RX ファミリ

SSI モジュール Firmware Integration Technology

要旨

本アプリケーションノートは、Firmware Integration Technology (FIT)を使用した SSI (Serial Sound Interface) モジュールについて説明します。本ソフトウェア・モジュールは、SSI を使用した PCM データ送信と受信動作を提供します。

対象デバイス

- RX64M グループ
- RX71M グループ
- RX113 グループ
- RX231 グループ
- RX230 グループ
- RX23W グループ

SSI モジュールは SSI を 2 チャネルサポートします。 有効な SSI チャネル数は対象デバイスにより異なります。

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

対象コンパイラ

・Renesas Electronics C/C++ Compiler Package for RX Family

・GCC for Renesas RX

・IAR C/C++ Compiler for Renesas RX

各コンパイラの動作確認内容については 4.1 動作確認環境を参照してください。

関連ドキュメント

- Firmware Integration Technology ユーザーズマニュアル(R01AN1833)
- ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)

目次

1. 概要

SSI (Serial Sound Interface)モジュールは、SSI を使用した PCM データ送信と受信動作を提供します。 モジュールは複数の関数からなります。それらを適切な手順で使用することで PCM データの送信と受信動作を行うことができます。 本書で「モジュール」と記載された場合、SSI モジュールを示します。

1.1 SSI モジュールの使用について

本モジュールは、外部の DAC と ADC デバイスと指定された転送フォーマットとサンプリング周波数での PCM データの送信と受信の機能を提供します。本書で「Fs」と記載された場合、サンプリング周波数を示します。転送フォーマットは外部デバイスにより異なるので、それらとの接続のため複数の転送フォーマット（例：I²S）とサンプリング周波数が選択可能です。

本モジュールを使用する場合、最初に下記の手順に従いターゲットプロジェクトに組み込んで下さい。

1) 本モジュールをターゲットプロジェクトに組み込みます。

2) アプリケーションに合うよう r_ssi_api_rx_config.h を変更してください。

1.2 API の概要

本モジュールには以下の関数が含まれています。

関数	関数説明
R_SSI_Open ()	指定した SSI チャネルをロックし、r_ssi_api_rx_config.h の設定に基づき初期化します。この関数は SSI を使用する前にチャネル個別に必ず一度実行してください。
R_SSI_Close ()	指定した SSI チャネルのロックを解除します。
R_SSI_Start ()	指定した SSI チャネルの送信と受信動作を許可します。
R_SSI_Stop ()	指定した SSI チャネルの送信と受信動作を禁止します。
R_SSI_Write ()	送信動作のため、指定した SSI チャネルに PCM データを書き込みます。ユーザは送信動作中、この関数を繰り返し実行しなければなりません。
R_SSI_Read ()	受信動作のため、指定した SSI チャネルから PCM データを読み出します。ユーザは受信動作中、この関数を繰り返し実行しなければなりません。
R_SSI_Mute()	送信動作中の SSI チャネルをミュートに設定、または解除します。ミュートの間、指定した SSI チャネルから出力される PCM データは 0 になります。
R_SSI_GetVersion ()	モジュールのバージョンを返します。
R_SSI_GetFlagTxUnderFlow ()	送信アンダフローの状態を返します。状態は指定した SSI チャネルの SSISR の TUIRQ フラグに相当する値です。
R_SSI_GetFlagTxOverFlow ()	送信オーバフローの状態を返します。状態は指定した SSI チャネルの SSISR の TOIRQ フラグに相当する値です。
R_SSI_GetFlagRxUnderFlow ()	受信アンダフローの状態を返します。状態は指定した SSI チャネルの SSISR の RUIRQ フラグに相当する値です。
R_SSI_GetFlagRxOverFlow ()	受信オーバフローの状態を返します。状態は指定した SSI チャネルの SSISR の ROIRQ フラグに相当する値です。
R_SSI_ClearFlagTxUnderFlow ()	指定した SSI チャネルの SSISR の TUIRQ フラグを 0 にクリアします。
R_SSI_ClearFlagTxOverFlow ()	指定した SSI チャネルの SSISR の TOIRQ フラグを 0 にクリアします。
R_SSI_ClearFlagRxUnderFlow ()	指定した SSI チャネルの SSISR の RUIRQ フラグを 0 にクリアします。
R_SSI_ClearFlagRxOverFlow ()	指定した SSI チャネルの SSISR の ROIRQ フラグを 0 にクリアします。

指定した SSI チャネルに上表の関数を使い、下記手順で PCM データ送信と受信動作を行うことができます。

PCM データ送信の基本手順

1) R_SSI_Open () を実行し、指定 SSI チャネルをロックし初期化する。
2) R_SSI_Start () を実行し、指定 SSI チャネルでの PCM データ送信を開始する。
3) R_SSI_Write () を送信データエンプティフラグがセットされるたびに実行し、指定 SSI チャネルの PCM データ送信を行います。
4) R_SSI_Stop ()を実行し、指定 SSI チャネルの PCM データ送信を停止する。
5) R_SSI_Close () を実行し SSI のロックを解除する。

PCM データ送信の基本手順

1) R_SSI_Open () を実行し、指定 SSI チャネルをロックし初期化する。
2) R_SSI_Start () を実行し、指定 SSI チャネルでの PCM データ受信を開始する。
3) R_SSI_Read () を受信データフルフラグがセットされるたび実行し、指定 SSI チャネルの PCM データ受信を行います。
4) R_SSI_Stop ()を実行し、指定 SSI チャネルの PCM データ受信を停止する。
5) R_SSI_Close () を実行し SSI のロックを解除する。

2. API 情報

2.1 ハードウェアの要求

SSI モジュールは、SSI を持つ RX MCU を必要とします。

2.2 ハードウェアリソースの要求

2.2.1 MCU 周辺機能

本モジュールは、SSI 以外の周辺機能を必要としません。

2.2.2 メモリ使用量

本モジュールの ROM サイズ、RAM サイズ、最大使用スタックサイズを下表に示します。1 チャネルデバイス、2 チャネルデバイスから代表して 1 ずつ掲載しています。

ROM (コードおよび定数) と RAM (グローバルデータ) のサイズは、ビルド時の「2.7 コンフィグレーション概要」のコンフィグレーションオプションによって決まります。

下表の値は下記条件で確認しています。

モジュールリビジョン: r_ssi_api_rx rev1.23

コンパイラバージョン: Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
(統合開発環境のデフォルト設定に”-lang = c99”オプションを追加)
GCC for Renesas RX 4.8.4.201801
(統合開発環境のデフォルト設定に”-std = gnu99”オプションを追加)
IAR C/C++ Compiler for Renesas RX version 4.10.1
(統合開発環境のデフォルト設定)

コンフィグレーションオプション: デフォルト設定

ROM、RAM およびスタックのコードサイズ

デバイス	分類	使用メモリ		
		Renesas Compiler	GCC	IAR Compiler
RX231	ROM	**2196 バイト**	**3990 バイト**	**2683 バイト**
	RAM	**0 バイト**	**0 バイト**	**0 バイト**
	スタック	**36 バイト**	**-**	**44 バイト**
RX64M	ROM	**2434 バイト**	**4917 バイト**	**3261 バイト**
	RAM	**0 バイト**	**0 バイト**	**0 バイト**
	スタック	**40 バイト**	**-**	**44 バイト**

2.3 ソフトウェアの要求

本モジュールは、以下の FIT モジュールを必要とします。
ボードサポートパッケージ (r_bsp) v5.10 以上

2.4 サポートされているツールチェーン

本モジュールは、「4.1 動作確認環境」に示すツールチェーンで動作確認を行っています。

2.5 ヘッダファイル

すべての API とそれらをサポートするインタフェースは r_ssi_api_rx_if.h に定義されています。

2.6 整数型

このドライバは ANSI C99 を使用しています。整数は stdint.h で型定義されています。

2.7　コンフィグレーションの概要

SSI モジュールの動作はユーザが r_ssi_rx_config.h を使って設定します。設定はビルド時に反映されます。設定方法は表 1 に示す 2 種類があります。

最初にユーザは 2 種類のうち一つの設定方法を選択します。

次に動作を設定するため、ユーザは表 2 または表 3 の項目ごとに ”()” で括られた番号のうち一つを選択します。

- スタンダードコンフィグレーション
  比較的容易な設定方法で、ユーザは項目ごと設定する値を PCM データ転送で一般に知られている選択肢から選択します。選択した値はユーザのターゲットプロジェクトのビルド時に SSI の I/O レジスタに適した値に変換されます。
- ユーザユニークコンフィグレーション
  ユーザが SSI の I/O レジスタに値を直接設定したい場合、この設定方法を使用してください。

設定方法は表 1 に示すマクロを r_ssi_rx_config.h に定義することで選択されます。

表 1　設定方法

設定方法	r_ssi_rx_config.h に記述するマクロ定義
スタンダードコンフィグレーション - デフォルト	#define　SSI_STANDARD_CONFIG
ユーザユニークコンフィグレーション	#define　SSI_USER_UNIQUE_CONFIG

何れの場合でも I/O レジスタに適切な値を設定するため、設定に先立ち使用するデバイスの「ユーザーズマニュアル　ハードウェア編」を参照してください。

2.7.1　スタンダードコンフィグレーション

表 2 はスタンダードコンフィグレーションの場合にユーザが設定する項目を示します。右の列に示す選択肢から値を選択して下さい。

本設定方法を使用する場合、必ず ”SSI_STANDARD_CONFIG” をマクロ定義してください。

表 2　スタンダードコンフィグレーション時の設定項目

r_ssi_rx_config.h の設定項目	
SSI_CH0_IO_MODE - **デフォルト値 (2)**	SSI チャネル 0 または 1 を送信と受信に設定します (0) 未使用 (1) 受信 (2) 送信 (3) 送受信（チャネル 1 では設定禁止）
SSI_CH1_IO_MODE - **デフォルト値 (0)**	
SSI_CH0_SERIAL_IF_FMT - **デフォルト値 (0)**	SSI チャネル 0 または 1 のシリアルオーディオインタフェースフォーマットを設定します。 (0) I²S (1) 左詰め (2) 右詰め
SSI_CH1_SERIAL_IF_FMT - **デフォルト値 (0)**	

SSI_CH0_DATA_WIDTH - **デフォルト値 (16)**	SSI チャネル 0 または 1 の PCM データ幅を設定します。 (8)　8 ビット (16) 16 ビット (18) 18 ビット (20) 20 ビット (22) 22 ビット (24) 24 ビット
SSI_CH1_DATA_WIDTH - **デフォルト値 (16)**	
SSI_CH0_BCLK - **デフォルト値 (64)**	SSI チャネル 0 または 1 の ビットクロックのレートを設定します。 (16) 16Fs (32) 32Fs (48) 48Fs (64) 64Fs
SSI_CH1_BCLK - **デフォルト値 (64)**	
SSI_MCLK - **デフォルト値 (256)**	マスタクロックの周波数を指定します。設定値は SSI_CHn_BCLK の整数倍でなければなりません。この設定は両 SSI チャネルで共通です。 (16)　16Fs (32)　32Fs :　: (256) 256Fs :　: (8192) 8192Fs
SSI_CH0_CLK_MODE - **デフォルト値 (0)**	SSI チャネル 0 または 1 の BCLK と LRCK クロックが入力か出力かを設定します。SSI がこれらのクロックを出力する場合、マスタモードを選択してください。SSI へクロックを入力する場合スレーブモードを選択してください。 (0) マスタモード (1) スレーブモード
SSI_CH1_CLK_MODE - **デフォルト値 (0)**	
SSI_CH0_TTRG_NUMBER - **デフォルト値 (4)**	送信データエンプティフラグ (TDE) がセットされる条件を設定します。設定値は以下のように PCM データ幅によって異なります。 <u>PCM データ幅 : 8 ビット</u> TDE は送信 FIFO 内の PCM データ数が下記の時にセットされます。 (12) 12 以下 (8)　8 以下 (4)　4 以下 <u>PCM データ幅 : 16 ビット</u> TDE は送信 FIFO 内の PCM データ数が下記の時にセットされます。 (6)　6 以下 (4)　4 以下 (2)　2 以下 <u>PCM データ幅 : 18, 20, 22, 24 ビット</u> TDE は送信 FIFO 内の PCM データ数が下記の時にセットされます。; (3)　3 以下
SSI_CH1_TTRG_NUMBER - **デフォルト値 (4)**	

	(2)　2 以下 (1)　1 以下
SSI_CH0_RTRG_NUMBER - **デフォルト値 (4)**	受信データフルフラグ (RDF) がセットされる条件を設定します。設定値は以下のように PCM データ幅によって異なります。 PCM データ幅 : 8 ビット RDF は受信 FIFO 内の PCM データ数が下記の時にセットされます。 (4)　4 以上 (8)　8 以上 (12) 12 以上 PCM データ幅 : 16 ビット RDF は受信 FIFO 内の PCM データ数が下記の時にセットされます。 (2)　2 以上 (4)　4 以上 (6)　6 以上 PCM データ幅 : 18, 20, 22, 24 ビット RDF は受信 FIFO 内の PCM データ数が下記の時にセットされます。 (1)　1 以上 (2)　2 以上 (3)　3 以上
SSI_CH1_RTRG_NUMBER - **デフォルト値 (4)**	

2.7.2　ユーザユニークコンフィグレーション

表 3 は、ユーザユニークコンフィグレーションの場合にユーザが設定する項目を示します。使用するデバイスの「ユーザーズマニュアル　ハードウェア編」を参照し適切な値を設定してください。

本設定方法を使用する場合、必ず ”SSI_USER_UNIQUE_CONFIG” をマクロ定義してください。

表 3　ユーザユニークコンフィグレーション時の設定項目

r_ssi_rx_config.h の設定項目	
SSI_CH0_IO_MODE - **デフォルト値 (3)**	SSI チャネル 0 または 1 を送信と受信に設定します。 (0)　未使用 (1)　受信 (2)　送信 (3)　送受信（チャネル 1 では設定禁止）
SSI_CH1_IO_MODE - **デフォルト値 (3)**	
SSI_CH0_TTRG - **デフォルト値 (3)**	SSI チャネル 0 または 1 の SSIFCR.TTRG レジスタに値を設定します。 設定値については、使用するデバイスの「ユーザーズマニュアル　ハードウェア編」を参照してください。
SSI_CH1_TTRG - **デフォルト値 (3)**	

SSI_CH0_RTRG - **デフォルト値 (3)**	SSI チャネル 0 または 1 の SSIFCR.RTRG レジスタに値を設定します。 設定値は、使用するデバイスの「ユーザーズマニュアル ハードウェア編」を参照してください。
SSI_CH1_RTRG - **デフォルト値 (3)**	
SSI_CH0_DEL - **デフォルト値 (0)**	SSI チャネル 0 または 1 の SSICR.DEL レジスタに値を設定します。 設定値は、使用するデバイスの「ユーザーズマニュアル ハードウェア編」を参照してください。
SSI_CH1_DEL - **デフォルト値 (0)**	
SSI_CH0_PDTA - **デフォルト値 (0)**	SSI チャネル 0 または 1 の SSICR.PDTA レジスタに値を設定します。 設定値は、使用するデバイスの「ユーザーズマニュアル ハードウェア編」を参照してください。
SSI_CH1_PDTA - **デフォルト値 (0)**	
SSI_CH0_SDTA - **デフォルト値 (0)**	SSI チャネル 0 または 1 の SSICR.SDTA レジスタに値を設定します。 設定値は、使用するデバイスの「ユーザーズマニュアル ハードウェア編」を参照してください。
SSI_CH1_SDTA - **デフォルト値 (0)**	
SSI_CH0_SPDP - **デフォルト値 (0)**	SSI チャネル 0 または 1 の SSICR.SPDP レジスタに値を設定します。 設定値は、使用するデバイスの「ユーザーズマニュアル ハードウェア編」を参照してください。
SSI_CH1_SPDP - **デフォルト値 (0)**	
SSI_CH0_SWSP - **デフォルト値 (0)**	SSI チャネル 0 または 1 の SSICR.SWSP レジスタに値を設定します。 設定値は、使用するデバイスの「ユーザーズマニュアル ハードウェア編」を参照してください。
SSI_CH1_SWSP - **デフォルト値 (0)**	
SSI_CH0_SCKP - **デフォルト値 (0)**	SSI チャネル 0 または 1 の SSICR.SCKP レジスタに値を設定します。 設定値は、使用するデバイスの「ユーザーズマニュアル ハードウェア編」を参照してください。
SSI_CH1_SCKP - **デフォルト値 (0)**	
SSI_CH0_SWL - **デフォルト値 (0)**	SSI チャネル 0 または 1 の SSICR.SWL レジスタに値を設定します。 設定値は、使用するデバイスの「ユーザーズマニュアル ハードウェア編」を参照してください。
SSI_CH1_SWL - **デフォルト値 (0)**	

SSI_CH0_DWL - **デフォルト値 (0)**	SSI チャネル 0 または 1 の SSICR.DWL レジスタに値を設定します。 設定値は、使用するデバイスの「ユーザーズマニュアル ハードウェア編」を参照してください。
SSI_CH1_DWL - **デフォルト値 (0)**	
SSI_CH0_DWSD - **デフォルト値 (0)**	SSI チャネル 0 または 1 の SSICR.DWSD レジスタに値を設定します。 設定値は、使用するデバイスの「ユーザーズマニュアル ハードウェア編」を参照してください。
SSI_CH1_DWSD - **デフォルト値 (0)**	
SSI_CH0_SCKD - **デフォルト値 (0)**	SSI チャネル 0 または 1 の SSICR.SCKD レジスタに値を設定します。 設定値は、使用するデバイスの「ユーザーズマニュアル ハードウェア編」を参照してください。
SSI_CH1_SCKD - **デフォルト値 (0)**	
SSI_CH0_AUCKE - **デフォルト値 (0)**	SSI チャネル 0 または 1 の SSICR.AUCKE レジスタに値を設定します。 設定値は、使用するデバイスの「ユーザーズマニュアル ハードウェア編」を参照してください。
SSI_CH1_AUCKE - **デフォルト値 (0)**	
SSI_CH0_CKDV - **デフォルト値 (0)**	SSI チャネル 0 または 1 の SSICR.CKDV レジスタに値を設定します。 設定値には、使用するデバイスの「ユーザーズマニュアル ハードウェア編」を参照してください。
SSI_CH1_CKDV - **デフォルト値 (0)**	

2.8 API のデータ構造

本節では SSI モジュールの API 関数に用いられるデータ構造を示します。

2.8.1 データ型

API 関数で用いられるパラメータは r_ssi_api_rx_if.h で定義されます。 同ファイルはパブリックなインタフェースファイルであり、使用可能な値が定義されています。

2.9 戻り値

本モジュールの全ての API 関数は下記の 3 種の何れかに定義されます。

- int8_t
- int32_t
- ssi_ret_t

  "ssi_ret_t"はファイル r_ssi_api_rx_if.h で列挙型の typedef として定義されています。

R_SSI_Write()と R_SSI_Read()は “int8_t”、R_SSI_GetVersion() は “int32_t”、その他は全て“ssi_ret_t”型です。R_SSI_Write() は、正の値で送信 FIFO に書き込んだ PCM データのサンプル数を返し、またエラーを示すため負の値を返します。その場合の負の戻り値の定義は ”ssi_ret_t” と同じ意味です。同様に R_SSI_Read() も受信 FIFO から読み出した PCM データのサンプル数とエラーを返します。

```
typedef enum {
    SSI_SET       =  1,    /* A specified error flag of SSISR is 1. */
    SSI_CLR       =  0,    /* A specified error flag of SSISR is 0. */
    SSI_SUCCESS   =  0,    /* Function is finished successfully. */
    SSI_ERR_PARAM = -1,    /* Function is finished unsuccessfully because of
                              incorrect argument. */
    SSI_ERR_CHANNEL = -2, /* Function is finished unsuccessfully because the
                              specified SSI channel is already occupied. */
    SSI_ERR_EXEPT = -3,    /* Function is finished unsuccessfully because of
                              unwanted hardware condition. */
} ssi_ret_t;
```

2.10　SSI モジュールの追加方法

本モジュールは、使用するプロジェクトごとに追加する必要があります。ルネサスでは、Smart Configurator を使用した(1)、(3)の追加方法を推奨しています。ただし、Smart Configurator は、一部の RX デバイスのみサポートしています。サポートされていない RX デバイスについては(2)、(4)の方法を使用してください。

(1) e² studio 上で Smart Configurator を使用して FIT モジュールを追加する場合
e² studio の Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e² studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(2) e² studio 上で FIT Configurator を使用して FIT モジュールを追加する場合
e² studio の FIT Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加することができます。詳細は、アプリケーションノート「RX ファミリ e² studio に組み込む方法 Firmware Integration Technology (R01AN1723)」を参照してください。

(3) CS+上で Smart Configurator を使用して FIT モジュールを追加する場合
CS+上で、スタンドアロン版 Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e² studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(4) CS+上で FIT モジュールを追加する場合
CS+上で、手動でユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」を参照してください。

2.11 for 文、while 文、do while 文について

本モジュールでは、レジスタの反映待ち処理等で for 文、while 文、do while 文（ループ処理）を使用しています。これらループ処理には、「WAIT_LOOP」をキーワードとしたコメントを記述しています。そのため、ループ処理にユーザがフェイルセーフの処理を組み込む場合は、「WAIT_LOOP」で該当の処理を検索できます。

以下に記述例を示します。

```
while 文の例：
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}

for 文の例：
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}

do while 文の例：
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /* WAIT_LOOP */
```

3. API 関数

3.1 R_SSI_Open

指定した SSI チャネルをロックし、r_ssi_api_rx_config.h の設定に基づき初期化します。

Format

```
ssi_ret_t R_SSI_Open ( const ssi_ch_t Channel );
```

Parameters

Channel

ロックする SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESS: *正常終了、指定 SSI チャネルは設定された。*
SSI_ERR_PARAM: *異常終了、パラメータが不正。*
SSI_ERR_CHANNEL: *異常終了、指定 SSI チャネルがロックできなかった。*
SSI_ERR_EXCEPT: *異常終了、指定 SSI チャネルが想定外のハードウェアの状態であった。*

Properties

ファイル r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

SSI を使用する前にチャネル個別に必ず一度、本関数を実行してください。

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルをロックします。
- 指定 SSI チャネルのモジュールストップ状態を解除します。
- 指定 SSI チャネルの I/O レジスタを初期化します。
- ファイル r_ssi_api_rx_config.h を参照し、指定 SSI チャネルの I/O レジスタに値を設定します。

Reentrant

この関数はリエントラントです。

3.2 R_SSI_Close

指定した SSI チャネルのロックを解除します。

Format

```
ssi_ret_t R_SSI_Close ( const ssi_ch_t Channel );
```

Parameters

Channel

ロック解除する SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

```
typedef enum
{
   SSI_CH0 = 0,     /* SSI channel 0 */
   SSI_CH1 = 1,     /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESS: *正常終了、指定 SSI チャネルはロック解除された。*

SSI_ERR_PARAM: *異常終了、パラメータが不正。*

SSI_ERR_CHANNEL: *異常終了、指定 SSI チャネルがロック解除できなかった。或いはロックされていなかった。*

Properties

ファイル r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

SSI の使用を終了する際、本関数を実行してください。

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルがロックされているかを確認し、ロックされていない場合は SSI_ERR_CHANNEL を返します。
- 指定 SSI チャネルをロック解除します。
- 指定 SSI チャネルをモジュールストップ状態に設定します。

Reentrant

この関数はリエントラントです。

3.3 R_SSI_Start

指定した SSI チャネルの送信と受信動作を許可します。

Format

```
ssi_ret_t R_SSI_Start ( const ssi_ch_t Channel );
```

Parameters

Channel

送信と受信動作を許可する SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESS:　　*正常終了、指定 SSI チャネルの PCM データ送信と受信動作は許可された。*
SSI_ERR_PARAM:　　*異常終了、パラメータが不正。*
SSI_ERR_CHANNEL:　　*異常終了、指定 SSI チャネルがロックされていなかった。*

Properties

r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

SSI の PCM データ送信と受信動作を許可する際、本関数を実行してください。

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルがロックされているかを確認し、ロックされていない場合は SSI_ERR_CHANNEL を返します。
- 指定 SSI チャネルが送信に用いられる場合、その SSI チャネルの送信 FIFO をクリアします。
- 指定 SSI チャネルが受信に用いられる場合、その SSI チャネルの受信 FIFO をクリアします。
- 指定 SSI チャネルが送信に用いられる場合、SSIFCR と SSICR レジスタの割り込み許可ビット TIE、TOIRQ 及び TUIRQ をセットします。
- 指定 SSI チャネルが受信に用いられる場合、SSIFCR と SSICR レジスタの割り込み許可ビット RIE、ROIRQ 及び RUIRQ をセットします。
- 指定 SSI チャネルが送信に用いられる場合、SSICR レジスタの TEN ビットをセットし PCM データ送信を許可します。
- 指定 SSI チャネルが受信に用いられる場合、SSICR レジスタの REN ビットをセットし PCM データ受信を許可します。

割り込みで動作する SSI モジュールのアプリケーションソフトウェアを設計する場合、ICU を使いマスク解除してください。

Reentrant

この関数はリエントラントではありません。

3.4 R_SSI_Stop

指定した SSI チャネルの送信と受信動作を禁止します。

Format

```
ssi_ret_t R_SSI_Stop ( const ssi_ch_t Channel );
```

Parameters

Channel

送信と受信動作を禁止する SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

```
typedef enum
{
    SSI_CH0 = 0,     /* SSI channel 0 */
    SSI_CH1 = 1,     /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESS:　正常終了、指定 SSI チャネルの PCM データ送信と受信動作は禁止された。
SSI_ERR_PARAM:　異常終了、パラメータが不正。
SSI_ERR_CHANNEL:　異常終了、指定 SSI チャネルがロックされていなかった。
SSI_ERR_EXCEPT:　異常終了、指定した SSI チャネルが想定外のハードウェアの状態であった。

Properties

r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

SSI の PCM データ送信と受信動作を禁止する際、本関数を実行してください。

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルがロックされているかを確認し、ロックされていない場合は SSI_ERR_CHANNEL を返します。
- 指定 SSI チャネルが送信に用いられる場合、SSIFCR と SSICR レジスタの割り込み許可ビット TIE、TOIRQ 及び TUIRQ をクリアします。
- 指定 SSI チャネルが受信に用いられる場合、SSIFCR と SSICR レジスタの割り込み許可ビット RIE、ROIRQ 及び RUIRQ をクリアします。
- 指定 SSI チャネルが送信に用いられる場合、SSICR レジスタの TEN ビットをクリアし PCM データ送信を禁止します。
- 指定 SSI チャネルが受信に用いられる場合、SSICR レジスタの REN ビットをクリアし PCM データ受信を禁止します。

割り込みで動作する SSI モジュールのアプリケーションソフトウェアを設計する場合、ICU を使いマスクしてください。

Reentrant

この関数はリエントラントではありません。

3.5 R_SSI_Write

送信動作のため、指定した SSI チャネルに PCM データを書き込みます。ユーザは送信する PCM データを格納するために、必要なサイズのメモリを用意します。

Format

```
int8_t R_SSI_Write ( const ssi_ch_t Channel, const void * pBuf, const uint8_t Samples );
```

Parameters

Channel

PCM データを書き込む SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

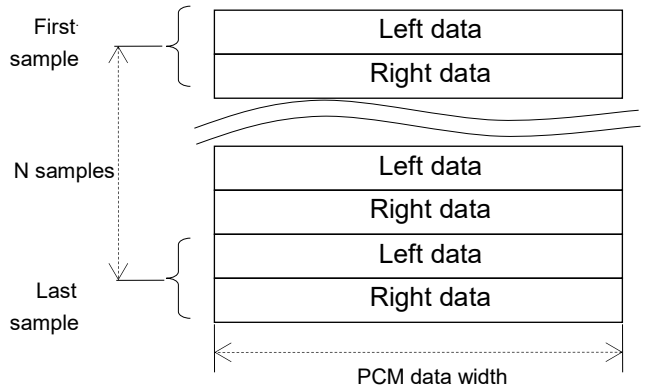
```
typedef enum
{
    SSI_CH0 = 0,     /* SSI channel 0 */
    SSI_CH1 = 1,     /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

pBuf

送信 FIFO データレジスタに書き込む PCM データが格納された PCM バッファメモリの先頭アドレスを指定します。

PCM buffer memory

- The storing order of PCM data is shown as following figure.
- In case 8 or 16 bit width PCM data, width of buffer memory is the same as PCM data width.
- In case 18, 20, 22 or 24 bit width PCM data, width of buffer memory must be 32bit (LSB justified).
- Total bytes of the buffer memory must be multiple of 8.



Samples

送信 FIFO データレジスタに書き込む PCM データの要求サンプル数を指定します。

Return Values

書き込みサンプル数：　指定 SSI チャネルの送信 FIFO データレジスタに書き込まれた PCM データのサンプル数を示します。
SSI_ERR_PARAM:　異常終了、パラメータが不正。
SSI_ERR_CHANNEL:　異常終了、指定 SSI チャネルがロックされていなかった。または、指定 SSI チャネルが受信に設定されていた。
SSI_ERR_EXCEPT:　異常終了、指定した SSI チャネルが想定外のハードウェアの状態であった。

Properties

ファイル r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

SSI の送信 FIFO データレジスタへの PCM データの書き込みの際、本関数を実行してください。

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルがロックされているかを確認し、ロックされていない場合は SSI_ERR_CHANNEL を返します。また指定 SSI チャネルが受信に設定されていた場合にも SSI_ERR_CHANNEL を返します。
- ２つのパラメータ *pBuf* と *Samples* に応じ PCM データを送信 FIFO データレジスタに書き込みます。送信 FIFO データレジスタがフルになると、この関数は要求したサンプル数の書き込みを完了する前に書き込みをやめ、書き込みに成功したサンプル数を返します。

なお、ユーザは送信動作中 R_SSI_Write ()を使って指定 SSI チャネルに繰り返し PCM データを書き込まなければなりません。

Reentrant

この関数はリエントラントではありません。

3.6 R_SSI_Read

受信動作のため、指定した SSI チャネルから PCM データを読み出し、ユーザが指定したメモリに格納します。ユーザは受信した PCM データを格納するための適切なサイズのメモリを用意します。

Format

```
int8_t R_SSI_Read ( const ssi_ch_t Channel, const void * pBuf, const uint8_t Samples );
```

Parameters

Channel

PCM データを読み出す SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

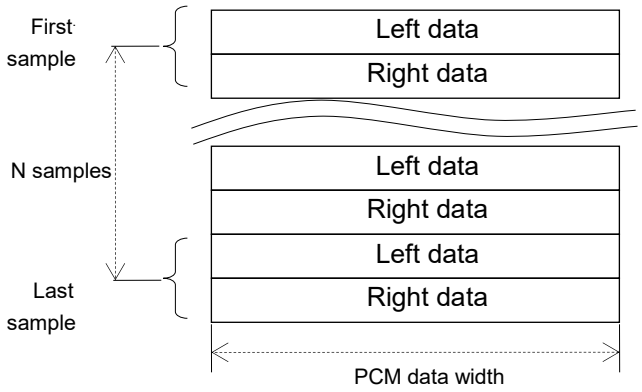
```
typedef enum
{
    SSI_CH0 = 0,     /* SSI channel 0 */
    SSI_CH1 = 1,     /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

pBuf

受信 FIFO データレジスタから読み出した PCM データを格納する PCM バッファメモリの先頭アドレスを指定します。

PCM buffer memory

- The storing order of PCM data is shown as following figure.
- In case 8 or 16 bit width PCM data, width of buffer memory is the same as PCM data width.
- In case 18, 20, 22 or 24 bit width PCM data, width of buffer memory must be 32bit (LSB justified).
- Total bytes of the buffer memory must be multiple of 8.



Samples

受信 FIFO データレジスタから読み出す PCM データの要求サンプル数を指定します。

Return Values

読み出しサンプル数：　指定 SSI チャネルの受信 FIFO データレジスタから読み出された PCM データのサンプル数を示します。
SSI_ERR_PARAM:　異常終了、パラメータが不正。
SSI_ERR_CHANNEL:　異常終了、指定 SSI チャネルがロックされていなかった。
または、指定 SSI チャネルが送信に設定されていた。
SSI_ERR_EXCEPT:　異常終了、指定した SSI チャネルが想定外のハードウェアの状態であった。

Properties

ファイル r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

SSI の受信 FIFO データレジスタからの PCM データの読み出しの際、本関数を実行してください。

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルがロックされているかを確認し、ロックされていない場合は SSI_ERR_CHANNEL を返します。また指定 SSI チャネルが送信に設定されていた場合にも SSI_ERR_CHANNEL を返します。
- ２つのパラメータ *pBuf* と *Samples* に応じ PCM データを受信 FIFO データレジスタから読み出します。 しかし受信 FIFO データレジスタがエンプティになると、この関数は要求したサンプル数の読み出しを完了する前に読み出しを停止、読み出しに成功したサンプル数を返します。

なお、ユーザは受信動作中 R_SSI_Read () を使って指定 SSI チャネルから繰り返し PCM データを読み出さなければなりません。

Reentrant

この関数はリエントラントではありません。

3.7 R_SSI_Mute

送信動作中の SSI チャネルをミュートに設定、または解除します。

ミュートの間、指定した SSI チャネルから出力される PCM データは 0 になります。

Format

```
ssi_ret_t R_SSI_Mute ( const ssi_ch_t Channel, const ssi_mute_t OnOff );
```

Parameters

Channel

ミュートに設定、または解除する SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

```
typedef enum
{
    SSI_CH0 = 0,     /* SSI channel 0 */
    SSI_CH1 = 1,     /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

OnOff

ミュート設定または解除を選択します。以下に示す列挙体 ssi_mute_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

```
typedef enum
{
    SSI_MUTE_ON  = 0,
    SSI_MUTE_OFF = 1,
} ssi_mute_t;
```

Return Values

SSI_SUCCESS:	*正常終了、指定 SSI チャネルはミュートに設定、または解除された。*
SSI_ERR_PARAM:	*異常終了、パラメータが不正。*
SSI_ERR_CHANNEL:	*異常終了、指定 SSI チャネルがロックされていなかった。 または、指定 SSI チャネルが送信以外に設定されていた。*
SSI_ERR_EXCEPT:	*異常終了、指定した SSI チャネルが想定外のハードウェアの状態であった。*

Properties

ファイル r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

送信動作中の SSI チャネルをミュートに設定、または解除する場合、本関数を実行してください

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルがロックされているかを確認し、ロックされていない場合、または、指定 SSI チャネルが送信以外に設定されていた場合に SSI_ERR_CHANNEL を返します。
- SSI_MUTE_ON に設定されると、いくつかの割り込みを禁止し、WS コンティニュモードで PCM データとして 0 を送信するように SSI を設定します。レジスタは以下のように設定されます。
  SSICR の TUIEN と SSIFCR の TIE をクリア、SSIFTDR に 0 を書き込み、SSITDMR の CONT をセット、SSICR の TEN と SSISR の TUIRQ をクリア。
- SSI_MUTE_OFF に設定されると、WS コンティニュモードを解除、いくつかの割り込みを許可し、送信 FIFO データレジスタに書き込んだ PCM データを送信できるように SSI を設定します。レジスタは以下のように設定されます。
  SSIFTDR に 0 を書き込み、SSICR の TEN をセット、SSITDMR の CONT をクリア、SSIFSR の TDE と SSISR の TUIRQ をクリア、SSIFCR の TIE と SSICR の TUIEN をセット。

Reentrant

この関数はリエントラントではありません。

3.8 R_SSI_GetVersion

モジュールのバージョンを返します。

Format

```
uint32_t R_SSI_GetVersion ( void );
```

Parameters

None

Return Values

一つの 32 ビットの数値に格納されたメジャーとマイナーからなるバージョン番号。

Properties

ファイル r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

この関数は本モジュールのバージョンを返します。上位２バイトはメジャーバージョン番号、下位２バイトはマイナーバージョン番号を示します。

Reentrant

この関数はリエントラントです。

Example

```
/* Retrieve the version number and convert it to a string. */

uint32_t   version, version_high, version_low;
char      version_str[9];

version = R_SSI_GetVersion();
version_high = (version >> 16)&0xf;
version_low  =  version & 0xff;

sprintf(version_str, "SSI v%1.1hu.%2.2hu", version_high, version_low);
```

3.9 R_SSI_GetFlagTxUnderFlow

送信アンダフローの状態を返します。状態は指定した SSI チャネルの SSISR の TUIRQ フラグに相当する値です。

Format

```
ssi_ret_t R_SSI_GetFlagTxUnderFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

TUIRQ フラグの状態を取得する SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

```
typedef enum
{
   SSI_CH0 = 0,     /* SSI channel 0 */
   SSI_CH1 = 1,     /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_CLR: *指定 SSI チャネルの TUIRQ フラグが 0 である。*
SSI_SET: *指定 SSI チャネルの TUIRQ フラグが 1 である。*
SSI_ERR_CHANNEL: *異常終了、指定 SSI チャネルがロックされていなかった。*
SSI_ERR_PARAM: *異常終了、パラメータが不正。*

Properties

ファイル r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

SSI の SSISR レジスタの TUIRQ フラグの状態を確認する際、本関数を実行してください。

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルがロックされているかを確認し、ロックされていない場合は SSI_ERR_CHANNEL を返します。
- TUIRQ フラグの状態を読み、状態に応じ SSI_SET 或いは SSI_CLR を返します。

Reentrant

この関数はリエントラントです。

3.10 R_SSI_GetFlagTxOverFlow

送信オーバフローの状態を返します。状態は指定した SSI チャネルの SSISR の TOIRQ フラグに相当する値です。

Format

```
ssi_ret_t R_SSI_GetFlagTxOverFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

TOIRQ フラグの状態を取得する SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_CLR: *指定 SSI チャネルの TOIRQ フラグが 0 である。*
SSI_SET: *指定 SSI チャネルの TOIRQ フラグが 1 である。*
SSI_ERR_CHANNEL: *異常終了、指定 SSI チャネルがロックされていなかった。*
SSI_ERR_PARAM: *異常終了、パラメータが不正。*

Properties

ファイル r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

SSI の SSISR レジスタの TOIRQ フラグの状態を確認する際、本関数を実行してください。

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルがロックされているかを確認し、ロックされていない場合は SSI_ERR_CHANNEL を返します。
- TOIRQ フラグの状態を読み、状態に応じ SSI_SET 或いは SSI_CLR を返します。

Reentrant

この関数はリエントラントです。

3.11 R_SSI_GetFlagRxUnderFlow

受信アンダフローの状態を返します。状態は指定した SSI チャネルの SSISR の RUIRQ フラグに相当する値です。

Format

```
ssi_ret_t R_SSI_GetFlagRxUnderFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

RUIRQ フラグの状態を取得する SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

```
typedef enum
{
   SSI_CH0 = 0,     /* SSI channel 0 */
   SSI_CH1 = 1,     /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_CLR: *指定 SSI チャネルの RUIRQ フラグが 0 である。*
SSI_SET: *指定 SSI チャネルの RUIRQ フラグが 1 である。*
SSI_ERR_CHANNEL: *異常終了、指定 SSI チャネルがロックされていなかった。*
SSI_ERR_PARAM: *異常終了、パラメータが不正。*

Properties

ファイル r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

SSI の SSISR レジスタの RUIRQ フラグの状態を確認する際、本関数を実行してください。

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルがロックされているか確認し、ロックされていない場合は SSI_ERR_CHANNEL を返します。
- RUIRQ フラグの状態を読み、状態に応じ SSI_SET 或いは SSI_CLR を返します。

Reentrant

この関数はリエントラントです。

3.12 R_SSI_GetFlagRxOverFlow

受信オーバフローの状態を返します。状態は指定した SSI チャネルの SSISR の ROIRQ フラグに相当する値です。

Format

```
ssi_ret_t R_SSI_GetFlagRxOverFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

ROIRQ フラグの状態を取得する SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

```
typedef enum
{
   SSI_CH0 = 0,     /* SSI channel 0 */
   SSI_CH1 = 1,     /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_CLR: *指定 SSI チャネルの ROIRQ フラグが 0 である。*
SSI_SET: *指定 SSI チャネルの ROIRQ フラグが 1 である。*
SSI_ERR_CHANNEL: *異常終了、指定 SSI チャネルがロックされていなかった。*
SSI_ERR_PARAM: *異常終了、パラメータが不正。*

Properties

ファイル r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

SSI の SSISR レジスタの ROIRQ フラグの状態を確認する際、本関数を実行してください。

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルがロックされているか確認し、ロックされていない場合は SSI_ERR_CHANNEL を返します。
- ROIRQ フラグの状態を読み、状態に応じ SSI_SET 或いは SSI_CLR を返します。

Reentrant

この関数はリエントラントです。

3.13 R_SSI_ClearFlagTxUnderFlow

指定した SSI チャネルの SSISR の TUIRQ フラグを 0 にクリアします。

Format

```
ssi_ret_t R_SSI_ClearFlagTxUnderFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

TUIRQ フラグをクリアする SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

```
typedef enum
{
   SSI_CH0 = 0,     /* SSI channel 0 */
   SSI_CH1 = 1,     /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESS: *正常終了、指定 SSI チャネルの TUIRQ フラグはクリアされた。*
SSI_ERR_CHANNEL: *異常終了、指定 SSI チャネルがロックされていなかった。*
SSI_ERR_PARAM: *異常終了、パラメータが不正。*

Properties

ファイル r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

TUIRQ フラグが 1 の時、フラグをクリアするため本関数を実行してください。

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルがロックされているか確認し、ロックされていない場合は SSI_ERR_CHANNEL を返します。
- 指定 SSI チャネルの SSISR を読み TUIRQ フラグを 0 にクリア、SSI_SUCCESS を返します。

Reentrant

この関数はリエントラントです。

3.14 R_SSI_ClearFlagTxOverFlow

指定した SSI チャネルの SSISR の TOIRQ フラグを 0 にクリアします。

Format

```
ssi_ret_t R_SSI_ClearFlagTxOverFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

TOIRQ フラグをクリアする SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

```
typedef enum
{
   SSI_CH0 = 0,     /* SSI channel 0 */
   SSI_CH1 = 1,     /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESS: *正常終了、指定 SSI チャネルの TOIRQ フラグはクリアされた。*
SSI_ERR_CHANNEL: *異常終了、指定 SSI チャネルがロックされていなかった。*
SSI_ERR_PARAM: *異常終了、パラメータが不正。*

Properties

ファイル r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

TOIRQ フラグが 1 の時、フラグをクリアするため本関数を実行してください。

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルがロックされているか確認し、ロックされていない場合は SSI_ERR_CHANNEL を返します。
- 指定 SSI チャネルの SSISR を読み TOIRQ フラグを 0 にクリア、SSI_SUCCESS を返します。

Reentrant

この関数はリエントラントです。

3.15 R_SSI_ClearFlagRxUnderFlow

指定した SSI チャネルの SSISR の RUIRQ フラグを 0 にクリアします。

Format

```
ssi_ret_t R_SSI_ClearFlagRxUnderFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

RUIRQ フラグをクリアする SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

```
typedef enum
{
   SSI_CH0 = 0,     /* SSI channel 0 */
   SSI_CH1 = 1,     /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESS: *正常終了、指定 SSI チャネルの RUIRQ フラグはクリアされた。*
SSI_ERR_CHANNEL: *異常終了、指定 SSI チャネルがロックされていなかった。*
SSI_ERR_PARAM: *異常終了、パラメータが不正。*

Properties

ファイル r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

RUIRQ フラグが 1 の時、フラグをクリアするため本関数を実行してください。

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルがロックされているか確認し、ロックされていない場合は SSI_ERR_CHANNEL を返します。
- 指定 SSI チャネルの SSISR を読み RUIRQ フラグを 0 にクリア、SSI_SUCCESS を返します。

Reentrant

この関数はリエントラントです。

3.16 R_SSI_ClearFlagRxOverFlow

指定した SSI チャネルの SSISR の ROIRQ フラグを 0 にクリアします。

Format

```
ssi_ret_t R_SSI_ClearFlagRxUnderFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

ROIRQ フラグをクリアする SSI チャネルを指定します。以下に示す列挙体 ssi_ch_t の定義から一つの列挙体メンバを選択してください。列挙体はファイル r_ssi_api_rx_if.h に記述されています。

```
typedef enum
{
   SSI_CH0 = 0,     /* SSI channel 0 */
   SSI_CH1 = 1,     /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESS:　　正常終了、指定 SSI チャネルの ROIRQ フラグはクリアされた。
SSI_ERR_CHANNEL:　異常終了、指定 SSI チャネルがロックされていなかった。
SSI_ERR_PARAM:　　異常終了、パラメータが不正。

Properties

ファイル r_ssi_api_rx_if.h にプロトタイプ宣言されています。

Description

ROIRQ フラグが 1 の時、フラグをクリアするため本関数を実行してください。

指定した SSI チャネルに以下を実行します。

- パラメータの正当性を確認し、不正時には SSI_ERR_PARAM を返します。
- 指定 SSI チャネルがロックされているか確認し、ロックされていない場合は SSI_ERR_CHANNEL を返します。
- 指定 SSI チャネルの SSISR を読み ROIRQ フラグを 0 にクリア、SSI_SUCCESS を返します。

Reentrant

この関数はリエントラントです。

4. 付録

4.1 動作確認環境

本 FIT モジュールの動作確認環境を以下に示します。

表 4.1 動作確認環境 (Rev.1.23)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio V7.3.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.01.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 4.8.4.201801 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std = gnu99
	IAR C/C++ Compiler for Renesas RX version 4.10.1 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev1.23
使用ボード	Renesas Starter Kit for RX231（型名：R0K505231SxxxBE）

表 4.2 動作確認環境 (Rev.1.24)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio V7.5.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.01.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev1.24
使用ボード	Renesas Solution Starter Kit for RX23W

4.2 トラブルシューティング

(1) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「Could not open source file "platform.h"」エラーが発生します。

A：FIT モジュールがプロジェクトに正しく追加されていない可能性があります。プロジェクトへの追加方法をご確認ください。

- CS+を使用している場合
  アプリケーションノート RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」
- e^2 studio を使用している場合
  アプリケーションノート RX ファミリ e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)」

また、本 FIT モジュールを使用する場合、ボードサポートパッケージ FIT モジュール(BSP モジュール)もプロジェクトに追加する必要があります。BSP モジュールの追加方法は、アプリケーションノート「ボードサポートパッケージモジュール(R01AN1685)」を参照してください。

(2) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「#error "ERROR !!! The value set to SSI_CH0_IO_MODE is invalid."」エラーが発生します。

A：“r_ssi_api_rx_config.h”ファイルの設定値が間違っている可能性があります。
“r_ssi_api_rx_config.h”ファイルを確認して正しい値を設定してください。詳細は「2.7 コンフィグレーションの概要」を参照してください。

5. 参考ドキュメント

ユーザーズマニュアル：ハードウェア
（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデート／テクニカルニュース
（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：開発環境
RX ファミリ CC-RX コンパイラ ユーザーズマニュアル（R20UT3248）
（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデートの対応について

本モジュールは以下のテクニカルアップデートの内容を反映しています。

- TN-RX*-A133B/J

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2014/4/1	-	初版発行
1.10	2014/12/5	3, 21~28	・ 以下の関数を追加。 R_SSI_GetFlagTxUnderFlow (); R_SSI_GetFlagTxOverFlow (); R_SSI_GetFlagRxUnderFlow (); R_SSI_GetFlagRxOverFlow (); R_SSI_ClearFlagTxUnderFlow (); R_SSI_ClearFlagTxOverFlow (); R_SSI_ClearFlagRxUnderFlow (); R_SSI_ClearFlagRxOverFlow ();
		10	・ 列挙体 ”ssi_ret_t” に以下のメンバを追加。 SSI_FLAG_SET = 1, SSI_FLAG_CLR = 0,
1.11	2014/12/12	-	・ RX71M グループのサポートを追加。
		4	・ “2.3 ソフトウェアの要求” の変更。 要求される r_bsp のバージョンを v2.60 から v2.80 以上に変更。
1.20	2015/4/28	-	・ RX113、RX231 と RX230 グループのサポートを追加。
		1	・ 使用可能な SSI チャネル数に関するコメントを追加。
		4	・ メモリ使用量を変更 ・ 要求される r_bsp のバージョンを v2.80 から v2.90 以上に変更。
		12~19, 20~28	・ typedef enum ssi_ch_t を示す領域のコメントを変更。
1.21	2017/4/7	-	・ 本書全体にわたり誤記と表現を修正。
		29	・ 4 章を追加。
1.22	2019/2/1	-	機能関連 Smart Configurator での GUI によるコンフィグオプション設定機能に対応 ■内容 GUI によるコンフィグオプション設定機能に対応するため、設定ファイルを追加。
1.23	2019/5/20	-	・ GCC、IAR コンパイラサポートを追加。
		3, 23	・ R_SSI_Mute について未記載のため追記。
		33, 34	・ “4. 付録”を追加。
1.24	2019/6/20	-	・ RX23W グループのサポートを追加。

製品ご使用上の注意事項

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

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

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

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1　2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

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

www.renesas.com/contact/

商標について

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