

Design Guide: TIDM-02018

AM263x Arm[®] ベースの MCU デバイス向けユニバーサル モーター制御のリファレンス デザイン

概要

このリファレンス デザインは、テキサス・インスツルメンツの AM263x Arm[®] ベースの MCU 向けユニバーサル モーター制御の設計を示しています。このデザインでは、センサレス (eSMO) およびセンサ付き (インクリメンタル エンコーダ、ホール センサ) など、さまざまなタイプの FOC モーター制御方式に AM263x MCU を使用する方法を紹介しています。このデザインは、AM263x LaunchPad[™] と 3PHGANINV BoosterPack[™] を使用した低電圧設定と、AM263x controlCARD[™] と TMDSHVMTRINSPIN モーター制御キットを使用した高電圧設定という、2 つの主要なハードウェア設定をサポートしています。また、ドキュメントではカスタム ボードへの移行方法、および新しいデバイスへのプロジェクトのポーティング方法についても説明しています。

参照情報

| | |
|-----------------|------------|
| TIDM-02018 | デザイン フォルダ |
| AM2634-Q1 | プロダクト フォルダ |
| AM263x MCU+ SDK | ツール フォルダ |
| AM263x Academy | 学習用教材 |



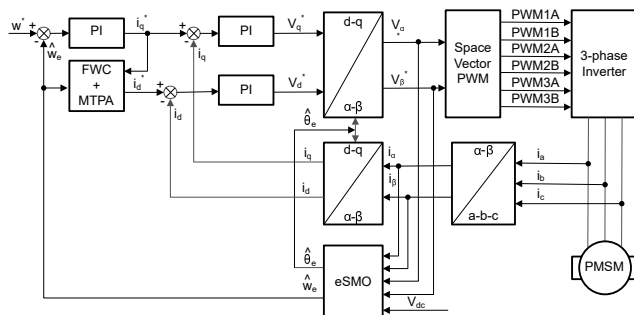
テキサス・インスツルメンツの[™] E2E サポート エキスパートにお問い合わせください。

特長

- このリファレンス デザインに関する包括的なソフトウェア パッケージ、ツール、ドキュメントは、AM263x MCU をベースとするモーター制御システムの開発期間の短縮できるようにします。
- さまざまな FOC モーター制御方式: センサレス (eSMO) およびセンサ付き (インクリメンタル エンコーダ、ホール センサ) 制御がサポートされています。
- 多くの 3 相インバータ モーター評価キットと互換性のあるテキサス・インスツルメンツのシステム機能とデバッグ インターフェイスが含まれています。
- Sysconfig ベースのプロジェクトでは、異なるデバイスやボード間でも簡単に移行できます。これは、ピン、ペリフェラル、ソフトウェア スタック、クロック ツリー、その他の要素を調整できる使いやすいグラフィカル ユーザー インターフェイスによって実現され、ソフトウェア開発がスピードアップします。

アプリケーション

- HEV/EV のインバータおよびモーター制御
- モーター ドライブ
- AC インバータと VF ドライブ
- AC ドライブ制御モジュール



1 システムの説明

このリファレンス デザインで説明するユニバーサル モーター制御プロジェクトは、さまざまなモーター制御アルゴリズムを試してみるだけでなく、独自の設計のリファレンスとして使用することを目的としています。この単一プロジェクトにはさまざまなビルド構成オプションがあり、テキサス・インスツルメンツのシステム機能やデバッグ インターフェイスとともに、センサレス (eSMO) およびセンサ付き (インクリメンタル エンコーダ、ホール センサ) を含むさまざまな FOC モーター制御方式を利用できます。このプロジェクトでは、データ ロギング、ソフトウェア周波数応答アナライザ (SFRA)、モーター PI 調整、弱め界磁、アンペアあたり最大トルク (MPTA)、CPU 時間計算、EPWM DAC モード、ステップ応答モジュール、位相調整などのさまざまな機能を提供しています。このリファレンス デザインでは、Ti.com から注文できる 2 つのハードウェア キットの評価を行っています。

- [BOOSTXL-3PHGANINV + LVSERVOMTR + LP-AM263](#)
- [TMDSHVMTRINSPIN + HVPMSMMTR + TMDSCNCD263](#)

1.1 用語

| | |
|--------------|----------------------|
| FOC | フィールド オリエンテッド コントロール |
| eSMO | 拡張スライディング モード オブザーバ |
| PMSM | 永久磁石同期型モーター |
| EEMF | 拡張起電力 |
| PLL | 位相ロック ループ |
| IPMS | 内部 PMSM |
| FW | 弱め界磁 |
| MTPA | 最大トルク / 電流 |
| SVPWM | 空間ベクトル変調 |
| PWM | パルス幅変調 |
| RPM | 回転数 / 分 |

1.2 主なシステム仕様

- シャント方式のインライン モーター位相電流センシング評価ボードを搭載した **48V 3 相インバータ**の詳細については、[BOOSTXL-3PHGANINV](#) のデザイン ファイルおよび技術資料を参照してください。
- 高電圧モーター制御キットの詳細については、[TMDSHVMTRINSPIN](#) を参照してください。
- 低電圧サーボ モーター、エンコーダ、ワイヤ ハーネスの詳細については、[LVSERVOMTR](#) のデータシートを参照してください。
- 高電圧永久磁石同期モーターの詳細については、[HVPMSMMTR](#) のデータシートを参照してください。

2 システム概要

2.1 ブロック図

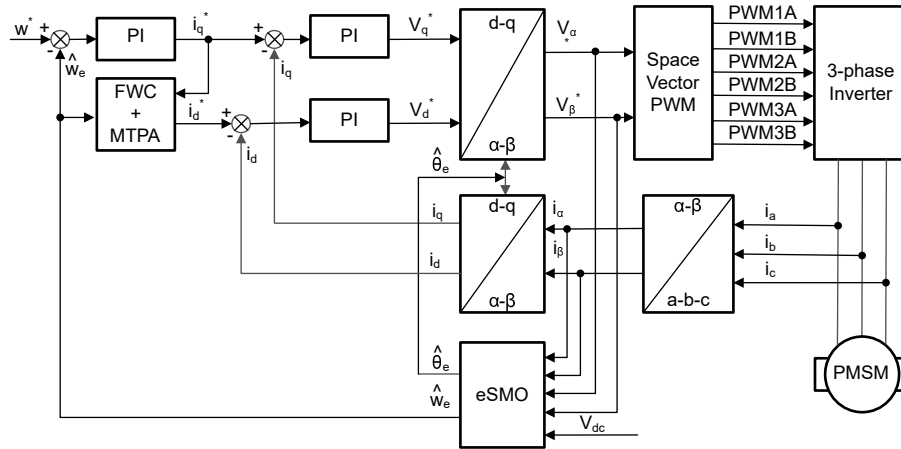


図 2-1. PMSM モーターの eSMO ベース センサレス FOC のブロック図

2.2 主な使用製品

2.2.1 AM263x マイクロコントローラ

AM263x Arm® ベースのマイクロコントローラは、次世代の産業用および車載用組み込み製品の複雑なリアルタイム処理ニーズを満たすように構成されています。AM263x MCU ファミリーは、最大 4 つの 400MHz Arm® Cortex®-R5F コアを内蔵した複数のピン互換デバイスで構成されています。複数の Arm® コアを任意にプログラミングして、各種の機能安全構成用にロックステップのオプションで動作させることもできます。産業用通信サブシステム (ICSS) を使用すると、PROFINET IRT、TSN、EtherCAT® など (その他多数) の内蔵の産業用イーサネット通信、または標準的なイーサネットコネクティビティやカスタム I/O インターフェイスを実現できます。AM263x ファミリーは、高度なアナログ モジュールを使用した高度モーター制御およびデジタル電源制御アプリケーション用に設計されています。

2.2.1.1 TMDSCNCD263

AM263x controlCARD 評価ボード (EVM) は、テキサス・インスツルメンツの Sitara™ AM263x シリーズのマイクロコントローラ (MCU) 用の評価 / 開発ボードです。この EVM には、プログラミングおよびデバッグ用のオンボード エミュレーションが搭載され、ボタンや LED を備えたシンプルなユーザー インターフェイスを利用できるため、AM263x MCU でのトラクション インバータの開発を簡単に始めることができます。また、この controlCARD を使用すると、ヘッダー ピンからキーにアクセスして、迅速にプロトタイプを製作できます。

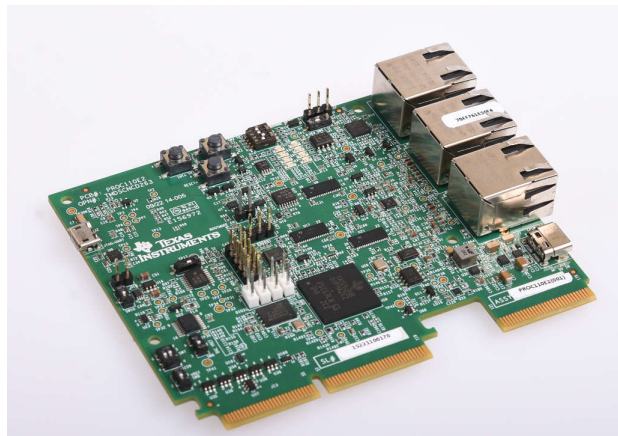


図 2-2. AM263x controlCARD™

2.2.1.2 LP-AM263

LP-AM263 は、Sitara™ 高性能マイクロコントローラ (MCU) である AM263x シリーズ向けのコスト最適化済み開発ボードです。このボードは、新しいアプリケーションを開発するための標準化された使いやすいプラットフォームを実現できるので、初期評価やプロトタイプ製作に最適です。

LP-AM263 は、Sitara AM2634 プロセッサと付加的なコンポーネントを採用しています。開発ユーザーは、産業用イーサネット (IE)、標準的なイーサネット、高速シリアル インターフェイス (FSI) などの各種デバイス インターフェイスを使用して、プロトタイプを容易に製作できます。AM2634 は、EtherCAT、EtherNet/IP、PROFINET® などのさまざまな IE (産業用イーサネット) プロトコルをサポートしています。

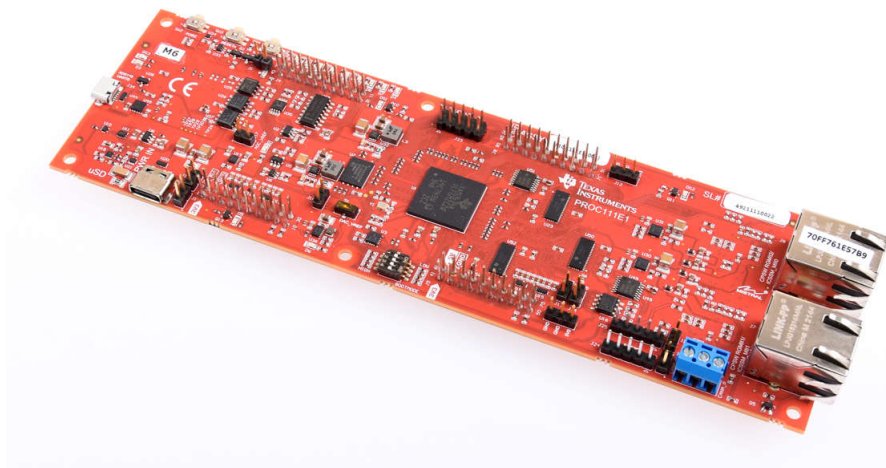


図 2-3. AM263x LaunchPad™

3 システム設計理論

3.1.3 相 PMSM 駆動

永久磁石同期モーター (PMSM) は、巻線固定子、永久磁石回転子アセンブリ、回転子位置をセンシングする内部デバイスや外部デバイスを備えています。センシング デバイスは、磁石アセンブリの回転を維持するために、固定子電圧リファレンスの周波数と振幅を適切に調整するための位置帰還を提供するものです。内部の永久磁石回転子と外部巻線を組み合わせることで、回転子の慣性が小さく、放熱が効率的で、モーターを小型化できるという利点があります。

- 同期モーターの構造: 永久磁石は回転軸にしっかりと固定され、一定の回転子フラックスを生み出します。この回転子フラックスの振幅は通常一定の大きさです。通电されると、固定子巻線によって回転磁場が生成されます。回転磁場を制御するには、固定子電流を制御する必要があります。
- 回転子の実際の構造は、機械の出力範囲と定格速度によって異なります。数キロワットまでの同期機には、永久磁石が最も適しています。高電力定格の場合、回転子は通常 DC 電流が循環する巻線で構成されます。回転子の機械的構造は、必要な極数と必要なフラックス勾配に応じて設計されます。
- 固定子フラックスと回転子フラックスの相互作用によってトルクが生成されます。固定子はフレームにしっかりと取り付けられ、回転子は自由に回転できるため、[図 3-1](#) に示すように、回転子が回転することにより実際の機械的出力が得られます。
- 最大トルクを生成して、高い電気機械的変換効率を達成するには、回転子磁場と固定子磁場の間の角度を慎重に制御する必要があります。このためには、センサレス アルゴリズムを使用して速度ループを閉じた後、速度とトルクが同じ条件のもとで最小量の電流を流すように微調整する必要があります。
- 回転する固定子磁場は、回転子の永久磁場と同じ周波数で回転する必要があります。そうでない場合、回転子には正と負のトルクが急速に交互に発生します。その結果、最適なトルクが得られなくなり、機械部品に過度の機械的な振動やノイズ、機械的ストレスが生じることとなります。さらに、回転子の慣性が原因で、回転子がこれらの振動に反応できなくなると、回転子は同期周波数での回転を停止し、静止している回転子から見た平均トルク、つまりゼロに反応することとなります。これは機械がブルアウトと呼ばれる現象を起こしていることを意味しており、同期機が自己起動しない理由でもあります。
- 最高の相互トルク生成を実現するには、回転子磁場と固定子磁場の間の角度が 90° でなければなりません。この同期化には、回転子の位置を把握して、適切な固定子の磁場を生成する必要があります。
- 固定子の磁場は、異なる固定子相からの影響を組み合わせることで固定子フラックスを生成することにより、任意の方向と大きにすることができます。

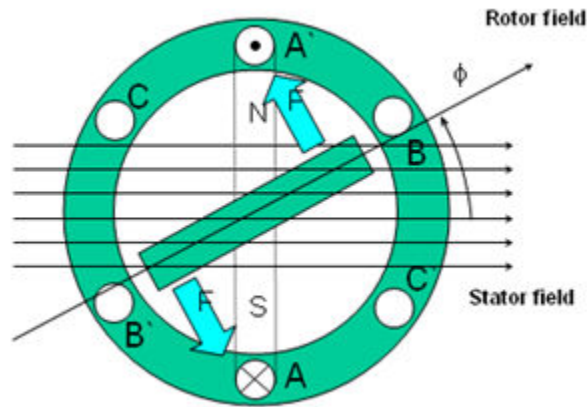


図 3-1. 回転する固定子フラックスと回転子フラックスの相互作用によって生成されるトルク

3.1.1 PMSM の数学モデルと FOC 構造

PMSM の FOC 構造を [図 2-1](#) に示します。このシステムでは、IPMSM システムのセンサレス制御を実現するために eSMO を使用し、eSMO モデルは、逆起電力モデルと PLL モデルを組合せて、回転子の位置と速度を推定するように設計されています。

IPMSM は、3 相の固定子巻線 (a 軸、b 軸、c 軸) と励起用の永久磁石 (PM) 回転子で構成されています。モーターは、標準的な 3 相インバータによって制御されます。IPMSM は 位相 a-b-c の量を用いてモデル化できます。適切な座標変

換を行うことで、PMSM の動的モデルを **d-q** 回転リファレンス フレームと **α-β** 固定リファレンス フレームで実現できます。これらのリファレンス フレームは **式 1** のような関係にあります。一般的な PMSM の動的モデルは、**d-q** 回転リファレンス フレームにおいて次のように記述できます。

$$\begin{bmatrix} v_d \\ v_q \end{bmatrix} = \begin{bmatrix} R_s + pL_d & -\omega_e L_q \\ \omega_e L_d & R_s + pL_q \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} 0 \\ \omega_e \lambda_{pm} \end{bmatrix} \quad (1)$$

ここで、

- V_d は、**q** 軸の固定子端子電圧です
- V_q は、**d** 軸の固定子端子電圧です
- i_d は、**d** 軸の固定子電流です
- i_q は、**q** 軸の固定子電流です
- L_d は、**q** 軸のインダクタンスです
- L_q は、**d** 軸のインダクタンスです
- p は、微分演算子です
- **式 2** の短い表記は、永久磁石によって生成される磁束結合です
- R_s は、固定子巻線の抵抗です
- ω_e は、回転子の電気角速度です

$$\frac{d}{dt} \lambda_{pm} \quad (2)$$

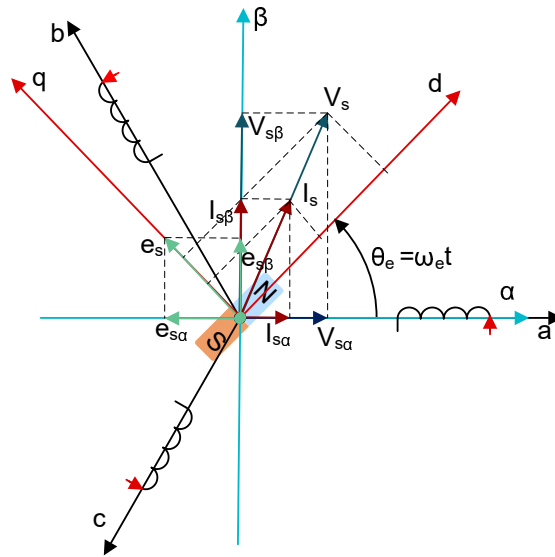


図 3-2. PMSM モデル化のための座標リファレンス フレームの定義

図 3-2 に示すように逆パーク変換を使用することで、PMSM の動的特性は、次に示すように **α-β** 固定リファレンス フレームでモデル化できます。

$$\begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} = \begin{bmatrix} R_s + pL_d & \omega_e(L_d - L_q) \\ -\omega_e(L_d - L_q) & R_s + pL_q \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \begin{bmatrix} e_\alpha \\ e_\beta \end{bmatrix} \quad (3)$$

e_α および e_β は、**α-β** 軸の拡張起電力 (EEMF) の成分であり、次に示すように定義できます。

$$\begin{bmatrix} e_\alpha \\ e_\beta \end{bmatrix} = (\lambda_{pm} + (L_d - L_q)i_d)\omega_e \begin{bmatrix} -\sin(\theta_e) \\ \cos(\theta_e) \end{bmatrix} \quad (4)$$

式 3 と式 4 によると、等価変換と EEMF の概念を導入することで、回転子の位置情報はインダクタンス マトリックスから切り離すことができ、その結果、EEMF が回転子の極位置情報を含む唯一の項となります。さらに、EEMF の位相情報をそのまま回転子の位置観測に利用することができます。固定子電流を状態変数として、IPMSM の電圧式 5 を状態式に書き換えます。

$$\begin{bmatrix} \dot{i}_\alpha \\ \dot{i}_\beta \end{bmatrix} = \frac{1}{L_d} \begin{bmatrix} -R_s & -\omega_e(L_d - L_q) \\ \omega_e(L_d - L_q) & -R_s \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \frac{1}{L_d} \begin{bmatrix} V_\alpha - e_\alpha \\ V_\beta - e_\beta \end{bmatrix} \quad (5)$$

直接測定できる唯一の物理量は固定子電流であるため、スライディング サーフェスは固定子電流の経路上で選択されま

$$S(x) = \begin{bmatrix} \hat{i}_\alpha - i_\alpha \\ \hat{i}_\beta - i_\beta \end{bmatrix} = \begin{bmatrix} \tilde{i}_\alpha \\ \tilde{i}_\beta \end{bmatrix} \quad (6)$$

ここで、

- 式 7 および 式 8 は、推定電流です。
- 上付きの添え字 $\hat{\cdot}$ は推定値を示します。
- 上付きの添え字 $\tilde{\cdot}$ は、観測値と実測値との差を意味する変数誤差を示します。

$$\hat{i}_\alpha \quad (7)$$

$$\hat{i}_\beta \quad (8)$$

3.1.2 PM 同期モーターの磁界方向制御

より優れた動的性能を実現するには、より複雑な制御方式を適用して PM モーターを制御する必要があります。マイクロコントローラがもたらす数学的処理能力により、PM モーターのトルク生成と磁化機能をデカップリングする数学的変換を使用した高度な制御方式を実装できます。トルクと磁化をデカップリングするこのような制御は、一般的に回転子フラックスオリエンテッドコントロール、または単にフィールドオリエンテッドコントロール (FOC) と呼ばれます。

図 3-3 に示すように、直流 (DC) モーターでは、固定子と回転子の励磁は別々に制御され、生成されたトルクとフラックスは別々の調整が可能です。界励磁の強さ (たとえば、界励磁電流の大きさ) によって、フラックスの値が設定されます。回転子巻線に流れる電流によって、生成されるトルクの大きさが決まります。トルク生成において特に興味深い役割を果たすのが、回転子の整流子です。整流子はブラシと接触しており、機械的構造上、最大トルクを生成するように機械的に整列された巻線が回路に切り替わるように設計されています。この配置によって、機械のトルク生成は常に最適にかなり近い状態になります。ここで重要なポイントは、回転子巻線によって生成されるフラックスが固定子磁場と直交するように巻線が制御されていることです。

より優れた動的性能を実現するには、より複雑な制御方式を適用して PM モーターを制御する必要があります。マイクロコントローラがもたらす数学的処理能力により、PM モーターのトルク生成と磁化機能をデカップリングする数学的変換を使用した高度な制御方式を実装できます。トルクと磁化をデカップリングするこのような制御は、一般的に回転子フラックスオリエンテッドコントロール、または単にフィールドオリエンテッドコントロール (FOC) と呼ばれます。

図 3-3 に示すように、直流 (DC) モーターでは、固定子と回転子の励磁は別々に制御され、生成されたトルクとフラックスは別々の調整が可能です。界励磁の強さ (たとえば、界励磁電流の大きさ) によって、フラックスの値が設定されます。回転子巻線に流れる電流によって、生成されるトルクの大きさが決まります。トルク生成において特に興味深い役割を果たすのが、回転子の整流子です。整流子はブラシと接触しており、機械的構造上、最大トルクを生成するように機械的に整列された巻線が回路に切り替わるように設計されています。この配置によって、機械のトルク生成は常に最適にかなり近い状態になります。ここで重要なポイントは、回転子巻線によって生成されるフラックスが固定子磁場と直交するように巻線が制御されていることです。

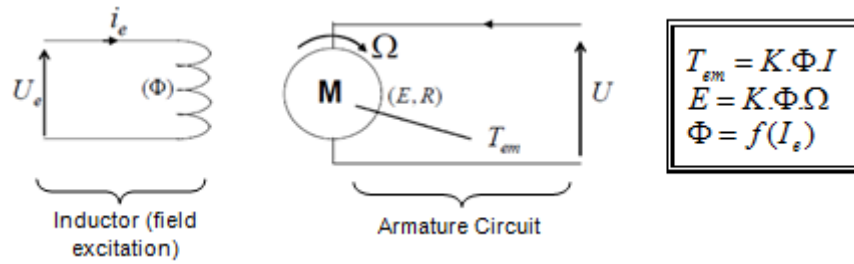


図 3-3. フラックスとトルクが別々に制御される DC モーター モデル

同期機と非同期機の FOC (ベクトル制御とも呼ぶ) の目的は、トルク生成成分とフラックス磁化成分を別々に制御できるようにすることです。FOC 制御を使用すると、固定子電流のトルク生成成分とフラックス磁化成分をデカップリングできます。磁化のデカップリング制御を行うことで、固定子フラックスのトルク生成成分は独立したトルク制御として考えることができるようになりました。トルクとフラックスをデカップリングするには、いくつかの数学的変換を行います。ここでマイクロコントローラが最も価値を発揮します。マイクロコントローラによる処理能力によって、このような数学的変換を非常に高速で行うことができます。これは、モーターを制御するアルゴリズム全体を高速で実行できることを意味し、より高度な動的性能が実現できるのです。現在では、デカップリングに加えて、回転子フラックスの角度や回転子の速度など、数多くの量の計算にモーターの動的モデルが使用されています。つまり、これらの効果が考慮されて、全体的な制御の質が向上しているのです。

電磁法則によれば、同期機で生成されるトルクは、式 9 に示すように、既存の 2 つの磁場のベクトル外積に等しくなります。

$$\tau_{em} = \vec{B}_{stator} \times \vec{B}_{rotor} \quad (9)$$

この式は、固定子と回転子の磁場が直交している場合、つまり負荷が 90 度である場合にトルクが最大になることを示しています。この状態を常に維持でき、フラックスを正しい方向に向けることができれば、トルクリップルを減らし、より優れた動的応答を維持することができます。ただし、回転子の位置を把握していなければならないという制約があります。インクリメンタル エンコーダのような位置センサを使用すると実現できます。回転子にアクセスできないような低コストのアプリケーションでは、位置センサを排除するために異なる回転子位置オブザーバ方式が適用されます。

簡単に説明すると、回転子フラックスと固定子フラックスを直交させた状態で維持することです。固定子フラックスを回転子フラックスの q 軸に、たとえば回転子フラックスに直交するように合わせることを目標です。そのために、回転子フラックスと直交する固定子電流成分は指令されたトルクを生成するように制御され、直接成分はゼロに設定されます。固定子電流の直接成分は場合によって弱め界磁に使用することができ、回転子フラックスを逆向きにし、逆 EMF を減少させるので、より高速での動作が可能になります。

フィールド オリエンテッド コントロールは、ベクトルで表される固定子電流を制御します。この制御は、時間と速度に依存する 3 相座標系が時不変の 2 座標系 (d 座標と q 座標) に変換される投影に基づいています。このような投影によって、DC 機制御と同じような構造になります。フィールド オリエンテッド コントロールに基づいている機械は、入力ファレンスとして、トルク成分 (q 座標) とフラックス成分 (d 座標) の 2 つの定数を必要とします。フィールド オリエンテッド コントロールは単純に投影に基づいているため、制御構造が瞬間的な電気量を取り扱います。これによって、あらゆる動作 (定常状態と過渡状態) について正確な制御が実現し、帯域幅に制限がある数学モデルに依存することがありません。これにより、FOC は従来の方式の問題を次のように解決します。

- 一定のリファレンスに到達しやすい (固定子電流のトルク成分とフラックス成分)
- (d, q) リファレンス フレームではトルクの式が 式 10 で定義されることから、直接トルク制御が適用しやすい

$$\tau_{em} \propto \psi_R \times i_{sq} \quad (10)$$

回転子フラックス (ψ_R) の振幅を固定値に保つことで、トルクとトルク成分 (i_{sq}) の間に線形関係が得られます。固定子電流ベクトルのトルク成分を制御することで、トルクを制御できます。

3.1.2.1 (a, b) → (α, β) クラーク変換

空間ベクトルは、直交する 2 つの軸 (α, β) だけを持つ別のリファレンス フレームで表すことができます。a 軸と alpha 軸が同じ方向であると仮定すると、図 3-4 のようなベクトル図になります。

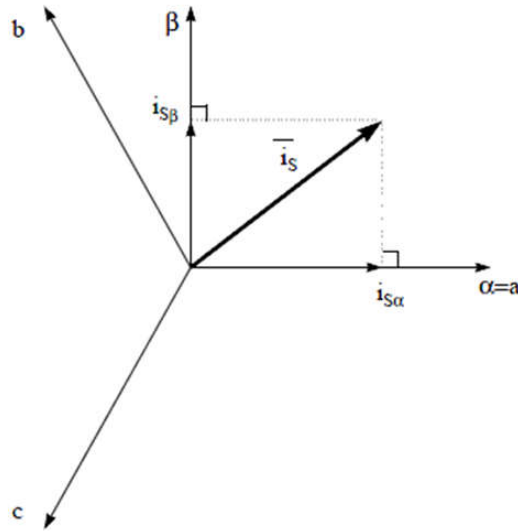


図 3-4. 固定リファレンス フレームの固定子電流空間ベクトル

3 相座標系を (α, β) 2 次元直交系に変更する投影を式 11 に示します。

$$\begin{aligned} i_{s\alpha} &= i_a \\ i_{s\beta} &= \frac{1}{\sqrt{3}}i_a + \frac{2}{\sqrt{3}}i_b \end{aligned} \quad (11)$$

2 相 (α, β) 電流は、依然として時間と速度に依存します。

3.1.2.2 (α, β) → (d, q) パーク変換

これは FOC における最も重要な変換です。実際には、この投影は (d, q) 回転リファレンス フレームの 2 相直交座標系 (α, β) を変更するものです。d 軸が回転子フラックスと一直線にあるものとして、図 3-5 に 2 つのリファレンス フレームの電流ベクトルの関係を示しています。

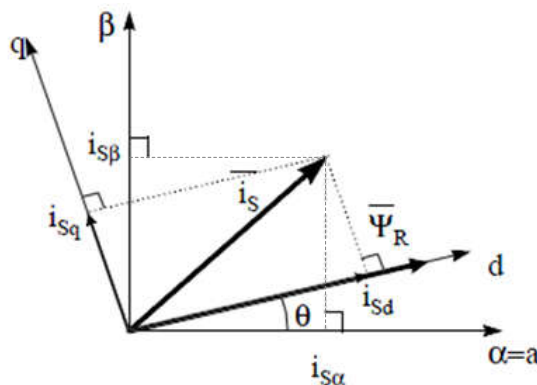


図 3-5. (d, q) 回転リファレンス フレームの固定子電流空間ベクトル

電流ベクトルのフラックス成分とトルク成分は式 12 で決定されます。

$$\begin{aligned} i_{sd} &= i_{s\alpha}\cos(\theta) + i_{s\beta}\sin(\theta) \\ i_{sq} &= -i_{s\alpha}\sin(\theta) + i_{s\beta}\cos(\theta) \end{aligned} \quad (12)$$

θ は回転子フラックスの位置です。

これらの成分は、電流ベクトル (α, β) の成分と回転子フラックスの位置に依存します。適切な回転子フラックスの位置がわかると、この投影によって d, q 成分は一定になります。これで 2 相電流は DC 量 (時不変) に変わります。この時点で、一定の i_{sd} (フラックス成分) と i_{sq} (トルク成分) の電流成分が別々に制御されるため、トルク制御が容易になります。

3.1.2.3 AC モーターの FOC 基本方式

図 3-6 に、FOC によるトルク制御の基本方式をまとめます。

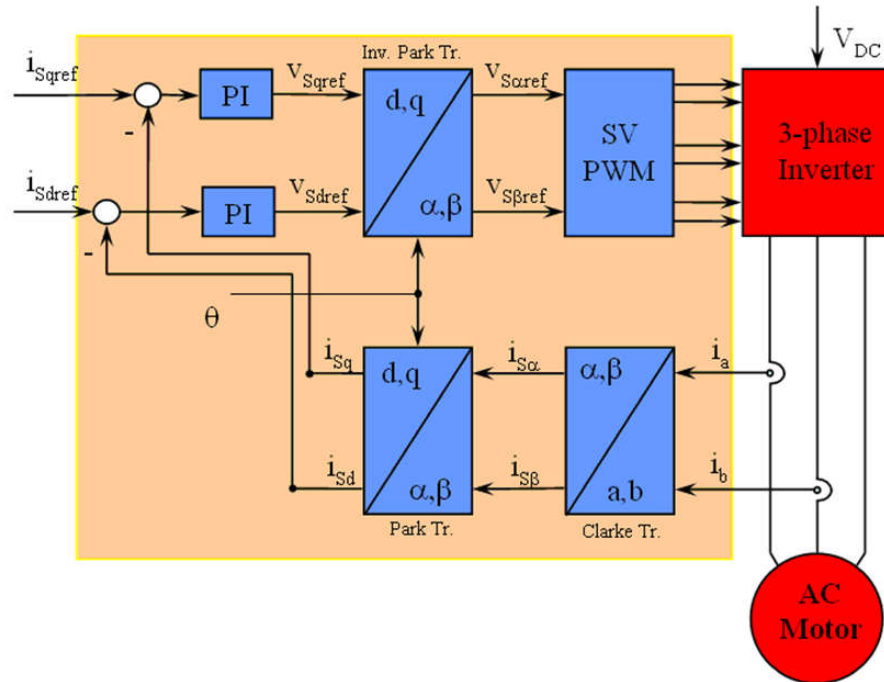


図 3-6. AC モーターの FOC 基本方式

2 つのモーター相電流が測定されて、測定値がクラーク変換モジュールに供給されます。この投影の出力は i_{sa} と i_{sb} となります。この電流の 2 つの成分は、 d, q 回転リファレンス フレームでの電流をもたらすパーク変換の入力です。 i_{sd} と i_{sq} 成分は、リファレンス i_{sdref} (フラックスリファレンス成分) と i_{sqref} (トルクリファレンス成分) と比較されます。ここで、この制御構造に興味深い利点があることがわかります。つまり、フラックスリファレンスを変更して、回転子フラックスの位置を取得するだけで、同期機と誘導機のどちらを制御するにもこの制御構造を使用できるということです。永久磁石同期モーターの場合、回転子フラックスは磁石によって固定されているため、フラックスの生成は必要ありません。したがって、PMSM を制御する場合、 i_{sdref} はゼロに設定されます。AC 誘導モーターは動作するために回転子フラックスを生成する必要があるため、フラックスリファレンスはゼロであってはなりません。これにより、従来の制御構造の大きな欠点の 1 つである、非同期ドライブから同期ドライブへの移行が簡単に解決されます。速度 FOC を使用する場合、トルク指令 i_{sqref} を速度レギュレータの出力とすることができます。電流レギュレータの出力は V_{sdref} と V_{sqref} であり、逆パーク変換に適用されます。この投影の出力は、 (α, β) 固定直交リファレンス フレームにおける固定子ベクトル電圧の成分である V_{saref} と V_{sbref} であり、空間ベクトル PWM の入力になります。このブロックの出力はインバータを駆動する信号です。パーク変換と逆パーク変換の両方には回転子フラックスの位置が必要になることに注意してください。この回転子フラックスの位置の取得方法は、AC 機のタイプ (同期機または非同期機) によって異なります。

3.1.2.4 回転子フラックスの位置

FOC では、回転子フラックスの位置情報を把握することが中心となります。実際、この変数に誤差があると、回転子フラックスは d -軸と一直線にならず、 i_{sd} と i_{sq} は固定子電流の正しいフラックス成分とトルク成分とはなりません。図 3-7 は (a, b, c)、 (α, β) 、 (d, q) の各リファレンス フレームを示し、同期速度で d, q リファレンスで回転する、回転子フラックス、固定子電流、固定子電圧の各空間ベクトルの正しい位置を示しています。

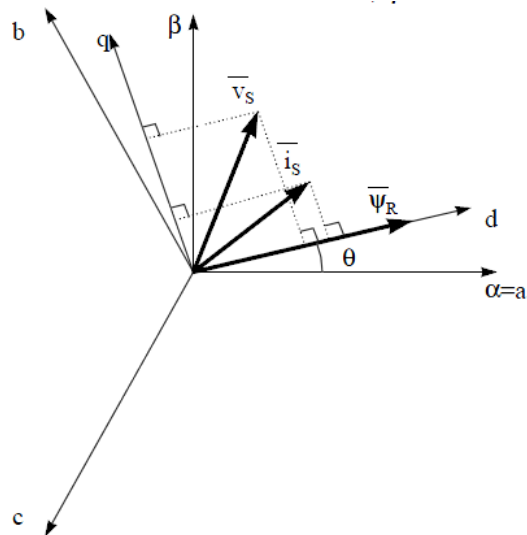


図 3-7. 回転リファレンス フレーム (d, q) の電流、電圧、回転子フラックスの各空間ベクトル

同期モーターと非同期モーターでは、回転子フラックス位置の測定方法が異なります。

- 同期モーターでは、回転子速度は回転子フラックス速度と等しくなります。したがって、 θ (回転子フラックスの位置) は位置センサによって直接測定されるか、回転子速度の積分によって求められます。
- 非同期モーターでは、回転子速度は回転子フラックス速度と等しくないため (スリップ速度があるため)、 θ の算出には特定の方法が必要になります。基本的な方法としては、d, q リファレンス フレームにおけるモーター モデルの 2 つの式を必要とする電流モデルを使用します。

理論的には、PMSM ドライブのフィールド オリエンテッド コントロールにより、DC モーターの動作のようにモーター トルクをフラックスとは無関係に制御することができます。言い換えれば、トルクとフラックスは互いにデカップリングされていることとなります。固定リファレンス フレームから同期回転リファレンス フレームへの変数変換を行うには、回転子位置を知る必要があります。この変換 (いわゆるパーク変換) の結果、q-軸の電流がトルクを制御し、d-軸の電流は強制的にゼロになります。したがって、このシステムの重要なモジュールは、拡張スライディング モード オブザーバ (eSMO) を使用した回転子位置の推定になります。

3.1.3 PM 同期モーターのセンサレス制御

家電アプリケーションでは、機械センサを使用することにより、コスト、サイズ、信頼性の問題が増します。これらの問題を克服するために、センサレス制御方式が導入されています。機械的な位置センサを使用せずに回転子の速度と位置の情報を取得するために、複数の推定方法が使用されます。スライディング モード オブザーバ (SMO) には、信頼性、期待される性能、システム パラメータの変動に対する堅牢性など、数多くの魅力的な特長があるため、一般的に利用されています。

3.1.3.1 位相ロック ループを備えた拡張スライディング モード オブザーバ

モデル ベースの手法を使用して、モーターが中速または高速で動作する場合に、IPMSM ドライブ システムの位置センサレス制御を実現しています。モデル ベースの手法では、逆 EMF モデルまたは磁束結合モデルによって回転子位置を推定します。スライディング モード オブザーバは、スライディング モード制御に基づいたオブザーバの設計手法です。システムの構造は固定ではなく、システムの現在の状態に応じて意図的に変更され、あらかじめ決められたスライディング モードの軌道に従って強制的に動かされます。応答が速く、堅牢性が優れ、パラメータの変化や外部の変動に対して影響を受けにくいのが利点です。

3.1.3.1.1 PMSM 向け ESMO の設計

図 3-8 に、SMO に組み込まれた通常の PLL を示します。

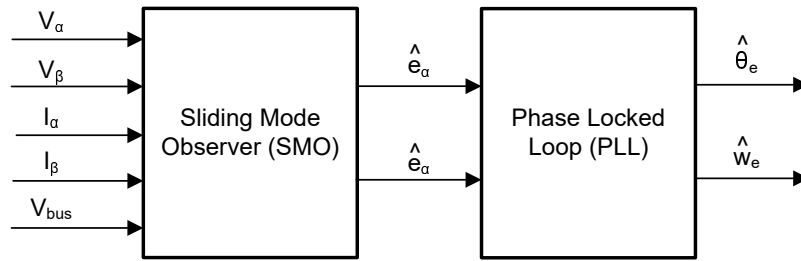


図 3-8. PMSM 向け PLL 搭載 eSMO のブロック図

従来の低次スライディングモードオブザーバは、式 13 に示す数学モデルと、図 3-9 に示すブロック図で構築されます。

$$\begin{bmatrix} \dot{\hat{i}}_{\alpha} \\ \dot{\hat{i}}_{\beta} \end{bmatrix} = \frac{1}{L_d} \begin{bmatrix} -R_s & -\hat{\omega}_e(L_d - L_q) \\ \hat{\omega}_e(L_d - L_q) & -R_s \end{bmatrix} \begin{bmatrix} \hat{i}_{\alpha} \\ \hat{i}_{\beta} \end{bmatrix} + \frac{1}{L_d} \begin{bmatrix} V_{\alpha} - \hat{e}_{\alpha} + z_{\alpha} \\ V_{\beta} - \hat{e}_{\beta} + z_{\beta} \end{bmatrix} \quad (13)$$

ここで、 z_{α} および z_{β} はスライディングモードの帰還成分であり、次のように定義されます。

$$\begin{bmatrix} z_{\alpha} \\ z_{\beta} \end{bmatrix} = \begin{bmatrix} k_{\alpha} \text{sign}(\hat{i}_{\alpha} - i_{\alpha}) \\ k_{\beta} \text{sign}(\hat{i}_{\beta} - i_{\beta}) \end{bmatrix} \quad (14)$$

ここで、 k_{α} および k_{β} はリアプノフ安定性解析によって設計された一定スライディングモードゲインです。 k_{α} と k_{β} が正であり、SMO の安定動作を維持するのに十分な大きさであるなら、 k_{α} と k_{β} は通常、式 15 と式 16 を保持するのに十分な大きさになります。

$$k_{\alpha} > \max(|e_{\alpha}|) \quad (15)$$

$$k_{\beta} > \max(|e_{\beta}|) \quad (16)$$

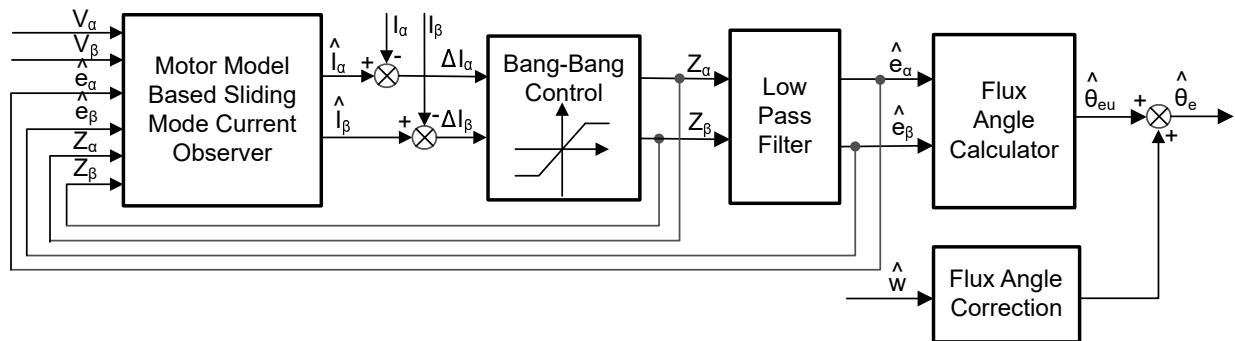


図 3-9. 従来のスライディングモードオブザーバのブロック図

α - β 軸における EEMF の推定値 (式 18、式 19) は、不連続スイッチング信号 z_{α} および z_{β} からのローパスフィルタによって得られます。

$$\begin{bmatrix} \hat{e}_{\alpha} \\ \hat{e}_{\beta} \end{bmatrix} = \frac{\omega_c}{s + \omega_c} \begin{bmatrix} z_{\alpha} \\ z_{\beta} \end{bmatrix} \quad (17)$$

$$\hat{e}_{\alpha} \quad (18)$$

$$\hat{e}_{\beta} \quad (19)$$

式 20 は LPF のカットオフ角周波数で、通常は固定子電流の基本周波数に応じて選択されます。

$$\omega_c = 2\pi f_c \quad (20)$$

したがって、回転子位置は、次に定義するように、逆起電力のアークタンジェントから直接計算できます。

$$\hat{\theta}_e = -\tan^{-1}\left(\frac{\hat{e}_\alpha}{\hat{e}_\beta}\right) \quad (21)$$

ローパスフィルタは、スライディングモード関数の高周波項を除去することにより、位相遅延を引き起こします。これは、カットオフ周波数 ω_c と逆起電力周波数 ω_e の関係を使用して補償できます。これは、次のように定義されます。

$$\Delta\theta_e = -\tan^{-1}\left(\frac{\omega_e}{\omega_c}\right) \quad (22)$$

次に、SMO 方式によって推定された回転子位置は、次で求められます。

$$\hat{\theta}_e = -\tan^{-1}\left(\frac{\hat{e}_\alpha}{\hat{e}_\beta}\right) + \Delta\theta_e \quad (23)$$

デジタル制御アプリケーションでは、SMO の時間分散式が必要です。時間分散オブザーバに変換するには、オイラー法が適切です。 α - β 座標における式 13 の時間分散システムマトリックスは、式 24 によって次のように求められます。

$$\begin{bmatrix} \hat{i}_\alpha(n+1) \\ \hat{i}_\beta(n+1) \end{bmatrix} = \begin{bmatrix} F_\alpha \\ F_\beta \end{bmatrix} \begin{bmatrix} \hat{i}_\alpha(n) \\ \hat{i}_\beta(n) \end{bmatrix} + \begin{bmatrix} G_\alpha \\ G_\beta \end{bmatrix} \begin{bmatrix} V_\alpha^*(n) - \hat{e}_\alpha(n) + z_\alpha(n) \\ V_\beta^*(n) - \hat{e}_\beta(n) + z_\beta(n) \end{bmatrix} \quad (24)$$

ここで、行列 $[F]$ と $[G]$ は式 25 と式 26 によって次のように求められます。

$$\begin{bmatrix} F_\alpha \\ F_\beta \end{bmatrix} = \begin{bmatrix} e^{-\frac{R_s}{L_d}} \\ e^{-\frac{R_s}{L_q}} \end{bmatrix} \quad (25)$$

$$\begin{bmatrix} G_\alpha \\ G_\beta \end{bmatrix} = \frac{1}{R_s} \begin{bmatrix} 1 - e^{-\frac{R_s}{L_d}} \\ 1 - e^{-\frac{R_s}{L_q}} \end{bmatrix} \quad (26)$$

式 17 の時間分散形式は、式 27 によって次のように求められます。

$$\begin{bmatrix} \hat{e}_\alpha(n+1) \\ \hat{e}_\beta(n+1) \end{bmatrix} = \begin{bmatrix} \hat{e}_\alpha(n) \\ \hat{e}_\beta(n) \end{bmatrix} + 2\pi f_c \begin{bmatrix} z_\alpha(n) - \hat{e}_\alpha(n) \\ z_\beta(n) - \hat{e}_\beta(n) \end{bmatrix} \quad (27)$$

3.1.3.1.2 PLL による回転子位置および速度の推定

アークタンジェント法では、ノイズや高調波の成分が存在するため、位置と速度の推定精度に影響が出ます。この問題を解消するために、IPMSM のセンサレス制御構造では、PLL モデルを速度と位置の推定に使用できます。セクション 3.1.3.1.1 に、SMO と一緒に使用される PLL 構造を示します。逆起電力の推定値式 18 および式 19 を PLL モデルで使用して、図 3-10 に示すように、モーターの角速度と位置を推定できます。

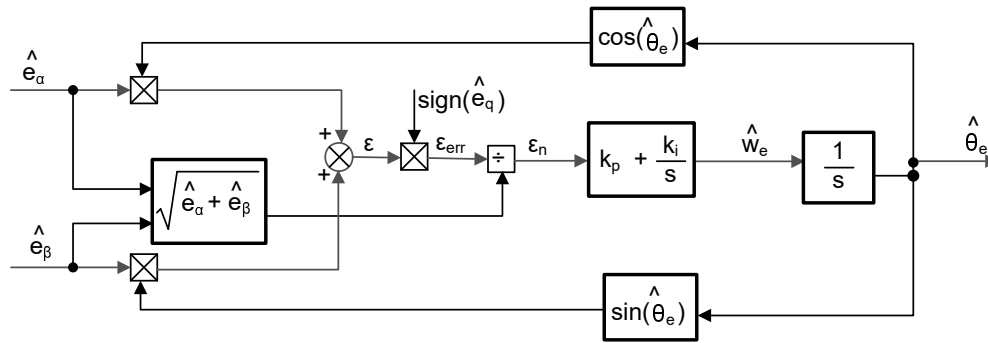


図 3-10. 位相ロック ループ位置トラッカーのブロック図

式 28、式 29、および 式 30 から、

$$e_\alpha = E \cos(\theta_e) \quad (28)$$

$$e_\beta = E \sin(\theta_e) \quad (29)$$

$$E = \omega_e \lambda_{pm} \quad (30)$$

位置誤差は次のように定義できます。

$$\varepsilon = \hat{e}_\beta \cos(\hat{\theta}_e) - \hat{e}_\alpha \sin(\hat{\theta}_e) = E \sin(\theta_e) \cos(\hat{\theta}_e) - E \cos(\theta_e) \sin(\hat{\theta}_e) = E \sin(\theta_e - \hat{\theta}_e) \quad (31)$$

E は EEMF の大きさを、モーター速度 ω_e に比例します。式 32 の場合、式 31 は次のように簡略化できます。

$$(\theta_e - \hat{\theta}_e) < \frac{\pi}{2} \quad (32)$$

$$\varepsilon = E(\theta_e - \hat{\theta}_e) \quad (33)$$

さらに、EEMF の正規化後の位置誤差を求めることができます。

$$\varepsilon_n = \theta_e - \hat{\theta}_e \quad (34)$$

解析に従うと、直角位相ロック ループの位置トラッカーの概略ブロック図は、図 3-11 のようになります。PLL の閉ループ伝達関数は、次のように表すことができます。

$$\frac{\hat{\theta}_e}{\theta_e} = \frac{k_p s + k_i}{s^2 + k_p s + k_i} = \frac{2\xi\omega_n s + \omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} \quad (35)$$

ここで、 k_p と k_i は標準的な PI レギュレーターの比例利得と積分利得であり、固有周波数 ω_n と減衰比 ξ は次のように与えられます

$$k_p = 2\xi\omega_n, \quad k_i = \omega_n^2 \quad (36)$$

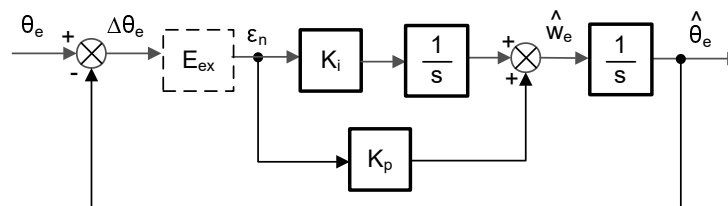


図 3-11. 位相ロック ループ位置トラッカーの概略ブロック図

3.1.4 モーター駆動のハードウェア要件

モーターの制御アルゴリズムは、DC バス電源電圧、各モーター相の電流など、モーターの状態に関するサンプリング測定値を利用します。モーターを正しく識別し、フィールド オリエンテッド コントロール (FOC) を使用してモーターを効果的に動作させるには、電流スケール値、電圧スケール値、電圧フィルタ極など、正しく設定する必要があるハードウェア依存のパラメータがいくつかあります。

3.1.5 その他の制御機能

3.1.5.1 弱め界磁 (FW) および最大トルク / 電流 (MTPA) 制御

永久磁石同期モーター (PMSM) は、高電力密度、高効率、幅広い速度範囲により、家電アプリケーションで広く使用されています。PMSM には、表面実装型 PMSM (SPM) と内部実装型 PMSM (IPM) の 2 つの主要なタイプがあります。SPM モーターは、トルクと q-軸電流が線形関係にあるため、制御が容易になっています。一方、IPMSM には、大きな突極性比による電磁トルクとリラクタンストルクがあります。総トルクは、回転子角度に対して非線形です。その結果、IPM モーターで MTPA 技術を使用して、定トルク領域でのトルク生成を最適化することができます。弱め界磁制御は、PMSM ドライブの電力と効率を最大限に高めるために最適化することが目的です。弱め界磁制御は、基本速度以上のモーター動作を可能にし、動作限界を拡大して定格速度を上回る速度に到達させ、速度と電圧の全範囲にわたって最適な制御ができるようになります。

IPMSM の数学モデルの電圧式は、式 37 と式 38 に示すように、d-q 座標で記述できます。

$$v_d = L_d \frac{di_d}{dt} + R_s i_d - p \omega_m L_q i_q \quad (37)$$

$$v_q = L_q \frac{di_q}{dt} + R_s i_q + p \omega_m L_d i_d + p \omega_m \psi_m \quad (38)$$

IPM 同期モーターの動的な等価回路を、図 3-12 に示します。

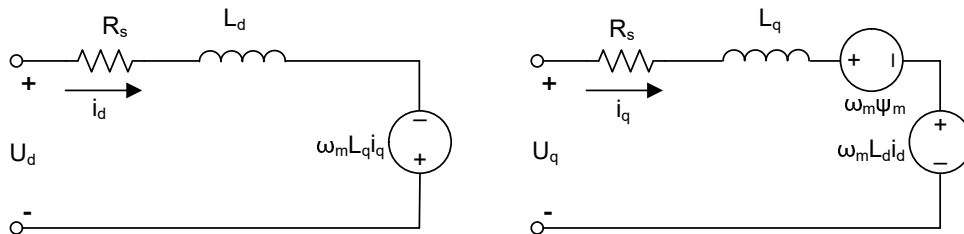


図 3-12. IPM 同期モーターの等価回路

IPMSM によって生成される総電磁トルクは式 40 によって表すことができ、生成されるトルクは 2 つの異なる項で構成されます。最初の項は、トルク電流 i_q と永久磁石の間で発生する相互応答トルクに対応します。

$$\psi_m \quad (39)$$

一方、2 番目の項は d 軸と q 軸のインダクタンスの違いによるリラクタンストルクに対応します。

$$T_e = \frac{3}{2} p [\psi_m i_q + (L_d - L_q) i_d i_q] \quad (40)$$

ほとんどのアプリケーションでは IPMSM ドライブに速度とトルクの制約があり、これは主にインバータまたはモーターの定格電流と、使用可能な DC リンク電圧の制限によるものです。これらの制約は、数式 式 41 と 式 42 で表すことができます。

$$I_a = \sqrt{i_d^2 + i_q^2} \leq I_{\max} \quad (41)$$

$$V_a = \sqrt{v_d^2 + v_q^2} \leq V_{\max} \quad (42)$$

ここで、 V_{\max} および I_{\max} は、インバータまたはモーターの最大許容電圧および電流です。2 レベル 3 相電圧源インバータ (VSI) によって駆動される機械では、達成可能な最大位相電圧は DC リンク電圧と PWM 方式によって制限されます。空間ベクトル変調 (SVPWM) を採用する場合、最大電圧は 式 43 に示す値に制限されます。

$$\sqrt{v_d^2 + v_q^2} \leq v_{\max} = \frac{v_{dc}}{\sqrt{3}} \quad (43)$$

固定子抵抗 R_s は通常、高速動作時は無視できる程度で、定常状態では電流の微分はゼロであるため、式 44 は以下のようになります。

$$\sqrt{L_d^2 \left(i_d + \frac{\psi_{pm}}{L_d} \right)^2 + L_q^2 i_q^2} \leq \frac{V_{\max}}{\omega_m} \quad (44)$$

式 41 の電流制限により、d-q 平面で半径 I_{\max} の円が生成され、加速すると 式 43 の電圧制限によって半径 V_{\max} が減少する楕円が生成されます。結果として得られる d-q 平面の電流ベクトルは、電流と電圧の制約に同時に従うように制御されなければなりません。これらの制約に従って、IPMSM の動作領域は、図 3-13 に示すように 3 つに分けられます。

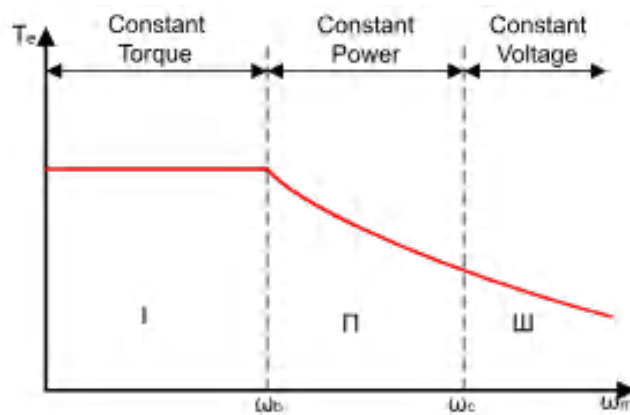


図 3-13. IPMSM 制御の動作領域

1. 定トルク領域: この動作領域では、MTPA を実装して最大トルクの生成を維持することができます。
2. 定電力領域: 弱め界磁制御の適用が必要であり、トルク容量は電流制約に達すると低下します。
3. 定電圧領域: この動作領域では、深い弱め界磁制御により固定子電圧が一定に保たれ、トルク生成が最大になります。

定トルク領域では、式 40 に基づき、IPMSM の総トルクには、磁束結合による電磁トルクと、 L_d および L_q 間の突極性からのリラクタンストルクが含まれます。電磁トルクは q 軸の電流 i_q に比例し、リラクタンストルクは d 軸の電流 i_d 、q 軸の電流 i_q 、および L_d と L_q の差の乗算に比例します。

SPM モーターの通常のベクトル制御システムでは、非弱め界磁モードで指令された i_d をゼロに設定することにより、電磁トルクのみを利用します。ただし、IPMSM はモーターのリラクタンストルクを利用するため、d-軸電流も制御する必要があります。MTPA 制御の目的は、リファレンス電流 i_d と i_q を計算し、生成される電磁トルクとリラクタンストルク間の比を最大化することです。 i_d と i_q の関係と固定子電流 I_s のベクトル和を以下の式に示します。

$$I_s = \sqrt{i_d^2 + i_q^2} \quad (45)$$

$$I_d = I_s \cos \beta \quad (46)$$

$$I_q = I_s \sin \beta \quad (47)$$

β は同期 (d-q) リファレンス フレームにおける固定子の電流角度です。式 40 は 式 48 と表すことができます。 I_s は i_d と i_q に代入します。

式 48 は、モーター トルクが固定子電流ベクトルの角度に依存することを示しています。

$$T_e = \frac{3}{2} p I_s \sin \beta \left[\psi_m + (L_d - L_q) I_s \cos \beta \right] \quad (48)$$

最大効率点は、モーターのトルク差がゼロのときに計算できます。MTPA 点は、この差分

$$\frac{dT_e}{d\beta} \quad (49)$$

が、式 50 で示されるように、ゼロのときに見つけることができます。

$$\frac{dT_e}{d\beta} = \frac{3}{2} p \left[\psi_m I_s \cos \beta + (L_d - L_q) I_s^2 \cos 2\beta \right] = 0 \quad (50)$$

この式に従うと、MTPA 制御の電流角度は、式 51 のように導くことができます。

$$\beta_{\text{mtpa}} = \cos^{-1} \frac{-\psi_m + \sqrt{\psi_m^2 + 8 \times (L_d - L_q)^2 \times I_s^2}}{4 \times (L_d - L_q) \times I_s} \quad (51)$$

したがって、実際の d 軸と q 軸のリファレンス電流は、MTPA 制御の電流角度を用いて、式 52 と式 53 で表すことができます。

$$I_d = I_s \times \cos \beta_{\text{mtpa}} \quad (52)$$

$$I_q = I_s \times \sin \beta_{\text{mtpa}} \quad (53)$$

ただし、式 51 に示すように、MTPA 制御の角度 β_{mtpa} は、d 軸と q 軸のインダクタンスに関係します。つまり、変動するインダクタンスの影響によって、最適な MTPA 点を見つけ出すことができなくなるということです。モーター駆動の効率を高めるには、d 軸と q 軸のインダクタンスをオンラインで推定する必要がありますが、 L_d と L_q パラメータはオンラインでは簡単に測定されず、飽和効果の影響を受けます。堅牢なルックアップ テーブル (LUT) 方式により、電気的パラメータが変動しても制御可能な状態を維持します。通常、数学モデルの簡略化のために、d-軸と q-軸のインダクタンス間のカップリング効果は無視することができます。したがって、 L_d は i_d のみに応じて変化し、 L_q は i_q のみに応じて変化すると仮定しています。その結果、d 軸および q 軸のインダクタンスは、式 54 と式 55 に示すように、それぞれ d-q 電流の関数としてモデル化できます。

$$L_d = f_1(i_d, i_q) = f_1(i_d) \quad (54)$$

$$L_q = f_2(i_q, i_d) = f_2(i_q) \quad (55)$$

式 51 を簡略化することで、ISR の計算負担を軽減しています。モーター パラメータに基づく定数 K_{mtpa} は、式 56 の代わりとして表されます。ここで、 K_{mtpa} は、更新された L_d および L_q を使用して、バックグラウンド ループで計算されます。

$$K_{\text{mtpa}} = \frac{\psi_m}{4 \times (L_q - L_d)} = 0.25 \times \frac{\psi_m}{(L_q - L_d)} \quad (56)$$

$$\beta_{\text{mtpa}} = \cos^{-1} \left(K_{\text{mtpa}} \div I_s - \sqrt{(K_{\text{mtpa}} \div I_s)^2 + 0.5} \right) \quad (57)$$

2 番目の中間変数 G_{mtpa} (式 58 に記載) が、計算をさらに簡単にするために定義されます。 G_{mtpa} (MTPA 制御の角度) を使用すると、 β_{mtpa} は式 59 のように計算できます。これら 2 つの計算を ISR で行い、実際の電流角度 β_{mtpa} を達成します。

$$G_{\text{mtpa}} = K_{\text{mtpa}} \div I_s \quad (58)$$

$$\beta_{\text{mtpa}} = \cos^{-1} \left(G_{\text{mtpa}} - \sqrt{G_{\text{mtpa}}^2 + 0.5} \right) \quad (59)$$

いずれの場合も、直軸の電流 i_d に作用することで磁束を弱め、実現可能な速度範囲を拡大することができます。この定電力動作領域に入ったことにより、定電力領域と定電圧領域で使用される MTPA 制御の代わりに、弱め界磁制御が選択されます。インバータの最大電圧が制限されるため、永久磁石の磁場とモーター速度にほぼ比例する逆起電力がインバータの最大出力電圧を上回るような速度領域では、PMSM モーターは動作できません。PM モーターでは、磁束を直接制御することはできません。ただし、 d 軸電機子反作用による減磁効果により、負の i_d を加えることでエアギャップフラックスを弱めることができます。電圧と電流の制約を考慮すると、電機子電流と端子電圧は式 41 と式 42 のように制限されます。インバータの入力電圧 (DC リンク電圧) の変動により、モーターの最大出力が制限されます。さらに、モーターの最大基本電圧も使用する PWM 方式によって異なります。式 44 では、IPMSM には 2 つの要素があります。1 つは永久磁石の値で、もう 1 つはインダクタンスとフラックスの電流によって作られています。

図 3-14 に、弱め界磁を実装するために使用される代表的な制御構造を示します。

β_{fw} は弱め界磁 (FW) PI コントローラの出力で、基準 i_d と i_q を生成します。電圧振幅が限界に達する前は、FW の PI コントローラの入力は常に正であるため、出力は常に 0 で飽和しています。

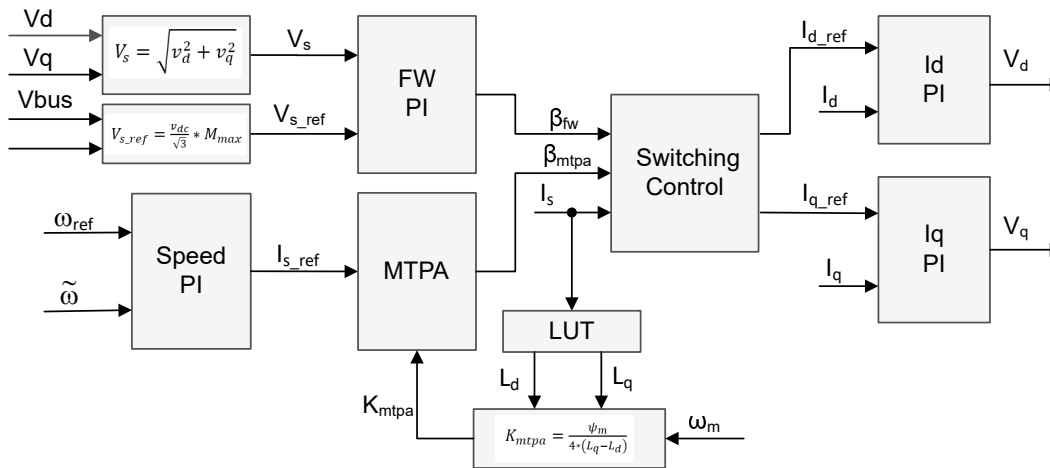


図 3-14. 弱め界磁と最大トルク / 電流制御のブロック図

モーター駆動 FOC システムには、MTPA 制御と弱め界磁制御の 2 つの制御モジュールがあります。これら 2 つのモジュールは、図 3-15 に示す入力パラメータに基づいてそれぞれ電流角度 β_{mtpa} と β_{fw} を生成します。

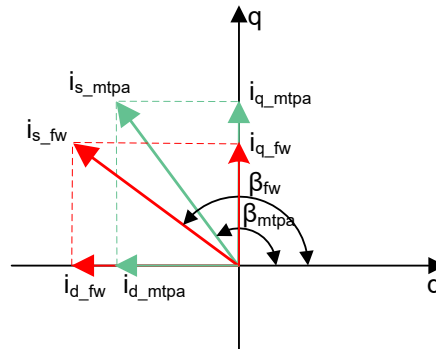


図 3-15. FW および MTPA 時の IPMSM の電流位相図

スイッチング制御モジュールは、適用できる角度を決定してから、式 46 と式 47 に示すように基準 i_d および i_q を計算します。電流角度は、式 60 と式 61 のように選択されます。

$$\beta = \beta_{fw} \text{ if } \beta_{fw} > \beta_{mtpa} \quad (60)$$

$$\beta = \beta_{mtpa} \text{ if } \beta_{fw} < \beta_{mtpa} \quad (61)$$

図 3-16 に、このリファレンス デザインにおける、弱め界磁制御 (FWC) と最大トルク / 電流 (MTPA) を備えた eSMO を使用した、PMSM のセンサレス FOC の全体ブロック図を示します。

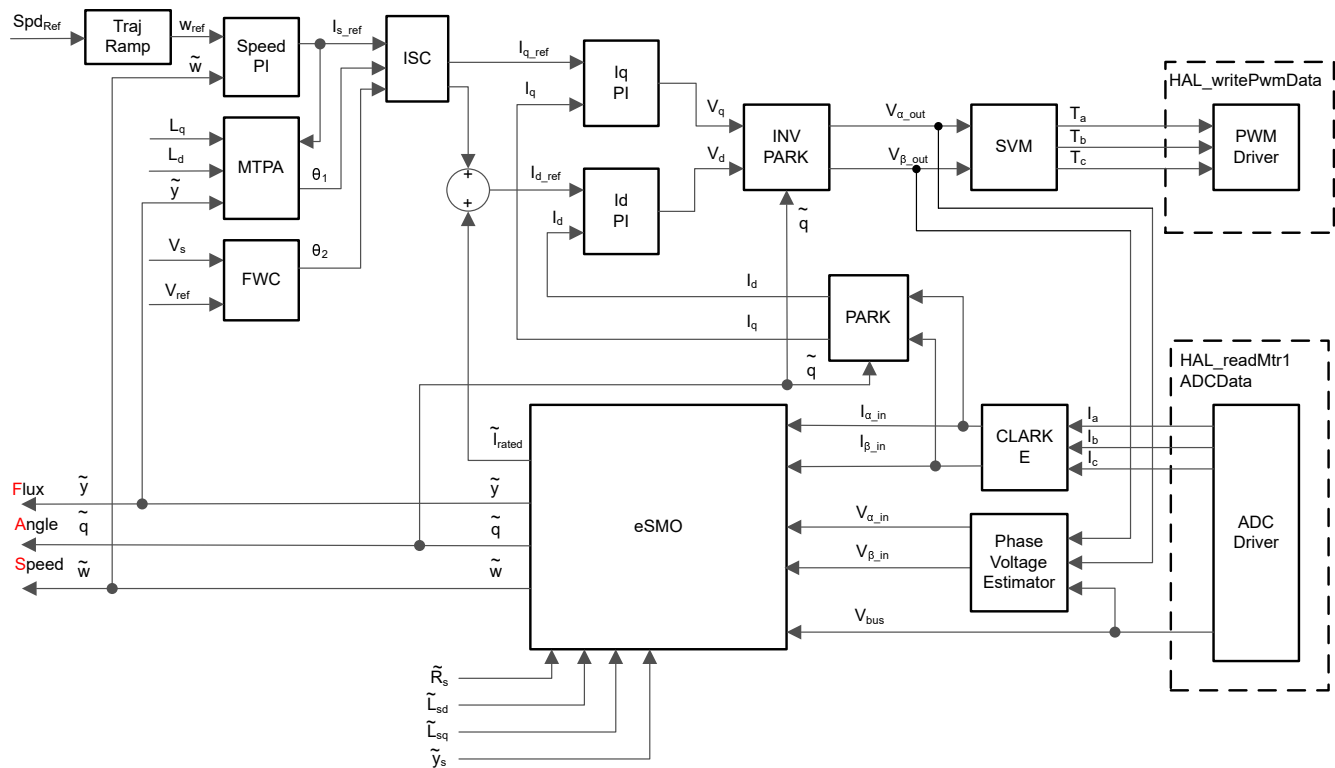


図 3-16. FWC と MTPA を備えた eSMO を使用した PMSM のセンサレス FOC

3.1.5.2 フライング スタート

フライング スタート (FS) は、ドライブが回転しているモーターの速度と方向を判定し、その速度と方向で出力電圧と周波数を開始する機能です。フライング スタートなしの場合は、ドライブは 0V および速度ゼロで出力を開始し、指示された速度まで上昇しようと試みます。負荷の慣性または回転方向によりモーターに大きなトルクを発生させる必要がある場合、過電流が発生し、ドライブに過電流トリップが発生することがあります。フライング スタートでこれらの問題は解消できます。

フライング スタートとは、**ゼロ**以外の任意の速度で制御を開始できる機能であり、エアコン アプリケーションのファン駆動で重要な機能です。

モーターを通常モードで起動すると、制御は最初に周波数 0Hz を適用し、目的の周波数まで上昇します。モーターがすでにゼロ以外の周波数で回転している状態で、このモードでドライブを起動すると、大電流が発生します。電流リミッタが迅速に応答できない場合、過電流トリップが発生する可能性があります。電流リミッタが過電流トリップを防止するのに十分な速さであっても、同期には許容できないほどの時間がかかり、モーターが目的の周波数に達するまで時間がかかることがあります。それに加えて、より大きな機械的ストレスがアプリケーションにかかります。

フライング スタート モードでは、スタートコマンドに対するドライブの応答は、モーターの速度 (周波数と位相) および電圧に同期することになります。その後、モーターは指令された周波数まで加速します。このプロセスにより過電流トリップが防止され、モーターが指令された周波数に達するまでの時間が大幅に短縮されます。ドライブは回転速度でモーターと同期し、適切な速度まで上昇するため、機械的ストレスはほとんどありません。

フライング スタート機能には、回転子速度を検索するアルゴリズムが実装されています。このアルゴリズムは、モーターに印加される励磁電流に対応するモーター電圧を検索します。

モーターが回転しているときは、BEMF 電圧から速度と位置の情報を推定できます。固定子電圧を測定するため、インバータを切り替えることで速度と位置を簡単に取得できます。モーターにゼロトルク電流を印加し、生成された電流と固定子電圧を測定した後、FOC モジュールはこれらの信号を使用して回転子の位置と速度を推定します。

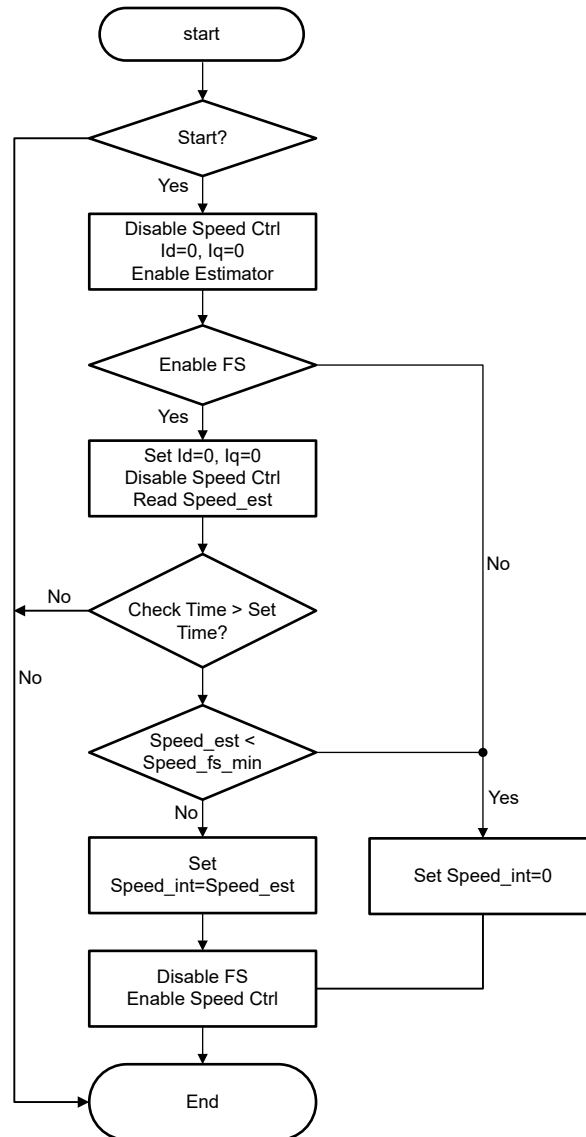


図 3-19. フライング スタート モジュール プログラムのフローチャート

4 ハードウェア、ソフトウェア、テスト要件、テスト結果

4.1 ハードウェア要件

表 4-1 に、ユニバーサル モーター制御プロジェクトでサポートされている最新の評価キットを示します。

表 4-1. ユニバーサル モーター制御でサポートされているモーター ドライブ評価キット

| モーター ドライブ評価ボード | | テキサス・インスツルメンツの MCU 評価ボード | 電流センシングトポロジ | 回転子位置センシング方式 | テスト対象モーター |
|--|---------------------------|---|----------------------------------|--|---|
| 部品番号 | 説明 | | | | |
| BOOSTXL-3PHGAN INV | 12-60V、3.5A 3 相 GaN インバータ | LP-AM263 | 3 つのシャント ベースのインライン モーター位相電流センシング | eSMO オブザーバ ベースのセンサレス FOC QEP エンコーダ ベースのセンサ付き FOC ホール センサ ベースのセンサ付き FOC | LVSERVOMTR (エンコーダとホール センサを内蔵) |
| TMDSHVMTRINSPI N⁽¹⁾ | 400V、10A 3 相インバータ | TMDSCNCD263 と TMSADAP180TO100 | 3 つのローサイド電流シャント | eSMO オブザーバ ベースのセンサレス FOC QEP エンコーダ ベースのセンサ付き FOC | HVPMSMMTR (エンコーダを内蔵) |

- (1) [LVSERVOMTR](#) のような低電圧モーターを高電圧キットで動作させる場合、J1、J2、J3、J4 にジャンパを実装し、820k の抵抗をバイパスして位相電圧と DC バス電圧をセンシングする必要があります。また、以下のコードに示すように、`user_mtr1.h` にパラメーターを設定します。高電圧キットでは、大電流で低インダクタンスの低電圧モーターを動作させないことを推奨します。

```
// Bypass the 820k resistor for low voltage motor on this kit
#define LV_JUMPER_EN // Bypass the 820k resistor
```

プロジェクトがエンコーダまたはホール ベースのセンサ付き FOC を使用するように設定されている場合は、物理的な接続が正しい順序で接続されていることを確認します。モーター、エンコーダ、またはホール ワイヤの接続順序が誤っていると、プロジェクトが正しく機能せず、モーターが回転できない可能性があります。モーター位相のワイヤについては、インバータ ボード上でモーター位相が正しい位相に接続されていることを確認します。テキサス・インスツルメンツのモーター制御リファレンス キットに付属のモーターの場合、表 4-2 に示すように、正しい位相接続が提供されています。

エンコーダの場合は A が A、B が B、I が I に接続されていること、ホール センサの場合は A が A、B が B、C が C に接続されていることを確認します。多くの場合、+5V DC およびグランド接続も必要になります。表 4-2 に示されているものとは異なるホール センサまたはエンコーダを使用している場合、使用しているホール センサまたはエンコーダのユーザー マニュアルを参照して、ワイヤが正しく接続されていることを確認してください。

ENC モジュールの設定と構成では、エンコーダの 1 回転あたりのスロット数が示されていることを確認してください。これにより、ENC モジュールはエンコーダ信号を正しく角度に変換できます。`user_mtr1.h` ファイルで定義されている `USER_MOTOR1_NUM_ENC_SLOTS` 定数は、使用するエンコーダ用の正しい値に更新する必要があります。この値が正しくない場合、設定された値に応じて、モーターの回転が高速または低速になります。この値は、直交精度を算出した後のカウント数ではなく、エンコーダのスロット数に設定されることに注意してください。

表 4-2. リファレンス キットとモーターのモーター位相、エンコーダ、ホール センサの接続

| | | LVSERVOMTR | HVPMSMMTR |
|-----------|-----|----------------------------|---------------------------|
| モーター位相ライン | U | BLACK (16AWG) | 赤色 |
| | V | RED (16AWG) | 青色 / 黒色 |
| | W | WHITE (16AWG) | 白色 |
| エンコーダ | GND | BLACK (J4-1) | 黒色 |
| | +5V | RED (J4-2) | 赤色 |
| | I | BROWN (J4-3) | 黄色 |
| | B | ORANGE (J4-4) | 緑色 |
| | A | BLUE (J4-1) | 青色 |

表 4-2. リファレンス キットとモーターのモーター位相、エンコーダ、ホール センサの接続 (続き)

| | | LVSERVOMTR | HVPMSMMTR |
|---------|-----|---------------------|--------------------------------------|
| ホール センサ | GND | BLACK (J10-1) | ホール センサ ベースのセンサ付き FOC はサポートしていません |
| | +5V | RED (J10-2) | |
| | A | GRAY-WHITE (J10-3) | |
| | B | GREEN-WHITE (J10-4) | |
| | C | GREEN (J10-5) | |

テキサス・インスツルメンツのリアルタイム制御マイクロコントローラ (MCU) を使用して、モーター制御の実装を開始します。

- ステップ 1: [表 4-1](#) に示すように、目的のモーター ドライブ評価ボード、テキサス・インスツルメンツの MCU 評価ボード、モータを注文します。
- ステップ 2: [MOTOR-CONTROL-SDK-AM263X](#) の最新バージョンをダウンロードします。
- ステップ 3: [Code Composer Studio IDE](#) の最新バージョンをダウンロードします。
- ステップ 4: 技術資料の指示に従ってハードウェアを設定し、以降のセクションで説明するプロジェクトを実行します。
- ステップ 5: 設計に関する質問への回答については、既存の回答を検索するか、[TI C2000 E2E 設計サポート フォーラム](#) を使用して自分で質問できます。

4.2 ソフトウェア要件

1. **Code Composer Studio (CCS) 統合開発環境 (IDE)** ツール フォルダから **Code Composer Studio** をダウンロードしてインストールします。バージョン **12.6** またはそれ以降をお勧めします。CCS のインストールと実装の詳細については、『**CCS ユーザー ガイド**』を参照してください。
2. テキサス・インスツルメンツが提供するリンクから **MOTOR-CONTROL-SDK-AM263X** ソフトウェア パッケージをダウンロードしてインストールし、この **Motor Control SDK** ソフトウェアをデフォルト フォルダにインストールします。**MOTOR-CONTROL-SDK-AM263X** は次の 2 つのいずれかの方法でインストールできます。
 - a. **MOTOR-CONTROL-SDK-AM263X** ダウンロード フォルダからソフトウェアをダウンロードします。
 - b. CCS にアクセスし、[View] → [Resource Explorer] に進みます。TI Resource Explorer で、[Arm®-based microcontrollers] → [MOTOR CONTROL SDK for AM263x] に進み、[Install] ボタンをクリックします。
3. インストールが完了したら、CCS を閉じ、プロジェクトをインポートするための新しいワークスペースを作成します。以降のセクションで説明するように、異なるインクリメント ビルドでこのプロジェクトをビルドして実行する手順に従います。

4.2.1 プロジェクトのインポートと構成

このプロジェクトは、テキサス・インスツルメンツの EVM モーター ドライバ キットをサポートし、AM263x MCU デバイスと一緒に使用できるユニバーサル モーター制御設計です。プロジェクトのビルド構成とプロパティを設定することで、テキサス・インスツルメンツの異なる EVM キットを実行できます。以降のセクションでは、**LP-AM263** を **BOOSTXL-3PHGANINV** ラボと組み合わせて使用し、このキットでサンプル ラボをインポートして実行する方法を示します。

1. [Project] → [Import CCS Project...] をクリックして CCS 内にプロジェクトをインポートし、[Browse...] をクリックして、
 - a. <install_location>\examples\ 検索ディレクトリを選択し、universal_motorcontrol_lab フォルダを選択します。
2. プロジェクトは、2 つのモーター ドライバ キットで実行するように構成できます。図 4-1 に示すように、インポートしたプロジェクト名を右クリックし、適切なビルド構成 (**3phGaN_3SC** など) を選択することで、これらのキットのいずれかを選択できます。
3. インポートしたプロジェクト名を右クリックしてプロジェクトのサポート機能を選択するようにプロジェクトを構成し、図 4-2 に示すように、[Properties] コマンドをクリックしてプロジェクトの事前定義シンボルを設定します。
 - a. 事前定義シンボルは、名前の **_N** を削除または追加することで有効または無効になります。たとえば、弱め界磁制御をイネーブルにするには、**MOTOR1_FWC_N** の **_N** を削除して **MOTOR1_FWC** に変更し、弱め界磁制御をディセーブルにするには、**MOTOR1_FWC** シンボル名を **MOTOR1_FWC_N** に変更します。
 - b. 上記のように関連する事前定義シンボルをイネーブルにして、モーターとハードウェア ボードに基づいて、適切にサポートされるモーター制御アルゴリズムを選択します。サポートされるアルゴリズムと関連するモーター マトリクスを表 4-3 に示します。
 - c. 図 4-2 に示すように、事前定義シンボルをイネーブルにして、適切にサポートされる機能を選択します。
4. 図 4-4 に示すように、ファイル名を右クリックし、ポップアップ メニューで [Set as Active Target Configuration] および [Set as Default Target Configuration] を選択して、適切なターゲット構成ファイル (.ccxml) を選択します。
 - a. **AM263_LP.ccxml** は、**LP-AM263** ベースのハードウェア キット向けです。
 - b. **AM263_CC.ccxml** は、**TMDSCNCD263** ベースのハードウェア キット向けです。
5. **user_mtr1.h** ファイルと **user_common.h** ファイルで適切なモーター モデルを選択または定義します。これらのファイルは、Project Explorer ウィンドウにある **src_board** フォルダの下にあります。テスト対象のモーターに対応する **#define** のコメントを解除し、残りの **#define** モーターがコメントアウトされたままであることを確認します。コード内のモーター パラメータが、接続されているモーターの仕様と一致していることを確認してください。
6. セクション 4.3 に示すように、ハードウェア キットを設定し、モーター、エンコーダ、ホール センサをキットに接続します。

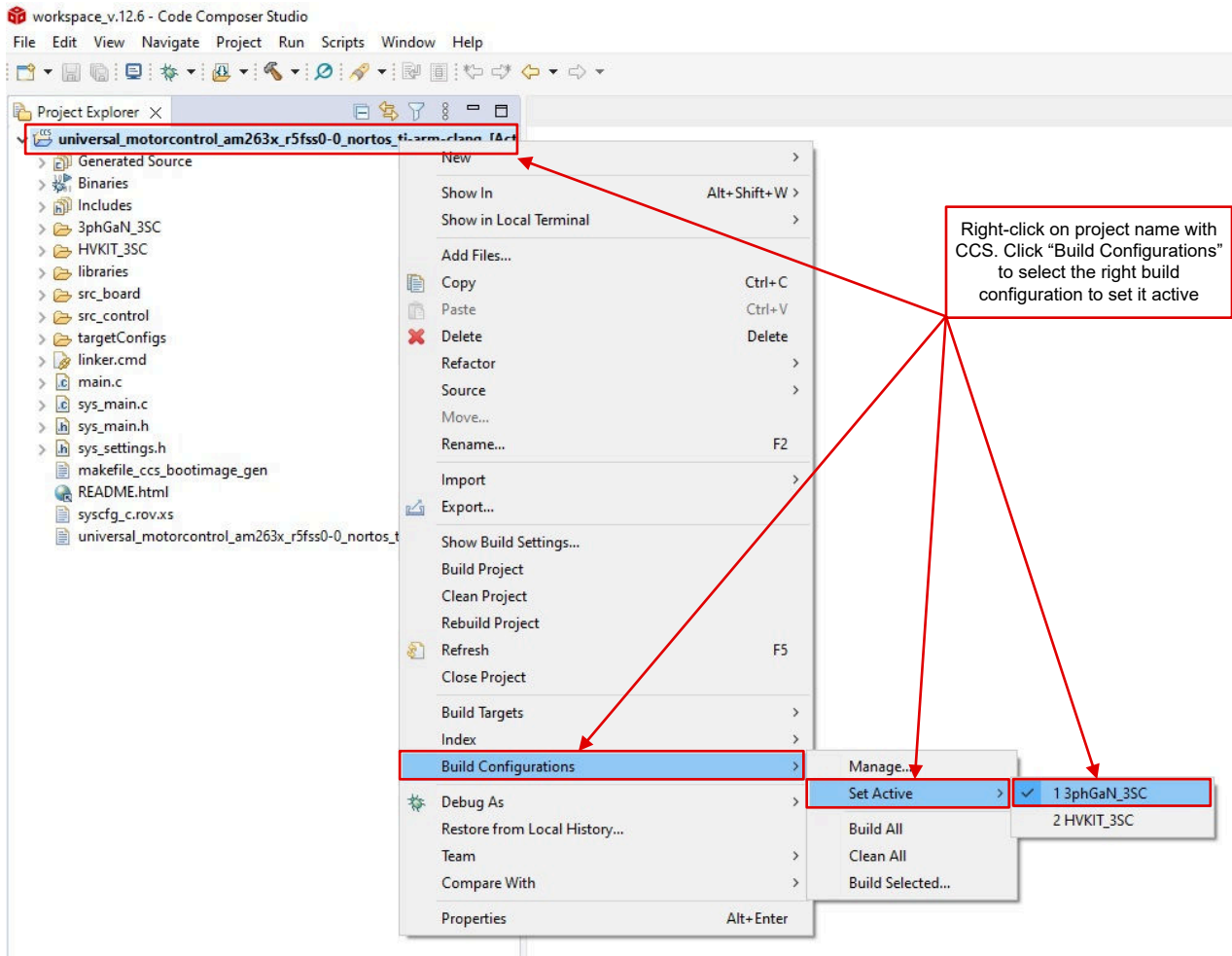


図 4-1. CCS 内で適切なビルド構成を選択する

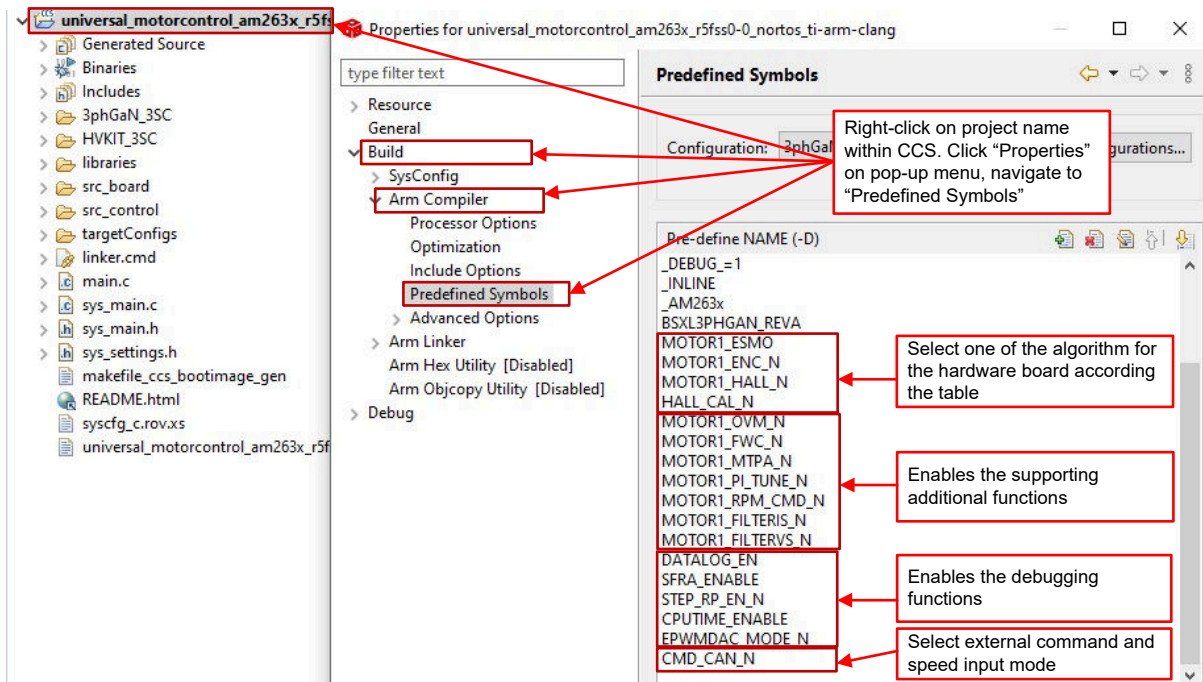


図 4-2. [Project Properties] で目的の事前定義シンボルを選択する

表 4-3. ユニバーサル モーター制御においてサポートされるアルゴリズム、機能、モーター マトリクス

| アルゴリズムまたは機能 | 事前定義シンボル | LaunchPad | controlCARD |
|-------------------------|-------------------------|-------------------|-----------------|
| | | BOOSTXL-3PHGANINV | TMDSHVMTRINSPIN |
| eSMO ベースのセンサレス FOC | MOTOR1_ESMO | ✓、LVSERVOMTR | ✓、HVPMSMMTR |
| QEP エンコーダ ベースのセンサ付き FOC | MOTOR1_ENC | ✓、LVSERVOMTR | ✓、HVPMSMMTR |
| ホール センサ ベースのセンサ付き FOC | MOTOR1_HALL HALL_CAL | ✓、LVSERVOMTR | ✗ |
| グラフ ツールによるデータログ | DATALOG_EN | ✓ | ✓ |
| PWMDAC | EPWMDAC_MODE | ✗ | ✓ |
| SFRA ツール | SFRA_ENABLE | ✓ | ✓ |
| グラフ ツールによるステップ応答 | STEP_RP_EN | ✓ | ✓ |

4.2.2 プロジェクト構造

プロジェクトの一般構造を図 4-3 に示します。デバイス ペリフェラルの構成は、TI SysConfig に基づいています。リファレンス デザイン ソフトウェアをカスタム ボードや別のデバイスに移行する場合は、hal.c ファイルと hal.h ファイルのコードと定義、user_mtr1.h ファイルのパラメータを変更することだけが必要です。

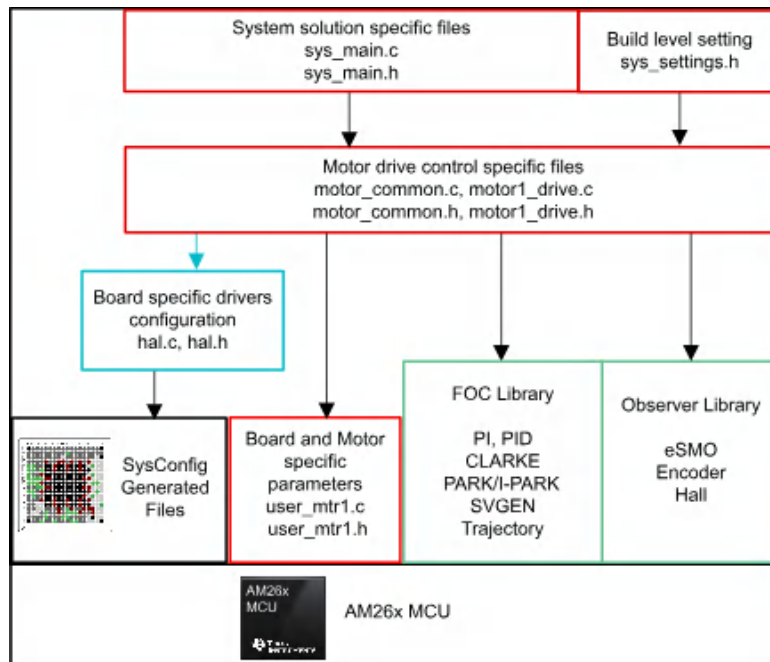


図 4-3. プロジェクト構造の概要

プロジェクトが CCS にインポートされると、図 4-4 に示すように、CCS 内に Project Explorer が表示されます。

transforms フォルダには、パーク、クラーク、逆パーク、SVGEN など、モータードライブ ISR の一部で、特定のデバイスやボードに依存しない標準的な FOC モジュールが含まれています。

libraries フォルダには、推定ライブラリ、および特定のデバイスやボードに固有ではないその他のライブラリが含まれています。

src_control フォルダには、割り込みサービスルーチンおよびバックグラウンド タスク内でモーター制御コア アルゴリズム関数を呼び出すモータードライブ制御ファイルが含まれています。

src_sys フォルダには、特定のデバイスやボードに依存しない、システム制御のために予約されたいくつかのファイルが含まれています。システム制御や通信などのコードを追加できます。

ボード固有のファイルとモーター固有のファイルは、`src_board` フォルダにあります。これらのファイルは、デザインを実行するデバイス固有のドライバで構成されています。独自のボード用にプロジェクトを移行する場合、または別のデバイスにプロジェクトを移行する場合、当該ボードのデバイス パリフェラルの使用方法に基づいて、`hal.c` ファイル、`hal.h` ファイル、`xxx.syscfg` ファイル、`user_mtr1.h` ファイルを変更することだけが必要です。

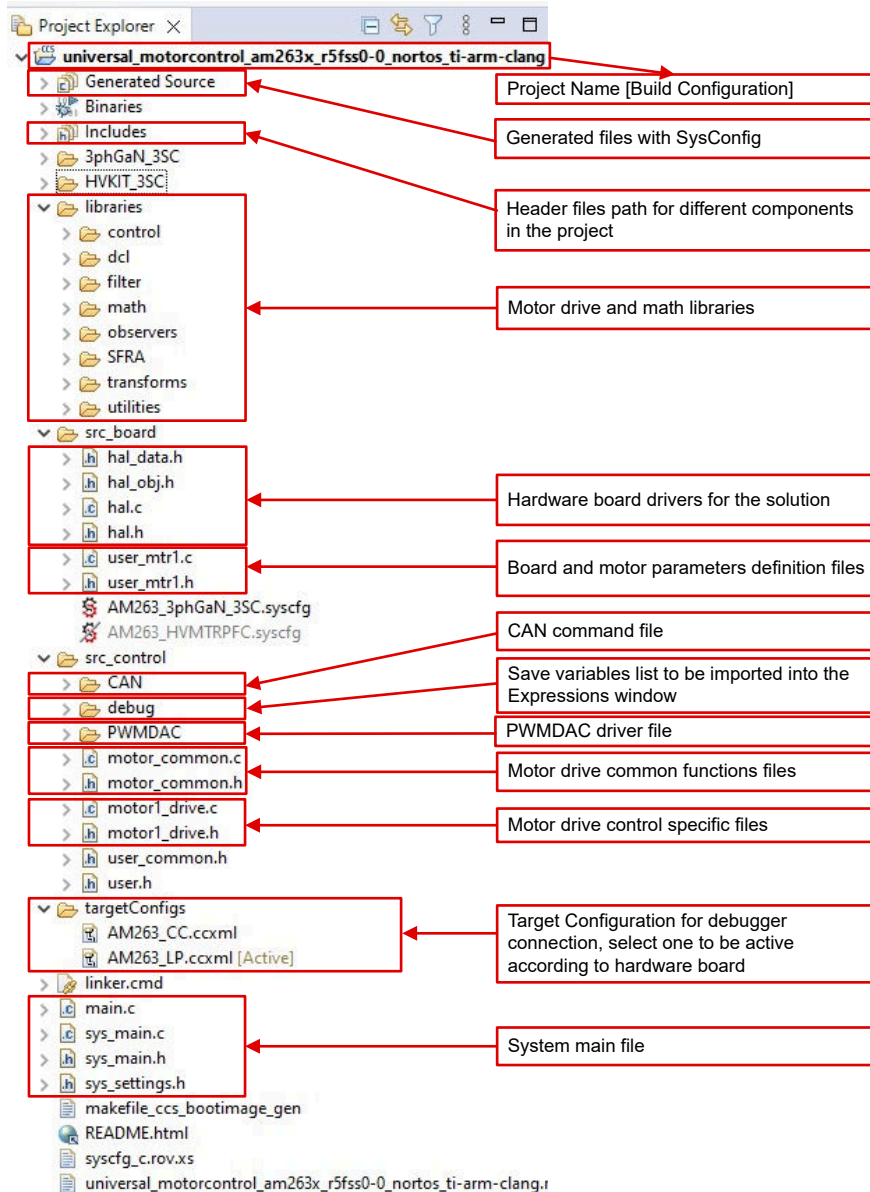


図 4.4. ユニバーサル モーター制御プロジェクトの Project Explorer ビュー

4.2.3 ラボ ソフトウェアの概要

図 4-5 に、ファームウェアのプロジェクトソフトウェアのフローチャートを示します。これには、リアルタイム モーター制御用の ISR と、バックグラウンド ループで更新されるモーター制御パラメータ用のメイン ループがそれぞれ 1 つ含まれています。ISR は、ADC の変換終了 (EOC) によってトリガされます。

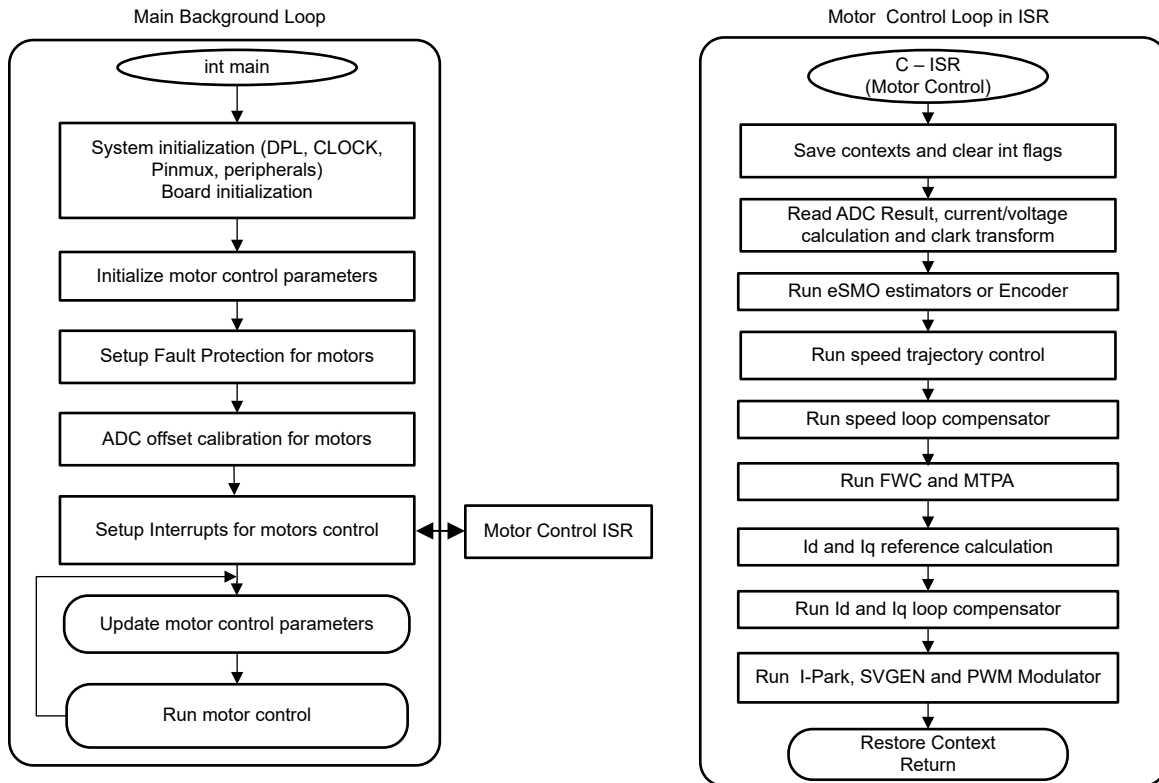


図 4-5. プロジェクトソフトウェアのフローチャート

システムの立ち上げとデザインを簡素化するため、ソフトウェアは 4 つのインクリメンタルビルドで構成されており、これによりボードやソフトウェアに慣れ、理解することが容易になります。また、このアプローチは、基板のデバッグやテストにも適しています。

表 4-4 に、このプロジェクトで使用するフレームワーク モジュールを示します。

表 4-4. プロジェクトにおけるモーター制御モジュールの使用

| モジュール名 | 説明 | アルゴリズム |
|---------------------------------|--|---------------|
| ANGLE_GEN_run | 開ループ動作用回転子角度ジェネレータ | eSMO、ENC、HALL |
| CLARKE_run | 電流または電圧のクラーク変換 | eSMO、ENC、HALL |
| collectRMSData、calculateRMSData | サンプリング値を収集し、位相電流と位相電圧の RMS 値を計算する | eSMO、ENC、HALL |
| DATALOG_update | グラフ ツールで表示するためにリアルタイム値を保存する | すべてのアルゴリズム |
| ENC_run | エンコーダに基づいて回転子角度を計算する | ENC |
| ESMO_run | センサレス FOC 用の拡張スライディング モード オブザーバ (eSMO) | eSMO |
| HAL_readMtr1ADCData | ADC 変換値を浮動小数点フォーマットで返す | すべてのアルゴリズム |
| HAL_writePWMDACData | ソフトウェア変数を PWM 信号に変換する | すべてのアルゴリズム |
| HAL_writePWMData | モーター用 PWM ドライブ | すべてのアルゴリズム |
| HALL_run | 回転子の角度と速度をホール センサに基づいて計算する | HALL |
| IPARK_run | 逆パーク変換 | eSMO、ENC、HALL |
| PARK_run | パーク変換 | eSMO、ENC、HALL |
| PI_run | 電流および速度の PI レギュレータ | すべてのアルゴリズム |
| PI_run_series | PI コントローラを連続して実行する | SFRA、MPTA |
| SPDCALC_run | エンコーダ信号からの角度に基づく速度の測定 | ENC |
| SPDFR_RUN | オブザーバからの角度に基づく速度の測定 | eSMO |
| SVGEN_runMin | 直交制御による空間ベクトル PWM | eSMO、ENC、HALL |
| TRAJ_run | 速度リファレンス設定軌道 | すべてのアルゴリズム |
| VS_FREQ_run | レベル 2 で V_d と V_q を計算するために、 v/f プロファイルでベクトル電圧を生成する。 特定のモーターに基づいて手動で行うことが可能。 | eSMO、ENC、HALL |

各インクリメンタル システム ビルドでテストされたモジュールを [表 4-5](#) にまとめています。

表 4-5. 各インクリメンタル ビルドで使用されるモーター制御モジュール

| ソフトウェア モジュール | DMC_LEVEL_1 | DMC_LEVEL_2 | DMC_LEVEL_3 | DMC_LEVEL_4 |
|---------------------|---|-------------------------------|--------------------------------------|--------------------------|
| | 50% PWM デューティ、ADC オフセット校正、PWM 出力、位相シフトの検証 | モーターの電流と電圧のセンシング信号を検証する開ループ制御 | ボード上の電流センシングと、PID による電流制御を検証する閉電流ループ | エスティメータ / オブザーバによる閉ループ動作 |
| HAL_readMtr1ADCData | √√ | ✓ | ✓ | ✓ |
| HAL_writePWMDData | √√ | ✓ | ✓ | ✓ |
| ANGLE_GEN_run | | √√ | ✓ | √(eSMO、ENC、HALL)* |
| VS_FREQ_run | | √√ | | |
| CLARKE_run | | ✓ | ✓ | ✓ |
| TRAJ_run | | √√ | ✓ | √√ |
| ESMO_run | | √(eSMO)* | √(eSMO)* | √√ (eSMO)* |
| SPDFR_RUN | | √(eSMO)* | √(eSMO)* | √√ (eSMO)* |
| ENC_run | | √(ENC)* | √(ENC)* | √√(ENC)* |
| SPDCALC_run | | √(ENC)* | √(ENC)* | √√(ENC)* |
| HALL_run | | √(HALL)* | √(HALL)* | √√(HALL)* |
| PARK_run | | ✓ | ✓ | ✓ |
| PI_run (Id) | | | √√ | ✓ |
| PI_run (Iq) | | | √√ | ✓ |
| PI_run (速度) | | | | √√ |
| IPARK_run | | √√ | ✓ | ✓ |
| SVGEN_runMin | | √√ | ✓ | ✓ |
| HAL_writePWMDACData | | √** | √** | √** |
| DATALOG_update | | ✓ | ✓ | ✓ |

1. √ は、このモジュールが使用されていることを意味します。√√ は、このモジュールはテスト中であることを意味します。
2. √(eSMO)* は、このモジュールが eSMO によってのみ使用されることを意味します。√(ENC)* は、このモジュールが ENC によってのみ使用されることを意味します。√(HALL)* は、このモジュールが HALL によってのみ使用されることを意味します。
3. √** は、[表 4-1](#) に示すように、このモジュールが一部のハードウェアキットでサポートされていることを意味します。

ユニバーサル プロジェクトでは、モーター制御用に FOC アルゴリズムのいずれかを個別に使用することも、eSMO アルゴリズムとエンコーダ FOC アルゴリズムの内の 2 つを同時に使用することもできます。プロジェクトに 2 つのアルゴリズムが実装されている場合、使用中のエスティメータをその場でスムーズに切り替えることができます。

4.3 テスト設定

このセクションでは、モータードライバ評価ボードをテキサス・インスツルメンツの開発ツールと組み合わせた場合のモーター制御用ハードウェアボードの設定方法について説明します。以降のセクションでは、異なるモータードライバ評価ボードでの詳細な操作手順を示します。

4.3.1 LP-AM263 の設定

LP-AM263 は、テキサス・インスツルメンツの Arm® ベースリアルタイム マイクロコントローラ向けの低コスト開発ボードです。この LaunchPad キットには開発用のピンが追加されており、2 つの BoosterPack™ プラグイン モジュールの接続をサポートしています。

- LP-AM263 の詳細については、『LP-AM263 LaunchPad ユーザー ガイド』を参照してください。
- LP-AM263 のブートスイッチが [図 4-6](#) に示すように設定されていることを確認してください。
 - QSPI_D0 (SOP0) の場合、スイッチを左側 (ロジック High) にします。
 - QSPI_D1 (SOP1) の場合、スイッチを左側 (ロジック High) にします。
 - SPI0_CLK_pad (SOP2) の場合、スイッチを右側 (ロジック Low) にします。
 - SPI0_D0_pad (SOP3) の場合、スイッチを左側 (ロジック High) にします。
- CMPSS が正常に動作するように、LP-AM263 の DAC VREF スイッチ (S1) が AM263x オンダイ LDO に設定されていることを確認してください。

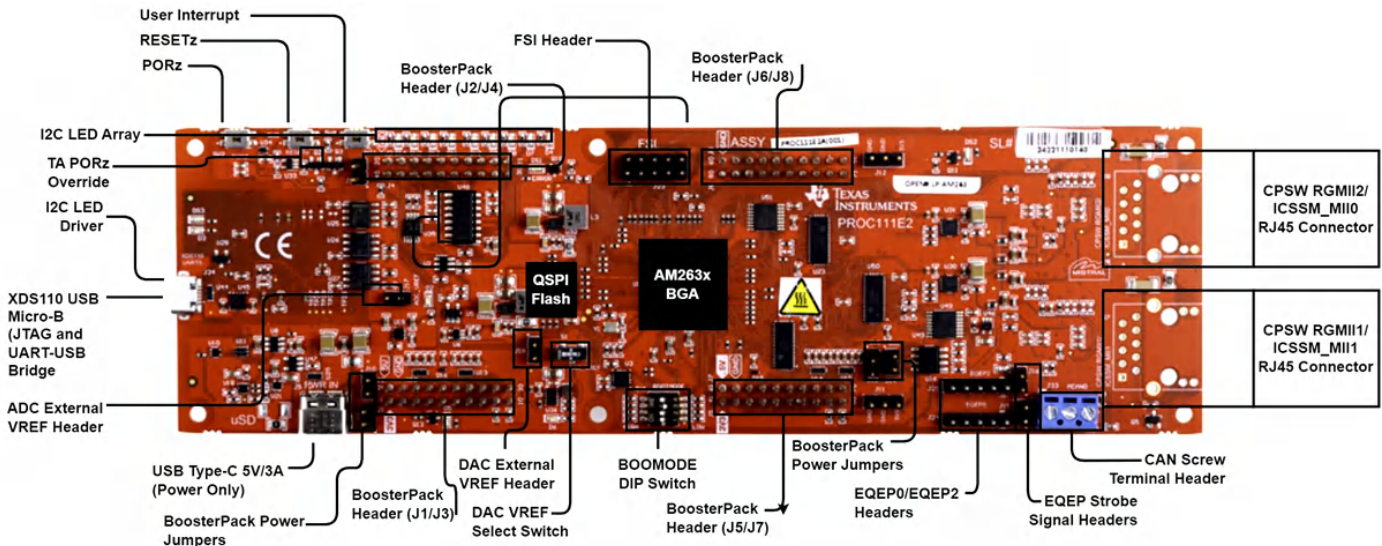
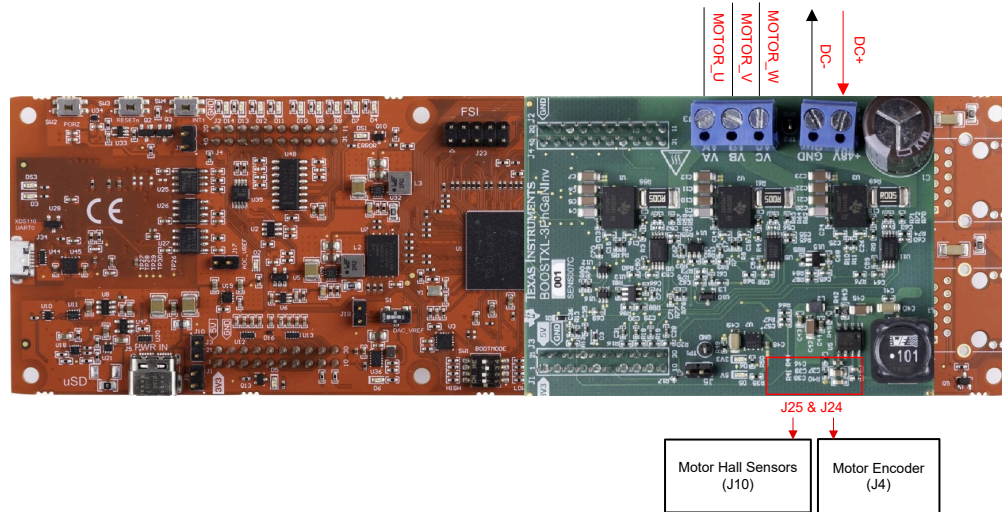


図 4-6. LP-AM263 LaunchPad™ ボードの概要とスイッチの設定

4.3.2 BOOSTXL-3PHGANINV の設定

BOOSTXL-3PHGANINV 評価ボードは、サーボドライブなどの高精度なドライブ制御のための高精度インライン シャントベース位相電流センシング機能を備えた 48V/10A の 3 相 GaN インバータを搭載しています。また、この評価ボードは個別の DC バスと 3 相電圧センシング機能も備えているため、センサレス FOC アルゴリズムを使用することを想定したテキサス・インスツルメンツの LaunchPad™ 開発キットを使用して、BLDC/PMSM 制御を行うことができます。

- ハードウェア ファイルなどの詳細については、TI.com の BOOSTXL-3PHGANINV ページを参照してください。
- BOOSTXL-3PHGANINV の詳細については、対応する『ユーザー ガイド』を参照してください。
- 次の項目が説明のとおり完了していることを確認し、[図 4-7](#) に示すように、BOOSTXL-3PHGANINV を LP-AM263 の J6/J8 および J5/J7 に接続します。
- [図 4-7](#) と [表 4-2](#) に示すように、モーター、エンコーダ、ホール センサを BOOSTXL-3PHGANINV および LP-AM263 に接続します。
- バッテリまたは DC 電圧源から 24V の電源電圧を電圧電源ピンに接続します。[セクション 4.4](#) の操作手順に従って、電源をオンにします。


図 4-7. LP-AM263 と BOOSTXL-3PHGANINV の接続

4.3.3 TMDSCNCD263 の設定

TMDSCNCD263 は、テキサス・インスツルメンツの Arm® ベース MCU シリーズの AM26x デバイスに適した評価 / 開発ボードです。TMDSCNCD263 は HSEC180 (180 ピンの高速エッジコネクタ) 搭載しており、**TMDSADAP180TO100** アダプタで既存の 100 ピン DIMM ベースの **TMDSHVMTRINSPIN** にて使用できます。

- **TMDSCNCD263** の詳細については、『**AM263x Sitara 制御カード ハードウェア ユーザー ガイド**』を参照してください。
- **TMDSCNCD263** のブートスイッチが、**図 4-8** に示すように設定されていることを確認してください。
 - SW3.1, SW3.2, SW3.4 は、スイッチを **LEFT** に、クワッド読み取り UART フォールバック モードでオンカード XDS110 エミュレータを使用する場合は SW3.2 を **RIGHT** にしてください。
- **TMDSCNCD263** の DAC VREF スイッチ (SW6) を **UP** に切り替えて、AM263x オンダイ LDO のリファレンス電圧を適切な CMPSS 動作に設定してください。

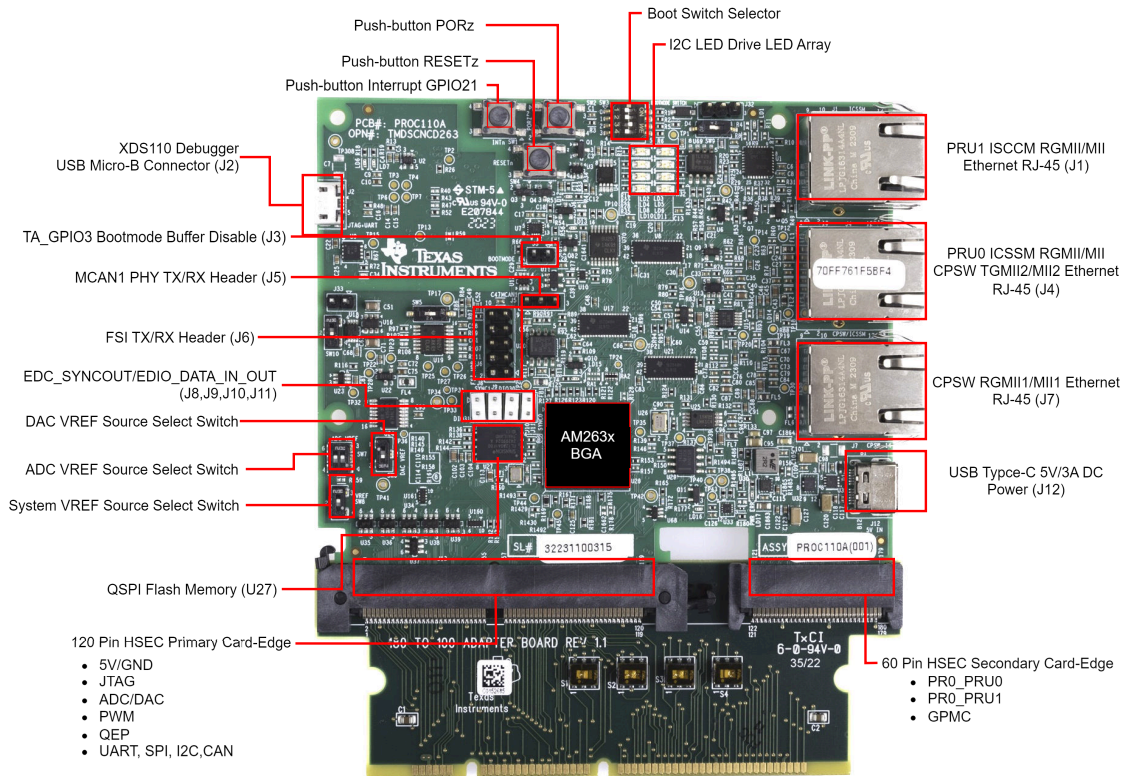


図 4-8. TMDSCNCD263 controlCARD とスイッチの設定

4.3.4 TMSADAP180TO100 の設定

TMSADAP180TO100 アダプタを使用すると、既存の 100 ピン DIMM ベースの評価ツールとともに、180 ピンのテキサス・インスツルメンツの controlCARD を使用できます。TMDSCNCD263 controlCARD を TMDSHVMTRINSPIN で使用するには TMSADAP180TO100 が必要です。

- ハードウェア ファイルは、C2000Ware の <install_location>\boards\controlCARDS\TMSADAP180TO100 フォルダにあります。
- スイッチ TMSADAP180TO100 が、図 4-9 に示すように設定されていることを確認してください。
 - S1 スイッチ、S2 スイッチ、S3 スイッチは **RIGHT** に、S4 スイッチは **LEFT** に配置する必要があります。

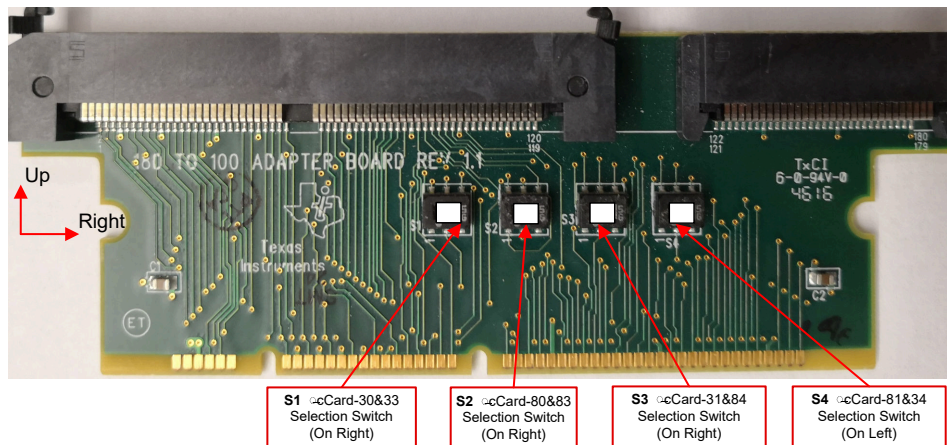


図 4-9. TMSADAP180TO100 のアダプタとスイッチの設定

4.3.5 TMDSHVMTRINSPIN の設定

警告

- この EVM はラボ環境のみで動作するものであり、テキサス・インスツルメンツでは一般消費者の使用に適した最終製品とはみなしていません。
- この EVM は、高電圧の電気機械部品、システム、およびサブシステムの取り扱いに関連するリスクを熟知した有資格のエンジニアや技術者のみが使用してください。
- 取り扱いが不適切な場合、この EVM は感電、火災、人身傷害をもたらす可能性のある電圧および電流で動作します。機器は注意深く使用して、人身傷害や物的損害を避けるために適切な安全対策が講じられる必要があります。
- 高電圧が印加されているため、EVM の電子機器を使用するときは常に注意してください。DC バスコンデンサは、主電源を切断した後も長時間充電されたままです。
- この EVM は、AC 主電源 / 壁面コンセントから電力供給を受けることができ、壁面コンセントからのライブラインとニュートラルラインのみを使用し、保護用のアースは接続されていません (フローティング状態です)。電源グランドは保護接地と接続されておらず、すべてのグランドプレーンが同じです。そのため、スコープや他のテスト機器をボードに接続する前に、適切な注意を払い、正しい絶縁要件を満たす必要があります。接地された機器を EVM に接続するときは、絶縁トランスを使用する必要があります。
- ボード上の電力段は個別に定格されています。ユーザーは、これらの電源ブロックを相互に接続してボードに電力を供給する前に、これらの定格 (たとえば、電圧、電流、電力などのレベル) をよく理解し、準拠していることを確認する責任があります。電力が供給されているときは、EVM や EVM に接続されている部品に触れないでください。

TMDSHVMTRINSPIN は、DIMM100 controlCARD ベースのマザーボード評価ボードで、AC 誘導 (ACI)、ブラシレス DC (BLDC)、永久磁石同期モーター (PMSM) などの最も一般的なタイプの高電圧 3 相モーターの制御を示します。高電圧モーター制御キットは個別の DC バスと 3 相電圧センシング機能を備えているため、テキサス・インスツルメンツの controlCARD™ を使用した BLDC/PMSM 制御向けのこのボードは、センサレス FOC アルゴリズムでの使用に最適です。

- ハードウェア ファイルは、**C2000WARE-MOTORCONTROL-SDK** の `<install_location>\solutions\tmdshvmtrinspin\hardware` フォルダにあります。

このセクションでは、MotorControl SDK を通じて提供されるソフトウェアで TMDSHVMTRINSPIN を実行するために必要な手順について説明します。キットは、controlCARD と接続するために、ジャンパとスイッチの設定が正しく配置された状態で出荷されます。以下に示すように、これらの設定がボード上で有効であることを確認し、[図 4-10](#) に示すように、TMDSHVMTRINSPIN ボードに TMDADAP180TO100 アダプタ付き controlCARD を挿入します。

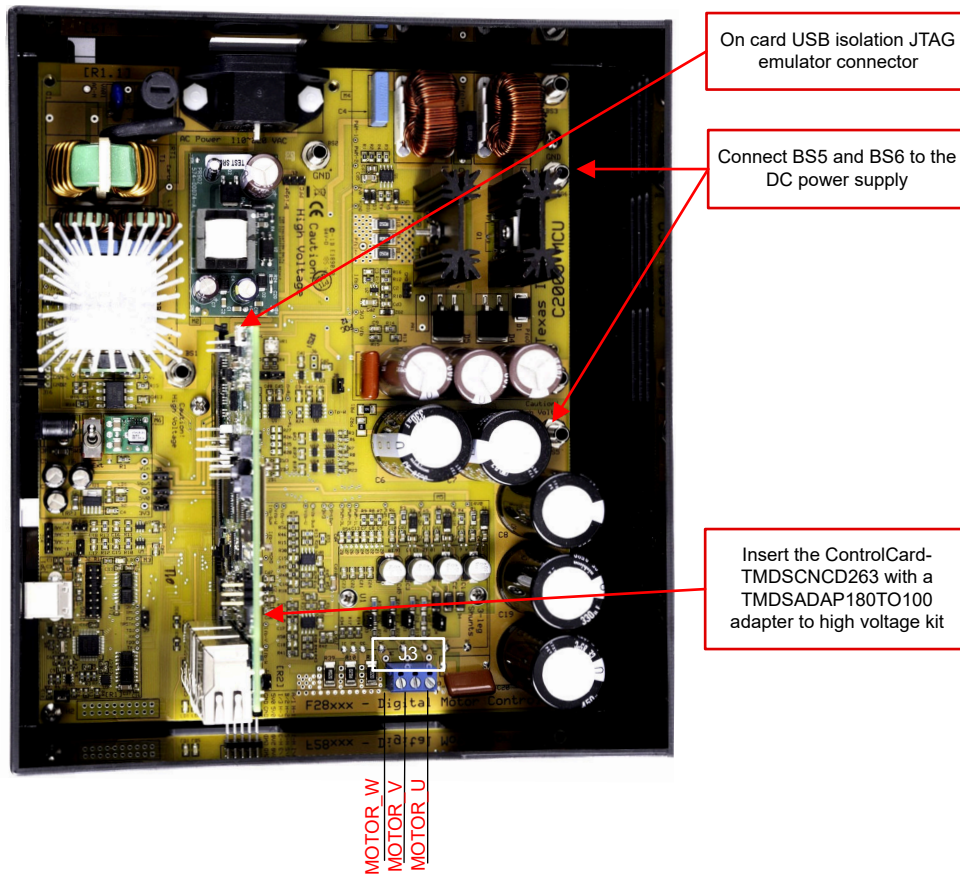


図 4-10. TMDSHVMTRINSPIN を TMSADAP180TO100 と一緒に TMDSCNCD263 に接続する

- ボードには何も接続されておらず、電力も供給されていないことを確認してください。
- **TMSADAP180TO100** アダプタ付きの controlCARD がまだ実装されていない場合は、[Main]-J1 controlCARD コネクタに挿入します。
- 図 4-11 に示すように、以下のジャンパ & コネクタの設定が正しく実装されていることを確認してください。
 - [Main]-J3, J4, J5, J8 は実装されています。
 - [Main]-J9 と [M3]-J5 は、controlCARD をオンボード エミュレーションで使用し、HVKIT の XDS100 をディスプレイにするために実装されていません。
 - [Main]-J7 は、ピン 2 とピン 3 (DIMM 100 ソケットから最も離れたピン) の間に実装されています。
 - 負荷 > 150W でモーターを動作させる場合、キットに付属の DC ファンが DC ファンジャンパ [Main]-J17 に接続されていることを確認してください。
- DC バス電源を得るための 2 つのオプションは次のとおりですが、外部 15V DC 電源の使用をお勧めします。
 - [Main]-J2 は、外部 15V DC 電源の +15V を使用する場合は実装されません。[M6]-SW1 がオフの位置にあることを確認し、15V DC 電源を [M6]-JP1 に接続します。
 - [Main]-J2 は、補助電源モジュールの +15V 電源を使用する場合、ブリッジと中央ピンの間にジャンパが実装されています。
- [M6]-SW1 をオンにします。これで [M6]-LD1 が点灯します。controlCARD の LED も点灯し、controlCARD がボードから電力供給を受けていることを示します。
- 図 4-11 と表 4-2 に示すように、モーター、エンコーダ、ホール センサをキットに接続します。
- AC 電源または DC 電圧源から電源電圧を電圧電源ピンに接続します。セクション 4.4 で指示があるときは電力が供給されますが、そうでない場合は切断したままにしてください。

表 4-6 に、ボード上のさまざまな接続を示します。ボード上のこれらの接続の位置は 図 4-11 に示します。

表 4-6. 主なジャンパ、コネクタの説明

| | |
|------------|-------------------------|
| [Main]-P1 | AC 入力コネクタ (110V~220VAC) |
| [Main]-TB3 | モーターを接続する端子台 |

表 4-6. 主なジャンパ、コネクタの説明 (続き)

| | |
|-----------------|--|
| [Main]-BS1 | AC 整流器からの出力用バナナ ジャック |
| [Main]-BS2、BS6 | GND の接続用バナナ ジャック |
| [Main]-BS3 | PFC 段の入力電圧の接続用バナナ ジャックで、通常は [Main]-BS1 コネクタからの整流 AC 電圧です。 |
| [Main]-BS4 | PFC 段からの出力への負荷の接続用バナナ ジャックで、PFC+モーター プロジェクトを使用する場合、PFC 段の出力は、たとえば [Main]-BS5 のようなインバータ バスの入力に接続されます。 |
| [Main]-BS5 | インバータ用 DC バス電圧入力用バナナ ジャック |
| [Main]-J2 | 補助電源モジュールの入力電圧選択ジャンパです。 <ul style="list-style-type: none"> ジャンパをブリッジの位置に接続すると、補助電源モジュールは AC 整流器ブリッジ出力から電力を供給します。 ジャンパを PFC の位置に接続すると、補助電源モジュールは PFC 段の出力から電力を供給します。 |
| [Main]-J3、J4、J5 | ジャンパ J3、ジャンパ J4、ジャンパ J5 は、それぞれ 15V、5V、3.3V の電力を 15V DC 電源からボードに供給するために使用します。 |
| [Main]-J7 | J7 は、過電流保護スレッシュホールド ソースの選択に使用します。 |
| [Main]-J8 | J8 は、IPM 過電流保護のイネーブル / ディセーブルに使用します。 |
| [Main]-J9 | JTAG TRSTn 切断ジャンパです。このジャンパを実装すると、マイクロコントローラへの JTAG 接続が可能になります。FLASH からブートする場合など、JTAG 接続が必要ない場合は、ジャンパを未実装にする必要があります。 |
| [Main]-J14 | PWMDAC 出力:PWM が 1 次ローパス フィルタに接続された結果の電圧出力です。ピン 1、ピン 2、ピン 3、ピン 4 はそれぞれローパス フィルタ処理された PWM 出力ピンに接続されており、オシロスコープでシステム変数を観測します。 |
| [Main]-J16 | 絶縁型 CAN バス コネクタ |
| [Main]-J17 | IPM ヒートシンクに取り付けられている DC ファン (ボードに同梱) に電力を供給するためのコネクタです。 |
| [Main]-H1 | QEP コネクタ:0-5V の QEP センサに接続し、モーターの速度と位置に関する情報を収集します。 CAP / ホール エフェクト センサ コネクタ:0-5V のセンサに接続し、モーターの速度と位置に関する情報を収集します。 |
| [M1]-F1 | AC 入力用ヒューズ |
| [M3]-JP1 | オンボード エミュレーション用 USB ケーブル |
| [M3]-J2 | 外部 JTAG インターフェイス:このコネクタから JTAG エミュレーション ピンにアクセスできます。外部エミュレーションが必要な場合は、[M3]-J5 にジャンパを取り付け、エミュレータをボードに接続します。エミュレーション ロジックに電力を供給するには、引き続き USB コネクタを [M3]-JP1 に接続する必要があります。 |
| [M3]-J5 | オンボード エミュレーション ディスエーブル ジャンパ:ここにジャンパを取り付けると、オンボード エミュレータがディスエーブルになり、外部インターフェイスにアクセスできるようになります。 |

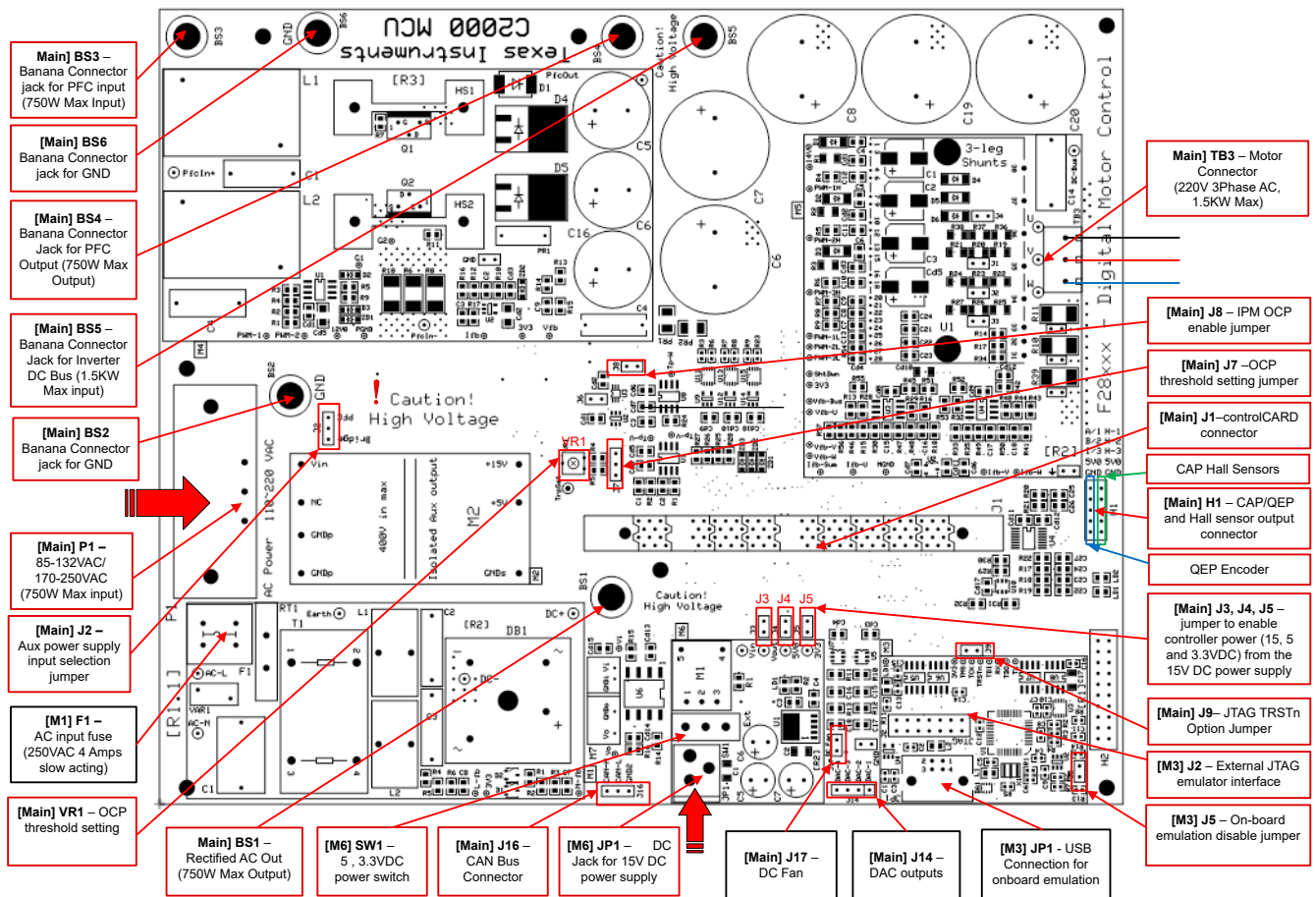


図 4-11. TMS320F28389 Motor Control キットのジャンパとコネクタの図

4.4 テスト結果

システムは、少しずつテストされ、最終的なシステムが自信を持って操作できるように、複数の段階で検証されます。特定のビルド オプションを選択するには、`sys_settings.h` ファイルで `DMC_BUILDLEVEL` 定義の値を目的の `DMC_LEVEL_X` オプションに変更します。ビルド オプションを選択した後、プロジェクト名を右クリックし、[Rebuild Project] をクリックしてプロジェクトをコンパイルします。

4.4.1 レベル 1 インクリメンタル ビルド

このビルド レベルの目標:

- HAL オブジェクトを使用して、モーター ドライバ ハードウェア用 MCU のペリフェラルを初期化する
- PWM ドライバ モジュール と ADC ドライバ モジュールを検証する
- ADC オフセット検証を確認する
- CCS の操作に慣れる CCS の詳細については、『[CCS ユーザー ガイド](#)』を参照してください。

このビルド レベルでは、ボードは固定 PWM デューティ サイクルの開ループ モードで実行されます。デューティ サイクルは 50% に設定されています。このビルド レベルは、電力段からの帰還値のセンシングと PWM ゲートドライバの動作を検証し、ハードウェアに問題がないことを確認します。さらに、入出力電圧センシングの較正もこのビルド レベルで実行できます。このプロセス中は、モーターの接続を切り離れたままにしておく必要があります。このビルド レベルのソフトウェア ブロック図を [図 4-12](#) に示します。

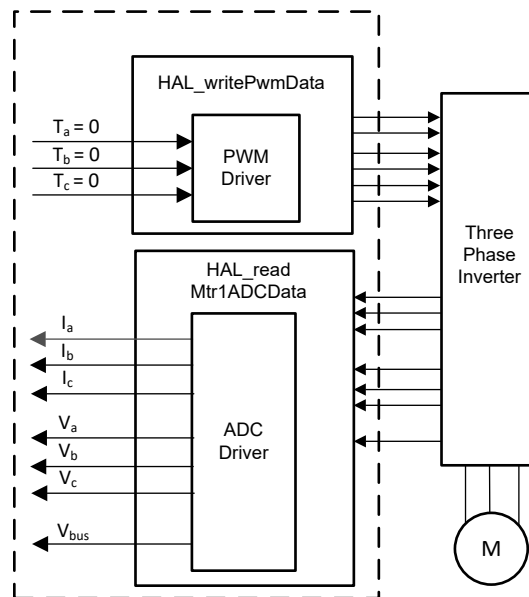


図 4-12. ビルドレベル 1 ソフトウェア ブロック図 - オフセット検証

4.4.1.1 プロジェクトのビルドとロード

1. このビルド レベルではモーターをモーター ドライバ ボードに接続する必要がないことを除き、モーター ドライバ ハードウェア ボードと、テキサス・インスツルメンツの LaunchPad または controlCARD を [セクション 4.3](#) に示すように設定します。
2. USB ケーブルをコンピュータからテキサス・インスツルメンツの LaunchPad または controlCARD のオンボード USB コネクタに接続し、MCU に対する絶縁型 JTAG エミュレーションをイネーブルにします。
3. [セクション 4.3](#) に示すように、バス電圧入力端子に適切な電圧を印加して、モーター ドライバ ボードに電力供給します。
4. ユニバーサル モーター制御プロジェクトを CCS にインポートし、[セクション 4.2.1](#) に示すように、正しいビルド構成を選択します。`sys_settings.h` ファイルを開き、`DMC_BUILDLEVEL` を `DMC_LEVEL_1` に設定します。これにより、プロジェクトが最初のインクリメンタル ビルドを実行するように構成されていることを確認できます。
5. Project Explorer ウィンドウで、目的のターゲット構成ファイル名を右クリックし、[Set as Active Target Configuration] を選択して、正しいターゲット構成ファイルが Active に設定されていることを確認します。また、ファイル名を右クリックし、[Set as Default Target Configuration] を選択して、目的のターゲット構成ファイルをデフォルトとして設定するこ

とをお勧めします。これを行う一つの理由は、どのファイルが **Active** であるかを示すインジケータが表示されないからです。ファイルがデフォルトに設定されている場合は、**Project Explorer** ウィンドウでファイル名の横に **[Default]** インジケータが表示されます。また、ファイルをデフォルトに設定すると、別の構成ファイルが特に **Active** に設定されない限り、そのファイルがデフォルトとして使用されるようになります。**[View] > [Target Configurations]** に進んで、**[Target Configurations]** ビューでターゲット構成名を右クリックし、**[Link to Project]** を選択すると、ターゲット構成をワークスペース内のプロジェクトにリンクすることもできます。

- プロジェクト名を右クリックし、**[Rebuild Project]** をクリックします。**Console** ウィンドウを確認します。プロジェクトのエラーはすべて **Console** ウィンドウに表示されます。
- ビルドが正常に完了したら、**[Debug]** ボタンをクリックするか、**[Run] → [Debug]** をクリックします。これで IDE はターゲットに自動的に接続され、出力ファイルがデバイスにロードされて、**[Debug]** パースペクティブに変更されます。右上隅に **CCS Debug** アイコンが表示され、**[Debug Perspective]** ビューに移動したことを示しています。プログラムは **main()** の開始地点で停止する必要があります。

4.4.1.2 デバッグ環境設定ウィンドウ

コードのデバッグ中にローカル変数とグローバル変数を注意深く観察することは、デバッグの標準的なやり方です。**CCS** では、このような変数の観察を行うために、メモリビューやウォッチビューなど、さまざまな方法を提供しています。さらに、**CCS** には時間 (および周波数) ドメインのプロットを作成する機能があります。この機能により、ユーザーはグラフツールを使用して波形を表示できます。グラフツールの設定と構成の方法については、[セクション 4.5.1](#) を参照してください。**[Expressions]** ウィンドウの設定については、次の手順を参照してください。

- [Watch]** ウィンドウの設定: メニューバーの **[View] → [Expressions]** をクリックして、**[Expressions]** ウォッチウィンドウを開きます。**[Expressions]** ウィンドウで **[Add new expression]** をクリックし、変数名を入力して、**[Enter]** キーを押すと、**[Expressions]** ウィンドウに変数を追加できます。変数値が表示される数値形式は、変数値が宣言されたときに変数に関連付けられた数値形式に基づいています。変数を右クリックし、**[Number Format]** に移動して目的の形式を選択することで、特定の変数の数値形式を変更できます。
- 代わりに、**[Expressions]** ウィンドウ内で右クリックして **[Import]** をクリックすることで、変数のグループを **[Expressions]** ウィンドウにインポートすることもできます。
`<workspace>\universal_motorcontrol_am263x_r5fss0-0_nortos_ti-arm-c\lang\src_control\debug\` にあるプロジェクトのディレクトリを参照し、`universal_motor_control_level1.txt` ファイルを選択して **[OK]** をクリックすると、[図 4-13](#) に示す変数がインポートされます。

注

メインコードではこの時点で初期化されていない変数もあり、役に立たない値が含まれている可能性があります。

- 注: 構造体変数 `motorVars_M1` には、モータードライブ制御に関連するほとんどの変数へのリファレンスが含まれています。
- [Expressions Window]** タブの右上隅にある **[Continuous Refresh]** ボタンをクリックし、マイクロコントローラからの定期的なデータキャプチャを有効にします。**[View Menu]** ボタン (**[Expressions]** ウィンドウの右上隅にある 3 つのドット) をクリックすることで、**[Continuous Refresh Interval]** を選択して **[Expressions]** ウィンドウのリフレッシュレートを編集できます。リフレッシュ間隔が速すぎると、パフォーマンスに影響することがあるので注意してください。

4.4.1.3 コードの実行

1. [Tools] > [ARM Advanced Features] で [Data Cache Enabled] のチェックを外して、データ キャッシュをディスエーブルにします。
2. [Resume] ボタンを押してコードを実行するか、[Debug] タブで [Run] → [Resume] をクリックします。
3. これでプロジェクトが実行され、グラフおよびウォッチ ウィンドウの値が常に更新されます。
4. ウォッチ ウィンドウで、`systemVars.flagEnableSystem` が自動的に 1 に設定された後、[Expressions] ウィンドウで変数 `motorVars_M1.flagEnableRunAndIdentify` を 1 に設定します。
5. これでプロジェクトが実行され、このプロジェクトを使用している間は、[図 4-13](#) に示すように、グラフと [Expressions] ウィンドウの値が常に更新されます。ウィンドウのサイズは、お好みに合わせて変更できます。
6. ウォッチ ビューでは、フォルトが発生していない場合、変数 `motorVars_M1.flagRunIdentAndOnLine` は自動的に 1 に設定されます。[ISRCCount] は連続的に増加します。
7. モータードライバ ボードの較正オフセットをチェックします。[図 4-13](#) に示すように、モーター位相電流センシング値のオフセット値は、ADC のスケール電流の半分程度に相当します。
8. グラフ ツールを使用する場合、グラフに表示される変数は、u 相、v 相、w 相の FOC 角度と位相電流です。
9. `MotorVars_M1.faultMtrPrev.bit` 構造体を展開してチェックし、フォルト フラグが設定されていないことを確認します。
10. オシロスコープを使用して、モーター駆動制御に使用する PWM 出力を測定します。このビルド レベルでは、3 つの PWM のデューティ サイクルが 50% に設定され、予測される PWM 出力の波形は、[図 4-14](#) のようになります。PWM スwitching 周波数は、`user_mtr1.h` ファイルの `USER_M1_PWM_FREQ_KHz` 定義に設定された値と同じです。
11. `motorVars_M1.flagEnableRunAndIdentify` 変数を 0 に設定して、PWM をディスエーブルにします。
12. これまでの手順のいずれかで予期しない結果が得られた場合は、追加のデバッグが必要です。以下を確認してください。
 - a. 使用するモータードライバ ボードが、ビルド構成で選択したボードと同じであることを確認してください。
 - b. 適切な事前定義が設定されていることを確認してください。
 - c. [セクション 4.3](#) に示すように、Lunchpad/ControlCARD で各スイッチが正しく構成されていることを確認してください。
13. これまでの手順が完了したら、コントローラを停止し、デバッグ接続を終了できます。まずツールバーの [Halt] ボタンをクリックするか、[Target] → [Halt] をクリックして、コントローラを停止します。最後に、ボタンをクリックするか、[Run] → [Reset] → [CPU Reset] をクリックして、コントローラをリセットします。
14. [Terminate Debug Session] ボタンをクリックするか、[Run] → [Terminate] をクリックして、CCS デバッグ セッションを終了します。これによりプログラムが停止し、コード コンポーザが MCU から切り離されます。
15. コードを変更または再実行するたびにデバッグ セッションを終了する必要はありません。代わりに、次の手順を行うことができます。プロジェクトを再ビルドしたら、ボタンを押すか、[Run] → [Reset] → [CPU Reset] をクリックした後、[Restart] ボタンを押すか、[Run] → [Restart] をクリックします。目標デバイスや構成が変更された場合は、CCS をシャットダウンする前に、プロジェクトを終了する必要があります。

| Expression | Type | Value | Address |
|--|-----------------------|---|------------|
| (*)= systemVars.boardKit | enum Board_Kit_e | BOARD_BSXL3PHGAN_REVA | 0x0008354E |
| (*)= systemVars.estType | enum EST_Type_e | EST_TYPE_ESMO | 0x00083550 |
| (*)= systemVars.currentSenseType | enum CURRENTSEN_T... | CURSEN_TYPE_INLINE_SHUNT | 0x00083551 |
| (*)= motorVars_M1.motorState | enum MOTOR_Status_e | MOTOR_CL_RUNNING | 0x00082ED2 |
| (*)= motorVars_M1.estimatorMode | enum ESTIMATOR_Mo... | ESTIMATOR_MODE_ESMO | 0x00082ED0 |
| (*)= motorVars_M1.ISRCount | unsigned int | 2511665 | 0x00082F6C |
| (*)= motorVars_M1.speedRef_Hz | float | 60.0 | 0x00082F88 |
| (*)= motorVars_M1.speed_Hz | float | -2.14497733 | 0x00082F90 |
| (*)= motorVars_M1.flagEnableRunAndIdentify | unsigned short | 1 | 0x00082E94 |
| (*)= motorVars_M1.flagRunIdentAndOnLine | unsigned short | 1 | 0x00082E96 |
| (*)= motorVars_M1.flagClearFaults | unsigned short | 0 | 0x00082EBA |
| (*)= motorVars_M1.faultMtrUse.all | unsigned short | 0 | 0x00082ECA |
| > motorVars_M1.faultMtrPrev.bit | struct FAULT_MTR_BITS | {overVoltage=0,underVoltage=0,motorOv... | 0x00082ECE |
| (*)= motorSetVars_M1.dacCMPValH | unsigned short | 2978 | 0x00082C74 |
| (*)= motorSetVars_M1.dacCMPValL | unsigned short | 1118 | 0x00082C76 |
| (*)= motorSetVars_M1.overCurrent_A | float | 7.5 | 0x00082CEC |
| (*)= motorVars_M1.angleFOC_rad | float | 0.673689544 | 0x0008305C |
| (*)= motorVars_M1.adcData.VdcBus_V | float | 25.3888645 | 0x00082F10 |
| (*)= motorVars_M1.Vdq_out_V.value[0] | float | -0.667759538 | 0x00083010 |
| (*)= motorVars_M1.Vdq_out_V.value[1] | float | -0.456603527 | 0x00083014 |
| motorVars_M1.Irms_A | float[3] | [0.030551888,0.0419260897,0.027164869] | 0x0008308C |
| (*)= [0] | float | 0.030551888 | 0x0008308C |
| (*)= [1] | float | 0.0419260897 | 0x00083090 |
| (*)= [2] | float | 0.027164869 | 0x00083094 |
| motorVars_M1.adcData | struct HAL_ADCData_t | {VdcBus_V=25.368969,I_A={value=[0.0080... | 0x00082F10 |
| (*)= VdcBus_V | float | 25.3490715 | 0x00082F10 |
| > I_A | struct MATH_Vec3 | {value=[-0.0241699219,0.0322265625,-0.02... | 0x00082F14 |
| > V_V | struct MATH_Vec3 | {value=[12.853611,12.8337135,12.5352545]} | 0x00082F20 |
| > offset_I_ad | struct MATH_Vec3 | {value=[2115.22485,2116.18799,2116.7482... | 0x00082F2C |
| value | float[3] | [2115.22485,2116.18799,2116.74829] | 0x00082F2C |
| (*)= [0] | float | 2115.22485 | 0x00082F2C |
| (*)= [1] | float | 2116.18799 | 0x00082F30 |
| (*)= [2] | float | 2116.74829 | 0x00082F34 |
| > offset_V_sf | struct MATH_Vec3 | {value=[0.0,0.0,0.0]} | 0x00082F38 |
| (*)= current_sf | float | -0.00805664062 | 0x00082F44 |
| (*)= voltage_sf | float | 0.01989723 | 0x00082F48 |
| (*)= dcBusvoltage_sf | float | 0.01989723 | 0x00082F4C |
| (*)= motorVars_M1.flagEnableForceAngle | unsigned short | 1 | 0x00082E9C |
| (*)= motorVars_M1.flagMotorIdentified | unsigned short | 1 | 0x00082E9A |
| (*)= motorSetVars_M1.Kp_Id | float | 0.239317492 | 0x00082CA0 |
| (*)= motorSetVars_M1.Ki_Id | float | 0.0344771557 | 0x00082CA4 |
| (*)= motorSetVars_M1.Kp_Iq | float | 0.239317492 | 0x00082CA8 |
| (*)= motorSetVars_M1.Ki_Iq | float | 0.0344771557 | 0x00082CAC |
| (*)= motorSetVars_M1.Kp_spd | float | 0.011634578 | 0x00082C98 |
| (*)= motorSetVars_M1.Ki_spd | float | 0.00209439523 | 0x00082C9C |
| (*)= motorVars_M1.faultMtrNow.all | unsigned short | 0 | 0x00082EC8 |

図 4-13. ビルドレベル 1:[Expressions] ウィンドウの変数

ゲートドライブの入力へのデッドバンド出力付き PWM を、[図 4-14](#) に示します。

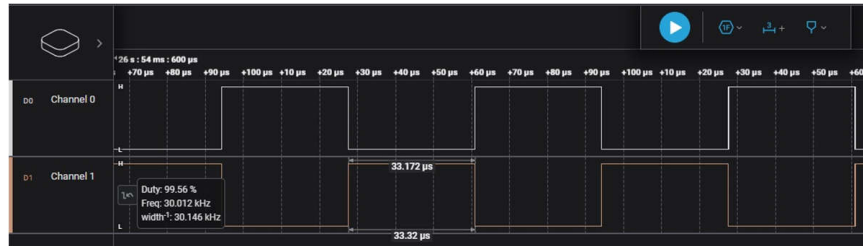


図 4-14. ビルドレベル 1: PWM 出力の波形

4.4.2 レベル 2 インクリメンタルビルド

このビルドレベルの目標:

- 電流と電圧のセンシング回路とゲートドライバ回路の検証のための、モーター駆動用の簡単なスカラー v/f 制御の実装
- モーター制御用 eSMO モジュールのテスト

このビルドレベルでは、システムが開ループ制御で動作するため、ADC 値は確認と検証のためだけに使用され、ADC 値が実際にモーターの制御ループで使用されることはありません。このビルドレベルのソフトウェアフローを [図 4-15](#) に示します。

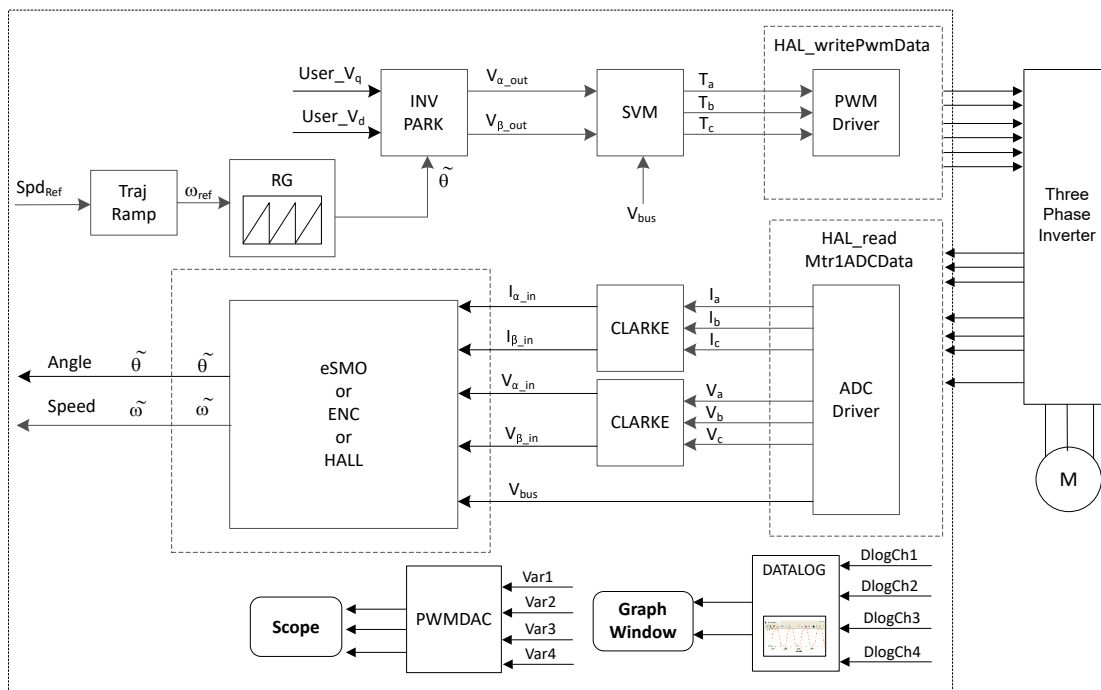


図 4-15. ビルドレベル 2 ソフトウェアブロック図 - 開ループ制御

4.4.2.1 プロジェクトのビルドとロード

モーターをモータードライバ評価ボードの適切な端子に接続します。[セクション 4.4.1.1](#) のステップ 2~7 に従って、プロジェクトをビルドして、ロードします。ステップ 4 で、DMC_BUILDLEVEL を DMC_LEVEL_2 に設定します。

4.4.2.2 デバッグ環境設定ウィンドウ

[セクション 4.4.1.2](#) の手順に従って、変数を [Expressions] ウィンドウにインポートします。ビルドレベル 2 では、universal_motor_control_level2.txt ファイルを選択します。[図 4-16](#) に示すように、[Expressions] ウィンドウが表示されます。

4.4.2.3 コードの実行

- 適切な電源に電力を供給し、電源の出力電圧を徐々に上げて、適切な DC バス電圧を取得します。
- グラフ ツールを使用する場合、レベル 2 ではラボ 1 と同じグラフ構成とパラメータを使用して、位相電流の 2 相を監視します。
- [Tools] > [Arm Advanced Features] で [Data Cache Enabled] のチェックを外して、データ キャッシュをディスエーブルにします。
- [Resume] ボタンをクリックしてプロジェクトを実行するか、[Debug] タブで [Run] → [Resume] をクリックします。一定の時間が経過したら、`systemVars.flagEnableSystem` は 1 に設定されます。これは、オフセット較正が完了したことを意味します。フォルト フラグ `motorVars_M1.faultMtrUse.all` は 0 になります。そうでない場合は、[セクション 4.4.1.3](#) に示すように、レベル 1 の電流および電圧のセンシング回路を再確認する必要があります。また、`motorVars_M1.faultMtrPrev.bit` の `moduleOverCurrent` が 1 の場合、`motorSetVars_M1.overcurrent_A` を高い値に設定して、初期の大電流フォルトを回避する必要があります。
- モータードライバの電流および電圧のセンシング回路を検証するには、[図 4-16](#) に示すように、[Expressions] ウィンドウで変数 `motorVars_M1.flagEnableRunAndIdentify` を 1 に設定します。モーターは、電圧 / 周波数 (v/f) の開ループで動作します。モーターがスムーズに回転しない場合、モーターの仕様に応じて、`user_mtr1.h` ファイルの `v/f` プロファイル パラメータを以下に示すように調整します。注:これらのパラメータを変更するには、プロジェクトをリビルドする必要があります。デバッグ モードでのプロジェクトのリビルドの詳細については、[セクション 4.4.1.3](#) のステップ 15 を参照してください。

```
#define USER_MOTOR1_FREQ_LOW_HZ      (5.0)           // Hz
#define USER_MOTOR1_FREQ_HIGH_HZ    (400.0)          // Hz
#define USER_MOTOR1_VOLT_MIN_V      (1.0)            // Volt
#define USER_MOTOR1_VOLT_MAX_V      (24.0)           // Volt
```

- `motorVars_M1.speedRef_Hz` 変数は、モーターの速度リファレンスを設定するために使用されます。[Expressions] ウィンドウで `motorVars_M1.speed_Hz` 変数の値をチェックして、[図 4-16](#) に示すように、モーター速度 (`motorVars_M1.speed_Hz`) がリファレンス速度 (`motorVars_M1.speedRef_Hz`) に近いことを確認します。
- このビルド レベルでは、電流センシング、電圧センシング、回転子角度エスティメータ、ジェネレータを検証する必要があります。これは、[セクション 4.5.2](#) に示すように、HV モーター キットの PWM DAC を使用して実行できます。さらに、DATALOG モジュールを使用して、これらのセンシング波形を表示することもできます。DATALOG モジュールを使用して電流、電圧、角度信号を表示する方法の詳細については、ステップ 8 を参照してください。
- DATALOG モジュールをグラフ ツールと一緒に使用して電流センシング信号、電圧センシング信号、および角度出力をチェックする場合は、次に説明する手順に従ってください。DATALOG モジュールの詳細については、[セクション 4.5.1](#) を参照してください。注:コードを変更した後、次の各ステップの間でプロジェクトをリビルドする必要があります。
 - DATALOG モジュールを使用して位相電流をテストするには、`sys_main.c` ファイルで次のコードを設定する必要があります。注:このコードは、デフォルトでビルド レベル 2 にすでに構成されています。グラフ ツールに表示される位相電流サンプリング信号の波形は、[図 4-18](#) のとおりです。

```
dataLogObj->iptr[0] = (float32_t*) &motorVars_M1.adcData.I_A.value[0];
dataLogObj->iptr[1] = (float32_t*) &motorVars_M1.adcData.I_A.value[1];
```

- DATALOG モジュールを使用して位相電圧をテストするには、`sys_main.c` ファイルで次のコードを設定する必要があります。グラフ ツールに表示される位相電圧サンプリング信号の波形は、[図 4-19](#) のとおりです。

```
dataLogObj->iptr[2] = (float32_t*) &motorVars_M1.adcData.V_V.value[0];
```

- フォース角度ジェネレータまたはエスティメータからの角度は、[図 4-21](#) に示すように、グラフ ツールで監視できます。フォース角度ジェネレータの角度が eSMO エスティメータの推定回転子角度と非常に類似していることに注意してください。

```
dataLogObj->iptr[3] = (float32_t*) &motorVars_M1.angleFOC_rad;
```

- 変数 `motorVars_M1.overCurrent_A` の値を小さくすることで、過電流フォルト保護を検証します。過電流保護は CMPSS モジュールによって実装されています。[図 4-17](#) に示すように、`motorVars_M1.overCurrent_A` がモーター位相電流の実際値よりも小さい値に設定され、PWM 出力がディセーブルされ、`motorVars_M1.flagEnableRunAndIdentify` が 0 にクリアされると、過電流フォルトがトリガされます。
- 変数 `motorVars_M1.flagEnableRunAndIdentify` を 0 に設定し、モーターの動作を停止します。

- 完了したら、コントローラを停止して、デバッグ接続を終了できます。まずツールバーの [Halt] ボタンをクリックするか、[Target] → [Halt] をクリックして、コントローラを完全に停止します。最後に、[CPU Reset] をクリックするか、[Run] → [Reset] をクリックして、コントローラをリセットします。
- [Terminate Debug Session] ボタンをクリックするか、[Run] → [Terminate] をクリックして、CCS デバッグ セッションを終了します。
- インバータ キットの電源を切ります。

The screenshot shows the 'Expressions' window with a list of variables and their values. Red boxes highlight specific variables, and red arrows point to callout boxes explaining their functions.

| Expression | Type | Value |
|-------------------------------------|-----------------------|--|
| systemVars.flagEnableSystem | unsigned short | 1 |
| motorVars_M1.ISRCCount | unsigned int | 7844767 |
| systemVars.boardKit | enum Board_Kit_e | BOARD_BSXL3PHGAN_REVA |
| systemVars.estType | enum EST_Type_e | EST_TYPE_ESMO |
| motorVars_M1.motorState | enum MOTOR_Status_e | MOTOR_CTRL_RUN |
| motorVars_M1.estimatorMode | enum ESTIMATOR_Mo... | ESTIMATOR_MODE_ESMO |
| motorVars_M1.speedRef_Hz | float | 60.0 |
| motorVars_M1.speed_Hz | float | 59.8954887 |
| motorVars_M1.flagRunIdentAndOnLine | unsigned short | 1 |
| motorVars_M1.speedENC_Hz | unknown | member 'speedENC_Hz' not found at (m... |
| motorVars_M1.speedPLL_Hz | float | 59.9017487 |
| motorVars_M1.speedHall_Hz | unknown | member 'speedHall_Hz' not found at (mo... |
| motorVars_M1.angleFOC_rad | float | 0.866301477 |
| motorVars_M1.accelerationMax_Hzps | float | 20.0 |
| motorVars_M1.accelerationStart_Hzps | float | 10.0 |
| motorVars_M1.torque_Nm | float | 0.0 |
| motorVars_M1.flagClearFaults | unsigned short | 0 |
| motorVars_M1.faultMtrUse.all | unsigned short | 0 |
| motorVars_M1.faultMtrPrev.bit | struct FAULT_MTR_BITS | {overVoltage=0,underVoltage=0,motorOv... |
| motorSetVars_M1.dacCMPValH | unsigned short | 2978 |
| motorSetVars_M1.dacCMPValL | unsigned short | 1118 |
| motorVars_M1.adcData.VdcBus_V | float | 25.2097912 |
| motorVars_M1.Vdq_out_V.value[0] | float | 0.0 |
| motorVars_M1.Vdq_out_V.value[1] | float | 4.0 |
| motorVars_M1.maxCurrent_A | float | 6.5999999 |
| motorSetVars_M1.overCurrent_A | float | 7.5 |
| motorVars_M1.lrms_A | float[3] | {4.02242804,4.02526426,3.99164796} |
| motorVars_M1.adcData | struct HAL_ADCData_t | {VdcBus_V=25.2097912,I_A={value=[3.689... |
| motorSetVars_M1.flux_VpHz | float | 0.0399353318 |
| VsFreq_M1 | struct VS_FREQ_Obj | {maxVsMag_pu=0.660000026,Freq=0.0,Lo... |
| angleGen_M1 | struct ANGLE_GEN_Obj | {freq_Hz=60.0,angleDeltaFactor=0.000418... |
| freq_Hz | float | 60.0 |
| angleDeltaFactor | float | 0.000418879034 |
| angleDelta_rad | float | 0.0251327418 |
| angle_rad | float | 2.75123787 |

Annotations:

- Click this button to enable periodic capture of data from the microcontroller
- Check if these variables meet the board and estimator selections
- Set target speed value (Hz) to this variable
- Check if the estimation speed (Hz) is equal/close to the setting target speed (Hz)
- Set this variable value equal to 1 to start run the motor
- Means the inverter/controller has fault when run the motor if the variable value is not zero
- The sensing conversion value should be equal to the dc bus voltage
- The threshold value of the over current protection

図 4-16. ビルドレベル 2:[Expressions] ウィンドウの変数

図 4-17 に示すように、[Expression] ウィンドウで motorVars_M1.overCurrent_A の値を調整して、過電流フォルトをトリガします。

The screenshot shows a subset of variables from the Expressions window, focusing on fault-related parameters. Red boxes highlight specific variables, and red arrows point to callout boxes explaining their functions.

| | | |
|--|-----------------------|--|
| motorVars_M1.flagClearFaults | unsigned short | 0 |
| motorVars_M1.faultMtrUse.all | unsigned short | 0 |
| motorVars_M1.faultMtrPrev.bit | struct FAULT_MTR_BITS | {overVoltage=0,underVoltage=0,motorOv... |
| motorSetVars_M1.dacCMPValH | unsigned short | 2978 |
| motorSetVars_M1.dacCMPValL | unsigned short | 1118 |
| motorVars_M1.adcData.VdcBus_V | float | 25.2097912 |
| motorVars_M1.Vdq_out_V.value[0] | float | 0.0 |
| motorVars_M1.Vdq_out_V.value[1] | float | 4.0 |
| motorVars_M1.maxCurrent_A | float | 6.5999999 |
| motorSetVars_M1.overCurrent_A | float | 7.5 |
| GRP(CONTROLSS_CMPSSA3_CONTROLSS_CMPSSA3) | | |
| GRP(CONTROLSS_CMPSSA5_CONTROLSS_CMPSSA5) | | |
| GRP(CONTROLSS_CMPSSA7_CONTROLSS_CMPSSA7) | | |

Annotations:

- The value will be non-zero if there is an over-current fault
- Set the right current threshold value to verify the over current function
- The values will be non-zero if there is an over-current fault

図 4-17. ビルドレベル 2:電流保護の設定

図 4-18 に示すように、データログをグラフ ツールと一緒に使用して、モーターの 3 相センシング電流を監視します。

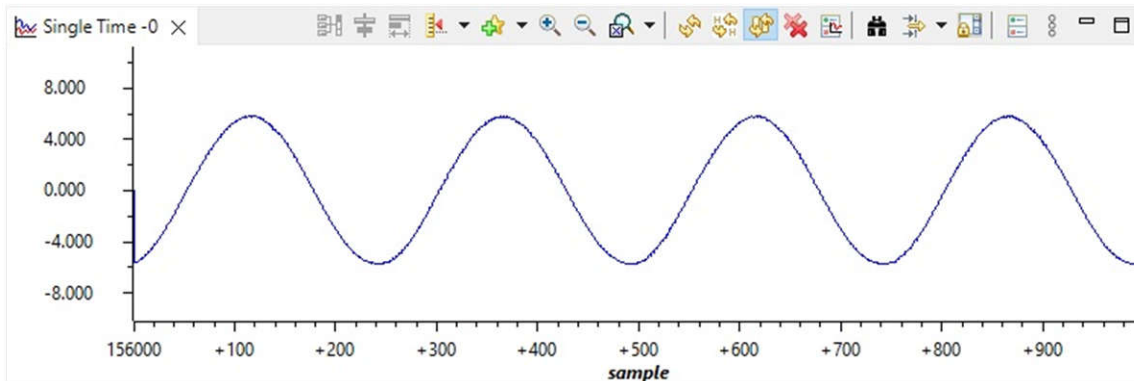


図 4-18. ビルドレベル 2: グラフ ツールによるモーター位相電流の波形

図 4-19 に示すように、データログをグラフ ツールと一緒に使用して、モーターの 3 相センシング電圧を監視します。ここで選択されている SVM モードは、最小変調の DPWM です。

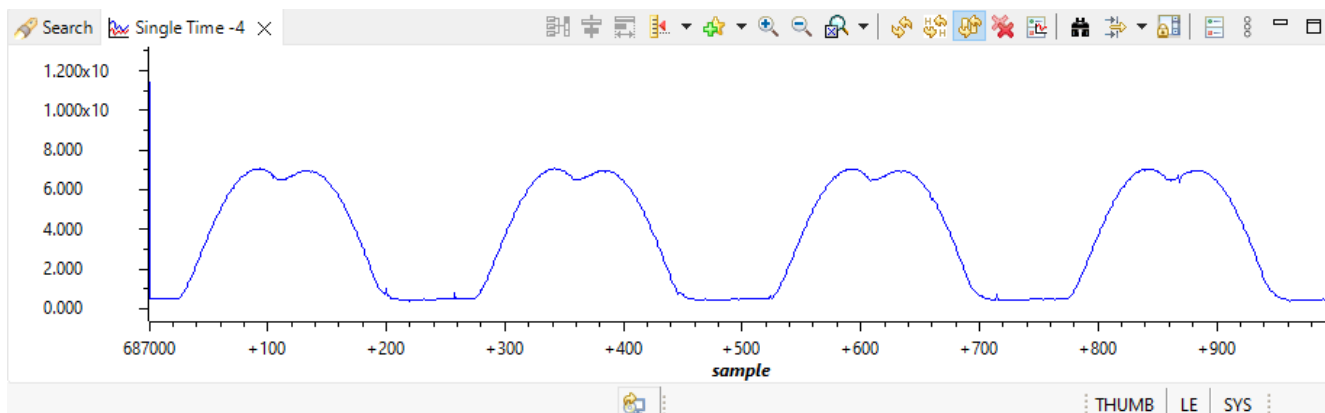


図 4-19. ビルドレベル 2: グラフ ツールによるモーター位相電圧の波形 - 最小 DPWM

図 4-20 に示すように、データログをグラフ ツールと一緒に使用して、SVPWM の一般的な変調によるモーターの 3 相センシング電圧を監視します。SVM タイプは motor1_drive.c ファイルで選択できます。

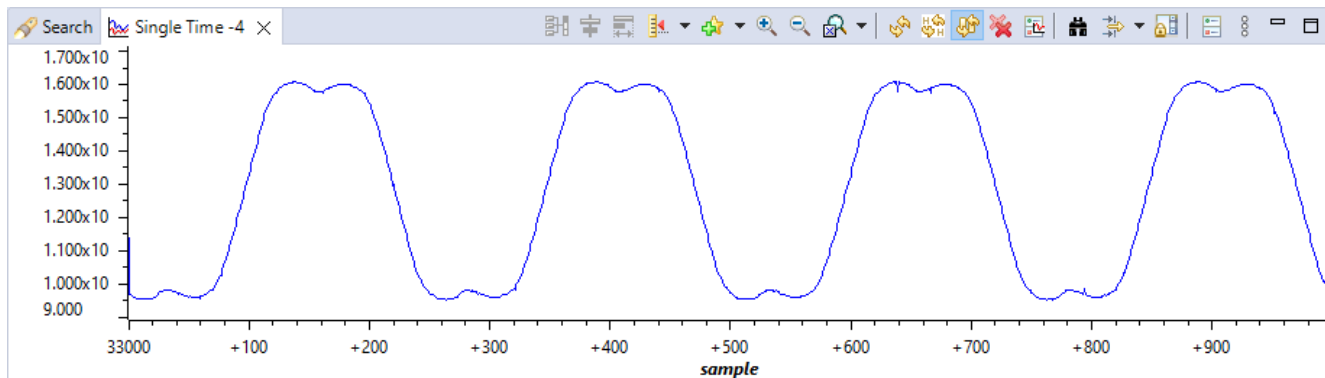


図 4-20. ビルドレベル 2: グラフ ツールによるモーター位相電圧の波形 - 一般的な SVPWM

図 4-21 に示すように、データログをグラフ ツールと一緒に使用して、角度ジェネレータからのモーター角度と eSMO エスティメータからの角度を監視します。

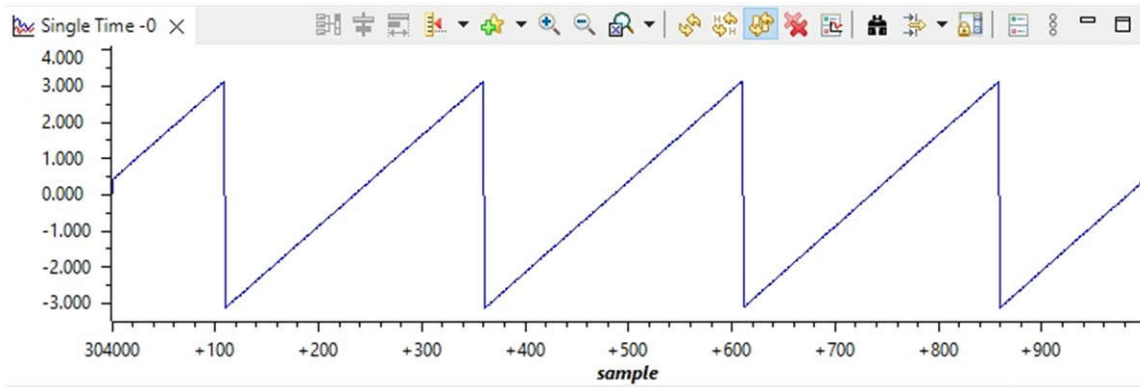


図 4-21. ビルドレベル 2: グラフツールによるモーターの回転子角度の波形

4.4.3 レベル 3 インクリメンタルビルド

このビルドレベルの目標:

- モーターの閉電流ループ動作を評価する
- 電流センシングパラメータ設定を検証する

このビルドレベルでは、モーターは *if* 制御を使用して制御され、回転子角度はランプジェネレータモジュールから生成されます。このビルドレベルのソフトウェアフローを 図 4-22 に示します。

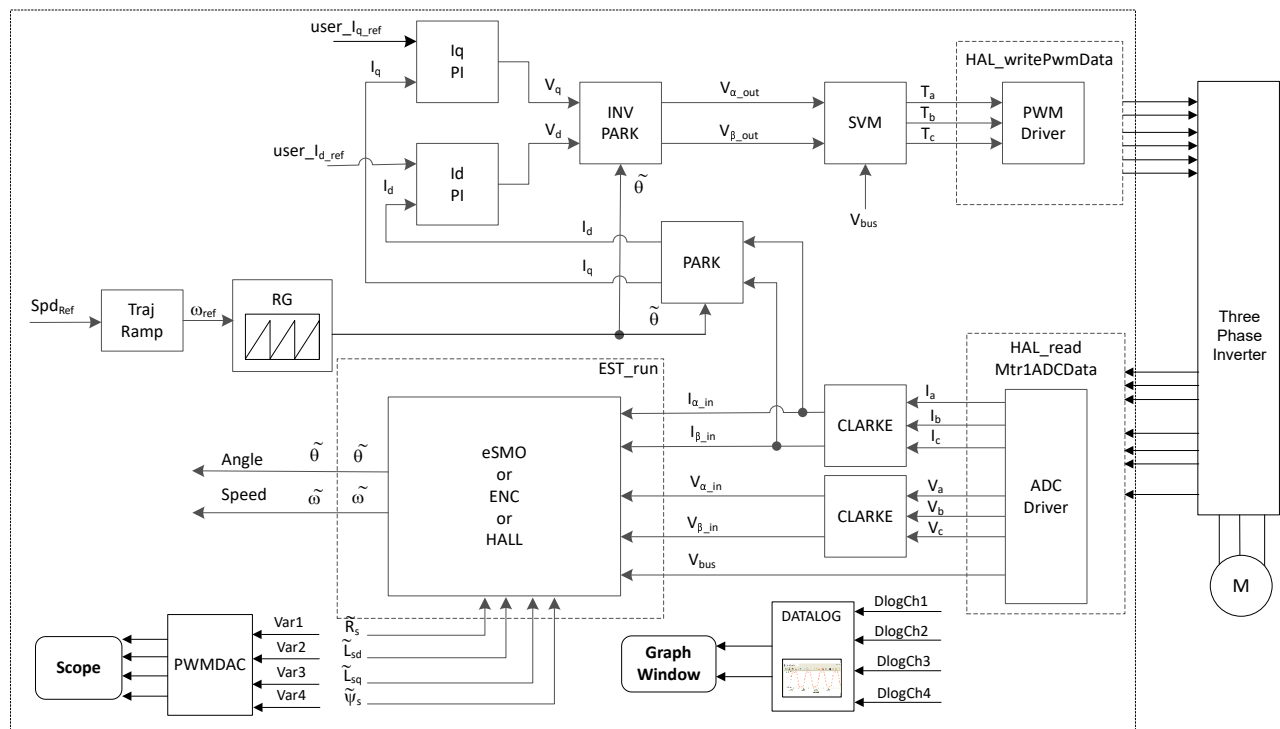


図 4-22. ビルドレベル 3 ソフトウェアのブロック図 – 電流の閉ループ制御

4.4.3.1 プロジェクトのビルドとロード

モーターを電力インバータボードの関連する端子に接続します。セクション 4.4.1.1 の操作手順に従って、`sys_settings.h` ファイルで `DMC_BUILDLEVEL` を `DMC_LEVEL_3` に設定し、プロジェクトをビルドしてロードします。

4.4.3.2 デバッグ環境設定ウィンドウ

セクション 4.4.1.2 の操作手順に従って、`universal_motor_control_level3.txt` を選択して、変数を [Expressions] ウィンドウにインポートします。図 4-23 に示すように、[Expressions] ウィンドウが表示されます。

4.4.3.3 コードの実行

1. AC 電源または DC 電源に電力を供給し、電源の出力電圧を徐々に上げて、適切な DC バス電圧を取得します。
2. [Tools] > [Arm Advanced Features] で [Data Cache Enabled] のチェックを外して、データ キャッシュをディスエーブルにします。
3. [Resume] ボタンをクリックしてプロジェクトを実行するか、[Debug] タブで [Run] → [Resume] をクリックします。一定の時間が経過したら、`systemVars.flagEnableSystem` を 1 に設定する必要があります。これによって、オフセット較正が完了します。フォルトフラグ `motorVars_M1.faultMtrUse.all` は 0 ですが、そうでない場合は、セクション 4.4.1 に示すように、電流および電圧のセンシング回路をチェックする必要があります。
4. 電流閉ループ制御でモーターが動作することを確認するには、図 4-23 に示すように、[Expressions] ウィンドウで変数 `motorVars_M1.flagEnableRunAndIdentify` を 1 に設定します。モーターは、変数 `motorVars_M1.speedRef_Hz` の設定速度で角度ジェネレータからの角度を使用して、閉ループ制御で動作します。[Expressions] ウィンドウで、`motorVars_M1.speed_Hz` の値を確認します。両変数は非常に近い値です。
5. オシロスコープのプロブを EPWMDAC (HV キット用) 出力とモーターの位相ラインに接続し、角度と電流信号、電流をプローブします。これらの波形はオシロスコープで、図 4-24 のように表示されます。[Expressions] ウィンドウで `motorVars_M1.Idq_set_A.value[1]` を変更してリファレンストルク電流を設定すると、それに応じてモーター位相電流が同じ割合で増加します。
6. 電流閉ループ制御でモーターが動作せず、過電流フォルトが表示される場合は、`motorVars_M1.adcData.current_sf` の符号と `userParams_M1.current_sf` の値がハードウェア ボードに応じて正しく設定されているかどうかを確認します。両変数の値は、`user_mtr1.h` ファイルの定義定数 `USER_M1_ADC_FULL_SCALE_CURRENT_A` に関連しています。
7. 変数 `motorVars_M1.flagEnableRunAndIdentify` を 0 に設定し、モーターの動作を停止します。
8. 完了したら、コントローラを停止して、デバッグ接続を終了できます。まずツールバーの [Suspend] ボタンをクリックするか、[Target] → [Halt] をクリックして、コントローラを完全に停止します。最後に、[CPU Reset] ボタンをクリックするか、[Run] → [Reset] をクリックして、コントローラをリセットします。
9. [Terminate Debug Session] ボタンをクリックするか、[Run] → [Terminate] をクリックして、CCS デバッグ セッションを終了します。

| Expression | Type | Value |
|---------------------------------------|-----------------------|--|
| systemVars.flagEnableSystem | unsigned short | 1 |
| motorVars_M1.ISRCCount | unsigned int | 1388274 |
| systemVars.boardKit | enum Board_Kit_e | BOARD_B5XL3PHGAN_REVA |
| systemVars.estType | enum EST_Type_e | EST_TYPE_ESMO |
| motorVars_M1.motorState | enum MOTOR_Status_e | MOTOR_CTRL_RUN |
| motorVars_M1.estimatorMode | enum ESTIMATOR_Mo... | ESTIMATOR_MODE_ESMO |
| motorSetVars_M1.flux_VpHz | float | 0.0399353318 |
| motorVars_M1.adcData.VdcBus_V | float | 25.2893791 |
| motorVars_M1.speedRef_Hz | float | 60.0 |
| motorVars_M1.speed_Hz | float | 60.0243912 |
| motorVars_M1.flagEnableRunAndIdentify | unsigned short | 1 |
| motorVars_M1.flagRunIdentAndOnLine | unsigned short | 1 |
| motorVars_M1.flagEnableForceAngle | unsigned short | 1 |
| motorVars_M1.enableSpeedCtrl | unsigned short | 1 |
| motorVars_M1.angleFOC_rad | float | -0.303307295 |
| motorVars_M1.accelerationMax_Hzps | float | 20.0 |
| motorVars_M1.accelerationStart_Hzps | float | 10.0 |
| motorVars_M1.flagClearFaults | unsigned short | 0 |
| motorVars_M1.faultMtrUse.all | unsigned short | 0 |
| motorVars_M1.faultMtrPrev.bit | struct FAULT_MTR_BITS | {overVoltage=0,underVoltage=0,motorOv... |
| motorSetVars_M1.dacCMPValH | unsigned short | 2978 |
| motorSetVars_M1.dacCMPValL | unsigned short | 1118 |
| motorSetVars_M1.overCurrent_A | float | 7.5 |
| motorVars_M1.startCurrent_A | float | 3.5 |
| motorVars_M1.maxCurrent_A | float | 6.5999999 |
| motorVars_M1.Idq_set_A.value[0] | float | 0.0 |
| motorVars_M1.Idq_set_A.value[1] | float | 3.5 |
| motorVars_M1.IdqRef_A.value[0] | float | 0.0 |
| motorVars_M1.IdqRef_A.value[1] | float | 3.5 |
| motorSetVars_M1.Kp_Id | float | 0.239317492 |
| motorSetVars_M1.Ki_Id | float | 0.0344771557 |
| motorSetVars_M1.Kp_Iq | float | 0.239317492 |
| motorSetVars_M1.Ki_Iq | float | 0.0344771557 |
| motorSetVars_M1.Kp_spd | float | 0.011634578 |
| motorSetVars_M1.Ki_spd | float | 0.00209439523 |
| pi_spd_M1 | struct PI_Obj | {Kp=0.011634578,Ki=0.00209439523,Umax... |
| pi_Iq_M1 | struct PI_Obj | {Kp=0.239317492,Ki=0.0344771557,Umax... |
| pi_Id_M1 | struct PI_Obj | {Kp=0.239317492,Ki=0.0344771557,Umax... |
| motorVars_M1.adcData | struct HAL_ADCData_t | {VdcBus_V=25.3291741_A={value=[-3.158... |
| motorVars_M1.Irms_A | float[3] | [2.46620107,2.47039008,2.47602439] |

図 4-23. ビルドレベル 3:[Expressions] ウィンドウの変数

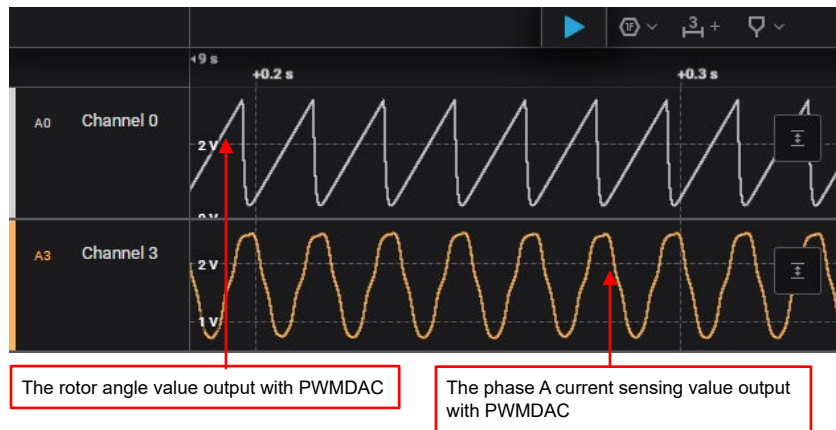


図 4-24. ビルドレベル 3:EPMWDAC によるモーターの回転子角度と位相電流波形の監視

4.4.4 レベル 4 インクリメンタル ビルド

このビルドレベルの目標:

- eSMO ベースのセンサレス FOC、エンコーダ ベースのセンサ付き FOC、ホール ベースのセンサ付き FOC を使用して、モータードライブ全体を評価する
- 弱め界磁制御、フライング スタート、MTPA、ブレーキなどの追加機能を評価する

このビルドレベルでは、回転子角度が eSMO モジュール、エンコーダ モジュール、またはホール センサー モジュールから取得されるモーターの場合、外側の速度ループは内側の電流ループで閉じられます。このビルドレベルのソフトウェアフローを 図 4-25 に示します。

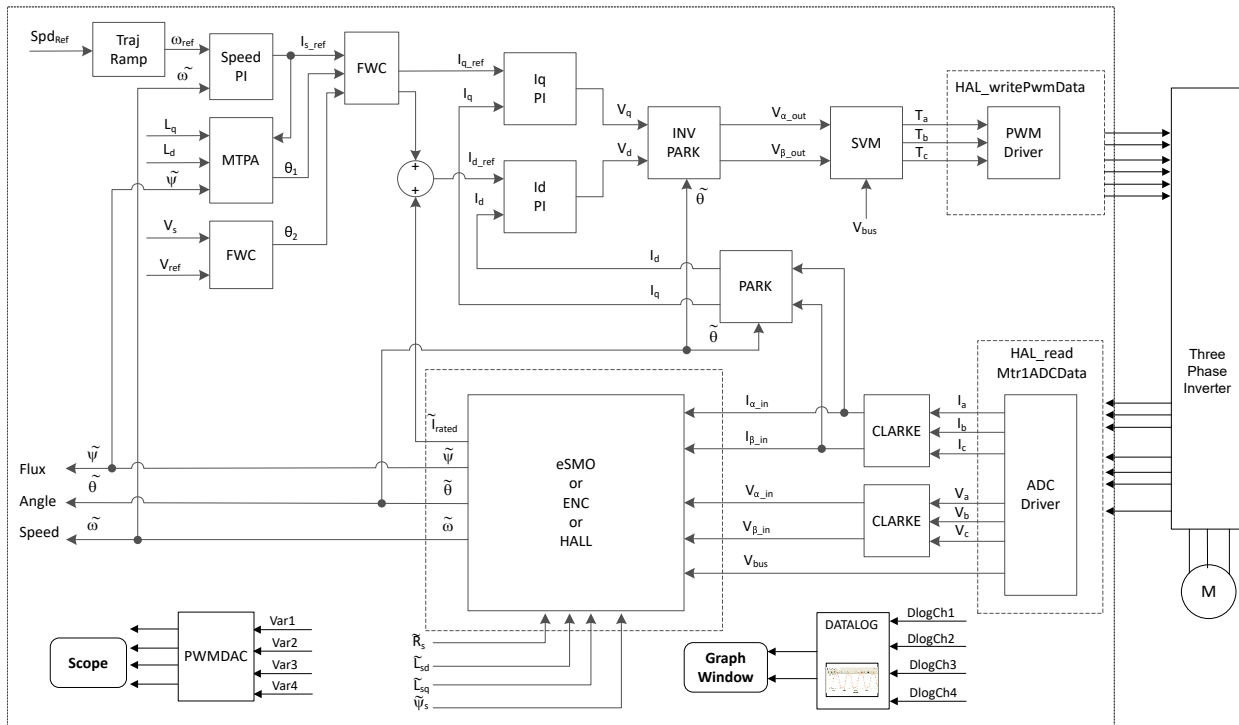


図 4-25. ビルドレベル 4 ソフトウェアのブロック図 – 速度と電流の閉ループ制御

4.4.4.1 プロジェクトのビルドとロード

モーターを電力インバータボードの関連する端子に接続します。セクション 4.4.1.1 の操作手順に従って、sys_settings.h ファイルで DMC_BUILDLEVEL を DMC_LEVEL_4 に設定し、プロジェクトをビルドしてロードします。

4.4.4.2 デバッグ環境設定ウィンドウ

セクション 4.4.1.2 の操作手順に従って、universal_motor_control_level4.txt を選択して、変数を [Expressions] ウィンドウにインポートします。図 4-26 に示すように、[Expressions] ウィンドウが表示されます。

4.4.4.3 コードの実行

1. AC 電源または DC 電源に電力を供給し、電源の出力電圧を徐々に上げて、適切な DC バス電圧を取得します。
2. 以下のサンプルコードに示すように、必要なモーターパラメータは user_mtr1.h ヘッダーファイルで定義されている必要があります。

```
#define USER_MOTOR1_TYPE MOTOR_TYPE_PM
#define USER_MOTOR1_NUM_POLE_PAIRS (4)
#define USER_MOTOR1_Rr_Ohm (NULL)
#define USER_MOTOR1_Rs_Ohm (0.38157931f)
#define USER_MOTOR1_Ls_d_H (0.000188295482f)
#define USER_MOTOR1_Ls_q_H (0.000188295482f)
#define USER_MOTOR1_RATED_FLUX_VpHz (0.0396642499f)
```

3. プロジェクトをビルドしてコードをコントローラにロードし、[Tools] > [ARM Advanced Features] で [Data Cache Enabled] のチェックを外してデータ キャッシュを無効にし、[Resume] ボタンをクリックしてプロジェクトを実行するか、[Debug] タブで [Run] → [Resume] をクリックします。一定の時間が経過したら、systemVars.flagEnableSystem は 1 に設定されます。これは、オフセット較正が完了し、導通時には電力リレーがオンになったことを意味します。フォルトフラグ motorVars_M1.faultMtrUse.all は 0 になります、そうでない場合は、セクション 4.4.1 に示すように、電流および電圧のセンシング回路をチェックする必要があります。
4. 図 4-26 に示すように、[Expressions] ウィンドウで変数 motorVars_M1.flagEnableRunAndIdentify を 1 に設定します。
5. モーターを始動した後、以下の手順に従ってください。
 - a. 目標速度値を変数 motorVars_M1.speedRef_Hz に設定し、モーター シャフトの速度がどのように設定速度を追従するかを注意深く観察します。

- b. 加速度を変更するには、変数 `motorVars_M1.accelerationMax_Hzps` に異なる加速度値を入力します。
- c. PWM DAC モジュールを使用して、[セクション 4.5.2](#) に示すように監視変数を表示します。モーターの角度と電流波形を [図 4-27](#) に示します。
6. FOC システムの電流コントローラのデフォルトの比例ゲイン (Kp) と積分ゲイン (Ki) は、関数 `setupControllers()` で計算されます。`setupControllers()` が呼び出されると、グローバル変数 `motorSetVars_M1.Kp_Id`、`motorSetVars_M1.Ki_Id`、`motorSetVars_M1.Kp_Iq`、`motorSetVars_M1.Ki_Iq` が、新しく計算された Kp ゲインおよび Ki ゲインで初期化されます。[図 4-26](#) に示すように、これら 4 つの変数の Kp と Ki の値を、予想される電流制御の帯域幅と応答が得られるように、電流コントローラの [Expressions] ウォッチ ウィンドウで調整します。Kp ゲインにより、モーターの固定子の極を打ち消すゼロが生成され、簡単に計算できます。Ki ゲインにより、電流コントローラ - モーター システムの帯域幅が調整されます。速度制御システムに一定のダンピングが必要な場合、電流コントローラの Kp ゲインは速度制御システムの時定数に関連しています。
7. 変数 `motorVars_M1.flagEnableRunAndIdentify` を 0 に設定し、モーターの動作を停止します。
8. 完了したら、コントローラを停止して、デバッグ接続を終了できます。まずツールバーの [Suspend] ボタンをクリックするか、[Target] → [Halt] をクリックして、コントローラを完全に停止します。最後に、[CPU Reset] ボタンをクリックするか、[Run] → [Reset] をクリックして、コントローラをリセットします。
9. [Terminate Debug Session] ボタンをクリックするか、[Run] → [Terminate] をクリックして、CCS デバッグ セッションを終了します。

The screenshot shows the 'Expressions' window in CCS, displaying a list of motor control variables. Red boxes highlight specific variables, and red callout boxes provide explanations for each. A red arrow points to a button in the toolbar.

| Expression | Type | Value |
|---------------------------------------|-----------------------|--|
| systemVars.flagEnableSystem | unsigned short | 1 |
| motorVars_M1.ISRCCount | unsigned int | 3994207 |
| systemVars.boardKit | enum Board_Kit_e | BOARD_BSXL3PHGAN_REVA |
| systemVars.estType | enum EST_Type_e | EST_TYPE_ESMO |
| systemVars.currentSenseType | enum CURRENTSEN_T... | CURSEN_TYPE_INLINE_SHUNT |
| motorVars_M1.speed_Hz | float | 59.8866501 |
| motorVars_M1.speedRef_Hz | float | 60.0 |
| motorVars_M1.flagEnableRunAndIdentify | unsigned short | 1 |
| motorVars_M1.flagRunIdentAndOnLine | unsigned short | 1 |
| motorVars_M1.accelerationMax_Hzps | float | 20.0 |
| motorVars_M1.accelerationStart_Hzps | float | 10.0 |
| motorVars_M1.flagEnableForceAngle | unsigned short | 1 |
| motorVars_M1.flagMotorIdentified | unsigned short | 1 |
| motorVars_M1.motorState | enum MOTOR_Status_e | MOTOR_CTRL_RUN |
| motorVars_M1.estimatorMode | enum ESTIMATOR Mo... | ESTIMATOR_MODE_ESMO |
| motorVars_M1.adcData.VdcBus_V | float | 25.2893791 |
| motorSetVars_M1.Kp_Id | float | 0.239317492 |
| motorSetVars_M1.Ki_Id | float | 0.0344771557 |
| motorSetVars_M1.Kp_Iq | float | 0.239317492 |
| motorSetVars_M1.Ki_Iq | float | 0.0344771557 |
| motorSetVars_M1.Kp_spd | float | 0.011624578 |
| motorSetVars_M1.Ki_spd | float | 0.00209439523 |
| motorVars_M1.flagClearFaults | unsigned short | 0 |
| motorVars_M1.faultMtrUse.all | unsigned short | 0 |
| motorVars_M1.faultMtrNow.all | unsigned short | 128 |
| motorVars_M1.faultMtrPrev.bit | struct FAULT_MTR_BITS | {overVoltage=0,underVoltage=0,motorOv... |
| motorSetVars_M1.dacCMPValH | unsigned short | 2978 |
| motorSetVars_M1.dacCMPValL | unsigned short | 1118 |
| motorSetVars_M1.overCurrent_A | float | 7.5 |
| motorVars_M1.speedPLL_Hz | float | 60.1450272 |
| motorVars_M1.speedENC_Hz | unknown | member 'speedENC Hz' not found at (m... |
| motorVars_M1.speedHall_Hz | unknown | member 'speedHall Hz' not found at (m... |
| motorVars_M1.angleFOC_rad | float | -2.26526546 |
| motorVars_M1.anglePLL_rad | float | -1.95956504 |
| userParams_M1.maxVsMag_V | float | 15.8400002 |
| userParams_M1.maxVsMag_pu | float | 0.660000026 |
| motorVars_M1.Vs_V | float | 2.65994358 |
| motorVars_M1.VsRef_V | float | 31.046402 |
| motorVars_M1.VsRef_pu | float | 0.646800041 |
| motorVars_M1.startCurrent_A | float | 3.5 |
| motorVars_M1.alignCurrent_A | float | 1.5 |
| motorVars_M1.maxCurrent_A | float | 6.5999999 |
| motorVars_M1.Is_A | float | 0.153146818 |

Annotations:

- Click this button to enable periodic capture of data from the microcontroller
- Supporting estimator
- Estimation feedback speed (Hz)
- Set target speed value (Hz) to this variable
- Set this variable value equal to 1 to start motor
- Motor operation state
- Using estimator
- Tune these Kp or Ki of current and speed regulators to achieve the required response
- The threshold value of the over current protection
- Measured speed when encoder is enabled
- Measured speed when Hall sensor is enabled

図 4-26. ビルド レベル 4: [Expressions] ウィンドウの変数

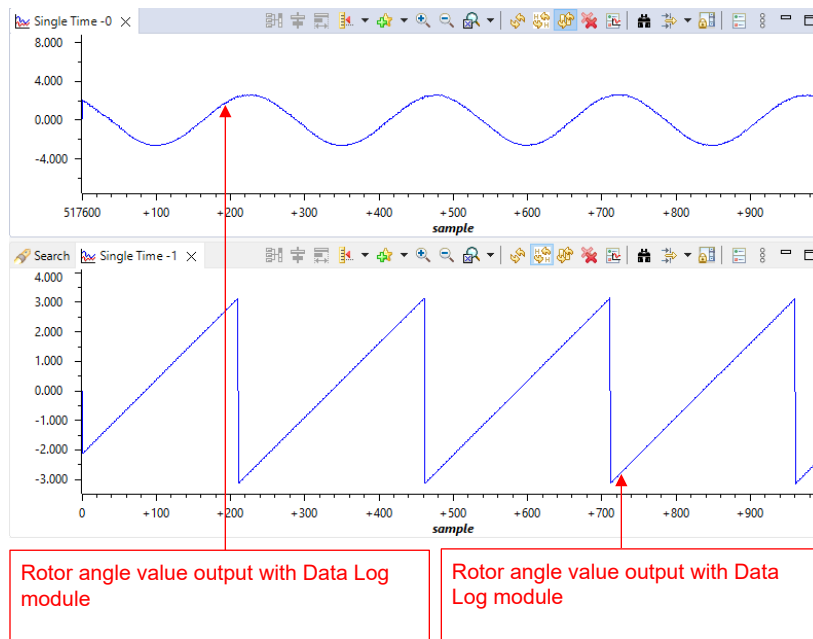


図 4-27. ビルド レベル 4: 順方向動作時の eSMO 波形による位相電流と回転子角度

セクション 4.2.2 に示すように、このプロジェクトでは複数の FOC アルゴリズムをサポートすることが可能であり、モーター制御に 1 つ (eSMO、ホール、エンコーダ) のアルゴリズムまたは 2 つのアルゴリズム (eSMO + エンコーダ) を使用できます。

セクション 4.2.1 に示すように、プロジェクト プロパティに MOTOR1_ESMO と MOTOR1_ENC という事前定義名を追加することにより、eSMO エスティメータとエンコーダ エスティメータをプロジェクトに同時に実装できます。上記の操作手順に従って、プロジェクトをリビルド、ロード、実行します。

- `systemVars.estType` の値は `EST_TYPE_ESMO_ENC` に等しく、これはこのプロジェクトでは eSMO エスティメータとエンコーダ エスティメータが有効になっていることを意味します。
- `motorVars_M1.estimatorMode` が `ESTIMATOR_MODE_ESMO` に等しいのは、eSMO エスティメータがセンサレス FOC に使用されていることを意味し、`ESTIMATOR_MODE_ENC` に等しいのは、エンコーダ エスティメータがセンサ付き FOC に使用されていることを意味します。
- eSMO およびエンコーダから推定された回転子角度を図 4-28 に示します。`motorVars_M1.speedRef_Hz` を正の値に設定することで、モーターは eSMO により順方向で動作します。
- この値を `ESTIMATOR_MODE_ENC` に変更すれば、センサ付き FOC 用のエンコーダ エスティメータを選択できます。また、この値を変更すれば、使用するエスティメータをその場で切り替えることもできます。

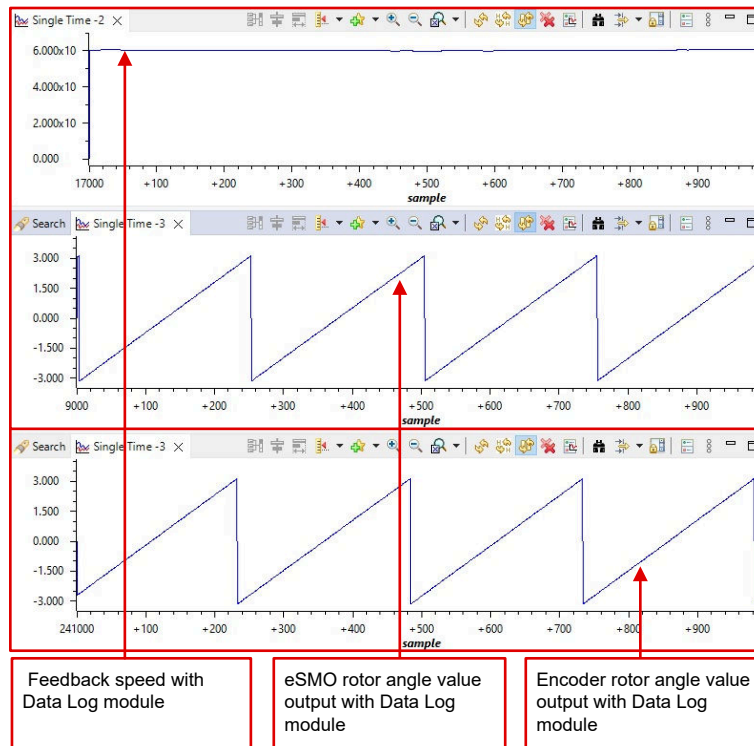


図 4-28. ビルドレベル 4:eSMO とエンコーダによる回転子角度、順方向動作時の位相電流波形

4.5 モーター制御プロジェクトへのその他の追加の機能

4.5.1 DATALOG 機能の使用

図 4-29 に示すように、DATALOG モジュールでは、ユーザーが選択可能なソフトウェア変数 (デフォルトでは 4 つの変数) のリアルタイム値がテキサス・インスツルメンツの MCU に搭載されているデータ RAM に保存されます。これら 4 つの変数は、モジュール入力を 4 つの変数のアドレスに構成することで選択されます。4 つの RAM バッファ位置の開始アドレスは、`&((datalog).datalogBuff)[0]`、`&((datalog).datalogBuff)[1]`、`&((datalog).datalogBuff)[2]`、`&((datalog).datalogBuff)[3]` となります。これらのデータログ バッファは、グラフに表示できる値トリガー データを含む大きな配列です。データログ プリスケアラは構成可能であるため、データ ログ機能により、プリスケアラのサンプリングごとに 1 つのサンプルのみをログに記録できます。データログ バッファの数、バッファ サイズ、データタイプは、`datalog_input.h` ファイルで選択できます。

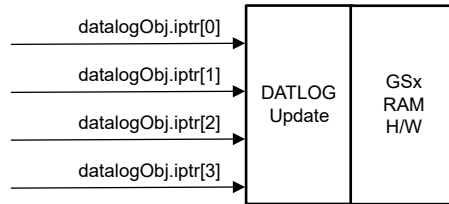


図 4-29. DATALOG モジュールのブロック図

データログ機能をイネーブルにするには、図 4-2 に示すように、プロジェクト プロパティに事前定義シンボル `DATALOG_EN` を追加する必要があります。

以下のコードは、1 つの DATALOG のオブジェクトとハンドルの宣言を示しています。このコードは `datalog.c` ファイルにあります。

```
__attribute__((section("datalog_data"))) DATALOG_Obj datalog;
DATALOG_Handle datalogHandle; //!< the handle for the Datalog object
```

これにより、DATALOG オブジェクトはメモリの `datalog_data` セクションに格納されます。このセクションは TCM または OCRAM のいずれかになります。一般的に、TCM は容量が限られており、ソフトウェアの時間的制約がある部分で必要とされるため、OCRAM の使用をお勧めします。CCS12.6 で OCRAM にデータをロギングできるように、データ キャッシュを無効にします。データキャッシュを無効にするには、図 4-30 に示すように、[Tools] > [ARM Advanced Features] で [Data Cache Enabled] のチェックを外す必要があります。

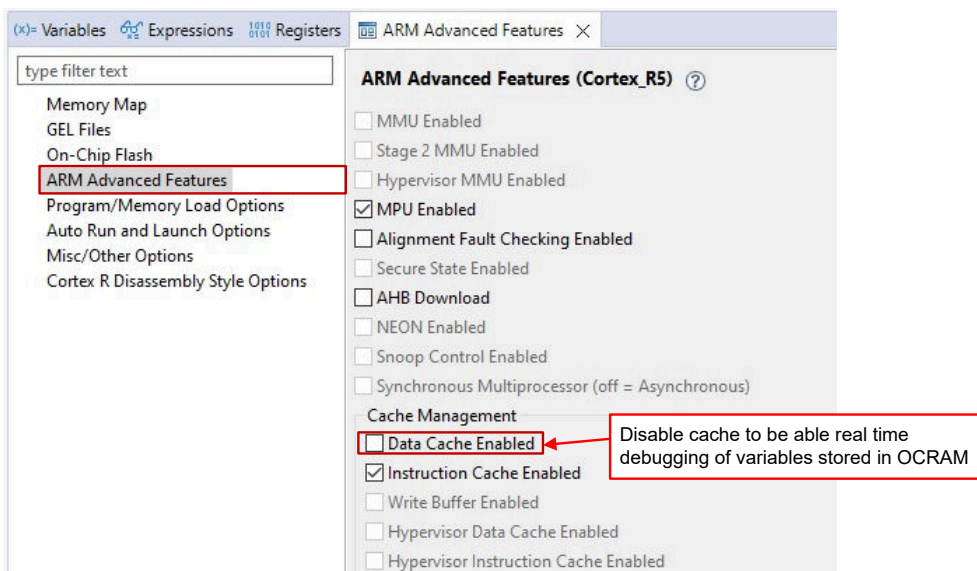


図 4-30. リアルタイム デバッグのためにデータ キャッシュを無効にする

以下のコードは、**DATALOG** のオブジェクト、ハンドル、およびパラメータの初期化と設定を示しています。このコードは **sys_main.c** ファイルにあります。

```
// Initialize Datalog
datalogHandle = DATALOG_init(&datalog, sizeof(datalog), manual, 0, 1);
DATALOG_Obj *datalogObj = (DATALOG_Obj *)datalogHandle;
```

以下のコードは、変数のアドレスを指すようにモジュール入力の構成を示しています。**DATALOG** モジュールの入力は、ビルドレベルに応じて異なるシステム変数を指します。このコードは **sys_main.c** ファイルにあります。

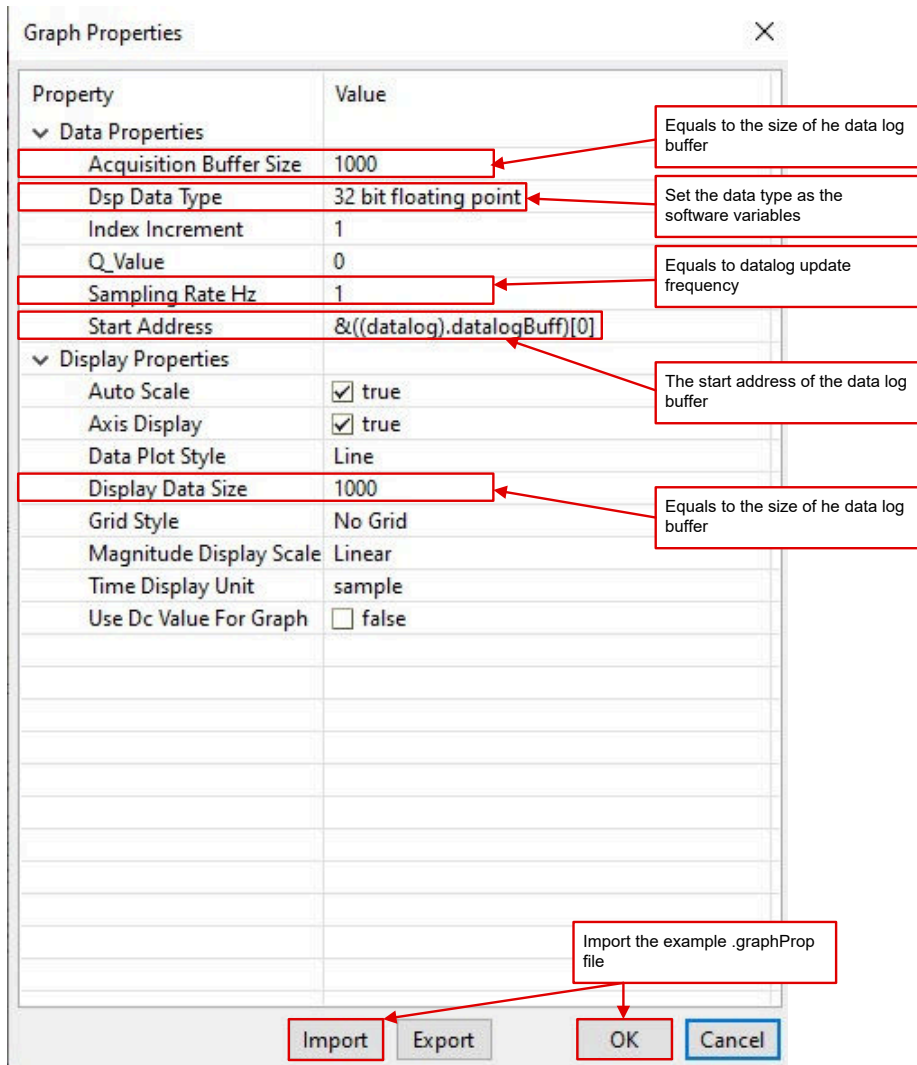
```
datalogObj->iptr[0] = (float32_t*) &motorVars_M1.adcData.V_V.value[0];
datalogObj->iptr[1] = (float32_t*) &motorVars_M1.adcData.I_A.value[0];
datalogObj->iptr[2] = (float32_t*) &motorVars_M1.adcData.I_A.value[1];
datalogObj->iptr[3] = (float32_t*) &motorVars_M1.angleFOC_rad;
```

以下のコードは、**motor1ctrlISR()** 割り込みの実行中に、データログバッファが新しいデータで定期的に更新されることを示しています。このコードは **motor1_drive.c** ファイルにあります。

```
#if defined(DATALOG_EN)
DATALOG_update(datalogHandle);
#endif // DATALOG_EN
```

DATALOG モジュールはグラフツールと一緒に使用され、変数を視覚的に表示して検査し、システム性能を判定する手段を提供します。**グラフツール**は **CCS** で使用可能であり、データの配列をさまざまな種類のグラフで表示できます。データの配列は、さまざまな形式でデバイスのメモリに保存されます。

プロジェクトがデバッグモードの間に、[図 4-31](#) に示すように、データログバッファをプロットするための時間グラフウィンドウを開いて設定します。または、プロジェクトフォルダにあるグラフ構成ファイルをインポートすることもできます。インポートするには、[Tools] -> [Graph] -> [Single Time...] をクリックして [Import] を選択し、次の場所 `<workspace>\universal_motorcontrol_am263x_r5fss0-0_nortos_ti-arm-c1ang\src_control\debug\` を参照して、**datalog.graphProp** ファイルを選択します。[OK] をクリックすると、[Debug] パースペクティブにグラフが追加されます。[Graph] タブの左上隅にある [Continuous Refresh] ボタンをクリックします。



| Property | Value |
|-------------------------|--|
| ▼ Data Properties | |
| Acquisition Buffer Size | 1000 |
| Dsp Data Type | 32 bit floating point |
| Index Increment | 1 |
| Q Value | 0 |
| Sampling Rate Hz | 1 |
| Start Address | &((datalog).datalogBuff)[0] |
| ▼ Display Properties | |
| Auto Scale | <input checked="" type="checkbox"/> true |
| Axis Display | <input checked="" type="checkbox"/> true |
| Data Plot Style | Line |
| Display Data Size | 1000 |
| Grid Style | No Grid |
| Magnitude Display Scale | Linear |
| Time Display Unit | sample |
| Use Dc Value For Graph | <input type="checkbox"/> false |

Annotations:

- Acquisition Buffer Size: Equals to the size of the data log buffer
- Dsp Data Type: Set the data type as the software variables
- Sampling Rate Hz: Equals to datalog update frequency
- Start Address: The start address of the data log buffer
- Display Data Size: Equals to the size of the data log buffer
- Import button: Import the example .graphProp file

図 4-31. グラフ ウィンドウの設定

4.5.2 PWMDAC 機能の使用

PWMDAC モジュールは、[図 4-32](#) に示すように、ePWM 5A、5B、6A、6B を使用してソフトウェア変数を PWM 信号に変換します。PWMDAC モジュールは、RC フィルタが適用された追加 PWM 出力がボード上にあるため、高電圧キット (TMDSHVMTRINSPIN) のみでサポートされます。PWMDAC モジュールが PWMDAC モジュールをサポートしていないモータードライバボードと一緒に使用される場合、PWM 信号はテキサス・インスツルメンツの LaunchPad 上の予備の PWM にルーティングされ、ユーザーは PWMDAC デザインを利用するためにこれらのピンに RC フィルタを追加する必要があります。

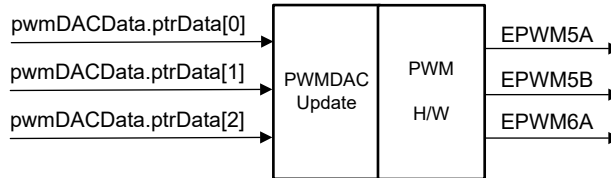


図 4-32. PWMDAC モジュールのブロック図

PWMDAC モジュールを使用すると、外部ローパス フィルタを介した関連するピンの出力で、変数で表される信号を確認できます。そのため、[図 4-33](#) に示すように、実際の信号波形を表示するには、外部ローパス フィルタが必要です。(1 次) RC ローパス フィルタは、実際の低周波信号に組み込まれている高周波成分をフィルタリングするために使用されます。R 値と C 値を選択するには、次の式に示すように、時定数をカットオフ周波数 (f_c) で表すことができます。

$$\tau = RC = \frac{1}{2\pi f_c} \quad (62)$$

$$f_c = 2\pi RC \quad (63)$$

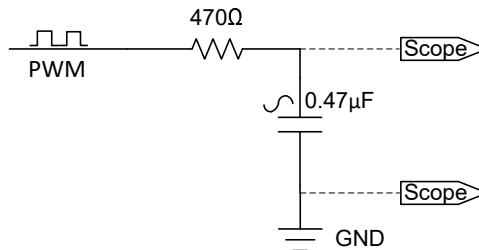


図 4-33. PWM ピンに接続した外部 RC ローパス フィルタ

ePWM DAC 機能を有効にするには、[図 4-2](#) に示すように、事前定義されたシンボル EPWMDAC_MODE をプロジェクトプロパティに追加する必要があります。

次のコードは、PWMDAC オブジェクトの宣言を示しています。このコードは sys_main.c ファイルにあります。

```
#if defined(EPWMDAC_MODE)
#if defined(HVMTRPFC_REV1P1)
__attribute__((section("sys_data"))) HAL_PWMDACData_t pwmDACData;
// HVMTRPFC_REV1P1
#else
#error EPWMDAC is not supported on this kit!
#endif // !HVMTRPFC_REV1P1
#endif // EPWMDAC_MODE
```

以下のコードは、PWMDAC のオブジェクト、ハンドル、およびパラメータの初期化と設定を示しています。4 つのモジュール入力である ptrData[0]、ptrData[1]、ptrData[2]、ptrData[3] は、4 つの変数のアドレスを指すように構成されています。PWMDAC モジュールの入力は、ビルドレベルに応じて異なるシステム変数を指します。このコードは sys_main.c ファイルにあります。

```
// set DAC parameters
pwmDACData.periodMax =
    PWMDAC_getPeriod(halHandle->pwmDACHandle[PWMDAC_NUMBER_1]);
```



```

pwmDACData.ptrData[0] = &motorVars_M1.angleFOC_rad;           // PWMDAC1
pwmDACData.ptrData[1] = &motorVars_M1.speedAbs_Hz;           // PWMDAC2
pwmDACData.ptrData[2] = &motorVars_M1.speedAbs_Hz;           // PWMDAC3
pwmDACData.ptrData[3] = &motorVars_M1.adcData.I_A.value[1];   // PWMDAC4

pwmDACData.offset[0] = 0.5f; // PWMDAC1
pwmDACData.offset[1] = 0.0f; // PWMDAC2
pwmDACData.offset[1] = 0.0f; // PWMDAC3
pwmDACData.offset[3] = 0.5f; // PWMDAC4

pwmDACData.gain[0] = 1.0f / MATH_TWO_PI; // PWMDAC1
pwmDACData.gain[1] = 1.0f / USER_MOTOR1_FREQ_MAX_HZ; // PWMDAC2
pwmDACData.gain[2] = 1.0f / USER_MOTOR1_FREQ_MAX_HZ; // PWMDAC3
pwmDACData.gain[3] = 2.0f / USER_M1_ADC_FULL_SCALE_CURRENT_A; // PWMDAC4
    
```

以下のコードは、**motor1ctrlISR()** 割り込みの実行中に、PWM 出力が新しいデータで更新されることを示しています。このコードは **motor1_drive.c** ファイルにあります。

```

// connect inputs of the PWMDAC module.
HAL_writePWMDACData(handle, &pwmDACData);
    
```

4.5.3 CAN 機能の追加

CAN 機能をラボ プロジェクトに追加することで、スタート/ストップ コマンドを送信し、フィードバックの実行状態を取得するための通信バスを提供することができます。これを利用するには、[図 4-2](#) に示すように、プロジェクトビルドプロパティで事前定義シンボル **CMD_CAN** を有効にします。**PCAN-View** は、CAN データトラフィックの監視、送信、記録を行うために使用されます。**motor_common.h** ファイルでは、異なる種類のコマンドメッセージを定義できます。このプロジェクトでは例として、送信者は開始コマンドを指定し、目標速度の値を定義します。受信者は、目標速度を含むメッセージを受信します。

キットの **LaunchPad** または **EVM** のどちらを使用するかによっては、特定のピン多重化設定が必要であることを注意してください。これらの構成は、次のコードで説明するように、**mcanEnableTransceiver** 関数と **tca6416ConfigOutput** 関数を使用して実現されます。

```

#if defined(AM263_CC)
void tca6416ConfigOutput(uint16_t port, uint16_t pin, uint16_t level);
#endif // AM263_CC
    
```

```

#if defined(CMD_CAN)
#if defined(AM263_LP)

void mcanEnableTransceiver(void)
{
    uint32_t    gpioBaseAddr, pinNum;

    gpioBaseAddr = (uint32_t)AddrTranslateP_getLocalAddr(MCAN_ENABLE_BASE_ADDR);
    pinNum       = MCAN_ENABLE_PIN;

    GPIO_setDirMode(gpioBaseAddr, pinNum, GPIO_DIRECTION_OUTPUT);

    GPIO_pinwriteLow(gpioBaseAddr, pinNum);
}
#endif // AM263_LP

#if defined(AM263_CC)
/* ===== */
/*                               Macros & Typedefs                               */
/* ===== */

/* Input status register */
#define TCA6416_REG_INPUT0    ((UInt8) 0x00U)
#define TCA6416_REG_INPUT1    ((UInt8) 0x01U)

/* Output register to change state of output BIT set to 1, output set HIGH */
#define TCA6416_REG_OUTPUT0    ((uint8_t) 0x02U)
#define TCA6416_REG_OUTPUT1    ((uint8_t) 0x03U)

/* Configuration register. BIT = '1' sets port to input, BIT = '0' sets
 * port to output */
    
```

```

#define TCA6416_REG_CONFIG0          ((uint8_t) 0x06U)
#define TCA6416_REG_CONFIG1        ((uint8_t) 0x07U)

/* ===== */
/*                               Function Declarations                               */
/* ===== */
static void SetupI2CTransfer(I2C_Handle handle, uint32_t targetAddr,
                             uint8_t *writeData, uint32_t numWriteBytes,
                             uint8_t *readData, uint32_t numReadBytes);

void mcanEnableTransceiver(void)
{
    I2C_Handle    i2cHandle;
    uint8_t       dataToSlave[4];

    i2cHandle = gI2cHandle[CONFIG_I2C0];
    dataToSlave[0] = TCA6416_REG_CONFIG0;
    dataToSlave[1] = 0x0U;
    SetupI2CTransfer(i2cHandle, 0x20, &dataToSlave[0], 1, &dataToSlave[1], 1);
    /* set the P00 to 0 make them output ports. */
    dataToSlave[1] &= ~(0x1U);
    SetupI2CTransfer(i2cHandle, 0x20, &dataToSlave[0], 2, NULL, 0);

    /* Get the port values. */
    dataToSlave[0] = TCA6416_REG_INPUT0;
    dataToSlave[1] = 0x0U;
    SetupI2CTransfer(i2cHandle, 0x20, &dataToSlave[0], 1, &dataToSlave[1], 1);

    /* Set P10 and P11 to 0.
       */
    dataToSlave[0] = TCA6416_REG_OUTPUT0;
    dataToSlave[1] &= ~(0x1);
    SetupI2CTransfer(i2cHandle, 0x20, &dataToSlave[0], 2, NULL, 0);
}

static void SetupI2CTransfer(I2C_Handle handle, uint32_t targetAddr,
                             uint8_t *writeData, uint32_t numWriteBytes,
                             uint8_t *readData, uint32_t numReadBytes)
{
    int32_t status;
    I2C_Transaction i2cTransaction;

    /* Enable Transceiver */
    I2C_Transaction_init(&i2cTransaction);
    i2cTransaction.targetAddress = targetAddr;
    i2cTransaction.writeBuf = (uint8_t *)&writeData[0];
    i2cTransaction.writeCount = numWriteBytes;
    i2cTransaction.readBuf = (uint8_t *)&readData[0];
    i2cTransaction.readCount = numReadBytes;
    status = I2C_transfer(handle, &i2cTransaction);
    DebugP_assert(SystemP_SUCCESS == status);
}

#endif // AM263_CC
#endif // CMD_CAN

```

PCAN-View を起動したら、[図 4-34](#) のように CAN アダプタのセットアップを行います。

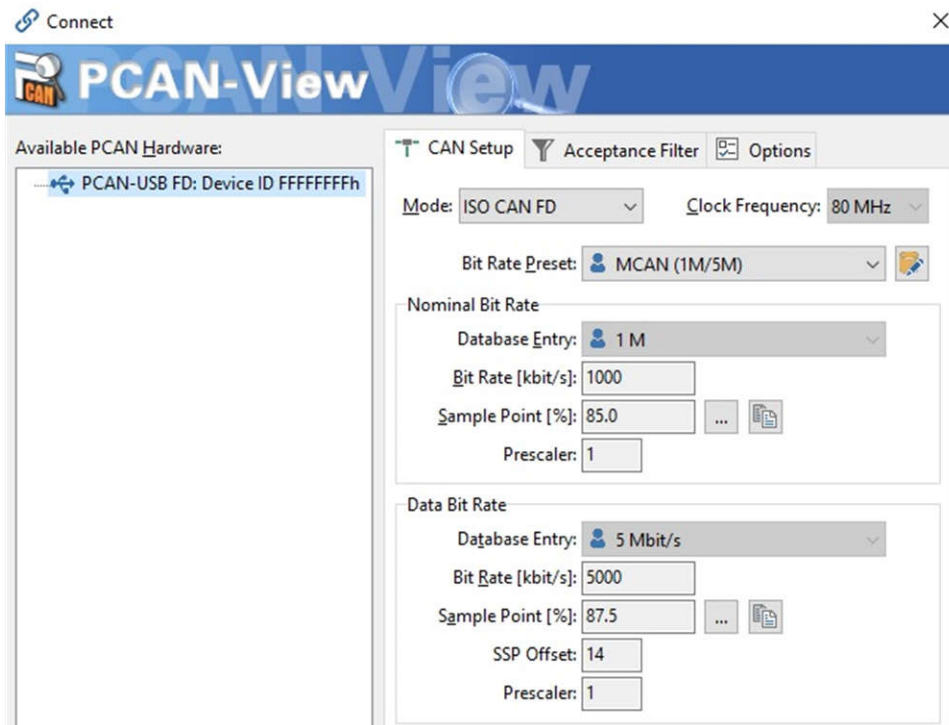


図 4-34. PCAN-View の設定

図 4-35 に示すように、ダブルクリックして CAN データ送信を開始します。

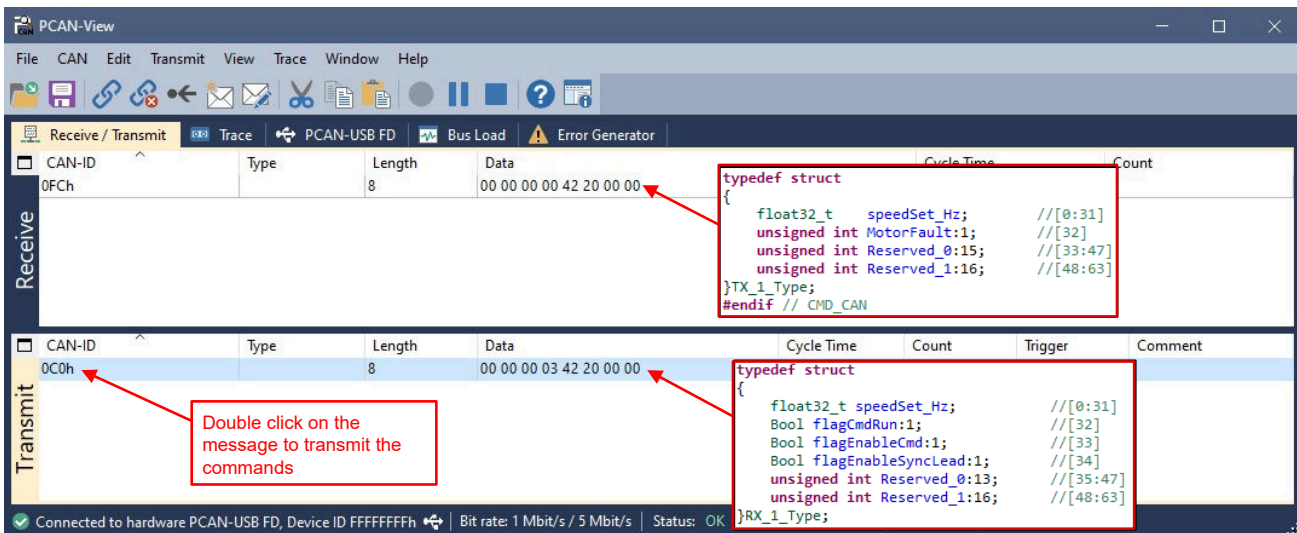


図 4-35. CAN データ送信の開始

図 4-36 に示すように、motorVars_M1.flagEnableRunAndIdentify は 1、motorVars_M1.speedRef_Hz は 40Hz が設定されています。この速度値は CAN 通信を介して受信者に送信されます。

| Expression | Type | Value | Address |
|---------------------------------------|----------------------|--------------------------|------------|
| systemVars.flagEnableSystem | unsigned short | 1 | 0x00083AF2 |
| systemVars.boardKit | enum Board_Kit_e | BOARD_BSXL3PHGAN_REVA | 0x00083AEE |
| systemVars.estType | enum EST_Type_e | EST_TYPE_ESMO | 0x00083AF0 |
| systemVars.currentSenseType | enum CURRENTSEN_T... | CURSEN_TYPE_INLINE_SHUNT | 0x00083AF1 |
| motorVars_M1.motorState | enum MOTOR_Status_e | MOTOR_CTRL_RUN | 0x0008345E |
| motorVars_M1.estimatorMode | enum ESTIMATOR_Mo... | ESTIMATOR_MODE_ESMO | 0x0008345C |
| motorVars_M1.ISRCCount | unsigned int | 342407 | 0x00083498 |
| motorVars_M1.speedRef_Hz | float | 40.0 | |
| motorVars_M1.speed_Hz | float | 39.9297409 | |
| motorVars_M1.flagEnableRunAndIdentify | unsigned short | 1 | 0x00083420 |
| motorVars_M1.flagRunIdentAndOnLine | unsigned short | 1 | 0x00083422 |
| motorVars_M1.flagClearFaults | unsigned short | 0 | 0x00083446 |

図 4-36. CAN コマンド:[Expressions] ウィンドウの変数

4.5.4 SFRA 機能の追加

テキサス・インスツルメンツのソフトウェア周波数応答アナライザ (SFRA) ライブラリは、外部の周波数応答アナライザを使用せずに、ソフトウェアのみを使用してパワー コンバータの周波数応答解析が行えるように設計されています。最適化されたライブラリは、高周波電力変換アプリケーションにおいて、閉ループ パワー コンバータのプラント特性、閉ループ ゲイン特性、開ループ ゲイン特性を特定するために使用でき、ゲイン マージン、位相マージン、開ループ ゲイン クロスオーバー周波数などの安定性情報を取得し、制御ループの性能評価に使用できます。

図 4-37 に示すような、デジタル制御の閉ループ パワー コンバータを考えてみましょう。

- H は、制御が必要なプラントの伝達関数です
- G は、デジタル補償器です
- GH は、開ループ伝達関数と呼ばれます
- CL は、閉ループ伝達関数と呼ばれ、 $GH/(1+GH)$ です
- r は、瞬間的な設定点またはコンバータのリファレンスです
- Ref は、DC 設定点のリファレンスです
- y は、A/D コンバータ (ADC) の帰還です
- e は、瞬間的な誤差です
- d は、センサのノイズと外乱です
- u は、PWM デューティ サイクルです

閉ループ システムにおける補償器には、主に次のような目的があります。

- システムが安定していることを確認する (たとえば、システムは漸近的にリファレンスを追従する)

$$e = \lim_{t \rightarrow \infty} e(t) = \lim_{t \rightarrow \infty} \frac{r}{(1+GH)} \rightarrow 0 \quad (64)$$

- システムによる外乱の排除により、堅牢な動作を維持する

$$S = \frac{y}{d} = \frac{1}{1+GH} \rightarrow 0 \quad (65)$$

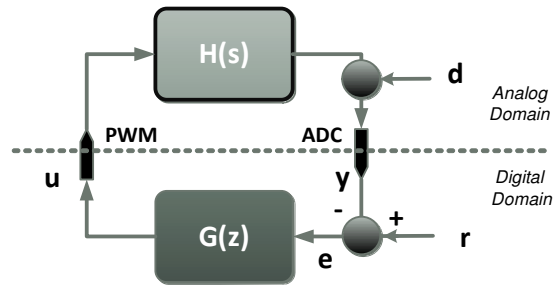


図 4-37. デジタル制御のパワー コンバータ

システムが目的を達成しているかどうかは、式 64 と 式 65 で示される開ループ伝達関数 (GH) を知ることで判断できます。

この目的のために、開ループ伝達関数 GH のボード線図が頻繁に使用され、ゲイン マージン (GM)、位相マージン (PM)、開ループ ゲインクロスオーバー周波数 (Fol_{g_cf}) などの数値が、閉ループ パワー コンバータの安定性や堅牢性を評価するためによく使用されます。

閉ループ伝達関数 (GH/(1+GH)) は、システムがコマンドリファレンスにどの程度追従できるかを示すものです。

SFRA ライブラリは、GH、GH/(1+GH)、H 周波数応答をソフトウェアを使用して測定できます。このデータは、次のような目的で使用できます。

- プラント モデル (H) の検証またはプラント モデル (H) の抽出
- 閉ループ プラントの補償器 (G) の設計
- 開ループ (GH) または閉ループ [GH/(1+GH)] のボード線図をプロットし、システムの閉ループ性能を検証します。

GH と H の周波数応答はプラントの情報を伝達するため、周波数応答を定期的に測定することで、データを使用して出力段の状態を評価することができます。

SFRA ライブラリは正弦波注入の原理に基づいており、注入振幅によってコンバータの通常の動作点に対する偏差が非常に小さくなることを前提としています。SFRA ライブラリは、パワー コンバータの制御コードに統合することができます。本書では、その手順について詳しく説明します。GH、H、CL の計算はすべて MCU で行われ、GH、H、CL の振幅と位相応答の配列全体がコントローラに保存されます。

コードに統合された後は、SFRA ライブラリを使用してコントローラの設計や微調整を行うことができます。SFRA ライブラリの一般的な使用フローは次のとおりです。

1. 開ループで SFRA 掃引を開始し、データを Excel ファイルに保存します。この情報をもとに、SFRA 掃引が実行された定常状態の動作点について、プラント モデルを特定することができます。
2. このプロジェクトに付属している MATLAB® スクリプトを使用すると、そのデータを MATLAB に読み取り、応答を伝達関数の曲線に当てはめることができます。これで、Sisotool を使用した補償器の設計が可能になります。
3. 新しい補償器の値は、MATLAB から Code Composer Studio™ プロジェクトにコピーできます。
4. 新しい係数を含むコードをコンパイルし、電力段を制御するマイクロコントローラにロードします。SFRA アルゴリズム (ステップ 1) を再実行し、開ループ ゲイン GH (文献ではループ ゲインとも呼ぶ) を測定することで、閉ループのシステム性能を検証できます。

まとめとして、テキサス・インスツルメンツのソフトウェア周波数応答アナライザは、パワー コンバータを系統的な方法で調整する手法を提供し、外部接続や装置を必要とすることなく、迅速かつ簡単にパワー コンバータの周波数応答を解析することができます。外部接続を使用しないため、SFRA を繰り返し実行して、定期的にパワー コンバータの状態を評価し、診断情報を取得できます。

4.5.4.1 動作原理

ソフトウェア周波数応答アナライザは、小信号の正弦波注入の原理に基づいています。図 4-38 に示すように、コントローラのリファレンスに小信号を注入し、フィードバック出力とコントローラ出力の周波数応答が計算されます。これにより、プラントの周波数応答特性と、閉ループ システムの開ループ周波数応答が得られます。

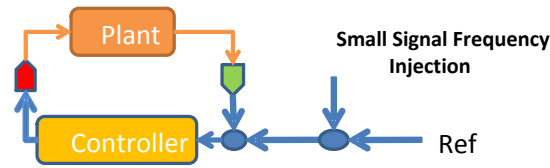


図 4-38. SFRA の動作原理

4.5.4.2 オブジェクトの定義

SFRA ライブラリでは、浮動小数点ベースの SFRA 構造を次のように定義しています。

```
typedef struct{
    float32_t *h_magVect;      //!< Plant Mag SFRA Vector
    float32_t *h_phaseVect;   //!< Plant Phase SFRA Vector
    float32_t *gh_magVect;    //!< Open Loop Mag SFRA Vector
    float32_t *gh_phaseVect;  //!< Open Loop Phase SFRA Vector
    float32_t *cl_magVect;    //!< Closed Loop Mag SFRA Vector
    float32_t *cl_phaseVect;  //!< Closed Loop Phase SFRA Vector
    float32_t *freqVect;      //!< Frequency Vector
    float32_t amplitude;      //!< Injection Amplitude
    float32_t isrFreq;        //!< SFRA ISR frequency
    float32_t freqStart;      //!< Start frequency of SFRA sweep
    float32_t freqStep;       //!< Log space between frequency points (optional)
    int16_t start;            //!< Command to start SFRA
    int16_t state;            //!< State of SFRA
    int16_t status;          //!< Status of SFRA
    int16_t vecLength;       //!< No. of Points in the SFRA
    int16_t freqIndex;       //!< Index of the frequency vector
    int16_t storeH;          //!< Flag to indicate if H vector is stored
    int16_t storeGH;         //!< Flag to indicate if GH vector is stored
    int16_t storeCL;         //!< Flag to indicate if CL vector is stored
    int16_t speed;           //!< variable to change the speed of the sweep
}SFRA_F32;
```

4.5.4.3 モジュール インターフェイスの定義

表 4-7. 浮動小数点モジュール インターフェイスの定義

| モジュール要素名 | 種類 | 説明 | 許容範囲 |
|---|----|--|--|
| h_magVect, gh_magVect, cl_magVect | 入力 | SFRA による H、GH、CL の振幅測定値を格納する配列へのポインタ。SFRA がそのベクトルを保存しない場合は NULL を渡します。 | 32 ビット位置へのポインタ。この位置には、単精度 (32 ビット) 浮動小数点で振幅ベクトルの値が格納されます。 |
| h_phaseVect, gh_phaseVect, cl_phaseVect | 入力 | SFRA による H、GH、CL の位相測定値を格納する配列へのポインタ。SFRA がそのベクトルを保存しない場合は NULL を渡します。 | 32 ビット位置へのポインタ。この位置には、単精度 (32 ビット) 浮動小数点で位相ベクトルの値が格納されます。 |
| freqVect | 入力 | SFRA が実行される周波数値の配列へのポインタ。 | 32 ビット位置へのポインタ。この位置には、単精度 (32 ビット) 浮動小数点で周波数ベクトルの値が格納されます。 |
| amplitude | 入力 | 小信号注入の振幅 (pu 単位)。 | 単精度 (32 ビット) 浮動小数点 (-1,1) |
| isrFreq | 入力 | SFRA ルーチンが呼び出される周波数。 | 単精度 (32 ビット) 浮動小数点 |
| freqStart | 入力 | 最初の周波数掃引データポイントの周波数。 | 単精度 (32 ビット) 浮動小数点 |
| freqStep | 入力 | $10^{(1/(1 \text{ デイケードあたりのステップ数}))}$ 。 | 単精度 (32 ビット) 浮動小数点 |
| start | 入力 | SFRA を起動するコマンド。 | int16_t |
| 状態 | 出力 | SFRA の状態。SFRA 注入が実行中の場合はゼロ以外、SFRA 注入がアクティブ / 実行中でない場合は 0。 | int16_t |
| ステータス | 出力 | SFRA のステータス。1 は SFRA 注入が実行中、0 は SFRA 注入がアクティブ / 進行中。 | int16_t |
| vecLength | 入力 | SFRA が実行されるポイントの数。 | int16_t |

表 4-7. 浮動小数点モジュール インターフェイスの定義 (続き)

| モジュール要素名 | 種類 | 説明 | 許容範囲 |
|-----------|----|---|-----------------------|
| freqIndex | 出力 | SFRA が実行されている freqVect の周波数インデックス番号。 | int16_t (0-vecLength) |
| storeH | 出力 | SFRA の構成を反映します。1 の場合は H ベクトルが格納されます。0 の場合は H ベクトルは格納されません。SFRA の構成中に H mag または位相ベクトルに NULL ベクトルが渡されたときに発生します。 | int16_t (0 または 1) |
| storeGH | 出力 | SFRA の構成を反映します。1 の場合は GH ベクトルが格納されます。0 の場合は GH ベクトルは格納されません。SFRA の構成中に GH mag または位相ベクトルに NULL ベクトルが渡されたときに発生します。 | int16_t (0 または 1) |
| storeCL | 出力 | SFRA の構成を反映します。1 の場合は CL ベクトルが格納されます。0 の場合は CL ベクトルは格納されません。SFRA の構成中に CL mag または位相ベクトルに NULL ベクトルが渡されたときに発生します。 | int16_t (0 または 1) |
| speed | 入力 | 掃引の速度を変更するために使用され、1 より大きい必要があります。1 の場合、STB の例ではテンプレートの掃引に約 58 秒かかります。システムでの実際の速度は、測定される周波数ポイントと SFRA モジュールの呼び出しに使用される ISR レートによって異なります。速度の数値が大きいほど、掃引は遅くなります。 | int16_t (1 より大きい) |

4.5.4.4 SFRA の使用

SFRA をプロジェクトに統合するには、次の手順を行います。

1. SFRA 機能を有効にするには、図 4-2 に示すように、事前定義シンボル SFRA_ENABLE をプロジェクトプロパティに追加する必要があります。
2. SFRA 掃引を開始するには、SFRA オブジェクトを [Watch] ウィンドウに配置します。
3. 図 4-39 に示すように、SFRA 掃引を開始する場合は、SFRA_OBJ.start を 1 に設定します。

| sfra1 | struct SFRA_F32 | {h_magVect=0x00083640 {15.5954409},h_... | 0x000838A8 |
|----------------|-----------------|--|------------|
| > h_magVect | float * | 0x00083640 {15.5954409} | 0x000838A8 |
| > h_phaseVect | float * | 0x00083698 {0.306601793} | 0x000838AC |
| > gh_magVect | float * | 0x000836F0 {7.98486328} | 0x000838B0 |
| > gh_phaseVect | float * | 0x00083748 {-75.6564407} | 0x000838B4 |
| > cl_magVect | float * | 0x000837A0 {-1.32463789} | 0x000838B8 |
| > cl_phaseVect | float * | 0x000837F8 {-19.3731956} | 0x000838BC |
| > freqVect | float * | 0x00083850 {20.0} | 0x000838C0 |
| (x) amplitude | float | 0.004999999989 | 0x000838C4 |
| (x) isrFreq | float | 15000.0 | 0x000838C8 |
| (x) freqStart | float | 20.0 | 0x000838CC |
| (x) freqStep | float | 1.26335502 | 0x000838D0 |
| (x) start | short | 0 | 0x000838D4 |
| (x) state | short | 0 | 0x000838D6 |
| (x) status | short | 0 | 0x000838D8 |
| (x) vecLength | short | 22 | 0x000838DA |
| (x) freqIndex | short | 22 | 0x000838DC |
| (x) storeH | short | 1 | 0x000838DE |
| (x) storeGH | short | 1 | 0x000838E0 |
| (x) storeCL | short | 1 | 0x000838E2 |
| (x) speed | short | 1 | 0x000838E4 |

図 4-39. SFRA 機能の開始

4. SFRA_OBJ.FreqIndex 変数を監視します。SFRA 掃引が実行されると、変数は徐々に増加します。
5. SFRA_OBJ.FreqIndex が VEC_Length に達すると、SFRA 掃引が完了します。

| | | | |
|------------------|-----------|--|------------|
| > freqVect | float[22] | [20.0,25.2671013,31.92132,40.327961,50.94... | 0x00083850 |
| > olMagVect | float[22] | [7.98486328,6.04782486,4.30153942,2.6732... | 0x000836F0 |
| > plantMagVect | float[22] | [15.5954409,15.4647503,15.5659208,15.621... | 0x00083640 |
| > plantPhaseVect | float[22] | [0.306601793,-1.85650432,-3.8041153,-4.7... | 0x00083698 |

図 4-40. SFRA のデータ配列

6. SFRA の初期化の一環として、開ループとプラントの振幅と位相は、特定の配列に格納されます。

```
__attribute__((section(".sfradata"))) float32_t plantMagVect[SFRA_FREQ_LENGTH];
__attribute__((section(".sfradata"))) float32_t plantPhaseVect[SFRA_FREQ_LENGTH];
__attribute__((section(".sfradata"))) float32_t olMagVect[SFRA_FREQ_LENGTH];
__attribute__((section(".sfradata"))) float32_t olPhaseVect[SFRA_FREQ_LENGTH];
__attribute__((section(".sfradata"))) float32_t freqVect[SFRA_FREQ_LENGTH];
```

7. これらを [Watch] ウィンドウに配置して、応答を確認および調査します。
8. 掃引が完了したら、CCS 内で [View] -> [MemoryBrowser] をクリックします。
9. Memory Browser 内で、&freqVect と入力して周波数ベクトルを表示し、32 ビット浮動小数点を選択します。

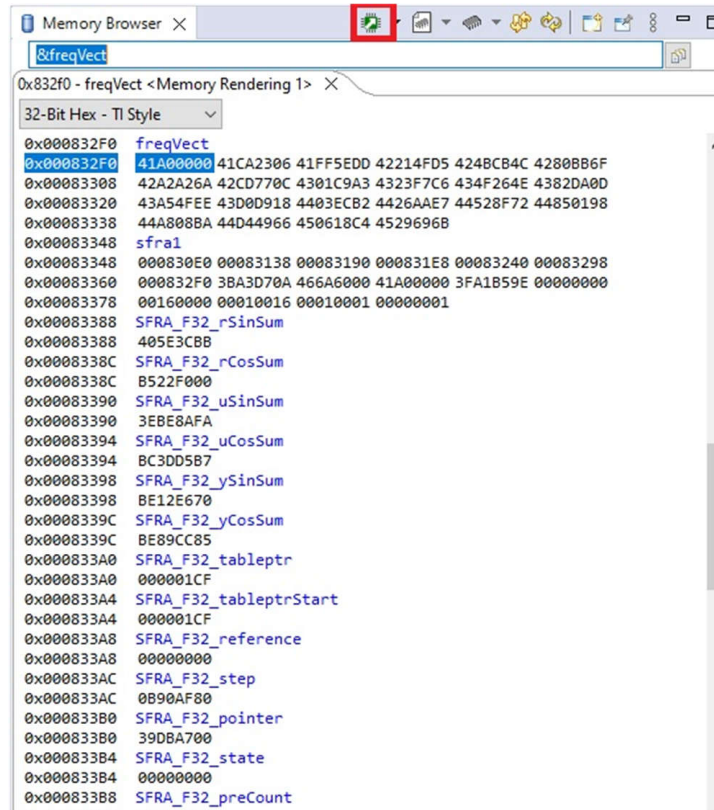


図 4-41. 格納された SFRA ベクトルの Memory Browser ビュー

10. 図 4-41 で囲みで示した [Save Memory] をクリックします。

11. ポップアップ ウィンドウが表示されます。テキサス・インスツルメンツのデータを選択し、希望する場所にファイル名 *.dat を指定します。

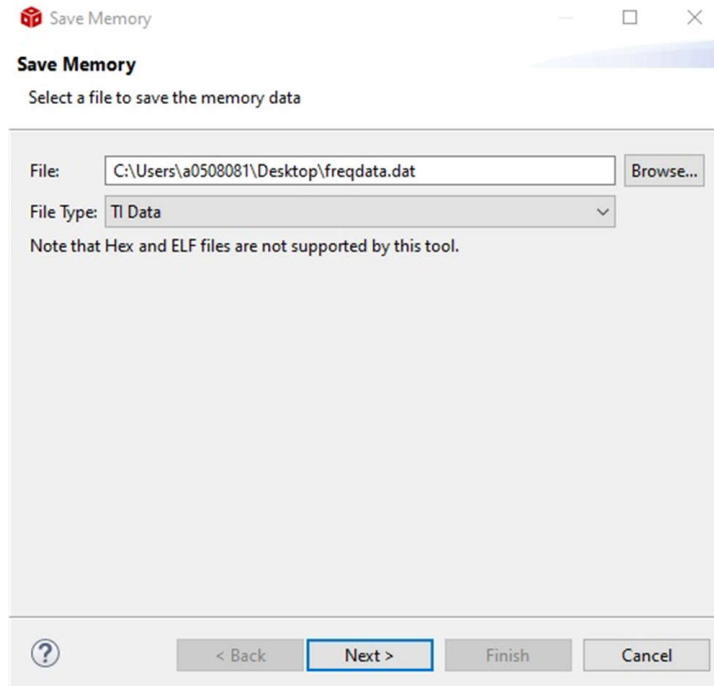


図 4-42. Save Memory ポップアップ ウィンドウ

12. [Next] をクリックし、Memory Browser から配列の開始アドレスを指定し、次に長さを指定します。

13. 32 ビット浮動小数点を選択されていることを確認してください。“Finish”をクリックします。

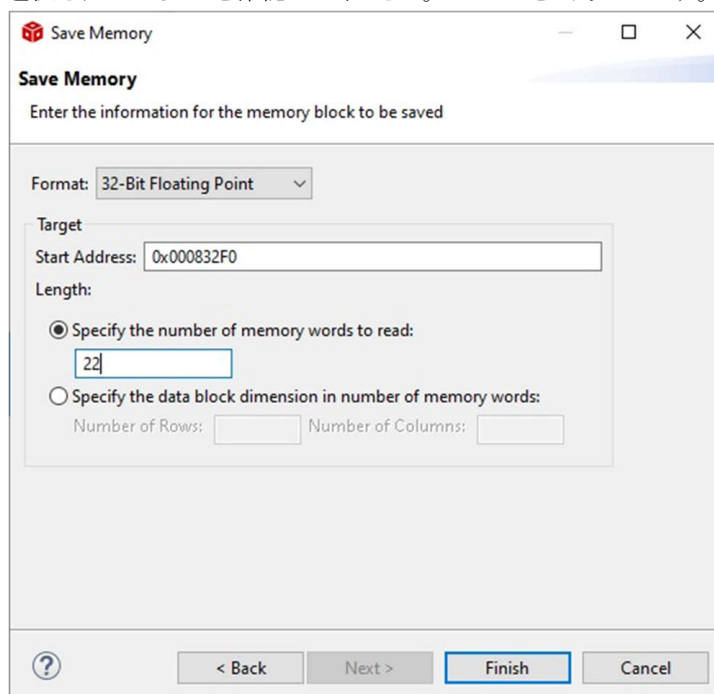


図 4-43. Save Memory オプション

14. これにより、データは *.dat ファイルに保存されます。
15. plantMagVect、plantPhaseVect、olMagVect、olPhaseVect についてこの手順を繰り返すと、5 つの *.dat ファイルが作成されます。
16. このデータを MATLAB や他のツールで使用する場合は、データを Excel ファイルに入力できます。
17. <project directory>\libraries\SFRA\scripts にある SFRA.xlsx ファイルを Excel で開きます。
18. ファイルは別名で保存することもできます。
19. この Excel シートには 5 列あり、最初の列は周波数データです。
20. 保存した *.dat ファイルを開きます。

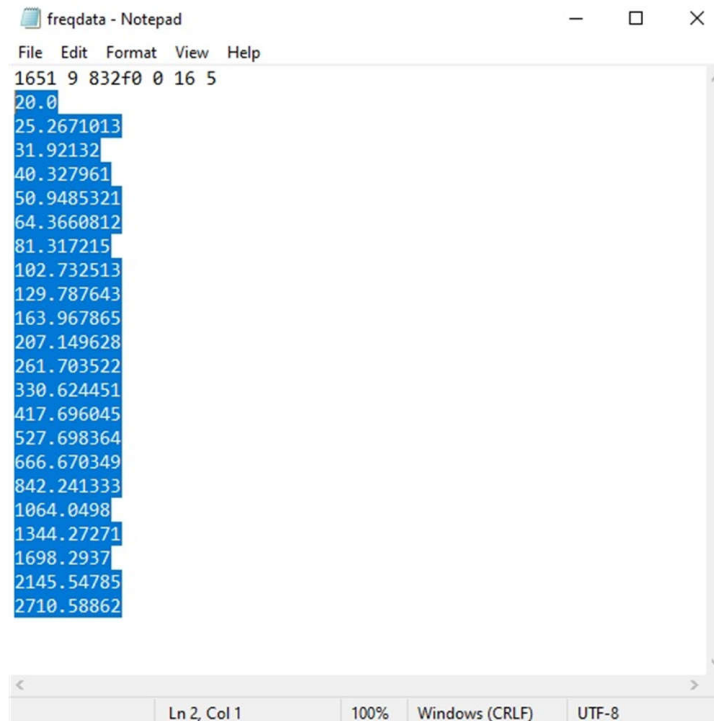


図 4-44. .dat ファイルからデータを選択して Excel に入力する

21. 2 行目からファイル末尾までのデータを選択し、Ctrl キーと C キーを押してデータをコピーします。
22. Excel ファイルを開き、対応するベクトルの下にある最初の要素に移動し、Ctrl キーと V キーを押して配列をコピーします。

| | A | B | C | D | E | F | G |
|----|----------------|-------------------|----------------|-------------------|----------------|---|---|
| 1 | Frequency (Hz) | OL Magnitude (dB) | OL Phase (Deg) | CL Magnitude (dB) | CL Phase (Deg) | | |
| 2 | 20 | 7.207811356 | -79.19759369 | 14.76470852 | -3.324057102 | | |
| 3 | 25.26710129 | 5.42373085 | -77.01151276 | 14.89926052 | -4.582312107 | | |
| 4 | 31.92131996 | 3.688827038 | -72.40774536 | 14.90183258 | -4.466232777 | | |
| 5 | 40.32796097 | 1.923423886 | -70.51013184 | 14.92250633 | -7.282536507 | | |
| 6 | 50.9485321 | -0.125275463 | -56.76221848 | 14.20130825 | -0.285738766 | | |
| 7 | 64.36608124 | -1.563976288 | -83.30471039 | 14.03975582 | -32.29851151 | | |
| 8 | 81.31721497 | -2.304516792 | -49.57841492 | 14.94800282 | -4.848727703 | | |
| 9 | 102.7325134 | -3.312849045 | -52.57070923 | 14.48073673 | -15.37914753 | | |
| 10 | 129.7876434 | -4.069205761 | -50.4241333 | 14.3869276 | -19.67580223 | | |
| 11 | 163.967865 | -4.63369894 | -48.89741516 | 14.30565166 | -24.02565956 | | |
| 12 | 207.1496277 | -5.352626801 | -49.90703964 | 13.82841015 | -30.16960144 | | |
| 13 | 261.7035217 | -5.818374634 | -52.38605499 | 13.50661755 | -36.94898987 | | |
| 14 | 330.6244507 | -6.837041378 | -56.68714905 | 12.54745388 | -44.62264633 | | |
| 15 | 417.6960449 | -7.818169594 | -61.64614487 | 11.58655357 | -52.24248123 | | |
| 16 | 527.6983643 | -8.935320854 | -67.09944153 | 10.47279167 | -59.65838242 | | |
| 17 | 666.6703491 | -10.31351948 | -73.40723419 | 9.049505234 | -67.21392822 | | |
| 18 | 842.241333 | -11.86766434 | -80.59755707 | 7.448073864 | -75.09159851 | | |
| 19 | 1064.049805 | -13.48760986 | -87.23434448 | 5.848493099 | -82.07510376 | | |
| 20 | 1344.272705 | -15.32057476 | -96.00611877 | 4.050667763 | -90.75597382 | | |
| 21 | 1698.293701 | -17.30090523 | -103.1983414 | 2.142414331 | -97.84544373 | | |
| 22 | 2145.547852 | -19.28324127 | -111.6833649 | 0.289372593 | -105.8947372 | | |
| 23 | 2710.588623 | -21.23872375 | -121.0471802 | -1.494733214 | -114.936348 | | |

図 4-45. SFRA データを Excel ファイルにコピーする

23. 各列についてこの手順を繰り返します。
24. Excel ファイルが 5 列すべてで更新されたら、MATLAB スクリプトを使用して SFRA データをインポートします。次に、sisotool 内のスクリプトを使用して補償器を設計し、安定性解析を実行します。

4.6 カスタム ボードの製作

4.6.1 新しいカスタム ボードの製作

このセクションでは、モーター駆動用のアプリケーション ボードを設計する方法と、独自のボードで使用するために子のプロジェクトを移行する方法について説明します。

4.6.1.1 ハードウェア設定

カスタム ボードを使用する場合は、マイクロコントローラとゲートドライバの電源が適正であること、JTAG エミュレータが正常に接続できていることを確認してください。以降のセクションの説明に従って、リファレンス コードをカスタム ボードと互換性があるように変更し、セクション 4.4 に示すように、ビルドレベル 1 から開始して、ビルドレベル 4 までコードを実行します。

4.6.1.2 リファレンス コードのカスタム ボードへの移行

リファレンス コードを新しいテキサス・インスツルメンツのモーター ドライバ キットまたはカスタム ボードに移行するには、以降のセクションで説明するように、モーター ドライバ回路に応じて、`user_mtr1.h` ファイル内のハードウェア パラメータとモーター制御パラメータを構成し、`AM263_xxx.syscfg` ファイル、`hal.h` ファイル、`hal.c` ファイル内の関連ペリフェラルを構成する必要があります。

次のブロック図では、モーター制御設定とテキサス・インスツルメンツの MCU ペリフェラルの構成に使用する関数呼び出しをまとめています (図 4-46)。

このプロジェクトでは、ハードウェアの構成に関連して、呼び出しが一度だけの HAL 関数がいくつかあります。このような関数はすべて、ペリフェラルまたはモーター ドライバ IC の構成を処理するものです。

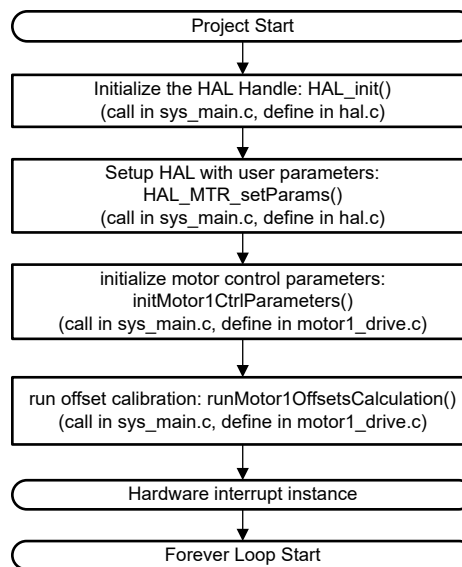


図 4-46. HAL の構成とモーター制御設定のブロック図

4.6.1.2.1 ハードウェア ボード パラメータの設定

`user_mtr1.h` ファイルには、モーター制御用のすべてのユーザー パラメータが格納されます。AD コンバータへの入力における相電流と相電圧の最大値は、ハードウェアに依存するものであり、電流と電圧のセンシング回路および ADC 入力でのスケールリングに基づいている必要があります。位相電流センサと位相電圧センサの数も、`user_mtr1.h` ファイルで定義されます。これらの値はハードウェアに依存します。

`user_mtr1.h` ファイルで定義されている構成可能なパラメータはすべて、Excel® スプレッドシートの `Motor Control Parameters Calculation.xlsx` を使用して計算できます。このファイルは

\examples\universal_motorcontrol_lab\doc にあるプロジェクト フォルダに含まれています。以下のコードに示すように、太字で示されたパラメータを user_mtr1.h ファイルにコピーしてください。

```

//! \brief Defines the maximum voltage at the AD converter
#define USER_M1_ADC_FULL_SCALE_VOLTAGE_V (57.52845691f)

//! \brief Defines the analog voltage filter pole location, Hz
#define USER_M1_VOLTAGE_FILTER_POLE_HZ (680.4839141f) // 47nF

//! \brief Defines the maximum current at the AD converter
#define USER_M1_ADC_FULL_SCALE_CURRENT_A (47.14285714f) // gain=10

```

4.6.1.2.2 モーター制御パラメータの変更

PMSM モーターの user_mtr1.h ファイルに含まれるパラメータは、以下のコードに示すとおりです。モーターのパラメータは、モーターのデータシートから確認できます。

```

#define USER_MOTOR1_TYPE MOTOR_TYPE_PM
#define USER_MOTOR1_NUM_POLE_PAIRS (4)

#define USER_MOTOR1_Rs_Ohm (0.38157931f)
#define USER_MOTOR1_Ls_d_H (0.000188295482f)
#define USER_MOTOR1_Ls_q_H (0.000188295482f)
#define USER_MOTOR1_RATED_FLUX_VpHz (0.0396642499f)
#define USER_MOTOR1_MAX_CURRENT_A (6.0f)

```

4.6.1.2.3 ピン配置の変更

AM263_xxx.syscfg ファイルは、GPIO ピンの機能を設定および構成し、使用するハードウェア モーター ドライバ ボード/キットに応じて、指定されたピンの方向とモードを設定します。カスタム ボード向け、現時点でユニバーサル ラボ コードをサポートしていないテキサス・インスツルメンツのモーター ドライバ EVM 向け、または別のテキサス・インスツルメンツの MCU で使用するためにコードを変更する場合、これらの GPIO 割り当ては、モーター ドライバ ボードに適切に対応するように変更する必要があります。

4.6.1.2.4 PWM モジュールの構成

SysConfig ファイルは PWM チャネルの設定や構成を定義します。モーター コントローラの PWM 入力に使用される PWM チャネルのベース アドレスは hal.h ファイルで定義され、ベース アドレスは hal.c ファイルの PWM ハンドルに割り当てられます。図 4-47 に、LP-AM263 と BOOSTXL-3PHGANINV の間の PWM 信号の接続図を示します。

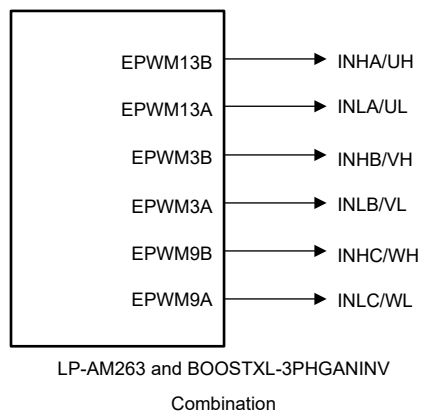


図 4-47. PWM の接続図

以下に、.syscfg ファイル、hal.h ファイル、hal.c ファイルから、PWM 信号を構成するコードを示します。

1. PWM モジュールのベース アドレスは、以下に示すように hal.h ファイルで定義されます。

```

#define MTR1_PWM_U_BASE CONFIG_EPWM13_BASE_ADDR
#define MTR1_PWM_V_BASE CONFIG_EPWM3_BASE_ADDR
#define MTR1_PWM_W_BASE CONFIG_EPWM9_BASE_ADDR

```

2. GPIO は、.syscfg ファイルで PWM 出力として設定されます。

| EPWM Instance | | EPWM3 | |
|---|------------|--------------|-----------|
| <input checked="" type="checkbox"/> Signals ↓ | Pins | Pull Up/Down | Slew Rate |
| <input checked="" type="checkbox"/> EPWMxA Pin(EPWM3_A) | EPWM3_A/E2 | No Pull | Low |
| <input checked="" type="checkbox"/> EPWMxB Pin(EPWM3_B) | EPWM3_B/E3 | No Pull | Low |

図 4-48. PWM モジュールの GPIO 構成

3. 以下のコードは、hal.c ファイルにある HAL_MTR1_init() 関数において、PWM モジュールの対応するベースアドレスを PWM ハンドルに割り当てています。以下のコードは、新しいボードやテキサス・インスツルメンツの MCU に適合させるときに変更する必要はありません。以下のコードブロックは、コード内で PWM ハンドルがどのように初期化されるかを示しています。

```
// initialize PWM handles for Motor 1
obj->pwmHandle[0] = MTR1_PWM_U_BASE;    //!< the PWM handle
obj->pwmHandle[1] = MTR1_PWM_V_BASE;    //!< the PWM handle
obj->pwmHandle[2] = MTR1_PWM_W_BASE;    //!< the PWM handle
```

4. 図 4-49 に、EPWM タイムベースの構成を示します。位相 A の SYNC OUT パルスは、他の PWM の SYNC IN パルスのソースとして使用されます。

| EPWM Time Base | |
|-------------------------------------|--|
| Emulation Mode | Free run |
| Time Base Clock Divider | Divide clock by 1 |
| High Speed Clock Divider | Divide clock by 1 |
| Time Base Period | 0 |
| Time Base Period Link | Disable Linking |
| Enable Time Base Period Global Load | <input type="checkbox"/> |
| Time Base Period Load Mode | PWM Period register access is directly |
| Initial Counter Value | 0 |
| Counter Mode | Up - down - count mode |
| Counter Mode After Sync | Count up after sync event |
| Enable Phase Shift Load | <input type="checkbox"/> |
| Sync In Pulse Source | Sync-in source is EPWM3 sync-out signal |
| Sync Out Pulse | Counter zero event generates EPWM sync-out pulse |
| One-Shot Sync Out Trigger | Trigger is OSH sync |
| Force A Sync Pulse | <input type="checkbox"/> |

図 4-49. EPWM タイムベースの構成

EPWM アクションクオリファイアの構成に、LP-AM263 と BOOSTXL-3PHGANINV の組み合わせの EPWM アクションクオリファイア出力イベントの構成を示します。PWM アクションクオリファイア出力は、ハードウェアボードに基づいて設定する必要があります。

| EPWM Action Qualifier | |
|--|---|
| Continuous SW Force Global Load | <input type="checkbox"/> |
| Continuous SW Force Shadow Mode | Shadow mode load when counter equals zero |
| T1 Trigger Source | Digital compare event A 1 |
| T2 Trigger Source | Digital compare event A 1 |
| EPWMA Output Configuration | |
| EPWMA Global Load Enable | <input type="checkbox"/> |
| EPWMA Shadow Mode Enable | <input checked="" type="checkbox"/> |
| EPWMA Shadow Load Event | Load when counter equals zero |
| EPWMA One-Time SW Force Action | No change in the output pins |
| EPWMA Continuous SW Force Action | Software forcing disabled |
| Events Configured For EPWMA Output | None |
| EPWMA OutputEvent Output Configuration | |
| EPWMA TBCTR Equals Zero | Set output pins to low |
| EPWMA TBCTR Equals Period | Set output pins to High |
| EPWMA TBCTR Up Equals COMPA | Set output pins to High |
| EPWMA TBCTR Down Equals COMPA | Set output pins to low |
| EPWMA TBCTR Up Equals COMPB | No change in the output pins |
| EPWMA TBCTR Down Equals COMPB | No change in the output pins |
| EPWMA T1 Event On Count Up | No change in the output pins |
| EPWMA T1 Event On Count Down | No change in the output pins |
| EPWMA T2 Event On Count Up | No change in the output pins |
| EPWMA T2 Event On Count Down | No change in the output pins |
| EPWMB Output Configuration | |

図 4-50. EPWM アクション クオリファイアの構成

図 4-51 に、LP-AM263 の EPWM デッドバンドの構成を示します。EPWMA-B のスワップ出力が、LaunchPad™ と Booster Pack™ のハイサイドとローサイドの PWM と一致するかどうかチェックされます。

| EPWM Dead-Band | |
|-------------------------------------|--------------------------------------|
| Rising Edge Delay Input | Input signal is ePWMA |
| Falling Edge Delay Input | Input signal is ePWMA |
| Rising Edge Delay Polarity | DB polarity is not inverted |
| Falling Edge Delay Polarity | DB polarity is inverted |
| Enable Rising Edge Delay | <input checked="" type="checkbox"/> |
| Rising Edge Delay Value | 10 |
| Enable Falling Edge Delay | <input checked="" type="checkbox"/> |
| Falling Edge Delay Value | 10 |
| Swap Output for EPWMA | <input checked="" type="checkbox"/> |
| Swap Output for EPWMB | <input checked="" type="checkbox"/> |
| Enable Deadband Control Global Load | <input type="checkbox"/> |
| Enable Deadband Control Shadow Mode | <input type="checkbox"/> |
| Enable RED Global Load | <input type="checkbox"/> |
| Enable RED Shadow Mode | <input type="checkbox"/> |
| Enable FED Global Load | <input type="checkbox"/> |
| Enable FED Shadow Mode | <input type="checkbox"/> |
| Dead Band Counter Clock Rate | Dead band counter runs at TBCLK rate |

図 4-51. EPWM デッドバンドの構成

4.6.1.2.5 ADC モジュールの構成

前述の PWM のセクションと同様に、ADC 接続も、ユニバーサル モーター制御プロジェクトでサポートされていないカスタム ボードやテキサス・インスツルメンツのモーター制御キット用に変更できます。.syscfg ファイルは、ADC チャンネルをモ

ータードライバ ボードに正しく対応するように設定や構成を定義します。例として、**LP-AM263** と **BOOSTXL-3PHGANINV** の組み合わせの接続図を **図 4-52** に示します。ADC モジュールの構成について以下の手順で説明します。

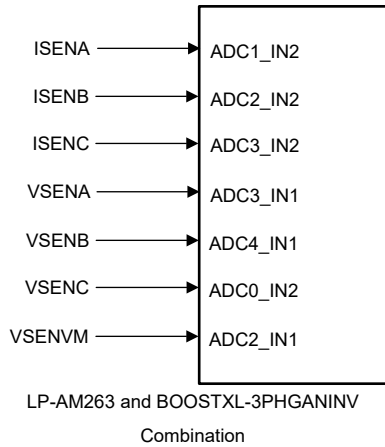


図 4-52. ADC の接続図

- 以下のコードは、`hal.h` ファイル内の ADC モジュールのベース アドレス、割り当てられたチャンネル、SOC の定義を示しています。SOC 番号については、異なる ADC モジュール (以下の場合にはモジュール A とモジュール C) に属する限り、複数の ADC を同じ SOC 番号に関連付けることができることに注意してください。すべての電流とすべての電圧をできるだけ近接してサンプリングするようにして、この点を考慮して SOC 番号を構成してください。以下のコードは、新しいボードやテキサス・インスツルメンツの MCU に適合させるときに変更する必要はありません。以下のコードは、ADC がどのように初期化されるかを示すだけで、変更は `.syscfg` ファイルで行うことができます。

```
#define MTR1_IU_ADC_BASE      CONFIG_ADC1_BASE_ADDR //J7.67 ADC1_AIN2
#define MTR1_IV_ADC_BASE      CONFIG_ADC2_BASE_ADDR //J7.68 ADC2_AIN2
#define MTR1_IW_ADC_BASE      CONFIG_ADC3_BASE_ADDR //J7.69 ADC3_AIN2
#define MTR1_VU_ADC_BASE      CONFIG_ADC3_BASE_ADDR //J7.64 ADC3_AIN1
#define MTR1_VV_ADC_BASE      CONFIG_ADC4_BASE_ADDR //J7.65 ADC4_AIN1
#define MTR1_VW_ADC_BASE      CONFIG_ADC0_BASE_ADDR //J7.66 ADC0_AIN2
#define MTR1_VDC_ADC_BASE     CONFIG_ADC2_BASE_ADDR //J7.63 ADC2_AIN1

#define MTR1_IU_ADCRES_BASE    CONFIG_ADC1_RESULT_BASE_ADDR
#define MTR1_IV_ADCRES_BASE    CONFIG_ADC2_RESULT_BASE_ADDR
#define MTR1_IW_ADCRES_BASE    CONFIG_ADC3_RESULT_BASE_ADDR
#define MTR1_VU_ADCRES_BASE    CONFIG_ADC3_RESULT_BASE_ADDR
#define MTR1_VV_ADCRES_BASE    CONFIG_ADC4_RESULT_BASE_ADDR
#define MTR1_VW_ADCRES_BASE    CONFIG_ADC0_RESULT_BASE_ADDR
#define MTR1_VDC_ADCRES_BASE    CONFIG_ADC2_RESULT_BASE_ADDR

#define MTR1_IU_ADC_CH_NUM     ADC_CH_ADCIN2
#define MTR1_IV_ADC_CH_NUM     ADC_CH_ADCIN2
#define MTR1_IW_ADC_CH_NUM     ADC_CH_ADCIN2
#define MTR1_VU_ADC_CH_NUM     ADC_CH_ADCIN1
#define MTR1_VV_ADC_CH_NUM     ADC_CH_ADCIN1
#define MTR1_VW_ADC_CH_NUM     ADC_CH_ADCIN2
#define MTR1_VDC_ADC_CH_NUM     ADC_CH_ADCIN1

#define MTR1_IU_ADC_SOC_NUM     ADC_SOC_NUMBER0 // SOC0-PPB1
#define MTR1_IV_ADC_SOC_NUM     ADC_SOC_NUMBER0 // SOC0-PPB1
#define MTR1_IW_ADC_SOC_NUM     ADC_SOC_NUMBER0 // SOC0-PPB2
#define MTR1_VU_ADC_SOC_NUM     ADC_SOC_NUMBER1 // SOC1
#define MTR1_VV_ADC_SOC_NUM     ADC_SOC_NUMBER1 // SOC1
#define MTR1_VW_ADC_SOC_NUM     ADC_SOC_NUMBER1 // SOC1
#define MTR1_VDC_ADC_SOC_NUM     ADC_SOC_NUMBER1 // SOC1

#define MTR1_IU_ADC_PPB_NUM     ADC_PPB_NUMBER1 // SOC0-PPB1
#define MTR1_IV_ADC_PPB_NUM     ADC_PPB_NUMBER1 // SOC0-PPB1
#define MTR1_IW_ADC_PPB_NUM     ADC_PPB_NUMBER1 // SOC0-PPB2
```

- 図 4-53** に、`.syscfg` ファイル内の ISR の割り込みソースの定義を示します。

Enable ADC Converter

INT Configurations Interrupt Configurations

ADC Interrupt Pulse Mode Occurs at the end of the conversion

ADC Interrupt Cycle Offset 0

INT1 ADC Interrupt 1

Enable ADC Interrupt1

Interrupt1 SOC Source SOC/EOC number 1

Enable Continue to Interrupt Mode

図 4-53. ADC 割り込みの構成

3. 図 4-54 に、ADC の変換開始トリガのソースを定義します。この ePWM SOC トリガは、コード内でイネーブルにされた同じ ePWM SOC、および pwmHandle[0] に関連付けられた同じ ePWM に対応する必要があります。この場合、EPWM3 A が ADC 用の SOC として使用されます。

SOC Configurations Start of Conversion Configurations

SOC0 Start of Conversion 0

SOC0 Channel single-ended, ADCIN0

SOC Triggers

SOC0 Trigger ePWM3, ADCSOCA

SOC0 Interrupt Trigger No ADCINT will trigger the SOC

SOC0 Sample Window [SYSCLK Counts] 16

SOC0 Sample Time In Nanoseconds 5

図 4-54. ADC 変換開始の構成

4.6.1.2.6 CMPSS モジュールの構成

CMPSS モジュールは、位相電流の過電流監視に使用されます。CMPSS DAC を使用してスレッショルドが設定されており、電流センス アンプの出力がこのスレッショルドを超えると、CMPSS 出力がトリップします。

カスタム モータードライバ ボードを使用する場合や、現在のユニバーサル モーター制御プロジェクトでサポートされていないテキサス・インスツルメンツの MCU やモータードライバ EVM にコードを移行する場合、ADC ピンと CMPSS モジュールの間の接続は、モータードライバとテキサス・インスツルメンツの MCU の接続に基づいて .syscfg ファイルで適切に変更する必要があります。CMPSS モジュールの内部接続の詳細については、『AM263x Sitara™ マイクロコントローラ データシート』の「ADC 信号の説明」の表を参照してください。

.syscfg ファイルは、使用するモータードライバ ボードに従って CMPSS モジュールを構成します。例として、LP-AM263 と BOOSTXL-3PHGANINV の接続図を、図 4-55 に示します。図 4-56 に、CMPSSA のブロック図を示します。CMPSSA は、COMPL の正の信号に対する多重化可能な入力として、INH と INL を追加サポートしています。

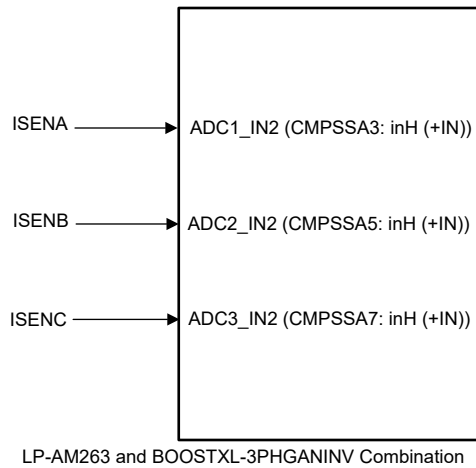


図 4-55. CMPSS の接続図

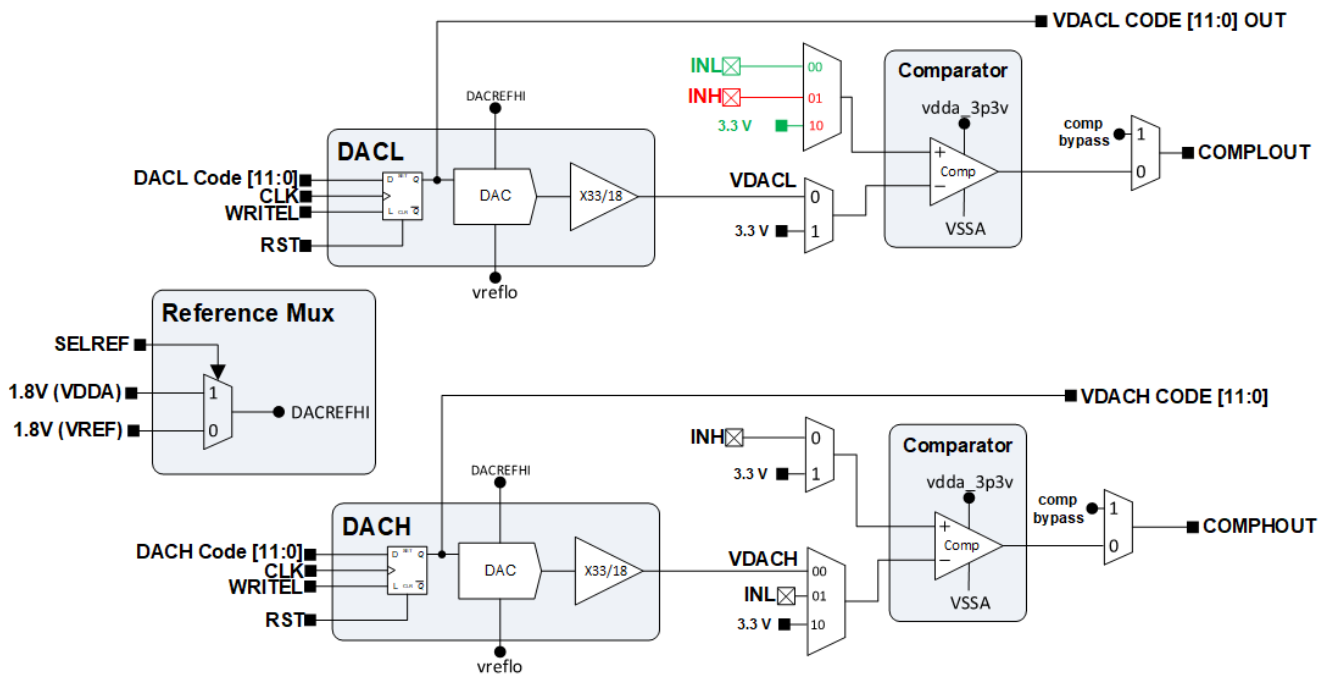


図 4-56. CMPSSA のブロック図

各 CMPSS コンパレータはハイとローのコンパレータを備えているため、信号は目的のコンパレータの目的の入力に合わせて適切に多重化される必要があります。これらの接続の詳細については、使用するマイクロコントローラのデータシートの「アナログピンと内部接続」の表を参照してください。図 4-57 に、LP-AM263 と BOOSTXL-3PHGANINV の組み合わせで位相電流のウィンドウ比較を行う CMPSS コンパレータの構成を示します。DAC 値は、定義された最大電流に基づいて、コード内で更新されます。AM263x デバイスの場合、ADCx_AIN1 と ADCx_AIN3 は (INL) にのみ接続されているため、正と負の両方の過電流トリップが制限されることに注意してください。TMDSHVMTRINSPIN と TMDSCNCD263 の組み合わせでは、U 相電流測定に使用する ADC は ADC1_AIN3 に接続されます。

また、LaunchPad または EVM controlCARD で DAC リファレンス電圧と一致する適切な電圧リファレンスを選択することに注意してください。たとえば、AM263x LaunchPad では、DAC VREF スイッチ (S1) を使用して AM263x オンダイ LDO を選択します。

| | |
|--------------------------------------|-------------------------------------|
| Name | CONFIG_CMPSS_IU |
| CMPSS Instance | CMPSSA3 |
| Enable Module | <input checked="" type="checkbox"/> |
| High Comparator Configuration | |
| Negative Input Source | Input driven by internal DAC |
| Output Is Inverted | <input type="checkbox"/> |
| Asynch OR Latch | <input type="checkbox"/> |
| Signal Driving CTRIPOUTH | Filter output drives CTRIPOUTH |
| Signal Driving CTRIPH | Filter output drives CTRIPH |
| Set High Comparator DAC Value | 3584 |
| Digital Filter Configuration | |
| Ramp Generator Configuration | |
| Low Comparator Configuration | |
| Positive Input Source | Input driven by external pin INH |
| Output Is Inverted | <input checked="" type="checkbox"/> |
| Asynch OR Latch | <input type="checkbox"/> |
| Signal Driving CTRIPOUTL | Filter output drives CTRIPOUTL |
| Signal Driving CTRIPL | Filter output drives CTRIPL |
| Set Low Comparator DAC Value | 512 |

図 4-57. CMPSS コンパレータの構成

図 4-58 は、過電流や低電流の場合に EPWM XBAR がトリップを行うために、対応する CMPSS CTRIPL と CTRIPH を選択することを示しています。

| | |
|----------------------------|-----------------------------------|
| MTR1_IS_TRIP_CMPSS | |
| Name | MTR1_IS_TRIP_CMPSS |
| XBAR Output | CMPSSA3_CTRIPH, CMPSSA3_CTRIPL +4 |
| Invert Output Before Latch | |
| Instance | |

- Select All
- CMPSSA0_CTRIPL
- CMPSSA0_CTRIPH
- CMPSSA1_CTRIPL
- CMPSSA1_CTRIPH
- CMPSSA2_CTRIPL
- CMPSSA2_CTRIPH
- CMPSSA3_CTRIPL
- CMPSSA3_CTRIPH
- CMPSSA4_CTRIPL
- CMPSSA4_CTRIPH
- CMPSSA5_CTRIPL
- CMPSSA5_CTRIPH
- CMPSSA6_CTRIPL
- CMPSSA6_CTRIPH
- CMPSSA7_CTRIPL
- CMPSSA7_CTRIPH
- CMPSSA8_CTRIPL

図 4-58. EPWM XBAR の構成

5 テキサス・インスツルメンツの高電圧評価基板 (TI HV EVM) におけるユーザーの安全のための一般的な指針



テキサス・インスツルメンツの設定および使用の手順に常に従い、すべてのインターフェイス コンポーネントを推奨される電氣的定格電圧および電力制限範囲内で使用してください。電気に関する安全上の注意事項に常に従い、自分自身と周囲の作業者の安全を確保してください。詳細については、テキサス・インスツルメンツの[技術問い合わせ窓口](#)までご連絡ください。

警告

警告および手順に従わないと、感電ややけどの危険により、人身傷害、物的損害、あるいは死亡事故が発生する可能性があります。

TI HV EVM という用語は、電子デバイスが通常オープン フレームの、密封されていないプリント基板アセンブリで提供されていることを意味します。テキサス・インスツルメンツの HV EVM は、開発ラボ環境で使用することを厳密に意図しており、高電圧電気回路の開発および応用における電氣的安全性の訓練を受け、技能と知識を有する有資格者のみが使用してください。その他の使用および/または応用は、テキサス・インスツルメンツにより厳密に禁止されています。資格を有していない場合は、HV EVM の使用をただちに停止してください。

1. 作業場の安全性

- a. 作業領域を清潔で整理整頓された状態に保ちます。
- b. 回路への電源投入は、必ず資格を有するオペレーターの立ち合いの下に行います。
- c. TI HV EVM およびインターフェイス電子機器に電源を投入する領域には、効果的なバリアと標識を必ず設け、不用意なアクセスがないように、アクセス可能な高電圧が存在する可能性があることを明記します。
- d. 開発環境で使用されるすべてのインターフェイス回路、電源、評価基板、計器、メーター、スコープ、およびその他関連の装置で 50Vrms/75VDC を超えるものは、緊急電源遮断 EPO で保護された電源タップ内に電氣的に配置する必要があります。
- e. 安定した非導電性の作業台を使用します。
- f. 適切に絶縁されたクランプおよびワイヤを使用して測定用プローブおよび計器を接続します。可能な限りフリーハンドテストは行わないでください。

2. 電氣的安全性

- 予防措置として、EVM 全体が完全にアクセス可能で、アクティブな高電圧が印加されていると想定します。
- a. 電気測定またはその他の診断測定を行う前に、テキサス・インスツルメンツの HV EVM およびすべての入力、出力、電気負荷の電源を遮断します。TI HV EVM の電源が安全に切断されていることを再確認します。
 - b. EVM の電源が切断されていることを確認した上で、EVM 回路および測定装置が電氣的に導通していると想定して、必要な電気回路構成、配線、測定装置の接続、およびその他の応用ニーズを実施します。
 - c. EVM の準備が整ったら、意図されたように EVM に電源を投入します。

警告

EVM に電源が投入されている間、EVM または電気回路に触らないでください。高電圧により感電の危険性があります。

3. 個人の安全

- a. 個人用保護具 (ゴム手袋やサイドシールド付き保護メガネなど) を身につけ、EVM を適切なインターロック付きの透明のプラスチック箱に入れて保護するなどして、不用意に触ることがないようにします。

安全使用の制限:

EVM は、量産ユニットのすべてまたは一部として使用することを意図していません。

6 設計とドキュメントのサポート

6.1 デザイン ファイル

6.1.1 回路図

BOOSTXL-3PHGANINV の回路図をダウンロードするには、[BOOSTXL-3PHGANINV](#) のデザイン ファイルを参照してください。

TMDSHVMTRINSPIN の回路図をダウンロードするには、[C2000WARE-MOTORCONTROL-SDK](#) の `<install_location>\solutions\tmdshvmtrinspin\hardware` フォルダにあるハードウェア ファイルを参照してください。

6.1.2 BOM (部品表)

BOOSTXL-3PHGANINV の部品表 (BOM) をダウンロードするには、[BOOSTXL-3PHGANINV](#) のデザイン ファイルを参照してください。

TMDSHVMTRINSPIN の部品表 (BOM) をダウンロードするには、[C2000WARE-MOTORCONTROL-SDK](#) の `<install_location>\solutions\tmdshvmtrinspin\hardware` フォルダにあるハードウェア ファイルを参照してください。

6.1.3 PCB レイアウトに関する推奨事項

6.1.3.1 レイアウト プリント

BOOSTXL-3PHGANINV のレイアウト プリントをダウンロードするには、[BOOSTXL-3PHGANINV](#) のデザイン ファイルを参照してください。

TMDSHVMTRINSPIN のレイアウト プリントをダウンロードするには、[C2000WARE-MOTORCONTROL-SDK](#) の `<install_location>\solutions\tmdshvmtrinspin\hardware` フォルダにあるハードウェア ファイルを参照してください。

6.2 ツールとソフトウェア

ツール

- [TMDSCNCD263](#) TMDSCNCD263 は、HSEC180 controlCARD をベースとする、Sitara™ 高性能マイコンである AM263x シリーズ向けの評価 / 開発ツールです。このボードは、新しいアプリケーションを開発するための標準化された使いやすいプラットフォームを実現するために、初期評価とプロトタイプ製作用に設計されています。
- [Code Composer Studio™](#) Code Composer Studio™ IDE は完全統合スイートで、テキサス・インスツルメンツのすべての組み込みプロセッサ (Sitara、DSP、車載、KeyStone)、マイクロコントローラ (SimpleLink™、C2000 デジタル制御、MSP430、TM4C、Hercules)、デジタル電源 (UCD) デバイス、プログラマブル ゲイン アンプ (PGA) デバイスのアプリケーションの作成とデバッグを行うことができます。
- [ARM-CGT-CLANG](#) tiarmclang コンパイラ ツールは、コンパイラ、アセンブラ、リンカなどのソフトウェア開発ツールを提供しています。これらのツールを使用すると、Arm Cortex-M シリーズや Cortex-R シリーズのコア プロセッサにロードして実行するための、C/C++ ソース コードによるアプリケーションを開発できます。
- [SYSCONFIG](#) SysConfig は、ハードウェアとソフトウェアの構成に関する課題の簡素化と、ソフトウェア開発の迅速化に役立つ設計を採用した構成ツールです。SysConfig は、ピン、ペリフェラル、無線、ソフトウェア スタック、RTOS、クロック ツリーなどのコンポーネントを構成するための直観的なグラフィカル ユーザー インターフェイスを採用しています。SysConfig は、ソフトウェア開発を迅速化するために、競合の検出、表示、解決を自動的に実行します。

ソフトウェア

- [MCU-PLUS-SDK-AM263X](#) AM263x MCU+ SDK (マイコン + ソフトウェア開発キット) は テキサス・インスツルメンツの組み込みプロセッサ向けの統合ソフトウェア プラットフォームであり、セットアップが容易で、サンプルとベンチマークとデモをすぐに利用できます。

**MOTOR-
CONTROL-
SDK-AM263X**

MOTOR CONTROL SDK for AM263x には、RTOS ベースと no-RTOS (RTOS を使用しない) ベースのアプリケーションを開発するためのサンプル、ライブラリ、ツールが付属しており、モーターからの位置センスのリアルタイム通信、Arm R5F CPU や関連ペリフェラルのリアルタイム制御ライブラリが可能です。

6.3 ドキュメントのサポート

1. テキサス・インスツルメンツ:『[モーター制御 SDK ユニバーサル プロジェクトおよびラボ](#)』、ユーザー ガイド
2. テキサス・インスツルメンツ:『[AM263x Sitara™ マイクロコントローラ](#)』データシート
3. テキサス・インスツルメンツ:『[テキサス・インスツルメンツ製品ファミリの AM263x Sitara™ マイクロコントローラ](#)』、テクニカル リファレンス マニュアル
4. テキサス・インスツルメンツ:『[AM263x controlCARD ハードウェア](#)』、ユーザー ガイド

6.4 サポート・リソース

テキサス・インスツルメンツ E2E™ サポート・フォーラムは、エンジニアが検証済みの回答と設計に関するヒントをエキスパートから迅速かつ直接得ることができる場所です。既存の回答を検索したり、独自の質問をしたりすることで、設計に必要な支援を迅速に得ることができます。

リンクされているコンテンツは、各寄稿者により「現状のまま」提供されるものです。これらはテキサス・インスツルメンツの仕様を構成するものではなく、必ずしもテキサス・インスツルメンツの見解を反映したものではありません。テキサス・インスツルメンツの[使用条件](#)を参照してください。

6.5 商標

LaunchPad™, BoosterPack™, controlCARD™, テキサス・インスツルメンツの™, Sitara™, Code Composer Studio™, and テキサス・インスツルメンツ E2E™ are trademarks of Texas Instruments.

Arm® and Cortex® are registered trademarks of Arm Limited.

EtherCAT® and PROFINET® are registered trademarks of Beckhoff Automation GmbH.

MATLAB® is a registered trademark of The MathWorks, Inc.

すべての商標は、それぞれの所有者に帰属します。

7 著者について

Masoud Farhadi は、車載アプリケーション専用 MCU のシステム エンジニアで、半導体業界の最重要ニーズの設計に携わっています。テキサス大学ダラス校を卒業し、電気工学 (パワー エレクトロニクス) の博士号を取得しています。パワー エレクトロニクス システムの設計に熱心に取り組み、最新の特許動向を活用して EV テクノロジーの発展に努めています。

重要なお知らせと免責事項

テキサス・インスツルメンツは、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、テキサス・インスツルメンツ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テキサス・インスツルメンツ製品の選定、(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テキサス・インスツルメンツ製品を使用するアプリケーションの開発の目的でのみ、テキサス・インスツルメンツ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。テキサス・インスツルメンツ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、テキサス・インスツルメンツおよびその代理人を完全に補償するものとし、テキサス・インスツルメンツは一切の責任を拒否します。

テキサス・インスツルメンツの製品は、[テキサス・インスツルメンツの販売条件](#)、または [ti.com](https://www.ti.com) やかかるテキサス・インスツルメンツ製品の関連資料などのいずれかを通じて提供する適用可能な条項の下で提供されています。テキサス・インスツルメンツがこれらのリソースを提供することは、適用されるテキサス・インスツルメンツ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。

お客様がいかなる追加条項または代替条項を提案した場合でも、テキサス・インスツルメンツはそれらに異議を唱え、拒否します。

郵送先住所：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2025, Texas Instruments Incorporated