# RX ファミリ

## RI3C 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) を使用した Renesas I3C (Improved Inter-Integrated Circuit) モジュールについて説明します。I3C 通信インタフェースを用いたデバイス間の通信を行うには本モジュールが必要です。

### 対象デバイス

- RX26T 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 動作確認環境詳細」を参照してください。

## Contents

# 1. 概要

Firmware Integration Technology (FIT) を用いた Renesas I3C モジュール (RI3C FIT モジュール) は、コントローラとターゲットの間で RI3C を用いてデータを送受信するための手段を提供します。RI3C は MIPI I3C に準拠しています。

**制限事項**

- 本モジュールは HDR (I3C High Data Rate) モードをサポートしていません。

## 1.1 Renesas I3C FIT モジュール

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 モジュールの追加方法」を参照してください。

## 1.2 API の概要

表 1.1 API 関数を示します。

**表 1.1 API 関数一覧**

| 関数 | 関数説明 |
|---|---|
| R_RI3C_Open() | この関数は RI3C インスタンスを設定します。 |
| R_RI3C_Enable() | この関数は RI3C デバイスを有効にします。 |
| R_RI3C_DeviceCfgSet() | この関数はデバイスのコンフィグレーションを行います。 |
| R_RI3C_ControllerDeviceTableSet() | この関数はコントローラデバイステーブルにエントリを設定します。 |
| R_RI3C_TargetStatusSet() | この関数は、GETSTATUS コマンドが実行された際にコントローラに返すステータスを設定します。 |
| R_RI3C_DeviceSelect() | コントローラモードでは、この関数は次の送信先となるデバイスを選択します。 |
| R_RI3C_DynamicAddressAsignmentStart() | この関数はダイナミックアドレス割り当てプロセスを開始します。 |
| R_RI3C_CommandSend() | この関数は、バス上のターゲットへブロードキャストコマンドまたはダイレクトコマンドを送信します。 |
| R_RI3C_Write() | この関数は、転送時に使用するライトバッファを設定します。コントローラモードでこの関数が実行されると転送が開始します。転送完了時にこの関数はストップコンディションまたはリスタートコンディションを送信します。 |
| R_RI3C_Read() | この関数は、転送時に使用するリードバッファを設定します。コントローラモードでこの関数が実行されると転送が開始します。転送完了時にこの関数はストップコンディションまたはリスタートコンディションを送信します。 |
| R_RI3C_IbiWrite() | この関数は IBI 書き込み動作を開始します。 |
| R_RI3C_IbiRead() | この関数は、受信した IBI データを格納するためのリードバッファを設定します。 |
| R_RI3C_Close() | この関数は RI3C インスタンスをクローズします。 |

## 1.3 RI3C FIT モジュールの概要

### 1.3.1 RI3C FIT モジュールの仕様

1. 本モジュールは、コントローラの送信と受信、およびターゲットの送信と受信をサポートします。
2. 本モジュールは、FIFO 転送の SDR (I3C Single Data Rate) モードをサポートします。
3. RI3C は、データとコマンドが書き込まれると自律的に転送を開始します。
4. 次の状況で割り込みが発生します。1) レスポンスキューフル (RESPI)、2) コマンドキューエンプティ (CMDI)、3) IBI キューエンプティ／フル (IBII)、4) 受信ステータスキューフル (RCVI)、5) 受信データフル (RXI)、6) 送信データエンプティ (TXI)、7) 通信エラー／通信イベント (EEI)
5. コントローラは 7 ビットアドレスにより複数のターゲットと通信できます。

## 1.4 RI3C モジュールの使用方法

### 1.4.1 RI3C FIT モジュールを C++プロジェクト内で使用する方法

C++プロジェクトでは、RI3C FIT 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_ri3c_rx_if.h”
}
```

# 2. API 情報

本モジュールの API はルネサスの API の命名基準に従っています。

## 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。

- RI3C モジュール

## 2.2 ソフトウェアの要求

このドライバは以下のパッケージに依存しています。

- ボードサポートパッケージモジュール (r_bsp) Rev.7.30 以上

## 2.3 サポートされているツールチェーン

このドライバは下記ツールチェーンで動作確認を行っています。詳細は、「6.1 動作確認環境詳細」を参照ください。

## 2.4 使用する割り込みベクタ

R_RI3C_Enable 関数を実行したとき、In-band 割り込みが有効になります。

表 2.1 に RI3C 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 |
|---|---|
| RX26T | RESPI 割り込み（ベクタ番号: 40）<br>CMDI 割り込み（ベクタ番号: 41）<br>IBII 割り込み（ベクタ番号: 42）<br>RCVI 割り込み（ベクタ番号: 43）<br>RXI 割り込み（ベクタ番号: 118）<br>TXI 割り込み（ベクタ番号: 119）<br>EEI 割り込み（ベクタ番号: 113） |

## 2.5 ヘッダファイル

すべての API 呼び出しと使用されるインタフェース定義は r_ri3c_rx_if.h と r_ri3c_rx_api.h に記載しています。

## 2.6 整数型

このプロジェクトは ANSI C99 を使用しています。これらの型は stdint.h で定義されています。

## 2.7 コンパイル時の設定

本モジュールのコンフィギュレーションオプションの設定は、r_ri3c_rx_config.h で行います。オプション名および設定値に関する説明を、下表に示します。

| Configuration options in *r_ri3c_rx_config.h* | |
|---|---|
| **RI3C_CFG_PARAM_CHECKING_ENABLE**<br>**※デフォルト値は BSP_CFG_PARAM_CHECKING_ENABLE** | APIパラメータチェック処理のコードを挿入するかどうかを指定します。<br>パラメータチェック処理を行うには1を設定します。0を設定するとパラメータチェック処理は除外されます。<br>このオプションの使用には注意が必要です。 |
| **RI3C_CFG_UNALIGNED_SUPPORT**<br>**※デフォルト値は 1** | 本デバイスでアラインされていないバッファをサポートするかどうかを選択します。<br>1：サポートする（システムデフォルト）<br>0：サポートしない |
| **RI3C_CFG_DEVICE_TYPE**<br>**※デフォルト値は** RI3C_DEVICE_TYPE_TARGET | I3C バス上で使用する I3C インスタンスの役割を指定します。<br>RI3C_DEVICE_TYPE_PRIMARY_CONTROLLER<br>RI3C_DEVICE_TYPE_TARGET（システムデフォルト） |
| **RI3C_CFG_CONTROLLER_SUPPORT**<br>**※デフォルト値は 1** | 本デバイスのコントローラモードを有効にするかどうかを選択します。<br>1：有効（システムデフォルト）<br>0：無効<br>コントローラモードのサポートだけが必要でターゲットモードのサポートが不要な場合は、ターゲットモードのサポートを無効にし、このオプションを省略してください（コードサイズが小さくなります）。 |
| **RI3C_CFG_TARGET_SUPPORT**<br>**※デフォルト値は 1** | 本デバイスのターゲットモードを有効にするかどうかを選択します。<br>1：有効（システムデフォルト）<br>0：無効<br>ターゲットモードのサポートだけが必要でコントローラモードのサポートが不要な場合は、コントローラモードのサポートを無効にし、このオプションを省略してください（コードサイズが小さくなります）。 |
| **RI3C_CFG_PCLKA_REF_VALUE**<br>**※デフォルト値は 48000000** | 周辺クロック A の基準値を指定します。<br>48 MHz（システムデフォルト）<br>64 MHz<br>FIT RI3C モジュールは 48 MHz と 64 MHz のどちらかで正常に動作します。 |
| **RI3C_CFG_STANDARD_OPEN_DRAIN_LOGIC_HIGH_PERIOD**<br>**※デフォルト値は 167** | 標準モードでオープンドレイン転送を行うときの SCL のロジックが High になる期間を指定します。 |
| **RI3C_CFG_STANDARD_OPEN_DRAIN_FREQUENCY**<br>**※デフォルト値は 1000000** | 標準モードでオープンドレイン転送を行うときの SCL の周波数を指定します。 |
| **RI3C_CFG_STANDARD_PUSH_PULL_LOGIC_HIGH_PERIOD**<br>**※デフォルト値は 167** | 標準モードでプッシュプル転送を行うときの SCL のロジックが High になる期間を指定します。 |
| **RI3C_CFG_STANDARD_PUSH_PULL_FREQUENCY**<br>**※デフォルト値は 3400000** | 標準モードでプッシュプル転送を行うときの SCL の周波数を指定します。 |

| Configuration options in *r_ri3c_rx_config.h* | |
|---|---|
| **RI3C_CFG_EXTENDED_OPEN_DRAIN_LOGIC_HIGH_PERIOD**<br>**※デフォルト値は 167** | 拡張モードでオープンドレイン転送を行うときの SCL のロジックが High になる期間を指定します。 |
| **RI3C_CFG_EXTENDED_OPEN_DRAIN_FREQUENCY**<br>**※デフォルト値は 1000000** | 拡張モードでオープンドレイン転送を行うときの SCL の周波数を指定します。 |
| **RI3C_CFG_EXTENDED_PUSH_PULL_LOGIC_HIGH_PERIOD**<br>**※デフォルト値は 167** | 拡張モードでプッシュプル転送を行うときの SCL のロジックが High になる期間を指定します。 |
| **RI3C_CFG_EXTENDED_PUSH_PULL_FREQUENCY**<br>**※デフォルト値は 3400000** | 拡張モードでプッシュプル転送を行うときの SCL の周波数を指定します。 |
| **RI3C_CFG_OPEN_DRAIN_RISING_TIME**<br>**※デフォルト値は 0** | オープンドレインの立ち上がり時間を ns（ナノ秒）単位で指定します。<br>0 以上の値を指定します。 |
| **RI3C_CFG_OPEN_DRAIN_FALLING_TIME**<br>**※デフォルト値は 0** | オープンドレインの立ち下がり時間を ns（ナノ秒）単位で指定します。<br>0 以上の値を指定します。 |
| **RI3C_CFG_PUSH_PULL_RISING_TIME**<br>**※デフォルト値は 0** | プッシュプルの立ち上がり時間を ns（ナノ秒）単位で指定します。<br>0 以上の値を指定します。 |
| **RI3C_CFG_PUSH_PULL_FALLING_TIME**<br>**※デフォルト値は 0** | プッシュプルの立ち下がり時間を ns（ナノ秒）単位で指定します。<br>0 以上の値を指定します。 |
| **RI3C_CFG_ADDRESS_ASSIGNMENT_PHASE**<br>**※デフォルト値は 0** | ENTDAA のアドレス割り当てフェーズにおいてクロックストールを有効にするかどうかを指定します。<br>0：無効（システムデフォルト）<br>1：有効 |
| **RI3C_CFG_TRANSITION_PHASE**<br>**※デフォルト値は 0** | リード転送の遷移ビットフェーズにおいてクロックストールを有効にするかどうかを指定します。<br>0：無効（システムデフォルト）<br>1：有効 |
| **RI3C_CFG_PARITY_PHASE**<br>**※デフォルト値は 0** | ライト転送のパリティビットフェーズにおいてクロックストールを有効にするかどうかを指定します。<br>0：無効（システムデフォルト）<br>1：有効 |
| **RI3C_CFG_ACK_PHASE**<br>**※デフォルト値は 0** | 転送の ACK フェーズにおいてクロックストールを有効にするかどうかを指定します。<br>0：無効（システムデフォルト）<br>1：有効 |
| **RI3C_CFG_CLOCK_STALLING_TIME**<br>**※デフォルト値は 0** | アドレス割り当てフェーズ、遷移フェーズ、パリティフェーズ、および ACK フェーズにおいてクロックをストールする時間を指定します。<br>整数を指定します。0 以上かつ PCLKA 未満の値を指定してください。 |
| **RI3C_CFG_CONTROLLER_ACK_HOTJOIN_REQ**<br>**※デフォルト値は 0** | このオプションを有効にした場合、RI3C インスタンスが Hot-Join 要求に肯定応答しアプリケーションに通知します。<br>0：無効（システムデフォルト）<br>1：有効 |

| Configuration options in *r_ri3c_rx_config.h* | |
|---|---|
| **RI3C_CFG_CONTROLLER_NOTIFY_REJECTED_HOTJOIN_REQ**<br>**※デフォルト値は 0** | このオプションを有効にした場合、IBI の Hot-Join 要求が拒否されたときにアプリケーションはコールバックを受けます。<br>0：無効（システムデフォルト）<br>1：有効 |
| **RI3C_CFG_CONTROLLER_NOTIFY_REJECTED_CONTROLLERROLE_REQ**<br>**※デフォルト値は 0** | このオプションを有効にした場合、IBI のコントローラロール要求が拒否されたときにアプリケーションはコールバックを受けます。<br>0：無効（システムデフォルト）<br>1：有効 |
| **RI3C_CFG_CONTROLLER_NOTIFY_REJECTED_INTERRUPT_REQ**<br>**※デフォルト値は 0** | このオプションを有効にした場合、IBI の割り込み要求が拒否されたときにアプリケーションはコールバックを受けます。<br>0：無効（システムデフォルト）<br>1：有効 |
| **RI3C_CFG_TARGET_IBI_REQ**<br>**※デフォルト値は 0** | ターゲットによる IBI 要求の発行可否を指定します。<br>0：否（システムデフォルト）<br>1：可 |
| **RI3C_CFG_TARGET_HOTJOIN_REQ**<br>**※デフォルト値は 0** | ターゲットによる Hot-Join 要求の発行可否を指定します。<br>0：否（システムデフォルト）<br>1：可 |
| **RI3C_CFG_TARGET_CONTROLLERROLE_REQ**<br>**※デフォルト値は 0** | ターゲットによるコントローラロール要求の発行可否を指定します。<br>0：否（システムデフォルト）<br>1：可 |
| **RI3C_CFG_TARGET_INCLUDE_MAX_READ_TURNAROUND_TIME**<br>**※デフォルト値は 0** | 最大リードターンアラウンドタイムを送信するかどうかを指定します。<br>0：しない（システムデフォルト）<br>1：する |
| **RI3C_CFG_TARGET_ENTER_ACTIVITY_STATE**<br>**※デフォルト値は RI3C_ACTIVITY_STATE_ENTAS0** | ターゲットの開始アクティビティステートを設定します。<br>RI3C_ACTIVITY_STATE_ENTAS0：1 ns（レイテンシフリー動作（デフォルト））<br>RI3C_ACTIVITY_STATE_ENTAS1：100 ns<br>RI3C_ACTIVITY_STATE_ENTAS2：2 μs<br>RI3C_ACTIVITY_STATE_ENTAS3：50 μs（最低アクティビティ動作） |
| **RI3C_CFG_TARGET_MAX_WRITE_LENGTH**<br>**※デフォルト値は 65535** | ターゲットモードにおける最大ライト長を設定します。<br>8～65535 の範囲の値を指定します。 |
| **RI3C_CFG_TARGET_MAX_READ_LENGTH**<br>**※デフォルト値は 65535** | ターゲットモードにおける最大リード長を設定します。<br>8～65535 の範囲の値を指定します。 |
| **RI3C_CFG_TARGET_MAX_IBI_PAYLOAD_LENGTH**<br>**※デフォルト値は 0** | 最大 IBI ペイロードサイズを設定します。0 を設定すると無制限となります。<br>0～255 の範囲の値を指定します。 |

| Configuration options in *r_ri3c_rx_config.h* | |
|---|---|
| **RI3C_CFG_TARGET_WRITE_DATA_RATE**<br>**※デフォルト値は RI3C_DATA_RATE_SETTING_2MHZ** | ターゲットモードにおける最大ライトデータレートを設定します。<br>RI3C_DATA_RATE_SETTING_FSCL_MAX：FSCL_MAX<br>RI3C_DATA_RATE_SETTING_8MHZ：8 MHz<br>RI3C_DATA_RATE_SETTING_6MHZ：6 MHz<br>RI3C_DATA_RATE_SETTING_4MHZ：4 MHz<br>RI3C_DATA_RATE_SETTING_2MHZ：2 MHz（デフォルト） |
| **RI3C_CFG_TARGET_READ_DATA_RATE**<br>**※デフォルト値は RI3C_DATA_RATE_SETTING_2MHZ** | ターゲットモードにおける最大リードデータレートを設定します。<br>RI3C_DATA_RATE_SETTING_FSCL_MAX：FSCL_MAX<br>RI3C_DATA_RATE_SETTING_8MHZ：8 MHz<br>RI3C_DATA_RATE_SETTING_6MHZ：6 MHz<br>RI3C_DATA_RATE_SETTING_4MHZ：4 MHz<br>RI3C_DATA_RATE_SETTING_2MHZ：2 MHz（デフォルト） |
| **RI3C_CFG_TARGET_CLK_TURNAROUND_RATE**<br>**※デフォルト値は RI3C_CLOCK_DATA_TURNAROUND_8NS** | ターゲットモードにおける最大クロックターンアラウンドレートを設定します。<br>RI3C_CLOCK_DATA_TURNAROUND_8NS：8 ns（デフォルト）<br>RI3C_CLOCK_DATA_TURNAROUND_9NS：9 ns<br>RI3C_CLOCK_DATA_TURNAROUND_10NS：10 ns<br>RI3C_CLOCK_DATA_TURNAROUND_11NS：11 ns<br>RI3C_CLOCK_DATA_TURNAROUND_12NS：12 ns<br>RI3C_CLOCK_DATA_TURNAROUND_EXTENDED：12 ns 超 |
| **RI3C_CFG_TARGET_INCLUDE_MAX_READ_TURNAROUND_TIME**<br>**※デフォルト値は 0** | ターゲットモードにおいて最大リードターンアラウンドタイムを送信するかどうかを設定します。<br>0：送信しない（システムデフォルト）<br>1：送信する |
| **RI3C_CFG_TARGET_MAX_READ_TURNAROUND_TIME**<br>**※デフォルト値は 0** | ターゲットモードにおける最大リードターンアラウンドタイムを設定します。<br>0～255 の範囲の値を指定します。 |
| **RI3C_CFG_TARGET_FREQUENCY_BYTE**<br>**※デフォルト値は 0** | ターゲットモードにおける内部発振周波数を 0.5 MHz 刻みで設定します。<br>0～255 の範囲の値を指定します。 |
| **RI3C_CFG_TARGET_INACCURACY_BYTE**<br>**※デフォルト値は 0** | ターゲットモードにおける内部発振器の Inaccuracy Byte を 0.5 MHz 刻みで設定します。<br>0～255 の範囲の値を指定します。 |
| **RI3C_CFG_BUS_FREE_DETECT_TIME**<br>**※デフォルト値は 38.4** | STOP から START までの最短期間を指定します。<br>38.4（ns） 以上の値を指定します。 |
| **RI3C_CFG_BUS_AVAILABLE_CONDITION_DETECT_TIME**<br>**※デフォルト値は 1** | ターゲットが IBI 要求を発行できるとき、バスフリー条件が満足した後に生じる最短期間を指定します。<br>1（µs） 以上の値を指定します。 |

| Configuration options in *r_ri3c_rx_config.h* | |
|---|---|
| **RI3C_CFG_BUS_IDLE_CONDITION_DETECT_TIME**<br>**※デフォルト値は 1000** | ターゲットが Hot-Join 要求を発行できるとき、バス使用可能条件が満足した後に生じる最短期間を指定します。<br>1000（ms） 以上の値を指定します。 |
| **RI3C_CFG_TIMEOUT_DETECTION**<br>**※デフォルト値は 0** | このオプションを有効にした場合、SCL のロジックが High レベルまたは Low レベルでスタック状態になり、RI3C ソースクロックが 65,535 サイクルを超えたときに、アプリケーションはコールバックを受けます。<br>0：無効（システムデフォルト）<br>1：有効 |
| **RI3C_CFG_INTERRUPT_PRIORITY_LEVEL**<br>**※デフォルト値は 2** | RI3C モジュールの割り込み優先レベルを設定します。<br>0～15 の範囲の値を指定します。 |

**表 2.2 RI3C モジュールのコンフィギュレーションオプション。**

## 2.8　コードサイズ

本モジュールのコードサイズを下表に示します。

ROM (コードおよび定数) と RAM (グローバルデータ) のサイズは、ビルド時の「2.7 コンパイル時の設定」のコンフィギュレーションオプションによって決まります。掲載した値は、「2.3 サポートされているツールチェーン」の C コンパイラでコンパイルオプションがデフォルト時の参考値です。コンパイルオプションのデフォルトは最適化レベル：2、最適化のタイプ：サイズ優先、データ・エンディアン：リトルエンディアンです。コードサイズは C コンパイラのバージョンやコンパイルオプションにより異なります。

下表の値は下記条件で確認しています。

モジュールリビジョン: r_ri3c_rx rev1.00

コンパイラバージョン: Renesas Electronics C/C++ Compiler Package for RX Family V3.05.00
(統合開発環境のデフォルト設定に”-lang = c99”オプションを追加)
GCC for Renesas RX 8.03.00.202204
(統合開発環境のデフォルト設定に” -std=gnu99”オプションを追加)
IAR C/C++ Compiler for Renesas RX version 4.20.03
(統合開発環境のデフォルト設定)

コンフィグレーションオプション: デフォルト設定

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、ロック有効 | パラメータチェックなし、ロック有効 | パラメータチェック処理あり、ロック必須 | パラメータチェックなし、ロック有効 | パラメータチェック処理あり、ロック必須 | パラメータチェックなし、ロック有効 |
| RX26T | ROM | コントローラ | 6569 バイト | 5788 バイト | 7508 バイト | 6804 バイト | 1401 バイト | 1401 バイト |
| | | ターゲット | 5931 バイト | 5474 バイト | 6876 バイト | 6428 バイト | 1401 バイト | 1401 バイト |
| | RAM | コントローラ | 292 バイト | | 384 バイト | | 96 バイト | |
| | | ターゲット | 292 バイト | | 384 バイト | | 96 バイト | |
| | 最大使用スタックサズ *1 | | 268 バイト | 300 バイト | - | | 300 バイト | 260 バイト |

注 1. 割り込み関数の最大使用スタックサイズを含みます。

## 2.9 引数

API 関数の引数である構造体を示します。この構造体に r_ri3c_rx_if.h と r_ri3c_rx_api.h で記載されています。

```
typedef struct st_ri3c_instance_ctrl
{
    uint32_t open;

#if((__CCRX__) || (__GNUC__))
    volatile struct RI3C0_Type R_BSP_EVENACCESS * p_reg;
#elif(__ICCRX__)
    struct RI3C0_Type R_BSP_VOLATILE_SFR * p_reg;
#endif

    volatile uint32_t              internal_state;
    uint8_t                        current_command_code;
    uint32_t                       device_index;
    ri3c_bitrate_mode_t            device_bitrate_mode;
    ri3c_target_info_t             current_target_info;
    uint32_t                       next_word;
    uint32_t                       ibi_next_word;
    ri3c_write_buffer_descriptor_t write_buffer_descriptor;
    ri3c_read_buffer_descriptor_t  read_buffer_descriptor;
    ri3c_read_buffer_descriptor_t  ibi_buffer_descriptor;
    volatile uint32_t              read_transfer_count_final;
    volatile uint32_t              ibi_transfer_count_final;
    ri3c_cfg_t const             * p_cfg;
} ri3c_instance_ctrl_t
```

```
typedef struct s_ri3c_extended_cfg
{
    ri3c_bitrate_settings_t             bitrate_settings;
    ri3c_ibi_control_t                  ibi_control
    uint32_t                            bus_free_detection_time;
    uint32_t                            bus_available_detection_time;
    uint32_t                            bus_idle_detection_time;
    bool                                timeout_detection_enable;
    ri3c_target_command_response_info_t target_command_response_info;
    uint8_t                             ipl;
    uint8_t                             eei_ipl;
} ri3c_extended_cfg_t;
```

```
typedef struct s_target_command_response_info
{
    bool                          inband_interrupt_enable;
    bool                          controllerrole_request_enable;
    bool                          hotjoin_request_enable;
    ri3c_activity_state_t         activity_state;
    uint16_t                      write_length;
    uint16_t                      read_length;
    uint8_t                       ibi_payload_length;
    ri3c_data_rate_setting_t      write_data_rate;
    ri3c_data_rate_setting_t      read_data_rate;
    ri3c_clock_data_turnaround_t clock_data_turnaround;
    bool                          read_turnaround_time_enable;
    uint32_t                      read_turnaround_time;
    uint8_t                       oscillator_frequency;
    uint8_t                       oscillator_inaccuracy;
} ri3c_target_command_response_info_t;
```

```
typedef struct s_ri3c_clock_stalling
{
    uint32_t assigned_address_phase_enable : 1;
    uint32_t transition_phase_enable       : 1;
    uint32_t parity_phase_enable           : 1;
    uint32_t ack_phase_enable              : 1;
    uint16_t clock_stalling_time;
} ri3c_clock_stalling_t;
```

```
typedef struct s_ri3c_bitrate_settings
{
    uint32_t              icsbr; ///< 標準ビットレートの設定
    uint32_t              icebr; ///< 拡張ビットレートの設定
    ri3c_clock_stalling_t clock_stalling;
} ri3c_bitrate_settings_t;
```

```
typedef struct s_ri3c_ibi_control
{
    uint32_t hot_join_acknowledge                    : 1;
    uint32_t notify_rejected_hot_join_requests       : 1;
    uint32_t notify_rejected_controllerrole_requests : 1;
    uint32_t notify_rejected_interrupt_requests      : 1;
} ri3c_ibi_control_t;
```

```
typedef struct s_ri3c_read_buffer_descriptor
{
    uint8_t * p_buffer;
    uint32_t  count;
    uint32_t  buffer_size;
    bool      read_request_issued;
} ri3c_read_buffer_descriptor_t;
```

```
typedef struct s_ri3c_write_buffer_descriptor
{
    uint8_t * p_buffer;
    uint32_t  count;
    uint32_t  buffer_size;
} ri3c_write_buffer_descriptor_t;
```

```
typedef enum e_ri3c_common_command_code
{
    /* ブロードキャスト共通コマンドコード */
    I3C_CCC_BROADCAST_ENEC     = (0x00), ///< ターゲット開始イベントを許可
    I3C_CCC_BROADCAST_DISEC    = (0x01), ///< ターゲット開始イベントを禁止
    I3C_CCC_BROADCAST_ENTAS0   = (0x02), ///< アクティビティステート 0 に入る
    I3C_CCC_BROADCAST_ENTAS1   = (0x03), ///< アクティビティステート 1 に入る
    I3C_CCC_BROADCAST_ENTAS2   = (0x04), ///< アクティビティステート 2 に入る
    I3C_CCC_BROADCAST_ENTAS3   = (0x05), ///< アクティビティステート 3 に入る
    I3C_CCC_BROADCAST_RSTDAA   = (0x06), ///< ダイナミックアドレス割り当てをリセット
    I3C_CCC_BROADCAST_ENTDAA   = (0x07), ///< ダイナミックアドレス割り当てに入る
    I3C_CCC_BROADCAST_DEFSVLS  = (0x08), ///< ターゲットの一覧を定義
    I3C_CCC_BROADCAST_SETMWL   = (0x09), ///< 最大ライト長を設定
    I3C_CCC_BROADCAST_SETMRL   = (0x0A), ///< 最大リード長を設定
    I3C_CCC_BROADCAST_ENTTM    = (0x0B), ///< テストモードに入る
    I3C_CCC_BROADCAST_ENTHDR0  = (0x20), ///< HDR モード 0 に入る
    I3C_CCC_BROADCAST_ENTHDR1  = (0x21), ///< HDR モード 1 に入る
    I3C_CCC_BROADCAST_ENTHDR2  = (0x22), ///< HDR モード 2 に入る
    I3C_CCC_BROADCAST_ENTHDR3  = (0x23), ///< HDR モード 3 に入る
    I3C_CCC_BROADCAST_ENTHDR4  = (0x24), ///< HDR モード 4 に入る
    I3C_CCC_BROADCAST_ENTHDR5  = (0x25), ///< HDR モード 5 に入る
    I3C_CCC_BROADCAST_ENTHDR6  = (0x26), ///< HDR モード 6 に入る
    I3C_CCC_BROADCAST_ENTHDR7  = (0x27), ///< HDR モード 7 に入る
    I3C_CCC_BROADCAST_SETXTIME = (0x28), ///< 交換タイミング情報を設定
    I3C_CCC_BROADCAST_SETAASA  = (0x29), ///< すべてのアドレスをスタティックアドレスに設定

    /* ダイレクト共通コマンドコード */
    I3C_CCC_DIRECT_ENEC      = (0x80),   ///< ターゲット開始イベントを許可
    I3C_CCC_DIRECT_DISEC     = (0x81),   ///< ターゲット開始イベントを禁止
    I3C_CCC_DIRECT_ENTAS0    = (0x82),   ///< アクティビティステート 0 に入る
    I3C_CCC_DIRECT_ENTAS1    = (0x83),   ///< アクティビティステート 1 に入る
    I3C_CCC_DIRECT_ENTAS2    = (0x84),   ///< アクティビティステート 2 に入る
    I3C_CCC_DIRECT_ENTAS3    = (0x85),   ///< アクティビティステート 3 に入る
    I3C_CCC_DIRECT_RSTDAA    = (0x86),   ///< ダイナミックアドレス割り当てをリセット
    I3C_CCC_DIRECT_SETDASA   = (0x87),   ///< スタティックアドレスからダイナミックアドレスを設定
    I3C_CCC_DIRECT_SETNEWDA  = (0x88),   ///< 新しいダイナミックアドレスを設定
    I3C_CCC_DIRECT_SETMWL    = (0x89),   ///< 最大ライト長を設定
    I3C_CCC_DIRECT_SETMRL    = (0x8A),   ///< 最大リード長を設定
    I3C_CCC_DIRECT_GETMWL    = (0x8B),   ///< 最大ライト長を取得
    I3C_CCC_DIRECT_GETMRL    = (0x8C),   ///< 最大リード長を取得
    I3C_CCC_DIRECT_GETPID    = (0x8D),   ///< 暫定 ID を取得
    I3C_CCC_DIRECT_GETBCR    = (0x8E),   ///< バス特性レジスタを取得
    I3C_CCC_DIRECT_GETDCR    = (0x8F),   ///< デバイス特性レジスタを取得
    I3C_CCC_DIRECT_GETSTATUS = (0x90),   ///< デバイスステータスを取得
    I3C_CCC_DIRECT_GETACCMST = (0x91),   ///< コントローラロールを取得
    I3C_CCC_DIRECT_GETMXDS   = (0x94),   ///< 最大データスピードを取得
    I3C_CCC_DIRECT_SETXTIME  = (0x98),   ///< 交換タイミング情報を設定
    I3C_CCC_DIRECT_GETXTIME  = (0x99),   ///< 交換タイミング情報を取得
} ri3c_common_command_code_t;
```

```
typedef struct s_ri3c_target_device_cfg
{
    uint8_t            static_address;
    uint8_t            dynamic_address;
    ri3c_target_info_t target_info;
} ri3c_device_cfg_t;
```

```
typedef enum e_ri3c_event
{
    RI3C_EVENT_ENTDAA_ADDRESS_PHASE,
    RI3C_EVENT_IBI_READ_COMPLETE,
    RI3C_EVENT_IBI_READ_BUFFER_FULL,
    RI3C_EVENT_READ_BUFFER_FULL,
    RI3C_EVENT_IBI_WRITE_COMPLETE,
    RI3C_EVENT_HDR_EXIT_PATTERN_DETECTED,
    RI3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE,
    RI3C_EVENT_COMMAND_COMPLETE,
    RI3C_EVENT_WRITE_COMPLETE,
    RI3C_EVENT_READ_COMPLETE,
    RI3C_EVENT_TIMEOUT_DETECTED,
    RI3C_EVENT_INTERNAL_ERROR,
} ri3c_event_t;
```

```
typedef enum e_ri3c_type
{
    RI3C_DEVICE_TYPE_PRIMARY_CONTROLLER,
    RI3C_DEVICE_TYPE_TARGET,
} ri3c_device_type_t;
```

```
typedef enum e_ri3c_device_protocol
{
    RI3C_DEVICE_PROTOCOL_I2C,
    RI3C_DEVICE_PROTOCOL_I3C,
} ri3c_device_protocol_t;
```

```
typedef enum e_ri3c_address_assignment_mode
{
    RI3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA  = I3C_CCC_BROADCAST_ENTDAA,
    RI3C_ADDRESS_ASSIGNMENT_MODE_SETDASA = I3C_CCC_DIRECT_SETDASA,
} ri3c_address_assignment_mode_t;
```

```
typedef enum e_ri3c_ibi_type
{
    RI3C_IBI_TYPE_INTERRUPT,
    RI3C_IBI_TYPE_HOT_JOIN,
    RI3C_IBI_TYPE_CONTROLLERROLE_REQUEST
} ri3c_ibi_type_t;
```

```
typedef struct s_ri3c_device_status
{
    uint8_t pending_interrupt;
    uint8_t vendor_status;
} ri3c_device_status_t;
```

```
typedef struct s_ri3c_device_table_cfg
{
    uint8_t static_address;             ///< I3C スタティックアドレス / 本デバイスの
I2C アドレス

    /** 本デバイスのダイナミックアドレス（ダイナミックアドレス割り当て時に割り当てられる）。
*/
    uint8_t dynamic_address;

    ri3c_device_protocol_t device_protocol;     ///< 本デバイスとの通信に用いられるプ
ロトコル (I3C / I2C Legacy)
    bool ibi_accept;                            ///< 本デバイスからの IBI 要求の受け付
け／拒否
    bool controllerrole_request_accept;         ///< 本デバイスからのコントローラロー
ル要求の受け付け

    /**
     * 本デバイスからの IBI 要求にはデータペイロードがある。
     *
     * 注意：ENTDAA で本デバイスが設定される際に BCR の値に基づいて ibi_payload が自動的に
     *       更新される。
     */
    bool ibi_payload;
} ri3c_device_table_cfg_t;
```

```
typedef struct s_ri3c_target_info
{
    uint8_t pid[6];
    union
    {
        uint8_t bcr;
        struct
        {
            uint8_t max_data_speed_limitation : 1;
            uint8_t ibi_request_capable       : 1;
            uint8_t ibi_payload               : 1;
            uint8_t offline_capable           : 1;
            uint8_t                           : 2;
            uint8_t device_role               : 2;
        } bcr_b;
    };

   uint8_t dcr;
} ri3c_target_info_t;
```

```
typedef struct s_ri3c_command_descriptor
{
    uint8_t   command_code;
    uint8_t * p_buffer;
    uint32_t  length;
    bool      restart;
    bool      rnw;
} ri3c_command_descriptor_t;
```

```
typedef struct s_ri3c_callback_args
{
    ri3c_event_t               event;
    uint32_t                   event_status;
    uint32_t                   transfer_size;
    ri3c_target_info_t const * p_target_info;
    uint8_t                    dynamic_address;
    ri3c_ibi_type_t            ibi_type;
    uint8_t                    ibi_address;
    uint8_t                    command_code;
    void const               * p_context;
} ri3c_callback_args_t;
```

```
typedef struct st_ri3c_cfg
{
    uint32_t            channel;
    ri3c_device_type_t  device_type;
    void (* p_callback) (ri3c_callback_args_t const * const p_args);
    void const        * p_context;
    void const        * p_extend;
} ri3c_cfg_t;
```

```
typedef void ri3c_ctrl_t;

typedef struct st_ri3c_instance
{
    ri3c_ctrl_t * p_ctrl;
    ri3c_cfg_t  * p_cfg;
    ri3c_api_t  * p_api;
} ri3c_instance_t;
```

```
typedef struct st_ri3c_api
{
    fsp_err_t (* open)
              (ri3c_ctrl_t * const p_ctrl, ri3c_cfg_t const * const p_cfg);

    fsp_err_t (* enable)
              (ri3c_ctrl_t * const p_ctrl);

    fsp_err_t (* deviceCfgSet)
              (ri3c_ctrl_t * const p_ctrl,
               ri3c_device_cfg_t const * const p_device_cfg);

    fsp_err_t (* controllerDeviceTableSet)
              (ri3c_ctrl_t * const p_ctrl,
               uint32_t device_index,
               ri3c_device_table_cfg_t const * const p_device_table_cfg);

    fsp_err_t (* deviceSelect)
              (ri3c_ctrl_t * const p_ctrl,
               uint32_t device_index,
               uint32_t bitrate_mode);

    fsp_err_t (* dynamicAddressAssignmentStart)
              (ri3c_ctrl_t * const p_ctrl,
               ri3c_address_assignment_mode_t address_assignment_mode,
               uint32_t starting_device_index,
               uint32_t device_count);

    fsp_err_t (* targetStatusSet)
              (ri3c_ctrl_t * const p_ctrl,
               ri3c_device_status_t device_status);

    fsp_err_t (* commandSend)
              (ri3c_ctrl_t * const p_ctrl,
               ri3c_command_descriptor_t * p_command_descriptor);

    fsp_err_t (* write)
              (ri3c_ctrl_t * const p_ctrl,
               uint8_t const * const p_data,
               uint32_t length,
               bool restart);

    /**
     * 【コントローラモードの場合】リード転送を開始する。転送完了時、ストップコンディション
     *    またはリスタートコンディションを送信する。
     * 【ターゲットモードの場合】転送時に読み出されるデータを格納するリードバッファを
     *    設定する。
     *    バッファが満杯になるとアプリケーションはコールバックを受け、
     *    新しいリードバッファを要求する。
     *    バッファが用意されていない場合、ドライバは読み出したデータの残りバイトを廃棄する。
     *
     * @param[in]   p_ctrl     制御ブロックが本インスタンスを設定する。
     * @param[in]   p_data     転送時に読み出されるバイトデータを格納するバッファへの
     *                         ポインタ
     * @param[in]   length     転送バイト数
     * @param[in]   restart    真ならば、転送完了後にリスタートコンディションを発行する
     *                         （コントローラモード時のみ）。
     */
```

```
    fsp_err_t (* read)(ri3c_ctrl_t * const p_ctrl, uint8_t * const p_data,
uint32_t length, bool restart);

    /**
     * IBI 書き込み動作を開始する。
     *
     * 注意：この関数はコントローラモードでは用いない。
     *
     * @param[in]   p_ctrl     制御ブロックが本インスタンスを設定する。
     * @param[in]   p_data     転送時に読み出されるバイトデータを格納するバッファへの
     *                         ポインタ
     * @param[in]   length     転送バイト数
     */
    fsp_err_t (* ibiWrite)(ri3c_ctrl_t * const p_ctrl,
                           ri3c_ibi_type_t ibi_type,
                           uint8_t const * const p_data,
                           uint32_t length);

    /**
     * 受信した IBI データを格納するリードバッファの設定（この関数はターゲットモードでは用いな
い）
     *
     * @param[in]   p_ctrl     制御ブロックが本インスタンスを設定する。
     * @param[in]   p_data     転送時に読み出されるバイトデータを格納するバッファへの
     *                         ポインタ
     * @param[in]   length     転送バイト数
     */
    fsp_err_t (* ibiRead)(ri3c_ctrl_t * const p_ctrl, uint8_t * const p_data,
uint32_t length);

    /** ドライバの再設定を許可し消費電力を低減する可能性あり。
     *
     * @param[in]   p_ctrl     制御ブロックが本インスタンスを設定する。
     */
    fsp_err_t (* close)(ri3c_ctrl_t * const p_ctrl);
} ri3c_api_t;
```

## 2.10 戻り値

API 関数の戻り値を示します。この列挙型は fsp_common_api.h で記載されています。

```
/** 共通エラーコード */
typedef enum e_fsp_err
{
    FSP_SUCCESS                    = 0,

    FSP_ERR_ASSERTION              = 1,
    FSP_ERR_INVALID_POINTER        = 2,
    FSP_ERR_INVALID_ARGUMENT       = 3,
    FSP_ERR_INVALID_CHANNEL        = 4,
    FSP_ERR_INVALID_MODE           = 5,
    FSP_ERR_UNSUPPORTED            = 6,
    FSP_ERR_NOT_OPEN               = 7,
    FSP_ERR_IN_USE                 = 8,
    FSP_ERR_OUT_OF_MEMORY          = 9,
    FSP_ERR_HW_LOCKED              = 10,
    FSP_ERR_IRQ_BSP_DISABLED       = 11,
    FSP_ERR_OVERFLOW               = 12,
    FSP_ERR_UNDERFLOW              = 13,
    FSP_ERR_ALREADY_OPEN           = 14,
    FSP_ERR_APPROXIMATION          = 15,
    FSP_ERR_CLAMPED                = 16,
    FSP_ERR_INVALID_RATE           = 17,
    FSP_ERR_ABORTED                = 18,
    FSP_ERR_NOT_ENABLED            = 19,
    FSP_ERR_TIMEOUT                = 20,
    FSP_ERR_INVALID_BLOCKS         = 21,
    FSP_ERR_INVALID_ADDRESS        = 22,
    FSP_ERR_INVALID_SIZE           = 23,
    FSP_ERR_WRITE_FAILED           = 24,
    FSP_ERR_ERASE_FAILED           = 25,
    FSP_ERR_INVALID_CALL           = 26,
    FSP_ERR_INVALID_HW_CONDITION   = 27,
    FSP_ERR_INVALID_FACTORY_FLASH  = 28,
    FSP_ERR_INVALID_STATE          = 30,
    FSP_ERR_NOT_ERASED             = 31,
    FSP_ERR_SECTOR_RELEASE_FAILED  = 32,
    FSP_ERR_NOT_INITIALIZED        = 33,
    FSP_ERR_NOT_FOUND              = 34,
    FSP_ERR_NO_CALLBACK_MEMORY     = 35,
    FSP_ERR_BUFFER_EMPTY           = 36,
    ...
} fsp_err_t;
```

## 2.11 コールバック関数

本モジュールでは、以下のいずれかのタイミングでユーザーが設定したコールバック関数が読み出されます。

(1) 送信データエンプティ

(2) 受信データフル

(3) IBI キューエンプティ／フル

(4) コマンドキューエンプティ

(5) レスポンスキューフル

(6) 受信ステータスキューフル

(7) 通信エラー／通信イベント

コールバック関数は、ユーザー関数のアドレスを ri3c_cfg_t 構造体の p_callback 引数に格納することで設定されます。コールバック関数のデフォルト値は関数ポインタです。これをユーザー関数に変更するには、「ri3c_callback_args_t const *const p_args」を引数に持つ任意の関数に g_ri3c0_cfg.p_callback を割り当ててください。

**Example**

```
/* 関数プロトタイプ */
void g_i3c0_user_callback(ri3c_callback_args_t const *const p_args);

void main(void)
{
    /* コールバック関数を RI3C FIT モジュールに割り当てる */
    g_ri3c0_cfg.p_callback = &g_i3c0_user_callback;
    R_RI3C_Open(&g_ri3c0_ctrl, &g_ri3c0_cfg);
}

void g_i3c0_user_callback(ri3c_callback_args_t const *const p_args)
{
    user_program();
}
```

## 2.12 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「RX スマート・コンフィグレータ ユーザーガイド: e² studio 編 (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「RX スマート・コンフィグレータ ユーザーガイド: CS+編 (R20AN0470)」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。

(5) IAREW 上で Smart Configurator を使用して FIT モジュールを追加する場合
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「RX スマート・コンフィグレータ ユーザーガイド: IAREW 編 (R20AN0535)」を参照してください。

## 2.13 for 文、while 文、do while 文について

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

以下に記述例を示します。

while 文の例：

```
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}
```

for 文の例：

```
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}
```

do while 文の例：

```
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /* WAIT_LOOP */
```

# 3. API 関数

## R_RI3C_Open()

RI3C インスタンスの設定を行います。

### Format

fsp_err_t R_RI3C_Open(
ri3c_ctrl_t *const p_api_ctrl
ri3c_cfg_t const *const p_cfg
)

### Parameters

*ri3c_ctrl_t *const p_api_ctrl*

RI3C 制御ブロックへのポインタ。この構造体のすべての要素は R_RI3C_Open()を呼び出すことで初期化されます。

*ri3c_cfg_t* const *const p_cfg*

RI3C コンフィグレーション構造体へのポインタです。この構造体の要素はすべてあらかじめ定義されています。ただし、R_RI3C_Open()を呼び出す前にユーザーが変更することは可能です。

この構造体の詳細については「2.9 引数」を参照してください。

### Return Values

*FSP_SUCCESS /* 処理に成功しました。 */*
*FSP_ERR_ASSERTION /* NULL ポインタがありました。 */*
*FSP_ERR_ALREADY_OPEN /* ドライバは既に開いています。 */*
*FSP_ERR_UNSUPPORTED /* 選択した機能は、現在の構成ではサポートされていません。 */*

### Properties

r_ri3c_rx_if.h にプロトタイプ宣言されています。

### Description

RI3C インスタンスの設定を行います。

- ri3c_instance_ctrl_t を初期化します。
- RI3C FIT モジュールレジスタ配置を初期化します。
- ビットレート設定の評価を行います。
- RI3C リセットビットを解放します。
- スタートコンディションを生成します。

### Example

```
fsp_err_t err;

err = R_RI3C_Open(&g_api_ctrl, &g_ri3c0_cfg);
```

### Special Notes

なし。

## R_RI3C_Enable()

RI3C デバイスを有効にします。

### Format
fsp_err_t R_RI3C_Enable(ri3c_ctrl_t *const p_api_ctrl)

### Parameters
*ri3c_ctrl_t *const p_api_ctrl*

RI3C 制御ブロックへのポインタ。

この構造体の詳細については「2.9 引数」を参照してください。

### Return Values
*FSP_SUCCESS /* 処理に成功しました。 */*
*FSP_ERR_ASSERTION /* NULL ポインタがありました。 */*
*FSP_ERR_NOT_OPEN /* 制御ブロックが開いていません。 */*
*FSP_ERR_INVALID_MODE /* インスタンスは既に有効化されている */*

### Properties
r_ri3c_rx_if.h にプロトタイプ宣言されています。

### Description
RI3C デバイスを有効にします。

• RI3C のステートフラグおよび各種ステータスフラグを設定します。
• ビットレート設定を初期化します。
• RI3C 割り込みを許可します。

### Example
```
fsp_err_t err;

err = R_RI3C_Enable(&g_api_ctrl);
```

### Special Notes
なし。

## R_RI3C_DeviceCfgSet()

本デバイスのコンフィグレーションを設定します。

### Format

```
fsp_err_t R_RI3C_DeviceCfgSet(
        ri3c_ctrl_t *const p_api_ctrl,
        ri3c_device_cfg_t const *const p_device_cfg
)
```

### Parameters

*p_api_ctrl*

RI3C 制御ブロックへのポインタ。

*p_device_cfg*

ドライバがターゲットモードのときに対象アドレスの設定を行うための構造体

この構造体の詳細については「2.9 引数」を参照してください。

### Return Values

*FSP_SUCCESS /* 処理に成功しました。 */*
*FSP_ERR_ASSERTION /* NULL ポインタがありました。 */*
*FSP_ERR_NOT_OPEN /* 制御ブロックが開いていません。 */*
*FSP_ERR_UNSUPPORTED /* コントローラモードのサポートが無効の場合、本デバイスは 2 次コントローラとして使用不可。 */*

### Properties

r_ri3c_rx_if.h にプロトタイプ宣言されています。

### Description

本デバイスのコンフィグレーションを設定します。
デバイス種別が「コントローラ」の場合：
• RI3C のダイナミックアドレスを設定します。
デバイス種別が「ターゲット」の場合：
• デバイスのスタティックアドレスを設定します。
• デバイスのダイナミックアドレスを設定します。
• BCR レジスタに基づいて IBI ペイロード設定を行います。
• ターゲットアドレステーブルレジスタへ設定を書き込みます。
• ターゲット特性テーブルレジスタへ BCR 設定と DCR 設定を書き込みます。
• ターゲット特性テーブル PID レジスタへ PID 設定を書き込みます。
• 対象アドレスを有効にします。

### Example

```
fsp_err_t err;
ri3c_device_cfg_t controller_device_cfg =
{
    /* デバイスメーカが定義したスタティック I3C/I2C Legacy アドレス */
    .static_address  = RI3C_CONTROLLER_DEVICE_STATIC_ADDRESS,
    /* コントローラが CCC ENTDAA を使用して本デバイスの設定を行う際にダイナミックアドレスが自動的に更新される */
    .dynamic_address = RI3C_CONTROLLER_DEVICE_DYNAMIC_ADDRESS
};

err = R_RI3C_DeviceCfgSet(
&g_api_ctrl,
&controller_device_cfg
);
```

### Special Notes

なし。

## R_RI3C_ControllerDeviceTableSet()

コントローラテーブルにエントリを設定します。

### Format

```
fsp_err_t R_RI3C_ControllerDeviceTableSet(
        ri3c_ctrl_t *const p_api_ctrl,
        uint32_t device_index,
        ri3c_device_table_cfg_t const *const p_device_table_cfg
)
```

### Parameters

*p_api_ctrl*

RI3C 制御ブロックへのポインタ。

*device_index*

デバイステーブルのインデックス

*p_device_table_cfg*

コントローラテーブル内エントリのテーブル設定へのポインタ

この構造体の詳細については「2.9 引数」を参照してください。

### Return Values

*FSP_SUCCESS /* 処理に成功しました。 */*
*FSP_ERR_ASSERTION /* NULL ポインタがありました。 */*
*FSP_ERR_NOT_OPEN /* 制御ブロックが開いていません。 */*
*FSP_ERR_UNSUPPORTED /* ターゲットモードのサポートが無効の場合、コントローラロール要求は拒否されなければならない。 */*

### Properties

r_ri3c_rx_if.h にプロトタイプ宣言されています。

### Description

コントローラテーブルにエントリを設定します。

• 本デバイスの IBI 設定を行います。
• デバイスのスタティックアドレスを設定します。
• 本デバイスによる転送に Legacy I2C プロトコルが使用されるようデバイス種別を設定します。

### Example

```
fsp_err_t err;

err = R_RI3C_ControllerDeviceTableSet(
&g_ri3c0_ctrl,
0,
&g_device_table_cfg
);
```

### Special Notes

なし。

## R_RI3C_TargetStatusSet()

GETSTATUS コマンドへの応答としてコントローラに返されるステータスを設定します。

### Format

```
fsp_err_t R_RI3C_TargetStatusSet(
        ri3c_ctrl_t *const p_api_ctrl,
        ri3c_device_status_t status
)
```

### Parameters

*p_api_ctrl*

RI3C 制御ブロックへのポインタ。

*status*

ターゲットの現在のステータス。

### Return Values

*FSP_SUCCESS /* 処理に成功しました。 */*
*FSP_ERR_ASSERTION /* NULL ポインタがありました。 */*
*FSP_ERR_NOT_OPEN /* 制御ブロックが開いていません。 */*
*FSP_ERR_INVALID_MODE /* インスタンスはターゲットモードではない。 */*
*FSP_ERR_UNSUPPORTED /* ターゲットモードのサポートは無効。 */*

### Properties

r_ri3c_rx_if.h にプロトタイプ宣言されています。

### Description

GETSTATUS コマンドへの応答としてコントローラに返されるステータスを設定します。

• ICDSR レジスタの保留割り込みフィールドとベンダ予約フィールドをクリアします。
• ICDSR レジスタに新しい保留割り込みフィールドとベンダ予約フィールドを書き込みます。

### Example

```
ri3c_device_status_t g_target_device_status =
{
     .pending_interrupt = 0,
     .vendor_status = 0,
};
fsp_err_t err;

err = R_RI3C_TargetStatusSet(&g_ri3c0_ctrl, g_target_device_status);
```

### Special Notes

なし。

## R_RI3C_DeviceSelect()

コントローラモードにおいて次の転送先デバイスを選択します。

### Format

```
fsp_err_t R_RI3C_DeviceSelect(
        ri3c_ctrl_t *const p_api_ctrl,
        uint32_t device_index,
        uint32_t bitrate_mode
)
```

### Parameters

*p_api_ctrl*

RI3C 制御ブロックへのポインタ。

*device_index*

コントローラテーブルのインデックス

*bitrate_mode*

選択したデバイスのビットレート設定

この構造体の詳細については「2.9 引数」を参照してください。

### Return Values

*FSP_SUCCESS /* 処理に成功しました。 */*
*FSP_ERR_ASSERTION /* NULL ポインタがありました。 */*
*FSP_ERR_NOT_OPEN /* 制御ブロックが開いていません。 */*
*FSP_ERR_INVALID_MODE /* ターゲットモードで禁止された動作が行われた。 */*
*FSP_ERR_UNSUPPORTED /* コントローラモードのサポートが無効化されている。 */*

### Properties

r_ri3c_rx_if.h にプロトタイプ宣言されています。

### Description

コントローラモードにおいて次の転送先デバイスを選択します。

### Example

```
fsp_err_t err;

ret = R_RI3C_DeviceSelect(
&g_ri3c0_ctrl,
0,
RI3C_BITRATE_MODE_RI3C_SDR4_ICEBR_X4
);
```

### Special Notes

なし。

## R_RI3C_DynamicAddressAssignmentStart()

ダイナミックアドレス割り当てプロセスを開始します。

### Format

```
fsp_err_t R_RI3C_DynamicAddressAssignmentStart(
        ri3c_ctrl_t *const p_api_ctrl,
        ri3c_address_assignment_mode_t address_assignment_mode,
        uint32_t starting_device_index,
        uint32_t device_count
)
```

### Parameters

*p_api_ctrl*

RI3C 制御ブロックへのポインタ。

*address_assignment_mode*

コントローラテーブルのインデックス

*starting_device_index*

選択したデバイスのビットレート設定

*device_count*

割り当てるデバイスの数 (I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA 指定時のみ使用)

### Return Values

*FSP_SUCCESS /* 処理に成功しました。 */*
*FSP_ERR_ASSERTION /* NULL ポインタがありました。 */*
*FSP_ERR_NOT_OPEN /* 制御ブロックが開いていません。 */*
*FSP_ERR_INVALID_MODE /* ターゲットモードで禁止された動作が行われた。 */*
*FSP_ERR_IN_USE /* ドライバがビジーのため動作を完了できなかった。 */*
*FSP_ERR_UNSUPPORTED /* コントローラモードのサポートが無効化されている。 */*

### Properties

r_ri3c_rx_if.h にプロトタイプ宣言されています。

### Description

ダイナミックアドレス割り当てプロセスを開始します。

### Example

```
fsp_err_t err;

err = R_RI3C_DynamicAddressAssignmentStart(
&g_ri3c0_ctrl,
RI3C_ADDRESS_ASSIGNMENT_MODE_SETDASA,
1,
0
);
```

### Special Notes

なし。

## R_RI3C_CommandSend()

バス上のターゲットにブロードキャストコマンドまたはダイレクトコマンドを送信します。

### Format

```
fsp_err_t R_RI3C_CommandSend(
        ri3c_ctrl_t *const p_api_ctrl,
        ri3c_command_descriptor_t *p_command_descriptor
)
```

### Parameters

*p_api_ctrl*

RI3C 制御ブロックへのポインタ。

p_command_descriptor

コマンドを実行するためのディスクリプタ。

### Return Values

*FSP_SUCCESS /* 処理に成功しました。 */*
*FSP_ERR_ASSERTION /* NULL ポインタがありました。 */*
*FSP_ERR_NOT_OPEN /* 制御ブロックが開いていません。 */*
*FSP_ERR_IN_USE /* ドライバがビジーのため動作を完了できなかった。 */*
*FSP_ERR_INVALID_MODE /* ターゲットモードで禁止された動作が行われた。 */*
*FSP_ERR_INVALID_ALIGNMENT /* バッファは 4 バイトアラインで配置しなければならない。リード動作ではデータ長は 4 バイトの倍数でなければならない。 */*
*FSP_ERR_UNSUPPORTED /* 関数の呼び出しに必要なコントローラモードのサポートが無効化されている／GETACCMST コマンドの送信に必要なターゲットモードのサポートが無効化されている。 */*

### Properties

r_ri3c_rx_if.h にプロトタイプ宣言されています。

### Description

コマンドディスクリプタの内容に基づいてバス上のターゲットにブロードキャストコマンドまたはダイレクトコマンドを送信します。コマンドディスクリプタの詳細については、アプリケーションノート「RX26T グループ　ユーザーズマニュアル　ハードウェア編」 (R01UH0979) を参照してください。

### Example

```
/* すべてのターゲットにダイナミックアドレス割り当てリセット要求を送信 */
fsp_err_t err;

ri3c_command_descriptor_t command_descriptor;
command_descriptor.command_code = I3C_CCC_BROADCAST_RSTDAA;
command_descriptor.p_buffer     = NULL;
command_descriptor.length       = 0;
command_descriptor.restart      = false;
command_descriptor.rnw          = false;
err = R_RI3C_CommandSend(
    &g_ri3c0_ctrl,
    &command_descriptor
);
```

### Special Notes

command_code フィールドの詳細については、「2.9 引数」の「ri3c_common_command_code_t」を参照してください。

コマンドバッファは 4 バイトアラインで配置しなければなりません。

## R_RI3C_Write()

転送に使用するライトバッファを設定します。コントローラモードでは、転送を開始します。転送が完了するとストップコンディションまたはリスタートコンディションを送信します。

### Format

```
fsp_err_t R_RI3C_Write(
        ri3c_ctrl_t *const p_api_ctrl,
        uint8_t const *const p_data,
        uint32_t length,
        bool restart
)
```

### Parameters

*p_api_ctrl*

RI3C 制御ブロックへのポインタ。

*p_data*

書き込み先バッファへのポインタ。

*length*

転送バイト数。

*restart*

真の場合、転送完了後にリスタートコンディションを発行します（コントローラモード時のみ）。

### Return Values

*FSP_SUCCESS /* 処理に成功しました。 */*
*FSP_ERR_ASSERTION /* NULL ポインタがありました。 */*
*FSP_ERR_NOT_OPEN /* 制御ブロックが開いていません。 */*
*FSP_ERR_IN_USE /* ドライバがビジーのため動作を完了できなかった。 */*
*FSP_ERR_INVALID_MODE /* インスタンスは既に有効化されている。 */*
*FSP_ERR_INVALID_ALIGNMENT /* バッファは 4 バイトアラインで配置しなければならない。 */*

### Properties

r_ri3c_rx_if.h にプロトタイプ宣言されています。

### Description

転送に使用するライトバッファを設定します。コントローラモードでは、転送を開始します。転送が完了するとストップコンディションまたはリスタートコンディションを送信します。

R_RI3C_Write()関数は FIFO 方式でデータ転送を行います。

### Example

```
uint8_t g_write_data[MAX_WRITE_DATA_LEN];

fsp_err_t err;

err = R_RI3C_Write(
   &g_ri3c0_ctrl, /* RI3C 制御ブロックへのポインタ */
    g_write_data, /* ライト転送用に割り当てられたメモリ */
    sizeof(g_write_data), /* 割り当てられた転送メモリのサイズ */
false /* リスタートコンディションのブール値 */
);
```

### Special Notes

なし。

## R_RI3C_Read()

転送に使用するリードバッファを設定します。コントローラモードでは、転送を開始します。転送が完了するとストップコンディションまたはリスタートコンディションを送信します。

### Format

```
fsp_err_t R_RI3C_Read(
        ri3c_ctrl_t *const p_api_ctrl,
        uint8_t const *const p_data,
        uint32_t length,
        bool restart
)
```

### Parameters

*p_api_ctrl*

RI3C 制御ブロックへのポインタ。

*p_data*

転送時に読み出したバイトデータを格納するバッファへのポインタ。

*length*

転送バイト数

*restart*

真の場合、転送完了後にリスタートコンディションを発行します（コントローラモード時のみ）。

### Return Values

*FSP_SUCCESS /* 処理に成功しました。 */*
*FSP_ERR_ASSERTION /* NULL ポインタがありました。 */*
*FSP_ERR_NOT_OPEN /* 制御ブロックが開いていません。 */*
*FSP_ERR_IN_USE /* ドライバがビジーのため動作を完了できなかった。 */*
*FSP_ERR_INVALID_MODE /* インスタンスは既に有効化されている。 */*
*FSP_ERR_INVALID_ALIGNMENT /* バッファは 4 バイトアラインで配置しなければならない。 */*

### Properties

r_ri3c_rx_if.h にプロトタイプ宣言されています。

### Description

転送に使用するリードバッファを設定します。コントローラモードでは、転送を開始します。転送が完了するとストップコンディションまたはリスタートコンディションを送信します。

R_RI3C_Read()関数は FIFO 方式でデータ転送を行います。

### Example

```
uint8_t g_read_data[MAX_READ_DATA_LEN];

fsp_err_t err;

err = R_RI3C_Read(
   &g_ri3c0_ctrl, /* RI3C 制御ブロックへのポインタ */
    g_read_data,  /* リード転送用に割り当てられたメモリ */
    sizeof(g_read_data), /* 割り当てられた転送メモリのサイズ */
    false /* リスタートコンディションのブール値 */
);
```

### Special Notes

なし。

## R_RI3C_IbiWrite()

IBI ライト動作を開始します。

### Format

```
fsp_err_t R_RI3C_IbiWrite(
        ri3c_ctrl_t *const p_api_ctrl,
        ri3c_ibi_type_t ibi_type,
        uint8_t const *const p_data,
        uint32_t length
)
```

### Parameters

*p_api_ctrl*

RI3C 制御ブロックへのポインタ。

*ibi_type*

IBI の種別。

*p_data*

転送時に読み出されるバイトデータを格納するバッファへのポインタ。

*length*

転送バイト数。

### Return Values

*FSP_SUCCESS /* 処理に成功しました。 */*
*FSP_ERR_ASSERTION /* NULL ポインタがありました。 */*
*FSP_ERR_NOT_OPEN /* 制御ブロックが開いていません。 */*
*FSP_ERR_IN_USE /* ドライバがビジーのため動作を完了できなかった */*
*FSP_ERR_INVALID_MODE /* この関数はターゲットモードでしか呼び出すことができない。 */*
*FSP_ERR_INVALID_ALIGNMENT /* バッファは 4 バイトアラインで配置しなければならない。 */*
*FSP_ERR_UNSUPPORTED /* ターゲットモードのサポートが無効化されている。 */*

### Properties

r_ri3c_rx_if.h にプロトタイプ宣言されています。

### Description

IBI ライト動作を開始します。

- デバイスに IBI ペイロードがある場合は、IBI のデータ部を書き込むためのバッファが設定されます。
- IBI を開始するコマンドディスクリプタを作成します。
- コマンドキューにディスクリプタを書き込みます。

### Example

```
fsp_err_t err;

err = R_RI3C_IbiWrite(
   &g_ri3c0_ctrl,
    RI3C_IBI_TYPE_HOT_JOIN,
    NULL,
    0
);
```

### Special Notes

なし。

## R_RI3C_IbiRead()

受信した IBI データを格納するリードバッファを設定します。

### Format

```
fsp_err_t R_RI3C_IbiRead(
        ri3c_ctrl_t *const p_api_ctrl,
        uint8_t *const p_data,
        uint32_t length
)
```

### Parameters

*p_api_ctrl*

RI3C 制御ブロックへのポインタ。

*p_data*

転送時に読み出されるバイトデータを格納するバッファへのポインタ。

*length*

転送バイト数。

### Return Values

*FSP_SUCCESS /* 処理に成功しました。 */*
*FSP_ERR_ASSERTION /* NULL ポインタがありました。 */*
*FSP_ERR_NOT_OPEN /* 制御ブロックが開いていません。 */*
*FSP_ERR_IN_USE /* ドライバがビジーのため動作を完了できなかった */*
*FSP_ERR_INVALID_MODE /* この関数はターゲットモードでしか呼び出すことができない。 */*
*FSP_ERR_INVALID_ALIGNMENT /* バッファは 4 バイトアラインで配置しなければならない。 */*
*FSP_ERR_UNSUPPORTED /* ターゲットモードのサポートが無効化されている。 */*

### Properties

r_ri3c_rx_if.h にプロトタイプ宣言されています。

### Description

受信した IBI データを格納するリードバッファを設定します。

### Example

```
uint8_t g_ibi_read_data[MAX_IBI_PAYLOAD_SIZE];

fsp_err_t err;

err = R_RI3C_IbiRead(
   &g_ri3c0_ctrl,
    g_ibi_read_data,
    MAX_IBI_PAYLOAD_SIZE
);
```

### Special Notes

この関数はコントローラモードでだけ使用できます。

## R_RI3C_Close()

RI3C インスタンスをクローズします。

### Format

```
fsp_err_t R_RI3C_Close(
        ri3c_ctrl_t *const p_api_ctrl,
)
```

### Parameters

*p_api_ctrl*

RI3C 制御ブロックへのポインタ。

### Return Values

*FSP_SUCCESS /* 処理に成功しました。 */*
*FSP_ERR_ASSERTION /* NULL ポインタがありました。 */*
*FSP_ERR_NOT_OPEN /* 制御ブロックが開いていません。 */*

### Properties

r_ri3c_rx_if.h にプロトタイプ宣言されています。

### Description

RI3C インスタンスをクローズします。

- RI3C 割り込み要求を無効にします。
- I3C バスの動作を禁止します。

### Example

```
fsp_err_t err;

err = R_RI3C_Close(&g_api_ctrl);
```

### Special Notes

なし。

## 4. 端子設定

RI3C FIT モジュールを使用する場合は、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当ててください。本ドキュメントでは、以降、端子の割り当ては「端子設定」と呼びます。

注: 端子設定は、R_RI3C_Open() 関数を呼び出した後に行ってください。

e$^2$ studio で端子設定を行う場合、FIT Configurator または Smart Configurator の端子設定機能が使用できます。端子設定機能の使用時、FIT Configurator または Smart Configurator の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生成されます。そのソースファイルで定義した関数を呼び出すことによって端子が設定されます。詳細については、「表 4.1 Smart Configurator が出力する関数。」を参照してください。

**表 4.1 Smart Configurator が出力する関数。**

| 使用マイコン | 出力される関数 | 備考 |
|---|---|---|
| RX26T | R_RI3C_PinSet_RI3C0() | |

# 5. サンプルコード

## 5.1 RI3C コントローラの基本例

アプリケーションにおいて RI3C コントローラを最小限に使用する基本例を示します。

この例では、コントローラの初期化を実装し、データを転送し、コールバック関数のプロトタイプを提供します。一連の手続きとして、モジュールレジスタへメモリを割り当て、ビットレートモードと共にデバイスロールを（1 次コントローラとして）設定し、プロトコル（I2C／I3C）を選択し、I3C プロトコル用ダイナミックアドレスを割り当て、モジュール割り込み要求を許可します。ライトバッファからデータをターゲットに送信し、リードバッファからデータを受信します。

```
void ri3c_controller_basic_example (void)
{
    /* モジュールを初期化する */
    fsp_err_t status = R_RI3C_Open(&g_ri3c_ctrl, &g_ri3c_cfg);
    assert(FSP_SUCCESS == status);
    static ri3c_device_cfg_t controller_device_cfg =
    {
        /* デバイスメーカが定義したスタティック I3C/I2C Legacy アドレス */
        .static_address  = EXAMPLE_CONTROLLER_STATIC_ADDRESS,
        /* メインコントローラであるデバイスは自身でダイナミックアドレスを設定しなければならない */
        .dynamic_address = EXAMPLE_CONTROLLER_DYNAMIC_ADDRESS,
    };
    status = R_RI3C_DeviceCfgSet(&g_ri3c_ctrl, &controller_device_cfg);
    assert(FSP_SUCCESS == status);
    static ri3c_device_table_cfg_t device_table_cfg =
    {
        /* デバイスメーカが定義したスタティック I3C/I2C Legacy アドレス */
        .static_address        = EXAMPLE_STATIC_ADDRESS,
        /* ダイナミックアドレスは I2C では使用しない */
        .dynamic_address       = EXAMPLE_DYNAMIC_ADDRESS,
        /* デバイスの種別 （I2C デバイス／I3C デバイス） */
        .device_protocol       = I3C_DEVICE_PROTOCOL_I3C,
        .ibi_accept            = false,
        /* デバイスによっては IBI 要求にデータペイロードあり
         * デバイスが ENTDAA で設定される場合、このフィールドは自動的に更新される
         */
        .ibi_payload           = false,
        /* 2 次コントローラが未サポートのためコントローラ要求は受け付け不可 */
        .controllerrole_request_accept = false,
    };
    /* コントローラテーブルにデバイスコンフィグレーションを設定する */
    status = R_RI3C_ControllerDeviceTableSet(&g_ri3c_ctrl, 0, &device_table_cfg);
    assert(FSP_SUCCESS == status);
    /* RI3C デバイスを有効化する */
    status = R_RI3C_Enable(&g_ri3c_ctrl);
    assert(FSP_SUCCESS == status);
    /* ENTDAA コマンドによるバス上のデバイスへのダイナミックアドレスの割り当てを開始する */
    status = R_RI3C_DynamicAddressAssignmentStart(&g_ri3c_ctrl,
                                                  I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA,
                                                  0,
                                                  1);
    assert(FSP_SUCCESS == status);
    /* ダイナミックアドレス割り当ての完了を待つ */
    ri3c_app_event_wait(RI3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
    /* 次の動作について設定済デバイスとビットレートモードを選択する */
    status = R_RI3C_DeviceSelect(&g_ri3c_ctrl, 0, I3C_BITRATE_MODE_I3C_SDR0_STDBR);
    assert(FSP_SUCCESS == status);
    /* ライト転送を開始する */
    static uint8_t p_write_buffer[] = {1, 2, 3, 4, 5};
    status = R_RI3C_Write(&g_ri3c_ctrl, p_write_buffer, sizeof(p_write_buffer), false);
    assert(FSP_SUCCESS == status);
    /* ライト転送の完了を待つ */
    ri3c_app_event_wait(RI3C_EVENT_WRITE_COMPLETE);
    /* リード転送を開始する */
    static uint8_t p_read_buffer[16];
    status = R_RI3C_Read(&g_ri3c_ctrl, p_read_buffer, sizeof(p_read_buffer), false);
    assert(FSP_SUCCESS == status);
    /* リード転送の完了を待つ */
    ri3c_app_event_wait(RI3C_EVENT_READ_COMPLETE);
}
```

```
/* この関数は各 ISR から I3C ドライバによって呼び出されアプリケーションに I3C イベントを通知する */
void ri3c_controller_basic_example_callback (ri3c_callback_args_t const * const p_args)
{
    switch (p_args->event)
    {
        case RI3C_EVENT_ENTDAA_ADDRESS_PHASE:
        {
            /* デバイスの PID レジスタ、DCR レジスタ、および BCR レジスタは、
             * ri3c_callback_args_t::p_target_info で取得できる */
            break;
        }
        case RI3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE:
        {
            ri3c_app_event_notify(RI3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
            break;
        }
        case RI3C_EVENT_WRITE_COMPLETE:
        {
            ri3c_app_event_notify(RI3C_EVENT_WRITE_COMPLETE);
            break;
        }
        case RI3C_EVENT_READ_COMPLETE:
        {
            /* ターゲットから読み出したバイト数は
             * ri3c_callback_args_t::transfer_size で取得できる */
            ri3c_app_event_notify(RI3C_EVENT_READ_COMPLETE);
            break;
        }
        default:
        {
            break;
        }
    }
}
```

## 5.2 RI3C ターゲットの基本例

アプリケーションにおいて RI3C ターゲットを最小限に使用する基本例を示します。

この例では、ターゲットの初期化を実装し、コールバック関数のプロトタイプを提供します。一連の手続きとして、モジュールレジスタへメモリを割り当て、スタティックアドレスおよびターゲット情報と共にデバイスロールを（ターゲットとして）設定し、データ転送用メモリを割り当てます。

```
void ri3c_target_basic_example (void)
{
    /* モジュールを初期化する */
    fsp_err_t status = R_RI3C_Open(&g_ri3c_ctrl, &g_ri3c_cfg);
    assert(FSP_SUCCESS == status);
    static ri3c_device_cfg_t target_device_cfg =
    {
        /* デバイスメーカが定義したスタティック I3C/I2C Legacy アドレス */
        .static_address  = EXAMPLE_STATIC_ADDRESS,
        /* コントローラが ENTDAA で本デバイスを設定する際にダイナミックアドレスは
         * 自動的に更新される */
        .dynamic_address = 0,
        /* コントローラが読み出すデバイスレジスタ */
        .target_info     =
        {
            .bcr = EXAMPLE_BCR_SETTING,
            .dcr = EXAMPLE_DCR_SETTING,
            .pid =
            {
                0, 1, 2, 3, 4, 5
            }
        }
    };
    /* 本デバイスのデバイスコンフィグレーションを設定する */
    status = R_RI3C_DeviceCfgSet(&g_ri3c_ctrl, &target_device_cfg);
    assert(FSP_SUCCESS == status);
    /* ターゲットモードを有効化する */
    status = R_RI3C_Enable(&g_ri3c_ctrl);
    assert(FSP_SUCCESS == status);
    static uint8_t p_read_buffer[EXAMPLE_READ_BUFFER_SIZE];
    static uint8_t p_write_buffer[EXAMPLE_WRITE_BUFFER_SIZE];
    /* リード転送時に受信するデータを格納するバッファを設定する */
    status = R_RI3C_Read(&g_ri3c_ctrl, p_read_buffer, sizeof(p_read_buffer), false);
    assert(FSP_SUCCESS == status);
    /* コントローラによるリード転送の完了を待つ */
    ri3c_app_event_wait(RI3C_EVENT_READ_COMPLETE);
    /* ライト転送時に送信されるライトバッファを設定する */
    status = R_RI3C_Write(&g_ri3c_ctrl, p_write_buffer, sizeof(p_write_buffer), false);
    assert(FSP_SUCCESS == status);
    /* コントローラによるライト転送の完了を待つ */
    ri3c_app_event_wait(RI3C_EVENT_WRITE_COMPLETE);
}
```

```
void ri3c_target_basic_example_callback (ri3c_callback_args_t const * const p_args)
{
    switch (p_args->event)
    {
        case RI3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE:
        {
            ri3c_app_event_notify(RI3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
            break;
        }
        case RI3C_EVENT_READ_BUFFER_FULL:
        {
            /* ユーザー指定リードバッファがない場合、またはユーザー指定リードバッファが満杯の場合、
             * ドライバはアプリケーションにバッファフルを通知する。
             * アプリケーションが R_RI3C_READ()関数を呼び出して新しいリードバッファを用意する場合もある。
             * リードバッファが用意されない場合、残り転送バイトは破棄される。 */
            uint8_t * p_read_buffer = ri3c_app_next_read_buffer_get();
            R_RI3C_Read(&g_ri3c_ctrl, p_read_buffer, EXAMPLE_READ_BUFFER_SIZE, false);
            break;
        }
        case RI3C_EVENT_READ_COMPLETE:
        {
            /* ターゲットが読み出したデータのバイト数は
             * ri3c_callback_args_t::transfer_size で取得できる */
            ri3c_app_event_notify(RI3C_EVENT_READ_COMPLETE);
            /* 次転送で使用する転送バッファを設定するためにアプリケーションがこのイベントから R_RI3C_READ()
             * または R_RI3C_WRITE()を呼び出す場合もある */
            break;
        }
        case RI3C_EVENT_WRITE_COMPLETE:
        {
            /* ターゲットが書き込んだデータのバイト数は
             * ri3c_callback_args_t::transfer_size で取得できる */
            ri3c_app_event_notify(RI3C_EVENT_WRITE_COMPLETE);
            /* 次転送で使用する転送バッファを設定するためにアプリケーションがこのイベントから R_RI3C_READ()
             * または R_RI3C_WRITE()を呼び出す場合もある
             */
            break;
        }
        default:
        {
            break;
        }
    }
}
    /* リード転送の完了を待つ */
    ri3c_app_event_wait(RI3C_EVENT_READ_COMPLETE);
}
```

# 6. 付録

## 6.1 動作確認環境

本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 | Contents |
|---|---|
| 統合開発環境 | ルネサスエレクトロニクス製 e$^2$ studio Version 2022-10 (22.10.0)<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 RX ファミリ用 C/C++コンパイラパッケージ V3.05.00<br>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します。<br>-lang = c99 |
| | GCC for Renesas RX 8.3.0.202204<br>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します。<br>-std=gnu99<br>リンクオプション：「Optimize size (サイズ最適化) (-Os)」を使用する場合、統合開発環境のデフォルト設定に以下のユーザ定義オプションを追加します。<br>-Wl, --no-gc-sections<br>これは、FIT 周辺モジュール内で宣言された割り込み関数をリンカが誤って破棄してしまうという GCC リンカ問題を回避するための対策です。 |
| | IAR C/C++ Compiler for Renesas RX version 4.20.3<br>コンパイルオプション：統合開発環境のデフォルト設定 |
| エンディアン | 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 Flexible Motor Control Kit for RX26T (型名：RTK0EMXE70S00020BJ) |

## 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_ri3c_rx module.」エラーが発生します。

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

(3) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「コンフィグ設定が間違っている場合のエラーメッセージ」エラーが発生します。

A：「r_ri3c_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_ri3c_rx_config.h」ファイルを確認して正しい値を設定してください。詳細は「2.7 コンパイル時の設定」を参照してください。

## 7. 参考ドキュメント

ユーザーズマニュアル：ハードウェア
（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデート／テクニカルニュース
（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：開発環境
RX ファミリ CC-RX コンパイラ 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。

なし

## 改訂記録

| Rev. | 発行日 | 改訂内容 ページ | Summary |
|---|---|---|---|
| 1.00 | Aug 15, 2022 | — | 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。