RZ/T1 グループ

R01AN2636JJ0130
Rev.1.30
Dec 07, 2017

USB Host Human Interface Device Class Driver (HHID)

要旨

本アプリケーションノートでは、Host 用ヒューマンインターフェースデバイスクラスドライバ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 HHID 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 Human Interface Devices Version 1.1
3. HID Usage Tables Version 1.1
   【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

HHID は、USB-BASIC-FW と組み合わせることで、USB Host ヒューマンインターフェースデバイスクラスドライバとして動作します。

以下に、本モジュールがサポートしている機能を示します。
- 接続されたHIDデバイス（USBマウス、USBキーボード）とデータ通信
- 接続されたHIDデバイスへのHIDクラスリクエスト発行
- 複数のHIDデバイスの接続

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。）
- HHID はレポートディスクリプタ解析をしません。デバイスから取得したインターフェースプロトコル（Keyboard/Mouse）からレポートフォーマットを決定します。
- Interrupt Out 転送を行う HID デバイスは動作対応していません。

用語一覧

APL	:	Application program
HCD	:	Host control driver of USB-BASIC-FW
HDCD	:	Host device class driver (device driver and USB class driver)
HHID	:	USB Host Human Interface Device Class Driver
HID	:	Human interface device class
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、HHID 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。

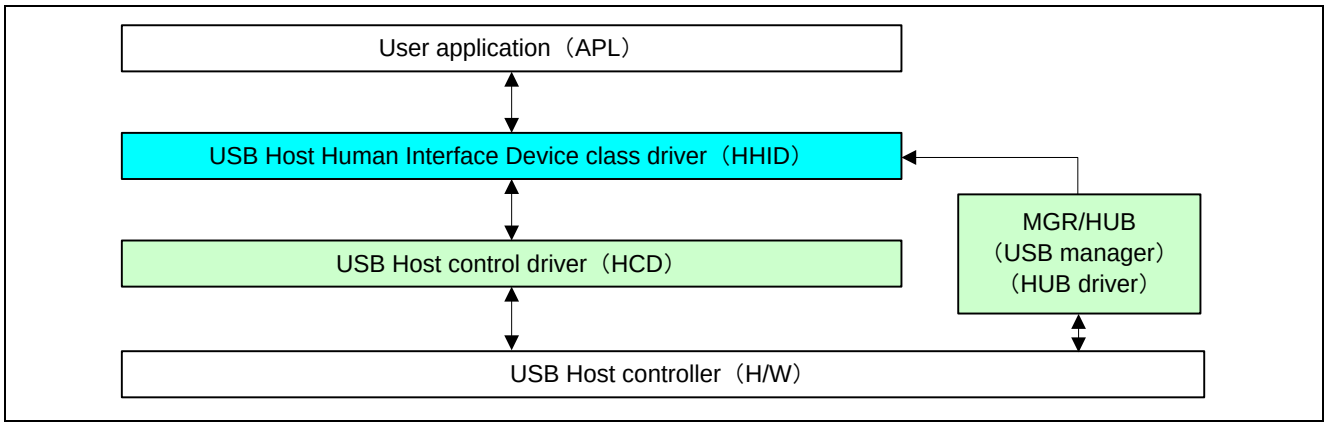


Figure 2-1 モジュール構成図

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（システムに合わせてご用意ください）
HHID	ホスト HID クラスドライバ ・ HID 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 HHID 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 ホストヒューマンインターフェースデバイスクラスドライバ（HHID）

HHID は、ヒューマンインターフェースデバイスクラス仕様に準拠しています。

3.1 基本機能

HHID の基本機能を以下に示します。

1. HID デバイスの照合
2. HID デバイスへのクラスリクエスト通知
3. HID デバイスとのデータ通信

3.2 クラスリクエスト

Table 3-1 に HHID で対応しているクラスリクエストを示します。

Table 3-1 HID クラスリクエスト

リクエスト	コード	説明
USB_SET_REPORT	0x09	USB デバイスにレポートを通知する
USB_SET_PROTOCOL	0x0B	USB デバイスにプロトコルを通知する

HHID 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。

a). SetReport リクエストフォーマット

Table 3-2 に SetReport リクエストのフォーマットを示します。
コントロール転送によりレポートデータをデバイスに送信します。

Table 3-2 SetReport フォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_REPORT (0x09)	ReportType & ReportID	Interface	ReportLength	Report

b). SetProtocol リクエストフォーマット

Table 3-3 に SetProtocol リクエストのフォーマットを示します。
プロトコル（ブートプロトコル又はレポートプロトコル）の設定を行います。

Table 3-3 SetProtocol フォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_PROTOCOL (0x0B)	0(BootProtocol) / 1(ReportProtocol)	Interface	0x0000	Not applicable

3.2.1 レポートフォーマット

HID で扱うレポートフォーマットを以下に記します。

(1). 受信レポートフォーマット

Table 3-4 に HID デバイスから通知される受信レポートフォーマットを示します。
インタラプト IN 転送及び、クラスリクエスト GetReport により受信します。

Table 3-4 受信レポートフォーマット

offset (データ長)	Keyboard モード (8 バイト)	Mouse モード (3 バイト)
0(Top Byte)	Modifier keys	b0 : Button 1 b1 : Button 2 b2 : Button 3 b2-7 : Reserved
+1	Reserved	X displacement
+2	Keycode 1	Y displacement
+3	Keycode 2	−
+4	Keycode 3	−
+5	Keycode 4	−
+6	Keycode 5	−
+7	Keycode 6	−

(2). 送信レポートフォーマット

Table 3-5 に HID デバイスに通知する送信レポートフォーマットを示します。
クラスリクエスト SetReport で送信を行います。

Table 3-5 送信レポートフォーマット

offset (データ長)	Keyboard モード (1 バイト)	Mouse モード (非サポート)
0(Top Byte)	b0：NumLock b1：CapsLock b2：ScrollLock b3：Compose b4：Kana	−
+1～+16	−	−

(3). 注意事項

データ通信で用いるレポートフォーマットはレポートディスクリプタに従う必要があります。本ドライバではレポートディスクリプタの取得と解析は行わず、インターフェースプロトコルコードに従ってレポートフォーマットを決定しています。HID クラス仕様にあわせて変更してください。

3.3 スケジューラ設定

Table 3-6 に HHID のスケジューラ設定を示します。

Table 3-6 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_hhid_task	USB_HHID_TSK	USB_PRI_3	USB_HHID_MBX	USB_HHID_MPL	HHID 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

3.4 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_hhid_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_hhid_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。

Table3-7 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

Table 3-8 に HHID API 一覧を示します。

Table 3-8 HHID API 関数一覧

関数名	機能概要
R_usb_hhid_task	HHID タスク
R_usb_hhid_class_check	ディスクリプタチェック処理
R_usb_hhid_TransferEnd	USB データ転送強制終了
R_usb_hhid_ChangeDeviceState	デバイス状態変更処理
R_usb_hhid_protcol_code	プロトコルコード取得
R_usb_hhid_GetMaxPacketSize	MaxPacketSize 取得
R_usb_hhid_driver_start	HHID ドライバ起動
R_usb_hhid_set_report	SET_REPORT 送信
R_usb_hhid_set_protocol	SET_PROTOCOL 送信
R_usb_hhid_transfer_start	USB データ転送要求

3.4.1 R_usb_hhid_task

HHID タスク

形式

void	R_usb_hhid_task(void)

引数

―	―

戻り値

―	―

解説

HHID 処理タスクです。
APL から要求された処理を行い、APL 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_hhid_task();          /* HHID Task */
   }
  }
}
```

3.4.2 R_usb_hhid_class_check

ディスクリプタ情報取得

形式

void	R_usb_hhid_class_check(uint16_t **table)

引数

**table	ポインタテーブル [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。
スタートアップ時の HHID 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してください。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、HID デバイスのディスクリプタ情報を取得して、パイプ情報を登録します。

補足

—

使用例

```
void  hid_registration(void)
{
  USB_HCDREG_t  driver;

  driver.classcheck   = &R_usb_hhid_class_check;

  R_usb_hstd_DriverRegistration(ptr, (USB_HCDREG_t*)&driver);
}
```

3.4.3 R_usb_hhid_TransferEnd

USB データ転送強制終了

形式

USB_ER_t R_usb_hhid_TransferEnd(uint16_t pipe)

引数

pipe	パイプ番号

戻り値

USB_OK	正常終了
USB_ERROR	失敗

解説

本 API は、HCD に指定したパイプ番号のデータ転送強制終了要求をします。

補足

―

使用例

```
void  usb_smp_task(void)
{
  USB_ER_t err ;

  /* 転送終了要求 */
  err = R_usb_hhid_TransferEnd(IN_PIPE);

  return err;
}
```

3.4.4 R_usb_hhid_ChangeDeviceState

デバイス状態変更処理

形式

void R_usb_hhid_ChangeDeviceState(uint16_t msginfo, uint16_t devadr)

引数

msginfo	通信ステータス
devadr	デバイスアドレス

戻り値

―	―

解説

本 API は、デバイスの状態変更を行います。
msginfo に以下の値を設定し、本関数を呼び出すことで USB デバイスステートの変更を HCD に要求します。

msginfo	機能
USB_DO_GLOBAL_SUSPEND	サスペンド遷移要求
USB_DO_GLOBAL_RESUME	レジューム遷移要求

補足

―

使用例

```
void usb_smp_task(void)
{
  /* 状態遷移要求 */
  R_usb_hhid_ChangeDeviceState(USB_DO_GLOBAL_SUSPEND);
}
```

3.4.5 R_usb_hhid_protocol_code

プロトコルコード取得

形式

uint8_t R_usb_hhid_protocol_code(uint16_t devadr)

引数

devadr デバイスアドレス

戻り値

－ USB デバイスのプロトコルコード（bInterfaceProtocol）

解説

接続された USB デバイスのプロトコルコード（bInterfaceProtocol）を取得します。

補足

bInterfaceProtocol は、受信した Interface Descriptor 含まれます。

使用例

```
void  usb_smp_task(void)
{
  uint8_t    protocol;

  protocol = R_usb_hhid_protocol_code(devadr);
}
```

3.4.6 R_usb_hhid_GetMaxPacketSize

MaxPacketSize 取得

形式

uint16_t R_usb_hhid_GetMaxPacketSize(uint16_t devadr)

引数

devadr デバイスアドレス

戻り値

uint16_t データ受信時に使用する Endpoint の MaxPacketSize

解説

デバイスからデータを受信する際に使用する Endpoint の MaxPacketSize を取得します。

補足

MaxPacketSize は Endpoint Descriptor に含まれます。

使用例

```
void  usb_smp_task(void)
{
  uint16_t      maxps;

  maxps = R_usb_hhid_GetMaxPacketSize(devadr);
}
```

3.4.7 R_usb_hhid_driver_start

HHID ドライバ起動

形式

void	R_usb_hhid_driver_start(void)

引数

―	―

戻り値

―	―

解説

HHID 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

初期設定時にユーザアプリケーションで本 API を呼び出してください。

使用例

```
void usb_hstd_task_start(void)
{
  R_usb_hhid_driver_start();      /* Host Class Driver Task Start Setting */
}
```

3.4.8 R_usb_hhid_set_report

SET_REPORT 送信

形式

USB_ER_t R_usb_hhid_set_report(uint16_t devadr, uint8_t *p_data, uint16_t length, USB_UTR_CB_t complete)

引数

devadr	デバイスアドレス
*p_data	SET_REPORT データ格納バッファ領域のポインタ
length	SET_REPORT データ長
complete	コールバック関数

戻り値

USB_OK	正常終了
USB_ERROR	送信失敗

解説

HID デバイスに対して SET_REPORT を送信します。

補足

―

使用例

```
void hid_class_request(uint16_t devadr)
{
  USB_ER_t    err;

  hid_dev_info[devadr].set_report = NUM_LOCK;
  err = R_usb_hhid_set_report( devadr, &hid_dev_info[devadr].set_report,
                    SET_REPORT_LENGTH, &hid_class_request_complete );
}
```

3.4.9 R_usb_hhid_set_protocol

SET_PROTOCOL 送信

形式

USB_ER_t R_usb_hhid_set_protocol(uint16_t devadr, uint8_t protocol, USB_UTR_CB_t complete)

引数

devadr	デバイスアドレス
protocol	SET_PROTOCOL データ(0:Boot, 1:Report)
complete	コールバック関数

戻り値

USB_OK	正常終了
USB_ERROR	送信失敗

解説

HID デバイスに対して SET_PROTOCOL を送信します。

補足

―

使用例

```
void hid_class_request(uint16_t devadr)
{
  USB_ER_t    err;

  err = R_usb_hhid_set_protocol(devadr, BOOT_PROTOCOL, &hid_class_request_complete);
}
```

3.4.10 R_usb_hhid_transfer_start

USB データ転送要求

形式

USB_ER_t	R_usb_hhid_transfer_start(uint8_t *table, uint32_t size, USB_UTR_CB_t complete, uint16_t pipe)

引数

*table	データ格納バッファ領域へのポインタ
size	読み出しデータサイズ
complete	コールバック関数
pipe	パイプ番号

戻り値

USB_OK	正常終了
USB_ERROR	異常終了

解説

USB デバイスに対し、指定したパイプでデータ転送要求を行います。
転送完了すると、第 3 引数で指定したコールバック関数がコールされます。
このコールバック関数の引数には、転送結果が格納されています。

補足

―

使用例

```
uint8_t  buf[8];

USB_ER_t usb_smp_task(void)
{
  USB_ER_t err;

  err = R_usb_hhid_transfer_start(&buf, 8, usb_data_received, IN_PIPE);

  return err;
}

/* データ受信完了通知コールバック関数 */
void usb_data_received(USB_UTR_t *utr)
{
}
```

4. サンプルアプリケーション

本章では、HHID 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 動作環境

HHID の動作環境例を Figure 4-1 に示します。

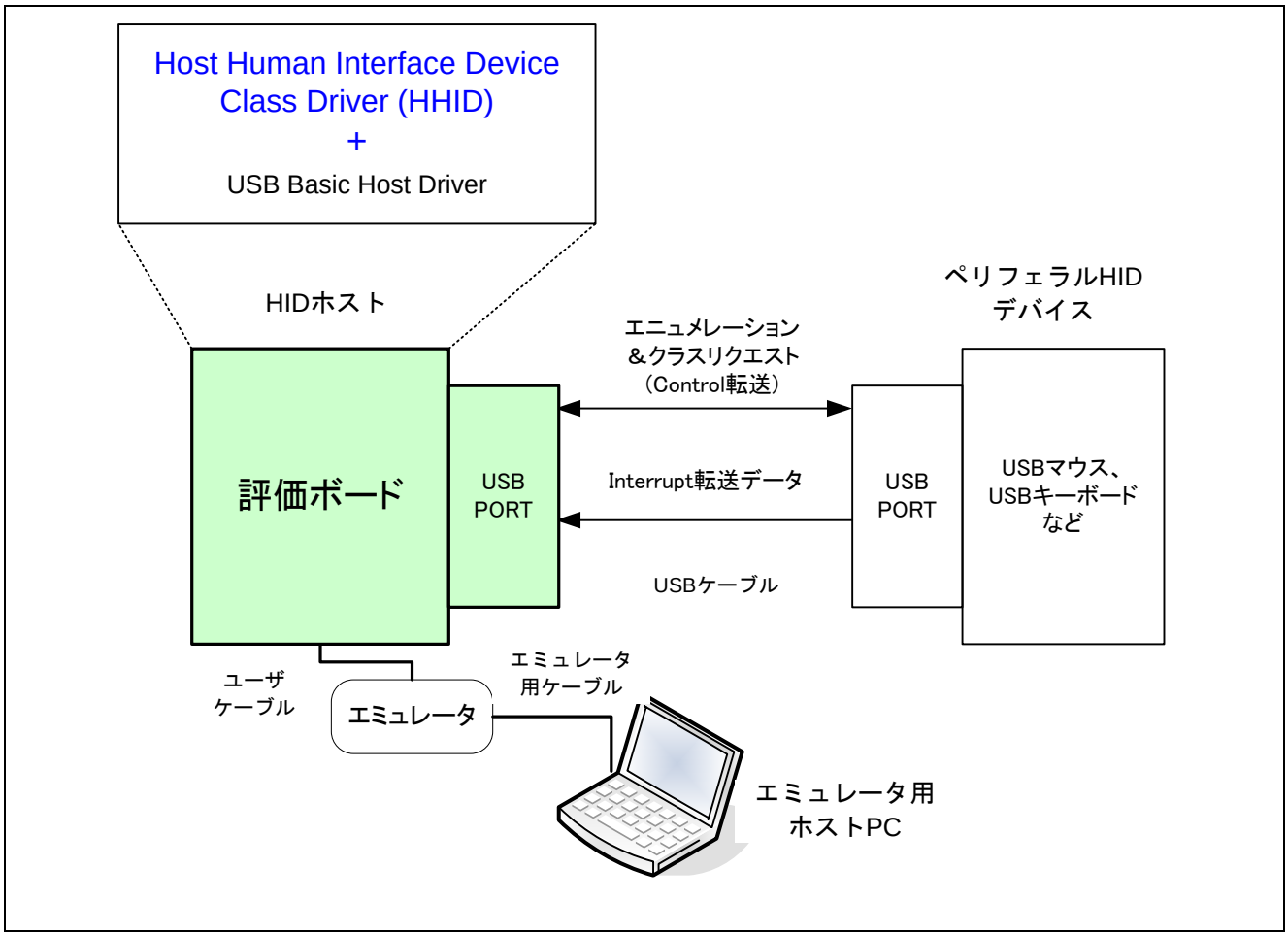


Figure 4-1 動作環境例

4.2 仕様概要

HHID 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) HID デバイス(マウス/キーボード)とのデータ転送

接続された HID デバイスとのデータ転送を行います。USB Hub を使用すれば、複数の HID デバイスを接続することができ、データ転送を同時に行います。

a) マウス接続時、動作に対応した LED を点灯します。

左クリック	: LED0
右クリック	: LED1
中クリック	: LED2
移動	: LED3

b) キーボード接続時、入力されたキーデータに対応した LED を点灯します。

アルファベット	: LED1
数字	: LED2
その他	: LED0

2) HID デバイスのサスペンド/レジューム実行（USB_HHID_SUSPEND_ENABLE が定義されている場合）

無操作状態で一定時間が経過すると、接続された HID デバイスに対してサスペンドします。
サスペンド状態で一定時間が経過すると、接続された HID デバイスに対してレジュームします。

※ USB ポートに直接 HID デバイスを接続した場合のみ行います。HUB を介して接続された HID デバイスに対するサスペンド/レジュームはサポートしていません。
※ 時間はソフトウェアで計測しています。間隔は IDLE_LOOP_COUNT の定義値で設定してください。

4.3 初期設定

初期設定例を以下に示します。

```
void usb_hhid_apl(void)
{
    /* MCU の端子設定（「4.3.1」参照*/
    usb_mcu_setting();

    /* USB ドライバの設定（「4.3.2」参照） */
    R_usb_hstd_MgrOpen();
    R_usb_cstd_SetTaskPri(USB_HUB_TSK,  USB_PRI_3);      // (注)
    R_usb_hhub_Registration(USB_NULL);                   // (注)
    hid_registration();
    R_usb_hhid_driver_start();

    /* メインルーチン */
    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。

4.3.1 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。

4.3.2 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. HHID クラスドライバの API 関数（R_usb_hhid_driver_start()）を呼び出し、HHID 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 hid_registration(void)
{
   USB_HCDREG_t  driver;        // クラスドライバ登録用構造体

   /* 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_HID;
   /* 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_hhid_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     = &hid_confugured;
   /* USB デバイス切断時に呼ばれる関数を設定 */
   driver.devdetach     = &hid_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    = &hid_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     = &hid_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。

4.4 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_hhid_task();          // HID ドライバタスク
    }
    hid_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。

4.4.1 APL

APL 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 4-1 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(4.4.2 参照)のメンバに格納されています。
2. 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。なお、このイベントは、APL が管理する構造体(4.4.2 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。

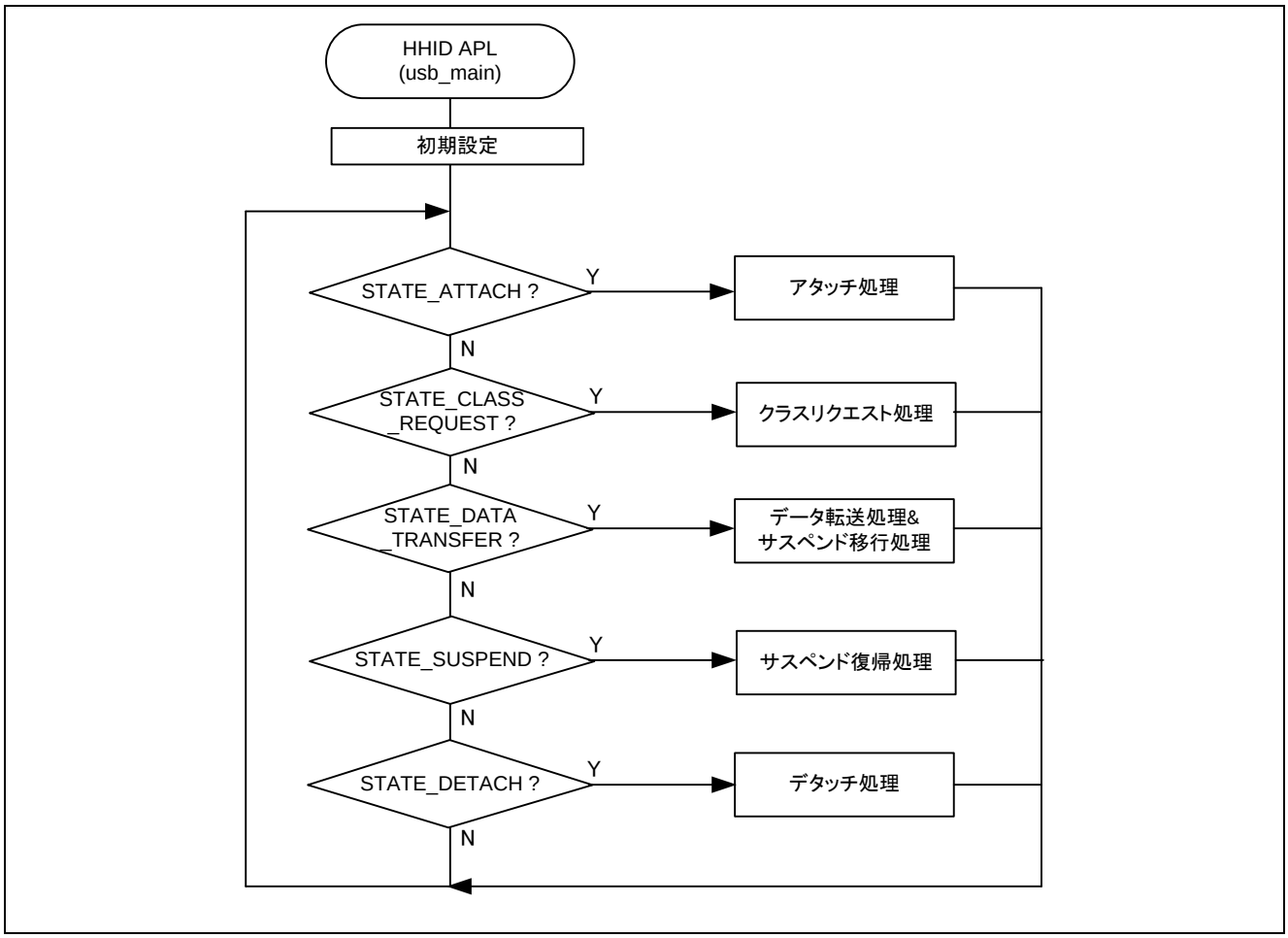


Figure 4-2 メインループ処理

4.4.2 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、接続する HID デバイスのデバイスアドレスをもとに管理されます。

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”が返されます。

```
/* Structure for HID device control */
typedef struct DEV_INFO
{
    uint16_t    state;               /* State for application */
    uint16_t    event_cnt;           /* Event count */
    uint16_t    event[EVENT_MAX];    /* Event no. */
    uint16_t    pipe;                /* Use pipe no. */
    uint16_t    protocol;            /* HID Protocol(Mouse/Keybord/none) */
    uint8_t     cr_seq;              /* Class Request Sequence */
    uint8_t     set_report;          /* SetReprot (Send)DATA */
    uint8_t     data[REPORT_LENGTH]; /* Receive Data */
}
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_USB_TRANSFER_COMPLETE
STATE_DATA_TRANSFER	データ転送& サスペンド移行処理	EVENT_USB_TRANSFER_START
		EVENT_USB_TRANSFER_COMPLETE
		EVENT_SWITCH_INPUT
STATE_SUSPENDED	サスペンド復帰処理	EVENT_SWITCH_INPUT
STATE_DETACH	デタッチ処理	'--

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_TRANSFER_START	データ転送要求
EVENT_USB_TRANSFER_COMPLETE	データ転送完了
EVENT_SWITCH_INPUT	スイッチ入力
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、HID 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、イベントに EVENT_NONE をセットします。

② HID デバイスが接続されるまで STATE_ATTACH 状態が継続され、hid_connect_wait()がコールされます。

③ HID 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 hid_configured()が USB ドライバよりコールされ、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。

④ イベント EVENT_CONFIGURD では、hid_connect_wait()関数内で以下の処理が行われ、次のループ処理でクラスリクエスト処理を開始します。

a) 接続された HID デバイスの情報を変数(hid_dev_info)に設定

b) 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。

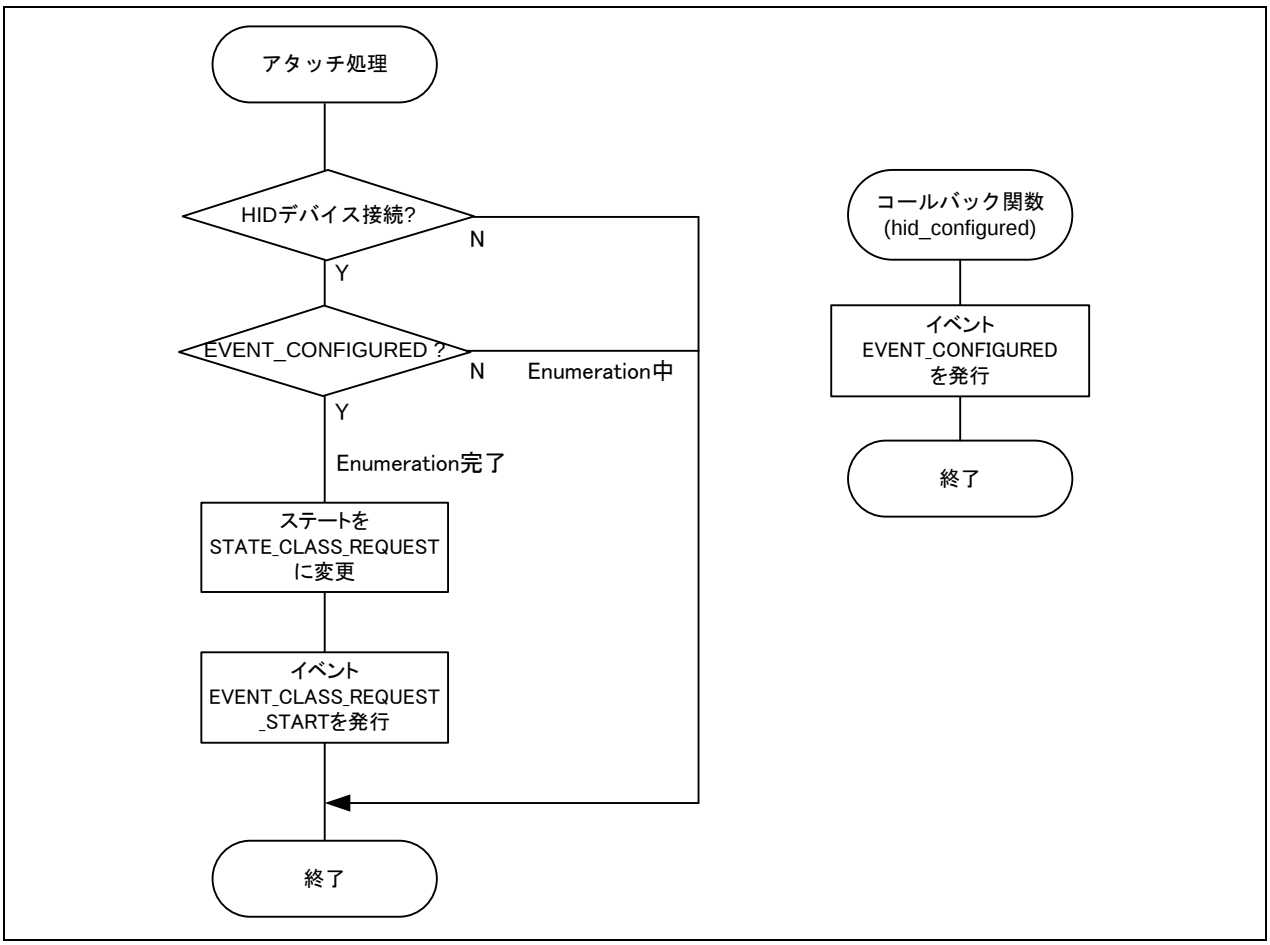


Figure 4-3 アタッチ処理概略フロー

2. クラスリクエスト処理 (STATE_CLASS_REQUEST)

== 概要 ==

このステートでは、HID 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。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、はじめに hid_class_request()で 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 hid_class_request_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_CLASS_REQUEST_COMPLETE を発行します。

③ EVENT_CLASS_REQUEST_COMPLETE では、上記①で送信したクラスリクエストによって以下の通りに処理が分かれます。

a) マウスに対しクラスリクエスト SetProtocol を送信した場合、ステートを STATE_DATA_TRANSFER に変更し、イベント EVENT_USB_TRANSFER_START を発行します。次のループ処理でデータ転送処理を開始します。

b) キーボードに対しクラスリクエスト SetProtocol を送信した場合、イベント EVENT_CLASS_REQUEST_START を発行し、クラスリクエスト SetReport 送信処理を開始します。

c) クラスリクエスト SetReport を送信した場合、ステートを STATE_DATA_TRANSFER に変更し、イベント EVENT_USB_TRANSFER_START を発行します。

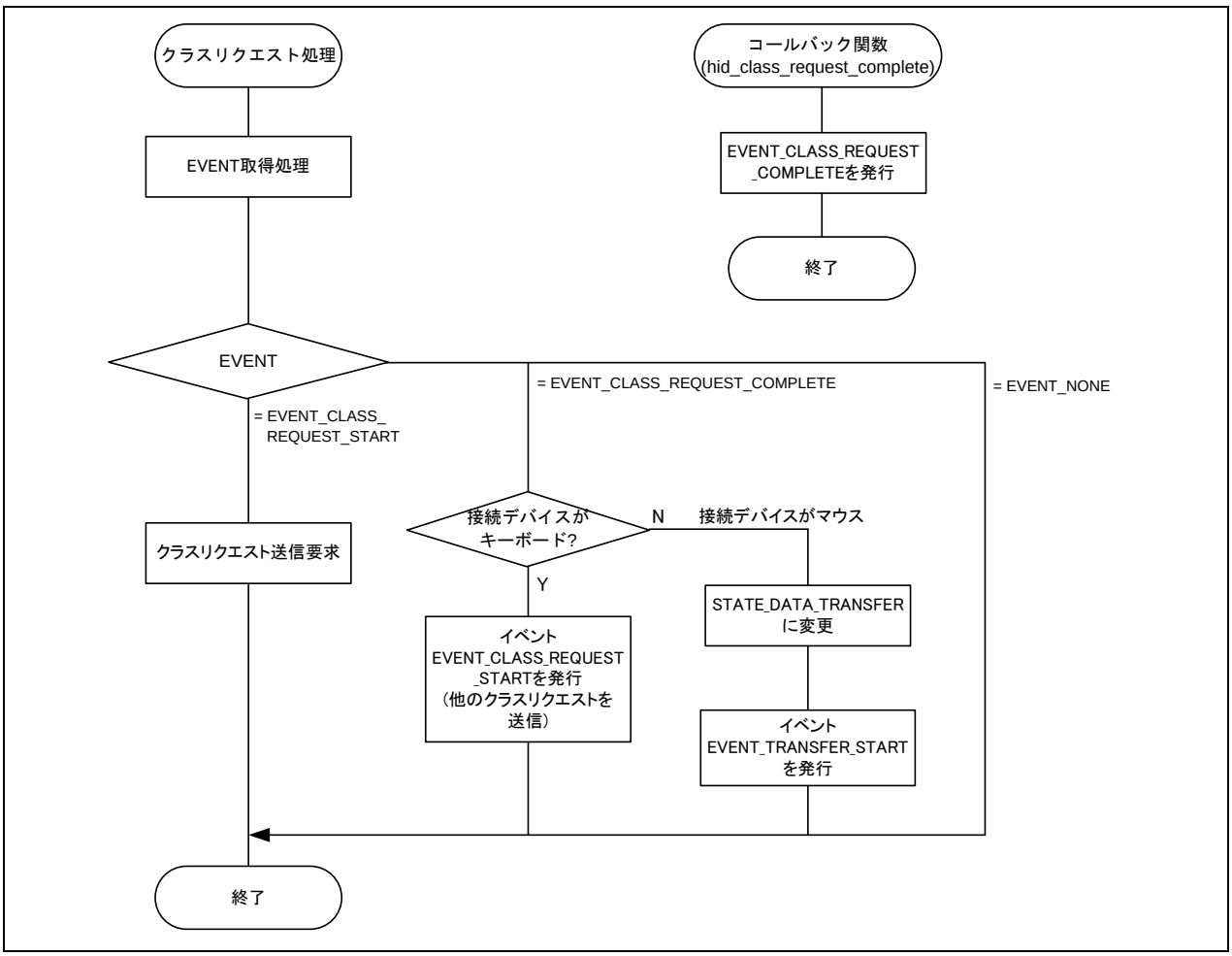


Figure 4-4 クラスリクエスト処理概略フロー

3. データ転送処理&サスペンド移行処理 (STATE_DATA_TRANSFER)

== 概要 ==

このステートでは、HID デバイスからのデータ受信処理およびそのデータ処理を行います。その他、HID デバイスに対するサスペンド信号送信処理も行います。

== 内容 ==

① このステートでは、hid_data_transfer()内の EVENT_USB_TRANSFER_START が処理され、USB ドライバに対しデータ転送処理要求を行います。

② データ転送処理が完了するとコールバック関数 hid_receive_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_TRANSFER_COMPLETE を発行します。

③ EVENT_USB_TRANSFER_COMPLETE では、hid_data_transfer()は受信データに応じた処理をして、イベント EVENT_USB_TRANSFER_START を発行します。

④ 上記①から③までのデータ転送処理中にサスペンドするとステートを STATE_SUSPENDED に変更し、R_usb_hhid_ChangeDeviceState 関数を使って接続された HID デバイスに対し SUSPEND 信号を送信します。

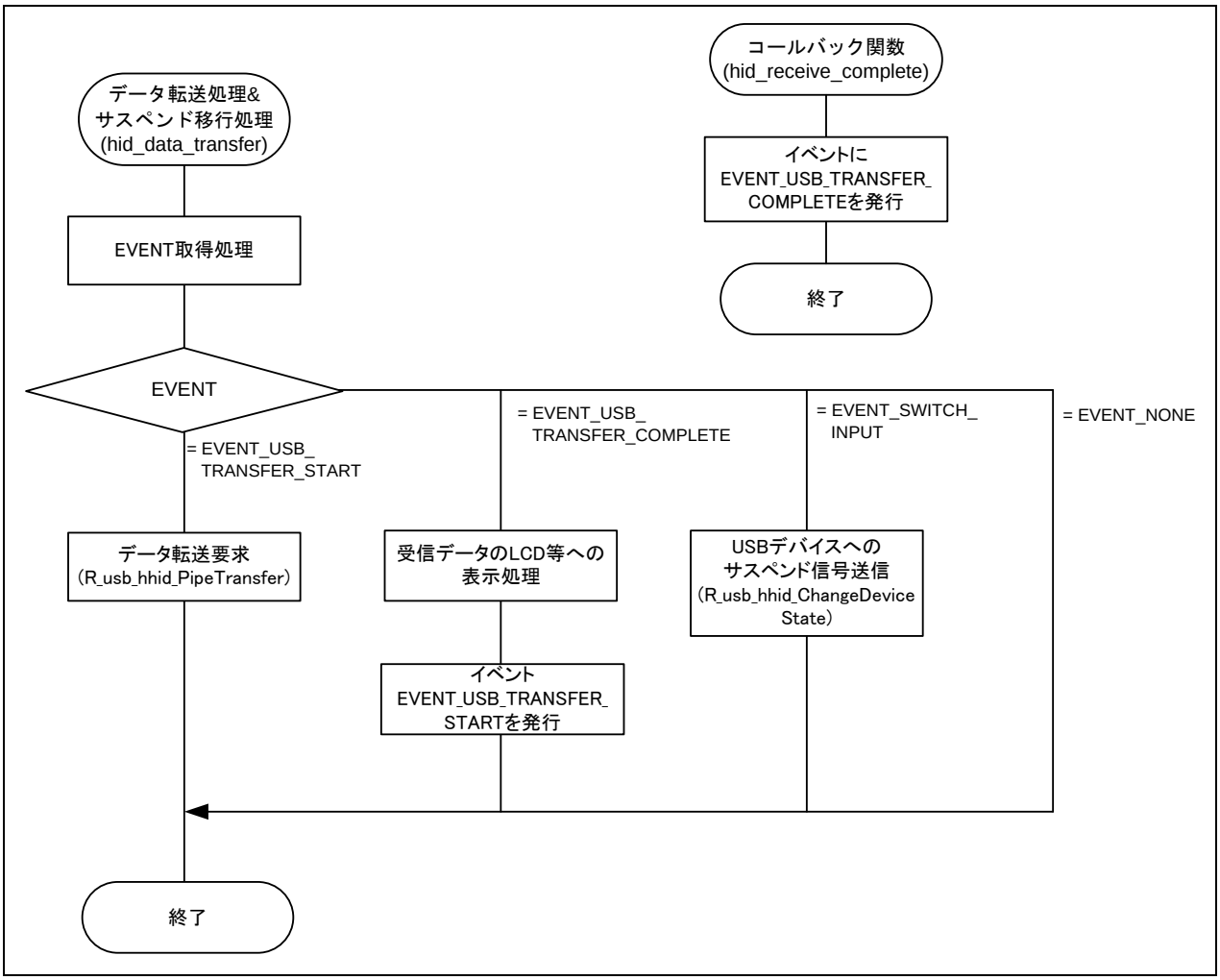


Figure 4-5 データ転送処理 & サスペンド移行処理概略フロー

4. サスペンド復帰処理 (STATE_SUSPENDED)

== 概略 ==

このステートでは、サスペンド状態の HID デバイスに対し RESUME 信号を送信し、サスペンド状態からの復帰処理を行います。RESUME 信号の送信完了は、コールバック関数 hid_resume()がコールされることによって通知されます。

== 内容 ==

① 一定時間経過すると、hid_suspended()関数は R_usb_hhid_ChangeDeviceState()関数を使って、接続された HID デバイスに対し RESUME 信号を送信します。

② RESUME 信号の送信が完了すると USB ドライバよりコールバック関数 hid_resume()がコールされます。このコールバック関数は、ステートを STATE_DATA_TRANSFER に変更し、イベント EVENT_USB_TRANSFER_START を発行します。

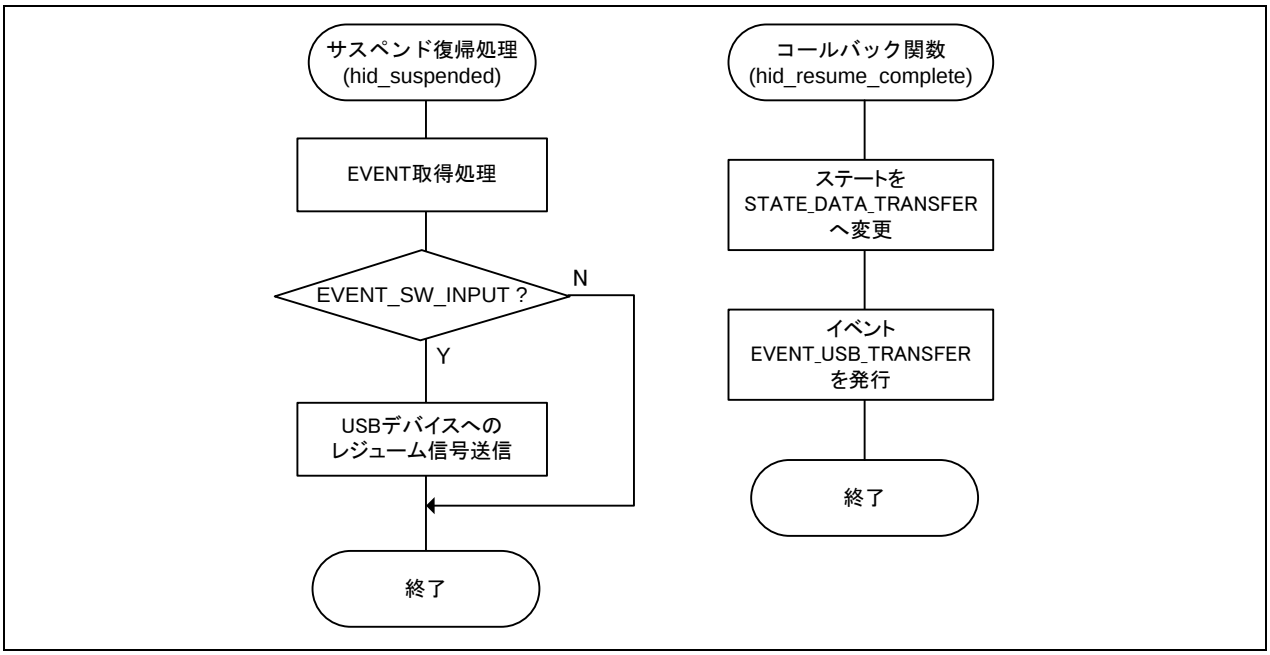


Figure 4-6 サスペンド復帰処理概略フロー

5. デタッチ処理 (STATE_DETACH)

接続された HID 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 hid_detach()がコールされます。このコールバック関数は、ステートを STATE_DETACH に変更します。STATE_DETACH では、変数のクリア処理、ステートを STATE_ATTACH に変更するなどの処理を行います。

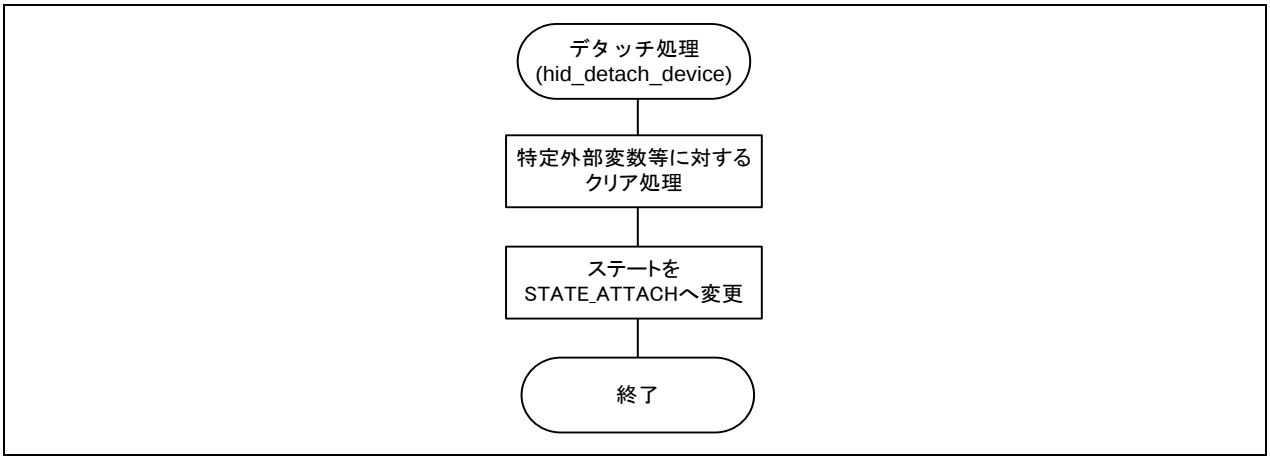


Figure 4-7 デタッチ処理概略フロー

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

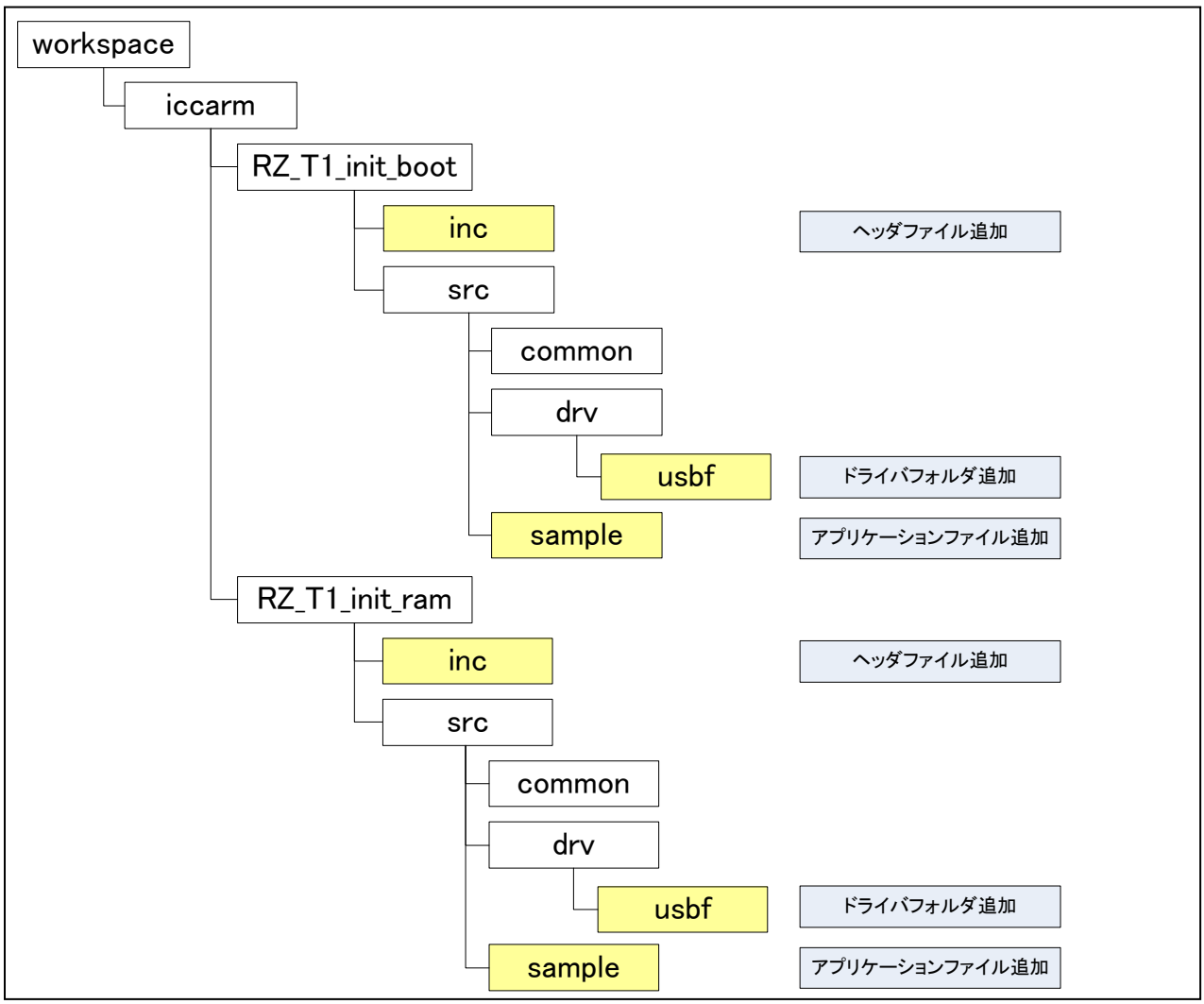
「RZ/T1 グループ初期設定 Rev.1.30」では、開発環境とブート方法によってフォルダ構成が異なります。
全ての開発環境とブート方法の各フォルダに対して、下記の変更をしています。

・”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_hhid_config.h
　r_usb_hhid_if.h

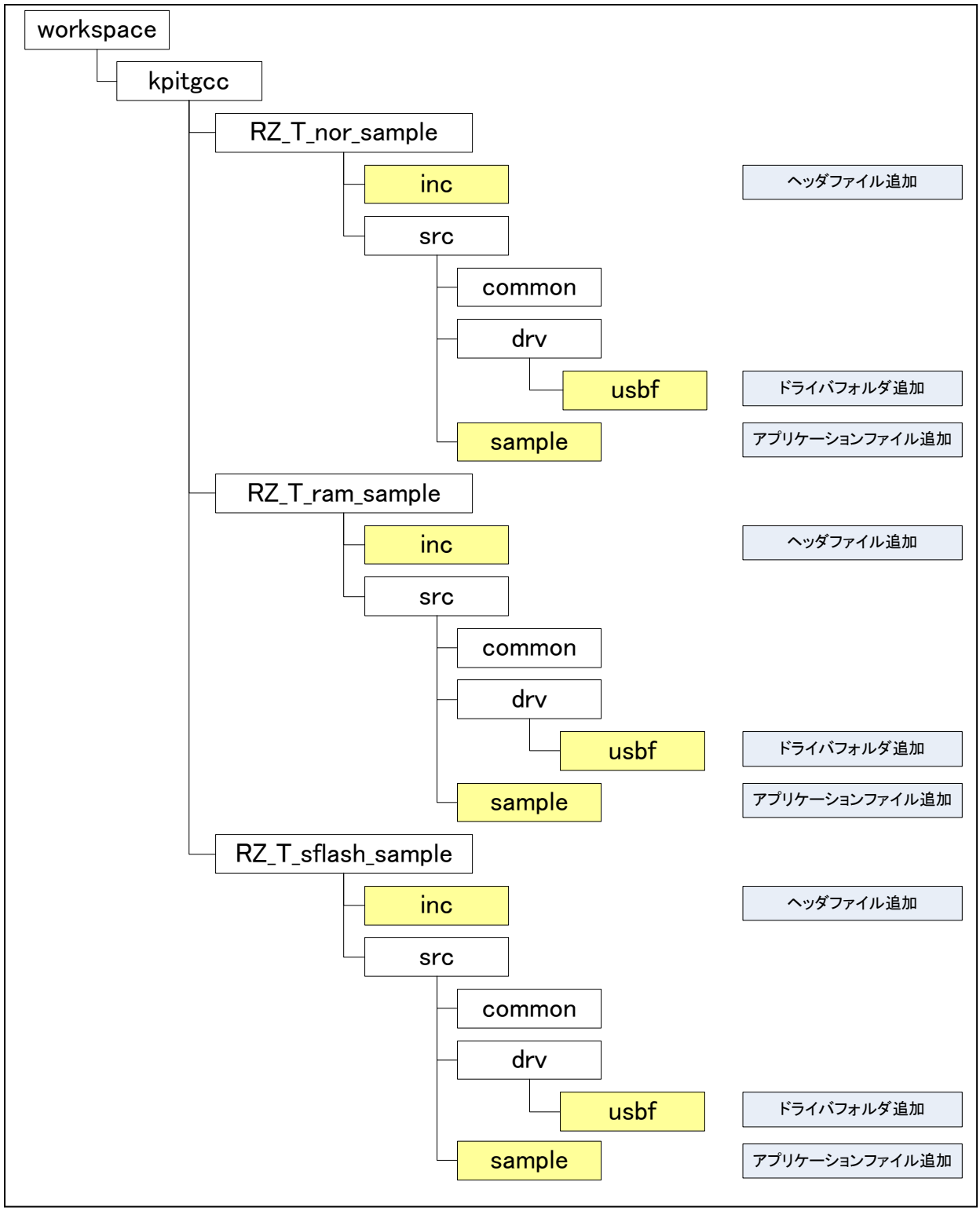
・”sample”フォルダに下記のファイルを追加
　r_usb_main.c
　r_usb_hhid_apl.c
　r_usb_hhid_apl.h

・”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

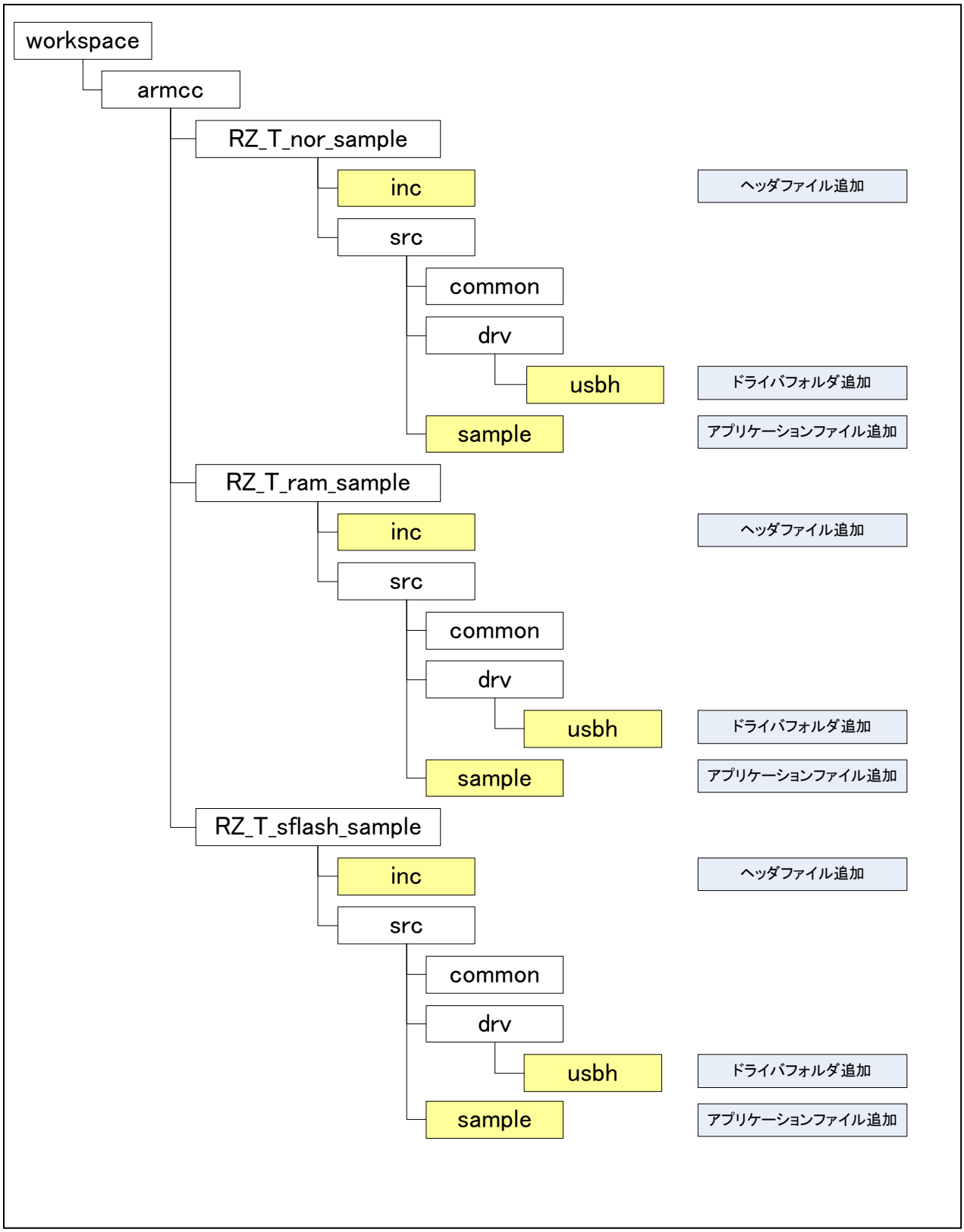
以下に EWARM のフォルダ構成を示します。



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

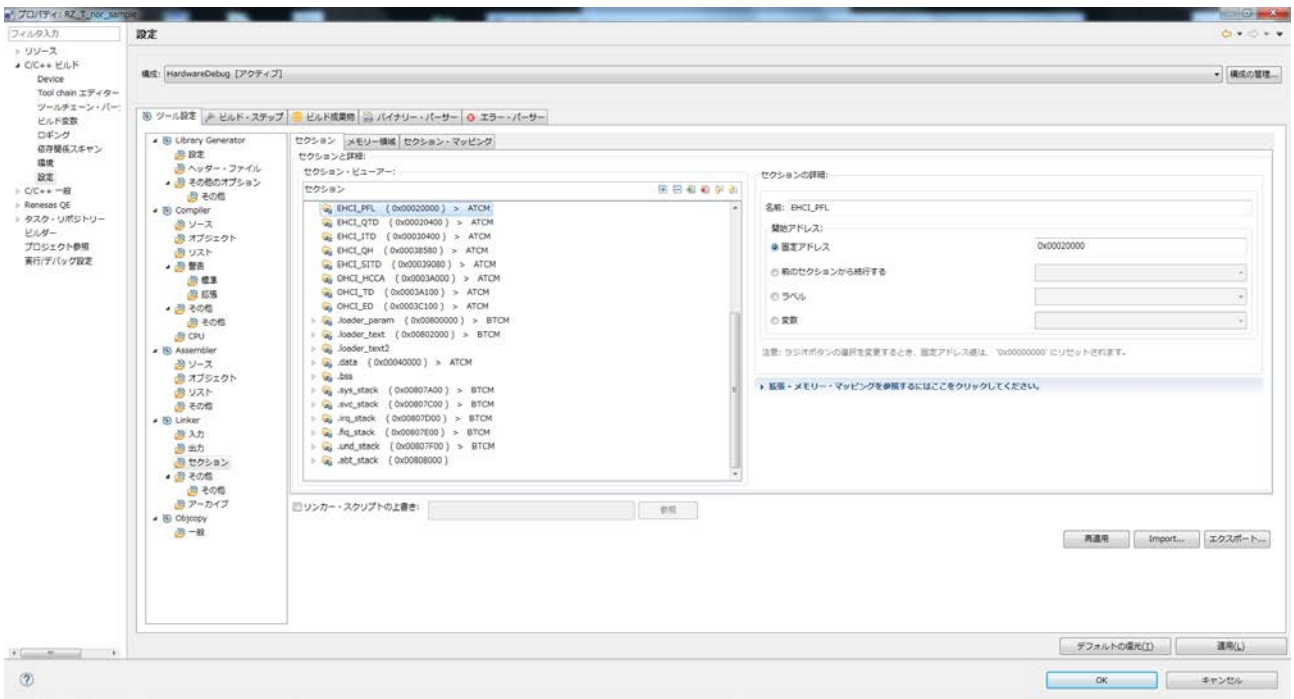
コード領域とデータ領域のセクションサイズを変更して、以下のセクションを追加しています。

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 ];    /* 4KByte alignment */
#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
		2	制限事項に Interrupt OUT 転送を行う HID デバイスについて追加
		6,12	R_usb_hhid_GetMaxPacketSize の説明を追加

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/