RX ファミリ
EEPROM アクセス クロック同期式制御モジュール Firmware Integration Technology

要旨

本アプリケーションノートでは、ルネサス エレクトロニクス製 MCU を使用したルネサス エレクトロニクス製 R1EX25xxx/HN58X25xxx シリーズの SPI Serial EEPROM 制御方法とその使用方法を説明します。

なお、本制御ソフトウェアは、スレーブデバイスとして Serial EEPROM を制御するための上位層に位置するソフトウェアです。

別途、マスタデバイスとしての各 MCU 個別の SPI モードを制御するための下位層に位置するソフトウェア（クロック同期式シングルマスタ制御ソフトウェア）を用意していますので、以下の URL から入手してください。なお、クロック同期式シングルマスタ制御ソフトウェアにおいて、新規 MCU が対応可能になった場合でも、本アプリケーションノートの更新が間に合わないことがあります。最新のサポート MCU とその制御ソフトウェアの組み合わせ情報は、以下の URL のページに記載されている「クロック同期式シングルマスタドライバ（下位層ソフトウェア）」を参照してください。

- SPI シリアル EEPROM 制御

  http://japan.renesas.com/driver/spi_serial_eeprom

本制御ソフトウェアは、Firmware Integration Technology（以下、FIT と略す）を使用しています。FIT 対応開発環境での説明では、本制御ソフトウェアを EEPROM FIT モジュールと称します。また、同様に FIT を使った他の機能制御モジュールを FIT モジュールもしくは“機能名”FIT モジュールと表します。

また、FIT 未対応の開発環境では、FIT 機能を無効に設定し組み込みが可能です。

対象デバイス

Serial EEPROM　　ルネサス エレクトロニクス R1EX25xxx シリーズ SPI Serial EEPROM
RX ファミリ MCU

動作確認に使用した MCU
- RX111、RX110、RX113、RX130 グループ（RSPI）
- RX230、RX231、RX23T、RX24T グループ（RSPI）
- RX64M、RX71M グループ（RSPI、QSPI、SCI）
- RX72T、RX72N グループ（RSPI、SCI）

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合わせて変更し、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「4.1 動作確認環境」を参照してください。

目次

1. 概要 4
1.1 Serial EEPROM 制御ソフトウェアの FIT 対応化について 5
1.2 API の概要 5
1.3 関連アプリケーションノート 6
1.3.1 FIT モジュール関連のアプリケーションノート 6
1.4 ハードウェア設定 7
1.4.1 ハードウェア構成例 7
1.5 ソフトウェア説明 8
1.5.1 動作概要 8
1.5.2 SPI モードで制御 8
1.5.3 Serial EEPROM の Chip select 端子制御 8
1.5.4 ソフトウェア構成 9
1.5.5 本制御ソフトウェアとクロック同期式シングルマスタ制御ソフトウェアの関係 10
1.5.6 データバッファと送信／受信データの関係 11
1.5.7 状態遷移図 12

2. API 情報 13
2.1 ハードウェアの要求 13
2.2 ソフトウェアの要求 13
2.3 サポートされているツールチェイン 13
2.4 ヘッダファイル 13
2.5 整数型 13
2.6 コンパイル時の設定 14
2.7 引数 15
2.8 コードサイズ 16
2.9 戻り値 18
2.10 ソフトウェアの追加方法 19
2.11 FIT モジュール以外の環境で使用する場合の組み込み方法 20
2.12 端子の状態 21
2.13 for 文、while 文、do while 文について 22

3. API 関数 23
3.1 R_EEPROM_SPI_Open() 23
3.2 R_EEPROM_SPI_Close() 24
3.3 R_EEPROM_SPI_Read_Status() 25
3.4 R_EEPROM_SPI_Set_Write_Protect() 27
3.5 R_EEPROM_SPI_Write_Di() 30
3.6 R_EEPROM_SPI_Read_Data() 31
3.7 R_EEPROM_SPI_Write_Data_Page() 33
3.8 R_EEPROM_SPI_Polling() 38
3.9 R_EEPROM_SPI_GetMemoryInfo() 39
3.10 R_EEPROM_SPI_GetVersion() 40
3.11 R_EEPROM_SPI_Set_LogHdlAddress() 41

3.12 R_EEPROM_SPI_Log() 43
3.13 R_EEPROM_SPI_1ms_Interval() 45

4. 付録 46
4.1 動作確認環境 46

5. 参考ドキュメント 48

テクニカルアップデートの対応について 48

改訂記録 49

1. 概要

ルネサス エレクトロニクス製 MCU を使用し、ルネサス エレクトロニクス製 R1EX25xxx/HN58X25xxx シリーズの SPI Serial EEPROM を制御します。

別途、MCU 個別のクロック同期式シングルマスタ制御ソフトウェアが必要です。

表 1-1 に使用する周辺機器と用途を、図 1-1 に使用例を示します。

以下に、機能概略を示します。

- マスタデバイスをルネサス エレクトロニクス製 MCU、スレーブデバイスをルネサス エレクトロニクス製 R1EX25xxx/HN58X25xxx シリーズの SPI Serial EEPROM としたブロック型デバイスドライバ
- MCU 内蔵のシリアル通信機能（クロック同期式モード）を使用し、SPI モードで制御

  組み合わせ対象のシリアル通信FITモジュール（「1.3.1 FITモジュール関連のアプリケーションノート」を参照してください。）

  ・RSPI FITモジュール

  ・QSPI FITモジュール

  ・SCI FITモジュール
- 最大 2 個の SPI Serial EEPROM の制御が可能
- デバイス毎に SPI Serial EEPROM を設定可能。
- ビッグエンディアン／リトルエンディアンでの動作が可能（指定デバイス依存）

表 1-1　使用する周辺機器と用途

周辺機器	用途
MCU 内蔵のシリアル通信（クロック同期式モード）	シリアル通信機能（クロック同期式モード）による SPI スレーブデバイスとの通信：1 もしくは複数チャネル（必須）
Port	スレーブデバイスセレクト制御信号用：使用デバイス数分のポートが必要（必須）

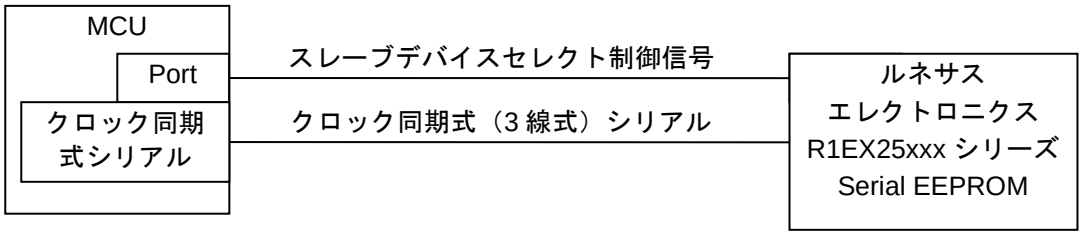


図 1-1　使用例

注意事項：

IAR コンパイラのビッグエンディアンは RSPI モジュールを正常に動作させていませんので、この場合 RSPI モジュールを使用しないでください。

1.1 Serial EEPROM 制御ソフトウェアの FIT 対応化について

Serial EEPROM 制御ソフトウェアは、他の FIT モジュールと組み合わせることにより、組み込みが容易になります。

また、Serial EEPROM 制御ソフトウェアは API として、プロジェクトに組み込んで使用します。Serial EEPROM 制御ソフトウェアの組み込み方については、「2.10 ソフトウェアの追加方法」を参照してください。

1.2 API の概要

表 1-2 に Serial EEPROM 制御ソフトウェアに含まれる API 関数を示します。

表 1-2 API 関数

関数名	説明
R_EEPROM_SPI_Open()	本制御ソフトウェアの初期化処理
R_EEPROM_SPI_Close()	本制御ソフトウェアの終了処理
R_EEPROM_SPI_Read_Status()	ステータスレジスタ読み出し処理
R_EEPROM_SPI_Set_Write_Protect()	ライトプロテクト設定処理
R_EEPROM_SPI_Write_Di()	WRDI コマンド処理
R_EEPROM_SPI_Read_Data()　注１	データ読み出し処理
R_EEPROM_SPI_Write_Data_Page()　注１	データ書き込み（1Page 書き込み用）処理
R_EEPROM_SPI_Polling()	ポーリング処理
R_EEPROM_SPI_GetMemoryInfo()	メモリサイズ取得処理
R_EEPROM_SPI_GetVersion()	本制御ソフトウェアのバージョン情報取得処理
R_EEPROM_SPI_Set_LogHdlAddress()	LONGQ FIT モジュールのハンドラアドレス設定処理
R_EEPROM_SPI_Log()	LONGQ FIT モジュールを使ったエラーログ取得処理
R_EEPROM_SPI_1ms_Interval()　注２	クロック同期式シングルマスタ制御ソフトウェアのインターバルタイマカウンタ処理

注１：データ転送の高速化のために、送信／受信データ格納バッファポインタを指定する場合、開始アドレスを 4 バイト境界に合わせてください。DMAC 転送もしくは DTC 転送を使用する場合、データサイズの制限があります。設定可能なデータサイズについては、使用する MCU 用のクロック同期式シングルマスタ制御ソフトウェアを確認してください。

注２：DMAC 転送もしくは DTC 転送を使用する場合、タイムアウト検出のため、ハードウェアタイマやソフトウェアタイマ等を使って、1ms 間隔でコールする必要があります。

1.3　関連アプリケーションノート

Serial EEPROM 制御ソフトウェアに関連するアプリケーションノートを以下に示します。合わせて参照してください。

1.3.1　FIT モジュール関連のアプリケーションノート

- RX ファミリ RSPI Firmware Integration Technology（R01AN1827）
- RX ファミリ QSPI クロック同期式シングルマスタ制御モジュール Firmware Integration Technology（R01AN1940）
- RX ファミリ SCI Firmware Integration Technology（R01AN1815）
- RX ファミリ DMAC モジュール Firmware Integration Technology（R01AN2063）
- RX ファミリ DTC モジュール Firmware Integration Technology（R01AN1819）
- RX ファミリ CMT モジュール Firmware Integration Technology（R01AN1856）
- RX Family GPIO Driver Module Using Firmware Integration Technology（R01AN1721）
- RX Family MPC Module Using Firmware Integration Technology（R01AN1724）
- RX Family LONGQ Module Using Firmware Integration Technology（R01AN1889）
- RX ファミリ Serial Flash memory アクセス クロック同期式制御モジュール Firmware Integration Technology（R01AN2662）
- RX ファミリ メモリアクセス用ドライバインタフェース Firmware Integration Technology(R01AN4548)

1.4　ハードウェア設定

1.4.1　ハードウェア構成例

図 1-2 に接続図を示します。MCU とシリアル・インタフェースにより端子名が異なります。表 1-3 に使用端子と機能を参照し、使用する MCU の端子に割り当ててください。

なお、高速で動作させた場合を想定し、各信号ラインの回路的マッチングを取るためのダンピング抵抗やコンデンサの付加を検討してください。

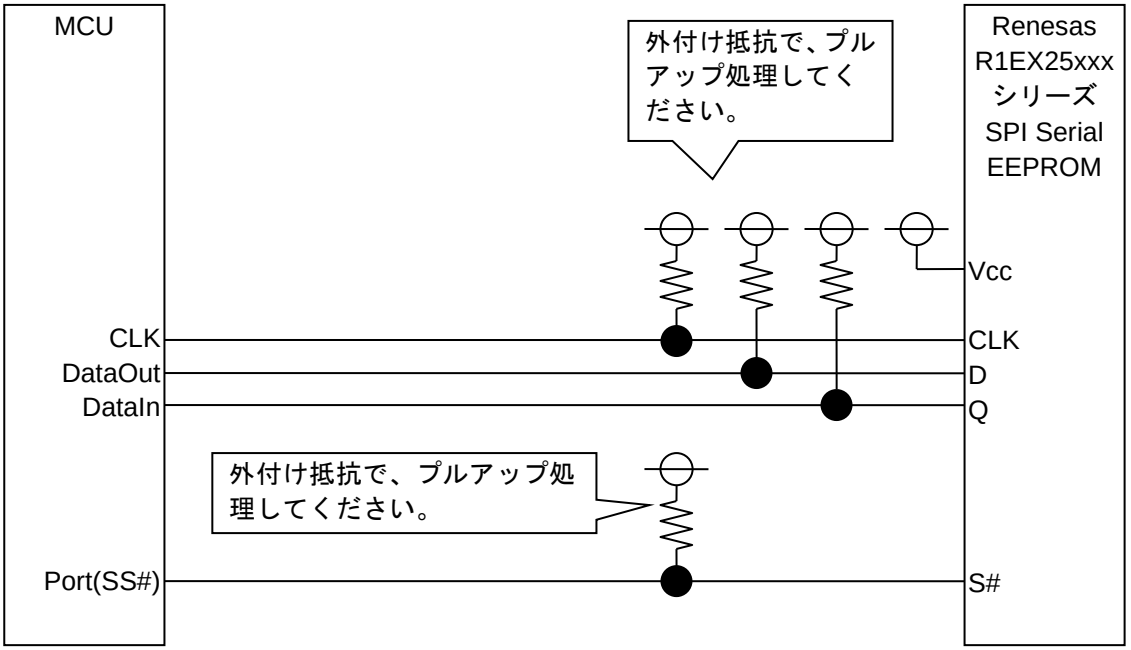


図 1-2　MCUI と SPI スレーブデバイスの接続例

表 1-3　使用端子と機能

端子名	入出力	内容
CLK	出力	クロック出力
DataOut	出力	マスタデータ出力
DataIn	入力	マスタデータ入力
Port（図 1-2 の Port(SS#)）	出力	スレーブデバイスセレクト(SS#)出力

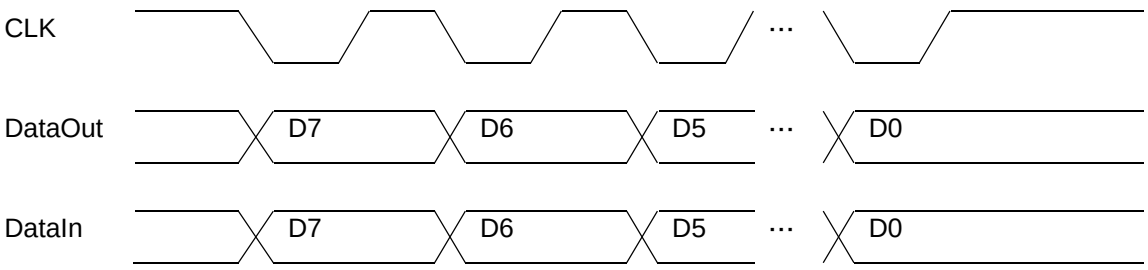
1.5　ソフトウェア説明

1.5.1　動作概要

MCU のクロック同期式シリアル通信機能を使って、内部クロックを使用したクロック同期式シングルマスタ制御を実現します。

1.5.2　SPI モードで制御

図 1-3 に示す SPI モード 3（CPOL=1、CPHA=1）で制御します。



・MCU->スレーブデバイスの送信時：転送クロックの立ち下がりで送信データ出力開始

・スレーブデバイス->MCU の受信時：転送クロックの立ち上がりで受信データの入力取り込み

・MSB ファーストでの転送

・転送を行っていないときの CLK 端子のレベル：”H”

図 1-3　制御可能なスレーブデバイスのタイミング

使用可能なシリアルクロック周波数は、MCU のユーザーズマニュアル ハードウェア編およびスレーブデバイスのデータシートで、確認してください。

1.5.3　Serial EEPROM の Chip select 端子制御

Serial EEPROM の Chip select 端子を MCU の Port に接続し、MCU 汎用ポート出力で制御します。

Serial EEPROM の Chip select（MCU の Port(SS#)）信号の立ち下がりから、Serial EEPROM の Clock（MCU の CLK）信号の立ち下がりまでの時間は、Serial EEPROM の Chip select セットアップ時間待ちのために、ソフトウェア・ウェイトで制御しています。

Serial EEPROM の Clock（MCU の CLK）信号の立ち上がりから、Serial EEPROM の Chip select（MCU の Port(SS#)）信号の立ち上がりまでの時間は、Serial EEPROM の Chip select ホールド時間待ちのために、ソフトウェア・ウェイトで制御しています。

本モジュールでは、Chip select セットアップ時間待ち、および Chip select ホールド時間待ちを約 1us としています。

1.5.4　ソフトウェア構成

図 1-4 にソフトウェア構成を示します。

本制御ソフトウェアを使用して、スレーブデバイスを制御するためのソフトウェアを作成してください。

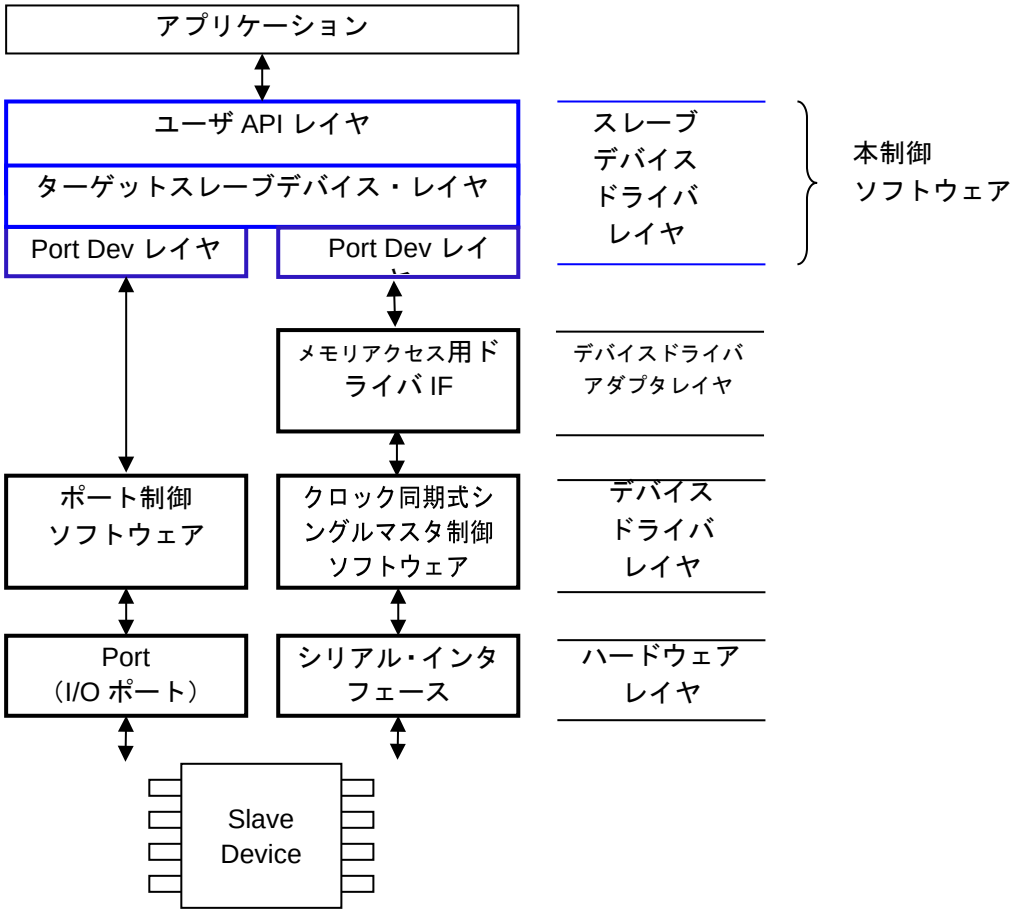


図 1-4　ソフトウェア構成

(1)　ユーザ API レイヤ（r_eeprom_spi.c）

EEPROM 制御部で、下位層のデバイスドライバに依存しない部分です。

(2)　サブモジュール・レイヤ（r_eeprom_spi_sub.c）

EEPROM 制御サブモジュールで、下位層のデバイスドライバに依存しない部分です。

(3)　ドライバ・インタフェース（I/F）・レイヤ（r_eeprom_spi_drvif.c）

下位層のデバイスドライバとの接続部分、共通ドライバ変換レイヤです。

MCU 毎のクロック同期式シングルマスタ制御ソフトウェア毎に、ドライバ I/F 関数が必要です。

(4)　Port Dev レイヤ（r_eeprom_spi_dev_port.c）

スレーブデバイスセレクト信号（SS#）をポートで制御するための制御部分です。

GPIO FIT モジュールと MPC FIT モジュールを使用できます。

(5)　アプリケーション

ルネサス エレクトロニクス製 R1EX25xxx シリーズ Serial EEPROM の制御例を同梱していますので、参照してください。

1.5.5　本制御ソフトウェアとクロック同期式シングルマスタ制御ソフトウェアの関係

本制御ソフトウェアとクロック同期式シングルマスタ制御ソフトウェアの組み合わせ方法について説明します。

最大 2 個のスレーブデバイスを最大 2 種類のクロック同期式シングルマスタ制御ソフトウェアを使って、制御できます。ドライバ I/F 関数に、使用するクロック同期式シングルマスタ制御ソフトウェアを登録してください。

以下のように、デバイス毎にデバイスドライバを設定できます。ドライバ・インタフェース・レイヤのドライバ I/F 関数にデバイス番号毎に、使用するデバイスドライバ API を使った処理を作成してください。

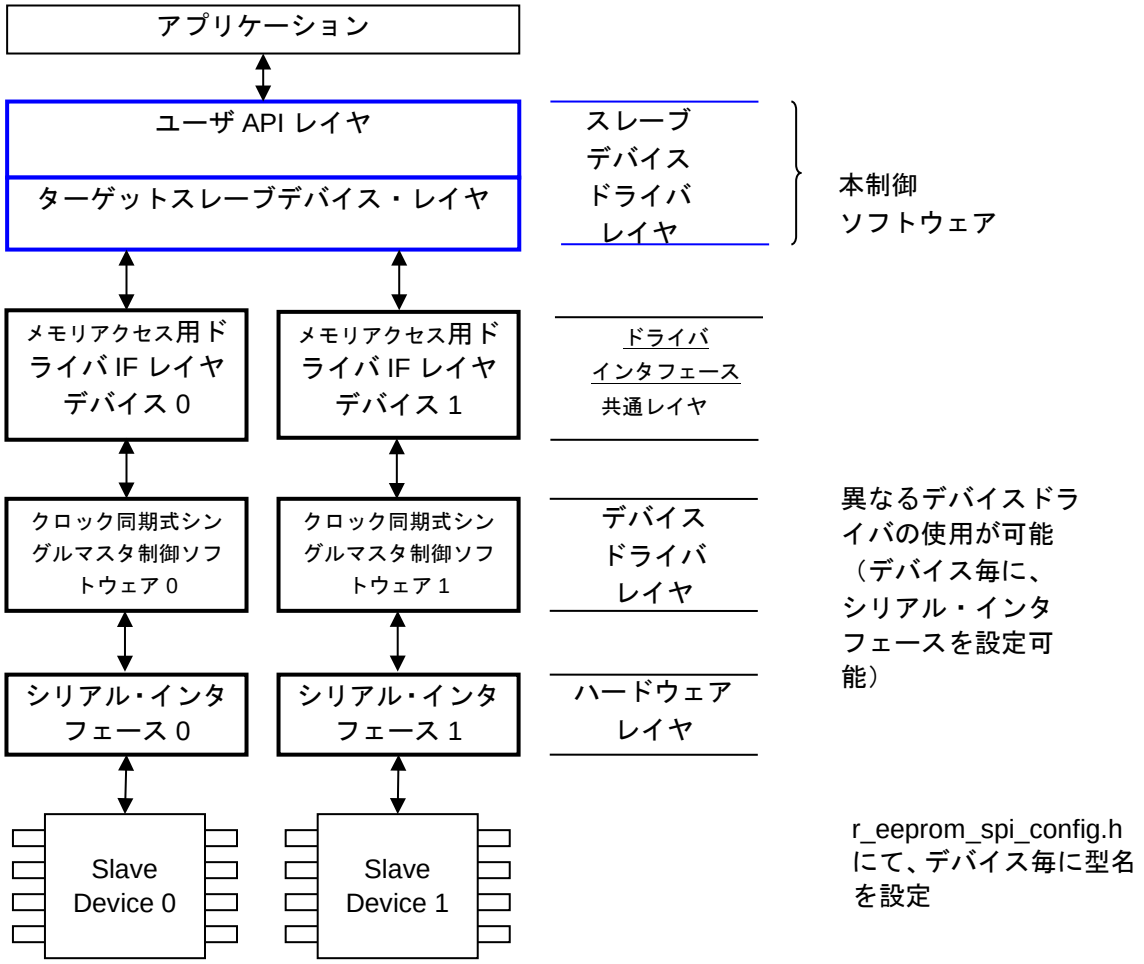


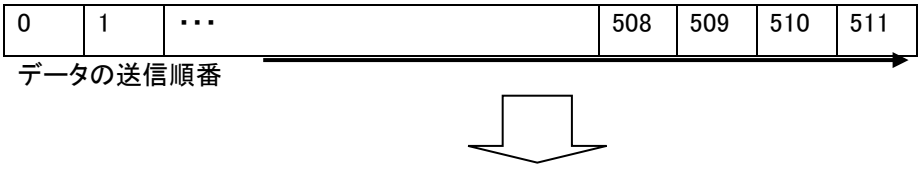
図 1-5　クロック同期式シングルマスタ制御ソフトとウェアとのソフトウェア構成

1.5.6　データバッファと送信／受信データの関係

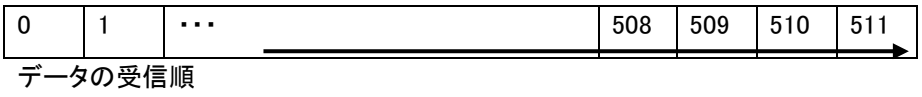
本制御ソフトウェアは、ブロック型デバイスドライバであり、送信／受信データポインタを引数として設定します。RAM 上のデータバッファのデータ並びと送信／受信順番の関係は、以下のとおりで、エンディアンや使用するシリアル通信機能に関係なく、送信データバッファの並びの順に送信し、また、受信の順に受信データバッファに書き込みます。

マスタ送信時

RAM 上の送信データバッファ（バイト表示）

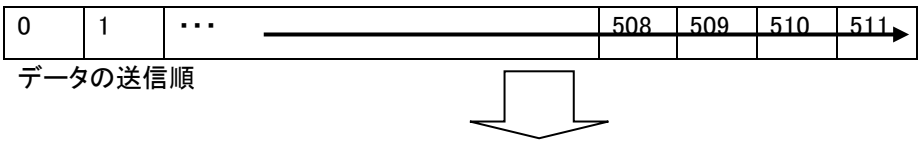


スレーブデバイスへの書き込み（バイト表示）



マスタ受信時

スレーブデバイスからの読み出し（バイト表示）



RAM 上のデータバッファ（バイト表示）

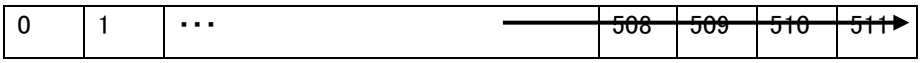


図 1-6　データバッファと送信／受信データの関係

1.5.7 状態遷移図

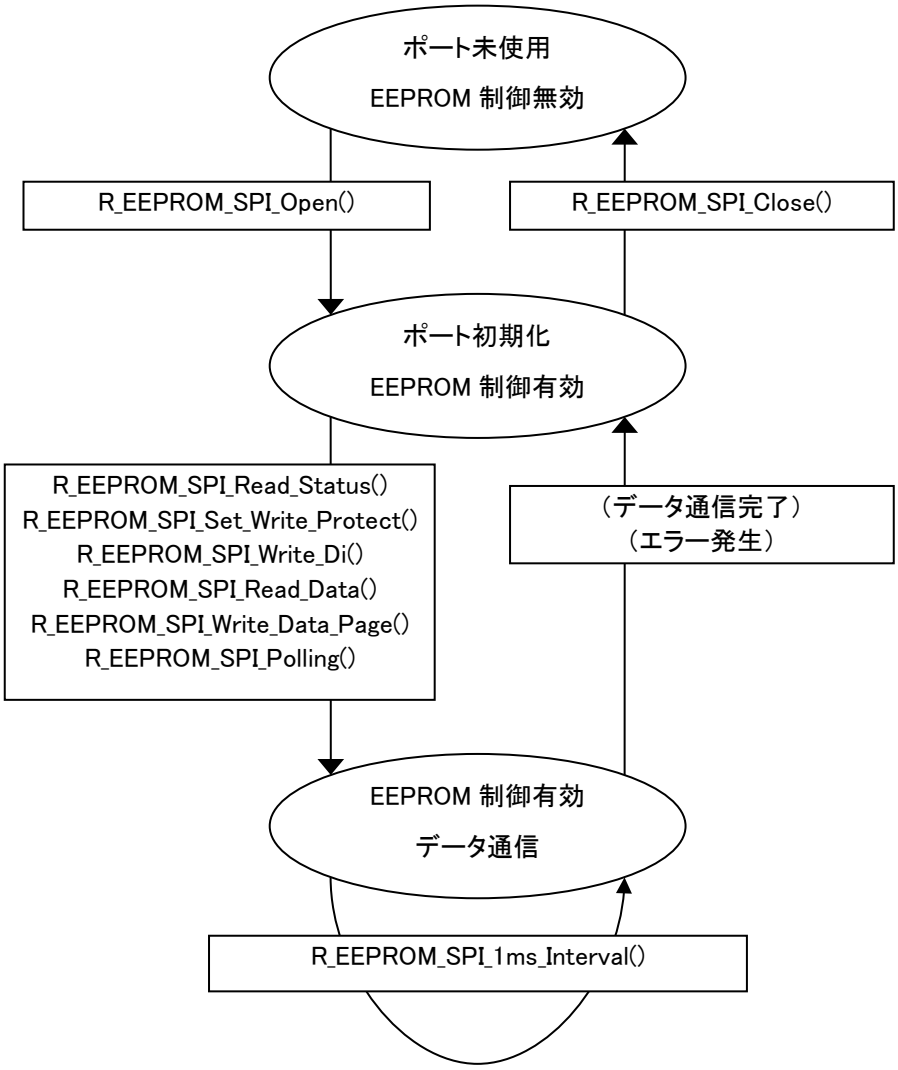


図 1-7 状態遷移図

2. API 情報

本制御ソフトウェアの API は、ルネサスの API 命名基準に従っています。

2.1 ハードウェアの要求

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。なお、別途クロック同期式シングルマスタ制御ソフトウェアを用意してください。

- I/O ポート

2.2 ソフトウェアの要求

本制御ソフトウェアを FIT 対応させて使用する場合、以下のパッケージに依存しています。

- r_bsp Rev.5.00 以上
- r_memdrv_rx
- r_rspi _rx（RSPI FIT モジュールを使用する場合）
- r_qspi_smstr_rx（クロック同期式シングルマスタ制御に QSPI FIT モジュールを使用する場合）
- r_sci_rx（SCI FIT モジュールを使用する場合）
- r_dmaca_rx（DMACA FIT モジュールを用いて、DMAC 転送を使用する場合のみ）
- r_dtc_rx（DTC FIT モジュールを用いて、DTC 転送を使用する場合のみ）
- r_cmt_rx（DMAC 転送もしくは DTC 転送を使用し、かつコンペアマッチタイマ CMT FIT モジュールを使用する場合のみ）

他タイマやソフトウェアタイマで代用できます。

- r_gpio_rx（GPIO, MPC FIT モジュールを用いて、GPIO を制御する場合のみ）
- r_mpc_rx（GPIO, MPC FIT モジュールを用いて、MPC を制御する場合のみ）

2.3 サポートされているツールチェイン

本制御ソフトウェアは、「4.1 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 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_eeprom_spi_if.h に記載しています。

ビルド毎の構成オプションは、r_eeprom_spi_config.h と r_eeprom_spi_pin_config.h で選択します。以下の順番でインクルードしてください。なお、r_eeprom_spi_pin_config.h のインクルードは不要です。

```
#include "r_eeprom_spi_if.h"
#include "r_eeprom_spi_config.h"
```

2.5 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。

2.6 コンパイル時の設定

本制御ソフトウェアのコンフィギュレーションオプションの設定は、r_eeprom_spi_config.h と r_eeprom_spi_pin_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_eeprom_spi_config.h*	
#define EEPROM_SPI_CFG_WEL_CHK ※デフォルト値は“有効”	WREN コマンド発行後に、WEL ビット確認を実行するかを選択できます。
#define EEPROM_SPI_CFG_LONGQ_ENABLE ※デフォルトは“無効”	FIT モジュールの BSP 環境で使用する場合、デバッグ用のエラーログ取得処理を使用するか選択できます。 無効にした場合、処理をコードから省略します。 有効にした場合、処理をコードに含めます。 使用するためには、別途 LONGQ FIT モジュールが必要です。 また、指定デバイスで使用するクロック同期式シングルマスタ制御ソフトウェアの#define xxx_LONGQ_ENABLE を有効にしてください。
#define EEPROM_SPI_CFG_USE_GPIO_MPC_FIT	SS#端子の制御に GPIO FIT モジュール、MPC FIT モジュールを使用するか選択できます。 無効にした場合、GPIO FIT モジュール、MPC FIT モジュールを使用せずに SS#端子を制御します。 有効にした場合、GPIO FIT モジュール、MPC FIT モジュールを使用し、SS#端子を制御します。 使用するためには別途 GPIO FIT モジュール、MPC FIT モジュールが必要です。
define EEPROM_SPI_CFG_DEVx_INCLUDED ※デバイス 0 のデフォルト値は、“有効” ※DEVx の“x ”はデバイス番号（x=0 or 1）	デバイス x に関する定義です。 デバイス x に関して、必ず有効にしてください。
#define EEPROM_SPI_CFG_DEVx_SIZE_002K #define EEPROM_SPI_CFG_DEVx_SIZE_004K #define EEPROM_SPI_CFG_DEVx_SIZE_008K #define EEPROM_SPI_CFG_DEVx_SIZE_016K #define EEPROM_SPI_CFG_DEVx_SIZE_032K #define EEPROM_SPI_CFG_DEVx_SIZE_064K #define EEPROM_SPI_CFG_DEVx_SIZE_128K #define EEPROM_SPI_CFG_DEVx_SIZE_256K #define EEPROM_SPI_CFG_DEVx_SIZE_512K ※デバイス 0 のデフォルト値は “EEPROM_SPI_CFG_DEVx_SIZE_008K ” ※DEVx の“x”はデバイス番号（x=0 or 1）	デバイス x の EEPROM を 1 つだけ設定してください。
#define EEPROM_SPI_CS_DEVx_CFG_PORTNO ※デバイス 0 のデフォルト値は“‘X’” ※DEVx の“x”はデバイス番号（x=0 or 1）	デバイス x 用の SS#を割り付けるポート番号を設定してください。 設定値の前後にシングルコーテーション「‘」「’」をつけてください。 デバイス X のポート番号を 0 - 9, A - X で構成してください。
#define EEPROM_SPI_CS_DEVx_CFG_BITNO ※デバイス 0 のデフォルト値は“‘0’” ※DEVx の“x”はデバイス番号（x=0 or 1）	デバイス x 用の SS#を割り付けるビット番号を設定してください。 設定値の前後にシングルコーテーション「‘」「’」をつけてください。 デバイス X のビット番号を 0 - 7 で構成してください。

2.7 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_eeprom_spi_if.h で記載されています。

```
/* EEPROM information */
typedef struct
{
   uint32_t    addr;               /* Address to issue a command          */
   uint32_t    cnt;                /* Number of bytes to be read/written  */
   uint32_t    data_cnt;
/* Temporary counter or Number of bytes to be written in a page    */
   uint8_t     * p_data;           /* Data storage buffer pointer         */
} eeprom_info_t;                   /* 16 bytes                            */

/* EEPROM size information */
typedef struct
{
   uint32_t    mem_size;           /* Max memory size                     */
   uint32_t    wpag_size;          /* Write page size                     */
} eeprom_mem_info_t;               /* 8 bytes                             */
```

2.8 コードサイズ

本モジュールの ROM サイズ、RAM サイズ、最大使用スタック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「2.6 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。

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_eeprom_spi rev3.10

コンパイラバージョン: Renesas Electronics C/C++ Compiler Package for RX Family V3.04.00
(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 8.03.00.202104
(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 4.20.3
(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

動作周波数: RX111, RX113 ICLK：32MHz、PCLKB：32MHz
RX231 ICLK：54MHz、PCLKB：27MHz
RX64M ICLK：120MHz、PCLKA：120MHz、PCLKB：60MHz
RX71M ICLK：240MHz、PCLKA：120MHz、PCLKB：60MHz

電源電圧: 3.3V

エンディアン: リトルエンディアン

クロック同期式シングルマスタ制御ソフトウェア: RSPI

データ転送モード: Software

対象ソース: r_eeprom_spi.c, r_eeprom_spi_sub.c, r_eeprom_spi_dev_port_iodefine.c, r_eeprom_spi_drvif.c

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		
		Renesas Compiler	GCC	IAR Compiler
RX111	ROM	**2,208 バイト**	**5,068 バイト**	**8,877 バイト**
	RAM	**4 バイト**	**4 バイト**	**37 バイト**
	スタック	**176 バイト**	**–**	**152 バイト**
RX113	ROM	**2,205 バイト**	**5,088 バイト**	**8,877 バイト**
	RAM	**4 バイト**	**4 バイト**	**37 バイト**
	スタック	**176 バイト**	**–**	**152 バイト**
RX231	ROM	**2,207 バイト**	**5,104 バイト**	**8,875 バイト**
	RAM	**4 バイト**	**4 バイト**	**37 バイト**

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		
		Renesas Compiler	GCC	IAR Compiler
	スタック	**176 バイト**	**–**	**152 バイト**
RX64M	ROM	**2,207 バイト**	**5,124 バイト**	**9,335 バイト**
	RAM	**4 バイト**	**4 バイト**	**37 バイト**
	スタック	**188 バイト**	**–**	**156 バイト**
RX71M	ROM	**2,207 バイト**	**5,104 バイト**	**9,331 バイト**
	RAM	**4 バイト**	**4 バイト**	**49 バイト**
	スタック	**188 バイト**	**–**	**156 バイト**

2.9 戻り値

API 関数の戻り値を示します。この列挙型は API 関数のプロトタイプ宣言とともに r_eeprom_spi_if.h で記載されています。

```
typedef enum e_eeprom_status
{
   EEPROM_SPI_SUCCESS_BUSY =  1,    /* Successful operation (EERPOM is busy) */
   EEPROM_SPI_SUCCESS      =  0,    /* Successful operation                  */
   EEPROM_SPI_ERR_PARAM    =  -1,   /* Parameter error                       */
   EEPROM_SPI_ERR_HARD     =  -2,   /* Hardware error                        */
   EEPROM_SPI_ERR_WP       =  -4,   /* Write-protection error                */
   EEPROM_SPI_ERR_OTHER    =  -7    /* Other error                           */
} eeprom_status_t;
```

2.10 ソフトウェアの追加方法

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、スマート・コンフィグレータを使用した(1)、(3)、(5)の追加方法を推奨しています。ただし、スマート・コンフィグレータは、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 上でスマート・コンフィグレータを使用して FIT モジュールを追加する場合
e2 studio のスマート・コンフィグレータを使用して、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 コンフィグレータを使用して FIT モジュールを追加する場合
e2 studio の FIT コンフィグレータを使用して、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+上でスマート・コンフィグレータを使用して FIT モジュールを追加する場合
CS+上で、スタンドアロン版スマート・コンフィグレータを使用して、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 上でスマート・コンフィグレータを使用して FIT モジュールを追加する場合
スタンドアロン版スマート・コンフィグレータを使用して、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。

2.11 FIT モジュール以外の環境で使用する場合の組み込み方法

r_bsp 等の FIT モジュールを使用しない環境下で動作させる場合、以下を実施してください。

#r_eeprom_spi_if.h の#include "platform.h"をコメントアウトしてください。

#r_eeprom_spi_if.h に以下のヘッダファイルをインクルードしてください。

```
#include “iodefine.h”
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <machine.h>
```

#r_eeprom_spi_if.h の「#define EEPROM_SPI_CFG_USE_FIT」を無効にしてください。

#r_eeprom_spi_if.h に「#define EEPROM_SPI_CFG_xxx（xxx は MCU 名、英字は大文字）」を定義してください。例えば RX64M であれば EEPROM_SPI_CFG_RX64M としてください。

r_eeprom_spi_if.h に以下の enum 定義を追加してください。また、以下の#define 定義を追加してください。「BSP_ICLK_HZ」にはシステムクロック（ICLK）の値を設定してください。なお、これらの定義は他の FIT モジュールの定義と重複する可能性があります。定義の先頭に「#ifndef SMSTR_WAIT」「#define SMSTR_WAIT」を記述し、最後に「#endif」を記述してください。

```
#ifndef SMSTR_WAIT
#define SMSTR_WAIT
typedef enum
{
    BSP_DELAY_MICROSECS = 1000000,
    BSP_DELAY_MILLISECS = 1000,
    BSP_DELAY_SECS = 1
} bsp_delay_units_t;

#define BSP_ICLK_HZ  (120000000)    /* ICLK=120MHz */
#endif      /* #ifndef SMSTR_WAIT */
```

2.12 端子の状態

Power on Reset 後、および API 関数実行後の端子の状態を表 2-1 に示します。

本モジュールは「1.5.2 SPI モードで制御」に示すとおり、SPI モード 3（CPOL=1、CPHA=1）をサポートします。ハードウェア構成によらず、**Power on Reset 後はユーザ側で GPIO 制御を行い、スレーブデバイスセレクト端子を H 出力状態にしてください。**

また、R_EEPROM_SPI_Close()後のスレーブデバイスセレクト端子の状態は GPIO H 出力です。必要に応じて端子設定を見直してください。

表 2-1　関数実行後の端子の状態

関数名	スレーブデバイスセレクト端子（注１）
（Power on Reset 後）	GPIO 入力状態
R_EEPROM_SPI_Open()前	GPIO H 出力状態 **ユーザ側で設定**
R_EEPROM_SPI_Open()後	GPIO H 出力状態 本モジュールで設定
R_EEPROM_SPI_Close()後	GPIO H 出力状態 本モジュールで設定

注１：スレーブデバイスセレクト端子は、外付け抵抗でプルアップ処理してください。「1.4.1 ハードウェア構成例」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

3.1 R_EEPROM_SPI_Open()

Serial EEPROM 制御ソフトウェアを使用する際に、最初に使用する関数です。

Format

```
eeprom_status_t R_EEPROM_SPI_Open(
uint8_t devno
)
```

Parameters

devno

デバイス番号（0, 1）

Return Values

EEPROM_SPI_SUCCESS　　*/* 正常終了した場合 */*

EEPROM_SPI_ERR_PARAM　　*/* パラメータ異常の場合 */*

Properties

r_eeprom_spi_if.h にプロトタイプ宣言されています。

Description

引数 devno で指定したデバイス番号のスレーブデバイスセレクト端子を初期化します。初期化後は、汎用出力ポート H 出力状態になります。
通信中は本関数をコールしないでください。通信中にコールした場合の通信は保証されません。

Example

```
eeprom_status_t   ret = EEPROM_SPI_SUCCESS;

ret = R_EEPROM_SPI_Open(EEPROM_SPI_DEV0);
```

Special Notes:

本ユーザ API コール後、R_EEPROM_SPI_Polling()により、EEPROM の書き込みサイクルの完了を確認することを推奨します。EEPROM は書き込みサイクル中、次の読み出しや書き込み処理を受け付けません。例えば、EEPROM の書き込みサイクル中にシステムリセットが発生し、EEPROM の制御を最初からやり直す場合、書き込みサイクル中の EEPROM にアクセスする可能性があります。

3.2 R_EEPROM_SPI_Close()

使用中の Serial EEPROM 制御ソフトウェアを終了する際に使用する関数です。

Format

```
eeprom_status_t R_EEPROM_SPI_Close(
uint8_t devno
)
```

Parameters

devno

デバイス番号（0, 1）

Return Values

EEPROM_SPI_SUCCESS */* 正常終了した場合 */*
EEPROM_SPI_ERR_PARAM */* パラメータ異常の場合 */*
EEPROM_SPI_ERR_OTHER */*他のエラーの場合*/*

Properties

r_eeprom_spi_if.h にプロトタイプ宣言されています。

Description

引数 devno で指定したデバイス番号のスレーブデバイスセレクト端子の機能を汎用入出力ポートにします。
実行後は、汎用出力ポート H 出力状態になります。
通信中は本関数をコールしないでください。通信中にコールした場合の通信は保証されません。

Example

```
eeprom_status_t   ret = EEPROM_SPI_SUCCESS;

ret = R_EEPROM_SPI_Close(EEPROM_SPI_DEV0);
```

Special Notes:

本関数コール後のスレーブデバイスセレクト端子はリセット後の状態（汎用入力ポート状態）とは異なります。必要に応じて、端子設定を見直してください。
本ユーザ API コール前に、R_EEPROM_SPI_Polling()により、EEPROM の書き込みサイクルの完了を確認することを推奨します。これにより、EEPROM が書き込みサイクルに遷移していない状態で、EEPROM の制御を再開することができます。

3.3 R_EEPROM_SPI_Read_Status()

ステータスレジスタを読み出す際に使用する関数です。

Format

```
eepr_status_t R_EEPROM_SPI_Read_Status
uint8_t devno,
uint8_t * p_status
)
```

Parameters

devno

デバイス番号（0, 1）

** p_status*

ステータスレジスタ格納バッファ（サイズ 1 バイト）

Return Values

EEPROM_SPI_SUCCESS	*/* 正常終了した場合 */*
EEPROM_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
EEPROM_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
EEPROM_SPI_ERR_OTHER	*/* 他タスクがシングル同期式シングルマスタ制御ソフトウェアのリソース取得済の場合、またはその他のエラーの場合 */*

Properties

r_eeprom_spi_if.h にプロトタイプ宣言されています。

Description

ステータスレジスタを読み出し、p_status に格納します。p_status には下記情報が格納されます。

＜4 Kbit 以下の場合＞

Bit 7 to 4:Reserved (All ”1”)	
Bit 3 to 2:BP1, BP0	00: No protection
	01: Upper-quarter protection
	10: Upper-half protection
	11: Whole memory protection
Bit 1:WEL	0: Write disabled
	1: Write enabled
Bit 0:WIP	1: During write operation

＜4 Kbit を超える場合＞

Bit 7:SRWD	0: Status register can be changed.
	1: Status register cannot be changed.
Bit 6 to 4:Reserved (All ”0”)	
Bit3 to 2:BP1, BP0	00: No protection
	01: Upper-quarter protection
	10: Upper-half protection
	11: Whole memory protection
Bit 1:WEL	0: Write disabled
	1: Write enabled
Bit 0:WIP	1: During write operation

Example

```
eeprom_status_t   ret = EEPROM_SPI_SUCCESS;
uint32_t          stat = 0;

ret = R_EEPROM_SPI_Read_Status(EEPROM_SPI_DEV0, &stat);
```

Special Notes:

処理の最初に下位層のクロック同期式シングルマスタ制御ソフトウェアのリソースを確保し、処理の最後にリソースを解放します。

3.4 R_EEPROM_SPI_Set_Write_Protect()

ライトプロテクトを設定する際に使用する関数です。

Format

```
eeprom_status_t R_EEPROM_SPI_Set_Write_Protect(
uint8_t devno,
uint8_t wpsts
)
```

Parameters

devno

デバイス番号（0, 1）

wpsts

ライトプロテクト設定データ

以下から 1 つ設定してください。

```
EEPROM_SPI_WP_NONE          /* No protection */
EEPROM_SPI_WP_UPPER_QUART   /* Upper-quarter protection setting */
EEPROM_SPI_WP_UPPER_HALF    /* Upper-half protection setting */
EEPROM_SPI_WP_WHOLE_MEM     /* Whole memory protection setting */
```

Return Values

```
EEPROM_SPI_SUCCESS      /* 正常終了した場合 */
EEPROM_SPI_ERR_PARAM    /* パラメータ異常の場合 */
EEPROM_SPI_ERR_HARD     /* ハードウェアエラーの場合 */
EEPROM_SPI_ERR_OTHER    /* 他タスクがシングル同期式シングルマスタ制御ソフトウェアのリソース取得済の場合、またはその他のエラーの場合 */
```

Properties

r_eeprom_spi_if.h にプロトタイプ宣言されています。

Description

ライトプロテクトを設定します。SRWD は、0 に設定されます。
本ユーザ API が正常終了した場合、EEPROM は書き込みサイクルに遷移しています。必ず、書き込み完了を R_EEPROM_SPI_Polling()で確認してください。書き込みサイクル中に次の読み出しや書き込み処理を行った場合、EEPROM はその処理を受け付けません。
R_EEPROM_SPI_Polling()はユーザの任意のタイミングでコールすることができます。そのため、書き込みサイクル中にユーザアプリケーションの他の処理を行うことができます。

詳しくは、図 3-1 を参照してください。

Example

```
  eeprom_status_t   ret   = EEPROM_SPI_SUCCESS;
  uint32_t          wait  = 0;
  uint32_t          loop_cnt = 0;

    Ret = R_EEPROM_SPI_Set_Write_Protect(EEPROM_SPI_DEV0, EEPROM_SPI_WP_NONE);
    if (EEPROM_SPI_SUCCESS > Ret)
    {
        /* Error */
    }

    loop_cnt = 50000;
    do
    {
        Ret = R_EEPROM_SPI_Polling(EEPROM_SPI_DEV0);
        if (EEPROM_SPI_SUCCESS == Ret)
        {
           /* EEPROM is ready. */
           break;
        }
        else if (EEPROM_SPI_SUCCESS_BUSY == Ret)
        {
           /* EEPROM is busy.
              User application can perform other processing while eeprom is busy.*/
           for (wait = 0; wait < 10; wait++)
           {
              /* Do nothing */
           }
        }
        else
        {
           /* Error */
        }
        loop_cnt--;
    }
    while (0 != loop_cnt);

    if (0 == loop_cnt)
    {
        /* Error */
    }
```

Special Notes:

処理の最初に下位層のクロック同期式シングルマスタ制御ソフトウェアのリソースを確保し、処理の最後にリソースを解放します。

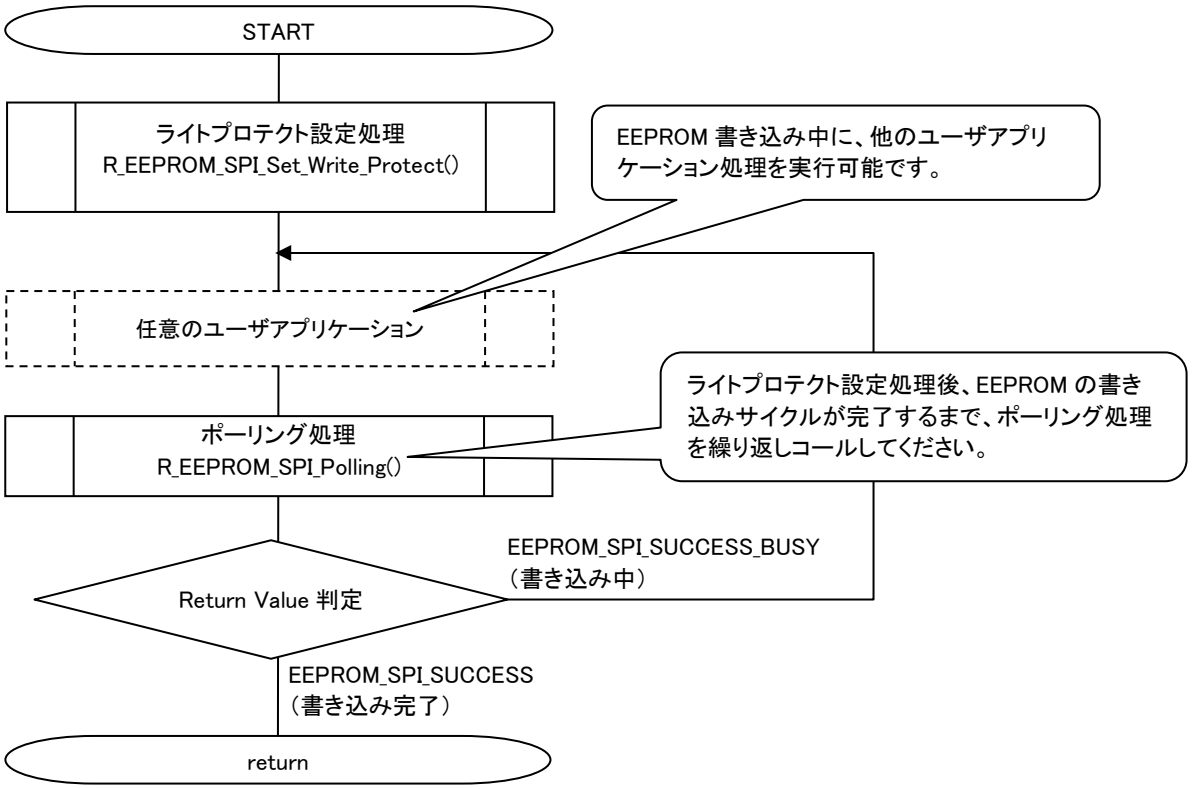


図 3-1　R_EEPROM_SPI_Set_Write_Protect()の処理例

3.5 R_EEPROM_SPI_Write_Di()

書き込みを禁止する際に使用する関数です。

Format

```
eeprom_status_t R_EEPROM_SPI_Write_Di(
uint8_t devno
)
```

Parameters

devno

デバイス番号（0, 1）

Return Values

EEPROM_SPI_SUCCESS	*/* 正常終了した場合 */*
EEPROM_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
EEPROM_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
EEPROM_SPI_ERR_OTHER	*/* 他タスクがシングル同期式シングルマスタ制御ソフトウェアのリソース取得済の場合、またはその他のエラーの場合 */*

Properties

r_eeprom_spi_if.h にプロトタイプ宣言されています。

Description

WRDI コマンドを送信し、ステータスレジスタの WEL ビットをクリアします。

Example

```
eeprom_status_t    ret = EEPROM_SPI_SUCCESS;

ret = R_EEPROM_SPI_Write_Di(EEPROM_SPI_DEV0);
```

Special Notes:

処理の最初に下位層のクロック同期式シングルマスタ制御ソフトウェアのリソースを確保し、処理の最後にリソースを解放します。

3.6 R_EEPROM_SPI_Read_Data()

EEPROM からデータを読み出す際に使用する関数です。

Format

```
eeprom_status_t R_EEPROM_SPI_Read_Data(
uint8_t devno,
eeprom_info_t * p_eeprom_info
)
```

Parameters

devno

デバイス番号（0, 1）

** p_eeprom_info*

EEPROM 情報構造体。構造体のアドレスは 4 バイトの境界値としてください。

addr

メモリの読み出し開始アドレスを設定してください。

cnt

読み出しバイト数を設定してください。設定可能範囲は 1～4,294,967,295 です。0 を設定した場合、エラーを返します。

data_cnt

読み出しバイト数（本制御ソフトウェアで使用するため、設定禁止）

*p_data

読み出しデータ格納バッファのアドレスを設定してください。バッファのアドレスは 4 バイトの境界値としてください。

Return Values

EEPROM_SPI_SUCCESS	*/* 正常終了した場合 */*
EEPROM_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
EEPROM_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
EEPROM_SPI_ERR_OTHER	*/* 他タスクがシングル同期式シングルマスタ制御ソフトウェアのリソース取得済の場合、またはその他のエラーの場合 */*

Properties

r_eeprom_spi_if.h にプロトタイプ宣言されています。

Description

EEPROM 上の指定アドレスから指定バイト数分、データを読み出し、p_data に格納します。
最大読み出しアドレスは、EEPROM 容量-1 です。
読み出しバイト数 cnt と指定アドレス addr の合計値が最大読み出しアドレスを超える場合、EEPROM_SPI_ERR_PARAM を返します。
DMAC 転送もしくは DTC 転送は、使用するクロック同期式シングルマスタ制御ソフトウェアの転送サイズ条件に合致した場合に行います。それ以外の場合、Software 転送に切り替わります。

Example

```
eeprom_status_t   ret = EEPROM_SPI_SUCCESS;
eeprom_info_t     Eep_Info_R;
uint32_t          buf2[EEPROM_SPI_DEV0_WPAG_SIZE/sizeof(uint32_t)];
                                /* the buffer boundary (4-byte unit) */

Eep_Info_R.addr   = 0;
Eep_Info_R.cnt    = 32;
Eep_Info_R.p_data = (uint8_t *)&buf2[0];
ret = R_EEPROM_SPI_Read_Data(EEPROM_SPI_DEV0, &Eep_Info_R);
```

Special Notes:

データ転送の高速化のために、データ格納バッファポインタを指定する場合、開始アドレスを 4 バイト境界に合わせてください。

処理の最初に下位層のクロック同期式シングルマスタ制御ソフトウェアのリソースを確保し、処理の最後にリソースを解放します。

3.7 R_EEPROM_SPI_Write_Data_Page()

EEPROM へ、1Page 単位でデータを書き込む際に使用する関数です。

Format

```
eeprom_status_t R_EEPROM_SPI_Write_Data_Page(
uint8_t devno,
eeprom_info_t * p_eeprom_info
)
```

Parameters

devno

デバイス番号（0, 1）

* *p_eeprom_info*

EEPROM 情報構造体。構造体のアドレスは 4 バイトの境界値としてください。

addr

メモリの書き込み開始アドレスを設定してください。

cnt

書き込みバイト数を設定してください。設定可能範囲は 1～4,294,967,295 です。0 を設定した場合、エラーを返します。

data_cnt

書き込みバイト数（本制御ソフトウェアで使用するため、設定禁止）

*p_data

書き込みデータ格納バッファのアドレスを設定してください。

Return Values

EEPROM_SPI_SUCCESS	*/* 正常終了した場合 */*
EEPROM_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
EEPROM_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
EEPROM_SPI_ERR_WP	*/* ライトプロテクトエラーの場合 */*
EEPROM_SPI_ERR_OTHER	*/* 他タスクがシングル同期式シングルマスタ制御ソフトウェアのリソース取得済の場合、またはその他のエラーの場合 */*

Properties

r_eeprom_spi_if.h にプロトタイプ宣言されています。

Description

p_data のデータを EEPROM 上の指定アドレスから指定バイト数分（最大 1 Page サイズ）書き込みます。
大容量のデータ書き込みの際、Page 単位に通信を分割するため、通信中に他の処理ができなくなることを防ぐことができます。
EEPROM への書き込みは、ライトプロテクト解除状態の場合のみ可能です。
プロテクトされたページへの書き込みはできません。エラーEEPROM_SPI_ERR_WP を返します。
最大書き込みアドレスは、EEPROM 容量-1 です。
書き込みバイト数 cnt と指定アドレス addr の合計値が最大書き込みアドレスを超える場合、エラー EEPROM_SPI_ERR_PARAM を返します。
DMAC 転送もしくは DTC 転送は、使用するクロック同期式シングルマスタ制御ソフトウェアの転送サイズ条件に合致した場合に行います。それ以外の場合、Software 転送に切り替わります。

1Page を超えるバイト数が設定されている場合でも、1Page 書き込み処理完了後、残バイト数と次アドレス情報が EEPROM 情報構造体（p_eeprom_info）に残ります。未変更のまま再びその p_eeprom_info を本ユーザ API にセットすることで残バイト数の書き込みが可能です。
本ユーザ API が正常終了した場合、EEPROM は書き込みサイクルに遷移しています。必ず、書き込み完了を R_EEPROM_SPI_Polling()で確認してください。書き込みサイクル中に次の読み出しや書き込み処理を行った場合、EEPROM はその処理を受け付けません。
R_EEPROM_SPI_Polling()はユーザの任意のタイミングでコールすることができます。そのため、書き込みサイクル中にユーザアプリケーションの他の処理を行うことができます。
詳しくは、図 3-2 を参照してください。

Example

```
 eeprom_status_t   ret = EEPROM_SPI_SUCCESS;
 eeprom_info_t     Eep_Info_W;
 uint32_t          buf1[EEPROM_SPI_DEV0_WPAG_SIZE/sizeof(uint32_t)];
                                  /* the buffer boundary (4-byte unit) */
 uint32_t          wait  = 0;
 uint32_t          loop_cnt = 0;

 Eep_Info_W.addr   = 0;
 Eep_Info_W.cnt    = 128;
 Eep_Info_W.p_data = (uint8_t *)&buf1[0];

  do
  {
      Ret = R_EEPROM_SPI_Write_Data_Page(EEPROM_SPI_DEV0, &Eep_Info_W);
      if (EEPROM_SPI_SUCCESS > Ret)
      {
         /* Error */
      }
      loop_cnt = 50000;
      do
      {
         Ret = R_EEPROM_SPI_Polling(EEPROM_SPI_DEV0);
         if (EEPROM_SPI_SUCCESS == Ret)
         {
            /* EEPROM is ready. */
            break;
         }
         else if (EEPROM_SPI_SUCCESS_BUSY == Ret)
         {
         /* EEPROM is busy.
            User application can perform other processing while eeprom is busy.*/
            for (wait = 0; wait < 10; wait++)
            {
               /* Do nothing */
            }
         }
         else
         {
            /* Error */
         }
         loop_cnt--;
      }
      while (0 != loop_cnt);
  }
  while (0 != Eep_Info_W.cnt);

  if (0 == loop_cnt)
  {
      /* Error */
  }
```

Special Notes:

データ転送の高速化のために、データ格納バッファポインタを指定する場合、開始アドレスを 4 バイト境界に合わてください。

処理の最初に下位層のクロック同期式シングルマスタ制御ソフトウェアのリソースを確保し、処理の最後にリソースを解放します。

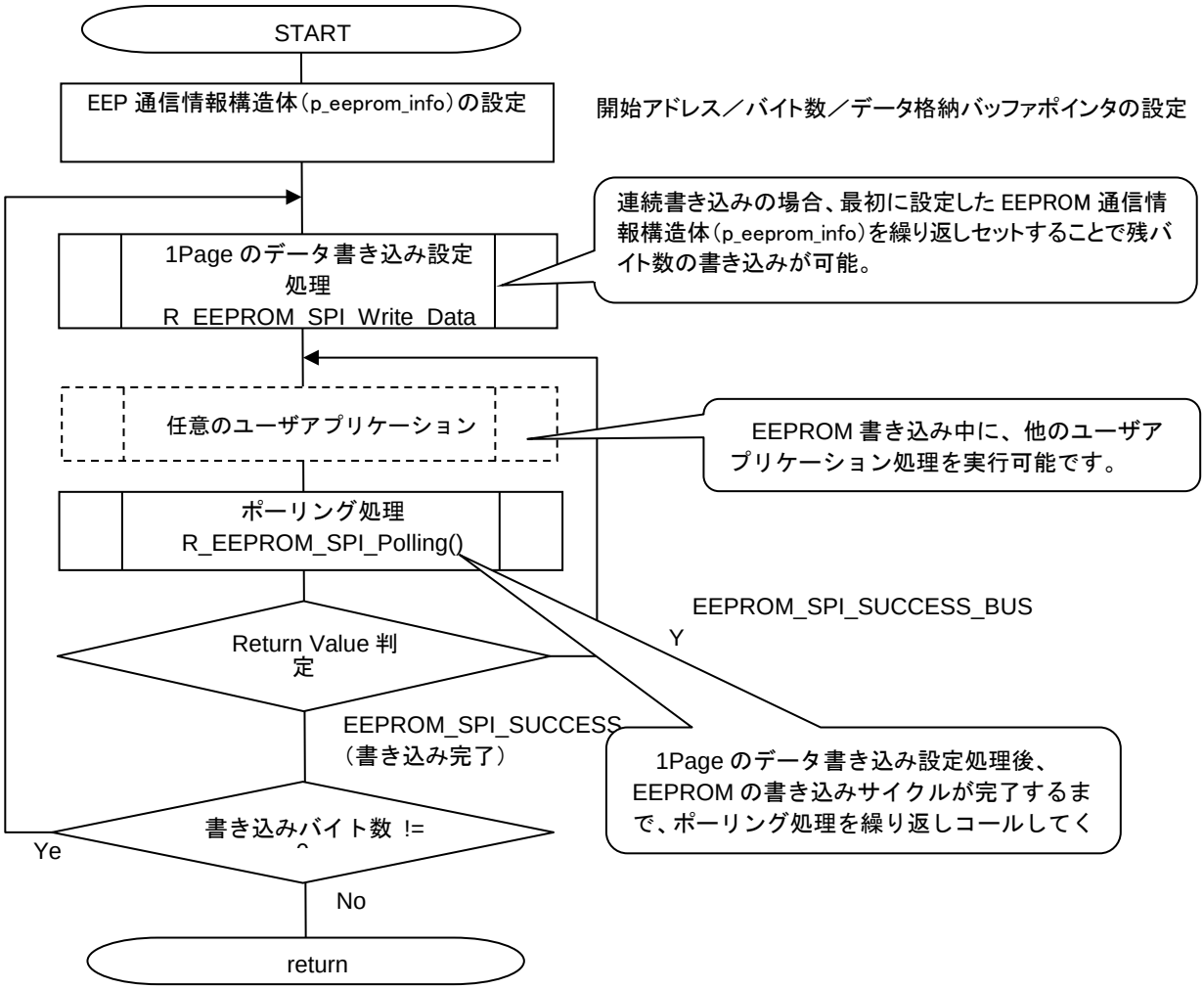


図 3-2　R_EEPROM_SPI_Write_Data_Page()の処理例

3.8 R_EEPROM_SPI_Polling()

書き込み完了ポーリングする際に使用する関数です。

Format

```
eeprom_status_t R_EEPROM_SPI_Polling(
uint8_t devno
)
```

Parameters

devno

デバイス番号（0, 1）

Return Values

EEPROM_SPI_SUCCESS	*/* 正常終了、かつ、書き込み完了の場合 */*
EEPROM_SPI_SUCCESS_BUSY	*/* 正常終了、かつ、書き込み中の場合 */*
EEPROM_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
EEPROM_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
EEPROM_SPI_ERR_OTHER	*/* 他タスクがシングル同期式シングルマスタ制御ソフトウェアのリソース取得済の場合、またはその他のエラーの場合 */*

Properties

r_eeprom_spi_if.h にプロトタイプ宣言されています。

Description

書き込みサイクルの完了判定を行います。

Example

図 3-1 もしくは図 3-2 をご参照ください。

Special Notes:

R_EEPROM_SPI_Polling()はユーザの任意のタイミングでコールすることができます。そのため、書き込みサイクル中にユーザアプリケーションの他の処理を行うことができます。
処理の最初に下位層のクロック同期式シングルマスタ制御ソフトウェアのリソースを確保し、処理の最後にリソースを解放します。

3.9 R_EEPROM_SPI_GetMemoryInfo()

Serial EEPROM のサイズ情報を取得する際に使用する関数です。

Format

```
eeprom_status_t R_EEPROM_SPI_GetMemoryInfo(
uint8_t devno,
eeprom_mem_info_t * p_eeprom_mem_info
)
```

Parameters

devno

デバイス番号（0, 1）

* *p_eeprom_mem_info*

EEPROM サイズ情報構造体。構造体のアドレスは 4 バイトの境界値としてください。

mem_size

最大メモリサイズ

wpag_size

ページサイズ

Return Values

EEPROM_SPI_SUCCESS　　/ 正常終了した場合 */*
EEPROM_SPI_ERR_PARAM　　/ パラメータ異常の場合 */*

Properties

r_eeprom_spi_if.h にプロトタイプ宣言されています。

Description

引数 devno で指定したデバイス番号の EEPROM サイズ情報を取得します。

Example

```
eeprom_status_t    ret = EEPROM_SPI_SUCCESS;
eeprom_mem_info_t  Eep_MemInfo;

ret = R_EEPROM_SPI_GetMemoryInfo(EEPROM_SPI_DEV0, &Eep_MemInfo);
```

Special Notes:

なし

3.10 R_EEPROM_SPI_GetVersion()

Serial EEPROM 制御ソフトウェアのバージョン情報を取得する際に使用する関数です。

Format

```
uint32_t R_EEPROM_SPI_GetVersion(void)
```

Parameters

なし

Return Values

バージョン番号　*上位 2 バイト：メジャーバージョン、下位 2 バイト：マイナーバージョン*

Properties

r_eeprom_spi_if.h にプロトタイプ宣言されています。

Description

バージョン情報を返します。

Example

```
uint32_t version;
version = R_EEPROM_SPI_GetVersion();
```

Special Notes:

なし

3.11 R_EEPROM_SPI_Set_LogHdlAddress()

LONGQ FIT モジュールのハンドラアドレスを設定する関数です。エラーログ取得処理を使用する場合、コールしてください。

Format

```
eeprom_status_t R_EEPROM_SPI_Set_LogHdlAddress(
uint32_t user_long_que
)
```

Parameters

user_long_que

LONGQ FIT モジュールのハンドラアドレスを設定してください。

Return Values

EEPROM_SPI_SUCCESS　　/ 正常終了した場合 */*

Properties

r_eeprom_spi_if.h にプロトタイプ宣言されています。

Description

LONGQ FIT モジュールのハンドラアドレスを Serial EEPROM 制御ソフトウェアと指定デバイスで使用するクロック同期式シングルマスタ制御ソフトウェアに設定します。
LONGQ FIT モジュールを使用し、エラーログを取得するための準備処理です。R_EEPROM_SPI_Open()をコールする前に処理を実行してください。

Example

```
#define ERR_LOG_SIZE (16)
#define USER_LONGQ_IGN_OVERFLOW   (1)

eeprom_status_t ret = EEPROM_SPI_SUCCESS;
uint32_t          MtlLogTbl[ERR_LOG_SIZE];
longq_err_t       ret_longq = LONGQ_SUCCESS;
longq_hdl_t       p_user_long_que;
uint32_t          long_que_hndl_address = 0;

/* Open LONGQ module. */
ret_longq = R_LONGQ_Open(&MtlLogTbl[0],
                         ERR_LOG_SIZE,
                         USER_LONGQ_IGN_OVERFLOW,
                         &p_user_long_que
);

long_que_hndl_address = (uint32_t)p_user_long_que;
R_EEPROM_SPI_Set_LogHdlAddress(long_que_hndl_address);
```

Special Notes:
別途 LONGQ FIT モジュールを組み込んでください。
r_eeprom_spi_config.h の#define EEPROM_SPI_CFG_LONGQ_ENABLE を有効にしてください。また、指定デバイスで使用するクロック同期式シングルマスタ制御ソフトウェアの#define xxx_LONGQ_ENABLE を有効にしてください。
LONGQ FIT モジュールの R_LONGQ_Open()の引数 ignore_overflow を“1”に設定してください。これによりエラーログバッファは、リングバッファとして使用することが可能です。

3.12 R_EEPROM_SPI_Log()

エラーログを取得する関数です。エラー発生時、ユーザ処理を終了する直前にコールしてください。

Format

```
uint32_t R_EEPROM_SPI_Log(
uint32_t flg,
uint32_t fid,
uint32_t line
)
```

Parameters

flg

0x00000001（固定値）を設定してください。

fid

0x0000003f（固定値）を設定してください。

line

0x0001ffff（固定値）を設定してください。

Return Values

```
0                       /* 正常終了した場合 */
1                       /* 異常終了した場合 */
```

Properties

r_eeprom_spi_if.h にプロトタイプ宣言されています。

Description

エラーログを取得する関数です。エラー発生時、ユーザ処理を終了する直前にコールしてください。

Example

```
#define  USER_DRIVER_ID    (0x00000001)
#define  USER_LOG_MAX      (0x0000003f)
#define  USER_LOG_ADR_MAX  (0x00001fff)

eeprom_status_t   ret = EEPROM_SPI_SUCCESS;
eeprom_info_t     Eep_Info_W;
uint32_t          buf1[EEPROM_SPI_DEV0_WPAG_SIZE/sizeof(uint32_t)];
                                /* the buffer boundary (4-byte unit) */

Eep_Info_W.addr   = 0;
Eep_Info_W.cnt    = 32;
Eep_Info_W.p_data = (uint8_t *)&buf1[0];
ret = R_EEPROM_SPI_Write_Data_Page(EEPROM_SPI_DEV0, &Eep_Info_W);
if (EEPROM_SPI_SUCCESS != ret)
{
   /* Set last error log to buffer. */
   R_EEPROM_SPI_Log(USER_DRIVER_ID, USER_LOG_MAX, USER_LOG_ADR_MAX);
   R_EEPROM_SPI_Close(EEPROM_SPI_DEV0);
}
```

Special Notes:

別途 LONGQ FIT モジュールを組み込んでください。
r_eeprom_spi_config.h の#define EEPROM_SPI_CFG_LONGQ_ENABLE を有効にしてください。また、指定デバイスで使用するクロック同期式シングルマスタ制御ソフトウェアの#define xxx_LONGQ_ENABLE を有効にしてください。

3.13 R_EEPROM_SPI_1ms_Interval()

クロック同期式シングルマスタ制御ソフトウェアのインターバルタイマカウンタ関数をコールする関数です。DMAC もしくは DTC を使用する場合、タイマを使用して 1ms 毎に本関数をコールしてください。

Format
```
void R_EEPROM_SPI_1ms_Interval(void)
```

Parameters
なし

Return Values
なし

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

Description
DMAC もしくは DTC 転送完了待ち時にクロック同期式シングルマスタ制御ソフトウェアの内部タイマカウンタをインクリメントします。

Example
```
void cmt_callback (void * pdata)
{
    uint32_t channel;

    channel = (uint32_t)pdata;

    if (channel == gs_cmt_channel)
    {
        R_EEPROM_SPI_1ms_Interval();
    }
}
```

Special Notes:
タイマ等を使用して本関数を 1ms 毎にコールしてください。
上記 Example は、1ms 毎に発生するコールバック関数で本関数をコールする例です。

4. 付録

4.1 動作確認環境

本 FIT モジュールの動作確認環境を以下に示します。

表 4.1 動作確認環境 (Rev.3.00)

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 V7.3.0
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 V3.01.00 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
モジュールのリビジョン	Rev.3.00
使用ボード	Renesas Starter Kit for RX111（型名：R0K505111xxxxxx） Renesas Starter Kit for RX113（型名：R0K505113xxxxxx） Renesas Starter Kit for RX231（型名：R0K505231xxxxxx） Renesas Starter Kit+ for RX64M（型名：R0K50564Mxxxxxx） Renesas Starter Kit+ for RX71M（型名：R0K50571Mxxxxxx）

表 4.2 動作確認環境 (Rev.3.01)

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 V7.3.0 IAR Embedded Workbench for Renesas RX 4.10.01
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.01.00 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
	GCC for Renesas RX 4.08.04.201803 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 -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.10.01 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	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.3.01
使用ボード	Renesas Starter Kit+ for RX65N（型名：RTK500565Nxxxxxx）

表 4.3 動作確認環境 (Rev.3.02)

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 2020-07 IAR Embedded Workbench for Renesas RX 4.14.01
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.02.00 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
	GCC for Renesas RX 8.03.00.202002 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 -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.14.01 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	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.3.02
使用ボード	Renesas Starter Kit+ for RX72N（型名：RTK5572Nxxxxxxxxxx）

表 4.4　動作確認環境 (Rev.3.10)

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 2022-04 IAR Embedded Workbench for Renesas RX 4.20.03
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.04.00 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
	GCC for Renesas RX 8.03.00.202104 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 -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.20.03 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	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.3.10
使用ボード	Renesas Starter Kit+ for RX65N（型名：RTK500565Nxxxxxx）

5. 参考ドキュメント

データシート
（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。）
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

本モジュールに該当するテクニカルアップデートはありません。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
2.30	2014.11.28	—	初版発行
2.31	2014.12.26	1	対象デバイス　RX64M グループ　に、QSPI と SCIF を追加。
		1	対象デバイス　に、RX71M を追加。
		1	FIT 関連ドキュメント　に、「CS+に組み込む方法(R01AN1826JJ)」を追加。
		2	1. 概要　の組み合わせ対象シリアル通信 FIT モジュール　に、SCIFA FIT モジュールを追加。
		5	1.2.2 動作環境とメモリサイズ　(1)RX111 の場合　表 1-3 動作確認条件 使用ボード　型名　元は、Renesas Starter Kit for RX111 であった。
		6	1.2.2 動作環境とメモリサイズ　(2)RX113 の場合　表 1-3 動作確認条件 使用ボード　型名　元は、Renesas Starter Kit for RX113 であった。
		7	1.2.2 動作環境とメモリサイズ　(3)RX64M の場合　表 1-3 動作確認条件 使用ボード　型名　元は、Renesas Starter Kit for RX64M であった。
		8	1.2.2 動作環境とメモリサイズ　(4)RX71M の場合　を追加。
		9	1.3 関連アプリケーションノート　に、SCIF クロック同期式シングルマスタ制御モジュール（R01AN2280JJ）を追加。
		16	2.2. ソフトウェアの要求　にて、 「r_rspi_smstr_rx」　元は、「r_rspi_smster_rx」であった。 「r_qspi_smstr_rx」　元は、「r_qspi_smster_rx」であった。
		16	2.2. ソフトウェアの要求　に、r_scifa_smstr_rx を追加。
		20	2.9.1 Serial EEPROM 制御ソフトウェアの追加手順（手動で追加する場合）　8　にて、r_eeprom_spi_pin_config_reference.h　元は、r_eeprom_spi_config_pin_reference.h であった。
2.32	2015.06.15	1	対象デバイス　に、RX230, RX231 を追加。
		5-10	1.2.2 動作環境とメモリサイズ　表 1-4, 表 1-6, 表 1-8, 表 1-10, 表 1-12 備考　ソース改定に伴い、r_eeprom_spi_target_dev_port.c を r_eeprom_spi_dev_port.c に変更。
		9	1.2.2 動作環境とメモリサイズ　(5)RX231 の場合　を追加。
		10	1.3.1 FIT モジュール関連のアプリケーションノート　に、RX ファミリ Serial Flash memory アクセス クロック同期式制御モジュール Firmware Integration Technology（R01AN2662JJ）を追加。
		10	1.3.1 FIT モジュール関連のアプリケーションノート RX64M グループ RX Driver Package ユーザーズマニュル（R01AN2144JJ）を削除
		10	1.3.1 FIT モジュール関連のアプリケーションノート DTC モジュール 和文版に更新
		12	1.5.3 Serial EEPROM の Chip select 端子制御　元は、「Chip select」は「S#」であった。また、「Clock」は「C」であった。
		13	1.5.4 ソフトウェア構成　元は、「Port Dev レイヤ」は「ターゲット MCU」であった。
		18	2.6 コンパイル時の設定　表 Configuration options in r_eeprom_spi_config.h に#define EEPROM_SPI_CFG_DEVx_DRVIF_CH_NO を追加

Rev.	発行日	改訂内容	
		ページ	ポイント
2.32	2015.06.15	32	3.6 R_EEPROM_SPI_Read_Data() Parameters　元は構造体のアドレスに対する記載が無かった。
		37	3.7 R_EEPROM_SPI_Write_Data_Page()　元は構造体のアドレスに対する記載が無かった。
		43	3.9 R_EEPROM_SPI_GetMemoryInfo()　元は構造体のアドレスに対する記載が無かった。
2.33	2015.12.29	1	対象デバイス　の「動作確認に使用したデバイス」を削除 対象デバイス　に、「RX ファミリ MCU」を追加 動作確認に使用した MCU　に、「RX130」,「RX23T」,「RX24T」を追加。
		21	2.9 ソフトウェアの追加方法　を更新した。
2.34	2017.07.31	17	2.2 ソフトウェアの要求　r_cgc_rx を削除した。
3.00	2019.02.20	1	対象デバイス RX72T グループの追加 FIT 関連ドキュメント タイトル「R01AN1685JJ」直前に RX ファミリの文字を追加 タイトル「R01AN1723JU」直前に RX ファミリの文字を追加 タイトル「R01AN1826JJ」直前に RX ファミリの文字を追加。
		3	概要:SCIFA から SCI に変更
		5-9	1.2.2 動作環境とメモリサイズを更新。
		10	1.3.1 FIT モジュール関連のアプリケーションノート 以下タイトルの文字を変更 R01AN2063JJ：モジュール名前は DMAC に変更 R01AN1819JJ：RX ファミリに RX Family を変更 R01AN1856JJ：モジュール名前は CMT に変更 R01AN1721EU：モジュール名前は GPIO に変更 R01AN1724EU：モジュール名前は MPC に変更。 以下関連のアプリケーションノートを削除: R01AN1914JJ、R01AN2280JJ。 以下関連のアプリケーションノートを追加: R01AN1827JJ、R01AN1815JJ、R01AN4548JJ。
		13	1.5.3 ソフトウェア構成 図 1-4 を更新。
		14	1.5.4 本制御ソフトウェアとクロック同期式シングルマスタ制御ソフトウェアの関係 図 1-5 を更新。
		17	2.2 ソフトウェアの要求 r_memdrv_rx を追加 r_rspi_smstr_rx が r_rspi_rx に変更 r_scifa_smstr _rx か r_sci_rx に変更。

Rev.	発行日	改訂内容	
		ページ	ポイント
		18	2.6 コンパイル時の設定 下記チャネル関連マクロを削除 EEPROM_SPI_CFG_DEVx_DRVIF_CH_NO EEPROM_SPI_CFG_DEVx_MODE EEPROM_SPI_CFG_DEVx_DMAC_CH_NO_Tx EEPROM_SPI_CFG_DEVx_DMAC_CH_NO_Rx EEPROM_SPI_CFG_DEVx_DMAC_INIT_PRIORITY_LEVEL_Tx EEPROM_SPI_CFG_DEVx_DMAC_INIT_PRIORITY_LEVEL_Rx EEPROM_SPI_CFG_DEVx_BR EEPROM_SPI_CFG_DEVx_BR_WRITE_DATA EEPROM_SPI_CFG_DEVx_BR_READ_DATA。
3.00	2019.02.20	20	2.9 ソフトウェアの追加方法　を更新した。
3.01	2019.05.20	—	以下のコンパイラに対応 ・GCC for Renesas RX ・IAR C/C++ Compiler for Renesas RX
		1	「関連ドキュメント」を削除
		1	「対象コンパイラ」を追加
		3	IAR コンパイラの制限事項を追加
		4	1.2 API の概要とメモリサイズ　を 1.2 API の概要に修正 1.2.2 動作環境とメモリサイズ　を削除
		12	2.2 ソフトウェアの要求　依存する r_bsp モジュールのリビジョンを追加
		15	2.8 コードサイズ　を追加
		20	2.13　for 文、while 文、do while 文について を追加
		43	4.1 動作確認環境　を追加
3.02	2020.12.10	1	動作確認に使用した MCU タイトル RX72N グループを追加。
		15	2.8 コードサイズ　を更新
		18	2.10 ソフトウェアの追加方法　を更新
		22-43	API 説明ページの「Reentrant」項目を削除
		44	4.1 動作確認環境　を更新
		45	5 参考ドキュメント　を更新

Rev.	発行日	改訂内容	
		ページ	ポイント
		プログラム	ソフトウェア不具合のため、EEPROM_SPI FIT モジュールを修正 ■内容 GPIO モジュール Firmware Integration Technology および MPC モジュール Firmware Integration Technology 併用してビルド時に警告およびリンクエラーが発生する。 ■発生条件 1.統合開発環境 CS+を使用する。 2.EEPROM FIT モジュールの汎用入出力ポートの制御を以下の FIT モジュールの両方で行う。 ・GPIO モジュール Firmware Integration Technology ・MPC モジュール Firmware Integration Technology ■対策 EEPROM_SPI FIT モジュール Rev3.02 をご使用ください。 対応ツールニュース番号：R20TS0609 ソフトウェア不具合のため、EEPROM_SPI FIT モジュールを修正 ■内容 RX72M/RX72N/RX66N で、r_eeprom_spi_pin_config.h のデバイスポートを”H”、”K”、”M”、”N”、”Q”のいずれかに設定すると、ビルドエラーが発生する。 ■発生条件 EEPROM_SPI_CS_DEV0_CFG_PORTNO 若しくは EEPROM_SPI_CS_DEV1_CFG_PORTNO を”H”、”K”、”M”、”N”、”Q”のいずれかに設定してビルドする。 ■対策 EEEPROM_SPI FIT モジュール Rev3.02 をご使用ください。
3.10	2022.06.30	13	2.6 コンパイル時の設定 新規マクロを追加 #define EEPROM_SPI_CS_DEVx_CFG_PORTNO #define EEPROM_SPI_CS_DEVx_CFG_BITNO
		16	2.8 FIT モジュールのバージョンおよびコンパイラのバージョンを更新
		47	「4.1 動作確認環境」： Rev.3.10 に対応する表を追加。
		プログラム	SS#端子として使用される汎用ポートを指定するための新規マクロを追加 if 文の誤った条件式の問題を修正しました。 SS#に割り当てられたデフォルトのポートを PORTX に設定します。

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。