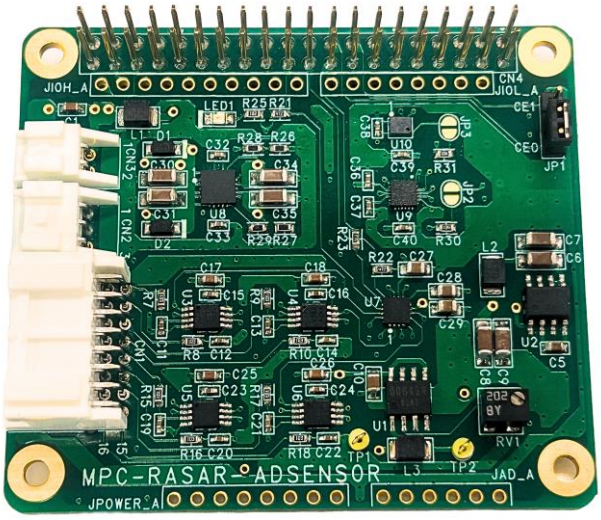
取扱説明書



製品型番	MPC-RASAR-ADSENSOR
品名	Raspberry Pi 12 ビット 8CH AD コンバータ （9 軸センサ、気圧計搭載）
発行日付	令和 7 年 8 月 12 日

EMBEDDED TECHNOLOGY
Corporation
株式会社エンベデッドテクノロジー
〒578-0946
大阪府東大阪市瓜生堂 3-8-13
TEL06-6224-1137
FAX06-6224-1138

変更履歴

日付	版数	訂正内容
令和 7 年 8 月 12 日	初版	

はじめに

１．　製品の保証について

・無償修理

製品ご購入後1年間は無償で修理いたします。

（但し、下記「有償修理」に該当するものを除く）

・有償修理

1)製品ご購入後1年を経過したもの。

2)製品購入1年以内で故障の原因がお客様の取り扱い上のミスによるもの。

3)製品購入1年以内で故障の原因がお客様の故意によるもの。

・免責事項

当社製品の故障、不具合、誤動作あるいは停電によって生じた損害等の純粋経済損失につきましては、当社は一切その責任を負いかねますので、あらかじめご了承ください。

２．　製品について

・当社製品は部品メーカーの製造中止等によりやむを得ず製品の供給を続けることが出来なくなることが有ります。

・当社製品の無断での複製を禁止します。

・当社製品は一般商工業用として設計されており生命、財産に関わるような状況下で使用されることを意図して設計、製造されたものではありません．本製品の故障、誤動作が人命を脅かしたり、人体に危害を与えたりする恐れのある用途（生命維持、監視のための医療用）、および高い信頼性が要求される用途（航空・宇宙用、運輸用、海底中継器、原子力制御用、走行制御用、移動体用）にはご利用されないようご注意ください。すべての電子機器はある確率で故障が発生します．当社製品の故障により、人畜や財産が被害を受けたり、火災事故や社会的損害が生じたりしないように安全設計をお願いします。また長時間連続運転や仕様外の環境でのご使用は避けてください。但し、長時間運転でご使用された場合の故障に付きましては通常どおりの修理保証（1年以内無償、1年以上有償）が受けられます。

３．　カタログ、取扱説明書の記載事項について

・当社製品のカタログ及び取扱説明書は予告無く変更する場合があります。

・取扱説明書に記載されている内容及び回路図の一部又は全部を無断での転載、転用を禁止します。

・本資料に記載された情報、回路図は機器の応用例であり動作、性能を保証するものではなく、実際の機器への搭載を目的としたものではありません。またこれらの情報、回路を使用することにより起因する第三者の工業所有権、知的所有権、その他権利侵害に関わる問題が生じた際、当社はその責を負いませんのであらかじめご了承ください。

４．　海外への輸出について

・当社製品を使用した機器を海外へ持ち出される場合、当社製品のCOCOMパラメーターシートが必要です。その都度お申しつけ頂ければパラメーターシートを発行いたします。

５．　本書に記載された使用条件の範囲内でご使用願います。使用条件の範囲を超えたご使用の場合は本製品の保証は致しかねますのであしからずご了承願います。

目次

１．概要

MPC-RASAR-ADSENSOR は、加速度・角速度・地磁気の 9 軸センサ、気圧・温度センサ、さらに 8CH の 12 ビット SAR 型 AD コンバータを搭載した※Raspberry Pi 向け拡張ボードです。

I2C および SPI 通信に対応し、マルチモーダルセンシング環境の構築に最適です。

ADC は使いやすい 8 チャネル、マルチプレクサ、12 ビット、1MSPS、SAR（逐次比較）型です。

9 軸センサは世界最小の消費電力を誇る 9 軸モーショントラッキングデバイスです。

気圧計は高精度かつ低消費電流の小型デジタル気圧センサです。

※Arduino 用コネクタにも変更可能です。別途お問い合わせください。

２．特徴

ADC は 8 つのチャネルをアナログ入力として独立し構成することが可能です。

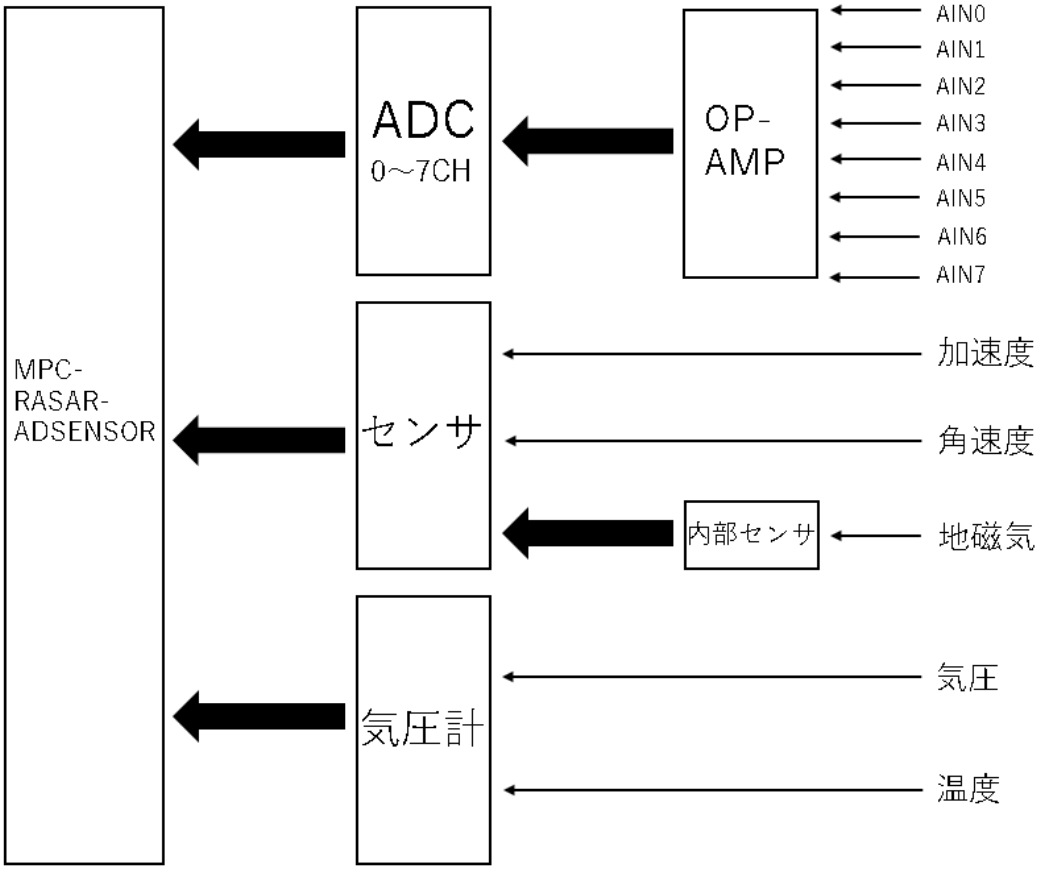
センサは 3 軸加速度センサ、3 軸ジャイロスコープ、3 軸コンパス、デジタルモーションプロセッサ™（DMP™）をパッケージに搭載しています。

気圧計は気圧と温度の両方を測定できます。

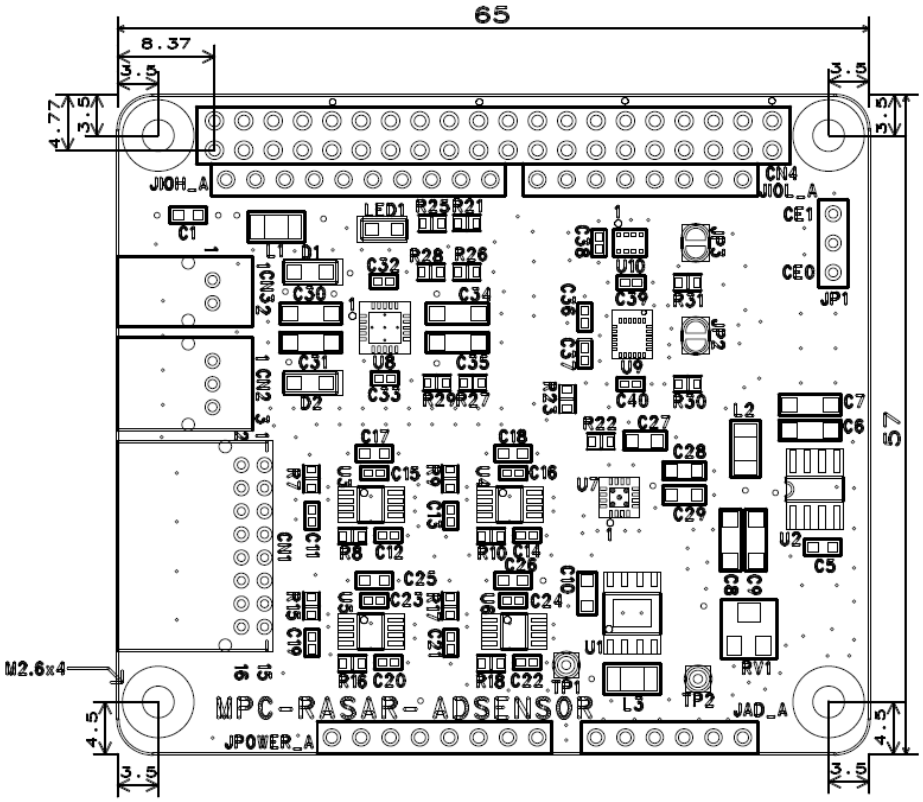
３．仕様

	項目	内容
ADC	ADC 素子	ADS7028IRTER
	分解能	12 ビット
	リニアリティ	±2LSB
	入力チャンネル	8CH
	入力インピーダンス	10kΩ
	アナログ入力範囲	0～5V
	変換レンジ	0～5V
	最大入力電圧	0～5V
	変換時間	600ns（データ送信時間は含まれない）
	カットオフ周波数	7.23kHz（変更可）
	OP-AMP	MAX44246AUA+ 搭載 備考：0V～30mV の範囲は不感域となります。
	通信方式	SPI
9 軸センサ	センサ素子	ICM-20948：加速度センサ、ジャイロセンサ AK09916（ICM-20948 内部搭載）：地磁気センサ
	分解能	16 ビット
	軸	9 軸 3 軸加速度、3 軸角速度、3 軸地磁気
	通信方式	I2C
気圧計	センサ素子	DPS368XTSA1：気圧センサ、温度センサ
	分解能	24 ビット
	通信方式	I2C
その他	電源	Raspberry Pi から供給
	消費電流	19.3mA
	使用・保存温度範囲	0℃～60℃
	湿度範囲	結露無きこと、静電気等の放電を受けないこと

４．ブロック図


MPC-
RASAR-
ADSENSOR
ADC
0～7CH
OP-
AMP
AIN0
AIN1
AIN2
AIN3
AIN4
AIN5
AIN6
AIN7
センサ
加速度
角速度
内部センサ
地磁気
気圧計
気圧
温度

５．実装図



U1：BD00GA3WEFJ-E2
U2：MAX860ESA
U3-6：MAX44246
U7：ADS7028IRTER
U8：TPS7A8701RTJT
U9：ICM-20948
U10：DPS368XTSA1

６．アドレスマップ

６－１．ADC（SPI 通信）

アドレス	機能	READ	WRITE
0x00	SYSTEM_STATUS	ステータス確認	-
0x04	OPMODE_CFG	-	動作モード設定
0x05	PIN_CFG	入力ピン	ピン設定
0x10	SEQUENCE_CFG	-	シーケンス設定
0x11	CHANNEL_SEL	-	シングル変換用チャネル選択

その他のアドレスは ADS7028IRTER(TI)のデータシートを参照してください。

６－２－１．センサ（I2C 通信）

バンク	アドレス	機能	READ	WRITE
0	0x00	WHO_AM_I	デバイス ID 確認	-
0	0x03	USER_CTRL	-	I2C/SPI 制御設定
0	0x06	PWR_MGMT_1	-	電源管理設定 （スリープ解除）
0	0x2D	ACCEL_XOUT_H	加速度 X 軸データ (H)	-
0	0x33	GYRO_XOUT_H	角速度 X 軸データ (H)	-
0	0x3B	EXT_SLV_SENS_DATA_00	磁気センサデータ読み出し	-
3	0x01	I2C_MST_CTRL	-	I2C マスタークロック設定
3	0x03	I2C_SLV0_ADDR	-	磁気センサ I2C アドレス設定
3	0x04	I2C_SLV0_REG	-	スレーブ読み出し 開始レジスタ設定
3	0x05	I2C_SLV0_CTRL	-	スレーブの制御設定 〈バイト数〉
3	0x06	I2C_SLV0_DO	-	スレーブ書き込みデータ
3	0x13	I2C_SLV4_ADDR	-	スレーブ 4 アドレス
3	0x14	I2C_SLV4_REG	-	スレーブ 4 レジスタ
3	0x15	I2C_SLV4_CTRL	-	スレーブ 4 制御
3	0x16	I2C_SLV4_DO	-	スレーブ 4 書き込みデータ

その他のアドレスは ICM-20948(TDK)のデータシートを参照してください。

６－２－２．内部センサ（I2C 通信）

アドレス	機能	READ	WRITE
0x10	ST1	ステータス 1	-
0x31	CNTL2	-	測定モード設定

その他のアドレスは ICM-20948(TDK)のデータシートを参照してください。

６－3．気圧計（I2C 通信）

アドレス	機能	READ	WRITE
0x00	PSR_B2	圧力 MSB	-
0x01	PSR_B1	圧力 MID	-
0x02	PSR_B0	圧力 LSB	-
0x03	TMP_B2	温度 MSB	-
0x04	TMP_B1	温度 MID	-
0x05	TMP_B0	温度 LSB	-
0x06	PRS_CFG	-	圧力設定
0x07	TMP_CFG	-	温度設定
0x08	MEAS_CFG	-	OSR/INT 設定
0x09	CFG_REG	-	ソフトリセット
0x10～0x21	COEF	補正係数	-
0x28	COEF_SRCE	補正係数ソース	-

その他のアドレスは DPS368XTSA1(infineon)のデータシートを参照してください。

７．レジスタ解説

７－１． ADC（SPI 通信）

①SYSTEM_STATUS

アドレス＝0x00

ビット	7	6	5	4	3	2	1	0
名称	RSVD	SEQ_ STATUS	RESERVED	RMS_DONE	OSR_DONE	CRC_ERR_ FUSE	CRC_ERR_ IN	BOR
R/W	R	R	R	R/W	R/W	R	R/W	R/W
Reset	1b	0b	0b	0b	0b	0b	0b	1b

このレジスタは、本ボード内の AD 変換回路やチャンネルシーケンサの状態、および電源投入時の自己診断結果や CRC エラー、電源電圧低下（Brown-out）検出状況などを一括で確認するためのステータスレジスタです。

主に以下の情報を含んでいます。

- **チャンネルシーケンサ状態（Bit 6）**

  変換シーケンスが進行中か停止中かを示します。シーケンスが進行中の場合はシーケンスビットがセットされます。

- **RMS 演算およびオーバーサンプリング演算状態（Bit 4-3）**

  RMS 演算結果や平均化処理が完了しているかを示します。演算が完了すると各フラグがセットされ、結果が読み取り可能になります。

- **電源投入時の CRC チェック結果（Bit 2）**

  電源投入時に行われる設定内容の CRC 自己診断の結果を示します。異常が検出されるとフラグがセットされます。

- **データ受信時の CRC エラー状態（Bit 1）**

  データ通信時の CRC エラーの有無を示します。CRC エラーが検出された場合、該当フラグがセットされます。

- **電源電圧低下（BOR）検出（Bit 0）**

  Brown-out 条件を検出した場合や電源が再投入された場合にフラグがセットされます。

②OPMODE_CFG

アドレス＝0x04

ビット	7	6-5	4	3-0
名称	CONV_ON_ERR	CONV_MODE[1:0]	OSC_SEL	CLK_DIV[3:0]
R/W	R/W	R/W	R/W	R/W
Reset	0b	0b	0b	0b

このレジスタは、AD コンバータの動作モード、CRC エラー発生時の挙動、内部クロック選択、およびオートモニタリングモード時のサンプリング速度を設定するための制御レジスタです。

- **CRC エラー発生時の動作制御（Bit 7）**

  通信インターフェースで CRC エラーが発生した際、自律動作モードを継続するかどうかを制御します。

  0b：CRC エラー発生時もチャンネルシーケンスやピン設定はそのまま継続されます。

  1b：CRC エラー発生時は全チャネルが強制的にアナログ入力に切り替わり、シーケンス動作は一時停止します。エラー解除後に設定が復元されます。

- **変換モード設定（Bit 6-5）**

  AD 変換の動作モードを選択します。

  00b：マニュアルモード（ホストからの指示で変換開始）

  01b：自律動作モード（内部ステートマシンが自動で変換を実行）

- **オシレータ選択（Bit 4）**

  タイミング生成に使用する内部オシレータを選択します。

  0b：高速度オシレータを使用

  1b：低消費電力オシレータを使用

- **サンプリング速度制御（Bit 3-0）**

  自律モニタリングモード（CONV_MODE=01b）で使用するサンプリング速度を設定します。

  この値によって内部クロック分周比が設定され、サンプリング間隔が決まります。

③PIN_CFG

アドレス＝0x05

“00000000”固定です。

④SEQUENCE_CFG

アドレス＝0x10

ビット	7-5	4	3-2	1-0
名称	RESERVED	SEQ_START	RESERVED	SEQ_MODE[1:0]
R/W	R	R/W	R	R/W
Reset	0b	0b	0b	0b

このレジスタは、複数チャネルを順次スキャンする際のチャンネルシーケンサ動作を制御するための設定レジスタです。マルチチャネルの自動スキャンを行う場合に、スキャン開始制御やシーケンスモードを設定できます。

- **シーケンス開始制御（Bit 4）**

  シーケンサ機能（SEQ_MODE＝01b）を使用する場合、スキャンを開始・停止するためのビットです。

  0b：シーケンス停止

  1b：有効チャネルを昇順でスキャン開始（AUTO_SEQ_CHSEL レジスタで選択したチャネルが対象）

- **シーケンスモード設定（Bit 1-0）**

  アナログ入力チャネルのスキャン方式を設定します。

00b：マニュアルシーケンスモード

→ ホストが MANUAL_CHID フィールドでチャネルを指定して変換

01b：自動シーケンスモード

→ 内部チャンネルシーケンサで有効化されたチャネルを順次自動でスキャン

10b：オンザフライ（On-the-fly）シーケンスモード

→ 入力変更に応じて即座にスキャンモードを切り替えるモード

11b：予約（使用しないでください）

⑤CHANNEL_SEL

アドレス＝0x11

ビット	7-4	3-0
名称	ZCD_CHID[3:0]	MANUAL_CHID[3:0]
R/W	R/W	R/W
Reset	0b	0b

このレジスタは、アナログ入力チャネルの選択と、ZCD（Zero Crossing Detection）入力チャネルの設定を行うための制御レジスタです。

シーケンサ動作の有無や ZCD 機能の利用に応じて、適切なチャネルを選択してください。

- **ZCD 入力チャネル設定（Bit 7-4）**

  Zero Crossing Detection（ZCD）用に使用する入力チャネルを指定します。

  指定されたチャネルがアナログ入力として設定されている場合は内部的に ZCD 信号が生成され、デジタル入力の場合はその信号が ZCD 信号として直接使用されます。

- **マニュアルチャネル選択（Bit 3-0）**

  マニュアルモード（SEQ_MODE＝00b）で使用する場合、次回の AD 変換で使用する 4 ビットのチャネル ID を設定します。

  有効な ADC データを取得するには、選択したチャネルが PIN_CFG で GPIO として設定されていない必要があります。

設定値	対応チャネル
0000b	AIN0
0001b	AIN1
0010b	AIN2
0011b	AIN3
0100b	AIN4
0101b	AIN5
0110b	AIN6
0111b	AIN7
1000b～	予約

７－２－１． センサ（I2C 通信）

①WHO_AM_I：デバイス識別用 ID レジスタ

バンク＝0、アドレス＝0x00

ビット	7-0
名称	WHO_AM_I[7:0]
R/W	R

このレジスタの全 8 ビット（Bit 7:0）には、固定のデバイス ID コードが格納されています。

このレジスタは、ICM-20948 デバイスが正しく接続されているかどうかを確認するために使用します。ICM-20948 が正しく接続されている場合、この値は 0xEA です。ホストマイコンなどから読み出すことで、接続先が ICM-20948 であることを確認でき、誤配線や通信不良を簡易的に検出することができます。

②USER_CTRL

バンク=0、アドレス＝0x03

ビット	7	6	5	4	3	2	1	0
名称	DMP_EN	FIFO_EN	I2C_MST_EN	I2C_IF_DIS	DMP_RST	SRAM_RST	I2C_MST_RST	RESERVED
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R

このレジスタは、ICM-20948 内部の各種モジュール（DMP、FIFO、I²C Master など）の有効化・リセット制御を行うための多機能制御レジスタです。FIFO モードや DMP 機能の ON/OFF、I²C Master モジュールの使用有無、リセット操作などを個別に設定することができます。

- **DMP 機能有効化（Bit 7）**

  1 = DMP 機能を有効化

  0 = 現在の処理が完了した後に DMP 機能を無効化

- **FIFO 動作モード有効化（Bit 6）**

  1 = FIFO モードを有効化し、シリアルインターフェース経由で FIFO アクセスを可能にする

  0 = FIFO モードを無効化

  - FIFO を DMA 書き込みで停止するには FIFO_EN を無効にする
  - DMP からの FIFO 書き込みも停止する場合は、DMP も無効にする

- **I²C マスターインターフェース有効化（Bit 5）**

  1 = I²C マスターインターフェースを有効化し、I²C バスをホストから分離（ES_DA/ES_SCL が SDA/SDI と SCL/SCLK から分離）

  0 = I²C マスターインターフェースを無効化し、ES_DA/ES_SCL が SDA/SDI と SCL/SCLK と論理的に接続される

- **I²C スレーブ無効化（Bit 4）**

  1 = I²C スレーブモジュールをリセットし、シリアルインターフェースを SPI モードのみに設定

- **DMP モジュールリセット（Bit 3）**

  1 = DMP モジュールをリセット（非同期リセット）。内部 20MHz クロックの 1 サイクル後に自動でクリアされる。

- **SRAM モジュールリセット（Bit 2）**

  1 = SRAM モジュールをリセット（非同期リセット）。内部 20MHz クロックの 1 サイクル後に自動でクリアされる。

- **I²C マスターモジュールリセット（Bit 1）**

  1 = I²C マスターモジュールをリセット（非同期リセット）。内部 20MHz クロックの 1 サイクル後に自動でクリアされる。

  注意： I²C マスタートランザクション中にこのビットをセットすると I²C スレーブがハングする可能性があるため、ホスト側で I²C スレーブをリセットする必要があります。

③PWR_MGMT_1

バンク=0、アドレス＝0x06

ビット	7	6	5	4	3	2-0
名称	DEVICE_RESET	SLEEP	LP_EN	RESERVED	TEMP_DIS	CLKSEL[2:0]
R/W	R/W	R/W	R/W	R	R/W	R/W

このレジスタは、ICM-20948 の電源管理およびクロックソース選択を制御する主要な設定レジスタです。内部レジスタのリセット、スリープモードの設定、低消費電力モードの有効化、温度センサの無効化、そしてクロックソースの選択が行えます。初期値は 0x41 です。

- **内部レジスタリセット（Bit 7）**

  1 = 内部レジスタをリセットしてデフォルト設定に復帰

  （書き込み後、自動的にビットはクリアされます）

  0 = リセットしない

- **スリープモード設定（Bit 6）**

  1 = スリープモードを有効化（全アナログ部は電源 OFF）

  0 = スリープ解除（チップを動作状態に戻す）

- **低消費電力モード有効化（Bit 5）**

  LP_EN はデジタル回路の低消費電力機能を制御します。センサ自体の LP モード設定と組み合わせて使用すると、全体の消費電流をさらに低減できます。

  1 = 低消費電力機能 ON（デジタル回路向け）

  0 = 低消費電力機能 OFF

  ※ LP_EN はセンサがローノイズモードの場合は無効です。

- **温度センサ無効化（Bit 3）**

  1 = 温度センサ機能を無効化

  0 = 温度センサ機能有効（デフォルト）

- **クロックソース選択（Bit 2-0）**

  センサの動作クロックソースを選択します。

設定値	クロックソース
0	内部 20MHz オシレータ
1-5	使用可能な最適なクロックソースを自動選択
6	内部 20MHz オシレータ
7	クロックを停止し、タイミングジェネレータも停止

※ クロック性能を最大限活用してジャイロスコープの性能を十分に発揮するには、CLKSEL[2:0] は 1～5 に設定することが推奨されています。

④ACCEL_XOUT_H

バンク＝0、アドレス＝0x2D

ビット	7-0
名称	ACCEL__XOUT__H[7:0]
R/W	R

このレジスタは、ICM-20948 の加速度センサ（Accelerometer）X 軸方向の出力データの上位バイト（MSB）を格納している読み取り専用のデータレジスタです。X 軸の加速度値を取得する際は、このレジスタとペアの ACCEL_XOUT_L レジスタを組み合わせて 16 ビットのフルスケールデータとして扱います。上位 8 ビットは X 軸の最新の加速度計測結果が格納されており、下位 8 ビットは ACCEL_XOUT_L(0x2E)から取得し、両者を結合して 16 ビットデータを形成します。

⑤GYRO_XOUT_H

バンク＝0、アドレス＝0x33

ビット	7-0
名称	GYRO__XOUT__H[7:0]
R/W	R

このレジスタは、ICM-20948 のジャイロスコープ（Gyroscope）X 軸方向の角速度出力データの上位バイト（MSB）を格納している読み取り専用のデータレジスタです。X 軸の角速度データを取得する際は、このレジスタとペアの GYRO_XOUT_L レジスタを組み合わせて 16 ビットのフルスケールデータとして扱います。上位 8 ビットは X 軸の最新の角速度計測結果が格納されており、下位 8 ビットは GTRO_XOUT_L(0x34)から取得し、両者を結合して 16 ビットデータを形成します。

⑥EXT_SLV_SENS_DATA_00～EXT__SLV__SENS_DATA_00[7:0]

バンク＝0、アドレス＝0x3B～0x3F

ビット	7-0
名称	EXT__SLV__SENS_DATA_00[7:0]～EXT__SLV__SENS_DATA_00[7:0]
R/W	R

このレジスタは、ICM-20948 の I²C マスターインタフェース経由で外部 I²C デバイスから取得した 8 ビットのセンサデータを格納する読み取り専用のバッファレジスタです。このレジスタは EXT_SLV_SENS_DATA_00 から始まり、連続したアドレスに複数のバイトが用意されています。データは、外部センサスレーブの設定（I2C_SLV(0- 4)_ADDR、I2C_SLV(0- 4)_REG、I2C_SLV(0- 4)_CTRL）に基づいて自動的に更新されます。これにより、外部スレーブから複数バイトの連続データをバッファリングできます。

⑦I2C_MST_CTRL

バンク＝3、アドレス＝0x01

ビット	7	6-5	4	3-0
名称	MULT_MST_EN	RESERVED	I2C_MST_P_NSR	I2C_MST_CLK[3:0]
R/W	R/W	R	R/W	R/W

このレジスタは、ICM-20948 の I²C マスターインタフェースの動作モードやクロック周波数を設定する制御レジスタです。I²C バス上でのマルチマスター機能の有効化、スレーブ間のリード動作切り替え、マスタークロックの設定などを行います。

- **マルチマスター機能の有効化（Bit 7）**

  このビットを 1 に設定すると I²C バス上のマルチマスター動作が可能になります。

  無効化（0）すると、I2C_MST_IF へのクロック供給が停止され、

  未使用時の電力を低減し、ロストアービトレーション検出ロジックも無効になります。

設定値	動作内容
0	マルチマスター機能　無効
1	マルチマスター機能　有効

- **スレーブ間リード操作の遷移モード（Bit4）**

  スレーブからの連続リード操作の間に STOP か RESTART を挿入するかを指定します。

設定値	動作内容
0（0b0）	スレーブ間のリードは RESTART を挿入
1（0b1）	スレーブ間のリードは STOP を挿入

- **I²C マスタークロック周波数設定（Bit3-0）**

  マスタークロックの分周設定を指定します。必要な通信速度に応じて設定値を選択してください。

⑧I2C_SLV0_ADDR

バンク＝3、アドレス＝0x03

ビット	7	6-0
名称	I2C_SLV0_RNW	I2C_ID_0[6:0]
R/W	R/W	R/W

このレジスタは、ICM-20948 の I²C マスターインタフェースが外部スレーブ 0（Slave 0）と通信する際に使用するスレーブアドレスおよびリード/ライト方向の指定 を行う制御レジスタです。

- **スレーブ 0 へのリード/ライト方向指定（Bit 7）**

  外部 I2C スレーブ 0 への転送方向を設定します。

設定値	動作内容
0	書き込み〈マスターからスレーブへ送信〉
1	読み込み〈マスターがスレーブから受信〉

- **スレーブ 0 の物理アドレス設定（Bit6-0）**

  I2C スレーブデバイスの 7 ビットの物理アドレスを設定します。

  この値は通信開始時にアドレスフェーズとして送信されます。

⑨I2C_SLV0_REG

バンク＝3、アドレス＝0x04

ビット	7-0
名称	I2C_SLV0_REG[7:0]
R/W	R/W

このレジスタは、ICM-20948 の I²C マスターインタフェースが外部 I²C スレーブ 0（Slave 0）と通信を開始する際にアクセスするスレーブ側レジスタアドレスを設定するための制御レジスタです。外部 I²C スレーブ 0 とデータ転送を開始する際、スレーブ内部のどのレジスタからリードまたはライトを行うかを指定します。8 ビット幅で指定するため、スレーブデバイスが複数のレジスタを持つ場合でも柔軟にアクセスできます。

⑩I2C_SLV0_CTRL

バンク＝3、アドレス＝0x05

ビット	7	6	5	4	3-0
名称	I2C_SLV0_EN	I2C_SLV0_BYTE_SW	I2C_SLV0_REG_DIS	I2C_SLV0_GRP	I2C_SLV0_LENG[3:0]
R/W	R/W	R/W	R/W	R/W	R/W

このレジスタは、ICM-20948 の I²C マスターインタフェースが外部 I²C スレーブ 0（Slave 0）と通信する際の動作制御に使用します。

主にスレーブからのデータ取得有効化、バイトスワップ、レジスタ書き込み制御、グループ化、転送バイト数などを設定します。

- **スレーブ 0 からのデータ取得有効化（Bit 7）**

設定値	動作内容
0	このスレーブからのデータ取得を無効化
1	データ取得を有効化し、EXT_SENS_DATA_00 以降に取得データを格納

- **ワードリード時のバイトスワップ制御（Bit 6）**

  ロー/ハイバイトの順序を逆にして読み取る場合に使用します。

  最初のバイト読み込み（I2C_SLV0_REG[0]=1）や末尾バイトの LSB=0 の場合はスワップしません。

設定値	動作内容
0	スワップしない
1	2 バイト単位でスワップ

- **レジスタアドレス無効化（Bit 5）**

  これをセットすると、通信時にスレーブ内部のレジスタアドレスを送信せず、連続したデータだけを送受信します。

設定値	動作内容
0	通常動作〈レジスタアドレス送信あり〉
1	レジスタアドレス送信なし〈データのみ〉

- **2 バイトデータのグルーピング制御（Bit 4）**

  外部センサデータを 2 バイト単位で扱う場合のアドレスの組み合わせを定義します。

- **スレーブ 0 からの転送バイト数（Bit 3-0）**

  I2C スレーブ 0 から一度の転送で取得するバイト数。

⑪I2C_SLV0_DO

バンク＝3、アドレス＝0x06

ビット	7-0
名称	I2C_SLV0_DO[7:0]
R/W	R/W

このレジスタは、ICM-20948 の I²C マスターインタフェースが外部 I²C スレーブ 0（Slave 0）へデータを書き込む際に送信するデータを格納する送信バッファです。スレーブ 0 に対してライト（書き込み）転送を行う場合に使用されます。書き込みデータはこの 8 ビットの送信バッファに設定され、I²C マスターにより外部スレーブへ送信されます。リードモード（I2C_SLV0_RNW = 1）の場合、このレジスタは使用されません。

⑫I2C_SLV4_ADDR

バンク＝3、アドレス＝0x13

ビット	7	6-0
名称	I2C_SLV4_RNW	I2C_ID_4[6:0]
R/W	R/W	R/W

このレジスタは、ICM-20948 の I²C マスターインタフェースが外部 I²C スレーブ 4（Slave 4）と通信す

る際に使用するスレーブアドレスおよびリード/ライト方向を設定する制御レジスタです。

- **スレーブ 4 へのリード/ライト方向設定（Bit 7）**

設定値	動作内容
0	書き込み〈マスターからスレーブへ送信〉
1	読み込み〈マスターがスレーブから受信〉

I²C バス上の R/W ビットに相当する機能で、外部スレーブ 4 とのデータ転送方向を切り替えます。

- **スレーブ 4 の物理アドレス設定（Bit 6-0）**

  外部 I2C スレーブ 4 の 7 ビット物理アドレスを設定します。

  設定したアドレスは通信開始時に送信されます。

⑬I2C_SLV4_REG

バンク＝3、アドレス＝0x14

ビット	7-0
名称	I2C_SLV4_REG[7:0]
R/W	R/W

このレジスタは、ICM-20948 の I²C マスターインタフェースが外部 I²C スレーブ 4（Slave 4）と通信する際にアクセスを開始するスレーブ側の内部レジスタアドレスを指定する制御レジスタです。外部 I2C スレーブ 4 とリードまたはライト通信を開始する際に、スレーブ内部のどのレジスタからデータ転送を行うかを指定します。このフィールドに設定した値が、I2C スタート条件の後にスレーブアドレス送信後に続いて送られる。アドレス幅は 8 ビットで、スレーブ側の内部アドレス構造に合わせて設定してください。

⑭I2C_SLV4_CTRL

バンク＝3、アドレス＝0x15

ビット	7	6	5	4-0
名称	I2C_SLV4_EN	I2C_SLV4_INT_EN	I2C_SLV4_REG_DIS	I2C_SLV4_DL　Y[4:0]
R/W	R/W	R/W	R/W	R/W

このレジスタは、ICM-20948 の I²C マスターインタフェースが外部 I²C スレーブ 4（Slave 4）とのデータ転送を制御する設定レジスタです。データ転送の有効化、完了割り込み、レジスタアドレス送信の有無、サンプリングディレイなどを設定します。

- **スレーブ 4 とのデータ転送有効化（Bit 7）**

設定値	動作内容
0	スレーブ 4 との通信を無効化
1	スレーブ 4 とのデータ転送を有効化

リード時は I2C_SLV4_DI に結果が格納され、ライト時は I2C_SLV4_DO に設置した値が送信される。

単発の転送が完了するとこのビットは自動でクリアされます。

ライト通信を行う前に必ず I2C_SLV4_DO を設定すること。

- **スレーブ 4 転送完了割り込み有効化（Bit 6）**

設定値	動作内容
0	スレーブ 4 の転送完了で割り込みを発生させない
1	スレーブ 4 のデータ転送完了時に割り込みを生成する

- **レジスタアドレス送信無効化（Bit 5）**

  通常、外部スレーブに内部レジスタアドレスを送信してからデータを転送しますが、このビットを有効化するとレジスタアドレスを送信せずに連続したデータのみの転送が可能です。

設定値	動作内容
0	レジスタアドレス送信あり〈通常モード〉
1	レジスタアドレス送信なし〈REG_DIS 有効〉

- **サンプリングディレイ設定（Bit 4-0）**

  I2C_MST_DELAY_CTRL を有効化した場合にのみ適用されます。

  スレーブ 4 のデータ取得を 1/(1+I2C_SLV4_DLY)サンプル周期で間引いて実行することができます。

  間引き率は I2C_MST_ODR_CONFIG により決まります。

⑮I2C_SLV4_DO

バンク＝3、アドレス＝0x16

ビット	7-0
名称	I2C_SLV4_DO[7:0]
R/W	R/W

このレジスタは、ICM-20948 の I²C マスターインタフェースが外部 I²C スレーブ 4（Slave 4）へデータを書き込む際に送信するデータを設定する送信バッファです。外部 I²C スレーブ 4 とのライト（書き込み）通信時に送信する 8 ビットのデータを設定します。書き込み方向（I2C_SLV4_ADDR の I2C_SLV4_RNW = 0）に設定されているときのみ有効です。送信データは I2C_SLV4_CTRL の I2C_SLV4_EN をセットして転送を有効化すると外部スレーブへ送信されます。

７－２－２． 内部センサ（I2C 通信）

①ST1

アドレス＝0x10

ビット	7-2	1	0
名称	RESERVED	DOR	DRDY
R/W	R	R	R

データオーバーラン検出（Bit 1）

新しいデータが上書きされたことを示す。

- **0：通常状態（オーバーランなし）**
- **1：オーバーラン発生**

連続測定モード 1～4 でデータが読み取られないまま次のデータが来ると 1 になる。

ST2 または測定データレジスタ（HXL～TMPS）を読むと自動的に 0 にクリアされる。

測定データが準備完了であることを示す（Bit 0）

- **0：通常状態（データ未準備）**
- **1：新しいデータが準備完了**

シングル測定モード・連続測定モード 1～4・セルフテストモードで、データが得られるとセットされる。

ST2 または測定データレジスタ（HXL～TMPS）を読むと自動的に 0 にクリアされる。

②CNTL2

アドレス＝0x31

ビット	7-5	4	3-0
名称	0	MPDE4	MODE[3:0]
説明	常に 0〈未使用〉	モード設定ビット[4]	モード設定ビット[3:0]
R/W	R/W	R/W	R/W

AK09916（ICM-20948 内部の磁気センサ）の動作モード（電源 ON/OFF、測定モード）を設定するためのレジスタです。

- **MPDE4（Bit 4）**

MODE[4]に相当する上位ビットです。

測定モードを 5 ビットで設定するための「最上位ビット」です。

例えば Self-test mode は 10000b なので、MPDE4=1 になります。

- **MODE[4:0] (Bit3-0)**

  どのモードで磁気センサを動作させるかを決めます。

MODE[4:0]	モード名	説明
00000	Power-down mode	パワーダウンモード 〈電源オフで待機状態〉
00001	Single measurement mode	シングル測定モード 〈1 回測定〉
00010	Continuous measurement mode 1	連続測定モード 1 〈測定周波数は仕様で決まっている〉
00100	Continuous measurement mode 2	連続測定モード 2
00110	Continuous measurement mode 3	連続測定モード 3
01000	Continuous measurement mode 4	連続測定モード 4
10000	Self-test mode	セルフテストモード

７－３． 気圧計（I2C 通信）

①PSR_B2

アドレス＝0x00

ビット範囲	名称	説明	R/W
7-0	PRS23:16	24 ビット圧力データの上位 8 ビット 〈2 の補数〉	R

PRS_B2 は、DPS368 の 24 ビット圧力測定値の最上位バイト(MSB)を保持します。3 バイト分のデータ(PRS_B2、PRS_B1、PRS_B0)を組み合わせて、フル 24 ビットの圧力値として扱います。

2 の補数形式(2`s complement)なので、負の圧力オフセットも扱える形になっています。

FIFO バッファが有効であれば、FIFO からのデータが格納されます。そうでない場合は単独の測定値を保持し、読み出しても値はクリアされません。

②PSR_B1

アドレス＝0x01

ビット範囲	名称	説明	R/W
7-0	PRS15:8	24 ビット圧力データの中位 8 ビット 〈2 の補数〉	R

PRS_B1 は、24 ビット（3 バイト）圧力データの中位バイト（8 ビット）を格納するレジスタです。

データは 2 の補数形式で表現され、気圧センサが測定した圧力値の一部として使用されます。

リセット時は 0x00 に初期化されます。

FIFO が有効な場合は、FIFO に格納された圧力値や温度値を含む場合があります。FIFO が無効な場合は、常に最新の圧力データを保持し、読み取り後もクリアされません。

③PSR_B0

アドレス＝0x02

ビット範囲	名称	説明	R/W
7-0	PRS7:0	24 ビット圧力データの下位 8 ビット 〈2 の補数〉	R

PSR_B0 は DPS368 の 24 ビット圧力データの最下位バイト（8 ビット）を格納するレジスタです。

データは 2 の補数形式で表現され、センサが測定した圧力値の最も下位の部分を構成します。

リセット時は 0x00 に初期化されます。

FIFO が有効な時は FIFO に格納された複数の測定結果を連続的に読み取ることが可能です。FIFO が無効な場合は、常に最新の圧力データが保持され、読み取り後もクリアされません。

④TMP_B2

アドレス＝0x03

ビット範囲	名称	説明	R/W
7-0	TMP23:16	24 ビット温度データの上位 8 ビット 〈2 の補数〉	R

TMP_B2 は、DPS368XTSA1 内部の温度センサで取得した温度データ（24 ビット）の最上位バイトを格納するレジスタです。

DPS368XTSA1 では温度を 24 ビットの 2 の補数形式で表します。負の温度も含めて正しく扱うために、符号拡張をします。

生データはそのままだと単位が不明なので、補数係数（キャリブレーション値）を使い、℃に換算します。

FIFO モードを有効化していない場合は、最新の単一測定値を読みます。FIFO モードを有効化している場合は、FIFO に入っている複数の測定値を順に読みます。

⑤TMP_B1

アドレス＝0x04

ビット範囲	名称	説明	R/W
7-0	TMP15:8	24 ビット温度データの中位 8 ビット 〈2 の補数〉	R

TMP_B2 は、DPS368XTSA1 内部の温度センサで取得した温度データ（24 ビット）の中位バイトを格納するレジスタです。

DPS368XTSA1 では温度を 24 ビットの 2 の補数形式で表します。負の温度も含めて正しく扱うために、符号拡張をします。

生データはそのままだと単位が不明なので、補数係数（キャリブレーション値）を使い、℃に換算します。

FIFO モードを有効化していない場合は、最新の単一測定値を読みます。FIFO モードを有効化している場合は、FIFO に入っている複数の測定値を順に読みます。

⑥TMP_B0

アドレス＝0x05

ビット範囲	名称	説明	R/W
7-0	TMP7:0	24 ビット温度データの下位 8 ビット 〈2 の補数〉	R

TMP_B2 は、DPS368XTSA1 内部の温度センサで取得した温度データ（24 ビット）の下位バイトを格納するレジスタです。

DPS368XTSA1 では温度を 24 ビットの 2 の補数形式で表します。負の温度も含めて正しく扱うために、符号拡張をします。

生データはそのままだと単位が不明なので、補数係数（キャリブレーション値）を使い、℃に換算します。

FIFO モードを有効化していない場合は、最新の単一測定値を読みます。FIFO モードを有効化している場合は、FIFO に入っている複数の測定値を順に読みます。

⑦PRS_CFG

アドレス＝0x06

ビット	7	6-4	3-0
名称	RESERVED	PM_RATE[2:0]	PM_PRC[3:0]
R/W	-	R/W	R/W

- **PM_RATE[2:0] (Bit 6-4)**

  圧力測定を 1 秒間に何回行うかを設定するビット。

  1Hz～128 Hzまで設定可能。

  このレート設定は Background モードにのみ適用されます。

  測定レートを上げるほど応答性が速くなりますが、消費電流も大きくなります。

ビット値	測定レート（1 秒あたりの測定回数）
000	1 measurement/sec
001	2 measurement/sec
010	4 measurement/sec
011	8 measurement/sec
100	16 measurement/sec
101	32 measurement/sec
110	64 measurement/sec
111	128 measurement/sec

- **PM_PRC[3:0] (Bit 3-0)**

  圧力測定のオーバーサンプリングレート（分解能）を設定するビット。

  ビットが大きいほど平均化されるので、SNR が向上し、分解能が上がりますが応答時間が遅くなります。

設定値〈ビット〉	サンプル回数	測定時間の目安（ms）	分解能の目安(PaRMS)
0000	Single（1 回）	3.6	2.5
0001	2 times	5.2	1
0010	4 times	8.4	0.5
0011	8 times	14.8	0.4
0100	16 times	27.6	0.35
0101	32 times	53.2	0.3
0110	64 times	104.4	0.2
0111	128 times	206.8	-

※1xxx は予約ビットなので使用不可です。

※表は「温度測定を行わない場合」の圧力測定レートとオーバーサンプリングの組み合わせ例です。

温度測定を行う場合は、Rate_temperature × Measurement Time_temperature + Rate_pressure × Measurement Time_pressure < 1 秒の制約があります。

温度の測定時間とオーバーサンプリングレートの関係も圧力と同様の傾向になります。

⑧TMP_CFG

アドレス＝0x07

ビット	7	6-4	3-0
名称	TMP_EXT	TMP_RATE[2:0]	TMP_PRC[2:0]
R/W	R/W	R/W	R/W

- **TMP_EXT (Bit 7)**

  内部センサ(0)はチップ内の ASIC 温度を測定

  外部センサ(1)は MEMS 素子(圧力センサ内)に搭載された温度センサを使います。

  キャリブレーションと同じセンサを選択することが推奨されています。

- **TMP_RATE[2:0] (Bit 6-4)**

  温度測定のサンプリングレート（1～128 回/秒）を設定

  レートが高いほどレスポンスが速くなりますが、消費電流が増えます。

  111 はバックグラウンド測定専用です。

設定値〈ビット〉	測定レート
000	1 回/sec
001	2 回/sec
010	4 回/sec
011	8 回/sec
100	16 回/sec
101	32 回/sec
110	64 回/sec
111	128 回/sec

- **TMP_PRC[2:0] (Bit 3-0)**

  オーバーサンプリングレート（分解能）を設定するビット。

  ビットが大きいほどサンプリング数を多く平均化するのでノイズが減り、SNR が向上します。

  但し、応答時間は長くなります。

設定値〈ビット〉	サンプル回数
0000	Single
0001	2 times
0010	4 times
0011	8 times
0100	16 times
0101	32 times
0110	64 times
0111	128 times

※1xxx は予約ビットなので使用不可です。

⑨MEAS_CFG

アドレス＝0x08

ビット	7	6	5	4	3	2-0
名称	COEF_RDY	SENSOR_RDY	TMP_RDY	PRS_RDY	RESERVED	MEAS_CTRL
R/W	R	R	R	R	-	R/W

DPS368XTSA1 の「測定モード設定」と「測定状態監視」をするためのレジスタです。

- **COEF_RDY (Bit 7)**

  起動時に内部のキャリブレーション係数（補正値）が読み出せる状態かを示します。

  0：キャリブレーション係数がまだ準備されていない

  1：キャリブレーション係数が準備完了

- **SENSOR_RDY (Bit 6)**

  センサ内部の自己初期化の完了状態を示すビット

  0：初期化中（内部補正の準備中）

  1：初期化完了

- **TMP_RDY (Bit 5)**

  温度データが新しく取得され、読み取り可能になったことを示すビット

  0：まだ新しいデータがない、または既に読み取られてクリアされた

  1：新しい温度データが準備完了

- **PRS_RDY (Bit 4)**

  圧力データが新しく取得され、読み取り可能になったことを示すビット

  0：まだ新しいデータがない、または既に読み取られてクリアされた

  1：新しい圧力データが準備完了

- **MEAS_CTRL (Bit 2-0)**

  センサの動作モードを決める制御ビット

ビット値	動作モード	説明
000	Standby	待機モード〈停止〉
001	Command	圧力を単発で 1 回測定
010	Command	温度を単発で 1 回測定
011	Command	未使用
100	Command	未使用
101	Background	圧力を連続測定
110	Background	温度を連続測定
111	Background	圧力+温度を連続測定

⑩CFG_REG

アドレス＝0x09

ビット	7	6	5	4	3	2	1	0
名称	INT_HL	INT_FIFO	INT_TMP	INT_PRS	T_SHIFT	P_SHIFT	FIFO_EN	SPI_MODE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- **INT_HL (Bit 7)**

  割り込み信号の極性（SDO ピン）

  0：アクティブ Low

  1：アクティブ High

- **INT_FIFO (Bit 6)**

  FIFO がフルになったときに割り込みを生成するか

  0：割り込み無効

  1：割り込み有効

- **INT_TMP (Bit 5)**

  温度測定が完了したときに割り込みを生成するか

  0：割り込み無効

  1：割り込み有効

- **INT_PRS (Bit 4)**

  圧力測定が完了したときに割り込みを生成するか

  0：割り込み無効

  1：割り込み有効

- **T_SHIFT (Bit 3)**

  温度データの右シフト設定

  0：シフトしない

  1：右に 1 ビットシフト

- **P_SHIFT (Bit 2)**

  圧力データの右シフト設定

  0：シフトなし

  1：右に 1 ビットシフト

- **FIFO_EN (Bit 1)**

  FIFO 機能の有効化

  0：FIFO 無効

  1：FIFO 有効

- **SPI_MODE (Bit 0)**

  SPI の配線モード選択

  0：標準 4 線式 SPI

  1：3 線式 SPI

⑪COEF

アドレス＝0x10～0x21

校正係数レジスタには、補正された圧力値と温度値を計算するために使用される 2 の補数係数が含ま

れています。

係数	アドレス	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
c0	0x10	c0[11:4]							
c0/c1	0x11	c0[3:0]				c1[11:8]			
c1	0x12	c1[7:0]							
c00	0x13	c00[19:12]							
c00	0x14	c00[11:4]							
c00/c10	0x15	c00[3:0]				c10[19:16]			
c10	0x16	c10[15:8]							
c10	0x17	c10[7:0]							
c01	0x18	c01[15:8]							
c01	0x19	c01[7:0]							
c11	0x1A	c11[15:8]							
c11	0x1B	c11[7:0]							
c20	0x1C	c20[15:8]							
c20	0x1D	c20[7:0]							
c21	0x1E	c21[15:8]							
c21	0x1F	c21[7:0]							
c30	0x20	c30[15:8]							
c30	0x21	c30[7:0]							

※キャリブレーション係数レジスタのデータから小数点以下の数値を生成します。

この補数係数テーブルは DPS368XTSA1 が出力する補正済みの圧力値・温度値を計算する際に使用します。

各係数は 2 の補数係式で保存されており、バイト連結後に符号拡張する必要があります。

係数はそれぞれ役割が異なり、例えば c0 は温度オフセット、c00 は圧力スケール補正に用います。

使用する際は、[ビット範囲]に従ってレジスタから値を読み出し、結合・符号拡張して計算式に組み込んでください。

⑫COEF_SRCE

アドレス＝0x28

ビット	7	6-0
名称	TMP_COEF_SRCE	RESERVED
R/W	R	-

補正係数（キャリブレーションデータ）がどの温度センサを基準に算出されているかを示す読み取り専用のフラグです。

ビット値	意味
0	内部温度センサ〈ASIC 内蔵〉の温度を基準に補正係数が生成されている
1	MEMS 圧力素子の温度センサを基準に補正係数が生成されている

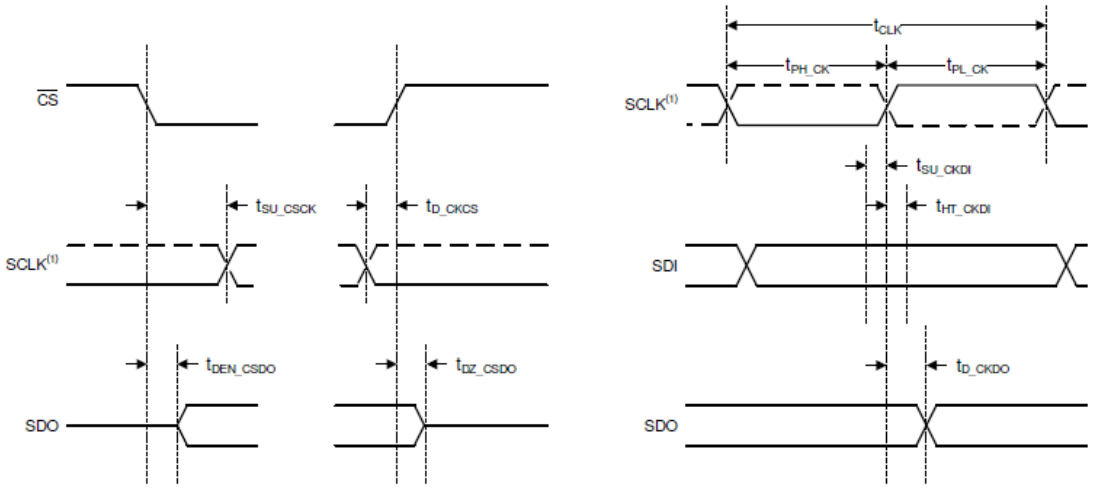
DPS368XTSA1 には温度センサが２種類あります。

- ASIC 内蔵の内部温度センサ
- MEMS 素子（圧力センサ内）に付属の外部温度センサ

８．使用方法

MPC-RASAR-ADSENSOR を Raspberry Pi に接続します。

８－１． ADC（SPI 通信）



(1) The SCLK polarity, launch edge, and capture edge depend on the SPI protocol selected.

図 2. SPI-Compatible Serial Interface Timing

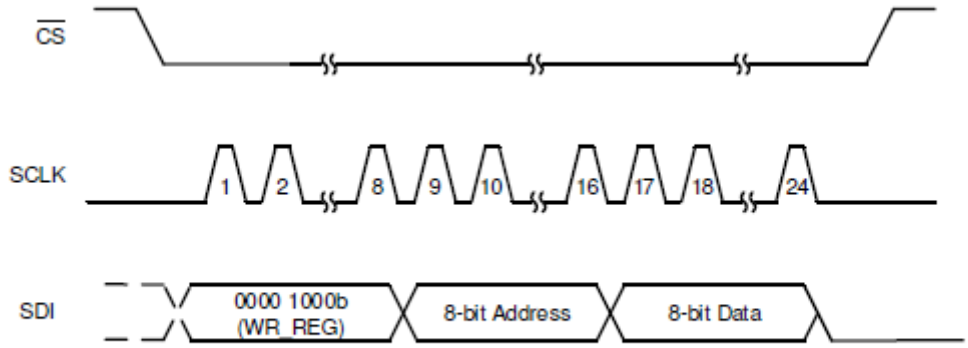


図 13. Register Write Operation

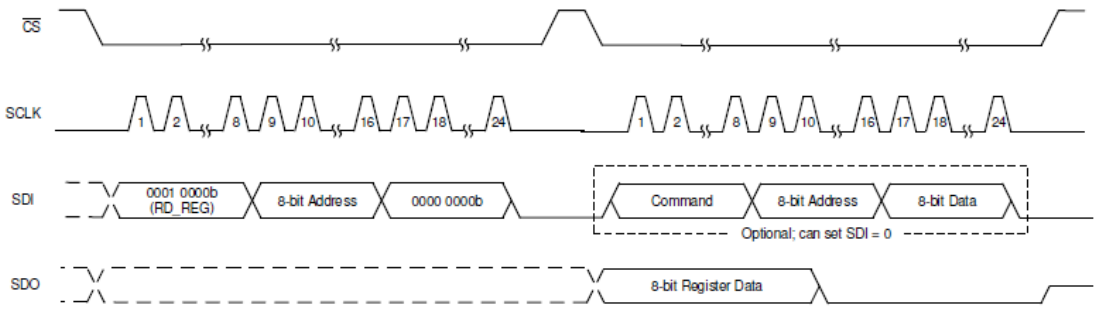


図 14. Register Read Operation

ADS7028IRTER(TI)のデータシートより引用

本プログラムは、ADS7028IRTER の各アナログ入力チャネルの電圧を SPI 通信で取得し、
コンソールに電圧値（V）を表示する Python プログラムです。

①使用構成

接続：

・SPIO MISO → SDO

・SPIO MOSI → SDI

・SPIO SCLK → SCLK

・SPIO CE0 → CS

・Vref → 5.00V(基準電圧)

SPI 設定：

・クロック周波数：最大 24MHz

・SPI モード：Mode 0

プログラム例：

```
Vref = 5.00
spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 24000000
spi.mode = 0
```

②初期設定フロー

項目	アドレス	内容
PIN_CFG	0x05	ピン設定〈アナログ入力〉
SEQUENCE_CFG	0x10	シーケンス設定〈マニュアルモード〉
OPMODE_CFG	0x04	動作モード設定〈通常動作〉

本説明書のレジスタ解説を参照

プログラム例：

```
spi.xfer2([0x08, 0x05, 0x00])
spi.xfer2([0x08, 0x10, 0x00])
spi.xfer2([0x08, 0x04, 0x00])
```

spi.xfer2()は、Python の spidev モジュールに含まれる SPI 通信でデータを送受信するためのメソッドです。

◎一般的な 3 バイト構造パターン

バイト	例	説明
1	コマンド＋R/W ビット	書き込み or 読み出し、アクセスモード
2	レジスタアドレス	どのレジスタか
3	データ or ダミー	書き込みなら設定値、読み出しならダミー

③チャネル選択とデータ取得フロー

チャネル選択：

・spi.xfer2([0x08, 0x11, チャネル番号])

・0x11 は CHANNEL_SEL レジスタ

本説明書のレジスタ解説を参照

データ取得：

・spi.xfer2([0x10, 0x00, 0x00])

・取得後、上位 12bit を抽出し、Vref に基づいて電圧に換算

本説明書のレジスタ解説を参照

プログラム例：

```
spi.xfer2([0x08, 0x11, 0x00)
spi.xfer2([0x10, 0x00, 0x00])
```

④電圧換算方法

・ADS7028IRTER は 12bit の SAR ADC です。
入力されたアナログ電圧(0V～Vref)を 2^12 = 4096 段階に分割して 0～4095 のデジタル値に変換します。

・SPI 通信で受信した 16bit のデータのうち、
raw_12bit = ((raw_high << 8) | raw_low) >> 4 により必要な上位 12bit だけを抽出して、これをもとに電圧に換算します。

プログラム例：

```
raw_high = adc[0]
raw_low = adc[1]
raw_12bit = ((raw_high << 8) | raw_low) >> 4
voltage = (raw_12bit / 4095.0) * Vref
```

８－２． センサ（I2C 通信）

Single-Byte Write Sequence

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Burst Write Sequence

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Burst Read Sequence

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

ICM-20948(TDK)のデータシートより引用

I2C 用語

信号	説明
S	開始条件：SCL が High のときに SDA が High から Low に変化する
AD	スレーブ I2C アドレス
W	書き込みビット(0)
R	読み取りビット(1)
ACK	確認応答：9 番目のクロックサイクルで SCL ラインが High である間に SDA ラインは Low になります。
NACK	非確認応答：SDA ラインは 9 番目のクロックサイクルで High のままです。
RA	ICM-20948 内部レジスタアドレス
DATA	送信または受信したデータ
P	停止条件：SCL が High のときに SDA が Low から High になる。

８－２－１． 加速度センサ（I2C 通信）

本プログラムは、ICM-20948 センサモジュールの 3 軸加速度データを I2C 通信で連続取得し、リアルタイムにシェル表示する Python プログラムです。

①システム構成

接続方法：

・I2C アドレス：0x68 or 0x69

使用ライブラリ：

・smbus：I2C 通信用

プログラム例：

```
import smbus2
ICM20948_ADDR = 0x69
```

②初期化処理

・PWR_MGMT_1(0x06)に 0x01 を書き込む

　→デバイスをスリープ解除・クロック設定

・WHO_AM_I(0x00)を読み取り、ICM-20948 が正しく接続されているか確認

　→返り値が 0xEA なら正常認識

本説明書のレジスタ解説を参照

プログラム例：

```
REG_PWR_MGMT_1 = 0x06
REG_WHO_AM_I = 0x00

bus.write_byte_data(ICM20948_ADDR, 0x06, 0x01)

who_am_i = bus.read_byte_data(ICM20948_ADDR, 0x00)
        if who_am_i == 0xEA:
            print("ICM-20948 detected (Acceleration Mode).")
        else:
            print("ICM-20948 not detected.")
```

③加速度データ取得の流れ

・ACCEL_XOUT_H(0x2D)から 6 バイト連続読み出し

　→X/Y/Z 各軸 2 バイト(MSB/LSB)

・2 バイトを 16bit の符号付き整数に変換

　→convert_raw_data()

・生データは 2 の補数形式

・0x8000 以上なら負の値に補正するため value-65536 の処理が必須

・物理単位に換算

　→±2g レンジの場合、スケールファクタは 1LSB=1/16384g

　→×9.81 で m/s²に換算

本説明書のレジスタ解説を参照

プログラム例：

```
REG_ACCEL_XOUT_H = 0x2D

def convert_raw_data(raw_data):
    """2バイトの生データを16ビット整数に変換"""
    value = (raw_data[0] << 8) | raw_data[1]
    return value if value < 32768 else value - 65536

def read_accel():
    """加速度データを取得（単位：m/s²）"""
    raw_data = read_sensor_data(0x2D, 6)
    ax = convert_raw_data(raw_data[0:2]) / 16384.0 * 9.81  # X軸 (m/s²)
    ay = convert_raw_data(raw_data[2:4]) / 16384.0 * 9.81  # Y軸 (m/s²)
    az = convert_raw_data(raw_data[4:6]) / 16384.0 * 9.81  # Z軸 (m/s²)
    return ax, ay, az
```

④リアルタイム表示

・シェルに現在の加速度を表示

・単位は m/s²

プログラム例：

```
def display_sensor_data(ax, ay, az):
    print(f"Acceleration: X={ax:.2f} m/s², Y={ay:.2f} m/s², Z={az:.2f} m/s²")
```

８－２－２．　角速度センサ（I2C通信）

本プログラムは、ICM-20948センサモジュールの3軸角速度データをI2C通信で連続取得し、リアルタイムにシェル表示するPythonプログラムです。

①システム構成

８－２－1.　加速度センサ参照

②初期化処理

８－２－1.　加速度センサ参照

③角速度データ取得の流れ

・GYRO_XOUT_H(0x33)から 6 バイト連続読み出し

・2 バイトずつで各軸の角速度データ(MSB＋LSB)を構成

・2 の補数形式

→convert_raw_data()で 16bit 符号付き整数に変換

・フルスケール±°/s 時は、1LSB=約 1/131°/s

・/131.0 で実際の角速度(°/s)に変換

本説明書のレジスタ解説を参照

プログラム例：

```
REG_GYRO_XOUT_H = 0x33

def convert_raw_data(raw_data):
    """2 バイトの生データを 16 ビット整数に変換"""
    value = (raw_data[0] << 8) | raw_data[1]
    return value if value < 32768 else value - 65536

def read_gyro():
    """ジャイロスコープデータを取得（単位：°/s）"""
    raw_data = read_sensor_data(0x33, 6)
    gx = convert_raw_data(raw_data[0:2]) / 131.0  # X 軸（°/s)
    gy = convert_raw_data(raw_data[2:4]) / 131.0  # Y 軸（°/s)
    gz = convert_raw_data(raw_data[4:6]) / 131.0  # Z 軸（°/s)
    return gx, gy, gz
```

④リアルタイム表示

・シェルに現在の角速度を表示

・単位は°/s

プログラム例：

```
def display_sensor_data(gx, gy, gz):
    print(f"Gyroscope:  X={gx:.2f} °/s, Y={gy:.2f} °/s, Z={gz:.2f} °/s")
```

８－２－３． 地磁気センサ（I2C 通信）

本プログラムは、ICM-20948 に内蔵された磁気センサ(AK09916)を I2C マスター経由で制御し、3 軸磁場データ(X/Y/Z)をμT 単位で取得し、リアルタイムにシェル表示する Python プログラムです。

①システム構成

接続方法：

・I2C アドレス(ICM-20948)：0x68 or 0x69

・I2C アドレス(AK09916)：0x0C

使用ライブラリ：

・smbus：I2C 通信用

プログラム例：

```
import smbus
ICM20948_ADDR = 0x69
AK09916_ADDRESS = 0x0C
```

②初期化処理

Bank 0 初期化：

・電源 ON・スリープ解除

・I2C マスター有効化

Bank 3 初期化：

・I2C マスタークロック設定(400kHz)

・I2C_SLV4 経由で AK09916 に書き込み(測定モードの設定)

本説明書のレジスタ解説を参照

プログラム例：

```
set_bank(0)
    bus.write_byte_data(ICM20948_ADDRESS, PWR_MGMT_1, 0x01)
    bus.write_byte_data(ICM20948_ADDRESS, USER_CTRL, 0x20)

set_bank(3)
    bus.write_byte_data(ICM20948_ADDRESS, I2C_MST_CTRL, 0x17)
    bus.write_byte_data(ICM20948_ADDRESS, I2C_SLV4_ADDR, 0x0C)
    bus.write_byte_data(ICM20948_ADDRESS, I2C_SLV4_REG, 0x31)
    bus.write_byte_data(ICM20948_ADDRESS, I2C_SLV4_DO, 0x08)
    bus.write_byte_data(ICM20948_ADDRESS, I2C_SLV4_CTRL, 0x89)
```

③測定モード設定

・AK09916 に 1 回測定モード停止指令を送信 (0x00)

・以降、必要に応じて測定モード制御が可能に

本説明書のレジスタ解説を参照

プログラム例：

```
def set_measurement_mode():
    bus.write_byte_data(ICM20948_ADDRESS, I2C_SLV0_ADDR, 0x8C)
    bus.write_byte_data(ICM20948_ADDRESS, I2C_SLV0_REG, MAG_CNTL2)
    bus.write_byte_data(ICM20948_ADDRESS, I2C_SLV0_DO, 0x00)
    bus.write_byte_data(ICM20948_ADDRESS, I2C_SLV0_CTRL, 0x88)
```

④データ読み取り設定：

・AK09916 の ST1 レジスタ(0x10)から 8 バイト(ST1+XYZ+ST2)を連続取得

・データは ICM-20948 の EXT_SLV_SENS_DATA_00(0x3B)に格納される

本説明書のレジスタ解説を参照

プログラム例：

```
def prepare_magnetometer_read():
    bus.write_byte_data(ICM20948_ADDRESS, I2C_SLV0_ADDR, 0x8C)
    bus.write_byte_data(ICM20948_ADDRESS, I2C_SLV0_REG, MAG_ST1)
    bus.write_byte_data(ICM20948_ADDRESS, I2C_SLV0_CTRL, 0x89)
```

⑤地磁気データ取得の流れ

AK09916(磁気センサー)から読み取った X/Y/Z 軸の磁場データを

・16 ビットの符号付き整数に変換し、

・μT 単位に変換

本説明書のレジスタ解説を参照

プログラム例：

```
data = bus.read_i2c_block_data(ICM20948_ADDRESS, EXT_SLV_SENS_DATA_00, 8)
                x_raw = (data[1] | (data[2] << 8))
                y_raw = (data[3] | (data[4] << 8))
                z_raw = (data[5] | (data[6] << 8))

                x = x_raw if x_raw < 32768 else x_raw - 65536
                y = y_raw if y_raw < 32768 else y_raw - 65536
                z = z_raw if z_raw < 32768 else z_raw - 65536
```

```
x_ut = x * 0.15
y_ut = y * 0.15
z_ut = z * 0.15
```

⑥リアルタイム表示

・シェルに現在の地磁気を表示

・単位はμT

プログラム例：

```
print(f"Magnetometer: X={x_ut:.2f} μT, Y={y_ut:.2f} μT, Z={z_ut:.2f} μT")
```

８－３． 気圧計（I2C 通信）

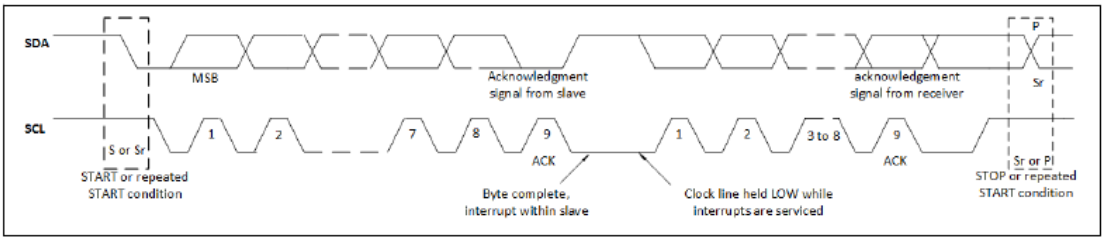


Figure 11　　I2C timing diagram

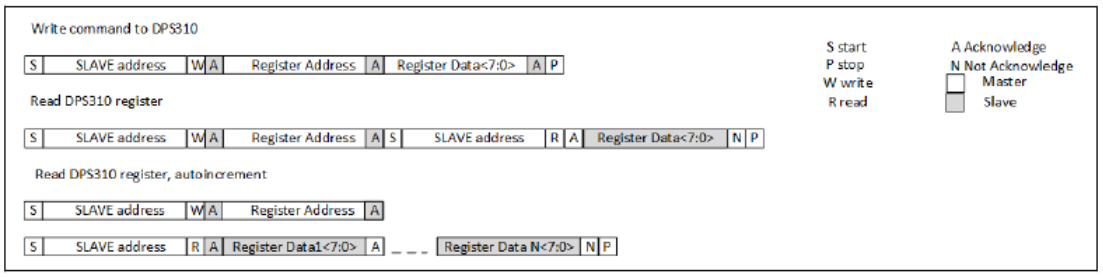


Figure 12　　I2C write and read commands

DPS368XTSA1(infineon)のデータシートより引用

本プログラムは、DPS368 センサ使って気圧と温度を測定し、補正後の値を表示する Python プログラムです。

①ライブラリのインポート

```
import smbus
from time import sleep
```

②２の補数変換関数（getTwosComplement）

DPS368 が返す 24 ビットの生データは 2 の補数であるため、補正する関数を定義します。

プログラム例：

```
def getTwosComplement(raw_val, length):
    if raw_val & (1 << (length - 1)):
        val = raw_val - (1 << length)
    return val
```

③DPS368 クラスの初期化

センサの操作（初期化・補正・測定）をクラスとしてまとめます。

温度補正とオーバーサンプリング設定を行います。

プログラム例：

```
class DPS368:
    __bus = smbus.SMBus(1)
    __addr = 0x77
    __kP = 1040384  # 圧力スケール係数（64 倍オーバーサンプリング）
    __kT = 1040384  # 温度スケール係数（64 倍オーバーサンプリング）

def __init__(self):
        self.__correctTemperature()
        self.__setOversamplingRate()
```

・__correctTemperature()：温度誤差表示対策

・__setOversamplingRate()：温度・気圧の設定

④温度補正処理

起動直後に異常に高い温度を返すことがある為、起動時に初期化手順を実行します。

但し、異常な温度が出なければ、補正処理は省略しても問題ありません。

プログラム例：

```
def __correctTemperature(self):
    DPS368.__bus.write_byte_data(DPS368.__addr, 0x0E, 0xA5)
    DPS368.__bus.write_byte_data(DPS368.__addr, 0x0F, 0x96)
    DPS368.__bus.write_byte_data(DPS368.__addr, 0x62, 0x02)
    DPS368.__bus.write_byte_data(DPS368.__addr, 0x0E, 0x00)
    DPS368.__bus.write_byte_data(DPS368.__addr, 0x0F, 0x00)
```

・0x0E、0x0F：内部制御キー

・0x62：温度補正モード設定

※Infineon 社の GitHub リポジトリ及び技術フォーラムを参照

⑤オーバーサンプリング設定

気圧・温度の精度を上げるため、オーバーサンプリング設定を行います。これにより、ノイズの少ない高精度なデータが取得できます。

本説明書のレジスタ解説を参照

プログラム例：

```
def __setOversamplingRate(self):
    DPS368.__bus.write_byte_data(DPS368.__addr, 0x06, 0x26)
```

```
    DPS368.__bus.write_byte_data(DPS368.__addr, 0x07, 0xA6)
    DPS368.__bus.write_byte_data(DPS368.__addr, 0x08, 0x07)
    DPS368.__bus.write_byte_data(DPS368.__addr, 0x09, 0x0C)
```

⑥測定処理

・生データの取得

DPS368 は、気圧・温度をそれぞれ 24 ビットの 2 の補数形式で出力します。

本説明書のレジスタ解説を参照

プログラム例：

気圧取得

```
def __getRawPressure(self):
        p1 = DPS368.__bus.read_byte_data(DPS368.__addr, 0x00)
        p2 = DPS368.__bus.read_byte_data(DPS368.__addr, 0x01)
        p3 = DPS368.__bus.read_byte_data(DPS368.__addr, 0x02)
        p = (p1 << 16) | (p2 << 8) | p3
        p = getTwosComplement(p, 24)
        return p
```

温度取得

```
def __getRawTemperature(self):
        t1 = DPS368.__bus.read_byte_data(DPS368.__addr, 0x03)
        t2 = DPS368.__bus.read_byte_data(DPS368.__addr, 0x04)
        t3 = DPS368.__bus.read_byte_data(DPS368.__addr, 0x05)
        t = (t1 << 16) | (t2 << 8) | t3
        t = getTwosComplement(t, 24)
        return t
```

・スケーリング（正規化）

読み取った生の気圧・温度を、DPS368 のデータシートに基づいてスケーリングします。

プログラム例：

```
def calcScaledPressure(self):
        raw_p = self.__getRawPressure()
        scaled_p = raw_p / DPS368.__kP
        return scaled_p
```

```
def calcScaledTemperature(self):
        raw_t = self.__getRawTemperature()
        scaled_t = raw_t / DPS368.__kT
        return scaled_t
```

・補正演算（キャリブレーション）

DPS368 内部には個体差を補正するための係数（c00, c10, c01 など）が格納されています。これらを読み取り、補正演算を行います。

本説明書のレジスタ解説を参照

プログラム例：

温度補正

```
def calcCompTemperature(self, scaled_t):
        c0, c1 = self.__getTemperatureCalibrationCoefficients()
        comp_t = c0 * 0.5 + scaled_t * c1
        return comp_t
```

気圧補正

```
def calcCompPressure(self, scaled_p, scaled_t):
        c00, c10, c20, c30, c01, c11, c21 = self.__getPressureCalibrationCoefficients()
        comp_p = (c00 + scaled_p * (c10 + scaled_p * (c20 + scaled_p * c30))
                + scaled_t * (c01 + scaled_p * (c11 + scaled_p * c21)))
        return comp_p
```

・補正式は DPS368 のデータシートに記載されている近似式をそのまま使用しています。

・温度と気圧に応じて非線形補正が行われます。

・最終測定値の取得

プログラム例：

単発測定

```
def measureTemperatureOnce(self):
        t = self.calcScaledTemperature()
        temperature = self.calcCompTemperature(t)
        return temperature
```

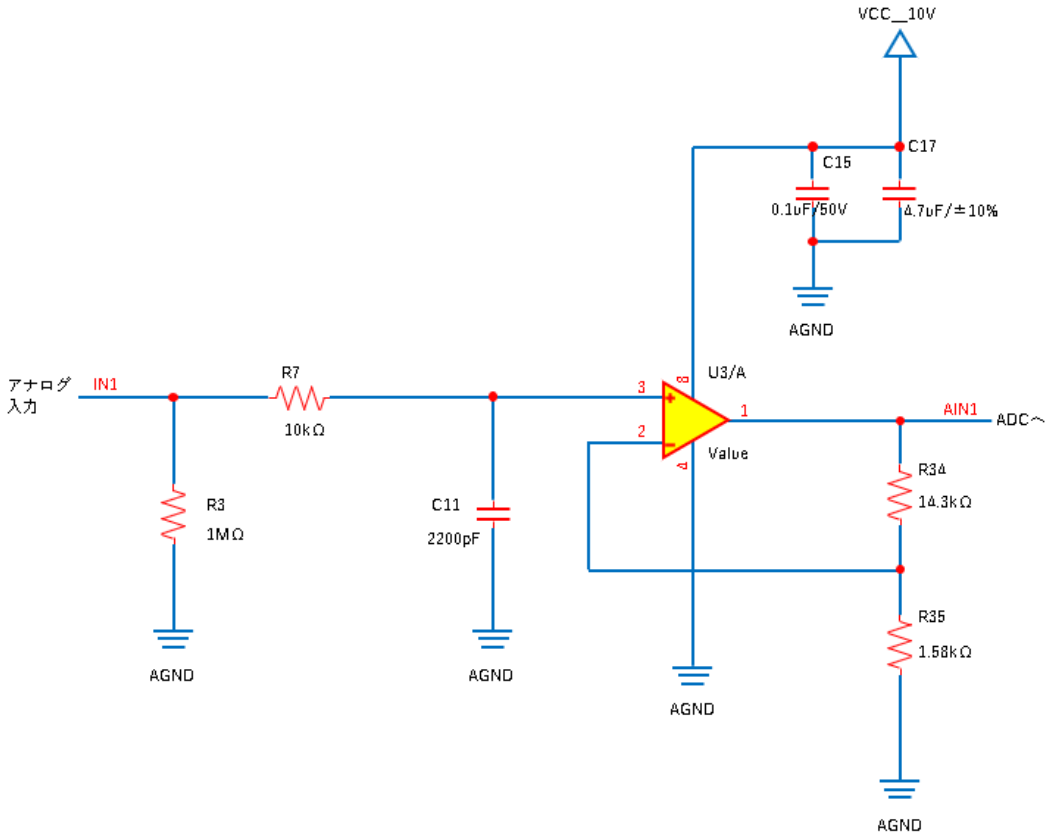
```
def measurePressureOnce(self):
        p = self.calcScaledPressure()
        t = self.calcScaledTemperature()
        pressure = self.calcCompPressure(p, t)
        return pressure
```

平均化測定（例：10 回）

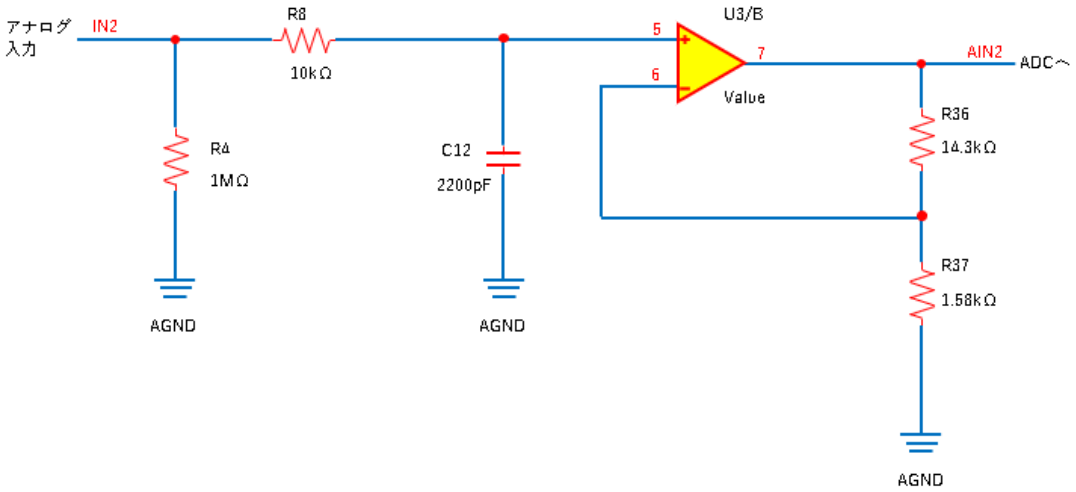
```
def measurePressureAndTemperatureAverage(self, num_measurements=10):
        total_pressure = 0
        total_temperature = 0

        for _ in range(num_measurements):
            total_pressure += self.measurePressureOnce()
            total_temperature += self.measureTemperatureOnce()
            sleep(0.1)

        average_pressure = total_pressure / num_measurements
        average_temperature = total_temperature / num_measurements

        return average_pressure / 100, average_temperature
```

９．入出力回路説明

ＡＤ入力回路



上記は 0CH の回路図です。2、4 及び 6CH も同様です。



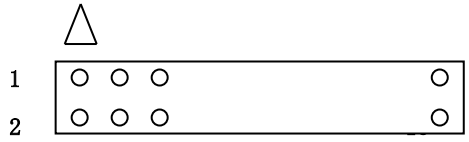
上記は 1CH の回路図です。3、5 及び 7CH も同様です。

１０．ピンアサイン

CN1　アナログ入力コネクタ

ピン	信号	ピン	信号
1	IN1(入力)	2	GND
3	IN2(入力)	4	GND
5	IN3(入力)	6	GND
7	IN4(入力)	8	GND
9	IN5(入力)	10	GND
11	IN6(入力)	12	GND
13	IN7(入力)	14	GND
15	IN8(入力)	16	GND

CN1 コネクタピン配置



１１．電気的特性

Raspberry Pi インターフェース

ピン	信号	ピン	信号
1		2	VCC_5V
3	GPIO2(SDA)	4	
5	GPIO3(SCL)	6	GND
7		8	
9		10	
11		12	
13		14	
15		16	
17		18	
19	GPIO10(MOSI)	20	
21	GPIO9(MISO)	22	
23	GPIO11(SCLK)	24	GPIO8(CE0)
25		26	
27		28	
29		30	
31		32	
33		34	
35		36	
37		38	
39		40	

１２．規格

ROHS 指令準拠

MPC-RASAR-ADSENSOR 取扱説明書

EMBEDDED TECHNOLOGY
Corporation

株式会社エンベデッドテクノロジー

〒578-0946

大阪府東大阪市瓜生堂 3 丁目 8-13

奥田ビル 2F

TEL：06-6224-1137

FAX：06-6224-1138

http://www.emb-tech.co.jp/