XMC™ MCU を使用した PMSM FOC モーター制御

XMC1300/XMC1400 および XMC4400/XMC4800

本書について

適用範囲と目的

このアプリケーションノートでは、インフィニオン XMC1302, XMC1402, XMC1404, XMC4400, または XMC4800 マイクロコントローラーを使用した、3 相モーター用の永久磁石同期モーター (PMSM) のフィールド指向制御 (FOC) モーター制御ソフトウェアの実装について説明します。

対象者

本書は、XMC シリーズマイクロコントローラーを使用した、センサーレスフィードバックを備えた FOC 制御用の設定可能なシステムを設計したいお客様を対象とします。

参考資料

[1] XMC1300 AB-Step Reference Manual, XMC1000 Family

[2] XMC1400 AA-Step Reference Manual, XMC1000 Family

[3] XMC4400 Reference Manual

[4] XMC4800 Reference Manual

目次

本書について 1
目次 2
1 はじめに 4
1.1 主な機能 5
1.2 略語と頭字語 6
1.3 XMC™ リソースの割り当て 7
1.4 XMC™ハードウェアモジュールの相互接続性 9
1.5 実行時間とメモリ消費 11
1.6 ソフトウェアの概要 12
1.7 PMSM FOC ソフトウェアの使用制限 14
2 PMSM FOC センサーレスソフトウェアコンポーネント 15
2.1 モーター始動/速度変更/モーター停止制御 16
2.2 ランプジェネレーター 17
2.3 制御方式 18
2.3.1 オープンループ電圧制御 18
2.3.2 速度制御 18
2.3.3 トルク制御ダイレクト起動 20
2.3.4 Vq 制御ダイレクト起動 21
2.3.5 D 軸と Q 軸のデカップリング 21
2.4 直交座標から極座標への変換 22
2.5 空間ベクトル変調 (SVM) 23
2.5.1 7 セグメント SVM 24
2.5.2 5 セグメント SVM 25
2.5.3 擬似ゼロベクトル (PZV) 26
2.5.4 4 セグメント SVM 27
2.5.5 過変調 SVM 29
2.6 DC Link 電圧 30
2.7 クラーク変換 30
2.8 パーク変換 32
2.9 保護 33
2.10 スケーリング 33
2.11 磁束とトルク電流の PI ゲインの決定 37
3 電流検出と計算 40
3.1 シングルシャント電流検出 (XMC1300/XMC1400 のみ) 42
3.2 3 シャント電流検出 45
3.2.1 非同期変換 (XMC1300/XMC1400 のみ) 47
3.2.2 同期変換 47
3.2.3 同期測定実装 50
4 センサーレス FOC 制御におけるモーター速度と位置のフィードバック 52
5 割込み 54
5.1 PWM ピリオドマッチ割込み 54
5.2 CTrap 割込み 56
5.3 ADC ソース割込み (XMC1300/XMC1400 のみ) 56
5.4 セカンダリーループ割込み 56
5.5 過電圧/低電圧保護 56
6 モーターステートマシン 57
7 コンフィギュレーション 60

目次

7.1 ユーザー設定 ... 60
7.1.1 概要 ... 60
7.1.2 カスタムキットの設定 ... 63
7.1.3 高度なユーザー設定 ... 65
7.1.4 トルク制御仕様 ... 66
7.1.5 VQ 制御仕様 (V/f) ... 67
7.1.6 MET 仕様 ... 67
7.2 ハードウェア設定 ... 68
7.2.1 コントローラーカード ... 68
7.2.2 電流および電圧検出用 インバーターボード設定 ... 75
7.2.3 モーター固有の設定 ... 78
7.2.3.1 モーターパラメーター ... 78
7.2.3.2 PI 設定 ... 79
8 PMSM FOC ソフトウェア データ構造体 ... 82
8.1 FOC 制御モジュールの入力データ構造体 ... 82
8.2 FOC 制御モジュールの出力データ構造体 ... 83
8.3 FOC 制御モジュールのデータタイプ ... 83
8.4 SVM モジュールデータ構造体 ... 84
8.5 電流検出ソフトウェア モジュールのデータの構造体 ... 84
9 PMSM FOC ソフトウェア API 関数 ... 86
9.1 ユーザーファンクション ... 87
9.2 コントロールループ ISR ... 90
9.2.1 起動 ... 90
9.2.2 入出力処理 ... 93
9.2.3 概要 ... 96
9.2.4 コントローラー ... 97
9.3 セカンダリーループ ISR ... 100
9.4 FPU ライブラリ ... 101
9.4.1 ライブラリ理論 ... 101
9.4.2 ライブラリ API ... 103
10 リソース ... 105
改訂履歴 ... 106
免責事項 ... 107

1 はじめに

このソフトウェアは、PMSM をセンサーレスモードまたはセンサーモードで駆動する機能を提供します。汎用ドライバとしての各種モードに必要なすべての共通モジュールが含まれており、さまざまなセグメントに対応するための高レベルの設定とモジュール性を提供します。

フィールド指向制御 (FOC) は、周波数と振幅が容易に制御して電流を最小限に抑えることができる 3 相の正弦波信号を生成するモーター制御方式です。これにより効率が最大化されます。基本的な考え方は、3 相信号をローター回転に沿う 2 つの信号に変換する、また、その逆を行うことです。

FOC モーター制御では、ローター位置とローター速度のフィードバックが必要です。フィードバックは、センサーレス FOC、またはセンサー付き FOC、のいずれでも得られます。

- センサーレス FOC では、モーターの構造, モーター各相に印可される電圧, およびモーターの 3 相電流に基づいて、ローターの位置とローターの速度を導出します。
- センサー付き FOC では、ホールセンサーやエンコーダーなどのローターセンサーからローターの位置とローターの速度を得ます。

相電流は、モーターの各相、レッグシャントまたはローサイド MSOFET の DC-Link シャントにより測定できます。このソフトウェアでは、レッグシャント、もしくは DC-Link シャントでの相電流検出を想定しています。

Figure 1 に、シングルシャント、または 3 シャントのローサイド電流検出がサポートされている PMSM FOC モーター制御の一般的なブロック図を示します。

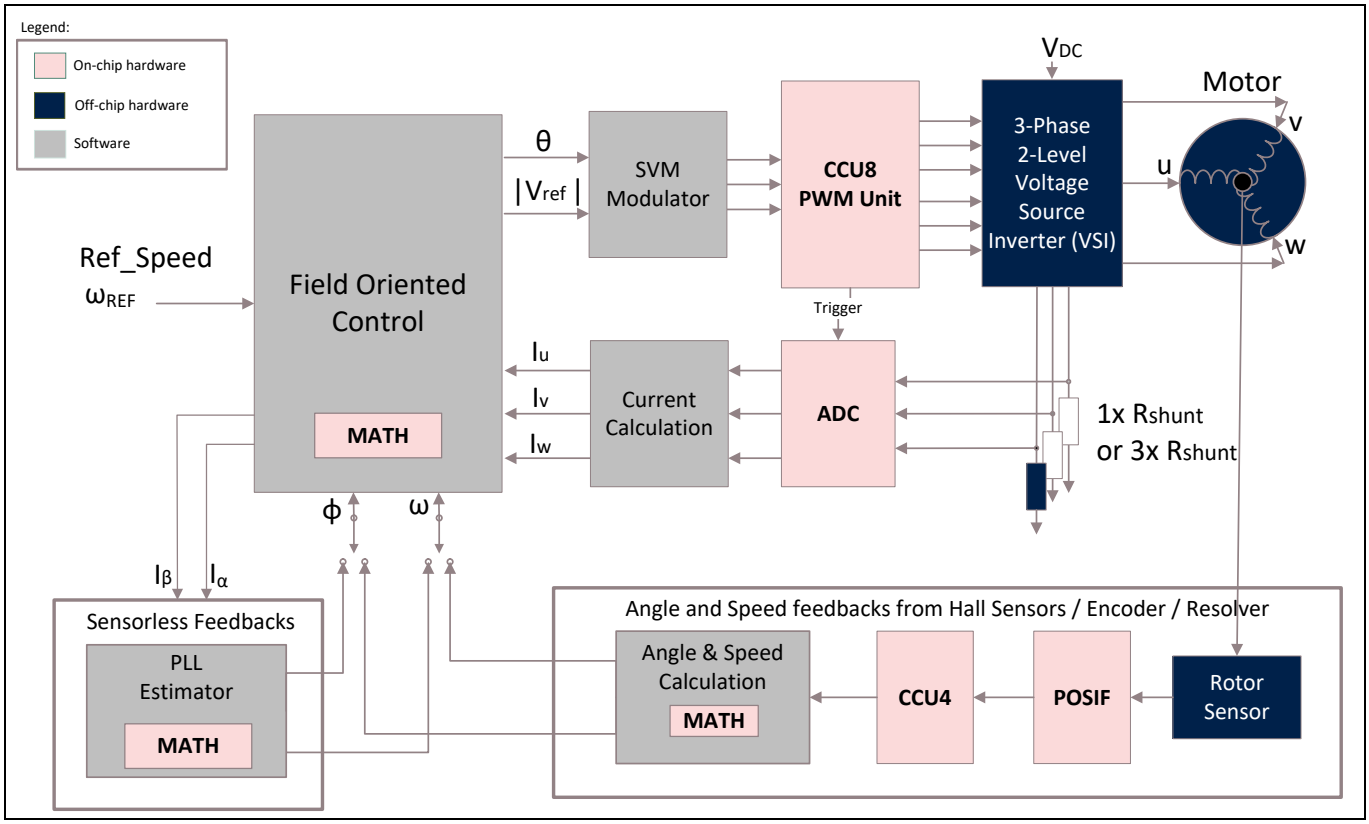


Figure 1 PMSM モーター FOC 制御のブロック図

1.1 主な機能

センサーレス PMSM FOC ソフトウェアには、次のようなインフィニオンのイノベーションと独自の機能が含まれます。

- 最適化された FOC
  - 逆パーク変換なし
  - 外部オペアンプをなくすことでコストを削減
- シングルシャント電流検出用の擬似ゼロベクトル (PZV) を備えた空間ベクトル変調 (SVM)
- V/f オープンループから FOC クローズドループへのスムーズな遷移を実現する最大効率トラッキング (MET)
- PLL エスティメーター; ローターの速度と位置のフィードバックに 1 つのモーターパラメーター (ステーターのインダクタンス, L) のみを必要とするセンサーレスフィードバック機構

サポートされている主な機能を次の表に示します。

Table 1 サポートされている主なソフトウェア機能

特長	
Math 制御ブロック	クラーク変換
	パーク変換
	Id,Iq 電流磁束/トルク PI コントローラー
	速度 PI コントローラー
	直交座標から極座標への変換
	ランプ機能
制御方式	速度制御
	トルク制御
	Vq 制御
空間ベクトル変調	5 セグメント SVM
	7 セグメント SVM
	擬似ゼロベクトル SVM; 4 セグメント SVM
ローサイド電流検出	レッグシャント: 3 または 2 をサポート、過変調および 7 セグメント SVM をサポート
	DC Link シングルシャント: PZV および 4 セグメント SVM をサポート、過変調のサポートはありません
起動アルゴリズム	ダイレクト FOC 起動
保護	相電流の過電流保護
	DC Link の低電圧と過電圧保護
デバイスの特長	電流検出用の ADC オンチップゲイン
	同期 ADC 変換: モーター相電流検出
制御機能	モーター制御ステートマシン
	急ブレーキ時の DC バス電圧クランプ
	モーターストップ/ブレーキ
ローターの速度と角度の計算	ハードウェア CORDIC を用いてのセンサーレス PLL エスティメーター
その他	S 字カーブランプジェネレーター/リニアランプジェネレーター
	速度制御の PI アンチワインドアップ
	Dq 軸デカップリング

1.2 略語と頭字語

Table 2 本書で使用する略語と頭字語

用語	定義
API	application programming interface (アプリケーション プログラミング インターフェース)
BOM	bill of material (部品表)
CCU8	Capture and Compare Unit (キャプチャ&コンペアユニット)
CPU	central processing unit (中央演算処理装置)
FOC	field oriented control (フィールド指向制御)
GPIO	general purpose input/output (汎用入出力)
IP	intellectual property (知的財産)
ISR	Interrupt service routine (割込みサービスルーチン)
LLD	low level driver (ローレベルドライバ)
MCU	microcontroller unit (マイクロコントローラー ユニット)
MET	maximum efficiency tracking (最大効率トラッキング)
PI	proportional integral controller (比例積分コントローラー、PI コントローラー)
PLL	phase locked loop (位相同期ループ)
PMSM	permanent magnet synchronous motors (永久磁石同期モーター)
PWM	pulse width modulation (パルス幅変調)
PZV	pseudo zero vector (擬似ゼロベクトル)
SRAM	static random access memory (スタティック RAM)
SVM	space vector modulator (空間ベクトル変調器)
UART	universal asynchronous receiver transmitter (ユニバーサル非同期送受信器)
VADC	versatile analog-to-digital converter (多機能アナログ-デジタル変換器)
XMC	Arm®ベース XMC™ MCU ファミリー
XMC1000	Arm® Cortex®-M0 コアベース XMC™ MCU ファミリー
XMC1300	XMC™ MCU シリーズ、32 MHz コア、64MHz ペリフェラル周波数
XMC1302	特定の機能セットを備えた XMC™ MCU 製品 例:CORDIC
XMC1400	XMC™ MCU シリーズ、48 MHz コア、96 MHz ペリフェラル周波数
XMC1402	特定の機能セットを備えた XMC™ MCU 製品 例: CORDIC
XMC1404	特定の機能セットを備えた XMC™ MCU 製品 例: CORDIC、CAN
XMC4400	特定の機能セットを備えた XMC™ MCU 製品 例:FPU
XMC4800	144 MHz のコアと周辺周波数を備えた XMC™ MCU シリーズ

1.3 XMC™ リソースの割り当て

XMC1302, XMC1402, XMC1404, XMC4400, および XMC4800 は、いずれも PMSM FOC モーター制御システムに最適なマイクロコントローラーです。これらの製品は、モーター制御用のペリフェラルとして、POSIF, CCU8, ADC, および CCU4 を備えています。XMC1300/XMC1400 では、CORDIC コプロセッサが数学計算に使用され、一方 XMC4400 および XMC4800 では浮動小数点ユニット (FPU) が使用されます。PMSM FOC モーター制御ソフトウェアにおいて使用するハードウェアペリフェラルを次の表に示します。

Note: デフォルトのリソース割り当ては、インフィニオン XMC1000/XMC4400/XMC4800 Motor Control Application Kit 用 (注文番号は、それぞれ KIT_XMC1X_AK_MOTOR_001, KIT_XMC4400_DC_V1, EVAL-XMC4800PSOC6M5) です。

Table 3 3 シャント電流検出でのセンサーレス PMSM FOC に使用される XMC™ MCU ペリフェラル

XMC™ペリフェラル	使用方法	デフォルトのリソース割り当て (XMC1300 または XMC1400)	デフォルトのリソース割り当て (XMC4400)	デフォルトのリソース割り当て (XMC4800)
CCU80	U 相の PWM 信号生成	CCU80 スライス 0	CCU80 スライス 0	CCU81 スライス 0
	V 相の PWM 信号生成	CCU80 スライス 1	CCU80 スライス 1	CCU81 スライス 1
	W 相の PWM 信号生成	CCU80 スライス 2	CCU80 スライス 3	CCU81 スライス 2
	ADC トリガー用タイマー	CCU80 スライス 3	CCU80 スライス 2	CCU81 スライス 3
VADC	U 相の電流検出	G0 チャネル 4 [1] G1 チャネル 3 [1]	G0 チャネル 1	G0 チャネル 0
	V 相の電流検出	G0 チャネル 3 [1] G1 チャネル 2 [1]	G1 チャネル 7	G1 チャネル 0
	W 相の電流検出	G0 チャネル 2 [1] G1 チャネル 4 [1]	G2 チャネル 1	G2 チャネル 4
	エイリアス チャネル (3 シャント同期変換用)	G0 チャネル 0 G0 チャネル 1 G1 チャネル 0 G1 チャネル 1	G0 チャネル 0 G1 チャネル 0 G2 チャネル 0	G0 チャネル 0 G1 チャネル 0 G1 チャネル 1
	DC Link 電圧検出	G1 チャネル 5	G1 チャネル 1	G1 チャネル 2
	DC Link 平均電流検出	G1 チャネル 6	G1 チャネル 0 (実装されていません)	G0 チャネル 6 (実装されていません)
	速度変更用ポテンショメーター	G1 チャネル 7	G1 チャネル 5	G1 チャネル 3
MATH		CORDIC	FPU	FPU
ネスト型ベクタ割込み	PWM ピリオドマッチ割込み	CCU80.SR0	CCU80.SR0	CCU81.SR0
	CTrap 割込み	CCU80.SR1	CCU80.SR3	CCU81.SR3

[1] 同期サンプリングを実行するためには、同じ入力ピンを両方の ADC グループ チャネルに接続する必要があります。詳細については、3.2.1 を参照してください。

XMC™ペリフェラル	使用方法	デフォルトのリソース割り当て (XMC1300 または XMC1400)	デフォルトのリソース割り当て (XMC4400)	デフォルトのリソース割り当て (XMC4800)
コントローラー (NVIC)	シングルシャント電流検出用 ADC 割込み	VADC0.G1SR1 [2]	使用不可	使用不可

[2] DC Link 電流検出が VADC Group0 のチャネルを使用する場合、VDC0.G0SR1 割込みノードを使用します。

1.4 XMC™ハードウェアモジュールの相互接続性

XMC1000, XMC4400 および XMC4800 ファミリーでは、包括的なハードウェアの相互接続性を備えています。

次の図に、XMC1302 ハードウェアペリフェラルモジュール間の相互接続を示します。これは、XMC1402 と XMC1404 においても当てはまります。

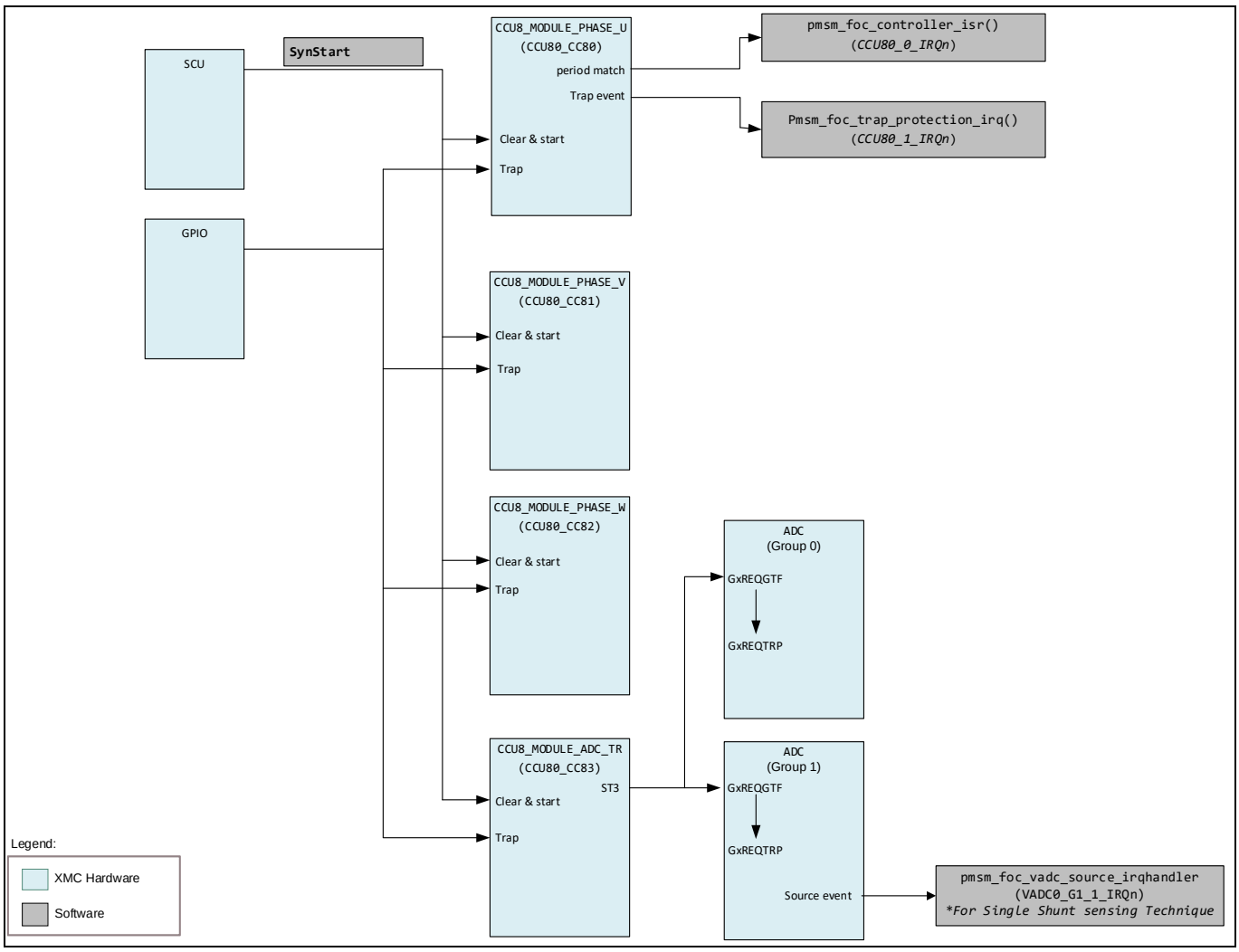


Figure 2 ハードウェアの相互接続

Note:

1. *CCU8 スライスタイマーは、SCU (system control unit) からの同期開始信号に同期してスタートします。*
2. *VADC 変換は、CCU8 スライス 3 のコンペアマッチステータス信号によりトリガーされます。*

Figure 3 に、XMC4400 ハードウェアペリフェラルモジュール間の相互接続を示します。

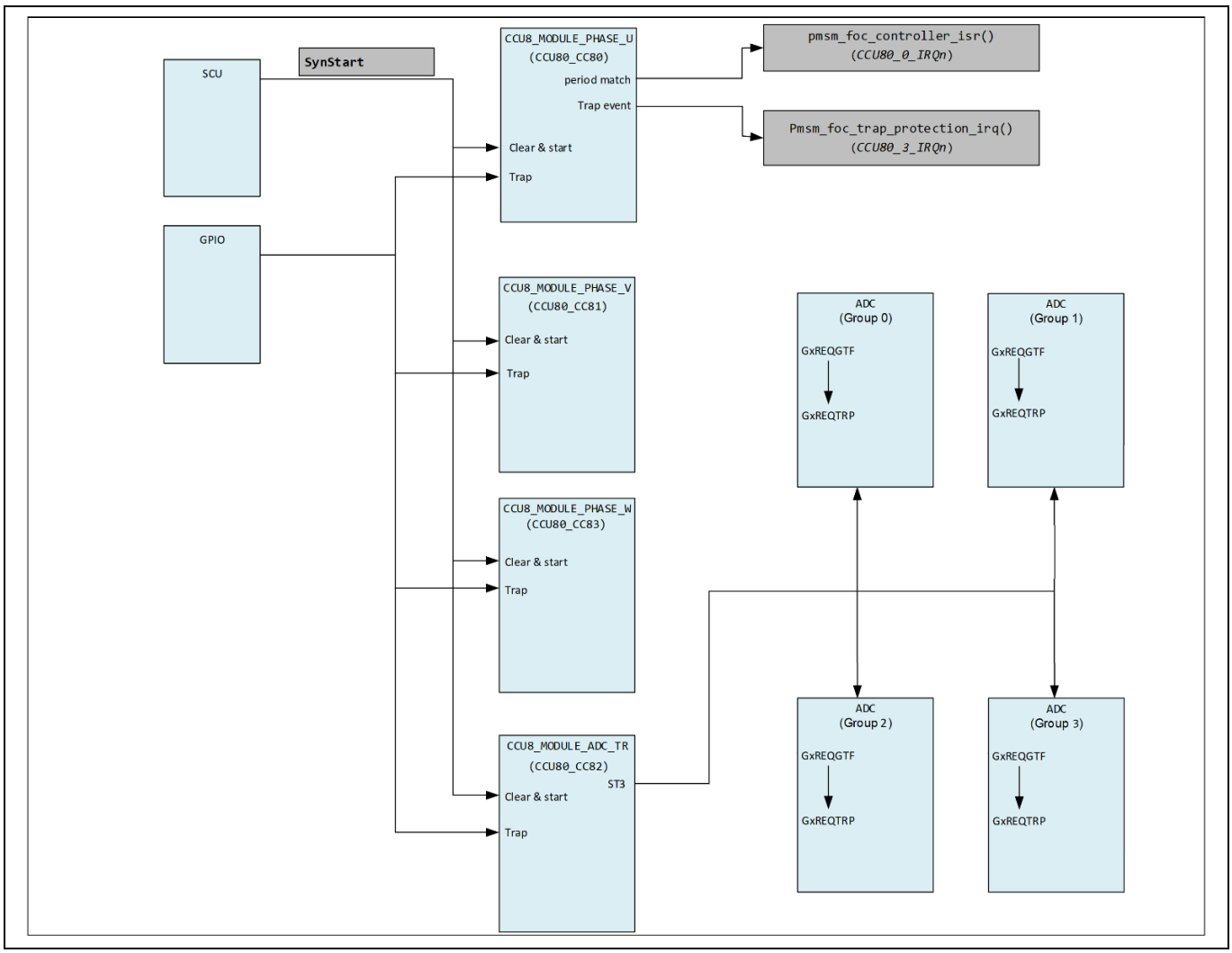


Figure 3 XMC4400 ハードウェア接続

Note:

3. *CCU8 スライスタイマーは、SCU (System Control Unit) からの同期開始信号に同期してスタートします。*
4. *VADC 変換は、CCU8 スライス 2 のコンペアマッチステータス信号によりトリガーされます。*
5. *XMC4800 は、CCU81 スライス 0, 1, 2, 3 をそれぞれ U, V, W 相および ADC トリガーに使用します。CCU81_0 および CCU81_3 は、それぞれコントローラーとトラップ保護割込みに使用されます。*

1.5 実行時間とメモリ消費

XMC1000 ファミリーと XMC4000 ファミリーにおいては SRAM へのアクセスにおける待ち時間は必要ありません。ソフトウェアの主要部分は SRAM から実行されます。割込みサービスルーチン (ISR)、FOC アルゴリズムのすべての数学的ブロック, SVM, およびモーター相電流の検出と計算は、SRAM 内で実行されます。これにより、FOC アルゴリズムの実行時間は約 30%短縮されるため、パフォーマンスが向上します。

Note: SRAM で実行される API のリストについては、9 章を参照してください。

メモリ使用量と CPU 使用時間の内訳を次の表に示します。これは、インフィニオンの XMC1000 Motor Control Application Kit (XMC1302), XMC1400 Boot Kit (XMC1404), XMC4400 Motor Control Application Kit, および XMC4800PSOC6M5 ボードのデフォルト設定に基づいています。

- 制御方式
  - オープンループ to FOC クローズドループ速度制御
- 電流検出手法
  - 3 シャント同期 ADC 変換

Table 4 XMC1300, XMC1400, XMC4400, および XMC4800 による 3 シャント電流検出での CPU 使用時間とメモリ使用量

PWM 周波数	20 kHz - ISR は 50 µs ごとに実行されます			
DAVE™ 4 GCC コンパイラ最適化レベル	最大の最適化 (-O3)			
MCU	**XMC1300@32MHz**	**XMC1400@48MHz**	**XMC4400@120MHz**	**XMC4800@144MHz**
CPU 使用率	31 µs (62%)	21 µs (42%)	12.5 µs (25%)	10.7 µs (21.4%)
フラッシュコードサイズ (バイト)	10416	10822	24790	24262
SRAM コードサイズ (バイト)	2376	2376	3060	3060
SRAM データサイズ (バイト)	348	352	176	176

1.6 ソフトウェアの概要

PMSM FOC モーター制御ソフトウェアは、明確に定義された階層型アプローチに基づいて開発されています。

階層化アーキテクチャは、モジュールをグループに分けるようにデザインされています。これにより、他のモジュールのパフォーマンスやシステム全体の構造に影響を与えることなく、特定のレイヤー内のさまざまなモジュールを置き換えることができます。

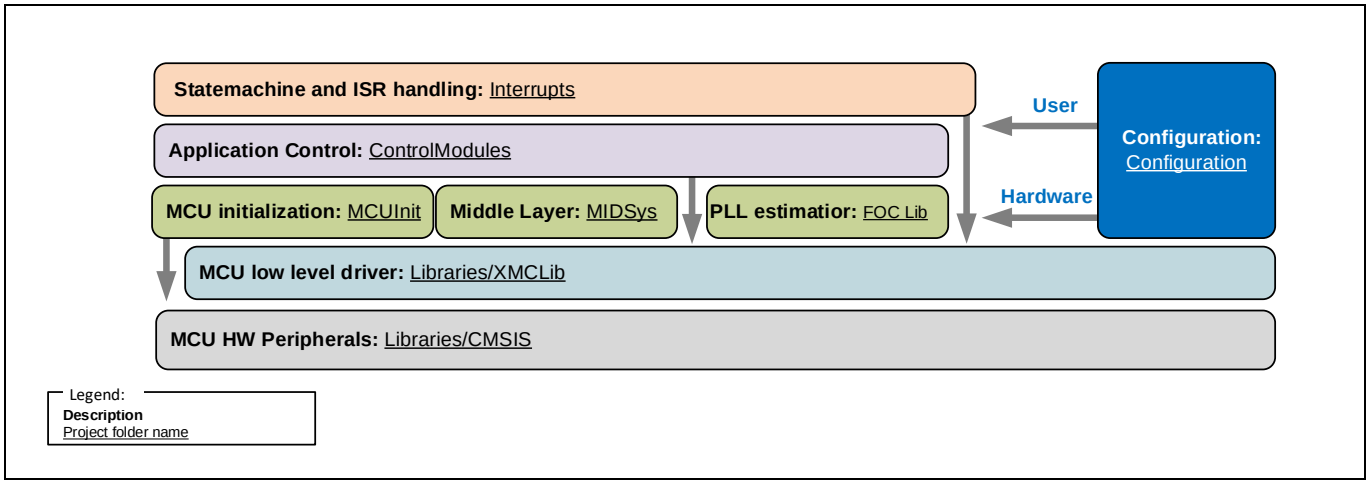


Figure 4 PMSM FOC ソフトウェアの概要: レイヤー構造

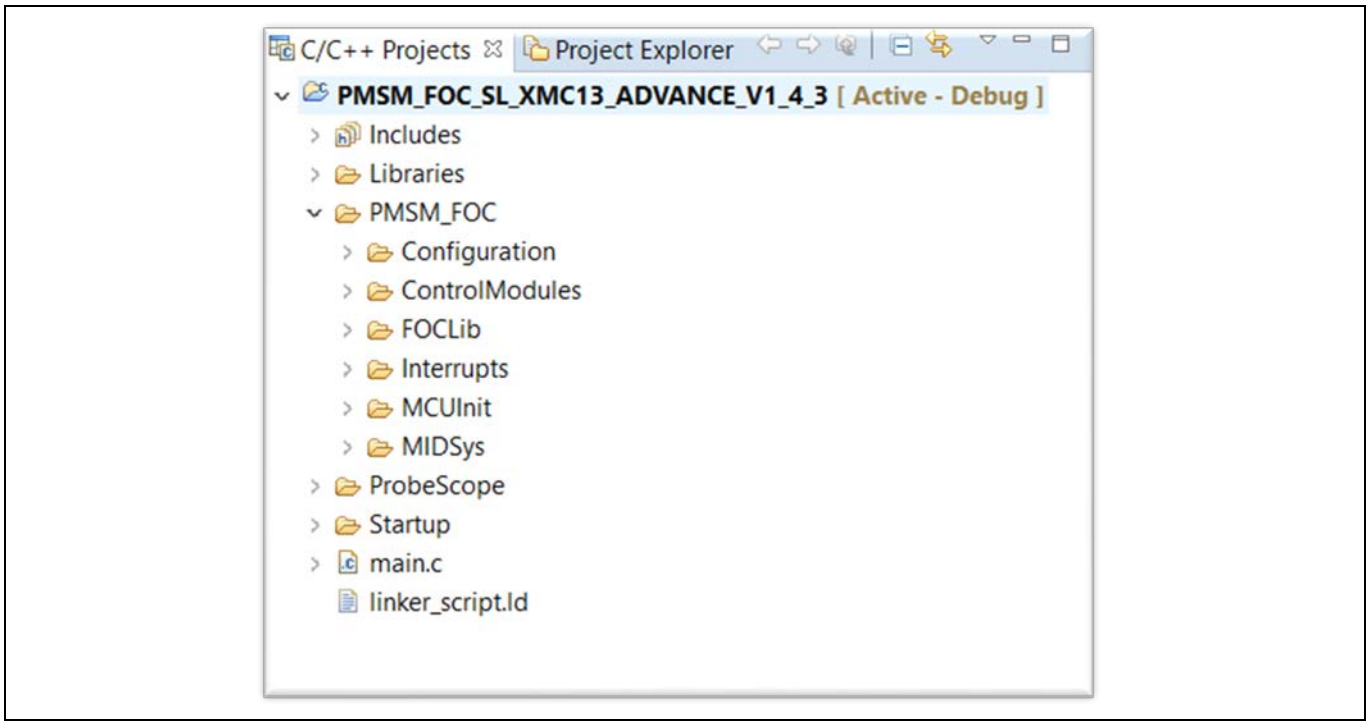


Figure 5 プロジェクトのフォルダ構造

ステートマシンと ISR 処理: Interrupts

このレイヤーは、CCU8 トラップ割込み処理機能, CCU8 ピリオドマッチ ISR 機能, およびシャント電流検出用の VADC ISR 機能で構成されています。すべてのファイルは *Interrupts* フォルダ内に保存されます。

アプリケーション制御: Control Modules

このレイヤーは、FOC SW 制御モジュールで構成されています。これには、クラーク変換, パーク変換, 直交から極座標への変換, 電流再構成, PI コントローラー, オープンループ, およびランプなどが含まれます。記載されているすべてのルーチンは、CCU80 ピリオドマッチ ISR から呼び出されます。

このレイヤーのすべてのファイルは *ControlModules* フォルダ内にあります。

設定: Configuration

ユーザー設定は、ソフトウェアの全体的な動作に影響します。*pmsm_foc_user_config.h* ファイルは編集可能です。事前に定義されたハードウェアキットが使用可能です。これらのハードウェア設定により、お客様の使用するハードウェアに対するより詳細な適合が可能です。

このレイヤーは、次のように分類されます。

- Controller card (コントローラーカード)
- Inverter card (インバーターカード)
- Motors (モーター)

特定の関連する **.h* ファイルは、関連するフォルダ内にあります。

静的な設定と要求されるスケーリングは、次のファイルからアクセスできます。

- *pmsm_foc_const.h*
- *pmsm_foc_macro.h*
- *pmsm_foc_variable_scaling.h*

Note: *これらの設定と定義ファイルを変更しないでください。*

また、すべての設定は *Configuration* フォルダ内にあります。

PLL エスティメーター: FOCLib

インフィニオンの特許取得済み IP と PLL エスティメーターは、コンパイルされたライブラリファイルとして提供されます。

このファイルは *FOCLib* フォルダ内にあります。

ミドルレイヤー: MIDSys

このレイヤーは、PWM 生成, ADC の測定, および角度と速度の情報を FOC 制御モジュールレイヤーに渡すルーチンを提供しています。このレイヤーの主な目的は、FOC ソフトウェアにフィードバックセンサーモジュールを柔軟に追加/削除できるようにすることです。例えば、ホールセンサーを使用する場合、このレイヤーにファイルを追加して、上部のレイヤーに大きな変更を加えることなく、ホールセンサーからの位置のフィードバックを得ることができます。このレイヤーは、XMC4400 および XMC4800 MCU 用の数学ライブラリも提供します。

このレイヤーのすべてのファイルは、*MIDSys* フォルダ内にあります。

MCU の初期化: MCUInit

このレイヤーは、すべての MCU ペリフェラルの初期化を担います。これには、XMCLib データ構造の初期化ファンクションとペリフェラルの初期化ファンクションが含まれています。このレイヤーは、XMCLib および MIDSys レイヤーと密接に連携して、各ペリフェラルを設定します。

このレイヤーのすべてのファイルは、フォルダ *MCUInit* 内にあります。

MCU ローレベルドライバ: Libraries/XMCLib

MCU ハードウェアペリフェラル: Libraries/CMSIS

このレイヤーは、MCU ペリフェラルに対するハードウェアアブストラクションレイヤーです。このレイヤーのすべてのファイルは、*Libraries* フォルダ内にあります。

1.7 PMSM FOC ソフトウェアの使用制限

このアプリケーションノートでは、PMSM FOC software v1.5.x を使用します。

このサンプルソフトウェアのリリース時点では、使用において次の制限があります。

- シングルモータードライブのみサポートされます。デュアルモーター制御はサポートされません。
- ホールセンサー/エンコーダーからの位置と速度のフィードバックはサポートされません
- このソフトウェアは DAVE™バージョン 4 で開発されています。他の IDE プラットフォームではテストされていません。
- 以下については、現時点において文書化されていません。
  - `pmsm_foc_set_motor_target_torque()`のスケーリング
  - `pmsm_foc_set_motor_target_voltage()`のスケーリング
  - PT1 フィルター
- 電流測定用の T_{min} は扱えません。
- ドライバの遅延設定はサポートされません。この設定値は、ドライバ IC の遅延を補正するために ADC トリガーをシフトする目的で使用されます。例えば、このシフトにより、3 シャントの測定が PWM パターンの中央で開始されるようになります。
- 過電流保護は、DC Link シャントを介して行われます。これによりインバーターは保護されますが、モーターの相間の電流は測定されません。
- キャッチフリーランニングの実装はできません。
- UART デバッグは XMC1300/XMC1400 でのみ使用でき、テストは行われていません。
- SETTING_TARGET_SPEED のオプション、BY_POT_ONLY と BY_UART_ONLY はテストされていません。
- XMC4400 および XMC4800 では、シングルシャント電流検出は使えません。

2 PMSM FOC センサーレスソフトウェアコンポーネント

この PMSM FOC モーター制御ソフトウェアは、センサーモジュールまたはセンサーレスモジュールで PMSM モーターを駆動する機能を提供します。現在のバージョンでは、センサーレスモジュールのサポートとなります。さまざまなモーター制御アプリケーションに対応するために、高度な設定が可能で、モジュール適応性も備えています。

次の 5 種類の制御方式がサポートされています。

- 速度制御 遷移 FOC 起動方式
- 速度制御 ダイレクト FOC 起動方式
- トルク制御 ダイレクト FOC 起動方式
- Vq 制御 ダイレクト FOC 起動方式
- オープンループ電圧制御方式

PMSM FOC ソフトウェアの主要コンポーネントを次の図に示します。

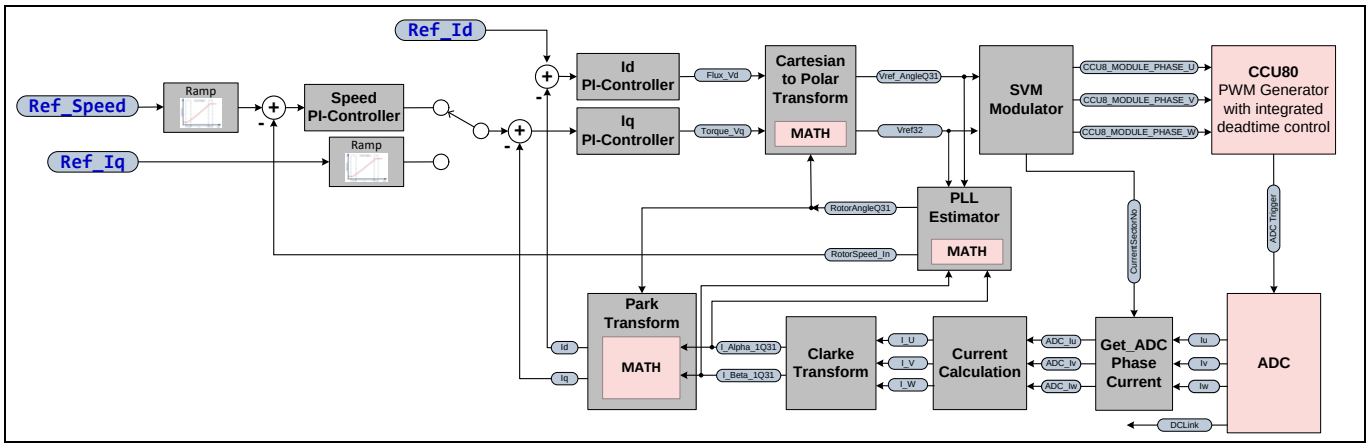


Figure 6 PMSM FOC ブロック図

2.1 モーター始動/速度変更/モーター停止制御

モーターは start コマンドで始動します。

ターゲット速度は、次の間で変更できます。

- USER_SPEED_LOW_LIMIT_RPM
- USER_SPEED_HIGH_LIMIT_RPM

ターゲット速度はポテンショメーターを使用して制御できます。

速度制御方式における ADC 値とモーターターゲット速度との関係は次のとおりです。

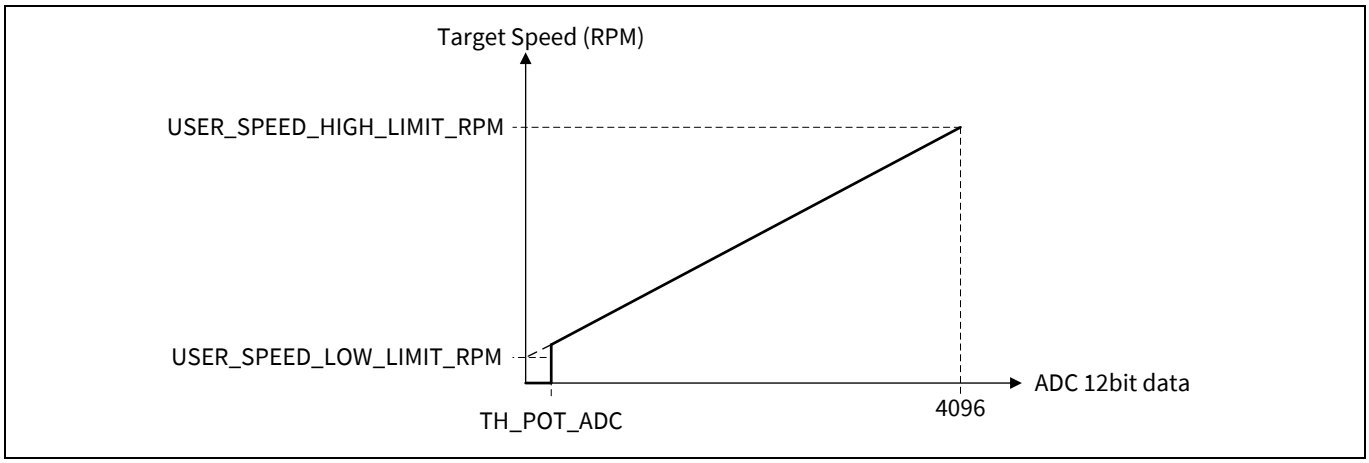


Figure 7 速度制御方式におけるポテンショメーターの ADC の値と速度の関係

FOC CLOSED LOOP ステートからモーターが始動してしまわないようにするために、3 つのオプションが使用できます。

- ターゲット速度を EXIT_FOC_SPEED 未満に設定する
- ブレーク関数 pmsm_foc_motor_break()を呼び出す
- 停止の関数 pmsm_foc_motor_stop()を呼び出す

ターゲット速度を EXIT_FOC_SPEED 未満に設定する

モーター速度が最高速度の 10%未満の場合、ステートは MOTOR_HOLD に変更されます。

PWM ON/OFF Duty 50% がすべての相に適用されます。モーターの回転は維持されます。

ブレーク関数 pmsm_foc_motor_break()を呼び出す

ソフトウェアはターゲット速度を無視し、モーターを FOC_EXIT_SPEED まで減速します。この値以下になると、ステートは MOTOR_STOP に変更されます。インバーターはディセーブル状態になり、出力はトライステートに設定されます。その結果、モーターの回転は制御不能なフリーホイーリング状態になります。ランプダウンにより、モーターの速度は非常に低くなります。

停止の関数 pmsm_foc_motor_stop()を呼び出す

ステートは即座に MOTOR STOP に変更されます。インバーターはディセーブル状態になり、出力はトライステートに設定されます。その結果、モーターの回転は制御不能なフリーホイーリング状態になります。そのときのモーター速度と摩擦に応じて、モーターはフリーホイール状態で回転します。

2.2 ランプジェネレーター

PMSM FOC モーター制御ソフトウェアは、次の 2 種類のランプ機能を提供します。

- リニアカーブ
- S 字カーブ

入力パラメーターは初期値から最終値までランプ的に増加します。ランプジェネレーター入力は、設定に応じて、ユーザー設定値またはアナログ入力に接続されます。

線形領域でのランプ増加率は、ユーザー設定ファイル pmsm_foc_motor_XXXX.h の USER_SPEED_RAMPUP_RPM_PER_S として定義されます (7.2.3.2 を参照)。ランプジェネレーター機能は、PWM 周波数サイクルごとに呼び出されます。

S 字カーブ ランプジェネレーターでは、最初のランプ増加率は定義した増加率の半分になります。その後は、定義した値までゆっくりと増加します。これにより、最初の S 字カーブが生成されます。

2 番目の S 字カーブは、速度が Ref_Speed から SPEED_TH_2ND_S になったときに開始されます。

定数 SPEED_TH_2ND_S は pmsm_foc_interface.c ファイルで定義されます。

S 字カーブ ランプジェネレーターは速度制御でのみ使用されます。

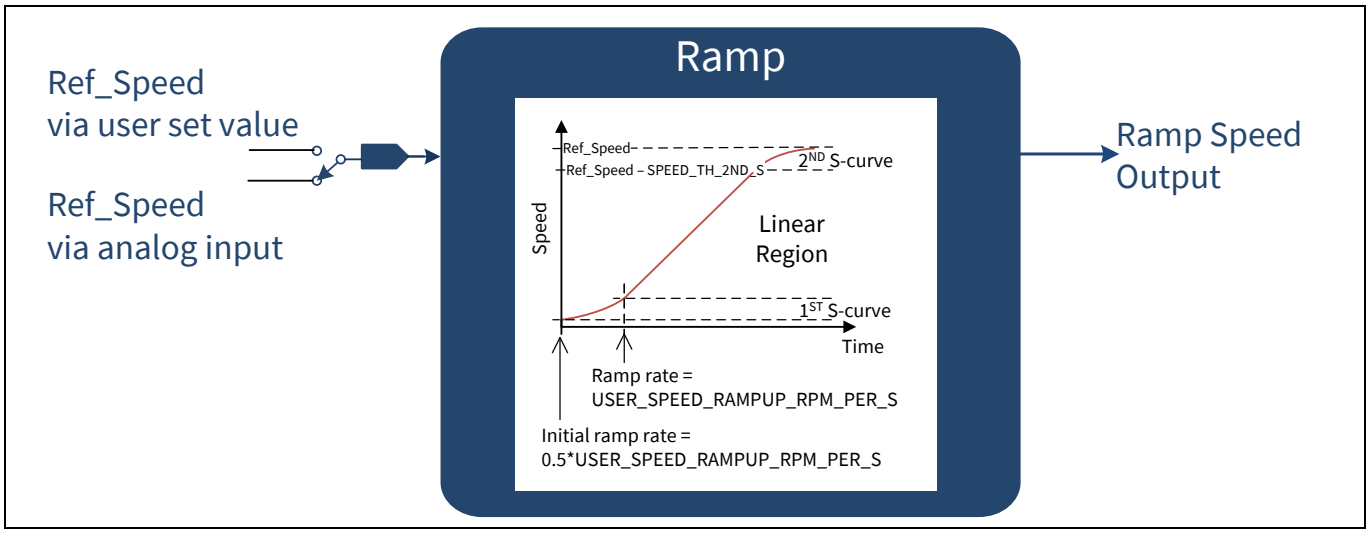


Figure 8 S 字カーブ ランプジェネレーター

いずれのランプジェネレーター機能でも、ランプダウン動作中に DC Link 電圧がモニターされます。DC Link 電圧がユーザー設定の電圧リミットを超えると、速度/トルクのランプダウンが停止します。これは、急ブレーキ時の過電圧状態を回避するためです。この電圧のリミット値 VDC_MAX_LIMIT は、ヘッダーファイル pmsm_foc_variables_scaling.h で定義されています。

ランプ出力は、制御におけるリファレンス信号になります。リニア ランプジェネレーターの場合、選択した制御方式に応じて、ランプ出力は速度, トルク電流, またはトルク Vq になります。

2.3 制御方式

このソフトウェアにおける 3 相 PMSM FOC モーターの制御方式は次のとおりです。

- オープンループ電圧制御
- 速度制御
- トルク制御
- Vq 制御

2.3.1 オープンループ電圧制御

オープンループ電圧制御では、リファレンス電圧 (Vref) に応じて、パワーインバーターがモーターに所定の電圧を印可します。機械的負荷は、PMSM モーターの速度と電流に影響を与えます。

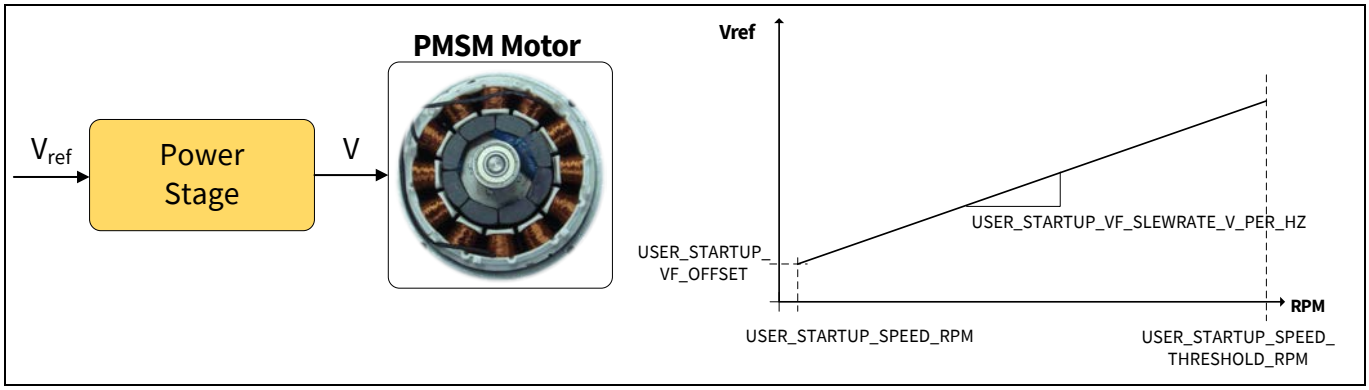


Figure 9 オープンループ電圧制御

2.3.2 速度制御

速度制御方式は、クローズドループ制御です。この方式は、速度制御と電流制御がカスケード接続された構造となっています。このとき、速度制御ループの変化に対する応答は電流ループの変化に対する応答よりもはるかに遅いです。

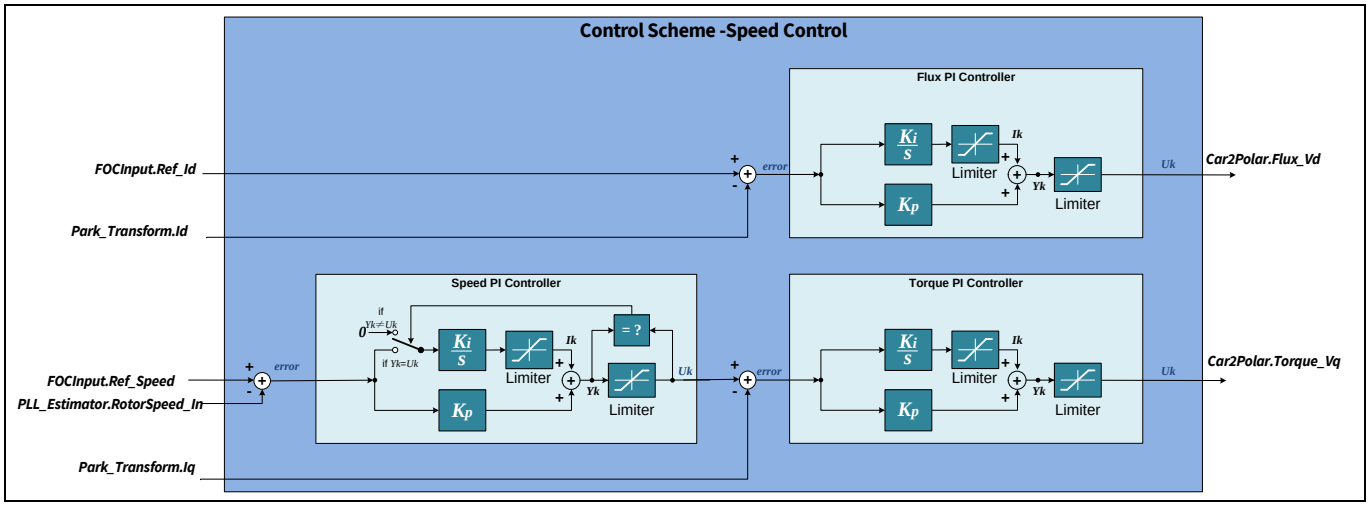


Figure 10 速度制御

速度制御では、ダイレクト FOC 起動モードと遷移起動モード (オープンループからクローズドループ) がサポートされています。

速度 PI コントローラーは、積分アンチワインドアップをサポートしています。PI 出力または積分出力のいずれかがリミットに達すると、積分出力は安定した状態に保持されます。

速度 PI コントローラーの出力は、トルク PI コントローラーのリファレンスとして使用されます。

遷移モード – maximum efficiency tracker (MET) による 3 ステップ起動

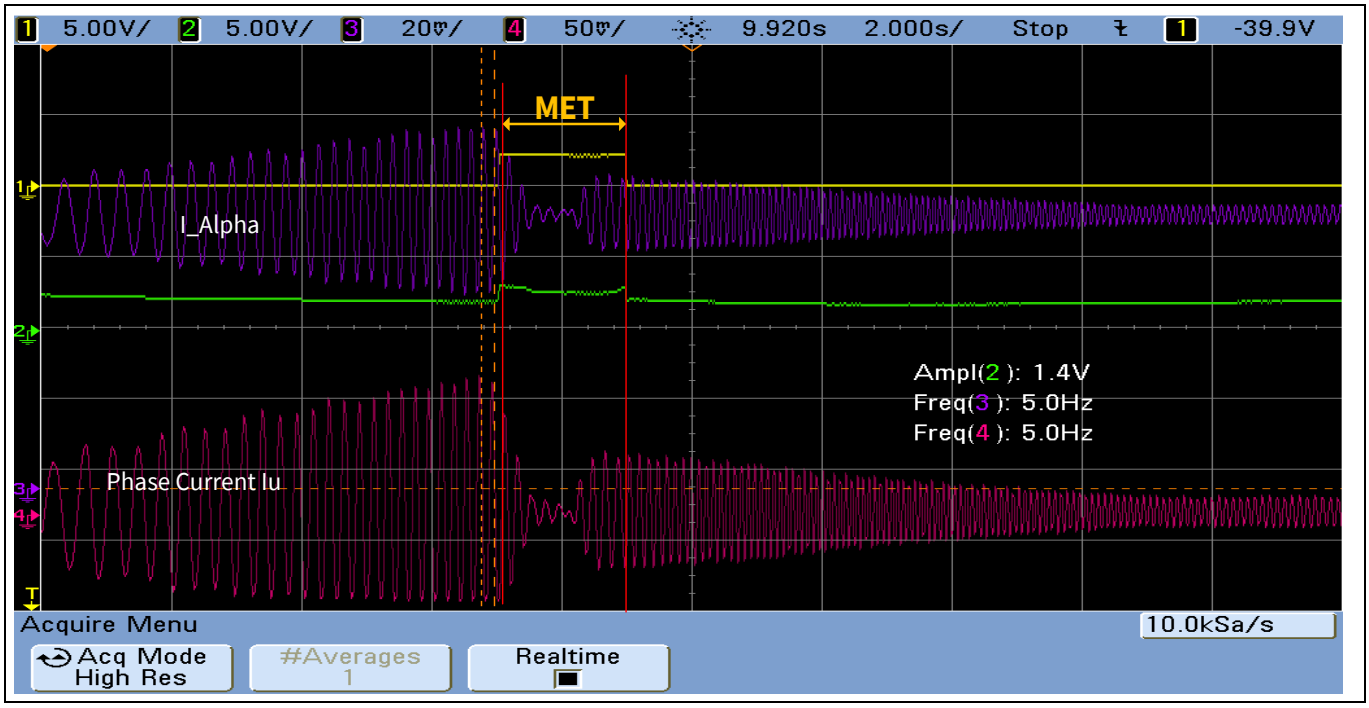


Figure 11　3 ステップモーター起動メカニズム

3 つのステップは次のとおりです。

1. モーターは V/f オープンループ制御で起動し、ユーザー設定の起動速度まで上昇します。
2. センサーレス MET クローズドループ制御がこれを引き継ぎます。この状態は、ステーター磁束がローター磁束に対してスムーズかつ制御された方法で垂直になるように追加されたものです。
3. ステートマシンは FOC_CLOSED_LOOP ステートに切り替わり、モーター速度をユーザー設定のターゲット速度まで上昇させます。

利点:

- エネルギー効率の高い MET および FOC クローズドループ
- 3 つのステップ間すべてでのスムーズな遷移
- V/f オープンループ -> 低モーター速度での MET クローズドループ、したがって起動において低電力を実現

2.3.3 トルク制御ダイレクト起動

PMSM FOC モーター制御ソフトウェアは、トルク制御ダイレクト起動を提供しています。制御ループは、d 軸 (磁束) と q 軸 (トルク) の PI コントローラーで構成されています。モータートルクはトルクリファレンス値 (Iq_ref) に維持されます。負荷が変化するとモーターの速度は変化しますが、トルクは一定のままです。

この制御方式は、電動自転車やバッテリー駆動のデバイスなど、直流の制御が重要となるアプリケーションで役立ちます。

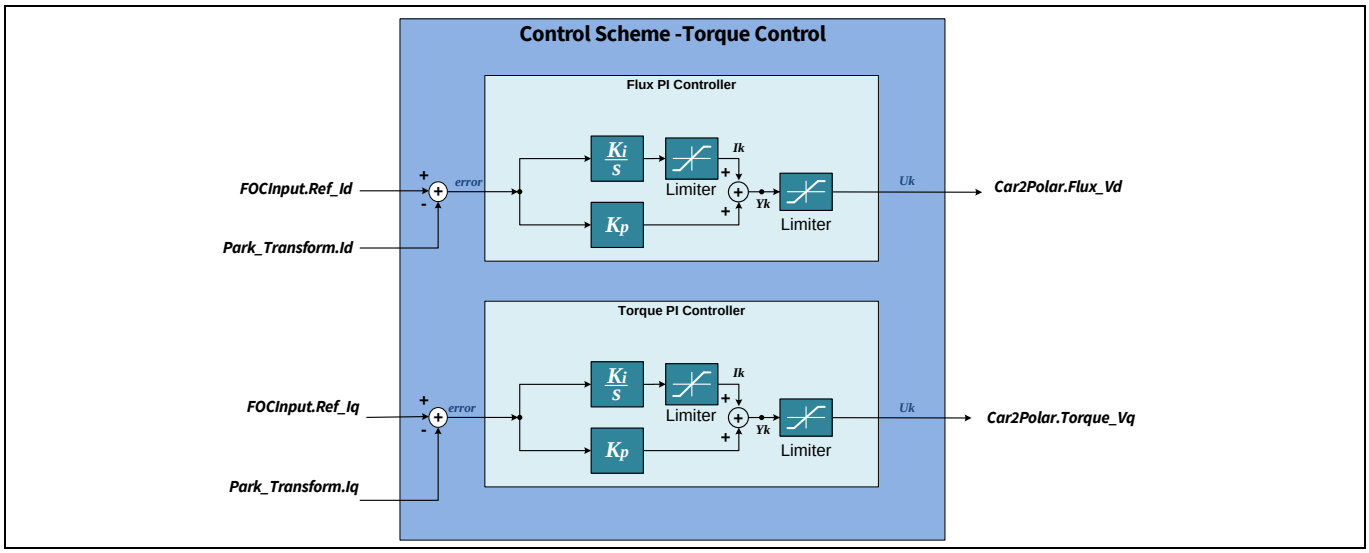


Figure 12 トルク制御方式

2.3.4 Vq 制御ダイレクト起動

Vq 制御は、高速な応答が必要であり、モーター速度の変化が問題にならない場合に使用されます。

速度 PI 制御ループとトルク PI 制御ループは無効になります。

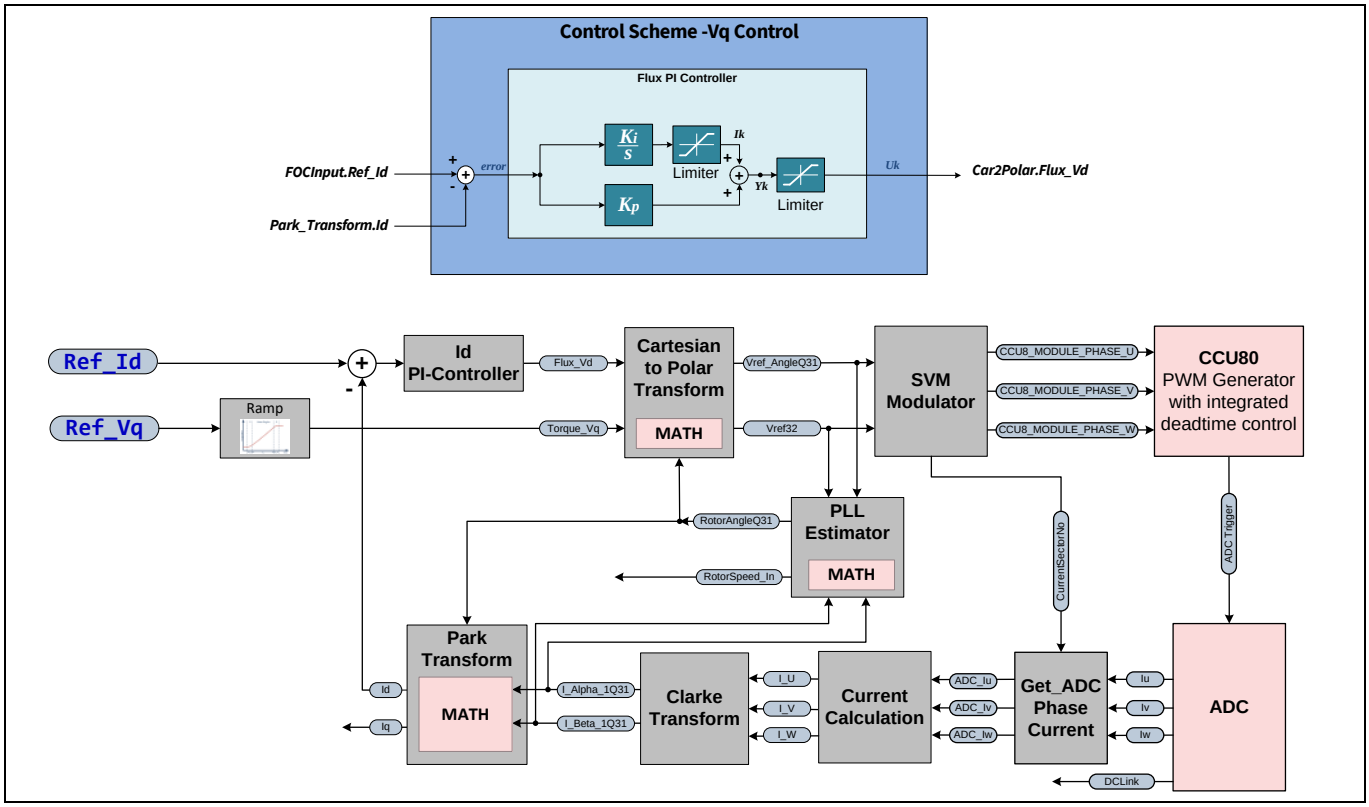


Figure 13 Vq 制御方式

2.3.5 D 軸と Q 軸のデカップリング

I_d 電流と I_q 電流の制御は、互いに独立しているわけではありません。I_d 電流は I_q 電流に影響を与え、その逆も同様です。

この相互作用は、高速での遷移状態において顕著な外乱として現れます。この相互作用の対策として、フィードフォワードデカップリングがそれぞれの軸に適用され、外乱が除去されます。

トルク成分電圧:

$$V_q = V_q - \omega L_q I_q$$

Equation 1

磁束成分電圧:

$$V_d = V_d + \omega L_d I_d$$

Equation 2

トルクインダクタンスと磁束インダクタンスが等しいと仮定すると、ωは PLL エスティメーター出力からの推定速度を表します。

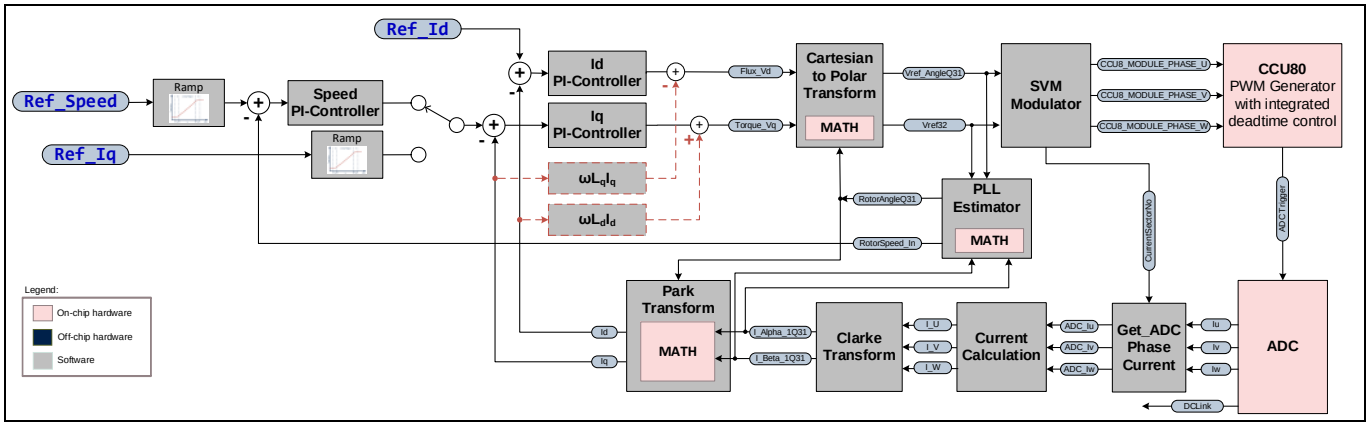


Figure 14　dq デカップリングによる FOC 制御

2.4　直交座標から極座標への変換

XMC1300/XMC1400 は、トルクおよび磁束 PI コントローラーからの出力を使用して、サーキュラーベクトルモードにてハードウェア CORDIC コプロセッサにより直交座標から極座標への変換の計算がなされます。

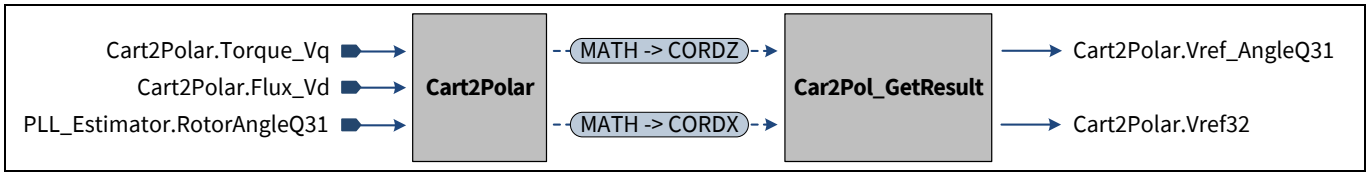


Figure 15　直交座標から極座標への変換

Table 5　直交座標から極座標への変換のための XMC1000 CORDIC 設定

パラメーター	設定
CORDIC Control Mode	Circular vectoring mode
K	≈ 1.646760258121
Magnitude Prescaler, MPS	2
CORDX	Cart2Polar.Flux_Vd
CORDY	Cart2Polar.Torque_Vq
CORDZ	PLL_Estimator.RotorAngleQ31

計算式:

$$CORDX = K * \left|V_{ref\,32}\right| / MPS = K * \sqrt{V_{Flux_Vd}{}^2 + V_{Torque_Vq}{}^2} / MPS,$$

$$\left|V_{ref\,32}\right| = CORDX * MPS / K = CORDX * 1.2145$$

$$V_{ref_AngleQ31}, \theta = CORDZ = RotorAngleQ31 + \arctan\left(\frac{V_{Torque_Vq}}{V_{Fluz_Vd}}\right)$$

Equation 3

実行性能を向上させるために、ファンクションは並列処理用に 2 つに分割されています。開始 CORDIC ファンクションと読み出し CORDIC 結果ファンクションです。CORDIC コプロセッサが計算を行っている間、CPU は PI コントローラーなどの他のソフトウェア機能を実行し、後で CORDIC の結果を読み出すことができます。代わりに XMC4400/4800 では、fpu_math2 ライブラリに実装されている `fpu_cart2polar()`関数を呼び出して、FPU ですぐに計算が (並列ではなく) 行われます。直交座標から極座標への変換の結果は、空間ベクトル変調 (SVM) モジュールに送られます。

2.5 空間ベクトル変調 (SVM)

SVM は、ステーターの電圧ベクトルをパルス幅変調 (PWM) 信号に変換します (コンペアマッチの値)。

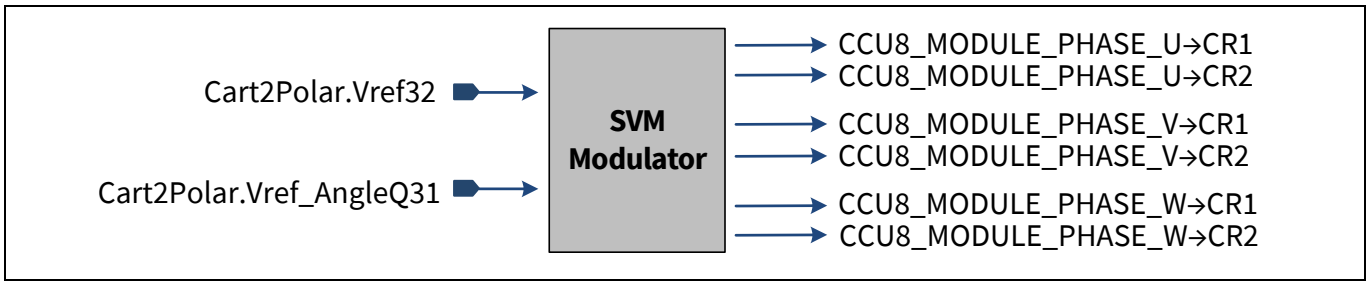


Figure 16 PWM SVM ファンクション

PMSM FOC モーター制御ソフトウェアは、SVM のさまざまなモードをサポートします。

- 7 セグメント SVM
- 5 セグメント SVM
- SVM と擬似ゼロベクトル
- 4 セグメント SVM
- 過変調 SVM

それぞれの CCU80 タイマースライスは、コンプリメンタリ出力でインバーターの各相を制御します。DC Link の短絡を防ぐためにデッドタイムが挿入されます。*pmsm_foc_invertercard_parameter.h* にてデッドタイム値は、ユーザーにより設定できます。

CCU80 で使用されているタイマーカウント方式は、アシンメトリカルエッジアライメントモードです。これは、7 セグメント SVM, 4 セグメント SVM, および疑似ゼロベクトル PWM に対して同じ CCU80 の設定となるためです。アシンメトリカルモードでは、ADC を介してシャント電流をサンプリングする際の柔軟性が向上します。

CCU80 モジュールのデフォルトの初期設定は、インフィニオンモーター制御アプリケーションキット XMC1300 KIT_XMC1X_AK_MOTOR_001, XMC4400 KIT_XMC4400_DC_V1, および XMC4800 XMC4800PSOC6M5 ボード向けです。

Table 6 CCU80 3 相 SVM 生成デフォルト初期設定

パラメーター	XMC1300/XMC1400 および XMC4400/XMC4800 の設定 [3]
Timer counting mode	Edge-aligned mode
Shadow transfer on clear	有効
Prescaler mode	中

[3] XMC4800 は、位相 U, V, W, および ADC トリガーにそれぞれ CCU81 スライス 0, 1, 2, 3 を使用します。CCU81_0 と CCU81_3 は、それぞれコントローラーとトラップ保護割込みに使用されます。

パラメーター	XMC1300/XMC1400 および XMC4400/XMC4800 の設定 [3]
Passive level	低
Asymmetric PWM	有効
Output selector for CCU80.OUTy0	Connected to inverted CC8yST1
Output selector for CCU80.OUTy1	Connected to CC8yST1
Dead-time clock control	Time slice clock frequency, f_{tclk}
Dead-time value	USER_DEAD_TIME_US*USER_PCLK_FREQ_MHZ (750 ns)
Phase U slice period match interrupt event	有効
Trap interrupt event	有効

2.5.1 7 セグメント SVM

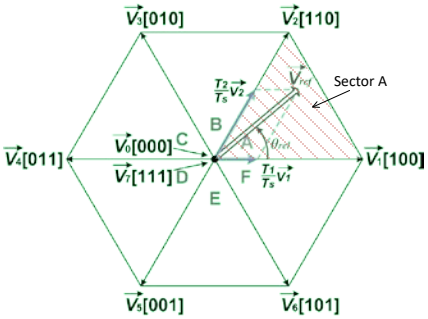


Figure 17 7 セグメント SVM

例として、セクターA の電圧空間ベクトルを取り上げます。SVM の PWM オン時間を計算するために、次の式が使用されます。

$$\vec{V}_{ref} = \frac{T_0}{T_S}\vec{V}_0 + \frac{T_1}{T_S}\vec{V}_1 + \frac{T_2}{T_S}\vec{V}_2$$

Equation 4

$$T_S = T_0 + T_1 + T_2$$

Equation 5

$$T_1 = \frac{\sqrt{3}|V_{ref}|T_S}{V_{DC}} sin\left(\frac{\pi}{3} - \theta_{rel}\right)$$

Equation 6

$$T_2 = \frac{\sqrt{3}|V_{ref}|T_S}{V_{DC}} sin(\theta_{rel})$$

Equation 7

$$T_0 = T_S - T_1 - T_2$$

Equation 8

ここで、

T_S　サンプリング期間

$\vec{V}_0$　ゼロベクトル

$\vec{V}_1$ $\vec{V}_2$　アクティブベクトル

T_0　ゼロベクトルの時間が適用されます。ゼロベクトルは$\vec{V}_0$[000], $\vec{V}_7$[111], またはその両方です。

T_1　アクティブベクトル$\vec{V}_1$の時間が 、1 回のサンプリング時間内で適用されます。

T_2　アクティブベクトル$\vec{V}_2$の時間が 、1 回のサンプリング時間内で適用されます。

T_{DC}　インバーターの DC Link 電圧

θ_{rel}　Vref と V1 の間の相対角度 ($0 \le \theta_{rel} \le \frac{\pi}{3}$)

例えば、SVM セクターA は、U 相の PWM オン時間は PWM 時間から $T_0/2$を引いた値, V 相は PWM 時間から ($T_0/2 + T_1$) を引いた値, および W 相は $T_0/2$です。

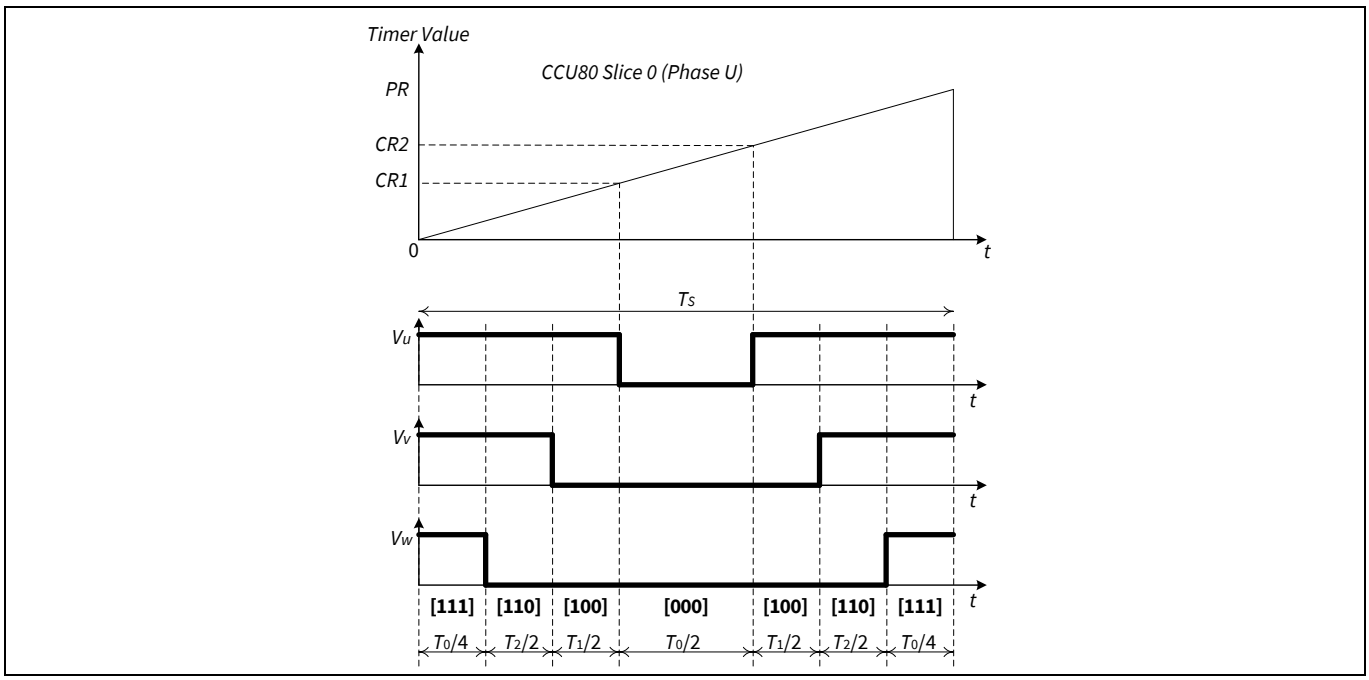


Figure 18　SVM セクターA の 7 セグメント SVM タイミング図

2.5.2　5 セグメント SVM

5 セグメント SVM は、7 セグメント SVM と同じ計算式を使用して T0, T1, および T2 のタイミングを計算します。5 セグメント SVM では、ゼロベクトルは $\vec{V}_0$[000] のみです。7 セグメント SVM とは異なり、ゼロベクトルは $\vec{V}_0$[000] および $\vec{V}_7$[111] です。

例えば、SVM セクターA は、U 相の PWM オン時間は PWM 時間からT_0を引いた値 、V 相は PWM 時間から ($T_0 + T_1$) を引いた値、W 相は 0 になります。

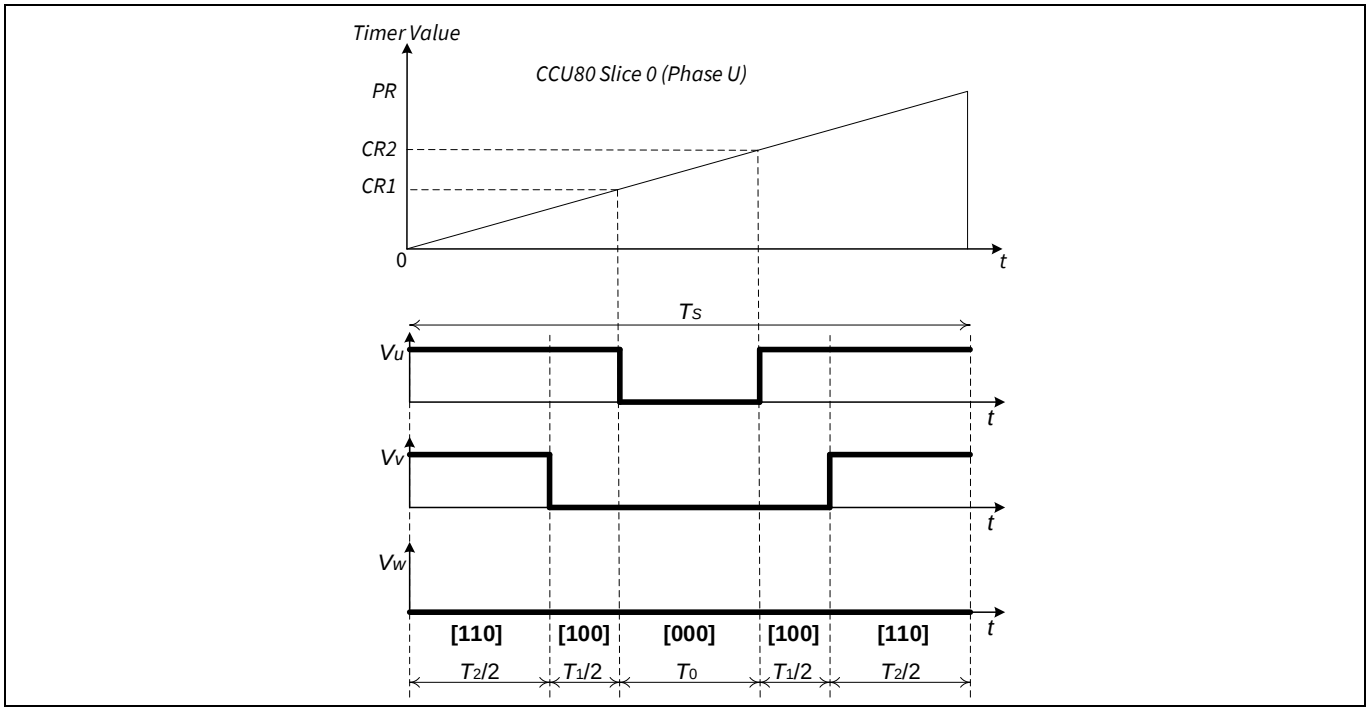


Figure 19　SVM セクターA の 5 セグメント SVM タイミング図

2.5.3　擬似ゼロベクトル (PZV)

シングルシャントの構成では、各アクティブベクトル中に、相の 1 つを流れる電流をシャント抵抗で検出します。ただし、セクター間を越える場合やベクトルの長さが短い場合など、特定の条件下では、一方または両方のアクティブベクトルの持続時間が短すぎて ($T_1 < T_{min}$ または $T_2 < T_{min}$)、信頼性の高い相電流のサンプリングが行えません。このような条件は、Figure 20 に示すように、空間ベクトル図の網掛け部分になります。

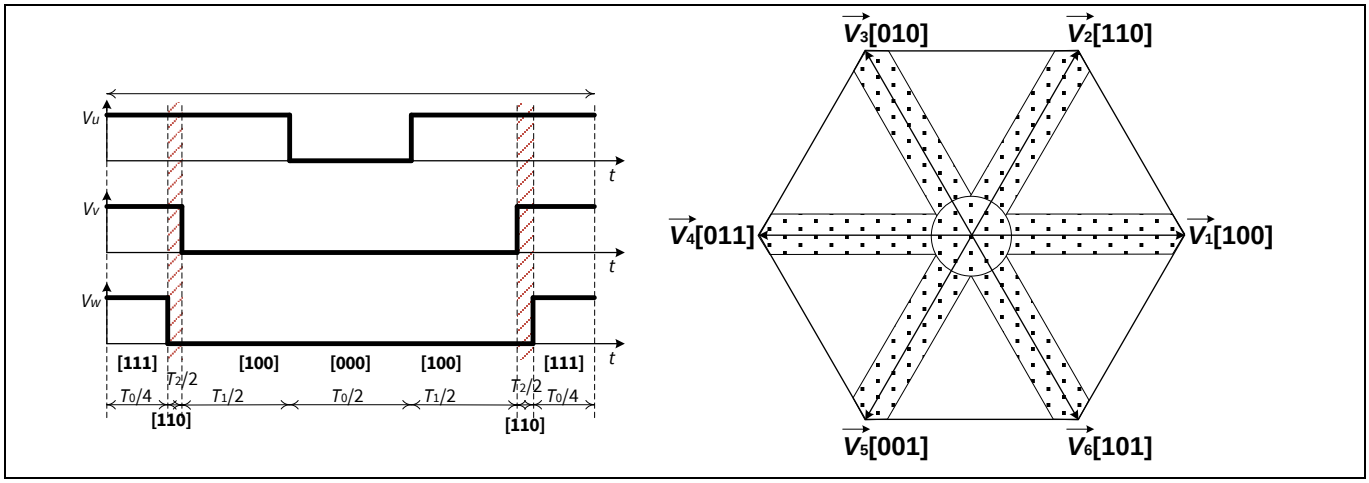
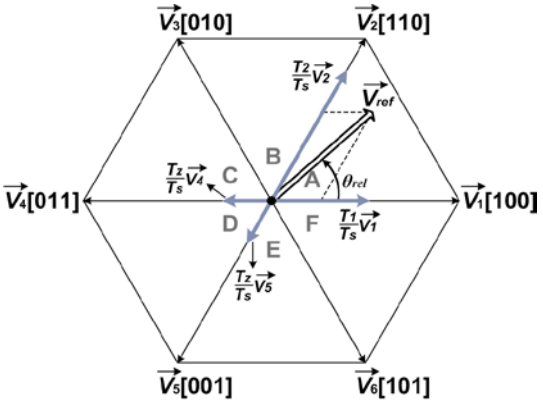


Figure 20　7 セグメント SVM - シングルシャント検出

これを解決するためには、こうした条件下では、シングルシャント電流の検出に擬似ゼロベクトル (PZV) が使用されます。擬似ゼロベクトル時間 T_Z は、相電流検出において十分な ADC サンプリング時間を確保するように調整されます。



$$T_1 = \frac{\sqrt{3}\cdot T_s \cdot |V_{ref}|}{V_{DC}} \sin\left(\frac{\pi}{3} - \theta_{rel}\right) + T_{\min} \qquad (T_Z = T_{\min})$$

$$T_2 = \frac{\sqrt{3}\cdot T_s \cdot |V_{ref}|}{V_{DC}} \sin(\theta_{rel}) + T_{\min}$$

$$T_1 + T_2 = \frac{\sqrt{3}\cdot T_s \cdot |V_{ref}|}{V_{DC}} \cdot \sin\left(\frac{\pi}{3} + \theta_{rel}\right) + 2T_{\min}$$

$$T_0 = T_s - T_1 - T_2 - 2T_{\min} = T_s - \frac{\sqrt{3}\cdot T_s \cdot |V_{ref}|}{V_{DC}} \cdot \sin\left(\frac{\pi}{3} + \theta_{rel}\right) - 4T_{\min}$$

Note: $T_{\min}$ is PWM dead time + driver delay + ADC sampling time

Figure 21　擬似ゼロベクトル

上図に、T_1 と T_2 のタイミングを計算する計算式を示します。

Figure 22 に、PZV のタイミング図と SVM 図を示します。

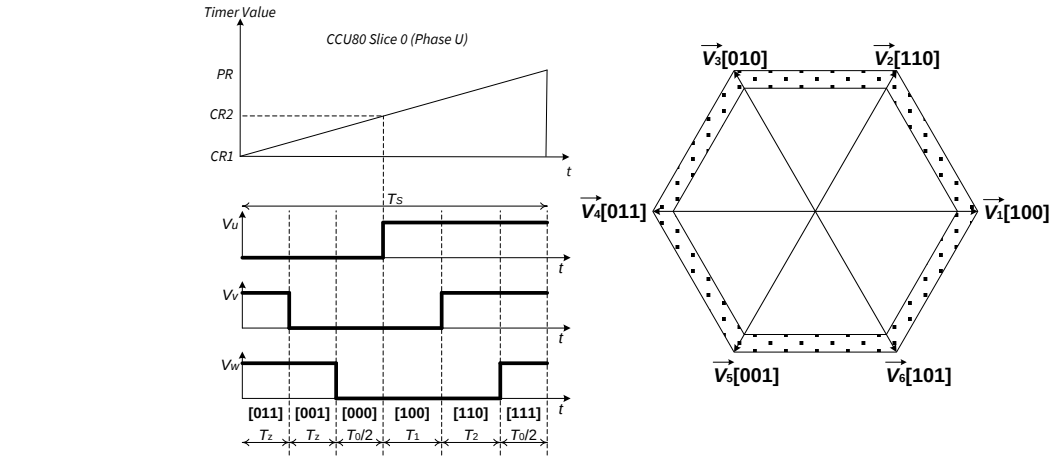


Figure 22　セクターA の PZV タイミング図

2.5.4　4 セグメント SVM

シングルシャント電流検出手法では、この SVM パターンも使用されます。PZV は、特定の条件, セクター間を越えるクロスオーバー, またはベクトルの長さが短い場合に便利です。ただし、PZV を全体で使用すると、電圧振幅の制限により、モーターが最大ターゲット速度まで回転できない場合があります。T_Z の値が大きいほど、到達可能なモーター速度は低くなります。これは、Figure 22 の SVM 図の網掛けの領域です。この問題を解決するために、4 セグメント SVM が使用されます。

PMSM_FOC ソフトウェアでは、この問題を解決するために、PZV と 4 セグメント SVM 間にて PWM 生成の移行が実装されています。T_1 と T_2 が T_{min} より大きく、モーター速度が最大モーター速度の 75%を超える場合は、4 セグメント SVM が使用されます。このようにして、最大ターゲット速度を達成します。

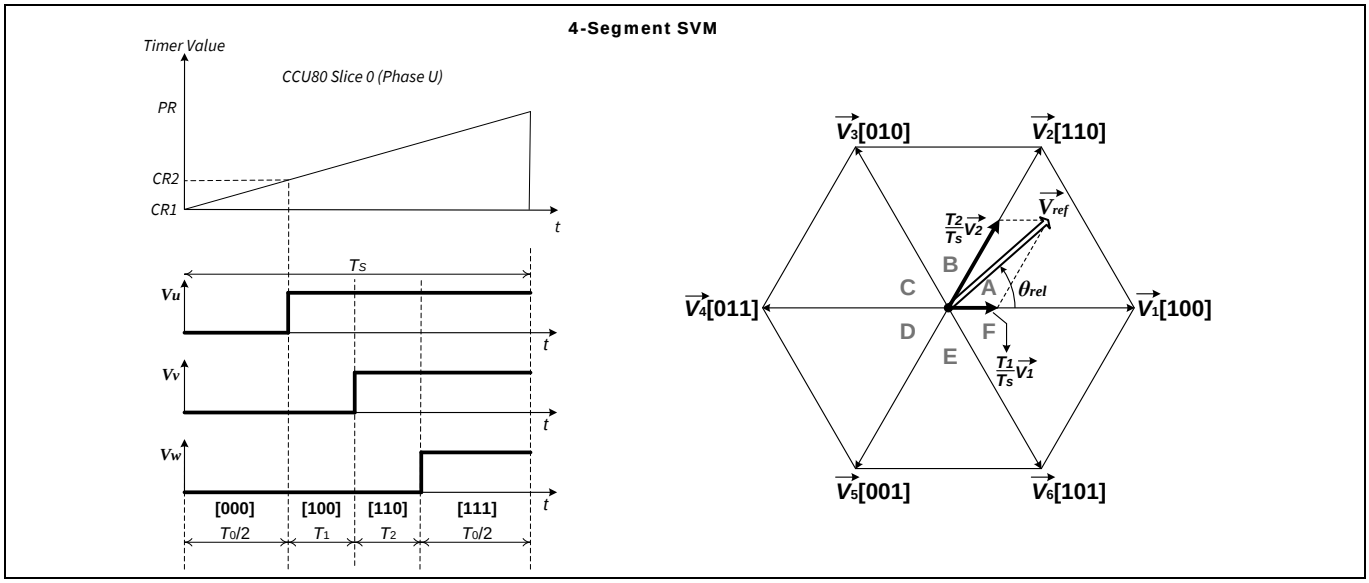


Figure 23　4 セグメント SVM

2.5.5 過変調 SVM

正弦波整流の場合、V_{ref} は最大 V_{ref} 値の 86%より小さくする必要があります。

非正弦波整流の場合、SVM の V_{ref} をより大きくできます。この手法は SVM の「過変調」と呼ばれます。

Figure 24 に、過変調が使用されている領域 (赤で塗りつぶされている部分) を示します。

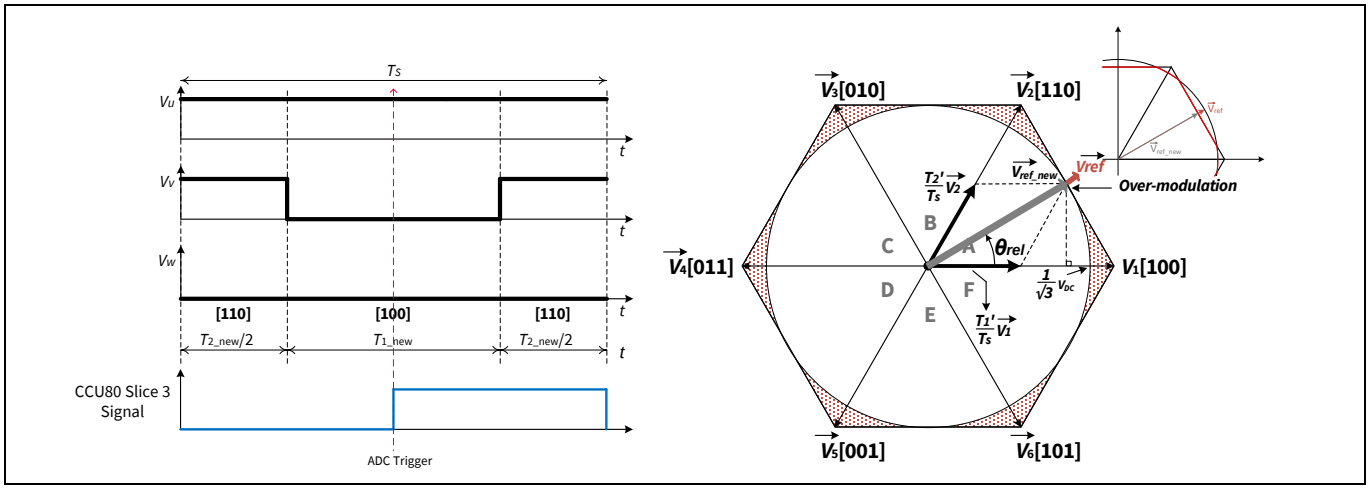


Figure 24 過変調

PMSM FOC ソフトウェアでは、V_{ref} の振幅を V_{ref_new} に減らすことで過変調が実装されます。T_1+T_2 が PWM 時間 T_S より大きい場合、ベクトルは六角形のエッジに沿って縮小されます。これは、T_1 と T_2 のタイミングを比例して減らすことによって行われます。

T_1 と T_2 は、次のように再計算されます。

$$T_{1_new} = T_1 \times \frac{T_S}{T_1+T_2}$$

Equation 9

$$T_{2_new} = T_S - T_{1_new}$$

Equation 10

過変調では、ゼロベクトルの時間はゼロに短縮されます。SVM セクターA の新たなタイミングを Figure 24 の左図に示します。この図から、2 相の電流、すなわち、I_v と I_w のみが測定できることがわかります。

モーターが高速で回転しているときは、DC バスの使用率を最大化するために過変調が使用されます。過変調の欠点は、出力電圧が正弦波ではなく、音響ノイズの原因となる高次の高調波が含まれていることです。

2.6 DC Link 電圧

DC Link 電圧は、パワーインバーターボード上の分圧器を介して測定されます。測定値は 2^12 でスケーリングされます。

電圧分圧比の値は *pmsm_foc_invertercard_parameter.h* で定義されています (7.2.2 を参照)。

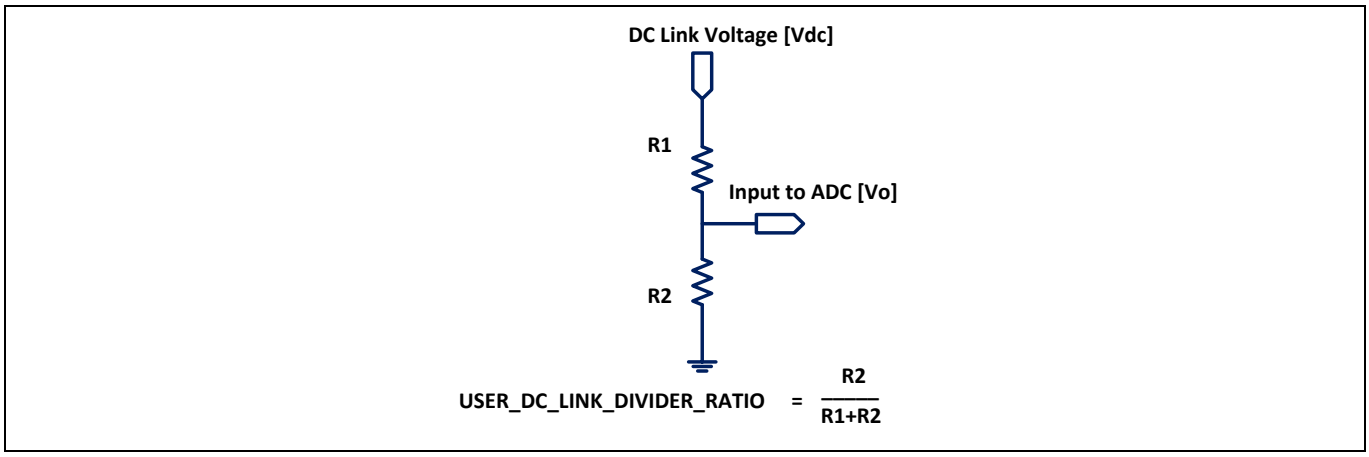


Figure 25 DC Link 電圧分圧器

2.7 クラーク変換

このモジュールでは、電流検出モジュールから得られる相電流 ($I_{I_u}, I_{I_v}, I_{I_w}$) が、2 相直交リファレンスフレーム上の *I_Alpha* および *I_Beta* 電流に変換されます。

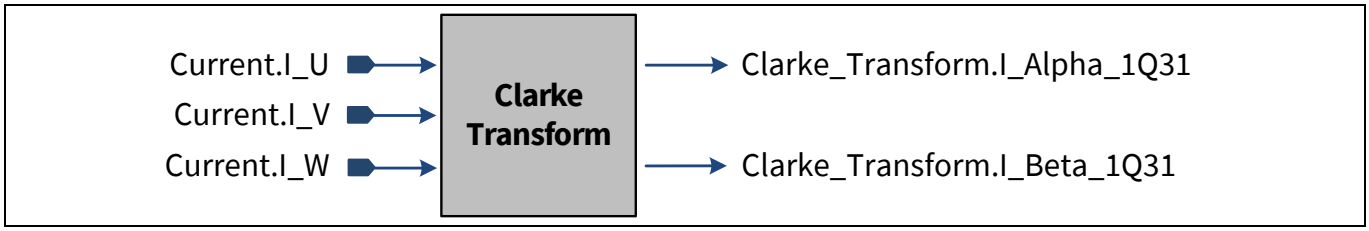


Figure 26 クラーク変換

クラーク変換式:

$$I_{\alpha} = I_{I_u}$$

Equation 11

$$I_{\beta} = \frac{1}{\sqrt{3}} \cdot I_{I_u} + \frac{2}{\sqrt{3}} \cdot I_{I_v} = \frac{1}{\sqrt{3}} \cdot \left(I_{I_u} + 2 \cdot I_{I_v}\right)$$

Equation 12

ここで、

$$I_{I_u} + I_{I_v} + I_{I_w} = 0$$

Equation 13

電流 I_U, I_V, および I_W のスケーリングファクターは、電流のスケーリングに基づきます (2.10 を参照)。XMC1300/XMC1400 では、XMC™ CORDIC ハードウェアモジュールのコード最適化により、クラーク変換ファンクションの出力は 14 ビット左にシフトされます (CORDIC_SHIFT)。

2.8 パーク変換

パーク変換では、電流 I_Alpha と I_Beta は、ローター角度の回転直交フレームに分解されます。

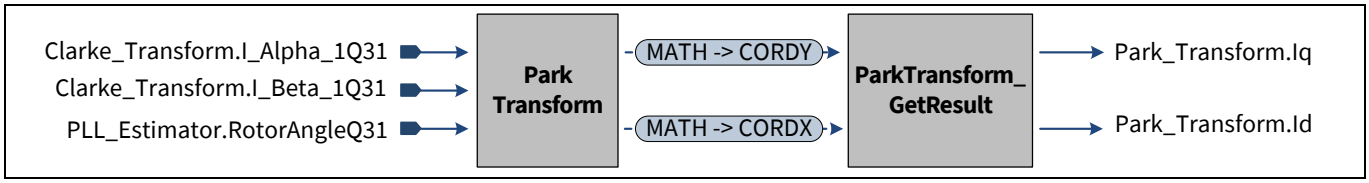


Figure 27 パーク変換

XMC1300/XMC1400 では、パーク変換ファンクションは CORDIC コプロセッサにより計算がなされます。

$$I_q = \left(-I_\alpha \cdot \sin(RotorAngle) + I_\beta \cdot \cos(RotorAngle)\right) * CORDIC_GAIN$$

Equation 14

$$I_d = \left(I_\alpha \cdot \cos(RotorAngle) + I_\beta \cdot \sin(RotorAngle)\right) * CORDIC_GAIN$$

Equation 15

Note: CORDIC_GAIN = K/MPS

入力 `PLL_Estimator.RotorAngleQ31` は、XMC™ CORDIC ハードウェアモジュール用に最適化されたコードにより、14 ビット左にシフトされます (XMC1300 または XMC1400 リファレンスマニュアル [1] を参照)。

I_q と I_d のスケーリングファクターは、電流スケーリングに基づきます (2.10 を参照)。

Table 7 パーク変換の XMC1000 CORDIC 設定

パラメーター	設定
CORDIC control mode	Circular rotating mode
K	≈ 1.646760258121
Magnitude prescaler, MPS	2
CORDX	`Clarke_Transform.I_Beta_1Q31`
CORDY	`Clarke_Transform.I_Alpha_1Q31`
CORDZ	`PLL_Estimator.RotorAngleQ31`

XMC4400/4800 では、パーク変換ファンクションは、*fpu_math2* ライブラリに実装されている `fpu_park_q31()`関数を使用して FPU によって計算されます。

2.9 保護

PMSM FOC モーター制御ソフトウェアは、次の保護機能をサポートしています。

- CCU80 CTrap 機能
- 過電流保護
- 過電圧/低電圧保護

CCU80 CTrap 機能

CCU8 モジュールのトラップ機能は、ハードウェアの過負荷状態に対する保護を提供します。CTrap 入力ピンは、ゲートドライバのフォールトピンに接続されています。ゲートドライバが障害を検出すると、CTrap ピンはアクティブ状態となり、PWM 出力は PASSIVE レベルになります。CTrap 割込みがトリガーされます。ISR では、ゲートドライバをディセーブルにし、モーターのステートは TRAP_PROTECTION に設定されます。

過電流保護 (XMC1300/XMC1400 のみ)

DC Link シャント抵抗を流れる平均電流は、PWM サイクルごとにサンプリングされます。この値は、過電流状態を検出するために読み出されます。この状況が発生すると、モーターリファレンス速度は、電流がユーザー設定ファイルで定義されたリミット (USER_IDC_MAXCURRENT_A) 内に収まるまで、所定の係数でスケールダウンされます。

変数 FOCInput.overcurrent_factor は、次の式を使用してモーター速度を更新するために使用されます。

$$FOCInput.Ref_Speed = \frac{Motor.Ref_{Speed} * FOCInput.overcurrent_factor}{4096}$$

Equation 16

平均 DC Link 電流がリミットを超えると、FOCInput.overcurrent_factor は減少します。

FOCInput.overcurrent_factor のノミナル値は 4096 です。平均 DC Link 電流がリミット内になると、FOCInput.overcurrent_factor の値はノミナル値に戻ります。XMC4400/4800 では、この機能はサポートされません。

過電圧/低電圧保護

DC Link 電圧は連続して AD 変換されます。DC Link 電圧が指定のユーザーリミットを下回るか上回る場合に割込みが発生し、pmsm_foc_over_under_voltage_isr () 関数が呼び出されます。ゲートドライバは、モーターの駆動を停止するためにディセーブル状態になります。CCU8 タイマーはまだ動作しており、モーターステートは DC_LINK_UNDER_VOLTAGE または DC_LINK_OVER_VOLTAGE ステートに切り替わります。

電圧の保護チェックは、モーターのランプ中およびオープンループ状態では行われません。

2.10 スケーリング

PMSM FOC ソフトウェアは、整数を使用して、角度, 電流, および電圧などの実際の浮動小数点変数を表します。最適な分解能とするために、ソフトウェアはターゲット値に応じ、それらの物理量をスケーリングした値を用います。

例えば、相電流はターゲット値の 0 ～ 100%で表されます。ここで、ターゲット値は、電流検出回路で測定できる最大電流です。

次の式は、物理量をソフトウェアで表される標準値に変換する計算式を示します。

$$Norm\ Value = \frac{Physical\ Value * 2^N}{Target\ Value}$$

Equation 17

Table 8　PMSM FOC ソフトウェアで使用されるスケーリング

パラメーター	スケーリング		範囲	
	ターゲット値 [単位]	N	[%]	[16 進]
SVM 振幅 (V_{ref})	$N_{_Vref_(SVM)}$ [V]	15	0% ～ 100%	0x0 ～ 0x7FFF
電流 I_U, I_V, I_W, I_q, I_d	$N_{_I_(\alpha,\beta)}$ [A]	15	-100% ～ 100%	0x8001 ～ 0x7FFF
ローター位置の角度、空間ベクトルの角度	360° [度]	16 + USER_RES_INC	0 ～ 360°	0x0 ～ 0xXFFFF (16 + USER_RES_INC ビット)
速度	N_{max_speed} [度/秒]	$\log_2(max_speed_integer)$	0% ～ 100%	0x0 ～ max_speed_interger

以下のセクションでは、ターゲット値の計算について説明します。

SVM 電圧のスケーリング

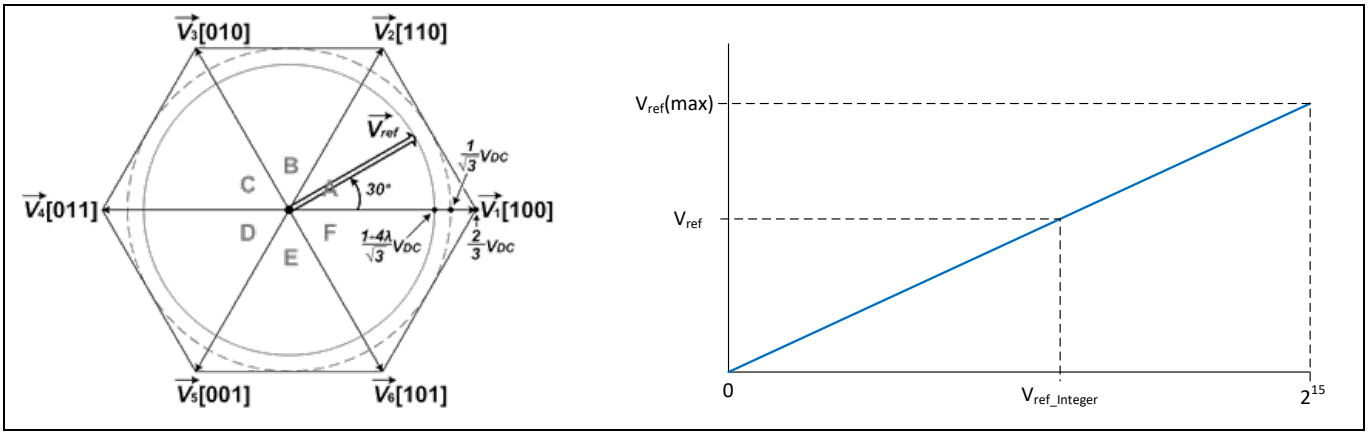


Figure 28　SVM 電圧スケーリング

$$V_{ref} = \frac{N_Vref_SVM}{2^{15}} \cdot V_{ref_Integer}$$

Equation 18

$N_{_Vref_SVM}$ は SVM 最大リファレンス電圧

$$N_Vref_SVM = \frac{1-4\lambda}{\sqrt{3}} V_{DC}$$

Equation 19

$\lambda = T_Z/T_S$ は擬似ゼロベクトル比で、スタンダード SVM では λ=0 です。

V_{DC} はインバーターの DC Link 電圧、USER_VDC_LINK_V

例:

USER_VDC_LINK_V は、24.0f、λ=0

⇨ $N_{_Vref_(SMV)}$ = 13.86 V

V_{ref} = 0.5 V を表すソフトウェア整数値 $V_{ref_integer}$ = 1182

相電流のスケーリング

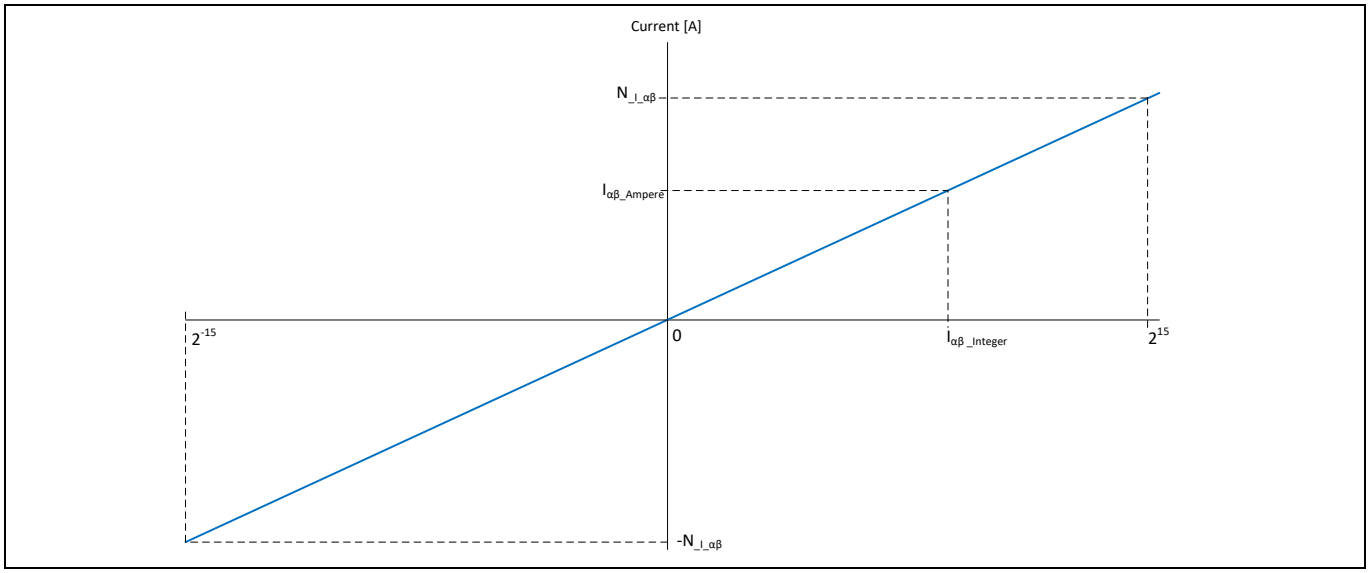


Figure 29　相電流スケーリング

電流のターゲット値は、電流検出回路で測定できる最大の電流です。

$$N_{_I_(\alpha,\beta)} = \frac{V_{AREF}/2}{R_{shunt} \times G_{OpAmp}}$$

Equation 20

内部 ADC ゲインが使用されている場合は、G_{OpAmp} は ADC ゲインファクターの設定に置き換えられます。

角度と速度のスケーリング

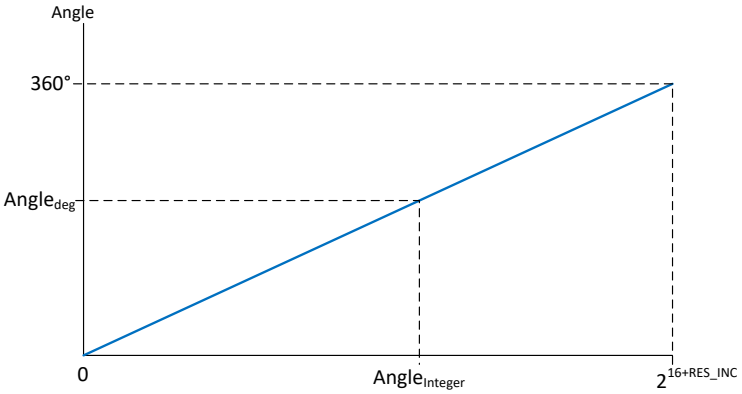


Figure 30　角度スケーリング

PMSM_FOC ソフトウェアでは、0°～ 360°の角度を表すために 16 ビット (または 16 + USER_RES_INC ビット、USER_RES_INC: 0 ～ 8) の整数を使用します。角度スケーリングの式は次のとおりです。

$$Angle_{deg} = \frac{360°}{2^{16+RES_INC}} \cdot Angle_{integer}$$

Equation 21

角度のスケーリングに続いて、速度のスケーリングは次のようになります。

$$\omega_{degree/second} = \frac{N_{max_speed}}{2^{N}} \cdot \omega_{integer}$$

Equation 22

ここで、

$\omega_{integer}$ は、CCU8 PWM サイクルごとに増減する角度です (つまり、整数速度)。

速度のターゲット値は次のとおりです。

$$N_{max_speed} = \frac{USER_SPEED_HIGH_LIMIT_RPM * USER_MOTOR_POLE_PAIR}{60} * 360°\ degree/seconds$$

Equation 23

$$max_speed_integer = \frac{N_{max_speed} * 2^{16+USER_RES_INC}}{60 * f_{CCU8_PWM}}$$

Equation 24

$$N = \log_2(max_speed_integer)$$

Equation 25

Note:　速度の制御がCCU8 PWM サイクルごとに行われない場合は、速度制御レートに基づいて上記のスケーリングを調整する必要があります。

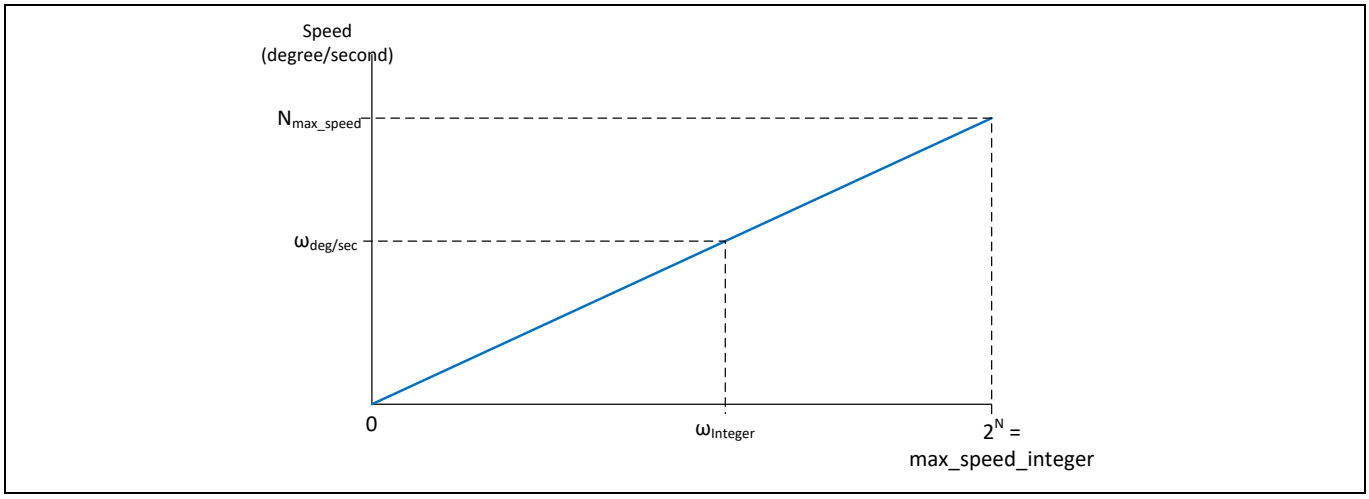


Figure 31　速度スケーリング

例:

- モーター最高速度、USER_SPEED_HIGH_LIMIT_RPM = 10,000 rpm
- モーター対極数、USER_MOTOR_POLE_PAIR = 4
- CCU8 PWM 周波数 = 25 kHz
- USER_RES_INC = 3

$$N_{max_speed} = \frac{(10000\ rpm * 4) * 360^{\circ}}{60} = 240{,}000\ degree/second$$

Equation 26

$$max_speed_integer = \frac{240{,}000 * 2^{16+3}}{360 * 25{,}000} = 13{,}981$$

Equation 27

$$N = \log_2(max_speed_integer) = \log_2(13{,}981) = 13.77$$

Equation 28

速度 2,000 rpm、つまり 48,000 (度/秒) を表すソフトウェア整数値は $\omega_{integer}$ = 2796

2.11　磁束とトルク電流の PI ゲインの決定

トルクおよび磁束電流制御の PI ゲインの初期値を計算するには、モーターの電気的パラメーターを知っておく必要があります。SPMSM モーターの場合、トルクインダクタンスと磁束インダクタンスは等しいと見なされます。

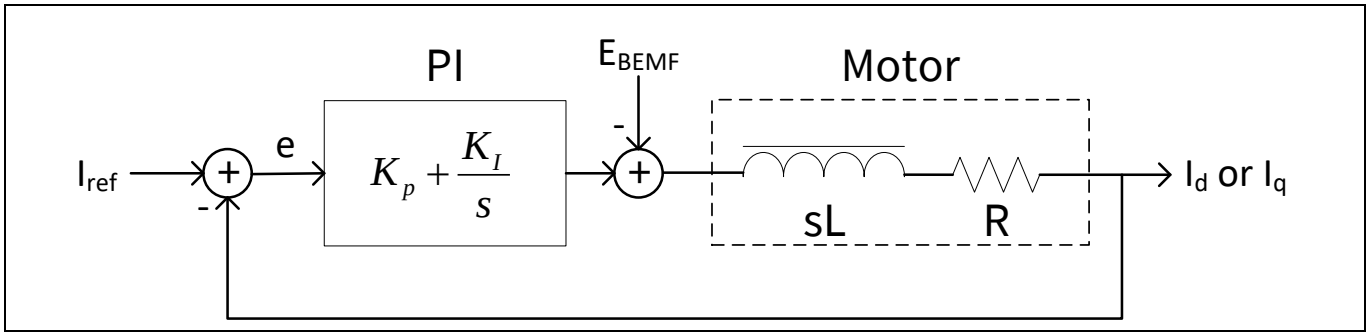


Figure 32　トルク/磁束電流制御ループ

PI ゲインの計算は、図に示すようにポールゼロキャンセル手法を使用して行われます。$K_P/K_I = L/R$ とすることにより、コントローラーはモーターのポールをキャンセルします。これにより、制御ループの伝達関数は、時定数 T_c を持つ 1 次 LPF になります T_c。さらに、比例ゲインの計算はモーターのインダクタンスに基づき、積分ゲインはモーター抵抗に基づきます。

モーター速度が一定の場合、モーターの逆起電力はほぼ一定です。したがって、周波数領域では無視できます。次の図に、ポールゼロキャンセル後に簡素化された図を示します。

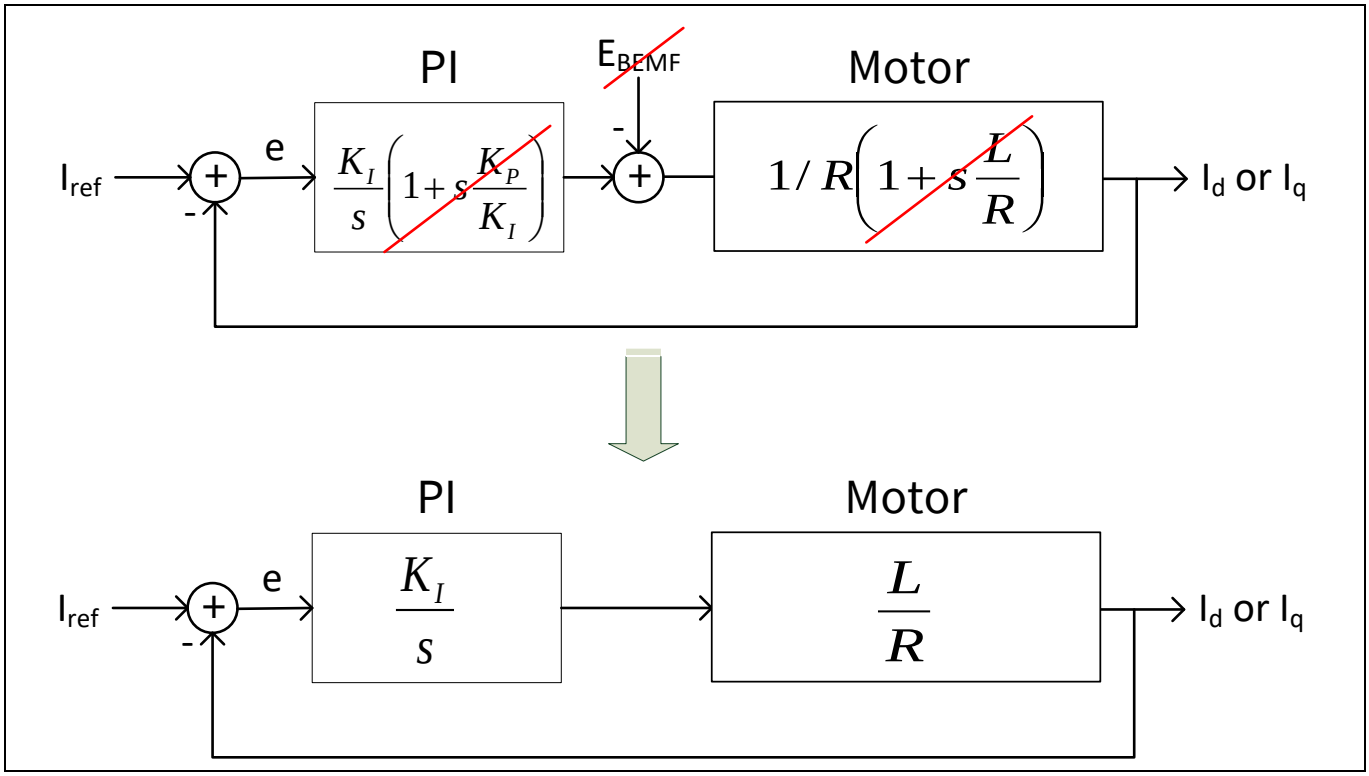


Figure 33　ポールゼロキャンセルによる簡素化された電流制御ループ

$K_P/L = K_I/R = \omega_c$として、PI コントローラーのゲインは次のようになります。

比例ゲイン　$K_P = \omega_c L$

積分ゲイン　$K_I = \omega_c R$

ここで、

- ω_cは 1 次 LPF のカットオフ周波数です。
- L はモーターのインダクタンスです。

- R はモーターの抵抗です。

デジタルコントローラーの実装では、積分部分はデジタルアキュムレーターです。したがって、K_I ゲインには、PWM 周波数である サンプリング時間 T_S のスケーリングファクターを含める必要があります。

変更後の式:

比例ゲイン　$K_P = \omega_c L \times A$

積分ゲイン　$K_I = \omega_c R \times T_S = RT_S K_P / L$

ここで、

- A は XMC™ハードウェア最適化スケーリングファクターです。

過去の経験に基づいて、ダイナミック応答と測定ノイズに対する感度の間の良好なトレードオフを得るために、カットオフ周波数をモーター最大速度の 3 倍に設定します。

3 電流検出と計算

このモジュールは、VADC ペリフェラルを使用してモーターの相電流を測定するために使用されます。

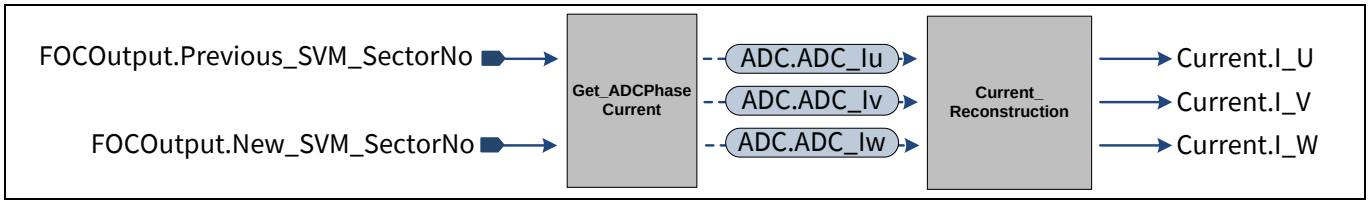


Figure 34 電流検出および計算機能

相電流を測定するための 2 つの手法

- シングルシャント電流検出 (XMC1300/XMC1400 のみ)
- 3 シャント電流検出

ユーザー設定ファイルにて電流検出手法を選択できます。

相電流の測定は、PWM SVM パターン生成に同期します。CCU80 モジュールの 4 番目のスライスであるスライス 3 (XMC4400 のスライス 2) は、ADC 変換をトリガーするために使用されます。CCU80 モジュールと VADC モジュールのさまざまな電流検出手法の初期設定は、それぞれのサブセクション、3.1 および 3.2 に記載されています。

次の図に、同期 ADC 変換を使用した 3 シャント電流検出手法でのタイミング図を示します。

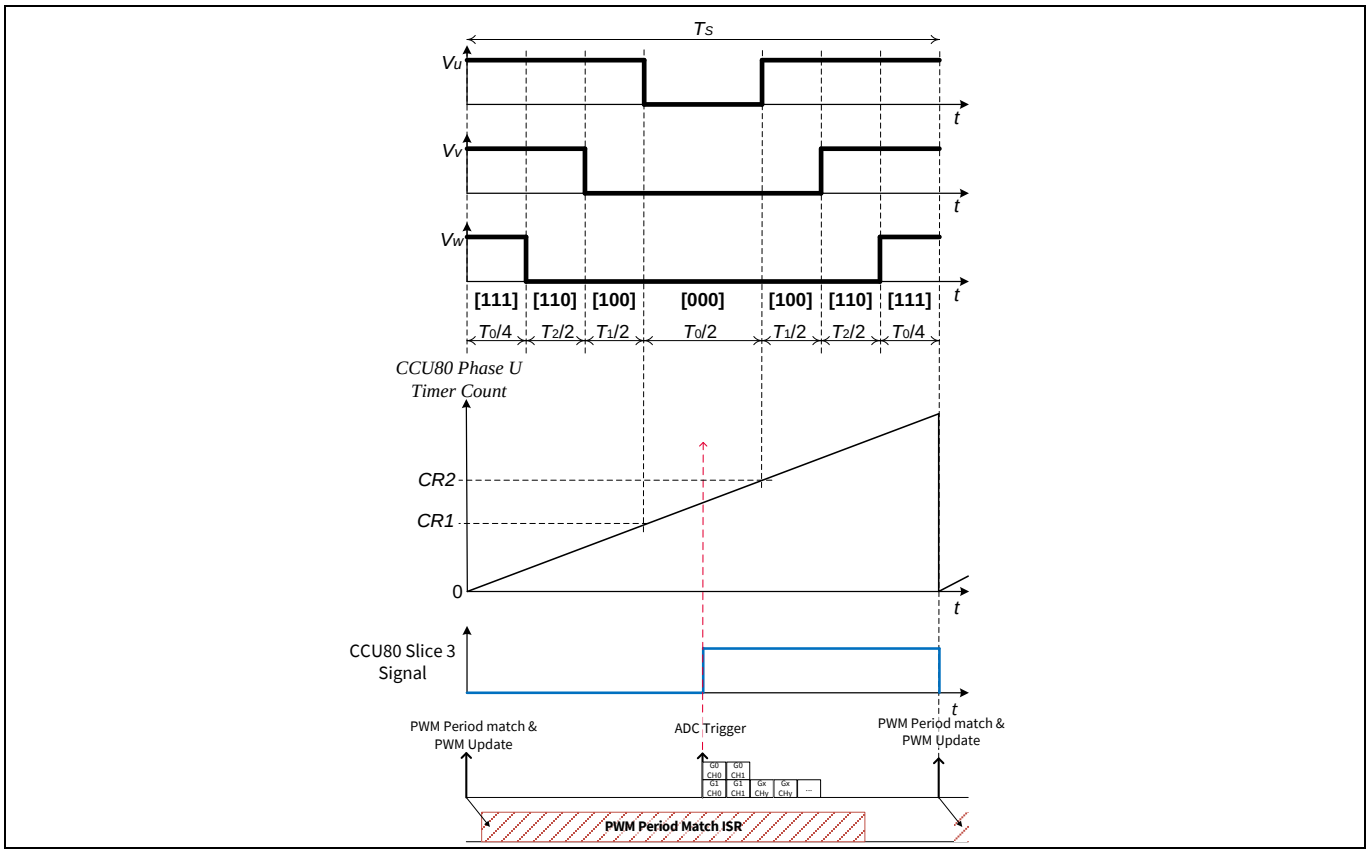


Figure 35 同期変換 ADC を使用した 3 シャント電流検出タイミング図

内蔵 ADC ゲイン機能

デフォルトのアプリケーションでは、オペアンプを使用してシャント抵抗での電圧降下分を増幅し、低電力損失かつ高い ADC 変換精度を実現します。この方法は、XMC™ PMSM FOC モーター制御ソフトウェアにてサポートされます。

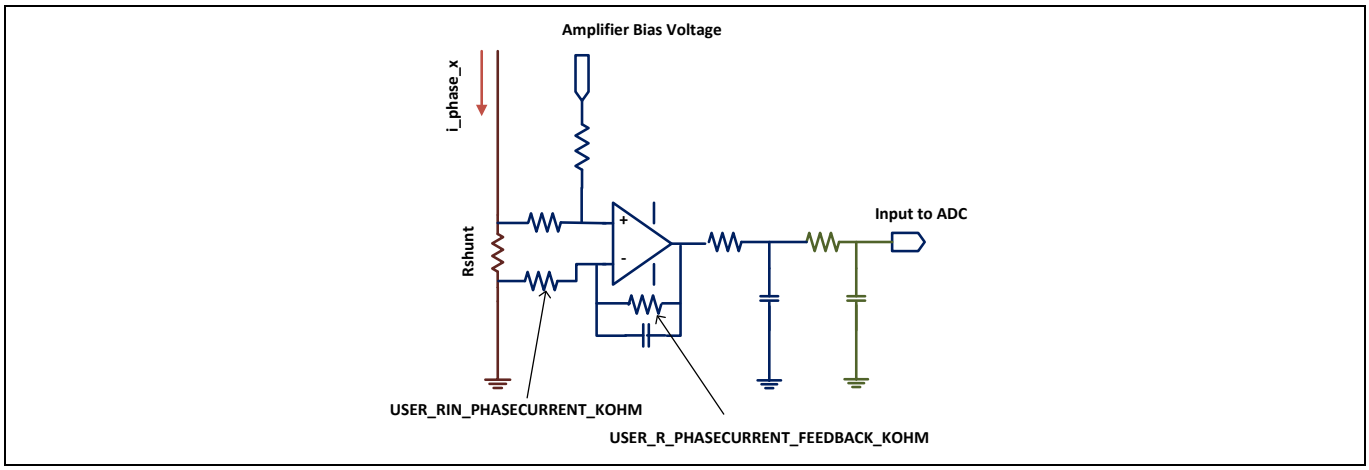
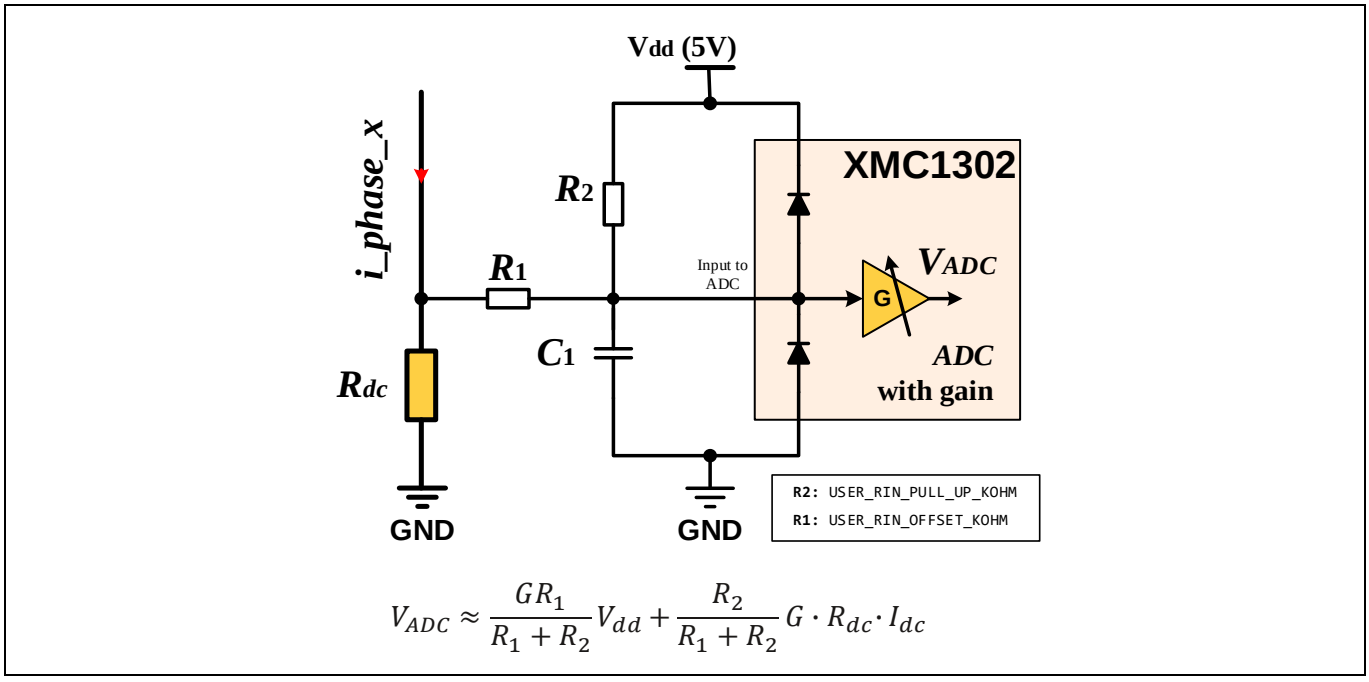


Figure 36　相電流外部アンプ

さらに、XMC™ MCU は ADC (VADC) のアナログゲイン段をサポートしています。この機能により、相電流検出に外部の高速オペアンプは必要ありません。これは、BOM のコスト削減につながります。



$$V_{ADC} \approx \frac{GR_1}{R_1 + R_2} V_{dd} + \frac{R_2}{R_1 + R_2} G \cdot R_{dc} \cdot I_{dc}$$

Figure 37　相電流内部アンプ (オンチップゲイン付き)

3.1 シングルシャント電流検出 (XMC1300/XMC1400 のみ)

シングルシャント電流測定手法は、電源電流を測定し、スイッチング状態に関する知見を適用して、モーターの 3 つの相の電流値を得ます。

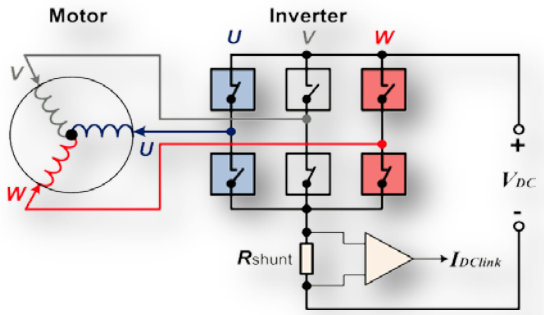


Figure 38　シングルシャント検出手法

擬似ゼロベクトル (PZV) SVM は、シングルシャント電流検出に十分な時間が確保されるようにするために使用されます。

Figure 39 に、電圧空間ベクトルの方向と、その状態で測定できる電流を示します。CCU80 スライス 3 は、Figure 39 に示すように、特定の時間に ADC 変換を自動的にトリガーするタイマーとして使用されます。ADC 変換は、CCU80 スライス 3 信号 (XMC4400 のスライス 2) の立ち上りエッジと立ち下りエッジの両方でトリガーされます。4 セグメント SVM を使用すると、ADC 変換トリガー ポイントも変更されます。Figure 40 を参照してください。

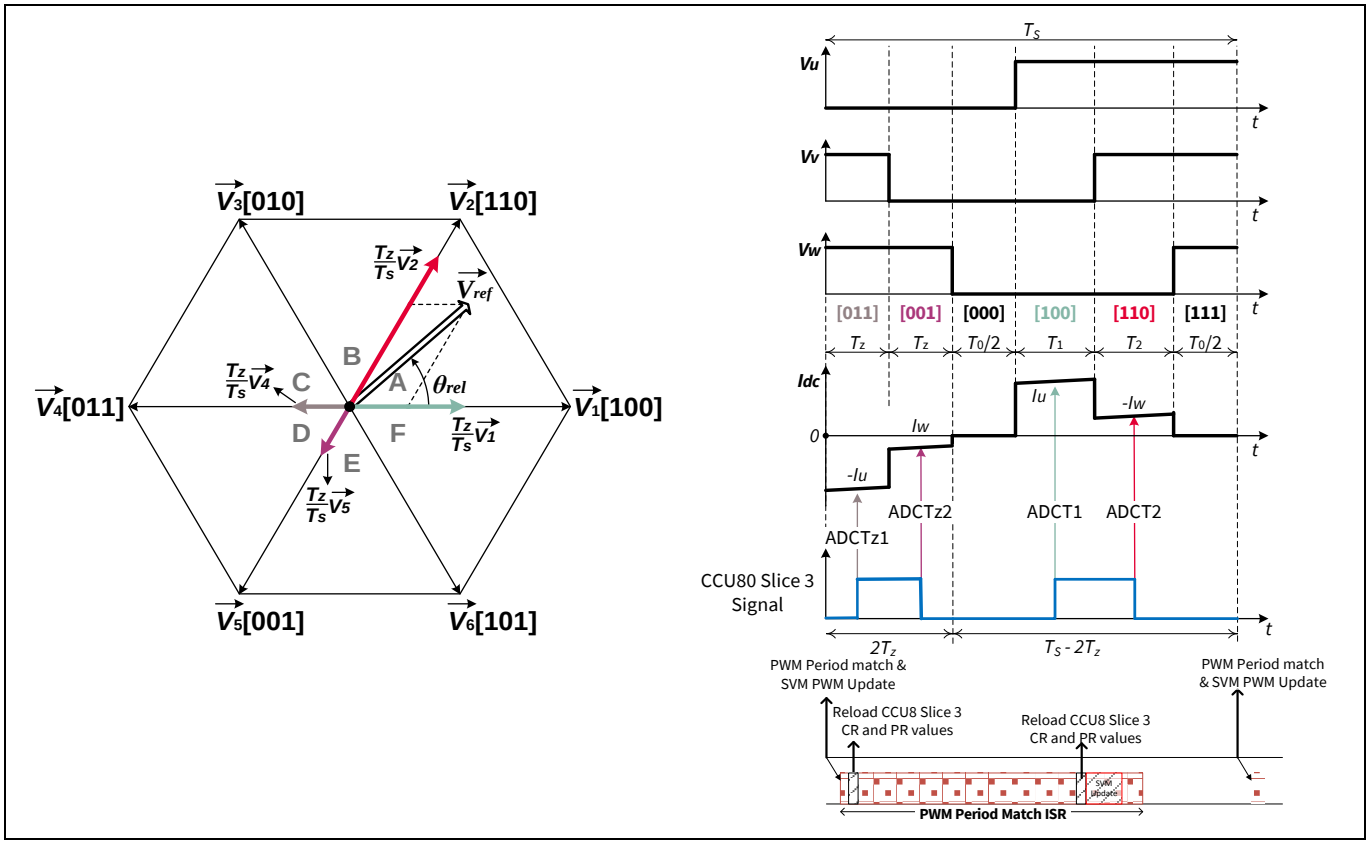


Figure 39　シングルシャント – セクターA での 3 相電流検出

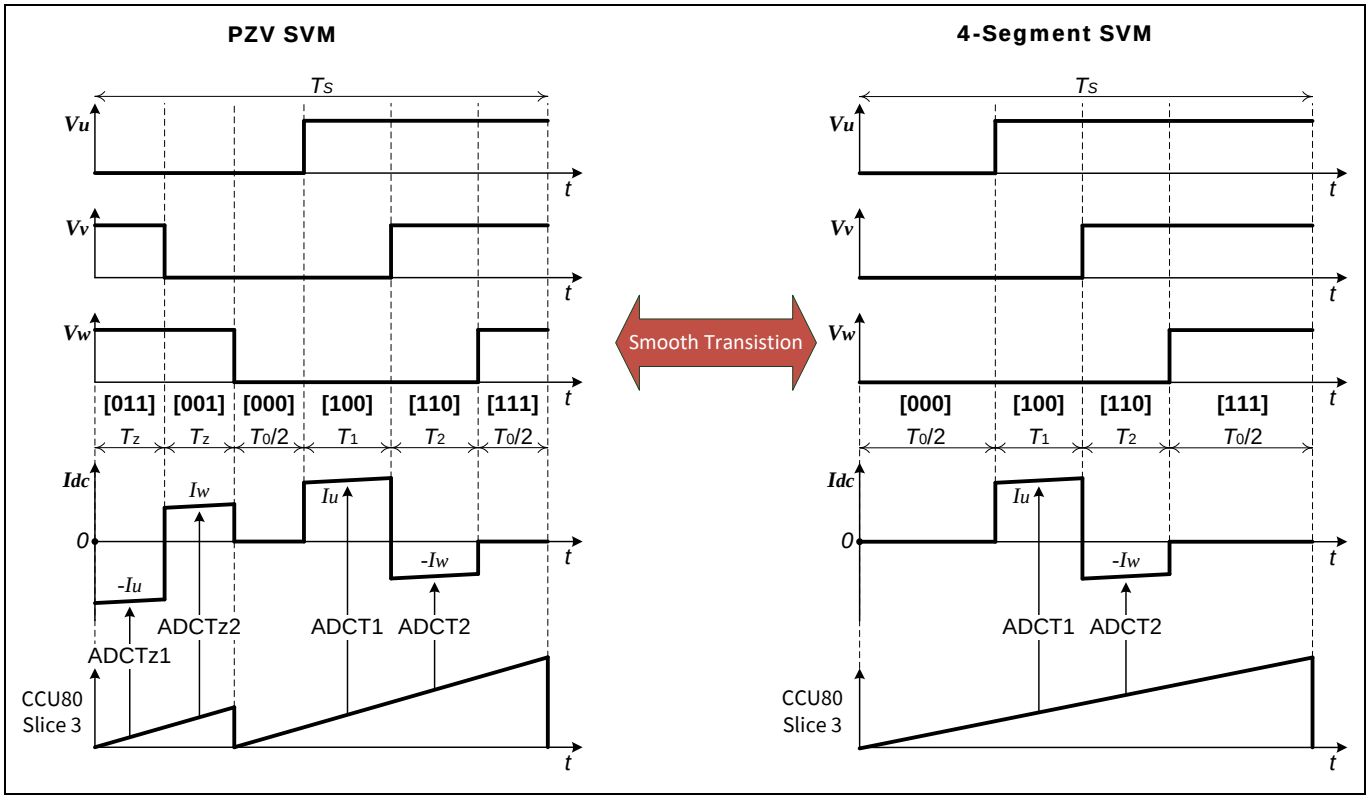


Figure 40　セクターA の PZV から 4 セグメント SVM へのスムーズな移行

VADC ソース割込みイベントが有効。そのサービスルーチンにて以下のタスクを実行します。

- ADC 変換の結果を読み出します。
- 相電流値を導出し、値を 2^{15} でスケーリングします

測定された 12 ビットの電流値は、1Q15 フォーマットでスケーリングされます。

PZV の場合:

- 電流 -> $I_U = (I_{ADCT1} - I_{ADCTz1}) * 2^3$
- 電流 -> $I_W = (I_{ADCT2} - I_{ADCTz2}) * 2^3$

4 セグメント SVM の場合:

- 電流 -> $I_U = \left(I_{ADCT1} - I_{ADC_Bias}\right) * 2^4$
- 電流 -> $I_W = \left(I_{ADCT2} - I_{ADC_Bias}\right) * 2^4$

以下の 2 つの表に、シングルシャント電流検出手法の VADC および CCU80 スライス 3 の初期設定を示します。

Table 9　シングルシャントの VADC 初期設定

パラメーター	設定
Request Source for Single-Shunt	Queue
Request Source for other Channels	Background scan
FIFO for Single-Shunt	2-Stage Buffer
Source Interrupt	有効
ADC Conversion Trigger Signal	CCU80.ST3A (through gating select input)
ADC Conversion Trigger Edge	Both Rising and Falling Edges

Table 10　CCU80 スライス 3 シングルシャントの初期設定

パラメーター	設定
Timer Counting Mode	Edged Aligned
Single Shot Mode	有効
Period Register	$2 * T_Z$
Compare Register Channel 1, CR1	$T_Z * 0.85$
Compare Register Channel 2, CR2	$T_Z + T_Z * 0.85$

3.2 3 シャント電流検出

3 シャント電流測定手法はシングルシャント検出と比較してより堅牢です。XMC1300/XMC1400 においてこの手法を使用する場合は、電流の計算のために 3 つの相電流のうち 2 つを選択できます。XMC4400/4800 では、3 相すべてを使用します。

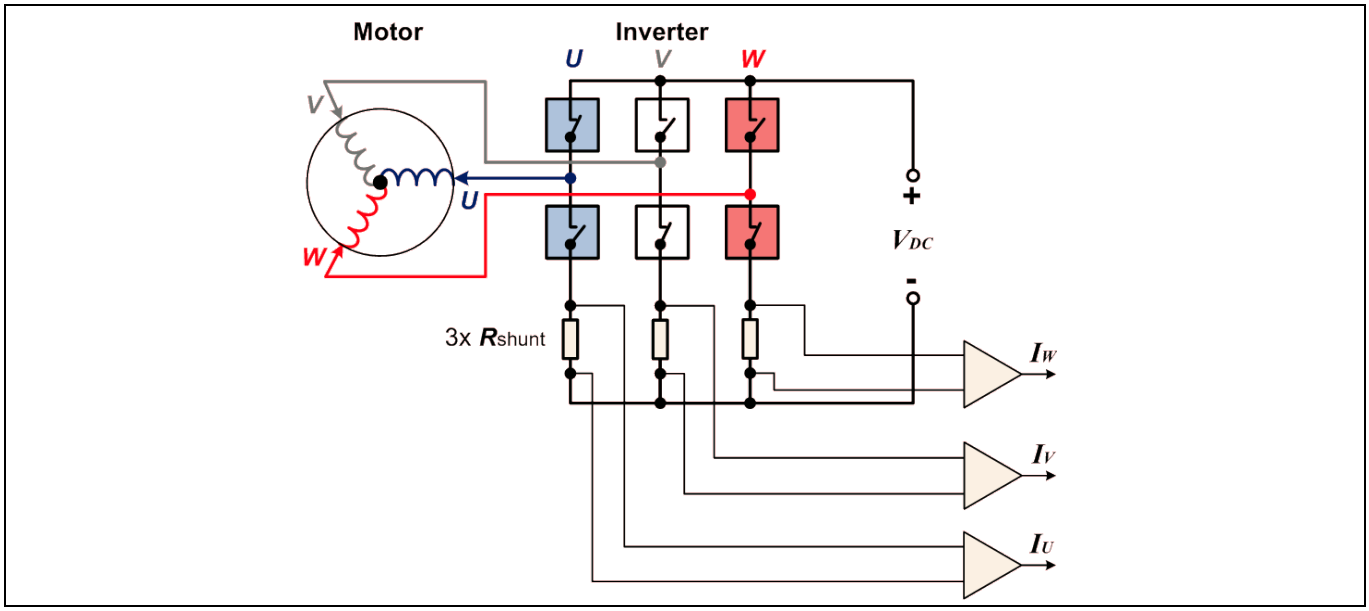


Figure 41 3 シャント検出手法

3 シャント電流検出の場合、ADC 変換トリガーは、すべてのローサイドスイッチがオンになる PWM サイクルの半分に設定されます (Figure 42 を参照)。ローサイドスイッチがオンでハイサイドスイッチがオフのとき、電流は常にシャント抵抗を流れます。

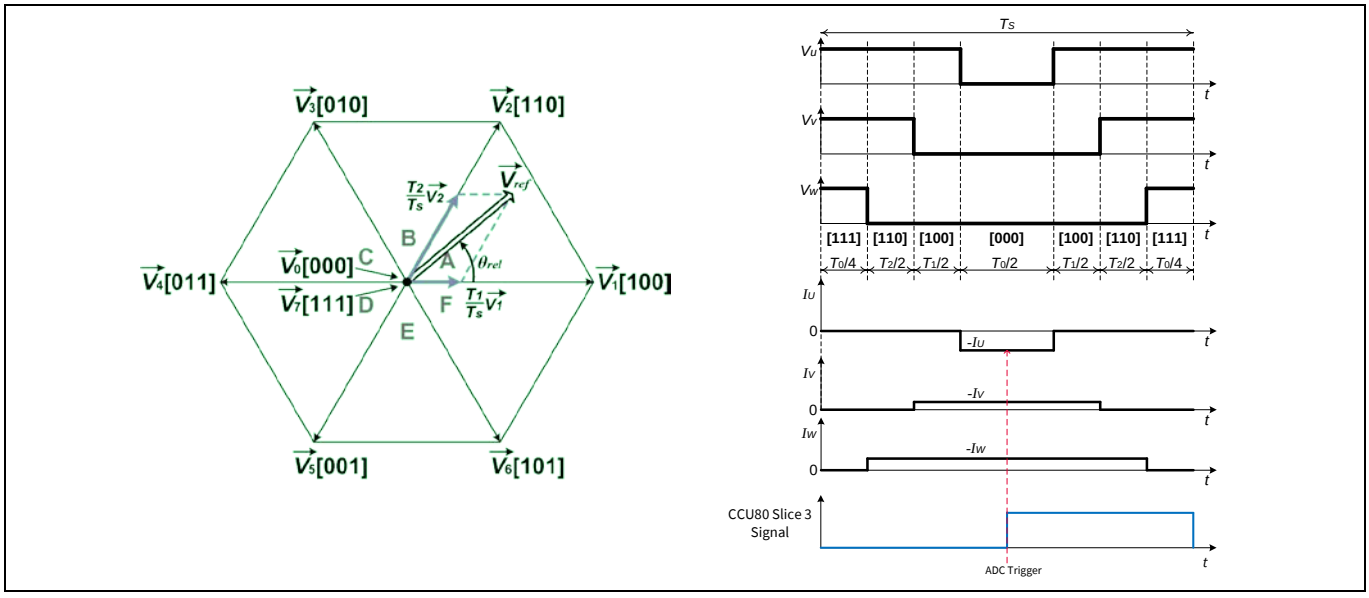


Figure 42 3 シャント、セクターA の 3 相電流検出

電流計算ファンクション内では、測定した 12 ビットの電流値は 1Q15 フォーマットでスケーリングされます。

- 電流 -> $I_U = \left(ADC_Bias_I_u - I_{ADC_Iu}\right) * 2^3$
- 電流 -> $I_V = \left(ADC_Bias_I_v - I_{ADC_Iv}\right) * 2^3$
- 電流 -> $I_W = \left(ADC_Bias_I_w - I_{ADC_Iw}\right) * 2^3$

VADC モジュールと CCU80 スライス 3 の初期設定については、次の表で詳しく説明します。

Table 11　3 シャント検出の VADC 初期設定

パラメーター	設定
Request Source for Three Shunt	Queue
Request Source for Other Channels	Background Scan
FIFO	無効
Source Interrupt	無効
ADC Conversion Trigger Signal	CCU80.ST3A (through gating select input)
ADC Conversion Trigger Edge	Rising Edge

Table 12　3 シャント検出の CCU80 スライス 3 初期設定

パラメーター	設定
Timer Counting Mode	Edged Aligned
Single Shot Mode	無効
Period Register	Same as Period Register value for 3-phase PWM
Compare Register Channel 1	Half of Period Register value
Compare Register Channel 2	Compare Register Channel 1 value + 1

Note:　XMC4800 は、ADC 変換トリガー信号に CCU81.ST3A を使用します。また、初期の 3 シャント設定には CCU81 スライス 3 を使用します。

3.2.1 非同期変換 (XMC1300/XMC1400 のみ)

非同期変換という用語は、グループの独立性に関連しています。このモードは簡単に実装でき、グループ 1 は使う必要がなく、ADC がリロードされず、理解しやすいという利点があります。

デフォルトの設定では、3 つの ADC 入力すべてが順番にサンプリングされ、3 つの電流すべてが順番に測定されます。このモードは ADC のリロードを必要とせず、3 つの ADC が使用可能な場合に特に適しています (XMC1000 には当てはまりません)。チャネルを手動でグループに分配できます。主な欠点は、測定にかかる時間がアドバンス実装の最大 2 倍になることです。長時間の電流測定ウィンドウが必要なため、この実装はデモンストレーション目的でのみ使用することを推奨します。XMC4400/4800 ファミリーでは実装されていません。

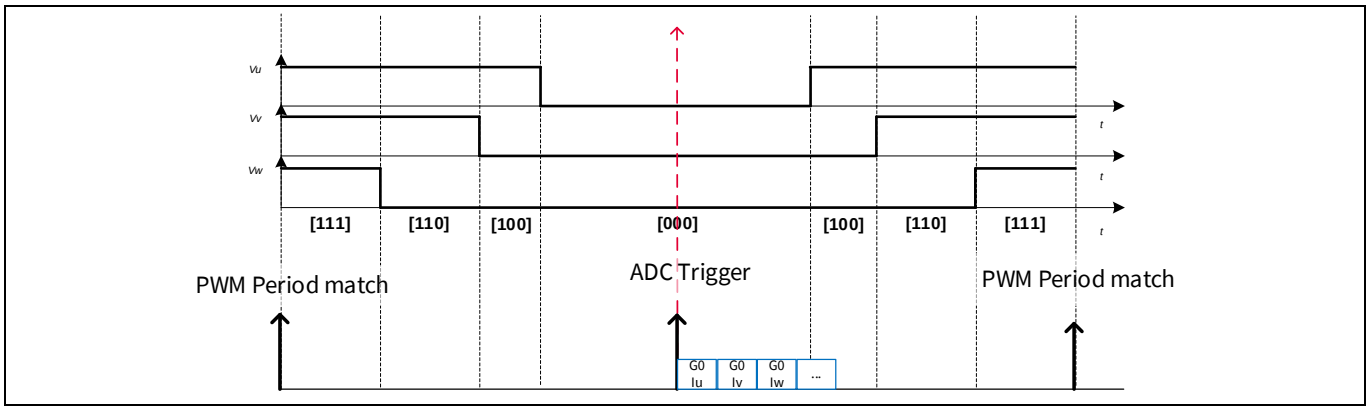


Figure 43 非同期変換シーケンス

3.2.2 同期変換

XMC1300 ファミリーと XMC1400 ファミリーでは、この実装は次の 3 つのハードウェア機能を使用します。

1 つ目の機能は、2 シャント電流の同期サンプリングとシーケンシャルな変換を可能にします。これにより、精度が向上し、最小測定ウィンドウが短縮されます。この機能では、VADC サンプルユニットとホールドユニットの両方を使用して、2 つの相の電流 (例えば、U 相と V 相) を測定します。この方法は、別の測定がバックグラウンドで実行されていても、影響を受けません。これにより、「synchronous conversion (同期変換)」という実装名になっています。

2 つ目のハードウェア機能は、大電流での測定を改善します。3 相レッグシャント電流測定では、電流はすべてのハイサイドスイッチがオフになっている中央で測定されます。この測定ウィンドウは、電流の振幅が大きくなると減少し、より高いトルクを生成します。測定ウィンドウが小さい相の決定は、セクターによって異なります (Figure 44 を参照)。

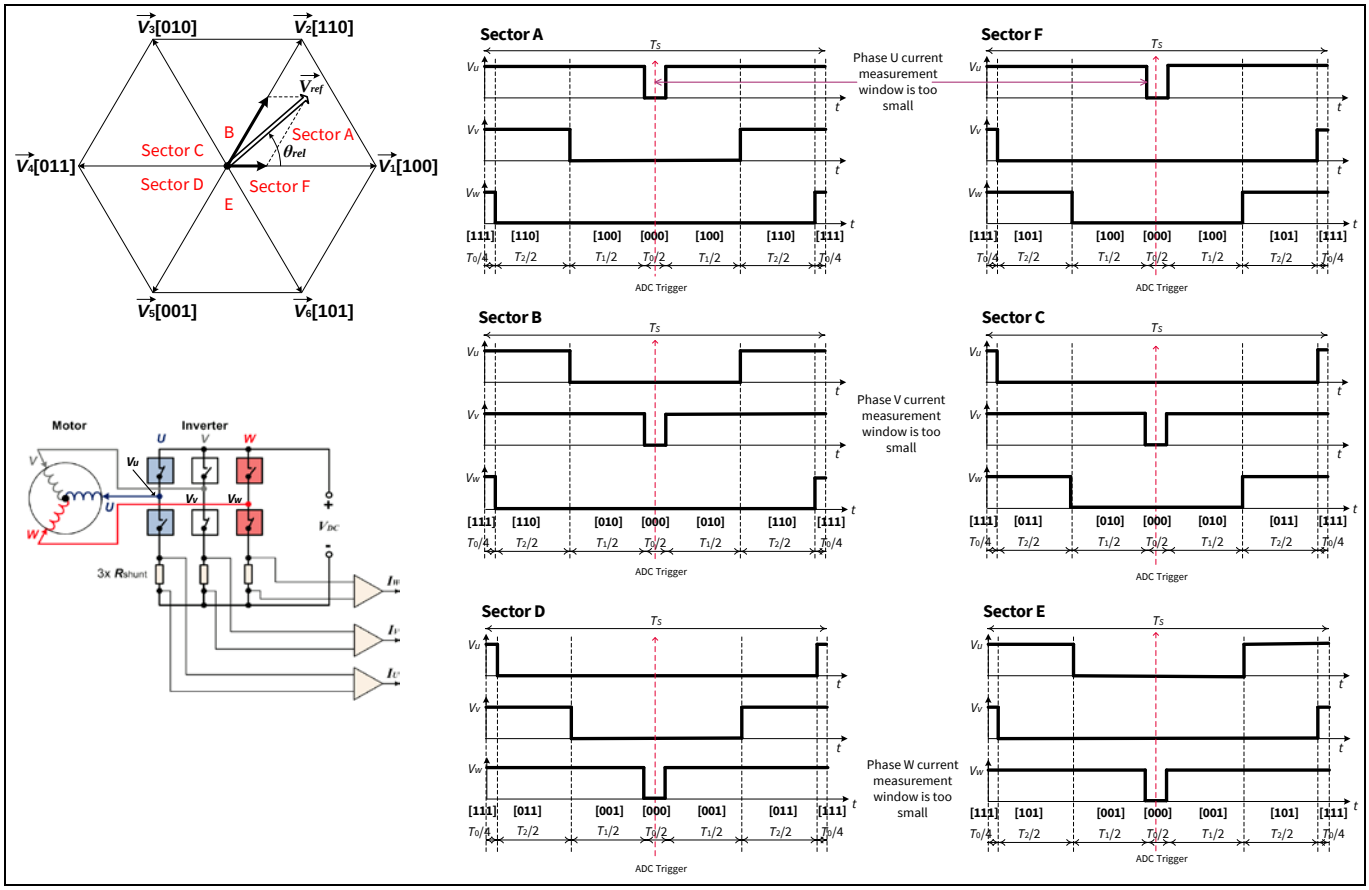


Figure 44　モーター高トルク時の SVM セクター

ソフトウェアは、セクターに応じて測定シーケンスを変更します。これにより、クリティカルではない 2 つの相の電流を測定できます。例えば、セクターA と F の場合、I_V と I_W の ADC チャネルを割り当てることができます。

次の表に、セクターに応じた同期測定がなされる相を示します。

Table 13　SVM セクターごとの相測定

SVM セクター	測定されるシャント電流の相
セクターA および F	W 相
	V 相
セクターB および C	U 相
	W 相
セクターD および E	V 相
	U 相

2 つ目のハードウェア機能は、ADC のエイリアス機能です。シーケンスの迅速な変更が可能なため、大きな電流振幅でも測定できます。その結果、測定ウィンドウが最小測定時間 T_{min} より小さい場合、ソフトウェアは 3 番目の相の測定を破棄します。このような場合、ソフトウェアは 3 レッグ シャント測定から 2 レッグ シャント測定に切り替えます。3 つの領域で、最小測定ウィンドウが 2 相で T_{min} を下回ります。Figure 45 に、3 レッグまたは 2 レッグ シャント測定で測定できる領域を示します。

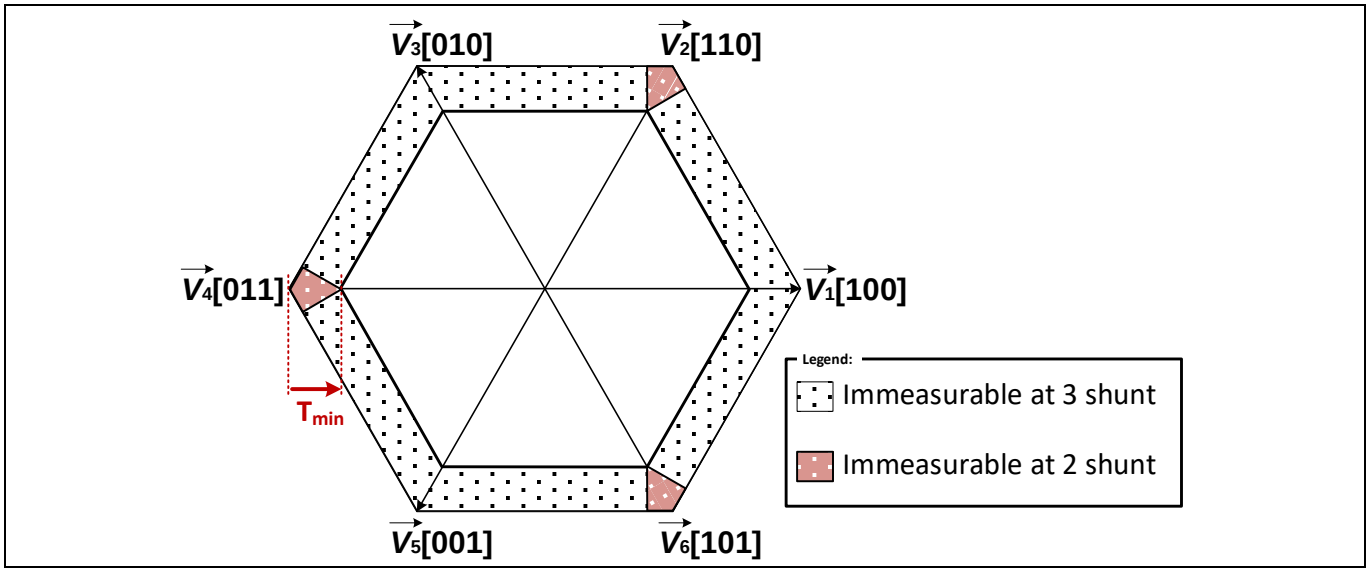


Figure 45　SVM 2/3 レッグシャント測定不能領域

同時サンプリングの相を切り替えるには、3 つの入力 (I_U, I_V, I_W) すべてが両方のグループ (G0 と G1) で使用できる必要があります。通常、これにより使われるピン数が 2 倍になります。XMC1300 ファミリーと XMC1400 ファミリーの 3 つ目のハードウェア機能は、グループチャネルをオーバーラップさせることで、この倍増を回避します。両方のグループから最大 4 つのピンにアクセスできます。

XMC4400/4800 ファミリーでは 4 つの VADC が存在するため、すべての相を同期して測定できます。各相は異なる ADC に接続され、各チャネルはチャネル 0 にエイリアスされています。

3.2.3 同期測定実装

ADC モジュールのエイリアス機能を使用すると、異なる ADC 入力チャネルを並行に変換するように割り当てることができます。XMC1300/XMC1400 では、すべての SVM セクターでクリティカルではない 2 相の電流を測定できます。セクターA と F には、I_V と I_W の ADC チャネルを割り当てることができます。一方、XMC4400/4800 ファミリーでは、3 相の電流すべてを無条件で測定できます。

次の表に、XMC1300/XMC1400 のセクターごとに同期測定する相を示します。

Table 14 XMC1300/XMC1400 の SVM セクターのエイリアス設定

SVM セクター	測定されるシャント電流の相	CH0 のエイリアスチャネル
セクターA および F	W 相 (ピン P2.9)	グループ 0、チャネル 2
	V 相 (ピン P2.10)	グループ 1、チャネル 2
セクターB および C	U 相 (ピン P2.11)	グループ 0、チャネル 4
	W 相 (ピン P2.9)	グループ 1、チャネル 4
セクターD および E	V 相 (ピン P2.10)	グループ 0、チャネル 3
	U 相 (ピン P2.11)	グループ 1、チャネル 3

すべてのセクターの実装は次の図に示します。マスター (G0) とスレーブ (G1) の CH0 は同期して測定されます。変換後、マスター (G0) とスレーブ (G1) の CH1 が測定されます。

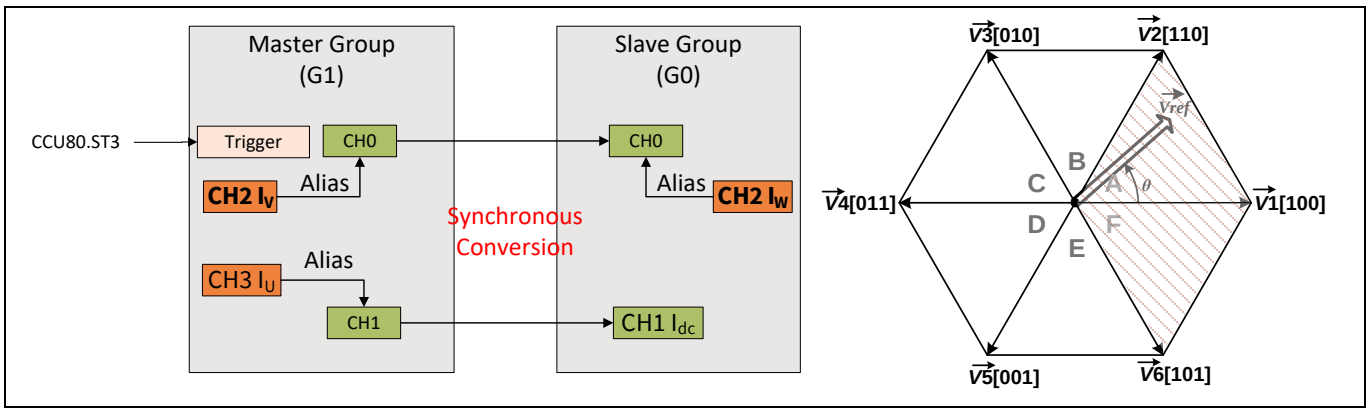


Figure 46 XMC1300 と XMC1400 のエイリアス機能を使用した同期変換 - セクターA と F

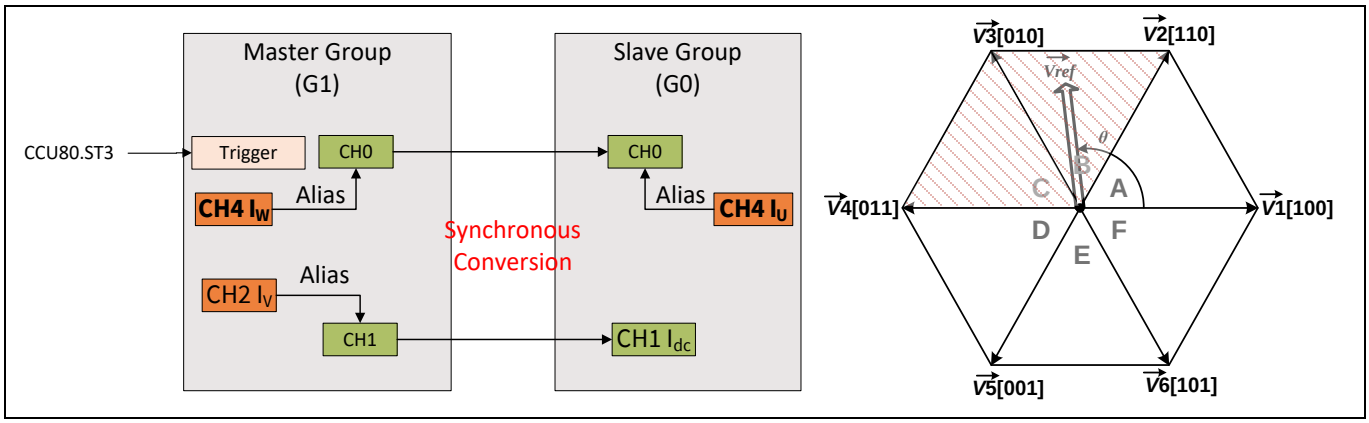


Figure 47 XMC1300 と XMC1400 のエイリアス機能を使用した同期変換 - セクターB と C

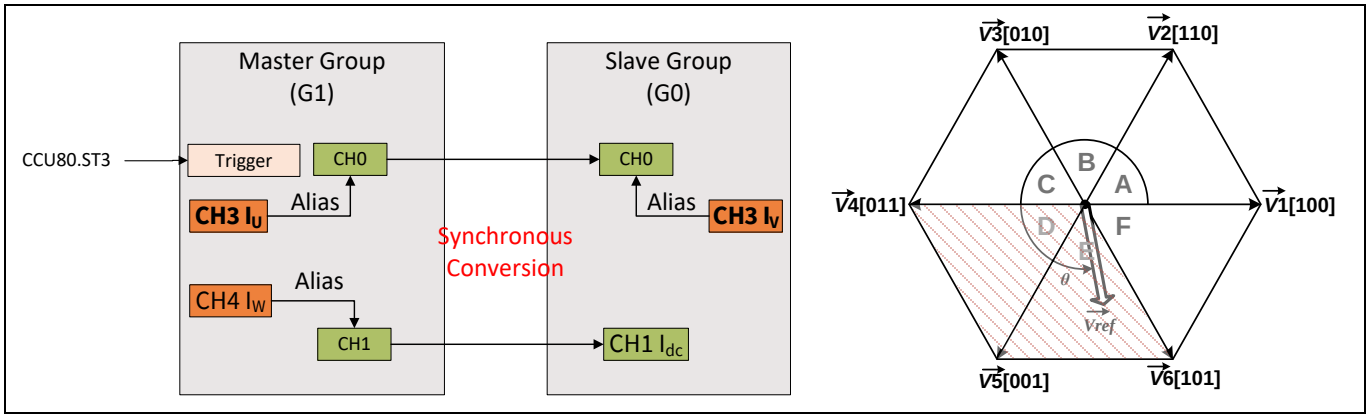


Figure 48 XMC1300 と XMC1400 のエイリアス機能を使用した同期変換 - セクターD と E

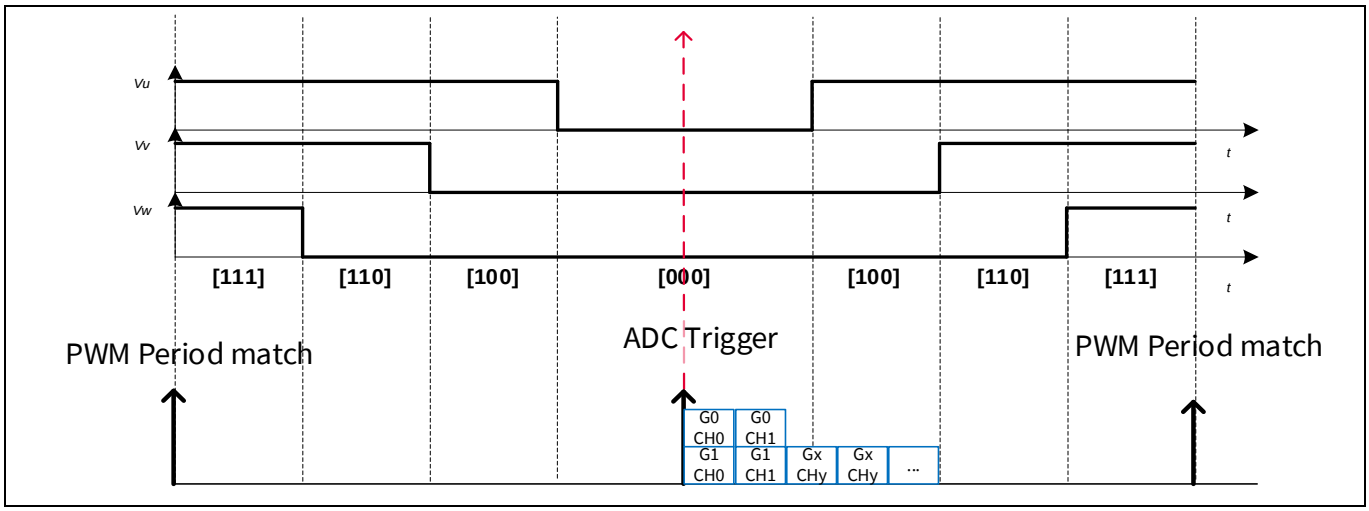


Figure 49 XMC1300/XMC1400 の同期変換シーケンス

XMC4400/4800 では、3 相すべてがトリガー ポイントで同期して検出されます。

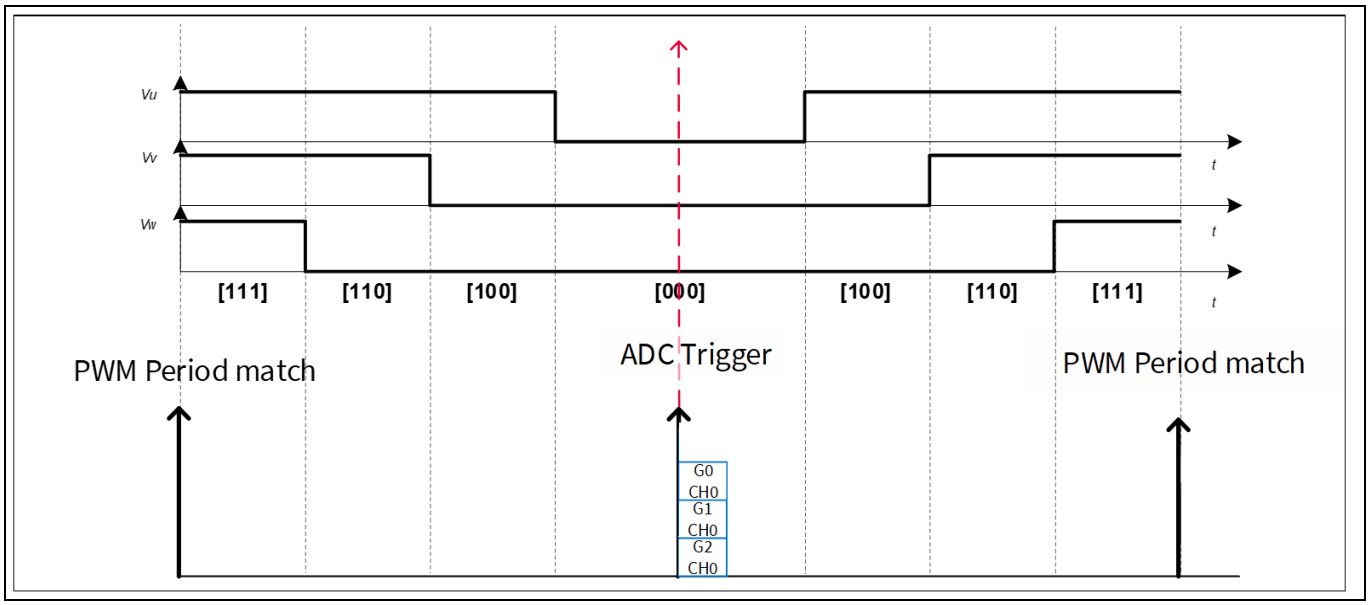


Figure 50 XMC4400/4800 ファミリーの同期変換シーケンス

4 センサーレス FOC 制御におけるモーター速度と位置のフィードバック

モーターのローター速度と位置のフィードバックは、PLL エスティメータソフトウェアライブラリでなされます。このライブラリには、インフィニオンの特許取得済み IP が含まれており、コンパイルされた libPLL_Estimator.a ファイルとして提供されます。以下は、ライブラリでは以下の API が提供されます。

Note: これらの API が確実に示されている正しい順序で呼び出されることを確認する必要があります。

1. `PLL_Imag(int32_t Vref_AngleQ31, int32_t I_Alpha_1Q31, int32_t I_Beta_1Q31)`
2. `PLL_Imag_GetResult(PLL_EstimatorType* const HandlePtr)`
3. `PLL_Vref(int32_t Delta_IV, uint32_t Vref32, int32_t PLL_UK, int32_t Phase_L, PLL_EstimatorType* const HandlePtr)`
4. `PLL_Vref_GetResult(PLL_EstimatorType* const HandlePtr)`
5. `PLL_GetPosSpd(PLL_EstimatorType* const HandlePtr)`

これらの API と必要なパラメーターについては、以下のように説明します。

Table 15 `PLL_Imag()` 関数

名称	`PLL_Imag(int32_t Vref_AngleQ31, int32_t I_Alpha_1Q31, int32_t I_Beta_1Q31)`	
説明	XMC1300/XMC1400 では、この関数は、センサーレスエスティメーターの初回の CORDIC 計算を開始します。XMC4400/4800 では、この関数は初回の FPU 演算を行い、`PLL_Estimator` 構造体に結果をロードします。	
入力パラメーター	`Vref_AngleQ31`	電圧空間ベクトルの角度
	`I_Alpha_1Q31`	電流空間ベクトルのアルファ座標
	`I_Beta_1Q31`	電流空間ベクトルのベータ座標
	`HandlePtr`	PLL_Estimator 構造体ポインター
戻り値 **(XMC4400/4800 のみ)**	`Current_I_Mag`	電流の大きさ
	`Delta_IV`	2 回目の CORDIC 計算の入力パラメーターとして提供されます

Table 16 `PLL_Imag_GetResult()` 関数

名称	`PLL_Imag_GetResult(PLL_EstimatorType* const HandlePtr)`	
説明	XMC1300/XMC1400 では、この関数はセンサーレスエスティメーターの初回の CORDIC 計算の結果を読み出します。XMC4400/4800 では、この関数の呼び出しはありません。	
入力パラメーター	`HandlePtr`	`PLL_Estimator` 構造体ポインター
戻り値 **(XMC1300/XMC1400 のみ)**	`Current_I_Mag`	電流の大きさ
	`Delta_IV`	2 回目の CORDIC 計算の入力パラメーターとして提供されます

Table 17 PLL_Vref() 関数

名称	PLL_Vref(int32_t Delta_IV, uint32_t Vref32, int32_t PLL_UK, int32_t Phase_L, PLL_EstimatorType* const HandlePtr)	
説明	XMC1300/XMC1400 では、この関数はセンサーレスエスティメーターの 2 回目の CORDIC 計算を開始します。XMC4400/4800 では、この関数は 2 回目の FPU 演算を行い、結果を PLL_Estimator ポインターにロードします。	
入力パラメーター	Delta_IV	センサーレスエスティメーターの初回 CORDIC 計算の結果
	Vref32	最後の PWM サイクルの SVM 電圧の大きさ
	PLL_Uk	PLL Estimator PI コントローラー出力
	Phase_L	モーターステータ巻線の相インダクタンス
	HandlePtr	PLL_Estimator 構造へのポインター
戻り値 (XMC1300/XMC1400 の場合)	Current_I_Mag	アップデートされた電流の大きさ
戻り値 (XMC4400/4800 の場合)	VrefxSinDelta	PLL_Estimator 構造に使用されます

Table 18 PLL_Vref_GetResult() 関数

名称のみ	PLL_Vref_GetResult(PLL_EstimatorType* const HandlePtr)	
説明	XMC1300/XMC1400 では、この関数はセンサーレスエスティメーターの 2 回目の CORDIC 計算の結果を読み出します。XMC4400/4800 では、この関数の呼び出しはありません。	
入力パラメーター	HandlePtr	PLL_Estimator 構造へのポインター
戻り値 (XMC1300/XMC1400)	VrefxSinDelta	PLL_Estimator 構造に使用されます

Table 19 PLL_GetPosSpd() 関数

名称	PLL_GetPosSpd(PLL_EstimatorType* const HandlePtr)	
説明	センサーレスエスティメーターからローターの位置とローター速度を計算して読み出します。	
入力パラメーター	HandlePtr	PLL_Estimator 構造へのポインター
戻り値	RotorAngleQ31	推定ローター位置
	RotorSpeeds_In	推定ローター速度

5 割込み

PMSM FOC ソフトウェアの割込みイベントと優先順位を次の表に示します。

Table 20 割込みの優先順位

割込みイベント	優先順位	コメント
CTrap	0 (最高優先順位)	フォールト検出
ADC キューソース	1	シングルシャント電流検出用のみ (XMC4400/4800 では実装されません)。3 シャント電流検出では無効です
PWM ピリオドマッチ (U 相)	2	ステートマシンの実行
セカンダリーループ	3	優先度が低く、低サイクル周波数の API。
過電圧/低電圧保護	1	イベントは、DC バスの電圧が範囲外にある場合にのみ発生します

5.1 PWM ピリオドマッチ割込み

PMSM_FOC ステートマシンは、U 相の PWM 周波数ピリオドマッチ ISR で実行されます。ISR は、ステートマシンで構成されます (6 章を参照)。

PWM ピリオドマッチ割込みの流れの例を Figure 51 に示します。この例では、FOC ダイレクト起動制御方式のフローを示します。選択した電流検出手法は、3 シャント同期変換です。

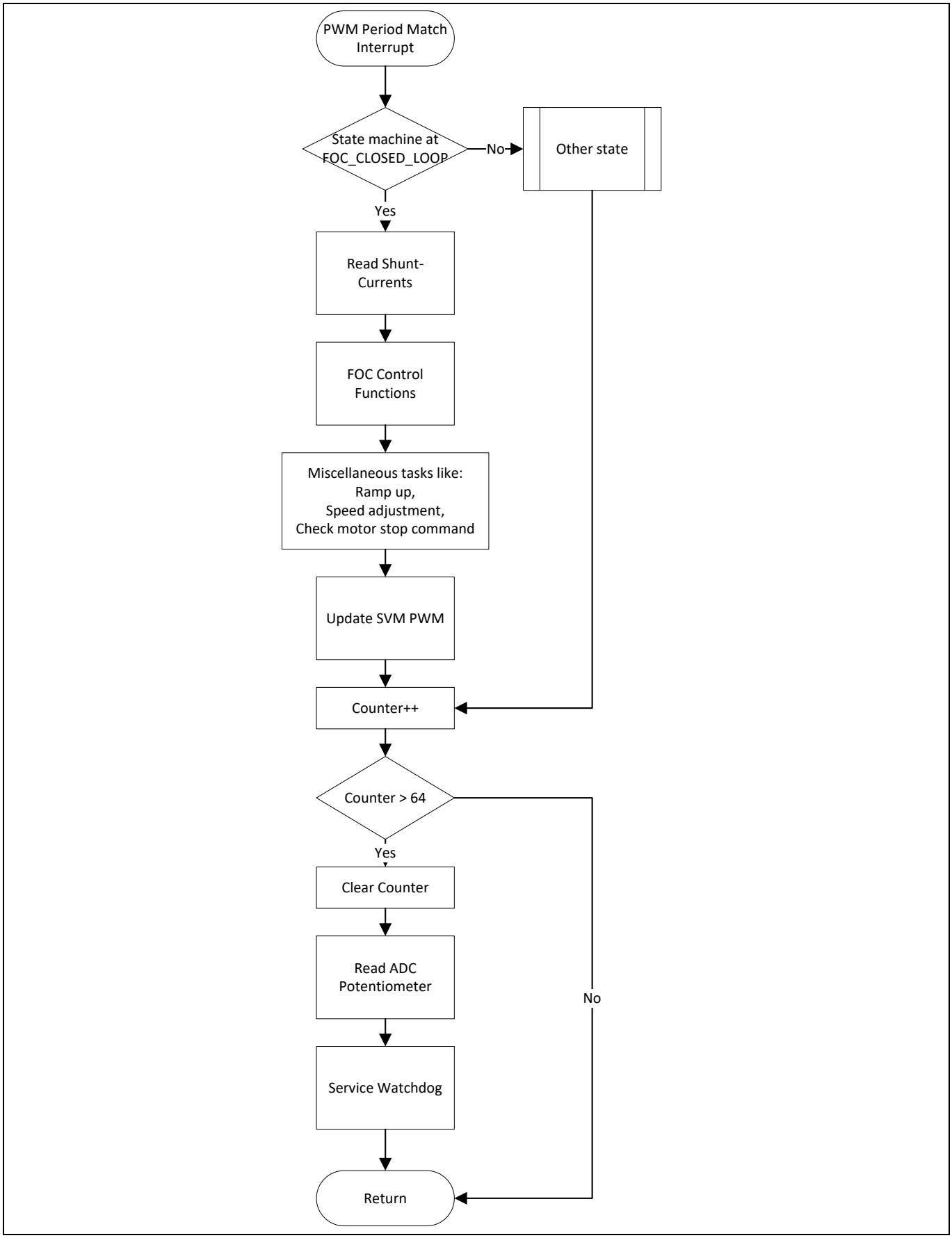


Figure 51　PWM ピリオドマッチ ISR のフローチャート

5.2 CTrap 割込み

選択した入力端子 (P0.12) にてトラップ状態が検出されると、CCU80 の出力がパッシブレベルに設定され、`Trap_Protection_INT()`が実行されます。ISR 内で、ゲードライバはディセーブル状態になり、モーターのステートは `TRAP_PROTECTION` に設定されます。この ISR は、最も高い優先順位であるレベル 0 で実行されます。

5.3 ADC ソース割込み (XMC1300/XMC1400 のみ)

この割込みは、シングルシャント電流検出の場合にのみ有効で、ADC 変換終了時にトリガーされます。ISR では、ADC の結果が読み出されます。

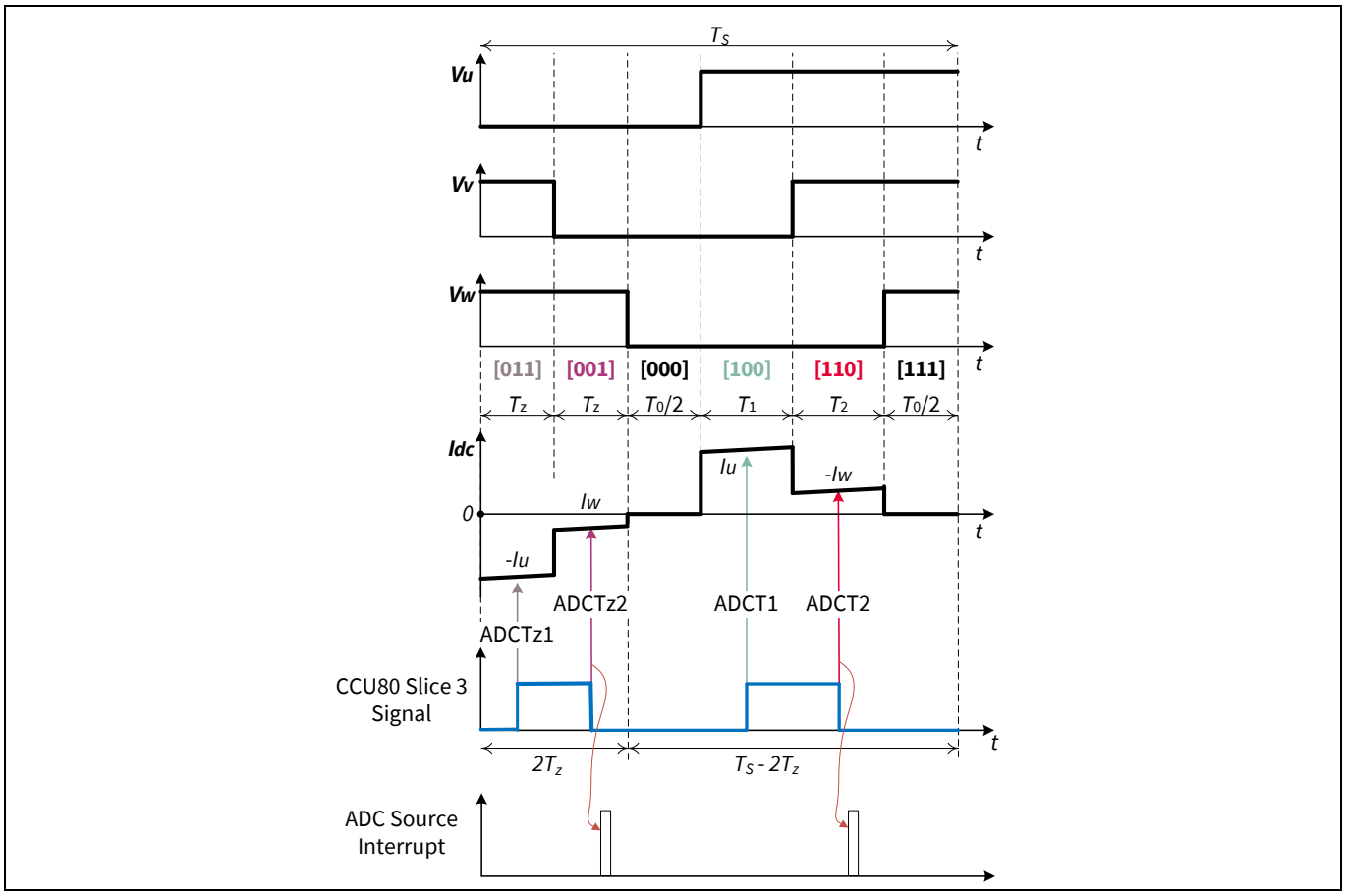


Figure 52 ADC ソース割込みタイミング図

5.4 セカンダリーループ割込み

このループは、通信やウォッチドッグサービスなどの低速なタスクに使用されます。トリガーは、独立したタイマーにて生成されます。これは、周波数が PWM ピリオドマッチの割込みと同期していることが必須ではないことを意味します。結果を予測しやすくするため、その周波数は USER_CCU8_PWM_FREQ_HZ の分周した値 (デフォルトは 20 kHz) としています。

5.5 過電圧/低電圧保護

この割込みは、DC バス電圧が範囲外にある場合にのみ ADC からトリガーされます。

6 モーターステートマシン

PMSM FOC ソフトウェアには、内部ステートマシンがあります。

MOTOR_IDLE

電源投入後またはソフトウェアリセット後の最初のステートです。このステートでは、インバーターはディセーブル状態になり、モーターの相電流測定のために接続されている ADC ピンのバイアス電圧を読み出します。ステートマシンおよびタイマーが始動します。モーター始動コマンドを受信すると、このステートから抜け出します。

EN_INVERTER_BOOTSTRAP

このステートでは、インバーターがイネーブル状態になり、ブートストラップコンデンサが定義された時間充電されます。モーターの相電流測定のために接続されている ADC ピンのバイアス電圧を読み出します。

ブートストラップ時間の経過後、このステートから抜け出しします。

PRE-POSITIONING

このステートは、ダイレクト FOC 起動制御方式専用です。このステートでは、最大始動トルクを得るためにローターは既知の位置に位置合わせされます。SVM ファンクションへの振幅入力は、特定の時間 (USER_ROTOR_PREPOSITION_TIME_MS) にわたり、定義された値 (USER_STARTUP_VF_OFFSET_V) まで徐々に増加します。これらのマクロは、*pmsm_foc_motor_XXXX.h* ファイルにて定義されます (7.2.3.2 を参照)。

VF_OPENLOOP_RAMPUP

このステートでは、モーターは V/f オープンループ制御モードで始動します。

モーター速度がマクロ (USER_STARTUP_SPEED_THRESHOLD_RPM) で定義された起動しきい値速度に達すると、このステートから抜け出します。

MET_FOC

このステートは、オープンループからクローズドループへのスムーズな遷移を可能とし、エネルギー効率を最大化します。

FOC_CLOSED_LOOP

このステートでは、モーターは FOC モードで動作しています。FOC ファンクションが実行されます。

FOC_CLOSED_LOOP_BREAK

このファンクションは、ターゲット値の入力は不可であることを除き、FOC_CLOSED_LOOP と同じです。ターゲット値は S 字カーブで低下していきます。

FOC_EXIT_SPEED を下回ると、ステートは MOTOR_STOP になります。

MOTOR_HOLD

モーター速度が最大速度の 10%を下回ると、このステートになります。モーターは PWM が 50%オン、50%オフを維持するように設定されます。このステートでモーターブレークコマンドが実行されると、直に MOTOR_STOP になります。

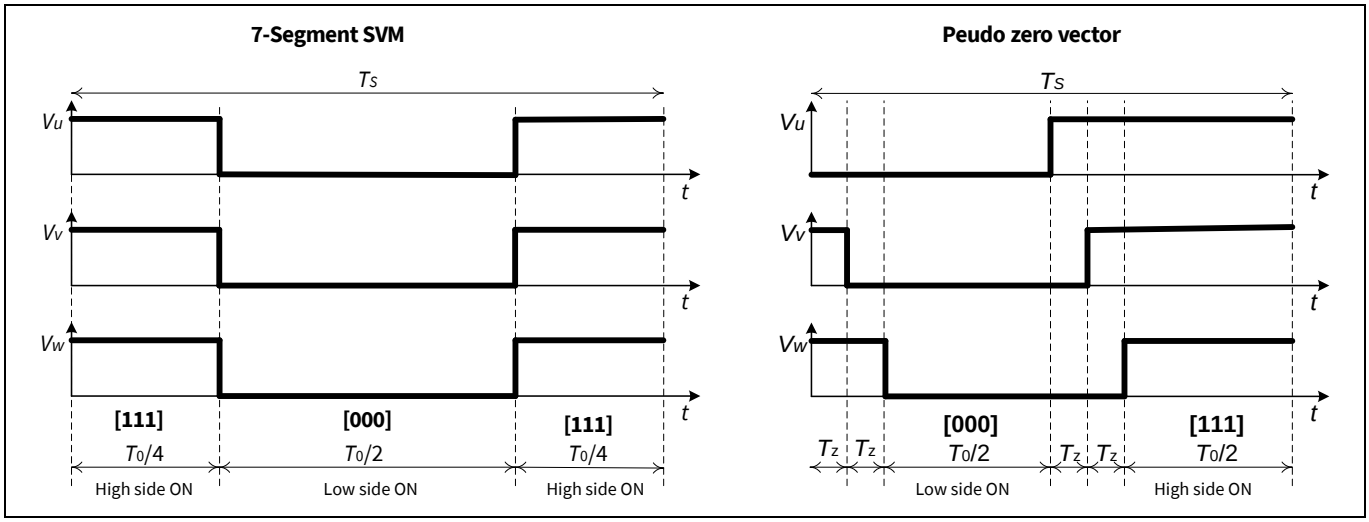


Figure 53　モーターブレーキ条件での SVM 出力

MOTOR_STOP

このステートは、停止コマンド(およびその他のステート関連条件)によってすべてのステートから入ることができます。モーター出力はトライステートになり、インバーターはディセーブル状態になります。これにより、モーターは制御されていないフリーホイール状態になります。処理後は、アイドルステートに戻ります。

TRAP_PROTECTION

CTrap がトリガーされると、このステートになります。

このステートを抜けるためには、ターゲット値を `MOTOR_HOLD_THRESHOLD` より下に設定し、ブレークまたは停止コマンドをセットします。

DCLINK_UNDER_VOLTAGE

DC Link 電圧がユーザーによって設定されたリミット値を下回ると、このステートになります。ゲートドライバはディセーブル状態になり、モーターはフリーランニング状態になります。

モーター停止コマンドまたはモーターブレーキコマンドによってのみ、このステートから抜け出します。

DCLINK_OVER_VOLTAGE

DC Link 電圧がユーザーによって設定されたリミット値を上回ると、このステートになります。ゲートドライバはディセーブル状態になり、モーターはフリーランニング状態になります。

モーター停止コマンドまたはモーターブレーキコマンドによってのみ、このステートから抜け出します。

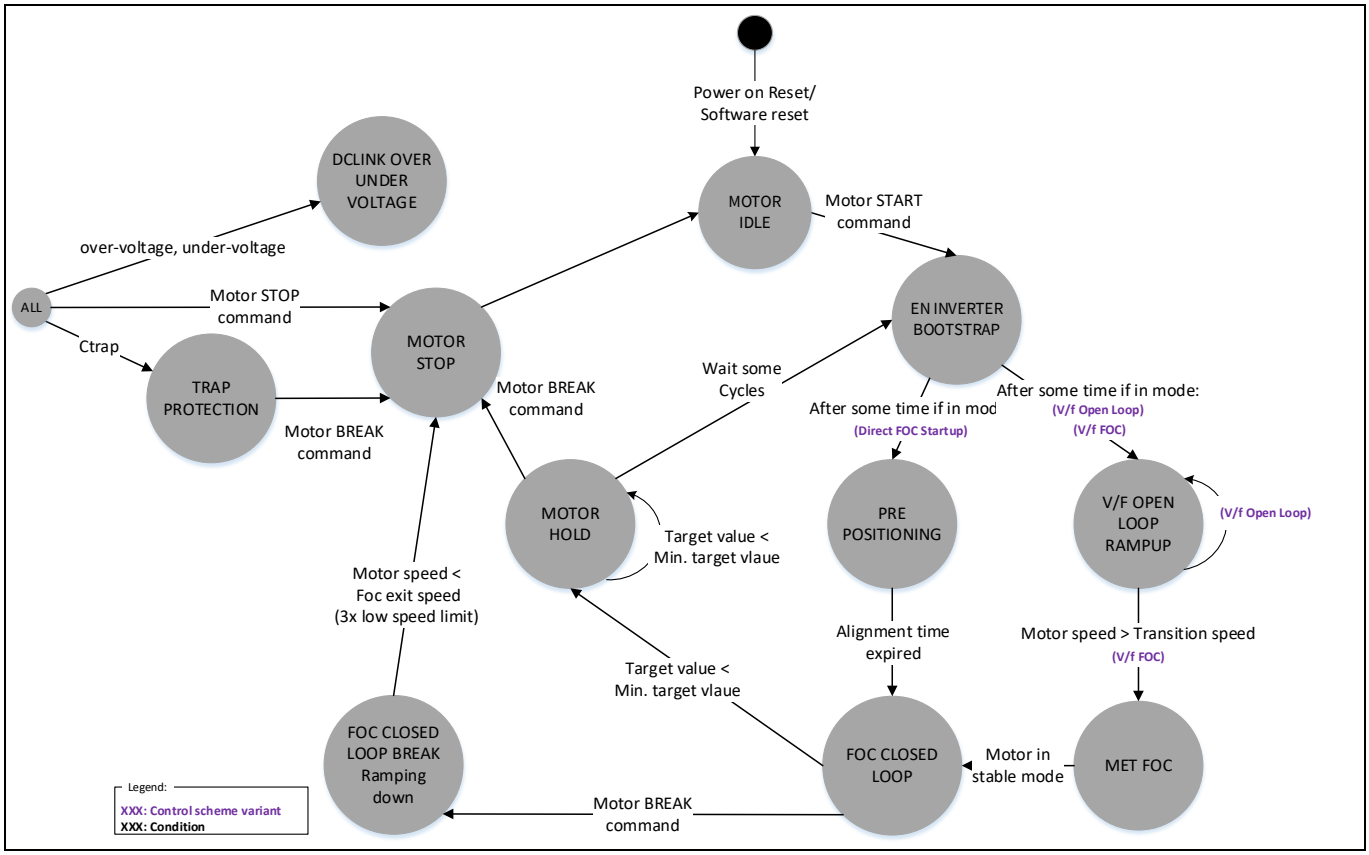


Figure 54 PMSM FOC ステートマシン

制御方式が異なれば、ステートマシンのフローも異なります。

次の制御方式が使用できます。

- ダイレクト FOC 起動
  - SPEED_CONTROLLED_DIRECT_FOC
  - TORQUE_CONTROLLED_DIRECT_FOC
  - VQ_CONTROLLED_DIRECT_FOC
- V/f FOC
  - SPEED_CONTROLLED_VF_MET_FOC
- V/f オープンループ
  - SPEED_CONTROLLED_VF_ONLY

7 コンフィギュレーション

PMSM FOC ソフトウェアのパラメーターのデフォルトの設定は、XMC1000 モーター制御アプリケーションキットに基づいています。設定は、3 つのレベルに分かれています。

- ユーザー設定
  - FOC 制御方式などの一般的な方式への迅速な適応や、購入可能な複数の事前定義済ハードウェアボードの選択を可能にします。
- ハードウェア設定
  - コントローラーカード, インバーターカード, およびモーターへの適合を可能にします。この設定により、ハードウェア設定をアプリケーションとボードに合わせて調整できます。ピン配置や最大速度などの設定については、以下に説明があります。
- FOC 設定
  - 基本的な値と定数計算の変更が可能です。1 つの例は、DC Link 電圧の 120%の過電圧の定義です。これらの設定のほとんどは必須であり、計算されるため、ユーザーが変更することはできません。そのため、これらの設定は本書では説明しません。

7.1 ユーザー設定

ユーザー設定により、FOC 制御方式などの一般的な方式に迅速に適応したり、購入可能な事前定義済ハードウェアボードの 1 つを選択したりできます。すべての設定オプションは、*PMSM_FOC\Configuration\file pmsm_foc_user_config.h* ファイルにあります。

デフォルトの設定は、インフィニオン XMC1302 モーター制御アプリケーションキット (KIT_XMC1X_AK_MOTOR_001) で使用されている「Maxon モーター 267121」用です。

7.1.1 概要

PMSM FOC ハードウェアキット

ソフトウェアにはさまざまな設定オプションが用意されます。これらの設定は、主に互いに独立しています。ハードウェアは、次のもので構成されます。

- コントローラーカード (MCUCARD_TYPE)
- インバーターカード (INVERTERCARD_TYPE)
- モーター (MOTOR_TYPE)

本書では、これら 3 つを組み合わせたものを*ハードウェアキット (PMSM_FOC_HARDWARE_KIT)* と呼びます。

多くの購入可能なボードは事前に定義されています。また、ハードウェアボードの部品交換も可能です。これらのオプションをサポートするには、カスタムハードウェアキットを選択し、MCUCARD_TYPE, INVERTERCARD_TYPE, MOTOR_TYPE, および関連するパスを適用します。

`#define PMSM_FOC_HARDWARE_KIT KIT_XMC1X_AK_MOTOR_001`

- 事前に定義されたハードウェアキットを選択します。

オプション:

- KIT_XMC1X_AK_MOTOR_001 – インフィニオン XMC1000 モーター制御アプリケーションキット
- KIT_XMC750WATT_MC_AK_V1 – XMC™ 750 Watt モーター制御アプリケーションキット
- KIT_XMC14_BOOT_001 – KIT_XMC1X_AK_MOTOR_001 用の XMC1404 CPU カード

- KIT_XMC750WATT_MC_AK_V1 – XMC1404 バージョンあり
- KIT_MOTOR_DC_250W_24V
- IFX_MADK_EVAL_M1_05F310 Kit
- IFX_MADK_EVAL_M1_05_65D_V1
- IFX_MADK_EVAL_M1_CM610N3
- CUSTOM_KIT - ユーザー定義のモーター制御システム
- KIT_XMC4400_EXAGON
- KIT_XMC4400_ISOLATED - KIT_MOTOR_DC_250W_24V 用 XMC4400 CPU カード
- KIT_XMC4800PSOC_IOT – KIT_MOTOR_DC_250W_24V 用 XMC4800 CPU カード

電流検出

`#define CURRENT_SENSING` `USER_THREE_SHUNT_SYNC_CONV`

- 使用する電流検出手法を定義します。

オプション:

- `USER_SINGLE_SHUNT_CONV` – 擬似ゼロベクトル PWM 生成を使用したシングルシャント電流検出手法、3.1 を参照 (XMC4400/4800 には実装されていません)
- `USER_THREE_SHUNT_ASSYNC_CONV` – ADC 標準変換を使用した 3 シャント電流検出手法、3.2 を参照 (XMC4400/4800 には実装されていません)
- `USER_THREE_SHUNT_SYNC_CONV` – ADC 同期変換を使用した 3 シャント電流検出技術、3.2.1 を参照

FOC 制御

`#define MY_FOC_CONTROL_SCHEME` `SPEED_CONTROLLED_DIRECT_FOC`

- FOC 制御方式を定義します。

オプション:

- `SPEED_CONTROLLED_DIRECT_FOC` – 速度制御を使用したダイレクト FOC 起動、2.3.2 を参照
- `SPEED_CONTROLLED_VF_ONLY` –オープンループ速度制御、2.3.1 を参照
- `SPEED_CONTROLLED_VF_MET_FOC` –オープンループの起動から MET へのクローズドループ FOC 速度制御、2.3.2 を参照
- `TORQUE_CONTROLLED_DIRECT_FOC` – トルク制御を使用したダイレクト FOC スタートアップ、2.3.3 を参照
- `VQ_CONTROLLED_DIRECT_FOC` – 電圧トルク制御を使用したダイレクト FOC スタートアップ、2.3.4 を参照

ターゲット値の入力選択

`#define SETTING_TARGET_SPEED` `SET_TARGET_SPEED`

- ターゲット値の入力方法を定義します。

Note: 次のオプションのうち、同時に使用できるオプションは1 つだけです。さらに、`MY_FOC_CONTROL_SCHEME` に応じて、入力は`Target_Speed`, `Target_Torque`, または `Target_Voltage` として保存されます。

オプション:

- SET_TARGET_SPEED – ターゲット値を保存するために API 関数を利用できます。また、µC/Probe はアップデートにも使用されます。
- BY_POT_ONLY – ポテンショメーターは、ADC 用のピンを介してモーターの動作を制御するために使用されます。
- BY_UART_ONLY –リファレンス速度は UART 通信を介して設定されます。(XMC4400/4800 は利用できません)

SVM スイッチング

`#define SVM_SWITCHING_SCHEME STANDARD_SVM_7_SEGMENT`

- SVM スイッチング方式を定義します。

オプション:

- STANDARD_SVM_7_SEGMENT – 7 セグメントスイッチングモード(2.5.1 を参照)
- STANDARD_SVM_5_SEGMENT – 5 セグメントスイッチングモード(2.5.2 を参照)

µC/プローブ GUI 選択

`#define uCPROBE_GUI_OSCILLOSCOPE ENABLED`

- Micrium µC/Probe GUI およびオシロスコープツールのファームウェアサポートを有効または無効にします。有効にすると、`pmsm_foc_controlloop_isr()`で呼び出されます。

保護および制限

`#define VDC_UNDER_OVERVOLTAGE_PROTECTION ENABLED`

- DC Link 電圧保護を有効または無効にします。

`#define VDC_UNDER_OVERVOLTAGE_PERCENTAGE (20U)`

- 過電圧と低電圧のリミットを+-20%に設定します。

`#define OVERCURRENT_PROTECTION ENABLED`

- DC Link 電流保護を有効または無効にします。(XMC4400/4800 には実装されていません)

`#define USER_IDC_MAXCURRENT_A (10.0f)`

- DC リンクの最大電流制限。この制限は、過電流保護機能が有効になっている場合にチェックされます。この制限に達すると、リファレンス速度が低下します (2.9 の過電流保護を参照)。これは XMC4400/4800 には実装されていません。

`#define VDC_MAX_LIMIT ((VADC_DCLINK * 19U)>>4)`

- ランプダウン動作時の DC Link 最大電圧のリミットを設定します。デフォルトの設定は、ノミナルの DC Link 電圧より 18.7%高い値です。このリミット値は、ハードウェアの設計に応じて変更する必要があります。

`#define WATCH_DOG_TIMER ENABLED`

- ウォッチドッグタイマー機能を有効または無効にします。

7.1.2 カスタムキットの設定

Controller_Card

```
#define MCUCARD_TYPE                    defined by PMSM_FOC_HARDWARE_KIT
#define MCUCARD_TYPE_PATH               defined by PMSM_FOC_HARDWARE_KIT
```

- デフォルトの設定ではハードウェアボードに基づき定義されています。カスタム MCU カードまたは組合せを設定するには、ハードウェアボード (CUSTOM_KIT) を使用する必要があります。ハードウェアボードの組み合わせのみテストされています。

MCUCARD_TYPE のオプション:

- CUSTOM_MCU
- EVAL_M1_1302
- KIT_XMC13_BOOT_001
- KIT_XMC1300_DC_V1
- KIT_XMC14_BOOT_001
- KIT_XMC1400_DC_V1
- KIT_XMC4400_DC_V1
- KIT_XMC4400_EE1_001
- KIT_XMC4800PSOC_IOT

Note: 選択に応じて MCUCARD_TYPE_PATH を適合させる必要があります。

Inverter_Card

```
#define INVERTERCARD_TYPE               defined by PMSM_FOC_HARDWARE_KIT
#define INVERTERCARD_TYPE_PATH          defined by PMSM_FOC_HARDWARE_KIT
```

- デフォルトの設定ではハードウェアボードに基づき定義されています。カスタムインバーターカードまたは組合せを設定するには、ハードウェアボード (CUSTOM_KIT) を使用する必要があります。ハードウェアボードの組み合わせのみテストされています。
- CUSTOM_INVERTER
- EVAL_M1_05_65A
- EVAL_M1_05F310
- EVAL_M1_CM610N3
- KIT_MOTOR_DC_250W_24V
- PMSM_LV15W
- POWERINVERTER_750W
- KIT_XMC4X_MOT_GPDLV_001

Note: 選択に応じて INVERTERCARD_TYPE_PATH を適合させる必要があります。

モーター

```
#define MOTOR_TYPE                      defined by PMSM_FOC_HARDWARE_KIT
#define MOTOR_TYPE_PATH                 defined by PMSM_FOC_HARDWARE_KIT
```

- デフォルトの設定ではハードウェアボードに基づき定義されています。カスタムモーターまたは組合せを設定するには、ハードウェアボード (CUSTOM_KIT) を使用する必要があります。ハードウェアボードの組み合わせのみテストされています。
- CUSTOM_MOTOR
- MAXON_MOTOR_267121
- NANOTEC_MOTOR_DB42S03

Note: 選択に応じて INVERTERCARD_TYPE_PATH を適合させる必要があります。

7.1.3 高度なユーザー設定

ADC 固有の設定

`#define ADC_STARTUP_CALIBRATION` `DISABLED`

- ADC スタートアップキャリブレーション機能を有効または無効にします。XMC1302AA step を使用している場合はこの機能を有効にしてください。 (XMC4400/4800 では実装されません。)

`#define ADC_ALTERNATE_REFERENCE` `DISABLED`

- ADC オルタネートリファレンス機能を有効または無効にします。有効にすると、チャンネル 0 がリファレンスとして使用されます。 (XMC4400/4800 では実装されません。)

`#define ADC_ALTERNATE_REF_PHASEUVW` `DISABLED`

- UVW 相のオルタネートリファレンスを有効または無効にします。このオプションは、オルタネートリファレンスが有効になっている場合にのみ使用できます。 (XMC4400/4800 では実装されません。)

`#define ADC_ALTERNATE_REF_SINGLESHUNT` `DISABLED`

- シングルシャントのオルタネートリファレンスを有効または無効にします。このオプションは、オルタネートリファレンスが有効になっている場合にのみ使用できます。 (XMC4400/4800 では実装されません。)

`#define MOTOR_HOLD_THRESHOLD`
`Defined by MY_FOC_CONTROL_SCHEME`

- 有効デジタル値の最小値を定義します。この値はポテンショメーターの実動作とアナログからデジタルへの変換のために必要です。この値を下回る値はすべて、ゼロまたはオフと見なされます。

`#define FOC_EXIT_SPEED` `(SPEED_LOW_LIMIT * 3)`

- モーター速度がランプダウンから抜け出すリミット値を定義します。ステートは `FOC_CLOSED_LOOP_BREAKING` から `MOTOR_STOP` に移ります。

`#define SPEED_LOW_LIMIT` `SPEED_LOW_LIMIT_RPM`

`#define SPEED_LOW_LIMIT_RPM` `calculated`

- この値は、許容される最低速度として使用されます。`USER_SPEED_LOW_LIMIT_RPM` から計算されますが、MCU の計算用にスケーリングされています。

セカンダリーループ コールバック

`#define PMSM_FOC_SECONDARYLOOP_CALLBACK` `DISABLED`

- セカンダリーループのコールバックを有効または無効にします。有効にした場合は、ユーザーが関数を作成する必要があります。コールバック関数は、SRAM の消費を減らすために、デフォルトではフラッシュメモリ内で実行されます。高速で実行する場合は、SRAM コード属性を手動で追加する必要があります。

`#define USER_SECONDARY_LOOP_FREQ_HZ` `(1000U)`

- この値は、セカンダリーループのターゲット周波数を Hz 単位で定義します。このループは、通信やウォッチドッグサービスなどの低速なタスクに使用されます。この周波数は、`USER_CCU8_PWM_FREQ_HZ` の分周した値 (デフォルトは 20 kHz) となるように意図されています。

FOC 制御

`#define DQ_DECOUPLING ENABLED`

- dq デカップリング機能を有効または無効にします。

7.1.4 トルク制御仕様

これらの設定は、次の場合にのみ使用できます。

`MY_FOC_CONTROL_SCHME` が `TRQUE_CONTROLLED_DIRECT_FOC` に設定されている場合。

`#define USER_IQ_CURRENT_ALLOWED_A (2.0f)`

- リファレンストルク電流の上限リミットをアンペア単位で設定します。

`#define USER_IQ_REF_LOW_LIMIT (0U)`

- リファレンストルク電流の下限リミットを設定します。

`#define USER_IQ_REF_HIGH_LIMIT (32768* USER_IQ_CURRENT_ALLOWED_A/I_MAX_A)`

- リファレンストルク電流の上限リミットを 1Q15 形式でスケーリングします。

`#define USER_IQ_RAMPUP (10U)`

- リニアランプジェネレーターで使用されるトルク電流のランプアップステップです。
- 1 ステップは `(1/32768) * I_MAX_A` で、`I_MAX_A` は `(5.0 V/(USER_R_SHUNT_OHM * OP_GAIN_FACTOR) ) / 2` に等しくなります。

`#define USER_IQ_RAMPDOWN (10U)`

- リニアランプジェネレーターで使用されるトルク電流のランプダウンステップです。
- 1 ステップは `(1/32768) * I_MAX_A` で、`I_MAX_A` は `(5.0 V/(USER_R_SHUNT_OHM * OP_GAIN_FACTOR) ) / 2` に等しくなります。

`#define USER_IQ_RAMP_SLEWRATE (50U)`

- I_q をランプアップ/ランプダウンする周波数を定義します。
- `USER_IQ_RAMP_SLEWRATE * PWM` サイクルごとに、I_q を `USER_IQ_RAMPUP` ステップで増加させるか、I_q を `USER_IQ_RAMPDOWN` ステップで減少させます。

7.1.5 VQ 制御仕様 (V/f)

```
#define USER_VQ_VOLTAGE_ALLOWED_V                    (10U)
```

- トルク電圧のリミット値をボルト単位で設定します。この値は VREF_MAX_V 未満でなければいけません。

```
#define USER_VQ_REF_LOW_LIMIT                        (0U)
```

- リファレンストルク電圧の下限リミット値を設定します。

```
#define USER_VQ_REF_HIGH_LIMIT
      (32768* USER_VQ_VOLTAGE_ALLOWED_V/VREF_MAX_V)
```

- リファレンストルクトルク電圧のリミット値を 1Q15 形式にスケーリングします。
- VREF_MAX_V は、DC Link 電圧を 3 の平方根で割った値として定義されます。

```
#define USER_VQ_RAMPUP                               (2U)
```

- リニアランプジェネレーターのトルク電圧をランプアップさせるステップ数を定義します。

```
#define USER_VQ_RAMPDOWN                             (2U)
```

- リニアランプジェネレーターのトルク電圧をランプダウンするステップ数を定義します。

```
#define USER_VQ_SLEWRATE                             (10U)
```

- V_q をランプアップ/ランプダウンする周波数を定義します。USER_VQ_RAMP_SLEWRATE * PWM サイクルごとに、V_q は USER_VQ_RAMPUP ステップでランプアップするか、V_q は USER_VQ_RAMPDOWN ステップでランプダウンします。

7.1.6 MET 仕様

```
#define USER_MET_THRESHOLD_HIGH                      (64U)
```

- MET ステートにおける速度ヒステリシス制御の上限しきい値を定義します。

```
#define USER_MET_THRESHOLD_LOW                       (16U)
```

- MET ステートにおける速度ヒステリシス制御の下限しきい値を定義します。

```
#define USER_MET_LPF                                 (2U)
```

- MET しきい値計算におけるローパスフィルタ係数。Y[n] = Y[n-1] + (X[n] – Y[n-1]) >> USER_MET_LPF

7.2 ハードウェア設定

詳細なハードウェア設定により、コントローラーカード, インバーターカード, およびモーターへの特別化が可能になります。この設定により、ハードウェア設定をアプリケーションやボードに適合させることができます。ピン配置や最大速度などの設定については、こちらを参照してください。

設定は、3 つの種類に分けられます。

- *PMSM_FOC\Configuration\Controller_Card*
- *PMSM_FOC\Configuration\Inverter_Card*
- *PMSM_FOC\Configuration\Motors*

7.2.1 コントローラーカード

Table 21 GPIO コンフィギュレーション

定義	XMC1300/XMC1400 の値	XMC4400 の値	XMC4800 の値	注意事項
`#define TRAP_PIN`	P0_12	P0_7	P5_0	外部 CCU80 CTrap 入力ピン割り当て
`#define INVERTER_EN_PIN`	P0_11	P0_12	P8_9	インバータースイッチ/ゲートドライバのイネーブルピンに接続。
`#define PHASE_U_HS_PIN`	P0_0	P0_5	P1_15	-
`#define PHASE_U_LS_PIN`	P0_1	P0_2	P1_12	-
`#define PHASE_V_HS_PIN`	P0_7	P0_4	P1_14	-
`#define PHASE_V_LS_PIN`	P0_6	P0_1	P1_11	-
`#define PHASE_W_HS_PIN`	P0_8	P0_6	P1_13	-
`#define PHASE_W_LS_PIN`	P0_9	P0_11	P1_10	-

Table 22　XMC™ オルタネート出力ピン レジスタ設定

定義	XMC1300/XMC1400 の値	XMC4400/4800 の値
#define PHASE_U_HS_ALT_SELECT	XMC_GPIO_MODE_OUTPUT _PUSH_PULL_ALT5	XMC_GPIO_MODE_OUTPUT _PUSH_PULL_ALT3
#define PHASE_U_LS_ALT_SELECT	XMC_GPIO_MODE_OUTPUT _PUSH_PULL_ALT5	XMC_GPIO_MODE_OUTPUT _PUSH_PULL_ALT3
#define PHASE_V_HS_ALT_SELECT	XMC_GPIO_MODE_OUTPUT _PUSH_PULL_ALT5	XMC_GPIO_MODE_OUTPUT _PUSH_PULL_ALT3
#define PHASE_V_LS_ALT_SELECT	XMC_GPIO_MODE_OUTPUT _PUSH_PULL_ALT5	XMC_GPIO_MODE_OUTPUT _PUSH_PULL_ALT3
#define PHASE_W_HS_ALT_SELECT	XMC_GPIO_MODE_OUTPUT _PUSH_PULL_ALT5	XMC_GPIO_MODE_OUTPUT _PUSH_PULL_ALT3
#define PHASE_W_LS_ALT_SELECT	XMC_GPIO_MODE_OUTPUT _PUSH_PULL_ALT5	XMC_GPIO_MODE_OUTPUT _PUSH_PULL_ALT3

- XMC™ オルタネート出力ピン レジスタ値の設定。CCU80 PWM 出力が選択されています。
- オルタネート出力設定については、XMC1300 または XMC1400 リファレンスマニュアル [1] を参照してください。

Table 23　プライムラリーループと SVM リソース (CCU8)

定義	XMC1300/XMC1400 の値	XMC4400 の値	XMC4800 の値
#define CCU8_MODULE	CCU80	CCU80	CCU81
#define CCU8_MODULE_PHASE_U	CCU80_CC80	CCU80_CC80	CCU81_CC80
#define CCU8_MODULE_PHASE_V	CCU80_CC81	CCU80_CC81	CCU81_CC81
#define CCU8_MODULE_PHASE_W	CCU80_CC82	CCU80_CC83	CCU81_CC82
#define CCU8_MODULE_ADC_TR	CCU80_CC83	CCU80_CC82	CCU81_CC83
#define CCU8_MODULE_PRESCALER_VALUE	(0U)	(0U)	(0U)

- SVM 生成用に XMC CCU8 モジュールを割り当てます。複数のモジュールが利用可能ですがモジュール数はデバイスによって異なります。
- U 相, V 相, W 相の PWM 生成および電流トリガーに CCU8 モジュールのタイマースライスを割り当てます。
- ADC 変換トリガー (TR) に CCU8 モジュールタイマースライスを割り当てます。プリスケーラは分解能に影響します。周波数を変更すると、この値は再計算される必要があります。可能な限り常に小さい値にする必要があります。

Table 24　セカンダリーループ リソース (CCU4)

定義	XMC1300/XMC1400 の値	XMC4400/4800 の値
#define SECONDARY_LOOP_MODULE	CCU40	CCU41
#define SECONDARY_LOOP_SLICE	CCU40_CC42	CCU41_CC43
#define SECONDARY_LOOP_MODULE_NUM	0	1
#define SECONDARY_LOOP_SLICE_PRESCALER	(2U)	(1U)
#define SECONDARY_LOOP_SLICE_NUM	(2U)	(3U)
#define SECONDARY_LOOP_SLICE_SHADOW_TRANS_ENABLE_Msk	XMC_CCU4_SHADOW_TRANSFER_SLICE_2	XMC_CCU4_SHADOW_TRANSFER_SLICE_1

- セカンダリーループに CCU4 モジュールタイマースライスを割り当てます。スライス名, スライス番号, およびスライスシャドウ転送有効化マスクを一致させる必要があります。プリスケーラは分解能に影響します。周波数を変更すると、この値は再計算される必要があります。

XMC1300/XMC1400 の 3 シャント同期変換用 ADC リソース (VADC)

```
#define VADC_IU_G1_CHANNEL                (3U)          // P2.11
#define VADC_IU_G0_CHANNEL                (4U)          // P2.11
#define VADC_IV_G1_CHANNEL                (2U)          // P2.10
#define VADC_IV_G0_CHANNEL                (3U)          // P2.10
#define VADC_IW_G1_CHANNEL                (4U)          // P2.9
#define VADC_IW_G0_CHANNEL                (2U)          // P2.9
```

  - チャネル番号はピンと等価です。デフォルトの設定では、G0 と G1 のチャネル IU, IV, IW は同じピンに接続されているため、使用するピンを節約できます。

Note: グループチャネル番号とピンの接続の詳細については、リファレンスマニュアルを参照してください。

XMC1300/XMC1400 の 3 シャント非同期変換用 ADC リソース (VADC)

```
#define VADC_IU_GROUP                    VADC_G1
#define VADC_IU_GROUP_NO                 (1U)
#define VADC_IV_GROUP                    VADC_G1
#define VADC_IV_GROUP_NO                 (1U)
#define VADC_IW_GROUP                    VADC_G1
#define VADC_IW_GROUP_NO                 (1U)
```

- この設定では、測定に使用する ADC グループを選択します。グループ名と番号を一致させる必要があります。さらに、デフォルトでは、IU, IV, IW のグループは同じである必要があります。2 つのグループを使用する場合は、グループごとに `.external_trigger = true` を指定した XMC_VADC_QUEU_ENTRY が 1 つ必要です。

```
#define VADC_IU_CHANNEL                  (3U)        // P2.11
#define VADC_IU_RESULT_REG               (3U)
#define VADC_IV_CHANNEL                  (2U)        // P2.10
#define VADC_IV_RESULT_REG               (2U)
#define VADC_IW_CHANNEL                  (4U)        // P2.9
#define VADC_IW_RESULT_REG               (4U)
```

- チャネル番号はピンと等価です。グループチャネル番号とピンの接続の詳細については、リファレンスマニュアルを参照してください。result レジスタを選択する際の唯一の制限は、すべてのレジスタに固有の番号がなければいけないことです。

XMC4400 の 3 シャント同期変換用 ADC リソース (VADC)

```
#define VADC_IU_GROUP                    VADC_G0
#define VADC_IU_GROUP_NO                 (0U)
#define VADC_IV_GROUP                    VADC_G1
#define VADC_IV_GROUP_NO                 (1U)
#define VADC_IW_GROUP                    VADC_G2
#define VADC_IW_GROUP_NO                 (2U)
#define VADC_IU_CHANNEL                  (1U)
#define VADC_IU_RESULT_REG               (15U)
#define VADC_IV_CHANNEL                  (7U)
#define VADC_IV_RESULT_REG               (3U)
#define VADC_IW_CHANNEL                  (1U)
#define VADC_IW_RESULT_REG               (0U)
```

XMC4800 の 3 シャント同期変換用 ADC リソース (VADC)

```
#define VADC_IU_GROUP                    VADC_G0
```

```
#define VADC_IU_GROUP_NO                    (0U)
#define VADC_IV_GROUP                       VADC_G1
#define VADC_IV_GROUP_NO                    (1U)
#define VADC_IW_GROUP                       VADC_G2
#define VADC_IW_GROUP_NO                    (2U)
#define VADC_IU_CHANNEL                     (0U)
#define VADC_IU_RESULT_REG                  (0U)
#define VADC_IV_CHANNEL                     (0U)
#define VADC_IV_RESULT_REG                  (1U)
#define VADC_IW_CHANNEL                     (4U)
#define VADC_IW_RESULT_REG                  (4U)
```

シングルシャント変換用 ADC リソース (VADC) (XMC1300/XMC1400 のみ)

```
#define VADC_ISS_GROUP                      VADC_G1
#define VADC_ISS_GROUP_NO                   (1U)
#define VADC_ISS_CHANNEL                    (1U)        // P2.7
#define VADC_ISS_RESULT_REG                 (15U)
```

- この設定にて、測定に使用する ADC グループを決定します。グループ名と番号を一致させる必要があります。チャネル番号はピンに相当します。グループチャネル番号とピンの接続の詳細については、リファレンスマニュアルを参照してください。グループ, チャネル, または result レジスタの選択に関するその他の制限はありません。

DC Link 電圧測定用 ADC リソース (VADC)

```
#define VADC_VDC_GROUP                      VADC_G1
#define VADC_VDC_GROUP_NO                   (1U)
#define VADC_VDC_CHANNEL                (5U) //(1U) in XMC4400, (2U) in XMC4800
#define VADC_VDC_RESULT_REG             (5U) //(4U) in XMC4400, (2U) in XMC4800
```

- この設定にて、測定に使用する ADC グループを決定します。グループ名と番号を一致させる必要があります。チャネル番号はピンに相当します。グループチャネル番号とピンの接続の詳細については、リファレンスマニュアルを参照してください。グループ, チャネル, または result レジスタの選択に関するその他の制限はありません。

平均 DC Link 電圧測定用 ADC リソース(VADC) (XMC1300/XMC1400 のみ)

```
#define VADC_IDC_GROUP                      VADC_G0
#define VADC_IDC_GROUP_NO                   (0U)
#define VADC_IDC_CHANNEL                    (6U)        // P2.1
#define VADC_IDC_RESULT_REG                 (6U)
```

- この設定にて、測定に使用する ADC グループを決定します。グループ名と番号を一致させる必要があります。チャネル番号はピンに相当します。グループチャネル番号とピンの接続の詳細については、リファレンスマニュアルを参照してください。グループ, チャネル, または result レジスタの選択に関するその他の制限はありません。

ポテンショメーター測定用 ADC リソース (VADC)

```
#define VADC_POT_GROUP                          VADC_G1
#define VADC_POT_GROUP_NO                       (1U)
#define VADC_POT_CHANNEL            (7U)   //(5U) in XMC4400, (3U) in XMC4800
#define VADC_POT_RESULT_REG         (7U)   //(14U) in XMC4400 and XMC4800
```

- この設定にて、測定に使用する ADC グループを決定します。グループ名と番号を一致させる必要があります。チャネル番号はピンに相当します。グループチャネル番号とピンの接続の詳細については、リファレンスマニュアルを参照してください。グループ, チャネル, または result レジスタの選択に関するその他の制限はありません。

UART ピン設定 (UART) (XMC1300/XMC1400 のみ)

```
#define UART_ENABLE                             USIC0_CH1_P1_2_P1_3
```

- SETTING_TARGET_SPEED が BY_UART_ONLY に設定されている場合、入出力ピンを設定できます。

オプション:

- USIC0_CH0_P1_4_P1_5 – UART チャネル 0 がモーターを制御するコマンドを受信するために使用されます。ポートピン P1.4 と P1.5 は、データの受信と送信を行うように設定されています。ポテンショメーターの ADC 変換結果は破棄されます。
- USIC0_CH1_P1_2_P1_3 – UART チャネル 1 がモーターを制御するコマンドを受信するために使用されます。ポートピン P1.2 と P1.5 は、データの送信と受信を行うように設定されています。ポテンショメーターの ADC 変換結果は破棄されます。

デバッグ用 PWM (CCU4) (XMC1300/XMC1400 のみ)

```
#define DEBUG_PWM_0_ENABLE                      (0U)
#define DEBUG_PWM_1_ENABLE                      (0U)
```

- 最大 2 つのデバッグ PWM を使用できます。この設定にて、デバッグ PWM のインスタンス 0 またはインスタンス 1 を有効にします。

オプション:

- 0 - 無効
- 1 – 有効

```
#define DEBUG_PWM_CCU4_MODULE                   CCU40
#define DEBUG_PWM_PERIOD_CNTS                   (400U)
#define DEBUG_PWM_50_PERCENT_DC_CNTS ((uint16_t)(DEBUG_PWM_PERIOD_CNTS >> 1))
```

- この設定にて、使用するモジュール, 周期カウント経由の周波数, および 50% dc を定義します。

```
#define REVERSE_CRS_OR_0                        (- Tmp_CRS)
```

- この設定にて、負の値がどのように扱われるかを定義します。

オプション:

- 0 - ゼロ
- Tmp_CRS - 絶対値

```
#define DEBUG_PWM_0_SLICE                       CCU40_CC40
```

```
#define DEBUG_PWM_0_SLICE_NUM                                    (0U)
#define DEBUG_PWM_0_SLICE_SHADOW_TRANS_ENABLE_Msk XMC_CCU4_SHADOW_TRANSFER_SLICE_0
```

- CCU4 モジュールのタイマー スライスを debugPWM 0 に割り当てます。スライス名, スライス番号, およびスライスシャドウ転送イネーブルマスクがすべて一致している必要があります。最大周波数をサポートするために、プリスケーラはデフォルトで 0 です。

```
#define DEBUG_PWM_0_PORT                                         XMC_GPIO_PORT1
#define DEBUG_PWM_0_PIN                                          (0U)
#define DEBUG_PWM_0_ALT_OUT
      XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT2
```

- これにより、デバッグインスタンス 0 の PWM の出力が設定されます。関連するポートピン, オルタネート出力, および CCU4 スライスをすべて選択する必要があります。詳細については、リファレンスマニュアルを参照してください。

```
#define DEBUG_PWM_1_SLICE                                        CCU40_CC41
#define DEBUG_PWM_1_SLICE_NUM                                    (1U)
#define DEBUG_PWM_1_SLICE_SHADOW_TRANS_ENABLE_Msk XMC_CCU4_SHADOW_TRANSFER_SLICE_1
```

- debugPWM 1 に CCU4 モジュールのタイマースライスを割り当てます。スライス名, スライス番号, およびスライスシャドウ転送イネーブルマスクがすべて一致している必要があります。最大周波数をサポートするために、プリスケーラはデフォルトで 0 です。

```
#define DEBUG_PWM_1_PORT                                         XMC_GPIO_PORT0
#define DEBUG_PWM_1_PIN                                          (4U)
#define DEBUG_PWM_1_ALT_OUT
      XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT4
```

- これにより、デバッグインスタンス 1 の PWM の出力が設定されます。関連するポートピン, オルタネート出力, および CCU4 スライスを選択する必要があります。詳細については、リファレンスマニュアルを参照してください。

7.2.2 電流および電圧検出用 インバーターボード設定

概要

```
#define USER_CCU8_PWM_FREQ_HZ                    (20000U)
```

- PWM 周波数を Hz 単位で定義します。これは最速のループであり、制御ループです。FOC の主なタスクは、このループまたはその一部で実行されます。PMSM FOC ソフトウェアは、最大 25 kHz (場合によっては最大 30 kHz) をサポートします。

```
#define USER_MAX_ADC_VDD_V                       (5.0f)
```

- XMC™ ADC の最大入力をボルト単位で定義します。XMC1000 ファミリーは広い入力範囲を有します。一般的なのは 5 V と 3.3 V です。入力の値はフロート (float) 型です。ほとんどの物理的なスケーリングは、この定義にリンクされます。

電源電圧

```
#define USER_VDC_LINK_V                          (24.0f)
```

- モーターのインバーターボードで使用されるノミナルの DC 電圧をボルト単位で定義します。過電圧および低電圧状態のスケーリングおよびリミットチェック +-20% (デフォルト) に使用されます。

```
#define USER_DC_LINK_DIVIDER_RATIO               (5.1f/(5.1f+47.0f))
//In XMC4400 and XMC4800 (5.6f)/(5.6f+56.0f)
```

- 2.6 を参照してください。DC リンク分圧比は R2/(R1+R2) です。
- この値は、ADC 変換結果のスケーリングに影響します。

出力, ブリッジドライバ, および B6 ブリッジ

```
#define USER_DEAD_TIME_US                        (0.75f)
```

- このマクロは、デッドタイムをマイクロ秒単位で定義します。この値は、スイッチとブリッジドライバに従って定義する必要があります。値が小さすぎるとハイサイドスイッチとローサイドスイッチが短絡し、値を大きいと印加できる最大電圧が減少します。デフォルト設定では、立ち上りエッジと立ち下りエッジに同じデッドタイムが適用されます。補正されていない場合、デッドタイムによって一定の誤差が加わります。

```
#define USER_BOOTSTRAP_PRECHARGE_TIME_MS         (20U)
```

- これは、ブートストラップにおけるコンデンサの初期プリチャージ時間 (ミリ秒単位) です。ドライバ IC とドライバ回路によっては、この時間を調整する必要があります。一部のドライバ IC はブートストラップチャージポンプを実装しており、ブートストラップは必要ありません。この時間が短かすぎると、ハイサイドスイッチは最初のサイクルにてオンになりません。この時間が長すぎる場合は、開始要求と開始の間の時間がブートストラップ時間だけ遅れることを除いて、欠点はありません。

```
#define CCU8_INPUT_TRAP_LEVEL
        XMC_CCU8_SLICE_EVENT_LEVEL_SENSITIVITY_ACTIVE_LOW
```

- ゲートドライバのフォールト信号のアクティブレベルに従って、CCU8 入力トラップ信号の論理レベルを定義します。

オプション:

- `XMC_CCU8_SLICE_EVENT_LEVEL_SENSITIVITY_ACTIVE_LOW`
- `XMC_CCU8_SLICE_EVENT_LEVEL_SENSITIVITY_ACTIVE_HIGH`

`#define GATE_DRIVER_INPUT_LOGIC PASSIVE_LOW`

- ゲートドライバの論理レベルを定義します。この定義から、`MOTOR_COASTING_xx` と `MOTOR_RUN_xx` の設定が定義されます。

オプション:

- `PASSIVE_HIGH`
- `PASSIVE LOW`

`#define INVERTER_ENABLE_PIN (1U)`

- インバーターのピンの論理を定義します。

オプション:

- 1 = アクティブ HIGH
- 0 = アクティブ LOW

電流測定

`#define INTERNAL_OP_GAIN DISABLED`

- XMC VADC は内部にゲインアンプを有しています。これを使用する場合は外部にオペアンプは必要ありません。この設定は、ADC 内部ゲインを有効または無効にします。この設定に基づいて、電流の測定の設定は変更されます。

`#define USER_R_SHUNT_OHM (0.05f)`

- 電流シャント抵抗の抵抗値 (単位Ω)です。DC Link 測定での値、またはレッグシャント測定でのそれぞれの相の値。この値は、I_MAX を計算するために使用され、これは後でトルク制御モードや角度推定などの他の FOC 機能でリファレンス電流を制限するために使用されます。

`#define USER_DC_SHUNT_OHM (0.05f)`

- DC Link シャント抵抗の抵抗値 (単位Ω)。
- この値は、DC Link シャントによる過電流保護に使用されます。

電流測定: 内部オペアンプ有効

`#define OP_GAIN_FACTOR (3U)`

- 内部 ADC チャネルゲインファクターが有効になっている場合、`OP_GAIN_FACTOR` は XMC™ 内部ゲインファクター (1, 3, 6, および 12) に従います。
- ハードウェアが使用可能なゲインファクターのみ使用する必要があります。

`#define USER_RIN_OFFSET_KOHM (2.0f)`

- オフセット抵抗の値 (kΩ 単位 浮動小数点数フォーマット)
- この目的については、Figure 37 に説明があります。

`#define USER_RIN_PULL_UP_KOHM (10.0f)`

- オフセット抵抗の値 (kΩ 単位 浮動小数点数フォーマット)
- この目的については、Figure 37 に説明があります。

電流測定: 内部オペアンプ無効

`#define USER_RIN_PHASECURRENT_KOHM (1.0f)`

- 相電流アンプの RIN 抵抗値 (kΩ 単位) (Figure 36 を参照)。

`#define USER_R_PHASECURRENT_FEEDBACK_KOHM (16.4f)`

- 相電流アンプのフィードバック抵抗値 (kΩ 単位)。
- この USER_RIN_PHASECURRENT_KOHM により、電流アンプのゲインが計算されます。Figure 36 参照。

`#define USER_RIN_DCCURRENT_KOHM (10.0f)`

- DC Link 電流アンプの RIN の抵抗値 (kΩ 単位)。Figure 36 参照。

`#define USER_R_DCCURRENT_FEEDBACK_KOHM (75.0f)`

- DC Link 電流アンプのフィードバック抵抗値 (kΩ 単位)。
- USER_RIN_DCCURRENT_KOHM により、DC Link 電流アンプのゲインが計算されます。Figure 36 参照。

7.2.3 モーター固有の設定

7.2.3.1 モーターパラメーター

```
#define USER_MOTOR_R_PER_PHASE_OHM                    (6.8f)
```

- モーターの各相の中性点までの抵抗をオーム単位で定義します。

```
#define USER_MOTOR_L_PER_PHASE_uH                     (3865.0f)
```

- モーターの各相の中性点までのインダクタンスを µH 単位で定義します。
- IPMSM (interior permanent magnet synchronous motor) ブラシレス DC モーターの場合、モーターの 1 つの相の q 軸インダクタンス (Lq) を用います。

```
#define USER_MOTOR_POLE_PAIR                          (4U)
```

- モーターの対局数で、ローターの回転数を計算するために使用されます。

```
#define USER_SPEED_HIGH_LIMIT_RPM                     (4530.0f)
```

- この値は、最大許容ターゲット速度として使用されます。追加の制御パラメーターは、この値から計算されます。モーターのノミナルの速度を使用する必要があります。

```
#define USER_SPEED_LOW_LIMIT_RPM                (USER_SPEED_HIGH_LIMIT_RPM/30)
```

- この値は、最小許容ターゲット速度として使用されます。センサーレスモーター制御では、相電流の測定が必須です。低トルク (通常は低速) では、十分なモーター制御を行うことはできません。最小速度はアプリケーションによって異なります。デフォルトでは、SPEED_HIGH_LIMIT の 30%が設定されています。

```
#define USER_SPEED_RAMPUP_RPM_PER_S                   (500U)
#define USER_SPEED_RAMPDOWN_RPM_PER_S                 (500U)
```

- スムーズな制御を確保するために、ターゲット入力と PI コントローラーの間にランプジェネレーターが実装されています。この設定にて、最大ランプアップと最大ランプダウンを RPM/秒で定義します。

```
#define PWM_THRESHOLD_USEC                            (2U)
```

- 3 レッグシャント測定モードにおける電流測定の最小時間。この時間を下回ると、2 レッグシャント測定が適用されます。この時間には、電流リンギングと ADC の測定時間が含まれます。

```
#define USER_STARUP_SPEED_RPM                         (0U)
```

- V/f オープンループの初速度を RPM で定義します。

```
#define USER_STARTUP_SPEED_THRESHOLD_RPM              (500U)
```

- オープンループ制御からクローズドループ制御に遷移する速度を定義します。

```
#define USER_STARTUP_VF_OFFSET_V                      (1.0f)
```

- V/f オープンループ制御の起動電圧オフセットです。

```
#define USER_STARTUP_VF_SLEWRATE_V_PER_HZ             (0.1f)
```

- V/f オープンループ制御の起動スルーレート (V/Hz) です。

```
#define USER_ROTOR_PREPOSITION_TIME_MS                (100)
```

- モーターの事前位置決めのための時間 (ミリ秒単位) です。

7.2.3.2 PI 設定

各モーターのデフォルトの PI 設定は、「*..\PMSM_FOC\Configuration\Motors*」に格納されています。インフィニオン XMC1000 モーター制御アプリケーションキット、KIT_XMC1X_AK_MOTOR_001 で使用される「Maxon motor 267121」の場合、*pmsm_foc_motor_MAXON_MOTOR_267121.h* ファイルに関連する設定が含まれます。

速度コントローラーの PI 設定

`#define PI_SPEED_KP (1U << 15)`

- 速度制御ブロックの比例ゲインの設定です。最小値は 1、最大値は 32767 です。

`#define PI_SPEED_KI (3U)`

- 速度コントローラーの積分ゲインの設定です。最小値は 0、最大値は 32767 です。

`#define PI_SPEED_SCALE_KPKI (10U + USER_RES_INC)`

- スケーリングファクター K_P と K_I の設定です。
- K_P と K_I は 2^PI_SPEED_SCALE_KPKI でスケールアップされます。
  USER_RES_INC は *pmsm_foc_const.h* ファイルで定義される定数です。PI_SPEED_SCALEKPKI の最大値は 15、最小値は USER_RES_INC です。

`#define PI_SPEED_IK_LIMIT_MIN (-(((1<<15)*3)>>2))`

- 積分バッファの最小出力値の設定です。最小値は -32768 です。

`#define PI_SPEED_IK_LIMIT_MAX (((1<<15)*3)>>2)`

- 積分バッファの最大出力値の設定です。最大値は 32767 です。

`#define PI_SPEED_UK_LIMIT_MIN (16U)`

- 速度 PI 制御ブロックの最小出力値の設定です。最小値は -32768 です。

`#define PI_SPEED_UK_LIMIT_MAX (32767U)`

- 速度 PI 制御ブロックの最大出力値の設定です。最大値は 32767 です。

`#define USER_RES_INC (3U)`

- この定義により、角度の計算分解能が増加します。8U を超えてはならず、PI_SPEED_SCALE_KPKI は 15U を超えてはいけません。

トルクコントローラーの PI 設定

`#define PI_TORQUE_KP (USER_DEFAULT_IQID_KP)`

- トルク制御ブロックの比例ゲインの設定です。ゲインの値は、次の式によって決定されます: $K_P = \omega_c L \times A$ (2.11 を参照)。

`#define PI_TORQUE_KI (USER_DEFAULT_IQID_KI)`

- トルク制御ブロックの積分ゲインの設定です。ゲインの値は、次の式によって決定されます: $K_I = RT_S K_P / L$ (2.11 を参照)。

`#define PI_TORQUE_SCALE_KPKI (SCALING_CURRENT_KPKI)`

- スケーリングファクター K_P と K_I の設定です。K_P と K_I は 2^PI_TORQUE_SCALE_KPKI でスケールアップされます。

`#define PI_TORQUE_IK_LIMIT_MIN (-32768)`

- 積分バッファの最小値を設定します。最小値は -32768 です。

```
#define PI_TORQUE_IK_LIMIT_MAX                    (32767)
```

- 積分バッファの最大値を設定します。最大値は 32767 です。

```
#define PI_TORQUE_UK_LIMIT_MIN                    (-32768)
```

- トルク PI 制御ブロックの最小出力値の設定です。最小値は -32768 です。

```
#define PI_TORQUE_UK_LIMIT_MAX                    (32767)
```

- トルク PI 制御ブロックの最大出力値の設定です。最大値は 32767 です。

磁束コントローラーの PI 設定

```
#define PI_FLUX_KP                                (USER_DEFAULT_IQID_KP)
```

- 磁束制御ブロックの比例ゲインの設定です。ゲインの値は、次の式によって決定されます: $K_P = \omega_c L \times A$ (2.11 を参照)。

```
#define PI_FLUX_KI                                (USER_DEFAULT_IQID_KI >> 0)
```

- 磁束制御ブロックの積分ゲインの設定です。ゲインの値は、次の式によって決定されます: $K_I = RT_S K_P / L$ (2.11 を参照)。

```
#define PI_FLUX_SCALE_KPKI                        (SCALING_CURRENT_KPKI + 0)
```

- スケーリングファクター K_P と K_I の設定です。K_P と K_I は 2^PI_FLUX_SCALE_KPKI でスケールアップされます。

```
#define PI_FLUX_IK_LIMIT_MIN                      (-32768)
```

- 積分バッファの最小値を設定します。最小値は -32768 です。

```
#define PI_FLUX_IK_LIMIT_MAX                      (32767)
```

- 積分バッファの最大値を設定します。最大値は 32767 です。

```
#define PI_FLUX_UK_LIMIT_MIN                      (-32768)
```

- 磁束 PI 制御ブロックの最小出力値の設定です。最小値は -32768 です。

```
#define PI_FLUX_UK_LIMIT_MAX                      (32767)
```

- 磁束 PI 制御ブロックの最大出力値の設定です。最大値は 32767 です。

PLL Estimator コントローラーの PI 設定

```
#define PI_PLL_KP                                 (1U << 8)
```

- PLL Estimator 制御ブロックの比例ゲインの設定です。最小値は 1、最大値は 32767 です。

```
#define PI_PLL_KI                                 (1U << 6)
```

- PLL Estimator 制御ブロックの積分ゲインの設定です。最小値は 0、最大値は 32767 です。

```
#define PI_PLL_SCALE_KPKI                         (19U - USER_RES_INC)
```

- スケーリングファクター K_P と K_I の設定です。K_P と K_I は 2^PI_PLL_SCALE_KPKI でスケールアップされます。USER_RES_INC は、*pmsm_foc_feature_config.h* ファイルで定義されている定数です。PI_PLL_SCALEKPKI の最小値は 15 です。

```
#define PI_PLL_IK_LIMIT_MIN                 (-(1 << (30 - PI_PLL_SCALE_KPKI)))
```

- 積分バッファの最小値を設定します。最小値は -32768 です。

```
#define PI_PLL_IK_LIMIT_MAX                       (1 << (30 - PI_PLL_SCALE_KPKI))
```

– 積分バッファの最大値を設定します。最大値は 32767 です。

```
#define PI_PLL_UK_LIMIT_MIN                    (SPEED_LOW_LIMIT >> 4)
```

– PLL エスティメーター PI 制御ブロックの最小出力値の設定です。最小値は -32768 です。

```
#define PI_PLL_UK_LIMIT_MAX             (SPEED_HIGH_LIMIT + SPEED_LOW_LIMIT)
```

– PLL エスティメーター PI 制御ブロックの最大出力値の設定です。最大値は 32767 です。

8 PMSM FOC ソフトウェア データ構造体

8.1 FOC 制御モジュールの入力データ構造体

このデータ構造は FOC ファンクションで使用される入力変数を定義します。

Table 25 FOCInput データ構造体

カテゴリー (構造体)	変数名	データタイプ/範囲	説明	備考
FOCInput	Phase_L	符号付き 32 ビット	モーターの各相のインダクタンス	モーターパラメーターは 1 回のみ初期化
	Phase_R	符号付き 32 ビット	モーターの各相の抵抗	
	Phase_L_Scale	16 ビット / 8 – 18	インダクタンス用スケーリング 実インダクタンス SW = Phase_L/($2^{Phase_L_Scale}$)	
	CCU8_Period	16 ビット	CCU8 周期。CCU8 PWM 1 KHz～90 KHz	1 回のみ初期化
	Res_Inc	16 ビット / 0 – 8	分解能増加ビット数、SW は (16 + Res_Inc) ビットで 360° を表します	
	LPF_N_BEMF	16 ビット / 0 – 8	BEMF の LPF 係数	
	Threshold	32 ビット	FOC へ遷移する BEMF しきい値	
	Threshold_LOW	16 ビット / 0 – 512	LOW BEMF しきい値 (FOC への遷移用)	
	Threshold_HIGH	16 ビット / 0 – 512	HIGH BEMF しきい値 (FOC への遷移用)	
	Flag_State	16 ビット / 0 または 1	0: モーターは FOC で動作 1: 遷移状態	
	BEMF1	符号付き 32 ビット	センサーレス推定器の BEMF	
	BEMF2	符号付き 32 ビット	センサーレス推定器の BEMF	
	overcurrent_factor	16 ビット / 0-4096	過電流が検出されモーター速度を下げるときに使用されます	
	Ref_Speed	符号付き 32 ビット	速度 PI コントローラーのモーターリファレンス速度	
	Vq_Flag	16 ビット / 0 または 1	0: Iq PI コントローラーから Vq 入力 1: 外部から Vq 入力	
	Vq	符号付き 32 ビット / 0 – 32767	外部からの Vq 入力値、例: 電動バイクのハンドラー	
	Ref_Id	符号付き 32 ビット	PI コントローラーの Id リファレンス	デフォルト = 0
	Ref_Iq	符号付き 32 ビット	PI コントローラーの外部からの Iq リファレンス	
	Iq_PI_Flag	16 ビット / 0 または 1	0: PI コントローラーIq リファレンスは速度 PI 出力を使用 1: PI コントローラー Iq リファレンスは外部からの Ref_Iq を使用	

8.2 FOC 制御モジュールの出力データ構造体

このデータ構造は FOC ファンクションの出力変数を定義します。

Table 26 FOCOutput データ構造

カテゴリー (構造体)	変数名	データタイプ/範囲	説明
FOCOutput	Rotor_PositionQ31	符号付き 32 ビット	センサーレス推定器からのローター角度フィードバック
	Speed_By_Estimator	符号付き 32 ビット	センサーレス推定器からのローター速度フィードバック
	Previous_SVM_SectorNo	16 ビット / 0 – 5	前回の PWM サイクルにおける SVM セクター番号
	New_SVM_SectorNo	16 ビット / 0 – 5	現在の PWM サイクルにおける SVM セクター番号

8.3 FOC 制御モジュールのデータタイプ

次の表に FOC 制御ファンクションにて使用される変数のデータ構造を示します。

Table 27 FOC ファンクションにて使用されるデータの構造体

カテゴリー (構造体)	変数名	データタイプ/範囲	説明
Current	I_U	符号付き 16 ビット/ 1Q15 データフォーマット	U 相の検出電流
	I_V	符号付き 16 ビット/ 1Q15 データフォーマット	V 相の検出電流
	I_W	符号付き 16 ビット/ 1Q15 データフォーマット	W 相の検出電流
Clarke_Transform	I_Alpha_1Q31	符号付き 16 ビット/ 1Q15 データフォーマット	電流 I_Alpha、クラーク変換の出力
	I_Beta_1Q31	符号付き 16 ビット/ 1Q15 データフォーマット	電流 I_Beta、クラーク変換の出力
Park_Transform	Id	符号付き 16 ビット/ 1Q15 データフォーマット	電流 Id、パーク変換の出力
	Iq	符号付き 16 ビット/ 1Q15 データフォーマット	電流 Iq、パーク変換の出力
Car2Polar	Flux_Vd	符号付き 32 ビット	電圧 Vd、磁束 PI コントローラーの出力
	Torque_Vq	符号付き 32 ビット	電圧 Vq、トルク PI コントローラの出力
	Vref32	32 ビット	最後の PWM サイクルの SVM 振幅電圧
	Vref_AngleQ31	符号付き 32 ビット	最後の PWM サイクルの SVM 角度電圧
	SVM_Vref16	16 ビット	オープンループ制御における SVM 振幅電圧
	SVM_Angle16	16 ビット	オープンループ制御における SVM 角度

8.4 SVM モジュールデータ構造体

このデータ構造は SVM モジュールで使用される変数を定義します。

Table 28 SVM モジュールで使用されるデータ構造体

カテゴリー(構造体)	変数名	データタイプ/範囲	説明	備考
SVM	CurrentSectorNo	16 ビット /0 – 5	SVM 新セクター番号 0 から 5 はセクターA から F を表します	
	PreviousSectorNo	16 ビット /0 – 5	セクターをトラッキングする最後の PWM サイクル番号	
	Flag_3or2_ADC	16 ビット /0 または 0xBB	相電流検出に使用する ADC の数 (2 または 3) 0: USE 3 ADC 0xBB: USE 2 ADC	3 シャント電流検出手法の場合
	SVM_Flag	16 ビット /0 または 0xAD	PZV 付き SVM または標準 4 セグメント SVM のどちらを使用するかの選択 0: PZV 0xAD: 標準 4 セグメント SVM	シングルシャント電流検出手法の場合

8.5 電流検出ソフトウェア モジュールのデータの構造体

このデータ構造は相電流 ADC 結果を読み出す Get Current モジュールで使用される変数を定義します。

Table 29 Get Current モジュールで使用されるデータの構造体

カテゴリー(構造体)	変数名	データタイプ/範囲	説明	備考
ADC	ADC_Iu	16 ビット/0 – 4095	U 相電流 ADC の結果	シングルシャント電流検出手法の場合
	ADC_Iv	16 ビット/0 – 4095	V 相電流 ADC の結果	
	ADC_Iw	16 ビット/0 – 4095	W 相電流 ADC の結果	
	ADC_Bias_Iu	16 ビット/0 – 4095	ADC_Iu のバイアス	
	ADC_Bias_Iv	16 ビット/0 – 4095	ADC_Iv のバイアス	
	ADC_Bias_Iw	16 ビット/0 – 4095	ADC_Iw のバイアス	
	ADCTrig_Point	16 ビット/0 – CCU8 Slice 3 period	VADC トリガー位置; これは、CCU8 スライス 3 タイマーのコンペアマッチの値です	
	ADC_Bias	16 ビット/0 – 4095	シングルシャント電流測定時のバイアス値	
	ADC_ResultTz1	16 ビット/0 – 4095	1 回目の相電流 ADC サンプリングの ADC 値	
	ADC_ResultTz2	16 ビット/0 – 4095	2 回目の相電流 ADC サンプリングの ADC 値	

カテゴリー (構造体)	変数名	データタイプ/範囲	説明	備考
	ADC_Result3	16 ビット/0 – 4095	3 回目の相電流 ADC サンプリングの ADC 値	
	ADC_Result4	16 ビット/0 – 4095	4 回目の相電流 ADC サンプリングの ADC 値	
	ADC_Result1	符号付き 32 ビット	電流再構成のために最も重要となる 2 つの相電流の ADC 結果	
	ADC_Result2	符号付き 32 ビット		
	ADC3Trig_Point	16 ビット/0 ～ ADC4Trig_Point	VADC トリガー位置; これは、CCU8 スライス 3 タイマーのコンペアマッチの値です	
	ADC4Trig_Point	16 ビット /ADC3Trig_Point+1 ～ CCU8 Slice3 Period	VADC トリガー位置; これは、CCU8 スライス 3 タイマーのコンペアマッチの値です	
	Result_Flag	16 ビット/0 – 2	ADC の結果が最初の CCU8 スライス 3 サイクルか、2 回目の CCU8 スライス 3 サイクルか、または標準 4 セグメント SVM か を示すフラグ 0: First cycle, PZV 1: Second cycle, PZV 2: 標準 4 セグメント SVM	
	ADC_POT	0 – 4095	ポテンショメーターの ADC 結果	
	ADC_DCLink	0 – 4095	DC Link 電圧の ADC 結果	
	ADC_IDCLink	0 – 4095	平均 DC Link 電流の ADC 結果	

9 PMSM FOC ソフトウェア API 関数

ここでは、PMSM FOC ソフトウェア API 関数について説明します。API は、ユーザー関数とコントローラーAPI にグループ分けされています。

パフォーマンスを向上させ、CPU の負荷を軽減するために、ほとんどの場合、重要な API は SRAM で実行されます。SRAM で実行される API については、下表を参照してください。

Table 30 API 関数リスト

種類		API 関数名	SRAM にて実行	フラッシュにて実行
ユーザー関数		pmsm_foc_motor_start()		Yes
		pmsm_foc_motor_stop()		Yes
		pmsm_foc_motor_brake ()		Yes
		pmsm_foc_set_motor_target_speed ()		Yes
		pmsm_foc_set_motor_target_torque ()		Yes
		pmsm_foc_set_motor_target_voltage ()		Yes
		pmsm_foc_get_motor_speed ()		Yes
割込みサービスルーチン		pmsm_foc_controlloop_isr()	Yes	
		pmsm_foc_secondaryloop_isr() pmsm_foc_over_under_voltage_isr()	Yes	Yes
コントロールループ ISR	起動	pmsm_foc_bootstrap_charge()		Yes
		pmsm_foc_enable_inverter()		Yes
		pmsm_foc_disable_inverter()		Yes
		pmsm_foc_directfocrotor_pre_positioning()		Yes
		pmsm_foc_vf_foc_openloop_rampup()		Yes
		pmsm_foc_vf_smooth_transition_to_foc()		Yes
		pmsm_foc_misc_works_of_met()		Yes
	入出力処理	pmsm_foc_get_IDCLink_current()	Yes	
		pmsm_foc_linear_ramp_generator()	Yes	
		pmsm_foc_linear_torque_ramp_generator()	Yes	
		pmsm_foc_Linear_vq_ramp_generator()	Yes	
		pmsm_foc_scurve_ramp_generator()	Yes	
		pmsm_foc_svpwm_update()	Yes	
		pmsm_foc_adc34_triggersetting()	Yes	
		pmsm_foc_adctz12_triggersetting()	Yes	
		pmsm_foc_get_adcphasecurrent()	Yes	
	概要	pmsm_foc_error_handling()		Yes
		pmsm_foc_motor_hold()		Yes
		pmsm_foc_misc_works_of_foc()	Yes	
	コントローラー	pmsm_foc_speed_controller()	Yes	
		pmsm_foc_torque_controller()	Yes	

種類	API 関数名	SRAM にて実行	フラッシュにて実行
	pmsm_foc_vq_controller()	Yes	
セカンダリーループ ISR	pmsm_foc_misc_works_of_irq()	Yes	
	XMC_WDT_Service()	Yes	
	pmsm_foc_secondaryloop_callback()		Yes
低電圧/過電圧 ISR	pmsm_foc_disable_inverter()		Yes

9.1 ユーザーファンクション

ユーザー関数は、外部から呼び出される関数です。他のアプリケーションとのインターフェースを提供します。

pmsm_foc_motor_start ()

この API は、モーターを始動するときに呼び出されます。

Motor.State が TRAP_PROTECTION の場合、トラップはクリアされ、MCU は再初期化されます。

Table 31 pmsm_foc_motor_start()

入力パラメーター	-	-
戻り値	-	-
更新される変数	Motor.State	EN_INVERTER_BOOTSTRAP

pmsm_foc_motor_stop ()

この関数は、インバーターをディセーブル状態にし、すべての出力ピンをトライステートにするときに呼び出されます。モーターは制御できないフリーホイーリングの状態になります。ランプダウン制御を行なう場合は、*pmsm_foc_set_motor_target_speed/torque* を設定する必要があります。

- Motor.State が TRAP_PROTECTION の場合、トラップはクリアされ、MCU は再初期化されます。

Table 32 pmsm_foc_motor_stop()

入力パラメーター	-	-
戻り値	-	-
更新される変数	Motor.State	MOTOR_STOP

pmsm_foc_motor_brake ()

この関数は、モーターをランプダウンするときに呼び出されます。

Table 33 pmsm_foc_motor_stop()

入力パラメーター	-	-
戻り値	-	-
更新される変数	Motor.State	FOC_CLOSED_LOOP_BRAKE (ただし、ステートが TRAP_PROTECTION 以外の場合)

pmsm_foc_set_motor_target_speed ()

この関数は、ターゲット速度を設定するときに呼び出されます。スケーリングはアプリケーション固有であり、入力パラメーターは SPEED_HIGH_LIMIT と SPEED_LOW_LIMIT により制限されます。モーターを停止する場合は、pmsm_foc_motor_stop()関数を使用する必要があります。MCU は、ランプジェネレーターによりリファレンス速度をターゲット速度に近づけていきます。

この関数は、SETTING_TARGET_SPEED が SET_TARGET_SPEED に設定され、MY_FOC_CONTROL_SCHEME が SPEED_CONTROLLED_VF_MET_FOC または SPEED_CONTROLLED_DIRECT_FOC の場合にのみ使用できます。

Table 34 pmsm_foc_set_motor_target_speed ()

入力パラメーター	(uint32) motor_target_speed	-
戻り値	-	-
更新される変数	Motor.Target_Speed	-

pmsm_foc_set_motor_target_torque ()

この関数は、ターゲット速度を設定するときに呼び出されます。スケーリングはアプリケーション固有であり、入力パラメーターは USER_IQ_REF_HIGH_LIMIT と USER_IQ_REF_LOW_LIMIT によって制限されます。モーターを停止するには、pmsm_foc_motor_stop()関数を使用する必要があります。MCU は、ランプジェネレーターによりリファレンス速度をターゲット速度に近づけていきます。

この関数は、SETTING_TARGET_SPEED が SET_TARGET_SPEED に設定され、MY_FOC_CONTROL_SCHEME が TORQUE_CONTROLLED_DIRECT_FOC の場合にのみ使用できます。

Table 35 pmsm_foc_set_motor_target_torque ()

入力パラメーター	(uint32) motor_target_torque	-
戻り値	-	-
更新される変数	Motor.Target_Torque	-

pmsm_foc_set_motor_target_voltage ()

この関数は、ターゲット速度を設定するときに呼び出されます。スケーリングはアプリケーション固有であり、入力パラメーターは USER_VQ_REF_HIGH_LIMIT と USER_VQ_REF_LOW_LIMIT によって制限されます。モーターを停止するには、pmsm_foc_motor_stop() 関数を使用する必要があります。MCU は、ランプジェネレーターによりリファレンス速度をターゲット速度に近づけていきます。

この関数は、SETTING_TARGET_SPEED が SET_TARGET_SPEED に設定され、MY_FOC_CONTROL_SCHEME が VQ_CONTROLLED_DIRECT_FOC の場合にのみ使用できます。

Table 36 pmsm_foc_set_motor_target_voltage ()

入力パラメーター	(uint32) motor_target_voltage	-
戻り値	-	-
更新される変数	Motor.Target_Voltage	-

pmsm_foc_get_motor_speed ()

この関数は、`Motor.Speed` のモーター速度を RPM で返します。

Table 37 pmsm_foc_get_motor_speed ()

入力パラメーター	-	-
戻り値	速度 [RPM]	-
更新される変数	-	-

pmsm_foc_get_Vdc_link ()

この関数は、DC バスの電圧値を返します。

Table 38 pmsm_foc_get_Vdc_link ()

入力パラメーター	-	-
戻り値	電圧 [V]	-
更新される変数	-	-

9.2 コントロールループ ISR

9.2.1 起動

FOC を直接起動する場合は、次のリストの API が実行されます。

pmsm_foc_bootstrap_charge ()

この API における動作は、モーターブレーキを実行し、ゲートドライバーのブートストラップコンデンサを充電することです。同時に、モーターの各相のバイアス電流が読まれ、更新されます。

この API は、モーターステート EN_INVERTER_BOOTSTRAP にて呼び出されます。

Table 39 pmsm_foc_bootstrap_charge ()

入力パラメーター	無	-
戻り値	無	-
更新される変数	ADC.ADC_Bias_Iu	U 相 ADC バイアス値
	ADC.ADC_Bias_Iv	V 相 ADC バイアス値
	ADC.ADC_Bias_Iw	W 相 ADC バイアス値
	Motor.State	モーター始動コマンドを受信するとモーターステートを PRE_POSITIONING に更新
	Motor.Transition_Status	モーターステートの遷移フラグ モーター始動コマンドを受けとると、このフラグを MOTOR_TRANSITION に更新

pmsm_foc_enable_inverter ()

このファンクションは、インバーターピンをイネーブル状態にし、PWM GPIO を PWM 信号に設定します。

Table 40 pmsm_foc_enable_inverter ()

入力パラメーター	無	-
戻り値	無	-
更新される変数	無	-

pmsm_foc_disalbe_inverter ()

このファンクションは、インバーターピンをディセーブル状態にし、PWM GPIO をトライステートに設定します。

Table 41 pmsm_foc_disable_inverter ()

入力パラメーター	無	-
戻り値	無	-
更新される変数	無	-

pmsm_foc_directfocrotor_pre_positioning ()

この関数は、ローターの初期位置アライメントを実行するためのものです。モーターステート PRE_POSITIONING にて呼び出されます。PI コントローラー変数と FOCInput 変数を初期化し、モーターステートは FOC_CLOSED_LOOP に移ります。

Table 42 pmsm_foc_directfocrotor_pre_positioning ()

入力パラメーター	無	-
戻り値	無	-
更新される変数	Current.I_u, Current.I_v, Current.I_w	電流再構成関数が呼び出され、I_u, I_v, I_w が更新されます
	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	I_Alpha と I_Beta が更新されます
	Car2Polar.Vref	SVM の Vref 値を徐々に増やします
	Motor.State	プリポジションタイマーが切れるとモーターステートを FOC_CLOSED_LOOP に更新
	FOCInput	FOC_CLOSED_LOOP ステートに移る前に FOCInput 変数を初期化
	PI_Speed PI_Torque PI_Flux PI_PLL	FOC_CLOSED_LOOP ステートに移る前に PI コントローラー変数を初期化

pmsm_foc_vf_foc_openloop_rampup ()

この API は、モーターステート VFOPENLOOP_RAMP_UP にて呼び出されます。ローターの初期位置アライメントを実行し、その後、モーターをオープンループでランプアップさせます。モーター速度が VF_TRANSITION_SPEED で定義された速度になると、モーターステートは MET_FOC に移ります。

Table 43 pmsm_foc_vf_foc_openloop_rampup()

入力パラメーター	無	-
戻り値	無	-
更新される変数	Current.I_u, Current.I_v, Current.I_w	電流再構成関数が呼び出され、I_u, I_v, および I_w が更新されます
	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	I_alpha と I_Beta が更新されます
	Car2Polar.Vref	SVM の Vref 値を徐々に増やします
	Motor.State	モーター速度が定義された値まで上昇すると、モーターステートを MET_FOC に更新
	FOCInput	MET_FOC ステートに移る前に FOCInput 変数を初期化
	PLL_Estimator	MET_FOC ステートに移る前に PLL_Estimator API を呼び出し変数を更新

pmsm_foc_vf_smooth_transition_to_foc ()

この関数は、モーターステート MET_FOC にて呼び出されます。MET 制御用です。ステータ磁束がローター磁束に対して垂直になると、ステートを MOTOR_STABLE に戻します。

Table 44 pmsm_foc_vf_smooth_transition_to_foc ()

入力パラメーター	無	-
戻り値	Motor.Transition_Status	モーターモードステータス MOTOR_STABLE: MET が終了し、次のステートに移ることができます MOTOR_TRANSITION: MET 関数は終了していません
更新される変数	Current.I_u, Current.I_v, Current.I_w	電流再構成関数が呼び出され、I_u, I_v, および I_w が更新されます
	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	クラーク変換の関数が呼び出され、I_alpha と I_Beta が更新されます
	Car2Polar.Vref, Car2Polar.Vref_AngleQ31	Car2Polar Vref と Vref_AngleQ31 が更新されます
	FOCInput	FOCInput 変数を更新します
	PLL_Estimator	PLL_Estimator API が呼び出され、変数が更新されます

pmsm_foc_misc_works_of_met ()

このルーチンは、MET ステートが終了するのを待っている期間、モーター停止コマンドをチェックします。

MET が終了 (Motor.Mode_Flag == MOTOR_STABLE) すると、モーターステートは FOC_CLOSED_LOOP に移ります。

Table 45 pmsm_foc_misc_works_of_met()

入力パラメーター	Motor.Mode_Flag	モーターモードステータス
戻り値	無	-
更新される変数	Motor.State	Motor.Mode_Flag == MOTOR_STABLE になると、モーターステートを FOC_CLOSED_LOOP に更新
	FOCInput	FOC_CLOSED_LOOP ステートに移る前に FOCInput 変数を初期化
	PI_Speed PI_Torque PI_Flux PI_PLL	FOC_CLOSED_LOOP ステートに移る前に PI コントローラー変数を初期化
	PLL_Estimator.RotorAngleQ31	FOC_CLOSED_LOOP ステートに移る前に初回の FOC PWM サイクルのローター角度を初期化

9.2.2 入出力処理

pmsm_foc_get_IDCLink_current ()

この関数は、DC Link 値をスケーリングされた値に更新します。

Table 46 pmsm_foc_get_IDCLink_current ()

入力パラメーター	-	-
戻り値	-	-
更新される変数	ADC.ADC_IDCLink	-

pmsm_foc_get_adcphasecurrent ()

この関数は、3 シャント電流検出手法でのみ使用されます。3 相の電流の ADC 変換結果を読み出します。

同期変換を使用する場合は、VADC エイリアスチャネルの設定も SVM セクター (XMC1300/XMC1400 のみ) に従って更新されます。

Table 47 pmsm_foc_get_adcphasecurrent ()

入力パラメーター	SVM.PreviousSectorNo	前回の PWM サイクルの SVM セクター番号
	SVM.CurrentSectorNo	現在の SVM サイクルのセクター番号
戻り値	ADC.ADC_Iu	U 相電流の ADC 変換結果
	ADC.ADC_Iv	V 相電流の ADC 変換結果
	ADC.ADC_Iw	W 相電流の ADC 変換結果
更新される変数	-	

pmsm_foc_linear_ramp_generator ()

このルーチンでは、速度制御用のリニアランプジェネレーターを実装します。

Table 48 pmsm_foc_linear_ramp_generator()

入力パラメーター	set_val	ターゲット速度値
	rampup_rate	速度のランプアップレート
	rampdown_rate	速度のランプダウンレート
	speedrampstep	(rampup_rate または rampdown_rate) x PWM サイクルごとに変更されるステップ数
戻り値	Motor.Ref_speed	速度 PI コントローラーのモーターリファレンス速度
更新される変数	Motor.Ramp_Up_Rate	モーター速度ランプアップレート
	Motor.Ramp_Counter	ランプカウンター

pmsm_foc_linear_torque_ramp_generator ()

この関数では、トルク制御用のリニアランプジェネレーターを実装します。

Table 49 pmsm_foc_linear_torque_ramp_generator()

入力パラメーター	current_set	ターゲットトルク値
	inc_step	トルクランプアップレート
	dec_step	トルクランプダウンレート
戻り値	FOCInput.Ref_Iq	トルク PI コントローラーのモーターリファレンストルク
更新される変数	-	

pmsm_foc_linear_vq_ramp_generator ()

この関数では、Vq 制御用のリニアランプジェネレーターを実装します。

Table 50 pmsm_foc_linear_vq_ramp_generator()

入力パラメーター	set_val	ターゲット Vq 値
	inc_step	Vq ランプアップレート
	dec_step	Vq ランプダウンレート
戻り値	FOCInput.Vq	直交座標から極座標への変換の Vq リファレンス
更新される変数	-	

pmsm_foc_scurve_ramp_generator ()

この関数では、速度制御用の S カーブランプジェネレーターを実装します。

Table 51 pmsm_foc_scurve_ramp_generator()

入力パラメーター	set_val	ターゲット速度値
	rampup_rate	速度ランプアップレート
	rampdown_rate	速度ランプダウンレート
	speedrampstep	(rampup_rate または rampdown_rate) x PWM サイクルごとに変更されるステップ数
戻り値	Motor.Ref_speed	速度 PI コントローラーのモーターリファレンス速度
更新される変数	Motor.Ramp_Up_Rate	モーター速度ランプアップレート
	Motor.Ramp_Dn_Rate	モーター速度ランプダウンレート
	Motor.Ramp_Counter	ランプカウンター

pmsm_foc_svpwm_update ()

この関数では、SVM アルゴリズムが実行され、3 相 PWM デューティサイクルが更新されます。

Table 52 pmsm_foc_svpwm_update ()

入力パラメーター	`Amplitude`	電圧空間ベクトルの振幅
	`Angle`	電圧空間ベクトルの角度
戻り値	-	-
更新される変数	`SVM.PreviousSectorNo`	更新前 SVM セクター番号
	`SVM.CurrentSectorNo`	更新後 SVM セクター番号

pmsm_foc_adctz12_triggersetting ()

この関数は、シングルシャント電流検出手法においてのみ呼び出されます。ADC 変換の最初の 2 つのトリガーポイントを設定します。トリガーポイントの設定は、設定ファイルにて事前に計算された定数によります。

Table 53 pmsm_foc_adctz12_triggersetting()

入力パラメーター	-	-
戻り値	-	-
更新される変数	-	-

pmsm_foc_adc34_triggersetting ()

この関数は、シングルシャント電流検出手法においてのみ呼び出されます。この関数は、ADC 変換の 3 番目と 4 番目のトリガーポイントを設定します。

Table 54 pmsm_foc_adc34_triggersetting()

入力パラメーター	`ADC.ADC3Trig_Point`	3 番目のトリガーポイント設定
	`ADC.ADC4Trig_Point`	4 番目のトリガーポイント設定
戻り値	-	-
更新される変数	-	-

9.2.3 概要

pmsm_foc_misc_works_of_foc ()

この関数では、モーターの速度/トルク/Vq を上昇させ、リファレンス速度またはリファレンストルクまたは Vq をランプレートに基づいて更新します。

Table 55 pmsm_foc_misc_works_of_foc()

入力パラメーター	Motor.Mode_Flag	モーターモードステータス
戻り値	無	-
更新される変数	`Motor.State`	この変数はモーター停止コマンドを受け取った場合は、モーターステートを MOTOR_HOLD に更新します
	`Motor.Ref_Speed`	この変数は SPEED_CONTROLLED_DIRECT_FOC または SPEED_CONTROLLED_VF_MET_FOC のいずれかの制御方式が選択されている場合に更新されます
	`FOCInput.Ref_Iq`	この変数は TORQUE_CONTROLLED_DIRECT_FOC 制御方式が選択されている場合に更新されます
	`FOCInput.Vq`	この変数は VQ_CONTROLLED_DIRECT_FOC 制御方式が選択されている場合に更新されます

pmsm_foc_error_handling ()

この関数は、トラップステートからの戻りを処理します。トラップへの対応を入力できます。1 つのオプションは、一定期間後にモーター制御を再初期化することです。デフォルトでは、トラップステートからの自動復帰はありません。

Table 56 pmsm_foc_error_handling ()

入力パラメーター	無	-
戻り値	無	-
更新される変数	`Motor.State`	MOTOR_HOLD

pmsm_foc_Clear_trap ()

この関数は、ハードウェアトラップステートおよびモーターステートをクリアします。トラップ発生時、モーターストップとモーターブレーキによって呼び出されます。

Table 57 pmsm_foc_Clear_trap ()

入力パラメーター	無	-
戻り値	無	-
更新される変数	`Motor.CCU8_Trap_Status`	0x00
	`Motor.State`	MOTOR_IDLE

pmsm_foc_motor_hold ()

このファンクションは、起動とアクティブフリーホイールに使用されます。アクティブフリーホイール状態では、速度はゼロに低下しますが、PWM パターンは更新されます。

Table 58 pmsm_foc_motor_hold ()

入力パラメーター	無	-
戻り値	無	-
更新される変数	無	-

9.2.4 コントローラー

pmsm_foc_speed_controller ()

この関数は、SPEED_CONTROLLED_DIRECT_FOC または SPEED_CONTROLLED_VF_MET_FOC の制御方式が選択されている場合に呼び出されます。FOC 速度制御アルゴリズムと FOC 計算を実行します。モーターステート FOC_CLOSED_LOOP にて呼び出されます。

math コプロセッサ (XMC1302, XMC1402, および XMC1404) では、CPU 計算と CORDIC 計算をインターリーブすることで、数マイクロ秒のタイミングゲインが得られます (Figure 55 のフローチャートを参照)。

Table 59 pmsm_foc_speed_controller()

入力パラメーター	ADC.ADC_Bias_Iu	U 相電流の ADC バイアス値
	ADC.ADC_Bias_Iv	V 相電流の ADC バイアス値
	ADC.ADC_Bias_Iw	W 相電流の ADC バイアス値
	FOCInput.Ref_Speed	リファレンス速度
	SVM.CurrentSectorNo	SVM セクター番号
戻り値	Motor.Speed	PLL エスティメーターから得られる現在のモーター速度
更新される変数	Current.I_u, Current.I_v, Current.I_w	電流再構成関数が呼び出され、I_u, I_v, I_w が更新されます
	Park_Transform.Iq, Park_Transform.Id	パーク変換関数が呼び出され、Iq と Id が更新されます
	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	クラーク変換関数が呼び出され、I_alpha および I_Beta が更新されます
	Car2Polar.Vref, Car2Polar.Vref_AngleQ31	Car2Polar 関数が呼び出され、Vref と Vref_AngleQ31 が更新されます
	PI_Speed PI_Torque PI_Flux PI_PLL	PI コントローラー関数が呼び出され、PI 出力が更新されます
	PLL_Estimator	PLL Estimator API が呼び出され、変数が更新されます
	FOCOuput.Speed_by_Estimator	PLL Estimator からの推定ローター速度
	FOCOutput.Rotor_PositionQ31	PLL Estimator からの推定ローター位置

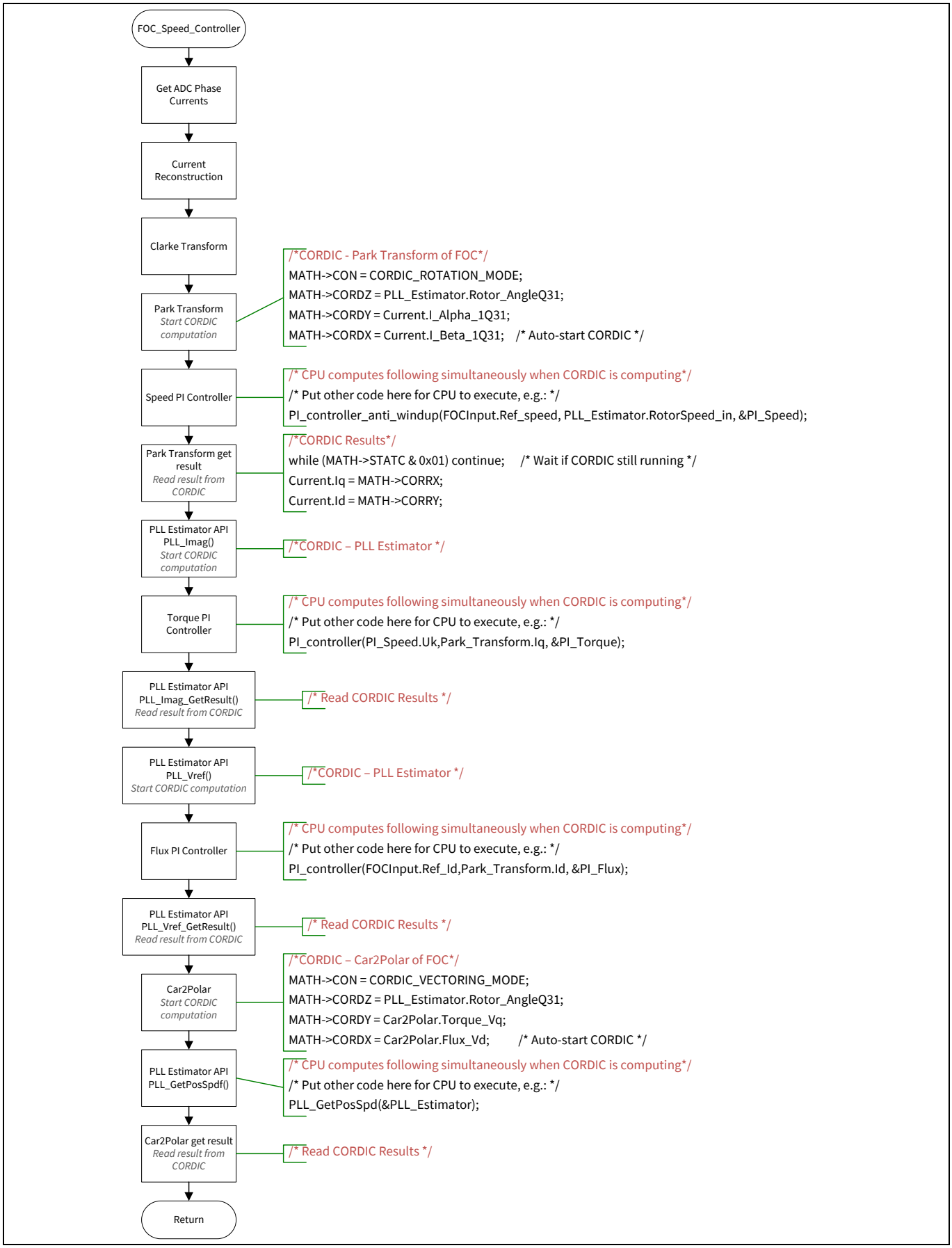


Figure 55 FOC 速度制御フローチャート

pmsm_foc_torque_controller ()

この API は、TORQUE_CONTROLLED_DIRECT_FOC 制御方式が選択されている場合に呼び出されます。FOC トルク制御とその計算を実行します。モーターステート FOC_CLOSED_LOOP にて呼び出されます。

Table 60 pmsm_foc_torque_controller ()

入力パラメーター	ADC.ADC_Bias_Iu	U 相電流の ADC バイアス値
	ADC.ADC_Bias_Iv	V 相電流の ADC バイアス値
	ADC.ADC_Bias_Iw	W 相電流の ADC バイアス値
	FOCInput.Ref.Iq	リファレンストルク値
	SVM.CurrentSectorNo	SVM セクター番号
戻り値	Motor.Speed	PLL エスティメーターから得られる現在のモーター速度
更新される変数	Current.I_u, Current.I_v, Current.I_w	電流再構成関数が呼び出され、I_u, I_v, I_w が更新されます
	Park_Transform.Iq, Park_Transform.Id	パーク変換関数が呼び出され、Iq と Id が更新されます
	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	クラーク変換関数が呼び出され、I_Alpha と I_Beta が更新されます
	Car2Polar.Vref, Car2Polar.Vref_AngleQ31	Car2Polar 関数が呼び出され、Vref と Vref_AngleQ31 が更新されます
	PI_Torque PI_Flux PI_PLL	PI コントローラー関数が呼び出され、PI 出力が更新されます
	PLL_Estimator	PLL Estimator API が呼び出され、変数が更新されます
	FOCOuput.Speed_by_Estimator	PLL エスティメーターから得られる現在のモーター速度
	FOCOutput.Rotor_PositionQ31	PLL エスティメーターから得られる現在のモーター位置

pmsm_foc_vq_controller (void)

この関数は、VQ_CONTROLLED_DIRECT_FOC 制御方式が選択されている場合に呼び出されます。FOC VQ 制御と FOC 計算を実行します。モーターステート FOC_CLOSED_LOOP にて呼び出されます。

Table 61 pmsm_foc_vq_controller(void)

入力パラメーター	ADC.ADC_Bias_Iu	U 相電流の ADC バイアス値
	ADC.ADC_Bias_Iv	V 相電流の ADC バイアス値
	ADC.ADC_Bias_Iw	W 相電流の ADC バイアス値
	FOCInput.Vq	Vq リファレンス値
	SVM.CurrentSectorNo	SVM セクター番号
戻り値	Motor.Speed	PLL エスティメーターから得られる現在のモーター速度
更新される変数	Current.I_u, Current.I_v, Current.I_w	電流再構成関数が呼び出され、I_u, I_v, I_w が更新されます
	Park_Transform.Iq, Park_Transform.Id	パーク変換関数が呼び出され、Iq と Id が更新されます
	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	クラーク変換関数が呼び出され、I_Alpha と I_Beta が更新されます
	Car2Polar.Vref, Car2Polar.Vref_AngleQ31	Car2Polar 関数が呼び出され、Vref と Vref_AngleQ31 が更新されます
	PI_Flux PI_PLL	PI コントローラー関数が呼び出され、PI 出力が更新されます
	PLL_Estimator	PLL Estimator API が呼び出され、変数が更新されます
	FOCOuput.Speed_by_Estimator	PLL エスティメーターから得られる現在のモーター速度
	FOCOutput.Rotor_PositionQ31	PLL エスティメーターから得られる現在のモーター位置

9.3 セカンダリーループ ISR

pmsm_foc_misc_works_of_irq ()

この関数は、デフォルト 1 kHz の周期でセカンダリーループ内から呼び出されます。設定したモーター速度を読み出し、スケールアップします (スケーリングについては 2.10 を参照)。

Table 62 pmsm_foc_misc_works_of_irq()

入力パラメーター	無	-
戻り値	無	-
更新される変数	Motor.Target_Speed	SETTING_TARGET_SPEED が BY_POT_ONLY または BY_UART_ONLY に設定されている場合、値は更新されます。どの値が更新されるかは MY_FOC_CONTROL_SCHEME 設定によって異なります。
	Motor.Target_Torque	
	Motor.Target_Voltag	

pmsm_foc_secondaryloop_callback ()

この関数は、PMSM_FOC_SECONDARYLOOP_CALLBACK が有効である場合にのみ使用できます。この関数の動作を定義する必要があります。セカンダリーループは、規模が大きい関数をサポートするためフラッシュ内に置かれます。実行時間を短縮するために、それらの関数を手動で RAM に配置できます。

Table 63 pmsm_foc_misc_works_of_irq()

入力パラメーター	無	-
戻り値	無	-
更新される変数	ユーザー定義	-

9.4 FPU ライブラリ

フィールド指向制御を行うために必要なすべての数学関数は、FPU *library fpu_math2* に実装されています。特に、サイン, コサイン, アークタンジェント, マグニチュード, およびパーク変換が実装されています。

9.4.1 ライブラリ理論

サイン, コサイン, およびアークタンジェントを実装するための良い選択肢は、ルックアップテーブル (LUT) を使用することです。サイン, コサイン, およびパーク変換関数は、相対 LUT を使用して CMSIS ライブラリからインポートされます (サポートについては、CMSIS サポートマニュアルを参照してください)。アークタンジェントの場合、そのドメインは無限で周期的ではないため、ルックアップテーブルのみを使用することは不可能です。そのため、ライブラリでは LUT と数学近似を組み合わせて使用します。アークタンジェント LUT では、0 から特定の値 (ライブラリでは 5.02 に設定) の領域で 300 個の可変ステップサンプルが使用されます。

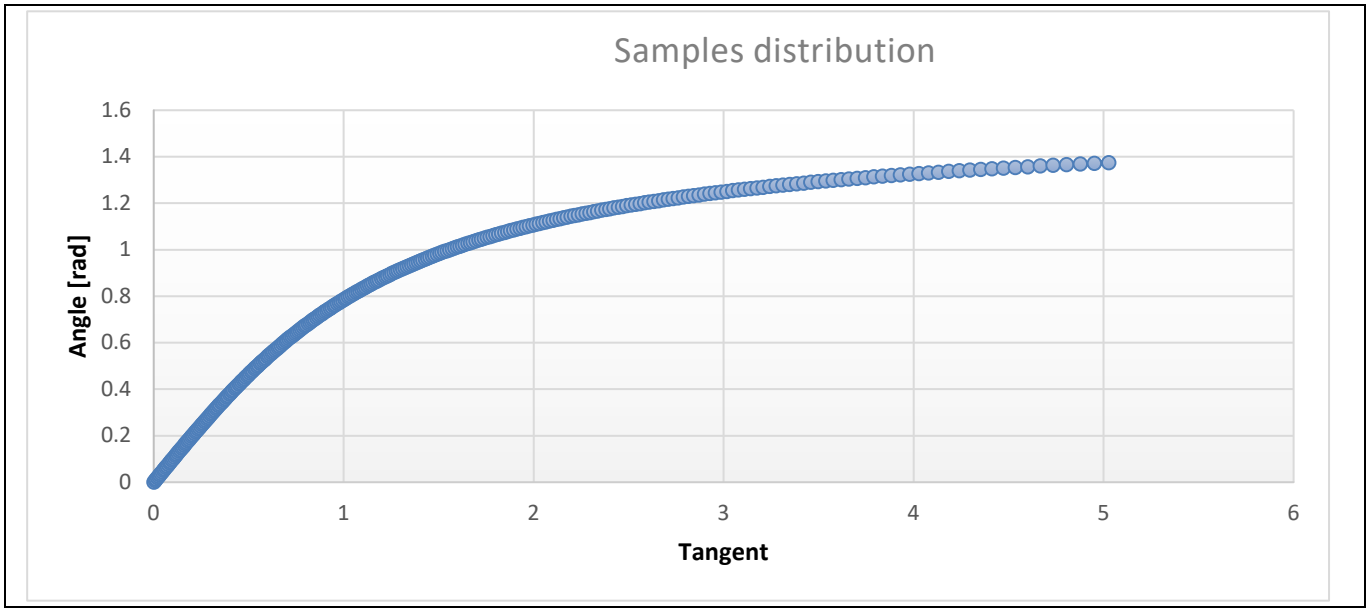


Figure 56 サンプル分布

タンジェントの値をサンプルベクトルインデックスに変換する数学的マッピングには、アークタンジェント関数の 2 次導関数と関係があり、適切にスケーリングされて適用されます。

$$y = \frac{\left(\frac{x}{AT_X_SCALING} + AT_X_OFFSET\right)^2}{1 + \left(\frac{x}{AT_X_SCALING} + AT_X_OFFSET\right)^2} \cdot AT_Y_SCALING - AT_Y_OFFSET$$

Equation 29

ここで、x はタンジェントの値、y はインデックスベクトルです。パラメーターの最適な値は、ライブラリで定義されます。float インデックス値が見つかると、角度の値を見つけるための線形補間に使用されます。変数の間隔は、関数がより湾曲している点に関数をマッピングできるように使用されます。アークタンジェントの凹面のため、線形補間は常に小さめに評価された値を返します。この現象を補正するために、すべてのサンプルに定数が追加されます (AT_CC_ERROR)。

最後の LUT 要素よりも大きいタンジェントの場合、ライブラリは次の数学近似を使用します。

$$angle = -\frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} + \frac{\pi}{2}$$

Equation 30

ここで、x はタンジェントで、結果はラジアンで表されます。

関数アークタンジェントの精度は約 2E-6 ラジアン (0.00012 度、Q31 フォーマット で 20 ビット) です。ライブラリのサポートは、精度を 5E-7 ラジアン (0.00003 度、Q31 で 22 ビット) にするエラー訂正でもあります。`fpu_cart2polar()`関数を使用する前に `fpu_tangent_lookup_table_generation()`関数を呼び出すことが必須です。

9.4.2 ライブラリ API

fpu_sin_q31 ()

この関数は、入力された角度のサインを計算します。入力、戻り値とも Q31 フォーマットです。

Table 64 fpu_sin_q31 ()

入力パラメーター	Angle	Q31 フォーマット
戻り値	Sin (Angle)	Q31 フォーマット
更新される変数	無	-

fpu_cos_q31 ()

この関数は、入力された角度のコサインを計算します。入力、戻り値とも Q31 フォーマットです。

Table 65 fpu_cos_q31 ()

入力パラメーター	Angle	Q31 フォーマット
戻り値	Cos (Angle)	Q31 フォーマット
更新される変数	無	-

fpu_park_q31 ()

この関数は、入力された I_alpha, I_beta, およびローター角度のサイン/コサインのパーク変換を計算します。

Table 66 fpu_park_q31 ()

入力パラメーター	`Clarke_Transform.I_Alpha`, `Clarke_Transform.I_Beta`	任意の物理量の Y 成分と X 成分。この場合は、クラーク変換電流
	`Angle_sine` `Angle_cosine`	回転角度のサインとコサイン。この場合はローター角度
戻り値	`Park_Transform.Iq`, `Park_Transform.Id`	Y および X の回転成分。この場合は Id と Iq
更新される変数	無	-

fpu_cart2polar ()

この関数は、任意の物理量の直交座標から極座標を計算し、極座標にオフセット角度を追加します。使用する前に、`fpu_tangent_lookup_table_generation()`関数を呼び出す必要があります。

Table 67 fpu_cart2polar ()

入力パラメーター	`Car2Polar.Flux_Vd` `Car2Polar.Torque_Vq`	任意の物理量の Y 成分と X 成分。この場合は磁束成分とトルク成分
	`Rotor Angle`	極座標に追加する角度。この場合は、ローター角度
戻り値	`Car2Polar.Vref32` `Car2Polar.Vref_AngleQ31`	物理量の極座標成分
更新される変数	無	-

fpu_tangent_lookup_table_generation()

この関数は、アークタンジェントのルックアップテーブルのサンプルを計算します。

Table 68 fpu_tangent_lookup_table_generation()

入力パラメーター	無	-
戻り値	無	-
更新される変数	Arctangent lookup table	-

アークタンジェントの値

```
#define     AT_X_SCALING          2
#define     AT_X_OFFSET           0.675f
#define     AT_Y_SCALING          500.467269f
#define     AT_Y_OFFSET           156.6511975f
#define     AT_ERROR_CORRECTION   1
```

— エラー訂正を有効にする

```
#define     AT_LAST_LUT_TANGENT   5.02719008f
```

— LUT テーブルの最後の要素

```
#define     FPU_PI                3.1415926535f
#define     FPU_HALF_PI           1.57079632679f
#define     AT_CC_ERROR           0.000000261111f
```

— テーブル値のオフセット

10 リソース

- インフィニオンの XMC1000 モーター制御アプリケーションキットのドキュメント: XMC1000 Motor Control Application Kit。
- XMC サンプルコードはインフィニオンの GitHub ページにあります。
- AP32370 - XMC1000/XMC4000 - PMSM FOC motor control software using XMC™。
- インフィニオンの XMC1000/XMC4000 モーター制御パワーカードのドキュメント: 3-phase DC Motor Control Power Card。
- M5 コネクタ付き MADKS の XMC4800 制御ボード: EVAL-XMC4800PSOC6M5。

改訂履歴

版数	発行日	変更内容
V1.6	2025-02-12	本版は英語版 AP32370 PMSM FOC motor control software using XMC™ V 1.6 について、インフィニオン DEVELOPER COMMUNITY の参画者によって日本語に翻訳されたドキュメントです。
V1.7	2025-03-03	これは英語版 AP32370 PMSM FOC motor control using XMC™ MCUs V 1.7 を翻訳した日本語版 AP32370_JA V1.7 です。

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

Edition 2025-03-03
Published by
Infineon Technologies AG
81726 Munich, Germany

© 2025 Infineon Technologies AG. All Rights Reserved.

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

Document reference
AP32370_JA

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

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

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