RX ファミリ
SCI モジュール
Firmware Integration Technology

R01AN1815JJ0180
Rev. 1.80
2016.10.01

要旨

本ドキュメントは、Firmware Integration Technology（FIT）を使用した SCI モジュールについて説明します。

本モジュールは、RX MCU のシリアルコミュニケーションインタフェース（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

本モジュールは以下のデバイスで使用できます。

- RX110、RX111、RX113 グループ
- RX130 グループ
- RX210 グループ
- RX230、RX231、RX23T グループ
- RX24T グループ
- RX63N、RX631 グループ
- RX64M グループ
- RX65N 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)
- ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)
- バイト型キューバッファ（BYTEQ）モジュール Firmware Integration Technology (R01AN1683)
- e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)
- CS+に組み込む方法 Firmware Integration Technology (R01AN1826)

目次

1. 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
RX110			○	○			
RX111			○	○			
RX113			○	○			
RX130					○	○	
RX210	○	○					
RX230	○	○					
RX231	○	○					
RX23T					○		
RX24T					○		
RX63N	○	○					
RX631	○	○					
RX64M					○	○	
RX65N					○	○	○
RX71M					○	○	

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_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。この割り込みの間に、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、送信が可能になります。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。ただし、TEI 割り込みを使用する場合には、R_SCI_Control()関数で、対象のチャネルに対して TEI 割り込みを有効にする必要があります。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()関数を呼び出し、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、Receive()または 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。Receive()または 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「2.10 引数」を参照ください。

エラー検出について

受信デバイスが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「2.12 コールバック関数」を参照ください。割り込みでは、コールバック関数の有無に関わらず、エラーコンディションがクリアされるまで、繰り返し SSR エラーフラグに 0 を書き込みます。

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、2.13 FIT モジュールの追加方法を参照してください。

1.2 API の概要

表 1.2 に本モジュールに含まれる API 関数を示します。また、2.9 コードサイズに本モジュールで必要なメモリサイズを示します。

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。受信デバイスで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ユーザに通知するために、コールバック関数のポインタを取得します。
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()	データをキューに配置し、送信デバイスが動作中でなければ、送信処理を行います。
R_SCI_Receive()	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。
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。送信および受信デバイスが動作中でなければ、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。

2. API 情報

2.1 ハードウェアの要求

本モジュールを使用するには、ご使用の MCU が以下の機能をサポートしていることが要求されます。

- SCI 周辺機能

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 SCI

本モジュールでは、SCI 周辺機能の機能を使用できます。SCI チャネルは、“r_sci_rx_config.h”ファイルで無効に設定して、個別に省くことができます。

2.2.2 GPIO

本モジュールでは、SCI の各チャネルに対応した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。SSPI モードでは、SS 端子用のポート端子が必要です。

2.3 ソフトウェアの要求

本モジュールは以下のソフトウェアに依存します。

- r_bsp
- r_byteq（調歩同期式モードのみ）

2.4 制限事項

本モジュールには以下の制限があります。

- 対象デバイスが RX63N、RX631 の場合、SCI のグループ割り込みのベクタ設定を本モジュール内で行っているため、SCI と使用するグループ割り込みが競合する RSPI は同時に使用できません。

2.5 対応ツールチェーン

本モジュールは下記ツールチェーンで動作確認を行っています。

- Renesas RX Toolchain v.2.04.01（RX110、RX111、RX113、RX130、RX210、RX230、RX231、RX23T、RX24T、RX63N、RX631、RX64M、RX71M）
- Renesas RX Toolchain v.2.05.00（RX65N）

2.6 ヘッダファイル

API 呼び出しと対応するインタフェース定義はすべて、r_sci_rx_if.h ファイルに記述されています。ビルド時に設定可能なコンフィギュレーションオプションは、r_sci_rx_config.h に記述されています。

上記 2 ファイルは、ユーザアプリケーションによってインクルードする必要があります。

2.7 整数型

コードをわかりやすく、また移植が容易に行えるように、本プロジェクトでは ANSI C99 (Exact width integer types (固定幅の整数型)) 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

ビルド時に設定可能なコンフィギュレーションオプションは r_sci_rx_config.h ファイルに含まれます。下表に各設定の概要を示します。

コンフィギュレーションオプション（r_sci_rx_config.h）（1/2）

定義	説明
#define 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。
#define SCI_CFG_ASYNC_INCLUDED ※デフォルト値は“1” #define SCI_CFG_SYNC_INCLUDED ※デフォルト値は“0” #define 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。
#define 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。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値です。
#define SCI_CFG_CH0_INCLUDED #define SCI_CFG_CH1_INCLUDED #define SCI_CFG_CH2_INCLUDED #define SCI_CFG_CH3_INCLUDED #define SCI_CFG_CH4_INCLUDED #define SCI_CFG_CH5_INCLUDED #define SCI_CFG_CH6_INCLUDED #define SCI_CFG_CH7_INCLUDED #define SCI_CFG_CH8_INCLUDED #define SCI_CFG_CH9_INCLUDED #define SCI_CFG_CH10_INCLUDED #define SCI_CFG_CH11_INCLUDED #define 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。**
#define SCI_CFG_CH0_TX_BUFSIZ #define SCI_CFG_CH1_TX_BUFSIZ #define SCI_CFG_CH2_TX_BUFSIZ #define SCI_CFG_CH3_TX_BUFSIZ #define SCI_CFG_CH4_TX_BUFSIZ #define SCI_CFG_CH5_TX_BUFSIZ #define SCI_CFG_CH6_TX_BUFSIZ #define SCI_CFG_CH7_TX_BUFSIZ #define SCI_CFG_CH8_TX_BUFSIZ #define SCI_CFG_CH9_TX_BUFSIZ #define SCI_CFG_CH10_TX_BUFSIZ #define SCI_CFG_CH11_TX_BUFSIZ #define 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。

コンフィギュレーションオプション（r_sci_rx_config.h）（2/2）	
定義	**説明**
#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。
#define 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、送信完了割り込みの処理をコードに含めます。R_SCI_Control()のコマンドを使用して、指定のチャネルの TEI 割り込みを許可します。TEI 割り込みは、データの最終バイトの最終ビットが送信され、送信デバイスがアイドル状態になった場合にのみ発生します。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。
#define 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。受信デバイスのグループ 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。
#define SCI_CFG_ERI_TEI_PRIORITY ※デフォルト値は“3”	このオプションは RX64M、RX71M のみに適用されます。受信デバイスのエラー割り込み（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、送信デバイスがアイドル状態になったことを示します（調歩同期式モード）。
#define SCI_CFG_CH10_FIFO_INCLUDED #define 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。
#define SCI_CFG_CH10_TX_FIFO_THRESH #define 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。
#define SCI_CFG_CH10_RX_FIFO_THRESH #define 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。

2.9 コードサイズ

ツールチェーンでのコードサイズは、最適化レベル 2、およびコードサイズ重視の最適化を前提としたサイズです。ROM (コードおよび定数) と RAM (グローバルデータ) のサイズは、本モジュールのコンフィギュレーションヘッダファイルで設定される、ビルド時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。

本モジュールは他の FIT モジュールに依存するため、それらの FIT モジュールのメモリサイズも考慮する必要があります。本モジュールが依存する FIT モジュールに必要なメモリサイズについては、セクション「2.3 ソフトウェアの要求」に記載された各モジュールのドキュメントでご確認ください。

以下に RX130、RX231、RX64M、RX65N の場合の各通信方式のサイズを示します。

ROM および RAM の最小サイズ（バイト）

デバイス	分類		使用メモリ		備考
			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	2759 バイト	2494 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	2592 バイト	2198 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	3788 バイト	3346 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		168 バイト		
RX231	調歩同期	ROM	2692 バイト	2426 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	2415 バイト	2131 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	3741 バイト	3346 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		136 バイト		
RX64M	調歩同期	ROM	2833 バイト	2683 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	2204 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	3876 バイト	3467 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		136 バイト		

ROM および RAM の最小サイズ（バイト）					
デバイス	分類		使用メモリ		備考
			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	
RX65N	調歩同期	ROM	2767 バイト	2521 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	2476 バイト	2192 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	3875 バイト	3455 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		136 バイト		
	FIFO モード + 調歩同期	ROM	3428 バイト	3084 バイト	1 チャネル使用
		RAM	196 バイト	196 バイト	1 チャネル使用
	FIFO モード + クロック同期	ROM	3473 バイト	2965 バイト	1 チャネル使用
		RAM	40 バイト	40 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	4999 バイト	4517 バイト	計 2 チャネル使用
		RAM	400 バイト	400 バイト	計 2 チャネル使用
	最大使用スタックサイズ		152 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多少異なります。

2.10 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ギュ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
    uint8_t         tx_dflt_thresh; // 送信 FIFO しきい値
#endif
} sci_ch_ctrl_t;
```

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_sci_rx_if.h に記載されています。

```
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。デバイスがビジーの状態です。

      // 調歩同期式モードのみが有効な場合
      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。

      // クロック同期式、または SSPI モードのみが有効な場合
      SCI_ERR_XFER_NOT_DONE         // データ転送は処理中です。
} sci_err_t;
```

2.12 コールバック関数

本モジュールではエラー受信割り込み発生時、調歩同期式モードにおける 1 バイト受信時、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。

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されます。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
} 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；送信デバイスはアイドル
    SCI_EVT_RX_CHAR,          // 文字が受信された；キューに配置済み
    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,      // 受信デバイスでフレーミングエラー発生
    SCI_EVT_PARITY_ERR,       // 受信デバイスで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
    // 共通イベント
    SCI_EVT_OVFL_ERR          // オーバランエラー ；受信デバイスのハードウェア
} 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。

SCI_EVT_TEI、SCI_EVT_XFER_DONE、SCI_ECT_XFER_ABORTED のイベントが発生した場合、または FIFO 機能が有効な場合、受信データは格納されません。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();
     }
#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();
     }
}
```

2.13 FIT モジュールの追加方法

本モジュールは、e² studio で、使用するプロジェクトごとに追加する必要があります。

プロジェクトへの追加方法は、FIT プラグインを使用する方法と、手動で追加する方法があります。

FIT プラグインを使用すると、簡単にプロジェクトに FIT モジュールを追加でき、またインクルードファイルパスも自動的に更新できます。このため、プロジェクトへ FIT モジュールを追加する際は、FIT プラグインの使用を推奨します。

FIT プラグインを使用して FIT モジュールを追加する方法は、アプリケーションノート「e2 studio に組み込む方法(R01AN1723)」の「3. FIT プラグインを使用して FIT モジュールをプロジェクトに追加する方法」を参照してください。

FIT プラグインを使用せず手動で FIT モジュールを追加する方法は、「4. 手作業で FIT モジュールをプロジェクトに追加する方法」を参照してください。

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。

3. API 関数

3.1 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

chan
初期化するチャネル

mode
動作モード（以下の列挙型参照）。

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
RXI が検出されたとき、受信デバイスでエラーが検出されたとき、またはコンディションが送信終了（TEI）のときに、割り込みから呼び出される関数のポインタ
（詳細は「2.12 コールバック関数」を参照ください）

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「2.10 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、SEMR/BRR レジスタに対応し、構造体の clk_src にて、指定される、外部クロックの 8x および 16x または内部クロックと、構造体の Boud_rate で指定されるビットレートから、BRR レジスタ、SEMR レジスタの設定を行います。ただし、ビットレート誤差を保障するものではありません。

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の設定が可能です。)

```
#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 //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。
#define SCI_DATA_7BIT 0x40
#define SCI_DATA_8BIT 0x00
#define SCI_PARITY_ON 0x20
#define SCI_PARITY_OFF 0x00
#define SCI_ODD_PARITY 0x10
#define SCI_EVEN_PARITY 0x00
#define SCI_STOPBITS_2 0x08
#define SCI_STOPBITS_1 0x00
```

調歩同期式モードの実行時の設定オプションは、以下の構造体（“p_cfg”の要素）で宣言されます。

```
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;
```

SSPI またはクロック同期式モードが設定される場合、設定構造体の以下の列挙型が使用されます。

```
typedef enum e_sci_spi_mode
{
     SCI_SPI_MODE_OFF = 1, // チャネルはクロック同期式モードで使用されています。
     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;
```

SSPI および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_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;
```

“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;
```

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 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	*/* 指定されたモードには現在対応していません。*/*
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。TEI 割り込みの有効設定については、Control()関数のセクションをご覧ください。

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

```
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:

本モジュールは、BSP_PCLKB_HZ（r_bsp の mcu_info.h で定義）を使って、BRR、SEMR.ABCS、SMR.CKS の最適値を算出するためにアルゴリズムを使用します。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。

外部クロックが調歩同期式モードで使用される場合、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.P00.PFS.BYTE = 0x0A;     //端子機能選択 P00 を MOSI として使用
MPC.P01.PFS.BYTE = 0x0A;     //端子機能選択 P01 を MISO として使用
MPC.P02.PFS.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
```

3.2 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

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ます。モジュールストップに移行しますが、データ領域は解放しません。次にチャネルを有効にしたときに、その領域を使うことができます。

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

```
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。

3.3 R_SCI_Send()

送信デバイスが動作中でなければ、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

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。

p_src

送信データへのポインタ

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	*/*そのモードは現在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

調歩同期式モードでは、ハンドルで指定された SCI チャネルの送信動作のために、送信キューにデータを配置します。送信中のバイトデータがなければ、送信はすぐに開始されます。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、送信および受信デバイスが動作中でない場合は、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。

また、FIFO 機能を使用するチャネルにおいて、送信処理中は本 API 関数を呼び出さないでください。送信処理の完了は送信完了割り込みが発生することにより確認できます。送信完了割り込みを使用するには、下記の 2 つの設定が必要です。

・r_sci_rx_config.h のコンフィギュレーションオプション設定で SCI_CFG_TEI_INCLUDED を"1"にする
・R_SCI_Control()で送信完了割り込みを許可にする

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。

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 : 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。しかし、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

3.4 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、受信デバイスが動作中でなければ、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

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。

p_dst

取得したデータを配置するバッファへのポインタ

length

読み込むバイト数

Return Values

SCI_SUCCESS	*/*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	*/*そのモードは現在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、送信デバイスが動作中でなければ、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によって処理されます。このとき、エラーコードは提供されません。

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。

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 : 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:

コールバック関数によるエラーレポートについては、「2.12 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。

3.5 R_SCI_SendReceive()

この関数はクロック同期式および SSPI モードでのみ使用されます。送信および受信デバイスのいずれも動作していなければ、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

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。

p_src

送信データへのポインタ

p_dst

データを配置するバッファへのポインタ

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

送信および受信デバイスのいずれも動作していなければ、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。

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: 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:

コールバック関数によるエラーレポートについては、「2.12 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。

3.6 R_SCI_Control()

この関数は、対象の SCI チャネルに対して、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。

Format

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

Parameters

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。

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。）

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
SCI_CMD_CHANGE_RX_FIFO_THRESH, // 受信 FIFO しきい値変更

// 調歩同期式モードで使用されるコマンド
SCI_CMD_EN_NOISE_CANCEL,      // ノイズ除去機能を有効にする
SCI_CMD_EN_TEI,               // TEI 割り込みを許可
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

// 調歩同期式／クロック同期式モードで使用されるコマンド
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,
SCI_CMD_XFER_LSB_FIRST,
SCI_CMD_XFER_MSB_FIRST,
SCI_CMD_INVERT_DATA,

// SSPI コマンド
SCI_CMD_CHANGE_SPI_MODE
} sci_cmd_t;
```

ほとんどのコマンドは引数を必要とせず、“p_args”には NULL を取ります。

SCI_CMD_CHANGE_BAUD の引数の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、カウント値を保持する uint16_t 変数へのポインタを取ります。

SCI_CMD_CHANGE_SPI_MODE の引数には、要求される新規モードを含む列挙型の変数へのポインタを取ります。

SCI_CMD_SET_TXI_PRIORITY および SCI_CMD_SET_RXI_PRIORITY（RX64M／RX71M）の引数には、優先レベルを保持する uint8_t 変数へのポインタを取ります。送信および受信割り込みの優先レベルには、同じ値が設定されます（R_SCI_Open()関数のデフォルト）。

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	*/*そのモードは現在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されます。

デフォルトでは、受信デバイスがデータを受信できる状態であれば、SCI 周辺機能によって RTSn#CTSn#端子から Low が出力されます。端子は RTS 出力信号として機能します。SCI_CMD_EN_CTS_IN を発行することで、端子は CTS の入力信号を受け取ります。この場合、RTSn#CTSn#端子が High のとき、再度 Low になるまで送信デバイスは無効となります。RTSn#CTSn#が High になったときに、送信デバイスがバイトデータを送信中の場合、停止する前にその送信を完了します。

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 : 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; // 8MHz
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:

本モジュールは、BRR、SEMR.ABCS、SMR.CKS の最適値を算出するためにアルゴリズムを使用します。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。

コマンド SCI_CMD_EN_CTS_IN が使用される場合、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
```

コマンド SCI_CMD_OUTPUT_BAUD_CLK が使用される場合、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
```

3.7 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。

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_SCI_GetVersion();
```

Special Notes:

本関数は#pragma ディレクティブを使ったインライン関数となります。

4. 端子設定

本モジュールで使用する端子の設定は、R_SCI_Open 関数を呼び出した後に行ってください。

5. デモプロジェクト

デモプロジェクトは、独立したプログラムになっています。モジュールには依存モジュール(例えば r_bsp)を利用した main()関数を含みます。デモプロジェクトの標準的な命名規則は、<module>_demo_<board>となり、<module>は周辺の略語(例えば s12ad、CMT、SCI）、<board>は標準 RSK（例えば rskrx113）です。例えば、RSKRX113 用の s12ad FIT モジュールのデモプロジェクトは s12ad_demo_rskrx113 として名前が付けられます。同様にエクスポートされた.zip ファイルは <module>_demo_<board>.zip となります。同じような例として、zip 形式のエクスポート/インポートされたファイルは、s12ad_demo_rskrx113.zip として名前が付けられます。

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。

Setup and Execution

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。この場合、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 ポートに接続します。

4. PC 上のターミナルエミュレーションプログラムを開きます、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 のターミナルプログラムの準備が整ったら、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。

Boards Supported

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。

Setup and Execution

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。この場合、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 ポートに接続します。

3. PC 上のターミナルエミュレーションプログラムを開きます、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 のターミナルプログラムの準備が整ったら、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。

Boards Supported

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。

Setup and Execution

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。この場合、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 ポートに接続します。

4. PC 上のターミナルエミュレーションプログラムを開きます、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 のターミナルプログラムの準備が整ったら、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。

Boards Supported

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。

Setup and Execution

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。この場合、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 ポートに接続します。

4. PC 上のターミナルエミュレーションプログラムを開きます、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 のターミナルプログラムの準備が整ったら、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。

Boards Supported

RSKRX71M

5.5 ワークスペースへのデモ追加

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。

6. 提供するモジュール

提供するモジュールは、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。

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/J

ホームページとサポート窓口

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

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

改訂記録	RX ファミリ　アプリケーションノート SCI モジュール Firmware Integration Technology

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	・サポートしている MCU のリストに RX65N を追加
		3	・「1.概要」の SCI 周辺機能の記載内容を見直し
		4	・「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
		5	・「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
		6	・「2.5 対応ツールチェーン」の記載内容を見直し
		8	・「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
		9~10	・「2.11 コードサイズ」を 2.9 章に移動し記載内容を見直し
		11	・「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
		12	・「3.2 戻り値」を 2.11 章に移動し記載内容を見直し
		13~15	・「2.12 コールバック関数」の章を追加
		16~19	・「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
		20	・「3.2 R_SCI_Close」を一部見直し
		21	・「3.3 R_SCI_Send()」を一部見直し
		23~24	・「3.4 R_SCI_Receive()」を一部見直し
		26	・「3.5 R_SCI_SendReceive()」を一部見直し
		27	・「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
		33	・「4. 端子設定」の章を追加
		34~37	・「5. デモプロジェクト」の記載内容見直し
		38	・テクニカルアップデート(TN-RX*-A151A/J)の対応を明記

製品ご使用上の注意事項

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

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子
   （または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

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

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。
注2. 本資料において使用されている「当社製品」とは、注１において定義された当社の開発、製造製品をいいます。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

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

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

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