RX23T / RX24T / RX66T / RX72M / RX72T グループ

レゾルバ信号変換 IC 制御ドライバ（Rev2.10）向けアプリケーションノート

要旨

本書は RDC-IC を制御するために必要な RDC-IC ドライバ（Rev2.10）のアプリケーションノートです。本ドライバは、RX24T グループ　レゾルバ信号変換 IC 制御ドライバ向けアプリケーションノート（Rev1.20）に同梱するドライバライブラリのアップグレードバージョンです。

動作確認デバイス

- RX23T（R5F523T5ADFM）
- RX24T（R5F524TAADFM）
- RX66T（R5F566TKCDFB）
- RX72M（R5F572MNDDFC）
- RX72T（R5F572TKCDFB）
- RDC-IC（RAA3064002GFP/RAA3064003GFP）

目次

1. 概要 7
1.1 ドライバの機能 7
1.2 開発環境 7
1.3 プログラムサイズ 7
1.4 関連資料 7

2. 全体構成 8
2.1 システム構成 8
2.2 RDC-IC の機能説明 8

3. 機能説明 9
3.1 ドライバ初期化処理 10
3.1.1 SC によるペリフェラル初期設定 10
3.1.2 システム情報の指定 11
3.1.3 関数テーブル設定 11
3.1.4 ペリフェラルスタート 11
3.2 RDC-IC 設定 12
3.2.1 RDC-IC 初期設定 12
3.3 RDC-IC 動作クロック出力 12
3.3.1 RDC-IC 動作クロック出力開始 12
3.4 RDC-IC 通信機能 13
3.4.1 RDC-IC レジスタ書き込み 13
3.4.2 RDC-IC レジスタ読み込み 13
3.4.3 RDC-IC 通信処理 13
3.5 励磁信号出力 14
3.5.1 励磁信号周期割り込み 15
3.5.2 励磁信号出力開始 15
3.5.3 励磁信号出力停止 15
3.5.4 励磁信号出力開始タイミング調整 16
3.6 角度信号入力 18
3.6.1 角度信号入力割り込み 19
3.6.2 角度信号入力開始 19
3.6.3 角度信号入力停止 19
3.6.4 角度信号入力開始タイミング調整 19
3.7 断線検知機能 20
3.7.1 断線検知時に使用する機能 20
3.8 ALARM 解除機能 21
3.9 レゾルバ信号の位相調整信号出力 22
3.9.1 位相調整信号出力開始 22
3.9.2 位相調整信号出力停止 22
3.9.3 位相調整信号 Duty バッファ設定 22
3.9.4 位相調整信号 Duty レジスタ設定 22
3.9.5 位相調整信号 Duty バッファ読み込み 22
3.10 角度誤差補正信号出力 23
3.10.1 角度誤差補正信号出力開始 24

3.10.2 角度誤差補正信号出力停止 24
3.10.3 角度誤差補正信号 Duty 更新設定 24
3.10.4 角度誤差補正信号 Duty 更新割り込み 24
3.11 自動調整機能 25
3.11.1 パラメータ調整時に使用する機能 25
3.11.2 レゾルバ信号ゲイン・位相調整機能 26
3.11.3 角度誤差補正信号調整機能 28
3.12 タイミングチャート（励磁信号、角度信号入力、角度誤差補正信号） 30

4. ソフトウェア構成 31
4.1 フォルダ・ファイル構成 31

5. ペリフェラル設定 32
5.1 各ドライバ機能のマクロ定義名一覧 32
5.2 各ドライバ機能に割り当てるペリフェラル一覧（推奨） 33
5.3 SC による各ドライバ機能の設定 39
5.3.1 励磁信号出力 39
5.3.2 レゾルバ信号の位相調整信号出力 43
5.3.3 角度誤差補正信号出力 46
5.3.4 角度誤差補正信号 Duty 更新割り込み 49
5.3.5 角度信号入力 52
5.3.6 RDC-IC 動作クロック出力 55
5.3.7 RDC-IC 通信機能 58
5.4 関数テーブル設定 61
5.4.1 タイマ開始・停止関数 62
5.4.2 カウント値取得・設定関数 62
5.4.3 Duty 値取得・設定関数 62
5.4.4 キャプチャ値取得関数 63
5.4.5 ポートレベル取得関数 63
5.4.6 SPI 送受信関数 64

6. API 66
6.1 API 関数一覧 66
6.2 API 関数説明 70
6.2.1 関数テーブル設定 API 関数 70
6.2.2 システム情報の指定 API 関数 70
6.2.3 RDC-IC ドライバ設定情報取り出し API 関数 70
6.2.4 MTU3 タイマ同時スタート制御 API 関数 71
6.2.5 RDC-IC ドライババージョン取得 API 関数 71
6.2.6 角度誤差補正信号出力開始 API 関数 71
6.2.7 角度誤差補正信号出力停止 API 関数 72
6.2.8 角度誤差補正信号 Duty 更新 API 関数 72
6.2.9 角度誤差補正信号同期スタート API 関数 72
6.2.10 角度誤差補正信号出力状態取得 API 関数 72
6.2.11 角度検出タイマスタート API 関数 73
6.2.12 角度検出値取り込み API 関数 73
6.2.13 角度検出値取り込み割り込みのトリガ取り出し API 関数 73

6.2.14 レゾルバ角度カウンタ（取り込みトリガ：立ち下がりエッジ）取り出し API 関数 74
6.2.15 レゾルバ角度差分カウンタ（取り込みトリガ：立ち下がりエッジ）取り出し API 関数 74
6.2.16 レゾルバ角度カウンタ（取り込みトリガ：立ち上がりエッジ）取り出し API 関数 74
6.2.17 レゾルバ角度差分カウンタ（取り込みトリガ：立ち上がりエッジ）取り出し API 関数 74
6.2.18 励磁信号出力開始 API 関数 75
6.2.19 励磁信号出力停止 API 関数 75
6.2.20 励磁信号出力開始タイミング設定 API 関数 75
6.2.21 待機時間確認カウンタ API 関数 75
6.2.22 位相調整信号出力開始 API 関数 75
6.2.23 位相調整信号出力停止 API 関数 76
6.2.24 位相調整信号 Duty バッファ設定 API 関数 76
6.2.25 位相調整信号 Duty レジスタ設定 API 関数 76
6.2.26 位相調整信号 Duty バッファ読み込み API 関数 76
6.2.27 RDC-IC 初期値設定 API 関数 77
6.2.28 RDC-IC 初期化シーケンス実行 API 関数 77
6.2.29 RDC-IC 通信処理 API 関数 77
6.2.30 RDC-IC 各レジスタへの書き込み API 関数 77
6.2.31 RDC-IC 各レジスタの読み込み API 関数 77
6.2.32 RDC-IC レジスタアクセス状態取得 API 関数 78
6.2.33 RDC-IC レジスタバッファからの読み込み API 関数 78
6.2.34 RDC-IC レジスタバッファへの書き込み API 関数 78
6.2.35 RDC-IC 通信送受信完了割り込みコールバック API 関数 78
6.2.36 RDC-IC 通信エラー割り込みコールバック API 関数 78
6.2.37 RDC-IC 通信の異常発生有無通知 API 関数 79
6.2.38 RDC-IC ALARM 解除開始 API 関数 79
6.2.39 RDC-IC ALARM 解除シーケンス制御 API 関数 79
6.2.40 レゾルバ信号ゲインおよび位相調整 API 関数 79
6.2.41 角度誤差補正信号調整 API 関数 80
6.2.42 ユーザ作成のコールバック関数ポインタ設定 API 関数 80
6.2.43 AD 変換実施状態取得 API 関数 80
6.2.44 遅れ位相設定用 API 関数 80
6.2.45 断線検知処理 API 関数 80
6.3 構造体説明 81
6.3.1 R_RSLV_SetFuncTable API 関数の構造体 81
6.3.2 R_RSLV_SetSystemInfo API 関数の構造体 83
6.3.3 R_RSLV_GetRdcDrvSettingInfo API 関数の構造体 84
6.3.4 R_RSLV_ADJST_GainPhase API 関数の構造体 85
6.3.5 R_RSLV_ADJST_Carrier API 関数の構造体 87
6.3.6 R_RSLV_ADJST_SetPtrFunc API 関数の構造体 88
6.3.7 R_RSLV_DiscDetection_Seq API 関数の構造体 88

7. API 関数の実装例 89
7.1 ペリフェラル準備 90
7.1.1 SC 設定 90
7.1.2 ユーザ作成コード 90
7.2 初期化処理 91
7.2.1 MCU 初期化 91

7.2.2　ドライバ初期化 91
7.2.3　サンプルコード 95
7.3　メインループ 103
7.3.1　実装例 103
7.3.2　サンプルコード 104
7.4　励磁信号出力 106
7.4.1　実装例 106
7.4.2　サンプルコード 107
7.5　位相調整信号出力 108
7.5.1　実装例 108
7.5.2　サンプルコード 109
7.6　角度誤差補正信号出力 110
7.6.1　実装例 110
7.6.2　サンプルコード 111
7.7　角度信号入力 113
7.7.1　実装例 113
7.7.2　サンプルコード 114
7.8　ゲイン&位相自動調整機能 115
7.8.1　実装例 115
7.8.2　ゲイン＆位相調整詳細 116
7.8.3　サンプルコード 118
7.9　角度誤差補正信号自動調整機能 119
7.9.1　実装例 119
7.9.2　角度誤差補正信号調整詳細 120
7.9.3　フィルタ回路による遅れ位相 123
7.9.4　サンプルコード 124
7.10　RDC-IC 通信機能 125
7.10.1　実装例 125
7.10.2　サンプルコード 126
7.11　断線検知機能 130
7.11.1　実装例 130
7.11.2　サンプルコード 132
7.12　ALARM 解除機能 134
7.12.1　実装例 134
7.12.2　サンプルコード 135

8.　Rev1.20 以前から Rev2.10 への移行方法 136
8.1　フォルダ・ファイル構成変更 136
8.1.1　ライブラリとヘッダーファイルの差し替えと SC コードの追加 136
8.1.2　プロジェクトへの登録 137
8.2　ソースコード修正 138
8.2.1　ペリフェラル初期化処理 138
8.2.2　SC コードの修正 139
8.2.3　API 関数の修正 143
8.2.4　その他の修正 147

9.　注意事項 148

9.1　初期設定手順 148
9.2　同一ペリフェラルへの複数機能設定 148
9.3　同一機能への複数ペリフェラル設定 148
9.4　RDC-IC 通信用変数の初期化 148
9.5　位相調整信号のペリフェラル設定 148
9.6　タイマスタートタイミング設定 148
9.7　調整機能の動作について 148
9.8　角度誤差補正位相シフト量 149
9.9　関数テーブルの設定順序 150
9.10　角度誤差補正信号調整機能について 150

10.　トラブルシューティング 151
10.1　カウンタ値異常 152
10.2　回転方向異常 153
10.3　角度異常 154
10.4　断線検知 155

改訂記録 161

1. 概要

1.1 ドライバの機能

ドライバの機能は下記の通りです。

- RDC-IC 設定
- RDC-IC 動作クロック出力
- RDC-IC 通信機能
- 励磁指令信号出力
- 角度信号入力
- 断線検知機能
- ALARM 解除機能
- 位相調整用信号出力
- 角度誤差補正信号出力
- 自動調整機能

1.2 開発環境

以下に本ドライバの動作確認環境を示します。

表 1-1　開発環境（ソフトウェア）

IDE バージョン	ツールチェイン	スマート・コンフィグレータ
CS+：V8.08.00 e^2 studio：V2022-10	CC-RX V3.02.00	Version：2.15.0

1.3 プログラムサイズ

以下に本ドライバのプログラムサイズを示します。

表 1-2　プログラムサイズ

ROM サイズ	RAM サイズ	使用 STACK サイズ
12570 Byte	1075 Byte	164 Byte

1.4 関連資料

RX23T グループ　ユーザーズマニュアル　ハードウェア編（R01UH0520）

RX24T グループ　ユーザーズマニュアル　ハードウェア編（R01UH0576）

RX66T グループ　ユーザーズマニュアル　ハードウェア編（R01UH0749）

RX72M グループ　ユーザーズマニュアル　ハードウェア編（R01UH0804）

RX72T グループ　ユーザーズマニュアル　ハードウェア編（R01UH0803）

RX スマート・コンフィグレータ　ユーザーガイド：e^{2}studio 編（R20AN0451）

RX スマート・コンフィグレータ　ユーザーガイド：CS+編（R20AN0470）

レゾルバ信号変換 IC ユーザーズマニュアル　ハードウェア編（R03UZ0002）

レゾルバ信号変換 IC の周辺部品選定ガイド（R03AN0012）

2.　全体構成

2.1　システム構成

以下に RDC-IC と MCU のシステム構成を示します。

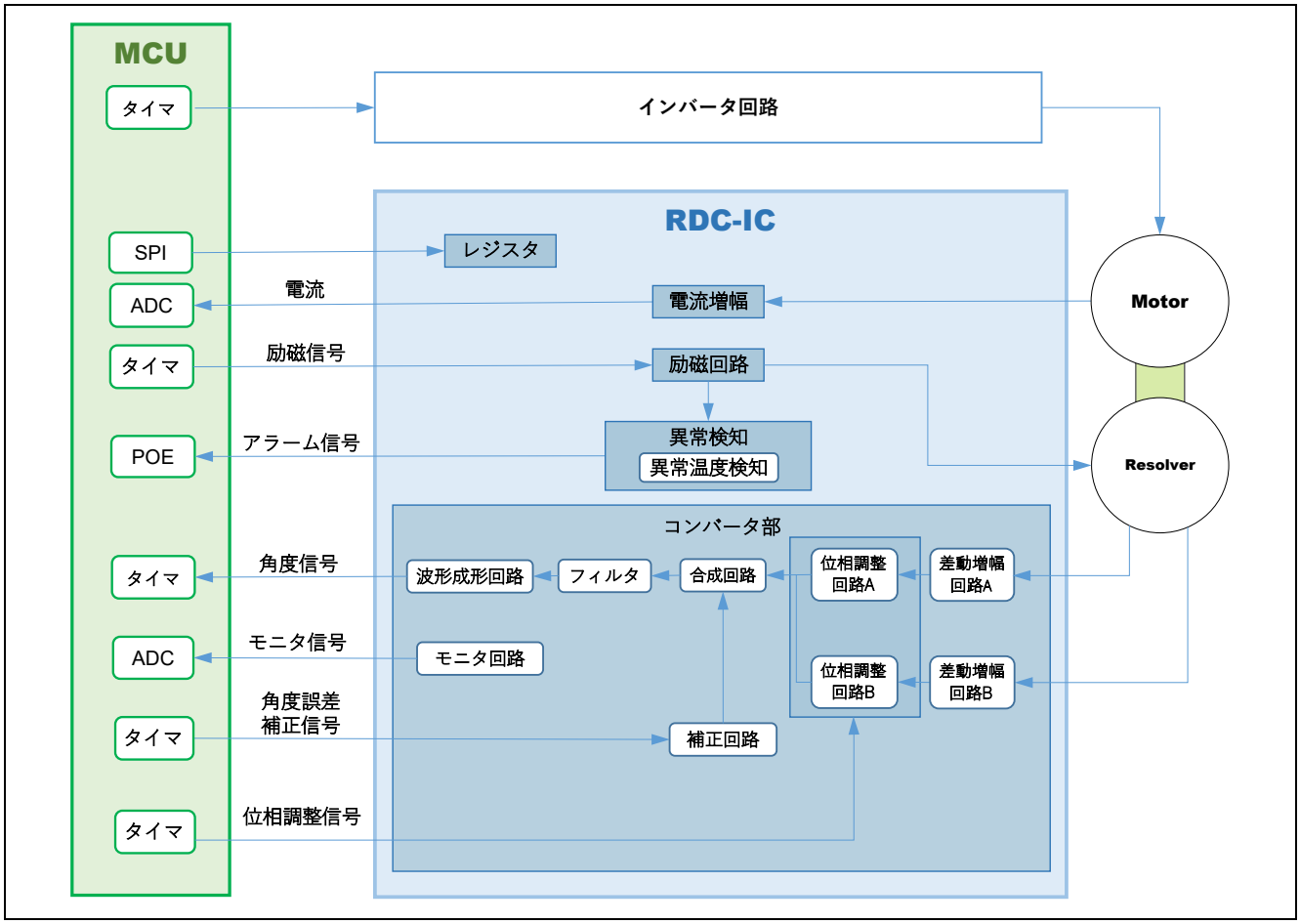

図 2.1　RDC-IC と MCU のシステム構成

2.2　RDC-IC の機能説明

RDC-IC は、レゾルバセンサを励磁する励磁回路、レゾルバセンサが出力するアナログ信号をデジタル信号に変換するコンバータを内蔵しています。

励磁回路は、MCU から出力した矩形波をアナログ信号に変換して、レゾルバセンサを励磁します。

コンバータ部は、レゾルバセンサが出力する 2 相信号（電気角度情報）を角度信号（矩形波）に変換します。MCU のタイマ機能を使用して、励磁矩形波と角度信号の位相差から、角度情報を得ることができます。またコンバータ部はゲイン調整機能、位相調整機能、角度誤差補正機能を実装しています。

ゲイン調整機能は、RDC-IC の設定を調整することにより、レゾルバセンサの 2 相信号の振幅を同一レベルに調整します。

位相調整機能は、MCU から RDC-IC へ位相調整用の補正信号を出力して、レゾルバセンサの 2 相信号の位相差を 90 度に調整します。

角度誤差補正機能は、レゾルバセンサのアナログ誤差を補正する機能です。MCU から RDC-IC へ角度誤差補正信号出力をすると、コンバータ部の補正回路を通して角度信号に合成します。

本ドライバソフトウェアは、MCU から RDC-IC へ入力する矩形波信号や補正信号出力、コンバータ部が出力する角度信号を検出する機能を実装します。

3.　機能説明

ドライバソフトウェアが実装する各ドライバ機能の説明を以下に示します。

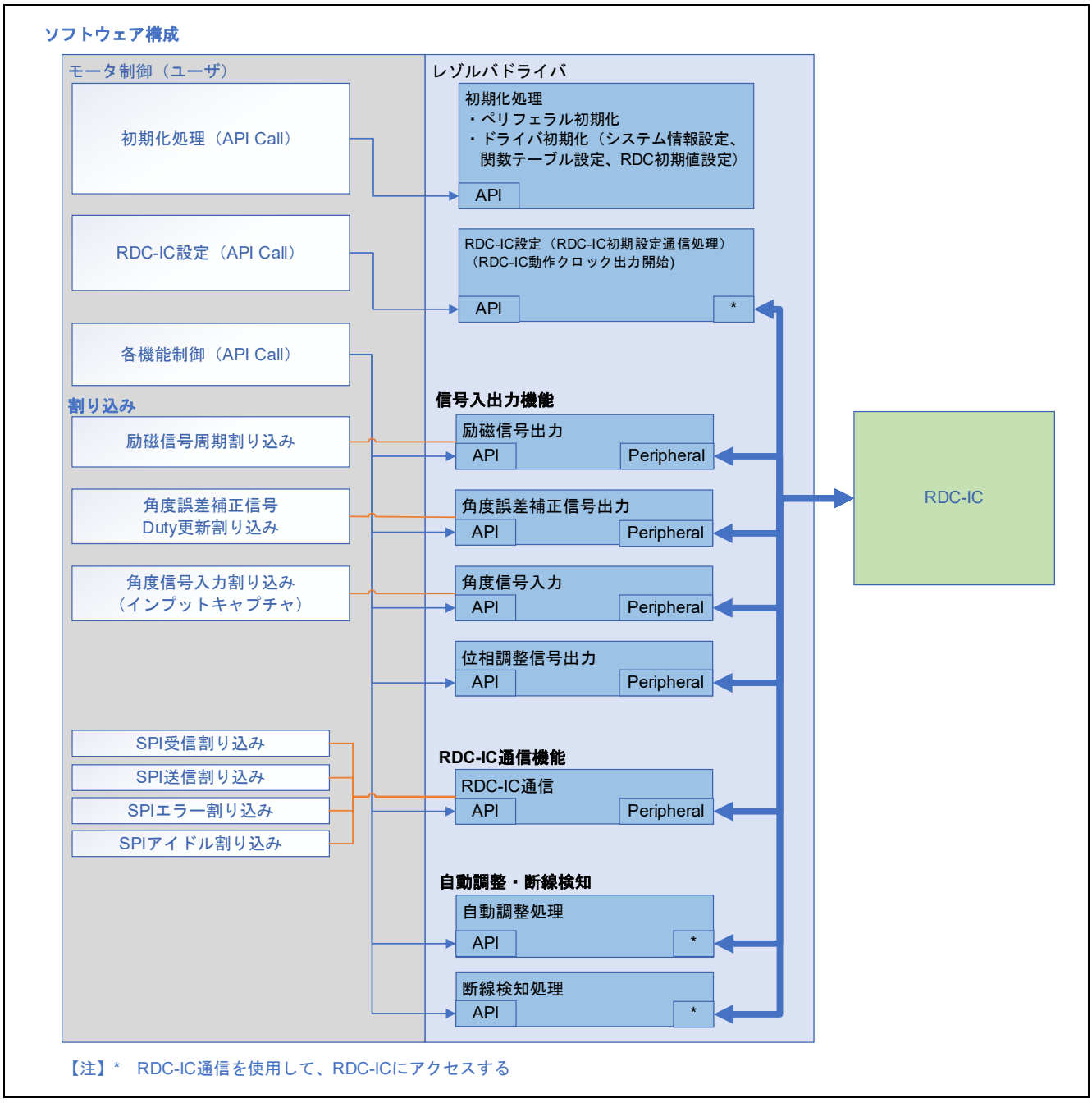

図 3.1　ソフトウェア構成

3.1　ドライバ初期化処理

レゾルバドライバを初期化するためには、ペリフェラルの初期設定、システム情報設定、関数テーブル設定を実施します。その後、本ドライバ機能に割り当てた各ペリフェラルをスタートします。ペリフェラルの初期設定は、スマート・コンフィグレータ（以降 SC とする）やユーザが作成したペリフェラル初期化関数を使用して設定します。

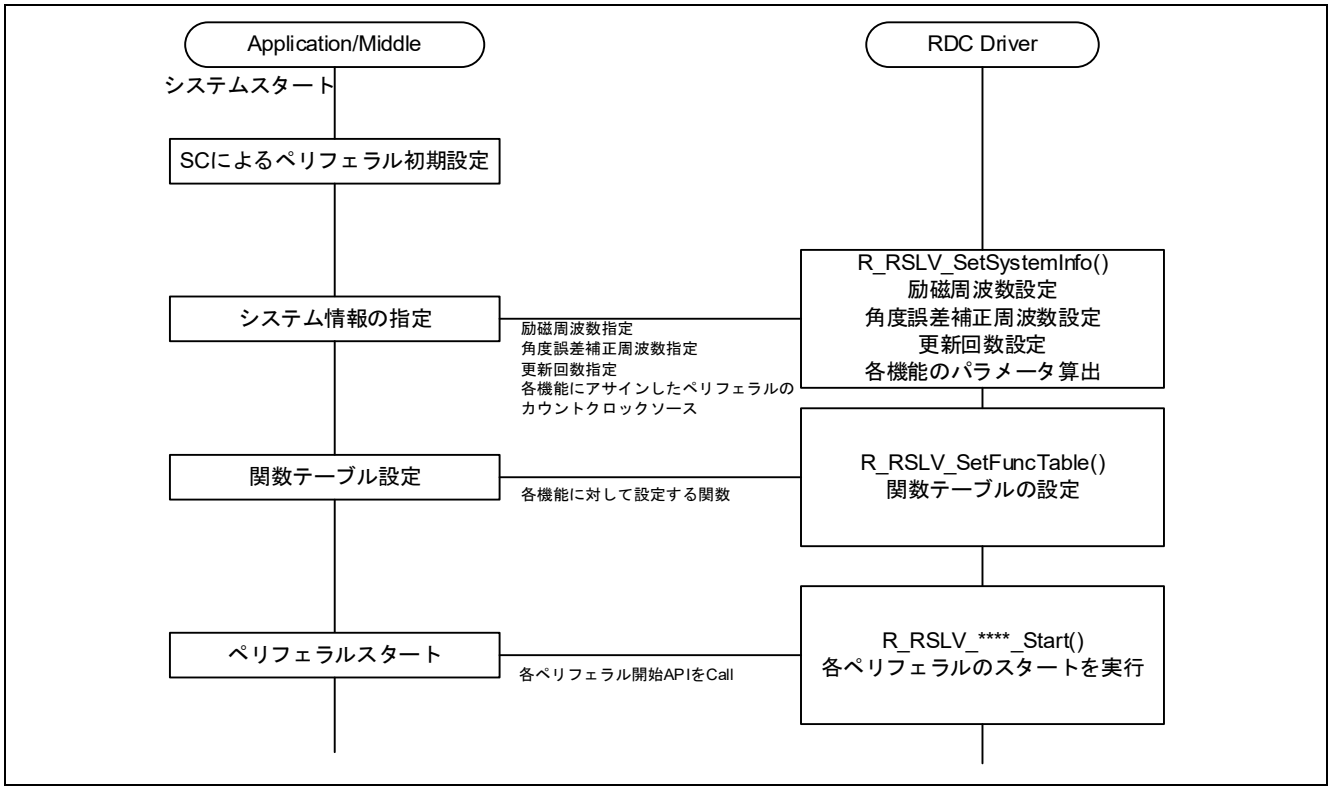

図 3.2　初期設定シーケンス

3.1.1　SC によるペリフェラル初期設定

各ドライバ機能に割り当てるペリフェラルの初期化関数は、ユーザが SC を使用して生成します。MCU 起動時に生成した初期化関数を呼び出し、ペリフェラルを初期化します。

本ドライバは、SC で生成したペリフェラル初期化関数（参考用）をサンプルコードとして同梱しています。

3.1.2　システム情報の指定

システム情報は、励磁周波数、角度誤差補正周波数、角度誤差補正信号の更新回数、また、各ドライバ機能のペリフェラルに対するクロックソースなどを指定し、システム情報設定 API 関数を呼び出して設定します。システム情報の指定については、「6.3.2　R_RSLV_SetSystemInfo API 関数の構造体」を参照してください。

本関数を呼び出すと、ドライバ内部で使用する位相調整信号の Duty 初期値、角度誤差補正信号の最大/最小カウント値、角度誤差補正信号 Duty 更新周期、角度信号入力の最大/最小カウント値が設定されます。

API 関数：R_RSLV_SetSystemInfo
(ST_SYSTEM_PARAM *rdc_sys_param, ST_USER_PERI_PARAM *user_peri_param)

3.1.3　関数テーブル設定

関数テーブルは、レゾルバドライバがペリフェラルレジスタをアクセスするために使用するテーブルです。SC やユーザが作成したレジスタアクセス関数を本テーブルに設定する事で、ドライバはペリフェラルレジスタにアクセスできるようになります。関数テーブル設定方法については、「6.2.1　関数テーブル設定 API 関数」を参照してください。

API 関数：R_RSLV_SetFuncTable
(unsigned char set_func, ST_FUNCTION_TABLE user_func_table)

3.1.4　ペリフェラルスタート

各ドライバ機能の動作を開始させるため、ペリフェラルスタートの API 関数を用意しています。詳細は、「6.1　API 関数一覧」を参照してください。ペリフェラルスタート API 関数は、励磁信号出力、角度誤差補正信号出力、位相調整信号出力、角度信号入力を用意しています。

＜ペリフェラルスタート API 関数＞

励磁信号出力：R_RSLV_ESig_Start(void)
角度誤差補正信号出力：R_RSLV_CSig_Start(unsigned short phase_diff, unsigned short amp_level)
位相調整信号出力：R_RSLV_Phase_AdjStart(void)
角度信号入力：R_RSLV_Capture_Start(void)

3.2　RDC-IC 設定

レゾルバを制御するためには RDC-IC の動作設定をする必要があります。設定は、SPI 通信機能を用いて、RDC-IC のレジスタを設定することで行います。

3.2.1　RDC-IC 初期設定

RDC-IC の動作設定を初期化するためには、RDC-IC レジスタの初期値を API 関数で指定し、RDC-IC 初期化シーケンス API 関数を実行します。レジスタの初期値は、使用するレゾルバセンサの仕様に合わせてユーザが指定します。

RDC-IC 初期値設定 API 関数：
R_RSLV_Rdc_VariableInit((unsigned char*)s_u1_rdc_init_data)
RDC-IC 初期化シーケンス実行 API 関数：
R_RSLV_Rdc_Init_Sequence(unsigned short *init_status)

3.3　RDC-IC 動作クロック出力

RDC-IC を動作させるための動作クロックを出力します。出力するクロック信号は、4MHz の矩形波信号となります。

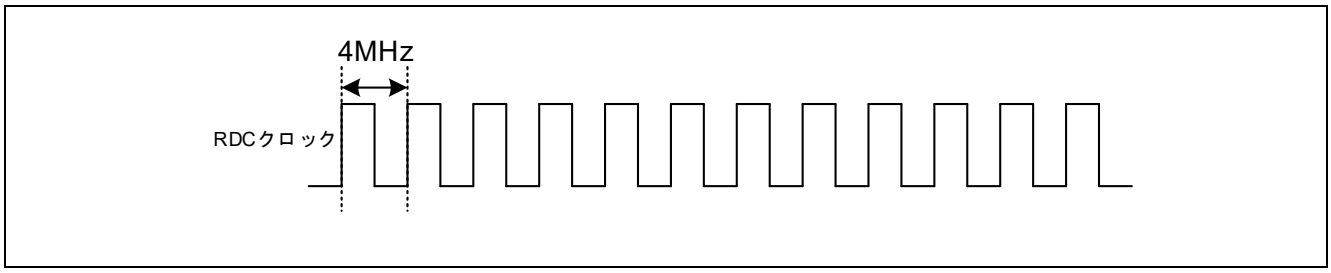

図 3.3　RDC-IC クロック

3.3.1　RDC-IC 動作クロック出力開始

RDC-IC 動作クロックは、RDC-IC 初期化シーケンス API 関数が出力を開始します。本ドライバは、RDC-IC 動作クロックを停止することはありません。

API 関数：R_RSLV_Rdc_Init_Sequence(unsigned short *init_status)

3.4　RDC-IC 通信機能

MCU と RDC-IC の通信は、SPI 通信を使用します。以下に RDC-IC 通信ブロックのシステム概要を示します。

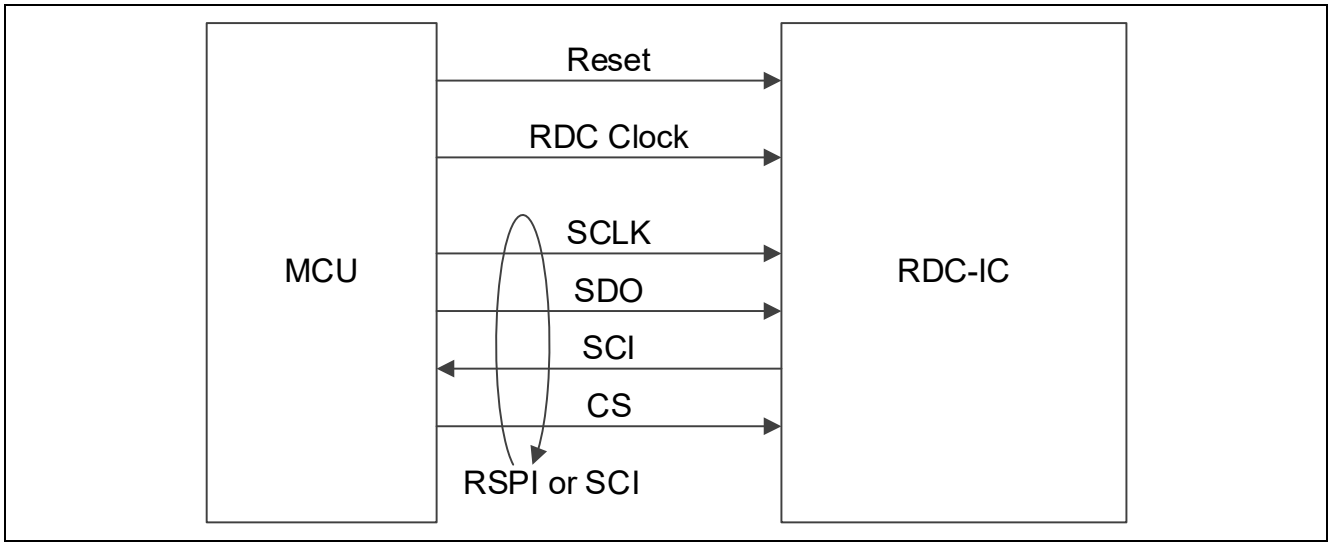

図 3.4　RDC-IC 通信ブロックシステム概要

3.4.1　RDC-IC レジスタ書き込み

RDC-IC レジスタへ任意の設定値を書き込む場合、レゾルバドライバにレジスタ設定値を渡す API 関数と、書き込み開始用 API 関数を使用します。レジスタ設定用 API 関数を呼び出した後、書き込み開始 API 関数を呼び出すことで、RDC-IC レジスタへの書き込みを行います。

RDC-IC レジスタバッファへの書き込み API 関数（レゾルバドライバにレジスタ設定値を渡す）：
R_RSLV_Rdc_SetRegisterVal(unsigned char wt_data, unsigned char address)
RDC-IC レジスタへの書き込み API 関数（書き込み開始）：
R_RSLV_Rdc_RegWrite(unsigned char *write_status)

3.4.2　RDC-IC レジスタ読み込み

RDC-IC レジスタの設定値を読み込む場合、読み込み開始用 API 関数と、レゾルバドライバのレジスタ値を受け取る API 関数を使用します。読み込み開始 API 関数を呼び出した後、レゾルバドライバのレジスタ値を受け取る API 関数を呼び出すことで、RDC-IC レジスタからの読み込みを行います。

RDC-IC レジスタの読み込み API 関数（読み込み開始）：
R_RSLV_Rdc_RegRead(unsigned char address)
RDC-IC レジスタバッファからの読み込み API 関数（レゾルバドライバのレジスタ値を受け取る）：
R_RSLV_Rdc_GetRegisterVal(unsigned char *rd_data, unsigned char address)

3.4.3　RDC-IC 通信処理

RDC-IC と通信を実施する場合に、RDC-IC 通信処理 API 関数を呼び出します。RDC-IC 初期設定、RDC-IC レジスタ書き込み、RDC-IC レジスタ読み込み処理を実施するには、本 API 関数をメインループ等で繰り返し呼び出す必要があります。

API 関数：R_RSLV_Rdc_Communication(void)

3.5　励磁信号出力

角度・速度を検出するためには、レゾルバに対して励磁信号を出力する必要があります。励磁信号は矩形波で出力し、MCU と RDC-IC 間の周辺回路にて正弦波に変換します。

励磁信号は、励磁信号出力（1 本の信号）、または励磁信号と 60 度の位相差をもつ 2 本の矩形波を合成して RDC-IC に入力します。励磁周波数は、5kHz、10kHz、20kHz から選択が可能です。以下に矩形波合成時の波形イメージを示します。

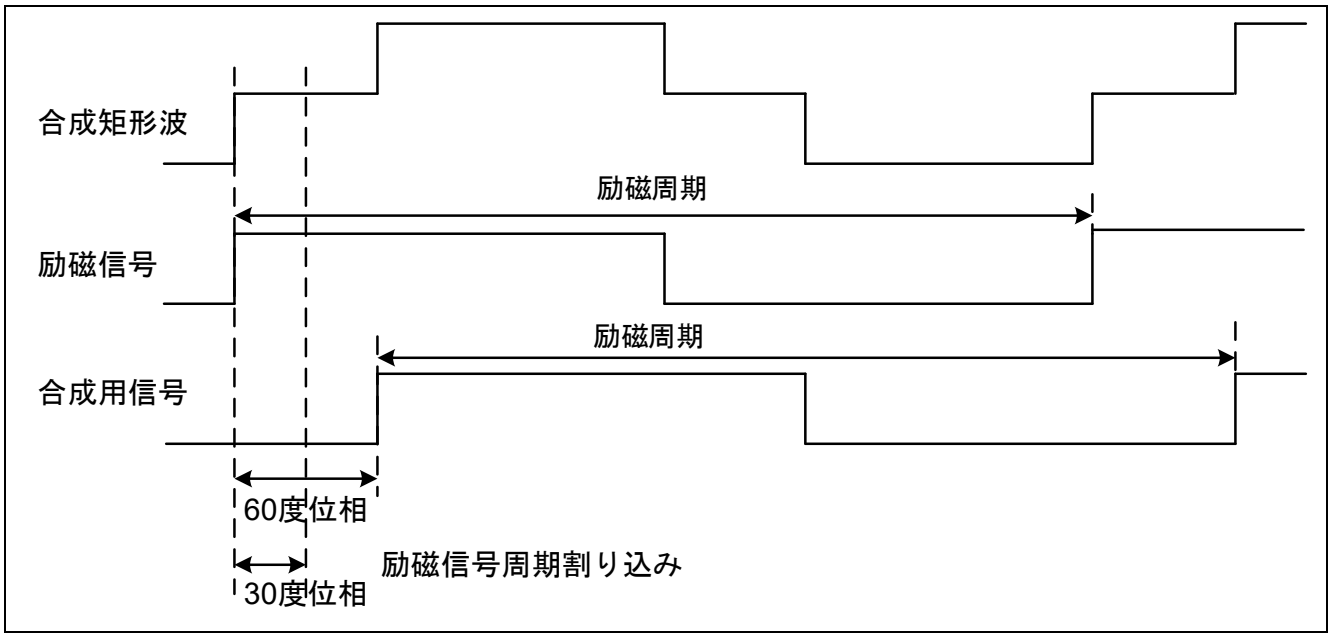

図 3.5　矩形波合成信号

3.5.1　励磁信号周期割り込み

励磁信号周期割り込みは、励磁信号の出力周期で発生させる割り込みです。励磁信号を 1 本の PWM 信号で出力する場合は、矩形波の立ち上がりで発生します。2 本の PWM 信号で出力する場合は、割り込みが発生するのは励磁信号から 30 度遅れたタイミングです。割り込み設定は、SC で生成したペリフェラル初期化処理で行います。

本割り込みは、励磁信号と角度誤差補正信号、角度誤差補正信号の Duty 更新割り込み用タイマを同期してスタートするために使用します。

また 2 本の PWM 信号を 1Ch のタイマから出力する場合、励磁信号出力をタイマのコンペアマッチでトグル出力するように設定します。この場合、割り込みは励磁信号周期に対して 2 回発生します。よって、励磁信号周期に対して 2 回目に発生する割り込みを無視してください。以下に励磁信号と割り込みタイミングの概要を示します。

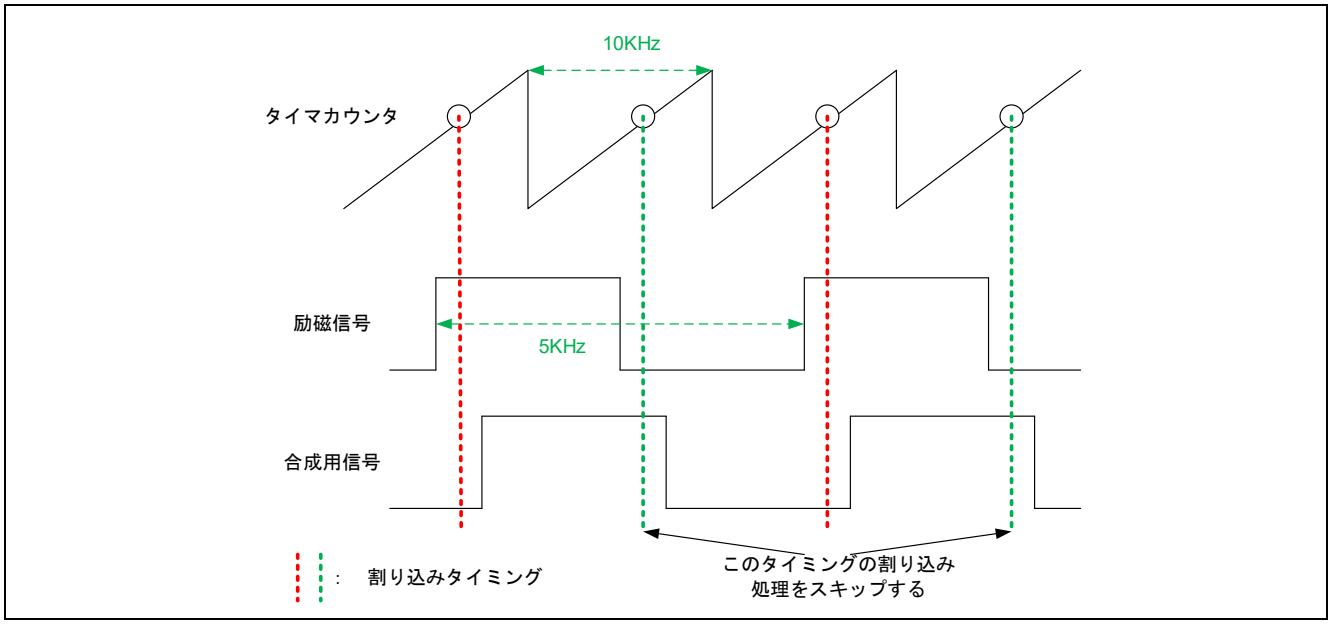

図 3.6　励磁信号（タイマ 1Ch、2 本出力）と割り込みタイミング

3.5.2　励磁信号出力開始

励磁信号を出力する場合は、励磁信号出力開始 API 関数を使用します。励磁信号出力と角度信号入力は同期してタイマをスタートする必要があります。同期スタートについては、「3.6.2　角度信号入力開始」を参照ください。

API 関数：R_RSLV_ESig_Start(void)

3.5.3　励磁信号出力停止

励磁信号出力を停止する場合は、励磁信号出力停止 API 関数を呼び出します。励磁信号出力と同期して角度信号入力も開始しているため、本 API 関数内で角度信号入力も停止します。

API 関数：R_RSLV_ESig_Stop(void)

3.5.4　励磁信号出力開始タイミング調整

レゾルバドライバは、励磁信号の割り込みタイミングを調整する機能を実装しています。モータ制御部の割り込み処理と励磁信号割り込みのタイミングをずらすことにより、処理負荷を分散する事ができます。励磁信号と角度信号入力のタイミングを調整する API 関数で設定します。

API 関数：R_RSLV_ESigCapStartTiming
(unsigned short esig_start_tcnt, unsigned short cap_start_tcnt)

以下に R_RSLV_ESigCapStartTiming の使用方法について記載します。

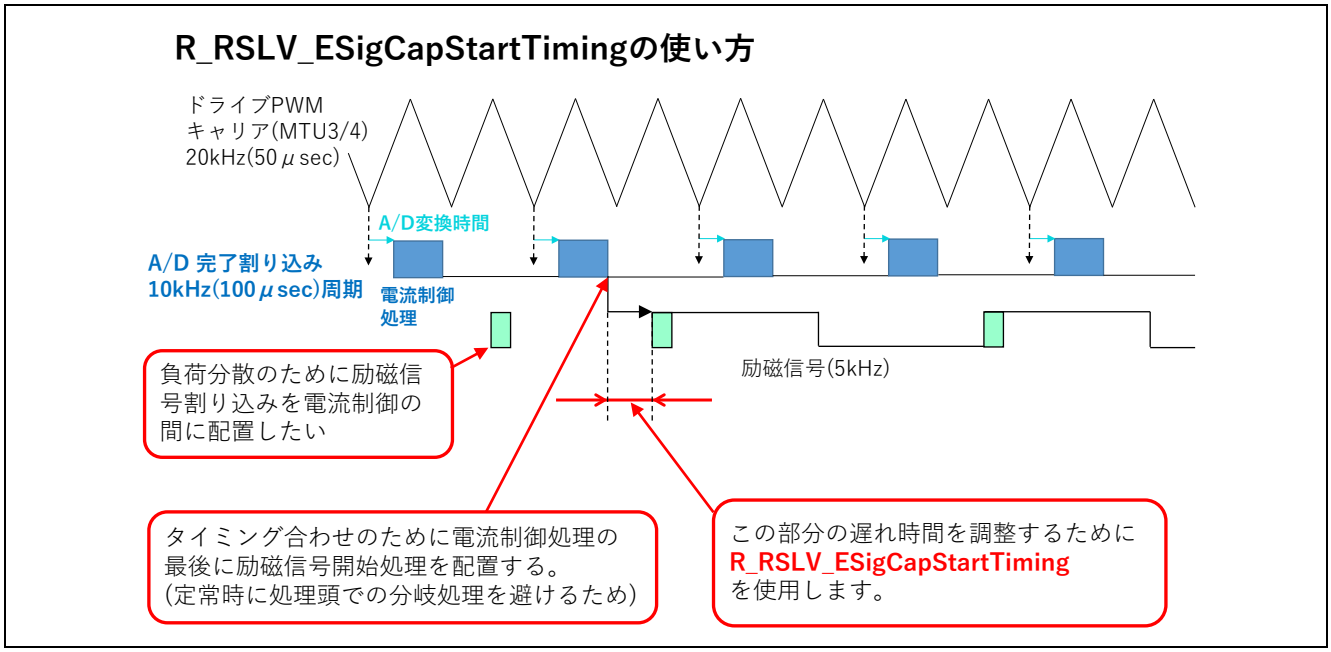

図 3.7　R_RSLV_ESigCapStartTiming（ESig）の使用例

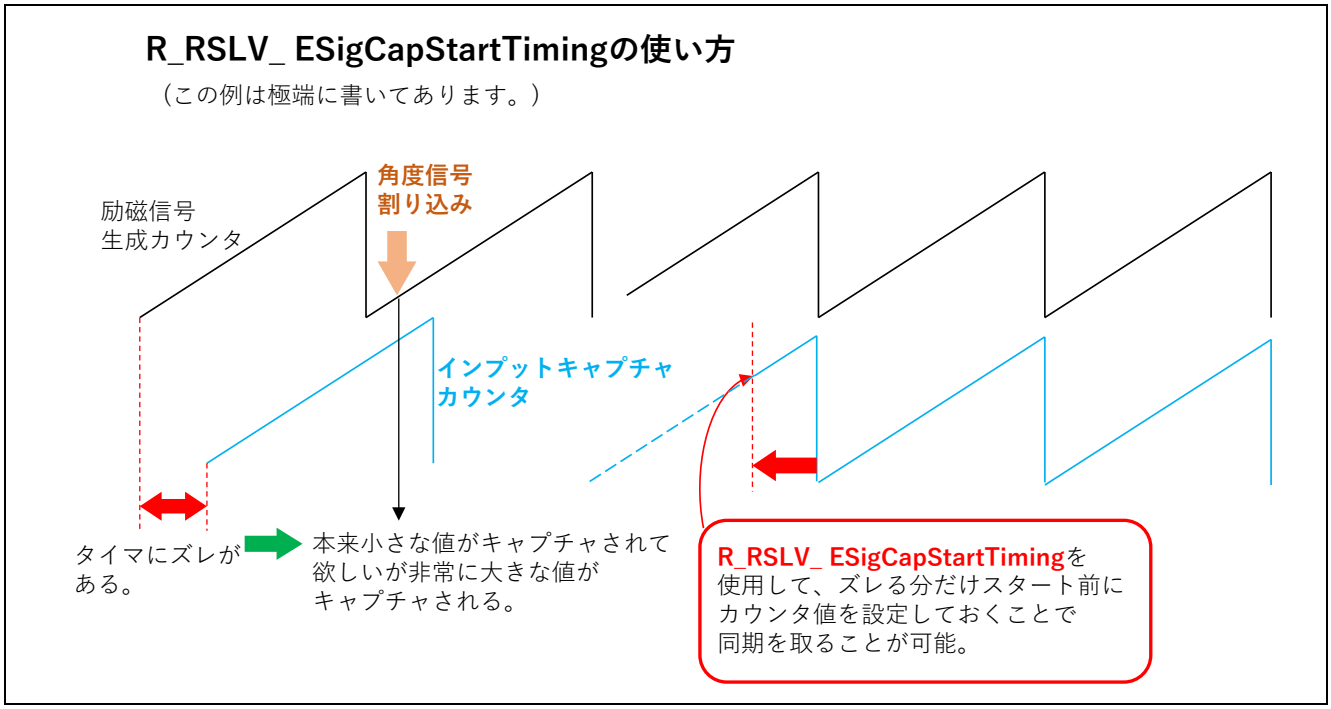

図 3.8　R_RSLV_ESigCapStartTiming（Capture）の使用例

励磁信号出力開始タイミングの設定可能範囲は下記の範囲になります。

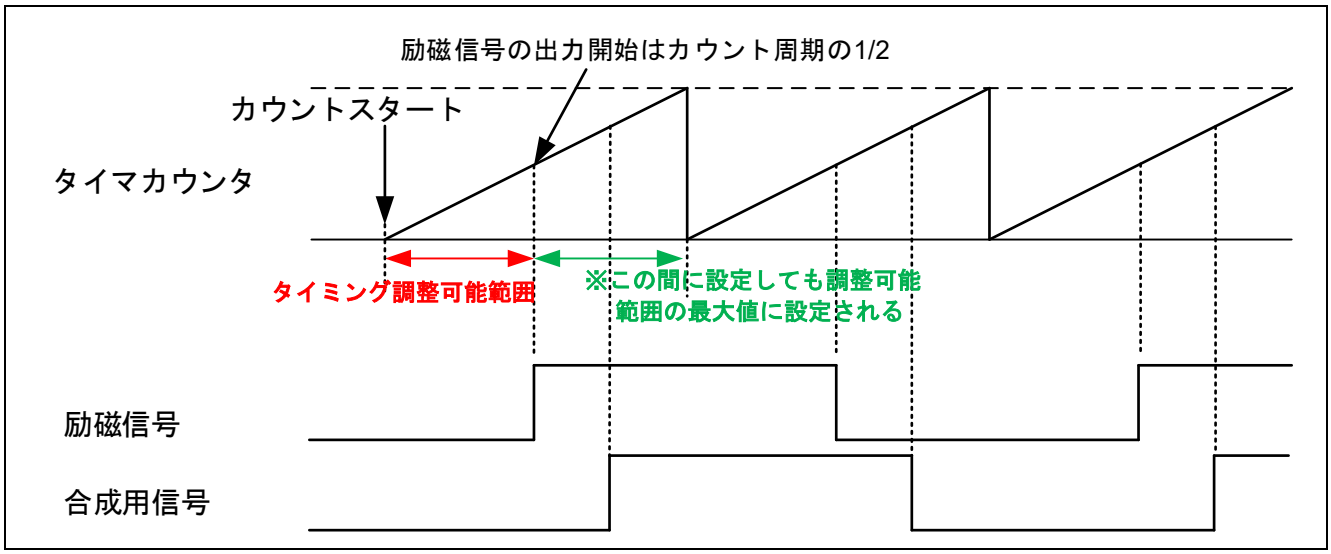

図 3.9　励磁信号出力開始タイミング調整可能範囲

3.6　角度信号入力

RDC-IC から出力される角度信号は、外部割り込み（インプットキャプチャ機能）により検出します。なお、MTU3/GPT/TPU などのインプットキャプチャ機能を持つタイマを使用します。

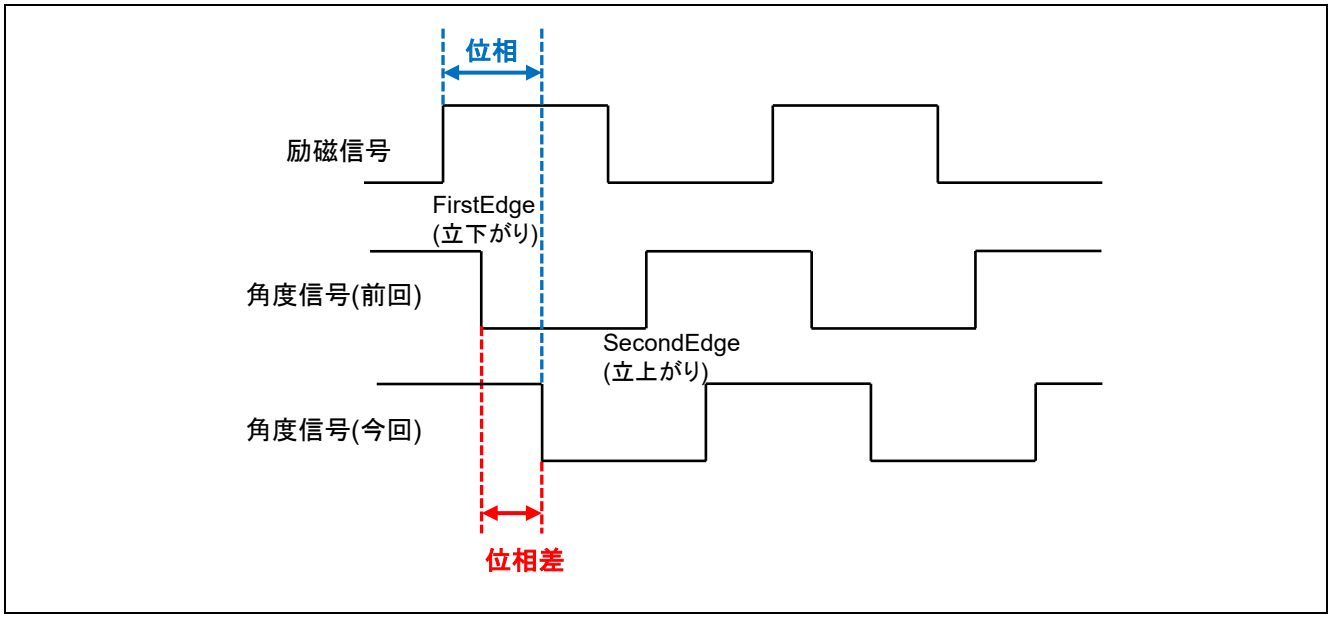

図 3.10　角度信号

角度信号の分解能は、励磁信号周波数とタイマのカウントクロック、レゾルバセンサの極対数によって決まります。

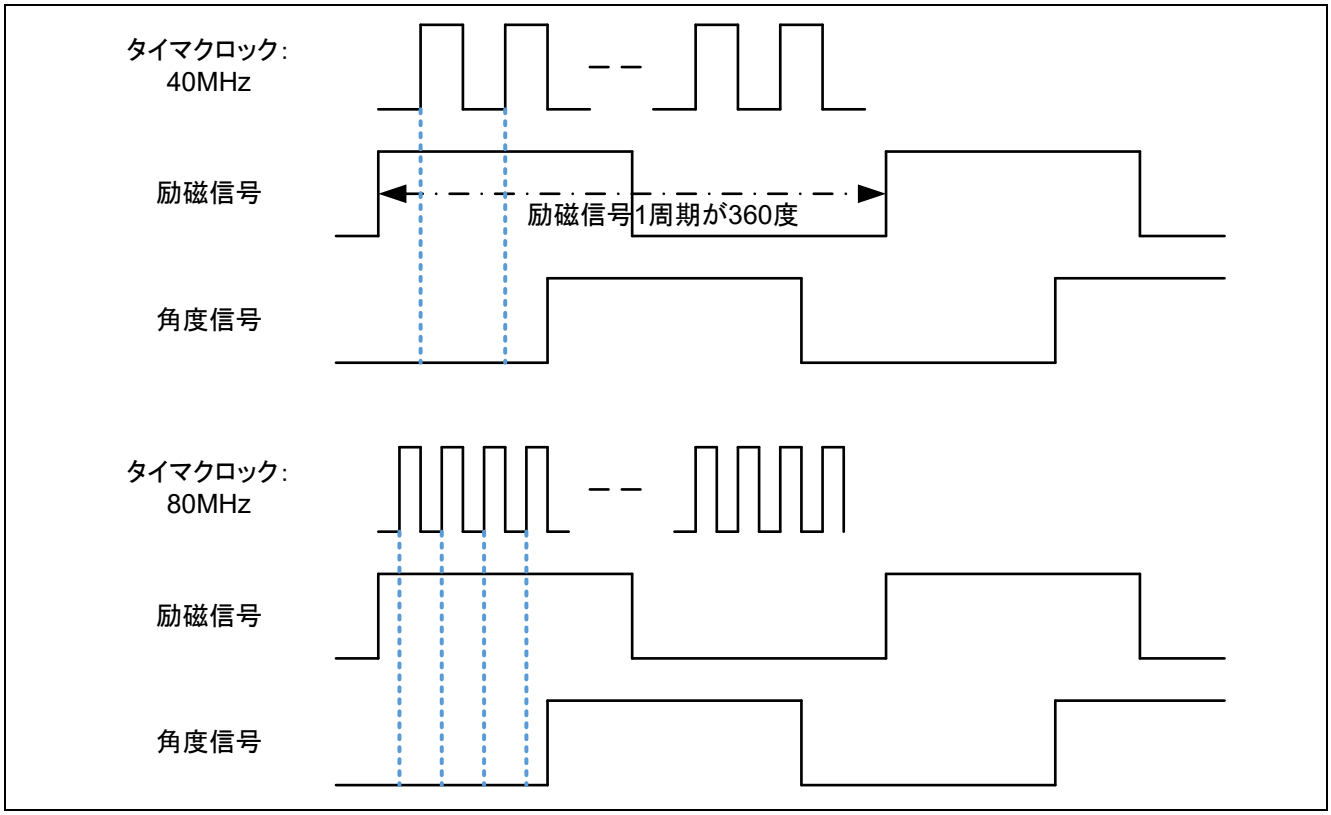

図 3.11　分解能の考え方

角度信号の分解能（機械角）は、励磁信号 1 周期の最大カウント数とレゾルバセンサの極対数を乗算して求めることができます。励磁信号 1 周期の最大カウント数は、励磁信号の出力周波数とタイマのカウントクロックによって決まります。上図を例にタイマクロック：40MHz、励磁信号：10KHz、レゾルバセンサの極対数：4X とした場合、励磁信号 1 周期の最大カウント数は 4000 カウント（= 40MHz / 10KHz）になるので、角度分解能は 16000 カウント（= 4000 カウント×4）になります。タイマクロックが 80MHz の場合は、32000 カウントになります。

3.6.1　角度信号入力割り込み

角度信号入力のエッジでインプットキャプチャ割り込みを発生させ、その時のタイマカウンタ値により角度を求めます。なお、First Edge（立ち下がり）、Second Edge（立ち上がり）、および両エッジの割り込みを設定できます。

3.6.2　角度信号入力開始

角度信号入力は、励磁信号出力と同期してタイマカウントを開始する必要があります。同期させる方法は、励磁信号出力開始 API 関数内で同時にタイマスタートする方法、MTU3 タイマ同時スタート制御 API 関数を使用する方法（MTU 使用の場合）、励磁信号割り込みのタイミングで角度検出タイマスタート API 関数を使用する方法があります。

励磁信号出力開始　API 関数：「3.5.2　励磁信号出力開始」を参照してください
角度検出タイマスタート　API 関数：R_RSLV_Capture_Start(void)
MTU3 タイマ同時スタート制御 API 関数：R_RSLV_MTU_SyncStart(unsigned char start_ch)

3.6.3　角度信号入力停止

角度信号入力を停止する場合は、励磁信号を停止する必要があるため、励磁信号出力停止 API 関数を呼び出して停止します。

API 関数：「3.5.3　励磁信号出力停止」を参照してください

3.6.4　角度信号入力開始タイミング調整

角度信号入力と励磁信号のタイマカウンタが同期してスタートしなければ、角度を正確に取得できません。本機能は、角度信号入力用タイマのカウント開始タイミングを調整する機能です。励磁信号と角度信号入力のタイミングを調整する API 関数で設定します。詳細は「3.5.4　励磁信号出力開始タイミング」を参照してください。

3.7　断線検知機能

以下に断線検知機能のシステム概要を示します。

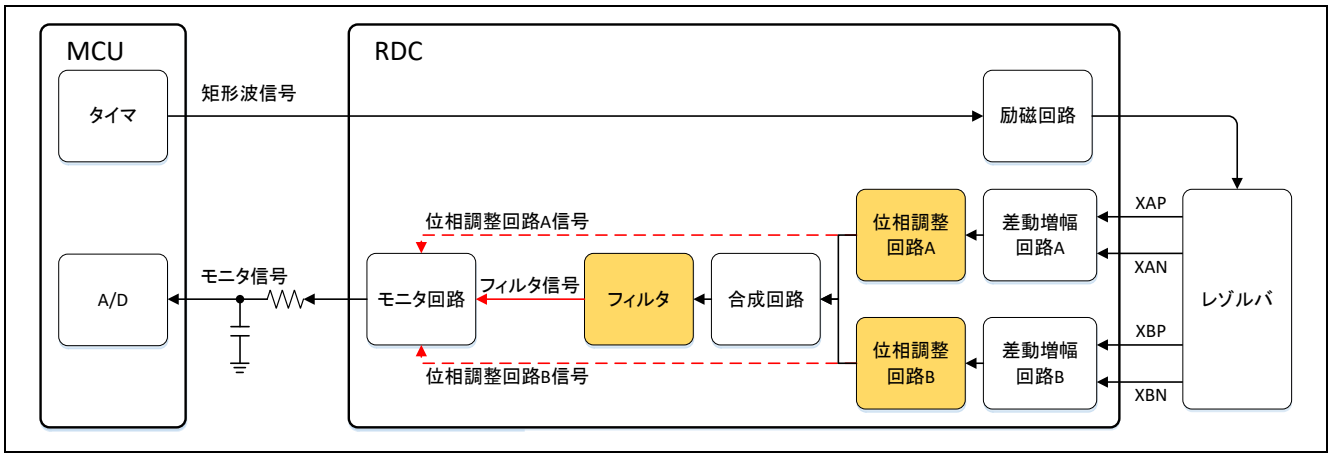

図 3.12　RDC-IC 断線検知機能システム概要

RDC-IC を用いた断線検知機能は、正常時と異常時のレゾルバ信号電圧を比較して、その差分により断線を判断します。

そのため、正常時のレゾルバ信号の電圧をあらかじめ保持する必要があります。電圧を観測する際には、モニタ回路からの出力信号を用います。観測する箇所は以下の 5 ヶ所となります。

- フィルタ信号（モニタ回路：フィルタ回路 1 出力）
- XAP 信号（モニタ回路：位相調整回路 A 出力）
- XAN 信号（モニタ回路：位相調整回路 A 出力）
- XBP 信号（モニタ回路：位相調整回路 B 出力）
- XBN 信号（モニタ回路：位相調整回路 B 出力）

3.7.1　断線検知時に使用する機能

断線検知機能は以下の機能を使用し、断線検知を行います。

3.7.1.1　RDC-IC 通信

断線検知処理に必要な RDC-IC のレジスタ設定を SPI 通信で行います。

3.7.1.2　RDC-IC モニタ信号測定

RDC-IC のモニタ信号を 12 ビット AD コンバータの連続スキャンで測定します。

3.8　ALARM 解除機能

RDC-IC が過熱状態を検知すると ALARM が発生します。この ALARM を解除する場合は、以下の API 関数を使用して解除します。ALARM 解除開始後、ALARM 解除シーケンス制御 API 関数を繰り返し呼び出して ALARM 解除します。

ALARM 解除開始 API 関数：R_RSLV_Rdc_AlarmCancelStart(void)
ALARM 解除シーケンス制御 API 関数：R_RSLV_Rdc_AlarmCancel(void)

3.9　レゾルバ信号の位相調整信号出力

RDC-IC は、レゾルバセンサから出力された 2 相信号を角度信号に変換して MCU に出力します。その際 2 相信号 A/B の位相差が 90 度にならないと正しい位相信号を MCU に出力できません。そのため MCU から RDC-IC へ位相差を調整する信号（レゾルバ位相信号 A/B の調整信号）を出力し、位相差が 90 度となるように調整します。位相調整信号は、400kHz の PWM 信号となります。

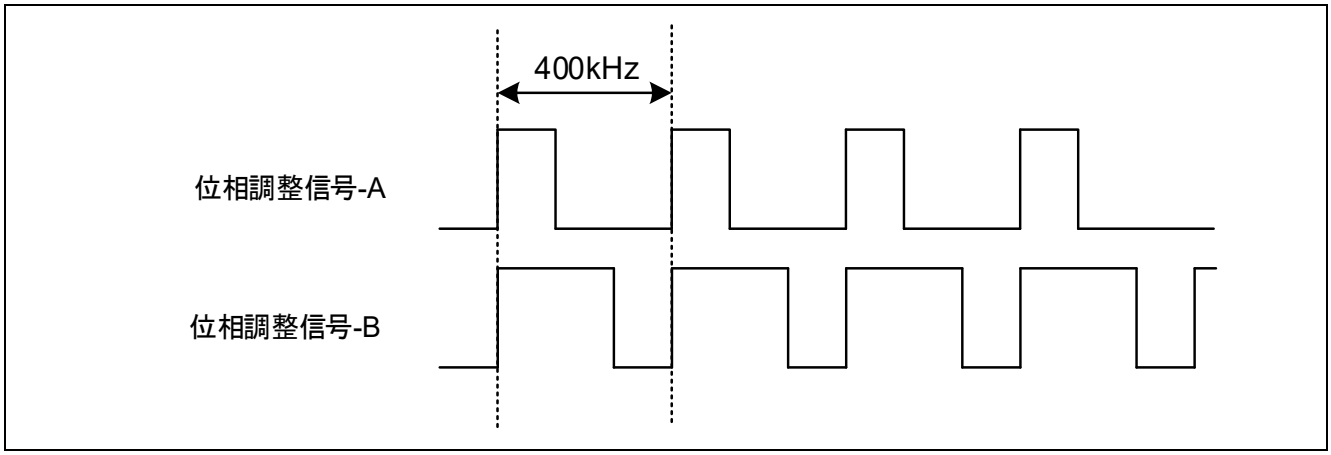

図 3.13　位相調整信号例

3.9.1　位相調整信号出力開始

位相調整信号を出力する場合は、位相調整信号出力開始 API 関数を使用します。

API 関数：R_RSLV_Phase_AdjStart(void)

3.9.2　位相調整信号出力停止

位相調整信号出力を停止する場合は、位相調整信号出力停止 API 関数を使用します。

API 関数：R_RSLV_Phase_AdjStop(void)

3.9.3　位相調整信号 Duty バッファ設定

位相調整信号の Duty を格納バッファに設定する場合は、位相調整信号 Duty バッファ設定 API 関数を使用します。

API 関数：R_RSLV_Phase_AdjUpdateBuff(unsigned short duty, unsigned char ch)

3.9.4　位相調整信号 Duty レジスタ設定

3.9.3 で設定した Duty 値を位相調整用タイマに反映する場合は、位相調整信号 Duty レジスタ設定 API 関数を使用します。

API 関数：R_RSLV_Phase_AdjUpdate(void)

3.9.5　位相調整信号 Duty バッファ読み込み

位相調整信号の Duty を読み込む場合は、位相調整信号 Duty バッファ読み込み API 関数を使用します。

API 関数：R_RSLV_Phase_AdjReadBuff(unsigned short *duty, unsigned char ch)

3.10 角度誤差補正信号出力

モータ駆動時、レゾルバセンサのアナログ誤差によって 2 相信号の合成信号に一次歪が生じ、振幅に変動が生じます。この変動は、RDC-IC から MCU へ出力する角度信号に誤差として重畳されます。

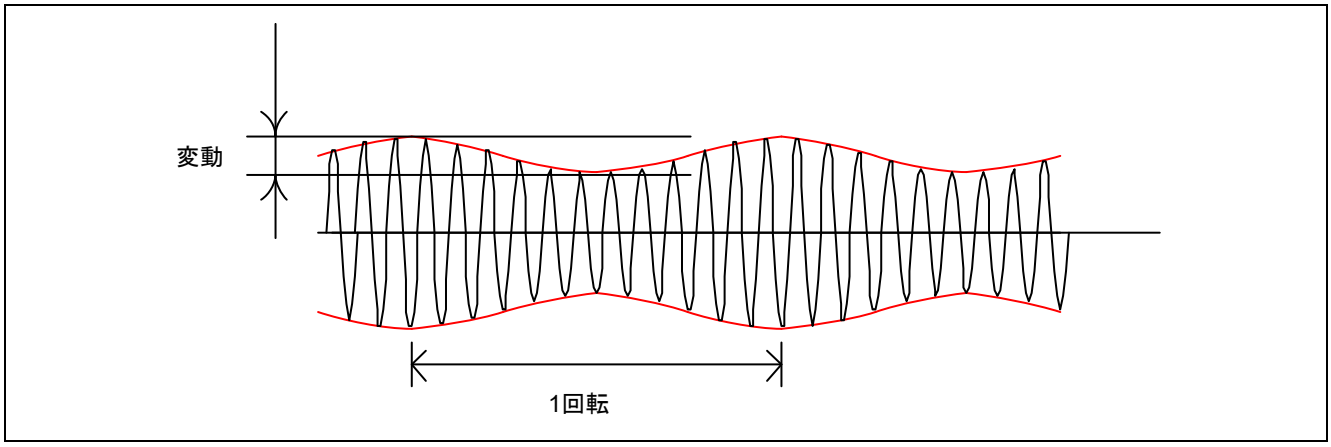

図 3.14　振幅振動（RDC-IC 内部信号）

この変動を低減させるために、MCU から RDC-IC へ補正信号を出力します。出力する補正信号は、一次歪と振幅が同じで位相が逆位相になる信号となります。

角度誤差補正信号は、キャリア周波数 200kHz、400kHz いずれか（選択可）の PWM 信号となります。この信号を LPF に通してアナログ信号（正弦波）として RDC-IC へ入力します。角度誤差補正信号は励磁信号と同期する必要があります。正弦波を構成するための Duty の更新は、励磁信号周期に対し 2 回または 4 回（選択可）で行います。以下に角度誤差補正信号出力の概念図を示します。

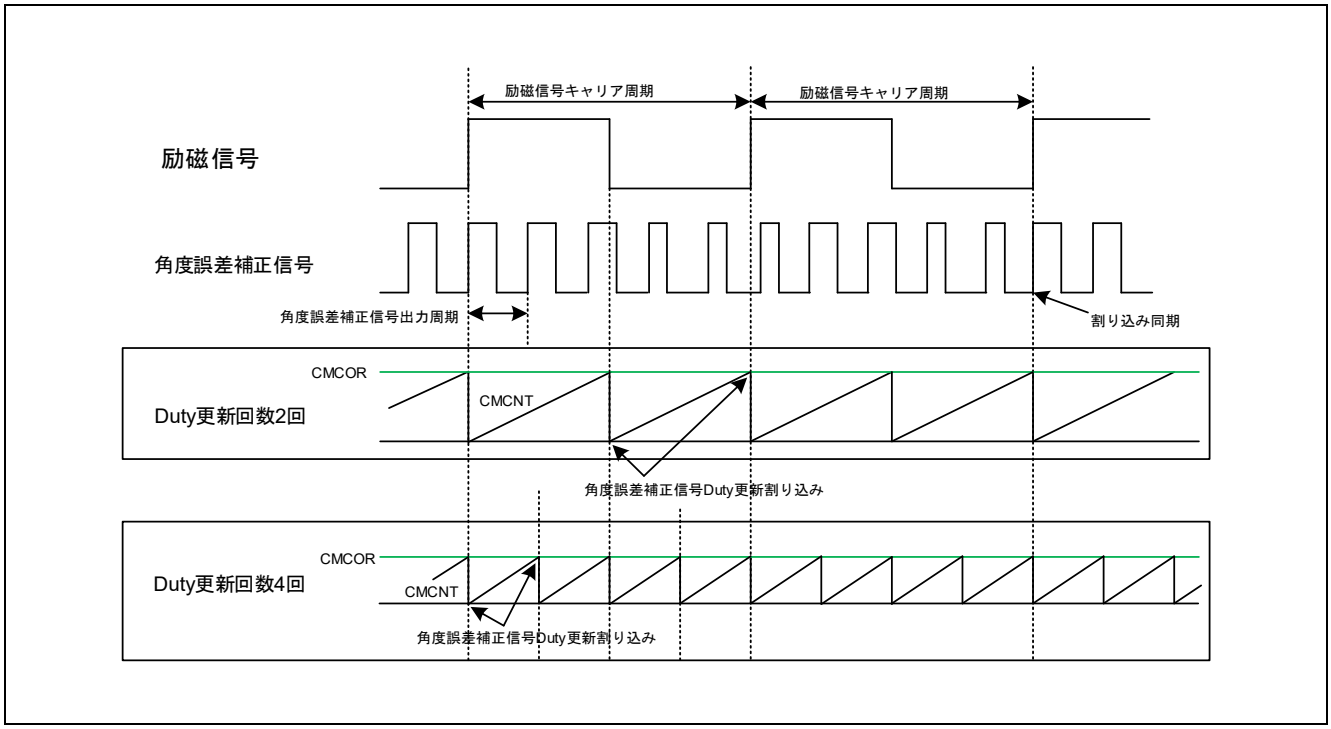

図 3.15　角度誤差補正信号出力

角度誤差補正信号（PWM 信号）の Duty 比は、Duty 更新割り込みを用いて変更します。上図では CMT を使用して Duty 更新割り込みを構成した例を示しています。CMT のカウンタ値は、更新回数が 2 回の場合は励磁信号周期の 1/2、4 回の場合は 1/4 となります。

3.10.1 角度誤差補正信号出力開始

角度誤差補正信号を出力する場合は、角度誤差補正信号出力開始 API 関数を使用します。本 API 関数で指定した位相シフト量と振幅レベル、角度誤差補正信号 Duty 更新回数を基に角度誤差補正信号の出力用タイマの設定値を計算します。位相シフト量と振幅レベルの値は、自動調整機能を使用することで取得が可能です。「3.11　自動調整機能」を参照してください。

また、角度誤差補正信号 Duty 更新用のタイマ設定値も計算します。角度誤差補正信号出力と角度誤差補正信号 Duty 更新タイマは励磁信号と同期してスタートする必要があります。同期スタートは角度誤差補正信号同期スタート API 関数を使用します。

角度誤差補正信号出力開始 API 関数：
R_RSLV_CSig_Start(unsigned short phase_diff, unsigned short amp_level)
角度誤差補正信号同期スタート API 関数：
R_RSLV_INT_CSig_SyncStart (void)

3.10.2 角度誤差補正信号出力停止

角度誤差補正信号の設定内容を更新するなど、補正信号を停止する場合は角度誤差補正信号出力停止 API 関数を用いて停止します。同時に角度誤差補正信号 Duty 更新タイマも停止します。

API 関数：R_RSLV_CSig_Stop(void)

3.10.3 角度誤差補正信号 Duty 更新設定

角度誤差補正信号の出力周波数および Duty 更新回数は、システム情報設定 API 関数を使用して設定します。この設定により、本ドライバは角度誤差補正信号の位相シフト量と振幅レベルの調整範囲を算出します。

API 関数：「3.1.2　システム情報の指定」を参照してください

3.10.4 角度誤差補正信号 Duty 更新割り込み

本割り込みは、角度誤差補正信号の Duty 値を更新するために使用する割り込み処理です。励磁信号に同期した割り込みタイミングを生成し、割り込み内で角度誤差補正信号 Duty 更新処理 API を呼び出して Duty 値を更新します。本割り込みは、励磁信号周期内に 2 回または 4 回発生します。発生回数は、システム情報設定で指定した Duty 更新回数によって決まります。

API 関数：R_RSLV_INT_CSig_UpdatePwmDuty(void)

3.11 自動調整機能

本ドライバは以下の自動調整機能を実装しています。

- レゾルバ信号ゲイン調整
- レゾルバ信号位相調整機能
- 角度誤差補正信号調整機能

3.11.1 パラメータ調整時に使用する機能

自動調整機能は以下の各ドライバ機能を使用し、パラメータ調整を行います。

- RDC-IC 通信
  RDC-IC のレジスタ操作を SPI 通信経由で行います。
- 角度誤差補正信号出力
  レゾルバセンサの一次歪誤差を補正するための信号出力です。
- 位相調整用 PWM 出力
  レゾルバセンサが出力する 2 相信号の位相差を調整するための PWM 信号です。
- 位相カウント取得
  RDC-IC により得られる角度情報です。
- RDC-IC モニタ信号測定
  RDC-IC 内部の合成信号をモニタ端子に出力し、この信号情報を元にレゾルバ信号ゲイン調整と角度誤差補正信号調整をします。モニタ信号を検出するために、アプリケーション側で 12bit AD コンバータへのアクセス関数を用意する必要があります。

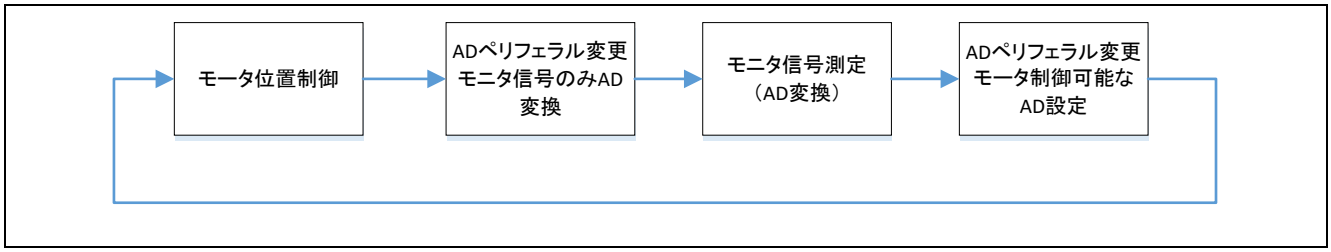

図 3.16　角度誤差補正時のモニタ信号測定処理フロー概略

- モータ位置制御
  角度誤差補正信号調整でモータ位置制御を使用します。レゾルバ角 1 度単位での制御を要求します。
- モータ速度制御
  角度誤差補正信号調整でモータ速度制御を使用します。
- 速度データ参照
  角度誤差補正信号調整で速度制御時に速度データ（単位：rad/s）を参照します。

3.11.2 レゾルバ信号ゲイン・位相調整機能

3.11.2.1 レゾルバ信号ゲイン調整

以下にレゾルバ信号ゲイン調整のブロック図を示します。

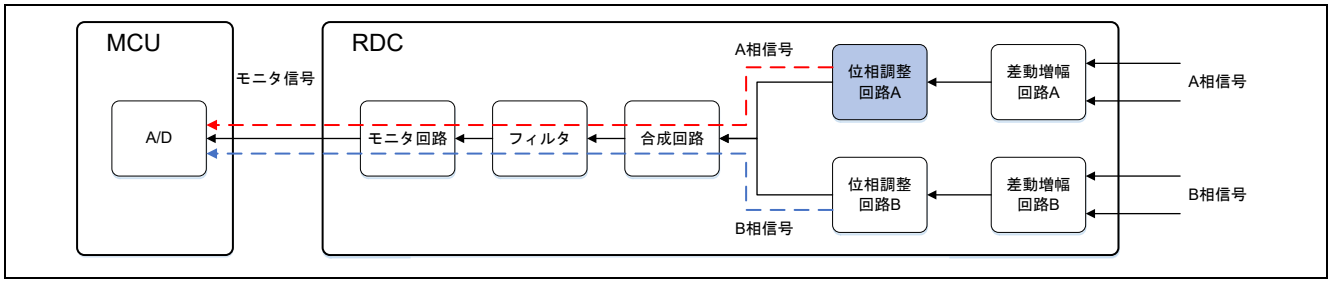

図 3.17　レゾルバ信号ゲイン調整ブロック図

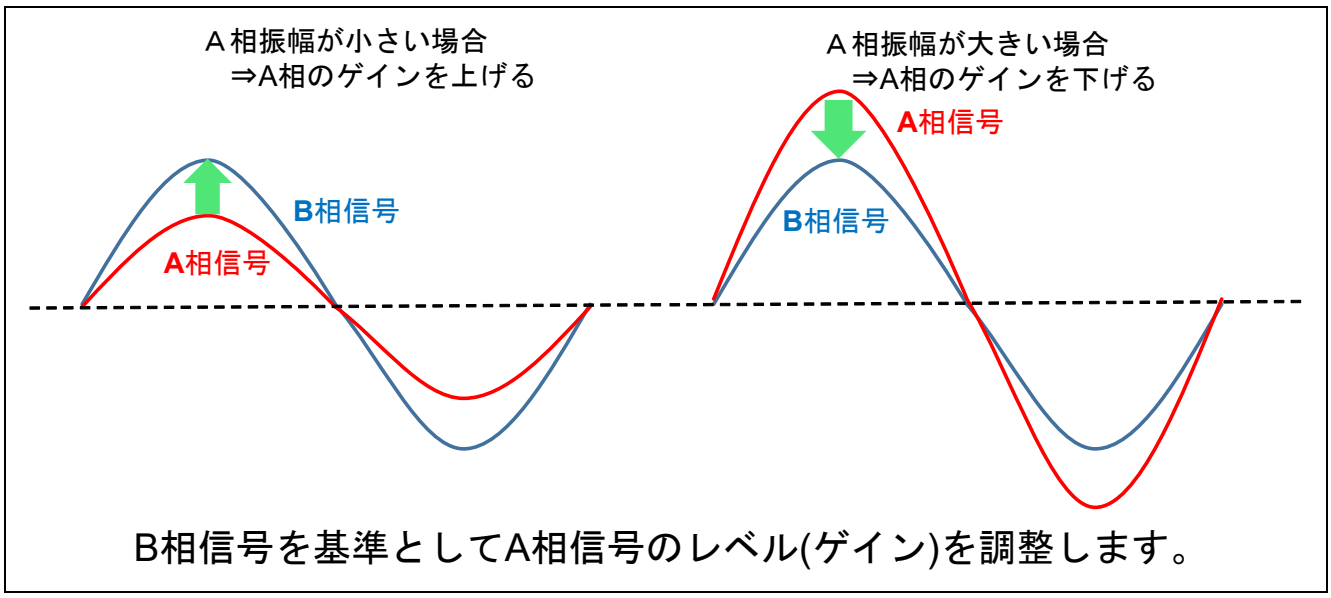

図 3.18　レゾルバ信号ゲイン調整

A 相/B 相信号に振幅差がある場合、レゾルバからの角度情報に誤差が生じてしまいます。そこで、レゾルバ信号ゲイン調整では、A 相/B 相信号の振幅が同一レベルになるように RDC-IC を調整します。A 相信号と B 相信号の振幅の相対誤差が±0.28%以内になるように調整を行います。

3.11.2.2 レゾルバ信号位相調整

以下にレゾルバ信号位相調整のブロック図を示します。

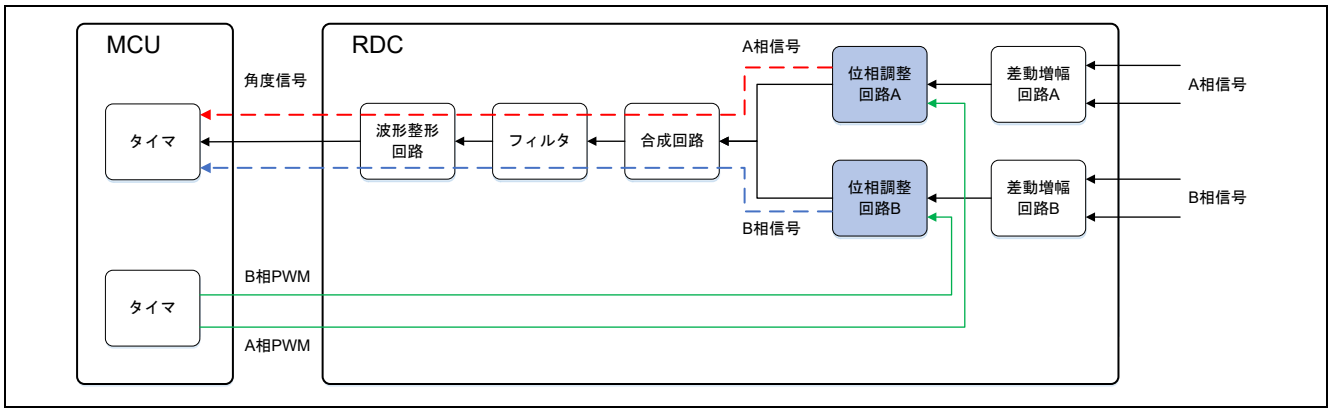

図 3.19　レゾルバ信号位相調整ブロック図

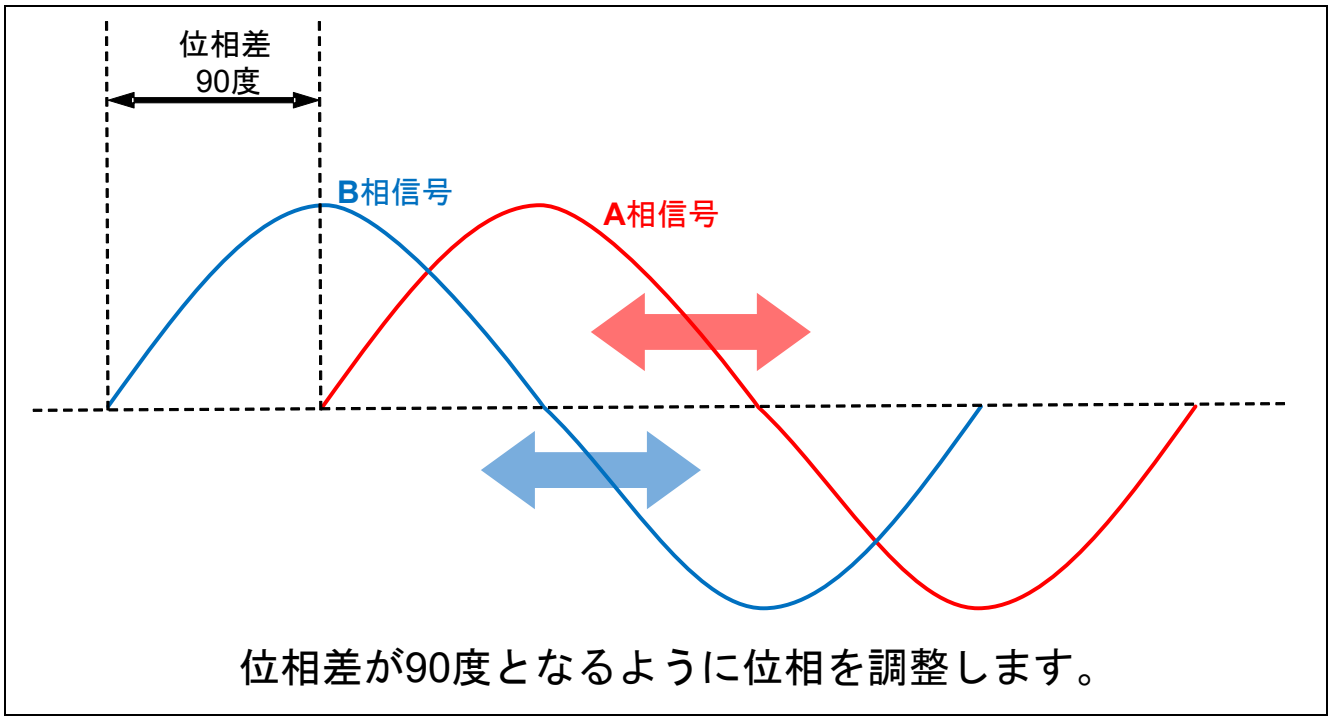

図 3.20　レゾルバ信号位相調整

レゾルバ信号位相調整では、A 相/B 相信号用の位相調整信号の duty を変更して、A 相信号と B 相信号の位相差が 90 度±0.3%（±0.27 度）以内になるように調整を行います。

Duty 調整範囲：5～90%（1%単位）

3.11.3 角度誤差補正信号調整機能

以下に角度誤差補正信号調整機能のブロック図を示します。

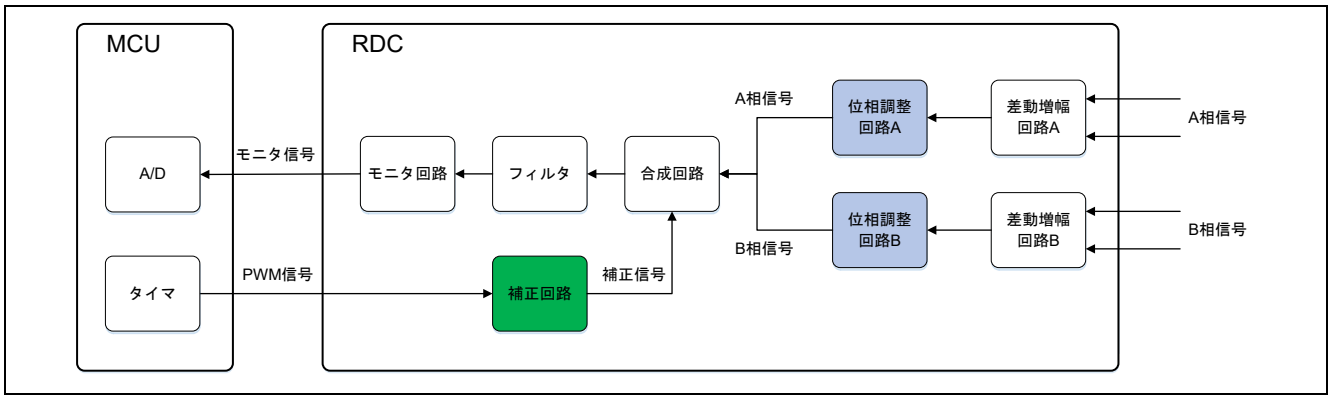

図 3.21　角度誤差補正信号調整機能ブロック図

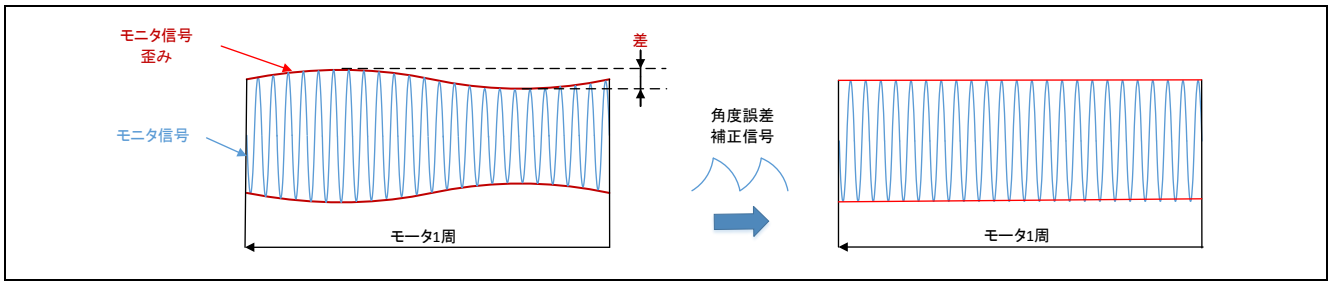

図 3.22　角度誤差補正信号調整機能

角度誤差補正信号調整機能では、補正回路に入力する角度誤差補正信号の位相シフト量と振幅値を調整します。調整した補正信号を RDC-IC 内で角度信号に重畳し、レゾルバセンサのアナログ誤差に起因する角度誤差を補正します。角度誤差補正信号の位相シフト量、振幅値の入力範囲は以下の通りです。

表 3-1　位相シフト量の入力範囲（0~下記設定値）

MCU		RX23T	RX24T			RX66T / RX72T			RX72M			
ペリフェラル		CMT	MTU	GPT	CMT	MTU	GPT	CMT	MTU	GPT	TPU	CMT
ソースクロック設定（MHz）		5	80	80	5	160	160	5	120	120	60	7.5
励磁周波数	5kHz	999	15999	15999	999	31999	31999	999	23999	23999	11999	1499
	10kHz	499	7999	7999	499	15999	15999	499	11999	11999	5999	749
	20kHz	249	3999	3999	249	7999	7999	249	5999	5999	2999	374

【注】 CMT はソースクロックの 8 分周で設定します

表 3-2　振幅値の入力範囲（0~下記設定値）

MCU		RX23T	RX24T		RX66T / RX72T		RX72M		
ペリフェラル		MTU	MTU	GPT	MTU	GPT	MTU	GPT	TPU
ソースクロック設定(MHz)		40	80	80	160	160	120	120	60
角度誤差補正信号周期	200kHz	199	399	399	799	799	599	599	299
	400kHz	99	199	199	399	399	299	299	149

3.11.3.1 フィルタ回路による遅れ位相

補正回路に入力する角度誤差補正信号の位相シフト量を正しく調整するためには、RDC-IC 周辺回路に実装されているフィルタによる遅れ位相を考慮する必要があります。

ユーザが遅れ位相の値を初期値から変更する場合は、遅れ位相設定用 API 関数を使用します。

API 関数：R_RSLV_ADJST_SetFilterDelay(float bpf_delay_deg, float csig_delay_deg)
詳細は「7.9.3　フィルタ回路による遅れ位相」を参照してください。

3.12 タイミングチャート（励磁信号、角度信号入力、角度誤差補正信号）

励磁信号、角度信号入力、角度誤差補正信号に関連するタイマと波形のタイミングチャートを以下に示します。励磁割り込みのタイミングで角度信号入力用タイマ、角度誤差補正信号出力、角度誤差補正信号 Duty 更新タイマをスタートさせてください。

各設定内容の詳細は、「3.1　ドライバ初期化処理」の項以降を参照してください。

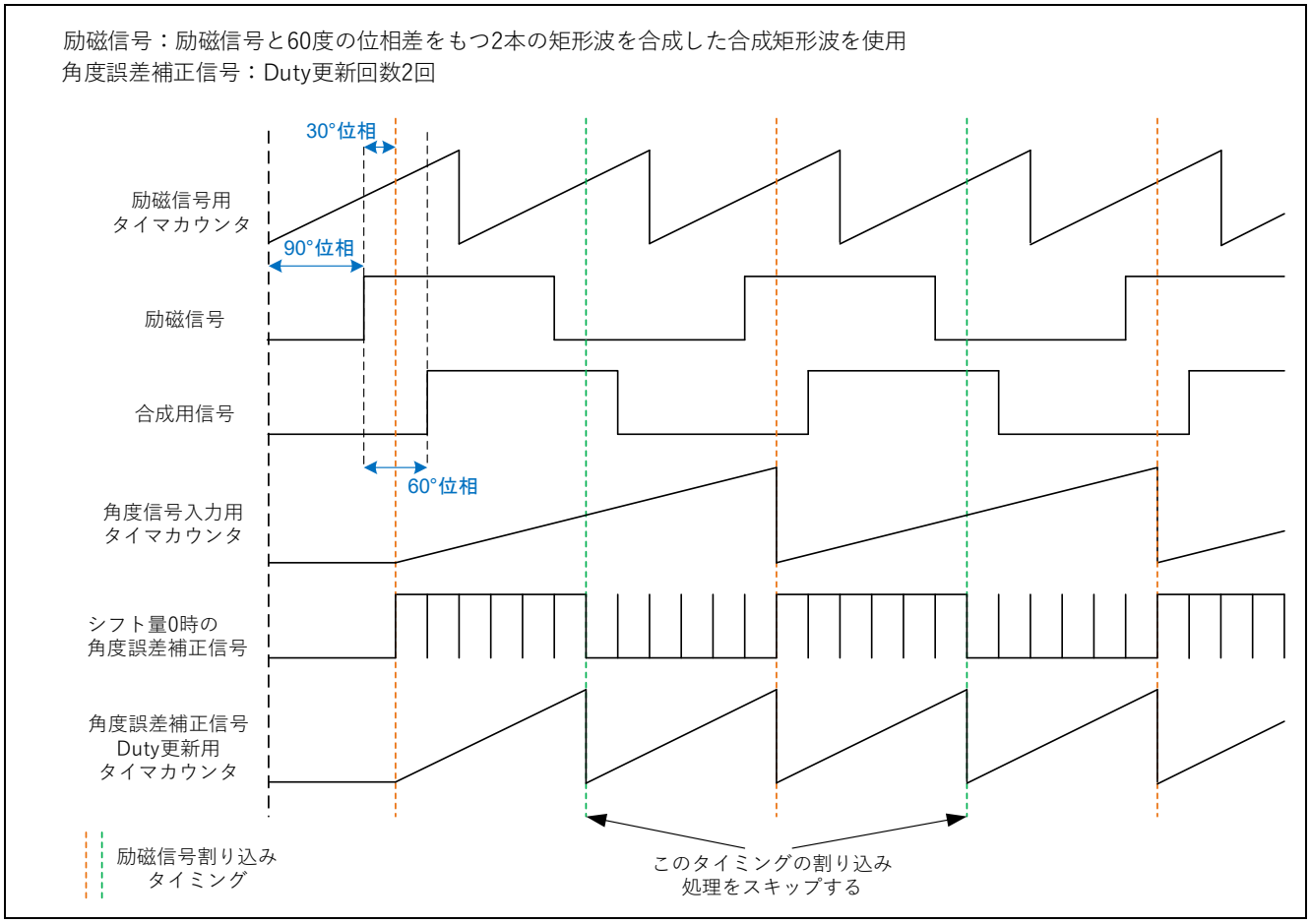

図 3.23　タイマカウンタ及び波形のタイミングチャート

4.　ソフトウェア構成

4.1　フォルダ・ファイル構成

本ドライバのプロジェクトフォルダ、ファイルの構成を以下に示します。

表 4-1　フォルダ・ファイル構成一覧

¥rx_rslv_drv			
	¥api*		
		r_rslv_api.h	RDC-IC ドライバ用ヘッダーファイル （パラメータ構造体、API 関数、共通の定数定義ファイル）
	¥lib		
		rdc_driver_library_RX.lib	ライブラリファイル
¥sample¥PeripheralCode_XXX (XXX : MCU 製品名)			
	¥src¥smc_gen¥Config_peri_func		
		Config_peri_func.c Config_peri_func_user.c Config_peri_func.h	SC 生成サンプルソース peri：ペリフェラル名（MTU0、TMR0 など） func：機能名（Esig、Csig など） ※本命名は SC 生成関数名も同様
	¥src¥sample_src		
		r_sample_func_table.c	関数テーブル用サンプルソース

【注】 本ドライバはライブラリとして提供します。¥api 配下のファイルはライブラリアクセス用として提供するファイルです。

5. ペリフェラル設定

5.1 各ドライバ機能のマクロ定義名一覧

本ドライバが実装する各ドライバ機能のマクロ定義名一覧を以下に示します。

表 5-1　各ドライバ機能のマクロ定義名一覧

定義名	定義値	内容
F_ESIG1	0	励磁信号（単相出力）設定用
F_ESIG2_1	1	励磁信号（合成出力 0 度・2 タイマ使用）設定用
F_ESIG2_2	2	励磁信号（合成出力 60 度・2 タイマ使用）設定用
F_ESIG12	3	励磁信号（合成出力・1 タイマ使用）設定用
F_CSIG	4	角度誤差補正信号出力設定用
F_PHASE_A	5	位相調整信号出力設定用（A 相）
F_PHASE_B	6	位相調整信号出力設定用（B 相）
F_PHASE_AB	7	位相調整信号出力設定用（1 タイマ AB 相出力用）
F_CAPTURE	8	角度信号入力設定用
F_CSIG_UPD_TIMER	9	角度誤差補正 Duty 更新タイマ設定用
F_RDC_COM	10	RDC-IC 通信設定用
F_RDC_CLK	11	RDC-IC クロック出力設定用

5.2　各ドライバ機能に割り当てるペリフェラル一覧（推奨）

各ドライバ機能に割り当てるペリフェラルの一覧（推奨）を以下に示します。

表 5-2　各ドライバ機能に割り当てられるペリフェラル組み合わせ一覧（RX23T）

			各ドライバ機能の定義名											
			F_ESIG1	F_ESIG2_1	F_ESIG2_2	F_ESIG12	F_CSIG	F_PHASE_A	F_PHASE_B	F_PHASE_AB	F_CAPTURE	F_CSIG_UPD_TIMER	F_RDC_COM	F_RDC_CLK
ペリフェラル	TMR	TMR0						○	○					○
		TMR1						○	○					○
		TMR2						○	○					○
		TMR3						○	○					○
	MTU	MTU0	○			○								
		MTU1					○				○			
		MTU2					○				○			
	CMT	CMT0										○		
		CMT1										○		
		CMT2										○		
		CMT3										○		
	RSPI	RSPI0											○	
	SCI	SCI1											○	
		SCI5											○	

表 5-3　各ドライバ機能に割り当てられるペリフェラル組み合わせ一覧（RX24T）

			各ドライバ機能の定義名											
			F_ESIG1	F_ESIG2_1	F_ESIG2_2	F_ESIG12	F_CSIG	F_PHASE_A	F_PHASE_B	F_PHASE_AB	F_CAPTURE	F_CSIG_UPD_TIMER	F_RDC_COM	F_RDC_CLK
ペリフェラル	TMR	TMR0						○	○					○
		TMR1						○	○					○
		TMR2						○	○					○
		TMR3						○	○					○
		TMR4						○	○					○
		TMR5						○	○					○
		TMR6						○	○					○
		TMR7						○	○					○
	MTU	MTU0	○	○	○	○	○	○	○	○	○	○		○
		MTU1	○	○	○		○	○	○		○	○		○
		MTU2	○	○	○		○	○	○		○	○		○
		MTU6	○	○	○		○	○	○	○	○	○		○
		MTU7	○	○	○		○	○	○	○	○	○		○
		MTU9	○	○	○	○	○	○	○	○	○	○		○
	GPT	GPT0	○	○	○	○	○	○	○	○	○	○		○
		GPT1	○	○	○	○	○	○	○	○	○	○		○
		GPT2	○	○	○	○	○	○	○	○	○	○		○
		GPT3	○	○	○	○	○	○	○	○	○	○		○
	CMT	CMT0										○		
		CMT1										○		
		CMT2										○		
		CMT3										○		
	RSPI	RSPI0											○	
	SCI	SCI1											○	
		SCI5											○	
		SCI6											○	

表 5-4　各ドライバ機能に割り当てられるペリフェラル組み合わせ一覧（RX66T）

			各ドライバ機能の定義名											
			F_ESIG1	F_ESIG2_1	F_ESIG2_2	F_ESIG12	F_CSIG	F_PHASE_A	F_PHASE_B	F_PHASE_AB	F_CAPTURE	F_CSIG_UPD_TIMER	F_RDC_COM	F_RDC_CLK
ペリフェラル	TMR	TMR0						○	○					○
		TMR1						○	○					○
		TMR2						○	○					○
		TMR3						○	○					○
		TMR4						○	○					○
		TMR5						○	○					○
		TMR6						○	○					○
		TMR7						○	○					○
	MTU	MTU0	○	○	○	○	○	○	○	○	○	○		○
		MTU1	○	○	○		○	○	○		○	○		○
		MTU2	○	○	○		○	○	○		○	○		○
		MTU6	○	○	○		○	○	○	○	○	○		○
		MTU7	○	○	○		○	○	○	○	○	○		○
		MTU9	○	○	○	○	○	○	○	○	○	○		○
	GPT	GPT0	○	○	○	○	○	○	○	○	○	○		○
		GPT1	○	○	○	○	○	○	○	○	○	○		○
		GPT2	○	○	○	○	○	○	○	○	○	○		○
		GPT3	○	○	○	○	○	○	○	○	○	○		○
		GPT4	○	○	○	○	○	○	○	○	○	○		○
		GPT5	○	○	○	○	○	○	○	○	○	○		○
		GPT6	○	○	○	○	○	○	○	○	○	○		○
		GPT7	○	○	○	○	○	○	○	○	○	○		○
		GPT8	○	○	○	○	○	○	○	○	○	○		○
		GPT9	○	○	○	○	○	○	○	○	○	○		○
	CMT	CMT0										○		
		CMT1										○		
		CMT2										○		
		CMT3										○		
	RSPI	RSPI0											○	
	SCI	SCI1											○	
		SCI5											○	
		SCI6											○	
		SCI8											○	
		SCI9											○	
		SCI11											○	
		SCI12											○	

表 5-5　各ドライバ機能に割り当てられるペリフェラル組み合わせ一覧（RX72M）[1/2]

			各ドライバ機能の定義名											
			F_ESIG1	F_ESIG2_1	F_ESIG2_2	F_ESIG12	F_CSIG	F_PHASE_A	F_PHASE_B	F_PHASE_AB	F_CAPTURE	F_CSIG_UPD_TIMER	F_RDC_COM	F_RDC_CLK
ペリフェラル	TMR	TMR0						○	○					○
		TMR1						○	○					○
		TMR2						○	○					○
		TMR3						○	○					○
	MTU	MTU0	○	○	○	○	○	○	○	○	○	○		○
		MTU1	○	○	○		○	○	○		○	○		○
		MTU2	○	○	○		○	○	○		○	○		○
		MTU6	○	○	○	○	○	○	○	○	○	○		○
		MTU7	○	○	○	○	○	○	○	○	○	○		○
		MTU8	○	○	○	○					○	○		○
	GPT	GPT0	○	○	○	○	○	○	○	○	○	○		○
		GPT1	○	○	○	○	○	○	○	○	○	○		○
		GPT2	○	○	○	○	○	○	○	○	○	○		○
		GPT3	○	○	○	○	○	○	○	○	○	○		○
	TPU	TPU0	○	○	○	○	○	○	○	○	○	○		○
		TPU1	○	○	○		○	○	○		○	○		○
		TPU2	○	○	○		○	○	○		○	○		○
		TPU3	○	○	○	○	○	○	○	○	○	○		○
		TPU4	○	○	○		○	○	○		○	○		○
		TPU5	○	○	○		○	○	○		○	○		○
	CMT	CMT0										○		
		CMT1										○		
		CMT2										○		
		CMT3										○		

表 5-6　各ドライバ機能に割り当てられるペリフェラル組み合わせ一覧（RX72M）[2/2]

			各ドライバ機能の定義名											
			F_ESIG1	F_ESIG2_1	F_ESIG2_2	F_ESIG12	F_CSIG	F_PHASE_A	F_PHASE_B	F_PHASE_AB	F_CAPTURE	F_CSIG_UPD_TIMER	F_RDC_COM	F_RDC_CLK
ペリフェラル	RSPI	RSPI0											○	
		RSPI1											○	
		RSPI2											○	
	SCI	SCI0											○	
		SCI1											○	
		SCI2											○	
		SCI3											○	
		SCI4											○	
		SCI5											○	
		SCI6											○	
		SCI7											○	
		SCI8											○	
		SCI9											○	
		SCI10											○	
		SCI11											○	
		SCI12											○	

表 5-7　各ドライバ機能に割り当てられるペリフェラル組み合わせ一覧（RX72T）

			各ドライバ機能の定義名											
			F_ESIG1	F_ESIG2_1	F_ESIG2_2	F_ESIG12	F_CSIG	F_PHASE_A	F_PHASE_B	F_PHASE_AB	F_CAPTURE	F_CSIG_UPD_TIMER	F_RDC_COM	F_RDC_CLK
ペリフェラル	TMR	TMR0						○	○					○
		TMR1						○	○					○
		TMR2						○	○					○
		TMR3						○	○					○
		TMR4						○	○					○
		TMR5						○	○					○
		TMR6						○	○					○
		TMR7						○	○					○
	MTU	MTU0	○	○	○	○	○	○	○	○	○	○		○
		MTU1	○	○	○		○	○	○		○	○		○
		MTU2	○	○	○		○	○	○		○	○		○
		MTU6	○	○	○		○	○	○	○	○	○		○
		MTU7	○	○	○		○	○	○	○	○	○		○
		MTU9	○	○	○	○	○	○	○	○	○	○		○
	GPT	GPT0	○	○	○	○	○	○	○	○	○	○		○
		GPT1	○	○	○	○	○	○	○	○	○	○		○
		GPT2	○	○	○	○	○	○	○	○	○	○		○
		GPT3	○	○	○	○	○	○	○	○	○	○		○
		GPT4	○	○	○	○	○	○	○	○	○	○		○
		GPT5	○	○	○	○	○	○	○	○	○	○		○
		GPT6	○	○	○	○	○	○	○	○	○	○		○
		GPT7	○	○	○	○	○	○	○	○	○	○		○
		GPT8	○	○	○	○	○	○	○	○	○	○		○
		GPT9	○	○	○	○	○	○	○	○	○	○		○
	CMT	CMT0										○		
		CMT1										○		
		CMT2										○		
		CMT3										○		
	RSPI	RSPI0											○	
	SCI	SCI1											○	
		SCI5											○	
		SCI6											○	
		SCI8											○	
		SCI9											○	
		SCI11											○	
		SCI12											○	

5.3　SC による各ドライバ機能の設定

各ドライバ機能にアサインするペリフェラル機能を初期化するためには、SC で出力した初期化関数を使用します。以下に SC 設定例を示します。なお、MCU は RX72M、設定するシステム情報は下記とします。

＜条件＞

励磁信号周波数：5kHz
角度誤差補正信号周波数：200kHz
角度誤差補正信号 Duty 更新回数：2 回

5.3.1　励磁信号出力

励磁信号出力機能のペリフェラルアサインは、MTU、GPT、TPU（TPU は RX72M のみ）を推奨します。また、励磁信号の出力方式には、単相出力・合成出力方式があります。以下に 1 チャネルのタイマで合成出力方式を選択する場合の SC 設定例を示します。

5.3.1.1　MTU を使用した場合の SC 設定例

表 5-8　励磁信号出力コンポーネント選択（MTU）

コンポーネント選択	選択内容
コンポーネント選択	ノーマルモードタイマ
コンフィグレーション名	Config_MTU0_Esig12
インプットキャプチャ／アウトプットコンペア端子	4 端子
リソース	MTU0

表 5-9　励磁信号周波数 5kHz、出力端子を MTIOC0A, 0B とした場合

項目	設定内容
グループ：同期動作設定	本項目は設定不要
グループ：TCNT0 カウンタ設定	下記を設定する
カウンタクリア要因	TGRD0 コンペアマッチ／インプットキャプチャ
カウンタクロックの選択	PCLK
グループ：外部クロック端子設定	本項目は設定不要
グループ：ジェネラルレジスタ	下記を設定する。それ以外は設定不要
TGRA	アウトプットコンペアレジスタ　50μs
TGRB	アウトプットコンペアレジスタ　83.33μs
TGRC	アウトプットコンペアレジスタ　66.67μs
TGRD	アウトプットコンペアレジスタ　100μs
グループ：入出力端子の設定	下記を設定する。それ以外は設定不要 （端子出力は無効）
MTIOC0A 端子	端子初期出力は 0、コンペアマッチでトグル出力
MTIOC0B 端子	端子初期出力は 0、コンペアマッチでトグル出力
グループ：ノイズフィルタ設定	本項目は設定不要
グループ：A/D 変換開始トリガ設定	本項目は設定不要
グループ：割り込み設定	下記を設定する。それ以外は設定不要
TGRC	許可 優先順位：レベル 11

ESIG12 で励磁信号を出力する場合、タイマカウンタのカウント周期の 1/2 の位置に励磁信号の出力開始を設定します。本例の場合、カウント周期が 100μs ですので励磁信号の出力開始は 50μs の位置となります。

5.3.1.2　GPT を使用した場合の SC 設定例

表 5-10　励磁信号出力コンポーネント選択（GPT）

コンポーネント選択	選択内容
コンポーネント選択	汎用 PWM タイマ
コンフィグレーション名	Config_GPT0_Esig12
作業モード	のこぎり波 PWM モード
リソース	GPT0

表 5-11　励磁信号周波数 5kHz、出力端子を GTIOC0A, 0B とした場合

項目	設定内容
グループ：カウント設定	下記を設定する
クロックソース	PCLKA (120.000 MHz)
タイマ動作周期	100μs
周期レジスタ値	11999
バッファ動作	バッファ動作しない
カウント方向	アップカウント
カウンタ初期値	0
インプットキャプチャはカウント停止時に動作	設定不要
グループ：コンペアマッチレジスタ、端子設定	—
TAB：GTCCRA	下記を設定する
GTCCRA 機能	コンペアマッチ：5999
バッファ動作	バッファ動作しない
GTIOC0A 端子機能	PWM 出力端子
ノイズフィルタ	設定不要
GTIOC0A 端子出力デューティ	コンペアマッチによって決定
GTIOC0A 端子ネゲート制御	禁止
開始／停止時の出力レベル	開始時 0 出力、停止時 0 出力
コンペアマッチ時の出力レベル	トグル出力
周期の終わり時の出力レベル	出力保持
デューティサイクル解除後の出力	設定不要
TAB：GTCCRA インプットキャプチャ要因	本項目は設定不要
TAB：GTCCRB	下記を設定する
GTCCRB 機能	コンペアマッチ：9999
バッファ動作	バッファ動作しない
GTIOC0B 端子機能	PWM 出力端子
ノイズフィルタ	設定不要
GTIOC0B 端子出力デューティ	コンペアマッチによって決定
GTIOC0B 端子ネゲート制御	禁止
開始／停止時の出力レベル	開始時 0 出力、停止時 0 出力
コンペアマッチ時の出力レベル	トグル出力
周期の終わり時の出力レベル	出力保持
デューティサイクル解除後の出力	設定不要
TAB：GTCCRB インプットキャプチャ要因	本項目は設定不要
グループ：GTCCRC、GTCCRD、GTCCRE、GTCCRF 設定	下記を設定する。それ以外は設定不要
GTCCRC 機能	コンペアマッチ：7999
グループ：カウントソース設定	本項目は設定不要
グループ：出力停止設定	本項目は設定不要
グループ：A/D 変換開始要求設定	本項目は設定不要
グループ：割り込み設定	下記を設定する。それ以外は設定不要
GTCCRC コンペアマッチ割り込み許可	優先順位：レベル 11
グループ：割り込み、A/D 変換開始要求間引き設定	本項目は設定不要
グループ：拡張割り込み間引き機能	本項目は設定不要
グループ：拡張バッファ転送間引き設定	本項目は設定不要

5.3.1.3　TPU を使用した場合の SC 設定例

表 5-12　励磁信号出力　コンポーネント選択（TPU）

コンポーネント選択	選択内容
コンポーネント選択	ノーマルモードタイマ
コンフィグレーション名	Config_TPU0_Esig12
インプットキャプチャ／アウトプットコンペア端子	4 端子
リソース	TPU0

表 5-13　励磁信号周波数 5kHz、出力端子を TIOCA0, B0 とした場合

項目	設定内容
グループ：同期動作設定	本項目は設定不要
グループ：TCNT0 カウンタ設定	下記を設定する。
カウンタクリア要因	TGRD0 コンペアマッチ／インプットキャプチャ
カウンタクロックの選択	PCLK
グループ：ジェネラルレジスタ	下記を設定する。それ以外は設定不要
TGRA0	アウトプットコンペアレジスタ　50μs
TGRB0	アウトプットコンペアレジスタ　83.333μs
TGRC0	アウトプットコンペアレジスタ　66.667μs
TGRD0	アウトプットコンペアレジスタ　100μs
グループ：入出力端子の設定	下記を設定する。それ以外は設定不要
TIOCA0 端子	端子初期出力は 0、コンペアマッチでトグル出力
TIOCB0 端子	端子初期出力は 0、コンペアマッチでトグル出力
グループ：ノイズフィルタ設定	本項目は設定不要
グループ：A/D 変換開始トリガ設定	本項目は設定不要
グループ：割り込み設定	下記を設定する。それ以外は設定不要（禁止）
TGRC インプットキャプチャ／コンペアマッチ割り込み許可	優先順位：レベル 11

5.3.2　レゾルバ信号の位相調整信号出力

位相調整信号出力機能のペリフェラルアサインは、MTU、GPT、TMR、TPU（TPU は RX72M のみ）を推奨します。以下に SC 設定例を示します。

5.3.2.1　MTU を使用した場合の SC 設定例

表 5-14　位相調整信号出力コンポーネント選択（MTU）

コンポーネント選択	選択内容
コンポーネント選択	PWM モードタイマ
コンフィグレーション名	Config_MTU0_PhaseA
動作	PWM モード 1
リソース	MTU0

表 5-15　位相調整信号 PWM 周波数 400kHz、出力端子を MTIOC0A とした場合

項目	設定内容
グループ：同期動作設定	本項目は設定不要
グループ：TCNT0 カウンタ設定	下記を設定する
カウンタクリア要因	TGRA0 コンペアマッチ／インプットキャプチャ
カウンタクロックの選択	PCLK
グループ：外部クロック端子設定	本項目は設定不要
グループ：ジェネラルレジスタ	本項目は設定不要
グループ：出力端子の設定	下記を設定する。それ以外は設定不要（端子出力は無効）
MTIOC0A 端子	端子初期出力は 1、コンペアマッチで 1 出力
TGRB コンペアマッチ一致時の動作	MTIOC0A から 0 出力
グループ：PWM 出力	下記を設定する。それ以外は設定不要
PWM 周期	2.5 μs
TGRA 初期値	299
TGRB 初期値	149
グループ：A/D 変換開始トリガ設定	本項目は設定不要
グループ：割り込み設定	本項目は設定不要

5.3.2.2　GPT を使用した場合の SC 設定例

表 5-16　位相調整信号出力コンポーネント選択（GPT）

コンポーネント選択	選択内容
コンポーネント選択	汎用 PWM タイマ
コンフィグレーション名	Config_GPT0_PhaseA
作業モード	のこぎり波 PWM モード
リソース	GPT0

表 5-17　位相調整信号 PWM 周波数 400kHz、出力端子を GTIOC0A とした場合

項目	設定内容
グループ：カウント設定	下記を設定する
クロックソース	PCLKA　120.000 MHz
タイマ動作周期	2.5 μs
周期レジスタ値	299
バッファ動作	バッファ動作しない
カウント方向	アップカウント
カウンタ初期値	0
インプットキャプチャはカウント停止時に動作	設定不要
グループ：コンペアマッチレジスタ、端子設定	—
TAB：GTCCRA	下記を設定する
GTCCRA 機能	コンペアマッチ：149
バッファ動作	バッファ動作しない
GTIOC0A 端子機能	PWM 出力端子
ノイズフィルタ	設定不要
GTIOC0A 端子出力デューティ	コンペアマッチによって決定
GTIOC0A 端子ネゲート制御	禁止
開始／停止時の出力レベル	開始時 1 出力、停止時 0 出力
コンペアマッチ時の出力レベル	0 出力
周期の終わり時の出力レベル	1 出力
デューティサイクル解除後の出力	設定不要
TAB：GTCCRA インプットキャプチャ要因	本項目は設定不要
TAB：GTCCRB	本項目は設定不要
TAB：GTCCRB インプットキャプチャ要因	本項目は設定不要
グループ：GTCCRC、GTCCRD、GTCCRE、GTCCRF 設定	本項目は設定不要
グループ：カウントソース設定	本項目は設定不要
グループ：出力停止設定	本項目は設定不要
グループ：A/D 変換開始要求設定	本項目は設定不要
グループ：割り込み設定	本項目は設定不要
グループ：割り込み、A/D 変換開始要求間引き設定	本項目は設定不要
グループ：拡張割り込み間引き機能	本項目は設定不要
グループ：拡張バッファ転送間引き設定	本項目は設定不要

5.3.2.3　TMR を使用した場合の SC 設定例

表 5-18　位相調整信号出力コンポーネント選択（TMR）

コンポーネント選択	選択内容
コンポーネント選択	8 ビットタイマ
コンフィグレーション名	Config_TMR0_PhaseA
カウントモード	8 ビット
リソース	TMR0

表 5-19　位相調整信号 PWM 周波数 400kHz、出力端子を TMO0 とした場合

項目	設定内容
グループ：カウント設定	下記を設定する
クロックソース	PCLK (60000.0kHz)
カウンタクリア	コンペアマッチ A によりクリア
コンペアマッチ A の値	2.5 µs
S12AD A/D 変換開始要求	設定不要
コンペアマッチ B の値	1.25 µs
グループ：TMO0 出力設定	下記を設定する
コンペアマッチ A 時の出力レベル	1 出力
コンペアマッチ B 時の出力レベル	0 出力
グループ：割り込み設定	本項目は設定不要

5.3.2.4　TPU を使用した場合の SC 設定例

表 5-20　位相調整信号出力　コンポーネント選択（TPU）

コンポーネント選択	選択内容
コンポーネント選択	PWM モードタイマ
コンフィグレーション名	Config_TPU0_PhaseA
動作	PWM モード 1
リソース	TPU0

表 5-21　位相調整信号 PWM 周波数 400kHz、出力端子を TIOCA0 とした場合

項目	設定内容
グループ：同期動作設定	本項目は設定不要
グループ：TCNT0 カウンタ設定	下記を設定する
カウンタクリア要因	TGRA0 コンペアマッチ
カウンタクロックの選択	PCLK
グループ：ジェネラルレジスタ	本項目は設定不要
グループ：入出力端子の設定	下記を設定する。それ以外は設定不要（端子出力は無効）
TIOCA0 端子	端子初期出力は 1、コンペアマッチで 1 出力
TGRB コンペアマッチ一致時の動作	TIOCA0 から 0 出力
グループ：PWM 出力設定	下記を設定する。それ以外は設定不要
PWM 周期	2.5 µs
TGRA 初期値	149
TGRB 初期値	78
グループ：A/D 変換開始トリガ設定	本項目は設定不要
グループ：割り込み設定	本項目は設定不要

5.3.3　角度誤差補正信号出力

角度誤差補正信号出力機能のペリフェラルアサインは、MTU、GPT、TPU（TPU は RX72M のみ）を推奨します。以下に SC 設定例を示します。

5.3.3.1　MTU を使用した場合の SC 設定例

表 5-22　角度誤差補正信号出力　コンポーネント選択（MTU）

コンポーネント選択	選択内容
コンポーネント選択	PWM モードタイマ
コンフィグレーション名	Config_MTU0_Csig
動作	PWM モード 1
リソース	MTU0

表 5-23　角度誤差補正信号周波数 200kHz、出力端子を MTIOC0A とした場合

項目	設定内容
グループ：同期動作設定	本項目は設定不要
グループ：TCNT0 カウンタ設定	下記を設定する
カウンタクリア要因	TGRA0 コンペアマッチ
カウンタクロックの選択	PCLK
グループ：外部クロック端子設定	本項目は設定不要
グループ：ジェネラルレジスタ	本項目は設定不要
グループ：出力端子の設定	下記を設定する。それ以外は設定不要（端子出力は無効）
MTIOC0A 端子	端子初期出力は 1、コンペアマッチで 1 出力
TGRB コンペアマッチ一致時の動作	MTIOC0A から 0 出力
グループ：PWM 出力	下記を設定する。それ以外は設定不要
PWM 周期	5 μs
TGRA 初期値	599
TGRB 初期値	299
グループ：A/D 変換開始トリガ設定	本項目は設定不要
グループ：割り込み設定	本項目は設定不要

5.3.3.2　GPT を使用した場合の SC 設定例

表 5-24　角度誤差補正信号出力　コンポーネント選択（GPT）

コンポーネント選択	選択内容
コンポーネント選択	汎用 PWM タイマ
コンフィグレーション名	Config_GPT0_Csig
作業モード	のこぎり波 PWM モード
リソース	GPT0

表 5-25　角度誤差補正信号周波数 200kHz、出力端子を GTIOC0A とした場合

項目	設定内容
グループ：カウント設定	下記を設定する
クロックソース	PCLKA (120.000 MHz)
タイマ動作周期	5 µs
周期レジスタ値	599
バッファ動作	バッファ動作しない
カウント方向	アップカウント
カウンタ初期値	0
インプットキャプチャはカウント停止時に動作	設定不要
グループ：コンペアマッチレジスタ、端子設定	—
TAB：GTCCRA	下記を設定する
GTCCRA 機能	コンペアマッチ：299
バッファ動作	バッファ動作しない
GTIOC0A 端子機能	PWM 出力端子
ノイズフィルタ	設定不要
GTIOC0A 端子出力デューティ	コンペアマッチによって決定
GTIOC0A 端子ネゲート制御	禁止
開始/停止時の出力レベル	開始時 1 出力、停止時 0 出力
コンペアマッチ時の出力レベル	0 出力
周期の終わり時の出力レベル	1 出力
デューティサイクル解除後の出力	設定不要
TAB：GTCCRA インプットキャプチャ要因	本項目は設定不要
TAB：GTCCRB	本項目は設定不要
TAB：GTCCRB インプットキャプチャ要因	本項目は設定不要
グループ：GTCCRC、GTCCRD、GTCCRE、GTCCRF 設定	本項目は設定不要
グループ：カウントソース設定	本項目は設定不要
グループ：出力停止設定	本項目は設定不要
グループ：A/D 変換開始要求設定	本項目は設定不要
グループ：割り込み設定	本項目は設定不要
グループ：割り込み、A/D 変換開始要求間引き設定	本項目は設定不要
グループ：拡張割り込み間引き機能	本項目は設定不要
グループ：拡張バッファ転送間引き設定	本項目は設定不要

5.3.3.3　TPU を使用した場合の SC 設定例

表 5-26　角度誤差補正信号出力コンポーネント選択（TPU）

コンポーネント選択	選択内容
コンポーネント選択	PWM モードタイマ
コンフィグレーション名	Config_TPU0_Csig
動作	PWM モード 1
リソース	TPU0

表 5-27　角度誤差補正信号周波数 200kHz、出力端子を TIOCA0 とした場合

項目	設定内容
グループ：同期動作設定	本項目は設定不要
グループ：TCNT0 カウンタ設定	下記を設定する
カウンタクリア要因	TGRA0 コンペアマッチ
カウンタクロックの選択	PCLK
グループ：ジェネラルレジスタ	本項目は設定不要
グループ：入出力端子の設定	下記を設定する。それ以外は設定不要 （端子出力は無効）
TIOCA0 端子	端子初期出力は 1、コンペアマッチで 1 出力
TGRB コンペアマッチ一致時の動作	TIOCA0 から 0 出力
グループ：PWM 出力設定	下記を設定する。それ以外は設定不要
PWM 周期	5 μs
TGRA 初期値	299
TGRB 初期値	149
グループ：A/D 変換開始トリガ設定	本項目は設定不要
グループ：割り込み設定	本項目は設定不要

5.3.4　角度誤差補正信号 Duty 更新割り込み

角度誤差補正信号 Duty 更新割り込みのペリフェラルアサインは、MTU、GPT、CMT、TPU（TPU は RX72M のみ）を推奨します。以下に SC 設定例を示します。

5.3.4.1　MTU を使用した場合の SC 設定例

表 5-28　角度誤差補正信号 Duty 更新割り込みコンポーネント選択（MTU）

コンポーネント選択	選択内容
コンポーネント選択	ノーマルモードタイマ
コンフィグレーション名	Config_MTU0_CsigUpdTim
インプットキャプチャ／アウトプットコンペア端子	2 端子、4 端子双方可能
リソース	MTU0

表 5-29　励磁信号周波数 5kHz、更新回数 2 回とした場合

項目	設定内容
グループ：同期動作設定	本項目は設定不要
グループ：TCNT0 カウンタ設定	下記を設定する
カウンタクリア要因	TGRA0 コンペアマッチ／インプットキャプチャ
カウンタクロックの選択	PCLK
グループ：外部クロック端子設定	本項目は設定不要
グループ：ジェネラルレジスタ	下記を設定する。それ以外は設定不要
TGRA0	アウトプットコンペアレジスタ　　100μs
グループ：入出力端子の設定	本項目は設定不要
グループ：ノイズフィルタ設定	本項目は設定不要
グループ：A/D 変換開始トリガ設定	本項目は設定不要
グループ：割り込み設定	下記を設定する。それ以外は設定不要（禁止）
TGRA	許可 優先順位：レベル 14

5.3.4.2　GPT を使用した場合の SC 設定例

表 5-30　角度誤差補正信号 Duty 更新割り込み　コンポーネント選択（GPT）

コンポーネント選択	選択内容
コンポーネント選択	汎用 PWM タイマ
コンフィグレーション名	Config_GPT0_CsigUpdTim
作業モード	のこぎり波 PWM モード
リソース	GPT0

表 5-31　励磁信号周波数 5kHz、更新回数 2 回とした場合

項目	設定内容
グループ：カウント設定	下記を設定する
クロックソース	PCLKA (120.000 MHz)
タイマ動作周期	100 µs
周期レジスタ値	11999
バッファ動作	バッファ動作しない
カウント方向	アップカウント
カウンタ初期値	0
インプットキャプチャはカウント停止時に動作	設定不要
グループ：コンペアマッチレジスタ、端子設定	—
TAB：GTCCRA	本項目は設定不要
TAB：GTCCRA インプットキャプチャ要因	本項目は設定不要
TAB：GTCCRB	本項目は設定不要
TAB：GTCCRB インプットキャプチャ要因	本項目は設定不要
グループ：GTCCRC、GTCCRD、GTCCRE、GTCCRF 設定	本項目は設定不要
グループ：カウントソース設定	本項目は設定不要
グループ：出力停止設定	本項目は設定不要
グループ：A/D 変換開始要求設定	本項目は設定不要
グループ：割り込み設定	下記を設定する。それ以外は設定不要
GTCNT オーバーフロー（GTPR コンペアマッチ）割り込みを許可	優先順位：レベル 14
グループ：割り込み、A/D 変換開始要求間引き設定	本項目は設定不要
グループ：拡張割り込み間引き機能	本項目は設定不要
グループ：拡張バッファ転送間引き設定	本項目は設定不要

5.3.4.3　TPU を使用した場合の SC 設定例

表 5-32　角度誤差補正信号 Duty 更新割り込みコンポーネント選択（TPU）

コンポーネント選択	選択内容
コンポーネント選択	PWM モードタイマ
コンフィグレーション名	Config_TPU0_CsigUpdTim
動作	PWM モード 1
リソース	TPU0

表 5-33　励磁信号周波数 5kHz、更新回数 2 回とした場合

項目	設定内容
グループ：同期動作設定	本項目は設定不要
グループ：TCNT0 カウンタ設定	下記を設定する
カウンタクリア要因	TGRA0 コンペアマッチ／インプットキャプチャ
カウンタクロックの選択	PCLK
グループ：ジェネラルレジスタ	本項目は設定不要
TGRA0	アウトプットコンペアレジスタ　　100μs
グループ：入出力端子の設定	本項目は設定不要
グループ：PWM 出力設定	本項目は設定不要
グループ：A/D 変換開始トリガ設定	本項目は設定不要
グループ：割り込み設定	下記を設定する。それ以外は設定不要
TGRA インプットキャプチャ／コンペアマッチ割り込み許可	優先順位：レベル 14

5.3.4.4　CMT を使用した場合の SC 設定例

表 5-34　角度誤差補正信号 Duty 更新割り込みコンポーネント選択（CMT）

コンポーネント選択	選択内容
コンポーネント選択	コンペアマッチタイマ
コンフィグレーション名	Config_CMT0_CsigUpdTim
リソース	CMT0

表 5-35　励磁信号周波数 5kHz、更新回数 2 回とした場合

項目	設定内容
グループ：クロック設定	下記を設定する
PCLK/8、PCLK/32、PCLK/128、PCLK/512	PCLK/8
グループ：入出力端子の設定	下記を設定する。
インターバル時間	100 μs
レジスタ	749
コンペアマッチ割り込み許可	許可
優先順位	レベル 14

5.3.5　角度信号入力

角度信号入力機能のペリフェラルアサインは、MTU、GPT、TPU（TPU は RX72M のみ）を推奨します。以下に SC 設定例を示します。

5.3.5.1　MTU を使用した場合の SC 設定例

表 5-36　角度信号入力コンポーネント選択（MTU）

コンポーネント選択	選択内容
コンポーネント選択	ノーマルモードタイマ
コンフィグレーション名	Config_MTU0_Cap
インプットキャプチャ／アウトプットコンペア端子	2 端子、4 端子双方可能（MTU1 / MTU2 は 2 端子のみ）
リソース	MTU0

表 5-37　励磁信号周波数 5kHz、入力端子を MTIOC0B とした場合

項目	設定内容
グループ：同期動作設定	本項目は設定不要
グループ：TCNT0 カウンタ設定	下記を設定する
カウンタクリア要因	TGRA0 コンペアマッチ／インプットキャプチャ
カウンタクロックの選択	PCLK
グループ：外部クロック端子設定	本項目は設定不要
グループ：ジェネラルレジスタ	下記を設定する。それ以外は設定不要
TGRA0	アウトプットコンペアレジスタ　　200μs
TGRB0	インプットキャプチャレジスタ
グループ：入出力端子の設定	下記を設定する。それ以外は設定不要（端子出力は無効）
MTIOC0B 端子	MTIOC0B 端子入力の立ち下がりエッジでインプットキャプチャ
グループ：ノイズフィルタ設定	本項目は設定不要
グループ：A/D 変換開始トリガ設定	本項目は設定不要
グループ：割り込み設定	下記を設定する。それ以外は設定不要（禁止）
TGRB インプットキャプチャ／コンペアマッチ割り込み許可	優先順位：レベル 13

5.3.5.2　GPT を使用した場合の SC 設定例

表 5-38　角度信号入力　コンポーネント選択（GPT）

コンポーネント選択	選択内容
コンポーネント選択	汎用 PWM タイマ
コンフィグレーション名	Config_GPT0_Cap
作業モード	のこぎり波 PWM モード
リソース	GPT0

表 5-39　励磁信号周波数 5kHz、入力端子を GTIOC0A とした場合

項目	設定内容
グループ：カウント設定	下記を設定する
クロックソース	PCLKA (120.000 MHz)
タイマ動作周期	200us
周期レジスタ値	23999
バッファ動作	バッファ動作しない
カウント方向	アップカウント
カウンタ初期値	0
インプットキャプチャはカウント停止時に動作	設定不要
グループ：コンペアマッチレジスタ、端子設定	—
TAB：GTCCRA	下記を設定する。それ以外は設定不要
GTCCRA 機能	インプットキャプチャ
バッファ動作	バッファ動作しない
GTIOC0A 端子機能	入力端子
TAB：GTCCRA インプットキャプチャ要因	下記を設定する。それ以外は設定不要
GTIOC0A 端子立ち下がりエッジ選択	GTIOC0A 入力の立ち下がり
TAB：GTCCRB	本項目は設定不要
TAB：GTCCRB インプットキャプチャ要因	本項目は設定不要
グループ：GTCCRC、GTCCRD、GTCCRE、GTCCRF 設定	本項目は設定不要
グループ：カウントソース設定	本項目は設定不要
グループ：出力停止設定	本項目は設定不要
グループ：A/D 変換開始要求設定	本項目は設定不要
グループ：割り込み設定	下記を設定する。それ以外は設定不要
GTCCRA コンペアマッチ／インプットキャプチャ割り込み許可	優先順位：レベル 13
グループ：割り込み、A/D 変換開始要求間引き設定	本項目は設定不要
グループ：拡張割り込み間引き機能	本項目は設定不要
グループ：拡張バッファ転送間引き設定	本項目は設定不要

5.3.5.3　TPU を使用した場合の SC 設定例

表 5-40　角度信号入力コンポーネント選択（TPU）

コンポーネント選択	選択内容
コンポーネント選択	ノーマルモードタイマ
コンフィグレーション名	Config_TPU0_Cap
インプットキャプチャ／アウトプットコンペア端子	2 端子、4 端子双方可能
リソース	TPU0

表 5-41　励磁信号周波数 5kHz、入力端子を TIOCB0 とした場合

項目	設定内容
グループ：同期動作設定	本項目は設定不要
グループ：TCNT0 カウンタ設定	下記を設定する
カウンタクリア要因	TGRA0 コンペアマッチ／インプットキャプチャ
カウンタクロックの選択	PCLK
グループ：ジェネラルレジスタ	下記を設定する。それ以外は設定不要
TGRA0	アウトプットコンペアレジスタ　　200µs
TGRB0	インプットキャプチャレジスタ
グループ：入出力端子の設定	本項目は設定不要
TIOCB0 端子	MTIOC0B 端子入力の立ち下がりエッジでインプットキャプチャ
グループ：ノイズフィルタ設定	本項目は設定不要
グループ：A/D 変換開始トリガ設定	本項目は設定不要
グループ：割り込み設定	下記を設定する。それ以外は設定不要（禁止）
TGRB インプットキャプチャ／コンペアマッチ割り込み許可	優先順位：レベル 13

5.3.6　RDC-IC 動作クロック出力

RDC IC 動作クロック出力機能のペリフェラルアサインは、MTU、GPT、TMR、TPU（TPU は RX72M のみ）を推奨します。以下に SC 設定例を示します。

5.3.6.1　MTU を使用した場合の SC 設定例

表 5-42　RDC-IC 動作クロック出力　コンポーネント選択（MTU）

コンポーネント選択	選択内容
コンポーネント選択	PWM モードタイマ
コンフィグレーション名	Config_MTU0_RdcClk
動作	PWM モード 1
リソース	MTU0

表 5-43　RDC Clock 4MHz、出力端子を MTIOC0A とした場合

項目	設定内容
グループ：同期動作設定	本項目は設定不要
グループ：TCNT0 カウンタ設定	下記を設定する
カウンタクリア要因	TGRA0 コンペアマッチ（TGRA0 を周期レジスタとして使用）
カウンタクロックの選択	PCLK
グループ：外部クロック端子設定	本項目は設定不要
グループ：ジェネラルレジスタ	本項目は設定不要
グループ：出力端子の設定	下記を設定する。それ以外は設定不要
MTIOC0A 端子	端子初期出力は 1、コンペアマッチで 1 出力
TGRB コンペアマッチ一致時の動作	MTIOC0A 端子から 0 出力
グループ：PWM 出力設定	下記を設定する。それ以外は設定不要
PWM 周期	250 ns
TGRA 初期値	29
TGRB 初期値	14
グループ：A/D 変換開始トリガ設定	本項目は設定不要
グループ：割り込み設定	本項目は設定不要

5.3.6.2　GPT を使用した場合の SC 設定例

表 5-44　RDC-IC 動作クロック出力コンポーネント選択（GPT）

コンポーネント選択	選択内容
コンポーネント選択	汎用 PWM タイマ
コンフィグレーション名	Config_GPT0_RdcClk
作業モード	のこぎり波 PWM モード
リソース	GPT0

表 5-45　RDC Clock 4MHz、出力端子を GTIOC0A とした場合

項目	設定内容
グループ：カウント設定	下記を設定する
クロックソース	PCLKA　120.000 MHz
タイマ動作周期	250 ns
周期レジスタ値	29
バッファ動作	バッファ動作しない
カウント方向	アップカウント
カウンタ初期値	0
インプットキャプチャはカウント停止時に動作	設定不要
グループ：コンペアマッチレジスタ、端子設定	—
TAB：GTCCRA	下記を設定する
GTCCRA 機能	コンペアマッチ：14
バッファ動作	バッファ動作しない
GTIOC0A 端子機能	PWM 出力端子
ノイズフィルタ	設定不要
GTIOC0A 端子出力デューティ	コンペアマッチによって決定
GTIOC0A 端子ネゲート制御	禁止
開始/停止時の出力レベル	開始時 1 出力、停止時 0 出力
コンペアマッチ時の出力レベル	0 出力
周期の終わり時の出力レベル	1 出力
デューティサイクル解除後の出力	設定不要
TAB：GTCCRA インプットキャプチャ要因	本項目は設定不要
TAB：GTCCRB	下記を設定する。それ以外は設定不要
GTCCRB 機能	コンペアマッチ：28 *
TAB：GTCCRB インプットキャプチャ要因	本項目は設定不要
グループ：GTCCRC、GTCCRD、GTCCRE、GTCCRF 設定	下記を設定する。
GTCCRC 機能	コンペアマッチ：28 *
GTCCRD 機能	コンペアマッチ：28 *
GTCCRE 機能	コンペアマッチ：28 *
GTCCRF 機能	コンペアマッチ：28 *
グループ：カウントソース設定	本項目は設定不要
グループ：出力停止設定	本項目は設定不要
グループ：A/D 変換開始要求設定	本項目は設定不要
グループ：割り込み設定	本項目は設定不要
グループ：割り込み、A/D 変換開始要求間引き設定	本項目は設定不要
グループ：拡張割り込み間引き機能	本項目は設定不要
グループ：拡張バッファ転送間引き設定	本項目は設定不要

【注】 *　初期値だと設定範囲外のエラーとなるため、最大値を設定する

5.3.6.3　TMR を使用した場合の SC 設定例

表 5-46　RDC-IC 動作クロック出力　コンポーネント選択（TMR）

コンポーネント選択	選択内容
コンポーネント選択	8 ビットタイマ
コンフィグレーション名	Config_TMR0_RdcClk
カウントモード	8 ビット
リソース	TMR0

表 5-47　RDC Clock 4MHz、出力端子を TMO0 とした場合

項目	設定内容
グループ：カウント設定	下記を設定する
クロックソース	PCLK (60000.0kHz)
カウンタクリア	コンペアマッチ A によりクリア
コンペアマッチ A の値	250 ns
S12AD A/D 変換開始要求	設定しない
コンペアマッチ B の値	125 ns
グループ：TMO0 出力設定	下記を設定する
TMO0 出力許可	許可
コンペアマッチ A 時の出力レベル	1 出力
コンペアマッチ B 時の出力レベル	0 出力
グループ：割り込み設定	本項目は設定不要

5.3.6.4　TPU を使用した場合の SC 設定例

表 5-48　RDC-IC 動作クロック出力　コンポーネント選択（TPU）

コンポーネント選択	選択内容
コンポーネント選択	PWM モードタイマ
コンフィグレーション名	Config_TPU0_RdcClk
動作	PWM モード 1
リソース	TPU0

表 5-49　RDC Clock 4MHz、出力端子を TIOCA0 とした場合

項目	設定内容
グループ：同期動作設定	本項目は設定不要
グループ：TCNT0 カウンタ設定	下記を設定する
カウンタクリア要因	TGRA0 コンペアマッチ（TGRA0 を周期レジスタとして使用）
カウンタクロックの選択	PCLK
グループ：ジェネラルレジスタ	設定不要
グループ：出力端子の設定	下記を設定する。それ以外は設定不要
TIOCA0 端子	端子初期出力は 1、コンペアマッチで 1 出力
TGRB コンペアマッチ一致時の動作	TIOCA0 端子から 0 出力
グループ：PWM 出力設定	下記を設定する。それ以外は設定不要
PWM 周期	250 ns
TGRA 初期値	14
TGRB 初期値	7
グループ：A/D 変換開始トリガ設定	本項目は設定不要
グループ：割り込み設定	本項目は設定不要

5.3.7　RDC-IC 通信機能

RDC IC 通信機能のペリフェラルアサインは、RSPI、SCI となります。以下に SC 設定例を示します。

5.3.7.1　RSPI（SSLA0 を選択時）を使用した場合の SC 設定例

表 5-50　RDC IC 通信機能コンポーネント選択（RSPI）

コンポーネント選択	選択内容
コンポーネント選択	SPI 動作モード（4 線式）
コンフィグレーション名	Config_RSPI0_RdcCom
動作	マスタ送信/受信機能
リソース	RSPI0

表 5-51　RDC IC 通信機能 RSPI0 にアサインした場合 (1/2)

項目	設定内容
グループ：送信／受信データバッファ設定	下記を設定する
バッファアクセス幅	16 ビット
グループ：パリティ設定	下記を設定する
バイトスワップ	無効
パリティビット	送信データパリティビットを付加しない 受信データのパリティチェックを行わない
グループ：転送速度設定	下記を設定する
ベースビットレート	1000 kbps
グループ：出力タイミング設定	下記を設定する
SSL 信号アサート開始から RSPCK 発振までの期間	1 RSPCK
最終 RSPCK エッジ送出から SSL 信号ネゲートまでの期間	1 RSPCK
転送終了後の SSL 信号の非アクティブ期間	1 RSPCK + 2 PCLK
グループ：自動停止機能設定	下記を設定する
自動停止機能有効	無効（設定しない）
グループ：端子制御設定	下記を設定する
MISO アイドル値	Low
SSLA0 端子	アクティブ Low
SSLA1 端子	無効（チェックを外す）
SSLA2 端子	無効（チェックを外す）
SSLA3 端子	無効（チェックを外す）
RSPI 端子制御設定	CMOS 出力
ループバックモード	通常モード
グループ：データ処理設定	下記を設定する
転送データ処理	割り込みサービスルーチンで処理する
グループ：割り込み設定	下記を設定する
SPTI0 優先順位	レベル 9
SPRI0 優先順位	レベル 9
エラー割り込み許可	許可
SPEI0、SPII0 優先順位	レベル 9

表 5-50　RDC IC 通信機能 RSPI0 にアサインした場合 (2/2)

項目	設定内容
グループ：コマンド設定	下記を設定する
TAB：コマンド 0	
コマンド数、フレーム数	コマンド数：1、転送フレーム数：1
ビット長	16 ビット
フォーマット	MSB ファースト
RSPCK 位相	奇数エッジでデータ変化、偶数エッジでデータサンプル
RSPCK 極性	アイドル時の RSPCK が High
ビットレートの選択	ベースのビットレート
SSL アサート信号	SSL0（ボード依存）
SSL ネゲート動作	転送終了時に全 SSL 信号をネゲート
RSPCK 遅延	1 RSPCK
SSL ネゲート遅延	1 RSPCK
次アクセス遅延	1 RSPCK + 2 PCLK

5.3.7.2　SCI を使用した場合の SC 設定例

表 5-52　RDC IC 通信機能コンポーネント選択（SCI）

コンポーネント選択	選択内容
コンポーネント選択	SPI クロック同期モード（3 線式）
コンフィグレーション名	Config_SCI0_RdcCom
動作	マスタ送信/受信機能
リソース	SCI0

表 5-53　RDC IC 通信機能 SCI0 にアサインした場合

項目	設定内容
グループ：データ転送方向設定	下記を設定する
LSB ファースト／MSB ファースト	MSB ファースト
グループ：送受信データ・レベル設定	下記を設定する
標準／反転	標準
グループ：転送速度設定	下記を設定する
転送クロック	内部クロック
ベースビットレート	1000 kbps
グループ：クロック設定	下記を設定する
クロック遅れあり／クロック極性判定あり	双方なし（チェックしない）
グループ：データ処理設定	下記を設定する
送信データ処理	割り込みサービスルーチンで処理する
受信データ処理	割り込みサービスルーチンで処理する
グループ：割り込み設定	下記を設定する
TXI0 優先順位	レベル 9
RXI0 優先順位	レベル 9
受信エラー割り込み許可	許可
TEII0、ERII0 優先順位	レベル 9
グループ：コールバック機能設定	下記を設定する
送信完了　受信完了　エラー検出	全て有効にする（チェックを入れる）

SCI にて RDC-IC 通信を実施する場合、RDC-IC チップセレクトの処理を組み込む必要があります。組み込み箇所は下記になります。組み込み例は、「5.4.6　SPI 送受信関数」を参照してください。

チップセレクト ON（ACTIVE）　：送受信開始処理に組み込む
チップセレクト OFF（INACTIVE）：受信完了処理に組み込む

5.4　関数テーブル設定

各ドライバ機能に割り当てたペリフェラルのレジスタにアクセスするために、SC で生成された関数とユーザが作成した関数を関数テーブルに設定する必要があります。各ドライバ機能に設定する関数テーブル一覧表を以下に示します。

表 5-54　各ドライバ機能に設定する関数テーブル一覧 (1/2)

ドライバ機能 / 関数テーブル	ESIG1	ESIG2_1	ESIG2_2	ESIG12	CSIG	PHASE_A
タイマ開始関数	◎	◎	◎	◎	◎	◎
タイマ停止関数	◎	◎	◎	◎	◎	◎
カウント値取得関数	○	○	○	○	○	×
カウント値設定関数	○	○	○	○	○	×
Duty 値取得関数	×	×	×	×	○	×
Duty 値設定関数	×	×	×	×	○	○
Duty 値設定関数 (1 タイマで A/B 相出力)	×	×	×	×	×	×
キャプチャ値取得関数	×	×	×	×	×	×
ポートレベル取得関数	×	×	×	×	×	×
RDC IC SPI 送受信関数	×	×	×	×	×	×

◎：設定必須（SC 生成コード）　○：設定必須（ユーザ作成コード）　×：設定不要

表 5-55　各ドライバ機能に設定する関数テーブル一覧 (2/2)

ドライバ機能 / 関数テーブル	PHASE_B	PHASE_AB	CAPTURE	CSIG_UPD_TIMER	RDC_CLK	RDC_COM
タイマ開始関数	◎	◎	◎	◎	◎	×
タイマ停止関数	◎	◎	◎	◎	◎	×
カウント値取得関数	×	×	○	×	×	×
カウント値設定関数	×	×	○	×	×	×
Duty 値取得関数	×	×	×	○	×	×
Duty 値設定関数	○	×	×	○	×	×
Duty 値設定関数（1 タイマで A/B 相出力）	×	○	×	×	×	×
キャプチャ値取得関数	×	×	○	×	×	×
ポートレベル取得関数	×	×	○	×	×	×
RDC IC SPI 送受信関数	×	×	×	×	×	◎

◎：設定必須（SC 生成コード）　○：設定必須（ユーザ作成コード）　×：設定不要

次項に関数テーブルに設定する処理の詳細を示します。

5.4.1　タイマ開始・停止関数

SC が生成するモジュールの Start、Stop 関数を関数テーブルに設定します。

5.4.2　カウント値取得・設定関数

カウント値取得・設定関数は SC では生成されないため、ユーザが作成して関数テーブルに設定します。例として、MTU0 を使用した場合のカウント値取得・設定関数を以下に示します。（機能例は角度誤差補正信号（CSig））

```
/* Function to get the counter value */
void R_Config_MTU0_Csig_GetTcnt (unsigned short *tcnt)
{
    *tcnt = MTU0.TCNT;
}

/* Function to set the counter value */
void R_Config_MTU0_Csig_SetTcnt (unsigned short tcnt)
{
    MTU0.TCNT = tcnt;
}
```

5.4.3　Duty 値取得・設定関数

Duty 値取得・設定関数は SC では生成されないため、ユーザが作成して関数テーブルに設定します。

例として、MTU0 の TGRA が出力信号の Duty を変更できるジェネラルレジスタとした場合の Duty 値取得・設定関数を以下に示します。（機能例は角度誤差補正信号（CSig））

```
/* Function to get the duty value */
void R_Config_MTU0_Csig_GetDuty (unsigned short *duty)
{
    *duty = MTU0.TGRA;
}

/* Function to set the duty value */
void R_Config_MTU0_Csig_SetDuty (unsigned short duty)
{
    MTU0.TGRA = duty;
}

/* Function to set the duty value */
void R_Config_MTU0_Csig_SetDuty_2val (unsigned short ch, unsigned short duty)
{
    If (PHASE_CH_A == ch)
    {
        MTU0.TGRA = duty;       /* Phase A signal duty setting */
    }
    else if (PHASE_CH_B == ch)
    {
        MTU0.TGRC = duty;       /* Phase B signal duty setting */
    }
}
```

5.4.4　キャプチャ値取得関数

キャプチャ値取得関数は SC では生成されないため、ユーザが作成して関数テーブルに設定します。

例として、MTU2 を使用した場合のキャプチャ値取得関数を以下に示します。

```
/* Function to get the capture value */
void R_Config_MTU2_Cap_GetCapVal (unsigned short *current_angle_count)
{
    *current_angle_count= MTU2.TGRA;
}
```

5.4.5　ポートレベル取得関数

ポートレベル取得関数は SC では生成されないため、ユーザが作成して関数テーブルに設定します。

例として、P00 のポートレベルを取得する場合の関数を以下に示します。

```
/* Function to get the port level */
void R_Config_MTU2_Cap_GetPortLvl (unsigned char *port_level)
{
    *port_level = PORT0.PIDR.BIT.B0;
}
```

5.4.6　SPI 送受信関数

SC で生成した RSPI または SCI の送受信関数を関数テーブルに設定します。SCI で作成した場合、汎用ポートを使用してチップセレクト信号を出力する必要があります。また、SCI は通信フォーマットが 8 ビットのため、RDC-IC と通信するためには 16 ビットフォーマットに対応する必要があります。コード例は「7.10.2.3　SCI を使用した例」を参照してください。下記に、SCI、RSPI の送受信関数例を示します。

5.4.6.1　SCI を使用した場合

```
/* 送受信開始処理（SC 生成コード）*/
MD_STATUS R_Config_SCI1_RdcCom_SPI_Master_Send_Receive
(uint8_t * const tx_buf, uint16_t tx_num, uint8_t * const rx_buf, uint16_t
rx_num)
{
    MD_STATUS status = MD_OK;

    if (1U > tx_num)
    {
        status = MD_ARGERROR;
    }
    else
    {

        R_Config_SCI0_Start();        // SCI モジュールスタート（追加が必要です）

        g_sci0_tx_count = tx_num;
        gp_sci0_tx_address = tx_buf;
        gp_sci0_rx_address = rx_buf;
        g_sci0_rx_count = 0U;
        g_sci0_rx_length = rx_num;

        /* Set SMOSI0 pin */
        PORT2.PMR.BYTE |= 0x01U;

        /* Set low to CS port */
        PORT9.PODR.BIT.B2 = 0U;     // チップセレクト：Chip ACTIVE（追加が必要です）

        /* Set TE, TIE, RE, RIE bits simultaneously */
        SCI0.SCR.BYTE |= 0xF0U;
    }

    return (status);
}
```

5.4.6.2 RSPI を使用した場合

```
/* 送受信開始処理（SC 生成コード）*/
MD_STATUS R_Config_RSPI0_RdcCom_Send_Receive
(uint16_t * const tx_buf, uint16_t tx_num, uint16_t * const rx_buf)
{
    MD_STATUS status = MD_OK;

    if (tx_num < 1U)
    {
        status = MD_ARGERROR;
    }
    else
    {
        R_Config_RSPI0_RSPI0_Start();      // RSPI モジュールスタート（追加が必要です）

        /* Initialize the global counters */
        gp_rspi0_tx_address = tx_buf;
        g_rspi0_tx_count = tx_num;
        gp_rspi0_rx_address = rx_buf;
        g_rspi0_rx_length = tx_num;
        g_rspi0_rx_count = 0U;

        /* Enable transmit interrupt */
        RSPI0.SPCR.BIT.SPTIE = 1U;

        /* Enable receive interrupt */
        RSPI0.SPCR.BIT.SPRIE = 1U;

        /* Enable error interrupt */
        RSPI0.SPCR.BIT.SPEIE = 1U;

        /* Enable RSPI function */
        RSPI0.SPCR.BIT.SPE = 1U;
    }

    return (status);
}
```

6. API

6.1 API 関数一覧

API 関数は Application/Middle から呼び出すことが可能なドライバの関数です。以下に一覧を示します。詳細は、「6.2 API 関数説明」を参照してください。

表 6-1 API 関数一覧（r_rslv_api.h）（1/4）

ファイル名	カテゴリ	I/F 関数名	処理概要
r_rslv_api.h	初期化関連 システム関連	R_RSLV_SetSystemInfo 入力：ST_SYSTEM_PARAM *rdc_sys_param / システム情報 ST_USER_PERI_PARAM *user_peri_param / ユーザペリフェラル設定情報 出力：unsigned char result / 処理結果	引数で渡された情報から、使用するタイマカウンタ値等の選択を行う。
		R_RSLV_SetFuncTable 入力：unsigned char set_func, / ドライバ機能 FUNCTION_TABLE user_func_table/ 関数ポインタ設定値 出力：unsigned char result / 処理結果	引数で渡された関数ポインタを関数テーブルに設定する。
		R_RSLV_GetRdcDrvSettingInfo 入力：ST_RDC_DRV_SETTING_INFO *rdc_setting_info / 設定情報用構造体ポインタ 出力：unsigned char result / 処理結果	RDC-IC ドライバで設定している励磁周波数と角度検出タイマカウンタ最大値を引数のポインタ変数にセットしてユーザに通知する。
		R_RSLV_MTU_SyncStart 入力：unsigned char start_ch / MTU チャネル 出力：unsigned char result / 処理結果	MTU シンクロスタートレジスタに引数で渡された値を書き込んで、MTU の該当チャネルのタイマを同時にスタートする。
		R_RSLV_GetDriverVer 入力：unsigned long *drv_ver / ドライババージョン格納バッファポインタ 出力：unsigned char result / 処理結果	RDC-IC ドライバのバージョンを取得する。
	角度誤差補正信号	R_RSLV_CSig_Start 入力：unsigned short phase_diff / 位相シフト量 unsigned short amp_level / 振幅レベル 出力：unsigned char result / 処理結果	角度誤差補正 Duty 演算等、角度誤差補正信号の出力開始準備を行う。
		R_RSLV_CSig_Stop 入力：無し 出力：unsigned char result / 処理結果	角度誤差補正信号出力を停止する。
		R_RSLV_INT_CSig_UpdatePwmDuty 入力：無し 出力：unsigned char result / 処理結果	角度誤差補正信号の PWM Duty を更新する。
		R_RSLV_INT_CSig_SyncStart 入力：無し 出力：unsigned char result / 処理結果	励磁信号と角度誤差補正信号の同期スタートを行う。
		R_RSLV_GetCSigStatus 入力：unsigned char *status / 角度誤差補正信号出力状態取得ポインタ 出力：unsigned char result / 処理結果	角度誤差補正信号出力状態を取得する。

表 6-1　API 関数一覧（r_rslv_api.h）（2/4）

ファイル名	カテゴリ	I/F 関数名	処理概要
r_rslv_api.h	角度信号入力	R_RSLV_Capture_Start 入力：無し 出力：unsigned char result / 処理結果	角度検出タイマをスタートする。
		R_RSLV_INT_GetCaptureCount 入力：無し 出力：unsigned char result / 処理結果	角度検出（現在の角度カウント）値を取り込み、前回値の差を計算して、変数にセットする。
		R_RSLV_GetCaptureEdge 入力：unsigned char *cap_edge / キャプチャポート状態 出力：unsigned char result / 処理結果	前回キャプチャが立ち上がりエッジか、立ち下がりエッジか判定するための情報。
		R_RSLV_GetAngleCountFirstEdge 入力：unsigned short *angle_cnt / 角度 出力：unsigned char result / 処理結果	変数に保存している現在の角度カウントを取出す。（立ち下がりエッジ）
		R_RSLV_GetAngleDifferenceFirstEdge 入力：unsigned short *angle_diff_cnt / 角度差 出力：unsigned char result / 処理結果	変数に保存している現在の角度と前回の角度差を取出す。（立ち下がりエッジ）
		R_RSLV_GetAngleCountSecondEdge 入力：unsigned short *angle_cnt / 角度 出力：unsigned char result / 処理結果	変数に保存している現在の角度カウントを取出す。（立ち上がりエッジ）
		R_RSLV_GetAngleDifferenceSecondEdge 入力：unsigned short *angle_diff_cnt / 角度差 出力：unsigned char result / 処理結果	変数に保存している現在の角度と前回の角度差を取出す。（立ち上がりエッジ）
	励磁信号	R_RSLV_ESig_Start 入力：無し 出力：unsigned char result / 処理結果	励磁信号の出力を開始する。
		R_RSLV_ESig_Stop 入力：無し 出力：unsigned char result / 処理結果	励磁信号の出力を停止する。
		R_RSLV_EsigCapStartTiming 入力：unsigned short esig_start_tcnt / ESIG タイマカウンタ値 unsigned short cap_start_tcnt / Capture タイマカウンタ値 出力：unsigned char result / 処理結果	励磁信号出力タイミングと角度検出タイマ開始タイミングを調整する。
		R_RSLV_INT_ESigCounter 入力：無し 出力：unsigned char result / 処理結果	自動調整処理内でのウェイトタイマのカウントダウンを行う。
	位相調整	R_RSLV_Phase_AdjStart 入力：無し 出力：unsigned char result / 処理結果	位相調整信号の出力を開始する。
		R_RSLV_Phase_AdjStop 入力：無し 出力：unsigned char result / 処理結果	位相調整信号の出力を停止する。
		R_RSLV_Phase_AdjUpdateBuff 入力：unsigned short duty / Duty 値 unsigned char ch / A 相/B 相の選択 出力：unsigned char result / 処理結果	位相調整信号 Duty をバッファにセットする。
		R_RSLV_Phase_AdjUpdate 入力：無し 出力：unsigned char result / 処理結果	位相調整信号 Duty をレジスタにセットする。

表 6-1　API 関数一覧（r_rslv_api.h）（3/4）

ファイル名	カテゴリ	I/F 関数名	処理概要
r_rslv_api.h	位相調整	R_RSLV_Phase_AdjReadBuff 入力：unsigned short *duty / 読み込んだ Duty 値 unsigned char ch / 読み込む相指定（A/B 相） 出力：unsigned char result / 処理結果	位相調整信号の Duty をレジスタから読み込む。
	RDC-IC 設定	R_RSLV_Rdc_VariableInit 入力：unsigned char *u1_init_data / RDC-IC 初期化コマンドテーブル 出力：unsigned char result / 処理結果	RDC-IC 通信の初期値を設定する。
		R_RSLV_Rdc_Init_Sequence 入力：unsigned short *init_status / 通信状態 出力：unsigned char result / 処理結果	RDC-IC 初期設定を行う。
		R_RSLV_Rdc_Communication 入力：無し 出力：unsigned char result / 処理結果	RDC-IC との通信を行う。通信シーケンスを持ち、繰り返し呼び出されることでシーケンスが進行する。
		R_RSLV_Rdc_RegWrite 入力：unsigned char *write_status / 書き込みステータス 出力：unsigned char result / 処理結果	RDC-IC レジスタバッファ変数に値を書き込む。
		R_RSLV_Rdc_RegRead 入力：unsigned char address / 読み込みアドレス 出力：unsigned char result / 処理結果	RDC-IC レジスタの読み込みを開始する 【注】本関数は読み込み開始のトリガである
		R_RSLV_Rdc_ChkIfRun 入力：無し 出力：unsigned char result / 処理結果	RDC-IC レジスタアクセス有無状態を戻り値として返す
		R_RSLV_Rdc_GetRegisterVal 入力：unsigned char *rd_data / 変数からの読み込みデータ unsigned char address / 読み込みアドレス 出力：unsigned char result / 処理結果	変数に格納されている RDC-IC レジスタの値を読み込む
		R_RSLV_Rdc_SetRegisterVal 入力：unsigned char wt_data / 変数への書き込みデータ unsigned char address / 書き込みアドレス 出力：unsigned char result / 処理結果	RDC-IC レジスタの値を変数に書き込む
		R_RSLV_Rdc_CallComEndCb 入力：無し 出力：unsigned char result / 処理結果	RDC-IC 通信送受信完了割り込みコールバック処理
		R_RSLV_Rdc_CallErrorCb 入力：無し 出力：unsigned char result / 処理結果	RDC-IC 通信エラー割り込みコールバック処理
		R_RSLV_RdcCom_GetErrorInfo 入力：unsigned char *err_info / RDC-IC 通信異常情報 出力：unsigned char result / 処理結果	RDC-IC 通信の異常発生有無を取得する
		R_RSLV_Rdc_AlarmCancelStart 入力：無し 出力：unsigned char result / 処理結果	RDC-IC Alarm キャンセルの開始
		R_RSLV_Rdc_AlarmCancel 入力：無し 出力：unsigned char result / 処理結果	RDC-IC Alarm キャンセルのシーケンス制御

表 6-1　API 関数一覧（r_rslv_api.h）（4/4）

ファイル名	カテゴリ	I/F 関数名	処理概要
r_rslv_api.h	自動調整	R_RSLV_ADJST_GainPhase 入力：unsigned char u1_call_state / 調整実行リクエスト 出力：st_adjst_gainphase_return_t / 処理結果	レゾルバ信号ゲイン調整とレゾルバ信号位相調整を実行します。
		R_RSLV_ADJST_Carrier 入力：st_adjst_carrier_arg_t arg_value / 調整実行リクエスト 出力：st_adjst_carrier_return_t return_val / 調整処理実行状況、処理結果	角度誤差補正信号調整を実行します。
		R_RSLV_ADJST_SetPtrFunc 入力：st_ptr_func_arg_t *ptr_arg / コールバック関数ポインタ 出力：無し	ユーザが作成したコールバック関数、及び変数のポインタを調整機能へセットします。
		R_RSLV_ADJST_Ad_Processing 入力：無し 出力：unsigned char gs_u1_ad_condition / AD 変換実行状態	モニタ信号 AD 変換中の場合 1 を返す。 それ以外では 0 を返す。
		R_RSLV_ADJST_SetFilterDelay 入力：float bpf_delay_deg / BPF 遅れ位相 float csig_delay_deg / 角度誤差補正信号向け LPF 遅れ位相 出力：unsigned char result / 処理結果	BPF による遅れ位相、及び角度誤差補正信号向け LPF による遅れ位相を設定します。
	断線検知	R_RSLV_DiscDetection_Seq 入力：st_rdc_ddmnt_arg_t arg_value / 断線検出用パラメータ 出力：unsigned char return_valt / 動作状態	断線検知シーケンス処理を行う

6.2　API 関数説明

6.2.1　関数テーブル設定 API 関数

項目	内容	
関数名	R_RSLV_SetFuncTable	
引数	unsigned char set_func, ST_FUNCTION_TABLE user_func_table	関数テーブルをセットするドライバ機能 関数テーブル
戻り値	unsigned char	処理結果
機能	ドライバ内部で使用する関数テーブルの設定を行います。 • ドライバ機能の指定 • 関数テーブルの指定	
備考	ST_FUNCTION_TABLE は構造体であり、関数テーブルの設定は「5.4　関数テーブル設定」を参照、ペリフェラルと機能の組み合わせなどは「5.2　各ドライバ機能に割り当てるペリフェラル一覧（推奨）」を参照してください。	

6.2.2　システム情報の指定 API 関数

項目	内容	
関数名	R_RSLV_SetSystemInfo	
引数	ST_SYSTEM_PARAM *rdc_sys_param, ST_USER_PERI_PARAM *user_peri_param	システム設定情報 使用ペリフェラルのカウントクロックソース
戻り値	unsigned char	処理結果
機能	システム情報の指定を行います。 • 励磁信号周波数の指定 • 角度誤差補正信号出力周波数の指定 • 角度誤差補正 Duty 更新回数の指定 • モータータイプの指定 • RDC-IC の MNTOUT 端子出力方式の指定 • 励磁信号出力割当ペリフェラルのカウントクロックソース指定（MHz 単位） • 角度誤差補正信号出力割当ペリフェラルのカウントクロックソース指定（MHz 単位） • 角度信号入力割当ペリフェラルのカウントクロックソース指定（MHz 単位） • 角度誤差補正信号 Duty 更新割当ペリフェラルのカウントクロックソース指定（MHz 単位） • 位相調整信号 A 出力割当ペリフェラルのカウントクロックソース指定（MHz 単位） • 位相調整信号 B 出力割当ペリフェラルのカウントクロックソース指定（MHz 単位）	
備考	ST_SYSTEM_PARAM は構造体であり、システム情報の指定の詳細は「6.3.2 R_RSLV_SetSystemInfo API 関数の構造体」を参照してください。	

6.2.3　RDC-IC ドライバ設定情報取り出し API 関数

項目	内容	
関数名	R_RSLV_GetRdcDrvSettingInfo	
引数	ST_RDC_DRV_SETTING_INFO *rdc_setting_info	設定情報用構造体ポインタ
戻り値	unsigned char	処理結果
機能	ドライバに設定されたカウンタ値などの情報を取得します。 • 励磁信号周波数 • 角度検出タイマカウンタの最大値 • モータータイプ	
備考	ST_RDC_DRV_SETTING_INFO は構造体であり、詳細は「6.3.3 R_RSLV_GetRdcDrvSettingInfo API 関数の構造体」を参照してください。	

6.2.4　MTU3 タイマ同時スタート制御 API 関数

項目	内容	
関数名	R_RSLV_MTU_SyncStart	
引数	unsigned char start_ch	同時スタートしたいチャネル （複数 ch 指定）
戻り値	unsigned char	処理結果
機能	MTU3 の指定したチャネルを同時にスタートします。	
備考	角度誤差補正信号を MTU3_0 で設定している場合、角度誤差補正信号タイマを同時にスタートしないでください	

6.2.5　RDC-IC ドライババージョン取得 API 関数

項目	内容	
関数名	R_RSLV_GetDriverVer	
引数	unsigned long *drv_ver	RDC-IC ドライババージョン格納バッファポインタ
戻り値	unsigned char	処理結果
機能	RDC-IC ドライババージョンを指定したバッファに設定します。	
備考	例：0x00010000 のとき、Rev.1.00.00 となります	

6.2.6　角度誤差補正信号出力開始 API 関数

項目	内容	
関数名	R_RSLV_CSig_Start	
引数	unsigned short phase_diff unsigned short amp_level	位相シフト量 振幅レベル
戻り値	unsigned char	処理結果（常に正常終了を示す値が返る）
機能	引数で指定した位相シフト量・振幅レベルに従い、角度誤差補正信号を出力します。 各設定範囲値は、「3.11.3　角度誤差補正信号調整機能」を参照してください	
備考	• 引数に従って角度誤差補正信号出力の設定を行います。 設定変更は、R_RSLV_CSig_Stop で信号停止してから行ってください。	

6.2.7　角度誤差補正信号出力停止 API 関数

項目	内容	
関数名	R_RSLV_CSig_Stop	
引数	void	
戻り値	unsigned char	処理結果（OK のみ）
機能	角度誤差補正信号出力を停止する	
備考	• 本関数をコールすると、信号出力を即停止します。 • 補正信号の設定を変更する場合は、本関数をコールして停止してから R_RSLV_CSig_Start で再設定してください。	

6.2.8　角度誤差補正信号 Duty 更新 API 関数

項目	内容	
関数名	R_RSLV_INT_CSig_UpdatePwmDuty	
引数	void	
戻り値	unsigned char	処理結果
機能	角度誤差補正信号 PWM の Duty の更新を行う処理です。角度誤差補正 Duty 更新タイマ割り込みから呼び出してください。	
備考		

6.2.9　角度誤差補正信号同期スタート API 関数

項目	内容	
関数名	R_RSLV_INT_CSig_SyncStart	
引数	void	
戻り値	unsigned char	処理結果
機能	励磁信号と同期して角度誤差補正信号の出力を開始する処理です。励磁信号と同期した割り込み処理で呼び出してください。	
備考		

6.2.10 角度誤差補正信号出力状態取得 API 関数

項目	内容	
関数名	R_RSLV_GetCSigStatus	
引数	unsigned char *status	角度誤差補正信号出力状態
戻り値	unsigned char	処理結果
機能	角度誤差補正信号出力状態を取得します。 E_OUTPUT_SIGNAL_OFF　：出力停止 E_OUTPUT_SIGNAL_ON　　：出力中 E_OUTPUT_SIGNAL_START：出力開始	
備考		

6.2.11 角度検出タイマスタート API 関数

項目	内容	
関数名	R_RSLV_Capture_Start	
引数	void	
戻り値	unsigned char	処理結果
機能	インプットキャプチャ機能割り込みを有効にして、タイマをスタートします。	
備考		

6.2.12 角度検出値取り込み API 関数

項目	内容	
関数名	R_RSLV_INT_GetCaptureCount	
引数	void	
戻り値	unsigned char	処理結果
機能	インプットキャプチャ機能で検出したカウント値を取得します。 • カウンタ値の取り出しは以下で行います。 現在位置　（立ち下りエッジ）　: R_RSLV_GetAngleCountFirstEdge 前回、現在位置差分（立ち下りエッジ間）: R_RSLV_GetAngleDifferenceFirstEdge 現在位置　（立ち上りエッジ）　: R_RSLV_GetAngleCountSecondEdge 前回、現在位置差分（立ち上りエッジ間）: R_RSLV_GetAngleDifferenceSecondEdge • トリガエッジ情報の取り出しは以下で行います。 R_RSLV_GetCaptureEdge	
備考		

6.2.13 角度検出値取り込み割り込みのトリガ取り出し API 関数

項目	内容	
関数名	R_RSLV_GetCaptureEdge	
引数	unsigned char *cap_edge	角度検出トリガ情報格納変数
戻り値	unsigned char	処理結果
機能	インプットキャプチャ機能による割り込み時のトリガ情報を取り出します。 （ポートレベルにより立ち上がり/立ち下がりの判定が可能）	
備考		

6.2.14 レゾルバ角度カウンタ（取り込みトリガ：立ち下がりエッジ）取り出し API 関数

項目	内容	
関数名	R_RSLV_GetAngleCountFirstEdge	
引数	unsigned short *angle_cnt	カウンタ値格納ポインタ
戻り値	unsigned char	処理結果
機能	インプットキャプチャ機能で検出したカウント値を取り出します。	
備考	• 取得したカウント値は、角度信号の立ち下がりエッジで検出したカウント値です。 • 取得は R_RSLV_INT_GetCaptureCount で行います。	

6.2.15 レゾルバ角度差分カウンタ（取り込みトリガ：立ち下がりエッジ）取り出し API 関数

項目	内容	
関数名	R_RSLV_GetAngleDifferenceFirstEdge	
引数	signed short *angle_diff_cnt	差分値格納ポインタ
戻り値	unsigned char	処理結果
機能	1 サンプリング前と今回のキャプチャ値のカウント差を取り出します。	
備考	• 計算に用いるカウント値は、角度信号の立ち下がりエッジで検出したカウント値です。 • 取得は R_RSLV_INT_GetCaptureCount で行います。	

6.2.16 レゾルバ角度カウンタ（取り込みトリガ：立ち上がりエッジ）取り出し API 関数

項目	内容	
関数名	R_RSLV_GetAngleCountSecondEdge	
引数	unsigned short *angle_cnt	カウンタ値格納ポインタ
戻り値	unsigned char	処理結果
機能	インプットキャプチャ機能で検出したカウント値を取り出します。	
備考	• 取得したカウント値は、角度信号の立ち上がりエッジで検出したカウント値です。 • 取得は R_RSLV_INT_GetCaptureCount で行います。	

6.2.17 レゾルバ角度差分カウンタ（取り込みトリガ：立ち上がりエッジ）取り出し API 関数

項目	内容	
関数名	R_RSLV_GetAngleDifferenceSecondEdge	
引数	signed short *angle_diff_cnt	差分値格納ポインタ
戻り値	unsigned char	処理結果
機能	1 サンプリング前と今回のキャプチャ値のカウント差を取り出します。	
備考	• 計算に用いるカウント値は、角度信号の立ち上がりエッジで検出したカウント値です。 • 取得は R_RSLV_INT_GetCaptureCount で行います。	

6.2.18 励磁信号出力開始 API 関数

項目	内容	
関数名	R_RSLV_ESig_Start	
引数	void	
戻り値	unsigned char	処理結果
機能	励磁信号出力を開始します。	
備考		

6.2.19 励磁信号出力停止 API 関数

項目	内容	
関数名	R_RSLV_ESig_Stop	
引数	void	
戻り値	unsigned char	処理結果
機能	励磁信号出力を停止します。	
備考	励磁信号を停止する場合、角度誤差補正信号と角度検出タイマも併せて停止します。	

6.2.20 励磁信号出力開始タイミング設定 API 関数

項目	内容	
関数名	R_RSLV_ESigCapStartTiming	
引数	unsigned short esig_start_tcnt unsigned short cap_start_tcnt	励磁信号出力開始タイミング設定値 角度検出タイマ開始タイミング設定値
戻り値	unsigned char	処理結果
機能	励磁信号出力開始タイミング、および、角度検出タイマ開始タイミングを設定します。	
備考	設定された値が上限値を越えた場合は上限値を設定し、処理結果として NG を返します。	

6.2.21 待機時間確認カウンタ API 関数

項目	内容	
関数名	R_RSLV_INT_ESigCounter	
引数	void	
戻り値	unsigned char	処理結果
機能	自動調整処理内でのウェイトタイマのカウントダウンを行います。	
備考	自動調整処理実施中のみカウントダウン動作を実施します。	

6.2.22 位相調整信号出力開始 API 関数

項目	内容	
関数名	R_RSLV_Phase_AdjStart	
引数	void	
戻り値	unsigned char	処理結果
機能	位相調整信号の出力を開始する	
備考	F_PHASE_A と F_PHASE_B で指定した位相調整信号用のタイマをスタートします。	

6.2.23 位相調整信号出力停止 API 関数

項目	内容	
関数名	R_RSLV_Phase_AdjStop	
引数	void	
戻り値	unsigned char	処理結果
機能	位相調整信号の出力を停止します。	
備考		

6.2.24 位相調整信号 Duty バッファ設定 API 関数

項目	内容	
関数名	R_RSLV_Phase_AdjUpdateBuff	
引数	unsigned short duty unsigned char ch	設定する Duty 値 A 相／B 相の指定（0：A 相、1：B 相）
戻り値	unsigned char	処理結果
機能	位相調整信号の Duty をバッファに設定します。	
備考		

6.2.25 位相調整信号 Duty レジスタ設定 API 関数

項目	内容	
関数名	R_RSLV_Phase_AdjUpdate	
引数	void	
戻り値	unsigned char	処理結果
機能	位相調整信号の Duty をレジスタに設定します。	
備考	現在値と異なる Duty 値がバッファに設定された場合に位相調整信号の Duty 値を更新します。	

6.2.26 位相調整信号 Duty バッファ読み込み API 関数

項目	内容	
関数名	R_RSLV_Phase_AdjReadBuff	
引数	unsigned short *duty unsigned char ch	位相調整信号 Duty 値 A／B 相指定値（0：A 相、1：B 相）
戻り値	unsigned char	処理結果
機能	位相調整信号の Duty を格納バッファから読み込みます。	
備考		

6.2.27 RDC-IC 初期値設定 API 関数

項目	内容	
関数名	R_RSLV_Rdc_VariableInit	
引数	unsigned char *u1_init_data	RDC-IC 通信初期化データ列ポインタ
戻り値	unsigned char	処理結果
機能	RDC-IC 通信の初期化データを設定します。	
備考	初期化する RDC-IC レジスタ PS1（02h：パワーセーブ制御レジスタ 1） PS2（04h：パワーセーブ制御レジスタ 2） PS3（0Ah：パワーセーブ制御レジスタ 3） ALMOUT（16h：ALARM#出力設定レジスタ） GCGSL（2Eh：差動増幅回路利得選択レジスタ） CSACTL（42h：シャント電流増幅回路制御レジスタ）	

6.2.28 RDC-IC 初期化シーケンス実行 API 関数

項目	内容	
関数名	R_RSLV_Rdc_Init_Sequence	
引数	unsigned short *init_status	初期化処理状態（初期化中／終了）
戻り値	unsigned char	処理結果
機能	RDC-IC 初期化シーケンスを実行します。	
備考		

6.2.29 RDC-IC 通信処理 API 関数

項目	内容	
関数名	R_RSLV_Rdc_Communication	
引数	void	
戻り値	unsigned char	処理結果
機能	RDC-IC との通信を行います。 アプリケーションから繰り返し呼び出されることで通信シーケンスが進行します。	
備考	シーケンス制御を実施するため、周期呼び出ししてください。	

6.2.30 RDC-IC 各レジスタへの書き込み API 関数

項目	内容	
関数名	R_RSLV_Rdc_RegWrite	
引数	unsigned char *write_status	書き込みのステータス
戻り値	unsigned char	処理結果
機能	RDC-IC レジスタに引数で指定した値を書き込みます。	
備考		

6.2.31 RDC-IC 各レジスタの読み込み API 関数

項目	内容	
関数名	R_RSLV_Rdc_RegRead	
引数	unsigned char address	読み込みたい RDC-IC レジスタアドレス
戻り値	unsigned char	処理結果
機能	引数で指定したアドレスの RDC-IC レジスタ値をバッファへ読み込みます。	
備考	読み込んだデータは R_RSLV_Rdc_GetRegisterVal で取得してください。	

6.2.32 RDC-IC レジスタアクセス状態取得 API 関数

項目	内容	
関数名	R_RSLV_Rdc_ChkIfRun	
引数	void	
戻り値	unsigned char	処理結果
機能	RDC-IC レジスタアクセス（R/W）有無を処理結果として返します。	
備考		

6.2.33 RDC-IC レジスタバッファからの読み込み API 関数

項目	内容	
関数名	R_RSLV_Rdc_GetRegisterVal	
引数	unsigned char *rd_data unsigned char address	読み込みデータポインタ 読み込みたい RDC-IC レジスタアドレス
戻り値	unsigned char	処理結果
機能	引数で指定した RDC-IC レジスタのアドレスのバッファ値を取り込みます。	
備考		

6.2.34 RDC-IC レジスタバッファへの書き込み API 関数

項目	内容	
関数名	R_RSLV_Rdc_SetRegisterVal	
引数	unsigned char wt_data unsigned char address	書き込むデータ 書き込む RDC-IC レジスタアドレス
戻り値	unsigned char	処理結果
機能	引数で指定したアドレスの RDC-IC レジスタのバッファに指定したデータを書き込みます。	
備考		

6.2.35 RDC-IC 通信送受信完了割り込みコールバック API 関数

項目	内容	
関数名	R_RSLV_Rdc_CallComEndCb	
引数	void	
戻り値	unsigned char	処理結果
機能	送受信完了割り込みコールバック処理の呼び出し、ドライバの RDC R/W 状態を終了にする。	
備考	送信及び受信割り込み処理で呼び出してください。	

6.2.36 RDC-IC 通信エラー割り込みコールバック API 関数

項目	内容	
関数名	R_RSLV_Rdc_CallErrorCb	
引数	void	
戻り値	unsigned char	処理結果
機能	エラー割り込みコールバック処理の呼び出しを行う。	
備考		

6.2.37 RDC-IC 通信の異常発生有無通知 API 関数

項目	内容	
関数名	R_RSLV_RdcCom_GetErrorInfo	
引数	unsigned char *err_info	RDC-IC 通信異常情報の格納
戻り値	unsigned char	処理結果
機能	RDC-IC 通信時の異常情報を取得します。 RSLV_MD_OK　　：エラー無し RSLV_MD_ERROR：エラー発生	
備考		

6.2.38 RDC-IC ALARM 解除開始 API 関数

項目	内容	
関数名	R_RSLV_Rdc_AlarmCancelStart	
引数	void	
戻り値	unsigned char	処理結果
機能	RDC-IC の ALARM 解除処理を開始します。	
備考		

6.2.39 RDC-IC ALARM 解除シーケンス制御 API 関数

項目	内容	
関数名	R_RSLV_Rdc_AlarmCancel	
引数	void	
戻り値	unsigned char	処理結果
機能	RDC-IC の ALARM 検出状態を解除するシーケンス処理です。	
備考	シーケンス制御を実施するため、周期呼び出ししてください。	

6.2.40 レゾルバ信号ゲインおよび位相調整 API 関数

項目	内容	
関数名	R_RSLV_ADJST_GainPhase	
引数	unsigned char u1_call_state	ユーザ指示ステータス レゾルバ信号ゲイン・位相調整の実行／中止 0：実行 （定数：ADJST_USRREQ_RUN） 1：実行中止 （定数：ADJST_USRREQ_STOP）
戻り値	st_adjst_gainphase_return_t	処理結果
機能	レゾルバ信号のゲイン、及び位相調整シーケンスを実施します。	
備考	st_adjst_gainphase_return_t は構造体であり、レゾルバ信号ゲイン・位相調整の終了情報、ゲイン調整結果、位相調整結果の対応は「6.3.4　R_RSLV_ADJST_GainPhase API 関数の構造体」を参照してください。	

6.2.41 角度誤差補正信号調整 API 関数

項目	内容	
関数名	R_RSLV_ADJST_Carrier	
引数	st_adjst_carrier_arg_t arg_value	ユーザ指示ステータス モータ制御情報
戻り値	st_adjst_carrier_return_t	処理結果
機能	角度誤差補正信号の調整シーケンスを実施します。	
備考	st_adjst_carrier_arg_t、及び st_adjst_carrier_return_t は構造体です。内容詳細は、「6.3.5 R_RSLV_ADJST_Carrier API 関数の構造体」を参照してください。	

6.2.42 ユーザ作成のコールバック関数ポインタ設定 API 関数

項目	内容	
関数名	R_RSLV_ADJST_SetPtrFunc	
引数	st_ptr_func_arg_t *ptr_arg	ユーザ作成の関数ポインタ
戻り値	unsigned char	処理結果
機能	ユーザが作成したコールバック関数のポインタを自動調整処理内部のポインタ変数に設定します。	
備考	st_ptr_func_arg_t は構造体であり、コールバック関数ポインタの設定は「6.3.6 R_RSLV_ADJST_SetPtrFunc API 関数の構造体」を参照してください。	

6.2.43 AD 変換実施状態取得 API 関数

項目	内容	
関数名	R_RSLV_ADJST_Ad_Processing	
引数	void	
戻り値	unsigned char	処理結果（AD 変換実施状態）
機能	AD 変換実施状態を返します。AD 変換実施中の場合は 1 を、そうでない場合は 0 を返します。	
備考		

6.2.44 遅れ位相設定用 API 関数

項目	内容	
関数名	R_RSLV_ADJST_SetFilterDelay	
引数	float bpf_delay_deg float csig_delay_deg	BPF 遅れ位相 [度] 角度誤差補正信号向け LPF 遅れ位相 [度]
戻り値	unsigned char	処理結果
機能	BPF による遅れ位相、及び角度誤差補正信号向け LPF による遅れ位相を設定します。	
備考	ユーザが遅れ位相の値を設定する場合、R_RSLV_SetSystemInfo をコールした後に本関数をコールしてください。	

6.2.45 断線検知処理 API 関数

項目	内容	
関数名	R_RSLV_DiscDetection_Seq	
引数	st_rdc_ddmnt_arg_t arg_value	断線検知処理用構造体
戻り値	unsigned char	処理結果
機能	断線検知のシーケンスを実施します。	
備考	st_rdc_ddmnt_arg_t は構造体です。内容詳細は「6.3.7　R_RSLV_DiscDetection_Seq API 関数の構造体」を参照してください。	

6.3　構造体説明

下記の API 関数は構造体を使用しています。この API 関数の構造体について説明します。

- R_RSLV_SetFuncTable　（6.2.1 章）
- R_RSLV_SetSystemInfo（6.2.2 章）
- R_RSLV_GetRdcDrvSettingInfo（6.2.3 章）
- R_RSLV_ADJST_GainPhase（6.2.40 章）
- R_RSLV_ADJST_Carrier（6.2.41 章）
- R_RSLV_ADJST_SetPtrFunc（6.2.42 章）
- R_RSLV_DiscDetection_Seq（6.2.45 章）

6.3.1　R_RSLV_SetFuncTable API 関数の構造体

R_RSLV_SetFuncTable API 関数の引数 set_func と構造体 ST_FUNCTION_TABLE の定義を以下に示します。

API 関数：R_RSLV_SetFuncTable (unsigned char set_func,
ST_FUNCTION_TABLE user_func_table)

表 6-2　R_RSLV_SetFuncTable API 関数の set_func に指定するマクロ定義名

変数名	型	内容		定義値	マクロ定義名
set_func	unsigned char	機能を指定する	ESIG1	0	F_ESIG1
			ESIG2_1	1	F_ESIG2_1
			ESIG2_2	2	F_ESIG2_2
			ESIG12	3	F_ESIG12
			CSIG	4	F_CSIG
			PHASE_A	5	F_PHASE_A
			PHASE_B	6	F_PHASE_B
			PHASE_AB	7	F_PHASE_AB
			CAPTURE	8	F_CAPTURE
			CSIG_UPD_TIMER	9	F_CSIG_UPD_TIMER
			RDC_COM	10	F_RDC_COM
			RDC_CLK	11	F_RDC_CLK

表 6-3　R_RSLV_SetFuncTable API 関数の構造体定義 (1/2)

構造体	メンバー名	型	内容		定義値	マクロ定義名
ST_FUNCTION_TABLE (引数)	void (*Start)(unsigned char u1_sync_start)	void	タイマ開始関数ポインタ	—	—	—
	void (*Stop)(void)	void	タイマ停止関数ポインタ	—	—	—
	void (*GetTcnt)(unsigned short *tcnt)	void	タイマ値取得関数ポインタ	—	—	—
	void (*SetTcnt)(unsigned short tcnt)	void	タイマ値設定関数ポインタ	—	—	—
	void (*GetDuty)(unsigned short *duty)	void	Duty 値取得関数ポインタ	—	—	—
	void (*SetDuty)(unsigned short duty)	void	Duty 値設定関数ポインタ	—	—	—
	void (*SetDuty_2val)(unsigned short ch, unsigned short duty)	void	Duty 値設定関数ポインタ (PHASE_AB 用)	—	—	—

表 6-3　R_RSLV_SetFuncTable API 関数の構造体定義 (2/2)

構造体	メンバー名	型	内容		定義値	マクロ定義名
ST_FUNCTION_TABLE (引数)	void (*GetCaptureVal)(unsigned short *capture_val)	void	角度検出値取得関数ポインタ	—	—	—
	void (*GetPortLevel)(unsigned char *port_level)	void	ポートレベル取得関数ポインタ	—	—	—
	void (*ComSendReceive) (unsigned short *tx_buf, unsigned short tx_num, unsigned short *rx_buf)	void	RDC-IC 送受信開始関数ポインタ	—	—	—

6.3.2　R_RSLV_SetSystemInfo API 関数の構造体

R_RSLV_SetSystemInfo API 関数の引数 ST_SYSTEM_PARAM と ST_USER_PERI_PARAM の構造体定義を以下に示します。

API 関数：R_RSLV_SetSystemInfo (ST_SYSTEM_PARAM *rdc_sys_param,
ST_USER_PERI_PARAM *user_peri_param)

表 6-4　R_RSLV_SetSystemInfo API 関数の構造体定義

構造体	メンバー名	型	内容		定義値	マクロ定義名
ST_SYSTEM_PARAM（引数）	u1_esig_freq	unsigned char	励磁信号周波数	5kHz	1	R_ESIG_SET_FREQ_5K
				10kHz	2	R_ESIG_SET_FREQ_10K
				20kHz	3	R_ESIG_SET_FREQ_20K
	u1_csig_freq	unsigned char	角度誤差補正信号出力周波数	200kHz	1	R_CSIG_SET_FREQ_200K
				400kHz	2	R_CSIG_SET_FREQ_400K
	u1_csig_upd_duty_cycle	unsigned char	角度誤差補正 Duty 更新回数	2 回	1	R_CSIG_SET_DCNT_02
				4 回	2	R_CSIG_SET_DCNT_04
	u1_mtu3_sync_start	unsigned char	励磁信号および角度検出用タイマ開始フラグ	同期スタート *1	0	SYNCMD_ESIG_API
				同期スタート *2	1	SYNCMD_OTHER_API
	u1_motor_kind	unsigned char	モータータイプ	BLDC 型	1	MOTOR_BLDC
				ステッパー型	2	MOTOR_STM
	u1_mntout_type	unsigned char	RDC-IC MNTOUT 端子出力方式	AC 出力	1	RSLV_MNTOUT_TYPE_AC
				DC 出力	2	RSLV_MNTOUT_TYPE_DC
ST_USER_PERI_PARAM（引数）	f_esig1_peri_clk_src	float	励磁信号出力割当ペリフェラルのカウントクロックソース	—	—	
	f_csig_peri_clk_src	float	角度誤差補正信号出力割当ペリフェラルのカウントクロックソース	—	—	
	f_capture_peri_clk_src	float	角度信号入力割当ペリフェラルのカウントクロックソース	—	—	
	f_csig_upd_timer_peri_clk_src	float	角度誤差補正信号 Duty 更新割当ペリフェラルのカウントクロックソース	—	—	
	f_phase1_peri_clk_src	float	位相調整信号 A 出力割当ペリフェラルのカウントクロックソース	—	—	
	f_phase2_peri_clk_src	float	位相調整信号 B 出力割当ペリフェラルのカウントクロックソース	—	—	

【注】 1. SYNCMD_ESIG_API 指定時は、励磁信号出力開始 API 内で、励磁信号用タイマと角度検出用タイマをカウントスタートします。

2. SYNCMD_OTHER_API 指定時は、MTU3 タイマ同時スタート制御 API または励磁信号割り込み処理から角度検出タイマスタート API を呼び出してカウントスタートします。

6.3.3　R_RSLV_GetRdcDrvSettingInfo API 関数の構造体

R_RSLV_GetRdcDrvSettingInfo API 関数の引数 ST_RDC_DRV_SETTING_INFO の構造体定義を以下に示します。

API 関数：R_RSLV_GetRdcDrvSettingInfo (ST_RDC_DRV_SETTING_INFO *rdc_setting_info)

表 6-5　R_RSLV_GetRdcDrvSettingInfo API 関数の構造体定義

構造体	メンバー名	型	内容			備考
ST_RDC_DRV_SETTING_INFO（引数）	f_esig_freq	float	励磁周波数 5kHz：5000、10kHz：10000、20kHz：20000			
	u2_capture_cnt_max	unsigned short	角度検出タイマカウンタの最大値			
	u1_motor_kind	unsigned char	モータータイプ	定義値	マクロ定義名	
			BLDC 型	1	MOTOR_BLDC	
			ステッパー型	2	MOTOR_STM	

6.3.4　R_RSLV_ADJST_GainPhase API 関数の構造体

R_RSLV_ADJST_GainPhase API 関数の戻り値 st_adjst_gainphase_return_t の構造体定義を以下に示します。

API 関数：st_adjst_gainphase_return_t R_RSLV_ADJST_GainPhase (unsigned char u1_call_state)

表 6-6　R_RSLV_ADJST_GainPhase API 関数の構造体定義（1/2）

構造体	メンバー名	型	内容		定義値	マクロ定義名
st_adjst_gainphase_return_t (戻り値)	u1_adjst_state	unsigned char	実行中	内部処理待ち	0	ADJST_APIINFO_RUN_MODE
			正常終了	位相調整が正常に終了	1	ADJST_APIINFO_END_NORMAL
			ゲイン調整：増幅上限エラー終了	RDC-IC のレゾルバ A 相信号増幅上限に達しても調整合格範囲に入らなかった場合	3	ADJST_APIINFO_ERR_GAIN_HI_LMT
			ゲイン調整：増幅下限エラー終了	RDC-IC のレゾルバ A 相信号増幅下限に達しても調整合格範囲に入らなかった場合	4	ADJST_APIINFO_ERR_GAIN_LO_LMT
			ゲイン調整：ゲイン不定エラー終了	RDC-IC のレゾルバ A 相信号調整において調整合格範囲に入らなかった場合	5	ADJST_APIINFO_ERR_GAIN_SWAY
			位相調整：A 相 duty 上限、B 相 duty 下限エラー終了	A 相 duty 上限、B 相 duty 下限に達しても調整合格範囲に入らなかった場合	6	ADJST_APIINFO_ERR_PHASE_AHI_BLO
			位相調整：A 相 duty 下限、B 相 duty 上限エラー終了	A 相 duty 下限、B 相 duty 上限に達しても調整合格範囲に入らなかった場合	7	ADJST_APIINFO_ERR_PHASE_ALO_BHI
			位相調整：位相不定エラー終了	B 相 duty が上下限に到達せず、調整合格範囲に入らなかった場合	8	ADJST_APIINFO_ERR_PHASE_SWAY
			位相調整：位相調整不可エラー終了	A 相カウント、B 相カウントの差が調整可能範囲を超えていた場合	9	ADJST_APIINFO_ERR_PHASE_OUT_RANGE
			ゲイン・位相調整：RDC-IC エラー終了	モニタ信号取得、AB 相カウント取得時に正常動作しなかった場合	10	ADJST_APIINFO_ERR_RDC
			中止による終了	u1_call_state による実行中止の場合	13	ADJST_APIINFO_END_USER_STOP
	u1_res_dlcgsl	unsigned char	u1_adjst_state 実行中（0）	—	0xFF	—
			u1_adjst_state 正常終了（1）	RDC-IC レジスタ DLCGSL 調整結果	0～31	—
			u1_adjst_state エラー（3～10，13）	調整前の RDC-IC レジスタ DLCGSL ユーザ設定値	—	—
	u2_res_a_duty	unsigned short	u1_adjst_state 実行中（0）	—	0xFFFF	—
			u1_adjst_state 正常終了（1）	A 相 PWM duty 調整結果 [%]	5～90	—
			u1_adjst_state エラー（3～10，13）	調整前の A 相 PWM duty ユーザ設定値	—	—

表 6-6　R_RSLV_ADJST_GainPhase API 関数の構造体定義（2/2）

構造体	メンバー名	型	内容		定義値	マクロ定義名
st_adjst_gainphase_return_t (戻り値)	u2_res_b_duty	unsigned short	u1_adjst_state 実行中（0）	—	0xFFFF	—
			u1_adjst_state 正常終了（1）	B 相 PWM duty 調整結果 [%]	5～90	—
			u1_adjst_state エラー（3～10，13）	調整前の B 相 PWM duty ユーザ設定値	—	—

6.3.5　R_RSLV_ADJST_Carrier API 関数の構造体

R_RSLV_ADJST_Carrier API 関数の戻り値 st_adjst_carrier_return_t と引数 st_adjst_carrier_arg_t の構造体定義を以下に示します。

API 関数：st_adjst_carrier_return_t R_RSLV_ADJST_Carrier (st_adjst_carrier_arg_t arg_value)

表 6-7　R_RSLV_ADJST_Carrier API 関数の構造体定義

構造体	メンバー名	型	内容		定義値	マクロ定義名
st_adjst_carrier_return_t（戻り値）	adjst_state	unsigned char	角度誤差補正信号調整の実行状況	実行中	0	ADJST_APIINFO_RUN_MODE
				正常終了	1	ADJST_APIINFO_END_NORMAL
				制御完了待ち	2	ADJST_APIINFO_WAITING
				角度誤差補正エラー終了	11	ADJST_APIINFO_ERR_CARRIER
				モータ回転不可エラー終了	12	ADJST_APIINFO_ERR_MOTOR
				中止による終了	13	ADJST_APIINFO_END_USER_STOP
	req_mtr_ctrl	unsigned char	角度誤差補正信号調整のモータ制御要求	制御要求なし	0	ADJST_APIREQ_NONE
				位置制御要求	1	ADJST_APIREQ_POS_CTRL
				位置制御停止要求	2	ADJST_APIREQ_POS_STOP
				速度制御要求	3	ADJST_APIREQ_SPD_CTRL
				速度制御停止要求	4	ADJST_APIREQ_SPD_STOP
	mtr_ctrl_data	unsigned short	req_mtr_ctrl (1)	位置制御角度	0～360	—
			req_mtr_ctrl (3)	速度データ[rpm]	—	—
	res_ccgsl	unsigned char	調整結果	調整中	0xFF	—
				調整完了時	0～5	—
				エラー終了時	ユーザ設定値	—
	res_csig_shift	unsigned short	調整結果位相シフト量	調整中	0xFF	—
				調整完了時	*	—
				エラー終了時	ユーザ設定値	—
	res_csig_amp	unsigned short	調整結果振幅値	調整中	0xFF	—
				調整完了時　CSIG 200kHz CSIG 400kHz	*	—
				エラー終了時	ユーザ設定値	—
st_adjst_carrier_arg_t（引数）	call_state	unsigned char	角度誤差補正信号調整の実行／中止	実行継続	0	ADJST_USRREQ_RUN
				実行中止	1	ADJST_USRREQ_STOP
	req_state	unsigned char	モータ制御の実行状況	モータ制御完了	0	ADJST_USRINFO_COMPLETE
				モータ制御実行中	1	ADJST_USRINFO_PROCESSING

【注】 *　定義値は、「3.11.3　角度誤差補正信号調整機能」を参照してください

6.3.6　R_RSLV_ADJST_SetPtrFunc API 関数の構造体

R_RSLV_ADJST_SetPtrFunc API 関数の引数 st_ptr_func_arg_t の構造体定義を以下に示します。

API 関数：void R_RSLV_ADJST_SetPtrFunc (st_ptr_func_arg_t *ptr_arg)

表 6-8　R_RSLV_ADJST_SetPtrFunc API 関数の構造体定義

構造体	メンバー名	型	内 容		定義値	マクロ定義名
st_ptr_func_arg_t（引数）	(*ad_data)(void);	unsigned short	A/D データ参照の関数ポインタ	—	—	—
	(*ad_ctrl)(unsigned char);	void	A/D 変換開始／停止の関数ポインタ	—	—	—
	(*ad_peri_adjst)(void);	void	A/D コンバータ設定調整機能用の関数ポインタ	—	—	—
	(*ad_peri_user)(void);	void	A/D コンバータ設定ユーザ作成の関数ポインタ	—	—	—
	resolver_pole_num	unsigned short	使用するモータのレゾルバ極数	—	—	—
	*mtr_speed	float	速度データ参照の変数ポインタ	[rad/s]	—	—
	req_speed	unsigned short	誤差の自動調整実行時の指令速度	[rpm]	—	—

6.3.7　R_RSLV_DiscDetection_Seq API 関数の構造体

R_RSLV_DiscDetection_Seq API 関数の引数 st_rdc_ddmnt_arg_t の構造体定義を以下に示します。

API 関数：unsigned char R_RSLV_DiscDetection_Seq (st_rdc_ddmnt_arg_t arg_value)

表 6-9　R_RSLV_DiscDetection_Seq API 関数の構造体定義

構造体	メンバー名	型	内 容		定義値	マクロ定義名
st_rdc_ddmnt_arg_t（引数）	call_state	unsigned char	断線検知処理の実行状況	実行中	0	DDMNT_APIINFO_RUN_MODE
				断線未検知	1	DDMNT_APIINFO_END_NOMAL
				断線検知	2	DDMNT_APIINFO_ERR_DISCONNECT
				中止による終了	3	DDMNT_APIINFO_ENC_USER_STOP
	wire_state	unsigned char	レゾルバ線の状態	正常	0	DDMNT_WIRE_STATE_NOMAL
				異常	1	DDMNT_WIRE_STATE_ABNOMAL

7.　API 関数の実装例

以下に本ドライバを用いたソフトウェアアーキテクチャの一例を示します。

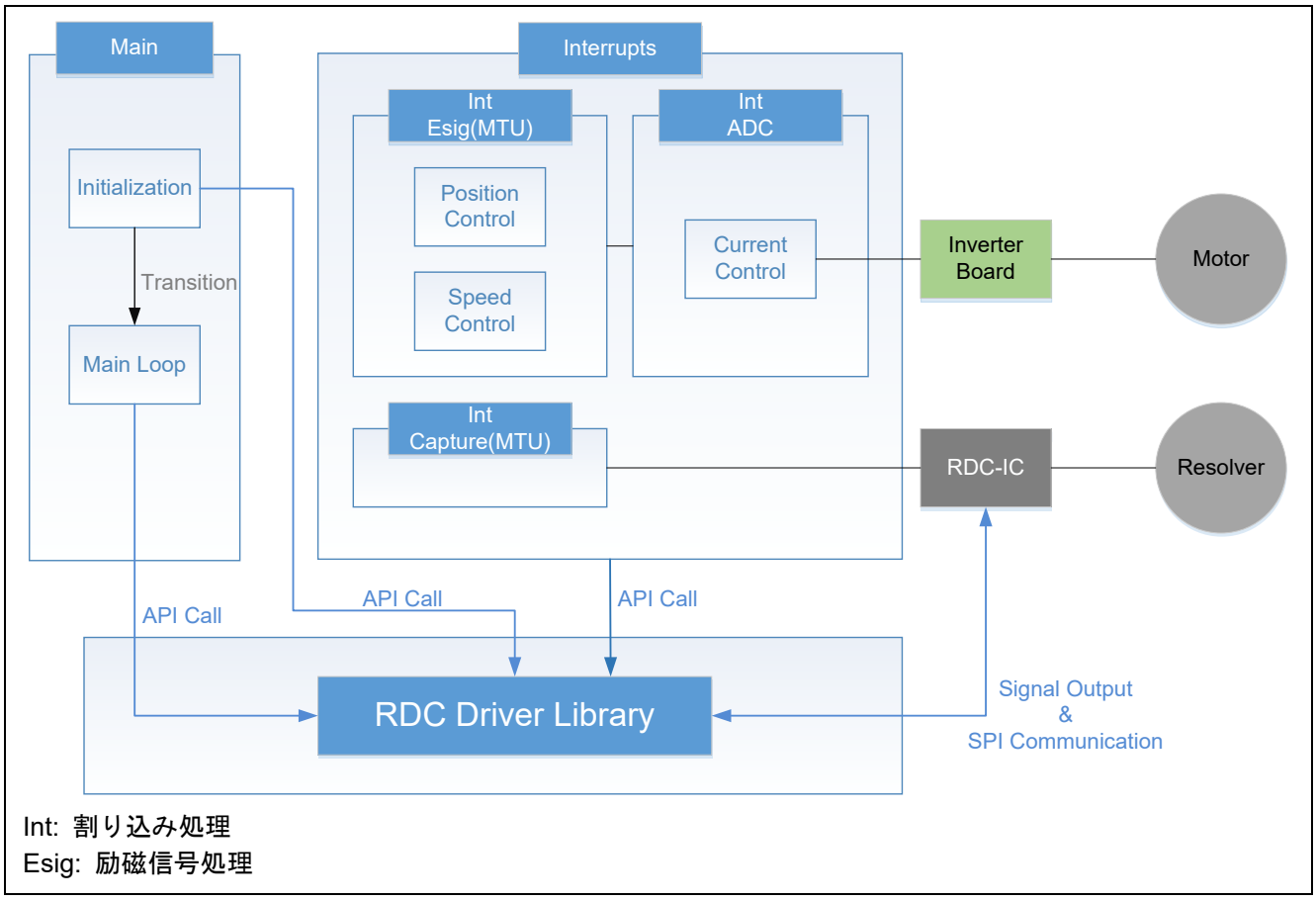

図 7.1　ソフトウェアアーキテクチャ例

本ドライバは初期化処理により初期化され、メインループで信号の開始等の実行処理を、割り込み処理でロータ位置情報の取り込み（インプットキャプチャ機能）や信号同期処理などを API 呼び出しにより実施します。また、RDC-IC との SPI 通信と各信号出力を行います。

以下にそれぞれの実装方法詳細を記載します。

7.1　ペリフェラル準備

ペリフェラル設定関数は、SC を使用してユーザが作成します。SC で生成可能な設定処理は、ペリフェラル初期化、ペリフェラルタイマスタート、ペリフェラルタイマストップ関数となります。その他、必要な関数については、ユーザ作成コードとして別途作成する必要があります。

本サンプルコードにはユーザ作成コードも含む、処理関数が用意されていますので、必要に応じてご使用ください。

7.1.1　SC 設定

SC を使用して、本ドライバが実装する各ドライバ機能のペリフェラル設定を行います。各ドライバ機能に割り当てる推奨ペリフェラル設定については、「5.　ペリフェラル設定」を参照してください。SC を使用すると、Config_(peri_func).c、Config_(peri_func)_user.c、Config_(peri_func).h の各ファイルが生成されます。ファイル名の説明は、「4.1　フォルダ・ファイル構成」を参照してください。

7.1.2　ユーザ作成コード

SC 出力コードの他に、関数テーブルに設定するペリフェラルアクセス関数を作成する必要があります。作成が必要な関数は下記になります。

- タイマカウンタ値取得関数
- タイマカウンタ値設定関数
- Duty 値取得関数
- Duty 値設定関数
- キャプチャ値取得関数
- ポートレベル取得関数

関数テーブルとの関連については、「5.4　関数テーブル設定」を参照してください。

7.2　初期化処理

7.2.1　MCU 初期化

SC でコードを出力すると、自動的に R_Systeminit 関数が作成されてペリフェラル初期化関数がこの関数に組み込まれます。MCU 起動時に R_Systeminit が呼び出されることで各ペリフェラルが初期化されます。

ペリフェラル初期化関数：R_Config_(peri_func)_Create()

7.2.2　ドライバ初期化

本ドライバを初期化するには、MCU 初期化後に下記の処理を実施します。

- システム情報設定
- 励磁信号と角度信号入力のタイマスタートタイミング設定
- 関数テーブル設定
- 自動調整用コールバック関数ポインタ設定
- RDC-IC レジスタ初期値設定

サンプルコードは、「7.2.3.2　ドライバ初期化」を参照してください。

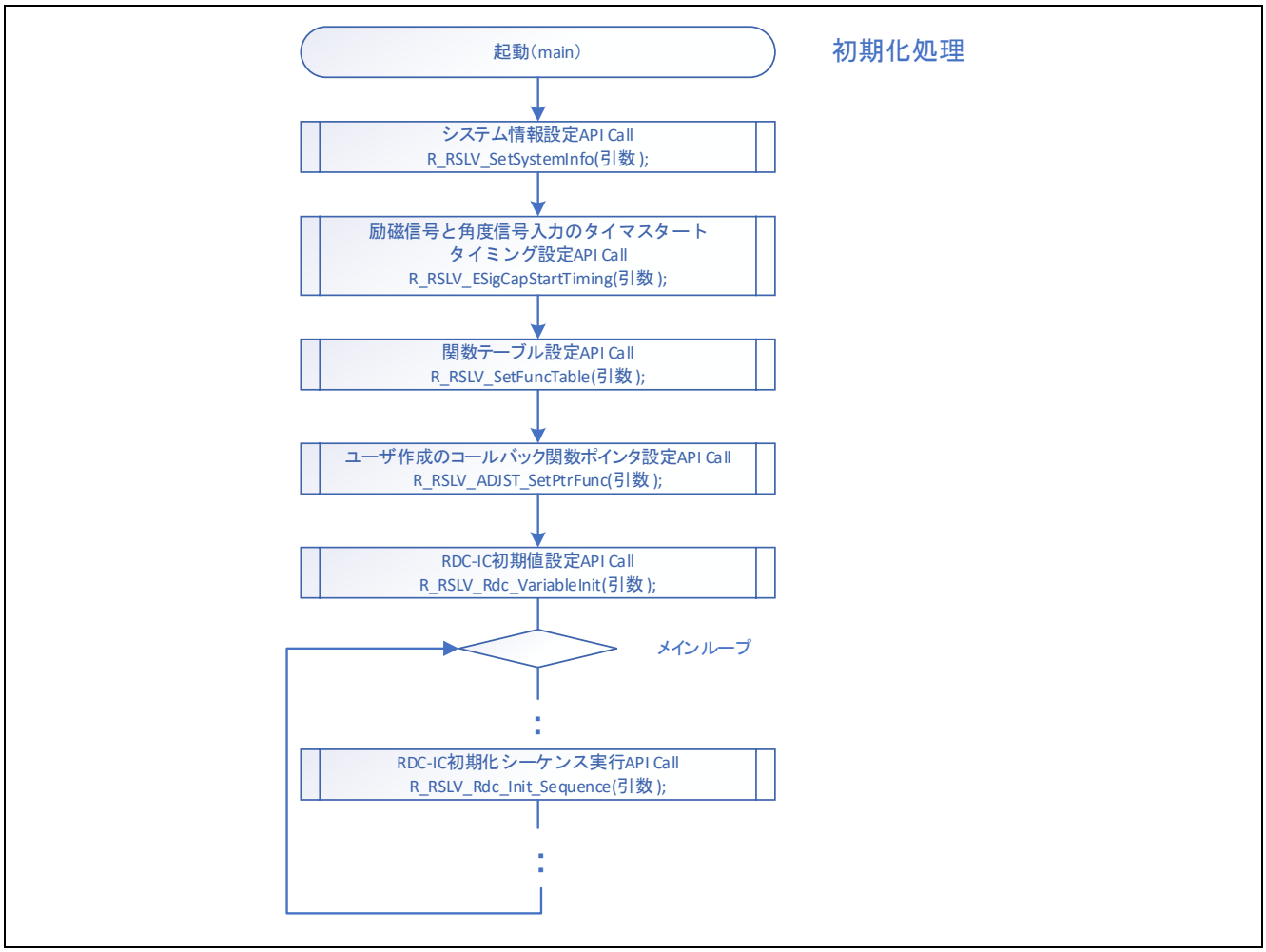

図 7.2　初期化フロー

7.2.2.1　システム情報設定

システム情報を設定するには、励磁周波数、角度誤差補正信号周波数、角度誤差補正信号の更新回数、各ドライバ機能のペリフェラルに対するクロックソースなどを指定し、システム情報設定 API 関数で設定します。なお、カウントクロックを分周する場合は、クロックソース／分周比を、指定してください。

サンプルコードは、「7.2.3.2　ドライバ初期化」を参照してください。

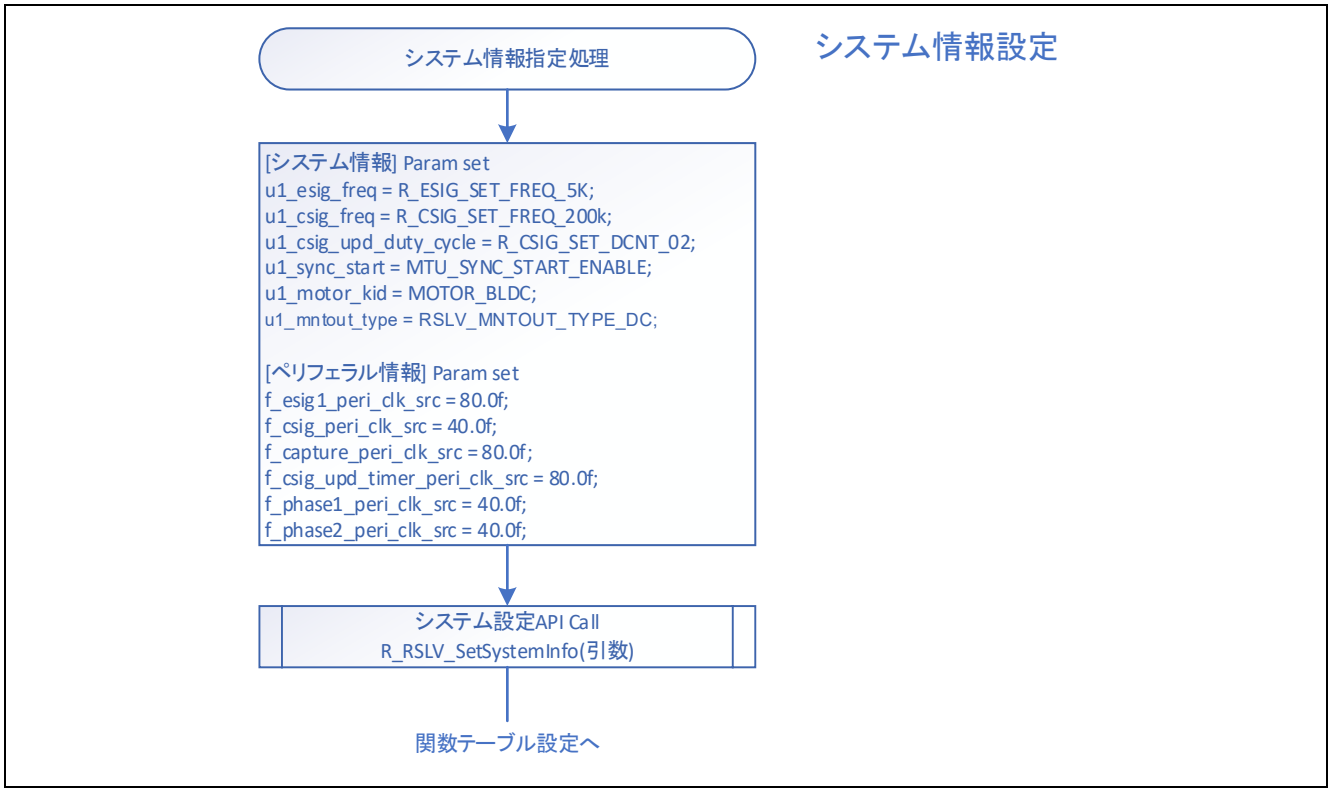

図 7.3　システム情報指定処理

7.2.2.2　励磁信号と角度信号入力のタイマスタートタイミング設定

励磁信号と角度信号入力のタイマスタートタイミングを設定するには、励磁信号出力開始タイミング設定 API 関数で設定します。サンプルコードではドライバ初期化処理内で実施していますが、励磁信号および角度信号入力のタイマスタート前であれば、どこで設定しても構いません。

サンプルコードは、「7.2.3.2　ドライバ初期化」を参照してください。

API 関数：**R_RSLV_ESigCapStartTiming**(DEF_DELAY_ADJ_ESIG, DEF_SFT_ADJ_ESIG);

7.2.2.3　関数テーブル設定

関数テーブルを設定するには、SC で出力したコードまたはユーザ作成コードを該当するテーブルにセットして、関数テーブル設定 API 関数で設定します。

サンプルコードは、「7.2.3.2　ドライバ初期化」を参照してください。

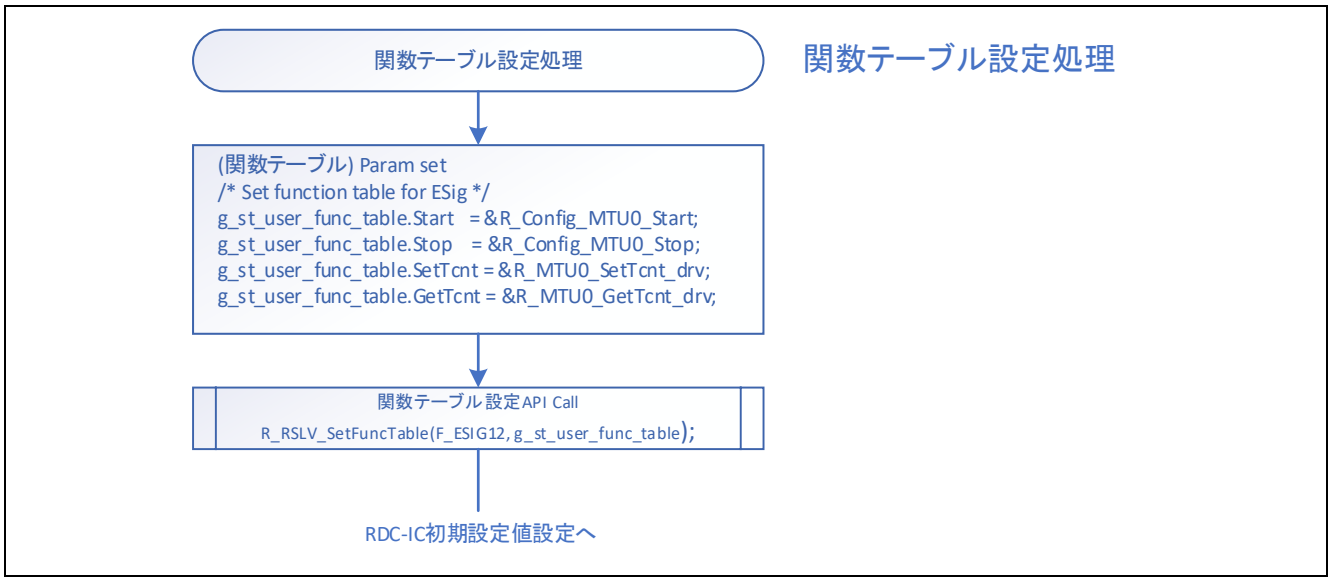

図 7.4　関数テーブル設定処理

7.2.2.4　ユーザ作成のコールバック関数ポインタ設定

自動調整用コールバック関数ポインタを設定するには、自動調整用の AD 変換関数ポインタと調整処理に必要な値を該当する構造体のメンバーにセットし、ユーザ作成のコールバック関数ポインタ設定 API で設定します。

サンプルコードは、「7.2.3.2　ドライバ初期化」を参照してください。

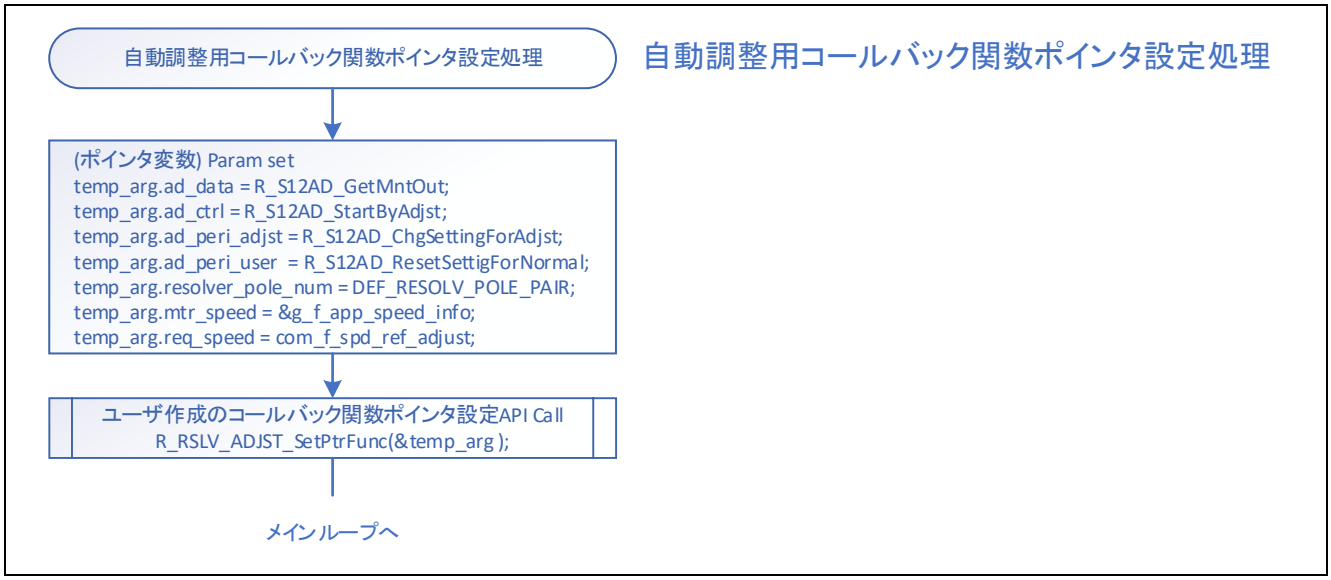

図 7.5　自動調整用初期設定（関数ポインタ設定）処理

7.2.2.5　RDC-IC 初期設定

RDC-IC のレジスタを初期化するためには、RDC-IC 初期値設定 API を使用します。各レジスタ初期値は、ユーザが指定します。設定レジスタは「6.2.27　RDC-IC 初期値設定 API 関数」を参照ください。

サンプルコードは、「7.2.3.2　ドライバ初期化」を参照してください。

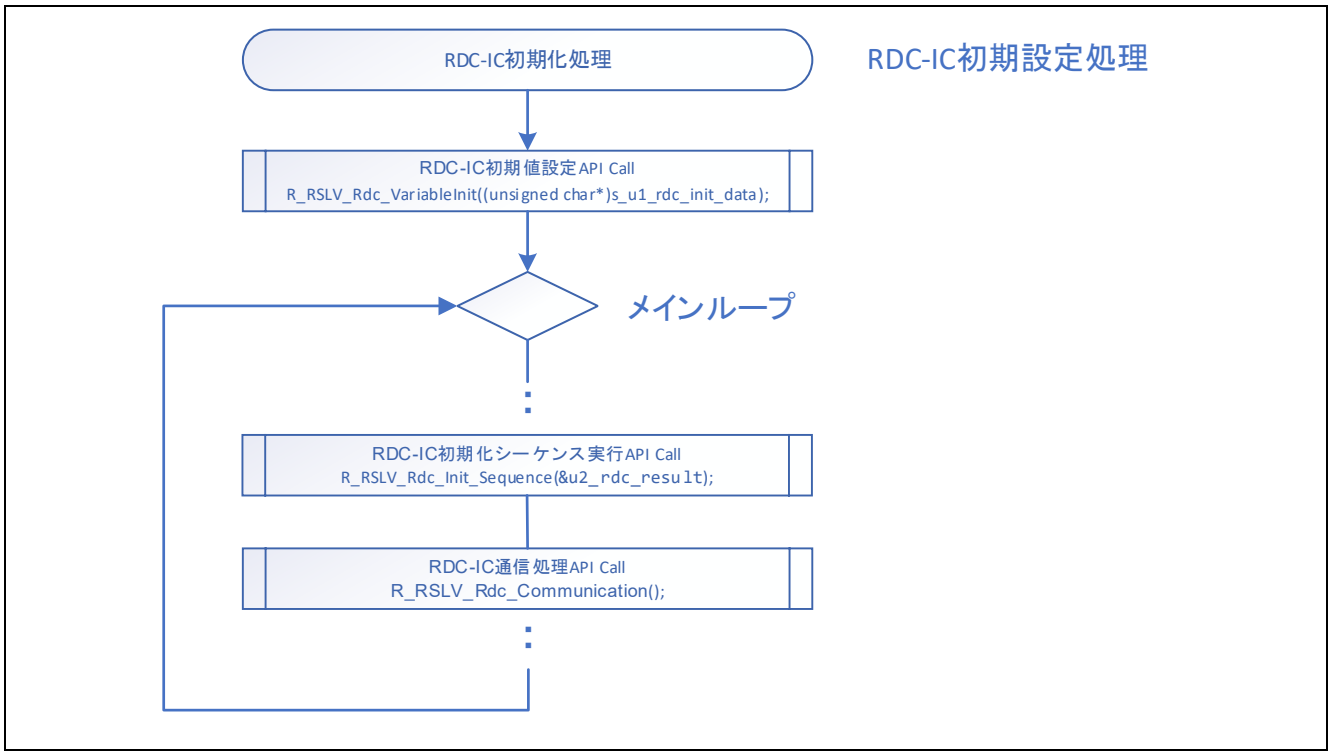

図 7.6　RDC-IC 初期化処理

RDC-IC 初期値を設定した後は、メインループで RDC-IC 初期化シーケンス API を実行します。本 API の引数から初期化状況を取得出来るので、終了するまで継続して実行してください。また RDC-IC 初期化シーケンス API は RDC-IC 通信を行うため、RDC-IC 通信処理 API もメインループで実行してください。

7.2.3　サンプルコード

7.2.3.1　MCU 初期化（ペリフェラル初期化）

下記にペリフェラル初期化コーディング例を記載します。本コーディング例は、SC が出力する初期設定関数 R_Systeminit を使用する例です。R_Systeminit を使わない場合は、「8.2.1　ペリフェラル初期化処理」の移行例を参照してください。

```
/***********************************************************************
* Function Name: R_Systeminit
* Description  : This function initializes every configuration.
* Arguments    : None
* Return Value : None
***********************************************************************/
void R_Systeminit(void)
{
    /* Enable writing to registers related to operating modes, LPC, CGC, and
software reset */
    SYSTEM.PRCR.WORD = 0xA50BU;

    /* Enable writing to MPC pin function control registers. */
    MPC.PWPR.BIT.B0WI = 0U;
    MPC.PWPR.BIT.PFSWE = 1U;

    /* Write 0 to the target bits in the POECR2 register. */
    POE.POECR2.WORD = 0x0000U;

    /* Initialize clock settings. */
    R_CGC_Create();

    /* Make peripheral module settings. */
    R_Config_RSPI0_RdcCom_Create();
    R_Config_TMR0_PhaseA_Create();
    R_Config_TMR3_RdcClk_Create();
    R_Config_TMR4_PhaseB_Create();
    R_Config_MTU2_Cap_Create();
    R_Config_CMT1_CsigUpdTim_Create();
    R_Config_MTU0_Csig_Create();
    R_Config_MTU9_Esig_Create();

    /* Make interrupt settings. */
    R_Interrupt_Create();

    /* Register undefined interrupt. */
    R_BSP_InterruptWrite(BSP_INT_SRC_UNDEFINED_INTERRUPT,(bsp_int_cb_t)
    r_undefined_exception);

    /* Register group BL0 interrupt TEI5 (SCI5). */
    R_BSP_InterruptWrite(BSP_INT_SRC_BL0_SCI5_TEI5,(bsp_int_cb_t)
    r_Config_SCI5_transmitend_interrupt);

    /* Register group BL0 interrupt ERI5 (SCI5). */
    R_BSP_InterruptWrite(BSP_INT_SRC_BL0_SCI5_ERI5,(bsp_int_cb_t)
    r_Config_SCI5_receiveerror_interrupt);

    /* Register group BL0 interrupt TEI12 (SCI12). */
    R_BSP_InterruptWrite(BSP_INT_SRC_BL0_SCI12_TEI12,(bsp_int_cb_t)
    r_Config_SCI12_transmitend_interrupt);
```

```
    /* Register group BL0 interrupt ERI12 (SCI12). */
    R_BSP_InterruptWrite(BSP_INT_SRC_BL0_SCI12_ERI12,(bsp_int_cb_t)
    r_Config_SCI12_receiveerror_interrupt);

    /* Register group BL1 interrupt OEI2 (POE3). */
    R_BSP_InterruptWrite(BSP_INT_SRC_BL1_POE3_OEI2,(bsp_int_cb_t)
    r_Config_POE_oei2_interrupt);

    /* Register group BL1 interrupt OEI3 (POE3). */
    R_BSP_InterruptWrite(BSP_INT_SRC_BL1_POE3_OEI3,(bsp_int_cb_t)
    r_Config_POE_oei3_interrupt);

    /* Register group AL0 interrupt SPII0 (RSPI0). */
    R_BSP_InterruptWrite(BSP_INT_SRC_AL0_RSPI0_SPII0,(bsp_int_cb_t)
    r_Config_RSPI0_idle_interrupt);

    /* Register group AL0 interrupt SPEI0 (RSPI0). */
    R_BSP_InterruptWrite(BSP_INT_SRC_AL0_RSPI0_SPEI0,(bsp_int_cb_t)
    r_Config_RSPI0_error_interrupt);

    /* Disable writing to MPC pin function control registers. */
    MPC.PWPR.BIT.PFSWE = 0U;
    MPC.PWPR.BIT.B0WI = 1U;

    /* Enable protection. */
    SYSTEM.PRCR.WORD = 0xA500U;
}
```

7.2.3.2　ドライバ初期化

下記にドライバ初期化のコーディング例を記載します。本処理はメイン処理から呼び出すようにしてください。呼び出し元のコード例は、「7.3.2　サンプルコード」（メインループ）を参照してください。

```
/***********************************************************************
* Function Name : R_RSLVADP_Init
* Description   : Resolver related processing initialization
* Arguments     : None
* Return Value  : None
***********************************************************************/
void R_RSLVADP_Init (void)
{

    /* Setting of function for resolver */
    RESOLVER_init_func();     // システム情報設定、および、関数テーブル設定

    ////////////////////////////////////////
    /// RDC 初期値設定
    ////////////////////////////////////////
    /* Initializes RDC register values. */
    R_RSLV_Rdc_VariableInit((unsigned char*)s_u1_rdc_init_data);

    /* Get resolver settings. */
    R_RSLV_GetRdcDrvSettingInfo(&st_drv_info);

}
```

下記に、システム情報と関数テーブル設定のコーディング例を記載します。本処理はメイン処理から呼び出すようにしてください。コーディング例は、上記の R_RSLVADP_Init()から呼び出すようにしています。

```
/***********************************************************************
* Function Name : RESOLVER_init_func
* Description   : Resolver driver system initialization
* Arguments     : None
* Return Value  : None
***********************************************************************/
static void RESOLVER_init_func(void)
{
    ST_SYSTEM_PARAM    st_system_param;
    ST_USER_PERI_PARAM st_user_peri_param;

    /* Initialize GPIO to output a low level as the reset signal and place the
       RDC in the reset state. */
    /* STM board uses P43 as RDC reset pin */
    PORT4.PODR.BIT.B3   = 0;
    PORT4.PDR.BIT.B3    = 1;

    ////////////////////////////
    /// システム情報設定
    ////////////////////////////
    /* Excitation signal (ESig) frequency 20 kHz */
    st_system_param.u1_esig_freq = R_ESIG_SET_FREQ_20K;
    /* Correction signal (CSig) frequency 200 kHz */
    st_system_param.u1_csig_freq = R_CSIG_SET_FREQ_200K;
    /* Update the duty cycle 2 times.*/
    st_system_param.u1_csig_upd_duty_cycle = R_CSIG_SET_DCNT_02;
    /* Use MTU synchronous start. */
    st_system_param.u1_sync_start = SYNCMD_OTHER_API;
    /* Target motor is a STM motor. */
    st_system_param.u1_motor_kind = MOTOR_STM;
    /* RDC IC MNTOUT output method */
    st_system_param.u1_mntout_type = RSLV_MNTOUT_TYPE_AC;


    st_user_peri_param.f_esig1_peri_clk_src = 80.0f;
    st_user_peri_param.f_csig_peri_clk_src = 80.0f;
    st_user_peri_param.f_csig_upd_timer_peri_clk_src = 80.0f;
    st_user_peri_param.f_capture_peri_clk_src = 80.0f;
    st_user_peri_param.f_phase1_peri_clk_src = 40.0f;
    st_user_peri_param.f_phase2_peri_clk_src = 40.0f;

    R_RSLV_SetSystemInfo(&st_system_param, &st_user_peri_param);

    ////////////////////////////////////////////////////
    /// 励磁信号と角度信号入力のタイマスタートタイミング設定
    ////////////////////////////////////////////////////
    /* Esig & Capture start timing*/
    R_RSLV_ESigCapStartTiming(DEF_DELAY_ADJ_ESIG, DEF_SFT_ADJ_ESIG);

    ////////////////////////////////////////
    /// 関数テーブル設定（励磁信号出力）
    ////////////////////////////////////////
    /* Set up the function table for ESig. */
    g_st_user_func_table.Start   = &R_Config_MTU9_Esig12_Start;
```

```
    g_st_user_func_table.Stop    = &R_Config_MTU9_Esig12_Stop;
    g_st_user_func_table.SetTcnt = &R_Config_MTU9_Esig12_SetTcnt;
    g_st_user_func_table.GetTcnt = &R_Config_MTU9_Esig12_GetTcnt;
    R_RSLV_SetFuncTable(F_ESIG12, g_st_user_func_table);

    ////////////////////////////////////////
    /// 関数テーブル設定（角度誤差補正信号出力）
    ////////////////////////////////////////
    /* Set up the function table for CSig. */
    g_st_user_func_table.Start   = &R_Config_MTU0_Csig_Start;
    g_st_user_func_table.Stop    = &R_Config_MTU0_Csig_Stop;
    g_st_user_func_table.SetTcnt = &R_Config_MTU0_Csig_SetTcnt;
    g_st_user_func_table.GetTcnt = &R_Config_MTU0_Csig_GetTcnt;
    g_st_user_func_table.SetDuty = &R_Config_MTU0_Csig_SetDuty;
    g_st_user_func_table.GetDuty = &R_Config_MTU0_Csig_GetDuty;
    R_RSLV_SetFuncTable(F_CSIG, g_st_user_func_table);

    ////////////////////////////////////////
    /// 関数テーブル設定（角度信号入力）
    ////////////////////////////////////////
    /* Set up the function table for Capture. */
    g_st_user_func_table.Start   = &R_Config_MTU2_Cap_Start;
    g_st_user_func_table.Stop    = &R_Config_MTU2_Cap_Stop;
    g_st_user_func_table.SetTcnt = &R_Config_MTU2_SetTcnt;
    g_st_user_func_table.GetTcnt = &R_Config_MTU2_GetTcnt;
    g_st_user_func_table.GetCaptureValue = &R_Config_MTU2_GetCapVal;
    g_st_user_func_table.GetPortLevel = &R_Config_MTU2_GetPortLvl;
    R_RSLV_SetFuncTable(F_CAPTURE, g_st_user_func_table);

    ////////////////////////////////////////
    /// 関数テーブル設定（RDC-IC クロック）
    ////////////////////////////////////////
    /* Set up the function table for RDC IC clock. */
    g_st_user_func_table.Start   = &R_Config_TMR3_RdcClk_Start;
    g_st_user_func_table.Stop    = &R_Config_TMR3_RdcClk_Stop;
    R_RSLV_SetFuncTable(F_RDC_CLK, g_st_user_func_table);

    ////////////////////////////////////////
    /// 関数テーブル設定（位相調整信号出力 A）
    ////////////////////////////////////////
    /* Set up the function table for phase A/B. */
    g_st_user_func_table.Start   = &R_Config_TMR0_PhaseA_Start;
    g_st_user_func_table.Stop    = &R_Config_TMR0_PhaseA_Stop;
    g_st_user_func_table.SetDuty = &R_Config_TMR0_PhaseA_SetDuty;
    R_RSLV_SetFuncTable(F_PHASE_A, g_st_user_func_table);

    ////////////////////////////////////////
    /// 関数テーブル設定（位相調整信号出力 B）
    ////////////////////////////////////////
    /* Set up the function table for phase A/B. */
    g_st_user_func_table.Start   = &R_Config_TMR4_PhaseB_Start;
    g_st_user_func_table.Stop    = &R_Config_TMR4_PhaseB_Stop;
    g_st_user_func_table.SetDuty = &R_Config_TMR4_PhaseB_SetDuty;
    R_RSLV_SetFuncTable(F_PHASE_B, g_st_user_func_table);

    ////////////////////////////////////////
    /// 関数テーブル設定（RDC 通信機能）
    ////////////////////////////////////////
```

```
    /* Set up the function table for phase B. */
    g_st_user_func_table.ComSendReceive =
    & R_Config_RSPI0_RdcCom_Send_Receive;
    R_RSLV_SetFuncTable(F_RDC_COM, g_st_user_func_table);

}
```

下記にユーザ作成の自動調整用コールバック関数ポインタ設定のコーディング例を記載します。本処理はメイン処理から呼び出すようにしてください。

呼び出し元のコード例は、「7.3.2　サンプルコード」（メインループ）を参照してください。

```
/***********************************************************************
* Function Name: r_mtr_init_adjst_interface
* Description  : Initialize interface functions and variables with library
* Arguments    : void
* Return Value : void
***********************************************************************/
void r_mtr_init_adjst_interface( void )
{
    st_ptr_func_arg_t    temp_arg;

    temp_arg.ad_data = R_S12AD_GetMntOut;
    temp_arg.ad_ctrl = R_S12AD_StartByAdjst;
    temp_arg.ad_peri_adjst = R_S12AD_ChgSettingForAdjst;
    temp_arg.ad_peri_user  = R_S12AD_ResetSettigForNormal;
    temp_arg.resolver_pole_num = DEF_RESOLV_POLE_PAIR;
    temp_arg.mtr_speed = &(mtr_p[0]->spd_ctrl.f_speed);
    temp_arg.req_speed = com_f_spd_ref;

    R_RSLV_ADJST_SetPtrFunc( &temp_arg );
}
```

下記に RDC-IC 初期化シーケンスのコーディング例を記載します。本処理はメインループから呼び出すようにしてください。

```
/***********************************************************************
* Function Name : R_RSLVADP_MainLoopProcess
* Description   : Resolver management process for main loop
* Arguments     : None
* Return Value  : None
***********************************************************************/
void R_RSLVADP_MainLoopProcess(void)
{
    uint16_t rdc_result = RSLV_MD_BUSY1;

    resolver_csig_ui();

    if (TRUE == com_u1_flg_rdc_sequence)
    {
         g_u1_flg_rdc_state_ready = FALSE;

        if(RDC_RESET_STATE_NON == g_u1_rdc_reset_wait_status)
        {
            write_rdc_reset_gpio(1);
            g_u1_rdc_reset_wait_status = RDC_RESET_STATE_ACT;
            g_u2_rdc_reset_wait_count = 0;
        }
        else if(RDC_RESET_STATE_ACT == g_u1_rdc_reset_wait_status)
        {
            if(PRV_RDC_SPI_WAIT < g_u2_rdc_reset_wait_count)   /* Wait 1000 count
* 50[us] = 50[ms] */
            {
                g_u1_rdc_reset_wait_status = RDC_RESET_STATE_FIN;
            }
        }
        else if(RDC_RESET_STATE_FIN == g_u1_rdc_reset_wait_status)
        {
            R_RSLV_Rdc_Init_Sequence(&rdc_result);
            if (RSLV_MD_OK == rdc_result)
            {
                com_u1_flg_rdc_sequence = FALSE;
                g_u1_flg_rdc_state_ready = TRUE;
                /* Start of IRQ5 */
                R_ICU_Start_irq5();
            }
        }
        else
        {
            ;
        }
    }

    /* RDC SPI main function */
    R_RSLV_Rdc_Communication();

    /* Setting PWM duty of MTU3 channel 7 */
    R_RSLV_Phase_AdjUpdate();
}
```

7.3　メインループ

7.3.1　実装例

以下にメインループの実装例を示します。

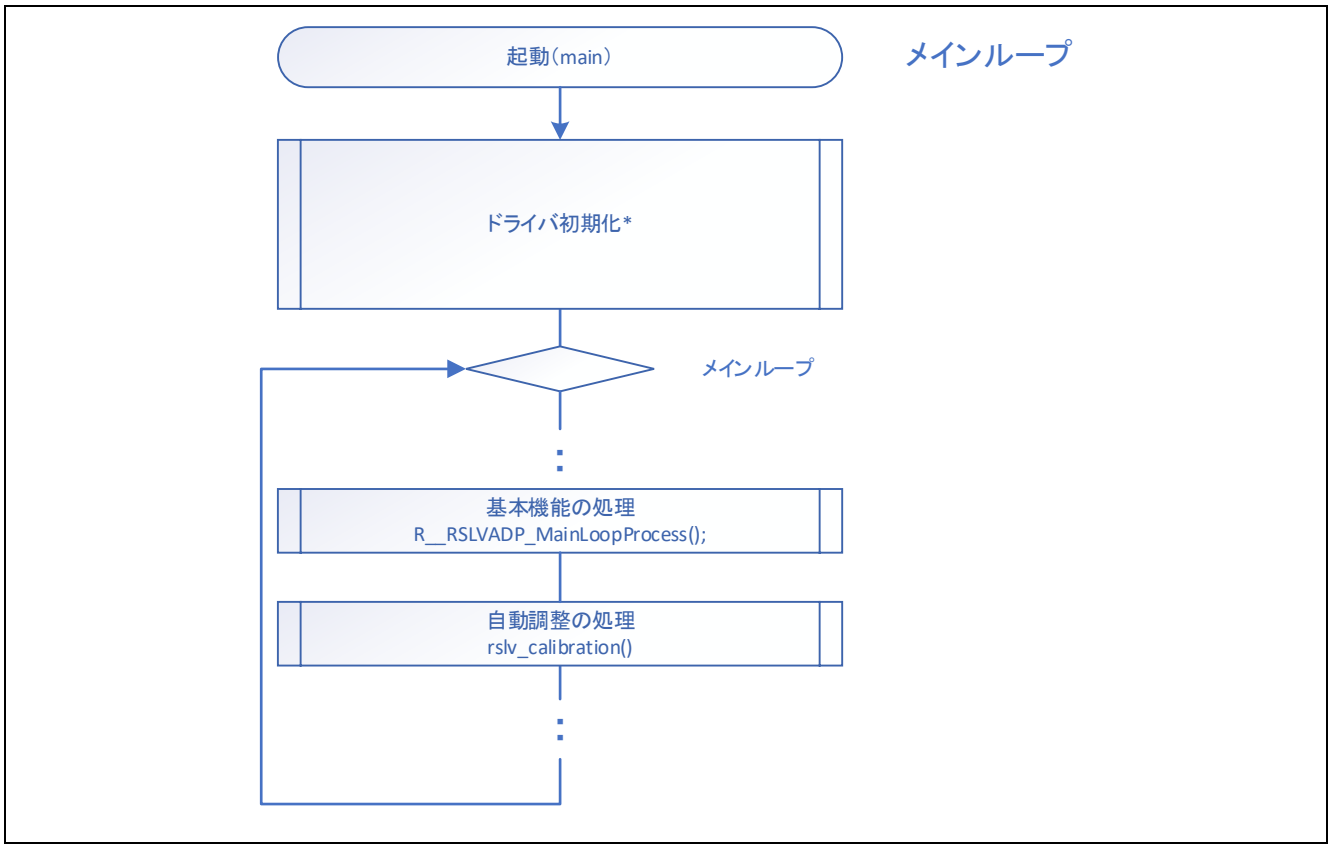

図 7.7　メインループ実装例

メインループでは、RDC-IC 通信処理と位相調整信号 Duty 更新処理を周期的に呼び出してください。また、「7.11　断線検知機能」に示す断線検知処理も実装しておくことを推奨します。本サンプルコードでは、基本機能の処理で RDC-IC 初期設定処理と位相調整信号 Duty 更新処理を実施、自動調整の処理でゲイン・位相自動調整と角度誤差補正信号自動調整処理を実施しています。

【注】 *　ドライバ初期化については、「3.1　ドライバ初期化処理」を参照してください

7.3.2　サンプルコード

下記に、main 関数（メインループ）の関数例を記載します。

```
/***********************************************************************
* Function Name : main
* Description   : Initialization and main routine
* Arguments     : None
* Return Value  : None
***********************************************************************/
void main(void)
{
    float f4_temp;
clrpsw_i();                                     /* Interrupt disabled */
    /* Initialize peripheral functions */
  // MCU 初期化　「7.2.3.1 MCU 初期化（ペリフェラル初期化）」を参照してください
    R_MTR_InitHardware()
    // ドライバ初期化　「7.2.3.2 ドライバ初期化」を参照してください
    R_RSLVADP_Init();

    /* Initialize ICS. */
    ics2_init((void*)dtc_table, ICS_SCI1_PD3_PD5, ICS_INT_LEVEL, ICS_BRR,
    ICS_INT_MODE);

    /* Start of A/D converter */
    R_MTR_Start_s12ad();

    /* Start of CMT0 */
    R_MTR_Start_cmt0();

    /* Initialize private global variables. */
    variables_init();

    /* Execute reset event. */
    R_MTR_SR_Foc_ExecEvent( MTR_ID_A,MTR_EVENT_RESET);

setpsw_i();                                     /* Interrupt enabled */

    /* Start peripheral modules related to the resolver. The following must be
       called after enabling interrupts. */
  // 励磁信号出力開始　「7.4.2 サンプルコード」を参照してください
    R_RSLVADP_Start();
    // ユーザ作成のコールバック関数ポインタ設　「7.2.3.2 ドライバ初期化」を参照してください
    mtr_init_adjst_interface();

    /*** Main routine ***/
    while (1)
    {
        /* User interface */
        ui_main();

        R_MTR_SR_Foc_GetSpeed(MTR_ID_A, &f4_temp, &g_f4_adjst_rslv_speed_rad);
        // 基本機能処理　RDC-IC 通信（RDC-IC 初期設定通信）、位相調整信号 Duty 更新
        R_RSLVADP_MainLoopProcess();
        // 自動調整処理　ゲイン・位相調整、角度誤差補正信号調整処理
        rslv_calibration();
        /* Clear watch dog timer. */
        R_MTR_ClearWdt();
```

```
    }
} /* End of function main */
```

7.4　励磁信号出力

7.4.1　実装例

以下は励磁信号出力に関連する API 関数の実装例を示すブロック図です。

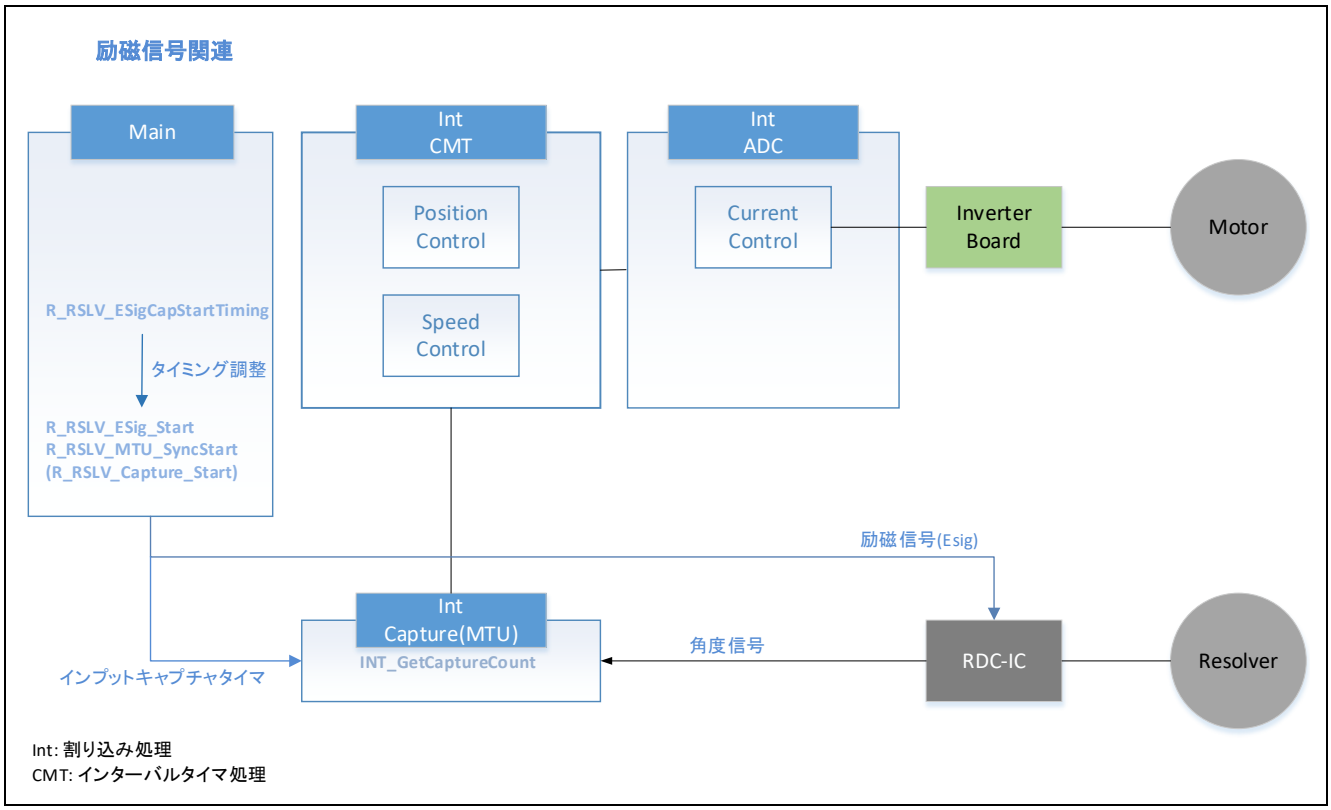

図 7.8　励磁信号出力実装例

励磁信号を出力するには、R_RSLV_ESig_Start（「6.2.18　励磁信号出力開始 API 関数」を参照）を使用します。

7.4.2　サンプルコード

サンプルコードを以下に示します。

以下は、励磁信号と角度検出用タイマを同期スタートする実装例です。

```
/***********************************************************************
* Function Name : R_RSLVADP_Start
* Description   : Resolver start processing
* Arguments     : None
* Return Value  : None
***********************************************************************/
void R_RSLVADP_Start(void)
{
    /* Initialize resolver settings */
    R_RSLV_ESig_Start();
    R_RSLV_MTU_SyncStart(MTU_TCSYSTR_BIT_MTU9 | MTU_TCSYSTR_BIT_MTU2);

    /* Output the angle error correction signal (current default is "TRUE"). */
    if (TRUE == com_u1_flg_csig)
    {
        R_RSLV_CSig_Start(com_u2_csig_shiftnum, com_u2_csig_amplvl);
    }
    else
    {
        R_RSLV_CSig_Stop();
    }
    g_u1_flg_pre_csig = com_u1_flg_csig;
}
```

7.5　位相調整信号出力

7.5.1　実装例

以下は位相調整信号出力用 API の実装例を示すブロック図です。

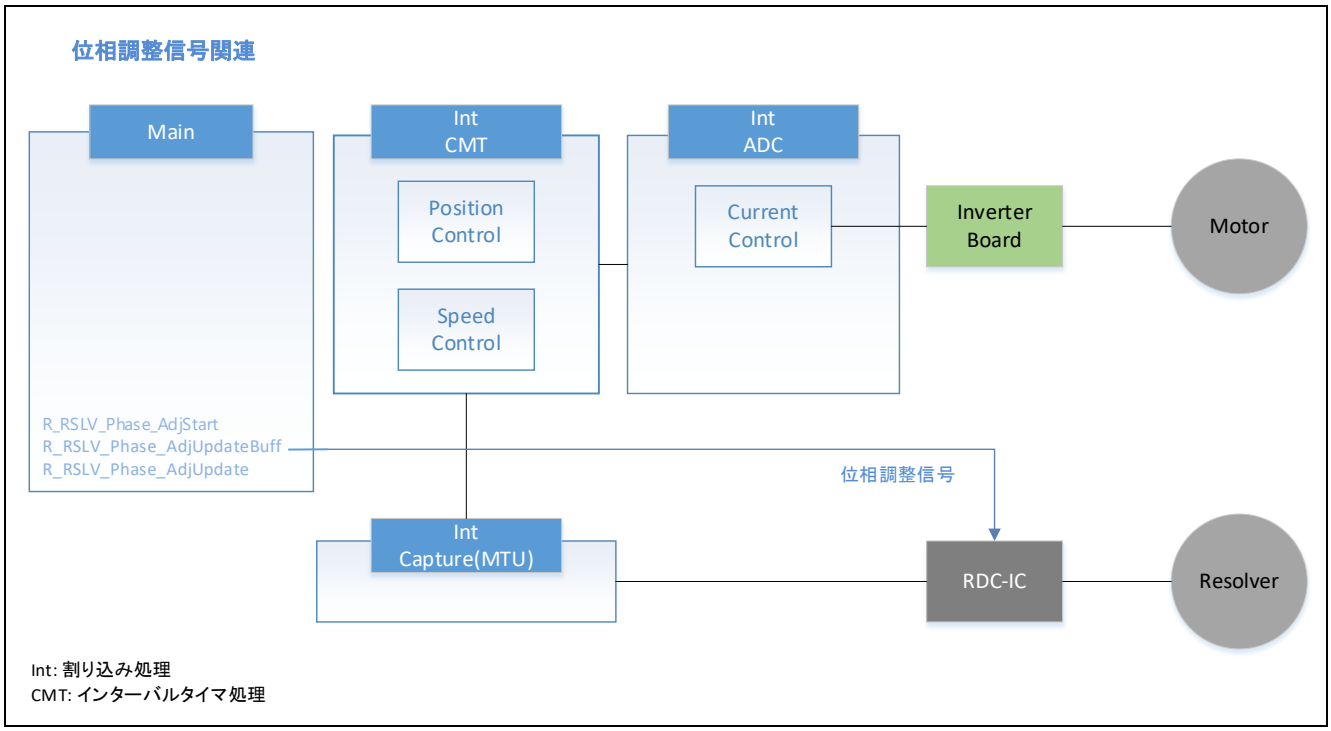

図 7.9　位相調整信号出力実装例

位相調整信号を出力するには、R_RSLV_Phase_AdjStart（「6.2.22　位相調整信号出力開始 API 関数」参照）、R_RSLV_Phase_AdjUpdateBuff（「6.2.24　位相調整信号 Duty バッファ設定 API 関数」参照）、R_RSLV_Phase_AdjUpdate（「6.2.25　位相調整信号 Duty レジスタ設定 API 関数」参照）を使用します。

R_RSLV_Phase_AdjUpdateBuff でドライバ内の Duty 情報を更新後、R_RSLV_Phase_AdjUpdate で Duty 出力用レジスタを設定します。その後、R_RSLV_Phase_AdjStart で PWM 信号を出力します。

7.5.2　サンプルコード

サンプルコードを以下に示します。

7.5.2.1　位相調整信号出力処理

以下では、A 相 Duty：65%、B 相 Duty：22%の設定をメインループ内に実装した例を示します。

```
unsigned char u1_flg_phase_started = 0U;   /* 位相調整信号開始フラグ */

void main(void)
{
    /* 初期化処理 */

    /* メインループ */
    while (1)
    {
        /* RDC-IC 通信処理 */

        /* 位相調整信号処理 */
        if (0U == u1_flg_phase_started)
        {
            R_RSLV_Phase_AdjUpdateBuff(65, PHASE_CH_A);
            R_RSLV_Phase_AdjUpdateBuff(22, PHASE_CH_B);
        }

        R_RSLV_Phase_AdjUpdate();  /* R_RSLV_Phase_AdjUpdate は周期呼び出し */

        if (0U == u1_flg_phase_started)
        {
            R_RSLV_Phase_AdjStart();
            u1_flg_phase_started = 1U;
        }
    }
}
```

7.6　角度誤差補正信号出力

7.6.1　実装例

以下は角度誤差補正信号出力用 API の実装例を示すブロック図です。

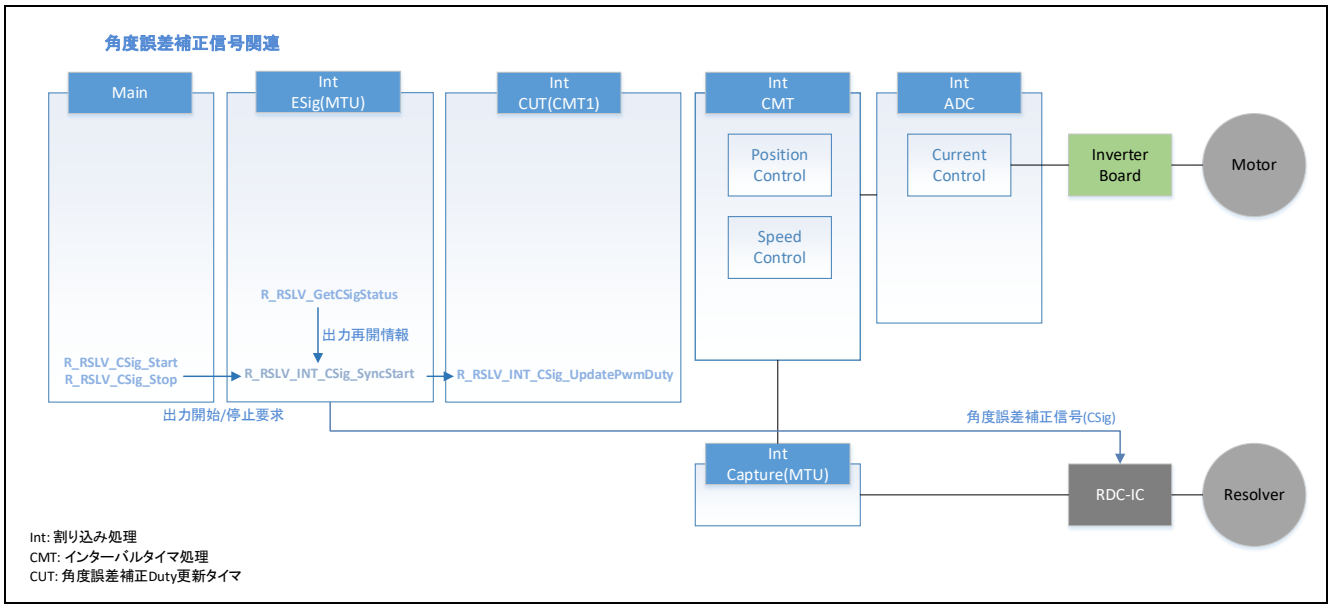

図 7.10　角度誤差補正信号出力実装例

角度誤差補正信号を出力するには、R_RSLV_CSig_Start（「6.2.6　角度誤差補正信号出力開始 API 関数」参照）、R_RSLV_INT_CSig_SyncStart（「6.2.9　角度誤差補正信号同期スタート API 関数」参照）、R_RSLV_INT_CSig_UpdatePwmDuty（「6.2.8　角度誤差補正信号 Duty 更新 API 関数」参照）を使用します。

7.6.2　サンプルコード

サンプルコードを以下に示します。

7.6.2.1　角度誤差補正信号出力開始・停止処理

下記処理をメインループから呼び出すように実装してください。

```
/***********************************************************************
* Function Name : R_RSLVADP_Start
* Description   : Resolver start processing
* Arguments     : None
* Return Value  : None
***********************************************************************/
void R_RSLVADP_Start(void)
{
    /* Initialize resolver settings. */
    R_RSLV_ESig_Start();
    R_RSLV_MTU_SyncStart(MTU_TCSYSTR_BIT_MTU9 | MTU_TCSYSTR_BIT_MTU2);

    /* Output the angle error correction signal (current default is "TRUE"). */
    if (TRUE == com_u1_flg_csig)
    {
        R_RSLV_CSig_Start(com_u2_csig_shiftnum, com_u2_csig_amplvl);
    }
    else
    {
        R_RSLV_CSig_Stop();
    }
    g_u1_flg_pre_csig = com_u1_flg_csig;
}
```

7.6.2.2　PWM Duty 更新割り込み処理

下記 API 関数を角度誤差補信号 Duty 更新タイマ割り込みに実装してください。

```
#pragma interrupt (mtr_csig_interrupt(vect = VECT_RSLV_CSIG))
static void mtr_csig_interrupt(void)
{
setpsw_i();                                    /* Interrupt enabled */
    R_RSLV_INT_CSig_UpdatePwmDuty();
} /* End of function mtr_csig_interrupt */
```

7.6.2.3　角度誤差補正信号同期スタート

下記 API 関数を励磁割り込みに実装してください。

```
#pragma interrupt (rslv_esig_interrupt(vect = VECT_RSLV_ESIG))
static void rslv_esig_interrupt(void)
{
setpsw_i();

    if(mtu9_interrupt_decimation_flag == 0)
    {
        R_RSLV_INT_CSig_SyncStart();
        mtu9_interrupt_decimation_flag ++;
        R_RSLV_INT_ESigCounter();
    }
    else
    {
        mtu9_interrupt_decimation_flag = 0;
    }

} /* End of function rslv_esig_interrupt */
```

7.7　角度信号入力

7.7.1　実装例

以下は角度信号入力用 API の実装例を示すブロック図です。

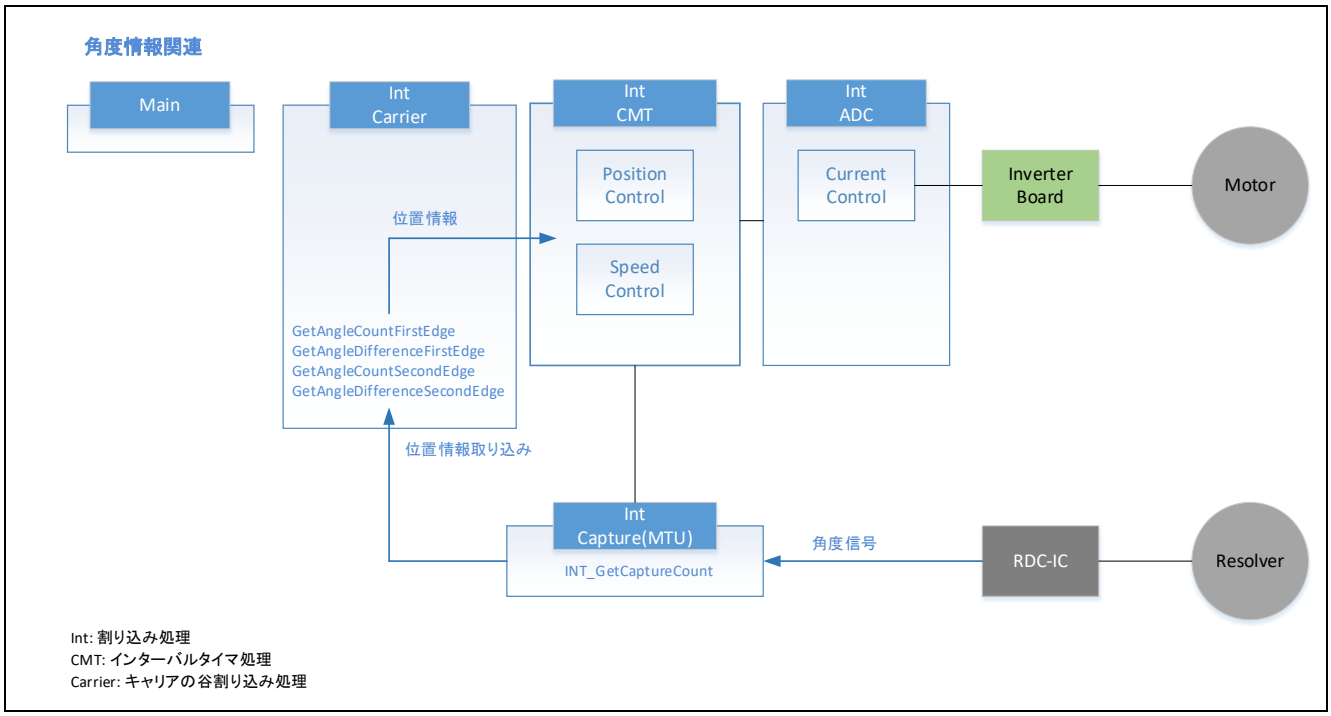

図 7.11　角度信号入力実装例

角度信号（立ち下がりエッジ）のカウンタ、及びカウンタ差分情報を取得する場合は FirstEdge を、角度信号（立ち上がりエッジ）のカウンタ値を取得する場合は SecondEdge を用いてください。

7.7.2　サンプルコード

サンプルコードを以下に示します。

7.7.2.1　角度信号割り込み処理

下記 API 関数をインプットキャプチャ割り込みに実装してください。

```
#pragma interrupt (rslv_capture_interrupt(vect = VECT_RSLV_CAPTURE))
static void rslv_capture_interrupt(void)
{
    R_RSLV_INT_GetCaptureCount();
} /* End of function rslv_capture_interrupt */
```

7.7.2.2　位置情報取り込み処理

下記は、モータ駆動用 PWM タイマの谷割り込みに実装した例を示しています。

```
#pragma interrupt mtr_mtu3_tciv4_interrupt(vect=VECT(MTU4,TCIV4))
static void mtr_mtu3_tciv4_interrupt( void )
{
    uint16_t u2_angle_cnt;
    int16_t  s2_angle_diff;

//初回エッジと次のエッジの両方でカウンタ値を取得する場合
//    uint16_t s2_AngleDiffHi;
//    uint16_t s2_AngleDiffLo;

setpsw_i();                                     /* Interrupt enabled *

    R_RSLV_GetAngleCountFirstEdge(&u2_angle_cnt);
    R_RSLV_GetAngleDifferenceFirstEdge(&s2_angle_diff);
    R_MTR_SR_Foc_SetAngleInfo(MTR_ID_A, u2_angle_cnt, s2_angle_diff);

// 下記は、初回エッジと次のエッジの両方でカウンタ値を取得する場合の処理追加例です
//    /* Get angle count value of resolver. */
//    if(RSLV_HIGH == R_RSLV_GetCaptureEdge())
//    {
//        R_RSLV_GetAngleCountFirstEdge(&g_st_foc.u2_rslv_angle_cnt);
//    }
//    else
//    {
//        R_RSLV_GetAngleCountSecondEdge(&g_st_foc.u2_rslv_angle_cnt);
//    }
//    R_RSLV_GetAngleDifferenceFirstEdge(&s2_AngleDiffHi);
//    R_RSLV_GetAngleDifferenceSecondEdge(&s2_AngleDiffLo);
//    g_st_foc.s2_angle_err_cnt = u2_AngleDiffHi + u2_AngleDiffLo;
//    g_st_foc.s2_angle_err_cnt *= 0.5f;
//

    R_RSLVADP_IncreaseWaitTimer();


} /* End of function mtr_mtu3_tciv4_interrupt */
```

7.8　ゲイン&位相自動調整機能

7.8.1　実装例

以下はゲイン＆位相自動調整機能用 API の実装例を示すブロック図です。

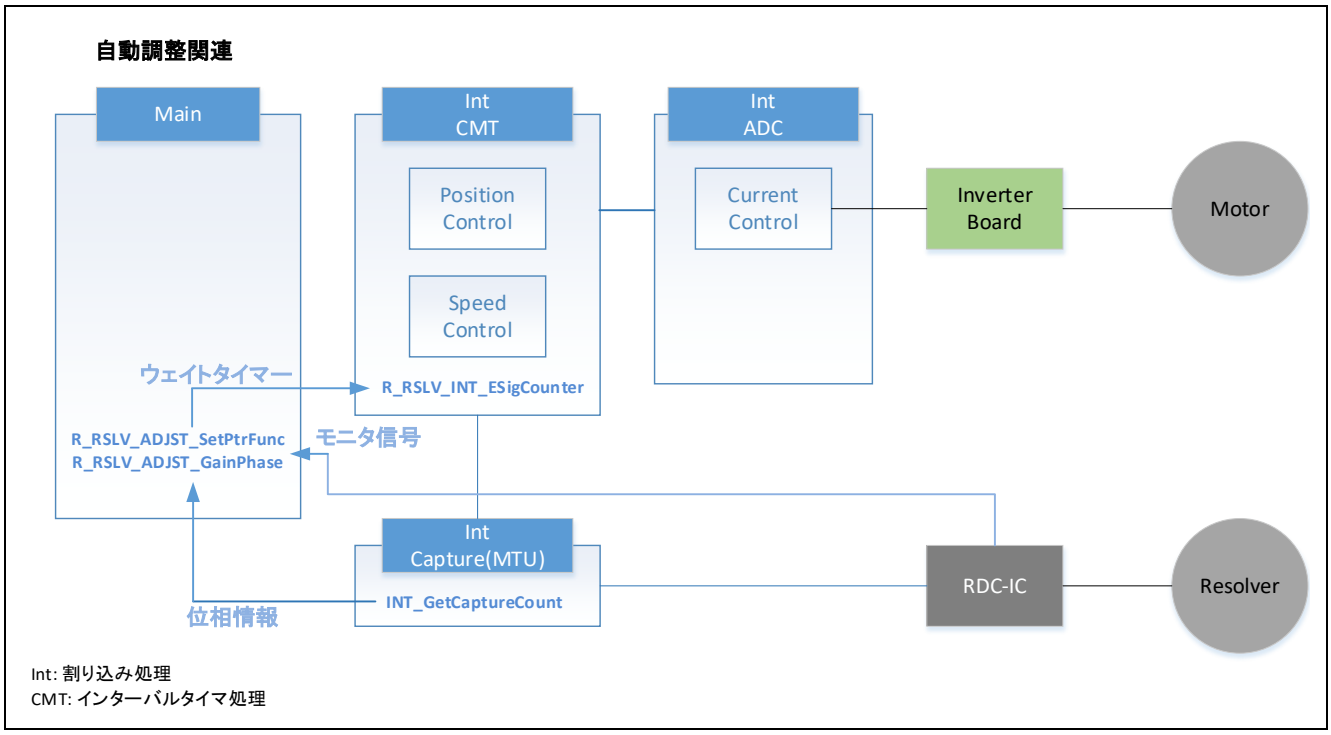

図 7.12　ゲイン＆位相自動調整機能実装例

ゲイン＆位相自動調整を実行するには R_RSLV_ADJST_SetPtrFunc（「6.2.42　ユーザ作成のコールバック関数ポインタ設定 API 関数」参照）、R_RSLV_ADJST_GainPhase（「6.2.40　レゾルバ信号ゲインおよび位相調整 API 関数」参照）、R_RSLV_INT_ESigCounter（「6.2.21　待機時間確認カウンタ API 関数」参照）を使用します。

R_RSLV_INT_GetCaptureCount（「6.2.12　角度検出値取り込み API 関数」参照）は、位相調整時に位相情報を取得するために使用します。角度検出割り込みに実装してください。

R_RSLV_INT_ESigCounter は、調整処理内でのウェイトタイマとして使用します。励磁信号割り込みに実装してください。

7.8.2　ゲイン＆位相調整詳細

以下にゲイン＆位相調整機能の実装例を示します。

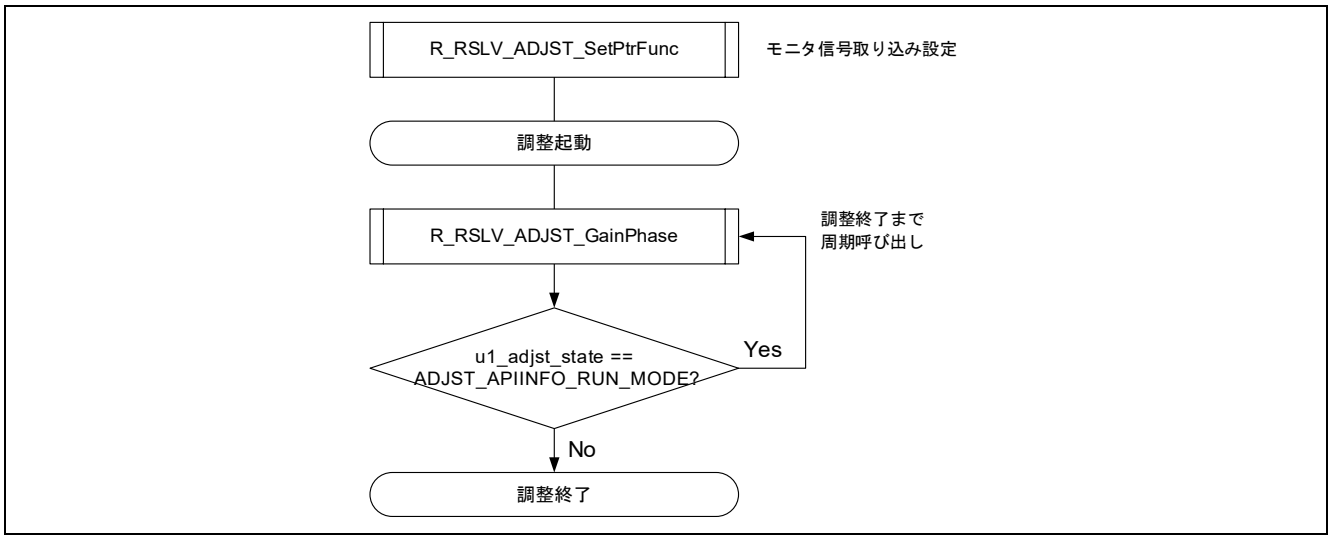

図 7.13　ゲイン＆位相調整シーケンス

ゲイン＆位相調整機能は、RDC-IC が出力するモニタ信号を ADC で変換して取得します。そのためモニタ信号を割り当てた AD チャネル情報を、コールバック関数設定用 API を用いてドライバに設定する必要があります。詳細は「6.3.6　R_RSLV_ADJST_SetPtrFunc API 関数の構造体」を参照してください。

レゾルバ信号ゲインおよび位相調整 API 関数 R_RSLV_ADJST_GainPhase は、調整を終了するまで呼び出してください。

7.8.2.1　調整開始

調整を開始する場合、R_RSLV_ADJST_GainPhase の引数に ADJST_USRREQ_RUN(0)を設定して呼び出します。

7.8.2.2　調整継続

R_RSLV_ADJST_GainPhase の戻り値構造体 st_adjst_gainphase_return_t のメンバーu1_adjst_state が ADJST_APIINFO_RUN_MODE(0U)であれば調整中です。R_RSLV_ADJST_GainPhase の引数に ADJST_USRREQ_RUN(0)を設定して、継続して呼び出してください。

調整を中断したい場合は引数を ADJST_USRREQ_STOP(1)にして呼び出してください。

但し、中断から通常状態への復帰処理を実行する必要があるため、上記同様戻り値 u1_adjst_state が ADJST_APIINFO_END_USER_STOP(13U)になるまで R_RSLV_ADJST_GainPhase を継続して呼び出してください。

7.8.2.3　調整終了判断

u1_adjst_state が ADJST_APIINFO_RUN_MODE(0U)以外になれば調整は終了です。R_RSLV_ADJST_GainPhase の呼び出しを止めてください。

終了状況は u1_adjst_state に格納されます。正常終了（ADJST_APIINFO_END_NORMAL(1U)）した場合は、戻り値構造体 st_adjst_gainphase_return_t のメンバーに調整結果を反映します。

調整処理内で調整結果に合わせて、必要情報が適宜変更されているため、API による再設定等は不要です。

戻り値構造体 st_adjst_gainphase_return_t のメンバー内容を以下に示します。詳細は「表 6-6 R_RSLV_ADJST_GainPhase API 関数の構造体定義（1/2）」を参照してください。

表 7-1　st_ptr_func_arg_t 構造体メンバー

メンバー名	型	内容
u1_adjst_state	unsigned char	ゲイン＆位相調整処理状況、及び処理終了状況
u1_res_dlcgsl	unsigned char	RDC-IC レジスタ DLCGSL 調整結果値 (A 相ゲイン調整結果値)
u2_res_a_duty	unsigned short	A 相位相調整信号調整結果 Duty 値
u2_res_b_duty	unsigned short	B 相位相調整信号調整結果 Duty 値

7.8.3　サンプルコード

サンプルコードを以下に示します。

7.8.3.1　ゲイン＆位相調整 API 呼び出し部

下記処理をメインループから周期的に呼び出すように実装してください。

```
/***********************************************************************
* Function Name: mtr_rdc_AdjstGainPhaseProcess
* Description  : Process for adjustment of RDC gain & phase parameters
* Arguments    : req -
*                  Request of sequence continuation (0:Continue, 1:Halt)
* Return Value : Active status of process (1:Active, 0:Finished)
***********************************************************************/
uint8_t mtr_rdc_AdjstGainPhaseProcess( uint8_t req )
{
    uint8_t result = TRUE;

    /* Call gain & phase adjustment API function. */
    gp_api_ret = R_RSLV_ADJST_GainPhase(req);

    /* 戻り値で処理分岐 */
    /* 動作中は、動作が継続されていることを通知 */
    switch (gp_api_ret.u1_adjst_state)
    {
        default:
        case ADJST_APIINFO_RUN_MODE:
            {
                result = TRUE;
            }
        break;

        case ADJST_APIINFO_END_NORMAL:
        case ADJST_APIINFO_ERR_GAIN_HI_LMT:
        case ADJST_APIINFO_ERR_GAIN_LO_LMT:
        case ADJST_APIINFO_ERR_GAIN_SWAY:
        case ADJST_APIINFO_ERR_PHASE_AHI_BLO:
        case ADJST_APIINFO_ERR_PHASE_ALO_BHI:
        case ADJST_APIINFO_ERR_PHASE_SWAY:
        case ADJST_APIINFO_ERR_MOTOR:
        case ADJST_APIINFO_END_USER_STOP:
            {
                result = FALSE;
            }
        break;
    }

    return (result);
}
```

7.9　角度誤差補正信号自動調整機能

7.9.1　実装例

以下は角度誤差補正信号自動調整機能用 API の実装例を示すブロック図です。

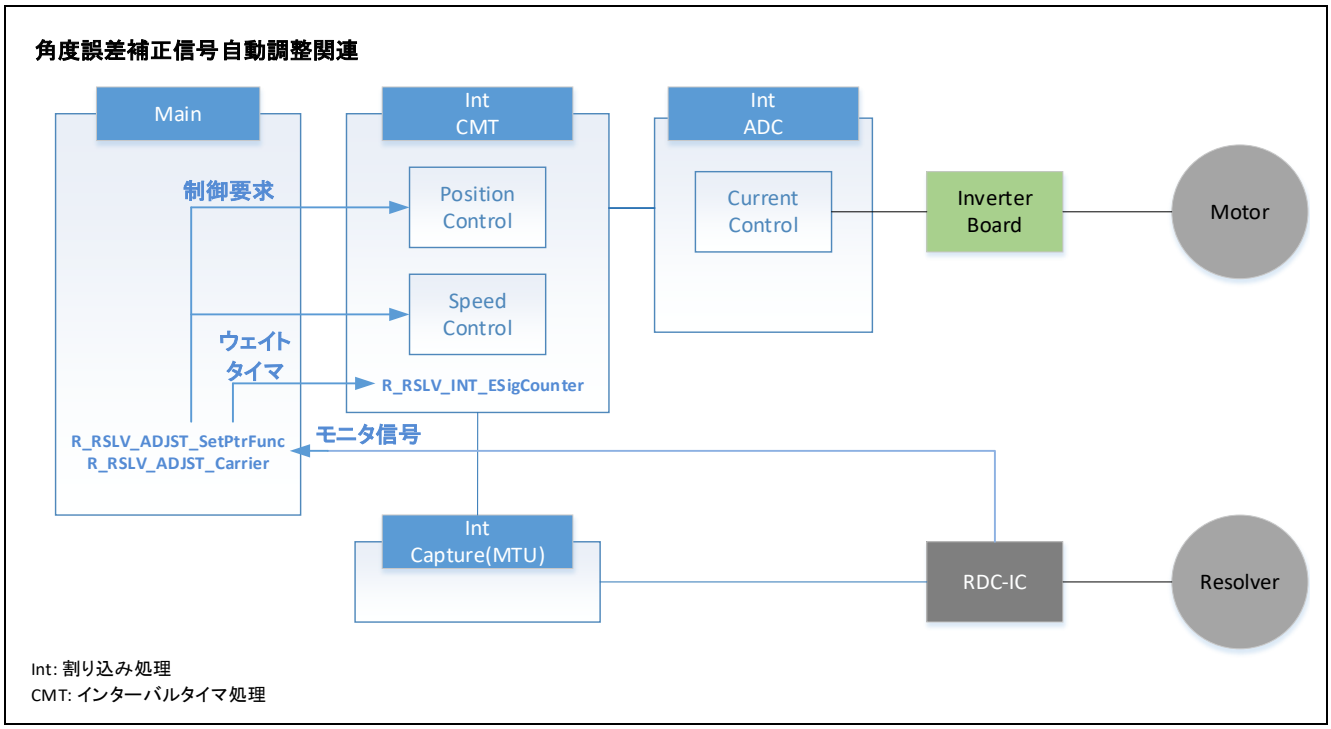

図 7.14　角度誤差補正信号自動調整機能実装例

角度誤差補正信号自動調整を実行する場合は、R_RSLV_ADJST_Carrier（「6.2.41　角度誤差補正信号調整 API 関数」参照）を使用します。

R_RSLV_INT_ESigCounter()の機能内容は「7.8　ゲイン&位相自動調整機能」時と変わりません。同様の実装をしてください。

7.9.2　角度誤差補正信号調整詳細

角度誤差補正信号調整は、調整中にモータを制御する必要があります。

以下に角度誤差補正信号調整の実装例を示します。

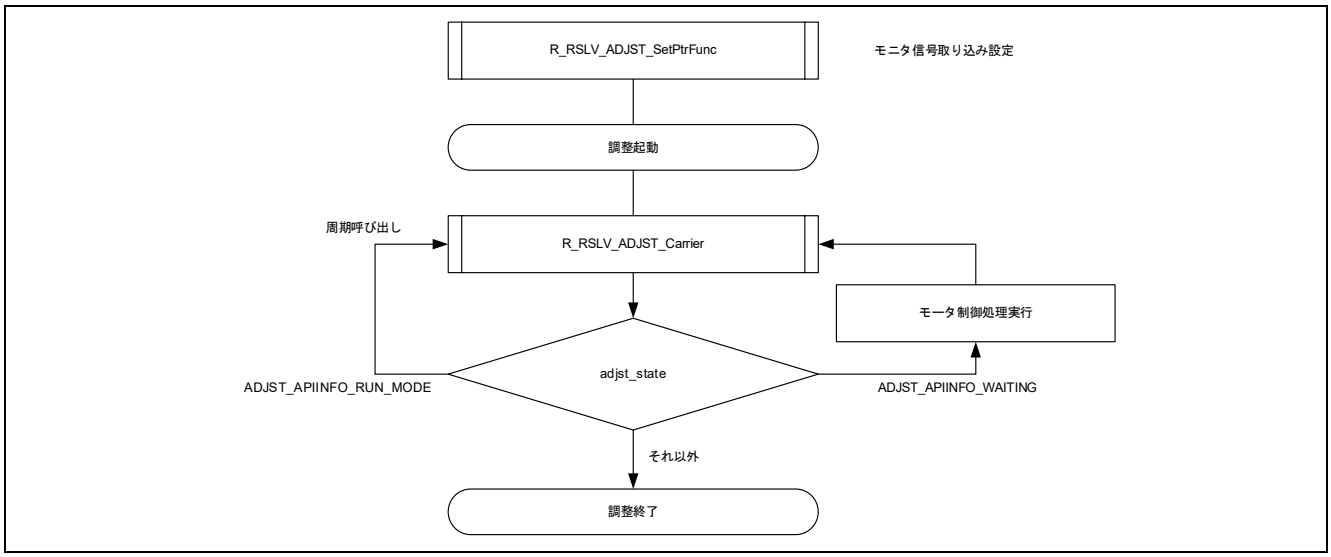

図 7.15　角度誤差補正信号調整シーケンス

起動までの流れは「7.8　ゲイン&位相自動調整機能」と変わりません。adjst_state の値で実行する処理が異なります。本調整機能がモータ制御を要求する場合、戻り値は ADJST_APIINFO_WAITING になります。

調整起動から調整終了までの呼び出し側（アプリケーション）とドライバ間のシーケンス図を以下に示します。

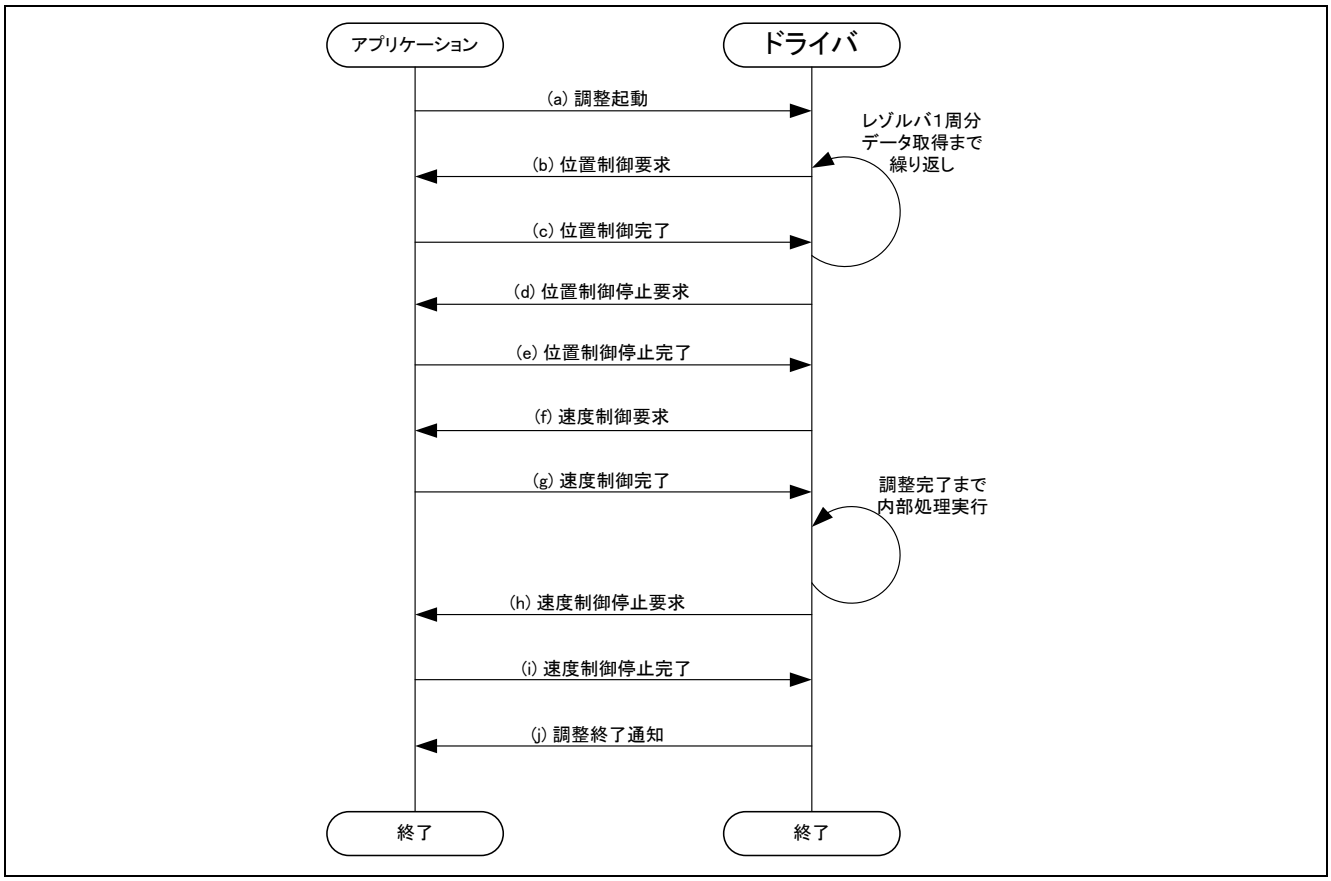

図 7.16　角度誤差補正信号調整シーケンス

以下に各シーケンス(a)～(j)での処理内容等に関して説明します。

(a)　調整起動

調整を開始する場合、R_RSLV_ADJST_Carrier の引数構造体 st_adjst_carrier_arg_t のメンバーcall_state に ADJST_USRREQ_RUN(0)を設定して呼び出します。詳細は「表 6-7　R_RSLV_ADJST_Carrier API 関数の構造体定義」を参照してください。

(b)　位置制御要求

調整を開始すると、R_RSLV_ADJST_Carrier から位置制御の要求が来ます。要求は R_RSLV_ADJST_Carrier の戻り値構造体 st_adjst_carrier_return_t のメンバーadjst_state と req_mtr_ctrl で通知されます。

adjst_state = ADJST_APIINFO_WAITING(2)
req_mtr_ctrl = ADJST_APIREQ_POS_CTRL(1)
mtr_ctrl_data = 0（最初はレゾルバ角 0 度から開始）

本調整処理は上記のようにモータ制御の設定を戻り値として要求するので、本設定に従い位置制御を開始してください。

またモータ制御を設定中に再度 R_RSLV_ADJST_Carrier を呼び出す際には、引数構造体メンバー req_state に ADJST_USRINFO_PROCESSING(1)を設定してください。ドライバ側にユーザアプリケーション側が設定中であることを示すことができます。

(c)　位置制御完了

位置制御要求に従い位置制御（要求された指定角度への位置制御）が完了したら、引数構造体メンバー req_state に ADJST_USRINFO_COMPLETE(0)を設定してください。

この後ドライバ側がデータ取得を開始し、データ取得を終えると再度位置制御を要求します。この時に要求位置情報 mtr_ctrl_data が更新されているので、これに従い再度位置制御を行ってください。ドライバが必要なデータ取得を完了するまで位置制御要求⇒位置制御完了の処理を繰り返します。レゾルバ角 1 周分のデータ取得を終了すると位置制御停止要求に進みます。

(d)　位置制御停止要求

全てのデータ取得を終了すると、R_RSLV_ADJST_Carrier から位置制御停止の要求が来ます。

adjst_state = ADJST_APIINFO_WAITING(2)
req_mtr_ctrl = ADJST_APIREQ_POS_STOP(2)

上記のように API の戻り値が更新されたら、位置制御を停止してください。位置制御停止処理中、位置制御要求処理と同様に R_RSLV_ADJST_Carrier の呼び出す際には、引数構造体メンバーreq_state に ADJST_USRINFO_PROCESSING(1)を設定してください。

(e)　位置制御停止完了

位置制御の停止が完了したら引数構造体メンバーreq_state に ADJST_USRINFO_COMPLETE(0)を設定してください。引き続き、速度制御要求処理へ移行します。

(f)　速度制御要求

R_RSLV_ADJST_Carrier から速度制御の要求が来ます。

adjst_state = ADJST_APIINFO_WAITING(2)
req_mtr_ctrl = ADJST_APIREQ_SPD_CTRL(3)
mtr_ctrl_data = 1000[rpm]

上記のように API の戻り値が更新されたら、速度制御を開始してください。

(g)　速度制御完了

指定速度に速度制御が達したら、完了通知として R_RSLV_ADJST_Carrier の引数構造体メンバーreq_state に ADJST_USRINFO_COMPLETE(0)を設定してください。

速度制御動作が実行されると、調整処理が角度誤差補正信号の調整パラメータを操作して調整を実行します。調整処理が終了するまで R_RSLV_ADJST_Carrier を継続して呼び出してください。調整が終了すると速度制御停止要求処理へ進みます。

(h)　速度制御停止要求

調整を終了すると、R_RSLV_ADJST_Carrier が速度制御停止を要求します。

adjst_state = ADJST_APIINFO_WAITING(2)
req_mtr_ctrl = ADJST_APIREQ_SPD_STOP(4)

上記のように API の戻り値が更新されたら、速度制御を停止してください。

(i)　速度制御停止完了

速度制御の停止を完了したら、R_RSLV_ADJST_Carrier の引数構造体メンバーreq_state に ADJST_USRINFO_COMPLETE(0)を設定してください。引き続き、調整終了通知処理へ移行します。

(j)　調整終了通知

全ての調整処理を終了すると、R_RSLV_ADJST_Carrier から調整終了の通知が来ます。

adjst_state が ADJST_APIINFO_RUN_MODE(0)、もしくは ADJST_APIINFO_WAITING(2)以外になれば調整終了です。

各戻り値の内容は「表 6-7　R_RSLV_ADJST_Carrier API 関数の構造体定義」を参照してください。

戻り値が ADJST_APIINFO_END_NORMAL(1)の場合は、調整は正常に終了しています。その場合、戻り値構造体メンバーres_XXXX には調整された値が戻ります。

調整処理内で必要情報へは反映されていますので、API による再設定等は不要です。

7.9.3　フィルタ回路による遅れ位相

角度誤差補正信号調整で、補正回路に入力する角度誤差補正信号の位相シフト量を正しく調整するためには、RDC-IC 周辺回路に実装されているフィルタによる遅れ位相を考慮する必要があります。

フィルタ回路を含む、角度誤差補正信号周辺の回路構成図を以下に示します。

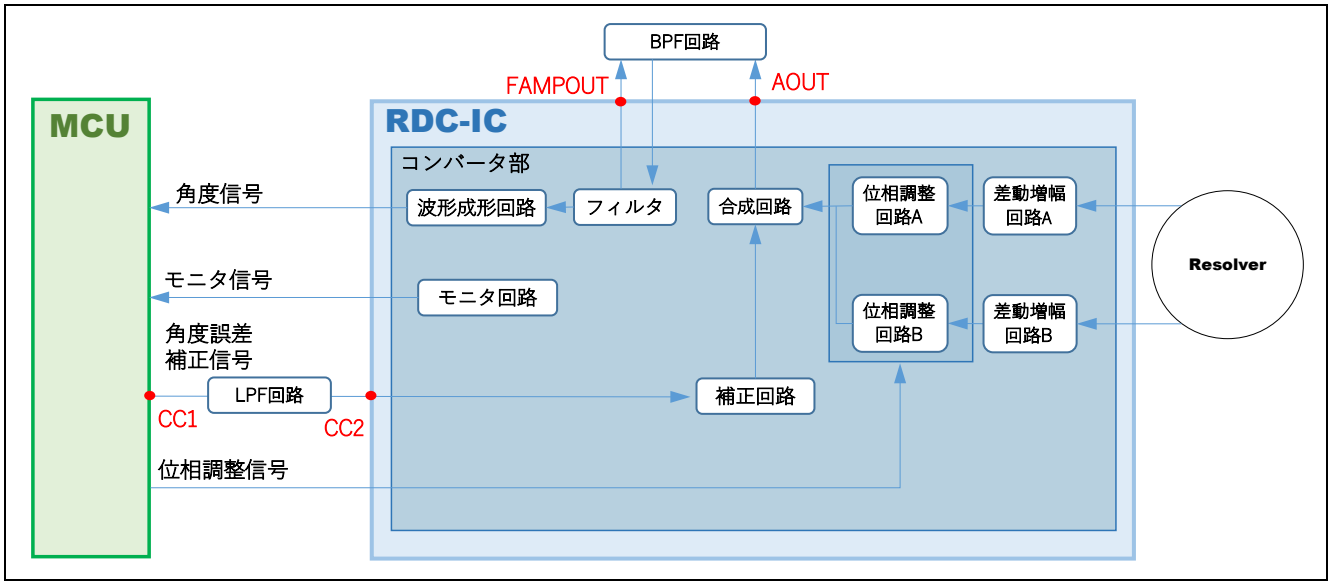

図 7.17　角度誤差補正信号周辺回路構成図

以下に各フィルタ回路による遅れ位相に関して説明します。

(a)　バンドパスフィルタ回路による遅れ位相

図 7.17 中の AOUT、FAMPOUT 間にある BPF 回路によって、位相の遅れが生じます。

(b)　角度誤差補正信号用 LPF 回路による遅れ位相

図 7.17 中の CC1、CC2 間にある LPF 回路によって、位相の遅れが生じます。

(a), (b)の値は、R_RSLV_SetSystemInfo でシステム情報の指定を行った際に、励磁信号の周波数に応じて初期化されます。初期値は、周辺部品選定ガイドに記載の各部品の定数例によって算出された遅れ位相の値が設定されます。定数例の詳細に関しては、「1.4　関連資料」に記載の「レゾルバ信号変換 IC の周辺部品選定ガイド」を参照してください。

ユーザが遅れ位相の値を初期値から変更する場合は、R_RSLV_ADJST_SetFilterDelay を使用します。第一引数に(a)の値、第二引数に(b)の値を設定してください。また、R_RSLV_SetSystemInfo をコールした後に本 API をコールしてください。

7.9.4　サンプルコード

サンプルコードを以下に示します。

7.9.4.1　周期呼び出し処理

下記処理をメインループから呼び出すように実装してください。

```
/***********************************************************************
* Function Name: r_mtr_rdc_AdjstCarrierProcess
* Description  : Process for adjustment of angle error correction signal
* Arguments    : req -
*                  Request of sequence continuation (0:Continue, 1:Halt)
* Return Value : Active status of process (1:Active, 0:Finished)
***********************************************************************/
static uint8_t r_mtr_rdc_AdjstCarrierProcess( uint8_t req )
{
    uint8_t result = TRUE;

    cc_api_req.call_state = req;

    /* Call angle error adjustment API function. */
    cc_api_ret = R_RSLV_ADJST_Carrier (cc_api_req);

    /* 戻り値で制御変更 */
    switch (cc_api_ret.adjst_state)
    {
        default:
        case ADJST_APIINFO_RUN_MODE:
            {
                result = TRUE;       /* 実行継続を通知 */
            }
        break;

        /* Application of motor control is required. */
        case ADJST_APIINFO_WAITING:
            {
                /* モータ制御処理実行 */
                r_mtr_ctrl_posspd_for_ccadjust_seq();
            }
        break;

        case ADJST_APIINFO_END_NORMAL:
        case ADJST_APIINFO_ERR_CARRIER:
        case ADJST_APIINFO_ERR_MOTOR:
        case ADJST_APIINFO_END_USER_STOP:
            {
                result = FALSE;      /* 実行終了を通知 */
            }
        break;
    }

    return (result);

}
```

7.10 RDC-IC 通信機能

7.10.1 実装例

以下は RDC-IC 通信機能 API の実装例を示すブロック図です。

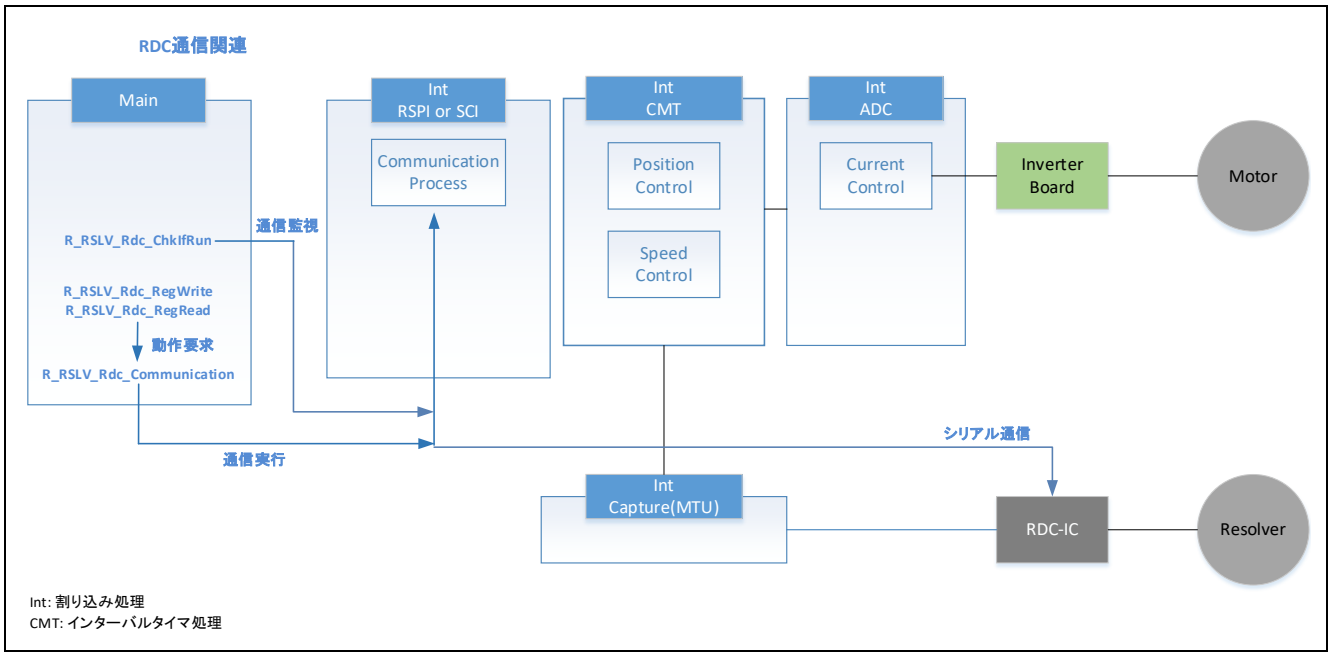

図 7.18　RDC-IC 通信処理実装例

RDC-IC 通信は、RSPI もしくは SCI を用いて実行します。どちらのペリフェラルを使用した場合も使用する API 関数は変わりません。通信処理は R_RSLV_Rdc_Communication（「6.2.29　RDC-IC 通信処理 API 関数」）で実行します。この処理は順次処理となるので周期的に呼び出してください。読み込みを実行する場合は、R_RSLV_Rdc_RegRead（「6.2.31　RDC-IC 各レジスタの読み込み API 関数」）を使用してください。書き込みを実行する場合は、R_RSLV_Rdc_RegWrite（「6.2.30　RDC-IC 各レジスタへの書き込み API 関数」）を使用してください。R_RSLV_Rdc_ChkIfRun（「6.2.32　RDC-IC レジスタアクセス状態取得 API 関数」）は、現在の通信状態を返します。なお、実行中は書き込み／読み込みの要求をしないでください。

通信割り込み処理の実装は、RSPI の場合 SC 出力コードを使用します。SCI の場合 SC が 16 ビット通信フォーマットに対応していないため、SC 出力の送信割り込み処理を 16 ビット対応の送信割り込みに対応する必要があります。「7.10.2.3　SCI を使用した例」を参考に実装してください。

7.10.2 サンプルコード

サンプルコードを以下に示します。

7.10.2.1 RDC-IC レジスタへの書き込み

以下に、RDC-IC レジスタ書き込み機能の実装例を示します。

```
main( void )
{
   while (1U)
   {
     if (TRUE == flg_write_req)
     {
         /* RDC-IC レジスタのバッファにデータを書き込み */
         R_RSLV_Rdc_SetRegisterVal(rdc_write_data, rdc_address);
         /* 書き込み要求を発行 */
         R_RSLV_Rdc_RegWrite(&rdc_write_status);
         flg_write_req = FALSE;
     }
     /* RDC-IC との通信シーケンス */
     R_RSLV_Rdc_Communication();
   }
}
```

7.10.2.2 RDC-IC レジスタ値の読み込み

以下に、RDC-IC レジスタ読み込み機能の実装例を示します。

```
main( void )
{
   while (1U)
   {
     if (TRUE == flg_read_req)
     {
         /* RDC-IC レジスタ値をバッファに読み込み */
         R_RSLV_Rdc_RegRead(rdc_address);
         flg_read_req = FALSE;
     }
     /* RDC-IC との通信シーケンス */
     R_RSLV_Rdc_Communication();

     /* RDC-IC レジスタのバッファ値の取り込み */
     R_RSLV_Rdc_GetRegisterVal(&rdc_read_data, rdc_address);
   }
}
```

7.10.2.3 SCI を使用した例

SCI を使用する場合、SC により自動出力した送信割り込み処理を、16 ビット対応にする必要があります。下記例では、SC 出力コードとは別に 16 ビット対応送信割り込み関数を作成して、SC 出力コードの送信割り込み処理に実装しています。

```
/* 送信割り込み処理 （SC 生成コード）*/
#pragma interrupt r_Config_SCI0_transmit_interrupt(vect=VECT(SCI0, TXI0))
static void r_Config_SCI0_transmit_interrupt(void)
{
    // 下記処理を削除して、16 ビット対応 送信割り込み処理関数を Call してください
//    if (0U < g_sci0_tx_)
//    {
//        SCI0.TD count R = *gp_sci0_tx_address;
//        gp_sci0_tx_address++;
//        g_sci0_tx_count--;
//    }
//    else
//    {
//        SCI0.SCR.BIT.TIE = 0U;
//        SCI0.SCR.BIT.TEIE = 1U;
//    }
    R_SCI0_Trans_Intr_Process();      // 追加：16 ビット対応 送信割り込み処理の項を参照してください
}

/* 受信割り込み処理 （SC 生成コード）*/
#pragma interrupt r_Config_SCI0_receive_interrupt(vect=VECT(SCI0, RXI0))
static void r_Config_SCI0_receive_interrupt(void)
{
    if (g_sci0_rx_length > g_sci0_rx_count)
    {
        *gp_sci0_rx_address = SCI0.RDR;
        gp_sci0_rx_address++;
        g_sci0_rx_count++;

        if (g_sci0_rx_length == g_sci0_rx_count)
        {
            SCI0.SCR.BIT.RIE = 0;

              /* Set the CS port to the high level. */
              PORT9.PODR.BIT.B2 = 1U;     // チップセレクト：Chip INACTIVE（追加が必要です）

            /* Clear the TE and RE bits. */
            if((0U == SCI0.SCR.BIT.TIE) && (0U == SCI0.SCR.BIT.TEIE))
            {
                SCI0.SCR.BYTE &= 0xCFU;
                R_Config_SCI0_Stop();        // SCI モジュールストップ（追加が必要です）
            }

            r_Config_SCI0_callback_receiveend();
        }
        else
        {
            R_SCI0_Trans_Intr_Process();    // 次のデータ受信のため（追加が必要です）
        }
    }
}
```

```
/* 受信エラー割り込み処理（SC 生成コード）*/
#pragma interrupt r_Config_SCI0_receiveerror_interrupt(vect=VECT(SCI0, ERI0))
void r_Config_SCI0_receiveerror_interrupt(void)
{
    uint8_t err_type;

    r_Config_SCI0_callback_receiveerror();

    /* Clear the overrun error flag. */
    err_type = SCI0.SSR.BYTE;
    err_type &= 0xDFU;
    err_type |= 0xC0U;
    SCI0.SSR.BYTE = err_type;
}
```

以下は、RSPI 使用時も同じように作成してください

```
/* 送信完了コールバック処理（SC 生成コード）*/
void r_Config_SCI1_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI1_callback_transmitend. Do not edit
       comment generated here. */
    R_RSLV_Rdc_CallComEndCb();     // 通信終了コールバック API 関数を追加してください
    /* End user code. Do not edit comment generated here. */
}


/* 受信完了コールバック処理（SC 生成コード）*/
void r_Config_SCI1_callback_receiveend(void)
{
    /* Start user code for r_Config_SCI1_callback_receiveend. Do not edit
       comment generated here. */
    R_RSLV_Rdc_CallComEndCb();     // 通信終了コールバック API 関数を追加してください
    /* End user code. Do not edit comment generated here. */
}


/* 受信エラーコールバック処理（SC 生成コード）*/
void r_Config_SCI1_callback_receiveerror(void)
{
    /* Start user code for r_Config_SCI1_callback_receiveerror. Do not edit
       comment generated here. */
    R_RSLV_Rdc_CallErrorCb();       // 受信エラーコールバック API 関数を追加してください
    /* End user code. Do not edit comment generated here. */
}
```

```
/* 16 ビット対応 送信割り込み処理関数（ユーザ作成コードとなります） */
static void R_SCI0_Trans_Intr_Process(void)
{
    uint16_t com_data;

    if (0U == s_u1_pass_flg)
    {
        if (g_sci0_tx_count > 0U)
        {
            /* Determine whether to send the upper data or lower data */
            if (g_sci0_tx_count & 0x01)
            {
                com_data = *gp_sci0_tx_address & 0x00FF;
            }
            else
            {
                com_data = *gp_sci0_tx_address & 0xFF00;
                com_data >>= 8;
                s_u1_pass_flg = 1U;
            }
            /* Write data for transmission. */
            SCI0.TDR = com_data;
            g_sci0_tx_count--;
        }
        else
        {
            SCI0.SCR.BIT.TIE = 0U;
            SCI0.SCR.BIT.TEIE = 0U;
        }
    }
    else
    {
        s_u1_pass_flg = 0U;
    }
```

7.11 断線検知機能

7.11.1 実装例

以下は断線機能 API の実装例を示すブロック図です。

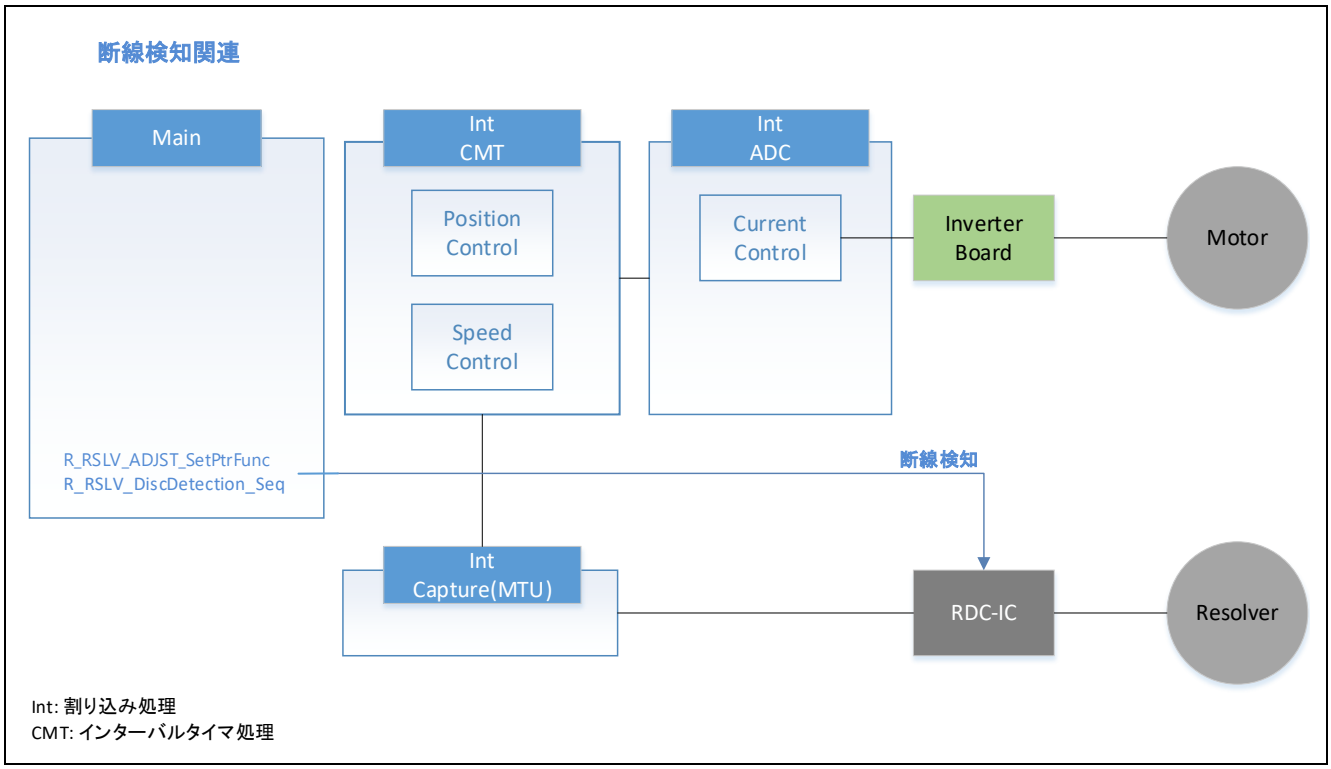

図 7.19　断線検知処理実装例

断線検知機能を実行するには、R_RSLV_ADJST_SetPtrFunc（「6.2.42　ユーザ作成のコールバック関数ポインタ設定 API 関数」）、R_RSLV_DiscDetection_Seq（「6.2.45　断線検知処理 API 関数」）を使用します。断線検知処理は順次処理を実行するため、断線検知処理 API 関数を周期的に呼び出してください。

ユーザ作成のコールバック関数ポインタ設定 API の使用方法は「7.8　ゲイン&位相自動調整機能」を参照してください。

以下に断線検知処理の実装例を示します。

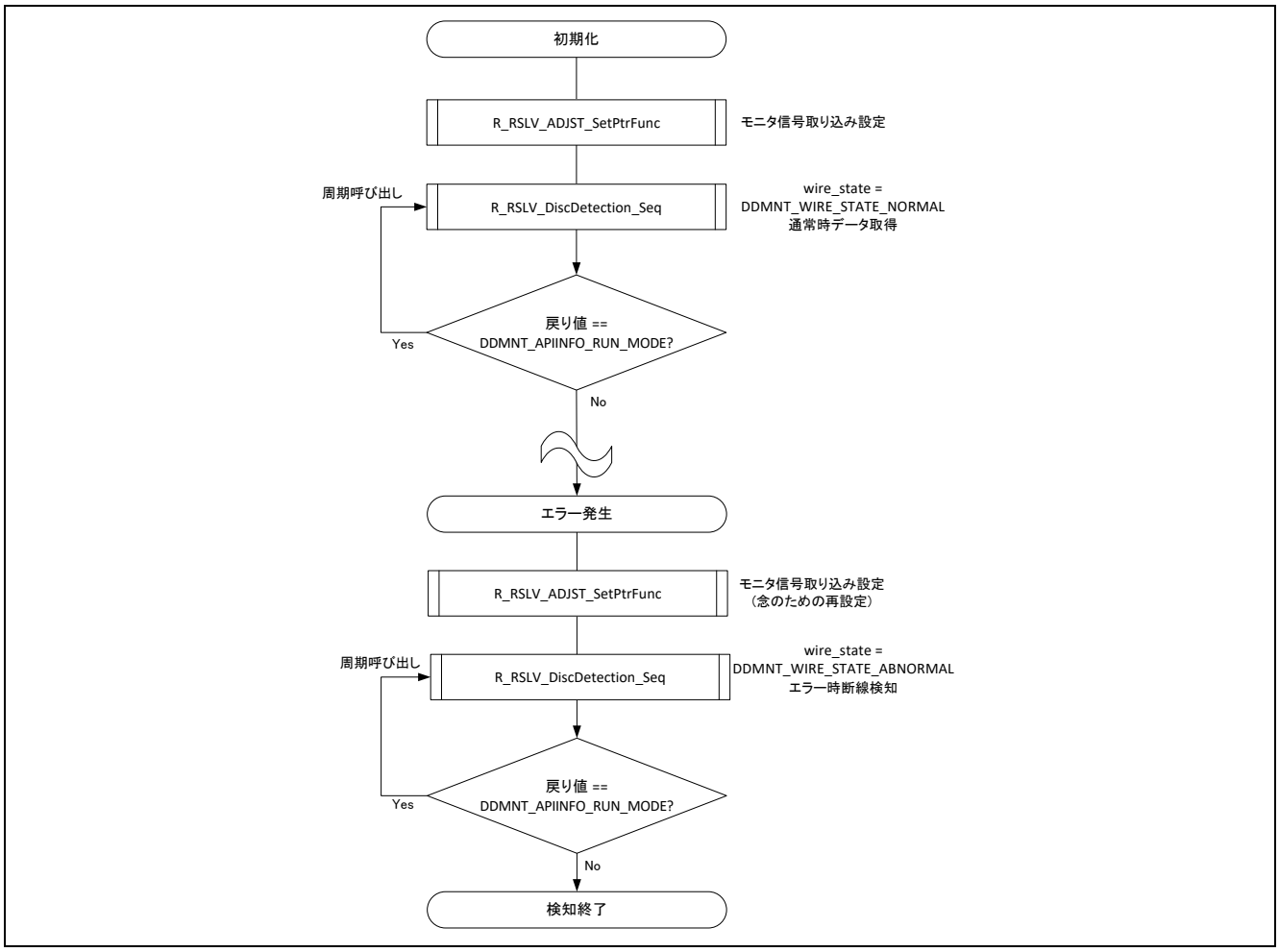

図 7.20　断線検知シーケンス例

断線検知処理は、通常の接続状態とエラー発生時の接続状態を比較して、レゾルバ信号接続線の断線状態を判定します。従って、通常の接続状態のデータを取得する必要があります。

通常時のデータを取得する場合、断線検知処理 API 関数 R_RSLV_DiscDetection_Seq の引数構造体 st_rdc_ddmnt_arg_t のメンバーwire_state に DDMNT_WIRE_STATE_NORMAL(0U)を設定して実行します。詳細は「6.3.7　R_RSLV_DiscDetection_Seq API 関数の構造体」を参照してください。API の戻り値が DDMNT_APIINFO_RUN_MODE（断線検知実行中）以外になれば通常状態のデータ取得は終了です。

この通常状態取得処理は RDC-IC ドライバを用いた各種初期化設定後、通常動作開始前に実行してください。

何らかの形でモータ動作にエラーが発生した場合（例：速度制御時に位置情報が更新されなくなった）、その原因がレゾルバ信号の断線に因るものなのかを判断するために、本断線検知処理を使用します。従って断線検知処理はユーザで必要時（エラー発生時）に動作させてください。

断線状態を判定するためには、R_RSLV_DiscDetection_Seq を引数

arg_value.call_state = DDMNT_USRREQ_RUN
arg_value.wire_state = DDMNT_WIRE_STATE_ABNORMAL

に設定して呼び出します。API の戻り値が DDMNT_APIINFO_RUN_MODE（断線検知実行中）以外になれば処理終了です。

初期化処理、エラー時断線判断処理いずれの場合も、R_RSLV_DiscDetection_Seq を引数 arg_value.call_state = DDMNT_USRREQ_STOP に設定して呼び出せば、処理を中断することができます。

7.11.2 サンプルコード

サンプルコードを以下に示します。

7.11.2.1 断線検知処理

以下に、断線検知処理の実装例を示します。r_mtr_DetectDisconnect_Seq()はメインループから呼び出されています。

```
/***********************************************************************
* Function Name: r_mtr_DetectDisconnect_Seq
* Description  : Sequence to detect resolver disconnection
* Arguments    : None
* Return Value : None
***********************************************************************/
/* 断線検知実装ステートマシン */
static void r_mtr_DetectDisconnect_Seq( void )
{
    st_rdc_ddmnt_arg_t temp_arg;                        /* API 引数用テンポラリ変数 */
    unsigned char dd_ret = DDMNT_APIINFO_RUN_MODE;   /* 戻り値受け取り用変数 */

    /* 非動作時は常に停止で要求 */
    temp_arg.call_state = DDMNT_USRREQ_STOP;

    switch (s_u1_sts_ddcnct)
    {
        case STS_DDCNCT_NONE:
        default:
            /* Do nothing. */
        break;

        /* 初期化開始 */
        case STS_DDCNCT_INIT_START:
            {
                /* Set interface functions */
                /*この関数内で R_RSLV_ADJST_SetPtrFunc を呼んでいます */
                r_mtr_init_ddiscnct_interface();
                SetDdiscnctStatus(STS_DDCNCT_INIT);   /* ステート設定用マクロ */
            }
        break;

        /* 初期化終了待ち周期呼び出し */
        case STS_DDCNCT_INIT:
            {
                temp_arg.call_state = DDMNT_USRREQ_RUN;
                temp_arg.wire_state = DDMNT_WIRE_STATE_NORMAL;
                dd_ret = R_RSLV_DiscDetection_Seq(temp_arg);

                /* 戻り値が DDMNT_APIINFO_RUN_MODE 以外になれば終了 */
                if (DDMNT_APIINFO_RUN_MODE != dd_ret)
                {
                    SetDdiscnctStatus(STS_DDCNCT_INIT_FIN);
                }
            }
        break;

        /* 初期化終了後処理 */
    case STS_DDCNCT_INIT_FIN:
```

```
            {
                /* Set interface functions for adjustment. */
                r_mtr_init_adjst_interface();
                /* All system initialization finished. */
                s_u1_flg_system_init_fin = TRUE;
                SetDdiscnctStatus(STS_DDCNCT_NONE);
            }
        break;

        /* エラー発生時断線検知開始 */
        case STS_DDCNCT_CONF_START:
            {
                /* SetPtrFunc を再設定 */
                r_mtr_init_ddiscnct_interface();
                SetDdiscnctStatus(STS_DDCNCT_CONF);
            }
        break;

        /* 断線検知待ち周期呼び出し */
        case STS_DDCNCT_CONF:
            {
                temp_arg.call_state = DDMNT_USRREQ_RUN;
                temp_arg.wire_state = DDMNT_WIRE_STATE_ABNORMAL;
                dd_ret = R_RSLV_DiscDetection_Seq(temp_arg);

                /* 通常終了ならそのまま終了 */
                if (DDMNT_APIINFO_END_NORMAL == dd_ret)
                {
                    SetDdiscnctStatus(STS_DDCNCT_CONF_FIN);
                }
                /* 断線が検知されたらその情報を変数に設定 */
                else if (DDMNT_APIINFO_ERR_DISCONNECT == dd_ret)
                {
                    g_u2_err_status |= MTR_ERR_RSLV_DISCNCT;
                    SetDdiscnctStatus(STS_DDCNCT_CONF_FIN);
                }
                /* それ以外は周期呼び出し */
                else
                {
                    /* Do nothing. */
                }
            }
        break;

        /* 検知後処理 */
        case STS_DDCNCT_CONF_FIN:
                /* Set interface functions for adjustment again. */
                r_mtr_init_adjst_interface();
                SetDdiscnctStatus(STS_DDCNCT_NONE);
        break;
    }
} /* End of function r_mtr_DetectDisconnect_Seq() */
```

7.12 ALARM 解除機能

7.12.1 実装例

以下は ALARM 解除機能 API の実装例を示すブロック図です。

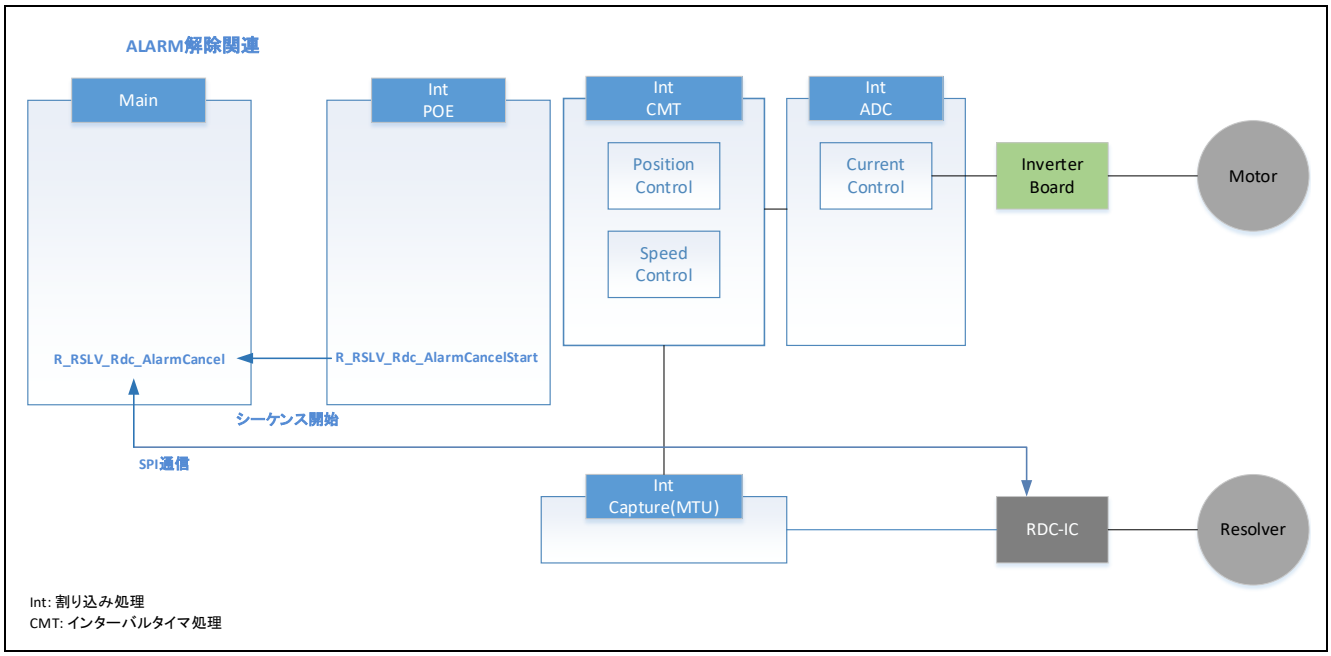

図 7.21　ALARM 解除処理実装例

RDC-IC が過熱状態を検知すると ALARM 信号端子を Low レベルにします。一般的には ALARM 信号を POE 端子に接続し、強制遮断機能を用いてモータを停止してください。

RDC-IC の ALARM 状態を解除するためには、R_RSLV_Rdc_AlarmCancelStart（「6.2.38　RDC-IC ALARM 解除開始 API 関数」参照）を実行してドライバのステータスを解除状態に変更し、R_RSLV_Rdc_AlarmCancel（「6.2.39　RDC-IC ALARM 解除シーケンス制御 API 関数」参照）を実行します。

ALARM 解除 API 関数 R_RSLV_Rdc_AlarmCancel は、順次処理を実行するために内部にステートマシン構造を持っているので、周期的に呼び出してください。R_RSLV_Rdc_AlarmCancel は通常 RSLV_MD_BUSY1 を返します。解除を成功すると RSLV_MD_OK を、解除不可（ALARM 発生状態継続）の場合 RSLV_MD_ERROR を返します。

7.12.2 サンプルコード

サンプルコードを以下に示します。

7.12.2.1 R_RSLV_Rdc_AlarmCancelStart

ALARM 発生後はいつ呼び出しても問題ありません。以下は、ALARM 信号による POE 発生時の POE 割り込み（POE1）内に実装した例を示しています。

```
#pragma interrupt r_mtr_rslv_foc_poe3_oei1_intr_example (vect=VECT(POE,OEI1))
void r_mtr_rslv_foc_poe3_oei1_intr_example( void )
{
    /* POE 事後処理 */
    R_POE3_Stop();

    /* Start the alarm cancellation sequence. */
    R_RSLV_Rdc_AlarmCancelStart();
}
```

7.12.2.2 R_RSLV_Rdc_AlarmCancel

以下に、メインループ内で周期的に呼び出されるように実装した例を示します。

```
main( void )
{
  while (1)
  {
     unsigned char ret;

     ret = R_RSLV_Rdc_AlarmCancel();

     if (RSLV_MD_OK == ret)
     {
        /* 解除成功時処理 */
     }
     else if (RSLV_MD_ERROR == ret)
     {
       /* 解除不可時処理 */
     }
  }
}
```

8.　Rev1.20 以前から Rev2.10 への移行方法

下記に RX24T 版レゾルバドライバ Rev1.20 以前から Rev2.10 へ移行する際の手順を示します。移行例で使用するサンプルコードは、RX24T_MRSSK_STM_RSLV_FOC_CSP_RV120（以降、STM 版サンプルコードと称します）とします。

8.1　フォルダ・ファイル構成変更

Rev2.10 へ移行するためには、レゾルバドライバライブラリとヘッダーファイルの差し替え、SC で生成したペリフェラル用のコード追加が必要です。

8.1.1　ライブラリとヘッダーファイルの差し替えと SC コードの追加

下記 rdc_driver_RX 配下のレゾルバドライバライブラリとヘッダーファイルを差し替えてください。また、src フォルダを作成して、¥smc_gen フォルダをコピーしてください。プロジェクトへの登録は「8.1.2 プロジェクトへの登録」を参照してください。

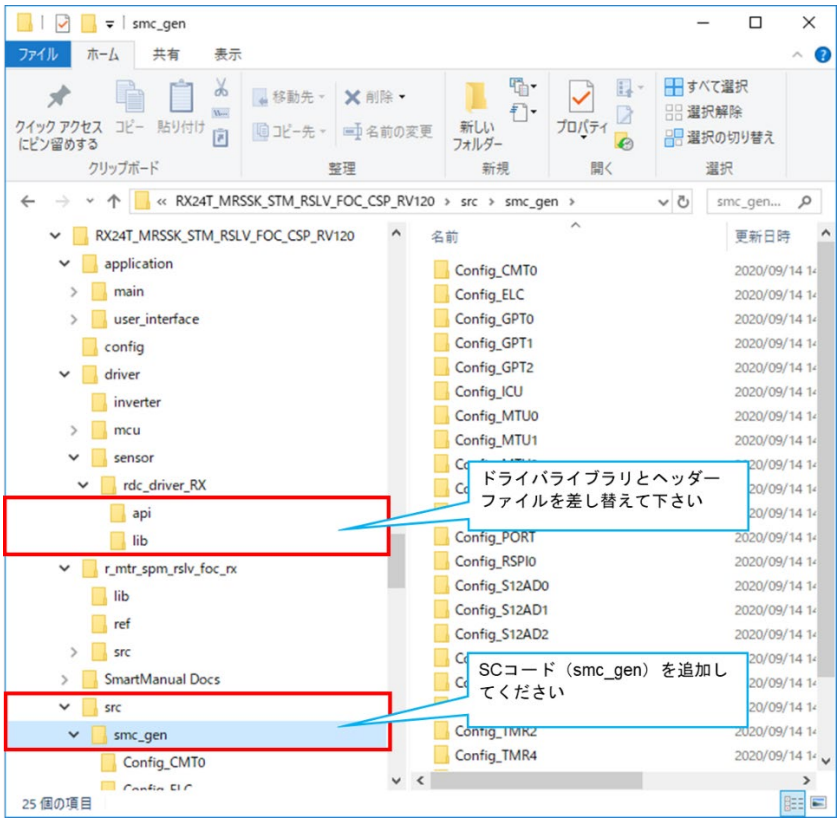

図 8.1　ファイル差し替え・SC コード追加

SC でコードを生成すると、以下のフォルダ（SC コード）が作成されます。

```
¥smc_gen¥
        ¥Config_(peri_func)
        ¥general
        ¥r_bsp
        ¥r_config
        ¥r_pincfg
```

本移行例では、¥r_bsp、¥r_config、¥r_pincfg を使いません。¥Config_(peri_func)と¥general のみプロジェクトに登録するようにしてください。また一部の SC コードを修正する必要があります。SC コードの修正内容は、「8.2.2　SC コードの修正」を参照してください。

8.1.2　プロジェクトへの登録

ファイル差し替えと SC コードの追加を終了したら、IDE のプロジェクトへ各ファイルを登録します。下記の様に登録してください。

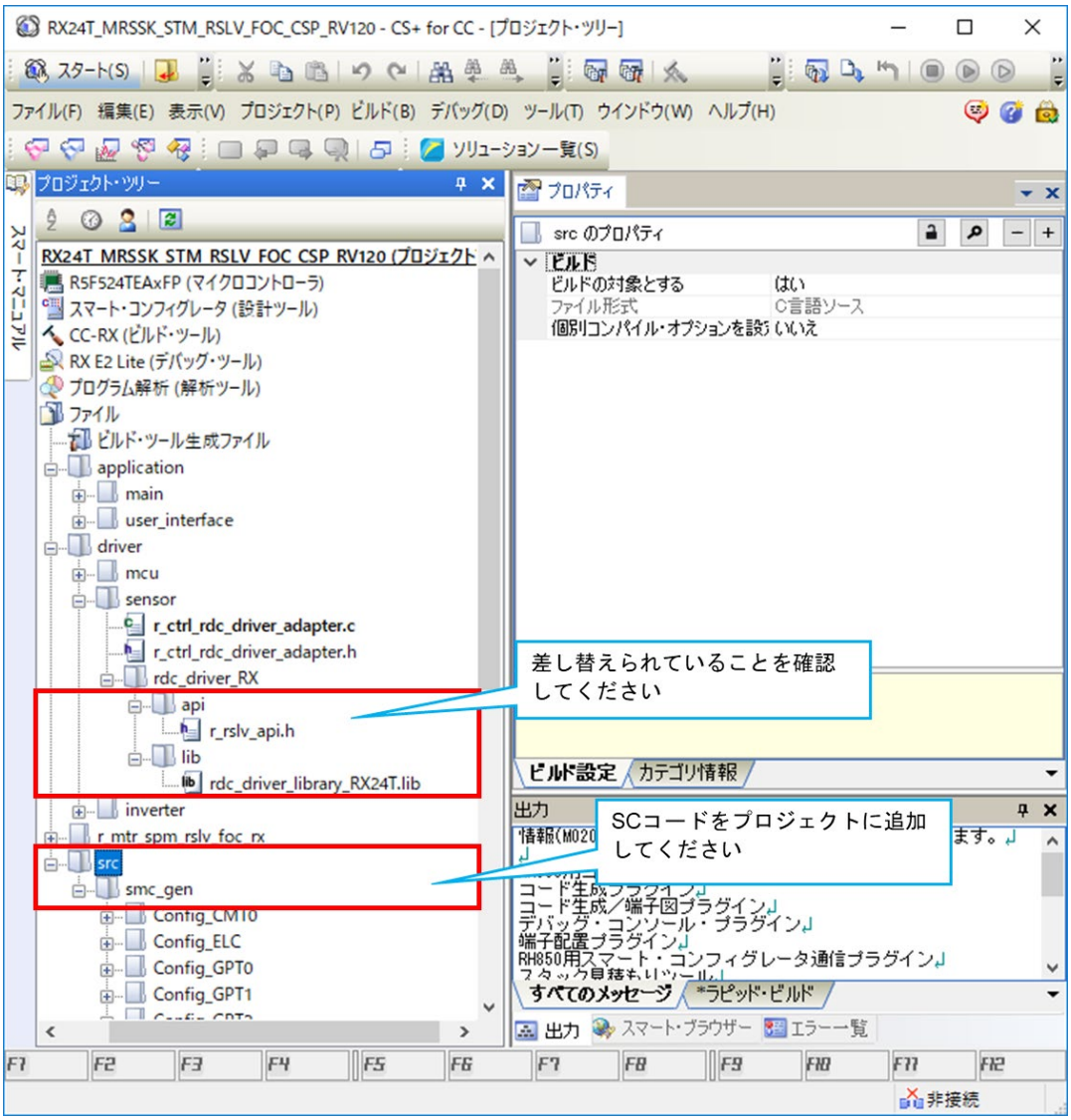

図 8.2　プロジェクトへの登録

8.2　ソースコード修正

8.2.1　ペリフェラル初期化処理

SC で生成したペリフェラル初期化関数を、R_MTR_InitHardware に追加してください。

SC でコードを生成すると R_Systeminit()から初期化関数が呼び出されますが、今回はモータ制御部の初期化関数を使用する例として説明します。

```
void R_MTR_InitHardware (void)
{

    /*=========================*/
    /*     Initialize port     */
    /*=========================*/
    mtr_init_port();

    /*=========================*/
    /*     Initialize clock    */
    /*=========================*/
    mtr_init_clock();

    /*=========================*/
    /*     Initialize WDT      */
    /*=========================*/
    mtr_init_wdt();

    /*=========================*/
    /*     Initialize CMT0     */
    /*=========================*/
    mtr_init_cmt0();
        .
        .
    SYSTEM.PRCR.WORD = 0xA50FU;
    /* Enable writing to MPC pin function control registers */
    MPC.PWPR.BIT.B0WI = 0U;
    MPC.PWPR.BIT.PFSWE = 1U;

    R_Config_MTU9_Esig12_Create();
    R_Config_MTU0_Csig_Create ();
    R_Config_MTU2_Cap_Create ();
    R_Config_CMT1_CsigUpdTim_Create ();
    R_Config_TMR0_PhaseA_Create ();
    R_Config_TMR4_PhaseB_Create ();
    R_Config_TMR3_RdcClk_Create ();
    R_Config_RSPI0_RdcCom_Create ();

    /* Disable writing to MPC pin function control registers */
    MPC.PWPR.BIT.PFSWE = 0U;
    MPC.PWPR.BIT.B0WI = 1U;
    /* Enable protection */
    SYSTEM.PRCR.WORD = 0xA500U;
        .
        .
```

8.2.2　SC コードの修正

プロジェクトに登録後、下記の様に SC コードを修正して下さい。

8.2.2.1　r_cg_userdefine.h へのインクルード宣言追加

r_cg_userdefine.h

```
2   /* DISCLAIMER...
19
21  * File Name    : r_cg_userdefine.h...
27
28  #ifndef CG_USER_DEF_H
29  #define CG_USER_DEF_H
30
32  Includes...
34    /* Start user code for include. Do not edit comment generated here */
35    #include <stdint.h>
36    #include "iodefine.h"
37    #include "r_rslv_api.h"
38    /* End user code. Do not edit comment generated here */
39
41  Macro definitions (Register bit)...
43  /* Start user code for register. Do not edit comment generated here */
44    /* End user code. Do not edit comment generated here */
```

8.2.2.2　各 Config_(peri_func)_user.c へのユーザ作成コード追加

「5.4　関数テーブル設定」に従って、関数テーブル用のユーザ作成コードを追記してください。また、Config_(peri_func).h にプロトタイプ宣言を追加してください。下記は MTU0 に角度誤差補正信号を割り当てた場合の例です。

Config_MTU0_user.c

```
40    /* End user code. Do not edit comment generated here */
41    #include "r_cg_userdefine.h"
42
44  Global variables and functions...
46  /* Start user code for global. Do not edit comment generated here */
47    /* End user code. Do not edit comment generated here */
48
50  * Function Name: R_Config_MTU0_Create_UserInit...
55
56  void R_Config_MTU0_Create_UserInit(void)
57  {
58      /* Start user code for user init. Do not edit comment generated here */
59      /* End user code. Do not edit comment generated here */
60  }
61
62    /* Start user code for adding. Do not edit comment generated here */
63  void R_MTU0_GetTcnt (unsigned short *tcnt)
64  {
65      *tcnt = MTU0.TCNT;
66  }
67
68    /* The function to set the count value. */
69  void R_MTU0_SetTcnt (unsigned short tcnt)
70  {
71      MTU0.TCNT = tcnt;
72  }
73
74    /* The function to get the duty value. */
75  void R_MTU0_GetDuty (unsigned short *duty)
76  {
77      *duty = MTU0.TGRA;
78  }
79
80    /* The function to set the duty value. */
81  void R_MTU0_SetDuty (unsigned short duty)
82  {
83      MTU0.TGRA = duty;
84  }
85    /* End user code. Do not edit comment generated here */
```

8.2.2.3　SC コード生成による割り込み処理への移行前プロジェクト各割り込み処理移植

SC で割り込み設定を有効にすると、割り込み処理関数が自動で作成されます。移行前に作成した割り込み処理は、SC で生成された割り込み処理に移してください。その後、移行前の割り込み処理を削除してください（SC コード作成の割り込み関数を有効にします）。

- 励磁信号割り込み
- 角度誤差補正信号 Duty 更新割り込み
- 角度検出値取り込み割り込み

コーディング例）角度誤差補正信号 Duty 更新割り込み

（移行前）

```
r_mtr_interrupt.c
/***********************************************************************
* Function Name : mtr_csig_interrupt
* Description   : CMI1 interrupt(Duty update of PWM for angle error correction)
* Arguments     : None
* Return Value  : None
***********************************************************************/
#pragma interrupt (mtr_csig_interrupt(vect = VECT_RSLV_CSIG))
static void mtr_csig_interrupt(void)
{
setpsw_i();                                    /* Interrupt enabled */
    R_RSLV_INT_CSig_UpdatePwmDuty();
} /* End of function mtr_csig_interrupt */

Config_CMT0.c：
#pragma interrupt r_Config_CMT0_cmi0_interrupt(vect=VECT(CMT0,CMI0))
static void r_Config_CMT0_cmi0_interrupt(void)
{
    /* Start user code for r_Config_CMT0_ cmi0_interrupt. Do not edit comment
       generated here. */
    /* End user code. Do not edit comment generated here. */
}
```

（移行後）

```
r_mtr_interrupt.c
    関数ごと削除してください
/***********************************************************************
* Function Name : mtr_csig_interrupt
* Description   : CMI1 interrupt(Duty update of PWM for angle error correction)
* Arguments     : None
* Return Value  : None
***********************************************************************/
// #pragma interrupt (mtr_csig_interrupt(vect = VECT_RSLV_CSIG))
// static void mtr_csig_interrupt(void)
// {
// setpsw_i();                                    /* Interrupt enabled */
//    R_RSLV_INT_CSig_UpdatePwmDuty();  // 削除
//} /* End of function mtr_csig_interrupt */

Config_CMT0.c：
#pragma interrupt r_Config_CMT0_cmi0_interrupt(vect=VECT(CMT0,CMI0))
static void r_Config_CMT0_cmi0_interrupt(void)
{
    /* Start user code for r_Config_CMT0_ cmi0_interrupt. Do not edit comment
```

```
    generated here. */
    R_RSLV_INT_CSig_UpdatePwmDuty();   // 追加
    /* End user code. Do not edit comment generated here. */
}
```

8.2.2.4　通信ペリフェラル（RSPI、SCI）ヘッダーファイルの修正

SC にて作成された通信ペリフェラル用ヘッダーファイルのプロトタイプ宣言には、uint16_t など定義が記載されているため、“r_cg_userdefine.h”のインクルード追加が必要となります。

Config_RSPI0.h

```
/* DISCLAIMER */
/* File Name    : Config_RSPI0.h */
#ifndef CFG_Config_RSPI0_H
#define CFG_Config_RSPI0_H
/* Includes */
#include "r_cg_userdefine.h"
#include "r_cg_rspi.h"
/* Macro definitions (Register bit) */
/* Macro definitions */
#define _13_RSPI0_DIVISOR               (0x13U) /* SPBR(RSPI bit rate) register value */
/* Typedef definitions */
/* Global functions */
void R_Config_RSPI0_Create(void);
void R_Config_RSPI0_Start(void);
void R_Config_RSPI0_Stop(void);
MD_STATUS R_Config_RSPI0_Send_Receive(uint16_t * const tx_buf, uint16_t tx_num, uint16_t * const rx_buf);
static void r_Config_RSPI0_callback_transmitend(void);
static void r_Config_RSPI0_callback_receiveend(void);
static void r_Config_RSPI0_callback_error(uint8_t err_type);
void R_Config_RSPI0_Create_UserInit(void);
/* Start user code for function. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
#endif
```

8.2.2.5　r_cg_macrodriver.h の修正

本移行方式では r_bsp を使用しないため、r_cg_macrodriver.h から platform.h のインクルード部分を削除してください。r_bsp を使用する場合は対応不要です。

r_cg_macrodriver.h

```
/* DISCLAIMER */
/* File Name    : r_cg_macrodriver.h */
#ifndef MACRODRIVER_H
#define MACRODRIVER_H
/* Includes */
//#include "platform.h"
#include "r_smc_interrupt.h"
#include <machine.h>
/* Macro definitions (Register bit) */
```

8.2.3　API 関数の修正

レゾルバドライバを Rev1.20 以前から Rev2.10 に変更する場合、いくつかの API 仕様が異なるため、実装方法も変更が必要になります。以下に変更が必要な API 関数一覧を示します。修正については、「8.2.3.1 API 関数　R_RSLV_CreatePeripheral の削除」の項以降を参照してください。

表 8-1　修正する API 関数一覧

API 関数	変更内容	変更方法
R_RSLV_CreatePeripheral(ST_INIT_REG_PARAM　*rdc_init_param)	削除	本関数の使用箇所を削除してください。
R_RSLV_SetFuncTable(unsigned char set_func, FUNCTION_TABLE user_func_table)	追加	初期化処理に実装してください。 RESOLVER_peripheral_init()、RDC_peripheral_init 内の、R_RSLV_CreatePeripheral()設定を削除し、同じ個所に関数テーブル設定処理を記載してください。
R_RSLV_SetSystemInfo(ST_SYSTEM_PARAM *rdc_sys_param)	変更	下記の様に変更してください。 R_RSLV_SetSystemInfo(ST_SYSTEM_PARAM *rdc_sys_param, ST_USER_PERI_PARAM *user_peri_param)
R_RSLV_SetCaptureTiming(uint16_t tcnt)	変更	下記関数に置き換えてください。 R_RSLV_ESigCapStartTiming (uint16_t esig_start_tcnt, uint16_t cap_start_tcnt)
R_RSLV_EsigStartTiming(uint16_t tcnt)	変更	
R_RSLV_Rdc_RegWrite(uint8_t wt_data, uint8_t address, uint8_t *write_status)	変更	呼び出し関数を、下記に変更してください。 R_RSLV_Rdc_RegWrite(unsigned char *write_status) また、呼び出し方を下記の様に変更してください。 （修正前） R_RSLV_Rdc_SetRegisterVal (data1,address1); R_RSLV_Rdc_SetRegisterVal (data2,address2); R_RSLV_Rdc_RegWrite(data3,address3,&com_sts); （修正後） R_RSLV_Rdc_SetRegisterVal (data1,address1); R_RSLV_Rdc_SetRegisterVal (data2,address2); R_RSLV_Rdc_SetRegisterVal (data3,address3); R_RSLV_Rdc_RegWrite(&com_sts);
R_RSLV_INT_RdcCom_Recv(void)	削除	呼び出し箇所を削除してください。
R_RSLV_INT_RdcCom_Trans(void)	削除	呼び出し箇所を削除してください。
R_RSLV_INT_RdcCom_Error(void)	削除	呼び出し箇所を削除してください。
R_RSLV_INT_RdcCom_Idle(void)	削除	呼び出し箇所を削除してください。
R_RSLV_SetFunctionPointer(UNSIGNED_CHAR_POINTER *func, unsigned char func_id)	削除	呼び出し箇所を削除してください。
R_RSLV_Rdc_CallComEndCb(void)	追加	通信送受信完了割り込みコールバック処理に追加してください。
R_RSLV_Rdc_CallErrorCb()	追加	通信エラー割り込みコールバック処理に追加してください。
R_RSLV_ADJST_SetPtrFunc(st_ptr_func_arg_t *ptr_arg)	変更	戻り値を返すように変更しています。 必要に応じて戻り値を判定してください

8.2.3.1　API 関数　R_RSLV_CreatePeripheral の削除

Rev2.10 から本ペリフェラル初期化 API 関数は削除したため、使用箇所を削除してください。STM 版サンプルコードでは、以下の関数で使用しています。

- RESOLVER_peripheral_init(void)
- RDC_peripheral_init(void)

例）
（削除）

```
    // MTU3_9 ESig12
//    rdc_init_param.u1_sel_reg_type    = T_MTU3_9;
//    rdc_init_param.u1_sel_reg_func    = F_ESIG12;
//    rdc_init_param.u1_sel_int_flg     = INT_ENABLE;
//    rdc_init_param.u1_sel_int_priorty = 11;
//    rdc_init_param.u1_capture_trig    = CAPTURE_TRIG_NONE;
//    rdc_init_param.u1_use_port1       = P_P21;
//    rdc_init_param.u1_use_port2       = P_PE0;
//    rdc_init_param.u1_use_port3       = 0xFF;       // Not used
//    rdc_init_param.u1_use_port4       = 0xFF;       // Not used
//    R_RSLV_CreatePeripheral(&rdc_init_param);
```

8.2.3.2　API 関数　R_RSLV_SetFuncTable の追加

R_RSLV_CreatePeripheral を削除した箇所に関数テーブルの設定処理を組み込みます。STM 版サンプルコードの対象コードは、8.2.3.1 と同様に以下の関数になります。

- RESOLVER_peripheral_init(void)
- RDC_peripheral_init(void)

例）
（削除）

```
    // MTU3_9 ESig12
//    rdc_init_paramu1_sel_reg_type    = T_MTU3_9;
//    rdc_init_param.u1_sel_reg_func    = F_ESIG12;
//    rdc_init_param.u1_sel_int_flg     = INT_ENABLE;/
//    rdc_init_param.u1_sel_int_priorty = 11;
//    rdc_init_param.u1_capture_trig    = CAPTURE_TRIG_NONE;
//    rdc_init_param.u1_use_port1       = P_P21;
//    rdc_init_param.u1_use_port2       = P_PE0;
//    rdc_init_param.u1_use_port3       = 0xFF;       // Not used
//    rdc_init_param.u1_use_port4       = 0xFF;       // Not used
//    R_RSLV_CreatePeripheral(&rdc_init_param);
```

（追加）

```
    /* Set up the function table for ESig */
    g_st_user_func_table.Start   = &R_Config_MTU9_Esig_Start;
    g_st_user_func_table.Stop    = &R_Config_MTU9_Esig_Stop;
    g_st_user_func_table.SetTcnt = &R_Config_MTU9_Esig_SetTcnt;
    g_st_user_func_table.GetTcnt = &R_Config_MTU9_Esig_GetTcnt;
    R_RSLV_SetFuncTable(F_ESIG12, g_st_user_func_table);
```

8.2.3.3　API 関数　R_RSLV_SetSystemInfo の変更

R_RSLV_SetSystemInfo の引数（パラメータ）を変更します。引数（パラメータ）の内容については、「6.3.2　R_RSLV_SetSystemInfo API 関数の構造体」を参照してください。STM 版サンプルコードで変更する箇所は、以下の関数になります。

- RESOLVER_peripheral_init(void)
  本関数内で、下記の様に修正してください。

（修正前）

```
    /* RX24T 100 pins */
    st_system_param.u1_mcu_type = MCU_TYPE_R5F524TAADFP;
    /* Excitation signal (ESig) frequency 5 kHz */
    st_system_param.u1_esig_freq = R_ESIG_SET_FREQ_20K;
    /* Correction signal (CSig) frequency 400 kHz */
    st_system_param.u1_csig_freq = R_CSIG_SET_FREQ_200K;
    /* Update the duty cycle 2 times. */
    st_system_param.u1_csig_upd_duty_cycle = R_CSIG_SET_DCNT_02;
    /* Use MTU synchronous start. */
    st_system_param.u1_mtu3_sync_start = MTU_SYNC_START_ENABLE;
    /* Target motor is a BLDC motor. */
    st_system_param.u1_motor_kind = MOTOR_STM;
    st_system_param.u1_extension_use = R_EXT_INACTIVE;
    R_RSLV_SetSystemInfo(&st_system_param);
```

（修正後）

```
    /* Excitation signal (ESig) frequency 20 kHz */
    st_system_param.u1_esig_freq = R_ESIG_SET_FREQ_20K;
    /* Correction signal (CSig) frequency 200 kHz */
    st_system_param.u1_csig_freq = R_CSIG_SET_FREQ_200K;
    /* Update the duty cycle 2 times. */
    st_system_param.u1_csig_upd_duty_cycle = R_CSIG_SET_DCNT_02;
    /* Use MTU synchronous start. */
    st_system_param.u1_sync_start = SYNCMD_OTHER_API;
    /* Target motor is a STM motor. */
    st_system_param.u1_motor_kind = MOTOR_STM;
    /* RDC IC MNTOUT output method */
    st_system_param.u1_mntout_type = RSLV_MNTOUT_TYPE_AC;
    st_user_peri_param.f_esig1_peri_clk_src = 80.0f;
    st_user_peri_param.f_csig_peri_clk_src = 80.0f;
    st_user_peri_param.f_csig_upd_timer_peri_clk_src = 5.0f; // CMT:PCLKB/8
    st_user_peri_param.f_capture_peri_clk_src = 80.0f;
    st_user_peri_param.f_phase1_peri_clk_src = 40.0f;
    st_user_peri_param.f_phase2_peri_clk_src = 40.0f;
    R_RSLV_SetSystemInfo(&st_system_param, &st_user_peri_param);
```

8.2.3.4　API 関数　R_RSLV_SetCaptureTiming、R_RSLV_EsigStartTiming の変更

R_RSLV_SetCaptureTiming と R_RSLV_EsigStartTiming の API 関数を削除して、適切な場所に、R_RSLV_Set_EsigCapTiming を追加してください。API 関数の使用方法については、「6.2.20　励磁信号出力開始タイミング設定 API 関数」を参照してください。

下記の様に修正してください。

（修正前）

```
    R_RSLV_SetCaptureTiming(DEF_SFT_ADJ_ESIG);      /* Capture start timing */
    R_RSLV_ESigStartTiming(DEF_DELAY_ADJ_ESIG);     /* Esig start timing*/
```

（修正後）

```
    R_RSLV_ESigCapStartTiming(DEF_DELAY_ADJ_ESIG, DEF_SFT_ADJ_ESIG); /* Esig &
Capture start timing*/
```

8.2.3.5　API 関数　R_RSLV_Rdc_RegWrite の変更

本関数の引数を変更してください。API 関数の使用方法については、「6.2.30　RDC-IC 各レジスタへの書き込み API 関数」を参照してください。また、API 関数の呼び出し方を下記の様に変更してください。

（修正前）

```
R_RSLV_Rdc_SetRegisterVal (data1,address1);
R_RSLV_Rdc_SetRegisterVal (data2,address2);
R_RSLV_Rdc_RegWrite(data3,address3,&com_sts);
```

（修正後）

```
R_RSLV_Rdc_SetRegisterVal (data1,address1);
R_RSLV_Rdc_SetRegisterVal (data2,address2);
R_RSLV_Rdc_SetRegisterVal (data3,address3);
R_RSLV_Rdc_RegWrite(&com_sts);
```

8.2.3.6　API 関数　R_RSLV_INT_RdcCom_Recv の削除

SC にて SPI 通信の受信割り込み処理を作成するため、Rev2.10 から本 API 関数を削除しています。従って、SC で生成された受信割り込み処理を使用するようにしてください。SC で生成される受信割り込み処理については、「7.10.2.3　SCI を使用した例」の受信割り込み処理（SC **生成**コード）を参照してください。

8.2.3.7　API 関数　R_RSLV_INT_RdcCom_Trans の削除

SC にて SPI 通信の送信割り込み処理を作成するため、Rev2.10 から本 API 関数を削除しています。従って、SC で生成された送信割り込み処理を使用するようにしてください。SC で生成される送信割り込み処理については、「7.10.2.3　SCI を使用した例」の送信割り込み処理（SC **生成**コード）を参照してください。

8.2.3.8　API 関数　R_RSLV_INT_RdcCom_Error の削除

SC にて SPI 通信のエラー割り込み処理を作成するため、Rev2.10 から本 API 関数を削除しています。従って、SC で生成されたエラー割り込み処理を使用するようにしてください。SC で生成されるエラー割り込み処理については、「7.10.2.3　SCI を使用した例」のエラー割り込み処理（SC **生成**コード）を参照してください。

8.2.3.9　API 関数　R_RSLV_INT_RdcCom_Idle の削除

SC にて SPI 通信のアイドル割り込み処理を作成するため、Rev2.10 から本 API 関数を削除しています。従って、SC で生成されたアイドル割り込み処理を使用するようにしてください。但し、SC にて SCIx を設定している場合、アイドル割り込み処理は作成されないため、本 API 関数を削除するだけとなります。

8.2.3.10 API 関数　R_RSLV_SetFunctionPointer の削除

SC で生成したコードにてチップセレクト信号を出力するため、Rev2.10 から本 API 関数を削除しています。従って、呼び出している箇所を削除してください。

8.2.3.11 API 関数　R_RSLV_Rdc_CallComEndCb の追加

SC で生成した SPI 通信割り込みコールバック関数（r_Config_(peri_func)_callback_transmitend()、r_Config_(peri_func)_callback_receiveend()）から、本 API 関数を呼び出すようにしてください。「7.10.2.3 SCI を使用した例」のコールバック処理を参照してください。

8.2.3.12 API 関数　R_RSLV_Rdc_CallErrorCb の追加

SC で生成した SPI 通信エラー割り込みコールバック関数（r_Config_(peri_func)_callback_error()）から、本 API 関数を呼び出すようにしてください。「7.10.2.3　SCI を使用した例」のコールバック処理を参照してください。

8.2.3.13 API 関数　R_RSLV_ADJST_SetPtrFunc の修正

本関数は戻り値を返すように修正しています。必要に応じて戻り値の対応をしてください。API 関数の使用方法は、「6.2.42　ユーザ作成のコールバック関数ポインタ設定 API 関数」を参照してください。

8.2.4　その他の修正

8.2.4.1　割り込み

SC で各ドライバ機能の割り込みを有効にした場合、割り込み関数が作成されます。既に同じ割り込みの関数が作成されている場合、SC で生成した割り込み関数に処理を組み込み、元の割り込み関数を削除することを推奨します。

8.2.4.2　構造体の追加と削除

Rev2.10 から ST_USER_PERI_PARAM を追加、ST_INIT_REG_PARAM を削除しています。下記の様に STM 版サンプルコードを修正してください。

- RESOLVER_peripheral_init(void)
  ST_INIT_REG_PARAM の定義と使用箇所を削除してください。
- RDC_peripheral_init(void)
  ST_INIT_REG_PARAM の定義と使用箇所を削除してください。
  ST_USER_PERI_PARAM の定義を追加し、R_RSLV_SetSystemInfo()の引数に追加してください。

9. 注意事項

初期設定時における注意事項を以下に示します。

9.1 初期設定手順

初期設定は下記の手順で行ってください。

1. システム情報の指定（R_RSLV_SetSystemInfo()）
2. 各関数テーブル設定（R_RSLV_SetFuncTable()）
3. RDC-IC ドライバ設定情報取り出し（R_RSLV_GetRdcDrvSettingInfo()）
4. その他設定

上記手順以外で設定を行うと、使用するタイマ値や RDC-IC ドライバ設定情報の異常となる場合があります。

9.2 同一ペリフェラルへの複数機能設定

同じペリフェラルに複数の機能を設定しないでください。設定異常にはなりませんが、最後に行った設定のみ有効となります。

設定例）MTU3_9 に ESIG12 と CAPTURE を設定する
TMR0 に RDC_CLK と PHASE_A を設定する

9.3 同一機能への複数ペリフェラル設定

同じ機能に複数のペリフェラルを設定しないでください。設定異常にはなりませんが、最後に行った設定のみ有効となります。

設定例）ESIG12 に MTU3_C0 と MTU3_C9 を設定する
PHASE_A に TMR0 と TMR1 を設定する

9.4 RDC-IC 通信用変数の初期化

RDC-IC 用通信変数初期化（R_RSLV_Rdc_VariableInit）実施前に RDC-IC 通信処理を実施しないでください。RDC-IC レジスタ値が異常となる場合があります。

9.5 位相調整信号のペリフェラル設定

位相調整用信号 F_PHASE_A と F_PHASE_B に同じペリフェラルを設定しないでください。設定異常になりませんが、位相調整信号が正常に出力されなくなります。

設定例）PHASE_A に TMR0、PHASE_B に TMR0 を設定する

9.6 タイマスタートタイミング設定

励磁信号、角度信号入力タイマのスタートタイミング設定は、タイマ開始前に行ってください。タイマのカウントが異常となり、角度信号の取り込み値が期待通りの値にならない場合があります。

9.7 調整機能の動作について

調整機能が動作するためには、基本機能が動作している必要があります。項 7.4~項 7.7、項 7.10 の機能を停止した状態で、調整機能を動作させないでください。

9.8　角度誤差補正位相シフト量

位相シフト量を 0 付近に設定すると、角度誤差補正信号の位相が設定値通りに変化しない場合があります。これは、励磁信号割り込みの中で角度誤差補正信号同期スタート API 関数（R_RSLV_INT_CSig_SyncStart）を実行した場合、プログラム処理時間や割込み禁止区間の影響により、Duty 更新割り込みタイマのスタートが遅れてしまうためです。また角度誤差補正信号 Duty 更新タイマの割り込みが発生した時に他の割り込み（角度検出など）を処理している場合も、先に発生した割り込みを処理してから更新割り込みを処理するため、補正信号の Duty 更新が遅れてしまいます。

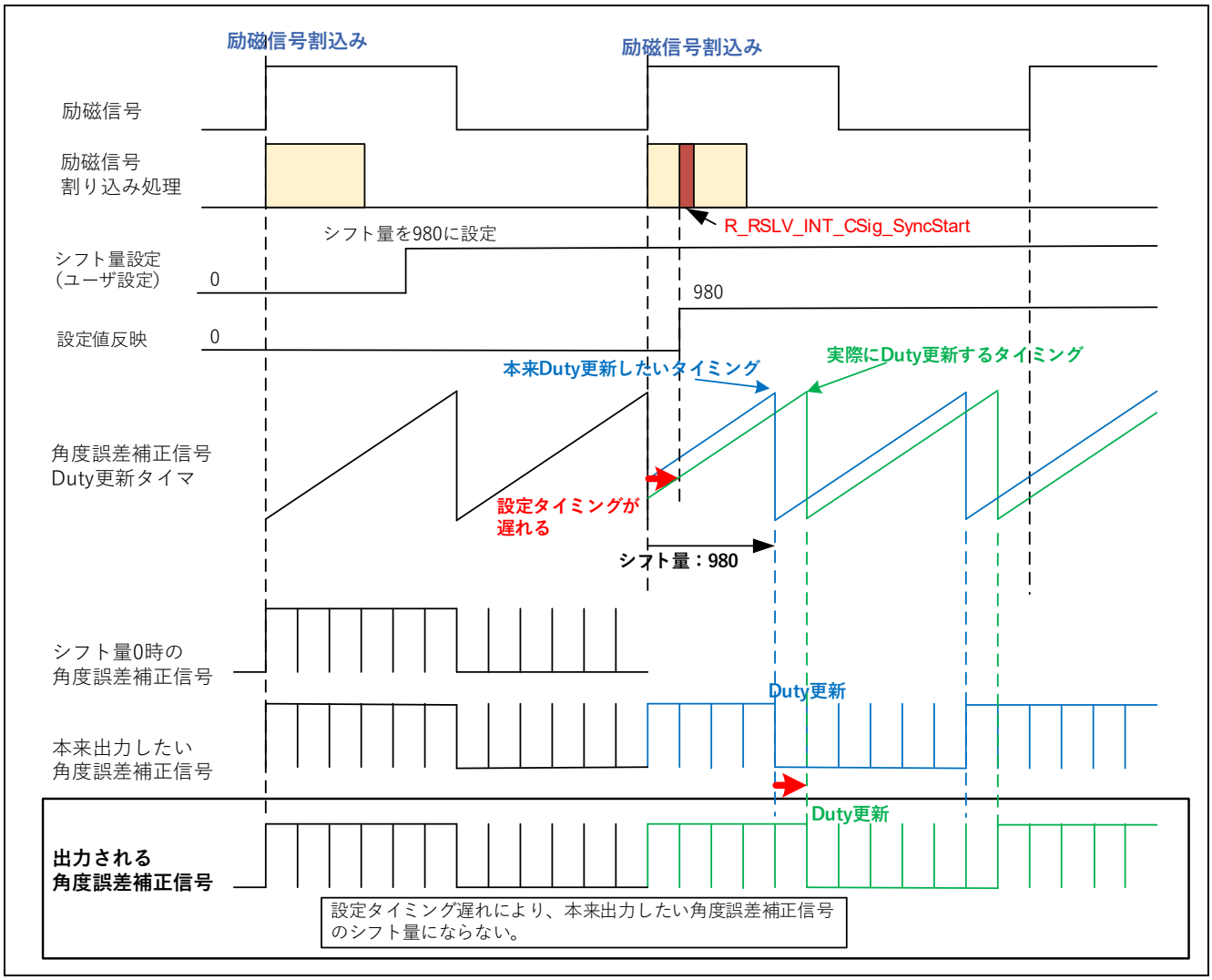

図 9.1　発生のメカニズム

この現象を解消するには、以下のように設定してください。

1. 角度誤差補正信号 Duty 更新割り込みスタート用タイマを設定してください。
   — 励磁信号と同じ周期で割り込みを生成するよう、空きタイマを設定してください。（以降、更新割り込みスタート用タイマとします）
   — 更新割り込みスタート用タイマのカウントスタートは、励磁信号割り込み処理で実施してください。また、カウントスタートする前に以下を設定してください。
   - 励磁信号と更新割り込みスタート用タイマの割り込みタイミングが一致するよう、更新割り込みスタート用タイマのカウンタ初期値を調整してください。
   - 更新割り込みスタート用タイマの割り込みを許可してください。
   - 更新割り込みスタート用タイマの割り込みを優先的に実施するよう、励磁信号・角度検出・角度誤差補正信号 Duty 更新割り込みより高い割り込みレベルを設定してください。また励磁信号割り込みは、多重割り込みを許可してください。
   — 角度誤差補正信号を停止している場合に上記を設定してください。（角度誤差補正信号の設定値を更新する場合など）

2. 更新割り込みスタート用タイマ処理内で以下を実施してください。
- R_RSLV_INT_CSig_SyncStart をコールしてください
- 更新割り込みスタート用タイマを禁止してください

上記設定により、角度誤差補正信号の位相シフト量が 0 付近（=励磁信号とほぼ同位相）であっても、R_RSLV_INT_CSig_SyncStart を正しいタイミングで実行する事が出来ます。また角度誤差 Duty 更新割り込みの発生タイミングも改善します。以下に動作イメージを示します。

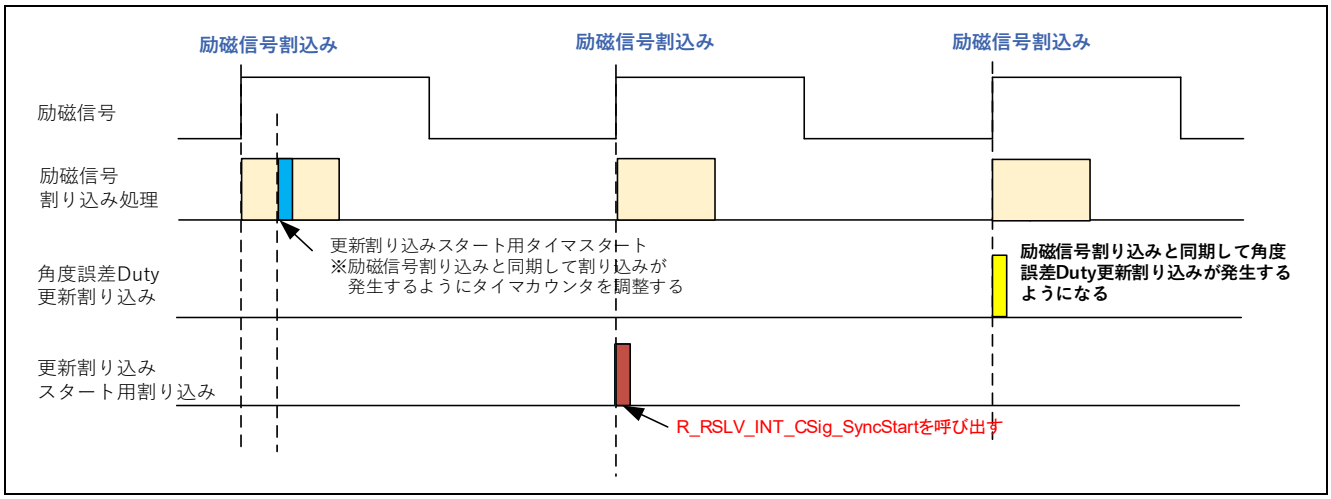

図 9.2　解消方法例

9.9　関数テーブルの設定順序

励磁信号を F_ESIG2_1 と F_ESIG2_2 で出力する場合、関数テーブルの設定は、F_ESIG2_1 を先に設定して下さい。励磁信号が正しく出力されなくなります。

9.10 角度誤差補正信号調整機能について

角度誤差補正信号調整機能を使用する際は、角度誤差補正信号 Duty 更新回数を 2 回に設定してください。角度誤差補正信号の位相シフト量が正しく調整されない場合があります。

10. トラブルシューティング

本章では、レゾルバ信号を検出できない場合の対応例を示します。万一、トラブルが発生した際は図 10.1 を参考に原因を特定してください。

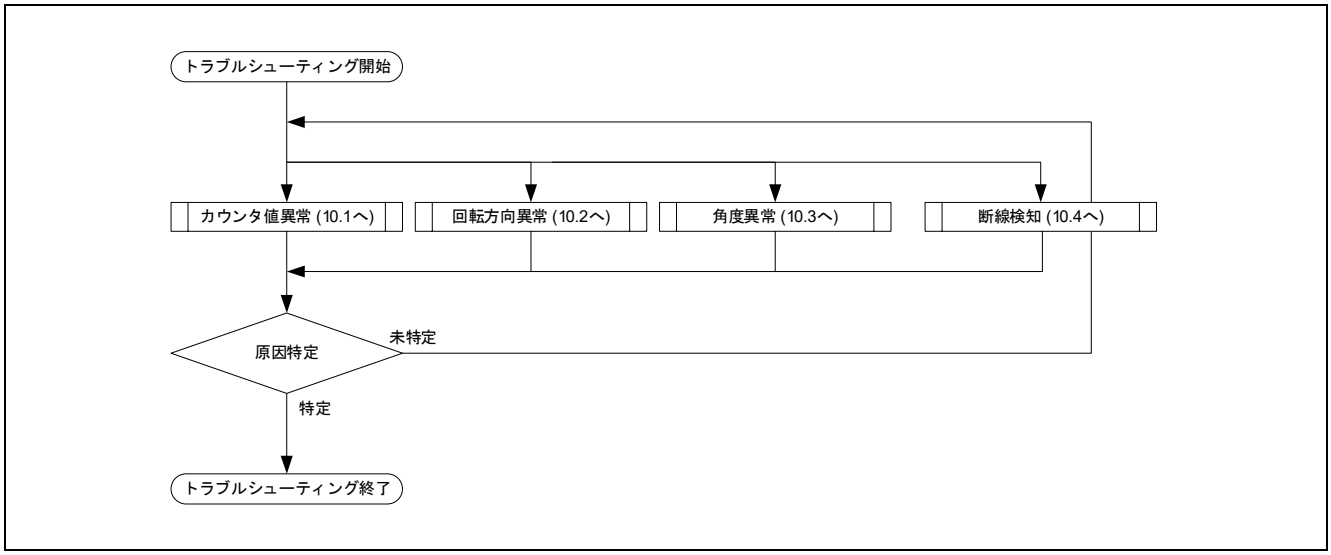

図 10.1　トラブルシューティング全体フロー

10.1 カウンタ値異常

MCU で位相情報のカウンタ値に異常があった場合は図 10.2 に従って原因を特定してください。なお、断線検知の詳細については「10.4　断線検知」を参照してください。

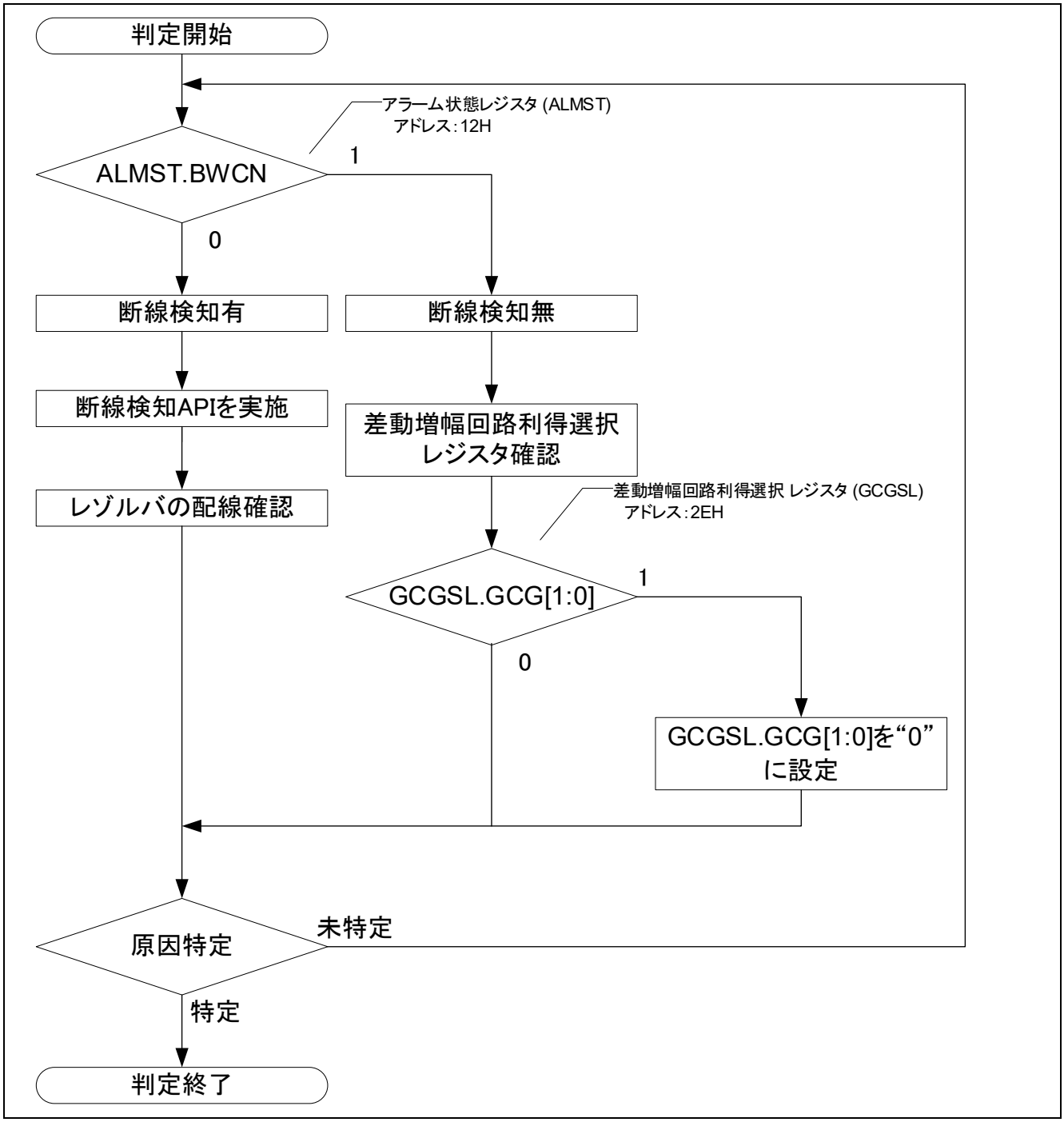

図 10.2　カウンタ値異常

10.2 回転方向異常

回転方向が想定している方向と逆となった場合やレゾルバを物理的に電気角 1 回転させたが、位相情報として 1 回転しなかった場合等は図 10.3 のフローに従って原因を特定してください。

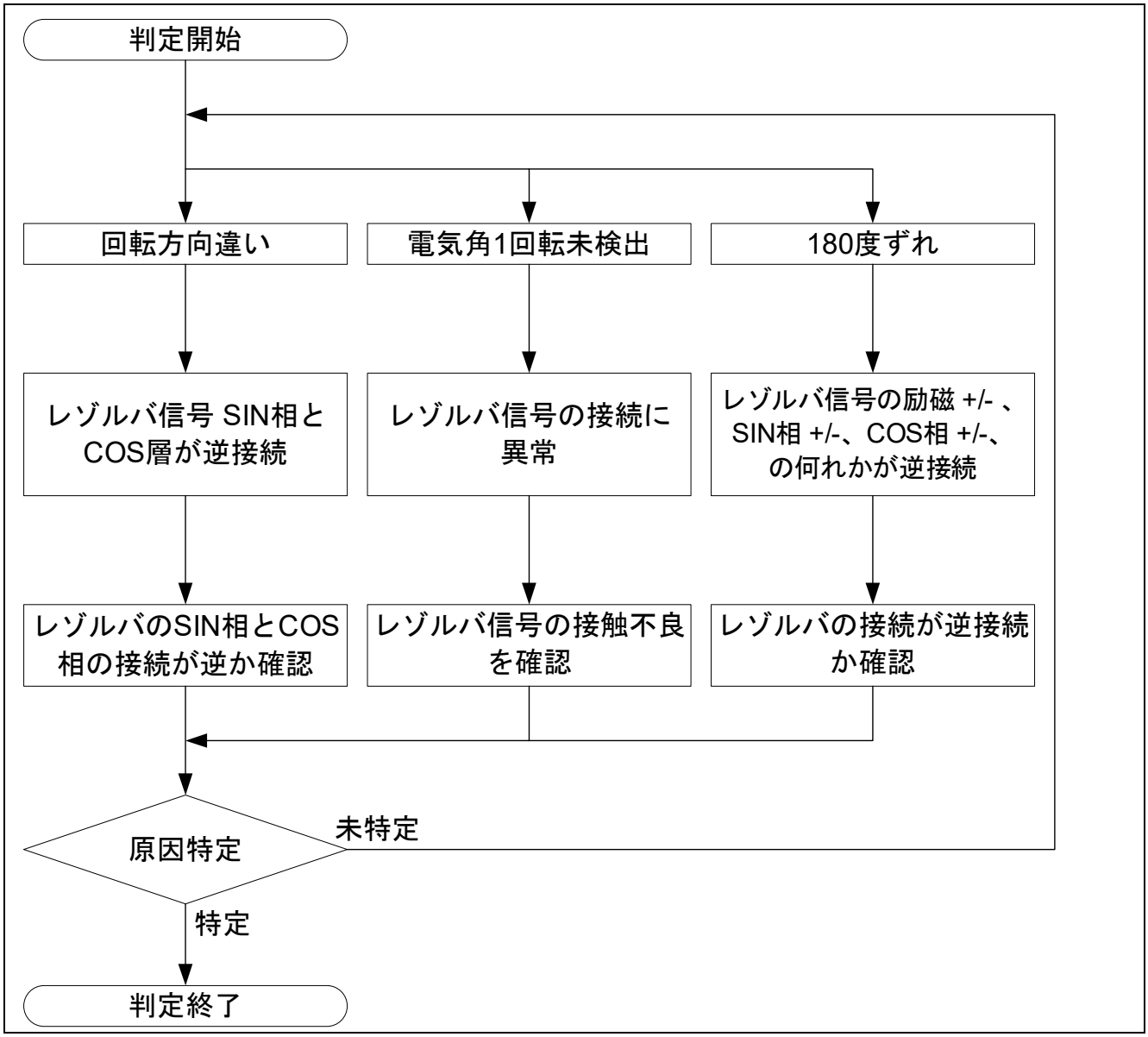

図 10.3　回転方向異常

10.3 角度異常

レゾルバの位相情報が想定した角度と異なる場合は信号波形に異常がある可能性があります。その際は、アナログモニタ出力の波形を確認してください。なお、アナログモニタ出力に波形を出力する場合は、パワーセーブ制御レジスタ 3（PS3）の 5bit 目 PSMON を“1”、モニタ出力選択レジスタを所望の値に設定してください。

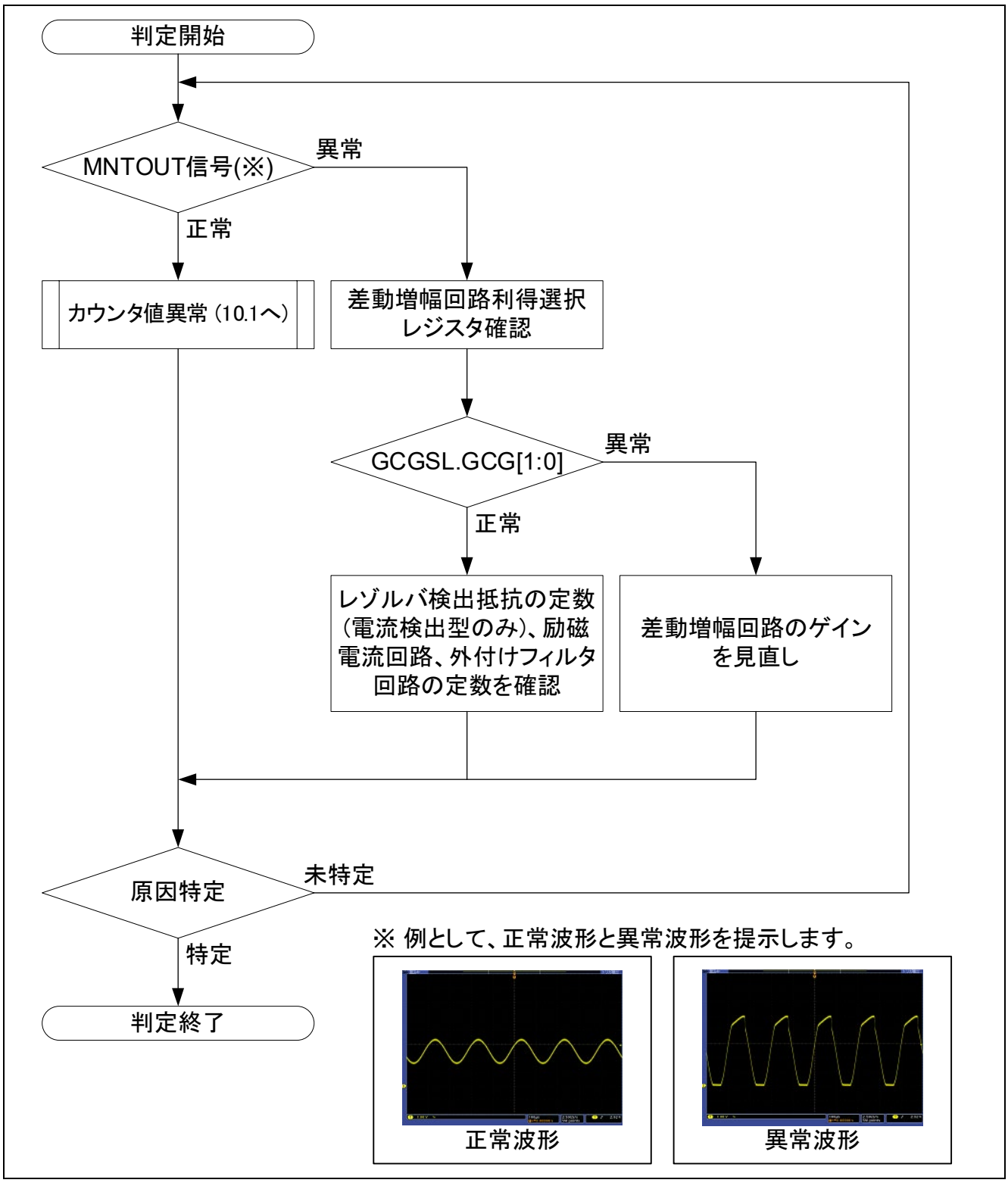

図 10.4　角度異常

10.4 断線検知

RAA3064002GFP/RAA3064003GFP は断線を検知するのみであり、断線を検知した後は MCU で励磁指令信号を停止する等の処理を実施してください。断線検知の設定方法については、「7.11　断線検知機能」を参照してください。

次に、断線検知のパターンについて説明します。断線検知はご使用になるレゾルバの構成により検知方法が異なります。

トランス型レゾルバの場合を次に示します。

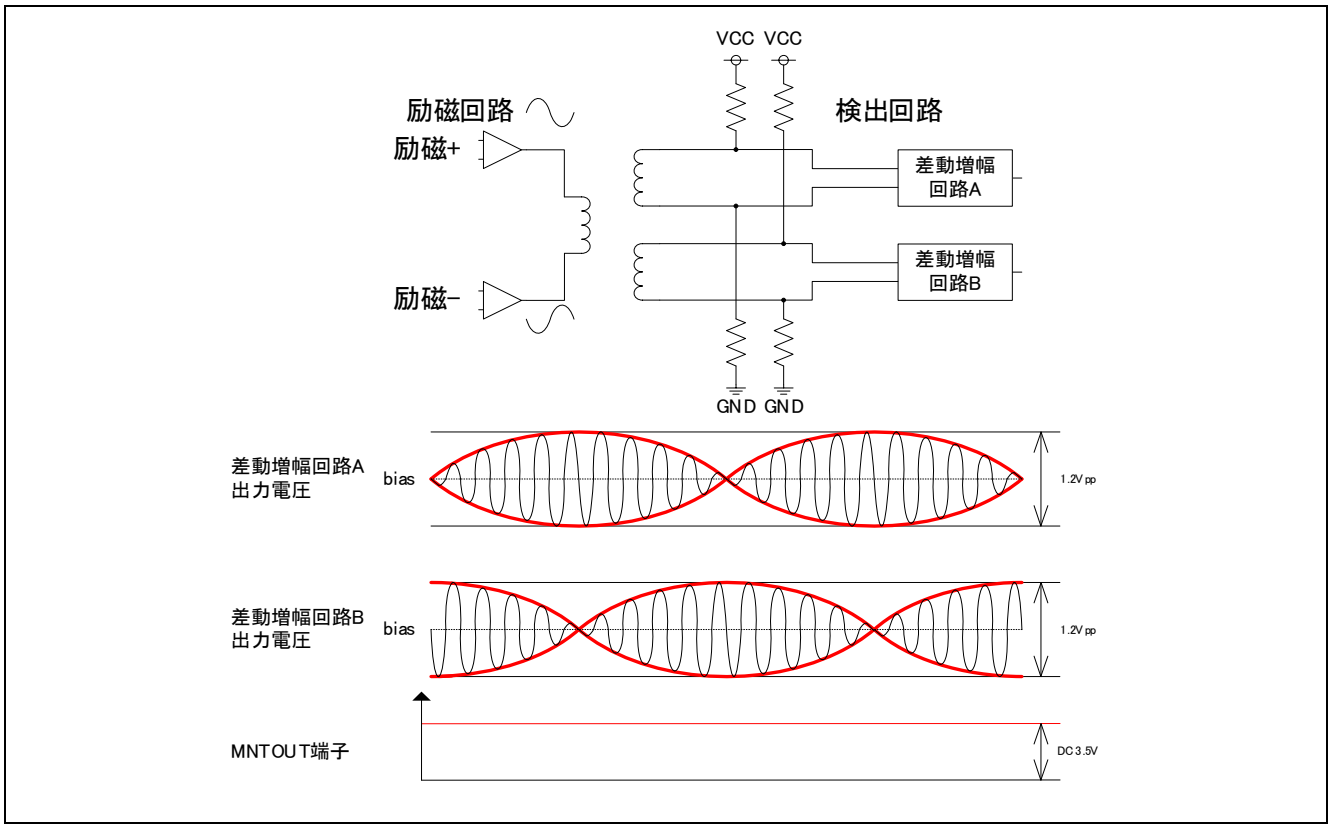

図 10.5　通常状態

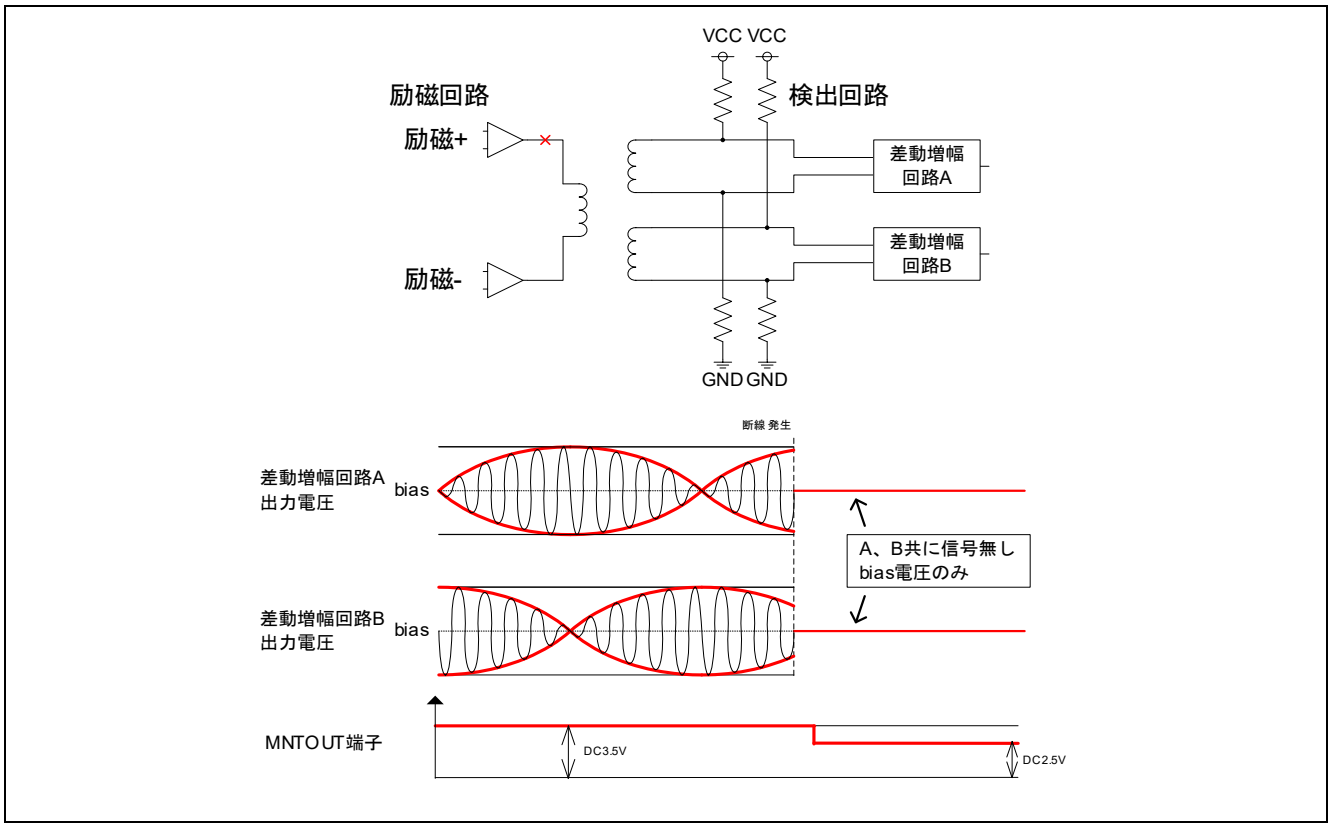

図 10.6　励磁側断線

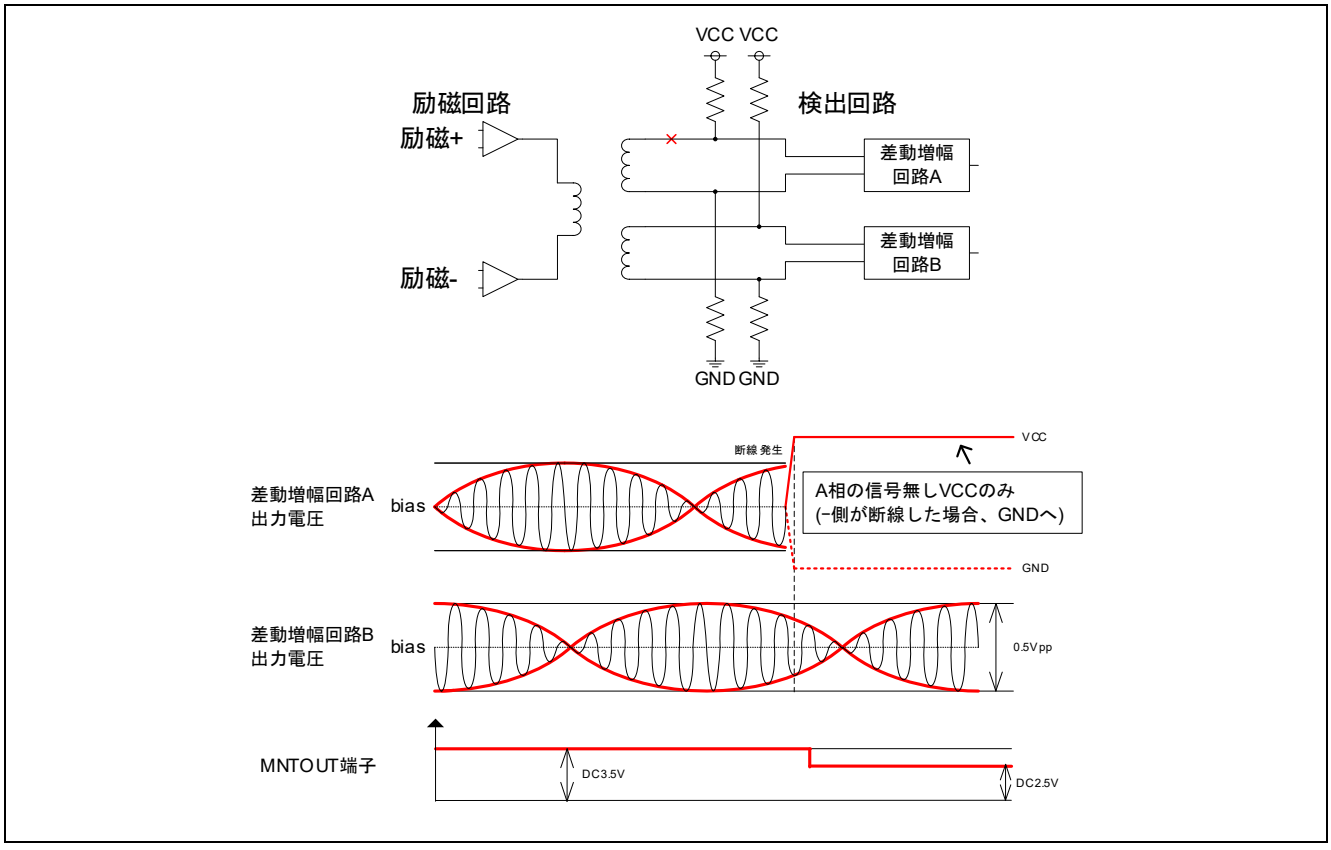

図 10.7　SIN+側断線

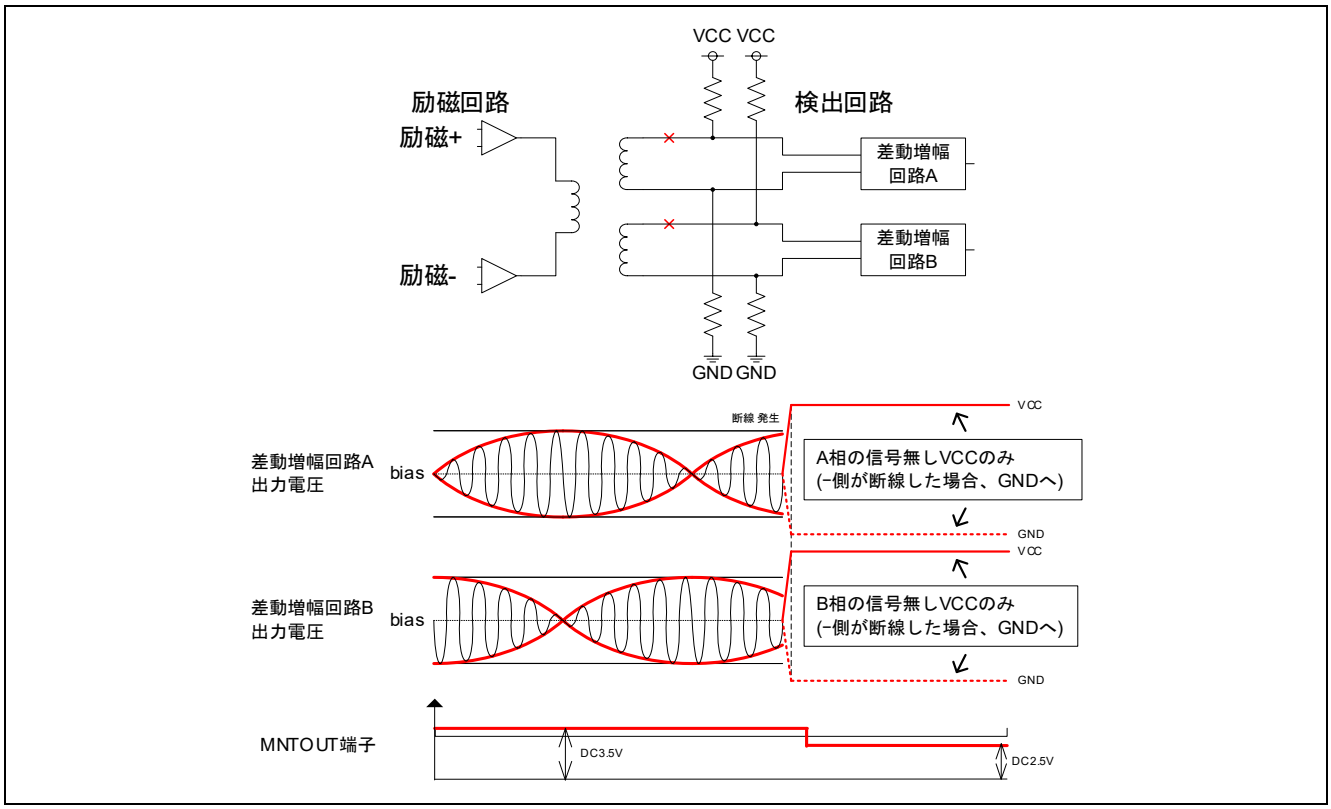

図 10.8　SIN+、COS+側断線

電流検出型レゾルバの場合を次に示します。

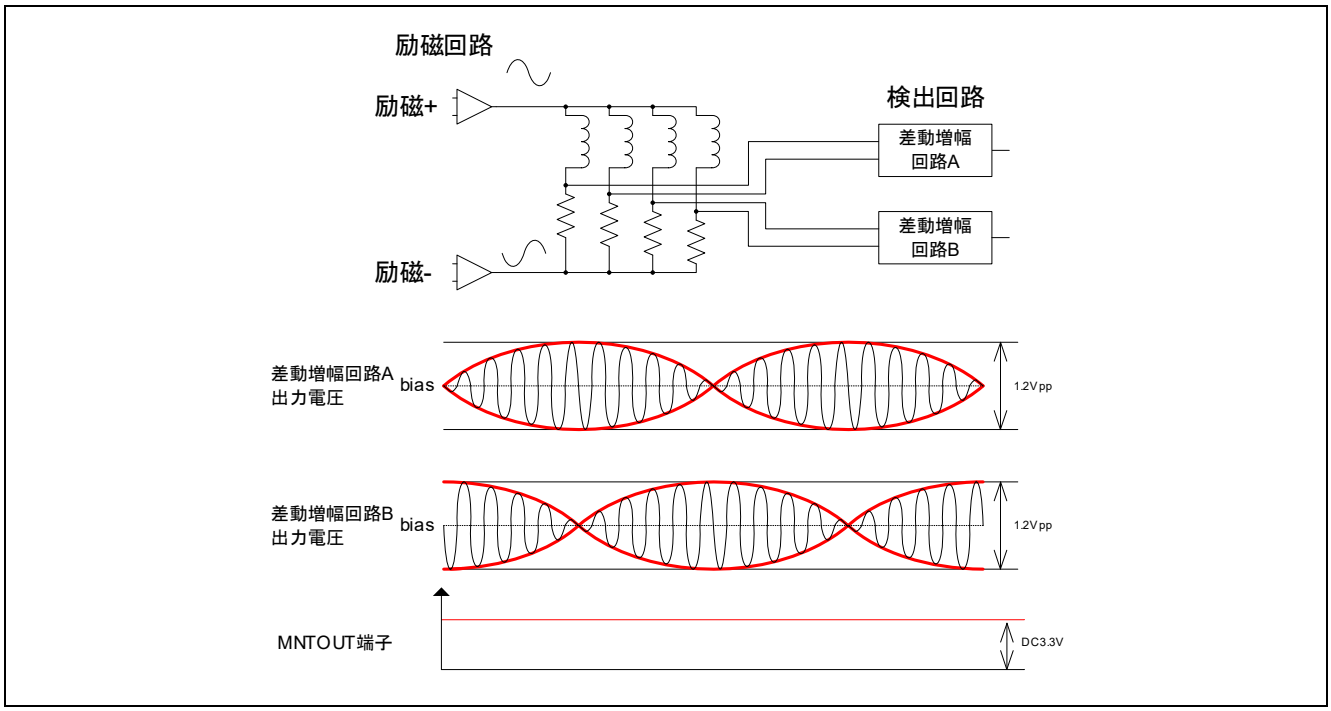

図 10.9　通常時

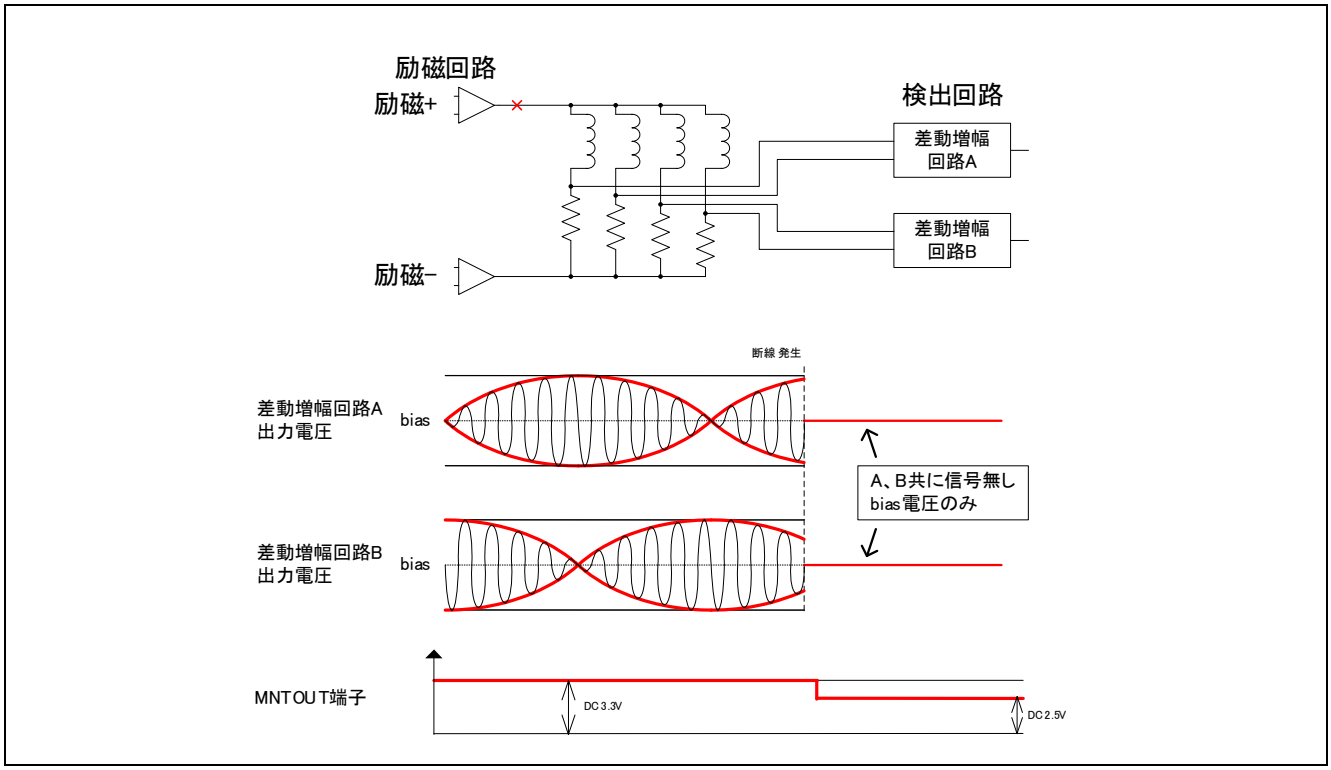

図 10.10　励磁側断線

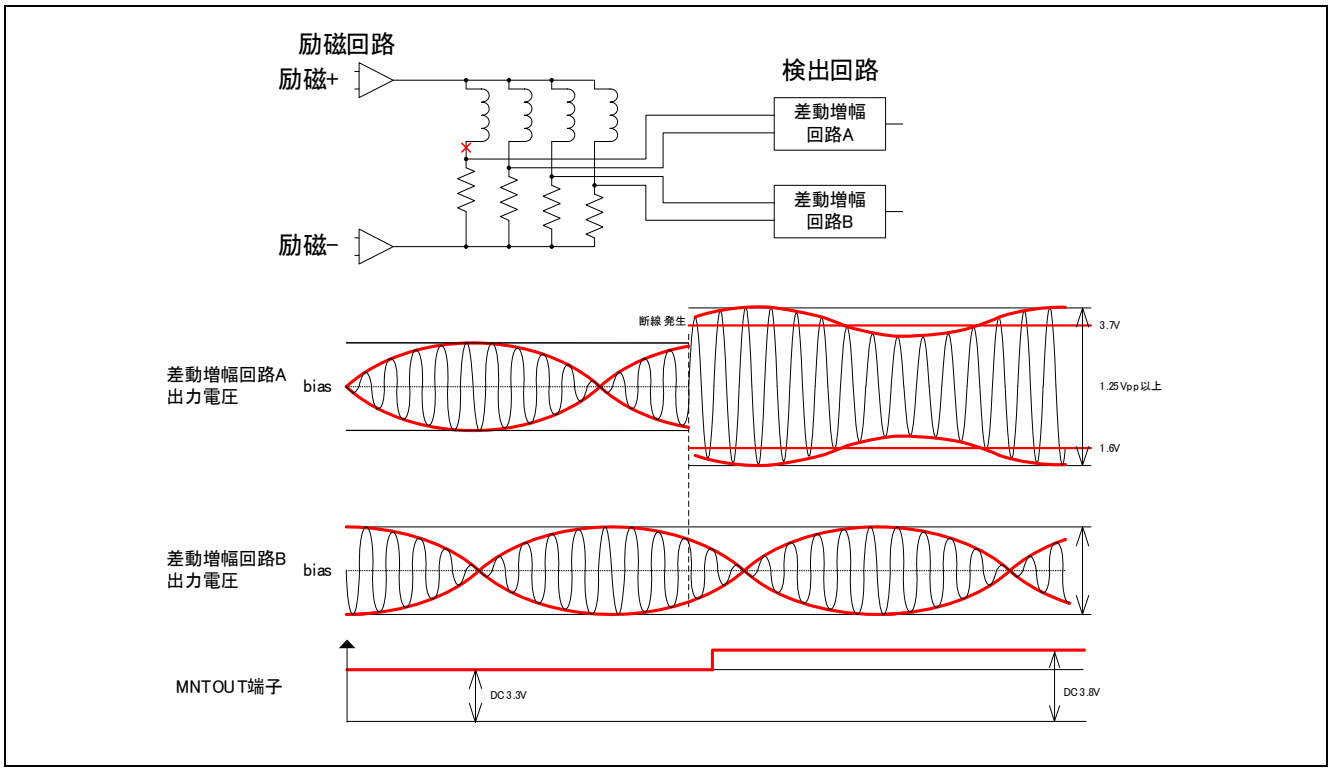

図 10.11　0 度－側断線

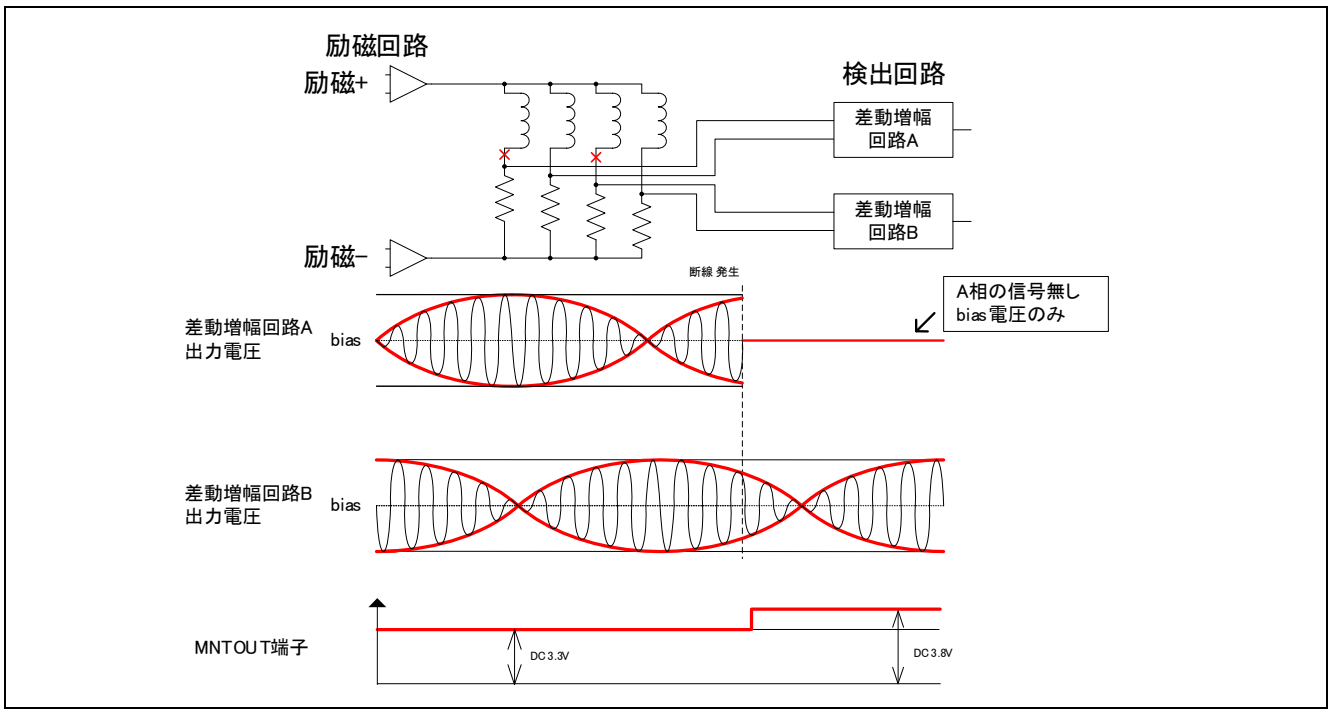

図 10.12　0 度－、180 度－側断線

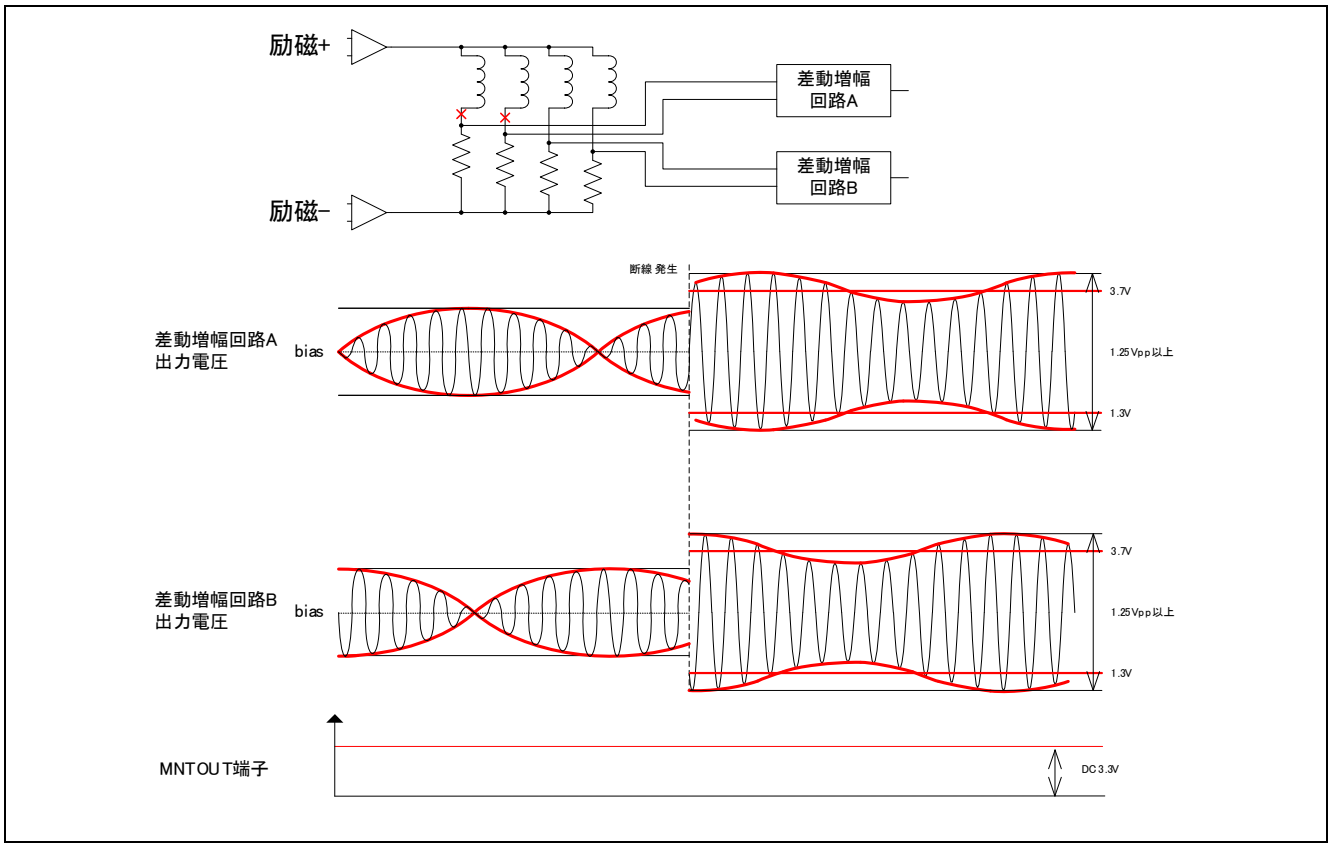

図 10.13　0 度－、90 度－側断線

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Jan.29.21	—	初版発行
1.10	Feb.06.23	全体	対象 MCU に RX72T を追加。
		7	表 1-1 の開発環境の各バージョンを更新。 表 1-2 の ROM サイズ、RAM サイズを更新。 “1.4 関連資料”にレゾルバ信号変換 IC の周辺部品選定ガイドを追加。
		24	“3.10.1 角度誤差補正信号出力開始”に、位相シフト量と振幅レベルの値は”3.11 自動調整機能”を参照する旨の記載を追加。
		28	表 3-1 の RX66T / RX72T、CMT アサイン時のソースクロック設定、及び位相シフト量の入力範囲を修正。
		29	“3.11.3.1 フィルタ回路による遅れ位相”を追加。
		30	“3.12 タイミングチャート（励磁信号、角度信号入力、角度誤差補正信号）”を追加。
		66	表 6-1 に R_RSLV_GetCSigStatus を追加。
		69	表 6-1 に R_RSLV_ADJST_SetFilterDelay を追加。
		72	“6.2.10 角度誤差補正信号出力状態取得 API 関数”を追加。
		80	“6.2.44 遅れ位相設定用 API 関数”を追加。
		110	図 7.10 に R_RSLV_GetCSigStatus の実装例を追加。
		123	“7.9.3 フィルタ回路による遅れ位相”を追加。
		143	表 8-1 から R_RSLV_GetCSigStatus を削除。
		147	“8.2.3.13 API 関数　R_RSLV_GetCSigStatus の削除”を削除。
		150	“9.10 角度誤差補正信号調整機能について”を追加。

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に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}（Max.）から V_{IH}（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}（Max.）から V_{IH}（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うと、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、ノイズ幅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。

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。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、お客様の責任において、お客様の機器・システムを設計ください。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. 当社製品または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. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 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。
6. 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）、交通制御（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。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 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。
9. 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。
10. 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て、当社は、一切その責任を負いません。
11. 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。
12. 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といたします。
13. 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。
14. 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.5.0-1　2020.10)

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。