XENSIV™ PAS CO2 のプログラミングガイド

対象アプリケーション: デマンド換気制御システム

本書について

適用範囲と目的

このアプリケーションノートはプログラミングのスタートガイドとして、マイクロコントローラによって駆動される XENSIV™ PAS CO2 センサーの設定と通信に関する内容を記述します。主な焦点は、インフィニオン PSOC™ 6 WiFi-BT Pioneer キットおよび Arduino Due を使用して I^2C 経由で PAS CO2 センサーの基本的な測定や初期設定を行う方法を簡単に紹介します。

対象者

HVAC システムのアプリケーションエンジニア、システムエンジニア、およびシステムアーキテクト

目次

本書について 1
目次 2
1 はじめに 3
1.1 I²C バスの紹介 4
1.2 I²C 通信プロトコル 4
2 I²C アプリケーション回路 6
3 初期化シーケンス 7
4 PSOC™ 6 WiFi-BT Pioneer キットのクイックスタート 8
4.1 ブリッジコントロールパネル 9
4.2 測定を開始するための基本コード 11
4.3 追加機能 13
5 Arduino Due のクイックスタート 15
5.1 Arduino IDE 15
5.2 測定を開始するための基本的な Arduino コード 17
5.3 Arduino ライブラリ 21
6 UART インターフェース 23
6.1 書き込みトランザクション 23
6.2 読み出しトランザクション 24
6.3 Arduino の例 25
7 PWM インターフェース 30
改訂履歴 34
免責事項 35

1 はじめに

XENSIV™ PAS CO2 センサーは、前例のない小さなフォームファクターで実際の二酸化炭素 (CO_2) を測定するセンサーです。光音響分光法 (PAS) のコンセプトに基づいて設計された本センサーは、既存の市販の実際の NDIR CO2 センサーに比べて 75%以上の省スペース化を実現しています。直接に ppm の数値を読み出し、SMD 対応でシンプルな設計により、少量から大量生産のアプリケーションにおいても、顧客のシステムに迅速かつ簡単に統合できます。

光音響の原理は、100 年以上前にさかのぼることができます。1880 年にアレクサンダー・グラハム・ベルによって最初に発見されました。光音響効果は、物質試材料内での光吸収に続いて音波(圧力変化)が形成される現象です。音声信号は、マイクなどの検出器によって量子化されます。この効果を得るためには、光の強度を変化させる必要があります。PAS 方式のガスセンサーは、ガスが赤外線スペクトルの特定の波長で光を吸収するという原理に基づいています。例えば、CO_2分子は $\lambda = 4.2\ \mu m$ の波長で強い吸収能力を持っています。

XENSIV™ PAS CO2 センサー モジュールは、PAS トランスデューサー, 信号処理用マイクロコントローラー, アルゴリズム, および MOSFET が同じ PCB 上に実装されています。ブロック図に示すように、PAS トランスデューサは、i)MOSFET によって周期的に出力される黒体放射を有する独自の赤外線エミッタ。ii) 湿度を含む他のガスと比較してセンサーの選択性を大幅に向上させるために、CO_2 固有の波長 $\lambda = 4.2\ \mu m$ を通過させる狭帯域光学フィルター。そして、iii)CO_2分子によって生成される圧力変化を検出する、インフィニオンの高 SNR (信号対雑音比) MEMS マイクロフォン XENSIV™ IM69D130 で構成されています。すべての部品は、インフィニオンの高品質ガイドラインに従って社内で開発・設計されています。そのため、このセンサーは、インフィニオンの MEMS 設計と音響性能における輝かしい実績の恩恵を受けており、価格/性能比でクラス最高を実現しています。

XENSIV™ PAS CO2 センサーは、スマートホームやビルの自動化だけでなく、空気清浄機, サーモスタット, 気象観測所, およびパーソナルアシスタントなどのさまざまな室内空気品質センシング用 IoT デバイスにも最適です。このセンサーにより、エンドユーザーは、周囲の空気の質をタイムリーかつエネルギー効率の高い方法で追跡、認識、改善することができます。

以下では、I^2C インターフェースについて詳細に説明し、アプリケーションノートの最後に UART および PWM インターフェースについて説明します。

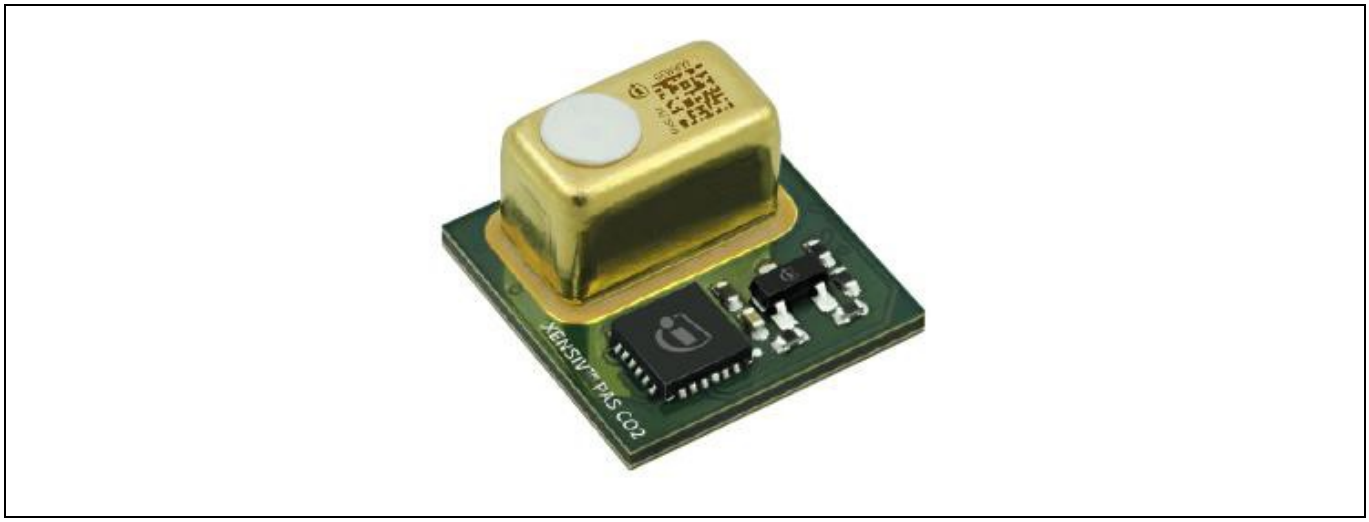


Figure 1 XENSIV™ PAS CO2 センサー

1 はじめに

1.1 I²C バスの紹介

I²C バスは双方向の 2 ラインバスであり、このプロトコルをサポートするあらゆる種類の集積回路間の通信を、ハードウェアまたはソフトウェアによって可能にします。このような IC の例としては、LCD コントローラ, EEPROM, RAM, データコンバータ, または汎用マイクロコントローラがあります。このプロトコルの主な利点は、Figure 2 に示すように、2 ラインのインターフェースです。

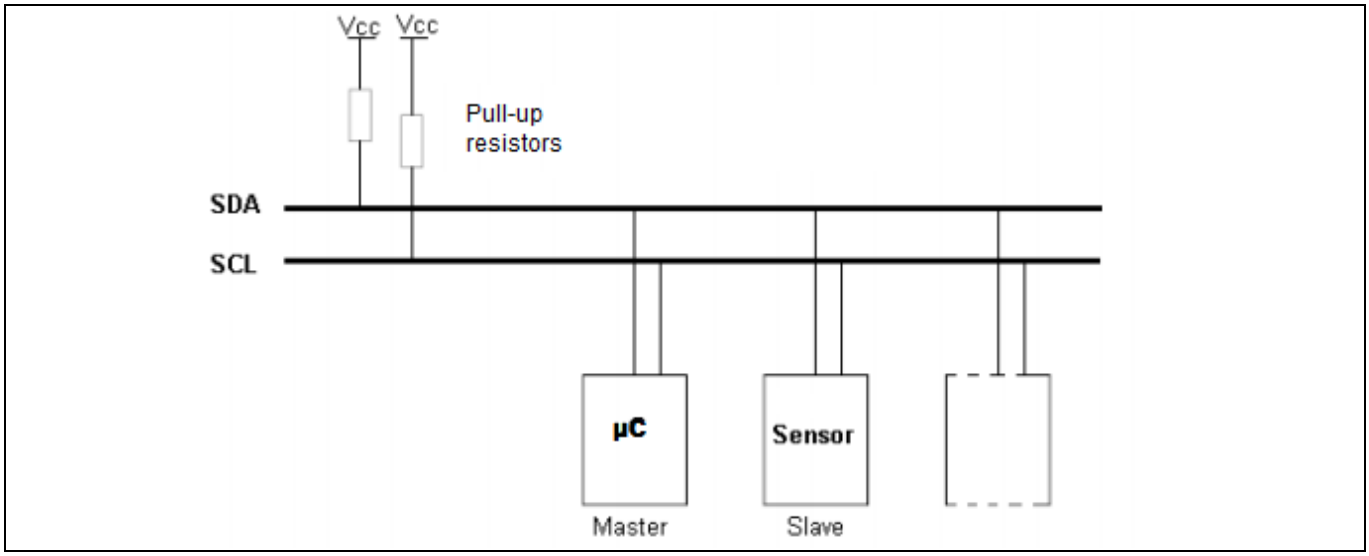


Figure 2 I²C バスの設定例

1.2 I²C 通信プロトコル

「マスター」という言葉は、転送を開始および終了し、ライン SCL にクロック信号も提供するデバイスを指します。マスターデバイスは、送信機または受信機として動作します。

各転送の開始時に、スレーブは独自の一意のアドレスでアドレス指定されます。転送は、開始条件、データビット、確認応答ビット、および場合によっては停止条件で構成されます。この概念を Figure 3 に示します。

開始条件は、SCL が High のときの SDA の立ち下りエッジによって定義されます。停止条件は、SCL ラインが High のときの SDA の立ち上りエッジによって定義されます。データを送信する際、クロックが High の間、SDA ラインでの変更は許可されません。そうでない場合、停止または開始の条件が発生します！これを回避するには、SCL ラインが Low のときだけ SDA でデータを変更する必要があります。

8 つのデータビットの送信後、マスターは SDA ラインを High に設定し、スレーブは SDA ラインをグランドにプルダウンして転送を確認します。これは、転送が成功したことを示します。

マスター読み出しモードはまったく同じではありません。マスターは引き続きクロックを供給しますが、スレーブは SDA ラインで (マスターから要求された) データを送信します。送信の終了時に、マスターは確認応答しません (SDA が設定され、High のままです)。

詳細については、オリジナルの I²C バス仕様 (UM10204 Rev 6, NXP Semiconductors) を参照してください。

XENSIV™ PAS CO2 センサーのステータスを読み出す例を Figure 4 に示します。基本レジスタの紹介は、このアプリケーションノートの後半で行います。XENSIV™ PAS CO2 センサーの利用可能なすべてのレジスタと機能の詳細については、別のアプリケーションノートを参照してください。

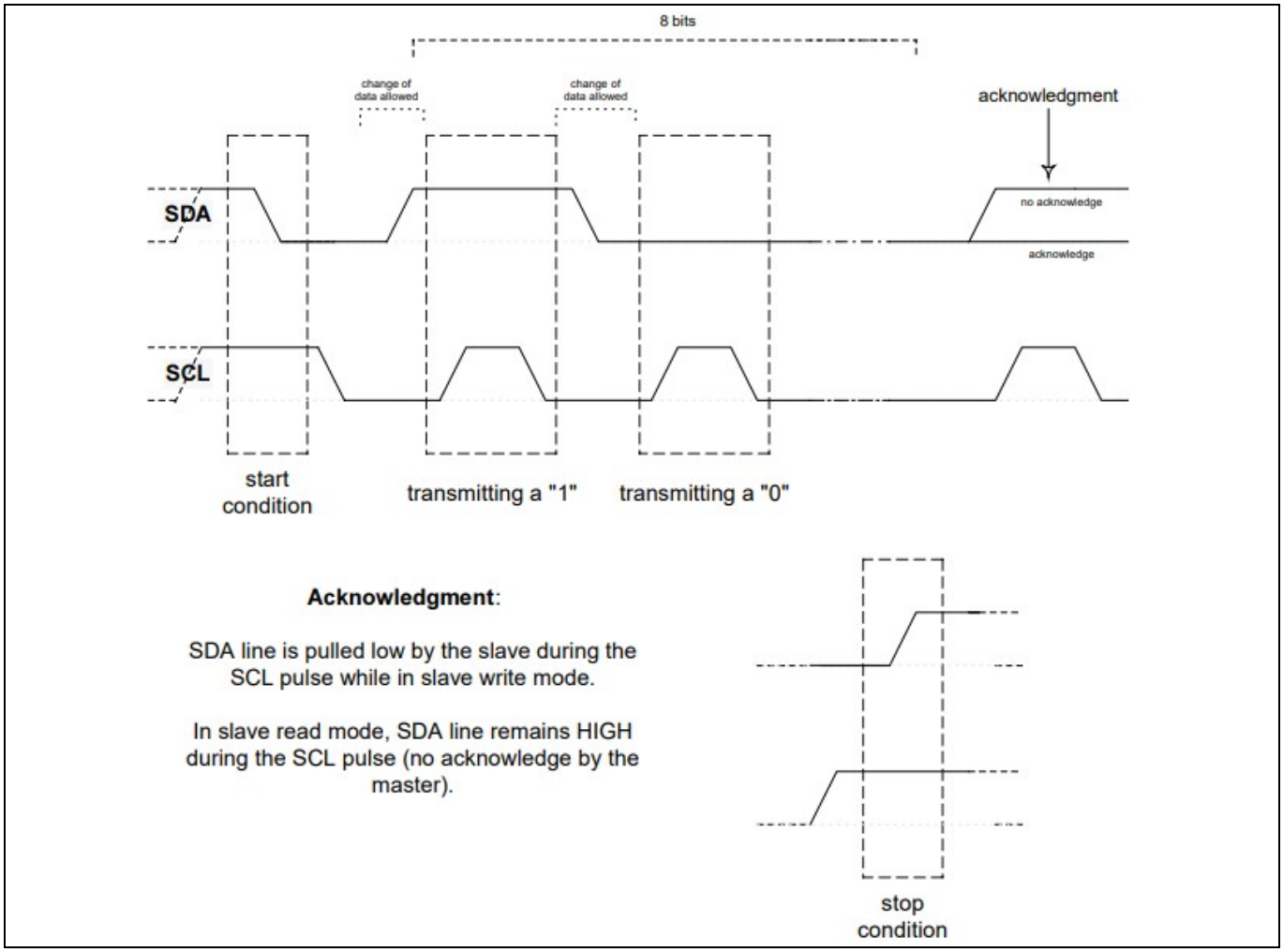


Figure 3　I²C トランザクション例

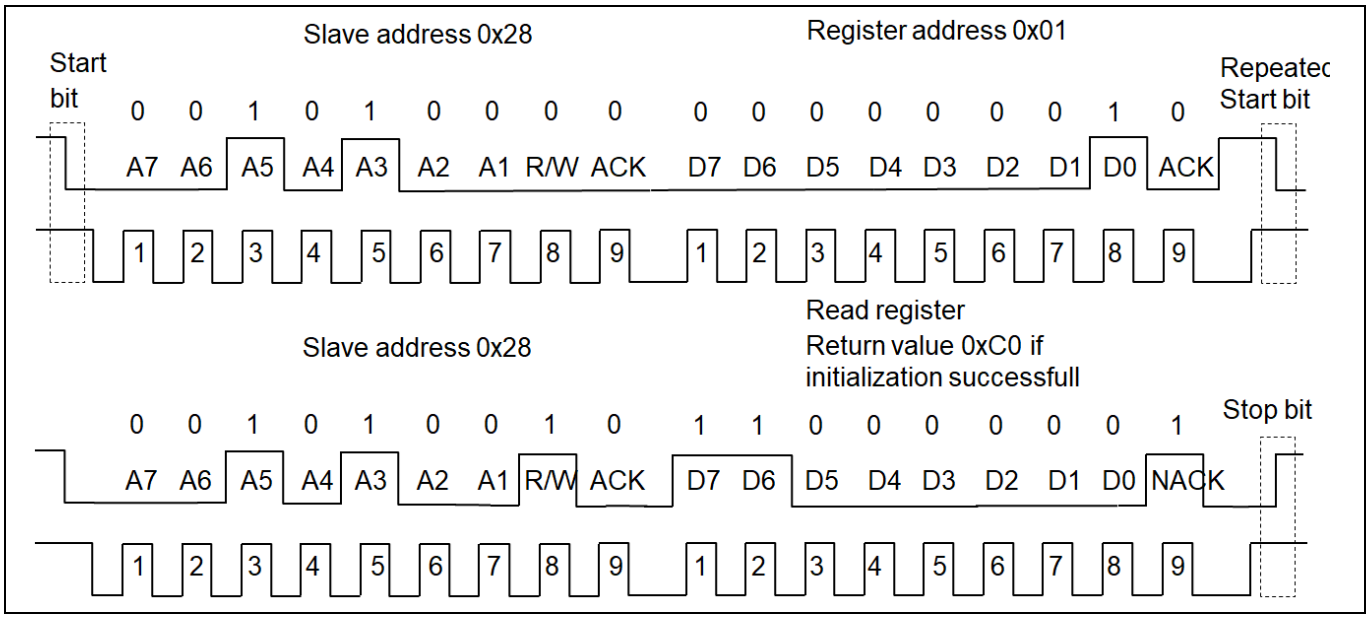


Figure 4　センサーの状態を読み出すための I²C トランザクション

2 I²C アプリケーション回路

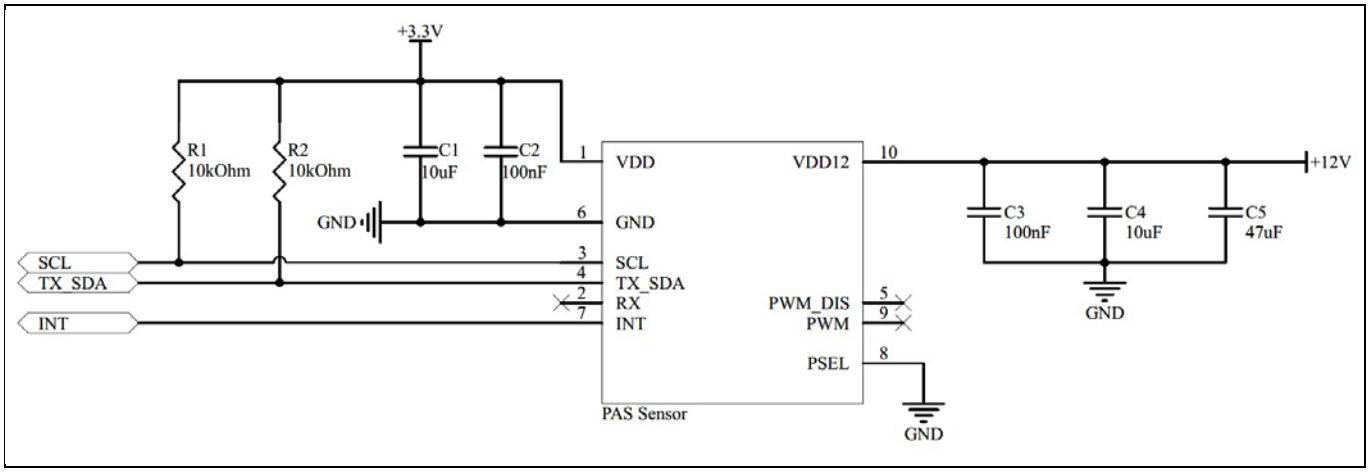


Figure 5 応用回路例

Table 1 ピンの説明

ピン	記号	種類	機能
1	VDD	電源 (3.3 V)	3.3 V デジタル電源
2	RX	入力	UART 受信ピン (3.3 V ドメイン)
3	SCL	入力/出力	I²C クロックピン (3.3 V ドメイン) [3]
4	TX_SDA	入力/出力	UART トランスミッタピン/I²C データピン (3.3 V ドメイン) [3]
5	PWM_DIS	入力	PWM ディセーブル入力ピン (3.3 V ドメイン) [2]
6	GND	グランド	グランド
7	INT	出力	割込み出力ピン (3.3 V ドメイン)
8	PSEL	入力	通信インターフェース選択入力ピン (3.3 V ドメイン) [1]
9	PWM	出力	PWM 出力ピン (3.3 V ドメイン)
10	VDD12/VDD5*	電源 (12 V / 5 V*)	IR エミッタ用 12 V / 5 V*電源

[1]High レベルは UART を選択し、Low レベルは I²C を選択します。必要なインターフェースに応じて、ピンを VDD または GND に配線することを推奨します。

[2] PWM 出力を有効にするために PWM_DIS が GND に配線されている場合、デバイスは連続モードで起動し、測定周期を変更するときに考慮する必要があるアイドルモードではありません。

[3] プルアップ抵抗の値は、アプリケーション全体と使用するクロック周波数に応じて調整する必要があります。

*: XENSIV™ PAS CO2 と比較してエミッタに 12V ではなく 5V を使用した、XENSIV™ PAS CO2 5V を指します。

3 初期化シーケンス

センサーや他の部品を損傷させないために、特定な初期化シーケンスに従う必要があります。

- XENSIV™ PAS CO2 Sensor2Go キットの Figure 5 に示すように、必要なすべてのピンをマイクロコントローラに接続します (電源オフ)。
- マイクロコントローラに電力供給します (通信用の 3.3 V 電源)。
- ヒーターに 12 V / 5 V*を外部から供給します。
- スクリプト/コードの実行 (設定、測定の開始など)。

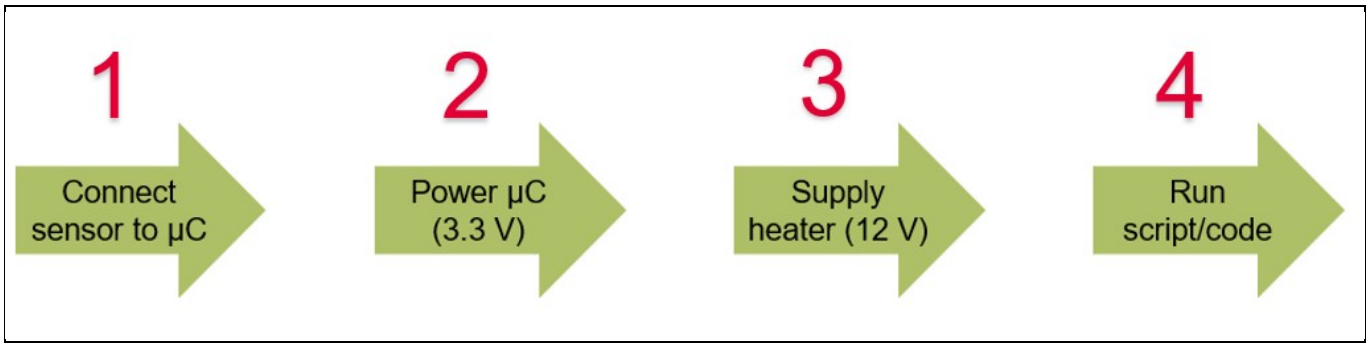


Figure 6 初期化シーケンス

センサーの電源を切る際は、順序を逆にする必要があります。

*Note: 競合状態を防ぐために、3.3 V のロジック電源を最初に完全に起動し、次に 12 V / 5 V*のヒーター電源を起動することを推奨します。*

4 PSOC™ 6 WiFi-BT Pioneer キットのクイックスタート

PSOC™ 6 は、高価で電力を大量に消費するアプリケーションプロセッサと低パフォーマンスのマイクロコントローラの間のギャップを埋めます。超低消費電力 PSOC™ 6 マイクロコントローラアーキテクチャは、IoT デバイスに必要な処理パフォーマンスを提供し、電力とパフォーマンスの間のトレードオフを排除します。 PSOC™ 6 マイクロコントローラには、両方の CPU が単一チップ上にあるデュアル CPU アーキテクチャが含まれます。高性能タスク用の Arm® Cortex®-M4 と、低消費電力タスク用の Arm® Cortex®-M0+ を備えています。セキュリティ機能が組み込まれているため、IoT システムは保護されます。

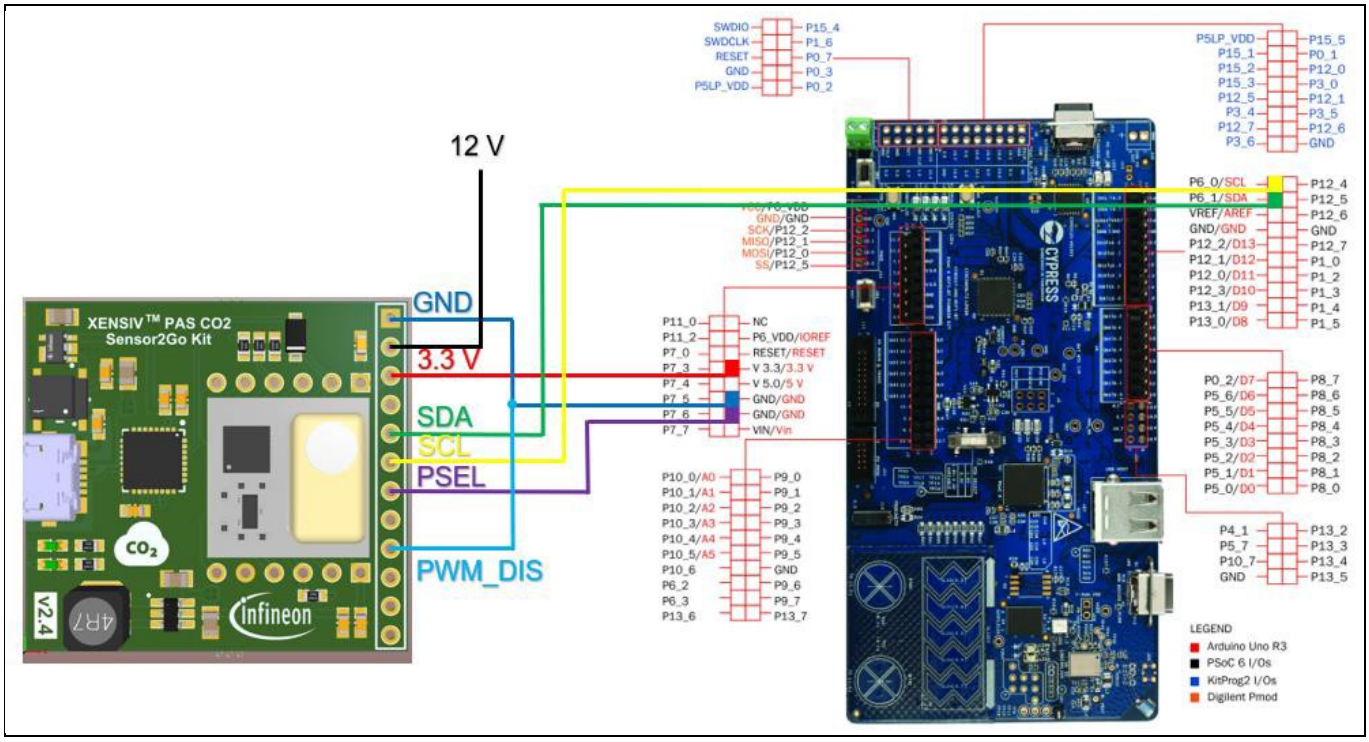


Figure 7 XENSIV™ PAS CO2 Sensor2Go キットを PSOC™ 6 WiFi-BT Pioneer キットに接続するための I²C インターフェース

Table 2 ピン接続

位置	記号	PSOC™ 6 WiFi-BT Pioneer キットへの接続
1	GND	グランド
2	VDD12 / VDD5*	12 V / 5 V*電源 (外部) [3]
3	VDD3.3	3.3 V デジタル電源 [3]
4	RX	未接続
5	TX/SDA	I²C データ ピン (3.3 V ドメイン)
6	SCL	I²C クロック ピン (3.3 V ドメイン)
7	PSEL	グランド
8	INT	未接続 (この場合)
9	PWM_DIS	グランド

位置	記号	PSOC™ 6 WiFi-BT Pioneer キットへの接続
10	PWM	未接続 (この場合)
11	SWD	未接続 (この場合)
12	SWCLK	未接続 (この場合)

[3] 電源許容差 ±10%

4.1 ブリッジコントロールパネル

ブリッジコントロールパネルは、 PSOC™ プログラマに付属するシンプルなデバッグ ツールです。 I2C/UART/SPI シリアル通信インターフェースを介してターゲットデバイスと通信するために使用されます。

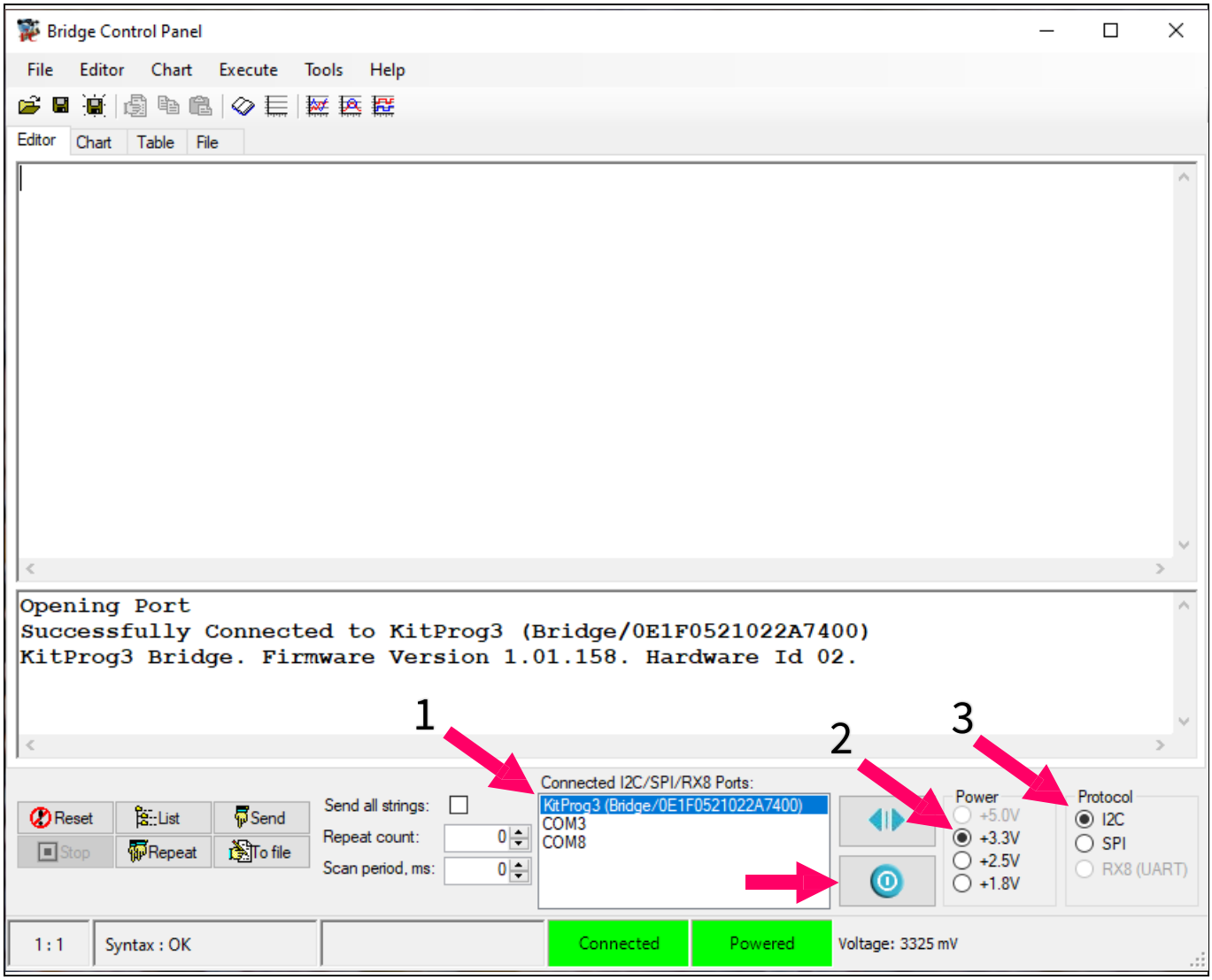


Figure 8 デバイスの初期化

Figure 7 に示すように、XENSIV™ PAS CO2 Sensor2Go キットと PSOC™ 6 WiFi-BT Pioneer キットを配線した後、次の手順で初期化とセンサーとの通信を行います。

4 PSOC™ 6 WiFi-BT Pioneer キットのクイックスタート

1. それに応じて COM ポートを選択します (KitProg3)。
2. Power メニューで 3.3 V を選択します。
3. I²C プロトコルを選択します。

「Toggle Power」ボタンで、3.3 V 電源のオンとオフを切り替えられます。3.3 V が供給されて初めて、12 V / 5 V*を安全に外部から供給できます。デバイスの損傷を防ぐために、「Toggle Power」ボタンで 3.3 V 電源をオフにする場合は、最初に外部 12 V / 5 V*電源をオフにしてから 3.3 V 電源をオフにしてください。

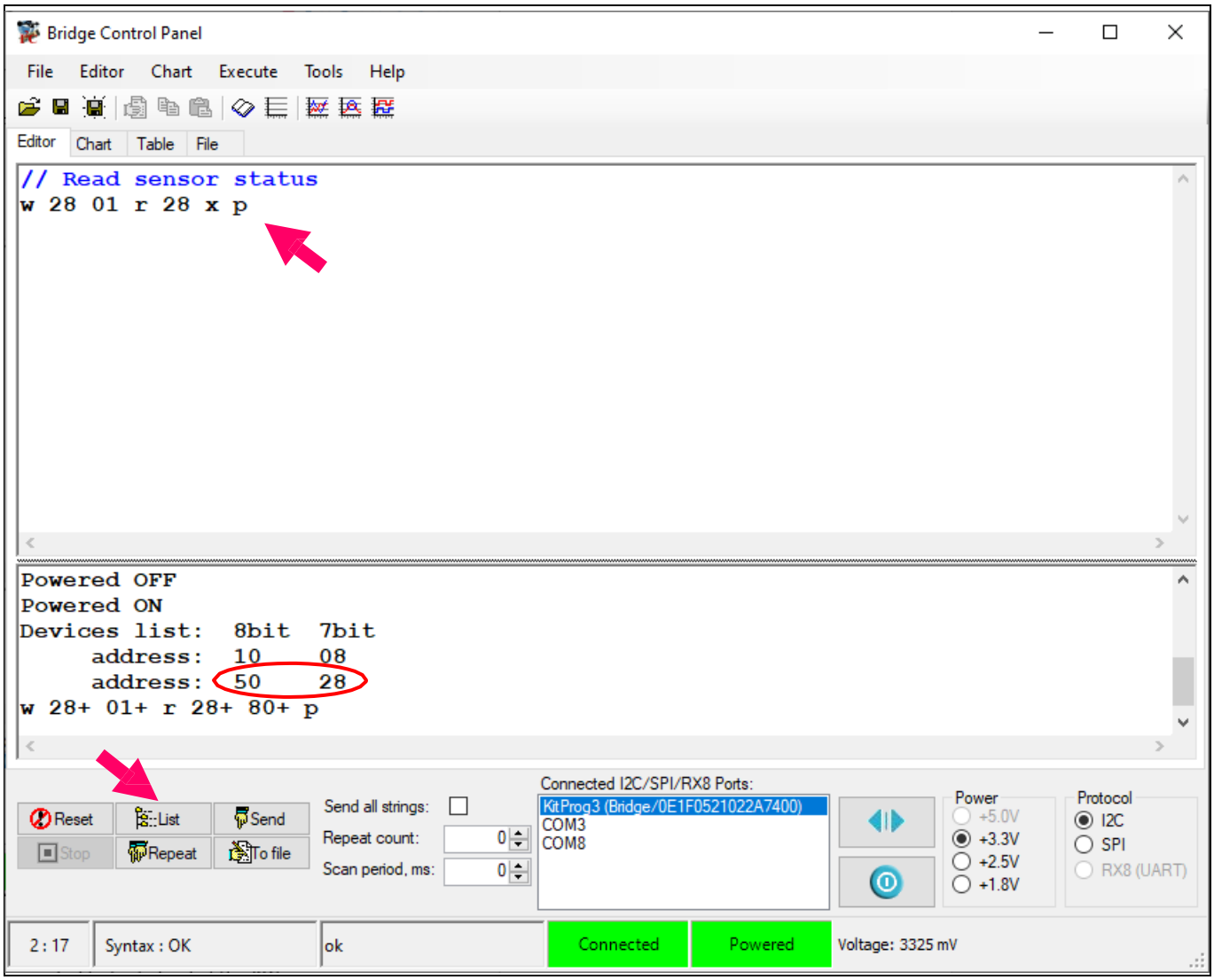


Figure 9 I²C の通信テスト

両方の電源を順番に入れた後、最初に「List」ボタンを押して基本的な I²C 通信をチェックすることを推奨します。ブリッジコントロールパネルには、バス上で利用可能なすべての I²C スレーブデバイスが一覧表示されます。XENSIV™ PAS CO2 のアドレスであるスレーブアドレス 0x28 (0x50 8 ビット) の端末を確認してください。

既知の問題や間違いは次のとおりです。

- 「List」ボタンでスレーブ応答を確認すると、スレーブデバイスがリストされない (端末のエラーメッセージ:「No device found」)

  - すべてのワイヤが正しく接続され、緩んでいないことを確認してください。
- ブリッジコントロールパネルがクラッシュしている
  - 電源ピンの誤接続による電源不足がないことを確認してください。
- デバイスがコマンドを受信しない (例: w 28- 01- r 28- FF- p)
  - I²C ワイヤ (SDA/SCL) が正しく接続され、緩んでいないことを確認してください。

コマンドは「Editer」ウィンドウに記述され、「Enter」を押すことで送信および実行されます。

4.2 測定を開始するための基本コード

ここでは、XENSIV™ PAS CO2 で測定を開始するための基本的な操作について説明します。

第 3 章およびセクション 4.1 のシーケンスに従ってデバイスを初期化した後、最初に初期化がエラーなしで正しく行われたかどうかを確認することを推奨します。これは、**センサー ステータス レジスタ**をチェックすることで行うことができます。その後、測定を開始する前に、圧力補正を確認して設定することを推奨します。これは、2 つの**圧力補償レジスタ**で行うことができます。これらのレジスタで設定されているデフォルトの圧力は 1015 hPa です。これで、測定を開始するためのすべての設定が整いました。測定モードには、シングル ショット測定と連続測定の 2 種類があります。このモードは、**測定モード設定レジスタ**で設定できます。連続測定モードを使用する場合、測定周期は 2 つの**測定周期設定レジスタ**であらかじめ定義しておくことができます。いずれにせよ、測定モードを設定した後、測定シーケンスがトリガされます。**測定ステータスレジスタ**を読み出すことで、測定シーケンスが完了したかどうか、および **CO_2濃度結果レジスタ**に新しい CO_2濃度値が利用可能かどうかを確認できます。

Code listing 1 センサー ステータス レジスタ (SENS_STS、アドレス: 0x01)

```
001        W 28 01 R 28 X p
```

センサーが正しく初期化されると、戻り値は 0xC0 になります。

Code listing 2 圧力補償レジスタ (PRES_REF_H および PRES_REF_L、アドレス: 0x0B および 0x0C)

```
001        w 28 0B r 28 x p
002        w 28 0C r 28 x p
003        w 28 0B 03 p
004        w 28 0C F5 p
```

レジスタ PRES_REF_L と PRES_REF_H は、周囲大気圧を保存するために使用されます。PRES_REF_H (MSB) と PRES_REF_L (LSB) を連結することで、デバイスが読み出せる圧力値が定義されます。連結された圧力値は、符号なし短整数型 (1 ビット = 1 hPa) としてコード化されます。この例では、圧力は 1013 hPa に設定されています。正しく動作するために、ユーザーは、プログラムされた圧力値がデバイスの指定された圧力動作範囲内にあることを確認する必要があります。この有効な動作範囲は 750 hPa から 1150 hPa です。

ユーザーはまた、位置する高度に基づいて、周囲の大気圧を正しく計算する必要があります。高度と気圧の関係は反比例します。これは、高度が上がると気圧が下がることを意味します。この関係は、地表のある地点の上空の空気の密度によるものです。海面では、大気の密度が最大になるため、大気圧が高くなります。高度が高くなると、上空の空気の量が減り、空気の密度が減少し、気圧が下がります。この関係は、温度や湿度などの大気条件の変化によっても影響を受けます。特定の高度での大気圧を計算するためには、次の式を使用できます。

$$P = P_0 * \left[1 - \left(\frac{L * h}{T_0}\right)\right]^{\frac{g * M}{R * L}}$$

ここで、

- P は指定された高度での圧力です (ヘクトパスカル単位)
- P0 は海面での圧力 (ヘクトパスカル単位) で、1013.207 hPa です
- L は温度減率で、高度 11km までは-0.0065·K/m の定数値です
- h は海抜高度 (メートル単位)です
- T0 は海面の標準温度 (ケルビン単位) で、288.15 K の定数値です
- g は重力による加速度で、9.80665·m/s^2 の定数値です
- M は空気のモル質量で、0.0289644·kg/mol の定数値です
- R は普遍気体定数で、8.31447·J/(K*mol)の定数値です

簡易的に下記式に高さ「h」を挿入して算出することも可能です。

P = 1013.207*(1-((6.5*h)/(288150)))^5.255; // h ..メートル単位の高さ

例: デバイスには、入力として hPa 単位の値が必要です。その結果、高度 1000 m では 898 hPa に、高度 2000 m では 794 hPa になります。

Code listing 3 測定周期設定レジスタ (MEAS_RATE_H および MEAS_RATE_L、アドレス: 0x02 および 0x03)

```
001        w 28 02 00 p
002        w 28 03 0A p
```

レジスタ MEAS_RATE_H および MEAS_RATE_L は、連続モードで使用される測定期間を定義します。MEAS_RATE_H (MSB) と MEAS_RATE_L (LSB) を連結して測定期間を定義します。連結された値は、2 の補数の符号付き短整数型 (1 ビット = 1 秒) としてコード化されます。この例では、測定レートは 10 秒に設定されています。設定可能な範囲は、0005_H (5 秒) から $0FFF_H$ (4095 秒) です。MEAS_RATE_H と MEAS_RATE_L に書き込む場合、新しい値はデバイスによってすぐには反映されません。アイドル モードから連続モードへの次の遷移時に内部でラッチされます。

Code listing 4 測定モード設定レジスタ (MEAS_CFG、アドレス: 0x04)

```
001        w 28 04 02 p
002        W 28 04 01 P
```

このレジスタは、デバイスの動作設定を定義します。1 行目のコードではシングルショット測定がトリガされ、2 行目のコードでは連続測定モードが設定されます。1 回のシーケンス後、エミッタが冷却するまでに少なくとも 10 秒かかることに注意してください。5 秒未満の連続モードでの測定速度値は、5 秒に等しいものとして扱われます。シングル ショット測定の場合、正確な読み出しのために、次の測定シーケンスを少なくとも 60 秒遅らせてください。

Code listing 5 測定ステータスレジスタ (MEAS_STS、アドレス: 0x07)

```
001        w 28 07 r 28 x p
```

このレジスタは、センサーのステータス情報を表示します。測定シーケンスが完了し、新しい CO_2濃度値が利用可能になると、このレジスタの戻り値が 0x10 になります。

Code listing 6 CO2 濃度結果レジスタ (CO2PPM_H および CO2PPM_L、アドレス: 0x05 および 0x06)

```
001        w 28 05 r 28 x p
002        w 28 06 r 28 x p
```

レジスタ CO2PPM_H と CO2PPM_L は、最後の CO_2濃度測定の結果を保存するために使用されます。CO2PPM_H (MSB) と CO2PPM_L (LSB) を連結して CO_2濃度値を定義します。連結された CO_2濃度値は、2 の補数の符号付き短整数 (1 ビット = 1 ppm) としてコード化されます。このフィールドは、各測定シーケンスの終了時に更新されます。CO_2濃度値を読み出す場合、ユーザーは最初にレジスタ CO2PPM_H を読み出してから CO2PPM_L を読み出す必要があります。

Code listing 7 まとめ

```
001       w 28 01 r 28 x p // Read sensor status
002       w 28 0B r 28 x p // Read pressure (MSB)
003       w 28 0C r 28 x p // Read pressure (LSB)
004       w 28 0B 03 p // Set pressure (MSB)
005       w 28 0C F5 p // Set pressure (LSB)
006       w 28 02 00 p // Set measurement period (MSB)
007       w 28 03 0A p // Set measurement period (LSB)
008       w 28 04 02 p // Trigger continuous measurement
009       (w 28 04 01 p) // Trigger single shot measurement
010       w 28 07 r 28 x p // Read measurement status
011       w 28 05 r 28 x p // Read CO2 concentration (MSB)
012       w 28 06 r 28 x p // Read CO2 concentration (LSB)
```

Attention: 詳細なレジスタマップについては、別のアプリケーションノートで説明しています (製品ページを参照)

Note: 圧力補正レジスタは、気圧の変動を補正するために定期的に更新する必要があります。

4.3 追加機能

ここでは、XENSIV™ PAS CO2 のその他の機能を紹介します。利用可能な機能の詳細については、別のアプリケーションノートで説明します。

Code listing 8 割込み端子設定レジスタ (INT_CFG、アドレス: 0x08)

```
001       w 28 08 18 p // INT configuration as early measurement start
   notification
```

このレジスタは、ピン INT の設定を定義します。ピン INT は、複数の機能を実行するように設定できる多目的出力ピンです。電気的設定は、プッシュプルおよびローアクティブまたはプッシュプルおよびハイアクティブとして設定できます。

以下の機能を設定できます。アラーム閾値違反通知ピン, Data Ready 通知ピン, センサー ビジー通知ピン, および早期測定開始通知ピン。この例では、割込みはプッシュプルとハイアクティブの早期測定開始通知ピンとして設定されています。割込みが発生した場合の表示は、スティッキービットのクリアと同様に、測定ステータスレジスタで読み出すことができます。

Code listing 9 アラーム閾値レジスタ (ALARM_TH_H および ALARM_TH_L、アドレス: 0x09 および 0x0A)

```
001       w 28 09 03 p // Set alarm threshold (MSB)
002       w 28 0A E8 p // Set alarm threshold (LSB)
```

レジスタ ALARM_TH_H および ALARM_TH_L は、アラームの閾値として使用される値を定義します。ALARM_TH_H (MSB) と ALARM_TH_L (LSB) を連結することで、デバイスが認識できる閾値となります。連結された値は、2 の補数の符号付き短整数型 (1 ビット = 1 ppm) としてコード化されます。この例では、

アラーム閾値は 1000 ppm に設定されています。閾値違反が発生した場合の表示は、測定ステータスレジスタで読み出すことができます。

Code listing 10 自動ベースライン オフセット補正リファレンス (CARIB_REF_H および CALIB_REF_L、アドレス: 0x0D および 0x0E)

```
001       w 28 0D 01 p // Set baseline reference (MSB)
002       w 28 0E 90 p // Set baseline reference (LSB)
```

レジスタ CALIB_REF_H および CALIB_REF_L は、自動ベースライン オフセット補正または強制補正に使用される基準値を定義します。CALIB_REF_H (MSB) および CALIB_REF_L (LSB) を連結して基準値を定義します。連結されたオフセット値は、2 の補数符号付き短整数型 (1 ビット = 1 ppm) としてコード化されます。この例では、自動ベースライン オフセット補正基準が 400 ppm に設定されています。正しく動作させるためには、ユーザーは、プログラムされたオフセット補正値がデバイスの指定された動作範囲内にあることを確認する必要があります。この動作の有効範囲は 350 ppm から 1500 ppm です。自動ベースライン オフセット補正または強制補正は、測定モード設定レジスタで有効/無効にし、ソフト リセット レジスタでリセットできます。自動ベースライン オフセット補正と強制補正の詳細については、別のアプリケーションノートで説明します (製品ページを参照)。

Code listing 11 スクラッチ パッド レジスタ (SCRATCH_PAD、アドレス: 0x0F)

```
001       w 28 0F 01 p
002       w 28 0F r 28 x p
```

このレジスタは、実行時のデータ整合性テストのための読み出し、および書き込み可能なアドレス空間を提供します。このレジスタは、特定のハードウェア機能には関連付けられていません。

Code listing 12 ソフト リセット レジスタ (SENS_RST、アドレス: 0x10)

```
001       w 28 10 A3 p // Triggers soft reset
002       w 28 10 DF p // Disables filter
003       w 28 10 FE p // Enables filter
004       w 28 10 BC p // Resets ABOC context
005       w 28 10 FC p // Resets forced compensation
006       w 28 10 CF p // Saves forced compensation immediatel
```

このレジスタは、ソフト リセットをトリガするために使用されます。また、フィルタの設定と、自動ベースラインオフセット補正または強制補正のリセットについても説明します。デフォルトでは、フィルタは有効になっており、このレジスタに 0xDF を書き込むことで無効にできます。0xFE を書き込むと、フィルタが再び有効になります。自動ベースライン オフセット補正のコンテキストは、0xBC 書き込みでリセットでき、強制補正のリセットの場合には 0xFC です。

5 Arduino Due のクイックスタート

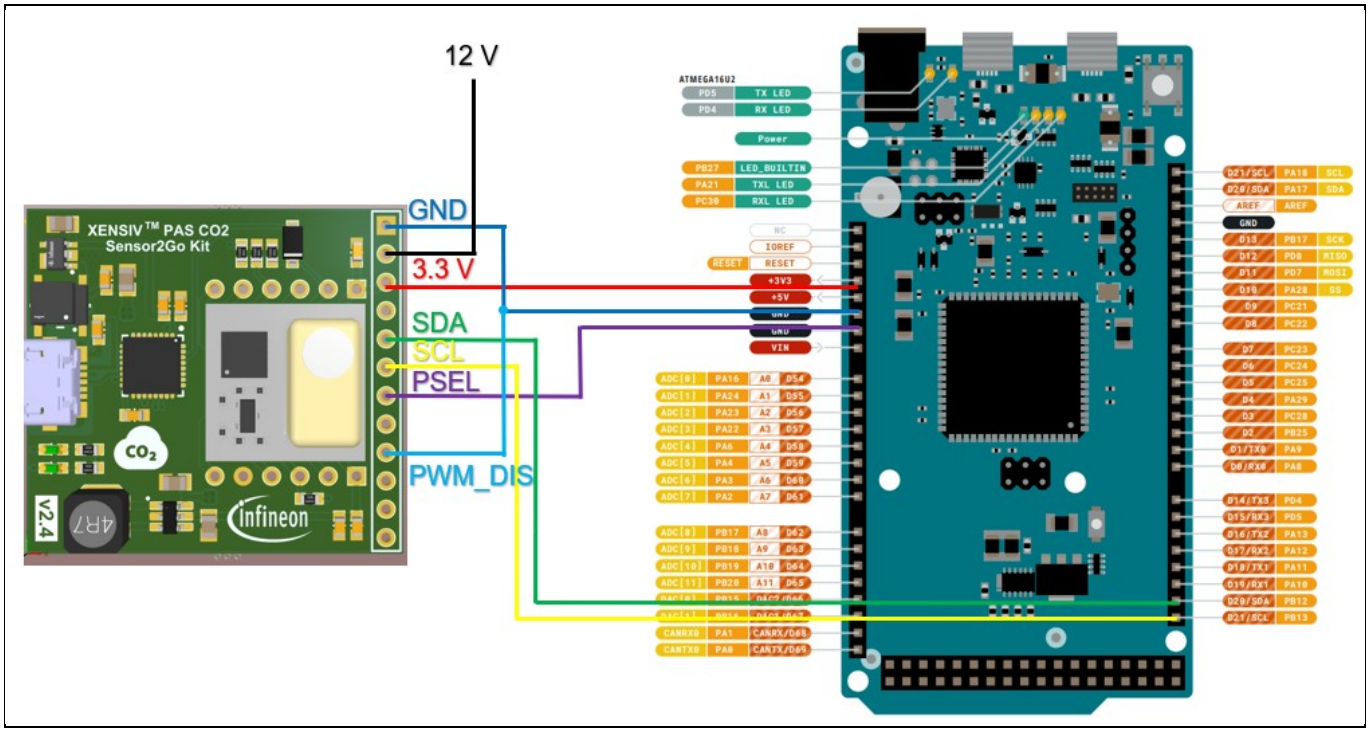


Figure 10 XENSIV™ PAS CO2 Sensor2Go キットの Arudino Due への I²C インターフェース接続 [4)]

デバイスの Arduino Due へのピン接続は、PSOC™ 6 WiFi-BT Pioneer キットへの接続に相当します。別の Arduino または他のデジタルピンを使用する場合は、それぞれのプルアップ抵抗が使用可能であることを確認してください。

[4)] https://content.arduino.cc/assets/Pinout-Due_latest.pdf からの Arduino のピン配置

5.1 Arduino IDE

Arduino IDE をインストールした後、ボードマネージャで Arduino Due を含む適切なパッケージ "Arduino SAM Boards" (32 ビット Arm® Cortex®-M3)をインストールしてください（Figure 11 参照）。Tools ドロップダウンメニューでそれぞれのボードと COM ポートを選択してください（Figure 12 参照）。スケッチのアップロードや Arduino Due との通信にはプログラミング ポートを使用します。まず、Arduino Due との通信が存在し、応答するかどうかを "Blink "の例で確認することを推奨します。Arduino Due との通信の設定と確認ができたら、コードの実装を開始します。

5 Arduino Due のクイックスタート

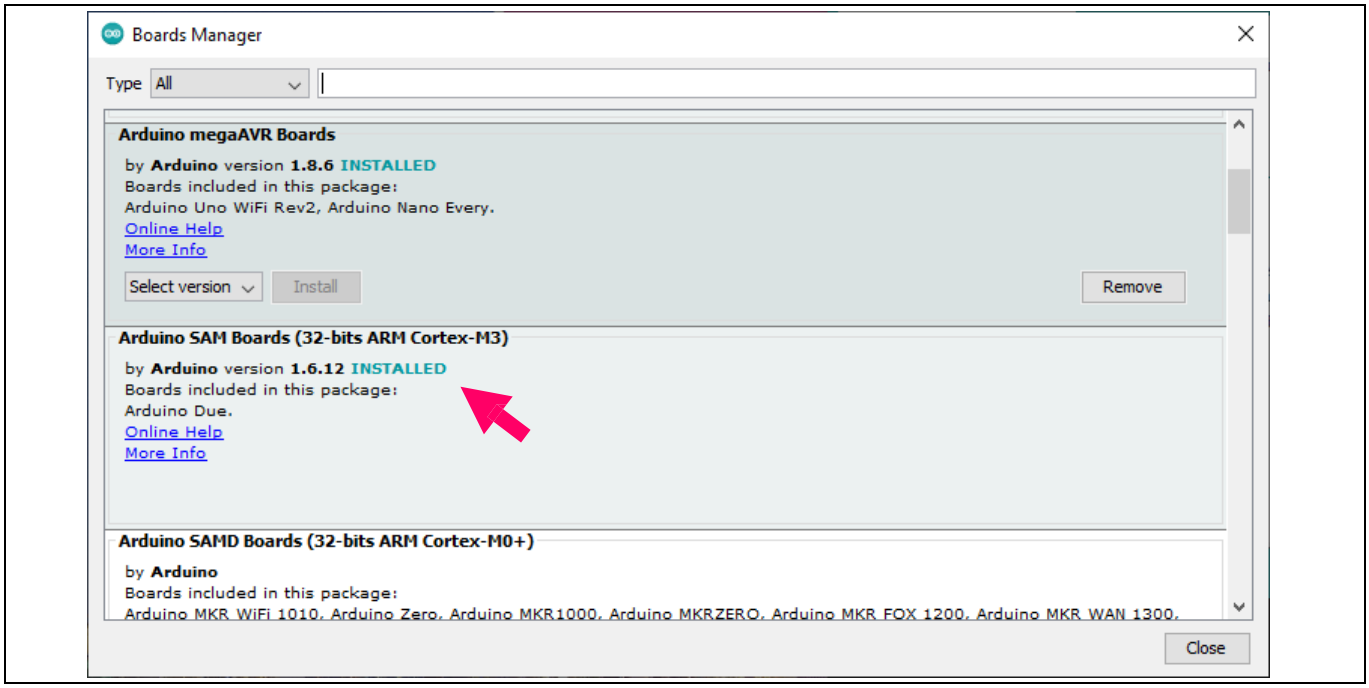


Figure 11　Arduino IDE 設定: ボード マネージャー

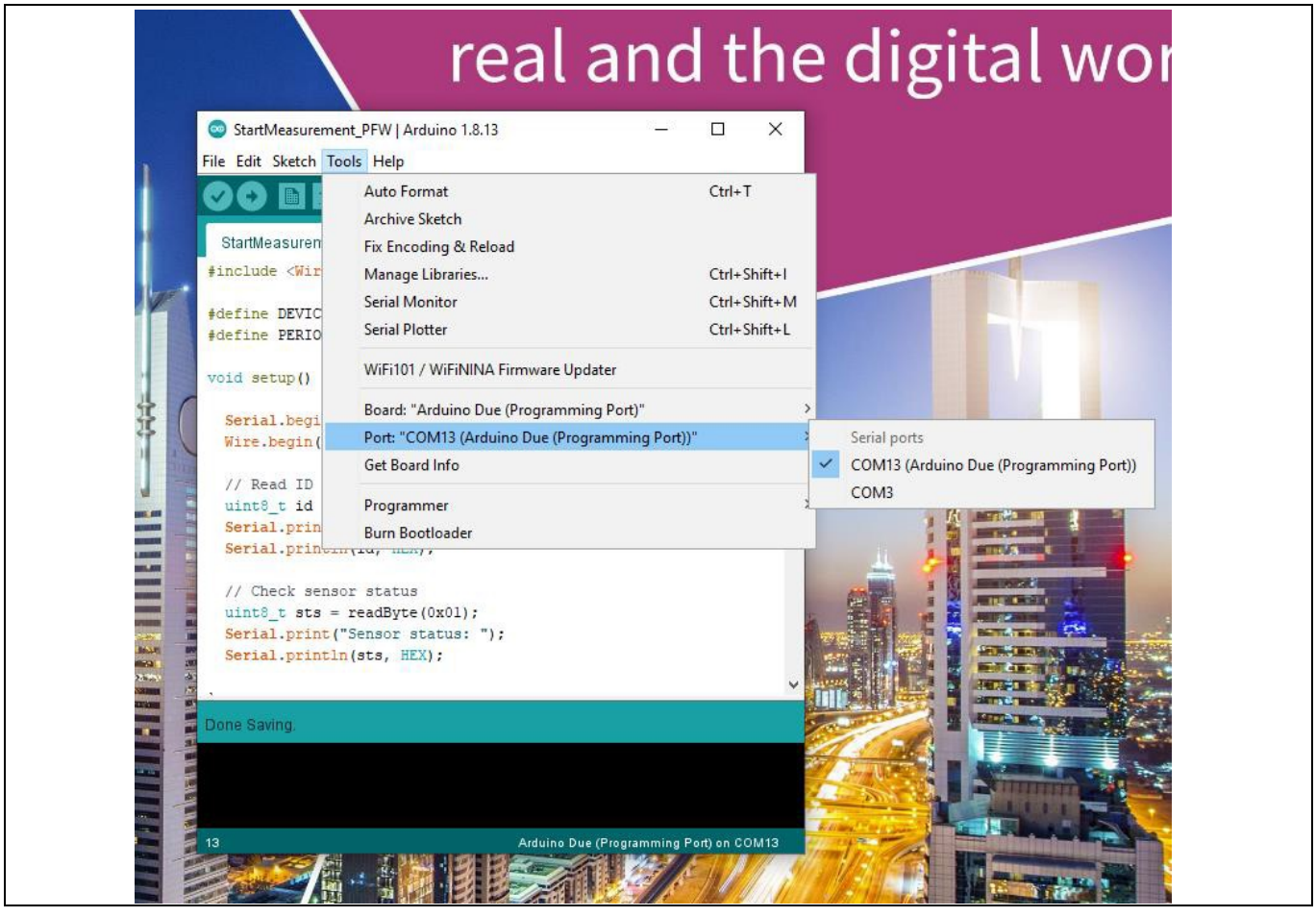


Figure 12　Arduino IDE 設定: ポートの選択

5.2 測定を開始するための基本的な Arduino コード

読み出しコマンドと書き込みコマンドの関数を実装することを推奨します。その後、デバイスの初期化, チェック, および操作を行うことができます。測定は、 PSOC™ 6 WiFi-BT Pioneer キットと同様に、対応するレジスタを書き込んだり読み出したりすることで開始できます。注意すべき点の 1 つは、コマンドパッケージが失われたりスキップされたりしないように、十分な遅延を設定することが重要です。Arduino のサンプル コードは、レジスタの読み出し用, レジスタへの書き込み用, およびこれらの機能を利用したシングルショット測定または連続モード測定を開始するためのスクリプトの 4 つです。

Code listing 13 readByte

```
uint8_t readByte(uint8_t regAddress)
{
  Wire.beginTransmission(deviceAddress);
  Wire.write(regAddress);
  Wire.endTransmission(false);
  //request 1 byte from slave
  if (Wire.requestFrom(deviceAddress, 1U, 1U) > 0)
  {
    return Wire.read();
  }
  else
  {
    return 0x0;
  }
}
```

Code listing 14 writeByte

```
bool writeByte(uint8_t regAddress, uint8_t data)
{

  Wire.beginTransmission(deviceAddress);
  Wire.write(regAddress);
  Wire.write(data);
  if (Wire.endTransmission() != 0)
  {
    return false;
  }
  else
  {
    return true;
  }
}
```

Code listing 15 シングルショット測定を開始するためのコード

```
#include <Wire.h>

deviceAddress 0x28 #define
#define PERIOD 10000

```

```
006       void setup() {
007
008         Serial.begin(115200);
009         Wire.begin();
010
011         // Check sensor status
012         uint8_t sts = readByte(0x01);
013         Serial.print("Sensor status: ");
014         Serial.println(sts、HEX);
015
016         // Idle mode
017         writeByte(0x04, 0x00);
018         delay(400);
019
020         // Set pressure
021         writeByte(0x0B, 0x03);
022         writeByte(0x0C, 0xF5);
023       }
024
025       void loop()
026       {
027         // Trigger single measurement
028         writeByte(0x04, 0x01);
029         delay(400);
030
031         // Get PPM value
032         uint8_t value1 = readByte(0x05);
033         delay(5);
034         uint8_t value2 = readByte(0x06);
035         delay(5);
036
037         // Calculate ppm value
038         int16_t result = value1 << 8 | value2;
039         Serial.print("CO2:");
040         Serial.print(result);
041         Serial.println( "ppm");
042
043         delay(PERIOD);
044
045       }
```

Code listing 16 連続モード測定を開始するためのコード

```
001       #include <Wire.h>
002       deviceAddress 0x28 #define
003
004       void setup() {
005
006         Serial.begin(115200);
007         Wire.begin();
```

```
008
009         // Read ID
010         uint8_t id = readByte(0x00);
011         Serial.print("ID: ");
012         Serial.println(id, HEX);
013
014         // Check sensor status
015         uint8_t sts = readByte(0x01);
016         Serial.print("Sensor status: ");
017         Serial.println(sts、HEX);
018
019         // Idle mode
020         writeByte(0x04, 0x00);
021         delay(400);
022
023         // Set measurement rate to 10 s
024         writeByte(0x02, 0x00);
025         writeByte(0x03, 0x0A);
026
027         // Configure continuous mode
028         writeByte(0x04, 0x02);
029       }
030
031       void loop() {
032
033         // Poll measurement status
034         uint8_t meas_sts = readByte(0x07);
035         delay(100);
036
037         if (meas_sts == 0x10) {
038
039           // Get PPM value
040           uint8_t value1 = readByte(0x05);
041           delay(5);
042           uint8_t value2 = readByte(0x06);
043           delay(5);
044
045           // Calculate ppm value
046           int16_t result = value1 << 8 | value2;
047           Serial.print("CO2:");
048           Serial.print(result);
049           Serial.println( "ppm");
050         }
051         delay(1000);
052       }
```

連続モードでは、アプリケーションマイクロコントローラとデバイス間の同期を考慮する必要があります。測定シーケンスが開始されると、デバイスは測定中 (~1 秒) はどの着信フレームにも応答せず、代わりに NACK で応答することに注意してください。この例に基づいて、これを処理するための 2 つのオプションがあります。

- 割込みをデータ準備完了として設定し、測定が完了したかどうかを監視します
- NACK 応答を受信した後、測定ステータスを再度ポーリングします (Arduino ライブラリの例が利用可能)

正確で信頼性の高い測定を確保するには、センサーを推奨される測定速度である 60 秒以内に動作させる必要があります。この測定速度は、消費電力を削減し、デバイスの寿命を延ばすのにも役立ちます。ただし、特定の応用例やデモの目的では、より高速な測定間隔が必要になる場合があります。したがって、CO2 濃度の変化に基づいて測定レートを調整するために使用できる適応ポーリングアルゴリズムが実装されています。このアルゴリズムには、精度を維持し、消費電力を削減しながら、必要に応じて高速測定を確保できるという利点があります。

Code listing 17　適応型ポーリングのコード

```
001       PAS_delta_change = 1.0 * (100.0 / (co2ppm_old + 1) * co2ppm) -
   100;
002       If (abs(PAS_delta_change) >= 7.0 || abs(co2ppm_old - co2ppm)
   >= 75)
   {
003         if (PERIODIC_MEAS_INTERVAL_IN_SECONDS != 5) {
004         PERIODIC_MEAS_INTERVAL_IN_SECONDS = 5;
005         PAS_ModeSwitch = 1;
006         } else {
007         PAS_ModeSwitch = 0;
008         Serial.println("PAS: already in FAST mode");
009         }
010       } else {
011         if (PERIODIC_MEAS_INTERVAL_IN_SECONDS != 60) {
012           if (PAS_min_FAST_records == 0) {
013             Serial.println("PAS: switch to SLOW mode");
014             PERIODIC_MEAS_INTERVAL_IN_SECONDS = 60;
015             PAS_ModeSwitch = 1;
016             PAS_min_FAST_records = 10;
017           } else {
018             Serial.print("PAS: IGNORE switch to SLOW mode: ");
019             Serial.print(PAS_min_FAST_records);
020             Serial.print(" Polling speed: ");
021             Serial.println(PERIODIC_MEAS_INTERVAL_IN_SECONDS);
022             PAS_min_FAST_records--;
023           }
024         } else {
025           PAS_ModeSwitch = 0;
026           Serial.println("PAS: already in SLOW mode");
027         }
028       }
029       co2ppm_old = co2ppm;
030       if (PAS_ModeSwitch = 1) {
031         err = cotwo.startMeasure(PERIODIC_MEAS_INTERVAL_IN_SECONDS);
032       }
```

上記のアルゴリズムは、CO2 濃度の変化に基づいてセンサーの測定速度を調整するように設計されています。このアルゴリズムは、PAS_delta_change 変数を使用して、前回の測定値からの CO2 濃度の変化率

を計算することから始まります。変化の絶対値が設定された閾値 (この場合は 7.0) 以上の場合、または現在と以前の CO2 濃度の差が 75 ppm 以上の場合、測定レートを 5 秒に変更して頻度を上げます。変更が指定された閾値よりも小さい場合、測定レートは 60 秒に変更され、頻度が減少します。このアルゴリズムには、低速測定レートに戻る前に、最小限の高速測定が含まれます。このようにして、アルゴリズムは測定が最適な間隔で行われることを保証し、不要な測定の数を減らし、精度と安定性を維持しながら電力を節約します。プログラムコードは、ユーザーのルーチンに統合することも、付属の Arduino ライブラリに統合することもできます。

5.3 Arduino ライブラリ

コアライブラリは C 言語ベースで、XENSIV™ PAS CO2 センサーにプラットフォームに依存しないドライバを提供します。センサーのすべての機能へのフルアクセスを提供します。ドライバは 4 つのファイルで構成されています。

Table 3 コア C ドライバ

ソースコード	説明
xensiv_pasco2_ver.h	XENSIV™ PAS CO2 センサーの正確なバージョンが含まれています
xensiv_pasco2_regs.h	XENSIV™ PAS CO2 センサーとのやりとりに必要なレジスタ定義が含まれています
xensiv_pasco2.h	XENSIV™ PAS CO2 センサーとのやりとりに必要な全機能を備えています。
xensiv_pasco2.c	XENSIV™ PAS CO2 センサーとのやりとりに必要な全機能を備えています。

ドライバの完全なドキュメントと概要は、GitHub にあります。

https://github.com/Infineon/sensor-xensiv-pasco2

また、すべてのマクロ, 列挙型, および関数の詳細な説明を含むドキュメントは、次の場所にあります。
https://infineon.github.io/sensor-xensiv-pasco2/html/group__group__board__libs.html

Arduino ライブラリは、このコア C ドライバと C ++ラッパーを使用してエコシステム設計パターンに従っているため、Arduino ユーザーはこのライブラリで使い慣れたものを見つけることができます。

Table 4 Arduino ライブラリ

ソースコード	説明
pas-co2-pal-ino.cpp	ターゲット プラットフォーム固有の実装
pas-co2-platf-ino.hpp	条件付きコンパイルによる既定のボードの定義と選択
pas-CO2-ino.hpp	コアドライバから採用された XENSIV™ PAS CO2 センサーとのやりとりに必要な機能が含まれています
pas-co2-ino.cpp	コアドライバから採用された XENSIV™ PAS CO2 センサーとのやりとりに必要な機能が含まれています

Arduino ライブラリには、ユーザーが変更して参照として使用できる 6 つの例が含まれています。

5 Arduino Due のクイックスタート

Table 5 Arduino の例

例	説明
alarm-notification	閾値超過の時にハードウェア割込みを介して同期されたセンサーの CO2 濃度の読み出し
device-id	センサー デバイスの製品およびリビジョン識別子の読み出し
single-shot-mode	シングルショット測定モードを使用したセンサーの CO2 濃度値の読み出し
continuous-mode	連続測定モードを使用したセンサーの CO2 濃度値の読み出し
forced-compensation	強制補正を使用した CO2 基準オフセットの設定
early-notification	ハードウェア割込みを介して同期された早期通知に基づいたセンサーの CO2 濃度の読み出し

Arduino ライブラリの完全なドキュメントは、GitHub にあります。

https://github.com/Infineon/arduino-pas-co2-sensor

6 UART インターフェース

ピン PSEL を High に設定して UART をシリアル通信インターフェースとして選択すると、デバイスは UART スレーブとして機能します。その結果、マスターがタイムアウトメカニズムを使用することが推奨されています。このデバイスは、ポイント ツー ポイント通信のために UART を介して動作するため、バス動作はサポートされていません。

有効な UART フレームの基本フォーマットは、1 つのスタートビット, 8 つのデータビット, パリティビットなし, および 1 つのストップビットです。ボーレートは 9.6 kbps です。マスターは、複数の UART フレームをメッセージ (読み出しまたは書き込み) に結合します。マスター要求とスレーブ応答の組み合わせによって、トランザクションが定義されます。

デバイスは、有効な着信メッセージを検出すると、ACK フレームで応答します。それ以外の場合は、NAK 通知が発行されます。

測定シーケンスが開始されると、デバイスは、測定シーケンス (ACK または NAK のいずれか) の間、いかなる着信フレームにも応答しないことに留意してください。したがって、測定シーケンス中に送信されたメッセージは、測定シーケンスが完了すると、マスターによって再送信されます。

デバイスは、一括読み出しおよび一括書き込み操作をサポートしていません。1 つのトランザクション内で読み出しまたは書き込みできるのは、単一のデータ バイトのみです。

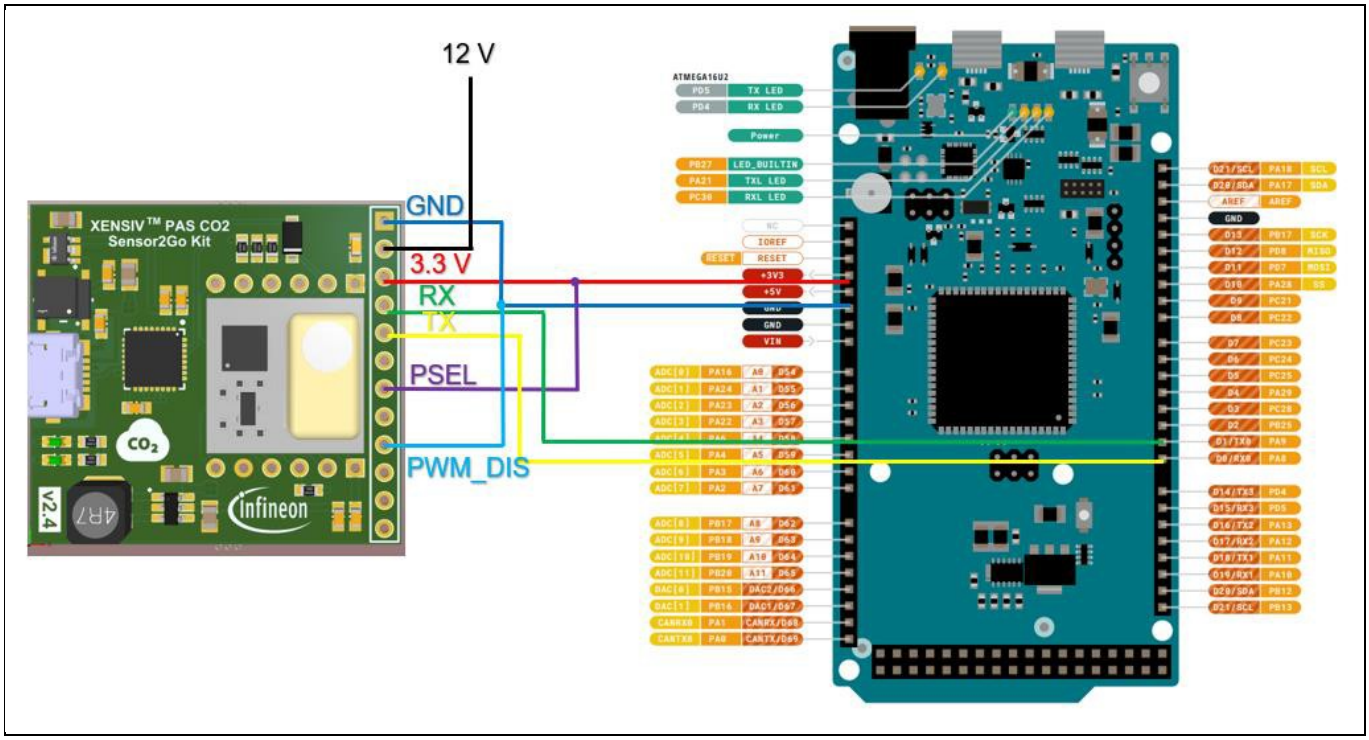


Figure 13 XENSIV™ PAS CO2 Sensor2Go キットと Arduino Due への UART インターフェース接続 [4)]

6.1 書き込みトランザクション

書き込みトランザクションは、以下のフレーム シーケンスによって作成されたメッセージによって開始されます。

フレーム	説明	フレーム ペイロード	備考
1	要求の開始	0x57 (「W」 の ASCII コード) または 0x77 (「w」 の ASCII コード)	
2	デリミタ	0x2C ("," の ASCII コード)	
3	アドレス 1 (最上位 4 ビット)	レジスタアドレスの最上位 4 ビットの 16 進値の ASCII コード	例: アドレス 0FH にレジスタを書き込むためには、ペイロードを 0x30 (「0」 の ASCII コード) にする必要があります。
4	アドレス 2 (最下位 4 ビット)	レジスタアドレスの最下位 4 ビットの 16 進値の ASCII コード	例: アドレス 0FH にレジスタを書き込むためには、ペイロードを 0x46 (「F」 の ASCII コード) にする必要があります。
5	デリミタ	0x2C ("," の ASCII コード)	
6	データ 1 (最上位 4 ビット)	書き込まれたデータの最上位 4 ビットの 16 進値の ASCII コード	例: データ 42H を書き込むためには、ペイロードを 0x34 (「4」 の ASCII コード) にする必要があります。
7	データ 2 (最下位 4 ビット)	書き込まれたデータの最下位 4 ビットの 16 進値の ASCII コード	例: データ 42H を書き込むためには、ペイロードを 0x32 (「2」 の ASCII コード) にする必要があります。
8	メッセージの終わり	0x0A (改行"\n"の ASCII コード)	
1-8	フルメッセージ	0x57 0x2C 0x30 0x46 0x2C 0x34 0x32 0x0A	結合され、変換された形式: W,0F,42\n

受信メッセージの受信が終了すると、デバイスは応答メッセージで応答します。

書き込み操作が有効な場合、デバイス (スレーブ) は次のメッセージで応答します。

フレーム	説明	値	備考
1	ACK	0x06 (「ACK」 の ASCII コード)	
2	メッセージの終わり	0x0A (改行 "\n" の ASCII コード)	

書き込み操作が有効でない場合、デバイスは次のメッセージで応答します。

フレーム	説明	値	備考
1	NAK	0x15 (「NAK」 の ASCII コード)	
2	メッセージの終わり	0x0A (改行 "\n" の ASCII コード)	

6.2 読み出しトランザクション

読み出しトランザクションは、以下のフレームシーケンスによって作成されたメッセージによって開始されます。

フレーム	説明	フレーム ペイロード	備考
1	要求の開始	0x52 (「R」 の ASCII コード) または 0x72 (「r」 の ASCII コード)	
2	デリミタ	0x2C ("," の ASCII コード)	
3	アドレス 1 (最上位 4 ビット)	レジスタアドレスの最上位 4 ビットの 16 進値の ASCII コード。	例: アドレス 0FH にレジスタを書き込むためには、ペイロードを 0x30 (「0」 の ASCII コード) にする必要があります。
4	アドレス 2 (最下位 4 ビット)	レジスタアドレスの最下位 4 ビットの 16 進値の ASCII コード。	例: アドレス 0FH にレジスタを書き込むためには、ペイロードを 0x46 (「F」 の ASCII コード) にする必要があります。
5	メッセージの終わり	0x0A (改行 "\n" の ASCII コード)	
1-5	フルメッセージ	0x52 0x2C 0x30 0x46 0x0A	結合され、変換された形式: R,0F\n

受信メッセージの受信が終了すると、デバイスは応答メッセージで応答します。

読み出し操作が有効な場合、デバイス (スレーブ) は次のメッセージで応答します。

フレーム	説明	値	備考
1	データ 1 (最上位 4 ビット)	書き込まれたデータの最上位 4 ビットの 16 進値の ASCII コード	
2	データ 2 (最下位 4 ビット)	書き込まれたデータの最下位 4 ビットの 16 進値の ASCII コード	
3	メッセージの終わり	0x0A (改行 "\n" の ASCII コード)	

読み出し操作が有効でない場合、デバイスは次のメッセージで応答します。

フレーム	説明	値	備考
1	NAK	0x15 (「NAK」 の ASCII コード)	
2	メッセージの終わり	0x0A (改行 "\n" の ASCII コード)	

6.3 Arduino の例

上記の書き込みトランザクションおよび読み出しトランザクションのコマンド例に続いて、小さい Arduino の例を示します。スクラッチパッド レジスタは、テスト用の読み出しおよび書き込み可能なフィールドを提供するために使用されます。

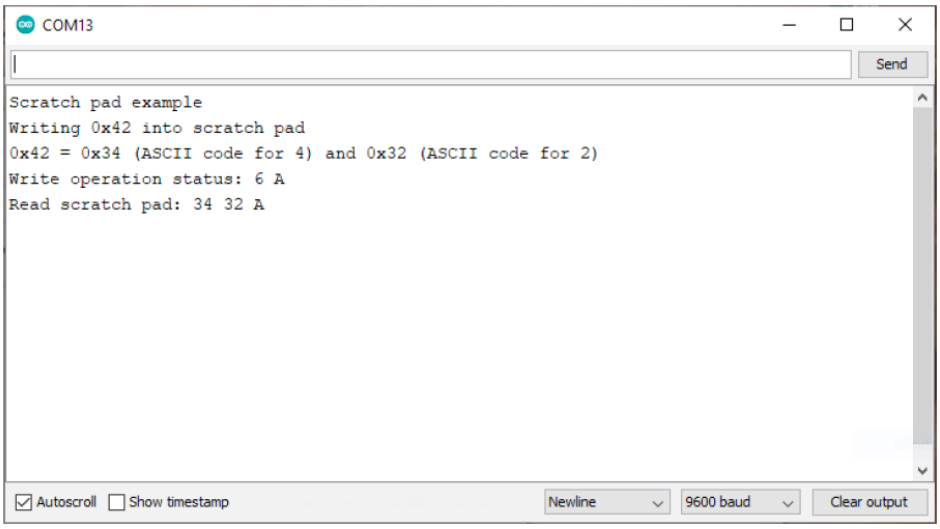


Figure 14　UART を介したスクラッチパッドの例

Code listing 18　スクラッチ パッドの例

```
001        byte i;
002        int incomingByte[4];
003
004        void setup() {
005
006          Serial.begin(9600);
007          Serial1.begin(9600);
008
009          Serial.println("Scratch pad example");
010          Serial.println("Writing 0x42 into scratch pad");
011          Serial.println("0x42 = 0x34 (ASCII code for 4) and 0x32
   (ASCII code for 2)");
012
013          // Write into scratch pad
014          Serial1.write("W,0F,42\n");
015          delay(50);
016
017          // Read scratch pad
018          Serial1.write("R,0F\n");
019          delay(50);
020
021          // Write received data to buffer
022          while (Serial1.available() > 0) {
023            incomingByte[i] = Serial1.read();
024            i++;
025          }
026
027          // Read received data from buffer
028          Serial.print("Write operation status: ");
029
030          for (i = 0; i < 2; i = i + 1) {
```

```
031           Serial.print(incomingByte[i], HEX);
032           Serial.print(" ");
033         }
034
035         Serial.println();
036
037         Serial.print("Read scratch pad: ");
038         for (i = 2; i < 5; i = i + 1) {
039           Serial.print(incomingByte[i], HEX);
040           Serial.print(" ");
041         }
042       }
043
044       void loop() {
045
046       }
```

次の例に、書き込みトランザクションと読み出しトランザクションを使用して、連続モードで測定を開始し、CO2 濃度を読み出す方法を示します。この例には、ASCII 値を ppm 単位で正しく読み出すために ASCII 値を HEX 値に変換する関数が含まれます。

Code listing 19　連続測定モードと ASCII 変換の例

```
001       byte i;
002       char incomingByte[255], CO2_MSB[255], CO2_LSB[255];
003       int CO2MSB, CO2LSB;
004       long res = 0L ;
005
006       void setup() {
007         Serial.begin(9600);
008         Serial1.begin(9600);
009
010         Serial1.write("R,00\n");
011         delay(50);
012
013         Serial.println("Read FW Version");
014
015         // Write received data to buffer
016         while (Serial1.available() > 0) {
017           incomingByte[i] = Serial1.read();
018           i++;
019         }
020
021         // Read received data from buffer
022         Serial.print("Response: ");
023         for (i = 0; i < 3; i = i + 1) {
024           Serial.print(incomingByte[i]);
025           Serial.print(" ");
026         }
027
028         Serial.println("Start measurement with 10s sampling rate");
```

```
029
030         // Idle mode
031         Serial1.write("W,04,00\n");
032         delay(100);
033
034         // Set measurement rate to 10 s
035         Serial1.write("W,02,00\n");
036         delay(100);
037
038         Serial1.write("W,03,0A\n");
039         delay(100);
040
041         // Configure continuous mode
042         Serial1.write("W,04,02\n");
043         delay(12000);
044
045         // Write received data to buffer
046         while (Serial1.available() > 0) {
047           incomingByte[i] = Serial1.read();
048           i++;
049         }
050       }
051
052       void loop() {
053
054         i = 0;
055         Serial1.write("R,05\n");
056         delay(50);
057
058         // Write received data to buffer
059         while (Serial1.available() > 0) {
060           CO2_MSB[i] = Serial1.read();
061           i++;
062         }
063
064         for (i = 0 ; i < 2 ; i++) {
065           res <<= 4 ;
066           res += hex2bin (CO2_MSB[i]) ;
067         }
068         CO2MSB = res;
069
070         i = 0;
071         Serial1.write("R,06\n");
072         delay(50);
073
074         // Write received data to buffer
075         while (Serial1.available() > 0) {
076           CO2_LSB[i] = Serial1.read();
077           i++;
078         }
```

```
079
080         for (i = 0 ; i < 2 ; i++) {
081           res <<= 4 ;
082           res += hex2bin (CO2_LSB[i]) ;
083         }
084         CO2LSB = res;
085
086         int16_t result = CO2MSB << 8 | CO2LSB;
087         Serial.print("CO2:");
088         Serial.print(result);
089         Serial.println( "ppm");
090
091         delay(10000);
092       }
093
094       int hex2bin (char c)
095       {
096         if (c >= '0' && c <= '9')
097           return c - '0' ;
098         if (c >= 'A' && c <= 'F')
099           return c - 'A' + 10 ;
100             if (c >= 'a' && c <= 'f')
101               return c - 'a' + 10 ;
102       }
```

7 PWM インターフェース

通信インターフェースが利用できない場合 PWM_DIS ピンを使用してデバイスを制御できます。PWM_DIS は、ブート シーケンスの電源投入後 (ソフト リセット後ではなく) に最初にアサートされ、ピンのレベルがチェックされます。Low レベルが検出されると、内部割込みルーチンがデバイスを連続モードに設定し、測定シーケンスを開始します。各測定シーケンスの最後に、ピン PWM_DIS のレベルがポーリングされます。この値が High の場合、デバイスはアイドル モードに構成され、出力ピンの PWM は無効になります。

ピン PWM は、タイミング情報に CO2 濃度値を含む PWM 信号を配信することにより、CO2 濃度を読み出すことができます。各測定シーケンスの最後に、デバイスは測定された CO2 濃度で PWM タイミングを更新します。PWM 出力を有効にするには、次の 2 つの条件を満たす必要があります。

- PWM 出力は、ソフトウェアで測定モード設定レジスタを有効にする必要があります
- PWM_DIS ピンは GND に設定する必要があります

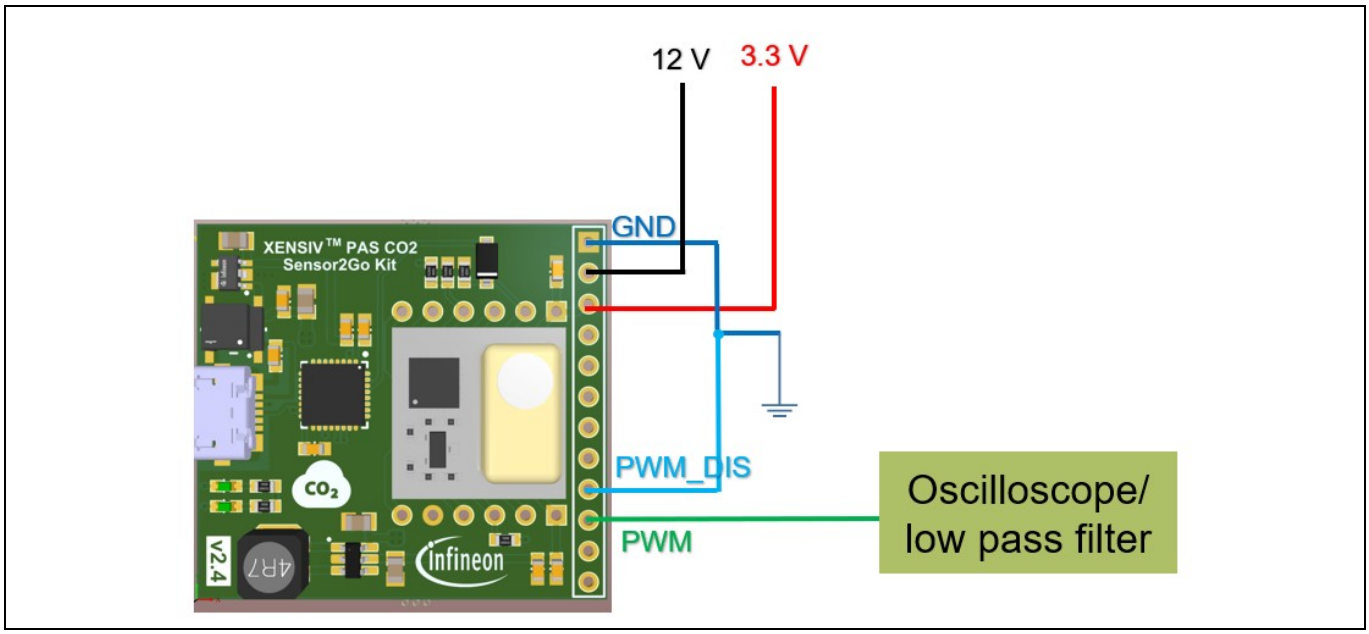


Figure 15 XENSIV™ PAS CO2 Sensor2Go キットの PWM 接続例

PWM 信号の主な仕様は、以下のとおりです。出力信号は、パルス幅を直接測定するか、ローパスフィルタを使用して出力電圧を測定することによって変換できます。

パラメーター	値
ベース周波数	80Hz
デューティ比	0% (0ppm) から 100% (10,000ppm) まで直線的
分解能	1ppm (1.25µs)

通常、PWM 信号はローパスフィルタを介して電圧信号に変換されます。設定時間, リップル, および消費電流の間には固有のトレードオフがあるため、ローパスフィルタの理想的なパラメーター設定はアプリケーションによって異なります。PWM のリップルは、CO2 濃度の読み出し値に誤差が生じる可能性があることを考慮する必要があります。生じた誤差の推定値を以下に示します。

7 PWM インターフェース

濃度	デューティ比	予期されるエラー	ON 時間	ターゲット電圧	リップル	生じた誤差
400 ppm	4%	+/- 42 ppm	0.5 ms	0.132 V	+/- 8 mV	+/- 24 ppm
5000 ppm	50%	+/- 180 ppm	6.25 ms	1.65 V	+/- 50 mV	+/- 150 ppm

PWM ピンで出力される PWM パルスの数は、測定モード設定レジスタでカバーされるデバイスの設定によって異なります。PWM シングル パルス モードでは、デバイスが非アクティブになる前に、1 つの PWM パルスのみが生成されます。PWM パルス トレイン モードでは、デバイスが非アクティブになる前に 160 パルス (約 2 秒) のパルス列が出力されます。シングル パルス モードで計算する場合、1 ppm の分解能が 1.25 µs に等しいことを考慮して、パルスの高デューティ時間を 80 Hz の周波数と比較する必要があります。パルス トレイン モードで計算する場合、デューティ比は 0% (0 ppm) から 100% (10,000 ppm) の値を考慮して計算する必要があります。

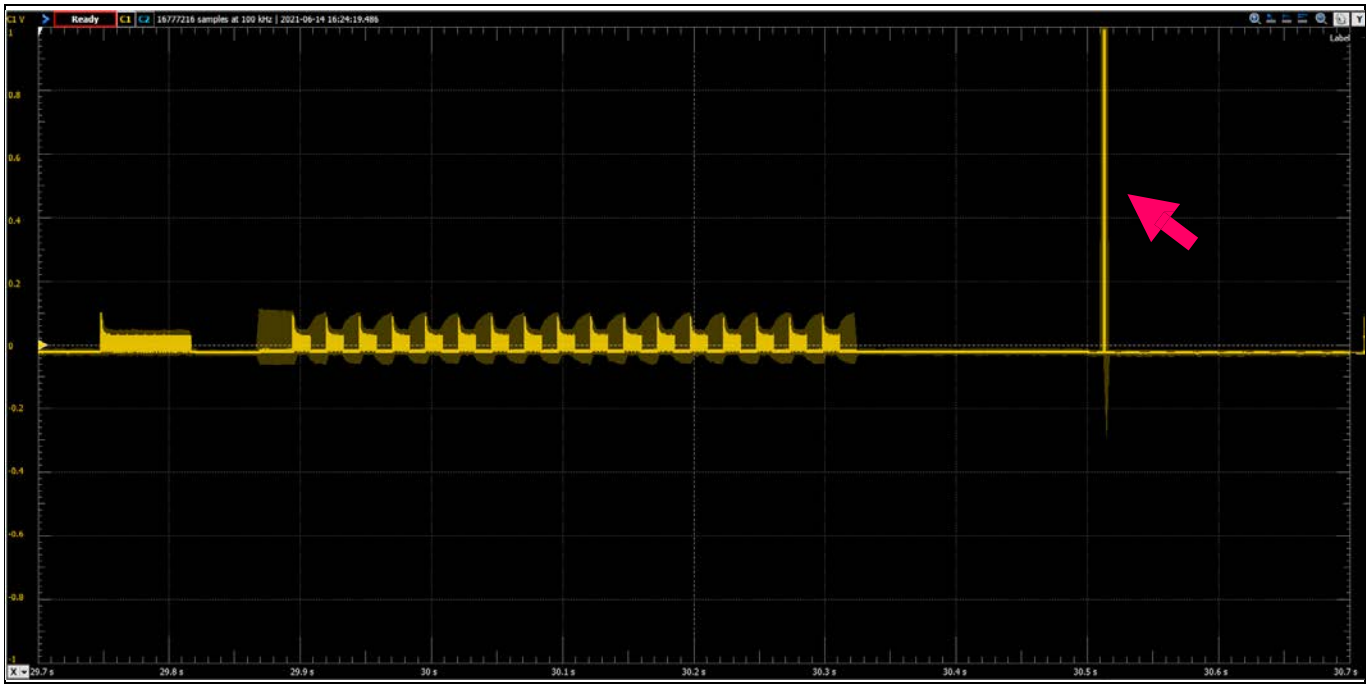


Figure 16　シングル パルス モードでの PWM 出力

7 PWM インターフェース

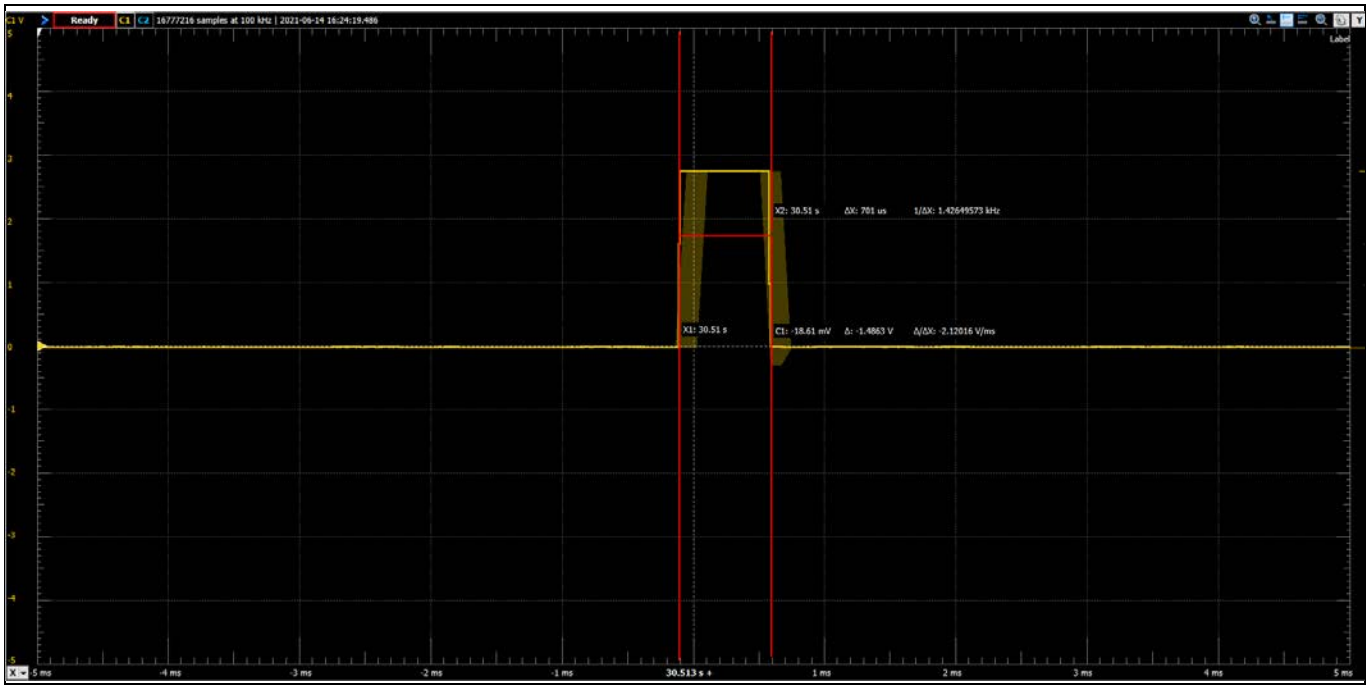


Figure 17　シングル パルス モードでの PWM 出力 (パルスにズーム)

Figure 15 の例で測定された PWM パルスの高デューティ時間は 701 µs です。 1.25 µs が 1 ppm に等しい分解能から計算すると、計算された CO2 濃度の読み出し値は 560 ppm です。

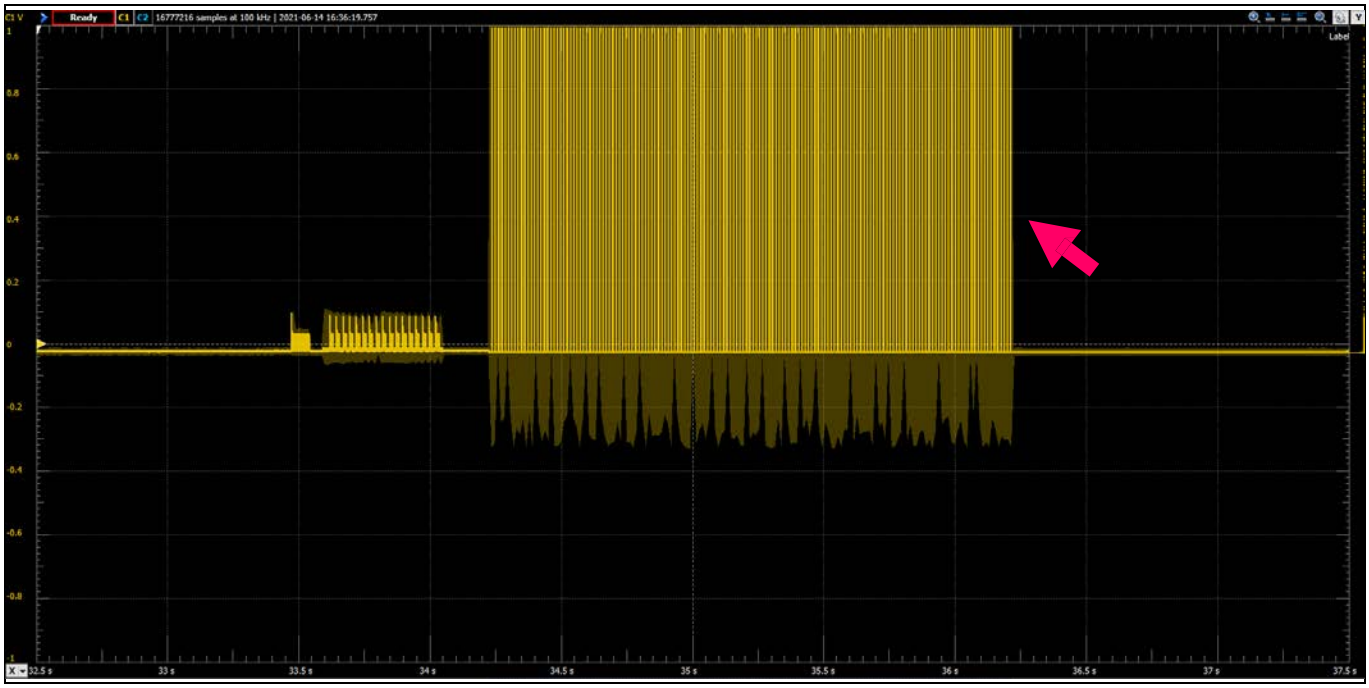


Figure 18　トレイン パルス モードでの PWM 出力

7 PWM インターフェース

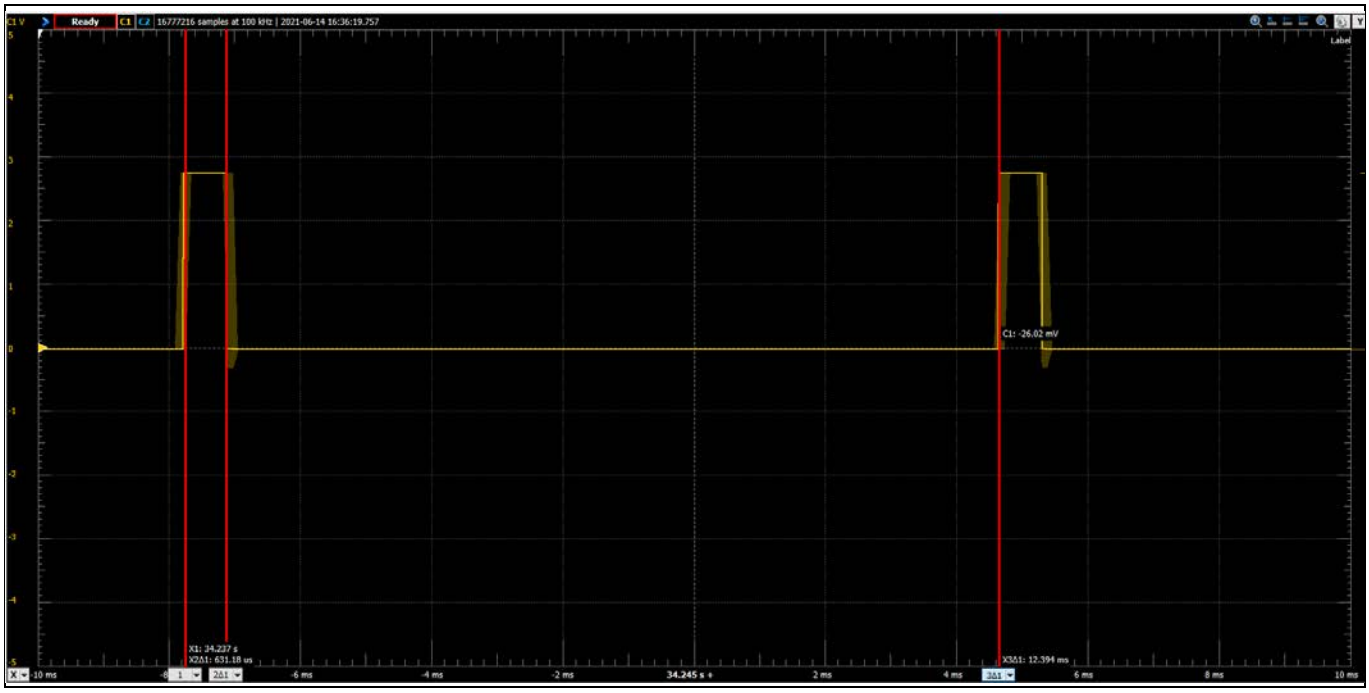


Figure 19　トレイン パルス モードでの PWM 出力 (パルスにズーム)

Figure 17 の例で測定された PWM パルス トレインの 1 つの PWM パルスの高デューティ時間は 631.18 µs です。測定された低デューティ時間は 11.76 ms で、デューティサイクルは 5.37% (= 537 ppm) です。

技術的な質問がある場合は、コミュニティフォーラムにアクセスして、同様の質問がすでに投稿されているかどうかを確認するか、新しい質問を作成してください。

https://community.infineon.com/t5/CO-sensor/bd-p/CO2Sensors

改訂履歴

版数	発行日	変更内容
V 2.1	2024-07-10	本版は英語版 AN_2011_PL38_2011_134235 Programming guide for XENSIV™ PAS V2.1 について、インフィニオン DEVELOPER COMMUNITY の参画者によって日本語に翻訳されたドキュメントです。
V2.2	2025-08-25	本版は英語版 AN_2011_PL38_2011_134235 Programming guide for XENSIV™ PAS V2.2 を翻訳した日本語版 V2.2 です。

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-08-25
Published by
Infineon Technologies AG
81726 Munich, Germany

© 2025 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?
Email: erratum@infineon.com

Document reference
AN_2011_PL38_2011_134235_JA

重要事項

本手引書に記載された本製品の使用に関する手引きとして提供されるものであり、いかなる場合も、本製品における特定の機能性能や品質について保証するものではありません。本製品の使用の前に、当該手引書の受領者は実際の使用環境の下であらゆる本製品の機能及びその他本手引書に記された一切の技術的情報について確認する義務が有ります。インフィニオンテクノロジーズはここに当該手引書内で記される情報につき、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

警告事項

技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。
インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。