

RZ/T1 グループ

ソリューションキットファームウェア アプリケーションノート

RZファミリ RZ/T1シリーズ

本資料に記載の全ての情報は本資料発行時点のものであり、ルネサス エレクトロニクスは、予告なしに、本資料に記載した製品または仕様を変更することがあります。
ルネサス エレクトロニクスのホームページなどにより公開される最新情報をご確認ください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
 2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
 3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
 4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
 5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、家電、工作機械、パーソナル機器、産業用ロボット等
高品質水準： 輸送機器（自動車、電車、船舶等）、交通管制（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
当社製品は、データシート等により高信頼性、Harsh environment向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
 6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
 7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
 8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関し、当社は、一切その責任を負いません。
 9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
 10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものいたします。
 11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
 12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。
- 注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。
- 注2. 本資料において使用されている「当社製品」とは、注1において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1 2017.11)

本社所在地

〒135-0061 東京都江東区豊洲3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策

CMOS製品の取り扱いの際は静電気防止を心がけてください。CMOS製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS製品を実装したボードについても同様の扱いをしてください。

2. 電源投入時の処置

電源投入時は、製品の状態は不定です。電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

3. 電源オフ時における入力信号

当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。

4. 未使用端子の処理

未使用端子は、「未使用端子の処理」に従って処理してください。CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。

5. クロックについて

リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後、切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

6. 入力端子の印加波形

入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS製品の入力がノイズなどに起因して、 $V_{IL}(\text{Max.})$ から $V_{IH}(\text{Min.})$ までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、 $V_{IL}(\text{Max.})$ から $V_{IH}(\text{Min.})$ までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。

7. リザーブアドレス（予約領域）のアクセス禁止

リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

8. 製品間の相違について

型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違っていると、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

Arm®およびCortex®は、Arm Limited（またはその子会社）のEUまたはその他の国における登録商標です。
All rights reserved.

EtherCAT®は、ドイツBeckhoff Automation GmbHによりライセンスされた特許取得済み技術であり登録商標です。

TwinCAT®は、Beckhoff Automation GmbHによりライセンスされた登録商標です。

IEEEは、the Institute of Electrical and Electronics Engineers, Inc.の登録商標です。

その他、本資料中の製品名やサービス名は全てそれぞれの所有者に属する商標または登録商標です。

このマニュアルの使い方

1. 目的と対象者

このマニュアルは、RZ/T1 ソリューションキットファームウェアの機能および使用方法をユーザに理解していただくためのマニュアルです。本ファームウェアを使用して、RZ/T1 ソリューションキットの応用例に基づいてモーション制御システムのアプリケーションを設計するユーザを対象にしています。このマニュアルを使用するには、C 言語およびアプリケーションプロジェクトのライブラリの使用に関する基本的な知識が必要です。

本マイコンは、注意事項を十分確認の上、使用してください。注意事項は、各章の本文中、各章の最後、注意事項の章に記載しています。

改訂記録は旧版の記載内容に対して訂正または追加した主な箇所をまとめたものです。改訂内容すべてを記録したものではありません。詳細は、このマニュアルの本文でご確認ください。

2. 略語および略称の説明

略語 / 略称	英語名	日本語名
ACIA	Asynchronous Communications Interface Adapter	調歩同期式通信アダプタ
bps	bits per second	転送速度を表す単位、ビット/秒
CRC	Cyclic Redundancy Check	巡回冗長検査
DMA	Direct Memory Access	CPU の命令を介さずに直接データ転送を行う方式
DMAC	Direct Memory Access Controller	DMA を行うコントローラ
GSM	Global System for Mobile Communications	FDD-TDMA の第二世代携帯電話の方式
Hi-Z	High Impedance	回路が電氣的に接続されていない状態
IEBus	Inter Equipment bus	-
I/O	Input/Output	入出力
IrDA	Infrared Data Association	赤外線通信の業界団体または規格
LSB	Least Significant Bit	最下位ビット
MSB	Most Significant Bit	最上位ビット
NC	Non-Connection	未接続
PLL	Phase Locked Loop	位相同期回路
PWM	Pulse Width Modulation	パルス幅変調
SFR	Special Function Registers	周辺機能を制御するためのレジスタ
SIM	Subscriber Identity Module	ISO/IEC 7816 規定の接触型 IC カード
UART	Universal Asynchronous Receiver/Transmitter	調歩同期式シリアルインタフェース
VCO	Voltage Controlled Oscillator	電圧制御発振器

すべての商標および登録商標は、それぞれの所有者に帰属します。

目次

1. はじめに	8
1.1 概要	8
1.2 機能	8
1.3 ファームウェアの構成	9
2. 動作環境	10
3. ファイル構成	11
4. ファームウェアの構造	15
4.1 概要	15
4.1.1 スタートアップ関数	18
4.1.2 非リアルタイム関数	19
4.1.3 周期リアルタイム関数	20
4.1.4 通信関数	21
4.2 データ型	22
4.3 データ構造 / 変数	23
4.4 列挙型 / マクロ / 定数	23
5. 初期化関数とスタートアップ関数	24
5.1 ブートローダ	24
5.2 周辺機能の初期化	24
5.3 ファームウェアの初期化	25
6. サーボ制御	26
6.1 モータ位置 — エンコーダインタフェース	26
6.2 モータ制御 — トルク生成	28
6.3 位置制御 — PIDレギュレータ	31
6.4 モーションプランニング — 速度プロファイルジェネレータ	32
6.5 モーション制御パラメータ	36
6.5.1 目標位置	36
6.5.2 最大速度	36
6.5.3 最大加速度 / 減速度	36
6.5.4 最大加速度 / 減速度変化率	36
6.5.5 動作開始モード	37
6.5.6 動作停止モード	37
7. システム制御関数	38
7.1 インターロック	38
7.2 電子ギア	40
7.3 デジタル入出力	41
7.4 データ記録	42
7.5 モータのフェーシング	44
7.6 モータのホーミング	45
8. ホスト通信	48
8.1 ASCII通信プロトコル	48
8.2 バイナリパケット通信プロトコル	49
8.2.1 通信バス	49
8.2.2 モジュールのアドレス指定	50

8.2.3	要求パケットの形式	50
8.3	EtherCATサポート	52
8.3.1	ソフトウェア構成	52
8.3.2	CiA402ドライブプロファイル	53
8.3.2.1	動作モード	54
8.3.2.2	状態遷移	55
8.3.2.3	オブジェクトディクショナリ	56
8.3.3	アプリケーションの開発手順	58
8.3.3.1	SSCツールによるサンプルプロトコルスタック	58
8.3.3.2	モータ制御用プログラムの実装	60
8.3.3.3	プロトコルスタック実装時の注意事項	66
8.3.4	共有メモリ	67
8.3.4.1	動作概要	67
8.3.4.2	ソフトウェア動作	67
8.3.4.3	ソフトウェアブロック図	69
8.3.4.4	ソフトウェア状態遷移	69
8.3.4.5	使用メモリサイズ	70
8.3.4.6	ファイル構成	71
8.3.4.7	ソフトウェアリソース	71
8.3.4.8	プリミティブデータ型	72
8.3.4.9	データ構造の定義	72
8.3.4.10	グローバル変数	72
8.3.4.11	値（マクロ）の定義	73
8.3.4.12	値（列挙型）の定義	73
8.3.4.13	エラーコード	73
8.3.4.14	関数	74
8.3.4.15	共有メモリドライバ定義の構造体	74
8.3.4.16	関数リファレンス	75
8.3.4.17	フローチャート	79
8.3.4.18	使用例	81
9.	リソース	82
9.1	ハードウェア	82
9.2	OS	82
9.3	メモリ	82
10.	関連ドキュメント	83
付録 A.	ASCII通信プロトコルコマンド	84
付録 B.	パケット通信プロトコルコマンド	90
B.1	Get Module Data（パケットコード：0）	91
B.2	Status Report Request（パケットコード：1）	92
B.3	Function Call（パケットコード：2）	94
B.4	Initialize PVT Stream（パケットコード：3）	94
B.5	Stream PVT Data（パケットコード：4）	94
B.6	Set Parameter（パケットコード：5）	95
B.6-1	16ビットパラメータ設定	95
B.6-2	32ビットパラメータ設定	95
B.7	Get Parameter（パケットコード：6）	96
B.8	Report Trace Buffers Content（パケットコード：7）	97
B.9	Set Communication Baud Rate（パケットコード：15）	97

1. はじめに

1.1 概要

RZ/T1ソリューションキットファームウェアは、デュアルチャネルの産業用サーボコントローラの全機能を実装した組み込みアプリケーションです。リファレンスコードが、高性能なCPUコアやフレキシブルなアプソリュートエンコーダI/F、多様な接続オプションなど、ルネサス製RZ/T1デバイスのモーション制御性能を実証します。

1.2 機能

本ファームウェアは主な機能として、以下を備えています。

- RZ/T1のコアおよび周辺機能の初期化

実行可能なコードをQSPIフラッシュメモリからデバイスRAMに転送します。各種アプソリュートエンコーダに対応するエンコーダI/Fの初期化を行います。ADCとタイマをデュアルチャネルのインバータに接続するよう設定します。PWMタイマ割り込みハンドラをリアルタイム制御関数を呼び出すよう設定します。SCI割り込みハンドラをホストコマンドに応答するよう設定します。GPIO端子を、ソリューションキットのBiplaneボードの各種デジタル入出力に対応するよう初期化します。

- ホストコマンドの処理

本ファームウェアは、ASCIIコマンドプロトコルとバイナリパケットプロトコルの2つの通信プロトコルに対応します。コマンドインタプリタは、コマンドタイプを検出し、適切なディスパッチャを起動します。ファームウェアは、100種類以上のコマンドを認識し、あらゆる制御パラメータとアルゴリズムへのアクセスが可能です。ホストは、各モータのステータスを追跡し、動作要求の実行を制御するために情報を定期的に取得することができます。また、ホストは、デバイスのバッファに格納された各種変数のサンプルデータを取得し、分析用のデータに再構成することができます。

- 制御ループアルゴリズムのリアルタイム処理

制御アルゴリズムは、PWMタイミング(50us)を生成するタイマのコンテキストから呼び出されません。これにより、制御関数の確実性とリアルタイム性が確保されます。リアルタイムタスクには、現在位置の取得、位置制御ループの実行、電流制御ループ(磁界方向制御)の実行、次のPWM周期のデューティ比の生成、診断のためのデータ収集などがあります。

1.3 ファームウェアの構成

ソリューションキットファームウェアは、RZ/T1ソリューションキットの一部です。本ファームウェアは、コアのモーション制御機能と2つの追加ライブラリが提供する機能で構成されています。ECLライブラリは、各種エンコーダプロトコルの初期化および接続に使用します。速度プロファイル生成（VPG）ライブラリは、PTP（Point to Point）動作に必要な一連の設定されたポイントを生成するために使用します。

以下に、ファームウェアコンポーネントにおけるソリューションキットファームウェアの配置を示します。

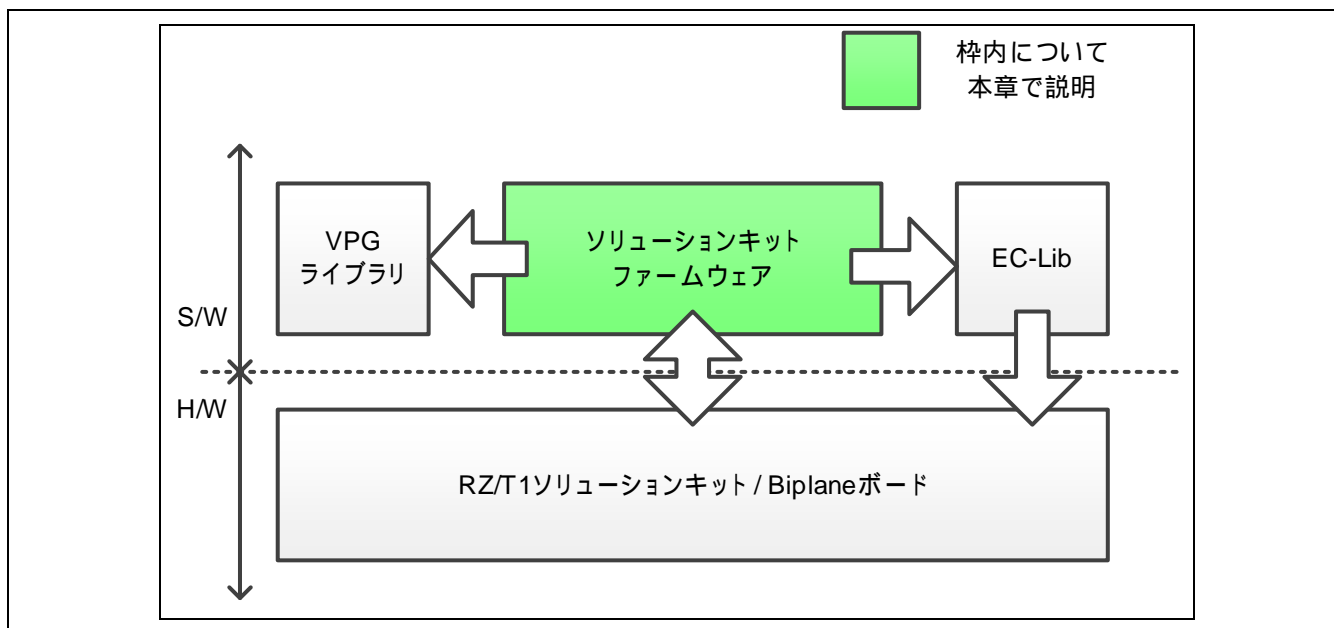


図1.1 ソリューションキットファームウェアの配置

2. 動作環境

本マニュアルに記載のソフトウェアは、下記の環境で動作することを想定しています。

表 2.1 動作環境

項目	説明
使用マイコン	RZ/T1 (Cortex®-R4)
動作周波数	600/450 MHz
動作電圧	供給電圧（入出力）：3.3V 供給電圧（内部）：1.2V
総合開発環境	ルネサスエレクトロニクス製 e ² studio version 6.3.0 IAR システムズ製 Embedded Workbench® for Arm version 8.22.2

3. ファイル構成

以下に、ソリューションキットファームウェアのファイル構成を示します。

表 3.1 ファイル構成 (Cortex-R4 用)

ファイル	内容
inclapl\m_common.h	ソリューションキットファームウェアのヘッダファイル。 モータデータ構造体およびすべてのグローバル関数のシグネチャを含む。
incl\common\r_typedefs.h	共通タイプ定義ヘッダファイル
incl\common\platform.h	共通タイプ定義ヘッダファイル
incl\common\iodefine.h	RZ/T1 デバイスのレジスタ設定
incl\scifa_uart	シリアル通信用ヘッダファイル群
incl\serial_flash	SPI フラッシュアクセス用ヘッダファイル群
incl\shm	共有メモリアクセス用ヘッダファイル群
src\drv\m_biplane.h	ソリューションキットのハードウェアと RZ/T1 デバイス用マクロ定義
src\drv\m_rzt.c	ソリューションキットのハードウェアと RZ/T1 デバイス用コード
src\drv\m_inputs.c	ソリューションキットのデジタル入力関数用コード
src\drv\scifa_uart	シリアル通信用コード
src\drv\serial_flash	SPI フラッシュアクセス用コード
src\drv\shm\r_shm.c	共有メモリアクセス用コード
src\apl\m_commands.c	呼び出し可能な全ホストコマンド用コード
src\apl\m_commutation.c	空間ベクトル変調やホールベースの台形整流などのモータ整流アルゴリズム用コード
src\apl\m_control.c	各状態と設定オプションに応じて実行分岐を制御するリアルタイムアルゴリズム
src\apl\m_homing.c	ホーミングアルゴリズムを実装するステートマシン
src\apl\m_interlocks.c	各種インターロック条件を確認する関数
src\apl\m_interpreter.c	ASCII コマンドのコマンドパーサーおよびバイナリパケットのコマンドデコーダ
src\apl\m_phasing.c	各種位相アルゴリズムを実装する関数
src\apl\m_pid_calc.c	位置制御ループアルゴリズムを実装する関数
src\apl\m_pos_read.c	エンコーダ位置読み出し制御アルゴリズム
src\apl\m_recorder.c	データ収集機能および開始・停止トリガの判定
src\apl\m_vpg_trap.c	台形速度プロファイルジェネレータ
src\encoder\BiSS	BiSS-C アブソリュートエンコーダ I/F 接続用ファイル群

ファイル	内容
src\encoder\EnDat	EnDat 2.2 アブソリュートエンコーダ I/F 接続用ファイル群
src\encoder\AFormat	A-format ^{™*1} アブソリュートエンコーダ I/F 接続用ファイル群
src\encoder\FACoder	FA-CODER アブソリュートエンコーダ I/F 接続用ファイル群
src\encoder\HIPERFACE_DSL	HIPERFACE DSL アブソリュートエンコーダ I/F 接続用ファイル群
src\encoder\r_ecl_rzt1_if.h	エンコーダ I/F ライブラリのマクロおよび関数定義
src\sharedmemory\shm_access.c	EtherCAT 通信経由のモータ制御時の共有メモリアクセス用コード
src\sharedmemory\shm_motor.c	EtherCAT 通信経由のモータ制御用コード
dat\r_as_rzt1.dat	A-format アブソリュートエンコーダの通信プロトコル設定ファイル
dat\r_biss_rzt1.dat	BiSS アブソリュートエンコーダの通信プロトコル設定ファイル
dat\r_endat_rzt1.dat	EnDat アブソリュートエンコーダの通信プロトコル設定ファイル
dat\r_fac_rzt1.dat	FA-CODER アブソリュートエンコーダの通信プロトコル設定ファイル
dat\r_hfdsl_rzt1.dat	HIPERFACE DSL アブソリュートエンコーダの通信プロトコル設定ファイル
lib\gcc\libVPG.a	gcc 用速度プロファイル生成ライブラリ
lib\gcc\r_ecl_rzt1.a	gcc 用エンコーダ I/F 初期化ライブラリ。 選択されているアブソリュートエンコーダの I/F プロトコル仕様に 応じた設定の読み込みに使用します。
lib\iar\r_vpg.a	IAR 用速度プロファイル生成ライブラリ
lib\iar\r_ecl_rzt1.a	IAR 用エンコーダ I/F 初期化ライブラリ。 選択されているアブソリュートエンコーダの I/F プロトコル 仕様に応じた設定の読み込みに使用します。
prj\e2studio\src\cg_src	このディレクトリの全ファイルは gcc 用 AP4 コード生成ツールで自動的に生成されます。 ファームウェアの動作に関わる RZ/T1 の各周辺機能の初期化に使用します。
prj\e2studio\settings\CodeGenerator\cgproject.cgp	gcc 用 AP4 プロジェクトファイル群
prj\iar\cg_src	このディレクトリの全ファイルは IAR 用 AP4 コード生成ツールで自動的に生成されます。 ファームウェアの動作に関わる RZ/T1 の各周辺機能の初期化に使用します。
prj\iar\Biplane.cgp	IAR 用 AP4 プロジェクトファイル群

ファイル	内容
<code>prjle2studio\src</code>	このフォルダのファイルは、ブート処理および SPI フラッシュから gcc 用デバイスメモリへのコード転送を行います。
<code>prj\iar\startup\spi</code>	このフォルダのファイルは、ブート処理および SPI フラッシュから IAR 用デバイスメモリへのコード転送を行います。

注 1. A-format は Nikon 社の商標です。

表 3.2 ファイル構成 (Cortex-M3 用)

ファイル	内容
cm3_boot_binary\cm3_GCC.bin	Cortex-M3 プロジェクトから出力される gcc 用バイナリファイル
cm3_boot_binary\cm3_IAR.bin	Cortex-M3 プロジェクトから出力される IAR 用バイナリファイル
CMSIS\Include	CMSIS Cortex-M のヘッダファイル群
Device\Renesas\RIN_Engine\Include\	
scifa_uart\r_scifa_uart.h	シリアル通信用ヘッダファイル
board.h	ソリューションキットのハードウェアと RZ/T1 デバイス用ヘッダファイル
interrupt_handlers.h	ベクタテーブルおよび割り込みハンドラ用ヘッダファイル
iodefine.h	RZ/T1 デバイスのレジスタ設定
itron.h	ITRON の一般定義
kernel.h	カーネル仕様書
RIN_Engine.h	R-IN エンジン用ヘッダファイル
system_RIN_Engine.h	R-IN エンジン用ヘッダファイル
typedefine.h	整数型のエイリアス
Device\Renesas\RIN_Engine\Library\GCC\libos.a	gcc 用 HW-RTOS ライブラリ
Device\Renesas\RIN_Engine\Library\IAR\libos.a	IAR 用 HW-RTOS ライブラリ
Device\Renesas\RIN_Engine\Source\	
Board\rskrzt1\board_RenesasBiplane.c	ソリューションキットのハードウェアと RZ/T1 デバイス用コード
Driver\scifa_uart	シリアル通信用コード
Device\Renesas\RIN_Engine\Source\Project_Dual\CiA402_Motion_Control\RenesasSDK	
ESI_File\Renesas_RZT1_DRIVE_IT!_CiA402.xml	EtherCAT スレーブ情報ファイル
shm	共有メモリアクセス用ヘッダファイル群およびコード
Src	SSC ツールで生成するスレーブスタックコード
ssc_project\RZT1_CiA402_Motion_Control.esp	SSC ツールを実行するためのプロジェクトファイル
apply_patch.bat	スレーブスタックコードにパッチファイルを適用するためのパッチファイル
RZT1_CiA402_Motion_Control_yymmdd.patch	スレーブスタックコードに適用するパッチファイル
Device\Renesas\RIN_Engine\Source\Templates	Cortex-M3 起動ファイル群

4. ファームウェアの構造

4.1 概要

RZ/T1ソリューションキットファームウェアは、それぞれが特定の目的を持つモジュールの集合体で、引数として渡された構造体データの内容に基づいて動作します。グローバル関数はできる限り使用しないようにしてください。関数の呼び出しは厳格に定義された優先順位とコンテキストにおいて行われます。これは、制御アルゴリズムの確実な動作を保証することを目的としています。

ファームウェアの動作は、確実な動作を行うため、複数の領域に分割されます。これらの領域を以下に示します。

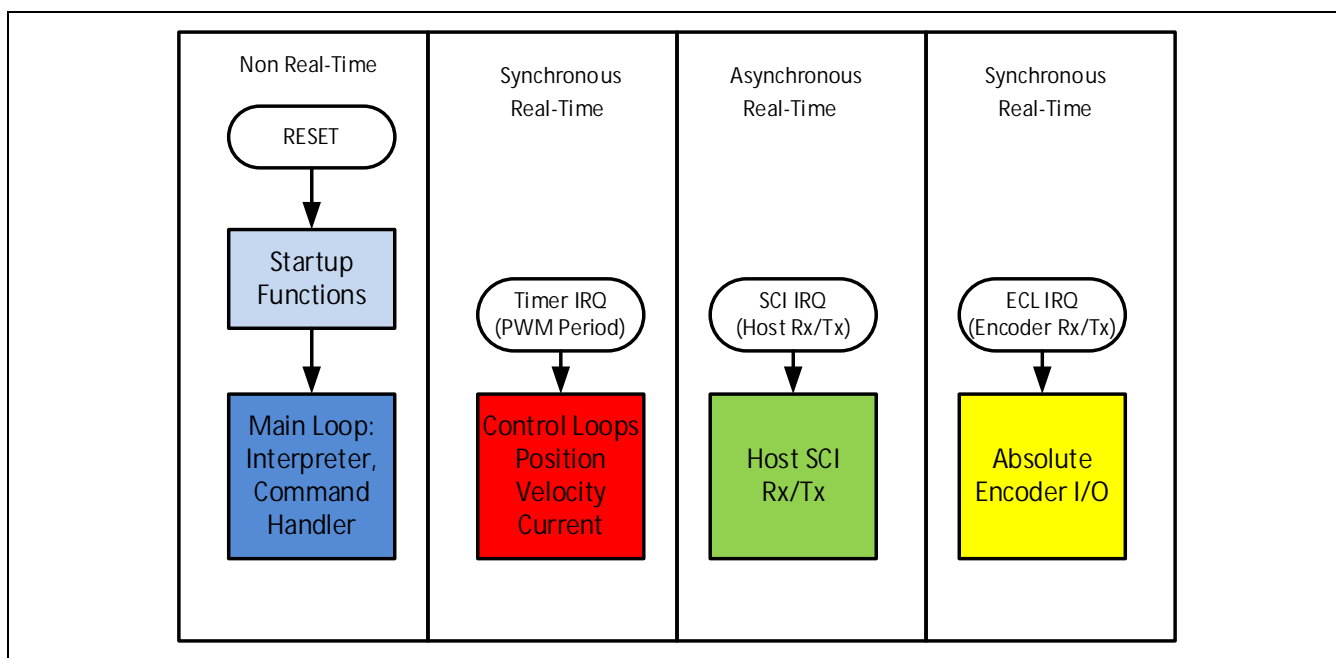


図4.1 ファームウェアコード実行領域

上記関数のランタイム時の動作を、関数の実行フローを表す以下のタイミング図に示します。RZ/T1 デバイスでは、CPUの介入を最小限にしてSCI通信を行うため、アブソリュートエンコーダおよびFIFOバッファに接続する専用ハードウェアブロックを採用しています。そのため、各通信タスクは他のタスクと重複して実行されます。

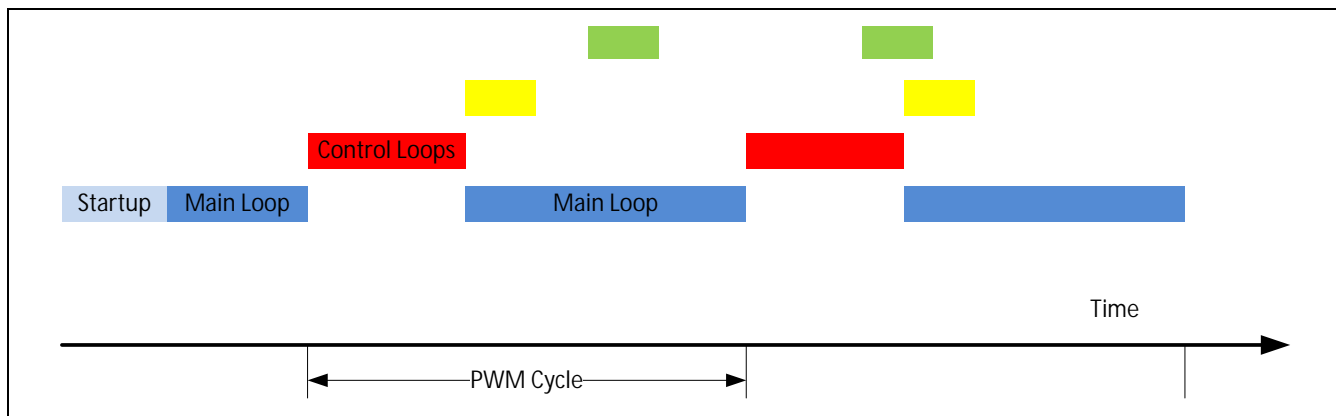


図4.2 ファームウェアタスクのスケジューリング (概略)

関数ブロック間の調整は、各モータと各通信インターフェースの情報をカプセル化する共有データ構造を用いて行われます。以下の図では、関数ブロック（長方形）と共有データ構造体（楕円形）間のデータフローを矢印で示します。

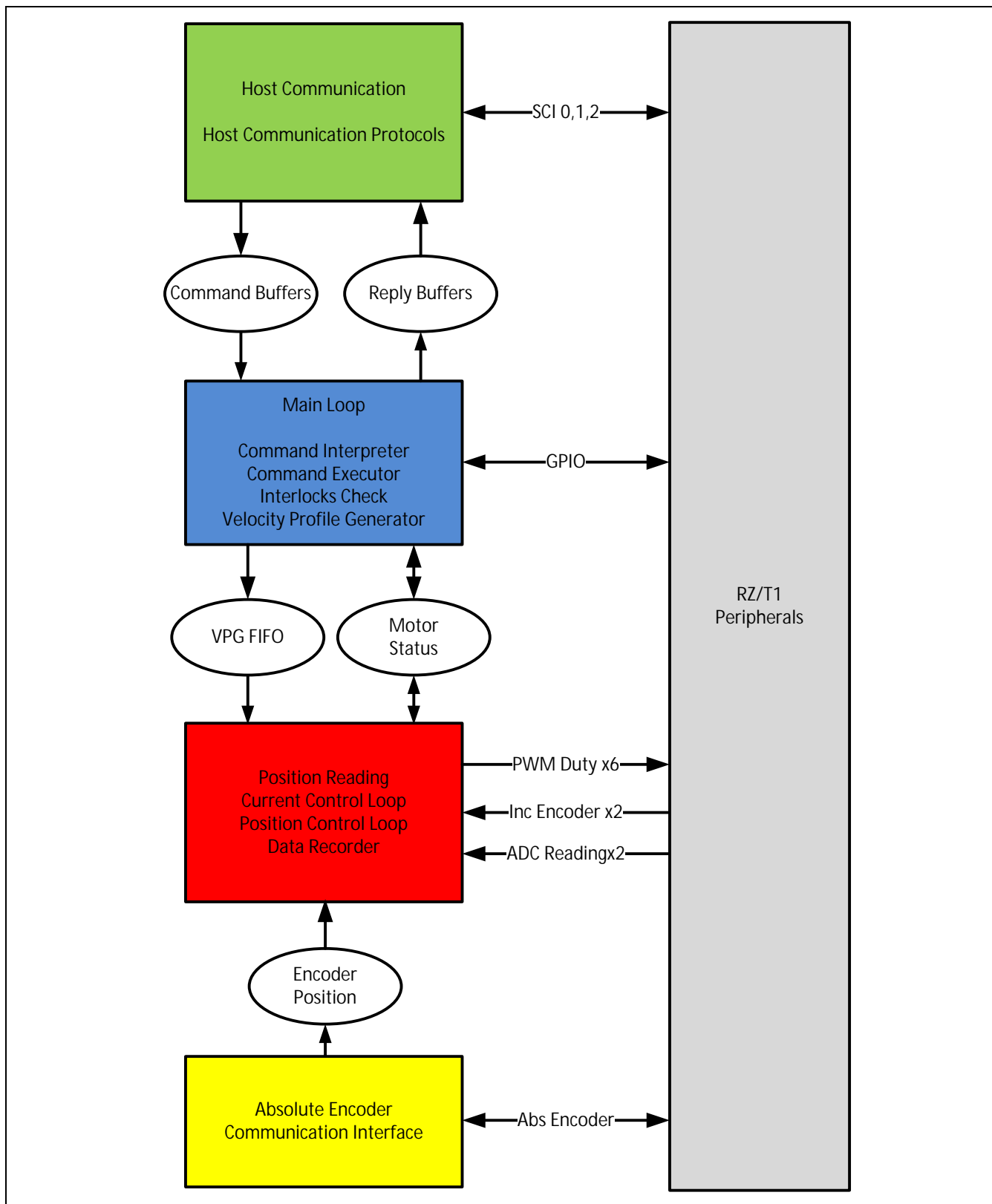


図4.3 データフロー

4.1.1 スタートアップ関数

スタートアップ関数はデバイスのリセット時に実行されます。本関数は1回のみ実行され、タイミング制約に依存しません。RZ/T1 デバイス初期化の主なフェーズと関連する関数を以下に示します。

スタートアップ動作	内容
ブートローダ	デバイス初期化の第1フェーズを実行します。実行可能なファームウェアをSPIフラッシュからデバイスメモリへ転送するために必要なすべてのクロックおよび周辺機能の設定が必要です。この処理は <code>prj\iar\startup\spi</code> フォルダ内のファイルに定義されている関数によって実行されます。
周辺機能の初期化	デバイス初期化の第2フェーズです。モーションコントロールファームウェアに必要な周辺機能を設定する関数を呼び出します。初期化コードはAP4 ツールで自動生成されます。生成されたファイルは <code>src\crg_src</code> フォルダに格納されます。
データ構造の初期化 <code>m_startup()</code>	ファームウェアの初期化には、静的データ構造体へのデフォルト値の設定、制御アルゴリズムで使用するハードウェアレジスタへのポインタの初期化、GPIOのデフォルト状態の設定などが含まれます。 初期化フェーズは、選択されたエンコーダタイプに応じて、プログラム可能なエンコーダプロトコルの設定を読み込むことで完了します。 この初期化フェーズは <code>src\m_rzt.c</code> ファイルの <code>m_startup()</code> 関数で実行されます。
永続パラメータの復元 <code>m_Restore()</code>	本ファームウェアでは、アプリケーション固有のパラメータを外部SPIフラッシュメモリに格納することが可能です。この初期化フェーズでは、パラメータをフラッシュメモリから復元して、RAM内のデータ構造にコピーします。この処理を行うことによって、新たなモータが接続された時や、制御ゲインの変更によって異なる動作条件が反映された後に、コードを再コンパイルする必要がなくなります。 保存および復元可能なアプリケーションパラメータには、モータ固有の設定（モータのタイプ、ポールペア数等）、エンコーダ固有の設定（エンコーダタイプ、分解能等）、電流制御ループおよび位置制御ループのゲイン設定などが含まれています。

4.1.2 非リアルタイム関数

非リアルタイム関数は、ファームウェアの初期化完了後に実行される無限ループのコンテキストから実行されます。実行する関数を以下の表に示します。

非リアルタイム関数	内容
ホストコマンドの読み取りと実行 m_interpreter()	ホストからの ASCII コマンドをコマンドディクショナリで探索し、呼び出すべきコマンド関数を検出するインタプリタ関数。ホストコマンドがパラメータの変更や通知を要求している場合には、値を直接設定・取得します。
速度プロファイル生成の実行 vpg_update()	速度プロファイルジェネレータは、PTP 動作の実行に必要な通過ポイント毎の速度プロファイルデータを生成するために、定期的に呼び出されます。処理時間は速度プロファイルデータの生成サイズに依存するため、結果は専用 FIFO バッファに格納され、位置制御ループ関数を呼び出すリアルタイムタスクが、専用 FIFO バッファを介してデータを取得します。
データ記録の開始 m_rec_begin()	データ記録機能の開始・停止条件を確認します。条件は、ユーザが利用可能なオプションセットから選択できます。
デジタル入力の読み出し inp_update()	ボード上のデジタル入力からデータを読み出し、ホストコンピュータがそのデータを使用できるようにします。すべての入力を、ホーミングシーケンスで使用するホームフラグとして設定することもできます。
インターロックの確認 interlocks()	ハードウェアの故障やアクチュエータの性能低下を示すエラーの条件。定期的に判定され、エラー条件が検出されるとサーボ制御を停止します。
ホーミングステートマシンの実行 fsm_homing()	サーボ軸を、指定された入力（ホームフラグ）が開始する位置やエンコーダインデックスを取得した位置によって定義された既知位置に移動させる一連の手順を実行します。
共有メモリアクセス shared_mem_access()	EtherCAT 通信によるモータ制御を行う場合の Cortex-M3 コア / Cortex-R4 コア間の通信に使用されます。コマンドやステータスの各種データが共有メモリを介して送受信されます。

上記関数はすべて、m_control.c ファイルに定義されている m_foreground() 関数から呼び出されます。

4.1.3 周期リアルタイム関数

周期リアルタイム関数は、モーション制御アルゴリズムを実行するもので、制御の精度は動作品質に直接影響します。たとえば、位置読み出しのタイミングでジッタが発生すると、位置ループの微分成分の計算が不正確になり、エラー補正結果の精度に影響を及ぼします。

リアルタイム演算は、PWM周期（50us）を生成するタイマによる IRQ ハンドラのコンテキストにおいて実行されます。

リアルタイム関数	内容
エンコーダ位置の読み出し <code>pos_read()</code>	位置読み出し関数は各種エンコーダ I/F に対応しています。インクリメンタルエンコーダが選択されている場合には、専用タイマカウンタレジスタを読み出します。アブソリュートエンコーダが選択されている場合には、直近のポーリング要求により結果が格納されたメモリアドレスから位置を読み出します。アブソリュートエンコーダの位置を取得すると次のポーリング処理が開始されるため、結果は次のタイムスライスで使用されます。 <code>pos_read()</code> 関数は <code>m_pos_read.c</code> ファイルに実装されています。
位置制御ループ <code>pos_loop()</code>	位置制御ループでは、従来の PID レギュレータを、速度・加速度フィードフォワード、出力バイアス、出力制限制御などの機能と合わせて使用します。 <code>pos_loop()</code> 関数は <code>m_control.c</code> ファイルに実装されています。
ADC 値の読み出し <code>crnt_read()</code>	電流フィードバックは、電流ループ制御アルゴリズムの実行およびモータとアンプの過負荷を監視するインターロックの判定を目的とします。 ADC 値の読み出しは、各 PWM サイクルの終了時に開始するようになっています。これは、ADC の動作へのスイッチングノイズの影響を防止するために必要です。 <code>crnt_read()</code> 関数は <code>m_rzt.c</code> ファイルに実装されています。
電流制御ループ <code>crnt_loop()</code>	電流制御ループは、トルクを発生する磁束を作る直流および直交電流を計算するために、ADC のフィードバック情報とエンコーダの位置情報を使用します。電流 PI レギュレータの動作は無効にできます。また、このアルゴリズムは、各モータ位相のデューティサイクルを生成する空間ベクトル変調関数を呼び出します。 本関数は、選択したモータ位相モードや位相ステータスによって異なるモードで動作します。 <code>crnt_loop()</code> 関数は <code>m_control.c</code> ファイルに実装されています。
速度制御ループ <code>vel_loop()</code>	位置制御ループの出力結果に基づき、目標速度と現在速度の偏差に応じた PID 制御を行います。 <code>vel_loop()</code> 関数は <code>m_control.c</code> ファイルに実装されています。
レコーダ <code>m_recorder()</code>	リアルタイムデータの収集を実現する重要な機能で、モーション制御パラメータの調整やファームウェアの動的パフォーマンスのトラブルシューティングなどを目的とします。レコーダは、最大 4 つの変数の現在値を循環バッファに格納します。データレコーダの開始と停止はホストコンピュータで設定可能です。 <code>m_recorder()</code> 関数は <code>m_recorder.c</code> ファイルに実装されています。

4.1.4 通信関数

1つ目の通信関数は、ホストコンピュータから受信したコマンドに回答して実行されます。関数は、ホストコンピュータから新しいデータを受信したことや、ホストコンピュータに対して更にデータを送信可能であることを知らせる割り込みハンドラによって呼び出されます。

2つ目の通信関数は、アブソリュートエンコーダのインタフェースを処理します。通常は、現在位置を定期的に取得するための要求を発行します。

通信関数	内容
ホストからのデータ受信 m_rx_interrupt()	<p>SCI からの割り込み要求に回答し、ホスト通信プロトコルの処理を行います。本関数は、コマンドタイプ (ASCII またはバイナリパケット) の識別と受信したコマンドのデコードを行います。バイナリパケットプロトコルの処理を行う場合には、チェックサムの計算とエラー処理も行います。</p> <p>本関数は、各物理インタフェースに対応するバッファを管理することで、同時に発生したコマンド要求を処理します。コマンドインタプリタは、コマンド要求と応答バッファを含むコマンドデータ構造体へのポインタとともに呼び出されます。これにより、異なるホストインタフェースを同時にサポートすることが可能となります。</p> <p>本関数は m_rzt.c ファイルに実装されています。</p>
ホストへのデータ送信 m_tx_interrupt()	<p>コマンド要求の結果をホストに返します。通信タスクにおける CPU の介入を最小限にするために、SCI FIFO バッファを使用します。</p> <p>本関数は m_rzt.c ファイルに実装されています。</p>
アブソリュートエンコーダデータのポーリング	<p>最新の位置フィードバックを制御アルゴリズムに提供することを目的とします。この処理は、RZ/T1 デバイスのエンコーダ I/F 部で行われ、メイン CPU を必要としません。</p> <p>ポーリングはリアルタイム制御タスクによって開始します。エンコーダ位置のポーリングの結果は、次のタイムスライスで使用できるよう共有メモリに格納されます。</p> <p>エンコーダ I/F 関数は、src\encoder フォルダ下の各エンコーダ通信プロトコルに対応した名前のファイルに実装されています。</p>

4.2 データ型

RZ/T1 ソリューションキットファームウェアでは、すべてのデータ型を `m_common.h` ファイルに定義しています。最も重要なデータ型とその用途を以下の表に示します。

データ型	説明
TReg32	32ビット値をそれぞれアクセス可能な2つの16ビット値と4つの8ビット値で表します。
TMotionParams	目標位置、速度、加速度、加加速度、速度プロファイルモードなど、PTP動作の定義に必要なすべてのモーションパラメータをカプセル化します。
TMotionProfile	特定時点でのモーションプロファイルの状態を保持します。速度プロファイル生成の状態、位置、速度、加速度、停止距離のスナップショットが含まれます。
TPosVel	ホストコンピュータからの速度プロファイルのストリーミング（PVTストリーミング）で使用する位置と速度をセットで保持します。
t_motor_pars	すべての永続モータパラメータを一つにまとめます。フラッシュメモリに保存および復元可能なデータ構造体の複製の維持に使用されます。
t_motor	1台のサーボモータを制御するために必要とされるすべての永続パラメータとランタイムパラメータにアクセスするためにファームウェアが使用するメインデータ構造体。この型のデータ構造のインスタンスを複数作成することによって、複数のモータの制御を容易にします。特定のモータインスタンスへのポインタを最初のパラメータとして受け取ることで、すべてのモーション制御関数が本データ構造上で動作します。
t_trace	データ記録アルゴリズムの動作を制御するすべての設定をカプセル化します。
t_console	ホストコンピュータと Biplane ポート上の各通信インタフェース間の通信リンクをカプセル化します。本データ構造体は、コマンドバッファ、応答バッファ、および割り込みハンドラがバッファアクセスに使用するポインタを含んでいます。
t_command	このデータ型は、コマンドを定義し、インタプリタコードを実装する <code>m_interpreter.c</code> ファイルに定義されています。本構造体は、変数や関数へのポインタと、それらをホストコンピュータに提示するために使用する ASCII コマンド名との対応付けを行います。

4.3 データ構造 / 変数

ファームウェアが使用するデータ構造は、前章で説明したデータ型のインスタンスです。特定の変数のインスタンスと、それらインスタンスのファームウェアでの用途を以下の表に示します。

データ構造のインスタンス	内容
<code>t_motor m1, m2</code>	本データ構造体は、ソリューションキットファームウェアがサポートする2つのサーボモータのそれぞれ固有の設定を保持します。
<code>t_console con0, con1, con2</code>	ホストコンピュータとのシリアル通信を行う各通信インタフェース用変数
<code>short g_counter</code>	本変数は、リアルタイム制御アルゴリズム動作終了時のPWMタイマカウンタの最新値を保持します。この値は、リアルタイムタスクのCPU使用率を監視する際に使用します。
<code>long g_tick</code>	本変数は、タイムスライス毎にインクリメントされます。リアルタイムタスクとメインループ間の処理の調整に使用します。
<code>short g_suspend</code>	リアルタイム関数の実行を一時的に阻止するためのフラグ。フラッシュメモリ書き込みなどのタイムセンシティブな動作が、リアルタイム関数による影響を受けないようにします。
<code>t_command Commands[]</code>	すべてのホストコマンドが定義されているコマンドデータ構造の配列。ASCII名、タイプ、およびコマンドを実行する関数へのポインタまたは参照パラメータを保持する変数へのポインタで構成されます。

4.4 列挙型 / マクロ / 定数

以下の表に、本ファームウェアで定義されているすべての列挙型と各値について説明します。

列挙型	説明
<code>ETYPE</code>	サポートする各種エンコーダタイプを定義します。
<code>VPG_STATE</code>	速度プロファイルジェネレータの状態（完了、加速、減速、等速等）を定義します。
<code>VPG_MODE</code>	VPGモード（台形、スプライン、ベジェ）を定義します。
<code>HOMING_STATES</code>	ホーミングアルゴリズムのステートマシンを定義します。
<code>CommutationModes</code>	電流ループアルゴリズムの動作を定義します。
<code>ParserStates</code>	バイナリプロトコルパーサの状態を定義します。
<code>ProtocolTypeRequest</code>	ホストメッセージのタイプ（ASCII、パケット）を定義します。
<code>PacketCode</code>	受信パケットのタイプを定義します。
<code>PacketError</code>	パケットプロトコルのインタプリタが通知する、起こり得るエラーを定義します。

5. 初期化関数とスタートアップ関数

5.1 ブートローダ

ブートローダは、QSPI インタフェースを初期化し、実行可能コードをフラッシュメモリから RZ/T1 の RAM へ転送します。データ転送が完了すると、実行フローは `r_cg_main.c` ファイルに定義されている `main()` 関数に進みます。

本関数を実装するファイルは、プロジェクトフォルダ `startup\spi` に格納されています。

5.2 周辺機能の初期化

本ファームウェアで使用される RZ/T1 のすべての周辺機能は、AP4 ツールで自動生成したコードにより初期化されます。ソースファイルは生成されたフォルダに格納されます。以下の表に、ソースファイルを示します。

ファイル名	周辺機能
<code>r_cg_cg.c</code>	クロック生成回路
<code>r_cg_gpt.c</code>	GPT タイマの初期化 (PWM 生成に使用)
<code>r_cg_intprg.c</code>	例外ハンドラ
<code>r_cg_main.c</code>	ブートローダによって呼び出されるメインエントリーポイント
<code>r_cg_mpc.c</code>	端子機能を GPIO または周辺機能に設定
<code>r_cg_mtu3.c</code>	MTU タイマをインクリメンタルエンコーダ位相の位置デコーダに設定
<code>r_cg_port.c</code>	GPIO 端子の状態、モード、強度を設定
<code>r_cg_s12ad.c</code>	電流フィードバック用 ADC を設定
<code>r_cg_scifa.c</code>	SCI 通信インタフェースを設定
<code>r_cg_poe3.c</code>	過電流検出用 POE3 ユニットを設定

5.3 ファームウェアの初期化

ファームウェアの初期化は、周辺機能の動作開始、内部データ構造の初期化、フラッシュメモリに保存されたモータパラメータの復元、およびエンコーダ IF ブロックの設定を行うためのものです。ファームウェア初期化シーケンスは、`m_rzt.c` ファイルに定義されている `m_startup()` 関数によって実行されます。

ファームウェアでは、すべてのモータ固有のパラメータを定義するデータ構造の 2 つのインスタンス `m1`、`m2` が定義されています。

最初に、スタートアップ関数は、各チャンネルに割り当てられたタイマのレジスタを指すメンバを初期化します。異なるタイマチャンネルへのポインタを使用することで、各タイマの特定アドレスに関係なく、2 チャンネル間で動作する関数の共有が可能になります。

次に、スタートアップアルゴリズムが各モータの `setup_motor()` 関数を呼び出します。この関数は、全モータのデータ構造メンバのデフォルト値を設定します。その後、`m_Restore()` 関数を呼び出し、フラッシュメモリに保存したパラメータを読み出します。

データ構造が初期化されると、スタートアップ関数は PWM 出力生成タイマを起動し (`R_GPTx_Start()`)、インクリメンタルエンコーダのカウント処理を行うタイマを起動します (`R_MTU3_Cx_Start()`)。

続いて、スタートアップルーチンで割り込みベクタ (`SCI0 ~ SCI2`) を新しいハンドラに割り当て直します。これは、これらのベクタをまとめて処理するために行います。`setup_scif()` 関数を呼び出すと、各 SCI インタフェースのデータ送受信バッファのポインタと、`t_console` 型のデータ構造 (`con0 ~ con2`) が設定され、それぞれのチャンネルが独立して動作可能となります。最後に、`R_SCIFAx_Start()` 関数を呼び出して、SCI インタフェースを有効にします。

スタートアップ手順では、「R」の文字を SCI2 インタフェースに出力し、シリアルインタフェースに接続されたホストに、モジュールの準備が完了したことを知らせます。

次に、データ記録機能のサポートに必要なデータ構造を初期化するため、`m_TraceSetup()` 関数を呼び出します。

最後に、スタートアップアルゴリズムがエンコーダインタフェースのハードウェアを初期化します。この処理は、モータのデータ構造にエンコーダタイプが設定されていることを前提としています。エンコーダタイプがインクリメンタル型でない場合、アルゴリズムは 150ms 待機した後、各モータの `setup_encoder()` 関数を呼び出します。各チャンネル (`m1`、`m2`) に対してセットアップ関数が呼び出されている間は、RZ/T1 デバイスと Biplane ボードはチャンネル 2 に接続されているアブソリュートエンコーダのみをサポートします。

これをもってファームウェアの初期化が完了し、実行フローは `m_foreground()` 関数を繰り返し呼び出すメインループに戻ります。

6. サーボ制御

サーボ制御ループは、モータの目標位置を維持することを目的とします。これは、目標位置と現在位置との差分を定期的に求め、モータが行う補正を計算することで実現されます。

はじめに、サーボ制御ループが有効になると、目標位置がモータの現在位置と同一になるように設定されます。この状態で、重力、電圧低下、機械力などの外乱を補正することで、アルゴリズムは現在位置を維持します。

モーションコマンドが発行されると、専用アルゴリズム（速度プロファイルジェネレータ）がサーボ制御ループを実行するごとに目標位置を算出します。目標位置に追従するサーボ制御ループの機能によって、設定した速度と加速度での動作を実現します。

サーボ制御ループの動作と、それに関わる主な関数を以下の図に示します。

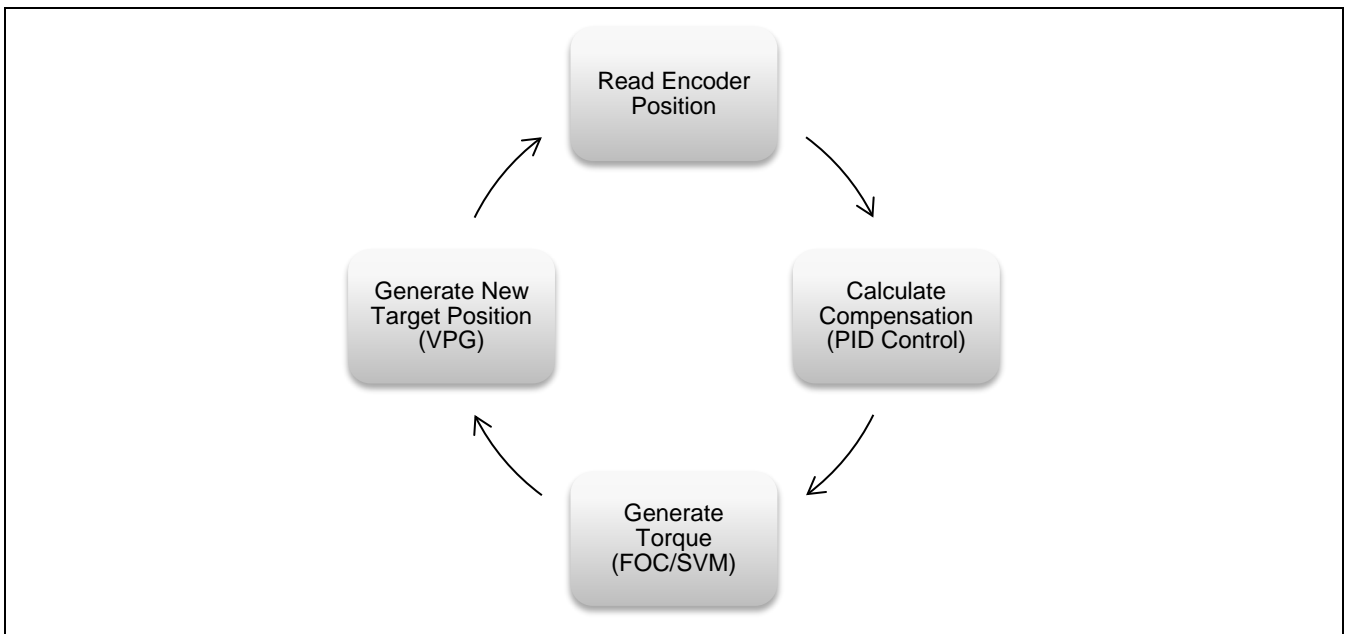


図6.1 サーボ制御ループ

サーボ制御ループは、設定された期間、継続的に実行されます。各動作間の時間の精度は、位置誤差補正の算出精度と、各動作終了時の処理時間を最小化するコントローラの機能にとって重要です。

6.1 モータ位置 エンコーダインタフェース

位置フィードバックは、モータ位置の正確な情報を提供することを目的とします。モータコミュテーション制御への入力パラメータおよび以下に記述する位置制御への入力パラメータとして使用されます。

本ファームウェアには、エンコーダフィードバックの各種オプションがあります。使用可能なエンコーダオプションは、列挙型の変数 `ETYPE` に記述されています。現在設定されているエンコーダタイプは、`t_motor` データメンバの `encoder_type` に格納されています。この変数の有効な設定を以下の表に示します。

ETYPE 列挙型	エンコーダタイプ
ETYPE_INCREMENTAL = 0	インクリメンタルエンコーダ：位置の変化は、パルス間の位相が動作の方向を示す 2 相パルスシーケンスで表されます。タイマ動作の専用モードで方向をデコードし、位置の変化を表すパルスをカウントします。ソフトウェアがタイマカウンタから現在位置を読み出します。エンコーダの分解能は 16 ビットで、ソフトウェアで 32 ビットに拡張されます。
ETYPE_APE_ENDAT = 1	EnDat 2.2 通信プロトコルのアブソリュートエンコーダ
ETYPE_APE_BISS = 2	BiSS 通信プロトコルのアブソリュートエンコーダ
ETYPE_APE_FACODER = 3	FA-CODER 通信プロトコルのアブソリュートエンコーダ
ETYPE_APE_AFORMAT = 4	A-format 通信プロトコルのアブソリュートエンコーダ
ETYPE_APE_HIPERFACE_DSL = 5	HIPERFACE DSL 通信プロトコルのアブソリュートエンコーダ

インクリメンタルエンコーダのフィードバック位置情報は、電源投入時は不明です。このため、フェージング（位相合わせ）によるロータ位置の特定や、ホーミング（原点出し）による機械位置の特定といったアルゴリズムの実行が必要となります。この他の特徴として、インクリメンタルエンコーダは、1 回転ごとにインデックスパルスを出力します。RZ/T1 のタイマは、エンコーダのインデックス信号をトリガ信号として使用し、その信号が発生した位置を取得します。この動作はハードウェアにて定義され、モータの回転速度やソフトウェア遅延による影響を受けません。このインデックス取得機能は、ホーミング処理の一環として、モータ調整システムの初期化を正確に行うために用いられます。

アブソリュートエンコーダでは、フェージングやホーミング処理は必要ありませんが、追加の設定パラメータ（ビットレート、バス遅延補正）が必要となります。アブソリュートエンコーダの仕様によっては、バッテリー駆動の多回転カウンタが必要となり、バッテリーの故障に備えて、ソフトウェアによるエンコーダのステータスの監視が必要となる場合があります。RZ/T1 のエンコーダ I/F 部は、アブソリュートエンコーダとのすべての通信を CPU の介入なしに行います。

アブソリュートエンコーダは、アプリケーション固有の情報を内部 EEPROM メモリに格納できます。ソリューションキットファームウェアには、接続されたアブソリュートエンコーダの EEPROM メモリへのアクセス手段が含まれています。

注意

RZ/T1エンコーダインタフェースは、電源投入時に一度だけ初期化できます。このため、設定されているアブソリュート型を別のアブソリュート型に変更するには、一旦、インクリメンタル型に切り替える必要があります。以下は、BiSSからEnDatエンコーダに切り替える場合の例です。

<現在のエンコーダタイプが ETYPE_APE_BISS の場合>

1. エンコーダタイプをインクリメンタルに設定
2. モーションパラメータをフラッシュメモリに保存
3. ファームウェアを再起動
4. エンコーダを EnDat に設定
5. モーションパラメータをフラッシュメモリに保存

アブソリュート型とインクリメンタル型エンコーダ間の切り替えに、電源再投入の必要はありません。

6.2 モータ制御 トルク生成

本ファームウェアには、DC ブラシモータ、ブラシレス DC モータの 2 つの主要なモータをサポートする関数があります。制御するモータのタイプは、モータ制御用データ構造のメンバ `motor_type` に定義されています。 `motor_type` の有効な設定を以下の表に示します。

motor_type 設定	モータタイプ
2	DC ブラシ / ボイスコイルモータ
3	3 相 BLDC/PMSM モータ

モータ制御アルゴリズムは、入力パラメータに比例したモータトルクの生成に使用されます。

直流 (DC) ブラシモータには、ボイスコイルや単相電力で制御可能なメカニズムに基づいたアクチュエータが内蔵されています。モータのトルクは常に印加電圧に比例します。2 つのコントローラ出力の PWM デューティサイクルの計算は、 `m_control.c` ファイルの `update_pwm2()` 関数が行います。

ブラシレス DC (BLDC) モータには、永久磁石同期モータ (PMSM) やリニアモータ、AC サーボモータなどがあります。これらモータはすべて、3 相固定子巻線と、異なる極対数の永久磁石ロータによってトルクを発生させます。

3 相モータは、任意の大きさや方向の磁束を生成する 3 相電圧を発生させることによって制御します。この磁束は空間ベクトルで表され、空間ベクトル変調と呼ばれる方法で計算されます。空間ベクトル変調では、目標とする相電圧を生成するために、2 つの磁束成分 (大きさ、角度) を利用し、タイマレジスタに格納されている値の 3 相 PWM デューティサイクルを生成します。

トルクを発生する磁束の角度は、常にロータの現在位置に依存します。コントローラ再開時、ロータの位置は特定されません (インクリメンタルエンコーダを使用している場合)。このため、後述するフェーシングアルゴリズムの実行が必要となります。フェーシングが完了するまでは、モータ制御は、ホールセンサ (使用可能な場合) が出力するロータ位置フィードバックのみに依存します。このモードでは、モータ制御は、ホールセンサで識別された 6 つの角度のうちの 1 つの角度で電圧ベクタを生成することによって実施されます。以下の図に、このモードにおけるモータ制御アルゴリズムの構造を示します。

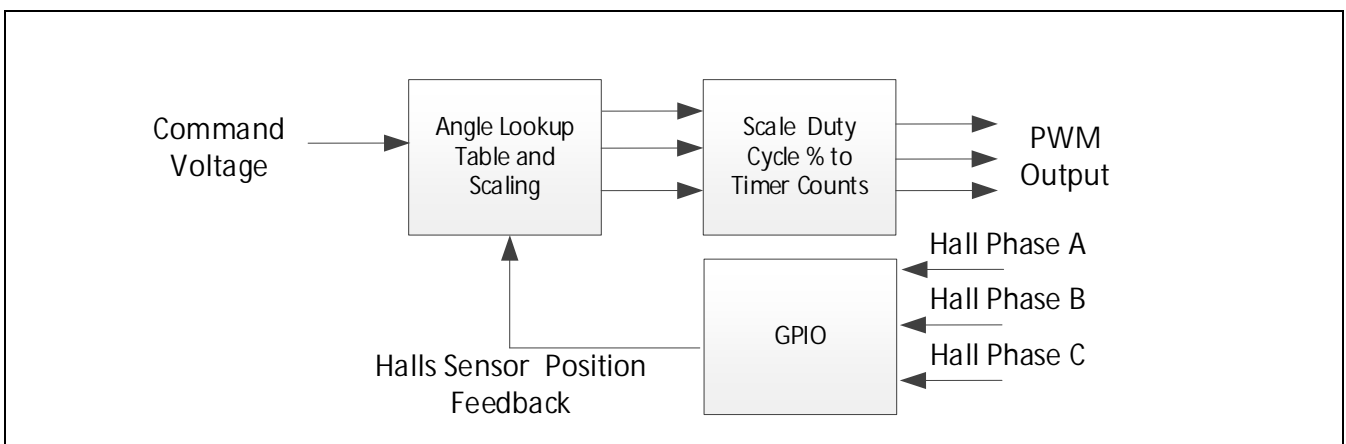


図6.2 ホールセンサベースの電圧制御

最大トルクは、磁束の方向がロータの N-S 極に対して 90° のときに生成されます。このため、磁束角はロータの現在位置から 1 電気サイクル内で取得されます。電気サイクルは、機械の 1 回転ごとのエンコーダカウント数をロータの極体数で割った数です。

トルク制御アルゴリズムの入力値は、目標電圧値または電流値として読み取られます。入力電圧を使用する方法は、計算に必要な情報がモータ位置のみであるため簡単です。この方法をオープンループ電圧制御と呼びます (図 6.3)。

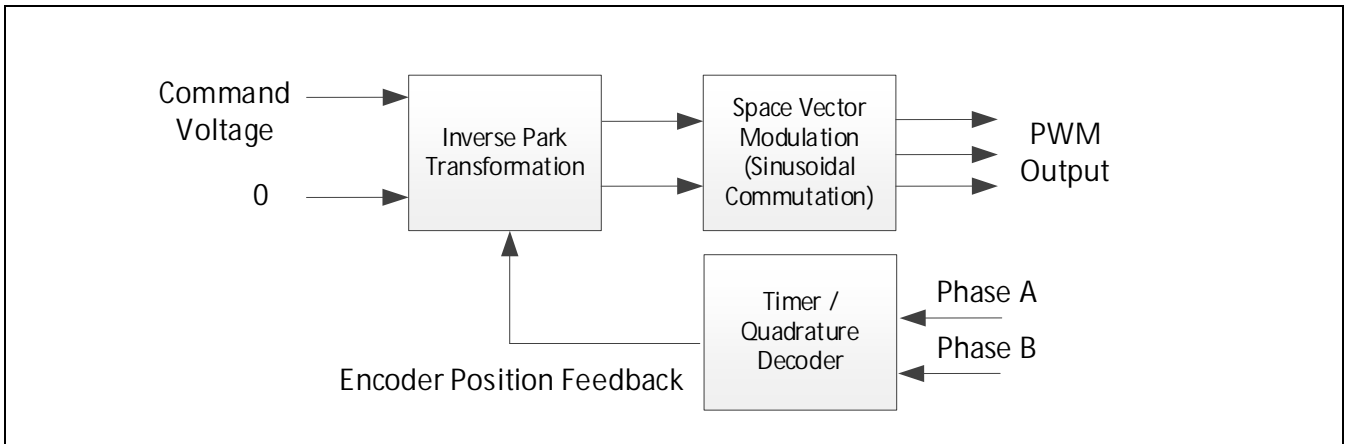


図6.3 エンコーダベースの電圧制御

電圧制御では、各モータが逆起電力 (BEMF) を生成するため、実際に生成されたトルクが反映されるわけではありません。BEMF 電圧は、モータの速度と各相の巻線数に比例します。そのため、モータ巻線に印加される有効電圧は、コントローラの PWM 出力電圧と BEMF 生成電圧との差となります。したがって、モータ巻線を通る電流はモータの速度に依存することになります。

磁界方向制御 (FOC : Field Oriented Control) アルゴリズムは、モータ速度に依存せずにトルクを生成することを目的として開発されたもので、生成中の実際の電流 (およびトルク) の測定値を出力する電流フィードバックを実施することによって実現されます。以下の図に、FOC アルゴリズムの構造を示します。

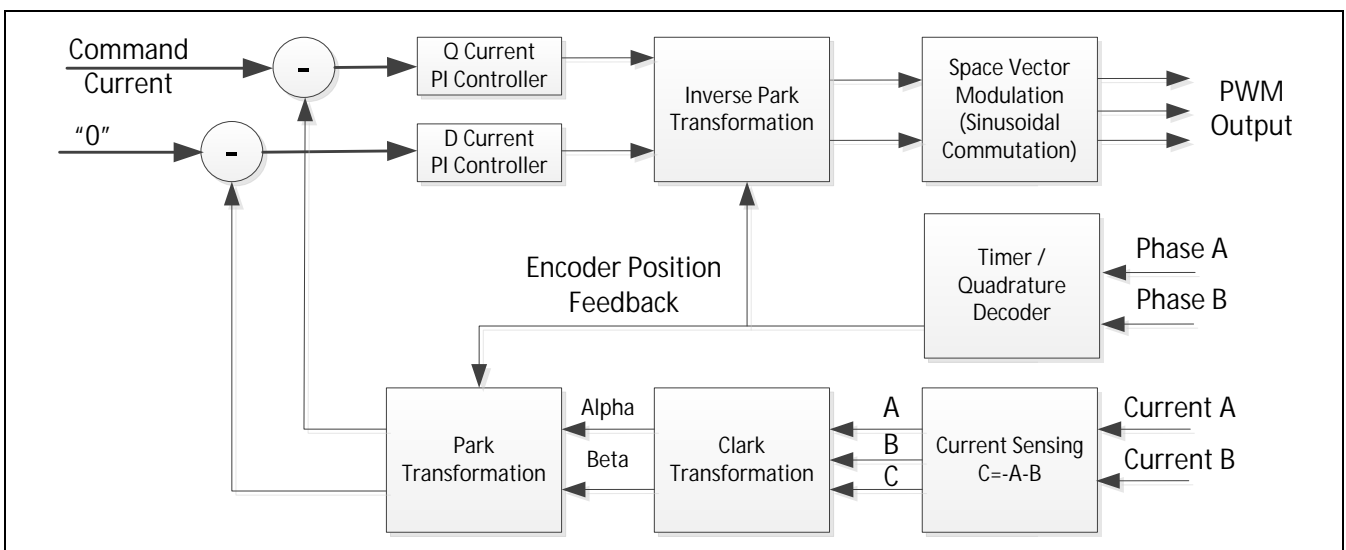


図6.4 磁界方向制御 (FOC) の構造

3 相のうち 2 相の電流は、コントローラの ADC によって継続的にサンプリングされます。3 相目の電流は 2

相の電流値を元に求めます ($I_a + I_b + I_c = 0$)。クラーク変換およびパーク変換では、トルクベクトルの成分として、以下2つの電流の現在値を表す2つの直交ベクトルを計算します。

- ・ 直交電流：ロータの N-S 極に垂直な、トルクを発生させる磁束を表す。
- ・ 直流：熱起電力を表す。常に 0 とする。

各電流はそれぞれの設定値と比較され、エラーは比例積分 (PI) レギュレータで補正されます。FOC アルゴリズムは、`m_control.c` ファイルの `commutate_foc()` 関数に実装されています。このアルゴリズムでは、入力値として以下のモータパラメータを使用します。

t_motor データメンバ	内容
counts2rad	1 電気サイクルあたりのエンコーダ分解能を表す係数
phase_angle	ロータの N 極に対する磁束ベクトルの方向を表すパラメータ
p_iu	サンプリングした U 相電流を保持する変数へのポインタ
p_iv	サンプリングした V 相電流を保持する変数へのポインタ
foc_id	直流電流の計算値
foc_iq	直交電流の計算値
foc_id_err	直流電流誤差の計算値
foc_iq_err	直交電流誤差の計算値
foc_vd	直流電圧の計算値
foc_vq	直交電流の計算値
foc_alpha	電圧ベクトルの 成分
foc_beta	電圧ベクトルの 成分

モータ制御アルゴリズム実行の最終ステップは、空間ベクトル変調です。空間ベクトル変調は、`m_commutation.c` ファイルの `commutate_svm()` 関数を使用します。この関数では、FOC アルゴリズムによって計算された 電圧と 電圧を受け取り、各相に対応するデューティサイクルを返します。

トルク生成のアルゴリズムは、モータ変数 `commutation_mode` の設定により、実行中に設定可能です。

commutation_mode の設定	内容
0	空間ベクトル変調による電圧制御
1	磁界方向制御による電流ループ
2	ホールセンサベースの制御
3	外部 / ユーザ定義のフェーズ電圧設定

6.3 位置制御 PIDレギュレータ

位置制御アルゴリズムでは、位置誤差を入力値として受け取り、位置誤差に応じた出力結果を計算します。位置ループ制御関数 `pid_calc()` は、`m_pid_calc.c` ファイルに実装されています。

PIDレギュレータの構造を以下の図に示します。

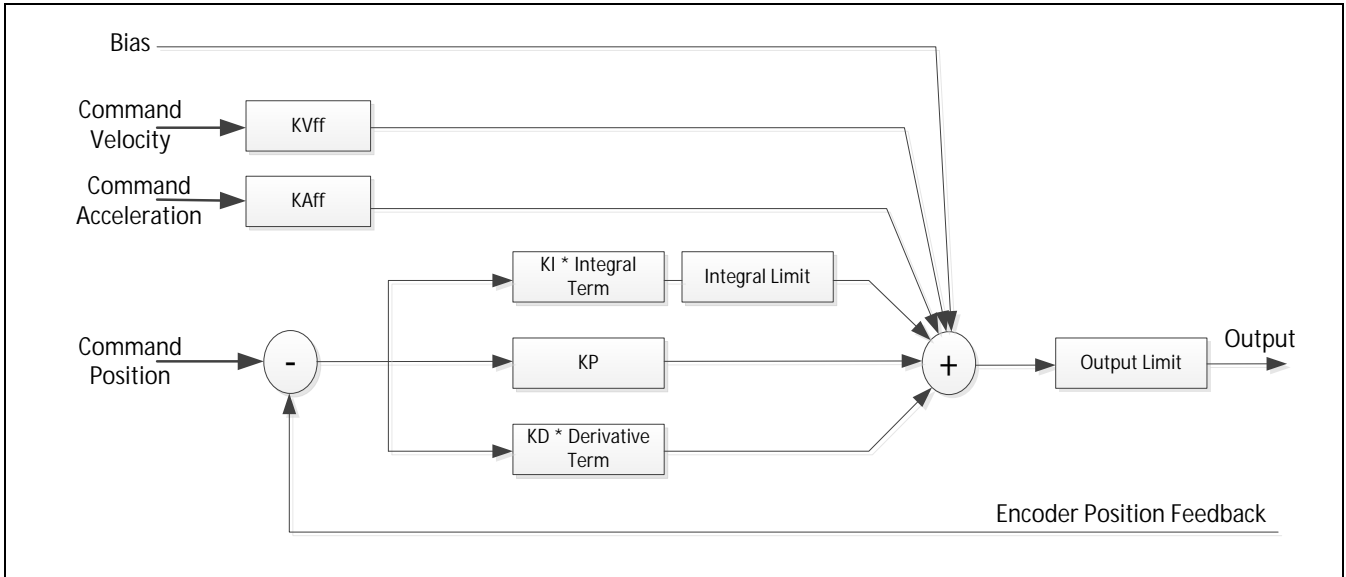


図6.5 PIDレギュレータ構造

位置誤差が計算され、入力パラメータとして `pid_calc()` 関数に渡されます。位置誤差はエンコーダカウントで表されます。

位置制御関数の設定パラメータは、`t_motor` データメンバに格納されます。以下の表に、`t_motor` データメンバの詳細を示します。

<code>t_motor</code> データメンバ	内容
<code>crnt_kp</code>	比例ゲイン (0 ~ 32767)
<code>crnt_ki</code>	積分ゲイン (0 ~ 32767)
<code>crnt_kd</code>	差動ゲイン (0 ~ 32767)
<code>integral_limit</code>	積分制限値 (0 ~ 32767)
<code>crnt_kvff</code>	速度フィードフォワード (速度ゲイン)
<code>crnt_kaff</code>	加速度フィードフォワード (加速度ゲイン)
<code>crnt_bias</code>	バイアス (出力値に直接加算)
<code>cmd_vel</code>	指令速度
<code>cmd_acc</code>	指令加速度
<code>pos_loop_limit</code>	出力制限値 (0 ~ 32767)
	一時変数:
<code>pos_error</code>	<code>pid_calc()</code> 関数の最後の呼び出しから位置誤差を格納
<code>derivative_err</code>	位置誤差の計算された導関数 (最終と現在の位置誤差との差分) を格納
<code>integral_err</code>	位置誤差の計算された積分値を格納

PID 制御パラメータは、ホストコンピュータから直接アクセスすることはできません。これらのパラメータは、サーボ制御有効時に、新たな動作が開始、または現在の動作が停止したときにのみ、バッファに格納され、pid_calc()関数で使用される変数にコピーされます。パラメータは、上述の関数によって呼び出される m_update()関数内で更新されます。パラメータのバッファリングメカニズムと関連する関数を以下の図に示します。

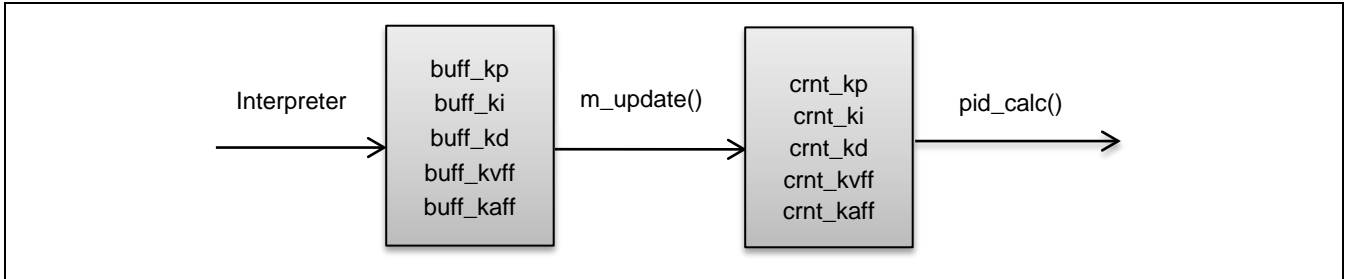


図6.6 位置ループパラメータのバッファリング

PID ゲインをバッファリングする目的は、それらを同期的に、すべて同時に更新することにあります。パラメータを1つずつ設定したり、更新を中断すると、制御関数の出力においてグリッチが発生する場合があります。この結果、不要な位置誤差が生じることがあります。

pid_calc()関数は、前章で記述したモータ整流アルゴリズムへの入力値として与えられる新たな出力値を返します。出力値の範囲は±32767 です。

6.4 モーションプランニング 速度プロファイルジェネレータ

モーションプランニングでは、ある地点から別の地点への移動をどのように実行するかを指定します。はじめに、アルゴリズムが現在位置と目標位置を使用して、目標とする移動距離を計算します。次に、指定された加速度、速度、減速度の最大値を使用して、加速動作期間と減速動作期間の長さを計算します。

すべての動作期間を含む、代表的な台形速度プロファイルを以下の図に示します。

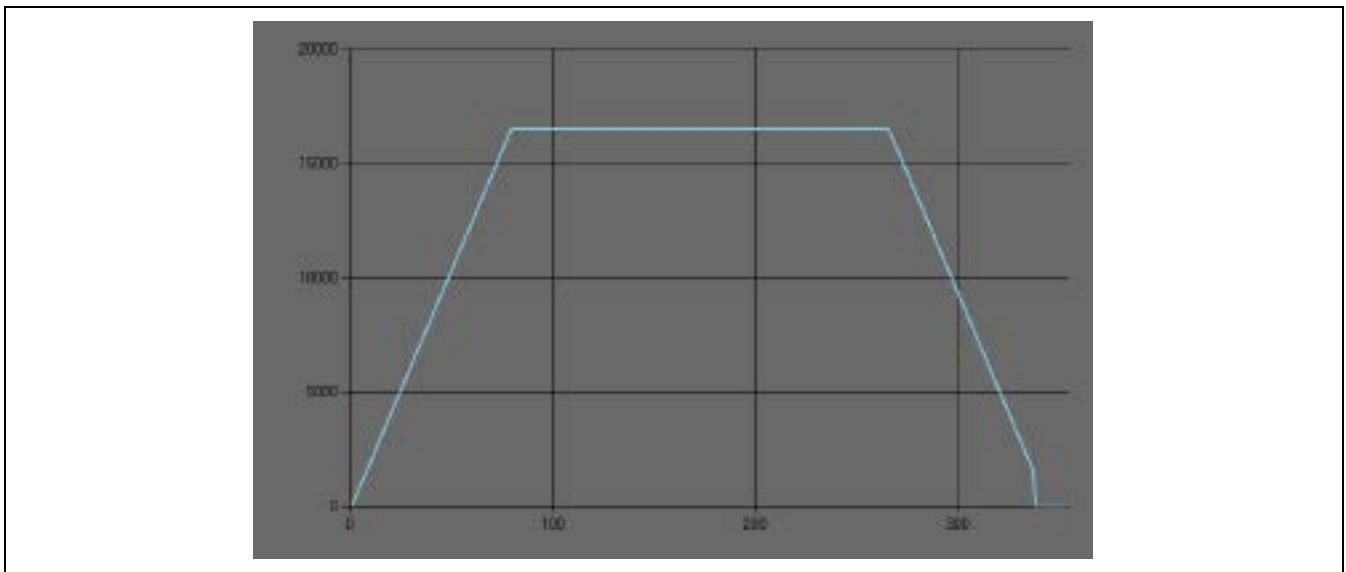


図6.7 台形速度プロファイル

アルゴリズムは、移動距離とモーションパラメータに基づいて、等速動作期間の長さも決定します。移動距離が短すぎると、動作が最大目標速度に到達しない場合があります。

以下の図は、移動距離が短いために最大速度に到達しない場合の例です。

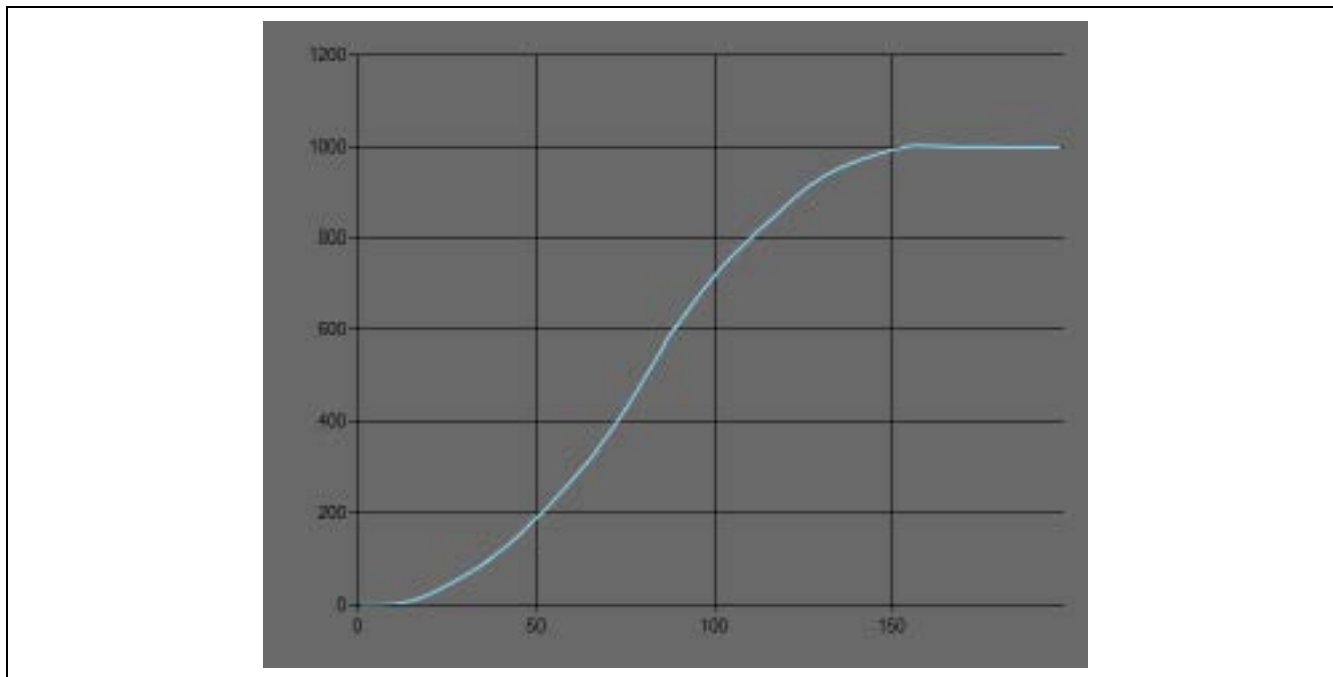


図6.8 1000までの短距離移動時の位置プロファイル

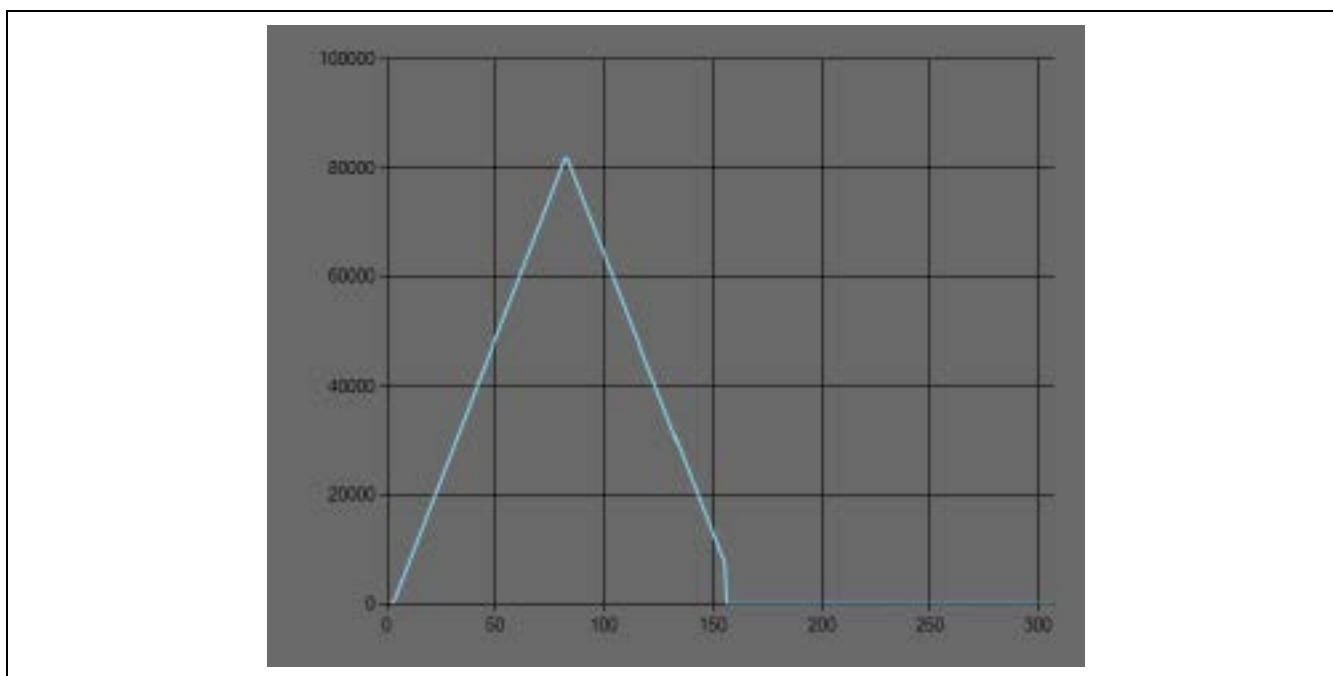


図6.9 短距離移動時の台形プロファイル

RZ/T1ソリューションキットファームウェアは、アプリケーション固有の要求に対応するために、複数の数学関数を使用して速度プロファイルを形成する、各種速度プロファイル生成アルゴリズムを実装しています。以下の図に、スプライン関数を使用して計算された速度プロファイルを示します。

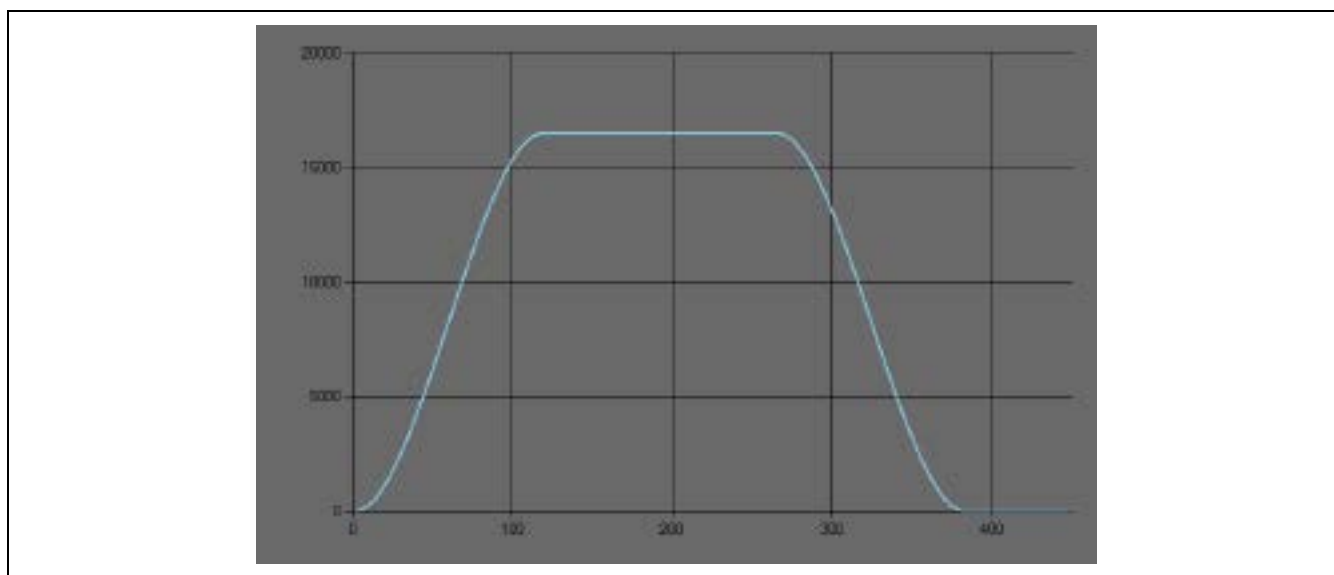


図6.10 スプラインベースの速度プロファイル

滑らかな速度プロファイルでは、台形プロファイル特有の振動が減少していますが、目標位置に到達するまでにより長い時間がかかります。特定の速度プロファイルの選択には、目標位置への到達時間と動作終了時の処理時間とのトレードオフが伴います。

以下の図は、ベジェ曲線を使用して生成された速度プロファイルです。この速度プロファイルは、加速度と減速度の設定だけでなく、それらの微分値（変化率）を指定することも可能です。

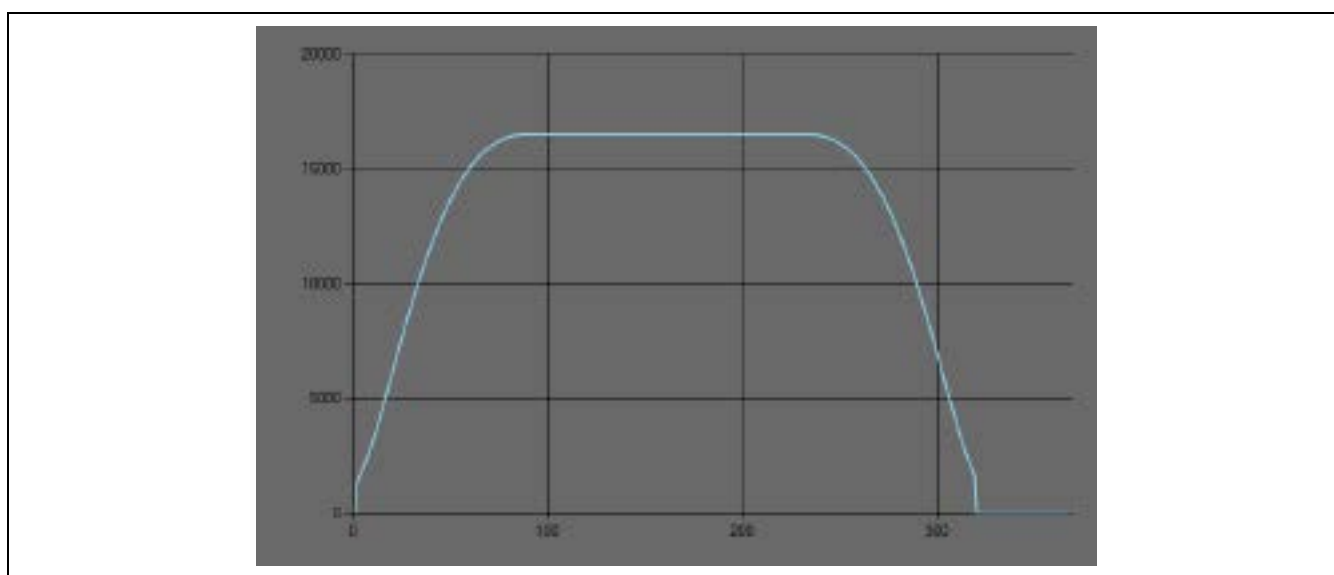


図6.11 ベジェ曲線ベースの速度プロファイル

上記すべての速度プロファイルは、ある地点から別の地点への単一動作のみ実行します。動作の開始・終了時の速度はゼロです。この仕組みは、モータの複雑な速度プロファイルを生成することが求められる、ロボット、ガントリーステージ、CNCマシンなどの複雑なメカニズムを制御するには不十分です。

ホストとコントローラ間の通信帯域は制限されているため、速度プロファイルは一定のタイムスライス時間（通常は 5ms ~ 20ms）、位置と速度のセットで示されます。このプロファイル時間に対する位置、速度、時間を PVT ポイントと言います。PVT ポイントは各コントローラにストリーミングされ、次にコントローラが補間アルゴリズムを実行し、目標位置と目標速度の設定値を 50us ごとに生成します。

速度プロファイルジェネレータ（VPG）の動作は、ステートマシンにより以下のように定義されます。

ステート	値	内容
アイドル状態 / 動作完了	0	デフォルト / 最終状態
加速	1	動作は加速中
減速	2	動作は減速中
等速	3	一定速度
ストリーミング / PVT 補間	4	補間

VPG の状態は、`t_motor` データメンバの `vpg_state` に格納されます。

動作を開始すると、すべての動作準備を `update_ctrl()` 関数で処理します。次に、`update_ctrl()` 関数は、現在選択されている VPG のスタートアップ関数を呼び出し、新たな VPG の状態を返します。これ以降、VPG は定期的呼び出され、位置制御ループの 1 サイクルごとに設定された位置と速度を生成し、その都度、新たな状態を返します。モーションプロファイルが完了すると VPG はアイドル状態となり、定期 VPG 関数は呼び出されなくなります。

各速度プロファイルタイプのスタートアップ関数および定期呼び出し関数を以下の表に示します。

VPG タイプ	スタートアップ VPG 関数 <code>update_ctrl()</code>	定期 VPG 関数 <code>vpg_update()</code>
台形	<code>vpg_trap_start()</code>	<code>vpg_trap_next()</code>
スプライン曲線	<code>vpg_spline_start()</code>	<code>vpg_spline_next()</code>
ベジェ曲線	<code>vpg_bezier_start()</code>	<code>vpg_bezier_next()</code>
PVT 補間	<code>vpg_pvt_start()</code>	<code>vpg_pvt_next()</code>

VPG で生成されたデータは直接位置制御ループに渡されず、専用の FIFO バッファに格納されます。この仕組みによって、位置制御ループの動作に対して、VPG 関数を非同期で実行することが可能となります。これには 2 つの目的があります。

1. 速度プロファイルの生成は複雑で、事前計算が必要な場合があります。非同期にすると、リアルタイム位置ループと電流ループに対する「過大な」計算量が不要になります。
2. VPG の計算をリアルタイム制御ループから分離させることによって、VPG を別の CPU コアや、別のネットワーク機器で動作させる場合の再設計が容易になります。

6.5 モーション制御パラメータ

モーション制御パラメータには、モータの移動位置とそこへの到達方法を定義した設定があります。また、実行する動作は、モーションコントローラの現在の状態（現在位置、位置誤差、現在の速度、現在の状態等）によって決まります。

6.5.1 目標位置

目標位置はエンコーダカウントで表され、絶対目標位置（ASCII コマンド ABS）への要求と合わせて明確に定義することができます。

また、目標位置は現在位置（ASCII コマンド REL）に対する距離として定義できます。この場合の目標位置は、動作開始時のモータ位置によって決まります。相対距離を指定してから動作開始までの間に、現在位置が変わる可能性があることにご注意ください。それにより予測目標位置が変更されます。

6.5.2 最大速度

最大速度パラメータは制限値を定義します。制限値に到達するか否かは、他のモーションパラメータに依存します。速度パラメータの単位は「位置ループ周期ごとのエンコーダカウント」と定義されています。位置ループの周期は非常に短い（50us）ため、速度値は 65536 を乗算した 16.16 ビット固定小数点の数値として処理されます。

1 秒あたりのエンコーダカウントからソリューションキットファームウェア形式への単位の変換では、数値に位置ループのタイムスライスを乗算した後、さらに 65536 を乗算します。

たとえば、位置ループが 100us で動作する場合、1 秒あたり 5000 のエンコーダカウントは、

$$5000 \times 0.0001 \times 65536 = 32768$$

に変換されます。1 秒あたり 5000 のエンコーダカウントに相当する最大速度値は、32768 となります。

6.5.3 最大加速度 / 減速度

最大加速度と減速度のパラメータは、速度プロファイルの勾配を定義します。速度プロファイルの勾配に到達するか否かは、制御ハードウェアの性能と特定のモータ負荷に依存します。加速度パラメータの単位は「位置ループ周期ごとのエンコーダカウントの二乗」と定義されています。位置ループの周期は非常に短い（50us）ため、加速度と減速度の値は 65536 を乗算した 16.16 ビット固定小数点の数値として処理されます。

1 秒あたりのエンコーダカウントからソリューションキットファームウェア形式への単位の変換では、数値に位置ループのタイムスライスの二乗を乗算した後、さらに 65536 を乗算します。

たとえば、位置ループが 100us で動作する場合、1 秒あたり 30000 のエンコーダカウントの二乗は、

$$30000 \times 0.0001 \times 0.0001 \times 65536 = 19$$

に変換されます。1 秒あたり 30000 のエンコーダカウントに相当する最大加速度は、19 となります。

6.5.4 最大加速度 / 減速度変化率

最大加速度と減速度の変化率のパラメータは、ベジェ曲線の速度プロファイルが使用されるときのみ、速度プロファイルの勾配を定義します。ベジェ曲線の接線方向を比率で定義しているため、変化率の単位は無次元となります。変化率の値は 0 ~ 1000 の整数で表されます。

6.5.5 動作開始モード

PTP動作は単一の関数（ASCIIコマンドGO）で開始します。これによって、update_ctrl()関数が呼び出されます。update_ctrl()関数は、割り込みを無効にし、すべてのモーションパラメータと位置ループパラメータをバッファから制御ループアルゴリズムで使用される変数にコピーします。これは、制御パラメータが相互に依存しているため、すべての制御パラメータを同時更新するために行います。

台形速度プロファイルでは、モーションパラメータの変更をまとめて行うことが可能です。これにより、最終動作の完了前に、新たな動作を開始できます。新たな動作では、そのパラメータの一部または全部が変更されている場合があります。

また、PVTバッファが一定のレベルまで埋まった後に、PVTストリーミングモードで動作を開始することもできます。

6.5.6 動作停止モード

動作停止モードでは、目標位置に到達しなかった場合でも、制御された方法で動作を停止することができます。使用可能な停止モードは以下の3つです。

1. Smooth Stop：このモードでは、コントローラが、現在の動作を正しく終了させます。これは、現在の速度プロファイルジェネレータを台形プロファイルに切り替え、設定された最大減速度に基づいて目標位置を計算することで実現されます。
2. Abrupt Stop：このモードでは、コントローラは、指定された最大値の32倍の減速度を使用します。このモードは緊急事態での使用を前提としています。
3. Servo Off Stop：このモードでは、サーボ制御をオフにして、モータ巻線をインバータで短絡します。これによって、インバータは動的ブレーキモードで動作します（逆起電力（BEMF）を停止力として使用）。

7. システム制御関数

7.1 インターロック

インターロックは、制御システムの異常状態を示すハードウェアで定義された条件、またはソフトウェアで算出された条件です。インターロック条件は、`m_interlock.c` ファイルの `interlocks()` 関数に実装されています。

以下に、`interlocks()`関数で実装されるアルゴリズムの概要を示します。

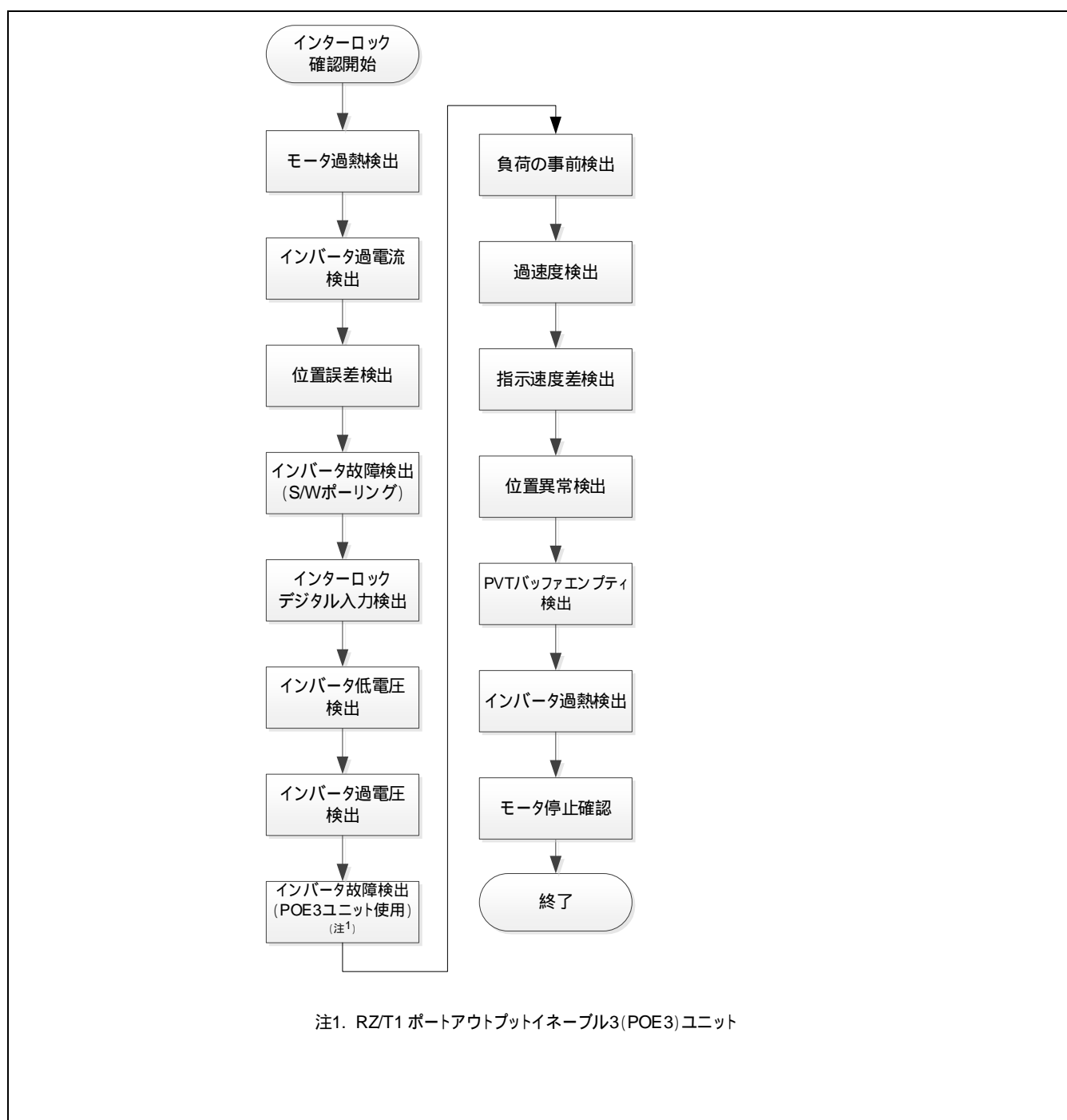


図7.1 インターロックでの判定

1. モータ過熱検出

最初のインターロックでは、消費電力を監視し、モータの発熱量を示す閾値と比較します。アルゴリズムは測定された消費電流値を二乗します。次に、公称消費電流のパラメータ (I2t_nominal) を減算します。計算結果を変数 I2t_integral で積分し、積分値を設定可能な制限値 (I2t_limit) と比較します。過熱プロセスが比較的遅いため、制限値を超えた場合、作動状況 (ACT_MtOverTemp) のフラグが立つのみで、積分値が制限値を下回るとフラグはクリアされます。

2. インバータ過電流検出

次のインターロックでは、消費電流の合計値と一定時間 (tc_limit_time) 以上超過した制限値 (tc_limit) を比較します。過電流インターロックは、機械的な障害などによるモータやモータに接続された機械への過負荷を防ぐことを目的としています。過負荷状態を検出すると、インターロック機能がモータ過電流状態 (ACT_OverCurrent) のフラグをセットし、サーボ制御ループを無効にします。その結果、すべての動作も停止します。

3. インバータ故障検出 (S/W ポーリング)

PWM アンプには、ケーブルの欠陥 (短絡) や部品不良などによる過電流からハードウェアを保護する機能があります。アンプの故障が発生すると、ハードウェア保護機能がブリッジを直ちに遮断し、専用デジタル入力信号を送信します。インターロック機能は、この入力信号を検出すると、モータアンプ異常 (ACT_AmpFault) のフラグをセットし、サーボ制御ループをオフにします。

4. 位置誤差検出

位置誤差インターロックでは、位置誤差の絶対値を監視します。位置誤差が制限値 (pos_error_limit) を超えると、専用タイマ (pos_error_timer) がエラー状態の継続時間を測定します。この時間が指定時間 (pos_error_time) を超えると、インターロック機能がすべての動作を停止します。変数 auto_stop_mode の値によっては、サーボ制御も無効にする場合があります。このインターロック条件は、ACT_PosError フラグのセットによりモータの作動状況に反映されます。

5. インターロックデジタル入力検出

インターロック機能は、すべての使用可能なデジタル入力をインターロック条件のトリガとして処理できるよう設定できます。判定される入力は、ビットマスクとして変数 dinputs_err_mask に定義されています。入力バイトとマスク値との論理積が 0 以外と判定されると、瞬時にインターロックが作動します。これによってサーボ制御がオフとなり、電源アンプ出力が無効になります。モータの作動状況は ACT_Inhibit フラグのセットで更新されます。

6. インバータ低電圧検出

インバータの母線電圧が閾値を下回ると、エラー検出フラグをセットします。

7. インバータ過電圧検出

インバータの母線電圧が閾値を上回ると、エラー検出フラグをセットします。

8. インバータ故障検出 (POE3 ユニット使用)

POE3 ユニットでインバータボードの故障信号を監視します。故障信号が検出されると、PWM 出力が自動的にハイインピーダンスに切り替わり、エラー検出フラグがセットされます。

9. 過負荷の事前検出

インバータの電流値が閾値を超えると、エラー検出フラグをセットします。「インバータ過電流検出」より低い閾値を設定し、サーボオフをマスクすることで、過電流によってサーボオフとなる前にアラームを検出することが可能です。

10. 過速度検出

モータ速度が閾値を超えると、エラー検出フラグをセットします。

11. 指示速度差検出

モータの回転速度の変化が5秒間継続して閾値を超えると、エラー検出フラグをセットします。

12. 位置異常検出

モータの位置情報が上下限閾値を超えると、エラー検出フラグをセットします。

13. PVT バッファエンプティ検出

100us ごとに速度プロファイルジェネレータの FIFO 状態を確認し、エンプティ状態の回数が閾値を超えると、エラー検出フラグをセットします。

14. インバータ過熱検出

インバータの温度が閾値を超えると、エラー検出フラグをセットします。

15. モータ停止確認

各エラー検出フラグとマスク設定の状態を確認し、条件を満たす場合にモータを停止します。

7.2 電子ギア

電子ギアにより、2つの独立したモータのソフトウェアによる同期化（カップリング）が可能です。電子ギア機能では、2つのモータの目標位置は線形的 / 比例的な関係にあります。

電子ギアが有効のとき、どちらか一方のモータ（マスタ）に目標位置のコマンドが与えられます。もう一方のモータ（スレーブ）の指令位置は、電子ギア比係数の乗算によってマスタから導き出されます。サーボモータのステータス、動作の開始・停止を制御する各コマンドも2つのモータに対して同時に作動します。

電子ギア比は、2つのモータ変数 `gear_in`、`gear_out` で定義されます。これらの変数は、電子ギア係数の計算において、それぞれ分子、分母として用いられます。

スレーブ軸の設定位置、速度、加速度の更新値は、`m_control.c` ファイルに定義されている関数 `set_slave()` に実装されます。この関数は、モータが電子ギアマスタとして指定されている場合に、サーボ制御ループで呼び出されます。電子ギアマスタは、変数 `module_type` を `TYPE_EGEAR` に設定することで指定できます。

7.3 デジタル入出力

ファームウェアは、デジタル入力の状態を継続的にポーリングします。デジタル入力は、ホールセンサへのインタフェースとして、またはインターロックやリミットスイッチを感知する汎用入力として機能します。

位置入力値は、サーボループサイクルごとに読み出されます。この処理は、`m_inputs.c` ファイルに定義されている `inp_update()` 関数で実行されます。

入力状態のフィルタ処理は、直近のサンプルがフィルタに入る前に、最新の2つのサンプルをバッファリングすることによって行われます。入力状態は、3つの同じ値のサンプルから2つを取り出して評価します。これは入力信号のノイズ耐性を向上させるために行います。

ホールセンサは、モータの適切なフェージング（位相調整）に必要な情報を提供します。センサはボード上の各種ポートに接続されます。1つは、エンコーダインタフェース（P1、P2）専用コネクタです。この場合、ホールセンサは、エンコーダチャンネルに応じて、Fn、Rn、Xn 端子（n = 1、2）に接続します。また、ホールセンサはポート P5（チャンネル1はピン2、3、4、チャンネル2はピン5、6、7）に接続することも可能です。どちらの接続方法を選択するかは、DIPスイッチ S2-1、S2-2の状態に基づいて、実行時に決められます。スイッチの設定によって、ホールセンサのチャンネル1、2への接続を指定します。

DIPスイッチ S2-1またはS2-2がON（デフォルト）のとき、ファームウェアはホールセンサの状態をコネクタ P1またはP2（F、R、X端子）から読み出します。OFFのときは、P5端子から読み出します。

ホールセンサの信号とモータ相巻線の対応は標準化されておらず、文書化されていない場合もあります。このため、ファームウェアには、コネクタを再配線せずにホールセンサを「リマッピング」する機能があります。これは `remap_halls()` と呼ばれる関数で、`m_inputs.c` ファイルに定義されています。`remap_halls()` 関数は、必要に応じて、ホールセンサ信号の所望の「配線」と信号の極性反転を定義するモータパラメータの `hall_inverted` を受け取ります。

Biplane ボードでは、RS232 と RS422 の両方のインタフェースをサポートしています。これらインタフェースはファームウェアで設定され、トランシーバ機能を制御する出力を駆動します。この出力の状態はDIPスイッチ（S2）のS2-8の状態に連動してソフトウェアで制御します。S2-8がON（デフォルト）のとき、動作モードはRS232となります。

Biplane ボードはまた、ポート P3 に接続可能な8本の24Vトレラント入力端子をサポートしています。ピン2~5はチャンネル1の入力として、ピン6~9はチャンネル2の入力として通知されます。

INP コマンドによってホストに通知される入力の一覧を以下の表に示します。INP コマンドは関数 `m_Inputs()` で実行され、両チャンネルの入力値を合わせた16ビット値を通知します。

Bit#	b7	b6	b5	b4	b3	b2	b1	b0
Name		Hall W2	Hall V2	Hall U2		Hall W1	Hall V1	Hall U1
Input Port S2-1/S2-2 dependent	0	X2	R2	F2	0	X1	R1	F1
	0	P5-7	P5-6	P5-5	0	P5-4	P5-3	P5-2
Bit #	b15	b14	b13	b12	b11	b10	b9	b8
Name	INP8	INP7	INP6	INP5	INP4	INP3	INP2	INP1
Input Port	P3-8	P3-8	P3-7	P3-6	P3-5	P3-4	P3-3	P3-2

Biplane ボードには、モータブレーキなど誘導負荷の制御を目的とした2本の高性能デジタル出力端子があります。これらはポート P4 にて使用可能です。ピン2はチャンネル1、ピン3はチャンネル2に割り当てられています。

7.4 データ記録

データ記録機能により、システム動作のリアルタイムでの分析が可能になります。この機能は、アプリケーション固有のコンテキストで、各種制御ループの性能、設定パラメータおよび効率性を分析するために必要不可欠です。データレコーダは、システム稼働中に、ユーザ定義のパラメータを最大4個バッファに格納します。サポート関数と設定パラメータにより、記録レートや、記録する変数、記録開始時間および終了予定時間が選択可能です。

記録期間は、サンプル数としてマクロ `TRACE_BUFFER_SIZE` で定義されています。デフォルトでは512に設定されています。この値は、アプリケーションがより長い記録を必要とし、RAMメモリが使用可能な場合に増加できます。全バッファは `traceData[]` という名前の単一の配列に結合されますが、配列のデータ型は短く、16ビット整数です。このため、32ビット変数が記録される時、その値は1番目と4番目のバッファ間で分割されます。データが通知されると、値は適宜結合されます。

ホストは、記録する各チャンネルのデータをコマンド CH1、CH2、CH3、CH4 で指定します。各コマンドは、該当データを示すコードをセットします。最初の8コードは32ビット変数用の予約コードです。残りは16ビットデータを参照します。

以下の表に、記録のために設定が必要な各種データ変数とコードの対応を示します。

CH1, CH2, CH3, CH4 のコード	参照モータ変数
0	モータ位置
1	指令速度
2	指令加速度
3	I2t 積分値
4 - 7	予約
8	位置誤差
9	PID レギュレータ出力値
10	消費電流の合計値
11	直流（熱発生）
12	直交電流（トルク発生）
13	直流誤差
14	直交電流誤差
15	原電流 A
16	原電流 B
17	PVT FIFO バッファ深さ
18	FOC 電圧出力 D
19	FOC 電圧出力 Q
20	リアルタイムタスクのタイミング
21	位相角

データレコーダの開始・停止条件は、ASCII コマンド TRACE (変数 `trace.Trigger`) で設定します。レコーダの各トリガ条件を表す使用可能な設定を以下の表に示します。

TRACE コード	開始トリガ条件	停止トリガ条件
0	N/A	データ記録を停止
1	即時開始	動作完了時に停止。このトリガは動作終了を確認するのに有効。
2	即時開始	バッファフル時に停止。このトリガは動作開始を確認するのに有効
3	動作開始時に開始	動作終了時に停止
4	即時開始	入力変化時に停止。入力ビットマスクは <code>trace.Level</code> 変数で定義
5	即時開始	<code>trace.Level</code> 変数で定義した閾値を超過した時
6	即時開始	<code>trace.Level</code> 変数で定義した閾値を下回った時
7	PWM 出力変化時に開始	バッファフル時に停止
8	入力変化時に開始。入力マスクは <code>trace.Level</code> 変数で定義	バッファフル時に停止
9	<code>trace.Level</code> 変数で定義した閾値を超過した時	バッファフル時に停止
10	<code>trace.Level</code> 変数で定義した閾値を下回った時	バッファフル時に停止

レコーダは、50us 間隔で実行されるリアルタイムタスクに同期して動作します。記録レートはこの時間間隔の倍数で表されます。乗数は ASCII コマンド TRATE で設定され、変数 `trace.RateMult` に格納されます。たとえば、記録レートを 1ms にしたい場合、TRATE は 20 に設定されている必要があります。

トリガ条件発生時に評価されるもう一つの変数は、ASCII コマンド TLEVEL (変数 `trace.Level`) です。この変数の値は、記録された変数と比較する閾値となります。トリガのコードによっては、記録開始条件で、値が閾値より大きいかまたは小さいかをテストすることができます。

開始トリガ条件をテストするために定期的に呼び出される関数は `m_rec_begin()` です。

停止トリガ条件をテストし、レコーディングを実施するために定期的に呼び出される関数は `m_recorder()` です。レコーダの動作モードは、ASCII コマンド TMODE (変数 `trace.Mode`) で通知します。格納されているコードの意味を以下の表に示します。

TMODE コード	動作モード/ステータス
0	アイドル状態。記録を開始した場合は停止。
1	レコーダが作動可能：動作開始時に記録を開始可能
2	記録中。バッファフルで記録停止。
3	循環バッファに記録中。動作完了で記録停止。

データ記録が完了すると、ホストはコマンド PLAY を使用して記録バッファの内容を取得できます。この要求を実装する関数は `m_Play()` です。この関数は、起動コンテキストがパケットコマンドハンドラである場合は、バイナリパケット応答の設定も行います。

7.5 モータのフェージング

フェージング処理（位相調整）の目的は、ブラシレスモータのロータの絶対角度を特定することにあります。絶対角度は、トルクを発生する磁束の方向を計算する制御アルゴリズムに必要です。インクリメンタルエンコーダからのフィードバックを使用する場合は、電源投入のたびにフェージング処理を行う必要があります。アブソリュートエンコーダはこの処理を必要としません。ただし、位相オフセット（エンコーダの1回転内位置でのロータ角を決める値）を設定する場合には、少なくとも一回はこの処理が必要となります。

選択されたフェージングモードは、モータ変数 `phasing_mode` の値によって定義されます。

フェージングに使用するアルゴリズムは要素によって異なります。以下の表に、フェージングの各種モード（オプション）と、各アルゴリズムの長所と短所を示します。

フェージングモード	内容
強制フェージング <code>phasing_mode == 0</code>	<p>このモードでは、ファームウェアが既知の角度の電圧ベクトルを形成します。電圧ベクトルは、3相の各出力に適切な PWM デューティサイクルを適用することによって形成されます。</p> <p>モータ変数 <code>phasing_time</code> で指定されている期間、モータ変数 <code>motor_power</code> で指定した大きさの電圧が付加されます。これら2つの変数を設定することで、モータが磁束方向に沿ってロータを回転させることができます。指定時間が経過すると、アルゴリズムが現在位置を保存し、そこから90度戻した地点を位相の原点としてセットします。</p> <p>この処理は、<code>m_phasing.c</code> ファイルの関数 <code>forced_phasing()</code> に実装されています。</p> <p>この機能はシンプルかつ安定性が高いことが利点として挙げられますが、処理の実行中にモータがランダムな方向に少し動いてしまうという欠点があります。もう一つの欠点は、モータに対して、ロータの方向に影響を及ぼす静摩擦や重力負荷のないようにしなければならないことです。</p>
ホールセンサ基準フェージング <code>phasing_mode == 1</code>	<p>このモードでは、ある設定位置から別の位置へのホールセンサの位置信号の変化をスキャンすることによってフェージングが行われます。</p> <p>起動時、位相の原点は、定常状態のホールセンサによって定義される6つの設定可能な角度のうちの1つにセットされます。これによってモータは動作しますが、効率性は下がり、大きな「コギング」が発生します。これは、定常状態のロータの方向を決める際のホールセンサ位置精度が60度であることに起因します。</p> <p>ある設定位置から別の位置へのホールセンサの位置信号の変化が通知されると、アルゴリズムは古いホールセンサ位置と新しいホールセンサ位置の組み合わせを分析し、ロータ位置を推測します。この後、モータのステータスが「位相を調整した」状態にセットされ、モータの整流が正弦波整流へと切り替わります。</p> <p>アルゴリズムは、<code>m_phasing.c</code> ファイルの <code>hall_phasing()</code> 関数に実装されています。</p> <p>この手法の利点は、機械的負荷に対して強いことと、最初の動作開始前に不要な動きを発生させないことです。最大の欠点は、ホールセンサおよび関連の接続を必要とし、コストと信頼性において問題があることです。</p>

フェージングモード	内容
<p>デザリング基準 フェージング <code>phasing_mode == 2</code></p>	<p>デザリングアルゴリズムは、強制フェージングアルゴリズムから派生したものです。強制フェージングアルゴリズムでは、既知の磁束を一定時間印加した後にロータ位置を観測することによって、ロータ位置を特定します。</p> <p>強制フェージングアルゴリズムとは異なり、デザリングアルゴリズムは、一定時間待機せずにロータ位置の変化を監視します。動作の方向が検出されると、逆方向に動作させるために磁束の向きが変わります。磁束の角度は、動作が検出されなくなるまで徐々に小さくなります。結果として、フェージングの一部として短時間のモータの微小な振動のみを伴うモータフェージングとなります。</p> <p>このアルゴリズムは、強制フェージングアルゴリズムの利点を持ち合わせながら、不要な動作を発生させません。欠点は、モータが取り付けられている機械システムの動特性に適合させるため、アルゴリズムのパラメータを入念に調整しなければならないという点です。静摩擦と重力負荷が生じることもマイナス要素です。</p> <p>この手法は、<code>m_phasing.c</code> ファイルの <code>dither_phasing()</code> 関数に実装されています。</p>

7.6 モータのホーミング

ホーミング（原点出し）は、アブソリュートエンコーダを使用しないアプリケーションにおいて、またはアブソリュートエンコーダの位置の校正を行わない場合に必要となります。ホーミング処理は、一連の動作を実行し、外部センサからのフィードバックを読み取ることで、制御軸の座標系の原点を確立します。

ホーミング処理は、位置フィードバックおよび使用するホームセンサに応じて、以下に示す3種類のモードで行うことができます。

ホーミングモード	内容
<p>インデックス基準</p>	<p>このホーミング処理は、モータパラメータ <code>home_mask</code> を 0 にセットすると開始します。</p> <p>このホーミングモードは、デバイスが回転軸を使用し、その座標系でモータの軸方向が 0 度となる角度を特定する必要がある場合に用いられます。</p> <p>インデックスパルスは、インクリメンタルエンコーダでインデックスパルスを出力する場合のみ、ホーム参照位置として使用可能です。通常、エンコーダ信号「Z+/Z-」として定義されています。エンコーダによっては、差分インデックス出力に対応していないものもあります。この場合、コネクタ側で Z- 入力に 3.3V のバイアスをかける必要があります。</p> <p>ホーミング処理は動作を開始し、インデックスパルスが記録されると直ちに位置取得ハードウェアを作動させます。インデックス位置を取得すると、その値は、ユーザ定義のホームオフセットとともに現在位置から差し引かれます。これにより、位置オフセットが効果的に調整され、インデックス位置（およびホームオフセット）が、軸座標系の新たな原点（ゼロ）として確立されます。</p> <p>この処理が完了すると、軸の目標位置がゼロにセットされ、動作が開始します。</p>

ホーミングモード	内容
ハードストップ基準	<p>このホーミング処理は、モータパラメータ <code>home_mask</code> を 3 にセットすると開始します。</p> <p>このホーミング処理では、位置誤差を継続的にポーリングします。エンコーダカウントが 100 を超えると、現在位置が軸座標系の原点と見なされます。</p> <p>インデックス基準ホーミングと同様に、ユーザ定義のホームオフセットが現在位置から差し引かれます。この処理が完了すると、軸の目標位置がゼロにセットされ、動作が開始します。</p>
ホームセンサ基準	<p>このホーミング処理は、モータパラメータ <code>home_mask</code> を 3 を除く 0 以外の値にセットすると開始します。<code>home_mask</code> 値はデジタル入力値と AND されます。これにより、効率的に「ホームスイッチ」とされるデジタル入力を選択することが可能です。</p> <p>ホーミング処理を実行すると、ホームスイッチのステータスに応じた方向で動作が開始します。動作の目的は入力状態を変化させることにあります。ホームスイッチ開始は、座標系の原点の近傍を検出したことを示しています。</p> <p>ホームセンサ開始の検知 (100us ごとにポーリング) に遅れがあるため、ホームセンサ位置の精度は高精度アプリケーションでは不十分です。このため、ホーミングは、インデックスをベースにした処理で続行されます。</p>

以下の図に、状態および状態遷移を示します。実行フローは、上述のモータ変数 `home_mask` で定義されます。

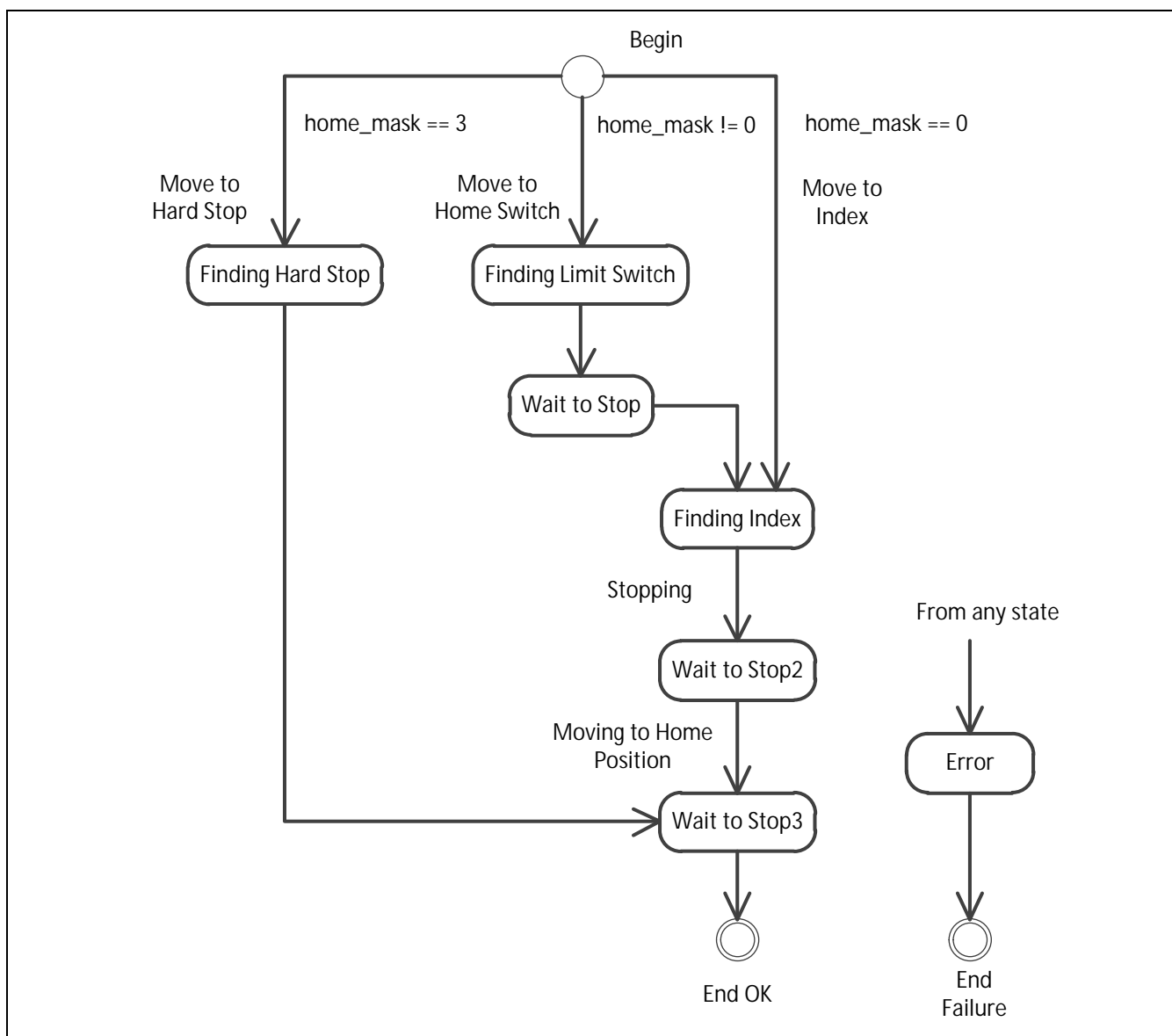


図7.2 ホーミング処理状態マシン

8. ホスト通信

8.1 ASCII通信プロトコル

ASCIIプロトコルは、復帰文字（CR、ASCII 13）で終わる ASCII 文字列で構成されるコマンドをベースとしています。コントローラはプロンプト直後のオプションデータ列を返します。

ASCIIプロトコルは、以下の通信パラメータを使用します。

115,200 bps、8 データビット、1 ストップビット、パリティ無し

コマンドを受け付けると、コマンドプロンプトには CR、改行文字（LF、ASCII 10）、大なり記号（>）が表示されます。コマンドが拒否されると、コマンドプロンプトには > の代わりに疑問符（?）が表示されま

例)

```
POS          ; CR で終わるホストコマンド
120          ; 応答データ列
>            ; 応答プロンプト
```

コマンドプロンプトに変数名を入力すると、参照先の変数の値が返されます。変数名の後にパラメータを続けて入力すると、その変数を新しい値に設定するリクエストとして認識されます。いくつかの変数は読み取り専用となっていますので、それらの変数に値を設定しようとすると、無効なコマンドとして返されま

例)

```
>POS          ; 変数 POS の値をリクエスト
2100
>POS 2000     ; POS を新しい値に設定
>POS          ; 新しい値を通知
2000
>
```

ASCII の全コマンドの一覧は、本アプリケーションノートの付録 A を参照してください。

8.2 バイナリパケット通信プロトコル

RS422 バスで複数のモーションコントローラを同じシリアルインタフェースに接続する場合、バイナリパケットプロトコルを使用します。このプロトコルを使用すると、コントローラはパケットのアドレスフィールドでパケットを識別し、自身宛てのパケットのみに応答します。もう一つの利点として、パケットの整合性を検証するチェックサムを使用できます。これにより通信の信頼性が向上するため、特にノイズの多い産業環境において有効な通信手段となります。

パケットプロトコルは半二重通信です。ホストモジュールと複数のスレーブモジュール（Biplane ボードやその他互換ボード等）は、バス型のトポロジで接続します。また、帯域幅をより有効に活用するため、可変長パケットを使用します。

パケットには固定長のヘッダと可変長のペイロードが含まれています。パケットの中身は8ビットのチェックサムで検証されます。応答パケットも個々の要求によって長さが異なり、可変長となります。応答パケットは、2バイトのステータスビットで始まり、オプションデータバイトがそれに続き、全応答バイトの合計で計算されるチェックサムバイトで終了します。

パケットプロトコルでは、最大 127 個のモジュールのアドレス指定が可能です。特定のモジュールをグループアドレスに割り当て、これらのモジュールをホストから同時にアドレス指定できます。オプションとして、あるモジュールをグループマスタとして指定し、グループの代表として応答させることも可能です。また、モジュールのアドレスやグループにかかわらず、全モジュールを宛先とするブロードキャストアドレスにも対応しています。

8.2.1 通信バス

Biplane ボードで使用するパケットプロトコルは、ホストコンピュータと、RS422 バスで接続された複数のモーション制御モジュールとの通信を効率的に行うことを目的としています。以下の図に、代表的なバス構成を示します。

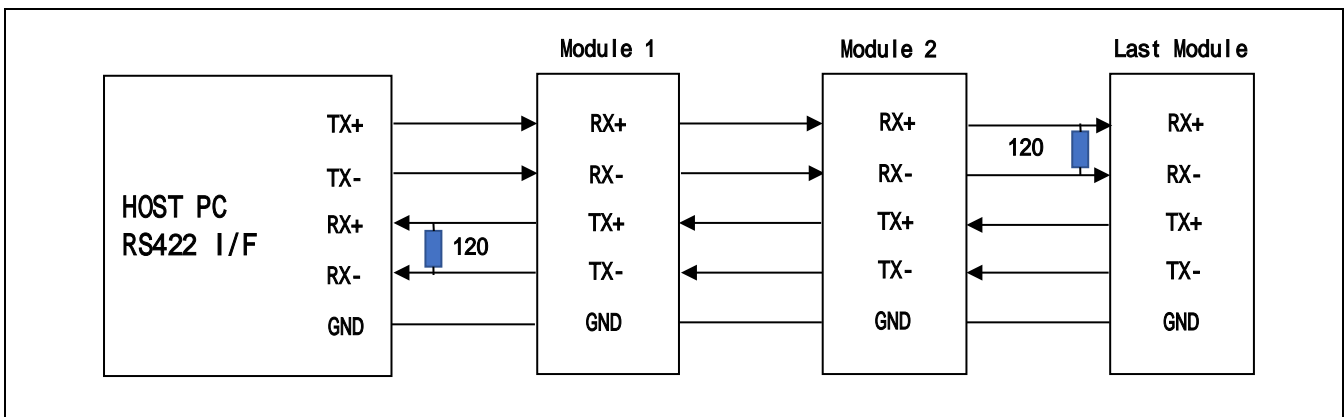


図8.1 RS422バス構成

受信側には 120 Ω の終端抵抗を推奨します。終端抵抗は、最後のモジュールに接続してください。

通信データ形式は、8 データビット、1 ストップビット、パリティ無しです。起動時の通信ボーレートは、デフォルトで 115.2Kbps となっています。この値は実行時に変更できますが、リセット後、元の値に戻りません。

8.2.2 モジュールのアドレス指定

ホストコンピュータとモジュール間の通信は半二重方式で行われます。ホストが常に要求を送信し、全モジュールが要求を聞きます。ホストに対し、アドレス指定されたモジュールが応答します。Biplane ボードのハードウェアとファームウェアは2つのモータに対応しているため、1つのモジュールが2つの異なるアドレス（各チャンネルに1アドレス）に応答することが可能です。

各モジュールに割り当てられるアドレスは予め決められており、変更できません。出荷時の設定では"Axis1"がアドレス1、"Axis2"がアドレス2として、Biplane ボードの2つのチャンネルに割り当てられています。これらのアドレスは、ボードがRS232バスに接続され、ASCIIプロトコルを使用している場合は変更可能です。新しいアドレスが割り当てられると、ファームウェアがコントローラの他のパラメータとともにそのアドレスをQSPIフラッシュに保存します。

このプロトコルでは、グループアドレスを指定できます。グループアドレスは、実行時に割り当てることが可能です。1つのモジュールをグループマスタとして指定でき、ホストがグループ全体をアドレス指定した場合に、グループマスタがグループを代表して応答します。これは、複数のモーション制御モジュール間の同期を取るのに有効です。

ブロードキャストアドレスとして255 (0xFF) が指定されており、ボーレート設定等のコマンドに有用です。モジュールはこのアドレスに送信されたパケットの処理を行いますが、応答は行いません。

アドレス指定の手順：RS232モードを使用して、ADDRコマンドおよびGROUPコマンドを使用して設定します。これらのコマンドは、SAVEコマンドを使用してQSPIフラッシュに保存可能です。

8.2.3 要求パケットの形式

一般的なパケット形式はヘッダとペイロードで構成されています。ヘッダは3バイトの固定長です。ペイロードは可変長で、ペイロードの最初のバイトであるパケットタイプによって変わります。

要求パケットのサイズは指定されています。ペイロード長はヘッダの3番目のバイトで定義されます。応答パケットのサイズは定義されていますが、要求に応じて変わります。

一般的な要求パケット：

ヘッダ			ペイロード		
開始バイト (0xAA)	アドレスバイト (ノードID)	ペイロード長 バイト (1~255)	パケットコード バイト (以下を参照)	可変長 (0~254)	チェックサム バイト
0	1	2	3	Nバイト	4+N

チェックサムでは、1番目のバイト(0xAA)を除くすべてのバイトが計算されます。

一般的な応答パケット：

応答コード	可変長	チェックサム
X	N	X+N

応答パケットのチェックサムでは、すべてのバイトが計算されます。要求パケットのチェックサムエラーは応答コード255で示されます。

要求パケットタイプのコード：

パケット タイプ	コマンド内容	要求パケット バイトサイズ	応答パケット バイトサイズ
0	Get Module Data	5	7
1	Data Report Request	7	可変
2	Function Call	6	3
3	Initialize PVT Mode	6	3
4	Stream Position / Velocity set points	5 + points * 8	3
5	Set Parameter (16-bit or 32-bit)	8または10	3
6	Get Parameter	6	5または7
7	Get Trace Buffer	6	可変
15	Set Communication Baud Rate	6	3

応答コード：

応答コード	内容
0	Command Accepted
1	Invalid Packet Code
2	Invalid Parameter Index
3	Invalid Data Size / Format
4	Parameter Value Out of Range
5	Invalid Request - Attempt to set a Read-Only variable
6	Interlock Active
7	Controller Fault
8	Invalid State
9	Buffer Overflow
10	Execution Error
255	Checksum error

各種パケット形式の詳細な説明は、本書の付録を参照してください。

8.3 EtherCATサポート



8.3.1 ソフトウェア構成

プロトコルスタックを含むソフトウェアの構成を以下に示します。

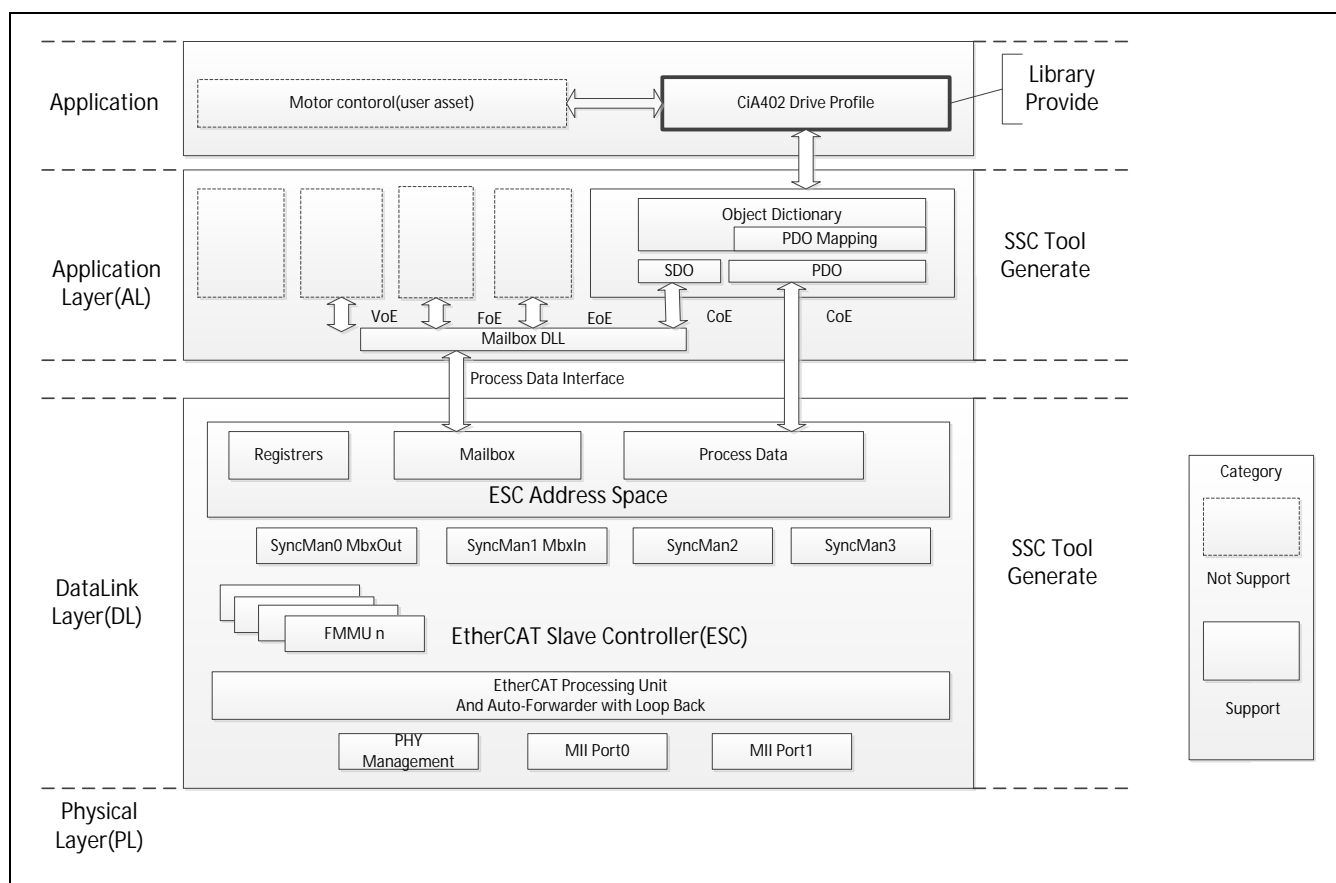


図8.2 ソフトウェア構成図

SSC ツール : Beckhoff 社製 EtherCAT スレーブサンプルコード生成ツール

ESC : R-IN エンジン内蔵の EtherCAT スレーブコントローラ

図 8.2 に示すように、EtherCAT 通信を CiA402 ドライブプロファイルの形式に対応させるため、AL 層に CoE (CAN application protocol over EtherCAT) を利用します。

サイクリック通信によるリアルタイムデータ転送時には、プロセスデータオブジェクト (PDO) を用います。PDO には、PLC 側からのデータを受信する RxPDO と、PLC に対してステータス情報等を送信する TxPDO があります。非同期でのメッセージ通信の場合には、メールボックス通信 (SDO) を利用してオブジェクトディクショナリの読み書きを行います。

8.3.2 CiA402ドライブプロファイル

CiA402 ドライブプロファイルは、ドライブおよびモーションコントロール用のデバイスプロファイルです。主にサーボドライブ、正弦波インバータ、ステッピングモータ用コントローラの各機能の動作を定義します。

このプロファイルでは、動作モードと対応するパラメータがオブジェクトディクショナリとして定義されます。また、状態ごとの内部および外部動作を規定する有限状態オートマトン (Finite State Automaton: FSA) も含まれます。状態を変更する場合はコントロールワードオブジェクトを通じて指定することで、現在の状態を示すステータスワードオブジェクトに遷移後の結果が反映されます。

コントロールワードと各種コマンド値 (目標位置など) は RxPDO に、ステータスワードと各種実査値 (実際の位置など) は TxPDO に割り当てられます。詳細については、CiA402 規格書を参照してください。

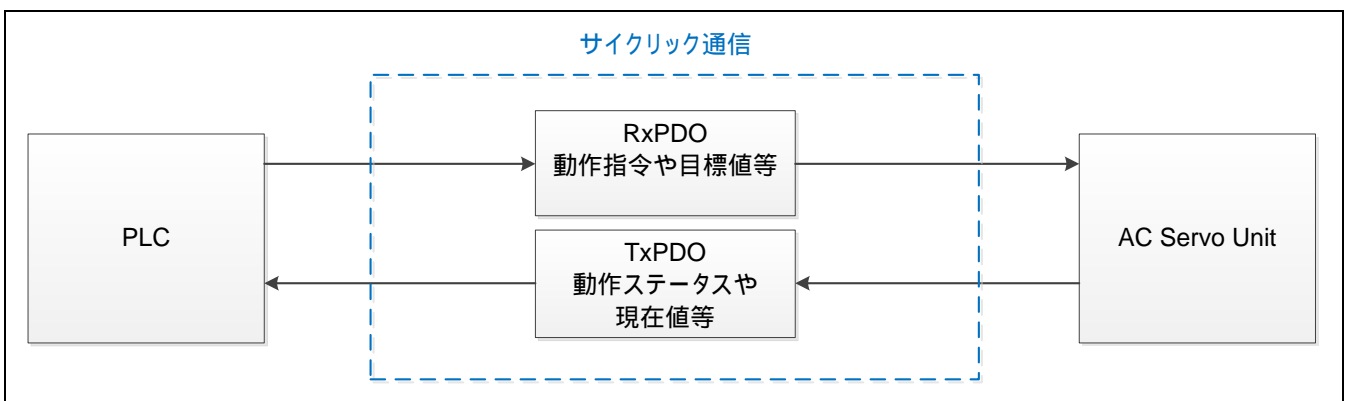


図8.3 CiA402通信フロー

CiA402 規格書 :

- ・ IEC 61800-7-201 Edition 1.0
Adjustable speed electrical power drive systems Part 7-201:
Generic interface and use of profiles for power drive systems Profile type 1 specification
- ・ IEC 61800-7-301 Edition 1.0
Adjustable speed electrical power drive systems Part 7-301:
Generic interface and use of profiles for power drive systems Mapping of profile type 1 to network technologies

8.3.2.1 動作モード

本アプリケーションノートでは、CiA402規格で規定されている動作モードのうち、以下をサポートしています。

表 8.1 対応動作モード一覧

動作モード	サポート
Profile position mode	
Velocity mode (frequency converter)	×
Profile velocity mode	○
Profile torque mode	×
Homing mode	
Interpolated position mode	×
Cyclic synchronous position mode	
Cyclic synchronous velocity mode	○
Cyclic synchronous torque mode	×
Cyclic synchronous torque mode with commutation angle	×
Manufacturer specific mode	×

8.3.2.2 状態遷移

本アプリケーションノートでは、CiA402規格で規定されているFSAをサポートしています。

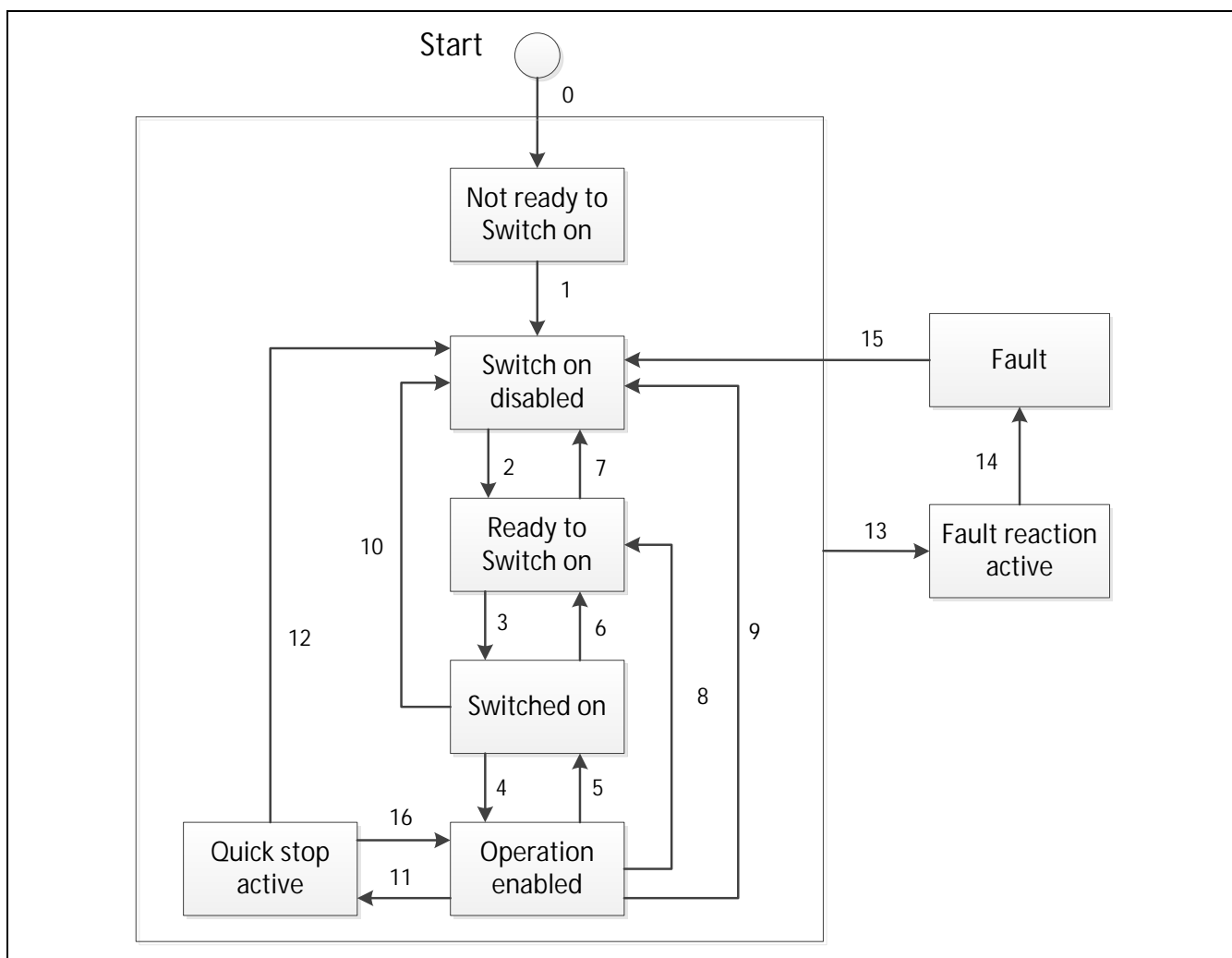


図8.4 CiA402状態遷移図

8.3.2.3 オブジェクトディクショナリ

本書でサポートしているオブジェクトディクショナリの一覧を以下に示します。

表 8.2 対応オブジェクトディクショナリ一覧 (動作モード)

動作モード	オブジェクト名	インデックス	分類	アクセス	データ型	マッピング
Profile position mode	Target position	0x607A	Mandatory	rw	INT32	RxPDO
	Profile velocity	0x6081	Mandatory	rw	UINT32	RxPDO
	Profile acceleration	0x6083	Mandatory	rw	UINT32	RxPDO
	Profile deceleration	0x6084	Optional	rw	UINT32	RxPDO
	Motion profile type	0x6086	Optional	rw	INT16	RxPDO
	Profile jerk	0x60A4	Optional	rw	UINT32	RxPDO
Profile velocity mode	Profile acceleration	0x6083	Mandatory	rw	UINT32	RxPDO
	Profile deceleration	0x6084	Optional	rw	UINT32	RxPDO
	Target velocity	0x60FF	Mandatory	rw	INT32	RxPDO
Homing mode	Home offset	0x607C	Recommended	rw	INT32	RxPDO
	Homing method	0x6098	Mandatory	rw	INT8	RxPDO
	Homing speeds	0x6099	Optional	rw	UINT32	RxPDO
	Homing acceleration	0x609A	Optional	rw	UINT32	RxPDO
Cyclic synchronous position mode	Torque actual value	0x6077	Recommended	ro	INT16	TxPDO
	Target position	0x607A	Mandatory	rw	INT32	RxPDO
	Motion profile type	0x6086	Optional	rw	INT16	RxPDO
	Profile jerk	0x60A4	Optional	rw	UINT32	RxPDO
	Velocity offset	0x60B1	Optional	rw	INT32	RxPDO
	Torque offset	0x60B2	Recommended	rw	INT16	RxPDO
Cyclic synchronous velocity mode	Torque actual value	0x6077	Recommended	ro	INT16	TxPDO
	Torque offset	0x60B2	Recommended	rw	INT16	RxPDO
	Target velocity	0x60FF	Mandatory	rw	INT32	RxPDO
All	Error code	0x603F	Optional	ro	UINT16	TxPDO
	Controlword	0x6040	Mandatory	rw	UINT16	RxPDO
	Statusword	0x6041	Mandatory	ro	UINT16	TxPDO
	Modes of operation	0x6060	Optional	rw	INT8	SDO
	Modes of operation display	0x6061	Optional	ro	INT8	TxPDO
	Position actual value	0x6064	Mandatory	ro	INT32	TxPDO
	Velocity actual value	0x606C	Conditional	ro	INT32	TxPDO

表 8.2 対応オブジェクトディクショナリー一覧 (その他オブジェクト)

その他オブジェクト	オブジェクト名	インデックス	分類	アクセス	データ型	マッピング
	Quick stop option code	0x605A	Optional	rw	INT16	No
	Shutdown option code	0x605B	Optional	rw	INT16	No
	Disable operation option code	0x605C	Optional	rw	INT16	No
	Fault reaction option code	0x605E	Optional	rw	INT16	No
	Following error window	0x6065	Optional	rw	UINT32	No
	Following error time out	0x6066	Optional	rw	UNIT32	No
	Position range limit	0x607B	Optional	rw	INT32	No
	Software position limit	0x607D	Optional	c,rw	INT32	No
	Max profile velocity	0x607F	Optional	rw	UINT32	No
	Max motor speed	0x6080	Optional	rw	UINT32	No
	Quick stop declaration	0x6085	Optional	rw	UINT32	No
	Position offset	0x60B0	Optional	rw	INT32	No
	Interpolation time period	0x60C2	Conditional	c,rw	Record	No
	Following error actual value	0x60F4	Optional	ro	INT32	No
	Motor type	0x6402	Optional	rw	INT16	No
	Supported drive modes	0x6502	Mandatory	ro	INT32	No

8.3.3 アプリケーションの開発手順

CiA402 サンプルソフトウェアを使用したアプリケーションの開発手順について以下に説明します。

8.3.3.1 SSCツールによるサンプルプロトコルスタック

(1) SSCツールによるサンプルソフトウェアの生成

サンプルソフトウェアのSSCプロジェクトファイルをダブルクリックし、SSCツールを起動します。

\\Cortex-M3\Device\Renesas\RIN_Engine\Source\Project_Dual\CiA402_Motion_Control\RenesasSDK\ssc_project
\\RZT1_CiA402_Motion_Control.esp

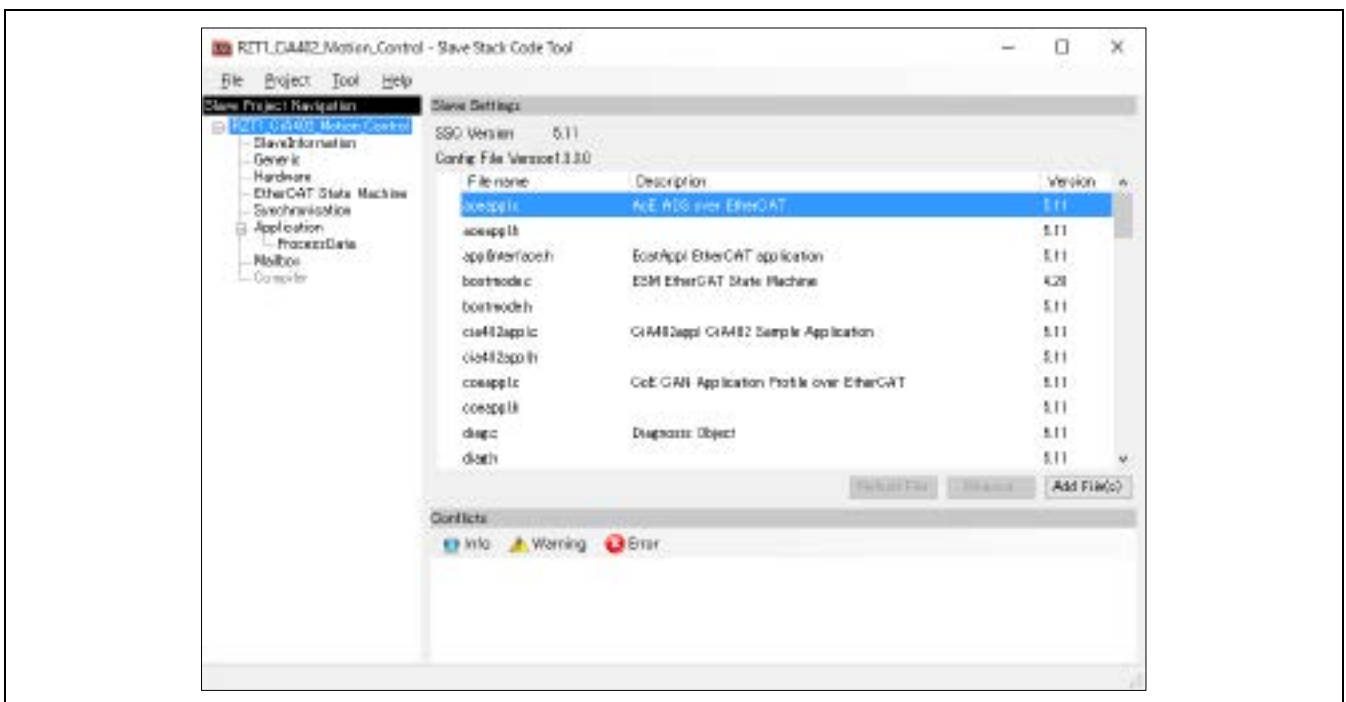


図8.5 SSCツール画面

注1. 本アプリケーションノートでは、SSCツールのバージョン5.11を使用します。

- (2) [Project] > [Create new Slave Files] > [start] > [OK]を選択します。
- (3) SSCツールはサンプルソースファイルを以下のフォルダに生成します。

“../Src”

(4) パッチコマンドの準備

GNU Patch Ver2.5.9以降が必要です。

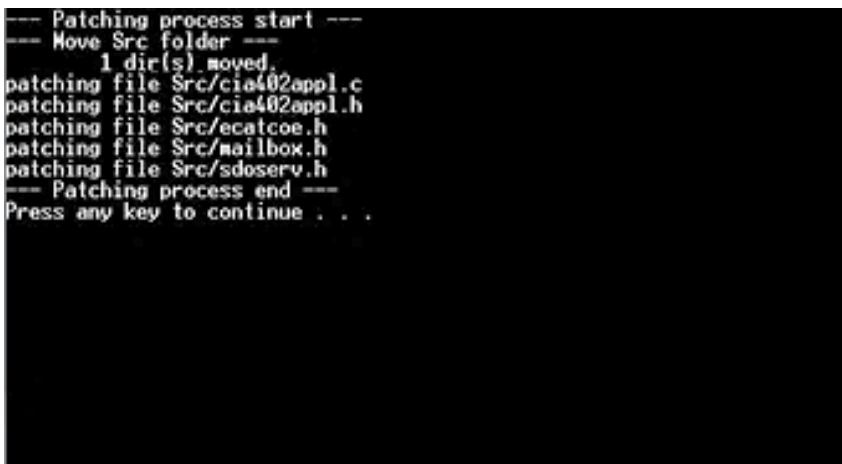
パッチコマンド (Ver2.5.9) を以下のウェブサイトからダウンロードし、ディレクトリパスに
“ patch.exe ” を保存します。

<http://gnuwin32.sourceforge.net/packages/patch.htm>

- (5) “ apply_patch.bat ” を右クリックし、“ Run as administrator ” を選択します。

パッチファイルは、SSC ソースファイルに対する修正を含んでいます。

```
\Cortex-M3\Device\Renesas\RIN_Engine\Source\Project_Dual\CiA402_Motion_Control\RenesasSDK\  
apply_patch.bat
```



```
--- Patching process start ---  
--- Move Src folder ---  
1 dir(s) moved.  
patching file Src/cia402appl.c  
patching file Src/cia402appl.h  
patching file Src/ecatcoe.h  
patching file Src/mailbox.h  
patching file Src/sdoserv.h  
--- Patching process end ---  
Press any key to continue . . .
```

図8.6 パッチコマンド

8.3.3.2 モータ制御用プログラムの実装

以下の表 8.3 および CiA402 規格に従って、モータ制御アプリケーションの実装を行ってください。各関数は、図 8.4 CiA402 状態遷移図に示す各状態遷移の番号とリンクしており、状態遷移が発生した際に対応する関数が呼び出されます。

各関数内では、モータ制御プログラムまたは Main CPU 側の当該処理を呼び出す処理を記述してください。

表 8.3 CiA402 プロトコルスタック I/F 関数一覧

CiA402_StateTransition1	
<u>Description</u>	CiA402FSAにて規定されている状態遷移1が発生した際に使用します。状態遷移が発生した際に行う処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに適切な値を設定して関数を終了してください。戻り値に1を設定した場合は状態遷移は行われません。
CiA402_StateTransition2	
<u>Description</u>	CiA402FSAにて規定されている状態遷移2が発生した際に使用します。状態遷移が発生した際に行う処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに適切な値を設定して関数を終了してください。戻り値に1を設定した場合は状態遷移は行われません。

CiA402_StateTransition3	
<u>Description</u>	CiA402FSAにて規定されている状態遷移3が発生した際に使用します。状態遷移が発生した際に実行する処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに適切な値を設定して関数を終了してください。戻り値に1を設定した場合は状態遷移は行われません。
CiA402_StateTransition4	
<u>Description</u>	CiA402FSAにて規定されている状態遷移4が発生した際に使用します。状態遷移が発生した際に実行する処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに適切な値を設定して関数を終了してください。戻り値に1を設定した場合は状態遷移は行われません。
CiA402_StateTransition5	
<u>Description</u>	CiA402FSAにて規定されている状態遷移5が発生した際に使用します。状態遷移が発生した際に実行する処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに適切な値を設定して関数を終了してください。戻り値に1を設定した場合は状態遷移は行われません。

CiA402_StateTransition6	
<u>Description</u>	CiA402FSAにて規定されている状態遷移6が発生した際に使用します。状態遷移が発生した際に実行する処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに適切な値を設定して関数を終了してください。戻り値に1を設定した場合は状態遷移は行われません。
CiA402_StateTransition7	
<u>Description</u>	CiA402FSAにて規定されている状態遷移7が発生した際に使用します。状態遷移が発生した際に実行する処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに適切な値を設定して関数を終了してください。戻り値に1を設定した場合は状態遷移は行われません。
CiA402_StateTransition8	
<u>Description</u>	CiA402FSAにて規定されている状態遷移8が発生した際に使用します。状態遷移が発生した際に実行する処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに適切な値を設定して関数を終了してください。戻り値に1を設定した場合は状態遷移は行われません。

CiA402_StateTransition9	
<u>Description</u>	CiA402FSAにて規定されている状態遷移9が発生した際に使用します。 状態遷移が発生した際に実行する処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに 適切な値を設定して関数を終了してください。戻り値に1を設定した 場合は状態遷移は行われません。
CiA402_StateTransition10	
<u>Description</u>	CiA402FSAにて規定されている状態遷移10が発生した際に使用します。 状態遷移が発生した際に実行する処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに 適切な値を設定して関数を終了してください。戻り値に1を設定した 場合は状態遷移は行われません。
CiA402_StateTransition11	
<u>Description</u>	CiA402FSAにて規定されている状態遷移11が発生した際に使用します。 状態遷移が発生した際に実行する処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに 適切な値を設定して関数を終了してください。戻り値に1を設定した 場合は状態遷移は行われません。

CiA402_StateTransition12	
<u>Description</u>	CiA402FSAにて規定されている状態遷移12が発生した際に使用します。状態遷移が発生した際に実行する処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに適切な値を設定して関数を終了してください。戻り値に1を設定した場合は状態遷移は行われません。
CiA402_LocalError	
<u>Description</u>	CiA402FSAにて規定されている状態遷移13が発生した際に使用します。状態遷移が発生した際に実行する処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	UINT16 ErrorCode
<u>Return Value</u>	なし
<u>Remark</u>	状態遷移13に相当するエラーが発生した場合はエラー発生箇所にて必要な処理、データの保存等を行ったうえで本関数をコールしてください。
CiA402_StateTransition14	
<u>Description</u>	CiA402FSAにて規定されている状態遷移14が発生した際に使用します。状態遷移が発生した際に実行する処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに適切な値を設定して関数を終了してください。戻り値に1を設定した場合は状態遷移は行われません。

CiA402_StateTransition15	
<u>Description</u>	CiA402FSAにて規定されている状態遷移15が発生した際に使用します。状態遷移が発生した際に実行する処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに適切な値を設定して関数を終了してください。戻り値に1を設定した場合は状態遷移は行われません。
CiA402_StateTransition16	
<u>Description</u>	CiA402FSAにて規定されている状態遷移16が発生した際に使用します。状態遷移が発生した際に実行する処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	処理中に異常が発生した場合はCiA402規格に従って、各オブジェクトに適切な値を設定して関数を終了してください。戻り値に1を設定した場合は状態遷移は行われません。
APPL_MOTOR_MotionControl_Main	
<u>Description</u>	CiA402 FSAの状態が"Operation enabled"であるときに実行するモーション制御用コードを記述します。動作モード毎に処理を記述してください。
<u>Usage</u>	#include "cia402appl.h"
<u>Parameters</u>	TCiA402Axis *pCiA402Axis
<u>Return Value</u>	0 正常終了 1 エラー終了
<u>Remark</u>	初期状態ではmain.cの中に配置されており、CiA402_DummyMotionControl関数を呼び出す形になっています。この関数を参考に処理を記述してください。

8.3.3.3 プロトコルスタック実装時の注意事項

- 注1. メールボックス使用時のアライメントについて
CiA402プロトコルスタックでは、PDO通信の他にSDO通信を使用します。SDO通信で使用する各構造体に対しては、“pack”オプション等を使用してアライメントが保証されるようにしてください。
- 注2. EtherCAT用のウォッチドッグタイマについて
マスタ機器に“TwinCAT + PC”を選択した場合、SMイベントの周期のリアルタイム性が確保できないことが原因でウォッチドッグタイマエラーが発生する可能性があります。
本アプリケーションのサンプルソフトは、SSCツールのウォッチドッグタイマ設定を無効にした状態で生成したソースコードを使用し、動作確認を行っています。

8.3.4 共有メモリ

8.3.4.1 動作概要

Cortex-R4 と Cortex-M3 のユーザアプリケーションがデータを共有する場合、アプリケーション間でアクセスの競合が発生する可能性があります。本ドライバは、セマフォレジスタを用いて共有メモリなどのリソースの使用状況を監視することで、排他アクセス制御を実現します。

監視を行うため、ドライバはセマフォレジスタを読み出します。各セマフォレジスタ内のセマフォビットは、対象リソースが利用可能かどうかを表示します。リードクリア機能を有効にした状態で、対象リソースが空き状態のときに対応するレジスタが読み出されると、そのレジスタのセマフォビットが0にクリアされ、リソースが使用中であることを示します。

表 8.4 セマフォレジスタ n (SYTSEMF_n : n = 0 ~ 7)

ビット	シンボル	ビット名	説明	R/W
b0	SEMF _n	セマフォビット	0 : リソースは使用中 1 : リソースは空き状態	R/W*1
b31-b1	Reserved	予約ビット	—	R/W

注 1. 本ビットに 1 を書くと値が保持されます。リードクリア機能有効時に本ビットを読むと、値は 0 にクリアされます。

8.3.4.2 ソフトウェア動作

以下に、共有メモリドライバによる、セマフォレジスタを使用したセマフォ制御の手順を説明します。

- (1) 一方のコアが、必要とするリソースに対応するセマフォレジスタを読み出します。セマフォビットの値が“0”の場合、“1”になるまでレジスタを読み出し続けます。
- (2) “1”を読み出し、対象リソースが空き状態であることを確認します。この時、セマフォビットは“0”にクリアされ、当該コアがリソースを使用中であることが表示されます。
- (3) 共有メモリにアクセスします。
- (4) アクセスが終了したら、該当ビットに“1”を書きこみ、そのリソースが空き状態であることを表示します。
- (5) 図8.7の(1)、(2)、(3)のアクセスの場合、リソースは空き状態で、共有メモリへのアクセスが直ちに行われます。(4)のアクセスでは、先行する(3)のアクセスが進行中です。(4)でアクセス要求中のコアは、(3)のアクセスが終了し、リソースが空き状態になるまで待機します。アクセスの競合はこのようにして解消されます。

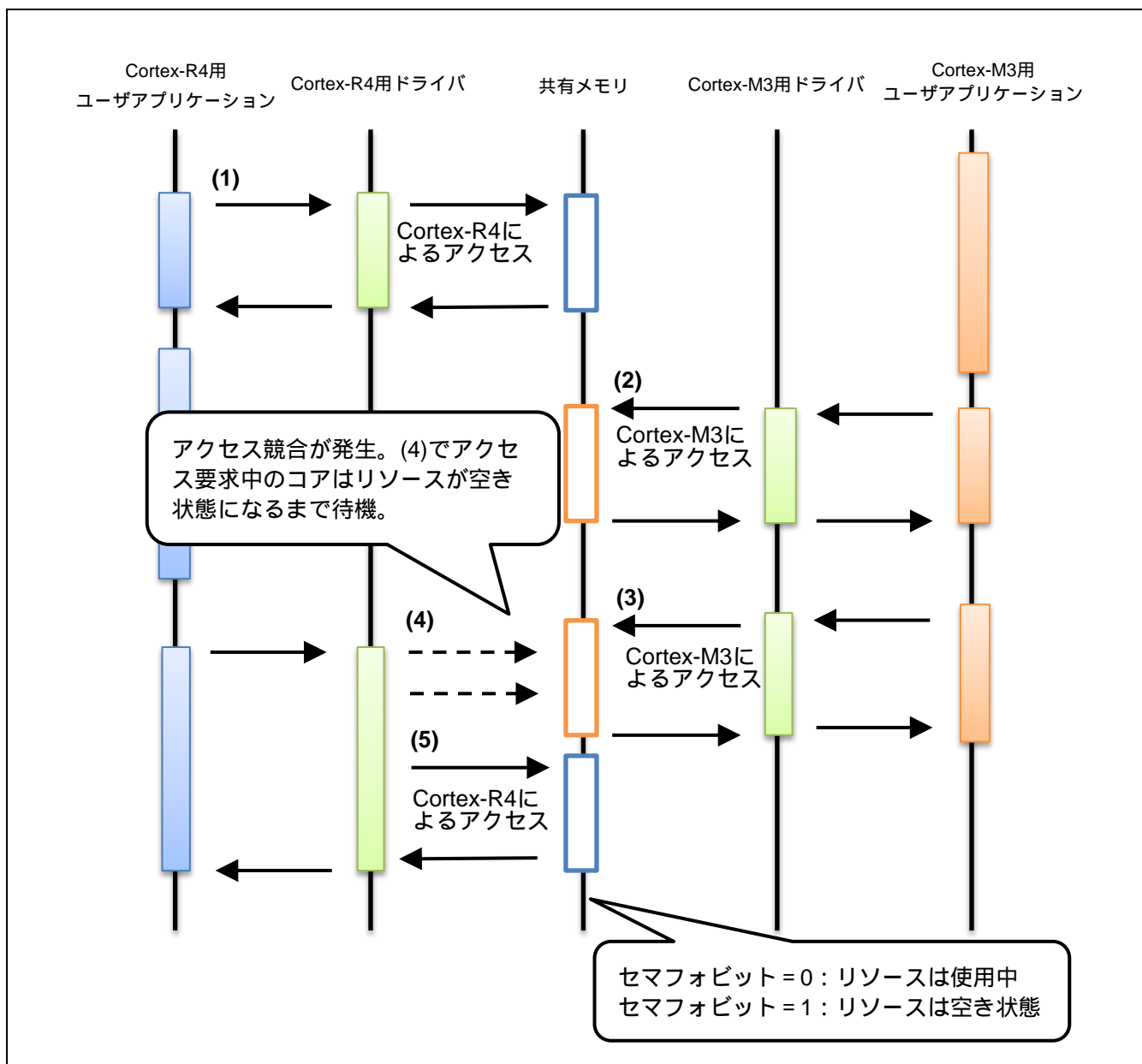


図8.7 ソフトウェア動作シーケンス

8.3.4.3 ソフトウェアブロック図

以下に、共有メモリドライバのシステムブロック図を示します。赤い点線で囲まれた領域が共有メモリドライバで行う動作です。

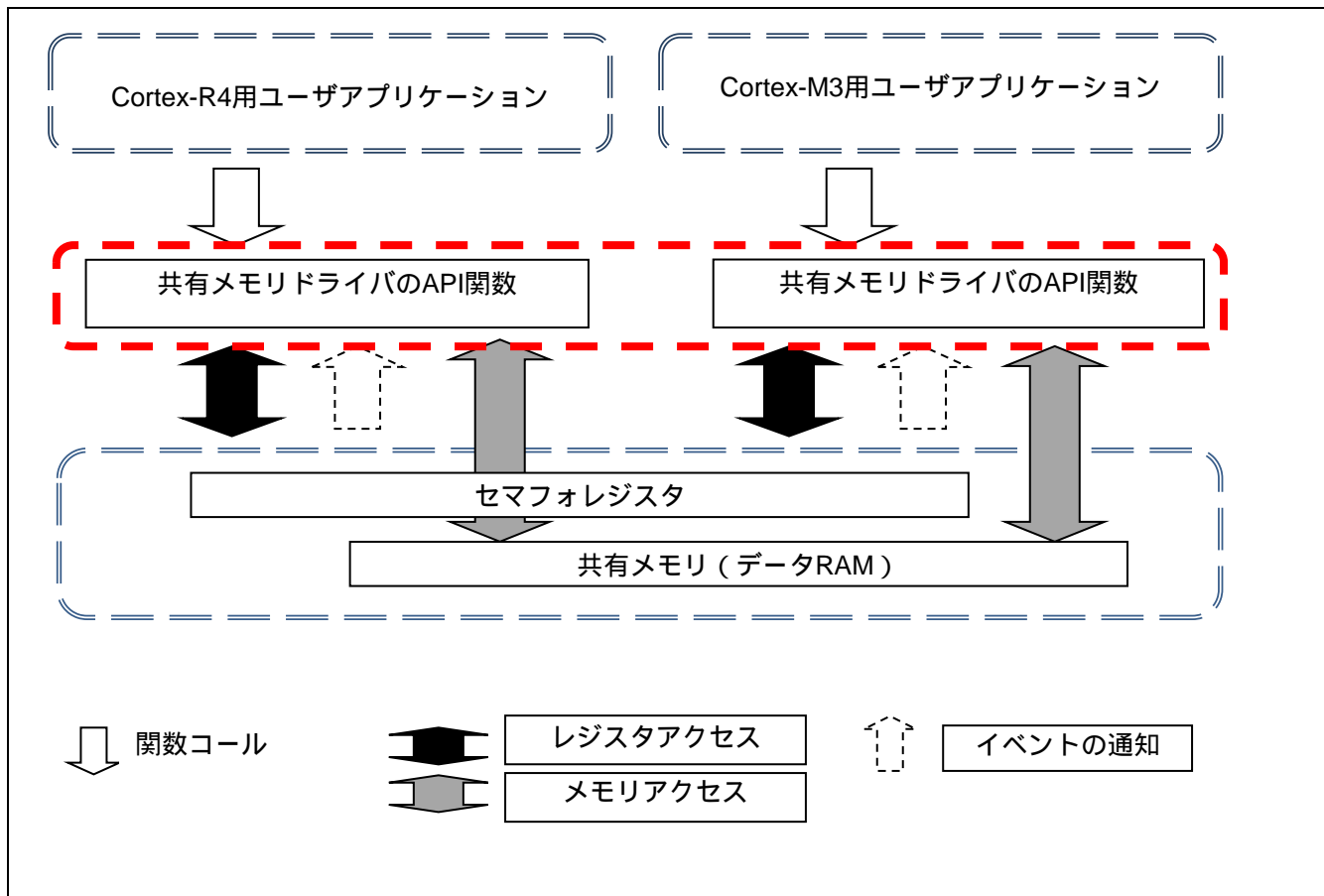


図8.8 共有メモリドライバシステム

8.3.4.4 ソフトウェア状態遷移

共有メモリドライバの状態遷移を、以下の図に示します。

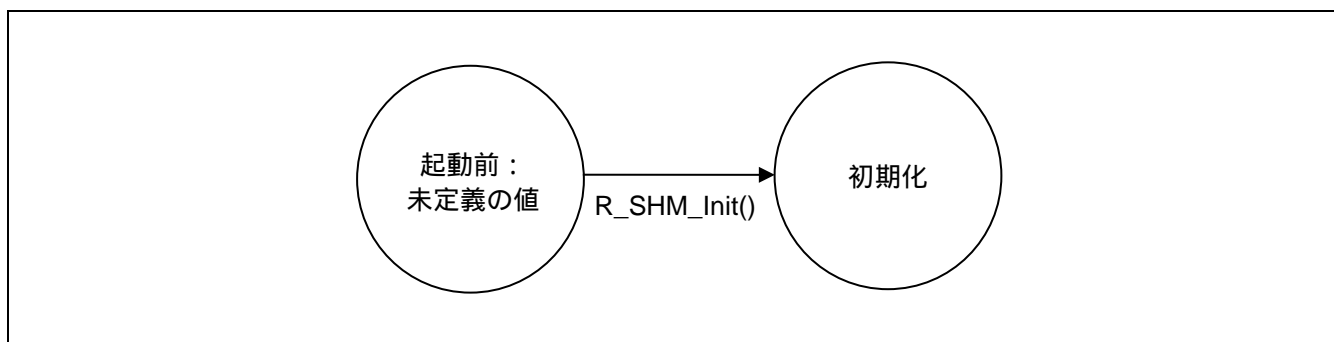


図8.9 共有メモリドライバの状態遷移

8.3.4.5 使用メモリサイズ

共有メモリ領域を、以下の図に示します。このドライバでは、データ RAM の先頭の 4KB が共有メモリの予約領域です。

Cortex-R4 は、Cortex-R4 用ドライバのコード領域、ドライバの管理情報を格納するデータ領域、スタック領域を TCM に確保しています。Cortex-M3 は、Cortex-M3 用ドライバのコード領域、ドライバの管理情報を格納するデータ領域、スタック領域を共有メモリ領域を除くデータ RAM 内に確保しています。



図8.10 共有メモリ領域

8.3.4.6 ファイル構成

Cortex-R4 および Cortex-M3 の各ドライバで使用するファイルを以下の表に示します。

表 8.5 Cortex-R4 ドライバで使用するファイル

ファイル名	内容	備考
/Cortex-R4/inc/common/iodefine.h	RZ/T1 で使用するレジスタを定義するヘッダファイル	共通ヘッダファイル
/Cortex-R4/inc/shm/r_shm_config.h	ドライバ構造を定義するヘッダファイル	
/Cortex-R4/inc/shm/r_shm.h	ドライバ用公開ヘッダファイル	
/Cortex-R4/src/drv/shm/r_shm.c	ドライバ用 API の実装に使用するファイル	

表 8.6 Cortex-M3 ドライバで使用するファイル

ファイル名	内容	備考
*1/Include/iodefine.h	RZ/T1 で使用するレジスタを定義するヘッダファイル	共通ヘッダファイル
*1/Include/RIN_Engine.h	R-IN Engine 割り込みを定義するヘッダファイル	共通ヘッダファイル
*1/Source/Project_Dual/CiA402_Motion_Control/RenesasSDK/shm/inc/r_shm_config.h	ドライバ構造を定義するヘッダファイル	
*1/Source/Project_Dual/CiA402_Motion_Control/RenesasSDK/shm/inc/r_shm.h	ドライバ用公開ヘッダファイル	
*1/Source/Project_Dual/CiA402_Motion_Control/RenesasSDK/shm/src/r_shm.c	ドライバ用 API の実装に使用するファイル	
*1/Source/Project_Dual/CiA402_Motion_Control/IAR/iram_with_shm.icf	IAR 環境のリンク設定ファイル	共有メモリのサイズを定義するファイル

注. “..*1” : /Cortex-M3/Device/Renesas/RIN_Engine

8.3.4.7 ソフトウェアリソース

本ドライバで使用するソフトウェアリソースはありません。

8.3.4.8 プリミティブデータ型

共有メモリドライバで使用するプリミティブデータ型の一覧を以下に示します。

表 8.7 プリミティブデータ型

シンボル	内容
int8_t	8ビット符号付き整数（標準ライブラリにて定義）
int16_t	16ビット符号付き整数（標準ライブラリにて定義）
int32_t	32ビット符号付き整数（標準ライブラリにて定義）
int64_t	64ビット符号付き整数（標準ライブラリにて定義）
uint8_t	8ビット符号なし整数（標準ライブラリにて定義）
uint16_t	16ビット符号なし整数（標準ライブラリにて定義）
uint32_t	32ビット符号なし整数（標準ライブラリにて定義）
uint64_t	64ビット符号なし整数（標準ライブラリにて定義）

8.3.4.9 データ構造の定義

データ構造を以下の表に示します。

表 8.8 データ構造

構造の定義	内容	定義ファイル
st_shm structure	共有メモリはこの構造体に格納されます。	r_shm_config.h

8.3.4.10 グローバル変数

グローバル変数を以下の表に示します。

表 8.9 グローバル変数

変数	変数	内容
st_shm structure	SHM	共有メモリはこの変数（構造体）に格納されます。

8.3.4.11 値（マクロ）の定義

マクロの一覧を以下の表に示します。

表 8.10 定数マクロ

マクロ名	値	内容	定義ファイル
SHM_SEMFNO_0	0	セマフォレジスタ 0	r_shm.h
SHM_SEMFNO_1	1	セマフォレジスタ 1	r_shm.h
SHM_SEMFNO_2	2	セマフォレジスタ 2	r_shm.h
SHM_SEMFNO_3	3	セマフォレジスタ 3	r_shm.h
SHM_SEMFNO_4	4	セマフォレジスタ 4	r_shm.h
SHM_SEMFNO_5	5	セマフォレジスタ 5	r_shm.h
SHM_SEMFNO_6	6	セマフォレジスタ 6	r_shm.h
SHM_SEMFNO_7	7	セマフォレジスタ 7	r_shm.h
SHM_SEMFNO_TOTAL	8	指定するセマフォレジスタの最大値	r_shm.h
SHM_SYS_SEMFNO	0	ユーザ定義のセマフォレジスタ番号。 セマフォレジスタ番号はこの定数で指定します。	r_shm.h

8.3.4.12 値（列挙型）の定義

共有メモリドライバでは列挙型は使用しません。

8.3.4.13 エラーコード

共有メモリドライバでは、正常終了時は 0 を、エラー発生時には 0 以外の値が返されます。エラーコードを以下の表に示します。

表 8.11 エラーコード

マクロ名	値	意味	定義ファイル
SHM_SUCCESS	0	正常終了	r_shm.h
SHM_ERR	-1	エラー終了	r_shm.h

8.3.4.14 関数

接続ドライバ用共有メモリドライバのインタフェース関数の一覧を以下の表に示します。

表 8.12 共有メモリドライバ用 API 関数

関数名	意味	定義ファイル
R_SHM_Init	共有メモリドライバの初期化	r_shm.h
R_SHM_memcpy	共有メモリから n バイトデータをロード	r_shm.h
R_SHM_Load_uint32	共有メモリから uint 4 バイトデータをロード	r_shm.h
R_SHM_Load_int32	共有メモリから int 4 バイトデータをロード	r_shm.h
R_SHM_Load_uint16	共有メモリから uint 2 バイトデータをロード	r_shm.h
R_SHM_Load_int16	共有メモリから int 2 バイトデータをロード	r_shm.h
R_SHM_Load_uint8	共有メモリから uint 1 バイトデータをロード	r_shm.h
R_SHM_Load_int8	共有メモリから int 1 バイトデータをロード	r_shm.h

8.3.4.15 共有メモリドライバ定義の構造体

共有メモリドライバで定義し、ユーザプログラムで使用する構造体を以下の表に示します。

表 8.13 st_shm 構造体

メンバの型と名前	通常値の範囲	用途 / 備考
任意のメンバ	任意の値	構造体のメンバは、アプリケーションに応じて定義する必要があります。

8.3.4.16 関数リファレンス

(1) R_SHM_Init

R_SHM_Init	
概要	共有メモリドライバ初期化処理
ヘッダ	r_shm.h r_shm_config.h
宣言	int32_t R_SHM_Init(uint32_t semfno)
引数	uint32_t semfno : 使用するセマフォレジスタ番号 定数SHM_SYS_SEMFNOを使用
リターン値	SHM_SUCCESS : 初期化成功 SHM_ERR : 初期化失敗 セマフォレジスタ番号が規定範囲外
実行条件	・ 共有メモリドライバの他のすべてのAPI関数を使用する前に実行します。
説明	本関数は以下の処理を行います。 § 引数semfnoで番号を指定したレジスタのアドレスをSEMFNO変数に格納。 § Cortex-R4では、指定したセマフォレジスタのリードクリア機能を有効にし、対象リソースが空き状態であることを確認。
エラー条件	・ semfnoがSHM_SEMFNO_TOTAL以下である場合
補足	なし
使用例	#define SHM_SYS_SEMFNO (0) int32_t errcd; errcd = R_SHM_Init(SHM_SYS_SEMFNO);

(2) R_SHM_memcpy

R_SHM_memcpy	
概要	共有メモリ領域用nバイトデータロード処理
ヘッダ	r_shm.h r_shm_config.h
宣言	int32_t R_SHM_memcpy(void *dst, void *src, size_t size)
引数	void *dst : 転送先の先頭アドレス void *src : 転送元の先頭アドレス size_t size : 転送バイト数
リターン値	SHM_SUCCESS : 初期化成功
実行条件	・ 初期化関数R_SHM_IIint()の実行
説明	nバイトのデータを共有メモリからローカルメモリへ転送します。
エラー条件	なし
補足	なし
使用例	int32_t errcd; uint32_t Counter; /* SHM <- Local */ errcd = R_SHM_memcpy((void *)&SHM.Counter, (void *)&Counter, sizeof(Counter));

(3) R_SHM_Load_uint32

R_SHM_Load_uint32	
概要	共有メモリ領域用uint 4バイトデータロード処理
ヘッダ	r_shm.h r_shm_config.h
宣言	int32_t R_SHM_Load_uint32(uint32_t *dst, uint32_t *src)
引数	uint32_t *dst 転送先アドレス uint32_t *src 転送元アドレス
リターン値	SHM_SUCCESS : 初期化成功
実行条件	・ 初期化関数R_SHM_IIint()の実行
説明	uint 4バイトデータを共有メモリからローカルメモリへ転送します。
エラー条件	なし
補足	なし
使用例	int32_t errcd; uint32_t Counter; /* SHM <- Local */ errcd = R_SHM_Load_uint32((uint32_t *)&SHM.Counter, &Counter);

(4) R_SHM_Load_int32

R_SHM_Load_int32	
概要	共有メモリ領域用int 4バイトデータロード処理
ヘッダ	r_shm.h r_shm_config.h
宣言	int32_t R_SHM_Load_int32(int32_t *dst, int32_t *src)
引数	int32_t *dst 転送先のアドレス int32_t *src 転送元のアドレス
リターン値	SHM_SUCCESS : 初期化成功
実行条件	・ 初期化関数R_SHM_IIint()の実行
説明	int 4バイトデータを共有メモリからローカルメモリへ転送します。
エラー条件	なし
補足	なし
使用例	int32_t errcd; int32_t Counter; /* Local <- SHM */ errcd = R_SHM_Load_int32(&Counter, (int32_t *)&SHM.Counter);

(5) R_SHM_Load_uint16

R_SHM_Load_uint16	
概要	共有メモリ領域用uint 2バイトデータロード処理
ヘッダ	r_shm.h r_shm_config.h
宣言	int32_t R_SHM_Load_uint16(uint16_t *dst, uint16_t *src)
引数	uint16_t *dst 転送先アドレス uint16_t *src 転送元アドレス
リターン値	SHM_SUCCESS : 初期化成功
実行条件	・ 初期化関数R_SHM_llint()の実行
説明	uint 2バイトデータを共有メモリからローカルメモリへ転送します。
エラー条件	なし
補足	なし
使用例	int32_t errcd; uint16_t Counter; /* SHM <- Local */ errcd = R_SHM_Load_uint16((uint16_t *)&SHM.Counter, &Counter);

(6) R_SHM_Load_int16

R_SHM_Load_int16	
概要	共有メモリ領域用int 2バイトデータロード処理
ヘッダ	r_shm.h r_shm_config.h
宣言	int32_t R_SHM_Load_int16(int16_t *dst, int16_t *src)
引数	int16_t *dst 転送先アドレス int16_t *src 転送元アドレス
リターン値	SHM_SUCCESS : 初期化成功
実行条件	・ 初期化関数R_SHM_llint()の実行
説明	int 2バイトデータを共有メモリからローカルメモリへ転送します。
エラー条件	なし
補足	なし
使用例	int32_t errcd; int16_t Counter; /* Local <- SHM */ errcd = R_SHM_Load_int16(&Counter, (int16_t *)&SHM.Counter);

(7) R_SHM_Load_uint8

R_SHM_Load_uint8	
概要	共有メモリ領域用uint 1バイトデータロード処理
ヘッダ	r_shm.h r_shm_config.h
宣言	int32_t R_SHM_Load_uint8(uint8_t *dst, uint8_t *src)
引数	uint8_t *dst 転送先アドレス uint8_t *src 転送元アドレス
リターン値	SHM_SUCCESS : 初期化成功
実行条件	・ 初期化関数R_SHM_llint()の実行
説明	uint 1バイトデータを共有メモリからローカルメモリへ転送します。
エラー条件	なし
補足	なし
使用例	int32_t errcd; uint8_t Counter; /* SHM <- Local */ errcd = R_SHM_Load_uint8((uint8_t *)&SHM.Counter, &Counter);

(8) R_SHM_Load_int8

R_SHM_Load_int8	
概要	共有メモリ領域用int 1バイトデータロード処理
ヘッダ	r_shm.h r_shm_config.h
宣言	int32_t R_SHM_Load_int8(int8_t *dst, int8_t *src)
引数	int8_t *dst 転送先アドレス int8_t *src 転送元アドレス
リターン値	SHM_SUCCESS : 初期化成功
実行条件	・ 初期化関数R_SHM_llint()の実行
説明	int 1バイトデータを共有メモリからローカルメモリへ転送します。
エラー条件	なし
補足	なし
使用例	int32_t errcd; int8_t Counter; /* Local <- SHM */ errcd = R_SHM_Load_int8(&Counter, (int8_t *)&SHM.Counter);

8.3.4.17 フローチャート

(1) ドライバ初期化処理

以下に、ドライバの初期化処理のフローチャートを示します。

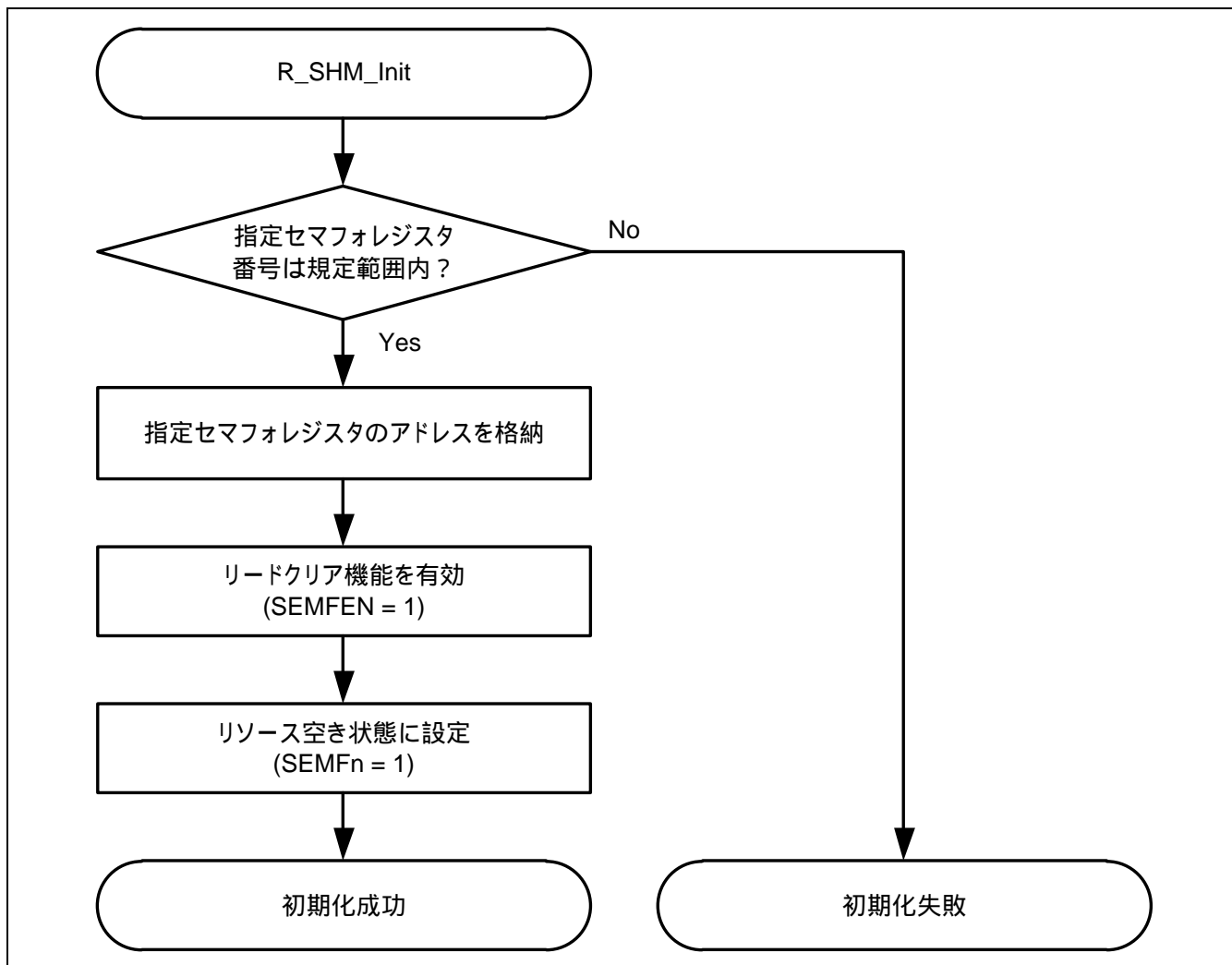


図8.11 ドライバ初期化処理

(2) 共有メモリ領域用nバイトデータロード処理

以下に、共有メモリ領域用 n バイトデータロード処理のフローチャートを示します。

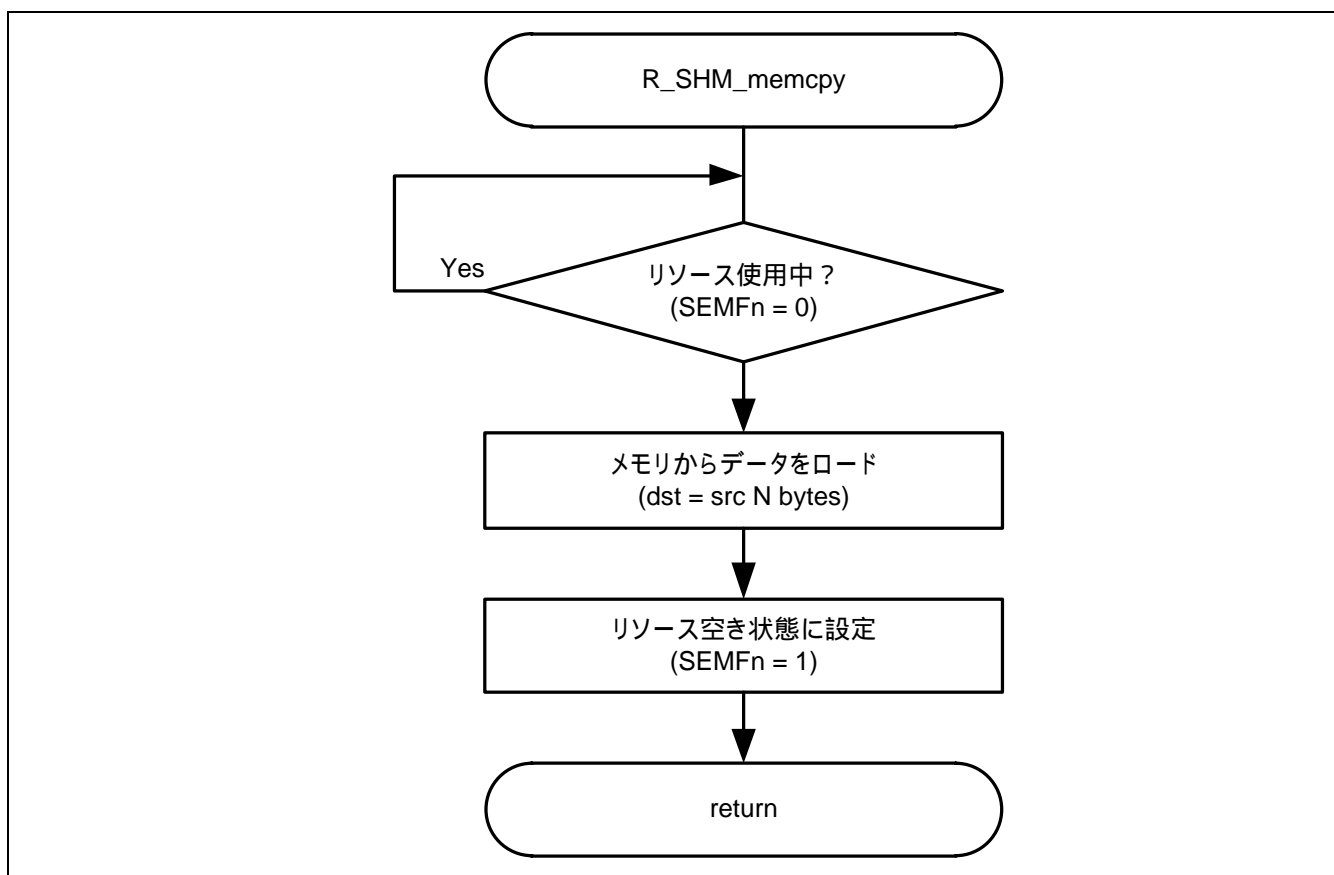


図8.12 共有メモリ領域用nバイトデータロード処理

その他データロード処理 (R_SHM_Load_uint32 等) については、上記フローチャートと同じです。

8.3.4.18 使用例

ユーザアプリケーションを作成する前に、共有メモリドライバで使用するセマフォレジスタ番号および共有メモリ構造体のメンバを定義してください。

- ・ ヘッダファイル (r_shm_config.h) を使用してドライバ構造体を定義
- ・ 同じヘッダファイルを使用してCortex-R4/Cortex-M3用ドライバ構造体を定義
- ・ ヘッダファイルの場所は「8.3.4.6 ファイル構成」で説明しています。

(1) セマフォレジスタ番号の定義

共有メモリドライバで使用するセマフォレジスタの番号を 0~7 の間で指定します。

```
#define SHM_SYS_SEMFNO    (0)    // System SEMFn for shared memory access (n = 0-7)
```

図8.13 セマフォレジスタの選択例

(2) 共有メモリ領域の構造体メンバの定義

共有メモリ領域のメンバはユーザアプリケーションに応じて定義します。定義されたメンバが共有メモリサイズを超えないようにしてください。

```
/* ----- Shared memory struct ----- */
struct st_shm
{
    TAxis Axis[2];
};
```

図8.14 共有メモリ領域の構造体メンバの定義例

(3) 共有メモリサイズの変更

本ドライバでは、共有メモリ用に 4 KB を確保しています。共有メモリサイズを変更したい場合は、Cortex-M3 リンカ設定ファイル内の下線部のアドレスを変更してください。

- ・ IARワークショップ(iram_with_shm.icf)で使われるリンカ設定ファイルを定義
- ・ ファイルの場所は、「8.3.4.6 ファイル構成」で説明しています。
- ・ “region shm”は、共有メモリの領域です。“region buf_dram”は、Cortex-M3用プログラムデータ格納領域です。

```
define region shm = mem:[from 0x20000000 to 0x20000FFF];    /*!<Shared Memory with Data RAM */

define region buf_dram = mem:[from 0x20001000 to 0x20027FFF];    /*!< Data RAM(SystemArea) */
```

図8.15 共有メモリドライバのサイズ定義

9. リソース

9.1 ハードウェア

ソリューションキットファームウェアは、RZ/T1ソリューションキットの Biplane ボードで動作するように設計されています。本ファームウェアは、ハードウェアリソースに依存します。

9.2 OS

ソリューションキットファームウェアは、OS に依存しません。

9.3 メモリ

コード、定数、未初期化データを格納するファームウェアメモリブロックは、マップファイルに記述されています。

10. 関連ドキュメント

RZ/T1グループユーザズマニュアル：モーションコントロール・ソリューションキット (R01UH0665JU)

RZ/T1グループユーザズマニュアル ハードウェア編 (R01UH0483JJ)

付録 A. ASCII通信プロトコルコマンド

以下の表に、Biplane ファームウェアで使用される全設定パラメータ、ステータス変数、および関数を示します。

R/O : リードオンリ R/W : リード/ライト Cmd : コマンド

パラメータ ID	名称	バイト サイズ	アクセス	内容
0	STA	4	R/O	ステータスワード (4桁の16進数ワードとして出力)。ビットフラグは、以下に示します。
1	ERR	2	R/O	位置誤差: 現在と目標エンコーダ位置との差。値はサーボ制御がオンのときにのみ有効です。
2	ADC1	2	R/O	ADC チャンネル1の出力電流の読み出し。出力値と実際の電流値との対応は、ハードウェアに依存します。
3	ADC2	2	R/O	ADC チャンネル2の出力電流の読み出し
4	TC	2	R/O	総電流: ADC1 および ADC2 電流の絶対値の合計として計算されます。
5	CV	2	R/O	サンプル間隔ごとのエンコーダカウントの現在速度 (位置ループサイクル時間)
6	PVT	2	R/O	PVT バッファのポイント数
7	ITIME	2	R/W	連続 PVT ポイント間の時間間隔
8	VEL	4	R/W	指令速度 - (サンプル間隔ごとのエンコーダカウント) × 65536
9	ACC		R/W	指令加速度 - (サンプル間隔ごとのエンコーダカウントの二乗) × 65536
10	DEC	4	R/W	指令減速度 - (サンプル間隔ごとのエンコーダカウントの二乗) × 65536。値がゼロのときは、加速度の値が代わりに使用されます。
11	AJERK	4	R/W	加速度変化率 (0 ~ 1000)
12	DJERK	4	R/W	減速度変化率 (0 ~ 1000)
13	PRO	2	R/W	速度プロファイルモード
14	KP	2	R/W	位置制御ループアルゴリズムの比例ゲイン (0 ~ 32767)
15	KI	2	R/W	位置制御ループアルゴリズムの積分ゲイン (0 ~ 32767)
16	KD	2	R/W	位置制御ループアルゴリズムの差動ゲイン (0 ~ 32767)
17	IL	2	R/W	位置制御ループアルゴリズムの積分制限値 (0 ~ 32767)
18	VFF	2	R/W	位置制御ループアルゴリズムの速度フィードフォワード (0 ~ 32767)
19	AFF	2	R/W	位置制御ループアルゴリズムの加速度フィードフォワード (0 ~ 32767)
20	MAX	2	R/W	最大位置誤差 (0 ~ 32767)
21	ETIME	2	R/W	最大位置誤差の時間 (サンプル間隔)
22	DS	2	R/W	50us の倍数の位置ループ導関数 (サンプル間隔)。2 の設定は 100us のサンプル間隔を示します。

パラメータ ID	名称	バイト サイズ	アクセス	内容
23	MLIMIT	2	R/W	位置ループ PID レギュレータからのモニタ出力制限値 (0~32767)
24	BIAS	4	R/W	PID レギュレータの出力に連続して追加される値
25	ASTOP	2	R/W	自動停止モード。位置誤差を超えた後の動作を決定します。 0: 動作を停止 1: 動作を停止した後、サーボ制御を停止します。
26	PIMODE	2	R/W	位相初期化モード 0: 強制 1: ホール基準 2: デザリング基準
27	PITIME	2	R/W	位相初期化時間 (サンプル間隔)
28	PIOUT	2	R/W	位相初期化出力 (32767 = 100%)
29	PMAP	2	R/W	PWM 出力チャネルへの位相マッピング。ソフトウェアによるコントローラ出力とモータ巻線の接続を可能にします。値の範囲は 0~5。
30	PORIGIN	4	R/W	位相の原点: 磁束が 0 度の現在のモータ回転内のエンコーダ位置
31	PCMODE	2	R/W	位相整流モード 0: 空間ベクトル変調による電圧制御 1: 磁界方向制御 2: ホール基準の整流 3: ホスト定義の位相角
32	PVECTOR	4	R/W	位相ベクトルの方向 x 60 度。値の範囲は 0~5。
33	PPAIRS	2	R/W	極対数: このパラメータはモータのロータ構造を反映します。
34	PCOUNTS	4	R/W	1 電気サイクルあたりのエンコーダカウント (極対数で割った 1 回転ごとのエンコーダカウント)
35	ECPR	4	R/W	1 回転あたりのエンコーダカウント (エンコーダ分解能)
36	PIOFFS	4	R/W	位相初期化オフセット (位相初期化処理終了時に追加される位置オフセット)
37	PANGLE	4	R/O	現在の磁束角
38	PADV	4	R/W	位相進みゲイン (未使用)
39	VCOMP	4	R/W	速度補正 (未使用)
40	CLIMIT	2	R/W	電流制限閾値
41	CTIME	2	R/W	電流制限時間 (サンプル間隔)
42	IDM	4	R/O	直流 (熱発生) 電流
43	IQM	4	R/O	直交 (トルク発生) 電流
44	IQERR	4	R/O	直交電流誤差
45	QKP	2	R/W	直交電流制御ループの比例ゲイン
46	QKI	2	R/W	直交電流制御ループの積分ゲイン
47	PCT	2	R/W	位置取得モード (インデックス取得のみサポート)
48	HMASK	2	R/W	ホームスイッチマスク: ホーミングアルゴリズムの種類とホームセンサに接続される特定の入力を定義します。

パラメータ ID	名称	バイト サイズ	アクセス	内容
49	INVERT	2	R/W	ホームスイッチ反転マスク：正しい方向で検出されるようホームスイッチ入力の反転を定義します。
50	HINVERT	4	R/W	ホールセンサ信号反転 0：反転なし 1：反転
51	HSHIFT	4	R/W	ホールセンサ位置ずれ（未使用）
52	HPOS	4	R/W	ホールセンサ位置の変化 （ホールセンサのステータスが変化した最終位置）
53	EMASK	2	R/W	デジタル入力エラーマスク：インターロック発生時に動作停止をトリガするインターロックとしてのデジタル入力を定義します。
54	ECP	4	R/W	次の誤差が最大閾値を超えた指令位置
55	ECV	4	R/W	次の誤差が最大閾値を超えた速度
56	EPO	4	R/W	次の誤差が最大閾値を超えた現在位置
57	U	4	R/W	U相の出力電圧（32767 = デューティサイクル 100%）
58	V	4	R/W	V相の出力電圧（32767 = デューティサイクル 100%）
59	W	4	R/W	W相の出力電圧（32767 = デューティサイクル 100%）
60	TYPE	2	R/W	モジュールのタイプ 1：単一チャンネル 2：デュアルチャンネル（電子ギア付き） 3：デュアルチャンネル（2つの独立したモータ）
61	HTYPE	2	R/W	ホールセンサのタイプ（並列型のみをサポート）
62	HOFFS	4	R/W	ホームオフセット （ホーミング処理終了時にゼロ座標に追加される値）
63	DATA0	4	R/W	ユーザ定義データ（ファームウェアでは特に使用しない）
64	DATA1	4	R/W	
65	DATA2	4	R/W	
66	DATA3	4	R/W	
67	DATA4	4	R/W	
68	DATA5	4	R/W	
69	DATA6	4	R/W	
70	DATA7	4	R/W	
71	PHASES	2	R/W	モータのタイプを定義します。 2：DC ブラシモータ 3：ブラシレス DC/PMSM モータ
72	BRAKE	2	R/W	ブレーキ制御モード（未使用）
73	TMODE	2	R/W	トレースモード 0：アイドル状態 1：作動可能（トリガイベント待ち状態） 2：即時開始、バッファフルで停止 3：即時開始、動作終了時に停止
74	TRATE	2	R/W	データ記録レート（50us 間隔）

パラメータ ID	名称	バイト サイズ	アクセス	内容
75	TLEVEL	4	R/W	データ記録の閾値
76	SIM	2	R/W	エンコーダシミュレーションを許可 (1) または禁止 (0: デフォルト) します。
77	TIMER	2	R/W	PWM キャリア周波数を生成するタイマレジスタ
78	ADDP	4	R/W	PVT ストリーミングの位置設定値の追加
79	ADDV	4	R/W	PVT ストリーミングの速度設定値の追加
80	ABS	4	R/W	目標絶対位置を定義します。
81	REL	4	R/W	現在位置に対する目標位置を定義します。
82	POS	4	R/W	現在のエンコーダ位置
83	INP	2	R/O	4 バイトの 16 進数でデジタル入力の状態を通知します。
84	IND	4	R/O	取得インデックス位置
85	GO	0	Cmd	指定目標位置への動作を開始
86	FWD	0	Cmd	正方向のジョグ開始
87	REV	0	Cmd	逆方向のジョグ開始
88	RESET	0	Cmd	ファームウェアリセット、ソフトウェア再起動
89	ON	0	Cmd	サーボ制御を有効にします。
90	OFF	0	Cmd	サーボ制御を無効にします。
91	ENABLE	0	Cmd	PWM アンプ (インバータ) を有効にします。
92	DISABLE	0	Cmd	PWM アンプ (インバータ) を無効にします。
93	STOP	0	Cmd	動作をスムーズに (設定された減速度で) 停止します。
94	ABORT	0	Cmd	動作を急激に (最大減速度で) 停止します。
95	HOME	0	Cmd	ホーム処理の実行を開始します。
96	ALIGN	0	Cmd	フェージング処理の実行を開始します。
97	VER	4	R/O	ファームウェアの現在のバージョンを通知します。
98	OUT1	2	R/W	出力 1 を制御
99	OUT2	2	R/W	出力 2 を制御
100	PWM	2	R/W	PWM デューティサイクル (32767 = 100%) として設定された出力電圧。サーボ制御をオフにする必要があります。
101	IQPKT	2	R/W	モータ磁束の直交電流成分を生成する出力電圧
102	IDPKT	2	R/W	モータ磁束の直流成分を生成する出力電圧
103	CH1	2	R/W	データレコーダのチャンネル 1 に記録するデータを指定します。
104	CH2	2	R/W	データレコーダのチャンネル 2 に記録するデータを指定します。
105	CH3	2	R/W	データレコーダのチャンネル 3 に記録するデータを指定します。
106	CH4	2	R/W	データレコーダのチャンネル 4 に記録するデータを指定します。
107	TRACE	2	R/W	新しいデータ記録セッションを初期化します。
108	PLAY	2	R/W	記録されたデータを出力します。
109	PLIMIT	2	R/W	I2T 保護の閾値制限
110	PTIME	2	R/W	I2T 保護の期間

パラメータ ID	名称	バイト サイズ	アクセス	内容
111	GEARIN	2	R/W	電子ギア率を定義するギアボックスの速度伝達比の入力値を指定します。
112	GEAROUT	2	R/W	電子ギア率を定義するギアボックスの速度伝達比の出力値を指定します。
113	SETUP	2	R/W	モニタ巻線とホールセンサの正しいマッピングを特定する処理を開始します。
114	PINVERT	2	R/W	エンコーダ位置フィードバックを反転させるかどうかを指定します。 0 : 反転なし 1 : 反転
115	SAVE		Cmd	永続パラメータを外部フラッシュメモリに保存します。
116	RESTORE		Cmd	永続パラメータを外部フラッシュメモリから復元します。
117	ETYPE	2	R/W	エンコーダタイプ 0 : インクリメンタル 1 : EnDat 2 : BiSS 3 : FA-CODER 4 : A-format
118	EID	4	R/W	エンコーダ ID
119	EADDR	2	R/W	エンコーダ EEPROM アドレス
120	EDATA	2	R/W	EADDR 変数で定義されたアドレスに保存するエンコーダ EEPROM データ、またはそのアドレスから読み出されるエンコーダ EEPROM データ
121	EBAUDRATE	2	R/W	エンコーダ通信ボーレート
122	ESTATUS	2	R/W	アブソリュートエンコーダのステータス
123	ADDR	2	R/W	パケットホスト通信プロトコルで使用するモジュールアドレス
124	GROUP	2	R/W	パケットホスト通信プロトコルで使用するモジュールグループ
125	TBSIZE	2	R/O	トレースバッファのサイズ。データレコーダから出力されるサンプルの最大数をホストに伝えます。
126	WMARK	2	R/W	PVT バッファの閾値：警告フラグが立つ前に占有する PVT のスロット数を定義します。これは、ホストコンピュータによる新しい PVT ポイントのストリーミングを正しく同期するために必要です。
127	QKD	2	R/W	直交電流制御ループにおける比例ゲイン
128	VKP	2	R/W	速度制御ループにおける比例ゲイン
129	VKI	2	R/W	速度制御ループにおける積分ゲイン
130	VKD	2	R/W	速度制御ループにおける微分ゲイン
131	ELVOLT	4	R/W	インバータ母線電圧の低電圧検出閾値（12ビットAD値）
132	EHVOLT	4	R/W	インバータ母線電圧の過電圧検出閾値（12ビットAD値）

パラメータ ID	名称	バイト サイズ	アクセス	内容
133	EWPOSMIN	4	R/W	位置異常最小閾値
134	EWPOSMAX	4	R/W	位置異常最大閾値
135	EOVS	4	R/W	過速度検出閾値 ((位置*65535)/100usの値)
136	EWOVS	4	R/W	指示速度差閾値 ((位置*65535)/100usの値) 連続検出時間 (5秒) 後にエラーとなるよう設定してください。
137	EEMP	4	R/W	PVTバッファエンプティ閾値 (エンプティ回数の値)
138	EOVTEMP	4	R/W	温度異常閾値 (12ビットAD値)
139	ERRMASK	4	R/W	異常状態マスク (ErrMsk の値 (16進数))
140	EVOLT	4	R/O	母線電圧表示 (12ビットAD値)
141	EQUERY	4	R/O	異常状態表示 (ErrSts の値 (16進数))
142	ERESET	0	Cmd	異常状態リセット
143	EOVC	4	R/W	過負荷 (電流) 閾値 (12ビットAD値)
144	ETEMP	4	R/O	現在温度 (12ビットAD値)

付録 B. パケット通信プロトコルコマンド

パケットコード：

パケットコード	コマンド内容	要求パケット バイトサイズ	応答パケット バイトサイズ
0	Get Module Data	5	7
1	Data Report Request	7	可変
2	Function Call	6	3
3	Initialize PVT Mode	6	3
4	Stream Position/Velocity set points	5 + points * 8	3
5	Set Parameter (16-bit or 32-bit)	8または10	3
6	Get Parameter	6	5または7
7	Get Trace Buffer	6	可変
15	Set Communication Baud Rate	6	3

応答コード：

応答コード	内容
0	Command Accepted
1	Invalid Packet Code
2	Invalid Parameter Index
3	Invalid Data Size / Format
4	Parameter Value Out of Range
5	Invalid Request - Attempt to set a Read-Only variable
6	Interlock Active
7	Controller Fault
8	Invalid State
9	Buffer Overflow
10	Execution Error
255	Checksum error

B.1 Get Module Data (パケットコード : 0)

このパケットタイプは、モジュールのタイプやパケットプロトコルのバージョンを識別するため、初期化時にのみ使用します。ホストは、このパケット形式に適合する場合、種類やバージョンの異なるモジュールへの対応が可能となります。

ヘッダ			ペイロード	
開始	アドレス	長さ	パケットコード	チェックサム
0xAA	0x01	0x00	0x00	0x01

返信パケット : 長いパラメータ値

応答コード	長いパラメータ値 (4 バイト)				チェックサム
0x00	B0	B1	B2	B3	X

B0 : プロトコルのバージョン

B1 : モジュールのタイプ

B2 : ファームウェアのマイナーバージョン番号

B3 : ファームウェアのメジャーバージョン番号

B.2 Status Report Request (パケットコード : 1)

このパケットタイプは、最もよく用いられるコントローラパラメータの通知をリクエストします。通知内容は 16 ビット値で定義されます。このコマンドは、ホストがコントローラのステータスを定期的にポーリングする場合に使用します。通知内容は、現在実行中の動作に応じて変わる場合があります。

ヘッダ			ペイロード			
開始	アドレス	長さ	パケットコード	リクエスト		チェックサム
0xAA	0x01	0x02	0x01	X	X	X

要求ワード定義

要求ビット番号	データの内容	データバイト長
0 (LSB)	現在位置	4
1	現在のステータス	2
2	位置誤差	2
3	消費電流	2
4	ホール入力	1
5	デジタル入力	1
6	PVT FIFO バッファフル	1
7	スキャンデータ FIFO バッファフル	1
8 ... 15	予約	N/A

返信パケット : 可変長データ

応答コード	データ 1	...	データ N	チェックサム
0	B0	...	Bn	X

以下の表に、現在のステータスの通知がリクエストされたときのステータスワードの戻り値のビットフィールドの内容を示します。

ビット番号	ビットマスク	内容
0 (LSB)	0x0001	動作完了
1	0x0002	サーボ制御がオン
2	0x0004	パワーアンプ有効
3	0x0008	位置取得
4	0x0010	アブソリュートエンコーダの異常
5	0x0020	アブソリュートエンコーダ警告
6	0x0040	PVT FIFO バッファの深さが定義された閾値以下
7	0x0080	フェーシング (位相調整) が完了
8	0x0100	予約
9	0x0200	電流過負荷
10	0x0400	入力トリガ禁止
11	0x0800	PVT バッファ枯渇
12	0x1000	オーバーヒート
13	0x2000	インバータ (アンプ) 異常
14	0x4000	位置誤差の超過
15 (MSB)	0x8000	位置カウンタ周回

B.3 Function Call (パケットコード : 2)

引数なし

ヘッダ			ペイロード		
開始	アドレス	長さ	パケットコード	関数 ID	チェックサム
0xAA	0x01	0x01	0x02	X	X

関数IDは、本章最後の表に変数IDと共に定義されています。

返信パケット (ステータス)

応答コード	チェックサム
0	0

B.4 Initialize PVT Stream (パケットコード : 3)

引数なし

ヘッダ			ペイロード		
開始	アドレス	長さ	パケットコード	タイムスライス	チェックサム
0xAA	0x01	0x01	0x03	X	X

関数IDは、本章最後の表に変数IDと共に定義されています。

返信パケット (ステータス)

応答コード	チェックサム
0	0

B.5 Stream PVT Data (パケットコード : 4)

ヘッダ			ペイロード																						
開始	アドレス	長さ	パケットコード	ポイント カウント	ポイント 1				ポイント 1				ポイント N				ポイント N				チェ ック サム				
					位置	速度	位置	速度	位置	速度	位置	速度													
0xAA	0x01	$8 \times N + 1$	0x04	N	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

返信パケット (ステータス)

応答コード	チェックサム
0	0

B.6 Set Parameter (パケットコード : 5)

このパケットタイプは、定義済みパラメータに新しい値を設定します。パラメータは 16 または 32 ビット値で定義します。値のサイズは、本章最後のコントローラパラメータ一覧表に示しています。

B.6-1 16ビットパラメータ設定

ヘッダ			ペイロード				
開始	アドレス	長さ	パケットコード	パラメータコード	パラメータ値		チェックサム
0xAA	0x01	0x03	0x05	X	B0	B1	X

B0 : LSB

B1 : MSB

B.6-2 32ビットパラメータ設定

ヘッダ			ペイロード						
開始	アドレス	長さ	パケットコード	パラメータコード	パラメータ値				チェックサム
0xAA	0x01	0x05	0x05	X	B0	B1	B2	B3	X

B0 : LSB

B3 : MSB

返信パケット (ステータス)

応答コード	チェックサム
0	0

B.7 Get Parameter (パケットコード : 6)

このパケットはコントローラパラメータの戻り値を要求します。応答パケットの長さは各パラメータによって異なります。全パラメータとそのサイズ一覧は本章の最後に記載しています。

ヘッダ			ペイロード		
開始	アドレス	長さ	パケットコード	パラメータコード	チェックサム
0xAA	0x01	0x01	0x06	X	X

返信パケット (短いパラメータ値 (2バイト))

応答コード	短いパラメータ値 (2バイト)		チェックサム
0	B0	B1	X

B0 : LSB

B1 : MSB

返信パケット (長いパラメータ値)

応答コード	長いパラメータ値 (4バイト)				チェックサム
0	B0	B1	B2	B3	X

B0 : LSB

B3 : MSB

B.8 Report Trace Buffers Content (パケットコード : 7)

ヘッダ			ペイロード		
開始	アドレス	長さ	パケットコード	要求バイト	チェックサム
0xAA	0x01	0x01	0x07	B0	X

バッファ定義 : バイト B0 (通知するチャンネルバッファを下位 4 ビットで定義)

B0:3 ~ B0:0	通知するバッファ
0	チャンネル 1
1	チャンネル 2
2	チャンネル 3
3	チャンネル 4

データ形式 (サンプルごとの 16 ビット / 32 ビットデータ) を最上位ビットで定義

B0:7	データサイズ
0	データを 16 ビット値で通知
1	データを 32 ビット値で通知

応答パケットの中身は要求バイトのビットフィールドで定義します。

返信パケット (可変長データ)

ステータス	データ 1	...	データ N	チェックサム
S0	B0	...	Bn	X

B.9 Set Communication Baud Rate (パケットコード : 15)

ヘッダ			ペイロード		
開始	アドレス	長さ	パケットコード	ボーレートコード	チェックサム
0xAA	0x01	0x01	0x0F	B0	X

返信パケット (ステータス)

応答コード	チェックサム
0	0

ボーレートコード定義

コード	ボーレート [Kbps]
0	115.2
1	230.4
2	250
3	460.8
4	500
5	921.6
6	1000
7	1250
8	1500

改訂記録	RZ/T1 グループソリューションキットファームウェア アプリケーションノート
------	--

Rev.	発行日	改訂内容	
		ページ	ポイント
1.01	2018.10.05	—	初版発行
1.02	2019.07.31	—	製品ご使用上の注意事項 登録商標に関する文言を追加
		10	表 2.1 動作環境 統合開発環境を修正 (e ² studio を追加、IAR Embedded Workbench のバージョン変更)
		11-13	表 3.1 ファイル構成 (Cortex-R4 用) ファイル名を修正、ファイルを追加
		14, 15	表 3.2 ファイル構成 (Cortex-M3 用) ファイル名を修正、ファイルを追加
		21	4.1.3 周期リアルタイム関数 誤記訂正 (pos_loop 関数、crnt_loop 関数の内容)、項目追加 (vel_loop 関数)
		25	5.2 周辺機能の初期化 ファイルの追加 (r_cg_poe3.c)
		28	6.1 モータ位置 エンコーダインタフェース 使用可能エンコーダを追加 (ETYPE_APE_HIPERFACE_DSL = 5)
		39-41	7. システム制御関数 誤記訂正、インターロック条件の説明文を修正 (各条件にタイトルを追加、項目 6~15 を追加)
		39	図 7.1 インターロックでの判定 図を変更
		55	表 8.1 対応動作モード一覧 動作モードの対応/非対応を修正
		57	表 8.2 対応オブジェクトディクショナリー一覧 (動作モード) オブジェクトを追加、動作モードを追加
		58	表 8.2 対応オブジェクトディクショナリー一覧 (その他オブジェクト) オブジェクトを追加・削除
		72	表 8.6 Cortex-M3 ドライバで使用するファイル ファイルを追加
89, 90	付録 A. ASCII 通信プロトコルコマンド パラメータ ID (127~144) を追加		

RZ/T1 グループ



ルネサスエレクトロニクス株式会社

R11AN0086JJ0102