RZ/A1H グループ

R01AN3428JJ0110
Rev.1.10
Sep 30, 2016

USB Host Communications Devices Class Driver (HCDC)

要旨

本アプリケーションノートでは、USB Host コミュニケーションデバイスクラスドライバ（HCDC）について説明します。本ドライバは USB Basic Firmware（USB-BASIC-FW）と組み合わせることで動作します。以降、本ドライバを HCDC と称します。

対象デバイス

RZ/A1H グループ

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

関連ドキュメント

1. Universal Serial Bus Revision 2.0 specification
   【http://www.usb.org/developers/docs/】
2. USB Class Definitions for Communications Devices Revision 1.2
3. USB Communications Class Subclass Specification for PSTN Devices Revision 1.2
   【http://www.usb.org/developers/docs/】
4. RZ/A1H グループ、RZ/A1M グループ　ユーザーズマニュアル ハードウェア編
   (ドキュメント No.R01UH0403JJ)
5. RZ/A1H グループ USB Host and Peripheral Interface Driver（ドキュメント No.R01AN3291JJ）
6. RZ/A1Hグループ ARM® Development Studio 5（DS-5™）のセミホスティング機能を使用したNOR型フラッシュメモリへのダウンロード例（ドキュメントNo.R01AN1957JJ）
7. RZ/A1H グループレジスタ定義ヘッダ・ファイル iodefine.h（R01AN1860JJ）
8. RZ/A1H グループ初期設定例（R01AN1864JJ）

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

目次

1. 概要

HCDC は、USB-BASIC-FWと組み合わせることで、USB Host コミュニケーションデバイスクラスドライバ（以降 HCDC と記述）として動作します。HCDC は、USB コミュニケーションデバイスクラス仕様（以降 CDC と記述）の PSTN デバイス・サブクラス Abstract Control Model に準拠し、CDC ペリフェラル装置との通信を行うことができます。

以下に、本モジュールがサポートしている機能を示します。

- 接続デバイスの照合
- 通信回線設定の実施
- 通信回線の状態取得
- CDC ペリフェラルデバイス機器とのデータ通信
- 一つの USB チャネルに対し USB Hub を使って最大 2 つの CDC デバイスの接続が可能。

1.1 必ずお読みください

お客様が、このドライバを使ってアプリケーションプログラムを作成する場合は、ドキュメント(Document No:R11AN3291JJ)に記載された API の使用をお勧めします。当該ドキュメントは、パッケージ内の "reference_documents"フォルダにあります。

[Note]

1. ドキュメント(Document No:R01AN3291JJ)に記載された API を使ったアプリケーションプログラムの作成方法は当該ドキュメントに記載されています。
2. ドキュメント(Document No:R11AN3291JJ)に記載された API を使用した場合、本書の「7.3 HCDC API 一覧」に記載された API を使用する必要はありません。

1.2 動作確認済環境

HMSC の動作確認済環境をTable 1.1に示します。

Table 1.1 動作確認条件

項目	内容
使用マイコン	RZ/A1H
動作周波数（注）	CPU クロック（Iφ）: 400MHz
	画像処理クロック（Gφ）: 266.37MHz
	内部バスクロック（Bφ）: 133.33MHz
	周辺クロック 1（P1φ）: 66.67MHz
	周辺クロック 0（P0φ）: 33.33MHz
動作電圧	電源電圧（I/O）: 3.3V
	電源電圧（内部）: 1.8V
統合開発環境	ARM®t統合開発環境
	・ARM Development Studio（DS-5™） Version 5.16
	IARt 統合開発環境
	・IAR Embedded Workbench for ARM Version 7.40
コンパイラ	ARM C/C++ Compiler/Linker/Assembler Ver.5.03 [Build 102]
	KPIT GNUARM-RZ v14.01
	IAR C/C++ Compiler for ARM 7.40
動作モード	ブートモード 0
	（CS0 空間 16 ビットブート）
ターミナルソフトの通信設定	・通信速度 ：115200bps
	・データ長 ：8 ビット
	・パリティ ： なし
	・ストップビット長 ：1 ビット
	・フロー制御 ： なし
使用ボード	GENMAI ボード
	・R7S72100 CPU ボード RTK772100BC00000BR
使用デバイス （ボード上で使用する機能）	・シリアルインターフェース（Dsub-9 コネクタ）
	・USB1 コネクタ、USB2 コネクタ

1.3 制限事項

本モジュールには以下の制限事項があります。

・ 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は構造体のメンバにアドレスアライメントずれを発生することがあります。）

用語一覧

APL	:	Application program
CDC	:	Communications Devices Class
CDCC	:	Communications Devices Class －Communications Class Interface
CDCD	:	Communications Devices Class －Data Class Interface
cstd	:	Prefix of function and file for Peripheral & Host Common Basic (USB low level) F/W
HCD	:	Host control driver of USB-BASIC-FW
HCDC	:	Host用 Communication Devices Class
HDCD	:	Host device class driver (device driver and USB class driver)
HUBCD	:	Hub class sample driver
MGR	:	Peripheral device state maneger of HCD
non-OS	:	USB basic firmware for OS less system
PP	:	プリプロセス定義
Scheduler	:	non-OSでタスク動作を簡易的にスケジューリングするもの
Scheduler Macro	:	non-OSでスケジューラ機能を呼び出すために使用されるもの
Task	:	処理の単位
USB-BASIC-FW	:	USB basic firmware for RZ/A1H グループ (non-OS)
USB	:	Universal Serial Bus

2. ソフトウェア構成

Figure 2-1に PCDC のモジュール構成、Table 2.1にモジュール機能概要を示します。

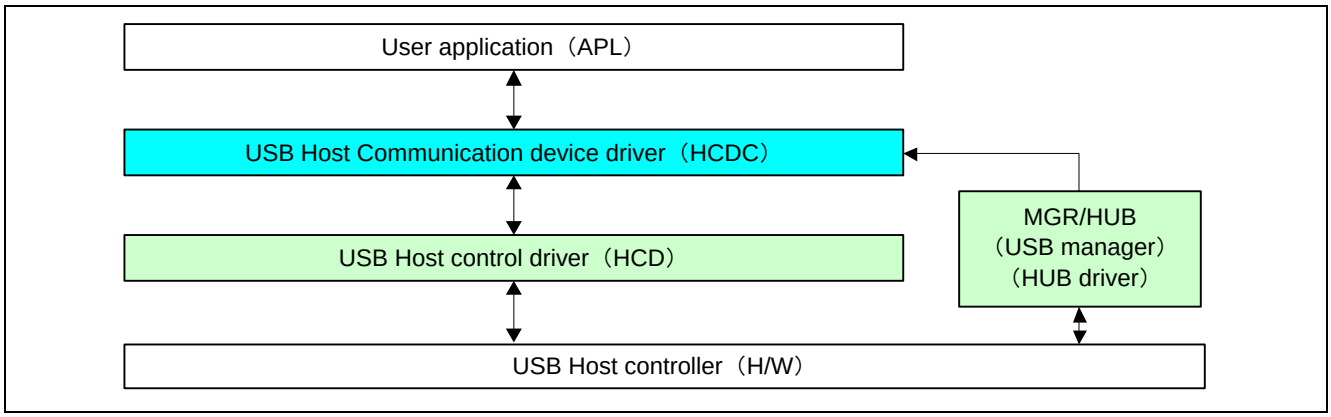

Figure 2-1 モジュール構成図

Table 2.1 モジュール説明

モジュール名	説明
HCDC	APL からの CDC に関するリクエストおよびデータ通信を HCD へ要求します。
MGR / HUB	接続されたデバイスとエニュメレーションをして HCDC を起動します。またデバイスの状態管理も行います。
HCD	USB Host H/W 制御ドライバです。

3. システム資源

Table 3.1～Table 3.3に、HCDC が使用しているシステム資源を示します。

Table 3.1 タスク情報

関数名	タスク ID	優先度	概要
usb_hcdc_Task	USB_HCDC_TSK	USB_PRI_3	HCDC タスク

Table 3.2 メールボックス情報

関数名	タスク ID	優先度	概要
usb_hcdc_Task	USB_HCDC_TSK	USB_PRI_3	HCDC タスク

Table 3.3 メモリプール情報

関数名	タスク ID	優先度	概要
usb_hcdc_Task	USB_HCDC_TSK	USB_PRI_3	HCDC タスク

（注）全システムのメモリブロックの最大数は、USB_BLKMAX で定義されます。初期値は 20 です。

4. ターゲットペリフェラルリスト（TPL）

USB ホストドライバ（USB-BASIC-F/W）とデバイスクラスドライバを組み合わせて使用する際、デバイスドライバごとにターゲットペリフェラルリスト（TPL）を作成する必要があります。

TPL の詳細は USB Host and Peripheral Interfece Driver アプリケーションノート(Document No.R01AN3291JJ)の「ターゲットペリフェラルリストの設定方法」を参照してください。

5. コンパイル時の設定

本プロジェクトを使用する場合、USB-BASIC-FWをホストに設定する必要があります。USB-BASIC-FWの設定は、ドキュメント(Document No:R01AN3291JJ)を参照してください。

本モジュールのコンフィギュレーションオプションの設定は、r_usb_hcdc_config.h で行います。

オプション名および設定値に関する説明を、下表に示します。

Configuration options in r_usb_hcdc_config.h	
USB_HCDC_IF_CLASS	接続する CDC デバイスのデバイスクラス ID を指定します。 ・ USB_IFCLS_VEN：ベンダクラスのデバイスの接続を許可する ・ USB_IFCLS_CDC：CDC クラスのデバイスの接続を許可する
USB_HCDC_MULTICONNECT	一つの USB モジュール(USB チャネル)に複数の CDC デバイスを接続する場合は、この定義を有効にしてください。
USB_HCDC_IN_DATA_PIPE USB_HCDC_OUT_DATA_PIPE USB_HCDC_IN_DATA_PIPE2 USB_HCDC_OUT_DATA_PIPE2	データ転送に使用されるパイプ番号を指定してください。 (PIPE1 から PIPE5 のうちいずれかを指定してください。同じパイプ番号は指定しないでください。)
USB_HCDC_STATUS_PIPE USB_HCDC_STATUS_PIPE2	Class Notification 用の使用されるパイプ番号を指定してください。 (PIPE6 ら PIPE9 のうちいずれかを指定してください。同じパイプ番号は指定しないでください。)

[Note]

1. CDC デバイスに市販されている USB－シリアル変換デバイスを使用される場合、デバイスクラス ID が CDC でなく、ベンダクラスになっている場合があります。 ご使用前に CDC デバイスの仕様をご確認下さい。
2. USB_HCDC_MULTICONNECT 有効時にのみ USB_HCDC_IN_DATA_PIPE2、USB_HCDC_OUT_DATA_PIPE2 および USB_HCDC_STATUS_PIPE2 定義に対しパイプ番号を指定してください。

6. コミュニケーションデバイスクラス（CDC）,PSTN and ACM

本 S/W は、コミュニケーションデバイスクラス仕様 Abstract Control Model（ACM）サブクラスに準拠しています。なお、Abstract Control Model 仕様は、”関連ドキュメント”に記載されている PSTN に仕様が定められています。

Abstract Control Model サブクラスは、USB 機器と従来のモデム（RS-232C 接続）との間を埋める技術で、従来のモデムを使用するアプリケーションプログラムが使用可能です。

6.1 基本機能

HCDC の主な機能を以下に示します。

(1). 接続デバイスの照合
(2). 通信回線設定の実施
(3). 通信回線の状態取得
(4). CDC ペリフェラルデバイス機器とのデータ通信

6.2 Abstract Control Model(ACM) クラスリクエスト（ホスト→デバイスへの要求）

HCDC が対応している ACM クラスリクエストをTable6.1に示します。

Table6.1 CDC クラスリクエスト

リクエスト	コード	説明
SendEncapsulatedCommand	0x00	プロトコルで定義された AT コマンド等を送信する。
GetEncapsulatedResponse	0x01	SendEncapsulatedCommand で送信したコマンドに対するレスポンスを要求する。
SetCommFeature	0x02	機器固有の 2 バイトコードや、カントリー設定の禁止／許可を設定する。
GetCommFeature	0x03	機器固有の 2 バイトコードや、カントリー設定の禁止／許可状態を取得する。
ClearCommFeature	0x04	機器固有の 2 バイトコードや、カントリー設定の禁止／許可設定をデフォルト状態に戻す。
SetLineCoding	0x20	通信回線設定を行う。（通信速度、データ長、パリティビット、ストップビット長）
GetLineCoding	0x21	通信回線設定状態を取得する。
SetControlLineState	0x22	通信回線制御信号 RTS、DTR の設定を行う。
SendBreak	0x23	ブレイク信号の送信を行う。

Abstract Control Model リクエストについては、USB Communications Class Subclass Specification for PSTN Devices Revision 1.2 の Table11：Requests-Abstract Control Model を参照して下さい。

本クラスドライバソフトが対応するクラスリクエストのデータフォーマットを以下に記します。

6.2.1 SendEncapsulatedCommand

SendEncapsulatedCommand データフォーマットをTable6.2に示します。

Table6.2 SendEncapsulatedCommand データフォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SEND_ENCAPSULATED_COMMAND (0x00)	0x0000	0x0000	データレングス	制御プロトコルコマンド

【注】 Data にはモデム制御の為に AT コマンド等を設定し、wLength にはその長さを設定します。

6.2.2 GetEncapsulatedResponse

GetEncapsulatedResponse データフォーマットをTable6.3に示します。

Table6.3 GetEncapsulatedResponse データフォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	GET_ENCAPSULATED_RESPONSE(0x01)	0x0000	0x0000	データレングス	プロトコルに依存したデータ

【注】 Data には SendEncapsulatedCommand に対する応答データが渡され、wLength にはその長さが設定されます。

6.2.3 SetCommFeature

SetCommFeature データフォーマットをTable6.4に示します。

Table6.4 SetCommFeature データフォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_COMM_FEATURE (0x02)	Feature Selector 注	0x0000	データレングス	State Feature Selector によりカントリーコード又は Abstract Control Model アイドル設定、多重化設定の何れかになります。

【注】Table6.6 Feature Selector 設定一覧参照

6.2.4 GetCommFeature

GetCommFeature データフォーマットをTable6.5に示します。

Table6.5 GetCommFeature データフォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	GET_COMM_FEATURE (0x03)	Feature Selector 注	0x0000	データレングス	Status Feature Selector によりカントリーコード又は Abstract Control Model アイドル設定、多重化設定の何れかになります。

【注】Table6.6 Feature Selector 設定一覧参照

Feature selector 設定をTable6.6、ABSTRACT_STATE 時の Status フォーマットをTable6.7に示します。

Table6.6 Feature Selector 設定一覧

Feature Selector	Code	Targets	Length of Data	Description
RESERVED	00h	None	None	リザーブ
ABSTRACT_STATE	01h	Interface	2	Abstract Control Model のアイドルステート、多重化信号についての設定を選択します。
COUNTRY_SETTING	02h	Interface	2	ISO3166 で定義される 16 進形式のカントリーコードを選択します。

Table6.7 ABSTRACT_STATE 選択時、Status フォーマット

ビット position	Description
D15～D2	リザーブ
D1	データ多重化設定 1 ： Data クラスでコールマネジメントコマンドの多重化を許可 0 ： 多重化禁止
D0	アイドル設定 1 ： 対象インタフェースの全てのエンドポイントにおいてホストからのデータを受け入れず、ホストに対してデータを提供しません。 0 ： エンドポイントにおいてデータの受け入れ、提供をし続けます。

6.2.5 ClearCommFeature

ClearCommFeature データフォーマットをTable6.8に示します。

Table6.8 ClearCommFeature データフォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	CLEAR_COMM_FEATURE (0x04)	Feature Selector 注	0x0000	0x0000	None

【注】Table6.6 Feature Selector 設定一覧参照

6.2.6 SetLineCoding

SetLineCoding データフォーマットをTable6.9に示します。

Table6.9 SetLineCoding データフォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_LINE_CODING(0x20)	0x0000	0x0000	0x0007	Line Coding Structure Table6.10 Line Coding Structure フォーマット参照

Line Coding Structure フォーマットをTable6.10に示します。

Table6.10 Line Coding Structure フォーマット

Offset	Field	Size	Value	Description
0	dwDTERate	4	Number	データ端末の速度（bps）
4	bCharFormat	1	Number	ストップビット 0 - 1 Stop ビット 1 - 1.5 Stop ビット 2 – 2 Stop ビット
5	bParityType	1	Number	パリティ 0 - None 1 - Odd 2 - Even 3 - Mask 4 - Space
6	bDataBits	1	Number	データビット（5、6、7、8）

6.2.7 GetLineCoding

GetLineCoding データフォーマットをTable6.11に示します。

Table6.11 GetLineCoding データフォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	GET_LINE_CODING (0x21)	0x0000	0x0000	0x0007	Line Coding Structure Table6.10 Line Coding Structure フォーマット参照

6.2.8 SetControlLineState

SetControllLineState データフォーマットをTable6.12に示します。

Table6.12 SetControlLineState データフォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_CONTROL_LINE_STATE (0x22)	Control Signal ビット map Table6.13 Control Signal ビット map フォーマット参照	0x0000	0x0000	None

Table6.13 Control Signal ビット map フォーマット

ビット Position	Description
D15...D2	予約（0 をセット）
D1	DCE の送信機能を制御 0 - RTS OFF 1 - RTS ON
D0	DTE がレディ状態かの通知 0 - DTR OFF 1 - DTR ON

6.2.9 SendBreak

SendBreak データフォーマットをTable6.14に示します。

Table6.14 SendBreak データフォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SEND_BREAK (0x23)	ブレーク信号出力時間	0x0000	0x0000	None

6.3 クラスノーティフィケーション（デバイス→ホストへの通知）

HCDC のクラスノーティフィケーション対応/非対応をTable6.15に示します。

Table6.15 CDC クラスノーティフィケーション

ノーティフィケーション	コード	説明	対応
NETWORK_CONNECTION	0x00	ネットワーク接続状況を通知する	×
RESPONSE_AVAILABLE	0x01	GET_ENCAPSLATED_RESPONSE への応答	○
SERIAL_STATE	0x20	シリアル回線状態を通知する	○

6.3.1 SerialState

SerialState データフォーマットをTable6.16に示します。

Table6.16 SerialState フォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	SERIAL_STATE (0x20)	0x0000	0x0000	0x0002	UART State ビット map Table6.17 UART State ビットマップフォーマット参照

UART State ビットマップフォーマットをTable6.17に示します。

Table6.17 UART State ビットマップフォーマット

ビット	Field	Description
D15～D7		予約
D6	bOverRun	オーバーランエラー検出
D5	bParity	パリティエラー検出
D4	bFraming	フレミングエラー検出
D3	bRingSignal	着信（Ring signal）を感知した
D2	bBreak	ブレーク信号検出
D1	bTxCarrier	Data Set Ready：回線が接続されて通信可能
D0	bRxCarrier	Data Carrier Detect：回線にキャリア検出

6.3.2 ResponseAvailable

ResponseAvailable データフォーマットをTable6.18に示します。

Table6.18 ResponseAvailable データフォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	RESPONSE_AVAILABLE(0x01)	0x0000	0x0000	0x0000	None

7. USB ホストコミュニケーションデバイスクラスドライバ（HCDC）

7.1 基本機能

HCDC は、コミュニケーションデバイスクラス仕様 Abstract Control Model サブクラスに準拠しています。
HCDC の主な機能を以下に示します。

(1).CDC ペリフェラルデバイスに対してクラスリクエスト要求
(2).CDC ペリフェラルデバイスとのデータ通信
(3).CDC ペリフェラルデバイスからのシリアル通信エラー情報受信

7.2 構造体

7.2.1 HCDC クラスリクエスト構造体

CDC のクラスリクエスト SetLineCoding 及び、GetLineCoding で使用する UART 設定パラメータ用の構造体をTable7.1に記します。

Table7.1 USB_HCDC_LineCoding_t 構造体

型	メンバ名	説明	備考
uint32_t	dwDTERate	回線速度	単位：bps
uint8_t	bCharFormat	ストップビット設定	
uint8_t	bParityType	パリティ設定	
uint8_t	bDataBits	データビット長	

CDC リクエスト SetControlLineState で使用する UART 設定パラメータ用の構造体をTable7.2に示します。

Table7.2 USB_HCDC_ControlLineState_t Structure

型	メンバ名	説明	備考
uint16_t (D1)	bRTS:1	Carrier control for half duplex modems 0 - Deactivate carrier, 1 - Activate carrier	
uint16_t (D0)	bDTR:1	Indicates to DCE if DTE is present or not 0 - Not Present, 1 - Present	

CDC リクエスト SendEncapsulatedCommand および GetEncapsulatedResponse で使用する AT コマンドパラメータ用の構造体をTable7.3に示します。

Table7.3 USB_HCDC_Encapsulated_t Structure

型	メンバ名	説明	備考
uint8_t	*p_data	AT コマンドデータが格納されている領域	
uint16_t	wLength	AT コマンドデータのサイズ	単位：byte

CDC リクエスト SendBreak で使用する Break 信号送出パラメータ用の構造体をTable7.4に示します。

Table7.4 USB_HCDC_BreakDuration_t Structure

型	メンバ名	説明	備考
uint16_t	wTime_ms	Duration of Break	単位：ms

7.2.2 CommFeature 機能選択共用体

CDC リクエスト *SetCommFeature* および *GetCommFeature* で使用する“Feature Selector”パラメータ用の構造体をTable7.5とTable7.6に、共用体をTable7.7 に示します。

Table7.5 USB_HCDC_AbstractState_t Structure

型	メンバ名	説明	備考
uint16_t	bDMS:1	Data Multiplexed State	
iomt16_t	bIS:1	Idle Setting	

Table7.6 USB_HCDC_CountrySetting_t Structure

型	メンバ名	説明	備考
uint16_t	country_code	Country code in hexadecimal format as defined in [ISO3166],	

Table7.7 USB_HCDC_CommFeature_t 共用体

型	メンバ名	説明	備考
USB_HCDC_AbstractState_t	abstractState	Abstract Control Model 選択時パラメータ	
USB_HCDC_CountrySetting_t	countrySetting	Country Setting 選択時パラメータ	

7.2.3 CDC リクエスト入力パラメータ共用体

CDC リクエスト毎のパラメータ構造体を共用体として定義したものをTable7.8に示します。.

Table7.8 USB_HCDC_ClassRequestParm_t 共用体

リクエスト	リクエストコード/構造体の型	メンバー名	説明
SetLineCoding	USB_HCDC_SET_LINE_CODING / USB_HCDC_LineCoding_t	*LineCoding	データステージで送受信するデータアドレス Table7.1参照
GetLineCoding	USB_HCDC_GET_LINE_CODING / USB_HCDC_LineCoding_t		
SetControlState	USB_HCDC_SET_ CONTROL_LINE_STATE / USB_HCDC_ControlLineState_t	ControlLineState	wValue に設定する値 Table7.2参照
SendEncapsulated Command	USB_HCDC_SEND_ ENCAPSULATED_COMMAND / USB_HCDC_Encapsulated_t	Encapsulated	データステージで送受信するデータアドレスと wValue に設定する値 Table7.3参照
GetEncapsulated Response	USB_HCDC_GET_ ENACAPSULATED_RESPONSE / USB_HCDC_Encapsulated_t		
SendBreak	USB_HCDC_SEND_BREAK / USB_HCDC_BreakDuration_t	BreakDuration	wValue に設定する値 Table7.4参照
SetCommFeature	USB_HCDC_SET_COMM_FEATURE / USB_HCDC_CommFeature_t	*CommFeature	データステージで送受信するデータアドレス Table7.7参照
GetCommFeature	USB_HCDC_GET_COMM_FEATURE / USB_HCDC_CommFeature_t		
ClearCommFeature	USB_HCDC_CLR_COMM_FEATURE No structure		

7.2.4 クラスリクエスト API 入力パラメータ構造体

クラスリクエスト用パラメータ構造体をTable7.9に記します。

Table7.9 USB_HCDC_ClassRequest_UTR_t 構造体

型	メンバ名	説明	備考
uint16_t	devadr	デバイスアドレス	
USB_REGADR_t	ipp	USB IP ベースアドレス	
uint16_t	ip	USB IP 番号	
USB_CDC_ABS_Req_t	bRequestCode	クラスリクエスト種別	
USB_CDC_ClassRequestParm_t	parm	パラメータ設定値	Table7.8参照
USB_CB_t	complete	クラスリクエスト処理完了コールバック関数	

7.2.5 CDC ノーティフィケーションフォーマット

Table7.10とTable7.11に CDC ノーティフィケーションのデータフォーマットを示します。

Table7.10 Response_Available notification format

Type	Member	Description	Remarks
uint8_t	bmRequestType	0xA1	
uint8_t	bRequest	RESPONSE_AVAILABLE(0x01)	
uint16_t	wValue	0x0000	
uint16_t	wIndex	Interface	
uint16_t	wLength	0x0000	
uint8_t	Data	none	

Table7.11 Serial_State notification format

Type	Member	Description	Remarks
uint8_t	bmRequestType	0xA1	
uint8_t	bRequest	SERIAL_STATE(0x20)	
uint16_t	wValue	0x0000	
uint16_t	wIndex	Interface	
uint16_t	wLength	0x0002	
uint16_t	Data	UART State bitmap	Table7.12参照

UART ポート状態変化の検出によりホストに通知されるクラスノーティフィケーション“SerialState”の UART State Bitmap 構造体をTable7.12に示します。

Table7.12 USB_HCDC_SerialState_t 構造体

型	メンバ名	説明	備考
uint16_t (D6)	bOverRun:1	オーバーランエラー検出	
uint16_t (D5)	bParity:1	パリティエラー検出	
uint16_t (D4)	bFraming:1	フレミングエラー検出	
uint16_t (D3)	bRingSignal:1	着信（Ring signal）を感知	
uint16_t (D2)	bBreak:1	ブレイク信号検出	
uint16_t (D1)	bTxCarrier:1	回線が接続されて通信可能	Data Set Ready
uint16_t (D0)	bRxCarrier:1	回線にキャリア検出	Data Carrier Detect

7.3 HCDC API 一覧

Table7.13に HCDC API 一覧を示します。

[Note]

1. ドキュメント(Document No: R01AN3291JJ)に記載された API を使用する場合、アプリケーションプログラムでは、以下の API を使用する必要はありません。

Table7.13 HCDC API 関数一覧

関数名	機能概要
R_usb_hcdc_receive_data	USB 受信処理
R_usb_hcdc_send_data	USB 送信処理
R_usb_hcdc_serial_state_trans	クラスノーティフィケーション SerialState 受信処理
R_usb_hcdc_class_check	ディスクリプタチェック処理
R_usb_hcdc_SetPipeRegistration	パイプ設定処理
R_usb_hcdc_class_request	クラスリクエスト処理
R_usb_hcdc_driver_start	HCDC ドライバタスクスタート処理
R_usb_hcdc_Task	HCDC タスク

7.3.1 R_usb_hcdc_class_check

ディスクリプタチェック処理

形式

void R_usb_hcdc_class_check(USB_UTR_t *ptr, uint16_t **table)

引数

*ptr	USB 通信用構造体
**table	デバイス情報テーブル [0]：デバイスディスクリプタ [1]：コンフィグレーションディスクリプタ [2]：インタフェースディスクリプタ [3]：ディスクリプタチェック結果 [4]：HUB 種別 [5]：ポート番号 [6]：通信速度 [7]：デバイスアドレス

戻り値

―	処理結果（USB_E_OK/USB_E_ERROR）

解説

本 API はクラスドライバレジストレーション関数です。この関数は、スタートアップ時の HCDC 登録時にドライバレジストレーション構造体メンバ classcheck にコールバック関数として登録され、エニュメレーション動作のコンフィグレーションディスクリプタ受信時に呼出されます。
チェック結果が OK の場合、ディスクリプタチェック結果（table[3]）に USB_DONE を、チェック結果が NG の場合は USB_ERROR を設定して終了します。
ペリフェラルデバイスのコンフィグレーションディスクリプタからエンドポイントディスクリプタを参照し、パイプ情報テーブル編集及び、使用するパイプ情報のチェックを行います。

補足

―

使用例

```
void  usb_hcdc_registration(USB_UTR_t *ptr)
{
  USB_HCDREG_t  driver;

  driver.ifclass      = (uint16_t)USB_IFCLS_CDCC;
  (省略)
  driver.classcheck   = (USB_CB_CHECK_t)&R_usb_hcdc_class_check;
  (省略)
  driver.devresume = (USB_CB_INFO_t)&usb_hcdc_dummy_function;
  R_usb_hstd_DriverRegistration(ptr, (USB_HCDREG_t*)&driver);
}
```

7.3.2 R_usb_hcdc_SetPipeRegistration

USB H/W パイプ設定処理

形式

USB_ER_t R_usb_hcdc_SetPipeRegistration(USB_UTR_t *ptr, uint16_t dev_addr)

引数

*ptr	USB 通信用構造体
dev_addr	デバイスアドレス

戻り値

－	処理結果(USB_E_OK/USB_E_ERROR)

解説

USB CDC 通信で使用するエンドポイントに対応した通信パイプの設定を、H/W に行います。
HCDC ではデータ通信用にバルク IN,バルク OUT の 2 つのパイプに加え、シリアルステートを受信する為にインタラプト IN パイプの合計 3 本のパイプ設定を行います。

補足

1. 本 API はユーザアプリケーションプログラムまたはクラスドライバから呼び出してください。
2. USB 通信用構造体 USB_UTR_t の以下のメンバ設定が必要です。
   USB_REGADR_t ipp : USB IP のアドレス
   uint16_t ip : USB IP 番号
3. 第 2 引数には自動変数(スタック)領域以外の領域を指定してください。

使用例

```
void usb_hcdc_smp_open(USB_UTR_t *ptr, uint16_t devadr, uint16_t data2)
{
  USB_ER_t      err;

  if (devadr != 0)
  {
    usb_shcdc_devadr = devadr;     /* Device Address store */

    /* Host CDC Pipe Registration */
    err = R_usb_hcdc_SetPipeRegistration(ptr, usb_shcdc_devadr);
    if (err != USB_OK)
    {
       USB_PRINTF0("Pipe Registration error !¥n");
    }
  }
}
```

7.3.3 R_usb_hcdc_class_request

CDC クラスリクエスト処理

形式

USB_ER_t R_usb_hcdc_class_request(void *pram)

引数

*pram クラスリクエストパラメータ

戻り値

－ エラーコード(USB_E_OK/USB_E_ERROR)

解説

CDC クラスリクエスト処理を行います。
HCDC が USB ペリフェラル CDC へ要求することが出来るクラスリクエストを以下に記します。

(1). SendEncapsulatedCommand
(2). GetEncapsulatedResponse
(3). SetCommFeature
(4). GetCommFeature
(5). ClearCommFeature
(6). SetLineCoding
(7). GetLineCoding
(8). SetControlLineState
(9). SendBreak

クラスリクエストの発行方法の詳細については、”使用例”を参照してください。

引数には、USB_HCDC_ClassRequestParm_構造体を、(void *)にキャスト指定してください。

USB_HCDC_ClassRequestParm 構造体については、Table7.8を参照してください。

補足

1. 本 API はユーザアプリケーションプログラムまたはクラスドライバで呼び出してください。
2. USB 送信処理結果はコールバック関数の引数”USB_UTR_t *”で得られます。
3. USB Basic Firmware アプリケーションノートの USB 通信用構造体（USB_UTR_t 構造体）を参照して下さい。

使用例

- ***SetEncapsulatedResponse***

```
{
   USB_ER_t    err;
   USB_HCDC_ClassRequest_UTR_t    utr_req;

   utr_req.parm.Encapsulated.p_data = p_data; /* Command data buffer */
   utr_req.parm.Encapsulated.wLength = length;
   utr_req.bRequestCode = USB_HCDC_SEND_ENCAPSULATED_COMMAND;
   utr_req.complete = smp_sendencapsulateresponse_cb;
   utr_req.devadr = devadr;              /* Device Address */
   utr_req.ip = USB_USBIP_0;             /* USB IP No (0/1) */
   utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );  /* USB IP address */

   /* CDC class request */
   err = R_usb_hcdc_class_request( (void*)&utr_req);

   return err;
}
/* Callback function */
void smp_sendencapsulateresponse_cb(USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
 /* クラスリクエスト終了時に行う処理を記述して下さい。 */
}
```

- ***GetEncapsulatedResponse***

```
{
  USB_HCDC_ClassRequest_UTR_t   utr_req; /* Line Coding Parameter */
  usb_er_t    err;

  /* Example of usage. */
  USB_ER_t    err;
  USB_HCDC_ClassRequest_UTR_t    utr_req;

  utr_req.parm.Encapsulated.p_data = p_data; /* Command data buffer */
  utr_req.parm.Encapsulated.wLength = length;
  utr_req.bRequestCode = USB_HCDC_GET_ENACAPSULATED_RESPONSE;
  utr_req.complete = smp_getencapsulateresponse_cb; /* Callback function */
  utr_req.devadr = devadr;     /* USB device address */
  utr_req.ip = USB_USBIP_0;    /* USB IP No (0/1)*/
  utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 ); /* USB IP address */

  /* CDC class request */
 err = R_usb_hcdc_class_request( (void*)&utr_req);
 return err;
}
/* Callback function */
void smp_getencapsulateresponse_cb(USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
 /* クラスリクエスト終了時に行う処理を記述して下さい。 */
}
```

- ***SetCommFeature***

```
{
    USB_HCDC_ClassRequest_UTR_t    utr_req;

    utr_req.bRequestCode = USB_HCDC_SET_COMM_FEATURE;
    utr_req.selector = selector; /* Feature Selector */
    utr_req.parm.CommFeature = p_commfeature; /* Feature Parameter set data */
    if( selector == USB_HCDC_ABSTRACT_STATE )
```

```
    {
       p_commfeature->abstractState.rsv = 0;
    }
    utr_req.complete = (USB_CB_t)&smp_setcommfeature_cb;
    utr_req.devadr = devadr;      /* Device Address */
    utr_req.ip = USB_USBIP_0;     /* USB IP No */
    utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );/* USB IP address */

    /* CDC class request */
    err = R_usb_hcdc_class_request( (void*)&utr_req);/

    return err;
}
/* Callback function for sending SetLineCoding class request */
void smp_setcommfeature_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
     /* クラスリクエスト終了時に行う処理を記述して下さい。 */
}
```

- ***GetCommFeature***

```
{
    USB_ER_t    err;
    USB_HCDC_ClassRequest_UTR_t    utr_req;

    utr_req.bRequestCode = USB_HCDC_GET_COMM_FEATURE;
    /* Feature Parameter storage address */
    utr_req.parm.CommFeature = p_commfeature;
    utr_req.complete = (USB_CB_t)& smp_getcommfeature_cb;
    utr_req.devadr = devadr;      /* Device Address */
    utr_req.selector = selector;  /* Feature Selector */
    utr_req.ip = USB_USBIP_0;     /* USB IP No */
    utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );/* USB IP address */

    /* CDC class request */
    err = R_usb_hcdc_class_request( (void*)&utr_req);

    return err;
}
/* Callback function for sending SetLineCoding class request */
void smp_getcommfeature_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
     /* クラスリクエスト終了時に行う処理を記述して下さい。 */
}
```

- ***ClearCommFeature***

```
{
   USB_ER_t    err;
   USB_HCDC_ClassRequest_UTR_t    utr_req;

   utr_req.bRequestCode = USB_HCDC_CLR_COMM_FEATURE;
   utr_req.complete = (USB_CB_t)&smp_clrcommfeature_cb;
   utr_req.devadr = devadr;        /* Device Address */
   utr_req.selector = selector;    /* Feature Selector */
   utr_req.ip = USB_USBIP_0;       /* USB IP No */
   utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );/* USB IP address */

   /* CDC class request */
   err = R_usb_hcdc_class_request( (void*)&utr_req);

   return err;
}
```

```
/* Callback function */
void smp_clrcommfeature_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
      /* クラスリクエスト終了時に行う処理を記述して下さい。 */
}
```

- ***SetLineCoding***

```
static   USB_HCDC_LineCoding_t   usb_shcdc_line_coding;

{
   USB_ER_t   err;
   USB_HCDC_ClassRequest_UTR_t     utr_req; /* Line Coding Parameter */

   usb_shcdc_line_coding.dwDTERate    = USB_HCDC_SPEED_9600;
   usb_shcdc_line_coding.bDataBits    = USB_HCDC_DATA_BIT_8;
   usb_shcdc_line_coding.bCharFormat  = USB_HCDC_STOP_BIT_1;
   usb_shcdc_line_coding.bParityType  = USB_HCDC_PARITY_BIT_NONE;

   utr_req.bRequestCode = USB_HCDC_SET_LINE_CODING;
   utr_req.complete = (USB_CB_t)&smp_setlinecoding_cb;
   utr_req.parm.LineCoding = &usb_shcdc_line_coding;
   utr_req.devadr = devadr;
   utr_req.ip = USB_USBIP_0;    /* USB IP No */;
   utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );/* USB IP address */

   /* CDC Class Request */
   err = R_usb_hcdc_class_request( (void*)&utr_req );
   return err;
}
/* Callback function */
void smp_setlinecoding_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
      /* クラスリクエスト終了時に行う処理を記述して下さい。 */
}
```

- ***GetLineCoding***

```
{
     USB_ER_t    err;
     USB_HCDC_ClassRequest_UTR_t     utr_req;

     utr_req.bRequestCode = USB_HCDC_GET_LINE_CODING;
     utr_req.parm.LineCoding = p_linecoding;   /* Line Coding table address */
     utr_req.complete = smp_getlinecoding_cb;
     utr_req.devadr = devadr;      /* Device Address */
     utr_req.ip = USB_USBIP_0;     /* USB IP No */
     utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );  /* USB IP address */

     /* CDC class request */
     err = R_usb_hcdc_class_request( (void*)&utr_req);

     return err;
}
/* Callback function for sending SetLineCoding class request  */
void smp_getlinecoding_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
      /* クラスリクエスト終了時に行う処理を記述して下さい。 */
}
```

- ***SetControlLineState***

```
{
   USB_ER_t    err;
   USB_HCDC_ClassRequest_UTR_t    utr_req;

   utr_req.bRequestCode = USB_HCDC_SET_CONTROL_LINE_STATE;
   utr_req.parm.ControlLineState.bDTR = dtr;    /* RS232 signal DTR */
   utr_req.parm.ControlLineState.bRTS = rts;    /* RS232 signal RTS */
   utr_req.complete = (USB_CB_t)smp_setcontrollinestate_cb;
   utr_req.devadr = devadr;   /* Device Address */
   utr_req.ip = USB_USBIP_0;  /* USB IP No */
   utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );     /* USB IP address */

   /* CDC class request */
   err = R_usb_hcdc_class_request( (void*)&utr_req);

   return err;
}
/* Callback function */
void smp_setcontrollinestate_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
      /* クラスリクエスト終了時に行う処理を記述して下さい。 */
}
```

- ***SendBreak***

```
{
   USB_ER_t    err;
   USB_HCDC_ClassRequest_UTR_t    utr_req;

   utr_req.bRequestCode = USB_HCDC_SEND_BREAK;
   /* Break Signal output time */
   utr_req.parm.BreakDuration.wTime_ms = time_ms;
   utr_req.complete = (USB_CB_t)smp_sendbreak_cb;
   utr_req.devadr = devadr;         /* Device Address */
   utr_req.ip = USB_USBIP_0;        /* USB IP No */
   utr_req.ipp = R_usb_cstd_GetUsbIpAdr( USB_USBIP_0 );     /* USB IP address */

   /* CDC class request */
   err = R_usb_hcdc_class_request( (void*)&utr_req);

   return err;
}
/* Callback function */
void smp_sendbreak_cb (USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
      /* クラスリクエスト終了時に行う処理を記述して下さい。 */
}
```

7.3.4 R_usb_hcdc_send_data

USB 送信処理

形式

USB_ER_t R_usb_hcdc_send_data(USB_UTR_t *ptr, uint8_t *buf, uint32_t size, USB_CB_t complete)

引数

ptr	USB 通信用構造体
buf	転送データアドレス
size	転送サイズ
complete	処理完了通知コールバック関数

戻り値

－	エラーコード（USB_E_OK/USB_E_ERROR）

解説

転送データアドレス buf で指定されたアドレスから、転送サイズ size 分のデータを USB 送信します。送信完了後、コールバック関数 complete が呼出されます。

補足

1. 本 API はユーザアプリケーションプログラムまたはクラスドライバで呼び出してください。
2. USB 通信用構造体 USB_UTR_t の以下のメンバ設定が必要です。
   USB_REGADR_t ipp : USB IP のアドレス
   uint16_t ip : USB IP 番号
3. 第 2 引数には自動変数(スタック)領域以外の領域を指定してください。
4. r_usb_hcdc_config.h ファイル内で USB_HCDC_MULTI_CONNECT 定義を有効にしている時は、USB 通信用構造体 USB_UTR_t のメンバ keyword に対しデータ送信を行う CDC デバイスのデバイスアドレスを設定してください。
5. USB 送信処理結果はコールバック関数の引数”USB_UTR_t *”で得られます。
6. USB Basic Firmware アプリケーションノートの USB 通信用構造体（USB_UTR_t 構造体）を参照して下さい。

使用例

```
{
  USB_UTR_t  *ptr;
  uint16_t   size = 5;                              /* USB 送信データ数 */

  ptr = (USB_UTR_t *)&utr;
  ptr->ip = USB_HOST_USBIP_NUM;                     /* USB IP 番号設定 */
  ptr->ipp = R_usb_cstd_GetUsbIpAdr( ptr->ip );  /* USB IP ベースアドレス取得 */

  R_usb_hcdc_send_data(ptr, send_data, size, (USB_CB_t)&usb_complete);
}

/* USB 送信完了通知用コールバック関数 */
void  usb_complete( USB_UTR_t *mess, uint16_t data1, uint16_t data2 )
{
  /* USB 送信完了時の処理を記述して下さい。 */
}
```

7.3.5 R_usb_hcdc_receive_data

USB 受信処理

形式

void	R_usb_hcdc_receive_data (USB_UTR_t *ptr, uint8_t *buf, uint32_t size, USB_CB_t complete)

引数

ptr	USB 通信用構造体
buf	転送データアドレス
size	転送サイズ
complete	処理完了通知コールバック関数

戻り値

―	―

解説

USB 受信要求を行います。
USB から転送サイズ size 分のデータ受信完了、又は MAX パケットサイズ未満のデータを受信した場合、コールバック関数 complete が呼出されます。
USB 受信データは、転送データアドレス buf で指定されたアドレスで指定された領域に格納されます。

補足

1. 本 API はユーザアプリケーションプログラムまたはクラスドライバで呼び出してください。
2. USB 通信用構造体 USB_UTR_t の以下のメンバ設定を行ってください。
   USB_REGADR_t ipp : USB IP のアドレス
   uint16_t ip : USB IP 番号
3. 第 2 引数には自動変数(スタック)領域以外の領域を指定してください。
4. 受信したデータが MaxPacketSize の n 倍、かつ引数 size に指定した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関数 complete は発生しません。
5. r_usb_hcdc_config.h ファイル内で USB_HCDC_MULTI_CONNECT 定義を有効にしている時は、USB 通信用構造体 USB_UTR_t のメンバ keyword に対しデータ受信を行う CDC デバイスのデバイスアドレスを設定してください。
6. USB 受信処理結果はコールバック関数の引数”USB_UTR_t *”で得られます。
7. USB Basic Firmware アプリケーションノートの USB 通信用構造体（USB_UTR_t 構造体）を参照して下さい。

使用例

```
{
  USB_UTR_t  *ptr;
  uint16_t   size = 64;                          /* USB 受信要求サイズ */

  ptr = (USB_UTR_t *)&utr;
  ptr->ip = USB_HOST_USBIP_NUM;                  /* USB IP 番号設定 */
  ptr->ipp = R_usb_cstd_GetUsbIpAdr( ptr->ip );  /* USB IP ベースアドレス取得 */

  R_usb_hcdc_receive_data(ptr, receive_data, size, (USB_CB_t)&usb_complete);
}

/* USB 受信完了通知用コールバック関数 */
void  usb_complete( USB_UTR_t *mess, uint16_t data1, uint16_t data2 )
{
  /* USB 受信完了時の処理を記述して下さい。 */
}
```

7.3.6 R_usb_hcdc_serial_state_trans

クラスノーティフィケーション **SerialState** 受信処理

形式

USB_ER_t	R_usb_hcdc_serial_state_trans(USB_UTR_t *ptr, USB_HCDC_SERIAL_ST_CB_t *complete)

引数

*ptr	USB 通信用構造体
complete	処理完了通知コールバック関数

戻り値

－	エラーコード（USB_E_OK/ USB_E_ERROR）

解説

CDC クラスノーティフィケーション・シリアルステートをペリフェラルデバイスから受信します。
受信完了後、コールバック関数 complete が呼出されます。
コールバック関数で受信したシリアルステータスを取出して下さい。

補足

1. 本 API はユーザアプリケーションプログラムまたはクラスドライバで呼び出してください。
2. シリアルステータスのビットパターンは”6.3 クラスノーティフィケーション（デバイス→ホストへの通知）”の”Table6.17　UART State ビットマップフォーマット”を参照下さい。
3. USB 受信処理結果はコールバック関数の引数”USB_UTR_t *”で得られます。
4. USB 通信用構造体 USB_UTR_t の以下のメンバ設定を行ってください。

USB_REGADR_t	ipp	: USB IP のアドレス
uint16_t	ip	: USB IP 番号

5. r_usb_hcdc_config.h ファイル内で USB_HCDC_MULTI_CONNECT 定義を有効にしている時は、USB 通信用構造体 USB_UTR_t のメンバ keyword に対し SerialState 受信を行う CDC デバイスのデバイスアドレスを設定してください。

使用例

```
void usb_hcdc_main_task(USB_VP_INT stacd)
{
  USB_UTR_t             *mess;
  USB_ER_t              err;

  while (1)
  {
  err = R_USB_RCV_MSG(USB_HCDCSMP_MBX,(USB_MSG_t**)&mess);
  if (err == USB_OK)
  {
    err = R_usb_hcdc_serial_state_trans( mess,
          (USB_HCDC_SERIAL_ST_CB_t *)&usb_hcdc_smp_SerialStateReceive );
    if( err != USB_OK )
    {
       USB_PRINTF0("### usb_pcdc_MainTask function bulk read error¥n");
    }
  }
}


/* R_usb_hcdc_serial_state_trans のコールバック関数例 */
void usb_hcdc_smp_SerialStateReceive(USB_UTR_t *mess, uint16_t data1, uint16_t
data2)
{
  uint16_t *status;
  uint16_t msginfo;

  if (mess->result == USB_OK)
  {
    /* Command set */
    msginfo = USB_HCDC_CMD_RCV_SERIAL_STATE;
  }
  else
  {
    /* Command set */
    msginfo = USB_HCDC_CMD_RCV_SERIAL_STATE_NG;
  }
  status = (uint16_t *)mess->tranadr;      /* Status set */
  /* [0] bmRequestType/bRequest */
  /* [1] wValue */
  /* [2] wIndex */
  /* [3] wLength :2 */
  /* [4] data : Serial State(UART State bitmap) */
  usb_hcdc_smp_message_send( mess, msginfo, 0, status[4]);
}
```

7.3.7 R_usb_hcdc_driver_start

Host CDC ドライバタスクスタート設定

形式

void	R_usb_hcdc_driver_start(void)

引数

―	―

戻り値

―	―

解説

本 API は、Host CDC ドライバタスクのスタート設定を行います。

補足

1. 本 API は初期化時に、ユーザアプリケーションプログラムで呼び出してください。

使用例

```
void usb_hcdc_task_start( void )
{
            :
  ptr->ipp = R_usb_cstd_GetUsbIpAdr( ptr->ip );
  R_usb_hstd_usbdriver_start( ptr );  /* Host USB Driver Start Setting */
  usb_hcdc_registration( ptr );     /* Host Application Registration */
  usb_hstd_HubRegistAll(ptr);       /* Hub registration */

  R_usb_hcdc_driver_start( ptr );   /* Host Class Driver Task Start Setting */
  usb_hapl_task_start( ptr );      /* Host Application Task Start Setting */
  R_usb_cstd_UsbIpInit( ptr, USB_HOST_PP );   /* Initialize USB IP */
            :
}
```

7.3.8 R_usb_hcdc_task

HCDC タスク

形式

void	R_usb_hcdc_task(USB_VP_INT_t stacd)

引数

stacd	タスクスタートコード（未使用）

戻り値

―	―

解説

HCDC 処理タスク。
アプリから要求された処理を行い、アプリに処理結果を通知します。

補足

1. 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。

使用例

```
void usb_apl_task_switch(void)
{
  while( 1 )
  {
    /* Scheduler */
    R_usb_cstd_Scheduler();

    if( USB_FLGSET == R_usb_cstd_CheckSchedule() )
    {
       R_usb_hstd_HcdTask((USB_VP_INT)0);         /* HCD Task */
       R_usb_hstd_MgrTask((USB_VP_INT)0);         /* MGR Task */
       R_usb_hhub_Task((USB_VP_INT)0);            /* HUB Task */
       usb_hcdc_main_task((USB_VP_INT)0);         /* HCDC Application Task */

       R_usb_hcdc_task((USB_VP_INT)0);            /* HCDC Task */

    }
    else
    {
       /* Idle Task (sleep sample) */
       R_usb_cstd_IdleTask(0);
    }
  }
}
```

8. サンプルアプリケーション

8.1 アプリケーション仕様

HCDC のサンプルアプリケーション(以降、APL) の主な機能を以下に示します。

1. CDC デバイスに対し、受信要求（Bulk In 転送）を行い、受信データを取得します。
2. Bulk Out 転送により受信データを CDC デバイスへ送信します（ループバック）。
3. クラスリクエスト SET_CONTROL_LINE_STATE で RTS、DTR の設定を行います。
4. 信速度等の設定は、クラスリクエスト SET_LINE_CODING を CDC デバイスに送信することにより行います。このクラスリクエストにより通信速度、データビット数、ストップビット長、パリティビットの設定を行えます。
5. クラスリクエスト GET_LINE_CODING を CDC デバイスに送信することにより、CDC デバイスの通信設定値を取得できます。

8.1.1 データ転送イメージ

データ転送イメージをFigure 8-1に示します。

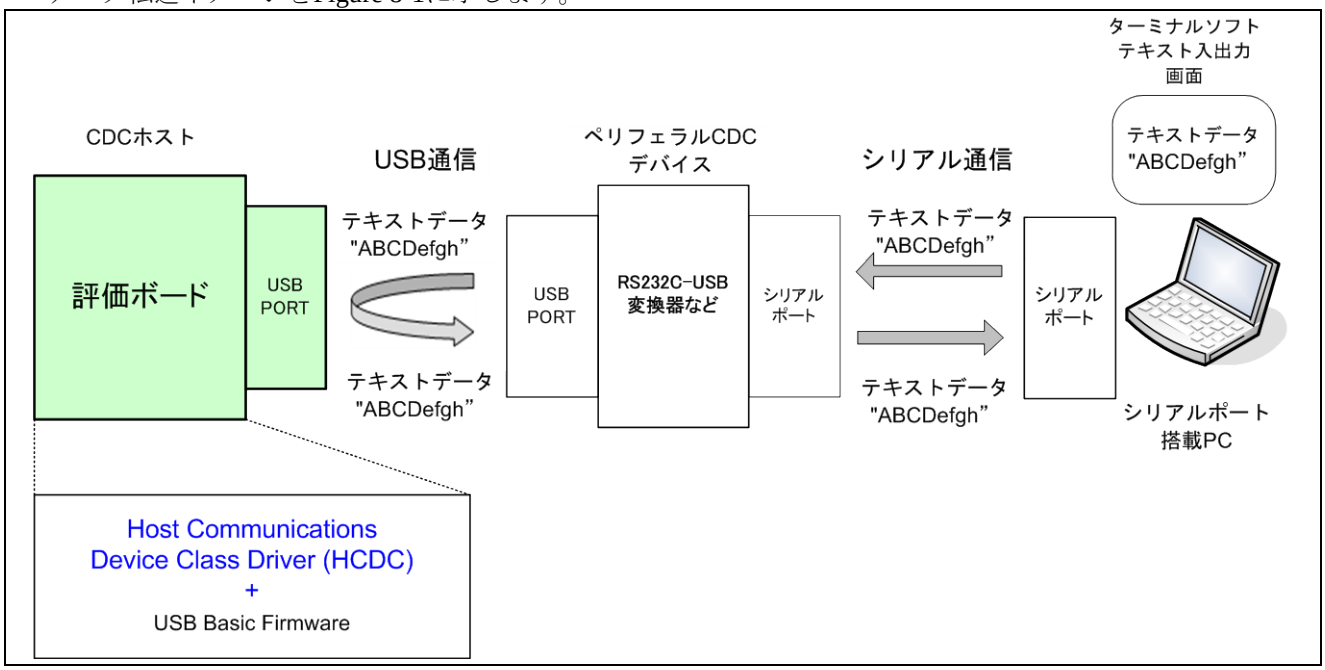

Figure 8-1 データ転送 (ループバック通信) イメージ

8.1.2 ボーレート設定

接続された CDC デバイスボーレート設定は common¥inc¥r_usb_hcdc_apl.h ファイル内の "INIT_COM_SPEED"定義に対しボーレートの設定をお願いします。1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200bps のいずれかを指定してください。
例)

```
#define  INIT_COM_SPEED  USB_HCDC_SPEED_57600
```

8.2 アプリケーション処理概要

APL は、初期設定、メインループの 2 つの部分から構成されます。以下にそれぞれの処理概要を示します。

8.2.1 初期設定

初期設定では、USB コントローラの初期設定およびアプリケーションプログラムの初期化処理を行います。

8.2.2 メインループ

このメインループでは、CDC デバイスから受信したデータをそのまま CDC デバイスへ送信するループバック処理をメインに行います。以下にメインループの処理概要を示します。

1. GENMAI に CDC デバイスが ATTACH され、Enumeration 完了後に R_USB_GetEvent 関数をコールすると戻り値に USB_STS_CONFIGURED がセットされます。APL では、USB_STS_CONFIGURED を確認するとクラスリクエスト SET_LINECODING を CDC デバイスに送信します。
2. クラスリクエスト処理の完了を確認すると R_USB_Read 関数をコールし、CDC デバイスから送信されるデータのデータ受信要求を行います。なお、データ受信要求のほか、CDC デバイスからの Class Notification 受信要求も行います。
3. CDC デバイスからのデータ受信が完了し、R_USB_GetEvent 関数をコールすると戻り値に USB_STS_READ_COMPLETE がセットされます。受信したデータは外部変数 g_data に格納されています。受信データサイズは、usb_ctrl_t 構造体のメンバ size により確認できます。APL では、メンバ size が 0(ゼロ)の場合、Null パケット受信と判断し、CDC デバイスに対し、再度データ受信要求を行います。メンバ size が 0(ゼロ)以外の場合、CDC デバイスからの送信データを受信したと判断します。受信したデータは CDC デバイスに対し、データ送信要求が行われます。
4. CDC デバイスへのデータ送信が完了し、R_USB_GetEvent 関数をコールすると戻り値に USB_STS_WRITE_COMPLETE がセットされます。APL では、USB_STS_WRITE_COMPLETE を確認すると R_USB_Read 関数をコールし、CDC デバイスから送信されるデータのデータ受信要求を行います。
5. 上記3と4の処理が繰り返し行われます。

Figure 8-2に、APL の処理概要を示します。

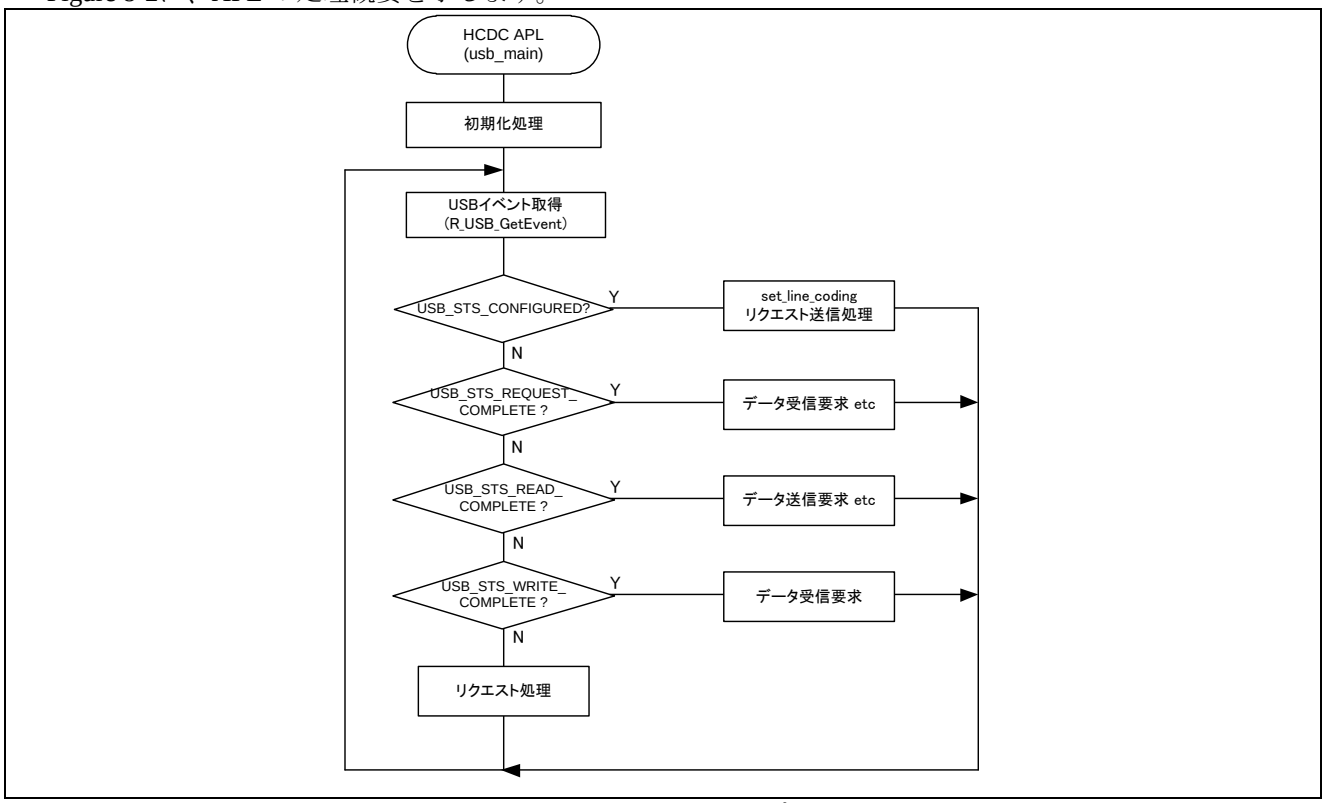

Figure 8-2 メインループ処理

ホームページとサポート窓口

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

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

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2016.09.01	—	初版発行
1.10	2016.09.30	—	USB-BASIC-F/W が改定されたため、バージョンアップ

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、本文を参照してください。なお、本マニュアルの本文と異なる記載がある場合は、本文の記載が優先するものとします。

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. リザーブアドレスのアクセス禁止

   【注意】リザーブアドレス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、事前に問題ないことをご確認ください。

   同じグループのマイコンでも型名が違うと、内部メモリ、レイアウトパターンの相違などにより、特性が異なる場合があります。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。

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/