Renesas USB MCU

R01AN0273JJ0220
Rev.2.20
Sep 30, 2015

USB Peripheral Communications Device Class Driver (PCDC)

要旨

本アプリケーションノートは、USB Peripheral コミュニケーションデバイスクラスドライバ（PCDC）について説明します。本ドライバは USB Basic Peripheral Driver (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 PCDC と称します。

対象デバイス

RX62N/RX621 グループ
RX630 グループ
RX63T 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
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. RX62N/RX621 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.R01UH0033JJ)
5. RX630 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.R01UH0040JJ)
6. RX63T 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.R01UH0238JJ)
7. USB Basic Host and Peripheral Driver アプリケーションノート(ドキュメント No. R01AN0512JJ)

— 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

PCDC は、USB-BASIC-FWと組み合わせることで、USB Peripheral コミュニケーションデバイスクラスドライバ（以降 PCDC と記述）として動作します。PCDC 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 Abstract Control Model に準拠し、USB ホスト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。

- USB ホストとのデータ転送
- CDC クラスリクエストに応答
- コミュニケーションデバイスクラスノーティフィケーション送信サービスの提供

1.1 動作確認済環境

HCDC の動作確認済環境を以下に示します。

＜評価ボード＞

Renesas Starter Kit+ for RX62N（RSK+RX62N): 型名: R0K5562N0C001BR
RX62N グループ ルネサスマイコン開発スタータキット　ルネサスエレクトロニクス製

Renesas Starter Kit for RX630（RSKRX630): 型名: R0K505630C001BR
RX630 グループ ルネサスマイコン開発スタータキット　ルネサスエレクトロニクス製

Renesas Starter Kit for RX63T（RSKRX63T): 型名: R0K5563THC010BR
RX63T グループ ルネサスマイコン開発スタータキット　ルネサスエレクトロニクス製

＜開発環境＞

a) 統合環境 e² studio　ルネサスエレクトロニクス製
b) RX ファミリ用 C/C++コンパイラパッケージ　Ver2.03.00　ルネサスエレクトロニクス製
c) E1 エミュレータまたは E20 エミュレータ　ルネサスエレクトロニクス製

＜その他＞

a) CDC ホスト(PC: Windows® 7、Windows® 8、Windows® 8.1、Windows® 10)
b) エミュレータ用ホスト PC (Windows® 7、Windows® 8、Windows® 8.1）
c) USB ケーブル
d) ユーザケーブル（E1 エミュレータまたは E20 エミュレータに同梱）
e) エミュレータ用ケーブル（E1 エミュレータまたは E20 エミュレータに同梱）

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。

ACM	:	Abstract Control Model.
APL	:	Application program
CDC	:	Communications Devices Class
CDCC	:	Communications Devices Class Communications Interface Class
CDCD	:	Communications Devices Class Data Class Interface
CPD	:	Serial Communication Port Driver
cstd	:	Peripheral & Host USB-BASIC-FW用の関数およびファイルのプレフィックス
H/W	:	Renesas USB device
non-OS	:	USB basic firmware for OS less system
PCD	:	Peripheral control driver of USB-BASIC-FW
PCDC	:	Periperal用 Communications Devices Class
PCDCD	:	Periperal Communications Devices Class Driver
PDCD	:	Peripheral device class driver (device driver and USB class driver)
PP	:	プリプロセス定義
pstd	:	Peripheral USB-BASIC-FW用の関数およびファイルのプレフィックス
PSTN	:	Public Switched Telephone Network, contains the ACM (above) standard.
RSK	:	Renesas Starter Kits
USB	:	Universal Serial Bus
USB-BASIC-FW	:	USB Basic Peripheral Driver for Renesas USB MCU (non-OS)
タスク	:	処理の単位
スケジューラ	:	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
スケジューラマクロ	:	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

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。

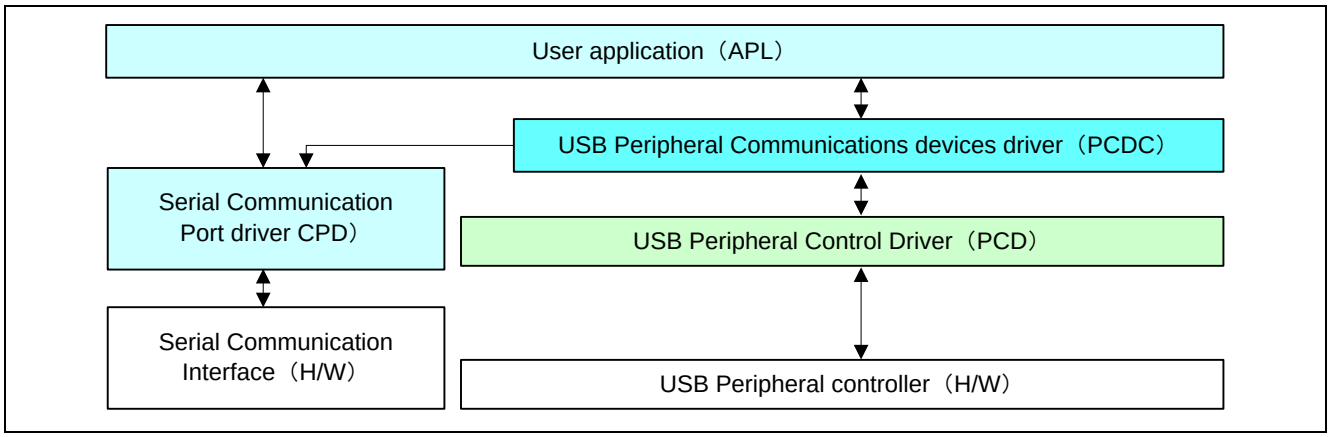


Figure 2-1 モジュール構成図

Table 2-1 各モジュール機能概要

モジュール名	機能概要
PCDC	APL からの CDC に関するリクエストおよび、データ通信を PCD へ要求します。
PCD	USB Peripheral H/W 制御ドライバです。
CPD	シリアルポートの制御ドライバです。

3. API 情報

本ドライバの API はルネサスの API の命名基準に従っています。

3.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- USB

3.2 サポートされているツールチェイン

このドライバは下記ツールチェインで動作確認を行っています。

- Renesas RX Toolchain v.2.03.00

3.3 ヘッダファイル

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

3.4 整数型

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

3.5 コンパイル時の設定

本ドライバを使用する場合、USB-BASIC-FWをペリフェラルとして設定する必要があります。USB-BASIC-FWの設定は、USB Basic Host and Peripheral Driver アプリケーションノート(Document No.R01AN0512JJ)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_pcdc_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_pcdc_config.h	
USB_PCDC_USE_PIPE_IN USB_PCDC_USE_PIPE_OUT	データ転送で使用されるパイプ番号を指定してください。(USB_PIPE1 から USB_PIPE5 のうちいずれかを指定してください。)
USB_PCDC_USE_PIPE_STATUS	クラスノーティフィケーション通知で使用されるパイプ番号を指定してください。(USB_PIPE6 から USB_PIPE9 のうちいずれかを指定してください。)
USB_UART_ENABLE	UART を使用する場合に、本定義を有効にしてください。

3.6 引数

API 関数の引数である構造体を示します。詳細は USB Basic Host and Peripheral Driver アプリケーションノート(Document No.R01AN0512JJ)を参照してください。

```
typedef struct USB_UTR
{
        USB_MH_t        msghead;                /* OS が使用するメッセージヘッダ/使用不可 */
        uint16_t        msginfo;                /* USB-BASIC-FWが使用するメッセージ情報*/
        uint16_t        keyword;                /* サブ情報（ポート番号、パイプ番号等） */
        union {
                USB_REGADR_t        ipp;        /* USB IP アドレス(USBb モジュール用)*/
                USB_REGADR1_t       ipp1;       /* USB IP アドレス(USBA/USBAa モジュール用) */
        };
        uint16_t        ip;                     /* USBIP 番号*/
        uint16_t        result;                 /* USB 通信結果 */
        USB_CB_t        complete;               /* コールバック関数 */
        void            *tranadr;               /* USB 通信バッファアドレス */
        uint32_t        tranlen;                /* USB 通信データ長 */
        uint16_t        *setup;                 /* Setup パケットデータ */
        uint16_t        status;                 /* USB 通信ステータス */
        uint16_t        pipectr;                /* PIPECTR レジスタ */
        uint8_t         errcnt;                 /* 転送中のエラー発生回数 */
        uint8_t         segment;                /* セグメント情報（データ通信の継続又は終了） */
        void            *usr_data;              /* 各種情報を格納 */
} USB_UTR_t;
```

4. デバイスクラスドライバの登録

作成したデバイスクラスドライバは、USB ドライバに登録することでデバイスクラスドライバとして機能します。

詳細は、USB Basic Host and Periphral Driver アプリケーションノート(Document No.R01AN0512JJ)の「ペリフェラルクラスドライバの登録方法」を参照してください。

5. システム資源

Table 5-1～Table 5-3に、PCDC が使用している non-OS 資源を示します。

Table 5-1 タスク情報

関数名	タスク ID	優先度	概要
usb_pcdc_Task	USB_PCDC_TSK	USB_PRI_3	PCDC タスク

Table 5-2 メールボックス情報

メールボックス名	使用タスク ID	待ちタスクキュー	概要
USB_PCDC_MBX	USB_PCDC_TSK	FIFO 順	PCDC 用メールボックス

Table 5-3 メモリプール情報

メモリプール名	待ちタスクキュー	メモリブロック（注）	概要
USB_PCDC_MPL	FIFO 順	40byte	PCDC 用固定長メモリプール

（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。

6. タスク ID 設定と優先度設定

ユーザタスク ID 等の設定値は以下の範囲内で定義してください。なお、タスク優先度設定は、R_usb_cstd_SetTaskPri(API)を使って設定してください。

タスク ID : 5 ～ (USB_IDMAX - 1)
メールボックス ID : タスク ID と同じ値を設定してください
メモリプール ID : タスク ID と同じ値を設定してください
タスク優先度 : 6 ～ (USB_PRIMAX - 1)

[Note]

1. USB_IDMAX と USB_PRIMAX は、ユーザによって r_usb_config.h 内に定義される値です。
2. ユーザタスク等の ID 番号は、アプリケーションファイル内で定義してください。

7. コミュニケーションデバイスクラス（CDC）、PSTN and ACM

7.1 基本機能

CDC 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（7.2章参照）に準拠しています。
CDC の主な機能を以下に示します。

1. USB ホストからの機能照会に対する応答
2. USB ホストからのクラスリクエストに対する応答
3. USB ホストとのデータ通信
4. USB ホストへのシリアル通信エラー報告

7.2 Abstract Control Model 概要

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。
以下に本 S/W でサポートするクラスリクエスト・クラスノーティフィケーションを記します。

7.2.1 クラスリクエスト（ホスト→デバイスへの通知）

PCDC で対応しているクラスリクエストをTable 7-1に示します。

Table 7-1 CDC クラスリクエスト

リクエスト	コード	説明	対応
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。

7.2.2 クラスリクエストのデータフォーマット

CDC 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。

(1). SetLineCoding

UART 回線設定を行う為にホストがデバイスに対して送信するクラスリクエストです。
SetLineCoding データフォーマットを以下に示します。

Table 7-2 SetLineCoding フォーマット

bmRequestType t	bReques	wValue	wIndex	wLength	Data
0x21	SET_LINE_CODING (0x20)	0x00	0x00	0x07	Line Coding Structure Table 7-3 Line Coding Structure フォーマット参照

Table 7-3 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 bit 1 - 1.5 Stop bit 2 - 2 Stop bit
5	BparityType	1	Number	パリティ 0 - None 1 - Odd 2 - Even
6	BdataBits	1	Number	データビット （5、6、7、8）

(2). GetLineCoding

UART 回線設定状態を要求する為にホストがデバイスに対して送信するクラスリクエストです。
GetLineCoding データフォーマットを以下に示します。

Table 7-4 GetLineCoding フォーマット

bmRequestType t	bReques	wValue	wIndex	wLength	Data
0xA1	GET_LINE_CODING (0x21)	0x00	0x00	0x07	Line Coding Structure Table 7-3 Line Coding Structure フォーマット参照

(3). SetControlLineState

UART のフロー制御用信号を設定する為にホストがデバイスに対して送信するクラスリクエストです。
本 S/W では RTS/DTR の制御をサポートしていません。
SET_CONTROL_LINE_STATE データフォーマットを以下に示します。

Table 7-5 SET_CONTROL_LINE_STATE フォーマット

bmRequestType t	bReques	wValue	wIndex	wLength	Data
0x21	SET_CONTROL_ LINE_STATE (0x22)	Control Signal Bitmap Table 7-6 Control Signal Bitmap フォーマット参照	0x00	0x00	None

Table 7-6 Control Signal Bitmap フォーマット

Bit Position	Description
D15～D2	予約（0 にリセット）
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

7.2.3 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）

本 S/W のクラスノーティフィケーション対応/非対応をTable 7-7下表に示します。

Table 7-7 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	○

(1). SerialState

UART ポートに状態変化を検出した場合、ホストへ状態通知を行います。

本 S/W ではオーバーランエラー、パリティエラー、フレーミングエラー検出をサポートしています。状態通知は正常状態からエラー検出した場合に行います。エラーを連続検出しても状態通知を連続送信しません。

SerialState データフォーマットを以下に示します。

Table 7-8 SerialState フォーマット

bmRequestType t	bReques	wValue	wIndex	wLength	Data
0xA1	SERIAL_STATE (0x20)	0x00	0x00	0x02	UART State bitmap Table 7-9 UART State bitmap フォーマット参照

Table 7-9 UART State bitmap フォーマット

Bits	Field	Description	対応
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	フレーミングエラー検出	○
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	×

7.3 PC の仮想 COM ポートについて（参考）

Windows OS 搭載 PC は CDC デバイスを仮想 COM ポートとして利用することが可能です。

Windows OS 搭載 PC に本 S/W を実装した RSK ボードを接続すると、エニュメレーション設定に続き、CDC クラスリクエストの GetLineCoding 及び SetControlLineState を行った後、仮想 COM デバイスとしてデバイスマネージャに登録されます。Windows デバイスマネージャに仮想 COM ポートとして登録された後は、WindowsOS 標準搭載のハイパーターミナル等のターミナルアプリで CDC デバイスとデータ通信が可能です。

ターミナルアプリのシリアルポート設定を行うことで、クラスリクエスト SetLineCoding による UART 設定が可能です。ターミナルアプリのウインドウから入力したデータ（又はファイル送信）は EP2 を使用して RSK ボードへ転送され、RSK ボード側から PC へのデータ転送は EP1 を使用して行われます。

ターミナルアプリによっては最後に受信したデータが MAX パケットサイズ（Full-Speed:64Byte）の場合、継続するデータがあると判断して受信データをターミナルに表示しないことがあります。この場合、MAX パケットサイズ未満のデータを受信することで、それまでに受信したデータがターミナルに表示されます。

8. USB ペリフェラルコミュニケーションデバイスクラスドライバ（PCDC）

8.1 基本機能

PCDC の基本機能を以下に示します。

1. USB ホストとのデータ転送
2. CDC クラスリクエストに応答
3. コミュニケーションデバイスクラスノーティフィケーション送信サービスの提供

8.2 PCDC API 一覧

Table 8-1に PCDC API 一覧を示します。

Table 8-1 PCDC API 一覧

関数名	機能概要
R_usb_pcdc_SendData	USB 送信処理
R_usb_pcdc_ReceiveData	USB 受信処理
R_usb_pcdc_SerialStateNotification	USB ホスト装置へクラスノーティフィケーション SerialState を送信
R_usb_pcdc_usr_ctrl_trans_function	CDC 用コントロール転送処理
R_usb_pcdc_task	PCDC タスク
R_usb_pcdc_driver_start	PCDC ドライバ起動

8.2.1 R_usb_pcdc_SendData

USB 送信処理

形式

```
void    R_usb_pcdc_SendData( 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

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. 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
2. USB 送信処理結果はコールバック関数の引数”USB_UTR_t *mess”で得られます。
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. 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

```
   #define USB_PERI_USBIP_NUMUSB_USBIP_0
 // #define  USB_PERI_USBIP_NUMUSB_USBIP_1

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

   ptr = (USB_UTR_t *)&utr;
   ptr->ip = USB_PERI_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_pcdc_SendData(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。 */
 }
```

8.2.2 R_usb_pcdc_ReceiveData

USB 受信処理

形式

```
void    R_usb_pcdc_ReceiveData (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. 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

2. USB 受信処理結果はコールバック関数の引数”USB_UTR_t *mess”で得られます。
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. 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

```
   #define USB_PERI_USBIP_NUM USB_USBIP_0
// #define  USB_PERI_USBIP_NUM USB_USBIP_1

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

  ptr = (USB_UTR_t *)&utr;
  ptr->ip = USB_PERI_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_pcdc_ReceiveData(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。 */
}
```

8.2.3 R_usb_pcdc_SerialStateNotification

USB ホスト装置へクラスノーティフィケーション SerialState を送信する

形式

void	R_usb_pcdc_SerialStateNotification(USB_UTR_t *ptr, uint16_t serial_state, USB_CB_t complete)

引数

*ptr	USB 通信用構造体
serial_state	シリアルステータス
complete	処理完了通知コールバック関数

戻り値

―	―

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 USB ホストに送信します。
シリアルステート送信はインタラプトパイプ EP3 を使用します。
送信完了後、コールバック関数 complete が呼出されます。

補足

1. シリアルステータスのビットパターンは"Table 7-9　UART State bitmap フォーマット"を参照してください。
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. USB 送信処理結果はコールバック関数の引数”USB_UTR_t *mess”で得られます。
4. 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

```
  #define USB_PERI_USBIP_NUMUSB_USBIP_0
// #define  USB_PERI_USBIP_NUMUSB_USBIP_1

{
  USB_UTR_t  utr;
  USB_UTR_t  *ptr;
  uint16_t   state;       /* シリアルステータス */

  ptr = (USB_UTR_t *)&utr;
  ptr->ip = USB_PERI_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 */

  state = 0x0020;         /* D5:パリティエラー */
  R_usb_pcdc_SerialStateNotification(ptr, state, (USB_CB_t)&usb_complete);
}

/* シリアルステート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 )
{
  /* シリアルステート送信完了時の処理を記述して下さい。 */
}
```

8.2.4 R_usb_pcdc_usr_ctrl_trans_function

CDC 用コントロール転送処理

形式

void R_usb_pcdc_usr_ctrl_trans_function(USB_UTR_t *ptr, USB_REQUEST_t *preq, uint16_t ctsq)

引数

*ptr	USB 通信用構造体
*preq	クラスリクエストメッセージへのポインタ
ctsq	コントロール転送ステージ情報 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：Sequence error

戻り値

― ―

解説

リクエストタイプが CDC クラスリクエストの場合、コントロール転送ステージに対応した処理を呼び出します。

本 API をデバイスクラスドライバ・レジストレーションでコントロール転送時に呼出されるコールバック関数として USB_PCDREG_t 構造体メンバ *ctrltrans* に登録してください。

補足

―

使用例

```
USB_PCDREG_t  driver;

/* Control Transfer */
driver.ctrltrans = (USB_CB_TRN_t)&R_usb_pcdc_usr_ctrl_trans_function;
R_usb_pstd_DriverRegistration(ptr, &driver);
```

8.2.5 R_usb_pcdc_Task

PCDC タスク

形式

void	R_usb_pcdc_Task(USB_VP_INT_t stacd)

引数

stacd	タスクスタートコード（非使用）

戻り値

―	―

解説

PCDC 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. 本 API はユーザプログラムで呼び出してください。
2. non-OS で動作させる場合、タスクスイッチング処理から本 API がスケジュールされるように登録します。

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_pstd_PcdTask((USB_VP_INT)0);         /* PCD Task */
       /* Peripheral Communications Devices Class Task */
       R_usb_pcdc_Task(0);
       /* Peripheral Communications Class Application Task */
       usb_pcdc_main_task(0);
    }
  }
}
```

8.2.6 R_usb_pcdc_driver_start

PCDC ドライバ起動

形式

void R_usb_pcdc_driver_start(USB_UTR_t *ptr)

引数

*ptr USB 通信構造体

戻り値

― ―

解説

PCDC ドライバタスクの優先度を設定します。

優先度が設定されることで、メッセージの送受信が可能になります。

補足

1. 初期設定時にユーザアプリケーションで呼び出してください。
2. 本関数コール前に 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

使用例

```
void usb_apl( void )
{
  USB_UTR_t  *ptr;
    :
  ptr->ip = USB_PERI_USBIP_NUM;      /* USB IP番号設定 */
  ptr->ipp = R_usb_cstd_GetUsbIpAdr( ptr->ip );  /* USB IPアドレス設定 */
    :
  R_usb_pcdc_driver_start( ptr );/* Peripheral Class Driver Task Start Setting */
   :
}
```

9. サンプルアプリケーション

9.1 アプリケーション仕様

PCDC のサンプルアプリケーション(以降、APL)は、RSK 上に実装された液晶表示器(LCD) を使用します。APL の主な機能を以下に示します。

1) USB ループバックモード（エコーモード）(注 1)

USB ホストから受信したデータを、USB ホストへ送信します。

2) USB-シリアル変換モード (注 1) (注 2)

USB ホストから受信したデータを COM ポート(SCI7)に送信し、COM ポート(SCI7)から受信したデータを USB ホストに送信します。なお、COM ポートのエラー（パリティエラー、フレーミングエラー、オーバーランエラー）が発生した場合、USB ホストにクラスノーティフィケーション“SerialState”を通知します。

3) 消費電力低減機能

USB の状態に応じて MCU を消費電力低減モードに遷移させる機能です。この機能を有効にするには、“r_usb_config.h”ファイル内のマクロ定義“USB_CPU_LPW_PP”を“USB_LPWR_USE_PP”に設定します。

a) USB サスペンド状態時に MCU をスリープモードに遷移させます。
b) USB デタッチ（切断）状態時に、MCU をソフトウエアスタンバイモードに遷移させます。

（注1）USB ループバックモード/USB-シリアル変換モードの選択は、“r_usb_pcdc_apl.h”ファイル内のマクロ定義“CDC_APL_MODE_PP”の設定より行います。

a) USB ループバックモード を選択する場合

#define CDC_APL_MODE_PP ECHO_PP

b) USB-シリアル変換モード を選択する場合

#define CDC_APL_MODE_PP SERIAL_CONVERTER_PP

（注2）USB-シリアル変換モードの場合、RSK に対し以下のジャンパ設定が必要です。

Table 9-1 USB-シリアル変換モードジャンパ設定

RX62N (SCI2)		
ポート	信号	ジャンパ設定
P13	TXD2-A	J15.2-3 & J6.2-3
P12	RXD2-A	J16.2-3 & J5.2-3
RX630 (SCI0)		
ポート	信号	ジャンパ設定
P20	TXD0	なし
P21	RXD0	なし
RX63T (SCI1)		
ポート	信号	ジャンパ設定
PD3	TXD1	なし
PD5	RXD1	なし

9.1.1 MCU 端子設定

このアプリケーションプログラムでは、VBUS 端子の設定は r_usb_pcdc_apl.c ファイル内の usb_mcu_init 関数で行われています。

9.2 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。

9.2.1 初期設定

初期設定では、MCU の端子設定、USB ドライバの設定、USB コントローラの初期設定を行います。

9.2.2 メインループ

このメインループでは、以下の処理を行います。

1. APL は、ステートとそのステートに関連するイベントにより管理を行っています。 APL では、まず、接続されたデバイスごとにステート (Table 9-2参照)の確認を行います。なお、このステートは、APL が管理する構造体(「9.2.3　ステートとイベントの管理」参照) のメンバに格納されています。
2. 次に APL は、そのステートに関連するイベント(
3. Table 9-3参照)の確認を行い、そのイベントに応じた処理を行います。そのイベント処理後、APL は、必要に応じてステートを変化させます。なお、このイベントは、APL が管理する構造体(「9.2.3 ステートとイベントの管理」参照) のメンバに格納されています。
4. メインループ処理中に USB ホストからのサスペンド要求やデタッチがあった場合、APL は CDC デバイス(RSK)を低消費電力モードに移行します。消費電力低減モードについては、「9.3 MCU 消費電力低減処理」を参照してください。

以下に、APL の処理概要を示します。

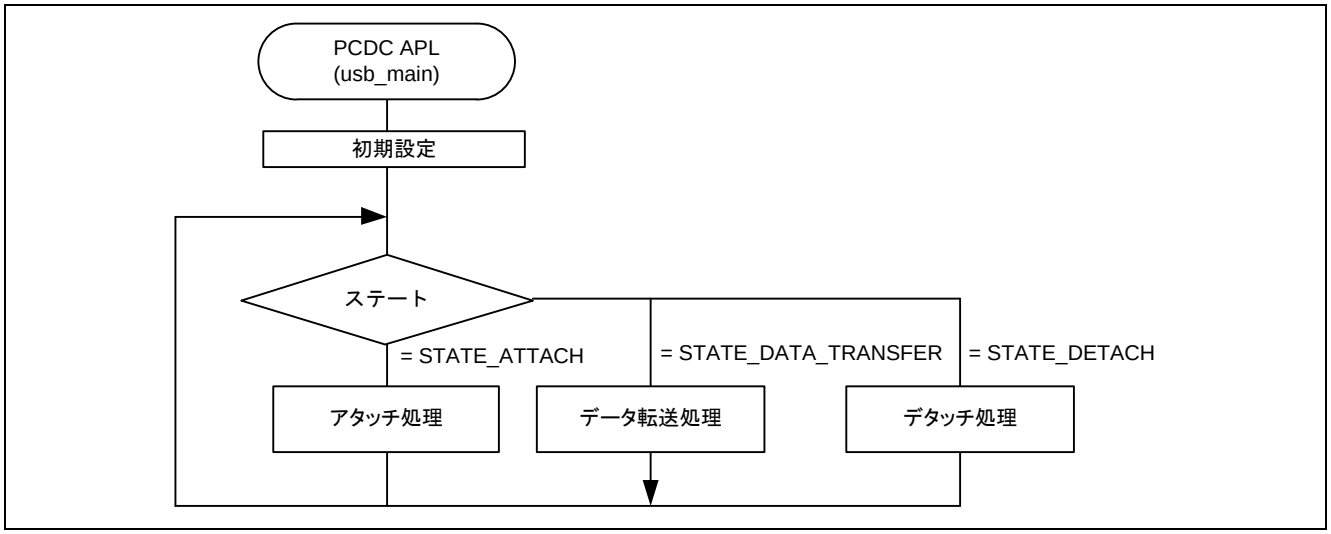


Figure 9-1　メインループ処理

Table 9-2 ステート一覧

ステート	ステート処理概要	関連イベント
STATE_ATTACH	アタッチ処理	EVENT_CONFIGURD
STATE_DATA_TRANSFER	データ転送処理	EVENT_USB_READ_START
		EVENT_USB_READ_COMPLETE
		EVENT_USB_WRITE_START
		EVENT_USB_WRITE_COMPLETE
		EVENT_COM_NOTIFY_START
		EVENT_COM_NOTIFY_COMPLETE
STATE_DETACH	デタッチ処理	--

Table 9-3 イベント一覧

イベント	概要
EVENT_CONFIGURD	USB デバイス接続完了
EVENT_USB_READ_START	USB データ受信要求
EVENT_USB_READ_COMPLETE	USB データ受信完了
EVENT_USB_WRITE_START	USB データ送信要求
EVENT_USB_WRITE_COMPLETE	USB データ送信完了
EVENT_COM_NOTIFY_START	Class Notification“SerialState” 送信要求
EVENT_COM_NOTIFY_COMPLETE	Class notification”SerialState” 送信完了
EVENT_NONE	イベントなし

9.2.3 ステートとイベントの管理

ステートとイベントは、以下の構造体のメンバ(*state, event[]*)によって管理されています。この構造体は、APLが用意している構造体です。

```
typedef struct DEV_INFO                 /* Structure for CDC device control */
{
    uint16_t    state;                  /* State for application */
    uint16_t    event_cnt;              /* Event count */
    uint16_t    event[EVENT_MAX];       /* Event. */
}
DEV_INFO_t;
```

[Note]

イベント取得処理で、取り出すイベントが存在しない場合、”EVENT_NONE”がセットされます。

以下にステートごとの処理概要を示します。

1) アタッチ処理 (STATE_ATTACH)

== 概要 ==

このステートでは、CDC デバイス(RSK)が USB ホストに接続され、Enumeration が完了したことを APL に通知する処理を行い、ステートを STATE_DATA_TRANSFER に変更します。

== 内容 ==

① APL では、はじめに初期化関数がステートを STATE_ATTACH にセットし、イベントを EVENT_NONE にセットします。
② CDC デバイス(RSK)を USB ホストに接続するまで STATE_ATTACH 状態が継続され、cdc_connect_wait()がコールされます。RSK を USB ホストに接続した後、USB ホストとの Enumeration が完了するとコールバック関数 cdc_configured が USB ドライバよりコールされ、このコールバック関数がイベント EVENT_CONFIGURD を発行します。なお、このコールバック関数は、USB_HCDREG_t 構造体のメンバ devconfig に設定した関数です。
③ EVENT_CONFIGURD では、ステートを STATE_DATA_TRANSFER に変更し、イベント EVENT_USB_READ_START を発行します。

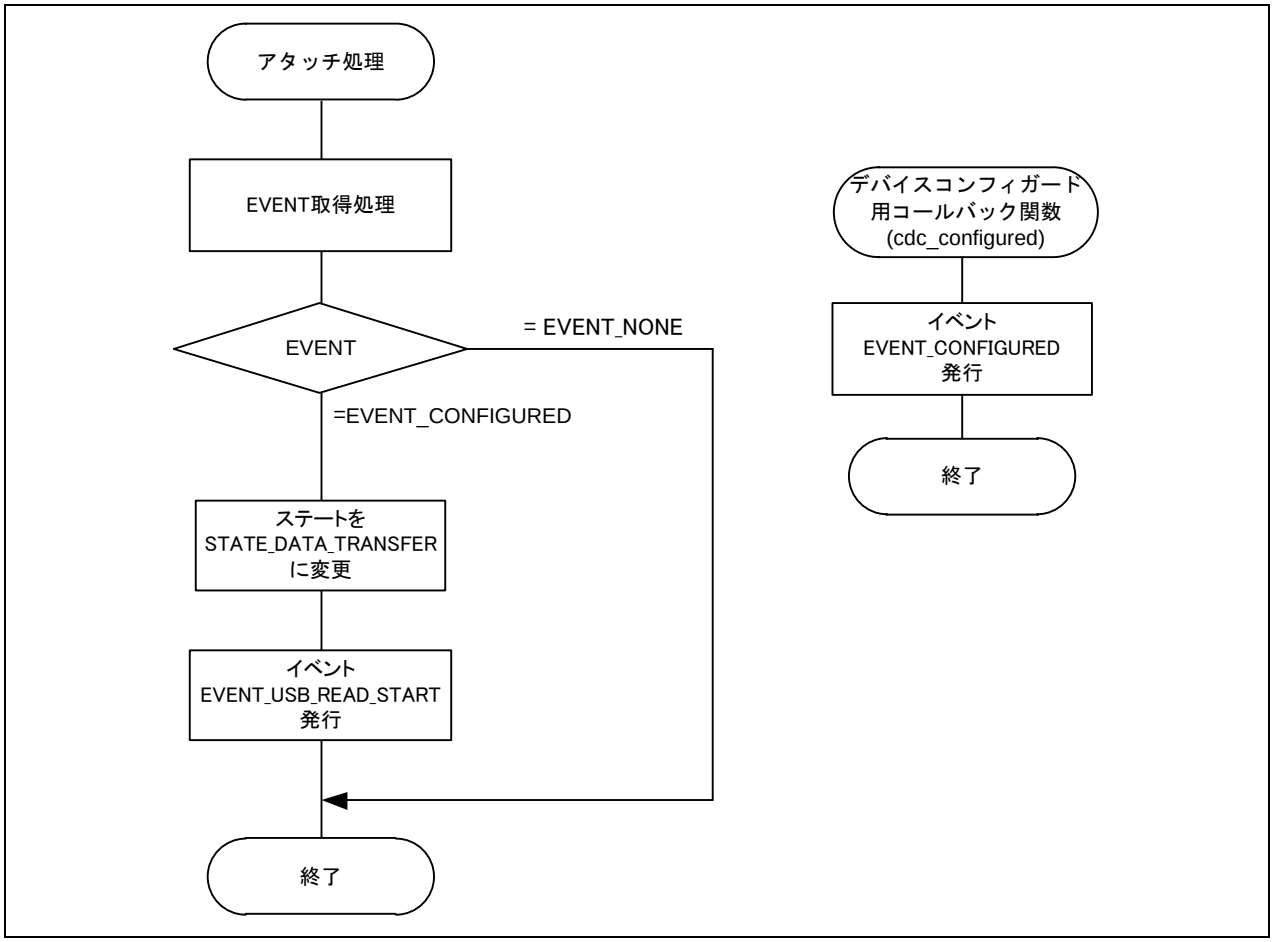


Figure 9-2 アタッチ処理概略フロー

2) データ転送処理(STATE_DATA_TRANSFER)

PCDC のデータ転送処理には、USB ループバックモードと USB-シリアル変換モードの 2 種類のデータ転送処理が存在します。

a) USB ループバックモード

== 概要 ==

このステートでは、USB ホストからのデータを受信し、その受信したデータをそのまま USB ホストに対し送信するループバック処理を行います。その他、USB ホストと接続後、USB ホストから最初のデータを受信するまでの間、定期間隔で USB ホストに対し初期接続メッセージ送信も行います。

== 内容 ==

[受信処理]

① EVENT_USB_READ_START では、USB ホストから送信されるデータを受信するため USB ドライバに対しデータ受信要求を行います。
② データ受信処理が完了するとコールバック関数 cdc_read_complete がコールされます。このコールバック関数では、EVENT_USB_READ_COMPLETE を発行します。
③ EVENT_USB_READ_COMLETE では、イベント EVENT_USB_WRITE_START を発行します。

[送信処理]

④ EVENT_USB_WRITE_START では、上記受信したデータを USB ホストに送信するため、USB ドライバに対しデータ送信要求を行います。
⑤ データ送信転送処理が完了するとコールバック関数 cdc_write_complete がコールされます。このコールバック関数では、EVENT_USB_WRITE_COMPLETE を発行します。
⑥ EVENT_USB_WRITE_COMLETE では、イベント EVENT_USB_READ_START を発行します。これにより、上記①が再度処理されます。

[初期接続メッセージ]

1. CDC デバイス(RSK)を USB ホストに接続すると RSK から USB ホストに対し以下のメッセージが一定周期で送信されます。

   PCDC.Virtual serial COM port. Type characters into terminal.
   The target will receive the characters over USB CDC, then copy them to a USB transmit buffer to be echoed back over USB."

2. PC 上の Terminal ソフトから CDC デバイス(RSK)へデータが転送されると”Echo Mode.”が Terminal ソフト上に表示されます。本表示以降は、ループバックモード処理が行われます。

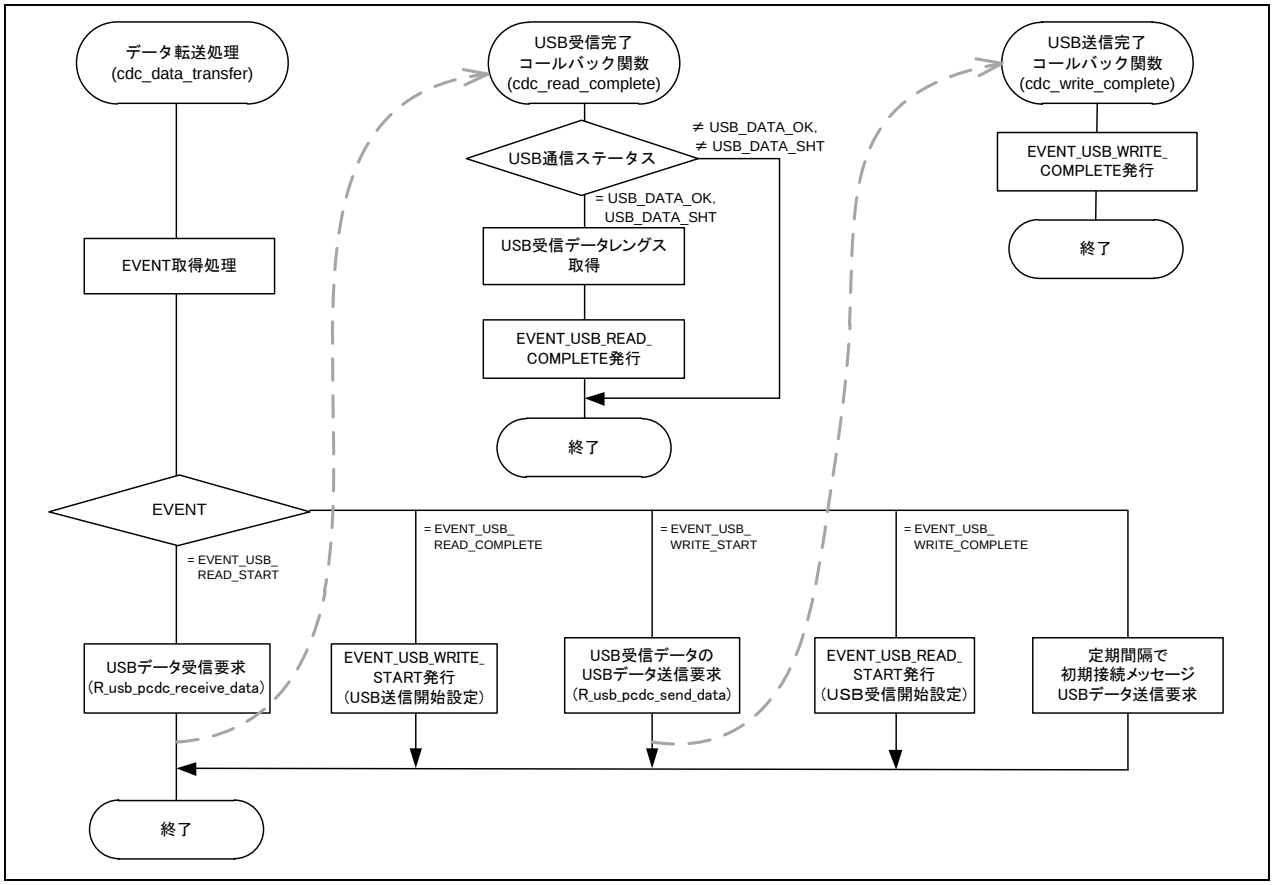


Figure 9-3　データ転送処理概略フロー（USB ループバックモード）

b) USB-シリアル変換モード

== 概要 ==

このステートでは、USB ホストからデータを受信し、その受信したデータ COM ポートへの送信および COM ポートから受信したデータを USB ホストに送信する処理を行います。その他、COM ポートでのエラーを検出した場合、USB ホストに対しクラスノーティフィケーション SerialState を送信します。

== 内容 ==

[USB データ受信->COM ポートへ USB 受信データを送信]

① EVENT_USB_READ_START では、USB ホストから送信されるデータを受信するため USB ドライバに対しデータ受信要求を行います。

② USB データ受信処理が完了するとコールバック関数 cdc_read_complete がコールされます。このコールバック関数では、EVENT_USB_READ_COMPLETE を発行します。

③ EVENT_USB_READ_COMPLETE では、受信データレングスをチェックし、受信データがある場合は受信フラグを設定します。受信データがない場合はエラーリカバリ処理として、イベント EVENT_USB_READ_START を発行し再度 USB ドライバに対しデータ受信要求を行います。

④ 受信フラグは EVENT_USB_NONE の場合に参照され、USB ホストから受信したデータを COM ポートに送信し、イベント EVENT_USB_READ_START を発行します。

[COM ポート受信->USB へ COM ポート受信データ送信]

① EVENT_USB_NONE では COM ポート受信データをチェックし、COM ポートからの受信データがあればイベント EVENT_USB_WRITE_START を発行します。

② EVENT_USB_WRITE_START では、送信フラグをセットし、COM ポートから受信したデータを USB ドライバに対しデータ送信要求を行います。送信フラグがセットされている間は、COM ポートの受信データチェックを保留します。

③ USB データ送信処理が完了するとコールバック関数 cdc_write_complete がコールされます。このコールバック関数ではイベント EVENT_USB_WRITE_COMPLETE を発行します。

④ EVENT_USB_WRITE_COMPLETE では、送信フラグをクリアし、COM ポート受信データチェックの保留を解除します。

[COM ポートエラー検出->USB へクラスノーティフィケーション SerialState 送信]

① COM ポートでのエラーを検出した場合、イベント EVENT_COM_NOTIFY_START がセットされ、クラスノーティフィケーション SerialState 送信を行います。

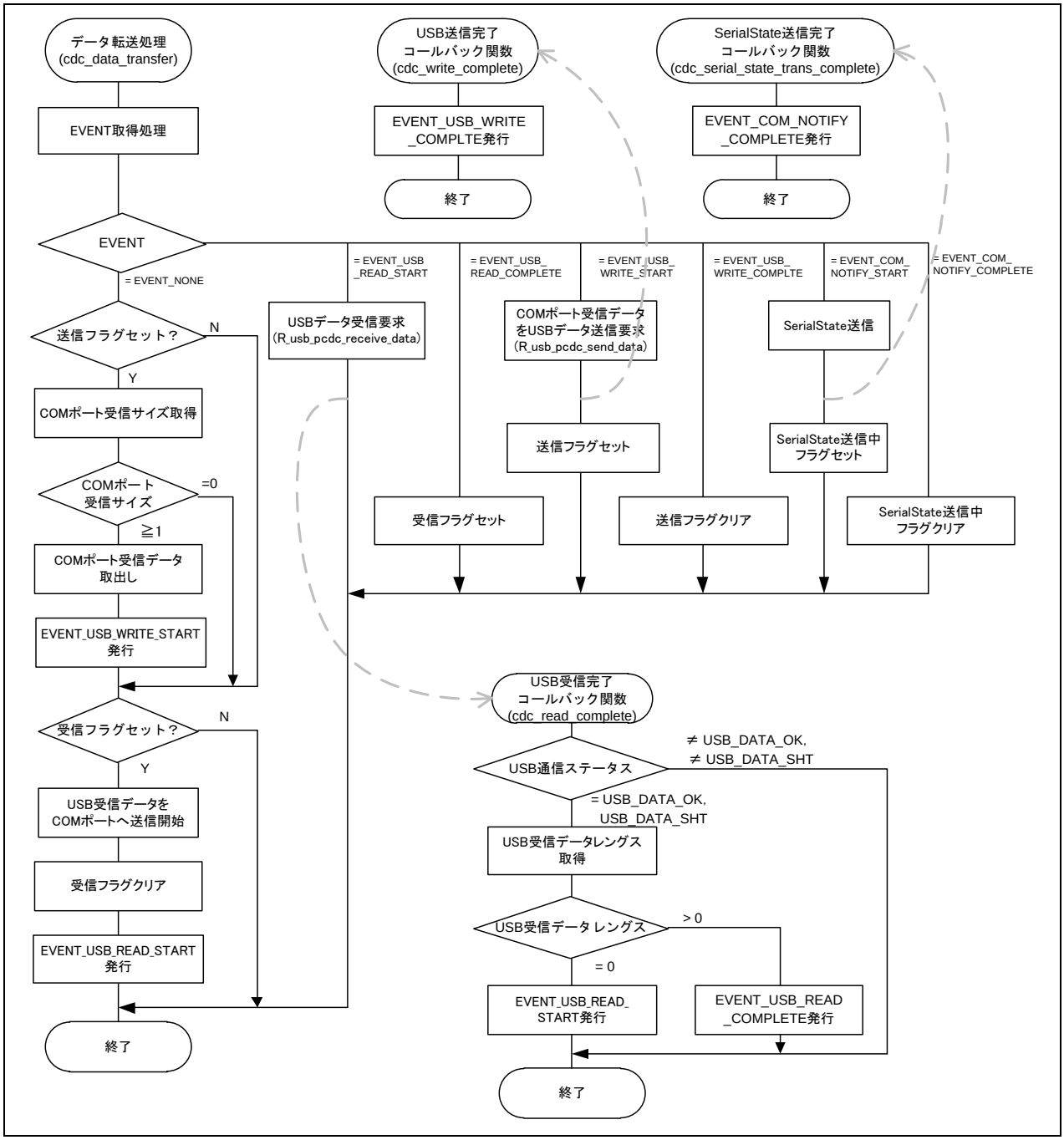


Figure 9-4　データ転送処理 処理概略フロー（USB-シリアル変換モード）

3) デタッチ処理 (STATE_DETACH)

接続されたUSBホストからデタッチするとUSBドライバよりコールバック関数cdc_detachがコールされます。このコールバック関数では、ステートをSTATE_DETACHに変更します。STATE_DETACHでは、特定の変数(cdc_dev_info)にクリア処理、ステートをSTATE_ATTACHに変更するなどの処理を行います。

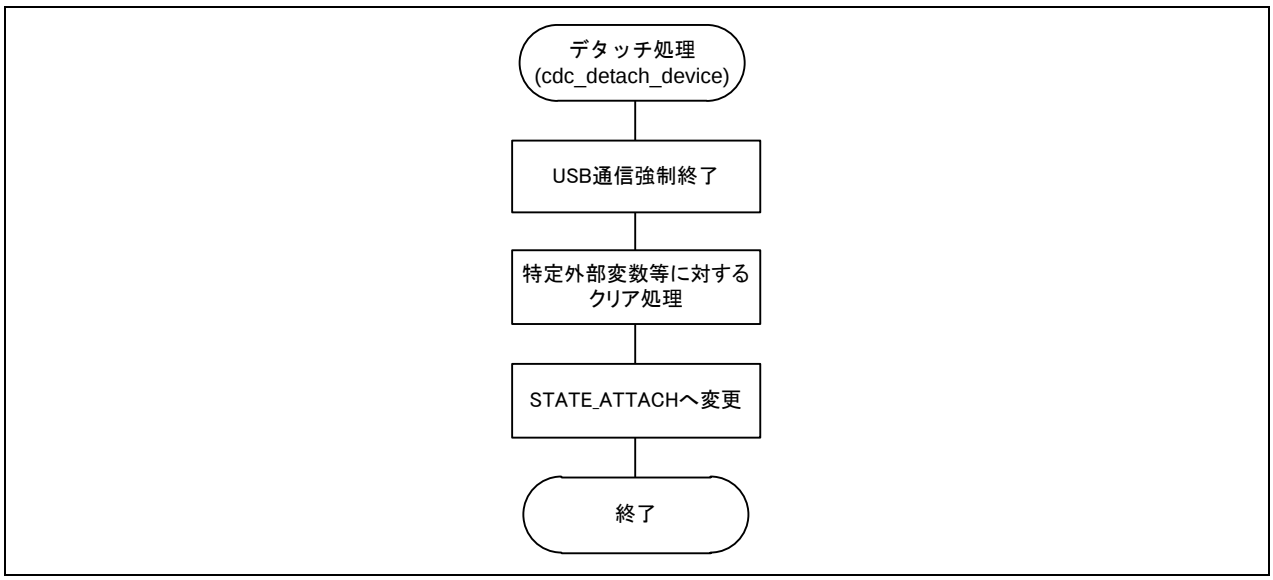


Figure 9-5 デタッチ処理概略フロー

9.3 MCU 消費電力低減処理

MCU 消費電力低減処理は、以下の条件が成立すると消費電力低減モードに移行する処理を行います。

Table 9-4 消費電力低減機能状態遷移条件

遷移条件			消費電力低減モード
USB デバイス状態		ステート	
VBUS	DVSQ		
OFF	ー	STATE_ATTACH	ソフトウエアスタンバイモード
ON	Suspend Configured	STATE_DATA_TRANSFER	スリープモード
ON	Suspend Configured 以外	ー	通常モード（プログラム実行状態）

a) デタッチ（VBUS OFF）状態で、APL のステートが STATE_ATTACH の場合、APL は MCU をソフトウエアスタンバイモードに遷移します。なお、ソフトウエアスタンバイモードからの復帰は、RSK を USB ホストにアタッチすることにより通常モードに移行します。

b) CDC デバイスがサスペンド状態で、APL のステートが STATE_DATA_TRANSFER の場合、APL は MCU をスリープモードに遷移します。なお、スリープモードからの復帰は、USB ホストから送信されるレジューム信号の受信により通常モードに移行します。

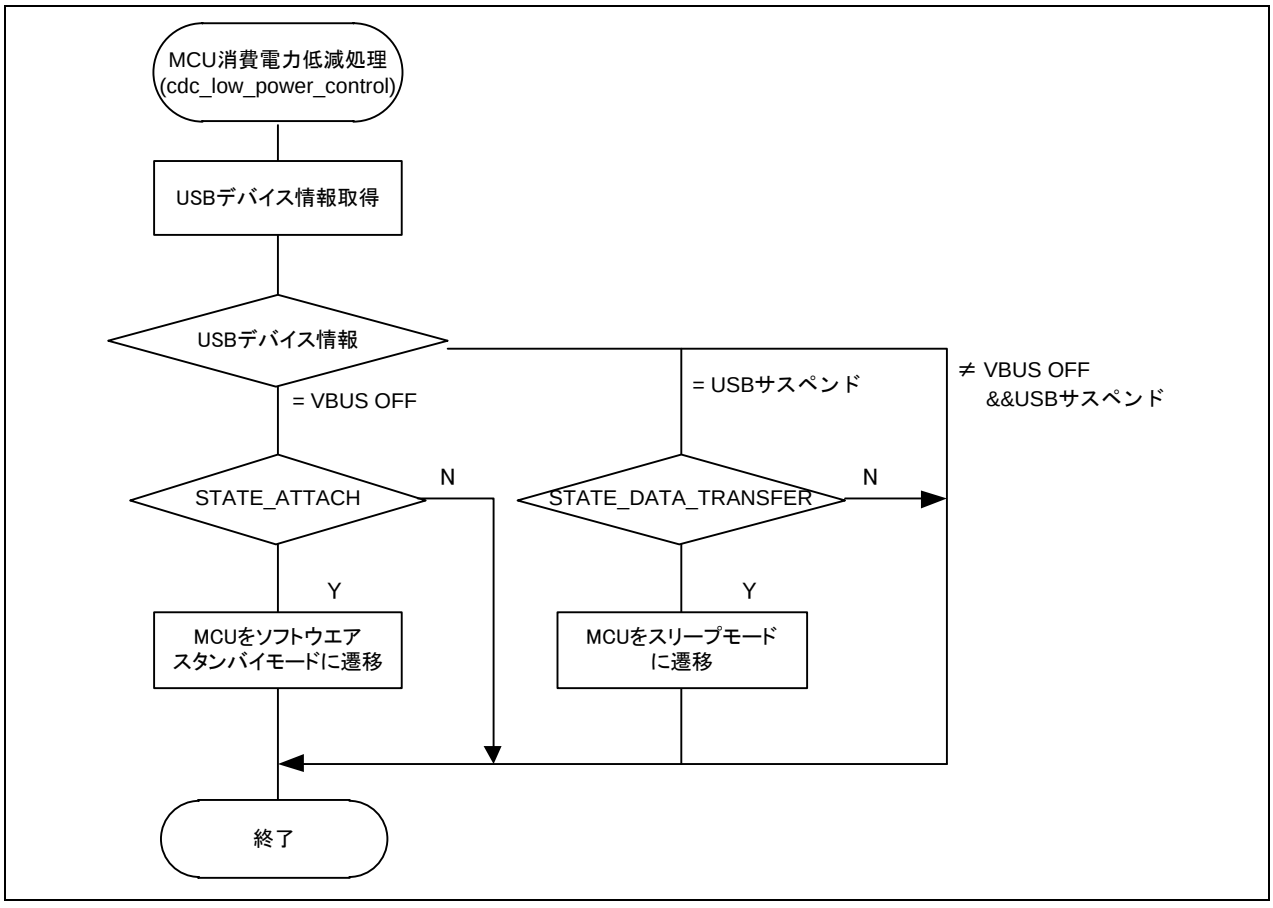


Figure 9-6 MCU 消費電力低減処理概略フロー

9.4 ディスクリプタ

PCDC のディスクリプタ情報は r_usb_pcdc_descriptor.c に記述しています。

9.5 セットアップ

9.5.1 ハードウェア

PCDC の動作環境例をFigure 9-7 に示します。評価ボードのセットアップ、エミュレータなどの使用方法については各取扱説明書を参照してください。

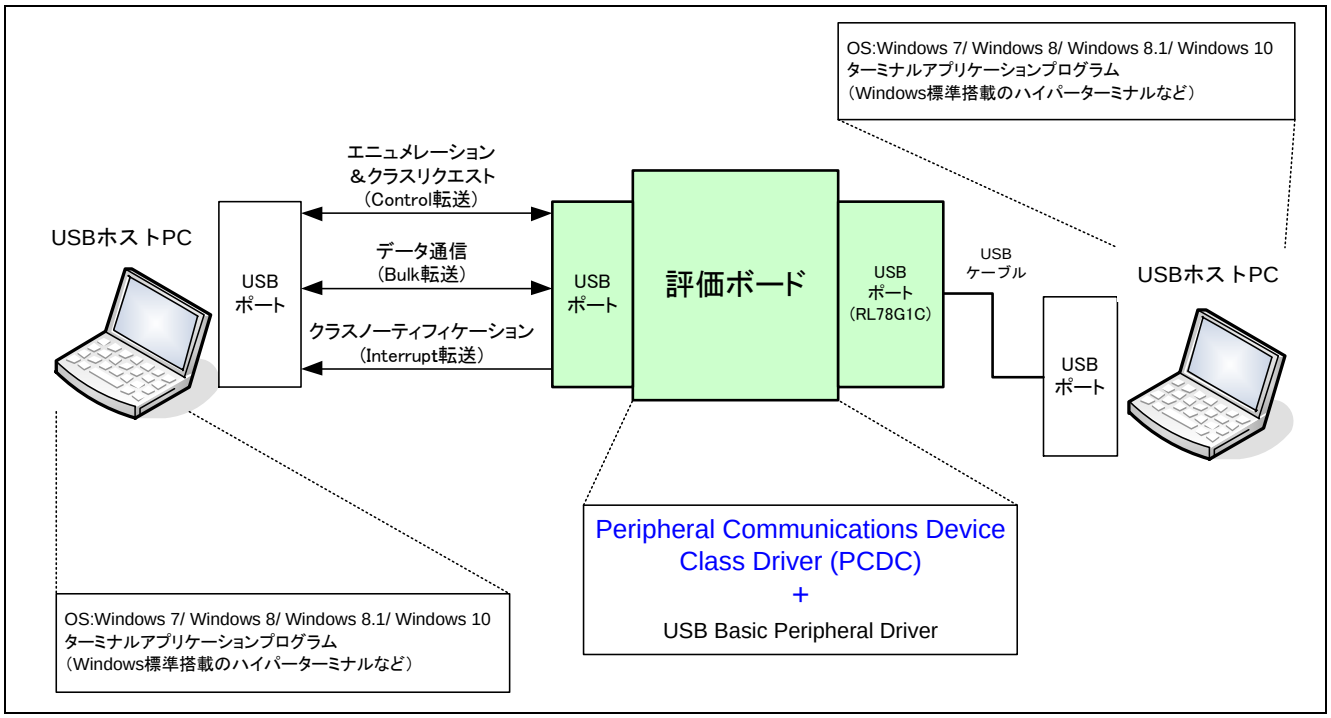
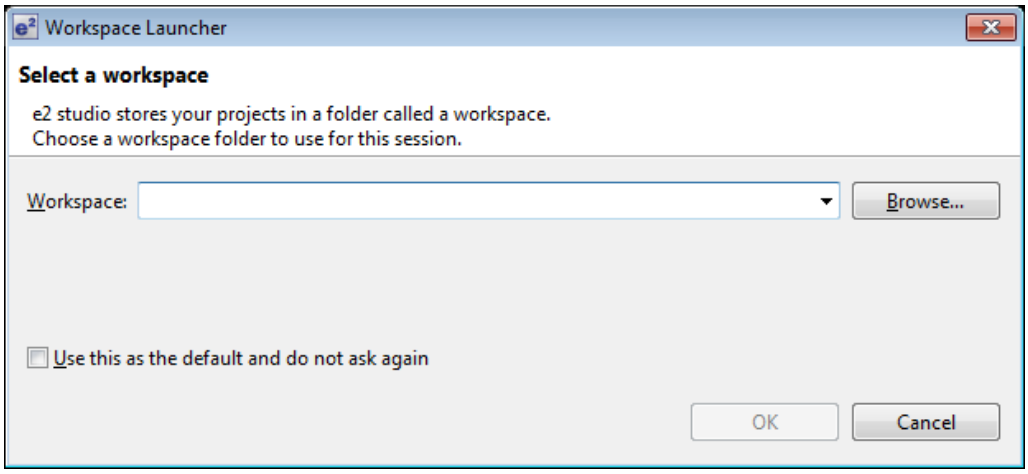


Figure 9-7 Example Operating Environment

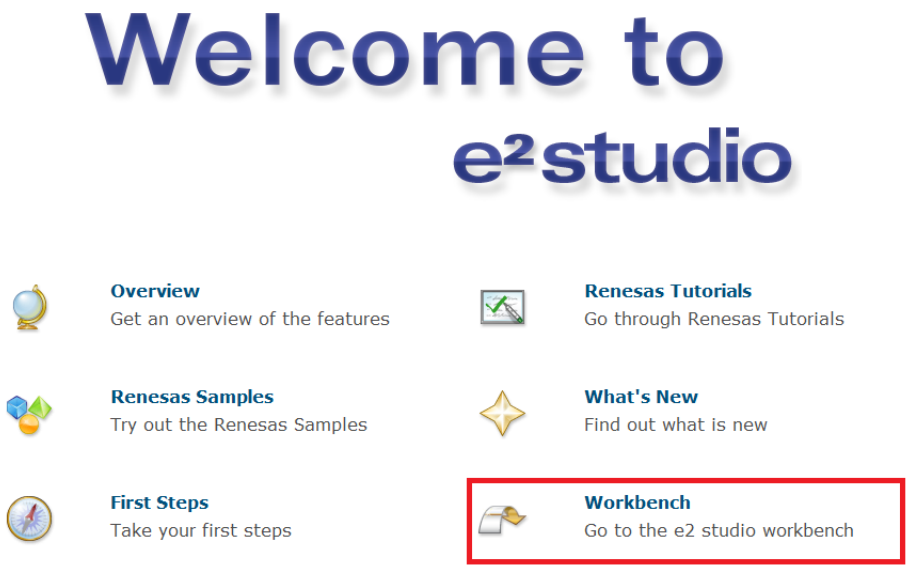
9.5.2 ソフトウェア

(1). e² studio を起動

a) e² studio を起動してください。
b) 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。

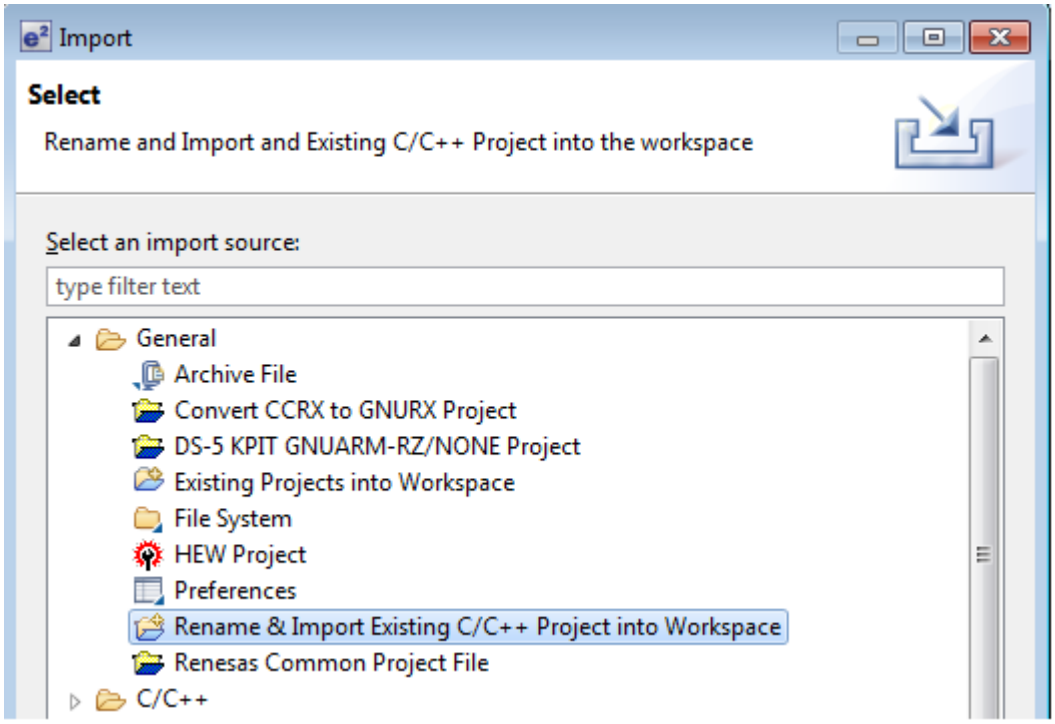


c) Workbench アイコンをクリックしてください。

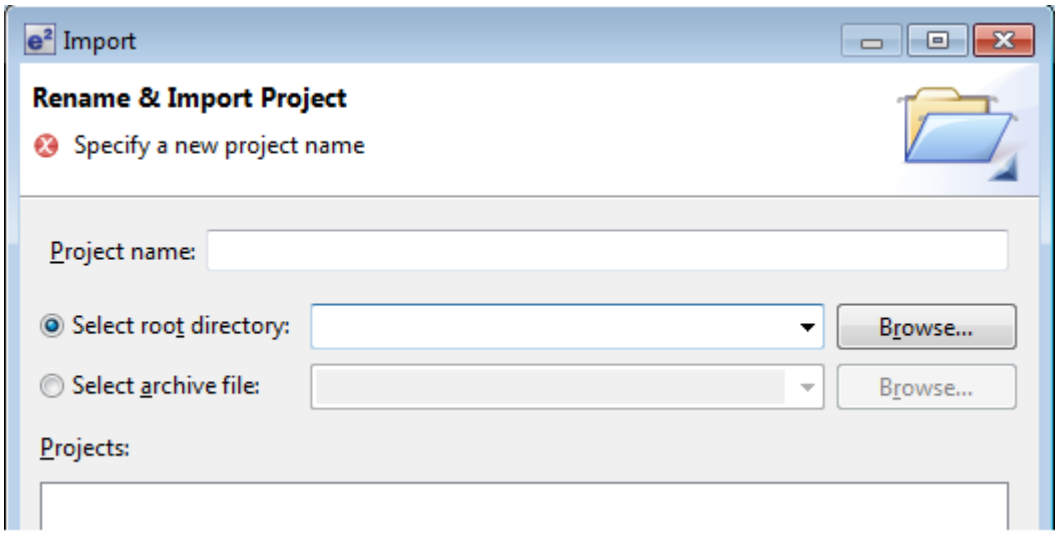


(2). ワークスペースへのプロジェクトの登録

a) [File] --> [Import]を選択してください。
b) General => Rename & Import Existing C/C++ Project into Workspace を選択してください。



プロジェクトファイル".cproject"が格納されたフォルダを”Select root directory”に入力してください。



c) “Finish”をクリック

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した。同様の方法で他のプロジェクトを同一のワークスペースへインポートすることができます。

(3). “Build”ボタンをクリックし、実行プログラムを生成してください。

(4). デバッガへの接続を行い、実行プログラムをダウンロードしてください。“Run”ボタンをクリックすると、プログラムが実行されます。

10. アプリケーションの作成方法

本章では、PCDC とUSB-BASIC-FWを組み合わせ、USB ドライバとして使用するために必要な初期設定の方法と、メインルーチン処理方法及び API 関数を使用したデータ転送例を示します。

10.1 USB ドライバの初期設定方法

USB ドライバを使用するためには、以下の設定を行う必要があります。

- MCU の端子設定
- USB コントローラの起動と設定
- USB ドライバの設定

以下に各設定の例を示します。

```
/* PCDC 用 USB 通信構造体 */
USB_UTR_t  usb_gpcdc_utr

void usb_pcdc_apl(void)
{
        /* USB 通信構造体 */
        USB_UTR_t               *ptr;
        USB_ER_t                err;
        USB_PCDREG_t            reg;

        /* MCU の端子設定（「10.1.1 MCU の端子設定」参照*/
        usb_mcu_setting();

        /* USB ドライバの設定（「10.1.2　USB ドライバの設定」参照） */
        ptr        = &usb_gpcdc_utr;
        ptr->ip   = USB_PERI_USBIP_NUM;
        ptr->ipp = R_usb_cstd_GetUsbIpAdr(ptr->ip);

        usb_cstd_ScheInit();
        R_usb_pstd_PcdOpen();

        usb_papl_registration(ptr);
        R_usb_pcdc_driver_start(ptr);

        /* USB モジュールの起動と設定（「10.1.3 USB モジュールの起動と設定」参照） */
        err = R_USB_Open( USB_IP0 );
        if(err != USB_E_OK)
        {
        /* エラー処理 */
        }


        /* メインルーチン */
        usb_papl_mainloop();
}
```

10.1.1 MCU の端子設定

USB コントローラを使用するためには、USB の入出力端子を設定する必要があります。以下に、設定が必要な USB 入出力端子例を示します。

Table10-1　ペリフェラル動作時の USB 入出力端子設定

端子名	入出力	機能
USB_VBUS	入力	USB 用 VBUS 端子

※ PCDC およびUSB-BASIC-FWは MCU の端子設定を行いません。端子設定の設定が必要な場合は各 MCU のユーザーズマニュアルを参照し、ご使用の評価ボードに合わせて端子設定を行ってください。

10.1.2 USB ドライバの設定

USB ドライバの設定では、スケジューラへのタスク登録及びUSB-BASIC-FWに対するクラスドライバの情報登録を行います。以下に、クラスドライバ情報の登録とスケジューラへのタスク登録手順を示します。

① USB_UTR_t 型で宣言された USB 通信構造体のメンバに USB モジュールの IP 番号を設定する。
② R_usb_cstd_GetUsbIpAdr()を呼び出し、USB 通信構造体に USB レジスタのベースアドレスを設定する。
③ USB-BASIC-FWの API 関数（R_usb_pstd_PcdOpen()）を呼び出し、スケジューラへ PCD タスクを登録する。
④ クラスドライバ登録用構造体（USB_PCDREG_t）の各メンバに情報を設定後、R_usb_pstd_DriverRegistration()を呼び出すことでUSB-BASIC-FWに対するクラスドライバの情報登録を行う。
⑤ クラスドライバの API 関数（R_usb_pcdc_driver_start()）を呼び出し、スケジューラへ PCDC タスクを登録する。

USB_PCDREG_t で宣言された構造体に設定する情報例を以下に示します。

```
void usb_papl_registration(USB_UTR_t *ptr)
{
   USB_PCDREG_t    driver;  ←クラスドライバ登録用構造体

   /* パイプ情報テーブルを設定 */
   driver.pipetbl          = (uint16_t*)&usb_gpcdc_SmplEpTbl[0]; // (注 1)
   /* Device Descriptor テーブルを設定 */
   driver.devicetbl        = (uint8_t*)&usb_gpcdc_DeviceDescriptor;    // (注 2)
   /* Qualifier Descriptor テーブルを設定 */
   driver.qualitbl         = (uint8_t*)USB_NULL; // (注 3)
   /* Configuration Descriptor テーブルを設定 */
   driver.configtbl        = (uint8_t**)usb_gpcdc_ConPtr; // (注 2, 4)
   /* Other Configuration Descriptor テーブルを設定 */
   driver.othertbl         = (uint8_t**)USB_NULL; // (注 3)
   /* String Descriptor テーブルを設定 */
   driver.stringtbl        = (uint8_t**)usb_gpcdc_StrPtr; // (注 2, 5)
   /* クラスドライバ登録時に呼び出される関数を設定 */
   driver.classinit        = (USB_CB_INFO_t)&usb_cstd_DummyFunction;
   /* デフォルトステート遷移時に呼び出される関数を設定 */
   driver.devdefault       = (USB_CB_INFO_t)&usb_cstd_DummyFunction;
   /* エニュメレーション完了時に呼び出される関数を設定 */
   driver.devconfig        = (USB_CB_INFO_t)&usb_pcdc_smpl_open; // (注 6)
   /* USB デバイス切断時に呼ばれる関数を設定 */
   driver.devdetach        = (USB_CB_INFO_t)&usb_pcdc_smpl_close; // (注 6)
   /* デバイスをサスペンド状態に移行時に呼ばれる関数を設定 */
   driver.devsuspend       = (USB_CB_INFO_t)&usb_cstd_DummyFunction;
   /* デバイスのサスペンド状態解除時に呼ばれる関数を設定 */
   driver.devresume        = (USB_CB_INFO_t)&usb_pcdc_smpl_resume_cb;
   /* インタフェース変更時に呼ばれる関数を設定 */
   driver.interface        = (USB_CB_INFO_t)&R_usb_pcdc_set_interface;
   /* 標準リクエスト以外のコントロール転送処理時に呼ばれる関数を設定 */
   driver.ctrltrans        = (USB_CB_TRN_t)&R_usb_pcdc_usr_ctrl_trans_function;

   /* PCD へクラスドライバ情報を登録 */
   R_usb_pstd_DriverRegistration(ptr, &driver);
}
```

（注1）パイプ情報テーブルは、アプリケーション内で定義してください。パイプ情報テーブル例を以下に示します。パイプ情報テーブルについては USB Basic Host and Peripheral Driver アプリケーションノート (Document No.R01AN0512JJ)を参照してください。

= パイプ情報テーブル例 =

```
uint16_t usb_gpcdc_EpTbl1[] =
{
    USB_PIPE1,
    USB_BULK | USB_BFREOFF | USB_DBLBON | USB_CNTMDOFF | USB_SHTNAKON |
    USB_DIR_P_IN | USB_EP1,
    USB_NONE,
    64,
    USB_IFISOFF | USB_IITV_TIME(0u),
    USB_CUSE,

    USB_PIPE2,
    USB_BULK | USB_BFREOFF | USB_DBLBON | USB_CNTMDOFF | USB_SHTNAKON |
    USB_DIR_P_OUT | USB_EP2,
    USB_NONE,
    64,
    USB_IFISOFF | USB_IITV_TIME(0u),
    USB_CUSE,

    USB_PIPE6,
    USB_INT    | USB_DIR_P_IN | USB_EP3,
    USB_NONE,
    16u,
    USB_IITV_TIME(0u),
    USB_CUSE,

    /* Pipe end */
    USB_PDTBLEND,
};
```

（注2）各ディスクリプタは USB 規格書をもとに作成してください。

（注3）これらのメンバには”USB_NULL”を指定しください。

（注4）このメンバにはコンフィグレーションディスクリプタテーブルを設定した配列(下記参照)の先頭アドレスを設定してください。

```
uint8_t *usb_gpcdc_ConPtr[] =
{
    usb_gpcdc_Configration
};
```

（注5）このメンバにはストリングディスクリプタを設定した配列の先頭アドレスを設定してください。下記参照。

```
uint8_t *usb_gpcdc_StrPtr[] =
{
    usb_gpcdc_StringDescriptor0,
    usb_gpcdc_StringDescriptor1,
    usb_gpcdc_StringDescriptor2,
}
```

（注6）これらのメンバに登録した関数では、外部変数の初期化等を行ってください。

10.1.3 USB モジュールの起動と設定

以下に、USB モジュールの起動と設定手順を示します。

① R_USB_Open()を呼び出すことで USB モジュールを起動させる。本 API 関数は初期設定時に一度だけ呼び出してください。

② R_usb_cstd_UsbIpInit()を呼び出すことで USB モジュールの初期設定を行う。

10.2 メインルーチン処理方法

USB ドライバは初期設定後アプリケーションのメインルーチン内でスケジューラ（R_usb_cstd_Scheduler()）を呼び出すことで動作します。

メインルーチン内で R_usb_cstd_Scheduler()を呼ぶことでイベントの有無を確認し、イベントがある場合、スケジューラにイベントが発生していることを通知するためのフラグをセットします。

R_usb_cstd_Scheduler()呼び出し後、R_usb_cstd_CheckSchedule()を呼び出しイベントの有無を確認してください。また、イベントの取得とそのイベントに対する処理は定期的に行う必要があります。（注 1）

```
void usb_papl_mainloop(void)
{
  while(1)      ←メインルーチン
  {
    R_usb_cstd_Scheduler();              ←イベントの確認と取得、フラグセット (注 1)

    if(USB_FLGSET == R_usb_cstd_CheckSchedule())    ←イベント有無の判定、フラグクリア
    {                                                                    } (注 2)
        R_usb_pstd_PcdTask((USB_VP_INT)0);          ←PCD タスク
        R_usb_pcdc_task((USB_VP_INT)0);             ←CDC ドライバタスク
    }
    pcdc_application();                  ←ユーザーアプリケーション
  }
}
```

（注1） R_usb_cstd_Scheduler()でイベントを取得後、処理を行う前に再度 R_usb_cstd_Scheduler()で他のイベントを取得すると、最初のイベントは破棄されます。イベント取得後は必ず各タスクを呼び出し、処理を行ってください。

（注2） これらの処理は、アプリケーションプログラムのメインループ内に必ず記述してください。

10.3 データ転送方法

アプリケーションから PCDC のクラス API 関数（R_usb_pcdc_SendData ()/R_usb_pcdc_ReceiveData ()）を呼び出すことで、接続された USB デバイスに対してデータ送受信要求（Bulk Out/Bulk In 転送）を行います。

R_usb_pcdc_SendData ()/R_usb_pcdc_ReceiveData ()は以下の引数を持ちます。

データ送信時

```
R_usb_pcdc_SendData (
        USB_UTR_t    *ptr,       ←USB 通信構造体
        uint8_t      *table,     ←デバイスに送信するデータ格納領域のアドレス（注 1）
        uint32_t     size,       ←USB デバイスへ送信するデータサイズ
        USB_CB_t     complete    ←データ送信完了時に呼ばれるコールバック関数（注 1）
)
```

（注1）データ格納領域およびデータ転送完了時に呼ばれるコールバック関数はアプリケーションで定義する必要があります。

データ受信時

```
R_usb_pcdc_ReceiveData (
        USB_UTR_t    *ptr,       ←USB 通信構造体
        uint8_t      *table,     ←デバイスから受信したデータ格納領域のアドレス（注 1）
        uint32_t     size,       ←デバイスから受信するデータサイズ
        USB_CB_t     complete    ←データ受信完了時に呼ばれるコールバック関数（注 1）
)
```

（注1）データ格納領域およびデータ転送完了時に呼ばれるコールバック関数はアプリケーションで定義する必要があります。

11. e² studio 用プロジェクトを CS+で使用する場合

PCDC のプロジェクトは、統合環境 e²studio で作成されています。PCDC を CS+で動作させる場合は、下記の手順にて読み込んでください。

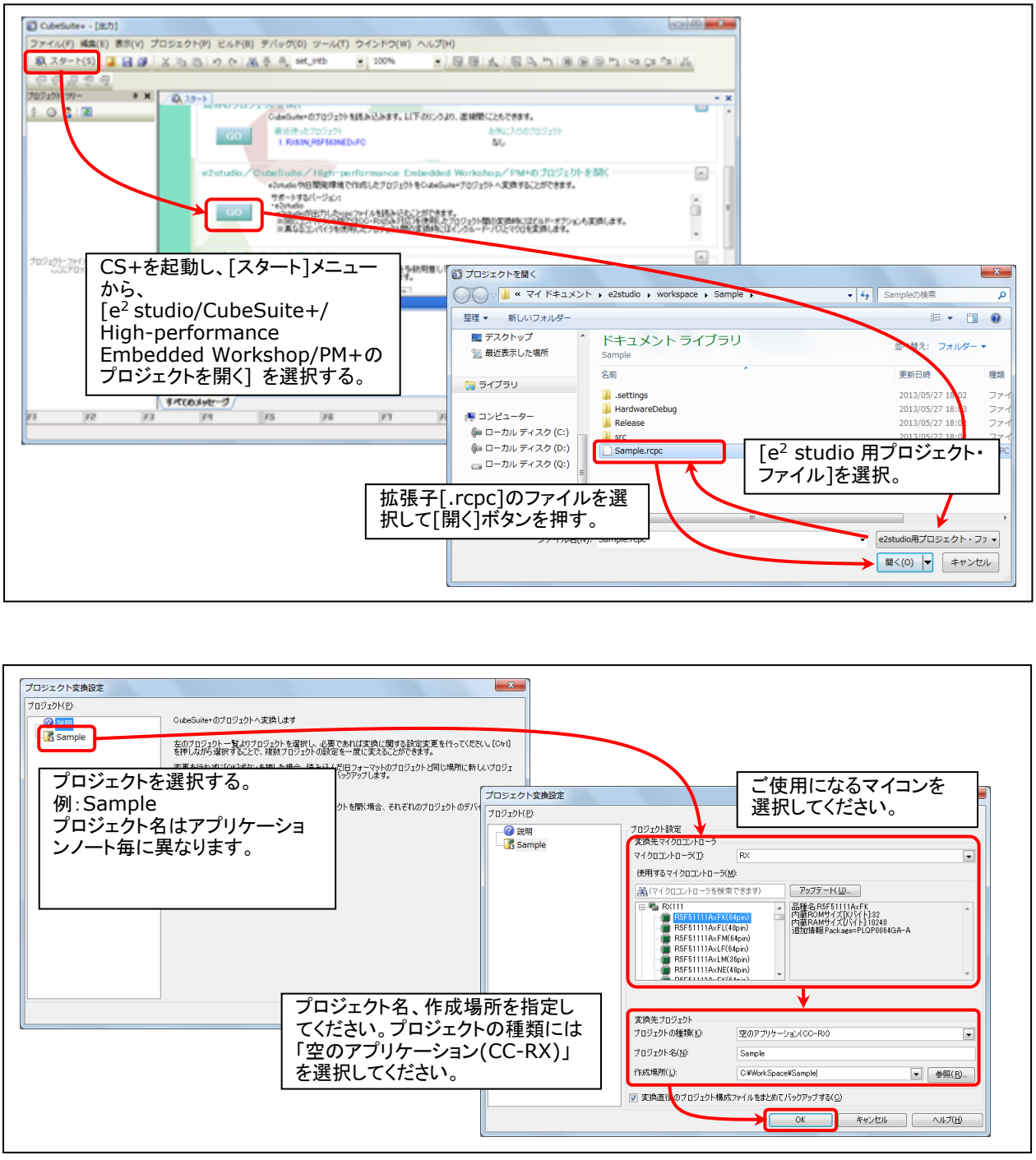


Figure 11-1 e² studio 用プロジェクトの CS+読み込み方法

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/inquiry

すべての商標および登録商標は、それぞれの所有者に帰属します。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2011.03.22	—	初版発行
1.10	2011.08.10	表紙,他	動作確認デバイスに RX630、R8A66597 の追加 上記追加に伴い、RX630 に関する内容、R8A66597(Hi-Speed USB)に関する内容を追加
		4	2.2 ファイル一覧のシリアルポートドライバのファイル削除
		29	5.ペリフェラル用 CDC サンプルアプリケーション(APL)のアプリケーション仕様変更 ・SW1 使用禁止 ・SW 押下表記を機能スイッチ表記に変更
		50	6.コミュニケーションポートドライバ(CPD)の関数名、定義値の変更 ・”R_usb_ComPort”、”usb_ComPort”を”USB_cpu_Sci”に変更 ・”USB_PCDC_”を”USB_SCI”に変更
2.00	2012.7.1	—	ファームウエアアップデートによるドキュメントの改訂
2.01	2013.2.1	—	“1. 4 本書の読み方” を追加、その他ケアレスミスの修正
2.10	2013.4.1	—	V.2.10 用 First Release 動作確認デバイスに RX63T, R8A66593 を追加。この追加に伴いこれらのデバイスに関する内容を追加
2.20	2015.9.30	—	アプリケーションプログラムを変更 フォルダ構成を変更 対象デバイスに RX63N と RX631 を追加。 対象デバイスから R8A66597 と R8A66593 を削除

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/

© 2015 Renesas Electronics Corporation. All rights reserved.
Colophon 4.0