RXファミリ

R01AN2029JJ0144
Rev.1.44
Mar 01, 2025

USB Peripheral Mass Storage Class Driver (PMSC) Firmware Integration Technology

要旨

本アプリケーションノートでは、Firmware Integration Technology(FIT）を使用した USB Peripheral Mass Storage Class Driver(PMSC)について説明します。本モジュールは USB Basic Host and Peripheral Driver（USB-BASIC-FW FIT モジュール）と組み合わせることで動作します。以降、本モジュールを USB PMSC FIT モジュールと称します。

対象デバイス

RX65N/RX651 グループ
RX64M グループ
RX71M グループ
RX66T グループ
RX72T グループ
RX72M グループ
RX66N グループ
RX72N グループ
RX671 グループ

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

関連ドキュメント

1. USB Revision 2.0 Specification
2. USB Mass Storage Class Specification Overview Revision 1.1
3. USB Mass Storage Class Bulk-Only Transport Revision 1.0
   【http://www.usb.org/developers/docs/】
4. RX64M グループユーザーズマニュアル ハードウェア編 (ドキュメント No.R01UH0377)
5. RX71M グループユーザーズマニュアル ハードウェア編 (ドキュメント No.R01UH0493)
6. RX65N/RX651 グループユーザーズマニュアル ハードウェア編 (ドキュメント No. R01UH0590)
7. RX65N/RX651-2M グループユーザーズマニュアル ハードウェア編
   (ドキュメント No. R01UH0659)
8. RX66T グループユーザーズマニュアル ハードウェア編 (ドキュメント No. R01UH0749)
9. RX72T グループユーザーズマニュアル ハードウェア編 (ドキュメント No. R01UH0803)
10. RX72M グループユーザーズマニュアル ハードウェア編 (ドキュメント No. R01UH0804)
11. RX66N グループユーザーズマニュアル ハードウェア編 (ドキュメント No. R01UH0825)
12. RX72N グループユーザーズマニュアル ハードウェア編 (ドキュメント No. R01UH0824)
13. RX671 グループユーザーズマニュアル ハードウェア編 (ドキュメント No. R01UH0824)
14. USB Basic Host and Peripheral Driver Firmware Integration Technology アプリケーションノート
    (ドキュメント No. R01AN2025)

— ルネサス エレクトロニクスホームページ
【http://japan.renesas.com/】
— USB デバイスページ
【http://japan.renesas.com/prod/usb/】

目次

1. 概要 3
2. ソフトウェア構成 4
3. API情報 5
4. クラスドライバ概要 9
5. デバイスクラスドライバ（PDCD） 10
6. API 11
7. コンフィグレーション（r_usb_pmsc_config.h) 12
8. コンフィグレーションファイル（RI600V4使用時のみ) 14
9. メディアドライバインタフェース 15
10. アプリケーションの作成方法 25

1. 概要

USB PMSC FIT モジュールは、USB-BASIC-FW FIT モジュールと組み合わせることで、Peripheral Mass Storage Class Driver(PMSC)として動作します。PMSC は、USB マスストレージクラスの Bulk-Only Transport(BOT)プロトコルで構築されています。USB ペリフェラルコントロールドライバ、メディアドライバと組み合わせることで、BOT 対応のストレージ機器として USB ホストと通信を行うことができます。

以下に、本モジュールがサポートしている機能を示します。

- BOTプロトコルによるストレージコマンド制御
- USBホストからのマスストレージデバイスクラスリクエストに対する応答

1.1 必ずお読みください

このドライバを使ってアプリケーションプログラムを作成する場合は、USB Basic Host and Peripheral Driver Firmware Integration Technology アプリケーションノート(ドキュメント No.R01AN2025)を参照いただきますようお願いします。このアプリケーションノートは、パッケージ内の"reference_documents"フォルダにあります。

1.2 制限事項

1. USB Host から送信されるマスストレージクラスコマンド(GetMaxLun)に対し、本ドライバは値 0 を返します。
2. 本ドライバがサポートするセクタサイズは 512 のみです。

1.3 注意事項

1. 本ドライバは、USB 通信動作を保証するものではありません。システムに適用される場合は、お客様における動作検証はもとより、多種多様なデバイスに対する接続確認を実施してください
2. ストレージ領域として使用するメディアを制御するメディアドライバ関数はお客様において実装いただく必要があります。

1.4 用語一覧

APL	:	Application program
BOT	:	Bulk Only Transport
Non-OS	:	USB Driver for OS-less
DDI	:	Device Driver Interface
PCD	:	Peripheral Control Driver of USB-BASIC-FW
PCI	:	PCD Interface
PMSCD	:	Peripheral Mass Storage USB Class Driver (PMSCF + PCI + DDI)
PMSCF	:	Peripheral Mass Storage Class Function
PMSDD	:	Peripheral Mass Storage Device Driver (ATAPI driver)
RSK	:	Renesas Starter Kits
RTOS	:	USB Driver for the real-time OS
USB-BASIC-FW	:	USB Basic Host and Peripheral Driver

1.5 USB PMSC FIT モジュール

本モジュールは、r_usb_basic を使用したプロジェクトに組み込む必要があります。プロジェクトに組み込み後、API を使用することで USB の H/W 制御を行います。

2. ソフトウェア構成

USB PMSC FIT モジュールは、PMSCD と PMSDD で構成されています。
PMSCD は、BOT プロトコル制御及びデータ送受信を行う PMSCF、PMSDD に対するインタフェース関数群（DDI）、PCD に対するインタフェース関数群（PCI）で構成されています。
PMSDD は、PCD を介してホストとの BOT プロトコル通信を行います。PMSDD では、PMSCD から受け取ったストレージコマンドを解析し実行します。
Figure 2-1にモジュール構成、Table 2-1にモジュール機能概要を示します。

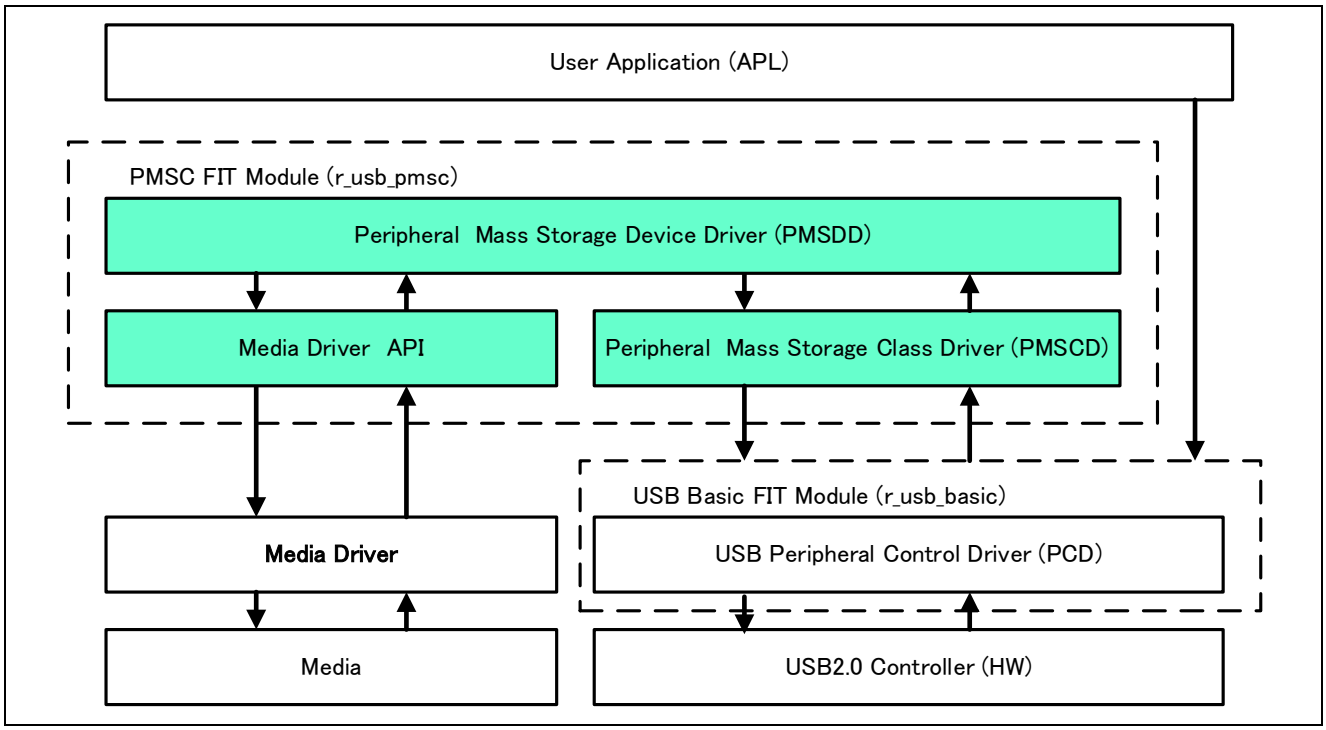

Figure 2-1 ソフトウェア構成図

Table 2-1 各モジュール機能概要

モジュール名	機能概要
PMSDD	マスストレージデバイスドライバ ・ PMSCD からのストレージコマンドの処理を行う ・ Media driver を介して Media へのアクセスを行う
DDI	PMSDD-PMSCD 間のインタフェース関数
PMSCF	マスストレージクラスドライバ ・ BOT プロトコルデータの制御、クラスリクエストの対応を行う ・ CBW の解析、データ送受信を行う ・ PMSDD/PCD との連携し CSW を生成する
PCI	PMSCD - PCD 間のインタフェース関数
PCD	USB Peripheral H/W 制御ドライバ

3. API 情報

本ドライバの API はルネサスの API の命名基準に従っています。

3.1 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。

- USB

3.2 ソフトウェアの要求

このドライバは以下のパッケージに依存しています。

- r_bsp
- r_usb_basic

3.3 動作確認環境

このドライバの動作確認環境を以下に示します。

Table 3-1 動作確認環境

項目	内容
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V.3.07.00 (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 8.3.0.202411 (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 version 5.10.1
リアルタイム OS	FreeRTOS V.10.0.0 RI600V4
エンディアン	リトルエンディアン / ビッグエンディアン
モジュールのリビジョン	Rev.1.44
使用ボード	Renesas Starter Kits for RX64M Renesas Starter Kits for RX71M Renesas Starter Kits for RX65N, Renesas Starter Kits for RX65N-2MB Renesas Starter Kits for RX72T Renesas Starter Kits for RX72M Renesas Starter Kits for RX72N Renesas Starter Kits for RX671
ホスト環境	下記の OS に接続し動作確認を行っています。 1. Windows® 10

3.4 使用する割り込みベクタ

このドライバが使用する割り込みベクタを以下に示します。

Table 3-2 使用する割り込みベクター覧

デバイス	割り込みベクタ

RX64M RX71M	USBI0 割り込み(ベクタ番号: 189, 割り込み要因番号 : 62, 選択型割り込み B) USB D0FIFO0 割り込み(ベクタ番号: 34) / USB D1FIFO0 割り込み(ベクタ番号: 35) USBR0 割り込み(ベクタ番号:90)
	USBAR 割り込み(ベクタ番号: 94) USB D0FIFO2 割り込み(ベクタ番号: 32) / USB D1FIFO2 割り込み(ベクタ番号: 33)
RX65N RX651 RX72M RX72N RX66N	USBI0(GROUPB)割り込み(ベクタ番号: 185, 割り込み要因番号 : 62, 選択型割り込み B) USB D0FIFO0 割り込み(ベクタ番号: 34) / USB D1FIFO0 割り込み(ベクタ番号: 35) USBR0 割り込み(ベクタ番号:90)
RX66T RX72T	USBI0 割り込み(ベクタ番号: 174) / USBR0 割り込み(ベクタ番号: 90) USB D0FIFO0 割り込み(ベクタ番号: 34) / USB D1FIFO0 割り込み(ベクタ番号: 35)
RX671	USBI0 割り込み(ベクタ番号: 185, 割り込み要因番号 : 62, 選択型割り込み B) USB D0FIFO0 割り込み(ベクタ番号: 34) / USB D1FIFO0 割り込み(ベクタ番号: 35) USBR0 割り込み(ベクタ番号:90)
	USBI1 割り込み(ベクタ番号: 182, 割り込み要因番号 : 63, 選択型割り込み B) USB D0FIFO1 割り込み(ベクタ番号: 36) / USB D1FIFO1 割り込み(ベクタ番号: 37)

3.5 ヘッダファイル

すべての API 呼び出しとそれをサポートするインタフェース定義は r_usb_basic_if.h と r_usb_pmsc_if.h に記載されています。

3.6 整数型

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

3.7 コンパイル時の設定

コンパイル時の設定については、「7. **コンフィグレーション (r_usb_pmsc_config.h)**」章および USB Basic Host and Peripheral Driver Firmware Integration Technology アプリケーションノート(ドキュメント No. R01AN2025)の「コンフィグレーション」章を参照してください。

3.8 ROM / RAM サイズ

本ドライバの ROM/RAM サイズを以下に示します。

1. CC-RX (最適化レベル: Default)

(1). Non-OS

	引数チェック実施時	引数チェック非実施時
ROM サイズ	24.5K バイト (Note 3)	24.1K バイト (Note 4)
RAM サイズ	10.0K バイト	10.0K バイト

(2). RTOS

a. FreeRTOS

	引数チェック実施時	引数チェック非実施時
ROM サイズ	37.9K バイト (Note 3)	37.5K バイト (Note 4)
RAM サイズ	27.5K バイト	27.5K バイト

b. RI600V4

	引数チェック実施時	引数チェック非実施時
ROM サイズ	39.7K バイト (Note 3)	39.3K バイト (Note 4)
RAM サイズ	16.2K バイト	16.2K バイト

2. GCC (最適化レベル: -O2)

	引数チェック実施時	引数チェック非実施時
ROM サイズ	30.1K バイト (Note 3)	29.7K バイト (Note 4)
RAM サイズ	9.8K バイト	9.8K バイト

3. IAR (最適化レベル: Medium)

	引数チェック実施時	引数チェック非実施時
ROM サイズ	23.8K バイト (Note 3)	23.3K バイト (Note 4)
RAM サイズ	8.5K バイト	8.5K バイト

[Note]

1. 上記のサイズには、BSP および USB Basic Driver の ROM/RAM サイズが含まれています。
2. 上記は V2 コアオプション指定時のサイズです。
3. 「引数チェック実施時」の ROM サイズは、r_usb_basic_config.h ファイル内の USB_CFG_PARAM_CHECKING 定義に対し USB_CFG_ENABLE を指定した時の値です。
4. 「引数チェック非実施時」の ROM サイズは、r_usb_basic_config.h ファイル内の USB_CFG_PARAM_CHECKING 定義に対し USB_CFG_DISABLE を指定した時の値です。
5. RAM サイズは､r_usb_pmsc_config.h ファイル内の USB_CFG_PMSC_TRANS_COUNT 定義に対し 8 (数値)を指定した時の値です。
6. RTOS には、リアルタイム OS の ROM/RAM サイズが含まれています。

3.9 引数

API 関数の引数に使用される構造体については、USB Basic Host and Peripheral Driver Firmware Integration Technology アプリケーションノート(ドキュメント No.R01AN2025)内の「構造体」の章を参照してください。

3.10 for 文、while 文、do while 文について

FIT 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。

3.11 モジュールの追加方法

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

(1) e² studio 上で Smart Configurator を使用して FIT モジュールを追加する場合

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

(2) e² studio 上で FIT Configurator を使用して FIT モジュールを追加する場合

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

(3) CS+上で Smart Configurator を使用して FIT モジュールを追加する場合

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

(4) CS+上で FIT モジュールを追加する場合

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

4. クラスドライバ概要

4.1 クラスリクエスト

本ドライバがサポートするクラスリクエストをTable 4-1に示します。

Table 4-1 クラスリクエスト

リクエスト	コード	説明
Bulk-Only Mass Storage Reset	0xFF	マスストレージデバイスと接続インタフェースのリセット
Get Max Lun	0xFE	デバイスがサポートする論理番号を通知

4.2 ストレージコマンド

本ドライバがサポートするストレージコマンドをTable 4-2に示します。下記以外のコマンドに対してはSTALL 応答あるいは CSW による FAIL エラーを返します。

Table 4-2 ストレージコマンド

コマンド名	コード	説明
TEST_UNIT_READY	0x00	ペリフェラル機器の状態確認
REQUEST_SENSE	0x03	直前のストレージコマンド実行結果のエラー情報取得
INQUIRY	0x12	論理ユニットのパラメータ情報取得
READ_FORMAT_CAPACITY	0x23	フォーマット可能な容量取得
READ_CAPACITY	0x25	論理ユニットの容量情報取得
READ10	0x28	データ読み出し
WRITE10	0x2A	データ書き込み
MODE_SENSE10	0x5A	論理ユニットのパラメータ取得

5. デバイスクラスドライバ（PDCD）

5.1 基本機能

PDCD の機能を以下に示します。

1. SFF-8070i（ATAPI）に対応
2. USB ホストからのマスストレージデバイスクラスリクエストに対する応答

5.2 BOT プロトコル概要

BOT（Bulk-Only Transport）とは、バルクイン/バルクアウトの 2 つの Endpoint のみを使用し、コマンド、データ、ステータス（コマンド処理の結果）を管理する転送プロトコルです。

USB 上で転送されるデータのうち、コマンドとステータスについては Command Block Wrapper（CBW）、Command Status Wrapper（CSW）の形式で転送を行います。BOT プロトコル概要をFigure 5-1に示します。

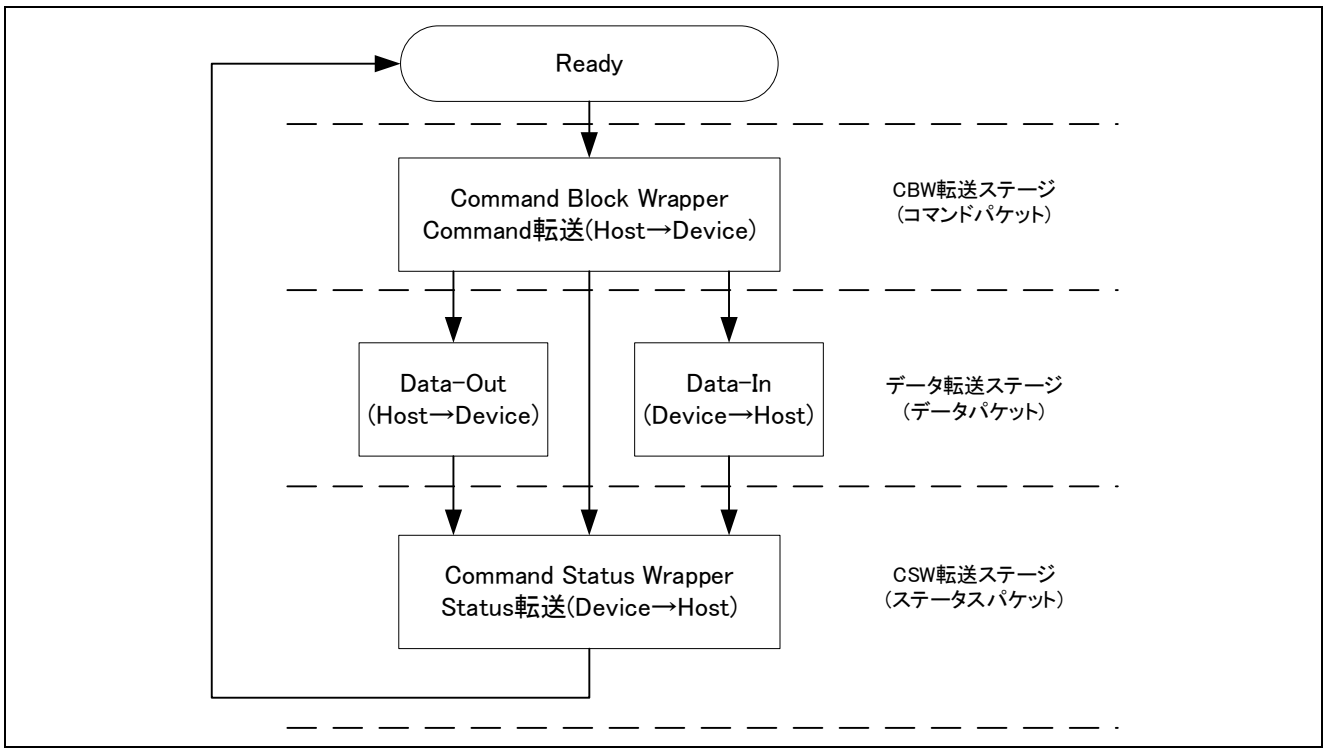

Figure 5-1 BOT プロトコル概要

6. API

アプリケーションプログラム内で使用する API については、USB Basic Host and Peripheral Driver Firmware Integration Technology アプリケーションノート(ドキュメント No.R01AN2025)内の「API」の章を参照してください。

7. コンフィグレーション (r_usb_pmsc_config.h)

お客様のシステムにあわせて以下の設定をお願いします。

[Note]

必ず r_usb_basic_config.h ファイルに対する設定もお願いします。r_usb_basic_config.h については、USB Basic Host and Peripheral Driver Firmware Integration Technology アプリケーションノート(ドキュメント No.R01AN2025)内の「コンフィグレーション」の章を参照してください。

1. 使用パイプ設定

Bulk IN, Bulk OUT 転送で使用するパイプ番号(PIPE1 から PIPE5)を指定してください。なお、USB_CFG_PMSC_BULK_IN と USB_CFG_PMSC_BULK_OUT に対し同じパイプ番号は指定しないでください。

```
#define     USB_CFG_PMSC_BULK_IN        パイプ番号 (USB_PIPE1 から USB_PIPE5)
#define     USB_CFG_PMSC_BULK_OUT       パイプ番号 (USB_PIPE1 から USB_PIPE5)
```

2. Inquiry コマンド応答データ設定

本ドライバは以下の各定義に指定したデータを Inquiry コマンドの応答データとして USB ホストに送信します。

(1). Vendor Information 設定

Inquiry コマンドの応答データである Vendor Information を指定してください。必ず 8 バイトのデータをダブルクォーテーションで括って指定してください。

```
#define     USB_CFG_PMSC_VENDOR     Vendor Information
```

例)

```
#define     USB_CFG_PMSC_VENDOR     "Renesas "
```

(2). Product Information 設定

Inquiry コマンドの応答データである Product Information を指定してください。必ず 16 バイトのデータをダブルクォーテーションで括って指定してください。

```
#define     USB_CFG_PMSC_PRODUCT    Product Information
```

例)

```
#define     USB_CFG_PMSC_PRODUCT    "Mass Storage    "
```

(3). Product Revision Level 設定

Inquiry コマンドの応答データである Product Revision Level を指定してください。必ず 4 バイトのデータをダブルクォーテーションで括って指定してください。

```
#define     USB_CFG_PMSC_REVISION   Product Revision Level
```

例)

```
#define     USB_CFG_PMSC_REVISION   "1.00"
```

3. 転送セクタ数設定

PCD(USB Peripheral Control Driver)に要求する 1 回のデータ転送の最大セクタサイズを指定してください。本ドライバは、PCD に対し"1 セクタ(512)×USB_CFG_PMSC_TRANS_COUNT "バイトの値を転送サイズとして指定します。

この値を大きくすることにより PCD に対するデータ転送要求回数が減るため転送速度性能が向上する可能性がありますが、"1 セクタ(512)×USB_CFG_PMSC_TRANS_COUNT "バイト分の RAM が消費されますのでご注意ください。

```
#define     USB_CFG_PMSC_TRANS_COUNT        転送セクタ数 (1 から 255)
```

例)

```
#define     USB_CFG_PMSC_TRANS_COUNT        8
```

8. コンフィグレーションファイル (RI600V4 使用時のみ)

RI600 を使用する場合、PMSC ドライバで使用する OS 資源を RI600 に登録する必要があります。以下の PMSC 関連の定義をコンフィグレーションファイルに追加してください。コンフィグレーションファイルの作成方法については、USB Basic Host and Peripheral Driver Firmware Integration Technology アプリケーションノート(ドキュメント No.R01AN2025)内の「**コンフィグレーションファイル(RI600V4 使用時のみ)**」の章を参照してください。

8.1 タスク定義

name	:	ID_USB_RTOS_PMSC_TSK
entry_address	:	usb_pstd_pmsc_task()
stack_size	:	512
initial_start	:	OFF
exinf	:	0

8.2 メールボックス定義

name	:	ID_USB_RTOS_PMSC_MBX
wait_queue	:	TA_FIFO
message_queue	:	TA_MFIFO

9. メディアドライバインタフェース

PMSC では、仕様の異なるメディアドライバへのアクセスを容易にするために共通のメディアドライバ API 関数を使用しています。

9.1 メディアドライバ API 関数

メディアドライバ API は、PMSC から呼び出され、ユーザによって実装されたメディアドライバ関数をコールします。本章では、メディアドライバ API 関数のプロトタイプと各関数の実装に必要な処理について説明します。

メディアドライバ API 関数一覧を示します。

Table 9-1 メディアドライバ API

メディアドライバ API	処理概要
R_USB_media_initialize	メディアドライバの初期化
R_USB_media_open	メディアドライバオープン
R_USB_media_close	メディアドライバクローズ
R_USB_media_read	メディアリード
R_USB_media_write	メディアライト
R_USB_media_ioctl	メディアデバイス特有のコントロール命令を処理

9.1.1 R_USB_media_initialize

メディアドライバ関数をメディアドライバに登録します。

形式

bool R_USB_media_initialize(media_driver_t * p_media_driver);

引数

p_meida_driver	メディアドライバ構造体領域へのポインタ

戻り値

TRUE	正常終了
FALSE	エラー発生

解説

ユーザによって実装されたメディアドライバ関数をメディアドライバに登録します。なお、ユーザアプリケーションプログラム内の初期化処理等において必ず本APIをコールしてください。

補足

1. 上記「引数」、「戻り値」および「解説」等に記載した内容をサポートするメディアドライバ関数をユーザにおいて実装する必要があります。
2. ユーザによって実装されたメディアドライバ関数の登録方法については、「9.3 ストレージメディアドライバの登録」を参照ください。
3. 本 API はデバイスのレジスタ初期化処理やデバイス上の動作を開始するものではありません。それらの処理は R_USB_media_open()関数で行います。
4. この PMSC は複数種類のメディアドライバ関数を登録するための機能をサポートしていません。

使用例

```
if (!R_USB_media_initialize(&g_ram_mediadriver))
{
        /* Handle the error */
}
result = R_USB_media_open();
if (USB_MEDIA_RET_OK != result)
{
        /* Process the error */
}
```

9.1.2 R_USB_media_open

メディアデバイスおよびメディアドライバの初期化処理を行います。

形式

usb_media_ret_t R_USB_media_open(void);

引数

--

戻り値

USB_MEDIA_RET_OK	正常終了
USB_MEDIA_RET_PARAERR	パラメータエラー
USB_MEDIA_RET_DEV_OPEN	デバイスがすでにオープン済
USB_MEDIA_RET_NOTRDY	デバイスが応答しないまたは存在しない
USB_MEDIA_RET_OP_FAIL	その他のエラー

解説

メディアデバイスおよびメディアドライバおよびの初期化を行い、メディアデバイスおよびメディアドライバをレディ状態にします。なお、ユーザアプリケーションプログラム内の初期化処理等において必ず本APIをコールしてください。

補足

1. 本関数を呼び出す前に R_USB_media_initialize()関数を呼び出す必要があります。
2. R_USB_media_close()関数をコールしない限り、R_USB_media_open()の呼び出し回数は 1 回のみです。なお、R_USB_media_close()関数を呼び出した後であれば、デバイス設定を初期状態に戻すために本関数を再び呼び出すことができます。
3. 上記「引数」、「戻り値」および「解説」等に記載した内容をサポートするメディアドライバ関数をユーザにおいて実装する必要があります。

使用例

```
if (!R_USB_media_initialize(&g_ram_mediadriver))
{
      /* Handle the error */
}

result = R_USB_media_open();
if (USB_MEDIA_RET_OK != result)
{
      /* Process the error */
}
```

9.1.3 R_USB_media_close

メディアドライバ用のリソースを解放し、ハードウェアを非アクティブな状態に戻します。

形式

usb_media_ret_t R_USB_media_close(void);

引数

--

戻り値

USB_MEDIA_RET_OK	正常終了
USB_MEDIA_RET_PARAERR	パラメータエラー
USB_MEDIA_RET_OP_FAIL	その他のエラー

解説

メディアドライバ用のリソースを解放し、ハードウェアを非アクティブな状態に戻します。

補足

1. 本関数を呼び出す前に R_USB_media_initialize()関数を呼び出す必要があります。
2. 上記「引数」、「戻り値」および「解説」等に記載した内容をサポートするメディアドライバ関数をユーザにおいて実装する必要があります。

使用例

```
result = R_USB_media_close();
if (USB_MEDIA_RET_OK != result)
{
    /* Process the error */
}
```

9.1.4 R_USB_media_read

メディアデバイスからデータブロックを読み出します。

形式

usb_media_ret_t R_USB_media_read(uint8_t *p_buf, uint32_t lba, uint8_t count);

引数

p_buf	メディアデバイスから読みだされたデータを格納する領域へのポインタ
lba	読み出し開始論理ブロックアドレス
count	読み出しブロック数 (セクタ数)

戻り値

USB_MEDIA_RET_OK	正常終了
USB_MEDIA_RET_PARAERR	パラメータエラー
USB_MEDIA_RET_NOTRDY	デバイスがレディ状態ではありません
USB_MEDIA_RET_OP_FAIL	その他のエラー

解説

メディアデバイスからデータブロックのリード処理を行います。(第 2 引数で指定された LBA(Logical BLock Address)から第 3 引数(count)に指定されたブロック数分のデータブロックをリードします。)

リードデータは第 1 引数(p_buf)で指定した領域に格納されます。

補足

1. 本関数を呼び出す前に R_USB_media_initialize()関数を呼び出す必要があります。
2. 上記「引数」、「戻り値」および「解説」等に記載した内容をサポートするメディアドライバ関数をユーザにおいて実装する必要があります。

使用例

```
result = R_USB_media_read(&buffer, lba, 1);
if (USB_MEDIA_RET_OK != result)
{
/* Process the error */
}
```

9.1.5 R_USB_media_write

メディアデバイスにデータブロックを書き込みます。

形式

usb_media_ret_t R_USB_media_write(uint8_t *p_buf, uint32_t lba, uint8_t count);

引数

p_buf	メディアデバイスに書き込むデータを格納する領域へのポインタ
lba	書き込み開始論理ブロックアドレス
count	書き込みブロック数 (セクタ数)

戻り値

USB_MEDIA_RET_OK	正常終了
USB_MEDIA_RET_PARAERR	パラメータエラー
USB_MEDIA_RET_NOTRDY	デバイスがレディ状態ではありません
USB_MEDIA_RET_OP_FAIL	その他のエラー

解説

メディアデバイスへデータブロックのライト処理を行います。(第 2 引数で指定された LBA(Logical BLock Address)へ第 3 引数(count)に指定されたブロック数分のデータブロックをライトします。)

ライトデータは第 1 引数(p_buf)が示す領域に格納してください。

補足

1. 本関数を呼び出す前に R_USB_media_initialize()関数を呼び出す必要があります。
2. 上記「引数」、「戻り値」および「解説」等に記載した内容をサポートするメディアドライバ関数をユーザにおいて実装する必要があります。

使用例

```
result = R_USB_media_write(&buffer, lba, 1);
if (MEDIA_RET_OK != result)
{
/* Process the error */
}
```

9.1.6 R_USB_media_ioctl

メディアドライバ等の情報を取得します。

形式

usb_media_ret_t R_USB_media_ioctl(ioctl_cmd_t command, void *p_data);

引数

command	コマンドコード
p_data	メディア情報を格納する領域へのポインタ

戻り値

USB_MEDIA_RET_OK	正常終了
USB_MEDIA_RET_PARAERR	パラメータエラー
USB_MEDIA_RET_NOTRDY	デバイスがレディ状態ではありません
USB_MEDIA_RET_OP_FAIL	その他のエラー

解説

本関数はメディアドライバ固有のコマンドを引数(command)に指定し、メディアドライバからの戻り情報を取得する処理を行います。

PMSCではメディアドライバに対するコマンドコードとして、以下のコマンドを使用しています。

MEDIA_IOCTL_GET_NUM_BLOCKS	メディア領域のブロック数
MEDIA_IOCTL_GET_BLOCK_SIZE	1 ブロックサイズ

補足

1. 本関数を呼び出す前に R_USB_media_initialize()関数を呼び出す必要があります。
2. ユーザは引数(command)に指定するコマンドコードを新たに定義することができます。
3. 上記「引数」、「戻り値」および「解説」等に記載した内容をサポートするメディアドライバ関数をユーザにおいて実装する必要があります。

使用例

```
uint32_t num_blocks;
uint32_t block_size;
uint64_t capacity;.

result = R_USB_media_ioctl(MEDIA_IOCTL_GET_NUM_BLOCKS, (void *)&num_blocks);
result = R_USB_media_ioctl(MEDIA_IOCTL_GET_BLOCK_SIZE, (void *)&block_size);

capacity = (uin64_t)block_size * (uint64_t)num_blocks;
```

9.2 構造体/列挙型定義

メディアドライバ API 関数で使用される構造体/列挙型を以下に示します。
これらは r_usb_media_driver_if.h ファイルで定義されています。

9.2.1 usb_media_driver_t (構造体)

usb_media_driver_t は、ユーザによって実装されたメディアドライバ関数へのポインタを保持する構造体です。
以下に、usb_media_driver_t 構造体を示します。

```
typedef struct media_driver_s
{
        usb_media_open_t    pf_media_open;    /* オープン関数のポインタ */
        usb_media_close_t   pf_media_close;   /* クローズ関数のポインタ */
        usb_media_read_t    pf_media_read;    /* リード関数のポインタ */
        usb_media_write_t   pf_media_write;   /* ライト関数のポインタ */
        usb_media_ioctl_t   pf_media_ctrl;    /* コントロール関数のポインタ */
} usb_media_driver_t
```

9.2.2 usb_media_ret_t (列挙型)

usb_media_ret_t には、メディアドライバ API が返す戻り値が定義されています。

```
typedef enum
{
        USB_MEDIA_RET_OK = 0,          /* 正常終了 */
        USB_MEDIA_RET_NOTRDY,          /* デバイスがレディ状態でない */
        USB_MEDIA_RET_PARERR,          /* パラメータエラー */
        USB_MEDIA_RET_OP_FAIL,         /* その他のエラー */
        USB_MEDIA_RET_DEV_OPEN,        /* デバイスは既にオープンしている */
} usb_media_ret_t
```

9.2.3 ioctl_cmd_t (列挙型)

ioctl_cmd_t には、R_USB_media_ioctl 関数に指定するコマンドコードが定義されています。

```
typedef enum
{
    USB_MEDIA_IOCTL_GET_NUM_BLOCKS,     /* 論理ブロック数取得 */
    USB_MEDIA_IOCTL_GET_BLOCK_SIZE,     /* 論理ブロックサイズ取得 */
} ioctl_cmd_t
```

[Note]

メディアドライバ実装にあたりユーザ独自のコマンドコードを追加する場合、上記 ioctl_cmd_t に追加してください。

9.3 ストレージメディアドライバの登録

PMSC のストレージメディアをフラッシュメモリなどの他のストレージメディアへ変更する場合、ユーザは、そのストレージメディアに対する読み出しまたは書き込みを行うためのメディアドライバ関数を実装し、メディアドライバ API に登録する必要があります。

以下に、シリアル SPI フラッシュのメディアドライバ関数登録手順を示します。

1. 登録するメディアドライバ関数の作成

ユーザがシリアル SPI フラッシュ用メディアドライバ関数として以下の関数を実装したものとします。

1. usb_media_ret_t spi_flash_open (void)
2. usb_media_ret_t spi_flash_close (void)
3. usb_media_ret_t spi_flash_read(uint8_t *p_buf,uint32_t lba, uint8_t count)
4. usb_media_ret_t spi_flash_write(uint8_t *p_buf,uint32_t lba, uint8_t count)
5. usb_media_ret_t spi_flash_ioctl(ioctl_cmd_t ioctl_cmd,void * ioctl_data)

2. メディアドライバ関数のメディア API への登録

(1). シリアル SPI フラッシュ用の usb_media_driver_t 構造体を定義してください。

この構造体の各メンバには、該当するメディアドライバ関数へのポインタを設定してください。

```
struct media_driver_t   g_spi_flash_mediadriver =
{
      &spi_flash_open,
      &spi_flash_close,
      &spi_flash_read,
      &spi_flash_write,
      &spi_flash_ioctl
};
```

(2). アプリケーションプログラムで、上記の usb_media_driver_t 構造体へのポインタを R_USB_media_initialize 関数(API)の引数に指定し、初期化処理を行ってください。

```
== アプリケーションプログラム ==
R_USB_media_initialize(& g_spi_flash_mediadriver );
```

上記手順をおこなうことにより、メディアドライバが呼び出すメディアドライバ関数としてシリアル SPI フラッシュ関数が登録されます。

9.4 ストレージメディアドライバの実装

ご使用になるストレージメディアに対応するメディアドライバ関数をお客様において実装いただく必要があります。実装したメディアドライバ関数は、PMSC から「9.1 **メディアドライバAPI関数**」に記載された API を経由してコールされます。

[Note]

メディアドライバ関数の実装に必要な処理については、「9.1 **メディアドライバAPI関数**」に記載された各 API 仕様を参考してください。

9.5 メディアドライバ関数のプロトタイプ宣言

メディアドライバ関数のプロトタイプ宣言を以下に示します。

```
1.  usb_media_ret_t  (*media_open_t) (uint8_t);                                  /* Open 関数型  */
2.  usb_media_ret_t  (*media_close_t)(uint8_t);                                  /* Close 関数型  */
3.  usb_media_ret_t (*media_read_t)(uint8_t, uint8_t*, uint32_t, uint8_t);       /* Read 関数型  */
4.  usb_media_ret_t (*media_write_t)(uint8_t, uint8_t*, uint32_t, uint8_t);      /* Write 関数型  */
5.  usb_media_ret_t  (*media_ioctl_t)(uint8_t, ioctl_cmd_t, void *);            /* Control 型関数  */
```

10. アプリケーションの作成方法

USB Basic Host and Peripheral Driver Firmware Integration Technology アプリケーションノート(ドキュメント No.R01AN2025)内の「**アプリケーションプログラムの作成方法**」の章を参照してください。

[Note]

ユーザアプリケーションプログラム内の初期化処理等において必ず R_USB_media_initialze 関数(API)および R_USB_media_open 関数(API)をコールしてください。

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/inquiry

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。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Aug 1, 2014	—	初版発行
1.10	Dec 26, 2014	—	対象デバイスに RX71M を追加
1.11	Sep 30, 2015	—	対象デバイスに RX63N と RX631 を追加
1.20	Sep 30, 2016	—	1. 対象デバイスに RX65N/RX651 を追加 2. DMA 転送をサポート 3. USB Host and Peripheral Interface Driver アプリケーションノート(ドキュメント No.R01AN3293JJ)に対応
1.21	Mar 16, 2017	—	Technical Update(発行番号: TN-RX*-A172A/J)に対応しました。
1.22	Sep 30, 2017	—	RX65N/RX651-2M をサポート
1.23	Mar 31, 2018	—	Smart Configurator に対応しました。
1.24	Dec 28, 2018	—	RTOS をサポート
1.25	Apr 16, 2019	—	対象デバイスに RX66T/RX72T を追加
1.26	May 31, 2019	—	1. GCC/IAR コンパイラをサポートしました。 2. 対象デバイスから RX63N を削除しました。
1.27	Jul 31, 2019	—	対象デバイスに RX72M を追加
1.30	Mar 1, 2020	—	1. リアルタイム OS(uITRON:RI600V4)をサポートしました。 2. 対象デバイスに RX72N/RX66N を追加
1.31	Mar 1, 2021	—	対象デバイスに RX671 を追加
1.42	Sep 29, 2023	—	3.10 章を追加
1.44	Mar 01, 2025	—	Disclaimer を変更。

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。

© 2025 Renesas Electronics Corporation. All rights reserved.