Smart Analog IC101

API 仕様書

R21AN0015JJ0100
Rev.1.00
2014.09.30

要旨

本仕様書は、Smart Analog IC101 (RAA730101)を制御するための API（Application Program Interface；以下 API）に関する仕様について記述します。

目次

1. 仕様

1.1 概要

Smart Analog IC101 (RAA730101　以下、SAIC101) を制御するための、ルネサス製 MCU のシリアル・アレイ・ユニット(SAU)の UART を使用した調歩同期式通信制御または 3 線シリアル I/O (CSI・SPI)を使用したクロック同期式通信制御を用いた API 関数を提供します。本 API 関数では、主に SAIC101 のレジスタ・A/D 変換・フラッシュ・メモリの制御を行います。

1.2 動作確認条件

本仕様書のソースコードは、以下の条件で動作を確認しています。

表 1.1　動作条件一覧

項目	内容
評価ボード	・Renesas Starter Kit for RL78/L13 [R0K5010WMS900BE] - Renesas Starter Kit for RL78/L13 CPU ボード ・Smart Analog IC 搭載 RSK オプション評価ボード [TSA-OP-IC101]
使用マイコン	R5F10WMGAFB（RL78/L13）
動作周波数	24MHz
動作電圧	5.0V
統合開発環境 (CubeSuite+)	V2.02.00 [21 Feb 2014]
C コンパイラ (CubeSuite+)	CA78K0R V4.02.00.03 [16 Jan 2014]
RL78/L13 コードライブラリ (CubeSuite+)	V1.02.01.02 [11 Jun 2014] 注 1
統合開発環境 (e2studio)	V3.0.0.22
C コンパイラ(e2studio)	GNURL78 v14.01
RL78/L13 コードライブラリ (e2studio)	V1.02.00.03 [11 Feb 2014] 注 2

注 1 : CubeSuite+用のコードライブラリはコード生成プラグインに内包されています。本ドキュメントでは「CubeSuite+ Code_Generator for RL78_78K V2.04.00」で動作確認を行っております。
注 2 : e2studio 用のコードライブラリは e2studio 本体に内包されています。

本 API 関数内で使用している変数の型宣言は以下の通りです。

```
int8_t      : signed   char
uint8_t     : unsigned  char
int16_t     : signed   short
uint16_t    : unsigned  short
int32_t     : signed   long
uint32_t    : unsigned  long
MD_STATUS   : unsigned  short
```

注 : API では CubeSuite+のコード生成機能で生成されるヘッダファイル[r_cg_macrodriver.h]をインクルードしています。

1.3 ソフトウェア構成

図 1.1 にソフトウェア構成図を示します。

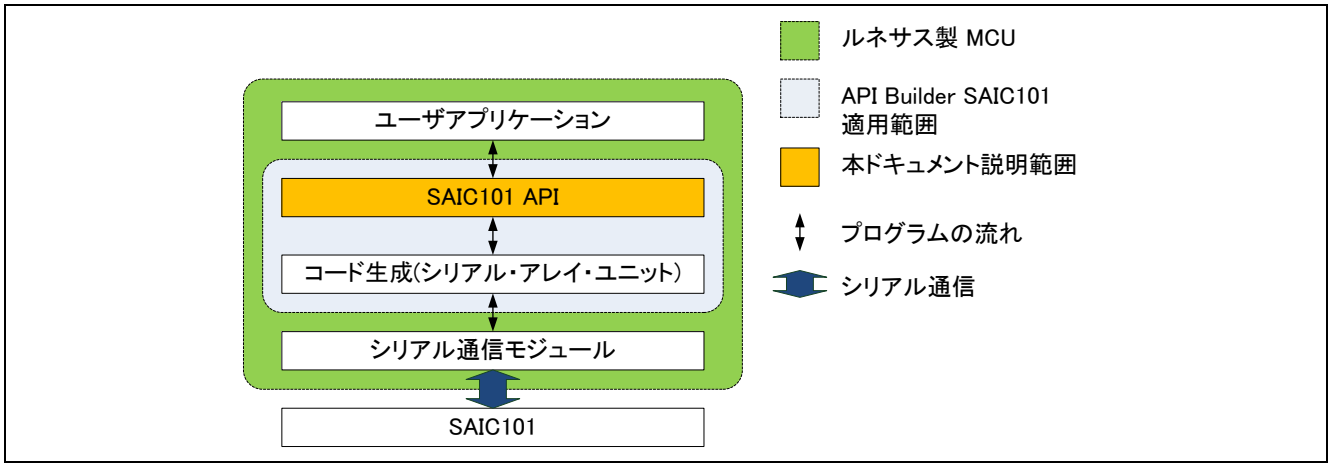


図 1.1 ソフトウェア構成図

2. API 使用時の注意事項

本章では、API を使用して SAIC101 を制御する場合の制約、注意事項について記載します。本章の内容について必ずご理解頂いた上で API をご使用ください。

2.1 [UART/SPI 共通] 使用上の注意

2.1.1 API ユーザー定義ファイルのグローバル変数、定義の書換えについて

API では、API ユーザー定義ファイルに「CPU と周辺クロック周波数(24MHz)」「シリアル・アレイ・ユニット(UART1)」などが、一例として設定されています。コード開発支援ツール「API Builder SAIC101」を使用しない場合には、グローバル変数、定義の一部をユーザー環境に合わせて手動で書き換える必要があります。詳しくは「Smart Analog IC101 サンプルコード導入手順書兼 API Builder SAIC101 仕様書 (RL78/L13 編)」(R21AN0012JJ) をご参照ください。

2.1.2 フラッシュ・メモリの書き換え回数について

ユーザープログラム上でループ中にフラッシュ・メモリを書き換えるなど、書き換え回数の上限(100 回)を超えるご使用はしないでください。詳しくは SAIC101 データシート「RAA730101 プログラマブル・ゲイン計装アンプ付き 16 ビット ΔΣ A/D コンバータ IC」(R02DS0014JJ) をご参照ください。

2.1.3 SBIAS を動作停止にする時の注意事項

「パワー／モード制御レジスタ (CHIPCNT)」の SENSPD ビットを 1 に設定すると SBIAS の動作が停止します。SBIAS が動作停止していると A/D 変換動作が行えませんのでご注意ください。詳しくは SAIC101 データシート「RAA730101 プログラマブル・ゲイン計装アンプ付き 16 ビット ΔΣ A/D コンバータ IC」(R02DS0014JJ) をご参照ください。

2.1.4 レジスタ・シャドウ領域書き込み時の注意事項

フラッシュ・メモリのレジスタ・シャドウ領域「00H 番地～1FH 番地」は、起動直後の通信を含む全ての動作に関わるレジスタ値であるため、書き込みには細心の注意が必要です。特に 01H 番地 [パワー／モード制御レジスタ (CHIPCNT)]・1FH 番地 [起動シーケンス／通信制御レジスタ (STARTUP)] への書き込みは注意してください。意図しない誤った値を書き込んだ場合には正常な通信が行えなくなる可能性があります。詳しくは SAIC101 データシート「RAA730101 プログラマブル・ゲイン計装アンプ付き 16 ビット ΔΣ A/D コンバータ IC」(R02DS0014JJ) をご参照ください。

2.1.5 API 関数内の待ち時間についての注意事項

本 API 関数内では指定時間待ち処理すべてにソフトウェアタイマを使用しています。ソフトウェアタイマの性質上、API 関数では期待する待ち時間以上の時間を設定していますのでご注意ください。なお、より正確な待ち時間が必要な場合には、ユーザー環境下で時間を測定し、NOP 回数を最適な値に変更してください。

2.2 [UART のみ] 使用上の注意

2.2.1 MOSI_RX 端子状態の注意事項

パリティ設定なしの状態で MOSI_RX 端子にキャラクタ長×3 以上のロウ・レベルが入力されると、以降の通信ができなくなりますのでご注意ください。詳しくは SAIC101 データシート「RAA730101 プログラマブル・ゲイン計装アンプ付き 16 ビット ΔΣ A/D コンバータ IC」(R02DS0014JJ) をご参照ください。

2.2.2 AREG を動作停止にする時の注意事項

電源供給構成が構成 1 および構成 2 の時、「パワー／モード制御レジスタ (CHIPCNT)」の AREGPD ビットを 1 に設定すると AREG の動作が停止します。AREG が動作停止していると、送受信動作が行えなくなりますのでご注意ください。誤って書き込みを行ってしまった場合は、SAIC101 をリセットする必要があります。詳しくは SAIC101 データシート「RAA730101 プログラマブル・ゲイン計装アンプ付き 16 ビット ΔΣ A/D コンバータ IC」(R02DS0014JJ) をご参照ください。

2.2.3 起動シーケンスのデータ転送

STARTUP レジスタの CPSOR ビットを 1 および SDCOR ビットを 1 とした場合、パワーオン・リセット直後にフラッシュ・メモリのデータが 256 バイト送信されますが、本 API では対応しておりませんのでご注意ください。

2.3 [SPI のみ] 使用上の注意

2.3.1 SPI モードの注意事項

SAIC101 は SPI モード 0 およびモード 3 で動作しますが、SAIC300・SAIC301・SAIC500・SAIC501・SAIC502 は SPI モード 3 のみ対応となります。SAIC101 と同一の SPI チャネル上に SAIC101 以外の SAIC を接続する際には SPI モードの設定にご注意ください。詳しくは各 SAIC のデータシートをご参照ください。

2.3.2 MCU 設定および SPI 制御の注意事項

MCU の「CPU と周辺クロック周波数」、SPI 通信制御の「ボー・レート設定値」および「通信制御方法[注1]」の組み合わせにより、正常な A/D 変換値を取得できないことがあります。正常に A/D 変換値を取得するために STATUS レジスタの読み込み時間とポーリング時間の合計値がオーバーサンプリングレートの 1 サンプリング期間内に収まるようにしてください。

「入力マルチプレクサ x（x = 1～5）A/D 変換設定レジスタ 3」の設定値が 2 以上の場合は、ご使用の CPU と周辺クロック周波数および SPI 通信ボー・レートの設定にご注意ください。

注 1：INT 端子割り込みを使用した制御、または STATUS レジスタ確認によるポーリング制御

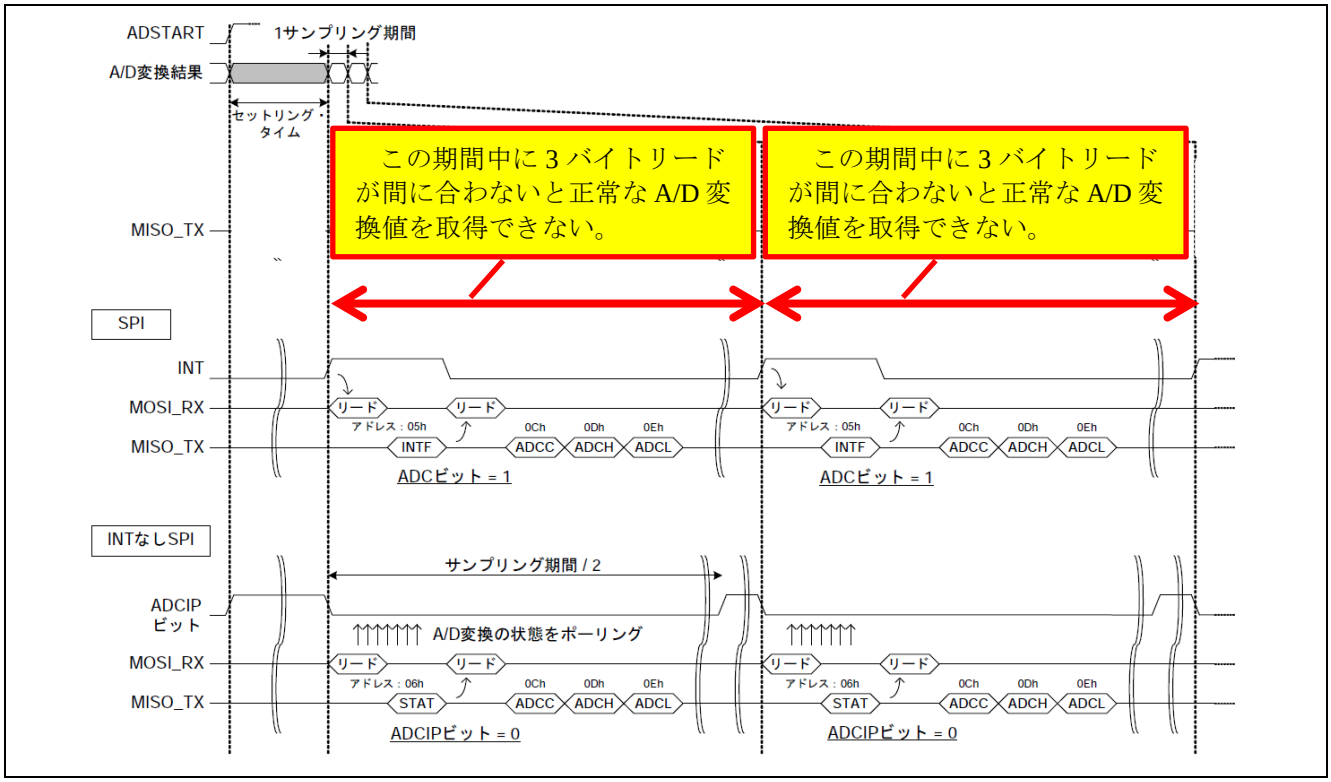


図 2.1 サンプリング期間と 3 バイトリード時間の関係

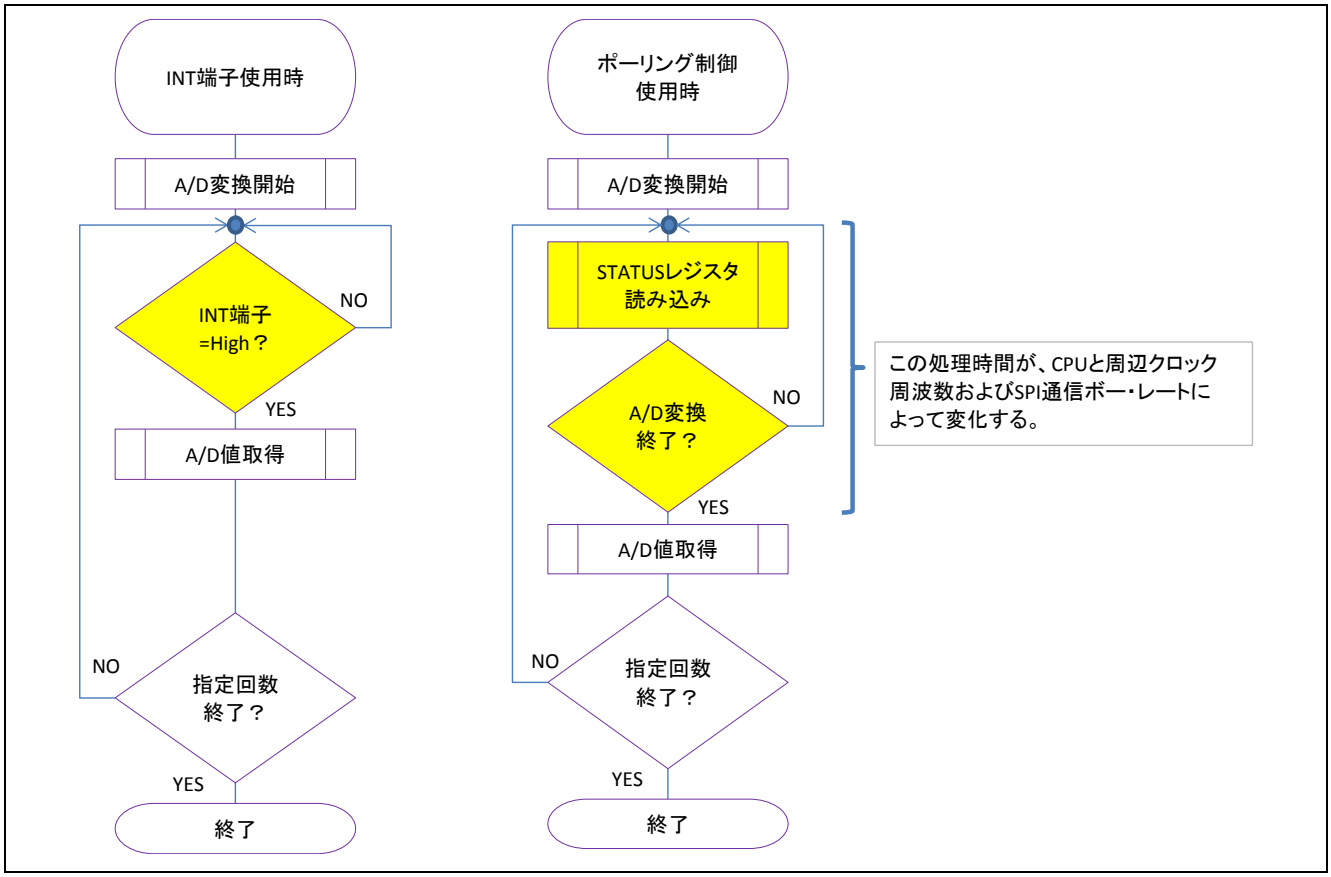


図 2.2 INT 端子使用時とポーリング制御使用時の処理の違い

3. API 関数一覧

本サンプルコードでは UART/SPI 制御毎に通信関連、フラッシュ・メモリ関連、A/D コンバータ関連、電源関連の API 関数を用意しています。各 API 関数を以下に示します。

3.1 UART 制御

3.1.1 通信関連

表 3-1 に UART 制御通信関連の API 関数一覧を示します。

表 3-1 UART 制御通信関連 API 関数一覧

関数名	概要
R_SAIC_UART_Init	SmartAnalog 初期化関数
R_SAIC_UART_Reset	SmartAnalog リセット関数
R_SAIC_UART_Read	UART 制御レジスタバイト読み出し関数
R_SAIC_UART_Write	UART 制御レジスタバイト書き込み関数
R_SAIC_UART_WriteVerify	UART 制御ベリファイ付きレジスタバイト書き込み関数
R_SAIC_UART_Negotiation	UART 制御通信設定ネゴシエーション関数。SAIC101 との通信設定を自動認識し、250kbps/Parity:Odd に設定する。
R_SAIC_UART_SAIC101 注	UART 制御 SAIC101 固有コマンド処理関数 以下の機能が実現可能。 レジスタバースト読み出し レジスタバースト書き込み

【注】 R_SAIC_UART_SAIC101 はレジスタアクセスの他にフラッシュ・メモリアクセスの機能も持ちます。フラッシュ・メモリアクセスの機能については 3.1.2 フラッシュ・メモリ関連をご参照ください。

3.1.2 フラッシュ・メモリ関連

表 3-2 に UART 制御フラッシュ・メモリ関連の API 関数一覧を示します。

表 3-2 UART 制御フラッシュ・メモリ関連 API 関数一覧

関数名	概要
R_SAIC_UART_SAIC101 注	UART 制御 SAIC101 固有コマンド処理関数 以下の機能が実現可能。 フラッシュ・メモリ読み出し フラッシュ・メモリ書き込み(01H、1FH 番地は書き込み禁止) フラッシュ・メモリ全消去 フラッシュ・メモリ・シャドウ領域レジスターコピー システム設定をバッファへコピー
R_SAIC_UART_FLASH_WRITE_01H	UART 制御フラッシュ・メモリ 01H 番地書き込み処理関数。01H 番地は電源に関する設定。
R_SAIC_UART_FLASH_WRITE_1FH	UART 制御フラッシュ・メモリ 1FH 番地書き込み処理関数。1FH 番地は起動時動作に関する設定。

【注】 R_SAIC_UART_SAIC101 はフラッシュ・メモリアクセスの他にレジスタアクセスの機能も持ちます。レジスタアクセスの機能については 3.1.1 通信関連をご参照ください。

3.1.3 A/D コンバータ関連

表 3-3 に UART 制御 A/D コンバータ関連の API 関数一覧を示します。

表 3-3 UART 制御 A/D コンバータ関連 API 関数一覧

関数名	概要
R_SAIC_UART_ADC_Start	UART 制御 A/D 変換開始処理関数
R_SAIC_UART_ADC_Stop	UART 制御 A/D 変換終了処理関数
R_SAIC_UART_ADC_GetResult	UART 制御 A/D 変換値取得関数。複数チャネル・複数回のデータを取得する。
R_SAIC_UART_ADC_GetResult_1Shot	UART 制御 A/D 変換値取得関数。単一チャネル・1 回のデータを取得する。
R_SAIC_UART_ADC_GetReceive	UART 制御 A/D 変換値受信データ取得関数
R_SAIC_UART_ADC_InitRegSet	UART 制御 A/D コンバータレジスタ初期設定関数

3.1.4 電源関連

表 3-4 に UART 制御電源関連の API 関数一覧を示します。

表 3-4 UART 制御電源関連 API 関数一覧

関数名	概要
R_SAIC_UART_AregOn	UART 制御 AREG オン設定関数
R_SAIC_UART_AregOff	UART 制御 AREG オフ設定関数
R_SAIC_UART_SbiasRegSet	UART 制御 SBIAS レジスタ設定関数
R_SAIC_UART_SbiasRegGet	UART 制御 SBIAS レジスタ取得関数
R_SAIC_UART_SleepModeOn	UART 制御スリープモード設定関数
R_SAIC_UART_SleepModeOff	UART 制御スリープモード解除関数

3.2 SPI 制御

3.2.1 通信関連

表 3-5 に SPI 制御通信関連の API 関数一覧を示します。

表 3-5 SPI 制御通信関連 API 関数一覧

関数名	概要
R_SAIC_SPI_Init	SmartAnalog 初期化関数
R_SAIC_SPI_Reset	SmartAnalog リセット関数
R_SAIC_SPI_Read	SPI 制御レジスタバイト読み出し関数
R_SAIC_SPI_Write	SPI 制御レジスタバイト書き込み関数
R_SAIC_SPI_WriteVerify	SPI 制御ベリファイ付きレジスタバイト書き込み関数
R_SAIC_SPI_ReadBit	SPI 制御レジスタビット読み出し関数
R_SAIC_SPI_WriteBit	SPI 制御レジスタビット書き込み関数
R_SAIC_SPI_WriteVerifyBit	SPI 制御ベリファイ付きレジスタビット書き込み関数
R_SAIC_SPI_CSEnable	SPI 制御用 CS 有効化処理関数
R_SAIC_SPI_CSCheck	SPI 制御用 CS チェック関数
R_SAIC_SPI_CSDisable	SPI 制御用 CS 無効化処理関数
R_SAIC_SPI_SAIC101 注	SPI 制御 SAIC101 固有コマンド処理関数 以下の機能が実現可能。 レジスタバースト読み出し レジスタバースト書き込み

【注】 R_SAIC_SPI_SAIC101 はレジスタアクセスの他にフラッシュ・メモリアクセスの機能も持ちます。フラッシュ・メモリアクセスの機能については 3.2.2 フラッシュ・メモリ関連をご参照ください。

備考 R_SAIC_SPI_SAIC101 以外の SPI 通信関連 API 関数は、Smart Analog IC101 の他に Smart Analog IC300, IC301, IC500, IC501, IC502 でも使用可能です。

3.2.2 フラッシュ・メモリ関連

表 3-6 に SPI 制御フラッシュ・メモリ関連の API 関数一覧を示します。

表 3-6 SPI 制御フラッシュ・メモリ関連 API 関数一覧

関数名	概要
R_SAIC_SPI_SAIC101 注	SPI 制御 IC101 固有コマンド処理関数 以下の機能が実現可能。 フラッシュ・メモリ読み出し フラッシュ・メモリ書き込み(01H、1FH 番地は書き込み禁止) フラッシュ・メモリ全消去 フラッシュ・メモリ・シャドウ領域レジスターコピー システム設定をバッファへコピー
R_SAIC_SPI_FLASH_WRITE_01H	SPI 制御フラッシュ・メモリ 01H 番地書き込み処理関数。01H 番地は電源に関する設定。
R_SAIC_SPI_FLASH_WRITE_1FH	SPI 制御フラッシュ・メモリ 1FH 番地書き込み処理関数。1FH 番地は起動時動作に関する設定。

【注】 R_SAIC_SPI_SAIC101 はフラッシュ・メモリアクセスの他にレジスタアクセスの機能も持ちます。レジスタアクセスの機能については 3.2.1 通信関連をご参照ください。

3.2.3 A/D コンバータ関連

表 3-7 に SPI 制御 A/D コンバータ関連の API 関数一覧を示します。

表 3-7　SPI 制御 A/D コンバータ関連 API 関数一覧

関数名	概要
R_SAIC_SPI_ADC_Start	SPI 制御 A/D 変換開始処理関数
R_SAIC_SPI_ADC_Stop	SPI 制御 A/D 変換終了処理関数
R_SAIC_SPI_ADC_GetResult	SPI 制御 A/D 変換値取得関数。複数チャネル・複数回のデータを取得する。
R_SAIC_SPI_ADC_GetResult_1Shot	SPI 制御 A/D 変換値取得関数。単一チャネル・1 回のデータを取得する。
R_SAIC_SPI_ADC_InitRegSet	SPI 制御 A/D コンバータレジスタ初期設定関数

3.2.4 電源関連

表 3-8 に SPI 制御電源関連の API 関数一覧を示します。

表 3-8　SPI 制御電源関連 API 関数一覧

関数名	概要
R_SAIC_SPI_AregOn	SPI 制御 AREG オン設定関数
R_SAIC_SPI_AregOff	SPI 制御 AREG オフ設定関数
R_SAIC_SPI_SbiasRegSet	SPI 制御 SBIAS レジスタ設定関数
R_SAIC_SPI_SbiasRegGet	SPI 制御 SBIAS レジスタ取得関数
R_SAIC_SPI_SleepModeOn	SPI 制御スリープモード設定関数
R_SAIC_SPI_SleepModeOff	SPI 制御スリープモード解除関数

4. API 共通定義

本章では各 API 関数で共通して使用する定義について示します。

4.1 API 関数共通の戻り値

本 API の関数は、一部関数を除き、共通した型でステータス値を返します。ユーザーアプリケーションでは、この返り値を判断し、正常であれば処理の続行、エラーであれば訂正処理を行う必要があります。

表 4-1 API 関数戻り値一覧 [UART/SPI 共通]

型名	マクロ名	定数値	内容
saic_status_t (uint8_t)	D_SAIC_OK	00H	正常終了
	D_SAIC_ERR_PARAM	01H	パラメータ・エラー
	D_SAIC_ERR_COM	02H	通信エラー（オーバラン・エラー、タイムアウト・エラー）
	D_SAIC_ERR_VERIFY	03H	ベリファイ・エラー

4.2 ユーザー環境依存設定用マクロ宣言

本 API ではユーザー環境、使用条件に依存する部分をマクロ宣言にて定義しています。ご使用の環境に応じて各定義を変更してください。

4.2.1 UART 通信使用時

(a) r_sa_uart_control_register.h

表 4-2 マクロ宣言一覧 1/2 [UART]

マクロ宣言	デフォルト設定値	入力範囲	内容
D_DEADLOCK_CNT	11000000L	uint32_t 注 1	通信中の Wait 時にデッドロックと判断するループ回数

【注 1】0 より大きな値を指定してください

表 4-3 マクロ宣言一覧 2/2 [UART]

マクロ宣言	内容
D_SAIC_FLASH_API_VALID	FLASH API 関数を有効にします。FLASH API を使用しない場合は無効にすることで RAM 容量を削減できます
D_UART_NEGOTIATION_250KBPS_PARITY_ODD	baudrate=250000bps, Parity=Odd 設定でのネゴシエーション処理を有効にします。使用しない場合は無効にしてください。
D_UART_NEGOTIATION_250KBPS_PARITY_EVEN	baudrate=250000bps, Parity=Even 設定でのネゴシエーション処理を有効にします。使用しない場合は無効にしてください。
D_UART_NEGOTIATION_250KBPS_PARITY_NONE	baudrate=250000bps, Parity=None 設定でのネゴシエーション処理を有効にします。使用しない場合は無効にしてください。
D_UART_NEGOTIATION_4800BPS_PARITY_ODD	baudrate=4800bps, Parity=Odd 設定でのネゴシエーション処理を有効にします。使用しない場合は無効にしてください。
D_UART_NEGOTIATION_4800BPS_PARITY_EVEN	baudrate=4800bps, Parity=Even 設定でのネゴシエーション処理を有効にします。使用しない場合は無効にしてください。
D_UART_NEGOTIATION_4800BPS_PARITY_NONE	baudrate=4800bps, Parity=None 設定でのネゴシエーション処理を有効にします。使用しない場合は無効にしてください。

(b) r_sa_uart_control_register_user.c

表 4-4 マクロ宣言一覧 [UART]

マクロ宣言	デフォルト設定値	入力範囲	内容
D_CPU_CLK_MHZ	24.0F	float	CPU と周辺クロック周波数の定義。注 1 注 2 API 内部で使用するソフトウェアウエイトの大まかな時間算出に使用します。単位:MHz
D_WAIT_PON_RST_TIME_MS	4.00F	float	パワーオンリセットの wait 時間の定義。注 1 リセット処理関数でパワーオンリセットを指定した場合の待ち時間の判定に使用します。単位:ms

【注 1】0 より大きな値を指定してください

【注 2】使用するマイコンの CPU クロックの設定値を指定してください

4.2.2 SPI 通信使用時

(a) r_sa_spi_control_register.h

表 4-5 マクロ宣言一覧 1/2 [SPI]

マクロ宣言	デフォルト設定値	入力範囲	内容
D_DEADLOCK_CNT	11000000L	uint32_t	通信中の Wait 時にデッドロックと判断するループ回数[注 1]
D_SPI_OPERATION	D_SPI_USE_INTERRUPT	D_SPI_USE_INTERRUPT または D_SPI_REGISTER_POLLING	通信モジュール 割り込み使用/不使用(ポーリング使用)選択定義

【注 1】0 より大きな値を指定してください

表 4-6 マクロ宣言一覧 2/2 [SPI]

マクロ宣言	内容
D_SAIC_FLASH_API_VALID	FLASH API 関数を有効にします。FLASH API を使用しない場合は定義しない事で無効となり、RAM 容量を削減できます

(b) r_sa_spi_control_register_user.c

表 4-7 マクロ宣言一覧 [SPI]

マクロ宣言	デフォルト設定値	入力範囲	内容
D_CPU_CLK_MHZ	24.0F	float	CPU と周辺クロック周波数の定義。[注 1][注 2] API 内部で使用するソフトウェアウエイトの大まかな時間算出に使用します。単位:MHz
D_WAIT_PON_RST_TIME_MS	4.00F	float	パワーオンリセットの wait 時間の定義。[注 1] リセット処理関数でパワーオンリセットを指定した場合の待ち時間の判定に使用します。単位:ms
D_WAIT_HARD_RESET_TIME_MS	0.01F	float	ハードリセットの wait 時間の定義。[注 1] リセット処理関数でハードリセットを指定した場合の待ち時間の判定に使用します。単位:ms

【注 1】0 より大きな値を指定してください

【注 2】使用するマイコンの CPU クロックの設定値を指定してください

4.3 マクロ宣言

本節では API で定義されているマクロ宣言について示します。

(a) r_sa_uart_control_register.h

表 4-8 マクロ宣言一覧 [UART]

マクロ宣言	値	内容
D_UART_USE_INTERRUPT	1U	通信モジュール 割り込み使用定義
D_UART_REGISTER_POLLING	2U	通信モジュール 割り込み不使用(ポーリング使用)定義
D_UART_OPERATION	D_UART_USE_INTERRUPT	通信モジュール 割り込み使用/不使用(ポーリング使用)選択定義

(b) r_sa_spi_control_register.h

表 4-9 マクロ宣言一覧 [SPI]

マクロ宣言	値	内容
D_SPI_USE_INTERRUPT	1U	通信モジュール 割り込み使用定義
D_SPI_REGISTER_POLLING	2U	通信モジュール 割り込み不使用(ポーリング使用)定義

(c) r_sa_uart_control_register.c

表 4-10 マクロ宣言一覧 [UART]

マクロ宣言名	値	内容
D_READ_MAX_SIZE	256U	最大受信データサイズ数判定用定義
D_COMMAND_LENGTH	3U	UART 応答パケットのバイト数判定用定義
D_BURST_MAX_SIZE	16U	SAIC101 バースト読み書き最大データサイズ判定用定義
D_REGISTER_MAX_ADDRESS	1FH	SAIC101 レジスタ最大アドレス判定用定義
D_FLASH_MAX_ADDRESS	FFH	SAIC101 フラッシュ・メモリ最大アドレス判定用定義
D_AD_DATA_LENGTH	3U	SAIC101 A/D 変換データレジスタサイズ判定用定義
D_FLASH_READ_MAX_SIZE	256U	SAIC101 フラッシュ・メモリリード最大データサイズ判定用定義
D_AUTO_ADC_MAX_RECEIVE_SIZE	10U	UART 自動 A/D 変換値受信バッファサイズ判定用定義
D_AUTO_ADC_REPLAY_MAX	5U	繰り返し処理回数判定用定義

(d) r_sa_spi_control_register.c

表 4-11 マクロ宣言一覧 [SPI]

マクロ宣言名	値	内容
D_BURST_MAX_SIZE	16U	SAIC101 のバースト読み書き最大データ長判定用定義
D_COMMAND_LENGTH	2U	レジスタバースト読み出し、レジスタバースト読み込み、フラッシュ読み出し、フラッシュ書き込み時のコマンドのバイト数判定用定義
D_FLASH_READ_MAX_SIZE	256U	SAIC101 フラッシュ・メモリリード最大データサイズ判定用定義
D_REGISTER_MAX_ADDRESS	1FH	SAIC101 レジスタ最大アドレス判定用定義
D_FLASH_MAX_ADDRESS	FFH	SAIC101 フラッシュ・メモリ最大アドレス判定用定義

4.4 型宣言

本節では API で定義されている独自型について示します。

(a) r_sa_uart_control_register.h / r_sa_spi_control_register.h

表 4-12 型宣言一覧 [UART / SPI]

型	定義	説明
saic_status_t	uint8_t	API 関数共通戻り値型

(b) r_sa_spi_control_register.c

表 4-13 型宣言一覧 [UART / SPI]

型	定義	説明
saic_func_bitRead_t	saic_status_t (*)(uint8_t saic_num, uint8_t *red_bit)	レジスタビット読み出し関数ポインタ

4.5 ユーザー変更必須列挙体

表 4-14・表 4-15 に示す列挙体は、4.9 で記載されている定数と共に用いられ、その設定に従って変更する必要があります。詳細は 4.9 を参照ください。

(a) r_sa_uart_control_register.h

表 4-14 シリアルモジュール情報格納グローバル変数指定用列挙体 [UART]

型名	デフォルト定義	内容
e_uart_ch_t	E_UART0	g_uart_serial_data_tbl で定義した配列の数に合わせて列挙体を定義してください。 定義名は g_uart_saic_data_tbl で用います。 デフォルト定義は一例で、g_uart_serial_data_tbl および g_uart_saic_data_tbl との対応が取れていれば任意の名前を付けることが可能です。
	E_UART1	
	E_UART2	
	E_UART3	
	E_UART4	
	E_UART5	
	E_UART6	
	E_UART_MAX	g_uart_serial_data_tbl[]の最大値判定用

(b) r_sa_spi_control_register.h

表 4-15 シリアルモジュール情報格納グローバル変数指定用列挙体 [SPI]

型名	デフォルト定義	内容
e_csi_ch_t	E_CSI00	g_spi_serial_data_tbl で定義した配列の数に合わせて列挙体を定義してください。 定義名は g_spi_saic_data_tbl で用います。 デフォルト定義は一例で、g_spi_serial_data_tbl および g_spi_saic_data_tbl との対応が取れていれば任意の名前を付けることが可能です。
	E_CSI01	
	E_CSI10	
	E_CSI11	
	E_CSI20	
	E_CSI21	
	E_CSI30	
	E_CSI31	
	E_CSI_MAX	g_spi_serial_data_tbl[]の最大値判定用

4.6 列挙体

本節では API で定義されている列挙体宣言について示します。

(a) r_sa_uart_control_register.h / r_sa_spi_control_register.h

表 4-16 SAIC 型名指定用列挙体 [UART / SPI]

型名	マクロ名	内容
e_saic_type_t	E_SAIC300	SAIC300 を指定
	E_SAIC301	SAIC301 を指定
	E_SAIC500	SAIC500 を指定
	E_SAIC501	SAIC501 を指定
	E_SAIC502	SAIC502 を指定
	E_SAIC101	SAIC101 を指定
	E_IC_TYPE_MAX	最大値判定用

表 4-17 リセット方法指定用列挙体 [UART / SP]

型名	マクロ名	内容
e_reset_process_t	E_SAIC_EXTERNAL_RESET	外部リセット(RESET 端子を L → H)
	E_SAIC_SPI_INTERNAL_RESET	内部リセット(SPI でリセット/リセット解除コマンド送信)
	E_SAIC_UART_INTERNAL_RESET	内部リセット(UART でリセット/リセット解除コマンド送信)
	E_SAIC_POWERON_RESET	POWER_ON リセット(電源投入からの待ち動作)
	E_SAIC_RESET_MAX	最大値判定用

表 4-18 SAIC101 固有コマンド指定用列挙体 [UART / SPI]

型名	マクロ名	内容
e_saic101_func_t	E_REGISTER_READ	レジスタ読み出し
	E_REGISTER_WRITE	レジスタ書き込み
	E_REGISTER_BURST_READ	レジスタバースト読み出し
	E_REGISTER_BURST_WRITE	レジスタバースト書き込み
	E_REGISTER_ALL_WRITE_FROM_FLASH	フラッシュ・シャドウ領域をレジスタへコピー
	E_BUFFER_REFRESH	フラッシュ・システム設定をバッファへコピー
	E_FLASH_READ	フラッシュ・メモリ読み出し
	E_FLASH_READ_1	フラッシュ・メモリ読み出しコマンド 1 回目
	E_FLASH_READ_2	フラッシュ・メモリ読み出しコマンド 2 回目
	E_FLASH_WRITE	フラッシュ・メモリ書き込み
	E_FLASH_ALL_ERASE	フラッシュ・メモリ全消去
	E_FLASH_WRITE_VERIFY	フラッシュ・メモリベリファイ書き込み

表 4-19 A/D コンバータ入力モード指定用列挙体 [UART / SPI]

型名	マクロ名	内容
e_adc_mode_t	E_ADC_DIFF	差動入力モード
	E_ADC_SINGLE	シングルエンド入力モード

表 4-20　PGIA のゲイン設定指定用列挙体 [UART / SPI]

型名	マクロ名	値	内容
e_adc_gain_t	E_ADC_GAIN_1_1_1	00H	GSET1 = x1, GSET2 = x1, Total = x 1
	E_ADC_GAIN_2_1_2	04H	GSET1 = x2, GSET2 = x1, Total = x 2
	E_ADC_GAIN_3_1_3	08H	GSET1 = x3, GSET2 = x1, Total = x 3
	E_ADC_GAIN_4_1_4	0CH	GSET1 = x4, GSET2 = x1, Total = x 4
	E_ADC_GAIN_8_1_8	10H	GSET1 = x8, GSET2 = x1, Total = x 8
	E_ADC_GAIN_1_2_2	01H	GSET1 = x1, GSET2 = x2, Total = x 2
	E_ADC_GAIN_2_2_4	05H	GSET1 = x2, GSET2 = x2, Total = x 4
	E_ADC_GAIN_3_2_6	09H	GSET1 = x3, GSET2 = x2, Total = x 6
	E_ADC_GAIN_4_2_8	0DH	GSET1 = x4, GSET2 = x2, Total = x 8
	E_ADC_GAIN_8_2_16	11H	GSET1 = x8, GSET2 = x2, Total = x16
	E_ADC_GAIN_1_4_4	02H	GSET1 = x1, GSET2 = x4, Total = x 4
	E_ADC_GAIN_2_4_8	06H	GSET1 = x2, GSET2 = x4, Total = x 8
	E_ADC_GAIN_3_4_12	0AH	GSET1 = x3, GSET2 = x4, Total = x12
	E_ADC_GAIN_4_4_16	0EH	GSET1 = x4, GSET2 = x4, Total = x16
	E_ADC_GAIN_8_4_32	12H	GSET1 = x8, GSET2 = x4, Total = x32
	E_ADC_GAIN_1_8_8	03H	GSET1 = x1, GSET2 = x8, Total = x 8
	E_ADC_GAIN_2_8_16	07H	GSET1 = x2, GSET2 = x8, Total = x16
	E_ADC_GAIN_3_8_24	0BH	GSET1 = x3, GSET2 = x8, Total = x24
	E_ADC_GAIN_4_8_32	0FH	GSET1 = x4, GSET2 = x8, Total = x32

表 4-21　オーバーサンプリング・レート指定用列挙体 [UART / SPI]

型名	マクロ名	値	内容
e_adc_osr_t	E_ADC_OSR_64	00H	15625.000 [sps]
	E_ADC_OSR_128	01H	7812.500 [sps]
	E_ADC_OSR_256	02H	3906.250 [sps]
	E_ADC_OSR_512	03H	1953.125 [sps]
	E_ADC_OSR_1024	04H	976.563 [sps]
	E_ADC_OSR_2048	05H	488.281 [sps]
	E_ADC_OSR_MAX	06H	最大値判定用

表 4-22　DC オフセット指定用列挙体 [UART / SPI]

型名	マクロ名	値	内容
e_adc_offset_t	E_ADC_OFFSET_164p06	1FH	164.06/GSET1 [mV]
	E_ADC_OFFSET_153p13	1EH	153.13/GSET1 [mV]
	E_ADC_OFFSET_142p19	1DH	142.19/GSET1 [mV]
	E_ADC_OFFSET_131p25	1CH	131.25/GSET1 [mV]
	E_ADC_OFFSET_120p31	1BH	120.31/GSET1 [mV]
	E_ADC_OFFSET_109p38	1AH	109.38/GSET1 [mV]
	E_ADC_OFFSET_98p44	19H	98.44/GSET1 [mV]
	E_ADC_OFFSET_87p50	18H	87.50/GSET1 [mV]
	E_ADC_OFFSET_76p56	17H	76.56/GSET1 [mV]
	E_ADC_OFFSET_65p63	16H	65.63/GSET1 [mV]
	E_ADC_OFFSET_54p69	15H	54.69/GSET1 [mV]
	E_ADC_OFFSET_43p75	14H	43.75/GSET1 [mV]
	E_ADC_OFFSET_32p81	13H	32.81/GSET1 [mV]
	E_ADC_OFFSET_21p88	12H	21.88/GSET1 [mV]
	E_ADC_OFFSET_10p94	11H	10.94/GSET1 [mV]
	E_ADC_OFFSET_0p00	10H	0.00/GSET1 [mV]
	E_ADC_OFFSET_M10p94	0FH	-10.94/GSET1 [mV]
	E_ADC_OFFSET_M21p88	0EH	-21.88/GSET1 [mV]
	E_ADC_OFFSET_M32p81	0DH	-32.81/GSET1 [mV]
	E_ADC_OFFSET_M43p75	0CH	-43.75/GSET1 [mV]
	E_ADC_OFFSET_M54p69	0BH	-54.69/GSET1 [mV]
	E_ADC_OFFSET_M65p63	0AH	-65.63/GSET1 [mV]
	E_ADC_OFFSET_M76p56	09H	-76.56/GSET1 [mV]
	E_ADC_OFFSET_M87p50	08H	-87.50/GSET1 [mV]
	E_ADC_OFFSET_M98p44	07H	-98.44/GSET1 [mV]
	E_ADC_OFFSET_M109p38	06H	-109.38/GSET1 [mV]
	E_ADC_OFFSET_M120p31	05H	-120.31/GSET1 [mV]
	E_ADC_OFFSET_M131p25	04H	-131.25/GSET1 [mV]
	E_ADC_OFFSET_M142p19	03H	-142.19/GSET1 [mV]
	E_ADC_OFFSET_M153p13	02H	-153.13/GSET1 [mV]
	E_ADC_OFFSET_M164p06	01H	-164.06/GSET1 [mV]
	E_ADC_OFFSET_M175p00	00H	-175.00/GSET1 [mV]

表 4-23　A/D 変換チャネル番号指定用列挙体 [UART / SPI]

型名	マクロ名	値	内容
e_adc_ch_t	E_ADC_CH1	00H	Ch1
	E_ADC_CH2	01H	Ch2
	E_ADC_CH3	02H	Ch3
	E_ADC_CH4	03H	Ch4
	E_ADC_CH5	04H	Ch5(温度センサ)

表 4-24　A/D 変換 ON/OFF 指定用列挙体 [UART / SPI]

型名	マクロ名	値	内容
e_adc_onoff_t	E_ADC_OFF	01H	A/D 変換を禁止
	E_ADC_ON	00H	A/D 変換を許可

表 4-25　ADSTART ビット指定用列挙体 [UART / SPI]

型名	マクロ名	値	内容
e_adc_start_t	E_ADC_STOP	00H	A/D 変換を停止
	E_ADC_START	01H	A/D 変換を開始

表 4-26　SBIAS 出力電圧設定指定用列挙体 [UART / SPI]

型名	マクロ名	値	内容
e_sbias_t	E_ADC_SBIAS_0p0	FFH	オフ(SENSEPD=1)
	E_ADC_SBIAS_1p2	00H	1.2V
	E_ADC_SBIAS_1p3	01H	1.3V
	E_ADC_SBIAS_1p4	02H	1.4V
	E_ADC_SBIAS_1p5	03H	1.5V
	E_ADC_SBIAS_1p6	04H	1.6V
	E_ADC_SBIAS_1p7	05H	1.7V
	E_ADC_SBIAS_1p8	06H	1.8V
	E_ADC_SBIAS_1p9	07H	1.9V
	E_ADC_SBIAS_2p0	08H	2.0V
	E_ADC_SBIAS_2p1	09H	2.1V
	E_ADC_SBIAS_2p2	0AH	2.2V
	E_ADC_SBIAS_MAX	0BH	最大値判定用

(b)　r_sa_uart_control_register.h

表 4-27　受信パケット判定用列挙体 [UART]

型名	マクロ名	内容
get_response_state_t	E_GET_HEADER	ヘッダ受信
	E_HEADER_DECODE	ヘッダデコード
	E_TYPE1	TYPE1 応答処理
	E_TYPE2	TYPE2 応答処理
	E_TYPE3	TYPE3 応答処理
	E_END	終了

表 4-28　UART 通信設定用 [UART]

型名	マクロ名	内容
e_uart_setting_t	E_UART_4800bps_None	baudrate=4800bps, Parity=None
	E_UART_4800bps_Odd	baudrate=4800bps, Parity=Odd
	E_UART_4800bps_Even	baudrate=4800bps, Parity=Even
	E_UART_250kbps_None	baudrate=250000bps, Parity=None
	E_UART_250kbps_Odd	baudrate=250000bps, Parity=Odd
	E_UART_250kbps_Even	baudrate=250000bps, Parity=Even

(c) r_sa_spi_control_register.c

表 4-29 ReadWrite 指定用列挙体 [C ソースファイル内で定義、SPI]

型名	マクロ名	値	内容
e_rw_t	E_READ	00H	リード用
	E_WRITE	01H	ライト用

4.7 構造体

本節では API で定義されている構造体宣言について示します。

(a) r_sa_uart_control_register.h / r_sa_spi_control_register.h

表 4-30 A/D 変換情報格納変数用構造体 [UART / SPI]

構造体型名	saic101_adc_t		
概要	SAIC101 の A/D 変換情報格納変数用構造体		
メンバ変数	型	名称	内容
	e_adc_onoff_t	onoff	A/D 変換 ON/OFF 指定用列挙体変数
	e_adc_mode_t	input_mode	A/D コンバータ入力モード指定用列挙体変数
	e_adc_offset_t	offset	DC オフセット指定用列挙体変数
	e_adc_osr_t	over_sampling_rate	オーバーサンプリング・レート指定用列挙体変数
	e_adc_gain_t	gain	PGIA のゲイン設定指定用列挙体変数
	uint8_t	count	AUTOSCAN の 1 サイクルにおける A/D 変換回数指定

表 4-31 バイト操作関数用構造体 [UART / SPI]

構造体型名	saic_data_t		
概要	バイト操作関数用構造体		
メンバ変数	型	名称	内容
	uint8_t	address	SAIC 制御レジスタのアドレス
	uint8_t	data	SAIC 制御レジスタのデータ

表 4-32 ビット操作関数用構造体 [UART / SPI]

構造体型名	saic_data_bit_t		
概要	ビット操作関数用構造体		
メンバ変数	型	名称	内容
	uint8_t	address	SAIC 制御レジスタのアドレス
	uint8_t	bit_number	SAIC 制御レジスタのビット番号(0～7)
	uint8_t	data	SAIC 制御レジスタのビット・データ(0/1)

(b) r_sa_uart_control_register.h

表 4-33 SAIC 情報格納用構造体 [UART]

構造体型名	uart_saic_t		
概要	SAIC の情報格納変数用構造体 (SA type、UARTch)		
メンバ変数	型	名称	内容
	e_uart_ch_t	uart_ch	UART ch 番号
	e_saic_type_t	sa_type	SAIC 型名

表 4-34　シリアル情報格納用構造体 (割り込み使用時) [UART]

構造体型名	uart_serial_t		
概要	シリアルの情報格納変数用構造体 (開始、停止、送信、受信、その他関数のポインタを格納する)		
メンバ変数	型	名称	内容
	void (*)(void)	UART_Start	UARTxx_Start 関数のポインタ
	void (*)(void)	UART_Stop	UARTxx_Stop 関数のポインタ
	MD_STATUS (*)(uint8_t * const rx_buf, uint16_t rx_num)	UART_Receive	UARTxx_Receive 関数のポインタ
	MD_STATUS (*)(uint8_t * const tx_buf, uint16_t tx_num)	UART_Send	UARTxx_Send 関数のポインタ
	uint8_t (*)(uint8_t *packet_data, uint8_t rx_buffer[], uint16_t read_pos)	UART_GetHeader	UARTxx_GetHeader 関数のポインタ
	uint8_t (*)(uint16_t rx_cnt)	UART_Getdata	UARTxx_Getdata 関数のポインタ
	void (*)(uint8_t setting)	UART_SettingChange	UARTxx_SettingChange 関数のポインタ

【注】 xx は SAIC との通信に使用するシリアル ch 番号となります

表 4-35　リセット情報格納用構造体 [UART]

構造体型名	uart_reset_t		
概要	SAIC の RESET 情報変数(リセット処理、RESET ポートに接続しているポートのアドレス、対応するビット番号、wait 用 NOP 実行回数、uart_saic_t 変数番号)		
メンバ変数	型	名称	内容
	e_reset_process_t	process	SAIC のリセット方法を指定
	uint8_t *	p_reset_addr	RESET 端子に接続しているポートレジスタのアドレス
	uint8_t	reset_bit_num	RESET 端子に接続しているポートレジスタのビット番号
	uint32_t	nop_cnt	wait する場合の NOP 命令カウント数
	uint8_t	uart_saic_num	SA 格納変数バッファ番号　※UART でリセットする場合などに使用。リセット端子共通などの場合は未指定とする

(c)　r_sa_spi_control_register.h

表 4-36　SAIC 情報格納用構造体 [SPI]

構造体型名	spi_saic_t		
概要	SAIC の情報格納変数用構造体 (SA type、CSIch、CS ポートに接続しているポートのアドレス、対応するビット番号、INT 端子ある場合は接続しているポートのアドレス、対応するビット番号)		
メンバ変数	型	名称	内容
	e_csi_ch_t	csi_ch	CSI ch 番号
	e_saic_type_t	sa_type	SAIC 型名
	uint8_t *	p_cs_addr	CS 端子に接続しているポートレジスタのアドレス
	uint8_t	cs_bit_num	CS 端子に接続しているポートレジスタのビット番号
	uint8_t *	p_int_addr	INT 端子に接続しているポートレジスタのアドレス
	uint8_t	int_bit_num	INT 端子に接続しているポートレジスタのビット番号

【注】 xx は SAIC との通信に使用するシリアル ch 番号となります

表 4-37　シリアル情報格納用構造体 (割り込み使用時) [SPI]

構造体型名	spi_serial_t		
概要	シリアルの情報格納変数用構造体 (開始、停止、R/W 実行関数のポインタを格納する)		
メンバ変数	型	名称	内容
	void (*)(void)	CSI_Start	CSIxx_Start 関数のポインタ
	void (*)(void)	CSI_Stop	CSIxx_Stop 関数のポインタ
	MD_STATUS (*)(uint8_t * const tx_buf, uint16_t tx_num, uint8_t * const rx_buf)	CSI_Send_Receive	CSIxx_Send_Receive 関数のポインタ

【注】 xx は SAIC との通信に使用するシリアル ch 番号となります

表 4-38　シリアル情報格納用構造体 (割り込み不使用(ポーリング使用)時) [SPI]

構造体型名	spi_serial_t		
概要	シリアルの情報格納変数用構造体 (開始、停止、処理に使用する各フラグを登録する)		
メンバ変数	型	名称	内容
	uint8_t *	p_trans_reg_addr	CSI ch に対応する送信データレジスタのアドレス
	uint8_t *	p_recv_reg_addr	CSI ch に対応する受信データレジスタのアドレス
	uint8_t *	p_csiif_addr	CSI ch に対応する割り込み要求フラグレジスタのアドレス
	uint8_t	csiif_bit_num	CSI ch に対応する割り込み要求フラグのビット番号
	uint16_t *	p_ssr_addr	CSI ch に対応する SSR レジスタのアドレス
	uint16_t *	p_sir_addr	CSI ch に対応する SIR レジスタのアドレス
	void (*)(void)	CSI_Start	CSIxx_Start 関数のポインタ
	void (*)(void)	CSI_Stop	CSIxx_Stop 関数のポインタ

表 4-39　リセット情報格納用構造体 [SPI]

構造体型名	spi_reset_t		
概要	SAIC の RESET 情報変数(リセット処理、RESET ポートに接続しているポートのアドレス、対応するビット番号、wait 用 NOP 実行回数、spi_saic_t 変数番号)		
メンバ変数	型	名称	内容
	e_reset_process_t	process	SAIC のリセット方法を指定未使用
	uint8_t *	p_reset_addr	RESET 端子に接続しているポートレジスタのアドレス
	uint8_t	reset_bit_num	RESET 端子に接続しているポートレジスタのビット番号
	uint32_t	nop_cnt	wait する場合の NOP 命令カウント数
	uint8_t	spi_saic_num	SA 格納変数バッファ番号　※SPI でリセットする場合などに使用。リセット端子共通などの場合は未指定とする

4.8 共用体

本節では API で定義されている共用体宣言について示します。

(a) r_sa_uart_control_register.h / r_sa_spi_control_register.h

表 4-40 SAIC101 のレジスタ情報格納変数用共用体定義 [UART / SPI]

構造体型名	uni_reg_t			
概要	レジスタ特定ビット値の取得および更新用			
メンバ変数	型	ビット数	名称	内容
	uint8_t	8	BYTE	8 ビットアクセス
	CHIPCNT_BIT	1 (下位)	slp	Sleep
		1	senspd	SENSPD
		2	-	未使用
		1	aregpd	AREGPD
		1	psthru	PSTHRU
		2 (上位)	-	未使用
	VSBIAS_BIT	4	sbias	SBIAS
		4	-	未使用
	INTFLAG_BIT	1	fr	FR
		1	fw	FW
		1	fae	FAE
		1	raw	RAW
		1	adc	ADC
		3	-	未使用
	STATUS_BIT	1	-	未使用
		1	fwip	FWIP
		1	faeip	FAEIP
		1	rawip	RAWIP
		1	adcip	ADCIP
		3	-	未使用
	ADCCNT_BIT	5	ainxadc	OnOff
		2	-	未使用
		1	adstart	ADSTART
	CHxCNT1_BIT	5	offset	offset
		2	-	未使用
		1	input_mode	AINSEL
	CHxCNT2_BIT	5	gain	GAIN
		3	osr	OSR
	CHxCNT3_BIT	5	low	下位 5 ビット
		3	high	上位 3 ビット
	STARTUP_BIT	1	cpsor	CPSOR
		1	sdcor	SDCOR
		2	-	未使用
		1	tglsm	TGLSM
		3	-	未使用

表 4-41　UART 通信関数用共用体(WORD アクセス用) [UART / SPI]

構造体型名	uni_adc_reg_t			
概要	WORD アクセス用			
メンバ変数	型	ビット数	名称	内容
	uint16_t	16	WORD	16 ビットアクセス
	BYTE	8	low	下位 8 ビットアクセス
		8	high	上位 8 ビットアクセス

表 4-42　A/D コンバータ関数用共用体 [UART / SPI]

構造体型名	uni_adcc_t			
概要	SAIC101 の ADCC 情報格納変数用ビットフィールド共用体			
メンバ変数	型	ビット数	名称	内容
	uint8_t	8	BYTE	ADCC レジスタ値
	BIT	4 (下位)	sum	チェックサムの値
		1	overflow	A/D 変換結果に対するオーバーフロー・フラグ
		3 (上位)	ch	変換結果に対応するチャネル番号

(b)　r_sa_uart_control_register.h

表 4-43　UART 通信関数用共用体(受信パケットデータ) [UART]

構造体型名	rcv_packet_t			
概要	受信データのパケット解析用			
メンバ変数	型	ビット数	名称	内容
	uint8_t	8	BYTE	8 ビットアクセス
	HEADER	5 (下位)	data	Data
		3 (上位)	type	type
	TYPE1	5	length	length
		3	type	type
	TYPE2	4	check_sum	チェックサムの値
		1	overflow	A/D 変換結果に対するオーバーフロー・フラグ
		3	ad_ch	変換結果に対応するチャネル番号
	TYPE3	1	f0	Flash Read 準備完了
		1	f1	Flash Write（FW フラグ）
		1	f2	Flash All Erase（FAE フラグ）
		1	f3	Register All Write（RAW フラグ）
		1	f4	前回の受信でパリティ・エラー発生（PE フラグ）
		3	type	type

(c) r_sa_uart_control_register.c / r_sa_spi_control_register.c

表 4-44 UART 通信関数用共用体(BYTE アクセス用) [UART/SPI]

構造体型名	uni_sum_t			
概要	BYTE アクセス用			
メンバ変数	型	ビット数	名称	内容
	uint8_t	8	BYTE	8 ビットアクセス
	BIT	4 (下位)	low	下位 4 ビット
		4 (上位)	high	上位 4 ビット

4.9 ユーザー環境依存設定グローバル定数

本節で示す定数を変更することにより、ユーザーが使用するハードウェア構成(シリアルモジュールや I/O ポートなどのインターフェース)に対応することができます。

4.9.1 UART を用いる場合のユーザー環境依存設定グローバル定数

(a) シリアルモジュール情報格納グローバル定数(UART)

定数名 g_uart_serial_data_tbl

宣言 const uart_serial_t g_uart_serial_data_tbl[] = { *定数値* }

説明 g_uart_serial_data_tbl は uart_serial_t 構造体の配列です。シリアル通信を行うための関数を登録します。配列数に応じてシリアルモジュール情報格納グローバル変数指定用列挙体 e_uart_ch_t を指定する必要があります。

uart_serial_t 構造体の設定

型	ユーザー設定	例
void (*)(void)	コード生成で生成した UART 開始関数名を記述	R_UART1_Start
void (*)(void)	コード生成で生成した UART 停止関数名を記述	R_UART1_Stop
MD_STATUS (*)(uint8_t * const rx_buf, uint16_t rx_num)	コード生成で生成した UART 受信関数名を記述	R_UART1_Receive
MD_STATUS (*)(uint8_t * const tx_buf, uint16_t tx_num)	コード生成で生成した UART 送信関数名を記述	R_UART1_Send
uint8_t (*)(uint8_t *packet_data, uint8_t rx_buffer[], uint16_t read_pos)	サンプルコードに含まれるヘッダデータ取得関数名を記述	R_UART1_GetHeader
uint8_t (*)(uint16_t rx_cnt)	サンプルコードに含まれる指定データ数受信完了チェック関数名を記述	R_UART1_Getdata
void (*)(uint8_t setting)	サンプルコードに含まれる UART 設定変更関数名を記述	R_UART1_SettingChange

(b) SAIC 情報格納グローバル定数(UART)

定数名　g_uart_saic_data_tbl

宣言　const uart_saic_t g_uart_saic_data_tbl[] = { *定数値* } ;

説明　uart_saic_t 構造体の配列。接続された SAIC との通信手段を指定します。シリアルモジュール情報格納グローバル変数指定用列挙体 e_uart_ch_t と、SAIC 型名指定用列挙体 e_saic_type_t を結び付け、この配列の添え字が SAIC 番号となります。

uart_saic_t 構造体の設定

型	ユーザー設定	例
e_uart_ch_t	シリアルモジュール情報格納グローバル変数指定用列挙体 e_uart_ch_t から該当する要素を選び記述	E_UART1
e_saic_type_t	SAIC 型名指定用列挙体 e_saic_type_t から該当する要素を選び記述	E_SAIC101

(c) RESET 情報格納グローバル定数(UART)

定数名　g_uart_reset_data_tbl

宣言　const uart_saic_t g_uart_saic_data_tbl[] = { *定数値* } ;

説明　uart_reset_t 構造体の配列。接続された各 SAIC のリセット方法を指定します。
複数 SAIC を接続する場合、通常 SAIC の個数分の配列が必要となりますが、パワーオンリセットする IC に関しては 1 つのみの記述で問題ありません。

uart_reset_t 構造体の設定

型	ユーザー設定	例
e_reset_process_t	SAIC リセット方法を記述	E_SAIC_POWERON_RESET
uint8_t *	リセット方法が外部リセットの場合、RESET 端子に接続しているポートレジスタのアドレスを 記述。それ以外の場合は NULL を記述	NULL
uint8_t	リセット方法が外部リセットの場合、RESET 端子に接続しているポートレジスタのビット番号を記述。それ以外の場合は 0 を記述	0U
uint32_t	wait する場合の NOP()の回数	D_PON_RST_NOP_CNT
uint8_t	内部リセット処理使用時の SAIC 番号。内部リセット処理以外の時は現在の API では使用していませんが、SAIC 番号を指定してください。(API Builder SAIC101 で使用しています。)	0U

(d) 記述例

- ケース 1
  SAIC101 が UART1 に接続されている。リセット方法にパワーオンリセットを指定。

```
r_sa_uart_control_register.h
typedef enum
{
    E_SAIC300 = 0x00U,                      /* SAIC300     */
    E_SAIC301,                             /* SAIC301     */
    E_SAIC500,                             /* SAIC500     */
    E_SAIC501,                             /* SAIC501     */
    E_SAIC502,                             /* SAIC502     */
    E_SAIC101,                             /* SAIC101     */
    E_IC_TYPE_MAX,                          /* 最大値判定用 */
} e_saic_type_t;

typedef enum
{
    E_UART0 = 0x00U,                /* UART0 */
    E_UART1,                       /* UART1 */
    E_UART2,                       /* UART2 */
    E_UART3,                       /* UART3 */
    E_UART_MAX,                     /* 最大値判定用 */
} e_uart_ch_t;

r_sa_uart_control_register_user.c
const uart_serial_t g_uart_serial_data_tbl[] =
{
    { NULL,         NULL,        NULL,           NULL,        NULL,            NULL,          NULL,               },
    { R_UART1_Start, R_UART1_Stop, R_UART1_Receive, R_UART1_Send, R_UART1_GetHeader, R_UART1_Getdata, R_UART1_SettingChange, },
};

const uart_saic_t g_uart_saic_data_tbl[] =
{
//  { UART_ch,    sa_type,      }, /* format              */
    { E_UART1,    E_SAIC101,    }, /* SAIC 番号=0 の SAIC 情報 */
};

const uart_reset_t g_uart_reset_data_tbl[] =
{
    //process,                Port address, Bit num, nop_cnt,              uart_saic_t 番号, }, /* format          */
    { E_SAIC_POWERON_RESET,      NULL,        0U,     D_PON_RST_NOP_CNT,     0U,           }, /* 1 番目の RESET 処理 */
};
```

- ケース 2

  SAIC101 が 4 つあり、UART0, 1, 2, 3 に接続されている。リセット方法にパワーオンリセットを指定。

```
r_sa_uart_control_register.h
typedef enum
{
    E_SAIC300 = 0x00U,                      /* SAIC300     */
    E_SAIC301,                             /* SAIC301     */
    E_SAIC500,                             /* SAIC500     */
    E_SAIC501,                             /* SAIC501     */
    E_SAIC502,                             /* SAIC502     */
    E_SAIC101,                             /* SAIC101     */
    E_IC_TYPE_MAX,                          /* 最大値判定用 */
} e_saic_type_t;

typedef enum
{
    E_UART0 = 0x00U,                /* UART0 */
    E_UART1,                       /* UART1 */
    E_UART2,                       /* UART2 */
    E_UART3,                       /* UART3 */
    E_UART_MAX,                     /* 最大値判定用 */
} e_uart_ch_t;

r_sa_uart_control_register_user.c
const uart_serial_t g_uart_serial_data_tbl[] =
{
    { R_UART0_Start, R_UART0_Stop, R_UART0_Receive, R_UART0_Send, R_UART0_GetHeader, R_UART0_Getdata, R_UART0_SettingChange, },
    { R_UART1_Start, R_UART1_Stop, R_UART1_Receive, R_UART1_Send, R_UART1_GetHeader, R_UART1_Getdata, R_UART1_SettingChange, },
    { R_UART2_Start, R_UART2_Stop, R_UART2_Receive, R_UART2_Send, R_UART2_GetHeader, R_UART2_Getdata, R_UART2_SettingChange, },
    { R_UART3_Start, R_UART3_Stop, R_UART3_Receive, R_UART3_Send, R_UART3_GetHeader, R_UART3_Getdata, R_UART3_SettingChange, },
};

const uart_saic_t g_uart_saic_data_tbl[] =
{
//  { UART_ch,    sa_type,      }, /* format             */
    { E_UART0,    E_SAIC101,    }, /* SAIC 番号=0 の SAIC 情報 */
    { E_UART1,    E_SAIC101,    }, /* SAIC 番号=1 の SAIC 情報 */
    { E_UART2,    E_SAIC101,    }, /* SAIC 番号=2 の SAIC 情報 */
    { E_UART3,    E_SAIC101,    }, /* SAIC 番号=3 の SAIC 情報 */
};


const uart_reset_t g_uart_reset_data_tbl[] =
{
    //process,                Port address, Bit num, nop_cnt,             uart_saic_t 番号, }, /* format          */
    { E_SAIC_POWERON_RESET,      NULL,        0U,     D_PON_RST_NOP_CNT,     3U,           }, /* 1 番目の RESET 処理 */
};
```

4.9.2 SPI を用いる場合のユーザー環境依存設定グローバル定数

(a) シリアルモジュール情報格納グローバル定数(SPI)

定数名　g_spi_serial_data_tbl

宣言　const spi_serial_t g_spi_serial_data_tbl[] = { *定数値* }

説明　spi_serial_t 構造体の配列。シリアル通信を行うための関数/レジスタアドレスを登録します。ユーザー環境依存設定用マクロ宣言 D_SPI_OPERATION の値により、spi_serial_t 型は内容が変化します。配列数に応じてシリアルモジュール情報格納グローバル変数指定用列挙体 e_csi_ch_t を指定する必要があります。

spi_serial_t 構造体の設定

割り込み関数使用時

型	ユーザー設定	例
void (*)(void)	コード生成で生成した SPI 開始関数名を記述	R_CSI10_Start
void (*)(void)	コード生成で生成した SPI 停止関数名を記述	R_CSI10_Stop
MD_STATUS (*)(uint8_t * const tx_buf, uint16_t tx_num, uint8_t * const rx_buf)	コード生成で生成したSPI送受信関数名を記述	R_CSI10_Send_Receive

レジスタポーリング使用時

型	ユーザー設定	例
uint8_t *	CSI ch に対応する送信データレジスタのアドレスを記述	&SIO10
uint8_t *	CSI ch に対応する受信データレジスタのアドレスを記述	&SIO10
uint8_t *	CSI chに対応する割り込み要求フラグレジスタのアドレスを記述	&IF1L
uint8_t	CSI ch に対応する割り込み要求フラグのビット番号を記述	1U
uint16_t *	CSI ch に対応する SMR レジスタのアドレスを記述	(uint16_t *)&SSR02
uint16_t *	CSI ch に対応する SIR レジスタのアドレスを記述	(uint16_t *)&SIR02
void (*)(void)	コード生成で生成した SPI 開始関数名を記述	R_CSI10_MaskStart
void (*)(void)	コード生成で生成した SPI 停止関数名を記述	R_CSI10_Stop

(b) SAIC 情報格納グローバル定数(SPI)

定数名　g_spi_saic_data_tbl

宣言　const spi_saic_t g_spi_saic_data_tbl[] = { *定数値* } ;

説明　spi_saic_t 構造体の配列。接続された SAIC との通信手段を指定します。シリアルモジュール情報格納グローバル変数指定用列挙体 spi_saic_t と、SAIC 型名指定用列挙体 e_saic_type_t を結び付けます。この配列の添え字が SAIC 番号となります。

spi_saic_t 構造体の設定

型	ユーザー設定	例
e_csi_ch_t	CSI ch 番号を記述	E_CSI10
e_saic_type_t	SAIC 型名を記述	E_SAIC101
uint8_t *	CS 端子に接続しているポートレジスタのアドレスを記述	&P0
uint8_t	CS 端子に接続しているポートレジスタのビット番号を記述	6U
uint8_t *	INT 端子に接続しているポートレジスタのアドレスを記述	&P0
uint8_t	INT 端子に接続しているポートレジスタのビット番号を記述	7U

(c) RESET 情報格納グローバル定数(SPI)

定数名　g_spi_reset_data_tbl

宣言　const spi_reset_t g_spi_reset_data_tbl[] = { *定数値* } ;

説明　spi_reset_t 構造体の配列。接続された各 SAIC のリセット方法を指定します。
複数 SAIC を接続する場合、通常 SAIC の個数分の配列が必要となりますが、パワーオンリセットする IC に関しては 1 つのみの記述で問題ありません。

uart_reset_t 構造体の設定

型	ユーザー設定	例
e_reset_process_t	SAIC リセット方法を記述	E_SAIC_POWERON_RESET
uint8_t *	リセット方法が外部リセットの場合、RESET 端子に接続しているポートレジスタのアドレスを 記述。それ以外の場合は NULL を記述	NULL
uint8_t	リセット方法が外部リセットの場合、RESET 端子に接続しているポートレジスタのビット番号を記述。それ以外の場合は 0 を記述	0U
uint32_t	wait する場合の NOP()の回数	D_PON_RST_NOP_CNT
uint8_t	内部リセット処理使用時の SAIC 番号。内部リセット処理以外の時は現在の API では使用していませんが、SAIC 番号を指定してください。(API Builder SAIC101 で使用しています。)	0U

(d) 記述例

- ケース 1
  SPI 割り込み使用
  SAIC101 を CSI10 に接続、CS_B 端子はポート P06、INT 端子はポート P07 に接続。リセット方法にパワーオンリセットを指定。

```
r_sa_spi_control_register.h
typedef enum
{
   E_SAIC300 = 0x00U,                    /* SAIC300     */
   E_SAIC301,                           /* SAIC301     */
   E_SAIC500,                           /* SAIC500     */
   E_SAIC501,                           /* SAIC501     */
   E_SAIC502,                           /* SAIC502     */
   E_SAIC101,                           /* SAIC101     */
   E_IC_TYPE_MAX,                        /* 最大値判定用 */
} e_saic_type_t;

typedef enum
{
   E_CSI00 = 0x00U,               /* CSI00 */
   E_CSI10,                      /* CSI10 */
   E_CSI_MAX                     /* 最大値判定用 */
} e_csi_ch_t;

r_sa_spi_control_register_user.c
const spi_saic_t g_spi_saic_data_tbl[] =
{
  { E_CSI10,   E_SAIC101,    &P0,      6U,        &P0,       7U,         }, /* SAIC 番号=0 の SAIC 情報 */
};

const spi_serial_t g_spi_serial_data_tbl[] =
{
//  { CSI_Start,    CSI_Stop,    CSI_Send_Receive,    }, /* format  */
    { NULL,         NULL,        NULL,               }, /* CSI00   */
    { R_CSI10_Start, R_CSI10_Stop, R_CSI10_Send_Receive, }, /* CSI10   */
};

const uart_reset_t g_uart_reset_data_tbl[] =
{
   //process,                Port address, Bit num, nop_cnt,             spi_saic_t 番号, }, /* format          */
   { E_SAIC_POWERON_RESET,      NULL,        0U,     D_PON_RST_NOP_CNT,     0U,           }, /* 1 番目の RESET 処理 */
};
```

- ケース 2
  SPI レジスタポーリング使用
  SAIC101 を CSI10 に接続、CS_B 端子はポート P16、INT 端子はポート P17 に接続。
  SAIC500 を CSI10 に接続、CS_B 端子はポート P14 に接続。RESET 端子はポート P40 に接続。
  SAIC501 を CSI00 に接続、CS_B 端子はポート P20 に接続。RESET 端子はポート P41 に接続。
  1 番目のリセット方法に P40 端子による外部リセットを指定。
  2 番目のリセット方法に P41 端子による外部リセットを指定。
  3 番目のリセット方法にパワーオンリセットを指定。

```
r_sa_spi_control_register.h
typedef enum
{
   E_SAIC300 = 0x00U,                    /* SAIC300     */
   E_SAIC301,                           /* SAIC301     */
   E_SAIC500,                           /* SAIC500     */
   E_SAIC501,                           /* SAIC501     */
   E_SAIC502,                           /* SAIC502     */
   E_SAIC101,                           /* SAIC101     */
   E_IC_TYPE_MAX,                        /* 最大値判定用 */
} e_saic_type_t;

typedef enum
{
   E_CSI00 = 0x00U,               /* CSI00 */
   E_CSI10,                      /* CSI10 */
   E_CSI_MAX                     /* 最大値判定用 */
} e_csi_ch_t;

r_sa_spi_control_register_user.c
const spi_saic_t g_spi_saic_data_tbl[] =
{
//  { csi_ch,    sa_type,      p_cs_addr, cs_bit_num, p_int_addr, int_bit_num, }, /* format             */
   { E_CSI10,   E_SAIC101,    &P1,      6U,       &P1,       7U,         }, /* SAIC 番号=0 の SAIC 情報 */
   { E_CSI10,   E_SAIC500,    &P1,      4U,       NULL,      0U,         }, /* SAIC 番号=1 の SAIC 情報 */
   { E_CSI00,   E_SAIC501,    &P2,      0U,       NULL,      0U,         }, /* SAIC 番号=2 の SAIC 情報 */
};

const spi_serial_t g_spi_serial_data_tbl[] =
{
   { &SIO00, &SIO000, &IF0H, 5U, (uint16_t *)&SSR00, (uint16_t *)&SIR00, R_CSI00_MaskStart, R_CSI00_Stop, }, /* CSI00   */
   { &SIO10, &SIO10, &IF1L, 1U, (uint16_t *)&SSR02, (uint16_t *)&SIR02, R_CSI10_MaskStart, R_CSI10_Stop, }, /* CSI10   */
};

const spi_reset_t g_spi_reset_data_tbl[] =
{
   //process,                Port address, Bit num, nop_cnt,              spi_saic_t 番号, }, /* format          */
   { E_SAIC_EXTERNAL_RESET,     &P4,         0U,     D_HARD_RESET_NOP_CNT1,  1U,           }, /* 1 番目の RESET 処理 */
   { E_SAIC_EXTERNAL_RESET,     &P4,         1U,     D_HARD_RESET_NOP_CNT2,  2U,           }, /* 2 番目の RESET 処理 */
   { E_SAIC_POWERON_RESET,      NULL,        0U,     D_PON_RST_NOP_CNT,     0U,           }, /* 3 番目の RESET 処理 */
};
```

4.10 グローバル定数

本節では API で定義されているグローバル定数について示します。

4.10.1 UART 制御

表 4-45 SAIC との通信に用いられる UART を規定するグローバル定数

型名	グローバル定数名	内容
const uint8_t	g_uart_saic_data_tbl_size	SAIC 情報格納グローバル定数の要素数
const uint8_t	g_uart_reset_data_tbl_size	RESET 情報格納グローバル定数の要素数

表 4-46 SAIC101 のフラッシュ・メモリ書き込みデータ例を規定するグローバル定数

型名	グローバル定数名	内容
const uint8_t	g_uart_flash_data_tbl_size	g_uart_smartanalog_flash_data の要素数
const saic_data_t	g_uart_smartanalog_flash_data[]	フラッシュ・メモリ上のレジスタシャドウ領域へ書き込む初期値例

表 4-47 CPU ウェイト時間を規定するグローバル定数

型名	グローバル定数名	内容
const uint16_t	g_uart_4800bps_half_bit_time	4800bps 時の 1bit 幅の半分の時間(約 105us)のループ回数カウンタ値。“D_UART_4800BPS_HALF_BIT”に定義した値から算出[注]
const uint16_t	g_uart_250kbps_half_bit_time	250kbps 時の 1bit 幅の半分の時間(約 2us)のループ回数カウンタ値。“D_UART_250kbps_HALF_BIT”に定義した値から算出[注]
const uint16_t	g_uart_response_step	レスポンス待ち関数ステップ計算用時間のループ回数カウンタ値。“D_UART_10MS”に定義した値から算出[注]
const uint32_t	g_uart_saic_auto_adc_timeout	OSR=2048 時のセットリングタイム値(8192ms) のループ回数カウンタ値。“D_UART_3US_NOP_CNT”に定義した値から算出[注]
const uint16_t	g_uart_5ms_nop_cnt	SBIAS 停止,スリープ・モード移行後安定待ち(5ms 以上) のループ回数カウンタ値。“D_UART_5MS”に定義した値から算出[注]
const uint16_t	g_uart_1800us_nop_cnt	AREG 動作 AREG 動作後安定待ち(1800us 以上)のループ回数カウンタ値。“D_UART_1800US”に定義した値から算出[注]
const uint16_t	g_uart_270us_nop_cnt	スリープ・モード復帰後安定待ち(270us 以上)のループ回数カウンタ値。“D_UART_270US”に定義した値から算出[注]
const uint16_t	g_uart_250us_nop_cnt	SBIAS 動作後安定待ち(250us 以上)のループ回数カウンタ値。“D_UART_250US”に定義した値から算出[注]

【注】基準周波数の“D_CPU_CLK_MHZ”からおおよその NOP 命令実行回数を算出

表 4-48 ビット操作用非公開グローバル定数

型名	グローバル定数名	内容
const uint8_t	gs_bit_tbl[]	2^n データテーブル定数。index に対応するデータを格納

4.10.2 SPI 制御

表 4-49 SAIC との通信に用いられる SPI を規定するグローバル定数

型名	グローバル定数名	内容
const uint8_t	g_spi_saic_data_tbl_size	SAIC 情報格納グローバル定数の要素数
const uint8_t	g_spi_reset_data_tbl_size	RESET 情報格納グローバル定数の要素数

表 4-50 SAIC101 のフラッシュ・メモリ書き込みデータ例を規定するグローバル定数

型名	グローバル定数名	内容
const uint8_t	g_spi_flash_data_tbl_size	g_spi_smartanalog_flash_data の要素数
const saic_data_t	g_spi_smartanalog_flash_data[]	フラッシュ・メモリ上のレジスタシャドウ領域へ書き込む初期値例

表 4-51 CPU ウェイト時間を規定するグローバル定数

型名	グローバル変数名	内容
const uint16_t	g_spi_5ms_nop_cnt	SBIAS 停止,スリープ･モード移行後安定待ち(5ms 以上)のループ回数カウンタ値。“D_SPI_5MS”に定義した値から算出[注]
const uint16_t	g_spi_1800us_nop_cnt	AREG 動作後安定待ち(1800us 以上) のループ回数カウンタ値。“D_SPI_1800US”に定義した値から算出[注]
const uint16_t	g_spi_820us_nop_cnt	スリープ・モード復帰後安定待ち(820us 以上) のループ回数カウンタ値。“D_SPI_820US”に定義した値から算出[注]
const uint16_t	g_spi_250us_nop_cnt	SBIAS 動作後安定待ち(250us 以上) のループ回数カウンタ値。“D_SPI_250US”に定義した値から算出[注]
const uint16_t	g_spi_3us_nop_cnt	Flash (Burst) Read のコマンド間遅延時間(3us 以上)のループ回数カウンタ値。“D_SPI_3US_NOP_CNT”に定義した値から算出[注]

【注】基準周波数の“D_CPU_CLK_MHZ”からおおよその NOP 命令実行回数を算出

表 4-52 ビット操作用非公開グローバル定数

型名	グローバル定数名	内容
const uint8_t	gs_bit_tbl[]	2^n データテーブル定数。index に対応するデータを格納

4.11 グローバル変数

本節では API で定義されているグローバル変数について示します。

(a) r_sa_uart_control_register.c

表 4-53 グローバル変数一覧 [UART]

型名	グローバル変数名	内容
static uint8_t	gs_adc_1shot	ADC 1Shot 取得フラグ。1=ADC 1Shot 取得時、0=それ以外
static uint16_t	gs_uart_half_bit_time	1bit 幅の半分の時間
static uint16_t	gs_uart_negotiation_timeout	通信ネゴシエーションタイムアウト値(ボーレートにより変更される)

5. 通信関連定義

5.1 API 関数仕様

5.1.1 SmartAnalog 初期化関数[UART/SPI 共通]

void R_SAIC_UART_Init(void)

void R_SAIC_SPI_Init(void)

説明	SmartAnalog 初期化関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	なし
グローバル変数	-UART の場合 gs_uart_half_bit_time: ストップビット待ちカウンタ g_uart_4800bps_half_bit_time: 4800bps 時の 1bit 幅の半分の時間 gs_uart_negotiation_timeout: 通信ネゴシエーションタイムアウト値 g_uart_reset_data_tbl_size: RESET 情報格納数 -SPI の場合 g_spi_reset_data_tbl_size: RESET 情報格納数
戻り値	なし
処理内容	1.初期化処理 -UART の場合 1.1.UART ストップビット待ちカウンタ初期化 1.2.UART 通信ネゴシエーションタイムアウト値初期化 -SPI の場合 1.1.CS 有効化処理関数をチャネル数分呼び出し全 SAIC を選択にする 1.2. CS 無効化処理関数をチャネル数分呼び出し全 SAIC を非選択にする 2. SmartAnalog リセット関数を要素数分繰り返し呼び出す

5.1.2 SmartAnalog リセット関数[UART/SPI 共通]

void R_SAIC_UART_Reset(uint8_t reset_num)

void R_SAIC_SPI_Reset(uint8_t reset_num)

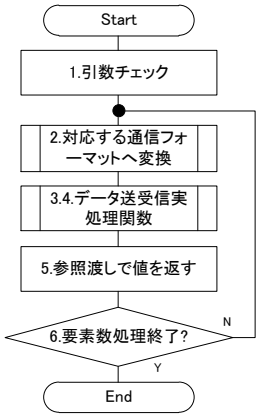
説明	SmartAnalog リセット関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	reset_num: RESET 情報格納グローバル変数の配列番号
グローバル変数	-UART の場合 g_uart_reset_data_tbl[]: RESET 情報格納グローバル変数 -SPI の場合 g_spi_reset_data_tbl[]: RESET 情報格納グローバル変数
戻り値	なし
処理内容	リセット処理に応じて対応するリセット処理用の内部関数を呼び出す

5.1.3 レジスタバイト読み出し関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_Read(uint8_t saic_num, saic_data_t *data, uint8_t num)

saic_status_t R_SAIC_SPI_Read(uint8_t saic_num, saic_data_t *data, uint8_t num)

説明	レジスタバイト読み出し関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *data: データバッファポインタ num: データ数
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 gs_uart_half_bit_time: ストップビット待ちカウンタ -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数データ数
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	エラー処理分岐はフローチャートでは省略



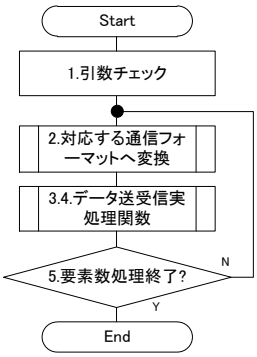
1. 引数エラーの場合 D_SAIC_ERR_PARAM を返し終了
2. 指定アドレスを元に対応する通信フォーマットへ変換
3. データ送受信実処理関数を実行
4. データ送受信実処理関数が異常終了の場合、対応するエラーを返し終了
5. 引数*data->data へ読み出し値を格納
6. 指定された要素数分 2～5 を繰り返す

5.1.4 レジスタバイト書き込み関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_Write(uint8_t saic_num, saic_data_t *data, uint8_t num)

saic_status_t R_SAIC_SPI_Write(uint8_t saic_num, saic_data_t *data, uint8_t num)

説明	レジスタバイト書き込み関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *data: データバッファポインタ num: データ数
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 gs_uart_half_bit_time: ストップビット待ちカウンタ -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	エラー処理分岐はフローチャートでは省略



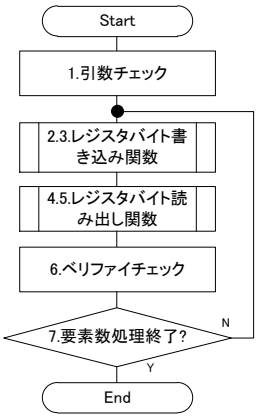
1. 引数エラーの場合D_SAIC_ERR_PARAMを返し終了
2. 指定アドレス、データを元に対応する通信フォーマットへ変換
3. 送受信関数実行
4. 送受信関数が異常終了の場合、対応するエラーを返し終了
5. 指定された要素数分2～4を繰り返す

5.1.5 ベリファイ付きレジスタバイト書き込み関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_WriteVerify
(uint8_t saic_num, saic_data_t *data, uint8_t num, uint8_t *err_index)

saic_status_t R_SAIC_SPI_WriteVerify
(uint8_t saic_num, saic_data_t *data, uint8_t num, uint8_t *err_index)

説明	ベリファイ付きレジスタバイト書き込み関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *data: データバッファポインタ num: データ数 *err_index: エラーとなった saic_data 配列インデックス値
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_VERIFY or D_SAIC_ERR_PARAM
処理内容	エラー処理分岐はフローチャートでは省略



1. 引数エラーの場合 D_SAIC_ERR_PARAM を返し終了
2. 指定アドレスへレジスタバイト書き込み関数を実行しデータを書き込む
3. レジスタバイト書き込み関数が異常終了の場合、対応するエラーを返し終了
4. 指定アドレスへレジスタバイト読み出し関数を実行しデータを読み出す
5. 異常終了の場合、対応するエラーを返し終了
6. 指定アドレスへの書き込み値と読み出し値を比較し不一致の場合、不一致の要素番号、D_SAIC_ERR_VERIFY を返し終了
7. 指定された要素数分 2～6 を繰り返す

5.1.6 SAIC101 固有コマンド処理関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_SAIC101
(uint8_t saic_num, e_saic101_func_t e_func, saic_data_t *data, uint16_t length)

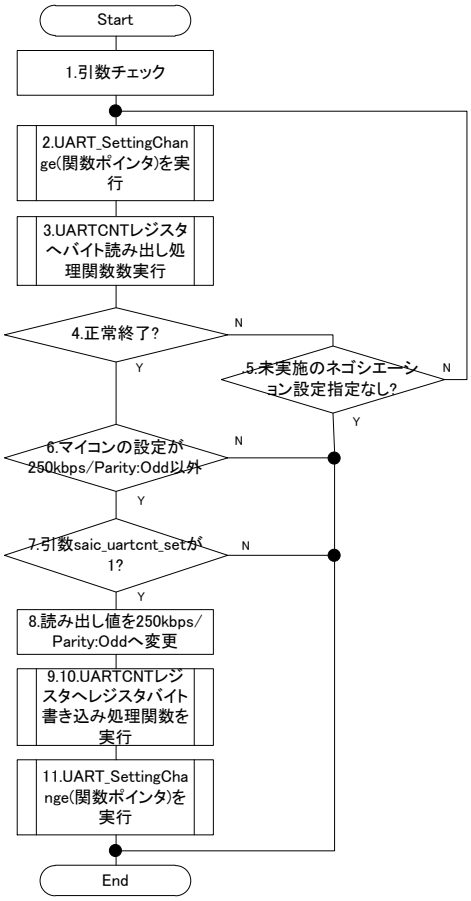
saic_status_t R_SAIC_SPI_SAIC101
(uint8_t saic_num, e_saic101_func_t e_func, saic_data_t *data, uint16_t length)

説明	SAIC101 固有コマンド処理関数。 引数の e_func によりレジスタバーストリード・ライト関数/フラッシュ・メモリ関数に分岐する。
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 e_func: SAIC101 固有機能指定 *data: データバッファポインタ length: データ長
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数 g_spi_saic_data_tbl[]: SAIC 情報格納グローバル変数
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM or D_SAIC_ERR_VERIFY
処理内容	1.引数エラーの場合 D_SAIC_ERR_COM を返し終了 2.指定処理を元に対応する内部関数を呼び出す 3.対応する内部関数が異常終了した場合、対応するエラーを返し終了

5.1.7 通信設定ネゴシエーション関数[UART のみ]

saic_status_t R_SAIC_UART_Negotiation(uint8_t saic_num, uint8_t saic_uartcnt_set)

説明	通信設定ネゴシエーション関数。SAIC101 との通信設定を自動認識し、250kbps/Parity Odd に設定する。
ヘッダ	r_sa_uart_control_register.h
引数	saic_num: SAIC 番号 saic_uartcnt_set: SAIC UART 通信設定変更処理可否判断フラグ(0 変更なし,1 変更あり)
グローバル変数	g_uart_saic_data_tbl_size: SAIC 情報格納数 gs_uart_half_bit_time: ストップビット待ちカウンタ gs_uart_negotiation_timeout: 通信ネゴシエーションタイムアウト値 g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 g_uart_serial_data_tbl: UART テーブル g_uart_4800bps_half_bit_time: 4800bps 時の 1bit 幅の半分の時間 g_uart_250kbps_half_bit_time: 250kbps 時の 1bit 幅の半分の時間(約 2us) g_uart_response_step: レスポンス待ち関数ステップ計算用時間
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	エラー処理分岐はフローチャートでは省略

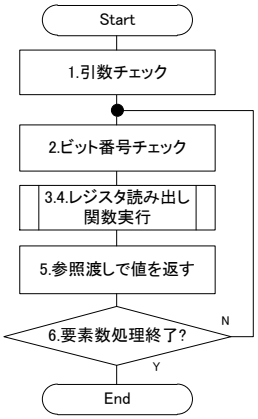


1. 引数エラーの場合 D_SAIC_ERR_PARAM を返し終了、関数ポインタ NULL の場合 D_SAIC_ERR_COM を返し終了
2. UART_SettingChange(関数ポインタ)を実行し、マイコンの UART 通信設定を変更
3. UARTCNT レジスタへレジスタバイト読み出し処理関数数を実行する
4. ネゴシエーションタイムアウトの判定
5. 未実施のネゴシエーション関数処理項目設定用宣言されている定義が有る場合処理 2 へ
6. 正常終了したときのマイコンの通信設定が 250kbps/Parity:Odd の場合処理処理完了
7. 引数 saic_uartcnt_set が 1 で無い場合処理完了
8. UARTCNT レジスタ読み出し値から UART 設定を 250kbps/Parity:Odd へ変更
9. UARTCNT レジスタへレジスタバイト書き込み処理関数を実行
10. 異常終了の場合、対応するエラーを返し終了
11. UART_SettingChange(関数ポインタ)を実行し、マイコンの UART 設定を 250kbps/Parity:Odd へ変更

5.1.8 レジスタビット読み出し関数[SPI のみ]

saic_status_t R_SAIC_SPI_ReadBit
(uint8_t saic_num, saic_data_bit_t *data, uint8_t num, uint8_t *err_index)

説明	レジスタビット読み出し関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *data: データバッファポインタ num: データ数 *err_index: エラーとなった saic_data 配列インデックス値
グローバル変数	g_spi_saic_data_tbl_size: SAIC 情報格納数 gs_bit_tbl[]: 8bit ビットテーブル配列
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	エラー処理分岐はフローチャートでは省略

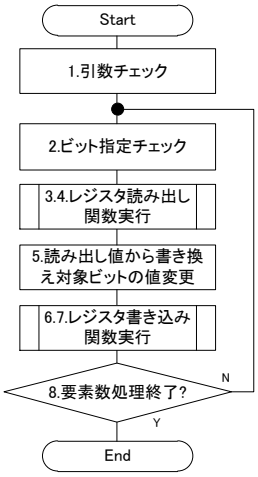


1. 引数エラーの場合 D_SAIC_ERR_PARAM を返し終了
2. ビット番号の指定がエラーの場合 D_SAIC_ERR_PARAM を返し終了
3. レジスタ読み出し関数実行
4. レジスタ読み出し関数が異常終了の場合、対応するエラーを返し終了
5. 引数*data->bit_data へ読み出しビットの値を格納
6. 指定された要素数分 2～5 を繰り返す

5.1.9 レジスタビット書き込み関数[SPI のみ]

saic_status_t R_SAIC_SPI_WriteBit
(uint8_t saic_num, saic_data_bit_t *data, uint8_t num, uint8_t *err_index)

説明	レジスタビット書き込み関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *data: データバッファポインタ num: データ数 *err_index: エラーとなった saic_data 配列インデックス値
グローバル変数	g_spi_saic_data_tbl_size: SAIC 情報格納数 gs_bit_tbl[]: 8bit ビットテーブル配列
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	エラー処理分岐はフローチャートでは省略

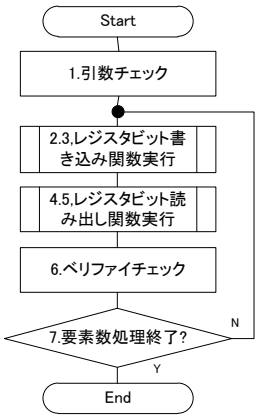


1. 引数エラーの場合 D_SAIC_ERR_PARAM を返し終了
2. ビット番号、書き込みデータの指定がエラーの場合、D_SAIC_ERR_PARAM を返し終了
3. レジスタ読み出し関数実行
4. レジスタ読み出し関数が異常終了の場合、対応するエラーを返し終了
5. 対象ビット値変更
6. レジスタ書き込み関数実行
7. レジスタ書き込み関数が異常終了の場合、対応するエラーを返し終了
8. 指定された要素数分 2～7 を繰り返す

5.1.10 ベリファイ付きレジスタビット書き込み関数[SPI のみ]

saic_status_t R_SAIC_SPI_WriteVerifyBit(uint8_t saic_num, saic_data_bit_t *data,
uint8_t num, uint8_t *err_index)

説明	ベリファイ付きレジスタビット書き込み関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *data: データバッファポインタ num: データ数 *err_index: エラーとなった saic_data 配列インデックス値
グローバル変数	g_spi_saic_data_tbl_size: SAIC 情報格納数
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM or D_SAIC_ERR_VERIFY
処理内容	エラー処理分岐はフローチャートでは省略

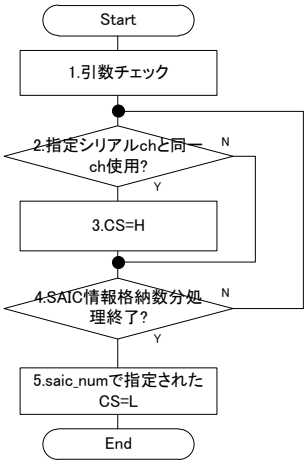


1. 引数エラーの場合 D_SAIC_ERR_PARAM を返し終了
2. 指定アドレスへレジスタビット書き込み関数実行
3. レジスタビット書き込み関数が異常終了の場合、対応するエラーを返し終了
4. 指定アドレスへレジスタビット読み出し関数実行
5. レジスタビット読み出し関数が異常終了の場合、対応するエラーを返し終了
6. 指定アドレスへの書き込み値と読み出し値を比較し不一致の場合、不一致の要素番号、D_SAIC_ERR_VERIFY を返し終了
7. 指定された要素数分 2～6 を繰り返す

5.1.11 CS 有効化処理関数[SPI のみ]

void R_SAIC_SPI_CSEnable(uint8_t saic_num)

説明	CS 有効化処理関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号
グローバル変数	g_spi_saic_data_tbl_size: SAIC 情報格納数 g_spi_saic_data_tbl[]: SAIC 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列
戻り値	なし
処理内容	エラー処理分岐はフローチャートでは省略



1. 引数エラーの場合及び、関数ポインタが NULL の場合終了
2. saic_num で指定されたシリアル ch と同一の ch を使用しているか判定
3. 同一の ch を使用している SAIC の CS 端子は H(非選択)にする
4. SAIC 情報格納数分 2～3 を繰り返す
5. saic_num で指定された CS 端子を L(選択)にする

5.1.12 CS チェック関数[SPI のみ]

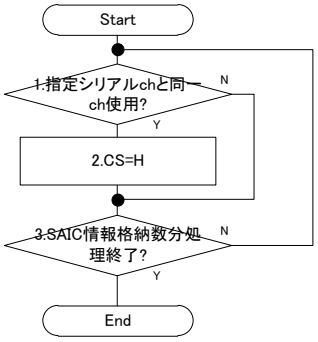
uint8_t R_SAIC_SPI_CSCheck(uint8_t saic_num)

説明	CS チェック関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号
グローバル変数	g_spi_saic_data_tbl_size: SAIC 情報格納数 g_spi_saic_data_tbl[]: SAIC 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列
戻り値	uint8_t: 0 有効 1 無効
処理内容	1.引数エラーまたは SAIC 情報格納グローバル変数に格納されている関数ポインタが NULL の場合 1 を返し終了 2.saic_num で指定された SAIC の CS 端子が L(選択)ならば 0、H(非選択)ならば 1 を返す

5.1.13 CS 無効化処理関数[SPI のみ]

void R_SAIC_SPI_CSDisable(e_csi_ch_t cs_ch)

説明	CS 無効化処理関数
ヘッダ	r_sa_spi_control_register.h
引数	cs_ch: 対象の CSIch 番号
グローバル変数	g_spi_saic_data_tbl_size: SAIC 情報格納数 g_spi_saic_data_tbl[]: SAIC 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列
戻り値	なし
処理内容	エラー処理分岐はフローチャートでは省略



1. 指定シリアル ch と同一の ch を使用しているか判定
2. 同一の ch を使用している SAIC の CS は H(非選択)にする
3. SAIC 情報格納数分 1～2 を繰り返す

5.2 内部関数仕様

5.2.1 SmartAnalog 外部リセット関数[UART/SPI 共通]

static void r_saic_uart_external_reset(uint8_t reset_num)

static void r_saic_spi_external_reset(uint8_t reset_num)

説明	SmartAnalog 外部リセット関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	reset_num: RESET 情報格納グローバル変数の配列番号
グローバル変数	-UART の場合 g_uart_reset_data_tbl[]: RESET 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列 -SPI の場合 g_spi_reset_data_tbl[]: RESET 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列
戻り値	なし
処理内容	1.RESET 端子に接続されているポート出力を L にし、外部リセット発生させる 2.NOP 実行関数を実行し、リセット時間を待つ 3.RESET 端子に接続されているポート出力を H にして外部リセットを解除

5.2.2 SmartAnalog 内部リセット関数[UART/SPI 共通]

static void r_saic_uart_internal_reset(uint8_t reset_num)

static void r_saic_spi_internal_reset(uint8_t reset_num)

説明	SmartAnalog 内部リセット関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	reset_num: RESET 情報格納グローバル変数の配列番号
グローバル変数	-UART の場合 g_uart_reset_data_tbl[]: RESET 情報格納グローバル変数 g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 -SPI の場合 g_spi_saic_data_tbl[]: SAIC 情報格納グローバル変数 g_spi_reset_data_tbl[]: RESET 情報格納グローバル変数
戻り値	なし
処理内容	1.指定の SAIC に対応したリセット制御レジスタに”1”を書き込み内部リセット発生させる 2.指定の SAIC に対応したリセット制御レジスタに”0”を書き込み内部リセットを解除

5.2.3 SmartAnalog パワーオンリセット待ち関数[UART/SPI 共通]

static void r_saic_uart_poweron_reset(uint8_t reset_num)

static void r_saic_spi_poweron_reset(uint8_t reset_num)

説明	SmartAnalog パワーオンリセット待ち関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	reset_num: RESET 情報格納グローバル変数の配列番号
グローバル変数	-UART の場合 g_uart_reset_data_tbl[]: RESET 情報格納グローバル変数 -SPI の場合 g_spi_reset_data_tbl[]: RESET 情報格納グローバル変数
戻り値	なし
処理内容	NOP 実行関数を呼び出し、引数として RESET 情報格納グローバル変数で指定された NOP 実行回数を渡してリセット時間を待つ

5.2.4 NOP 実行関数[UART/SPI 共通]

static void r_nop_wait(uint32_t nop_cnt)

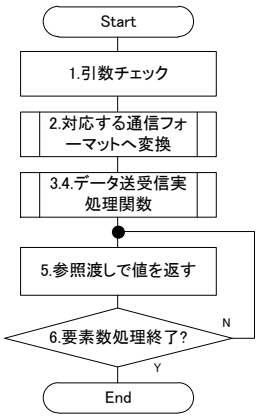
説明	NOP 実行関数。引数の nop_cnt 分 NOP()命令を実行する
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	nop_cnt: NOP 実行回数
グローバル変数	なし
戻り値	なし
処理内容	nop_cnt で指定された回数分 NOP()命令を繰り返し実行する

5.2.5 バースト読み出し処理関数[UART/SPI 共通]

static saic_status_t r_saic_uart_burst_read(uint8_t saic_num, saic_data_t *data, uint8_t length)

static saic_status_t r_saic_spi_burst_read(uint8_t saic_num, saic_data_t *data, uint8_t length)

説明	バースト読み出し処理関数。[Register Burst Read]コマンドを発行する。
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *data: データバッファポインタ length: データ長
グローバル変数	-UART の場合 gs_uart_half_bit_time: ストップビット待ちカウンタ
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	エラー処理分岐はフローチャートでは省略



1. 引数エラーの場合 D_SAIC_ERR_PARAM を返し終了
2. 指定アドレスを元に対応する通信フォーマットへ変換
3. 送受信関数実行
4. 送受信関数が異常終了の場合、対応するエラーを返し終了
5. 引数*data->data へ読み出し値を格納
6. 指定された要素数分アドレスインクリメントしながら 5 を繰り返す

5.2.6 バースト書き込み処理関数[UART/SPI 共通]

static saic_status_t r_saic_uart_burst_write
(uint8_t saic_num, saic_data_t *data, uint8_t length)

static saic_status_t r_saic_spi_burst_write
(uint8_t saic_num, saic_data_t *data, uint8_t length)

説明	バースト書き込み処理関数。[Register Burst Write]コマンドを発行する。
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *data: データバッファポインタ length: データ長
グローバル変数	-UART の場合 gs_uart_half_bit_time: ストップビット待ちカウンタ
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	1.引数エラーの場合 D_SAIC_ERR_PARAM を返し終了 2.指定アドレス、データを元に対応する通信フォーマットへ変換 3.送受信関数実行 4.送受信関数が異常終了の場合、対応するエラーを返し終了

5.2.7 SAIC101 専用通信コマンドフォーマット変換関数[UART/SPI 共通]

static void r_saic_uart_format_conv_saic101(uint8_t *address, e_saic101_func_t e_func)

static void r_saic_spi_format_conv_saic101(uint8_t *address, e_saic101_func_t e_func)

説明	SAIC101 専用通信フォーマット変換関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	e_func: SAIC101 固有機能指定 *address: ターゲットレジスタアドレス
グローバル変数	なし
戻り値	なし
処理内容	SAIC101 専用の指定コマンドを元に、対応する通信フォーマットへ変換

5.2.8 SPI フォーマット変換関数[SPI のみ]

static void r_saic_spi_format_conv(uint8_t saic_num, uint8_t *address, e_rw_t rw)

説明	SPI フォーマット変換関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *address: ターゲットレジスタアドレス rw: リード/ライト指定
グローバル変数	g_spi_saic_data_tbl[]: SAIC 情報格納グローバル変数
戻り値	なし
処理内容	SAIC 型名及び、指定コマンドを元に対応する通信フォーマットへ変換

5.2.9 オーバーランエラー発生チェック関数[SPI のみ]

static uint8_t r_saic_spi_overrun_err_check(uint8_t saic_num)

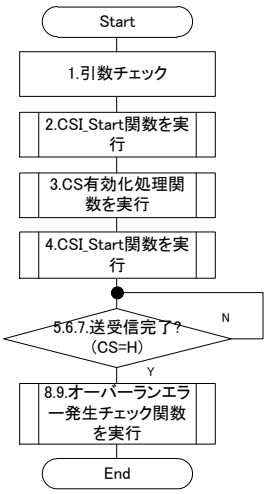
説明	オーバーランエラー発生チェック関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号
グローバル変数	g_spi_saic_data_tbl[]: SAIC 情報格納グローバル変数 g_csi_overrun_flag: SPI オーバランフラグ用変数 gs_bit_tbl[]: 8bit ビットテーブル配列
戻り値	uint8_t: 0 OK or 1 over run error 発生
処理内容	1.SPI オーバランフラグ用変数の指定した ch に対応するビットをチェック 2.オーバーラン発生していなければ 0、発生していればビットをクリアし、1 を返す

5.2.10 データ送受信実処理関数[SPI のみ]

static saic_status_t r_saic_spi_write_read
(uint8_t saic_num, uint8_t tx_buffer[], uint8_t rx_buffer[], uint16_t length)

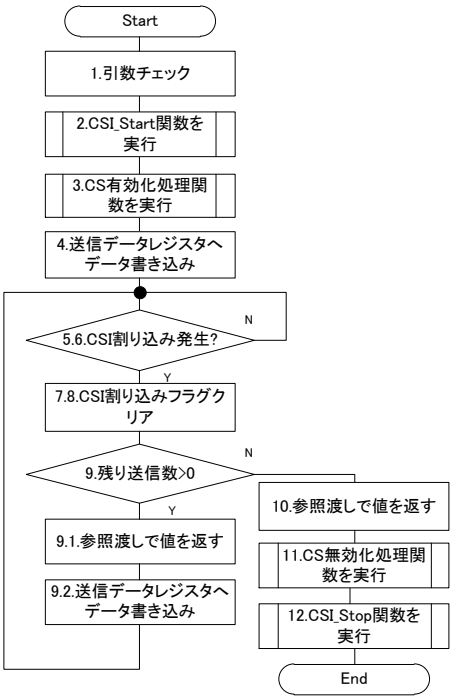
説明	データ送受信実処理関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 tx_buffer[]: 送信バッファ rx_buffer[]: 受信バッファ length: データ長
グローバル変数	g_spi_saic_data_tbl[]: SAIC 情報格納グローバル変数 g_spi_serial_data_tbl[]: シリアルモジュール情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列
戻り値	saic_status_t D_SAIC_OK or D_SAIC_ERR_COM
処理内容	エラー処理分岐はフローチャートでは省略

■割り込み使用処理



1. アドレス、関数ポインタが NULL の場合 D_SAIC_ERR_COM を返し終了
2. CSI_Start 関数を実行し CSI 通信許可にする
3. CS 有効化処理関数を実行しターゲットの SAIC の CS をアクティブにする
4. CSI_Start 関数を実行し送受信を開始する
5. 送受信完了までループ
6. CSIxx 送信完了割り込みにて CS=H、CSI_Stop 関数を実行
7. デッドロックが発生した場合エラー処理を行い D_SAIC_ERR_COM を返し終了
   CS=H、CSI_Stop 関数を実行し送受信停止
   CS 無効化処理関数を実行しターゲットの SAIC の CS を非アクティブにする
8. オーバーランエラー発生チェック関数を実行
9. オーバフローフラグが立っていたら D_SAIC_ERR_COM を返し終了

■割り込み不使用(ポーリング)処理

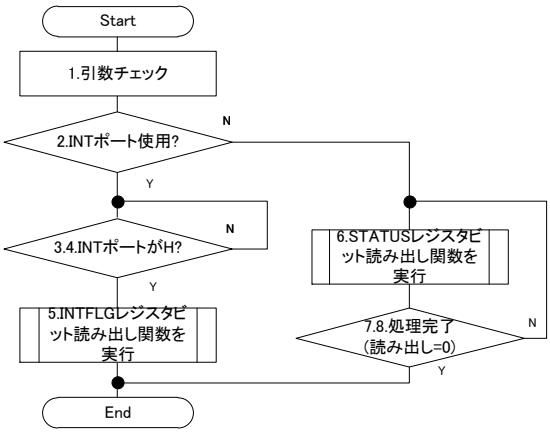


1. アドレス、関数ポインタが NULL の場合 D_SAIC_ERR_COM を返し終了
2. CSI_Start 関数を実行
3. CS 有効化処理関数を実行
4. 送信データレジスタへデータ書き込み
5. CSI 割り込みが発生するまでループ
6. デッドロックが発生した場合は D_SAIC_ERR_COM を返し終了
7. CSI 割り込みフラグクリア
8. 通信エラーが発生した場合は D_SAIC_ERR_COM を返し終了
9. 残り送信数>0 の場合、5～9 を繰り返す。
   9.1. 引数 rx_buffer[]へ受信データを格納
   9.2. 送信データレジスタへデータ書き込み
10. 引数 rx_buffer[]へ受信データを格納
11. CS 無効化処理関数を実行
12. CSI_Stop 関数を実行

5.2.11 ポーリング監視処理関数[SPI のみ]

static saic_status_t r_saic101_spi_polling
(uint8_t saic_num, saic_func_bitRead_t p_func_int, saic_func_bitRead_t p_func_status)

説明	ポーリング監視処理関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 p_func_int: INTFLG レジスタビット読み出し関数 p_func_status: STATUS レジスタビット読み出し関数
グローバル変数	g_spi_saic_data_tbl[]: SAIC 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列
戻り値	saic_status_t D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	エラー処理分岐はフローチャートでは省略



1. INT ポート不使用(ポーリング使用)、かつ p_func_status=NULL の場合 D_SAIC_ERR_COM を返し、ループを抜ける
2. INT ポート使用の判定。
3. INT ポート使用時 INT ポートが H になるまでループ
4. ループ中にデッドロックが発生した場合 D_SAIC_ERR_COM を返し、ループを抜ける
5. INTFLG レジスタビット読み出し関数を実行する
6. INT ポート不使用(ポーリング使用)時、STATUS レジスタビット読み出し関数を実行
7. 処理完了(読み出し値=0)になるまでループ
8. 処理中にデッドロックが発生した場合 D_SAIC_ERR_COM を返し、ループを抜ける

5.2.12 コマンド送信&応答受信関数[UART のみ]

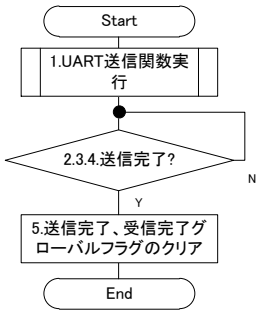
static saic_status_t r_saic_uart_write_read
(uint8_t saic_num, uint8_t tx_buffer[], uint8_t rx_buffer[], uint16_t tx_length, uint16_t * p_rx_length)

説明	コマンド送信&応答受信関数
ヘッダ	r_sa_uart_control_register.h
引数	saic_num: SAIC 番号 tx_buffer[]: 送信バッファ rx_buffer[]: 受信バッファ tx_length: 送信サイズ p_rx_length: 受信サイズ
グローバル変数	g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列 gs_adc_1shot: ADC 1Shot 取得フラグ。1=ADC 1Shot 取得時、0=それ以外 g_uart_rx_end_flag: UART 受信完了フラグ用変数 g_uart_serial_data_tbl: UART テーブル
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM
処理内容	1.関数ポインタが NULL の場合 D_SAIC_ERR_COM を返し終了 2.UART 受信関数を実行し受信待ち状態に設定 3.UART 通信開始関数を実行 4.コマンド送信関数を実行し送信バッファのデータを送信サイズ分送信する 5.コマンド送信関数が異常終了の場合、対応するエラーを設定 6.コマンド送信関数が正常終了の場合、UART 受信データパケット解析関数を実行する 7.UART 停止 8.ADC 1shot フラグクリア

5.2.13 コマンド送信関数[UART のみ]

static saic_status_t r_saic_uart_send_command
(uint8_t saic_num, uint8_t tx_buffer[], uint16_t tx_length)

説明	コマンド送信関数(割り込み使用時のみ動作)
ヘッダ	r_sa_uart_control_register.h
引数	saic_num: SAIC 番号 tx_buffer[]: 送信バッファ tx_length: 送信サイズ
グローバル変数	g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 g_uart_serial_data_tbl: UART テーブル gs_bit_tbl[]: 8bit ビットテーブル配列 g_uart_tx_end_flag: UART 送信完了フラグ用変数 g_uart_rx_end_flag: UART 受信完了フラグ用変数 gs_uart_negotiation_timeout: 通信ネゴシエーションタイムアウト値
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM
処理内容	エラー処理分岐はフローチャートでは省略

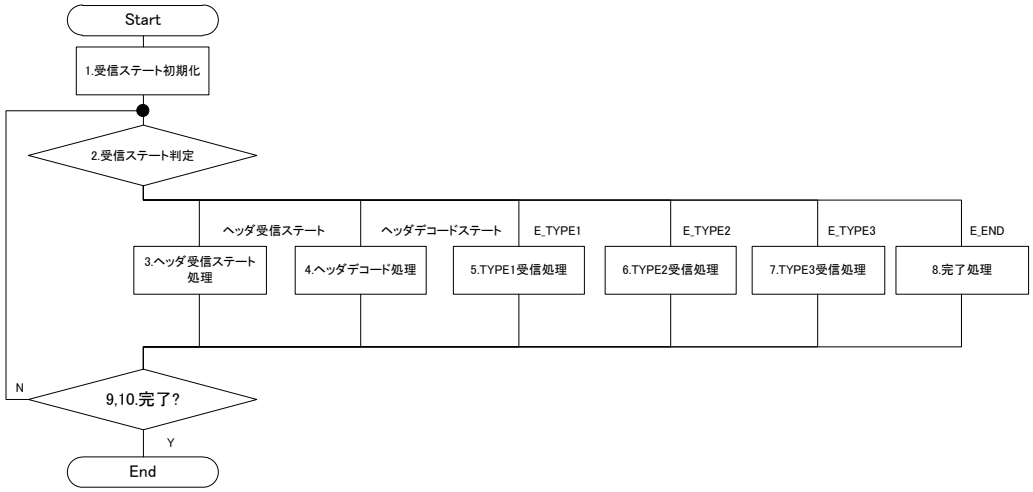


1. UART 送信関数実行
2. ネゴシエーションタイムアウトの場合 UART 停止後に D_SAIC_ERR_COM を返し終了
3. デッドロックが発生した場合 D_SAIC_ERR_COM を返し、ループを抜ける
4.送信完了までループ
5. 送信完了、受信完了グローバルフラグのクリア

5.2.14 UART 受信データパケット解析関数[UART のみ]

static saic_status_t r_saic_uart_get_response
(uint8_t saic_num, uint8_t rx_buffer[], uint16_t * p_rx_length)

説明	UART 受信データパケット解析関数。受信したヘッダから応答 type を判断し、1 応答分メッセージ受信する。(割り込み使用時のみ動作)
ヘッダ	r_sa_uart_control_register.h
引数	saic_num: SAIC 番号 rx_buffer[]: 受信バッファ p_rx_length: 受信サイズ
グローバル変数	g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 g_uart_serial_data_tbl: UART テーブル gs_bit_tbl[]: 8bit ビットテーブル配列 gs_adc_1shot: ADC 1Shot 取得フラグ。1=ADC 1Shot 取得時、0=それ以外 g_uart_rx_end_flag: UART 受信完了フラグ用変数 gs_uart_negotiation_timeout: 通信ネゴシエーションタイムアウト値
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM
処理内容	エラー処理分岐はフローチャートでは省略



1. 受信ステートにヘッダ受信ステートを設定
2. 受信ステートを判定
3. ヘッダ受信ステート処理
   1 パケット以上受信していれば受信ステートにヘッダデコードステートを設定
4. ヘッダデコード処理
   受信データからヘッダを解析し受信ステートに TYPE1～TYPE3 ステートを設定
5. TYPE1 受信処理
   TYPE1 フォーマット受信完了で受信ステートに END ステートを設定
6. TYPE2 受信処理
   TYPE2 フォーマット受信完了で受信ステートに END ステートを設定
7. TYPE3 受信処理
   TYPE3 フォーマット受信完了で受信ステートに END ステートを設定
8. 完了処理
   完了フラグをセット
9. 受信完了(完了フラグがセットされる)までループ
10. ネゴシエーションタイムアウト、デッドロックが発生した場合 D_SAIC_ERR_COM を返し終了

6. フラッシュ・メモリ制御関連定義

6.1 API 関数仕様

6.1.1 SAIC101 固有コマンド処理関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_SAIC101
(uint8_t saic_num, e_saic101_func_t e_func, saic_data_t *data, uint16_t length)

saic_status_t R_SAIC_SPI_SAIC101
(uint8_t saic_num, e_saic101_func_t e_func, saic_data_t *data, uint16_t length)

説明	SAIC101 固有コマンド処理関数。引数の e_func によりレジスタバーストリード・ライト関数/フラッシュ・メモリ関数に分岐する。
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 e_func: SAIC10 固有機能指定 *data: データバッファポインタ length: データ長
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数 g_spi_saic_data_tbl[]: SAIC 情報格納グローバル変数
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM or D_SAIC_ERR_VERIFY
処理内容	1.引数エラーの場合 D_SAIC_ERR_COM を返し終了 2.指定処理を元に対応する内部関数を呼び出す 3.対応する内部関数が異常終了した場合、対応するエラーを返し終了

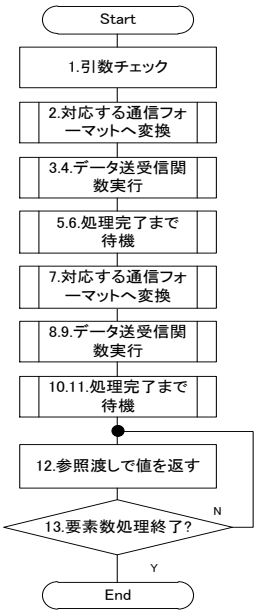
6.2 内部関数仕様

6.2.1 フラッシュ・データ読み出し処理関数[UART/SPI 共通]

static saic_status_t r_saic_uart_flash_read(uint8_t saic_num, saic_data_t *data, uint16_t length)

static saic_status_t r_saic_spi_flash_read(uint8_t saic_num, saic_data_t *data, uint16_t length)

説明	フラッシュ・データ読み出し処理関数。[Flash (Burst) Read]コマンドを発行する。
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *data: データバッファポインタ length: データ長
グローバル変数	-UART の場合 g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列 g_uart_rx_end_flag: UART 受信完了フラグ用変数 gs_uart_half_bit_time: ストップビット待ちカウンタ g_uart_serial_data_tbl: UART テーブル -SPI の場合 g_spi_saic_data_tbl[]: SAIC 情報格納グローバル変数 g_spi_3us_nop_cnt: Flash (Burst) Read のコマンド間遅延時間(3us 以上)のループ回数カウンタ値。
戻り値	saic_status_t D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	エラー処理分岐はフローチャートでは省略



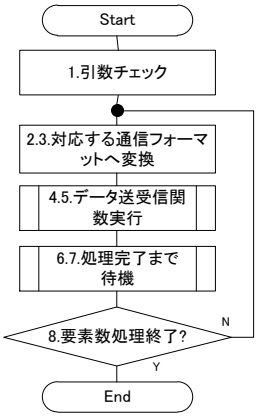
1. エラーの場合対応するエラーを返し終了
   -UART の場合引数エラーの場合 D_SAIC_ERR_PARAM を返す
   関数ポインタ NULL の場合 D_SAIC_ERR_COM を返す
   -SPI の場合
   引数エラーの場合 D_SAIC_ERR_PARAM を返す
2. 指定アドレスを元に対応する通信フォーマット(2Byte)へ変換
3. データ送受信関数を実行し、コマンド 1 を送受信する
   -UART の場合
   コマンド送信&応答受信関数を実行
   -SPI の場合
   データ送受信実処理関数を実行
4. データ送受信実処理関数が異常終了した場合、対応するエラーを返し終了
5. データ送受信実処理処理完了まで待機する
   -UART の場合
   UART 受信データパケット解析関数を実行する
   -SPI の場合
   INT 端子使用時は INT 割り込みを待つ
   INT 端子不使用(ポーリング使用)時は 3us 以上待つ
6. 処理完了まで待機する処理が異常終了した場合、対応するエラーを返し終了
7. 対応する通信フォーマットへ変換
8. データ送受信関数を実行し、コマンド 2 を送受信する
   -UART の場合
   コマンド送信&応答受信関数を実行
   -SPI の場合
   データ送受信実処理関数を実行
9. データ送受信関数が異常終了した場合、対応するエラーを返し終了
10. Flash (Burst) Read 処理完了まで待機する
   -UART の場合 UART 受信データパケット解析関数を実行する
11. 処理完了まで待機する処理が異常終了した場合、対応するエラーを返し終了
12. 引数*data->address へアドレス、*data->data へ読み出し値を格納
13. 引数 length で指定された要素数分アドレスインクリメントしながら 12 を繰り返す

6.2.2 フラッシュ・データ書き込み処理関数[UART/SPI 共通]

static saic_status_t r_saic_uart_flash_write(uint8_t saic_num, saic_data_t *data, uint16_t num)

static saic_status_t r_saic_spi_flash_write(uint8_t saic_num, saic_data_t *data, uint16_t num)

説明	フラッシュ・データ書き込み処理関数。01H、1FH 番地以外の書き込みが可能。[Flash Write]コマンドを発行する。
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *data: データバッファポインタ num: データ数
グローバル変数	-UART の場合 g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列 g_uart_rx_end_flag: UART 受信完了フラグ用変数 gs_uart_half_bit_time: ストップビット待ちカウンタ g_uart_serial_data_tbl: UART テーブル -SPI の場合 なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	エラー処理分岐はフローチャートでは省略



1. エラーの場合対応するエラーを返し終了
   -SPI の場合引数エラーの場合 D_SAIC_ERR_PARAM を返す
   -UART の場合引数エラーの場合 D_SAIC_ERR_PARAM を返す
   関数ポインタ NULL の場合 D_SAIC_ERR_COM を返す
2. 指定アドレスが 0x01H、0x1F の場合 D_SAIC_ERR_PARAM を返し終了
3. 指定アドレス、データを元に対応する通信フォーマットへ変換
4. データ送受信実処理関数を実行
   -SPI の場合データ送受信実処理関数を実行
   -UART の場合コマンド送信&応答受信関数を実行
5. データ送受信関数が異常終了した場合、対応するエラーを返し終了
6. Flash Write 処理完了まで待機する
   -SPI 制御の場合ポーリング監視処理関数を実行し、INT 割り込み待ちまたは、FWIP フラグをポーリングする
   -UART の場合 UART 受信データパケット解析関数を実行する
7. 処理完了まで wait の処理が異常終了の場合、対応するエラーを返し終了
8. 指定された要素数分 2～7 を繰り返す

6.2.3 フラッシュ・データ全消去処理関数[UART/SPI 共通]

static saic_status_t r_saic_uart_flash_all_erase(uint8_t saic_num)

static saic_status_t r_saic_spi_flash_all_erase(uint8_t saic_num)

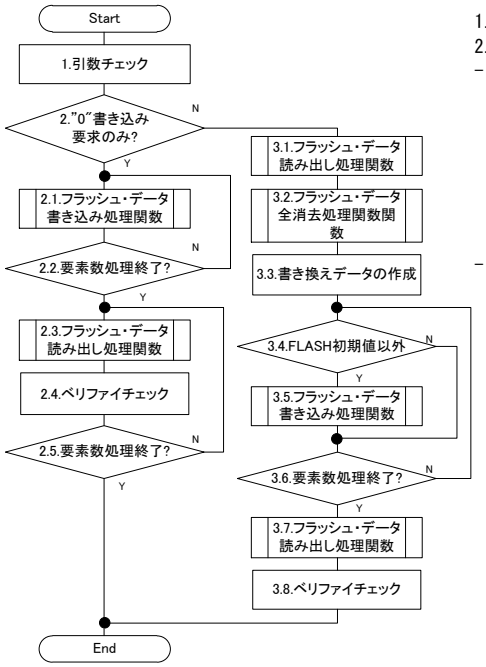
説明	フラッシュ・データ全消去処理関数。[Flash All Erase]コマンドを発行する。
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号
グローバル変数	-UART の場合 g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列 g_uart_rx_end_flag: UART 受信完了フラグ用変数 gs_uart_half_bit_time: ストップビット待ちカウンタ g_uart_serial_data_tbl: UART テーブル -SPI の場合 なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	1. エラーの場合対応するエラーを返し終了 -UART の場合 関数ポインタが NULL の場合 D_SAIC_ERR_COM を返す 2. Flash All Erase コマンド(2Byte)生成 3. データ送受信関数を実行し、コマンドを送信する -UART の場合コマンド送信&応答受信関数を実行 2Byte 送受信 -SPI の場合データ送受信実処理関数を実行 1Byte 目送受信完了後、2Byte 目送受信 4. データ送受信関数が異常終了した場合、対応するエラーを返し終了 5. Flash All Erase 処理完了まで待機する -UART の場合 UART 受信データパケット解析関数を実行する -SPI の場合 ポーリング監視処理関数を実行し、INT 割り込み待ちまたは、FAEIP フラグをポーリングする 6. 処理完了まで待機する処理が異常終了した場合、対応するエラーを返し終了

6.2.4 フラッシュ・ベリファイ付きメモリデータ書き込み関数[UART/SPI 共通]

saic_status_t r_saic_uart_flash_write_verify(uint8_t saic_num, saic_data_t data[], uint16_t num)

saic_status_t r_saic_spi_flash_write_verify(uint8_t saic_num, saic_data_t data[], uint16_t num)

説明	フラッシュ・ベリファイ付きメモリデータ書き込み関数。必要に応じて関数内でデータ全消去処理を行なう。
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 data[]: データバッファポインタ num: データ数
グローバル変数	なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM or D_SAIC_ERR_VERIFY
処理内容	エラー処理分岐はフローチャートでは省略



1. 引数エラーの場合 D_SAIC_ERR_PARAM を返し終了
2. "0"以外の書き込み要求があるか検査し
- "0"書き込み要求のみの場合
  2.1. 指定のアドレスに応じて対応するフラッシュ・データ書き込み処理関数を実行
    フラッシュ・データ書き込み処理関数が異常終了した場合、対応するエラーを返し終了
  2.2. 引数 num で指定された要素数分 2.1 処理を実行
  2.3. 書き込んだアドレスに対してフラッシュ・データ読み出し処理関数関数を実行
    フラッシュ・データ読み出し処理関数が異常終了した場合、対応するエラーを返し終了
  2.4. ベリファイチェックし、ベリファイエラーならば D_SAIC_ERR_VERIFY を返し終了
  2.5. 引数 num で指定された要素数分 2.3～2.4 処理を実行
- "1"書き込み要求有りの場合
  3.1. フラッシュ・データ読み出し処理関数を実行し FLASH 全メモリデータ読み出し格納
    フラッシュ・データ読み出し処理関数が異常終了した場合、対応するエラーを返し終了
  3.2. フラッシュ・データ全消去処理関数を実行し、FLASH メモリデータ全消去
    フラッシュ・データ全消去処理関数が異常終了した場合、対応するエラーを返し終了
  3.3. 読み出した全データの中から、書き込み対象アドレスのデータのみを書き換え
  3.4. 書き込みデータがフラッシュの初期値”0xFF”データであれば 3.6.へ
  3.5. 指定のアドレスに応じて対応するフラッシュ・データ書き込み処理関数を実行
    フラッシュ・データ書き込み処理関数が異常終了した場合、対応するエラーを返し終了
  3.6. 全アドレスに対して 3.4～3.5 処理を実行
  3.7. フラッシュ・データ読み出し処理関数を実行し全メモリデータ読み出し格納
    フラッシュ・データ読み出し処理関数が異常終了した場合、対応するエラーを返し終了
  3.8. 書き込みデータと読み出しベリファイチェック
    ベリファイエラーならば D_SAIC_ERR_VERIFY を返し終了

6.2.5 フラッシュ・シャドウ領域コピー関数[UART/SPI 共通]

static saic_status_t r_saic_uart_all_flash_to_reg(uint8_t saic_num)

static saic_status_t r_saic_spi_all_flash_to_reg(uint8_t saic_num)

説明	フラッシュ・シャドウ領域コピー関数。フラッシュ・シャドウ領域をレジスタへコピーする。[Register All Write from Flash]コマンドを発行する。
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号
グローバル変数	-UART の場合 g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列 g_uart_rx_end_flag: UART 受信完了フラグ用変数 gs_uart_half_bit_time: ストップビット待ちカウンタ g_uart_serial_data_tbl: UART テーブル -SPI の場合 なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	1.エラーの場合対応するエラーを返し終了 -UART の場合 関数ポインタが NULL の場合 D_SAIC_ERR_COM を返す 2.Register All Write from Flash コマンド生成 3.データ送受信実処理関数を実行し、コマンドを送信する -UART の場合コマンド送信&応答受信関数を実行 -SPI の場合データ送受信実処理関数を実行 4.送受信関数が異常終了した場合、対応するエラーを返し終了 5. Register All Write from Flash 処理完了まで待機する -UART の場合 UART 受信データパケット解析関数を実行する -SPI の場合 ポーリング監視処理関数を実行し、INT 割り込みまたは、RAWIP フラグポーリング 6. 処理完了まで待機する処理が異常終了した場合、対応するエラーを返し終了

6.2.6 フラッシュ・システム設定コピー関数[UART/SPI 共通]

static saic_status_t r_saic_uart_buffer_refresh(uint8_t saic_num)

static saic_status_t r_saic_spi_buffer_refresh(uint8_t saic_num)

説明	フラッシュ・システム設定コピー関数。フラッシュ・システム設定をバッファへコピーする。[Buffer Refresh]コマンドを発行する。
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号
グローバル変数	-UART の場合 g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列 g_uart_rx_end_flag: UART 受信完了フラグ用変数 gs_uart_half_bit_time: ストップビット待ちカウンタ g_uart_serial_data_tbl: UART テーブル -SPI の場合 なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	1.エラーの場合対応するエラーを返し終了 -UART の場合 関数ポインタが NULL の場合 D_SAIC_ERR_COM を返す 2.Buffer Refresh コマンド生成 3.データ送受信実処理関数を実行し、コマンドを送信する -UART の場合コマンド送信&応答受信関数を実行 -SPI の場合データ送受信実処理関数を実行 4.送受信関数が異常終了した場合、対応するエラーを返し終了 5.Buffer Refresh 処理完了まで待機する −UART の場合 UART 受信データパケット解析関数を実行する −SPI の場合 ポーリング監視処理関数を実行し、INT 割り込みまたは、RAWIP フラグポーリング 6.処理完了まで待機する処理が異常終了した場合、対応するエラーを返し終了

6.2.7 INTFLAG レジスタ FR ビット取得処理関数[SPI のみ]

static saic_status_t r_saic_spi_flash_read_fr_bit(uint8_t saic_num, uint8_t *fr_bit)

説明	INTFLAG レジスタ FR ビット取得処理関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *fr_bit: FR ビット戻り値
グローバル変数	なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	1.INTFLAG レジスタをレジスタバイト読み出し関数を実行しデータを読み出す 2.読み出し値の FR ビットの値を引数のポインタへ参照渡しで値を返す

6.2.8 INTFLAG レジスタ FW ビット取得処理関数[SPI のみ]

static saic_status_t r_saic_spi_flash_read_fw_bit(uint8_t saic_num, uint8_t *fw_bit)

説明	INTFLAG レジスタ FW ビット取得処理関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *fw_bit: FW ビット戻り値
グローバル変数	なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	1.INTFLAG レジスタをレジスタバイト読み出し関数を実行しデータを読み出す 2.読み出し値の FW ビットの値を引数のポインタへ参照渡しで値を返す

6.2.9 INTFLAG レジスタ FAE ビット取得処理関数[SPI のみ]

static saic_status_t r_saic_spi_flash_read_fae_bit(uint8_t saic_num, uint8_t *fae_bit)

説明	INTFLAG レジスタ FAE ビット取得処理関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *fae_bit: FAE ビット戻り値
グローバル変数	なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	1.INTFLAG レジスタをレジスタバイト読み出し関数を実行しデータを読み出す 2.読み出し値の FAE ビットの値を引数のポインタへ参照渡しで値を返す

6.2.10 INTFLAG レジスタ RAW ビット取得処理関数[SPI のみ]

static saic_status_t r_saic_spi_flash_read_raw_bit(uint8_t saic_num, uint8_t *raw_bit)

説明	INTFLAG レジスタ RAW ビット取得処理関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *raw_bit: RAW ビット戻り値
グローバル変数	なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	1.INTFLAG レジスタをレジスタバイト読み出し関数を実行しデータを読み出す 2.読み出し値の RAW ビットの値を引数のポインタへ参照渡しで値を返す

6.2.11 STATUS レジスタ FWIP ビット取得処理関数[SPI のみ]

static saic_status_t r_saic_spi_flash_read_fwip_bit(uint8_t saic_num, uint8_t *fwip_bit)

説明	STATUS レジスタ FWIP ビット取得処理関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *fwip_bit: FWIP ビット戻り値
グローバル変数	なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	1.STATUS レジスタをレジスタバイト読み出し関数を実行しデータを読み出す 2.読み出し値の FWIP ビットの値を引数のポインタへ参照渡しで値を返す

6.2.12 STATUS レジスタ FAEIP ビット取得処理関数[SPI のみ]

static saic_status_t r_saic_spi_flash_read_faeip_bit(uint8_t saic_num, uint8_t *faeip_bit)

説明	STATUS レジスタ FAEIP ビット取得処理関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *faeip_bit: FAEIP ビット戻り値
グローバル変数	なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	1.STATUS レジスタをレジスタバイト読み出し関数を実行しデータを読み出す 2.読み出し値の FAEIP ビットの値を引数のポインタへ参照渡しで値を返す

6.2.13 STATUS レジスタ RAWIP ビット取得処理関数[SPI のみ]

static saic_status_t r_saic_spi_flash_read_rawip_bit(uint8_t saic_num, uint8_t *rawip_bit)

説明	STATUS レジスタ RAWIP ビット取得処理関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *rawip_bit: RAWIP ビット戻り値
グローバル変数	なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	1.STATUS レジスタをレジスタバイト読み出し関数を実行しデータを読み出す 2.読み出し値の RAWIP ビットの値を引数のポインタへ参照渡しで値を返す

6.3 システム関数仕様

6.3.1 フラッシュ・データ 01H 番地書き込み処理関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_FLASH_WRITE_01H(uint8_t saic_num, uint8_t data)

saic_status_t R_SAIC_SPI_FLASH_WRITE_01H(uint8_t saic_num, uint8_t data)

説明	フラッシュ・データ 01H 番地書き込み処理関数。電源に関する設定を行います。
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 data: 書き込みデータ
グローバル変数	-UART の場合 g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列 g_uart_rx_end_flag: UART 受信完了フラグ用変数 gs_uart_half_bit_time: ストップビット待ちカウンタ g_uart_serial_data_tbl: UART テーブル -SPI の場合 なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	1. エラーの場合対応するエラーを返し終了 -UART の場合 関数ポインタ NULL の場合 D_SAIC_ERR_COM を返す 2. 書き込みデータが 01H 番地の書き換え不可ビット(bit7、6、3、2)に 1 を指定している場合 D_SAIC_ERR_PARAM を返し終了 3. 指定アドレス、書き込みデータを元に対応する通信フォーマットへ変換 4. データ送受信関数を実行 -UART の場合 コマンド送信&応答受信関数を実行 -SPI の場合 データ送受信実処理関数を実行 5. データ送受信関数が異常終了した場合、対応するエラーを返し終了 6. Flash Write 処理が完了まで待機する -SPI の場合ポーリング監視処理関数を実行し、INT 割り込み待ちまたは、FWIP フラグをポーリングする -UART の場合 UART 受信データパケット解析関数を実行する 7. 処理完了まで待機する処理が異常終了した場合、対応するエラーを返し終了

6.3.2 フラッシュ・データ 1FH 番地書き込み処理関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_FLASH_WRITE_1FH(uint8_t saic_num, uint8_t data)

saic_status_t R_SAIC_SPI_FLASH_WRITE_1FH(uint8_t saic_num, uint8_t data)

説明	フラッシュ・データ 1FH 番地書き込み処理関数。起動時動作に関する設定を行います。
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 data: 書き込みデータ
グローバル変数	-UART の場合 g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列 g_uart_rx_end_flag: UART 受信完了フラグ用変数 gs_uart_half_bit_time: ストップビット待ちカウンタ g_uart_serial_data_tbl: UART テーブル -SPI の場合 なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM
処理内容	1. エラーの場合対応するエラーを返し終了 -UART の場合 関数ポインタ NULL の場合 D_SAIC_ERR_COM を返す 2. 書き込みデータが 1FH 番地の書き換え不可ビット(bit7、6、5、3、2)に 1 を指定している場合 D_SAIC_ERR_PARAM を返し終了 3. 指定アドレス、書き込みデータを元に対応する通信フォーマットへ変換 4. データ送受信関数を実行 -UART の場合 コマンド送信&応答受信関数を実行 -SPI の場合 データ送受信実処理関数を実行 5. データ送受信関数が異常終了した場合、対応するエラーを返し終了 6. Flash Write 処理が完了まで待機する -SPI の場合ポーリング監視処理関数を実行し、INT 割り込み待ちまたは、FWIP フラグをポーリングする -UART の場合 UART 受信データパケット解析関数を実行する 7. 処理完了まで待機する処理が異常終了した場合、対応するエラーを返し終了

7. ADC 関連定義

7.1 API 関数仕様

7.1.1 A/D 変換開始処理関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_ADC_Start(uint8_t saic_num)

saic_status_t R_SAIC_SPI_ADC_Start(uint8_t saic_num)

説明	A/D 変換開始処理関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM
処理内容	1.引数エラーの場合 D_SAIC_ERR_PARAM を返し終了 2.関数内部で saic_data_t 変数配列を生成し、ADCCNT レジスタのアドレスをアドレス変数に代入する 3.レジスタバイト読み出し関数を実行する 4.レジスタバイト読み出し関数が異常終了の場合、対応するエラーを返し終了 5.読み出し値の ADSTART ビットに 1 を代入し、レジスタバイト書き込み関数を実行する 6.レジスタバイト書き込み関数が異常終了の場合、対応するエラーを返し終了

7.1.2 A/D 変換終了処理関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_ADC_Stop(uint8_t saic_num)

saic_status_t R_SAIC_SPI_ADC_Stop(uint8_t saic_num)

説明	A/D 変換終了処理関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 gs_uart_half_bit_time: ストップビット待ちカウンタ -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM
処理内容	1.引数エラーの場合 D_SAIC_ERR_PARAM を返し終了 2.関数内部で saic_data_t 変数配列を生成し、ADCCNT レジスタのアドレスをアドレス変数に代入する 3.レジスタ読み出し処理関数を実行する 4.レジスタバイト読み出し関数が異常終了の場合、対応するエラーを返し終了 5.読み出し値の ADSTART ビットに 0 を代入し、レジスタバイト書き込み関数を実行する - UART の場合 書き込んだアドレスをレジスタバイト読み出し関数で 2 回読み出す 2 回目のレジスタバイト読み出し関数が異常終了の場合、対応するエラーを返し終了 2 回目の読み出し値で ADSTART ビットが 1 ならば D_SAIC_ERR_COM を返し終了する 6.レジスタバイト書き込み関数が異常終了の場合、対応するエラーを返し終了

7.1.3 A/D コンバータレジスタ初期設定関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_ADC_InitRegSet(uint8_t saic_num, saic101_adc_t adc_setting[])

saic_status_t R_SAIC_SPI_ADC_InitRegSet(uint8_t saic_num, saic101_adc_t adc_setting[])

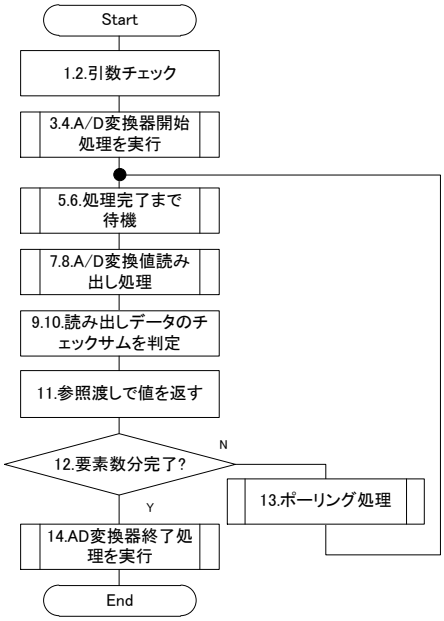
説明	A/D コンバータレジスタ初期設定関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 adc_setting[]: ADC 設定情報格納構造体
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 gs_bit_tbl[]: 8bit ビットテーブル配列 -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数 gs_bit_tbl[]: 8bit ビットテーブル配列
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM
処理内容	1.引数エラーの場合 D_SAIC_ERR_PARAM を返し終了 2.引数 adc_setting を元に対応するレジスタ設定値に変換 3.バースト書き込み処理関数を実行する 4.異常終了の場合、対応するエラーを返し終了

7.1.4 A/D 変換値取得関数(複数チャネル・複数回のデータを取得) [UART/SPI 共通]

saic_status_t R_SAIC_UART_ADC_GetResult
(uint8_t saic_num, uni_adcc_t adcc[], uint16_t adc_value[], uint16_t count)

saic_status_t R_SAIC_SPI_ADC_GetResult
(uint8_t saic_num, uni_adcc_t adcc[], uint16_t adc_value[], uint16_t count)

説明	A/D 変換値取得関数。複数チャネル・複数回のデータを取得する。
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 adcc[]: ADCC レジスタ値格納バッファ。ADC 回数分の領域が必要。 adc_value[]: ADC 値格納バッファ。ADC 回数分の領域が必要。 count: ADC 総回数。全 ch のトータル有効回数。
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列 g_uart_rx_end_flag: UART 受信完了フラグ用変数 g_uart_serial_data_tbl: シリアルモジュール情報格納グローバル変数 -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数 g_spi_saic_data_tbl[]: SAIC 情報格納グローバル変数
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM
処理内容	エラー処理分岐はフローチャートでは省略



1. エラーの場合対応するエラーを返し終了
   -SPI の場合、引数エラーの場合 D_SAIC_ERR_PARAM を返す
   -UART の場合、引数エラーの場合 D_SAIC_ERR_PARAM を返す
   関数ポインタ NULL の場合 D_SAIC_ERR_COM を返す
2. 引数の A/D 回数=0 の場合 SAIC_ERR_PARAM を返す
3. A/D 変換開始処理を実行
   -SPI の場合、A/D 変換開始処理関数を実行
   -UART の場合、レジスタ読み出し処理関数を実行し ADCCNT レジスタを読み出す
   読み出し値の ADSTART ビットに 1 を代入し、レジスタ書き込み処理を実行する
   レジスタ書き込みの返事を受け取るため、UART 受信データパケット解析関数
   (r_saic_uart_get_response)を実行する
4. A/D 変換開始処理が異常終了の場合、対応するエラーを返し終了
5. A/D 変換開始処理の処理完了まで待機する
   -SPI の場合、INT 端子使用時は INT 割り込みを待つ。不使用(ポーリング使用)時は ADCIP フラグポーリング
6. ポーリング監視処理が異常終了の場合、対応するエラーを返し終了
   -SPI の場合、異常終了の場合、対応するエラーを返し終了
7. A/D 値読み出し処理実行し、A/D 変換値を読み出す
   -SPI の場合、バースト読み出し処理関数を実行し、A/D 値を読み出す
   -UART の場合、(A/D 値読みだすため)UART 受信データパケット解析関数
   (r_saic_uart_get_response)を実行する
8. A/D 値読み出し処理が異常終了の場合、対応するエラーを返し終了
9. 読み出しデータのチェックサムを判定
10. チェックサムエラーの場合、D_SAIC_ERR_COM を返し終了
11. ADCC レジスタ値は引数 adcc[]、ADC 値は引数 adc_value[]へ代入する
12. 引数 count で指定された要素数分繰り返す
13. A/D 変換が完了するまでポーリング
   -SPI の場合、INT 端子使用時は処理 5 へ
   INT 端子不使用(ポーリング使用)時は内部関数 ADCIP ビット読み出し処理関数を実行し、ADCIP ビットが 1 になるまでループする。正常完了後処理 5 へ。
   ADCIP ビット読み出し関数が異常終了、またはデッドロックが発生した場合は対応するエラーを返し終了。
14. A/D 変換終了処理関数を実行

7.1.5 A/D 変換値取得関数(単一チャネル・1 回のデータを取得) [UART/SPI 共通]

saic_status_t R_SAIC_UART_ADC_GetResult_1Shot
(uint8_t saic_num, e_adc_ch_t ch, uint16_t *adc_value)

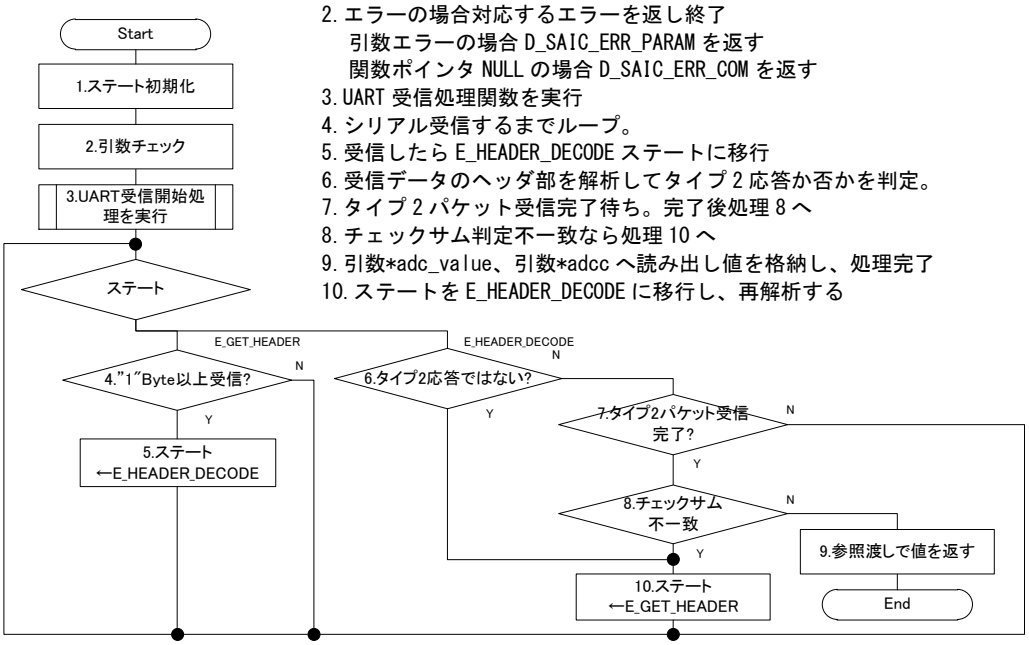
saic_status_t R_SAIC_SPI_ADC_GetResult_1Shot
(uint8_t saic_num, e_adc_ch_t ch, uint16_t *adc_value)

説明	A/D 変換値取得関数。単一チャネル・1 回のデータを取得する。
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 ch: ADC 変換対象 ch 番号。 *adc_value: ADC 値格納バッファ。
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 gs_bit_tbl[]: 8bit ビットテーブル配列 gs_adc_1shot: ADC 1Shot 取得フラグ。1=ADC 1Shot 取得時、0=それ以外 -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数 gs_bit_tbl[]: 8bit ビットテーブル配列
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM
処理内容	1.引数エラーの場合 D_SAIC_ERR_PARAM を返し終了 2.引数 ch を元に ADCCNT レジスタ設定データを作成 3.レジスタバイト書き込み関数を実行し入力マルチプレクサのみ有効に設定 4.レジスタバイト書き込み関数異常終了の場合、対応するエラーを返し終了 5.引数 ch を元に CHxCNT3 設定データを作成 6.レジスタバイト書き込み関数を実行し指定された ch の A/D 変換回数に 1 回を設定。 7.レジスタバイト書き込み関数異常終了の場合、対応するエラーを返し終了 8.A/D 変換、受信処理 -SPI の場合、A/D 変換開始処理関数を実行 A/D 変換開始関数異常終了の場合、対応するエラーを返し終了 A/D 変換開始処理の処理完了まで待機する。 INT 端子使用時は INT 割り込みを待つ。不使用(ポーリング使用)時は ADCIP フラグポーリング 処理完了まで待機した結果が異常終了の場合、対応するエラーを返し終了 バースト読み出し処理関数を実行し、A/D 変換値 A/D 変換値を読み出す バースト読み出し処理関数異常終了の場合、対応するエラーを返し終了 -UART の場合、ADCCNT レジスタをレジスタ読み出し処理関数を実行し読み出す 読み出し値の ADSTART ビットに 1 を代入 gs_adc_1shot 変数に 1 を代入するし コマンド送信&応答受信関数を実行する 9.読み出しデータのチェックサムを判定 10.チェックサムエラーの場合、D_SAIC_ERR_COM を返し終了 11.A/D 変換結果を引数*adc_value に代入

7.1.6 A/D 変換値受信データ取得関数[UART のみ]

saic_status_t R_SAIC_UART_ADC_GetReceive
(uint8_t saic_num, uni_adcc_t *adcc, uint16_t *adc_value)

説明	A/D 変換値受信データ取得関数
ヘッダ	r_sa_uart_control_register.h
引数	saic_num: SAIC 番号 *adcc: ADCC レジスタ値格納バッファ。 *adc_value: ADC 値格納バッファ。
グローバル変数	g_uart_saic_data_tbl_size: SAIC 情報格納数 g_uart_saic_data_tbl[]: SAIC 情報格納グローバル変数 gs_bit_tbl[]: 8bit ビットテーブル配列 g_uart_saic_auto_adc_timeout: OSR=2048 時のセットリングタイム値(8192ms) g_uart_rx_end_flag: UART 受信完了フラグ用変数 g_uart_serial_data_tbl: シリアルモジュール情報格納グローバル変数
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM
処理内容	エラー処理分岐はフローチャートでは省略



1. ステートを E_GET_HEADER に設定
2. エラーの場合対応するエラーを返し終了
   引数エラーの場合 D_SAIC_ERR_PARAM を返す
   関数ポインタ NULL の場合 D_SAIC_ERR_COM を返す
3. UART 受信処理関数を実行
4. シリアル受信するまでループ。
5. 受信したら E_HEADER_DECODE ステートに移行
6. 受信データのヘッダ部を解析してタイプ 2 応答か否かを判定。
7. タイプ 2 パケット受信完了待ち。完了後処理 8 へ
8. チェックサム判定不一致なら処理 10 へ
9. 引数*adc_value、引数*adcc へ読み出し値を格納し、処理完了
10. ステートを E_HEADER_DECODE に移行し、再解析する

7.2 内部関数仕様

7.2.1 A/D 変換値チェックサム値判定関数[UART/SPI 共通]

static saic_status_t r_saic_uart_adc_checksum(uint8_t adc_data[])

static saic_status_t r_saic_spi_adc_checksum(saic_data_t adc_data[])

説明	A/D 変換値チェックサム値判定関数
引数	adc_data[]: 3 バイトの ADC 値先頭アドレス。
グローバル変数	なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM
処理内容	1.引数 saic_data から 3 バイトを各共用体に代入する 2.ADC 値からチェックサムを算出する 3.ADCC レジスタのチェックサム値と比較する 4.チェックサム値一致ならば D_SAIC_OK を、不一致ならば D_SAIC_ERR_COM を返し終了

7.2.2 INTFLAG レジスタ ADC ビット取得処理関数[SPI のみ]

static saic_status_t r_saic_spi_adc_read_adc_bit(uint8_t saic_num, uint8_t *adc_bit)

説明	INTFLAG レジスタ ADC ビット取得処理関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *adc_bit: ADC ビット戻り値
グローバル変数	なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM
処理内容	1.INTFLAG レジスタをレジスタバイト読み出し関数を実行しデータを読み出す 2.読み出し値の ADC ビットの値を引数のポインタへ参照渡しで値を返す

7.2.3 STATUS レジスタ ADCIP ビット取得処理関数[SPI のみ]

static saic_status_t r_saic_spi_adc_read_adcip_bit(uint8_t saic_num, uint8_t *adcip_bit)

説明	STATUS レジスタ ADCIP ビット取得処理関数
ヘッダ	r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *adcip_bit: ADCIP ビット戻り値
グローバル変数	なし
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM
処理内容	1.STATUS レジスタをレジスタバイト読み出し関数を実行しデータを読み出す 2.読み出し値の ADCIP ビットの値を引数のポインタへ参照渡しで値を返す

8. 電源関連定義

8.1 API 関数仕様

8.1.1 AREG オン設定関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_AregOn(uint8_t saic_num)

saic_status_t R_SAIC_SPI_AregOn(uint8_t saic_num)

説明	AREG オン設定関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 g_uart_1800us_nop_cnt: AREG 動作後安定待ち用のカウンタ値(1800us 以上) -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数 g_spi_1800us_nop_cnt: AREG 動作後安定待ち用のカウンタ値(1800us 以上)
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM
処理内容	1.引数エラーの場合 D_SAIC_ERR_PARAM を返し終了 2.指定アドレスへレジスタバイト読み出し関数を実行し CHIPCNT レジスタのデータを読み出す 3.異常終了の場合、対応するエラーを返し終了 4. CHIPCNT レジスタ読み出し値から送信データを作成 1 コマンド目 AREGPD ビットを 0、SLP ビットを 1 2 コマンド目 AREGPD ビットを 0、SLP ビットを 1(1 コマンド目と同一コマンド) 3 コマンド目 SLP ビットを 0 5.レジスタバイト書き込み関数を実行しデータを書き込む 6.異常終了の場合、対応するエラーを返し終了 7. AREG 動作後安定待ち用のカウンタ値の回数分 NOP 命令を実行し 1800us 以上待つ

8.1.2 AREG オフ設定関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_AregOff(uint8_t saic_num)

saic_status_t R_SAIC_SPI_AregOff(uint8_t saic_num)

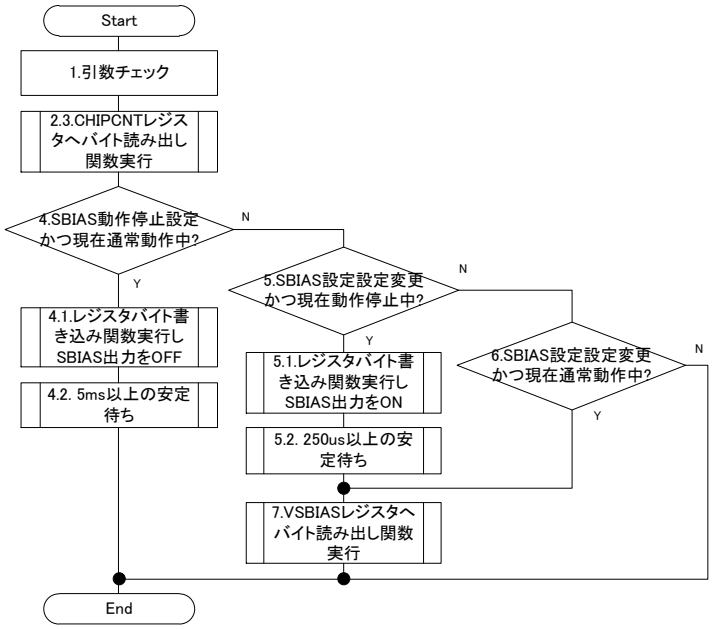
説明	AREG オフ設定関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM
処理内容	1.引数エラーの場合 D_SAIC_ERR_PARAM を返し終了 2.指定アドレスへレジスタバイト読み出し関数を実行し CHIPCNT レジスタのデータを読み出す 3.異常終了の場合、対応するエラーを返し終了 4.レジスタ読み出し値から、送信データを作成 1 コマンド目 AREGPD ビットを 1、SLP ビットを 1 2 コマンド目 AREGPD ビットを 1、SLP ビットを 1(1 コマンド目と同一コマンド) 3 コマンド目 SLP ビットを 0 5.レジスタバイト書き込み関数を実行しデータを書き込む 6.異常終了の場合、対応するエラーを返し終了

8.1.3 SBIAS レジスタ設定関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_SbiasRegSet(uint8_t saic_num, e_sbias_t sbias)

saic_status_t R_SAIC_SPI_SbiasRegSet(uint8_t saic_num, e_sbias_t sbias)

説明	SBIAS レジスタ設定関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 sbias: SBIAS 出力電圧設定
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 g_uart_5ms_nop_cnt: SBIAS 停止,スリープ・モード移行後安定待ち用のカウンタ値(5ms 以上) g_uart_250us_nop_cnt: SBIAS 動作後安定待ち用のカウンタ値(250us 以上) -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数 g_spi_5ms_nop_cnt: SBIAS 停止,スリープ・モード移行後安定待ち用のカウンタ値(5ms 以上) g_spi_250us_nop_cnt: SBIAS 動作後安定待ち用のカウンタ値(250us 以上)
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM
処理内容	エラー処理分岐はフローチャートでは省略



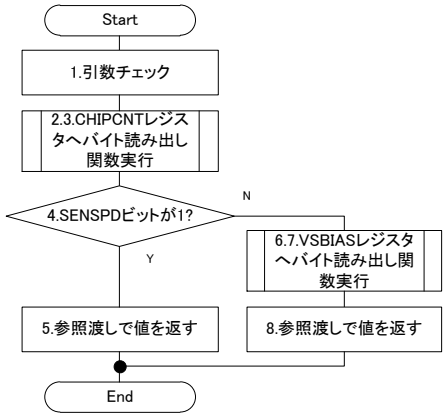
1. 引数エラーの場合 D_SAIC_ERR_PARAM を返し終了
2. レジスタバイト読み出し関数を実行し CHIPCNT レジスタのデータを読み出す
3. 異常終了の場合、対応するエラーを返し終了
4. 引数 sbias が E_ADC_SBIAS_Op0 の時かつ読み出した SENSPD ビットが 0 であった場合
   4.1. 読み出し値の SENSPD ビットに 1 を代入し、レジスタ書き込み処理関数を実行する。SBIAS は動作停止
   4.2. SBIAS 停止,スリープ・モード移行後安定待ち用のカウンタ値の回数文 NOP 命令を実行し、安定時間 5ms 以上待ち処理完了
5. 引数 sbias が E_ADC_SBIAS_Op0 以外の時かつ読み出した値の SENSPD ビットが 1 であった場合
   5.1. 読み出し値に SENSPD ビットをットをットを 0 を代入し、レジスタ書き込み処理関数を実行する。SBIAS は通常動作
   5.2. SBIAS=ON 後安定待ち用のカウンタ値の回数分 NOP 命令を実行し、安定時間 250us 以上待ち 7 処理へ
6. 引数 sbias が E_ADC_SBIAS_Op0 以外の時かつ読み出した値の SENSPD ビットが 0 であった場合 7 処理へ
   それ以外の場合は処理完了
7. レジスタバイト書き込み関数を実行し引数 sbias データを VSBIAS レジスタへ書き込む

8.1.4 SBIAS レジスタ取得関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_SbiasRegGet(uint8_t saic_num, e_sbias_t *sbias)

saic_status_t R_SAIC_SPI_SbiasRegGet(uint8_t saic_num, e_sbias_t *sbias)

説明	SBIAS レジスタ取得関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号 *sbias: SBIAS 出力電圧設定戻り値
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM
処理内容	エラー処理分岐はフローチャートでは省略



1. 引数エラーの場合 D_SAIC_ERR_PARAM を返し終了
2. レジスタバイト読み出し関数を実行し CHIPCNT レジスタのデータを読み出す
3. 異常終了の場合、対応するエラーを返し終了
4. SENSPD ビット値判定
5. SENSPD ビット値が 1 の場合、引数*sbias に E_ADC_SBIAS_Op0 を代入
6. SENSPD ビット値が 0 の場合、レジスタバイト読み出し関数を実行し、VSBIAS レジスタのデータを読み出す
7. 異常終了の場合、対応するエラーを返し終了
8. 引数*sbias に読み出し値を代入

8.1.5 スリープモード設定関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_SleepModeOn(uint8_t saic_num)

saic_status_t R_SAIC_SPI_SleepModeOn(uint8_t saic_num)

説明	スリープモード設定関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 g_uart_5ms_nop_cnt: SBIAS 停止,スリープ・モード移行後安定待ち用のカウンタ値(5ms 以上) -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数 g_spi_5ms_nop_cnt: SBIAS 停止,スリープ・モード移行後安定待ち用のカウンタ値(5ms 以上)
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM
処理内容	1.引数エラーの場合 D_SAIC_ERR_PARAM を返し終了 2.レジスタバイト読み出し関数を実行し CHIPCNT レジスタのデータを読み出す 3.異常終了の場合、対応するエラーを返し終了 4.読み出した CHIPCNT レジスタの SLP ビットを 1 に設定 5.レジスタバイト書き込み関数を実行し CHIPCNT レジスタへ書き込む 6.安定時間 5ms 以上待つ

8.1.6 スリープモード解除関数[UART/SPI 共通]

saic_status_t R_SAIC_UART_SleepModeOff(uint8_t saic_num)

saic_status_t R_SAIC_SPI_SleepModeOff(uint8_t saic_num)

説明	スリープモード解除関数
ヘッダ	-UART の場合 r_sa_uart_control_register.h -SPI の場合 r_sa_spi_control_register.h
引数	saic_num: SAIC 番号
グローバル変数	-UART の場合 g_uart_saic_data_tbl_size: SAIC 情報格納数 g_uart_270us_nop_cnt: スリープ・モード復帰後安定待ち(270us 以上) -SPI の場合 g_spi_saic_data_tbl_size: SAIC 情報格納数 g_spi_820us_nop_cnt: スリープ・モード復帰後安定待ち(820us 以上)
戻り値	saic_status_t: D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM
処理内容	1.引数エラーの場合 D_SAIC_ERR_PARAM を返し終了 2.レジスタバイト読み出し関数を実行し CHIPCNT レジスタのデータを読み出す 3.異常終了の場合、対応するエラーを返し終了 4.読み出した CHIPCNT レジスタの SLP ビットを 0 に設定 5.レジスタバイト書き込み関数を実行し CHIPCNT レジスタへ書き込む 6.安定時間以上待つ -UART の場合、270us 以上 -SPI の場合、820us 以上

9. 電源設定

9.1 電源供給構成

9.1.1 構成一覧

SAIC101 の電源電圧は、5.0V のシステム電源のみではなく、3.0V のシステム電源を用いる場合と、接続される MCU のみ 3.0V とする場合の 3 パターンの電源電圧を想定しています。

ただし、SAIC101 に内蔵されたフラッシュ・メモリを書き換える時(フラッシュ・プログラミング時)は 4.5V(構成 2 の場合は 4.6V)～5.5V の電圧の印加が必要となります。表 9.1 に各電源供給構成の一覧を示します。フラッシュ・プログラミングを使用しない場合を「通常起動時」と定義します。

表 9.1 電源供給構成一覧

電源供給構成	動作時電源電圧		通常起動時の レジスタ設定変更	フラッシュ・プログラミング時	
	SAIC101	MCU		電源構成変更	レジスタ設定変更
構成 1	3.3～5.5V (同電位)		不要	不要	不要
構成 2	3.3～5.5V	3.0V	不要	**必要**	**必要**
構成 3	2.7～3.6V (同電位)		**必要**	**必要**	**必要**

9.1.2 電源構成 1 (通常動作時)

SAIC101 および MCU を 3.3V～5.5V の電圧範囲かつ同電位で用いる場合(表 9.1　構成 1 に該当)の電源構成を図 9-1 に示します。

- 通常起動時電源構成

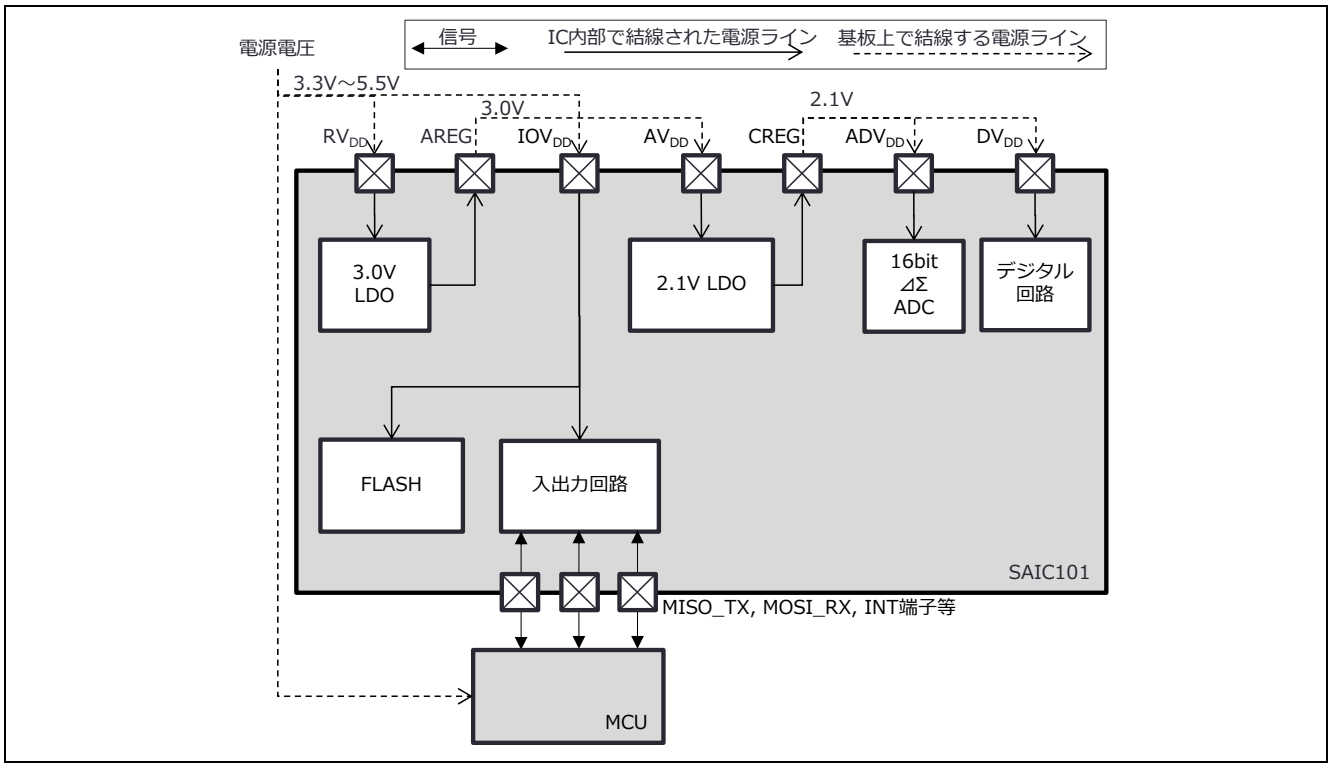


図 9-1 通常起動時電源構成

- 通常起動時のソフトウェア設定：不要

9.1.3 電源構成 1 (フラッシュ・プログラミング時)

SAIC101 および MCU を 3.3V～5.5V の電圧範囲かつ同電位で用いる場合(表 9.1　構成 1 に該当)の電源構成を図 9-2 に示します。

- フラッシュ・プログラミング時の電源構成変更
  電源電圧が 4.5V 未満の場合は、4.5V～5.5V の電圧を印加してください。

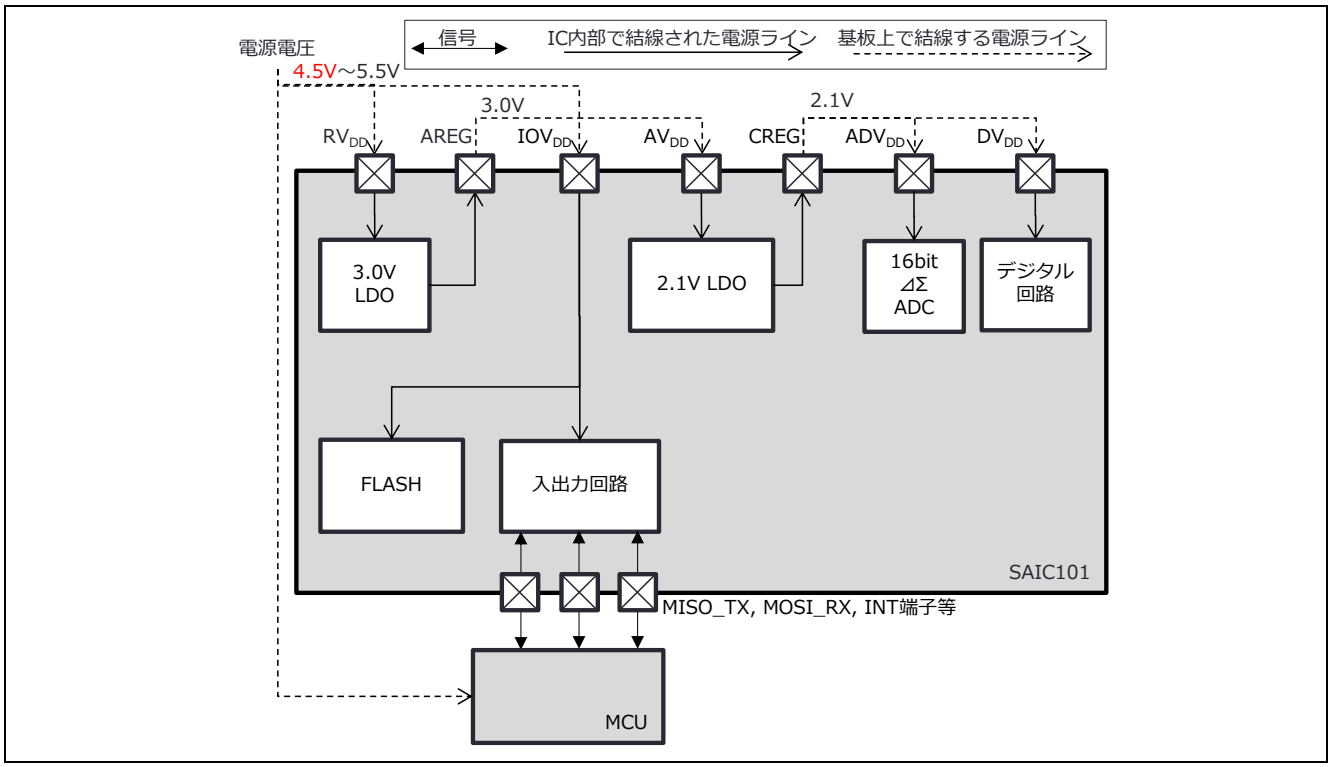


図 9-2　フラッシュ・プログラミング時電源構成

- フラッシュ・プログラミング時のソフトウェア設定：不要

9.1.4 電源構成 2 (通常動作時)

SAIC101 を 3.3V～5.5V の電圧範囲、MCU を 3.0V の電圧と異電位で用いる場合(表 9.1　構成 2 に該当)の電源構成を図 9-3 に示します。

- 通常起動時電源構成

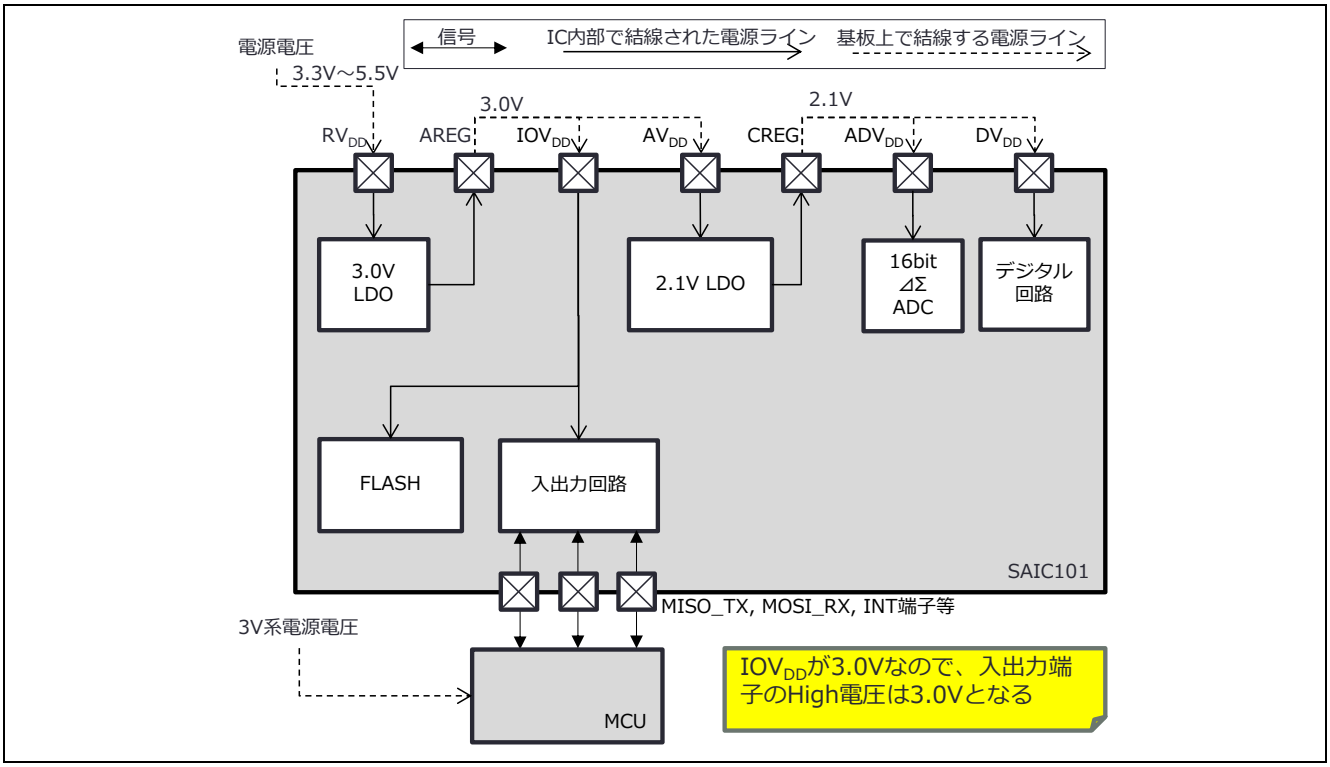


図 9-3　通常起動時電源構成

- 通常起動時のソフトウェア設定：不要

9.1.5 電源構成 2 (フラッシュ・プログラミング時)

SAIC101 を 3.3V～5.5V の電圧範囲、MCU を 3.0V の電圧と異電位で用いる場合(表 9.1　構成 2 に該当)の電源構成を図 9-4 に示します。

- フラッシュ・プログラミング時の電源構成変更

SAIC101 の電源電圧が 4.6V 未満の場合は、4.6V～5.5V の電圧を印加してください。

MCU の電源耐圧が 4.6V 未満の場合は、SAIC101 の電源と切り離すか、異電位対策を行ってください。

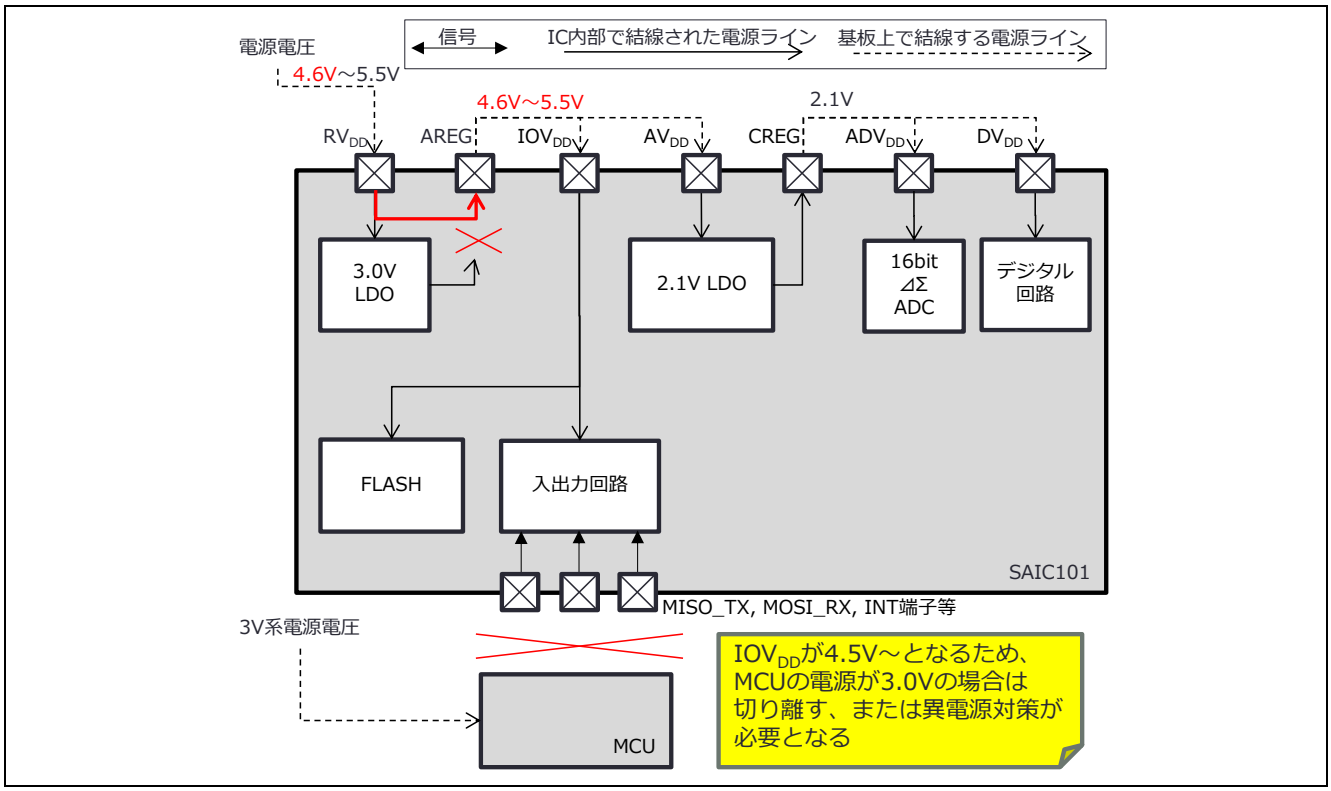


図 9-4　フラッシュ・プログラミング時の電源構成

- フラッシュ・プログラミング時のソフトウェア設定

CHIPCNT レジスタの PSTHRU ビットを 1U に設定し、IC 内部で RV_{DD} 端子と AREG 端子を接続します。

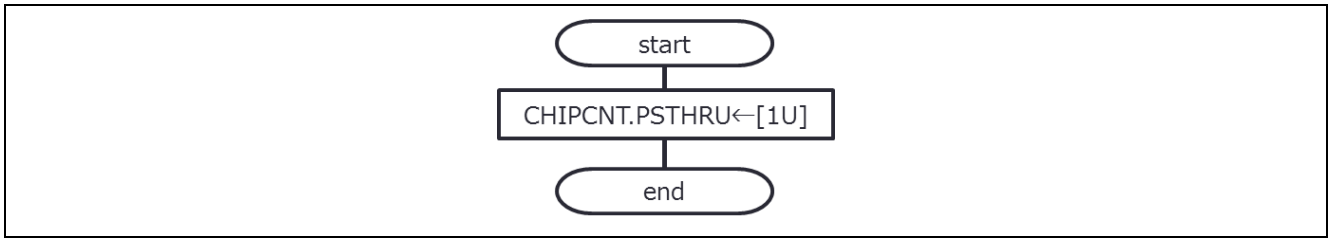


図 9-5　フラッシュ・プログラミング時のソフトウェア設定

PSTHRU ビット設定手順

ソフトウェアにて、CHIPCNT レジスタの PSTHRU ビットを 1U に設定します。

【注】リセット解除後、ADCCNT レジスタの ADSTART ビットを 1U に設定するまで設定が可能。
ADSTART ビットを 1U に設定した後は、次回リセットがかかるまで 0U に固定されます。

9.1.6 電源構成 3 (通常動作時)

SAIC101 および MCU を 2.7V～3.6V の電圧範囲かつ同電位で用いる場合(表 9.1　構成 3 に該当)の電源構成を図 9-6 に示します。

- 通常起動時電源構成

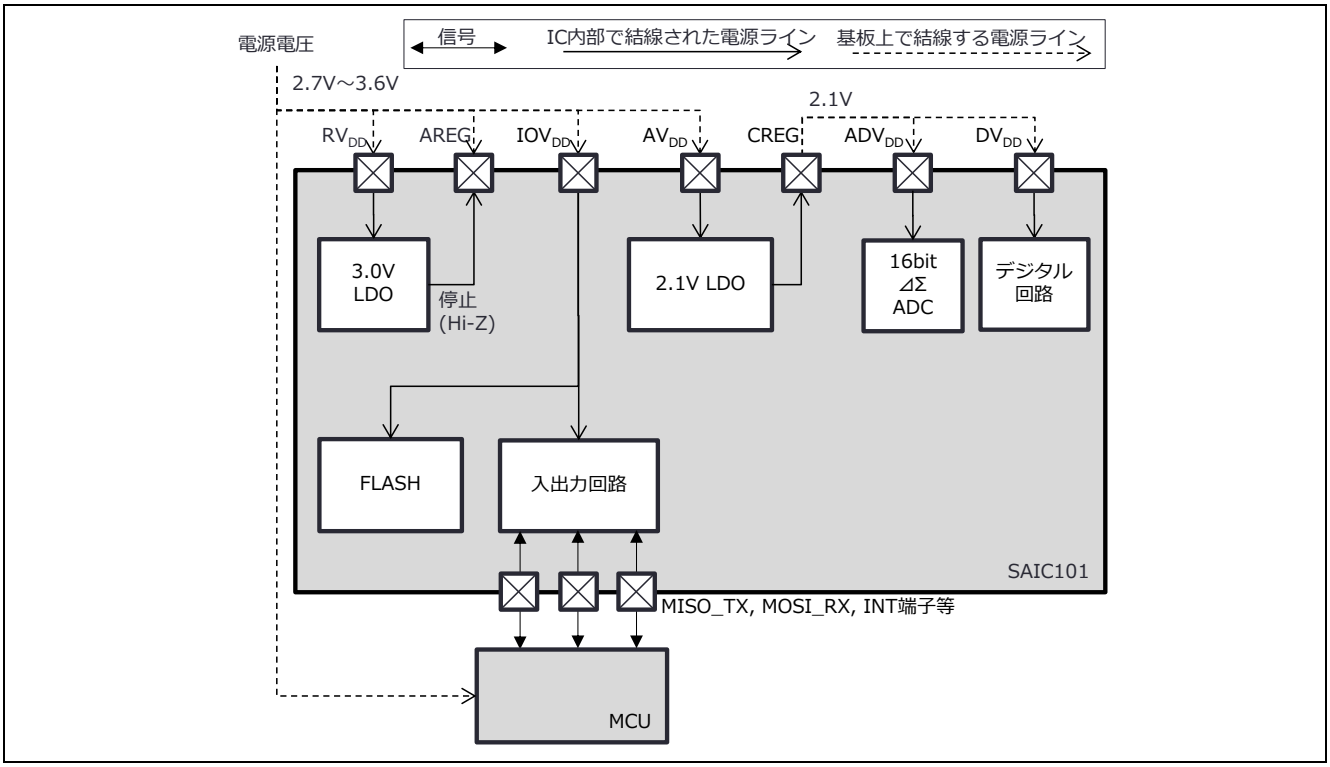


図 9-6　通常起動時電源構成

- 通常起動時のソフトウェア設定

CHIPCNT レジスタの AREGPD ビットを 1U に設定し、AREG の動作を停止します。

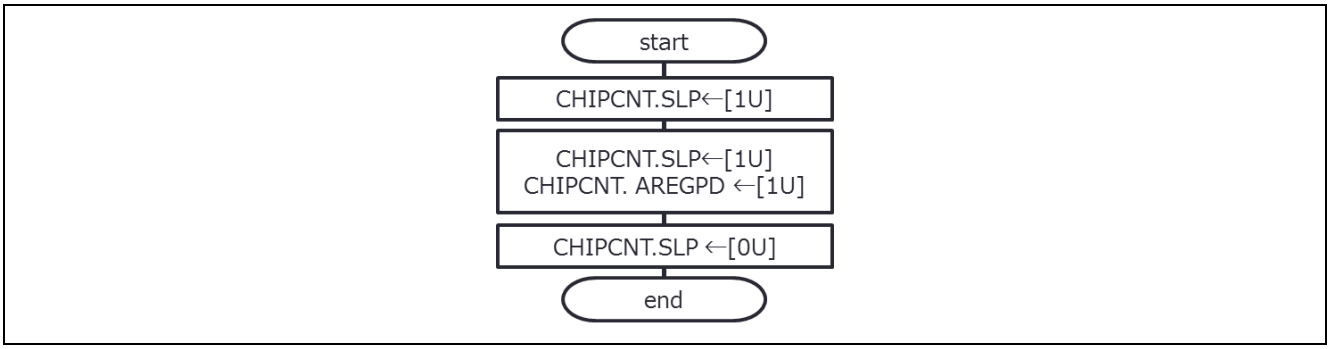


図 9-7　通常起動時のソフトウェア設定

AREG 停止設定手順

CHIPCNT レジスタの AREGPD ビットを 1U に設定する場合、誤書き込み防止のための保護機能を備えているため、SLP ビットを 1U に設定後、SLP ビット 1U、AREGPD ビット 1U を同時に書き込む必要があります。

9.1.7 電源構成 3 (フラッシュ・プログラミング時)

SAIC101 および MCU を 2.7V～3.6V の電圧範囲かつ同電位で用いる場合(表 9.1　構成 3 に該当)の電源構成を図 9-8 に示します。

- フラッシュ・プログラミング時の電源構成変更

SAIC101 の電源電圧が 4.5V 未満の場合は、4.5V～5.5V の電圧を印加してください。

MCU の電源耐圧が 4.5V 未満の場合は、SAIC101 の電源と切り離すか、異電位対策を行ってください。

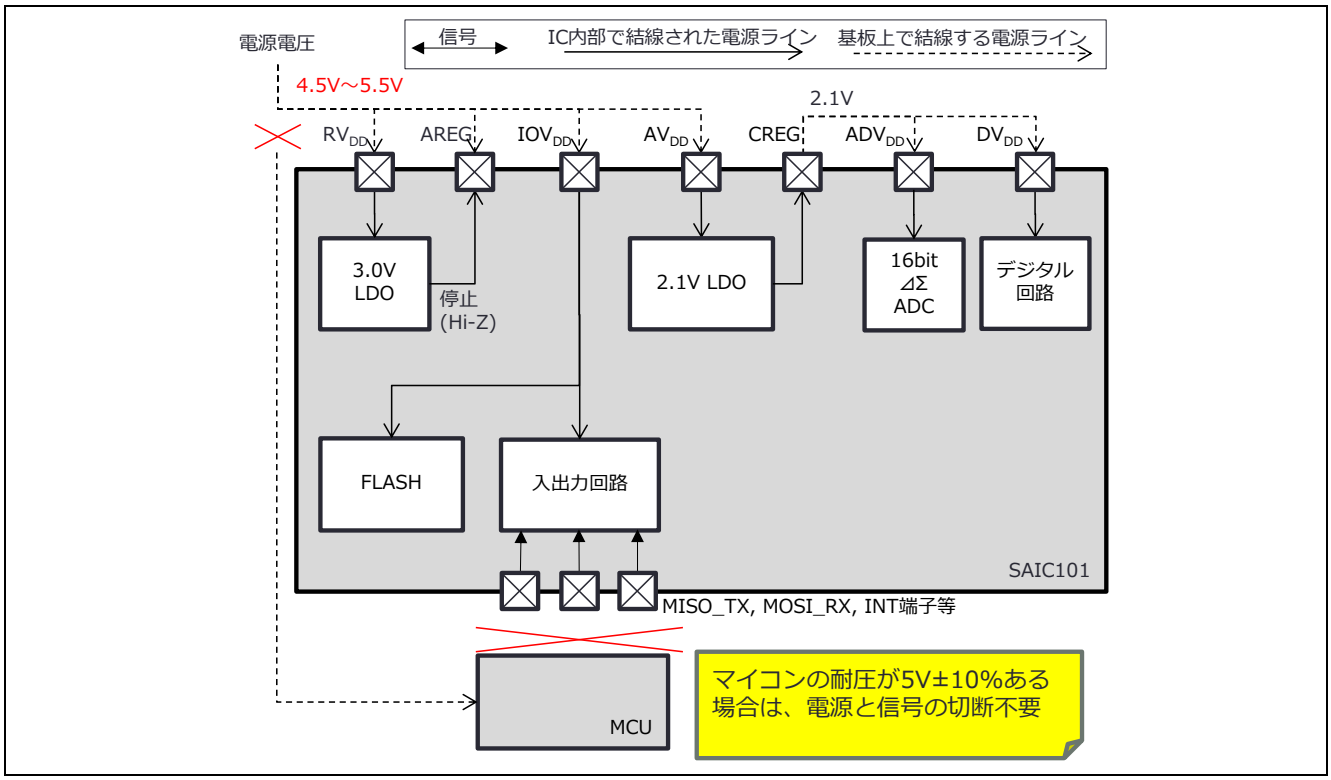


図 9-8　フラッシュ・プログラミング時の電源構成

- フラッシュ・プログラミング時のソフトウェア設定

CHIPCNT レジスタの AREGPD ビットを 1U に設定し、AREG の動作を停止します。

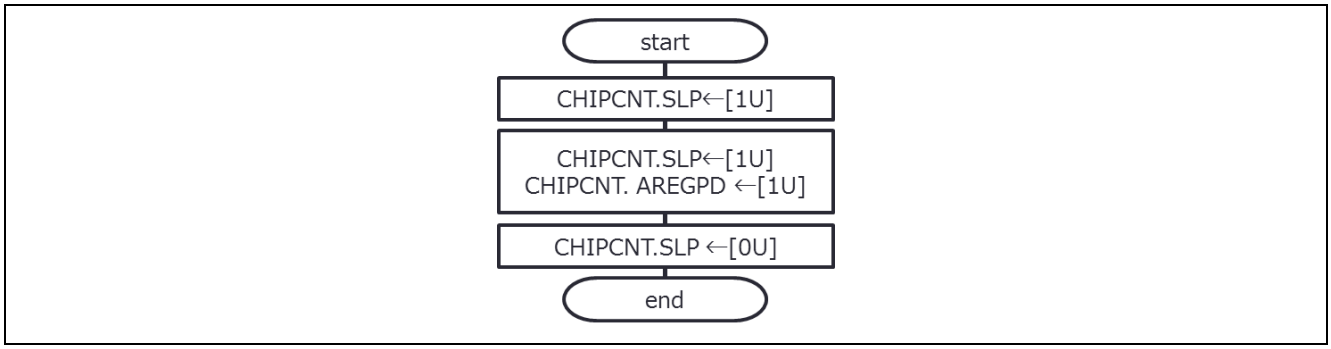


図 9-9　フラッシュ・プログラミング時のソフトウェア設定

AREG 停止設定手順

CHIPCNT レジスタの AREGPD ビットを 1U に設定する場合、誤書き込み防止のための保護機能を備えているため、SLP ビットを 1U に設定後、SLP ビット 1U、AREGPD ビット 1U を同時に書き込む必要があります。

9.2 低消費電力機能

9.2.1 動作モード一覧

SAIC101 は CHIPCNT レジスタの SENSPD ビット、SLP ビットの設定により、消費電力を低減できます。その場合に制限される機能を表 9.2 に示します。

表 9.2 動作モード一覧

動作モード	レジスタ設定		A/D 変換	FLASH への R/W	影響する機能
	SLP ビット	SENSPD ビット			
通常動作	0	0	可能	可能	-
SBIAS 動作停止	0	1	**不可**	可能	• AFE VREF 停止 • A/D コンバータ停止 • D/A コンバータ停止 • 内部バイアス電圧(VBIAS)停止 • センサ用電源回路(SBIAS)停止 • プログラマブル・ゲイン計装アンプ (PGIA)停止
スリープ・モード	1	Don't care	**不可**	**不可**	• 内部基準電圧生成回路(VREF)停止 高精度 BGR 注・アナログ回路用基準電圧生成回路(AFE VREF) • A/D コンバータ停止 • D/A コンバータ停止 • 内部バイアス電圧(VBIAS)停止 • センサ用電源回路(SBIAS) • プログラマブル・ゲイン計装アンプ (PGIA)停止 • CREG の出力電流 2mA に制限かつ 低電力 BGR に切り替え • 内蔵システム・クロック（OSC）用 発振回路停止注

【注】SPI モード時のみ停止

9.2.2 制御モジュール

SLP ビット、SENSPD ビットで制御されるモジュールを図 9-10 に示します。

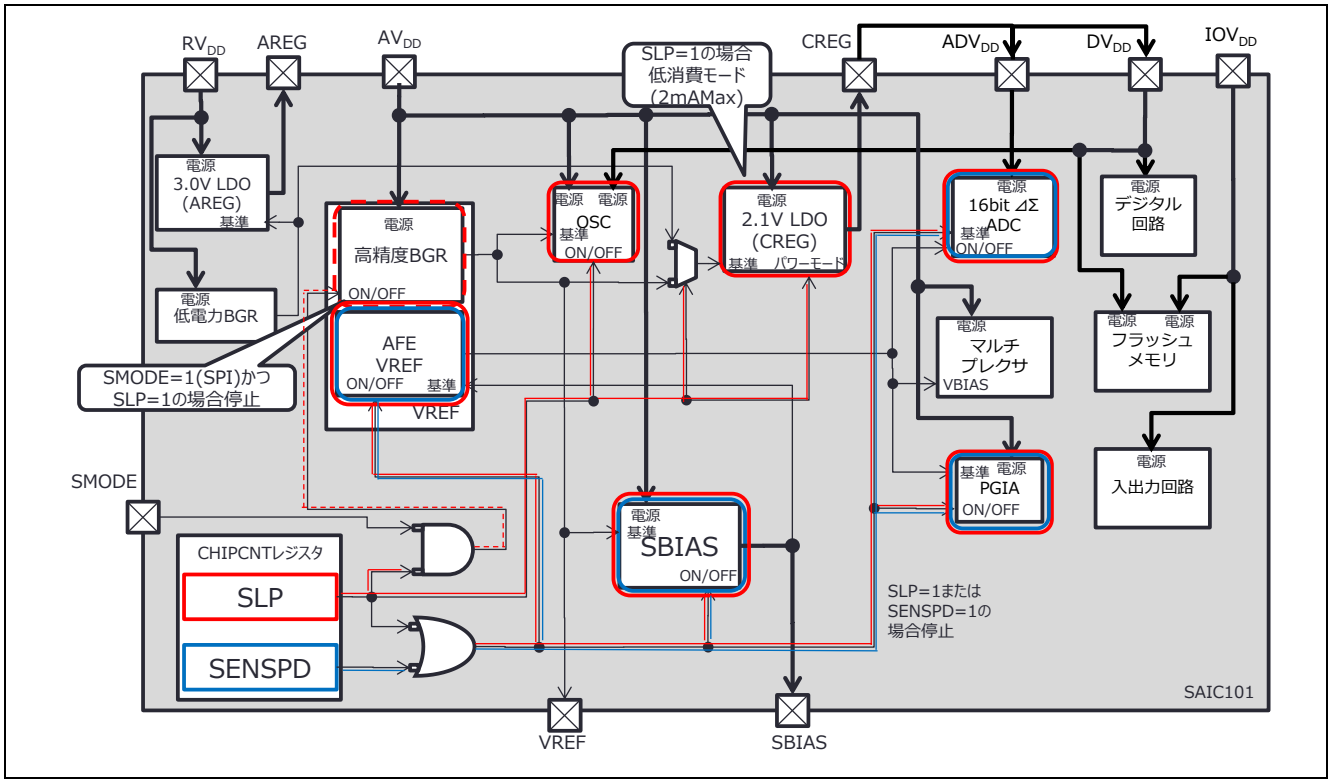


図 9-10 制御モジュール

9.2.3 SBIAS の動作停止/解除方法

- SBIAS の動作停止設定手順

  ソフトウェアにて、CHIPCNT レジスタの SLP ビットを 0U、SENSPD ビットを 1U に設定してください。

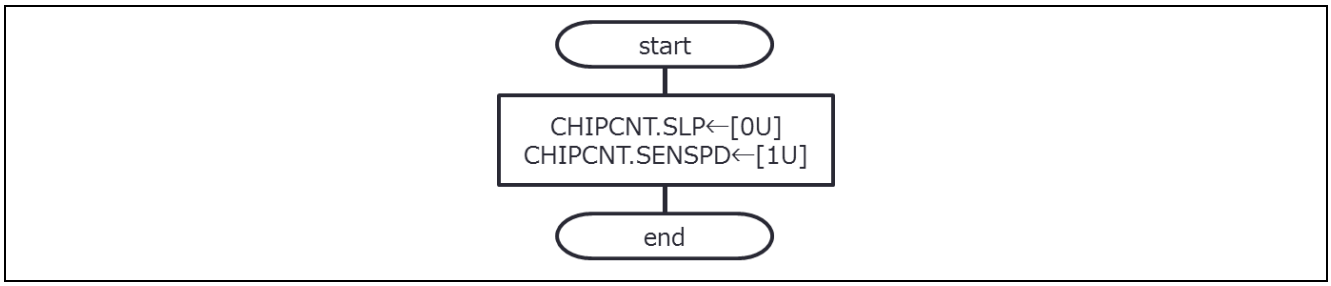


図 9-11 SBIAS の動作停止設定手順

- 遷移タイミング

モジュール	項目	max	単位	条件
SBIAS	ターンオフ時間	(5)	ms	V_{OUT} < 10%

【注】（ ）で示した数値は設計目標値

- SBIAS の動作停止解除手順

  ソフトウェアにて、CHIPCNT レジスタの SLP ビットを 0U、SENSPD ビットを 0U に設定してください。

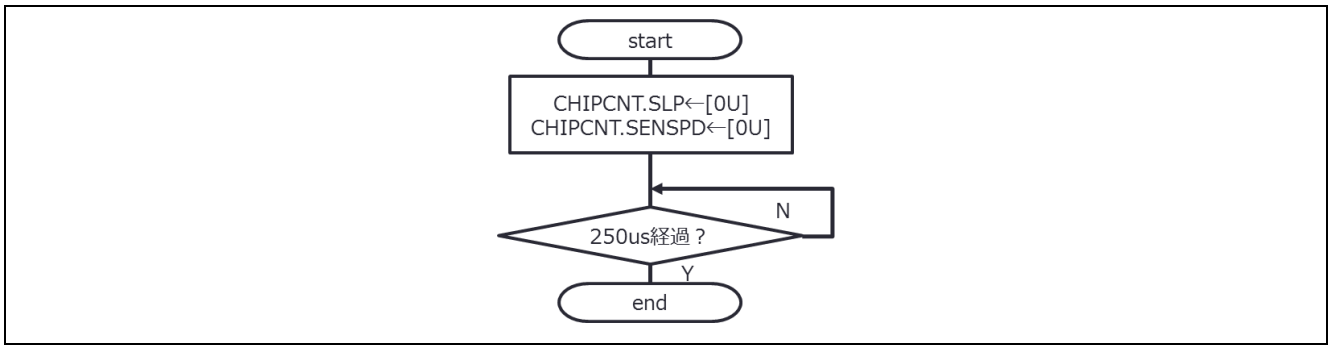


図 9-12 SBIAS の動作停止解除手順

- 遷移タイミング

モジュール	項目	max	単位	条件
SBIAS	ターンオン時間	(250)	μs	V_{OUT} > 90%

【注】（ ）で示した数値は設計目標値

9.2.4 スリープ・モード移行方法

- スリープ・モード移行設定手順

ソフトウェアにて、CHIPCNT レジスタの SLP ビットを 1U に設定設定してください。

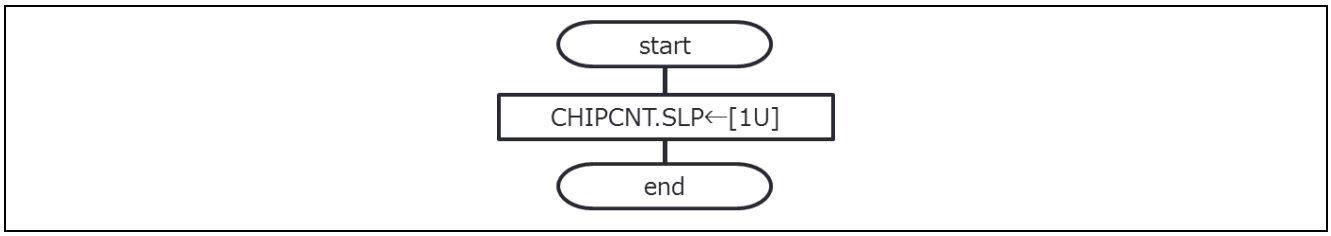


図 9-13 スリープ・モード設定手順

- 遷移タイミング

モジュール	項目	max	単位	条件
SBIAS	ターンオフ時間	(5)	ms	V_{OUT} < 10%
CREG	モード切替え時間 1	(400)	μs	通常動作→待機状態
VREF	-	-	μs	SPI モードのみ停止

【注】（ ）で示した数値は設計目標値

- スリープ・モード復帰設定手順

ソフトウェアにて、 CHIPCNT レジスタの SLP ビットを 0U に設定設定してください。

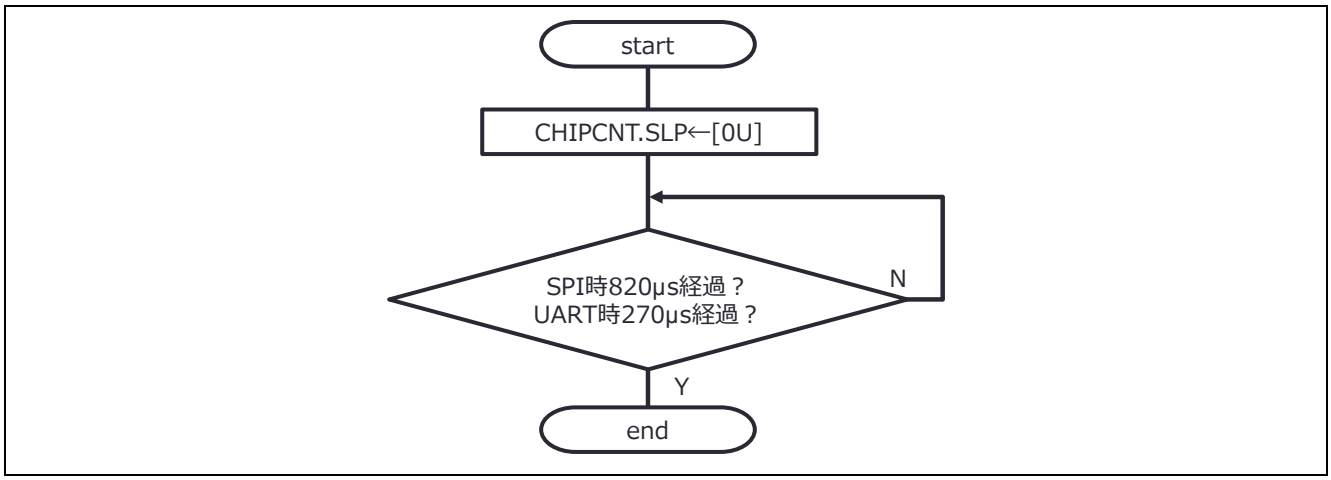


図 9-14 スリープ・モード設定解除手順

- 遷移タイミング

モジュール	項目	max	単位	条件
SBIAS	ターンオン時間	(250)	μs	V_{OUT} > 90%
CREG	モード切替え時間 2	(150)	μs	待機状態→通常動作 VREF ターンオン時間を除く
VREF	ターンオン時間	(550)	μs	SPI モードのみ
全体(SPI)	ウエイク・アップ時間	(820)	μs	SLP = 1U→0U から A/D 変換準備完了まで
全体(UART)	ウエイク・アップ時間	(270)	μs	

【注】（ ）で示した数値は設計目標値

9.2.5 AREG 停止/停止解除方法

- AREG 停止設定手順

ソフトウェアにて、CHIPCNT レジスタの AREGPD ビットを 1U に設定してください。

【注】AREGPD ビットに 1U を設定するには、SLP ビットを 1U に設定後、SLP ビットの 1U、AREGPD ビットの 1U を同時に書き込む必要があります。

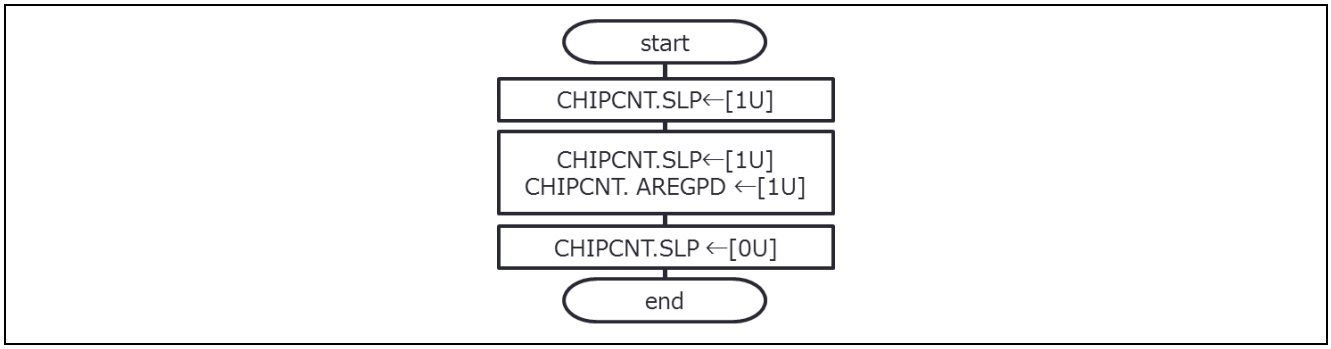


図 9-15 AREG 停止設定手順

- AREG 停止解除手順

ソフトウェアにて、CHIPCNT レジスタの AREGPD ビットを 0U に設定してください。

【注】AREGPD ビットに 0U を設定するには、SLP ビットを 1U に設定後、SLP ビットの 1U、AREGPD ビットの 0U を同時に書き込む必要があります。

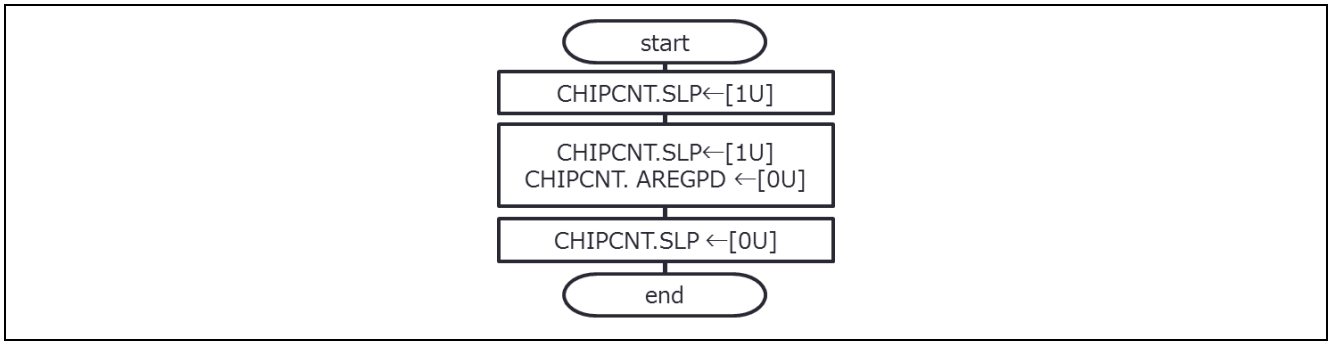


図 9-16 AREG 停止解除手順

- 遷移タイミング

モジュール	項目	max	単位	条件
AREG	ターンオン時間	(1800)	μs	RV_{DD} ≥ 3.3 V, V_{OUT} > 90%, I_{OUT} = 0 mA

【注】（ ）で示した数値は設計目標値

9.3 SAIC101 起動シーケンス一覧

SAIC101 はレジスタ・シャドウの設定によって起動シーケンスが変わり、UART モードの場合には 256Byte 送信[注1]や、A/D 結果送信動作を自動で行います。各設定での起動シーケンスを表 9.3 に示します。

表 9.3 各設定での起動シーケンス

CPSOR[注2] レジスタ・シャドウコピー 0=なし 1=コピー	SDCOR[注2] 256Byte データ送信 0=なし 1=送信	SLP[注2] スリープ・モード制御 0=動作 1=スリープ	SENSPD[注2] SBIAS の動作制御 0=動作 1=停止	ADSTART[注2] A/D 変換動作制御 0=停止 1=開始	SMODE 端子 シリアル・モード選択 L=UART H=SPI	SAIC101 起動時起動後動作 (レジスタ・シャドウ・コピー動作は省略)
0	-	-	-	-	-	-
1	0	0	0	0	-	-
				1	L	A/D 変換動作 A/D 結果自動転送
					H	A/D 変換動作
			1	-	-	SBIAS 動作停止
		1	-	-	-	スリープ・モード
	1[注1]	0	0	0	L	256Byte 自動送信[注1]
					H	-
				1	L	256Byte 自動送信[注1] A/D 変換動作 A/D 結果自動転送
					H	A/D 変換動作
			1	-	L	256Byte 自動送信[注1] SBIAS 動作停止
					H	SBIAS 動作停止
		1	-	-	L	256Byte 自動送信[注1] スリープ・モード
					H	スリープ・モード

【注 1】本 API では 256Byte 送信(STARTUP レジスタの CPSOR ビットを 1U、SDCOR ビットを 1U)には対応しておりません。

【注 2】CPSOR、SDCOR、SLP、SENSPD、ADSTART はフラッシュ・メモリ領域の対応するレジスタ・シャドウ値のビットを示す。

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
Rev.1.00	2014.09.30	---	初版発行

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. 未使用端子の処理
   【注意】未使用端子は、本文の「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。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】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. リザーブアドレス（予約領域）のアクセス禁止
   【注意】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、将来の機能拡張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。
4. クロックについて
   【注意】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。
5. 製品間の相違について
   【注意】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、内部ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 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に関し、当社は、何らの責任を負うものではありません。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. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、
   防災・防犯装置、各種安全装置等
   当社製品は、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（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。 たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。 なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様が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。
9. 本資料に記載されている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。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

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その総株主の議決権の過半数を直接または間接に保有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。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/