RZ/T1 グループ

R01AN2635JJ0130
Rev.1.30
Dec 07, 2017

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。
本アプリケーションノートのサンプルプログラムは「RZ/T1 グループ初期設定 Rev.1.30」をベースに作成しています。
動作環境については「RZ/T1 グループ初期設定アプリケーションノート(R01AN2554JJ0130)」を参照してください。

対象デバイス

RZ/T1 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. RZ/T1 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. R01UH0483JJ0130)
5. RZ/T1 グループ初期設定 (ドキュメント No. R01AN2554JJ0130)
6. USB Host Basic Firmware アプリケーションノート (ドキュメント No. R01AN2633JJ0130)

- 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の PSTN デバイス・サブクラス Abstract Control Model に準拠しています。

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 デバイスとのデータ通信
- 複数の CDC デバイス接続

制限事項

本モジュールには以下の制限があります。

- 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
HCD	:	Host control driver of USB-BASIC-FW
HCDC	:	Host 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
USB	:	Universal Serial Bus
USB-BASIC-FW	:	USB basic firmware for RZ/T1 グループ
タスク	:	処理の単位
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機能

2. ソフトウェア構成

Figure 2-1 に HCDC 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。

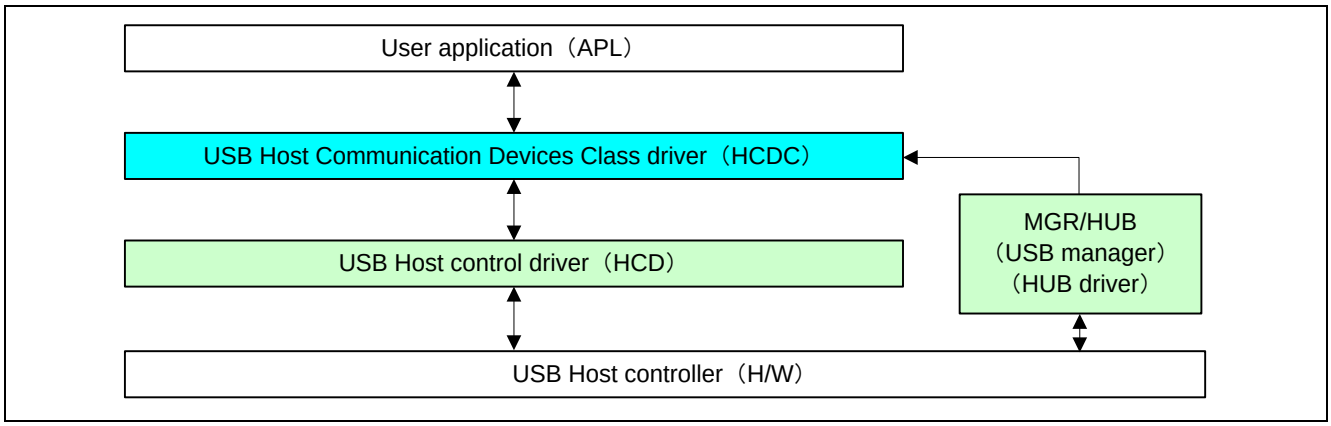


Figure 2-1 モジュール構成図

Table 2-1 モジュール機能概要

モジュール名	説明
APL	ユーザアプリケーションプログラム（システムにあわせてご用意ください）
HCDC	ホスト CDC クラスドライバ ・ CDC デバイスの照合 ・ APL からのリクエストおよびデータ通信を HCD に要求
MGR / HUB	USB マネージャ / HUB クラスドライバ（USB-BASIC-FW） ・ 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 ・ デバイスの状態管理
HCD	USB Host H/W 制御ドライバ（USB-BASIC-FW）

3. 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）

HCDC 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。

3.1 基本機能

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

3.2 Abstract Control Model(ACM) クラスリクエスト

HCDC が対応している ACM クラスリクエストを Table3-1 に示します。

Table3-1 CDC クラスリクエスト

リクエスト	コード	説明
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。

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。

3.2.1 SetLineCoding

SetLineCoding データフォーマットを Table3-2 に示します。

Table3-2 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 Table3-3 Line Coding Structure フォーマット参照

Line Coding Structure フォーマットを Table3-3 に示します。

Table3-3 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 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）

3.2.2 GetLineCoding

GetLineCoding データフォーマットを Table3-4 に示します。

Table3-4 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 Table3-3 Line Coding Structure フォーマット参照

3.2.3 SetControlLineState

SetControllLineState データフォーマットを Table3-5 に示します。

Table3-5 SetControlLineState データフォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_CONTROL_LINE_STATE (0x22)	Control Signal bitmap Table3-6 Control Signal bitmap フォーマット参照	0x0000	0x0000	None

Table3-6 Control Signal bitmap 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 - Deactivate carrier, 1 - Activate carrier
D0	DTE がレディ状態かの通知 0 - Not Present, 1 - Present

3.3 クラスノーティフィケーション

対応している HCDC のクラスノーティフィケーションを Table3-7 に示します。

Table3-7 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
SERIAL_STATE	0x20	シリアル回線状態を通知する

3.3.1 SerialState

SerialState データフォーマットを Table3-8 に示します。

Table3-8 SerialState フォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	SERIAL_STATE (0x20)	0x0000	0x0000	0x0002	UART State bitmap Table3-9 UART State bitmap フォーマット参照

Table3-9 UART State bitmap 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

3.4 構造体

3.4.1 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 Table3-10 に記します。

Table3-10 USB_HCDC_LineCoding_t 構造体

型	メンバ名	説明
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 のクラスリクエスト SetControlLineState で使用する UART 設定パラメータ用の構造体を Table3-11 に示します。

Table3-11 USB_HCDC_ControlLineState_t Structure

型	メンバ名	説明
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

3.4.2 CDC ノーティフィケーション構造体

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 Table3-12 に示します。

Table3-12 USB_HCDC_SerialState_t 構造体

型	メンバ名	説明
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	ブレーク信号検出
uint16_t (D1)	bTxCarrier:1	回線が接続されて通信可能
uint16_t (D0)	bRxCarrier:1	回線にキャリア検出

3.5 スケジューラ設定

Table 3-13 に、HCDC のスケジューラ設定を示します。

Table 3-13 スケジューラ設定

関数名	タスク ID	優先度	メールボックス名	メモリプール名	概要
R_usb_hcdc_task	USB_HCDC_TSK	USB_PRI_3	USB_HCDC_MBX	USB_HCDC_MPL	HCDC タスク
R_usb_hub_task	USB_HUB_TSK	USB_PRI_3	USB_HUB_MBX	USB_HUB_MPL	HUB タスク
R_usb_hstd_MgrTask	USB_MGR_TSK	USB_PRI_2	USB_MGR_MBX	USB_MGR_MPL	MGR タスク
r_usb_hstd_HciTask	USB_HCI_TSK	USB_PRI_1	USB_HCI_MBX	USB_HCI_MPL	HCD タスク

3.6 API

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_hcdc_if.h に記載しています。
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。

Table3-14 コンフィグレーションオプション

定義名	デフォルト値	説明
MAX_DEVICE_NUM	4	最大接続デバイス数
INIT_COM_SPEED	USB_HCDC_SPEED_9600	SetLineCoding 設定値
INIT_COM_DATA_BIT	USB_HCDC_DATA_BIT_8	
INIT_COM_STOP_BIT	USB_HCDC_STOP_BIT_1	
INIT_COM_PARITY	USB_HCDC_PARITY_BIT_NONE	

Table3-15 に HCDC API 一覧を示します。

Table3-15 HCDC API 関数一覧

関数名	機能概要
R_usb_hcdc_Task	HCDC タスク
R_usb_hcdc_driver_start	HCDC ドライバタスクスタート処理
R_usb_hcdc_class_check	ディスクリプタチェック処理
R_usb_hcdc_send_data	USB 送信処理
R_usb_hcdc_receive_data	USB 受信処理
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_set_line_coding	SetLineCoding リクエスト処理
R_usb_hcdc_get_line_coding	GetLineCoding リクエスト処理
R_usb_hcdc_set_control_line_state	SetControlLineState リクエスト処理

3.6.1 R_usb_hcdc_task

HCDC タスク

形式

void	R_usb_hcdc_task(void)

引数

—	—

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

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_MgrTask();       /* MGR Task */
       R_usb_hhub_Task();          /* HUB Task */
       R_usb_hcdc_task();          /* HCDC Task */
    }
  }
}
```

3.6.2 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

Host CDC ドライバタスクの優先度設定を行います。

補足

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 )
{
  cdc_registration();             /* Host Application Registration */
  R_usb_hcdc_driver_start();      /* Host Class Driver Task Start Setting */
}
```

3.6.3 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(uint16_t **table)

引数

**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

—	—

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。

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(void)
{
  USB_HCDREG_t  driver;

  driver.classcheck   = &R_usb_hcdc_class_check;

  R_usb_hstd_DriverRegistration(&driver);
}
```

3.6.4 R_usb_hcdc_send_data

USB 送信処理

形式

USB_ER_t	R_usb_hcdc_send_data(uint16_t pipe_id, uint8_t *table, uint32_t size, USB_UTR_CB_t complete)

引数

pipe_id	パイプ番号
*table	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_OK/USB_ERROR）

解説

指定したパイプで、指定されたアドレスから、転送サイズ分のデータを USB 送信要求します。
送信完了後、コールバック関数が呼出されます。

補足

USB 送信処理結果はコールバック関数の引数で得られます。

使用例

```
void cdc_data_transfer(uint16_t devadr)
{
  uint16_t   pipe_id;
  uint8_t    send_data[] = {0x01,0x02,0x03,0x04,0x05};  /* USB 送信データ */
  uint32_t   size = 5;                                  /* USB 送信データ数 */

  pipe_id = R_usb_hstd_GetPipeID(devadr, USB_EP_BULK, USB_EP_OUT, 1);

  R_usb_hcdc_send_data(pipe_id, send_data, size, &usb_complete);
}

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

3.6.5 R_usb_hcdc_receive_data

USB 受信処理

形式

USB_ER_t	R_usb_hcdc_receive_data(uint16_t pipe_id, uint8_t *table, uint32_t size, USB_UTR_CB_t complete)

引数

pipe_id	パイプ番号
*table	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_OK / USB_ERROR）

解説

USB 受信要求を行います。
指定したパイプで、USB から転送サイズ分のデータ受信完了、またはマックスパケットサイズ未満のデータを受信した場合、コールバック関数が呼出されます。
USB 受信データは、転送データアドレスで指定された領域に格納されます。

補足

USB 受信処理結果はコールバック関数の引数で得られます。

使用例

```
void cdc_data_transfer(uint16_t devadr)
{
  uint16_t   pipe_id;
  uint8_t    receive_data[64];                    /* USB 受信データ格納領域 */
  uint32_t   size = 64;                           /* USB 受信要求サイズ */

  pipe_id = R_usb_hstd_GetPipeID(devadr, USB_EP_BULK, USB_EP_IN, 1);

  R_usb_hcdc_receive_data(pipe_id, receive_data, size, &usb_complete);
}

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

3.6.6 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(uint16_t pipe_id, uint8_t *table, USB_UTR_CB_t complete)

引数

pipe_id	パイプ番号
*table	転送データアドレス
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_OK / USB_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り出して下さい。

補足

1. 転送データ領域は、10 バイト以上確保してください。
2. シリアルステータスのビットパターンは、Table3-9 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で得られます。

使用例

```
void cdc_data_transfer(uint16_t devadr)
{
  uint16_t   pipe_id;
  uint8_t    serial_data[10];              /* USB 受信データ格納領域 */

  pipe_id = R_usb_hstd_GetPipeID(devadr, USB_EP_INT, USB_EP_IN, 0);

  R_usb_hcdc_serial_state_trans(pipe_id, serial_data, &usb_complete);
}


/* R_usb_hcdc_serial_state_trans のコールバック関数例 */
void usb_complete(USB_UTR_t *utr)
{
  uint16_t *status;

  status = (uint16_t *)utr->tranadr;    /* Status set */
  /* [0] bmRequestType/bRequest */
  /* [1] wValue */
  /* [2] wIndex */
  /* [3] wLength */
  /* [4] data : Serial State(UART State bitmap) */
  check_status(status[4]);
}
```

3.6.7 R_usb_hcdc_set_line_coding

SetLineCoding リクエスト処理

形式

USB_ER_t R_usb_hcdc_set_line_coding(uint16_t devadr,
USB_HCDC_LineCoding_t *p_linecoding,
USB_UTR_CB_t complete)

引数

devadr	デバイスアドレス
*p_linecoding	UART 設定パラメータ
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_OK / USB_ERROR)

解説

SetLineCoding リクエスト処理をします。

補足

USB 送信処理結果はコールバック関数の引数で得られます。

使用例

```
 void cdc_class_request(uint16_t devadr)
{
  USB_HCDC_LineCoding_t     cdc_line_coding;

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

  R_usb_hcdc_set_line_coding(devadr, &cdc_line_coding, &cdc_setlinecoding_cb);
 }

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

3.6.8 R_usb_hcdc_get_line_coding

GetLineCoding リクエスト処理

形式

USB_ER_t R_usb_hcdc_get_line_coding(uint16_t devadr,
USB_HCDC_LineCoding_t *p_linecoding,
USB_UTR_CB_t complete)

引数

devadr	デバイスアドレス
*p_linecoding	UART 設定パラメータ
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_OK / USB_ERROR)

解説

GetLineCoding リクエスト処理をします。

補足

USB 送信処理結果はコールバック関数の引数で得られます。

使用例

```
 void cdc_class_request(uint16_t devadr)
{
   USB_HCDC_LineCoding_t     cdc_line_coding;

   R_usb_hcdc_get_line_coding(devadr, &cdc_line_coding, &cdc_getlinecoding_cb);
 }

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

3.6.9 R_usb_hcdc_set_control_line_state

SetControlLineState リクエスト処理

形式

USB_ER_t R_usb_hcdc_set_control_line_state(uint16_t devadr,
uint16_t dtr,
uint16_t rts,
USB_UTR_CB_t complete)

引数

devadr	デバイスアドレス
dtr	RS232 signal DTR
rts	RS232 signal RTS
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_OK / USB_ERROR)

解説

SetControlLineState リクエスト処理をします。

補足

USB 送信処理結果はコールバック関数の引数で得られます。

使用例

```
 void cdc_class_request(uint16_t devadr)
{
   R_usb_hcdc_set_control_line_state(devadr, USB_TRUE, USB_TRUE, &cdc_complete);
}

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

4. サンプルアプリケーション

本章では、HCDC 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。

4.1 動作確認環境

HCDC の動作確認環境を Figure 4-1 に示します。

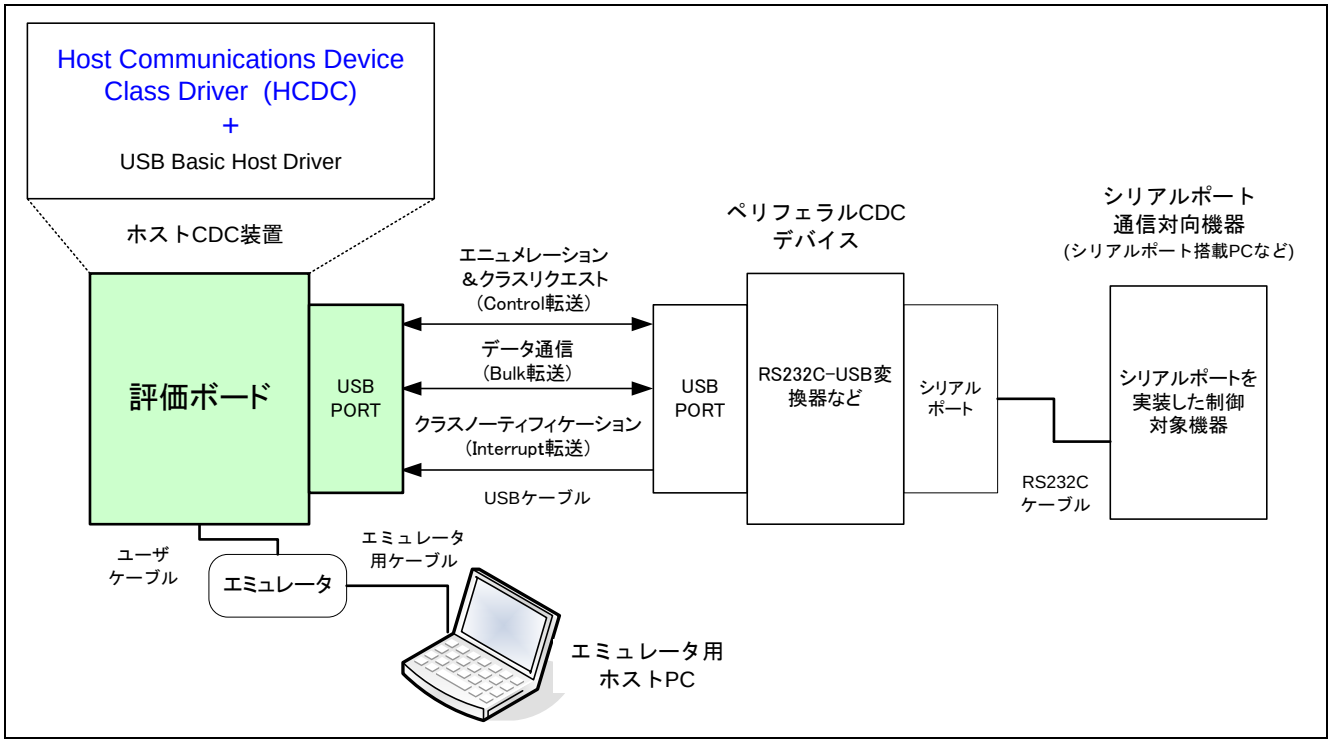


Figure 4-1 動作環境例

4.2 仕様概要

HCDCのサンプルアプリケーション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、受信データを取得する。
2. Bulk Out 転送により受信データを CDC デバイスへ送信する（ループバック）。
3. クラスリクエスト SET_CONTROL_LINE_STATE で RTS、DTR を設定します。
4. クラスリクエスト SET_LINE_CODING で通信速度、データビット数、ストップビット長、パリティビットを設定します。
5. クラスリクエスト GET_LINE_CODING で CDC デバイスの通信設定値を取得できます。
6. 回線状態の変化をコールバック関数によりアプリケーションプログラムに通知します。

4.3 データ転送イメージ

データ転送イメージを Figure 4-2 に示します。

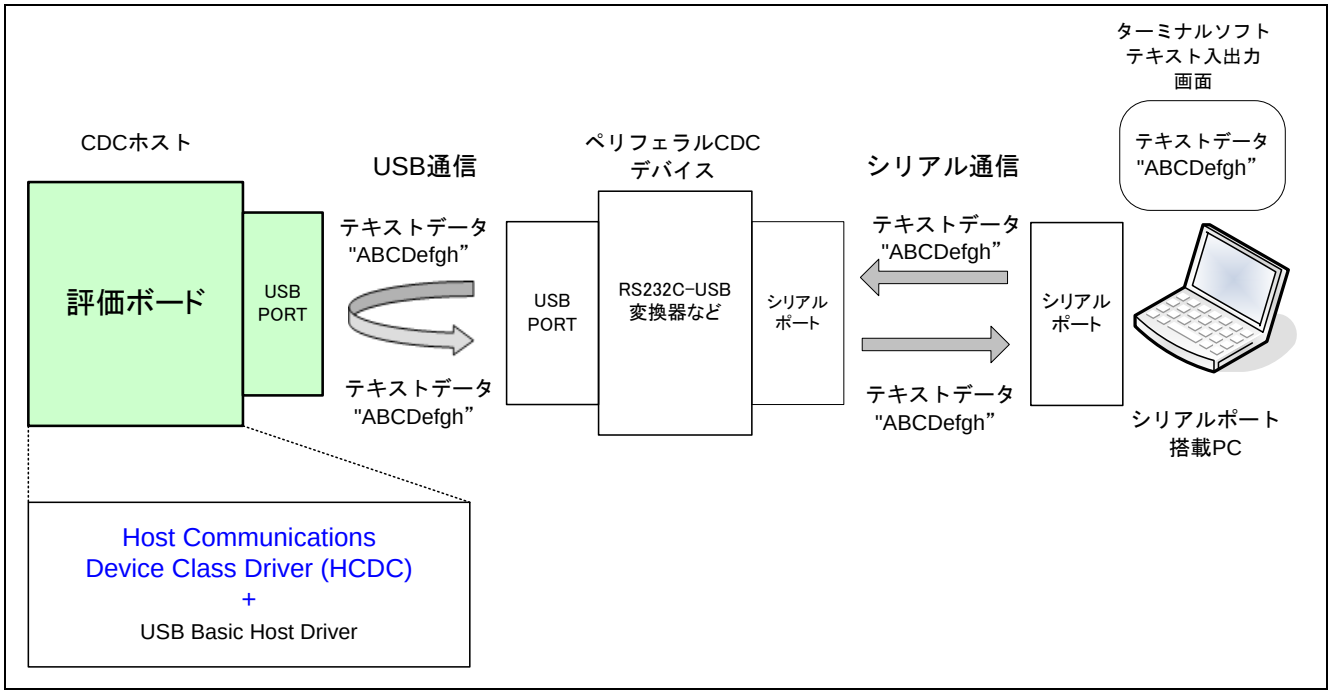


Figure 4-2 データ転送 (ループバック通信) イメージ

4.4 初期設定

初期設定例を以下に示します。

```
void usb_hcdc_apl(void)
{
    /* MCU の設定「4.4.1」参照*/
    usb_mcu_setting();

    /* USB ドライバの設定「4.4.2」参照 */
    R_usb_hstd_MgrOpen();
    R_usb_cstd_SetTaskPri(USB_HUB_TSK,  USB_PRI_3);  // (注)
    R_usb_hhub_Registration(USB_NULL);               // (注)
    cdc_registration();
    R_usb_hcdc_driver_start();

    /* メインルーチン「4.5」参照 */
    usb_hapl_mainloop();
}
```

(注) HUB を使用する場合のみ、本関数を呼び出す必要があります。

4.4.1 MCU 設定

USB モジュールをハードウェアマニュアルの初期設定シーケンスに従って設定し、USB 割り込みハンドラの登録と USB 割り込み許可設定をしています。

4.4.2 USB ドライバ設定

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、登録手順を示します。

1. USB-BASIC-FW の API 関数（R_usb_hstd_MgrOpen()）を呼び出し、HCD タスクと MGR タスクを登録する。
2. HUB クラスドライバ API 関数（R_usb_hhub_Registration()）を呼び出し、HUB タスクを登録する。
3. クラスドライバ登録用構造体（USB_HCDREG_t）の各メンバに情報を設定後、USB-BASIC-FW の API 関数（R_usb_hstd_DriverRegistration()）を呼び出し、クラスドライバを登録する。
4. HCDC クラスドライバの API 関数（R_usb_hcdc_driver_start()）を呼び出し、HCDC タスクを登録する。

USB_HCDREG_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 cdc_registration(void)
{
   /* クラスドライバ登録用構造体 */
   USB_HCDREG_t  driver;

   /* USB の規格で定められたクラスコードを設定 */
   driver.ifclass      = (uint16_t)USB_IFCLS_CDCC;
   /* ターゲットペリフェラルリストを設定 */
   driver.tpl          = (uint16_t*)&usb_gapl_devicetpl; (注 1)
   /* エニュメレーション中に行われるクラスチェック関数を設定 */
   driver.classcheck   = &R_usb_hcdc_class_check;
   /* 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    = &cdc_configured;
   /* USB デバイス切断時に呼ばれる関数を設定 */
   driver.devdetach    = &cdc_detach;
   /* 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   = &cdc_suspend;
   /* 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    = &cdc_resume;

   /* HCD へクラスドライバ情報を登録 */
   R_usb_hstd_DriverRegistration(&driver);
}
```

（注1）TPL(Target Peripheral List)はアプリケーション内で定義してください。TPL については USB Basic Firmware アプリケーションノート(Document No.R01AN2633JJ)を参照してください。

4.5 メインルーチン

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

    /*イベント有無の判定、フラグクリア */
    if(USB_FLGSET == R_usb_cstd_CheckSchedule())
    {
        R_usb_hstd_MgrTask();    /* MGR タスク */
        R_usb_hhub_Task();       /* HUB タスク (注 3) */
        R_usb_hcdc_task();       /* CDC ドライバタスク */
    }
    cdc_main();                  /* APL */
  }
}
```

(注 2)

（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。

（注3） HUB を使用する場合のみ、本関数を呼び出す必要があります。

4.5.1 APL

APL 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のステート (Table 4-1 参照)の確認を行います。このステートは、APL が管理する構造体(4.5.2 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(Table 4-2 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。このイベントは、APL が管理する構造体(4.5.2 参照)のメンバに格納されています。

以下に、APL の処理概要を示します。

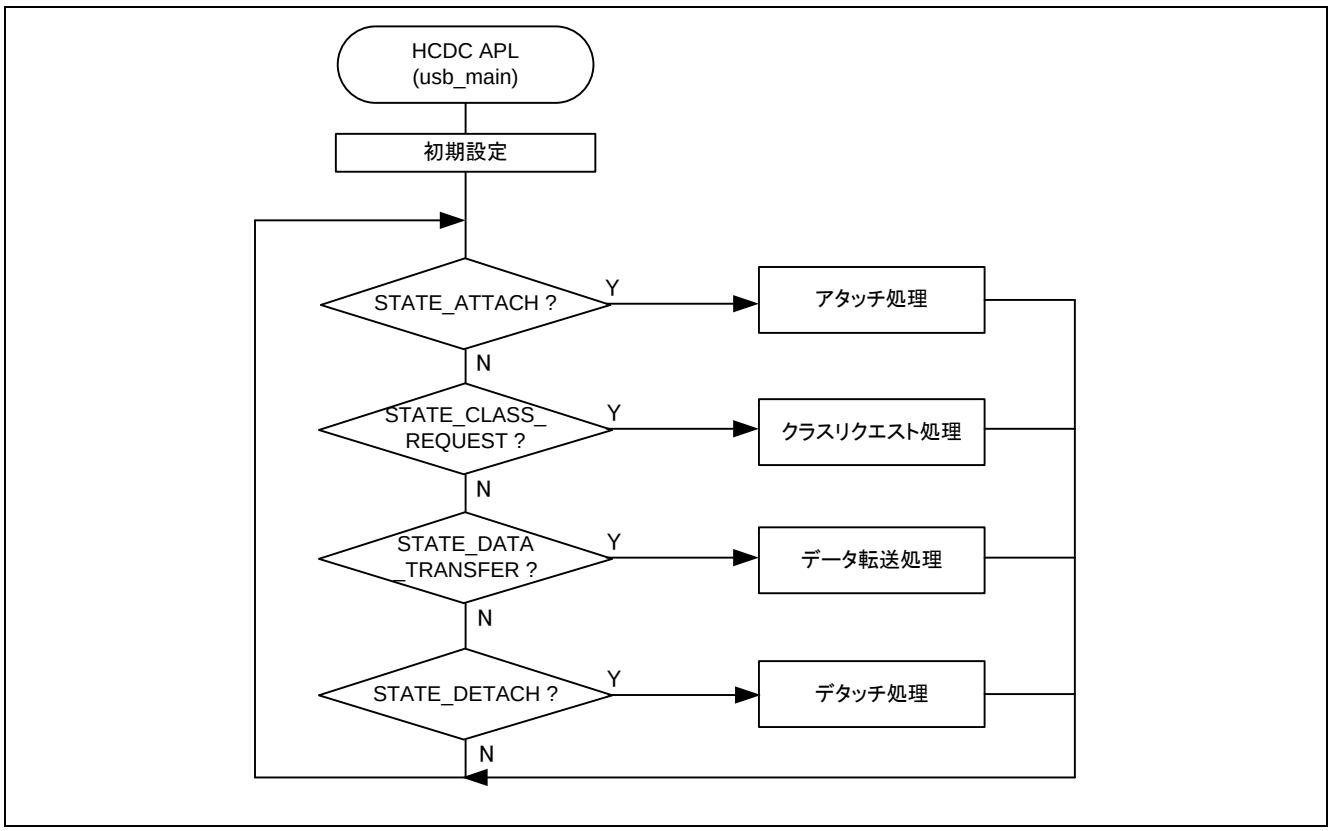


Figure 4-3　メインループ処理

4.5.2 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 no. */
    uint16_t    in_pipe;          /* Use pipe no. */
    uint16_t    out_pipe;         /* Use pipe no. */
    uint16_t    status_pipe;      /* Use pipe no. */
    uint16_t    cr_seq;           /* Class Request Sequence */
    uint16_t    trans_len;        /* TX Length */
    uint8_t     trans_data[ CDC_DATA_LEN + 4 ];      /* RX Data */
    uint8_t     serial_state_data[USB_HCDC_SERIAL_STATE_MSG_LEN];
    USB_HCDC_SerialState_t  serial_state_bitmap;
    USB_HCDC_LineCoding_t   com_parm;    /* Set Line Coding parameter */
} DEV_INFO_t;
```

Table 4-1 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_CLASS_REQUEST	クラスリクエスト処理	EVENT_CLASS_REQUEST_START
		EVENT_CLASS_REQUEST_COMPLETE
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_NOTIFY_READ_START
		EVENT_NOTIFY_READ_COMPLETE

Table 4-2 イベント一覧

イベント	概要
EVENT_CONFIGURD	USB デバイス接続完了
EVENT_CLASS_REQUEST_START	クラスリクエスト要求
EVENT_CLASS_REQUEST_COMPLETE	クラスリクエスト完了
EVENT_USB_READ_START	データリード要求
EVENT_USB_READ_COMPLETE	データリード完了
EVENT_USB_WRITE_START	データライト要求
EVENT_USB_WRITE_COMPLETE	データライト完了
EVENT_COM_NOTIFY_RD_START	ノーティフィケーション受信要求
EVENT_COM_NOTIFY_RD_COMPLETE	ノーティフィケーション受信完了
EVENT_DETACH	デタッチ
EVENT_NONE	イベント無し

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 デバイスが接続され、エニュメレーションが完了したことを通知する処理を行い、ステートを STATE_CLASS_REQUEST 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 デバイスが接続されるまで STATE_ATTACH 状態が継続され、cdc_connect_wait()がコールされます。

③ CDC デバイスが接続され、エニュメレーション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、コールバック関数 cdc_configured()は、USB_HCDREG_t 構造体のメンバ devconfig に設定した関数です。

④ イベント EVENT_CONFIGURD では、ステートを STATE_CLASS_REQUEST に変更し、イベント EVENT_CLASS_REQUEST_START を発行します。

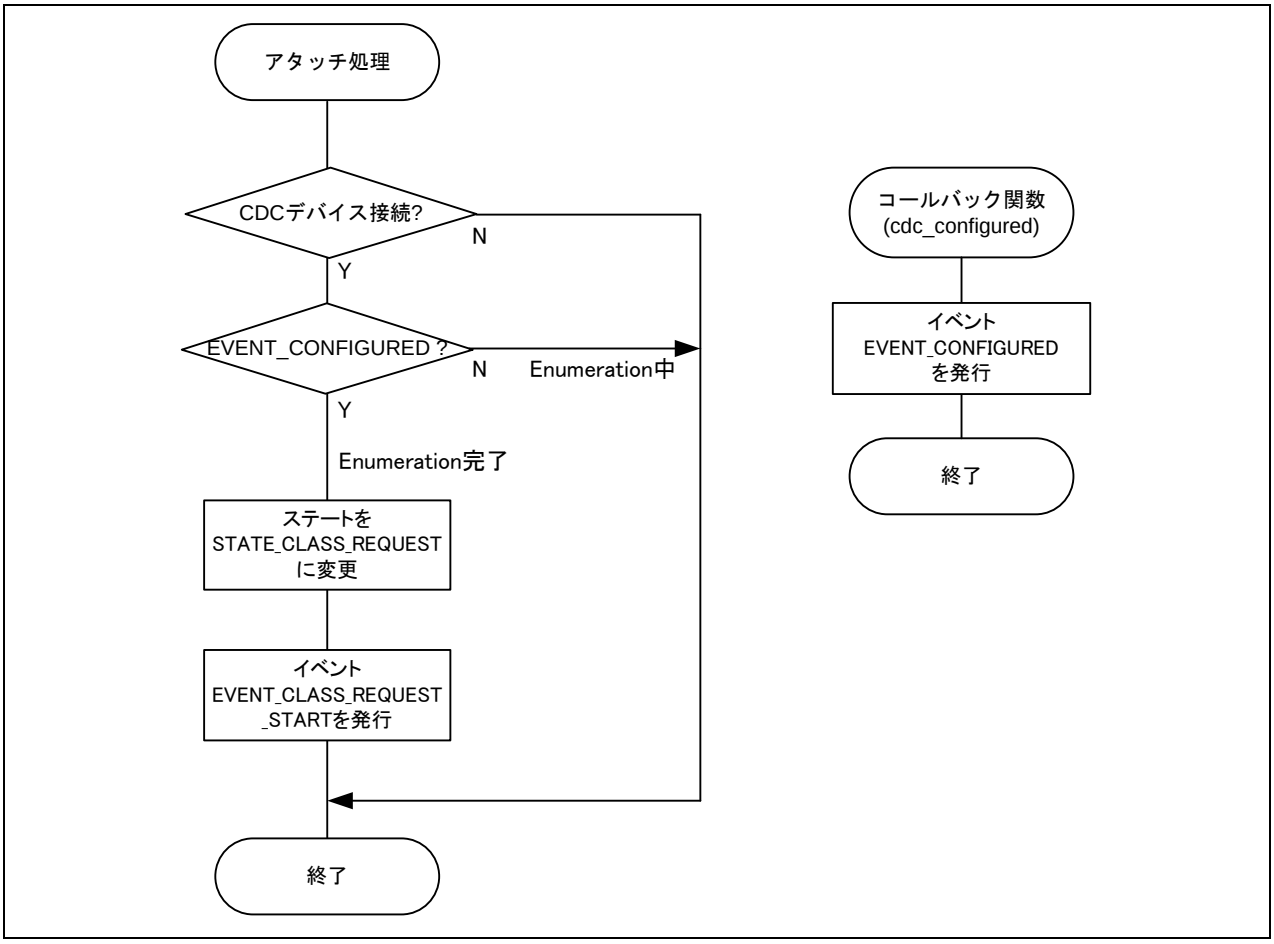


Figure 4-4 アタッチ処理概略フロー

2) クラスリクエスト処理 (STATE_CLASS_REQUEST)

== 概要 ==

このステートでは、CDC デバイスに対しクラスリクエストを送信する処理を行います。特定のクラスリクエスト送信が完了するとステートを STATE_DATA_TRANSFER に変更します。

== 内容 ==

① このステートでは、EVENT_CLASS_REQUEST_START が処理され、USB ドライバに対しクラスリクエスト送信要求を行います。

② クラスリクエスト送信処理が完了するとコールバック関数 cdc_class_request_complet()がコールされます。このコールバック関数によりイベント EVENT_CLASS_REQUEST_COMPLETE を発行します。

③ 上記①と②の処理を繰り返し行い、クラスリクエストを SetControlLineState、SetLineCoding、GetLineCoding の順番で CDC デバイスに送信します。

④ クラスリクエスト GetLineCoding の送信が完了するとステートを STATE_DATA_TRANSFER に変更し、イベント EVENT_USB_READ_START を発行します。

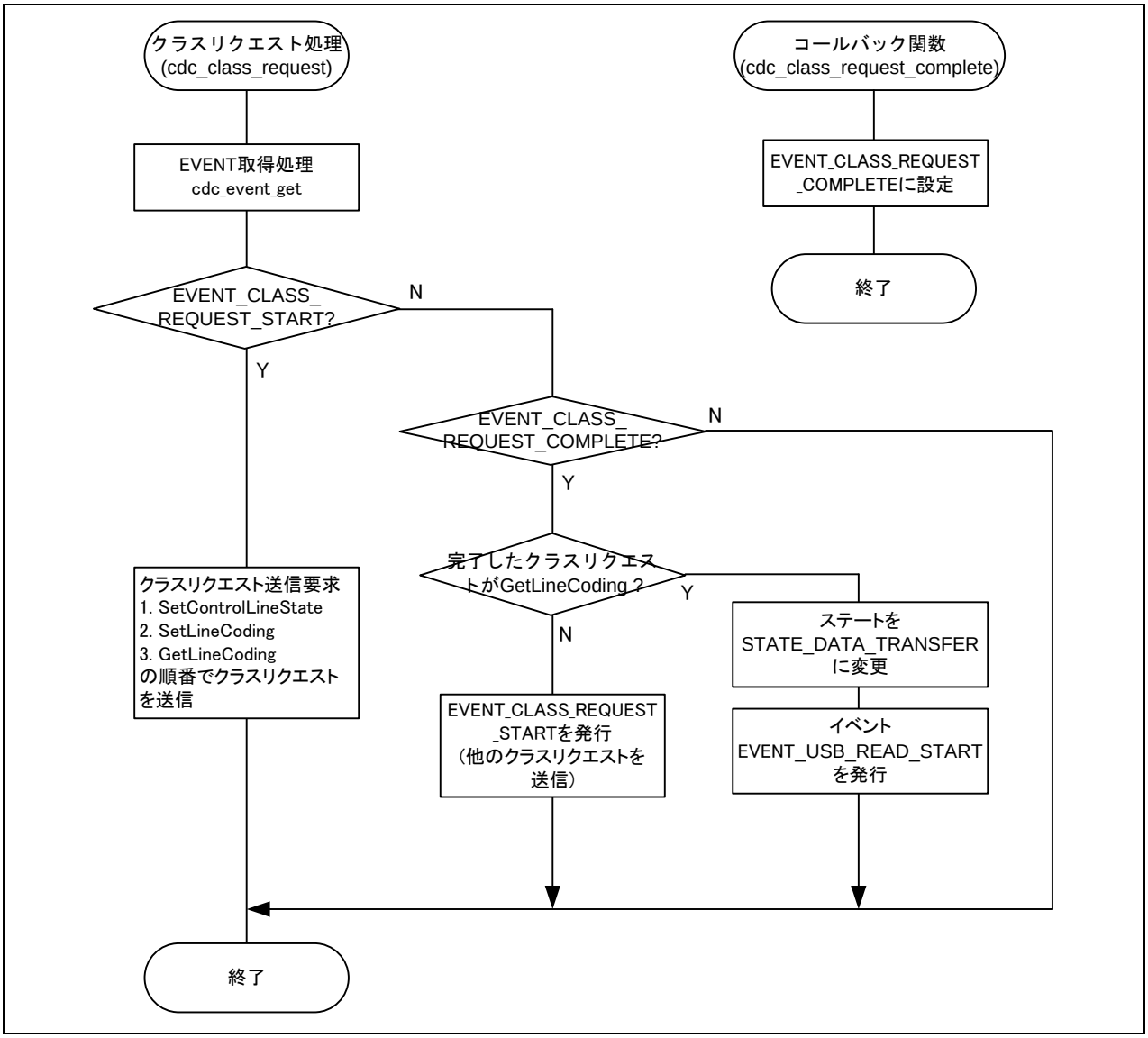


Figure 4-5 クラスリクエスト処理概略フロー

3) データ転送処理 (STATE_DATA_TRANSFER)

== 概要 ==

このステートでは、CDC デバイスからのデータを受信し、その受信したデータをそのまま CDC デバイスに対し送信するループバック処理を行います。

== 内容 ==

① CDC デバイスに対するクラスリクエストの送信が完了するとステートは STATE_DATA_TRANSFER に遷移します。このステートでは、EVENT_USB_READ_START が処理され、USB ドライバに対しデータ受信処理要求を行います。

② データリード処理が完了するとコールバック関数 cdc_receive_complete()がコールされます。このコールバック関数によりイベント EVENT_USB_READ_COMPLETE を発行します。

③ EVENT_USB_READ_COMPLETE では、イベント EVENT_USB_WRITE_START をセットします。

④ EVENT_USB_WRITE_START では、①で受信したデータを CDC デバイスに送信するため USB ドライバに対しデータライト要求を行います。

⑤ データライト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はイベント EVENT_USB_WRITE_COMPLETE を発行します。

⑥ EVENT_USB_WRITE_COMPLETE では、イベント EVENT_USB_READ_START を発行し、次のループで、①が再度処理され、①から⑥が繰り返し処理されます。

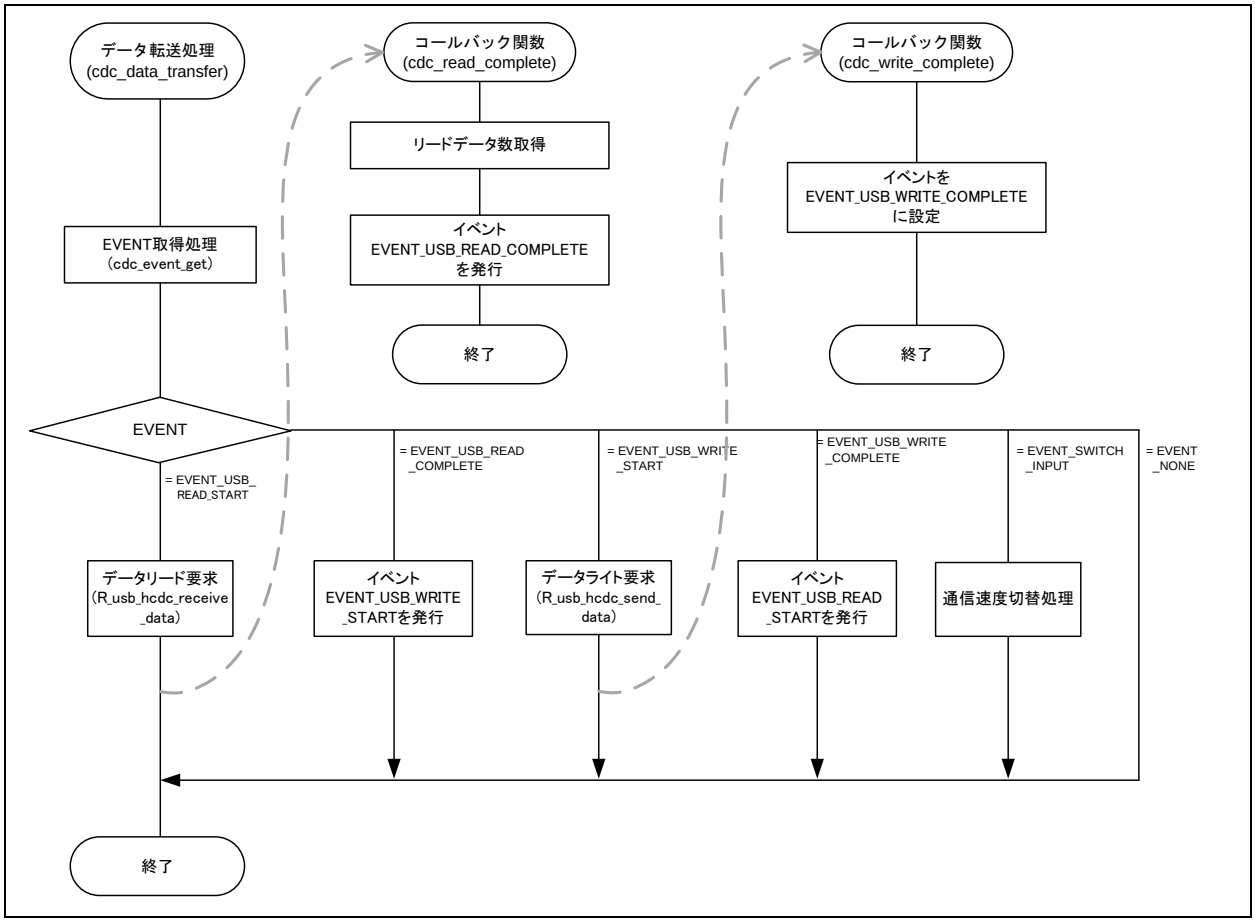


Figure 4-6 データ転送処理概略フロー

4) デタッチ処理 (STATE_DETACH)

接続された CDC デバイスが切断されると 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、変数の初期化、アプリケーションプログラムの初期化およびステートを STATE_ATTACH に変更するなどの処理を行います。なお、コールバック関数 cdc_detach()は、USB_HCDREG_t 構造体のメンバ devdetach に設定した関数です。

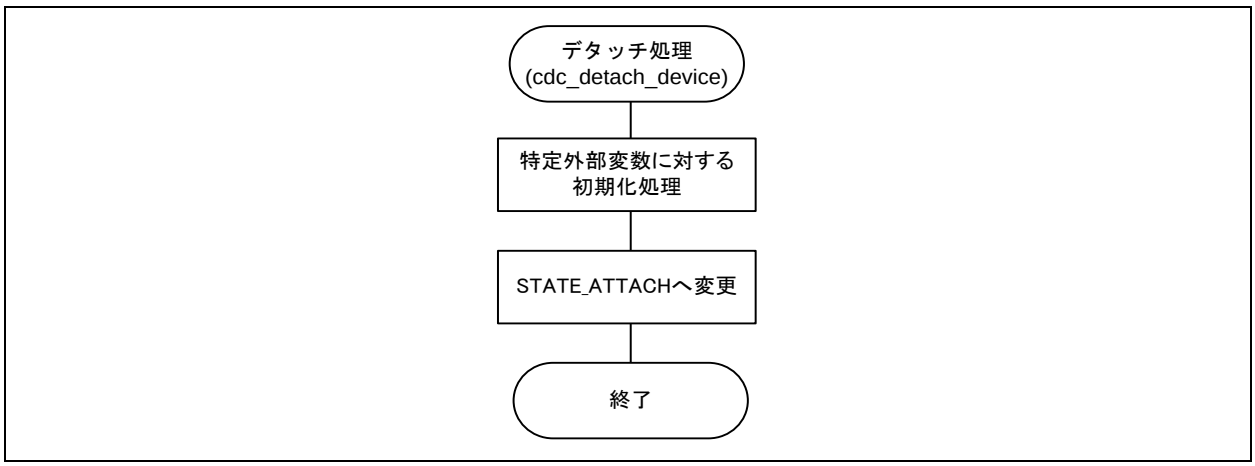


Figure 4-7 デタッチ処理概略フロー

Appendix A. 初期設定の変更点

USB-BASIC-FW を動作させるために「RZ/T1 グループ初期設定 Rev.1.30」を変更しています。
サンプルプログラムは、IAR embedded workbench for ARM(以下、EWARM)と DS-5 と e^2 studio の環境をサポートしています。
本章では、変更点について記述します。

フォルダとファイル

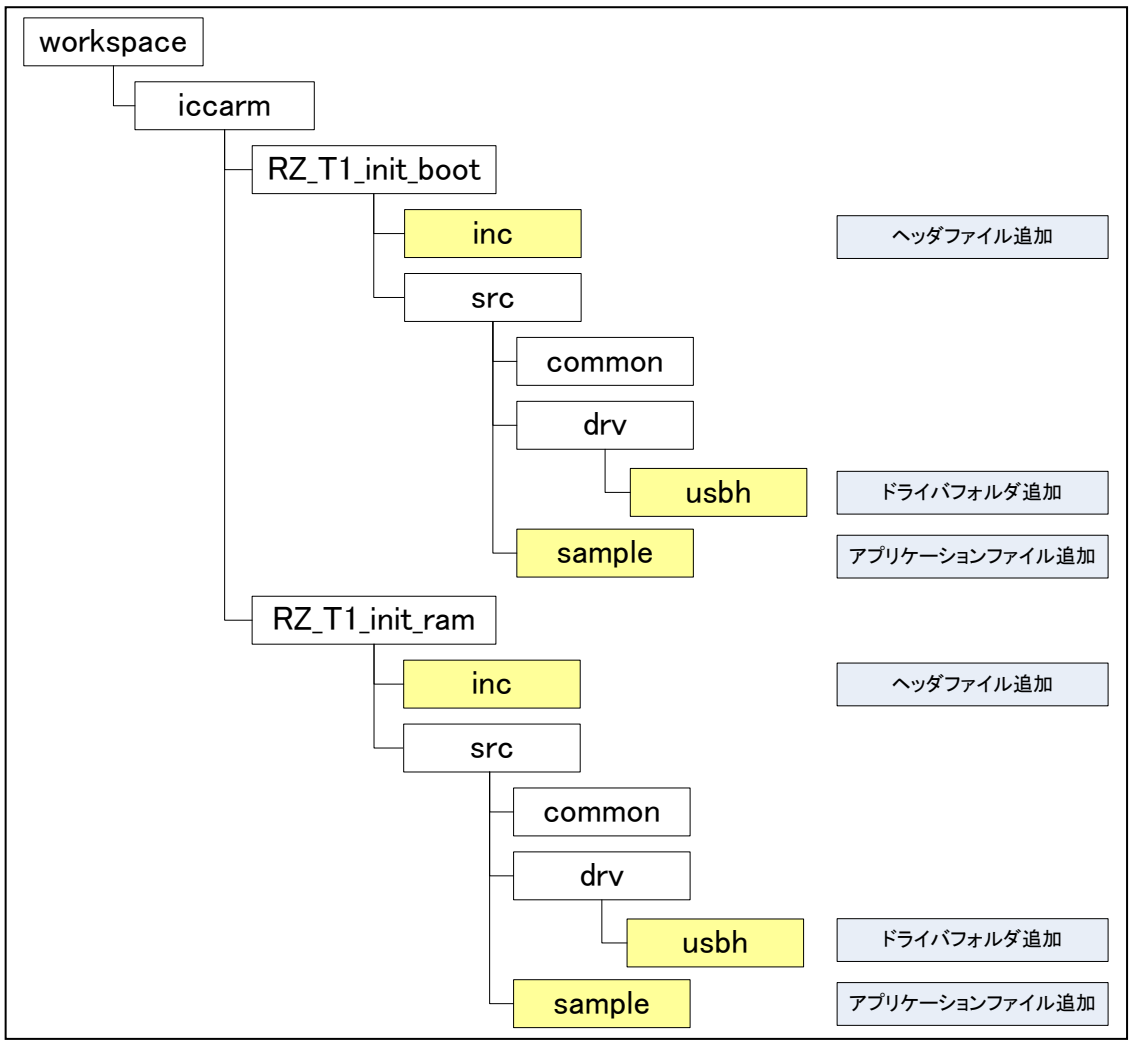
「RZ/T1 グループ初期設定 Rev.1.30」では、開発環境とブート方法によってフォルダ構成が異なります。
全ての開発環境とブート方法の各フォルダに対して、下記の変更をしています。

・”inc”フォルダに下記のファイルを追加
　r_usb_basic_config.h
　r_usb_basic_if.h
　r_usb_hcdc_config.h
　r_usb_hcdc_if.h

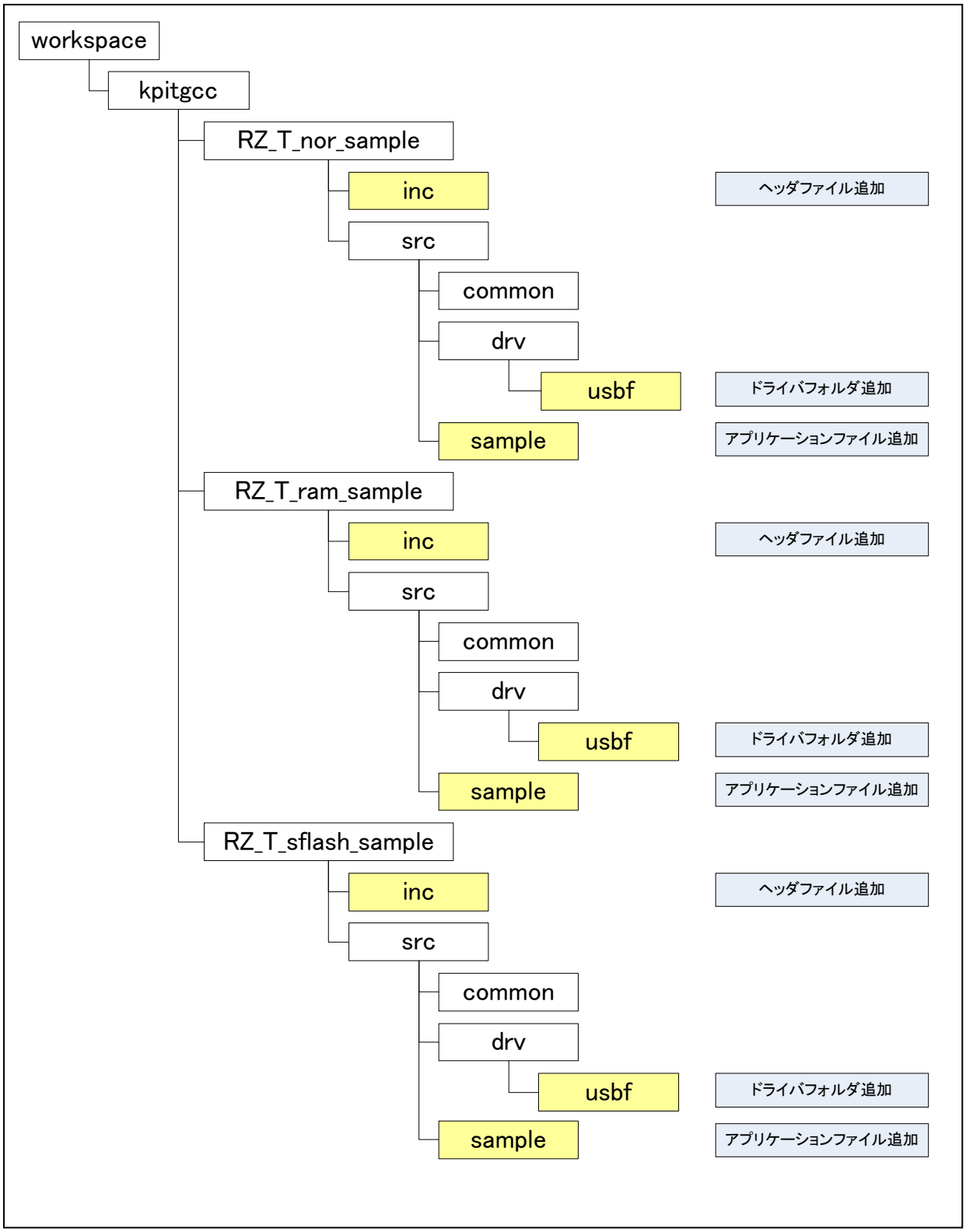
・”sample”フォルダに下記のファイルを追加
　r_usb_main.c
　r_usb_hcdc_apl.c
　r_usb_hcdc_apl.h

・”drv”フォルダに usbh フォルダと usbh フォルダ以下のファイルを追加

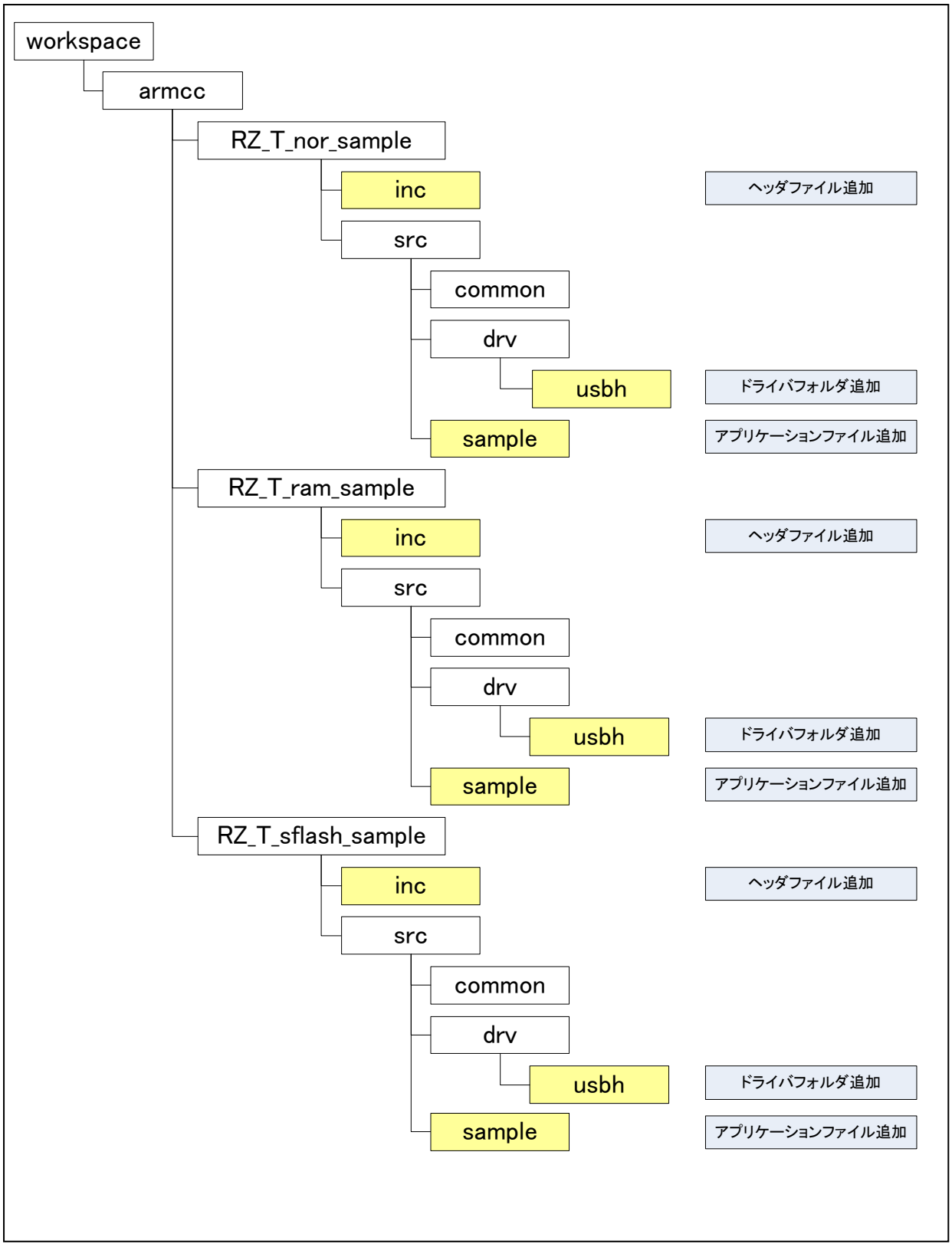
以下に EWARM のフォルダ構成を示します。



以下に e² studio のフォルダ構成を示します。



以下に DS-5 のフォルダ構成を示します。



セクション

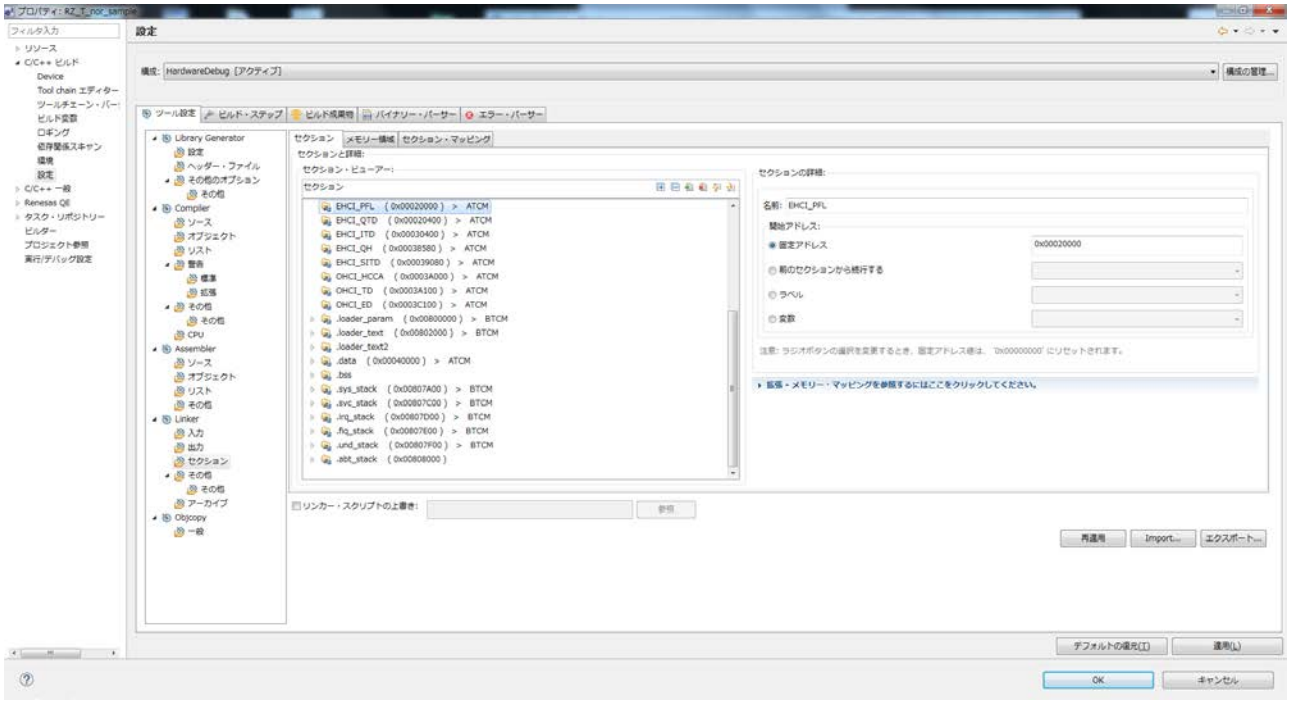
コード領域とデータ領域のセクションサイズを変更して、以下のセクションを追加しています。

セクション名	アドレス	割り当てる変数	ファイル
EHCI_PFL	0x00020000	ehci_PeriodicFrameList	r_usb_hEhciMemory.c
EHCI_QTD	0x00020400	ehci_Qtd	
EHCI_ITD	0x00030400	ehci_Itd	
EHCI_QH	0x00038580	ehci_Qh	
EHCI_SITD	0x00039080	ehci_Sitd	
OHCI_HCCA	0x0003A000	ohci_hcca	r_usb_hOhciMemory.c
OHCI_TD	0x0003A100	ohci_TdMemory	
OHCI_ED	0x0003c100	ohci_EdMemory	

e² studio

e² studio の設定画面でセクションを設定しています。
変更内容は下記の通りです。
・.data セクションの固定アドレスを 0x0007F000 から 0x00040000 に変更
・EHCI と OHCI のセクション設定を追加

[プロジェクト] → [プロパティ] → [C/C++ ビルド] → [設定] → [セクション] で参照できます。



コード内の変数定義は下記の通りです。

r_usb_hEhciMemory.c

```
#ifdef __GNUC__
static uint32_t         ehci_PeriodicFrameList[ USB_EHCI_PFL_SIZE ]
                __attribute__ ((section ("EHCI_PFL")));
static USB_EHCI_QH      ehci_Qh[ USB_EHCI_NUM_QH ]
                __attribute__ ((section ("EHCI_QH")));
static USB_EHCI_QTD     ehci_Qtd[ USB_EHCI_NUM_QTD ]
                __attribute__ ((section ("EHCI_QTD")));
static USB_EHCI_ITD     ehci_Itd[ USB_EHCI_NUM_ITD ]
                __attribute__ ((section ("EHCI_ITD")));
static USB_EHCI_SITD    ehci_Sitd[ USB_EHCI_NUM_SITD ]
                __attribute__ ((section ("EHCI_SITD")));
#endif  /* __GNUC__ */
```

r_usb_hOhciMemory.c

```
#ifdef __GNUC__
static USB_OHCI_HCCA_BLOCK              ohci_hcca
                __attribute__ ((section ("OHCI_HCCA")));
static USB_OHCI_HCD_TRANSFER_DESCRIPTOR ohci_TdMemory[USB_OHCI_NUM_TD]
                __attribute__ ((section ("OHCI_TD")));
static USB_OHCI_HCD_ENDPOINT_DESCRIPTOR ohci_EdMemory[USB_OHCI_NUM_ED]
                __attribute__ ((section ("OHCI_ED")));
#endif  /* __GNUC__ */
```

EWARM

EWARM では、リンカ設定ファイル(.icf ファイル)でセクションを設定しています。

RAM 領域の開始アドレスを 0x00070000 から 0x00040000 に、USER_PRG 領域の終了アドレスを 0x0001FFFF に変更しています。

```
define symbol __ICFEDIT_region_RAM_start__   = 0x00040000;

define symbol __region_USER_PRG_end__        = 0x0001FFFF;
```

EHCI と OHCI を固定アドレスにするため、メモリリージョン定義を追加しています。

```
define region EHCI_MEM1_region = mem:[from 0x00020000 to 0x000203FF];
define region EHCI_MEM2_region = mem:[from 0x00020400 to 0x00039FFF];

define region OHCI_MEM1_region = mem:[from 0x0003A000 to 0x0003A0FF];
define region OHCI_MEM2_region = mem:[from 0x0003A100 to 0x0003FFFF];

do not initialize  { section EHCI_PFL, section EHCI_QH, section EHCI_QTD, section EHCI_ITD, section
EHCI_SITD, section OHCI_HCCA, section OHCI_TD, section OHCI_ED };

place in EHCI_MEM1_region { section EHCI_PFL };
place in EHCI_MEM2_region { section EHCI_QH, section EHCI_QTD, section EHCI_ITD, section EHCI_SITD };

place in OHCI_MEM1_region { section OHCI_HCCA };
place in OHCI_MEM2_region { section OHCI_TD, section OHCI_ED };
```

コード内の変数定義は下記の通りです。

r_usb_hEhciMemory.c

```
#ifdef __ICCARM__
#pragma location="EHCI_PFL"
static uint32_t                 ehci_PeriodicFrameList[ USB_EHCI_PFL_SIZE ];
#pragma location="EHCI_QH"
static USB_EHCI_QH              ehci_Qh[ USB_EHCI_NUM_QH ];
#pragma location="EHCI_QTD"
static USB_EHCI_QTD             ehci_Qtd[ USB_EHCI_NUM_QTD ];
#pragma location="EHCI_ITD"
static USB_EHCI_ITD             ehci_Itd[ USB_EHCI_NUM_ITD ];
#pragma location="EHCI_SITD"
static USB_EHCI_SITD            ehci_Sitd[ USB_EHCI_NUM_SITD ];
#endif  /* __ICCARM__ */
```

r_usb_hOhciMemory.c

```
#ifdef __ICCARM__
#pragma location="OHCI_HCCA"
static USB_OHCI_HCCA_BLOCK                  ohci_hcca;
#pragma location="OHCI_TD"
static USB_OHCI_HCD_TRANSFER_DESCRIPTOR     ohci_TdMemory[USB_OHCI_NUM_TD];
#pragma location="OHCI_ED"
static USB_OHCI_HCD_ENDPOINT_DESCRIPTOR     ohci_EdMemory[USB_OHCI_NUM_ED];
#endif  /* __ICCARM__ */
```

DS-5

DS-5 では、scatter ファイルでセクションを設定しています。

RAM 領域の開始アドレスを 0x00040000 に変更、0 クリアするメモリ(ZI)領域を DATA 領域に続けて確保するよう修正しています。

```
    DATA            0x00040000    UNINIT
    {
        * (+RW)
    }
    BSS             +0
    {
        * (+ZI)
    }
```

EHCI と OHCI を固定アドレスにするため、メモリ定義を追加しています。

```
        EHCI_PERIODIC_FRAMELIST 0x00020000    0x400
        {
                r_usb_hEhciMemory.o(EHCI_PFL)
        }
        EHCI_QTD        +0
        {
                r_usb_hEhciMemory.o(ehci_Qtd)
        }
        EHCI_ITD        +0
        {
                r_usb_hEhciMemory.o(ehci_Itd)
        }
        EHCI_QH         +0
        {
                r_usb_hEhciMemory.o(ehci_Qh)
        }
        EHCI_SITd       +0
        {
                r_usb_hEhciMemory.o(ehci_Sitd)
        }
        OHCI_HCCA       0x0003A000      0x100
        {
                r_usb_hOhciMemory.o(OHCI_HCCA)
        }
        OHCI_TDMEMORY   +0
        {
                r_usb_hOhciMemory.o(OHCI_TD)
        }
        OHCI_EDMEMORY   +0
        {
                r_usb_hOhciMemory.o(OHCI_ED)
        }
```

コード内の変数定義は下記の通りです。

r_usb_hEhciMemory.c

```
#ifdef __CC_ARM
#pragma arm section zidata = "EHCI_PFL"
static uint32_t         ehci_PeriodicFrameList[ USB_EHCI_PFL_SIZE ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_QH"
static USB_EHCI_QH      ehci_Qh[ USB_EHCI_NUM_QH ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_QTD"
static USB_EHCI_QTD     ehci_Qtd[ USB_EHCI_NUM_QTD ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_ITD"
static USB_EHCI_ITD     ehci_Itd[ USB_EHCI_NUM_ITD ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_SITD"
static USB_EHCI_SITD    ehci_Sitd[ USB_EHCI_NUM_SITD ];
#pragma arm section zidata
#endif
```

r_usb_hOhciMemory.c

```
#ifdef __CC_ARM
#pragma arm section zidata = "OHCI_HCCA"
static USB_OHCI_HCCA_BLOCK                  ohci_hcca;
#pragma arm section zidata
#pragma arm section zidata = "OHCI_TD"
static USB_OHCI_HCD_TRANSFER_DESCRIPTOR    ohci_TdMemory[USB_OHCI_NUM_TD];
#pragma arm section zidata
#pragma arm section zidata = "OHCI_ED"
static USB_OHCI_HCD_ENDPOINT_DESCRIPTOR    ohci_EdMemory[USB_OHCI_NUM_ED];
#pragma arm section zidata
#endif
```

USB-BASIC-FW 関数の呼び出し

¥src¥sample¥int_main.c の main()に USB-BASIC-FW の usbh_main()の呼び出しを追加しています。

```
extern void usbh_main(void);

int main (void)
{
    /* Initialize the port function */
    port_init();

    /* Initialize the ECM function */
    ecm_init();

    /* Initialize the ICU settings */
    icu_init();

    /* USBh main */
    usbh_main();

    while (1)
    {
        /* Toggle the PF7 output level(LED0) */
        PORTF.PODR.BIT.B7 ^= 1;

        soft_wait();  // Soft wait for blinking LED0

    }

}
```

ホームページとサポート窓口

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

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

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Aug 21, 2015	—	初版発行
1.10	Dec 25, 2015	29	Appendix A 追加
1.20	Feb 29, 2016	31,35,36	DS-5 関連情報追加
1.30	Dec 07, 2017	—	RZ/T1 初期設定 Ver 1.30 に対応

製品ご使用上の注意事項

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

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子
   （または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、その他の不適切に使用しないでください。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）、交通制御（信号）、大規模通信機器、
   金融端末基幹システム、各種安全制御装置等

   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することはできません。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損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を、(1)核兵器、化学兵器、生物兵器等の大量破壊兵器およびこれらを運搬することができるミサイル（無人航空機を含みます。）の開発、設計、製造、使用もしくは貯蔵等の目的、(2)通常兵器の開発、設計、製造または使用の目的、または(3)その他の国際的な平和および安全の維持の妨げとなる目的で、自ら使用せず、かつ、第三者に使用、販売、譲渡、輸出、賃貸もしくは使用許諾しないでください。
   当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様の転売、貸与等により、本書（本ご注意書きを含みます。）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、当社は一切その責任を負わず、お客様にかかる使用に基づく当社への請求につき当社を免責いただきます。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載された情報または当社製品に関し、ご不明点がある場合には、当社営業にお問い合わせください。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が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。

(Rev.3.0-1 2016.11)

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

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

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

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