# RX ファミリ

## RSCI モジュール Firmware Integration Technology

### 要旨

本アプリケーションノートでは、Firmware Integration Technology（FIT）を採用した強化シリアルコミュニケーションインタフェース（RSCI）モジュールについて説明します。本モジュールでは、RSCI 周辺機能の全チャネルに非同期、同期、SPI（SSPI）サポートを提供するために RSCI を使用します。本書の中で「本モジュール（ドライバ）」とは、RSCI FIT モジュールを指します。

### 動作確認デバイス

- RX671 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。

### 対象コンパイラ

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX
- IAR C/C++ Compiler for Renesas RX

各コンパイラの動作確認内容については「6.1 動作確認環境」を参照してください。

# 目次

# 1. 概要

## 1.1 RSCI FIT モジュール

RSCI FIT モジュールは、API としてプロジェクトに組み込むことで使用できます。この FIT モジュールをプロジェクトに組み込む方法に関する詳細は、「2.13 FIT モジュールの追加方法」を参照してください。

## 1.2 RSCI FIT モジュールの概要

RSCI は、非同期およびクロック同期シリアル通信を処理できます。RSCI の送受信ブロックには、32 ステージの FIFO バッファがあり、FIFO コンポジションの選択、効率的な送受信のほか、連続通信も可能です。

さらに、本ドライバの非同期モードでは以下の機能をサポートします。

- ノイズキャンセル
- SCK 端子でのボークロック出力
- CTS または RTS の一方向フロー制御

すべての基本 UART、マスターSPI モード機能が本ドライバによってサポートされます。

本ドライバでサポートされない機能は以下のとおりです。

- 拡張マンチェスタモード
- 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
- IrDA 機能
- RZI コード

チャネルの処理

本ドライバは、周辺機能で全チャネルをサポートするマルチチャネルドライバです。必要に応じて本ドライバの RAM 使用量とコードサイズを縮小するために、特定のチャネルをコンパイル時の定義により除外できます。その際の定義は“r_rsci_rx_config.h”で指定されます。

個別のチャネルは、アプリケーション内で R_RSCI_Open()を呼び出すことにより初期化されます。この関数は、周辺機能に出力を適用し、指定モード特有の設定を初期化します。この関数により、チャネルを一意で識別するためのハンドルが返されます。ハンドルは内部ドライバ構造体を参照し、この構造体がチャネルのレジスタセット、バッファ、その他の重要な情報へのポインタを保持します。また、他の API 関数の引数としても使用されます。

割り込みおよび送受信

本ドライバによってサポートされる割り込みは、TXI、TEI、RXI、ERI です。非同期モードの場合、送受信データのキュー登録にはリング（循環）バッファが使用されます。コンパイル時にリングバッファのサイズも設定できます。

TXI および TEI 割り込みは、非同期モードでしか使用できません。TXI 割り込みは、TDR レジスタ内の送信データが TSR レジスタにシフトされたときに発生します。この割り込み時には、送信リングバッファ内の次のバイトが TDR レジスタに配置され、送信に備えます。R_RSCI_Open()呼び出しでコールバック関数が提供される場合、ここで呼び出され、TEI イベントが受け渡されます。TEI 割り込み用サポートは、“r_rsci_rx_config.h”内の設定により本ドライバから削除できます。

RXI 割り込みは、RDR レジスタの RDAT フィールドが受信データにシフトされるたびに発生します。非

同期モードでは、後で R_RSCI_Receive()を呼び出すことによりアプリケーションレベルでアクセスできるように、このバイトが割り込み時に受信リングバッファに読み込まれます。コールバック関数を提供する場合、受信イベント発生時に呼び出されます。受信用キューが満杯の場合、コールバック関数はキュー満杯イベント発生時に呼び出され、最後に受信したバイトは保存されません。SSPI および同期モードでは、シフトインバイトは、最後の R_RSCI_Receive()または R_RSCI_SendReceive()呼び出しから指定された受信用バッファに直接読み込まれます。R_RSCI_Receive()または R_RSCI_SendReceive()呼び出し前に受信したデータは無視されます。SSPI および同期モードでは、データは RXI 割り込みハンドラ内で送受信されます。残りの送受信用データの数は、R_RSCI_Open 関数の第 4 パラメータに設定されたハンドル内の送信カウンタ（tx_cnt）および受信カウンタ（rx_cnt）の値を用いてチェックできます。詳細は「2.10 パラメータ」を参照してください。

エラー検出

受信デバイスによりフレーミング、オーバラン、パリティエラーが検出されると、ERI 割り込みが発生します。コールバック関数を提供する場合、この割り込みは発生したエラーの種類を判別し、アプリケーションにイベントを通知します。詳細は「2.12 コールバック関数」を参照してください。

本 FIT モジュールは、コールバック関数の提供の有無に関係なく、ERI 割り込みハンドラ内のエラーフラグをクリアします。FIFO 関数が有効の場合、コールバック関数はエラーフラグのクリア前に呼び出されます。したがって、エラー発生場所のデータを判定するには、受信したデータの数に対して RDR レジスタを読み取ります。詳細は「2.12 コールバック関数」を参照してください。

## 1.3　API の概要

表1.1 に本モジュールに含まれる API 関数を示します。

表1.1　API 関数

| 関数名 | 説明 |
|---|---|
| R_RSCI_Open() | RSCI チャネルへの出力適用、関連レジスタの初期化、割り込みの有効化、他の API 関数で使用するためのチャネルハンドルの提供を行います。受信エラーまたはその他の割り込みイベントの発生時に呼び出されるコールバック関数を指定します。 |
| R_RSCI_Close() | RSCI チャネルへの出力を除去し、関連する割り込みを無効にします。 |
| R_RSCI_Send() | トランスミッタを使用しない場合に送信を開始します。 |
| R_RSCI_Receive() | 非同期モードの場合、RXI 割り込みによって満杯になったキューからデータを取得します。<br>同期および SSPI モードの場合、トランシーバを使用しない場合にダミーデータの送受信を開始します。 |
| R_RSCI_SendReceive() | 同期および SSPI モード専用。トランシーバを使用しない場合にデータを同時に送受信します。 |
| R_RSCI_Control() | RSCI チャネル用に特別ハードウェア/ソフトウェア動作を処理します。 |
| R_RSCI_GetVersion() | 実行時にドライバのバージョン番号を返します。 |

## 1.4　制限事項

なし。

## 1.5 RSCI FIT モジュールを使用する

### 1.5.1 RSCI FIT モジュールを C++プロジェクト内で使用する

C++プロジェクトでは、FIT RSCI モジュールのインタフェースヘッダファイルを extern “C”の宣言に追加してください。

```
Extern “C”
{
#include “r_smc_entry.h”
#include “r_rsci_rx_if.h”
}
```

# 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。

- RSCI
- 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) v6.10 以降
- r_byteq (非同期モード専用)

## 2.3 制限事項

### 2.3.1 RAM の配置に関する制限事項

FIT では、NULL と同じ値を API 関数のポインタ引数として設定すると、パラメータチェックによってエラーが返されることがあります。したがって、NULL と同じ値をポインタ引数として API 関数に受け渡さないでください。

ライブラリ関数仕様により、NULL 値は 0 として定義されます。したがって、上記の現象は、API 関数ポインタ引数に受け渡される変数または関数が、RAM の先頭アドレス（アドレス 0x0）に配置される場合に発生します。この場合、API 関数ポインタ引数に受け渡される変数または関数がアドレス 0x0 に位置付けられないように、セクション設定を変更するか、RAM の先頭に置くダミー変数を準備してください。

CCRX プロジェクト（e$^2$ studio V21.7.0）の場合、変数がアドレス 0x0 に位置付けられないようにするため、RAM 先頭アドレスは 0x4 として設定されます。GCC プロジェクト（e$^2$ studio V21.7.0）および IAR プロジェクト（EWRX V4.20.1）の場合、RAM 先頭アドレスは 0x0 であるため、前述の対策が必要です。

セクションのデフォルト設定は、IDE バージョンのアップグレードによって変更される場合があります。最新の IDE を使用する場合は、セクション設定を確認してください。

## 2.4 サポートされているツールチェーン

本ドライバは、「6.1 動作確認環境」に示すツールチェーンで動作確認を行っています。

## 2.5　使用する割り込みベクタ

RXIn および ERIn 割り込みを有効にするには、R_RSCI_Open 関数を実行します（非同期モードの場合）。

SSPI および同期モードでは、TXIn および TEIn 割り込みは使用されません。

表2.1 に RSCI FIT モジュールで使用される割り込みベクタを示します。

表2.1　RSCI FIT モジュールで使用される割り込みベクタ

| デバイス | 割り込みベクタ |
|---|---|
| RX671 | RXI 割り込み（ベクタ番号：32）<br>TXI 割り込み（ベクタ番号：33）<br>RXI 割り込み（ベクタ番号：42）<br>TXI 割り込み（ベクタ番号：43）<br>GROUPAL0 割り込み（ベクタ番号：112）<br>TEI 割り込み（グループ割り込みソース番号：24）<br>ERI 割り込み（グループ割り込みソース番号：25）<br>TEI 割り込み（グループ割り込みソース番号：27）<br>ERI 割り込み（グループ割り込みソース番号：28） |

## 2.6　ヘッダファイル

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

## 2.7　整数型

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

## 2.8　コンパイル時の設定

本モジュールのコンフィグレーションオプションの設定は、r_rsci_rx_config.h に記述されています。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。

| r_rsci_rx_config.h のコンフィグレーションオプション（1/2） | |
|---|---|
| RSCI_CFG_PARAM_CHECKING_ENABLE　1 | 1：パラメータチェック処理がビルドに含まれます。<br>0：パラメータチェック処理がビルドから省略されます。<br>この#define を BSP_CFG_PARAM_CHECKING_ENABLE に設定すると、システムデフォルト設定を使用します。 |
| RSCI_CFG_ASYNC_INCLUDED　1<br>RSCI_CFG_SYNC_INCLUDED　0<br>RSCI_CFG_SSPI_INCLUDED　0 | これらの#define は、各動作モード固有のコードをインクルードするために使用されます。1 の値を設定すると、サポートコードがインクルードされます。未使用のモードに 0 の値を設定すると、全体のコードサイズを縮小できます。 |
| RSCI_CFG_DUMMY_TX_BYTE　0xFF | この#define は、SSPI および同期モードでのみ使用されます。R_RSCI_Receive()関数の呼び出し中にクロック入力される各バイトに対してクロック出力されるダミーデータの値です。 |
| RSCI_CFG_CH10_INCLUDED　0<br>RSCI_CFG_CH11_INCLUDED　0 | 各チャネルには送受信バッファ、カウンタ、割り込み、その他のプログラムおよび RAM リソースが関連付けられています。#define を 1 に設定すると、当該チャネルにリソースが割り当てられます。<br>構成ファイルで使用する予定のチャネルを必ず有効にしてください。 |
| RSCI_CFG_CH10_TX_BUFSIZ　80<br>RSCI_CFG_CH11_TX_BUFSIZ　80 | これらの#define は、各チャネルで送信キューに使用するバッファのサイズを指定します（非同期モード）。<br>対応する RSCI_CFG_CHn_INCLUDED を 0 に設定するか、RSCI_CFG_ASYNC_INCLUDED を 0 に設定すると、バッファは割り当てられません。 |
| RSCI_CFG_CH10_RX_BUFSIZ　80<br>RSCI_CFG_CH11_RX_BUFSIZ　80 | これらの#define は、各チャネルで受信キューに使用するバッファのサイズを指定します（非同期モード）。<br>対応する RSCI_CFG_CHn_INCLUDED を 0 に設定するか、RSCI_CFG_ASYNC_INCLUDED を 0 に設定すると、バッファは割り当てられません。 |
| RSCI_CFG_TEI_INCLUDED　0 | この#define を 1 に設定すると、送信バッファエンプティ割り込みコードがインクルードされます。この割り込みは、データの最終バイトの最下位ビットが送信済みになると発生します。この割り込みは、ユーザのコールバック関数（R_RSCI_Open()で指定される）を呼び出し、それを RSCI_EVT_TEI イベントに受け渡します。 |
| RSCI_CFG_ERI_TEI_PRIORITY　3 | 受信エラー割り込み（ERI）および送信完了割り込み（TEI）の優先レベルを設定します。優先度最低が 1、最高が 15 です。ERI 割り込みは、全チャネルのオーバラン、フレーミング、パリティエラーを処理します。<br>TEI 割り込みは、最下位ビットが送信済みになり、トランスミッタがアイドル状態になるタイミングを示します（非同期モード）。 |

| r_rsci_rx_config.h のコンフィグレーションオプション（2/2） | |
|---|---|
| RSCI_CFG_CH10_FIFO_INCLUDED　0<br>RSCI_CFG_CH11_FIFO_INCLUDED　0 | 1：FIFO 関数に関する処理がビルドに含まれます。<br>0：FIFO 関数に関する処理がビルドから省略されます。 |
| RSCI_CFG_CH10_TX_FIFO_THRESH　8<br>RSCI_CFG_CH11_TX_FIFO_THRESH　8 | RSCI 動作モードがクロック同期モードまたは簡易 SPI モードの場合、受信 FIFO 閾値と同じ値を設定します。<br>0～31：送信 FIFO の閾値を指定します。 |
| RSCI_CFG_CH10_RX_FIFO_THRESH　8<br>RSCI_CFG_CH11_RX_FIFO_THRESH　8 | 1～31：受信 FIFO の閾値を指定します。 |
| RSCI_CFG_CH10_DATA_MATCH_INCLUDED　0<br>RSCI_CFG_CH11_DATA_MATCH_INCLUDED　0 | 1：データ照合関数に関する処理がビルドに含まれます。<br>0：データ照合関数に関する処理がビルドから省略されます。 |
| RSCI_CFG_CH10_TX_SIGNAL_TRANSITION_TIMING_INCLUDED　0<br>RSCI_CFG_CH11_TX_SIGNAL_TRANSITION_TIMING_INCLUDED　0 | 送信信号遷移タイミング調整機能を有効/無効にします。<br>有効 = 1、無効 = 0 |
| RSCI_CFG_CH10_RX_DATA_SAMPLING_TIMING_INCLUDED　0<br>RSCI_CFG_CH11_RX_DATA_SAMPLING_TIMING_INCLUDED　0 | 受信データサンプリングタイミング調整機能を有効/無効にします。<br>有効 = 1、無効 = 0 |

## 2.9　コードサイズ

本モジュールに関連する一般的な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、ビルド時のコンフィグレーションオプション（「2.8 コンパイル時の設定」を参照）によって決まります。表中の値は、C コンパイラのコンパイルオプションが既定値に設定されるときの参照値です（「2.4 サポートされているツールチェーン）。コンパイルオプションの既定値は、最適化レベル：2、最適化タイプ：サイズに合わせる、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とコンパイルオプションに応じて異なります。

| ROM および RAM 最小サイズ（バイト） | | | | | |
|---|---|---|---|---|---|
| | | | メモリ使用量 | | |
| | | | Renesas Compiler | | |
| デバイス | 分類 | | パラメータチェックあり | パラメータチェックなし | 説明 |
| RX671 | 非同期モード | ROM | 3472 バイト | 3122 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 |
| | クロック同期モード | ROM | 2990 バイト | 2596 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 |
| | 非同期モード + クロック同期モード（または簡易 SPI） | ROM | 4550 バイト | 4070 バイト | 合計 2 チャネル使用 |
| | | RAM | 392 バイト | 392 バイト | 合計 2 チャネル使用 |
| | 最大スタック使用量 | | 68 バイト | | |
| | FIFO モード + 非同期モード | ROM | 4372 バイト | 3917 バイト | 1 channel used |
| | | RAM | 200 バイト | 200 バイト | 1 channel used |
| | FIFO モード + クロック同期モード | ROM | 4024 バイト | 3571 バイト | 1 channel used |
| | | RAM | 44 バイト | 44 バイト | 1 channel used |
| | FIFO モード + 非同期モード + クロック同期モード | ROM | 5902 バイト | 5362 バイト | Total 2 channels used |
| | | RAM | 408 バイト | 408 バイト | Total 2 channels used |
| | 最大スタック使用量 | | 68 バイト | | |

| ROM および RAM 最小サイズ（バイト） | | | | | |
|---|---|---|---|---|---|
| デバイス | 通信方式 | | メモリ使用量 GCC パラメータチェックあり | メモリ使用量 GCC パラメータチェックなし | 説明 |
| RX671 | 非同期モード | ROM | 6704 バイト | 6016 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 |
| | クロック同期モード | ROM | 5604 バイト | 4883 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 |
| | 非同期モード + クロック同期モード（または簡易 SPI） | ROM | 8892 バイト | 7916 バイト | 合計 2 チャネル使用 |
| | | RAM | 392 バイト | 392 バイト | 合計 2 チャネル使用 |
| | 最大スタック使用量 | | - | | |
| | FIFO モード + 非同期モード | ROM | 8408 バイト | 7624 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 + クロック同期モード | ROM | 7636 バイト | 6756 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 + 非同期モード + クロック同期モード | ROM | 11516 バイト | 10420 バイト | 合計 2 チャネル使用 |
| | | RAM | 408 バイト | 408 バイト | 合計 2 チャネル使用 |
| | 最大スタック使用量 | | - | | |

| ROM および RAM 最小サイズ（バイト） | | | | | |
|---|---|---|---|---|---|
| | | | メモリ使用量 | | |
| | | | IAR Compiler | | |
| デバイス | 分類 | | パラメータチェックあり | パラメータチェックなし | 説明 |
| RX671 | 非同期モード | ROM | 5494 バイト | 4874 バイト | 1 チャネル使用 |
| | | RAM | 581 バイト | 581 バイト | 1 チャネル使用 |
| | クロック同期モード | ROM | 4404 バイト | 3793 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 |
| | 非同期モード + クロック同期モード（または簡易 SPI） | ROM | 7010 バイト | 6154 バイト | 合計 2 チャネル使用 |
| | | RAM | 781 バイト | 781 バイト | 合計 2 チャネル使用 |
| | 最大スタック使用量 | | 152 バイト | | |
| | FIFO モード + 非同期モード | ROM | 6751 バイト | 6034 バイト | 1 チャネル使用 |
| | | RAM | 589 バイト | 589 バイト | 1 チャネル使用 |
| | FIFO モード + クロック同期モード | ROM | 5905 バイト | 5173 バイト | 1 チャネル使用 |
| | | RAM | 48 バイト | 48 バイト | 1 チャネル使用 |
| | FIFO モード + 非同期モード + クロック同期モード | ROM | 8897 バイト | 7924 バイト | 合計 2 チャネル使用 |
| | | RAM | 797 バイト | 797 バイト | 合計 2 チャネル使用 |
| | 最大スタック使用量 | | 224 バイト | | |

RAM 要件は、構成したチャネル数に応じて異なります。各チャネルには、RAM 内の関連データ構造体が含まれます。また、非同期モードの場合、各非同期チャネルには送信キューおよび受信キューが含まれます。各キューのバッファは、最小サイズの 2 バイトか、チャネルごとに合計 4 バイトです。キューのバッファサイズはユーザが構成可能であるため、RAM 要件はバッファに割り当てられたサイズに直接連動して増減します。

非同期モードの RAM 要件の計算式は以下のとおりです。

使用チャネル数（1～2） × （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（RSCI_CFG_CHn_TX_BUFSIZ によって指定されるサイズ）

+ 受信キューのバッファサイズ（RSCI_CFG_CHn_RX_BUFSIZ によって指定されるサイズ））

* FIFO モードの場合、チャネルごとのデータ構造体は 36 バイトです。

同期および SPI モードの RAM 要件は、チャネル数 × チャネルごとのデータ構造体（固定 36 バイト、FIFO モードの場合は固定 40 バイト）です。

ROM 要件は、使用のため構成したチャネル数に応じて異なります。正確なサイズは、選択したチャネルの組み合わせとコンパイラコード最適化の影響に応じて異なります。

## 2.10 パラメータ

本モジュールで API 関数によって使用されるパラメータ構造体について説明します。パラメータ構造体は、API 関数のプロトタイプ宣言と同様に、r_rsci_rx_if.h に記述されます。

チャネル管理用構造体

この構造体は、RSCI チャネルの制御に必要な管理情報を保存するためのものです。構造体の内容は、コンフィグレーションオプションおよび使用デバイスの設定に応じて異なります。ユーザは構造体の内容に注意を払う必要はありませんが、クロック同期モード/SSPI モードを使用する場合は、tx_cnt または rx_cnt を用いて処理されるデータの数をチェックできます。

以下は RX671 の構造体の例です。

```
typedef struct st_rsci_ch_ctrl    // チャネル管理構造体
{
rsci_ch_rom_t const *rom;         // チャネルの RSCI レジスタの開始アドレス
rsci_mode_t mode;                 // 現在チャネルに設定されている RSCI 動作モード
uint32_t baud_rate;               // 現在チャネルに設定されているボーレート
void (*callback)(void *p_args);   // コールバック関数のアドレス
union
{
#if (RSCI_CFG_ASYNC_INCLUDED)
byteq_hdl_t que;                  // 送信バイトキュー（非同期モード）
#endif
uint8_t *buf;                     // 送信バッファの開始アドレス
//（クロック同期/SSPI モード）
} u_tx_data;
union
{
#if (RSCI_CFG_ASYNC_INCLUDED)
byteq_hdl_t que;                  // 受信バイトキュー（非同期モード）
#endif
uint8_t *buf;                     // 受信バッファの開始アドレス
         //（同期/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 (RSCI_CFG_SSPI_INCLUDED || RSCI_CFG_SYNC_INCLUDED)
bool save_rx_data;                // 受信データ保存（有効/無効）
uint16_t tx_cnt;                  // 送信カウンタ
uint16_t rx_cnt;                  // 受信カウンタ
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 RSCI_CFG_FIFO_INCLUDED
uint8_t fifo_ctrl;                // FIFO 関数（有効/無効）
uint8_t rx_dflt_thresh;           // 受信 FIFO 閾値（既定値）
uint8_t rx_curr_thresh;           // 受信 FIFO 閾値（現在の値）
uint8_t tx_dflt_thresh;           // 送信 FIFO 閾値（既定値）
uint8_t tx_curr_thresh;           // 送信 FIFO 閾値（現在の値）
#endif
} rsci_ch_ctrl_t;
```

## 2.11 戻り値

API 関数の戻り値について説明します。この列挙型は、API 関数のプロトタイプ宣言と同様に、r_rsci_rx_if.h に記述されます。

```
typedef enum e_rsci_err     // RSCI API エラーコード
{
    RSCI_SUCCESS=0,
    RSCI_ERR_BAD_CHAN,      // 存在しないチャネル番号
    RSCI_ERR_OMITTED_CHAN,  // config.h で RSCI_CHx_INCLUDED が 0
    RSCI_ERR_CH_NOT_CLOSED, // チャネルがまだ別のモードで実行中
    RSCI_ERR_BAD_MODE,      // チャネルのモードがサポートされていないか、間違っている
    RSCI_ERR_INVALID_ARG,   // パラメータの引数が無効
    RSCI_ERR_NULL_PTR,      // null ptr を受信。必要な引数が見つからない
    RSCI_ERR_XCVR_BUSY,     // データ転送を開始不可。トランシーバがビジー状態

    // 非同期
    RSCI_ERR_QUEUE_UNAVAILABLE,    // tx または rx キューあるいはその両方を開けない
    RSCI_ERR_INSUFFICIENT_SPACE,   // 送信キュー内のスペースが足りない
    RSCI_ERR_INSUFFICIENT_DATA,    // 受信キュー内のスペースが足りない

    // 同期/SSPI モード専用
    RSCI_ERR_XFER_NOT_DONE  // データ転送がまだ進行中
} rsci_err_t;
```

## 2.12 コールバック関数

本モジュールでは、ユーザが指定したコールバック関数は、RXIn、ERIn 割り込みが発生すると呼び出されます。

コールバック関数は、“void (* const p_callback)(void *p_args)”構造体メンバにユーザ関数のアドレスを保存することで指定されます（「2.10 パラメータ」を参照）。コールバック関数が呼び出されると、定数を保存する変数が引数として受け渡されます。

引数は void 型として受け渡されます。次に、コールバック関数の引数は void 型ポインタにキャストされます。下の例を参照してください。

コールバック関数内の値を使用する場合は、その値を型キャスト（データ型を変換）してください。

以下は、非同期モードにおけるコールバック関数のテンプレート例を示しています。

```
void MyCallback(void *p_args)
{
rsci_cb_args_t *args;
args = (rsci_cb_args_t *)p_args;
if (args->event == RSCI_EVT_RX_CHAR)
{
// RXI 割り込みから。キュー内に配置される文字の形式は args->byte
nop();
}
else if (args->event == RSCI_EVT_RX_CHAR_MATCH)
{
// RXI 割り込みから。受信データが比較データに一致
// キュー内に配置される文字の形式は args->byte
nop();
}

#if RSCI_CFG_TEI_INCLUDED
else if (args->event == RSCI_EVT_TEI)
{
// TEI 割り込みから。トランスミッタがアイドル状態
// ここで外部トランシーバを無効にできる
nop();
}
#endif
else if (args->event == RSCI_EVT_RXBUF_OVFL)
{
// RXI 割り込みから。受信キューが満杯
// 未保存の文字の形式は args->byte
// バッファサイズを増やすか、ボーレートを下げる必要がある
nop();
}
else if (args->event == RSCI_EVT_OVFL_ERR)
{
// ERI 割り込みから。レシーバオーバフローエラー発生
// エラー文字の形式は args->byte
// ERI ルーチン内でエラー条件がクリアされる
nop();
}
else if (args->event == RSCI_EVT_FRAMING_ERR)
{
// ERI 割り込みから。レシーバフレーミングエラー発生
```

```
// エラー文字の形式は args->byte。= 0 の場合、受信した BREAK 条件
// ERI ルーチン内でエラー条件がクリアされる
nop();
}
else if (args->event == RSCI_EVT_PARITY_ERR)
{
// ERI 割り込みから。レシーバパリティエラー発生
// エラー文字の形式は args->byte
// ERI ルーチン内でエラー条件がクリアされる
nop();
}
}
```

以下は、SSPI モードにおけるコールバック関数のテンプレート例を示しています。

```
void sspiCallback(void *p_args)
{
rsci_cb_args_t *args;
args = (rsci_cb_args_t *)p_args;
if (args->event == RSCI_EVT_XFER_DONE)
{
// データ転送完了
nop();
}
else if (args->event == RSCI_EVT_XFER_ABORTED)
{
// データ転送中断
nop();
}
else if (args->event == RSCI_EVT_OVFL_ERR)
{
// ERI 割り込みから。レシーバオーバフローエラー発生
// エラー文字の形式は args->byte
// ERI 割り込みルーチン内でエラー条件がクリアされる
nop();
}
}
```

本 FIT モジュールがユーザによって指定されたコールバック関数を呼び出すのは、受信エラー割り込みが発生するとき、非同期モードで 1 バイトデータが受信されるとき、クロック同期または SSPI モードで指定バイト数の送受信が完了したとき、送信完了割り込みが発生するときです。

非同期モードで FIFO 関数を有効にすると、RSCI_CFG_CHn_RX_FIFO_THRESH を用いて指定した最大回数の受信が完了したとき、あるいは最終受信データのストップビットから 15 etu*1 が経過したときにコールバック関数が実行されます。

コールバック関数を設定するには、コールバック関数のアドレスを R_RSCI_Open()の第 4 パラメータに指定します。コールバック関数が呼び出されると、以下のパラメータが設定されます。

```
typedef struct st_rsci_cb_args   // コールバック関数の引数
{
rsci_hdl_t hdl;                  // イベント発生時のハンドル
rsci_cb_evt_t event;             // 発生イベントをトリガしたイベント
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（FIFO の使用時にのみ有効）
```

```
} rsci_cb_args_t;

typedef enum e_rsci_cb_evt          // コールバック関数のイベント
{
// 非同期用イベント
RSCI_EVT_TEI,                       // TEI 割り込み発生。
RSCI_EVT_RX_CHAR,                   // 文字を受信。キュー内に配置。
RSCI_EVT_RX_CHAR_MATCH              // 受信データ照合。既にキュー内に配置。
RSCI_EVT_RXBUF_OVFL,                // 受信キューが満杯。これ以上データの保存不可。
RSCI_EVT_FRAMING_ERR,               // レシーバでフレーミングエラー発生。
// SSPI/クロック同期モード用イベント
RSCI_EVT_PARITY_ERR,                // レシーバでパリティエラー発生。
// SSPI/クロック同期モード用イベント
RSCI_EVT_XFER_DONE,                 // 転送完了。
RSCI_EVT_XFER_ABORTED,              // 転送取り消し。
// 共通イベント
RSCI_EVT_OVFL_ERR                   // 受信デバイスでオーバランエラーが発生
} rsci_cb_evt_t;
```

引数は void 型ポインタとして受け渡されるため、コールバック関数の引数は、void 型のポインタ変数でなければなりません。例えば、コールバック関数内の引数値を使用する場合、型キャストする必要があります。

【注】 1. etu（Elementary Time Unit）：1 ビットの転送にかかる時間

以下のイベントが発生すると、コールバック関数の引数に保存された受信データは未定義の値になります。

- RSCI_EVT_TEI
- RSCI_EVT_XFER_DONE
- RSCI_EVT_XFER_ABORTED
- RSCI_EVT_OVFL_ERR（FIFO 関数が有効なとき）
- RSCI_EVT_PARITY_ERR（FIFO 関数が有効なとき）
- RSCI_EVT_FRAMING_ERR（FIFO 関数が有効なとき）

## 2.13 FIT モジュールの追加方法

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

(1) 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。

## 2.14 for 文、while 文、do while 文について

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

以下に記述例を示します。

while 文の例：

```
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* PLL が安定したことを確認するために遅延時間が必要。 */
}
```

for 文の例：

```
/* 基準カウンタを 0 に初期化。 */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}
```

do while 文の例：

```
/* リセット完了待機中 */
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_RSCI_Open()

RSCI チャネルへの出力適用、関連レジスタの初期化、割り込みの有効化、他の API 関数で使用するためのチャネルハンドルの提供を行う関数です。この関数は、他のすべての API 関数を呼び出す前に呼び出す必要があります。

### Format

```
rsci_err_t      R_RSCI_Open (
                uint8_t   const           chan,
                          rsci_mode_t   const      mode,
                rsci_cfg_t * const      p_cfg,
                          void                (* const p_callback)(void *p_args),
                          rsci_hdl_t * const    p_hdl
)
```

### Parameters

*uint8_t　const chan*

初期化するチャネル

*rsci_mode_t　const mode*

動作モード（下の列挙型を参照）

*rsci_cfg_t * const p_cfg*

構成共用体へのポインタ。構造体要素（下記参照）はモード固有です。

*p_callback*

RXI またはレシーバエラーが検出されるか、送信終了（TEI）条件であるときに割り込みから呼び出される関数へのポインタ

詳細は「2.12 コールバック関数」を参照してください。

*rsci_hdl_t * const p_hdl*

チャネル用ハンドルへのポインタ（ここで値を設定）

R_RSCI_Open からの戻り値が“RSCI_SUCCESS”であることを確認してから、R_RSCI_GetVersion() を除く他の API に第 1 パラメータを設定します。「2.10 パラメータ」を参照してください。

現在、本ドライバモジュールでは以下の RSCI モードがサポートされています。指定したモードが、p_cfg parameter に使用される共用体/構造体要素を決定します。

```
typedef enum e_rsci_mode    // RSCI 動作モード
{
    RSCI_MODE_OFF=0,        // 未使用のチャネル
    RSCI_MODE_ASYNC,        // 非同期
    RSCI_MODE_SSPI,         // 簡易 SPI
    RSCI_MODE_SYNC,         // 同期
    RSCI_MODE_MAX           // 現在サポートされているモードの終了
} rsci_mode_t;
```

以降に記載する#define は、非同期モードの場合に構成構造体で使用される構成可能なオプションを示しています。各値は SMR レジスタ内のビット定義に対応し、データ長、パリティ関数、ストップビットを指定します。BRR レジスタおよび SEMR レジスタは、sci_uart_t 構造体の clk_src を用いて指定したクロックソース（内部/外部クロックの 8 倍/16 倍）および sci_uart_t 構造体の baud_rate を用いて指定したビットレートを使用して設定されます。なお、この設定は指定したビットレートを保証しないので注意してください（設定に応じて一部のエラーが発生する可能性があります）。また、FIFO 機能を用いて同期モードまたは SSPI モードでチャネル 10 および 11 を使用する場合、PCLKA/8 より高速のビットレートを設定できなくなります。（例えば、PCLKA が 120MHz の場合、設定可能なビットレートは 15Mbps 以下となります。）

以下は p_cfg の共用体を示しています。

```
typedef union
{
    rsci_uart_t     async;
    rsci_sync_sspi_t sync;
    rsci_sync_sspi_t sspi;
} rsci_cfg_t;
```

以下は、非同期モードでの設定に使用される構造体を示しています。

```
typedef struct st_rsci_uart
{
    uint32_t    baud_rate;     // 9600、19200、115200（内部クロックに有効）
    uint8_t     clk_src;       // RSCI_CLK_INT/EXT8/EXT16 を使用
    uint8_t     data_size;     // RSCI_DATA_nBIT を使用
    uint8_t     parity_en;     // RSCI_PARITY_ON/OFF を使用
    uint8_t     parity_type;   // RSCI_ODD/EVEN_PARITY を使用
    uint8_t     stop_bits;     // RSCI_STOPBITS_1/2 を使用
    uint8_t     int_priority;  // txi、tei、rxi、eri INT 優先度。1=低、15=高
} rsci_uart_t;
```

以下は、非同期モードで使用される構造体（rsci_uart_t）メンバの定義を示しています。

```
/* sck_src メンバの定義。 */
#define RSCI_CLK_INT        0x00  // ボーレート生成に内部クロックを使用
#define RSCI_CLK_EXT_8X     0x03  // 外部クロック 8 倍ボーレートを使用
#define RSCI_CLK_EXT_16X    0x02  // 外部クロック 16 倍ボーレートを使用

/* data_size メンバの定義。 */
#define RSCI_DATA_7BIT      0x30  // 7 ビット長
#define RSCI_DATA_8BIT      0x20  // 8 ビット長

/* parity_en メンバの定義。 */
#define RSCI_PARITY_ON      0x01  // パリティ ON
#define RSCI_PARITY_OFF     0x00  // パリティ OFF

/* parity_type メンバの定義。 */
#define RSCI_ODD_PARITY     0x01  // 奇数パリティ
#define RSCI_EVEN_PARITY    0x00  // 偶数パリティ
```

```
/* stop_bits メンバの定義。
#define RSCI_STOPBITS_2     0x01 // 2 ストップビット
#define RSCI_STOPBITS_1     0x00 // 1 ストップビット
```

以下は、SSPI および同期モードでの設定に使用される構造体を示しています。

```
typedef struct st_rsci_sync_sspi
{
    rsci_spi_mode_t  spi_mode;    // クロック極性および位相。同期には使用しない
    uint32_t       bit_rate;      // 1Mbps は 1000000
    bool          msb_first;
    bool          invert_data;
    uint8_t        int_priority;  // rxi、eri 割り込み優先度。1=低、15=高
} rsci_sync_sspi_t;
```

以下は、SSPI または同期モードで rsci_sync_sspi_t 構造体の spi_mode に使用される列挙型を示しています。

```
typedef enum e_rsci_spi_mode
{
    RSCI_SPI_MODE_OFF = 4,  // 同期モードで使用
    RSCI_SPI_MODE_0 = 0x01, // SCR3 レジスタ CPHA=1、CPOL=0
                            // モード 0： 00 CPOL=0 resting lo, CPHA=0 leading
edge/rising
    RSCI_SPI_MODE_1 = 0x02, // SPMR レジスタ CKPH=0、CKPOL=1
                            // モード 1： 01 CPOL=0 resting lo, CPHA=1 trailing
edge/falling
    RSCI_SPI_MODE_2 = 0x03, // SPMR レジスタ CKPH=1、CKPOL=1
                            // モード 2： 10 CPOL=1 resting hi, CPHA=0 leading
edge/falling
    RSCI_SPI_MODE_3 = 0x00  // SPMR レジスタ CKPH=0、CKPOL=0
                            // モード 3： 11 CPOL=1 resting hi, CPHA=1 trailing
edge/rising
} rsci_spi_mode_t;
```

### Return Values

```
[RSCI_SUCCESS]                /* 成功。チャネル初期化済み */
[RSCI_ERR_BAD_CHAN]           /* 一部のチャネル番号が無効 */
[RSCI_ERR_OMITTED_CHAN]       /* 対応する RSCI_CHx_INCLUDED が無効（0） */
[RSCI_ERR_CH_NOT_CLOSED]      /* チャネルが現在動作中。最初に R_RSCI_Close()を実行すること */
[RSCI_ERR_BAD_MODE]           /* 指定されたモードは現在サポートされていない */
[RSCI_ERR_NULL_PTR]           /* p_cfg ポインタが NULL*/
[RSCI_ERR_INVALID_ARG]        /* p_cfg 構造体の要素に無効な値が含まれる。 */
[RSCI_ERR_QUEUE_UNAVAILABLE] /* 送信/受信キューまたはその両方を開けない（非同期モード） */
```

### Properties

プロトタイプ宣言は“r_rsci_rx_if.h”ファイルに記述されています

### Description

特定のモード用に RSCI チャネルを初期化し、他の API 関数と併用するために*p_hdl* でハンドルを提供します。RXI および ERI 割り込みは、すべてのモードで有効です。TXI は非同期モードで有効です。

### Example: 非同期モード

```
    rsci_cfg_t   config;
    rsci_hdl_t   Console;
    rsci_err_t   err;

    config.async.baud_rate = 115200;
    config.async.clk_src = RSCI_CLK_INT;
    config.async.data_size = RSCI_DATA_8BIT;
    config.async.parity_en = RSCI_PARITY_OFF;
    config.async.parity_type = RSCI_EVEN_PARITY; // パリティが無効なので無視
    config.async.stop_bits = RSCI_STOPBITS_1;
    config.async.int_priority = 2;               // 1=最低、15=最高

    err = R_RSCI_Open(RSCI_CH10, RSCI_MODE_ASYNC, &config, MyCallback,
&Console);
```

### Example: SSPI モード

```
    rsci_cfg_t   config;
    rsci_hdl_t   sspiHandle;
    rsci_err_t   err;

    config.sspi.spi_mode = RSCI_SPI_MODE_0;
    config.sspi.bit_rate = 1000000;         // 1Mbps
    config.sspi.msb_first = true;
    config.sspi.invert_data = false;
    config.sspi.int_priority = 4;
    err = R_RSCI_Open(RSCI_CH10, RSCI_MODE_SSPI, &config, sspiCallback,
&sspiHandle);
```

### Example: 同期モード

```
    rsci_cfg_t   config;
    rsci_hdl_t   syncHandle;
    rsci_err_t   err;

    config.sync.spi_mode = RSCI_SPI_MODE_OFF;
    config.sync.bit_rate = 1000000;         // 1Mbps
    config.sync.msb_first = true;
    config.sync.invert_data = false;
    config.sync.int_priority = 4;
    err = R_RSCI_Open(RSCI_CH10, RSCI_MODE_SYNC, &config, syncCallback,
&syncHandle);
```

**Special Notes:**

本ドライバは、ボードサポートパッケージの mcu_info.h で定義される BSP_PCLKA_HZ および BSP_PCLKB_HZ を使用して、SCR2.BRR、SCR2.ABCS、SCR2.CKS に最適な値を計算します。ただし、これはすべての周辺クロック/ボーレートの組み合わせに対して、低ビットエラーレートを保証するものではありません。

外部クロックを非同期モードで使用する場合、R_RSCI_Open()関数を呼び出す前に端子方向を選択する必要があり、R_RSCI_Open()関数を呼び出した後に端子機能とモードを選択する必要があります。以下は、RX671 チャネル 10 の初期化例です。

```
Before the R_RSCI_Open() function call

   PORT8.PDR.BIT.B0 = 0;      // SCK010 端子方向を入力（dflt）に設定

After the R_RSCI_Open() function call

   MPC.P80PFS.BYTE = 0x2C ;   // 端子機能選択 P80 SCK010
   PORT1.PMR.BIT.B7 = 1;      // SCK 端子モードを周辺機能に設定
```

通信に使用する端子を設定する場合、R_RSCI_Open()関数を呼び出す前に端子方向とその出力を選択し、R_RSCI_Open()関数を呼び出した後に端子機能とモードを選択する必要があります。
RX671 で SSPI 用にチャネル 10 を初期化する例を以下に示します。

```
Before the R_RSCI_Open() function call

   PORT8.PODR.BIT.B2 = 0;     // ラインを低に設定
   PORT8.PODR.BIT.B0 = 0;     // ラインを低に設定
   PORT8.PDR.BIT.B0 = 1;      // クロック端子方向を出力に設定
   PORT8.PDR.BIT.B2 = 1;      // MOSI 端子方向を出力に設定
   PORT8.PDR.BIT.B1 = 0;      // MISO 端子方向を入力に設定

After the R_RSCI_Open() function call

   MPC.P82PFS.BYTE = 0x2C ;   // 端子機能選択 P82 MOSI
   MPC.P81PFS.BYTE = 0x2C ;   // 端子機能選択 P81 MISO
   MPC.P80PFS.BYTE = 0x2C ;   // 端子機能選択 P80  SCK010
   PORT8.PMR.BIT.B2 = 1;      // MOSI 端子モードを周辺機能に設定
   PORT8.PMR.BIT.B1 = 1;      // MISO 端子モードを周辺機能に設定
   PORT8.PMR.BIT.B0 = 1;      // クロック端子モードを周辺機能に設定
```

非同期モードの使用時は、1 つのチャネルに対して 2 つのバイトキューが使用されます。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「RX ファミリ 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_RSCI_Close()

RSCI チャネルから出力を除去し、関連する割り込みを無効にします。

### Format

```
rsci_err_t      R_RSCI_Close (
                rsci_hdl_t const     hdl
)
```

### Parameters

*rsci_hdl_t const hdl*

チャネル用ハンドル

R_RSCI_Open()が正常に処理されたら、*hdl* を設定してください

### Return Values

```
[RSCI_SUCCESS]          /* 成功。チャネル閉鎖 */
[RSCI_ERR_NULL_PTR]     /* hdl が NULL */
```

### Properties

プロトタイプ宣言は“r_rsci_rx_if.h”ファイルに記述されています

### Description

ハンドルによって指定された RSCI チャネルを無効にし、モジュール停止状態に入ります。

### Example

```
rsci_hdl_t   Console;
  ...
err = R_RSCI_Open(RSCI_CH10, RSCI_MODE_ASYNC, &config, MyCallback, &Console);
  ...
err = R_RSCI_Close(Console);
```

### Special Notes:

この関数を実行すると、進行中の送信または受信は中断します。

## R_RSCI_Send()

トランスミッタを使用しない場合に送信を開始します。非同期モード時に今後の送信用にデータをキューに登録します。

### Format

```
rsci_err_t      R_RSCI_Send (
                rsci_hdl_t const      hdl,
                        uint8_t                 *p_src,
                        uint16_t const          length
)
```

### Parameters

*rsci_hdl_t const hdl*

チャネル用ハンドル

R_RSCI_Open()が正常に処理されたら、*hdl* を設定してください

*uint8_t* p_src*

送信するデータへのポインタ

*uint16_t const length*

送信用バイトの数

### Return Values

```
[RSCI_SUCCESS]                  /* 送信開始またはキューに読み込み済み（非同期） */
[RSCI_ERR_NULL_PTR]             /* hdl 値が NULL */
[RSCI_ERR_BAD_MODE]             /* 指定されたモードは現在サポートされていない */
[RSCI_ERR_INSUFFICIENT_SPACE]       /* 全データを読み込むにはキュー内のスペースが足りない
                                        （非同期） */
[RSCI_ERR_XCVR_BUSY]            /* チャネルが現在ビジー状態（SSPI/同期） */
```

### Properties

プロトタイプ宣言は“r_rsci_rx_if.h”ファイルに記述されています

### Description

非同期モードでは、ハンドルによって参照される RSCI チャネル用トランスミッタが使用されていない場合、この関数がデータを送信キューに登録します。SSPI および同期モードでは、トランシーバがまだ使用されていない場合にデータはキューに登録されず、送信は直ちに開始されます。

なお、SSPI モード時のスレーブ選択ラインの切り換えは、本ドライバによって処理されません。対象デバイスのスレーブ選択ラインは、この関数を呼び出す前に有効にしておく必要があります。

また、同期/非同期モードでの CTS/RTS 端子の切り換えも本ドライバによって処理されません。

**Example: 非同期モード**

```
   #define STR_CMD_PROMPT "Enter Command: "
    rsci_hdl_t Console;
    rsci_err_t err;

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

    // この転送の完了をブロック不可。ただし、TEI 割り込みを使用して、
    // 送信するデータをこれ以上キューに残さない時期を決定することは可能。
```

**Example: SSPI モード**

```
    rsci_hdl_t  sspiHandle;
    rsci_err_t  err;
    uint8_t    flash_cmd,sspi_buf[10];

    // コマンドをフラッシュデバイスへ送信して ID を提供 */
    FLASH_SS = SS_ON;       // gpio フラッシュスレーブ選択を有効化
    flash_cmd = SF_CMD_READ_ID;

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

    /* フラッシュデバイスから ID 読み取り */
    R_RSCI_Receive(sspiHandle, sspi_buf, 5);
    while (RSCI_SUCCESS != R_RSCI_Control(sspiHandle, RSCI_CMD_CHECK_XFER_DONE,
NULL))
    {
    }

    FLASH_SS = SS_OFF;         // gpio フラッシュスレーブ選択を無効化
```

**Example: 同期モード**

```
    #define STRING1 "Test String"
    rsci_hdl_t  lcdHandle;
    rsci_err_t  err;


    // 文字列を LCD ディスプレイへ送信して完了まで待機 */
    R_RSCI_Send(lcdHandle, STRING1, sizeof(STRING1));

    while (RSCI_SUCCESS != R_RSCI_Control(lcdHandle, RSCI_CMD_CHECK_XFER_DONE,
NULL))
    {
    }
```

**Special Notes:**

なし。

## R_RSCI_Receive()

非同期モードの場合、RXI 割り込みによって満杯になったキューからデータを取得します。他のモードでは、トランシーバを使用しない場合に受信を開始します。

### Format

```
rsci_err_t      R_RSCI_Receive (
                rsci_hdl_t const     hdl,
                        uint8_t                 *p_dst,
                        uint16_t const          length
)
```

### Parameters

*rsci_hdl_t const hdl*

チャネル用ハンドル

R_RSCI_Open()が正常に処理されたら、*hdl* を設定してください

*uint8_t* p_dst*

データ読み込み先のバッファへのポインタ

*uint16_t const length*

読み取り用バイトの数

### Return Values

| | |
|---|---|
| [RSCI_SUCCESS] | /* 要求されたバイト数が p_dst に読み込まれた（非同期）データのクロック入力が開始された（SSPI/同期） |
| [RSCI_ERR_NULL_PTR] | /* hdl 値が NULL |
| [RSCI_ERR_BAD_MODE] | /* 指定されたモードは現在サポートされていない |
| [RSCI_ERR_INSUFFICIENT_DATA] | /* 全データを取得するには受信キュー内のデータが足りない（非同期） |
| [RSCI_ERR_XCVR_BUSY] | /* チャネルが現在ビジー状態（SSPI/同期） |

### Properties

プロトタイプ宣言は“r_rsci_rx_if.h”ファイルに記述されています

### Description

非同期モードでは、この関数は、ハンドルによって参照される RSCI チャネルで受信したデータをその受信キューから取得します。要求されたバイト数が足りない場合、この関数はブロックしません。SSPI/同期モードでは、トランシーバがまだ使用されていない場合、データのクロック入力は直ちに開始されます。r_rsci_config.h で RSCI_CFG_DUMMY_TX_BYTE に割り当てられる値は、受信データのクロック入力中にクロック出力されます。

受信中にエラーが発生すると、R_RSCI_Open()で指定したコールバック関数が実行されます。コールバック関数の引数を用いて受け渡されたイベントをチェックし、受信が正常に完了したか確認してください。詳細は「2.12 コールバック関数」を参照してください。

なお、SSPI モード時のスレーブ選択ラインの切り換えは、本ドライバによって処理されません。対象デバイスのスレーブ選択ラインは、この関数を呼び出す前に有効にしておく必要があります。

**Example: 非同期モード**

```
    rsci_hdl_t Console;
        rsci_err_t err;
    uint8_t   byte;

    /* echo 文字 */
    while (1)
    {
       while (RSCI_SUCCESS != R_RSCI_Receive(Console, &byte, 1))
       {
       }
       R_RSCI_Send(Console, &byte, 1);
    }
```

**Example: SSPI モード**

```
    rsci_hdl_t  sspiHandle;
        rsci_err_t  err;
    uint8_t    flash_cmd,sspi_buf[10];

    // コマンドをフラッシュデバイスへ送信して ID を提供 */

    FLASH_SS = SS_ON;       // gpio フラッシュスレーブ選択を有効化
    flash_cmd = SF_CMD_READ_ID;

    R_RSCI_Send(sspiHandle, &flash_cmd, 1);
    while (RSCI_SUCCESS != R_RSCI_Control(sspiHandle, RSCI_CMD_CHECK_XFER_DONE,
 NULL))
    {
    }

    /* フラッシュデバイスから ID 読み取り */
    R_RSCI_Receive(sspiHandle, sspi_buf, 5);
    while (RSCI_SUCCESS != R_RSCI_Control(sspiHandle, RSCI_CMD_CHECK_XFER_DONE,
 NULL))
    {
    }

    FLASH_SS = SS_OFF;         // gpio フラッシュスレーブ選択を無効化
```

**Example: 同期モード**

```
    rsci_hdl_t  sensorHandle;
        rsci_err_t  err;
    uint8_t    sensor_cmd,sync_buf[10];

    // コマンドをセンサへ送信して現在の読み取り値を提供 */

    sensor_cmd = SNS_CMD_READ_LEVEL;

    R_RSCI_Send(sensorHandle, &sensor_cmd, 1);
    while (RSCI_SUCCESS != R_RSCI_Control(sensorHandle,
 RSCI_CMD_CHECK_XFER_DONE, NULL))
    {
    }

    /* レベルをセンサから読み取り */
```

```
    R_RSCI_Receive(sensorHandle, sync_buf, 4);
    while (RSCI_SUCCESS != R_RSCI_Control(sensorHandle,
 RSCI_CMD_CHECK_XFER_DONE, NULL))
    {
    }
```

**Special Notes:**

コールバック関数の引数へ受け渡される値については、「2.12 コールバック関数」を参照してください。

非同期モードでは、データの一致が検出されると、受信データはキュー内に保存され、イベント RSCI_EVT_RX_CHAR_MATCH が発生し、コールバック関数によってユーザに通知します。

## R_RSCI_SendReceive()

同期および SSPI モード専用。トランシーバを使用しない場合にデータを同時に送受信します。

### Format

```
rsci_err_t      R_SCI_SendReceive (
                rsci_hdl_t const    hdl,
                uint8_t             *p_src,
                uint8_t             *p_dst,
                        uint16_t const      length
)
```

### Parameters

*rsci_hdl_t const hdl*

チャネル用ハンドル

R_RSCI_Open()が正常に処理されたら、*hdl* を設定してください

*uint8_t* p_src*

送信するデータへのポインタ

*uint8_t* p_dst*

データ読み込み先のバッファへのポインタ

*uint16_t const length*

送信用バイトの数

### Return Values

```
[RSCI_SUCCESS]              /* データ転送開始 */
[RSCI_ERR_NULL_PTR]         /* hdl 値が NULL */
[RSCI_ERR_BAD_MODE]         /* チャネルモードが SSPI または同期ではない */
[RSCI_ERR_XCVR_BUSY]        /* チャネルが現在ビジー状態 */
```

### Properties

プロトタイプ宣言は“r_rsci_rx_if.h”ファイルに記述されています

### Description

トランシーバを使用しない場合、この関数は p_src バッファからデータをクロック出力し、同時にデータをクロック入力して p_dst バッファに配置します。

なお、SSPI の場合にはスレーブ選択ラインの切り換えは、本ドライバによって処理されません。対象デバイスのスレーブ選択ラインは、この関数を呼び出す前に有効にしておく必要があります。

また、同期/非同期モードでの CTS/RTS 端子の切り換えも本ドライバによって処理されません。

**Example: SSPI モード**

```
    rsci_hdl_t  sspiHandle;
        rsci_err_t  err;
    uint8_t in_buf[2] = {0x55, 0x55};   // 不正な値を初期化


    /* API 呼び出しを 1 回実行してフラッシュステータスを読み取り */

    // クロック入力ステータス応答用に 1 ダミーバイトを追加送信するコマンドを実行して配列を読み
込む
    uint8_t out_buf[2] = {SF_CMD_READ_STATUS_REG, RSCI_CFG_DUMMY_TX_BYTE };

    FLASH_SS = SS_ON;

    err = R_RSCI_SendReceive(sspiHandle, out_buf, in_buf, 2);
    while (RSCI_SUCCESS != R_RSCI_Control(sspiHandle, RSCI_CMD_CHECK_XFER_DONE,
NULL))
    {
    }

    FLASH_SS = SS_OFF;

    // _buf[1]にステータスが格納されている
```

**Special Notes:**

コールバック関数の引数へ受け渡される値については、「2.12 コールバック関数」を参照してください。

## R_RSCI_Control()

RSCI チャネルの動作モードを構成し、制御します。

### Format

```
rsci_err_t      R_RSCI_Control (
                rsci_hdl_t const     hdl,
                        rsci_cmd_t const        cmd,
                        void                    *p_args
)
```

### Parameters

*rsci_hdl_t const hdl*

チャネル用ハンドル

R_RSCI_Open()が正常に処理されたら、*hdl* を設定してください

*rsci_cmd_t const cmd*

実行するコマンド（下の列挙型を参照）

*void *p_args*

コマンド固有の引数（下記参照）へのポインタ、void *へキャスト

有効な *cmd* 値は以下のとおりです。

```
typedef enum e_rsci_cmd // RSCI Control()コマンド
{
    // すべてのモード
    RSCI_CMD_CHANGE_BAUD,           // ボー/ビットレートを変更
    RSCI_CMD_CHANGE_TX_FIFO_THRESH,  // 送信 FIFO 閾値を変更
    RSCI_CMD_CHANGE_RX_FIFO_THRESH,  // 受信 FIFO 閾値を変更
    RSCI_CMD_SET_RXI_PRIORITY,      // 受信優先度（TXI および RXI に異なる優先レベルを
// 指定可能な MCU の場合）
    RSCI_CMD_SET_TXI_PRIORITY,      // 送信優先度（TXI および RXI に異なる優先レベルを
// 指定可能な MCU の場合）

    // 非同期コマンド
    RSCI_CMD_EN_NOISE_CANCEL,       // ノイズキャンセルを有効化
    RSCI_CMD_EN_TEI,                // このコマンドは無効
//  （旧バージョンとの互換性のために残す）
    RSCI_CMD_OUTPUT_BAUD_CLK,       // SCK 端子上の出力ボークロック
    RSCI_CMD_START_BIT_EDGE,        // RXDn 端子の立ち下がりエッジとして開始ビットを検出
                                    //  （デフォルトでは RXDn 端子で低レベルとして検出）
    RSCI_CMD_GENERATE_BREAK,        // ブレーク条件を生成
    RSCI_CMD_COMPARE_RECEIVED_DATA,  // 受信データと比較用データを比較

    // 非同期/同期コマンド
    RSCI_CMD_EN_CTS_IN,             // CTS 入力を有効化（デフォルトの RTS 出力）

    // SSPI/同期コマンド
    RSCI_CMD_CHECK_XFER_DONE,       // 送信、受信、またはその両方が完了したか確認。完了し
た場合は RSCI_SUCCESS
    RSCI_CMD_ABORT_XFER,            // 送信を中断
```

```
    RSCI_CMD_XFER_LSB_FIRST,        // LSB 優先に設定
    RSCI_CMD_XFER_MSB_FIRST,        // MSB 優先に設定
    RSCI_CMD_INVERT_DATA,           // クロック極性反転に設定

    // SSPI コマンド
    RSCI_CMD_CHANGE_SPI_MODE        // SPI モードを変更

} rsci_cmd_t;
```

以下のコマンド以外のコマンドには引数は不要です。p_args には FIT_NO_PTR を取得します。

RSCI_CMD_CHANGE_BAUD の引数は、使用する新しいビットレートを格納している rsci_baud_t 変数へのポインタです。rsci_baud_t 構造体を以下に示します。

```
typedef struct st_rsci_baud
{
    uint32_t    pclk;     // 周辺クロック速度。例：24000000 は 24MHz
    uint32_t    rate;     // 例：9600、19200、115200
} rsci_baud_t;
```

RSCI_CMD_TX_Q_BYTES_FREE および RSCI_CMD_RX_Q_BYTES_AVAIL_TO_READ の引数は、カウント値を保持する uint16_t 変数へのポインタです。

RSCI_CMD_CHANGE_SPI_MODE の引数は、使用する新しいモードを格納している列挙型（rsci_sync_sspi_t）変数へのポインタです。

RSCI_CMD_SET_TXI_PRIORITY および RSCI_CMD_SET_RXI_PRIORITY の引数（TXI および RXI に異なる優先レベルを指定可能な MCU の場合）は、優先レベルを保持する uint8_t 変数へのポインタです。

### Return Values

| | |
|---|---|
| [RSCI_SUCCESS] | /* 成功。チャネル初期化済み */ |
| [RSCI_ERR_NULL_PTR] | /* hdl または p_args ポインタは NULL（必要な場合） */ |
| [RSCI_ERR_BAD_MODE] | /* 指定されたモードは現在サポートされていない */ |
| [RSCI_ERR_INVALID_ARG] | /* cmd 値または p_args の要素に無効な値が含まれる。 */ |

### Properties

プロトタイプ宣言は“r_rsci_rx_if.h”ファイルに記述されています

### Description

この関数は、ドライバ構成の変更やドライバステータスの取得など、特別なハードウェア機能の構成に使用されます。

CTS/RTS 端子は、デフォルトのハードウェア制御によって RTS として機能します。RSCI_CMD_EN_CTS_IN を発行することで、端子は CTS として機能します。

### Example: 非同期モード

```
    rsci_hdl_t   Console;
    rsci_cfg_t   config;
    rsci_baud_t  baud;
    rsci_err_t   err;
    uint16_t    cnt;

    R_RSCI_Open(RSCI_CH10, RSCI_MODE_ASYNC, &config, MyCallback, &Console);
```

```
    R_RSCI_Control(Console, RSCI_CMD_EN_NOISE_CANCEL, NULL);
    R_RSCI_Control(Console, RSCI_CMD_EN_TEI, NULL);
      ...
    /* 低消費電力モードクロック切り換えによりボーレートをリセット */
    baud.pclk = 8000000;      // 8MHz
    baud.rate = 19200;
    R_RSCI_Control(Console, RSCI_CMD_CHANGE_BAUD, (void *)&baud);
      ...
    /* いくつかメッセージを送信後、tx キュー内に残っているスペースを判定 */
    R_RSCI_Control(Console, RSCI_CMD_TX_Q_BYTES_FREE, (void *)&cnt);
      ...
    /* 受信キューにデータが残っているか確認 */
    R_RSCI_Control(Console, RSCI_CMD_RX_Q_BYTES_AVAIL_TO_READ, (void *)&cnt);
```

**Example: SSPI モード**

```
    rsci_cfg_t   config;
    rsci_spi_mode_t   mode;
    rsci_hdl_t   sspiHandle;
    rsci_err_t   err;

    config.sspi.spi_mode    = RSCI_SPI_MODE_0;
    config.sspi.bit_rate    = 1000000;          // 1Mbps
    config.sspi.msb_first   = true;
    config.sspi.invert_data  = false;
    config.sspi.int_priority = 4;
    err = R_RSCI_Open(RSCI_CH10, RSCI_MODE_SSPI, &config, sspiCallback,
&sspiHandle);
      ...
      ...
    // 別モードで動作するスレーブデバイスへ変更する場合
    mode = RSCI_SPI_MODE_3;
    R_RSCI_Control(sspiHandle, RSCI_CMD_CHANGE_SPI_MODE, (void *)&mode);
```

**Special Notes:**

RSCI_CMD_CHANGE_BAUD が使用される場合、指定したビットレートに基づいて BRR、SEMR.ABCS、SMR.CKS の最適値が計算されます。ただし、これはすべての周辺クロック/ボーレートの組み合わせに対して、低ビットエラーレートを保証するものではありません。

RSCI_CMD_EN_CTS_IN コマンドを使用する場合、R_RSCI_Open()関数を呼び出す前に端子方向を選択する必要があり、R_RSCI_Open()関数を呼び出した後に端子機能とモードを選択する必要があります。以下は、RX671 チャネル 10 の初期化例です。

```
Before the R_RSCI_Open() function call

PORT1.PDR.BIT.B4 = 0;      // CTS/RTS 端子方向を入力（dflt）に設定

After the R_RSCI_Open() function call

MPC.PC4PFS.BYTE = 0x2C;      // 端子機能選択 PC4 CTS
PORTC.PMR.BIT.B4 = 1       // CTS/RTS 端子モードを周辺機能に設定
```

RSCI_CMD_OUTPUT_BAUD_CLK コマンドを使用する場合、R_RSCI_Open()関数を呼び出す前に端子方向を選択する必要があり、R_RSCI_Open()関数を呼び出した後に端子機能とモードを選択する必要があります。
以下は、RX671 チャネル 10 の初期化例です。

```
Before the R_RSCI_Open() function call

   PORT8.PDR.BIT.B0 = 1;        // SCK010 端子方向を出力に設定

After the R_RSCI_Open() function call

   MPC.P80PFS.BYTE = 0x2C;      // 端子機能選択 P80 SCK010
   PORT8.PMR.BIT.B0 = 1;        // SCK010 端子モードを周辺機能に設定
```

以下のコマンドは、送信中に実行できます。送信中は他のコマンドを実行しないでください。

- RSCI_CMD_TX_Q_BYTES_FREE
- RSCI_CMD_RX_Q_BYTES_AVAIL_TO_READ
- RSCI_CMD_CHECK_XFER_DONE
- RSCI_CMD_ABORT_XFER

この関数を実行すると、TXD 端子は一時的に Hi-Z になります。以下の方法のいずれかを使用すると、TXD 端子が Hi-Z になるのを防止できます。

RSCI_CMD_GENERATE_BREAK コマンドを使用するとき：

- TXD 端子をレジスタ（プルアップ）経由で Vcc に接続する。

上記以外のコマンドを使用するとき：

以下の方法のいずれかを実行してください。

- TXD 端子をレジスタ（プルアップ）経由で Vcc に接続する。
- RSCI_Control 関数を実行する前に、TXD 端子の端子機能を汎用入出力ポートに切り換え、RSCI_Control 関数の呼び出し後に周辺機能へ戻す。

## R_RSCI_GetVersion()

実行時にドライバのバージョン番号を返します。

### Format

uint32_t　R_RSCI_GetVersion (void)

### Parameters

なし

### Return Values

バージョン番号

### Properties

プロトタイプ宣言は“r_rsci_rx_if.h”ファイルに記述されています

### Description

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、上位 2 バイトがメジャーバージョン番号に、下位 2 バイトがマイナーバージョン番号となるように暗号化されます。

### Example

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

### Special Notes:

なし

# 4. 端子設定

RSCI FIT モジュールを使用するには、周辺機能の入出力信号をマルチファンクションピンコントローラ（MPC）の端子に割り当てます。本ドキュメントでは、端子の割り当てを「端子設定」と呼びます。

R_RSCI_Open 関数を呼び出した後に、端子設定を実行してください。

e$^2$ studio で端子設定を実行する場合、FIT Configurator または Smart Configurator の端子設定機能を使用できます。端子設定機能を使用すると、FIT Configurator または Smart Configurator の端子設定ウィンドウで選択したオプションに従って、ソースファイルが生成されます。次に、ソースファイルで定義される関数を呼び出すことで、端子は設定されます。詳細は「表4.1 FIT Configurator による関数出力」を参照してください。

表4.1　FIT Configurator による関数出力

| 使用 MCU | 出力される関数 | 説明 |
|---|---|---|
| すべての MCU | R_RSCI_PinSet_RSCIx | 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、main()関数が含まれます。この関数は、FIT モジュールとその依存モジュール（r_bsp など）を使用します。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 ワークスペースにデモを追加する

現在、サンプルプログラムはサポートされていません。

## 5.2 デモのダウンロード方法

現在、サンプルプログラムはサポートされていません。

# 6. 付録

## 6.1 動作確認環境

本 RSCI FIT モジュールの動作確認環境を以下に示します。

表6.1　動作確認環境（Rev.1.00）

| 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 V21.7.0<br>IAR Embedded Workbench for Renesas RX 4.20.3 |
| C コンパイラ | ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.03.00<br>コンパイラオプション：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。<br>-lang = c99 |
| | GCC for Renesas RX 8.3.0.202004<br>コンパイラオプション：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。<br>-std=gnu99<br>リンカオプション：“Optimize size (-Os)”を使用する場合は、以下のユーザ定義オプションを統合開発環境のデフォルト設定に追加してください。<br>-Wl,--no-gc-sections<br>リンカが誤って FIT 周辺モジュールで宣言された割り込み関数を破棄することによる GCC リンカ問題を解決します。 |
| | IAR C/C++ Compiler for Renesas RX version 4.20.3<br>コンパイラオプション：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.1.00 |
| 使用ボード | Renesas Starter Kit for RX671（型名：RTK55671xxxxxxxxxx） |

## 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”」エラーが発生しました。
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。アプリケーションノート「RX ファミリ ボードサポートパッケージモジュール Firmware Integration Technology（R01AN1685）」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_rsci_rx module」エラーが発生しました。
A：追加した FIT モジュールがユーザプロジェクトのターゲットデバイスに対応していない可能性があります。追加した FIT モジュールの対象デバイスを確認してください。

(3) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると、「ERROR - Unsupported channel chosen in r_rsci_config.h」エラーが発生しました。
A：“r_rsci_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_rsci_rx_config.h”ファイルを確認してください。設定が間違っている場合は、その設定に正しい値を設定してください。詳細は「2.8 コンパイル時の設定」を参照してください。

(4) Q：TXD 端子から送信データが出力されません。
A：端子設定が正しく実行さ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ときは、端子設定を実行する必要があります。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：開発ツール
　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

## 改訂記録

| Rev. | 発行日 | 改訂内容 | |
|---|---|---|---|
| | | ページ | ポイント |
| 1.00 | Mar.31.21 | — | 初版発行 |

# 製品ご使用上の注意事項

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

1. 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。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、お客様の責任において、お客様の機器・システムを設計ください。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. 当社製品または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. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 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。
6. 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。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 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。
9. 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。
10. 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。
11. 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。
12. 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。
13. 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。
14. 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.5.0-1　2020.10)

## 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。