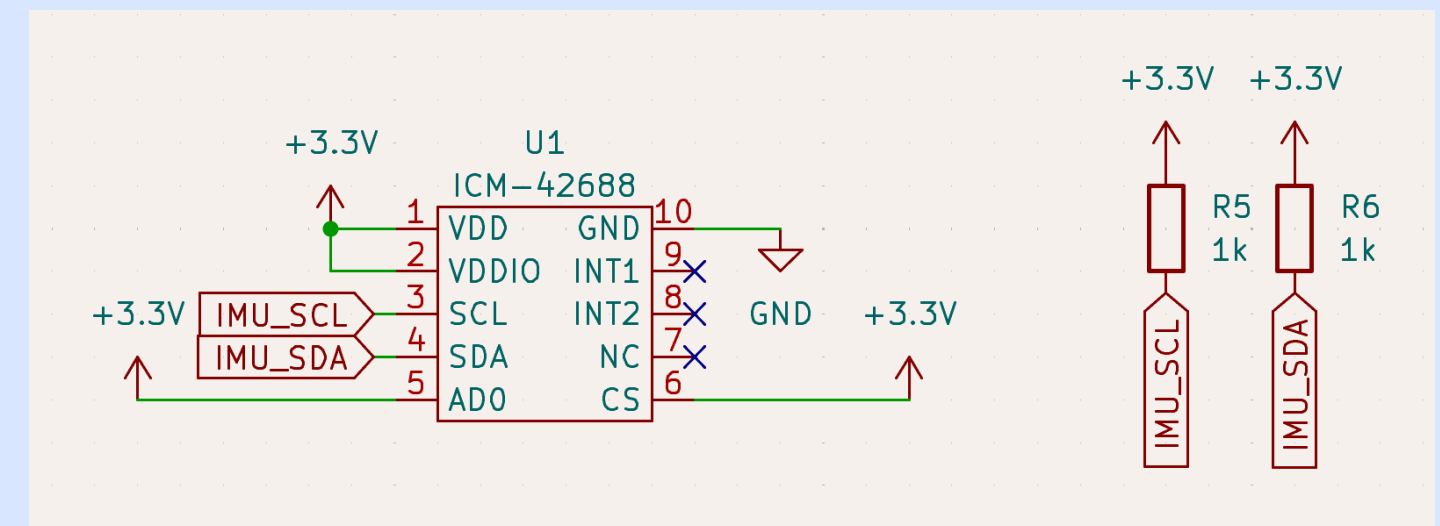
中点を決め、測定（printf・オシロ・ロジアナ）が対照実験で半分に割る。

■ 確実に動く設計

内蔵プルアップで動いてしまっても、実機は外部プルアップ。

■ 合宿への接続

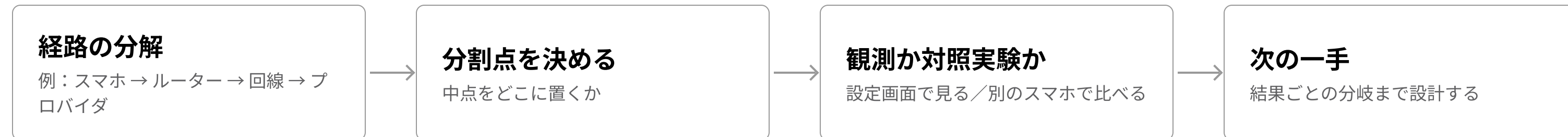
実機 TOP 基板の ICM-42688 も I2C。



模擬衛星 TOP 基板の ICM-42688 まわり
右側に R5・R6=1kΩ の外付けプルアップ

9.1 宿題

身近な機械のトラブルを、どう調べるか — 手順書を設計する



■ 題材は自由。

Wi-Fi ルーターにつながらない、リモコンが効かない、自転車のライトが点かない、など。AI に聞くのは前提。