RX ファミリ

SCI 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）を使用したシリアルコミュニケーションインタフェース (SCI)モジュールについて説明します。

本モジュールは、RX MCU の SCI の全チャネルに対応し、調歩同期式、クロック同期式、および簡易 SPI (SSPI)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 FIT モジュールと称します。

対象デバイス

- RX110、RX111、RX113 グループ
- RX130 グループ
- RX230、RX231、RX23T グループ
- RX24T グループ
- RX24U グループ
- RX64M グループ
- RX65N、RX651 グループ
- RX66T グループ
- RX71M グループ
- RX72T 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。

ターゲットコンパイラ

ルネサスエレクトロニクス製 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環境に関する詳細な内容は、セクション「6.1 動作確認環境」を参照してください。

目次

1. 概要

1.1 SCI 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、「2.12 FIT モジュールの追加方法」を参照してください。

1.2 SCI FIT モジュールの概要

SCI FIT モジュールは、RX MCU のグループに応じて、下記の SCI 周辺機能をサポートしています。

表 1.1 MCU グループに対応する SCI 周辺機能の一覧

	SCIc	SCId	SCIe	SCIf	SCIg	SCIh	SCIi	SCIj
RX110			○	○				
RX111			○	○				
RX113			○	○				
RX130					○	○		
RX230	○	○						
RX231	○	○						
RX23T					○			
RX24T					○			
RX24U					○			
RX64M					○	○		
RX65N					○	○	○	
RX651					○	○	○	
RX66T								○
RX71M					○	○		
RX72T						○	○	○

ご使用の RX MCU のハードウェアマニュアルで、シリアルコミュニケーションインタフェース (SCI)の章をご覧いただき、SCI 周辺機能についてご確認ください。本モジュールでは、基本的な UART、簡易 SPI モード（マスタのみ）、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
- SCK 端子への外部クロック出力
- 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:

- 拡張モード（チャネル 12）
- マルチプロセッサモード（全チャネル）
- イベントリンク
- DMAC/DTC データ転送

サポートチャネルについて

本モジュールは SCI 周辺機能に装備されているすべてのチャネルをサポートします。使用チャネルは r_sci_rx_config.h で定義できます。使用しないチャネルはこの定義を設定することで、コンパイル時の定義で省くことができ、RAM 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_SCI_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_SCI_Open()関数で、SCI 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_SCI_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。

割り込みと送受信について

本モジュールでは TXI、TEI、RXI、および ERI 割り込みを使用します。調歩同期式モードでは、本モジュールは、リングバッファを使用して、入力データおよび出力データをキューに置きます。これらのバッファサイズもコンパイル時に定義できます。

TXI および TEI 割り込みは調歩同期式モードでのみ使用されます。TXI 割り込みは、TDR レジスタの 1 バイト分のデータが TSR レジスタにシフトされたときに発生します。この割り込みで、送信リングバッファ内の次の 1 バイト分のデータが TDR レジスタにセットされます。R_SCI_Open()関数でコールバック関数がユーザによって指定されていれば、TEI 割り込みによって、その関数が呼び出されます。また、本モジュールでは、コンパイル時の設定で TEI 割り込み処理をコードから省くことも可能です。

RXI 割り込みは、RDR レジスタに 1 バイト分のデータが転送されたときに発生します。調歩同期式モードでは、RXI 割り込み処理で受信したデータを受信リングバッファにセットします。その後、R_SCI_Receive()関数が呼び出されると、受信リングバッファにセットされているデータにアクセスします。コールバック関数が指定されている場合、受信イベント（1 バイト受信）をトリガに、指定された関数が呼び出されます。受信キューがフルの場合、最後に受け取ったデータを未保存のまま、コールバック関数を呼び出します。SSPI およびクロック同期式モードでは、R_SCI_Receive()または R_SCI_SendReceive()関数で指定された受信用のバッファにデータが格納されます。R_SCI_Receive()または R_SCI_SendReceive()関数が呼び出される前に受信したデータは無視されます。SSPI およびクロック同期式モードでは、RXI 割り込み処理内でデータの送受信が行われます。送受信されるデータの残数は R_SCI_Open()の第 4 引数にセットされるハンドル内の送信用カウンタ(tx_cnt)、受信用カウンタ(rx_cnt)の値で確認することができます。詳細は「2.9 引数」を参照ください。

エラー検出について

受信時にフレーミングエラー、オーバランエラー、パリティエラーのいずれかのエラーを検出すると、ERI 割り込み要求が発生します。コールバック関数が指定されていれば、割り込み処理でエラーのタイプを判定し、イベントを通知します。詳細は「2.11 コールバック関数」を参照ください。

コールバック関数の有無にかかわらず、FIT モジュールが ERI 割り込み処理にてエラーフラグをクリアします。FIFO 機能を使用している場合は、エラーフラグをクリアする前にコールバック関数が呼び出されます。そのため、受信数分の FRDR レジスタを読み出すことで、どのデータを受信したときにエラーが発生したかを判定することができます。詳細は「2.11 コールバック関数」を参照ください。

1.3 API の概要

表 1.2　API 関数一覧に本モジュールに含まれる API 関数を示します。

表 1.2　API 関数一覧

関数	関数説明
R_SCI_Open()	SCI 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 関数に提供するチャネルのハンドルを設定します。受信エラーが発生した場合、あるいは他の割り込みイベントが発生した場合に呼び出すコールバック関数を設定します。
R_SCI_Close()	SCI 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_SCI_Send()	送信中でなければ、送信処理を行います。
R_SCI_Receive()	調歩同期の場合、RXI 割り込みによってキューに配置されたデータを取得します。 クロック同期および SSPI の場合、送信中でない、かつ受信中でなければ、ダミーデータの送信、および受信処理を開始します。
R_SCI_SendReceive()	クロック同期式および SSPI モードのみで使用します。送信中でない、かつ受信中でなければ、データの送信と受信を同時に行います。
R_SCI_Control()	対象の SCI チャネルに対し、特殊なハードウェアおよびソフトウェア動作を行います。
R_SCI_GetVersion()	本モジュールのバージョン番号を返します。

1.4 制限事項

なし

2. API 情報

本 FIT モジュールは、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。

- SCI
- GPIO

2.2 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) v5.20 以上
- r_byteq（調歩同期式モードのみ）

2.3 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。

2.4 使用する割り込みベクタ

調歩同期モードの場合、R_SCI_Open 関数を実行すると、RXIn 割り込み、ERIn 割り込みが有効になります。R_SCI_Send 関数を実行すると TXIn 割り込みが有効になります。

クロック同期モードおよび SSPI モードの場合、R_SCI_Open 関数を実行すると、RXIn 割り込み、ERIn 割り込みが有効になります。TXIn 割り込み、TEIn 割り込みは使用しません。

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
RX110、RX111 RX113、RX130 RX230 RX231、RX23T RX24T、RX24U (注 1)	ERI2 割り込み（ベクタ番号:186） RXI2 割り込み（ベクタ番号:187） TXI2 割り込み（ベクタ番号:188） TEI2 割り込み（ベクタ番号:189） ERI3 割り込み（ベクタ番号:190） RXI3 割り込み（ベクタ番号:191） TXI3 割り込み（ベクタ番号:192） TEI3 割り込み（ベクタ番号:193） ERI4 割り込み（ベクタ番号:194） RXI4 割り込み（ベクタ番号:195） TXI4 割り込み（ベクタ番号:196） TEI4 割り込み（ベクタ番号:197） ERI7 割り込み（ベクタ番号:206） RXI7 割り込み（ベクタ番号:207） TXI7 割り込み（ベクタ番号:208） TEI7 割り込み（ベクタ番号:209） ERI10 割り込み（ベクタ番号:210） RXI10 割り込み（ベクタ番号:211） TXI10 割り込み（ベクタ番号:212） TEI10 割り込み（ベクタ番号:213） ERI0 割り込み（ベクタ番号:214） RXI0 割り込み（ベクタ番号:215） TXI0 割り込み（ベクタ番号:216） TEI0 割り込み（ベクタ番号:217） ERI1 割り込み（ベクタ番号:218） RXI1 割り込み（ベクタ番号:219） TXI1 割り込み（ベクタ番号:220） TEI1 割り込み（ベクタ番号:221） ERI5 割り込み（ベクタ番号:222） RXI5 割り込み（ベクタ番号:223） TXI5 割り込み（ベクタ番号:224） TEI5 割り込み（ベクタ番号:225） ERI6 割り込み（ベクタ番号:226） RXI6 割り込み（ベクタ番号:227） TXI6 割り込み（ベクタ番号:228） TEI6 割り込み（ベクタ番号:229）

注1. MCU およびピン数によって、使用できる割り込みベクタが異なります。

デバイス	割り込みベクタ
RX110、RX111 RX113、RX130 RX230、RX231 RX23T、RX24T RX24U (注 1)	ERI8 割り込み（ベクタ番号:230） RXI8 割り込み（ベクタ番号:231） TXI8 割り込み（ベクタ番号:232） TEI8 割り込み（ベクタ番号:233） ERI9 割り込み（ベクタ番号:234） RXI9 割り込み（ベクタ番号:235） TXI9 割り込み（ベクタ番号:236） TEI9 割り込み（ベクタ番号:237） ERI12 割り込み（ベクタ番号:238） RXI12 割り込み（ベクタ番号:239） TXI12 割り込み（ベクタ番号:240） TEI12 割り込み（ベクタ番号:241） ERI11 割り込み（ベクタ番号 250） RXI11 割り込み（ベクタ番号:251） TXI11 割り込み（ベクタ番号:252） TEI11 割り込み（ベクタ番号:253）

注 1. MCU およびピン数によって、使用できる割り込みベクタが異なります。

デバイス	割り込みベクタ
RX64M、RX71M	RXI0 割り込み（ベクタ番号:58） TXI0 割り込み（ベクタ番号:59） RXI1 割り込み（ベクタ番号:60） TXI1 割り込み（ベクタ番号:61） RXI2 割り込み（ベクタ番号:62） TXI2 割り込み（ベクタ番号:63） RXI3 割り込み（ベクタ番号:80） TXI3 割り込み（ベクタ番号:81） RXI4 割り込み（ベクタ番号:82） TXI4 割り込み（ベクタ番号:83） RXI5 割り込み（ベクタ番号:84） TXI5 割り込み（ベクタ番号:85） RXI6 割り込み（ベクタ番号:86） TXI6 割り込み（ベクタ番号:87） RXI7 割り込み（ベクタ番号:98） TXI7 割り込み（ベクタ番号:99） RXI12 割り込み（ベクタ番号:116） TXI12 割り込み（ベクタ番号:117）

デバイス	割り込みベクタ
RX64M, RX71M	GROUPBL0 割り込み（ベクタ番号: 110） ● TEI0 割り込み（グループ割り込み要因番号：0） ● ERI0 割り込み（グループ割り込み要因番号：1） ● TEI1 割り込み（グループ割り込み要因番号：2） ● ERI1 割り込み（グループ割り込み要因番号：3） ● TEI2 割り込み（グループ割り込み要因番号：4） ● ERI2 割り込み（グループ割り込み要因番号：5） ● TEI3 割り込み（グループ割り込み要因番号：6） ● ERI3 割り込み（グループ割り込み要因番号：7） ● TEI4 割り込み（グループ割り込み要因番号：8） ● ERI4 割り込み（グループ割り込み要因番号：9） ● TEI5 割り込み（グループ割り込み要因番号：10） ● ERI5 割り込み（グループ割り込み要因番号：11） ● TEI6 割り込み（グループ割り込み要因番号：12） ● ERI6 割り込み（グループ割り込み要因番号：13） ● TEI7 割り込み（グループ割り込み要因番号：14） ● ERI7 割り込み（グループ割り込み要因番号：15） ● TEI12 割り込み（グループ割り込み要因番号：16） ● ERI12 割り込み（グループ割り込み要因番号：17）
RX65N	RXI0 割り込み（ベクタ番号:58） TXI0 割り込み（ベクタ番号:59） RXI1 割り込み（ベクタ番号:60） TXI1 割り込み（ベクタ番号:61） RXI2 割り込み（ベクタ番号:62） TXI2 割り込み（ベクタ番号:63） RXI3 割り込み（ベクタ番号:80） TXI3 割り込み（ベクタ番号:81） RXI4 割り込み（ベクタ番号:82） TXI4 割り込み（ベクタ番号:83） RXI5 割り込み（ベクタ番号:84） TXI5 割り込み（ベクタ番号:85） RXI6 割り込み（ベクタ番号:86） TXI6 割り込み（ベクタ番号:87） RXI7 割り込み（ベクタ番号:98） TXI7 割り込み（ベクタ番号:99） RXI8 割り込み（ベクタ番号:100） TXI8 割り込み（ベクタ番号:101） RXI9 割り込み（ベクタ番号:102） TXI9 割り込み（ベクタ番号:103） RXI10 割り込み（ベクタ番号:104） TXI10 割り込み（ベクタ番号:105） RXI11 割り込み（ベクタ番号:114） TXI11 割り込み（ベクタ番号:115） RXI12 割り込み（ベクタ番号:116） TXI12 割り込み（ベクタ番号:117）

デバイス	割り込みベクタ
RX65N	GROUPBL0 割り込み（ベクタ番号: 110） ● TEI0 割り込み（グループ割り込み要因番号：0） ● ERI0 割り込み（グループ割り込み要因番号：1） ● TEI1 割り込み（グループ割り込み要因番号：2） ● ERI1 割り込み（グループ割り込み要因番号：3） ● TEI2 割り込み（グループ割り込み要因番号：4） ● ERI2 割り込み（グループ割り込み要因番号：5） ● TEI3 割り込み（グループ割り込み要因番号：6） ● ERI3 割り込み（グループ割り込み要因番号：7） ● TEI4 割り込み（グループ割り込み要因番号：8） ● ERI4 割り込み（グループ割り込み要因番号：9） ● TEI5 割り込み（グループ割り込み要因番号：10） ● ERI5 割り込み（グループ割り込み要因番号：11） ● TEI6 割り込み（グループ割り込み要因番号：12） ● ERI6 割り込み（グループ割り込み要因番号：13） ● TEI7 割り込み（グループ割り込み要因番号：14） ● ERI7 割り込み（グループ割り込み要因番号：15） ● TEI12 割り込み（グループ割り込み要因番号：16） ● ERI12 割り込み（グループ割り込み要因番号：17） GROUPBL1 割り込み（ベクタ番号: 111） ● TEI8 割り込み（グループ割り込み要因番号：24） ● ERI8 割り込み（グループ割り込み要因番号：25） ● TEI9 割り込み（グループ割り込み要因番号：26） ● ERI9 割り込み（グループ割り込み要因番号：27） GROUPAL0 割り込み（ベクタ番号: 112） ● TEI10 割り込み（グループ割り込み要因番号：8） ● ERI10 割り込み（グループ割り込み要因番号：9） ● TEI11 割り込み（グループ割り込み要因番号：12） ● ERI11 割り込み（グループ割り込み要因番号：13）

デバイス	割り込みベクタ
RX66T、RX72T	RXI1 割り込み（ベクタ番号:60) TXI1 割り込み（ベクタ番号:61) RXI5 割り込み（ベクタ番号:84) TXI5 割り込み（ベクタ番号:85) RXI6 割り込み（ベクタ番号:86) TXI6 割り込み（ベクタ番号:87) RXI8 割り込み（ベクタ番号:100) TXI8 割り込み（ベクタ番号:101) RXI9 割り込み（ベクタ番号:102) TXI9 割り込み（ベクタ番号:103) RXI11 割り込み（ベクタ番号:114) TXI11 割り込み（ベクタ番号:115) RXI12 割り込み（ベクタ番号:116) TXI12 割り込み（ベクタ番号:117) GROUPBL0 割り込み（ベクタ番号:110) ● TEI1 割り込み（グループ割り込み要因番号：2) ● ERI1 割り込み（グループ割り込み要因番号：3) ● TEI5 割り込み（グループ割り込み要因番号：10) ● ERI5 割り込み（グループ割り込み要因番号：11) ● TEI6 割り込み（グループ割り込み要因番号：12) ● ERI6 割り込み（グループ割り込み要因番号：13) ● TEI12 割り込み（グループ割り込み要因番号：16) ● ERI12 割り込み（グループ割り込み要因番号：17) GROUPBL1 割り込み（ベクタ番号:111) ● TEI8 割り込み（グループ割り込み要因番号：24) ● ERI8 割り込み（グループ割り込み要因番号：25) ● TEI9 割り込み（グループ割り込み要因番号：26) ● ERI9 割り込み（グループ割り込み要因番号：27) GROUPAL0 割り込み（ベクタ番号:112) ● TEI11 割り込み（グループ割り込み要因番号：12) ● ERI11 割り込み（グループ割り込み要因番号：13)

2.5 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_sci_rx_if.h に記載しています。

2.6 整数型

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

2.7 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_sci_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。

コンフィギュレーションオプション（r_sci_rx_config.h）	
SCI_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の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の処理をコードから省略します。 このオプションに BSP_CFG_PARAM_CHECKING_ENABLE を設定すると、システムのデフォルト設定が使用されます。
SCI_CFG_ASYNC_INCLUDED ※デフォルト値は“1” SCI_CFG_SYNC_INCLUDED ※デフォルト値は“0” SCI_CFG_SSPI_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。
SCI_CFG_DUMMY_TX_BYTE ※デフォルト値は“0xFF”	このオプションは SSPI およびクロック同期式モードでのみ使用します。R_SCI_Receive()関数の呼び出しで、各バイトデータの受信に対して送信されるダミーデータの値です。
SCI_CFG_CH0_INCLUDED SCI_CFG_CH1_INCLUDED SCI_CFG_CH2_INCLUDED SCI_CFG_CH3_INCLUDED SCI_CFG_CH4_INCLUDED SCI_CFG_CH5_INCLUDED SCI_CFG_CH6_INCLUDED SCI_CFG_CH7_INCLUDED SCI_CFG_CH8_INCLUDED SCI_CFG_CH9_INCLUDED SCI_CFG_CH10_INCLUDED SCI_CFG_CH11_INCLUDED SCI_CFG_CH12_INCLUDED ※各デフォルト値は以下のとおり: CH0、CH2～CH12: 0、CH1: 1	チャネルごとに送受信バッファ、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。このオプションを“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。 デフォルトでは CH1 のみが有効に設定されています。config ファイルにて、使用するチャネルを確認してください。
SCI_CFG_CH0_TX_BUFSIZ SCI_CFG_CH1_TX_BUFSIZ SCI_CFG_CH2_TX_BUFSIZ SCI_CFG_CH3_TX_BUFSIZ SCI_CFG_CH4_TX_BUFSIZ SCI_CFG_CH5_TX_BUFSIZ SCI_CFG_CH6_TX_BUFSIZ SCI_CFG_CH7_TX_BUFSIZ SCI_CFG_CH8_TX_BUFSIZ SCI_CFG_CH9_TX_BUFSIZ SCI_CFG_CH10_TX_BUFSIZ SCI_CFG_CH11_TX_BUFSIZ SCI_CFG_CH12_TX_BUFSIZ ※デフォルト値はすべて“80”	調歩同期式モードで、各チャネルの送信キューに使用されるバッファサイズを指定します。 使用するチャネルに対応する “SCI_CFG_CHn_INCLUDED”、または “SCI_CFG_ASYNC_INCLUDED”が“0”に設定されている場合は、バッファは割り当てられません。

#define SCI_CFG_CH0_RX_BUFSIZ #define SCI_CFG_CH1_RX_BUFSIZ #define SCI_CFG_CH2_RX_BUFSIZ #define SCI_CFG_CH3_RX_BUFSIZ #define SCI_CFG_CH4_RX_BUFSIZ #define SCI_CFG_CH5_RX_BUFSIZ #define SCI_CFG_CH6_RX_BUFSIZ #define SCI_CFG_CH7_RX_BUFSIZ #define SCI_CFG_CH8_RX_BUFSIZ #define SCI_CFG_CH9_RX_BUFSIZ #define SCI_CFG_CH10_RX_BUFSIZ #define SCI_CFG_CH11_RX_BUFSIZ #define SCI_CFG_CH12_RX_BUFSIZ ※デフォルト値はすべて“80”	調歩同期式モードで、各チャネルの受信キューに使用されるバッファサイズを指定します。 使用するチャネルに対応する“SCI_CFG_CHn_INCLUDED”か“SCI_CFG_ASYNC_INCLUDED”が“0”に設定されている場合は、バッファは割り当てられません。
SCI_CFG_TEI_INCLUDED ※デフォルト値は“0”	このオプションを“1”に設定すると、送信データエンプティ割り込みの処理をコードに含めます。TEI 割り込みは、データの最終バイトの最終ビットが出力されたときに発生します。この割り込みで、ユーザ設定のコールバック関数（R_SCI_Open()で設定）が呼び出されます。
SCI_CFG_RXERR_PRIORITY ※デフォルト値は“3”	このオプションは RX63N、RX631 のみに適用されます。グループ 12 エラー割り込みの優先レベルを設定します。優先レベルは最低値が 1、最高値が 15 です。この割り込みで全チャネルのオーバランエラー、フレーミングエラー、パリティエラーを処理します。
SCI_CFG_ERI_TEI_PRIORITY ※デフォルト値は“3”	このオプションは RX64M、RX71M、RX65N のみに適用されます。エラー割り込み（ERI）と送信終了割り込み（TEI）の優先レベルを設定します。優先レベルは最低値が 1、最高値が 15 です。ERI 割り込みで全チャネルのオーバランエラー、フレーミングエラー、パリティエラーを処理します。TEI 割り込みで、最終ビットが送信され、送信完了状態になったことを示します（調歩同期式モード）。
SCI_CFG_CH10_FIFO_INCLUDED SCI_CFG_CH11_FIFO_INCLUDED ※デフォルト値は“0”	このオプションは FIFO 機能を搭載する SCI モジュール(SCIi)をサポートする MCU の場合のみ適用される定義となります。 ● 1：ビルド時に FIFO 機能に関する処理をコードに含めます。 ● 0：ビルド時に FIFO 機能に関する処理をコードから除外します。
SCI_CFG_CH10_TX_FIFO_THRESH SCI_CFG_CH11_TX_FIFO_THRESH ※デフォルト値は“8”	このオプションは FIFO 機能を搭載する SCI モジュール(SCIi)をサポートする MCU の場合のみ適用される定義となります。 SCI の動作モードがクロック同期式モード、簡易 SPI モードの場合は受信 FIFO のしきい値の設定と同じ値を設定してください。 ● 0~15：送信 FIFO のしきい値を設定します。
SCI_CFG_CH10_RX_FIFO_THRESH SCI_CFG_CH11_RX_FIFO_THRESH ※デフォルト値は“8”	このオプションは FIFO 機能を搭載する SCI モジュール(SCIi)をサポートする MCU の場合のみ適用される定義となります。 ● 1~15：受信 FIFO のしきい値を設定します。
SCI_CFG_CH1_DATA_MATCH_INCLUDED SCI_CFG_CH5_DATA_MATCH_INCLUDED SCI_CFG_CH6_DATA_MATCH_INCLUDED SCI_CFG_CH8_DATA_MATCH_INCLUDED SCI_CFG_CH9_DATA_MATCH_INCLUDED SCI_CFG_CH11_DATA_MATCH_INCLUDED ※デフォルト値は“0”	RX66T と RX72T のみ。データ比較関数を記述した SCI モジュール (SCIi、SCIj) があります。 ● 1: データ比較関数に関連する処理はビルド内に包含されます ● 0: データ比較関数に関連する処理はビルドから除外されます

2.8 コードサイズ

本モジュールのコード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「2.7 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。掲載した値は、「2.3 サポートされているツールチェーン」の C コンパイラでコンパイルオプションがデフォルト時の参考値です。コンパイルオプションのデフォルトは最適化レベル：2、最適化のタイプ：サイズ優先、データ・エンディアン：リトルエンディアンです。コードサイズは C コンパイラのバージョンやコンパイルオプションにより異なります。

ROM、RAM およびスタックのコードサイズ (1/3)

デバイス	分類		使用メモリ		備考
			ルネサス製コンパイラ		
			パラメータチェック処理あり	パラメータチェック処理なし	
RX130	調歩同期	ROM	2844 バイト	2502 バイト	1 チャネルを使用
		RAM	192 バイト	192 バイト	1 チャネル使用
	クロック同期	ROM	2569 バイト	2174 バイト	1 チャネル使用
		RAM	36 バイト	36 バイト	1 チャネル使用
	調歩同期 + クロック同期（または簡易 SPI）	ROM	3872 バイト	3386 バイト	計 2 チャネル使用
		RAM	392 バイト	392 バイト	計 2 チャネル使用
	最大使用スタックサイズ		164 バイト		
RX231	調歩同期	ROM	2763 バイト	2420 バイト	1 チャネル使用
		RAM	192 バイト	192 バイト	1 チャネル使用
	クロック同期	ROM	2488 バイト	2093 バイト	1 チャネル使用
		RAM	36 バイト	36 バイト	1 チャネル使用
	調歩同期 + クロック同期（または簡易 SPI）	ROM	3791 バイト	3376 バイト	計 2 チャネル使用
		RAM	392 バイト	392 バイト	計 2 チャネル使用
	最大使用スタックサイズ		132 バイト		

ROM、RAM およびスタックのコードサイズ(bytes) (2/3)					
デバイス	分類		使用メモリ		備考
			ルネサス製コンパイラ		
			パラメータチェック処理あり	パラメータチェック処理なし	
RX64M	調歩同期	ROM	2866 バイト	2505 バイト	1 チャネル使用
		RAM	192 バイト	192 バイト	1 チャネル使用
	クロック同期	ROM	2599 バイト	2186 バイト	1 チャネル使用
		RAM	36 バイト	36 バイト	1 チャネル使用
	調歩同期 + クロック同期（または簡易 SPI）	ROM	3900 バイト	3395 バイト	計 2 チャネル使用
		RAM	392 バイト	392 バイト	計 2 チャネル使用
	最大使用スタックサイズ		144 バイト		
RX65N	調歩同期	ROM	2854 バイト	2493 バイト	1 チャネル使用
		RAM	192 バイト	192 バイト	1 チャネル使用
	クロック同期	ROM	2587 バイト	2174 バイト	1 チャネル使用
		RAM	36 バイト	36 バイト	1 チャネル使用
	調歩同期 +クロック同期（または簡易 SPI）	ROM	3888 バイト	3383 バイト	計 2 チャネル使用
		RAM	392 バイト	392 バイト	計 2 チャネル使用
	最大使用スタックサイズ		144 バイト		
	FIFO モード + 調歩同期	ROM	3738 バイト	3327 バイト	1 チャネル使用
		RAM	200 バイト	200 バイト	1 チャネル使用
	FIFO モード + クロック同期	ROM	3715 バイト	3224 バイト	1 チャネル使用
		RAM	44 バイト	44 バイト	1 チャネル使用
	FIFO モード + 調歩同期 + クロック同期（または簡易 SPI）	ROM	5277 バイト	4698 バイト	計 2 チャネル使用
		RAM	408 バイト	408 バイト	計 2 チャネル使用
	最大使用スタックサイズ		184 バイト		

ROM、RAM およびスタックのコードサイズ(bytes) (3/3)					
デバイス	分類		使用メモリ		備考
			ルネサス製コンパイラ		
			パラメータチェック処理あり	パラメータチェック処理なし	
RX66T	調歩同期モード	ROM	2852 バイト	2488 バイト	1 チャネルを使用
		RAM	192 バイト	192 バイト	1 チャネルを使用
	クロック同期モード	ROM	2586 バイト	2173 バイト	1 チャネルを使用
		RAM	36 バイト	36 バイト	1 チャネルを使用
	調歩同期モード + クロック同期モード (またはシンプルな SPI)	ROM	3897 バイト	3389 バイト	合計 2 チャネルを使用
		RAM	392 バイト	392 バイト	合計 2 チャネルを使用
	最大のスタック使用量		140 バイト		
	FIFO モード +調歩同期モード	ROM	3732 バイト	3318 バイト	1 チャネルを使用
		RAM	200 バイト	200 バイト	1 チャネルを使用
	FIFO モード + クロック同期モード	ROM	3712 バイト	3221 バイト	1 チャネルを使用
		RAM	44 バイト	44 バイト	1 チャネルを使用
	FIFO モード +調歩同期モード + クロック同期モード (またはシンプルな SPI)	ROM	5281 バイト	4699 バイト	合計 2 チャネルを使用
		RAM	408 バイト	408 バイト	合計 2 チャネルを使用
	最大のスタック使用量		188 バイト		

ROM、RAM およびスタックのコードサイズ(bytes) (3/3)					
デバイス	分類		使用メモリ		備考
			ルネサス製コンパイラ		
			パラメータチェック処理あり	パラメータチェック処理なし	
RX72T	調歩同期モード	ROM	2845 バイト	2481 バイト	1 チャネルを使用
		RAM	192 バイト	192 バイト	1 チャネルを使用
	クロック同期モード	ROM	2579 バイト	2166 バイト	1 チャネルを使用
		RAM	36 バイト	36 バイト	1 チャネルを使用
	調歩同期モード + クロック同期モード (またはシンプルな SPI)	ROM	3890 バイト	3382 バイト	合計 2 チャネルを使用
		RAM	392 バイト	392 バイト	合計 2 チャネルを使用
	最大のスタック使用量		140 バイト		
	FIFO モード +調歩同期モード	ROM	3748 バイト	3338 バイト	1 チャネルを使用
		RAM	200 バイト	200 バイト	1 チャネルを使用
	FIFO モード + クロック同期モード	ROM	3705 バイト	3214 バイト	1 チャネルを使用
		RAM	44 バイト	44 バイト	1 チャネルを使用
	FIFO モード +調歩同期モード + クロック同期モード (またはシンプルな SPI)	ROM	5309 バイト	4726 バイト	合計 2 チャネルを使用
		RAM	408 バイト	408 バイト	合計 2 チャネルを使用
	最大のスタック使用量		188 バイト		

ROM と RAM の最小サイズ (バイト) (1/3)					
デバイス	カテゴリ		メモリ使用状況		備考
			GCC		
			パラメータチェック処理あり	パラメータチェック処理なし	
RX130	非同期モード	ROM	6960 バイト	6400 バイト	1 チャネルを使用
		RAM	160 バイト	160 バイト	1 チャネルを使用
	クロック同期モード	ROM	6612 バイト	5988 バイト	1 チャネルを使用
		RAM	0 バイト	0 バイト	1 チャネルを使用
	非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	8836 バイト	8020 バイト	合計 2 チャネルを使用
		RAM	320 バイト	320 バイト	合計 2 チャネルを使用
	最大のスタック使用量				
RX231	非同期モード	ROM	4856 バイト	4288 バイト	1 チャネルを使用
		RAM	160 バイト	160 バイト	1 チャネルを使用
	クロック同期モード	ROM	4492 バイト	3884 バイト	1 チャネルを使用
		RAM	0 バイト	0 バイト	1 チャネルを使用
	非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	6740 バイト	5924 バイト	合計 2 チャネルを使用
		RAM	320 バイト	320 バイト	合計 2 チャネルを使用
	最大のスタック使用量				

ROM と RAM の最小サイズ (バイト) (2/3)					
デバイス	通信方式		メモリ使用状況		備考
			GCC		
			パラメータチェック処理あり	パラメータチェック処理なし	
RX64M	非同期モード	ROM	5048 バイト	4432 バイト	1 チャネルを使用
		RAM	160 バイト	160 バイト	1 チャネルを使用
	クロック同期モード	ROM	4708 バイト	4044 バイト	1 チャネルを使用
		RAM	0 バイト	0 バイト	1 チャネルを使用
	非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	6964 バイト	6100 バイト	合計 2 チャネルを使用
		RAM	320 バイト	320 バイト	合計 2 チャネルを使用
	最大のスタック使用量		-		
RX65N	非同期モード	ROM	5056 バイト	4424 バイト	1 チャネルを使用
		RAM	160 バイト	160 バイト	1 チャネルを使用
	クロック同期モード	ROM	4700 バイト	4036 バイト	1 チャネルを使用
		RAM	0 バイト	0 バイト	1 チャネルを使用
	非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	6964 バイト	6092 バイト	合計 2 チャネルを使用
		RAM	320 バイト	320 バイト	合計 2 チャネルを使用
	最大のスタック使用量		-		
	FIFO モード + 非同期モード	ROM	6824 バイト	6112 バイト	1 チャネルを使用
		RAM	160 バイト	160 バイト	1 チャネルを使用
	FIFO モード + クロック同期モード	ROM	6980 バイト	6164 バイト	1 チャネルを使用
		RAM	0 バイト	0 バイト	1 チャネルを使用
	FIFO モード + 非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	9732 バイト	8740 バイト	合計 2 チャネルを使用
		RAM	320 バイト	320 バイト	合計 2 チャネルを使用
	最大のスタック使用量		-		

ROM と RAM の最小サイズ (バイト) (3/3)					
デバイス	通信方式		メモリ使用状況		備考
			GCC		
			パラメータチェック処理あり	パラメータチェック処理なし	
RX66T	非同期モード	ROM	5056 バイト	4424 バイト	1 チャネルを使用
		RAM	160 バイト	160 バイト	1 チャネルを使用
	クロック同期モード	ROM	4700 バイト	4036 バイト	1 チャネルを使用
		RAM	0 バイト	0 バイト	1 チャネルを使用
	非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	6964 バイト	6092 バイト	合計 2 チャネルを使用
		RAM	320 バイト	320 バイト	合計 2 チャネルを使用
	最大のスタック使用量		-		
	FIFO モード + 非同期モード	ROM	6824 バイト	6112 バイト	1 チャネルを使用
		RAM	160 バイト	160 バイト	1 チャネルを使用
	FIFO モード + クロック同期モード	ROM	6980 バイト	6164 バイト	1 チャネルを使用
		RAM	0 バイト	0 バイト	1 チャネルを使用
	FIFO モード + 非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	9572 バイト	8580 バイト	合計 2 チャネルを使用
		RAM	320 バイト	320 バイト	合計 2 チャネルを使用
	最大のスタック使用量		-		
RX72T	非同期モード	ROM	5056 バイト	4424 バイト	1 チャネルを使用
		RAM	160 バイト	160 バイト	1 チャネルを使用
	クロック同期モード	ROM	4700 バイト	4036 バイト	1 チャネルを使用
		RAM	0 バイト	0 バイト	1 チャネルを使用
	非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	6964 バイト	6092 バイト	合計 2 チャネルを使用
		RAM	320 バイト	320 バイト	合計 2 チャネルを使用
	最大のスタック使用量		-		
	FIFO モード + 非同期モード	ROM	6824 バイト	6112 バイト	1 チャネルを使用
		RAM	160 バイト	160 バイト	1 チャネルを使用
	FIFO モード + クロック同期モード	ROM	6996 バイト	6164 バイト	1 チャネルを使用
		RAM	0 バイト	0 バイト	1 チャネルを使用
	FIFO モード + 非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	9732 バイト	8740 バイト	合計 2 チャネルを使用
		RAM	320 バイト	320 バイト	合計 2 チャネルを使用
	最大のスタック使用量				

ROM と RAM の最小サイズ (バイト) (1/3)					
デバイス	カテゴリ		メモリ使用状況		備考
			IAR コンパイラ		
			パラメータチェック処理あり	パラメータチェック処理なし	
RX130	非同期モード	ROM	4431 バイト	3847 バイト	1 チャネルを使用
		RAM	576 バイト	576 バイト	1 チャネルを使用
	クロック同期モード	ROM	3791 バイト	3207 バイト	1 チャネルを使用
		RAM	36 バイト	36 バイト	1 チャネルを使用
	非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	5797 バイト	4989 バイト	合計 2 チャネルを使用
		RAM	776 バイト	776 バイト	合計 2 チャネルを使用
	最大のスタック使用量		180 バイト		
RX231	非同期モード	ROM	4428 バイト	3842 バイト	1 チャネルを使用
		RAM	576 バイト	576 バイト	1 チャネルを使用
	クロック同期モード	ROM	3786 バイト	3202 バイト	1 チャネルを使用
		RAM	36 バイト	36 バイト	1 チャネルを使用
	非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	5788 バイト	4978 バイト	合計 2 チャネルを使用
		RAM	776 バイト	776 バイト	合計 2 チャネルを使用
	最大のスタック使用量		180 バイト		

ROM と RAM の最小サイズ (バイト) (2/3)					
デバイス	通信方式		メモリ使用状況		備考
			IAR コンパイラ		
			パラメータチェック処理あり	パラメータチェック処理なし	
RX64M	非同期モード	ROM	4566 バイト	3962 バイト	1 チャネルを使用
		RAM	577 バイト	577 バイト	1 チャネルを使用
	クロック同期モード	ROM	3935 バイト	3333 バイト	1 チャネルを使用
		RAM	36 バイト	36 バイト	1 チャネルを使用
	非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	5940 バイト	5112 バイト	合計 2 チャネルを使用
		RAM	777 バイト	777 バイト	合計 2 チャネルを使用
	最大のスタック使用量		204 バイト		
RX65N	非同期モード	ROM	4565 バイト	3962 バイト	1 チャネルを使用
		RAM	577 バイト	577 バイト	1 チャネルを使用
	クロック同期モード	ROM	3924 バイト	3329 バイト	1 チャネルを使用
		RAM	36 バイト	36 バイト	1 チャネルを使用
	非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	5935 バイト	5108 バイト	合計 2 チャネルを使用
		RAM	777 バイト	777 バイト	合計 2 チャネルを使用
	最大のスタック使用量		204 バイト		
	FIFO モード + 非同期モード	ROM	5872 バイト	5172 バイト	1 チャネルを使用
		RAM	585 バイト	585 バイト	1 チャネルを使用
	FIFO モード + クロック同期モード	ROM	5577 バイト	4875 バイト	1 チャネルを使用
		RAM	44 バイト	44 バイト	1 チャネルを使用
	FIFO モード + 非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	7960 バイト	7026 バイト	合計 2 チャネルを使用
		RAM	793 バイト	793 バイト	合計 2 チャネルを使用
	最大のスタック使用量		240 バイト		

ROM と RAM の最小サイズ (バイト) (3/3)					
デバイス	通信方式		メモリ使用状況		備考
			IAR コンパイラ		
			パラメータチェック処理あり	パラメータチェック処理なし	
RX66T	非同期モード	ROM	4562 バイト	3961 バイト	1 チャネルを使用
		RAM	577 バイト	577 バイト	1 チャネルを使用
	クロック同期モード	ROM	3925 バイト	3332 バイト	1 チャネルを使用
		RAM	36 バイト	36 バイト	1 チャネルを使用
	非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	5815 バイト	4990 バイト	合計 2 チャネルを使用
		RAM	741 バイト	741 バイト	合計 2 チャネルを使用
	最大のスタック使用量		204 バイト		
	FIFO モード + 非同期モード	ROM	5869 バイト	5171 バイト	1 チャネルを使用
		RAM	585 バイト	585 バイト	1 チャネルを使用
	FIFO モード + クロック同期モード	ROM	5578 バイト	4878 バイト	1 チャネルを使用
		RAM	44 バイト	44 バイト	1 チャネルを使用
	FIFO モード + 非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	7837 バイト	6905 バイト	合計 2 チャネルを使用
		RAM	749 バイト	749 バイト	合計 2 チャネルを使用
	最大のスタック使用量		240 バイト		
RX72T	非同期モード	ROM	4567 バイト	3962 バイト	1 チャネルを使用
		RAM	577 バイト	577 バイト	1 チャネルを使用
	クロック同期モード	ROM	3926 バイト	3329 バイト	1 チャネルを使用
		RAM	36 バイト	36 バイト	1 チャネルを使用
	非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	5940 バイト	5111 バイト	合計 2 チャネルを使用
		RAM	777 バイト	777 バイト	合計 2 チャネルを使用
	最大のスタック使用量		204 バイト		
	FIFO モード + 非同期モード	ROM	5893 バイト	5191 バイト	1 チャネルを使用
		RAM	585 バイト	585 バイト	1 チャネルを使用
	FIFO モード + クロック同期モード	ROM	5579 バイト	4875 バイト	1 チャネルを使用
		RAM	44 バイト	44 バイト	1 チャネルを使用
	FIFO モード + 非同期モード + クロック同期モード (またはシンプルな SPI)	ROM	7965 バイト	7029 バイト	合計 2 チャネルを使用
		RAM	793 バイト	793 バイト	合計 2 チャネルを使用
	最大のスタック使用量		240 バイト		

RAM の要求サイズは設定されるチャネル数によって変わります。RAM には各チャネルのデータ構造体が格納されています。また、調歩同期式モードでは、チャネルごとに送信キューと受信キューが配置されます。バッファには、送信／受信キュー用に最低で 2 バイトが割り当てられますので、チャネルごとに最低 4 バイトが割り当てられることになります。キューバッファのサイズはユーザによる設定が可能なので、バッファに割り当てられるサイズによっては、RAM に要求されるサイズが増減します。

以下に調歩同期式モードで必要となる RAM サイズの計算方法を示します。

使用するチャネル数（1～12）×（チャネルごとのデータ構造体（32 バイト）

＋ 送信キューのバッファサイズ（SCI_CFG_CHn_TX_BUFSIZ によって指定されたサイズ）

＋ 受信キューのバッファサイズ（SCI_CFG_CHn_RX_BUFSIZ によって指定されたサイズ））

※FIFO モードの場合、チャネルごとのデータ構造体は 36 バイトとなります。

クロック同期式および SPI モードを使用する場合の RAM の要求サイズは、使用するチャネル数 × チャネルごとのデータ構造体（36 バイト、FIFO モードの場合は 40 バイト固定）となります。

ROM の要求サイズも設定されるチャネル数によって変わります。正確なサイズは、選択されたチャネルの組み合わせとコンパイラのコード最適化の状態によって異なります。

2.9　引数

API 関数の引数である構造体を示します。この構造体は、API 関数のプロトタイプ宣言とともに r_sci_rx_if.h に記載されています。

チャネル管理用構造体

SCI の各チャネルを制御するために必要な管理情報を格納するための構造体です。
この構造体はコンフィグレーションオプションの設定、およびデバイスの種類によって、内容が異なります。ユーザはチャネル管理用構造体の中身を意識する必要はありませんが、クロック同期式モード/SSPI モードの場合、tx_cnt、rx_cnt を参照することにより、処理すべき残データの数を確認することが出来ます。

以下に、デバイスの種類が RX65N の場合のチャネル管理用構造体を示します。

```
typedef struct st_sci_ch_ctrl       // チャネル管理用構造体
{
sci_ch_rom_t const *rom;            // チャネルに対応する SCI のレジスタの先頭アドレス
sci_mode_t mode;                    // 現在チャネルにセットされている SCI 動作モード
uint32_t baud_rate;       // 現在チャネルにセットされているビットレート
void (*callback)(void *p_args);    // コールバック関数のアドレス
union
{
#if (SCI_CFG_ASYNC_INCLUDED)
byteq_hdl_t que;                    // 送信用バイトキュー(調歩同期式モード)
#endif
uint8_t *buf;      // 送信用バッファの先頭アドレス
//(クロック同期式/SSPI モード)
} u_tx_data;
union
{
#if (SCI_CFG_ASYNC_INCLUDED)
byteq_hdl_t que;                    // 受信用バイトキュー(調歩同期式モード)
#endif
uint8_t *buf;               // 受信用バッファの先頭アドレス
                                    //(クロック同期式/SSPI モード)
} u_rx_data;
bool tx_idle;               // 送信アイドル状態(アイドル状態/送信中)
#if (SCI_CFG_SSPI_INCLUDED || SCI_CFG_SYNC_INCLUDED)
bool save_rx_data;                  // 受信用データ保存(有効/無効)
uint16_t tx_cnt;            // 送信用カウンタ
uint16_t rx_cnt;            // 受信用カウンタ
bool tx_dummy;              // ダミーデータ送信(有効/無効)
```

```
#endif
uint32_t pclk_speed;     // 周辺モジュールクロックの動作周波数
#if SCI_CFG_FIFO_INCLUDED
uint8_t fifo_ctrl;           // FIFO 機能(有効/無効)
uint8_t rx_dflt_thresh;    // 受信 FIFO しきい値（デフォルト）
uint8_t rx_curr_thresh;   // 受信 FIFO しきい値（カレント）
uint8_t tx_dflt_thresh;    // 送信 FIFO しきい値（デフォルト）
uint8_t tx_curr_thresh;   // 送信 FIFO しきい値（カレント）
#endif
} sci_ch_ctrl_t;
```

2.10 戻り値

API 関数の戻り値を示します。この列挙型は、API 関数のプロトタイプ宣言とともに r_sci_rx_if.h で記載されています。

```
typedef enum e_sci_err     // SCI API エラーコード
{
   SCI_SUCCESS=0,
   SCI_ERR_BAD_CHAN,      // 存在しないチャネルの番号
   SCI_ERR_OMITTED_CHAN,  // config.h の SCI_CHx_INCLUDED の値が 0 です。
   SCI_ERR_CH_NOT_CLOSED, // チャネルは別のモードで使用されています。
   SCI_ERR_BAD_MODE,      // チャネルに対応していないモード、または不正なモードです。
   SCI_ERR_INVALID_ARG,   // パラメータに対して引数が無効です。
   SCI_ERR_NULL_PTR,      // null ptr 受信; 要求された引数がありません。
   SCI_ERR_XCVR_BUSY,     // データ転送を開始できません。ビジー状態です。

   // 調歩同期式モードのみ有効なエラーコード
   SCI_ERR_QUEUE_UNAVAILABLE, // 送信、受信キューのいずれか、または両方とも開けません。
   SCI_ERR_INSUFFICIENT_SPACE, // 送信キューに十分なスペースがありません。
   SCI_ERR_INSUFFICIENT_DATA, // 受信キューに十分なデータがありません。

   // Synchronous/SSPI modes only
   SCI_ERR_XFER_NOT_DONE  // データ転送は処理中です。
} sci_err_t;
```

2.11 コールバック関数

本モジュールでは、RXIn と ERIn 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、「2.9 引数」に記載された構造体メンバ“void (* const p_callback)(void *p_args)”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。コールバック関数が呼び出されるとき、表 2.3 に示す定数が格納された変数が、引数として渡されます。

引数の型は void ポインタ型で渡されるため、コールバック関数の引数は以下の例を参考に void 型のポインタ変数としてください。

コールバック関数内部で引数の値を使用する際はキャストして使用してください。

以下は、調歩同期式モードのコールバック関数のテンプレート例です。

```
void MyCallback(void *p_args)
{
sci_cb_args_t *args;
args = (sci_cb_args_t *)p_args;
if (args->event == SCI_EVT_RX_CHAR)
{
//from RXI interrupt; character placed in queue is in args->byte
nop();
}
else if (args->event == SCI_EVT_RX_CHAR_MATCH)
{
//from RXI interrupt, received data match comparison data (RXI 割り込みから受信したデータが比較対象データと一致)
//character placed in queue is in args->byte (キュー内に配置される文字は args->byte の順序)
nop();
}

#if SCI_CFG_TEI_INCLUDED
```

```
else if (args->event == SCI_EVT_TEI)
{
// from TEI interrupt; transmitter is idle
// possibly disable external transceiver here
nop();
}
#endif
else if (args->event == SCI_EVT_RXBUF_OVFL)
{
// from RXI interrupt; receive queue is full
// unsaved char is in args->byte
// will need to increase buffer size or reduce baud rate
nop();
}
else if (args->event == SCI_EVT_OVFL_ERR)
{
// from ERI/Group12 interrupt; receiver overflow error occurred
// error char is in args->byte
// error condition is cleared in ERI routine
nop();
}
else if (args->event == SCI_EVT_FRAMING_ERR)
{
// from ERI/Group12 interrupt; receiver framing error occurred
// error char is in args->byte; if = 0, received BREAK condition
// error condition is cleared in ERI routine
nop();
}
else if (args->event == SCI_EVT_PARITY_ERR)
{
// from ERI/Group12 interrupt; receiver parity error occurred
// error char is in args->byte
// error condition is cleared in ERI routine
nop();
}
}
```

以下は、SSPI モードのコールバック関数のテンプレート例です。

```
void sspiCallback(void *p_args)
{
sci_cb_args_t *args;
args = (sci_cb_args_t *)p_args;
if (args->event == SCI_EVT_XFER_DONE)
{
// data transfer completed
nop();
}
else if (args->event == SCI_EVT_XFER_ABORTED)
{
// data transfer aborted
nop();
}
else if (args->event == SCI_EVT_OVFL_ERR)
{
// from ERI or Group12 (RX63x) interrupt; receiver overflow error occurred
// error char is in args->byte
// error condition is cleared in ERI/Group12 interrupt routine
nop();
}
}
```

本モジュールでは受信エラー割り込み発生時、調歩同期式モードにおける 1 バイト受信時、クロック同期式、または SSPI モードにおいて指定バイト数分の送受信完了時、送信完了割り込み発生時に、ユーザが指定したコールバック関数を呼び出します。ただし、FIFO 機能を使用した調歩同期式モードの場合は、最大 SCI_CFG_CHn_RX_FIFO_THRESH 回数受信するか、最後に受信したデータのストップビットから 15 etu の期間が経過したとき、コールバック関数が実行されます。（注 1）

コールバック関数は、R_SCI_Open()の第 4 引数にコールバック関数のアドレスを指定することで設定できます。コールバック関数が呼び出される際、以下の引数がセットされます。

```
typedef struct st_sci_cb_args     // コールバックの引数
{
sci_hdl_t hdl;              // イベント発生時のハンドル
sci_cb_evt_t event;         // イベント発生のトリガとなったイベント
uint8_t byte;               // イベント発生時の受信データ
uint8_t num;                      // 受信データサイズ数（FIFO 機能使用時のみ有効）
} sci_cb_args_t;

typedef enum e_sci_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
SCI_EVT_TEI,                      // TEI 割り込み発生;
SCI_EVT_RX_CHAR,                  // 文字が受信された; キューに配置済み
SCI_EVT_RX_CHAR_MATCH       // Received data match; already place in the queue. (受信したデータが一致。すでにキュー内に配置されている。)
SCI_EVT_RXBUF_OVFL,         // 受信キューがフル; これ以上のデータは保存不可
SCI_EVT_FRAMING_ERR,        // フレーミングエラー発生
SCI_EVT_PARITY_ERR,         // パリティエラー発生
// SSPI/クロック同期式モードのイベント
SCI_EVT_XFER_DONE,                // 転送完了
SCI_EVT_XFER_ABORTED,     // 転送中止
// Common event
SCI_EVT_OVFL_ERR            // オーバランエラー発生
} sci_cb_evt_t;
```

引数の型は void ポインタ型で渡されるため、コールバック関数の引数は以下の例を参考に void 型のポインタ変数としてください。コールバック関数内部で引数の値を使用する際はキャストして使用してください。

注1. etu（Elementary Time Unit）：1 ビットの転送期間

以下のイベント発生時は、コールバック関数の引数に格納される受信データは不定値となります。

- SCI_EVT_TEI
- SCI_EVT_XFER_DONE
- SCI_EVT_XFER_ABORTED
- SCI_EVT_OVFL_ERR (FIFO 機能が有効な場合)
- SCI_EVT_PARITY_ERR (FIFO 機能が有効な場合)
- SCI_EVT_FRAMING_ERR (FIFO 機能が有効な場合)

2.12 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) 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。

2.13 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

R_SCI_Open()

この関数は、SCI 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 関数に提供するチャネルのハンドルを設定します。この関数は他の API 関数を使用する前に実行される必要があります。

Format

```
sci_err_t       R_SCI_Open (
                uint8_t  const          chan,
                sci_mode_t  const       mode,
                sci_cfg_t * const       p_cfg,
                void                    (* const p_callback)(void *p_args),
                sci_hdl_t * const       p_hdl
)
```

Parameters

uint8_t const chan
初期化するチャネル

sci_mode_t const mode
動作モード（以下の列挙型参照）。

*sci_cfg_t * const p_cfg*
モードごとの設定共用体へのポインタ。構造体の要素（以下参照）はモードごとにあります。

p_callback
受信完了時、受信エラー発生時、送信完了時に割り込みから呼び出されるコールバック関数のポインタ（詳細は「2.11 コールバック関数」を参照ください）

*sci_hdl_t * const p_hdl*
チャネルのハンドルへのポインタ

R_SCI_Open()の戻り値が SCI_SUCCESS であることを確認し、R_SCI_GetVersion()を除く他の API 関数の第 1 引数にセットしてください。（詳細は「2.9 引数」を参照ください）

本モジュールでは、以下の SCI モードをサポートしています。指定されたモードによって、“p_cfg”に入る共用体の構造体要素が決定されます。

```
typedef enum e_sci_mode     // SCI 動作モード
{
   SCI_MODE_OFF=0,        // チャネルは使用されていません。
   SCI_MODE_ASYNC,         // 調歩同期式
   SCI_MODE_SSPI,         // SSPI
   SCI_MODE_SYNC,          // クロック同期式
   SCI_MODE_MAX           // 使用可能なモードの最大数
} sci_mode_t;
```

以下の#define は、調歩同期式モードの設定オプションを定義する構造体です。これらの値は SMR レジスタの定義に対応し、データ長、パリティ機能、STOP ビットの設定が可能です。また、sci_uart_t 構造体

の clk_src にて指定したクロックソース（内部クロック、外部クロックの 8x または 16x）と、sci_uart_t 構造体の baud_rate で指定したビットレートから、BRR レジスタ、SEMR レジスタの設定を行います。

ただし、指定したビットレートを保証するものではありません。（設定により多少の誤差が発生します。）

また、FIFO 機能が有効な場合にチャネル 10、11 をクロック同期モード、または簡易 SPI モードで使用する際は PCLKA/8 より速いビットレートを設定することは出来ません。

(例えば、PCLKA が 120MHz の場合は 15Mbps 以下のビットレートを設定してください。)

“p_cfg”の共用体を以下に示します。

```
typedef union
{
    sci_uart_t      async;
    sci_sync_sspi_t sync;
    sci_sync_sspi_t sspi;
} sci_cfg_t;
```

調歩同期式モードの設定で使用する構造体を以下に示します。

```
typedef struct st_sci_uart
{
    uint32_t    baud_rate;      // ie 9600, 19200, 115200 (内部クロックで有効)
    uint8_t     clk_src;        // use SCI_CLK_INT/EXT8/EXT16
    uint8_t     data_size;      // use SCI_DATA_nBIT
    uint8_t     parity_en;      // use SCI_PARITY_ON/OFF
    uint8_t     parity_type;    // use SCI_ODD/EVEN_PARITY
    uint8_t     stop_bits;      // use SCI_STOPBITS_1/2
    uint8_t     int_priority;   // txi, tei, rxi, eri INT priority; 1=low, 15=high
} sci_uart_t;
```

調歩同期式モードの設定で使用する構造体（sci_uart_t）の各メンバに使用する定義を以下に示します。

```
/* Definitions for the sck_src member.*/
#define SCI_CLK_INT         0x00  // 転送レートの生成に内部クロックを使用します。
#define SCI_CLK_EXT_8X      0x03  // 外部クロック 8 サイクルの期間が 1 ビット期間の転送レートになります。
#define SCI_CLK_EXT_16X     0x02  // u 外部クロック 16 サイクルの期間が 1 ビット期間の転送レートになります。

/* Definitions for the data_size member.*/
#define SCI_DATA_7BIT       0x40        // 7 ビット長
#define SCI_DATA_8BIT       0x00        // 8 ビット長

/* Definitions for the parity_en member.*/
#define SCI_PARITY_ON       0x20        // パリティあり
#define SCI_PARITY_OFF      0x00        // パリティなし

/* Definitions for the parity_type member.*/
#define SCI_ODD_PARITY      0x10        // 奇数パリティ
#define SCI_EVEN_PARITY     0x00        // 偶数パリティ

/* Definitions for the stop_bits member.
#define SCI_STOPBITS_2      0x08        // 2 ストップビット
#define SCI_STOPBITS_1      0x00        // 1 ストップビット
```

SSPI およびクロック同期式モードで使用する構造体を以下に示します。

```
typedef struct st_sci_sync_sspi
{
   sci_spi_mode_t  spi_mode;      // クロックの極性と位相; クロック同期式には使用されない
   uint32_t        bit_rate;      // ie 1Mbps の場合、1000000
   bool            msb_first;
   bool            invert_data;
   uint8_t         int_priority;  // RXI、ERI の割り込み優先レベル; 1=Low, 15=High
} sci_sync_sspi_t;
```

SSPIまたはクロック同期式モードの設定で使用する構造体(sci_sync_sspi_t)のspi_modeに使用する列挙型を以下に示します。

```
typedef enum e_sci_spi_mode
{
   SCI_SPI_MODE_OFF = 1, // クロック同期式モードで使用
   SCI_SPI_MODE_0 = 0x80,// SPMR レジスタ CKPH=1, CKPOL=0
                  // Mode 0: 00 Clock Polarity Low の状態, leading edge/立ち上がり
   SCI_SPI_MODE_1 = 0x40,// SPMR レジスタ CKPH=0, CKPOL=1
                  // Mode 1: 01 Clock Polarity Low の状態, trailing edge/立ち下がり
   SCI_SPI_MODE_2 = 0xC0,// SPMR レジスタ CKPH=1, CKPOL=1
                  // Mode 2: 10 Clock Polarity High の状態, leading edge/立ち下がり
   SCI_SPI_MODE_3 = 0x00 // SPMR レジスタ CKPH=0, CKPOL=0
                  // Mode 3: 11 Clock Polarity High の状態, trailing edge/立ち上がり
} sci_spi_mode_t;
```

Return Values

[SCI_SUCCESS]	*/* 成功; チャネルが初期化されました。 */*
[SCI_ERR_BAD_CHAN]	*/* チャネル番号が無効です。*/*
[SCI_ERR_OMITTED_CHAN]	*/* 対応する SCI_CHx_INCLUDED の値が無効(0) です。 */*
[SCI_ERR_CH_NOT_CLOSED]	*/* チャネルは現在使用中; R_SCI_Close() を実行してください。*/*
[SCI_ERR_BAD_MODE]	*/* 指定されたモードは対応していません。*/*
[SCI_ERR_NULL_PTR]	*/* “p_cfg”ポインタがNULL です。*/*
[SCI_ERR_INVALID_ARG]	*/* “p_cfg”の構造体要素に無効な値が含まれます。*/*
[SCI_ERR_QUEUE_UNAVAILABLE]	*/* 送信、受信キューのいずれか、または両方とも開けません。（調歩同期式モード） */*

Properties

ファイル r_sci_rx_if.h にプロトタイプ宣言されています。

Description

指定されたモードに SCI チャネルを初期化し、他の API 関数で使用するためのハンドルを*p_hdl で提供します。RXI および ERI 割り込みはすべてのモードで有効です。TXI 割り込みは調歩同期式モードで有効です。

Example : 調歩同期式モード

```
   sci_cfg_t   config;
   sci_hdl_t   Console;
   sci_err_t   err;
```

```
config.async.baud_rate = 115200;
config.async.clk_src = SCI_CLK_INT;
config.async.data_size = SCI_DATA_8BIT;
config.async.parity_en = SCI_PARITY_OFF;
config.async.parity_type = SCI_EVEN_PARITY;   // パリティが禁止のため無視
config.async.stop_bits = SCI_STOPBITS_1;
config.async.int_priority = 2;              // 1=最低値, 15=最高値

err = R_SCI_Open(SCI_CH1, SCI_MODE_ASYNC, &config, MyCallback, &Console);
```

Example : SSPI モード

```
sci_cfg_t   config;
sci_hdl_t   sspiHandle;
sci_err_t   err;

config.sspi.spi_mode = SCI_SPI_MODE_0;
config.sspi.bit_rate = 1000000;          // 1 Mbps
config.sspi.msb_first = true;
config.sspi.invert_data = false;
config.sspi.int_priority = 4;
err = R_SCI_Open(SCI_CH12, SCI_MODE_SSPI, &config, sspiCallback, &sspiHandle);
```

Example : クロック同期式モード

```
sci_cfg_t   config;
sci_hdl_t   syncHandle;
sci_err_t   err;

config.sync.spi_mode = SCI_SPI_MODE_OFF;
config.sync.bit_rate = 1000000;          // 1 Mbps
config.sync.msb_first = true;
config.sync.invert_data = false;
config.sync.int_priority = 4;
err = R_SCI_Open(SCI_CH12, SCI_MODE_SYNC, &config, syncCallback, &syncHandle);
```

Special Notes:

EMR.ABCS、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、低いビットエラーレートを保障するものではありません。

調歩同期式モードで外部クロックを使用する場合、R_SCI_Open()関数の呼び出し前に端子の方向を、R_SCI_Open()関数の呼び出し後に端子の機能とモードを選択するようにしてください。以下に RX111 でチャネル 1 を使用する場合の設定例を示します。

R_SCI_Open()関数呼び出し前

```
PORT1.PDR.BIT.B7 = 0;      // SCK 端子の方向を入力に設定（デフォルト）
```

R_SCI_Open()関数呼び出し後

```
MPC.P17PFS.BYTE = 0x0A;    // 端子機能選択 P17 を SCK1 として使用
PORT1.PMR.BIT.B7 = 1;      // SCK 端子のモードを周辺機能に設定
```

通信に使用される端子の設定は、R_SCI_Open()関数の呼び出し前に端子の方向およびその出力を、R_SCI_Open()関数の呼び出し後に端子の機能とモードを選択するようにしてください。以下に RX64M で SSPI のチャネル 6 を使用する場合の設定例を以下に示します。

R_SCI_Open()関数呼び出し前

```
PORT0.PODR.BIT.B2 = 0;      // Low に設定
```

```
   PORT0.PODR.BIT.B0 = 0;      // Low に設定
   PORT0.PDR.BIT.B2 = 1;       // SCK 端子の方向を出力に設定
   PORT0.PDR.BIT.B0 = 1;       // MOSI 端子の方向を出力に設定
   PORT0.PDR.BIT.B1 = 0;       // MISO 端子の方向を入力 に設定

R_SCI_Open()関数呼び出し後

   MPC.P00PFS.BYTE = 0x0A;     // 端子機能選択 P00 を MOSI として使用
   MPC.P01PFS.BYTE = 0x0A;     // 端子機能選択 P01 を MISO として使用
   MPC.P02PFS.BYTE = 0x0A;     // 端子機能選択 P02 を SCK として使用
   PORT0.PMR.BIT.B0 = 1;       // 端子のモードを周辺機能に設定
   PORT0.PMR.BIT.B1 = 1;       // 端子のモードを周辺機能に設定
   PORT0.PMR.BIT.B2 = 1;       // 端子のモードを周辺機能に設定
```

調歩同期式モードを使用する場合、1 チャネルにつきバイトキューを 2 つ使用します。必要に応じて、バイトキューの数を調整してください。詳細はアプリケーションノート「バイト型キューバッファ（BYTEQ）モジュール Firmware Integration Technology (R01AN1683) 」を参照してください。

R_SCI_Close()

この関数は SCI 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

```
sci_err_t       R_SCI_Close (
                sci_hdl_t const   hdl
)
```

Parameters

sci_hdl_t const hdl

チャネルのハンドル

R_SCI_Open()が正常に処理された際の hdl をセットしてください。

Return Values

[SCI_SUCCESS] */* 成功; チャネルを無効にしました。 */*

[SCI_ERR_NULL_PTR] */* “hdl”が NULL です。 */*

Properties

ファイル r_sci_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 SCI チャネルを無効にし、モジュールストップに移行します。

Example

```
sci_hdl_t   Console;
  ...
err = R_SCI_Open(SCI_CH1, SCI_MODE_ASYNC, &config, MyCallback, &Console);
  ...
err = R_SCI_Close(Console);
```

Special Notes:

本関数は実行中の送信または受信を中止します。

R_SCI_Send()

送信中でなければ、送信処理を行います。調歩同期式モードでは、以降の送信処理のためにデータをキューに配置します。

Format

```
sci_err_t       R_SCI_Send (
                sci_hdl_t const    hdl,
                 uint8_t             *p_src,
                 uint16_t const      length
)
```

Parameters

sci_hdl_t const hdl

チャネルのハンドル
R_SCI_Open()が正常に処理された際の hdl をセットしてください。

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

[SCI_SUCCESS]	*/* 送信初期化処理完了、または送信データをキューに配置（調歩同期式）*/*
[SCI_ERR_NULL_PTR]	*/* “hdl”が NULL です。*/*
[SCI_ERR_BAD_MODE]	*/* 指定されたモードはサポートされていません。*/*
[SCI_ERR_INSUFFICIENT_SPACE]	*/* キューに全データを配置できる十分なスペースがありません。*/*
[SCI_ERR_XCVR_BUSY]	*/* チャネルは現在使用中です */*

Properties

ファイル r_sci_rx_if.h にプロトタイプ宣言されています。

Description

歩同期式モードでは、ハンドルで指定された SCI チャネルが送信中でない場合、送信キューにデータを配置し送信を開始します。SSPI およびクロック同期式モードでは、データはキューに配置されず、送信中でない、かつ受信中でない場合は、送信がすぐに開始されます。送信はすべて割り込みで処理されます。

SSPI モードでの SS 端子の切り替えは、本モジュールでは対応していません。対象デバイスの SS 端子は、本関数を呼び出す前に有効にしておいてください。

同様に、クロック同期式、調歩同期式での CTS/RTS 端子も、端子の切り替えは、本モジュールでは対応していません。

Example : 調歩同期式モード

```
#define STR_CMD_PROMPT "Enter Command:"
sci_hdl_t Console;
sci_err_t err;

err = R_SCI_Send(Console, STR_CMD_PROMPT, sizeof(STR_CMD_PROMPT));

// 送信の完了を待たずに、この関数から復帰します。TEI 割り込みを
// 用いることで、キューに格納された全データの送信完了を検出できます。
```

Example : SSPI モード

```
sci_hdl_t  sspiHandle;
sci_err_t  err;
uint8_t    flash_cmd,sspi_buf[10];

// フラッシュデバイスにコマンドを送信して ID を供給する */
FLASH_SS = SS_ON;                       // GPIO のフラッシュスレーブ選択を有効にする
flash_cmd = SF_CMD_READ_ID;

R_SCI_Send(sspiHandle, &flash_cmd, 1);
while (SCI_SUCCESS != R_SCI_Control(sspiHandle, SCI_CMD_CHECK_XFER_DONE, NULL))
{
}

/* フラッシュデバイスから ID を読み込む */
R_SCI_Receive(sspiHandle, sspi_buf, 5);
while (SCI_SUCCESS != R_SCI_Control(sspiHandle, SCI_CMD_CHECK_XFER_DONE, NULL))
{
}

FLASH_SS = SS_OFF;                              // GPIO のフラッシュスレーブ選択を無効にする
```

Example : クロック同期式モード

```
#define STRING1 "Test String"
sci_hdl_t  lcdHandle;
sci_err_t  err;


// LCD ディスプレイに文字列を送付して、完了待ち */
R_SCI_Send(lcdHandle, STRING1, sizeof(STRING1));

while (SCI_SUCCESS != R_SCI_Control(lcdHandle, SCI_CMD_CHECK_XFER_DONE, NULL))
{
}
```

Special Notes:

なし

R_SCI_Receive()

調歩同期式モードで、RXI 割り込みによってセットされたデータをキューから取得します。その他のモードでは、送信、または受信中でなければ、受信処理を行います。

Format

```
sci_err_t       R_SCI_Receive (
                sci_hdl_t const   hdl,
                uint8_t           *p_dst,
                uint16_t const    length
)
```

Parameters

sci_hdl_t const hdl

チャネルのハンドル

R_SCI_Open()が正常に処理された際の hdl をセットしてください。

uint8_t p_dst*

取得したデータを配置するバッファへのポインタ

uint16_t const length

読み込むバイト数

Return Values

[SCI_SUCCESS]	*/* R 要求バイト数のデータが p_dst に配置されました（調歩同期式）。受信初期化処理が完了しました（SSPI/クロック同期式）。*
[SCI_ERR_NULL_PTR]	*/* “hdl”が NULL です。*
[SCI_ERR_BAD_MODE]	*/* 指定されたモードはサポートされていません。*
[SCI_ERR_INSUFFICIENT_DATA]	*/* 受信キューに十分なデータがありません（調歩同期式）。*
[SCI_ERR_XCVR_BUSY]	*/* チャネルは現在使用中です（SSPI/クロック同期式）。*

Properties

ファイル r_sci_rx_if.h にプロトタイプ宣言されています。

Description

調歩同期式モードでは、ハンドルで指定された SCI チャネルで受信されたデータを受信キューから取得します。もし受信データが要求したバイト数に満たない場合は、エラーコードをセットして この関数から戻ります。SSPI／クロック同期式モードでは、本 API は送受信で動作するために、送信または受信中でなければ、データの受信をすぐに開始します。データの受信中は、SCI_CFG_DUMMY_TX_BYTE（r_sci_config.h で定義）が送信されます。

受信中にエラーが発生した場合、R_SCI_Open()で指定されたコールバック関数が実行されます。正常に受信したかどうかは、コールバック関数に引数で渡されるイベントを参照して判断してください。詳細は「2.11 コールバック関数」を参照ください。

SSPI モードでの SS 端子の切り替えは、本モジュールでは対応していません。対象デバイスの SS 端子は、本関数を呼び出す前に有効にしておいてください。

Example : 調歩同期式モード

```
sci_hdl_t Console;
    sci_err_t err;
uint8_t   byte;

/* echo 文字列 */
while (1)
```

```
{
    while (SCI_SUCCESS != R_SCI_Receive(Console, &byte, 1))
    {
    }
    R_SCI_Send(Console, &byte, 1);
}
```

Example : SSPI モード

```
sci_hdl_t  sspiHandle;
    sci_err_t  err;
uint8_t    flash_cmd,sspi_buf[10];


// フラッシュデバイスにコマンドを送信して ID を提供する */

FLASH_SS = SS_ON;                       // GPIO のフラッシュスレーブ選択を有効にする
flash_cmd = SF_CMD_READ_ID;

R_SCI_Send(sspiHandle, &flash_cmd, 1);
while (SCI_SUCCESS != R_SCI_Control(sspiHandle, SCI_CMD_CHECK_XFER_DONE, NULL))
{
}

/* フラッシュデバイスから ID を読み込む */
R_SCI_Receive(sspiHandle, sspi_buf, 5);
while (SCI_SUCCESS != R_SCI_Control(sspiHandle, SCI_CMD_CHECK_XFER_DONE, NULL))
{
}

FLASH_SS = SS_OFF;                              // GPIO のフラッシュスレーブ選択を無効にする
```

Example : クロック同期式モード

```
sci_hdl_t  sensorHandle;
    sci_err_t  err;
uint8_t    sensor_cmd,sync_buf[10];


// SENSOR にコマンドを送信して、読み込んだデータを提供する */

sensor_cmd = SNS_CMD_READ_LEVEL;

R_SCI_Send(sensorHandle, &sensor_cmd, 1);
while (SCI_SUCCESS != R_SCI_Control(sensorHandle, SCI_CMD_CHECK_XFER_DONE, NULL))
{
}

/* SENSOR からレベルを読み込む */
R_SCI_Receive(sensorHandle, sync_buf, 4);
while (SCI_SUCCESS != R_SCI_Control(sensorHandle, SCI_CMD_CHECK_XFER_DONE, NULL))
{
}
```

Special Notes:

コールバック関数の引数に渡される内容については、「2.11 コールバック関数」の章で説明していますのでご確認ください。

調歩同期モードでは、データが一致することが検出された場合、受信したデータはキュー内に保存され、SCI_EVT_RX_CHAR_MATCH イベントを使用して、コールバック関数によりユーザへの通知を行います。

R_SCI_SendReceive()

この関数はクロック同期式および SSPI モードでのみ使用できます。送信中でない、かつ受信中でなければ、データの送信および受信を同時に行います。

Format

```
sci_err_t       R_SCI_SendReceive (
                sci_hdl_t const    hdl,
                uint8_t            *p_src,
                uint8_t            *p_dst,
                uint16_t const     length
)
```

Parameters

sci_hdl_t const hdl
チャネルのハンドル

R_SCI_Open()が正常に処理された際の hdl をセットしてください。

uint8_t p_src*
送信データへのポインタ

uint8_t p_dst*
データを配置するバッファへのポインタ

uint16_t const length
読み込むバイト数

Return Values

```
[SCI_SUCCESS]          /* データ転送が開始されました。 */
[SCI_ERR_NULL_PTR]     /* “hdl”が NULL です。 */
[SCI_ERR_BAD_MODE]     /* チャネルのモードが SSPI／クロック同期式モードではありません。 */
[SCI_ERR_XCVR_BUSY]    /* チャネルは現在使用中です。 */
```

Properties

ファイル r_sci_rx_if.h にプロトタイプ宣言されています。

Description

送信中でない、かつ受信中でない場合、p_src バッファからデータを送信し、同時にデータを受信して、p_dst バッファに配置します。

本モジュールでは、SSPI の SS 端子の切り替えには対応していません。本関数を呼び出す前に、対象デバイスの SS 端子を有効にしておく必要があります。

同様に、クロック同期式、調歩同期式での CTS/RTS 端子も、端子の切り替えは、本モジュールでは対応していません。

Example : SSPI モード

```
sci_hdl_t  sspiHandle;
    sci_err_t  err;
uint8_t in_buf[2] = {0x55, 0x55};        // 初期値設定
```

```
/* 一度の API 呼び出しで、フラッシュのステータスを読み込む */

//受信ステータスへの応答として、1 バイトのダミーデータを送信するために、コマンドの配列を呼び出す。
uint8_t out_buf[2] = {SF_CMD_READ_STATUS_REG, SCI_CFG_DUMMY_TX_BYTE };

FLASH_SS = SS_ON;

err = R_SCI_SendReceive(sspiHandle, out_buf, in_buf, 2);
while (SCI_SUCCESS != R_SCI_Control(sspiHandle, SCI_CMD_CHECK_XFER_DONE, NULL))
{
}

FLASH_SS = SS_OFF;

// “in_buf[1]”にステータスが格納される
```

Special Notes:

コールバック関数の引数に渡される内容については、「2.11 コールバック関数」の章で説明していますのでご確認ください。

R_SCI_Control()

この関数は、対象の SCI チャネルに対して、動作モードの設定および制御を行います。

Format

```
sci_err_t       R_SCI_Control (
                sci_hdl_t const     hdl,
                sci_cmd_t const     cmd,
                void                *p_args
)
```

Parameters

sci_hdl_t const hdl

チャネルのハンドル

R_SCI_Open()が正常に処理された際の hdl をセットしてください。

sci_cmd_t const cmd

実行するコマンド（以下にコマンドの列挙型を示します。）

*void *p_args*

コマンドごとの引数（以下参照）へのポインタ。void *に型変換されます。

有効な *cmd* 値を以下に示します。

```
typedef enum e_sci_cmd          // SCI Control() コマンド
{
   // 全モード
   SCI_CMD_CHANGE_BAUD,       // ビットレートを変更
   SCI_CMD_CHANGE_TX_FIFO_THRESH,     // 送信 FIFO しきい値変更(FIFO 機能を搭載する MCU のみ)
   SCI_CMD_CHANGE_RX_FIFO_THRESH,     // 受信 FIFO しきい値変更(FIFO 機能を搭載する MCU のみ)
   SCI_CMD_SET_RXI_PRIORITY,  // 受信プライオリティ（TXI、RXI で別の割り込み優先レベルを
// 設定できる MCU のみ）
   SCI_CMD_SET_TXI_PRIORITY,  // 送信プライオリティ（TXI、RXI で別の割り込み優先レベルを
// 設定できる MCU のみ）

   // 調歩同期式モードで使用可能なコマンド
   SCI_CMD_EN_NOISE_CANCEL,   // ノイズ除去機能を有効にする
   SCI_CMD_EN_TEI,            //本コマンドは無効なコマンドです(旧バージョンとの
// 互換性維持のために残してあります)
   SCI_CMD_OUTPUT_BAUD_CLK,   // SCK 端子のビットレートと同じ周波数のクロックを出力
   SCI_CMD_START_BIT_EDGE,    // RXDn 端子の立ち下がりでスタートビットを検出する
                              // （RXDn 端子の Low レベルで検出（デフォルト））
   SCI_CMD_GENERATE_BREAK,    // ブレークコンディションを生成する
   SCI_CMD_TX_Q_FLUSH,        // 送信キューをフラッシュ
   SCI_CMD_RX_Q_FLUSH,        // 受信キューをフラッシュ
   SCI_CMD_TX_Q_BYTES_FREE,   // 送信キューの未使用バイト数を取得
   SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ, // 読み込み可能なバイト数を取得
   SCI_CMD_COMPARE_RECEIVED_DATA, // Compare received data with comparison data (受信データを比較対象
データと比較する)

   // 調歩同期式／クロック同期式モードで使用可能なコマンド
   SCI_CMD_EN_CTS_IN,         // CTS 入力を有効にする（デフォルトは RTS 出力）
```

```
   // SSPI／クロック同期式モードで使用可能なコマンド
   SCI_CMD_CHECK_XFER_DONE,    // 送信、受信、または送受信の完了をチェック。完了している場合は“SCI_SUCCESS”を返す。
   SCI_CMD_ABORT_XFER,         // 通信を中断します。
   SCI_CMD_XFER_LSB_FIRST,     // LSB ファーストに設定します。
   SCI_CMD_XFER_MSB_FIRST,     // MSB ファーストに設定します。
   SCI_CMD_INVERT_DATA,        // 極性反転に設定します。

   // SSPI モードで使用可能なコマンド
   SCI_CMD_CHANGE_SPI_MODE     // SPI モードを変更します。

} sci_cmd_t;
```

以下のコマンド以外は引数を必要としません。"p_args"引数には FIT_NO_PTR を設定してください。

SCI_CMD_CHANGE_BAUD の引数には、変更するビットレートを指定した sci_baud_t 型変数へのポインタを設定してください。sci_baud_t 構造体を以下に示します。

```
typedef struct st_sci_baud
{
   uint32_t    pclk;       // 周辺クロックレート （例: 24000000 = 24MHz）
   uint32_t    rate;       // e.g. 9600, 19200, 115200
} sci_baud_t;
```

SCI_CMD_TX_Q_BYTES_FREE および SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ 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を設定してください。

SCI_CMD_CHANGE_SPI_MODE の引数には、変更する SPI モードを格納した列挙型（sci_sync_sspi_t）の変数へのポインタを設定してください。

SCI_CMD_SET_TXI_PRIORITY および SCI_CMD_SET_RXI_PRIORITY（TXI、RXI で別の割り込み優先レベルを設定できる MCU のみ）の引数には、優先レベルを格納した uint8_t 型変数へのポインタを設定してください。

Return Values

[SCI_SUCCESS]	*/* 成功; チャネルが初期化されました。 */*
[SCI_ERR_NULL_PTR]	*/* “hdl”または“p_args”が NULL です。 */*
[SCI_ERR_BAD_MODE]	*/*指定されたモードはサポートされていません。 */*
[SCI_ERR_INVALID_ARG]	*/* “cmd”、または“p_args”の要素に無効な値が含まれます。*/*

Properties

ファイル r_sci_rx_if.h にプロトタイプ宣言されています。

Description

この関数は、本モジュールの設定変更やモジュールのステータス取得など、ハードウェアの特殊な機能を設定するために使用します。

ハードウェア制御はデフォルトで RTS 機能になっています。SCI_CMD_EN_CTS_IN を発行することで、CTS 機能に変更することができます。

Example : 調歩同期式モード

```
   sci_hdl_t   Console;
   sci_cfg_t   config;
   sci_baud_t  baud;
   sci_err_t   err;
   uint16_t    cnt;
```

```
R_SCI_Open(SCI_CH1, SCI_MODE_ASYNC, &config, MyCallback, &Console);
R_SCI_Control(Console, SCI_CMD_EN_NOISE_CANCEL, NULL);
R_SCI_Control(Console, SCI_CMD_EN_TEI, NULL);
  ...
/* 低消費電力モードクロック切り替えのため、ビットレートをリセット */
baud.pclk = 8000000;      // 8 MHz
baud.rate = 19200;
R_SCI_Control(Console, SCI_CMD_CHANGE_BAUD, (void *)&baud);
  ...
/* 数メッセージ送信後、送信キューの空スペースを確認*/
R_SCI_Control(Console, SCI_CMD_TX_Q_BYTES_FREE, (void *)&cnt);
  ...
/* 受信キューにデータがあるかどうかを確認*/
R_SCI_Control(Console, SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ, (void *)&cnt);
```

Example : SSPI モード

```
sci_cfg_t   config;
sci_spi_mode_t   mode;
sci_hdl_t   sspiHandle;
sci_err_t   err;

config.sspi.spi_mode     = SCI_SPI_MODE_0;
config.sspi.bit_rate     = 1000000;          // 1 Mbps
config.sspi.msb_first    = true;
config.sspi.invert_data  = false;
config.sspi.int_priority = 4;
err = R_SCI_Open(SCI_CH12, SCI_MODE_SSPI, &config, sspiCallback, &sspiHandle);
  ...
  ...
// 別のモードで動作するスレーブデバイスに変更
mode = SCI_SPI_MODE_3;
R_SCI_Control(sspiHandle, SCI_CMD_CHANGE_SPI_MODE, (void *)&mode);
```

Special Notes:

SCI_CMD_CHANGE_BAUD を使用した場合、指定したビットレートから BRR、SEMR.ABSC、SMR.CKS の最適値を算出します。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、低いビットエラーレートを保障するものではありません。

SCI_CMD_EN_CTS_IN コマンドを使用する場合、R_SCI_Open()関数の呼び出し前に端子の方向を、R_SCI_Open()関数の呼び出し後に端子の機能とモードを選択するようにしてください。以下に RX111 でチャネル 1 を使用する場合の設定例を示します。

```
R_SCI_Open()関数呼び出し前

PORT1.PDR.BIT.B4 = 0;      // CTS/RTS 端子の方向を入力に設定（デフォルト）

R_SCI_Open()関数呼び出し後

MPC.P14PFS.BYTE = 0x0B;    // P14 の機能に CTS を選択
PORT1.PMR.BIT.B4 = 1;      // CTS/RTS 端子のモードを周辺機能端子に設定
```

SCI_CMD_OUTPUT_BAUD_CLK を使用する場合、R_SCI_Open()関数の呼び出し前に端子の方向を、R_SCI_Open()関数の呼び出し後に端子の機能とモードを選択するようにしてください。以下に RX111 でチャネル 1 を使用する場合の設定例を示します。

```
R_SCI_Open()関数呼び出し前
```

```
   PORT1.PDR.BIT.B7 = 1;      // SCK 端子の方向を出力に設定

R_SCI_Open()関数呼び出し後

   MPC.P17PFS.BYTE = 0x0A;    // P17 の機能に SCK1 を選択
   PORT1.PMR.BIT.B7 = 1;      // SCK 端子のモードを周辺機能端子に設定
```

以下のコマンドは送信中に実行可能です。それ以外のコマンドは、送信中に実行しないでください。

- SCI_CMD_TX_Q_BYTES_FREE
- SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ
- SCI_CMD_CHECK_XFER_DONE
- SCI_CMD_ABORT_XFER

本関数を実行すると一時的に TXD 端子が Hi-Z になります。下記の方法により、TXDn ラインがハイインピーダンスにならないようにしてください。

SCI_CMD_GENERATE_BREAK コマンドを使用する場合:

- TXD 端子は抵抗を介して Vcc に接続（プルアップ）してください。

上記以外のコマンドを使用する場合:

以下のいずれかの対応を行ってください。

- TXD 端子を抵抗を介して Vcc に接続（プルアップ）する。
- SCI_Control 関数を実行する前に、TXD 端子を汎用入出力ポートに切り替える。SCI_Control 関数を実行後、TXD 端子を周辺機能に設定する。

R_SCI_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_SCI_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_sci_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。

Example

```
uint32_t   version;
  ...
version = R_SCI_GetVersion();
```

Special Notes:

なし

4. 端子設定

SCI 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_SCI_Open 関数を呼び出した後に行ってください。

e² 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	出力される関数名	備考
全デバイス共通	R_SCI_PinSet_SCIx	x:チャネル番号

5. デモプロジェクト

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。本 FIT モジュールには以下のデモプロジェクトが含まれます。

5.1 sci_demo_rskrx113

sci_demo_rskrx113 は RSKRX113 スターターキットの RX113 シリアル通信インタフェース（SCI）のシンプルなデモです(FIT モジュール "r_sci_rx")。デモでは、UART として構成された SCI チャネルを介してターミナルと通信を行います。このデモでは RSKRX113 はオンボードで RS232 のインタフェースを持っていないため、USB 仮想 COM インタフェースを RSKRX113 のシリアルとして用いています。ターミナルエミュレーションアプリケーションを実行している PC が、ユーザとの入出力用に必要となります。

設定と実行

1. RSKRX113 基板のジャンパを準備します：J15 ジャンパを 1-2 に J16 は 2-3 に設定します。
2. このサンプルアプリケーションをビルドし、RSK ボードにダウンロードし、デバッガを使用しアプリケーションを実行します。
3. PC のシリアルポートに RSK ボードのシリアルポートを接続します。

   RSKRX113 のシリアルのデモでは USB 仮想 COM インタフェースを使用します。ルネサスの USB シリアルデバイスドライバがインストールされている PC の USB ポートに接続してください。
4. PC 上のターミナルエミュレーションプログラム（以下、ターミナル）を開きます、そして、RSK の USB シリアル仮想 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。
5. この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設定を行います。

   115200bps、8 ビットデータ、パリティなし、1 ストップビット、フロー制御なし。
6. ソフトウェアはターミナルから文字を受信するために待機します：

   PC のターミナルの準備が整ったら、PC のターミナルウィンドウでキーボードのキーを押し、ターミナル上に出力される、FIT モジュールのバージョン番号を確認します。
7. このアプリケーションは、エコーモードのままになります。ターミナルに入力された任意のキーが SCI ドライバによって受信され、その後、このアプリケーションはターミナルへ文字を戻します。

対応ボード

RSKRX113

5.2 sci_demo_rskrx231

sci_demo_rskrx231 は RSKRX231 スターターキットの RX231 シリアル通信インタフェース（SCI）のシンプルなデモです(FIT モジュール "r_sci_rx")。デモでは、UART として構成された SCI チャネルを介してターミナルと通信を行います。RSKRX231 のシリアルのデモでは USB 仮想 COM インタフェースを使用します。ターミナルエミュレーションアプリケーションを実行している PC がユーザとの入出力用に必要となります。

設定と実行

1. このサンプルアプリケーションをビルドし、RSK ボードにダウンロードし、デバッガを使用しアプリケーションを実行します。
2. PC のシリアルポートに RSK ボードのシリアルポートを接続します。

   RSKRX231 のシリアルのデモでは USB 仮想 COM インタフェースを使用します。ルネサスの USB シリアルデバイスドライバがインストールされている PC の USB ポートに接続してください。
3. PC 上のターミナルエミュレーションプログラム（以下、ターミナル）を開きます、そして、RSK の USB シリアル仮想 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。
4. この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設定を行います。

   115200bps、8 ビットデータ、パリティなし、1 ストップビット、フロー制御なし。
5. ソフトウェアはターミナルから文字を受信するために待機します：

   PC のターミナルの準備が整ったら、PC のターミナルウィンドウでキーボードのキーを押し、ターミナル上に出力される、FIT モジュールのバージョン番号を確認します。
6. このアプリケーションは、エコーモードのままになります。ターミナルに入力された任意のキーが SCI ドライバによって受信され、その後、このアプリケーションはターミナルへ文字を戻します。

対応ボード

RSKRX231

5.3 sci_demo_rskrx64m

sci_demo_rskrx64m は RSKRX64M スターターキットの RX64M シリアル通信インタフェース（SCI）のシンプルなデモです(FIT モジュール "r_sci_rx")。デモでは、UART として構成された SCI チャネルを介してターミナルと通信を行います。このデモでは RSKRX64M はオンボードで RS232 のインタフェースを持っていないため、USB 仮想 COM インタフェースを RSKRX64M のシリアルとして用いています。ターミナルエミュレーションアプリケーションを実行している PC が、ユーザとの入出力用に必要となります。

設定と実行

1. RSKRX64M 基板のジャンパを準備します：J16 と J18 を 2-3 に設定します。

2. このサンプルアプリケーションをビルドし、RSK ボードにダウンロードし、デバッガを使用しアプリケーションを実行します。

3. PC のシリアルポートに RSK ボードのシリアルポートを接続します。

   RSKRX64M のシリアルのデモでは USB 仮想 COM インタフェースを使用します。ルネサスの USB シリアルデバイスドライバがインストールされている PC の USB ポートに接続してください。

4. PC 上のターミナルエミュレーションプログラム（以下、ターミナル）を開きます、そして、RSK の USB シリアル仮想 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。

5. この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設定を行います。

   115200bps、8 ビットデータ、パリティなし、1 ストップビット、フロー制御なし。

6. ソフトウェアはターミナルから文字を受信するために待機します：

   PC のターミナルの準備が整ったら、PC のターミナルウィンドウでキーボードのキーを押し、ターミナル上に出力される、FIT モジュールのバージョン番号を確認します。

7. このアプリケーションは、エコーモードのままになります。ターミナルに入力された任意のキーが SCI ドライバによって受信され、その後、このアプリケーションはターミナルへ文字を戻します。

対応ボード

RSKRX64M

5.4 sci_demo_rskrx71m

sci_demo_rskrx71m は RSKRX71M スターターキットの RX71M シリアル通信インタフェース（SCI）のシンプルなデモです(FIT モジュール "r_sci_rx")。デモでは、UART として構成された SCI チャネルを介してターミナルと通信を行います。このデモでは RSKRX71M はオンボードで RS232 のインタフェースを持っていないため、USB 仮想 COM インタフェースを RSKRX71M のシリアルとして用いています。ターミナルエミュレーションアプリケーションを実行している PC は、ユーザの入力と出力のために必要となります。

設定と実行

1. RSKRX71M 基板のジャンパを準備します：J16 と J18 を 2-3 に設定します。

2. このサンプルアプリケーションをビルドし、RSK ボードにダウンロードし、デバッガを使用しアプリケーションを実行します。

3. PC のシリアルポートに RSK ボードのシリアルポートを接続します。

   RSKRX71M のシリアルのデモでは USB 仮想 COM インタフェースを使用します。ルネサスの USB シリアルデバイスドライバがインストールされている PC の USB ポートに接続してください。

4. PC 上のターミナルエミュレーションプログラム（以下、ターミナル）を開きます、そして、RSK の USB シリアル仮想 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。

5. この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設定を行います。

   115200bps、8 ビットデータ、パリティなし、1 ストップビット、フロー制御なし。

6. ソフトウェアはターミナルから文字を受信するために待機します：

   PC のターミナルの準備が整ったら、PC のターミナルウィンドウでキーボードのキーを押し、ターミナル上に出力される、FIT モジュールのバージョン番号を確認します。

7. このアプリケーションは、エコーモードのままになります。ターミナルに入力された任意のキーが SCI ドライバによって受信され、その後、このアプリケーションはターミナルへ文字を戻します。

対応ボード

RSKRX71M

5.5 sci_demo_rskrx65n

sci_demo_rskrx65n は RSKRX65N スターターキットの RX65N シリアル通信インタフェース（SCI）のシンプルなデモです(FIT モジュール "r_sci_rx")。デモでは、UART として構成された SCI チャネルを介してターミナルと通信を行います。このデモでは RSKRX65N はオンボードで RS232 のインタフェースを持っていないため、USB 仮想 COM インタフェースを RSKRX65N のシリアルとして用いています。ターミナルエミュレーションアプリケーションを実行している PC は、ユーザの入力と出力のために必要となります。

設定と実行

1. このサンプルアプリケーションをビルドし、RSK ボードにダウンロードし、デバッガを使用しアプリケーションを実行します。

2. PC のシリアルポートに RSK ボードのシリアルポートを接続します。

   RSKRX65N のシリアルのデモでは USB 仮想 COM インタフェースを使用します。ルネサスの USB シリアルデバイスドライバがインストールされている PC の USB ポートに接続してください。

3. PC 上のターミナルエミュレーションプログラム（以下、ターミナル）を開きます、そして、RSK の USB シリアル仮想 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。

4. この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設定を行います。

   115200bps、8 ビットデータ、パリティなし、1 ストップビット、フロー制御なし。

5. ソフトウェアはターミナルから文字を受信するために待機します：

   PC のターミナルの準備が整ったら、PC のターミナルウィンドウでキーボードのキーを押し、ターミナル上に出力される、FIT モジュールのバージョン番号を確認します。

6. このアプリケーションは、エコーモードのままになります。ターミナルに入力された任意のキーが SCI ドライバによって受信され、その後、このアプリケーションはターミナルへ文字を戻します。

対応ボード

RSKRX65N

5.6 sci_demo_rskrx65n_2m

sci_demo_rskrx65n_2m は RSKRX65N-2MB スターターキットの RX65N-2MB シリアル通信インタフェース（SCI）のシンプルなデモです(FIT モジュール "r_sci_rx")。デモでは、UART として構成された SCI チャネルを介してターミナルと通信を行います。このデモでは RSKRX65N-2MB はオンボードで RS232 のインタフェースを持っていないため、USB 仮想 COM インタフェースを RSKRX65N-2MB のシリアルとして用いています。ターミナルエミュレーションアプリケーションを実行している PC は、ユーザの入力と出力のために必要となります。

設定と実行

1. このサンプルアプリケーションをビルドし、RSK ボードにダウンロードし、デバッガを使用しアプリケーションを実行します。

2. PC のシリアルポートに RSK ボードのシリアルポートを接続します。

   RSKRX65N-2MB のシリアルのデモでは USB 仮想 COM インタフェースを使用します。ルネサスの USB シリアルデバイスドライバがインストールされている PC の USB ポートに接続してください。

3. PC 上のターミナルエミュレーションプログラム（以下、ターミナル）を開きます、そして、RSK の USB シリアル仮想 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。

4. この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設定を行います。

   115200bps、8 ビットデータ、パリティなし、1 ストップビット、フロー制御なし。

5. ソフトウェアはターミナルから文字を受信するために待機します：

   PC のターミナルの準備が整ったら、PC のターミナルウィンドウでキーボードのキーを押し、ターミナル上に出力される、FIT モジュールのバージョン番号を確認します。

6. このアプリケーションは、エコーモードのままになります。ターミナルに入力された任意のキーが SCI ドライバによって受信され、その後、このアプリケーションはターミナルへ文字を戻します。

対応ボード

RSKRX65N-2MB

5.7 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「終了」をクリックします。

5.8 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.3.00)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e2 studio V.7.4.0 IAR Embedded Workbench for Renesas RX 4.10.1
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package 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 4.8.4.201803 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 リンクオプション：「Optimize size (サイズ最適化) (-Os)」を使用する場合、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 -Wl,--no-gc-sections これは、FIT 周辺機器モジュール内で宣言されている割り込み関数をリンカが誤って破棄（discard）することを回避（work around）するための対策です。
	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	Rev.3.00
使用ボード	Renesas Starter Kit+ for RX65N-2MB（型名：RTK50565Nxxxxxxxxx）

表 6.2 動作確認環境 (Rev.2.20)

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 V.7.3.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package 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	Rev.2.20
使用ボード	Renesas Starter Kit for RX72T（型名：RTK5572Txxxxxxxxxx）

表 6.3 動作確認環境 (Rev.2.11)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e2 studio V.7.3.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package 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	Rev.2.11
使用ボード	Renesas Starter Kit for RX66T（型名：RTK50566T0SxxxxxBE） Renesas Starter Kit+ for RX 65N-2MB（型名：RTK50565N2SxxxxxBE） Renesas Starter Kit+ for RX130-512KB（型名：RTK5051308SxxxxxBE）

表 6.4 動作確認環境 (Rev.2.10)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e2 studio V.7.0.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family V3.00.00
	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	Rev.2.10
使用ボード	Renesas Starter Kit for RX66T（型名：RTK50566T0SxxxxxBE） Renesas Starter Kit+ for RX 65N-2MB（型名：RTK50565N2SxxxxxBE） Renesas Starter Kit+ for RX130-512KB（型名：RTK5051308SxxxxxBE）

表 6.5 動作確認環境 (Rev.2.01)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e2 studio V6.0.0
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 V2.07.00
	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	Rev2.01
使用ボード	Renesas Starter Kit+ for RX65N-2MB（型名：RTK50565N2SxxxxxBE） Renesas Starter Kit for RX130-512KB（型名：RTK5051308SxxxxxBE）

表 6.6 動作確認環境 (Rev.2.00)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e2 studio V5.4.0（WS パッチ仕様）
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 V2.07.00
	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	Rev2.00
使用ボード	Renesas Starter Kit+ for RX65N-2MB（型名：RTK50565N2SxxxxxBE） Renesas Starter Kit for RX130-512KB（型名：RTK5051308SxxxxxBE）

表 6.7 動作確認環境 (Rev.1.90)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e2 studio V5.3.0.023
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	Rev1.90
使用ボード	Renesas Starter Kit for RX24U（型名：RTK500524USxxxxxBE） Renesas Starter Kit for RX24T（型名：RTK500524TSxxxBE） Renesas Starter Kit for RX113（型名：R0K505113SxxxBE） Renesas Starter Kit for RX65N（型名：RTK500565NSxxxxxBE）

表 6.8 動作確認環境 (Rev.1.80)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e² studio V5.0.1.005 ルネサスエレクトロニクス製 e² studio V5.0.0.043 ルネサスエレクトロニクス製 e² studio V4.3.0.007 ルネサスエレクトロニクス製 e² studio V4.2.0.012
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 V2.05.00 ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V2.04.01
	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	Rev1.80
使用ボード	Renesas Starter Kit for RX65N（型名：RTK500565NSxxxxxBE）（注 1） Renesas Starter Kit for RX64M（型名：R0K50564MSxxxBE）（注 2） Renesas Starter Kit for RX71M（型名：R0K50571MSxxxBE）（注 3） Renesas Starter Kit for RX231（型名：R0K505231SxxxBE）（注 4） Renesas Starter Kit for RX130（型名：RTK5005130SxxxBE）（注 4） Renesas Starter Kit for RX111（型名：R0K505111SxxxBE）（注 4） Renesas Starter Kit for RX23T（型名：RTK500523TSxxxBE）（注 4） Renesas Starter Kit for RX24T（型名：RTK500524TSxxxBE）（注 4） Renesas Starter Kit for RX113（型名：R0K505113SxxxBE）（注 4） Renesas Starter Kit for RX210（型名：R0K505210SxxxBE）（注 4） Renesas Starter Kit+ for RX63N（型名：R0K50563NSxxxBE）（注 4）

注 1. V5.0.1.005 の e2 studio と V2.05.00 の C コンパイラの組み合わせで確認しています。
注 2. V4.3.0.007 の e2 studio と V2.04.01 の C コンパイラの組み合わせで確認しています。
注 3. V4.2.0.012 の e2 studio と V2.04.01 の C コンパイラの組み合わせで確認しています。
注 4. V5.0.0.043 の e2 studio と V2.04.01 の C コンパイラの組み合わせで確認しています。

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_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「コンフィグ設定が間違っている場合のエラーメッセージ」エラーが発生します。

A：“r_sci_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_sci_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 コンパイル時の設定」を参照してください。

(4) Q：TXD 端子から送信データが出力されません。

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。

7. 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 C/C++コンパイラ CC-RX 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*-A151A/E

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.70	2015.09.30	—	初版発行
1.80	2016.10.01	1 3 4 5 6 8 9~10 11 12 13~15 16~19 20 21 23~24 26 27 33 34~37 38	・サポートしている MCU のリストに RX65N を追加 ・「1.概要」の SCI 周辺機能の記載内容を見直し ・「1.概要」の「割り込みと送受信について」の記載内容を見直し ・「1.概要」の「エラー検出について」の記載内容を見直し ・「1.1 SCI FIT モジュールとは」の章を追加 ・「3.1 概要」を 1.2 章に移動 ・「2.5 対応ツールチェーン」の記載内容を見直し ・「2.8 コンパイル時の設定」に以下の define を追加 SCI_CFG_CH10_FIFO_INCLUDED SCI_CFG_CH11_FIFO_INCLUDED SCI_CFG_CH10_TX_FIFO_THRESH SCI_CFG_CH11_TX_FIFO_THRESH SCI_CFG_CH10_RX_FIFO_THRESH SCI_CFG_CH11_RX_FIFO_THRESH ・「2.11 コードサイズ」を 2.9 章に移動し記載内容を見直し ・「2.10 引数」の章を追加し、チャネル管理用構造体の内容を記載 ・「3.2 戻り値」を 2.11 章に移動し記載内容を見直し ・「2.12 コールバック関数」の章を追加 ・「3.1 R_SCI_Open()」を一部見直し ・コールバックに関する記載を「2.12 コールバック関数」の章へ移動 ・「3.2 R_SCI_Close」を一部見直し ・「3.3 R_SCI_Send()」を一部見直し ・「3.4 R_SCI_Receive()」を一部見直し ・「3.5 R_SCI_SendReceive()」を一部見直し ・「3.6 R_SCI_Control()」を一部見直し、コマンドを追加 SCI_CMD_CHANGE_TX_FIFO_THRESH SCI_CMD_CHANGE_RX_FIFO_THRESH ・「4. 端子設定」の章を追加 ・「5. デモプロジェクト」の記載内容見直し ・テクニカルアップデート(TN-RX*-A151A/J)の対応を明記
1.90	2017.02.28	— 3 4,8,18 4 5 6 9,10 13,14 20	・FIT モジュールの RX24U グループ対応 ・「表 1.1 MCU グループに対応する SCI 周辺機能の一覧」に RX24U を追加 ・SCI_CMD_EN_TEI コマンドの使用方法に関する記述を削除 ・「エラー検出について」にて、FIFO 機能に関する説明を変更 ・「表 1.2 API 関数一覧」で R_SCI_Send および R_SCI_Receive 関数の説明を変更 ・「2.5 対応ツールチェーン」に RXC v2.06.00 を追加 ・「2.9 コードサイズ」の各メモリサイズを更新 ・「2.12 コールバック関数」で以下を変更 ・概要説明: 一部変更し、FIFO 機能有効時の説明を追加 ・イベント発生時、コールバック関数の引数に受信データが格納されないイベントの記載を追加 ・「3.1 R_SCI_Open()」の Special Notes に FIFO 機能有効時の通信エラーの処理方法を追加

Rev.	発行日	改訂内容	
		ページ	ポイント
1.90	2017.02.28	22 24	・「3.3 R_SCI_Send()」の Description の記載を変更 ・「3.4 R_SCI_Receive()」の Description にて、受信エラー発生時のコールバック関数に関する記載を変更
		29 29,30 32	・「3.6 R_SCI_Control()」で以下を変更 ・概要説明: 一部変更 ・Parameters: - コマンドに SCI_CMD_SET_RXI_PRIORITY、SCI_CMD_SET_TXI_PRIORITY を追加。 - SCI_CMD_EN_TEI コマンドのコメントを変更 - コメント未記載だったコマンドにコメントを追加 ・Special Notes: - 送信中に実行可能なコマンドの記載を追加 - コマンド使用時の TXD 端子の対応に関する記載を追加
		ﾌﾟﾛｸﾞﾗﾑ	・誤記修正 ・SCI_CMD_EN_TEI を何も処理しない無効なコマンドに変更（不要なコマンドだが、旧バージョンとの互換性のため残す） ・引数のチェックを、NULL と FIT_NO_PTR の両方でチェックするように修正 ・簡易 SPI モードの場合にコマンドに SCI_CMD_EN_CTS_IN を指定した場合、R_SCI_Control 関数が SCI_ERR_INVALID_ARG を返すように変更（簡易 SPI モードでは CTS 入力は無効な機能のため） ・sci_error 関数において、エラーフラグのクリア処理前に不要な論理演算を行っているため削除した ・以下の不具合を修正 ● 対象デバイス RX110/RX111/RX113/RX130/RX210/RX230/RX231/RX23T/RX24T/RX63N/RX631/RX64M/RX651/RX65N/RX71M ● 内容 クロック同期式モードによる受信処理において、指定した数よりも多くのデータを受信する可能性がある。 ● 発生条件 クロック同期式モードにおいて、2byte 以上のデータ受信する際、1 回目のダミーデータをライトした後から 2 回目のダミーデータ分のカウンタがデクリメントされる前までの間に 1 フレーム分以上の時間が経過した場合。 ● 対策 sci_receive_sync_data 関数のダミーデータライトの回数を 1 回にした（Rev.1.70 時点の仕様へ戻す)。 Rev1.90 以降の SCI FIT モジュールを使用すること。

Rev.	発行日	改訂内容	
		ページ	ポイント
1.90	2017.02.28	ﾌﾟﾛｸﾞﾗﾑ	・以下の不具合を修正 ● 対象デバイス RX110/RX111/RX113/RX130/RX210/RX230/RX231/RX23T/RX24T/RX63N/RX631/RX64M/RX651/RX65N/RX71M ● 内容 調歩同期モードでエラーが発生した場合、エラー割り込みが繰り返し動作し続けてメイン処理が動作しなくなる可能性がある。 ● 発生条件 調歩同期モード、かつコールバック関数なしに設定した時に、パリティエラー、オーバランエラー、フレーミングエラーのいずれかの通信エラーが発生した場合。 ● 対策 sci_error 関数において、エラーフラグのクリア処理がコールバック関数ありの時しか行われていなかったため、エラーフラグのクリアは必ず行うように修正した（Rev.1.70 時点の仕様へ戻す)。 Rev1.90 以降の SCI FIT モジュールを使用すること。
2.00	2017.07.24	—	・FIT モジュールの RX130 グループ（ROM 512KB 版を含む）、RX65N グループ(ROM 2MB 版を含む)対応
		—	文言見直し
		1	・関連ドキュメントに以下のドキュメントを追加: Renesas e^2 studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)
		7～13	・2.6 使用する割り込みベクタ: 追加
		20	・FIFO 機能を使用した場合のコールバック関数の呼び出し回数を 1 回に修正
		22	・2.14 FIT モジュールの追加方法: 変更
		27	・調歩同期式モードを使用する場合のバイトキューに対する注意文言追加
		36	・SCI_CMD_SET_RXI_PRIORITY、SCI_CMD_SET_TXI_PRIORITY コマンドを全モードで使用できるように変更
		42	・4. 端子設定: 「Smart Configurator」の記載を追加
		47	・5.6 デモのダウンロード方法: 追加
		48～50	・付録追加
		ﾌﾟﾛｸﾞﾗﾑ	・以下の不具合を修正しました。 ● 対象デバイス RX65N ● 内容 エラーフラグが解除されないため、エラー割り込みが常時発生し続けます。 ● 発生条件 FIFO 有効、かつコールバック関数を設定せずにオープンした場合に、受信エラーが起こると発生します。 ● 対策 FIFO 有効時の受信エラー解除処理が無かったため、追加しました。また、コールバック関数の設定有無に関わらず必ずエラー割り込み終了前に受信エラーを解除するように修正しました。 Rev2.00 以降の SCI FIT モジュールを使用してください。

Rev.	発行日	改訂内容	
		ページ	ポイント
2.00	2017.07.24	ﾌﾟﾛｸﾞﾗﾑ	・以下の不具合を修正しました。 ● 対象デバイス RX65N ● 内容 FIFO の送信しきい値、受信しきい値を変更する時に引数を指定しなかった場合、不定な値をしきい値に設定します。 ● 発生条件 R_SCI_Control 関数のコマンドに SCI_CMD_CHANGE_TX_FIFO_THRESH / SCI_CMD_CHANGE_RX_FIFO_THRESH を設定し、引数に NULL を設定した場合に発生します。 ● 対策 R_SCI_Control 関数に引数の NULL チェック処理を追加しました。 Rev2.00 以降の SCI FIT モジュールを使用してください。 ・以下の不具合を修正しました。 ● 対象デバイス RX65N ● 内容 送信中に再度送信開始すると、送信中のデータが強制停止し新しい送信も開始しません。 ● 発生条件 FIFO 有効時、クロック同期に設定したチャネルで送信中に送信開始すると発生します。 ● 対策 送信中に送信開始した場合は SCI_ERR_XCVR_BUSY を返し、送信を中断しないように修正しました。 Rev2.00 以降の SCI FIT モジュールを使用してください。 ・以下の不具合を修正しました。 ● 対象デバイス RX65N ● 内容 FIFO の受信しきい値を変更しても、受信が完了するとしきい値が"8"になります。 ● 発生条件 FIFO 有効時、クロック同期に FIFO の受信しきい値を初期値(8)以外に変更して受信すると発生します。 ● 対策 送信しきい値、受信しきい値の設定値をハンドラに保持するようにし、受信処理中に書き換えた値を初期値ではなくハンドラに保持した値に戻すように修正しました。 Rev2.00 以降の SCI FIT モジュールを使用してください。

Rev.	発行日	改訂内容	
		ページ	ポイント
2.00	2017.07.24	ﾌﾟﾛｸﾞﾗﾑ	・以下の不具合を修正しました。 ● 対象デバイス RX65N ● 内容 FIFO の受信しきい値を受信バイト数が超えても、受信割り込みが発生しません。 ● 発生条件 FIFO 有効時、クロック同期に FIFO の受信しきい値を初期値(8)未満に変更して受信データ数が 8 バイト未満だと発生します。 ● 対策 送信しきい値、受信しきい値の設定値をハンドラに保持するようにし、受信処理中に書き換えた値を初期値ではなくハンドラに保持した値に戻すように修正しました。 Rev2.00 以降の SCI FIT モジュールを使用してください。 ・以下の不具合を修正しました。 ● 対象デバイス RX65N ● 内容 FIFO の受信しきい値が"8"の場合、8 バイト受信後にコールバック関数が連続して 8 回実行されます。 ● 発生条件 FIFO 有効時、コールバック関数を設定してオープンし複数バイト受信すると発生します。(8 バイト未満でも発生) ● 対策 FIFO 有効時は受信割り込み 1 回につき、コールバック関数を 1 回実行するように修正しました。 コールバック関数の引数に受信バイト数を格納するメンバ"num"を追加しました。 受信バイト数が受信バッファより大きい場合、格納可能な分だけバッファに格納し、残りは破棄されます。（この際、コールバック関数のイベントは"SCI_EVT_RXBUF_OVFL"になります） Rev2.00 以降の SCI FIT モジュールを使用してください。 ・以下の不具合を修正しました。 ● 対象デバイス RX64M/RX71M/RX65N ● 内容 送信優先レベル、受信優先レベルを変更した場合、不定なレベルが設定されます。 ● 発生条件 R_SCI_Control 関数のコマンドに SCI_CMD_SET_TXI_PRIORITY/SCI_CMD_SET_RXI_PRIORITY を設定し、引数に NULL を設定した場合に発生します。 ● 対策 R_SCI_Control 関数に引数の NULL チェック処理と割り込み優先レベルの範囲チェック処理を追加しました。 Rev2.00 以降の SCI FIT モジュールを使用してください。

Rev.	発行日	改訂内容	
		ページ	ポイント
2.00	2017.07.24	ﾌﾟﾛｸﾞﾗﾑ	以下の不具合を修正しました。 ● 対象デバイス RX64M/RX71M/RX65N ● 内容 割り込み優先レベルの変更が調歩同期の場合しか設定できません。 ● 発生条件 クロック同期の場合に R_SCI_Control 関数のコマンドに SCI_CMD_SET_TXI_PRIORITY/SCI_CMD_SET_RXI_PRIORITY を設定すると発生します。 ● 対策 クロック同期、調歩同期の両方で割り込み優先レベルの変更が有効になるように修正しました。 Rev2.00 以降の SCI FIT モジュールを使用してください。
2.01	2017.10.31	47 48 49 50	「5.5 sci_demo_rskrx65n」を追加。 「5.6 sci_demo_rskrx65n_2m」を追加。 「5.8 デモのダウンロード方法」を追加。 「6.1 動作確認環境」に、Rev.2.01 に対応する表を追加。
2.10	2018.09.28	1、3 14 17 49	RX66T のサポートを追加。 RX66T に対応する構成を追加。 RX66T に対応するコードサイズを追加。 「6.1 動作確認環境」：Rev 2.10 に対応する表を追加。
2.11	2018.11.16	— 1、3 49	XML 内にドキュメント番号を追加。 RX651 のサポートを追加。 Renesas Starter Kit+ for RX66T の型名を変更。 Rev 2.11 に対応する表を追加。
2.20	2019.02.01	— 1、3、12、14 18 25-42 50	RX72T グループのサポートを追加。 RX72T グループのサポートを追加。 RX72T に対応するコードサイズを追加。 各 API 関数で「Reentrant」の説明を削除。 「6.1 動作確認環境」Rev 2.20 に対応する表を追加。
3.00	2019.05.20	— 1 3 5 6 7 13 55 60 プログラム	以下のコンパイラをサポート。 - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX RX210、RX631、RX63N の更新終了につき、「対象デバイス」からこれらのデバイスを削除。 「ターゲットコンパイラ」のセクションを追加。 関連ドキュメントを削除。 「1.2」で RX210、RX63N、RX631 を削除。 「1.4」で RX63N, と RX631 を削除。 「2.2 ソフトウェアの要求」r_bsp v5.20 以上が必要 「2.4」で RX210、RX63N、RX631 を削除。 「2.8 コードサイズ」セクションを更新。 表 6.1「動作確認環境」： Rev.3.00 に対応する表を追加。 「Web サイトおよびサポート」のセクションを削除。 GCC と IAR コンパイラに関して、以下を変更。 1. R_SCI_GetVersion 関数のインライン展開を削除。 2. 「evenaccess」を、BSP のマクロ定義で置き換えた。 3. NOP を BSP の固有関数で置き換えた。 4. 割り込み関数の宣言を、BSP のマクロ定義で置き換えた。

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。