RX ファミリ

RSPIA モジュール Firmware Integration Technology

要旨

本書では、RX ファミリ MCU 向けの RSPIA モジュールについて説明します。このモジュールは、Firmware Integration Technology（FIT）を採用しています。RSPIA ドライバのアーキテクチャ、FIT モジュールのユーザアプリケーションへの統合、API の使用方法について詳しく解説します。

RSPIA モジュールによってサポートされる RX ファミリ MCU には、単一チャネル用強化シリアルペリフェラルインタフェース（RSPIA）が内蔵されています。RSPIA は、全二重またはシンプレックス（送信専用または受信専用）の同期シリアル通信を実行します。複数のプロセッサおよび周辺機能とシリアル通信を実行する関数が実装されています。

動作確認デバイス

この API によって現在サポートされているデバイスは、以下のとおりです。

- 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。

関連ドキュメント

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

目次

1. 概要

このソフトウェアは、RSPIA 周辺機能の動作と、SPI バス経由でデータ転送を実行するための準備用に、アプリケーションプログラミングインタフェース（API）を提供します。

RSPIA FIT モジュールは、ユーザアプリケーションと物理ハードウェアを接続し、RSPIA 周辺機能を管理する低レベルハードウェア制御タスクを処理します。

このソフトウェアを使用する前に、RX MCU ユーザーズマニュアル：ハードウェア編の RSPIA 周辺機能に関する章を確認することを推奨します。

1.1 RSPIA FIT モジュール

RSPIA 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 RSPIA FIT モジュールの概要

RSPIA FIT モジュールをプロジェクトに追加したら、インストール用にソフトウェアを構成するために、*r_rspia_rx_config.h* ファイルを変更する必要があります。構成オプションに関する詳細は、「2.8 コンパイル時の設定」を参照してください。

RSPIA FIT モジュールには、入出力ポートのレジスタを初期化する関数は含まれていません。入出力ポートの設定は、本モジュール以外で行う必要があります。入出力ポートに関する詳細は、「4. 端子設定」を参照してください。

実行時に RSPIA チャネルを使用する場合、必要な設定とパラメータを受け渡して最初に R_RSPIA_Open()関数を呼び出します。完了したら、入出力ポートを設定すると、RSPIA チャネルがアクティブになり、この API で使用可能な他の関数をすべて実行する準備が整います。この時点で、SPI データ転送処理を使用したり、各種制御処理を実行して設定を変更したりすることができます*1。

【注】 *1 クロック同期動作（3 線式）で、マスターモードで使用する場合、下記の手順に従ってデータ送信を準備します。この準備を怠ると、クロックの同期ギャップが発生する可能性があります。
(1) 通信用スレーブを無効にします（RSPIA スレーブの場合、SPE=0 に設定）
(2) R_RSPIA_Open()を呼び出します。この処理が完了するまで待ってください
(3) 入出力ポート設定により端子を周辺機能モジュールに設定します
(4) 通信用スレーブを有効にします

RSPIA レジスタの設定は、R_RSPIA_Open()を呼び出すことで実行されます。使用目的が汎用であるため、レジスタの既定値は RSPIA レジスタで設定する必要があります。また、R_RSPIA_Control()を呼び出すと、RSPIA FIT モジュール内に保存される RSPIA レジスタ情報を書き換えることができます。

R_RSPIA_Control()関数には、以下の 5 つのコマンドが提供されています。

- 基本ビットクロックレートを変更する。
- 転送処理を直ちに中断する。
- RSPIA レジスタ情報を書き換える。
- 送信 FIFO 閾値を変更する。
- 受信 FIFO 閾値を変更する。

SPI バス経由でデータ転送が実行されると、本ドライバはユーザが提供するコールバック関数を呼び出して、ユーザのアプリケーションに完了ステータス情報を伝達します。

RSPIA API 関数のほとんどには、「ハンドル」引数が必要になります。この引数は、動作用に選択される RSPIA チャネル番号の識別に使用されます。

ハンドルを取得するには、最初に R_RSPIA_Open()関数を呼び出します。

ハンドルを R_RSPIA_Open()に保存する場所のアドレスを提供する必要があります。

取得が完了すると、そのハンドルを使用できるようになります。後は、当該 RSPIA チャネル番号に提供されたハンドル値を他の API 関数の呼び出し時にその関数へ受け渡すだけです。

使用するアプリケーションで、所定のチャネルにどのハンドルが属するかを追跡する必要があります。各チャネルには独自のハンドルが割り当てられるからです。

1.2.1　サポートされる機能

本ドライバは、RSPIA 周辺機能で使用できる以下の機能のサブセットをサポートします。

RSPIA 転送機能：

- MOSI（マスター出力/スレーブ入力）、MISO（マスター入力/スレーブ出力）、SSL（スレーブ選択）、RSPCK（RSPIA クロック）信号を使用すると、SPI 動作（4 線式）またはクロック同期動作（3 線式）によりシリアル通信が可能になります。
- 全二重または単向（送信のみ/受信のみ）通信を選択できます。
- マスター/スレーブモードでシリアル通信が可能
- シリアル転送クロックの極性の切り換え
- シリアル転送クロックの位相の切り換え

データ形式：

- MSB ファースト/LSB ファーストを選択可能
- 転送ビット長を 4～32 ビットの範囲で変更可能
- 送信バッファサイズ/受信バッファサイズ：32 ビット × 4 ステージ FIFO

ビットレート：

- マスターモードでは、内蔵ボーレートジェネレータが PCLK を分周して RSPCK を生成（分周比は 2～4096 分周）します。
- スレーブモードでは、外部入力クロックがシリアルクロックとして使用されます（最大周波数については、MCU ユーザーズマニュアルを参照）。

エラー検出：

- モード障害エラー検出
- オーバランエラー検出
- パリティエラー検出
- アンダーラン検出
- 受信データ準備完了検出

SSL 制御機能：

- 各チャネル用に 4 つの SSL 信号（SSL00～SSL03）
- シングルマスターモード：SSL00～SSL03 信号が出力されます。
- スレーブモード：入力用 SSLn0 信号および SSL01～SSL03 信号は Hi-Z（未使用）です。
- SSL 出力アサーションから RSPCK 動作（RSPCK 遅延）まで遅延を制御可能
  範囲：1～8 RSPCK サイクル（RSPCK サイクル単位で設定）
- RSPCK ストップから SSL/OE 出力ネゲーション（SSL/OE ネゲーション遅延）まで遅延を制御可能
  範囲：1～8 RSPCK サイクル（RSPCK サイクル単位で設定）
- 次のアクセス SSL 出力アサーション（次のアクセス遅延）用の待機時間を制御可能
  範囲：1～8 RSPCK サイクル（RSPCK サイクル単位で設定）
- SSL 極性を変更可能

通信プロトコル：
- Motorola SPI
- TI SSP（Synchronous Serial Protocol）

マスター転送における制御：
- 各転送処理では、スレーブ選択番号、基本ビットレートの追加分割、SPI クロック極性/位相、転送データビット長、MSB/LSB ファースト、バースト（SSL 保持）、SPI クロック遅延、スレーブ選択ネゲーション遅延、次のアクセス遅延を設定できます。

1.2.2　サポートされない機能
- 小型メモリ MCU の限られた RAM リソースを節約するために、本ドライバでは、データバッファが本ドライバによって静的に割り当てられるのではなく、上位レベルでユーザアプリケーションによって割り当てられることを要求します。これにより、アプリケーションは RAM 割り当て方法を制御できます。
- シングルシーケンスデータ転送のみがサポートされます。本ドライバによって、RSPIA 周辺機能のマルチコマンドシーケンスデータ転送機能はサポートされません。
- 16 ビットタイプのバイトスワップはサポートされません。
- 本モジュールは、DMAC（ダイレクトメモリアクセスコントローラ）および DTC（データトランスファコントローラ）と連動して使用することはできません。

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。本モジュールによって使用されるコードセクションのサイズについては、「2.9 コードサイズ」の表も参照してください。

表 1.1　API 関数

関数	関数説明
R_RSPIA_Open()	使用するために指定した RSPIA チャネルを準備するために必要な関連レジスタを初期化し、他の API 関数で使用するハンドルを提供します。割り込みイベントへ応答するために、コールバック関数を実行します。
R_RSPIA_Control()	RSPIA チャネル用の特別ハードウェア/ソフトウェア動作を処理します
R_RSPIA_Read()	Read 関数は、SPI マスターまたはスレーブデバイスからデータを受信します。
R_RSPIA_Write()	Write 関数は、SPI マスターまたはスレーブデバイスにデータを送信します。
R_RSPIA_WriteRead()	Write Read 関数は、SPI マスターまたはスレーブデバイスへのデータ送信と、そのデバイスからのデータ受信を同時に実行します（全二重）。
R_RSPIA_Close()	指定した RSPIA チャネルを無効にします。
R_RSPIA_GetVersion()	ドライバのバージョン番号を返します。

1.4　ドライバアーキテクチャ

1.4.1　システム例

本ドライバは、シングルマスター/マルチスレーブモード動作、またはスレーブモード動作をサポートします。各 RSPIA チャネルは 1 つの SPI バスを制御します。本ドライバでは、同じバスでのマルチマスター動作はサポートされません。下の図は、1 つの SPI バスで複数のスレーブに接続されたシングルマスターの例です。

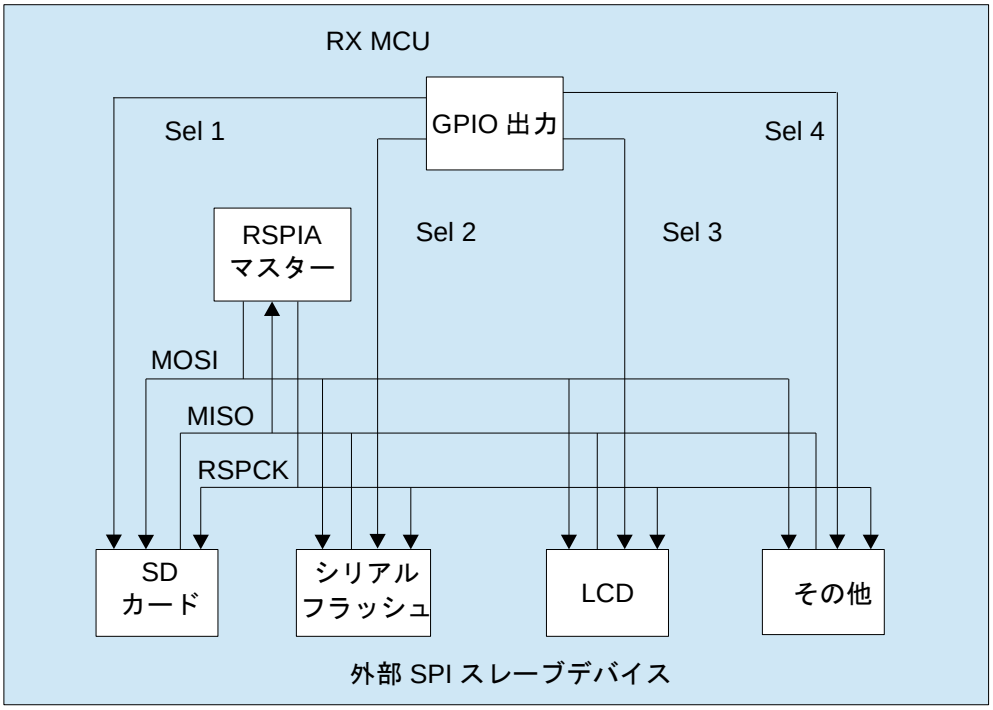

図 1.1　この例は、スレーブ選択信号として機能する GPIO ポートの使用を示しています（3 線式）

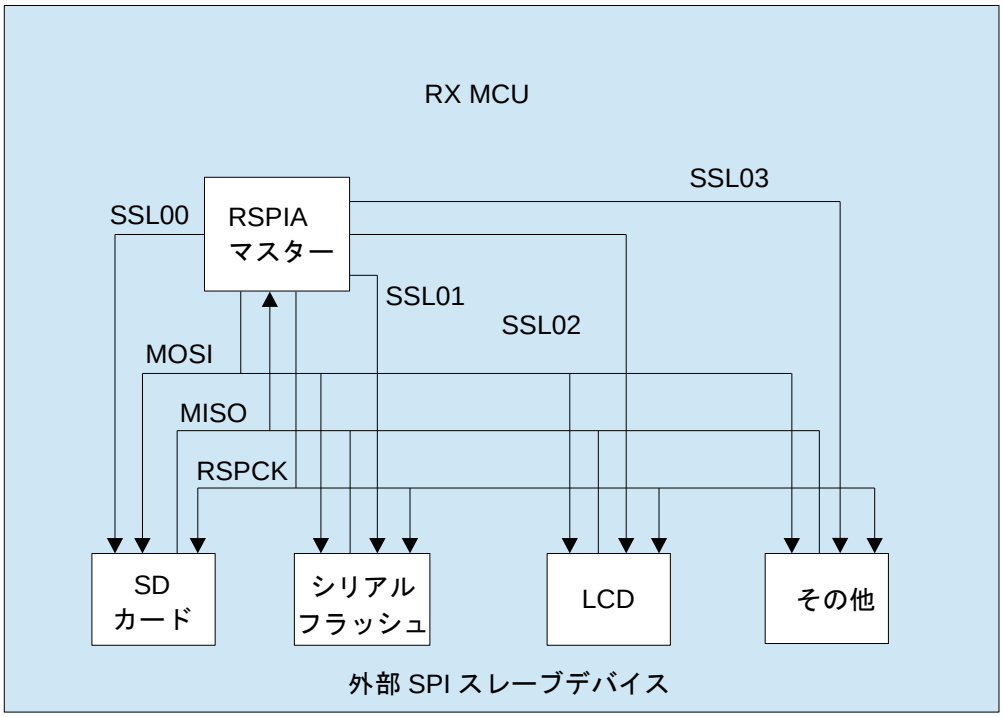

図 1.2　内蔵 RSPIA 周辺機能スレーブ選択ハードウェア（SSL）を信号生成に使用できます（SPI 4 線式）

2. API 情報

このドライバ API は、ルネサスの API 命名基準に従っています。

2.1 ハードウェアの要求

本ドライバでは、MCU が以下の機能をサポートすることを要求します。

本ドライバが要求するハードウェア周辺機能について説明します。明示的に規定しない限り、本ドライバ用にリソースを予約する必要があり、ユーザは個別にリソースを使用できません。

- 1 つ以上の RSPIA 周辺機能チャネルを使用できること。

2.2 ソフトウェアの要求

本ドライバは、以下のソフトウェアからのサポートを必要とします。

- 本ソフトウェアは、FIT 対応 BSP モジュール Rev.6.10 以上に依存します。本ソフトウェアの R_RSPIA_Open()を呼び出した後、関連する入出力ポートをどこかで正しく初期化する必要があります。
- 本ソフトウェアでは、本モジュールの API を呼び出す前に、BSP によって周辺機能クロック（PCLKA）が初期化済みであることを要求します。本ドライバによって、ビットレートレジスタ設定の計算に r_bsp マクロ‘BSP_PCLKx_HZ’が使用されます。r_bsp モジュールの外でユーザが PCLKx 設定を変更すると、ビットレート計算は無効になります。

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 V7.5.0）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 V7.5.0）および IAR プロジェクト（EWRX V4.12.1）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

RSPIA FIT モジュールは、「6.1 動作確認環境」に示すツールチェーンを使用して動作確認を行っています。

2.5　使用する割り込みベクタ

R_RSPIA_Open()関数を実行すると、引数チャネルおよび割り込み発生係数に従って割り込みが有効になります。

表 2.1 に FIT モジュールが使用する割り込みベクタを示します。

表 2.1　使用する割り込みベクター覧

デバイス	割り込みベクタ
RX671	SPRI 割り込み（ベクタ番号：48） SPTI 割り込み（ベクタ番号：49）

2.6　ヘッダファイル

すべての API 呼び出しには、本ソフトウェアのプロジェクトコードと共に提供される"r_rspia_rx_if.h"ファイルをインクルードすることでアクセスできます。

ビルドタイムコンフィグレーションオプションは、"r_rspia_rx_config.h"ファイルで選択または定義されます。

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_rspia _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_rspia _rx_config.h のコンフィグレーションオプション	
定義	説明
RSPIA_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。 0：パラメータチェック処理がビルドから省略されます。 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。
RSPIA_CFG_REQUIRE_LOCK　1	これを(1)に設定する場合、同時実行アクセス競合を防ぐために特定の処理を実行すると、RSPIA ドライバはチャネル用ロックを取得しようとします。 この関数は、DMAC/DTC を使用してデータ転送する場合には使用できません。 DMAC/DTC を使用してデータ転送を実行する場合は、0 に設定してください。
RSPIA_CFG_DUMMY_TXDATA　0xFFFFFFFF	受信専用処理中に、ユーザが指定したダミーデータを送信します。
RSPIA_CFG_USE_CH0　1	ビルド時に使用する RSPIA チャネルを有効にします。 (0) = 使用しない。(1) = 使用する。 **使用するチャネルを 1 つ以上有効にする必要があります。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。**
RSPIA_CFG_CH0_IR_PRIORITY　3	チャネルの共有割り込み優先順位を設定します。これは利便性のために提供されています。 優先順位は、チャネルに対して R_RSPIA_Open への呼び出しが実行された後、実行時に本モジュール外で引き続き変更できます。ただし、当該チャネルに対する R_RSPIA_Open への次回呼び出し時にこの設定値に戻ります。
RSPIA_CFG_CH0_TX_FIFO_THRESH　2	TX FIFO 閾値を設定します（RSPIA 対応 MCU のみ）。最低 0、最高 3 TX FIFO 閾値および RX FIFO 閾値に同じ値を設定します。
RSPIA_CFG_CH0_RX_FIFO_THRESH　2	RX FIFO 閾値を設定します（RSPIA 対応 MCU のみ）。最低 0、最高 3

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」を参照）。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、RAM およびスタックのコードサイズ

デバイス	分類	使用メモリ					
		ルネサス製コンパイラ		GCC		IAR コンパイラ	
		パラメータチェックあり	パラメータチェックなし	パラメータチェックあり	パラメータチェックなし	パラメータチェックあり	パラメータチェックなし
RX671	ROM	2866 バイト	2595 バイト	6620 バイト	6124 バイト	5162 バイト	4820 バイト
	RAM	64 バイト	64 バイト	44 バイト	44 バイト	60 バイト	60 バイト
	最大スタック使用量	80 バイト	72 バイト	-バイト	-バイト	160 バイト	156 バイト

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_rspia_rx_if.h に記述されます。

Open 用チャネル設定構造体

チャネルを開いたときに特定の動作モードを設定するには、R_RSPIA_Open()関数にこの構造体の初期化されたインスタンスへのポインタが必要です。

```
typedef struct rspia_chnl_settings_s
{
    rspia_interface_mode_t gpio_ssl;              /* RSPIA_IF_MODE_4WIRE：RSPIA
スレーブ選択、RSPIA_IF_MODE_3WIRE：GPIO スレーブ選択。 */
    rspia_master_slave_mode_t master_slave_mode; /* RSPIA_MS_MODE_MASTER または
RSPIA_MS_MODE_SLAVE。 */
    rspia_frame_select_t frame_mode;              /* RSPIA_IF_FRAME_TI_SSP または
RSPIA_IF_FRAME_MOTOROLA_SPI。 */
    uint32_t    bps_target;                       /* チャネルのターゲット bps（ビット
毎秒）設定値。 */
} rspia_chan_settings_t;
```

チャネルハンドルデータ構造体の抽象化

ユーザアプリケーションは、これをオープンチャネルへの参照として使用します。

```
typedef struct rspia_cfg_block_s *rspia_hdl_t;
```

```
typedef struct rspia_cfg_block_s
{
    uint8_t chan;
    uint8_t current_slave;                  /* 現在割り当てられているスレーブの数。 */
    bool rspia_chan_opened;                 /* この変数は、周辺機能がすでに初期化済みで
あるかどうかを判定。 */
    void   (*p_callback)(void *p_cbdat);    /* ユーザコールバック関数へのポインタ。 */
} rspia_cfg_block_t;
```

コマンド設定ワード用 Typedef 列挙型

このリストには、読み取り/書き込み処理用コマンドワードの特定の設定に使用可能な列挙型が含まれます。コマンドワードは、ビットフィールドの集合である 32 ビット値です。

これには、SPCMD レジスタの全ビットを設定するために正しい順序で上記の列挙型のいずれか 1 つが格納されます。

```
typedef union rspia_command_word_s
{
    R_BSP_ATTRIB_STRUCT_BIT_ORDER_RIGHT_14(
        rspia_spcmd_cpha_t        cpha         :1,
        rspia_spcmd_cpol_t        cpol         :1,
        rspia_spcmd_br_div_t      br_div       :2,
        rspia_spcmd_reserve_bit_t  rs0         :3,    /* 予約済み */
        rspia_spcmd_ssl_negation_t ssl_negate   :1,
        rspia_spcmd_reserve_bit_t  rs1         :4,    /* 予約済み */
        rspia_spcmd_bit_order_t    bit_order    :1,
```

```
        rspia_spcmd_spnden_t      next_delay    :1,
        rspia_spcmd_slnden_t      ssl_neg_delay :1,
        rspia_spcmd_sckden_t      clock_delay   :1,
        rspia_spcmd_bit_length_t   bit_length    :5,
        rspia_spcmd_reserve_bit_t  rs2          :3,    /* 予約済み */
        rspia_spcmd_ssl_assert_t   ssl_assert    :3,
        rspia_spcmd_reserve_bit_t  rs3          :5     /* 予約済み */
    );
    uint32_t word[1];
} rspia_command_word_t;
```

コマンドワードの初期化例

```
static const rspia_command_word_t my_command_reg_word = {
    RSPIA_SPCMD_CPHA_SAMPLE_ODD,
    RSPIA_SPCMD_CPOL_IDLE_LO,
    RSPIA_SPCMD_BR_DIV_1,
    RSPIA_SPCMD_RESERVE_BIT,
    RSPIA_SPCMD_SSL_KEEP,
    RSPIA_SPCMD_RESERVE_BIT,
    RSPIA_SPCMD_ORDER_MSB_FIRST,
    RSPIA_SPCMD_NEXT_DLY_SSLND,
    RSPIA_SPCMD_SSL_NEG_DLY_SSLND,
    RSPIA_SPCMD_CLK_DLY_SPCKD,
    RSPIA_SPCMD_BIT_LENGTH_8,
    RSPIA_SPCMD_RESERVE_BIT,
    RSPIA_SPCMD_ASSERT_SSL0,
    RSPIA_SPCMD_RESERVE_BIT,
};
```

これらの定義済みのリザーブビットの値は無視してください。ユーザがこの列挙型を入力する必要はありません。

コールバック関数データ構造体

チャネル番号とプロシージャ結果コードは、このデータ構造体の中でユーザ定義コールバック関数へ受け渡されます。

```
typedef struct rspia_callback_data_s
{
    rspia_hdl_t hdl;     /* チャネルハンドル */
    rspia_evt_t event;   /* イベントコード */
}rspia_callback_data_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_rspia_rx_if.h に記述されます。

```
typedef enum rspia_err_e        /* RSPIA API エラーコード */
{
    RSPIA_SUCCESS = 0,
    RSPIA_ERR_BAD_CHAN,         /* 無効なチャネル番号。 */
    RSPIA_ERR_OMITTED_CHAN,     /* config.h で RSPIA_CFG_USE_CHx が 0 */
    RSPIA_ERR_CH_NOT_OPENED,    /* チャネルがまだ開いていない。 */
    RSPIA_ERR_CH_NOT_CLOSED,    /* 前回開いてからチャネルが開いたまま。 */
    RSPIA_ERR_UNKNOWN_CMD,      /* 制御コマンドが認識されていない。 */
    RSPIA_ERR_INVALID_ARG,      /* パラメータの引数が無効。 */
    RSPIA_ERR_ARG_RANGE,        /* パラメータに対して引数が範囲外。 */
    RSPIA_ERR_NULL_PTR,         /* Null ポインタを受信。必要な変数が見つからない。 */
    RSPIA_ERR_LOCK,             /* ロックプロシージャでエラー発生。 */
    RSPIA_ERR_UNDEF,            /* 未定義/不明なエラー */
} rspia_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、SPRI、SPTI、SPEI 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 (*p_callback)(void *p_cbdat)”構造体メンバで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、コールバック関数のテンプレート例を示しています。

```
void my_callback(void *p_args)
{
    rspia_callback_data_t  *args;

    args = (rspia_callback_data_t *)p_args;
    callback_called_flag = true;


    if (args->event == RSPIA_EVT_TRANSFER_COMPLETE)
    {
        /* SPRI 割り込みから。 */
        R_BSP_NOP();
    }
    else if (args->event == RSPIA_EVT_TRANSFER_ABORTED)
    {
        /* データ転送中断。 */
        R_BSP_NOP();
    }
    else if (args->event == RSPIA_EVT_ERR_MODE_FAULT)
    {
        /* SPEI 割り込みから。モード障害エラー発生 */
        R_BSP_NOP();
    }
    else if (args->event == RSPIA_EVT_ERR_READ_OVF)
    {
        /* SPEI 割り込みから。レシーバオーバフローエラー発生 */
        R_BSP_NOP();
    }
    else if (args->event == RSPIA_EVT_ERR_PARITY)
    {
        /* レシーバパリティエラー割り込みから。エラー発生条件は
           割り込みルーチンの呼び出し中にクリア */
        R_BSP_NOP();
    }
    else if (args->event == RSPIA_EVT_ERR_UNDER_RUN)
    {
        /* アンダーランエラー割り込みから。エラー発生条件は
           割り込みルーチンの呼び出し中にクリア */
        R_BSP_NOP();
    }
    else if (args->event == RSPIA_EVT_ERR_UNDEF)
    {
```

```
            /* 未定義/不明なエラーイベント。 */
            R_BSP_NOP();
        }
    }
```

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。

3. API 関数

R_RSPIA_Open()

RSPIA 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。

Format

```
rspia_err_t   R_RSPIA_Open(uint8_t             chan,
                           rspia_chan_settings_t *p_cfg,
                           rspia_command_word_t  p_cmd,
                           void              (*p_callback)(void *p_cbdat),
                           rspia_hdl_t       *p_hdl);
```

Parameters

channel

初期化する RSPIA チャネルの数

**p_cfg*

RSPI チャネル構成データ構造体へのポインタ

p_cmd

SPCMD コマンドデータ構造体

*(*p_callback)(void * pcbdat)*

割り込みから呼び出されるユーザ定義関数へのポインタ

**p_hdl*

チャネル用ハンドルへのポインタ。ハンドル値はこの関数によって設定されます

Return Values

RSPIA_SUCCESS	/* 成功。チャネルは初期化された。 */
RSPIA_ERR_BAD_CHAN,	/* 無効なチャネル番号。 */
RSPIA_ERR_OMITTED_CHAN,	/* config.h で RSPIA_USE_CHx が 0 */
RSPIA_ERR_CH_NOT_CLOSED,	/* 前回開いてからチャネルが開いたまま。 */
RSPIA_ERR_INVALID_ARG,	/* パラメータの引数が無効。 */
RSPIA_ERR_ARG_RANGE,	/* パラメータに対して引数が範囲外。 */
RSPIA_ERR_NULL_PTR,	/* Null ポインタを受信。必要な変数が見つからない。 */
RSPIA_ERR_LOCK,	/* ロックプロシージャでエラー発生。 */

Properties

宣言は r_rspia_rx_if.h に記述されています。

Description

Open 関数は、処理のための RSPIA チャネルの準備を担当します。この関数は、他の RSPIA API 関数（R_RSPIA_GetVersion を除く）を呼び出す前に 1 回呼び出しておく必要があります。

正常に完了すると、選択した RSPI のステータスは"open"に設定されます。

その後、最初に R_RSPIA_Close()を呼び出して"close"を実行せずに、同じ RSPIA チャネルに対してこの関数を呼び出さないでください。

この処理が完了しても、コミュニケーションはまだ利用できません。入出力ポートで MPC および PMR を周辺機能モジュールに設定します。

Reentrant

不可

Example

条件：チャネルがまだ開いていない

```
    /* RSPIA 設定構造体を構成 */
    g_rspia_cfg.bps_target = 1000000;                       // 1Mbps ビットレートを
要求。
    g_rspia_cfg.master_slave_mode = RSPIA_MS_MODE_MASTER; // RSPIA を SPI マスター
として構成。
#if RSPIA_CFG_USE_GPIO_SSL == (0)
    g_rspia_cfg.gpio_ssl = RSPIA_IF_MODE_4WIRE;             // インタフェースモードを
4 線に設定。
    g_rspia_cfg.frame_mode = RSPIA_IF_FRAME_MOTOROLA_SPI; // 通信プロトコルを設定。
#else
    g_rspia_cfg.gpio_ssl = RSPIA_IF_MODE_3WIRE;             // インタフェースモードを
3 線に設定。
#endif
    /* RSPIA_CFG_USE_GPIO_SSL == (0) */
    /* API 関数を使用して RSPIA チャネルを開く */
    error = R_RSPIA_Open (RSPIA_CH0,&g_rspia_cfg, g_rspia_cmd,
&my_rspia_callback, &g_rspia_hdl);

    /* エラーが発生した場合、この関数は API 呼び出しのエラー検出をデモンストレーションします。
*/
    if (RSPIA_SUCCESS != error)
    {
        return error(); // エラー処理コードがここに記述されます。
    }

    /* RSPIA 周辺機能で使用するために入出力ポート端子を初期化。
     * これは、MCU および選択ポートに固有のものです。 */
    rspia_rx671_init_ports();
```

Special Notes:

この関数が呼び出されると、R_ RSPIA_Write()関数、R_RSPIA_Read()関数、R_ RSPIA_WriteRead()関数、R_RSPIA_Control()関数によって設定されたすべてのコンテンツは消去されます。

R_RSPIA_Control()

Control 関数は、RSPIA チャネル用の特別なハードウェア/ソフトウェア動作の処理を担当します。

Format

```
rspia_err_t  R_RSPIA_Control(rspia_hdl_t  hdl,
                        rspia_cmd_t     cmd,
                        void         *pcmd_data);
```

Parameters

hdl

チャネルのハンドル

cmd

列挙型コマンドコード

**pcmd_data*

実行完了のために必要なコマンド固有のデータの場所を参照するために使用される void 型のコマンドデータ構造体パラメータへのポインタ。サポートデータを必要としないコマンドは、FIT_NO_PTR を使用する必要があります。

有効な *cmd* 値は以下のとおりです。

```
typedef enum rspia_cmd_e
{
    RSPIA_CMD_SET_BAUD = 1,             /* 基本ビットレートを変更 */
    RSPIA_CMD_ABORT,                    /* 現在の読み取り/書き込み処理を直ちに停止。 */
    RSPIA_CMD_SET_REGS,                 /* 1 回の処理でサポートされるすべての RSPIA
                                           レジスタを設定。エキスパート以外使用禁止！ */
    RSPIA_CMD_CHANGE_TX_FIFO_THRESH,    /* TX FIFO 閾値を変更 */
    RSPIA_CMD_CHANGE_RX_FIFO_THRESH,    /* TX FIFO 閾値を変更 */
    RSPIA_CMD_UNKNOWN                   /* コマンドが無効。 */
} rspia_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。pcmd_data には FIT_NO_PTR を取得します。RSPIA_CMD_SET_BAUD コマンド以外のコマンドには引数は不要です。rspia_command_word_t には NULL を取得します。

RSPIA_CMD_SET_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 rspia_cmd_baud_t 変数へのポインタです。set baud コマンドのデータ構造体を以下に示します。

```
typedef struct rspia_cmd_baud_s
{
    uint32_t    bps_target;     /* チャネルのターゲット bps（ビット毎秒）設定値。 */
} rspia_cmd_baud_t;
```

RSPIA_CMD_CHANGE_TX_FIFO_THRESH および RSPIA_CMD_CHANGE_RX_FIFO_THRESH の引数（送信および受信バッファに異なる FIFO 閾値レベルを指定可能な MCU の場合）は、閾値レベルを保持する uint8_t 変数へのポインタです。

RSPIA_CMD_SET_REGS の引数は、RPSIA レジスタを設定するための各値を格納している

rspia_cmd_setregs_t 変数へのポインタです。RSPIA_CMD_SET_REGS コマンドを使用するには、最初に必要に応じた設定値を用いてインスタンスを作成し、次に R_RSPIA_Control()を呼び出してポインタを引数として受け渡します。

```
typedef struct rspia_cmd_setregs_s
{
    uint16_t spdcr_val;     /* RSPIA データ制御レジスタ（SPDCR） */
    uint8_t sslp_val;       /* RSPIA スレーブ選択極性レジスタ（SSLP） */
    uint8_t sppcr_val;      /* RSPIA 端子制御レジスタ（SPPCR） */
    uint8_t spckd_val;      /* RSPIA クロック遅延レジスタ（SPCKD） */
    uint8_t sslnd_val;      /* RSPIA スレーブ選択ネゲーション遅延レジスタ（SSLND） */
    uint8_t spnd_val;       /* RSPIA 次のアクセス遅延レジスタ（SPND） */
} rspia_cmd_setregs_t;
```

Return Values

RSPIA_SUCCESS	/* 成功。チャネルは初期化された。 */
RSPIA_ERR_CH_NOT_OPENED,	/* チャネルがまだ開いていない。 */
RSPIA_ERR_UNKNOWN_CMD,	/* 制御コマンドが認識されていない。 */
RSPIA_ERR_ARG_RANGE,	/* パラメータに対して引数が範囲外。 */
RSPIA_ERR_NULL_PTR,	/* Null ポインタを受信。必要な変数が見つからない。 */
RSPIA_ERR_LOCK,	/* ロックプロシージャでエラー発生。 */

Properties

宣言は r_rspia_rx_if.h に記述されています。

Description

この関数は、RSPIA チャネル用の特別なハードウェア/ソフトウェア動作の処理を担当します。選択した RSPIA を識別するための RSPI ハンドル、実行する処理を選択するための列挙型コマンド値、処理を完了するために必要な情報やデータを格納する場所への void 型ポインタを取得します。このポインタは、"r_rspia_rx_if.h"で提供される適切なデータ型を使用して特定のコマンド用に呼び出し元によって型キャストされたストレージを指定する必要があります。

Example

```
my_setbaud_struct.bps_target = 4000000; // 4Mbps に設定
result = R_RSPIA_Control(handle, RSPIA_CMD_SET_BAUD, &my_setbaud_struct);
if (RSPIA_SUCCESS != result)
{
return result;
}
...
/* 実行時間が長すぎるので、現在の転送を今すぐ停止すること！ */
result = R_RSPIA_Control(handle, RSPIA_CMD_ABORT, FIT_NO_PTR);
```

Reentrant

不可

R_RSPIA_Read()

Read 関数は、選択した SPI デバイスからデータを受信します。

Format

```
rspia_err_t  R_RSPIA_Read(rspia_hdl_t       hdl,
                      rspia_command_word_t  p_cmd,
                      void                 *p_dst,
                      uint16_t              length);
```

Parameters

hdl

チャネルのハンドル

p_cmd

この処理用 SPCMD のすべての RSPIA コマンドレジスタ設定から構成されるビットフィールドデータ。

コマンド設定ワードの使用については、「2.10 パラメータ」を参照してください。

**p_dst*

SPI デバイスから受信したデータのコピー先バッファへの void 型ポインタ。

要求されたデータカウントの保持に使用できる適切なスペースを確保する責任は、呼び出し元にあります。

引数は NULL であってはなりません。p_cmd.bit_length で指定されたデータフレームビット長に基づき、転送中に*p_dst pointer は、対応するデータ型へ型キャストされます。したがって、例えば、ビット長が 16 ビットに設定される場合、データはコピー先バッファに 16 ビット値として保存され、各ビット長設定に対し同様の処理が行われます。8、16、32 以外のビット長設定は、中に格納可能な最小のデータ型を使用します。例えば、24 ビットフレームは 32 ビットストレージに保存され、11 ビットフレームは 16 ビットストレージに保存されます。

length

転送するデータフレームの数を示す転送ビット長の変数。データワードのサイズは、p_cmd.bit_length 引数内の設定から決定されます。ビット長の引数がソースデータのストレージタイプに一致することを確認してください。これはフレームの数のカウントであって、バイト数ではありません。

Return Values

RSPIA_SUCCESS	/* 成功。チャネルは初期化された。 */
RSPIA_ERR_CH_NOT_OPENED	/* チャネルがまだ開いていない。 */
RSPIA_ERR_INVALID_ARG	/* パラメータの引数が無効。 */
RSPIA_ERR_NULL_PTR	/* null ポインタを受信。必要な引数が見つからない。 */
RSPIA_ERR_LOCK	/* ロックプロシージャエラー発生。 */

Properties

宣言は r_rspia_rx_if.h に記述されています。

Description

SPI デバイスからデータ受信を開始します。この関数は、処理の開始直後に結果を返し、要求されたビット長の受信が完了するまで、割り込みの制御下でデータの受信はバックグラウンドで継続します。

受信したデータは転送先バッファに保存されます。転送が完了すると、ユーザ定義コールバック関数が呼び出されます。

RSPIA の動作がマスターまたはスレーブのいずれであるかに応じて、処理は少し異なります。RSPIA がスレーブとして構成されている場合、マスターからクロックが受信されるときにのみデータは転送されます。

データ受信時に、RSPIA は構成ファイルでユーザが定義可能なダミーデータパターンも送信します。

Reentrant

不可

Example

```
/* 条件：チャネルが現在開いている。 */
g_transfer_complete = false;
result = R_RSPIA_Read(handle, my_command_word, dest, length);
if (RSPIA_SUCCESS != result)
{
return result;
}
while (!g_transfer_complete) // 割り込みコールバックがこれを設定するためのポーリング。
{
      // 転送が完了するまで待っている間に有益なことをすること。
      R_BSP_NOP();
}
```

Special Notes:

なし

R_RSPIA_Write()

Write 関数は、選択した SPI デバイスにデータを送信します。

Format

```
rspia_err_t  R_RSPIA_Write(rspia_hdl_t       hdl,
                      rspia_command_word_t  p_cmd,
                      void              *p_src,
                      uint16_t           length);
```

Parameters

hdl

チャネルのハンドル

p_cmd

この処理用 SPCMD のすべての RSPIA コマンドレジスタ設定から構成されるビットフィールドデータ。

コマンド設定ワードの使用については、「2.10 パラメータ」を参照してください。

**p_src*

データが SPI デバイスへ送信される元となるソースデータバッファへの Void 型ポインタ。

p_cmd.bit_length で指定されたデータフレームビット長に基づいて、転送中に*p_src pointer は対応するデータ型へ型キャストされます。したがって、例えば、ビット長が 16 ビットに設定される場合、ソースバッファデータは 16 ビットデータのブロックとしてアクセスされ、各ビット長設定に対し同様の処理が行われます。8、16、32 以外のビット長設定は、中に格納可能なデータ型を使用します。例えば、24 ビットフレームは 32 ビットストレージに保存され、11 ビットフレームは 16 ビットストレージに保存されます。

length

転送するデータフレームの数を示す転送ビット長の変数。データワードのサイズは、p_cmd.bit_length 引数内の設定から決定されます。ビット長の引数がソースデータのストレージタイプに一致することを確認してください。これはフレームの数のカウントであって、バイト数ではありません。

Return Values

RSPIA_SUCCESS	/* 成功。チャネルは初期化された。 */
RSPIA_ERR_CH_NOT_OPENED	/* チャネルがまだ開いていない。 */
RSPIA_ERR_INVALID_ARG	/* パラメータの引数が無効。 */
RSPIA_ERR_NULL_PTR	/* null ポインタを受信。必要な引数が見つからない。 */
RSPIA_ERR_LOCK	/* ロックプロシージャエラー発生。 */

Properties

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

Description

SPI デバイスへのデータ送信を開始します。この関数は、送信処理の開始直後に結果を返し、要求されたビット長の送信が完了するまで、割り込みの制御下でデータの送信はバックグラウンドで継続します。

送信が完了すると、ユーザ定義コールバック関数が呼び出されます。

転送が完了したことをユーザアプリケーションに通知するには、コールバック関数を使用してください。この関数は転送処理のみを実行します。RSPIA 送信中、データは受信されません。

Reentrant

不可

Example

```
/* 条件：チャネルが現在開いている。 */
g_transfer_complete = false;
result = R_RSPIA_Write(handle, my_command_word, source, length);
if (RSPIA_SUCCESS != result)
{
return result;
}
while (!g_transfer_complete) // 割り込みコールバックがこれを設定するためのポーリング。
{
      // 転送が完了するまで待っている間に有益なことをすること。
      R_BSP_NOP();
}
```

Special Notes:

なし

R_RSPIA_WriteRead()

Write Read 関数は、SPI デバイスからのデータ受信中、同時にデータを SPI デバイスへ送信します。

Format

```
rspia_err_t  R_RSPIA_WriteRead(rspia_hdl_t       hdl,
                           rspia_command_word_t  p_cmd,
                           void                 *p_src,
                           void                 *p_dst,
                           uint16_t              length);
```

Parameters

hdl

チャネルのハンドル

p_cmd

この処理用 SPCMD のすべての RSPIA コマンドレジスタ設定から構成されるビットフィールドデータ。

コマンド設定ワードの使用については、「2.10 パラメータ」を参照してください。

**p_src*

データが SPI デバイスへ送信される元となるソースデータバッファへの void 型ポインタ。

p_cmd.bit_length で指定されたデータフレームビット長に基づいて、転送中に*p_src pointer は対応するデータ型へ型キャストされます。したがって、例えば、ビット長が 16 ビットに設定される場合、ソースバッファデータは 16 ビットデータのブロックとしてアクセスされ、各ビット長設定に対し同様の処理が行われます。8、16、32 以外のビット長設定は、中に格納可能なデータ型を使用します。例えば、24 ビットフレームは 32 ビットストレージに保存され、11 ビットフレームは 16 ビットストレージに保存されます。

**p_dst*

SPI デバイスから受信したデータのコピー先バッファへの void 型ポインタ。

要求されたデータカウントの保持に使用できる適切なスペースを確保する責任は、呼び出し元にあります。引数は NULL であってはなりません。p_cmd.bit_length で指定されたデータフレームビット長に基づき、転送中に*p_dst pointer は、対応するデータ型へ型キャストされます。したがって、例えば、ビット長が 16 ビットに設定される場合、データはコピー先バッファに 16 ビット値として保存され、各ビット長設定に対し同様の処理が行われます。8、16、32 以外のビット長設定は、中に格納可能な最小のデータ型を使用します。例えば、24 ビットフレームは 32 ビットストレージに保存され、11 ビットフレームは 16 ビットストレージに保存されます。

length

転送するデータフレームの数を示す転送ビット長の変数。データワードのサイズは、p_cmd.bit_length 引数内の設定から決定されます。ビット長の引数がソースデータのストレージタイプに一致することを確認してください。これはフレームの数のカウントであって、バイト数ではありません。

Return Values

RSPIA_SUCCESS	/* 成功。チャネルは初期化された。 */
RSPIA_ERR_CH_NOT_OPENED	/* チャネルがまだ開いていない。 */
RSPIA_ERR_INVALID_ARG	/* パラメータの引数が無効。 */
RSPIA_ERR_NULL_PTR	/* null ポインタを受信。必要な引数が見つからない。 */
RSPIA_ERR_LOCK	/* ロックプロシージャエラー発生。 */

Properties

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

Description

SPI デバイスへの全二重送信および SPI デバイスからの全二重受信を開始します。この関数は、転送処理の開始直後に結果を返し、要求されたビット長の転送が完了するまで、割り込みの制御下でデータの転送はバックグラウンドで継続します。処理が完了すると、ユーザ定義コールバック関数が呼び出されます。転送が完了したことをユーザアプリケーションに通知するには、コールバック関数を使用してください。

RSPIA の動作がマスターまたはスレーブのいずれであるかに応じて、処理は少し異なります。RSPIA がスレーブとして構成されている場合、マスターからクロックが受信されるときにのみデータは転送されます。

送信用データはソースバッファから取得され、受信データは転送先バッファに保存されます。

Reentrant

不可

Example

```
/* 条件：チャネルが現在開いている。 */
g_transfer_complete = false;
result = R_RSPIA_WriteRead(handle, my_command_word, source, dest, length);
if (RSPIA_SUCCESS != result)
{
return result;
}
while (!g_transfer_complete) // 割り込みコールバックがこれを設定するためのポーリング。
{
      // 転送が完了するまで待っている間に有益なことをすること。
      R_BSP_NOP();
}
```

Special Notes:

なし

R_RSPIA_Close()

ハンドルによって指定された RSPIA チャネルを完全に無効にします。

Format

```
rspia_err_t R_RSPIA_Close(rspia_hdl_t hdl);
```

Parameters

hdl

チャネルのハンドル

Return Values

RSPIA_SUCCESS	/* 成功。チャネルは初期化された。 */
RSPIA_ERR_CH_NOT_OPENED,	/* チャネルがまだ開いていない。 */
RSPIA_ERR_NULL_PTR,	/* Null ポインタを受信。必要な変数が見つからない。 */

Properties

宣言は r_rspia_rx_if.h に記述されています。

Description

ハンドルによって指定された RSPIA チャネルを無効にします。RSPIA ハンドルは、'open'状態ではないことを示すために変更されます。RSPIA チャネルは、R_RSPIA_Open 関数によって再び開かれるまで使用できません。

Open 状態ではない RSPIA に対してこの関数が呼び出されると、エラーコードが返されます。

Reentrant

不可

Example

```
rspia_err_t result;
result = R_RSPIA_Close(handle);
if (RSPIA_SUCCESS != result)
{
return result;
}
```

Special Notes:

なし

R_RSPIA_GetVersion()

API のバージョン番号を返します。

Format

```
uint32_t R_RSPIA_GetVersion(void);
```

Parameters

なし

Return Values

バージョン番号

Properties

宣言は r_rspia_rx_if.h に記述されています。

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、上位 2 バイトがメジャーバージョン番号、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

```
/* バージョン番号を取得して文字列に変換。 */
uint32_t version, version_high, version_low;
char version_str[9];
version = R_RSPIA_GetVersion();
version_high = (version >> 16)&0xf;
version_low = version & 0xff;
sprintf(version_str, "RSPIAv%1.1hu.%2.2hu", version_high, version_low);
```

Reentrant

可能

Special Notes:

なし

4. 端子設定

RSPIA 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_RSPIA_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 または Smart Configurator による関数出力」を参照してください。

表 4.1　FIT Configurator または Smart Configurator による関数出力

選択したオプション	出力される関数
チャネル 0	R_RSPIA_PinSet_RSPIA0()

3 線インタフェースモードが使用されている場合は、スレーブ選択信号を処理するように GPIO ポートを構成する必要があります。

GPIO を構成するには、FIT GPIO モジュール API を使用するか、レジスタを直接設定します。

RSPCK 極性の設定

RSPCK 端子の極性を設定する rspia_command_word_t 構造体 rspia_spcmd_cpol_t の値の設定は R_RSPIA_Open()関数が呼び出されると更新されます。また、RSPCK 端子の出力は、表 4.1 の関数を実行すると最終処理されます。

5. サンプルプログラム

本アプリケーションノートには、FIT RSPIA モジュールの基本使用方法をデモンストレーションするための 1 つ以上のサンプルプログラムが記述されています。

サンプルプログラムは、使用する共通 API 関数呼び出しの簡単な関数サンプルを提供することを目的にしたものです。

提供されるサンプルアプリケーションは、ジャンパー線を使用してマスター出力データをマスター入力データにルーティングすることで、全二重送信（同時送受信）をシミュレートします。受信データは、送信データに一致することを確認するためにテストされます。

RSPIA モジュールのバージョン番号が取得され、希望する場合は、Renesas Virtual Debug Console ウィンドウに表示できます。

5.1 サンプルプログラムをワークスペースに追加

サンプルプログラムは、本アプリケーションノート用に配布されたファイルの FITDemos フォルダに格納されており、MCU とボード専用です。使用する予定のルネサス開発ボードに一致するサンプルプログラムを見つけてください。

5.2 サンプルプログラムの実行

1. ターゲットボードに応じて MOSIx 端子を MISOx 端子にジャンパ接続し、ボードを準備します。
   - RSKRX671
     i. 拡張ヘッダ JA3 のピン 7 を JA3 のピン 8 へ接続、SW4 のピン 3 をオフにします。
2. e2studio デバッガを使用してサンプルアプリケーションをビルドし、RSK ボードにダンロードします。
3. e2studio で[Renesas Virtual Debug Console]ビューを選択し、出力情報を表示します。
4. デバッガでアプリケーションを実行します。
5. デバッグコンソールウィンドウでバージョン番号出力を確認します。
6. 転送に成功すると”Success!”が、失敗すると”Failed.”がデバッグコンソールウィンドウに表示されます。

6. 付録

6.1 動作確認環境

本モジュールの動作テスト環境を詳しく説明します。

表 6.1 動作確認環境（Rev.1.00）

項目	内容
統合開発環境	Renesas Electronics e^2 studio Version 2021-07 IAR Embedded Workbench for Renesas RX 4.20.3
C コンパイラ	Renesas Electronics C/C++ Compiler Package for RX Family V3.03.00 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。 -lang = c99
	GCC for Renesas RX 8.3.0.202004 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。 -std=gnu99 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。 -WI,--no-gc-sections リンカが誤って FIT 周辺機能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 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	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）

表 6.2 動作確認環境（Rev.1.10）

項目	内容
統合開発環境	Renesas Electronics e^2 studio Version 2021-07 IAR Embedded Workbench for Renesas RX 4.20.3
C コンパイラ	Renesas Electronics C/C++ Compiler Package for RX Family V3.03.00 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。 -lang = c99
	GCC for Renesas RX 8.3.0.202004 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。 -std=gnu99 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。 -WI,--no-gc-sections リンカが誤って FIT 周辺機能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 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	リトルエンディアン
モジュールのリビジョン	Rev.1.10
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_rspia_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、構成設定が間違っている場合のエラーが発生しました。
A：“r_rspia_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_rspia_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。

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
テクニカルアップデート／テクニカルニュース
ユーザーズマニュアル：開発ツール
　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

本モジュールに該当するテクニカルアップデートはありません。

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
1.10	Sep.13.21	31 32	5 章サンプルプログラムの内容を更新。 5.2「サンプルプログラム実行」に、RX671 を追加 「6.1 動作確認環境」： Rev.1.10 に対応する表を追加。

製品ご使用上の注意事項

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

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

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。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。