参考資料

User's Guide

F28P55x および F28P65x 向け、デジタル電源アプリケーション用、電力計測ライブラリ (E-Metrology) ユーザー ガイド

TEXAS INSTRUMENTS

概要

現代の電気システムは、複雑な波形のリアルタイム処理や計算集約型アルゴリズムを含む、従来型のマイコンにとっては挑戦となるような、高精度の計測と電力品質 (PQ) の包括的な解析を必要とします。このホワイトペーパーでは、内蔵された 32 ビット浮動小数点ユニット (FPU32)、三角関数演算ユニット (TMU)、および固有の DSP 機能を活用してこれらの課題を克服する、テキサス インスツルメンツの C2000™ リアルタイム マイコン向けの高度な計測ソフトウェア ライブラリを紹介します。このライブラリは、RMS 電圧および電流の正確な値、多相の有効電力と無効電力、皮相電力と電力量、力率、ライン周波数、サグ / スウェル検出、高精度 FFT ベースの全高調波歪み (THD) など、豊富な高精度測定セットを提供します。これらはすべて、高精度の 32 ビット浮動小数点演算を使用して計算されます。TI の C2000 マイコンは、これらのデバイスの効率的な分割アーキテクチャにより、優れた性能を発揮します。このアーキテクチャでは、タイムクリティカルなバックグラウンド データの蓄積 (概念的には ADC の割り込みで駆動される) と、複雑なフォアグラウンド計算を分離しています。この資料では、要件の厳しい計測および PQ アプリケーションにおいて、汎用マイコンに比べて C2000 マイコンが非常に優れた性能と精度を発揮すること (ネイティブ浮動小数点 FFT 実行、専用 DSP 命令など) と、汎用マイコンではなく TI の C2000 マイコンを選択する明確な利点を明らかにします。

目次

商標

1 概要

スマートグリッド、再生可能エネルギーの統合、高度な産業用オートメーションなどの電気インフラの進化により、電気的パラメータ測定の精度と包括性に対する要求がますます厳しくなっています。正確な測定 (計測学) はもはや、基本的な電力消費量の測定だけに留まりません。現在の計測学には、グリッドの安定性、機器の耐用期間、運用効率を維持するために不可欠な詳細な電力品質 (PQ) 分析が含まれています。RMS 電圧と RMS 電流、有効 / 無効電力、力率、周波数、高調波歪みなどのパラメータにより、電気システムの健全性と性能に関する重要な洞察が得られます。ただし、堅牢な計測設計と PQ 分析プロセスを実装すると、技術的に大きな障壁が生じます。現代の電気負荷は多くの場合、非線形であり、電流や電圧の波形が歪んだ高調波を多く含むものです。これらの複雑な信号をリアルタイムで高精度処理すると同時に、高調波分析のための高速フーリエ変換 (FFT) や、真の RMS と電力計算のための高精度のドット積蓄積などの演算集中型アルゴリズムを実行すると、従来のマイコンの処理能力の限界に達します。さらに、計測規格で要求される高精度を達成するには、多くの場合、浮動小数点演算が必要となり、特にリソース制約のある組込みプラットフォームでは、設計の複雑さが増します。

2 ライブラリに関する注記

このライブラリは、テキサス インスツルメンツの C2000 マイコンを使用した計測アルゴリズムのコンセプトを実証するためのリファレンス コードとして提供されています。このライブラリは、量産対応のソリューションではなく、十分な修正と検証を行うことなく、製品の出荷に使用することはできません。

このコードはデフォルトで DEBUG モードで提供されており、ソフトウェアで生成された数学的に完全な正弦波を計測エンジンに提供します。このデフォルトは意図的なものです。DEBUG モードでは、ハードウェアを必要とせずに、ユーザーがアルゴリズムを検証し、測定出力を確認できます。ただし、DEBUG モードでは、実際の A/D コンバータ (ADC) 入力、実際の電流トランス、実際の分圧器、実際の商用電源信号を使用したシステムの動作を確認することはできません。

このライブラリを電力測定システムで使用するには、次の領域に対して、ハードウェア固有のエンジニアリング作業を行う必要があります。

- **ADC 構成** – システム構成 (**SysConfig**) メニューで、使用する基板のチャネル割り当て、サンプリング レート、トリガ ソース、アクイジション ウィンドウに一致するように、オンチップ ADC を構成する必要があります。デフォルト構成は、特定の LaunchPad™ 開発キットのピン配置を対象にしており、*.syscfg* ファイルと *adc_config.h* ファイルの両方に変更が加えられない限り、この構成はカスタムボードでは使用できません。
- **アナログ フロントエンド スケーリング** – *template.h* テンプレートの **PHASE_x_VOLTAGE_SCALE_FACTOR** および **PHASE_x_CURRENT_SCALE_FACTOR** フィールドの値はプレースホルダです。次の値のリストは、使用する実際のハードウェアから取得する必要があります。
  - 電圧分圧比
  - 変流器
  - シャント抵抗
  - オペアンプ (op-amp) ゲイン
  - ADC リファレンス電圧

  不正確なスケール係数を使用すると、数値的には妥当に見えても、不正確な読み取り値が生成されます。
- **ADC 較正** – 2 点較正 (DC オフセットとゲイン) は、セクション 7.2.2 に示すように、実際のハードウェアに対して実行する必要があります。
- **位相補正** – 電流トランスとアナログ フィルタ段により、電圧測定パスと電流測定パスの間に位相シフトが生じます。この位相シフトは、有効電力と力率の精度に直接影響を及ぼします。
- **外部 ADC のサポート** – 設計がオンチップ ADC ではなく、外部デルタ シグマまたは逐次比較型 (SAR) ADC を使用する場合、ADC ドライバと接続するために **ADC_readMetrologyData ()** 関数を完全に書き直す必要があります。ライブラリ アーキテクチャはこの関数の書き換えをサポートしていますが、*adc_config.c* ファイルの設定は別の ADC ですぐには機能しません。
- **ISR のタイミング** – 計測用のバックグラウンド タスクは、正確に SAMPLE_RATE サンプル/秒で実行する必要があります。デフォルトでは **CPUTimer0** タスクが使用されます。システムが別のタイマ、ダイレクト メモリ アクセス (DMA) トリガ、または ADC 変換終了割り込みを使用する場合、*metrology_main_file.c* ファイルの割り込みサービス ルーチン (ISR) の設定は、それに応じて変更する必要があります。

デフォルト値は、実際のハードウェア リファレンスに照らして検証されていません。これらの値は、DEBUG モードのデモで数値的に妥当な出力を生成できるように設定されています。*template.h* テンプレートと *adc_config.h* ファイルのすべての数値は、仕様ではなく、参考値として利用ください。

3 ライブラリ アーキテクチャ

継続的な電力監視の厳格なリアルタイム要件を満たすために、重要なデータが失われないように、この計測ライブラリでは効率的な分割処理アーキテクチャが採用されています。この設計では、時間に制約のあるタスクを、頻度の低い、より計算量の多い処理から切り離すことができます。また、リソース使用率を最適化し、システムの応答性を維持します。このアーキテクチャは、大きく次の 2 つのメイン処理レイヤに分けられます。

- バックグラウンド処理
- フォアグラウンド処理

3.1 バックグラウンド処理

バックグラウンド処理は高周波で実行され、通常は A/D コンバータ (ADC) のサンプリング レートと同期します。バックグラウンド処理タスクは、割り込みサービス ルーチン (ISR) 内で処理されます。このレイヤの主な役割は次のとおりです。

- **未加工サンプルの取得**:ADC から電圧と電流の最新のサンプルを読み取ります。
  - 基本的な前処理:DC 推定や DC フィルタリングなど、未加工サンプルに対する最小限のタイム クリティカルな処理を実行して、DC バイアスを除去します。
- **コアの累算**:定義された測定ウィンドウ (通常は電源の 1 周期または複数の周期) 全体にわたる基本ドット積 (電圧サンプルの 2 乗の合計、電流サンプルの 2 乗の合計、電圧と電流の積の合計など) の計算とその累算。これらの累算データが、以後のほとんどの計測学の計算の基礎になります。
- **データ バッファと累算器**:データ バッファの管理と、フォアグラウンド処理への安全な転送のために、累算データの準備。

バックグラウンド処理の重要な設計原理は、高いサンプリング忠実度を維持するために、ISR の実行時間をできるだけ短くし、システムが他の重要な割り込みに応答できるようにすることです。

3.2 フォアグラウンド処理

フォアグラウンド処理は、メイン アプリケーションループ内でより低い周波数で動作します。フォアグラウンド タスクは、バックグラウンド処理がデータのブロック全体を蓄積するとトリガされます。このレイヤでは、次のような演算集中型の計算が処理されます。

- **累算データの取得**:バックグラウンド処理のタスクによって記録されたドット積やその他の関連データを一貫して安全に読み取ります。
- **キャリブレーションの適用**:センサの誤差とアナログ フロントエンドの変動を補償するため、あらかじめ決められた較正係数を使用して未加工の累算値を補正します。
- **最終パラメータの計算**:RMS 電圧と RMS 電流、有効電力、無効電力、皮相電力、力率、周波数、THD、サグおよびスウェル イベントなどの計測パラメータ全体を、キャリブレーション済みのデータを使用して計算します。
- **電力量の累算**:時間の経過に応じて電力値を積分し、電力消費量を計算します。
  - 結果の更新:ディスプレイ、ロギング、通信のためにアプリケーションで最終的な較正測定値を使用できるようにします。

3.3 データ転送メカニズム

この分割アーキテクチャにおける重要な要素として、バックグラウンド ISR とフォアグラウンド タスク間の堅牢なデータ転送メカニズムがあります。このライブラリは、データの整合性を維持するために、ダブル バッファリングと呼ばれる同期受渡技術を採用しています。処理の内容は以下のとおりです。

1. バックグラウンド ISR は、1 つ 1 つのサンプル データに基づきメインの累算器セットを常に更新します。
2. 完全な計測ブロック (たとえば、定義されたサンプル数または主電源サイクル) が累算されると、バックグラウンド タスクにより、現在の値がこれらの 1 次の累算器から 2 次の「ログ」に記録される変数セットにコピーされます。
3. バックグラウンド タスクによってステータス フラグが設定された後、フォアグラウンド タスクに対して、「ログ」に記録される変数に新しいデータ ブロックが準備されたことが通知されます。
4. メイン アプリケーション ループで動作するフォアグラウンド タスクは、このステータス フラグを定期的にチェックします。フラグの設定が検出されることにより、フォアグランド タスクは、「ログ」に記録される累算器において、完全かつ一貫性のあるデータセットが使用可能になったことを認識します。その後、フォアグラウンド タスクはこのデータを使用して計算を完了します。

4 測定値の計算

実際の測定値の計算は、*meterology_caluclations.c* ファイルで行われます。

- このモジュールには、計測アルゴリズムを実装する数学関数ライブラリが含まれています。
- このモジュールは、ドット積、サンプル数、キャリブレーションデータの累算値などの入力を受け取ります。
- このモジュールは、電気的パラメータ (**calculateRMSVoltage**、**calculateActivePower**、**calculateFrequency_PLL** など) の計算値を出力します。

5 構成とデータ構造

これらのファイルは、定数 (電圧または周波数の公称値、サンプル レートなど)、機能を有効にするプリプロセッサ マクロ (**HARMONICS_SUPPORT**、**NEUTRAL_MONITOR_SUPPORT** など)、すべての計測関連情報を格納する主要なデータ構造を定義します。

- **metrologyData**:トップレベルのグローバル構造体。インスタンスは、すべての相、中性線、合計値、およびシステム状態のすべてのデータを保持します。
- **phaseMetrology** または **neutralMetrology**:各相および中性線のパラメータ (params) と読み取り値 (readings) を保持する **metrologyData** 内の構造体。
  - **params**:構成情報、バックグラウンドの中間累積情報 (dotProd 配列など) 、キャリブレーションから派生した値、および状態変数 (PLL、サグとスウェルなど) を格納します。
  - **readings**:最終的に計算された計測値 (RMS、電力、PF など) を格納します。
- **calibrationData** (*metrology_nv_structs.h*):較正係数を保持します。

6 較正モジュール

較正モジュールは、キャリブレーション データの読み込み、適用、保存 (概念として) を管理します。

- **Metrology_setup_init** (*metrology_setup.c* 内の同様の関数):データ構造を初期化し、位相補正などの較正の初期設定を適用します。

7 ユーザー固有の構成

ユーザーは、使用するアプリケーションの要件に合わせて計測パラメータを計算するように計測ライブラリを設定できます。ユーザー用の構成はすべて、次の 2 つのファイルに含まれています。

- *template.h*
- *adc_config.h*

これ以外のライブラリ ファイルは変更しないでください。

このライブラリは、コンパイル時に選択される 2 つの動作モードをサポートしています。

- DEBUG モード
- ADC モード

モードは、*template.h* ファイルの DEBUG プリプロセッサ シンボルで制御されます。

7.1 デバッグ モード

DEBUG モードでは、実際の ADC ハードウェアをソフトウェア信号ジェネレータに置き換えます。DEBUG モードは、初期の立ち上げ、アルゴリズム検証、およびハードウェアの接続前に計測エンジンが正しい結果を生成することの検証を目的としています。ADC ハードウェアやアナログ フロントエンドは不要です。

7.1.1 デバッグ モードの機能

DEBUG モードが定義されている場合、**cpuTimer0ISR** 関数は **ADC_readMetrologyData()** 関数の代わりに **get_debugData()** 関数を呼び出します。**get_debugData()** 関数は、起動時に **debugInit()** 関数で入力される 2 つの参照テーブルから、事前に計算されたサンプルを提供します。

debugInit() 関数は、1 周期 [0, 2*PI] にわたって等間隔に配置された位相角で C 標準関数 **sinf()** を計算し、voltage[] と current[] の各配列 (配列長:50Hz/8000Hz システムで **SAMPLES_PER_CYCLE** = 160 サンプル) に値を格納します。

- voltage[i] = (V_RMS × SQRT2) / (V_SCALING) × sinf(t)
- current[i] = ((I_RMS × SQRT2) / I_SCALING) × sinf(t - phaseDiff)

ここで、(i / SAMPLES_PER_CYCLE) × 2*PI および phaseDiff = acosf(POWER_FACTOR) です。

サンプルは、格納前に template.h ファイルで定義されている **V_SCALING** および **I_SCALING** マクロ変数で正規化されるため、計測エンジンに入力される値は、実際の ADC サンプルと同じく単位あたりの形式になります。次に、計測エンジンはキャリブレーション スケール係数 (**VscaleFactor** および **IscaleFactor**) を適用して、物理単位を再構築します。DEBUG モードでは、*metrology_main_file.c* ファイルの **main()** 関数で、これらのスケール係数が **V_SCALING** および **I_SCALING** パラメータと一致するようにオーバーライドされ、表示された読み取り値が **V_RMS** と **I_RMS** の値と正確に等しくなるようにします。

debugData() 関数を取得する各呼び出しは、循環参照テーブルを介して一連のインデックス ポインタ (**voltageIndex[]** および **currentIndex[]**) を進め、ISR は正しいサンプル レートで連続的な周期波形を認識します。

多相トポロジでは、異なる開始位置で **voltageIndex[1]** と **voltageIndex[2]** を初期化することで、各相を相 A から **PHASE_INCREMENT** = 53 サンプル分 (53/160×360 = 119.25 度の位相間隔に相当) オフセットします。

7.1.2 DEBUG モード パラメータの設定方法

すべての DEBUG パラメータは、*metrology_main_file.c* ファイルの先頭にある **#ifdef DEBUG** ブロック内に定義されます。

- **V_RMS** – 生成される正弦波の二乗平均平方根 (RMS) 電圧。単位:ボルト。デフォルト:120.0.
- **I_RMS** – 生成された正弦波の RMS 電流。単位:アンペア。デフォルト:10.0.
- **POWER_FACTOR** – シミュレーションされた負荷の力率 (0.0 ～ 1.0)。電圧と電流の間の位相角を phaseDiff = acosf(POWER_FACTOR) として設定します。デフォルト:0.5 (遅れ、位相差 60°)。
- **V_SCALING** – 生成されたサンプルを正規化するための電圧スケール係数。*template.h* テンプレートの **PHASE_A_VOLTAGE_SCALE_FACTOR** と一致している必要があります。デフォルト:1585.2.
- **I_SCALING** – 生成されたサンプルを正規化するための電流スケール係数。*template.h* テンプレートの **PHASE_A_CURRENT_SCALE_FACTOR** と一致している必要があります。デフォルト:125.0.
- **SAMPLES_PER_CYCLE** – ルックアップ テーブル内の商用電源サイクルあたりのサンプル数。**SAMPLE_RATE | MAINS_NOMINAL_FREQUENCY** と同じでなければなりませんデフォルト:160 (8000Hz / 50Hz)。
- **PHASE_INCREMENT** – ルックアップ テーブル内の各相間の開始インデックスのオフセット。デフォルト:53.

生成された信号への高調波の注入

全高調波歪み (THD) の計算をテストするため、*metrication_main_file.c* ファイルの **#ifdef DEBUG** ブロック内に **INJECT_HARMONIC** を定義して、生成された波形に高調波成分を注入できます。**INJECT_HARMONIC** が定義されると、**debugInit()** 関数の処理時に電圧と電流の両方のルックアップ テーブルに正弦波成分が追加されます。

- voltage[i] ± (V_RMS × SQRT2) / (V_SCALING) × HARMONIC_PERCENT × sinf(HARMONIC_ORDER × t)
- current[i] ± (I_RMS × SQRT2) / (I_SCALING) × HARMONIC_PERCENT × sinf(HARMONIC_ORDER × t - phaseDiff)

- **HARMONIC_ORDER** – 注入する高調波の整数次数 (たとえば、5 を指定すると、250Hz で第 5 次高調波が注入されます)。デフォルト:5.
- **HARMONIC_PERCENT** – 基本波の振幅に対する、注入された高調波の振幅の割合 (例:0.50 = 50%)。予測される THD の結果は、この値とほぼ等しくなります。デフォルト:0.50.

debugInit() 関数でループ内に手動で項を追加することで、複数の高調波を注入できます。

7.2 ADC モード

ADC モードでは、ライブラリは「**SysConfig**」メニューで構成されたオンチップ ADC ペリフェラルから、実際の電圧と電流のサンプルを読み取ります。これには、次の 2 つのケースがあります。

- オンチップ ADC を直接使用する
- オンチップ ADC を外部 ADC に置き換える

7.2.1 オンチップ ADC の使用

オンチップ ADC は、プロジェクトに含まれているシステム構成ファイル (*.syscfg*) で構成されます。「**SysConfig**」メニューで *.syscfg* ファイルを開き、以下のように構成します。

- ADC モジュールの選択 (デフォルトでは、電圧は ADCA、電流は ADCB)
- SOC (変換開始) トリガ ソース –計測 ISR で使用されるのと同じ **CPUTimer0** 周期に設定します。
- サンプリング ウィンドウ幅 – ADC 入力回路のセトリング タイムを満たすように設定します。
- 必要に応じて、アクイジション ウィンドウと EOC (変換終了) 割り込みを設定します。

.syscfg ファイルを保存した後、「**SysConfig**」メニューで ***board.h*** ヘッダー ファイルと ***board.c*** ソース ファイルを再生成すると、ライブラリで使用されるペリフェラル名 (**myADC0_RESULT_BASE**、**myADC0_SOC0** など) が含まれるようになります。

adc_config.h ヘッダー ファイルを開き、生成された名前を各フェーズのライブラリ マクロに割り当てます。

A 相 A (常に必要) :

- **ADC_VA_RESULT | ADC_VA_SOC** – A 相の電圧チャネルの結果ベースと SOC 番号。
- **ADC_IA_BASE | ADC_IA_RESULT | ADC_IA_SOC** – A 相の電流チャネルのベース アドレス、結果ベース、SOC 番号。

B 相 (**TWO_PHASE_SUPPORT**、**THREE_PHASE_SUPPORT**、または **SPLIT_PHASE_SUPPORT** に必要) :

- **ADC_VB_RESULT | ADC_VB_SOC** –B 相の電圧チャネル。
- **ADC_IB_RESULT | ADC_IB_SOC** – B 相の電流チャネル。

C 相 (**THREE_PHASE_SUPPORT** にのみ必要。現在コメントアウトされています。*SysConfig* GUI で 3 つ目の ADC を追加した場合は、**THREE_PHASE_SUPPORT** のコメントを削除します):

- **ADC_VC_RESULT | ADC_VC_SOC** – C 相の電圧チャネル。最初に、**SysConfig** メニューで 3 つ目の ADC SOC を追加する必要があります。
- **ADC_IC_RESULT | ADC_IC_SOC** – C 相の電流チャネル。

7.2.2 2 点較正

オンチップ ADC は、正確な測定を行うために 2 つのキャリブレーション ステップを必要とします。

- オフセット キャリブレーション
- ゲイン較正

これらを組み合わせることで 2 点較正が形成され、アナログ フロントエンドの DC バイアスを除去し、抵抗デバイダ、電流トランス、またはオペアンプ チェーンにより生じるゲイン誤差を補正します。

7.2.2.1 ステップ 1 — オフセット較正 (ゼロ入力ポイント)

ADC 入力に AC 信号が印加されていない場合:

1. 起動時に、**App_startDataCollection()** 関数の前に、**ADC_calibrateOffsets()** 関数を 1 回呼び出します。
2. この関数は、電圧および電流チャネルで 1,000 回連続の ADC 読み取り値を平均化し、DC 中点を求めます。
   - **ADC_calibrateOffsets();** // AC 入力を接続せずに呼び出し
3. この関数は、測定された平均値を内部変数 **g_voltageOffset** と **g_currentOffset** に格納します。これらの平均値は、以後の **ADC_readNormalized()** 関数でのすべての ADC 読み取りから減算されます。Vref/2 (1.65V) を中心とするバイポーラ信号を持つ 12 ビット ADC の理論的な中点は 2048 ですが、実際には部品の許容誤差により、この中点がシフトします。
4. **ADC_calibrateOffsets()** 関数が呼び出されていない場合、*adc_config.h* ファイルで定義されているように、**ADC_VOLTAGE_DC_OFFSET** および **ADC_CURRENT_DC_OFFSET** のデフォルト値が使用されます (いずれもデフォルトは 2048.0)。

7.2.2.2 ステップ 2 – ゲイン較正 (フルスケール ポイント)

1. 既知の AC リファレンス信号を印加し、ライブラリ出力を既知の値と比較します。
2. 次を使用して、*adc_config.h* ファイルの **ADC_VOLTAGE_GAIN** および **ADC_CURRENT_GAIN** を調整します。
   - gain_new = gain_old × (measured_reading / known_reference)
     - たとえば、既知の 120.0V リファレンスに対して、ライブラリが 118.2V RMS を報告した場合、gain_new = 2056.8× (120.0/118.2) = 2087.1 です
3. **ADC_CURRENT_GAIN** を使用して、電流について同じ処理を繰り返します。***ADC_readNormalized()*** 関数は、これらのゲインを次のように適用します。
   - normalized = (raw_count - dc_offset) / gain
   - 結果として得られる正規化値は、デバッグ信号ジェネレータ出力と同じ範囲の、単位あたりの浮動小数点となり、計測エンジンはこれを (template.h の) VscaleFactor / IscaleFactor でスケーリングして物理単位を生成します。

位相補正 (電流トランスの位相シフト):

電流トランスとアナログ フィルタ段を採用すると、電圧チャネルと電流チャネルの間で、測定可能な位相シフトがわずかに発生します。このような位相シフトは、力率や電力精度に直接的な影響を及ぼします。これは、**currentSensorCalibrationData** (*metrology_nv_structs.h*) の **phaseOffset** フィールドで補正されます。

デフォルトの位相補正値は、***metrology_calibration_defaults.h*** で定義されています。

- **DEFAULT_BASE_PHASE_A_CORRECTION** – A 相の位相補正。単位:マイクロ秒 × **SAMPLE_RATE** × 256。デフォルト:0.0 (DEBUG モードと適切なセンサの場合)。2.4 マイクロ秒の遅れがある実際の CT の場合、次のようになります。-2.4e-6 × 8000 × 256。

許容範囲は、約 –125 マイクロ秒 +125 マイクロ秒です。ユニティ力率で高精度電力アナライザを使用して実際の位相誤差を測定し、それに応じて補正を設定します。

7.2.3 外部 ADC の使用

オンチップ ADC を外部 ADC (SPI 経由で接続されたデルタ シグマ デバイスなど) に置き換える場合は、以下の変更が必要です。

1. *adc_config.c* ファイル – **ADC_readMetrologyData()** 関数内の **ADC_readNormalized ()** 関数呼び出しを外部 ADC ドライバの呼び出しに置き換えます。この関数は、正規化された浮動小数点 (ゼロを中心とし、フルスケールは約 ±1.0) を **workingData→rawVoltageData[PHASE_x]** と **workingData→rawCurrentData[PHASE_x]** に書き込む必要があります。正規化の規則は次のとおりです。

   normalized = (raw_sample - dc_offset) / gain

   ここで、dc_offset と gain は、上記の同じ 2 点較正定数であり、外部 ADC の未加工の出力カウントに適用されます。
2. *adc_config.h* ファイル – **SysConfig** メニューで生成された名前は適用されないため、**ADC_VA_RESULT | ADC_VA_SOC** マクロ定義を削除するか、置換します。代わりに、ドライバに必要な定数を定義します。**ADC_VOLTAGE_DC_OFFSET**、**ADC_CURRENT_DC_OFFSET**、**ADC_VOLTAGE_GAIN**、**ADC_CURRENT_GAIN** は有効のままで、外部 ADC の分解能とフロントエンド ゲインに一致するように設定する必要があります。
3. *.syscfg ファイル* – オンチップ ADC を使用しない場合は、メモリとクロック リソースの消費を避けるため、ADCA および ADCB ペリフェラル インスタンスを削除します。
4. *metrication_main_file.c* ファイル – 外部 ADC がタイマ トリガではなくデータ準備完了割り込みを使用する場合は、**cpuTimer0ISR** 関数の **ADC_readMetrologyData()** および **Metrology_perSampleProcessing()** 関数呼び出しを外部 ADC 割り込みハンドラに移動します。ISR が SAMPLE_RATE と同じレートで実行されることを確認します。
5. *template.h* ファイル – **PHASE_x_VOLTAGE_SCALE_FACTOR** および **PHASE_x_CURRENT_SCALE_FACTOR** マクロは、計測エンジンがこれらのマクロを使用して正規化されたサンプルをボルトとアンペアに変換するため、外部 ADC アナログ フロントエンドの物理スケーリングを反映する必要があります。

7.3 モード間の切り替え

DEBUG モードから実際の ADC 入力に切り替えるには、*mmersioning_main_file.c* ファイルの先頭付近にある ***#define*** DEBUG 行をコメントアウトするか、削除します。

//#define DEBUG

これで、ISR は **get_debugData()** 関数の代わりに **ADC_readMetrologyData()** 関数を呼び出すようになり、**#ifdef DEBUG** マクロで補正スケーリング係数が **main()** 関数をオーバーライドする処理 (モード切り替えのフックとして機能) は適用されなくなります。*template.h* ファイルのスケール係数が直接使用されます。

7.4 THD アルゴリズムの選択

このライブラリには、THD 計算の方法が 3 つ用意されています。この方法は、*template.h* テンプレートの **HARMONICS_SUPPORT** および **USE_GOERTZEL_THD** マクロによってコンパイル時に選択されます。

- 方法 1–PLL に基づく (HARMONICS_SUPPORT マクロが定義されていない場合のデフォルト)
- 方法 2–Goertzel DFT (HARMONICS_SUPPORT と USE_GOERTZEL_THD マクロの両方が定義されている場合)
- 方法 3 – 2048 ポイント実数 FFT (HARMONICS_SUPPORT マクロが定義されているが、USE_GOERTZEL_THD マクロが定義されていない場合)

7.4.1 方法 1–PLL に基づく (HARMONICS_SUPPORT マクロが定義されていない場合のデフォルト)

方法 1 では、次の関係を使用して THD を計算します。

$$THD = \sqrt{RMS\_total^2 - RMS\_fundamental^2} / RMS\_fundamental$$

ここで、RMS_total は未処理の波形のドット積から計算され、50Hz または 60Hz の成分を追跡する PLL を使用して RMS_fundamental が抽出されます。この方法は計算的に軽量ですが、ゼロ交差トリガの積分ウィンドウと PLL サイクル境界の間のウィンドウ ミスマッチが原因で、クリーンな正弦入力では約 0.1% の固有誤差フロアがあります。結果は、**phaseReadings** 構造体の **voltageTHD** フィールドと **currentTHD** フィールドに返されます。

メモリが制約されている場合、または THD が定性的な指標である場合は、方法 1 を使用します。

7.4.2 方法 2–Goertzel DFT (HARMONICS_SUPPORT と USE_GOERTZEL_THD マクロの両方が定義されている場合)

方法 2 では、Goertzel アルゴリズムを使用して、1600 サンプル ウィンドウ (50Hz/8000Hz で 10 サイクル、60Hz で 12 サイクル) にわたって個々の高調波ビン (2 次高調波から 50 次高調波) の DFT を計算します。Goertzel アルゴリズムは、単一の周波数ビンに対して FFT と同じ結果を生成する再帰フィルタですが、メモリ オーバーヘッドが小さく、ウィンドウ関数からのスペクトル リークがありません。THD は次のように計算されます。

$$THD = \sqrt{sum(H2^2 + H3^2 + ... + H50^2)} / H1$$

ここで、H1 は基本振幅、H2 ... H50 は高調波振幅です。結果は、**phaseReadings** 構造体の **vTHDfft** フィールドと **iTHDfft** フィールドに返されます。

この方法は 10 ～ 12 サイクル以内に安定し、IEC 61000-4-30 規格の Class A 精度要件を満たすため、量産計測アプリケーションに推奨される方法です。

7.4.3 方法 3 – 2048 ポイント実数 FFT (HARMONICS_SUPPORT マクロが定義されているが、USE_GOERTZEL_THD マクロが定義されていない場合)

方法 3 では、完全な 2048 ポイントの実数 FFT を使用し、変換の前に Hann ウィンドウを適用します。FFT ウィンドウは、50Hz / 8000Hz (2048 / 8000×50 = 12.8) で約 12.8 サイクルです。FFT 後、高調波は IEC 61000-4-7 規格に従ってグループ化され (各高調波を中心とする 5Hz のビン)、グループ化された高調波ビンから THD が計算されます。結果は、**vTHDfft** および **iTHDfft** フィールドに返されます。

この方法では完全なスペクトル情報が得られますが、Goertzel DFT 法に比べてかなり多くの RAM (実数および虚数 FFT バッファ用の 2048 要素の浮動小数点配列を 2 個) と、より長いセトリング タイムが必要になります。方法 3 は、THD を超える完全なスペクトル解析が必要な場合のみ使用してください。

7.5 トポロジの選択

測定対象のシステムに一致するトポロジ マクロを 1 つだけ定義します。

- **SINGLE_PHASE_SUPPORT** – 単相、A 相のみ (デフォルト)。
- **TWO_PHASE_SUPPORT** – 2 相、A 相と B 相
- **THREE_PHASE_SUPPORT** – 3 相 (A 相、B 相、C 相)
- **SPLIT_PHASE_SUPPORT** – 相を分割。A 相と B 相で中性線を共有。

7.6 機能を有効にする

個々の測定は、対応するマクロを定義または定義解除することで有効または無効にできます。次の機能が使用可能です。これらの機能は、*template.h* ファイルで定義されている必要があります。

- **HARMONICS_SUPPORT** – FFT ベースの高調波分析。THD の計算に必要です。
- **NEUTRAL_MONITOR_SUPPORT** – 中性電流監視。
- **VRMS_SUPPORT** – RMS 電圧。
- **IRMS_SUPPORT** – RMS 電流。
- **ACTIVE_POWER_SUPPORT** – 有効電源。
- **REACTIVE_POWER_SUPPORT** – 無効電力。
- **APPARENT_POWER_SUPPORT** – 皮相電力。
- **ACTIVE_ENERGY_SUPPORT** – 有効エネルギーの蓄積。
- **REACTIVE_ENERGY_SUPPORT** – 無効エネルギーの蓄積。
- **POWER_FACTOR_SUPPORT** – 力率。
- **POWER_FACTOR_ANGLE_SUPPORT** – 力率角度。
- **FREQUENCY_SUPPORT** – ライン周波数。
- **SAG_SWELL_SUPPORT** – 電圧サグおよびスウェル検出。
- **FUNDAMENTAL_PARAMETERS** – すべての基本測定 (50Hz または 60Hz 成分のみ) を有効にします。
- **VOLTAGE_THD_SUPPORT** – 電圧 THD。HARMONICS_SUPPORT マクロが必要です。
- **CURRENT_THD_SUPPORT** – 現在の THD。HARMONICS_SUPPORT マクロが必要です。

THREE_PHASE_SUPPORT マクロが定義されている場合は、以下の機能を追加で使用できます。

- **LINETOLINE_VOLTAGE_SUPPORT** –ライン間電圧。
- **AGGREGATE_POWER_FACTOR** – すべての相にわたって力率を集約します。
- **CURRENT_VECTOR_SUM** –電流ベクトル和。
- **PHASE_TO_PHASE_ANGLE_SUPPORT** – 相電圧間の角度

7.7 システム パラメータ

- **SAMPLE_RATE** – ADC サンプル レート (Hz)。許容値:2000、2930、3906、4000、5859、6000、6400、7812、8000。デフォルト:8000.
- **MAINS_NOMINAL_FREQUENCY** – 公称商用電源の周波数 (Hz) (50 または 60)。デフォルト:50.
- **MAINS_MIN_FREQUENCY | MAINS_MAX_FREQUENCY** – ゼロ交差検出の有効な周波数範囲。デフォルト: 40.0Hz ～ 70.0Hz。
- **MAINS_NOMINAL_VOLTAGE** – 公称 RMS 電圧 (V)。サグまたはスウェル検出の基準として使用されます。デフォルト:230.0V
- **ENERGY_PULSE_CONSTANT** – 1kWh あたりのエネルギー パルス レート。デフォルト:6400.

7.8 各相のスケーリング係数

スケーリング係数は、正規化した ADC 値を物理的な単位に変換し、アナログ フロントエンド ハードウェア (分圧比、電流トランスまたはシャント比、またはオペアンプ ゲイン) に一致している必要があります。

template.h ファイルで次のマクロを検索します。

- **PHASE_A_VOLTAGE_SCALE_FACTOR** – A 相の正規化された ADC 単位あたりの電圧 (V)
- **PHASE_A_CURRENT_SCALE_FACTOR** – A 相の正規化された ADC 単位あたりの電流

- **PHASE_A_POWER_SCALE_FACTOR** – 電圧と電流のスケール係数の積に設定されます。

B 相と C 相には、対応するトポロジが有効化されているときにアクティブになる等価マクロ (**PHASE_B_***、**PHASE_C_***) があります。

スタートアップ時にフィルタのセトリング タイムを短縮するため、DC 推定値 (**PHASE_A_VOLTAGE_DC_ESTIMATE** と **PHASE_A_CURRENT_DC_ESTIMATE**) をおおよその DC 中点に設定することができます。不明な場合は、DC 中点を 0.0 に設定します。

AC オフセット (**PHASE_A_VOLTAGE_AC_OFFSET**、**PHASE_A_CURRENT_AC_OFFSET**、**PHASE_A_ACTIVE_POWER_OFFSET**、**PHASE_A_REACTIVE_POWER_OFFSET**) は、計算値から減算され、補正後に残差測定誤差をトリミングするために使用されます。

8 例を実行する

まず、Code Composer Studio™ (CCS) ソフトウェアでプロジェクトを作成して実行します。

1. CPU1_FLASH ビルド構成を使用して、CCS で **energy_metrology_f28p55** または **energy_metrology_f28p65** プロジェクトをビルドします。
2. プロジェクトのビルド後にプロジェクトを右選択し、「**DEBUG**」ボタンを選択します。
3. CCS デバッグ ウィンドウの「**RUN**」ボタンを使用して、プロジェクトを実行します

次に、ターゲットがブレークポイントで停止している状態、またはリアルタイム モードで実行中に、CCS で新しい式を追加します。

- 「*Expressions*」ウィンドウを開き、「*Window*」、「*Show View*」、「*Expressions*」に移動します。
- 「*Expressions*」ペインで、緑色のプラス アイコンを選択し、「*Add New Expression*」を選択します。

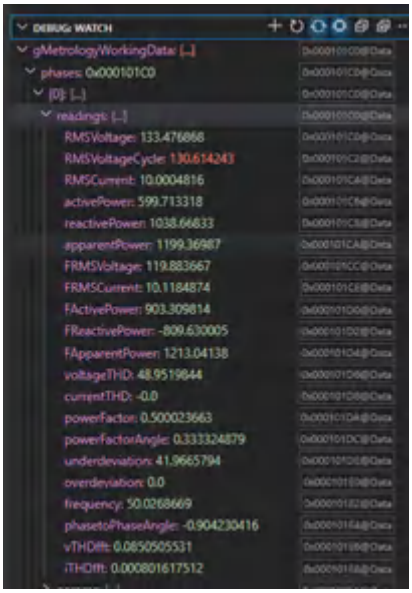

図 8-1. Code Composer Studio (CCS) ソフトウェアの「Expressions」ウィンドウ

8.1 各相のパラメータへのアクセス

gMetrologyWorkingData 構造体には、**PHASES** 列挙型でインデックス付けされた phases[] 配列が含まれています (**PHASE_ONE** = 0、**PHASE_TWO** = 1、**PHASE_THREE** = 2)。各相には、最終的な計算値を含む **phaseReadings** 型の読み取り値メンバーがあります。

表 8-1. phaseReadings ごとの計算値を表示するには

phaseReadings 名[1]	位相	計算値
gMetrologyWorkingData.phases[0].readings.RMSVoltage	相 A:	RMS 電圧[2]

表 8-1. phaseReadings ごとの計算値を表示するには (続き)

phaseReadings 名(1)	位相	計算値
gMetrologyWorkingData.phases[0].readings.activePower	相 A:	アクティブ時の消費電力
gMetrologyWorkingData.phases[0].readings.powerFactor	相 A:	力率
gMetrologyWorkingData.phases[0].readings.frequency	相 A:	周波数
PLL に基づく (HARMONICS_SUPPORT マクロが定義されていない場合)		
gMetrologyWorkingData.phases[0].readings.voltageTHD	相 A:	電圧 THD
Goertzel または FFT (HARMONICS_SUPPORT マクロが定義されている場合)		
gMetrologyWorkingData.phases[0].readings.vTHDfft	相 A:	電圧 THD

(1) 相 B では [0] を [1] に、相 C では [2] に置き換えます
(2) 相 A の RMS 電圧を表示するには、「*Expressions Watch*」ウィンドウに phaseReadings 構造を追加する必要があります。

8.2 システム全体の計測値へのアクセス

システム全体の計測値は、**gMetrologyWorkingData.totals** 構造体に次のように格納されます。

- gMetrologyWorkingData.totals.readings.activePower
- gMetrologyWorkingData.totals.readings.reactivePower
- gMetrologyWorkingData.totals.readings.apparentPower
- gMetrologyWorkingData.totals.activeEnergy.energy
- gMetrologyWorkingData.totals.reactiveEnergy.energy

8.3 相ステータス フラグへのアクセス

各フェーズには、**PHASE_STATUS** フラグのビットマスクを含むステータスフィールド (**gMetrologyWorkingData.phases[0].status**) があります。

次のリストに、*metrology_defines.h* ファイルで定義されている便利なフラグ値を示します。

- **PHASE_STATUS_NEW_LOG** (0x0001) – バックグラウンドで測定ウィンドウが完了しましたが、フォアグラウンドではまだ測定を処理していません。
- **PHASE_STATUS_V_OVERRANGE** (0x0010) – 電圧 ADC 入力が飽和しました (クリッピングを検出)。
- **PHASE_STATUS_I_OVERRANGE** (0x0020) – 電流 ADC 入力が飽和しました。
- **PHASE_STATUS_ZERO_CROSSING_MISSED** (0x4000) – 想定されたゼロ交差が有効なウィンドウ内で発生しませんでした (範囲外の周波数または信号の喪失)。

フラグがセットされているかどうかを確認するには、ステータス フィールドを 16 進数形式で表示するか、ビット単位の **AND** 式を使用します。ゼロ交差が検出されなかった場合、**gMetrologyWorkingData.phases[0].status** と 0x4000 はゼロ以外になります。

注

停止せずに真のリアルタイム監視を実現するには、CCS 連続リフレッシュ モードを有効にします (リアルタイム メモリ アクセス機能を備えた XDS デバッグ プローブが必要)。連続リフレッシュ モードを有効にするには:

1. 「*Expressions*」ウィンドウで右選択し、「*Continuous Refresh*」、「*Enable*」の順に選択します。
   - これにより、CPU が動作を継続している間、ターゲット メモリが定期的にポーリングされます。
   - これにより、バストラフィックが増加し、リアルタイムのパフォーマンスに影響を与えるおそれがあります。したがって、このモードは限定的に使用してください。

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、 TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月