RZ/A2M グループ

USB Basic Host Driver

要旨

本アプリケーションノートでは、USB Basic Host Driver Firmware について説明します。以降、本モジュールを USB-BASIC-F/W と称します。

動作確認デバイス

RZ/A2M 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。

目次

1. 概要 5
1.1 注意事項 6
1.2 用語一覧 6

2. 動作確認条件 7

3. 関連アプリケーションノート 8

4. ソフトウェア構成 9
4.1 モジュール構成 9
4.2 スケジューラ機能 10
4.3 ホストデバイスクラスコントロールドライバ（HDCD） 11
4.3.1 登録 11
4.3.2 タスク設定 11
4.3.3 クラスチェック処理 11
4.4 ホストコントロールドライバ（HCD） 12
4.4.1 基本機能 12
4.4.2 HCD に対する要求発行 12
4.4.3 複数のデバイスを接続する場合 12
4.5 ホストマネージャ（MGR） 13
4.5.1 基本機能 13
4.5.2 USB 標準リクエスト 13
4.5.3 クラスチェック 13
4.6 HUB クラスドライバ（HUBCD） 14
4.6.1 基本機能 14
4.6.2 ダウンポートの状態管理 14
4.6.3 ダウンポートへのデバイス接続 14
4.6.4 クラスリクエスト 14

5. コンフィグレーション 15
5.1 SmartConfigurator 設定(r_usbh0_basic_drv_sc_cfg.h) 15
5.1.1 OS/非 OS 選択 15
5.2 USB Host Basic 設定(r_usbh0_basic_config.h) 15
5.2.1 デバイスクラス設定 15
5.2.2 パイプの最大数設定 15
5.2.3 USB Embedded Host のコンプライアンステスト対応設定 15
5.2.4 コンプライアンステスト用情報表示関数設定 15
5.2.5 オーバーカレント検出時の関数設定 16
5.2.6 デバイスアドレスの最大値設定 16
5.2.7 タスク ID の最大値設定 16
5.2.8 プライオリティの最大値設定 16
5.2.9 メモリプールの最大数設定 16
5.2.10 メッセージプールの最大数設定 16
5.2.11 デバッグ情報出力設定 16
5.2.12 デバッグ情報出力関数設定 17

5.3 EHCI ユーザ定義情報ファイル(r_usbh0_hehci_def_usr.h） 18
5.3.1 EHCI ピリオディックフレームリストのサイズ指定 18
5.3.2 EHCI キューヘッドデータ構造メモリの最大数指定 18
5.3.3 EHCI qTD データ構造メモリの最大数指定 18
5.3.4 EHCI iTD データ構造メモリの最大数指定 18
5.3.5 EHCI siTD データ構造メモリの最大数指定 18
5.3.6 EHCI iTD データ構造の最大データ転送サイズ指定 19
5.3.7 EHCI siTD データ構造の最大データ転送サイズ指定 19
5.3.8 EHCI タイムアウト時間指定 19
5.4 OHCI ユーザ定義情報ファイル(r_usbh0_hohci_def_usr.h） 20
5.4.1 OHCI エンドポイントデータ構造メモリの最大数指定 20
5.4.2 OHCI エンドポイントディスクリプタデータ構造メモリの最大数指定 20
5.4.3 OHCI トランスファディスクリプタデータ構造メモリの最大数指定 20
5.4.4 OHCI アイソクロナスデバイスの最大数指定 20
5.4.5 OHCI アイソクロナスの最大データ転送サイズ指定 21
5.4.6 OHCI タイムアウト時間指定 21
5.5 ターゲットペリフェラルリスト（TPL） 22
5.6 構造体定義 23
5.6.1 st_usbh0_hcdreg_t 構造体 23
5.6.2 st_usbh0_setup_t 構造体 23
5.6.3 usbh0_utr 構造体 24
5.7 API 関数 25
5.8 R_USBH0_HstdTransferStart 26
5.9 R_USBH0_HstdTransferEnd 27
5.10 R_USBH0_HstdDriverRegistration 28
5.11 R_USBH0_HstdReturnEnuMGR 29
5.12 R_USBH0_HstdChangeDeviceState 30
5.13 R_USBH0_HstdSetPipe 31
5.14 R_USBH0_HstdGetPipeID 32
5.15 R_USBH0_HstdClearPipe 33
5.16 R_USBH0_HstdMgrOpen 34
5.17 R_USBH0_HstdMgrClose 35
5.18 R_USBH0_HstdMgrTask 36
5.19 R_USBH0_HhubTask 37
5.20 コールバック関数 38

6. USB 通信 40
6.1 パイプ 40
6.2 データ転送要求 40
6.3 転送結果の通知 41
6.4 転送のリトライ 41
6.5 受信時の注意事項 41
6.6 コントロール転送 42
6.7 データ転送 43

7. スケジューラ 44
7.1 概要 44

7.2 スケジューラマクロ 44
7.3 スケジューラ API 関数 44
7.4 usbh0_hstd_snd_msg 45
7.5 usbh0_hstd_rec_msg 46
7.6 usbh0_hstd_pget_blk 47
7.7 usbh0_hstd_rel_blk 48
7.8 R_USBH0_CstdScheduler 49
7.9 R_USBH0_CstdSetTaskPri 50
7.10 R_USBH0_CstdCheckSchedule 51

8. 参考ドキュメント 52

1. 概要

USB-BASIC-F/W は、USB2.0HS ホストモジュールの制御を行います。USB-BASIC-F/W は Renesas が提供するサンプルデバイスクラスドライバ又はユーザが作成したデバイスクラスドライバと組み合わせることで動作します。

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。

- デバイスの接続／切断、サスペンド／レジューム、USB バスリセット処理
- コントロール転送、バルク転送、インターラプト転送、アイソクロナス転送
- Low-Speed / Full-Speed / High-Speed ファンクションデバイスとエニュメレーション
  (動作スピードはデバイスにより異なります)
- 転送エラー判定および転送リトライ
- USB-BASIC-F/W をカスタマイズせずに、複数のデバイスクラスドライバを実装することが可能

API などの名称は、ポート 0 とポート 1 で異なります。

本書では、API 名称などはポート 0 のものを例として記載します。

表 1.1　使用する周辺機能と用途

周辺機能	用途
ホスト PC	サンプルコードのメッセージ出力

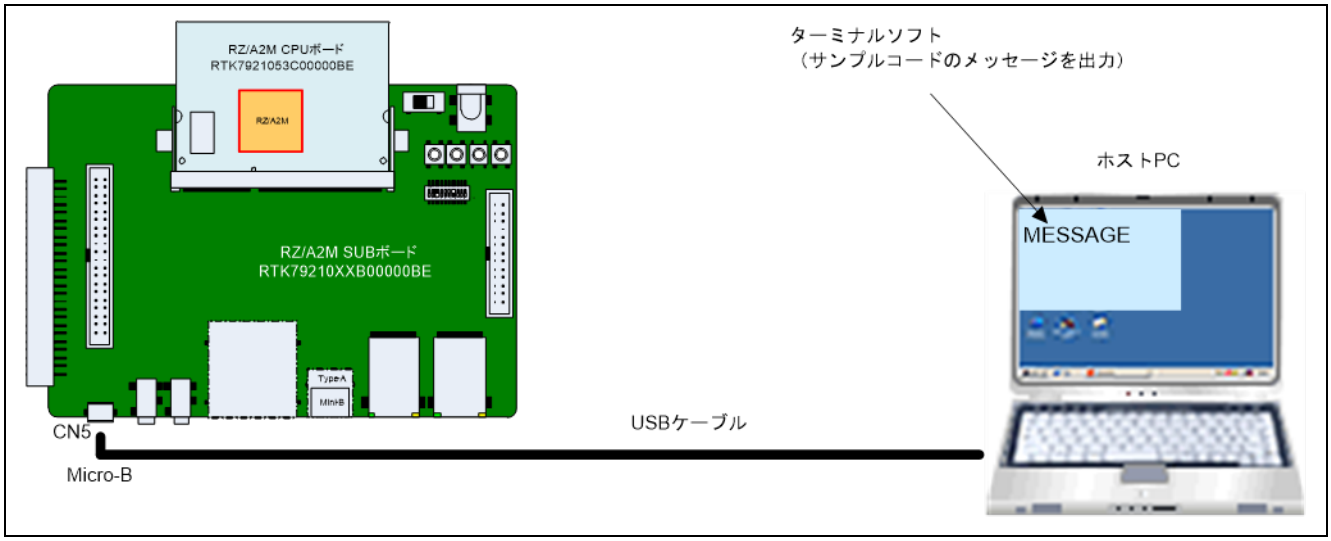

図 1.1　動作環境

1.1 注意事項

クラス規定やベンダ固有リクエストの発行が必要な場合をはじめ、通信速度、プログラム容量等を考慮する場合、さらにユーザインタフェースを個別に設定する場合には、お客様にてカスタマイズしてください。

USB-BASIC-F/W 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、ユーザ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。

1.2 用語一覧

本資料で使用される用語と略語は以下のとおりです。

用語／略語	詳細
APL	Application program
cstd	USB-BASIC-F/W 用の関数&ファイルのプレフィックス
HCD	Host control driver of USB-BASIC-F/W
HDCD	Host device class driver (device driver and USB class driver)
hstd	Host USB-BASIC-F/W用の関数&ファイルのプレフィックス
HUBCD	Hub class sample driver
H/W	Renesas USB device RZ/A2Mグループ
MGR	Device state manager of HCD
USB	Universal Serial Bus
USB-BASIC-F/W	USB Basic Host firmware for RZ/A2Mグループ
スケジューラ	OSLessでタスク動作を簡易的にスケジューリングするもの
スケジューラマクロ	OSLessでスケジューラ機能を呼び出すために使用されるもの
タスク	処理の単位

2. 動作確認条件

本アプリケーションノートのサンプルコードは、下記の条件で動作を確認しています。

表 2.1 動作確認条件

項目	内容
使用マイコン	RZ/A2M
動作周波数（注）	CPU クロック（Iφ）：528MHz 画像処理クロック（Gφ）：264MHz 内部バスクロック（Bφ）：132MHz 周辺クロック 1（P1φ）：66MHz 周辺クロック 0（P0φ）：33MHz QSPI0_SPCLK：66MHz CKIO：132MHz
動作電圧	電源電圧（I/O）：3.3V 電源電圧（1.8/3.3V 切替 I/O（PVcc_SPI））：3.3V 電源電圧（内部）：1.2V
統合開発環境	e2 studio V7.6.0
C コンパイラ	GNU Arm Embedded Toolchain 6.3.1
	コンパイラオプション（ディレクトリパスの追加は除く） Release: -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Os -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -Wstack-usage=100 -fabi-version=0 Hardware Debug: -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Og -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -g3 -Wstack-usage=100 -fabi-version=0
動作モード	ブートモード 3 （シリアルフラッシュブート 3.3V 品）
ターミナルソフトの通信設定	• 通信速度：115200bps • データ長：8 ビット • パリティ：なし • ストップビット長：1 ビット • フロー制御：なし
使用ボード	RZ/A2M CPU ボード　RTK7921053C00000BE RZ/A2M SUB ボード　RTK79210XXB00000BE
使用デバイス （ボード上で使用する機能）	• シリアルフラッシュメモリ（SPI マルチ I/O バス空間に接続） メーカ名：Macronix 社、型名：MX25L51245GXD • RL78/G1C（USB 通信とシリアル通信を変換し、ホスト PC との通信に使用） • LED1

【注】 クロックモード 1（EXTAL 端子からの 24MHz のクロック入力）で使用時の動作周波数です。

3. 関連アプリケーションノート

本アプリケーションノートに関連するアプリケーションノートを以下に示します。併せて参照してください。

・なし

4. ソフトウェア構成

4.1 モジュール構成

USB-BASIC-F/W は、ホストドライバ（HCD）、ホストマネージャ（MGR）、ハブクラスドライバ（HUBCD）から構成されています。HDCD は USB-BASIC-F/W の一部ではなくクラスドライバです。

MGR は、接続されたデバイスの状態管理とエニュメレーションをします。また、アプリケーション（APL）がデバイス状態を変更する場合は、APL から該当の API 関数をコールすることにより MGR が HCD もしくは HUBCD に状態変更を要求します。

HUBCD は、USB ハブのダウンポートに接続されたデバイスのエニュメレーションと状態管理をするサンプルプログラムです。

クラスやベンダ固有リクエストの発行が必要な場合をはじめ、通信速度、プログラム容量等を考慮する場合、さらにユーザインタフェースを個別に設定する場合には、ユーザにてカスタマイズして頂く必要があります。図 4.1 に USB-BASIC-F/W の構成図、表 4.1 にモジュール機能概要を示します。

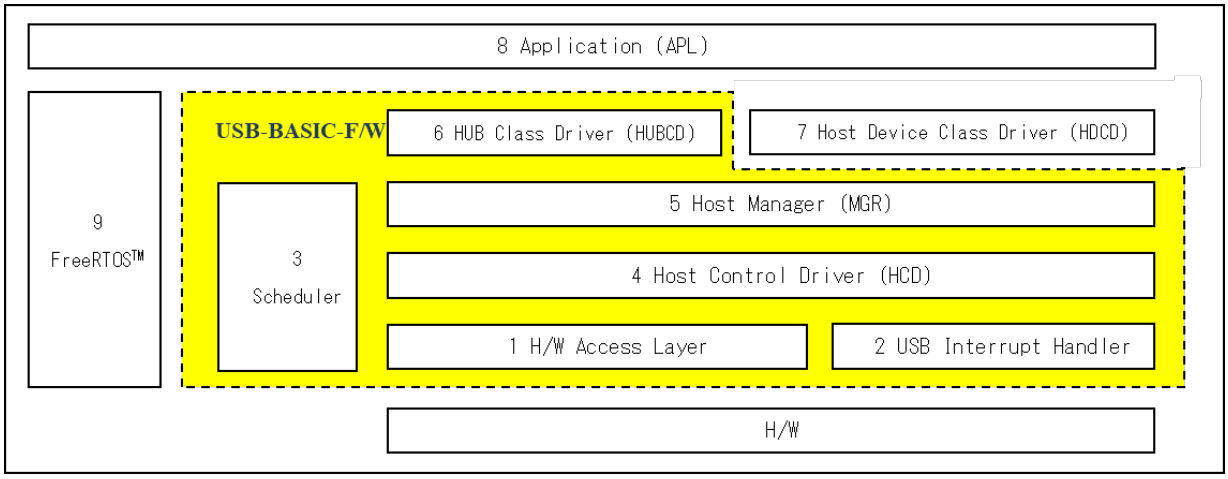

図 4.1 USB-BASIC-F/W のモジュール構成図

表 4.1 モジュール機能概要

No	Module Name	Description
1	H/W Access Layer	USB2.0 HS ホストモジュール制御
2	USB Interrupt Handler	USB 割り込みハンドラ
3	Scheduler	スケジューラ機能
4	Host Control Driver	ホストトランザクション管理
5	Host Manager	デバイス状態管理、エニュメレーション処理 HCD/HUBCD 制御メッセージ判定
6	HUB Class Driver	HUB ダウンポートデバイス状態管理、エニュメレーション処理
7	Host Device Class Driver	デバイスクラス処理 （システムにあわせてご用意ください）
8	Application	アプリケーション （システムにあわせてご用意ください）
9	FreeRTOS™	FreeRTOS™

4.2 スケジューラ機能

本モジュールは、スケジューラ機能を使用して各タスクや H/W の要求をタスクの優先順位にしたがって管理します。また優先順位が同じタスクに複数の要求が発生した場合は FIFO 構造で要求を実行します。

タスク間の要求はメッセージの送受信で実現しています。また要求の終了はコールバック関数によりタスクに応答されるためスケジューラ本体を改変することなくユーザシステムに適合したクラスドライバの実装が可能です。

スケジューラ機能の詳細は、7 章を参照してください。

4.3 ホストデバイスクラスコントロールドライバ（HDCD）

4.3.1 登録

ユーザシステムに合わせた HDCD を作成する必要があります。

作成した HDCD の各種情報を API 関数 R_USBH0_HstdDriverRegistration ()を使って st_usbh0_hcdreg_t 構造体に登録してください。

登録方法の詳細は、5.9 章を参照してください。

4.3.2 タスク設定

HDCD がスケジューラを使用する場合は、7 章のスケジューラを参照してください。

r_usbh0_basic_config.h ファイルに、追加するタスク毎にタスク ID、メールボックス ID、メモリプール ID を追加してください。

タスク優先度設定は、API 関数 R_USBH0_CstdSetTaskPri ()を使って設定してください。

これらの項目設定時には、以下の点に注意してください。

- タスク ID は続き番号を使用し、重複しないように設定してください。
- メールボックス ID とメモリプール ID は、タスク ID と同じ値を設定してください。
- 追加するタスクの優先度は、HCD(1), MGR(2), HUB(3)より低く(4～7)設定してください。優先度は 0 が最高で、7 が最低です。

以下に、r_usb_basic_config.h の設定追加例を示します。

```
#define  USB_SMP_TSK     USB_TID_6     : タスク ID
#define  USB_SMP_MBX     USB_SMP_TSK   : メールボックス ID
#define  USB_SMP_MPL     USB_SMP_TSK   : メモリプール ID
```

必要がある場合は、r_usbh0_basic_config.h ファイルで、以下の項目を設定してください。

・タスク ID の最大値設定（5.1.7 参照）

・メモリプールの最大数設定（5.1.9 参照）

・メッセージプールの最大数設定（5.1.10 参照）

4.3.3 クラスチェック処理

クラスチェックコールバック関数でエニュメレーション中にクラスチェック処理をする必要があります。

処理完了時に API 関数 R_USBH0_HstdReturnEnuMGR ()を使って結果を通知してください。

4.4 ホストコントロールドライバ（HCD）

4.4.1 基本機能

HCD は、H/W 制御用のプログラムです。HCD の機能を以下に示します。

1. Control 転送（ControlRead/ControlWrite/No-dataControl）および結果通知
2. Data 転送（Bulk /Interrupt/Isochronous）および結果通知
3. データ転送の強制終了（全パイプ）
4. USB 通信エラー判定および転送リトライ
5. USB バスリセット信号送出およびリセットハンドシェイク結果通知
6. サスペンド信号/レジューム信号送出
7. 割り込みによるアタッチ/デタッチ検出

4.4.2 HCD に対する要求発行

HCD に対して H/W 制御要求を発行する場合は、API 関数を用います。

なお、接続されているデバイスステートの更新要求は、HCD が直接制御する場合と USB ハブ経由で制御する場合があります。MGR がデバイスアドレスを判断し制御要求を HCD / HUBCD タスクに発行します。

HCD は上位タスクからの要求に対して、コールバック関数を用いて結果を通知します。

4.4.3 複数のデバイスを接続する場合

接続された HUB のデバイスアドレスを”1”としてエニュメレーションします。また、HUBCD は、ダウンポートに接続されたデバイスに対して”2”以降のデバイスアドレスを自動的に割り振ります。

複数のデバイスとエニュメレーション及び通信が可能ですが、複数のデバイスに対し同時にエニュメレーションをしません。この場合、最初に接続されたデバイスのエニュメレーションが終了してから次に接続されたデバイスのエニュメレーションを開始します。

4.5 ホストマネージャ（MGR）

4.5.1 基本機能

MGR は、HCD と HDCD 間の機能を補完するタスクです。MGR の機能を以下に示します。

1. HDCD の登録
2. 接続されたデバイスの状態管理
3. 接続されたデバイスのエニュメレーション
4. ディスクリプタからエンドポイント情報の検索

4.5.2 USB 標準リクエスト

MGR は、接続されたデバイスに対してエニュメレーションをします。

MGR が発行する USB 標準リクエストを以下に示します。また、デバイスから取得したディスクリプタ情報は内部保存され、API 関数にてパイプ情報として登録することができます。

- GET_DESCRIPTOR（Device Descriptor）
- SET_ADDRESS
- GET_DESCRIPTOR（Configuration Descriptor）
- SET_CONFIGURATION

4.5.3 クラスチェック

MGR は、エニュメレーション時に GET_DESCRIPTOR リクエストで取得した情報を HDCD に通知し、接続されたデバイスが動作可能であるか確認します。

HDCD からの API 関数 R_USBH0_HstdReturnEnuMGR ()での応答によりエニュメレーションを継続します。

4.6 HUB クラスドライバ（HUBCD）

4.6.1 基本機能

HUBCD は、接続された USB ハブのダウンポートの状態を管理し、HCD と HDCD 間の機能を補完します。HUBCD の機能を以下に示します。

1. USB ハブのエニュメレーション
2. USB ハブのダウンポートに接続されたデバイスの状態管理
3. USB ハブのダウンポートに接続されたデバイスのエニュメレーション

4.6.2 ダウンポートの状態管理

HUBCD は USB ハブが接続されるとすべてのダウンポートに対して

1) ポートパワー許可（HubPortSetFeature：USB_HUB_PORT_POWER）
2) ポートの初期化（HubPortClrFeature：USB_HUB_C_PORT_CONNECTION）
3) ポートステータスの取得（HubPortStatus：USB_HUB_PORT_CONNECTION）

をし、ダウンポートのデバイス接続状況を確認します。

4.6.3 ダウンポートへのデバイス接続

HUBCD は USB ハブからダウンポートのデバイスアタッチの通知を受けると、USB ハブに対して USB リセット信号要求を発行（HubPortSetFeature：USB_HUB_PORT_RESET）します。

その後、MGR タスクの資源を使用して、接続されたデバイスとエニュメレーションをします。

HUBCD はリセットハンドシェイクの結果を保存し、接続されたデバイスに対して使用していないデバイスアドレスを順次割り振ります。

4.6.4 クラスリクエスト

HUBCD がサポートしているクラスリクエストを表 4.2 に示します。

表 4.2 USB ハブクラスリクエスト

リクエスト	実装状況	関数名	機能
ClearHubFeature	×		
ClearPortFeature	○	usbh0_hhub_port_clr_feature	USB_HUB_C_PORT_CONNECTION USB_HUB_C_PORT_RESET
ClearTTBuffer	×		
GetHubDescriptor	○	usbh0_hhub_get_hub_descriptor	ハブディスクリプタ取得
GetHubStatus	×		
GetPortStatus	○	usbh0_hhub_get_status	ポートステータス取得
ResetTT	×		
SetHubDescriptor	×		
SetHubFeature	×		
SetPortFeature	○	usbh0_hhub_port_set_feature	USB_HUB_PORT_POWER USB_HUB_PORT_RESET
GetTTState	×		
StopTT	×		

5. コンフィグレーション

5.1 SmartConfigurator 設定(r_usbh0_basic_drv_sc_cfg.h)

下記の定義に対する指定を SmartConfigurator で行ってください。

5.1.1 OS/非 OS 選択

OSLess/FreeRTOS を選択してください。

```
#define     BSP_CFG_RTOS_USED   0     // OSLess
#define     BSP_CFG_RTOS_USED   1     // FreeRTOS
```

5.2 USB Host Basic 設定(r_usbh0_basic_config.h)

共通ユーザ定義情報ファイル（r_usbh0_basic_config.h）を書き換えることにより、USB-BASIC-F/W の機能設定をします。

システムに合わせて、下記項目を変更してください。

5.2.1 デバイスクラス設定

使用するデバイスクラスを設定します。

以下の定義のうち、お客様が使用する USB ドライバの定義を有効にしてください。なお、複数の定義を有効にすることはできません。有効にすることができる定義数は一つのみです。

```
#define   USB_CFG_HCDC_USE          // Host Communication Device Class
#define   USB_CFG_HHID_USE          // Host Human Interface Device Class
#define   USB_CFG_HMSC_USE          // Host Mass Storage Class
#define   USB_CFG_HVNDR_USE         // Host Vendor Class
```

5.2.2 パイプの最大数設定

パイプの最大数を設定します。

接続されたデバイスのエンドポイント毎に 1 つ使用します。

設定値を大きくすると使用メモリが増加します。

```
#define USBH0_MAXPIPE        32u
```

5.2.3 USB Embedded Host のコンプライアンステスト対応設定

USB Embedded Host のコンプライアンステスト対応設定をします。

コンプライアンステストに対応する場合、本マクロを有効にしてください。

```
#define USBH0_HOST_COMPLIANCE_MODE   : コンプライアンステストに対応
//#define USBH0_HOST_COMPLIANCE_MODE   : コンプライアンステストに非対応
```

5.2.4 コンプライアンステスト用情報表示関数設定

コンプライアンステスト対応時に表示機器（LCD 等）に情報を表示させるための関数を登録してください。

```
#define USBH0_COMPLIANCE_DISP(data1, data2)  usb_cstd_DummyFunction(data1, data2)
```

5.2.5 オーバーカレント検出時の関数設定

オーバーカレント検出時に呼び出される関数を登録してください。

```
#define USBH0_OVERCURRENT(rootport)
```

ローカル定義情報ファイル（r_usbh0_basic_local.h）は、以下の機能設定をしています

5.2.6 デバイスアドレスの最大値設定

デバイスアドレスの最大値（接続できるデバイスの最大数）を設定します。

設定値を大きくすると使用メモリが増加します。

```
#define USBH0_MAXDEVADDR      12u
```

5.2.7 タスク ID の最大値設定

スケジューラのタスク ID の最大数を設定します。

設定値を大きくすると使用メモリが増加します。

```
#define USBH0_IDMAX           11u
```

5.2.8 プライオリティの最大値設定

スケジューラのプライオリティの最大値を設定します。

設定値を大きくすると使用メモリが増加します。

```
#define USBH0_PRIMAX          8u
```

5.2.9 メモリプールの最大数設定

スケジューラのメモリプールの最大数を設定します。

設定値を大きくすると使用メモリが増加します。

```
#define USBH0_BLKMAX          20u
```

5.2.10 メッセージプールの最大数設定

スケジューラのメッセージプールの最大数を設定します。

設定値を大きくすると使用メモリが増加します。

```
#define USBH0_TABLEMAX        USBH0_BLKMAX
```

5.2.11 デバッグ情報出力設定

デバッグ情報の出力設定をします。

```
#define USBH0_DEBUG_OUTPUT    : デバッグ出力有効
//#define USBH0_DEBUG_OUTPUT  : デバッグ出力無効
```

5.2.12 デバッグ情報出力関数設定

UART および表示デバイスなどにデバッグ情報を出力するマクロです。シリアルドライバ、表示デバイス用ドライバが必要になります。

ユーザシステムに合わせて関数を登録してください。

```
#define USBH0_PRINTF0(FORM)                     printf(FORM)
#define USBH0_PRINTF1(FORM,x1)                  printf (FORM,x1)
#define USBH0_PRINTF2(FORM,x1,x2)               printf (FORM,x1,x2)
#define USBH0_PRINTF3(FORM,x1,x2,x3)            printf (FORM,x1,x2,x3)
#define USBH0_PRINTF4(FORM,x1,x2,x3,x4)         printf (FORM,x1,x2,x3,x4)
#define USBH0_PRINTF5(FORM,x1,x2,x3,x4,x5)      printf (FORM,x1,x2,x3,x4,x5)
#define USBH0_PRINTF6(FORM,x1,x2,x3,x4,x5,x6)   printf (FORM,x1,x2,x3,x4,x5,x6)
#define USBH0_PRINTF7(FORM,x1,x2,x3,x4,x5,x6,x7) printf (FORM,x1,x2,x3,x4,x5,x6,x7)
#define USBH0_PRINTF8(FORM,x1,x2,x3,x4,x5,x6,x7,x8)    printf
(FORM,x1,x2,x3,x4,x5,x6,x7,x8)
```

5.3 EHCI ユーザ定義情報ファイル(r_usbh0_hehci_def_usr.h)

5.3.1 EHCI ピリオディックフレームリストのサイズ指定

EHCI ピリオディックフレームリストのサイズを 256、512、1024 の値から指定してください。

このサイズ指定によりピリオディック転送のスケジューリング幅が変更されます。

例）256 バイトを指定する場合

```
#define  USBH0_EHCI_PFL_SIZE  256
```

5.3.2 EHCI キューヘッドデータ構造メモリの最大数指定

EHCI キューヘッドデータ構造メモリの最大数を指定してください。

キューヘッドデータ構造は、コントロール転送、バルク転送、インターラプト転送のエンドポイントパイプ数分必要となります。

ユーザシステムに合わせて設定してください。

例）16 を指定する場合

```
#define  USBH0_EHCI_NUM_QH  16
```

5.3.3 EHCI qTD データ構造メモリの最大数指定

EHCI qTD(デバイスステータスレジスタキュー要素転送記述子)データ構造メモリの最大数を指定してください。

qTD は、コントロール転送、バルク転送、インターラプト転送において転送管理用の記述子としてキューヘッドとリンクして使用されます。

一つの qTD で転送可能な最大データサイズは 20480byte です。

また、コントロール転送の場合は、SETUP ステージ、DATA ステージ、STATUS ステージ毎に一つの qTD が必要となります。

ユーザシステムに合わせて設定してください。

例）256 を指定する場合

```
#define  USBH0_EHCI_NUM_QTD  256
```

5.3.4 EHCI iTD データ構造メモリの最大数指定

EHCI iTD（ハイスピード用アイソクロナス転送記述子）データ構造メモリの最大数を指定してください。

iTD データ構造は、ハイスピードアイソクロナス転送のエンドポイントパイプ数分必要となります。

ユーザシステムに合わせて設定してください。

例）4 を指定する場合

```
#define  USBH0_EHCI_NUM_ITD  4
```

5.3.5 EHCI siTD データ構造メモリの最大数指定

EHCI siTD（スプリットトランザクションアイソクロナス転送記述子）データ構造メモリの最大数を指定してください。

siTD データ構造は、スプリットトランザクションアイソクロナス転送のエンドポイントパイプ数分必要となります。

ユーザシステムに合わせて設定してください。

例）4 を指定する場合

```
#define  USBH0_EHCI_NUM_SITD  4
```

5.3.6 EHCI iTD データ構造の最大データ転送サイズ指定

EHCI iTD データ構造の最大データ転送サイズを指定してください。

この転送サイズは、ハイスピードアイソクロナス転送の 1 回分（1 トランザクション）の最大転送サイズを指定するものです。

設定可能な最大データサイズは 1024 です。

ユーザシステムに合わせて設定してください。

例）512 バイトを指定する場合

```
#define  USBH0_EHCI_ITD_DATA_SIZE  512
```

5.3.7 EHCI siTD データ構造の最大データ転送サイズ指定

EHCI siTD データ構造の最大データ転送サイズは 184 で固定です。

5.3.8 EHCI タイムアウト時間指定

EHCI タイムアウト時間を msec 単位で指定してください。

例）3 秒を指定する場合

```
#define  USBH0_EHCI_TIMEOUT  3000
```

5.4 OHCI ユーザ定義情報ファイル(r_usbh0_hohci_def_usr.h)

5.4.1 OHCI エンドポイントデータ構造メモリの最大数指定

OHCI エンドポイントデータ構造メモリの最大数を指定してください。

OHCI エンドポイントデータ構造は、コントロール転送、バルク転送、インターラプト転送、アイソクロナス転送のエンドポイントパイプ数分必要となります。

ユーザシステムに合わせて設定してください。

例）16 を指定する場合

```
#define  USBH0_OHCI_NUM_ENDPOINT  16
```

5.4.2 OHCI エンドポイントディスクリプタデータ構造メモリの最大数指定

OHCI エンドポイントディスクリプタデータ構造メモリの最大数を指定してください。

OHCI エンドポイントディスクリプタデータ構造は、コントロール転送、バルク転送、インターラプト転送、アイソクロナス転送のエンドポイントパイプ数＋インターラプト転送のスケジューリング用に 31 個必要となります。

ユーザシステムに合わせて設定してください。

例）64 を指定する場合

```
#define  USBH0_OHCI_NUM_ED  64
```

5.4.3 OHCI トランスファディスクリプタデータ構造メモリの最大数指定

OHCI トランスファディスクリプタデータ構造メモリの最大数を指定してください。

OHCI トランスファディスクリプタデータ構造は、コントロール転送、バルク転送、インターラプト転送、アイソクロナス転送において転送管理用の記述子として使用されます。

一つのトランスファディスクリプタで転送可能な最大データサイズは 8192byte です。

ユーザシステムに合わせて設定してください。

例）256 を指定する場合

```
#define  USBH0_OHCI_NUM_TD  256
```

5.4.4 OHCI アイソクロナスデバイスの最大数指定

OHCI アイソクロナスデバイスの最大数を指定してください。

ユーザシステムに合わせて設定してください。

例）4 を指定する場合

```
#define  USBH0_OHCI_ISO_MAXDEVICE  4
```

5.4.5 OHCI アイソクロナスの最大データ転送サイズ指定

OHCI アイソクロナスデバイスの最大データ転送サイズを指定してください。

この転送サイズは、OHCI アイソクロナス転送の 1 回分（1 トランザクション）の最大転送サイズを指定するものです。

設定可能な最大データサイズは 1023 です。

ユーザシステムに合わせて設定してください。

例）256 バイトを指定する場合

```
#define  USBH0_OHCI_ISO_MAX_PACKET_SIZE  256
```

5.4.6 OHCI タイムアウト時間指定

OHCI タイムアウト時間を msec 単位で指定してください。

例）3 秒を指定する場合

```
#define  USBH0_OHCI_TIMEOUT  3000
```

5.5 ターゲットペリフェラルリスト（TPL）

USB-BASIC-F/W は、サポートする USB デバイスを指定することができます。

サポートする USB デバイスの指定は、ターゲットペリフェラルリスト(TPL)に、その USB デバイスのプロダクト ID およびベンダ ID をセットで指定してください。

なお、サポートする USB デバイスを特定する必要が無い場合は、ベンダ ID に USB_NOVENDOR、プロダクト ID に USB_NOPRODUCT を設定してください。

以下に、TPL の作成例を示します。

```
const uint16_t usbh0_hmsc_devicetpl [] =
{
   2,                   /* Number of list */
   0,                   /* Reserved */
   0xXXXX,              /* Vendor ID */
   0xXXXX,              /* Product ID */
};
```

[Note]

1. TPL は、アプリケーションプログラム用および Hub 用(r_usbh0_hhubsys.c)の 2 つが用意されています。コンプライアンステスト対応設定を選択している時は、この 2 つの TPL には、USB_NOVENDOR と USB_NOPRODUCT を設定しないようにしてください。(コンプライアンス対応設定については、”5.1 定義情報ファイル”の 5.1.3 を参照してください。)
2. コンプライアンステスト対応設定を選択している時、Hub 用 TPL には、コンプライアンステストで使用する Hub の VENDOR ID と PRODUCT ID を設定してください。

5.6 構造体定義

構造体について説明します。r_usb_basic_if.h ファイルで構造体定義をしています。

5.6.1 st_usbh0_hcdreg_t 構造体

st_usbh0_hcdreg_t は、HDCD の情報を登録するための構造体です。

st_usbh0_hcdreg_t に登録したコールバック関数は、デバイスステートの変化などで実行されます。

表 5.1 に st_usbh0_hcdreg_t 構造体のメンバを示します。

表 5.1 st_usbh0_hcdreg_t 構造体のメンバ

型	メンバ名	機能
uint16_t	rootport	接続されているポート番号が登録されます。
uint16_t	devaddr	デバイスアドレスが登録されます。
uint16_t	devstate	デバイスの接続状態が登録されます。
uint16_t	ifclass	HDCD のインタフェースクラスコードを登録してください。
uint16_t	*tpl	HDCD が動作するターゲットペリフェラルリストを登録してください。 「5.4 ターゲットペリフェラルリスト（TPL）」を参照してください。
usbh0_cb_check_t	classcheck	HDCD チェック時に起動する関数を登録してください。 TPL が一致した場合に呼び出します。
usbh0_cb_t	devconfig	Configured 状態遷移時に起動する関数を登録してください。 SET_CONFIGURATION リクエストが終了した場合に呼び出します。
usbh0_cb_t	devdetach	デタッチ状態遷移時に起動する関数を登録してください。
usbh0_cb_t	devsuspend	サスペンド状態遷移時に起動する関数を登録してください。
usbh0_cb_t	devresume	レジューム状態遷移時に起動する関数を登録してください。

5.6.2 st_usbh0_setup_t 構造体

USB リクエストのデータ構造体です。

表 5.2 に、st_usbh0_setup_t 構造体のメンバを示します。

表 5.2 st_usbh0_setup_t 構造体のメンバ

型	メンバ名	機能
uint16_t	type	bRequest[b15-b8], bmRequestType[b7-b0] を設定してください。
uint16_t	value	wValue を設定してください。
uint16_t	index	wIndex を設定してください。
uint16_t	length	wLength を設定してください。
uint16_t	devaddr	デバイスアドレス

5.6.3 usbh0_utr 構造体

USB 通信およびタスクメッセージ用の構造体です。

R_USBH0_HstdTransferStart ()の引数に設定することで、接続されたデバイスと USB 通信ができます。

表 5.3 に、usbh0_utr 構造体のメンバを示します。

表 5.3 usbh0_utr 構造体のメンバ

型	メンバ名	機能
uint16_t	msginfo	USB-BASIC-F/W のタスクが使用するメッセージ情報を設定してください。 USB 通信をする場合は設定不要です。
uint16_t	keyword	通信するパイプ番号を設定してください。
uint16_t	result	USB 通信結果が格納されます。 「6.3 転送結果の通知」を参照してください。
usbh0_utr_cb_t	complete	USB 通信が完了した場合に実行したい関数アドレスを設定してください。 「5.19 コールバック関数」を参照してください。
void	*tranadr	USB 通信バッファアドレスを設定してください。 ・受信または ControlRead 転送 受信データを格納するバッファのアドレスを指定します。 ・送信または ControlWrite 転送 送信データが格納してあるバッファアドレスを指定します。 ・NoDataControl 転送 指定されても無視します。
uint32_t	tranlen	USB 通信データ長を設定してください。 ・受信または ControlRead 転送 受信データ長を指定します。USB 通信終了後に、実際に受信したデータ長を引いた値に更新されます。 ・送信または ControlWrite 転送 送信データ長を指定します。 ・NoDataControl 転送 "0"を指定してください。
st_usbh0_setup_t	*setup	コントロール転送時の Setup パケット情報を設定してください。 「5.5.2 st_usbh0_setup_t 構造体」および「6.6 コントロール転送」を参照してください。

5.7 API 関数

MGR、HUBCDおよびHDCDは、API関数でH/Wの制御要求をします。

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。

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。

すべての API 呼び出しとそれをサポートするインターフェース定義は r_usbh0_basic_if.h に記載しています。

表5-4にAPI関数一覧を示します。

表 5-4　API 関数一覧

関数名	ポート	説明
R_USBH0_HstdTransferStart R_USBH1_HstdTransferStart	0 1	データ転送実行要求
R_USBH0_HstdTransferEnd R_USBH1_HstdTransferEnd	0 1	データ転送強制終了要求
R_USBH0_HstdDriverRegistration R_USBH1_HstdDriverRegistration	0 1	HDCD 登録
R_USBH0_HstdReturnEnuMGR R_USBH1_HstdReturnEnuMGR	0 1	クラスチェック完了通知
R_USBH0_HstdChangeDeviceState R_USBH1_HstdChangeDeviceState	0 1	接続デバイスの状態変更要求
R_USBH0_HstdSetPipe R_USBH1_HstdSetPipe	0 1	パイプ情報設定
R_USBH0_HstdGetPipeID R_USBH1_HstdGetPipeID	0 1	パイプ番号取得
R_USBH0_HstdClearPipe R_USBH1_HstdClearPipe	0 1	パイプ情報クリア
R_USBH0_HstdMgrOpen R_USBH1_HstdMgrOpen	0 1	MGR タスク起動
R_USBH0_HstdMgrClose R_USBH1_HstdMgrClose	0 1	MGR タスク停止
R_USBH0_HstdMgrTask R_USBH1_HstdMgrTask	0 1	MGR タスク
R_USBH0_HhubTask R_USBH1_HhubTask	0 1	HUB タスク

5.8 R_USBH0_HstdTransferStart

データ転送実行要求

形式

usbh0_er_t R_USBH0_HstdTransferStart(st_usbh0_utr_t * utr)

引数

*utr	USB 通信構造体

戻り値

USBH0_OK	成功
USBH0_ERROR	失敗

解説

本 API は、utr に格納されている転送情報をもとにデータ転送要求をします。
データ転送終了（指定データサイズ、ショートパケット受信、エラー発生）時にコールバック関数が呼び出されます。このコールバック関数の引数には、送受信の残りデータ長、ステータス、転送終了の情報が設定されています。

補足

1. 本 API はユーザアプリケーションまたはクラスドライバから呼び出してください。
2. 引数である構造体のメンバには、パイプ番号、転送データの先頭アドレス、転送データ長、終了時のコールバック関数を設定してください。
3. エニュメレーション中のデバイスが存在する場合、そのデバイスに対するエニュメレーションが完了するまで本 API の戻り値は USBH0_ERROR となります。

使用例

```
usbh0_utr    usb_smp_trn_Msg[USBH0_MAXPIPE_NUM + 1];

usbh0_er_t   usb_smp_task(usbh0_utr_cb_t complete, uint16_t pipe, uint32_t size,
uint8_t *table)
{
   :
  /* Set transfer information */
  trn_msg[pipe].keyword    = pipe;      /* Pipe no. */
  trn_msg[pipe].tranadr    = table;     /* Pointer to data buffer */
  trn_msg[pipe].tranlen    = size;      /* Transfer size */
  trn_msg[pipe].complete   = complete;  /* Callback function */

  /* Transfer start request */
  err = R_USBH0_HstdTransferStart(&trn_msg[pipe]);

  return err;
   :
}
```

5.9 R_USBH0_HstdTransferEnd

データ転送強制終了要求

形式

usbh0_er_t R_USBH0_HstdTransferEnd(uint16_t pipe_id)

引数

pipe_id	パイプ番号

戻り値

USBH0_OK	成功
USBH0_ERROR	失敗

解説

本 API は、指定したパイプのデータ転送の強制終了要求を行います。

補足

1. 本 API はユーザアプリケーションまたはクラスドライバから呼び出してください。

使用例

```
void   usb_smp_task(void)
{
  uint16_t pipe;
    :
  pipe    = USB_PIPEx

  /* Transfer end request */
  err     = R_USBH0_HstdTransferEnd(pipe);

  return err;
    :
}
```

5.10 R_USBH0_HstdDriverRegistration

Host device class driver (HDCD) 登録

形式

void	R_USBH0_HstdDriverRegistration(st_usbh0_hcdreg_t *callback)

引数

*callback	クラスドライバ構造体

戻り値

―	―

解説

クラスドライバ構造体に登録した HDCD の情報を HCD に登録します。HCD が管理する登録ドライバ数を更新し、新しい領域に HDCD の登録をします。登録完了後、初期化コールバック関数が実行されます。

補足

1. 初期設定時、本 API をユーザアプリケーションプログラムまたはクラスドライバから呼び出してください。
2. 登録する情報は、「表 5.1 st_usbh0_hcdreg_t 構造体のメンバ」を参照してください。

使用例

```
void   usb_smp_registration(void)
{
    st_usbh0_hcdreg_t    driver;

    /* Interface Class */
    driver.ifclass      = (uint16_t)USB_IFCLS_VEN;
    /* Target peripheral list */
    driver.tpl          = (uint16_t*)&usb_gapl_devicetpl;
    /* Driver check */
    driver.classcheck   = &usb_hvendor_class_check;
    /* Device configured */
    driver.devconfig    = &usb_hvendor_apl_init;
    /* Device detach */
    driver.devdetach    = &usb_hvendor_apl_close;
    /* Device suspend */
    driver.devsuspend   = &usb_hvendor_apl_suspend;
    /* Device resume */
    driver.devresume    = &usb_hvendor_apl_resume;

    /* Driver registration */
    R_USBH0_HstdDriverRegistration(&driver);
}
```

5.11 R_USBH0_HstdReturnEnuMGR

クラスチェック完了通知

形式

void	R_USBH0_HstdReturnEnuMGR(uint16_t cls_result)

引数

cls_result	クラスチェック結果

戻り値

―	―

解説

MGR にエニュメレーション処理の継続を要求します。MGR は本関数の引数を解析し、接続された USB デバイスを Configuard ステートに遷移させるか判断します。

cls_result には、クラスチェック結果(USBH0_OK または USBH0_ERROR)を指定してください。

補足

1. 本 API は、クラスチェック処理時に呼び出してください。

使用例

```
void   usb_smp_task(void)
{
    :
  (Enumeration processing)
    :
  R_USBH0_HstdReturnEnuMGR(USBH0_OK);
}
```

5.12 R_USBH0_HstdChangeDeviceState

接続されているデバイスの状態変更要求

形式

usbh0_er_t R_USBH0_HstdChangeDeviceState(usbh0_utr_cb_t complete,
int16_t msginfo, uint16_t member)

引数

complete	コールバック関数
msginfo	メッセージ情報
member	デバイスアドレス/パイプ番号

戻り値

USBH0_OK	成功
USBH0_ERROR	失敗

解説

msginfo に以下の値を設定し、本 API をコールすることで、接続されている USB デバイスの状態変更を HCD に要求します。

msginfo	概要
USBH0_DO_GLOBAL_SUSPEND	リモートウェイクアップ許可付きサスペンドへの遷移要求 (Hub 以下のデバイスに対してデバイス指定サスペンド要求)
USBH0_DO_GLOBAL_RESUME	グローバルレジューム実行要求 （Port 以下の全ての接続デバイスへのレジューム）
USBH0_DO_CLR_STALL	STALL クリア要求

補足

1. 本 API は、ユーザアプリケーションプログラムまたはクラスドライバから呼び出してください。
2. USBH0_DO_CLR_STALL を引数 msginfo に設定するとき、引数 member には、パイプ番号を設定してください。

使用例

```
void   usb_smp_task(void)
{
   R_USBH0_HstdChangeDeviceState(&usb_smp_callback, USBH0_DO_GLOBAL_SUSPEND, devaddr);
}
```

5.13 R_USBH0_HstdSetPipe

パイプ情報設定

形式

usbh0_er_t R_USBH0_HstdSetPipe(uint16_t **table)

引数

**table	クラスチェック情報テーブルポインタ（表 5-6 参照）

戻り値

USBH0_OK	成功
USBH0_ERROR	失敗

解説

受信したディスクリプタを解析してパイプ情報を設定します。

補足

1. 本 API は、クラスチェック処理時に呼び出してください。
2. パイプ情報領域に空きが無い場合は、USBH0_ERROR を返します。

使用例

```
void   usb_smp_classcheck(uint16_t **table)
{
   R_USBH0_HstdSetPipe(table);
}
```

5.14 R_USBH0_HstdGetPipeID

パイプ番号取得

形式

uint16_t R_USBH0_HstdGetPipeID(uint16_t devaddr, uint8_t type,
uint8_t direction, uint8_t ifnum)

引数

devaddr	デバイスアドレス
type	エンドポイント属性(USBH0_EP_ISO / USBH0_EP_BULK / USBH0_EP_INT)
direction	エンドポイント方向(USBH0_EP_IN / USBH0_EP_OUT)
ifnum	インターフェース番号

戻り値

uint16_t	パイプ番号

解説

引数で指定した条件に合致したパイプ番号を返却します。

補足

1. 本 API は、ユーザアプリケーションプログラムまたはクラスドライバから呼び出してください。
2. R_USBH0_HstdSetPipe()でパイプ情報を設定した後に呼び出してください。
3. インターフェース番号に 0xFF を指定すると、インターフェース番号を無視して検索します。

使用例

```
void usb_smp_configured(uint16_t devaddr)
{
    usb_gsmp_pipe = R_USBH0_HstdGetPipeID(devaddr, USBH0_EP_BULK, USBH0_EP_IN, 0);
}
```

5.15 R_USBH0_HstdClearPipe

パイプ情報クリア

形式

usbh0_er_t R_USBH0_HstdClearPipe(uint16_t devaddr)

引数

devaddr	デバイスアドレス

戻り値

USBH0_OK	成功

解説

引数で指定したデバイスアドレスのパイプ設定をクリアして、領域を開放します。

補足

1. 本 API は、ユーザアプリケーションプログラムまたはクラスドライバから呼び出してください。

使用例

```
void usb_smp_detach(uint16_t devaddr)
{
    R_USBH0_HstdClearPipe(devaddr);
}
```

5.16 R_USBH0_HstdMgrOpen

MGR タスク起動

形式

usbh0_er_t R_USBH0_HstdMgrOpen(void)

引数

ー ー

戻り値

USBH0_OK 正常終了

解説

MGR の登録状態を初期化します。
BSP_CFG_RTOS_USED = 1 のとき、ポート 0 に対するセマフォを生成します。
戻り値は常に USBH0_OK になります。

補足

1. 初期設定時、ユーザアプリケーションプログラムで本 API を呼び出してください。
2. MGR タスク起動後、本 API を呼び出さないでください。

使用例

```
void   usb_smp_task(void)
{
    R_USBH0_HstdMgrOpen();
}
```

5.17 R_USBH0_HstdMgrClose

MGR タスク停止

形式

void	R_USBH0_HstdMgrClose(void)

引数

―	―

戻り値

―	―

解説

BSP_CFG_RTOS_USED = 1 のとき、ポート 0 に対するセマフォを削除します。

補足

1. MGR タスク起動後、ユーザアプリケーションプログラムで本 API を呼び出してください。
2. MGR タスク起動前に、本 API を呼び出さないでください。

使用例

```
void   usb_smp_task(void)
{
    R_USBH0_HstdMgrOpen();
    :
    R_USBH0_HstdMgrClose();
}
```

5.18 R_USBH0_HstdMgrTask

MGR タスク

形式

void	R_USBH0_HstdMgrTask(void)

引数

―	―

戻り値

―	―

解説

ホスト機能選択時、HCD と HDCD 間の機能を補完します。

補足

1. スケジューラ処理をするループ内で呼び出してください。

使用例

```
void   usb_smp_mainloop(void)
{
    while(1)
    {
        /* Start scheduler */
        R_USBH0_CstdScheduler();
        /* Check flag */
        if(USBH0_FLGSET == R_USBH0_CstdCheckSchedule())
        {
            R_USBH0_HstdMgrTask();
        }
    }
}
```

5.19 R_USBH0_HhubTask

HUB タスク

形式

void	R_USBH0_HhubTask(void)

引数

―	―

戻り値

―	―

解説

接続された USB ハブのダウンポートの状態を管理し、HCD と HDCD 間の機能を補完します。

補足

1. スケジューラ処理をするループ内で本 API を呼び出してください。

使用例

```
void   usb_smp_mainloop(void)
{
  while(1)
  {
  /* Start scheduler */
  R_USBH0_CstdScheduler();
  /* Check flag */
  if(USBH0_FLGSET == R_USBH0_CstdCheckSchedule())
  {
       :
    R_USBH0_HhubTask();
       :
  }
}
```

5.20 コールバック関数

コールバック関数を使用することで、イベントをポーリングする必要がなくなります。

例えば、ユーザアプリケーションが HCD に対してデータ転送要求をする場合、コールバック関数を登録することで、デバイスとのデータ通信を行った結果を受け取ります。

以下に、コールバック関数を示します。

表 5.5 R_USBH0_HstdTransferStart コールバック

名称	データ転送要求のコールバック関数		
呼出形式	void (*usbh0_utr_cb_t)(st_usbh0_utr_t*)		
引数	st_usbh0_utr_t*	*p_utr	R_USBH0_HstdTransferStart()関数の引数の st_usbh0_utr_t ポインタ
戻り値	―	―	―
説明	データ転送終了（アプリケーションから指定されたサイズのデータ転送終了およびショートパケット受信等によるトランスファー終了）でコールされます。		
注意事項			

表 5.6 classcheck コールバック

名称	classcheck コールバック関数		
呼出形式	void (*usbh0_cb_check_t)(uint16_t **)		
引数	uint16_t	**table	table[0]デバイスディスクリプタ先頭アドレス table[1]コンフィグレーションディスクリプタ先頭アドレス table[2]インタフェースディスクリプタ先頭アドレス table[3]ディスクリプタチェック結果 table[4]HUB スペック table[5]ポート番号(未使用) table[6]通信速度(未使用) table[7]デバイスアドレス
戻り値	―	―	―
説明	レジストレーションで登録したクラスコード（st_usbh0_hcdreg_t 構造体のメンバ ifclass）と受信したディスクリプタのクラスコードが一致した場合に、st_usbh0_hcdreg_t 構造体のメンバ classcheck に登録した関数がコールされます。 table[3]：チェック結果 USBH0_OK ：HDCD 動作可能 USBH0_ERROR：HDCD 動作不能 のいずれかで HDCD が動作可能か否かを応答してください。 table[4]：HUB スペック(HUB ドライバ用) USBH0_FSHUB：Full-Speed HUB の場合 USBH0_HSHUBS：Hi-Speed HUB(Single)の場合 USBH0_HSHUBM：Hi-Speed HUB(Multi)の場合 table[7]：デバイスアドレス Host が Set Address で割り当てたデバイスアドレス		
注意事項			

表 5.7　ステートチェンジ時のコールバック

名称	ステートチェンジ時のコールバック関数		
呼出形式	void (*usbh0_cb_t)(uint16_t)		
引数	uint16_t	devaddr	接続されたデバイスのアドレス
戻り値	－	－	－
説明	ステートチェンジが発生した場合に、レジストレーションで st_usbh0_hcdreg_t 構造体のメンバに登録された関数がコールされます。 devconfig：Set_Configuration リクエスト発行時に実行されます。 devdetach：デタッチ検出時に実行されます。 devsuspend：サスペンド遷移時に実行されます。 devresume：レジューム遷移時に実行されます。		
注意事項			

6. USB 通信

6.1 パイプ

パイプは、USB デバイスのエンドポイントとの論理的な接続路です。

コントロール転送をするデフォルトパイプ（パイプ番号 0）は、USB-BASIC-F/W 内部で設定されます。

データ転送をするためには、クラスチェック処理時に API 関数の R_USBH0_HstdSetPipe()をコールして USB デバイスのディスクリプタからパイプ情報を設定する必要があります。パイプ情報は、エンドポイントディスクリプタ毎にパイプ番号 1 から順に割り振られます。

パイプの最大数はコンフィグファイルにより変更できます。5.1.2 章を参照してください。

パイプに設定する情報は、下記の通りです。

- インターフェース番号
- エンドポイント番号
- エンドポイント転送タイプ
- エンドポイント方向
- マックスパケットサイズ
- デバイスアドレス

データ転送要求には、パイプ番号を指定する必要があります。

API 関数の R_USBH0_HstdGetPipeID()をコールすることでパイプ番号を取得できます。

デバイスが切断された場合には、パイプ情報領域を開放する為パイプ情報をクリアする必要があります。

API 関数の R_USBH0_HstdClearPipe()をコールすることでパイプ情報をクリアできます。

6.2 データ転送要求

API 関数の R_USBH0_HstdTransferStart()の引数に転送情報を設定した st_usbh0_utr_t 構造体を渡すことで、データ転送要求ができます。

st_usbh0_utr_t 構造体の詳細は、「5.5.3 usbh0_utr 構造体」を参照してください。

6.3 転送結果の通知

USB-BASIC-F/W は、データ転送が終了すると HDCD に対してデータ転送要求時に登録されたコールバック関数でデータ転送の終了を通知します。

転送結果は、st_usbh0_utr_t 構造体のメンバ（result）に格納されます。

以下に、通信結果を示します。

通知内容	詳細
USBH0_CTRL_END	Control 転送が正常に終了した場合
USBH0_DATA_NONE	データ送信が正常終了した場合
USBH0_DATA_OK	データ受信が正常終了した場合
USBH0_DATA_SHT	データ受信は正常終了したが指定されたデータ長未満で終了した場合
USBH0_DATA_OVR	受信データがサイズオーバーした場合
USBH0_DATA_ERR	無応答もしくはオーバ／アンダーランエラーを検出した場合
USBH0_DATA_STALL	STALL 応答もしくは MaxPacketSize エラーを検出した場合
USBH0_DATA_STOP	データ転送を強制終了した場合

6.4 転送のリトライ

USB-BASIC-F/W は、各パイプの無応答に対して通信再実行をします。無応答を検出した場合は HDCD に対して USBH0_DATA_ERR を応答します。

6.5 受信時の注意事項

ショートパケットを受信した場合は、残り受信予定のデータ長を st_usbh0_utr_t 構造体の tranlen に格納して転送終了します。受信したデータがバッファサイズより大きい場合は、バッファサイズまでのデータを FIFO バッファから読み出して転送終了します。

6.6 コントロール転送

USB-BASIC-F/W は「4.5.2 USB 標準リクエスト」に記載されていないリクエスト（ベンダ、クラスリクエスト）をデバイスに送信する場合、st_usbh0_utr_t 構造体のメンバに以下の設定をした後に、R_USBH0_HstdTransferStart()を呼び出す必要があります。

- keyword ：パイプ番号（USB_PIPE0）
- tranadr ：データバッファ（データステージ）
- tranlen ：転送サイズ
- setup ：セットアップデータ
- complete ：コールバック関数

データ転送終了時に complete に設定したコールバック関数が呼び出され、転送結果が通知されます。

（「6.3 転送結果の通知」を参照してください）

以下に、ベンダリクエスト送信例を示します。

＜ベンダリクエスト送信例＞

```
uint8_t             usb_gsmp_VendorRequestData[16];
st_usbh0_setup_t    usb_gsmp_VendorRequest;
usbh0_utr           usb_gsmp_ControlUtr;
usbh0_utr_cb_t      usb_gsmp_VendorRequestCb;

usbh0_er_t          usb_hsmp_VendorRequestProcess(void)
{
   usbh0_er_t err;

   /* Set setup data */
   usb_gsmp_VendorRequest.type          = ((bRequest << 8) | bmRequestType);
   usb_gsmp_VendorRequest.value         = wValue;
   usb_gsmp_VendorRequest.index         = wIndex;
   usb_gsmp_VendorRequest.length        = wLength;
   usb_gsmp_VendorRequest.devaddr       = devaddr;

   /* Set transfer pipe (default pipe) */
   usb_gsmp_ControlUtr.keyword   = USBH0_PIPE0;
   /* Set transmit data */
   usb_gsmp_ControlUtr.tranadr   = usb_gsmp_VendorRequestData;
   /* Set transfer size */
   usb_gsmp_ControlUtr.tranlen   = usb_gsmp_VendorRequest.length;
   /* Set setup command */
   usb_gsmp_ControlUtr.setup     = &usb_gsmp_VendorRequest;
   /* Set callback function */
   usb_gsmp_ControlUtr.complete  = &usb_gsmp_VendorRequestCb;

   /* Data transmission request */
   err                           = R_USBH0_HstdTransferStart(&usb_gsmp_ControlUtr);

   return     err;
}
```

6.7 データ転送

USB-BASIC-F/W は、st_usbh0_utr_t 構造体のメンバに以下の設定をした後に、R_USBH0_HstdTransferStart()を呼び出す必要があります。

- keyword ：パイプ番号
- tranadr ：データバッファ
- tranlen ：転送サイズ
- complete ：コールバック関数

データ転送終了時に complete に設定したコールバック関数が呼び出され、転送結果が通知されます。

（「6.3 転送結果の通知」を参照してください）

以下に、BulkIn 転送例を示します。

＜BulkIn 転送例＞

```
uint8_t             usb_gsmp_VendorBulkInData[512];
usbh0_utr           usb_gsmp_DataUtr;
usbh0_utr_cb_t      usb_gsmp_VendorBulkInCb;

usbh0_er_t          usb_hsmp_VendorBulkInProcess(uint16_t devaddr)
{
   usbh0_er_t err;

   /* Set transfer pipe */
   usb_gsmp_DataUtr.keyword      = R_usb_htsd_GetPipeID(devaddr, USBH0_EP_BULK,
USBH0_EP_IN, 0);
   /* Set transmit data */
   usb_gsmp_DataUtr.tranadr      = usb_gsmp_VendorBulkInData;
   /* Set transfer size */
   usb_gsmp_DataUtr.tranlen      = 512;
   /* Set callback function */
   usb_gsmp_DataUtr.complete     = &usb_gsmp_VendorBulkInCb;

   /* Data transmission request */
   err                           = R_USBH0_HstdTransferStart(&usb_gsmp_DataUtr);

   return     err;
}
```

7. スケジューラ

7.1 概要

スケジューラは、タスク優先度に従いタスクスケジューリングをします。

スケジューラの特徴を以下に示します。

- 各タスクや H/W の要求をタスクの優先順位に従って管理する。
- 優先順位が同じタスクから複数の要求が発生した場合、FIFO 構造で要求を処理する。
- 要求の終了はコールバック関数によりタスクに通知されるため、スケジューラを変更することなくユーザシステムに適合したクラスドライバの実装が可能。

7.2 スケジューラマクロ

ユーザが使用可能なスケジューラマクロを表 7.1 に示します。

表 7.1 スケジューラマクロ

スケジューラマクロ	登録関数	ポート	概要
USBH0_SND_MSG USBH1_SND_MSG	usbh0_hstd_snd_msg usbh1_hstd_snd_msg	0 1	メッセージ BOX を指定して、メッセージを送信します。
USBH0_WAI_MSG USBH1_WAI_MSG	usbh0_hstd_wai_msg usbh1_hstd_wai_msg	0 1	スケジューラを指定回数実行後に USBH0_SND_MSG を実行します。
USBH0_RCV_MSG USBH1_RCV_MSG	usbh0_hstd_rec_msg usbh1_hstd_rec_msg	0 1	指定したメール BOX にメッセージ受信があるか確認します。
USBH0_PGET_BLK USBH1_PGET_BLK	usbh0_hstd_pget_blk usbh1_hstd_pget_blk	0 1	メッセージを格納する領域を確保します。
USBH0_REL_BLK USBH1_REL_BLK	usbh0_hstd_rel_blk usbh1_hstd_rel_blk	0 1	メッセージを格納する領域を開放します。

7.3 スケジューラ API 関数

スケジューラで使用する API 関数一覧を表 7.2 に示します。

表 7.2 スケジューラ API 関数一覧

関数名	ポート	説明
usbh0_hstd_snd_msg() usbh1_hstd_snd_msg()	0 1	優先テーブルに処理要求を送信する。
usbh0_hstd_rec_msg() usbh1_hstd_rec_msg()	0 1	優先テーブルに処理要求があるか確認する。
usbh0_hstd_pget_blk() usbh1_hstd_pget_blk()	0 1	処理要求を格納する領域を確保する。
usbh0_hstd_rel_blk() usbh1_hstd_rel_blk()	0 1	処理要求を格納する領域を解放する。
R_USBH0_CstdScheduler() R_USBH1_CstdScheduler()	0 1	各タスクからの処理要求を管理する。
R_USBH0_CstdSetTaskPri() R_USBH1_CstdSetTaskPri()	0 1	タスクの優先度を設定する。
R_USBH0_CstdCheckSchedule() R_USBH1_CstdCheckSchedule()	0 1	登録されたタスクに処理要求があるか確認する。

7.4 usbh0_hstd_snd_msg

処理要求を送信

形式

usbh0_er_t usbh0_hstd_snd_msg(uint8_t id, usbh0_msg_t *mess)

引数

id	メッセージを送信するタスク ID
*mess	送信するメッセージ

戻り値

USBH0_OK	正常終了
USBH0_ERROR	タスク ID が未設定 タスク優先度が未設定 優先テーブルに空きがなく通知不可

解説

優先テーブルに処理要求を発信します。本 API は USBH0_SND_MSG マクロで定義されています。

補足

1. 割り込み関数以外のユーザアプリケーションプログラムまたはクラスドライバで、本 API が登録されているスケジューラマクロ(USBH0_SND_MSG)を呼び出してください。
2. 送信するメッセージには、メモリバッファの先頭アドレスを指定してください。
3. サンプルプログラムでは、USBH0_PGET_BLK マクロで取得したメモリブロックの先頭アドレスを送信メッセージとして指定しています。

使用例

```
void   usb_smp_task()
{
  usbh0_utr          *p_blf;
  usbh0_er_t         err;
    :
  /* Secure a message storage area */
  err = USBH0_PGET_BLK(USB_SMP_MPL, &p_blf);
  if(err == USBH0_OK)
  {
    /* Set up messages */
    p_blf->msginfo      = USB_SMP_REQ;
    p_blf->keyword      = keyword;
    p_blf->complete     = complete;
  }
  /* Send messages */
  err = USBH0_SND_MSG(USB_SMP_MBX, (usbh0_msg_t*)p_blf);
  if(err != USBH0_OK)
  {
    /* Error processing */
  }
    :
}
```

7.5 usbh0_hstd_rec_msg

優先テーブルに処理要求があるか確認

形式

usbh0_er_t usbh0_hstd_rec_msg(uint8_t id, usbh0_msg_t** mess)

引数

id	メッセージを受信するタスク ID
*mess	受信するメッセージ

戻り値

USBH0_OK	正常終了
USBH0_ERROR	タスク ID が未設定

解説

優先テーブルにタスク ID の処理要求があるか確認します。本関数は USBH0_RCV_MSG に登録されています。

補足

1. ユーザアプリケーションプログラムまたはクラスドライバで、本 API が登録されているスケジューラマクロを呼び出してください。
2. R_USBH0_CstdCheckSchedule ()関数の戻り値が USBH0_FLGCLR の場合には、本 API を実行しないでください。

使用例

```
void   usb_smp_task()
{
  usbh0_utr         *mess;
  usbh0_er_t        err;
    :
  /* Check the message */
  err = USBH0_RCV_MSG(USB_SMP_MBX, (usbh0_msg_t**)&mess);
  if(err == USBH0_OK)
  {
    /* Processing for the message */
  }
    :
}
```

7.6 usbh0_hstd_pget_blk

処理要求を格納する領域を確保する

形式

usbh0_er_t	usbh0_hstd_pget_blk(uint8_t id, st_usbh0_utr_t **blk)

引数

id	領域を確保するタスク ID
**blk	確保する領域のポインタ

戻り値

USBH0_OK	領域確保完了
USBH0_ERROR	タスク ID が未設定 領域確保失敗

解説

メッセージを格納する領域を確保します。本 API では、1 ブロックあたり 40 バイト確保します。
本関数は USBH0_PGET_BLK に登録されています。

補足

1. ユーザアプリケーションプログラムまたはクラスドライバで、本 API が登録されているスケジューラマクロを呼び出してください。

使用例

```
void   usb_smp_task()
{
  usbh0_utr          *p_blf;
  usbh0_er_t         err;
    :
  /*Secure a message storage area */
  err = USBH0_PGET_BLK(USB_SMP_MPL, &p_blf);
  if(err == USBH0_OK)
  {
    /* Set up messages */
    p_blf->msginfo      = USB_SMP_REQ;
    p_blf->keyword      = keyword;
    p_blf->complete     = complete;
  }
  /* Send messages */
  err = USBH0_SND_MSG(USB_SMP_MBX, (usbh0_msg_t*)p_blf);
  if(err != USBH0_OK)
  {
    /* Error processing */
  }
    :
}
```

7.7 usbh0_hstd_rel_blk

処理要求を格納する領域を解放

形式

usbh0_er_t usbh0_hstd_rel_blk(uint8_t id, st_usbh0_utr_t* blk)

引数

id	領域を解放するタスク ID
*blk	解放する領域のポインタ

戻り値

USBH0_OK	領域解放完了
USBH0_ERROR	タスク ID が未設定 領域解放失敗

解説

メッセージを格納する領域を開放します。本関数は USBH0_REL_BLK に登録されています。

補足

1. ユーザアプリケーションプログラムまたはクラスドライバで、本 API が登録されているスケジューラマクロを呼び出してください。

使用例

```
void   usb_smp_task()
{
  usbh0_er_t        err;
  usbh0_utr         *mess;
    :
  /* Release the message storage area */
  err = USBH0_REL_BLK(USB_SMP_MPL, mess);
  if(err != USBH0_OK)
  {
    /* Error processing */
  }
    :
}
```

7.8 R_USBH0_CstdScheduler

処理要求を管理

形式

void	R_USBH0_CstdScheduler(void)

引数

―	―

戻り値

―	―

解説

各タスクからの処理要求を管理します。

補足

1. スケジューラ処理をするループ内で呼び出してください。

使用例

```
void   usb_smp_mainloop(void)
{
  while(1)
  {
  /* Check flag */
  if(USBH0_FLGSET == R_USBH0_CstdCheckSchedule())
  {
       :
    /* Task processing */
       :
  }
  /* Start scheduler */
  R_USBH0_CstdScheduler();
       :
}
```

7.9 R_USBH0_CstdSetTaskPri

タスクの優先度を設定

形式

void	R_USBH0_CstdSetTaskPri(uint8_t tasknum, uint8_t pri)

引数

tasknum	タスク ID
pri	タスク優先度

戻り値

―	―

解説

タスクの優先度を設定します。第一引数(tasknum)で指定したタスクに、第二引数で指定した優先度(pri)を設定します。

補足

1. 本 API は、初期設定時にユーザアプリケーションプログラムで呼び出してください。タスク ID の優先度が設定されることで、メッセージの送受信が可能になります。

使用例

```
void usb_smp_driver_start(void)
{
  /* Set the priority of the sample task to 4 */
  R_USBH0_CstdSetTaskPri(USB_SMP_TSK, USB_PRI_4);
}
```

7.10 R_USBH0_CstdCheckSchedule

処理要求があるか確認

形式

uint8_t R_USBH0_CstdCheckSchedule(void)

引数

― ―

戻り値

USBH0_FLGSET 処理要求あり
USBH0_FLGCLR 処理要求なし

解説

登録されたタスクに受信メッセージがあるか確認します。

補足

1. スケジューラ処理をするループ内で本 API を呼び出してください。

使用例

```
void   usb_smp_mainloop(void)
{
  while(1)
  {
  /* Check flag */
  if(USBH0_FLGSET == R_USBH0_CstdCheckSchedule())
  {
       :
    /* Task processing */
       :
  }
  /* Start scheduler */
  R_USBH0_CstdScheduler();
       :
}
```

8. 参考ドキュメント

ユーザーズマニュアル：ハードウェア

RZ/A2M グループ ユーザーズマニュアル ハードウェア編
（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。）

RTK7921053C00000BE（RZ/A2M CPU ボード）ユーザーズマニュアル
（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。）

RTK79210XXB00000BE（RZ/A2M SUB ボード）ユーザーズマニュアル
（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。）

Arm Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C
（最新版を Arm ホームページから入手してください。）

Arm Cortex™-A9 Technical Reference Manual Revision: r4p1
（最新版を Arm ホームページから入手してください。）

Arm Generic Interrupt Controller Architecture Specification - Architecture version2.0
（最新版を Arm ホームページから入手してください。）

Arm CoreLink™ Level 2 Cache Controller L2C-310 Technical Reference Manual Revision: r3p3
（最新版を Arm 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：統合開発

統合開発環境 e2 studio のユーザーズマニュアルは、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。
（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。）

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2019 年 4 月 15 日	-	初版発行
1.10	2019 年 5 月 17 日	6	表 2.1　動作確認条件 コンパイラオプション"-mthumb-interwork"を削除
1.20	2019 年 12 月 17 日	6	表 2.1　動作確認条件 コンパイラオプション"-g3"を"-None"に変更 FreeRTOS/OSLess 双方のサポート

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。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)

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。