Smart Analog IC101

サンプルコード使用例説明書

R21AN0014JJ0100
Rev.1.00
2014.09.01

要旨

本ドキュメントは、Smart Analog IC101 (RAA730101)を制御するための API 関数を使用したサンプルコードの代表的な使用例について説明します。

注: SAIC101 は Smart Analog IC101 の略称です。

動作確認デバイス

Smart Analog IC 101（品名：RAA730101）、RL78/L13 （品名：R5F10WMGAFB）

目次

1. 概要

本ドキュメントでは、Smart Analog IC101(以下、SAIC101)を制御するための API 関数を使用したサンプルコードのうち、代表的な使用例としてレジスタ操作(リード、ライト)[注]、フラッシュ・メモリ操作(リード、ベリファイライト)、A/D 変換制御について説明します。また、Renesas Starter Kit に対応した Smart Analog IC 搭載 RSK オプション評価ボード「TSA-OP-IC101」に搭載されたセンサを用いる例についても説明します。

注: SPI を使用したレジスタリードライトに関しては、SAIC101 以外の Smart Analog 製品にも適用可能です。

2. 動作確認条件

本使用例は、下記の条件で動作を確認しています。

表 2-1 動作確認条件

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 略称：RSK CPU ボード - Renesas Starter Kit LCD Application Board V2 略称：LCD 拡張ボード ・Smart Analog IC 搭載 RSK オプション評価ボード [TSA-OP-IC101] 略称：SAIC101 評価ボード
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]
統合開発環境 (e2studio)	V3.0.0.22
C コンパイラ(e2studio)	GNURL78 v14.01

3. 使用例説明

3.1 レジスタ操作

3.1.1 レジスタバイト読み出し（SPI/UART）

本使用例では SAIC101 注のレジスタに対してバイト読み出しを行います。

本使用例では、SAIC101 の CHIPID レジスタ(0x00 番地)を、SAIC101 API のレジスタバイト読み出し関数である[R_SAIC_SPI_Read]関数(SPI 使用時)または [R_SAIC_UART_Read]関数(UART 使用時)を使用して読み出します。CHIPID レジスタは SAIC101 のチップ ID を格納した読み出し専用のレジスタで、0x3A(固定値)が読み出せます。0x3A がリードされることで、シリアル通信の接続及びレジスタ読み出し処理関数の動作が正しいことを確認できます。

注: 本使用例の SPI 向けに関しては、SAIC101 以外の Smart Analog 製品にも適用可能です。

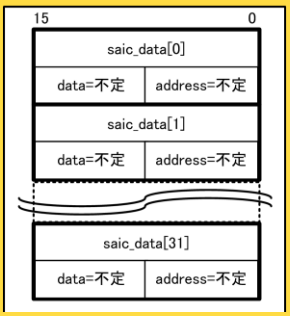
サンプルコード(UART 向けの例)

```
void main(void)
{
  R_MAIN_UserInit();
  {
    // ***
    // * 変数
    // ***
    uint8_t         ret      = D_SAIC_OK;
    uint8_t         saic_num = 0U;
    uint16_t        data_num;
    saic_data_t     saic_data[0x20U];
    uint8_t         err_index;
    saic101_adc_t   adc_setting[0x05U];
    saic_data_t     saic_flash_data[0x100U];
    {
      // ***
      // * レジスタバイト読み出し
      // ***
      // [例:アドレス 0x00U 番地読み出し]
      data_num = 1U;
      saic_data[0x00U].address = 0x00U;
      ret=R_SAIC_UART_Read(saic_num,&saic_data[0x00U],(uint8_t)data_num);
      if (D_SAIC_OK == ret)
      {
        /* 戻り値がD_SAIC_OKの時、saic_data[0x00U].dataに読み出し値が格納される*/
      }
      else
      {
        /* 戻り値がD_SAIC_ERR_COMの時、通信処理エラー。 */
      }
    }
  }
}
```

API 関数戻り値格納変数

API で使用する対象の SAIC 番号

API で使用する SAIC データ構造体



本サンプルでは不使用の変数

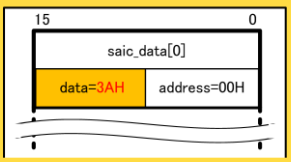
レジスタバイト読み出し関数(API)
SPI使用時はR_SAIC_SPI_Readに読み替える

SAIC101 の CHIP ID を読み出すサンプルコード

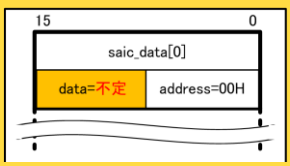
読み出しバイト数の指定
1Byte 読み出しのため 1U を設定

読み出しアドレスの設定
ChipID（0x00 番地）読み出しのため 0x00U を設定

戻り値が D_SAIC_OK の場合,saic_data[0x00U].data に読み出し値が格納される



戻り値がD_SAIC_OK以外の場合,saic_data[0x00U].dataは不定



3.1.2 レジスタバイト書き込み（SPI/UART）

本使用例では SAIC101 [注]のレジスタに対してバイト書き込みを行います。

本使用例では、SAIC101 のレジスタのうち書き込み可能な CH4CNT2 レジスタ(0x1A 番地)に、SAIC101 API のレジスタバイト書き込み関数である[R_SAIC_SPI_Write]関数(SPI 使用時)または [R_SAIC_UART_Write]関数(UART 使用時)を使用して 0x1F を書き込みます。CH4CNT2 レジスタの下位 5 ビットは SAIC101 入力マルチプレクサのチャネル 4 の DC オフセット設定値です。

注: 本使用例の SPI 向けに関しては、SAIC101 以外の Smart Analog 製品にも適用可能です。

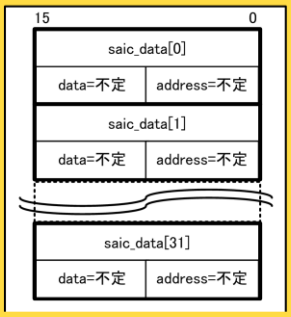
サンプルコード(UART 向けの例)

```
void main(void)
{
  R_MAIN_UserInit();
  {
    // ***
    // * 変数
    // ***
    uint8_t         ret      = D_SAIC_OK;
    uint8_t         saic_num = 0U;
    uint16_t        data_num;
    saic_data_t     saic_data[0x20U];
    uint8_t         err_index;
    saic101_adc_t   adc_setting[0x05U];
    saic_data_t     saic_flash_data[0x100U];
    {
      // ***
      // * レジスタバイト書き込み
      // ***
      // [例:アドレス0x1AU番地に0x1FU書き込み]
      data_num = 1U;
      saic_data[0x00U].address = 0x1AU;
      saic_data[0x00U].data    = 0x1FU;
      ret=R_SAIC_UART_Write(saic_num,&saic_data[0x00U],(uint8_t)data_num);
      if (D_SAIC_OK == ret)
      {
        /* 戻り値がD_SAIC_OKの時、書き込み正常完了。 */
      }
      else
      {
        /* 戻り値がD_SAIC_ERR_COMの時、通信処理エラー。 */
      }
    }
  }
}
```

API 関数戻り値格納変数

API で使用する対象の SAIC 番号

API で使用する SAIC データ構造体



本サンプルでは不使用の変数

SAIC101 のアドレス 0x1AU 番地に 0x1FU 書き込むサンプルコード

レジスタバイト書き込み関数(API) SPI使用時はR_SAIC_SPI_Writeに読み替える

書き込みバイト数の指定
1Byte 書き込みのため 1U を設定

書き込みアドレス設定
0x1A 番地へ書き込みのため 0x1A を設定

書き込みデータの設定
0x1F データを書き込むため、0x1F を設定

戻り値が D_SAIC_OK の場合，書き込み正常完了

戻り値がD_SAIC_OK以外の場合書き込み失敗

3.2 フラッシュ・メモリ操作

3.2.1 フラッシュ・メモリデータ読み出し（SPI/UART）

本使用例では SAIC101 のフラッシュ・メモリのデータの読み出しを行います。

本使用例では、SAIC101 のフラッシュ・メモリのユーザ領域から 0x20～0x22 番地を、SAIC101 API の SAIC101 固有コマンド処理関数である[R_SAIC_SPI_IC101]関数(SPI 使用時)または[R_SAIC_UART_IC101]関数(UART 使用時)を使用して読み出します。

注意: 制約として、フラッシュ・メモリから一度に読み出せるサイズは最大 256 バイトです。また、UART 使用時に 32 バイトを超えるフラッシュ・メモリの読み出しを行う場合は、読み出しバイト数の指定を 32 バイト単位としてください。その他の制約に関しては最新の SAIC101 データシート(R02DS0014J)の「13. フラッシュ・メモリ」の章を参照してください。

サンプルコード(UART 向けの例)

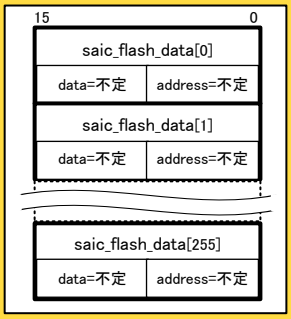
```
void main(void)
{
  R_MAIN_UserInit();
  {
    // ***
    // * 変数
    // ***
    uint8_t         ret      = D_SAIC_OK;
    uint8_t         saic_num = 0U;
    uint16_t        data_num;
    saic_data_t     saic_data[0x20U];
    uint8_t         err_index;
    saic101_adc_t   adc_setting[0x05U];
    saic_data_t     saic_flash_data[0x100U];
    {
      // ***
      // * FLASH メモリデータ読み出し
      // ***
      // [例:アドレス0x20U～0x22U番地読み出し]
      data_num = 3U;
      saic_flash_data[0x00U].address = 0x20U;
      ret=R_SAIC_UART_IC101(saic_num,
                            E_FLASH_READ,
                            &saic_flash_data[0x00U],data_num);
      if (D_SAIC_OK == ret)
      {
        /* 戻り値がD_SAIC_OKの時、以下の様に読み出し値が格納される。 */
      }
      else if (D_SAIC_ERR_PARAM == ret)
      {
        /* 戻り値がD_SAIC_ERR_PARAMの時、パラメータ設定エラー。 */
      }
      else
      {
        /* 戻り値がD_SAIC_ERR_COMの時、通信処理エラー。 */
      }
    }
  }
}
```

API 関数戻り値格納変数

API で使用する対象の SAIC 番号

本サンプルでは不使用の変数

API で使用する SAIC データ構造体



SAIC101 のフラッシュ・メモリを読み出すサンプルコード

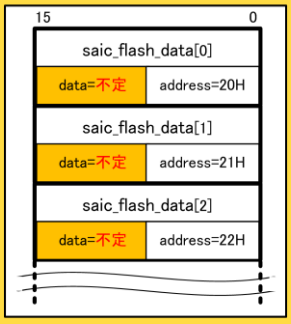
SAIC101 固有コマンド処理関数(API)
SPI 使用時は R_SAIC_SPI_IC101 に読み替える

読み出しバイト数の指定
3Byte 読み出しのため 3U を設定

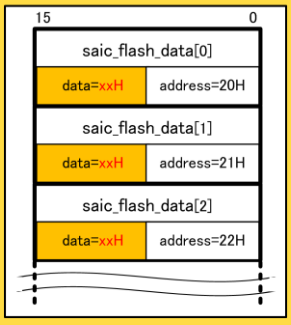
読み出しアドレスの設定
0x20 番地から読み出しのため先頭の 0x20U を設定

フラッシュ・リードののため、
E_FLASH_READ を引数に設定

戻り値が D_SAIC_OK 以外の場合,saic_flash_data[0x00U].data～saic_flash_data[0x02U].data に不定値が格納される



戻り値が D_SAIC_OK の場合,saic_flash_data[0x00U].data～saic_flash_data[0x02U].data に読み出し値が格納される



3.2.2 フラッシュ・メモリベリファイ付きデータ書き込み（SPI/UART）

本使用例では SAIC101 のフラッシュ・メモリへベリファイ付きデータ書き込みを行います。

SAIC101 のフラッシュ・メモリのうちユーザ領域である 0x20 番地から 0xFF 番地までは値を書き換えても SAIC101 の動作に影響を与えません。本使用例では、0x30 番地に 0x55、0x31 番地に 0xAA、0x32 番地に 0x5A を、SAIC101 API の SAIC101 固有コマンド処理関数である[R_SAIC_UART_IC101]関数(SPI 使用時)または[R_SAIC_UART_IC101]関数(UART 使用時)を使用して書き込みます。

注意: 本使用例で使用する SAIC101 API 関数では 1 バイトずつ書き込みを行うため、第 2 引数である SAIC データ構造体には書き込むデータごとに番地指定が必要です。値に 0x00 を書く以外の再書き込み時は全消去が必要です。
パワーオン・リセット後の起動から最初の A/D 変換開始までの間に、プログラミング・ウィンドウ期間が設けてあり、この期間のみ、フラッシュ・メモリ・プログラミングが可能です。その他の制約に関しては最新の SAIC101 データシート(R02DS0014J)の「13. フラッシュ・メモリ」の章を参照してください。

サンプルコード(UART 向けの例)

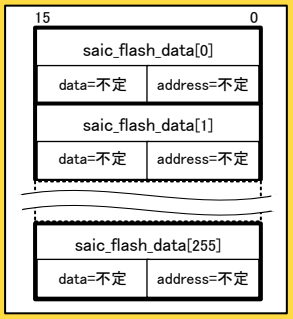
```
void main(void)
{
  R_MAIN_UserInit();
  {
    // ***
    // * 変数
    // ***
    uint8_t         ret      = D_SAIC_OK;
    uint8_t         saic_num = 0U;
    uint16_t        data_num;
    saic_data_t     saic_data[0x20U];
    uint8_t         err_index;
    saic101_adc_t   adc_setting[0x05U];
    saic_data_t     saic_flash_data[0x100U];
    {
      // ***
      // * FLASH ベリファイ付きメモリデータ書き込み
      // ***
      // [例:アドレス0x30番地に0x55U,0x31番地に0xAAU,0x32番地に0x5AU,書き込み]
      data_num = 3U;
      saic_flash_data[0x00U].address = 0x30U;
      saic_flash_data[0x00U].data    = 0x55U;
      saic_flash_data[0x01U].address = 0x31U;
      saic_flash_data[0x01U].data    = 0xAAU;
      saic_flash_data[0x02U].address = 0x32U;
      saic_flash_data[0x02U].data    = 0x5AU;
      ret=R_SAIC_UART_IC101(saic_num,
                            E_FLASH_WRITE_VERIFY,
                            &saic_flash_data[0x00U],data_num);
      if (D_SAIC_OK == ret)
      {
        /* 戻り値がD_SAIC_OKの時、書き込み正常完了。 */
      }
      else if (D_SAIC_ERR_VERIFY == ret)
      {
        /* 戻り値がD_SAIC_ERR_VERIFYの時、ベリファイエラー。 */
      }
      else
      {
        /* 戻り値がD_SAIC_ERR_COMの時、通信処理エラー。 */
      }
    }
  }
}
```

API 関数戻り値格納変数

API で使用する対象の SAIC 番号

本サンプルでは不使用の変数

API で使用する SAIC データ構造体



ベリファイ付きデータ書き込みサンプルコード

書き込みバイト数の指定
3Byte 書き込みのため 3U を設定

SAIC101 固有コマンド処理関数(API)
SPI 使用時は R_SAIC_SPI_IC101 に読み替える

書き込みアドレスとデータの設定
0x30 番地へ、0x55U 書き込み
0x31 番地へ、0xAAU 書き込み
0x32 番地へ、0x5AU 書き込み
を設定

SAIC101 関数固有コマンド用 ENUM 値。ベリファイ付きデータ書き込みのため、E_FLASH_WRITE_VERIFY を引数に設定

戻り値が D_SAIC_OK の場合，書き込み正常完了

戻り値が D_SAIC_OK 以外の場合書き込み失敗

3.3 A/D 変換器の制御

3.3.1 A/D 変換値取得(単一チャネル、単一測定[1shot])（SPI/UART）

本使用例では SAIC101 の A/D 変換器を使用して SAIC101 入力マルチプレクサの単一チャネルに対して A/D 変換を行いデータを取得します。

本使用例では、SAIC101 API の A/D 変換器レジスタ初期設定関数である[R_SAIC_SPI_ADC_InitRegSet]関数(SPI 使用時)または[R_SAIC_UART_ADC_InitRegSet]関数(UART 使用時)、SBIAS レジスタ設定関数である[R_SAIC_SPI_SbiasRegSet]関数(SPI 使用時)または[R_SAIC_UART_SbiasRegSet]関数(UART 使用時)、A/D 変換値取得関数(1shot)である[R_SAIC_SPI_ADC_GetResult_1Shot]関数(SPI 使用時)または[R_SAIC_UART_ADC_GetResult_1Shot]関数(UART 使用時)の 3 関数を使用します。A/D 変換値取得関数(1shot)を 1 度呼び出すと単一チャネル 1 回分の A/D 変換値を取得します。

注意: A/D 変換値取得関数を呼び出す前には必ず、SAIC101 API の A/D 変換器レジスタ初期設定関数を使用して A/D 変換器の設定を行う必要があります。A/D 変換器の設定には 5 チャネル分の saic101_adc_t 構造体配列が必要です。

サンプルコード(UART 向けの例)

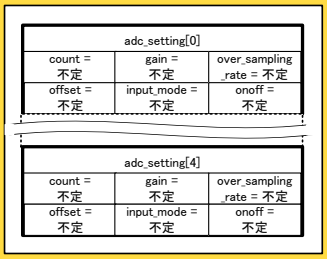
```
void main(void)
{
  R_MAIN_UserInit();
  {
    // ***
    // * 変数
    // ***
    uint8_t         ret      = D_SAIC_OK;
    uint8_t         saic_num = 0U;
    uint16_t        data_num;
    saic_data_t     saic_data[0x20U];
    uint8_t         err_index;
    saic101_adc_t   adc_setting[0x05U];
    saic_data_t     saic_flash_data[0x100U];
    {
      // ***
      // * A/D変換器レジスタ初期設定
      // ***
      //[例:SAIC101入力マルチプレクサ ch1およびch3～ch5無効、ch2のみ有効。設定は以下。]
      uint8_t    count;
      /* 全てのチャネル設定値を初期化 */
      for (count=0U; count<5U; count++)
      {
        adc_setting[count].onoff              = E_ADC_OFF;
        adc_setting[count].input_mode         = E_ADC_DIFF;
        adc_setting[count].offset             = E_ADC_OFFSET_0p00;
        adc_setting[count].over_sampling_rate = E_ADC_OSR_256;
        adc_setting[count].gain               = E_ADC_GAIN_1_1_1;
        adc_setting[count].count              = 0x01U;
      }
      /* ch2のみ個別設定 */
      adc_setting[E_ADC_CH2].gain             = E_ADC_GAIN_1_4_4;
      adc_setting[E_ADC_CH2].offset           = E_ADC_OFFSET_M153p13;
      ret = R_SAIC_UART_ADC_InitRegSet(saic_num, adc_setting);
```

API 関数戻り値格納変数

API で使用する対象の SAIC 番号

本サンプルでは不使用の変数

API で使用する ADC 情報格納変数

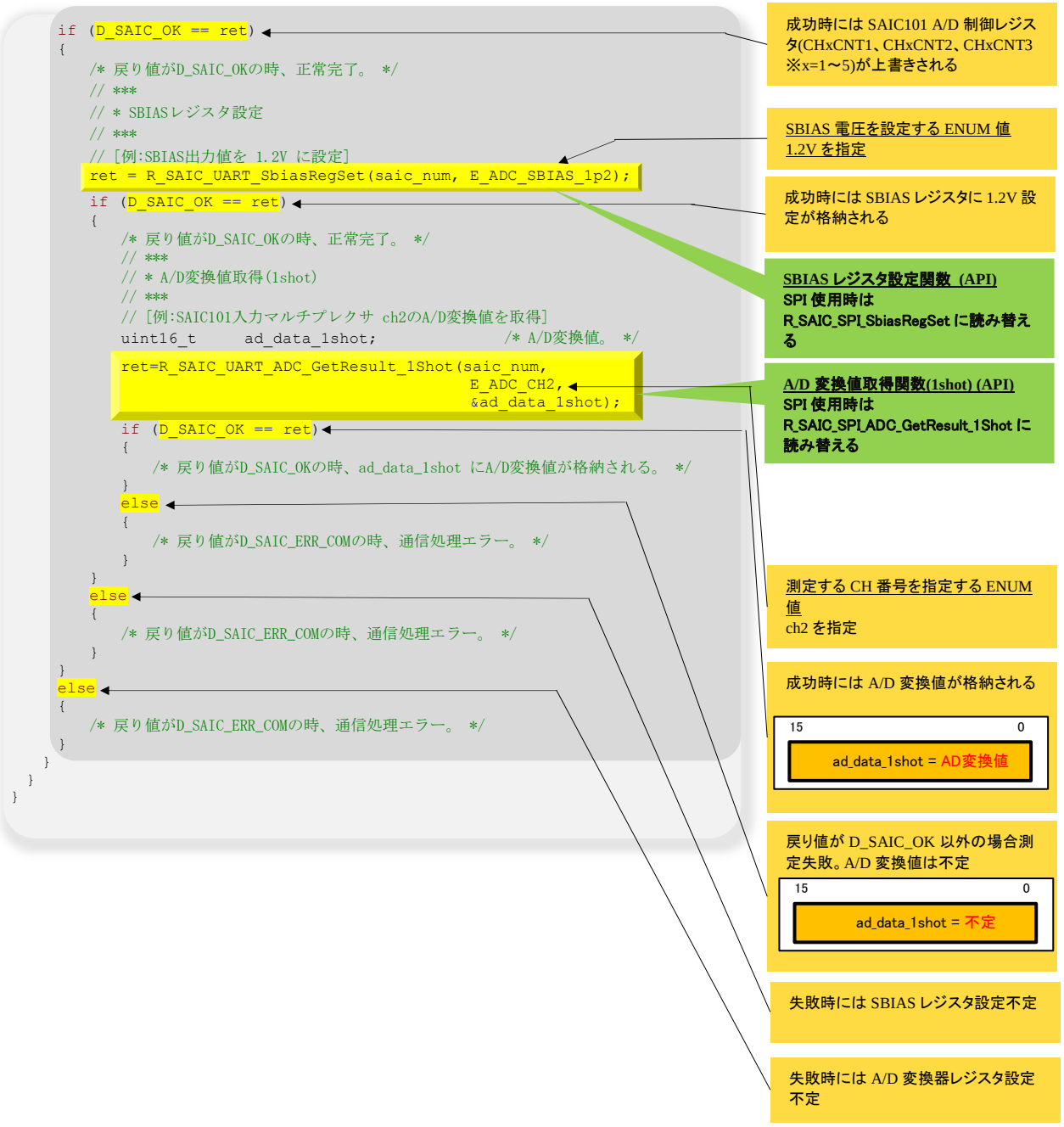


A/D 変換値取得[単一チャネル(ch2)・ワンショット] のサンプルコード

API で使用する ADC 情報格納変数の設定。初期値を代入する。
詳細はリンク参照
・ADC 無効に設定
・差動入力モードに設定
・DC オフセット:0mV に設定
・オーバー・サンプリング比:256 に設定
・ゲイン:1 倍に設定
・AD 変換回数:1 回に設定

API で使用する ADC 情報格納変数の設定。Ch2 の設定。
・DC オフセット:-153.13/GSET1 [mV] に設定
・ゲイン 1x4=4 倍に設定

A/D 変換器レジスタ初期設定関数 (API)
SPI 使用時は R_SAIC_SPI_ADC_InitRegSet に読み替える


if (D_SAIC_OK == ret)
{
/* 戻り値がD_SAIC_OKの時、正常完了。 */
// ***
// * SBIASレジスタ設定
// ***
// [例:SBIAS出力値を 1.2V に設定]
ret = R_SAIC_UART_SbiasRegSet(saic_num, E_ADC_SBIAS_1p2);
if (D_SAIC_OK == ret)
{
/* 戻り値がD_SAIC_OKの時、正常完了。 */
// ***
// * A/D変換値取得(1shot)
// ***
// [例:SAIC101入力マルチプレクサ ch2のA/D変換値を取得]
uint16_t ad_data_1shot; /* A/D変換値。 */
ret=R_SAIC_UART_ADC_GetResult_1Shot(saic_num,
E_ADC_CH2,
&ad_data_1shot);
if (D_SAIC_OK == ret)
{
/* 戻り値がD_SAIC_OKの時、ad_data_1shot にA/D変換値が格納される。 */
}
else
{
/* 戻り値がD_SAIC_ERR_COMの時、通信処理エラー。 */
}
}
else
{
/* 戻り値がD_SAIC_ERR_COMの時、通信処理エラー。 */
}
}
else
{
/* 戻り値がD_SAIC_ERR_COMの時、通信処理エラー。 */
}
}
}
}
成功時には SAIC101 A/D 制御レジスタ(CHxCNT1、CHxCNT2、CHxCNT3 ※x=1～5)が上書きされる
SBIAS 電圧を設定する ENUM 値 1.2V を指定
成功時には SBIAS レジスタに 1.2V 設定が格納される
SBIAS レジスタ設定関数 (API)
SPI 使用時は
R_SAIC_SPI_SbiasRegSet に読み替える
A/D 変換値取得関数(1shot) (API)
SPI 使用時は
R_SAIC_SPI_ADC_GetResult_1Shot に読み替える
測定する CH 番号を指定する ENUM 値
ch2 を指定
成功時には A/D 変換値が格納される
15
0
ad_data_1shot = AD変換値
戻り値が D_SAIC_OK 以外の場合測定失敗。A/D 変換値は不定
15
0
ad_data_1shot = 不定
失敗時には SBIAS レジスタ設定不定
失敗時には A/D 変換器レジスタ設定不定

3.3.2 A/D 変換値取得(複数チャネル、連続測定)（SPI/UART）

本使用例では SAIC101 の A/D 変換器を使用して SAIC101 入力マルチプレクサの複数チャネルに対して A/D 変換を行いデータを取得します。

本使用例では、SAIC101 API の A/D 変換器レジスタ初期設定関数である[R_SAIC_SPI_ADC_InitRegSet]関数(SPI 使用時)または[R_SAIC_UART_ADC_InitRegSet]関数(UART 使用時)、SBIAS レジスタ設定関数である[R_SAIC_SPI_SbiasRegSet]関数(SPI 使用時)または[R_SAIC_UART_SbiasRegSet]関数(UART 使用時)、A/D 変換値取得関数である[R_SAIC_SPI_ADC_GetResult]関数(SPI 使用時)または[R_SAIC_UART_ADC_GetResult]関数(UART 使用時)の 3 関数を使用します。A/D 変換値取得関数を 1 度呼び出すと、各チャネルの A/D 変換設定レジスタ 3 に設定した回数分の A/D 変換値を取得します。

本サンプルコードを実行すると、SAIC101 入力マルチプレクサの全チャネルを 2 回ずつ A/D 変換を行い、値を取得します。

注意: A/D 変換値取得関数を呼び出す前には必ず、SAIC101 API の A/D 変換器レジスタ初期設定関数を使用して A/D 変換器の設定を行う必要があります。A/D 変換器の設定には 5 チャネル分の saic101_adc_t 構造体配列が必要です。SAIC101 API の A/D 変換値取得関数の第 2 引数、第 3 引数の配列サイズは全チャネル回数の合計分ご用意ください。

サンプルコード(UART 向けの例)

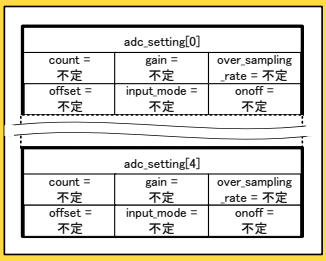
```
void main(void)
{
  R_MAIN_UserInit();
  {
    // ***
    // * 変数
    // ***
    uint8_t         ret      = D_SAIC_OK;
    uint8_t         saic_num = 0U;
    uint16_t        data_num;
    saic_data_t     saic_data[0x20U];
    uint8_t         err_index;
    saic101_adc_t   adc_setting[0x05U];
    saic_data_t     saic_flash_data[0x100U];
    {
      // ***
      // * A/D変換器レジスタ初期設定
      // ***
      // [例:SAIC101入力マルチプレクサ ch1～ch5有効。設定は以下。]
      uint16_t      ad_data[2U * 5U];
      uni_adcc_t    ad_adcc[2U * 5U];

      uint8_t       count;
      for (count=0U; count<5U; count++)
      {
        adc_setting[count].onoff              = E_ADC_ON;
        adc_setting[count].input_mode         = E_ADC_SINGLE;
        adc_setting[count].offset             = E_ADC_OFFSET_0p00;
        adc_setting[count].over_sampling_rate = E_ADC_OSR_128;
        adc_setting[count].gain               = E_ADC_GAIN_1_1_1;
        adc_setting[count].count              = 0x02U;
      }
      /* A/D変換器レジスタ初期設定 */
      ret = R_SAIC_UART_ADC_InitRegSet(saic_num, adc_setting);
```

API 関数戻り値格納変数

API で使用する対象の SAIC 番号

本サンプルでは不使用の変数

API で使用する ADC 情報格納変数



A/D 変換値取得[全チャネル・連続 2 回測定] のサンプルコード

A/D 変換値格納変数
5ch×2 回分

ADCC レジスタ値格納変数
5ch×2 回分

API で使用する ADC 情報格納変数の設定
・ADC 有効
・シングルエンド入力モード
・DC オフセット:0mV
・オーバー・サンプリング比:128
・ゲイン:1 倍
・A/D 変換回数:2 回

A/D 変換器レジスタ初期設定関数(API)
SPI 使用時は R_SAIC_SPI_ADC_InitRegSet に読み替える

```
if (D_SAIC_OK == ret)
{
  /* 戻り値がD_SAIC_OKの時、正常完了。 */
  // ***
  // * SBIASレジスタ設定
  // ***
  // [例:SBIAS出力値を 1.2V に設定]
  ret = R_SAIC_UART_SbiasRegSet(saic_num, E_ADC_SBIAS_1p2);

  if (D_SAIC_OK == ret)
  {
      /* 戻り値がD_SAIC_OKの時、正常完了。 */
      // ***
      // * A/D変換値取得
      // ***
      /* A/D変換値取得 */
      ret=R_SAIC_UART_ADC_GetResult(saic_num,ad_adcc,ad_data,(5U*2U));

      if ((D_SAIC_OK == ret) &&
        (ad_adcc[0x00U].BIT.ch==1U)&&(ad_adcc[0x01U].BIT.ch==1U)&&
        (ad_adcc[0x02U].BIT.ch==2U)&&(ad_adcc[0x03U].BIT.ch==2U)&&
        (ad_adcc[0x04U].BIT.ch==3U)&&(ad_adcc[0x05U].BIT.ch==3U)&&
        (ad_adcc[0x06U].BIT.ch==4U)&&(ad_adcc[0x07U].BIT.ch==4U)&&
        (ad_adcc[0x08U].BIT.ch==5U)&&(ad_adcc[0x09U].BIT.ch==5U))
      {
        /* 戻り値がD_SAIC_OKの時、ad_data にA/D変換値が、
           ad_adcc にADCCレジスタ値が格納される。 */
      }
      else
      {
        /* 戻り値がD_SAIC_ERR_COMの時、通信処理エラー。 */
      }
  }
  else
  {
    /* 戻り値がD_SAIC_ERR_COMの時、通信処理エラー。 */
  }
}
else
{
  /* 戻り値がD_SAIC_ERR_COMの時、通信処理エラー。 */
}
  }
}
```

成功時には SAIC101A/D 制御レジスタ(CHxCNT1、CHxCNT2、CHxCNT3 ※x=1～5)が上書きされる

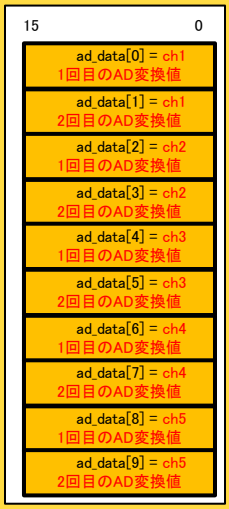
SBIAS 電圧を設定する ENUM 値 1.2V を指定

成功時には SBIAS レジスタに 1.2V 設定が格納される

SBIAS レジスタ設定関数 (API)
SPI 使用時は R_SAIC_SPI_SbiasRegSet に読み替えろ

A/D 変換値取得関数 (API)
SPI 使用時は R_SAIC_SPI_ADC_GetResult に読み替える

成功時には ad_data[0]～ad_data[9]に A/D 値が格納される



成功時には ad_adcc[0]～ad_adcc[9]に ADCC レジスタ値が格納される

戻り値が D_SAIC_OK 以外の場合 ad_adcc[0]～ad_adcc[9]、ad_data[0]～ad_data[9]は不定

失敗時には SBIAS レジスタ設定不定

失敗時には A/D 変換器レジスタ設定不定

3.4 Smart Analog IC 搭載 RSK オプション評価ボード応用例

3.4.1 サーミスタ制御

本使用例は Renesas Starter Kit for RL78/L13 と Smart Analog IC 搭載 RSK オプション評価ボード「TSA-OP-IC101」を接続し、 SAIC101 評価ボード上のサーミスタを使用して温度測定を行います。

SAIC101 評価ボードの SAIC101 入力マルチプレクサのチャネル 2 には、オンボードセンサのサーミスタが接続されています。本サンプルを実行することで、サーミスタ出力の A/D 変換結果を LCD パネル上に表示し、A/D 変換の動作を確認できます。本使用例では、サーミスタの温度が上昇すると A/D 変換値の値は下降します。サーミスタを指でつまむことで LCD パネル上の値が減少することを確認できます。なお、A/D 変換値の範囲は差動入力 16bit のため、－32768～32767 となります。

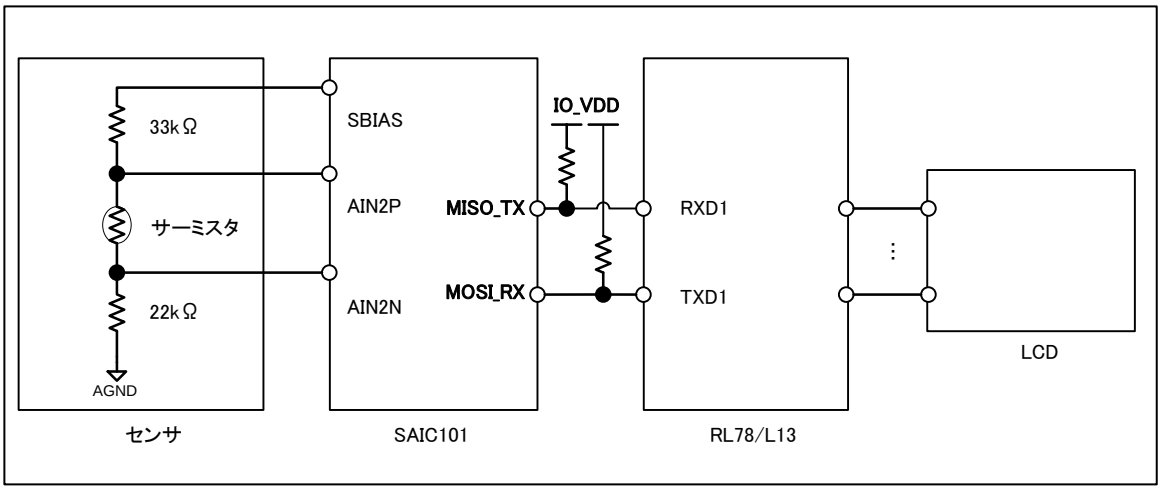


図 3-1 ブロック図

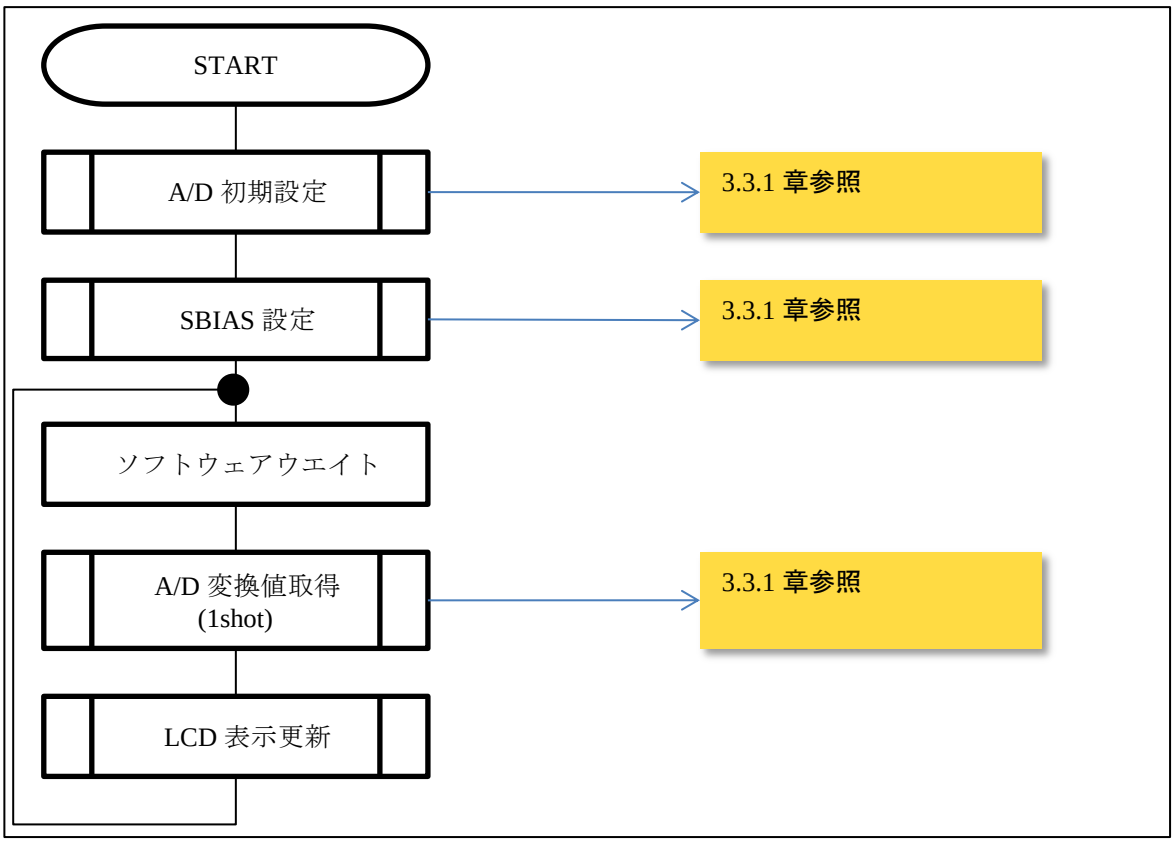


図 3-2 フローチャート

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	2014.09.01	---	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。

RENESAS

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/

Colophon 3.0