Renesas USB MCU

USB Host and Peripheral Basic Mini Firmware

R01AN0326JJ0215
Rev.2.15
Mar 28, 2016

要旨

本資料はRenesas USB MCUを使用した USB インタフェース制御用サンプルプログラムであるUSB Host and Peripheral Basic Mini Firmwareのアプリケーションノートです。.

動作確認デバイス

R8C/3MU, R8C/34U, R8C/3MK, R8C/34K, RL78/G1C,RL78/L1C

動作確認デバイスと同様の USB モジュールを持つ他の MCU でも本プログラムを使用することができます。このアプリケーションノートのご使用に際しては十分な評価を行ってください。

目次

1. 資料概要

本資料は、Renesas USB MCUを使用したUSBインタフェース制御用サンプルプログラムであるUSB Host and Peripheral Basic Mini Firmwareのアプリケーションノートです。

本書は必ず1.2 関連ドキュメントと併用してご利用ください。

1.1 機能と特徴

USB Host and Peripheral Basic Mini Firmwareは Universal Serial Bus Specification (以降、USB と記述)の Full-Speed および Low-Speed に対応しています。また USB ベンダホストデバイスあるいは USB ベンダペリフェラルデバイスと通信可能です。

1.2 関連ドキュメント

1. Universal Serial Bus Revision 2.0 specification
2. Battery Charging Specification Revision 1.2
   [http://www.usb.org/developers/docs/]
3. Renesas USB MCU ユーザーズマニュアル ハードウェア編
   ルネサス エレクトロニクスホームページ　より入手できます。

- 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/usb/】

1.3 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。

API	:	Application Program Interface
APL	:	Application program
cstd	:	USB-BASIC-F/Wの Peripheral & Host共通関数プレフィックス
CS+	:	ルネサス統合開発環境(RL78対応)
CDP	:	Charging Downstream Port
DCP	:	Dedicated Charging Port
HBC	:	Host Battery Charging control
Data Transfer	:	Generic name of Bulk transfer and Interrupt transfer (When the host function is selected, the Control transfer is contained.)
e^2 studio	:	Eclipse embedded studio
HCD	:	Host control driver of USB-BASIC-F/W
HDCD	:	Host device class driver (device driver and USB class driver)
HEW	:	High-performance Embedded Workshopルネサス統合開発環境
HM	:	Hardware Manual
hstd	:	USB-BASIC-F/Wのhost関数プレフィックス
H/W	:	Renesas USB device
MGR	:	Sequencer of HCD to manage the state of the peripheral device
PBC	:	Peripheral Battery Charging control
PCD	:	Peripheral control driver of USB-BASIC-F/W
PDCD	:	Peripheral device class driver (device driver and USB class driver)
psmpl	:	Peripheral Smple (code)
PP	:	プリプロセス定義
pstd	:	USB-BASIC-F/Wのpripheral関数プレフィックス
RSK	:	Renesas Starter Kit
SDP		Standard Downstream Port
UPL		User Programming Layer (USB-BASIC-F/Wの上位層:HDCD, PDCD, APL or etc)
USB	:	Universal Serial Bus

USB-BASIC-F/W	:	USB Host and Peripheral Basic Mini Firmware (Peripheral & HostUSB基本FW (USB low level) for Renesas USB MCU)
スケジューラ		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	:	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
データ転送	:	Bulk転送、Interrupt転送の総称（ホスト動作時はControl転送も含めて）

1.4 本書の読み方

本書は章の順番通りに読み進める必要はありません。はじめにサンプルプログラムの内容を確認し、ユーザ個別のソリューションに必要な関数およびインタフェースの情報をお読みください。

4.3章にソース一覧を掲載しています。MCU 固有ソースは、"***devicename***\src*HwResource*"にあります。アプリケーションに必要なファイルを確認してください。

本書の5章および6章はホスト機能、7章および8章はペリフェラル機能に関する説明になります。ユーザ独自のソリューションを作成するためにはアプリケーションの変更が必要です。5章はペリフェラルベンダアプリケーションの動作を説明しています。7章はホストベンダアプリケーションの動作を説明しています。

すべてのコードモジュールはタスクに分割されます。タスク間でメッセージの受け渡しが行われていることを予めご理解ください。関数(タスク)の実行順序はスケジューラが決定します。このため重要なタスクに優先権を持たせることができます。また、タスクに登録されたコールバックメカニズムを使用することで、各タスクは並列処理（ノンブロッキング）で動作します。タスクのメカニズムは9.1章で説明しています。USB-BASIC-F/Wのタスクについては4.4章を参照してください。

2. クラスドライバの登録方法

ユーザが作成したクラスドライバ及びアプリケーションをUSB-BASIC-F/Wに搭載する方法について記載します。

2.1 ペリフェラルクラスドライバの登録方法

r_usb_vendor_papl.c の中の関数 *usb_psmpl_driver_registration()*を参照にしてクラスドライバを登録してください。

詳細に関しては、6章.を参照してください。

下記にユーザが作成するクラスドライバおよびアプリケーションの登録例を示します。

```
USB_STATIC void usb_psmpl_driver_registration(void)
{
 usb_pcdreg_tdriver;
 /* Driver registration */
 driver.pipetbl     = g_usb_psmpl_EpTbl1;            /* Pipe define table */
 driver.devicetbl   = g_usb_psmpl_DeviceDescriptor;
 driver.configtbl   = g_usb_psmpl_Configuration;
 driver.stringtbl   = g_usb_psmpl_StringPtr;
 driver.statediagram  = &usb_psmpl_device_state;    /* Change device state
*/
 driver.ctrltrans   = &usb_psmpl_control_transfer;  /* Control transfer */
 R_usb_pstd_DriverRegistration(&driver);
}
```

2.2 ホストクラスドライバの登録方法

r_usb_vendor_hapl.c の中の関数 *usb_hsmpl_driver_registration()*へクラスドライバを登録してください。

詳細に関しては、8章を参照してください。

下記にユーザが作成するクラスドライバおよびアプリケーションの登録例を示します。

```
USB_STATIC void usb_hsmpl_driver_registration(void)
{
 usb_hcdreg_tdriver;
 /* Driver registration */
 driver.ifclass       = USB_IFCLS_VEN;              /* Device class */
 driver.classcheck    = &usb_hsmpl_class_check;     /* Operation judgment */
 driver.statediagram  = &usb_hsmpl_device_state;    /* Change device state
*/
 R_usb_hstd_DriverRegistration(&driver);
}
```

3. USB-BASIC-FW

3.1 開発目的

USB-BASIC-F/Wは以下の目的で開発しています。 .

1. お客様における Renesas USB MCU を使用した USB 通信プログラムの開発を容易にする。
2. USB 通信の H/W 制御について、ソースコードの動作例を提供する。

3.2 特長

USB-BASIC-F/Wはデバイス内蔵の H/W 制御用サンプルファームウェアとして以下のような特徴を所持しています。

3.2.1 全般

1. 対象デバイスとの組み合わせで、USB2.0 規格の Full-Speed／Low-Speed に対応
2. 共通のソースコードで、R8C/USB、RL78/USB を制御。(MCU 差異は 3.5 を参照)
3. USB ホスト機能もしくは USB ペリフェラル機能として動作が可能
4. H/W 制御（接続／切断、サスペンド／レジューム／リモートウェイクアップ）を行う API 関数を提供
5. データ転送（コントロール転送、バルク転送、インタラプト転送）を行う API 関数を提供
6. UPL がエンドポイントのデータトグルを管理するため、排他的なパイプ使用により同一パイプで複数のデータ転送が可能
7. H/W 制御結果、データ転送結果、USB ステート遷移を UPL に通知するためのコールバック関数が登録可能
8. USB-BASIC-F/W の使用例を示すサンプルアプリケーション、ベンダクラスドライバを提供
   (1). コントロール転送（エニュメレーション処理）
   (2). バルク転送、および、インタラプト転送
   (3). クラスリクエストの記述方法（コントロール転送）

3.2.2 ホスト機能モード

1. Full-Speed デバイス/Low-Speed デバイスとのエニュメレーション
   (Low-Speed デバイスと接続可能な MCU は RL78/USB のみ)
2. コントロール転送（エニュメレーション処理）のサンプルプログラムを提供
3. コントロール転送、バルク転送、インタラプト転送で共通のデータ転送用 API 関数を提供
4. サスペンド／レジューム処理用 API 関数を提供
5. USB Battery Charging Specification Revision 1.2 で定義された Charging Downstream Port または Dedicated Charging Port を動作させるサンプルプログラムを提供（RL78/USB のみ）

3.2.3 ペリフェラル機能モード

1. Full-Speed デバイス/Low-Speed デバイスとして USB1.1 / 2.0 / 3.0 ホストとのエニュメレーション
   （Low-Speed は、RL78/USB がサポート)
2. *USBCommandVerifier.exe* による動作確認が可能
   (USBCV は、http://www.usb.org/developers/tools/ より入手できます)
3. コントロール転送（エニュメレーション処理）のサンプルプログラムを提供
4. コントロール転送時の FIFO バッファアクセス用 API 関数を提供
5. バルク転送、インタラプト転送で共通のデータ転送用 API 関数を提供
6. リモートウェイクアップ処理用 API 関数を提供
7. USB Battery Charging Specification Revision 1.2 で定義された Charging Port Detection を動作させるサンプルプログラムを提供（RL78/USB のみ）

3.2.4 ユーザ作成が必要な機能

以下の機能は、お客様のシステムに合わせて作成してください。

- 過電流検出処理とディスクリプタ解析処理（ホスト機能選択時）
- デバイスクラスドライバ（HID，MSC，CDC，LibUSB，etc）
- データ転送用のパイプ情報テーブル
- ディスクリプタテーブル（ペリフェラル機能選択時）

3.3 動作確認済環境

3.3.1 コンパイラ

動作確認を行ったコンパイラは以下の通りです。

a. CA78K0R コンパイラ　V.1.71
b. CC-RL コンパイラ V.1.01
c. IAR C/C++ Compiler for RL78 version 2.10.4
d. KPIT GNURL78-ELF v15.02
e. C/C++ Compiler Package for M16C Series and R8C Family V.6.00 Release 00

3.3.2 評価ボード

動作確認を行った評価ボードは以下の通りです。

a. Renesas Starter Kit for RL78/G1C (型名: R0K5010JGC001BR)
b. Renesas Starter Kit for RL78/L1C (型名: R0K50110PC010BR)
c. R8C/34K Group USB Host 評価ボード(型名: R0K5R8C34DK2HBR)
d. R8C/34K Group USB Peripheral 評価ボード(型名: R0K5R8C34DK2PBR)

3.4 スケジューラ機能とタスク

スケジューラ機能は各タスクや H/W の要求を各タスク優先度にしたがって管理します。USB-BASIC-F/W は要求の終了をコールバック関数によりタスクに通知する構造で設計しています。UPL 追加や変更をする場合でもスケジューラ機能を変更する必要はありません。この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 UPL の実装が可能です。スケジューラ機能の詳細は9.1 を参照ください。

3.5 MCU による USB 機能差異

MCU による USB 機能をTable 3-1に示します。

Table 3-1 RL78 と R8C の USB 機能一覧

機能	R8C/USB	RL78/USB
MCU 種別	R8C/34U, R8C/3MU, R8C/34K, R8C/3MK	RL78/G1C, RL78/L1C
ペリフェラル機能 動作可能な通信速度	1Port Full-Speed デバイスとして動作可能	1Port*1 Full-Speed デバイスとして動作可能 Low-Speed デバイスとして動作可能
ホスト機能 接続可能な通信速度	R8C/34U, R8C/3MU は無し R8C/34K, R8C/3MK は 1PortHost Full-Speed デバイスと接続可能	RL78/L1C は無し RL78/G1C は 2PortHost*2 Full-Speed デバイスと接続可能 Low-Speed デバイスと接続可能
Control 転送	PIPE0	PIPE0
Bulk 転送	PIPE4, PIPE5	PIPE4, PIPE5
Interrupt 転送	PIPE6, PIPE7	PIPE6, PIPE7
Isochronous 転送	不可	不可
ホスト時の HUB 接続	不可	不可
Battery Charging 対応	非対応	対応

注意)

* 1:ユーザは、USB-BASIC-F/W と UPL にて、Full-Speed ペリフェラルデバイスまたは Low-Speed ペリフェラルデバイスのどちらで動作するかどうかをカスタマイズすることができます。

詳細に関しては 5.6を参照してください。

* 2:ターゲットボードが RSKRL78 の場合、ホスト動作は *USB-PORT1* 側のみで動作します。USB-BASIC-F/Wでは、*USB-PORT1* 側をホストで動作させるには 2PORTHOST としてビルド（実行ファイル作成）する必要があります。詳細に関しては Chapter 7.5を参照してください。

3.6 ホスト/ペリフェラルサンプルデモプログラム

この FW には、ベンダクラスのホスト/ペリフェラル用のサンプルデモプログラムが用意されています。動作概要は以下のとおりです。

1. ホストは、EP1(OUT)と EP3(OUT)を使用して 0x00 から 0xFF のデータを USB ペリフェラルに送信します。
2. ペリフェラルは、ホストから送信される 0x00 から 0xFF のデータを EP1 と EP3 からリードし続けます。
3. ペリフェラルは、EP2(IN)と EP4(IN)を使用して 0x00 から 0xFF のデータを USB ホストに送信します。
4. ホストは、ペリフェラルから送信される 0x00 から 0xFF のデータを EP2 と EP4 からリードし続けます。

3.7 注意

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、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。

4. ソフトウェア構成

4.1 モジュール構成

USB-BASIC-F/W を構成するソフトウェアは"タスク"構造で作成されています。Figure 4-1にUSB-BASIC-F/Wのタスク構成、Table 4-1にソフトウェア機能概要を示します。 これらのタスクはメッセージシステムを使用したスケジューラを介して動作します。

USB-BASIC-F/Wは、PCD(ペリフェラル制御機能)、HCD(ホスト制御機能)、および MGR(ホスト機能で USB ステート管理とシーケンス処理を行う)で構成されています。USB クラスドライバ（HDCD/PDCD）、ホストデバイスドライバ（HDD）および、アプリケーション（APL）はUSB-BASIC-F/Wの一部ではありません。

PCD は、UPL の要求でデータ転送と H/W 制御を行います。H/W 制御終了、データ転送結果、および割り込みハンドラからの要求（状態変更など）を各タスクに通知します。

HCD は、MGR タスクからの要求でデータ転送と H/W 制御を行います。また、UPL の要求でデータ転送を行います。H/W 制御終了、および割り込みハンドラからの要求（状態変更など）を MGR タスクに通知します。また、データ転送結果を MGR タスクと UPL に通知します。

MGR は、接続されたデバイスの USB ステート管理とエニュメレーションなどのシーケンス処理を行います。また、API 関数による UPL の要求に従い接続デバイスの USB ステート遷移を行います。MGR は USB ステート遷移に必要なシーケンス処理（H/W 制御やデータ転送）を HCD に要求します。USB ステート遷移の結果をコールバック関数で UPL に通知します。

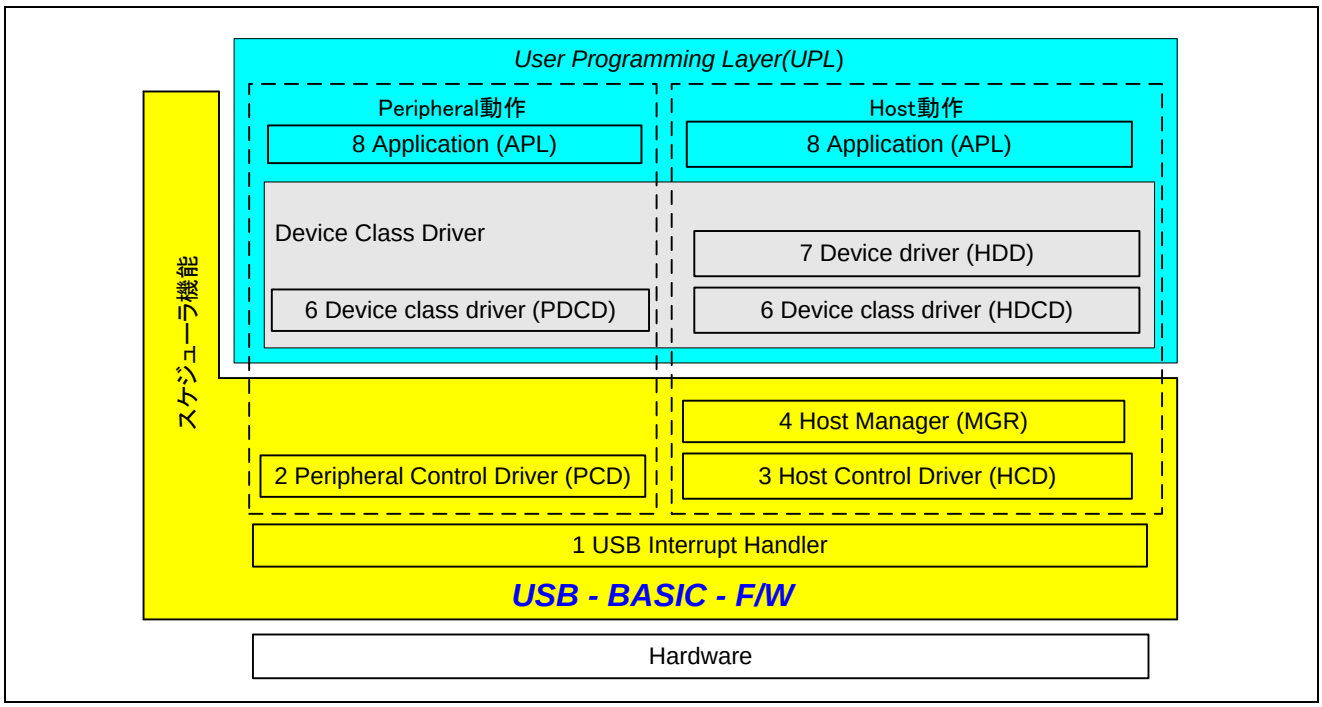

Figure 4-1 USB-BASIC-F/Wのタスク構成

Table 4-1 ソフトウェア機能概要

No	モジュール名	概要	備考
1	USB interrupt handler	USB 割り込みハンドラ (USB パケット送/受信終了と特殊信号検)	
2	Peripheral control driver (PCD)	H/W 制御 ペリフェラルトランザクション管理	
3	Host control driver (HCD)	H/W 制御 ホストトランザクション管理	
4	Host manager (MGR)	USB ステート管理 ホストエニュメレーション処理	
5	Device class driver (PDCD/HDCD)	システムにあわせてご用意ください (各種クラスドライバの例がダウンロード可能です)	
6	Device driver (HDD)	システムにあわせてご用意ください (各種クラスドライバの例がダウンロード可能です)	
7	Application(APL)	システムにあわせてご用意ください (各種サンプルアプリケーションの例がダウンロード可能です)	

4.2 アプリケーションプログラム機能概要

アプリケーションの主な機能は以下のとおりです。

1. 接続された USB デバイスとエニュメレーションを行う。
2. 接続された USB デバイスとバルク転送およびインタラプト転送にてデータ受信を行う。
3. 接続された USB デバイスとバルク転送およびインタラプト転送にてデータ送信を行う
4. ユーザが RSK 上の SW3→SW1 を押すと、接続された USB デバイスのデバイスステートが変更される。

接続されたペリフェラルデバイスが Low-Speed デバイスの場合、インタラプト転送のみデータ転送が可能です。

スイッチ入力動作をTable 4-2 と Table 4-3に示します。

Table 4-2 ホスト機能のユーザスイッチ入力動作

スイッチ機能	説明	スイッチ番号
SUSPEND	接続されたペリフェラルデバイスをサスペンドします。	SW1
RESUME	接続されたペリフェラルデバイスをレジュームします。	SW2
PORTCONTROL	VBUS 出力禁止<--->出力許可を行います	SW3

Table 4-3 ペリフェラル機能のユーザスイッチ入力動作

スイッチ機能	説明	スイッチ番号
REMOTEWAKEUP	ホストにリモートウェイクアップします。	SW1
PORT OFF	D+ or D- 信号の Pull-up 解除	SW2
PORT ON	D+ or D- 信号の Pull-up 許可	SW3

4.3 ファイル/フォルダ構成

4.3.1 フォルダ構成

以下にUSB-BASIC-F/Wが提供するファイルのフォルダ構成を示します。提供するファイルには、データ転送用のベンダクラスドライバ、アプリケーション及び H/W リソースのサンプルコードが含まれます。

統合化環境のプロジェクトファイルは各開発環境のフォルダ毎に分かれています。また制御用 MCU や評価ボードに依存するソースコードは HwResource フォルダ下に格納しています。

```
workspace
    + [ RL78 / R8C ]
        + [ CCRL / CS+ / IAR / e2 studio / HEW ]
            + [ RL78G1C / RL78L1C / R8C3MK / R8C3MU / R8C34K / R8C34U ]
                + ――― HOST                          ホストビルド結果 [Note]
                + ――― PERI                          ペリフェラルビルド結果
                + src
                  ＋―――USBSTDFW[USB-BASIC-F/Wが使用する共通 USB コード]
                  │          ＋――― inc              USB ドライバ共通ヘッダファイル
                  │          ＋――― src              USB ドライバ
                  ＋―――SmplMain[サンプルアプリケーション]
                  │          ＋――― APL              サンプルアプリケーション
                  ＋―――VENDOR  [クラスドライバ Vendor Class driver]
                  │          ＋――― inc              ベンダクラスドライバ共通ヘッダファイル
                  │          ＋――― src              ベンダクラスドライバ
                  ＋―――HwResource [MCU 初期化等のハードウェアアクセス層]
                             ＋――― inc              H/W リソースヘッダファイル
                             ＋――― src              H/W リソース
```

[Note]

a. MCU 名が RL78L1C には、HOST フォルダは用意されていません。

b. CS+フォルダ下には、CA78K0R コンパイラ用のプロジェクトが格納されています。

c. e2 studio フォルダ下には、KPIT GNU コンパイラ用のプロジェクトが格納されています。

d. CS+上で CC-RL コンパイラをご使用になる場合は、「12 e2 studio 用プロジェクトを CS+で使用する場合」を参照してください。

4.3.2 ファイル一覧

以下にUSB-BASIC-F/W が提供するファイル一覧を示します。

Table 4-4 ファイル一覧

Folder	ファイル名	説明	備考
USBSTDFW\src	r_usb_cstdapi.c	USB ライブラリ API 関数	
USBSTDFW\src	r_usb_cstdfunction.c	USB ライブラリ関数	
USBSTDFW\src	r_usb_h1port.c	1 ポートホスト用関数	
USBSTDFW\src	r_usb_h2port.c	2 ポートホスト用関数	
USBSTDFW\src	r_usb_hbc.c	USB Host Battery Charging control 関数	
USBSTDFW\src	r_usb_hdriver.c	USB Host Control Driver	
USBSTDFW\src	r_usb_hdriverapi.c	HCD API 関数	
USBSTDFW\src	r_usb_hp0function.c	Port0 制御関数	
USBSTDFW\src	r_usb_hp1function.c	Port1 制御関数	
USBSTDFW\src	r_usb_pbc.c	USB Peripheral Battery Charging control 関数	
USBSTDFW\src	r_usb_pdriver.c	USB Peripheral Control Driver	
USBSTDFW\src	r_usb_pdriverapi.c	PCD API 関数	
USBSTDFW\src	r_usb_hport.h	USB ホスト関数プロトタイプ	
USBSTDFW\src	r_usb_regaccess.h	USB レジスタアクセスマクロ定義	
USBSTDFW\inc	r_usb_api.h	USBAPI 関数プロトタイプ	
USBSTDFW\inc	r_usb_ckernelid.h	Macro definition of scheduler functions スケジューラ機能マクロ定義	
USBSTDFW\inc	r_usb_ctypedef.h	USB-BASIC-F/W変数型定義	
USBSTDFW\inc	r_usb_usrconfig.h	Macro definition of user configuration ユーザ設定（H/W 動作指定）定義	
SmplMain	main.c	メインプログラム	
SmplMain\APL	r_usb_vendor_descriptor.c	ペリフェラルサンプルディスクリプタとエンドポイント情報	
SmplMain\APL	r_usb_vendor_hapl.c	ホストサンプルアプリケーションプログラム	
SmplMain\APL	r_usb_vendor_papl.c	ペリフェラルサンプルアプリケーションプログラム	
SmplMain\APL	r_usb_vendor_apl.h	アプリケーション用共通定義ファイル	
VENDOR\src	r_usb_vendor_hapi.c	ホストサンプルベンダクラスドライバ API 関数	
VENDOR\src	r_usb_vendor_hdriver.c	ホストサンプルベンダクラスドライバ	
VENDOR\src	r_usb_vendor_papi.c	ペリフェラルサンプルベンダクラスドライバ API 関数	
VENDOR\src	r_usb_vendor_pdriver.c	ペリフェラルサンプルベンダクラスドライバ	
VENDOR\inc	r_usb_vendor_api.h	ベンダクラスドライバ用定義ファイル	
R8C3xx\src\HwResource\src	ncrt0.a30 adcdriver.c keydriver.c lcddriver.c leddriver.c r8cusbmcu.c iodefine_r8cusb_usb.h nc_define.inc sect30.inc sfr_r83xx.h	R8C-評価ボード用 HW リセット初期化処理 R8C-評価ボード用 AD 入力ドライバ R8C-評価ボード用キーボードドライバ R8C-評価ボード用 LCD ドライバ R8C-評価ボード用 LED ドライバ MCU H/W 処理（R8C-評価ボード） R8C-USB レジスタ定義ファイル R8C-評価ボード用マクロシンボル定義定義ファイル R8C-評価ボード用セクション定義ファイル R8C/3xx 用 SFR 定義ファイル	
R8C3xx\src\HwResource\inc	hw_resource.h r_usb_usbip.h	R8C-評価ボード用周辺ドライバ定義ファイル USB レジスタ定義	

RL78xxx\src\HwResource\src	adcdriver.c keydriver.c lcddriver.c leddriver.c rl78usbmcu.c	RSK ボード用 AD 入力ドライバ RSK ボード用キーボードドライバ RSK ボード用 LCD ドライバ RSK ボード用 LED ドライバ MCU H/W 処理 RSK ボード用 HW リソース定義ファイル	
RL78xxx\src\HwResource\inc	hw_resource.h r_usb_usbip.h	RSK ボード用周辺ドライバ定義ファイル USB レジスタ定義	

4.4 システムリソース

4.4.1 システムリソース 定義

タスク ID とスケジューラに登録するタスク優先順位をTable 4-5と Table 4-6に示します。これらは *r_usb_ckernelid.h* ヘッダファイルに定義します。

Table 4-5 List of Scheduler Registration IDs when host operates

スケジューラ登録タスク	概要
Task ID: USB_HVENDOR_TSK	HDCD (R_usb_hvndr_Task) Task priority: 2
Task ID: USB_HSMP_TSK	APL (usb_hsmpl_apl_task) Task priority: 3
Task ID: USB_HCD_TSK	HCD (R_usb_hstd_HcdTask) Task priority: 0
Task ID: USB_MGR_TSK	MGR (R_usb_hstd_MgrTask) Task priority: 1
メールボックス ID / Default receive task	**メッセージ概要**
USB_HVENDOR_MBX / USB_HVENDOR_TSK	APL -> HDCD メールボックス ID
USB_HSMP_MBX / USB_HSMP_TSK	HDCD -> APL メールボックス ID
USB_HCD_MBX / USB_HCD_TSK	HCD タスク メールボックス ID
USB_MGR_MBX / USB_MGR_TSK	MGR タスク メールボックス ID

Table 4-6 List of Scheduler Registration IDs when peripheral operates

スケジューラ登録タスク	概要
Task ID: USB_PVENDOR_TSK	PDCD (R_usb_pvndr_Task) Task priority: 3
Task ID: USB_PSMP_TSK	APL (usb_psmpl_apl_task) Task priority: 4
Task ID: USB_PHCD_TSK	PCD (R_usb_pstd_PcdTask) Task priority: 0
メールボックス ID / Default receive task	**メッセージ概要**
USB_PVENDOR_MBX / USB_PVENDOR_TSK	APL -> PDCD メールボックス ID
USB_PSMP_MBX / USB_PSMP_TSK	PDCD -> APL メールボックス ID
USB_PCD_MBX / USB_PCD_TSK	PCD タスク メールボックス ID

4.5 ご注意

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、お客様にてカスタマイズして頂く必要があります。

5. ペリフェラルサンプルプログラム(UPL)

本章では MCU が RL78 の場合について説明します。Low-Speed デバイスはバルク転送の通信を行いません。Low-Speed をサポートするデバイスをご使用の場合、バルク転送に関する記述はスキップしてください。Low-Speed をサポートしていないデバイスをご使用の場合、Low-Speed に関する記述はスキップしてください。USB ホストに接続された時、サンプルアプリケーションはデータ通信を行ないます。

5.1 動作環境

動作環境をFigure 5-1 と Figure 5-2 に示します。

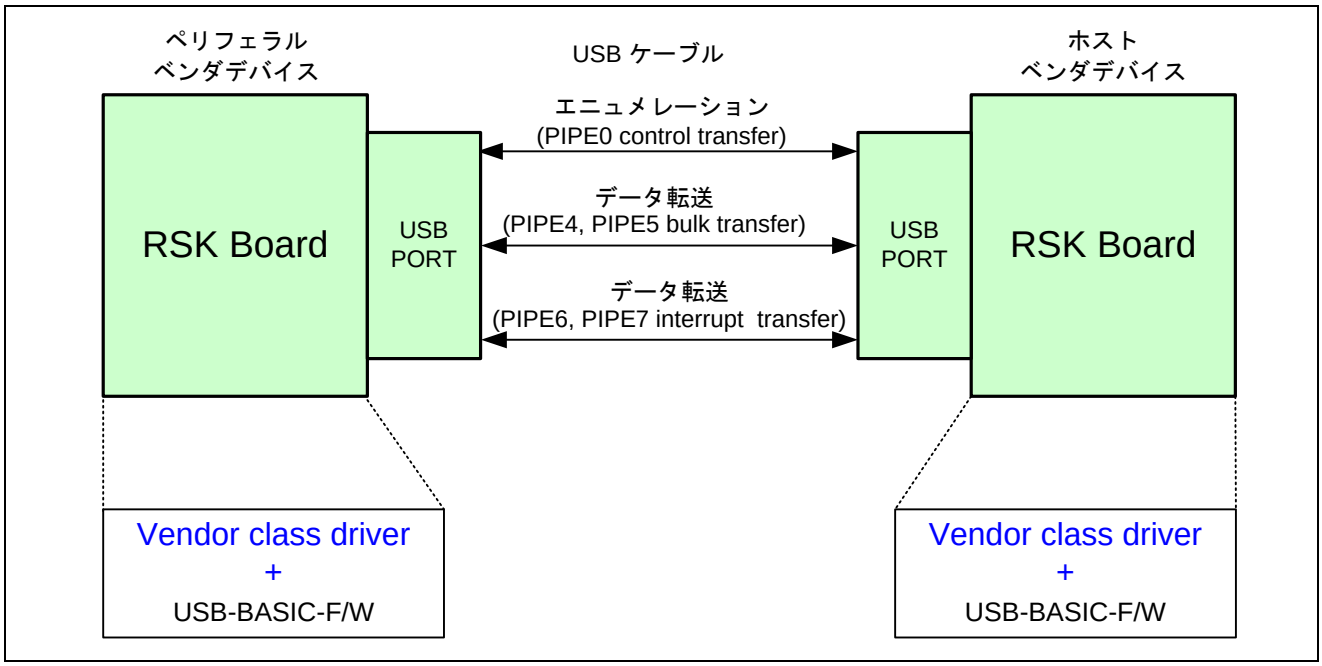

Figure 5-1 Full-Speed 動作環境例

1. コンパイラ

動作確認を行ったコンパイラは以下の通りです。

f. CA78K0R コンパイラ V.1.71

g. IAR C/C++ Compiler for RL78 version 2.10.4

h. KPIT GNURL78-ELF v15.02

i. C/C++ Compiler Package for M16C Series and R8C Family V.6.00 Release 00

2. 評価ボード

動作確認を行った評価ボードは以下の通りです。

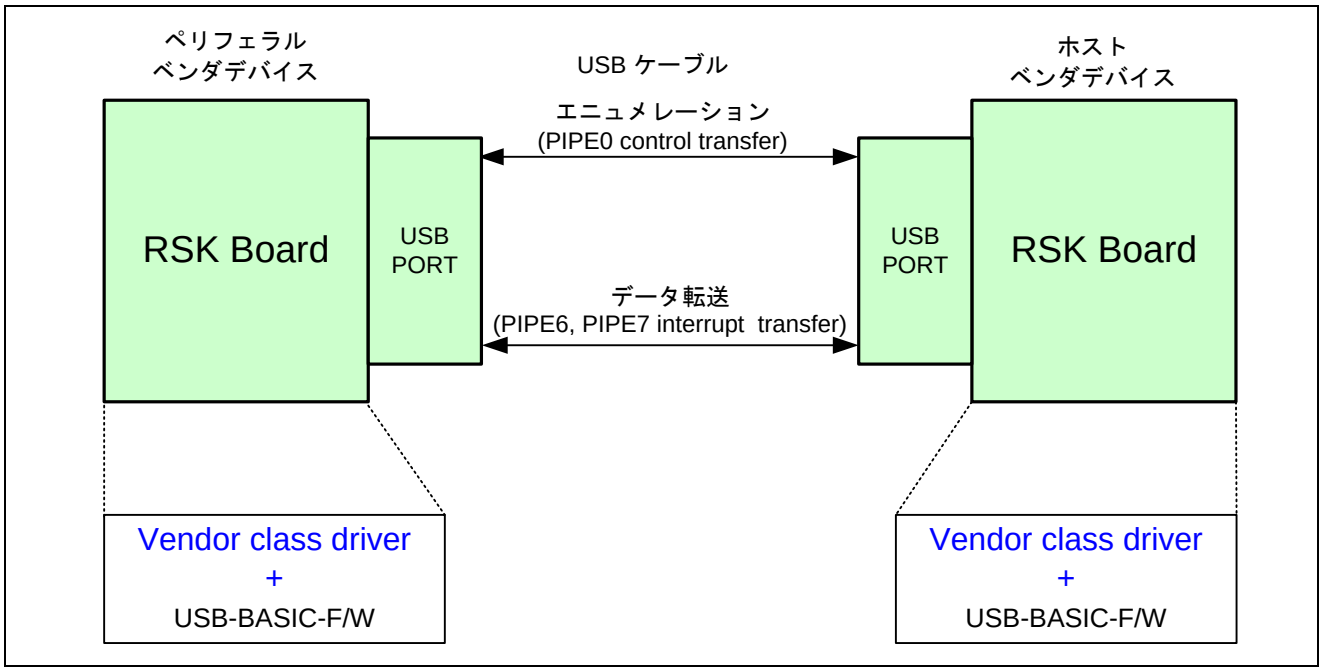

Figure 5-2 Low-Speed 動作環境例

5.2 サンプルプログラムの説明

USB-BASIC-F/Wのペリフェラルサンプルプログラムは、*r_usb_usrconfig.h* ファイルのユーザ仕様にて Full-Speed デバイスまたは Low-Speed デバイスを選択することができます。サンプルプログラムは、データ転送用のベンダクラスドライバとサンプルアプリケーションを搭載しています。バルク転送は、PIPE4,5 を使用します。また、インタラプト転送は PIPE6,7 を使用します。

お客様固有のクラスドライバ、アプリケーションを作成する場合は、*r_usb_vendor_papl.c* ファイル、*r_usb_vendor_descriptor.c* ファイル、*r_usb_vendor_pdriver.c* ファイルを参考に作成してください。USB ペリフェラルデバイスとして USB ホストと通信するためには以下の設定が必要になります。

1. Full-Speed デバイスまたは LowSpeed デバイス の動作選択
2. スケジューラの設定（タスク数やテーブルサイズ、タスク ID など、メールボックス ID など）
3. タスクの呼び出し
4. 実装するデバイスクラスドライバに対応したディスクリプタテーブルの作成
5. 実装するデバイスクラスドライバに対応したパイプ情報テーブルの作成
6. USB リクエストの応答

5.2.1 機能

1. サンプルアプリケーション

   USB ステート遷移をベンダクラスドライバに通知します。また、ベンダクラスドライバを制御します。このとき USB_STS_CONFIGURED を検出するとグローバル変数を初期化してベンダクラスドライバにデータ転送開始を要求します。データ転送は PIPE4,5 を使用したバルク転送と PIPE6,7 を使用したインタラプト転送です。ベンダクラスドライバからデータ転送終了が通知されると、通知をうけたパイプでデータ転送を再開します。USB_STS_SUSPEND が通知されると、APL は STOP/WAIT 命令を実行します。また定常処理でキー入力の通知を受けます。リモートウェイクアップ（サスペンド時）、ポート許可、ポート禁止のサンプルを実装しています。

2. ベンダクラスドライバ

   APL から通知される USB ステートに従ってグローバル変数を初期化します。またアプリケーションからデータ転送が要求されるとUSB-BASIC-F/Wにデータ転送を要求します。USB-BASIC-F/Wからデータ転送終了が通知されるとアプリケーションにデータ転送終了を通知します。ベンダクラスリクエストには対応していません。

3. エニュメレーション

   USB-BASIC-F/W は USB ホストと接続されると USB ホストはエニュメレーションを行います。USB ホストがベンダクラスドライバに対応していればエニュメレーションが正常終了し、コールバック関数でアプリケーションに USB_STS_CONFIGURED が通知されます。

4. データ転送

   エニュメレーションが正常終了した場合はデータ転送が可能です。アプリケーションは USB ステート遷移のコールバックでデータ転送を開始します。USB-BASIC-F/W を USB ホストとして動作させたデバイスと通信できます。

5. ベンダクラスリクエスト

   ベンダクラスリクエストには STALL 応答します。

6. USB ステート遷移

   USB ステート遷移の通知によって以下のように動作します

USB_STS_DETACH:	データ転送停止
USB_STS_DEFAULT:	データ転送サイズ初期化、コンフィグレーション番号初期化
USB_STS_ADDRESS:	コンフィグレーション番号初期化
USB_STS_CONFIGURED:	データトグルバッファ初期化、データ転送開始
USB_STS_SUSPEND:	データ転送割り込み、STOP/WAIT 命令実行
USB_STS_RESUME:	データ転送再開

サスペンド状態からレジューム信号による復帰が可能です。またアプリケーションからリモートウェイクアップをUSB-BASIC-F/Wに要求することも可能です。

7. USB デバイスフレームワーク

USB Implementers Forum（USB-IF）より配布されている"USBCommandVerifier.exe"（USBCV）を使用した、デバイスフレームワークテストによる動作確認を行うことができます。（対応するテスト項目は「Chapter9」のみとなります。

5.2.2 ペリフェラルサンプルプログラムの動作

1. 初期設定

- HEW/e^2 studio の場合

デバイスを H/W リセットすると、ncrt0.a30/resetprg.c 内の関数*_PowerON_Reset_PC* 関数が呼び出されます。
リセット関数は MCU の初期化を行い、ハードウェア初期化関数 *usb_cpu_mcu_initialize()*を呼び出します。ハードウェア初期化関数から戻ると、メモリ領域の初期化を行い、*main.c* ファイル内の *main()* 関数が呼び出されます。スタートアップ処理の詳細は HM、および統合開発環境のマニュアルを参照してください。

- CS+の場合

デバイスを H/W リセットすると、CS+で作成されたスタートアップファイルの *_@cstart* 関数が呼び出されます。スタートアップ関数は MCU の初期化を行い、ユーザ定義ハードウェア初期化関数 *hdwinit()* 関数を呼び出します。ハードウェア初期化関数で MCU 動作クロックの設定を行います。ハードウェア初期化関数から戻ると *saddr* 領域をはじめとするメモリ領域を初期化し、*main.c* ファイル内の *main()* 関数が呼び出されます。スタートアップ処理の詳細は HM、および統合開発環境のマニュアルを参照してください。

2. main 関数処理
   main() 関数では *usb_psmpl_main_init()* 関数でシステムの初期化（ターゲット MCU およびボードの初期化、USB モジュールの初期設定、USB-BASIC-F/Wの起動、UPL ドライバの登録、USB モジュールの動作許可）を行いメインループで要求の発生を待つ定常状態となります。メインループの動作は以下のようになります。

   ① スケジューラで要求判別する。

   ② 処理要求があった場合、タスクに制御を渡す（タスクが実行可能される）。

   ③ 定常的な処理を行う。

   ④ ①に戻る。

3. サンプルアプリケーションタスク (*usb_psmpl_apl_task()*)
   サンプルアプリケーションは、エニュメレーションが正常終了するとグローバル変数を初期化し、ベンダクラスドライバに対して API 関数 *R_usb_pvndr_TransferStart()* でデータ転送開始を要求します。また、ベンダクラスドライバから転送終了コールバックを受けると、API 関数 *R_usb_pvndr_TransferStart()*でデータ転送を繰り返します。USB_STS_SUSPEND が通知される場合、APL は *usb_cpu_stop_mode()*関数によって STOP/WAIT 命令を実行します。
4. ベンダクラスドライバ (*R_usb_psmpl_VendorTask()*)
   ベンダクラスドライバ（PDCD）は、サンプルアプリケーションからデータ転送要求が通知されると API 関数 *R_usb_pstd_TransferStart()* でUSB-BASIC-F/Wにデータ転送を要求します。また、USB-BASIC-F/Wから転送終了コールバックを受けると、コールバック関数でアプリケーションにデータ転送終了を通知します。
   ベンダクラスドライバは、サンプルアプリケーションから USB ステート遷移が通知されると USB ステートにしたがって以下のグローバル変数を初期化します。

   USB_STS_CONFIGURED:構成番号の記憶、グローバル変数の DATA-PID テーブルを初期化
   USB_STS_DETACH, USB_STS_ ADDRESS, USB_STS_ DEFAULT: 構成番号の"0 "クリア
   USB_STS_SUSPEND, USB_STS_RESUME: 処理無し

USB-BASIC-F/Wの概略フローをFigure 5-3に示します。

USB-BASIC-F/Wは USB データ送受信用の制御機能を実行するタスクを含んでいます。H/W の割り込みが発生すると USB-BASIC-F/W へメッセージとして通知されます。USB-BASIC-F/W は USB 割り込みハンドラからメッセージを受け取ると、割り込み要因を判別して適切な処理を実行します。

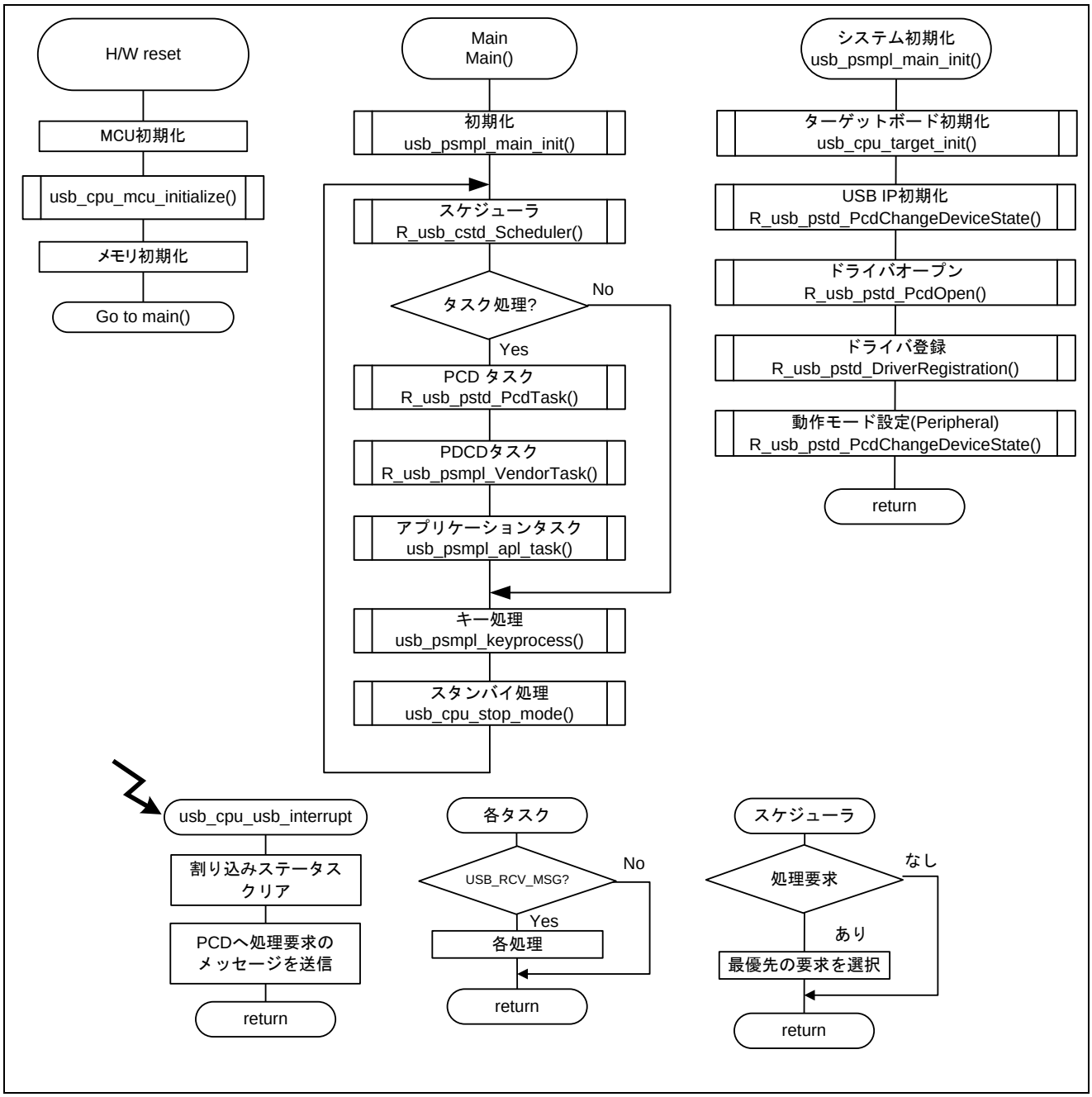

Figure 5-3 概略フロー

5.2.3 スケジューラの設定

タスク ID の最大値、タスク優先テーブルに格納するメッセージの最大値を *r_usb_cstd_kernelid.h* ファイルで設定します。

```
/* Please set with user system */
#define     USB_IDMAX           ((uint8_t)5)     /* Maximum Task ID +1 */
#define     USB_TABLEMAX        ((uint8_t)5)     /* Maximum priority table */
#define     USB_BLKMAX          ((uint8_t)5)     /* Maximum block */
```

5.2.4 タスク ID、メールボックス ID の設定

使用するタスク ID とメールボックス ID を *r_usb_cstd_kernelid.h* ファイルで設定します。
なお、タスク優先順位はタスク ID と同じです。（タスク ID 番号の小さい方が、優先度が高い）

```
#define     USB_PCD_TSK         USB_TID_0        /* Peripheral Control Driver Task */
#define     USB_PCD_MBX         USB_PCD_TSK      /* Mailbox ID */
#define     USB_PVEN_TSK        USB_TID_3        /* Vendor Class Driver ID */
#define     USB_PVEN_MBX        USB_PVEN_TSK     /* Mailbox ID */
#define     USB_PSMP_TSK        USB_TID_4        /* Peripheral Sample Application Task */
#define     USB_PSMP_MBX        USB_PSMP_TSK     /* Mailbox ID */
```

5.2.5 タスクの呼び出し

使用するタスクをメインループ(*main()* 関数)でコールしてください。

```
void main(void)
{
 /* Initialized USBIP */
 usb_psmpl_main_init();

 /* Sample main loop  */
    while( 1 )
    {
        if( R_usb_cstd_Scheduler() == USB_FLGSET )
        {
            R_usb_pstd_PcdTask();       /* PCD Task */
            R_usb_psmpl_VendorTask();
            usb_psmpl_apl_task();
        }
        keydata = usb_smpl_KeyRead();
        if (keydata != 0x00)
        {
            usb_psmpl_keyprocess(keydata);
        }
        if ( g_usb_suspend_flag == USB_YES )
        {
            usb_cpu_stop_mode();
        }
    }
```

5.2.6 UPL の起動

USB-BASIC-F/W はホストと構成が確立した場合（SET_CONFIGURATION リクエスト受信）に、UPL に対してコールバック関数(**g_usb_PcdDriver.statediagram*)でデバイス接続を通知します。UPL は第 2 引数の USB ステートを解析してシステムに適合した処理を行ってください。サンプルアプリケーションはベンダクラスドライバに USB ステート遷移を通知し、データ領域を初期化してデータ転送を開始します。USB ステート遷移が通知され初期化する場合、ベンダクラスドライバは構成番号を記憶します

5.2.7 USB リクエスト応答

USB-BASIC-F/Wが提供する API 関数を使用したコントロール転送のプログラム例を示します。例では、クラスリクエストを受信した場合のコントロール転送について示します。

```
void usb_psmp_ControlTransfer(usb_request_t* request, uint16_t ctsq)
{
 g_usb_psmp_Request= request;
 if ((g_usb_psmp_Request.wRequest & USB_BMREQUESTTYPETYPE) == USB_CLASS)
 {
    switch( ctsq )
    {
       case USB_CS_IDST: usb_psmp_control_trans0(request);break;
       case USB_CS_RDDS: usb_psmp_control_trans1(request);break;
       case USB_CS_WRDS: usb_psmp_control_trans2(request);break;
       case USB_CS_WRND:
          usb_psmp_control_trans3(request);
          R_USB_pstd_ControlEnd((uint16_t)USB_CTRL_END);  break;
       case USB_CS_RDSS:
          usb_psmp_control_trans4(request);
          R_USB_pstd_ControlEnd((uint16_t)USB_CTRL_END);  break;
       case USB_CS_WRSS:
          usb_psmp_control_trans5(request);
          R_USB_pstd_ControlEnd((uint16_t)USB_CTRL_END);  break;
       case USB_CS_SQER:
          R_USB_pstd_ControlEnd((uint16_t)USB_DATA_ERR);  break;
       default:
          R_USB_pstd_ControlEnd((uint16_t)USB_DATA_ERR);  break;
    }
 }
 else
 {
    R_USB_pstd_SetStallPipe0();
 }
}
```

1. データステージ処理

サポートしているリクエストの場合は、*R_usb_pstd_ControlRead()*/*R_usb_pstd_ControlWrite()* の API 関数を使用して USB ホストに対し、データ転送を行なってください。対応していないリクエストの場合は *R_usb_pstd_SetStallPipe0()* の API 関数を呼び出し、USB ホストに対し STALL 応答をしてください。

2. ステータスステージ処理

ステータスステージで USB_CTRL_END を引数に指定して *R_usb_pstd_ControlEnd()* の API 関数を呼び出してください。データステージで *R_usb_pstd_SetStallPipe0()*関数が実行されている場合は引数が USB_CTRL_END でも USB ホストに STALL 応答します。

3. 特記事項

USB-BASIC-F/W は API 関数 *R_usb_pstd_ControlRead()* / *R_usbh_pstd_ControlWrite()*. で指定されたデータサイズまでユーザバッファをアクセスします。 この為、ユーザバッファはコントロール転送のデータステージで送受信するデータサイズ以上の容量を確保してください。

5.2.8 アプリケーションの概要

USB-BASIC-F/Wは、コンフィガード後に以下の手順でデータ転送を開始します。コールバック関数 *usb_psmpl_device_state()* で、USB ステートを識別しベンダクラスドライバにデータ転送を要求します。

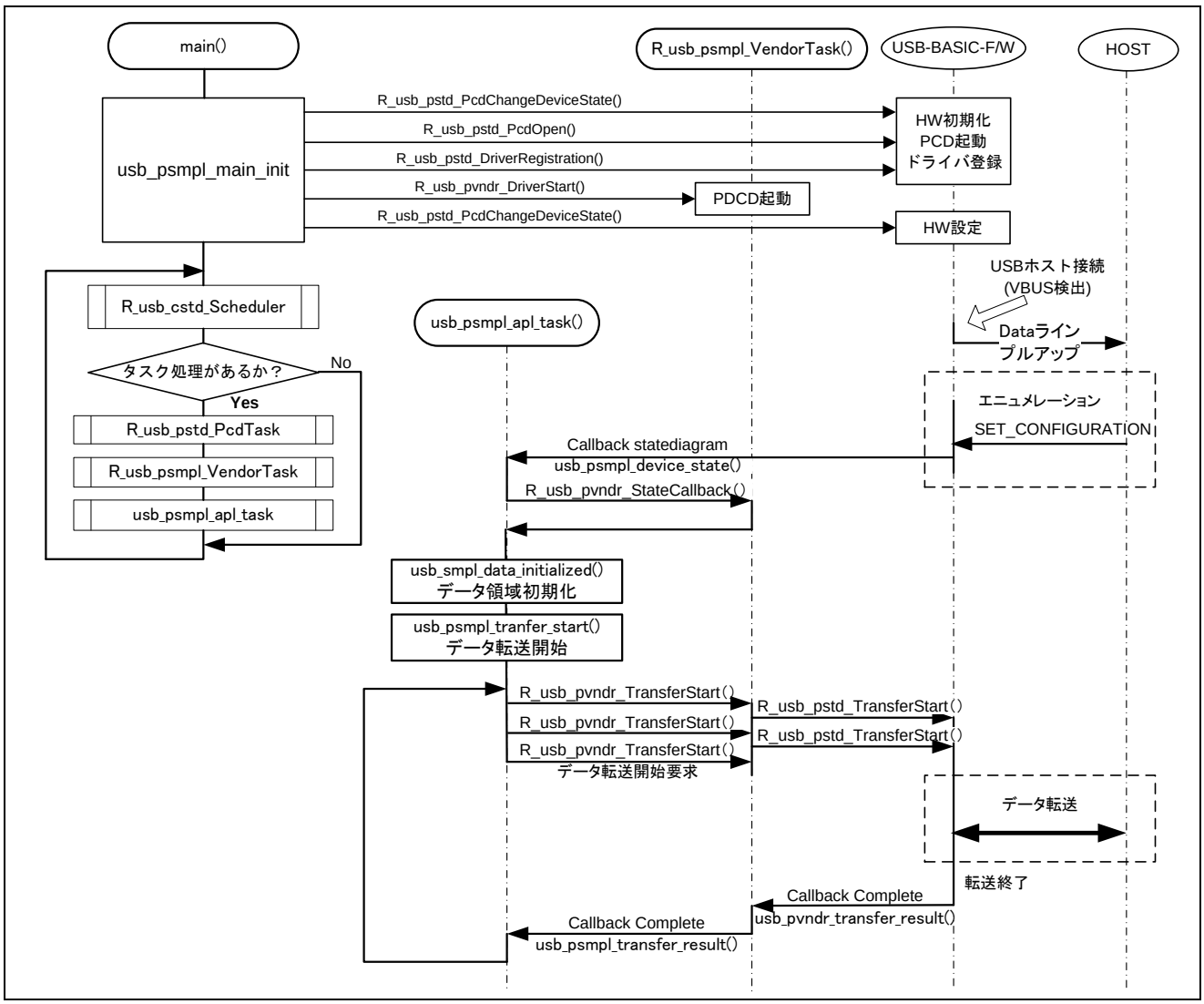

Figure 5-4 アプリケーション動作概要

5.3 データ転送

データ転送はお客様固有の機能仕様であり、転送方法、通信開始／終了タイミング、およびバッファ構成などは、システムにあわせて変更していただく必要があります.。

5.3.1 基本仕様

Table 6-7の *usb_utr_t* 構造体で指定されたユーザバッファでデータ転送が可能です。 USB-BASIC-F/W は　データ転送が終了すると *PID = NAK* に設定し、コールバックで転送終了を通知します。USB-BASIC-F/Wは、データ転送要求に対してパイプステータス（データトグル）を転送要求時に指定されたパイプステータス (*utr_table.pipectr*) に更新します。また、転送終了のコールバックでパイプステータスを通知します。このため、UPL がパイプステータスを記憶することにより、1 つのパイプで同時にデータ転送が行われない（排他的に制御が可能な）複数のエンドポイントのデータ転送が可能です。ただしパイプステータスは USB リセット、STALL 解除、SET_CONFIGURATION リクエスト、SET_INTERFACE リクエストなどの要因で"DATA0"に初期化する必要があります。なお、Bulk パイプのマックスパケットサイズは 64byte 固定です。

5.3.2 データ転送要求

*R_usb_pstd_TransferStart()*を使用してデータ転送を開始してください。

5.3.3 転送結果の通知

usb_utr_t 構造体で指定されたコールバック関数で、データ転送の終了を UPL に通知します。通知される内容はTable 6-7 を参照してください。

5.3.4 データ送信時の注意事項

1. 同一パイプの連続転送を行うことはできません。
2. 転送完了コールバックがコールされるまで次の転送を行うことはできません。

5.3.5 データ受信時の注意事項

(1) 受信パイプはトランザクションカウンタを使用します。

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*usb_utr_t* の*tranlen* に格納して転送を終了します。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を終了します。ユーザバッファ領域が転送するサイズの容量分を確保できない場合は、*usb_cstd_forced_termination()* 関数で受信パケットをクリアしてしまうことがあります。

(2) 受信コールバック

受信したデータがマックスパケットサイズの n 倍、かつ受信予定サイズに満たない場合は、データが転送途中であるか転送終了したかの判断ができないためコールバックが発生しません。

ショートパケット受信、または指定されたサイズのデータ受信をした場合に、転送終了と判断しコールバックします。

例:受信データサイズが 128byte でマックスパケットサイズが 64byte の場合

1～63byte 受信	受信コールバックが発生する。
64byte 受信	受信コールバックが発生しない。
65～128byte 受信	受信コールバックが発生する。

5.3.6 データ転送概要

usb_utr_t 構造体に必要な情報を設定し、*R_usb_pstd_TransferStart()* を呼び出してください。usb_utr_t 構造体の詳細については、Table 6-3を参照してください。

```
void usb_pvndr_transfer_start( uint16_t pipe )
{
 g_usb_PsmplTrnMsg[pipe].pipenum     = pipe;
 g_usb_PsmplTrnMsg[pipe].tranadr     = g_usb_PsmplTrnPtr[pipe];
 g_usb_PsmplTrnMsg[pipe].tranlen     = g_usb_PsmplTrnSize[pipe];
 g_usb_PsmplTrnMsg[pipe].pipectr     = g_usb_PsmplPipeCtr[pipe];
 g_usb_PsmplTrnMsg[pipe].setup       = USB_NULL;
 g_usb_PsmplTrnMsg[pipe].complete    = (usb_cb_t)&usb_pvndr_transfer_result;
 R_usb_pstd_TransferStart((usb_utr_t *)&g_usb_PsmplTrnMsg[pipe]);
}
```

コールバック関数（UPL タスクに転送終了をメッセージで通知）例を以下に示します。

```
void usb_pvndr_transfer_result(usb_utr_t *mess)
{
 usb_er_t    err;

 mess->msginfo = USB_SMPL_TRANSFER_END;

 err = R_USB_SND_MSG(USB_PVEN_MBX, (usb_msg_t*)mess);
 if( err != USB_E_OK )
 {
    while(1);
 }
}
```

5.4 パイプ情報

クラスドライバに適合したパイプ設定を、パイプ情報テーブルとして作成する必要があります。ペリフェラルサンプルプログラムは、ベンダクラスドライバのパイプ情報を *r_usb_vendor_descriptor.c* ファイルの *uint16_t g_usb_psmpl_EpTbl1[]*に記載しています。

5.4.1 パイプ情報テーブル

パイプ情報テーブルは、以下 4 項目で構成されます。

1. パイプウインドウ選択レジスタ（0x64 番地）
2. パイプコンフィグレーションレジスタ（0x68 番地）
3. パイプマックスパケットサイズレジスタ（0x6C 番地）
4. ダミーデータ　(削除不可)

5.4.2 パイプ定義

ペリフェラルサンプルプログラムで使用しているパイプ情報テーブルの構成を以下に示します。パイプ情報テーブルのパイプ定義項目で設定可能な値は、*r_usb_cstd_defusbip.h* でマクロ定義しています。

＜パイプ情報テーブルの構成例＞

```
uint16_t g_usb_psmpl_EpTbl1[] =                          ← パイプ情報テーブル
{
  USB_PIPE4,                                             ←パイプ定義項目 1
  USB_BULK | USB_BFREOFF | USB_DBLBON | USB_SHTNAKON | USB_DIR_P_IN  | USB_EP4, ←パイプ定義項目 2
  USB_MAX_PACKET(64),                                    ←パイプ定義項目 3
  USB_NULL,                                              ←ダミーデータ
        :
  USB_PDTBLEND,
}
```

(1). パイプ定義項目 1：パイプウインドウ選択レジスタに設定する値を指定します。

RL78/USB, R8C ： PIPE4 から PIPE7

(2). パイプ定義項目 2：パイプコンフィグレーションレジスタの設定値を指定します。

転送タイプ	：	USB_BULK(Bulk 転送)/USB_INT(Interrupt 転送)のどちらかを指定してください。
BRDY 割り込み動作指定	：	USB_BFREOFF を指定してください。
ダブルバッファモード	：	USB_DBLBON/USB_DBLBOFF のどちらかを指定してください。
SHTNAK 動作指定	：	USB_SHTNAKON/USB_SHTNAKOFF のどちらかを指定してください。
転送方向	：	USB_DIR_P_OUT、USB_DIR_P_IN のどちらかを指定してください。
エンドポイント番号	：	パイプに対するエンドポイント番号(EP1～EP15)を指定してください。

- 転送タイプはパイプにより、設定可能な値が異なります。詳細は HM を参照してください。
- エンドポイントディスクリプタにあわせてパイプ情報を記載してください。
- 受信方向パイプ（USB_DIR_P_OUT）の場合は USB_SHTNAKON 設定としてください

(3). パイプ定義項目 3：エンドポイントのマックスパケットサイズを指定します。

マックスパケットサイズ指定 ：USB 仕様に準拠した値を設定してください。

- 当該エンドポイントのマックスパケットサイズを指定してください。
- BULK パイプのマックスパケットサイズは 64 を指定してください。

(4). その他制限事項

- 同時に通信が可能なエンドポイントの数だけパイプ情報が必要です。
- UPL でトランスファー単位の通信同期を取ってください。
- テーブルの最後には、必ず USB_PDTBLEND を書いてください。
- *R_usb_pstd_DriverRegistration()*関数でパイプ情報テーブルを登録します。
- USB-BASIC-F/Wは SET_CONFIGURATION リクエストでパイプ情報をレジスタに設定します。
- インタフェースの代替設定には対応していません。

5.5 ディスクリプタ情報

お客様のシステムに合わせてディスクリプタを作成する必要があります。ペリフェラルサンプルプログラムでは、ディスクリプタのサンプルテーブルを*r_usb_vendor_descriptor.c* ファイルに記載しています。
ディスクリプタ定義は以下の3種類で構成されます。

1. Standard Device Descriptor
   - uint8_t g_usb_psmpl_DeviceDescriptor[]
2. Configuration/Interface/Endpoint
   - uint8_t g_usb_psmpl_ConfigurationF_1[]
3. String Descriptor
   - uint8_t g_usb_psmpl_StringDescriptor0[]
   - uint8_t g_usb_psmpl_StringDescriptor1[]
   - uint8_t g_usb_psmpl_StringDescriptor2[]
   - uint8_t g_usb_psmpl_StringDescriptor3[]
   - uint8_t g_usb_psmpl_StringDescriptor4[]

(1). ID 登録

ディスクリプタに展開されるベンダ ID、プロダクト ID を設定してください。
例）ベンダ ID=0x0000, プロダクト ID=0x00FF の場合

```
#define     USB_VENDORID        (0x0000u)           /* Vendor ID */
#define     USB_PRODUCTID       (0x00FFu)           /* Product ID */
```

(2). デバイス情報

デバイス情報は選択した転送スピードに応じて設定値が異なります。

```
#ifdef  USB_LSPERI_PP
    #define USB_PVDR_BLENGTH    32          /* Low Speed (PIPE 6-7) */
    #define USB_DCPMAXP         (8u)        /* DCP max packet size */
    #define USB_EPNUMS          (2)         /* Endpoint number */
    #define USB_INTEPMAXP       (8u)        /* Interrupt pipe max packet size */
#endif  /* USB_LSPERI_PP */
#ifdef  USB_FSPERI_PP
    #define USB_PVDR_BLENGTH    46          /* Full Speed (PIPE 4-7) */
    #define USB_DCPMAXP         (64u)       /* DCP max packet size */
    #define USB_EPNUMS          (4)         /* Endpoint number */
    #define USB_INTEPMAXP       (64u)       /* Interrupt pipe max packet size */
#endif  /* USB_FSPERI_PP */
```

(3). その他報告情報

ディスクリプタに展開される以下の情報を設定してください。

```
    #define  USB_BCDNUM         (0x0200u)       /* bcdUSB */
    #define  USB_RELEASE        (0x0100u)       /* Release Number */
    #define  USB_CONFIGNUM      (1u)            /* Configuration number */
```

注意事項

1. 各ディスクリプタの詳細は USB Revision 2.0 specification の Chapter9 を参照してください。
2. ディスクリプタ定義を変更する場合は、エンドポイントディスクリプタに合わせてパイプ情報テーブル（*r_usb_vendor_descriptor.c* 内にサンプルテーブルを記載）も変更してください。
3. インタフェース番号は 0 から始まる連番で指定してください

5.6 USB-BASIC-F/Wをペリフェラル機能で動作させるには

本章では、サンプルコードを例にUSB-BASIC-F/Wペリフェラル機能で動作させるための手順を示します。

5.6.1 デバイス選択

Table 5-1に、USB-BASIC-F/Wが提供する各デバイス用の統合開発環境と USB モジュールの機能概要、ハードウェアリソースを示します。各デバイスはご使用になるデバイスに対応したフォルダを使用してください。.

Table 5-1 サンプルコードのハードウェアリソース

デバイス	統合開発環境	データレート	ハードウェアリソースフォルダ
R8C/3MU R8C/34U R8C/3MK R8C/34K	HEW	FullSpeed	src\HwResource
RL78/G1C	CS+	FullSpeed LowSpeed	
RL78/L1C	CS+	FullSpeed LowSpeed	

5.6.2 ユーザシステム定義ファイル(*r_usb_usrconfig.h*)

USB-BASIC-F/Wは、"*inc*"フォルダ内のユーザ定義情報ファイル(*r_usb_usrconfig.h*) を書き換えることにより、 USB-BASIC-F/Wの機能設定を行います。システムに合わせて、下記項目を変更してください。

(1). 転送スピードを指定 (RL78/USB のみ)

USB 転送スピードを設定します。

```
//#define     USB_LSPERI_PP     Low-Speed ペリフェラルデバイス
 #define     USB_FSPERI_PP     Full-Speed ペリフェラルデバイス
```

(2). グローバル変数を static 宣言する機能の指定

グローバル変数の static 宣言する機能を使用する場合は下記を追記します。

```
#define USB_STATIC_USE
```

(3). エラー時に while(1)する機能の指定

エラー時に while(1)する機能を使用する場合は下記を追記します。

```
#define USB_DEBUG_HOOK_USE
```

(4). Battery Charging 機能の指定 (RL78/USB のみ)

Battery Charging 機能を使用する場合は下記を追記します。

```
#define USB_PERI_BC_ENABLE
```

なお、以下の定義は、開発環境のプロジェクトファイルで行っています。

RL78G1C/RL78L1C	:	USB_FUNCSEL_PP = USB_PERI_PP RL78USB
R8C	:	USB_FUNCSEL_PP = USB_PERI_PP R8CUSB

5.6.3 USB-BASIC-F/W の変更

USB-BASIC-F/Wを動作させるには、以下のプログラムおよびヘッダファイルの変更が必要です。.
Renesas USB MCU用のサンプル関数を提供しますがユーザシステムにあわせて変更してください。

- 制御用 MCU の初期化、割り込みハンドラ、割り込み制御など(Table 5-2.参照)
- 指定時間待ち関数の時間調整(*usb_cpu_delay_xms()* 関数, *usb_cpu_delay_1u()* 関数)
  ループ処理等で指定時間のウェイトを行っています。ご使用のシステムに合わせてループ回数を変更するなど指定時間になるように調整してください。
- スケジューラ機能で使用するための USB 関連の割り込みを禁止および許可する関数の設定
  (*usb_cpu_int_disable()* 関数, *usb_cpu_int_enable()* 関数)

USB-BASIC-F/Wでは、USB割り込み禁止関数(*usb_cpu_int_disable()* 関数) がUSB割り込みを禁止し、USB割り込み許可関数 (*usb_cpu_int_enable()* 関数)がUSB割り込みを許可しています。ご使用のMCUに合わせて設定を行ってください。

Table 5-2 関数一覧

型	関数名	機能概要
void	usb_cpu_mcu_initialize(void)	MCU 初期処理（発振制御など）
void	usb_cpu_target_init(void)	システム初期化(ピン/ポート/割り込み設定など)
void	usb_cpu_set_pin_function(void)	MCU の USB 機能設定（端子設定など）
void	usb_cpu_usb_interrupt (void)	USB 割り込みハンドラ
void	usb_cpu_usbint_init (void)	USB 割り込み許可
void	usb_cpu_int_enable(void)	スケジューラ用 USB 割込み許可
void	usb_cpu_int_disable(void)	スケジューラ用 USB 割込み禁止
void	usb_cpu_intp0_enable(void)	RSK 上スイッチ用 INTP0 割り込み許可
void	usb_cpu_intp0(void)	RSK 上スイッチ用 INTP0 割り込み
void	usb_cpu_usb_resume_interrupt(void)	USB レジューム用 USB 割り込み
void	usb_cpu_delay_1us(uint16_t)	1us 単位の待ち処理
void	usb_cpu_delay_xms(uint16_t)	1ms 単位の待ち処理
void	usb_cpu_stop_mode(void)	ストップ命令実行

6. ペリフェラルコントロールドライバ(PCD)

6.1 基本機能

PCD は、対象デバイスを USB ファンクションとして動作させる際の H/W 制御用のプログラムです。USB-BASIC-F/Wは UPL の要求を解析し H/W の制御を行います。H/W 制御結果は API 関数の戻り値もしくはコールバック関数で UPL に通知されます。また、H/W からの要求もUSB-BASIC-F/Wに登録されたドライバ情報のコールバック関数で UPL に通知します。USB-BASIC-F/Wをペリフェラルデバイスとして機能させるには PCD を起動(6.2.1 参照)して、UPL の登録(6.2.3 参照)を行ってください。USB-BASIC-F/W の PCD 機能は以下のとおりです。

1. 接続ホストの USB ステート遷移検出と変化結果通知: [6.2.3章]
2. 接続ホストとのエニュメレーション: [6.2.7章]
3. USB リクエストの通知: [6.2.4章]
4. データ転送と転送結果通知: [6.2.5章]
5. USB ステート管理（USB ステート制御と制御結果通知）: [6.2.6章]
6. USB Battery Charging Specification Revision 1.2 で定義された Charging Port Detection [6.2.8章]

6.2 操作概要

6.2.1 PCD の起動

API 関数 *R_usb_pstd_PcdOpen()*でUSB-BASIC-F/Wを起動してください。

6.2.2 UPL の登録

UPL はTable 6-1 の情報を API 関数 *R_usb_pstd_DriverRegistration()*でUSB-BASIC-F/Wに登録します。USB-BASIC-F/W はグローバル変数 (*g_usb_PcdDriver*)に情報を保存します。

```
typedef struct
{
 uint16_t          *pipetbl;       /* Pipe Define Table address */
 uint8_t           *devicetbl;     /* Device descriptor Table address */
 uint8_t           *configtbl;     /* Configuration descriptor Table address */
 uint8_t           **stringtbl;    /* String descriptor Table address */
 usb_cb_info_t     statediagram;   /* Device status */
 usb_cb_trn_t      ctrltrans;      /* Control Transfer */
}usb_pcdreg_t;
```

Table 6-1 usb_pcdreg_t 構造体のメンバ

メンバ名	説明	備考
*pipetbl	パイプ情報テーブルのアドレスを登録してください。	
*devicetbl	Device ディスクリプタテーブルのアドレスを登録してください。	
*configtbl	Configuration ディスクリプタテーブルのアドレスを登録してください。	
**stringtbl	String ディスクリプタアドレステーブルのアドレスを登録してください。	
statediagram	USB ステート遷移時に起動する関数を登録してください。	
ctrltrans	クラスリクエスト、ベンダリクエスト発生時に起動する関数を登録してください。	

6.2.3 USB ステート遷移通知

USB-BASIC-F/W は USB ステート遷移などを UPL に通知する為に、USB-BASIC-F/Wに登録されている USB ステート遷移コールバック関数 (**g_usb_PcdDriver.statediagram*) を実行します。USB-BASIC-F/Wはコールバック関数の第 2 引数で以下の情報を UPL に通知します。UPL は USB ステートを解析してシステムに適合した処理を行ってください。

USB ステート遷移

USB_STS_DETACH	デタッチ検出
USB_STS_ATTACH	アタッチ検出
USB_STS_DEFAULT	デフォルトステート遷移（USB バスリセット検出）
USB_STS_ADDRESS	アドレスステート遷移（*Set_Address* リクエスト受信）
USB_STS_CONFIGURED	コンフィガードステート遷移（*Set_Configuration* リクエスト受信）
USB_STS_SUSPEND	サスペンドステート遷移（サスペンド検出）
USB_STS_RESUME	サスペンドステート解除（レジューム検出）
USB_PORTENABLE	D+ を Pull up（RL78/USB は"Pull up D-"した場合を含む）

6.2.4 コントロール転送通知

USB-BASIC-F/W はスタンダードリクエストに自動応答しエニュメレーション(6.2.7章参照) を行います。デバイスクラス（ベンダクラス）リクエストを受信した場合は、USB-BASIC-F/Wに登録されているコントロール転送コールバック関数 (**g_usb_pstd_Driver.ctrltrans*) を実行します。USB-BASIC-F/Wは UPL にコールバック関数の第 1 引数でTable 6-2の情報を通知します。UPL は USB リクエストを解析して UPL に準拠した処理を行ってください。また、以下の標準リクエストの場合もコントロール転送コールバック関数を実行します。

Get_Descriptor リクエストで *bRecipient* がインタフェース、*Clear_Feature* 及び *Set_Feature* リクエストを受信スタンダードリクエストでコールバックする場合は第 2 引数で以下のリクエスト種別を通知します

USB_CLEARSTALL	*Clear_Feature* リクエスト受信（STALL 解除）
USB_CLEARREMOTE	*Clear_Feature* リクエスト受信（remote wakeup 禁止）
USB_SETREMOTE	*Set_Feature* request リクエスト受信（remote wakeup 許可）
USB_SETSTALL	*Set_Feature* リクエスト受信（STALL 設定）
USB_RECIPIENT	*Get_Descriptor* リクエストで bRecipient がインタフェース

```
typedef struct
{
 union {
   struct {                        /* Characteristics of request */
      uint8_t  bRecipient:5;       /* Recipient */
      uint8_t  bType:2;            /* Type */
      uint8_t  bDirection:1;       /* Data transfer direction */
      uint8_t  bRequest:8;         /* Specific request */
   } BIT;
   uint16_t wRequest;              /* Control transfer request */
  } WORD;
 uint16_t   wValue;                /* Value */
 uint16_t   wIndex;                /* Index */
 uint16_t   wLength;               /* Length */
} usb_request_t;
```

Table 6-2 usb_request_t 構造体のメンバ

メンバ名	機能	備考
wRequest	wRequest の値が入ります。(USBREQ レジスタの値が入ります。) wRequest は共用体でビット参照が可能です。	
wValue	wValue の値が入ります。(USBVAL レジスタの値が入ります。)	
wIndex	wIndex の値が入ります。(USBINDEX レジスタの値が入ります。)	
wLength	wLength の値が入ります。(USBLENG レジスタの値が入ります。)	

6.2.5 USB-BASIC-F/Wに対する転送要求発行

UPL がデータ転送を行う場合は以下の構造体を引数として R_usb_pstd_TransferStart() を呼び出してください。USB-BASIC-F/W はグローバル変数(g_usb_LibPipe)に引数のアドレス情報を保存します。このため UPL はデータ転送が終わるまで引数の実態を保持してください。

```
struct usb_utr_t
{
 usb_strct_t   msginfo;       /* Message Info for F/W */
 usb_strct_t   pipenum;       /* Pipe number */
 usb_strct_t   status;        /* Transfer status */
 usb_strct_t   flag;          /* Flag */
 usb_cb_t      complete;      /* Call Back Function Info */
 uint8_t       *tranadr;      /* Transfer data Start address */
 uint16_t      *setup;        /* Setup packet(for control only) */
 uint16_t      pipectr;       /* Pipe control register */
 usb_leng_t    tranlen;       /* Transfer data length */
 uint8_t       dummy;         /* Adjustment of the byte border */
}
```

Table 6-3 usb_utr_t 構造体のメンバ

メンバ名	機能	備考
msginfo	USB-BASIC-F/Wが使用するメッセージ情報です。API 関数が設定します。	
pipenum	UPL でパイプ番号を指定してください。	
status	USB-BASIC-F/Wが下記のステータス情報を応答します。 USB_DATA_OK： データ転送（送信/受信）が正常終了した USB_DATA_SHT： 指定されたデータ長未満でデータ受信が正常終了した USB_DATA_OVR： 受信データがサイズオーバーした USB_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 USB_DATA_DTCH： デタッチを検出した USB_DATA_STALL： STALL 応答もしくは Max packet size エラーを検出した USB_DATA_STOP： データ転送を強制終了した	
complete	UPL で転送終了時に実行したいコールバック関数を指定してください。 型宣言は次のとおりです。 typedef void (*usb_cb_t)(usb_utr_t*);	
*tranadr	UPL で以下の情報を指定してください。 受信：受信データを格納するバッファアドレス 送信：送信データが格納してあるバッファアドレス tranlen で指定するデータ長より大きな領域を確保してください	
pipectr	UPL で PIPExCTR レジスタ情報を指定してください。 当該メンバの bit6 に従い DATA0/DATA1 のシーケンスビット制御を行います。 初期状態は USB_NULL を 2 回目以降はUSB-BASIC-F/Wが応答した値を設定してください。USB-BASIC-F/Wが PIPExCTR レジスタ情報を応答します。	
tranlen	UPL で以下の情報を指定してください。 受信： 受信したいデータ長 送信： 送信したいデータ長 送受信が可能な最大長は 65535 バイトです。 USB-BASIC-F/Wはデータ転送終了後に送受信の残りデータ長を格納します。	
Others	未使用	

6.2.6 USB-BASIC-F/Wに対する USB ステート遷移要求と通知

UPL が USB ステートを変更したい場合は API 関数 *R_usb_pstd_PcdChangeDeviceState()* を呼び出してください。API 関数の引数で制御内容を指示します。API 関数を呼び出したコンテキストにて処理を実行し戻り値で結果を通知します。

USB-BASIC-F/Wが管理している情報は API 関数 *R_usb_pstd_DeviceInformation()* で取得することができます。

6.2.7 エニュメレーション

USB-BASIC-F/W は USB ホストから送信される標準リクエストに自動応答します。USB-BASIC-F/W がサポートする標準リクエストは以下のとおりです。

(1). GET_DESCRIPTOR

(2). SET_ADDRESS

(3). SET_CONFIGURATION

(4). GET_STATUS

(5). GET_CONFIGURATION

(6). GET_INTERFACE

(7). CLEAR_FEATURE

(8). SET_FEATURE

(9). SET_INTERFACE

USB-BASIC-F/W はホストと接続が確立した（Configured ステート遷移）場合に、登録されている USB ステート遷移のコールバック関数(**g_usb_PcdDriver.statediagram*) で UPL に構成を通知します。UPL は第 2 引数の USB ステートを解析しシステムに適合した処理を行ってください。サンプルアプリケーションは USB_STS_CONFIGURED ステート遷移でグローバル変数を初期化しデータ転送を許可します。

USB-BASIC-F/Wのコントロール転送でステージ遷移およびリクエスト情報の取得タイミングにタイムラグがあります。したがって、リクエスト情報が得られる前に新しいコントロール転送が開始されると、ステート情報とリクエストの間の不一致が生じる可能性があります。

6.2.8 ペリフェラルバッテリチャージング制御 (PBC)

PBC は、対象デバイスを USB Battery Charging Specification Revision 1.2 で定義された Charging Port Detection(CPD)を動作させる際の H/W 制御用プログラムです。

USB-BASIC-F/Wが USB ステート遷移（USB_STS_ATTACH）を UPL に通知した直後に CPD を実行します。CPD の結果は、USB-BASIC-F/Wが USB ステート遷移（USB_PORTENABLE）の第 1 引数で UPL に通知します。

UPL に通知する第 1 引数の値は以下の通りです。

0： Standard Downstream Port (SDP) を検出

1： Charging Downstream Port (CDP) を検出

2： Dedicated Charging Port (DCP) を検出

PBC の処理フローをFigure 6.1以下に示します。

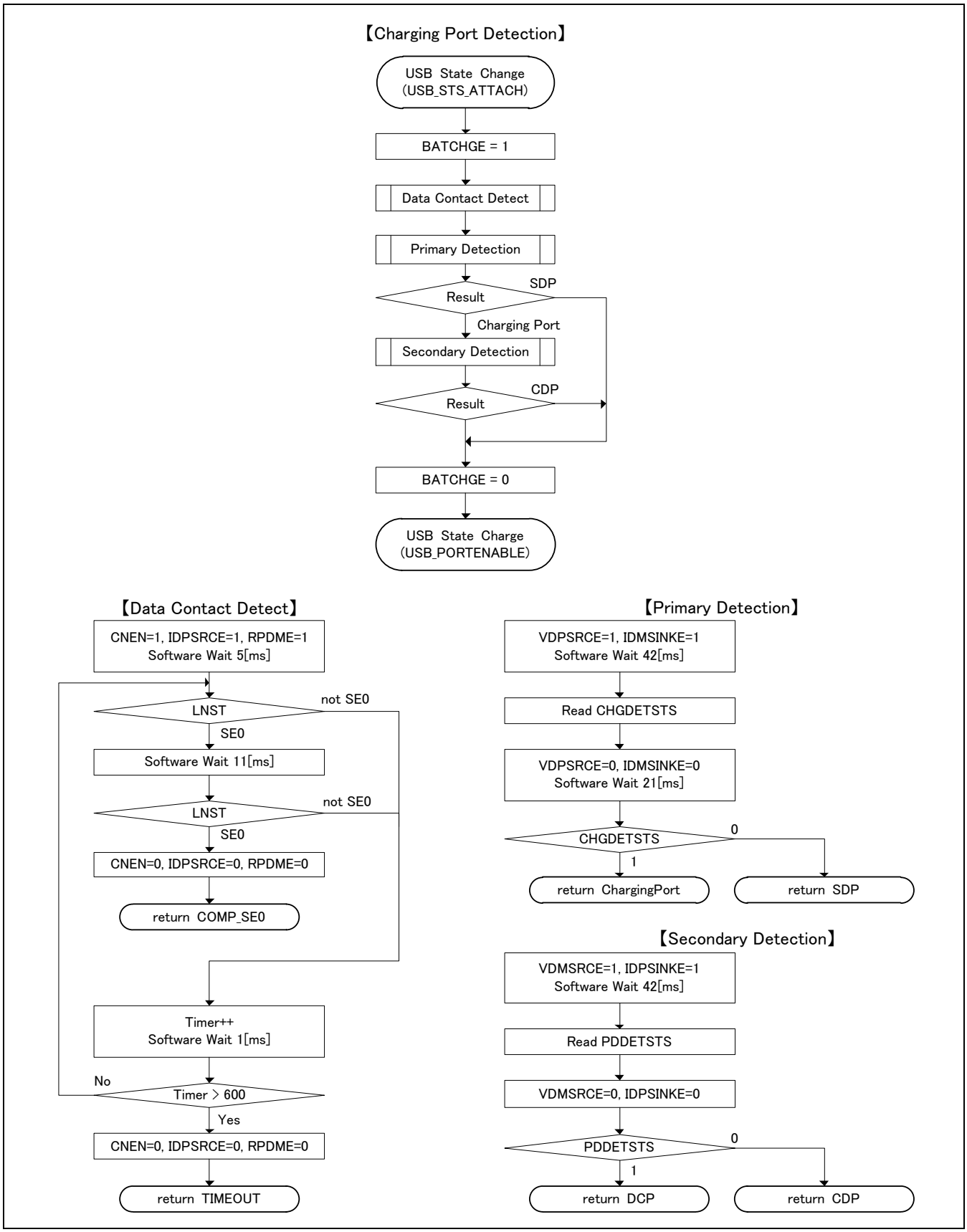

Figure 6.1 PBC フローチャート

6.2.9 USB-BASIC-F/W使用時の注意事項

1. USB-BASIC-F/Wはサスペンドが発生してもデータ転送を中断しません。
2. USB-BASIC-F/Wはデタッチ検出時にはデータ転送を停止します。
3. USB-BASIC-F/Wはマルチコンフィグレーション設定に対応していません。
4. USB-BASIC-F/Wはインタフェース代替設定に対応していません（パイプレジスタの変更ができません）。
5. USB-BASIC-F/Wには以下の機能もあります。詳細は6.3を参照してください。.
   （1） USB ポートの許可、禁止制御
   （2） USB ステート遷移（リモートウェイクアップ）
   （3） パイプの STALL
   （4） データ転送の中断
   （5） Control 転送用の FIFO バッファアクセス

6.3 PCD API 関数

UPL は API 関数で H/W の制御要求を行います。API 関数は *r_usb_pdriverapi.c* ファイルにあります。file. USB-BASIC-F/Wの API 関数を使用する場合は、Table 6-4に示した順番に従いヘッダファイルをインクルードしてください。また、Table 6-5 に PCD API 関数一覧を示します。

Table 6-4 PCD API ヘッダファイル一覧

ファイル名	説明	備考
r_usb_ctypedef.h	変数型定義	
r_usb_ckernelid.h	システムヘッダファイル	
r_usb_cdefusbip.h	USB ドライバ用各種定義	
r_usb_api.h	USB ドライバ API 関数定義	

Table 6-5 List of PCD API Function

関数名	説明	備考
R_usb_pstd_PcdTask	PCD タスク	
R_usb_pstd_PcdOpen	PCD タスク起動（PCD タスク初期化処理）	
R_usb_pstd_DriverRegistration	UPL 登録	
R_usb_pstd_TransferStart	データ転送実行要求	
R_usb_pstd_TransferEnd	データ転送強制終了要求	
R_usb_pstd_PcdChangeDeviceState	USB ステート遷移要求	
R_usb_pstd_DeviceInformation	USB デバイス情報を取得する	
R_usb_pstd_SetStallPipe0	パイプ 0 の PID を STALL に設定する	
R_usb_pstd_SetPipeStall	パイプ 0 以外のパイプの PID を STALL に設定する	
R_usb_pstd_ControlRead	コントロールリード転送用 FIFO アクセス実行要求	
R_usb_pstd_ControlWrite	コントロールライト転送用 FIFO アクセス実行要求	
R_usb_pstd_ControlEnd	コントロール転送終了要求	
R_usb_pstd_SetPipeRegister	パイプ情報をレジスタに設定する。	

6.4 PCD コールバック関数

USB-BASIC-F/Wは USB ステート遷移、データ転送終了などをコールバック関数で UPL に通知します。コールバック関数はドライバ登録時と、データ転送要求の API 関数呼び出し時に UPL が指定します。コールバック関数を新規作成する場合は API 関数使用時と同様にTable 6-4 に示した順番に従いヘッダファイルをインクルードしてください。またTable 6-6に PCD コールバック関数一覧を示します。

Table 6-6 PCD コールバック関数一覧

関数名	説明	備考
*g_usb_PcdDriver.statediagram	USB ステート遷移検出時のコールバック	
*g_usb_PcdDriver.ctrltrans	コントロール転送のコールバック	
* g_usb_LibPipe[pipe]->complete	データ転送のコールバック	

6.5 API 関数、コールバック関数詳細

API関数およびコールバック関数の詳細は以下のとおりです。

R_usb_pstd_PcdTask

PCD タスク

形式

void	R_usb_pstd_PcdTask(void)

引数

―	―

戻り値

―	―

解説

*usb_pstd_pcd_task()*を呼び出します。

*usb_pstd_pcd_task ()*は USB 割込みに応じた処理を実行します。

- USB スタンダードリクエストに応答します。
- クラスリクエスト、ベンダリクエスト検出時は UPL が登録したコントロール転送コールバック関数を呼び出します。
- USB ステート遷移検出時は UPL が登録した USB ステート遷移コールバック関数を呼び出します。

*usb_pstd_pcd_task ()*は API 関数を経由して要求されたデータ転送を行います。

- データ転送終了時は API 関数で指定されたコールバック関数を呼び出します。

補足

1. 本関数はスケジューラ処理を行うループ内で呼び出してください。
2. 無効なメッセージを受け取った場合はフック関数(R_usb_cstd_debug_hook())をコールします。フック関数については、9.3章を参照してください。

使用例

```
void main(void)
{
 usb_psmpl_main_init();
 while( 1 )
 {
  if(R_usb_cstd_Scheduler() == USB_FLGSET )
  {
       R_usb_pstd_PcdTask();
    usb_psmpl_apl_task();
  }
 }
}
```

R_usb_pstd_PcdOpen

PCD タスク起動

形式

void	R_usb_pstd_PcdOpen(void)

引数

—	—

戻り値

—	—

解説

PCD で使用するグローバル変数の初期化等を行います。

補足

1. USB-BASIC-F/W起動時に呼び出してください。

使用例

```
void usb_psmpl_main_init(void)
{
 usb_cpu_target_init();                    /* Target board initialize */

 /* USB-IP initialized */
 R_usb_pstd_PcdChangeDeviceState(USB_DO_INITHWFUNCTION)

 /* PCD driver open & registration */
  R_usb_pstd_PcdOpen();                    /* PCD task open */
  usb_psmpl_driver_registration();         /* Sample driver registration */

 /* USB-IP is set to the peripheral function */
 R_usb_pstd_PcdChangeDeviceState(USB_DO_SETHWFUNCTION);
}
```

R_usb_pstd_DriverRegistration

ペリフェラルデバイスクラスドライバ(PDCD)登録

形式

void	R_usb_pstd_DriverRegistration(usb_pcdreg_t *registinfo)

引数

*registinfo	クラスドライバ構造体

戻り値

―	―

解説

UPL をUSB-BASIC-F/Wに登録します。初期設定時に UPL から本 API をコールしてください。

補足

1. 登録可能なドライバは 1 つです。登録する情報は、Table 6-1を参照してください。.

使用例

```
void usb_psmpl_driver_registration(void)
{
 usb_pcdreg_t  driver;

 /* Driver registration */
 driver.pipetbl    = g_usb_psmpl_EpTbl1;
 driver.devicetbl     = g_usb_psmpl_DeviceDescriptor;
 driver.configtbl     = g_usb_psmpl_ConfigurationF_1;
 driver.stringtbl     = g_usb_psmpl_StringPtr;
 driver.statediagram = &usb_apl_change_device_state;
 driver.ctrltrans     = &usb_psmpl_control_transfer;
  R_usb_pstd_DriverRegistration(&driver);
}
```

R_usb_pstd_TransferStart

データ転送実行要求

形式

usb_er_t R_usb_pstd_TransferStart(usb_utr_t * utr_table)

引数

* utr_table	データ転送構造体(本構造体については、Table 6-3を参照してください。)

戻り値

USB_E_OK	成功
USB_E_ERROR	失敗、引数エラー
USB_E_QOVR	オーバラップ（パイプ使用中）

解説

各パイプのデータ転送要求を行います。
データ転送は、指定データサイズを満たした場合、ショートパケットを受信した場合、およびエラー発生で終了します。

データ転送終了時に、引数の構造体メンバ（*utr_table. complete*）のコールバック関数を呼び出します。このコールバック関数の引数（*utr_table*）には、送受信の残りデータ長、ステータス、及び転送終了の情報が設定されています。

同一パイプでデータ転送を再開する場合は、前回転送終了したパイプステータス（データトグル）と今回通信要求するパイプステータスをあわせる必要があります。引数の構造体メンバ（*utr_table.pipectr*）でパイプステータスを設定してください。なおパイプステータスは USB リセット、STALL 解除などの要因が発生したら"DATA0"に初期化する必要があります。

データ転送中のパイプに対して転送開始要求が発生した場合は USB_E_QOVR を応答します。

補足

1. コントロール転送はサポートしていません。
2. 受信したデータがマックスパケットサイズの n 倍、かつ受信予定サイズに満たない場合は、データ転送の途中であると判断しコールバックが発生しません。

使用例

```
usb_utr_t   g_usb_PsmplTrnMsg[USB_TBL_MAX];
void usb_pvndr_data_transfer(usb_pipe_t pipe)
{
 /* PIPE Transfer set */
 g_usb_PsmplTrnMsg[pipe].pipenum = pipe;
 g_usb_PsmplTrnMsg[pipe].tranadr = g_usb_PsmplTrnPtr[pipe];
 g_usb_PsmplTrnMsg[pipe].tranlen = g_usb_PsmplTrnSize[pipe];
 g_usb_PsmplTrnMsg[pipe].pipectr = g_usb_PsmplPipeCtr[pipe];
 g_usb_PsmplTrnMsg[pipe].setup= USB_NULL;
 g_usb_PsmplTrnMsg[pipe].complete= (usb_cb_t)&usb_pvndr_transfer_result;
  R_usb_pstd_TransferStart((usb_utr_t *)&g_usb_PsmplTrnMsg[pipe]);
}
```

R_usb_pstd_TransferEnd

データ転送強制終了要求

形式

usb_er_t R_usb_pstd_TransferEnd(usb_pipe_t pipe, usb_strct_t_t msginfo)

引数

pipe	パイプ番号
msginfo	通信ステータス

戻り値

USB_E_OK	成功
USB_E_ERROR	失敗、引数エラー
USB_E_QOVR	オーバラップ（転送終了中のパイプに対する転送終了要求）

解説

引数 *msginfo* に以下の値を設定し、USB-BASIC-F/Wにデータ転送の強制終了を要求します。

・USB_DO_TRANSFER_STP： データ転送強制終了（コールバックします。）
・USB_DO_TRANSFER_TMO： データ転送タイムアウト（コールバックしません。）

msginfo=USB_DO_TRANSFER_STP による強制終了は、データ転送要求時(*R_usb_pstd_TransferStart*)に設定したコールバック関数で転送終了を通知します。コールバック関数の引数（*usb_utr_t*）で、送受信の残りデータ長、パイプコントロールレジスタの値、転送ステータス=USB_DATA_STOP を設定します。

データ未転送のパイプに対する強制終了要求が発生した場合は USB_E_QOVR を応答します。

補足

1. データ送信を中断する場合は SIE 側の FIFO バッファはクリアされません。
   FIFO がダブルバッファで送信時は FIFO バッファ内に未送信データが残る場合があります。
2. RL78/USB の場合、引数 pipe がパイプ 0～パイプ 3 の場合は USB_E_QOVR のエラーを返します。また、パイプ 8 以上の場合は USB_E_ERROR のエラーを返します

使用例

```
void  usb_smp_task(void)
{
  R_usb_pstd_TransferEnd(USB_PIPE4, USB_DO_TRANSFER_STP);
}
```

R_usb_pstd_PcdChangeDeviceState

USB デバイスステート遷移要求

形式

usb_er_t R_usb_pstd_PcdChangeDeviceState(usb_strct_t msginfo)

引数

msginfo	変更させたい USB ステート

戻り値

USB_E_OK	成功
USB_E_ERROR	失敗、引数エラー

解説

引数 msginfo に以下の値を設定し、USB-BASIC-F/Wに USB ステート遷移を要求します。

・USB_DO_PORT_ENABLE

USB データライン（D+/D-ライン）のプルアップ要求（ホストに対して接続通知）を行います。

・USB_DO_PORT_DISABLE

USB データライン（D+/D-ライン）のプルアップ解除要求（ホストに対して切断通知）を行います。

・USB_DO_REMOTEWAKEUP

リモートウェイクアップ実行要求を行います。

・USB_DO_INITHWFUNCTION

USB-IP を起動し SW リセットを行います。USB-BASIC-F/W起動前に実行してください。

・USB_DO_SETHWFUNCTION

USB-IP の機能をペリフェラル設定にします。UPL 登録後に実行してください。

補足

1. USB-BASIC-F/Wが接続/切断を割り込みで検出した場合は、USB-BASIC-F/Wが自動的に USB データラインのプルアップ/解除を行います。
2. 当該関数は PCD タスクを介さずに処理を実行します。

使用例

```
void  usb_smp_task(void)
{
  R_usb_pstd_PcdChangeDeviceState(USB_DO_INITHWFUNCTION);
  R_usb_pstd_PcdOpen();                    /* PCD task open */
  usb_psmpl_driver_registration();         /* Sample driver registration */
  R_usb_pstd_PcdChangeDeviceState(USB_DO_SETHWFUNCTION);
  :
  :

}
```

R_usb_pstd_DeviceInformation

USB デバイスステート情報を取得

形式

void	R_usb_pstd_DeviceInformation (uint16_t *table)

引数

*table	取得した情報が格納されるテーブルのアドレス

戻り値

―	―

解説

USB デバイス情報を取得します。引数（**table*）に指定したアドレスに以下の情報を格納します。

[0]：USB ステート（INTSTS0 レジスタの VBSTS、DVSQ フィールド値）

[1]：コンフィギュレーション番号（SET_CONFIGURATION リクエストの wValue 値）

[2]：インタフェース数（g_usb_PcdDriver.configtbl[USB_CON_NUM_INTERFACE]）

[3]：リモートウェイクアップフラグ（許可：USB_YES, 禁止：USB_NO）

補足

1. 引数**table* は 4word の領域を用意してください。

使用例

```
void  usb_smp_task(void)
{
 uint16_t  res[4];
  :
  R_usb_pstd_DeviceInformation(res);
  :
}
```

R_usb_pstd_SetStallPipe0

パイプ 0 の PID を STALL 設定(コントロール転送用)

形式

void	R_usb_pstd_SetStallPipe0(void)

引数

―	―

戻り値

―	―

解説

PIPE0 の PID に STALL を設定します。

補足

USB リクエストに STALL 応答したい場合は本関数を呼び出してください。

本関数実行後に R_usb_pstd_ControlEnd(USB_CTRL_END)を実行しても STALL 応答します。

PID ついては、MCU のハードウェアマニュアルを参照してください。

使用例

```
void usb_psmpl_control_transfer(usb_request_t *data1, uint16_t data2)
{
 if (data1->TypeRecip == USB_INTERFACE )
 {
    R_usb_pstd_SetStallPipe0();
 }
 else
 {
  usb_smpl_vendore_request(data1);
 }
}
```

R_usb_pstd_SetPipeStall

パイプ x の PID を STALL 設定(データ転送用)

形式

void	R_usb_pstd_SetPipeStall(usb_pipe_t pipe)

引数

pipe	パイプ番号

戻り値

USB_E_OK	成功
USB_E_ERROR	失敗、引数エラー

解説

引数で指定したパイプ番号の PID に STALL を設定します。データ転送リクエストに対して STALL 応答する場合に、この API をコールしてください。

補足

1. 引数 pipe にパイプ 0 の設定はエラーです。*R_usb_pstd_SetStallPipe0()*関数を使用してください。
2. PID ついては、MCU のハードウェアマニュアルを参照してください。

使用例

```
void  usb_smp_task(void)
{
  :
  R_usb_pstd_SetPipeStall(USB_PIPE4);
  :
}
```

R_usb_pstd_ControlRead

コントロールリード転送用 FIFO アクセス実行要求

形式

uint16_t R_usb_pstd_ControlRead (usb_leng_t bsize, uint8_t *table)

Argument

bsize	送信データバッファのサイズ
*table	送信データバッファのアドレス

戻り値

USB_WRITESHRT	データ書き込み終了（ショートパケットデータ書き込み）
USB_WRITING	データ書き込み中（継続データあり：NULL パケット送信準備中）
USB_FIFOERROR	FIFO アクセスエラー発生

解説

コントロールリード転送のデータステージで使用します。
引数（*table*）が示す領域からデータを読み出し、FIFO バッファに書き込みます。

USB-BASIC-F/Wはショートパケットを送信するか OUT トークンが発生するまでデータステージを継続します。

補足

1. コントロールリード転送のデータステージで呼び出してください。
2. 指定データサイズがマックスパケットサイズと等しい場合は指定データ送信後の IN トークンで NULL パケットを送信します

使用例

```
uint8_t  g_usb_smp_buff[16];
void usb_smpl_vendore_reques1(usb_request_t *data1, uint16_t data2)
{
 if (data1->TypeRecip == USB_INTERFACE )
 {
    R_usb_pstd_ControlRead(10,(uint8_t*)&g_usb_smp_buff);
 }
 else
 {
  R_usb_pstd_SetStallPipe0();
 }
}
```

R_usb_pstd_ControlWrite

コントロールライト転送用 FIFO アクセス実行要求

形式

void	R_usb_pstd_ControlWrite(usb_leng_t bsize, uint8_t *table)

引数

bsize	受信データバッファのサイズ
*table	受信データバッファのアドレス

戻り値

―	―

解説

コントロールライト転送のデータステージで使用します。
FIFO バッファからデータを読み出し、引数（*table*）が示す領域に書き込みます。

補足

1. コントロールライト転送のデータステージで呼び出してください。
2. 指定したデータ長までデータを読み出します。
3. 受信データが指定したデータ長に満たなくてもショートパケット受信時は読み出しを終了します。

使用例

```
uint8_t  g_usb_smp_buff[16];
void usb_smpl_vendore_reques2(usb_request_t *data1, uint16_t data2)
{
 if (data1->TypeRecip == USB_INTERFACE )
 {
    R_usb_pstd_ControlWrite(10,(uint8_t*)&g_usb_smp_buff);
 }
 else
 {
  R_usb_pstd_SetStallPipe0();
 }
}
```

R_usb_pstd_ControlEnd

コントロール転送終了要求

形式

void	R_usb_pstd_ControlEnd(uint16_t status)

引数

status	ステータス

戻り値

—	—

解説

コントロール転送のステータスステージで使用します。
引数（*status*）には以下のいずれかの値を設定してください。

・USB_CTRL_END

ステータスステージを正常に終了させる。

・USB_DATA_STOP

ステータスステージでホストに NAK を返す。

・USB_DATA_ERR / USB_DATA_OVR

ステータスステージでホストに STALL を返す。

補足

1. コントロール転送のステータスステージで呼び出してください。
2. 引数（status）に USB_CTRL_END を指定した場合は PID=BUF 設定と CCPL=1 を行います。
3. PID=STALL 状態の場合（R_usb_pstd_SetStallPipe0()関数実行後など）は、で引数（status）に USB_CTRL_END を指定しても STALL 応答します。
4. PID、BUF および CCPL ついて、MCU のハードウェアマニュアルを参照してください。

使用例

```
uint8_t  g_usb_smp_buff[16];
void usb_smpl_vendore_reques3(usb_request_t *data1, uint16_t data2)
{
 if (data1->TypeRecip == USB_INTERFACE )
 {
    R_usb_pstd_ControlEnd(USB_CTRL_END);
 }
 else
 {
    R_usb_pstd_ControlEnd(USB_DATA_ERR);
 }
}
```

R_usb_pstd_SetPipeRegister

パイプ情報を USB レジスタに設定

形式

void	R_usb_pstd_SetPipeRegister(uint16_t* table, uint16_t command)

引数

table	パイプ情報テーブル
command	コマンド (USB_YES/USB_NO)

戻り値

―	―

解説

・コマンドが"USB_NO"の場合

パイプ情報テーブルで指定された全パイプを未使用状態に設定します。

・コマンドが"USB_YES"の場合

パイプ情報テーブルで指定された全パイプを未使用状態に設定します。

未使用状態にしたのち、パイプ情報テーブル従い全パイプの設定を行います。

補足

1. Set_Configuration リクエスト受信時はUSB-BASIC-F/Wが実施します。

使用例

```
void usb_pstd_set_configuration3(void)
{

 if( g_usb_PcdRequest.TypeRecip == USB_DEVICE )
 {
      :
   if( g_usb_PcdConfigNum != (uint8_t)g_usb_PcdRequest.wValue )
   {
      /* Configuration number set */
      g_usb_PcdConfigNum   = (uint8_t)g_usb_PcdRequest.wValue;
      R_usb_pstd_SetPipeRegister(g_usb_PcdDriver.pipetbl, USB_NO);
   }
   if( g_usb_PcdConfigNum > 0 )
   {
      R_usb_pstd_SetPipeRegister(g_usb_PcdDriver.pipetbl, USB_YES);
   }
   return;
      :
 }
 R_usb_pstd_SetStallPipe0();

}
```

*g_usb_PcdDriver.statediagram

USB ステート遷移検出時のコールバック

形式

void	(*g_usb_PcdDriver.statediagram)((uint16_t)data1, (uint16_t)device_state);

引数

data1	通常は未使用、Set_Configuration 時は構成番号
device_state	USB ステート

戻り値

―	―

解説

USB ステート遷移が発生したことを UPL に通知します。

- レジューム検出
  (*g_usb_PcdDriver.statediagram)(USB_NO_ARG, USB_STS_RESUME) ;
- ステート遷移割込み検出
  (*g_usb_PcdDriver.statediagram)(USB_NO_ARG, USB_STS_DEFAULT);
  (*g_usb_PcdDriver.statediagram)(USB_NO_ARG, USB_STS_ADDRESS);
  (*g_usb_PcdDriver.statediagram)(g_usb_PcdConfigNum, USB_STS_CONFIGURED);
  (*g_usb_PcdDriver.statediagram)(USB_NO_ARG, USB_STS_SUSPEND);
- デタッチ検出
  (*g_usb_PcdDriver.statediagram)(USB_NO_ARG, USB_STS_DETACH);
- アタッチ検出
  (*g_usb_PcdDriver.statediagram)(USB_NO_ARG, USB_STS_ATTACH)
- D+を Pull up に設定
  (*g_usb_PcdDriver.statediagram)(USB_NO_ARG, USB_PORTENABLE)

補足

1. USB リセット検出時のコールバックでデバイスの通信速度は通知しません。
2. *Set_Configuration* リクエスト受信時は構成番号に変更が無い場合はコールバックしません。
3. *Set_Configuration* リクエスト受信時に構成番号="0"の場合は ADDRESS ステート通知します。

使用例

Processing of the function that callback is done as an example.

```
void usb_apl_change_device_state(uint16_t data, uint16_t state)
 case USB_STS_CONFIGURED:      /* Device configured */
  configuratuion_num  = (uint8_t)data;
  usb_psmpl_open();
 break;
 case USB_STS_ATTACH:          /* Device attach */
 break;
 case USB_STS_DETACH:          /* Device detach */
   configuratuion_num= (uint8_t)0;
 break;
  case USB_STS_SUSPEND:        /* Device suspend */
 case USB_STS_RESUME:          /* Device resume */
 break;
 case USB_STS_DEFAULT:         /* Device default */
 case USB_STS_ADDRESS:         /* Device addressed */
   configuratuion_num= (uint8_t)0;
 break;
 case USB_PORTENABLE:          /* D+ line pull up */
 break;
 default:
  usb_apl_dummy_function(data,state);
 break;
 }
}
```

*g_usb_PcdDriver.ctrltrans

コントロール転送のコールバック

形式

void	(*g_usb_PcdDriver.ctrltrans)((usb_request_t *)request, (uint16_t)data;

引数

request	USB リクエスト
data	コントロール転送ステージ

戻り値

―	―

解説

リクラスリクエスト、ベンダリクエストのコントロール転送が発生したことを UPL に通知します。

第 2 引数の転送ステージ種別は以下のとおりです。詳細は HM を参照ください。

- USB_CS_IDST /* Idle or setup stage */
- USB_CS_RDDS: /* Control read data stage */
- USB_CS_WRDS: /* Control write data stage */
- USB_CS_WRND: /* Control write nodata status stage */
- USB_CS_RDSS: /* Control read status stage */
- USB_CS_WRSS: /* Control write status stage */
- USB_CS_SQER: /* Control sequence error */

(*g_usb_PcdDriver.ctrltrans)((usb_request_t*)&g_usb_PcdRequest, (uint16_t)intseq);

以下の標準リクエストを受信した場合も UPL に通知します。

- Clear_Feature リクエストでリモートウェイクアップが解除された場合
  (*g_usb_PcdDriver. ctrltrans)((usb_request_t*)&g_usb_PcdRequest, USB_CLEARREMOTE);
- Clear_Feature リクエストで ENDPOINT の STALL 設定が解除された場合
  (*g_usb_PcdDriver. ctrltrans)((usb_request_t*)&g_usb_PcdRequest, USB_ CLEARSTALL);
- Get_Descriptor リクエストで bRecipient が USB_INTERFACE の場合
  (*g_usb_PcdDriver. ctrltrans)((usb_request_t*)&g_usb_PcdRequest, USB_RECIPIENT);
- Set_Feature リクエストでリモートウェイクアップが許可された場合
  (*g_usb_PcdDriver. ctrltrans)((usb_request_t*)&g_usb_PcdRequest, USB_SETREMOTE);
- Set_Feature リクエストで ENDPOINT の STALL 設定された場合
  (*g_usb_PcdDriver. ctrltrans)((usb_request_t*)&g_usb_PcdRequest, USB_SETSTALL);

補足

1. USB-BASIC-F/Wは UPL が STALL 設定したパイプ情報を記憶していません。

   Clear_Feature リクエストを正常に受け付けると当該コールバックで UPL に通知されますので UPL 側で STALL 設定しているパイプに対する STALL 解除であるか判断してください。

2. Get_Interface リクエストで代替通知を要求された場合は必ず 0 を応答します

使用例

例としてコールバックされた関数内の処理を記述します。

```
void usb_psmpl_control_transfer(usb_request_t *request, uint16_t data)
{
 g_usb_SmplRequest= *request;

 switch(g_usb_SmplRequest.wRequest & USB_BMREQUESTTYPETYPE)
 {
  case USB_STANDARD:
    switch( data )
    {
       case USB_SETREMOTE:
          /* Enable Remote wakeup */
       break;
       case USB_CLEARREMOTE:
          /* Disable Remote wakeup */
       break;
       case USB_SETSTALL:
          /* Set stall */
       break;
       case USB_CLEARSTALL:
          /* Clear stall */
       break;
       default:
       break;
    }
  break;
  case USB_CLASS:
    R_usb_pstd_ControlEnd(USB_DATA_ERR);
  break;
  case USB_VENDOR:
    switch( data )
    {
       case USB_CS_IDST:          /* Idle or setup stage */
       case USB_CS_RDDS:          /* Control read data stage */
       case USB_CS_WRDS:          /* Control write data stage */
       case USB_CS_WRND:          /* Control write nodata status stage */
       case USB_CS_RDSS:          /* Control read status stage */
       case USB_CS_WRSS:          /* Control write status stage */
       case USB_CS_SQER:          /* Control sequence error */
       default:                   /* Illegal */
       break;
    }
    R_usb_pstd_SetStallPipe0();
  break;
  default:                                  /* Special function */
  break;
 }
}
```

*g_usb_LibPipe [pipe]->complete

データ転送終了のコールバック

形式

void (*g_usb_LibPipe[pipe]->complete)((usb_utr_t*)g_usb_LibPipe[pipe]);

引数

g_usb_LibPipe Transfer message

戻り値

― ―

解説

データ転送終了および中断を UPL に通知します。

補足

1. 転送要求時のメッセージを応答します。USB-BASIC-F/Wが更新する構造体メンバを Table 6-7に示します。
2. タイムアウト中断（R_usb_pstd_TransferEnd()関数で USB_DO_TRANSFER_TMO 指定）はコールバックしません。

Table 6-7 usb_utr_t 構造体のメンバ

メンバ名	更新有無	機能	補足
tranlen	更新有	転送要求に対する残りデータ長が通知されます。 （tranlen = 転送要求時の tranlen - 実際に送受信したデータ長）	
status	更新有	以下に、通信結果を示します。 USB_DATA_OK: データ転送（送信/受信）が正常終了した場合 USB_DATA_SHT: データ転送が指定されたデータ長未満で終了した場合 USB_DATA_OVR: 受信データがサイズオーバーした場合 USB_DATA_STOP: データ転送を強制終了した場合	
pipectr	更新有	パイプコントロールレジスタの値が設定されます（PIPExCTR レジスタ）	
上記以外	更新無	転送要求した内容が保存されています。	

使用例

例としてコールバックされた関数内の処理を記述します。

```
void usb_psmpl_transfer_result(usb_utr_t *mess)
{
 switch(mess->status)
 {
  case USB_DATA_OK:
  case USB_DATA_SHT:
    if (mess->keyword == USB_PIPE4)
    {
       usb_psmpl_DataTransfer(512, (uint8_t*)&g_usb_SmplTrnData);
    }
  break;
  case USB_DATA_OVR:
    if (mess->keyword == USB_PIPE5)
    {
       usb_psmpl_DataTransfer(512, (uint8_t*)&g_usb_SmplTrnData);
    }
  break;
 }
}
```

7. ホストサンプルプログラム(UPL)

本章では MCU が RL78 の場合について説明します。Low-Speed デバイスはバルク転送の通信を行いません。ユーザシステムが Low-Speed デバイスの場合、バルク転送に関する記述はスキップしてください。ユーザシステムの MCU が Low-Speed をサポートしない場合、Low-Speed に関する記述はスキップしてください。

ホストサンプルアプリケーションは、USB-BASIC-FWが動作する USB デバイスを接続したとき、接続された USB デバイスとデータ通信を行います。動作内容については、3.6章を参照してください。

7.1 動作環境

動作環境をFigure 7-1と Figure 7-2 に示します。

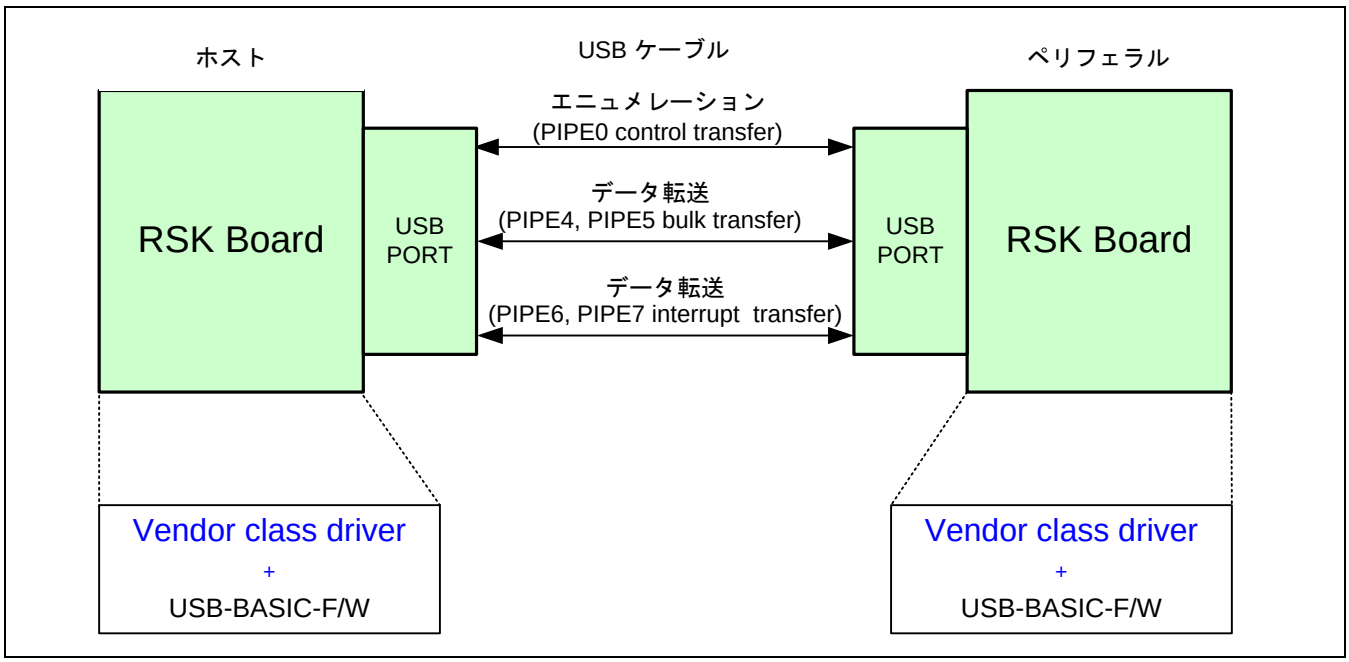

Figure 7-1 Full-Speed 動作環境例

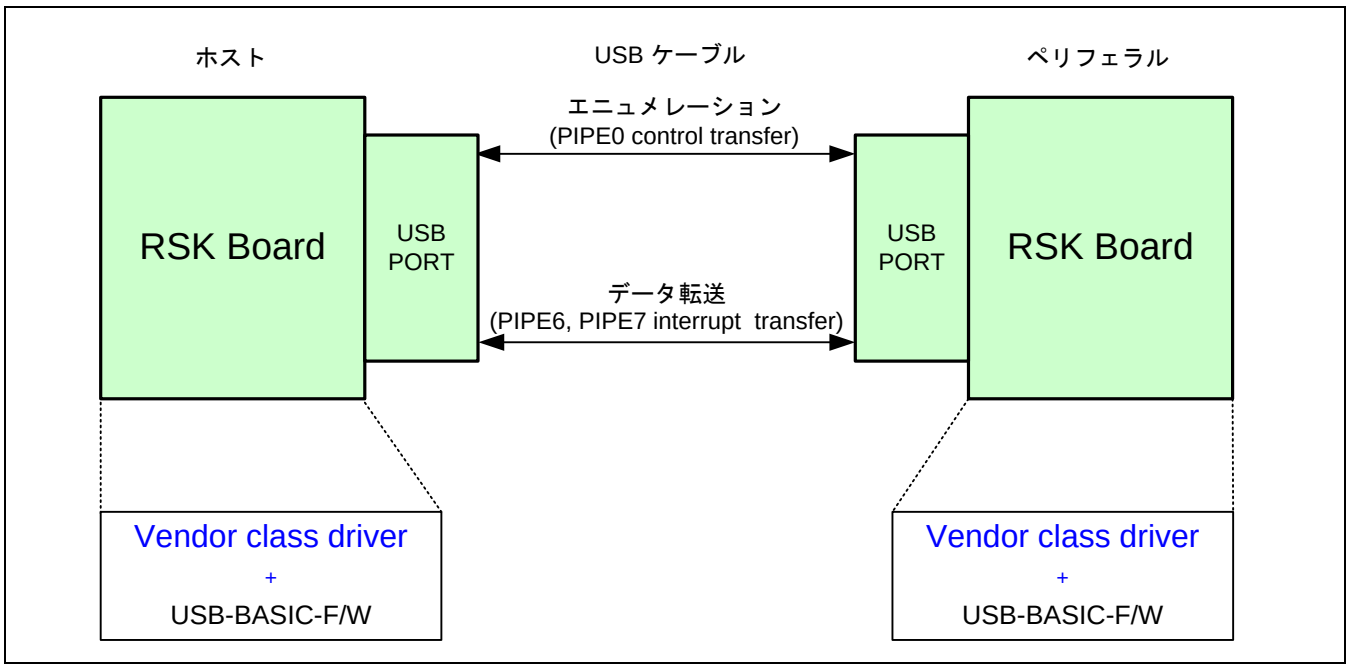

Figure 7-2 Low-Speed 動作環境例

7.2 ホストサンプルプログラムの説明

USB-BASIC-F/Wのホストサンプルプログラムは、接続されたデバイスによって Full-Speed デバイスまたは Low-Speed デバイスで動作します。サンプルプログラムは、データ転送用のベンダクラスドライバとサンプルアプリケーションを搭載しています。 バルク転送は、PIPE4,5 を使用します。また、インタラプト転送は PIPE6,7 を使用します。お客様固有のクラスドライバ、アプリケーションを作成する場合は、*r_usb_vendor_hapl.c* ファイル、*r_usb_vendor_hdriver.c* ファイルを参考に作成してください。USB ホストとして USB ペリフェラルデバイスと通信するためには以下の設定が必要になります。

1. スケジューラの設定（タスク数やテーブルサイズ、タスク ID、メールボックス ID など）
2. メインループ内でのアプリケーションタスクの呼び出し
3. 実装するデバイスクラスドライバに対応したディスクリプタ解析処理
4. 実装するデバイスクラスドライバに対応したパイプ情報テーブルの作成
5. 実装するデバイスクラスドライバに対応した USB リクエストの転送

7.2.1 機能

1. サンプルアプリケーション

USB ステート遷移をベンダクラスドライバに通知します。また、ベンダクラスドライバを制御します。このとき USB_STS_CONFIGURED を検出するとグローバル変数を初期化してベンダクラスドライバにデータ転送開始を要求します。データ転送は PIPE4,5 を使用したバルク転送と PIPE6,7 を使用したインタラプト転送です。ベンダクラスドライバからデータ転送終了が通知されると、通知をうけたパイプでデータ転送を再開します。また定常処理でキー入力の通知を受けます。サスペンド、レジューム（サスペンド時）、ポート禁止のサンプルを実装しています

2. ベンダクラスドライバ

APL から通知される USB ステートに従ってグローバル変数を初期化します。またアプリケーションからデータ転送が要求されるとUSB-BASIC-F/Wにデータ転送を要求します。USB-BASIC-F/Wからデータ転送終了が通知されるとアプリケーションにデータ転送終了を通知します。ベンダクラスリクエストはサポートしていません。

3. エニュメレーション

USB-BASIC-F/Wは USB デバイスの接続を検出すると自動的にエニュメレーションを行います。動作可能な USB デバイスが接続されていればエニュメレーションが正常終了し、コールバック関数でアプリケーションに USB_STS_CONFIGURED が通知されます。

4. データ転送

エニュメレーションが正常終了した場合はデータ転送が可能です。アプリケーションは USB ステート遷移のコールバックでデータ転送を開始します。USB-BASIC-F/Wを USB ペリフェラルデバイスとして動作させたデバイスと通信できます。

5. ベンダクラスリクエスト

ベンダクラスリクエストは発行しません。

6. USB ステート遷移

USB ステート遷移の通知によって以下のように動作します。

USB_STS_DETACH:	Stop the data transfer, Initialized application sequence mode
USB_STS_DEFAULT:	No processing
USB_STS_ADDRESS:	No processing
USB_STS_CONFIGURED:	Pipe registration, Start the data transfer
USB_STS_SUSPEND:	No processing
USB_STS_RESUME:	No processing

USB_STS_WAKEUP: The same as the resume processing

サスペンド状態からリモートウェイクアップ信号による復帰が可能です。またアプリケーションからサスペンド、レジュームをUSB-BASIC-F/Wに要求することも可能です。

7. ドライバ確認コールバック

USB-BASIC-F/Wはエニュメレーションのシーケンス処理で Configuration ディスクリプタを取得すると、USB-BASIC-F/Wに登録されているドライバ確認コールバック関数（**g_usb_hstd_Driver.classcheck*）を実行します。アプリケーションは *R_usb_hvndr_ClassCheck()*関数でベンダクラスドライバに接続デバイスの動作可否を確認します。ベンダクラスドライバが動作可否を判定する項目は以下のとおりです。

1) ベンダドライバに対応する VID と PID が通知されたか
2) プロダクト ID のストリングディスクリプタがあるか
3) インタフェース内に Bulk パイプが 2 本、Interrupt パイプが 2 本あるか

すべての条件が満たされていればUSB-BASIC-F/Wに API 関数 *R_usb_hstd_ReturnEnuMGR()*で USB_YES を応答します。

7.2.2 ホストサンプルプログラムの動作

1. 初期設定
   - HEW/e^2 studio の場合

   デバイスを H/W リセットすると、ncrt0.a30/resetprg.c 内の関数*_PowerON_Reset_PC* 関数が呼び出されます。
   リセット関数は MCU の初期化を行い、ハードウェア初期化関数 *usb_cpu_mcu_initialize()*を呼び出します。ハードウェア初期化関数から戻ると、メモリ領域の初期化を行い、*main.c* ファイル内の *main()* 関数が呼び出されます。スタートアップ処理の詳細は HM、および統合開発環境のマニュアルを参照してください。

   - CS+の場合

   デバイスを H/W リセットすると、CS+で作成されたスタートアップファイルの *_@cstart* 関数 */PowerON_Reset_PC* 関数が呼び出されます。スタートアップ関数は MCU の初期化を行い、ユーザ定義ハードウェア初期化関数 *hdwinit()/usb_cpu_mcu_initialize()* 関数を呼び出します。ハードウェア初期化関数で MCU 動作クロックの設定を行います。ハードウェア初期化関数から戻ると、メモリ領域を初期化し、*main.c* ファイル内の *main()* 関数が呼び出されます。スタートアップ処理の詳細は HM、および統合開発環境のマニュアルを参照してください。

2. main 関数処理

   main()関数では usb_hsmpl_main_init()関数でシステムの初期化（ターゲット MCU およびボードの初期化、USB モジュールの初期設定、USB-BASIC-F/Wの起動、UPL ドライバの登録、USB モジュールの動作許可）を行いメインループで要求の発生を待つ定常状態となります。

   メインループの動作は以下のようになります。

   ①スケジューラで要求判別する。

   ②処理要求があった場合、タスクに制御を渡す（タスクが実行可能される）。

   ③定常的な処理を行う。

   ④①に戻る。

3. サンプルアプリケーションタスク(*usb_hsmpl_apl_task()*)

   サンプルアプリケーションは、エニュメレーションが正常終了するとグローバル変数を初期化し、ベンダクラスドライバに対して API 関数 *R_usb_hvndr_TransferStart()*でデータ転送開始を要求します。また、ベンダクラスドライバから転送終了コールバックを受けると、API 関数 *R_usb_hvndr_TransferStart()*でデータ転送を繰り返します。

4. ベンダクラスドライバ(*R_usb_hsmpl_VendorTask()*)

ベンダクラスドライバ（HDCD）は、サンプルアプリケーションからデータ転送要求が通知されると API 関数 *R_usb_hstd_TransferStart()*でUSB-BASIC-F/Wにデータ転送を要求します。また、USB-BASIC-F/Wから転送終了コールバックを受けると、コールバック関数でアプリケーションにデータ転送終了を通知します。

ベンダクラスドライバは、サンプルアプリケーションから USB ステート遷移が通知されても特別処理は行いません。アプリケーションが USB ステートを判定してベンダクラスドライバの開始/終了やパイプ情報のレジスタ設定、データ転送開始などを行っています。

USB-BASIC-F/Wの概略フローをFigure 7-3に示します。

USB-BASIC-F/Wは USB データ送受信用の制御機能を実行するタスクを含んでいます。H/W の割り込みが発生すると USB-BASIC-F/W へメッセージとして通知されます。USB-BASIC-F/W は USB 割り込みハンドラからメッセージを受け取ると、割り込み要因を判別して適切な処理を実行します。

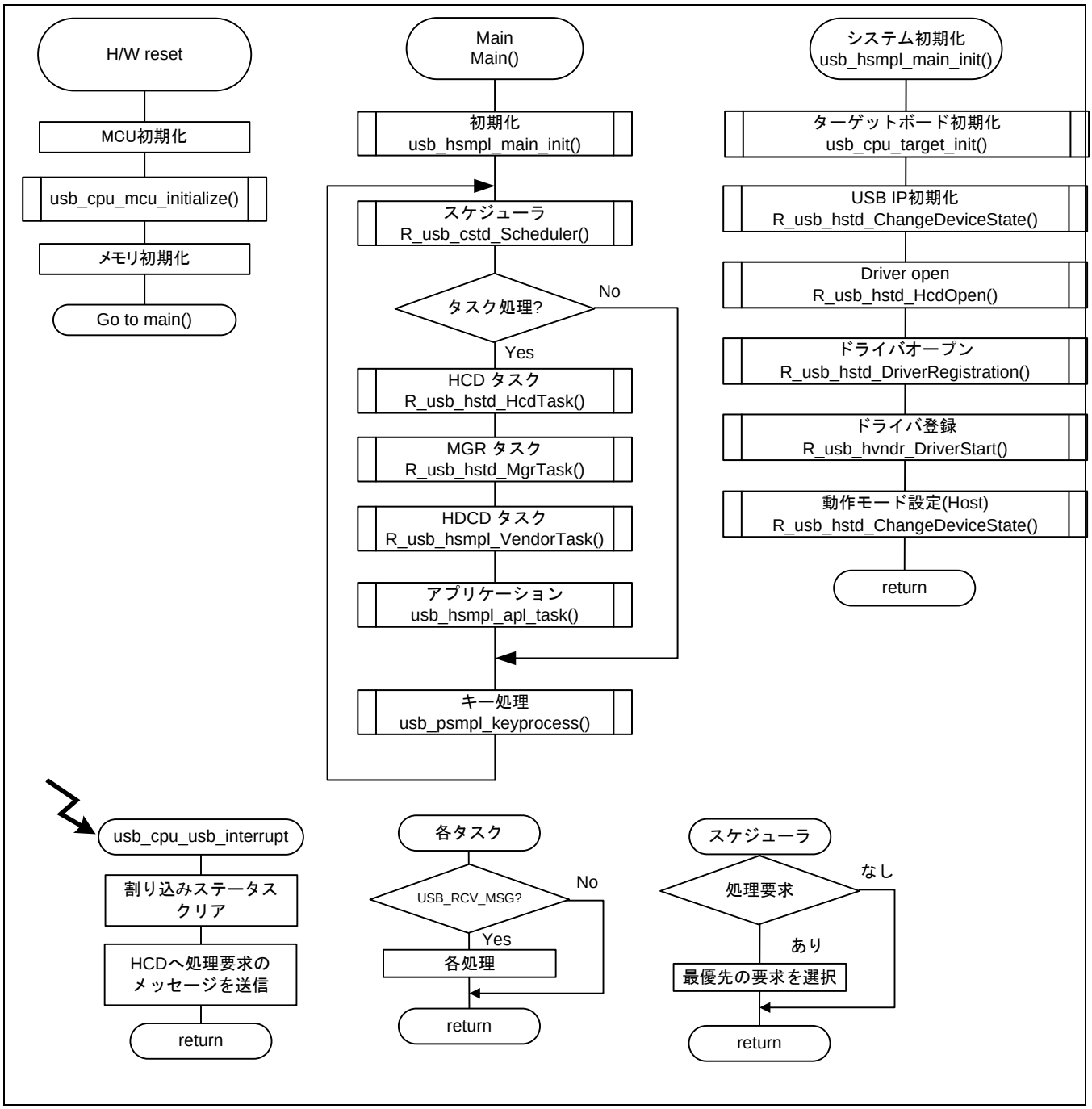

Figure 7-3 概略フロー

7.2.3 スケジューラの設定

タスク ID の最大値、タスク優先テーブルに格納するメッセージの最大値を *r_usb_cstd_kernelid.h* ファイルで設定します。

```
/* Please set with user system */
#define     USB_IDMAX           ((uint8_t)5)    /* Maximum Task ID +1 */
#define     USB_TABLEMAX        ((uint8_t)10)   /* Maximum priority table */
#define     USB_BLKMAX          ((uint8_t)5)    /* Maximum block */
```

7.2.4 タスク ID、メールボックス ID の設定

使用するタスク ID とメールボックス ID を *r_usb_cstd_kernelid.h* ファイルで設定します。なお、タスク優先順位はタスク ID と同じです。（タスク ID 番号の小さい方が、優先度が高い）

```
#define     USB_HCD_TSK         USB_TID_0       /* Host Control Driver Task */
#define     USB_HCD_MBX         USB_HCD_TSK     /* Mailbox ID */
#define     USB_MGR_TSK         USB_TID_1       /* Host Manager Task */
#define     USB_MGR_MBX         USB_MGR_TSK     /* Mailbox ID */
#define     USB_HVEN_TSK        USB_TID_2       /* Task ID */
#define     USB_HVEN_MBX        USB_HVEN_TSK    /* Mailbox ID */
#define     USB_HSMP_TSK        USB_TID_3       /* Host Sample Task */
#define     USB_HSMP_MBX        USB_HSMP_TSK    /* Mailbox ID */
```

7.2.5 タスクの呼び出し

使用するタスクをメインループ（main ()関数）で呼び出します。

```
void main (void)
{
 usb_hsmpl_main_init();

 /* Sample main loop  */
 while( 1 )
 {
  if( R_usb_cstd_Scheduler() == USB_FLGSET )
  {
   R_usb_hstd_HcdTask();    /* HCD Task */
   R_usb_hstd_MgrTask();    /* MGR Task */
   R_usb_hsmpl_VendorTask();
   usb_hsmpl_apl_task();
  }
 }
}
```

7.2.6 UPL の起動

USB-BASIC-F/Wはデバイスと構成が確立した場合（SET_CONFIGURATION リクエスト発行）に、動作可能な UPL に対してコールバック関数（**g_usb_HcdDriver.statediagram*）でデバイス接続を通知します。UPL は第 2 引数の USB ステートを解析してシステムに適合した処理を行ってください。サンプルアプリケーションはデータ領域を初期化し、パイプコンフィギュレーションレジスタを使用可能な状態にしてデータ転送を開始します。

7.2.7 アプリケーションの概要

USB-BASIC-F/W はコンフィガード後に以下の手順でデータ転送を開始します。コールバック関数 *usb_hsmpl_device_state()*で、USB ステートを識別しパイプ設定を行い、ベンダクラスドライバにデータ転送を要求します。

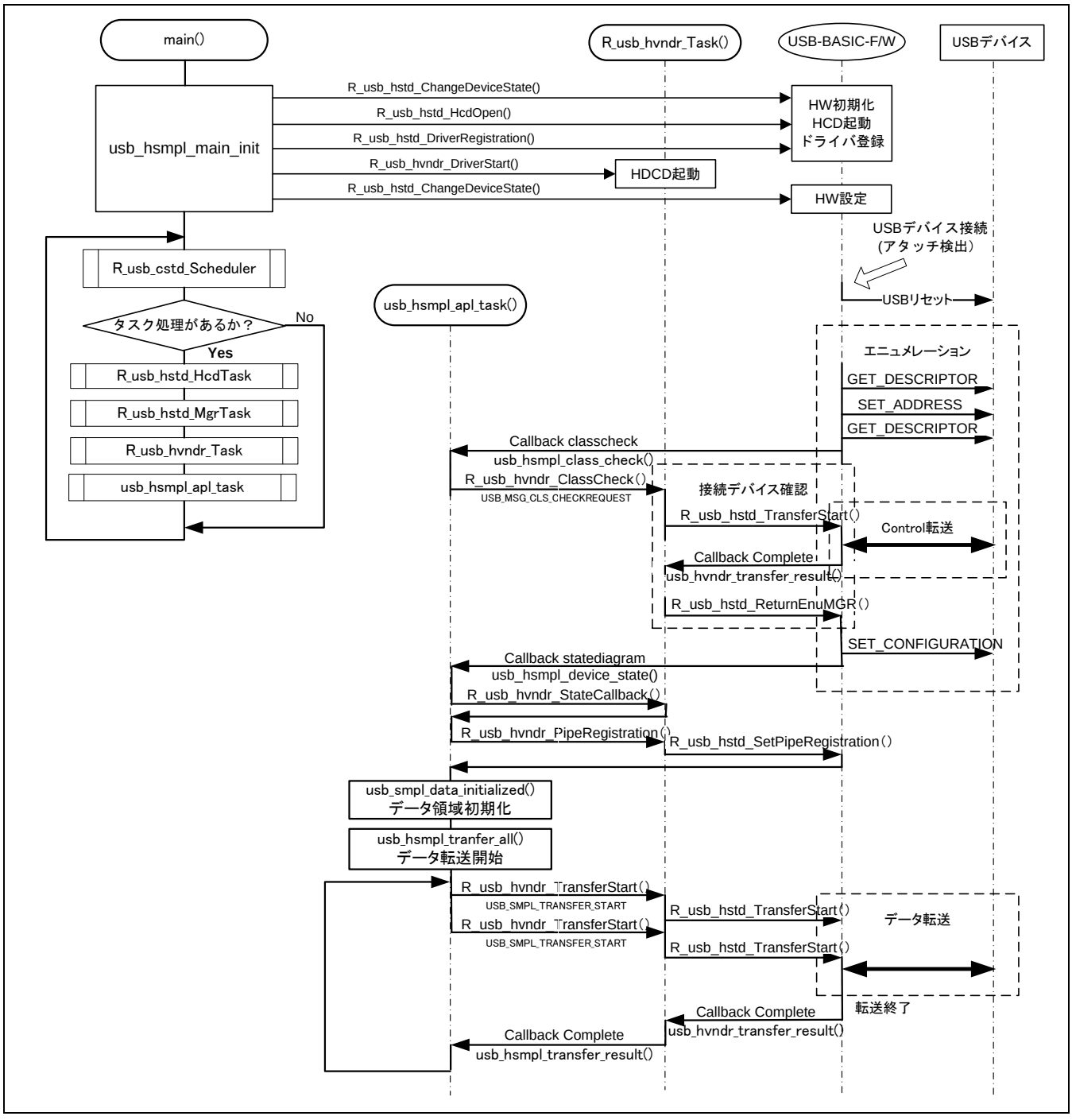

Figure 7-4 アプリケーション動作概要

7.3 データ転送及びコントロール転送

データ転送はお客様固有の機能仕様であり、転送方法、通信開始／終了タイミング、およびバッファ構成などは、システムにあわせて変更していただく必要があります。

7.3.1 基本仕様

Table 8-4の *usb_utr_t* 構造体で指定されたユーザバッファでデータ転送が可能です。USB-BASIC-F/Wはデータ転送が終了すると *PID = NAK* 設定にし、コールバックで転送終了を通知します。

USB-BASIC-F/W は、データ転送要求に対してパイプステータス（データトグル）を転送要求時に指定されたパイプステータス(*utr_table.pipectr*) に更新します。また転送終了のコールバックでパイプステータスを通知します。このため、UPL がパイプステータスを記憶することにより、1 つのパイプで同時にデータ転送が行われない（排他的に制御が可能な）複数のエンドポイントのデータ転送が可能です。ただしパイプステータスは USB リセット、STALL 解除、SET_CONFIGURATION リクエスト、SET_INTERFACE リクエストなどの要因で"DATA0"に初期化する必要があります。

なお、Bulk パイプのマックスパケットサイズは 64byte 固定の設定です。USB リセット発行直後のコントロール転送ではデフォルトパイプのマックスパケットサイズエラー判定は行いません。

7.3.2 データ転送要求

*R_usb_hstd_TransferStart()*を使用してデータ転送を開始してください。

7.3.3 コントロール転送要求

R_usb_hstd_TransferStart() を使用してコントロール転送を開始します。セットアップパケットの仕様についてはTable 8-3を参照してください。セットアップパケットにエラーがある場合は、コントロール転送を行いません。

7.3.4 転送結果の通知

usb_utr_t 構造体で指定されたコールバック関数で、データ転送の終了を UPL に通知します。通知される内容はTable 8-8を参照してください。

7.3.5 データ受信時の注意事項

① 受信パイプはトランザクションカウンタを使用します。

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 usb_utr_t 構造体の tranlen に格納して転送を終了します。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を終了します。ユーザバッファ領域が転送するサイズの容量分を確保できない場合は、usb_cstd_forced_termination()関数で受信パケットをクリアしてしまうことがあります。

② 受信コールバック

受信したデータがマックスパケットサイズの n 倍、かつ受信予定サイズに満たない場合は、データが転送途中であるか転送終了したかの判断ができないためコールバックが発生しません。

ショートパケット受信、または指定されたサイズのデータ受信をした場合に、転送終了と判断しコールバックします。

例、受信データサイズが 128byte でマックスパケットサイズが 64byte の場合

1～63byte 受信	受信コールバック発生する。
64byte 受信	受信コールバック発生しない。
65～128byte 受信	受信コールバック発生する。

7.3.6 データ転送概要

usb_uttr_t 構造体に必要な情報を設定し、*R_usb_hstd_TransferStart()*を呼び出してください。

コントロール転送例とデータ転送例を以下に示します。

・データ転送例

```
void usb_hsmpl_transfer_start( uint16_t pipe )
{
 if( g_usb_SmplTrnCnt[pipe] != 0 )
 {
  g_usb_SmplTrnMsg[pipe].keyword     = pipe;      /* Data area address */
  g_usb_SmplTrnMsg[pipe].tranadr     = g_usb_SmplTrnPtr[pipe];
  g_usb_SmplTrnMsg[pipe].tranlen     = g_usb_SmplTrnSize[pipe];
  g_usb_SmplTrnMsg[pipe].setup       = (uint16_t*)USB_NULL;
  g_usb_SmplTrnMsg[pipe].complete    = (usb_cb_t)&usb_hsmpl_transfer_result;
  R_usb_hstd_TransferStart((usb_utr_t*)&g_usb_SmplTrnMsg[pipe]);
 }
}
```

・コントロール転送例

```
usb_er_t usb_hstd_set_configuration(void)
{
 g_usb_MgrRequest.WORD.BYTE.bmRequestType  =
      USB_REQUEST_TYPE(USB_HOST_TO_DEV,USB_STANDARD,USB_DEVICE);
 g_usb_MgrRequest.WORD.BYTE.bRequest    = USB_SET_CONFIGURATION;
 g_usb_MgrRequest.wValue                =
     (uint16_t)(g_usb_MgrConfDescr[USB_CON_CONFIG_VAL]);
 g_usb_MgrRequest.wIndex                = 0x0000;
 g_usb_MgrRequest.wLength               = 0x0000;
 g_usb_MgrRequest.Address               = (uint16_t)g_usb_MgrDevAddr;

 g_usb_MgrControlMessage.tranadr    = (void*)data_table;
 g_usb_MgrControlMessage.complete   = (usb_cb_t)&usb_hstd_transfer_result;
 g_usb_MgrControlMessage.tranlen    = (usb_leng_t)g_usb_MgrRequest.wLength;
 g_usb_MgrControlMessage.pipenum    = USB_PIPE0;
 g_usb_MgrControlMessage.setup      = (void*)&g_usb_MgrRequest;
 R_usb_hstd_TransferStart(&g_usb_MgrControlMessage);
}
```

コールバック関数（UPL タスクに転送終了をメッセージで通知）例を以下に示します。

```
void usb_hsmpl_transfer_result(usb_utr_t *mess)
{
 mess->msginfo = USB_MSG_CLS_TASK;          /* Data transfer Callback */
 USB_SND_MSG(USB_HSMP_MBX, (usb_msg_t*)mess);
}
void usb_hstd_transfer_result(usb_utr_t *mess)
{
 g_usb_MgrSequence++;
 utrmsg->msginfo  = USB_MGR_CONTINUE;         /* Enumeration */
 USB_SND_MSG(USB_MGR_MBX, (usb_msg_t*)mess);
}
```

7.4 パイプ情報

クラスドライバに適合したパイプ設定を、パイプ情報テーブルとして保持する必要があります。

ベンダクラスドライバは、デバイスのパイプ情報を保持するサンプルテーブルを *r_usb_vendor_hdriver.c* ファイルの *uint16_t g_usb_hvndr_DefEpTbl[]*に記載しています。

7.4.1 パイプ情報テーブル

パイプ情報テーブルは、以下 4 項目で構成されます。

1. パイプウインドウ選択レジスタ（0x64 番地）
2. パイプコンフィグレーションレジスタ（0x68 番地）
3. パイプマックスパケットサイズレジスタ（0x6C 番地）
4. パイプインターバルレジスタ（0x6E 番地）

7.4.2 パイプ定義

ベンダクラスドライバで使用しているパイプ情報テーブルの構成を以下に示します。パイプ情報テーブルのパイプ定義項目で設定可能な値は、*r_usb_hvendor_driver.h* でマクロ定義しています。

Structure example of pipe information table:

```
uint16_t g_usb_hvnr_DefEpTbl[] =                                  ← Pipe information table
{
    USB_PIPE4,                                                    ← Pipe definition item 1
    USB_NULL|USB_BFREOFF|USB_DBLBOFF|USB_SHTNAKOFF,               ← Pipe definition item 2
    USB_NULL,                                                     ← Pipe definition item 3
    USB_NULL,                                                     ← Pipe definition item 4
            :
    USB_PDTBLEND,                                                 ← Pipe information table end definition
}
```

(1)．パイプ定義項目 1：パイプウインドウ選択レジスタに設定する値を指定します。

・パイプ選択　　：選択パイプ（USB_PIPE4～USB_PIPE7）を指定してください。

(2)．パイプ定義項目 2：パイプコンフィグレーションレジスタの設定値を指定します。

転送タイプ	：	USB_BULK/USB_INT のどちらかを指定してください。
BRDY 割り込み動作指定	：	USB_BFREOFF を指定してください。
ダブルバッファモード	：	USB_DBLBON/USB_DBLBOFF のどちらかを指定してください。
SHTNAK 動作指定	：	USB_SHTNAKON/USB_SHTNAKOFF のどちらかを指定してください。
転送方向	：	USB_DIR_H_OUT、USB_DIR_H_IN のどちらかを指定してください。
エンドポイント番号	：	パイプに対するエンドポイント番号(EP1～EP15)を指定してください。

・転送タイプは選択したパイプにより、設定可能な値が異なります。詳細は HM を参照してください。

・接続デバイスのエンドポイントディスクリプタにあわせてパイプ情報を記載してください。

・受信方向パイプ（USB_DIR_H_IN）の場合は USB_SHTNAKON 設定としてください。

.

(3)．パイプ定義項目 3：エンドポイントのデバイスアドレスとマックスパケットサイズを指定します。

- デバイスアドレス：*USB_ADDR2DEVSEL* マクロを使用してデバイスアドレスを設定してください。
- マックスパケットサイズ指定：USB 仕様に準拠した値を設定してください。

(4)．パイプ定義項目 4：エンドポイントのインターバル時間を指定します。
・インターバル時間指定：HM に従って値を設定してください。

(5)．その他制限事項
・同時に通信が可能な Endpoint の数だけパイプ情報が必要です。
・UPL でトランスファー単位の通信同期を取ってください。
・UPL で使用しているパイプ情報を管理してください。
・テーブルの最後には、必ず USB_PDTBLEND を書いてください。
・USB-BASIC-F/Wはデバイスステート遷移をコールバック関数で通知しますので UPL 側で API 関数を使用してパイプ情報のレジスタ設定（解除）処理を実装してください。

エンドポイントディスクリプタから転送タイプ、転送方向、エンドポイント番号、マックスパケットサイズ、インターバル時間を設定するAPI関数*R_usb_hstd_ChkPipeInfo()*を使用する場合は、それぞれのフィールドで"USB_NULL"を指定してください。

7.5 USB-BASIC-F/W をホスト機能で動作させるには

本章では、USB-BASIC-F/Wとサンプルコードを例に、USB-BASIC-F/Wをホスト機能で動作させるための手順を示します。

7.5.1 デバイス選択

Table 7-1に、USB-BASIC-F/Wが提供する各デバイス用の統合開発環境と USB モジュールの機能概要、ハードウェアリソースを示します。

Table 7-1 サンプルコードのハードウェアリソース

デバイス	統合開発環境	ホスト	データレート	ハードウェアリソースフォルダ
R8C/3MK, R8C/34K	HEW	1PortHost	FullSpeed	R8C\ HwResource
RL78/G1C	CS+	1PortHost	FullSpeed LowSpeed	RL78G1C\HwResource*1
		2PortHost	FullSpeed LowSpeed	RL78G1C\HwResource

注意)

RSKRL78G1C の USB ホスト用コネクタは *USB-PORT1* 側のみです。USB-BASIC-F/Wは、*USB-PORT1* 側のみの 1 ポートホストをサポートしません。つまり、実行ファイルは、2 ポートホスト (USB_PORTSEL_PP=USB_2PORT_PP)として作成し、*USB-PORT1* 側を 1 ポートホストとして動作させます。

7.5.2 ユーザシステム定義情報ファイル (*r_usb_usrconfig.h*)

USB-BASIC-F/Wは、*inc* フォルダ内のユーザ定義情報ファイル (*r_usb_usrconfig.h*) を書き換えることにより、USB-BASIC-F/Wの機能設定を行います。システムに合わせて、下記項目を変更してください。i

以下に、ユーザ定義情報ファイルで設定可能な項目とその概要を示します。

(1). 動作モードの指定

USB モジュールの動作モードを設定します。

#define USB_FUNCSEL_PP USB_HOST_PP ：USB をホストで動作

(2). USB ポート指定(RL78/USB のみ)

使用する USB ポートの数を設定します。

#define USB_PORTSEL_PP USB_1PORT_PP ：USB ポートを 1 つ使用

#define USB_PORTSEL_PP USB_2PORT_PP ：USB ポートを 2 つ使用

(3). グローバル変数を static 宣言する機能の指定

グローバル変数の static 宣言する機能を使用する場合は下記を追記します。
#define USB_STATIC_USE

(4). エラー時に while(1)する機能の指定

エラー時に while(1)する機能を使用する場合は下記を追記します。
#define USB_DEBUG_HOOK_USE

(5). Battery Charging 機能の指定 (RL78/USB のみ)

Battery Charging 機能を使用する場合は下記を追記します。

#define USB_HOST_BC_ENABLE

Dedicated Charging Port で動作させる場合は下記も追記します。

#define USB_BC_DCP_ENABLE

(6). コントロールリードデータバッファサイズ

コントロールリード転送で受信するデータバッファサイズを指定してください。

例）デバイスディスクリプタ 20Byte、コンフィグレーションディスクリプタ 256Byte の場合

```
#define  USB_DEVICESIZE  20u

#define  USB_CONFIGSIZE  256u
```

(7). デバイスアドレス

PORT0 に接続されるデバイスのデバイスアドレスを指定してください。

例）デバイスアドレスを 2 から開始する場合

```
#define  USB_DEVICEADDR   2u
```

1～5 の範囲でアドレス指定が可能です。

USB-PORT1 側を使用する場合は、1～4 の範囲でアドレス指定してください。

(8). デバウンス間隔

デバイスアタッチ後のデバウンス間隔を指定してください。

例）スケジューラが 3000 回経過されるまでを設定する場合

```
#define    USB_TATTDB              3000
```

なお、USB-BASIC-F/Wの動作モード指定と USB ポートはヘッダファイルで指定せずに、統合化環境（HEW のビルドコンフィギュレーション、および CS+のビルドモード）のプロジェクトファイルで行っています。

なお、以下の定義は、開発環境のプロジェクトファイルで行っています。

RL78G1C : USB_FUNCSEL_PP = USB_HOST_PP
RL78USB
R8C : USB_FUNCSEL_PP = USB_HOST_PP
R8CUSB

7.5.3 USB-BASIC-F/Wの変更

USB-BASIC-F/Wを動作させるには、以下のプログラムおよびヘッダファイルの変更が必要です。

- Renesas USB MCU用のサンプル関数を提供しますがユーザシステムにあわせて変更してください。
- 制御用 MCU の初期化、割り込みハンドラ、割り込み制御など(Table 7-2参照)
- 指定時間待ち関数の時間調整 (*usb_cpu_delay_xms()* 関数, *usb_cpu_delay_1u()* 関数)
  ループ処理等で指定時間のウェイトを行っています。ご使用のシステムに合わせてループ回数を変更するなど指定時間になるように調整してください。
- スケジューラ機能で使用するための USB 関連の割り込みを禁止および許可する関数の設定
  (*usb_cpu_int_disable()* 関数, *usb_cpu_int_enable()* 関数)
  USB-BASIC-F/Wでは、USB 割り込み禁止関数（*usb_cpu_int_disable()*関数）が USB 割り込みを禁止し、USB 割り込み許可関数（*usb_cpu_int_disable()*関数）が USB 割り込みを許可しています。ご使用の MCU に合わせて設定を行ってください。

Table 7-2 関数一覧

型	関数名	機能概要	備考
void	usb_cpu_mcu_initialize(void)	MCU 初期処理（発信制御など）	
void	usb_cpu_target_init(void)	システム初期化	
void	usb_cpu_set_pin_function(uint16_t function)	MCU の USB 機能設定（端子設定など）	
void	usb_cpu_usb_interrupt (void)	USB 割り込みハンドラ	
void	usb_cpu_usbint_init (void)	USB 割り込み許可	
void	usb_cpu_int_enable(void)	スケジューラ用 USB 割込み許可	
void	usb_cpu_int_disable(void)	スケジューラ用 USB 割込み禁止	
void	usb_cpu_intp0_enable(void)	RSK 上スイッチ用 INTP0 割り込み許可	
void	usb_cpu_intp0(void)	RSK 上スイッチ用 INTP0 割り込み	
void	usb_cpu_delay_1us(uint16_t time)	1us 待ち処理	
void	usb_cpu_delay_xms(uint16_t time)	1ms 待ち処理	
void	usb_cpu_stop_mode(void)	ストップ命令実行	

8. ホストコントロールドライバ (HCD)

8.1 基本機能

HCD は、対象デバイスを USB ホストとして動作させる際の H/W 制御用のプログラムです。USB-BASIC-F/Wは UPL の要求を解析し H/W の制御を行います。H/W 制御結果は API 関数の戻り値もしくはコールバック関数で UPL に通知します。また、H/W からの要求もUSB-BASIC-F/Wに登録されたドライバ情報のコールバック関数で UPL に通知します。USB-BASIC-F/Wを USB ホストデバイスとして機能させるにはUSB-BASIC-F/Wを起動(8.2.1章参照)して、UPL の登録(8.2.2章参照)を行ってください。

USB-BASIC-F/Wの機能は以下のとおりです。

1. 接続デバイスの USB ステート遷移検出と変化結果通知: [8.2.3]
2. 接続デバイスとのエニュメレーション: [8.2.8]
3. 接続デバイスの動作可否判定: [8.2.4]
4. データ転送と転送結果通知: Chapter 8.2.5
5. USB ステート管理（USB ステート制御と制御結果通知）: [8.2.7]
6. USB Battery Charging Specification Revision 1.2 で定義された Charging Downstream Port または Dedicated Charging Port :[8.2.9]

8.2 操作概要

8.2.1 HCD 起動

UPL は API 関数 *R_usb_hstd_HcdOpen()*でUSB-BASIC-F/Wを起動してください。

8.2.2 UPL の登録

UPL はTable 8-1 の情報を API 関数 *R_usb_hstd_DriverRegistration()*でUSB-BASIC-F/Wに登録します。

USB-BASIC-F/Wはグローバル変数(*g_usb_HcdDriver[]*)に情報を保存します。

```
typedef struct
{
 usb_port_t        rootport;       /* Root port */
 usb_addr_t        devaddr;        /* Device address */
 uint16_t          devstate;       /* Device state */
 uint16_t          ifclass;        /* Interface Class */
 usb_cb_check_t    classcheck;     /* Driver check */
 usb_cb_info_t     statediagram;   /* Device status */
} usb_hcdreg_t;
```

Table 8-1 usb_hcdreg_t 構造体のメンバ

変数名	機能	備考
rootport	USB-BASIC-F/Wが使用します。接続されているポート番号が登録されます。	
devaddr	USB-BASIC-F/Wが使用します。デバイスアドレスが登録されます。	
devstate	USB-BASIC-F/Wが使用します。デバイスの接続状態が登録されます。	
ifclass	UPL が動作するインタフェースクラスコードを登録してください。	
classcheck	エニュメレーション時に接続デバイスの動作可否をチェックする関数を登録してください。	
statediagram	USB ステート遷移時に起動する関数を登録してください。	

8.2.3 USB ステート遷移通知

USB-BASIC-F/Wは USB ステート遷移などを UPL に通知する為に、USB-BASIC-F/Wに登録されている UPL に対して USB ステート遷移コールバック関数(**g_usb_PcdDriver.statediagram*) を実行します。USB-BASIC-F/Wはコールバック関数の第 2 引数で以下の情報を UPL に通知します。UPL は USB ステートを解析してシステムに適合した処理を行ってください。

USB ステート遷移

USB_STS_DETACH:	デタッチ検出
USB_STS_ATTACH	アタッチ検出
USB_STS_DEFAULT	デフォルトステート遷移（USB バスリセット検出）
USB_STS_OVERCURRENT:	過電流検出
USB_STS_CONFIGURED:	コンフィガードステート遷移（Set_Configuration リクエスト送信）
USB_STS_WAKEUP:	コンフィガードステート遷移（リモートウェイクアップ処理終了）
USB_STS_POWER:	ポート許可（API 関数で要求）
USB_STS_PORTOFF:	ポート禁止（API 関数で要求）
USB_STS_SUSPEND:	サスペンド（API 関数で要求）
USB_STS_RESUME:	レジューム（API 関数で要求）
USB_STALL_SUCCESS:	ペリフェラルデバイスのSTALL解除（API関数で要求）

8.2.4 接続デバイスの動作可否判定

USB-BASIC-F/W はアタッチを検出するとエニュメレーション(8.2.8章参照) を行います。エニュメレーションのシーケンス処理で Configuration ディスクリプタを取得すると、USB-BASIC-F/Wに登録されているドライバ確認コールバック関数(**g_usb_hstd_Driver.classcheck*) を実行します。USB-BASIC-F/Wは UPL にコールバック関数の第 1 引数で Table 8-2の情報を通知します。UPL は受け取ったデバイス情報を解析し、必要な処理を行ってください。また、Table 8-2で示した内容以外の情報が必要であれば API 関数 *R_usb_hstd_DeviceInformation()*などで取得してください。UPL は接続デバイスの識別ができたら動作可否（USB_YES/USB_NO）を API 関数 *R_usb_hstd_ReturnEnuMGR()*でUSB-BASIC-F/Wに応答してください。USB-BASIC-F/Wは USB_YES が通知されればエニュメレーションを継続してデバイスを構成状態まで遷移させます。また USB_NO が通知された場合は他に動作が可能なドライバが登録されているか検索します。

```
table[0]   = (uint16_t*)&g_usb_MgrDeviceDescriptor;
table[1]   = (uint16_t*)&g_usb_MgrConfigurationDescriptor;
table[2]   = (uint16_t*)&g_usb_HcdDeviceAddr;
(*driver->classcheck)((uint16_t**)&table);
```

Table 8-2 classcheck 引数の配列

配列順	機能	備考
table[0]	Address of device descriptor storage area	
table[1]	Address of configuration descriptor storage area	
table[2]	Address of global variable that mean the Device Address	

8.2.5 USB-BASIC-F/Wに対する転送要求発行

UPL がデータ転送を行いたい場合は以下の構造体を引数として R_usb_pstd_TransferStart() を呼び出してください。USB-BASIC-F/W はグローバル変数(g_usb_LibPipe)に引数のアドレス情報を保存します。このため URL はデータ転送が終わるまで引数の実態を保持してください。

```
struct usb_utr_t
{
 usb_strct_t    msginfo;        /* Message Info for F/W */
 usb_strct_t    pipenum;        /* Pipe number */
 usb_strct_t    status;         /* Transfer status */
 usb_strct_t    flag;           /* Flag */
 usb_cb_t       complete;       /* Call Back Function Info */
 void           *tranadr;       /* Transfer data Start address */
 uint16_t       *setup;         /* Setup packet(for control only) */
 uint16_t       pipectr;        /* Pipe control register */
 usb_leng_t     tranlen;        /* Transfer data length */
 uint8_t        dummy;          /* Adjustment of the byte border */
}
```

8.2.6 セットアップパケット

コントロール転送を行う場合、以下の構造体 *usb_utr_t* の**setup* にアドレスをットします。

```
typedef struct
{
 union {
  struct {                      /* Characteristics of request */
    uint8_t  bmRequestType;     /* Characteristics of request */
    uint8_t  bRequest;          /* Specific request */
  } BYTE;
  uint16_t   wRequest;          /* Control transfer request */
 } WORD;
 uint16_t   wValue;             /* Control transfer value */
 uint16_t   wIndex;             /* Control transfer index */
 uint16_t   wLength;            /* Control transfer length */
 uint16_t   Address;
} usb_hcdrequest_t;
```

Table 8-3 usb_hcdrequest_t 構造体のメンバ

メンバ名	機能
bmRequestType	bmRequestType[D7-D0]の値が入ります。 *USB_REQUEST_TYPE* マクロを使用します。
bRequest	bRequest の値が入ります。
wRequest	リクエストの wRequest の値が入ります。(USBREQ レジスタ BREQUEST の値が入ります。) このビットはユニオンタイプの wRequest として参照することができます。
wValue	wValue の値が入ります。(USBVAL レジスタに値が設定されます。)
wIndex	wIndex の値が入ります。(USBINDEX レジスタに値が設定されます。)
wLength	wLength の値が入ります。(USBLENG レジスタに値が設定されます。)
Address	デバイスアドレス

Table 8-4 usb_utr_t 構造体のメンバ

メンバ名	機能	備考
msginfo	USB-BASIC-F/Wが使用するメッセージ情報です。API 関数が設定します。	
pipenum	UPL でパイプ番号を指定してください。	
status	USB-BASIC-F/Wが下記のステータス情報を応答します。USB_CTRL_END： Control 転送が正常に終了した場合 USB_DATA_OK： データ転送（送信/受信）が正常終了した場合 USB_DATA_SHT： 指定されたデータ長未満でデータ受信が正常終了した場合 USB_DATA_OVR： 受信データがサイズオーバーした場合 USB_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 USB_DATA_DTCH： デタッチを検出した場合 USB_DATA_STALL： STALL 応答もしくは Max packet size エラーを検出した場合 USB_DATA_STOP： データ転送を強制終了した場合	
flag	未使用	
complete	UPL で転送終了時に実行したいコールバック関数を指定してください。 型宣言は次のとおりです。 typedef void (*usb_cb_t)(usb_utr_t*);	
*tranadr	UPL で以下の情報を指定してください。 受信または ControlRead：受信データを格納するバッファアドレス 送信または ControlWrite：送信データが格納してあるバッファアドレス。 NoDataControl 転送： 指定されても無視します。 tranlen で指定するデータ長より大きな領域を確保してください。	
*setup	コントロール転送の場合、Table 8-3の構造体のアドレスを指定します。	
pipectr	UPL で PIPExCTR レジスタ情報を指定してください。 当該メンバの bit6 に従い DATA0/DATA1 のシーケンスビット制御を行います。 初期状態は USB_NULL を 2 回目以降はUSB-BASIC-F/Wが応答した値を設定してください。 USB-BASIC-F/Wが PIPECTR レジスタ情報を応答します。	
tranlen	UPL で以下の情報を指定してください。 受信または ControlRead 転送： 受信したいデータ長 送信または ControlWrite 転送： 送信したいデータ長 NoDataControl 転送： "0"を指定してください。 送受信が可能な最大長は 65535 バイトです。 USB-BASIC-F/Wはデータ転送終了後に送受信の残りデータ長を格納します。 コントロール転送時はデータステージに対する残りデータ長を格納します。	

8.2.7 HCD に対する USB ステート遷移要求と通知

UPL が USB ステートを変更したい場合は API 関数 *R_usb_hstd_MgrChangeDeviceState()*を呼び出してください。API 関数の引数で制御内容を指示します。UPL の要求はUSB-BASIC-F/Wにメッセージで通知され、MGR タスクがシーケンス管理を行いながらステート遷移を実現します。接続デバイスの USB ステート遷移が終了すると、コールバック関数で結果を通知します。
USB-BASIC-F/W が管理している情報も API 関数 *R_usb_hstd_DeviceInformation()*で取得することができます。

8.2.8 エニュメレーション

USB-BASIC-F/Wは USB デバイスの接続を検出すると、USB リセットを発行してエニュメレーションを行います。エニュメレーションのシーケンス処理では、以下の標準リクエストを発行します。このときUSB-BASIC-F/Wはポート 0 に接続されたデバイスにユーザマクロで定義した"USB_DEVICEADDR"のアドレス番号を割り振ります。 H/W がポート 1 をサポートしている場合は、ポート 1 に接続されたデバイスに"USB_DEVICEADDR+1"のアドレス番号を割り振ります。ただし、アドレス番号が"0x05"を超えないように"USB_DEVICEADDR"のマクロを定義してください。

(1) GET_DESCRIPTOR (Device ディスクリプタ)

(2) SET_ADDRESS

(3) GET_DESCRIPTOR (Configuration ディスクリプタ)

(4) SET_CONFIGURATION

コンフィギュレーションディスクリプタを取得した後、USB-BASIC-F/Wで登録されたコールバック関数(8.2.4参照)が実行されます。
UPLは、コールバック関数によって接続されているデバイスが対応しているかどうかを確認します。UPLは、API関数*R_usb_hstd_ClassCheckResult()*を使用して結果を(USB_YES/USB_NO)を通知します。
SET_CONFIGURATIONリクエスト発行後、USB-BASICF/Wは、動作中のUPLにコールバック関数によるデバイス接続を通知します。操作可能なクラスドライバが登録されていない場合、USB-BASICF/Wは接続して
いるデバイスへのSET_CONFIGURATIONリクエストを出します。この場合、ステート遷移をUPLに通知しません。

8.2.9 ホストバッテリチャージング制御 (HBC)

HBC は、対象デバイスを USB Battery Charging Specification Revision 1.2 で定義された CDP または DCP 機能を動作させる際の H/W 制御用プログラムです。

USB-BASIC-F/Wの VBUS ドライブ、アタッチ処理、デタッチ処理のタイミングでそれぞれに応じた処理を行います。また、PDDETINT 割り込み発生時に処理を行います。UPL からの制御の必要はなく、UPL への通知も行いません。

CDP と DCP はプログラム排他となります。DCP で動作させる場合は、USB 通信機能は使用できません。

HBC の処理フローをFigure 8.1に示します。

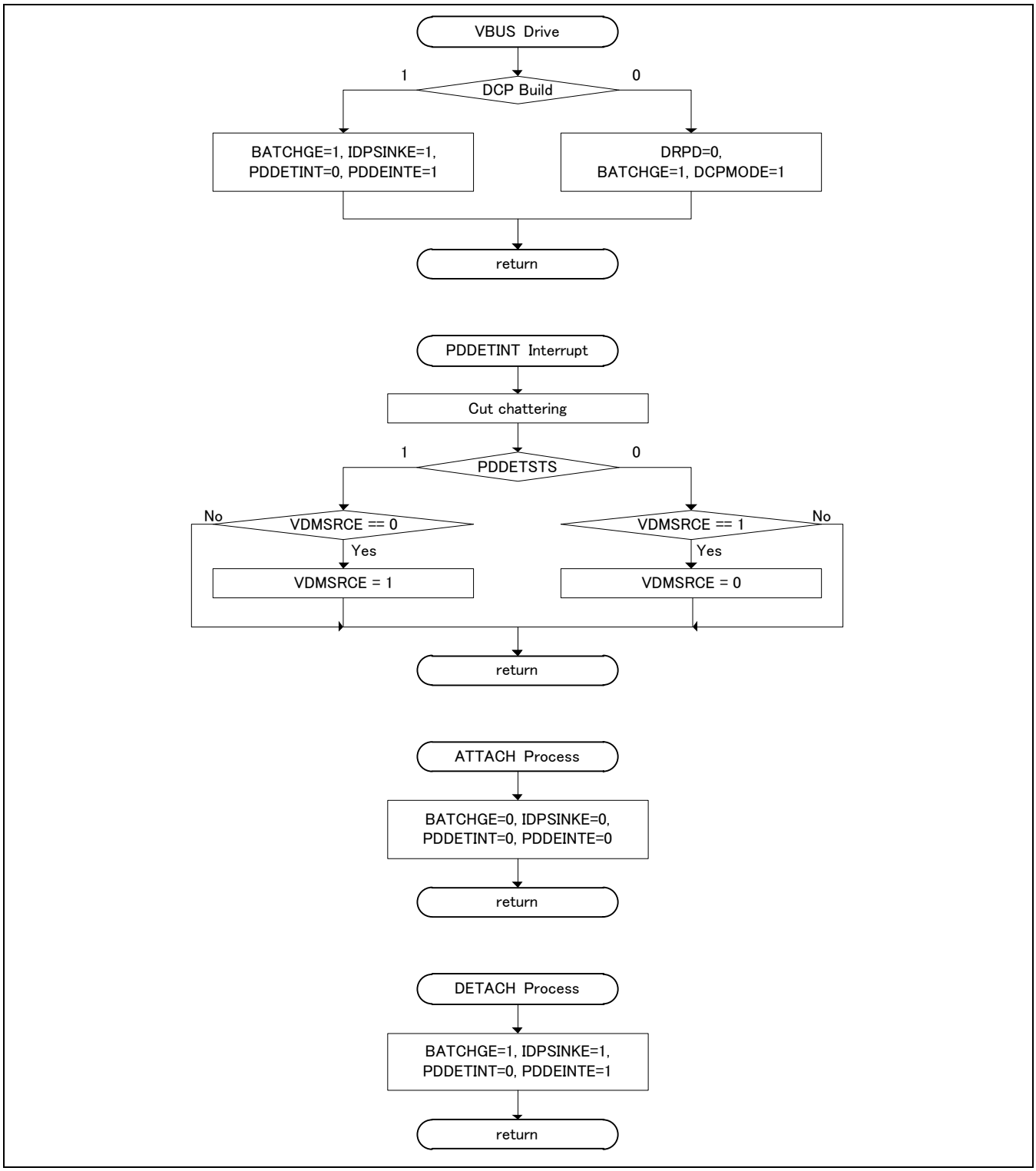

Figure 8.1 HBC フローチャート

8.2.10 USB-BASIC-F/W使用時の注意事項

1. USB-BASIC-F/Wは複数のデバイスに対して、同時進行でエニュメレーションを行うことができません。
2. USB-BASIC-F/Wはマルチコンフィグレーションデバイスに対応していません。
3. USB-BASIC-F/Wはマルチインタフェースデバイスに対応していません。
4. UPL がサスペンドを要求する場合はデータ転送を中断させてからサスペンド要求を発行してください。

5. UPL はレジューム完了やリモートウェイクアップ検出を受けた場合にデータ転送を再開してください。
6. USB-BASIC-F/Wはデタッチ検出時にはデータ転送を停止します。
7. USB-BASIC-F/Wには以下の機能もあります。詳細は8.3を参照してください。
   （1） USB ポートの禁止制御
   （2） USB ステート遷移（サスペンド、レジューム）
   （3） STALL されたパイプのクリア（接続デバイスに対する STALL 解除）
   （4） ディスクリプタから Endpoint 情報の検索
   （5） データ転送の中断
   （6） UPL の開放

8.3 HCD API 関数

UPL は API 関数で H/W の制御要求を行います。API 関数は *r_usb_hdriverapi.c* ファイルにあります。

USB-BASIC-F/Wの API 関数を使用する場合は、Table 8-5に示した順番に従いヘッダファイルをインクルードしてください。また、Table 8-6に HCD API 関数一覧を示します。

Table 8-5 HCD API ヘッダファイル一覧

ファイル名	説明	備考
r_usb_ctypedef.h	変数型定義	
r_usb_ckernelid.h	システムヘッダファイル	
r_usb_cdefusbip.h	USB ドライバ用各種定義	
r_usb_api.h	USB ドライバ API 関数定義	

Table 8-6 HCD API 関数一覧

関数名	説明	備考
R_usb_hstd_HcdTask	HCD タスク	
R_usb_hstd_MgrTask	MGR タスク	
R_usb_hstd_HcdOpen	MGR タスク、HCD タスクを起動（タスク初期化処理）	
R_usb_hstd_DriverRegistration	UPL のドライバ登録	
R_usb_hstd_DriverRelease	UPL のドライバ解放	
R_usb_hstd_TransferStart	データ転送開始要求	
R_usb_hstd_TransferEnd	データ転送強制終了要求	
R_usb_hstd_MgrChangeDeviceState	接続されているデバイスの USB ステート遷移要求	
R_usb_hstd_ChangeDeviceState	接続されているデバイスの状態変更要求	
R_usb_hstd_DeviceInformation	接続されているデバイスの状態取得要求	
R_usb_hstd_ChkPipeInfo	パイプ情報テーブルの設定	
R_usb_hstd_ReturnEnuMGR	エニュメレーション継続要求	
R_usb_hstd_SetPipeRegistration	パイプ情報をレジスタに設定する。	

8.4 HCD コールバック関数

USB-BASIC-F/Wは USB ステート遷移、データ転送終了などをコールバック関数で UPL に通知します。コールバック関数はドライバ登録時、API 関数呼び出し時に UPL が指定します。コールバック関数を新規作成する場合は API 関数使用時と同様に Table 8-5に示した順番に従いヘッダファイルをインクルードしてください。またTable 8-7. に HCD コールバック関数一覧を示します。

Table 8-7 HCD コールバック関数一覧

関数名	説明	備考
*g_usb_HcdDriver[x].classcheck	接続デバイスの動作可否確認のコールバック	
*g_usb_HcdDriver[x].statediagram	USB ステート遷移検出時のコールバック	
* g_usb_LibPipe[pipe]->complete	データ転送起動時のコールバック	
*g_usb_MgrCallback	API 関数による USB ステート遷移終了のコールバック	

8.5 API 関数、コールバック関数詳細

API関数およびコールバック関数の詳細は以下のとおりです。

R_usb_hstd_HcdTask

HCD タスク

形式

void	R_usb_hstd_HcdTask(void)

引数

―	―

戻り値

―	―

解説

*usb_hstd_hcd_task()*を呼び出します。

*usb_hstd_hcd_task()*は USB 割込みに応じた処理を実行します。

- USB コントロール転送を行います。
- コントロール転送終了時はコールバック関数を呼び出します。
- USB ステート遷移検出時は MGR タスクに通知します。

*usb_hstd_hcd_task()*は API 関数を経由して要求されたデータ転送を行います。

- データ転送終了時は API 関数で指定されたコールバック関数を呼び出します。

*usb_hstd_hcd_task()*は MGR タスクから要求された USB ステート制御(H/W 制御)を行います。

- USB ステート制御後はコールバック関数を呼び出します。

補足

1. 本関数は、スケジューラ処理を行うループ内で必ず呼び出してください。
2. 無効なメッセージを受け取った場合はフック関数(R_usb_cstd_debug_hook())をコールします。フック関数については、9.3章を参照してください。

使用例

```
void main(void)
{
 usb_hsmpl_main_init();
 while( 1 )
 {
  if(R_usb_cstd_Scheduler() == USB_FLGSET )
  {
       R_usb_hstd_HcdTask();
       R_usb_hstd_MgrTask();
       usb_hsmpl_apl_task();
  }
 }
}
```

R_usb_hstd_MgrTask

MGR タスク

形式

void	R_usb_hstd_MgrTask(void)

引数

―	―

戻り値

―	―

解説

*usb_hstd_mgr_task()*を呼び出します。

*usb_hstd_mgr_task()*は HCD タスクが検出した USB ステート遷移のシーケンス管理を行います。

- エニュメレーションのシーケンス管理を行います。
- リモートウェイクアップのシーケンス管理を行います。
- デタッチ、過電流検出のシーケンス管理を行います。
- シーケンス処理終了後、USB ステート遷移コールバック関数を呼び出します。

*usb_hstd_mgr_task()*は API 関数を経由して要求された USB ステート遷移のシーケンス管理を行います。

- サスペンド、レジュームのシーケンス管理を行います。
- ポート許可、禁止のシーケンス管理を行います。
- 接続デバイスの STALL 解除を行います。
- シーケンス処理終了後、API 関数で指定されたコールバック関数を呼び出します

補足

1. 本関数は、スケジューラ処理を行うループ内で必ず呼び出してください。

使用例

```
void main(void)
{
 usb_hsmpl_main_init();
 while( 1 )
 {
  if(R_usb_cstd_Scheduler() == USB_FLGSET )
  {
       R_usb_hstd_HcdTask();
       R_usb_hstd_MgrTask();
       usb_hsmpl_apl_task();
  }
 }
}
```

R_usb_hstd_HcdOpen

HCD タスク起動

形式

void	R_usb_hstd_HcdOpen(void)

引数

—	—

戻り値

—	—

解説

HCD で使用するグローバル変数の初期化を行います。

補足

1. 本関数は、初期起動時に呼び出してください。

使用例

```
void usb_hsmpl_main_init(void)
{
 usb_cpu_target_init();                    /* Target board initialize */

 /* USB-IP initialized */
 R_usb_hstd_ChangeDeviceState(USB_DO_INITHWFUNCTION)

 /* HCD driver open & registration */
 R_usb_hstd_HcdOpen();                     /* HCD task, MGR task open */
 usb_hsmpl_driver_registration();          /* Sample driver registration */

 /* USB-IP is set to the host function */
 R_usb_hstd_ChangeDeviceState(USB_DO_SETHWFUNCTION);
}
```

R_usb_hstd_DriverRegistration

ホストデバイスクラスドライバ(HDCD)登録

形式

void	R_usb_hstd_DriverRegistration(usb_hcdreg_t * registinfo)

引数

* registinfo	クラスドライバ構造体

戻り値

―	―

解説

UPL を USB-BASIC-F/Wに登録します。USB-BASIC-F/Wが管理する登録ドライバ数を更新し、新しい領域に UPL 情報の登録を行います。

補足

1. 初期設定時、UPL で本関数を呼び出してください。
2. 登録する情報は、Table 8-1 usb_hcdreg_t 構造体のメンバを参照してください。
3. 代表的なインタフェースクラスコードは *r_usb_cdefusbip.h* ファイルで定義しています。

使用例

```
void usb_hsmpl_driver_registration(void)
{
 usb_hcdreg_t  driver;

 /* Driver registration */
 driver.ifclass    = USB_IFCLS_VEN;         /* Vendor class */
 driver.classcheck= &usb_hsmpl_class_check;
 driver.statediagram = &usb_hsmpl_open_close;
 R_usb_hstd_DriverRegistration(&driver);
}
```

R_usb_hstd_DriverRelease

ホストデバイスクラスドライバ(HDCD)解放

形式

void	R_usb_hstd_DriverRelease(uint8_t devclass)

引数

devclass	デバイスクラス（USB2.0specification 規定のインタフェースクラスコード）

戻り値

―	―

解説

USB-BASIC-F/Wに登録されているデバイスクラスドライバを開放します。USB-BASIC-F/Wが管理する登録ドライバ数を更新し、使用していた領域を空き状態にします。

補足

1. ドライバを開放したい場合に、UPL で本関数を呼び出してください。
2. 開放する情報は、Table 8-1を参照してください。
3. 代表的なインタフェースクラスコードは *r_usb_cdefusbip.h* ファイルで定義しています。
4. この API をコールする前に R_usb_hstd_TransferEnd()を使ってデータ転送を停止してください。

使用例

```
ueb_er_t usb_smp_task(void)
{
 usb_hcdreg_t  driver;

  :
 R_usb_hstd_DriverRegistration(&driver);         /* Driver registration */
  :
 R_usb_hstd_DriverRelease(USB_IFCLS_HID);        /* Release HID class driver */

 /* Driver registration */
 driver.ifclass      = USB_IFCLS_VEN;            /* Vendor class */
 driver.classcheck   = &usb_hsmpl_class_check;
 driver.statediagram = &usb_hsmpl_open_close;
 R_usb_hstd_DriverRegistration(&driver);
  :

}
```

R_usb_hstd_TransferStart

データ転送実行要求

形式

usb_er_t R_usb_hstd_TransferStart(usb_utr_t * utr_table)

引数

* utr_table	データ転送構造体

戻り値

USB_E_OK	成功
USB_E_ERROR	失敗、引数エラー
USB_E_QOVR	オーバラップ　（パイプ使用中）

解説

各パイプのデータ転送要求を行います。

データ転送は、指定データサイズを満たした場合、ショートパケットを受信した場合、およびエラー発生で終了します。

データ転送終了時に、引数の構造体メンバ（*utr_table. complete*）のコールバック関数を呼び出します。このコールバック関数の引数（*utr_table*）には、送受信の残りデータ長、ステータス、及び転送終了の情報が設定されています。

同一パイプでデータ転送を再開する場合は、前回転送終了したパイプステータス（データトグル）と今回通信要求するパイプステータスをあわせる必要があります。引数の構造体メンバ（*utr_table.pipectr*）でパイプステータスを設定してください。なおパイプステータスは USB リセット、STALL 解除などの要因で"*DATA0*"に初期化する必要があります。

データ転送中のパイプに対する転送開始要求が発生した場合は USB_E_QOVR を応答します。

補足

1. データ転送構造体はTable 8-4を参照してください。
2. 受信したデータがマックスパケットサイズの n 倍、かつ受信予定サイズに満たない場合は、データ転送の途中であると判断しコールバックが発生しません。
3. コントロール転送の場合も本 API 関数を使用します。

使用例

```
usb_utr_t   g_usb_HsmplTrnMsg[USB_TBL_MAX];
void usb_hvndr_data_transfer(usb_pipe_t pipe)
{
 /* PIPE Transfer set */
 g_usb_HsmplTrnMsg[pipe].pipenum  = pipe;
 g_usb_HsmplTrnMsg[pipe].tranadr  = g_usb_HsmplTrnPtr[pipe];
 g_usb_HsmplTrnMsg[pipe].tranlen  = g_usb_HsmplTrnSize[pipe];
 g_usb_HsmplTrnMsg[pipe].pipectr  = g_usb_HsmplPipeCtr[pipe];
 g_usb_HsmplTrnMsg[pipe].setup    = 0;
 g_usb_HsmplTrnMsg[pipe].complete= (usb_cb_t)&usb_hvndr_transfer_result;
  R_usb_hstd_TransferStart((usb_utr_t *)&g_usb_HsmplTrnMsg[pipe]);
}
```

R_usb_hstd_TransferEnd

データ転送強制終了要求

形式

usb_er_t R_usb_hstd_TransferEnd(usb_pipe_t pipe, usb_strct_t msginfo)

引数

pipe	パイプ番号
msginfo	通信ステータス

戻り値

USB_E_OK	成功
USB_E_ERROR	失敗
USB_E_QOVR	オーバラップ（転送終了中のパイプに対する転送終了要求）

解説

引数 msginfo に以下の値を設定し、USB-BASIC-F/Wにデータ転送の強制終了を要求します。
- USB_DO_TRANSFER_STP： データ転送強制終了（コールバックします。）
- USB_DO_TRANSFER_TMO： データ転送タイムアウト（コールバックしません。）

msginfo=USB_DO_TRANSFER_STP による強制終了は、データ転送要求時(*R_usb_hstd_TransferStart*)に設定したコールバック関数で転送終了を通知します。コールバック関数の引数（*usb_utr_t*）で、送受信の残りデータ長、パイプコントロールレジスタの値、転送ステータス=USB_DATA_STOP を応答します。

データ未転送のパイプに対する強制終了要求が発生した場合は USB_E_QOVR を応答します。

補足

1. データ送信を中断する場合は SIE 側の FIFO バッファはクリアされません。
2. FIFO がダブルバッファで送信時は FIFO バッファ内に未送信データが残る場合があります。
3. RL78/USB の場合、引数 pipe がパイプ 0～パイプ 3 の場合は USB_E_QOVR のエラーを返します。また、パイプ 8 以上の場合は USB_E_ERROR のエラーを返します

使用例

```
void  usb_smp_task(void)
{
 usb_er_t err;
  :

 /* Transfer end request */
 err = R_usb_hstd_TransferEnd(USB_PIPE4, USB_DO_TRANSFER_TMO);

 return err;
  :
}
```

R_usb_hstd_MgrChangeDeviceState

USB デバイスステート遷移要求

形式

usb_er_t R_usb_hstd_MgrChangeDeviceStat(usb_cb_info_t complete,
usb_strct_t msginfo,
usb_strct_t keyword)

引数

complete	USB ステート遷移終了時に実行するコールバック関数
msginfo	変更させたい USB ステート
keyword	ポート番号、デバイスアドレス、パイプ番号など msginfo により内容が異なる

戻り値

USB_E_OK	成功
USB_E_ERROR	失敗、引数エラー

解説

引数 msginfo に以下の値を設定し、USB-BASIC-F/Wに USB ステート遷移を要求します。

・USB_DO_PORT_ENABLE / USB_DO_PORT_DISABLE

keyword で指定されたポートの許可/禁止（VBUS 出力の on/off 制御）を行います。

・USB_DO_GLOBAL_SUSPEND

keyword で指定されたポートをサスペンド状態にします。

・USB_DO_ GLOBAL _RESUME

keyword で指定されたポートをレジュームします。

・USB_DO_CLEAR_STALL

keyword で指定されたパイプの STALL 状態を解除します。

補足

1. USB-BASIC-F/Wが接続/切断を割り込みで検出した場合は、USB-BASIC-F/Wが自動的にエニュメレーションシーケンス処理、デタッチシーケンス処理を行います。
2. 本関数で USB ステート遷移させた場合は、API 関数 *R_usb_hstd_DriverRegistration()*で登録されたドライバ構造体の USB ステート遷移コールバックは呼び出しされません。

使用例

```
void  usb_smp_task(void)
{
  R_usb_hstd_MgrChangeDeviceState
  (usb_hsmpl_status_result, USB_DO_GLOBAL_SUSPEND, g_usb_hsmpl_Port);
}
```

R_usb_hstd_ChangeDeviceState

USB IP の状態設定要求

形式

usb_er_t R_usb_hstd_ChangeDeviceState(usb_strct_t msginfo)

引数

msginfo	変更させたい USB ステート

戻り値

USB_E_OK	成功
USB_E_ERROR	失敗、引数エラー

解説

引数 msginfo に以下の値を設定し、USB-BASIC-F/Wに USB ステート遷移を要求します。

・USB_DO_INITHWFUNCTION

USB-IP を起動し SW リセットを行います。USB-BASIC-F/W起動前に実行してください。

・USB_DO_SETHWFUNCTION

USB-IP の機能をホスト設定にします。UPL 登録後に実行してください。

補足

1. 当該関数は MGR タスク、HCD タスクを介さずに処理を実行します。

使用例

```
void  usb_smp_task(void)
{
  R_usb_hstd_ChangeDeviceState(USB_DO_INITHWFUNCTION);
  R_usb_hstd_HcdOpen();                    /* HCD task open */
  usb_hsmpl_driver_registration();         /* Sample driver registration */
  R_usb_hstd_ChangeDeviceState(USB_DO_SETHWFUNCTION);
   :
   :
}
```

R_usb_hstd_DeviceInformation

USB デバイスステート情報を取得

形式

void R_usb_hstd_DeviceInformation(usb_addr_t devaddr, uint16_t *table)

引数

devaddr	デバイスアドレス
*table	デバイス情報格納用テーブルアドレス

戻り値

―	―

解説

デバイスアドレスが一致したデバイスの USB デバイス情報を取得します。

引数（**table*）に指定したアドレスに以下の情報を格納します。

[0]：ルートポート番号（ポート 0：USB_0, ポート 1：USB_1）

[1]：USB ステート

（未接続：USB_STS_DETACH, エニュメレーション中：USB_STS_DEFAULT/USB_STS_ADDRESS, 接続中：USB_STS_CONFIGURED, サスペンド中：USB_STS_SUSPEND）

[2]：構成番号（*g_usb_HcdDevInfo[g_usb_MgrDevAddr].config*）

[3]：接続速度（FS：USB_FSCONNECT, LS：USB_LSCONNECT, 未接続：USB_NOCONNECT）

補足

1. 引数*table は 4word の領域を用意してください。
2. デバイスアドレスに"0"を指定した場合は以下の情報が応答されます。

   ① エニュメレーション中のデバイスがいない場合

   table[0] = USB_NOPORT, table[1] = USB_STS_DETACH

   ② エニュメレーション中のデバイスがいる場合

   table[0] = ポート番号, table[1] = USB_STS_DEFAULT

使用例

```
void  usb_smp_task(void)
{
 uint16_t  tbl[4];
  :
 /* Device information check */
 R_usb_hstd_DeviceInformation(devaddr, &tbl);
  :
}
```

R_usb_hstd_ChkPipeInfo

パイプ情報テーブルの設定

形式

usb_er_t R_usb_hstd_ChkPipeInfo(uint16_t *table, uint8_t *descriptor)

引数

table	パイプ情報テーブル
descriptor	エンドポイントディスクリプタ

戻り値

USB_DIR_H_IN	IN エンドポイントの設定を実施
USB_DIR_H_OUT	OPUT エンドポイントの設定を実施
USB_E_ERROR	失敗

解説

エンドポイントディスクリプタを解析して 1 パイプ分のパイプ情報テーブルを作成します。

情報が更新されるフィールドは以下のとおりです。

USB_TYPFIELD	USB_BULK または USB_INT
USB_SHTNAKFIELD	USB_SHTNAKON（USB_TYPFIELD == USB_DIR_H_IN 時）
USB_DIRFIELD	USB_DIR_H_IN または USB_DIR_H_OUT
USB_EPNUMFIELD	エンドポイントディスクリプタに示されたエンドポイント番号
USB_IITVFIELD	インターバルカウンタ（2^n の n を指定）

補足

1. パイプ情報テーブルに関しては7.4を参照してください。
2. インターバルカウンタ(フレーム数)は 2 の n 乗で設定します。
3. 接続デバイスが動作可能であればドライバ確認コールバック処理で呼び出してください。
4. インタフェース内に複数のエンドポイントが存在する場合、マルチインタフェースで複数エンドポイントの通信が必要な場合など複数パイプで情報テーブルを作成したい場合は、UPL 側でエンドポイントディスクリプタを検索し当該関数を繰り返し呼び出すなどの処理を実装してください。

使用例

```
void usb_hsmpl_pipe_info(uint8_t *table)
{
 usb_er_t   retval = USB_YES;
 uint16_t   *ptr;

 /* Check Endpoint Descriptor */
 ptr  = g_usb_hsmpl_DefEpTbl;
 for (; table[1] == USB_DT_ENDPOINT, retval != USB_ERROR; table += table[0], ptr
+= USB_EPL)
 {
  retval = R_usb_hstd_ChkPipeInfo(ptr, table);
 }
 return retval;
}
```

R_usb_hstd_ReturnEnuMGR

デバイスクラス判定通知

形式

void	R_usb_hstd_ReturnEnuMGR(uint16_t cls_result)

引数

cls_result	接続デバイスの動作可否

戻り値

―	―

解説

本 API は、接続デバイスが動作可能かどうか（USB_YES/USB_NO）をUSB-BASIC-F/Wに通知します。本関数でUSB-BASIC-F/Wに USB_NO を通知した場合は、他デバイスクラスドライバで動作可能かどうかを確認する必要があります。cls_result には、USB_YES もしくは USB_NO を指定してください。

補足

1. ドライバ確認コールバック関数の終了時に本 API を呼び出してください。
(g_usb_HcdDriver[x].classcheck コールバック関数を参照してください。)

使用例

```
void usb_hsmpl_enumaration(usb_tskinfo_t *mess)
{
  :
  retval = usb_hsmpl_pipe_info(g_usb_hsmpl_InterfaceTable,
           (uint8_t)g_usb_hsmpl_ConfigTable[2]);
  if( retval == USB_ERROR )
  {
    R_usb_hstd_ReturnEnuMGR(USB_NO);
  }
  else
  {
    R_usb_hstd_ReturnEnuMGR(USB_YES);
  }
}
```

R_usb_hstd_SetPipeRegistration

パイプ情報をレジスタに設定

形式

void	R_usb_hstd_SetPipeRegistation(uint16_t* table, uint16_t command)

引数

table	パイプ情報テーブル
command	コマンド

戻り値

―	―

解説

・コマンドが"USB_NO"の場合

パイプ情報テーブルで指定された全パイプを未使用状態に設定します。

・コマンドが"USB_YES"の場合

パイプ情報テーブルで指定された全パイプを未使用状態に設定します。未使用状態にしたのち、パイプ情報テーブルに従い全パイプの設定を行います。

補足

1. パイプ情報テーブルについては、7.4.1章を参照してください。

使用例

```
void usb_hsmpl_open_close(uint16_t data1, uint16_t device_state)
{
 switch(device_state)
 {
  case USB_DEVCONFIG:
    if( data1 == g_usb_hsmpl_Devaddr )
    {
       /* device address set */
       R_usb_hstd_SetPipeRegistration(g_usb_hsmpl_DefEpTbl, USB_YES);
       usb_hsmpl_task_operate(USB_SMPL_INIT);
    }
  break;
  case USB_DEVDETACH:
       :
}
```

*g_usb_HcdDriver[x]. classcheck

エニュメレーション時の接続デバイス動作可否確認のコールバック

形式

void	(*driver->classcheck)((uint16_t**)&table);

引数

table	デバイスクラスドライバへ通知するデバイス情報

戻り値

―	―

解説

接続デバイスの動作可否をデバイスクラスドライバが確認します。

引数の情報テーブルはTable 8-2 classcheckを参照してください。

動作可否は、API 関数 *R_usb_hstd_ReturnEnuMGR()*で通知してください。

補足

1. USB-BASIC-F/Wは Configuration ディスクリプタを受信してコールバックを実行します。

   (*driver->classcheck)((uint16_t**)&table);

2. チェックが終了したら API 関数 *R_usb_hstd_ReturnEnuMGR()*で結果をUSB-BASIC-F/Wに通知してください。

使用例

例としてコールバックされた関数内の処理を記述します。

```
void usb_hsmpl_class_check(uint16_t **table)
{
 g_usb_hsmpl_DeviceTable         = (uint8_t*)((uint16_t*)table[0]);
 g_usb_hsmpl_ConfigTable         = (uint8_t*)((uint16_t*)table[1]);
 g_usb_hsmpl_Devaddr             = (uint16_t)(*table[3]);
 g_usb_hsmpl_EnumerationSeq      = USB_SEQ_0;
 g_usb_hsmpl_Message.msginfo.w   = USB_MSG_CLS_CHECKREQUEST;

 /* Class check of enumeration sequence move to class function */
 if( USB_SND_MSG(USB_HSMP_MBX, (usb_msg_t*)&g_usb_hsmpl_Message) != USB_E_OK )
 {
  while(1);
 }
}
```

*g_usb_HcdDriver[x].statediagram

USB ステート遷移検出時のコールバック

形式

void	(*driver->statediagram)((uint16_t)data1, (uint16_t)device_state);

引数

data1	Device address
device_state	USB device state

戻り値

—	—

解説

USB ステート遷移変更が発生したことを UPL に通知します。

1. アタッチ検出
   (*driver->statediagram)(USB_NO_ARG, USB_STS_ATTACH);
2. USB リセット発行
   (*driver->statediagram)(USB_NO_ARG, USB_STS_DEFAULT);
3. エニュメレーションシーケンス処理終了
   (*driver->statediagram)(driver->devaddr, USB_STS_CONFIGURED);
4. デタッチ検出
   (*driver->statediagram)(g_usb_MgrDevAddr, USB_STS_DETACH);
5. 過電流検出
   (*driver->statediagram)(driver->devaddr, USB_STS_OVERCURRENT);
6. リモートウェイクアップシーケンス処理終了
   (*driver->statediagram)(g_usb_MgrDevAddr, USB_STS_WAKEUP);

補足

1. API 関数 R_usb_hstd_ChangeDeviceState()および *R_usb_hstd_MgrChangeDeviceState()*関数で USB ステート遷移させた場合は、当該コールバックは呼び出しません。
2. アタッチ検出時、USB リセット発行時のコールバック通知は、登録されているすべてのデバイスクラスドライバに実施します。

使用例

例としてコールバックされた関数内の処理を記述します。

```
void usb_hsmpl_device_state(uint16_t data, uint16_t state)
{
  case USB_STS_DETACH:
    usb_hsmpl_transfer_end_all();
    R_usb_hvndr_DriverStop();
  break;
  case USB_STS_ATTACH:
    R_usb_hvndr_DriverStart();
  break;
  case USB_STS_DEFAULT:
  case USB_STS_ADDRESS:
  break;
  case USB_STS_CONFIGURED:
    g_usb_gmpl_DeviceAddr = data1;
    if( g_usb_gmpl_DeviceAddr != 0 )
    {
       R_usb_hstd_SetPipeRegistration(g_usb_hsmpl_DefEpTbl, USB_YES);
    }
    usb_hsmpl_tranfer_all();
  break;

  case USB_STS_SUSPEND:
  break;
  case USB_STS_RESUME:
  case USB_STS_WAKEUP:
    usb_hsmpl_tranfer_all();
  break;
  case USB_STS_OVERCURRENT:
  break;
 }
}
```

*g_usb_LibPipe[pipe]->complete

データ転送終了のコールバック

形式

void (*g_usb_LibPipe [pipe]->complete)((usb_utr_t*)g_usb_LibPipe[pipe]);

引数

g_usb_LibPipe 転送メッセージ

戻り値

― ―

解説

データ転送終了および中断を UPL に通知します。

補足

1. 転送要求時のメッセージを応答します。USB-BASIC-F/W が更新する構造体メンバをTable 8-8に示します。
2. タイムアウト中断（*R_usb_hstd_TransferEnd()*関数で USB_DO_TRANSFER_TMO 指定）はコールバックしません。

Table 8-8 usb_utr_t 構造体メンバ

メンバ	更新有無	機能	備考
tranlen	更新有	USB 通信データ長を通知します。(tranlen は実際に送受信されて転送データ長です)	
status	更新有	以下に、通信結果を示します。 USB_DATA_OK: データ転送（送信/受信）が正常終了した場合 USB_DATA_SHT: データ転送が指定されたデータ長未満で終了した場合 USB_DATA_OVR: 受信データがサイズオーバーした場合 USB_DATA_STOP: データ転送を強制終了した場合 USB_CTRL_END: Control 転送が正常に終了した場(PIPE0 のみ)	
pipectr	更新有	PIPECTR レジスタ(PIPExCTR register)情報を応答します。	
その他	更新無	転送要求内容が格納されます。	

使用例

コールバックされるときの関数の処理例

```
void usb_hsmpl_transfer_result(usb_utr_t *mess)
{
 switch(mess->status)
 {
  case USB_DATA_OK:
  case USB_DATA_SHT:
  case USB_DATA_OVR:
    if ((mess->pipenum == USB_PIPE4) || (mess->pipenum == USB_PIPE5))
    {
                :
    }
  break;
 }
}
```

*g_usb_MgrCallback

API 関数(R_usb_hstd_MgrChangeDeviceState)による USB ステート遷移終了時のコールバック

形式

void (*g_usb_MgrCallback)((uint16_t)keyword, (uint16_t)msginfo);

引数

keyword	ポート番号、デバイスアドレス、パイプ番号など msginfo により内容が異なる
msginfo	USB デバイスステート

戻り値

― ―

解説

API 関数 R_usb_hstd_MgrChangeDeviceState ()の要求が終了したことを通知するコールバック関数です。

1. ポート許可終了
   (*g_usb_MgrCallback)(g_usb_MgrPort, USB_STS_POWER);
2. ポート禁止終了
   (*g_usb_MgrCallback)(g_usb_MgrPort,USB_STS_PORTOFF);
3. サスペンドシーケンス終了
   (*g_usb_MgrCallback)(g_usb_MgrDevAddr, USB_STS_SUSPEND);
4. レジュームシーケンス終了
   (*g_usb_MgrCallback)(g_usb_MgrDevAddr, USB_STS_RESUME);
5. パイプの STALL 解除
   (*g_usb_MgrCallback)(g_usb_CurrentPipe, USB_STALL_SUCCESS);

補足

サスペンド、レジュームはデバイスごと（デバイスクラスドライバごと）にコールバックします。

使用例

―

9. システム定義

9.1 スケジューラ

USB-BASIC-F/Wはスケジューラ関数を使用してタスクを制御します。スケジューラの特徴は以下の通りです。

1. 各タスクや H/W の要求をタスクのタスク ID に従って管理する。
2. 各タスクに複数の要求が発生した場合、FIFO 構造で要求を処理する。
3. 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 UPL 実装が可能。
4. スケジューラが管理するタスクは、関数として構築する。
5. スケジューラは、タスクから抜けるまで、他タスクへのディスパッチやプリエンプトをおこないません。
   注意：スケジューラは、タスクのディスパッチやプリエンプトをおこなわないため、UPL タスクの処理時間が長い場合、USB2.0 規格で規定された USB Control 転送の応答時間を満たさなくなる場合があります。ユーザは構築したシステムで USB2.0 規格を満たしているかの確認をおこなってください。

(1). システムとして定義する設定項

r_usb_ckernelid.h ファイルで設定します。

```
#define   USB_IDMAX      ((uint8_t)5)    : Maximum value of task IDs*1    [9.1.1]
#define   USB_TABLEMAX   ((uint8_t)10)   : Number of messages that can be stored with the task
                                         :  [9.1.2]
#define   USB_BLKMAX     ((uint8_t)5)    : Number of messages that can be obtained in a system
                                         :  [9.1.3]
```

*1: 使用するタスク番号の最大値に 1 を加えた値を設定してください。

(2). タスク情報のセットアップ

各々の追加タスクについては、タスク ID およびメールボックス ID を *r_usb_cKernelid.h* ファイルに追加してください。これらの項目を設定する場合、下記の内容のご注意ください。

・タスク ID は値が重ならないようにしてください。

・メールボックス ID とタスク ID は同じ値を設定してください。

下記にベンダクラスドライバの設定例を示します。

```
#define   USB_PVEN_TSK    USB_TID_3       : Task ID
#define   USB_PVEN_MBX    USB_PVEN_TSK    : Mailbox ID
```

9.1.1 タスク ID とタスク ID の最大値

タスク ID とタスク ID の最大値を設定してください。タスク ID は値が重ならないようにしてください。最大値を設定する場合、使用するタスクの ID の最大値＋1 を設定してください。使用する UPL タスクの ID は使用する数に応じて設定してください。

タスク優先順位はタスク ID 順位になります。最も高い優先順位は 0 になります。ホスト使用時はタスク ID の設定値を"HCD タスク＜MGR タスク＜UPL タスク"となるように設定をして下さい。ペリフェラル使用時はタスク ID の設定値を"PCD タスク＜UPL タスク"となるように設定をして下さい。

タスク ID 設定には *r_usb_cKernrlid.h* ファイル内で定義されたマクロをご使用ください。

9.1.2 タスクに保存可能なメッセージ数

優先テーブルは各タスクからの処理要求を優先順位ごとに格納するテーブルです。処理要求を格納できる最大数を設定します。

9.1.3 システムで取得可能なメッセージの数

システムが R_USB_GET_SND で取得可能なメッセージの数を設定します。R_USB_REL_BLK が実行されるまで領域を確保します。領域を全て使用している場合は R_USB_PGET_SND でエラー応答しますので、システムにあわせて設定を変更してください。

9.2 スケジューラマクロとスケジューラ関数

ユーザが使用可能なスケジューラマクロ及びスケジューラ API 関数をTable 9-2に示します。スケジューラの API 関数は *r_usb_cstd_libapi.c* ファイルにあります。

スケジューラ API関数を使用する場合は、Table 9-1に示した順番に従いヘッダファイルをインクルードしてください。

Table 9-1スケジューラ API ヘッダファイル一覧

ファイル名	説明	備考
r_usb_ctypedef.h	変数型定義	
r_usb_ckernelid.h	システムヘッダファイル	
r_usb_cdefusbip.h	USB ドライバ用各種定義	
r_usb_api.h	USB ドライバ API 関数定義	

Table 9-2 スケジューラマクロとスケジューラ関数一覧

マクロ名	関数名	説明	備考
	R_usb_cstd_Scheduler	スケジューラ処理	
R_USB_RCV_MSG	R_usb_cstd_RecMsg	実行要求(メッセージ送信)されているか確認します。	
R_USB_SND_MSG	R_usb_cstd_SndMsg	処理要求(メッセージ)を送信します。	
R_USB_ISND_MSG	R_usb_cstd_iSndMsg	割り込みから処理要求(メッセージ)を送信します。	
R_USB_WAI_MSG	R_usb_cstd_WaiMsg	スケジューラを指定回数実行後に R_USB_SND_MSG を実行します。	
R_USB_GET_SND	R_usb_cstd_PgetSend	メッセージの領域を確保後に R_USB_SND_MSG を実行します。	
R_USB_REL_BLK	R_usb_cstd_RelBlk	確保したメッセージの領域を開放します。	

R_usb_cstd_Scheduler

スケジューラ処理

形式

uint8_t R_usb_cstd_Scheduler(void)

引数

― ―

戻り値

USB_FLGSET タスク処理がある

USB_FLGCLR タスク処理がない

解説

スケジューラ処理を行います。

タスクの優先順位に従ってタスク間のメッセージをスケジューリングします。

戻り値が USB_FLGSET の場合にタスクを呼び出してください

戻り値が USB_FLGCLR の場合にはタスクを呼び出さないでください。

補足

―

使用例

```
void main(void)
{
 /* Initialized USBIP */
 usb_hsmpl_main_init();

 /* Sample main loop  */
 while( 1 )
 {
  if(R_usb_cstd_Scheduler() == USB_FLGSET ) /* Scheduler */
  {
    R_usb_hstd_HcdTask();    /* HCD Task */
    R_usb_hstd_MgrTask();    /* MGR Task */
    usb_hsmpl_apl_task();
  }
 }
}
```

R_usb_cstd_RecMsg

処理要求されているか確認

形式

usb_er_t R_usb_cstd_RecMsg(uint8_t id, usb_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のアドレスを格納するアドレス

戻り値

USB_E_OK	受信メッセージが有り
USB_E_ERROR	受信メッセージが無し

解説

受信メッセージを確認します。

受信メッセージがある場合は引数"*mess*"に受信したメッセージのアドレスを格納して、戻り値に USB_E_OK を返します。

補足

R_usb_cstd_Scheduler()関数の戻り値が USB_FLGCLR の場合には、R_USB_RCV_MSG を実行しないでください。

使用例

```
void usb_hsmpl_apl_task(void)
{
 usb_utr_t     *mess;
 usb_er_t   err;     /* Error code */

 /* Receive message */
 err = USB_TRCV_MSG( USB_HSMP_MBX, (usb_msg_t**)&mess );
 if( err != USB_E_OK )
 {
  return;
 }

 switch( mess->msginfo )
 {
 case USB_MSG_CLS_CHECKREQUEST:         /* Enumeration */
  usb_hsmpl_enumaration((usb_tskinfo_t *) mess);
  break;
 case USB_MSG_CLS_INIT:                 /* Initialize */
  usb_hsmpl_initialized();
  break;
 case USB_MSG_CLS_TASK:
  usb_hsmpl_application(mess);
  break;
 default:
  break;
 }
}
```

R_usb_cstd_SndMsg

処理要求の格納

形式

usb_er_t R_usb_cstd_SndMsg(uint8_t id, usb_msg_t* mess)

引数

id	メッセージの送信先のタスク ID
mess	送信するメッセージのアドレス

戻り値

USB_E_OK	メッセージの格納が完了した。
USB_E_ERROR	タスク ID が未設定 優先順位テーブルが満杯で通知不可

解説

優先順位テーブルにメッセージを格納します。

補足

1. *usb_cpu_int_disable()*関数で MCU の USB 割り込みを禁止して R_USB_ISND_MSG を呼び出します。
2. R_USB_SND_MSG を使って各タスクを周期的に動作させる場合、優先度が低いタスクは動作しません。優先度の低いタスクを周期的に動作させるには、R_USB_WAI_MSG を使用してください。

使用例

```
void usb_hsmpl_check_request(uint16_t result)
{
 usb_er_t   err;

 g_usb_hsmpl_Message.msginfo  = USB_MSG_CLS_CHECKREQUEST;
 g_usb_hsmpl_Message.status   = result;

 /* Class check of enumeration sequence move to class function */
 err = USB_SND_MSG(USB_HSMP_MBX, (usb_msg_t*)&g_usb_hsmpl_Message);
}
```

R_usb_cstd_iSndMsg

割り込みから処理要求を格納

形式

usb_er_t R_usb_cstd_iSndMsg(uint8_t id, usb_msg_t* mess)

引数

id	メッセージの送信先のタスク ID
mess	送信するメッセージのアドレス戻り値

戻り値

USB_E_OK	メッセージの格納が完了した。
USB_E_ERROR	タスク ID が未設定 優先順位テーブルが満杯で通知不可

解説

割り込みハンドラからメッセージを送信する場合に使用します。

優先順位テーブルにメッセージを格納します。

補足

―

使用例

```
void R_usb_hstd_InterruptHandler(void)
{
 usb_er_t      err;
 usb_intinfo_t    *ptr;

 /* Initialize Interrupt handler message */
 ptr = &g_usb_cstd_IntMsg[g_usb_cstd_IntMsgCnt];
 usb_hstd_check_interrupt_source(&ptr->keyword, &ptr->status);
 err = USB_ISND_MSG(USB_HCD_MBX, (usb_msg_t*)ptr);

 /* Renewal Message count  */
 g_usb_cstd_IntMsgCnt++;
 if( g_usb_cstd_IntMsgCnt == USB_INTMSGMAX )
 {
  g_usb_cstd_IntMsgCnt = 0;
 }

}
```

R_usb_cstd_WaiMsg

スケジューラを指定回数実行後に **R_USB_SND_MSG** を実行します。

形式

usb_er_t R_usb_cstd_WaiMsg(uint8_t id, usb_msg_t* mess, uint16_t count)

引数

id	メッセージ送信先のタスク ID
mess	送信するメッセージのアドレス
count	スケジューラの実行回数

戻り値

USB_E_OK	待ち行列にメッセージが格納できた。
USB_E_ERROR	タスク ID が未設定 待ち行列が満杯で通知不可

解説

スケジューラを指定回数実行後に R_USB_SND_MSG を実行します。

補足

1. メッセージ通知を遅延させる場合に使用します。また周期タスクの再起動として使用します。
2. 指定したタスクがすでに待ち状態にある場合は" *count* ".を無視して待ち行列に登録します。
3. R_USB_SND_MSG が USB_E_OK の場合は待ち行列を FIFO 構造で更新します。

   待ち行列に複数のメッセージが登録されている場合は、2 番目以降のメッセージを"*count =1*"として待ち回数を再カウントします。
4. R_USB_SND_MSG が USB_E_ERROR の場合は待ち行列を更新しません。

   カウントの終了したメッセージを"*count =1*"として待ち回数を再カウントします。
5. 引数"*count* "が"*0* "の場合は"*id*"で指定されたタスクの待ち行列をクリアします。

使用例

```
/* enumeration wait setting */
if( g_usb_HcdMgrMode[elseport] == USB_DEFAULT )
{
 err = USB_WAI_MSG(USB_MGR_MBX, (usb_msg_t*)g_usb_MgrMessage, 100);
 if( err != USB_E_OK )
 {
  USB_PRINTF1("### hMgrTask snd_msg error (%ld)\n", err);
 }
}
```

R_usb_cstd_PgetSend

メッセージ領域を確保後に **R_USB_SND_MSG** を実行します。

形式

usb_er_t R_usb_cstd_PgetSend(uint8_t id, usb_strct_t msginfo, usb_cbinfo_t complete, usb_strct_t keyword)

引数

id	メッセージの送信先のタスク ID
msginfo	送信するメッセージの情報
complete	送信するメッセージのコールバック関数
keyword	送信するメッセージのキーワード

戻り値

USB_E_OK	メッセージの送信が完了した。
USB_E_ERROR	タスク ID が未設定 優先順位テーブルが満杯で通知不可 全てのメッセージ領域が使用中

解説

メモリブロックからメッセージの領域を確保します。

確保した領域のメッセージに引数の id, msginfo, complete, keyword を格納します。

R_USB_SND_MSG が USB_E_OK の場合は確保した領域の flag をセットします。

補足

1. "*flag*"は確保した領域のインデックスです。R_USB_REL_BLK で領域を開放する時のインデックス番号として指定してください。

使用例

```
void usb_hstd_detach(usb_port_t port)
{
 /* ATTCH interrupt enable */
 USB_CLR_PAT(DVSTCTR0, (uint16_t)(USB_RWUPE | USB_USBRST | USB_RESUME |
USB_UACT));
 usb_hstd_attch_enable(port);
 USB_PGET_BLK
  (USB_MGR_MBX, USB_DO_DETACH, &usb_cstd_dummy_function ,(uint8_t)port);
}
```

R_usb_cstd_RelBlk

確保したメッセージの領域を開放します。

形式

usb_er_t R_usb_cstd_RelBlk(uint8_t blk_num)

引数

blk_num	領域を開放したいメッセージのインデックス番号

戻り値

USB_E_OK	指定されたメッセージ領域を開放した。
USB_E_ERROR	指定されたメッセージ領域を開放できなかった。

解説

引数"*blk_num*"をインデックスとして、開放したい領域の"*flag*"を検索します。

"*blk_num*"と"*flag*"が一致すると領域を開放します。

補足

―

使用例

```
void R_usb_pstd_PcdTask(usb_vp_int_t stacd)
{
 usb_tskinfo_t    *mess;
 /* Error code */
 usb_er_t   err;

 err = USB_TRCV_MSG(USB_PCD_MBX, (usb_msg_t**)&mess, (usb_tm_t)10000);
 if( (err != USB_E_OK) )
 {
  return;
 }

 g_usb_PcdMessage = (usb_tskinfo_t*)mess;

 switch( g_usb_PcdMessage->msginfo )
 {
  case USB_DO_REMOTEWAKEUP:
  case USB_PCD_DP_ENABLE:
  case USB_PCD_DP_DISABLE:
    (*g_usb_PcdCallback)((uint16_t)USB_NO_ARG, g_usb_PcdMessage->msginfo);
    USB_REL_BLK(g_usb_PcdMessage->flag);
  break;
  default:
  break;
 }
}
```

9.3 共通ライブラリ関数

ホスト/ペリフェラル共通で使用するユーザが使用可能な共通ライブラリ API 関数をTable 9-3に示します。

共通ライブラリAPI関数*r_usb_cstdapi.c*ファイルにあります。

共通ライブラリAPI関数を使用する場合は、*r_usb_api.h*をインクルードしてください。

Table 9-3 共通ライブラリ関数一覧

関数名	説明	備考
R_usb_cstd_SetBufPipe0	パイプ 0 の PID を BUF に設定する。	
R_usb_cstd_debug_hook()	異常処理時にコールされます。	

R_usb_cstd_SetBufPipe0

パイプ 0 の PID を BUF に設定する。

形式

void R_usb_cstd_SetBufPipe0(void)

引数

—	—

戻り値

—	—

解説

パイプ 0 の PID を BUF に設定する。

補足

PID、BUF ついては、MCU のハードウェアマニュアルを参照してください。

使用例

```
void usb_pstd_set_ccpl(void)
{
 R_usb_cstd_SetBufPipe0();              /* Request ok */
 USB_SET_PAT(DCPCTR, USB_CCPL);         /* Status stage start */
}
```

R_usb_cstd_debug_hook

不正処理発生時のデバッグフック処理

形式

void R_usb_cstd_debug_hook(uint16_t error_code)

引数

error_code	上位 8 ビット: エラー発生原因部 下位 8 ビット: エラーシリアル番号

戻り値

—	—

解説

1. 不正処理発生時に本 API をコールしてください。
2. 以下のコードはエラー発生原因を示しています。これらのコードは、r_user_config.h に定義されています。

エラーコード	内容
USB_DEBUG_HOOK_HOST	USB ホスト処理内でエラーが発生した場合、引数にこのコードを指定してください。
USB_DEBUG_HOOK_PERI	USB ペリフェラル処理内でエラーが発生した場合、引数にこのコードを指定してください。
USB_DEBUG_HOOK_HWR	ハードウェアによるエラーの場合、引数にこのコードを指定してください。
USB_DEBUG_HOOK_STD	ホスト、ペリフェラル共通処理内でエラーが発生した場合、引数にこのコードを指定してください。
USB_DEBUG_HOOK_CLASS	クラス処理内でエラーが発生した場合、引数にこのコードを指定してください。
USB_DEBUG_HOOK_APL	アプリケーション処理内でエラーが発生した場合、引数にこのコードを指定してください。

補足

—

使用例

```
void user_application(void)
{
                :
    if(error)
    {
        R_usb_cstd_debug_hook(USB_DEBUG_HOOK_APL | USB_DEBUG_HOOK_CODE1);
    }
}
```

10. 制限事項

USB-BASIC-F/Wには、以下の制限事項があります。

1. パイプ情報設定関数でパイプ使用方法を制限しています。
   ・受信パイプは SHTNAK 機能でトランザクションカウンタを使用します。
2. 型の異なるメンバで構造体を構成しています。
   （コンパイラによって構造体メンバのアドレスアライメントずれが発生することがあります）
3. UPL はユーザにてご用意ください。

11. e² studio 用プロジェクトのセットアップ

(1). e² studio を起動してください。

※ はじめてe² studio を起動する場合、Workspace Launcher ダイアログが表示されますので、プロジェクトを格納するためのフォルダを指定してください。

(2). [ファイル] → [インポート]を選択してください。インポートの選択ダイアログが表示されます。

(3). インポートの選択画面で、 [既存プロジェクトをワークスペースへ] を選択してください。

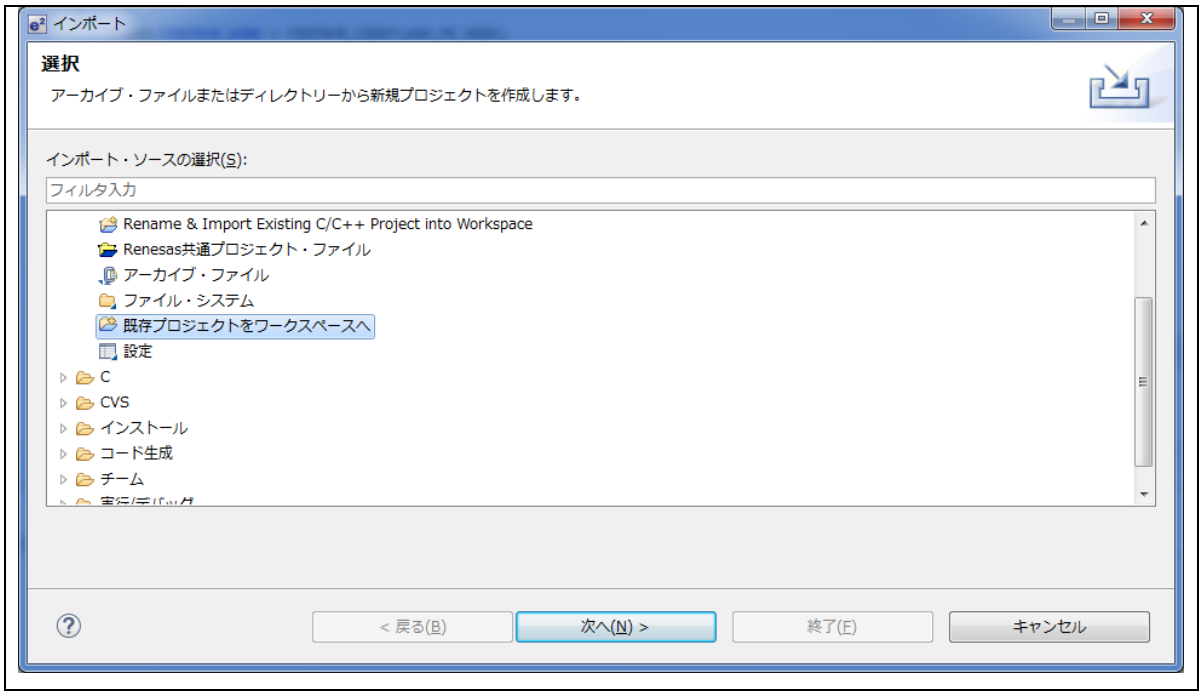

Figure 11-1 インポートの選択

(4). [ルートディレクトリの選択] の [参照] ボタンを押下して、「.cproject 」(プロジェクトファイル) が格納されたフォルダを選択して下さい。

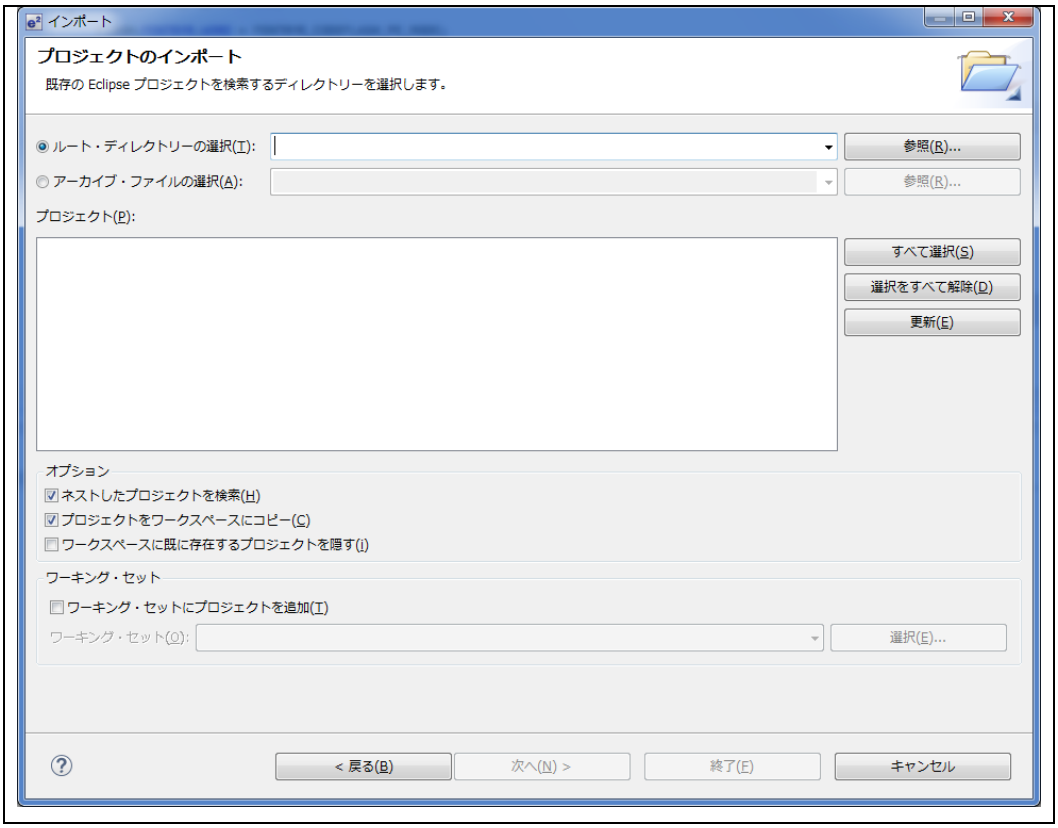

Figure 11-2　プロジェクトのインポート画面

(5). [終了]をクリック して下さい。

プロジェクトのワークスペースへのインポートが完了します。

12. e² studio 用プロジェクトを CS+で使用する場合

本プロジェクトは、統合環境 e² studio で作成されています。本プロジェクトを CS+で動作させる場合は、下記の手順を行ってください。

[Note]

rcpc ファイルは、workspace\RL78\CCRL\(MCU 名)フォルダ内に用意されています。

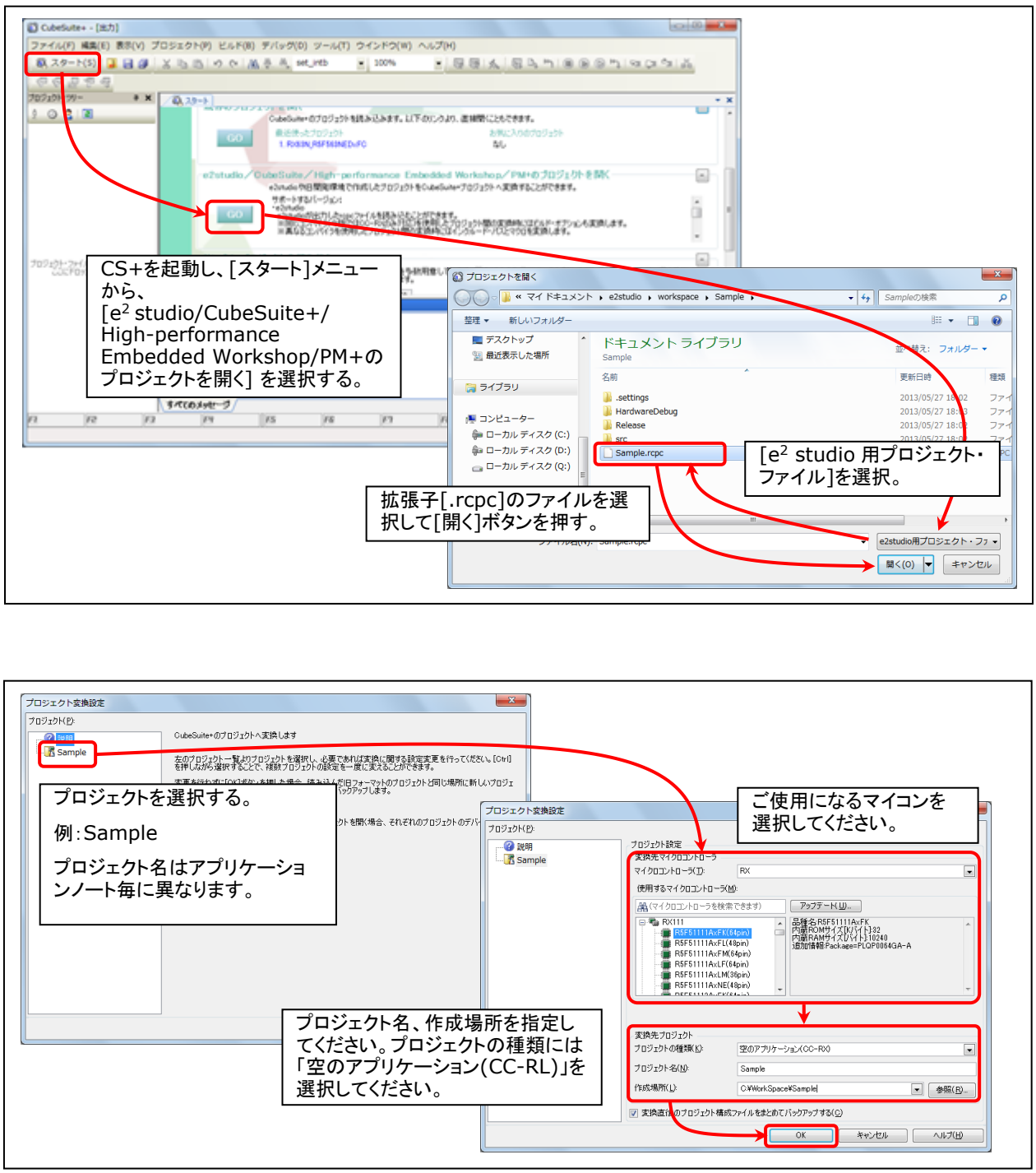

Figure 12-1 e² studio 用プロジェクトの CS+読み込み方法

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
Rev.1.00	2011.02.16	—	Rev.1.00 発行
Rev.2.00	2012.11.30	—	ファームウェアアップデートによるドキュメントの改訂
Rev.2.01	2013.03.26	—	IAR 環境について追記
Rev.2.10	2013.08.01	—	RL78/L1C, RX111 に対応、誤記訂正
Rev.2.11	2013.10.31	—	4-3-1 フォルダ構成を変更。これに伴い、以下の箇所のパス記述を修正。 - 1.4 本書の読み方 - 4.3.2 ファイル一覧 Table 4-4 ファイル一覧 誤記訂正。
Rev.2.12	2014.03.31	—	誤記訂正
Rev.2.13	2015.03.16	—	動作確認デバイスから RX111 を削除。
Rev.2.14	2016.01.18	—	Technical Update(発行番号: TN-RL*-A055A/J, TN-RL*-A033B/J)に対応しました。
Rev.2.15	2016.03.28	—	1. CC-RL コンパイラをサポートしました。 2. USB Host 時、MaxPacketSize×n サイズのデータを送信するコントロール転送において Null パケットが送信されないように変更しました。

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、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 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に関し、当社は、何らの責任を負うものではありません。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. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製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）、交通用信号機器、
   防災・防犯装置、各種安全装置等
   当社製品は、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（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。 たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。 なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障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、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、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。お客様が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. 本資料に記載されている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。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
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。

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その総株主の議決権の過半数を直接または間接に保有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。

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：http://japan.renesas.com/contact/