RX ファミリ

SCI FIFO 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 FIFO 内蔵シリアルコミュニケーションインタフェース（SCIFA）モジュールについて説明します。本モジュールは、SCIFA の全チャネルに対応し、調歩同期式モード、およびクロック同期式モードを使用できます。チャネル、およびモードは個別に有効／無効を設定できます。

以降、本モジュールを SCI FIFO FIT モジュールと称します。

対象デバイス

本モジュールは以下のデバイスで使用できます。

- RX64M グループ
- RX71M グループ

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。

関連ドキュメント

- Firmware Integration Technology ユーザーズマニュアル (R01AN1833)
- RX ファミリ ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)
- RX ファミリ e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)
- RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)
- Renesas e^2 studio スマート・コンフィグレータ ユーザーガイド(R20AN0451)

目次

1. 概要

1.1 SCI FIFO FIT モジュールとは

本モジュールは API として、プロジェクトに組み込んで使用します。本モジュールの組み込み方については、「2.13 FIT モジュールの追加方法」を参照してください。

1.2 SCI FIFO FIT モジュールの概要

SCI FIFO FIT モジュールは RX64M および RX71M の SCIFA をサポートします。RX64M/RX71M グループのユーザーズマニュアル ハードウェア編で、SCIFA の章をご覧いただき、SCIFA の機能についてご確認ください。本モジュールでは、調歩同期式モード、およびクロック同期式モード（マスタのみ）をサポートしています。また、調歩同期式モードでは、以下の機能をサポートしています。

- ノイズ除去
- MSB ファーストのビットオーダ
- CTS または RTS 端子を用いたハードウェアフロー制御

本モジュールでサポートされない機能:

- DRIF 割り込み（メッセージ長（バイト）がしきい値未満の場合のみ機能します。）

本モジュールは SCIFA の全チャネルをサポートします。使用チャネルは r_scif_rx_config.h で定義できます。この定義を設定することで、使用しないチャネルをコンパイル時の定義で省くことができ、RAM および ROM の使用サイズやコードサイズを抑えることができます。

ユーザによって R_SCIF_Open()関数が呼び出されると、本モジュールはチャネルを初期化します。R_SCIF_Open()関数で、SCIF 周辺機能を起動し、指定されたモードに応じて初期設定を行います。R_SCIF_Open()関数では、チャネルを識別するためのハンドルを返します。このハンドルは、対象チャネルに関連するレジスタ、バッファ、その他の必要な情報へのポインタを保持する内部構造体を参照しており、他の API 関数に引数として渡すことで、利用チャネルに対する処理を行うことができます。

本モジュールは割り込みを使用し、ノンブロッキングで動作します。調歩同期式モードでは、データは、オーバフローが発生するか、R_SCIF_Receive()関数が呼び出されるまで（いずれか早い方）、受信 FIFO に格納されます。本モジュールでは TXIF、RXIF、および GROUPAL0 の TEIF、ERIF、および BRIF 割り込みを使用します。

TXIF 割り込みは、送信 FIFO に格納されたデータがしきい値以下になった場合に発生します。TXIF 割り込み中、メッセージのデータがなくなるか、送信 FIFO がフルになるまでは（いずれか早い方）、送信メッセージのデータが FIFO に読み込まれます。TEIF 割り込みは、FIFO からの最終バイトの最終ビットが TSR レジスタからシフトされた後にのみ発生します。R_SCIF_Open()関数でコールバック関数が指定されていれば、指定された関数が呼び出され、SCIF_EVT_TX_DONE（調歩同期）または SCIF_EVT_XCV_DONE（クロック同期）イベントが渡されます。R_SCIF_Send()および R_SCIF_SendReceive()関数は、1 度に未処理の送信要求を 2 つ持つことができ、連続的にデータを送信できます。DONE イベントは、未処理の要求がすべて処理されるまでは発生しません。各メッセージの完了を把握する必要がある場合、未処理の要求は 1 度に 1 つとしてください。

RXIF 割り込みは、受信 FIFO にしきい値分のデータが溜まるごとに発生します。この割り込み中、要求されたバイト数が読み出されるか、FIFO にデータがなくなるまで、FIFO からデータを読み出し、メッセージバッファに配置します。要求された全バイト数が読み出されたとき、コールバック関数が指定されていれば、指定された関数が呼び出され、SCIF_EVT_RX_DONE（調歩同期）、または SCIF_EVT_XCV_DONE（クロック同期）イベントが渡されます。R_SCIF_Receive()および R_SCIF_SendReceive()関数は 1 度に未処理の受信要求を 2 つ持つことができ、連続的にデータを受信できます。DONE イベントは、未処理の要求がすべて処理されるまでは発生しません。各メッセージの完了を把握する必要がある場合、未処理の要求は 1 度に 1 つとしてください。

調歩同期式モードで、受信時にフレーミングまたはパリティエラーが検出された場合、ERIF 割り込みが発生します。また、Break が受信されるか、受信 FIFO のオーバフローが発生した場合、BRIF 割り込みが発生します。コールバック関数が指定されていれば、割り込みで発生したエラーを判定し、アプリケーションにイベントを通知します。コールバック関数の有無に関わらず、該当する FSR または LSR エラーフラグに“0”を書くことで、エラーコンディションがクリアされます。

1.3　API の概要

表 1.1 に本モジュールに含まれる API 関数を示します。

表 1.1　API 関数一覧

関数	説明
R_SCIF_Open	SCIF チャネルを有効にし、関連するレジスタを初期化します。また、割り込みを許可し、他の API 関数に渡すチャネルのハンドルを設定します。任意にコールバック関数を設定することができます。受信エラー、あるいはその他の割り込みイベントが発生すると、コールバック関数を呼び出し、ユーザに通知します。
R_SCIF_Close	SCIF チャネルを無効にし、関連する割り込みを禁止にします。
R_SCIF_Send	送信 FIFO に送信するメッセージを設定します。1 度に未処理の送信要求を 2 つまで持てます。送信アイドル状態であれば、すぐに送信を開始します。
R_SCIF_Receive	受信 FIFO から受信するメッセージを取得します。1 度に未処理の受信要求を 2 つまで持てます。クロック同期式モードでは、送受信アイドル状態であれば通信を開始します。
R_SCIF_SendReceive	クロック同期式モードのみで使用できます。データの送信と受信を同時に行います。1 度に未処理の要求を 2 つ持つことができます（Send()、Receive()、SendReceive()のいずれか 2 つ）。
R_SCIF_Control	SCIF チャネルに関する特定のハードウェアまたはソフトウェア動作を制御します。
R_SCIF_GetVersion	本モジュールのバージョン番号を返します。

2. API 情報

本アプリケーションノートのサンプルコードは、下記の条件で動作を確認しています。

2.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- SCIFA

2.2 ハードウェアリソースの要求

ここでは、本モジュールが要求するハードウェアの周辺機能について説明します。特に記載がない場合、ここで説明するリソースは本モジュールが使用できるように、ユーザは使用しないでください。

2.2.1 SCIFA

本モジュールは SCIFA 周辺機能を使用します。r_scif_rx_config.h で、チャネルを個別に有効／無効にすることができます。

2.2.2 GPIO

本モジュールでは、各チャネルに対応するポート端子を使用します。これらの端子は汎用入出力端子としては使用できません。

2.3 ソフトウェアの要求

このドライバは以下の FIT モジュールに依存しています。

- ボードサポートパッケージ（r_bsp）

2.4 サポートされているツールチェーン

本 FIT モジュールは「6.1 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　使用する割り込みベクタ

調歩同期式モードの場合、R_SCIF_Open 関数を実行すると、TXIFn 割り込み、RXIFn 割り込み、TEIFn 割り込み、ERIFn 割り込み、BRIFn 割り込みが有効になります。R_SCIF_Send 関数を実行すると、DRIFn 割り込みも有効になります。

クロック同期式モードの場合、R_SCIF_Open 関数を実行すると、TXIFn 割り込み、RXIFn 割り込み、TEIFn 割り込みが有効になります。R_SCIF_Send 関数を実行すると、DRIFn 割り込みも有効になります。

表 2.1 に本 FIT モジュールが使用する割り込みベクタを示します。

表 2.1　使用する割り込みベクター覧

デバイス	割り込みベクタ
RX64M RX71M	RXIF8 割り込み[チャネル 8]（ベクタ番号: 100） TXIF8 割り込み[チャネル 8]（ベクタ番号: 101） RXIF9 割り込み[チャネル 9]（ベクタ番号: 102） TXIF9 割り込み[チャネル 9]（ベクタ番号: 103） RXIF10 割り込み[チャネル 10]（ベクタ番号: 104） TXIF10 割り込み[チャネル 10]（ベクタ番号: 105） RXIF11 割り込み[チャネル 11]（ベクタ番号: 114） TXIF11 割り込み[チャネル 11]（ベクタ番号: 115） GROUPAL0 割り込み（ベクタ番号: 112） • TEIF8 割り込み[チャネル 8]（グループ割り込み要因番号: 0） • ERIF8 割り込み[チャネル 8]（グループ割り込み要因番号: 1） • BRIF8 割り込み[チャネル 8]（グループ割り込み要因番号: 2） • DRIF8 割り込み[チャネル 8]（グループ割り込み要因番号: 3） • TEIF9 割り込み[チャネル 9]（グループ割り込み要因番号: 4） • ERIF9 割り込み[チャネル 9]（グループ割り込み要因番号: 5） • BRIF9 割り込み[チャネル 9]（グループ割り込み要因番号: 6） • DRIF9 割り込み[チャネル 9]（グループ割り込み要因番号: 7） • TEIF10 割り込み[チャネル 10]（グループ割り込み要因番号: 8） • ERIF10 割り込み[チャネル 10]（グループ割り込み要因番号: 9） • BRIF10 割り込み[チャネル 10]（グループ割り込み要因番号: 10） • DRIF10 割り込み[チャネル 10]（グループ割り込み要因番号: 11） • TEIF11 割り込み[チャネル 11]（グループ割り込み要因番号: 12） • ERIF11 割り込み[チャネル 11]（グループ割り込み要因番号: 13） • BRIF11 割り込み[チャネル 11]（グループ割り込み要因番号: 14） • DRIF11 割り込み[チャネル 11]（グループ割り込み要因番号: 15）

2.6　ヘッダファイル

すべての API 呼び出しとそれをサポートするインタフェース定義は r_scif_rx_if.h に記載しています。

2.7　整数型

このドライバは ANSI C99 を使用しています。これらの型は stdint.h で定義されています。

2.8　コンパイル時の設定

本モジュールのコンフィギュレーションオプションの設定は、r_scif_rx_config.h で行います。
オプション名および設定値に関する説明を、下表に示します。

Configuration options in r_scif_rx_config.h	
定義	**説明**
#define SCIF_CFG_PARAM_CHECKING_ENABLE ※デフォルト値は“1”	• 1：ビルド時にパラメータチェック処理をコードに含めます。 • 0：ビルド時にパラメータチェック処理をコードから省略します。 • BSP_CFG_PARAM_CHECKING_ENABLE: BSP の設定を使用します。
#define SCIF_CFG_ASYNC_INCLUDED ※デフォルト値は“1” #define SCIF_CFG_SYNC_INCLUDED ※デフォルト値は“0”	モードに特定のコードを含むかどうかを定義します。 “1”を設定すると、対応する処理をコードに含めます。使用しないモードに対しては、“0”を設定してください。全体のコードサイズを小さくできます。
#define SCIF_CFG_CH8_INCLUDED #define SCIF_CFG_CH9_INCLUDED #define SCIF_CFG_CH10_INCLUDED #define SCIF_CFG_CH11_INCLUDED ※各デフォルト値は以下のとおり: CH8、CH10、CH11:“0”、CH9: “1”	チャネルごとに送受信ポインタ、カウンタ、割り込み、その他のプログラム、RAM などのリソースを持ちます。本定義を“1”に設定すると、そのチャネルに関連したリソースが割り当てられます。
#define SCIF_CFG_CH8_TX_FIFO_THRESHOLD #define SCIF_CFG_CH9_TX_FIFO_THRESHOLD #define SCIF_CFG_CH10_TX_FIFO_THRESHOLD #define SCIF_CFG_CH11_TX_FIFO_THRESHOLD ※各デフォルト値は“8”	送信 FIFO は 16 段です。FIFO のデータがしきい値以下になったとき、TXIF 割り込みが発生し、新たなデータを読み込みます。本定義に設定可能な値は 0～15 です。送信される全メッセージがしきい値の倍数であること、また、高速のビットレートで送信中に FIFO にデータを読み込む際、データ間に隙間ができない程度にしきい値を設定することが理想的です。
#define SCIF_CFG_CH8_RX_FIFO_THRESHOLD #define SCIF_CFG_CH9_RX_FIFO_THRESHOLD #define SCIF_CFG_CH10_RX_FIFO_THRESHOLD #define SCIF_CFG_CH11_RX_FIFO_THRESHOLD ※各デフォルト値は“8”	受信 FIFO は 16 段です。FIFO にしきい値分のデータが溜まると、RXIF 割り込みが発生し、新たなデータを読み出します。本定義に設定可能な値は 1～16 です。受信される全メッセージがしきい値の倍数であること、また、しきい値は、高速のビットレートで受信中に、FIFO からのデータ読み出し時間の不足によってオーバフローが発生しない程度の小さい値であることが理想的です。 クロック同期式モードでは、本定義の設定値を、同じチャネルの対応する TX_FIFO_THRESHOLD の値と同じにしてください。

2.9 コードサイズ

本モジュールの ROM サイズ、RAM サイズを下表に示します。

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「2.8 コンパイル時の設定」のコンフィギュ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_scif_rx rev1.22

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)

コンフィグレーションオプション: デフォルト設定

ROM および RAM のコードサイズ (1/2)

	パラメータチェックあり	パラメータチェックなし
調歩同期のみ 1 チャネル	ROM: 3426 バイト	ROM: 3141 バイト
	RAM: 57 バイト	RAM: 57 バイト
調歩同期のみ 2 チャネル	ROM: 3605 バイト	ROM: 3320 バイト
	RAM: 105 バイト	RAM: 105 バイト
調歩同期のみ 3 チャネル	ROM: 3787 バイト	ROM: 3502 バイト
	RAM: 153 バイト	RAM: 153 バイト
調歩同期のみ 4 チャネル	ROM: 3969 バイト	ROM: 1684 バイト
	RAM: 201 バイト	RAM: 201 バイト
クロック同期のみ 1 チャネル	ROM: 2420 バイト	ROM: 2203 バイト
	RAM: 52 バイト	RAM: 52 バイト
クロック同期のみ 2 チャネル	ROM: 2561 バイト	ROM: 2344 バイト
	RAM: 100 バイト	RAM: 100 バイト
クロック同期のみ 3 チャネル	ROM: 2703 バイト	ROM: 2486 バイト
	RAM: 148 バイト	RAM: 148 バイト
クロック同期のみ 4 チャネル	ROM: 2845 バイト	ROM: 2628 バイト
	RAM: 196 バイト	RAM: 196 バイト

ROM および RAM のコードサイズ (2/2)		
	パラメータチェックあり	パラメータチェックなし
調歩同期 & クロック同期 2 チャネル	ROM: 3941 バイト	ROM: 3606 バイト
	RAM: 105 バイト	RAM: 105 バイト
調歩同期 & クロック同期 3 チャネル	ROM: 4123 バイト	ROM: 3788 バイト
	RAM: 153 バイト	RAM: 153 バイト
調歩同期 & クロック同期 4 チャネル	ROM: 4305 バイト	ROM: 3970 バイト
	RAM: 201 バイト	RAM: 201 バイト

2.10 引数

本モジュールの API 関数で使用される構造体は、“r_scif_rx_if.h”に記載されます。詳細は「3 API 関数」をご覧ください。

2.11 戻り値

以下に本モジュールの API 関数で使用できる戻り値を示します。戻り値の列挙型は、API 関数の宣言と共に r_scif_rx_if.h に記載されています。

```
typedef enum e_scif_err        // SCIF で使用される API エラーコード
{
  SCIF_SUCCESS=0,
  SCIF_ERR_BAD_CHAN,           // 存在しないチャネル番号
  SCIF_ERR_OMITTED_CHAN,       // config.h で SCI_CHx_INCLUDED の値が"0"です。
  SCIF_ERR_CH_NOT_CLOSED,      // チャネルは他のモードで動作中です。
  SCIF_ERR_BAD_MODE,           // 対応していないモードです。
  SCIF_ERR_INVALID_ARG,        // パラメータに対して引数が無効です。
  SCIF_ERR_NULL_PTR,           // null ptr を受信；要求された引数がありません。
  SCIF_ERR_BUSY,               // 処理対象の要求が既に 2 つあります。
  SCIF_ERR_IN_PROGRESS,        // 要求を処理中です。
} scif_err_t;
```

2.12 コールバック関数

本モジュールでは、SCIF 割り込みが発生したタイミングで。ユーザが設定したコールバック関数を呼び出します。

コールバック関数は、R_SCIF_Open()関数の引数 p_callback に、ユーザの関数のアドレスを格納することで設定されます。

コールバック関数の詳細は、「3 API 関数」の R_SCIF_Open ()関数をご覧ください。

2.13 FIT 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) e^2 studio 上で Smart Configurator を使用して FIT モジュールを追加する場合
e^2 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 e^2 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) e^2 studio 上で FIT Configurator を使用して FIT モジュールを追加する場合
e^2 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 e^2 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 e^2 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。

2.14 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。

「WAIT_LOOP」を記述している対象デバイス

・RX64M, RX71M グループ

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

R_SCIF_Open()

本関数は SCIF チャネルを有効にし、関連するレジスタを初期化します。また、割り込みを許可し、他の API 関数に渡すチャネルのハンドルを設定します。

Format

```
scif_err_t  R_SCIF_Open(
  uint8_t       const    chan,
  scif_mode_t   const    mode,
  scif_cfg_t *  const    p_cfg,
  void          (* const p_callback)(void *p_args),
  scif_hdl_t *  const    p_hdl
)
```

Parameters

`uint8_t  chan`

初期化するチャネル; 8～11

`scif_mode_t mode`

動作モード (以下の列挙型参照)

`scif_cfg_t  p_cfg`

設定構造体へのポインタ。構造体の要素はモードごとに定義されます（次ページ以降の列挙型参照）。

`void     (* const p_callback)(void *p_args)`

メッセージの送信／受信完了、または受信エラー発生時に割り込みから呼び出される関数へのポインタ（任意）

`scif_hdl_t  p_hdl`

チャネルのハンドルへのポインタ

本モジュールでは以下の SCIF モードに対応しています。指定したモードによって、p_cfg パラメータで使用される union 共用体の要素が決定されます。

```
typedef enum e_scif_mode    // SCIF 動作モード
{
  SCIF_MODE_OFF=0,          // チャネル未使用
  SCIF_MODE_ASYNC,          // 調歩同期式モード
  SCIF_MODE_SYNC,           // クロック同期式モード
  SCIF_MODE_END_ENUM
} scif_mode_t;
```

以下の列挙型は、設定構造体で調歩同期式モードに対して設定可能なオプションを示します。これらの値は、SCR および SMR レジスタのビット定義に対応しています。

```
typedef enum e_scif_clk
{
  SCIF_CLK_INT = 0x00,      // ボーレートの生成に内部クロックを使用
  SCIF_CLK_EXT8X = 0x03,       // 外部クロック使用; ボーレート 8x
  SCIF_CLK_EXT16X = 0x02       // 外部クロック使用; ボーレート 16x
} scif_clk_t;

typedef enum e_scif_size
{
  SCIF_DATA_7BIT = 0x40,
  SCIF_DATA_8BIT = 0x00
} scif_size_t;

typedef enum e_scif_parity_en
{
  SCIF_PARITY_ON = 0x20,
  SCIF_PARITY_OFF = 0x00
} scif_parity_en_t;

typedef enum e_parity_t
{
  SCIF_ODD_PARITY = 0x10,
  SCIF_EVEN_PARITY = 0x00
} scif_parity_t;

typedef enum e_scif_stop_t
{
  SCIF STOPBITS 2 = 0x08,
  SCIF STOPBITS 1 = 0x00
} scif_stop_t;
```

調歩同期式モードで実行時の設定オプションは以下の構造体で定義されます。この構造体は p_cfg パラメータの要素です。

```
typedef struct st_scif_uart
{
  uint32_t            baud_rate;     // ie 9600, 19200, 115200
  scif_clk_t          clk_src;
  scif_size_t         data_size;
  scif_parity_en_t    parity_en;
  scif_parity_t       parity_type;
  scif_stop_t         stop_bits;
  uint8_t             txif_priority; // txif 割り込み優先レベル; 1=low, 15=high
  uint8_t             rxif_priority; // rxif 割り込み優先レベル; 1=low, 15=high
  uint8_t             group_priority;// teif, erif, brif 割り込み優先レベル;
                                     // rx_priority より高いレベルを設定してください。
} scif_uart_t;
```

クロック同期式モードの設定構造体を以下に示します。

```
typedef struct st_scif_sync
{
  uint32_t   bit_rate;       // 1Mbps にする場合は"1000000"を設定
  bool       msb_first;
  uint8_t    int_priority;   // 割り込み優先レベル; 1=low, 15=high
} scif_sync_t;
```

p_cfg の union 共用体を以下に示します。

```
typedef union
{
  scif_uart_t   async;
  scif_sync_t   sync;
} scif_cfg_t;
```

Return Values

SCIF_SUCCESS:	*/* 成功; チャネルが初期化されました。 */*
SCIF_ERR_BAD_CHAN:	*/* MCU に対して無効なチャネル番号です。 */*
SCIF_ERR_OMITTED_CHAN:	*/* 対応する SCIF_CHx_INCLUDED が"0"です。 */*
SCIF_ERR_CH_NOT_CLOSED:	*/* チャネルは動作中です。 */*
	/ 先に R_SCIF_Close() を実行してください。 */*
SCIF_ERR_BAD_MODE:	*/* 指定されたモードには対応していません。 */*
SCIF_ERR_NULL_PTR:	*/* p_cfg または p_hdl ポインタが NULL です。 */*
SCIF_ERR_INVALID_ARG:	*/* p_cfg 構造体の要素に無効な値が含まれています。 */*

Properties

ファイル r_scif_rx_if.h にプロトタイプ宣言されています。

Description

SCIF チャネルを使用するモードに合わせて初期化して、*p_hdl に他の API 関数に渡すハンドルを設定します。また、必要な割り込みを有効にします。

Reentrant

本関数は異なるチャネルに対して再入可能（リエントラント）です。

Example : 調歩同期式モード

```
scif_cfg_t config;
scif_hdl_t Console;
scif_err_t err;

config.async.baud_rate = 115200;
config.async.clk_src = SCIF_CLK_INT;          // 内部クロック使用
config.async.data size = SCIF DATA 8BIT;
config.async.parity_en = SCIF_PARITY_OFF;
config.async.parity_type = SCIF_EVEN_PARITY; // 無視されます（パリティは無効）
config.async.stop bits = SCIF STOPBITS 1;
config.async.tx priority = 2;
config.async.rx_priority = 2;
config.async.rx_err_priority = 3; // rx_priority より高いレベルを指定してください

err = R_SCIF_Open(SCI_CH9, SCI_MODE_ASYNC, &config, MyCallback, &Console);
```

Example : クロック同期式モード

```
scif_cfg_t config;
scif_hdl_t syncHandle;
scif_err_t err;

config.sync.bit_rate = 1000000; // 1 Mbps
config.sync.msb_first = true;
config.sync.int_priority = 4;
err = R_SCIF_Open(SCI_CH8, SCI_MODE_SYNC, &config, syncCallback, &syncHandle);
```

Special Notes:

本モジュールは、BSP_PCLKA_HZ（r_bsp の mcu_info.h で定義）を使って、BRR、MDDR、SEMR.ABCS0、SEMR.BGDM、および SMR.CKS の最適値を算出しています。ただし、すべての周辺クロックとビットレートの組み合わせに対して、低いビットレート誤差を保障するものではありません。

送信または受信開始前にクロックが安定するように、Open()関数呼び出し後はアプリケーションで 1 ビット時間を待つようにしてください。

調歩同期式モードで外部クロックを使用する場合、端子機能の選択とポート端子の初期設定を最初に行ってください。以下にチャネル 9 の SCK 端子の設定例を示します。

```
MPC.PB5PFS.BYTE = 0x0A; // 端子機能の選択 PB5 SCK9; 入力クロック
PORTB.PDR.BIT.B5 = 0;   // SCK 端子を入力に設定（デフォルト）
PORTB.PMR.BIT.B5 = 1;   // SCK 端子を周辺機能に設定
```

以下に、クロック同期式モードでのチャネル 9 の SCK 端子の設定例を示します。

```
MPC.PB5PFS.BYTE = 0x0A; // 端子機能の選択 PB5 SCK9; 出力クロック
PORTB.PDR.BIT.B5 = 1;   // SCK 端子を出力に設定
PORTB.PMR.BIT.B5 = 1;   // SCK 端子を周辺機能に設定
```

コールバック関数は引数を 1 つ持ちます。この引数は構造体への void 型ポインタです（他の FIT モジュールのコールバック関数との整合性を保持するため）。以下に構造体を示します。

```
typedef struct st_scif_cb_args // コールバック関数の引数
{
  scif_hdl_t hdl;
  scif_cb_evt_t event;
} scif_cb_args_t;
```

引数“hdl”にはチャネルのハンドルが設定されます。

以下に引数に設定されるイベントの列挙型を示します。

```
typedef enum e_scif_cb_evt // コールバック関数のイベント
{
  // 調歩同期式モードのイベント
  SCIF_EVT_TX_DONE,          // Send()要求を処理；最終ビット送信
  SCIF_EVT_RX_DONE,          // Receive()要求を処理；RX FIFO 内には
                             // データがある場合もない場合もある
  SCIF_EVT_RX_BREAK,         // BREAK コンディション受信
  SCIF_EVT_RX_OVERFLOW,      // 受信 FIFO オーバランエラー
  SCIF_EVT_RX_FRAMING_ERR,   // フレーミングエラー受信
  SCIF_EVT_RX_PARITY_ERR,       // パリティエラー受信


  // クロック同期式モードのイベント
  SCIF_EVT_XCV_DONE,      // すべての要求を処理
  SCIF_EVT_XCV_ABORTED    // 転送中止；FIFO をフラッシュ
} scif_cb_evt_t;
```

SCIF_EVT_FRAMING_ERR および SCIF_EVT_PARITY_ERR イベントは、FIFO から読み出される次のデータにエラーがあることを示します。このデータは直接コールバック関数に渡されませんが、受信バッファには読み込まれます。これによって、FIFO から読み出されたバイトと要求数が一致します。調歩同期式モードでのコールバック関数のテンプレート例を以下に示します。

```
void MyCallback(void *p_args)
{
   scif_cb_args_t *args;
   args = (scif_cb_args_t *)p_args;

   switch (args->event)
   {
      case SCIF_EVT_TX_DONE:
         // TEIF 割り込みから；全データを送信
         nop();
         break;

      case SCIF_EVT_RX_DONE:
         // 最終の RXIF 割り込みから；要求されたバイトをすべて受信
         // RX FIFO にはデータがある場合もない場合もある
         nop();
         break;

      case SCIF_EVT_RX_BREAK:
         // BRIF 割り込みから；BREAK コンディション受信
         // エラーコンディションは BRIF 処理でクリアされる
         nop();
         break;

      case SCIF_EVT_RX_OVERFLOW:
         // BRIF 割り込みから；受信時にオーバランエラー発生
         // エラーコンディションは BRIF 処理でクリアされる
         nop();
         break;

      case SCIF_EVT_RX_FRAMING_ERR:
         // ERIF 割り込みから；受信時にフレーミングエラー発生
         // エラーコンディションは ERIF 処理でクリアされる
         nop();
         break;

      case SCIF_EVT_RX_PARITY_ERR:
         // ERIF 割り込みから；受信時にパリティエラー発生
         // エラーコンディションは ERIF 処理でクリアされる
         nop();
         break;
   };
}
```

クロック同期式モードのコールバック関数のテンプレート例を以下に示します。

```
void syncCallback(void *p_args)
{
   scif cb args t *args;

   args = (scif cb args t *)p args;

   if (args->event == SCIF EVT XCV DONE)
   {
      // TEIF 割り込みから；全データを送信
      // データ転送要求の処理を完了
      nop();
    }
    else if (args->event == SCIF_EVT_XCV_ABORTED)
   {
      // データ転送を中止
      nop();
   }
}
```

R_SCIF_Close()

この関数は SCIF チャネルを無効にし、関連する割り込みを禁止にします。

Format

```
scif_err_t R_SCIF_Close(scif_hdl_t const hdl);
```

Parameters

```
scif_hdl_t const hdl
```

チャネルのハンドル

Return Values

SCIF_SUCCESS:	*/* 成功; チャネルを無効にしました。*/*
SCIF_ERR_NULL_PTR:	*/* “hdl”が NULL です。*/*

Properties

ファイル r_scif_rx_if.h にプロトタイプ宣言されています。

Description

ハンドルで示された SCIF チャネルを無効にします。

Reentrant

本関数は異なるチャネルに対して再入可能（リエントラント）です。

Example

```
scif_hdl_t Console;
err = R_SCIF_Open(SCI_CH9, SCI_MODE_ASYNC, &config, MyCallback, &Console);
err = R_SCIF_Close(Console);
```

Special Notes:

本関数では、処理中の送信、または受信動作は中止されます。

R_SCIF_Send()

送信 FIFO に送信メッセージを設定します。1 度に未処理の送信要求を 2 つまで持てます。データ送信中でなければ、送信を開始します。

Format

```
scif_err_t R_SCIF_Send(
  scif_hdl_t const    hdl,
  uint8_t             *p_src,
  uint16_t const      length
);
```

Parameters

`scif_hdl_t const hdl`

チャネルのハンドル

`uint8_t *p_src`

送信データへのポインタ

`uint16_t const length`

送信バイト数

Return Values

```
SCIF_SUCCESS:           /* 送信メッセージを FIFO に設定; */
                        /* 送信アイドル状態なら、送信を開始します。*/
SCIF_ERR_NULL_PTR:      /* “hdl”または“p_src”が NULL です。*/
SCIF_ERR_BAD_MODE:      /* 対応していないモードです。*/
SCIF_ERR_ INVALID_ARG:  /* “length”の値が“0”です。*/
SCIF_ERR_BUSY:          /* 要求を処理できません。要求が既に 2 つあります。*/
```

Properties

ファイル r_scif_rx_if.h にプロトタイプ宣言されています。

Description

要求が正常に処理されると、本モジュールは SCIF_SUCCESS を返します。処理待ちの要求が既に 2 つある場合は SCIF_ERR_BUSY を返します。メッセージが FIFO のサイズより大きい場合、データが config.h で設定されたしきい値に達するごとに、割り込みを使って自動的に FIFO にデータを読み込みます。

送信データがなくなると、Open()関数でコールバック関数を指定していれば、調歩同期式モードの場合は SCIF_EVT_TX_DONE イベントを、クロック同期式モードの場合は SCIF_EVT_XCV_DONE イベントをそれぞれコールバック関数に渡します。コールバック関数が指定されていなければ、Control()コマンドを使って、アプリケーションでポーリングして完了を確認する必要があります。

メッセージごとに送信完了を把握する必要がある場合､1 度に処理する Send()要求は 1 つとしてください。本モジュールでは、データを連続的に処理することを前提とし、DONE イベントを使って、全データの送信完了を示します。

Reentrant

この関数は異なるチャネルに対して再入可能（リエントラント）です。

Example 1: 調歩同期式モード　ブロッキング

```
uint8_t g_data_block[128];

scif_cfg_t config;
scif_hdl_t hdl;
scif_err_t err;

err = R_SCIF_Open(SCI_CH9, SCI_MODE_ASYNC, &config, NULL, &hdl);
:

/* トランスミッタがメッセージ送信可能かどうかを確認する。必要に応じて待機する。*/
while (R_SCIF_Send(hdl, g_data_block, 128) == SCIF_ERR_TX_BUSY)
{
   /* 送信データが FIFO に配置されるまで待機 */
}

/*メッセージの送信完了までブロック */
while (R SCIF Control(hdl,SCIF CMD CHECK TX DONE, NULL) == SCIF ERR IN PROGRESS)
{
   /* 送信完了の待機中に、必要な場合は別の処理を行う*/
}
```

Example 2: 調歩同期式モード　ノンブロッキング

```
uint8 t g data block[128];

scif cfg t config;
scif_hdl_t hdl;
scif_err_t err;

err = R_SCIF_Open(SCI_CH9, SCI_MODE_ASYNC, &config, MyCallback, &hdl);
  :
/* 処理待ちの要求が 1 つ、または要求がない場合、Send()を即時発行できます。*/
R_SCIF_Send(hdl, g_data_block, 128);

void MyCallback(void *p_args)
{
   scif_cb_args_t *args;

   args = (scif_cb_args_t *)p_args;
   switch (args->event)
   {
      case SCIF_EVT_TX_DONE:
         // 全データを正常に送信
         break;

      case SCIF_EVT_RX_BREAK:
         // break を受信; エラーコンディションに対応
         R_SCIF_Control(args->hdl, SCIF_CMD_RESET_TX, NULL);
         R_SCIF_Control(args->hdl, SCIF_CMD_RESET_RX, NULL);
         break;
   };
}
```

Example 3: クロック同期式モード　ブロッキング

```
#define STRING "Test String"

scif_cfg_t config;
scif_hdl_t lcdHandle;
scif_err_t err;

err = R_SCIF_Open(SCI_CH8, SCI_MODE_SYNC, &config, NULL, &lcdHandle);
  :

/* トランスミッタがメッセージ送信可能かどうかを確認する。必要に応じて待機する。*/
while (R_SCIF_Send(lcdHandle, STRING1, sizeof(STRING1)) == SCIF_ERR_BUSY)
{
   /* 送信データが FIFO に配置されるまで待機 */
}

/* メッセージ送信完了までブロック */
while(R SCIF Control(lcdHandle,SCIF CMD CHECK XCV DONE,NULL)
       == SCIF ERR IN PROGRESS)
{
   /*送信完了の待機中に、必要であれば別の処理を行う */
}
```

Example 4: クロック同期式モード　ノンブロッキング

```
#define STRING "Test String"

scif_cfg_t config;
scif_hdl_t lcdHandle;
scif_err_t err;

err = R_SCIF_Open(SCI_CH8, SCI_MODE_SYNC, &config, syncCallback, &lcdHandle);
:

/* 処理待ちの要求が 1 つ、または要求がない場合、Send()を即時発行できます。*/
R_SCIF_Send(lcdHandle, STRING1, sizeof(STRING1));

void syncCallback(void *p_args)
{
   scif_cb_args_t *args;

   args = (scif_cb_args_t *)p_args;
   if (args->event == SCIF_EVT_XCV_DONE)
   {
      // データ転送完了 ; 任意の処理をここで行う
      // nop();
   }
   else if (args->event == SCIF_EVT_XCV_ABORTED)
   {
      // データ転送中止 ; 任意の処理をここで行う
   }
}
```

Special Notes:

クロック同期式モードでは、メッセージの送信／受信（Send()、Receive()、SendReceive()によって）を行うために、SCIF がクロックを出力します。このモードでは、いずれかの転送要求を 1 度に 2 つまで持てます。そのため、Send()要求が発行されていなくても、SCIF_ERR_BUSY が返ってくることがあります。

前に指定したメッセージの送信完了が確認できるまで、p_src で指定されたのと同じバッファを使用しないでください。使用した場合、送信中のメッセージのデータが壊れる可能性があります。

R_SCIF_Receive()

受信 FIFO から受信メッセージを取得します。1 度に未処理の受信要求を 2 つまで持てます。クロック同期式モードでは、送受信中でなければ、クロックの生成を開始します。

Format

```
scif_err_t R_SCIF_Receive(
  scif_hdl_t const    hdl,
  uint8_t             *p_dst,
  uint16_t const      length
);
```

Parameters

`scif_hdl_t const hdl`

チャネルのハンドル

`uint8_t *p_dst`

バッファのデータ取り込み先へのポインタ

`uint16_t length`

読み込むデータのバイト数

Return Values

SCIF_SUCCESS:	*/* 受信 FIFO からメッセージを取得; クロック同期の場合、*/*
	/ 送受信アイドル状態なら、クロックの生成を開始します。*/*
SCIF_ERR_NULL_PTR:	*/* "hdl"が NULL です。*/*
SCIF_ERR_BAD_MODE:	*/* 対応していないモードです。*/*
SCIF_ERR_ INVALID_ARG:	*/* "length"の値が"0"です。*/*
SCIF_ERR_BUSY:	*/* 要求を処理できません。要求が既に 2 つあります。*/*

Properties

ファイル r_scif_rx_if.h にプロトタイプ宣言されています。

Description

要求が正常に処理されると、本モジュールは SCIF_SUCCESS を返します。処理待ちの要求が既に 2 つある場合は SCIF_ERR_BUSY を返します。メッセージが FIFO のサイズより大きい場合、本モジュールはデータが config.h で設定されたしきい値に達するごとに、割り込みを使って自動的に FIFO を読み出します。残りのデータがしきい値未満の場合、本モジュールが自動的にしきい値を調整します。

受信データがなくなると、コールバック関数を Open()関数で指定していれば、調歩同期式モードの場合は SCIF_EVT_RX_DONE イベントを、クロック同期式モードの場合は SCIF_EVT_XCV_DONE イベントをそれぞれコールバック関数に渡します。コールバック関数が指定されていなければ、Control()コマンドを使って、アプリケーションでポーリングして完了を確認する必要があります。ただし、受信中に発生したエラーは、コールバック関数からのみレポートされます。

メッセージごとに受信完了を把握する必要がある場合、1 度に処理する Receive()要求は 1 つとしてください。本モジュールでは、データを連続的に処理することを前提とし、DONE イベントを使って、要求された全データの受信完了を示します。

Reentrant

この関数は異なるチャネルに対して再入可能（リエントラント）です。

Example 1: 調歩同期式モード　ブロッキング

```
uint8_t g_data_block[128];

scif_cfg_t config;
scif_hdl_t hdl;
scif_err_t err;

err = R_SCIF_Open(SCI_CH9, SCI_MODE_ASYNC, &config, NULL, &hdl);
  :
/* レシーバがメッセージの受信可能かどうかを確認する。必要に応じて待機する。 */
while (R_SCIF_Receive(hdl, g_data_block, 128) == SCIF_ERR_RX_BUSY)
{
  /* Receive()の処理完了まで待機 */
}
/* 受信完了までブロック */
while (R_SCIF_Control(hdl,SCIF_CMD_CHECK_RX_DONE,NULL) == SCIF_ERR_IN_PROGRESS)
{
  /* 受信完了の待機中に、必要であれば別の処理を行う */
}
```

Example 2: 調歩同期式モード　ノンブロッキング

```
uint8 t g data[8];

scif cfg t config;
scif hdl t hdl;
scif err t err;

err = R_SCIF_Open(SCI_CH9, SCI_MODE_ASYNC, &config, MyCallback, &hdl);
  :

/* レシーバがメッセージの受信可能かどうかを確認する。必要に応じて待機する。
* 要求完了までの期間、ブロックしない。
*/
while (R SCIF Receive(hdl, g data, 8) == SCIF ERR RX BUSY);
{
  /* Receive()の処理完了まで待機 */
}

/* 受信イベントの処理例 */

void MyCallback(void *p_args)
{
   scif_cb_args_t *args;
   static bool    err_flg=false;
   uint8_t        byte;
   args = (scif_cb_args_t *)p_args;
   switch (args->event)
   {
      case SCIF EVT RX FRAMING ERR:
      case SCIF EVT RX PARITY ERR:
```

```
            /* メッセージの受信を継続，一方でエラー検出を示すフラグをセット */
            err_flg = true;
            break;

        case SCIF_EVT_RX_OVERFLOW:
            /* オーバランエラー発生。送信元に abort を発行し、リフレッシュを開始するために
            * err_flg をリセットします。本モジュールは、break 生成時、自動的に FIFO を
            * リセットします。
            */
            R SCIF Control(args->hdl, SCIF CMD GENERATE BREAK, NULL);
            err flg = false;
            break;

        case SCIF_EVT_RX_BREAK:
            /* break を受信。送信 FIFO、受信 FIFO、err_flg をリセット */
            R SCIF Control(args->hdl, SCIF CMD RESET TX, NULL);
            R SCIF Control(args->hdl, SCIF CMD RESET RX, NULL);
            err flg = false;
            break;

        case SCIF_EVT_RX_DONE:
            /* メッセージ受信完了。Err_flg に応じて ACK または NAK を発行 */
            byte = (err flg == true) ? NAK : ACK;
            R SCIF Send(hdl, &byte,1);
            err flg = false;
            break;
    };
}
```

Example 3: クロック同期式モード　ブロッキング

```
uint8_t  g_block[2][128];

scif_cfg_t config;
scif_hdl_t hdl;
scif_err_t err;

err = R_SCIF_Open(SCI_CH9, SCI_MODE_SYNC, &config, NULL, &hdl);
  :

/* Receive()の呼び出し+完了待機を 2 回行う */
while (R_SCIF_Receive(hdl, &g_data_block[0], 128) == SCIF_ERR_XCV_BUSY)
{
  /* Receive()の処理完了まで待機 */
}

while (R_SCIF_Receive(hdl, &g_data_block[1], 128) == SCIF_ERR_XCV_BUSY)
{
  /* Receive()の処理完了まで待機 */
}
```

```
// (上記の要求を、最初のアドレスで 256 バイトの単一の要求に置換可能。)
while (R_SCIF_Control(hdl,SCIF_CMD_CHECK_XCV_DONE, NULL)
          == SCIF_ERR_IN_PROGRESS)
{
  /* 受信完了の待機中に、必要であれば別の処理を行う */
}
```

Example 4: クロック同期式モード　ノンブロッキング

```
uint8 t  sensor cmd,sync buf[10];
scif cfg t config;
scif hdl t hdl;
scif err t err;

err = R SCIF Open(SCI CH9, SCI MODE SYNC, &config, syncCallback, &hdl);

/* センサにコマンドを送信し、読み出したデータを提供 */
sensor cmd = SNS CMD READ LEVEL;

/* FIFO が空かどうかを確認; 要求を 2 つまで持てます。*/
R_SCIF_Send(hdl, &sensor_cmd, 1);
R_SCIF_Receive(hdl, sync_buf, 4);

/* 応答待ちをしない */
```

Special Notes:

クロック同期式モードでは、メッセージの送信／受信（Send()、Receive()、SendReceive()によって）を行うために SCIF がクロックを出力します。このモードでは、いずれかの転送要求を 1 度に 2 つまで持てます。そのため、Receive()要求が発行されていなくても、SCIF_ERR_BUSY が返ってくることがあります。

前に指定したメッセージの受信完了が確認できるまで、p_dst で指定されたのと同じバッファを使用しないでください。使用した場合、前回受信したメッセージのデータが壊れる可能性があります。

R_SCIF_SendReceive()

クロック同期式モードでのみ使用できます。データの送受信を同時に行います。

Format

```
scif_err_t R_SCIF_SendReceive(
  scif_hdl_t const    hdl,
  uint8_t             *p_src,
  uint8_t             *p_dst,
  uint16_t const      length
);
```

Parameters

`scif_hdl_t const hdl`

チャネルのハンドル

`uint8_t p_src`

送信データへのポインタ

`uint8_t p_dst`

データを読み込むバッファへのポインタ

`uint16_t length`

送信バイト数

Return Values

```
SCIF_SUCCESS:          /* 転送データを FIFO に配置; 送受信アイドル状態なら */
                       /* 送信／受信を開始します。 */
SCIF_ERR_NULL_PTR:     /* “hdl”が NULL です。*/
SCIF_ERR_BAD_MODE:     /* モードがクロック同期式ではありません。*/
SCIF_ERR_ INVALID_ARG: /* “length”の値が“0”です。*/
SCIF_ERR_BUSY:         /* 要求を処理できません。要求が既に 2 つあります。*/
```

Properties

ファイル r_scif_rx_if.h にプロトタイプ宣言されています。

Description

本関数は送信と受信を同時に行います。要求が正常に処理されると、SCIF_SUCCESS を返します。処理待ちの要求が既に 2 つある場合は SCIF_ERR_BUSY を返します。メッセージが FIFO サイズより大きい場合、データが config.h で設定されたしきい値に達すると、割り込みを使って自動的に FIFO を処理します。

送信および受信データがなくなると、Open()関数でコールバック関数を指定していれば、SCIF_EVT_XCV_DONE イベントをコールバック関数に渡します。コールバック関数が指定されていなければ、Control()コマンドを使って、アプリケーションでポーリングして完了を確認する必要があります。

各メッセージの送信および受信の完了を把握する必要がある場合、1 度に処理する SendReceive()要求は 1 つとしてください。本モジュールでは、データを連続的に処理することを前提とし、DONE イベントを使って、全データの送信／受信が完了したことを示します。

Reentrant

この関数は異なるチャネルに対して再入可能（リエントラント）です。

Example: ブロッキング

```
scif_hdl_t  hdl;
scif_err_t  err;
uint8_t     out_buf[2] = {SF_CMD_READ_STATUS_REG, SCIF_CFG_DUMMY_TX_BYTE };
uint8_t     in_buf[2] = {0x55, 0x55}; // 不正な値を設定
  :

/* 2 バイトのデータを送受信。1 バイト目はコマンドで出力（入力バイトは無視）、
* 2 バイト目は応答で入力（ダミーバイトが出力される）
*/

/* FIFO が空かどうかを確認 */
R_SCIF_SendReceive(hdl, out_buf, in_buf, 2);

while (R_SCIF_Control(hdl, SCI_CMD_CHECK_XCV_DONE, NULL) == SCIF_ERR_BUSY)
{
  /* 完了を待機 */
}
// 応答は in_buf[1]に入る
```

Special Notes:

クロック同期式モードでは、メッセージの送信／受信（Send()、Receive()、SendReceive()によって）を行うために、SCIF がクロックを出力します。このモードでは、いずれかの転送要求を 1 度に 2 つまで処理できます。そのため、SendReceive()要求が発行されていなくても、SCIF_ERR_BUSY が返ってくることがあります。

前に指定したメッセージの処理完了が確認できるまで､p_src および p_dst で指定されたのと同じバッファを使用しないでください。使用した場合、前回受信したメッセージのデータが壊れる可能性があります。

R_SCIF_Control()

この関数は、SCIF チャネルに関する特定のハードウェアまたはソフトウェア動作を制御します。

Format

```
scif_err_t R_SCIF_Control(
  scif_hdl_t const    hdl,
  scif_cmd_t const    cmd,
  void                *p_args
);
```

Parameters

scif_hdl_t const hdl

チャネルのハンドル

scif_cmd_t const cmd

実行するコマンド（以下の列挙型参照）

void_args

コマンド固有の引数へのポインタ（次ページの説明参照） 。引数は void 型にキャストされます。

有効な cmd 値を以下に示します。

```
typedef enum e_scif_cmd          // SCIF Control() コマンド
{
   // 両モード共通
   SCIF_CMD_CHANGE_BAUD,         // ボー／ビットレートを変更

   // 調歩同期式モードで使用するコマンド
   SCIF_CMD_EN_FLOW_CTRL,        // CTS/RTS フロー制御有効
   SCIF_CMD_EN_NOISE_CANCEL,     // ノイズ除去有効
   SCIF_CMD_EN_MSB_FIRST,        // MSB ファーストで送受信
   SCIF_CMD_GENERATE_BREAK,      // break コンディション生成; FIFO をリセット
   SCIF_CMD_TX_BYTES_REMAINING,  // 未送信バイト数
   SCIF_CMD_RX_BYTES_PENDING,    // 未受信バイト数
   SCIF_CMD_CHECK_TX_DONE,       // 送信要求の完了を確認; 完了の場合 SCIF_SUCCESS
   SCIF_CMD_CHECK_RX_DONE,       // 受信要求の完了を確認; 完了の場合 SCIF_SUCCESS
   SCIF_CMD_RESET_TX,            // 送信要求を中止; 送信 FIFO をリセット
   SCIF_CMD_RESET_RX,            // 受信要求を中止; 受信 FIFO をリセット

   // クロック同期式モードで使用するコマンド
   SCIF_CMD_CHECK_XCV_DONE,      // 送信/受信/送受信要求の完了を確認;
                                 // 完了の場合 SCIF_SUCCESS
   SCIF_CMD_RESET_XCV            // 転送要求を中止; FIFO をリセット
} scif_cmd_t;
```

ほとんどのコマンドは引数を必要としません。その場合、引数“p_args”には NULL または FIT_NO_PTR が設定されます。SCIF_CMD_CHANGE_BAUD で使用する引数の構造体を以下に示します。このコマンドは、外部クロック使用時の調歩同期式モードでは使用されません。

```
typedef struct  sci baud
{
   uint32_t pclk;     // PCLKA の速度; ie 120000000 (120 MHz)
   uint32 t rate;     // ie 9600, 19200, 115200
} sci_baud_t;
```

SCIF_CMD_TX_BYTES_REMAINING および SCIF_CMD_RX_BYTES_PENDING の引数には、カウント値を格納する uint16_t 型変数へのポインタが設定されます。

コマンド SCIF_CMD_CHECK_TX_DONE、SCIF_CMD_CHECK_RX_DONE、および SCIF_CMD_CHECK_XCV_DON は、すべての要求が処理されると、SCIF_SUCCESS を返します。そうでない場合は SCIF_ERR_IN_PROGRESS を返します。

【注】 SCIF_CMD_RESET_TX コマンドが発行されたときに、メッセージが送信中の場合、その送信はその時点で中止されます。このとき、処理中のデータの送信完了まで待たずに送信を中止します。この場合、最後に送信された不完全なデータをレシーバがフレーミングエラーとして処理できるように、送信を再開する前に 1 バイト時間を待機することが推奨されます。

Return Values

SCIF_SUCCESS:	*/* 成功; チャネルが初期化されました。 */*
SCIF_ERR_NULL_PTR:	*/* “hdl”または“p_args”ポインタが NULL です。（必要な場合） */*
SCIF_ERR_BAD_MODE:	*/* 対応していないモードです。 */*
SCIF_ERR_ INVALID_ARG:	*/* “cmd”の値、または“p_args”の要素が無効な値です。 */*

Properties

ファイル r_scif_rx_if.h にプロトタイプ宣言されています。

Description

本関数を使って、ハードウェア機能の設定、および本モジュールの設定の変更が行えます。また、本モジュールの状態を取得できます。

Reentrant

この関数は異なるチャネルに対して再入可能（リエントラント）です。

Example 1: 調歩同期式モード

```
scif_hdl_t  Console;
scif_cfg_t  config;
scif_baud_t baud;
scif_err_t  err;
uint16_t    cnt;
  :
R_SCIF_Open(SCI_CH9, SCIF_MODE_ASYNC, &config, MyCallback, &Console);
R_SCIF_Control(Console, SCIF_CMD_EN_NOISE_CANCEL, NULL);
R_SCIF_Control(Console, SCIF_CMD_EN_MSB_FIRST, NULL);
  :
/* クロックを低消費電力モードに切り替えるためにボーレートをリセット */
baud.pclk = 8000000;    // 8MHz
baud.rate = 19200;
R_SCIF_Control(Console, SCIF_CMD_CHANGE_BAUD, &baud);
  :
/*大きいデータの送信開始後、残りの送信バイト数を確認*/
R_SCIF_Control(Console, SCIF_CMD_TX_BYTES_REMAINING, &cnt);
// for progress bar: (message size - cnt)/(message size) = % complete
  :
/*大きいデータの受信開始後、残りの受信バイト数を確認*/
R SCIF Control(Console, SCIF CMD RX BYTES PENDING, &cnt);
// for progress bar: (request size - cnt)/(request size) = % complete
  :
```

Example 2: クロック同期式モード

```
scif_cfg_t config;
scif_hdl_t syncHandle;
scif_err_t err;

config.sync.bit_rate = 1000000;  // 1 Mbps
config.sync.msb_first = true;
config.sync.int_priority = 4;
err = R_SCIF_Open(SCI_CH8, SCI_MODE_SYNC, &config, syncCallback, &syncHandle);
  :

// 大きいメッセージの転送開始後、転送を中止
R_SCIF_Control(syncHandle, SCIF_CMD_RESET_XCV, NULL);
```

Special Notes:

メッセージが送信されたかどうかの判定に SCIF_CMD_TX_BYTES_REMAINING で読み込まれた値は使用しないでください。値が“0”でも、シフトレジスタにビットが残っているかもしれません。メッセージ送信の判定には SCIF_CMD_TX_DONE を使用してください。

SCIF_CMD_CHANGE_BAUD の実行後は、新規速度でクロックが安定するための期間として 1 ビット時間を待機してください。ビット時間は低速のビットレートで測定してください。

SCIF_CMD_GENERATE_BREAK 実行後は、通信再開前に 2 ビット時間を待機してください。この間に Send()または Receive()が呼び出された場合、break が完了するまでの間、SCIF_ERR_BUSY が返されます。break コンディションは 1.5～2.0 バイト時間継続します。

本モジュールは、アルゴリズムを使って、BRR、MDDR、SEMR.ABCS0、SEMR.BGDM、SEMR.BRME、SEMR.MDDRS および SMR.CKS の最適値を算出しています。ただし、すべての周辺クロックとビットレートの組み合わせに対して、低いビットレート誤差を保障するものではありません。

SCIF_CMD_EN_FLOW_CTRL を使用する場合、端子機能の選択とポート端子の設定を最初に行ってください。以下にチャネル 9 の CTS 端子と RTS 端子の設定例を示します。

```
MPC.PB4PFS.BYTE = 0x0B; // 端子機能の選択 PB4 CTS
PORTB.PDR.BIT.B4 = 0;   // CTS 端子を入力に設定
PORTB.PMR.BIT.B4 = 1;   // CTS 端子を周辺機能に設定

MPC.PB5PFS.BYTE = 0x0B; // 端子機能の選択 PB5 RTS
PORTB.PDR.BIT.B5 = 1;   // RTS 端子を出力に設定
PORTB.PMR.BIT.B5 = 1;   // RTS 端子を周辺機能に設定
```

R_SCIF_GetVersion()

この関数は実行時に本モジュールのバージョンを返します。

Format

```
uint32_t R_SCIF_GetVersion(void)
```

Parameters

なし

Return Values

本モジュールのバージョン

Properties

ファイル r_scif_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。バージョン番号は符号化され、最上位の 2 バイトがメジャーバージョン番号を、最下位の 2 バイトがマイナーバージョン番号を示しています。

Reentrant

この関数は再入可能（リエントラント）です。

Example

```
uint32_t version;
  :
version = R_SCIF_GetVersion();
```

Special Notes:

なし

4. 端子設定

SCI FIFO FIT モジュールを使用するためには、マルチファンクションピンコントローラ（MPC）で周辺機能の入出力信号を端子に割り付ける（以下、端子設定と称す）必要があります。

端子設定は、R_SCIF _Open 関数を呼び出す前に行ってください。

e^2 studio の場合は「FIT Configurator」または「Smart Configurator」の端子設定機能を使用することができます。FIT Configurator、Smart Configurator の端子設定機能を使用すると、端子設定画面で選択したオプションに応じて、ソースファイルが出力されます。そのソースファイルで定義された関数を呼び出すことにより端子を設定できます。詳細は表 4.1 を参照してください。

表 4.1　FIT コンフィグレータが出力する関数一覧

使用マイコン	選択したオプション	出力される関数名	備考
RX64M RX71M	SCIF_SCIF8	R_SCIF_PinSet_SCIF8()	SCIF8 を使用する場合
	SCIF_SCIF9	R_SCIF_PinSet_SCIF9()	SCIF9 を使用する場合
	SCIF_SCIF10	R_SCIF_PinSet_SCIF10()	SCIF10 を使用する場合
	SCIF_SCIF11	R_SCIF_PinSet_SCIF11()	SCIF11 を使用する場合

5. デモプロジェクト

デモプロジェクトはスタンドアロンプログラムです。デモプロジェクトには、FIT モジュールとそのモジュールが依存するモジュール（例：r_bsp）を使用する main()関数が含まれます。

5.1 scif_demo_rskrx64m

本デモは SCIF API (r_scif_rx)のシンプルなデモで、USB の仮想 COM ポートに接続された SCIF のチャネル 8 を使ってターミナルと通信します。SCIF モジュールのバージョンが SCIF チャネルを介してターミナルに送信された所で、ユーザにターミナルで文字を入力するように促します。デモは連続的なループに入り、文字が入力されて、160 バイトのバッファ 2 つ分の char データが送信されるのを待ちます。データが送信されると、送信バイト数とデータ送信時に要求された TXIF 割り込み数の概要がターミナルに送信／表示されます。このデモでは、r_scif_r_config.h の SCIF_CFG_CHx_TX_FIFO_THRESHOLD の設定値が、SCIF Tx FIFO を使った 320 バイトの全データの処理に要求される割り込み数にどのように影響するのかを見ることができます。

設定と実行

1. r_scif_rx_config.h でチャネル 8 を有効にしてください。
   => #define SCIF_CFG_CH8_INCLUDED (1)
2. RSKRX64M のボードを準備します。
   - ジャンパ J12、J14、J16、J18 をオフにします。
   - ジャンパワイヤを使って、J12 端子 2 を J16 端子 3 に接続します。
   - ジャンパワイヤを使って、J14 端子 2 を J18 端子 3 に接続します。

   これで CH8 の Tx/Rx 信号が仮想 COM USB の Tx/Rx 信号に接続されます。
3. RSK ボードのシリアルポートを PC のシリアルポートに接続します。本デモでは、RSKRX64M のシリアルポートを USB 仮想 COM インタフェースに接続します。この場合、Renesas USB シリアルデバイスドライバで USB ポートを PC に接続してください。
   USB が仮想 COM ポートとして PC に列挙されます。COM ポート番号を確認してください。
4. PC で「Tera Term」などのターミナルエミュレーションプログラムを開き、仮想 COM インタフェースに割り当てられたシリアル COM ポートを選択します。本サンプルアプリケーションの設定と一致するように、ターミナルのシリアルの設定を以下のように行います。
   - ボーレート: 115200
   - データ長: 8 ビットデータ
   - パリティ: なし
   - ストップビット: 1 ビット
   - フロー制御: しない
5. 本サンプルアプリケーションをビルドして RSK ボードにダウンロードします。デバッガでアプリケーションを実行します。
6. これで、デモによって表示されるプロンプト「Enter a char>」がターミナルで表示されるはずです。

対応ボード

RSKRX64M

5.2 scif_demo_rskrx71m

本デモは SCIF API (r_scif_rx)のシンプルなデモで、USB の仮想 COM ポートに接続された SCIF のチャネル 8 を使ってターミナルと通信します。SCIF モジュールのバージョンが SCIF チャネルを介してターミナルに送信された所で、ユーザにターミナルで文字を入力するように促します。デモは連続的なループに入り、文字が入力されて、160 バイトのバッファ 2 つ分の char データが送信されるのを待ちます。データが送信されると、送信バイト数とデータ送信時に要求された TXIF 割り込み数の概要がターミナルに送信／表示されます。このデモでは、r_scif_r_config.h の SCIF_CFG_CHx_TX_FIFO_THRESHOLD の設定値が、SCIF Tx FIFO を使った 320 バイトの全データの処理に要求される割り込み数にどのように影響するのかを見ることができます。

設定と実行

1. r_scif_rx_config.h でチャネル 8 を有効にしてください。
   => #define SCIF_CFG_CH8_INCLUDED (1)
2. RSKRX71M のボードを準備します。
   - ジャンパ J12、J14、J16、J18 をオフにします。
   - ジャンパワイヤを使って、J12 端子 2 を J16 端子 3 に接続します。
   - ジャンパワイヤを使って、J14 端子 2 を J18 端子 3 に接続します。

   これで CH8 の Tx/Rx 信号が仮想 COM USB の Tx/Rx 信号に接続されます。
3. RSK ボードのシリアルポートを PC のシリアルポートに接続します。本デモでは、RSKRX71M のシリアルポートを USB 仮想 COM インタフェースに接続します。この場合、Renesas USB シリアルデバイスドライバで USB ポートを PC に接続してください。
   USB が仮想 COM ポートとして PC に列挙されます。COM ポート番号を確認してください。
4. PC で「Tera Term」などのターミナルエミュレーションプログラムを開き、仮想 COM インタフェースに割り当てられたシリアル COM ポートを選択します。本サンプルアプリケーションの設定と一致するように、ターミナルのシリアルの設定を以下のように行います。
   - ボーレート: 115200
   - データ長: 8 ビットデータ
   - パリティ: なし
   - ストップビット: 1 ビット
   - フロー制御: しない
5. 本サンプルアプリケーションをビルドして RSK ボードにダウンロードします。デバッガでアプリケーションを実行します。
6. これで、デモによって表示されるプロンプト「Enter a char>」がターミナルで表示されるはずです。

対応ボード

RSKRX71M

5.3 ワークスペースにデモを追加する

デモプロジェクトは、本アプリケーションノートで提供されるファイルの FITDemos サブディレクトリにあります。ワークスペースにデモプロジェクトを追加するには、「ファイル」→「インポート」を選択し、「インポート」ダイアログから「一般」の「既存プロジェクトをワークスペースへ」を選択して「次へ」ボタンをクリックします。「インポート」ダイアログで「アーカイブ・ファイルの選択」ラジオボタンを選択し「参照」ボタンをクリックして FITDemos サブディレクトリを開き、使用するデモの zip ファイルを選択して「完了」をクリックします。

5.4　デモのダウンロード方法

デモプロジェクトは、RX Driver Package には同梱されていません。デモプロジェクトを使用する場合は、個別に各 FIT モジュールをダウンロードする必要があります。「スマートブラウザ」の「アプリケーションノート」タブから、本アプリケーションノートを右クリックして「サンプル・コード（ダウンロード）」を選択することにより、ダウンロードできます。

6. 付録

6.1 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 6.1 動作確認環境 (Rev.1.22)

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
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	Rev1.22

表 6.2 動作確認環境 (Rev.1.21)

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 V7.1.0
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 V2.06.00
	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	Rev1.21

6.2 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「This MCU is not supported by the current r_sci_iic_rx module.」エラーが発生します。

A : 追加した FIT モジュールがユーザプロジェクトのターゲットデバイスに対応していない可能性があります。追加した FIT モジュールの対象デバイスを確認してください。

(3) 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 – SCIF_CFG_xxx_xxx - …」エラーが発生します。

A : “r_scif_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_scif_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「2.8 コンパイル時の設定」を参照してください。

(4) Q : シリアル通信が動作しない現象が発生します。

A : 正しく端子設定が行われていない可能性があります。本 FIT モジュールを使用する場合は端子設定が必要です。詳細は「4 端子設定」を参照してください。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.20	2017.05.31	—	初版発行
		ﾌﾟﾛｸﾞﾗﾑ	クロック同期式モードで高速で動作時、余分なクロックが送信される問題に対応
1.21	2018.12.07	1	「関連ドキュメント」に以下のドキュメントを追加: Renesas e2 studio スマート・コンフィグレータユーザーガイド (R20AN0451)
		4	「2.3 ソフトウェアの要求」内容を変更。
			「2.4 制限事項」を削除。
			「2.5 対応ツールチェーン」から「2.4 サポートされているツールチェーン」に章題を変更し、内容を変更。
		5	「2.5 使用する割り込みベクタ」を追加。
		10	「2.11 モジュールの追加方法」内容を変更。
		34	「4.デモプロジェクト」内容を変更。
		36	「4.4 デモのダウンロード方法」を追加。
		37	「5.1 動作確認環境」を追加。
			「5.2 トラブルシューティング」を追加。
		38	「テクニカルアップデートの対応について」を追加。
		ﾌﾟﾛｸﾞﾗﾑ	FIT モジュールのサンプルプログラムをダウンロードするためのアプリケーションノートのドキュメント番号を xml ファイルに追加。
1.22	2019.04.01	—	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。
		3	「1.1 SCI FIFO FIT モジュールとは」を追加
		4	「1.3 API の概要」を移動
		6	「2.6 ヘッダファイル」を変更 「2.7 整数型」を変更
		7	「2.8 コンパイル時の設定」を変更
		8, 9	「2.9 コードサイズ」を変更
		9	「2.10 引数」を変更 「2.11 戻り値」を移動
		10	「2.12 コールバック関数」を追加
		11	「2.14 for 文、while 文、do while 文について」を追加
		12	「R_SCIF_Open()」を変更
		20	「R_SCIF_Close()」を変更
		21	「R_SCIF_Send()」を変更
		25	「R_SCIF_Receive」を変更
		29	「R_SCIF_SendReceive()」を変更
		31	「R_SCIF_Control()」を変更
		35	「R_SCIF_GetVersion()」を変更
		36	「4. 端子設定」を追加

Rev.	発行日	改訂内容	
		ページ	ポイント
1.22	2019.04.01	40	「6.1 動作確認環境」 Rev.1.22 に対応する表を追加

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。