RX ファミリ

仮想 EEPROM モジュール Firmware Integration Technology

R01AN4142JU0111
Rev.1.11
2018.10.18

要旨

仮想 EEPROM（以下、VEE）モジュールは、RX マイコンに搭載しているデータフラッシュを EEPROM のように使用することができます。

VEE モジュールは、ルネサス RX コンパイラのみに対応します。

動作確認デバイス

このモジュールは次のデバイスをサポートします。

- フラッシュタイプ 1、フラッシュタイプ 3 またはフラッシュタイプ 4 の何れかのデータフラッシュを搭載した全ての RX マイコン。フラッシュタイプについては、「フラッシュモジュール Firmware Integration Technology」（R01AN2184JU）を参照してください。

関連ドキュメント

- Firmware Integration Technology ユーザーズマニュアル（R01AN1833JU）
- ボードサポートパッケージモジュール Firmware Integration Technology（R01AN1685JJ）
- フラッシュモジュール Firmware Integration Technology（R01AN2184JU）
- e² studio に組み込む方法 Firmware Integration Technology（R01AN1723JU）
- CS+に組み込む方法 Firmware Integration Technology（R01AN1826JJ）

目次

1. 概要

この VEE モジュールは、EEPROM の基本機能をエミュレートします。共通レコードおよび参照データ（製品のアセンブリまたはテスト時にプログラム済み）の読み出し/書き込みをサポートします。レコードは、固定長または可変長として設定できます。アプリケーションのライフタイムにおけるセグメントの消去回数は維持され、いつでもアクセスできます。ウェアレベリングはこの VEE モジュールによって自動的に処理されます。

この VEE モジュールがサポートするフラッシュタイプは、1、3、および 4 です（RX111/113/130、RX231/24T/24U、RX64M/71M、RX65N-2M はサポートしますが、データフラッシュを持たない RX110/23T/65N-1M はサポートしません）。フラッシュタイプについては、「フラッシュモジュール Firmware Integration Technology」（R01AN2184JU）を参照してください。

1.1 機能

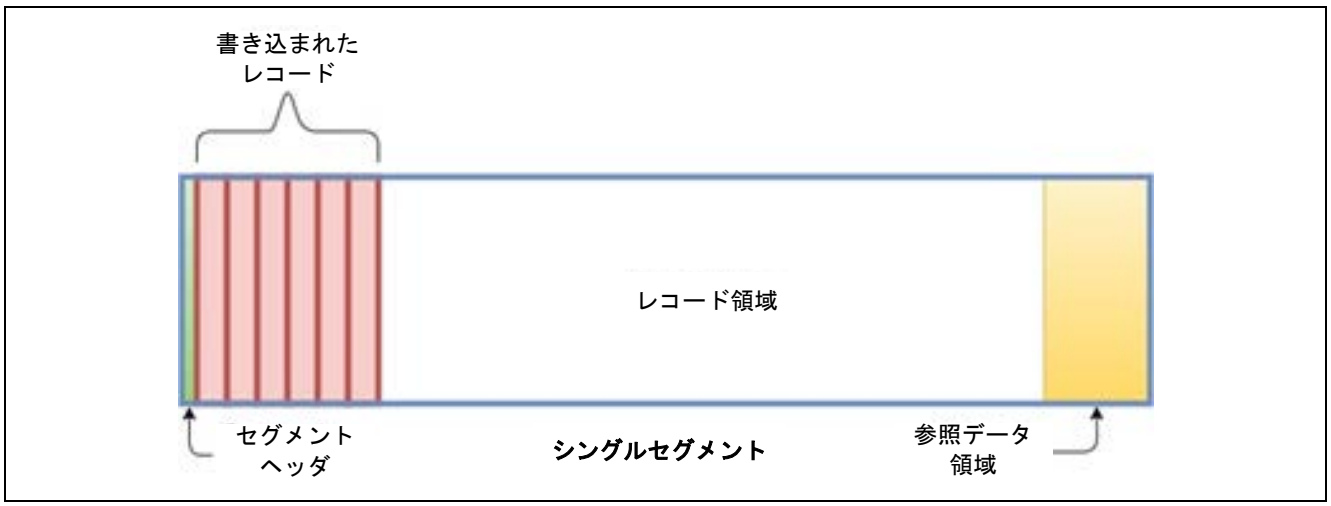
VEE モジュールがサポートする機能は以下のとおりです。

- ユーザ定義レコードのデータフラッシュへの書き込みと読み出し
- レコードは固定長または可変長として設定可能
- ウェアレベリングは自動処理
- アセンブリまたはテスト時にプログラムする校正データなどの参照データを保存
- 参照データを動作中に更新可能

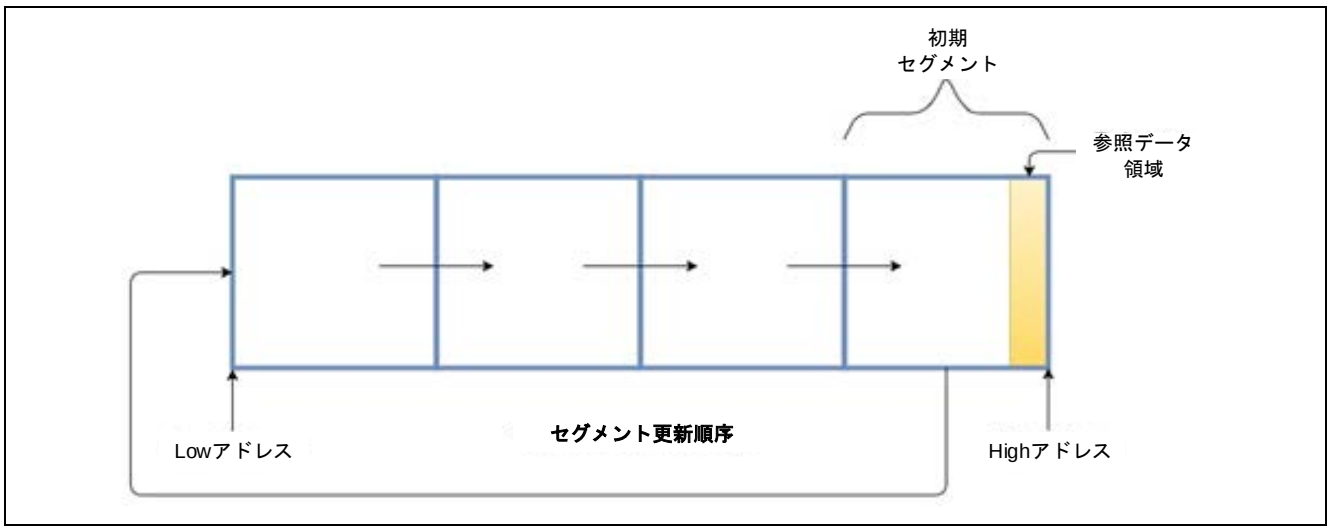
1.2 データフラッシュセグメンテーション

ウェアレベリングは、レコードの更新時にレコードが保存されるデータフラッシュ内の場所を変更することで処理されます。このレコードの物理的な場所における変更は、ユーザが意識する必要はありません。特定のレコード ID が更新されると、データフラッシュ内の次の未使用の場所に書き込まれ、その場所は後で素早く参照できるように RAM に保存されます。それらのレコードの最新バージョンのみが、定期的にデータフラッシュ内の次の空白セグメントにコピーされます。

データフラッシュ領域は、「r_vee_rx_config.h」で指定したセグメント数に均等サイズで分割されます。セグメント数のデフォルトは各 MCU のデータフラッシュサイズ/4096(4K)です。一度にアクティブなセグメントは 1 つのみです。1 つのセグメントは、セグメントヘッダとレコード領域、参照データ領域から構成されます。レコードと参照データは、アクティブなセグメントが埋まるまで書き込まれます。



セグメントにレコードまたは参照データを追加する十分なスペースがない場合、リフレッシュが実行されます。このプロセスにより、各 ID の最新レコードのほか、最新バージョンの参照データがある場合も次のセグメントにコピーされます。最初に VEE モジュールの初期化を実行すると、参照データの有無の設定に関係なく最後のセグメントがアクティブとしてマークされます。

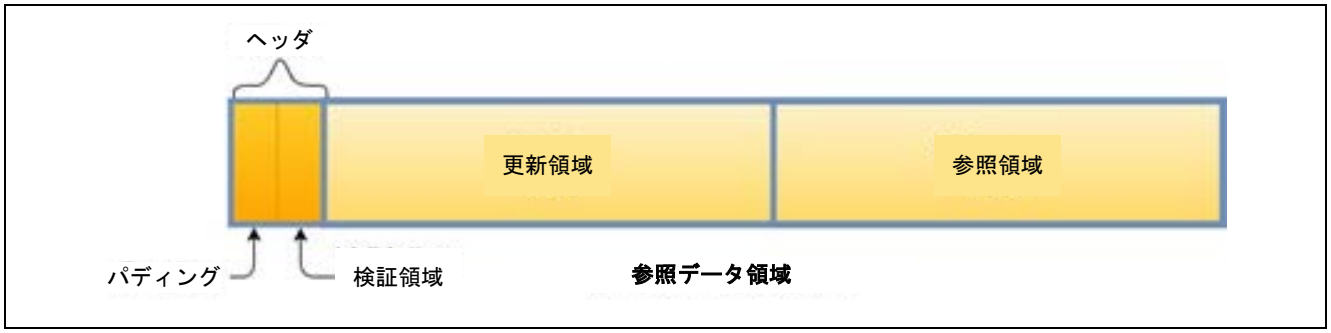


1.3　レコードフォーマット

VEE は、「r_vee_rx_config.h」でレコードを固定長または可変長に設定できます。デフォルトでは、レコードは可変長として設定されています。可変長レコードではレコードのデータ長を指定するヘッダが先頭に付加されています。トレイラには、VEE モジュール専用の検証コードが格納されます。CRC や ECC のようなエラーチェックが必要な場合、ユーザがモジュールに渡すレコードデータにそのエラーチェックを含めてください。レコードデータには 500 バイトの制限が設けられています。

1.4　参照データ領域

VEE は、参照データの有無を「r_vee_rx_config.h」で設定できます。デフォルトでは、参照データはなし（参照データのサイズが“0”）に設定されています。参照データをあり（参照データのサイズが“1”以上）に設定すると、最初の参照データは、データフラッシュの最後に配置されます。ただし、ユーザが VEE モジュールを使用せずに参照データを書き込む場合は、データフラッシュの最後に配置してください。参照データが更新されると同じサイズの更新領域にデータが書かれ、ヘッダが付加されます。



レコードと同様に、検証領域には VEE モジュール専用の検証コードが格納されます。CRC や ECC のようなエラーチェックが必要な場合、ユーザがモジュールに渡す参照データにそのエラーチェックを含めてください。参照データには 508 バイトの制限が設けられています。

2. API 情報

本 FIT モジュールは、下記の条件で動作を確認しています。

2.1 ハードウェアの必要条件

このモジュールでは、MCU が次の周辺機能をサポートする必要があります。

- データフラッシュ

2.2 ソフトウェアの必要条件

このモジュールは、次の FIT パッケージを活用します。

- ルネサスボードサポートパッケージ v3.60（オプション）
- FIT フラッシュモジュール v3.30（このバージョンの FIT フラッシュモジュールには、VEE モジュールで必要な「Close」関数が追加されています）

2.3 制限事項

- このコードはリエントラント（再入）不可で、複数の同時関数呼び出しをブロックします。

2.4 サポートされているツールチェーン

このモジュールは、次のツールチェーインで動作確認しています。

- ルネサス RX ツールチェーン v2.07.00

2.5 ヘッダファイル

すべての API 呼び出しとそれに対応するインタフェース定義は、「r_vee_rx_if.h」に記載されます。このファイルは、VEE モジュールの API を使用するすべてのファイルでインクルードする必要があります。

ビルド時の設定オプションは、「r_vee_rx_config.h」ファイルで選択または定義されます。

2.6 整数型

このプロジェクトでは、コードを分かりやすくして移植性を高めるため、ANSI C99 の「Exact width integer types」を採用しています。整数型は stdint.h で定義されます。

2.7 コンフィグレーションの概要

このモジュールのコンフィグレーションは、r_vee_rx_config.h ヘッダファイルで行います。各コンフィグレーション項目は、このファイル内のマクロ定義で表されます。コンフィグレーション可能な各項目は、表 2.1 で説明しています。

表 2.1　VEE 一般コンフィグレーション設定

r_vee_rx_config.h ファイルにあるコンフィグレーション項目		
定義	デフォルト値	説明
VEE_CFG_PARAM_CHECKING_ENABLE	1	1 に設定すると、パラメータチェックを有効にします。 0 に設定すると、パラメータチェックは無効です。
VEE_CFG_NUM_SEGMENTS	MCU_DATA_FLASH_SIZE_BYTES / 4096	データフラッシュ内に必要なセグメント数を指定します（最小は 2）。セグメントの数が少ないほどリフレッシュ回数も減りますが、リフレッシュが完了するまでの時間（消去時間）は長くなります。
VEE_CFG_REF_DATA_SIZE	0	データフラッシュの最後にある参照データのサイズを指定します。 0 は参照データなしを意味します。サイズは FLASH_DF_MIN_PGM_SIZE の倍数でなければなりません。
VEE_CFG_RECORD_FIXED_SIZE	0	レコードデータのサイズを指定します。 0 に設定すると、可変長レコードが使用されることを意味します。 固定長レコードを使用すると、速度と使用されるフラッシュスペースが最適化されます。サイズは FLASH_DF_MIN_PGM_SIZE の倍数でなければなりません。
VEE_CFG_RECORD_MAX_ID	16	ID は 0 から始まる連続した値でなければなりません。配列は、各 ID の最新レコードの場所とともに維持されます。使用する最大の ID を設定します。
VEE_CFG_REFRESH_BUF_SIZE	32	リフレッシュ処理の際、データをフラッシュセグメント間でコピーするには、内部バッファが必要です。バッファサイズの効率を最大限に高めるには、使用するレコードの最大サイズ+4 バイト（または可変長レコードの場合は+8 バイト）を指定します。サイズは FLASH_DF_MIN_PGM_SIZE の倍数でなければなりません。
VEE_CFG_FLASH_INT_PRIORITY	8	フラッシュ処理が BGO（割り込み）モードで実行されます。フラッシュ割り込み優先順位を 1（最低）から 15（最高）の間で設定します。
VEE_CFG_ALLOW_GETS_AFTER_FAIL	0	システムエラーの検出後、GetRecordPtr()または GetRefDataPtr（フラッシュの読み出しを実行）を呼び出す必要がある場合は 1 に設定します。 Close()および Control()の Get Status コマンドを除き、API 呼び出しを禁止するには 0 に設定します。

2.8　コードサイズ

コードサイズは、最適化レベル 2、および 2.4 サポートされているツールチェーンで指定される RXC ツールチェーンのサイズの最適化タイプに基づきます。

VEE ROM および RAM の使用量	
ROM の使用量: PARAM_CHECKING_ENABLE 1 > PARAM_CHECKING_ENABLE 0 VEE_CONFIG_REF_DATA_SIZE non-zero > VEE_CONFIG_REF_DATA_SIZE 0 VEE_CFG_RECORD_FIXED_SIZE 0 > VEE_CFG_RECORD_FIXED_SIZE non-zero VEE_CFG_ALLOW_GETS_AFTER_FAIL 1 > VEE_CFG_ALLOW_GETS_AFTER_FAIL 0	
最小サイズ	ROM：2,071 バイト
	RAM：97 バイト+VEE_CFG_REFRESH_BUF_SIZE + (VEE_CFG_RECORD_MAX_ID + 1) * 2
最大サイズ	ROM：2508 バイト
	RAM：97 バイト + VEE_CFG_REFRESH_BUF_SIZE + (VEE_CFG_RECORD_MAX_ID + 1) * 2

2.9　API データ型

API データ構造体は、「r_vee_rx_if.h」ファイル内に記載されます。詳しくは、3. API 関数を参照してください。

2.10　API の戻り値

API 関数の戻り値を示します。戻り値のタイプは「r_vee_rx_if.h」で定義されます。

```
typedef enum e_vee_err
{
    VEE_SUCCESS = 0,
    VEE_READY = 0,           // returned by VEE_CMD_GET_STATUS
    VEE_SUCCESS_RECOVERY,    // Open() succeeded; corrupted data erased
    VEE_ERR_CORRUPTION_FOUND,// (for driver internal use only)
    VEE_ERR_ALREADY_OPEN,    // Open() called again without a Close()in between
    VEE_ERR_ILLEGAL_CONFIG,  // invalid setting in vee or flash config.h file
    VEE_ERR_UNKNOWN_CMD,     // unrecognized Control() command
    VEE_ERR_NULL_PTR,        // missing argument pointer
    VEE_ERR_ILLEGAL_ARG,     // parameter value invalid
    VEE_ERR_FLASH_INIT_ERR,  // Flash FIT driver could not open properly
    VEE_ERR_BUSY,            // another VEE operation is still executing
    VEE_ERR_BUSY_REFRESH,    // a Refresh operation is in progress
    VEE_ERR_RECORD_NOT_FOUND,// no record has been written for specified ID
    VEE_ERR_OVERFLOW,        // segments too small to hold 1 record for each ID
    VEE_ERR_FLASH_PE_FAIL,   // should never happen; flash hardware failure
    VEE_ERR_TIMEOUT,         // should never happen; interrupts disabled by app
    VEE_ERR_UNKNOWN_WRITE    // should never happen; development error
} vee_err_t;
```

2.11　FIT VEE モジュールのプロジェクトへの追加

FIT モジュールをプロジェクトに追加する方法に関する詳細は、「FIT モジュールのプロジェクトへの追加」（R01AN1723JU）を参照してください。

2.11.1　ソースツリーおよびプロジェクトインクルードパスの追加

一般的に、FIT モジュールを追加する方法には以下の 3 通りがあります。

1. e² studio FIT ツールを使用する（File > New > Renesas FIT Module（v5.3.0 より前)、Renesas Views -> e2 ソリューションツールキット -> FIT Configurator（v5.3.0 以降）、または Smart Configurator を使用して作成されたプロジェクト（v5.3.0 以降）など）。これにより、モジュールおよびプロジェクトインクルードパスが追加されます。
2. e² studio で、プロジェクトコンテキストメニューから File > Import > General > Archive File の順に選択する。
3. .zip ファイルを Windows からプロジェクトディレクトリに直接解凍する。

2 または 3 の方法を使用する場合、インクルードパスを手動でプロジェクトに追加する必要があります。その手順は、e² studio でプロジェクトコンテキストメニューから Properties > C/C++ Build > Settings の順に選択し、ToolSettings タブで Compiler > Source を選択します。右のボックス内にある緑の「+」記号をクリックしてダイアログボックスを開き、インクルードパスを追加します。このボックスで Workspace ボタンをクリックし、表示されたプロジェクトツリー構造から必要なディレクトリを選択します。このモジュールに必要なディレクトリは以下のとおりです。

- ${workspace_loc:/${ProjName}/r_vee_rx
- ${workspace_loc:/${ProjName}/r_vee_rx/src
- ${workspace_loc:/${ProjName}/r_config

2.11.2　モジュール使用オプションの設定

¥r_config¥r_vee_rx_config.h には、VEE 固有のオプションがあり、編集が可能です。

デフォルト値が設定されたこのファイルの参考用コピー（編集不可）が ¥r_vee_rx¥ref¥r_vee_rx_config_reference.h に保存されています。

BGO（割り込み）処理を実行するには、FIT フラッシュモジュールのコンフィグレーションファイル ¥r_config¥r_flash_rx_config.h を設定する必要があります。

同様に、プロジェクト用デバッグコンフィグレーションでは、データフラッシュへの書き込みを許可する必要があります。（フラッシュモジュールのインストール手順およびコンフィグレーションの説明については、「フラッシュモジュール Firmware Integration Technology」（R01AN2184JU）アプリケーションノートを参照してください）。

3. API 関数

3.1 概要

この API には次の関数が含まれています。

関数	説明
R_VEE_Open()	VEE モジュールの内部構造体を初期化し、FIT フラッシュモジュールをオープンします。
R_VEE_WriteRecord ()	レコードをデータフラッシュに書き込みます。
R_VEE_GetRecordPtr()	所定のレコードの最新バージョンへのポインタを取得します。
R_VEE_WriteRefData()	新しい参照データを、データフラッシュ内の参照データ領域に書き込みます。
R_VEE_GetRefDataPtr()	最新の有効な参照データへのポインタを取得します。
R_VEE_Control()	特殊操作を実行します。
R_VEE_Close()	フラッシュモジュールと VEE モジュールを終了します。
R_VEE_GetVersion()	VEE モジュールのバージョンを返します。

3.2 R_VEE_Open()

VEE モジュールの内部構造体を初期化し、FIT フラッシュモジュールをオープンします。

Format

```
vee_err_t R_VEE_Open(void);
```

Parameters

なし

Return Values

VEE_SUCCESS:	*正常終了*
VEE_SUCCESS_RECOVERY:	*正常終了。ただし、破損したデータが検出されたため消去しました。*
VEE_ERR_BUSY:	*前回の API 呼び出しが引き続き実行中です。*
VEE_ERR_BUSY_REFRESH:	*セグメントのリフレッシュが進行中です。*
VEE_ERR_ALREADY_OPEN:	*この関数は既に呼び出されています。*
VEE_ERR_FLASH_INIT_ERR:	*フラッシュモジュールのオープンや設定でエラーが発生しました。*
VEE_ERR_TIMEOUT:	*VEE の外部で割り込みは無効です。*
VEE_ERR_FLASH_PE_FAIL:	*通常は発生しないエラー。付録を参照してください。*
VEE_ERR_UNKNOWN_WRITE:	*通常は発生しないエラー。付録を参照してください。*

Properties

プロトタイプは「r_vee_rx_if.h」ファイルにあります。

Description

VEE モジュールの内部構造体を初期化し、FIT フラッシュモジュールをオープンします。FIT フラッシュモジュールは、VEE をオープンする前に終了しておく必要があります。

戻り値 VEE_SUCCESS_RECOVERY は、VEE が破損したデータを検出したことを示します。このエラーの主な原因は、データフラッシュの書き込みまたは消去時のリセットまたはノイズなどが考えられます。その場合、自動でリフレッシュが実行され、部分的に書き込まれたデータは消失します。Control()の VEE_CMD_GET_LAST_ID_WRITTEN コマンドは、書き込みが成功した最後のレコード ID を示します。R_VEE_GetRefDataPtr()関数は、利用可能な最新の良好な参照データへのポインタを提供します。

Reentrant

不可

Example:

```
vee_err_t err;

err = R_VEE_Open();
if ((err != VEE_SUCCESS) || (err != VEE_SUCCESS_RECOVERY))
{
   while(1) ;  // fatal error
}
```

Special Notes:

なし

3.3　R_VEE_WriteRecord()

レコードをデータフラッシュに書き込みます。

Format

```
vee_err_t R_VEE_WriteRecord(uint16_t rec_id,
                            uint8_t *rec_data_ptr,
                            uint16_t num_bytes);
```

Parameters

rec_id

書き込むレコードの ID です。

rec_data_ptr

書き込むレコードデータへのポインタです。

num_bytes

書き込むデータの長さです。

Return Values

VEE_SUCCESS:	*書き込みは正常に開始されました。*
VEE_ERR_BUSY:	*前回の API 呼び出しが引き続き実行中です。*
VEE_ERR_BUSY_REFRESH	*セグメントのリフレッシュが進行中です。*
VEE_ERR_NULL_PTR:	*「rec_data_ptr」が NULL です。*
VEE_ERR_ILLEGAL_ARG:	*ID または「num_bytes」の値が無効です。*
VEE_ERR_OVERFLOW:	*通常は発生しないエラー。付録を参照してください。*
VEE_ERR_FLASH_PE_FAIL:	*通常は発生しないエラー。付録を参照してください。*
VEE_ERR_UNKNOWN_WRITE:	*通常は発生しないエラー。付録を参照してください。*

Properties

プロトタイプは「r_vee_rx_if.h」ファイルにあります。

Description

この関数は、「rec_data_ptr」で示した「num_bytes」分のデータをデータフラッシュに書き込みます。この関数はフラッシュ書き込みの開始直後に返します。書き込みが完了するまで、データバッファの内容を絶対に変更しないでください。これには、データバッファがローカル変数の場合に関数の呼び出しを終了する処理も含まれます（別の関数によってスタックが使用され、データバッファの内容が破損するおそれがあります）。この書き込みが完了するまでに、他の API を呼び出すと BUSY が返されます。レコードを書き込むためのレコードスペースが足りない場合、セグメントのリフレッシュプロセスが自動的に開始されます。リフレッシュプロセスの完了前に他の API を呼び出すと、BUSY REFRESH を返します。

Reentrant

不可

Example:

```
vee_err_t        err;
uint8_t          rec_data[TEMPERATURE_DATA_SIZE];
struct temp_data rec_data2;
uint16_t         rec_length;

/* Example: record data is an array of bytes */
err = R_VEE_WriteRecord(ID_TEMPERATURE, rec_data, TEMPERATURE_DATA_SIZE);

/* Example: record data has a structure format */
err = R_VEE_WriteRecord(ID_TEMPERATURE,
                        (uint8_t *)&rec_data2,
                         sizeof(rec_data2));
```

Special Notes:

リフレッシュが実行された場合、リフレッシュ中はロック状態となり、更新が完了するまで BUSY REFRESH を返します。

3.4　R_VEE_GetRecordPtr()

この関数は、指定された ID のレコードの最新バージョンへのポインタを取得します。

Format

```
vee_err_t R_VEE_GetRecordPtr(uint16_t rec_id,
                             uint8_t  **rec_data_ptr,
                             uint16_t *num_bytes_ptr);
```

Parameters

rec_id

検索対象レコードの ID です。

rec_data_ptr

レコードの最新バージョンへのポインタです。

num_bytes_ptr

レコード長が格納される変数です。

Return Values

VEE_SUCCESS:	*正常終了*
VEE_ERR_BUSY:	*前回の API 呼び出しが引き続き実行中です。*
VEE_ERR_BUSY_REFRESH:	*セグメントのリフレッシュが進行中です。*
VEE_ERR_NULL_PTR:	*「rec_data_ptr」または「num_bytes_ptr」が NULL です。*
VEE_ERR_ILLEGAL_ARG:	*ID 値が範囲外です。*
VEE_ERR_RECORD_NOT_FOUND:	*指定された ID のレコードは見つかりませんでした。*
VEE_ERR_OVERFLOW:	*通常は発生しないエラー。付録を参照してください。*
VEE_ERR_FLASH_PE_FAIL:	*通常は発生しないエラー。付録を参照してください。*
VEE_ERR_UNKNOWN_WRITE:	*通常は発生しないエラー。付録を参照してください。*

Properties

プロトタイプは「r_vee_rx_if.h」ファイルにあります。

Description

この関数は、「rec_id」で指定されたレコードの最新バージョンへのポインタを「rec_data_ptr」に設定します。また、「num_bytes_ptr」が示す変数にレコード長を格納します。

Reentrant

不可

Example:

```
vee_err_t err;
uint8_t          *rec_ptr;
struct temp_data *rec2_ptr;
uint16_t          rec_length;

/* Example: record is an array of bytes */
err = R_VEE_GetRecordPtr(ID_TEMPERATURE, &rec_ptr, &rec_length);

/* Example: record has a structure format */
err = R_VEE_GetRecordPtr(ID_TEMPERATURE, (uint8_t **)&rec2_ptr, &rec_length);
```

Special Notes:

フラッシュは同時に読み書きすることができません。フラッシュ書き込みを開始する前に「rec_data_ptr」によるデータの読み出しを推奨します。

3.5 R_VEE_WriteRefData()

新しい参照データを参照データ領域に書き込みます。

Format

```
vee_err_t R_VEE_WriteRefData(uint8_t *ref_data_ptr);
```

Parameters

ref_data_ptr

参照データ領域に書き込む参照データのポインタです。

Return Values

VEE_SUCCESS:	*書き込みは正常に開始されました。*
VEE_ERR_BUSY:	*前回の API 呼び出しが引き続き実行中です。*
VEE_ERR_BUSY_REFRESH:	*セグメントのリフレッシュが進行中です。*
VEE_ERR_NULL_PTR:	*「ref_data_ptr」が NULL です。*
VEE_ERR_ILLEGAL_CONFIG:	*r_vee_rx_config.h で設定されていない参照データです。*
VEE_ERR_OVERFLOW:	*通常は発生しないエラー。付録を参照してください。*
VEE_ERR_FLASH_PE_FAIL:	*通常は発生しないエラー。付録を参照してください。*
VEE_ERR_UNKNOWN_WRITE:	*通常は発生しないエラー。付録を参照してください。*

Properties

プロトタイプは「r_vee_rx_if.h」ファイルにあります。

Description

この関数は、「ref_data_ptr」で示した VEE_CFG_REF_DATA_SIZE バイト分のデータをデータフラッシュに書き込みます。この関数はフラッシュ書き込みの完了を待たずに戻ります。そのため、書き込みが完了するまで、データバッファの内容を絶対に変更しないでください。これには、バッファがローカル変数の場合に関数の呼び出しを終了する処理も含まれます（別の関数によってスタックが使用され、データバッファの内容が破損するおそれがあります）。この書き込みが完了するまで、他の API を呼び出すと BUSY が返されます。アクティブセグメントの更新領域にデータが存在する状態で、この関数を実行すると、リフレッシュプロセスが自動的に行われ、新しいアクティブセグメントの参照領域にデータが書き込まれます。リフレッシュプロセスの完了前に他の API を呼び出すと、BUSY REFRESH を返します。

Reentrant

不可

Example:

```
vee_err_t err;
uint8_t     ref_buf[VEE_CFG_REF_DATA_SIZE];
struct refdata  ref_type2;

/* Example: data is in byte array */
err = R_VEE_WriteRefData(ref_buf);

/* Example: data is in a structure */
err = R_VEE_WriteRefData((uint8_t *)&ref_type2);
```

Special Notes:

リフレッシュが実行された場合、リフレッシュ中はロック状態となり、更新が完了するまで BUSY REFRESH を返します。

3.6　R_VEE_GetRefDataPtr()

最新の参照データへのポインタを取得します。

Format

```
vee_err_t R_VEE_GetRefDataPtr(uint8_t **ref_data_ptr);
```

Parameters

ref_data_ptr

最新の参照データのポインタです。

Return Values

VEE_SUCCESS:	*正常終了*
VEE_ERR_BUSY:	*前回の API 呼び出しが引き続き実行中です。*
VEE_ERR_BUSY_REFRESH:	*セグメントのリフレッシュが進行中です。*
VEE_ERR_NULL_PTR:	*「ref_data_ptr」が NULL です。*
VEE_ERR_ILLEGAL_CONFIG:	*r_vee_rx_config.h で設定されていない参照データです。*
VEE_ERR_OVERFLOW:	*通常は発生しないエラー。付録を参照してください。*
VEE_ERR_FLASH_PE_FAIL:	*通常は発生しないエラー。付録を参照してください。*
VEE_ERR_UNKNOWN_WRITE:	*通常は発生しないエラー。付録を参照してください。*

Properties

プロトタイプは「r_vee_rx_if.h」ファイルにあります。

Description

この関数は、フラッシュ内の参照データの最新バージョンへのポインタを引数に設定します。

Reentrant

不可

Example:

```
vee_err_t       err;
uint8_t        *ref_data_ptr;
struct refdata  *ref_type2_ptr;

/* Example: reference data is a byte array */
err = R_VEE_GetRefDataPtr(&ref_data_ptr);

/* Example: reference data is in a structure format */
err = R_VEE_GetRefDataPtr((uint8_t **)&ref_type2_ptr);
```

Special Notes:

フラッシュは同時に読み書きすることができません。フラッシュ書き込みを開始する前に「ref_data_ptr」によるデータの読み出しを推奨します。

3.7　R_VEE_Control()

この関数は特殊操作を処理します。

Format

```
vee_err_t R_VEE_Control(vee_cmd_t cmd,
                        void     *arg_ptr);
```

Parameters

cmd

実行するコマンドを指定します。

typedef enum _vee_cmd

{

VEE_CMD_GET_STATUS,

VEE_CMD_GET_LAST_ID_WRITTEN,

VEE_CMD_GET_RECORD_SPACE_AVAIL,

VEE_CMD_QUERY_REFDATA_UPDATED,

VEE_CMD_REFRESH,

VEE_CMD_GET_SEG_ERASE_COUNT,

VEE_CMD_FORMAT,

VEE_CMD_END_ENUM

} vee_cmd_t;

arg_ptr

コマンド固有の引数へのポインタです。

Return Values

VEE_SUCCESS:　*正常終了*

VEE_READY:　*前回の API 呼び出しが完了しました（GET_STATUS コマンドのときのみ返されます）。*

VEE_ERR_BUSY:　*前回の API 呼び出しが引き続き実行中です。*

VEE_ERR_BUSY_REFRESH:　*セグメントのリフレッシュが進行中です。*

VEE_ERR_UNKNOWN_CMD:　*認識されていないコマンドです。*

VEE_ERR_NULL_PTR:　*「arg_ptr」が NULL なので、コマンドに引数を指定する必要があります。*

VEE_ERR_ILLEGAL_CONFIG:　*参照データが r_vee_rx_config.h で設定されていません。*

VEE_ERR_RECORD_NOT_FOUND:　*指定された ID のレコードは見つかりませんでした。*

VEE_ERR_OVERFLOW:　*通常は発生しないエラー。付録を参照してください。*

VEE_ERR_FLASH_PE_FAIL:　*通常は発生しないエラー。付録を参照してください。*

VEE_ERR_UNKNOWN_WRITE:　*通常は発生しないエラー。付録を参照してください。*

Properties

プロトタイプは「r_vee_rx_if.h」ファイルにあります。

Description

この関数は、VEE モジュールで特殊操作を処理します。コマンドには引数が必要な場合もあれば、不要の場合もあります（以下の例を参照）。

Reentrant

不可

Example: 前回の API 呼び出しが完了したか否かの確認 – Get Status

一般的に、このコマンドは、別の API 呼び出しを実行する前に、前回の書き込みまたはリフレッシュコマンドが完了したことを確認するために使用します。通常のリターンコードは READY、BUSY、または BUSY REFRESH のいずれかです。

```
vee_err_t   err;
flash_interrupt_event_t event;

 /* Check that that VEE is not in an error mode */
err = R_VEE_Control(VEE_CMD_GET_STATUS, &event);
if ((err == VEE_ERR_OVERFLOW) || (err == VEE_ERR_UNKNOWN_WRITE))
{
   while(1) ;  // development error
}
else if (err == VEE_ERR_FLASH_PE_FAIL)
{
   /* trying closing and re-opening VEE to clear error */
}
```

Example: 前回書き込まれたレコードの ID 取得

このコマンドは、前回書き込まれたレコードの ID を読み込みます。この呼び出しは、リセット（および Open 呼び出し）の後、リセット実行前のプロセスの場所を判定するために使用します。

```
vee_err_t err;
uint16_t  id;

/* variable “id” loaded with the ID of the last record written*/
err = R_VEE_Control(VEE_CMD_GET_LAST_ID_WRITTEN, &id);
if (err != VEE_SUCCESS)
{
   /* handle error (most likely BUSY; try again later) */
}
```

Example: レコード領域に残されたバイト数の取得

このコマンドを使用すると、レコード領域内で未使用のバイト数を確認できます。大量のデータの受信予定があり、自動リフレッシュの実行によって追加書き込みの実行に遅延が生じないように、当該データを書き込むレコード領域に十分なスペースが残っているか確認する際に便利なコマンドです。

```
vee_err_t err;
uint32_t  bytes_available;

/* “bytes_available” is loaded with number of unused bytes in record area */
err = R_VEE_Control(VEE_CMD_GET_RECORD_SPACE_AVAIL, &bytes_available);
if (err != VEE_SUCCESS)
{
   /* handle error (most likely BUSY; try again later) */
}

/* NOTE: When calculating how many records can fit in the remaining bytes,
 * be sure to include 4 bytes of internal overhead per record for fixed
 * length records, and 8 bytes of overhead for variable length records.
 */
```

Example: 参照データが既に更新済みか否かを判定

このコマンドの機能は GET RECORD SPACE AVAIL コマンドと似ています。つまり、このコマンドは、システムのタイミングに問題があり、WriteRefData()がセグメントリフレッシュを実行するか否かを事前に把握する必要がある場合に呼び出します。

```
vee_err_t err;
bool is_reference_updated;

/* “is_reference_updated” is loaded with “true” if reference data update area
 * has been used. If true, performing a WriteRefData() will trigger a Refresh.
 */
err = R_VEE_Control(VEE_CMD_QUERY_REFDATA_UPDATED, &is_reference_updated);
if (err != VEE_SUCCESS)
{
   /* handle error (most likely BUSY; try again later) */
}
```

Example: 強制リフレッシュ

本コマンドでは、任意のタイミングでセグメントリフレッシュを実行できます。アクティブセグメントの空きスペースが足りず、大容量データを記録したい場合、強制リフレッシュで新しいセグメントを使用できます。

```
   vee_err_t err;

   err = R_VEE_Control(VEE_CMD_REFRESH, NULL);
   if (err != VEE_SUCCESS)
   {
      /* handle error (most likely BUSY; try again later) */
   }
```

Example: セグメント消去回数の取得

このコマンドは、セグメントの消去回数を返します。これにより、データフラッシュ寿命を推定することができます。セグメントの消去中にリセットが起きた場合、部分消去は合計カウントに含まれないのでご注意ください。

```
vee_err_t err;
uint32_t  erase_count;

/* “erase_count” loaded with number of times segments have been erased */
err = R_VEE_Control(VEE_CMD_GET_SEG_ERASE_COUNT, &erase_count);
if (err != VEE_SUCCESS)
{
   /* handle error (most likely BUSY; try again later) */
}
```

Example: データフラッシュのフォーマット

このコマンドは開発専用です。データフラッシュのすべてを消去し、参照データが設定されている場合は参照データを書き込み、最後のセグメントをアクティブセグメントとしてマークします。

```
vee_err_t           err;
struct my_refdata   refdata1;
uint8_t             refdata2[VEE_CFG_REF_DATA_SIZE];

// (load reference data here)

/* Call format using reference data structure */
err = R_VEE_Control(VEE_CMD_FORMAT, &refdata1);
if (err != VEE_SUCCESS)
{
   /* handle error (most likely BUSY; try again later) */
}

/* Call format using reference data array */
err = R_VEE_Control(VEE_CMD_FORMAT, refdata2);
if (err != VEE_SUCCESS)
{
   /* handle error (most likely BUSY; try again later) */
}

/* Call format when reference data is not configured in r_vee_rx_config.h */
err = R_VEE_Control(VEE_CMD_FORMAT, NULL);
if (err != VEE_SUCCESS)
{
   /* handle error (most likely BUSY; try again later) */
}
```

Special Notes:

なし

3.8 R_VEE_Close()

フラッシュモジュールと VEE モジュールを終了します。

Format

```
vee_err_t R_VEE_Close(void);
```

Parameters

なし

Return Values

VEE_SUCCESS:	*正常終了*
VEE_ERR_BUSY:	*前回の API 呼び出しが引き続き実行中です。*
VEE_ERR_BUSY_REFRESH:	*セグメントのリフレッシュが進行中です。*

Properties

プロトタイプは「r_vee_rx_if.h」ファイルにあります。

Description

この関数は、フラッシュモジュールと VEE モジュールを終了します。別の API 処理が進行中の場合、BUSY エラーを返します。

Reentrant

不可

Example:

```
vee_err_t err;

err = R_VEE_Close();
```

Special Notes:

VEE の外部で割り込みが無効化されている場合、フラッシュモジュールを終了することができない場合があります。

3.9　R_VEE_GetVersion()

VEE モジュールのバージョンを返します。

Format

```
uint32_t R_VEE_GetVersion(void);
```

Parameters

なし

Return Values

バージョン番号

Properties

プロトタイプは「r_vee_rx_if.h」ファイルにあります。

Description

このモジュールのバージョンを返します。バージョン番号は、上位 2 バイトがメジャーバージョン番号で下位 2 バイトがマイナーバージョン番号になるようにコード化されます。

Example

```
uint32_t   version;

version = R_VEE_GetVersion();
```

Special Notes:

この関数は「#pragma inline」命令を使用してインライン化されます。

4. デモプロジェクト

デモプロジェクトはスタンドアロンプログラムです。main()関数では、VEE モジュールとその関連モジュール（フラッシュモジュールなど）を使用します。

4.1 vee_demo_rskrx231

これは RSKRX231 スターターキットのシンプルなデモです。このデモは、VEE モジュールの API を使用して、異なるレコード ID のレコードを書き込み、読み出します。

設定と実行

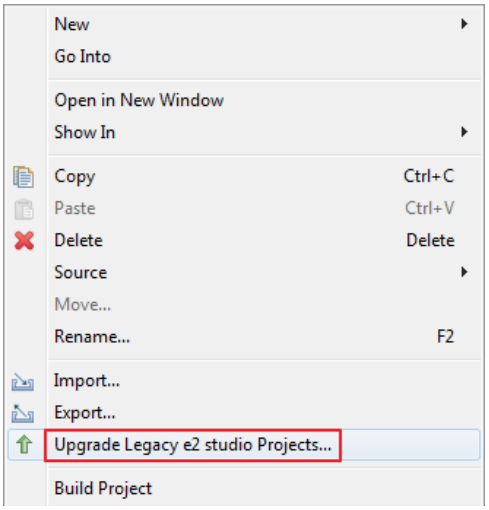
1. デモプロジェクトをコンパイルしてダウンロードします。
2. Reset Go をクリックしてソフトウェアを実行します。プログラムが main()で停止した場合、F8 キーを押すと再開します。

サポートされるボード

RSKRX231

4.2 デモのワークスペースへの追加

デモプロジェクトをワークスペースに追加するには、File > Import > General > Existing Projects into Workspace の順に選択し、「Next」をクリックします。Import Projects ダイアログで、「Select archive file」ラジオボタンを選択し、デモ用の.zip ファイルを参照します。e² studio v6.0.0 以降を使用している場合は、プロジェクトを適切にビルドするためにインポートした後にプロジェクトを更新してください。更新するには、プロジェクトフォルダを右クリックして「Upgrade Legacy e2studio Projects」を選択します。



付録：エラーモード

注：以下のすべてのエラーリターンコードは、前の API 呼び出しを処理している間にエラーが発生し、VEE がエラーモードに入ったことを示します。

エラーモードに入ると、呼び出し可能な関数は VEE_CMD_GET_STATUS コマンドが指定された R_VEE_Control()および R_VEE_Close()のみです。VEE_CFG_ALLOW_GETS_AFTER_FAIL オプションが「r_vee_rx_config.h」で 1 に設定されている場合、R_VEE_GetRecordPtr()関数および R_VEE_GetRefDataPtr()関数も呼び出すことができます。

VEE_ERR_OVERFLOW

このエラーは、動作中にのみ検出されます。セグメントサイズが小さすぎて、ユーザによって定義された各 ID に 1 つ以上のレコードを保持することができない場合に発生します。このエラーを解決するには、「r_vee_rx_config.h」で VEE_CFG_NUM_SEGMENTS によって定義されたセグメントの数を減らします。定義されたセグメントの数が 2 つのみでもエラーが解消されない場合は、よりサイズの大きなデータフラッシュを持つ MCU が必要です。

VEE_ERR_UNKNOWN_WRITE

このエラーは、VEE モジュール本体がユーザによって変更された場合にのみ発生します。これは、ユーザによって新しい書き込み状態が作成されたが、対応する処理コードは実装されていなかったことを示します。VEE モジュールは変更しないことを強く推奨します。

VEE_ERR_FLASH_PE_FAIL

このエラーは、フラッシュ処理中のリセット、過度のノイズやフラッシュの過度な劣化により、プログラムや消去、ブランクチェック処理が失敗したことを示します。エラーの原因がフラッシュ処理中のリセットまたはノイズの場合、VEE モジュールを終了してもう一度オープンすると、自動でリフレッシュが実行され、部分的に書き込まれたデータは消去されます。

ホームページとサポート窓口

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

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

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.01	2018.03.01	—	初版発行
1.10	2018.10.02		最初のセグメントがアクティブで参照データが存在しない状態から、リセットによりシステムを再起動した場合、その後の書き込みが最終的に失敗し、セグメントの更新が行われないというバグを修正
1.11	2018.10.18		デモと readme.txt の更新

製品ご使用上の注意事項

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

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】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、将来の機能拡張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。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子
   （または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、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、内部ROM、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、ノイズ輻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。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、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。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：　輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、
   金融端末基幹システム、各種安全制御装置等

   当社製品は、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。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、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。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様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。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載され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.4.0-1　2017.11)

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口　http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：https://www.renesas.com/contact/