RX ファミリ
Serial NOR Flash memory アクセス クロック同期制御モジュール

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 Serial Flash memory 制御方法とその使用方法を説明します。対象の Serial Flash memory は下記対象デバイスを参照してください。

なお、本制御ソフトウェアは、スレーブデバイスとして Serial Flash memory 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。

- Serial Flash memory 制御
  http://japan.renesas.com/driver/spi_serial_flash

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 Serial Flash memory 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 NOR Flash memory

Macronix International Co., Ltd 社製

- MX25/66L family serial NOR Flash memory 32Mbit - 1Gbit
- MX25R family serial NOR Flash memory 32Mbit - 64Gbit

Dialog Semiconductor Plc 社製

- AT25QF family serial NOR Flash memory 64Mbit.

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）
RX671 グループ（QSPIX, RSCI）

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。

なお、本アプリケーションノートでは、以下の略称を使用します。

- Single-SPI（Single SPI mode による通信）
- Dual-SPI（Dual SPI mode による通信）
- Quad SPI（Quad SPI mode による通信）

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「5.1 動作確認環境」を参照してください。

FIT 関連ドキュメント

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

目次

1. 概要

ルネサス エレクトロニクス製 MCU を使用し、Serial Flash memory 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 Serial Flash memory 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

  ・QSPIX FITモジュール

  ・RSCI FITモジュール
- 最大 2 個の Serial Flash memory 制御が可能
- デバイス毎に Serial Flash memory 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

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）

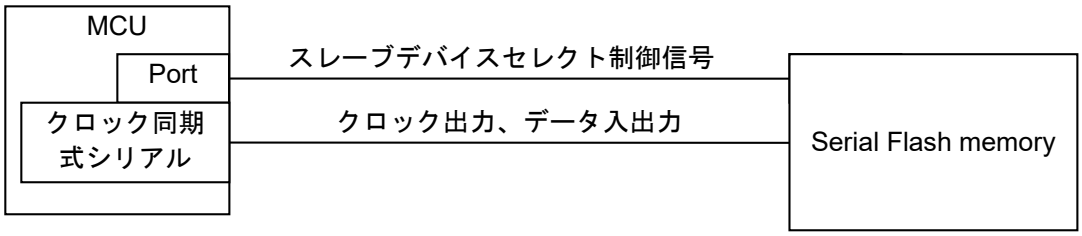


図 1-1　使用例

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

Serial Flash memory 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 Flash memory 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 Flash memory 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 Flash memory 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_FLASH_SPI_Open()	本制御ソフトウェアの初期化処理
R_FLASH_SPI_Close()	本制御ソフトウェアの終了処理
R_FLASH_SPI_Read_Status()	ステータスレジスタ読み出し処理
R_FLASH_SPI_Read_Status2()	ステータスレジスタ 2 読み出し処理
R_FLASH_SPI_Read_Status3()	ステータスレジスタ 3 読み出し処理
R_FLASH_SPI_Set_Write_Protect()	ライトプロテクト設定処理
R_FLASH_SPI_Write_Di()	WRDI コマンド処理
R_FLASH_SPI_Read_Data()　注 1	データ読み出し処理
R_FLASH_SPI_Write_Data_Page()　注 1	データ書き込み（1Page 書き込み用）処理
R_FLASH_SPI_Erase()	消去処理
R_FLASH_SPI_Polling()	ポーリング処理
R_FLASH_SPI_Read_ID()	ID 読み出し処理
R_FLASH_SPI_GetMemoryInfo()	メモリサイズ取得処理
R_FLASH_SPI_Read_Configuration()	コンフィグレーションレジスタ読み出し処理
R_FLASH_SPI_Write_Configuration()	コンフィグレーションレジスタ書き込み処理
R_FLASH_SPI_Write_Status()	ステータスレジスタ 1 書き込み処理
R_FLASH_SPI_Write_Status2()	ステータスレジスタ 2 書き込み処理
R_FLASH_SPI_Write_Status3()	ステータスレジスタ 3 書き込み処理
R_FLASH_SPI_Set_4byte_Address_Mode()	4 バイトアドレスモード設定処理
R_FLASH_SPI_Read_Security()	セキュリティレジスタ読み出し処理
R_FLASH_SPI_Read_Data_Security_Page()	データセキュリティページ読み出し処理
R_FLASH_SPI_Write_Data_Security_Page()	データセキュリティページ書き込み処理
R_FLASH_SPI_Quad_Enable()	Quad モード許可設定処理
R_FLASH_SPI_Quad_Disable()	Quad モード禁止設定処理
R_FLASH_SPI_GetVersion()	本制御ソフトウェアのバージョン情報取得処理
R_FLASH_SPI_Set_LogHdlAddress()	LONGQ FIT モジュールのハンドラアドレス設定処理
R_FLASH_SPI_Log()	LONGQ FIT モジュールを使ったエラーログ取得処理
R_FLASH_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 Flash memory 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 モジュール 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 アプリケーションノート SCI モジュール Firmware Integration Technology（R01AN1815）
- RX ファミリ QSPIX モジュール Firmware Integration Technology（R01AN5685）
- RX ファミリ RSCI モジュール Firmware Integration Technology（R01AN5759）
- 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 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 EEPROM アクセス クロック同期式制御モジュール Firmware Integration Technology（R01AN2325）
- RX ファミリアプリケーションノート 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　Serial Flash Memory モジュールの使用方法

1.4.1　Serial Flash Memory 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、Serial Flash Memory 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_flash_spi_if.h”
}
```

1.5　ハードウェア設定

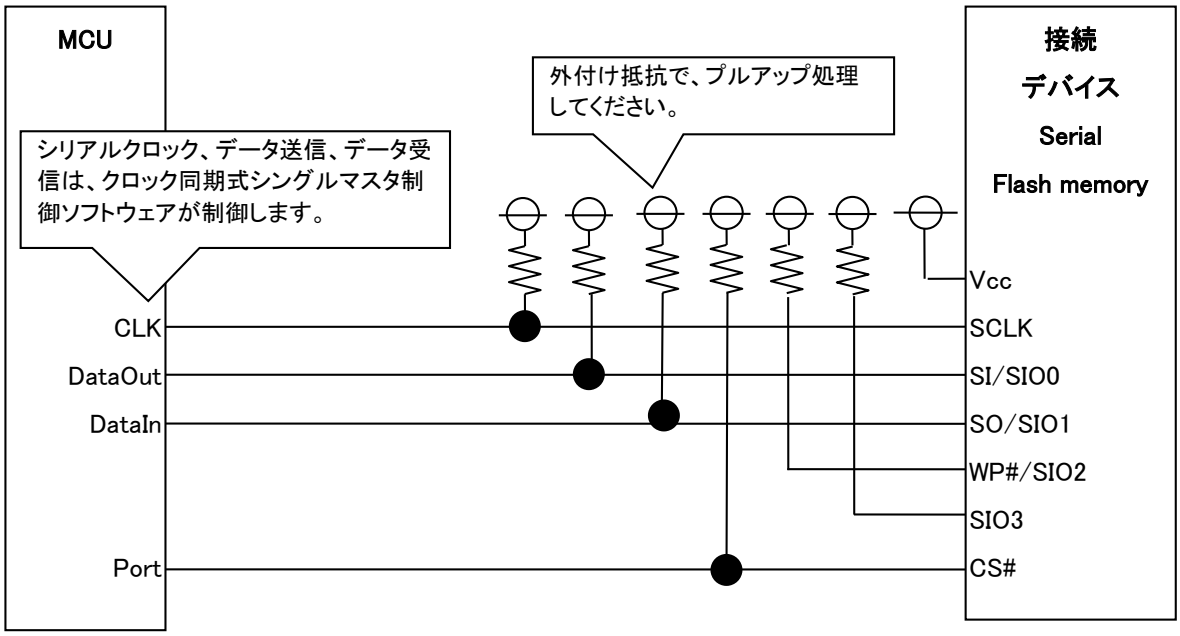
1.5.1　ハードウェア構成例

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。

1.5.1.1　Single-SPI 構成例

以下に Single-SPI 使用時の接続例を示します。



・MCU 上のシリアル I/O に使用される端子名は、MCU に依存します。
・WP#と RESET#を使用しない場合の例です。WP#と RESET#を使用する場合は接続デバイスの仕様書を確認してください。

図 1-2　Single-SPI 使用時の MCU と SPI スレーブデバイスの接続例

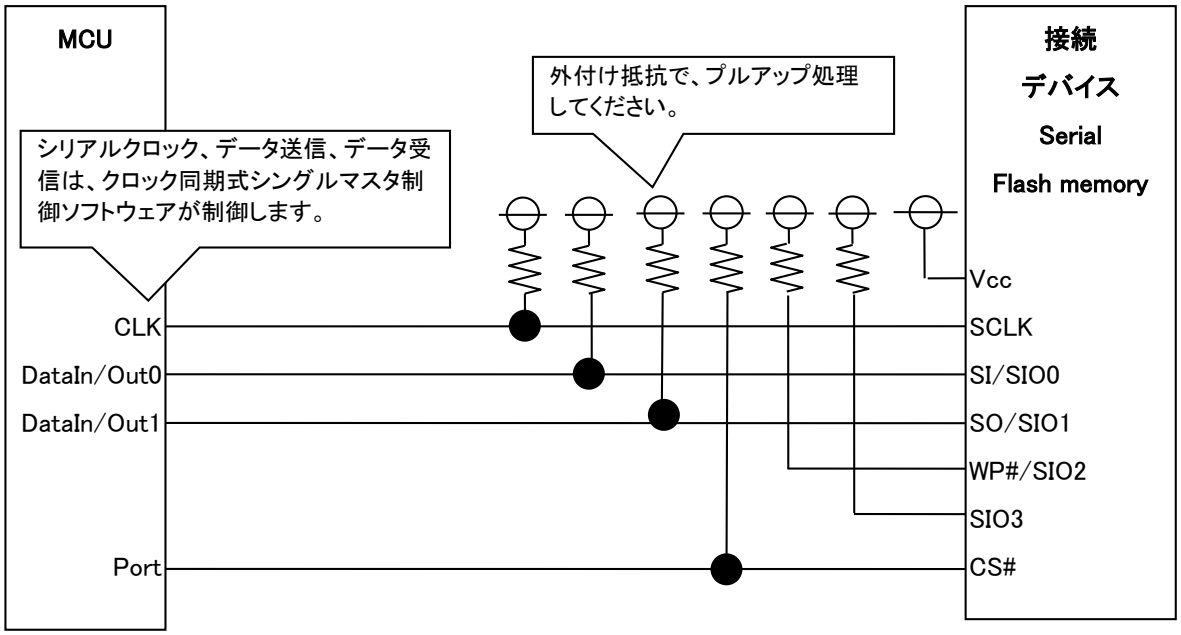
表 1-3　Single-SPI 使用端子と機能

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

1.5.1.2　Dual-SPI 構成例

以下に Dual-SPI 使用時の接続例を示します。

なお、Dual-SPI を使用するためには、対象 MCU にクワッドシリアルペリフェラルインターフェース機能が搭載されている必要があります。



・MCU 上のシリアル I/O に使用される端子名は、MCU に依存します。
・WP#と RESET#を使用しない場合の例です。WP#と RESET#を使用する場合は接続デバイスの仕様書を確認してください。

図 1-3　Dual-SPI 使用時の MCU と SPI スレーブデバイスの接続例

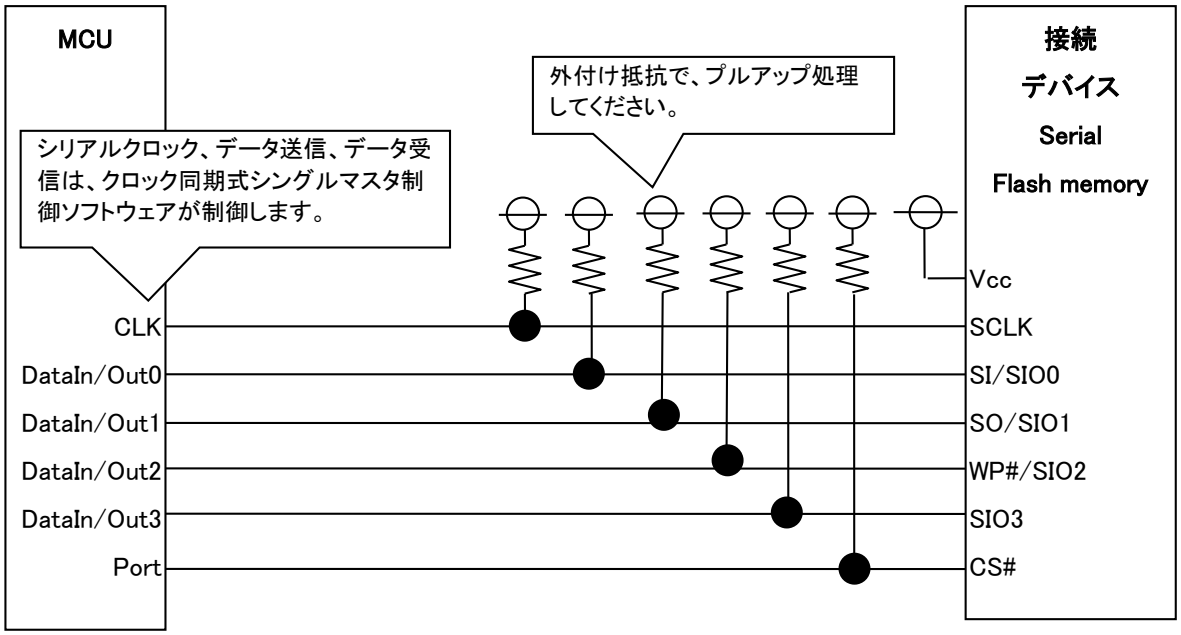
表 1-4　Dual-SPI 使用端子と機能

端子名	入出力	内容
CLK	出力	クロック出力
DataIn/Out0	入出力	マスタデータ入出力 0
DataIn/Out1	入出力	マスタデータ入出力 1
Port（図 1-2 の Port(SS#)）	出力	スレーブデバイスセレクト(SS#)出力

1.5.1.3　Quad-SPI 構成例

以下に Quad-SPI 使用時の接続例を示します。

なお、Quad-SPI を使用するためには、対象 MCU にクワッドシリアルペリフェラルインターフェース機能が搭載されている必要があります。



・MCU 上のシリアル I/O に使用される端子名は、MCU に依存します。
・WP#と RESET#を使用しない場合の例です。WP#と RESET#を使用する場合は接続デバイスの仕様書を確認してください。

図 1-4　Quad-SPI 使用時の MCU と SPI スレーブデバイスの接続例

表 1-5　Quad-SPI 使用端子と機能

端子名	入出力	内容
CLK	出力	クロック出力
DataIn/Out0	入出力	マスタデータ入出力 0
DataIn/Out1	入出力	マスタデータ入出力 1
DataIn/Out2	入出力	マスタデータ入出力 2
DataIn/Out3	入出力	マスタデータ入出力 3
Port（図 1-2 の Port(SS#)）	出力	スレーブデバイスセレクト(SS#)出力

1.6　ソフトウェア説明

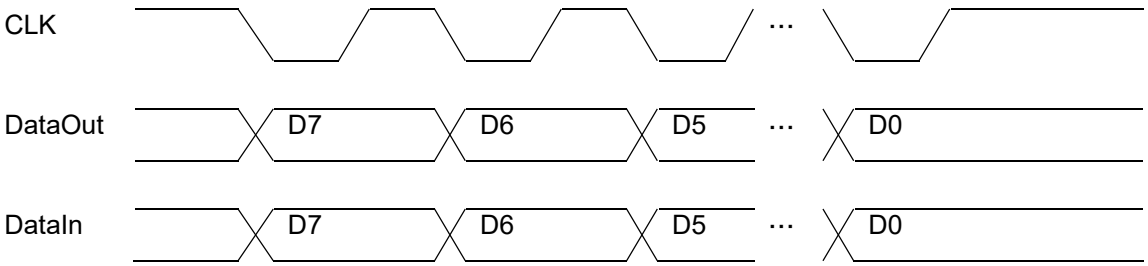
1.6.1　動作概要

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。

なお、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。

1.6.1.1　Single-SPI 制御時

図 1-5 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

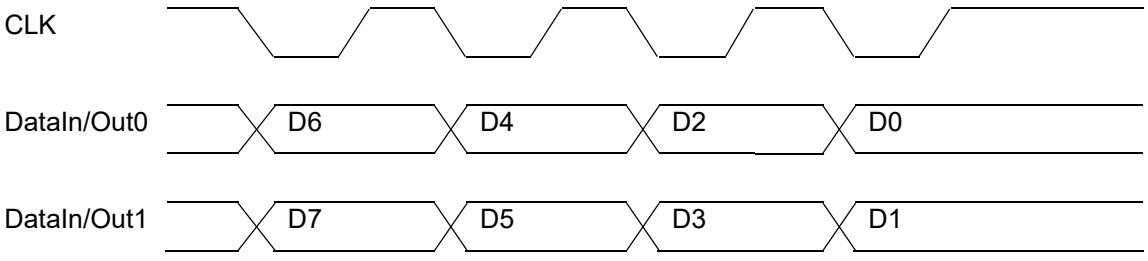
・MSB ファーストでの転送

・転送を行っていないときの CLK 端子のレベル：”H”

図 1-5　Single-SPI 時の制御可能なスレーブデバイスのタイミング

1.6.1.2　Dual-SPI 制御時

図 1-6 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

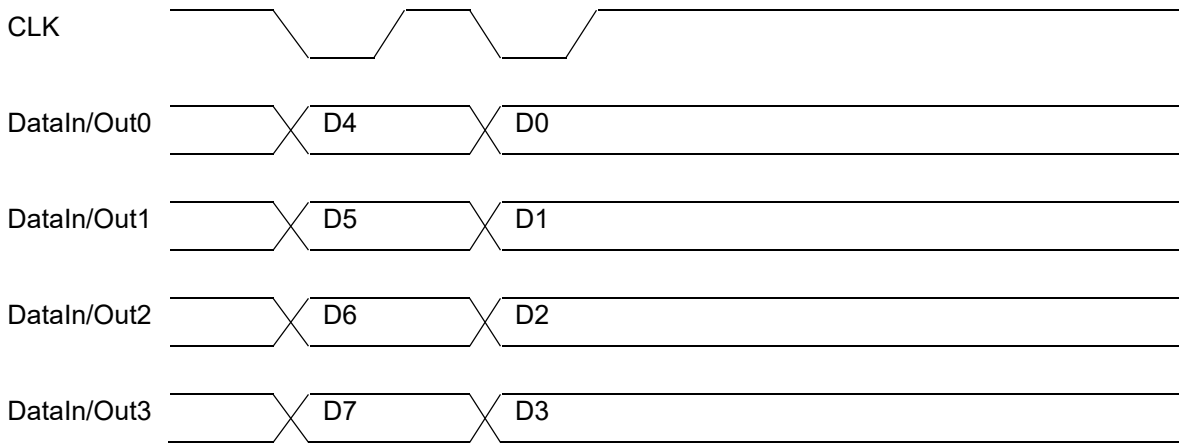
・MSB ファーストでの転送

・転送を行っていないときの CLK 端子のレベル：”H”

図 1-6　SDualSPI 時の制御可能なスレーブデバイスのタイミング

1.6.1.3　Quad-SPI 制御時

図 1-7 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 1-7　Quad-SPI 時の制御可能なスレーブデバイスのタイミング

1.6.2　Serial Flash memory の Chip select 端子制御

Serial Flash memory 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 Flash memory の Chip select（MCU の Port(SS#)）信号の立ち下がりから、Serial Flash memory の Clock（MCU の CLK）信号の立ち下がりまでの時間は、Serial Flash memory 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 Flash memory の Clock（MCU の CLK）信号の立ち上がりから、Serial Flash memory の Chip select（MCU の Port(SS#)）信号の立ち上がりまでの時間は、Serial Flash memory 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。

1.6.3　ソフトウェア構成

図 1-8 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。

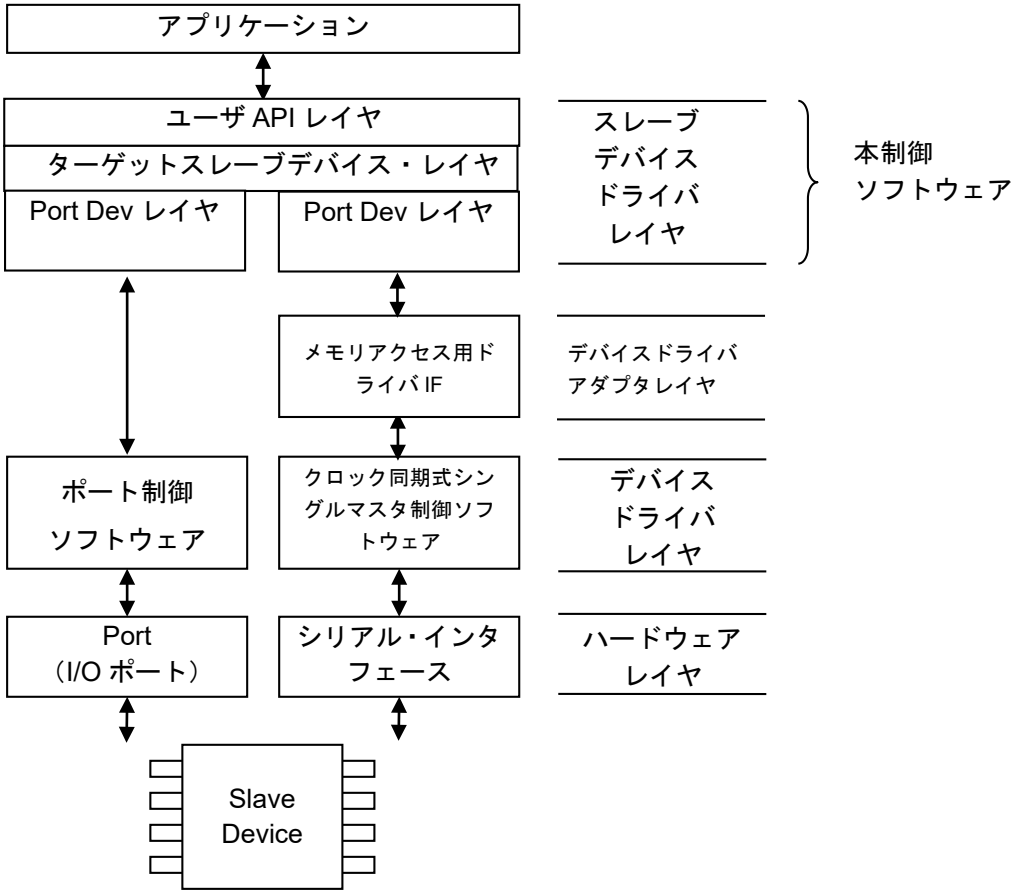


図 1-8　ソフトウェア構成

(1)　ユーザ API レイヤ（r_flash_spi.c）

ユーザインタフェース部であり、下位層のデバイスドライバに依存しない部分です。

(2)　ターゲットスレーブデバイス・レイヤ（r_flash_spi_type.c）

Serial Flash memory 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。

(3)　ドライバ・インタフェース（I/F）・レイヤ（r_flash_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_flash_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

Macronix International Co., Ltd 社製 MX25L/MX66L/MX25R family serial NOR Flash memory の制御例を同梱していますので、参照してください。

Dialog Semiconductor Plc.,社製 AT25QF family serial NOR Flash memory の制御例を同梱していますので、参照してください。

1.6.4　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

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。

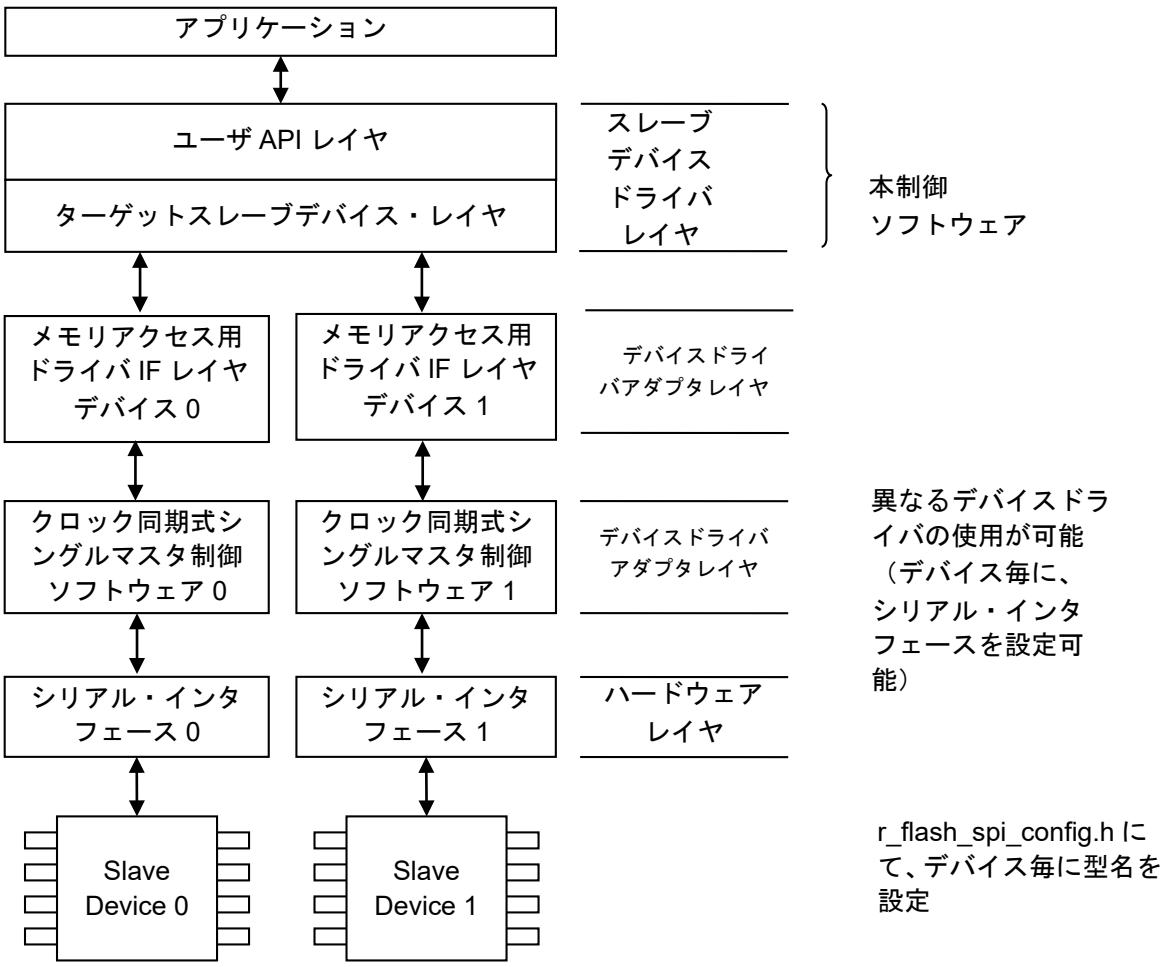


図 1-9　クロック同期式シングルマスタ制御ソフトとウェアとのソフトウェア構成

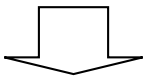
1.6.5　データバッファと送信／受信データの関係

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。

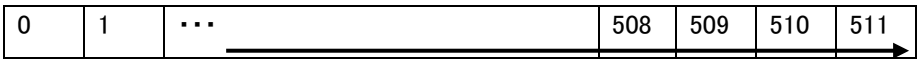
マスタ送信時

RAM 上の送信データバッファ（バイト表示）

データの送信順番



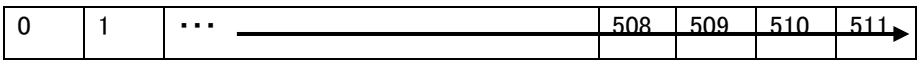
スレーブデバイスへの書き込み（バイト表示）



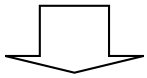
データの受信順

マスタ受信時

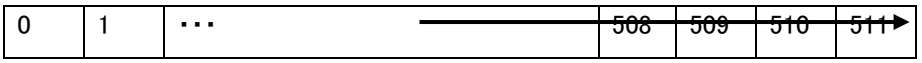
スレーブデバイスからの読み出し（バイト表示）



データの送信順



RAM 上のデータバッファ（バイト表示）



受信データバッファへの書き込み

図 1-10　データバッファと送信／受信データの関係

1.6.6　状態遷移図

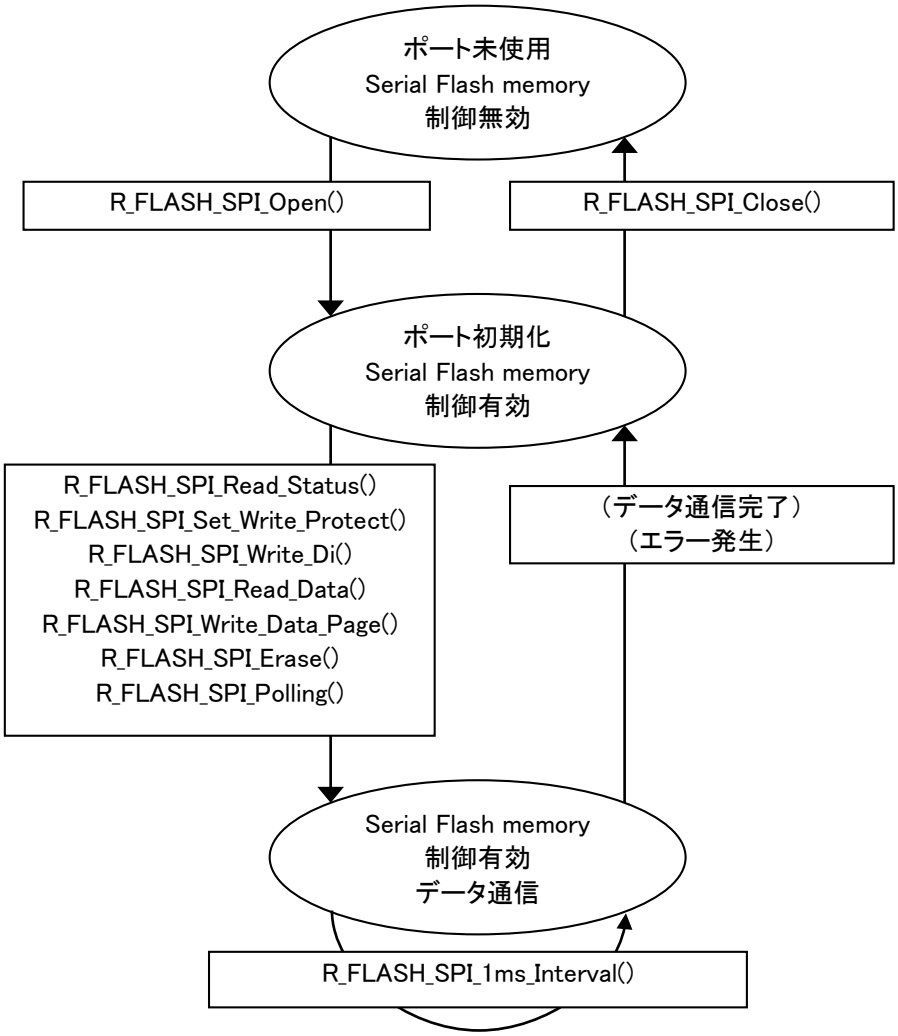


図 1-11　状態遷移図

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 Rev.1.04 以上
- r_rspi_rx（RSPI FIT モジュールを使用する場合）
- r_qspix_rx（QSPIX FIT モジュールを使用する場合）
- r_rsci_rx（RSCI 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_scifa_smstr_rx（クロック同期式シングルマスタ制御に SCIFA 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、「5.1」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_flash_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_flash_spi_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。

```
#include "r_flash_spi_if.h"
#include "r_flash_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グレーションオプションの設定は、r_flash_spi_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_flash_spi_config.h*	
#define FLASH_SPI_CFG_WEL_CHK ※デフォルト値は“1（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。（1：有効、0：無効）
#define FLASH_SPI_CFG_LONGQ_ENABLE ※デフォルトは“0（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。（1：有効、0：無効） 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 FLASH_SPI_CFG_USE_GPIO_MPC_FIT ※デフォルトは“0（無効）”	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。（1：有効、0：無効） 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 FLASH_SPI_CFG_DEVx_INCLUDED ※デバイス 0 のデフォルト値は、“1（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。（1：有効、0：無効） 最低でも 1 デバイスは有効にしてください。
#define FLASH_SPI_CFG_DEVx_MX25L #define FLASH_SPI_CFG_DEVx_MX66L #define FLASH_SPI_CFG_DEVx_MX25R #define FLASH_SPI_CFG_DEVx_AT25QF ※デバイス 0 のデフォルト値は、 FLASH_SPI_CFG_DEVx_MX25L を“1”、その他“0” ※DEVx の“x ”はデバイス番号（x=0 or 1）	デバイス x で制御する Serial Flash memory を 1 つだけ選択してください。（1：制御対象、0：制御非対象）

Configuration options in *r_flash_spi_config.h*	
#define FLASH_SPI_CFG_DEVx_SIZE_512K #define FLASH_SPI_CFG_DEVx_SIZE_2M #define FLASH_SPI_CFG_DEVx_SIZE_4M #define FLASH_SPI_CFG_DEVx_SIZE_8M #define FLASH_SPI_CFG_DEVx_SIZE_16M #define FLASH_SPI_CFG_DEVx_SIZE_32M #define FLASH_SPI_CFG_DEVx_SIZE_64M #define FLASH_SPI_CFG_DEVx_SIZE_128M #define FLASH_SPI_CFG_DEVx_SIZE_256M #define FLASH_SPI_CFG_DEVx_SIZE_512M #define FLASH_SPI_CFG_DEVx_SIZE_1G ※デバイス 0 のデフォルト値は FLASH_SPI_CFG_DEVx_SIZE_64M を“1”、その他“0” ※DEVx の“x”はデバイス番号（x=0 or 1）	デバイス x で制御する Serial Flash memory の容量を 1 つだけ選択してください。（1：制御対象、0：制御非対象）
#define FLASH_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 FLASH_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_flash_spi_if.h で記載されています。

```
/* FLASH Memory 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         */
   flash_spi_opmode_t op_mode;      /* SPI operating mode; ignore it when using
                                write/read data in a Security register page */
} flash_spi_info_t;                 /* 20 bytes                            */

/* FLASH Memory size information */
typedef struct
{
   uint32_t    mem_size;            /* Max memory size                     */
   uint32_t    wpag_size;           /* Write page size                     */
} flash_spi_mem_info_t;             /* 8 bytes                             */

/* FLASH Memory erase information */
typedef struct
{
   uint32_t    addr;                /* Address to issue a command          */
   flash_spi_erase_mode_t  mode;    /* Mode of erase                       */
} flash_spi_erase_info_t;           /* 8 bytes                             */

/* FLASH Memory register information */
typedef struct
{
   uint8_t     status;        /* Status register                          */
   uint8_t     config1;       /* Configuration or Configuration-1 register */
   uint8_t     config2;       /* Configuration-2 register                 */
   uint8_t     rsv[1];
} flash_spi_reg_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_flash_spi rev.3.20

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 8.03.00.202202
(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: 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_flash_spi.c, r_flash_spi_dev_port_iodefine.c, r_flash_spi_drvif.c,
r_flash_spi_type.c,r_flash_spi_type_sub.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
RX113	ROM	**4,491 バイト**	**11,396 バイト**	**7,825 バイト**
	RAM	**8 バイト**	**8 バイト**	**8 バイト**
	スタック	**140 バイト**	–	**200 バイト**
RX231	ROM	**4,495 バイト**	**11,436 バイト**	**7,860 バイト**
	RAM	**8 バイト**	**8 バイト**	**8 バイト**
	スタック	**140 バイト**	–	**200 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
RX64M	ROM	**4,535 バイト**	**11,436 バイト**	**7,902 バイト**
	RAM	**8 バイト**	**8 バイト**	**8 バイト**
	スタック	**140 バイト**	**–**	**232 バイト**
RX71M	ROM	**4,535 バイト**	**11,436 バイト**	**7,823 バイト**
	RAM	**8 バイト**	**8 バイト**	**8 バイト**
	スタック	**140 バイト**	**–**	**232 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_flash_spi_if.h で記載されています。

```
typedef enum e_flash_status
{
   FLASH_SPI_SUCCESS_BUSY  =  1,    /* Successful operation (EERPOM is busy) */
   FLASH_SPI_SUCCESS       =  0,    /* Successful operation                  */
   FLASH_SPI_ERR_PARAM     =  -1,   /* Parameter error                       */
   FLASH_SPI_ERR_HARD      =  -2,   /* Hardware error                        */
   FLASH_SPI_ERR_WP        =  -4,   /* Write-protection error                */
   FLASH_SPI_ERR_TIMEOUT   =  -6    /* time out error                        */
   FLASH_SPI_ERR_OTHER     =  -7    /* Other error                           */
} flash_spi_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)、(2)、(4)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(3)の方法を使用してください。

(1) e^2 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
e^2 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: e^2 studio 編 (R20AN0451)」を参照してください。

(2) 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。

(3) 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。

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

#r_flash_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_flash_spi_if.h の「#define FLASH_SPI_CFG_USE_FIT」を無効にしてください。

#r_flash_spi_if.h に「#define FLASH_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 FLASH_SPI_CFG_RX64M としてください。

r_flash_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「1.5.1(1) Single-SPI 制御時」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_FLASH_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_FLASH_SPI_Open()前	GPIO H 出力状態 **ユーザ側で設定**
R_FLASH_SPI_Open()後	GPIO H 出力状態 本モジュールで設定
R_FLASH_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。「1.5.1 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

R_FLASH_SPI_Open()

Serial Flash memory 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

```
flash_spi_status_t R_FLASH_SPI_Open(
    uint8_t devno
)
```

Parameters

devno
デバイス番号（0, 1）

Return Values

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

Properties

r_flash_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

```
flash_spi_status_t   ret = FLASH_SPI_SUCCESS;

ret = R_FLASH_SPI_Open(FLASH_SPI_DEV0);
```

Special Notes:

本ユーザ API コール後、R_FLASH_SPI_Polling()により、Serial Flash memory 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。Serial Flash memory 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、Serial Flash memory の書き込みサイクル中にシステムリセットが発生し、Serial Flash memory の制御を最初からやり直す場合、書き込みサイクル中の Serial Flash memory にアクセスする可能性があります。

R_FLASH_SPI_Close()

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

Format

```
flash_spi_status_t R_FLASH_SPI_Close(
    uint8_t devno
)
```

Parameters

devno
デバイス番号（0, 1）

Return Values

```
FLASH_SPI_SUCCESS       /* 正常終了した場合 */
FLASH_SPI_ERR_PARAM     /* パラメータ異常の場合 */
FLASH_SPI_ERR_OTHER     /* 他のエラーの場合 */
```

Properties

r_flash_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

```
flash_spi_status_t   ret = FLASH_SPI_SUCCESS;

ret = R_FLASH_SPI_Close(FLASH_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_FLASH_SPI_Polling()により、Serial Flash memory 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、Serial Flash memory が書き込みサイクルに遷移していない状態で、Serial Flash memory の制御を再開することができます。

R_FLASH_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

```
flash_spi_status_t R_FLASH_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

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_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。
なお、プロテクト領域とプロテクトビットの関係は、使用する Serial Flash memory のデータシートを参照してください。

- < Macronix International Co., Ltd 社製 MX25L/MX66L/MX25R family serial NOR Flash memory の場合 >

表 3-1　MX25L/MX66L/MX25R family serial NOR Flash memory の場合

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SRWD (status register write protect)	QE (Quad Enable)	BP3 (level of protected block)	BP2 (level of protected block)	BP1 (level of protected block)	BP0 (level of protected block)	WEL (write enable latch)	WIP (write in progress bit)
1=status register write disable	1=Quad Enable 0=not Quad Enable	*1	*1	*1	*1	1=write enable 0=not write enable	1=write operation 0=not in write operation
Non-volatile bit	Non-volatile bit	Non-volatile bit	Non-volatile bit	Non-volatile bit	Non-volatile bit	Volatile bit	Volatile bit

注 1：1 に設定すると、指定されたメモリ領域がプログラムとイレーズ操作から保護されます。

- < Dialog Semiconductor Plc.の AT25QF ファミリシリアル NOR フラッシュメモリ>

表 3-2　AT25QF family serial NOR Flash memory の場合

Bit	Mnemonic	Name	Type	Description	
7	SRP0	Status Register Protection bit 0	R/W		*4
6	SEC	Block Protection	R/W	*3	*4
5	TB	Top or Bottom Protection	R/W	*2	*4
4	BP2	Block Protection bit 2	R/W	*1	*4
3	BP1	Block Protection bit 1	R/W	*1	*4
2	BP1	Block Protection bit 0	R/W	*1	*4
1	WEL	Write Enable Latch Status	R	0	Device is not Write Enable (default).
				1	Device is Write Enable.
0	$\overline{\text{RDY}}$ /BSY	Ready/Busy Status	R	0	Device is ready.
				1	Device is busy with an internal operation.

注 1：0 または 1 に設定すると、保護される配列の量を決定します。
注 2：0 または 1 に設定すると、配列の上部または配列の下部の保護が選択されます。
注 3：0 または 1 に設定すると、大ブロックサイズと小ブロックサイズのどちらかが選択されます。
注 4：フラッシュメモリの仕様書をご確認ください。

Example

```
flash_spi_status_t   ret = FLASH_SPI_SUCCESS;
uint32_t             stat = 0;

ret = R_FLASH_SPI_Read_Status(FLASH_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。

R_FLASH_SPI_Read_Status2()

この関数は、レジスタ 2 のステータスを読み出すために使用されます。Dialog Semiconductor Plc の AT25QF ファミリ シリアル NOR フラッシュ メモリ用の専用 API 関数です。

Format

```
flash_spi_status_t R_FLASH_SPI_Read_Status2(
    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

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_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 2 を読み取り、その内容を p_status に格納します。 次の情報が p_status に格納されます。

- **<** Dialog Semiconductor Plc.の AT25QF ファミリシリアル NOR フラッシュメモリ**>**

表 3-3　AT25QF family serial NOR Flash memory の場合

Bit	Mnemonic	Name	Type	Description	
7	E_SUS	Erase Suspend Status	R	0	Erase operation is not suspended (default).
				1	Erase operation is suspended.
6	CMP	Complement Block Protection	R/W	*1	*2
5	LB3	Lock Security Register 3	R/W	0	Security Register page-3 is not locked (default).
				1	Security Register page-3 cannot be erased/programmed. *3
4	LB2	Lock Security Register 2	R/W	0	Security Register page-2 is not locked (default).
				1	Security Register page-2 cannot be erased/programmed. *3
3	BP1	Lock Security Register 1	R/W	0	Security Register page-1 is not locked (default).
				1	Security Register page-1 cannot be erased/programmed. *3
2	P_SUS	Program Suspend Status	R/	0	Program operation is not suspended (default).
				1	Program operation is suspended.
1	QE	Quad Enable	R/W	0	$\overline{\text{HOLD}}$ and $\overline{\text{WP}}$ function normally.
				1	$\overline{\text{HOLD}}$ and $\overline{\text{WP}}$ are I/O pins (default).
0	SRP1	Status Register Protect bit 1	R/W		*2

注 1 : 0 または 1 に設定すると、他のビットの効果が補完されます。
注 2 : フラッシュメモリの仕様書をご確認ください。
注 3 : ロック ビットが 1 に設定されると、対応するセキュリティレジスタが永久にロックされます。セキュリティレジスタページの消去命令は、ロックビットが設定されたセキュリティレジスタでは無視されます。

Example

```
flash_spi_status_t    ret = FLASH_SPI_SUCCESS;
uint32_t              stat = 0;

ret = R_FLASH_SPI_Read_Status2(FLASH_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。

R_FLASH_SPI_Read_Status3()

この関数は、レジスタ 3 のステータスを読み出すために使用されます。Dialog Semiconductor Plc の AT25QF ファミリ シリアル NOR フラッシュ メモリ用の専用 API 関数です。

Format

```
flash_spi_status_t R_FLASH_SPI_Read_Status3(
    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

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_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 3 を読み取り、その内容を p_status に格納します。 次の情報が p_status に格納されます。

- **<** Dialog Semiconductor Plc.の AT25QF ファミリシリアル NOR フラッシュメモリ**>**

表 3-4　AT25QF family serial NOR Flash memory の場合

Bit	Mnemonic	Name	Type	Description	
7	Res	Reserved	R	0	Reserved bit.
6:5	DRV[1:0]	Drive Strength	R/W	11	Drive level. Th DRV1 and DRV0 bits are used to determine the output drive strength during read operations. One of the setting of below allows the drive strength to be set by hardware based on the VCC level. Four drive settings are supported. This field is encoded as follows: 11: Auto (7pF base on VCC level. 10: 50% (15pF). 01: 75% (22pF). 00: 100% (30pF).
4:0	Res	Reserved	R	0	Reserved bit.

Example

```
flash_spi_status_t     ret = FLASH_SPI_SUCCESS;
uint32_t               stat = 0;

ret = R_FLASH_SPI_Read_Status3(FLASH_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。

R_FLASH_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

```
flash_spi_status_t R_FLASH_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

Return Values

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_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。
ライトプロテクト設定データ(wpsts)は下記のように設定してください。
プロテクト領域とプロテクトビットの関係は、使用する Serial Flash memory のデータシートを参照してください。

＜Macronix International Co., Ltd 社製 MX25L/MX66L/MX25R family serial NOR Flash memory の場合＞
Top/Bottom 設定は、コンフィグレーション書き込み処理で設定してください。

wpsts	BP3	BP2	BP1	BP0
0x00	0	0	0	0
0x01	0	0	0	1
0x02	0	0	1	0
0x03	0	0	1	1
0x04	0	1	0	0
0x05	0	1	0	1
0x06	0	1	1	0
0x07	0	1	1	1
0x08	1	0	0	0
0x09	1	0	0	1
0x0a	1	0	1	0
0x0b	1	0	1	1
0x0c	1	1	0	0
0x0d	1	1	0	1
0x0e	1	1	1	0
0x0f	1	1	1	1

＜Dialog Semiconductor Plc.の AT25QF ファミリシリアル NOR フラッシュメモリ＞
ステータスレジスタ 1 の書き込み処理中に大小ブロックサイズ、トップ、ボトムの設定を行います。
ステータスレジスタ 2 の書き込み処理中に補完設定を行います。

wpsts	BP2	BP1	BP0
0x00	0	0	0
0x01	0	0	1
0x02	0	1	0
0x03	0	1	1
0x04	1	0	0
0x05	1	0	1
0x06	1	1	0
0x07	1	1	1

本ユーザ API が正常終了した場合、Serial Flash memory 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_FLASH_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、Serial Flash memory はその処理を受け付けません。
R_FLASH_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

```
#define FLASH_WR_BUSY_WAIT (uint32_t)(40)     /* 40 * 1ms = 40ms */

flash_spi_status_t      ret    = FLASH_SPI_SUCCESS;
uint32_t                loop_cnt = 0;
flash_spi_poll_mode_t   mode;

ret = R_FLASH_SPI_Set_Write_Protect(FLASH_SPI_DEV0, 0);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

loop_cnt = FLASH_WR_BUSY_WAIT;
mode = FLASH_SPI_MODE_REG_WRITE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
   if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);     /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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。

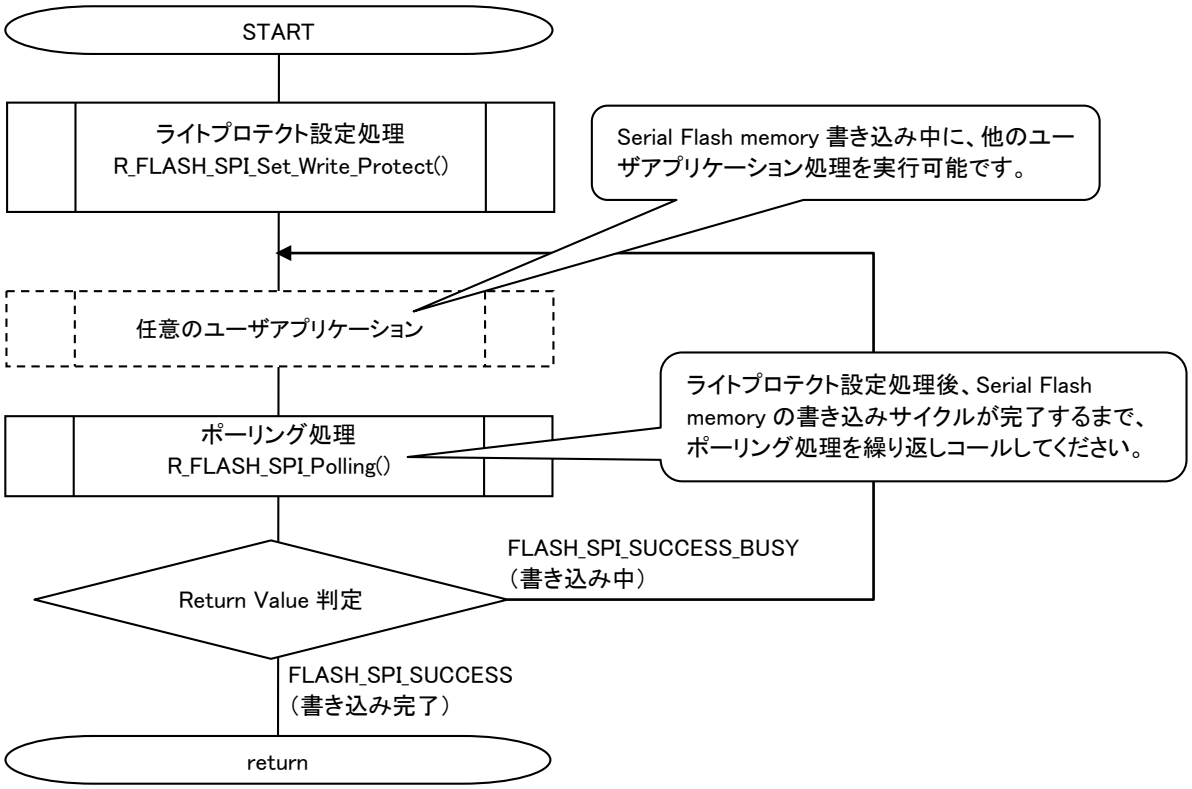


図 3-1　R_FLASH_SPI_Set_Write_Protect()の処理例

R_FLASH_SPI_Write_Di()

書き込みを禁止する際に使用する関数です。

Format

```
flash_spi_status_t R_FLASH_SPI_Write_Di(
    uint8_t devno
)
```

Parameters

devno

デバイス番号（0, 1）

Return Values

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_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

```
flash_spi_status_t    ret = FLASH_SPI_SUCCESS;

ret = R_FLASH_SPI_Write_Di(FLASH_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。

R_FLASH_SPI_Read_Data()

Serial Flash memory からデータを読み出す際に使用する関数です。

Format

```
flash_spi_status_t R_FLASH_SPI_Read_Data(
    uint8_t devno,
    flash_spi_info_t * p_flash_spi_info
)
```

Parameters

devno

デバイス番号（0, 1）

** p_flash_spi_info*

Serial Flash memory 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。

op_mode

SPI mode 設定

以下から 1 つ選択してください。

```
FLASH_SPI_SINGLE    /* Single-SPI（全二重通信） */
FLASH_SPI_DUAL      /* Dual-SPI（半二重通信） */
FLASH_SPI_QUAD      /* Quad-SPI（半二重通信） */
```

Return Values

```
FLASH_SPI_SUCCESS       /* 正常終了した場合 */
FLASH_SPI_ERR_PARAM     /* パラメータ異常の場合 */
FLASH_SPI_ERR_HARD      /* ハードウェアエラーの場合 */
FLASH_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_flash_spi_if.h にプロトタイプ宣言されています。

Description

Serial Flash memory 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、Serial Flash memory 容量-1 です。
ロールオーバーによる読み出しはできません。最終アドレスの読み出し後、一度処理を完了させて、再度アドレスを設定し直してから、本ユーザ API をコールしてください。
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、FLASH_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

```
flash_spi_status_t   ret = FLASH_SPI_SUCCESS;
flash_spi_info_t     Flash_Info_R;
uint32_t             buf2[128/sizeof(uint32_t)];
                                /* the buffer boundary (4-byte unit) */

Flash_Info_R.addr    = 0;
Flash_Info_R.cnt     = 32;
Flash_Info_R.p_data  = (uint8_t *)&buf2[0];
Flash_Info_R.op_mode = FLASH_SPI_QUAD;
ret = R_FLASH_SPI_Read_Data(FLASH_SPI_DEV0, &Flash_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。
QSPIX のメモリマップモードでデータを読み込む場合、指定されたバンクのアドレスからのみデータを読み取ることができます。バンクとは、QSPI デバイスのフラッシュメモリスペースに対する 64MB のスライディングアクセスウィンドウのことです。コードは、バンクをまたいでフラッシュメモリからデータを読み取るように設計されていません。そのため、2 つの連続するバンク領域のアドレス領域から連続してデータを要求した場合、この機能は「FLASH_SPI_ERR_OTHER」エラーを返します。

R_FLASH_SPI_Write_Data_Page()

Serial Flash memory へ、1Page 単位でデータを書き込む際に使用する関数です。

Format

```
flash_spi_status_t R_FLASH_SPI_Write_Data_Page(
    uint8_t devno,
    flash_spi_info_t * p_flash_spi_info
)
```

Parameters

devno

デバイス番号（0, 1）

** p_flash_spi_info*

Serial Flash memory 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。

op_mode

SPI mode 設定

以下から 1 つ選択してください。

```
FLASH_SPI_SINGLE    /* Single-SPI（全二重通信） */
FLASH_SPI_QUAD      /* Quad-SPI（半二重通信） */
```

※：FLASH_SPI_DUAL は設定禁止。

Return Values

```
FLASH_SPI_SUCCESS       /* 正常終了した場合 */
FLASH_SPI_ERR_PARAM     /* パラメータ異常の場合 */
FLASH_SPI_ERR_HARD      /* ハードウェアエラーの場合 */
FLASH_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_flash_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 Serial Flash memory 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。
Serial Flash memory 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FLASH_SPI_ERR_OTHER を返します。
最大書き込みアドレスは、Serial Flash memory 容量-1 です。
書き込みバイト数（cnt）に設定できる最大値は、Serial Flash memory 容量の値です。
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 FLASH_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 Serial Flash memory 情報構造体（p_flash_spi_info）に残ります。未変更のまま再びその p_flash_spi_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、Serial Flash memory 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_FLASH_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、Serial Flash memory はその処理を受け付けません。
R_FLASH_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

```
#define FLASH_PP_BUSY_WAIT (uint32_t)(3)     /* 3 * 1ms = 3ms */

flash_spi_status_t   ret = FLASH_SPI_SUCCESS;
flash_spi_info_t     Flash_Info_W;
uint32_t             buf1[128/sizeof(uint32_t)];
                                /* the buffer boundary (4-byte unit) */
uint32_t             loop_cnt = 0;

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

do
{
   ret = R_FLASH_SPI_Write_Data_Page(FLASH_SPI_DEV0, &Flash_Info_W);
   if (FLASH_SPI_SUCCESS > ret)
   {
      /* Error */
   }

   loop_cnt = FLASH_PP_BUSY_WAIT;
   mode = FLASH_SPI_MODE_PROG_POLL;
   do
   {
      /* FLASH is busy.
         User application can perform other processing while flash is busy. */

         ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
         if (FLASH_SPI_SUCCESS_BUSY != ret)
         {
            /* FLASH is ready or error. */
            break;
         }
         loop_cnt--;
         wait_timer(0, 1);    /* 1ms */
   }
   while (0 != loop_cnt);
}
while (0 != Flash_Info_W.cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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。

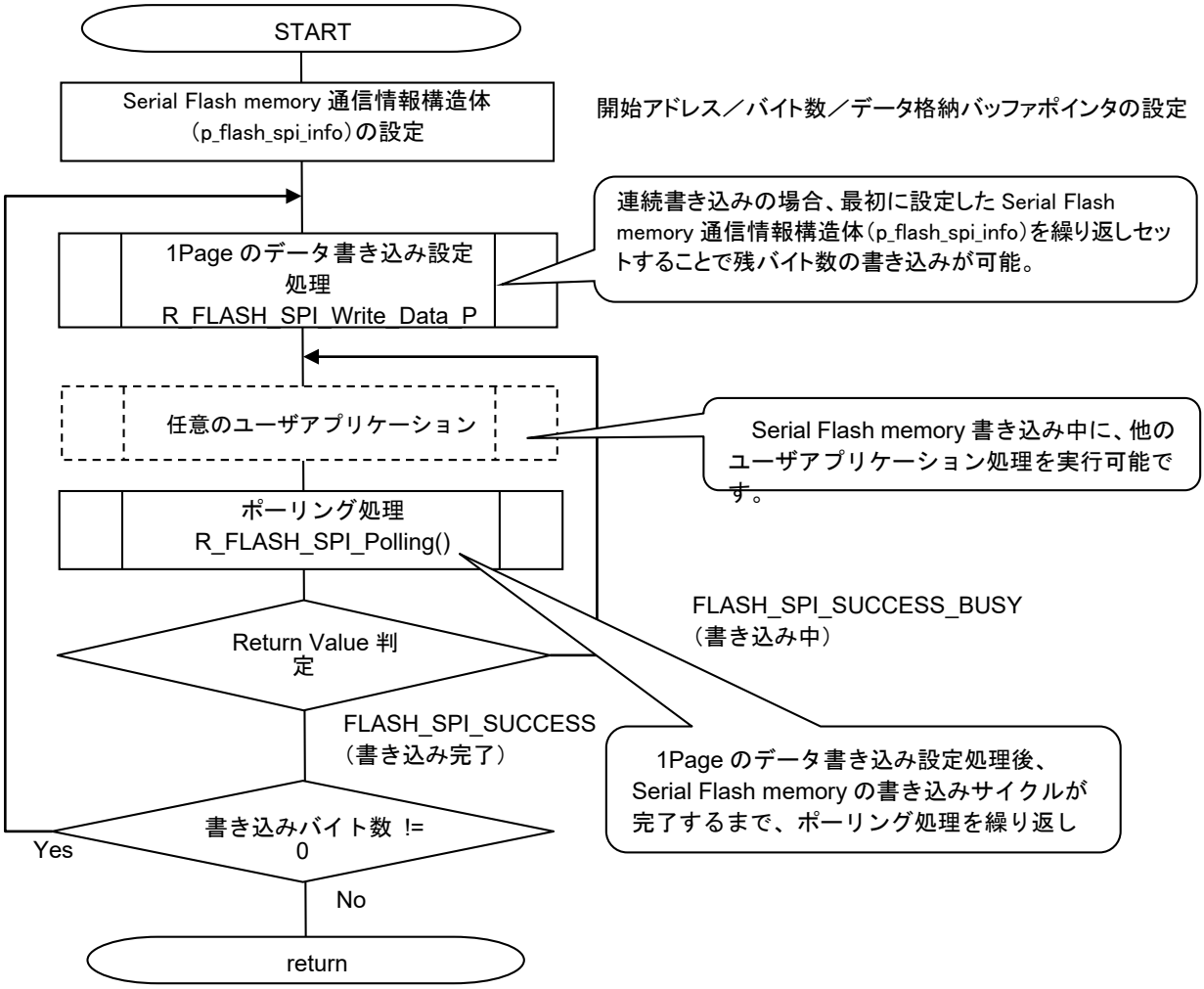


図 3-2　R_FLASH_SPI_Write_Data_Page()の処理例

R_FLASH_SPI_Erase()

mode の設定により、指定されたセクタの全データ消去（Sector Erase）、指定されたブロックの全データ消去（Block Erase：32KB ブロック、64KB ブロック）、指定されたチップの全データ消去（Chip Erase）を行います。

Format

```
flash_spi_status_t R_FLASH_SPI_Erase(
    uint8_t devno,
    flash_spi_erase_info_t * p_flash_spi_erase_info
)
```

Parameters

devno

デバイス番号（0, 1）

* p_flash_spi_erase_info

Serial Flash memory Erase 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。

mode

消去モード設定

以下から 1 つ選択してください。

＜Macronix International Co., Ltd 社製 MX25L/MX66L/MX25R family serial NOR Flash memory の場合＞

```
FLASH_SPI_MODE_C_ERASE          /* チップの全データ消去（Chip Erase） */
FLASH_SPI_MODE_S_ERASE          /* セクタの全データ消去（Sector Erase） */
FLASH_SPI_MODE_B32K_ERASE       /* ブロックの全データ消去（Block Erase：32KB） */
FLASH_SPI_MODE_B64K_ERASE       /* ブロックの全データ消去（Block Erase：64KB） */
```

＜Dialog Semiconductor Plc.の AT25QF ファミリシリアル NOR フラッシュメモリ＞

```
FLASH_SPI_MODE_C_ERASE          /* チップの全データ消去（Chip Erase） */
FLASH_SPI_MODE_BK4_ERASE        /* ブロックの全データ消去（Block Erase：4KB） */
FLASH_SPI_MODE_B32K_ERASE       /* ブロックの全データ消去（Block Erase：32KB） */
FLASH_SPI_MODE_B64K_ERASE       /* ブロックの全データ消去（Block Erase：64KB） */
FLASH_SPI_MODE_SCUR_ERASE       /* セキュリティレジスタページを消去します */
```

Return Values

```
FLASH_SPI_SUCCESS       /* 正常終了した場合 */
FLASH_SPI_ERR_PARAM     /* パラメータ異常の場合 */
FLASH_SPI_ERR_HARD      /* ハードウェアエラーの場合 */
FLASH_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_flash_spi_if.h にプロトタイプ宣言されています。

Description

Sector Erase の場合、addr にはセクタの先頭アドレスを設定してください。
Block Erase の場合、addr にはブロックの先頭アドレスを設定してください。
セキュリティ消去で、addr にセキュリティレジスタページの開始アドレスを指定します。

Chip Erase の場合、addr には 0x00000000 を設定してください。
Serial Flash memory への消去は、ライトプロテクト解除状態の場合のみ可能です。
プロテクトされた領域への消去はできません。エラーFLASH_SPI_ERR_OTHER を返します。
本ユーザ API が正常終了した場合、Serial Flash memory は消去サイクルに遷移しています。必ず、消去完了を R_FLASH_SPI_Polling()で確認してください。消去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、Serial Flash memory はその処理を受け付けません。
R_FLASH_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、消去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 3-3 を参照してください。

Example

```
#define FLASH_SE_BUSY_WAIT (uint32_t)(200)    /* 200 * 1ms = 200ms */

flash_spi_status_t      ret = FLASH_SPI_SUCCESS;
flash_spi_erase_info_t  Flash_Info_E;
uint32_t                loop_cnt = 0;

Flash_Info_E.addr    = 0;
Flash_Info_E.mode    = FLASH_SPI_MODE_S_ERASE;

ret = R_FLASH_SPI_Erase(FLASH_SPI_DEV0, &Flash_Info_E);
if (FLASH_SPI_SUCCESS > ret)
{
  /* Error */
}

loop_cnt = FLASH_SE_BUSY_WAIT;
mode = FLASH_SPI_MODE_ERASE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
   if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);     /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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。

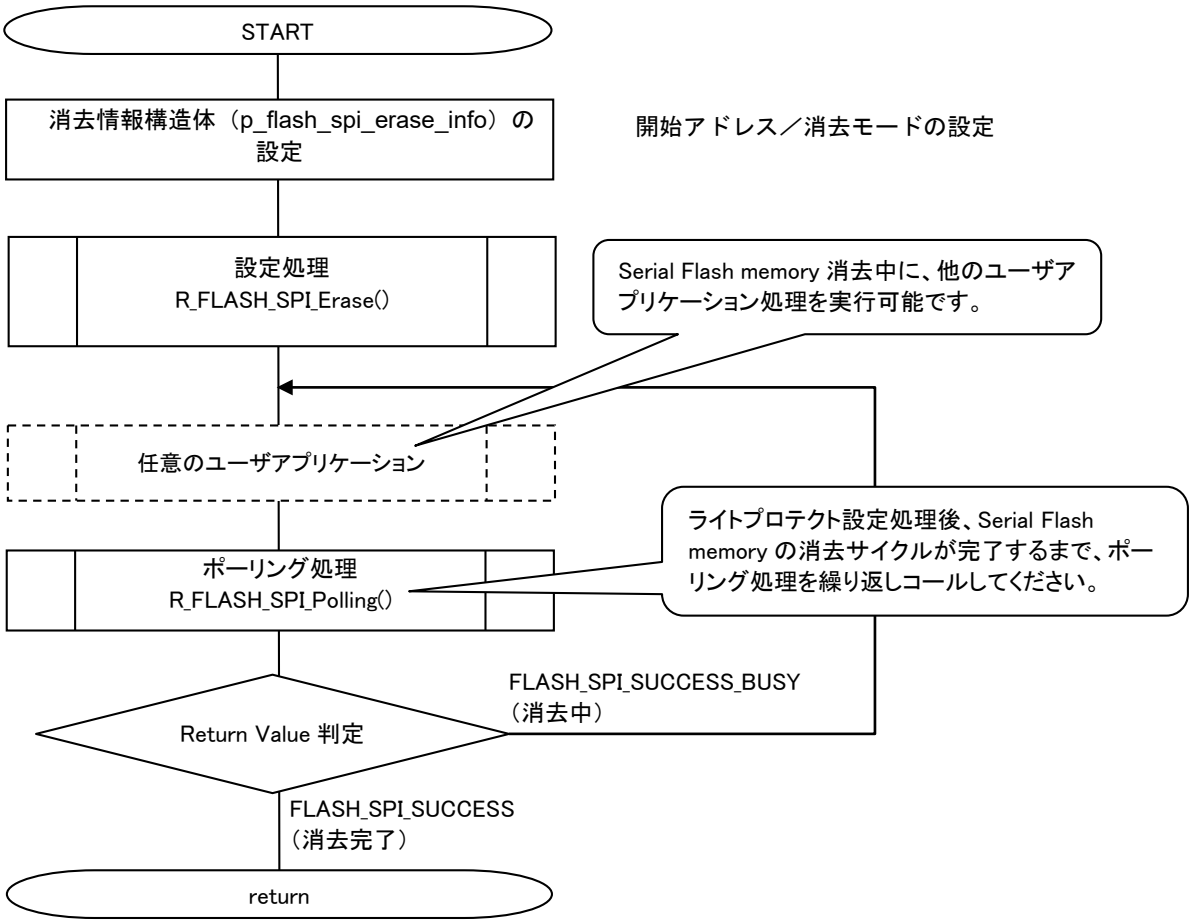


図 3-3　R_FLASH_SPI_Erase()の処理例

R_FLASH_SPI_Polling()

書き込み、または消去完了ポーリングする際に使用する関数です。

Format

```
flash_spi_status_t R_FLASH_SPI_Polling(
    uint8_t devno,
    flash_spi_poll_mode_t mode
)
```

Parameters

devno

デバイス番号（0, 1）

mode

完了待ち処理設定

以下から 1 つ選択してください。

```
FLASH_SPI_MODE_REG_WRITE_POLL    /* レジスタ書き込み完了待ち */
FLASH_SPI_MODE_PROG_POLL         /* データ書き込み完了待ち */
FLASH_SPI_MODE_ERASE_POLL        /* 消去完了待ち */
```

Return Values

```
FLASH_SPI_SUCCESS        /* 正常終了、かつ、書き込み完了の場合 */
FLASH_SPI_SUCCESS_BUSY   /* 正常終了、かつ、書き込み中の場合 */
FLASH_SPI_ERR_PARAM      /* パラメータ異常の場合 */
FLASH_SPI_ERR_HARD       /* ハードウェアエラーの場合 */
FLASH_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_flash_spi_if.h にプロトタイプ宣言されています。

Description

書き込み、または消去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_FLASH_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。

R_FLASH_SPI_Read_ID()

ID 情報を読み出す際に使用する関数です。

Format

```
flash_spi_status_t R_FLASH_SPI_Read_ID(
    uint8_t devno,
    uint8_t * p_data
)
```

Parameters

devno

デバイス番号（0, 1）

**p_data*

ID 情報格納バッファ。使用する Serial Flash memory によりサイズが異なります。下記を参照し、読み出しバッファを確保してください。

＜Macronix International Co., Ltd 社製 MX25L/MX66L/MX25R family serial NOR Flash memory の場合＞
Manufacture ID、及び Device ID を読み出します。読み出しバッファとして、3 バイトを設定してください。
(1) Manufacturer ID（1 バイト）
(2) Device ID（2 バイト）
＜Dialog Semiconductor Plc.の AT25QF ファミリシリアル NOR フラッシュメモリ＞
Manufacture ID、及び Device ID を読み出します。読み出しバッファとして、3 バイトを設定してください。
(1) Manufacturer ID（1 バイト）
(2) Device ID（2 バイト）

Return Values

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_spi_if.h にプロトタイプ宣言されています。

Description

Serial Flash memory の ID 情報を p_data に格納します。

Example

```
flash_spi_status_t    ret = FLASH_SPI_SUCCESS;
uint8_t    gID[4];

ret = R_FLASH_SPI_Read_ID(FLASH_SPI_DEV0, &gID[0]);
```

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。

R_FLASH_SPI_GetMemoryInfo()

Serial Flash memory のサイズ情報を取得する際に使用する関数です。

Format

```
flash_spi_status_t R_FLASH_SPI_GetMemoryInfo(
    uint8_t devno,
    flash_spi_mem_info_t * p_flash_spi_mem_info
)
```

Parameters

devno

デバイス番号（0, 1）

** p_flash_spi_mem_info*

Serial Flash memory 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

```
FLASH_SPI_SUCCESS        /* 正常終了した場合 */
FLASH_SPI_ERR_PARAM      /* パラメータ異常の場合 */
```

Properties

r_flash_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 Serial Flash memory サイズ情報を取得します。

Example

```
flash_spi_status_t    ret = FLASH_SPI_SUCCESS;
flash_spi_mem_info_t  Flash_MemInfo;

ret = R_FLASH_SPI_GetMemoryInfo(FLASH_SPI_DEV0, &Flash_MemInfo);
```

Special Notes:

なし

R_FLASH_SPI_Read_Configuration()

コンフィグレーションレジスタを読み出す際に使用する関数です。Macronix International Co., Ltd 社製 MX25L/MX66L/MX25R family serial NOR Flash memory 専用の API です。

Format

```
flash_spi_status_t R_FLASH_SPI_Read_Configuration (
    uint8_t devno,
    uint8_t * p_config
)
```

Parameters

devno

デバイス番号（0, 1）

**p_config*

コンフィグレーションレジスタ格納バッファ。使用する Serial NOR Flash memory によりサイズが異なります。下記を参照し、読み出しバッファを確保してください。

＜Macronix International Co., Ltd 社製 MX25L/MX66L family serial NOR Flash memory の場合＞
コンフィグレーションレジスタを読み出します。読み出しバッファとして、1 バイトを設定してください。

＜Macronix International Co., Ltd 社製 MX25R family serial NOR Flash memory の場合＞
コンフィグレーションレジス-1 とコンフィグレーションレジス-2 を読み出します。読み出しバッファとして、2 バイトを設定してください。

Return Values

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_spi_if.h にプロトタイプ宣言されています。

Description

Serial NOR Flash memory のコンフィグレーションレジスタを読み出し、p_config に格納します。
p_config には下記情報が格納されます。ただし、使用する Serial NOR Flash memory によっては、機能が割り当てられている場合や、Reserved bit の場合があります。詳細は、使用する Serial NOR Flash memory のデータシートを参照してください。

表 3-5　コンフィグレーションレジスター覧

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
DC1 (Dummy cycle 1)	DC0 (Dummy cycle 0)	4 BYTE	PBE (Preamble bit Enable)	TB (top/bottom selected)	ODS 2 (output driver strength)	ODS 1 (output driver strength)	ODS 0 (output driver strength)
※1	※1	0=3-byte address mode 1=4-byte address mode (Default=0)	0=Disable 1=Enable	0=Top area protect 1=Bottom area protect (Default=0)	※1	※1	※1
volatile bit	volatile bit	volatile bit	volatile bit	OTP	volatile bit	volatile bit	volatile bit

※1 ：See the specification of the Flash memory.

表 3-6　コンフィグレーションレジスタ-2 一覧

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	L/H Switch	Reserved
x	x	x	x	x	x	0 = Low power mode (default) 1 = High performance mode	x
x	x	x	x	x	x	volatile bit	x

Example

```
flash_spi_status_t    ret = FLASH_SPI_SUCCESS;
uint8_t    gConfig[4];    /* the buffer boundary (4-byte unit) */

ret = R_FLASH_SPI_Read_Configuration(FLASH_SPI_DEV0, &gConfig[0]);
```

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。

R_FLASH_SPI_Write_Configuration()

コンフィグレーションレジスタに値を書き込む際に使用する関数です。Macronix International Co., Ltd 社製 MX25L/MX66L/MX25R family serial NOR Flash memory 専用の API です。

Format

```
flash_spi_status_t R_FLASH_SPI_Write_Configration(
    uint8_t devno,
    flash_spi_reg_info_t * p_reg
)
```

Parameters

devno

デバイス番号（0, 1）

* p_reg

レジスタ情報構造体。構造体のアドレスは 4 バイトの境界値としてください。

status

ステータスレジスタ（本制御ソフトウェアで使用するため、設定禁止）

config1

コンフィグレーションレジスタ設定データ

config2

コンフィグレーションレジスタ-2 設定データ

なお、使用する Serial NOR Flash memory により構造体の構成が異なります。下記を参照し、値を設定してください。また、設定値は Description を参照してください。

＜Macronix International Co., Ltd 社製 MX25L/MX66L family serial NOR Flash memory の場合＞
p_reg->config1 に設定した値をコンフィグレーションレジスタに書き込みます。
p_reg->config2 の設定は無効です。

＜Macronix International Co., Ltd 社製 MX25R family serial NOR Flash memory の場合＞
p_reg->config1 に設定した値をコンフィグレーションレジスタ-1 に書き込みます。
p_reg->config2 に設定した値をコンフィグレーションレジスタ-2 に書き込みます。

Return Values

```
FLASH_SPI_SUCCESS       /* 正常終了した場合 */
FLASH_SPI_ERR_PARAM     /* パラメータ異常の場合 */
FLASH_SPI_ERR_HARD      /* ハードウェアエラーの場合 */
FLASH_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_flash_spi_if.h にプロトタイプ宣言されています。

Description

p_reg->config1 及び p_reg->config2 に設定した値をコンフィグレーションレジスタに書き込みます。

下記情報を参照し p_reg->config1 及び p_reg->config2 を設定してください。ただし、使用する Serial NOR Flash memory によっては、機能が割り当てられている場合や、Reserved bit の場合があります。詳細は、使用する Serial NOR Flash memory のデータシートを参照してください。

コンフィグレーションレジスタ
- Bits 7 to 6: DC1-DC0 (Dummy cycle)
  - See the specification of the Flash memory.
- Bit 5: 4BYTE (4BYTE Indicator)
  - 1: 4-byte address mode
  - 0: 3-byte address mode
- Bit 4: PBE (Preamble bit Enable)
  - 1: Enable
  - 0: Disable
- Bit 3: TB (Top/Bottom)
  - 1: Bottom area protect
  - 0: Top area protect
- Bits 2 to 0: ODS2-ODS0 (Output driver strength)
  - See the specification of the Flash memory.

コンフィグレーションレジスタ-2
- Bits 7 to 2: Reserved
- Bit 1: L/H Switch
  - 1: High performance mode
  - 0: Low power mode
- Bit 0: Reserved

本ユーザ API をコールする場合、事前にコンフィグレーションレジスタの値を読み出し、書き換えたい bit 値のみ変更し、p_reg->config1 及び p_reg->config2 に設定してください。
処理終了後、コンフィグレーションレジスタを読み出して、書き込み値を確認してください。
4BYTE bit は読み出し専用であるため、設定は無視されます。R_FLASH_SPI_Set_4byte_Address_Mode() によりこの bit に 1 をセットすることができます。
本ユーザ API が正常終了した場合、Serial Flash memory 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_FLASH_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、Serial Flash memory はその処理を受け付けません。
R_FLASH_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 3-4 を参照してください。

Example

```
#define FLASH_WR_BUSY_WAIT (uint32_t)(40)    /* 40 * 1ms = 40ms */

flash_spi_status_t   ret   = FLASH_SPI_SUCCESS;
uint32_t             loop_cnt = 0;
flash_spi_reg_info_t Reg;
uint8_t              gConfig[4];   /* the buffer boundary (4-byte unit) */

ret = R_FLASH_SPI_Read_Configuration(FLASH_SPI_DEV0, &gConfig[0]);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

Reg.config1 = (gConfig[0] | 0x10);     /* Set Preamble bit Enable */
ret = R_FLASH_SPI_Write_Configuration(FLASH_SPI_DEV0, &Reg);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

loop_cnt = FLASH_WR_BUSY_WAIT;
mode = FLASH_SPI_MODE_REG_WRITE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
   if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);    /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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。

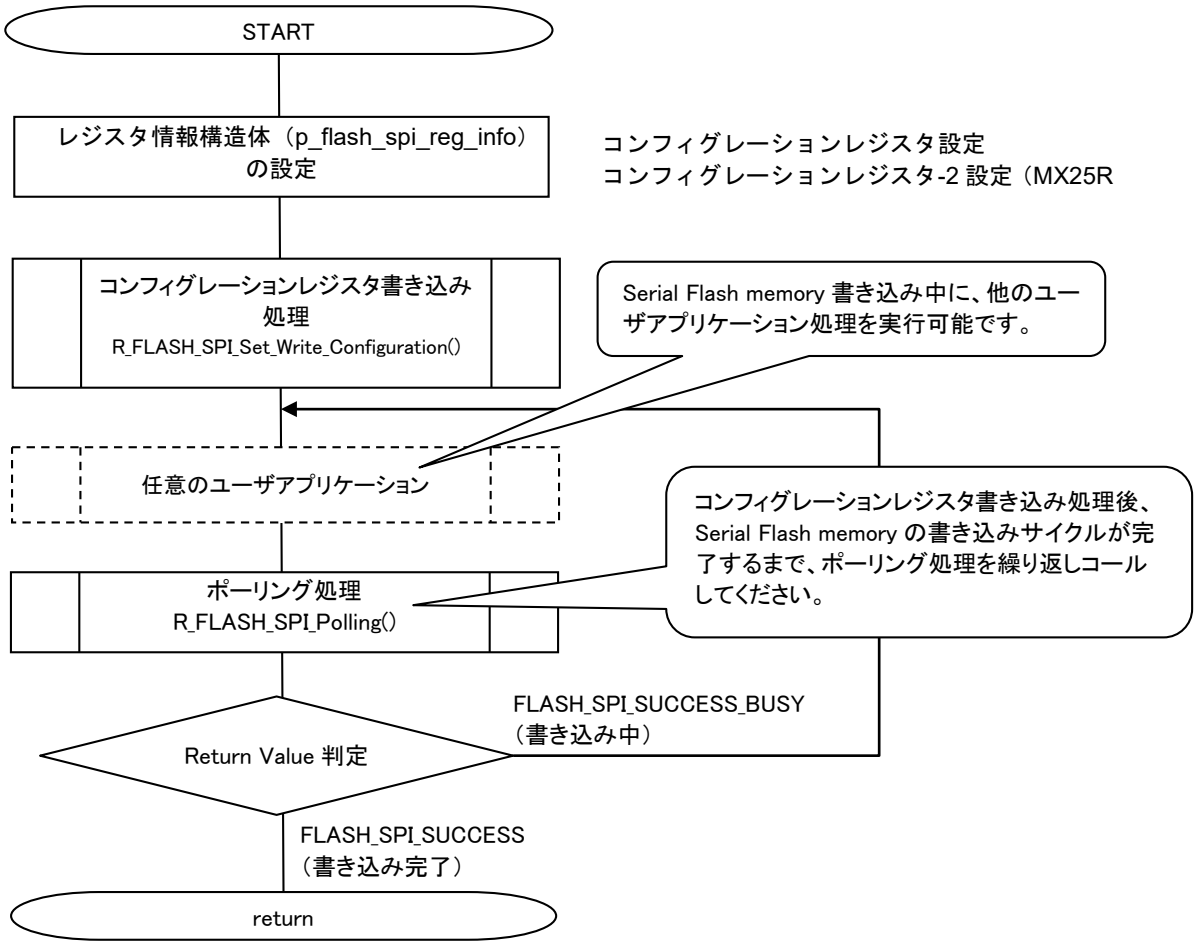


図 3-4　R_FLASH_SPI_Write_Configuration()の処理例

R_FLASH_SPI_Write_Status()

この関数はステータスレジスタ 1 への書き込みに使用されます。Dialog Semiconductor Plc の AT25QF ファミリ シリアル NOR フラッシュ メモリ用の専用 API 関数です。

Format

```
flash_spi_status_t R_FLASH_SPI_Write_Status(
    uint8_t devno,
    flash_spi_reg_info_t * p_reg
)
```

Parameters

devno
デバイス番号（0, 1）

* p_reg
ステータスレジスタ設定データバッファ

Return Values

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_spi_if.h にプロトタイプ宣言されています。

Description

p_reg に設定された値はステータスレジスタ 1 に書き込まれます。ただし、使用する Serial NOR Flash memory によっては、機能が割り当てられている場合や、Reserved bit の場合があります。詳細は、使用する Serial NOR Flash memory のデータシートを参照してください。

ステータス レジスタ 1
Bit 7: Status Register Protection bit 0
　See the specification of the Flash memory.
Bit 6: Block Protection
　See the specification of the Flash memory.
Bit 5: TB (Top/Bottom)
　See the specification of the Flash memory.
Bits 4 to 2: BP2-BP0 (Block Protection bit)
　See the specification of the Flash memory.

このユーザ API 関数を呼び出す前に、ステータスレジスタ 1 の値を読み取り、上書きが必要なビットのみ値を変更してください。 処理終了後、ステータスレジスタ 1 を読み出し、書き込んだ値が正しいことを確認してください。
本ユーザ API が正常終了した場合、Serial Flash memory 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_FLASH_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、Serial Flash memory はその処理を受け付けません。R_FLASH_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。

Example

```
#define FLASH_WR_BUSY_WAIT (uint32_t)(40)     /* 40 * 1ms = 40ms */

flash_spi_status_t   ret   = FLASH_SPI_SUCCESS;
uint32_t             loop_cnt = 0;
uint8_t              gStat;
uint8_t              Reg;

ret = R_FLASH_SPI_Read_Status(FLASH_SPI_DEV0, &gStat]);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

Reg = (gStat | 0x10);
ret = R_FLASH_SPI_Write_Status(FLASH_SPI_DEV0, &Reg);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

loop_cnt = FLASH_WR_BUSY_WAIT;
mode = FLASH_SPI_MODE_REG_WRITE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
   if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);     /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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。

R_FLASH_SPI_Write_Status2()

この関数はステータスレジスタ 2 への書き込みに使用されます。Dialog Semiconductor Plc の AT25QF ファミリ シリアル NOR フラッシュ メモリ用の専用 API 関数です。

Format

```
flash_spi_status_t R_FLASH_SPI_Write_Status2 (
    uint8_t devno,
    flash_spi_reg_info_t * p_reg
)
```

Parameters

devno
デバイス番号（0, 1）

* p_reg
ステータスレジスタ設定データバッファ

Return Values

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_spi_if.h にプロトタイプ宣言されています。

Description

p_reg に設定された値はステータスレジスタ 2 に書き込まれます。ただし、使用する Serial NOR Flash memory によっては、機能が割り当てられている場合や、Reserved bit の場合があります。詳細は、使用する Serial NOR Flash memory のデータシートを参照してください。

ステータス レジスタ 2
Bits 7: Complement Block Protection
See the specification of the Flash memory.
Bit 6: Lock Security Register 3
1: Security Register page-3 cannot be erased/programmed.
0: Security Register page-3 is not locked
Bit 5: Lock Security Register 2
1: Security Register page-2 cannot be erased/programmed.
0: Security Register page-2 is not locked
Bit 4: Lock Security Register 1
1: Security Register page-1 cannot be erased/programmed.
0: Security Register page-1 is not locked
Bits 1: Quad Enable
1: HOLD and WP are I/O pins
0: HOLD and WP function normally.

このユーザ API 関数を呼び出す前に、ステータスレジスタ 2 の値を読み取り、上書きが必要なビットのみ値を変更してください。 処理終了後、ステータスレジスタ 2 を読み出し、書き込んだ値が正しいことを確認してください。
本ユーザ API が正常終了した場合、Serial Flash memory 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_FLASH_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、Serial Flash memory はその処理を受け付けません。
R_FLASH_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。

Example

```
#define FLASH_WR_BUSY_WAIT (uint32_t)(40)     /* 40 * 1ms = 40ms */

flash_spi_status_t   ret    = FLASH_SPI_SUCCESS;
uint32_t             loop_cnt = 0;
uint8_t              gStat;
uint8_t              Reg;

ret = R_FLASH_SPI_Read_Status2(FLASH_SPI_DEV0, &gStat]);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

Reg = (gStat | 0x10);
ret = R_FLASH_SPI_Write_Status2(FLASH_SPI_DEV0, &Reg);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

loop_cnt = FLASH_WR_BUSY_WAIT;
mode = FLASH_SPI_MODE_REG_WRITE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
   if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);     /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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。

R_FLASH_SPI_Write_Status3()

この関数はステータスレジスタ 3 への書き込みに使用されます。Dialog Semiconductor Plc の AT25QF ファミリ シリアル NOR フラッシュ メモリ用の専用 API 関数です。

Format

```
flash_spi_status_t R_FLASH_SPI_Write_Status3 (
    uint8_t devno,
    flash_spi_reg_info_t * p_reg
)
```

Parameters

devno
デバイス番号（0, 1）

* p_reg
ステータスレジスタ設定データバッファ

Return Values

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_spi_if.h にプロトタイプ宣言されています。

Description

p_reg に設定された値はステータスレジスタ 3 に書き込まれます。ただし、使用する Serial NOR Flash memory によっては、機能が割り当てられている場合や、Reserved bit の場合があります。詳細は、使用する Serial NOR Flash memory のデータシートを参照してください。

ステータス レジスタ 3
Bit 7: Reserved
Bits 6 to 5: DRV[1:0] (Drive strength)
11: Auto (7 pF based on VCC level)
10: 50% (15 pF)
01: 75% (22 pF)
00: 100% (30 pF)
Bits 4 to 0: Reserved

このユーザ API 関数を呼び出す前に、ステータスレジスタ 3 の値を読み取り、上書きが必要なビットのみ値を変更してください。 処理終了後、ステータスレジスタ 3 を読み出し、書き込んだ値が正しいことを確認してください。
本ユーザ API が正常終了した場合、Serial Flash memory 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_FLASH_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、Serial Flash memory はその処理を受け付けません。
R_FLASH_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。

Example

```
#define FLASH_WR_BUSY_WAIT (uint32_t)(40)    /* 40 * 1ms = 40ms */

flash_spi_status_t   ret   = FLASH_SPI_SUCCESS;
uint32_t             loop_cnt = 0;
uint8_t              gStat;
uint8_t              Reg;

ret = R_FLASH_SPI_Read_Status3(FLASH_SPI_DEV0, &gStat]);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

Reg = (gStat | 0x10);
ret = R_FLASH_SPI_Write_Status3(FLASH_SPI_DEV0, &Reg);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

loop_cnt = FLASH_WR_BUSY_WAIT;
mode = FLASH_SPI_MODE_REG_WRITE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
   if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);    /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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。

R_FLASH_SPI_Set_4byte_Address_Mode()

アドレスモードを 4Byte アドレスモードに設定する際に使用する関数です。Macronix International Co., Ltd 社製 MX25L/MX66L/MX25R family serial NOR Flash memory 専用の API です。

Format

```
flash_spi_status_t R_FLASH_SPI_Set_4byte_Address_Mode(
    uint8_t devno
)
```

Parameters

devno

デバイス番号（0, 1）

Return Values

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_spi_if.h にプロトタイプ宣言されています。

Description

＜Macronix International Co., Ltd 社製 MX25L/MX66L/MX25R family serial NOR Flash memory の場合＞
EN4B（0xb7）コマンドを発行し、コンフィグレーションレジスタの 4BYTE bit に 1 を設定します。

Example

```
flash_spi_status_t    ret = FLASH_SPI_SUCCESS;

ret = R_FLASH_SPI_Set_4byte_Address_Mode(FLASH_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。

R_FLASH_SPI_Read_Security()

セキュリティレジスタを読み出す際に使用する関数です。Macronix International Co., Ltd 社製 MX25L/MX66L/MX25R family serial NOR Flash memory 専用の API です。

Format

```
eepr_status_t R_FLASH_SPI_Read_Security
    uint8_t devno,
    uint8_t * p_scur
)
```

Parameters

devno

デバイス番号（0, 1）

** p_scur*

セキュリティレジスタ格納バッファ（サイズ 1 バイト）

Return Values

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_spi_if.h にプロトタイプ宣言されています。

Description

セキュリティレジスタを読み出し、p_scur に格納します。

p_scur には下記情報が格納されます。ただし、使用する Serial NOR Flash memory によっては、機能が割り当てられている場合や、Reserved bit の場合があります。詳細は、使用する Serial NOR Flash memory のデータシートを参照してください。

Bit 7: WPSEL
　1: Individual mode
　0: Normal WP mode
Bit 6: E_FAIL
　1: Erase failed
　0: Erase succeed
Bit 5: P_FAIL
　1: Program failed
　0: Program succeed
Bit 4: Reserved
Bit 3: ESB (Erase Suspend Bit)
　1: Erase Suspended
　0: Erase is not suspended
Bit 2: PSB (Program Suspend Bit)
　1: Program Suspended
　0: Program is not suspended
Bit 1: LDSO (Indicate if lock-down)
　1: Lock-down (Cannot program/erase OTP)
　0: Not lock-down
Bit 0: Secured OTP indicator
　1: Factory lock
　0: Non-factory lock.

P_FAIL=1 の場合、次のプログラム処理が成功すると 0 クリアされます。
E_FAIL=1 の場合、次の消去処理が成功すると 0 クリアされます。

Example

```
flash_spi_status_t    ret = FLASH_SPI_SUCCESS;
uint8_t               scur = 0;

ret = R_FLASH_SPI_Read_Security(FLASH_SPI_DEV0, &dcur);
```

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。

R_FLASH_SPI_Read_Data_Security_Page()

この関数はセキュリティレジスタからのデータ読み出しに使用されます。Dialog Semiconductor Plc の AT25QF ファミリ シリアル NOR フラッシュ メモリ用の専用 API 関数です。

Format

```
flash_spi_status_t R_FLASH_SPI_Read_Data_Security_Page(
    uint8_t devno,
    flash_spi_info_t * p_flash_spi_info
)
```

Parameters

devno

デバイス番号（0, 1）

** p_flash_spi_info*

Serial Flash memory 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 1～256 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

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_spi_if.h にプロトタイプ宣言されています。

Description

セキュリティレジスタの指定されたアドレスから指定されたバイト数のデータを読み取り、そのデータを p_data に格納します。
最大読み取りアドレスは、ページサイズ - 1 です。
ロールオーバーによる読み出しはできません。最終アドレスの読み出し後、一度処理を完了させて、再度アドレスを設定し直してから、本ユーザ API をコールしてください。
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、FLASH_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

```
flash_spi_status_t   ret = FLASH_SPI_SUCCESS;
flash_spi_info_t     Flash_Info_R;
uint32_t             buf2[128/sizeof(uint32_t)];
                                /* the buffer boundary (4-byte unit) */

Flash_Info_R.addr    = 0x1000;/* Security Register 1 Address */
Flash_Info_R.cnt     = 32;
Flash_Info_R.p_data  = (uint8_t *)&buf2[0];
ret = R_FLASH_SPI_Read_Data_Security_Page(FLASH_SPI_DEV0, &Flash_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。
この API 関数は、QSPIX メモリ マップド モードでは省略されます。

R_FLASH_SPI_Write_Data_Security_Page()

この関数は、1 ページ単位でセキュリティレジスタページにデータを書き込むために使用されます。Dialog Semiconductor Plc の AT25QF ファミリ シリアル NOR フラッシュ メモリ用の専用 API 関数です。

Format

```
flash_spi_status_t R_FLASH_SPI_Write_Data_Security_Page(
    uint8_t devno,
    flash_spi_info_t * p_flash_spi_info
)
```

Parameters

devno

デバイス番号（0, 1）

** p_flash_spi_info*

Serial Flash memory 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

```
FLASH_SPI_SUCCESS       /* 正常終了した場合 */
FLASH_SPI_ERR_PARAM     /* パラメータ異常の場合 */
FLASH_SPI_ERR_HARD      /* ハードウェアエラーの場合 */
FLASH_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_flash_spi_if.h にプロトタイプ宣言されています。

Description

指定されたアドレスから開始して、p_data 内の指定されたバイト数 (最大サイズ 1 ページまで) のデータをセキュリティレジスタ ページに書き込みます。
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。
セキュリティレジスタ ページへの書き込みは、ロックされていない場合にのみ可能です。ロックされたページに書き込むことはできません。エラーFLASH_SPI_ERR_OTHER を返します。
最大書き込みバイト数 (cnt) の設定値は、セキュリティレジスタのページサイズの容量です。
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 FLASH_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

```
#define FLASH_PP_BUSY_WAIT (uint32_t)(3)     /* 3 * 1ms = 3ms */

flash_spi_status_t   ret = FLASH_SPI_SUCCESS;
flash_spi_info_t     Flash_Info_W;
uint32_t             buf1[128/sizeof(uint32_t)];
                                /* the buffer boundary (4-byte unit) */
uint32_t             loop_cnt = 0;

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

do
{
   ret = R_FLASH_SPI_Write_Data_Security_Page(FLASH_SPI_DEV0, &Flash_Info_W);
   if (FLASH_SPI_SUCCESS > ret)
   {
      /* Error */
   }

   loop_cnt = FLASH_PP_BUSY_WAIT;
   mode = FLASH_SPI_MODE_PROG_POLL;
   do
   {
      /* FLASH is busy.
         User application can perform other processing while flash is busy. */

         ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
         if (FLASH_SPI_SUCCESS_BUSY != ret)
         {
            /* FLASH is ready or error. */
            break;
         }
         loop_cnt--;
         wait_timer(0, 1);    /* 1ms */
   }
   while (0 != loop_cnt);
}
while (0 != Flash_Info_W.cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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。

R_FLASH_SPI_Quad_Enable()

Quad モードを許可する際に使用する関数です。MX25L/MX66L/MX25R family serial NOR flash memory と Dialog Semiconductor Plc 社製の AT25QF family serial NOR flash memory 専用の API です。

Format

```
flash_spi_status_t R_FLASH_SPI_Quad_Enable(
    uint8_t devno
)
```

Parameters

devno

デバイス番号（0, 1）

Return Values

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_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の Quad Enable (QE) bit を 1 に設定し、Quad モードにします。
Quad モードで動作させる場合、事前に本関数をコールしてください。
処理終了後、ステータスレジスタを読み出して、QE bit を確認してください。
Quad Enable (QE) bit は、Non-volatile bit です。一度 Quad モードに設定した後、Quad モードを禁止にする場合、R_FLASH_SPI_Quad_Disable()を実行してください。
本ユーザ API が正常終了した場合、Serial Flash memory 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_FLASH_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、Serial Flash memory はその処理を受け付けません。
R_FLASH_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 3-5 を参照してください。

Example

```
#define FLASH_WR_BUSY_WAIT (uint32_t)(40)     /* 40 * 1ms = 40ms */

flash_spi_status_t      ret    = FLASH_SPI_SUCCESS;
uint32_t                loop_cnt = 0;
flash_spi_poll_mode_t   mode;

ret = R_FLASH_SPI_Quad_Enable(FLASH_SPI_DEV0);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

loop_cnt = FLASH_WR_BUSY_WAIT;
mode = FLASH_SPI_MODE_REG_WRITE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
   if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);     /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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。

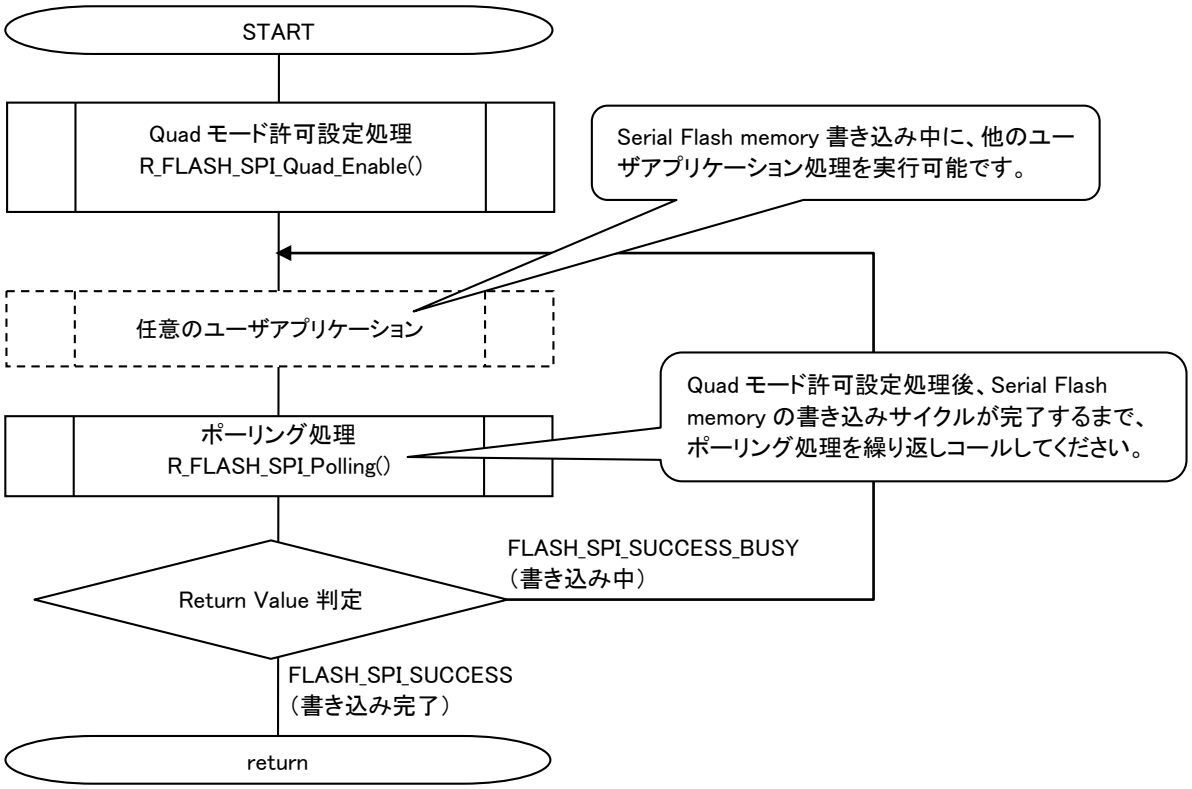


図 3-5　R_FLASH_SPI_Quad_Enable()の処理例

R_FLASH_SPI_Quad_Disable()

Quad モードを禁止する際に使用する関数です。MX25L/MX66L/MX25R family serial NOR flash memory と Dialog Semiconductor Plc 社製の AT25QF family serial NOR flash memory 専用の API です。

Format

```
flash_spi_status_t R_FLASH_SPI_Quad_Disable(
    uint8_t devno
)
```

Parameters

devno

デバイス番号（0, 1）

Return Values

FLASH_SPI_SUCCESS	*/* 正常終了した場合 */*
FLASH_SPI_ERR_PARAM	*/* パラメータ異常の場合 */*
FLASH_SPI_ERR_HARD	*/* ハードウェアエラーの場合 */*
FLASH_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_flash_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の Quad Enable (QE) bit を 0 に設定し、Quad モードキャンセルします。
処理終了後、ステータスレジスタを読み出して、QE bit を確認してください。
Quad Enable (QE) bit は、Non-volatile bit です。一度 Quad モードに設定した後、Quad モードを禁止にする場合、本ユーザ API を実行してください。
本ユーザ API が正常終了した場合、Serial Flash memory 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_FLASH_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、Serial Flash memory はその処理を受け付けません。
R_FLASH_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 3-6 を参照してください。

Example

```
#define FLASH_WR_BUSY_WAIT (uint32_t)(40)     /* 40 * 1ms = 40ms */

flash_spi_status_t      ret    = FLASH_SPI_SUCCESS;
uint32_t                loop_cnt = 0;
flash_spi_poll_mode_t   mode;

ret = R_FLASH_SPI_Quad_Disable(FLASH_SPI_DEV0);
if (FLASH_SPI_SUCCESS > ret)
{
   /* Error */
}

loop_cnt = FLASH_WR_BUSY_WAIT;
mode = FLASH_SPI_MODE_REG_WRITE_POLL;
do
{
   /* FLASH is busy.
      User application can perform other processing while flash is busy. */

   ret = R_FLASH_SPI_Polling(FLASH_SPI_DEV0, mode);
   if (FLASH_SPI_SUCCESS_BUSY != ret)
   {
      /* FLASH is ready or error. */
      break;
   }
   loop_cnt--;
   wait_timer(0, 1);     /* 1ms */
}
while (0 != loop_cnt);

if ((0 == loop_cnt) || (FLASH_SPI_SUCCESS > ret))
{
   /* 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。

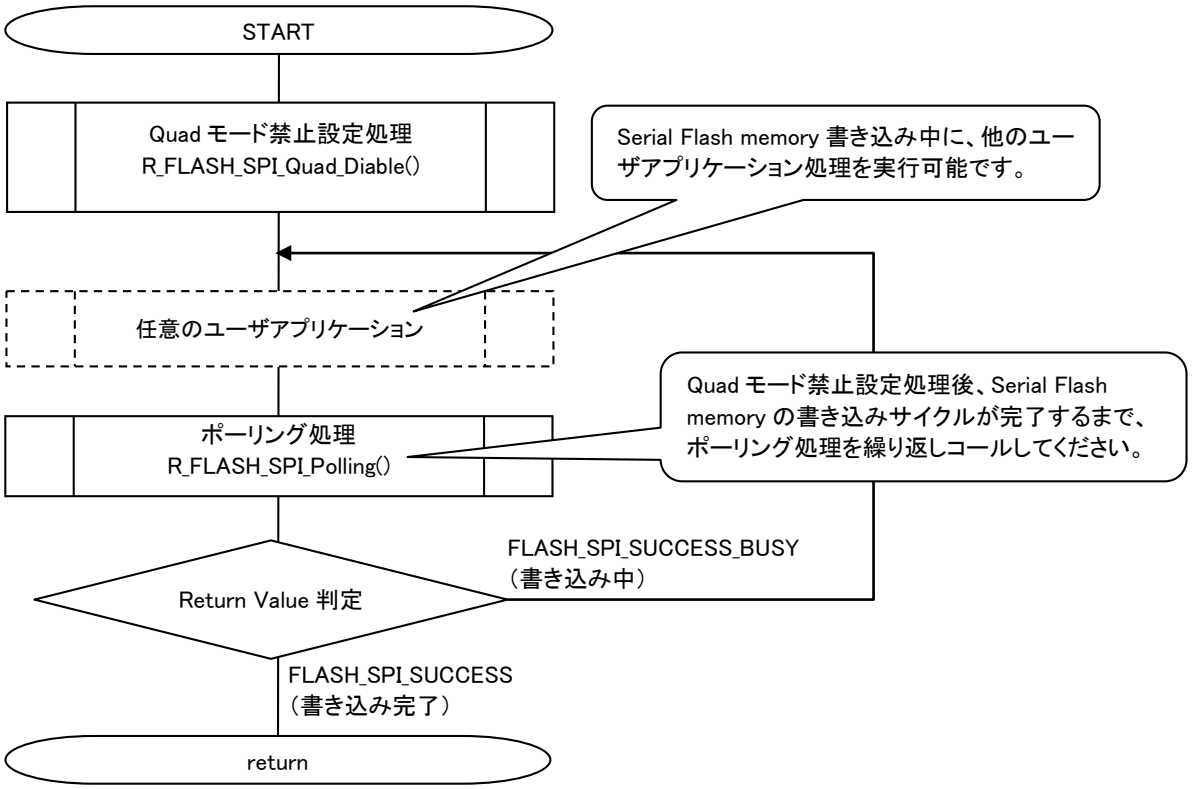


図 3-6　R_FLASH_SPI_Quad_Disable()の処理例

R_FLASH_SPI_GetVersion()

Serial Flash memory 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_FLASH_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_flash_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_FLASH_SPI_GetVersion();
```

Special Notes:
なし

R_FLASH_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

```
flash_spi_status_t R_FLASH_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

FLASH_SPI_SUCCESS　　/ 正常終了した場合 */*

Properties

r_flash_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 Flash memory 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_FLASH_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)

flash_spi_status_t ret = FLASH_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_FLASH_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_flash_spi_config.h の#define FLASH_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“‘n 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。

R_FLASH_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_FLASH_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_flash_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)

flash_spi_status_t   ret = FLASH_SPI_SUCCESS;
flash_spi_info_t     Flash_Info_W;
uint32_t             buf1[128/sizeof(uint32_t)];
                                /* the buffer boundary (4-byte unit) */

Flash_Info_W.addr    = 0;
Flash_Info_W.cnt     = 32;
Flash_Info_W.p_data  = (uint8_t *)&buf1[0];
ret = R_FLASH_SPI_Write_Data_Page(FLASH_SPI_DEV0, &Flash_Info_W);
if (FLASH_SPI_SUCCESS != ret)
{
   /* Set last error log to buffer. */
   R_FLASH_SPI_Log(USER_DRIVER_ID, USER_LOG_MAX, USER_LOG_ADR_MAX);
   R_FLASH_SPI_Close(FLASH_SPI_DEV0);
}
```

Special Notes:

別途 LONGQ FIT モジュールを組み込んでください。

r_flash_spi_config.h の#define FLASH_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。

R_FLASH_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_FLASH_SPI_1ms_Interval(void)

Parameters
なし

Return Values
なし

Properties
r_flash_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_FLASH_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. デモプロジェクト

デモプロジェクトには、FIT モジュールとそのモジュールが依存するモジュール（例：r_bsp）を使用する main()関数が含まれます。本 FIT モジュールには以下のデモプロジェクトが含まれます。

4.1 rx65n_rsk_flash_spi_sample, rx65n_rsk_flash_spi_sample_gcc

これは、Renesas RSKRX65N ボード用に作成された FLASH SPI FIT モジュールのサンプルアプリケーションです。 このプログラムは、API を使用して、クロック同期式シングルマスタ制御ソフトウェアである QSPI FIT モジュールを通してスレーブとして MX25L シリアルフラッシュメモリを制御し、マスタデバイスとして動作する方法を示します。

設定と実行

1. チャネル 0 のドライバサポートが r_qspi_smstr_rx_config.h で有効になっていることを確認します。
   #define QSPI_SMSTR_CFG_CH0_INCLUDED
2. e2studio デバッガを使用して、このサンプルアプリケーションをビルドし、RSK ボードにダウンロードします。
3. e2studio で Renesas Virtual Debug Console ビューを選択して、printf 情報を表示します。
4. デバッガでアプリケーションを実行します。
5. デバッグコンソールウィンドウに表示される FLASH SPI モジュールのバージョン番号とシリアルフラッシュメモリの ID を確認します。
6. 「Success!」がデバッグコンソールウィンドウに表示されます。
7. いずれかの操作が失敗した場合は、「Failed.」と表示されます。デバッグコンソールウィンドウに表示されます。

対応ボード

RSKRX65N

4.2　rx671_ek_flash_spi_sample, rx671_ek_flash_spi_sample_gcc

これは、Renesas EKRX671 ボード用に作成された FLASH SPI FIT モジュールのサンプルアプリケーションです。 このプログラムは、API を使用して、AT25QF64 シリアルフラッシュメモリをクロック同期シングルマスタ制御ソフトウェアを通してスレーブとして制御および使用する方法を示します。QSPIX メモリマップドモードでは、マスタデバイスとして動作します。

設定と実行

1. チャネル 0 のドライバサポートが r_qspix_rx_config.h で有効になっていることを確認します。
   #define QSPIX_CFG_USE_CH0　(1)
2. e2studio デバッガを使用して、このサンプルアプリケーションをビルドし、EK ボードにダウンロードします。
3. e2studio で Renesas Virtual Debug Console ビューを選択して、printf 情報を表示します。
4. デバッガでアプリケーションを実行します。
5. デバッグコンソールウィンドウに表示される FLASH SPI モジュールのバージョン番号とシリアルフラッシュメモリの ID を確認します。
6. 「Success!」がデバッグコンソールウィンドウに表示されます。
7. いずれかの操作が失敗した場合は、「Failed.」と表示されます。デバッグコンソールウィンドウに表示されます。

対応ボード

EK-RX671

5. 付録

5.1 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 5.1 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 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 5.2 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 5.3 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 5.4　動作確認環境 (Rev.3.03)

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 2021-07 IAR Embedded Workbench for Renesas RX 4.20.01
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 V3.03.00 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 8.03.00.202102 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 4.20.01 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.3.03
使用ボード	Renesas Starter Kit+ for RX671（型名：RTK55671xxxxxxxxxx）

表 5.5　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）

表 5.6　動作確認環境 (Rev.3.20)

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 2023-01 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 8.03.00.202202 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.3.20
使用ボード	Evaluation Kit+ for RX671（型名：RTK5EK671xxxxxxxxxx）

5.7　動作確認環境 (Rev.3.30)

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 2023-04 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 V3.05.00 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 8.03.00.202305 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.3.30
使用ボード	Renesas Starter Kit+ for RX65N（型名：RTK500565Nxxxxxx） Evaluation Kit+ for RX671（型名：RTK5EK671xxxxxxxxxx）

6. 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
2.32	2015.06.15	—	初版発行
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。
		24	2.9 ソフトウェアの追加方法　を更新した。
		36	Parameters に op_mode の説明を追加。
		37	Example に op_mode の処理を追加。
		38	Parameters に op_mode の説明を追加。
		40	Example に op_mode の処理を追加。
2.34	2017.07.31	20	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	動作確認に使用した MCU タイトル RX72T グループを追加。
		2	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。
		6-9	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。 R01AN2325JJ：“・”削除。 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
		16	1.5.3 ソフトウェア構成 図 1-8 を更新。
		17	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 1-9 を更新。
		20	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。

Rev.	発行日	改訂内容	
		ページ	ポイント
3.00	2019.02.20	21-22	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。 FLASH_SPI_CFG_DEVx_DRVIF_CH_NO FLASH_SPI_CFG_DEVx_MODE FLASH_SPI_CFG_DEVx_DMAC_CH_NO_Tx FLASH_SPI_CFG_DEVx_DMAC_CH_NO_Rx FLASH_SPI_CFG_DEVx_DMAC_INIT_PRIORITY_LEVEL_Tx FLASH_SPI_CFG_DEVx_DMAC_INIT_PRIORITY_LEVEL_Rx FLASH_SPI_CFG_DEVx_BR FLASH_SPI_CFG_DEVx_BR_WRITE_DATA FLASH_SPI_CFG_DEVx_BR_READ_DATA
		24	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
		2	「対象コンパイラ」を追加
		2	「関連ドキュメント」に R01AN1723 と R01AN1826 を削除
		6	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。
		17	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
		21-22	2.8 コードサイズ　を追加
		26	2.13 for 文、while 文、do while 文についてを追加
		33	3.4 R_FLASH_SPI_Set_Write_Protect()　Example の誤記を修正
		40	3.7 R_FLASH_SPI_Write_Data_Page()　Example の誤記を修正
		43	3.8 R_FLASH_SPI_Erase()　Example の誤記を修正
		52	3.13 R_FLASH_SPI_Write_Configuration()　Example の誤記を修正
		58	3.16 R_FLASH_SPI_Quad_Enable()　Example の誤記を修正
		61	R_FLASH_SPI_Quad_Disable()　Example の誤記を修正
		69	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。
		6	1.2 API の概要　「注 1」の誤記を修正
		21	2.8 コードサイズ　を更新
		23	2.10 ソフトウェアの追加方法　を更新
		27-68	API 説明ページの「Reentrant」項目を削除
		69	4.1 動作確認環境　を更新
		70	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
3.02	2020.12.10	プログラム	ソフトウェア不具合のため、FLASH_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.Serial Flash memory 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 FLASH_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
		プログラム	ソフトウェア不具合のため、FLASH_SPI FIT モジュールを修正 ■内容 SC のコンポーネントで r_flash_spi のデバイス容量を 1G-bit に設定すると、ビルドエラーが発生する。 ■発生条件 SC のコンポーネントで r_flash_spi のデバイス容量を 1G-bit に設定してビルドする。 ■対策 FLASH_SPI FIT モジュール Rev3.02 をご使用ください。
		プログラム	ソフトウェア不具合のため、FLASH_SPI FIT モジュールを修正 ■内容 RX72M/RX72N/RX66N で、r_flash_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 FLASH_SPI_CS_DEV0_CFG_PORTNO 若しくは FLASH_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 FLASH_SPI FIT モジュール Rev3.02 をご使用ください。

Rev.	発行日	改訂内容	
		ページ	ポイント
3.03	2021.11.30	1	「RX671 グループ（QSPIX）」を追加。
		5	概要に「QSPIX FIT モジュール」を追加。
		7	1.3.1 FIT モジュール関連アプリケーションノートの内容を更新。
		7	「1.4 Serial Flash Memory モジュールの使用方法」を追加。
		16	フローチャートを修正。
		17	2.2 ソフトウェアの要求に「r_qspix_rx」を追加。
		21-22	2.8 コードサイズ　を更新。
		70	4.1 動作確認環境　を更新。
3.10	2022.06.30	19	2.6 コンパイル時の設定 新規マクロを追加 #define FLASH_SPI_CS_DEVx_CFG_PORTNO #define FLASH_SPI_CS_DEVx_CFG_BITNO
		21	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
		70	「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。
3.20	2023.03.16	1, 6, 21, 31-52, 76, 79	AT25QF641B-SHB のサポートを追加。
		1, 5, 8, 18	RSCI FIT モジュールを追加。
		19	2.6 コンパイル時の設定：新規マクロを追加 #define FLASH_SPI_CFG_DEVx_AT25QF
		22	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。
		33-36, 61-67, 71-75	3 API 関数： 新しい API 関数を追加 R_FLASH_SPI_Read_Status2() R_FLASH_SPI_Read_Status3() R_FLASH_SPI_Write_Status() R_FLASH_SPI_Write_Status2() R_FLASH_SPI_Write_Status3() R_FLASH_SPI_Read_Data_Security_Page() R_FLASH_SPI_Write_Data_Security_Page()
		89	「4.1 動作確認環境」： Rev.3.20 に対応する表を追加。
		43	3 API 関数、の説明を追加 R_FLASH_SPI_Read_Data()の Special Notes を追記。
		プログラム	QSPIX の間接アクセスモードで AT25QF641B-SHB のサポートを追加 RSCI および QSPIX メモリマップモードのサポートを追加 サポートされていないその他デバイスに関連する処理を削除。

Rev.	発行日	改訂内容	
		ページ	ポイント
3.30	2023.06.15	15	「1.6.3 ソフトウェア構成」に、「 AT25QF family」を追加。
		25	「2.10 ソフトウェアの追加方法」から FIT configurator の記述を削除
		88, 89	「4 デモプロジェクト」を追加。
		92	「5.1 動作確認環境」： Rev.3.30 に対応する表を追加。
		プログラム	FIT Configurator の説明を削除しました。 デモプロジェクトの更新と追加

製品ご使用上の注意事項

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

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

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します