RX ファミリ

RSPI モジュール Firmware Integration Technology

R01AN1827JJ0170
Rev.1.70
2017.07.31

要旨

本アプリケーションノートは、Firmware Integration Technology (FIT) を使用した RSPI モジュールについて説明します。RSPI ドライバのアーキテクチャ、ユーザアプリケーションへの FIT モジュールの組み込み、および API の使用方法についての詳細を説明します。

このモジュールでサポートされる RX ファミリの MCU は、最大 3 チャネルのシリアルペリフェラルインタフェース（RSPI）を内蔵しています。RSPI は、全二重同期式のシリアル通信ができます。複数のプロセッサや周辺デバイスとの高速なシリアル通信機能を内蔵しています。

動作確認デバイス

この API は現時点で次のデバイスでサポートされています。

- RX110 グループ
- RX111 グループ
- RX113 グループ
- RX130 グループ
- RX210 グループ
- RX231 グループ
- RX23T グループ
- RX24T グループ
- RX24U グループ
- RX62N グループ
- RX62T グループ
- RX63N、RX631 グループ
- RX64M グループ
- RX65N、RX651 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（R01AN1833JU）
- ボードサポートパッケージモジュール Firmware Integration Technology（R01AN1685JJ）
- e² studio に組み込む方法 Firmware Integration Technology (R01AN1723JU)
- CS+に組み込む方法 Firmware Integration Technology (R01AN1826JJ)

目次

1. 概要

このソフトウェアは RSPI 周辺モジュールの動作の準備と SPI バス上でのデータ転送を実行するアプリケーションプログラミングインタフェース（API）を提供しています。

RSPI ドライバモジュールはユーザアプリケーションと物理的ハードウェアとの間に位置し、RSPI 周辺モジュールを管理する下位のハードウェア制御タスクを行います。

このソフトウェアをご使用になる前に、RX MCU ユーザーズマニュアル ハードウェア編の RSPI 周辺モジュールの章を確認することを推奨します。

図 1　プロジェクトレイヤの例

1.1 RSPI FIT モジュールとは

本モジュールは API として、プロジェクトに組み込んで使用します。本モジュールの組み込み方については、「2.12 FIT モジュールの追加方法」を参照してください。

1.2 RSPI FIT モジュールの概要

RSPI ドライバモジュールをプロジェクトに追加した後、使用環境に合わせたソフトウェアの設定のために r_rspi_rx_config.h ファイルの変更が必要です。コンフィグレーションオプションの詳細は、2.7 コンパイル時の設定を参照してください。

RSPI モジュールは入出力ポートのレジスタを初期化する機能を持っていません。入出力ポートの設定は、このモジュール以外で行われていなければなりません。

実行時に RSPI チャネルを使用する際の最初のステップは、必要な設定とパラメータを渡して R_RSPI_Open()関数をコールすることです。この関数の終了時には、I/O ポートを設定することで RSPI チャネルはアクティブとなり、この API で利用可能な他のすべての関数を実行する準備が完了します。この時点で SPI データ転送操作を行うことができます。また、設定を変更するさまざまな制御操作を行うこともできます（注 1）。

注 1 : クロック同期処理（3 線方式）のマスタモードで使用する場合は、以下の手順を実行してデータ転送を準備してください。それ以外の場合、クロックの同期ずれ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（RSPI スレーブの場合は SPE=0 に設定します）
(2) R_RSPI_Open()をコールします。
(3) I/O ポートの設定によりピンを周辺モジュールに設定します。
(4) R_RSPI_Write()をコールします。これはダミーデータ転送により SPI モードを決定するための処理です。
(5) 通信用スレーブを有効にします。

RSPI コマンドレジスタ（SPCMD）を除く RSPI レジスタ設定は、R_RSPI_Open()関数をコールすることで実行します。汎用的に使用されることを目的としていため、RSPI レジスタにはレジスタの初期値を設定します。また、R_RSPI_Control()関数をコールすることで、RSPI ドライバモジュールに格納されている RSPI レジスタ情報を書き換えることができます。書き換えた情報を RSPI レジスタに反映させるためには、R_RSPI_Close()関数を実行し、再度 R_RSPI_Open()を実行する必要があります。

R_RSPI_Control()関数には 3 つのコマンドが用意されています。

- ビットレートの変更。
- 転送動作の即時中断。
- RSPI ドライバモジュールに格納されている RSPI レジスタ情報の書き換え。

SPI バスでデータ転送が行われているとき、ドライバはユーザが用意したコールバック関数を呼び出すことで、ユーザアプリケーションに終了ステータスを通知します。

ほとんどの RSPI API 関数は「ハンドル」引数を必要とします。これは動作のために選択された RSPI チャネル番号を識別するために使用されます。ハンドルは最初に R_RSPI_Open()関数を呼び出すことで得られます。R_RSPI_Open()関数呼び出しでハンドルが格納される変数のアドレスを指定する必要があります。関数の終了時に、ハンドルが使用可能になります。他の API 関数が呼び出されるときには、このハンドルの値を単に引数として渡すことで RSPI チャネル番号が指定されます。チャネルごとにハンドルが割り当てられるため、アプリケーションではどのハンドルがどのチャネルに対応するかを管理することが必要です。

1.2.1 ドライバでサポートされる RSPI の機能

このドライバは RSPI 周辺モジュールが持つ機能の以下のサブセットをサポートしています。

RSPI 転送機能：

- MOSI（Master Out Slave In）、MISO（Master In Slave Out）、SSL（Slave Select）、および RSPCK（RSPI Clock）信号を使用して、SPI 動作（4 線式）／クロック同期式動作（3 線式）でシリアル通信が可能
- マスタモード／スレーブモードでのシリアル通信が可能
- シリアル転送クロックの極性を変更可能
- シリアル転送クロックの位相を変更可能

データフォーマット：

- MSB ファースト／LSB ファーストの切り替え可能
- 転送ビット長を 8、9、10、11、12、13、14、15、16、20、24、および 32 ビットから選択可能

ビットレート：

- マスタモード時、内蔵ボーレートジェネレータで PCLK を分周して RSPCK を生成（分周比は 2～4096 分周）
- スレーブモード時は、シリアルクロックとして外部入力クロックを使用（最大周波数は MCU のユーザーズマニュアル参照）

エラー検出：

- モードフォルトエラー検出
- オーバランエラー検出
- パリティエラー検出
- アンダランエラー検出

SSL 制御機能：

- 1 チャネルあたり 4 本の SSL 信号（SSLn0～SSLn3）
- シングルマスタモード時：SSLn0～SSLn3 信号を出力
- スレーブモード時：SSLn0 信号は入力で RSPI スレーブを選択、SSLn1～SSLn3 信号は未使用
- SSL 出力のアサートから RSPCK 動作までの遅延（RSPCK 遅延）を設定可能
  設定範囲：1～8 RSPCK　設定単位：1 RSPCK
- RSPCK 停止から SSL 出力のネゲートまでの遅延（SSL ネゲート遅延）を設定可能
  設定範囲：1～8 RSPCK　設定単位：1 RSPCK
- 次アクセスの SSL 出力アサートのウェイト（次アクセス遅延）を設定可能
  設定範囲：1～8 RSPCK　設定単位：1 RSPCK＋2 PCLK
- SSL 極性変更機能

マスタ転送時の制御方式：

- 転送動作ごとに以下を設定可能
  スレーブセレクト値、ベースビットレート分周、SPI クロックの極性/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、スレーブセレクトネゲート遅延、次アクセス遅延

割り込み要因：

- RSPI 受信割り込み（受信バッファフル）
- RSPI 送信割り込み（送信バッファエンプティ）
- RSPI エラー割り込み（モードフォルト、オーバラン、パリティエラー、アンダラン）

1.2.2 サポートされていない RSPI の機能

- このドライバは DMAC／DTC による転送をサポートしていません。
- RX111 のような RAM 容量の限られている MCU で RAM リソースを浪費しないため、このドライバではデータバッファを静的に確保せず、上位レベルのユーザアプリケーションでバッファを確保する必要があります。これにより、アプリケーションで RAM を確保する方法を制御できます。
- 1 シーケンスデータ転送のみをサポートします。このドライバでは、RSPI 周辺モジュールが持つマルチコマンドシーケンスデータ転送機能をサポートしていません。
- 1 フレームデータ転送のみをサポートします。このドライバでは、RSPI 周辺モジュールが持つマルチフレーム機能をサポートしていません。このため、サポートされる最大のデータフレームサイズは 32 ビットです。
- マルチマスタモードはサポートしていません。
- 16 ビット型のバイトスワップはサポートしていません。

1.3 API の概要

この API には次の関数が含まれています。

表 1-1 RSPI API 関数一覧

関数	説明
R_RSPI_Open()	指定された RSPI チャネルの準備で必要となる関連レジスタの初期化を行い、他の API 関数で使用されるハンドルを返します。割り込みイベントに答えるため、コールバック関数のポインタを引数としています。
R_ RSPI_Close()	指定された RSPI チャネルを無効にします。
R_RSPI_Control()	RSPI チャネルに固有なハードウェアまたはソフトウェアの操作を行います。
R_RSPI_Write()	Write 関数は SPI マスタまたはスレーブデバイスにデータを送信します。
R_RSPI_Read()	Read 関数は SPI マスタまたはスレーブデバイスからデータを受信します。
R_RSPI_WriteRead()	Write Read 関数は SPI マスタまたはスレーブデバイスにデータを送信し、同時にそのデバイスからデータを受信します（全二重）。
R_RSPI_GetVersion()	ドライバのバージョン番号を返します。

1.4　ドライバのアーキテクチャ

1.4.1　システム構成例

ドライバはシングルマスタ／マルチスレーブモードとスレーブモードの動作をサポートしています。各 RSPI チャネルは 1 つの SPI バスを制御します。このドライバは同じバス上でのマルチマスタ動作はサポートしていません。1 つの SPI バス上でのシングルマスタと複数のスレーブとの接続例を次に示します。

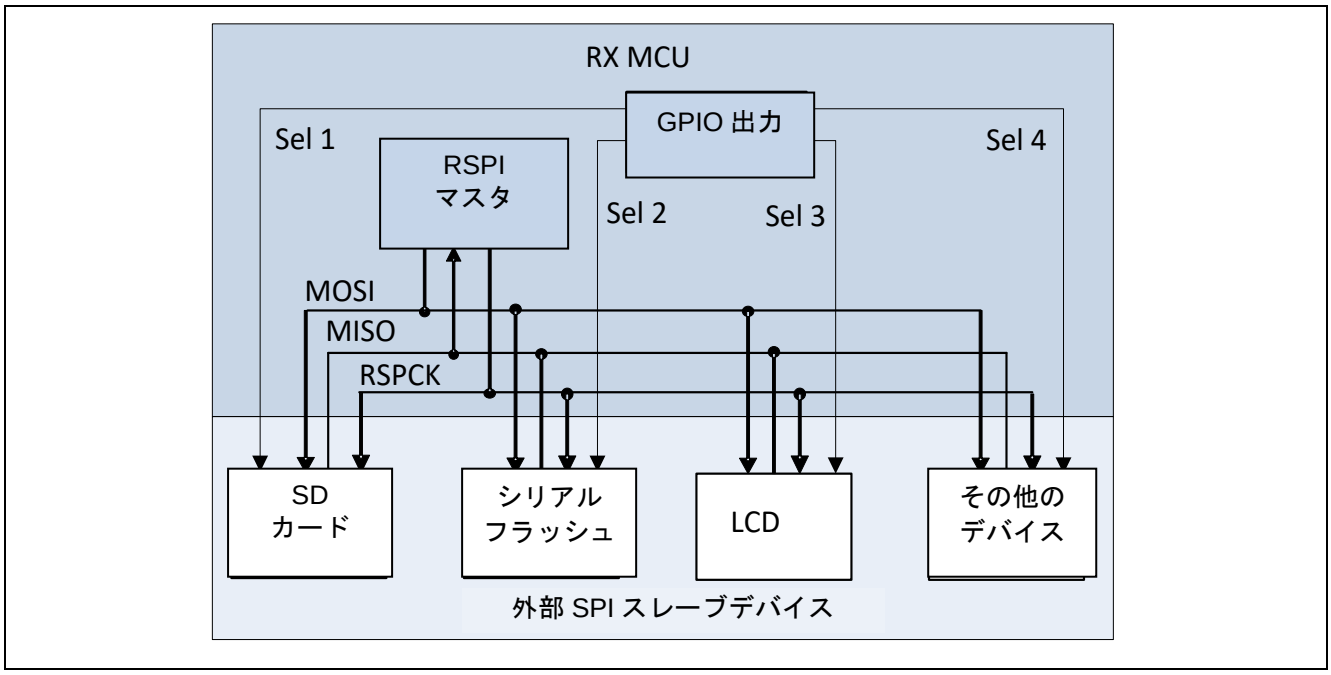

図 2　GPIO ポートをスレーブセレクト信号として使用する例（3 線モード）

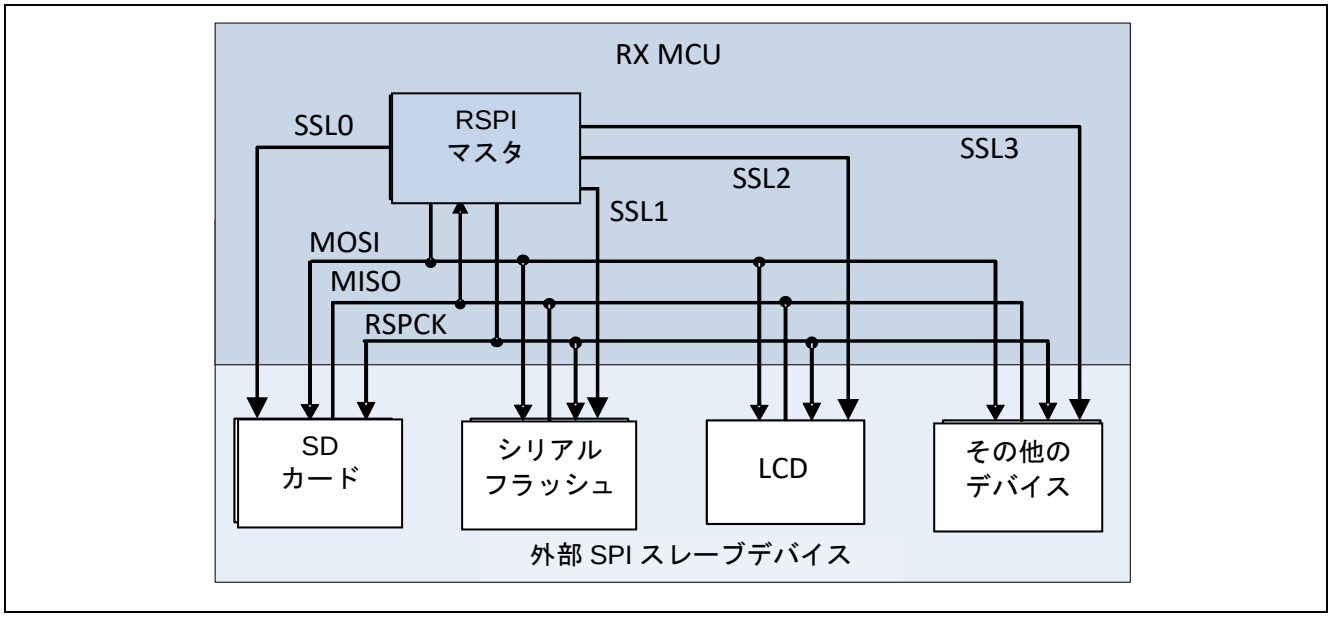

図 3　RSPI 周辺モジュールに組み込まれたスレーブセレクトハードウェア（SSL）を使用して信号を生成（SPI 4 線モード）

1.4.2　マルチチャネル RSPI サポート

サポートされている RX ファミリの MCU では複数のチャネルを持つ RSPI が内蔵されており、このドライバは同一のコードで、実装されているチャネルのすべてを個別に選択し使用することができます。各チャネルは、他の使用中のチャネルとは別に、独自に構成を設定することができます。

1.5　データ転送動作

RSPI ドライバは書き込み、読み込み、書き込み／読み込み（全二重）の 3 つのデータ転送関数を持っており、マスタモードとスレーブモードのいずれの動作も行うことができます。書き込みと読み込みの動作はすべて全二重モードに設定された RSPI で実行されます。すべての書き込みと読み込みの動作は、書き込みまたは読み込みの関数の呼び出しで転送動作を設定して制御を戻す、ノンブロッキング方式で行われます。関数は動作が正しく初期化されるか、エラーが生じたときに戻り値を返します。

コンフィグレーションオプションでロック機能が有効であれば、動作中の RSPI チャネルはロックされます。その後の転送動作の残り部分は RSPI 割り込みハンドラルーチンによって実行されます。

RSPI 送信バッファエンプティ割り込み（SPTI）ISR は最初の送信動作を実行するために実装されています。最初の 1 回または 2 回の SPTI 割り込みの発生（マスタモードかスレーブモードかに依存します）で、SPTI ISR はその後の SPTI 割り込みを禁止します。この後の残りの転送動作は受信バッファフル割り込み（SPRI）により行われます。この割り込みはフレームデータすべてが RSPI によって受信されたことを示しており、また、フレーム全体がクロックにより送出されたことも意味しています。

SPTI と SPRI 割り込みはいずれも全チャネルに共通のハンドラルーチンを呼び出します。1 回目は、データが送出されていないため、受信データは有効な内容にはならず、SPDR レジスタに最初のデータを格納する操作にとどまります。その後、SPDR レジスタは受信バッファをクリアするためだけに読み出され、データは破棄されます。更にスレーブモードでは、スレーブの送信データに対してダブルバッファが使用されているため、マスタによって連続的にクロック送出される場合でも、スレーブのシフトレジスタを空にすることはありません。スレーブモードの場合、SPRI 割り込みが発生する前に SPTI 割り込みが 2 回発生することになります。最後のデータが転送された後、SPRI ISR が呼び出され、転送動作は終了します。

指定された数のフレームが転送されると、ISR は転送動作を終了し、RSPI チャネルを停止し、その割り込みを禁止とします。その後、アプリケーションに転送終了を伝えるためにユーザ定義のコールバック関数を呼び出します。ロック機能が有効なときには、この時点でチャネルのロックは解除されます。

1.5.1　データ送信とデータ受信

(1)　RSPI からのデータ送信

マスタモードでは、データは RSPI マスタによって MOSI（Master Out Slave In）ラインに書き込まれます。スレーブモードでは、データは RSPI スレーブによって MISO（Master In Slave Out）ラインに書き込まれます。すべてのデータ転送動作が内部的には全二重で行われるため、送信のみの動作では RSPI ドライバは受信バッファをクリアするためにのみ SPDR レジスタを読み込み、データは廃棄します。送信されるデータはユーザアプリケーションで指示されたバッファから読み出され、その時点の動作として設定されているデータ型にキャストされた後、SPDR レジスタに書き込まれます。

(2)　SPI スレーブからデータを受信する RSPI マスタ

RSPI マスタは MISO（Master In Slave Out）ラインからデータを受信します。SPI バスマスタに構成された RSPI 周辺モジュールでは、SPI バス上のスレーブデバイスからデータを受信するために全二重動作に設定されます。このため RSPI マスタはスレーブにクロックを送出する必要があります。クロック送出は RSPI マスタがデータを送信しているときにのみ行われます。そのため、SPI バスからデータを読み出すには、マスタは同時にデータを書き込む必要があります。この際のデータは転送する実際のデータ（スレーブが全二重の通信が行える場合）であっても、スレーブで無視されるダミーデータであってもかまいません。このドライバではデータの受信はユーザがデータパターンを設定可能なダミーデータの送出でクロックによるデータ受信を行う構造となっています。

(3) SPI マスタにデータ書き込みを行う RSPI スレーブ

スレーブモードでは、書き込み動作はマスタモードの場合とほとんど同じです。相違点は、送信を設定した後、スレーブはマスタ SPI デバイスからのクロックを待つことです。更に、スレーブモードでは、スレーブのデータ送信でダブルバッファを使用しているため、マスタによって連続的にクロック送出されるフレームがスレーブのシフトレジスタを空にすることはありません。

データ書き込み中に読み出しを行わない場合は、フレームが送信されるたびに SPDR レジスタを読み込み、データは廃棄します。送信動作は指定された数のフレームが送信されるか、ユーザコマンドで中断が指示されたときに終了します。

(4) SPI マスタから読み込みを行う RSPI スレーブ

スレーブモードの読み込み動作はマスタモードの場合と同じです。相違点は、受信を設定した後、スレーブはマスタ SPI デバイスからのクロックを待つことです。受信中に有効なデータを送信しない場合は、シフトレジスタはダミーデータで埋められます。読み込み動作は指定された数のフレームが受信されるか、ユーザコマンドで中断が指示されたときに終了します。

1.5.2　データ出力と RAM の関係性

データ型が 16 ビットまたは 32 ビット、かつ、Little エンディアンの場合、RAM に格納されているデータ順にはデータが出力されないことに注意が必要です。必要に応じてバイトスワップ処理を行ってください。なお、RSPIc 以降の IP バージョンにはバイトスワップ機能が搭載されています。

(1)　データ送信

(a)　データ型 16 ビット【Little エンディアン】

エラー! 参照元が見つかりません。に示すとおり 1 フレームのデータ型が 16 ビットの場合、RAM のデータを SPDR レジスタに書き込むタイミングでデータが反転します。そのため、データの出力順番は Byte1、Byte0、Byte3、Byte2・・・となります。

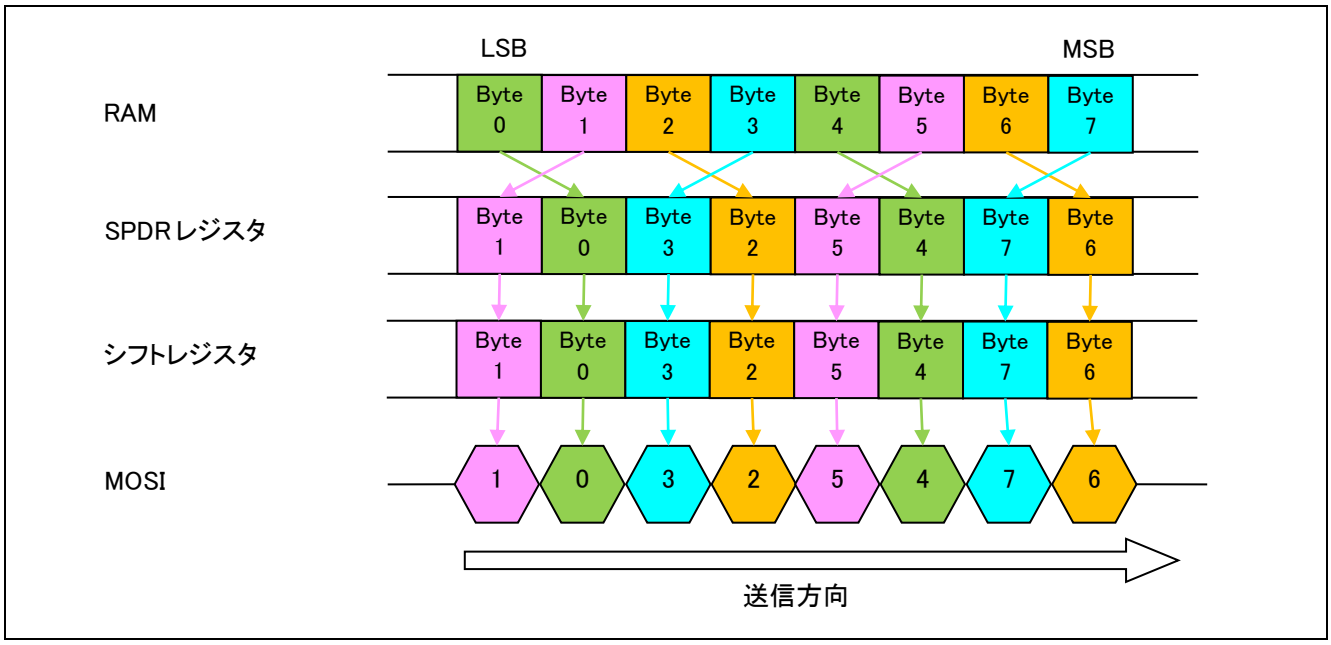

図 1-1 データ送信　データ型 16 ビット【Little エンディアン】バイトスワップ実行無し

RSPIc 以降の IP バージョンにはバイトスワップ機能が搭載されており、ハードウェアにてバイトスワップを行うことができます。ただし、RSPI ドライバは 16 ビットハードウェアバイトスワップをサポートしていません。

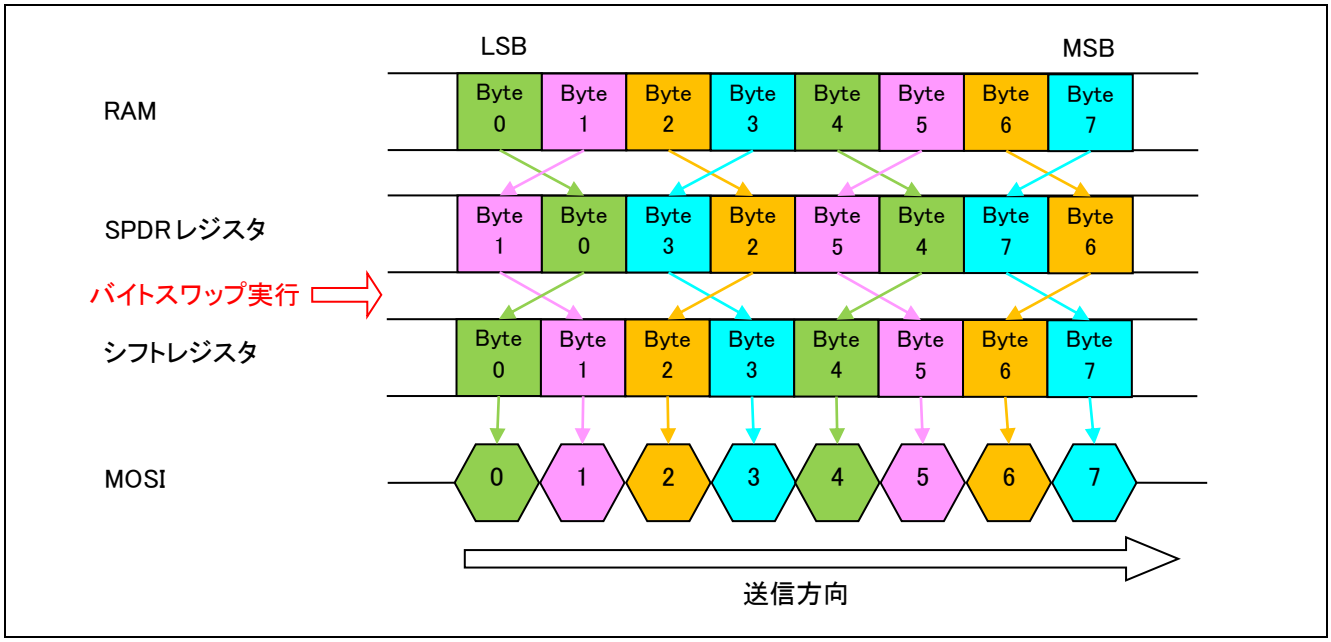

図 1-2　データ送信　データ型 16 ビット【Little エンディアン】バイトスワップ実行

(b)　データ型 32 ビット【Little エンディアン】

図 1-3 に示すとおり 1 フレームのデータ型が 32 ビットの場合、RAM のデータを SPDR レジスタに書き込むタイミングでデータが反転します。そのため、データの出力順番は Byte3、Byte2、Byte1、Byte0・・・となります。

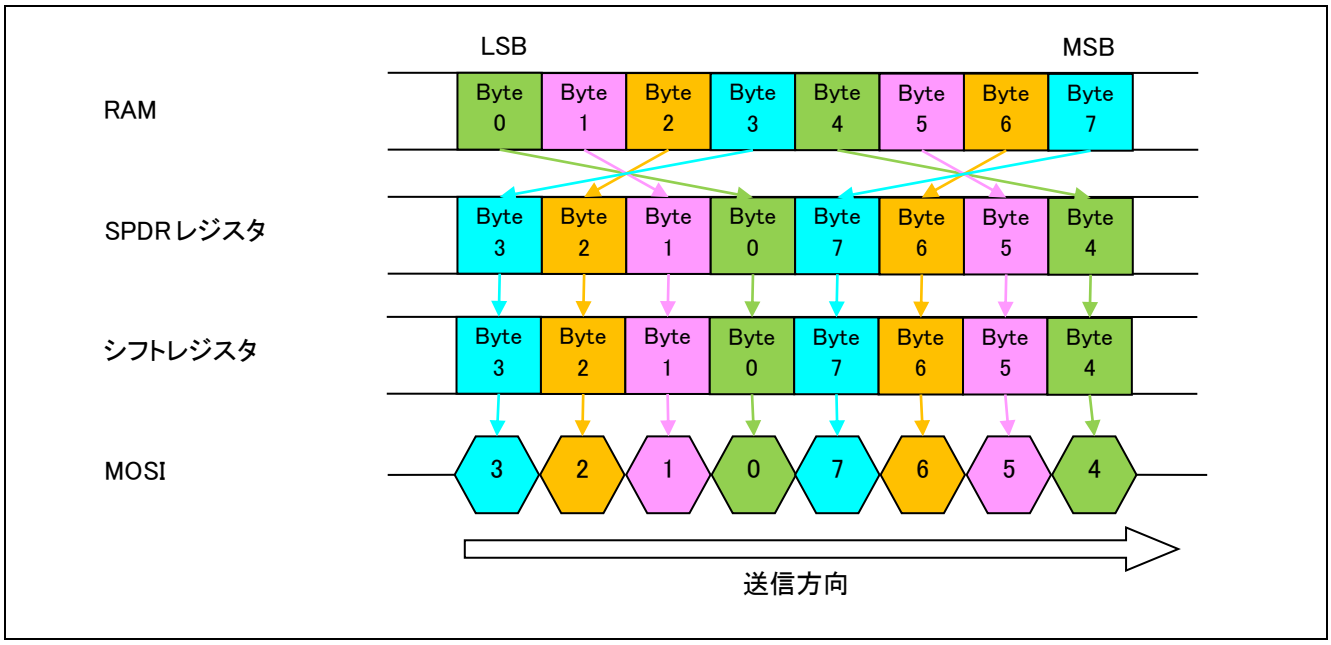

図 1-3　データ送信　データ型 32 ビット【Little エンディアン】バイトスワップ実行無し

RSPIc 以降の IP バージョンにはバイトスワップ機能が搭載されており、ハードウェアにてバイトスワップを行うことができます。RSPI ドライバは、32 ビットハードウェアバイトスワップをサポートしています。

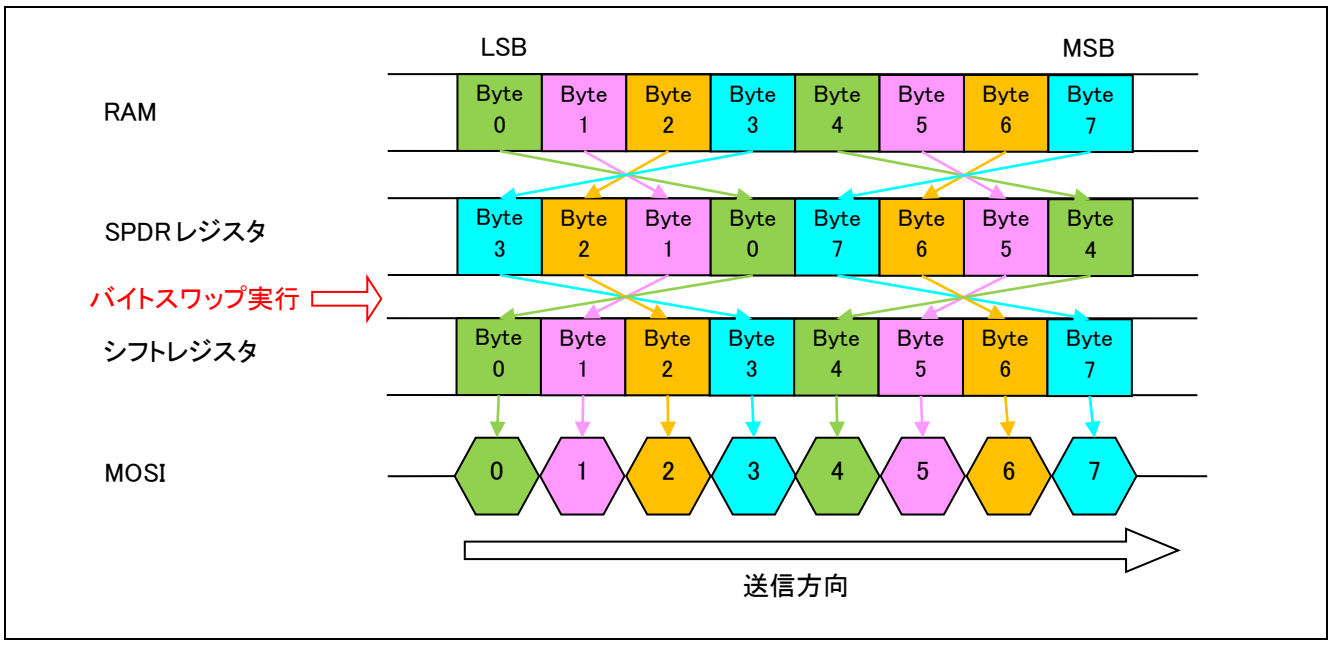

図 1-4　データ送信　データ型 32 ビット【Little エンディアン】バイトスワップ実行

(c)　その他データ型、および、エンディアン

以下に示すデータ型、および、エンディアンの場合、RAM に配置されているデータの順番にデータが出力されます。

- データ型 8 ビット【Little エンディアン／Big エンディアン】
- データ型 16 ビット【Big エンディアン】
- データ型 32 ビット【Big エンディアン】

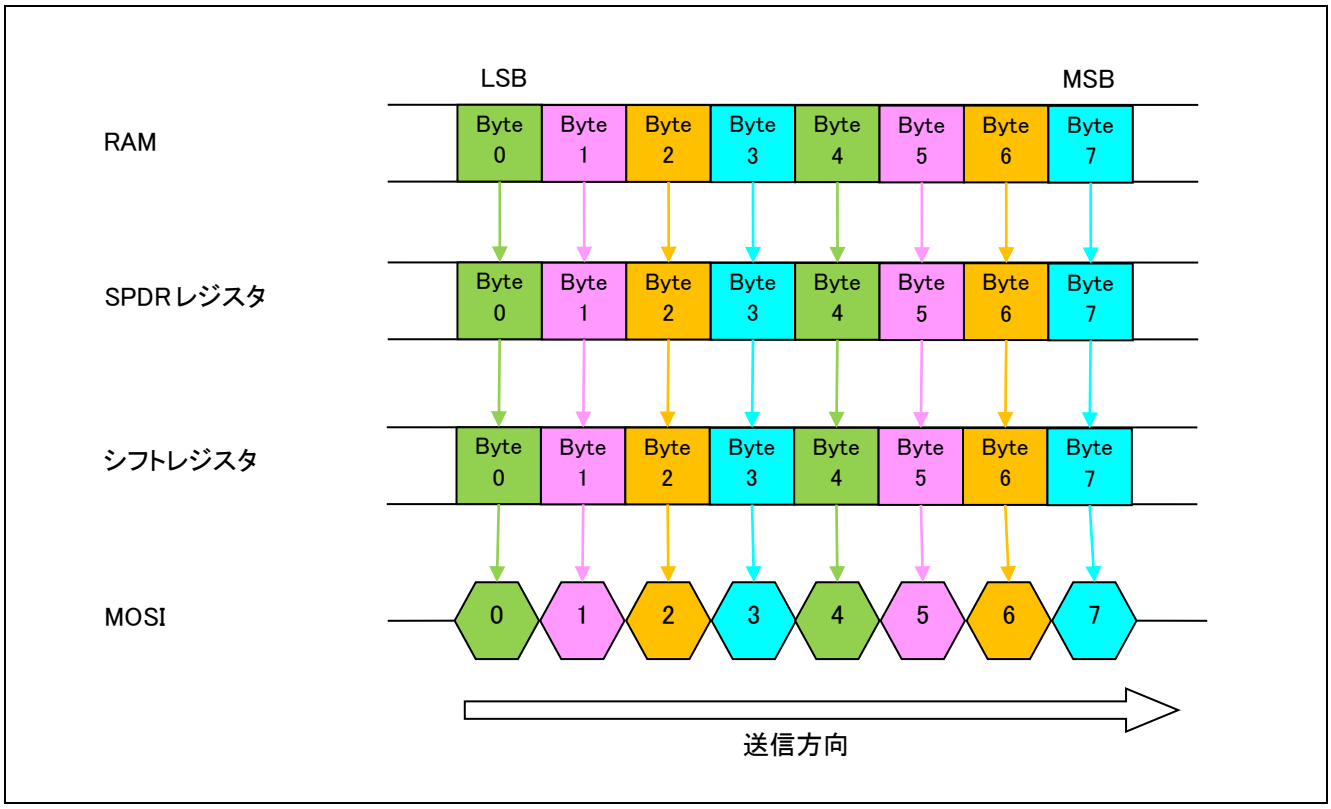

図 1-5　データ送信　その他データ型、および、エンディアン

(2)　データ受信

(d)　データ型 16 ビット【Little エンディアン】

図 1-6 に示すとおり 1 フレームのデータ型が 16 ビットの場合、SPDR レジスタから RAM へデータを読み出すタイミングでデータが反転します。そのため、RAM に格納されるデータの順番は Byte1、Byte0、Byte3、Byte2・・・となります。

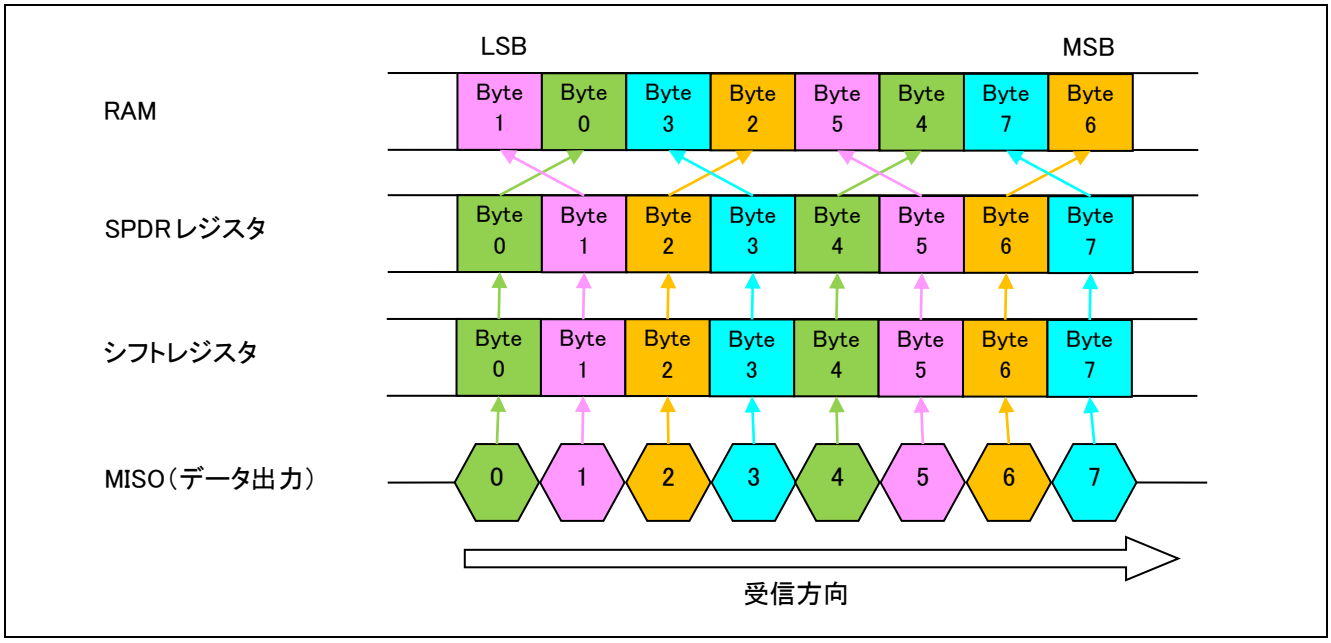

図 1-6　データ受信　データ型 16 ビット【Little エンディアン】バイトスワップ実行無し

RSPIc 以降の IP バージョンにはバイトスワップ機能が搭載されており、ハードウェアにてバイトスワップを行うことができます。

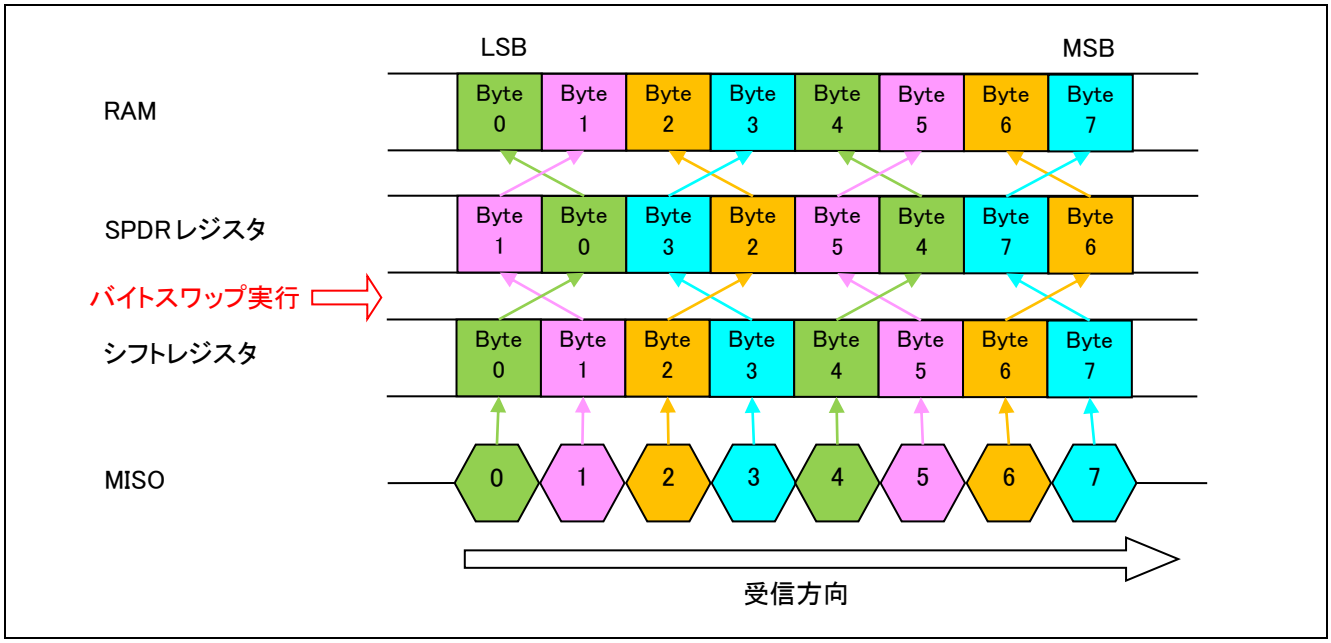

図 1-7　データ受信　データ型 16 ビット【Little エンディアン】バイトスワップ実行

(e)　データ型 32 ビット【Little エンディアン】

図 1-8 に示すとおり 1 フレームのデータ型が 32 ビットの場合、SPDR レジスタから RAM へデータを読み出すタイミングでデータが反転します。そのため、RAM に格納されるデータの順番は Byte3、Byte2、Byte1、Byte0・・・となります。

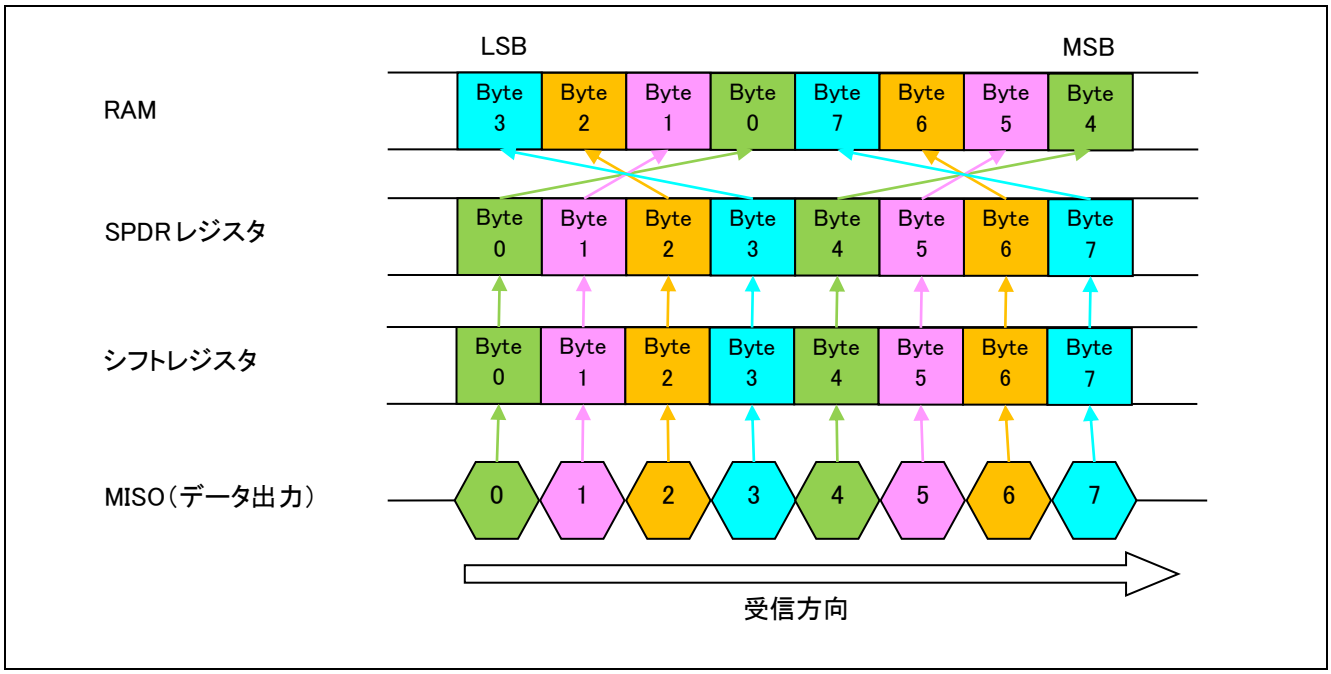

図 1-8　データ受信　データ型 32 ビット【Little エンディアン】バイトスワップ実行無し

RSPIc 以降の IP バージョンにはバイトスワップ機能が搭載されており、ハードウェアにてバイトスワップを行うことができます。

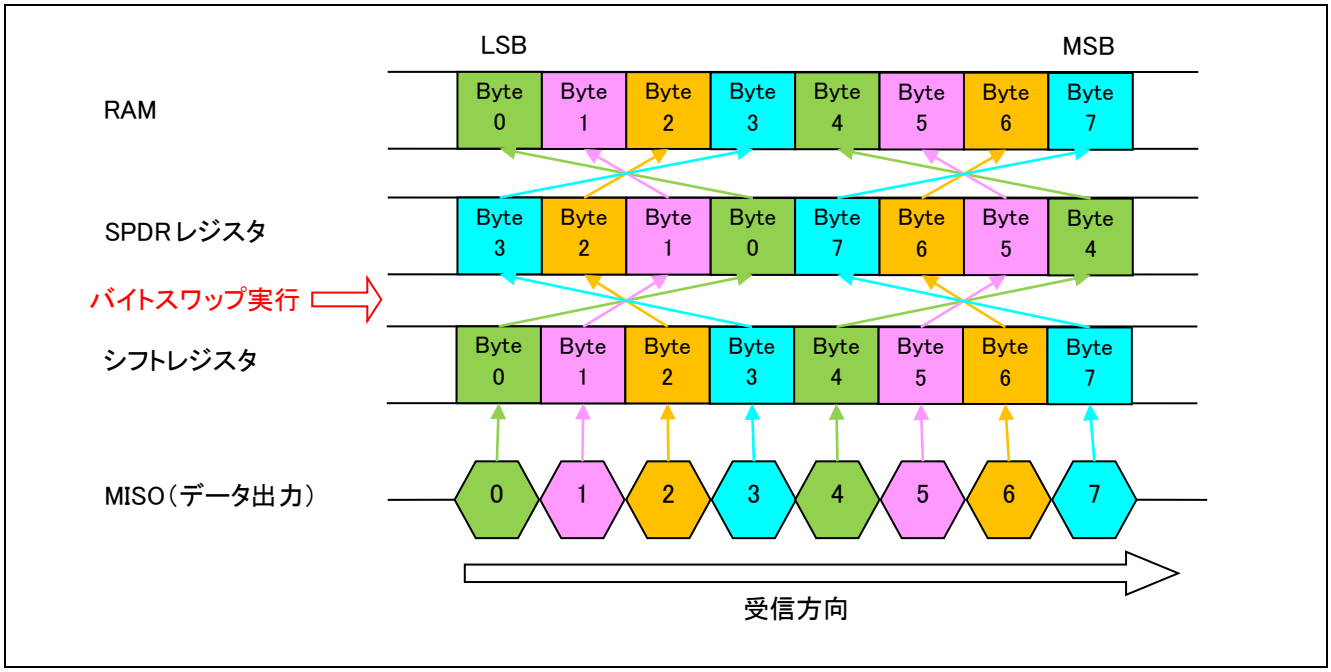

図 1-9　データ受信　データ型 32 ビット【Little エンディアン】バイトスワップ実行

(f)　その他データ型、および、エンディアン

以下に示すデータ型、および、エンディアンの場合、データ出力の順番にデータが RAM へ格納されます。

- データ型 8 ビット【Little エンディアン／Big エンディアン】
- データ型 16 ビット【Big エンディアン】
- データ型 32 ビット【Big エンディアン】

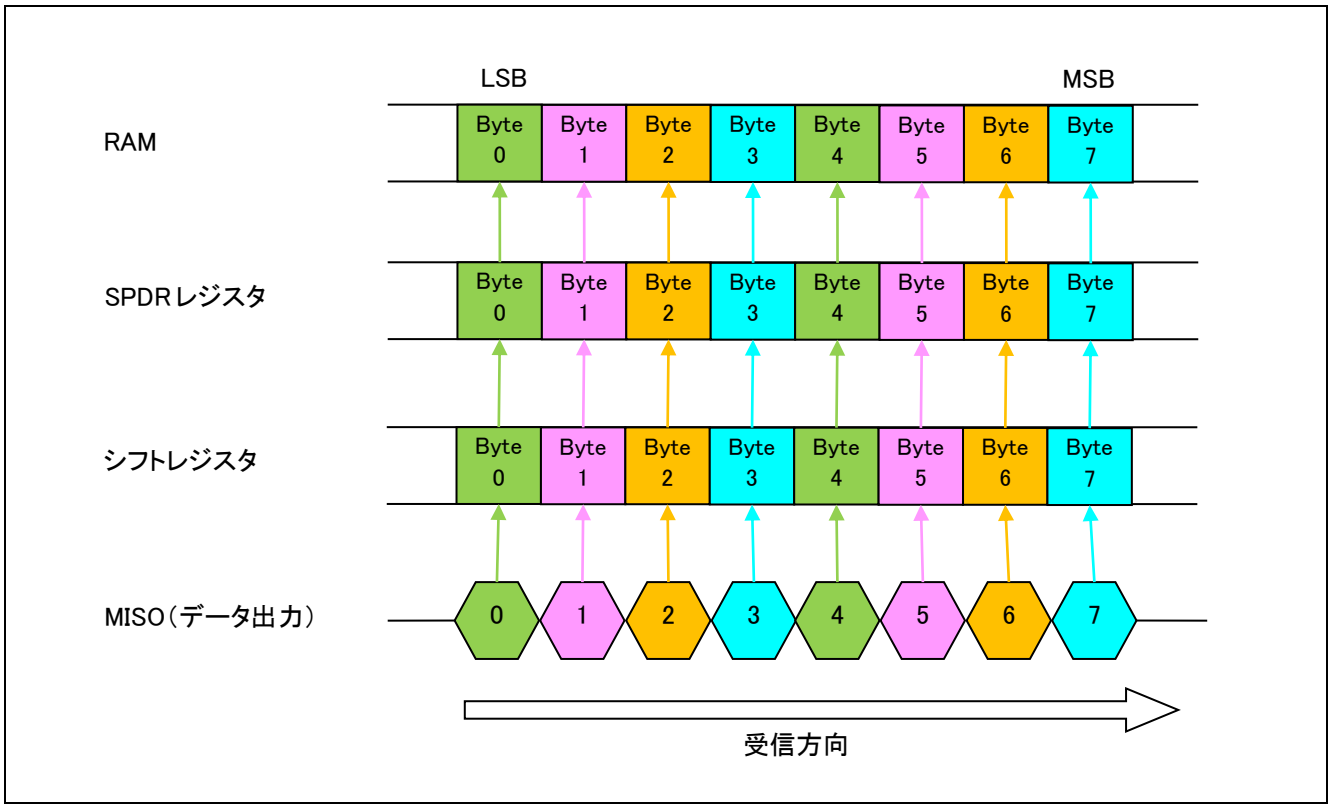

図 1-10　データ受信　その他データ型、および、エンディアン

1.6　割り込み

1.6.1　データ転送割り込み

RSPI ドライバは送信と受信の動作をノンブロッキング方式で実行します。データ転送動作は割り込みサービスルーチン中にイベントドリブンで実行されます。RSPI 送信バッファエンプティ割り込み（SPTI）および受信バッファフル割り込み（SPRI）が、1 個のフレームの送信と受信の手順を実行する共通の読み込み／書き込み関数を呼び出すために使われます。

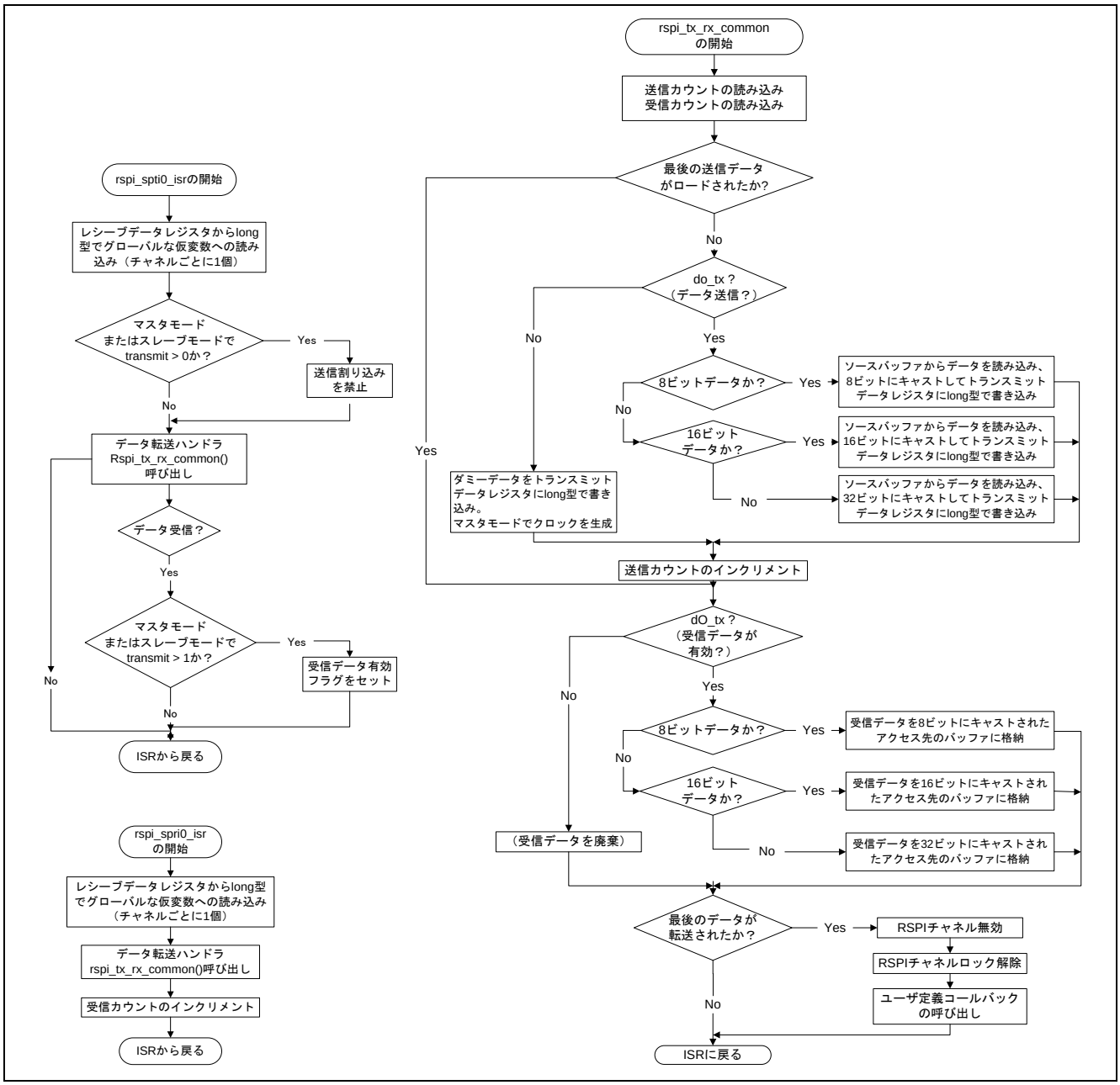

図 4　共通のデータ転送割り込みハンドラのアルゴリズム

1.6.2 エラー割り込み

RSPI エラー割り込み（SPEI）は、割り込みの原因を判断するためにステータスレジスタを読み出す共通ハンドラ関数を起動するために使われます。更に、データ転送動作を停止し、コールバック関数が呼び出されます。

エラー割り込みハンドラ処理では、OVRF→MODF→UDRF→PERF の順番で SPSR レジスタの各フラグ状態を確認します。最初に検出したエラーフラグの状態をコールバック関数の引数 event に設定します。

2. API 情報

このドライバの API はルネサスの命名規則に従っています。

2.1 ハードウェアの要求

このドライバでは、使用する MCU が次の機能をサポートしている必要があります。
また、このセクションではドライバが必要とする周辺回路ハードウェアについて説明します。特に明記されていない限り、周辺回路は専らドライバで使用され、ユーザアプリケーションで使用することはできません。

- 1 つまたは複数の使用可能な RSPI 周辺モジュールチャネル

2.2 ソフトウェアの要求

このドライバは次のソフトウェアからのサポートに依存しています。

- このソフトウェアは、FIT に準拠する BSP モジュールに依存しています。このソフトウェアの R_RSPI_Open()関数呼び出しの後に、関連する入出力ポートが正しく初期化されていることが前提となります。
- このソフトウェアでは、このモジュールの API 呼び出しの前に、BSP によって周辺モジュールクロック（PCLK）が初期化されている必要があります。r_bsp の BSP_PCLKx_HZ マクロは、このドライバでビットレートレジスタの設定値を計算するために使われています。ユーザが PCLKx の設定を r_bsp モジュール以外に変更した場合、ビットレートの計算が無効になります。

2.3 サポートされているツールチェイン

本 FIT モジュールは「6.1 動作確認環境」に示すツールチェインで動作確認を行っています。

2.4　使用する割り込みベクタ

R_RSPI_Read()関数／R_RSPI_Write()関数／R_RSPI_WriteRead()関数のいずれかを実行すると引数のチャネルに対応した割り込みが有効になります。

表 2-1 に本 FIT モジュールが使用する割り込みベクタを示します。

表 2-1　使用する割り込みベクター覧

デバイス	割り込みベクタ
RX110	SPRI0 割り込み[チャネル 0]（ベクタ番号：45）
	SPTI0 割り込み[チャネル 0]（ベクタ番号：46）
RX111	SPRI0 割り込み[チャネル 0]（ベクタ番号：45）
	SPTI0 割り込み[チャネル 0]（ベクタ番号：46）
RX113	SPRI0 割り込み[チャネル 0]（ベクタ番号：45）
	SPTI0 割り込み[チャネル 0]（ベクタ番号：46）
RX130	SPRI0 割り込み[チャネル 0]（ベクタ番号：45）
	SPTI0 割り込み[チャネル 0]（ベクタ番号：46）
RX210	SPRI0 割り込み[チャネル 0]（ベクタ番号：45）
	SPTI0 割り込み[チャネル 0]（ベクタ番号：46）
RX230／RX231	SPRI0 割り込み[チャネル 0]（ベクタ番号：45）
	SPTI0 割り込み[チャネル 0]（ベクタ番号：46）
RX23T	SPRI0 割り込み[チャネル 0]（ベクタ番号：45）
	SPTI0 割り込み[チャネル 0]（ベクタ番号：46）
RX24T	SPRI0 割り込み[チャネル 0]（ベクタ番号：45）
	SPTI0 割り込み[チャネル 0]（ベクタ番号：46）
RX24U	SPRI0 割り込み[チャネル 0]（ベクタ番号：45）
	SPTI0 割り込み[チャネル 0]（ベクタ番号：46）
RX63N	SPRI0 割り込み[チャネル 0]（ベクタ番号：39）
	SPTI0 割り込み[チャネル 0]（ベクタ番号：40）
	SPRI1 割り込み[チャネル 1]（ベクタ番号：42）
	SPTI1 割り込み[チャネル 1]（ベクタ番号：43）
	SPRI2 割り込み[チャネル 2]（ベクタ番号：45）
	SPTI2 割り込み[チャネル 2]（ベクタ番号：46）
RX64M	SPRI0 割り込み[チャネル 0]（ベクタ番号：38）
	SPTI0 割り込み[チャネル 0]（ベクタ番号：39）
RX65N／RX651	SPRI0 割り込み[チャネル 0]（ベクタ番号：38）
	SPTI0 割り込み[チャネル 0]（ベクタ番号：39）
	SPRI1 割り込み[チャネル 1]（ベクタ番号：40）
	SPRI1 割り込み[チャネル 1]（ベクタ番号：41）
	SPRI2 割り込み[チャネル 2]（ベクタ番号：108）
	SPRI2 割り込み[チャネル 2]（ベクタ番号：109）
RX71M	SPRI0 割り込み[チャネル 0]（ベクタ番号：38）
	SPTI0 割り込み[チャネル 0]（ベクタ番号：39）
	SPRI1 割り込み[チャネル 1]（ベクタ番号：40）
	SPTI1 割り込み[チャネル 1]（ベクタ番号：41）

2.5　ヘッダファイル

すべての API 呼び出しはこのソフトウェアのプロジェクトコードとともに提供されている 1 個のファイル r_rspi_rx_if.h をインクルードすることによって行われます。ビルド時のコンフィグレーションオプションは r_rspi_rx_config.h ファイルで選択または定義されます。

2.6　整数型

ご使用のツールチェインが C99 をサポートしている場合、以下に示すような stdint.h が含まれます。C99 がサポートされていない場合、ルネサスのコーディング規約文書で定義されるような typedefs.h ファイルがプロジェクトに含まれています。

このプロジェクトでは、コードをわかりやすく、移植性をより大きくするために、ANSI C99 の固定長整数型（Exact width integer types）を使用しています。これらの型は stdint.h で定義されています。

2.7　コンパイル時の設定

このソフトウェアの一部の機能や動作はユーザの指定が必要なコンフィグレーションオプションによってビルド時に決定されます。

表 2-2 RSPI ドライバのコンフィグレーション設定

Configuration options in *r_rspi_rx_config.h*	
RSPI_CFG_PARAM_CHECKING_ENABLE	RSPI API 関数に渡される引数のチェックを有効または無効に設定します。高速で小さなサイズのコードの必要性が高いシステムでは、引数チェックを無効にすることができます。 デフォルトでは、システム全体で有効な BSP_CFG_PARAM_CHECKING_ENABLE マクロの設定を使用するように設定されています。 RSPI_CFG_PARAM_CHECKING_ENABLE を再定義することで、この設定をローカルに上書きして RSPI モジュールの設定を優先させることもできます。 ローカルにパラメータチェックの有無を設定するには、チェックを行うときには RSPI_CFG_PARAM_CHECKING_ENABLE の値を 1 に、チェックを行わないときには値 0 を設定します。
RSPI_CFG_REQUIRE_LOCK	値が 1 に設定されているときには、RSPI ドライバが何らかの操作を行う際に同時アクセスによる競合を避けるため、チャネルのロックを確保しようと試みます。
RSPI_CFG_DUMMY_TXDATA	受信のみの動作の際に送信される、ユーザが指定するダミーデータを設定します。
RSPI_CFG_USE_CHANn	使用される RSPI チャネルをビルド時に有効にします。 (0) = 使用せず　(1) = 使用
RSPI_CFG_IR_PRIORITY_CHANn	チャネル内で共有される割り込み優先レベルの設定。この設定は便宜的なものです。優先レベルは、チャネルに対する R_RSPI_Open()関数が呼び出された後に、このモジュール外から実行時に変更できます。ただし、このチャネルに対する次の R_RSPI_Open()関数呼び出しで優先レベルはこの設定値に戻ります。
RSPI_CFG_MASK_UNUSED_BITS	長さが 8、16、32 ビットのいずれでもないデータフレームビットを RSPI 受信データレジスタから読み込む場合、上位のビットには送信データからのビットが残っています。便宜上、オプションとしてこの未使用の上位ビットはユーザデータバッファに移される際にドライバでマスクする（0 にクリア）ことができます。この操作はデータ転送割り込みハンドラで余分な処理時間を要するため、最大のビットレート設定での効率を低下させることになります。 データ転送が 8、16、32 ビットのいずれかに限られるときには、この機能は必要ありません。このオプションはデータフレームのビット長が 8、16、または 32 ビット以外のときにのみ有効としてください。 (0) = クリアしない　(1) = 未使用の上位ビットをクリアする
RSPI_CFG_USE_RX63_ERROR_INTERRUPT	RX63x グループの MCU では、RSPI エラー割り込みは SCI 周辺モジュールと共有されるグループ割り込みとなります。このため、RX63x グループでは SCI FIT モジュールとの競合を避ける目的で、デフォルトではエラー割り込みが禁止となっています。ただし、SCI FIT モジュールを使用しない場合は、RSPI_CFG_USE_RX63_ERROR_INTERRUPT を 1 に設定することで、エラー割り込みを許可することができます。

2.8　コードサイズ

表 2-3 コードサイズに最新バージョンのモジュールを使用した場合のコードサイズを示します。

表 2-3 コードサイズ

MCU	使用メモリ	サイズ（注１、注２、注３、注４）
RX111	ROM	1,691 バイト
	RAM	59 バイト
	最大使用ユーザスタック	60 バイト
	最大使用割り込みスタック（注５）	80 バイト
RX231	ROM	1,691 バイト
	RAM	59 バイト
	最大使用ユーザスタック	60 バイト
	最大使用割り込みスタック（注５）	80 バイト
RX65N	ROM	1,782 バイト
	RAM	59 バイト
	最大使用ユーザスタック	64 バイト
	最大使用割り込みスタック（注５）	80 バイト
RX71M	ROM	1,768 バイト
	RAM	59 バイト
	最大使用ユーザスタック	64 バイト
	最大使用割り込みスタック（注５）	80 バイト

注１：「2.7 コンパイル時の設定」のデフォルト設定を選択した場合の値です。選択する定義により、コードサイズは異なります。

注２：動作条件は以下のとおりです。

- r_rspi_rx.c

注３：必要メモリサイズは、C コンパイラのバージョンやコンパイルオプションにより異なります。

注４：リトルエンディアン時の値です。エンディアンにより、上記のメモリサイズは、異なります。

2.9　引数

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

詳細は 2.13 API のデータ構造を参照してください。

2.10　戻り値

以下は API 関数が返す値です。

戻り値の型:　rspi_err_t

表 2-4 戻り値

値	原因
RSPI_SUCCESS	関数はエラーなく終了しました。
RSPI_ERR_BAD_CHAN	チャネル番号が無効です。
RSPI_ERR_CH_NOT_OPENED	チャネルはオープンされていません。関数は終了していません。
RSPI_ERR_CH_NOT_CLOSED	チャネルは前回のオープン以降、まだオープン状態です。
RSPI_ERR_UNKNOWN_CMD	コントロールコマンドが認識できません。
RSPI_ERR_INVALID_ARG	パラメータの引数が無効です。
RSPI_ERR_ARG_RANGE	パラメータの引数が有効な値の範囲を逸脱しています
RSPI_ERR_NULL_PTR	NULL ポインタを受け取りました。必要な引数がありません。
RSPI_ERR_LOCK	ロックに失敗しました。
RSPI_ERR_UNDEF	未定義/不明なエラー

2.11　コールバック関数

コールバック関数の定義は FIT 1.0 仕様の規則に従います。

a. コールバック関数は 1 個の引数 void * pdata を使用します。
b. コールバック関数を呼び出す前に、関数ポインタが有効かをチェックします。少なくともポインタの値は次の点に関してチェックされます。
   i. NULL ではないこと
   ii. FIT_NO_FUNC マクロに等しくないこと

2.11.1　コールバック関数のプロトタイプ宣言の例

```
void callback(void * pdata)
```

2.11.2　コールバック関数の呼び出し

転送動作が終わるごとに、ユーザが定義したコールバック関数が呼び出されます。これは転送動作を処理する割り込みハンドラの中で行われます。割り込みを発生するエラー条件は多くの場合は受信オーバランエラーですが、これもコールバック関数を呼び出します。呼び出しでは、チャネル番号とコールバックを呼び出す割り込みの結果コードが納められた構造体のポインタが、唯一の引数として渡されます。これらの情報の適切な処理はユーザアプリケーションで行います。コールバックは割り込みの中で処理され、その時点では割り込みは禁止されているため、更なるシステム割り込みを見逃さないためにもユーザ定義のコールバック関数をできるだけ早く終了することが強く推奨されます。

最も一般的なコールバック関数の使用方法は、アプリケーションにデータ転送が完了したことを伝えることです。これは転送開始直前にビジーフラグをセットしておき、このビジーフラグをコールバック内でクリアするという方法で実現できます。RTOS 環境では、セマフォまたは他のフラグや OS で提供されているメッセージサービスをコールバック内で使用することができます。

送信開始の例:

```
/* Conditions: Channel currently open. */
g_transfer_complete = false;
rspi_result = R_RSPI_WriteRead(handle, my_command_word, source, dest, length);
if (RSPI_SUCCESS != rspi_result)
{
    return error;
}

while (!g_transfer_complete) // Poll for interrupt callback to set this.
{
    nop();// Do something useful while waiting for the transfer to complete.
}
```

コールバック関数の例:

```
void my_callback(void * pdata)
{
    /* Examine the event to check for abnormal termination of transfer. */
    g_test_callback_event = (*(rspi_callback_data_t *)pdata).event_code;

    g_transfer_complete = true;
}
```

2.12 FIT モジュールの追加方法

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

(1) e² studio 上で Smart Configurator を使用して FIT モジュールを追加する場合
e² studio の Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e² studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(2) e² studio 上で FIT Configurator を使用して FIT モジュールを追加する場合
e² studio の FIT Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加することができます。詳細は、アプリケーションノート「RX ファミリ e² studio に組み込む方法 Firmware Integration Technology (R01AN1723)」を参照してください。

(3) CS+上で Smart Configurator を使用して FIT モジュールを追加する場合
CS+上で、スタンドアロン版 Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e² studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(4) CS+上で FIT モジュールを追加する場合
CS+上で、手動でユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」を参照してください。

2.13 API のデータ構造

このセクションではドライバの API 関数で使用されるデータ構造の詳細を説明します。

2.13.1 固有のデータ型

十分な型チェックでエラーを削減するため、API 関数で使用される多くのパラメータでは予め用意された型定義を使用する引数が渡される必要があります。使用できる値は公開インタフェースファイル r_rspi_rx_if.h で定義されています。以下は定義されている固有のデータ型です。

SPI バスインタフェースモードの列挙定義

型: rspi_interface_mode_t

値:	RSPI_IF_MODE_3WIRE	スレーブ選択に GPIO を使用
	RSPI_IF_MODE_4WIRE	RSPI により制御されるスレーブ選択信号を使用

マスタ動作またはスレーブ動作のモード設定の列挙定義

型: rspi_master_slave_mode_t

値:	RSPI_MS_MODE_MASTER	チャネルは SPI マスタとして動作
	RSPI_MS_MODE_SLAVE	チャネルは SPI スレーブとして動作

RSPI コントロールコマンドのコード

型: rspi_cmd_t

値:	RSPI_CMD_SET_BAUD	ビットレートの設定
	RSPI_CMD_ABORT	実行中の読み出しまたは書き込み動作の即時停止
	RSPI_CMD_SETREGS	単一操作で複数の RSPI レジスタの設定動作（高度な使用）

RSPI コントロールコマンドのデータ構造

R_RSPI_Control()関数の章を参照してください。

ハンドル

型: rspi_handle_t

値: ハンドルを格納するメモリの割り当てにはこの型を使用してください。この変数のアドレスは R_RSPI_Open()関数呼び出し時に渡す必要があります。ハンドルの値は R_RSPI_Open()関数によって自動的に割り当てられ、指定された場所に返されます。

オープン時のチャネル設定構造

R_RSPI_Open()関数はチャネルオープン時の動作モードの設定のために、この構造体の初期化されたインスタンスのポインタを必要とします。

型: rspi_chnl_settings_t

メンバ:	rspi_interface_mode_t	gpio_ssl;	インタフェースモードの指定
	rspi_master_slave_mode_t	master_slave_mode;	マスタ/スレーブモード動作の指定
	uint32_t bps_target;		チャネルに対するターゲットのビットレート

コールバック関数のデータ構造

ユーザが定義したコールバック関数にはチャネル番号と処理の結果コードがこのデータ構造体の形で渡されます。イベントコードについては 2.13.2 イベントコードを参照。

型: rspi_callback_data_t

メンバ:	rspi_handle_t	handle;	チャネルのハンドル
	rspi_evt_t	event_code;	イベントコード

2.13.2 イベントコード

API イベントとして返されるコード

戻り値の型: rspi_evt_t

値	原因
RSPI_EVT_TRANSFER_COMPLETE	データ転送が完了しました。
RSPI_EVT_TRANSFER_ABORTED	データ転送は中断されました。
RSPI_EVT_ERR_MODE_FAULT	モードフォルトエラー
RSPI_EVT_ERR_READ_OVF	リードオーバフロー
RSPI_EVT_ERR_PARITY	パリティエラー
RSPI_EVT_ERR_UNDER_RUN	アンダランエラー
RSPI_EVT_ERR_UNDEF	未定義／不明なエラーイベント

2.14　コマンド設定ワードで使われる列挙値の **Typedef**

このリストは、コマンドワードへの書き込みと読み出しで使われる固有設定値の列挙型の一覧です。コマンドワードはビットフィールドの集まりからなる 32 ビットの値で、有効なデータは下位 16 ビットである点にご注意ください。下位 16 ビットデータは、読み出し／書き込みの関数のいずれかを呼び出すたびに SPCMD レジスタにコピーされます。下位 16 ビットデータのコマンド全体を組み立てるには、個々の型のメンバを 1 個のみ選択し、rspi_command_word_t 構造体の対応するメンバに代入します。上位 16 ビットについては、ダミーデータ（RSPI_SPCMD_DUMMY）を設定します。

クロックの位相

CPHA（クロック位相）と CPOL（クロック極性）の組み合わせで SPI モード設定が決まります。

【注】 スレーブモード動作では、RSPI は偶数エッジでのサンプルのみをサポートします。これは SPI Mode-1 もしくは Mode-3 と称されるものに相当します。

型:　rspi_spcmd_cpha_t

メンバ:		
	RSPI_SPCMD_CPHA_SAMPLE_ODD	奇数エッジでデータサンプル、偶数エッジでデータ変化
	RSPI_SPCMD_CPHA_SAMPLE_EVEN	奇数エッジでデータ変化、偶数エッジでデータサンプル

クロックの極性

型:　rspi_spcmd_cpol_t

メンバ:		
	RSPI_SPCMD_CPOL_IDLE_LO	アイドル時の RSPCK がロー（L）
	RSPI_SPCMD_CPOL_IDLE_HI	アイドル時の RSPCK がハイ（H）

ビットレート分周比

SPI クロックベースのビットレート設定は更にこの設定で分周されます（注 1）。

型:　rspi_spcmd_br_div_t

メンバ:		
	RSPI_SPCMD_BR_DIV_1	ベースのビットレートを選択
	RSPI_SPCMD_BR_DIV_2	ベースのビットレートの 2 分周を選択
	RSPI_SPCMD_BR_DIV_4	ベースのビットレートの 4 分周を選択
	RSPI_SPCMD_BR_DIV_8	ベースのビットレートの 8 分周を選択

注 1 ：R_RSPI_Open()関数、または、R_RSPI_Control()関数で設定したビットレートは分周なし（RSPI_SPCMD_BR_DIV_1）を前提しています。設定したビットレートを分周したい場合は本ビットの設定を変更してください。

転送動作中にアサートされるスレーブセレクト信号

型:　rspi_spcmd_ssl_assert_t

メンバ:		
	RSPI_SPCMD_ASSERT_SSL0	SSL0 を選択
	RSPI_SPCMD_ASSERT_SSL1	SSL1 を選択
	RSPI_SPCMD_ASSERT_SSL2	SSL2 を選択
	RSPI_SPCMD_ASSERT_SSL3	SSL3 を選択

スレーブセレクトのネゲート

このビットは各フレームの後で RSPI がスレーブセレクト信号をネゲートするかこの信号レベルを保持するかを指定します。

型:　rspi_spcmd_ssl_negation_t

メンバ:		
	RSPI_SPCMD_SSL_NEGATE	転送終了時に全 SSL 信号をネゲート
	RSPI_SSPCMD_SL_KEEP	転送終了後から次アクセス開始まで SSL 信号レベルを保持

フレームデータ長

各 SPI データフレームのビット数

型:　rspi_spcmd_bit_length_t

メンバ:
- RSPI_SPCMD_BIT_LENGTH_8　データ長 8 ビット
- RSPI_SPCMD_BIT_LENGTH_9　データ長 9 ビット
- RSPI_SPCMD_BIT_LENGTH_10　データ長 10 ビット
- RSPI_SPCMD_BIT_LENGTH_11　データ長 11 ビット
- RSPI_SPCMD_BIT_LENGTH_12　データ長 12 ビット
- RSPI_SPCMD_BIT_LENGTH_13　データ長 13 ビット
- RSPI_SPCMD_BIT_LENGTH_14　データ長 14 ビット
- RSPI_SPCMD_BIT_LENGTH_15　データ長 15 ビット
- RSPI_SPCMD_BIT_LENGTH_16　データ長 16 ビット
- RSPI_SPCMD_BIT_LENGTH_20　データ長 20 ビット
- RSPI_SPCMD_BIT_LENGTH_24　データ長 24 ビット
- RSPI_SPCMD_BIT_LENGTH_32　データ長 32 ビット

データ転送時のビット順序

型:　rspi_spcmd_bit_order_t

メンバ:
- RSPI_SPCMD_ORDER_MSB_FIRST　MSB ファースト
- RSPI_SPCMD_ORDER_LSB_FIRST　LSB ファースト

RSPI 信号遅延

型:　rspi_spcmd_spnden_t　次アクセス遅延

メンバ:
- RSPI_SPCMD_NEXT_DLY_1　次アクセス遅延は 1 RSPCK +2 PCLK
- RSPI_SPCMD_NEXT_DLY_SSLND　次アクセス遅延は RSPI 次アクセス遅延レジスタ（SPND）の設定値

型:　rspi_spcmd_slnden_t

メンバ:
- RSPI_SPCMD_SSL_NEG_DLY_1　SSL ネゲート遅延は 1 RSPCK
- RSPI_SPCMD_SSL_NEG_DLY_SSLND　SSL ネゲート遅延は RSPI スレーブセレクトネゲート遅延レジスタ（SSLND）の設定値

型:　rspi_spcmd_sckden_t

メンバ:
- RSPI_SPCMD_CLK_DLY_1　RSPCK 遅延は 1 RSPCK
- RSPI_SPCMD_CLK_DLY_SPCKD　RSPCK 遅延は RSPI クロック遅延レジスタ（SPCKD）の設定値

ダミーデータ

型:　rspi_spcmd_dummy_t

メンバ:
- RSPI_SPCMD_DUMMY　上位 16 ビットのダミーデータ

2.14.1 コマンドワード全体のデータ構造

以下のコマンドワードは、SPCMD レジスタの全ビットをセットするため上記のそれぞれの型を 1 個ずつ正しい順序で保持します。

```
#pragma bit_order right      // Match the order of description in the hardware manual.
typedef union rspi_command_word_s
{
      struct{
            rspi_spcmd_cpha_t             cpha              :1;
            rspi_spcmd_cpol_t             cpol              :1;
            rspi_spcmd_br_div_t           br_div            :2;
            rspi_spcmd_ssl_assert_t       ssl_assert        :3;
            rspi_spcmd_ssl_negation_t     ssl_negate        :1;
            rspi_spcmd_bit_length_t       bit_length        :4;
            rspi_spcmd_bit_order_t        bit_order         :1;
            rspi_spcmd_spnden_t           next_delay        :1;
            rspi_spcmd_slnden_t           ssl_neg_delay     :1;
            rspi_spcmd_sckden_t           clock_delay       :1;
            rspi_spcmd_dummy_t            dummy             :16
      };
   uint16_t word[2];
   };
} rspi_command_word_t;
```

コマンドワードのインスタンスの初期化例

```
static const rspi_command_word_t my_command_reg_word = {
       RSPI_SPCMD_CPHA_SAMPLE_ODD,
       RSPI_SPCMD_CPOL_IDLE_LO,
       RSPI_SPCMD_BR_DIV_1,
       RSPI_SPCMD_ASSERT_SSL0,
       RSPI_SPCMD_SSL_KEEP,
       RSPI_SPCMD_BIT_LENGTH_8,
       RSPI_SPCMD_ORDER_MSB_FIRST,
       RSPI_SPCMD_NEXT_DLY_SSLND,
       RSPI_SPCMD_SSL_NEG_DLY_SSLND,
       RSPI_SPCMD_CLK_DLY_SPCKD,
       RSPI_SPCMD_DUMMY,
};
```

3. API 関数

3.1 R_RSPI_Open()

この関数は RSPI チャネルに周辺モジュールクロックを供給し、関連レジスタの初期化と割り込みの許可を行い、他の API 関数で使用されるチャネルのハンドルを返します。

Format

```
rspi_err_t   R_RSPI_Open(uint8_t               channel,
                         rspi_chnl_settings_t  *pconfig,
                         void                 (*pcallback)(void *pcbdat),
                         rspi_handle_t         *phandle);
```

Parameters

channel	初期化される RSPI チャネルの番号
**pconfig*	RSPI チャネル設定データ構造体のポインタ
*(*pcallback)(void *pcbdat)*	割り込みから呼び出されるユーザ定義関数のポインタ
**phandle*	チャネルのハンドルへのポインタ。この関数でハンドル値が設定されます

Return Values

RSPI_SUCCESS	チャネルは正常に初期化されました。
RSPI_ERR_BAD_CHAN	チャネル番号が有効な値ではありません。
RSPI_ERR_CH_NOT_CLOSED	チャネルは動作中です。最初に R_RSPI_Close()関数を実行してください。
RSPI_ERR_NULL_PTR	*pconfig または*phandle ポインタが NULL です。
RSPI_ERR_INVALID_ARG	*pconfig 構造体の要素が無効な値です。
RSPI_ERR_LOCK	リソースをロックできませんでした。

Properties

r_rspi_rx_if.h ファイルにプロトタイプ宣言されています。

Description

オープン関数は RSPI チャネル動作の準備を行います。この関数は他の RSPI API 関数（R_RSPI_GetVersion を除く）を呼び出す前に呼び出す必要があります。正常に終了すると、指定された RSPI チャネルのステータスは open 状態にセットされます。その後は、R_RSPI_Close()関数コールでチャネルが close 状態になるまでは、その RSPI チャネルでオープン関数を再び呼び出すことはできません。

本処理が終了した時点ではまだ通信ができません。入出力ポートの MPC と PMR を周辺機能に設定してください。

Reentrant

リエントラントです。他のチャネルに対しては再入可能です。同じチャネルではロックエラーとなります。

Example

```
/* Conditions: Channel not yet open. */
uint8_t chan = 0;
rspi_handle_t handle;
rspi_chnl_settings_t my_config;
rspi_cmd_baud_t my_setbaud_struct;
rspi_err_t rspi_result;

my_config.gpio_ssl         = RSPI_4WIRE_MODE;
my_config.master_slave_mode = RSPI_MASTER_MODE;
my_config.bps_target       = 4000000; // Bit rate in bits-per-second.

rspi_result = R_RSPI_Open(chan, &my_config, &test_callback, &handle );

if (RSPI_SUCCESS != rspi_result)
{
   return rspi_result;
}

/* Initialize I/O port pins for use with the RSPI peripheral.
 * This is specific to the MCU and ports chosen. */
 rspi_64M_init_ports();
```

3.2 R_RSPI_Control()

Control 関数は RSPI チャネルに固有のハードウェアまたはソフトウェアの操作を行います。

Format

```
rspi_err_t     R_RSPI_Control(rspi_handle_t  handle,
                              rspi_cmd_t     cmd,
                              void          *pcmd_data);
```

Parameters

handle　チャネルのハンドル

cmd　実行されるコマンドコード

**pcmd_data*　個々のコマンドの実行に必要な固有のデータの場所を参照するために使用される、コマンドデータ構造体のパラメータのポインタで、void ポインタ型と規定されています。データを必要としないコマンドでは値として FIT_NO_PTR を使用します。

Return Values

RSPI_SUCCESS	コマンドは正常に終了しました。
RSPI_ERR_CH_NOT_OPEN	チャネルは未だオープンされていません。最初に R_RSPI_Open()関数を実行してください。
RSPI_ERR_BAD_CHAN	チャネル番号は有効な値ではありません。
RSPI_ERR_UNKNOWN_CMD	コントロールコマンドが認識できない値です。
RSPI_ERR_NULL_PTR	*pcmd_data または*phandle ポインタが NULL です。
RSPI_ERR_INVALID_ARG	*pcmd_data 構造体の要素が無効な値を含んでいます。
RSPI_ERR_LOCK	リソースをロックできません。

Properties

r_rspi_rx_if.h ファイルにプロトタイプ宣言されています。

Description

コントロール関数は RSPI チャネルに固有のハードウェアまたはソフトウェアの操作を行います。この関数は指定された RSPI チャネルを示す RSPI ハンドル、実行される操作を選択するコマンドの列挙値、および操作を行う上で必要なデータが格納される場所へのポインタ（void ポインタ型）を引数としています。このポインタでは、コマンドに応じて r_rspi_rx_if.h 内に用意されている適切な型を呼び出し時に使用できるよう、ポインタの型をキャストしています。

コマンド	引数 pcmd_data	内容
RSPI_CMD_SET_BAUD	rspi_cmd_baud_t *	RSPI チャネルを再度初期化することなく、ビットレート設定を変更します。
RSPI_CMD_ABORT	FIT_NO_PTR	実行中の読み出しまたは書き込み動作を直ちに中断します。
RSPI_CMD_SETREGS	rspi_cmd_setregs_t *	1 回の操作でサポートされている RSPI レジスタのすべてに対する設定を行います。これには高度な知識が必要です。

Reentrant

ロック機能が有効な場合はリエントラントです。同じ RSPI チャネルの操作に再入した場合には BSP ロックにより拒否されます。ロック機能が無効の場合は、再入は異なる RSPI チャネルに対してのみ安全に行えます。ロック機能が有効なときには、デッドロックを生じないように注意が必要です。呼び出し時には関数の戻り値をチェックし、ロックエラーの場合には、そのチャネルを使用中のプロセスの実行が終了するようにします。

Example

```
my_setbaud_struct.bps_target = 12000000; // Set for 12 Mbps
rspi_result = R_RSPI_Control(handle, RSPI_CMD_SET_BAUD,  &my_setbaud_struct);
if (RSPI_SUCCESS != rspi_result)
{
    return error;
}
/* This is taking too long, stop the current transfer now! */
rspi_result = R_RSPI_Control(handle, RSPI_CMD_ABORT,  FIT_NO_PTR);
```

Special Notes:

以下はコントロール関数のコマンドコードです。

```
typedef enum rspi_cmd_e
{
    RSPI_CMD_SET_BAUD = 1,
    RSPI_CMD_ABORT,     // Stop the current read or write operation immediately.
    RSPI_CMD_SETREGS,   // Set all supported RSPI regs in one operation.
} rspi_cmd_t;
```

ビットレート設定コマンドのデータ構造体です。このコマンドは、指定されたチャネルのビットレートを設定します。bps_target で指定された値がそのまま設定されない場合もあります。関数は設定が適合することを確認し、指定されたビットレートが分周回路で実現できなければ、次に低い、使用可能なビットレートを設定します。なお、SPCMD.BRDV[1:0]ビットは 0（分周なし）を前提としています。

```
typedef struct rspi_cmd_baud_s
{
    uint32_t    bps_target; // The target bits-per-second setting for the channel.
} rspi_cmd_baud_t;
```

RSPI_CMD_SETREGS コマンドを使用することで、ドライバが保持している複数の RSPI レジスタ設定情報を変更することができます。このコマンドでは、RSPI_CMD_SETREGS コマンドを利用するには、必要に応じた設定値を持つインスタンスを先ず作り、R_RSPI_Control()呼び出しでそのポインタを引数として渡します。

ただし、RSPI_CMD_SETREGS コマンドを実行した時点では RSPI レジスタの値は変更されません。本処理実行後、R_RSPI_Close()関数をコールし、R_RSPI_Open()関数をコールすることで初めて RSPI レジスタの値が変更されます。

```
typedef struct rspi_cmd_setregs_s
{
    uint8_t sslp_val;   /* RSPI Slave Select Polarity Register (SSLP) */
    uint8_t sppcr_val;  /* RSPI Pin Control Register (SPPCR) */
    uint8_t spckd_val;  /* RSPI Clock Delay Register (SPCKD) */
    uint8_t sslnd_val;  /* RSPI Slave Select Negation Delay Register (SSLND) */
    uint8_t spnd_val;   /* RSPI Next-Access Delay Register (SPND) */
    uint8_t spcr2_val;  /* RSPI Control Register 2 (SPCR2) */
    uint8_t spdcr2_val; /* RSPI Data Control Register 2 (SPDCR2) */
} rspi_cmd_setregs_t;
```

3.3 R_RSPI_Close()

ハンドルで指定された RSPI チャネルを、完全に無効にします。

Format

```
RSPI_err_t      R_RSPI_Close(rspi_handle_t handle);
```

Parameters

handle　チャネルのハンドル

Return Values

RSPI_SUCCESS	チャネルは正常に閉じられました。
RSPI_ERR_CH_NOT_OPEN	チャネルはオープンされていないため、クローズ指示は意味を持ちません。
RSPI_ERR_BAD_CHAN	チャネル番号が有効な値ではありません。
RSPI_ERR_NULL_PTR	必要なポインタ引数が NULL です。

Properties

r_rspi_rx_if.h ファイルにプロトタイプ宣言されています。

Description

この関数はハンドルで指定された RSPI チャネルを無効にします。RSPI ハンドルはチャネルが open 状態ではないことを示すために変更されます。この RSPI チャネルは R_RSPI_Open()関数で再度オープンされるまで使用できません。オープン状態ではない RSPI チャネルで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

リエントラントです。ただし、意図せずに同じ RSPI チャネルに対して再入すると、予期していない RSPI_ERR_NOT_OPEN コードが返されることがあります。

Example

```
RSPI_err_t  rspi_result;

rspi_result = R_RSPI_Close(handle);

if (RSPI_SUCCESS != rspi_result)
{
    return rspi_result;
}
```

3.4 R_RSPI_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

```
rspi_err_t     R_RSPI_Write(rspi_handle_t      handle,
                            rspi_command_word_t spcmd_command_word,
                            void               *psrc,
                            uint16_t            length);
```

Parameters

handle　チャネルのハンドル

spcmd_command_word

このビットフィールドデータはこの動作を行うための SPCMD レジスタ設定のすべてを含んでいます。2.14 コマンド設定ワードで使われる列挙値の Typedef をご覧ください。

**psrc*　SPI デバイスに送信されるデータが格納されているソースデータバッファの void 型のポインタ。引数は NULL にならないようにしてください。*psrc ポインタは転送時に、spcmd_command_word.bit_length で指定されたビット長のデータフレームに対応するデータ型にキャストされます。例えば、ビット長が 16 ビットに設定されているときには、ソースバッファデータは 16 ビットのデータブロックとしてアクセスされます。各ビット長についても同様にキャストが行われます。ビット長の設定が 8 ビット、16 ビット、または 32 ビットのいずれでもないときには、このビット長を格納できるデータ型が使用されます。例えば、24 ビットフレームは 32 ビットのメモリに格納されます。また、11 ビットフレームは 16 ビットのメモリに格納されます。

length　転送されるデータフレームの数を指定する転送長の変数。データフレームは、spcmd_command_word.bit_length の設定に応じて決まります。length 引数はソースデータのメモリ上の型に一致するようにしてください。これは、データフレームの数を示すものであり、バイト数を示すものではありません。

Return Values

RSPI_SUCCESS	書き込み動作は正常に終了しました。
RSPI_ERR_CH_NOT_OPEN	チャネルはオープンされていません。最初に R_RSPI_Open()関数を実行してください。
RSPI_ERR_BAD_CHAN	チャネル番号が無効です。
RSPI_ERR_NULL_PTR	必要とされるポインタ引数の値が NULL です。
RSPI_ERR_LOCK	リソースをロックできませんでした。チャネルはビジーです。

Properties

r_rspi_rx_if.h ファイルにプロトタイプ宣言されています。

Description

SPI デバイスへのデータ送信を開始します。データ送信を開始するとすぐに戻り値を返します。以降、length 引数に設定した数のデータフレーム送信が完了するまで、割り込みを起動要因としてバックグラウンドでデータ送信を繰り返します。データ送信が完了した時、ユーザ定義のコールバック関数がコールされます。このコールバック関数は、データ送信が完了したことをユーザアプリケーションに通知するために使用してください。

Write 関数の処理は、RSPI がマスタモード、もしくは、スレーブモードを選択しているかで若干異なります。RSPI がスレーブモードを選択している場合、マスタからクロックを受信した時のみデータを送信します。この際、通信は全二重で行われるため、データを受信します。Write 関数はデータ送信のみを行うため、受信したデータは破棄します。

Reentrant

リエントラントです。他のチャネルに対しては再入可能です。同じチャネルではロックエラーとなります。

Example

```
/* Conditions: Channel currently open. */
g_transfer_complete = false;

rspi_result = R_RSPI_Write(handle, my_command_word, source, length);
if (RSPI_SUCCESS != rspi_result)
{
   if (RSPI_ERR_LOCK == rspi_result)
    {
       // Channel must be busy. Try again later.
    }
    return error;
}

while (!g_transfer_complete) // Poll for interrupt callback to set this.
{
    nop(); // Do something useful while waiting for the transfer to complete.
}
```

3.5 R_RSPI_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

```
rspi_err_t     R_RSPI_Read(rspi_handle_t        handle,
                           rspi_command_word_t  spcmd_command_word,
                           void                 *pdest,
                           uint16_t             length);
```

Parameters

handle　チャネルのハンドル

spcmd_command_word

このビットフィールドデータはこの動作を行うための SPCMD レジスタ設定のすべてを含んでいます。をご覧ください。

*pdest　SPI デバイスから受信したデータが格納されるアクセス先のバッファの void 型のポインタ。呼び出し側は、要求されたデータ数を格納することができるスペースを確実に用意する必要があります。引数は NULL にならないようにしてください。*pdest ポインタは転送時に、spcmd_command_word.bit_length で指定されたビット長のデータフレームに対応するデータ型にキャストされます。例えば、ビット長が 16 ビットに設定されているときには、データはアクセス先のバッファで 16 ビット値として格納されます。各ビット長についても同様にキャストされます。ビット長の設定が 8 ビット、16 ビット、または 32 ビットのいずれでもないときには、このビット長を格納できる最も小さいデータ型が使用されます。例えば、24 ビットフレームは 32 ビットのメモリに格納されています。また、11 ビットフレームは 16 ビットのメモリに格納されます。

length　転送されるデータフレームの数を指定する転送長の変数。データフレームのサイズは、spcmd_command_word.bit_length の設定に応じて決まります。length 引数はソースデータのメモリ上の型に一致するようにしてください。これはデータフレームの数を示すものであり、バイト数を示すものではありません。

Return Values

RSPI_SUCCESS	読み込み動作は正常に完了しました。
RSPI_ERR_CH_NOT_OPEN	チャネルはオープンされていません。最初に R_RSPI_Open()関数を実行してください。
RSPI_ERR_BAD_CHAN	チャネル番号が無効です。
RSPI_ERR_NULL_PTR	必要とされるポインタ引数の値が NULL です。
RSPI_ERR_LOCK	リソースをロックできませんでした。チャネルはビジーです。

Properties

r_rspi_rx_if.h ファイルにプロトタイプ宣言されています。

Description

SPI デバイスへのデータ受信を開始します。データ受信を開始するとすぐに戻り値を返します。以降、length 引数に設定した数のデータフレーム受信が完了するまで、割り込みを起動要因としてバックグラウンドでデータ受信を繰り返します。受信データは、*pdest に設定したバッファに格納されます。データ受信が完了した時、ユーザ定義のコールバック関数がコールされます。このコールバック関数は、データ受信が完了したことをユーザアプリケーションに通知するために使用してください。

Read 関数の処理は、RSPI がマスタモード、もしくは、スレーブモードを選択しているかで若干異なります。RSPI がスレーブモードを選択している場合、マスタからクロックを受信した時のみデータを受信します。この際、通信は全二重で行われるため、ダミーデータを送信します。ダミーデータの値は#define RSPI_CFG_DUMMY_TXDATA で設定した値になります。

Reentrant

リエントラントです。他のチャネルに対しては再入可能です。同じチャネルではロックエラーとなります。

Example

```
/* Conditions: Channel currently open. */
g_transfer_complete = false;

rspi_result = R_RSPI_Read(handle, my_command_word, dest, length);
if (RSPI_SUCCESS != rspi_result)
{
   return error;
}

while (!g_transfer_complete) // Poll for interrupt callback to set this.
{
    nop(); // Do something useful while waiting for the transfer to complete.
}
```

3.6 R_RSPI_WriteRead()

Write Read 関数は SPI デバイスにデータを送信し、同時に SPI デバイスからデータを受信します。

Format

```
rspi_err_t     R_RSPI_WriteRead(rspi_handle_t        handle,
                                rspi_command_word_t  spcmd_command_word,
                                void                 *psrc,
                                void                 *pdest,
                                uint16_t             length);
```

Parameters

handle チャネルのハンドル

spcmd_command_word

このビットフィールドデータはこの動作を行うための SPCMD レジスタ設定のすべてを含んでいます。2.14 コマンド設定ワードで使われる列挙値の Typedef をご覧ください。

**psrc* SPI デバイスに送信されるデータが格納されているソースデータバッファの void 型のポインタ。引数は NULL にならないようにしてください。*psrc ポインタは転送時に、spcmd_command_word.bit_length で指定されたビット長のデータフレームに対応するデータ型にキャストされます。例えば、ビット長が 16 ビットに設定されているときには、ソースバッファデータは 16 ビットのデータブロックとしてアクセスされます。各ビット長についても同様にキャストが行われます。ビット長の設定が 8 ビット、16 ビット、または 32 ビットのいずれでもないときには、このビット長を格納できるデータ型が使用されます。例えば、24 ビットフレームは 32 ビットのメモリに格納されます。また、11 ビットフレームは 16 ビットのメモリに格納されます。

**pdest* SPI デバイスから受信したデータが格納されるアクセス先のバッファの void 型のポインタ。呼び出し側は、要求されたデータ数を格納することができるスペースを確実に用意する必要があります。引数は NULL にならないようにしてください。*pdest ポインタは転送時に、spcmd_command_word.bit_length で指定されたビット長のデータフレームに対応するデータ型にキャストされます。例えば、ビット長が 16 ビットに設定されているときには、データはアクセス先のバッファで 16 ビット値として格納されます。各ビット長についても同様にキャストされます。ビット長の設定が 8 ビット、16 ビット、または 32 ビットのいずれでもないときには、このビット長を格納できる最も小さいデータ型が使用されます。例えば、24 ビットフレームは 32 ビットのメモリに格納されています。また、11 ビットフレームは 16 ビットのメモリに格納されます。

length 転送されるデータフレームの数を指定する転送長の変数。データフレームのサイズは、spcmd_command_word.bit_length の設定に応じて決まります。length 引数はソースデータのメモリ上の型に一致するようにしてください。これは、データフレームの数を示すものであり、バイト数を示すものではありません。

Return Values

RSPI_SUCCESS 読み込み動作は正常に完了しました。

RSPI_ERR_CH_NOT_OPEN チャネルはオープンされていません。最初に R_RSPI_Open()関数を実行してください。

RSPI_ERR_BAD_CHAN チャネル番号は無効です。

RSPI_ERR_NULL_PTR 必要とされるポインタ引数の値が NULL です。

RSPI_ERR_LOCK リソースをロックできませんでした。チャネルはビジーです。

Properties

r_rspi_rx_if.h ファイルにプロトタイプ宣言されています。

Description

SPI デバイスへのデータ送受信を開始します。データ送受信を開始するとすぐに戻り値を返します。以降、length 引数に設定した数のデータフレーム送受信が完了するまで、割り込みを起動要因としてバックグラウンドでデータ送受信を繰り返します。データ送受信が完了した時、ユーザ定義のコールバック関数がコールされます。このコールバック関数は、データ送受信が完了したことをユーザアプリケーションに通知するために使用してください。

Write Read 関数の処理は、RSPI がマスタモード、もしくは、スレーブモードを選択しているかで若干異なります。RSPI がスレーブモードを選択している場合、マスタからクロックを受信した時のみデータを送受信します。

Reentrant

リエントラントです。他のチャネルに対しては再入可能です。同じチャネルではロックエラーとなります。

Example

```
/* Conditions: Channel currently open. */
g_transfer_complete = false;
rspi_result = R_RSPI_WriteRead(handle, my_command_word, source, dest, length);
if (RSPI_SUCCESS != rspi_result)
{
    return error;
}

while (!g_transfer_complete) // Poll for interrupt callback to set this.
{
    nop();// Do something useful while waiting for the transfer to complete.
}
```

3.7 R_RSPI_GetVersion()

この関数は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_RSPI_GetVersion(void);
```

Parameters

なし

Return Values

バージョン番号。メジャーバージョン番号とマイナーバージョン番号が 1 個の 32 ビット値に格納されています。

Properties

r_rspi_rx_if.h ファイルにプロトタイプ宣言されています。

Description

この関数はこのモジュールのバージョン番号を返します。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

Example

```
/* Retrieve the version number and convert it to a string. */

uint32_t   version, version_high, version_low;
char       version_str[9];

version = R_RSPI_GetVersion();

version_high = (version >> 16)&0xf;
version_low  =  version & 0xff;

sprintf(version_str, "RSPIv%1.1hu.%2.2hu", version_high, version_low);
```

4. 端子設定

RSPI FIT モジュールを使用するためには、マルチファンクションピンコントローラ（MPC）で周辺機能の入出力信号を端子に割り付ける（以下、端子設定と称す）必要があります。端子設定は、R_RSPI_Open 関数を呼び出した後に行ってください。

e² studio の場合は「FIT Configurator」または「Smart Configurator」の端子設定機能を使用することができます。FIT Configurator、Smart Configurator の端子設定機能を使用すると、端子設定画面で選択したオプションに応じて、ソースファイルが出力されます。そのソースファイルで定義された関数を呼び出すことにより端子を設定できます。詳細は表 4-1 を参照してください。

表 4-1「Smart Configurator」または「FIT Configurator」が出力する関数一覧

選択したオプション	出力される関数名	r_smstr_rx_pin_config.h に出力されるマクロ定義
チャネル 0	R_RSPI_PinSet_RSPI0()	
チャネル 1	R_RSPI_PinSet_RSPI1()	
チャネル 2	R_RSPI_PinSet_RSPI2()	

ただし、3 線インタフェースモードを使用する場合、スレーブセレクト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。

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 RSPI モジュールの基本使用をデモンストレーションするためのサンプルプログラムがいくつか記載されています。サンプルプログラムは、よく呼び出す API 関数の機能を簡単に説明するためのものです。

サンプルアプリケーションは、ジャンパ配線によってマスタ出力データをマスタ入力データにルーティングすることで、全二重転送（同時送受信）をシミュレートします。受信データは、送信データと一致することを確認するためにテストされます。RSPI モジュールのバージョン番号が取得され、必要に応じてルネサス仮想デバッグコンソールのウィンドウに表示できます。

5.1 ワークスペースへのサンプルプログラム追加

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実行

1. ターゲットボードに応じて MOSIA 端子を MISOA 端子にジャンパ接続し、ボードを準備します。
   a) RSKRX113
      i) 拡張ヘッダ J3 ピン 24 を J3 ピン 23 に接続します。
   b) RSKRX64M および RSKRX71M
      i) ボードのジャンパ J14 と J12 からジャンパプラグを取り外します。
      ii) J14 ピン 2 を J12 ピン 2 に接続します。
   c) RSKRX231
      i) 拡張ヘッダ J3 ピン 14 を J3 ピン 13 に接続します。
   d) RSKRX65N
      i) 拡張ヘッダ J13 ピン 2 を J11 ピン 2 に接続します。
   e) RSKRX65N-2MB
      i) 拡張ヘッダ JA3 ピン 7 を JA3 ピン 8 に接続します。
      ii) SW4 ピン 3 と SW4 ピン 4 を OFF 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

表 6-1 に動作確認環境を示す。

表 6-1 動作確認環境

項目	内容
統合開発環境	ルネサス エレクトロニクス製 e^2 studio V6.0.0
Cコンパイラ	ルネサス エレクトロニクス製 C/C++ compiler for RX family V.2.07.00
	コンパイルオプション : 統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン／リトルエンディアン
モジュールのバージョン	Ver.1.70
使用ボード	Renesas Starter Kit for RX130(RTK5005130SxxxxxBE) Renesas Starter Kit for RX130-512KB(RTK5051308SxxxxxBE) Renesas Starter Kit for RX24T(RTK500524TSxxxxxBE) Renesas Starter Kit for RX24U(RTK500524USxxxxxBE) Renesas Starter Kit for RX65N(RTK500565NSxxxxxBE) Renesas Starter Kit for RX65N-2MB(RTK50565N2SxxxxxBE)

6.2 トラブルシューティング

(1) Q:本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「Could not open source file "platform.h"」エラーが発生します。

A：FIT モジュールがプロジェクトに正しく追加されていない可能性があります。プロジェクトへの追加方法をご確認ください。

- CS+を使用している場合
  アプリケーションノート RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」
- e² 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 e² studio に組み込む方法 Firmware Integration Technology (R01AN1723)」

また、本 FIT モジュールを使用する場合、ボードサポートパッケージ FIT モジュール(BSP モジュール)もプロジェクトに追加する必要があります。BSP モジュールの追加方法は、アプリケーションノート「ボードサポートパッケージモジュール(R01AN1685)」を参照してください。

(2) Q:本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「This MCU is not supported by the current r_rspi_rx module.」エラーが発生します。

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

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
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情報

本モジュールには、以下のテクニカルアップデートが適用されています。

- TN-RX*-A147A/J
  テクニカルアップデートでは、割り込みなしにすべてのデータ転送の完了を確認する方法について記述しています。
  RSPI ドライバは転送の完了時に割り込みを使用するため、テクニカルアップデートの内容は適用されません。

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

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2013.11.15	-	初版発行
1.20	2014.04.04	-	サポート対象／テスト済み MCU のリストを更新
1.30	2015.01.20	-	サポート対象／テスト済み MCU のリストを更新
		8	2.9「コードサイズと RAM 使用」を追加
		32	「デモプロジェクト」セクションを追加 フォント修正
1.40	2015.06.29	1, 3, 9, 33	RX231 のサポート内容を更新
1.50	2016.09.30	1	RX65N、RX130、RX230、RX23T、RX24T のサポート内容を更新
		16	API 関数の章番号を 6 から 3 へ変更
		34	6.4「データ出力と RAM の関係」を追加
1.60	2017.03.31	-	日本語版のアプリケーションノートを公開
		1	RX24U を追加
		6	2.3「動作確認環境」の内容を更新
		8	2.7「サポートされているツールチェイン」の内容を更新
		8	2.10「コードサイズと RAM サイズ」の内容を更新
		41	7「サンプルプログラム」、7.1「ワークスペースへのサンプルプログラム追加」、7.2「サンプルプログラム実行」のタイトルと内容を更新
		42	8「付録」を追加
1.70	2017.07.31	-	次の章を追加した。 -2.4 使用する割り込みベクタ。 -2.7 コンパイル時の設定。 -2.9 引数 -2.10 戻り値 -2.11 コールバック関数 -2.12 FIT モジュールの追加方法 -4. 端子設定 -6.2 トラブルシューティング 次の章を移動した。 -1.5 データ転送動作：元は 6. データ転送動作であった。 -1.2.1 ドライバでサポートされている RSPI の機能：元は 2.5 ドライバでサポートされている RSPI の機能であった。 -1.2.2 サポートされていない RSPI の機能：元は 2.6 サポートされていない RSPI の機能であった。 次の章名を変更した。 -2.8 コードサイズ：元は 2.10 コードサイズと RAM サイズであった。 次の章の内容を移動した。 -2.1 API の概要：元は 3.1 概要であった。 次の章の内容を変更した。 -5.2 サンプルプログラム実行 -6.1 動作確認環境 次の記述を削除した。 -2.2 ソフトウェアの要求から、cgc に関する記述。
		1	対象デバイスに、RX651 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. 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. リザーブアドレスのアクセス禁止

【注意】リザーブアドレス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、事前に問題ないことをご確認下さい。

同じグループのマイコンでも型名が違うと、内部メモリ、レイアウトパターンの相違などにより、特性が異なる場合があります。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。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、その他の不適切に使用しないでください。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、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、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損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を、(1)核兵器、化学兵器、生物兵器等の大量破壊兵器およびこれらを運搬することができるミサイル（無人航空機を含みます。）の開発、設計、製造、使用もしくは貯蔵等の目的、(2)通常兵器の開発、設計、製造または使用の目的、または(3)その他の国際的な平和および安全の維持の妨げとなる目的で、自ら使用せず、かつ、第三者に使用、販売、譲渡、輸出、賃貸もしくは使用許諾しないでください。
   当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様の転売、貸与等により、本書（本ご注意書きを含みます。）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、当社は一切その責任を負わず、お客様にかかる使用に基づく当社への請求につき当社を免責いただきます。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載された情報または当社製品に関し、ご不明点がある場合には、当社営業にお問い合わせください。

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。

(Rev.3.0-1　2016.11)

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：https://www.renesas.com/contact/